Altova MapForce 2023 Enterprise Edition

User & Reference Manual

Altova MapForce 2023 Enterprise Edition
User & Reference Manual

Published: 2022

Table of Contents

18 ActiveX Integration 1827

1 Introduction

MapForce® 2023 Enterprise Edition is a visual data mapping tool for advanced data integration projects. MapForce® is a 32/64-bit Windows application that runs on Windows 7 SP1 with Platform Update, Windows 8, Windows 10, Windows 11, and Windows Server 2008 R2 SP1 with Platform Update or newer. 64-bit support is available for the Enterprise and Professional editions. MapForce also integrates with Visual Studio and Eclipse as well as Microsoft Office products. For more information, see Support Notes 1889.

Last updated: 7 October 2022

1.1 New Features

This section describes new features of each MapForce release. For more details, see the respective subsection.

1.1.1 Version 2023

Version 2023

- Support for the following themes has been added: *Classic*, *Light*, and *Dark*. For more information, see Window[1486].
- Internal updates and optimizations.
- Eclipse support has been updated and now covers the following versions: 2022-09, 2022-06, 2022-03, 2021-12 (*Professional and Enterprise editions*). For more details, see MapForce Plug-in for Eclipse[1429].
- Support for ODETTE EDI messages (*Enterprise Edition*). For details, see EDI[444].
- Support for the XII Transformation Registry 5 Specification (*Enterprise Edition*). For more information, see XBRL[744].
- It is now possible to create database-based UDF parameters[934] and variables[848] with a tree of related tables (*Professional and Enterprise editions*).
- It is now possible to send an `application/x-www-form-urlencoded` request structure to a REST service (*Enterprise Edition*). For details, see REST Web Service Parameters[593].
- Support for UN/EDIFACT D.21B and D.22A Directories (*Enterprise Edition*). For more information, see EDI[444].
- Support for SQLite 3.39.2, MariaDB 10.9.2, and PostgreSQL 14.5 (*Professional and Enterprise editions*). To find out more about all supported databases, see Databases[161].
- Support for XML Schema Manager[145] that provides a centralized way to install and manage XML schemas for use across all Altova's XBRL-enabled applications.
- Support for mappable EDI delimiters (*Enterprise Edition*). The feature is currently supported for the following EDI standards: EDIFACT, X12, and NCPDP SCRIPT. For details, see EDI Component Settings[453].

1.1.2 Version 2022

Version 2022 Release 2

- Internal updates and optimizations
- Eclipse support has been updated and now covers the following versions: 2021-12, 2021-09; 2021-06; 2021-03 (*Professional and Enterprise editions*). For details, see MapForce Plug-in for Eclipse[1429].
- Support for Visual Studio 2022 in the MapForce Plug-in for Visual Studio and code generation (*Professional and Enterprise editions*). For more information, see MapForce Plug-in for Visual Studio[1426] and code generation[1492].
- Support for .NET 6.0 in code generation (*Professional and Enterprise editions*). For details, see code generation[1492].
- New database versions are supported: PostgreSQL 14, SQLite 3.37.2, MariaDB 10.6.5, MySQL 8.0.28, IBM DB2 11.5.7 (*Professional and Enterprise editions*). To find out more about all supported databases, see Databases[161].

- It is now possible to preview images in the **Project** window (*Professional and Enterprise editions*). For more information, see Project Basics 118.
- It is now possible to create EBA-conformant filing indicators for target XBRL components (*Enterprise Edition*). For details, see Filing Indicators 748.

Version 2022

- Internal updates and optimizations
- Eclipse support has been updated and now covers the following versions: 2021-09; 2021-06; 2021-03; 2020-12 (*Professional and Enterprise editions*). For details, see MapForce Plug-in for Eclipse 1429.
- Copy-all connections 97 now support JSON. This feature is available only for compatible JSON types (*Enterprise Edition*).
- A new StyleVision output pane called *Text* has been introduced. If an SPS file is attached to a component, the new plain text output format can be previewed in MapForce (*Professional and Enterprise editions*). For more information, see StyleVision Output Panes 1336.
- Support for JSON Schema in variables 847 and UDF parameters 933 (*Enterprise Edition*).
- Support for NoSQL databases: MongoDB and CouchDB (*Enterprise Edition*). To find out more about all supported databases, see Databases 161.
- A new `bson` function library has now become available, which allows you to create and manipulate some of the BSON types (*Enterprise Edition*). For details, see bson 1072.
- Support for UN/EDIFACT D.20B and D.21A Directories. For more information, see EDI 444 (*Enterprise Edition*).
- Support for SWIFT 2021. For more information, see EDI 444 (*Enterprise Edition*).

1.1.3 Version 2021

Version 2021 Release 3

- Support for new JSON Schema Draft 2019-09 and Draft 2020-12 (*Enterprise Edition only*).

Version 2021 Release 2

- XSLT 3.0 is now supported as mapping language. See Generating XSLT Code 107. MapForce now includes new built-in functions that are supported when the mapping language is XSLT 3.0. For more information, see Function Library Reference 979.
- When generating C# code, you can select .NET Core 3.1 and .NET 5.0 as target frameworks from code generation options 1625 (this adds to existing support for .NET Framework projects). For details, see Generating C# code 1501.
- Internal updates and optimizations.

Version 2021

- All XBRL-enabled applications from Altova, including MapForce, benefit from simplified XBRL taxonomy management and release-independent taxonomy updates. A new XBRL Taxonomy Manager 751 is now available for that purpose, where you can view, manage, and update the currently installed XBRL taxonomies.

- When calling a REST Web service, you can supply the URL of the Web service as parameter to the mapping. This option provides extra flexibility when you need complete control over the URL and would like to specify it at mapping runtime, see Adding a Web Service Call (REST-Style) 587.
- Support for mapping data to or from SWIFT MT 485 format.
- You can now store custom EDI Collections in a directory of your choice, or in your user's profile directory, see Creating a Custom EDI Collection 519.
- When a mapping writes data to a target EDI component, you can switch the component to generate XML instead of EDI output, thus providing an alternative way to preview the generated data. You can also quickly convert EDI files to XML format, with a click of a button—this does not require any XML schema or drawing mapping connections. See Converting EDI to XML 458 for more information.
- A MapForce mapping can read BLOB (binary large object) data from binary files and write binary files to the disk. This makes it possible, for example, to read BLOB fields from a database and save them as image files on the disk, or to read binary files such as PDFs from the disk and save them as `xs:base64Binary` fields within an XML file. See Binary Files 827 for more information.
- New database versions are supported: MariaDB 10.4, 10.5
- New Eclipse versions are supported: 2019.09, 2019.12, 2020.03, 2020.06
- When joining multiple database tables or views using SQL join components in a mapping, you can set the join mode either as LEFT OUTER JOIN or INNER JOIN, see Changing the Join Mode 335.
- Internal updates and optimizations.

1.1.4 Version 2020

Version 2020 Release 2

- A new Manage Libraries window 35 is available that enables you to view and manage all function libraries imported at document and at program level (this includes MapForce user-defined functions and other kinds of libraries). This makes it possible, for example, to easily copy-paste user-defined functions from one mapping to another, see Copy-Pasting UDFs Between Mappings 932.
- When a mapping file imports libraries, the path of imported library files is relative to the mapping file by default, see Relative Library Paths 911. You can still import mappings at application level, like in previous releases, but in this case the library path is always absolute.
- If a mapping file imports XSLT or XQuery libraries, you can generate XSLT or XQuery code that references the imported library files using a relative path. The new option is available in the Mapping Settings 114 dialog box.
- The MapForce API 1645 has been enhanced with new members that enable you to manage imported libraries programmatically (for example, add or remove them).
- Code generated for XML schema wrapper libraries now provides more control over element namespaces and prefixes. New methods are available to declare or override namespaces for an element, or to append an element with a prefixed namespace. See Example: Purchase Order 1571.
- New database versions are supported: PostgreSQL 12.1 and Informix 14.10.
- In addition to generating Excel files from a mapping, MapForce now supports updating existing Excel files (when the transformation language is BUILT-IN). See Example: Updating Existing Excel Sheets 740.
- Support for UN/EDIFACT D.19A and D19B Directories, see EDI 444.
- When creating OAuth 2.0 credentials, you can configure the authorization details to be in the POST request body. This is an optional method in addition to the already supported standard method of supplying authorization details in the POST request header, see OAuth 2.0 Credentials 1301.
- Internal updates and optimizations

Version 2020

- It is now possible for mappings to call Web services that require OAuth 2.0 authorization[1301].
- HTTP or OAuth authentication details can be transferred to other execution environments with the help of credential objects, see Credentials[1297].
- Support for mapping data to or from JSON files using the JSON Lines[799] text format
- Support for Visual Studio 2019 in the MapForce Plug-in for Visual Studio[1426] and code generation[1492].
- Support for Eclipse 4.9 - 4.12, see MapForce Plug-in for Eclipse[1429].
- If an Oracle package contains public stored procedures or functions, those are also available to the mapping, see Adding Stored Procedures to the Mapping[392].
- You can configure a database component so that database object names are treated as relative to the default schema, not bound to a particular schema. This helps save time if you need to switch to a different database in future, see Switching Databases and Schemas[255].
- You can deploy Global Resources created in MapForce to FlowForce Server, see Deploying Global Resources to FlowForce Server[1416].
- When replacing values with the help of a look-up table, you can paste tabular data (key-value pairs) from external sources such as CSV or Excel into the mapping. Also, it is easier to handle cases when a value is not found in the predefined look-up table—processing such values no longer requires the use of `substitute-missing` function. See Using Value-Maps[884].
- Internal updates and optimizations

1.1.5 Version 2019

Version 2019 Release 3

- Major parts of the graphical user interface are now optimized for monitors with high pixel density (HiDPI)
- Support for Database Tracing and Error Logging[427]
- Better support for database transaction handling in generated program code and server execution. If an error occurs when a mapping updates a database, it is possible to continue the mapping execution even if some operations failed, see Handling Database Exceptions[415].
- Web service enhancements, including the following:
  - Ability to map the HTTP status code returned by the Web service (applicable for RESTful Web services)
  - Ability to supply the username, password, and URL of the Web service from the mapping (applies both to WSDL-style and RESTful Web services), see Web Service Call Settings[606].
  - Ability to create the raw body (MIME entity) of the Web service from the mapping, or map the raw body returned by the Web service call, with the help of new mime[1131] functions. Applicable for RESTful Web services.
  - Ability to handle the Web service response conditionally, depending on the HTTP status code and content type (for RESTful Web services), see Handling the HTTP Response Conditionally[603].
- A mapping can encode or decode Base64 data with the help of the new charset-encode[1117] and charset-decode[1116] functions.
- Support for UN/EDIFACT D.18B Directory, see EDI[444].
- New database versions are supported: PostgreSQL 11, Db2 for i 7.2 and 7.3
- Support for Eclipse 4.7 - 4.10, see MapForce Plug-in for Eclipse[1429]

- Support for explicitly setting the Java Virtual Machine path from MapForce, see Java Settings 1476
- Internal updates and optimizations

Version 2019

- MapForce (and MapForce Server) can now map data to or from binary files in Protocol Buffers format, see Protocol Buffers 811.
- New database versions are supported: Firebird 3.0, Informix 12.10, MariaDB 10.2, MySQL 8.0, PostgreSQL 10, SQL Server 2017, SQL Server for Linux (adds to support for previous versions)
- You can selectively apply a default value or node function to specific nodes that meet your custom-defined criteria. For example, you can apply a node function only to nodes of specific type whose name matches a regular expression, see Applying Node Functions and Defaults Conditionally 921.
- When creating node functions, you can process data conditionally based on metadata such as node name or node annotation, see Supplying Node Metadata to Node Functions 923.
- Mappings that contain grouping functions run much faster in generated C#, C++, or Java code
- Internal updates and optimizations

1.2 What Is MapForce?

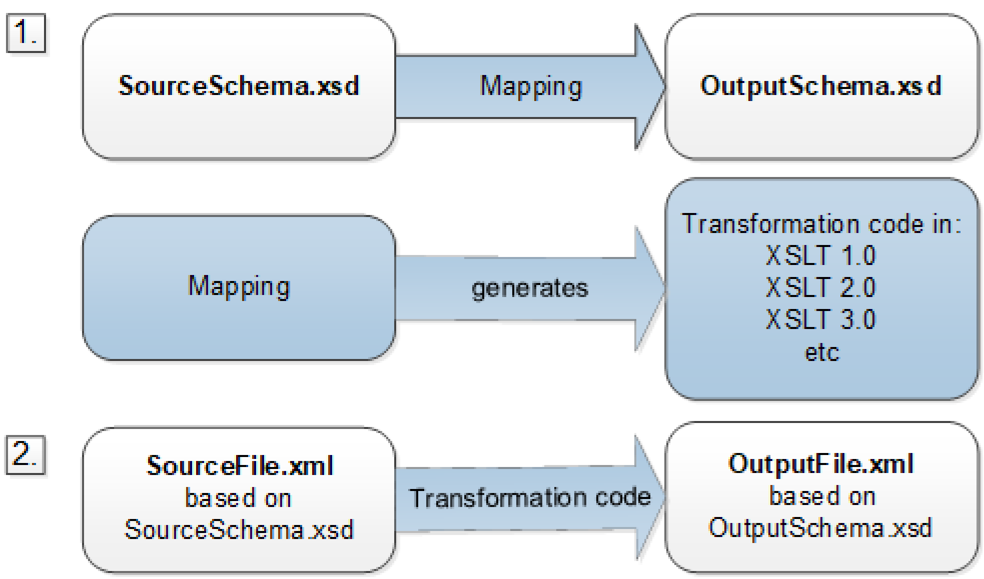
Altova website: Data mapping tool

MapForce is a powerful and flexible tool that allows any-to-any graphical mapping of different data formats. See Mapping: Sources and Targets [26] for a complete list of available data formats. MapForce enables you to map one source to one target, one source to multiple targets, multiple sources to one target, or multiple sources to multiple targets. To find out more about mapping scenarios, see Mapping Scenarios [28] and Tutorials [42]. MapForce also provides an extensive range of data processing and filtering options, such as functions [904], variables [847], filters and conditions [878], that allow you to manipulate your data.

In order to be able to carry out a mapping, you must provide a data structure that describes the structure of each of your source and target files. For example, an XML schema defines the structure of an XML document. The mapping (from source to target) is achieved by means of a drag-and-drop graphical user interface. You do not have to write any program code for the mapping. The code is generated for you by MapForce. You can then use this code to transform documents having the source data structure to documents having the target data structure.

Abstract model

The abstract model below illustrates one of the basic scenarios of data transformation in MapForce. The first step shows that one abstract structure called `SourceSchema.xsd` is mapped to another abstract structure called `OutputSchema.xsd`. The mapping generates transformation code in the selected transformation language. The second step shows that the content of the source XML file, which is based on `SourceSchema.xsd`, is mapped to the target XML file, which is based on `OutputSchema.xsd`. The mapping of the content from the source to the target file is carried out by means of the transformation code generated in the previous step.



All editions of MapForce are available as 32-bit applications. MapForce Professional and Enterprise editions are additionally available as 64-bit applications.

Conventions

Mapping files illustrated or referenced in the manual can be found in the following locations:

- C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples
- C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\Tutorial
- C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\Tutorial\BasicTutorials

In this section

This section is organized into the following topics:

1.2.1 Mapping: Sources and Targets

In MapForce, *source* and *target* are essential terms that refer to data structures from which or to which data is mapped, respectively. Technologies that can be used as mapping sources and targets are listed below.

MapForce Basic Edition

- XML and XML Schema

MapForce Professional Edition

- XML and XML Schema
- Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format
- Databases: all major relational databases, including Microsoft Access and SQLite databases
- Binary files (raw BLOB content)

MapForce Enterprise Edition

- XML and XML Schema
- Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format
- Data from legacy text files can be mapped and converted to other formats with MapForce FlexText
- SQL Databases: all major relational databases, including Microsoft Access and SQLite databases
- NoSQL Databases
- Binary files (raw BLOB content)
- EDI family of formats, including UN/EDIFACT, ANSI X12, HL7, IATA PADIS, SAP IDoc, TRADACOMS
- JSON files
- Microsoft Excel 2007 and later files
- XBRL instance files and taxonomies
- Protocol Buffers

1.2.2 Transformation Languages

In MapForce, a transformation language is used to generate transformation code that carries out mappings. You can select/modify a transformation language at any time. MapForce allows viewing the transformation code in the selected language. For more information, see Code Generation [107]. You can also generate this code via the menu command **File | Generate Code in** and use this code for transforming any data document that is valid according to the source component's schema. Depending on the MapForce edition, you can choose the following languages for your data transformation:

MapForce Basic Edition	**MapForce Professional and Enterprise editions**
• XSLT 1.0 • XSLT 2.0 • XSLT 3.0	• XSLT 1.0 • XSLT 2.0 • XSLT 3.0 • BUILT-IN • XQuery • Java • C# • C++

To select a transformation language, do one of the following:

- In the **Output** menu, click the name of the language you wish to use for transformation.
- Click the name of the language in the **Language Selection** toolbar (*shown below*).

When you change the transformation language of the mapping, certain MapForce features may not be supported for that language. For more information, see Support Notes [1889].

As you design or preview mappings, MapForce validates the integrity of your schemas or transformations. If any validation errors occur, MapForce displays them in the Messages window [36]. This is helpful because you can immediately review and correct these errors.

Transformation languages in MapForce Professional and Enterprise editions
When you choose Java, C# or C++ as a transformation language, MapForce generates the required projects and solutions so that you can open them directly in Visual Studio or Eclipse. For advanced data integration scenarios, you can also extend the generated program with your own code, using Altova libraries and the MapForce API.

BUILT-IN
When you select BUILT-IN as a transformation language for your mapping, MapForce uses its native transformation engine to execute the data mapping. MapForce also uses this option implicitly whenever you preview the output of a mapping where the selected transformation language is Java, C#, or C++.

The BUILT-IN engine executes mappings without the need for any external processors, which may be a good choice if memory usage is an issue. If you do not need to generate program code in a specific language, use BUILT-IN as a default option, because it supports most MapForce features compared to other languages (see

Support Notes [1889]). Furthermore, if you select BUILT-IN as a transformation language, you will be able to automate the mapping with MapForce Server. For more information, see Automation with Altova Products [1369].

1.2.3 Mapping Scenarios

The scenarios can differ on the following criteria: (i) sources and targets, and (ii) complexity of mappings. Different data structures can be used as sources and targets: e.g., XML Schema, an XML file with an assigned schema, databases etc. To find out more about the acceptable formats of sources and targets, see Mapping: Sources and Targets [26].

The complexity of mapping designs is illustrated in but not limited to the following scenarios:

- Mapping one source to one target. For more information about this type of mapping, see Tutorial 1 [43].
- Merging multiple data sources into one target. For more information, see Tutorial 2 [53].
- Filtering the data in such a way that only a subset of this data is mapped to the target file. See Tutorial 3 [60].
- Mapping the structure and content of the source to the target file. See Tutorial 4 [67].

Regardless of the technology you work with, MapForce typically determines automatically the structure of your data or suggests supplying a schema for your data. MapForce can also generate schemas from a sample instance file. For example, if you have an XML instance file but no schema definition, MapForce can generate it for you. Thus, MapForce makes the data inside the XML file available for mapping to other files or formats. To find out more about the basic terms and features of MapForce, see Basic Tasks [76] and User Interface Overview [32].

For easier access and management, you can organize your data mapping designs into mapping projects. This feature is available for MapForce Professional and Enterprise editions. In addition to generating code for individual mappings within the project, you can generate program code from entire projects.

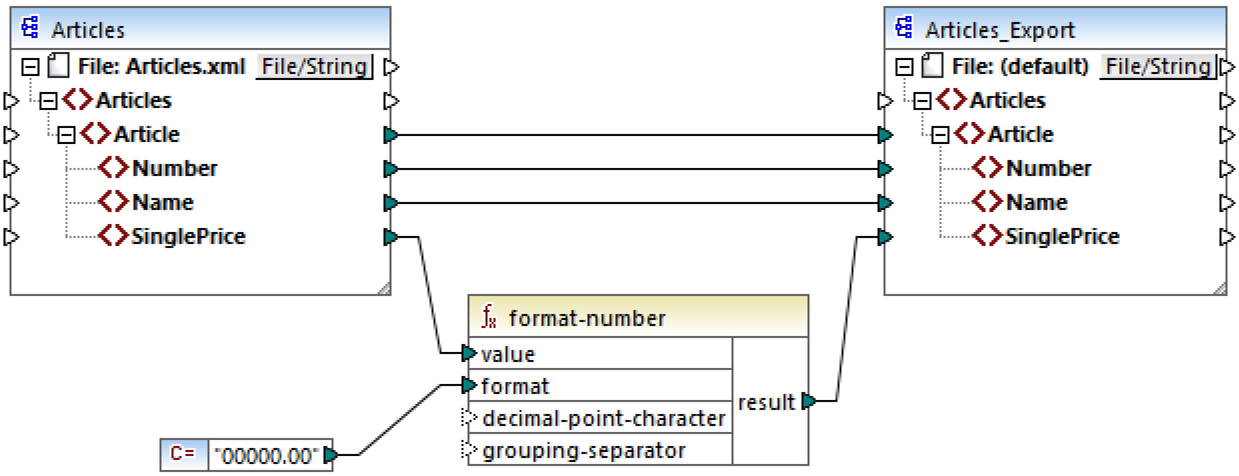
In MapForce, you can completely customize not only the look and feel of the development environment, but also various settings related to each technology and mapping component type. For example:

- When mapping to or from XML, you can choose (i) whether to include a schema reference, or (ii) whether the XML declaration must be suppressed in the output XML files. You can also choose the encoding of the generated files (for example, UTF-8).
- When mapping to or from databases, you can define settings such as the time-out period for executing database statements. It is also possible to choose whether MapForce should use database transactions, or whether it should strip the database schema name from table names when it generates code.
- In the case of XBRL, you can select the structure views that MapForce should display: the **Presentation and definition linkbases** view, the **Table Linkbase** view, or the **All concepts** view.

The examples below illustrate mapping designs that use the same (*Example 1*) and different (*Example 2*) types of source and target structures. Both mapping examples are simple in that only one data source and one data target are used. To find out about more advanced mappings, tasks and procedures, see Advanced Mapping Scenarios [1245].

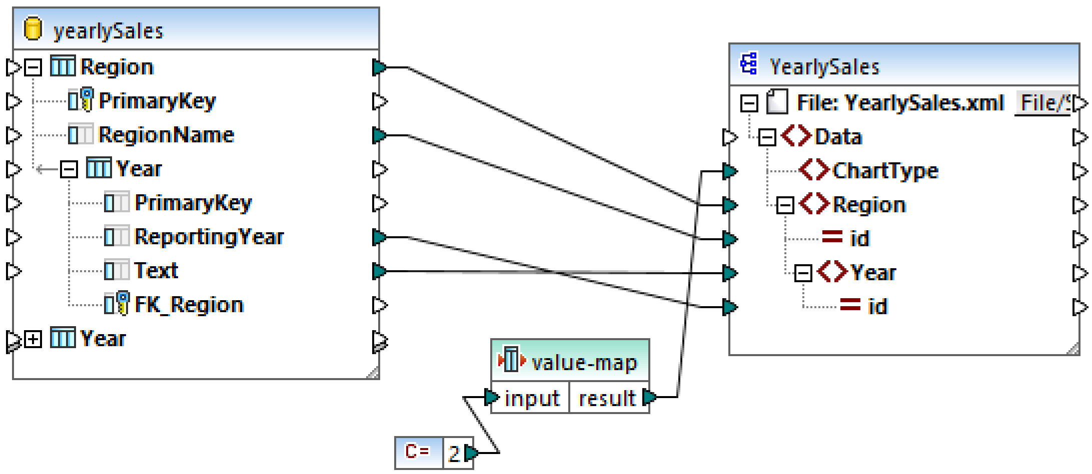
Example 1: XML mapping

MapForce allows designing all mapping transformations visually. For example, in the case of XML, you can connect any element and attribute in an XML file to an element or attribute of another XML file. Thus, you instruct MapForce to read the data from the source node and write it to the target node. The transformation of one XML file into another XML file is illustrated below:



Example 2: Database mapping

When you work with databases in MapForce Professional or Enterprise editions, you can see any database column in the MapForce mapping area and map data to or from it by making visual connections. As with other Altova MissionKit products, when you set up a database connection from MapForce, you can flexibly choose a database driver and a connection type (ADO, ADO.NET, ODBC, or JDBC) depending on your existing infrastructure and data mapping needs. Additionally, you can visually build SQL queries, use stored procedures or query a database directly (support depends on the database type, edition and driver). An example of data transformation from a database into an XML file is given below:



1.2.4 Integration with Altova Products

Transformations can be run inside MapForce using built-in XSLT/XQuery engines. MapForce can also be used in tandem with other Altova products (*see below*).

XMLSpy
If XMLSpy is installed on the same machine, you can conveniently open and edit any supported file types by opening XMLSpy directly from the relevant MapForce contexts. For example, the menu command **Component | Edit Schema Definition in XMLSpy** is available when you click an XML component.
RaptorXML Server
You can choose to run the generated XSLT code directly in MapForce and preview the data transformation result immediately. When you need increased performance, you can process the mapping using RaptorXML Server, an ultra-fast XML transformation engine.

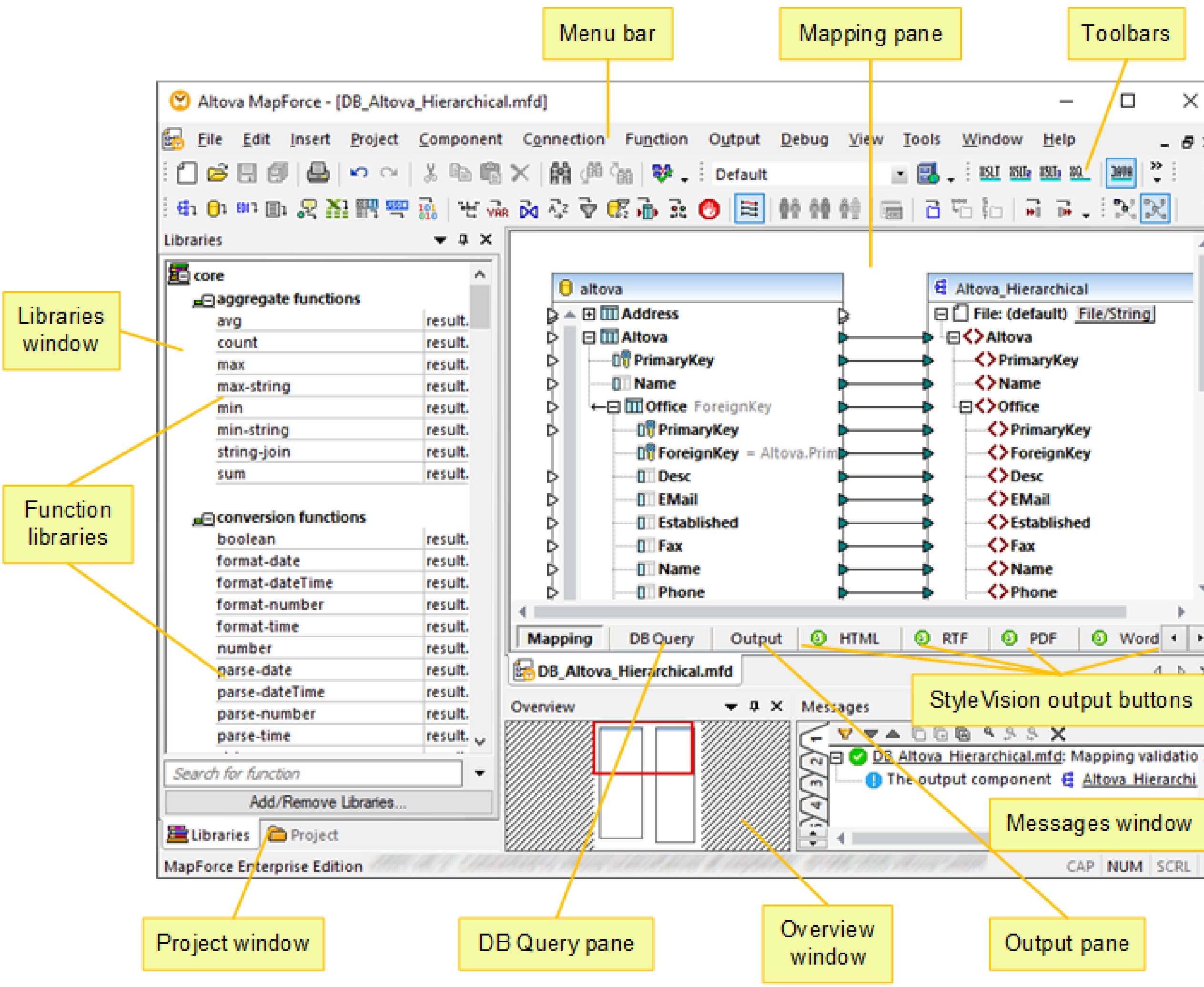
MapForce Server (Enterprise and Professional editions)
You can automate MapForce tasks with the help of Altova MapForce Server, which can be installed on Windows, Linux, and macOS systems. MapForce Server enables you to run the transformations specified in a mapping, not only from the command line of the respective OS but also through API calls (.NET, COM, Java).
FlowForce Server (Enterprise and Professional editions)
You can also automate MapForce tasks with the help of Altova FlowForce Server, which can be installed on Windows, Linux, and macOS systems. FlowForce Server enables you to carry out MapForce Server tasks according to a schedule.
StyleVision (Enterprise and Professional editions)
With the help of StyleVision, you can design or reuse existing StyleVision Power Stylesheets and preview the result of the mapping transformations as HTML, RTF, PDF or Word 2007+ documents.
MapForce as a plug-in
MapForce Professional and Enterprise editions can be installed as a plug-in of Visual Studio and Eclipse integrated development environments. This way, you can design mappings and get access to the MapForce functionality without leaving your preferred development environment.

For more information about automating tasks, see Automating MapForce Tasks with Altova Products[1369]. To find out more about using MapForce as a plug-in, see Plug-in for Visual Studio[1426] and Plug-in for Eclipse[1429].

1.3 User Interface Overview

The graphical user interface of MapForce is organized as an integrated development environment. The main interface components are illustrated below. You can change the interface settings by using the menu command **Tools | Customize**. Use the buttons displayed in the upper-right corner of each window to show, hide, pin, or dock it. If you need to restore toolbars and windows to their default state, use the menu command **Tools | Restore Toolbars and Windows**.

The image below illustrates the main parts of the MapForce graphical user interface.



For more information about the features and functions of each part, see the respective topic below.

In this section

This section is organized in the following way:

- Bars 33

- Windows 33
- Messages Window 36
- Panes 38

1.3.1 Bars

This topic gives an overview of the available bars.

Menu bar and toolbars

The **Menu** bar displays the menu items. Each toolbar displays a group of buttons representing MapForce commands. You can reposition the toolbars by dragging their handles to a desired location. The screenshot below illustrates the the **Menu** bar and toolbars. The actual interface depends on your MapForce edition and the settings you choose.

Application status bar

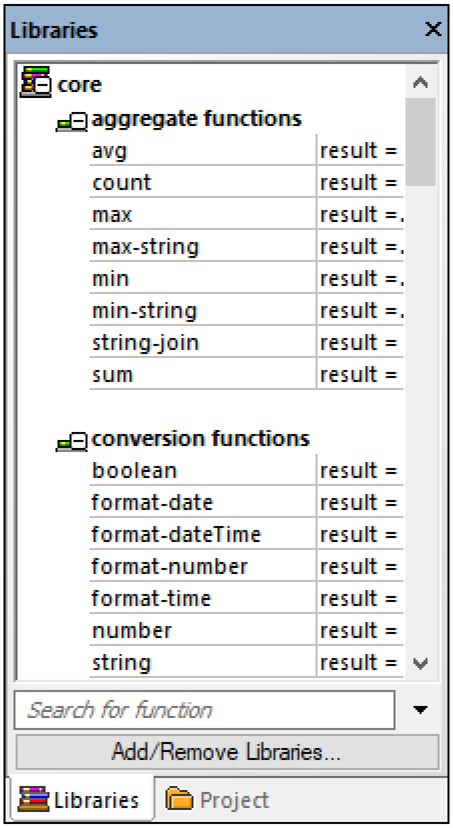
The application status bar appears at the bottom of the MapForce window and shows application-level information. Tooltips are displayed when you move the mouse over a toolbar button. If you are using the 64-bit version of MapForce, the application name appears in the status bar with the x64 suffix. There is no suffix for the 32-bit version.

1.3.2 Windows

This topic gives an overview of the available windows.

Libraries window

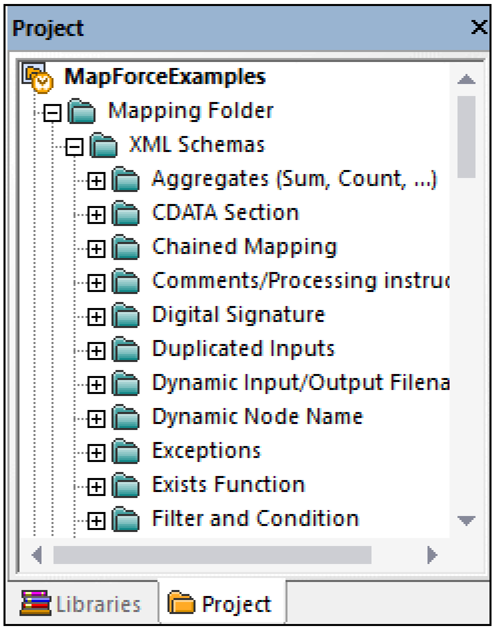
The **Libraries** window lists the MapForce built-in functions organized by library. The list of available functions changes depending on the transformation language you select either from the **Output** menu or from the **Language Selection** toolbar. For more information, see Transformation Languages 27. If you have created user-defined functions or imported external libraries, they also appear in the **Libraries** window.



To search functions by name or by description, enter the search value in the text box at the bottom of the **Libraries** window. To find all occurrences of a function (within the currently active mapping), right-click the function and select **Find All Calls** from the context menu. You can also view the function data type and description directly from the **Libraries** window. For more information, see Functions [904].

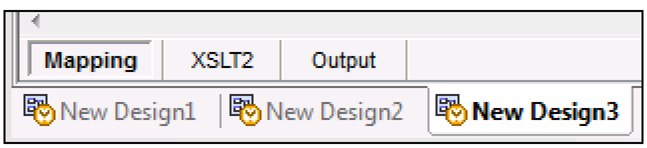
Project window (Enterprise and Professional editions)

MapForce supports the Multiple Document Interface and allows grouping your mappings into mapping projects. The **Project** window shows all files and folders that have been added to the project. Project files have a **`*.mfp`** (MapForce Project) extension. To search for mappings inside projects, click anywhere inside the **Project** window and press **CTRL + F**. For more information, see Projects [116].



Mapping window(s)

MapForce uses a Multiple Document Interface (MDI). Each mapping file you open in MapForce has a separate window. This enables you to work with multiple mapping windows and arrange or resize them in various ways inside the main (parent) MapForce window. You can also arrange all open windows using the standard Windows layouts: Tile Horizontally, Tile Vertically, Cascade. When multiple mappings are open in MapForce, you can quickly switch between them using the tabs displayed under the **Mapping** pane (*see screenshot below*).



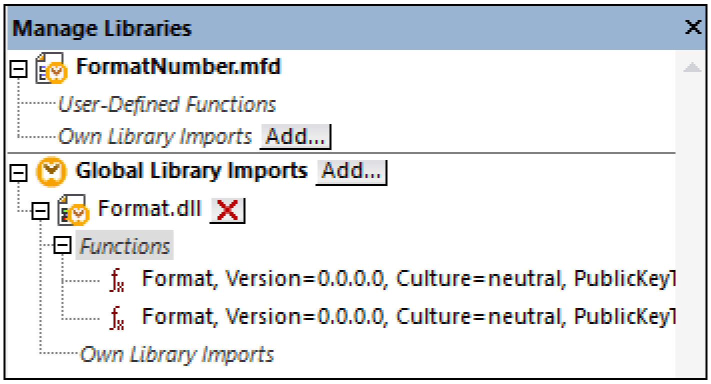
You can access Window management options using the menu command **Window | Windows**. The **Windows** dialog box allows you to perform various actions including activating, saving, closing, or minimizing open mapping windows. To select multiple windows in the **Windows** dialog box, click the required entries while holding the **Ctrl** key pressed.

Manage Libraries window

From this window you can view and manage all user-defined functions (UDFs) and imported custom libraries (including compiled Java .class files and .NET DLL assembly files) that are used by the currently open mappings.

By default, the **Manage Libraries** window is not visible. To display it, do one of the following:

- In the **View** menu, click **Manage Libraries**.
- Click **Add/Remove Libraries** at the bottom of the **Libraries** window.



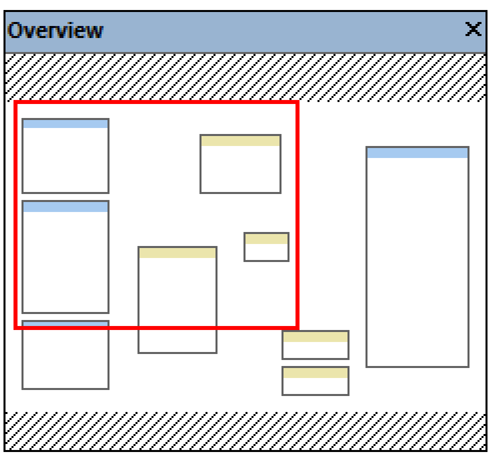
You can choose to view UDFs and libraries only for the mapping document that is currently active or for all open mapping documents. To view imported functions and libraries for all of the currently open mapping documents, right-click inside the window and select **Show Open Documents** from the context menu.

To display the path of the open mapping document instead of the name, right-click inside the window and select **Show File Paths** from the context menu.

For more information, see Manage Function Libraries 908.

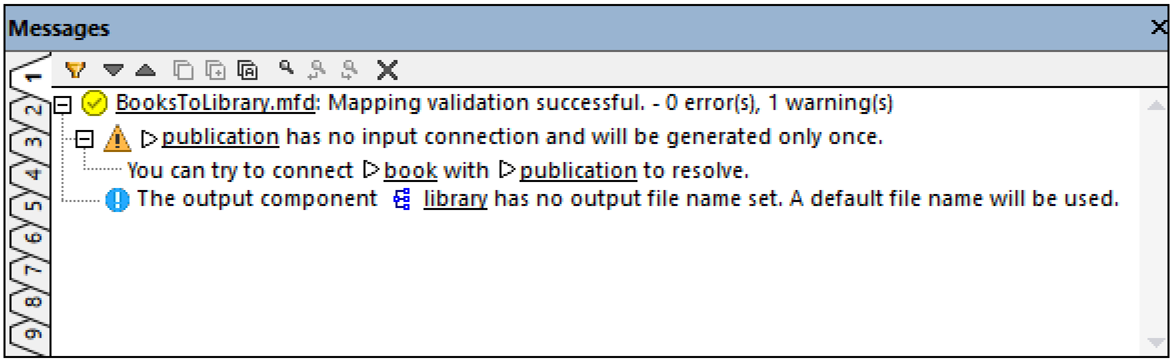
Overview window

The **Overview** window gives a bird's-eye view of the Mapping pane 38. Use it to navigate quickly to a particular location in the mapping area when the size of the mapping is very large. To navigate to a particular location in the mapping, click and drag the red rectangle.



1.3.3 Messages Window

The **Messages** window (*see screenshot below*) shows validation statuses, messages, errors, and/or warnings when you preview or validate 105 a mapping. Click the underlined text in the **Messages** window to see a component or structure which caused the information, warning, or error message.



Validation status icons

When you validate a mapping, MapForce checks, for example, for unsupported component kinds, incorrect or missing connections. The validation result is displayed in the **Messages** window with one of the following status icons:

Icon	Meaning
	Validation has completed successfully.
	Validation has completed with warnings.
	Validation has failed.

The **Messages** window may additionally display any of the following message types: information messages, warnings, and errors.

Icon	Meaning
	Indicates an information message. Information messages do not stop the mapping execution.
	Indicates a warning message. Warnings do not stop the mapping execution. They appear, for example, when you do not create connections to some mandatory input connectors. In such cases, the output will still be generated for those components where valid connections exist.
	Indicates an error. When an error occurs, the mapping execution fails, and no output is generated. The preview of the XSLT or XQuery code is not possible.

To highlight the component or structure which caused the information, warning, or error message, click the underlined text in the **Messages** window.

Message-related actions

The **Messages** window enables you to take the following actions:

Icon	Description
	Filter messages by severity: information messages, errors, and warnings. Select **Check All** to include all severity levels (this is the default behavior). Select **Uncheck All** to remove all severity levels from the filter. In this case, only the general execution or validation status message is displayed.
	Jump to the next line.
	Jump to the previous line.
	Copy the selected line to the clipboard.
	Copy the selected line to the clipboard, including any lines nested under it.
	Copy the full contents of the **Messages** window to the clipboard.
	Find a specific text in the **Messages** window. Optionally, to find only words, select **Match whole word only**. To find text while preserving the upper or lower case, select **Match case**.
	Find a specific text starting from the currently selected line up to the end.
	Find a specific text starting from the currently selected line up to the beginning.
	Clear the Messages window.

When you work with multiple mapping files simultaneously, you might want to display information, warning, or error messages in individual tabs for each mapping. In this case, click the numbered tabs available on the left side of the **Messages** window before validating the mapping.

1.3.4 Panes

This topic gives an overview of the available panes.

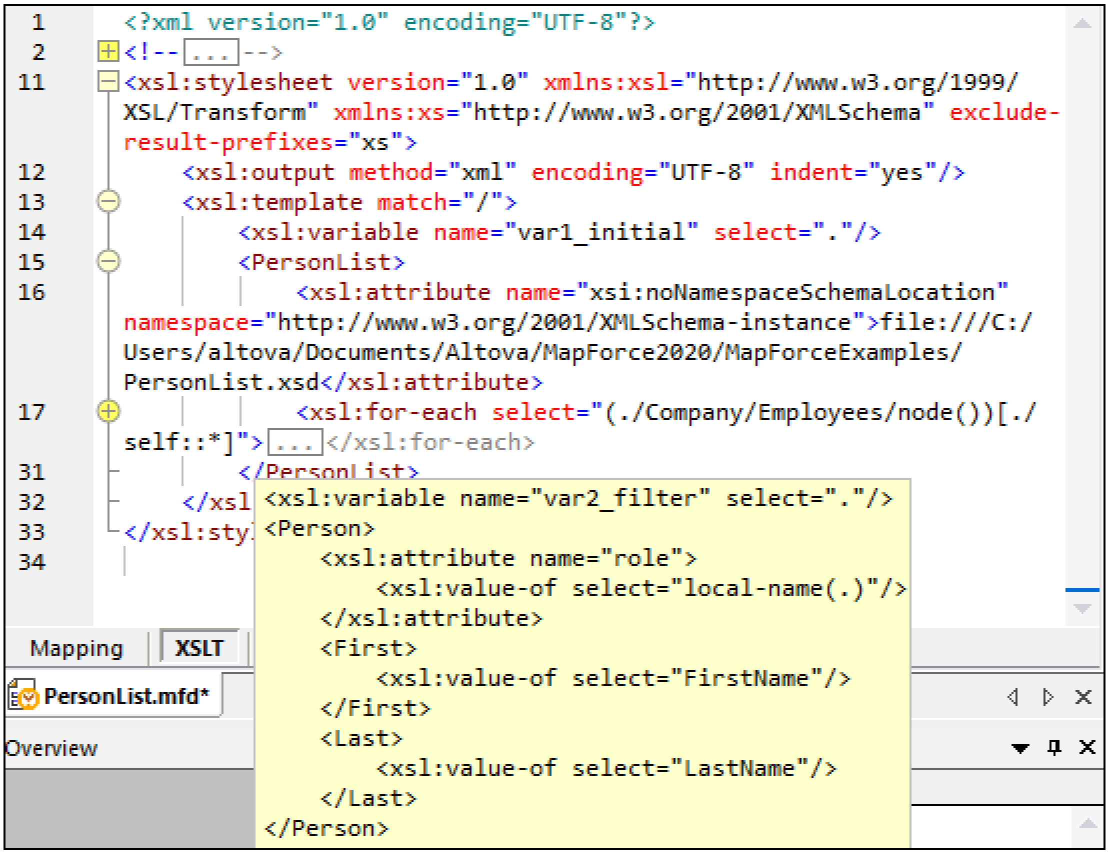
Mapping pane

The **Mapping** pane is the working area where you design mappings [105]. You can add mapping components (e.g., files, schemas, constants, variables, and so on) to the mapping area from the **Insert** menu. For more information, see Add Components to Mapping [80]. You can also drag functions from the **Libraries** window into the **Mapping** pane. For details, see Add a Function to the Mapping [905].

XSLT pane

The **XSLT** pane displays the XSLT transformation code generated from your mapping. To switch to this pane, select XSLT, XSLT 2 or XSLT3 as a transformation language [27] and click the tab with the same name.

This pane provides line numbering and code folding functionality. To expand or collapse portions of code, click the + and - icons at the left side of the window. Any portions of collapsed code are displayed with an ellipsis symbol. To preview the collapsed code without expanding it, move the mouse cursor over the ellipsis. This opens a tooltip that displays the code being previewed, as shown in the image below. Note that, if the previewed text is too big to fit into the tooltip, an additional ellipsis appears at the end of the tooltip.



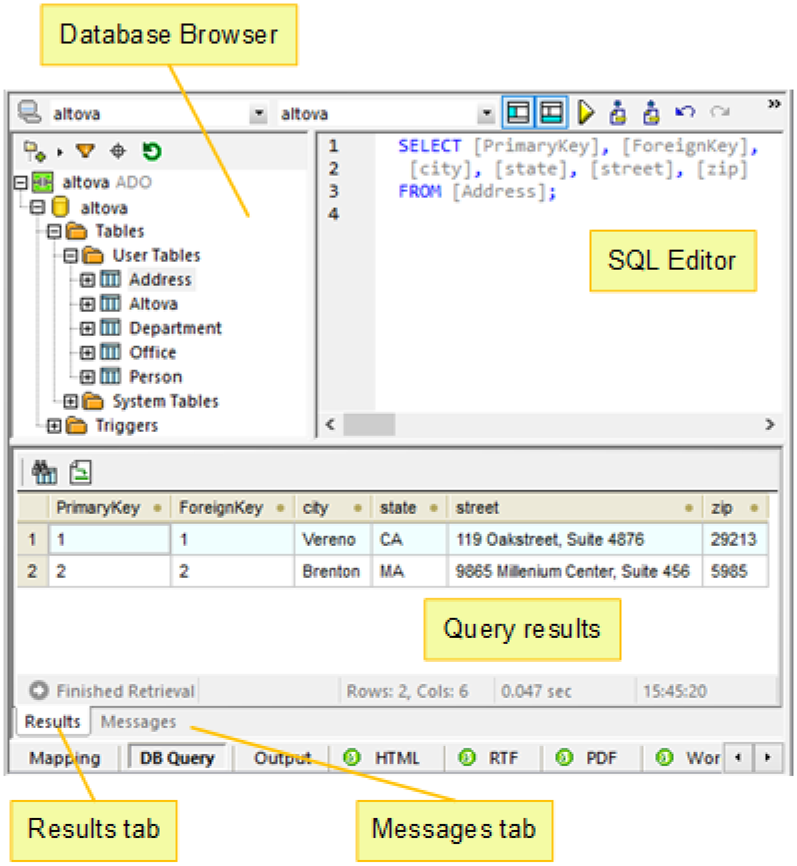
To configure the display settings, including the indentation, end of line markers, and others, right-click the pane and select **Text View Settings** from the context menu. Alternatively, click (**Text View Settings**) in the toolbar.

XQuery pane (Enterprise and Professional editions)

The **XQuery** pane displays the XQuery transformation code generated from your mapping when you click the **XQuery** button. This pane is available when you select XQuery as a transformation language. This pane also provides line numbering and code folding functionality, which works in a similar way as in the XSLT pane (see above).

DB Query Pane (Enterprise and Professional editions)

The **DB Query** pane allows you to directly query any major database. You can work with multiple active connections to different databases. For more information, see Browsing and Querying Databases 376.



Output pane

The **Output** pane displays the result of the mapping transformation. If the mapping generates multiple files, you can navigate sequentially through each generated file.

This pane also provides line numbering and code folding functionality, which works in a similar way as in the XSLT pane (see above).

StyleVision Output Panes (Enterprise and Professional editions)

If you have installed Altova StyleVision, the StyleVision output panes will become available next to the **Output** pane. The StyleVision output panes enable you to preview and save the mapping output in HTML, RTF, PDF, and Word 2007+ formats. This is possible thanks to StyleVision Power Stylesheet (SPS) files designed in StyleVision and assigned to a mapping component in MapForce.

2 Tutorials

With the help of these tutorials, you will be able to understand and use the basic data transformation capabilities of MapForce. You will be guided through the basics step by step. The tutorials gradually grow in complexity. Therefore, it is recommended to follow them sequentially. Basic knowledge of XML and XML Schema will be advantageous.

Example files
The mapping files illustrated or referenced in these tutorials are available in the BasicTutorials folder 26. When you are in doubt about the possible effects of changing the original MapForce examples, create back-ups before changing them.

List of tutorials

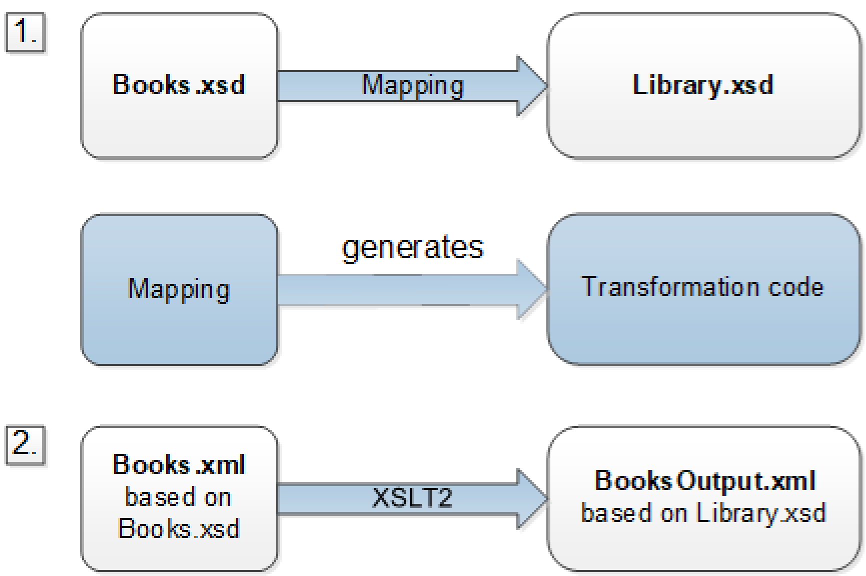
One source to one target
This tutorial 43 shows how to use key MapForce mechanisms to map the nodes of a source file to the nodes of a target file. The tutorial goes on to explain how to convert an XML file defined by one XML schema to an XML file defined by a different XML schema.
Multiple sources to one target
This tutorial 53 shows how to merge data from multiple source XML files to one target file.
Chained mappings
In this tutorial 60, we create a simple mapping as in the first tutorial, then filter the data produced by this mapping and pass the filtered data to the second target file.
Multiple sources to multiple targets
This tutorial 67 shows how to read data from multiple XML instance files located in the same folder and write this data to multiple XML files generated on the fly.

2.1 One Source to One Target

This tutorial describes how to create a mapping for one of the most basic scenarios. Our goal is to take the data from XML file A with XML schema A assigned to it and put this data into XML file B with XML schema B assigned to it. Thus, our target file will have the same data as in the source but with a different schema (structure). Thanks to the transformation code, you will be able to see how the structure of the source file has changed. Note the mapping could be carried out only between the structures, but we will not be able to preview the result of the transformation in the Output pane 40. Therefore, for illustration purposes, we use an XML file called **`Books.xml`** (*see code listing below*). The broad outline of our method will be as follows:

1. Since we are using two data structures, we will create two components (*Source* and *Target*) in our mapping design.
2. Then we need to map nodes by connecting a source node to the desired target node. It is these connections that constitute the mapping and determine what source node maps to what target node.
3. Since the transformation of one document into another is carried out by using a suitable transformation language, such as XSLT, we select a transformation language.
4. We use MapForce's built-in transformation engines to transform the source XML document (**`Books.xml`**) into the required target document. This target document will be an XML document that is valid according to the target XSD (**`Library.xsd`**).
5. Finally, we can save the output XML file.

The image below illustrates an abstract model of the data transformation used in this tutorial:



The abstract model above shows two steps of the mapping process. In the first step, the structure of **`Books.xsd`** is mapped to a new structure called **`Library.xsd`**. The mapping is carried out by means of a transformation language 27. The choice of a transformation language depends on your MapForce edition. In our case, XSLT2 is chosen as a transformation language. In the second step, the content of **`Books.xml`**, which has **`Books.xsd`** assigned to it, is mapped to the target file (**`BooksOutput.xml`**) and based on a new schema (**`Library.xsd`**). The code listing below shows sample data from **`Books.xml`** that will be used as a data source.

```
<books>
   <book id="1">
```

```
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <category>Fiction</category>
      <year>1876</year>
   </book>
   <book id="2">
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <category>Fiction</category>
      <year>1912</year>
   </book>
</books>
```

This is how we want our data to look in the target file called **`BooksOutput.xml`**:

```
<library>
   <last_updated>2015-06-02T16:26:55+02:00</last_updated>
   <publication>
      <id>1</id>
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <genre>Fiction</genre>
      <publish_year>1876</publish_year>
   </publication>
   <publication>
      <id>2</id>
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <genre>Fiction</genre>
      <publish_year>1912</publish_year>
   </publication>
</library>
```

Some element names in the source and target XML are not the same. Our goal is to populate the elements `<author>`, `<title>`, `<genre>` and `<publish_year>` of the target file with the content of the equivalent elements in the source file (`<author>`, `<title>`, `<category>`, `<year>`). The attribute `id` in the source file must be mapped to the `<id>` element in the target file. Finally, we must populate the `<last_updated>` element of the target file with the date and time indicating when the file was last updated.

To carry out the required data transformation, take the steps described in the subsections below.

2.1.1 Create and Save Design

This topic explains how to create a new design, select a transformation language, validate and save your mapping.

Create a new design

To be able to carry out a transformation, you will need to create a new mapping design, which can be done in one of the following ways:

- Go to the **File** menu and click **New**. Then select **Mapping** and click **OK**.

- Click in the toolbar. Then select **Mapping** and click **OK**.

Select a transformation language

Depending on your MapForce edition, different transformation languages [27] are available. For this tutorial, we have selected XSLT2. You can select this transformation language in one of the following ways:

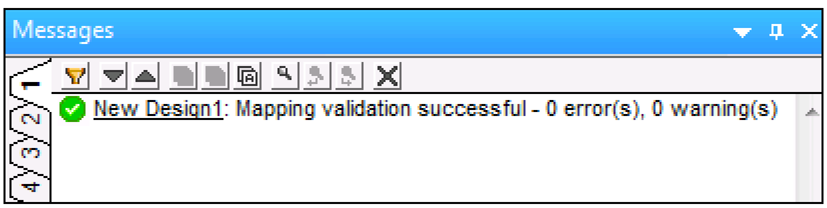
- Click (**XSLT2**) in the toolbar.
- In the **Output** menu, click **XSLT 2.0**.

Validate and save the design

Validating a mapping is an optional step that enables you to see and correct potential mapping errors and warnings before you run the mapping. You can validate your mapping at any stage. To check whether the mapping is valid, do one of the following:

- In the **File** menu, click **Validate Mapping**.
- Click (**Validate**) in the toolbar.

The **Messages** window displays the validation results as follows:



To save the mapping, do one of the following:

- Click **Save** in the **File** menu.
- Click (**Save**) in the toolbar.

For your convenience, the mapping created in this tutorial is saved as **`Tut1_SchemaToSchema.mfd`**.

2.1.2 Add Source Component

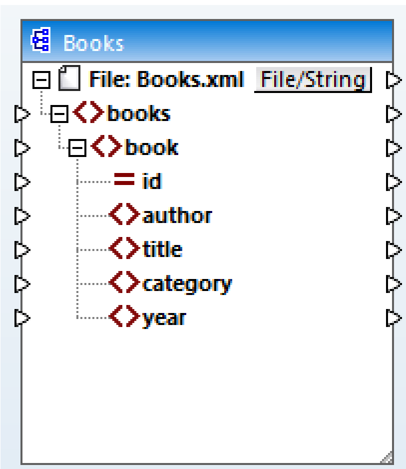
At this stage, we want to add an XSD file that will be the structure of the first component and an XML file that will provide the data for this component. The source file called **`Books.xsd`** can be added to the mapping in one of the following ways:

- Click (**Insert XML Schema/File**) in the toolbar.
- In the **Insert** menu, click **XML Schema/File**.
- Drag **`Books.xsd`** from the Windows Explorer into the mapping area.

When you add a schema, MapForce suggests adding a sample XML file. Click **Browse** and search for **`Books.xml`** that is located in the same folder. Thus, our source file contains both a schema and content. In the properties of every source or target component, we can specify an XSD file and XML fie. The XSD file defines the structure of the document in that component. The XML file provides the data of that component (source or target) and must be valid by the schema of that component. If a component is created from an XSD file, then you are prompted for an XML file that will be used as the component's data file. If a component is created from an XML file, then the XSD that is referenced from the XML file will be used to define the structure of the component's data. If no reference to an XSD exists, MapForce will ask you if it may generate an XSD file for this component.

View the structure

Now that the source file has been added to the mapping area, you can see its structure. In MapForce, this structure is known as a mapping component or simply a component[76]. You can expand elements in the component by clicking the ⊞ icon. Alternatively, you can press the **+** key on the numeric keypad. The screnshot below illustrates the source component:



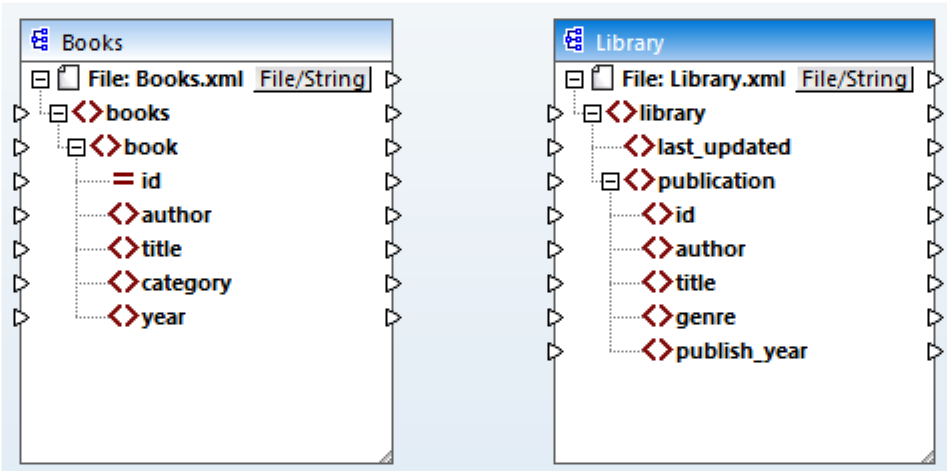
`Books` in the title bar indicates the name of the component. The top level node represents the name of the XML instance file: **`File: Books.xml`**. The XML elements in the structure are represented by the icon. XML attributes are represented by the icon. The small triangles, displayed on both sides of the component, represent data inputs on the left side and outputs on the right side. In MapForce, these triangles are called *input connectors* and *output connectors*, respectively.

Move and resize components

To move the component inside the mapping pane, click the component header and drag the mouse to a new position. To resize the component, drag the bottom right-hand corner of the component. You can also double-click this corner so that MapForce adjusts the size automatically.

2.1.3 Add Target Component

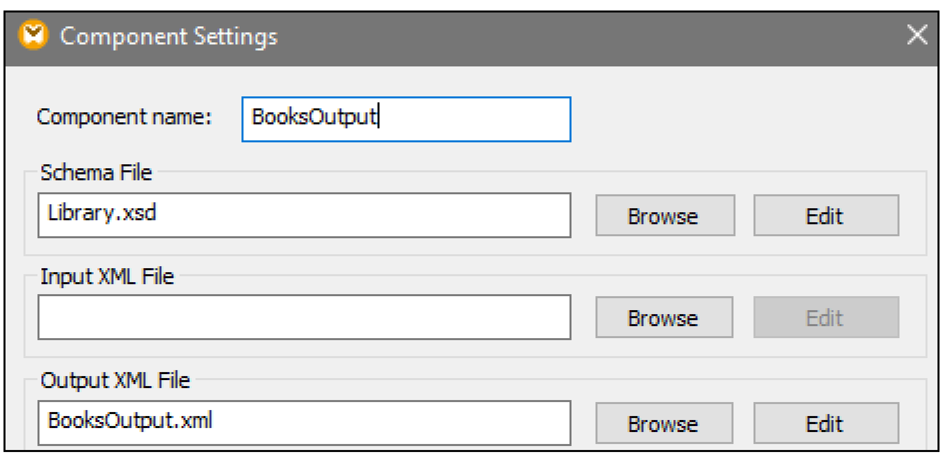
The next step is to add a target component and define its settings. To add the target file called **`Library.xsd`** to the mapping, click (**Insert XML Schema/File**). Click **Skip** when MapForce suggests supplying an instance file. At this stage, the mapping design looks as follows:



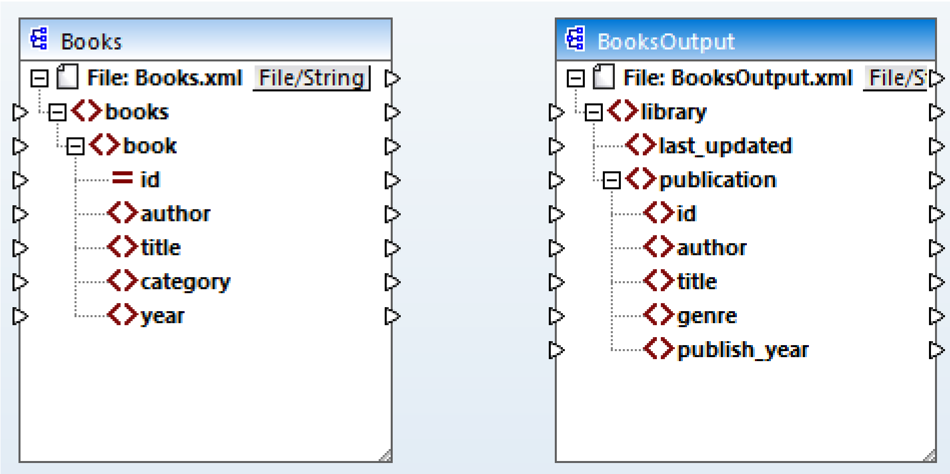
Note that when you open **`Library.xsd`**, it is displayed as an XML file in the component. In fact, MapForce only creates a reference to the XML file called **`Library.xml`**, but this XML file itself does not yet exist. Thus, out target component has a schema but no content.

Component settings

Now we need to rename the target component **`BooksOutput.xml`**. This will allow us to avoid confusion in the next tutorials, as we are going to use a separate file called **`Library.xml`**, which has its own content and is based on the same **`Library.xsd`** schema. In order to rename the target file, double-click the header of the target component. This opens the Component Settings dialog box 123 (*see screenshot below*), in which we need to change the name of the target file as follows:



The mapping design now looks as follows:



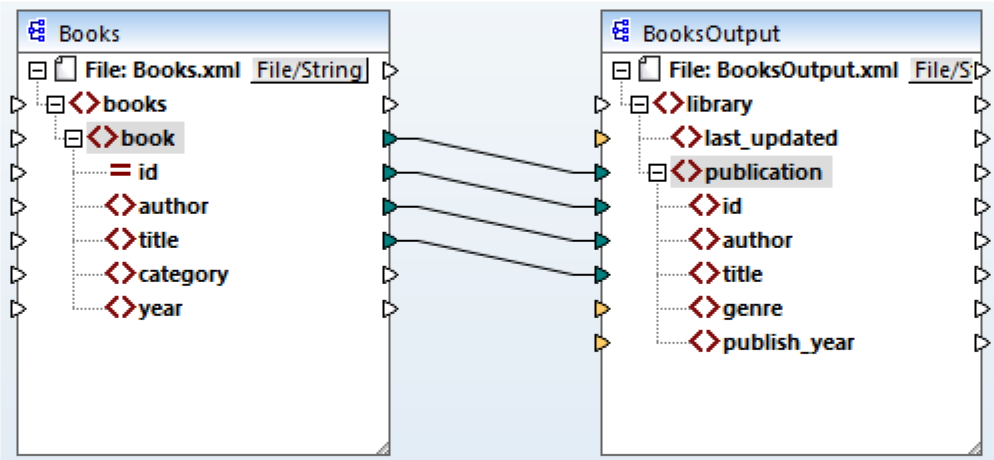
Note: The name of the header only refers to the name of the component and not the name of the schema this file is based on. To see the name of the schema, open the **Component Settings** dialog box.

2.1.4 Connect Source and Target

In this step, we will map the data in the source file to the target file. The goal is to map two types of input to the target nodes: (i) source nodes and (ii) data, both of which are mapped simultaneously. Most of the data comes from **`Books.xml`**. In our example, we will also supply information about the current date and time using the XPath function `current-dateTime` 1146.

Automatic connections

We will now create a mapping connection between the `<book>` element in the source component and the `<publication>` element in the target component. To do this, click the output connector (the small triangle) to the right of the `<book>` element and drag it to the input connector of the `<publication>` element in the target. When you do this, MapForce may automatically connect all the child elements of `<book>` in the source file to the elements with the same names in the target file. In our example, four connections have been created simultaneously (*see screenshot below*). This feature is called *Auto Connect Matching Children* and can be disabled and customized if necessary.



You can enable or disable **Auto Connect Matching Children** in one of the following ways:

- Click (**Toggle auto connect of children**) in the toolbar.
- In the **Connection** menu, click **Auto Connect Matching Children**.

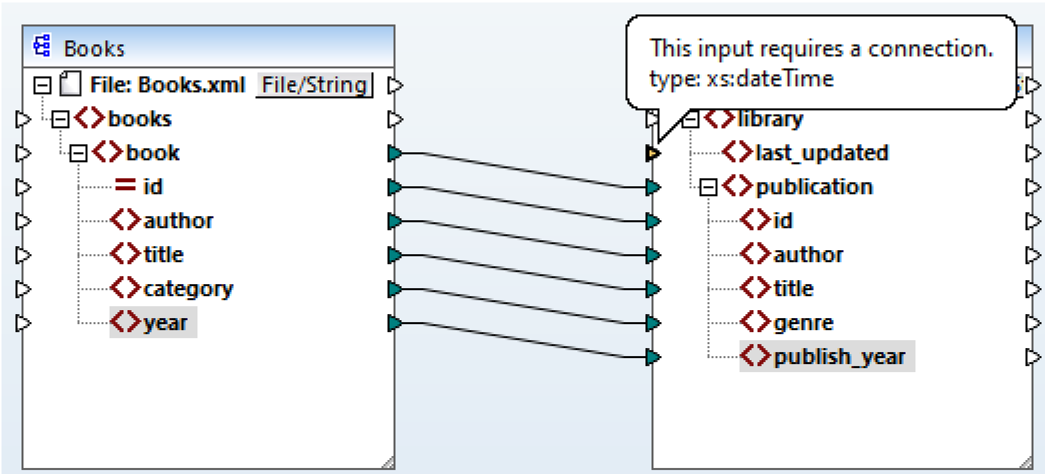
Connect mandatory items

Notice that some of the input connectors in the target component have been highlighted by MapForce in orange, which indicates that these items are mandatory. They are mandatory, because they were set in such a way in the the file's schema. To ensure the validity of the target XML file, provide values for the mandatory items as follows:

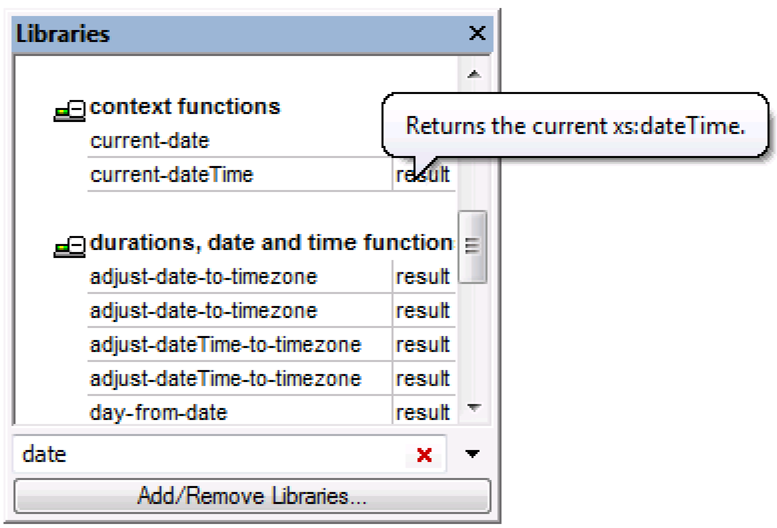
- Connect the `<category>` element in the source with the `<genre>` element in the target component.
- Connect the `<year>` element in the source with the `<publish_year>` element in the target component.

Add the current date and time

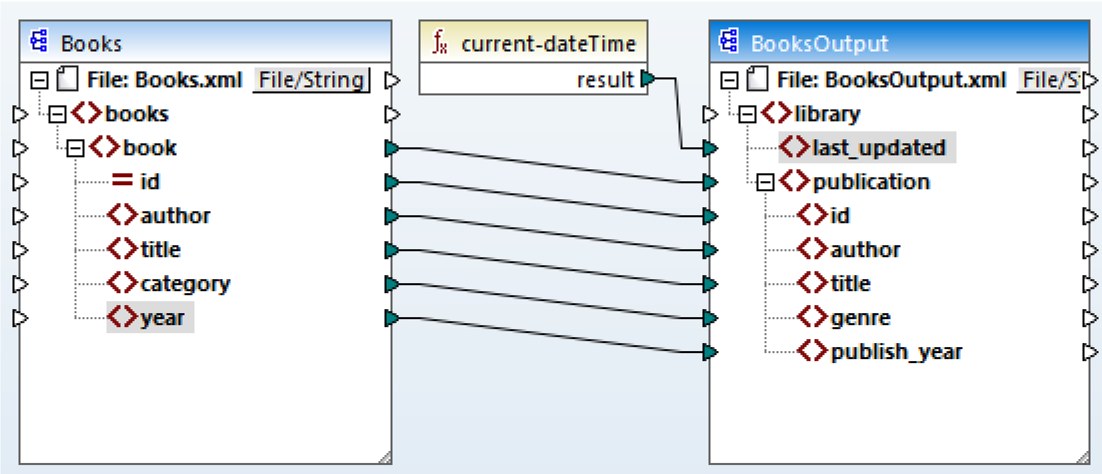
Finally, you need to supply a value for the `<last_updated>` element. If you hover over its input connector, you can see that the element is of type `xs:dateTime` (*see screenshot below*). To be able to see tips, press the toolbar button (**Show tips**). By clicking (**Show Data Types**) in the toolbar, you can also make the data type of each item visible at all times.



You can get the current date and time by means of an XSLT2 function **`current-dateTime`**. To find this function, type it in the text box located at the bottom of the Libraries window[33] (*see screenshot below*). Alternatively, double-click an empty area inside the **Mapping** pane and start typing `current-date`.



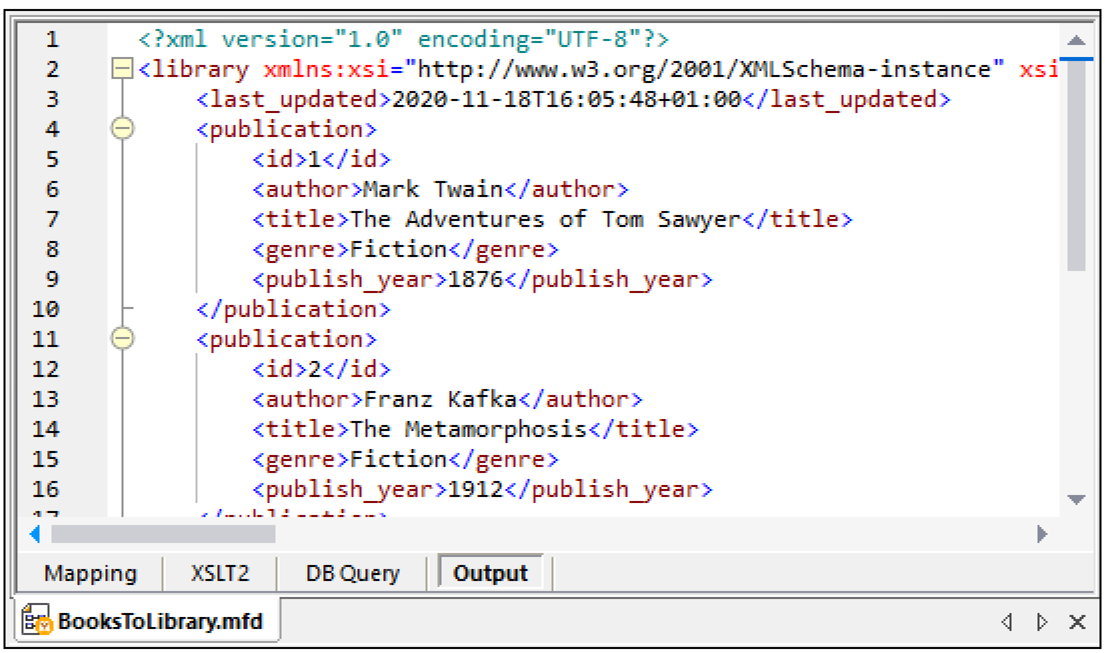
To add the function to the mapping, drag the function into the **Mapping** pane and connect its output to the input of the `<last_updated>` element (*see screenshot below*).



You can now validate and save your mapping, as shown in Create and Save Design 44.

2.1.5 Preview Mapping Result

MapForce uses its built-in engines to generate the output and allows previewing the result of the mapping directly in the **Output** pane (*see screenshot below*).



By default, the files displayed in the **Output** pane are not saved to disk. Instead, MapForce creates temporary files. To save the output, open the **Output** pane and select the menu command **Output | Save Output File** or click (**Save generated output**) in the toolbar.

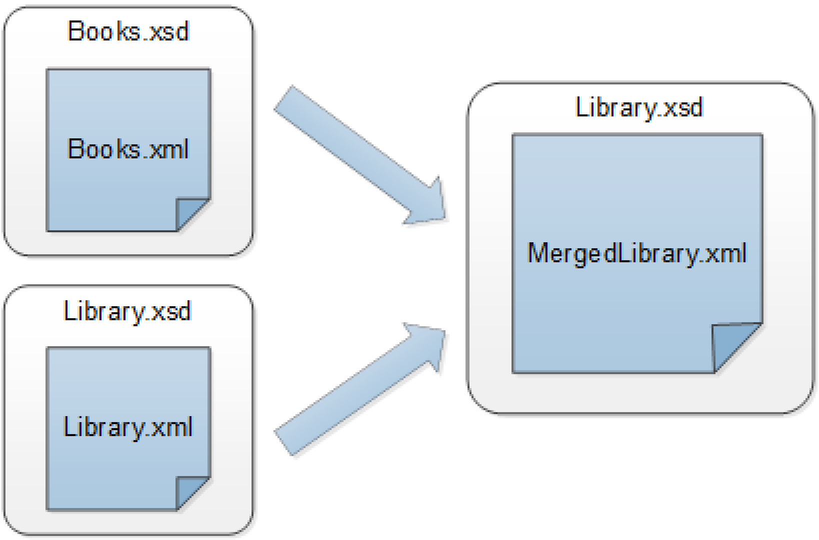
To configure MapForce to write the output directly to final files instead of temporary ones, go to **Tools | Options | General** and select the check box **Write directly to final output files**. Note that enabling this option is not recommended while you follow this tutorial, because you may unintentionally overwrite the original tutorial files.

You can also preview the generated XSLT code that performs the transformation. To preview the code, click the **XSLT2** button located at the bottom of the **Mapping** pane. To generate the XSLT2 code and save it to a file, select the menu item **File | Generate Code in | XSLT 2.0**. When prompted, select a folder where the generated code must be saved. After the code generation has been completed, the destination folder will include the following two files:

1. An XSLT transformation file, named after the target schema. This transformation file has the following format: **`MappingMapTo<TargetFileName>.xslt`**.
2. A **`DoTransform.bat`** file, which enables you to run the XSLT transformation with [Altova RaptorXML Server]() from the command line. In order to run the command, you will need to install RaptorXML.

2.2 Multiple Sources to One Target

In this tutorial, you will learn to merge the data from a new file called **`Library.xml`** with the data from **`Books.xml`**. The result will be a target file called **`MergedLibrary.xml`**, which will contain the data from both source files. The target file will be based on the **`Library.xsd`** schema. Note that the source files have different schemas. If the source files had the same schema, you could also merge their data using a different approach, described in Multiple Sources to Multiple Targets [67]. The image below represents an abstract model of the data transformation described in this tutorial.



The code listing below shows an extract from **`Books.xml`**, the file that will be used as the first data source.

```
<books>
   <book id="1">
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <category>Fiction</category>
      <year>1876</year>
   </book>
</books>
```

The code listing below shows an extract from **`Library.xml`**, the file that will be used as the second data source:

```
<library>
   <publication>
      <id>5</id>
      <author>Alexandre Dumas</author>
      <title>The Three Musketeers</title>
      <genre>Fiction</genre>
      <publish_year>1844</publish_year>
   </publication>
</library>
```

This is how we want our merged data to look in the target file called **`MergedLibrary.xml`**:

```
<library>
   <publication>
      <id>1</id>
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <genre>Fiction</genre>
      <publish_year>1876</publish_year>
   </publication>
   <publication>
      <id>5</id>
      <author>Alexandre Dumas</author>
      <title>The Three Musketeers</title>
      <genre>Fiction</genre>
      <publish_year>1844</publish_year>
   </publication>
</library>
```

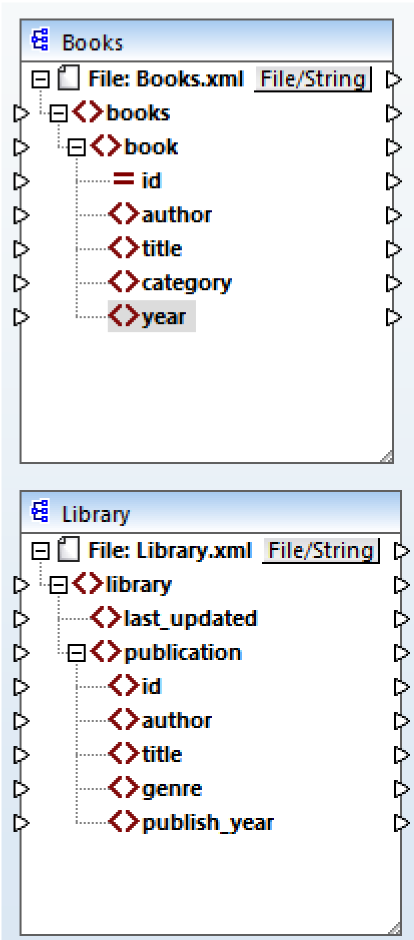
To carry out the transformation, take the steps described in the subsections below.

2.2.1 Prepare Source Files

The starting point of this tutorial is two source files, both of which have a schema (**`Books.xsd`** and **`Library.xsd`**) and data (**`Books.xml`** and **`Library.xml`**). To prepare the source files for the mapping, take the following steps:

1. Open **`Books.xsd`**.
2. When MapForce suggests adding a sample XML file, click **Browse** and open **`Books.xml`**.
3. Open **`Library.xsd`**.
4. When MapForce suggests adding a sample XML file, click **Browse** and open **`Library.xml`**.
5. For convenience, place the source files one on top of the other (*see screenshot below*).

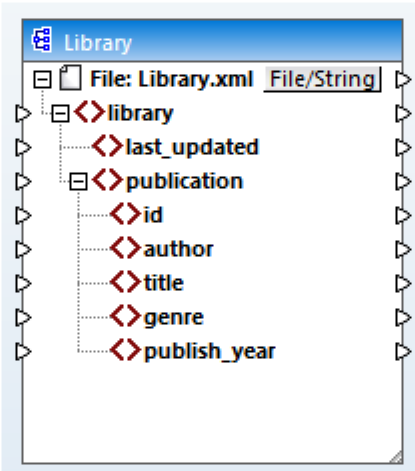
At this stage, our mapping design looks as follows:



As the new mapping references several files from the same folder, make sure to save this new mapping in the **`BasicTutorials`** folder.

2.2.2 Add Target Component

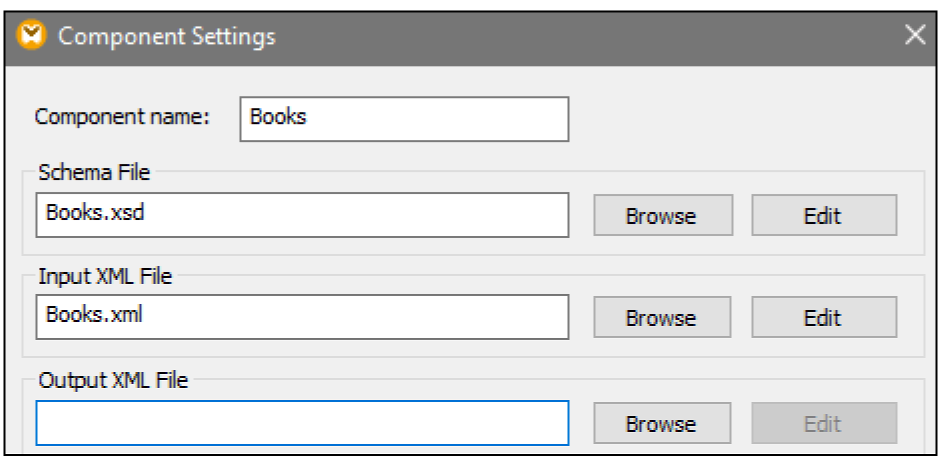
The next step is to add a target file called **`Library.xsd`**. To add the target file, click (**Insert XML Schema/File**). Click **Skip** when MapForce suggests supplying an instance file. Then click the header of the new component and drag it to the right of the two source components. You can always move mapping components in any direction. Nevertheless, placing a source component to the left of a target component will make your mapping easier to read and understand. This is also the convention for all the mappings illustrated in this documentation and the sample mapping files accompanying your MapForce installation.

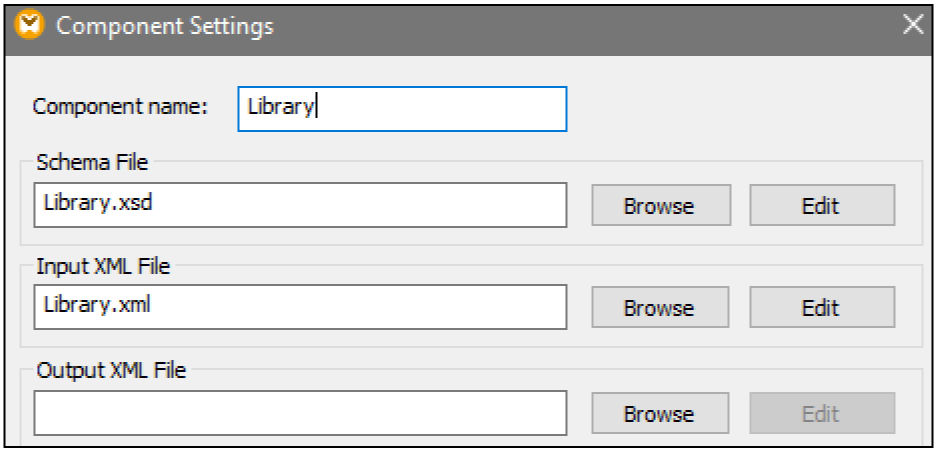


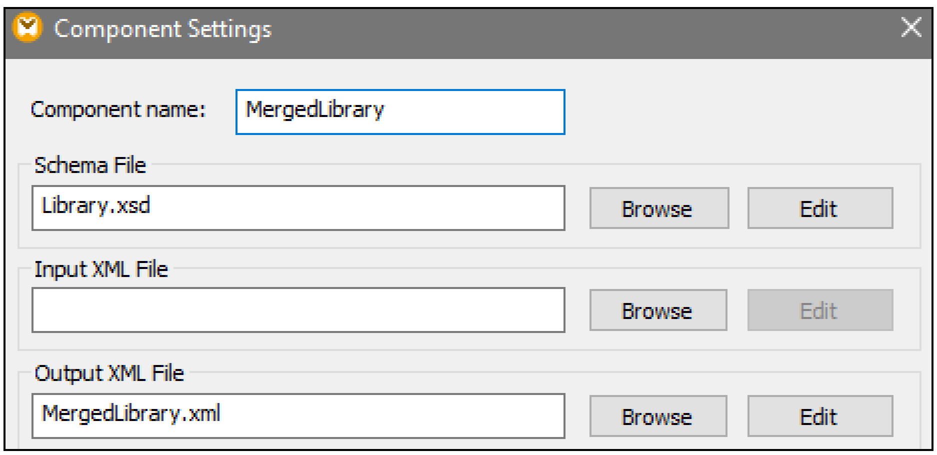
At this stage, the target file only has structure but no data. The data will later be merged from the two source files and mapped to the target file.

2.2.3 Verify and Set Input/Output Files

At his stage, the mapping has two source components (`Books` and `Library`) and one target component (`Library`). Now we have two components with the same name - `Library`. To avoid confusion, we need to change the settings in the Component Settings dialog box[123]. Double-click the header of each component. Then verify and change the name and the input/output files of each component as shown below.







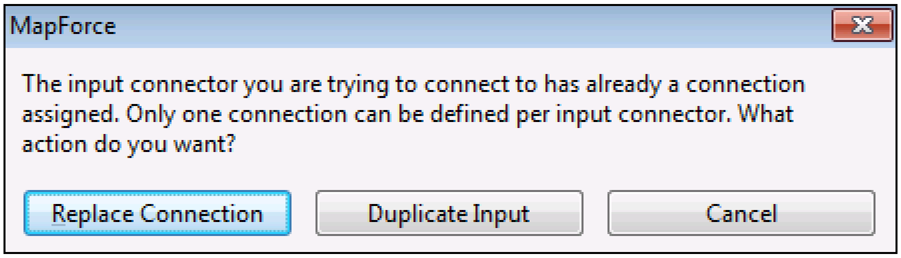
As shown above, the first source component reads data from **`Books.xml`**. The second source component reads data from **`Library.xml`**. Finally, the target component outputs data to a file called **`MergedLibrary.xml`**.

2.2.4 Connect Sources and Target

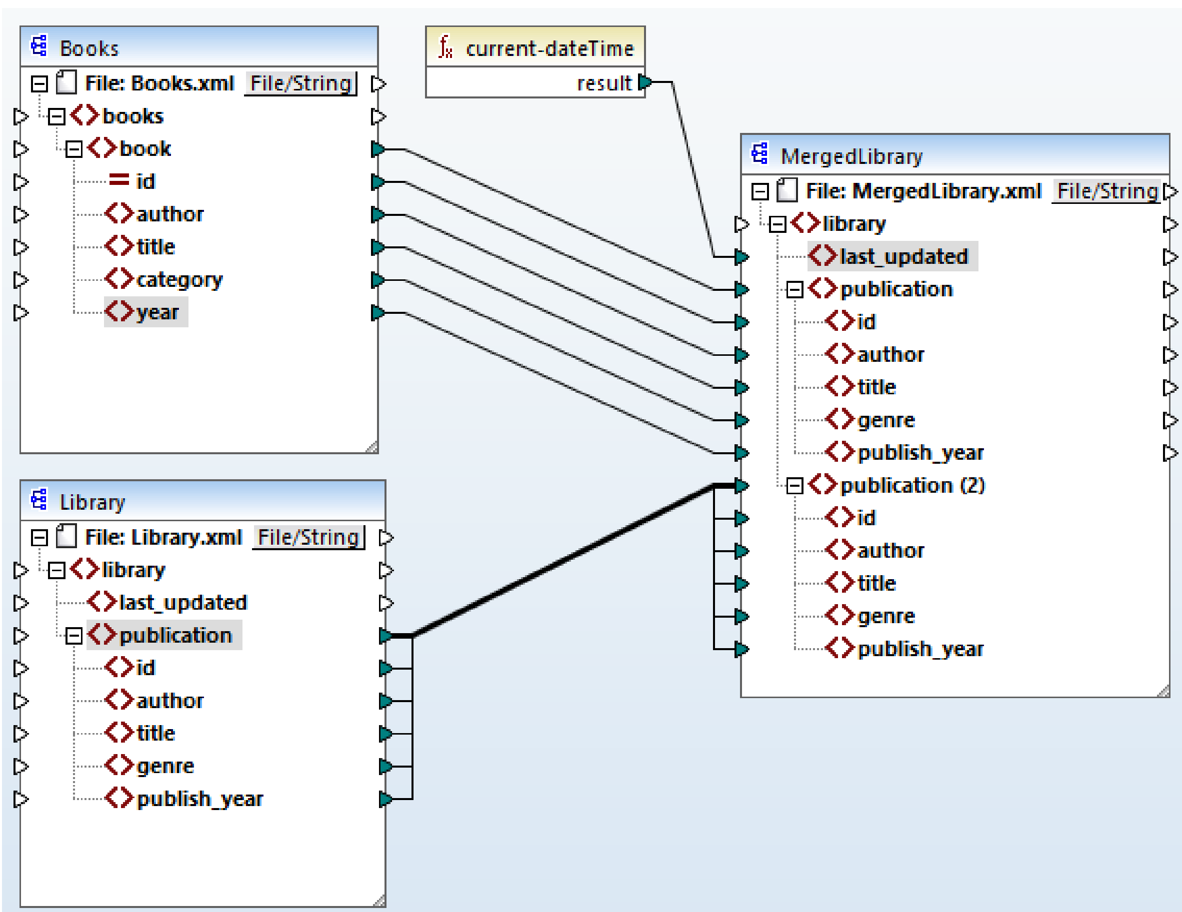
Now we want to map the data from two source files (**`Books.xml`** and **`Library.xml`**) to the target file (**`MergedLibrary.xml`**). To achieve the goal, take the following steps:

- Connect the `<book>` element in the first source component with the `<publication>` element in **`MergedLibrary.xml`**. When you do this, MapForce may automatically connect all the child elements of `<book>` in the source file to the elements with the same names in the target file. In our example, four connections have been created simultaneously. To find out more about the automatic connection of child elements, see the first tutorial 48.
- When you connect the `<book>` element with the `<publication>` element, you will notice that some of the input connectors in the target component have been highlighted in orange. This indicates that these items are mandatory. To ensure the validity of the target XML file, connect the `<category>` element with the `<genre>` element and the `<year>` element with the `<publish_year>` element.

- To supply a value for the `<last_updated>` element, find the function called **`current-dateTime`** in the XSLT2 library. Drag the function to the mapping area and connect `result` with the `<last_updated>` element in **`MergedLibrary.xml`**.
- To instruct MapForce to write data from the second source to the target, press and hold the output connector of the `<publication>` element in **`Library.xml`** and drag it to the input connector of the `<publication>` element in **`MergedLibrary.xml`**. Since the target input connector already has a connection, the following message appears:



In this tutorial, our goal is to map data from two sources to one target. Therefore, click **Duplicate Input**. By doing so, you configure the target component in such a way that it will accept the data from the new source, too. The mapping now looks as follows:



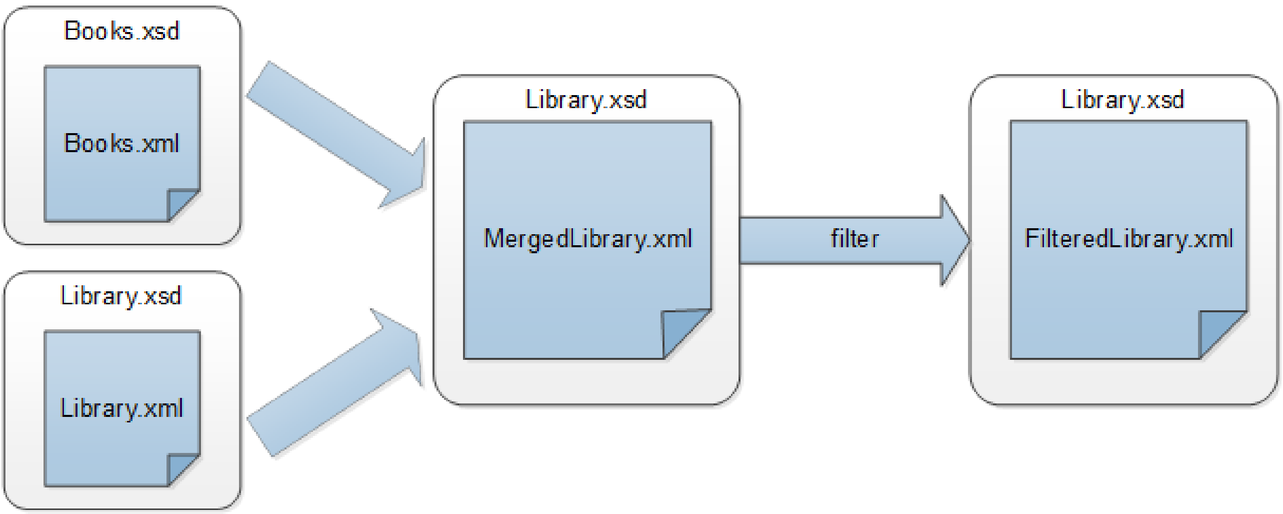
The screenshot above demonstrates that the `publication` item in the target component has been duplicated. The new `publication(2)` node will accept the data from the source component `Library`. Importantly, even though the name of this node appears as `publication(2)` in the mapping, its name in the target XML file will be `publication`, which is our goal in this case.

Click the **Output** button at the bottom of the **Mapping** pane to view the mapping result. You will notice that the data from both **`Books.xml`** and **`Library.xml`** has now been merged into the new **`MergedLibrary.xml`** file. To save the output, open the **Output** pane and select the menu command **Output | Save Output File** or click (**Save generated output**) in the toolbar.

For your convenience, the mapping design in this tutorial is saved as **`Tut2_MultipleToOne.mfd`**. This is useful because this mapping will be used as a starting point in the next tutorial 60.

2.3 Chained Mapping

The goal of this tutorial is to filter the data in **`MergedLibrary.xml`**, which was created in the [previous tutorial](53) 53, in such a way that only a subset of this data is displayed in the output. Specifically, we will only need the books published after 1900. The image below illustrates an abstract model of the data transformation described in this tutorial.

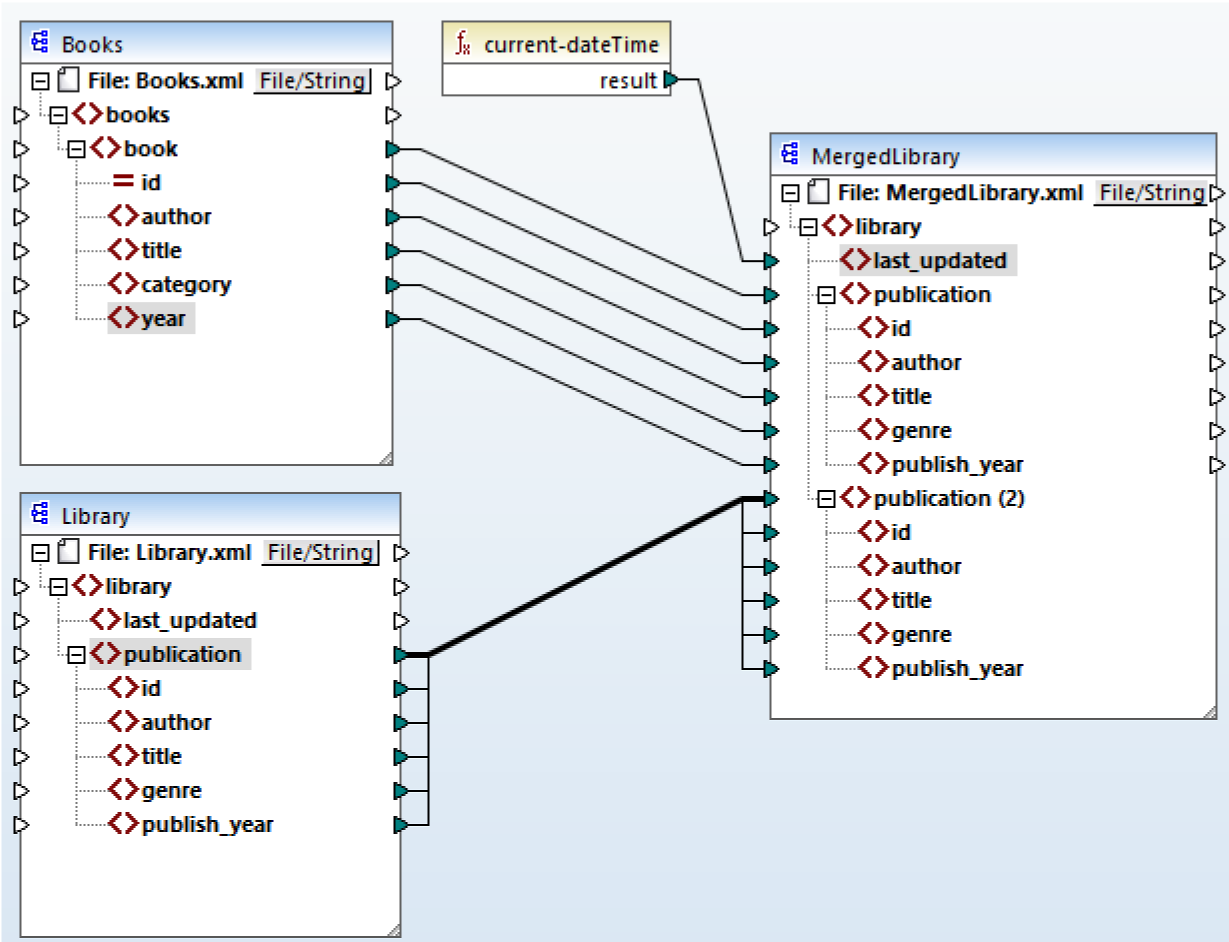


In the diagram above, the data is first merged from two source files (**`Books.xml`** and **`Library.xml`**) into a single target file called **`MergedLibrary.xml`**. Then the data is transformed with a filtering function and passed further to the next component called **`FilteredLibrary.xml`**. Note that **`FilteredLibrary.xml`** is based on the **`Library.xsd`** schema. The intermediate component acts both as a data target and source. In MapForce, this technique is known as *chained mappings* 1246.

To carry out the mapping, take the steps described in the subsections below.

2.3.1 Prepare Mapping Design

The starting point of this tutorial is **`Tut2_MultipleToOne.mfd`** (*see screenshot below*). This mapping was designed in the previous tutorial 53.

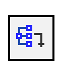


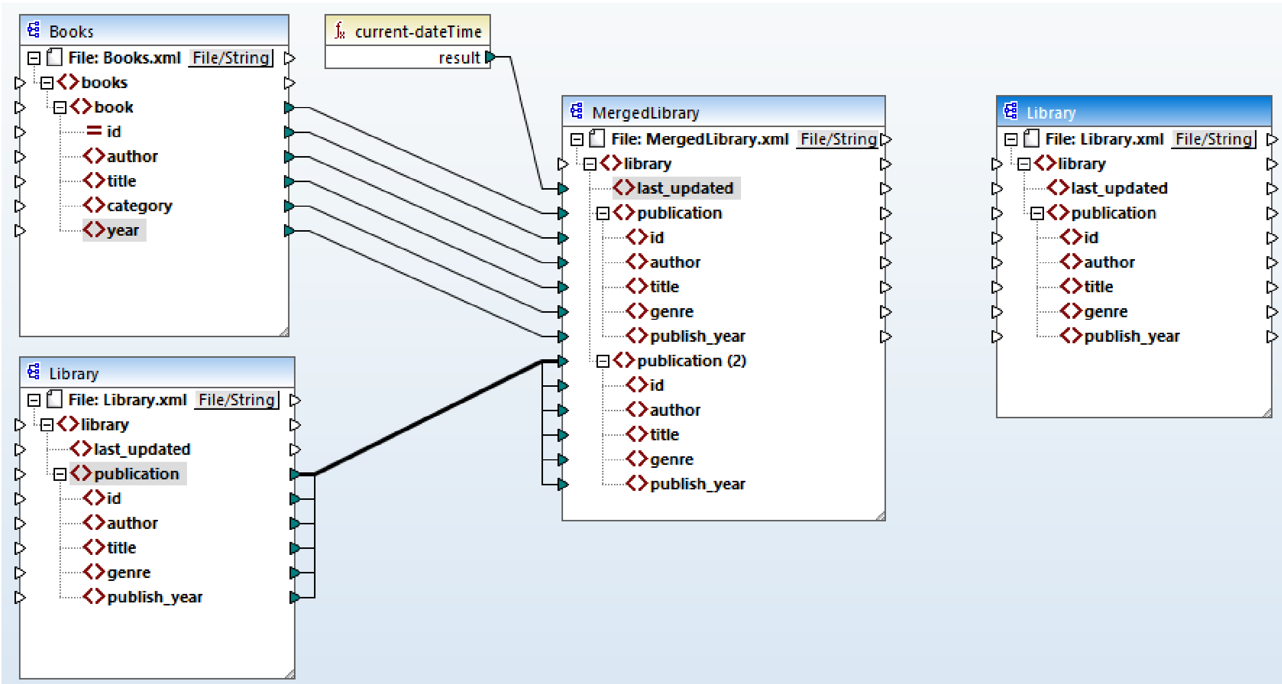
As the new mapping references several files from the same folder, make sure to save this new mapping in the **`BasicTutorials`** folder.

2.3.2 Configure Second Target

Now we need to add and configure the second target file (**`FilteredLibrary.xml`**), which will contain only a subset of <book> elements from **`MergedLibrary.xml`**.

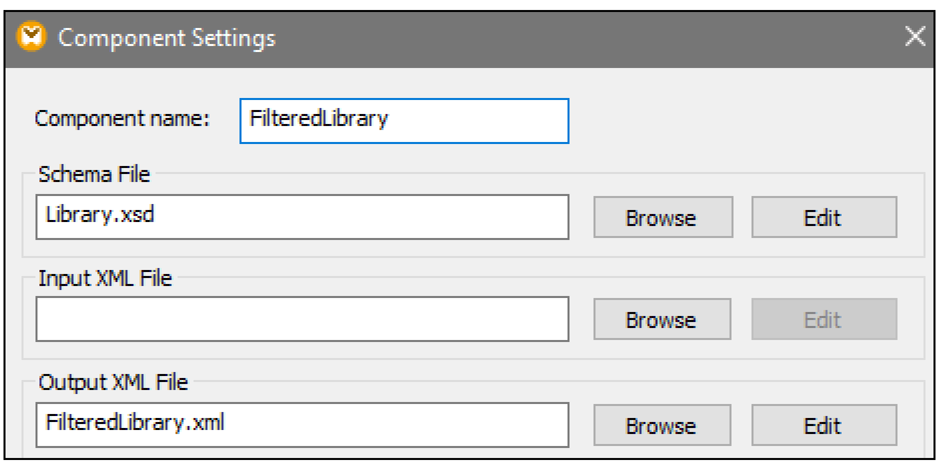
Add the second target component

To add the second target component, click the toolbar button  (**Insert XML Schema/File**) and open **`Library.xsd`**. Click **Skip** when prompted to supply a sample instance file. The second target component has only structure but no content. At a later stage, we will map the filtered data to this target file. The mapping design now looks as follows:



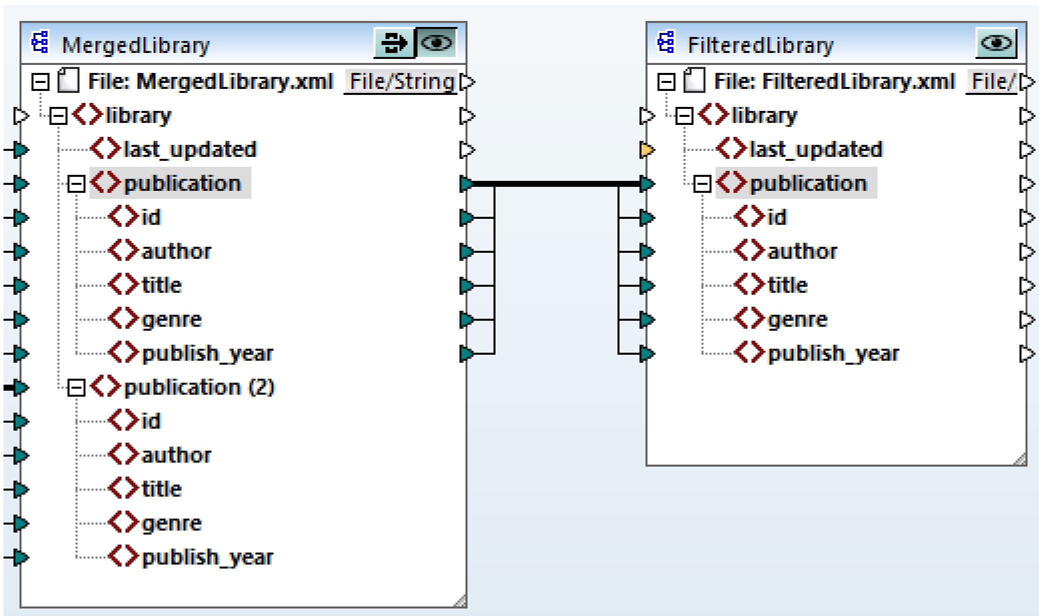
Configure the second target component

As shown above, the mapping now has two source components (`Books` and `Library`) and two target components (`MergedLibrary` and `Library`). To avoid confusion, we will rename the newly added component `FilteredLibrary`. To do this, double-click the header of the right-most component and edit the component settings (123) as follows:



2.3.3 Connect Targets

The next step is to map the `<publication>` element in `MergedLibrary` to the `<publication>` element in `FilteredLibrary`. When you connect the output connector[77] of `MergedLibrary` with the input connector of `FilteredLibrary`, the following notification message is displayed: "You have created multiple target components in the mapping project. To preview the output of a specific target component, click Preview button in the title bar of that component, then click the XSLT, or Output tab, to preview the result." Click **OK**. Notice that new buttons are now available in the upper-right corner of both target components: (**Preview**) and (**Pass-through**). These buttons will be used and explained in the next steps.

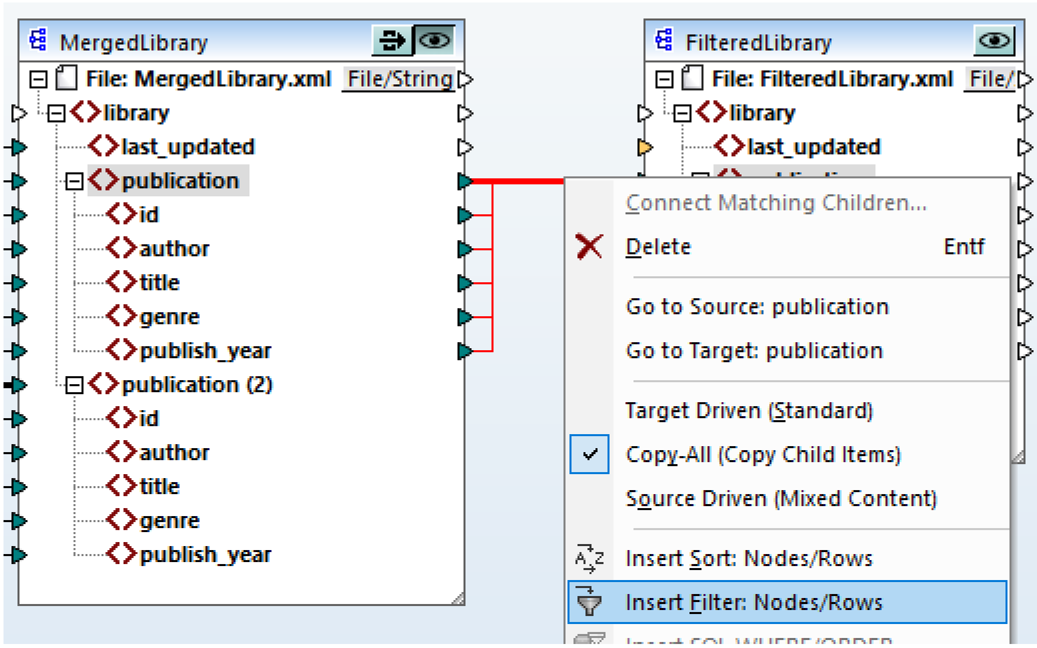


2.3.4 Filter Data

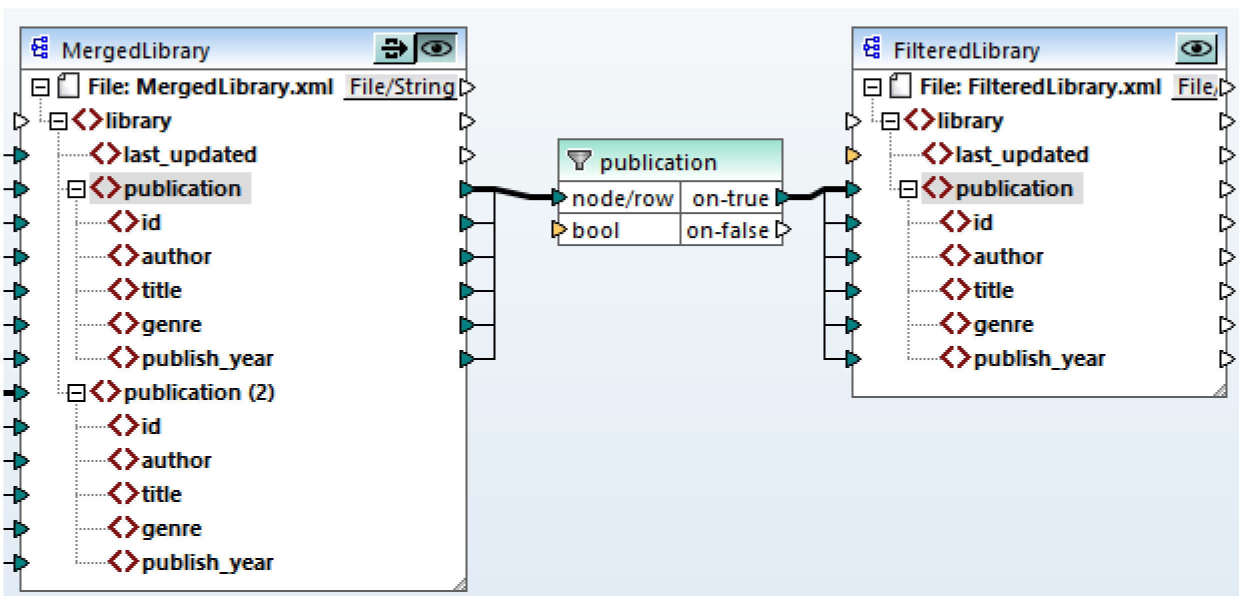
In this step, we will filter the data from `MergedLibrary` in such a way that only the books published after 1900 will be passed to the `FilteredLibrary` component. We will use a **Filter** component for this purpose.

Add a filter

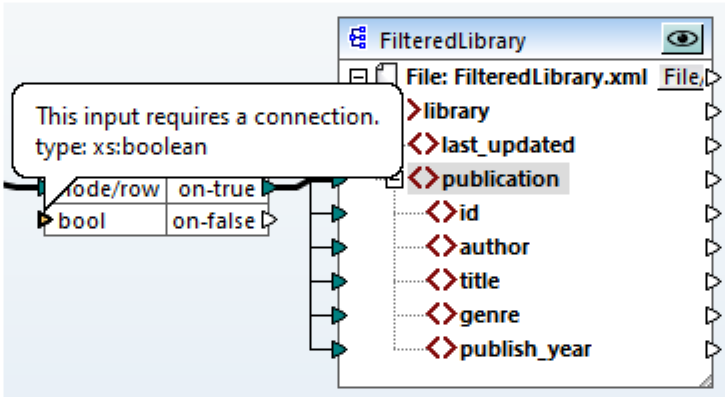
To add a filter, right-click the connection between `MergedLibrary` and `FilteredLibrary` and select **Insert Filter: Nodes/Rows** from the context menu.



The filter component has now been added to the mapping (*see screenshot below*).



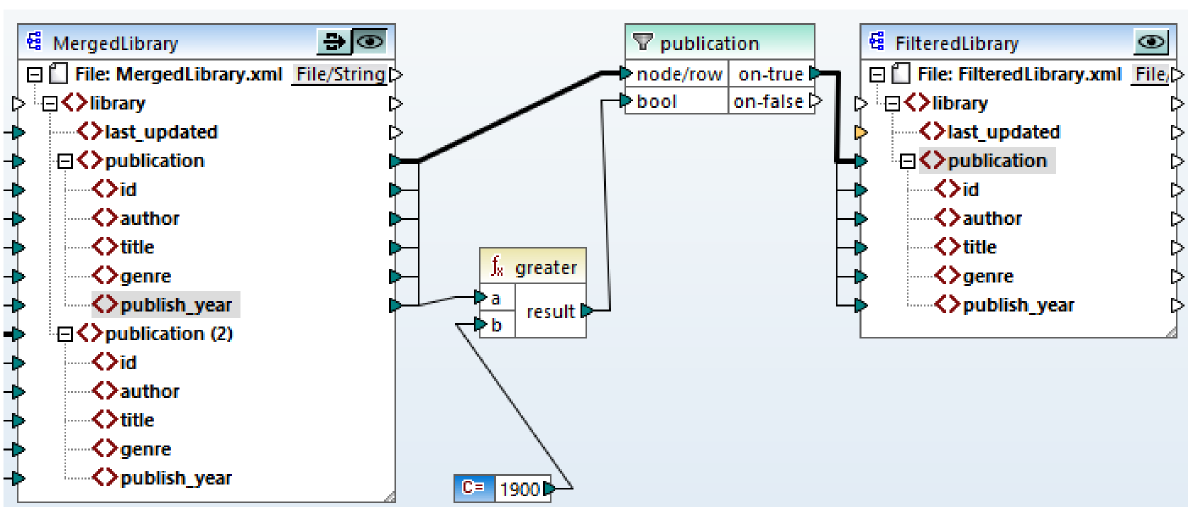
As shown above, the `bool` input connector is highlighted in orange, which means that an input is mandatory. If you hover over the connector, you will see that an input of type `xs:boolean` is required (*see screenshot below*). To see tips, click (**Show tips**) in the toolbar.



Only books after 1900

The filter component requires a condition that returns either `true` or `false`. When the Boolean condition returns `true`, the data of the current `publication` sequence will be copied over to the target. When the condition returns `false`, the data will not be copied. In this tutorial, the required condition is to filter all the books that were published after 1900. To create the condition, do the following:

1. Click **Constant** in the **Insert** menu. Add a constant with the value *1900*. Choose **Number** as a type.
2. In the **Libraries** window, locate the `greater` function and drag it to the mapping pane.
3. Make the mapping connections to and from the `greater` function, as shown below. By doing this, you are instructing MapForce to copy the current source `<publication>` element to the `<publication>` element in the target component when `publish_year` is greater than 1900.



2.3.5 Preview and Save Output

We are now ready to preview and save the output of each target component. When multiple target components exist in the same mapping, you can choose which one to preview by clicking the  button. When the

Preview button is in a pressed state, it indicates that that specific component is currently enabled for preview, and it will generate the output in the **Output** pane. Only one component at a time can have the preview enabled.

Therefore, when you want to view and save the output of the intermediate component `MergedLibrary`, do the following:

1. Click  in the `MergedLibrary` component.
2. Click the **Output** button at the bottom of the **Mapping** pane.
3. In the **Output** menu, click **Save Output File** if you want to save the output to a file.

When you want to view and save the output of the `FilteredLibrary` component:

1. Click  in the `MergedLibrary` component.
2. Click  in the `FilteredLibrary` component.
3. Click the **Output** button at the bottom of the **Mapping** pane.
4. In the **Output** menu, click **Save Output File** if you want to save the output to a file.

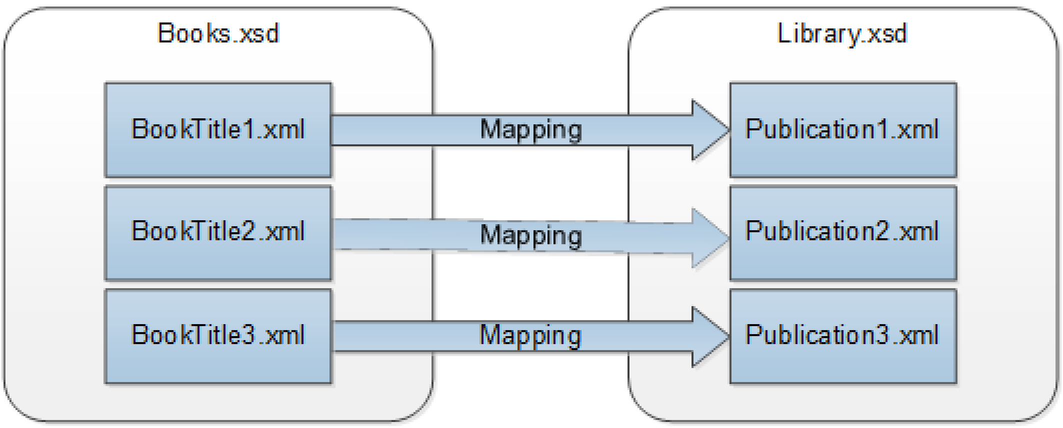
You have now finished designing the mapping which has two target components. For further information about working with pass-through components, see Chained Mappings[1246]. For your convenience, the mapping design in this tutorial is saved as **`Tut3_ChainedMapping.mfd`**.

2.4 Multiple Sources to Multiple Targets

This tutorial shows you how to map data from multiple source files to multiple target files in the same transformation. To illustrate this technique, we will create a mapping with the following goals:

1. To read data from multiple XML files in the same directory.
2. To map the schema of each file to a new schema.
3. For each source XML file, to generate a new XML target file with the new schema.
4. To strip the XML and namespace declaration from the generated files.

The image below illustrates an abstract model of the data transformation used in this tutorial:



Starting point

We will use three source XML files as examples. They are named **`BookTitle1.xml`**, **`BookTitle2.xml`**, and **`BookTitle3.xml`**. Each of the three files is based on **`Books.xsd`** and stores a single book (*see below*).

`BookTitle1.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Books.xsd">
   <book id="1">
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <category>Fiction</category>
      <year>1876</year>
   </book>
</books>
```

`BookTitle2.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Books.xsd">
   <book id="2">
      <author>Franz Kafka</author>
```

```
        <title>The Metamorphosis</title>
        <category>Fiction</category>
        <year>1912</year>
    </book>
</books>
```

BookTitle3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Books.xsd">
    <book id="3">
        <author>Herman Melville</author>
        <title>Moby Dick</title>
        <category>Fiction</category>
        <year>1851</year>
    </book>
</books>
```

Further steps

This is how are going to proceed:

1. First of all, we need to map the source schema (**Books.xsd**) to the target schema (**Library.xsd**).
2. After the transformation, the mapping will generate three files according to this new schema (*see code listings below*).
3. We will also configure the mapping so that the names of the generated files will be **Publication1.xml**, **Publication2.xml**, and **Publication3.xml**. Notice that the XML declaration and the namespace declaration must be stripped.

Publication1.xml

```
<library>
    <publication>
        <id>1</id>
        <author>Mark Twain</author>
        <title>The Adventures of Tom Sawyer</title>
        <genre>Fiction</genre>
        <publish_year>1876</publish_year>
    </publication>
</library>
```

Publication2.xml

```
<library>
    <publication>
        <id>2</id>
        <author>Franz Kafka</author>
        <title>The Metamorphosis</title>
        <genre>Fiction</genre>
        <publish_year>1912</publish_year>
    </publication>
</library>
```

Publication3.xml

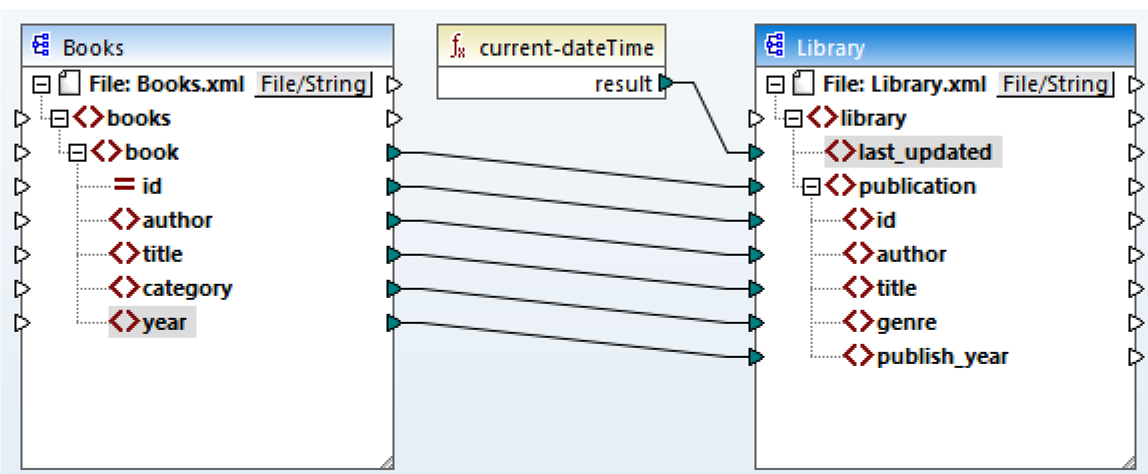
```
<library>
   <publication>
      <id>3</id>
      <author>Herman Melville</author>
      <title>Moby Dick</title>
      <genre>Fiction</genre>
      <publish_year>1851</publish_year>
   </publication>
</library>
```

To carry out the required data transformation, take the steps described in the subsections below.

2.4.1 Prepare Mapping Design

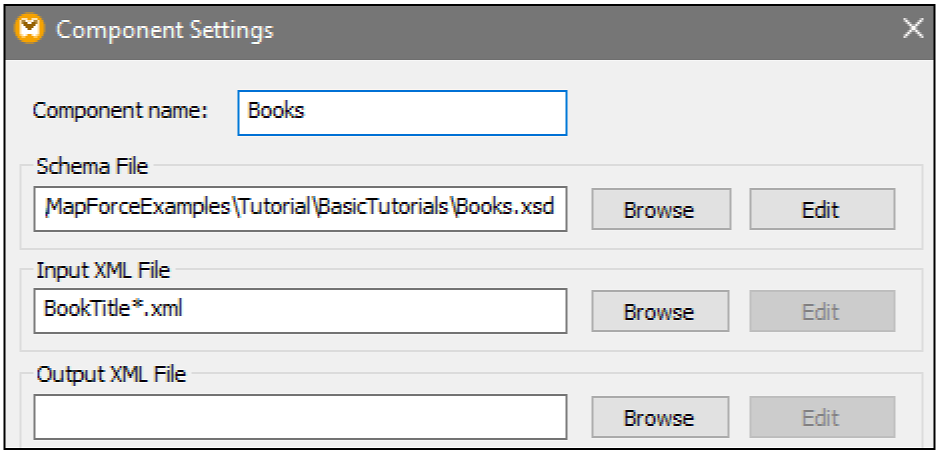
The starting point of this tutorial is similar to the **`BooksToLibrary.mfd`** mapping from the first tutorial[43], the difference being that no XML files are supplied at this stage. Our goal in this step is to map one abstract structure (**`Books.xsd`**) to another abstract structure (**`Library.xsd`**). To reproduce the scenario illustrated in the screenshot below, take the following steps:

1. Open **`Books.xsd`**.
2. When MapForce suggests adding a sample XML file, click **Skip**.
3. Open **`Library.xsd`**.
4. When MapForce suggests adding a sample XML file, click **Skip**.
5. Connect the respective nodes, as shown in the screenshot below.
6. Locate the **`current-dateTime`** function in the **Libraries** window.
7. Drag the function to the mapping area and connect `result` with the `last_updated` element.



2.4.2 Configure Input

Now we want three XML files, each containing one book, to be based on the same schema called **Books.xsd**. These XML files, called **BookTitle1.xml**, **BookTitle2.xml**, and **BookTitle3.xml**, will be used as input files. Before changing the component settings, save your mapping as an **.mfd** file in the **BasicTutorials** folder. To instruct MapForce to process multiple XML instance files, double-click the header of the source component. Enter **BookTitle*.xml** as an input file in the Component Settings dialog box 123. The asterisk (*) in the file name instructs MapForce to use all the files with the BookTitle prefix as mapping inputs. Because the path is relative, MapForce will look for all BookTitle files in the same directory as the mapping file. You can also enter an absolute path if necessary.

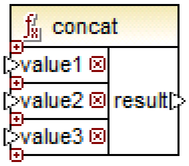


2.4.3 Configure Output Part 1

The next step is to create the file name of each output file. For this purpose, we will use the **concat** function, which concatenates (joins) all the values supplied to it as arguments. When these values are joined together, they will create an output file name (e.g., **Publication1.xml**). To generate the file names using the **concat** function, take the steps described below.

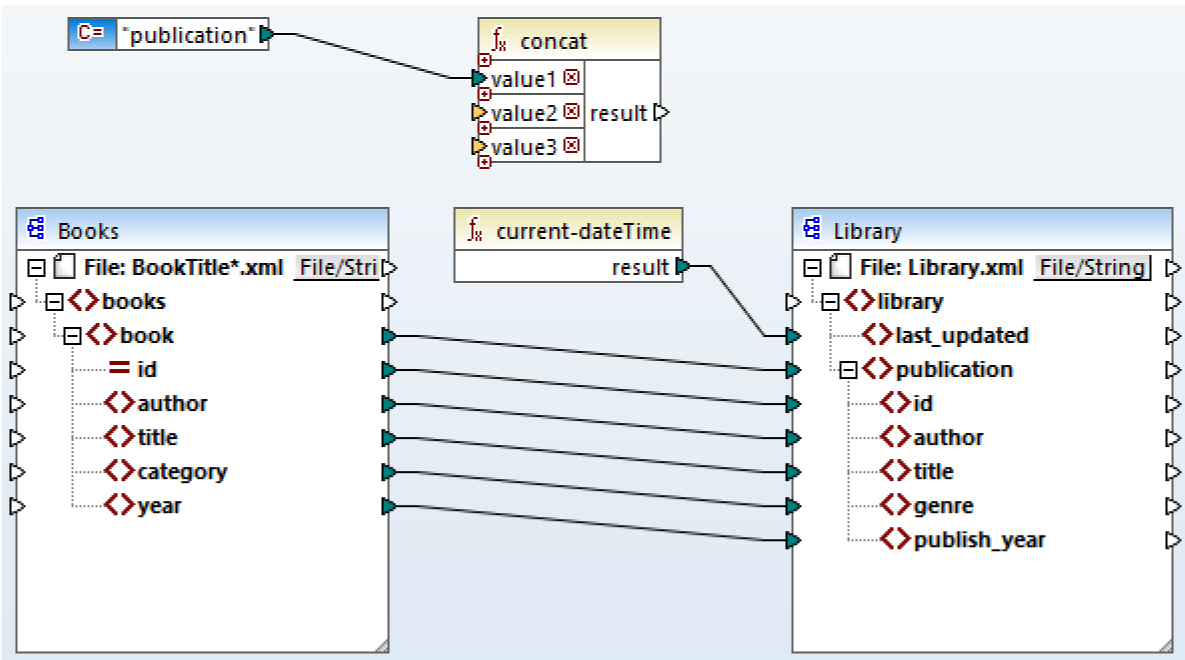
Add the concat function

Search for the **concat** function in the **Libraries** window and drag it to the mapping area (*see screenshot below*). By default, this function has two parameters when it is added to the mapping. In our example, we need three parameters. Click ⊞ (**Add parameter**) inside the function component and add a third parameter to it. Note that clicking ⊠ (**Delete parameter**) deletes a parameter.



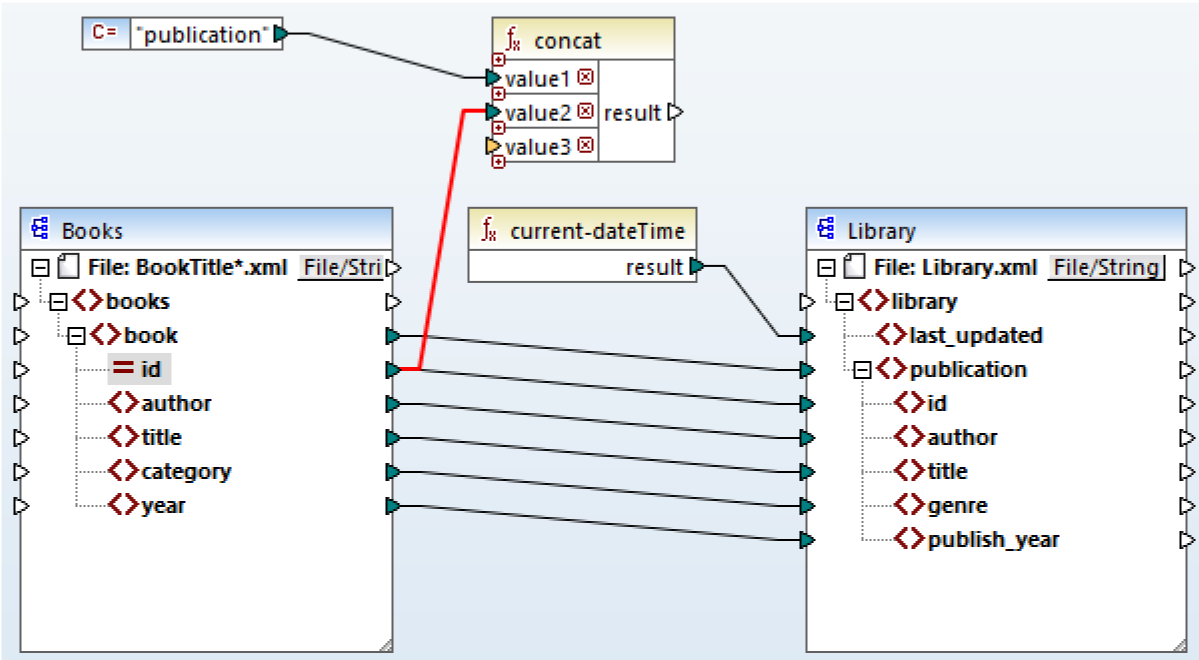
Insert a constant

To add a constant, click **Constant** in the **Insert** menu. When you are prompted to supply a value, enter `publication` and leave the **String** option unchanged. The constant `publication` supplies the constant string value `publication`. Connect the constant with `value1` of the `concat` function, as shown in the screenshot below:



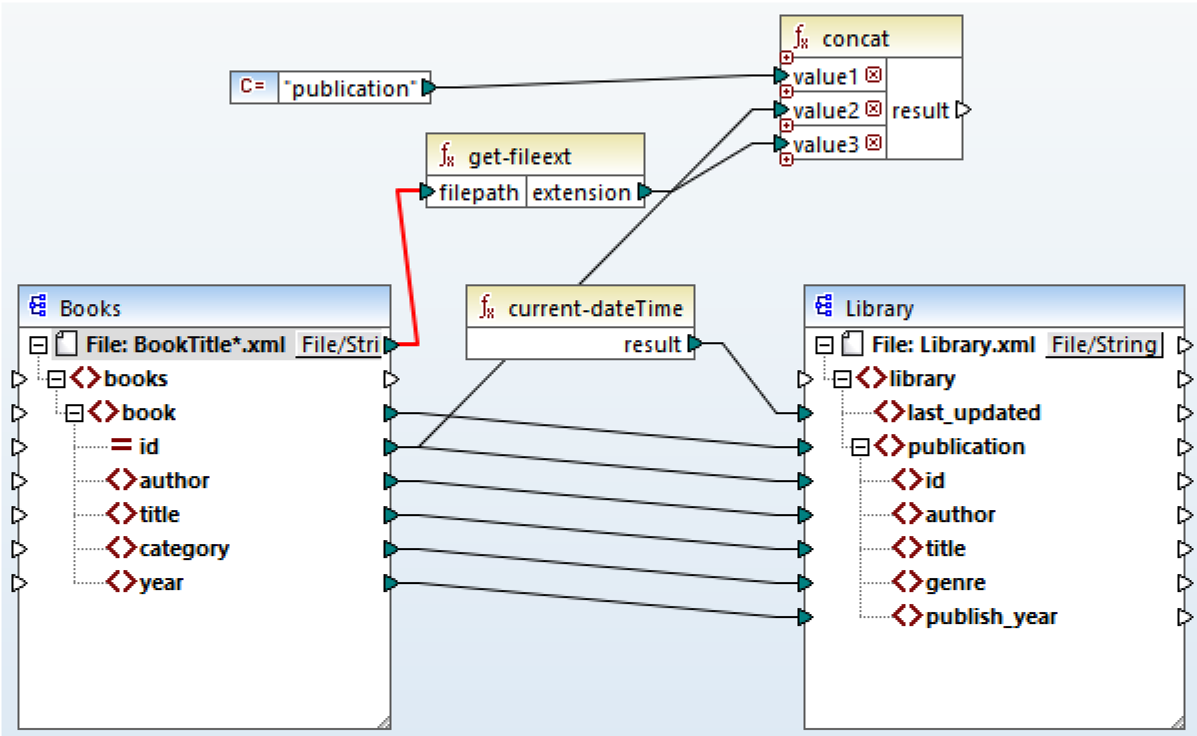
Supply the id

Connect the `id` attribute of the source component with `value2` of the `concat` function. The attribute `id` of the source XML file supplies a unique identifier value for each file. This is to prevent all files from being generated with the same name. The connection becomes red when you click on it.



Extract the file extension

Search for the **get-fileext** function in the **Libraries** window and drag it to the mapping area. Create a connection from the top node of the source component (**`File: BookTitle*.xml`**) to the `filepath` parameter of this function. Then connect the `extension` parameter of the **get-fileext** function to `value3` of the **concat** function. By doing this, you are extracting only the extension part (in this case, **`.xml`**) from the source file name and pass it to the output file name.

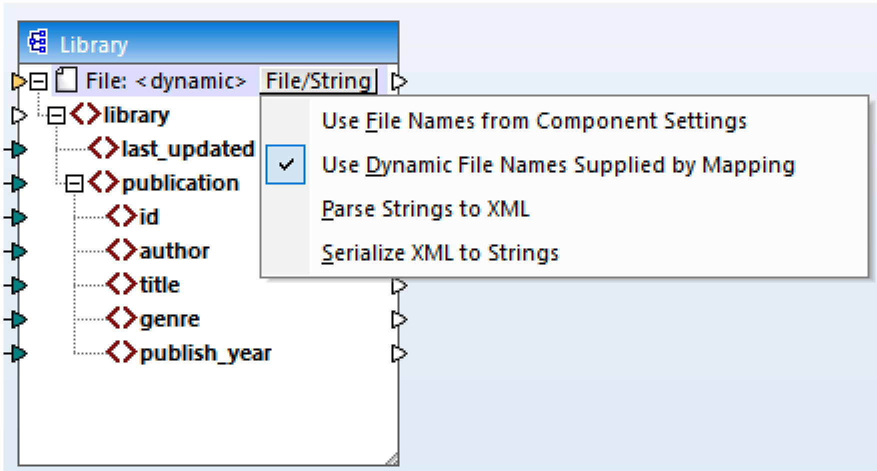


2.4.4 Configure Output Part 2

We can now instruct MapForce to create the file names when the mapping runs. In order to do this, we will use dynamic file names (*see subsections below*).

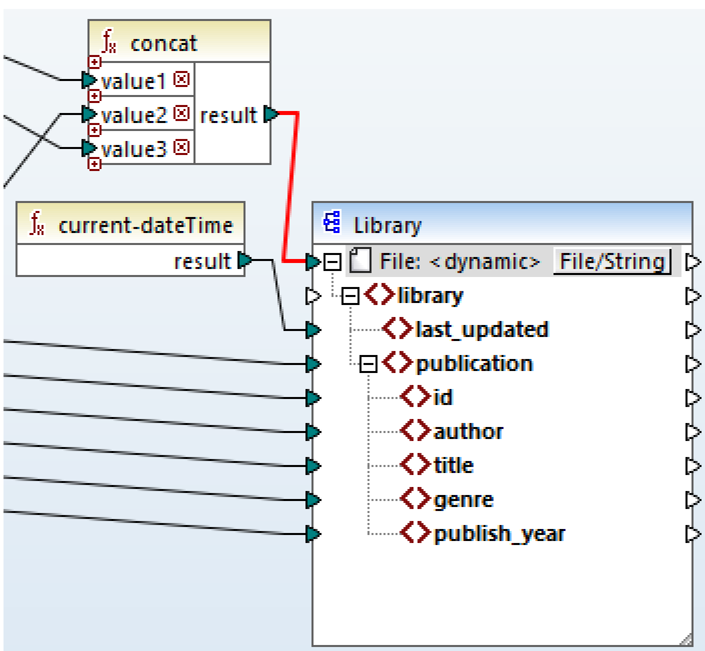
Dynamic file names

At this stage, we need to instruct MapForce to generate the instance files dynamically, which means that the every output file will receive its name based on the arguments supplied to the **concat** function. To do this, click File or File/String of the target component and select **Use Dynamic File Names Supplied by Mapping**.



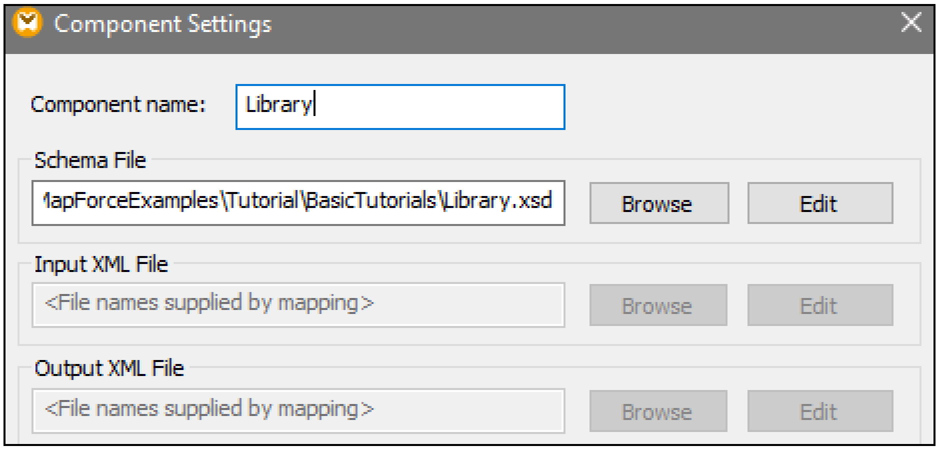
Connect the function with the dynamic node

The next step is to connect the result of the **concat** function with the `File: <dynamic>` node of the target component.

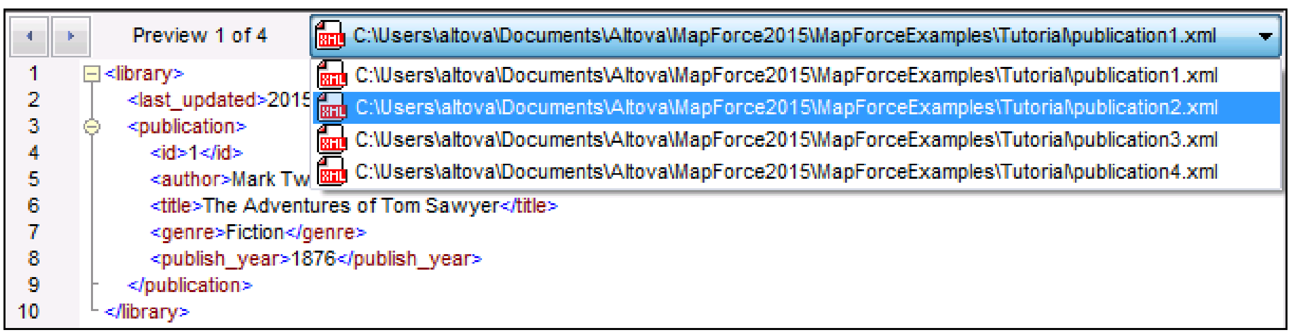


Check the component settings

If you double-click the header of the target component this time, you will notice that the **Input XML File** and **Output XML File** text boxes are disabled, and their value shows <*File names supplied by the mapping*> (*see screenshot below*). This indicates that you have supplied the instance file names dynamically from the mapping. Therefore, it is no longer relevant to define them in the component settings.



You can now run the mapping and see the result as well as the names of the generated files. This mapping generates multiple output files. You can navigate through the output files using the left and right buttons in the upper left corner of the **Output** pane or by picking a file from the adjacent drop-down list (*see screenshot below*).

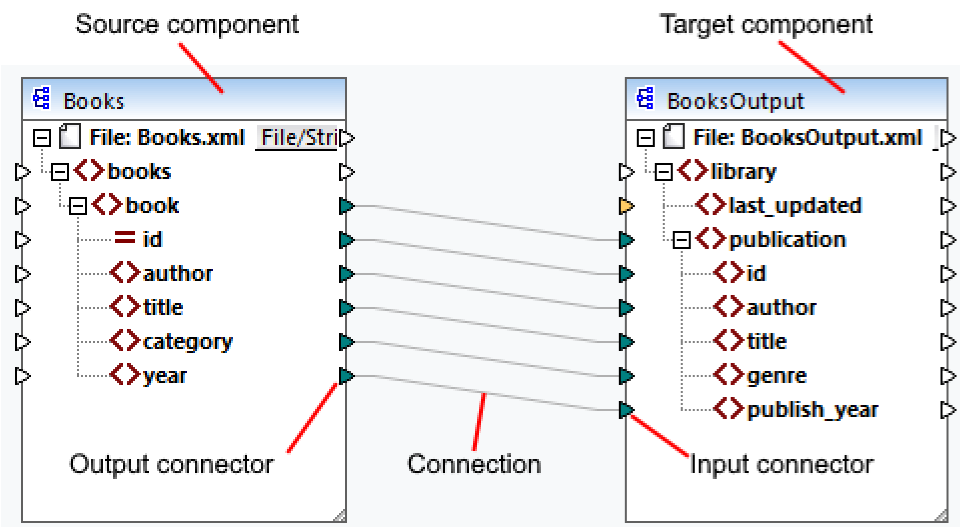


For your convenience, the mapping design in this tutorial is saved as `Tut4_MultipleToMultiple.mfd`.

3 Mapping Fundamentals

A MapForce mapping design (or simply mapping) is the visual representation of how data is to be transformed from one format into another. A mapping consists of *components* that you add to the mapping area to create your data transformations. A valid mapping consists of one or several *source components* connected to one or several *target components*. You can run a mapping and preview its result directly in MapForce. You can generate code and execute it externally. You can also compile a mapping to a MapForce execution file and automate this mapping execution using [MapForce Server] or [FlowForce Server], for example. MapForce saves mappings as `.mfd` files.

The screenshot below illustrates the basic structure of a mapping:



New mapping

To create a new mapping, click (**New**) in the toolbar. Alternatively, click **New** in the **File** menu. Then select **Mapping** and click **OK**. The next step is to add components [80] to the mapping and create connections [89].

Main parts of a mapping

The subsections below describe the main parts of a mapping design.

Component

In MapForce, the term *component* is what represents visually the structure of your data, or how data is to be transformed. Components are the central building pieces of any mapping and are represented as rectangular boxes. Components can be divided into two large groups:

- Source and target components;
- Structural [121] and transformation [832] components.

Note that these two groups are not mutually exclusive. The first group reflects the relations between components: e.g., one component can be a source for one component and a target for another component. MapForce reads data from a source component and writes this data to a target component. When you run a mapping, the target component instructs MapForce to generate a file (or multiple files) or output the result as a string value for further processing in an external program. The types of components from the first group are described below:

- A *source* is located on the left of a target component. MapForce reads data from the source.
- A *target* is located on the right of a source. MapForce writes data to the target component.
- A *pass-through* component is a subtype of source and target components. A pass-through component acts both as a source and target. For more information, see Chained Mappings 1246. Note that only structural components can be pass-through.

The second group shows whether a component has a data structure or is used to transform data mapped from another component.

To find out more about components and component-related actions, see Components 78.

Connector
A connector is a small triangle located on the left or right side of a component. Input connectors are on the left of a component and show data entry points *to that component*. Output connectors are on the right of a component and show data exit points *from that component*.

Connection
A connection is a line that you can draw between two connectors. When you create connections, you instruct MapForce to transform data in a specific way: for example, to read data from an XML document and write it to another XML document.

In this section

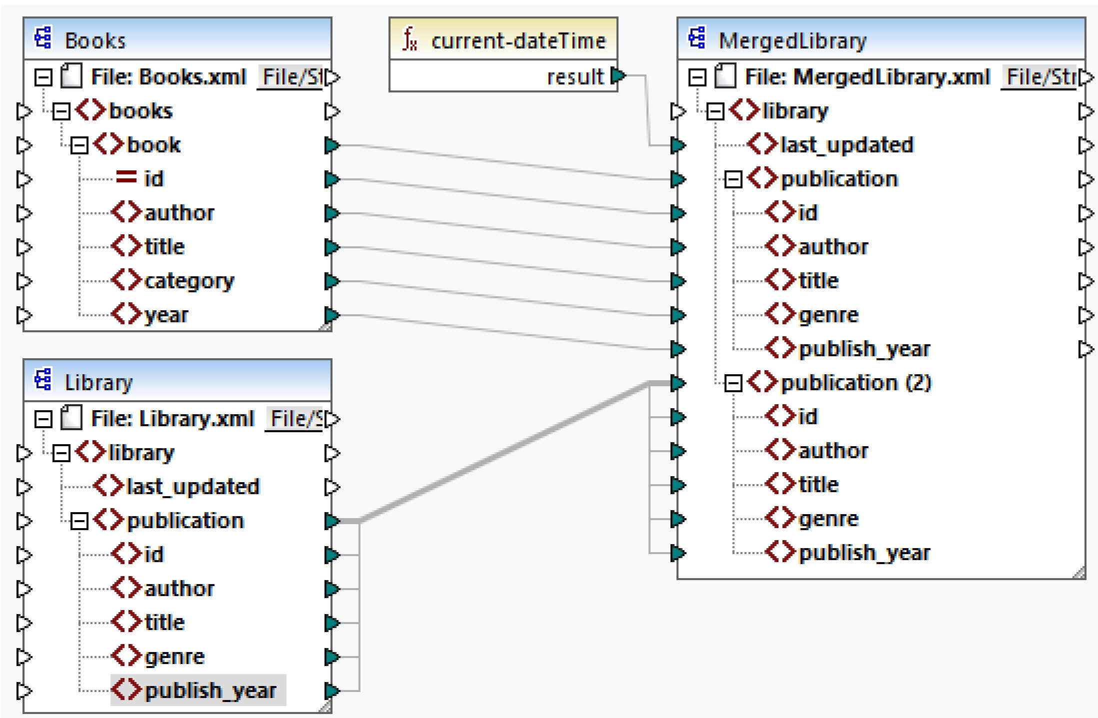
This section describes the most common MapForce tasks and concepts. The section is organized into the following subsections:

3.1 Components

Components are the central elements of any mapping design in MapForce. Visually, components are represented as rectangular boxes in the mapping area. This topic gives an overview of structural and transformation components (*see example below*). The distinction is based on whether a component has a data structure or is used to transform data. See the description of these types in the subsections below. See also Mapping Fundamentals 76.

Components example

The sample mapping below illustrates two data source components (`Books` and `Library`), one data target component (`MergedLibrary`), and one transformation component (the **`current-dateTime`** function).



Structural components

Structural components represent an abstract structure of your data (e.g., an XML file). The list of structural components that can be used as data sources and targets is given in Structural Components 121. Structural components can read data from files or other sources, write data to files or other sources, and store data at some intermediary stage in the mapping process (e.g., to preview the data). The table below gives an overview of structural components and their respective toolbar buttons.

Icon	Description
	XML component
	Text component (*Professional and Enterprise editions*)

Icon	Description
	Database component (*Professional and Enterprise editions* for SQL Databases; *Enterprise Edition* for NoSQL databases)
	JSON component (*Enterprise Edition*)
	Microsoft Excel component (*Enterprise Edition*)
	WSDL component (*Enterprise Edition*)
	EDI component (*Enterprise Edition*)
	XBRL component (*Enterprise Edition*)
	Protocol Buffers (*Enterprise Edition*)

Transformation components

Transformation components help you transform data [904], store an intermediate mapping result [847] for further processing, replace a value by another value [884], sort [872], group [894], join [860], and filter [878] your data. You can also add an exception [900], which stops the mapping process and displays an error when a condition defined by a filter occurs. The table below gives an overview of transformation components and their respective toolbar buttons.

Icon	Description
	Simple input
	Simple output
	Filter component
	Sort component
	Built-in function
	User-defined function
	SQL/NoSQL-WHERE/ORDER component (*Professional and Enterprise editions*)
	Value-Map component
	Variable
	Web service function (*Enterprise Edition*)
	Exception (*Professional and Enterprise editions*)
	Constant
	If-Else Condition

Icon	Description
	Join component (*Professional and Enterprise editions*)

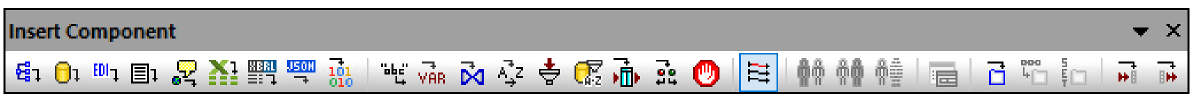
In this section

This section gives an overview of components and is organized into the following topics:

3.1.1 Add Components

This topic explains how to add components to a mapping. To add a component, you need to create a new mapping design 76 first and then do one of the following:

- In the **Insert** menu, choose a component type (e.g., **XML Schema/File**).
- Drag a file from Windows File Explorer into the mapping area. Note that this operation is only possible for compatible file-based components.
- Click the relevant button in the **Insert Component** toolbar (*see screenshot below*).



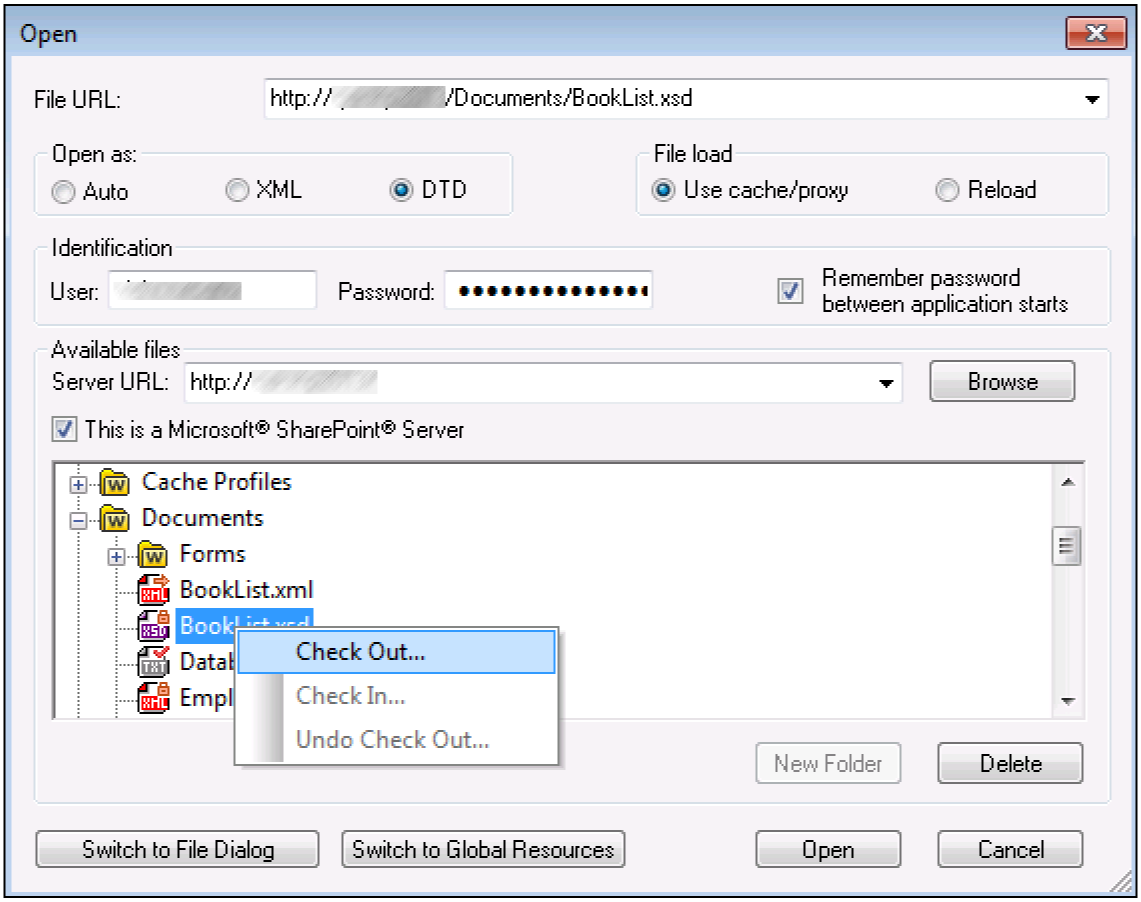
Each component type has a specific purpose and function. To get an overview of components, see Components 78. To find out more about data structures that can be used as sources and targets, see Structural Components 121. For information about MapForce built-in components used to store data temporarily or to transform it, see Transformation Components 832.

For a complete list of components, see the component icon reference below.

Add components from URL

Besides adding local files as mapping components, you can also add files from a URL. Note that this operation is supported when you add a file as a source component 76. The supported protocols are HTTP, HTTPS, and FTP. To add a component from a URL, take the following steps:

1. In the **Insert** menu, select the component type you wish to add (e.g., **XML Schema/File**).
2. In the **Open** dialog box (*screenshot below*), click **Switch to URL**.
3. Enter the URL of the file in the **File URL** text box and click **Open**. Make sure that the file type in the **File URL** text box is the same as the file type you specified in Step 1.



The list below describes the available options in the **Open** dialog.

- *Remember the password:* If the server requires password authentication, you will be prompted to enter the user name and password. If you want your user name and password to be remembered next time, start MapForce, enter your user name and password in the **Open** dialog box and select the check box **Remember password between application starts**.

- *Open as:* Defines the grammar for the parser. The default and recommended option is **Auto**.

- *Use cache/proxy:* If the file you are loading is not likely to change, select the **Use cache/proxy** option to cache the data and speed up file loading. Otherwise, select **Reload** if you want the file to be reloaded every time you open the mapping.

- *Server URL:* For servers with Web Distributed Authoring and Versioning (WebDAV) support, you can browse files after entering the server URL in the **Server URL** text box and clicking **Browse**. Although the preview shows all file types, make sure to choose to open the same file type as in Step 1 above. Otherwise, errors will occur.

- *Check in/out:* If you use a Microsoft SharePoint Server, select the check box **This is a Microsoft SharePoint Server**. This displays the check-in or check-out state of the file in the preview area. If you want to make sure that no one else can edit the file on the server while you are using it, right-click the file and select **Check Out** (*see screenshot above*). To check in any file that you previously checked out, right-click the file and select **Check In**.

3.1.2 Component Basics

This topic explains how to set, search and manipulate structural components [78]. For more information, see the subsections below.

Change component settings

After you add a component to the mapping area, you can configure its settings in the **Component Settings** dialog box. You can open the **Component Settings** dialog box in one of the following ways:

- Double-click the component header.
- Select the component and click **Properties** in the **Component** menu.
- Right-click the component header and click **Properties**.

Note that the available options depend on the type of a component. The list of settings for each component type is given below:

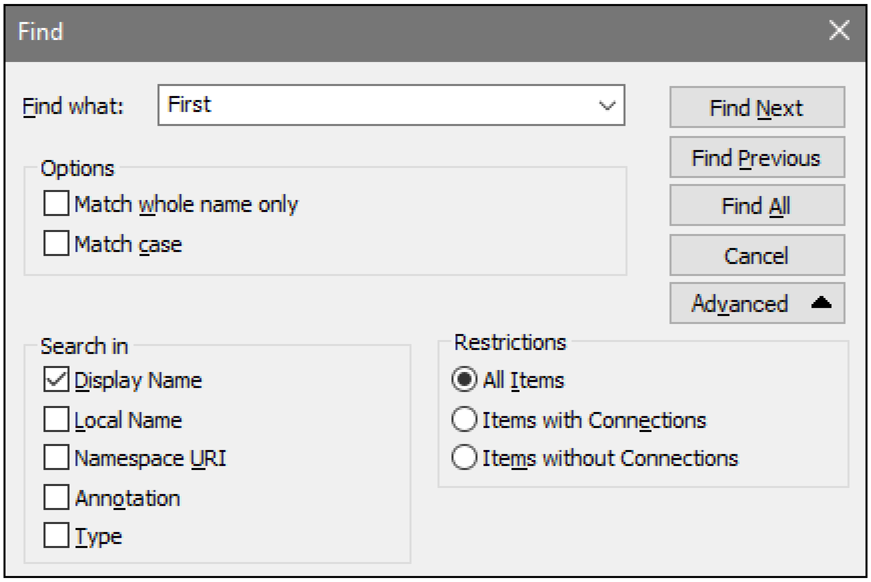
- XML Component Settings [123]
- Database Component Settings [279]
- CSV Component Settings [572]
- Fixed-Length Field Component Settings [580]
- FlexText Component Settings [673]
- JSON Component Settings [796]
- Excel 2007+ Component Settings [729]
- XBRL Component Settings [767]
- EDI Component Settings [450]
- Protocol Buffers Component Settings [815]

For any file-based component (e.g., an XML file) the File (*Basic Edition*) or File/String (*Professional and Enterprise editions*) button appears next to the root node. This button specifies advanced options for processing or generating multiple files in a single mapping. For more information, see Processing Multiple Input or Output Files [1313]. This button also allows setting options for parsing strings and serializing data to strings [1324].

Search a component

To search for a specific node in a component, take the following steps:

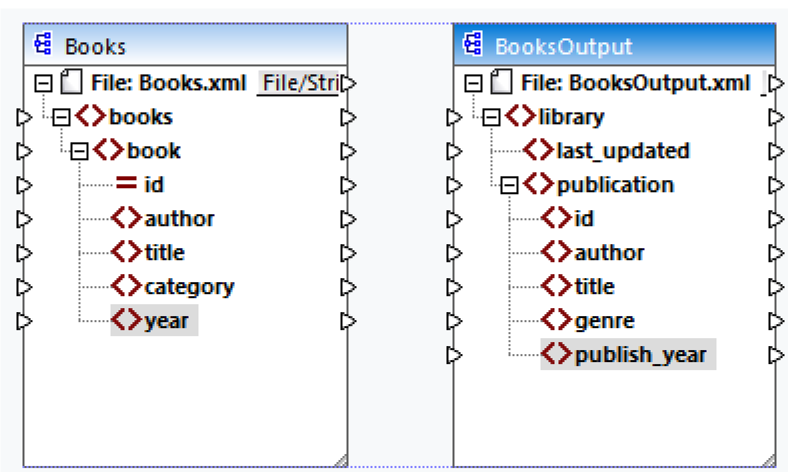
1. Click the component you want to search and press **CTRL+F**.
2. Enter a search term and click **Find Next/Previous/All** (*see screenshot below*).



Use the **Advanced** options to define which items (nodes) are to be searched. You can also restrict the search options based on specific connections.

Align components

When you move components in the mapping area, MapForce shows guide lines (dotted lines) that help you align components (*see screenshot below*).



To enable this option, take the steps below:

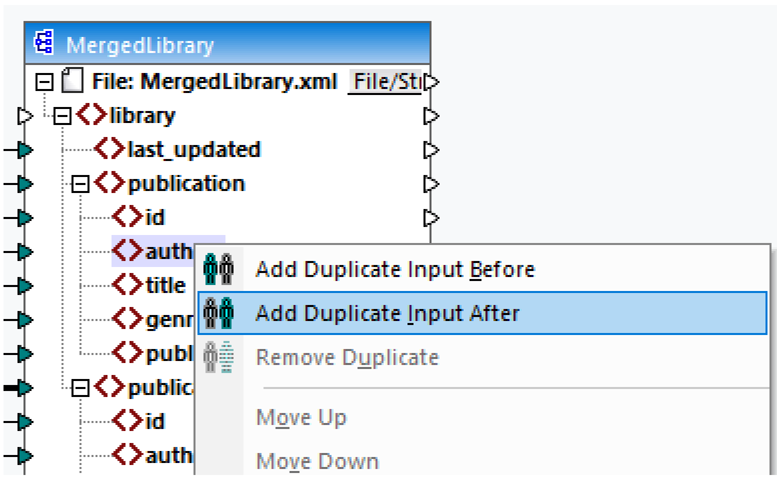
1. Go to the **Tools** menu and click **Options**.
2. In the **Editing** group, select the check box **Align components on mouse dragging**.

Duplicate input

Sometimes, you may need to configure a component so that it can accept data from more than one source. If you want the target schema to accept data from more than one source schema, you can duplicate any input nodes in your target component. Duplicating input is meaningful only for a target component: Duplicated nodes can only accept data, but it is not possible to map data from duplicated nodes. You can duplicate as many nodes as you need.

There are two ways of duplicating input: (i) selecting **Add Duplicate Input Before/After** from the context menu and (ii) connecting a source node with a target node that is already connected to a different node. The first option is described below. Information about the second option can be found in the second tutorial 53.

Add Duplicate Input Before/After
To duplicate a particular input node, right-click it and select **Add Duplicate Input Before/After** from the context menu (*see screenshot below*). In the image below, the `author` node is about to be duplicated so that data can be mapped to the duplicated node from another source node.



Note: Duplication of XML attributes is not allowed, as it will make the XML instance invalid.

3.1.3 File Paths

A mapping design (**`*.mfd`**) may have references to several schema and instance files. MapForce uses the schema files to determine the structure of the data to be mapped. In MapForce Professional and Enterprise editions, mappings can also include references to StyleVision Power Stylesheet (**`*.sps`**) files, which are used to format data for outputs such as PDF, HTML and Word. Mappings can also have references to file-based databases such as Microsoft Access or SQLite.

References to files are created by MapForce when you add a component to the mapping. However, you can always set or change such path references manually if required.

This section provides instructions on how to set or change paths to different file types referenced by a mapping. The section is organized into the following topics:

3.1.3.1 Relative and Absolute Paths

This topic explains how to use absolute and relative paths of the files referenced by your component. An absolute path shows the full location of a file, starting with the root directory (*see Example: XML Component below*). A relative path shows the file location which is relative to the current working directory: e.g., **`Books.xml`**.

In the **Component Settings** dialog box (*see example below*), you can specify absolute or relative paths for various files which can be referenced by the component. The list of these files is given below:

- Input files from which MapForce reads data;
- Output files to which MapForce writes data;
- Schema files, which are applicable to components with a schema;
- Structural files, which are used as input or output parameters of user-defined functions and variables;
- StyleVision Power Stylesheet (**`*.sps`**) files, which are used to format data for outputs such as PDF, HTML and Word;
- Database files in the case of database components (*Professional and Enterprise editions*).

Copy-paste and relative paths
When you copy a component from a mapping and paste it into another mapping, MapForce checks whether the relative paths of schema files can be resolved against the folder of the destination mapping. If the paths cannot be resolved, you will be prompted to make the relative paths absolute.

Broken paths
When you add or change a file reference in a mapping, and the path cannot be resolved, MapForce displays a warning message. Broken path references may happen in the following cases:

- You use relative paths and then move the mapping file to a new directory without moving the schema and instance files.
- You use absolute paths to files in the same directory as the mapping file and then move the directory to another location.

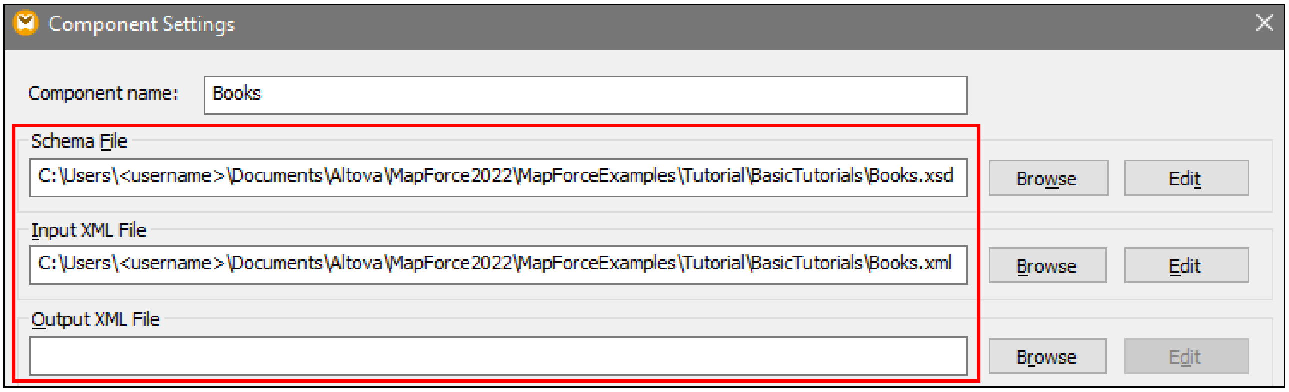
When one of these scenarios happens, MapForce highlights the component in red. The solution is to double-click the red component header and update any broken path references in the **Component Settings** dialog box. See also Change Component Settings 82.

Example: XML component

The example below shows how file paths are used in an XML component. If you want to save all the mapping-related files relative to the mapping file (**`.mfd`**), check the box *Save all file paths relative to MFD file* at the bottom of the **Component Settings** dialog box. This is the default and recommended option that affects all the files referenced by the component (*shown in the red frame in the image below*). If you have not saved your mapping yet, you will see absolute paths to the schema and/or instance files in the **Component Settings** dialog box. To see relative paths in the **Component Settings** dialog box, take the following steps:

1. Create a new mapping 76 and add a structural component 80: e.g., an XML file with an XML schema assigned to it.
2. Double-click the header of the component to open the **Component Settings** dialog box.
3. Check the box *Save all file paths relative to MFD file* at the bottom of the **Component Settings** dialog box.
4. Save your mapping.
5. You can now open **Component Settings** again to check the relative paths in the relevant text fields.

Note: Paths that reference a non-local drive or use a URL will not be made relative.



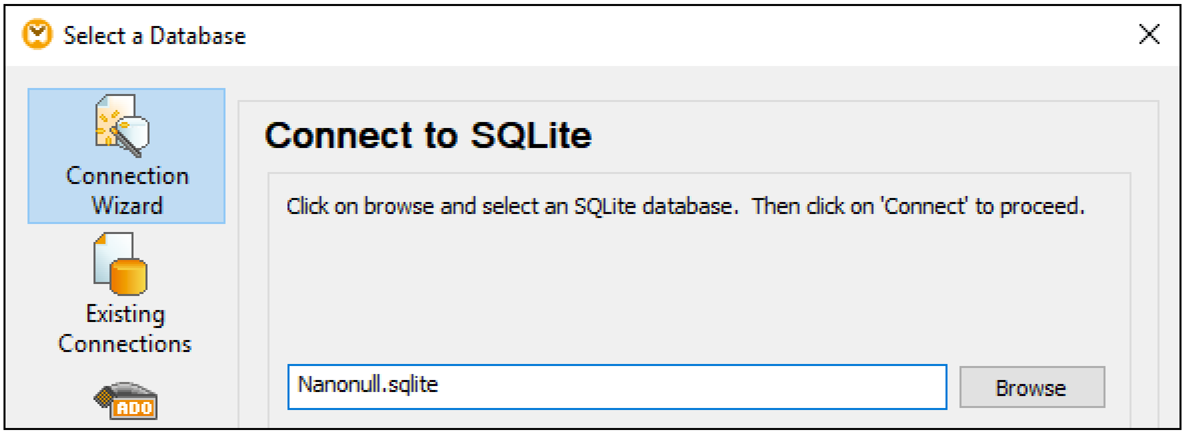
When the check box *Save all file paths relative to MFD file* is selected, MapForce will keep track of the files referenced by the component even when you save the mapping to a new folder. If all the files are in the same directory as the mapping, the path references will not be broken when you move the entire directory to a new location on the disk.

The setting *Save all file paths relative to MFD file* applies to the following files:

- Structural files used by complex input or output parameters of user-defined functions and variables of complex type;
- Input or output flat files (*Professional and Enterprise editions*);
- Schema files referenced by database components which support XML fields (*Professional and Enterprise editions*);
- Database files (*Professional and Enterprise editions*);
- Input or output XBRL, FlexText, EDI, Excel 2007+, JSON files (*Enterprise Edition only*).

Example: Database component (Professional and Enterprise editions)

When you add a database file such as Microsoft Access or SQLite to the mapping, you can enter a relative path instead of an absolute one in the **Select a Database** dialog box (*see screenshot below*). Before entering relative file paths, make sure to save the `.mfd` file first. If you want to change the path of a database component which is already in the mapping, click **Change** in the **Component Settings** dialog box. For more information about connecting to a database source, see Start Database Connection Wizard[165].



Note: When you generate program code, compile MapForce Server execution files (**`.mfx`**), or deploy the mapping to FlowForce Server, a relative path will be converted to an absolute path if you select the check box *Make paths absolute in generated code* in the mapping settings [114]. To find out more, see About Paths in Generated Code [87].

3.1.3.2 Paths in Execution Environments

If you generate code from mappings, compile mappings to MapForce Server execution files (**`.mfx`**), or deploy mappings to FlowForce Server, the generated files are run by the target environment you have chosen: for example, RaptorXML Server, MapForce Server, or a C# application. For the mapping to run successfully, any relative paths must be meaningful in the environment where the mapping runs. The base paths for each target language are given below:

Target language	**Base path**
XSLT, XSLT2, XSLT3	Path of the XSLT file.
XQuery*	Path of the XQuery file.
C++, C#, Java*	Working directory of the generated application.
Built-In* (when previewing the mapping in MapForce)	Path of the mapping file (**`.mfd`**).
Built-In* (when running the mapping with MapForce Server)	The current working directory.
Built-In* (when running the mapping with MapForce Server under FlowForce Server control)	The working directory of the job or the working directory of FlowForce Server.

** Languages available in MapForce Professional and Enterprise editions*

Relative path to absolute path
When you generate program code, compile MapForce Server execution files (**`.mfx`**), or deploy the mapping to FlowForce Server, a relative path will be converted to an absolute path if you select the check box **Make paths absolute in generated code** in the mapping settings [114].

When you generate code and the check box is selected, MapForce resolves any relative paths based on the directory of the **.mfd** and makes them absolute in the generated code. This setting affects the paths of the following files:

- Input and output instance files for all file-based components;
- Access and SQLite database files used as mapping components (*Professional and Enterprise editions*).

Library paths in generated code

Mapping files may optionally contain path references to different libraries. For example, you can import user-defined functions from another mapping file, from custom XSLT, XQuery*, C#* or Java* libraries, or from **.mff*** (MapForce Function) files. For more information, see Managing Function Libraries 908.

** Features available in MapForce Professional and Enterprise editions*

The option **Make paths absolute in generated code** applies only to mapping components and does not affect paths to external libraries. For all libraries other than XSLT and XQuery, the library path will be converted to an absolute path in generated code. For example, if your mapping file contains library references such as .NET **.dll** or Java **.class** files, and if you want to run the generated code in some other environment, the referenced libraries must exist at the same path in the target environment.

If you plan to generate an XSLT or XQuery file from a mapping, make the library path relative to the generated XSLT or XQuery file:

1. Open the mapping settings 114.
2. Select the check box **Reference libraries with paths relative to generated XSLT/XQuery file**. Make sure that the XSLT or XQuery library file exists at that path.

3.2 Connections

A connection is a line that connects a source to a target. Connections represent visually how data is mapped from one node to another. The subsections below describe different connection-related actions you can perform.

Create, copy, delete a connection

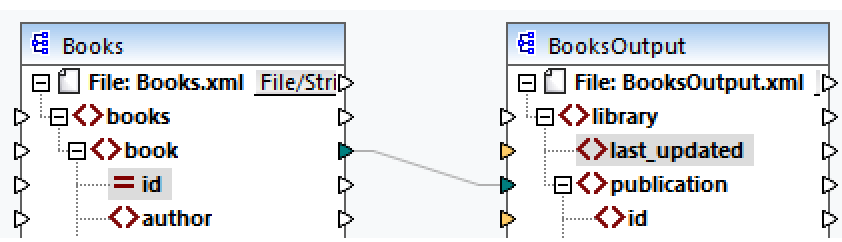
To create a connection between two items, press and hold the output connector 77 of a source node and drag it to a destination node. An input connector accepts *only one* connection. If you try to add a second connection to the same input, MapForce will ask if you want to replace the connection with a new one or duplicate the input item 83. An output connector can have several connections, each to a different input.

To copy a connection to a different item, press and hold the thick section at the end of the connection (*see screenshot in Move a connection*) and drag it to the selected destination while holding the **Ctrl** key.

To delete a connection, click the connection and press the **Delete** key. Alternatively, right-click the connection and click **Delete** in the context menu.

Mandatory inputs

To help you in the mapping process, MapForce highlights mandatory inputs in orange in target components. The example below shows that as soon as you connect the `book` element of the `Books` component to the `publication` element of the `BookOutput` component, the connectors of the mandatory nodes of the `BooksOutput` component will be highlighted. If you do not connect mandatory inputs, the respective nodes will not be mapped to the target, and the mapping will be invalid.



Missing parent connections

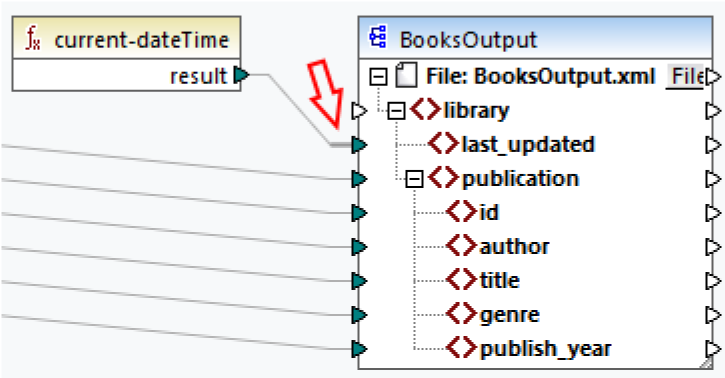
When you create connections between source and target node manually, MapForce analyzes possible mapping outcomes. If you connect two child nodes without connecting their parent nodes, you will see a notification message that suggests connecting the parent of the source node with the parent of the target node. This notification message helps you prevent situations where a single child node appears in the **Output** pane.

If you want to disable such notifications, take the following steps:

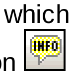
1. Go to the **Tools** menu and click **Options**.
2. Open the **Messages** group.
3. Clear the check box **When creating a connection, suggest connecting ancestor items**.

Move a connection

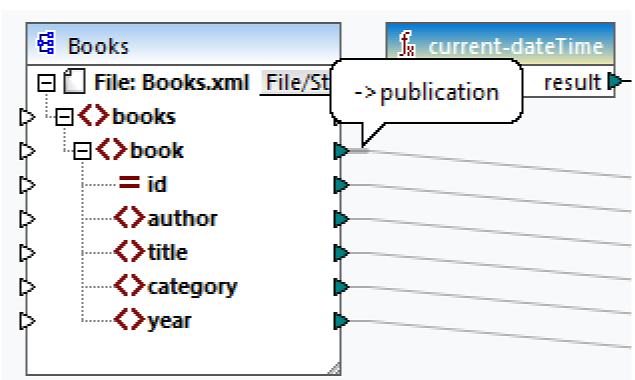
To move a connection to a different node, press and hold the thick section at the end of the connection (*see screenshot below*) and drag it to the selected destination.



See connection tooltips

Connection tooltips allow you to see the names of (i) nodes to which data is mapped or (ii) nodes from which data is mapped. To be able to see tips, press the toolbar button  (**Show tips**). To see the names of nodes to which data is mapped, point the cursor at the thick section of a connection near the output connector (*see screenshot below*). To see the name of a node from which data is mapped, point the cursor at the thick section of a connection near the input connector. If multiple connections have been defined from the same output, maximum ten item names will be displayed in the tooltip.

In the example below, the target node to which the data from the `book` element is mapped is called `publication`.



Change connection settings

To change the connection settings, do one of the following:

- Select a connection. Then go to the **Connection** menu and click **Properties**.
- Double-click the connection.
- Right-click the connection and click **Properties**.

For more information, see Connection Settings 98.

Highlight connections selectively

MapForce allows you to selectively highlight connections in a mapping. This feature may be useful when your mapping has many components with multiple connections. Highlighting connections selectively will make it easier to check whether the nodes of the selected component are mapped correctly. Note that the term *connector* used for the toolbar buttons below refers to a connection, i.e., a line connecting component nodes. See the available options below.

	Show selected component connectors (only direct connections)
	Show connectors from source to target (direct and indirect connections)

<u>Only direct connections</u>

When the **Direct-connections** button is *not* pressed, you can see all connections in black. When the **Direct-connections** button is pressed, only connections related to the currently selected component are black. Other connections are light gray.

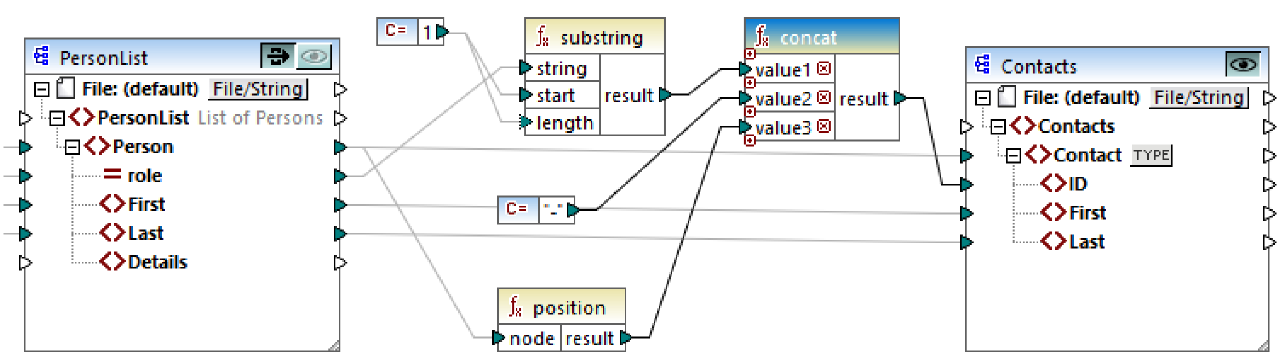
<u>Direct and indirect connections</u>

The **Source-to-target Connections** button becomes available only when the **Direct-connections** button is pressed. When the **Source-to-target Connections** button is pressed, you can trace connections of the currently selected component, including its direct connections and connections of its connected components up to the source and target files.

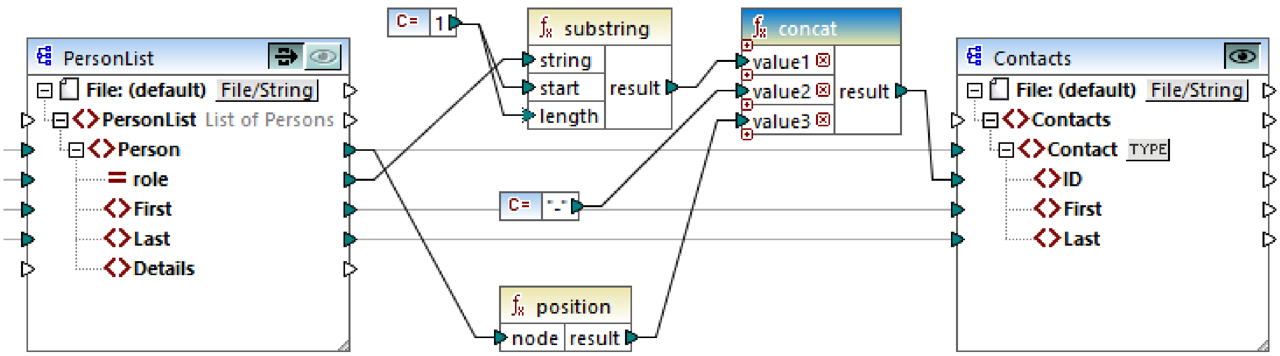
To understand how these two options work, see the example below.

<u>Example</u>

The screenshot below illustrates a part of the **`ChainedPersonList.mfd`** mapping, which is available in the **`MapForceExamples`** folder. In the mapping below, we have pressed the **Direct-connections** button, clicked the header of the **`concat`** component but have not yet pressed the **Source-to-target Connections** button. Therefore, we can now see that only the direct connections of the **`concat`** function to the **`"-"`** constant, the `substring`, `position`, and `Contacts` components are black. The other connections in the mapping are light gray.



The next step is to press the **Source-to-target Connections** button. The screenshot below reflects the changes:



With the **Source-to-target Connections** button pressed, some other connections have become black: (i) the connections of the **`substring`** function to the **`1`** constant and the `PersonList` component, and (ii) the connection of the **`position`** function to the `PersonList` component. However, the connections between the `PersonList` component and its preceding component remain light gray. Thus, when you press the **Source-to-target Connections** button and click on a component, you can trace the component's direct connections. If the selected component is connected to some transformation components [79] (e.g, functions, constants, filters, sort components, SQL-NoSQL-WHERE/ORDER components, if-else conditions, value-maps), you will also be able to see their connections up to the structural components [78] (such as `PersonList` above), variables, join components, or Web service functions, to which these transformation components are connected.

In this section

This section gives an overview of connections and is organized into the following topics:

3.2.1 Connection Types

The following connection types are available in MapForce:

- Target-driven connections [92] (Standard)
- Source-driven connections [93] (Mixed Content)
- Matching-children connections [95]
- Copy-all connections [97] (Copy Child Items)

Target-driven vs. source-driven connections

Target-driven and source-driven connections are mutually exclusive. The choice between these two options depends on the order in which nodes need to be mapped. In target-driven connections, the order of nodes in the output is determined by the *target* schema. This connection type is suitable for most mapping scenarios and is the default connection type in MapForce. A target-driven connection is shown as a solid line (*see screenshot below*).

Target-driven connections might not be suitable when you want to map XML nodes with mixed context (child nodes and text). In this case, a source-driven connection 93 is recommended: The order of nodes in the output is determined by the *source* schema.

Matching-children and copy-all connections

Matching-children and copy-all connections belong to a subset of target-driven and source-driven connections. Matching-children and copy-all connections map data between nodes with child nodes that are similar or the same in the source and target components. Copy-all connections are similar to matching-children connections but have only one thick connection instead of multiple connections, which prevents the mapping area from being visually cluttered.

This section provides information about each connection type and the scenarios when these connection types are useful.

3.2.1.1 Source-Driven Connections

A source-driven connection enables you to automatically map mixed content (text and child nodes) in the same order as in the XML *source* file. A mixed-content connection is shown as a dotted line at parent-node level (*see Mapping the <para> element*). This topic explains how to map mixed content. It also shows the effect of using standard (target-driven) connections with mixed content.

Note: Source-driven connections can also be used in database fields with mixed-content (*Professional and Enterprise editions*).

Note: In order to accept mixed content, target components must have mixed-content nodes.

Mapping mixed content

This topic explains how to map mixed content using a source-driven connection. You will need the following files: **`Tut-OrgChart.xml`**, **`Tut-Orgchart.mfd`**, **`Tut-Person.xsd`**, and **`Tut-OrgChart.xsd`**, which are available in the Tutorial folder 26.

Source XML instance

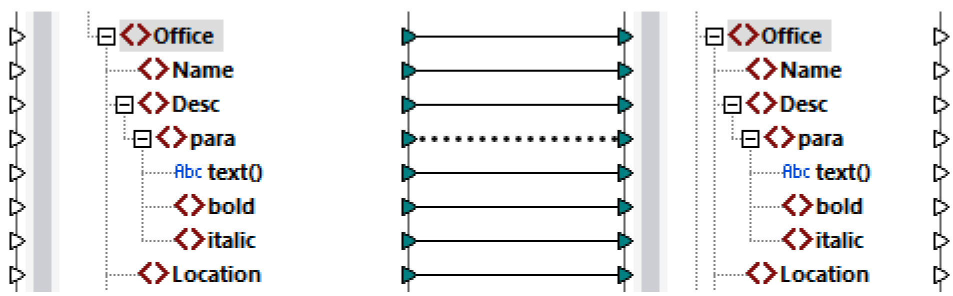
A snippet of **`Tut-OrgChart.xml`** is shown below. In this example, we will focus on the mixed-content element `<para>` with its child nodes `<bold>` and `<italic>`. The `<para>` element also contains a processing instruction (`<?sort alpha-ascending?>`) and a comment (`<!--Company details... -->`), both of which can also be mapped, as shown below. Note the sequence of the `text` and `bold/italic` nodes in the XML instance file.

```
<Desc>
  <para>The company was established in <bold>Vereno</bold> in 1995. Nanonull develops nanoelectronic technologies for <italic>multi-core processors.</italic> February 1999 saw the unveiling of the first prototype <bold>Nano-grid.</bold> The company hopes to expand its operations <italic>offshore</italic> to drive down operational costs.
    <?sort alpha-ascending?>
    <!--Company details: location and general company information.-->
  </para>
  <para>White papers and further information will be made available in the near future.
  </para>
</Desc>
```

Mapping the <para> element

The image below illustrates a portion of **`Tut-Orgchart.mfd`**. In the example below, the dotted line shows that the `<para>` element has mixed content. To create mixed-content connections, take the following steps:

1. Select the menu command **Connection | Auto Connect Matching Children**, which will connect matching child nodes [95] automatically. Alternatively, you can manually map the `<para>` node with its child nodes.
2. Connect the `<para>` item in the source component with the `<para>` item in the target component. A message box will ask if you would like to define the connection as source-driven.
3. Click **Yes** to create a mixed-content connection.
4. Click the **Output** pane to see the result of the mapping. Click the button (**Wrap**) in the **Output** pane toolbar to view the full (i.e., not going beyond the scroll bar) code listing in the **Output** pane. The mixed content of the `<para>` node has been mapped in the same order it appears in the XML source file.



Processing instructions and comments

If your mapping has processing instructions and/or comments and you want to map them, take the steps below:

1. Right-click the mixed content connection (dotted line) and select **Properties**.
2. Under **Source-Drive (Mixed content)**, select the check boxes **Map Processing Instructions** and/or **Map Comments**.

Target-driven connections with mixed content

Choosing target-driven connections for mixed content may have undesirable consequences. To see how target-driven connections affect the order of mixed-content nodes, follow the instructions below:

1. Open **`Tut-OrgChart.mfd`** from the **`Tutorial`** folder.

2. Press the toolbar button (Auto Connect Matching Children [95]). Clear the check box **Create copy-all connections** in the settings for matching-children connections [95]. This will prevent MapForce from creating copy-all connections [97] automatically.
3. Create a connection between the `para` node in the source and the `para` node in the target. A message will ask if you would like to define the connections as source-driven. Click **No**. This creates a target-driven connection.
4. Click the **Output** pane to see the result of the mapping (*screenshot below*).

```
6           <Desc>
7               <para>The company was established in  in 1995. Nanonull develops nanoelectronic
  technologies for  February 1999 saw the unveiling of the first prototype  The company hopes
  to expand its operations  to drive down operational costs.
8                   <bold>Vereno</bold><bold>Nano-grid.</bold><italic>multi-core processors.</
  italic><italic>offshore</italic></para>
9               <para>White papers and further information will be made available in the near
  future.
10              </para>
11          </Desc>
```

The screenshot above shows that the content of the `text()` item in the source has been mapped to the target. However, the order of the child nodes (`bold` and `italic`) in the output corresponds to the order of these nodes in the target XML schema. This means that the `bold` and `italic` elements are not integrated into the text but are mapped separately.

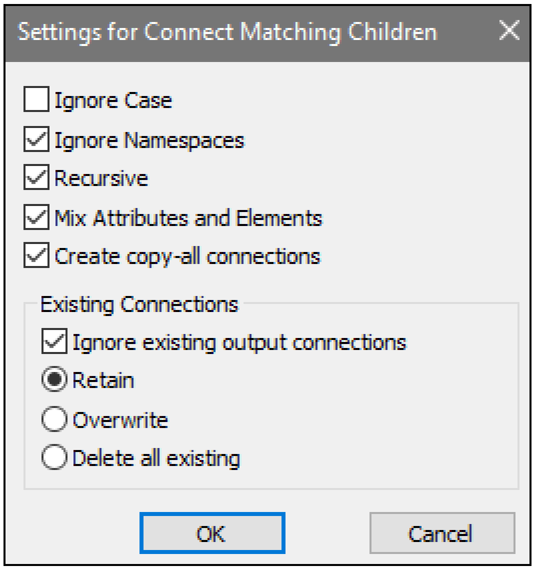
3.2.1.2 Matching-Children Connections

Matching-children connections automatically connect all child nodes which have the same names in the source and target files. To enable this option, do one of the following:

- Click the toolbar button (**Auto Connect Matching Children**).
- Go to the **Connection** menu and click **Auto Connect Matching Children**.

Settings for matching-children connections

To change the settings for matching-children connections, right-click any connection and select the option **Connect Matching Children** from the context menu (*see screenshot below*). Alternatively, go to the **Connection** menu and click **Settings for Connect Matching Children**.



The list below describes the options available in the dialog box **Settings for Connect Matching Children**. The settings in this dialog box apply only when the toolbar button (**Toggle auto connect of children**) is active.

General settings

- *Ignore Case:* Matching children will be connected regardless of the case of child node names.
- *Ignore Namespaces:* Matching children will be connected regardless of the namespaces of child nodes.
- *Recursive:* This option creates new connections between any matching nodes if they have the same names. It does not matter how deep the nodes are nested in the tree.
- *Mix Attributes and Elements:* This option allows creating connections between attributes and elements that have the same names. For example, a connection is created if two `Name` nodes exist, even though one is an element, and the other is an attribute.
- *Create copy-all connections:* This setting is active by default. It creates (if possible) a copy-all connection 97, which maps data between nodes with child nodes that are similar or the same.

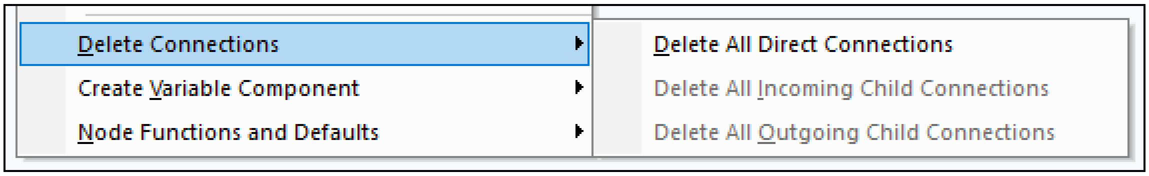
Existing connections

- *Ignore existing output connections:* This option creates additional connections for any matching nodes even if they already have outgoing connections.
- *Retain:* This option keeps existing connections.
- *Overwrite:* This option overwrites existing connections.
- *Delete all existing:* This option deletes all existing connections before creating new ones.

Delete connections as a group

If you want to delete connections as a group, follow the instructions below:

1. Right-click a node name in the component.
2. Select **Delete Connections | Delete All <...> Connections** from the context menu (*see screenshot below*).



- *Delete All Direct Connections:* This option deletes all connections that are directly mapped to or from the selected node.
- *Delete All Incoming Child Connections:* This option is active only if you have right-clicked a parent node in a target component. This option deletes all incoming child connections of the selected parent node.
- *Delete All Outgoing Child Connections:* This option is active only if you have right-clicked a parent node in a source component. This option deletes all outgoing child connections of the selected parent node.

3.2.1.3 Copy-All Connections

Copy-all connections map data between nodes with child nodes that are similar or the same. Copy-all connections are possible only for identical formats (e.g., JSON to JSON or XML to XML). This principle also applies to all text components: flat files, FlexText and EDI files. Since these formats are all text files, you can combine any of them and create a copy-all connection between EDI and FlexText files, for example. See Mapping: Sources and Targets[26] to find out more about the formats that can be used as data sources and targets.

The main benefit of copy-all connections is that they visually simplify the mapping workspace: One connection, represented by a thick line, is created instead of multiple connections (*see example in Create Copy-all Connections Manually*). The subsections below explain how to create copy-all connections automatically and manually.

Create copy-all connections automatically

To create a copy-all connection automatically, take the following steps:

1. Go to the **Connection** menu.
2. Click **Settings for Connect Matching Children**.
3. Check the box **Create copy-all connections** and click **OK**.
4. Press the toolbar button **Toggle auto connect of children**. Alternatively, go to the **Connection** menu and click **Auto Connect Matching Children**.

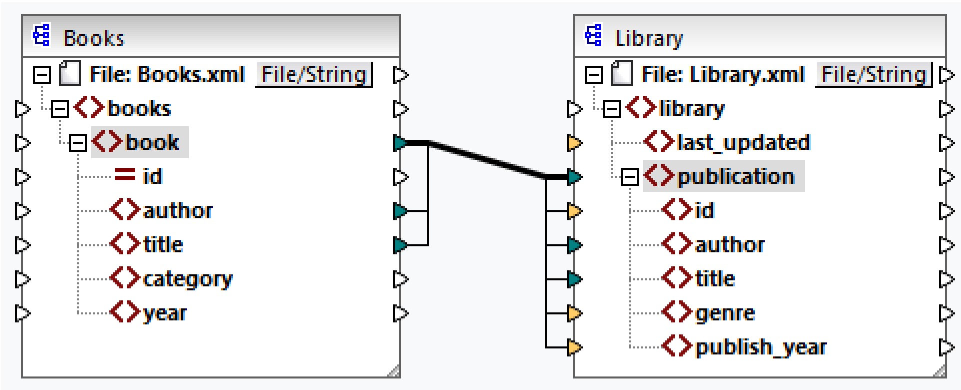
If types and/or names of child nodes in the source and target are not the same, a copy-all connection will not be created automatically, and you will need to create it manually.

Create copy-all connections manually

To create a copy-all connection manually, take the following steps:

1. Add a source file: Click **XML Schema/File** in the **Insert** menu and browse for **`Books.xml`** located in the BasicTutorials folder[26].
2. Add a target file: Click **XML Schema/File** in the **Insert** menu and browse for **`Library.xsd`** located in the same folder as **`Books.xml`**. Click **Skip** when MapForce suggests adding an XML sample file.

3. Map the <book> node of the **Books** component to the <publication> node of the **Library** component. As the structures of the <book> and <publication> elements do not fully coincide, the copy-all connection is not created. Instead, the **Auto Connect Matching Children** function automatically connects all the child nodes with the same name, which is discussed in Tutorial 1 (48).
4. To change the automatic connection to a copy-all connection, right-click the connection between <book> and <publication> and select **Copy-All (Copy Child Items)** from the context menu.
5. A pop-up window will suggest replacing the existing connections with a copy-all connection. Click **OK**. Now the source and target have a copy-all connection (*see screenshot below*).



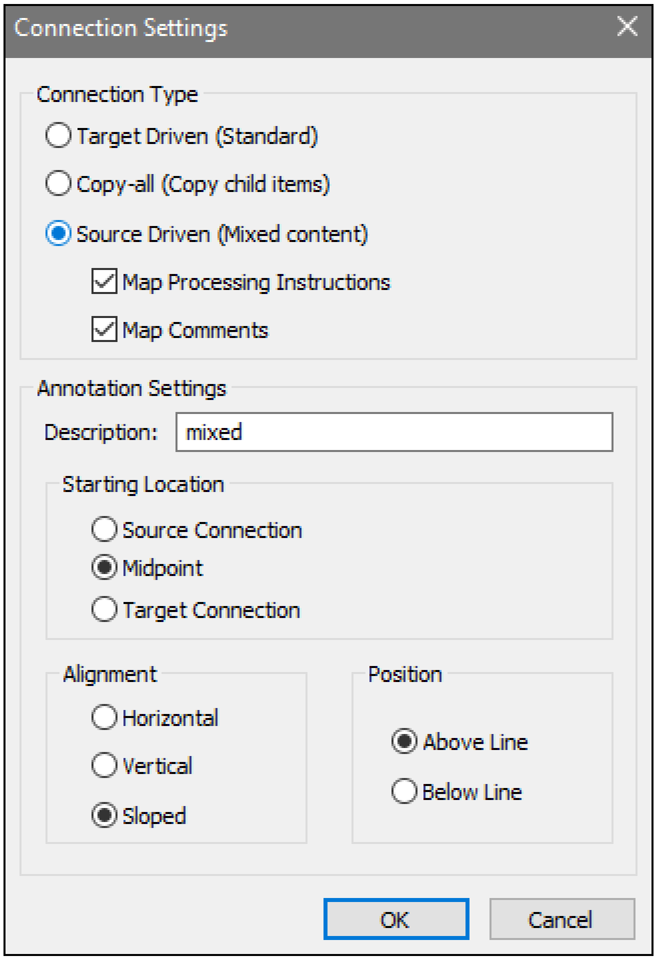
In the mapping above, only two child nodes are identical in the two structures: <author> and <title>. Therefore, a copy-all connection exists between these nodes. Child nodes that are not the same cannot be connected. The screenshot shows that id is not included in the copy-all connection, because its type is not the same in the source and target: id is an attribute in the source and an element in the target. If you try to create a connection between nodes that are not the same, e.g., <category> and <genre>, MapForce prompts you to replace this connection or duplicate the input.

Duplicating input (83) only makes sense if you want the target to accept data from more than one input, which is not required here. If you choose to replace the copy-all connection, a message box prompts you again to resolve or delete the copy-all connection. Click **Resolve copy-all connection** if you want to replace the copy-all connection with individual target-driven connections (92). If you prefer to remove the copy-all connection completely, click **Delete child connections**.

> **Important:**
>
> When you create a copy-all connection between a schema and a parameter of a user-defined function (927), the two components must be based on the same schema. It is not necessary that they both have the same root elements, however.

3.2.2 Connection Settings

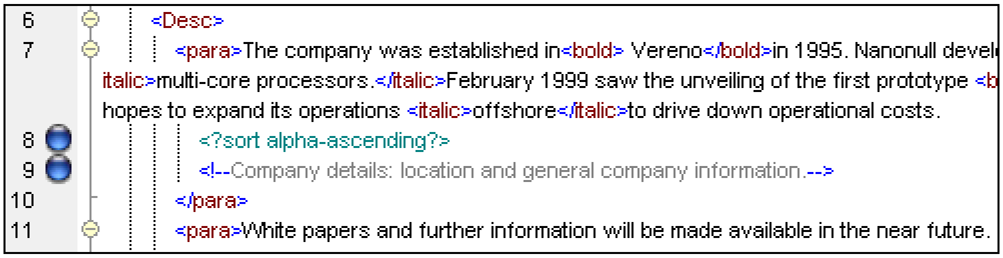
The **Connection Settings** dialog box defines the settings of a connection. To open this dialog box, double-click the connection. Alternatively, right-click a connection and select **Properties** from the context menu. The settings are divided into two parts: connection types and annotation settings. For more information, see the subsections below.



Connection types

You can choose one of the connection types described below:

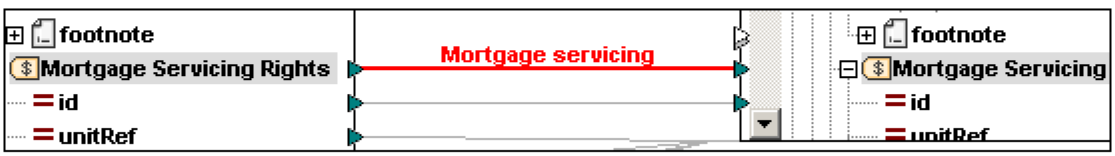
- Target-driven (Standard) [92] connections are suitable for most mapping scenarios.
- Copy-all (Copy child items) [97] connections: If a source and target components have identical or similar nodes with matching child nodes, a copy-all connection will automatically be created between these matching nodes.
- Source-driven (mixed content) [93] connections map mixed content (text and child nodes) in the same order as in the XML source file. If you select **Map Processing Instructions** and/or **Map Comments**, you will be able to include these data groups in the output file (*see screenshot below*).



Annotation settings

The **Annotation Settings** section enables you to label a connection. This option is available for all connection types. To annotate a connection, follow the instructions below:

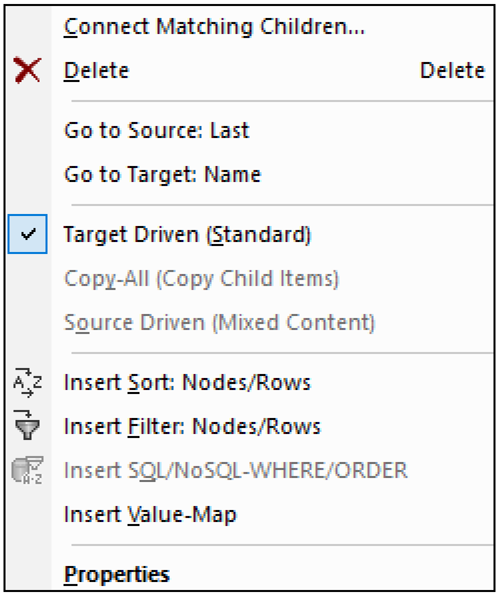
1. Right-click the connection and select **Properties** from the context menu. Alternatively, double-click the connection.
2. Enter the name of the selected connection in the **Description** field. This enables all the options in the **Annotation Settings** section.
2. Use the remaining groups to define the **Starting Location**, **Alignment** and **Position** settings of the label.
3. Press the toolbar button (**Show Annotations**). If the button is not yet visible in the toolbar, activate the button in the **View Options** toolbar.



Note: If the **Show annotations** toolbar button is inactive, you can still see the annotation if you place the cursor over the connection. The annotation will appear as a tooltip if the toolbar button (**Show tips**) is active in the **View Options** toolbar.

3.2.3 Connection Context Menu

This topic describes commands available in the connection context menu. When you right-click a connection, the following context commands become available:



For more information, see the subsections below.

General settings

- *Connect Matching Children:* Opens the Connect Matching Children [95] dialog box. This command is enabled when the connection is allowed to have matching children.
- *Delete:* Deletes the selected connection.
- *Go to Source: <item name>:* Highlights the output connector [77] of the selected connection.
- *Go to Target: <item name>:* Highlights the input connector [77] of the selected connection.

Connection types

See details about connection types in Connection Types [92] and Connection Settings [98].

Insert commands

- *Insert Sort: Nodes/Rows:* Adds a sort [872] component between a source node and a target node.
- *Insert Filter: Nodes/Rows:* Adds a filter [878] component between a source node and a target node.
- *Insert SQL/NoSQL-WHERE/ORDER:* Adds an SQL/NoSQL-WHERE/ORDER component between a source node and a target node (*Professional and Enterprise editions*). For details, see Filter and Sort Database Data [350].
- *Insert Value-Map:* Adds a value-map [884] between a source node and a target node.

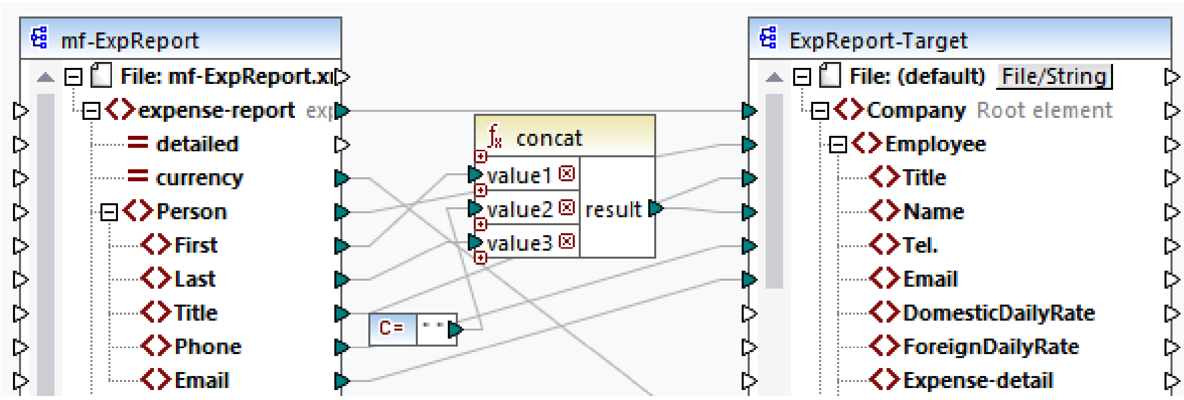
Properties

Opens the Connection Settings [98] dialog box.

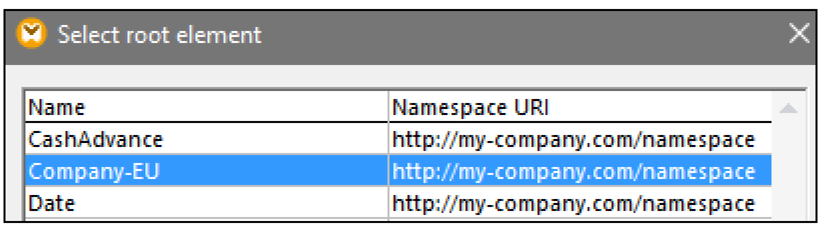
3.2.4 Faulty Connections

There are situations in which you might want to change a schema of a source or target. Changes to a schema can affect the validity of your mapping and result in several faulty connections. This topic explains how to fix such connections after you have changed the schema file. Follow the instructions in the example below to understand how to deal with faulty connections.

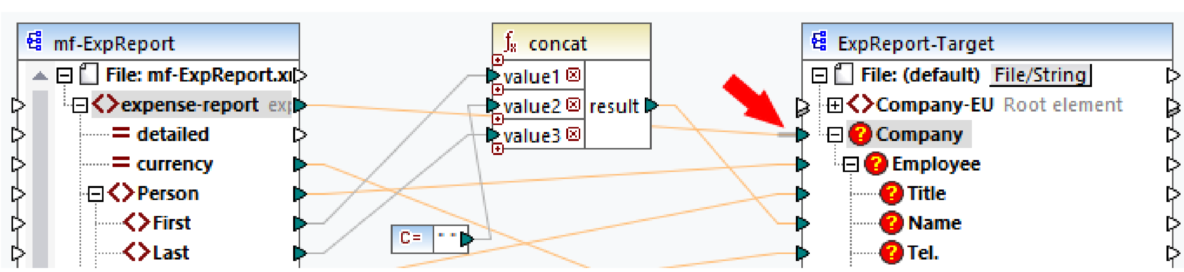
1. Open **`Tut-ExpReport.mfd`** available in the Tutorial folder 26. The portion of this mapping is shown below.



2. Open **`ExpReport-Target.xsd`** in an editor (e.g., Altova XMLSpy) and change the `Company` root element in the target schema to `Company-EU`. You do not need to close MapForce.
3. After you have edited the root element of the target schema**,** the **Changed files** prompt appears in MapForce. Click the **Reload** button. Since the root element has been changed, the component displays multiple faulty nodes.
4. Click **Select new root element** at the top of the component (*see screenshot below*). You can also change the root element by right-clicking the component header and selecting **Change Root Element** from the context menu.



5. Select `Company-EU` as the new root element and click **OK**. The `Company-EU` root element is now visible at the top of the component.
6. Now you need to move the connection from the faulty `Company` node to the new root element. Press and hold the thick section (*see red arrow below*) of `Company`'s connection. Then drag the connection to the `Company-EU` root element.



A notification dialog box will ask whether you would like to move all the matching connected child nodes. You can choose between moving only the selected connection or the selected connection with its child nodes that match the child nodes in the new root element. In our example, we have chosen the option **Include descendent connections**. As soon as you click this button, all the faulty nodes will disappear from the component.

Note: If the node to which you are mapping has the same name as the source node but a different namespace, the notification dialog box will have an additional button **Include descendants and map namespace**. Clicking this button moves child connections of the same namespace as the source parent node to the same child nodes under the different namespace node.

Alternative solution

An alternative solution to the problem discussed above could be deleting the faulty nodes you may no longer need in your mapping. For example, when you delete the connection between the **`concat`** function and `Name`, the `Name` node will disappear from the `ExpReport-Target` component.

Faulty connections in databases (Professional and Enterprise editions)

If your database component has faulty connections, you will need to change the component settings [82]. Clicking the **Change** button in the **Component Settings** dialog box allows you to select a different database or change tables in your database component. All valid/correct connections and relevant database data will be kept if you select a database with the same structure.

3.2.5 Keep Connections after Deleting Components

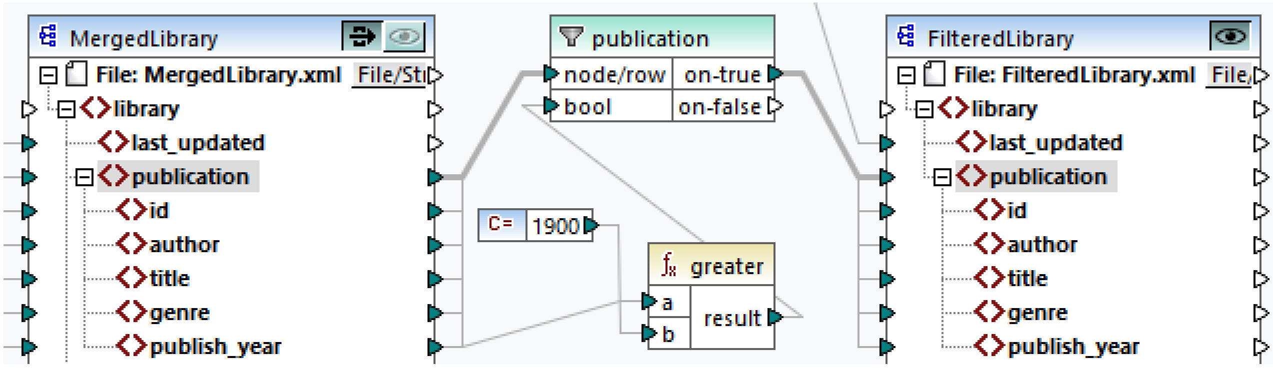
MapForce allows you to keep connections even after deleting some transformation components [78]: e.g., variables, sort and filter components, value-maps, simple inputs, SQL/NoSQL-WHERE/ORDER components. Connections can be single or multiple. Keeping connections might be particularly useful with multiple child connections, because you will not have to restore every single child connection manually after deleting a transformation component. To enable this option, go to **Tools | Options | Editing** and select **Smart component deletion (keep useful connections)**. By default, this option is disabled, which means that deleting a transformation component will also delete its direct connections.

Example

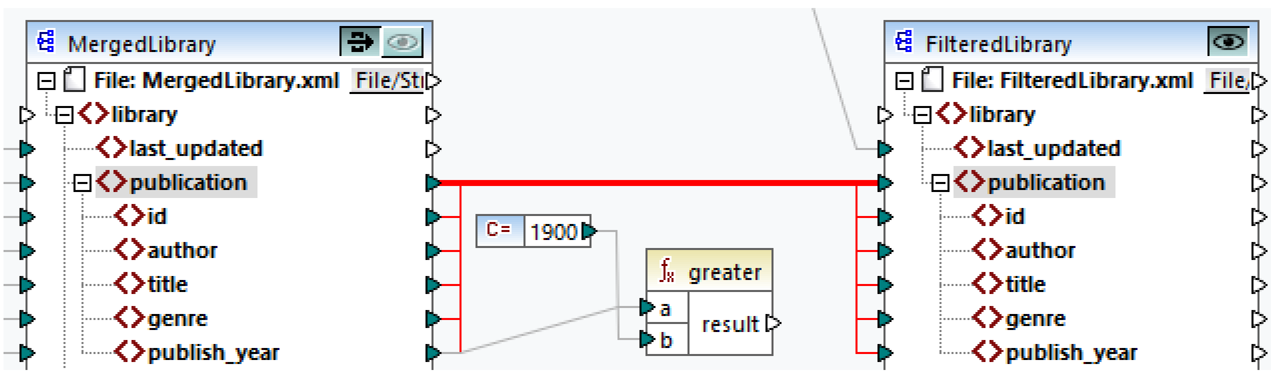
The sample file called **`Tut3-ChainedMapping`** is used to illustrate smart component deletion. The sample file is available in the BasicTutorials [26] folder.

Before deletion

The screenshot below shows that copy-all connections [97] exist between the `MergedLibrary` component and the `publication` filter, and between the `publication` filter and the `FilteredLibrary` component. Now we want to delete the `publication` filter but keep the copy-all connections. In order to do that, select the check box **Smart component deletion** in the **Options** dialog box (*see above*).



After deletion
After the `publication` function has been deleted, the copy-all connection has been created directly between the `publication` node in `MergedLibrary` and the `publication` node in `FilteredLibrary` (*see screenshot below*).



Note: If a filter component has both `on-true` and `on-false` outputs connected, the connections of both outputs will be kept.

3.3 General Procedures and Features

In addition to creating mappings, you can also validate your mapping and output, generate code, use text view features and define mapping settings. This section is organized into the following topics:

3.3.1 Validation

This topic explains how to validate mappings. The topic also shows how to preview, save and validate your output.

Validate mappings

MapForce validates mappings automatically when you click the **Output** pane. You can also validate your mapping manually, which can help you identify and correct potential errors and warnings before running the mapping. To validate a mapping manually, click the **Mapping** pane and then do one of the following:

- Click **Validate Mapping** in the **File** menu.
- Click  (**Validate**) in the toolbar.

When you validate a mapping, MapForce checks, for example, for unsupported component types, incorrect or missing connections. To find out more about validation statuses, see Messages Window [37]. The **Messages** window also allows you to take message-related actions [37]. To display the result of each validation in an individual tab, click the numbered tabs available on the left side of the **Messages** window. This may be useful, for example, if you work with multiple mapping files simultaneously.

Validation of transformation components

Validation of transformation components [79] works as follows:

- If a mandatory **input connector** is not connected, an error message is generated, and the transformation is stopped.
- If an **output connector** is not connected, a warning is generated, and the transformation process continues. The component, which has caused the warning, and its data are ignored and not mapped to the target.

Preview and validate output

MapForce allows you to preview the output without having to run and compile the generated code with an external processor or compiler. In general, it is a good idea to preview the transformation output in MapForce before processing the generated code externally. When you preview the mapping results, MapForce executes the mapping and shows the output in the Output pane [40].

Once the data is available in the **Output** pane, you can validate and save it if necessary. You can also use the **Find** command (**Ctrl + F**) to quickly locate a particular text pattern in the output file. For more information, see Text View Search[111]. Any error, warning or information messages related to the mapping execution are displayed in the Messages window[36].

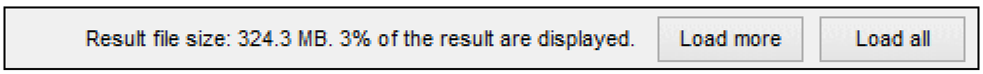
If you select C++, C#, or Java (*Professional and Enterprise editions*) as a transformation language[27], MapForce executes the mapping using its built-in transformation engine and displays the result in the **Output** pane.

To save the transformation output, click the **Output** pane and then do one of the following:

- Click **Save Output File** in the **Output** menu.
- Click (**Save generated output**) in the toolbar.

Load options

When you preview large output files, MapForce limits the amount of data displayed in the **Output** pane. In this case, the **Load more** button appears in the lower area of the pane (*see screenshot below*). Clicking the **Load more** button adds the next piece of data. You can configure the preview settings from the **General** tab of the **Options** dialog box. For more information, see MapForce Options[1473].

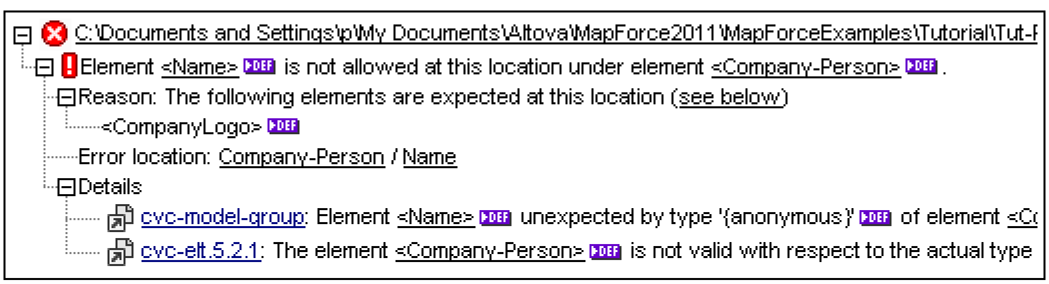


Validate output

As soon as the output becomes available in the **Output** pane, you can validate the output against the schema associated with it. Note that the **Validate Output** button and its corresponding menu command (**Output | Validate Output File**) are enabled only if the output file supports validation against a schema. The result of the validation is displayed in the **Messages** window. To validate the output, do one of the following:

- Open the **Output** pane and click (**Validate Output**) in the toolbar.
- Open the **Output** pane and click **Validate Output File** in the **Output** menu.

The screenshot below illustrates unsuccessful validation. The **Messages** window contains detailed information on the errors. For example, if you click the `<Name>` link, MapForce will highlight this element in the **Output** pane.



3.3.2 Code Generation

You can generate code from a mapping depending on the language you have selected as a data transformation language 27. You can generate code in the following languages:

- XSLT 1.0/XSLT 2.0/XSLT 3.0
- XQuery (*Professional and Enterprise editions*)
- Java (*Professional and Enterprise editions*)
- C# (*Professional and Enterprise editions*)
- C++ (*Professional and Enterprise editions*)

Example
If you want to generate XSLT code, follow the instructions below:

1. Select the menu item **File | Generate Code in | XSLT**.
2. Select the folder where you want to save the generated XSLT file and click **OK**. MapForce generates the code and displays the result of the operation in the Messages window 36.

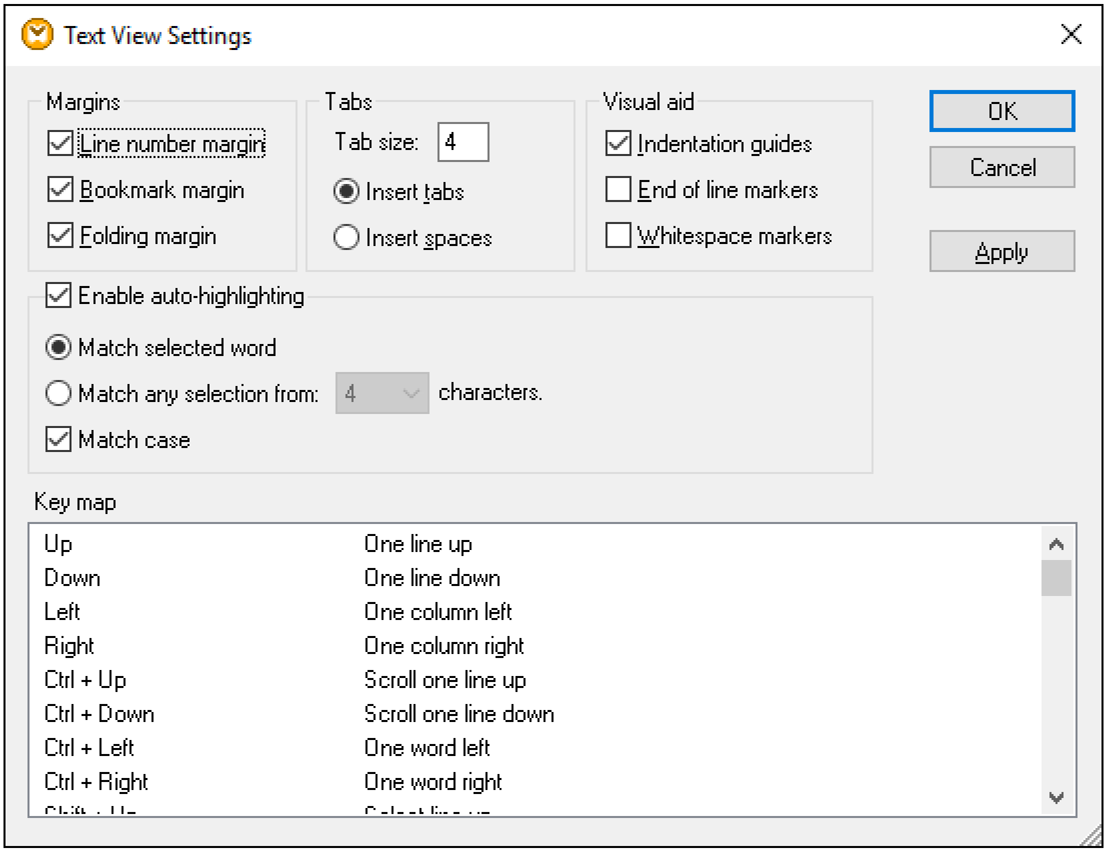
To preview the generated XSLT code, click the **XSLT** pane at the bottom of the **Mapping** window. The same steps apply to any of the languages listed above.

After the code generation has been completed, the destination folder will include the following two files:

1. An XSLT transformation file, named after the target schema. This transformation file has the following format: **`<Mapping>MapTo<TargetFileName>.xslt`**. `<Mapping>` is the value of the **Application Name** field in the mapping settings 114. `<TargetFileName>` is the name of the target mapping component. To change this value, open the settings of the target component and edit the value of the **Component Name** field. For more information, see Change Component Settings 82 and Library paths in generated code 88.
2. A **`DoTransform.bat`** file, which enables you to run the XSLT transformation with Altova RaptorXML Server from the command line. In order to run the command, you will need to install RaptorXML.

3.3.3 Text View Features

The Output pane 40, the XQuery pane 39, and the XSLT pane 38 have multiple visual aids to make the display of text easier: e.g., margins, text highlighting, indentation guides, end-of-line and whitespace markers. You can customize these features in the **Text View Settings** dialog box (*see screenshot below*). The settings in this dialog box apply to the entire application.



To open the **Text View Settings** dialog box, do one of the following:

- Select **Output | Text View Settings**.
- Click  (**Text View Settings**) in the toolbar.
- Right-click the blank area in the **Output** pane and select **Text View Settings** from the context menu.

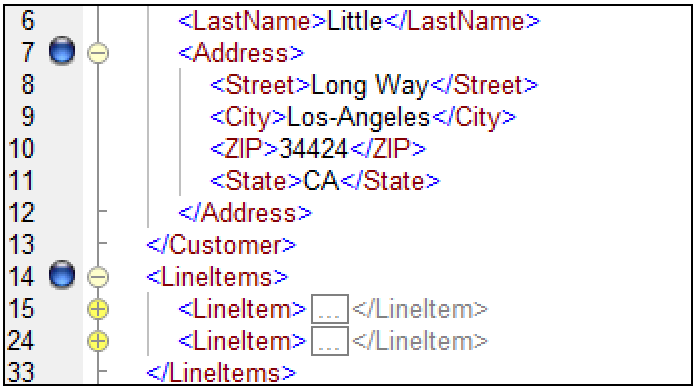
Some of the navigation aids can also be toggled from the **Text View** toolbar, the application menu, or keyboard shortcuts. For more more information about shortcuts, see the **Key map** section of the **Text View Settings** dialog box shown above.

See the list of available settings below.

Margins

Line number margin
Line numbers are displayed in the line number margin, which can be toggled on and off in the **Text View Settings** dialog box. When a section of text is collapsed, the line numbers of the collapsed text are also hidden.

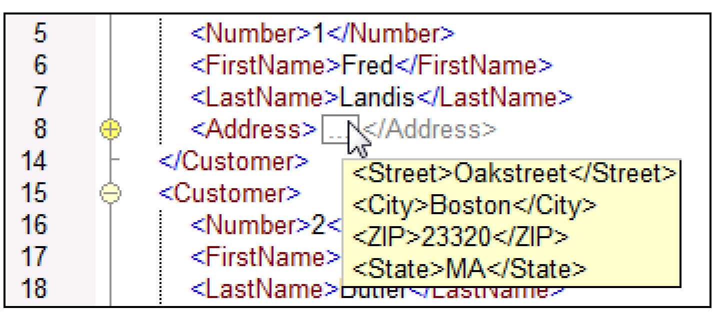
Bookmark margin
Lines in the document can be bookmarked for quick reference and access. If the **Bookmark margin** check box in the **Text View Settings** dialog box is selected, bookmarks are displayed in the bookmarks margin (*see screenshot below*). If the **Bookmark margin** check box is not selected, the bookmarked lines are highlighted in cyan.



You can edit and navigate bookmarks using the commands given in the table below. The commands are available in the **Output** menu and also through the context menu when you right-click the **Output**, **XSLT** or **XQuery** pane.

	Insert/Remove Bookmark (Ctrl + F2)
	Go to Next Bookmark (F2)
	Go to Previous Bookmark (Shift + F2)
	Delete All Bookmarks (Ctrl + Shift + F2)

Folding margin
Source folding refers to the ability to expand and collapse nodes. This feature is displayed in the source folding margin. The margin can be activated or disabled in the **Text View Settings** dialog box. To expand or collapse portions of text, click the + and - nodes at the left side of the window. Any portions of collapsed code are displayed with an ellipsis symbol (*see screenshot below*). To preview the collapsed code without expanding it, hover over the ellipsis. This opens a tooltip that displays the previewed code, as shown in the screenshot below. Note that if the previewed text is too big to fit in the tooltip, an additional ellipsis appears at the end of the tooltip.



⊟ Enable auto-highlighting

The **Enable auto-highlighting** setting allows you to see all the matches of the selected piece of text. The selection is highlighted in pale blue, and the matches are highlighted in light brown. The selection and its matches are indicated as gray marker-squares in the scroll bar. The current cursor position is shown as the blue cursor-marker in the scroll bar. A selection can be an entire word or a fixed number of characters. You can also specify whether case should be taken into account or not.

For character selection, you can specify the minimum number of characters that must match, starting from the first character in the selection. For example, you can choose to match two or more characters. For word searches, the following items are considered to be separate words: element names (without angular brackets), the angular brackets of element tags, attribute names, and attribute values without quotes.

⊟ Visual aid

Indentation guides
Indentation guides are vertical lines that indicate the extent of a line's indentation. They can be toggled on and off in the **Text View Settings** dialog box. The **Insert tabs** and **Insert spaces** options take effect when you use the option **Output | Pretty-Print XML Text**.

End-of-line and whitespace markers
End-of-line and whitespace markers (*see screenshot below*) can be toggled on in the **Text View Settings** dialog box. The arrows represent tab characters. The *CR* abbreviation stands for a carriage return. The dots represent space characters.

```
<?xml·version="1.0"·encoding="UTF-8"?>CR
<books·xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"·
xsi:noNamespaceSchemaLocation="books.xsd">CR
→<book·id="1">CR
→→<author>Mark·Twain</author>CR
→→<title>The·Adventures·of·Tom·Sawyer</title>CR
→→<category>Fiction</category>CR
→→<year>1876</year>CR
→</book>CR
</books>CR
```

⊟ Other text view settings

Syntax coloring
Syntax coloring is another visual aid that makes code listings more reader-friendly. Syntax coloring depends on the semantic value of the text. For example, in XML documents, depending on whether the XML node is an element, attribute, content, CDATA section, comment, or processing instruction, the node name (and in some cases the node's content) is colored differently.

Zooming in and out
You can zoom in and out by scrolling (with the scroll-wheel of the mouse) while holding the **Ctrl** key pressed. Alternatively, press the - or + keys while holding the **Ctrl** key pressed.

Pretty-printing
The **Pretty-Print XML Text** command reformats the active XML document in **Text View** to give a structured display of the document. By default, each child node is separated from its parent by four space characters. This can be customized in the **Text View Settings** dialog box. To pretty-print an XML document, select the menu command **Output | Pretty-Print XML Text** or click (**Pretty-print**) in the toolbar.

Word wrapping

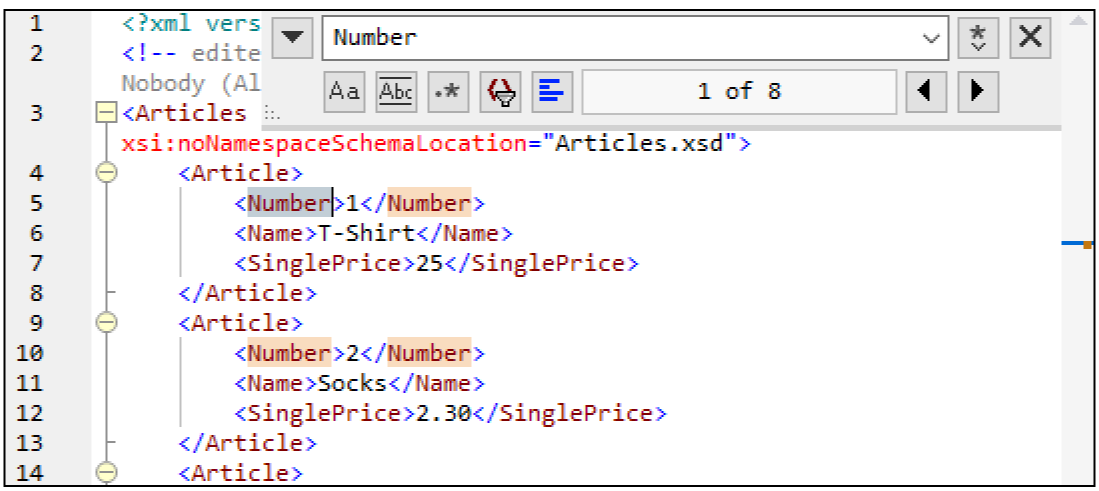
Word wrapping helps display a code listing within the borders of the working area. If the word wrap setting is not enabled, some portions of text may not be fully visible in the working area. To toggle word wrapping in the currently active document, select the menu command **Output | Word Wrap** or click  (**Word Wrap**) in the toolbar.

3.3.4 Text View Search

The text in the **Output** pane, the **XQuery** pane, and the **XSLT** pane can be searched with an extensive range of options and visual aids.

You can search for a term in the entire document or within a text selection. To start a search , press **Ctrl+F** or select the menu command **Edit | Find**. You can enter a string or use the combo box to select a string from one of the last 10 strings. When you enter or select a string, all matches are highlighted, and the positions of the matches are indicated by orange markers in the scroll bar (*see screenshot below*). The position of the currently selected match (highlighted in gray) depends on where the cursor was last located.

You can see the total number of matches and the index position of the currently selected match. Use the  (**Previous**) and  (**Next**) buttons to switch between the matches.



Find options

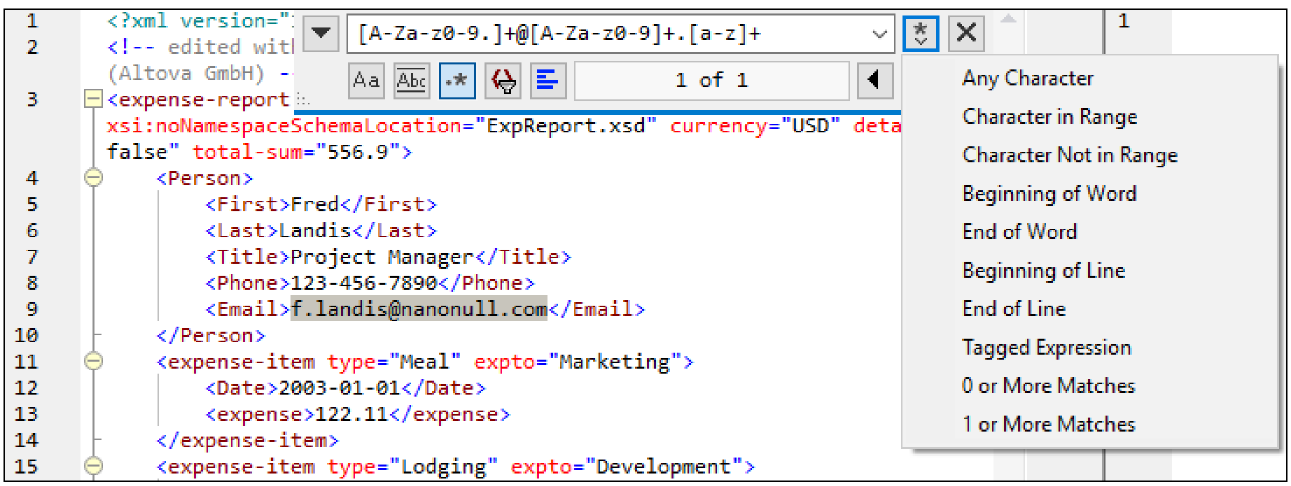
You can specify find criteria with the help of the buttons located under the search field. The list of the available options is given in the table below.

Option	Icon	Description
Match case		Does a case-sensitive search: e.g., *Address* is not the same as *address*.

Option	Icon	Description
Match whole word		Only identical words will match.
Use regular expression		If this option is toggled on, the search term will be read as a regular expression. See *Regular expressions* below.
Find anchor		The position of an anchor depends on the place where the cursor was last located. Clicking the **Previous** and **Next** buttons does not change the position of the anchor.
Find in selection		A selection is a marked piece of text. To find a term within a selection, mark a piece of text, press **Ctrl+F**, make sure the **Find in selection** button is pressed, and type the term in the search field.

Regular expressions

You can use regular expressions to find a text string. To do this, switch on the **Use regular expressions** option (*see table above*). Then enter a regular expression in the search field. Clicking (**Regular Expression Builder**) gives you a list of sample regular expressions (*see below*). The screenshot below shows a regular expression that helps find email addresses.



Regular expression metacharacters

The table below shows metacharacters that you can use to find and replace text. All the metacharacters except for the last two correspond to the menu items in **Regular Expression Builder** (*see above*).

Menu item	Metacharacters	Description
Any Character	`.`	Matches any character. This is a placeholder for a single character.
Character in Range	`[...]`	Matches any characters in this set. For example, **`[abc]`** matches any of the characters `a`, `b` or `c`. You can also use ranges: e.g., **`[a-z]`** for any lower case character.
Character Not in	`[^...]`	Matches any characters not in this set. For example, **`[^A-Za-z]`**

Menu item	Metacharacters	Description
Range		matches any character except an alphabetic character.
Beginning of Word	`\<`	Matches the beginning of a word.
End of Word	`\>`	Matches the end of a word.
Beginning of Line	`^`	Matches the beginning of a line unless it is used inside a set (*see above*).
End of Line	`$`	Matches the end of a line. For example, **`A+$`** matches one or more `A`'s at the end of a line.
Tagged Expression	`(abc)`	The parentheses mark the start and end of a tagged expression. Tagged expressions may be useful when you need to tag ("remember") a matched region to refer to it later. Up to nine sub-expressions can be tagged and then back-referenced later. For example, **`(the) \1`** matches the string `the the`. This expression can be explained as follows: Match the string `the` and remember it as a tagged region; the expression must be followed by a space character and a back-reference to the tagged region matched previously.
0 or More Matches	`*`	Matches zero or more matches of the preceding expression. For example, **`Sa*m`** matches `Sm`, `Sam`, `Saam`, `Saaam` and so on.
1 or More Matches	`+`	Matches one or more occurrences of the preceding expression. For example, **`Sa+m`** matches `Sam`, `Saam`, `Saaam` and so on.
	`\n`	Where `n` is `1` through `9`, `n` refers to the first through ninth tagged region (*see above*).
	`\x`	Allows using a character `x`, which would otherwise have a special meaning. For example, `\[` would be interpreted as `[` and not as the start of a character set.

Find special characters

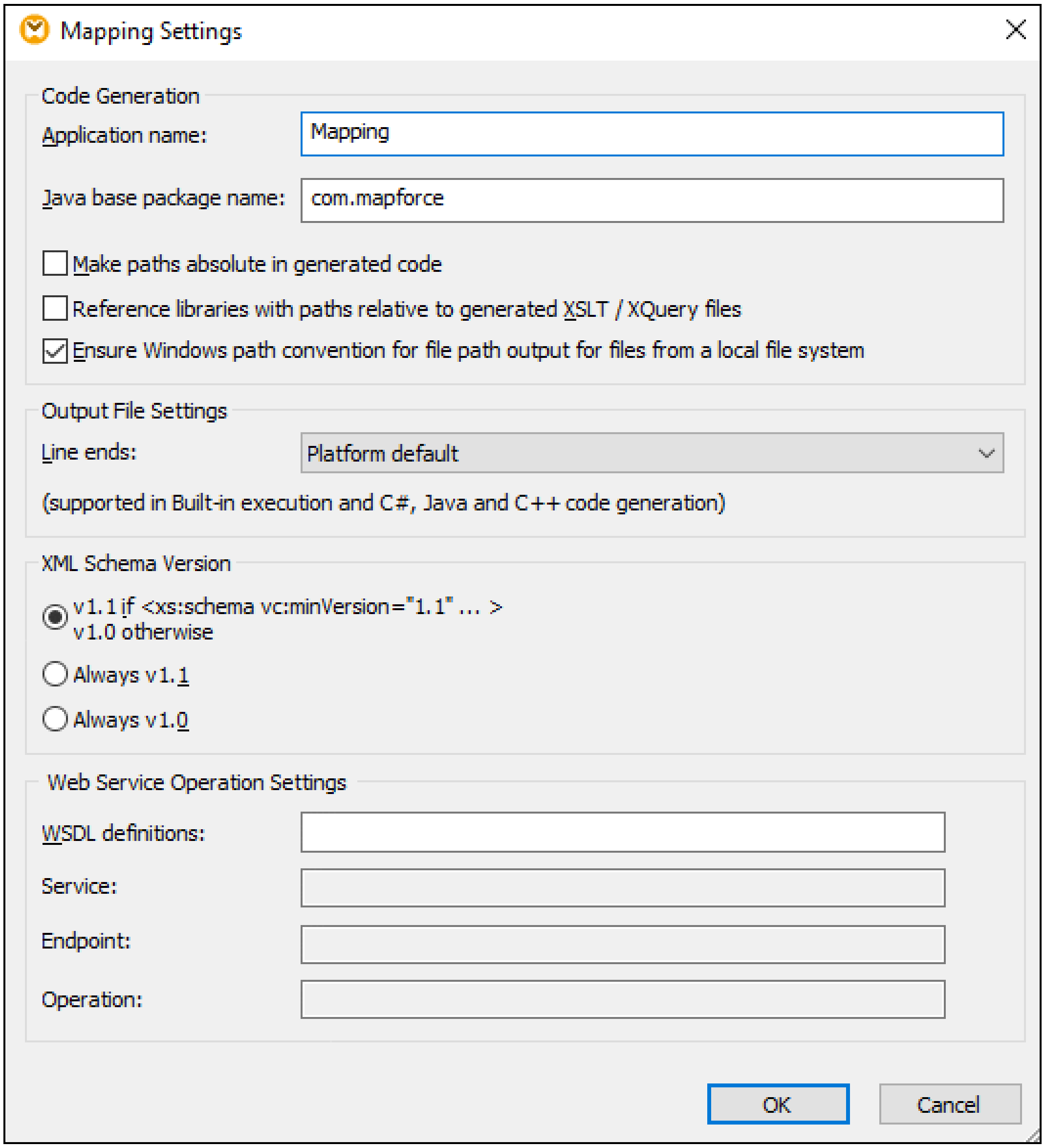
If the **Use regular expressions** option is enabled, you can search for any of the following special characters within the text:

- `\t` (Tab)
- `\r` (Carriage Return)
- `\n` (New line)
- `\\` (Backslash)

For example, to find a tab character, press **Ctrl + F**, select the **Use regular expressions** option, and enter **`\t`** in the **Find** dialog box.

3.3.5 Mapping Settings

The **Mapping Settings** dialog box (*see screenshot below*) allows you to define document-specific settings. To open this dialog box, go to the **File** menu and click **Mapping Settings**. Alternatively, right-click the empty area in the mapping pane and select **Mapping Settings** from the context menu.



The available settings are described in the subtopics below.

Code Generation

- *Application name:* Defines the prefix of the generated XSLT file or the name of the generated Java, C#, or C++ application (*Professional and Enterprise editions*).

- *Java base package name (Professional and Enterprise editions):* This option applies when Java is selected as a transformation language. The option defines the base package name of the Java output.

- Make paths absolute in generated code: This check box affects all paths in mapping components, except paths to external library files (e.g., XSLT libraries). The check box defines whether the file paths should be relative or absolute in the generated program code, in MapForce Server Execution files (`.mfx`) and in mapping functions deployed to FlowForce Server. For more information, see Paths in Execution Environments 87.

- *Reference libraries with paths relative to the generated XSLT/XQuery files:* This check box is applicable when the mapping language is XQuery (*Professional and Enterprise editions*) or XSLT. This option is useful if your mapping references an XSLT or XQuery library, and you plan to generate XSLT or XQuery files from the mapping. Select this check box if you want the library paths to be relative to the directory of the generated XSLT or XQuery code. If the check box is not selected, the library paths will be absolute in the generated code. See also Library paths in generated code 88.

- *Ensure Windows path convention for file path:* This check box is applicable when the mapping language is XQuery (*MapForce Professional and Enterprise editions*), XSLT 2.0 or XSLT 3.0. The check box makes sure that Windows path conventions are followed. When you output XSLT 2.0, XSLT 3.0 or XQuery, the currently processed file name is internally retrieved with the help of the **`document-uri`** function, which returns a path in the `file://URI` format for local files. When this check box is selected, a `file://URI` path specification is automatically converted to a complete Windows file path (e.g., `C:\...`) to simplify further processing.

Output File Settings (Professional and Enterprise editions)

The **Line ends** combo box allows you to specify the line endings of the output files. *Platform default* means the default option for the target operating system: e.g., Windows (CR+LF), macOS (LF), or Linux (LF). You can also select a specific line ending manually. The settings you select here are important when you compile a mapping to a MapForce Server Execution file (`.mfx`) or when you deploy a mapping to FlowForce Server running on a different operating system.

XML Schema Version

This option allows you to define the XML schema version used in the mapping file. Note that not all version 1.1 specific features are currently supported. If the `xs:schema vc:minVersion="1.1"` declaration is present, version 1.1 will be used; if not, version 1.0 will be used.

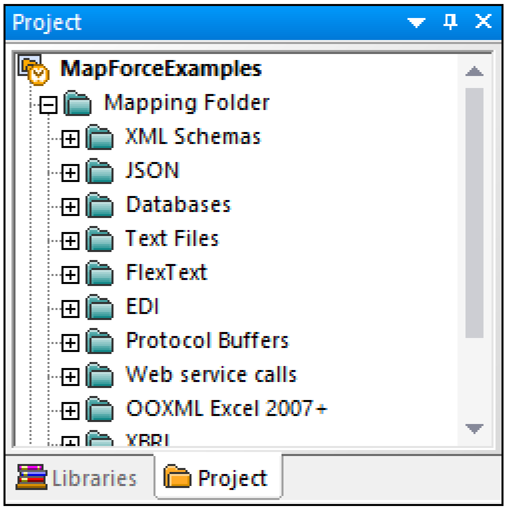
If the XSD document has no `vc:minVersion` attribute or the value of the `vc:minVersion` attribute is other than `1.0` or `1.1`, XSD 1.0 will be the default mode. Do not confuse the `vc:minVersion` attribute with the `xsd:version` attribute. The first attribute has the XSD version number, while the second attribute has the document version number. Changing this setting in an existing mapping causes the reloading of all schemas of the selected XML schema version and might also change its validity.

Web Service Operation Settings (Enterprise editions)

The **WSDL Definitions**, **Service**, **Endpoint** and **Operation** fields are automatically filled if the mapping document is part of a Web service implementation.

3.4 Projects

In addition to creating standalone mappings, you can also create mapping projects that include multiple mappings. Mappings added to a project are accessible from the **Project** window (*see screenshot below*).



The main advantage of projects is that you can define common code generation settings (e.g., the target language and the output directory) for all the mappings included in that particular project. You can also create folders inside projects and specify custom code generation settings for each individual folder in a project. For more information about MapForce-generated program code (in C++, C#, and Java), see Code Generator 1492. MapForce Project files are saved with the `.mfp` extension.

In MapForce Enterprise Edition, you can additionally create Web Service projects. Such projects enable you to generate Java or C# program code that implements SOAP Web services, based on existing WSDL (Web Services Description Language) files. For further information about Web Service projects, see Implementing Web Services 1204.

3.4.1 Project Basics

The subsections below will help you get started with a project. Procedures associated with projects can be broadly divided into (i) creating a project, (ii) organizing a project, and (iii) performing different actions. See information about these procedures in the subsections below.

New project

To create a new project, take the steps below:

1. Click the button in the toolbar. Alternatively, go to the **File** menu and click **New**.
2. Select **Project File** and click **OK**.
3. Enter the project name in the **Save Project As** dialog box and click **Save**. The new project is now displayed in the **Project** window.

To close a project, go to the **Project** menu and click **Close Project**.

Project organization

Add a mapping to a project

If you want to add a currently open mapping to a project, do one of the following:

- Go to the **Project** menu and click **Add Active File to Project**.
- Right-click the relevant project in the **Project** window and select **Add Active File to Project** from the context menu.

To add existing mapping files to a project, do one of the following:

- Go to the **Project** menu and click **Add Files to Project**.
- Right-click the relevant project in the **Project** window and select **Add Files to Project**.

Tip: If you want to add multiple files, hold the **Ctrl** key while selecting the files in the **Open** dialog box.

Delete a file from a project

To remove a file or folder from a project, do one of the following:

- Choose the file you would like to delete in the **Project** window. Right-click the file and select **Delete** from the context menu.
- Select the relevant file in the **Project** window and press **Delete**.

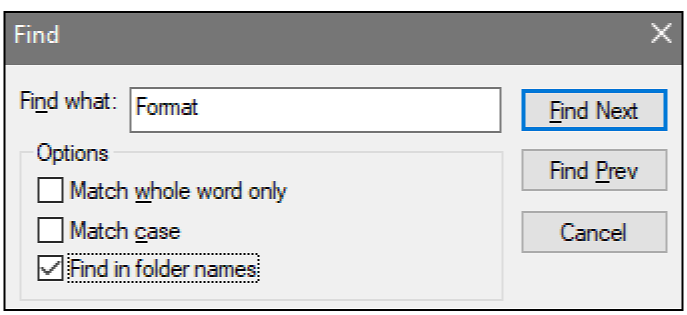
MapForce project files have a **`.mfp`** extension. You can open existing MapForce projects in the same way as mappings: Go to the **File** menu and click **Open**. By default, when you run MapForce for the first time, you will see the **`MapForceExamples.mfp`** project in the **Project** window.

Project-related actions

Search a project

To search a project for files, follow the instruction below:

1. In the **Project** window, click the project or folder to be searched.
2. Press **Ctrl + F**. The **Find** dialog box allows you to define your search options. For example, if you want to include folder names in the search, select the **Find in folder names** option (*see screenshot below*).



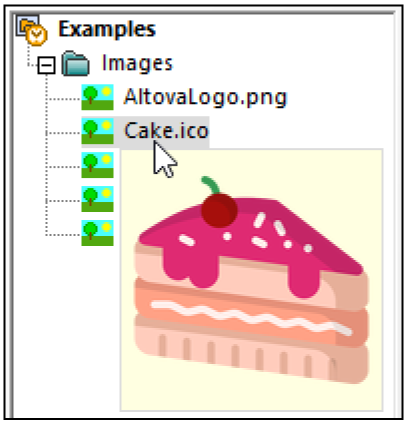
Generate code for your project

In projects, you can generate code for (i) individual mappings, (ii) a specific folder, or (iii) the entire project.

To generate code for a mapping or folder in your project, right-click the relevant mapping or folder and select **Generate Code** or **Generate code in**. If you select **Generate Code**, the code will be generated in the language specified in the project settings 118. You can also choose to generate code in one of the languages available in your MapForce edition. For more information, see Code Generation 107.

To generate code for the entire project, go to the **Project** menu and select **Generate Code for Entire Project**. Alternatively, right-click the name of the project in the **Project** window and select **Generate Code**. The code will be generated in the language specified in the project settings 118. For the entire project, you can also select a language to generate code in in the **Project** menu or in the project's context menu. The choice of languages depends on your MapForce edition. For more information, see Code Generation 107.

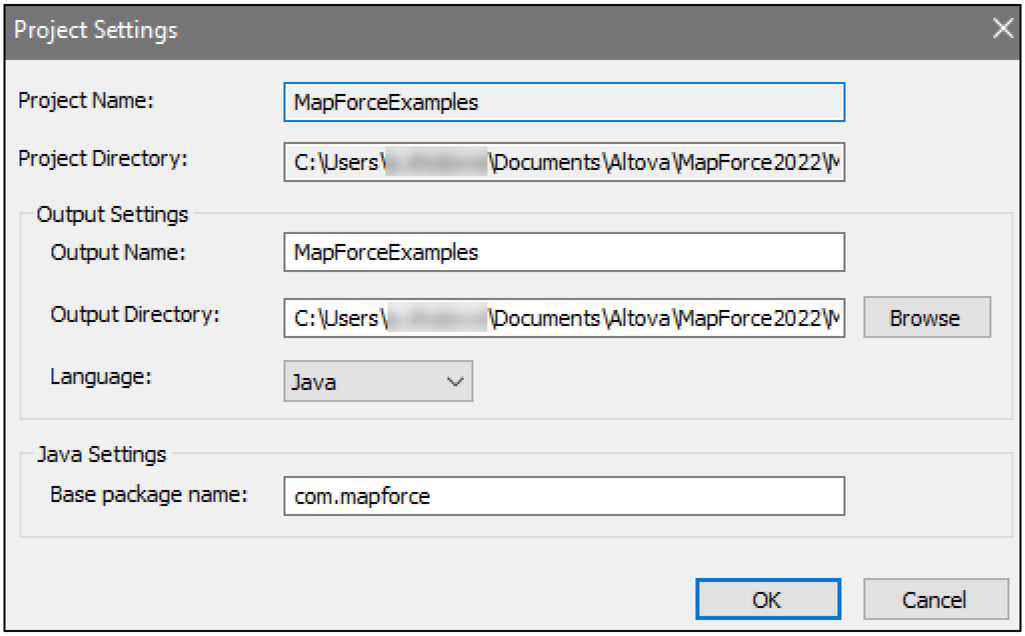
Preview images
The **Project** window allows you to preview images of the following formats: **`.png`**, **`.jpeg`**, **`.gif`**, **`.bmp`**, **`.tiff`**, and **`.ico`** (*see screenshot below*). Double-clicking an image file will open it in an external application, which depends on file association in Windows.



3.4.2 Project Settings

For any project, you can specify code generation settings that will affect all the mappings inside your project. To open the **Project Settings** dialog box (*see screenshot below*), do one of the following:

- Right-click the project name in the **Project** window and select **Properties** in the context menu.
- Go to the **Project** menu and click **Properties**.



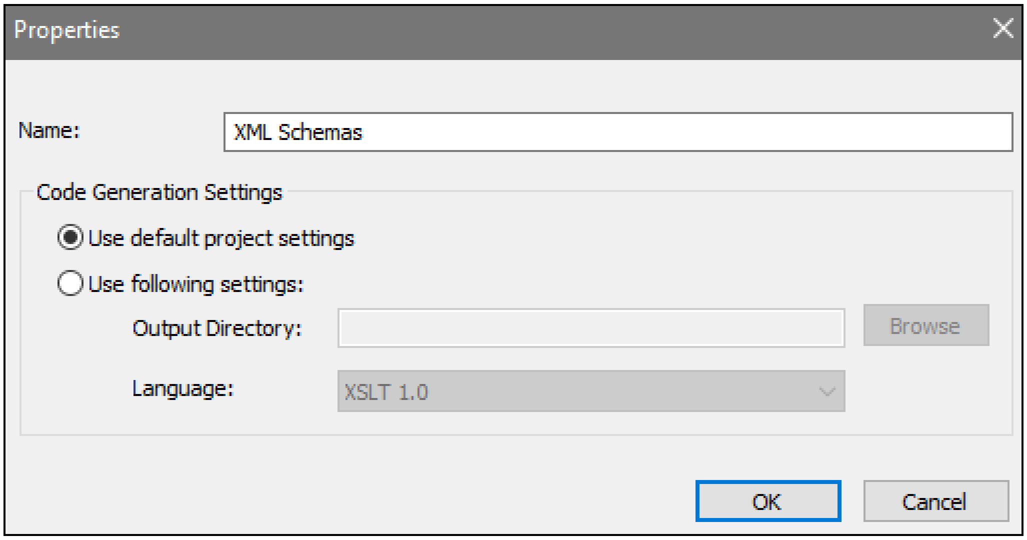
The available settings are listed below. Note that the project name and the project directory cannot be changed after the project has been created.

- *Output name:* The value entered in this text field determines the names of the generated project/solution and other objects in the generated code.
- *Output directory:* Defines the Windows folder where the generated code from all mappings in this project will be saved. By default, the output is saved to the **`MapForceExamples\output\`** directory.
- *Language:* Defines a code generation language for all mapping files in this project.
- *Base package name:* This setting applies if Java has been selected as a transformation language. The setting defines the name of the base package in the generated Java project.

3.4.3 Project Folders

MapForce enables you to organize mappings inside a project into folders. You can create as many folders as you need and add mappings to them. Such folders are virtual and exist only inside a MapForce project: These folders do not correspond to the folders on your operating system. One of the advantages of creating folders in a project is that you can define common code generation settings for all the mapping files in that particular folder . To create a folder inside a MapForce project, take the steps below:

1. Go to the **Project** menu and click **Create Folder**. Alternatively, right-click the project in the **Project** window and select **Create Folder**.
2. In the **Properties** dialog box (*see screenshot below*), enter the required code generation settings and click **OK**.



The list below describes the settings you can define in the **Properties** dialog box.

- *Name:* This is the name of the folder in your project.
- *Use default project settings:* This option means that the code generation settings in the current folder are the same as in the entire project. Therefore, when you generate code from your project, MapForce will use the code generation settings defined in the project settings (118). If your folder requires custom code generation settings, select **Use following settings** and specify the code output directory and language as required.
- *Output directory:* This is the folder where the generated code from all the mappings in this folder will be saved.
- *Language:* This option defines the code generation language for all the mapping files in this folder.

4 Structural Components

This section provides information about various data formats that you can use as data sources and targets:

- XML and XML Schema 122
- Databases 161
- EDI 444 (this includes EDI-based or EDI-related formats, such as ASC X12, HIPAA X12, HL7 Version 2, IATA PADIS, NCPDP SCRIPT, SAP IDoc, TRADACOMS, UN/EDIFACT)
- CSV and Text Files 563
- Web Services 586
- MapForce FlexText 658
- Microsoft OOXML Excel 2007+ 715
- XBRL 744
- JSON 793
- Protocol Buffers 811
- Binary Files 827

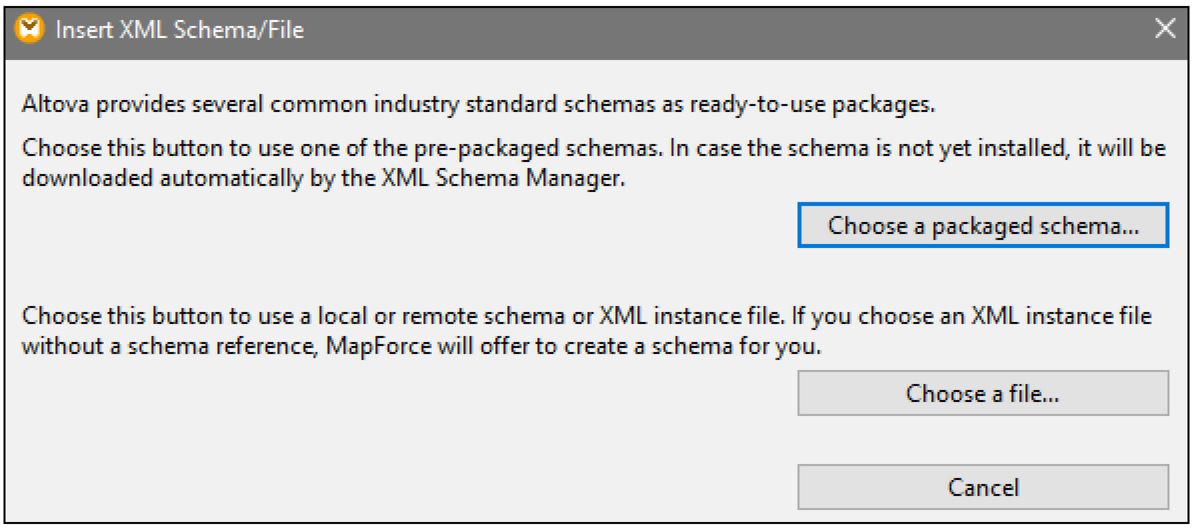
4.1 XML and XML Schema

Altova website: XML Mapping

This section provides information about aspects specific to XML components. For information about basic data transformation scenarios, see Tutorials 42.

Insert XML schema/file

To insert an XML schema/file, select the menu command **Insert | XML Schema/File** or the toolbar button. The dialog box (*see screenshot below*) will prompt you to choose between a packaged industry-standard schema and a local or remote schema/instance file. If you choose a packaged schema, you will be prompted to select an entry point. If the schema you wish to use is not yet installed, it will automatically be downloaded by the XML Schema Manager 145.



Generate an XML schema
When you add a local or remote XML file without a schema reference, MapForce will suggest generating an XML schema for you. You will then be prompted to select the directory where the generated schema should be saved.

When MapForce generates a schema from an XML file, data types for elements/attributes must be inferred from the XML instance document and may not be exactly what you expect. It is recommended that you check whether the generated schema is an accurate representation of the instance data.

If elements or attributes in more than one namespace are present, MapForce generates a separate XML schema for each distinct namespace; therefore, multiple files may be created on the disk.

DTD as a document structure
Starting with MapForce 2006 SP2, namespace-aware DTDs are supported for source and target components. To make mappings possible, the namespace-URIs are extracted from the DTD `xmlns` attribute declarations. However, some DTDs contain `xmlns*` attribute declarations without namespace URIs (e.g., DTDs used by StyleVision). To make such DTDs usable in MapForce, define the `xmlns` attribute with the namespace URI as follows:


```
<!ATTLIST fo:root
   xmlns:fo CDATA #FIXED 'http://www.w3.org/1999/XSL/Format'
   ...
>
```

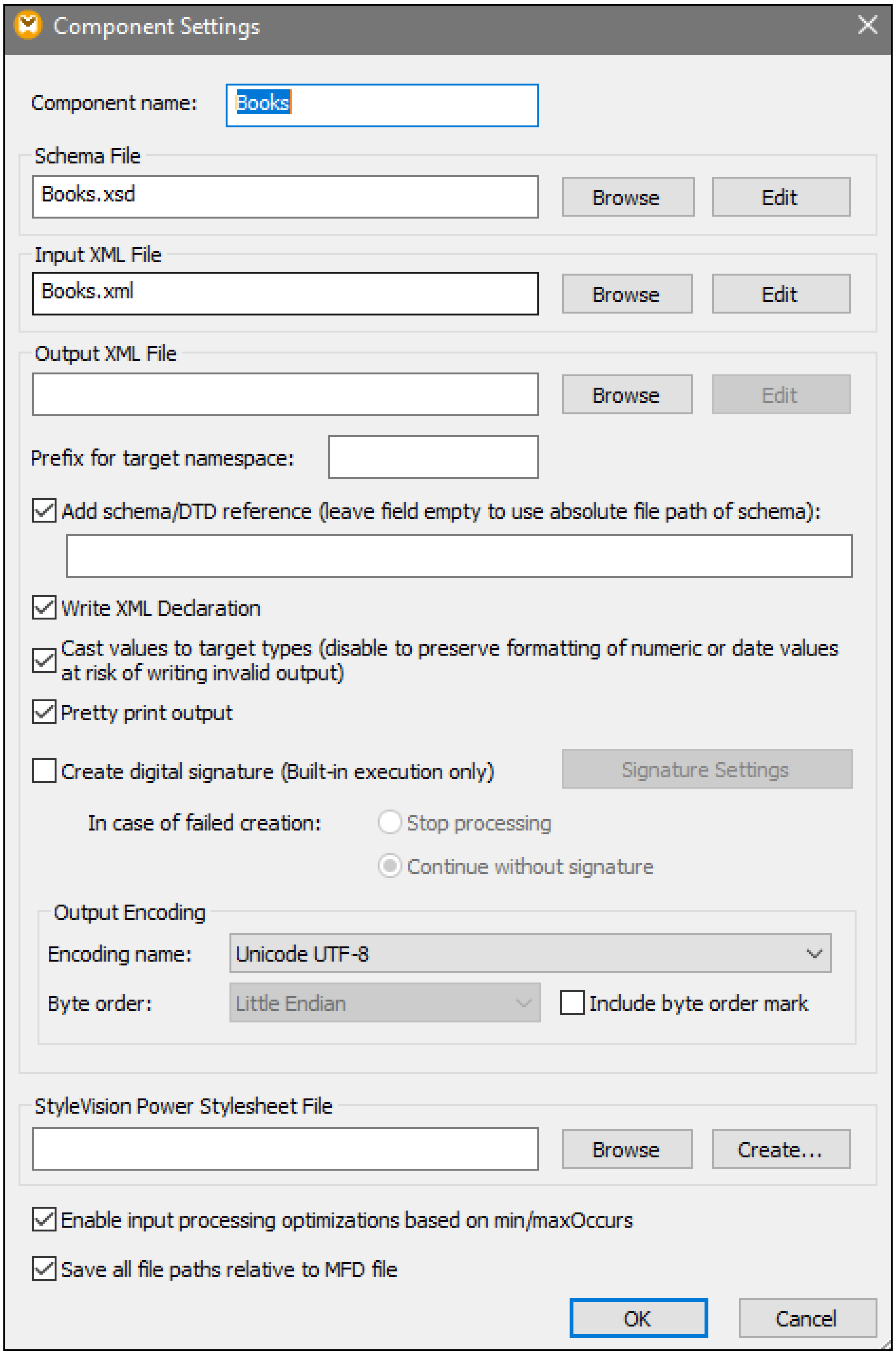
In this section

The section is organized into the following topics:

4.1.1 XML Component Settings

After you add an XML component to the mapping area, you can configure its settings in the **Component Settings** dialog box (*see screenshot below*). You can open the **Component Settings** dialog box in one of the following ways:

- By double-clicking the component header.
- By right-clicking the component header and selecting **Properties**.
- By selecting the component in the mapping and clicking **Properties** in the **Component** menu.



The available settings are described in the subsections below.

General settings

⊟ Component name

The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change the name of the component, be aware of the following:

- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.
- It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.

Schema File

Specifies the name or path of the XML schema file used by MapForce to validate and map data. To change the schema file, click **Browse** and select a new file. To edit the file in Altova XMLSpy, click **Edit**.

Input XML File

Specifies the XML instance file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign an XML instance file to this component. In a source component, the instance file name is also used to detect the XML root element and the referenced schema and to validate against the selected schema. To change the schema file, click **Browse** and select a new file. To edit the file in Altova XMLSpy, click **Edit**.

Output XML File

Specifies the XML instance file to which MapForce will write data. This field is meaningful for a target component. To change the schema file, click **Browse** and select a new file. To edit the file in Altova XMLSpy, click **Edit**.

Prefix for target namespace

Allows you to enter a prefix for the target namespace. Before assigning the prefix, make sure the target namespace is defined in the target schema.

Add schema/DTD reference

Adds the path of the referenced XML schema file to the root element of the XML output. The path of the schema entered in this field is written into the generated target instance file(s) in the `xsi:schemaLocation` attribute or into the `DOCTYPE` declaration if a DTD is used.

MapForce Professional and Enterprise editions: If you generate code in XQuery or C++, adding the DTD reference is not supported.

Entering a path in this field allows you to define where the schema file referenced by the XML instance file is to be located. This ensures that the output instance can be validated at the mapping destination when the mapping is executed. You can enter an **`http://`** address as well as an absolute or relative path in this field.

Deactivating this option allows you to disconnect the XML instance from the referenced XML schema or DTD. This may be useful, for example, if you want to send the XML output to someone who does not have access to the underlying XML schema.

Write XML declaration

By default, the option is enabled, which means that the XML declaration is written to the output. The table below shows how this feature is supported in MapForce target languages and execution engines.

Target language/Execution engine	**When output is a file**	**When output is a string**
Built-in (*Professional and Enterprise editions*)	Yes	Yes
MapForce Server (*Professional and Enterprise editions*)	Yes	Yes
XSLT, XQuery	Yes	No
Code generator (C++, C#, Java) (*Professional and Enterprise editions*)	Yes	Yes

Cast values to target types

This option allows you to define (i) whether the target XML schema types should be used in the mapping or (ii) whether all data mapped to the target component should be treated as string values. By default, this setting is enabled. Deactivating this option allows you to retain the precise formatting of values. For example, this is useful to satisfy a pattern facet in a schema that requires a specific number of decimal digits in a numeric value. You can use mapping functions to format the number as a string in the required format and then map this string to the target.

Note that disabling this option will also disable the detection of invalid values, e.g. writing letters into numeric fields.

Pretty print output

Reformats the output XML document to give it a structured look. Each child node is offset from its parent by a single tab character.

Create digital signature (*Enterprise Edition*)

Allows you to add a digital signature to the XML output instance file. Adding a digital signature is possible when you select Built-In as a transformation language (see also Digital Signatures 139).

Output Encoding

Allows you to specify the following settings of the output instance file:

- Encoding name
- Byte order
- Whether the byte order mark (BOM) character should be included

By default, any new components have the encoding defined in the *Default encoding for new components* option. You can access this option from **Tools | Options** (*General* section).

If the mapping generates XSLT 1.0/2.0, activating the *Byte Order Mark* check box does not have any effect, as these languages do not support Byte Order Marks.

StyleVision Power Stylesheet file

This option allows you to select or create an Altova StyleVision stylesheet file. Such files enable you to output data from the XML instance file to a variety of formats suitable for reporting, such as HTML, RTF, and others. See also Using Relative Paths on a Component 85.

Other settings

⊟ Enable input processing optimizations based on min/maxOccurs

This option allows special handling for sequences that are known to contain exactly one item, such as required attributes or child elements with `minOccurs` and `maxOccurs="1"`. In this case, the first item of the sequence is extracted, then the item is directly processed as an atomic value (and not as a sequence).

If the input data is **not valid** against the schema, an empty sequence might be encountered in a mapping, which stops the mapping with an error message. To allow the processing of such **invalid input**, disable this check box.

⊟ Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the **Component Settings** dialog box relative to the location of the MapForce Design (**`.mfd`**) file. See also Relative and Absolute Paths 85.

4.1.2 Derived Types

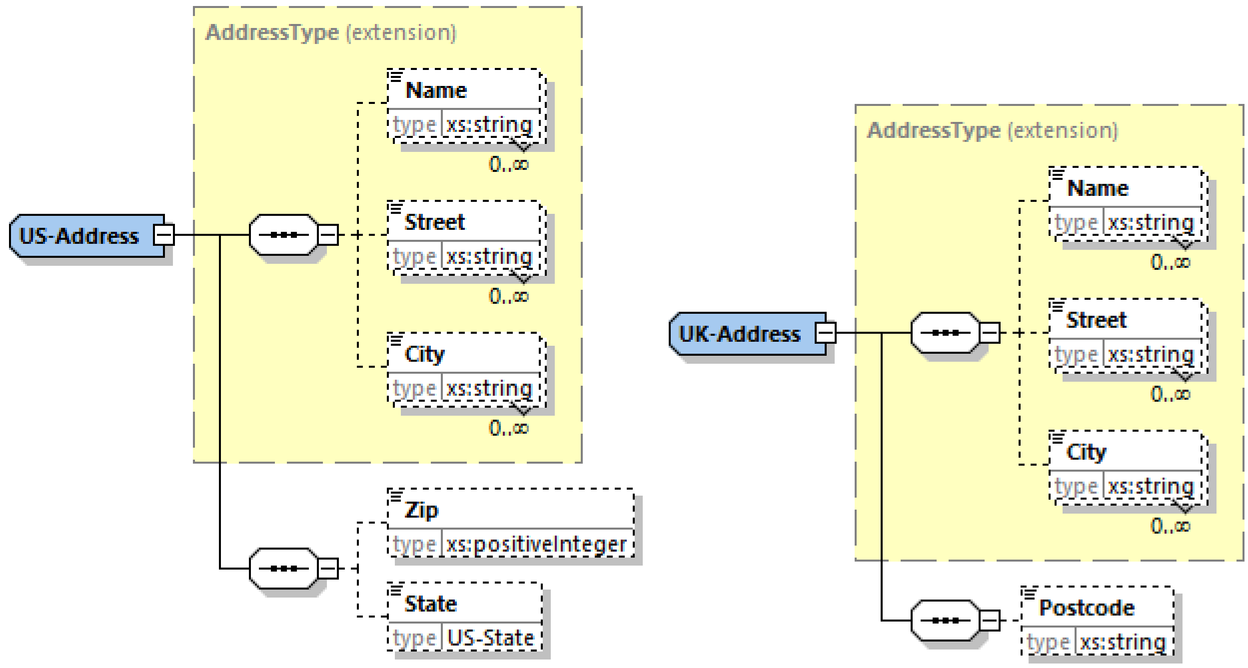
This topic explains how to use derived types in mappings. Derived types are defined in the W3C XML Schema Specification (Section 2.5.2). For a brief overview of primitive and derived types, see the Microsoft documentation. In order to use derived types in a mapping, you must specify the `xsi:type` attribute in your XML file (e.g., `<Address xsi:type="UK-Address">`).

Possible scenario

This subsection describes a possible scenario of using a derived type. For example, we have a company with two branches: one in the UK and the other in the US. Now we would like to have two lists (`UKCustomers` and `USCustomers`), each of which will include information about the respective branch address and all the customers associated with this branch.

Definition of derived types

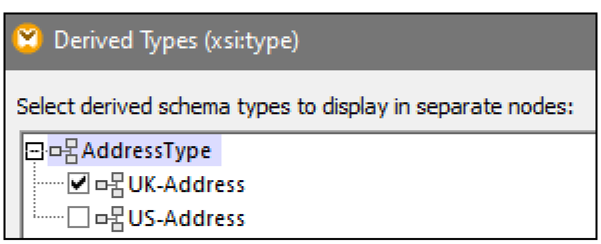
The screenshots below illustrate the definition of derived types called `US-Address` and `UK-Address` (*XMLSpy Schema view*). The `UK-Address` and `UK-Address` elements have the same base type called `AddressType` that includes the `Name`, `Street`, and `City` elements. In the `US-Address` element, the base type has been extended to include `Zip` and `State`, whereas the `UK-Address` element includes the base type and the `Postcode` element. For illustration purposes, we will map only the `UK-Address` element to the target file.



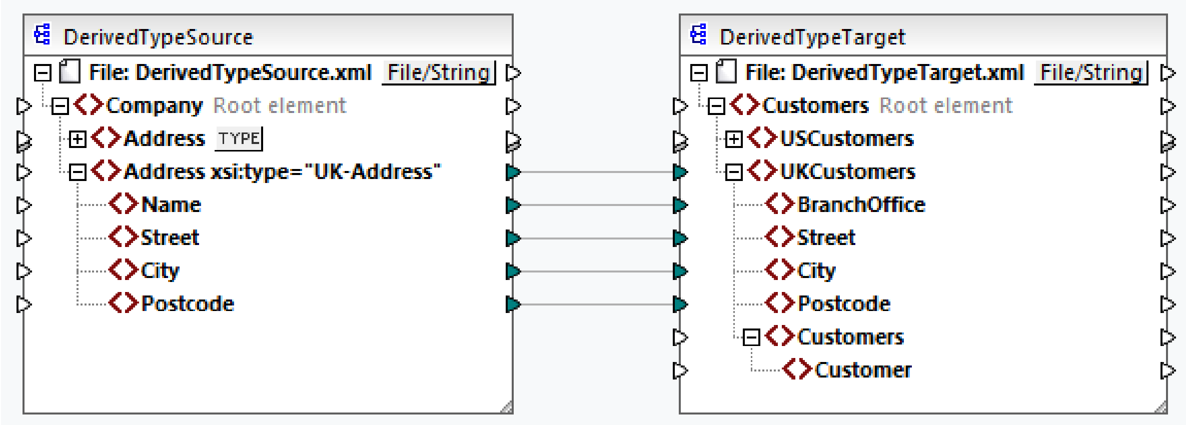
Derived type in a mapping

The instructions below show how to map data from the derived type. Our goal is to map information about the UK office to the `UKCustomers` element. The sample files are available in the **`Tutorial`** folder.

1. Go to the **Insert** menu, click **XML Schema/File**, and open **`DerivedTypeSource.xml`**. This XML file is based on **`DerivedTypeSource.xsd`**.
2. Insert the target file called **`DerivedTypeTarget.xsd`**. Note that the target schema does not have to include the `xsi:type` attribute.
3. Click the TYPE button next to the `Address` element in the source component. This button indicates that derived types exist for this element in the schema.
4. The **Derived Types** dialog box (*see screenshot below*) allows you to select any derived types available for this specific element. In our sample mapping, we want only `UK-Address` to be mapped.



5. As soon as you select the check box next to the `UK-Address` derived type, a new element called `Address xsi:type="UK-Address"` appears in the component.
6. Now connect the nodes as shown in the mapping below.



Output
Clicking the **Output** pane will show the following result:

```
<UKCustomers>
   <BranchOffice>Sleuth Corp. UK</BranchOffice>
   <Street>222 Baker St</Street>
   <City>London</City>
   <Postcode>NW1 6XE</Postcode>
</UKCustomers>
```

The sample mapping is saved as **`Tutorial\DerivedType.mfd`**. You can also add another source XML file that includes information about the customers in the UK and map this data to the `Customers` node in the target component. This way, the `UKCustomers` element will include information about the UK address and all the customers associated with this branch.

4.1.3 NULL Values

This section describes how MapForce handles NULL values in source and target components. To be able to use the `xsi:nil="true"` attribute in your XML file, you must specify the `nillable="true"` attribute for the relevant element(s) in your schema file. To find out more about the `nillable` and `xsi:nil` attributes, see the W3C Specification. Note that the `xsi:nil` attribute is not visible in a component's tree in the **Mapping** pane.

The subsections below describe some of the possible scenarios of mapping NULL values.

NULL values in XML components

This subsection discusses some of the possible scenarios of mapping elements with an `xsi:nil="true"` attribute.

Only the source element has xsi:nil="true"/Both source and target elements have xsi:nil="true"
This scenario has the following conditions:

- The connection is target-driven [92].
- The source element has an `xsi:nil="true"` attribute. The corresponding target element does not have this attribute.
- Alternatively, both source and target elements can have `xsi:nil="true"` attributes.

- The `nillable="true"` attributes must be set in the source and target schemas.
- The source and target element are of simple type.

In this case, the target element will have the `xsi:nil="true"` attribute in the output file, as shown in the sample output file below (*highlighted in yellow*).

```
<book id="7">
      <author>Edgar Allan Poe</author>
      <title>The Murders in the Rue Morgue</title>
      <category xsi:nil="true"/>
      <year>1841</year>
      <OrderID id="213"/>
</book>
```

Note: If the `nillable="true"` attribute is *not* set in the target schema, the corresponding target element will be empty in the output.

Only the target element has xsi:nil="true"
This scenario has the following conditions:

- The connection is target-driven 92.
- The source element does not have an `xsi:nil="true"` attribute.
- The corresponding target element has an `xsi:nil="true"` attribute.
- The source and target elements can be of simple or complex type.

In this case, the source element will overwrite the target element containing the `xsi:nil="true"` attribute. The example below shows a sample output file. The `<genre>` element includes the `xsi:nil="true"` attribute in the target element. However, this element has been overwritten at mapping runtime. Therefore, the `<genre>` element (*highlighted in yellow*) has `Fiction` in the output.

```
<publication>
      <id>1</id>
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <genre>Fiction</genre>
      <year>1876</year>
      <OrderID id="124"/>
</publication>
```

Complex-type source element/both complex-type elements have xsi:nil="true"
This scenario has the following conditions:

- The connection is target-driven 92.
- The source element is of complex type. In our example, the source element has an `id="213"` attribute and an `xsi:nil="true"` attribute. The corresponding target element is also of complex type and has an `id="124"` attribute, but does not have an `xsi:nil="true"` attribute.
- Alternatively, the source and target elements, both of which are of complex type, can have `xsi:nil="true"` attributes.

In this case, the source element will overwrite the target element (*highlighted in yellow below*). However, the `xsi:nil="true"` attribute will not be written to the output file automatically. To see the `xsi:nil="true"` attribute in the target element in the output file, use a copy-all 97 connection.

```
<book id="7">
```

```
            <author>Edgar Allan Poe</author>
            <title>The Murders in the Rue Morgue</title>
            <year>1841</year>
            <OrderID id="213"/>
      </book>
```

Useful functions

The following functions could help you check, replace, and assign NULL values:

- **is-xsi-nil**[1022]: Helps to check explicitly whether a source element has a `xsi:nil` attribute set to `true`.
- **substitute-missing**[1056]: Substitutes a NULL value in the source element with something specific.
- **set-xsi-nil**[1025]: Assigns `xsi:nil="true"` attribute to a target element. This works for target elements of simple and complex types.
- **substitute-missing-with-xsi-nil**[1026]: If there is content, it will be written to the target element; if there are any missing values, using this function will result in a target element with a `xsi:nil="true"` attribute in the output.
- Connecting the **exists**[1032] function to a source element with a NULL value returns `true` even though the element has no content.

Note that functions which generate `xsi:nil` cannot be passed through functions or components which only operate on values (such as the **if-else** function).

NULL values in database components

This subsection shows how NULL values are treated in database components.

Mapping NULL database fields to NULL elements
Target elements that receive NULL values from database fields are not created in the output automatically. To see such elements in the output, you need to (i) add `nillable="true"` attributes to the relevant target elements in the schema file and (ii) use the **substitute-missing-with-xsi-nil**[1026] function in the mapping. The example below shows how to handle NULL values in mappings with a source database component.

Applications table in DB Query pane
The sample mapping is located at the following path: **Tutorial\DBNullToXML.mfd**. For our example, we have chosen only one table (`Application`) from the `Accounts` database (*see below*).

	AppID	AppName	Description	Category	URL
1	1	Altova MapForce	Best data mapping tool!	IDE	https://www.altova.com/mapforce
2	2	Notepad	*[NULL]*	*[NULL]*	*[NULL]*

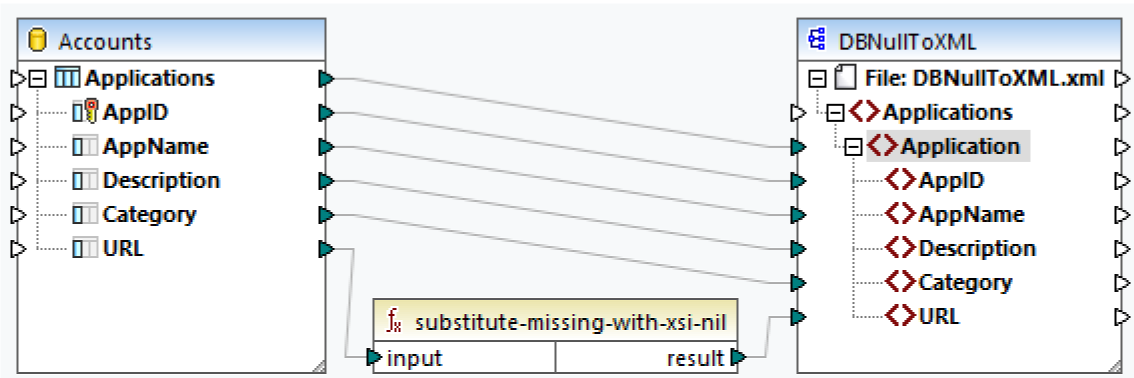
To see the `Application` table, take the steps below:

- Open the **DB Query** pane.
- Select the `Accounts` database to see its structure in the Database Browser.
- Right-click the `Application` table and click **Show in SQL Editor | SELECT**.
- Click the (**Execute Query**) button. The `Application` table will appear in the **Results** tab.

To find out more about querying databases, see Browsing and Querying Databases [376].

Mapping

The `Application` table above shows that the second record has NULL values in the `Description`, `Category`, and `URL` fields. For illustration purposes, we will map almost all the columns directly to the corresponding target elements. For the `URL` column, we will use the **`substitute-missing-with-xsi-nil`** function so that the NULL value in the target element has an `xsi:nil="true"` attribute (*see mapping below*).



Output

The output file below shows that the first record from the table has been fully written to the output, whereas the second record has been written only partly. The NULL database values are absent from the output, except for the `URL` element. Since the `URL` element has the `nillable="true"` attribute in the schema file and we are using the **`substitute-missing-with-xsi-nil`** function, the `URL` element has now the `xsi:nil="true"` attribute in the output (*highlighted yellow*).

```
<Application>
	<AppID>1</AppID>
	<AppName>Altova MapForce</AppName>
	<Description>Best data mapping tool!</Description>
	<Category>IDE</Category>
	<URL>https://www.altova.com/mapforce</URL>
</Application>
<Application>
	<AppID>2</AppID>
	<AppName>Notepad</AppName>
	<URL xsi:nil="true"/>
</Application>
```

Mapping NULL elements to NULL database fields

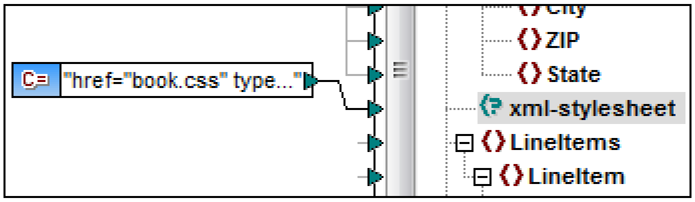
When you map a NULL XML element to a database column, MapForce writes the NULL value to the corresponding database column. You can also use the **`set-null`** [1076] function if you want to set a database field to NULL. To find out more about database-related functions, see the DB library [1075].

4.1.4 Comments and Processing Instructions

Comments and processing instructions are defined in the W3C Specification. This topic explains how to insert comments and processing instructions into target XML components. Note that comment and processing instruction nodes have only input connections. Comments and processing instructions cannot be defined for nodes that are part of a copy-all connection [97].

Insert a comment/processing instruction
To insert a processing instruction or a comment, take the steps below:

1. Right-click an element in the component and select **Add Comment/Processing Instruction Before/After**. When you insert a processing instruction, you will also need to enter its name. In the example below, a processing instruction called `xml-stylesheet` has been inserted after the `State` element.



3. To supply the value of a comment or a processing instruction, you can use a constant, for example (*see screenshot above*).

Note: Multiple processing instructions can be added before or after any element in the target component.

Note: Only one comment can be added before and after a single target node. To create multiple comments, use the duplicate input function [83].

Delete a comment/processing instruction
To delete a comment/processing instruction, right-click the respective node, select **Comment/Processing Instruction**, then select **Delete Comment/Processing Instruction** in the context menu.

4.1.5 CDATA Sections

CDATA sections are used to represent parts of a document as character data which would normally be interpreted as markup. For more information about CDATA sections, see the W3C Specification. Target nodes receiving data as CDATA sections can be any of the following: XML data, XML data embedded in database fields, and XML child elements of typed dimensions in an XBRL target. CDATA sections can also be defined on duplicate nodes and `xsi:type` nodes.

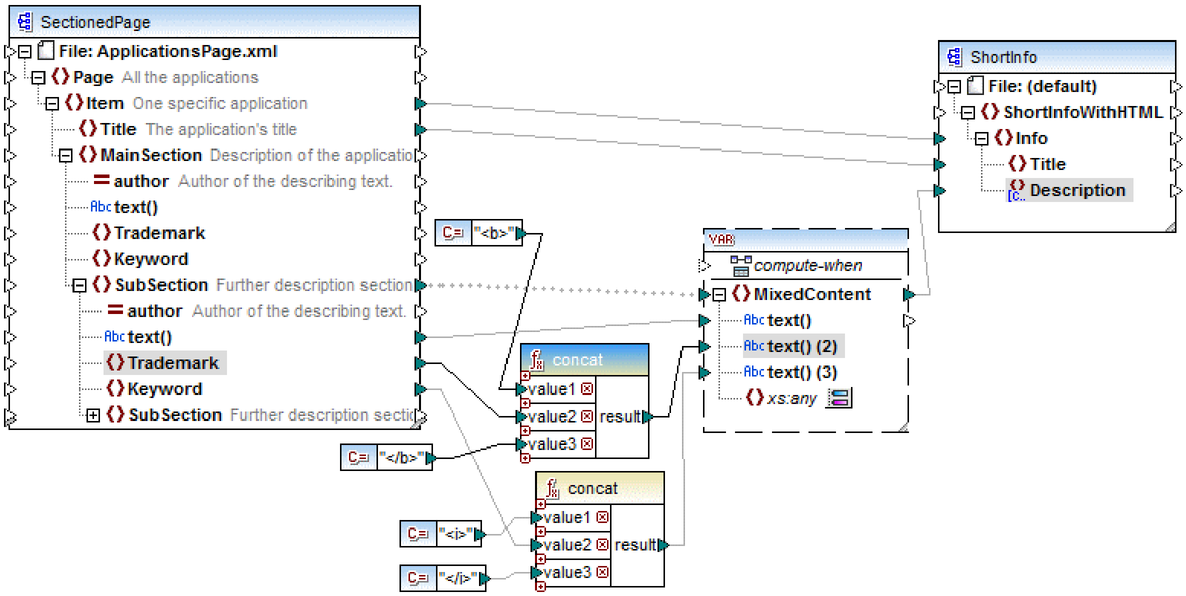
To create a CDATA section, right-click the relevant target node and select **Write Content as CDATA Section**. A prompt will warn you that the input data should not contain the CDATA section close-delimiter **`]]>`**. The **`[C..`** icon appears below the element tag, which indicates that this node is now defined as a CDATA section.

Example

The example below shows a scenario in which a CDATA section might be useful. The sample mapping called **`MapForceExamples\HTMLinCDATA.mfd`** (*see screenshot below*) has the following aspects:

- The `SubSection` element has mixed content. For more information about mixed-content nodes, see Source-Driven Connections [93].
- With the help of the **`concat`** function, the content of the `Trademark` element will have the `<b></b>` tags.
- The content of the `Keyword` element will have the `<i></i>` tags.

- The data with the new tags is passed on to the duplicate `text()` nodes in the same order as in the source document.
- The output of the `MixedContent` node is then passed on to the `Description` node in the `ShortInfo` target component. The `Description` node has been defined as a CDATA section.



Output

Click the **Output** pane to see the CDATA section in the `Description` node (*screenshot below*).

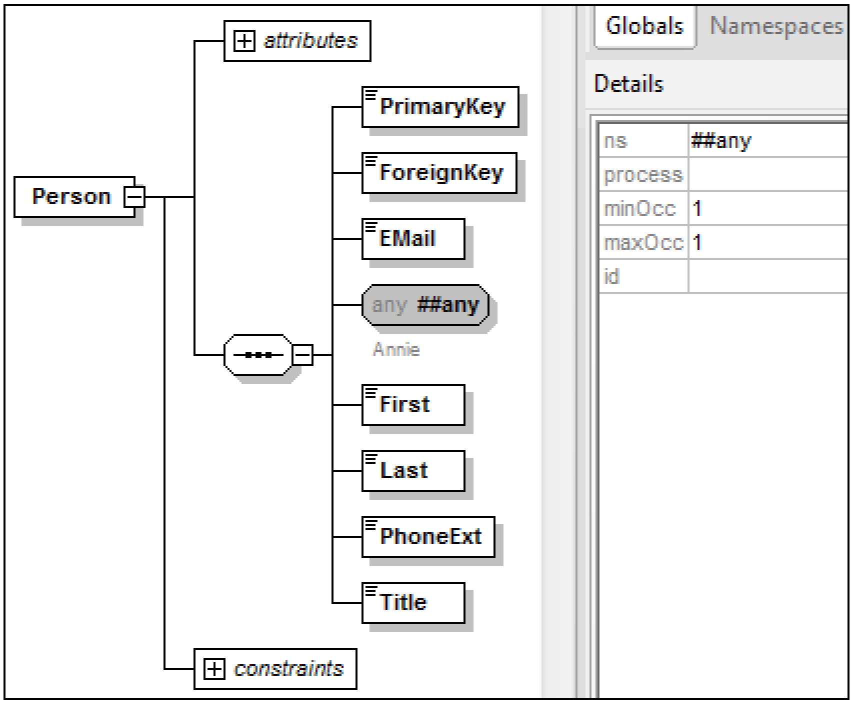
```
<Info>
    <Title>MapForce</Title>
    <Description><![CDATA[Altova <b>MapForce</b> 2014 Enterprise Edition is the premier <i>XML</i>
/ <i>database</i> / <i>flat file</i> / <i>EDI</i> data mapping tool that auto-generates mapping code in
<i>XSLT</i> 1.0/2.0, <i>XQuery</i> , <i>Java</i> , <i>C++</i> and <i>C#</i> . It is the definitive tool for
data integration and information leverage.]]></Description>
</Info>
```

4.1.6 Wildcards: xs:any/xs:anyAttribute

This topic explains how to deal with wildcards in mappings. The wildcards `xs:any` and `xs:anyAttribute` allow you to use any elements/attributes defined in your schema file. For more information about wildcards, see [the W3C Specification].

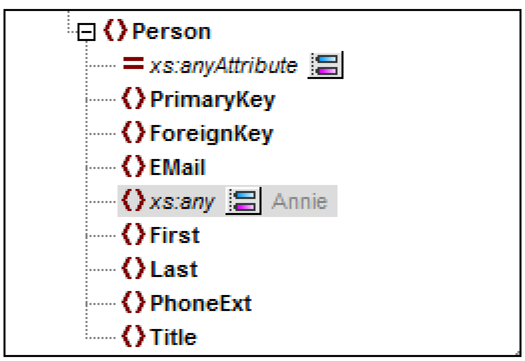
Wildcards in schema definition

The screenshot below shows that an `xs:any` element has been defined as a child element of the `Person` element (*Schema view in [Altova XMLSpy]*).



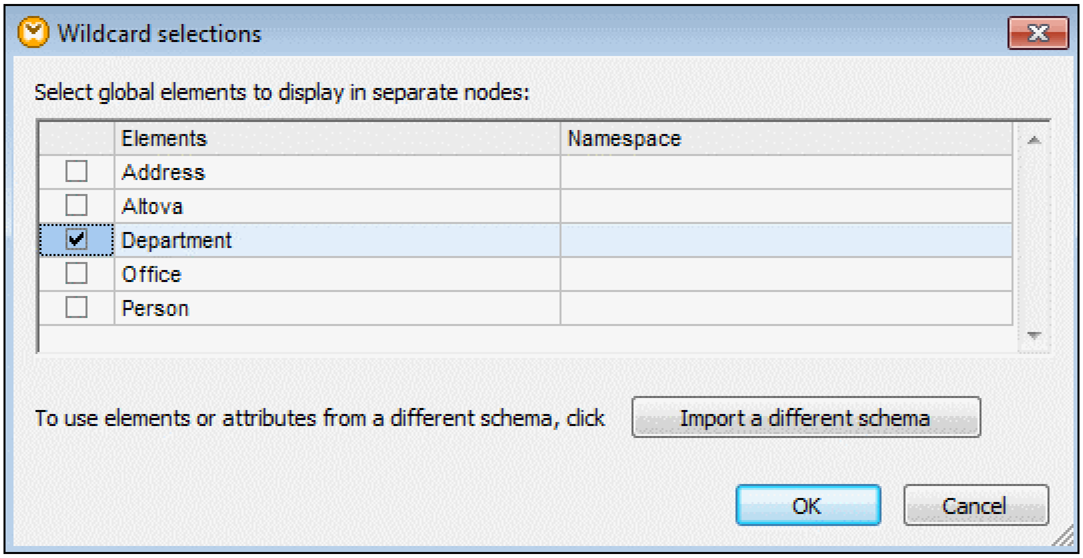
Wildcards in MapForce

When a wildcard is defined for an element and/or attribute, this wildcard node will have a (**Change Selection**) button next to it (*see screenshot below*).

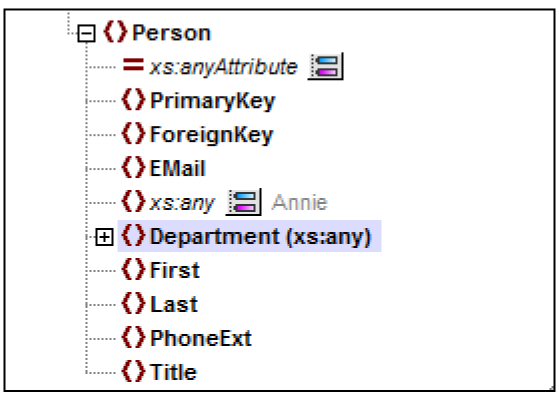


Wildcard selection

Now our goal is to add another element as a separate node. Click the button to see the list of elements that you can add to the tree. Note that only elements and attributes that are globally declared in your schema can be seen in the **Wildcard selections** dialog box (*see screenshot below*).



For our example, we have selected `Department`. Note that wildcard elements and attributes are inserted after the node with the button. Now our component looks as follows:



You can now map to/from these nodes as usual. In a component, wildcard elements and attributes are marked with (`xs:any`) and (`xs:anyAttribute`), respectively (*see screenshot above*).

Remove wildcards
To remove a wildcard node, click the button and clear the corresponding check box in the **Wildcard selections** dialog box.

Elements/attributes from a different schema

The **Wildcard selections** dialog box (*see above*) allows you to use elements/attributes from a different schema. Clicking the **Import a different schema** button will give you the following options: (i) importing a schema file and (ii) generating a wrapper schema (*see description below*).

Import schema
The **Import schema** option imports the external schema into the current schema assigned to the component. Note that this option overrides the existing schema on the disk. If the current schema is a remote schema that

has been opened from a URL (see Adding Components from a URL[80]) and not from the disk, the schema cannot be modified. In this case, use the **Generate wrapper schema** option.

Generate wrapper schema
The **Generate wrapper schema** option creates a new schema file called *wrapper schema*. The advantage of using this option is that the existing schema of the component is not modified. Instead, a new schema will be created which will include both the existing schema and the imported schema. When you select this option, you are prompted to choose where the wrapper schema should be saved. By default, the wrapper schema has the following format: **`somefile-wrapper.xsd`**.

After you save the wrapper schema, it is by default automatically assigned to the component. MapForce will also ask you whether you want to adjust the schema location so that you can reference the previous main schema. Click **Yes** to revert to the previous schema; otherwise, click **No** to have the newly created wrapper schema assigned to the component.

Wildcards vs. dynamic node names

There are situations in which elements and/or attributes in an instance are too many to be declared in the schema. Consider the following sample file:

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
   <line1>1</line1>
   <line2>2</line2>
   <line3>3</line3>
   ................
   <line999></line999>
</message>
```

For such situations, it is recommended to use dynamic node names instead of wildcards. For more information, see Mapping Node Names[1255].

4.1.7 Custom Namespaces

When a mapping produces XML output, MapForce automatically derives the namespace (or set of namespaces) of each element and attribute from the target schema. This is the default behavior that is suitable for most scenarios of XML output generation. However, there are cases in which you may want to manually declare the namespace of an element directly from the mapping.

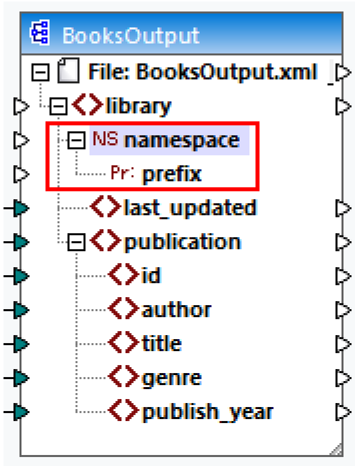
Declaring custom namespaces is meaningful only for target XML components and applies to elements only. The **Add Namespace** command is not available for attributes, wildcard nodes, and for nodes which receive data from a copy-all connection[97].

To understand how custom namespaces work, follow the instructions in the subsection below.

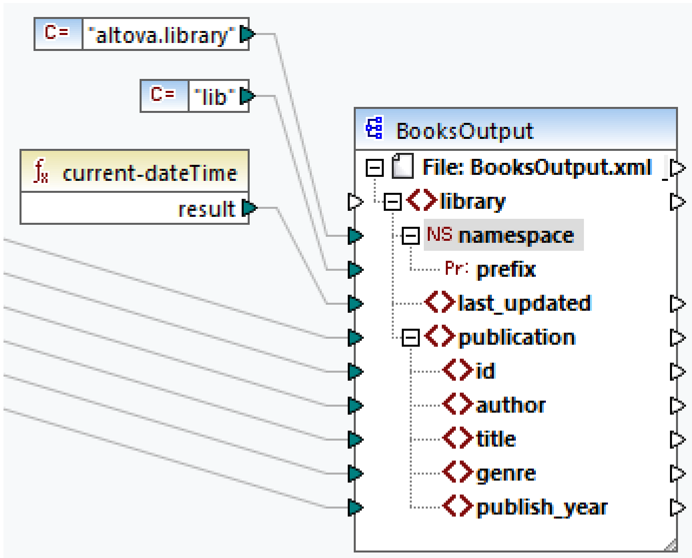
Declare namespace manually

For this example, you will need the following mapping: **`BasicTutorials\Tut1-SchemaToSchema.mfd`**.

Add a namespace
Open the mapping, right-click the `library` node in the `BooksOutput` component, and select **Add Namespace** from the context menu. Now two new nodes are available under the `library` element: `namespace` and `prefix` (*see screenshot below*).



Supply namespace values
The next step is to supply values to the `namespace` and `prefix` nodes. In order to do it, we will use two constants with the following string values: `altova.library` and `lib` (*see screenshot below*).



Note: Both the `namespace` and `prefix` input connectors must be mapped even if you provide empty values to them.

Output
In the generated output, an `xmlns:<prefix>="<namespace>"` attribute is added to the element, where `<prefix>` and `<namespace>` are values that are supplied by the mapping. The output will now look as follows (*note the highlighted part*):

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:lib="altova.library" xsi:noNamespaceSchemaLocation="Library.xsd">
...
```

You can also declare multiple namespaces for the same element, if necessary. To do this, right-click the node again and select **Add Namespace** from the context menu. A new pair of namespace and prefix nodes become available, to which you can connect new prefix and namespace values.

Declare a default namespace
If you want to declare a default namespace, map an empty string value to prefix. The output would then looks as follows (*note the highlighted part*):

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="altova.library" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="Library.xsd">
...
```

If you need to create prefixes for attribute names, for example <number prod:id="prod557">557</number>, you can achieve this by using dynamic access to a node's attributes (see Mapping Node Names 1255) or by editing the schema so that it has a prod:id attribute for <number>.

Remove a namespace
To remove a previously added namespace declaration, right-click the ns:namespace node and select **Remove Namespace** from the context menu.

4.1.8 Digital Signatures

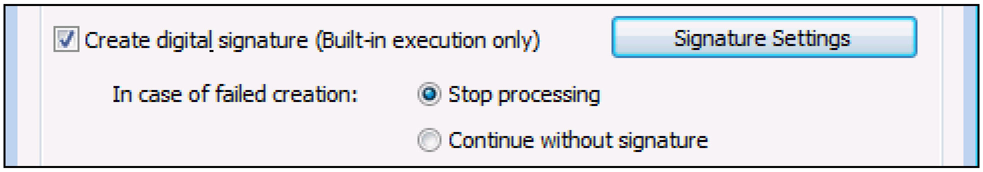
Digital signatures allow you to digitally sign an XML document with an encrypted code that can be used to verify that the XML document has not been modified. For more details about XML signatures, see the W3C Specification. MapForce supports only certificates of type RSA-SHA1 and DSA-SHA1.

MapForce supports creating XML digital signatures for XML and XBRL output files. Digital signatures are supported only in Built-In and only in the preview. Digital signatures can be embedded as the last child of the root element of the output document (*Enveloped*) or stored in a separate file (*Detached*). For more information, see XML Signature Settings 140. For a comparison between Detached and Enveloped options, see Detached vs Enveloped 143.

Note: MapForce Server does not support digital signatures.

Activate signature generation
To be able to generate digital signatures, take the following steps:

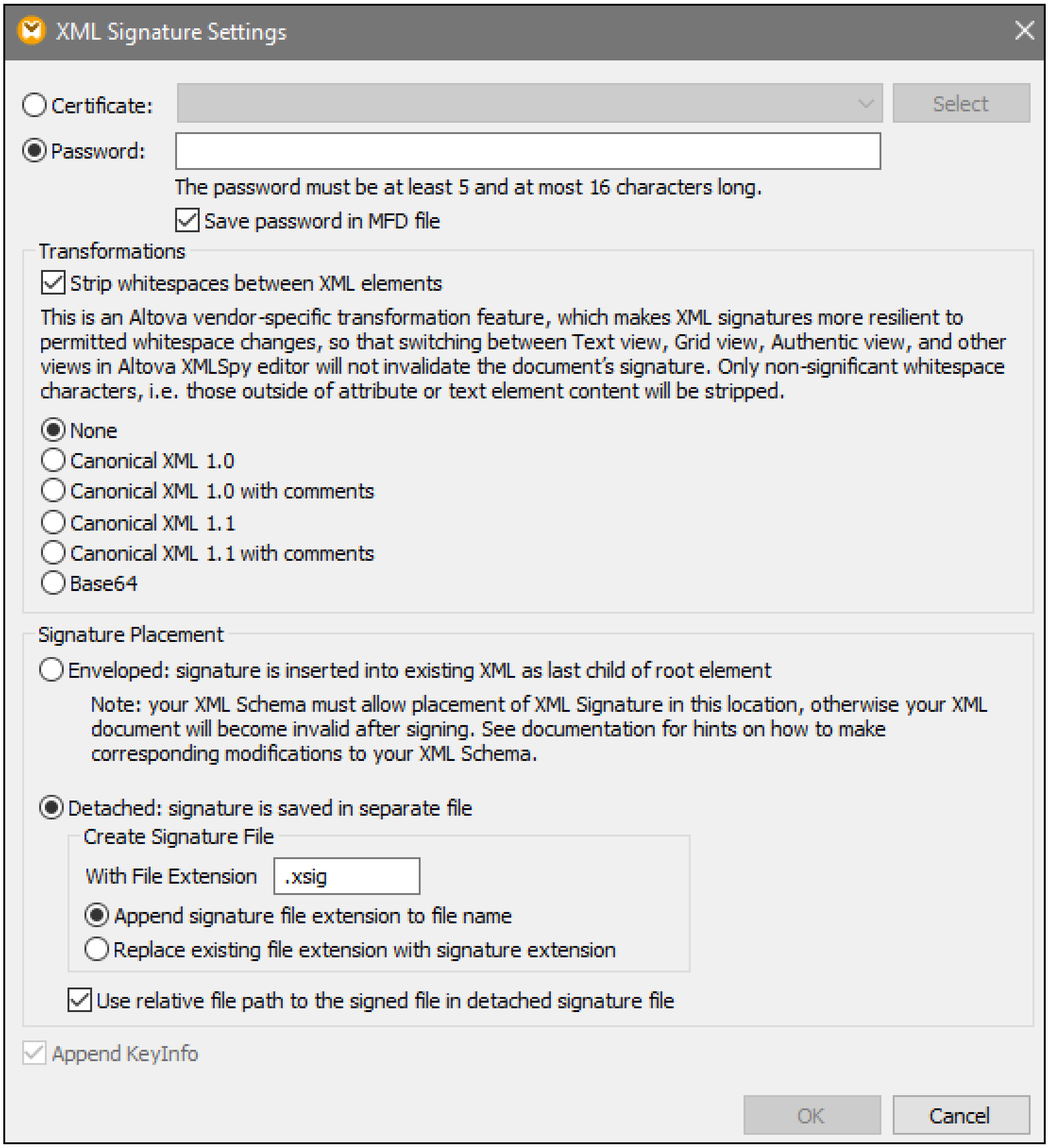
1. Open the **Component Settings** dialog box of the output component: Double-click the component's header or select **Component | Properties**.
2. Select the *Create digital signature* check box (*see screenshot below*).



3. The **XML Signature Settings** dialog box opens, in which you can define the required settings.

4.1.8.1 XML Signature Settings

Signature settings (*see screenshot below*) are stored for each component individually and can be accessed in the Component Settings [123] dialog box. All the signature settings are stored in the MFD file when it is saved.



The available settings are descried below.

Authentication method: Certificate or Password

The signature can be based on a certificate or a password.

- *Certificate:* If you wish to use a certificate, the certificate must have a private key and be located in an accessible certificate store. The signature is generated using the private key of the certificate. To verify the signature, access to the certificate (or a public-key version of it) is required. The public key of the certificate is used to verify the signature. To select the private-public-key certificate you wish to use, click the **Select** button and browse for the relevant certificate.

- *Password:* Enter a password with a length of 5 to 16 characters. This password is used to create the signature and will be required to verify the signature. The **OK** button of the dialog box becomes active only if this requirement is fulfilled.

- *Save password in MFD file:* When active, the password entered in the *Password* field is encrypted and not human-readable. Note that anyone who has access to the MFD file can create signatures using this password.

⊟ Transformations

The XML data is transformed, and the result of the transformation is used for the creation of the signature. You can specify the algorithm to be applied to the file's XML data (the `SignedInfo` content) prior to performing signature calculations. The following options are available:

- *None:* No transformation is carried out, the XML data from the binary file is saved on disk and passed directly for signature creation. Any subsequent change in the data will result in a failed verification of the signature. However, if the check box *Strip whitespaces between XML elements* is selected, all whitespaces will be stripped, and changes in whitespaces will be ignored.

- *Canonical XML with/without comments:* If comments are included for signature calculation, any change to comments in the XML data will result in verification failure. Otherwise, comments may be modified or added to the XML document after the document has been signed, and the signature will still be verified as authentic.

  Note that the *Canonical XML* option with comments is available only in *Detached* placement.

- *Base64:* The root element of the XML document is considered to be Base64-encoded. If the root element is not Base64, an error is returned or the element is read as empty, depending on the type of element.

The difference between *None* and *Canonical XML* is that the second option produces an XML data stream, in which some differences, such as attribute order, are normalized. Note, however, that a default canonicalization is performed if the signature is embedded (enveloped). The XML data will be used as is (i.e., no transformation) when the following is true: (i) the signature is detached, (ii) *None* is selected, and (iii) the *Strip whitespaces* check box is not selected.

⊟ Signature Placement

The following signature placement options are available:

- *Enveloped:* The signature element is created as the last child element of the root (document) element. For the output XML file to be valid, the associated XML schema must contain the signature definition elements.

- *Detached:* The XML signature is created as a separate file. In this case, you can specify the file extension of the signature file and whether the file name is created with (i) the extension appended to the name of the XML file (e.g., `test.xml.xsig`) or (ii) the extension replacing the XML extension of the XML file (e.g., `test.xsig`). You can also specify whether the signature file should have the reference to the XML file with a relative or an absolute path.

Note: XML signatures for XML Schema (`.xsd`) files and for XBRL files can be created only as external signature files. For WSDL files, signatures can be created as external files and can be enveloped in the WSDL file.

Note: If the XML signature is created as a separate file, the XML file and signature file are associated with each other via a reference in the signature file. Consequently, signature verification in cases where the signature is in an external fie must be done with the signature file active—not with the XML file active.

Append KeyInfo

The *Append Keyinfo* option is available only when the signature is certificate-based. If *Append KeyInfo* is active/checked, public-key information is placed inside the signature. Otherwise the key information is not included in the signature. The advantage of including key information is that the certificate itself (specifically the public-key information in it) will not be required for the verification process, because the key information is present in the signature.

Invalid signature settings

MapForce cannot digitally sign an output if the signature settings are invalid. Signature settings are invalid if (i) the selected certificate is not accessible or not suitable for signing XML documents; (ii) no password is set.

If no password or certificate is chosen, the processing stops or continues without a signature, which can be specified in the **Component Settings** dialog box. If the mapping is executed from the command line, no prompt dialog box appears. The mapping execution stops with an error or continues without signature.

4.1.8.2 Detached vs Enveloped

The examples below illustrate the usage of detached and embedded signatures.

Detached signatures

For the sample mapping below, the *Detached signature* option has been selected. Therefore, two separate files have been generated: (i) the mapping output file called **`MarketingExpenses.xml`** and (ii) a temporary digital signature file called **`MarketingExpenses.xml.xsig`** (*see screenshots below*). The mapping **`MarketingExpenses_DetachedSignature.mfd`** is available in the **`MapForceExamples`** folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<marketing-expenses xsi:noNamespaceSchemaLocation="
C:/Users/altova/Documents/Altova/MapForce2015/MapForceExamples/MarketingExpenses.xsd" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance">
  <Person>
    <FullName>Fred Landis</FullName>
    <Title>Project Manager</Title>
    <Phone>123-456-7890</Phone>
    <Email>f.landis@nanonull.com</Email>
  </Person>
  <expense-item>
    <type>Meal</type>
    <Date>2003-01-01</Date>
    <expense>122.11</expense>
  </expense-item>
  <expense-item>
    <type>Lodging</type>
    <Date>2003-01-02</Date>
    <expense>299.45</expense>
  </expense-item>
</marketing-expenses>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsig:Signature xmlns:xsig="http://www.w3.org/2000/09/xmldsig#">
  <xsig:SignedInfo>
    <xsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"></xsig:CanonicalizationMethod>
    <xsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"></xsig:SignatureMethod>
    <xsig:Reference URI="~mf92B5.tmp">
      <xsig:Transforms>
        <xsig:Transform Algorithm="http://www.altova.com/xml-signatures/strip-whitespaces"></xsig:Transform>
      </xsig:Transforms>
      <xsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></xsig:DigestMethod>
      <xsig:DigestValue>g23J8QKWoBQnz3YZbSH/19KmQEw=</xsig:DigestValue>
    </xsig:Reference>
  </xsig:SignedInfo>
  <xsig:SignatureValue>expKkRUNufQquwOr4+gcsk7bHHU=</xsig:SignatureValue>
</xsig:Signature>
```

To generate the **.xml** and **.xsig** files in the output directory, click the **Save all generated outputs** toolbar button.

Enveloped signatures
If you select the *Enveloped signature* option, the `<xsig:Signature>` element will be integrated into the output file after the `<expense-item>` (*see screenshots above*). This feature is illustrated in **MapForceExamples\MarketingExpenses_EnvelopedSignature.mfd**.

XML document validity

If an XML signature is embedded in the XML document, a `Signature` element in the namespace `http://www.w3.org/2000/09/xmldsig#` is added to the XML document. In order for the document to remain valid according to its schema, the schema must contain the appropriate element declarations. If you do not wish to modify the schema of the XML document, use the *Detached* option (*see above*).

The code listings below show how the `Signature` element of an enveloped signature can be allowed. In the first listing, the XML signature schema is imported into the user's schema (*highlighted yellow*).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
            xmlns:xsig="http://www.w3.org/2000/09/xmldsig#"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">
   <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
              schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-
schema.xsd"/>
   <xs:element name="Root">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="FirstChildOfRoot"/>
            <xs:element ref="SecondChildOfRoot" minOccurs="0"/>
            <xs:element ref="xsig:Signature" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   ...
</xs:schema>
```

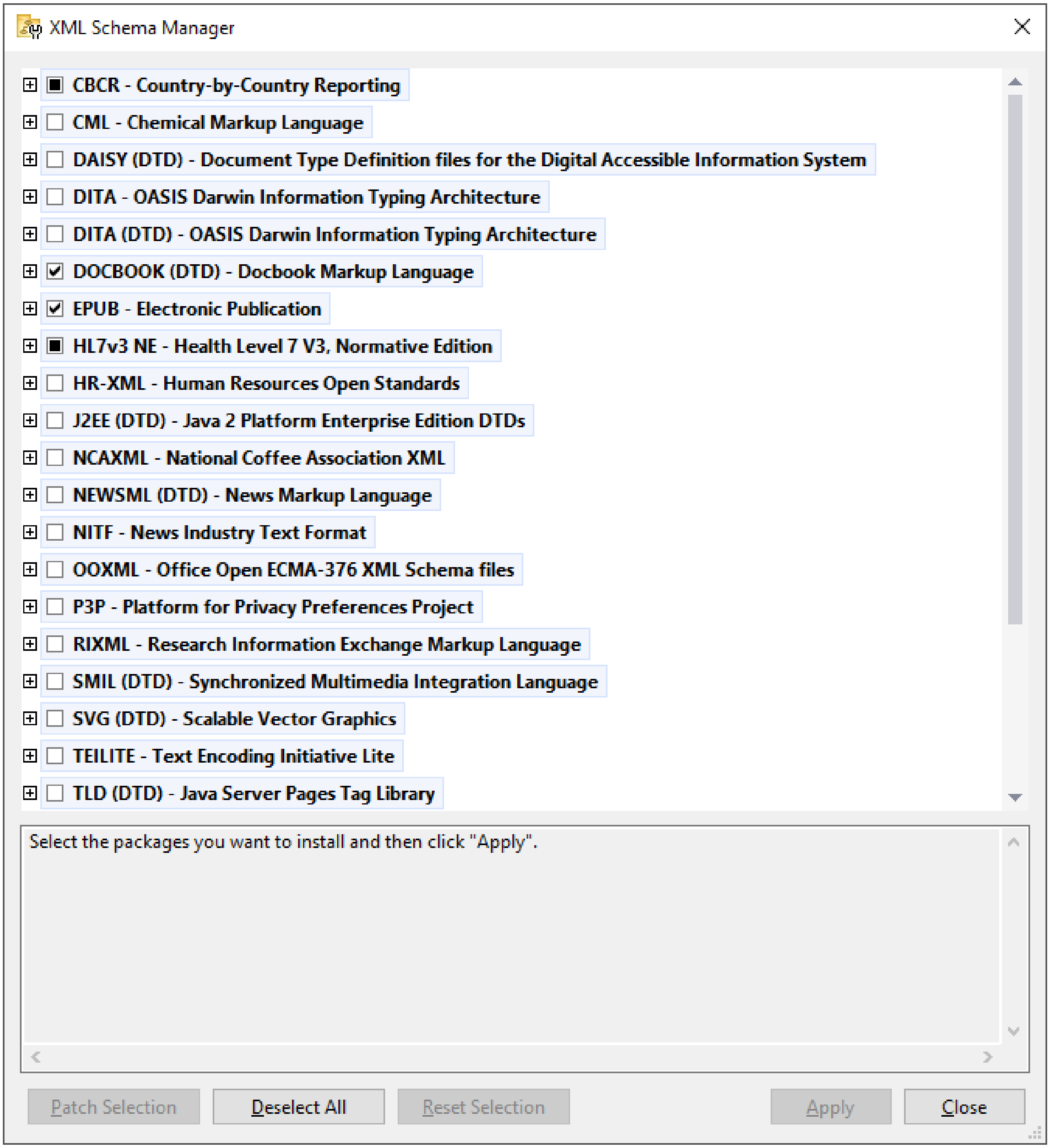
The second option is to add a generic wildcard element which matches any element from other namespaces (*highlighted yellow below*). Setting the `processContents` attribute to `lax` causes the validator to skip over this element, because no matching element declaration is found. Consequently, the user does not need to reference the XML signature schema. The drawback of this option, however, is that any element (not just the `Signature` element) can be added at the specified location in the XML document without invalidating the XML document.

```
<xs:element name="Root">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="selection"/>
         <xs:element ref="newsitems" minOccurs="0"/>
         <xs:element ref="team" minOccurs="0"/>
         <xs:any namespace="##other" minOccurs="0" processContents="lax"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

4.1.9 Schema Manager

XML Schema Manager is an Altova tool that provides a centralized way to install and manage XML schemas (DTDs for XML and XML Schemas) for use across all Altova's XBRL-enabled applications, including MapForce.

- On Windows, Schema Manager has a graphical user interface (*screenshot below*) and is also available at the command line. (Altova's desktop applications are available on Windows only; *see list below*.)
- On Linux and macOS, Schema Manager is available at the command line only. (Altova's server applications are available on Windows, Linux, and macOS; *see list below*.)



Altova applications that operate with Schema Manager

Desktop applications (Windows only)	Server applications (Windows, Linux, macOS)
XMLSpy (all editions)	RaptorXML Server, RaptorXML+XBRL Server

MapForce (all editions)	StyleVision Server
StyleVision (all editions)	
Authentic Desktop Enterprise Edition	

Installation and de-installation of Schema Manager

Schema Manager is installed automatically when you first install a new version of Altova Mission Kit Enterprise Edition or of any of Altova's XML-schema-aware applications (*see table above*).

Likewise, it is removed automatically when you uninstall the last Altova XML-schema-aware application from your computer.

Schema Manager features

Schema Manager provides the following features:

- Shows XML schemas installed on your computer and checks whether new versions are available for download.
- Downloads newer versions of XML schemas independently of the Altova product release cycle. (Altova stores schemas online, and you can download them via Schema Manager.)
- Install or uninstall any of the multiple versions of a given schema (or all versions if necessary).
- An XML schema may have dependencies on other schemas. When you install or uninstall a particular schema, Schema Manager informs you about dependent schemas and will automatically install or remove them as well.
- Schema Manager uses the XML catalog mechanism to map schema references to local files. In the case of large XML schemas, processing will therefore be faster than if the schemas were at a remote location.
- All major schemas are available via Schema Manager and are regularly updated for the latest versions. This provides you with a convenient single resource for managing all your schemas and making them readily available to all of Altova's XML-schema-aware applications.
- Changes made in Schema Manager take effect for all Altova products installed on that machine.

How it works

Altova stores all XML schemas used in Altova products online. This repository is updated when new versions of the schemas are released. Schema Manager displays information about the latest available schemas when invoked in both its GUI form as well as on the CLI. You can then install, upgrade or uninstall schemas via Schema Manager.

Schema Manager also installs schemas in one other way. At the Altova website (https://www.altova.com/schema-manager) you can select a schema and its dependent schemas that you want to install. The website will prepare a file of type **`.altova_xmlschemas`** for download that contains information about your schema selection. When you double-click this file or pass it to Schema Manager via the CLI as an argument of the `install` 156 command, Schema Manager will install the schemas you selected.

Local cache: tracking your schemas

All information about installed schemas is tracked in a centralized cache directory on your computer, located here:

Windows	`C:\ProgramData\Altova\pkgs\.cache`

Linux	`/var/opt/Altova/pkgs\.cache`
macOS	`/var/Altova/pkgs`

This cache directory is updated regularly with the latest status of schemas at Altova's online storage. These updates are carried out at the following times:

- Every time you start Schema Manager.
- When you start MapForce for the first time on a given calendar day.
- If MapForce is open for more than 24 hours, the cache is updated every 24 hours.
- You can also update the cache by running the `update` 159 command at the command line interface.

The cache therefore enables Schema Manager to continuously track your installed schemas against the schemas available online at the Altova website.

> **Do not modify the cache manually!**
> The local cache directory is maintained automatically based on the schemas you install and uninstall. It should not be altered or deleted manually. If you ever need to reset Schema Manager to its original "pristine" state, then, on the command line interface (CLI): (i) run the `reset` 157 command, and (ii) run the `initialize` 155 command. (Alternatively, run the **`reset`** command with the **`--i`** option.)

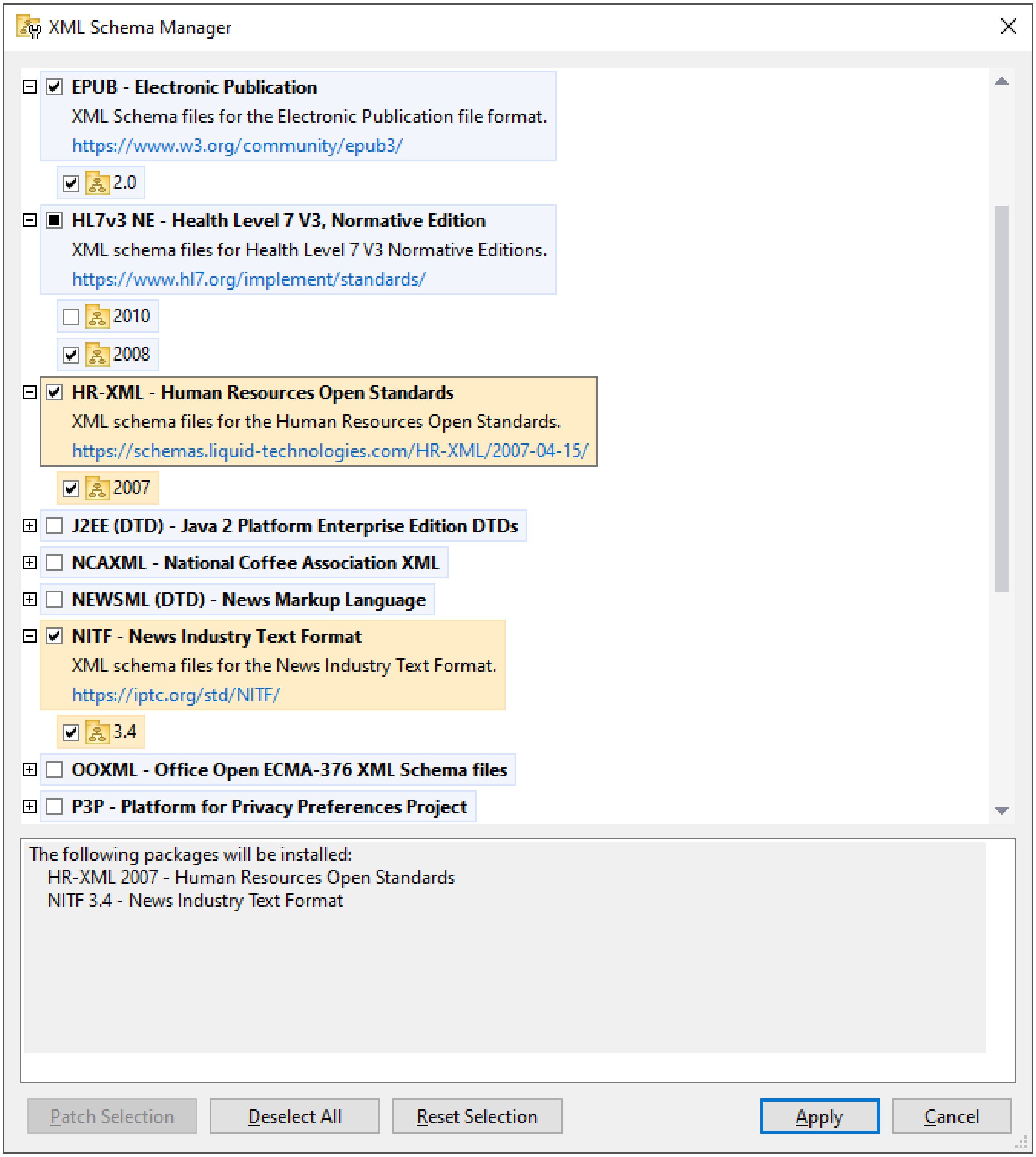
4.1.9.1 Run Schema Manager

Graphical User Interface

You can access the GUI of Schema Manager in any of the following ways:

- *During the installation of MapForce:* Towards the end of the installation procedure, select the check box *Invoke Altova XML-Schema Manager* to access the Schema Manager GUI straight away. This will enable you to install schemas during the installation process of your Altova application.
- *After the installation of MapForce:* After your application has been installed, you can access the Schema Manager GUI at any time, via the menu command **Tools | XML Schema Manager**.
- Via the **`.altova_xmlschemas`** file downloaded from the Altova website: Double-click the downloaded file to run the Schema Manager GUI, which will be set up to install the schemas you selected (at the website) for installation.

After the Schema Manager GUI (*screenshot below*) has been opened, already installed schemas will be shown selected. If you want to install an additional schema, select it. If you want to uninstall an already installed schema, deselect it. After you have made your selections and/or deselections, you are ready to apply your changes. The schemas that will be installed or uninstalled will be highlighted and a message about the upcoming changes will be posted to the Messages pane at the bottom of the Schema Manager window (*see screenshot*).



Command line interface

You can run Schema Manager from a command line interface by sending commands to its executable file, `xmlschemamanager.exe`.

The **`xmlschemamanager.exe`** file is located in the following folder:

- *On Windows:* `C:\ProgramData\Altova\SharedBetweenVersions`
- *On Linux or macOS (server applications only):* `%INSTALLDIR%/bin`, where `%INSTALLDIR%` is the program's installation directory.

You can then use any of the commands listed in the CLI command reference section 154.

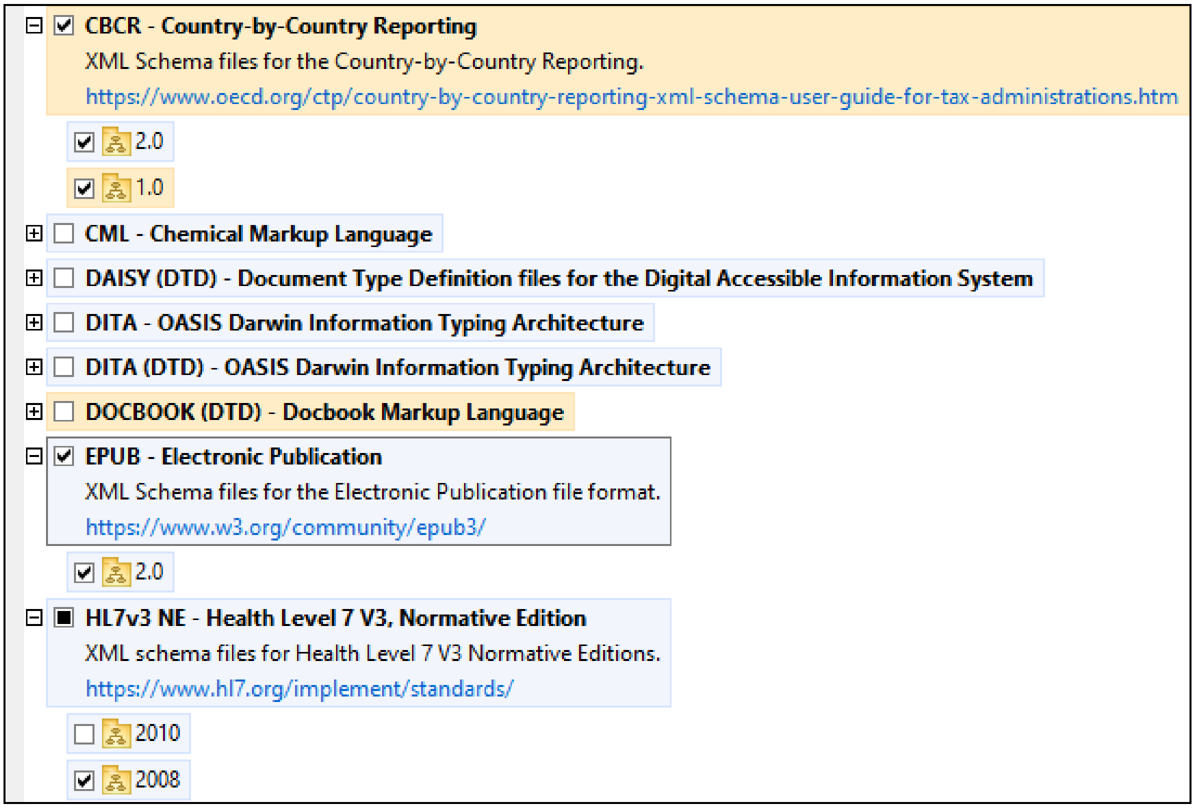
To display help for the commands, run the following:

- *On Windows:* `xmlschemamanager.exe --help`
- *On Linux or macOS (server applications only):* `sudo ./xmlschemamanager --help`

4.1.9.2 Status Categories

Schema Manager categorizes the schemas under its management as follows:

- *Installed schemas.* These are shown in the GUI with their check boxes selected (*in the screenshot below the checked and blue versions of the EPUB and HL7v3 NE schemas are installed schemas*). If all the versions of a schema are selected, then the selection mark is a tick. If at least one version is unselected, then the selection mark is a solid colored square. You can deselect an installed schema to **uninstall** it; (*in the screenshot below, the DocBook DTD is installed and has been deselected, thereby preparing it for de-installation*).
- *Uninstalled available schemas.* These are shown in the GUI with their check boxes unselected. You can select the schemas you want to **install**.



- *Upgradeable schemas* are those which have been revised by their issuers since they were installed. They are indicated in the GUI by a icon. You can **patch** an installed schema with an available revision.

Points to note

- In the screenshot above, both CBCR schemas are checked. The one with the blue background is already installed. The one with the yellow background is uninstalled and has been selected for installation. Note that the HL7v3 NE 2010 schema is not installed and has not been selected for installation.
- A yellow background means that the schema will be modified in some way when the **Apply** button is clicked. If a schema is unchecked and has a yellow background, it means that it will be uninstalled when the **Apply** button is clicked. In the screenshot above the DocBook DTD has such a status.
- When running Schema Manager from the command line, the `list` 156 command is used with different options to list different categories of schemas:

`xmlschemamanager.exe list`	Lists all installed and available schemas; upgradeables are also indicated
`xmlschemamanager.exe list -i`	Lists installed schemas only; upgradeables are also indicated
`xmlschemamanager.exe list -u`	Lists upgradeable schemas

Note: On Linux and macOS, **`use sudo ./xmlschemamanager list`**

4.1.9.3 Patch or Install a Schema

Patch an installed schema

Occasionally, XML schemas may receive patches (upgrades or revisions) from their issuers. When Schema Manager detects that patches are available, these are indicated in the schema listings of Schema Manager and you can install the patches quickly.

In the GUI

Patches are indicated by the icon. (*Also see the previous topic about status categories*[150].) If patches are available, the **Patch Selection** button will be enabled. Click it to select and prepare all patches for installation. In the GUI, the icon of each schema that will be patched changes from to , and the Messages pane at the bottom of the dialog lists the patches that will be applied. When you are ready to install the selected patches, click **Apply**. All patches will be applied together. Note that if you deselect a schema marked for patching, you will actually be uninstalling that schema.

On the CLI

To apply a patch at the command line interface:

1. Run the `list -u`[156] command. This lists any schemas for which upgrades are available.
2. Run the `upgrade`[159] command to install all the patches.

Install an available schema

You can install schemas using either the Schema Manager GUI or by sending Schema Manager the install instructions via the command line.

Note: If the current schema references other schemas, the referenced schemas are also installed.

In the GUI

To install schemas using the Schema Manager GUI, select the schemas you want to install and click **Apply**.

You can also select the schemas you want to install at the Altova website and generate a downloadable **`.altova_xmlschemas`** file. When you double-click this file, it will open Schema Manager with the schemas you wanted pre-selected. All you will now have to do is click **Apply**.

On the CLI

To install schemas via the command line, run the `install`[156] command:

```
xmlschemamanager.exe install [options] Schema+
```

where **`Schema`** is the schema (or schemas) you want to install or a **`.altova_xmlschemas`** file. A schema is referenced by an identifier of format **`<name>-<version>`**. (The identifiers of schemas are displayed when you run the `list`[156] command.) You can enter as many schemas as you like. For details, see the description of the `install`[156] command.

Note: On Linux or macOS, use the **`sudo ./xmlschemamanager`** command.

Installing a required schema
When you run an XML-schema-related command in MapForce and MapForce discovers that a schema it needs for executing the command is not present or is incomplete, Schema Manager will display information about the missing schema/s. You can then directly install any missing schema via Schema Manager.

In the Schema Manager GUI, you can view all previously installed schemas at any time by running Schema Manager from **Tools | Schema Manager**.

4.1.9.4 Uninstall a Schema, Reset

Uninstall a schema

You can uninstall schemas using either the Schema Manager GUI or by sending Schema Manager the uninstall instructions via the command line.

Note: If the schema you want to uninstall references other schemas, then the referenced schemas are also uninstalled.

In the GUI
To uninstall schemas in the Schema Manager GUI, clear their check boxes and click **Apply**. The selected schemas and their referenced schemas will be uninstalled.

To uninstall all schemas, click **Deselect All** and click **Apply**.

On the CLI
To uninstall schemas via the command line, run the `uninstall` 158 command:

```
xmlschemamanager.exe uninstall [options] Schema+
```

where each **`Schema`** argument is a schema you want to uninstall or a **`.altova_xmlschemas`** file. A schema is specified by an identifier that has a format of **`<name>-<version>`**. (The identifiers of schemas are displayed when you run the `list` 156 command.) You can enter as many schemas as you like. For details, see the description of the `uninstall` 158 command.

Note: On Linux or macOS, use the **`sudo ./xmlschemamanager`** command.

Reset Schema Manager

You can reset Schema Manager. This removes all installed schemas and the cache directory.

- In the GUI, click **Reset Selection**.
- On the CLI, run the `reset` 157 command.

After running this command, make sure to run the `initialize`[155] command in order to recreate the cache directory. Alternatively, run the `reset`[157] command with the `-i` option.

Note that `reset -i`[157] restores the original installation of the product, so it is recommended to run the `update`[159] command after performing a reset. Alternatively, run the `reset`[157] command with the `-i` and `-u` options.

4.1.9.5 Command Line Interface (CLI)

To call Schema Manager at the command line, you need to know the path of the executable. By default, the Schema Manager executable is installed here:

```
C:\ProgramData\Altova\SharedBetweenVersions\XMLSchemaManager.exe
```

Note: On Linux and macOS systems, once you have changed the directory to that containing the executable, you can call the executable with **`sudo ./xmlschemamanager`**. The prefix **`./`** indicates that the executable is in the current directory. The prefix **`sudo`** indicates that the command must be run with root privileges.

Command line syntax

The general syntax for using the command line is as follows:

```
<exec> -h | --help | --version | <command> [options] [arguments]
```

In the listing above, the vertical bar **`|`** separates a set of mutually exclusive items. The square brackets **`[]`** indicate optional items. Essentially, you can type the executable path followed by either `--h`, `--help`, or `--version` options, or by a command. Each command may have options and arguments. The list of commands is described in the following sections.

4.1.9.5.1 help

This command provides contextual help about commands pertaining to Schema Manager executable.

Syntax

```
<exec> help [command]
```

Where `[command]` is an optional argument which specifies any valid command name.

Note the following:

- You can invoke help for a command by typing the command followed by **`-h`** or **`--help`**, for example: **`<exec> list -h`**
- If you type **`-h`** or **`--help`** directly after the executable and before a command, you will get general help (not help for the command), for example: **`<exec> -h list`**

Example

The following command displays help about the `list` command:

```
xmlschemamanager help list
```

4.1.9.5.2 info

This command displays detailed information for each of the schemas supplied as a `Schema` argument. This information for each submitted schema includes the title, version, description, publisher, and any referenced schemas, as well as whether the schema has been installed or not.

Syntax

```
<exec> info [options] Schema+
```

- The **`Schema`** argument is the name of a schema or a part of a schema's name. (To display a schema's package ID and detailed information about its installation status, you should use the `list` 156 command.)
- Use **`<exec> info -h`** to display help for the command.

Example

The following command displays information about the latest **`DocBook-DTD`** and **`NITF`** schemas:

```
xmlschemamanager info doc nitf
```

4.1.9.5.3 initialize

This command initializes the Schema Manager environment. It creates a cache directory where information about all schemas is stored. Initialization is performed automatically the first time a schema-cognizant Altova application is installed. You would not need to run this command under normal circumstances, but you would typically need to run it after executing the **`reset`** command.

Syntax

```
<exec> initialize | init [options]
```

Options

The **`initialize`** command takes the following options:

`--silent, --s`	Display only error messages. The default is **`false`**.
`--verbose, --v`	Display detailed information during execution. The default is **`false`**.
`--help, --h`	Display help for the command.

Example

The following command initializes Schema Manager:

```
xmlschemamanager initialize
```

4.1.9.5.4 install

This command installs one or more schemas.

Syntax

```
<exec> install [options] Schema+
```

To install multiple schemas, add the **Schema** argument multiple times.

The **Schema** argument is one of the following:

- A schema identifier (having a format of **<name>-<version>**, for example: **cbcr-2.0**). To find out the schema identifiers of the schemas you want, run the list[156] command. You can also use an abbreviated identifier if it is unique, for example **docbook**. If you use an abbreviated identifier, then the latest version of that schema will be installed.
- The path to a **.altova_xmlschemas** file downloaded from the Altova website. For information about these files, see *Introduction to SchemaManager: How It Works*[145].

Options

The **install** command takes the following options:

--silent, --s	Display only error messages. The default is **false**.
--verbose, --v	Display detailed information during execution. The default is **false**.
--help, --h	Display help for the command.

Example

The following command installs the CBCR 2.0 (Country-By-Country Reporting) schema and the latest DocBook DTD:

```
xmlschemamanager install cbcr-2.0 docbook
```

4.1.9.5.5 list

This command lists schemas under the management of Schema Manager. The list displays one of the following

- All available schemas
- Schemas containing in their name the string submitted as a **Schema** argument
- Only installed schemas
- Only schemas that can be upgraded

Syntax

```
<exec> list | ls [options] Schema?
```

If no **Schema** argument is submitted, then all available schemas are listed. Otherwise, schemas are listed as specified by the submitted options (*see example below*). Note that you can submit the **Schema** argument multiple times.

Options
The **list** command takes the following options:

`--installed, --i`	List only installed schemas. The default is **false**.
`--upgradeable, --u`	List only schemas where upgrades (patches) are available. The default is **false**.
`--help, --h`	Display help for the command.

Examples

- To list all available schemas, run: **xmlschemamanager list**
- To list installed schemas only, run: **xmlschemamanager list -i**
- To list schemas that contain either "doc" or "nitf" in their name, run: **xmlschemamanager list doc nitf**

4.1.9.5.6 reset

This command removes all installed schemas and the cache directory. You will be completely resetting your schema environment. After running this command, be sure to run the initialize 155 command to recreate the cache directory. Alternatively, run the **reset** command with the **-i** option. Since **reset -i** restores the original installation of the product, we recommend that you run the update 159 command after performing a reset and initialization. Alternatively, run the **reset** command with both the **-i** and **-u** options.

Syntax

```
<exec> reset [options]
```

Options
The **reset** command takes the following options:

`--init, --i`	Initialize Schema Manager after reset. The default is **false**.
`--update, --u`	Updates the list of available schemas in the cache. The default is **false**.

`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.
`--help, --h`	Display help for the command.

Examples

- To reset Schema Manager, run: **xmlschemamanager reset**
- To reset Schema Manager and initialize it, run: **xmlschemamanager reset -i**
- To reset Schema Manager, initialize it,and update its schema list, run: **xmlschemamanager reset -i -u**

4.1.9.5.7 uninstall

This command uninstalls one or more schemas. By default, any schemas referenced by the current one are uninstalled as well. To uninstall just the current schema and keep the referenced schemas, set the option `--k`.

Syntax

```
<exec> uninstall [options] Schema+
```

To uninstall multiple schemas, add the **Schema** argument multiple times.

The **Schema** argument is one of the following:

- A schema identifier (having a format of **<name>-<version>**, for example: **cbcr-2.0**). To find out the schema identifiers of the schemas that are installed, run the `list -i` 156 command. You can also use an abbreviated schema name if it is unique, for example **docbook**. If you use an abbreviated name, then all schemas that contain the abbreviation in its name will be uninstalled.
- The path to a **.altova_xmlschemas** file downloaded from the Altova website. For information about these files, see *Introduction to SchemaManager: How It Works* 145.

Options
The **uninstall** command takes the following options:

`--keep-references, --k`	Set this option to keep referenced schemas. The default is **false**.
`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.
`--help, --h`	Display help for the command.

Example

The following command uninstalls the CBCR 2.0 and EPUB 2.0 schemas and their dependencies:

```
xmlschemamanager uninstall cbcr-2.0 epub-2.0
```

The following command uninstalls the **eba-2.10** schema but not the schemas it references:

```
xmlschemamanager uninstall --k cbcr-2.0
```

4.1.9.5.8 update

This command queries the list of schemas available from the online storage and updates the local cache directory. You should not need to run this command unless you have performed a [reset](157) and [initialize](155).

Syntax

```
<exec> update [options]
```

Options

The **update** command takes the following options:

`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.
`--help, --h`	Display help for the command.

Example

The following command updates the local cache with the list of latest schemas:

```
xmlschemamanager update
```

4.1.9.5.9 upgrade

This command upgrades all installed schemas that can be upgraded to the latest available *patched* version. You can identify upgradeable schemas by running the [list -u](156) command.

Note: The **upgrade** command removes a deprecated schema if no newer version is available.

Syntax

```
<exec> upgrade [options]
```

Options

The **upgrade** command takes the following options:

`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.

`--help, --h`	Display help for the command.

4.2 Databases

Altova website: Database mapping

MapForce 2023 provides powerful support for mapping databases to XML, flat files, and other database formats. With MapForce Enterprise edition, you can additionally map databases to EDI formats, Excel 2007+, JSON, XBRL, and Web services.

The table below lists all the supported databases. While Altova endeavors to support other databases, successful connection and data processing have only been tested with the databases listed below. If your Altova application is a 64-bit version, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

Database	Notes
Apache CouchDB 3.x	NoSQL database. For details, see the CouchDB documentation. To find out how NoSQL databases work in MapForce, see NoSQL Databases 437.
Firebird 2.x, 3.x	
IBM DB2 8.x, 9.x, 10.x, 11.x	
IBM Db2 for i 6.x, 7.4	Logical files are supported and shown as views.
IBM Informix 11.70 and later	Informix supports connections via ADO, JDBC and ODBC. The implementation does not support large object data types in any of the code generation languages. MapForce will generate an error message (during code generation) if any of these data types are used.
MariaDB 10 and later	
Microsoft Access 2003 and later	At the time of writing (early September 2019), there is no Microsoft Access Runtime available for Access 2019. You can connect to an Access 2019 database from Altova products only if Microsoft Access 2016 Runtime is installed and only if the database does not use the "Large Number" data type.
Microsoft Azure SQL Database	SQL Server 2016 codebase
Microsoft SQL Server 2005 and later Microsoft SQL Server on Linux	
MongoDB 4.x	NoSQL database. For more details, see the MongoDB documentation. To find out how NoSQL databases work in MapForce, see NoSQL Databases 437.
MySQL 5 and later	
Oracle 9i and later	
PostgreSQL 8 and later	PostgreSQL connections are supported both as native

Database	Notes
	connections and driver-based connections through interfaces (drivers) such as ODBC or JDBC. Native connections do not require any drivers.
Progress OpenEdge 11.6	
SQLite 3.x	SQLite connections are supported as native, direct connections to the SQLite database file. No separate drivers are required.
Sybase ASE 15, 16	
Teradata 16	Connections are supported through ADO.NET, JDBC, and ODBC. When a mapping inserts data into a database table, database-generated identity fields are not supported.

Database mappings in various execution environments

When you generate program code from a mapping, or when you compile a mapping to MapForce Server execution files, or when you deploy a mapping to FlowForce Server, the database connection details saved with the generated files are adapted to drivers applicable or supported for the chosen target environment, as shown in the table below. For example, if the mapping transformation language is set to Java, ADO connections are converted to JDBC when Java code is generated from the mapping.

When the mapping is executed in an environment other than MapForce, you will need to make sure that the database connection details are meaningful on the machine which executes the mapping (for example, the database driver is installed, the database path is correct, the database server is accessible, etc.).

Some database connection types are not supported in some target environments, as shown in the table below.

Connection type/Execution Environment	C#	C++	Java	MapForce Server on Windows	MapForce Server on Linux/Mac
ADO	ADO bridge	As is	Converted to JDBC	As is	Converted to JDBC
ADO.NET	As is	User defined	Converted to JDBC	As is	Converted to JDBC
JDBC	User defined	User defined	As is	As is	As is
ODBC	ODBC bridge	ODBC bridge	Converted to JDBC	As is	Converted to JDBC
Native PostgreSQL	Not supported	Not supported	Not supported	As is	As is

Connection type/Execution Environment	C#	C++	Java	MapForce Server on Windows	MapForce Server on Linux/Mac
Native SQLite	Not supported	Not supported	Not supported	As is	As is

Table legend:

- "As is" means that the database connection type (for example, JDBC, ODBC) remains as defined in MapForce.
- "Converted to JDBC" means that the database connection will be converted into a JDBC-like database connection URL.
- "ADO bridge" or "ODBC bridge" means that the connection string remains as defined in MapForce, but the generated code will use a suitable class which acts as an ADO bridge (or ODBC bridge, respectively), for example, **System.Data.OleDb.OleDbConnection** or **System.Data.Odbc.OdbcConnection**.
- "User defined" means that, in order for the connection to work in generated code, you will need to manually enter the connection details into the Database Component Settings 279 dialog box. Depending on the case, these connection details must be entered under **ADO/OLE-DB-specific settings** or under **JDBC-specific settings**.

See also:

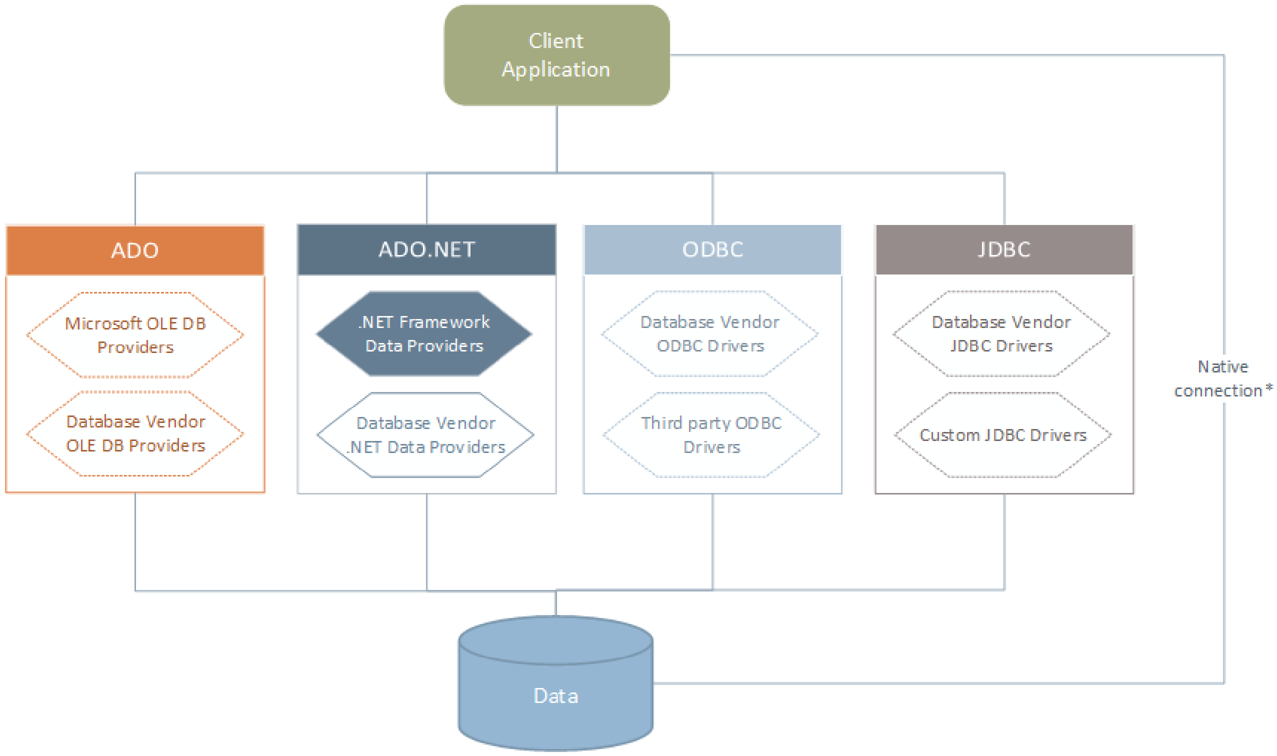
- Preparing Mappings for Server Execution 1372
- Compiling Mappings to MapForce Server Execution Files 1378
- Deploying Mappings to FlowForce Server 1381
- Code Generator 1492

4.2.1 Connecting to a Data Source

In the most simple case, a database can be a local file such as a Microsoft Access or SQLite database file. In a more advanced scenario, a database may reside on a remote or network database server which does not necessarily use the same operating system as the application that connects to it and consumes data. For example, while MapForce runs on a Windows operating system, the database from which you want to access data (for example, MySQL) might run on a Linux machine.

To interact with various database types, both remote and local, MapForce relies on the data connection interfaces and database drivers that are already available on your operating system or released periodically by the major database vendors. In the constantly evolving landscape of database technologies, this approach caters for better cross-platform flexibility and interoperability.

The following diagram illustrates, in a simplified way, data connectivity options available between MapForce (illustrated as a generic client application) and a data store (which may be a database server or database file).



** Direct native connections are supported for SQLite, PostgreSQL, CouchDB and MongoDB databases. To connect to such databases, no additional drivers are required to be installed on your system.*

As shown in the diagram above, MapForce can access any of the major database types through the following data access technologies:

- ADO (Microsoft® ActiveX® Data Objects), which, in its turn, uses an underlying OLE DB (Object Linking and Embedding, Database) provider
- ADO.NET (A set of libraries available in the Microsoft .NET Framework that enable interaction with data)
- JDBC (Java Database Connectivity)
- ODBC (Open Database Connectivity)

Note: Some ADO.NET providers are not supported or have limited support. See ADO.NET Support Notes 181.

About data access technologies

The data connection interface you should choose largely depends on your existing software infrastructure. You will typically choose the data access technology and the database driver which integrates tighter with the database system to which you want to connect. For example, to connect to a Microsoft Access 2013 database, you would build an ADO connection string that uses a native provider such as the **Microsoft Office Access Database Engine OLE DB Provider**. To connect to Oracle, on the other hand, you may want to download and install the latest JDBC, ODBC, or ADO.NET interfaces from the Oracle website.

While drivers for Windows products (such as Microsoft Access or SQL Server) may already be available on your Windows operating system, they may not be available for other database types. Major database vendors routinely release publicly available database client software and drivers which provide cross-platform access to the respective database through any combination of ADO, ADO.NET, ODBC, or JDBC. In addition to this, several third party drivers may be available for any of the above technologies. In most cases, there is more than one way to connect to the required database from your operating system, and, consequently, from MapForce. The available features, performance parameters, and the known issues will typically vary based on the data access technology or drivers used.

4.2.1.1 Start Database Connection Wizard

MapForce provides a Database Copnnection Wizard that guides you through the steps required to set up a connection to a data source. Before you go through the wizard steps, be aware that for some database types it is necessary to install and separately configure several database prerequisites, such as a database driver or database client software. These are normally provided by the respective database vendors, and include documentation tailored to your specific Windows version. For a list of database drivers grouped by database type, see Database Drivers Overview 167.

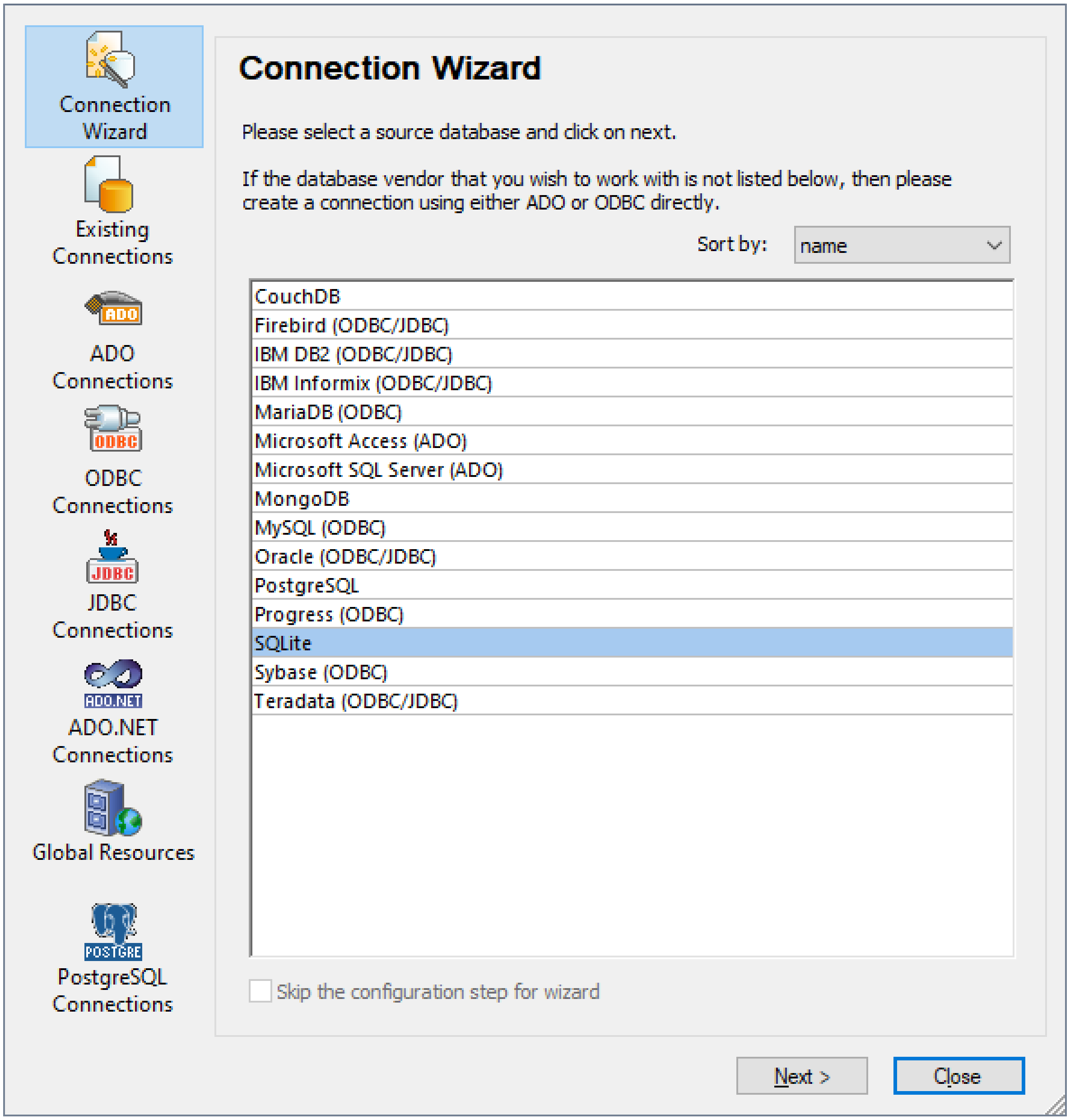
To start the Database Connection Wizard (*see screenshot below*), do the following:

- On the **Insert** menu, click **Database**.

The Database Connection Wizard (*screenshot below*) is started. On the left hand side of the window, you can select the most suitable from the following ways to connect to your database:

- Connection Wizard, which prompts you to choose your database type and then guides you through the steps for connecting to a database of that type
- Select an existing connection
- Select a data access technology: ADO, ADO.NET, ODBC, or JDBC
- Use an Altova global resource in which database connection is stored
- A native PostgreSQL connection

In the Connection Wizard pane (*see screenshot below*) databases can be sorted alphabetically by the name of the database type or by recent usage. Select the option you want in the *Sort By* combo box. After you have selected the database type to which you want to connect, click **Next**.



The wizard will take you through the next steps according to the database type, connection technology (ADO, ADO.NET, ODBC, JDBC), and driver that will be used. For examples applicable to each database type, see Database Connection Examples 195.

Alternatively to using Connection Wizard, you can use one of the following database access technologies:

- Setting up an ADO Connection 170
- Setting up an ADO.NET Connection 175
- Setting up an ODBC Connection 182
- Setting up a JDBC Connection 185

4.2.1.2 Database Drivers Overview

The following table lists common database drivers you can use to connect to a particular database through a particular data access technology. Note that this list does not aim to be either exhaustive or prescriptive; you can use other native or third party alternatives in addition to the drivers shown below.

Even though a number of database drivers might be already available on your Windows operating system, you may still need to download an alternative driver. For some databases, the latest driver supplied by the database vendor is likely to perform better than the driver that shipped with the operating system.

Database vendors may provide drivers either as separate downloadable packages, or bundled with database client software. In the latter case, the database client software normally includes any required database drivers, or provides you with an option during installation to select the drivers and components you wish to install. Database client software typically consists of administration and configuration utilities used to simplify database administration and connectivity, as well as documentation on how to install and configure the database client and any of its components.

Configuring the database client correctly is crucial for establishing a successful connection to the database. Before installing and using the database client software, it is strongly recommended to read carefully the installation and configuration instructions of the database client; these may vary for each database version and for each Windows version.

To understand the capabilities and limitations of each data access technology with respect to each database type, refer to the documentation of that particular database product and also test the connection against your specific environment. To avoid common connectivity issues, note the following:

- Some ADO.NET providers are not supported or have limited support. See ADO.NET Support Notes 181.
- When installing a database driver, it is recommended that it has the same platform as the Altova application (32-bit or 64-bit). For example, if you are using a 32-bit Altova application on a 64-bit operating system, install the 32-bit driver, and set up your database connection using the 32-bit driver, see also Viewing the Available ODBC Drivers 184.
- When setting up an ODBC data source, it is recommended to create the data source name (DSN) as System DSN instead of User DSN. For more information, see Setting up an ODBC Connection 182.
- If the target database is MySQL or MariaDB through ODBC, the option **Return matched rows instead of affected rows** must be enabled in the **Cursor/Results** tab of MySQL ODBC Connector. Alternatively, if you enter the connection string manually through the Database Connection wizard, add `Option=2` to the connection string , for example: `Dsn=mydsn;Option=2;`
- When setting up a JDBC data source, ensure that JRE (Java Runtime Environment) or Java Development Kit (JDK) is installed and that the CLASSPATH environment variable of the operating system is configured. For more information, see Setting up a JDBC Connection 185.
- For the installation instructions and support details of any drivers or database client software that you install from a database vendor, check the documentation provided with the installation package.

Database	Interface	Drivers
Firebird	ADO.NET	Firebird ADO.NET Data Provider (https://www.firebirdsql.org/en/additional-downloads/)
	JDBC	Firebird JDBC driver (https://www.firebirdsql.org/en/jdbc-driver/)
	ODBC	Firebird ODBC driver (https://www.firebirdsql.org/en/odbc-driver/)

Database	Interface	Drivers
IBM DB2	ADO	IBM OLE DB Provider for DB2
	ADO.NET	IBM Data Server Provider for .NET
	JDBC	IBM Data Server Driver for JDBC and SQLJ
	ODBC	IBM DB2 ODBC Driver
IBM DB2 for i	ADO	• IBM DB2 for i5/OS IBMDA400 OLE DB Provider • IBM DB2 for i5/OS IBMDARLA OLE DB Provider • IBM DB2 for i5/OS IBMDASQL OLE DB Provider
	ADO.NET	.NET Framework Data Provider for IBM i
	JDBC	IBM Toolbox for Java JDBC Driver
	ODBC	iSeries Access ODBC Driver
IBM Informix	ADO	IBM Informix OLE DB Provider
	JDBC	IBM Informix JDBC Driver
	ODBC	IBM Informix ODBC Driver
Microsoft Access	ADO	• Microsoft Jet OLE DB Provider • Microsoft Access Database Engine OLE DB Provider
	ADO.NET	.NET Framework Data Provider for OLE DB
	ODBC	• Microsoft Access Driver
MariaDB	ADO.NET	In the absence of a dedicated .NET connector for MariaDB, use **Connector/NET** for MySQL (https://dev.mysql.com/downloads/connector/net/).
	JDBC	MariaDB Connector/J (https://downloads.mariadb.org/)
	ODBC	MariaDB Connector/ODBC (https://downloads.mariadb.org/)
Microsoft SQL Server	ADO	• Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL) • Microsoft OLE DB Provider for SQL Server (SQLOLEDB) • SQL Server Native Client (SQLNCLI)
	ADO.NET	• .NET Framework Data Provider for SQL Server • .NET Framework Data Provider for OLE DB
	JDBC	• Microsoft JDBC Driver for SQL Server (https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server)
	ODBC	• ODBC Driver for Microsoft SQL Server (https://docs.microsoft.com/en-us/SQL/connect/odbc/download-odbc-driver-for-sql-server)
MySQL	ADO.NET	• Connector/NET (https://dev.mysql.com/downloads/connector/net/)

Database	Interface	Drivers
	JDBC	Connector/J (https://dev.mysql.com/downloads/connector/j/)
	ODBC	Connector/ODBC (https://dev.mysql.com/downloads/connector/odbc/)
Oracle	ADO	• Oracle Provider for OLE DB • Microsoft OLE DB Provider for Oracle
	ADO.NET	Oracle Data Provider for .NET (http://www.oracle.com/technetwork/topics/dotnet/index-085163.html)
	JDBC	• JDBC Thin Driver • JDBC Oracle Call Interface (OCI) Driver These drivers are typically installed during the installation of your Oracle database client. Connect through the OCI Driver (not the Thin Driver) if you are using the Oracle XML DB component.
	ODBC	• Microsoft ODBC for Oracle • Oracle ODBC Driver (typically installed during the installation of your Oracle database client)
PostgreSQL	JDBC	PostgreSQL JDBC Driver (https://jdbc.postgresql.org/download.html)
	ODBC	psqlODBC (https://odbc.postgresql.org/)
	Native Connection	Available. There is no need to install any drivers if using native connection.
Progress OpenEdge	JDBC	JDBC Connector (https://www.progress.com/jdbc/openedge)
	ODBC	ODBC Connector (https://www.progress.com/odbc/openedge)
SQLite	Native Connection	Available. There is no need to install any drivers if using native connection.
Sybase	ADO	Sybase ASE OLE DB Provider
	JDBC	jConnect™ for JDBC
	ODBC	Sybase ASE ODBC Driver
Teradata	ADO.NET	.NET Data Provider for Teradata (https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata)
	JDBC	Teradata JDBC Driver (https://downloads.teradata.com/download/connectivity/jdbc-driver)
	ODBC	Teradata ODBC Driver for Windows (https://downloads.teradata.com/download/connectivity/odbc-driver/windows)

4.2.1.3 ADO Connection

Microsoft ActiveX Data Objects (ADO) is a data access technology that enables you to connect to a variety of data sources through OLE DB. OLE DB is an alternative interface to ODBC or JDBC; it provides uniform access to data in a COM (Component Object Model) environment. ADO is a precursor of the newer ADO.NET175 and is still one of the possible ways to connect to Microsoft native databases such as Microsoft Access or SQL Server, although you can also use it for other data sources.

Importantly, you can choose between multiple ADO providers, and some of them must be downloaded and installed on your workstation before you can use them. For example, for connecting to SQL Server, the following ADO providers are available:

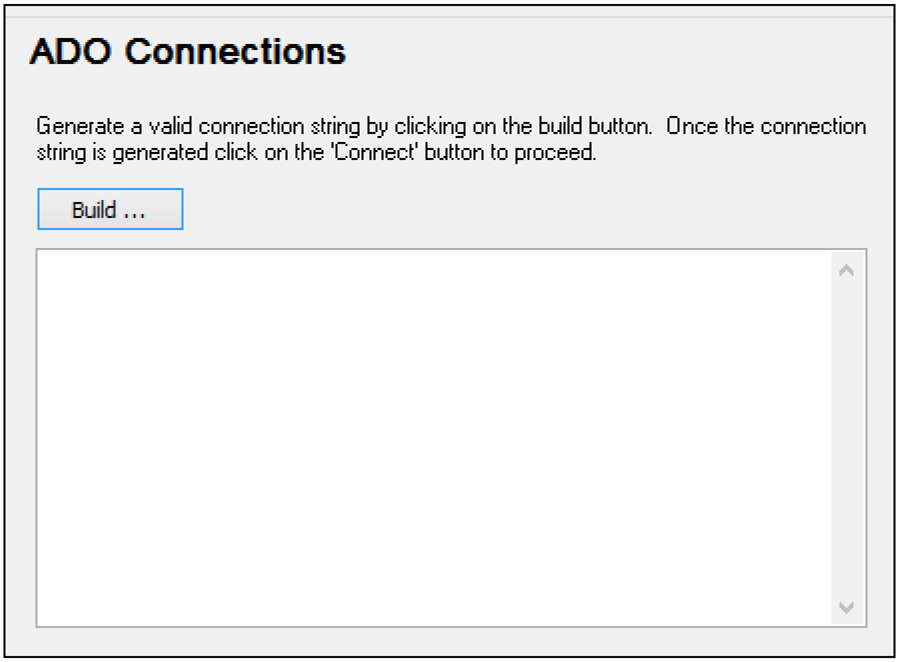
- Microsoft OLE DB *Driver* for SQL Server (MSOLEDBSQL)
- Microsoft OLE DB *Provider* for SQL Server (SQLOLEDB)
- SQL Server Native Client (SQLNCLI)

From the providers listed above, the recommended one is MSOLEDBSQL; you can download it from https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15. Note that it must match the platform of MapForce (32-bit or 64-bit). The SQLOLEDB and SQLNCLI providers are considered deprecated and thus are not recommended.

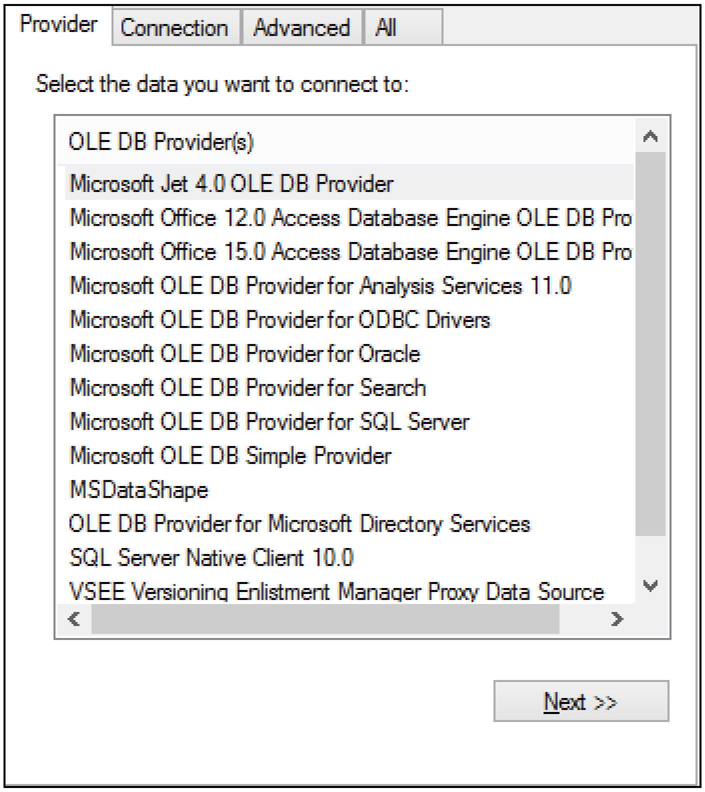
The **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)** is known to have issues with parameter binding of complex queries like Common Table Expressions (CTE) and nested SELECT statements.

To set up an ADO connection:

1. Start the database connection wizard165.
2. Click **ADO Connections**.



3. Click **Build**.



4. Select the data provider through which you want to connect. The table below lists a few common scenarios.

To connect to this database...	Use this provider...
Microsoft Access	• **Microsoft Office Access Database Engine OLE DB Provider** (recommended) • **Microsoft Jet OLE DB Provider** If the **Microsoft Office Access Database Engine OLE DB Provider** is not available in the list, make sure that you have installed either Microsoft Access or the Microsoft Access Database Engine Redistributable (https://www.microsoft.com/en-us/download/details.aspx?id=54920) on your computer.
SQL Server	• **Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL)** - this is the recommended OLE DB provider. In order for this provider to appear in the list, it must be downloaded from https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15 and installed. • **Microsoft OLE DB Provider for SQL Server (OLEDBSQL)** • **SQL Server Native Client (SQLNCLI)**
Other database	Select the provider applicable to your database. If an OLE DB provider to your database is not available, install the required driver from the database vendor (see Database Drivers Overview [167]). Alternatively, set up an ADO.NET, ODBC, or JDBC connection. If the operating system has an ODBC driver to the required database, you could also use the **Microsoft OLE DB Provider for ODBC Drivers**, or preferably opt for an ODBC connection [182].

5. Having selected the provider of choice, click **Next** and complete the wizard.

The subsequent wizard steps are specific to the provider you chose. For SQL Server, you will need to provide or select the host name of the database server, the authentication method, the database name, as well as the database username and password. For an example, see Connecting to Microsoft SQL Server (ADO) [216]. For Microsoft Access, you will be asked to browse for or provide the path to the database file. For an example, see Connecting to Microsoft Access (ADO) [214].

The complete list of initialization properties (connection parameters) is available in the **All** tab of the connection dialog box—these properties vary depending on the chosen provider and may need to be set explicitly in order for the connection to be possible. The following sections provide guidance on configuring the basic initialization properties for Microsoft Access and SQL Server databases:

- Setting up the SQL Server Data Link Properties [173]
- Setting up the Microsoft Access Data Link Properties [174]

4.2.1.3.1 Connecting to an Existing Microsoft Access Database

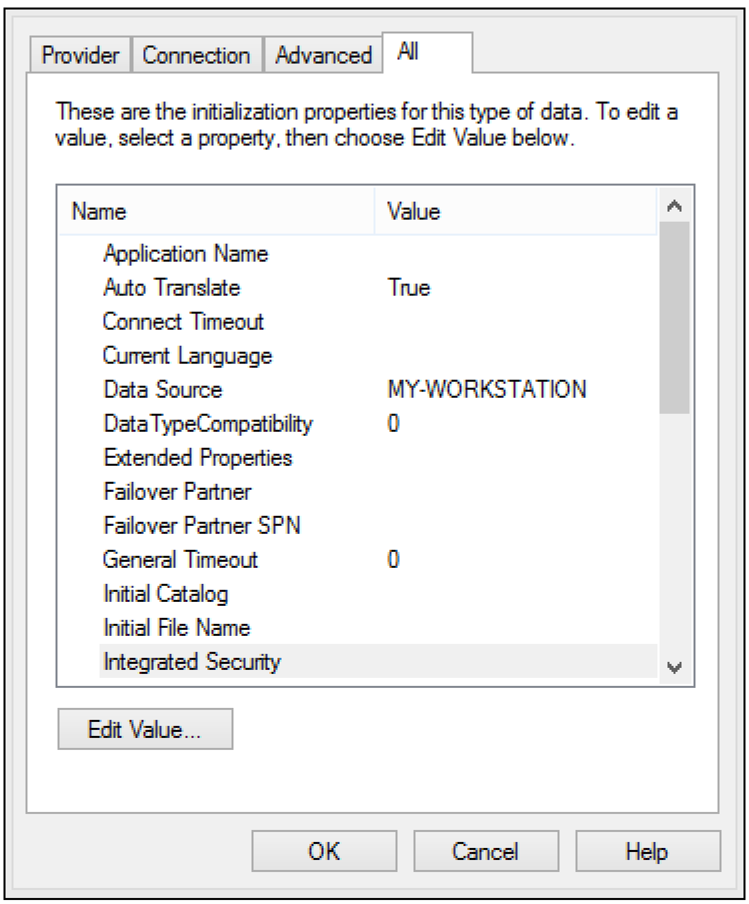
This approach is suitable when you want to connect to a Microsoft Access database which is not password-protected. If the database is password-protected, set up the database password as shown in Connecting to Microsoft Access (ADO)[214].

To connect to an existing Microsoft Access database:

1. Run the database connection wizard (see Starting the Database Connection Wizard[165]).
2. Select **Microsoft Access (ADO)**, and then click **Next**.
3. Browse for the database file, or enter the path to it (either relative or absolute).
4. Click **Connect**.

4.2.1.3.2 Setting up the SQL Server Data Link Properties

When you connect to a Microsoft SQL Server database through ADO[170], you may need to set the following connection properties in the **All** tab of the Data Link Properties dialog box.

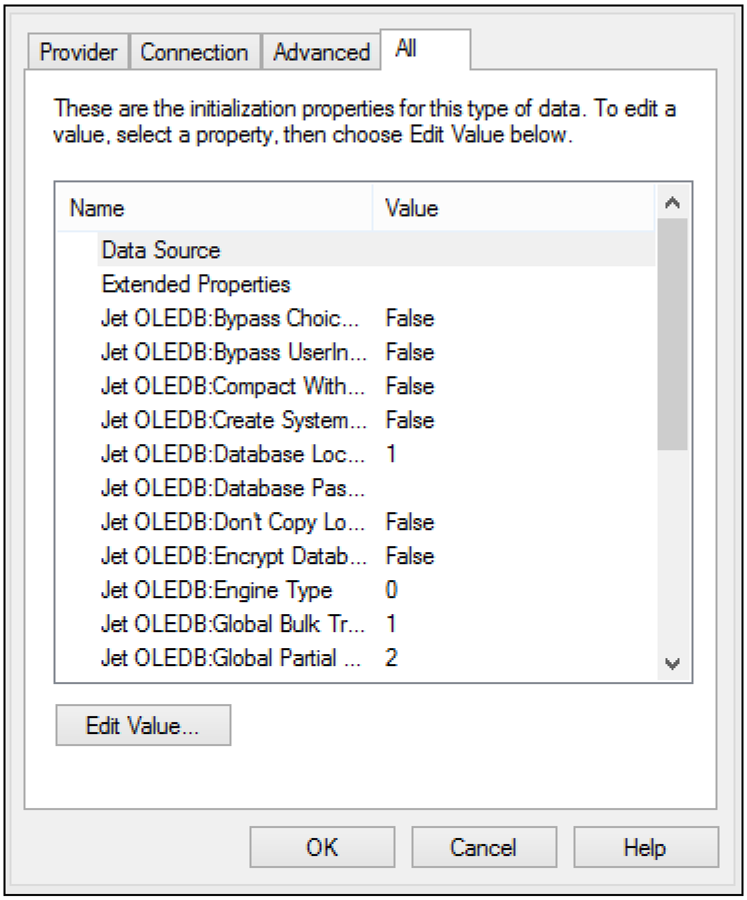


Data Link Properties dialog box

Property	Notes
Integrated Security	If you selected the **SQL Server Native Client** data provider on the **Provider** tab, set this property to a space character.
Persist Security Info	Set this property to **True**.

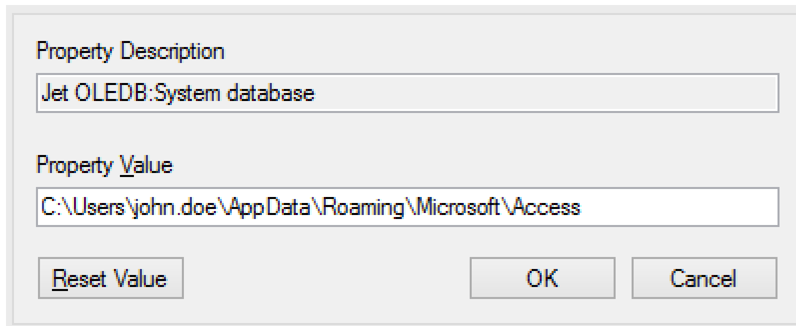
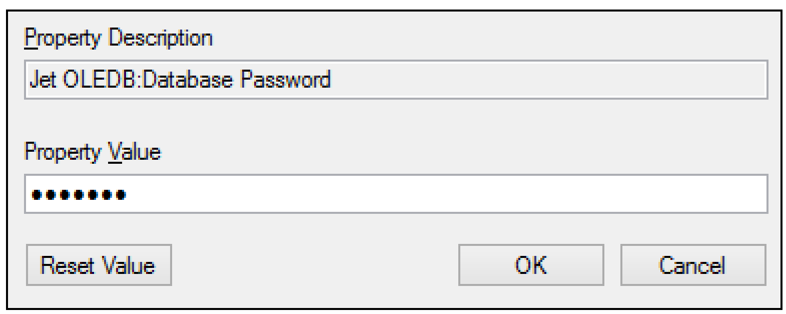
4.2.1.3.3 Setting up the Microsoft Access Data Link Properties

When you connect to a Microsoft Access database through ADO 170, you may need to set the following connection properties in the **All** tab of the Data Link Properties dialog box.



Data Link Properties dialog box

Property	Notes
Data Source	This property stores the path to the Microsoft Access database file. To avoid database connectivity issues, it is recommended to use the UNC (Universal Naming Convention) path format, for example: `\\anyserver\share$\filepath`

Property	Notes
Jet OLEDB:System Database	This property stores the path to the workgroup information file. You may need to explicitly set the value of this property before you can connect to a Microsoft Access database. If you cannot connect due to a "workgroup information file" error, locate the workgroup information file (**System.MDW**) applicable to your user profile, and set the property value to the path of the **System.MDW** file. 
Jet OLEDB:Database Password	If the database is password-protected, set the value of this property to the database password. 

4.2.1.4 ADO.NET Connection

ADO.NET is a set of Microsoft .NET Framework libraries designed to interact with data, including data from databases. To connect to a database from MapForce through ADO.NET, Microsoft .NET Framework 4 or later is required. As shown below, you connect to a database through ADO.NET by selecting a .NET provider and supplying a connection string.

A .NET data provider is a collection of classes that enables connecting to a particular type of data source (for example, a SQL Server, or an Oracle database), executing commands against it, and fetching data from it. In other words, with ADO.NET, an application such as MapForce interacts with a database through a data provider. Each data provider is optimized to work with the specific type of data source that it is designed for. There are two types of .NET providers:

1. Supplied by default with Microsoft .NET Framework.

2. Supplied by major database vendors, as an extension to the .NET Framework. Such ADO.NET providers must be installed separately and can typically be downloaded from the website of the respective database vendor.

Note: Certain ADO.NET providers are not supported or have limited support. See ADO.NET Support Notes (181).

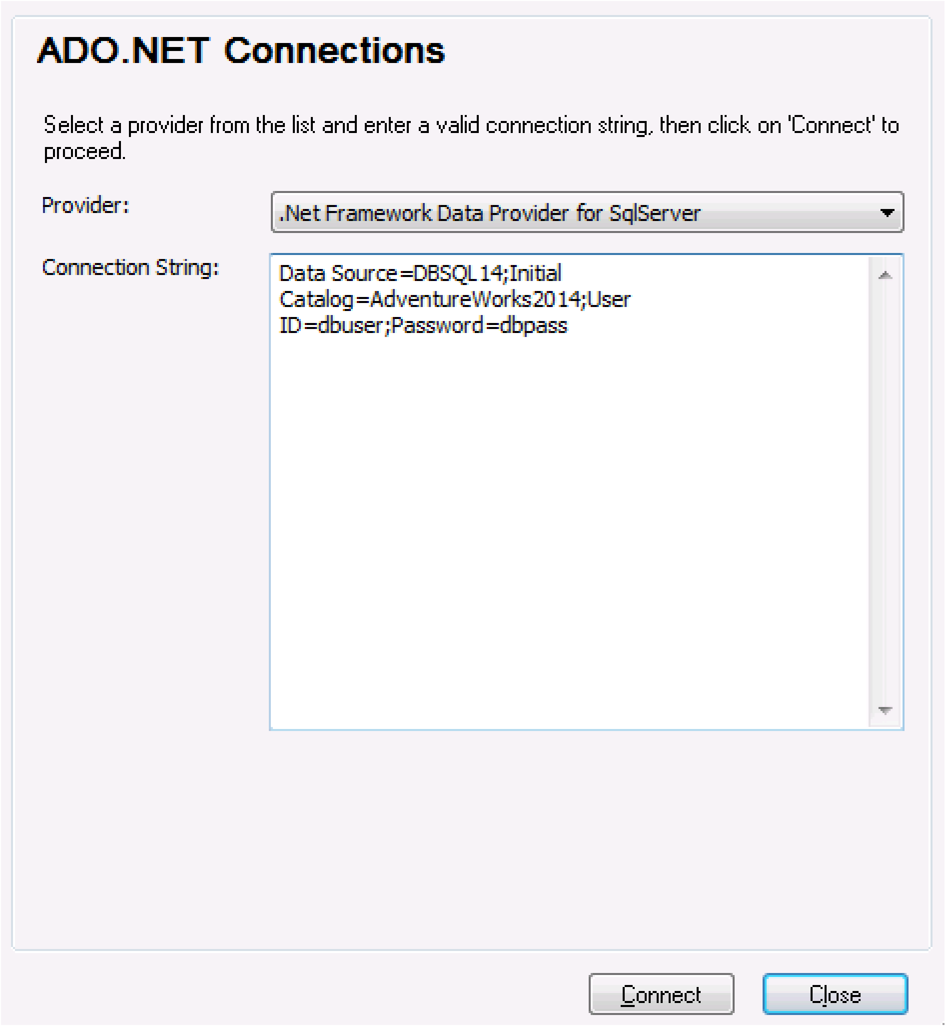
To set up an ADO.NET connection:

1. Start the database connection wizard (165).
2. Click **ADO.NET Connections**.
3. Select a .NET data provider from the list.

> The list of providers available by default with the .NET Framework appears in the "Provider" list. Vendor-specific .NET data providers are available in the list only if they are already installed on your system. To become available, vendor-specific .NET providers must be installed into the GAC (Global Assembly Cache), by running the .msi or .exe file supplied by the database vendor.

4. Enter a database connection string. A connection string defines the database connection information, as semicolon-delimited key/value pairs of connection parameters. For example, a connection string such as **`Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass`** connects to the SQL Server database `ProductsDB` on server `DBSQLSERV`, with the user name `dbuser` and password `dbpass`. You can create a connection string by typing the key/value pairs directly into the "Connection String" dialog box. Another option is to create it with Visual Studio (see Creating a Connection String in Visual Studio (177)).

> The syntax of the connection string depends on the provider selected from the "Provider" list. For examples, see Sample ADO.NET Connection Strings (180).



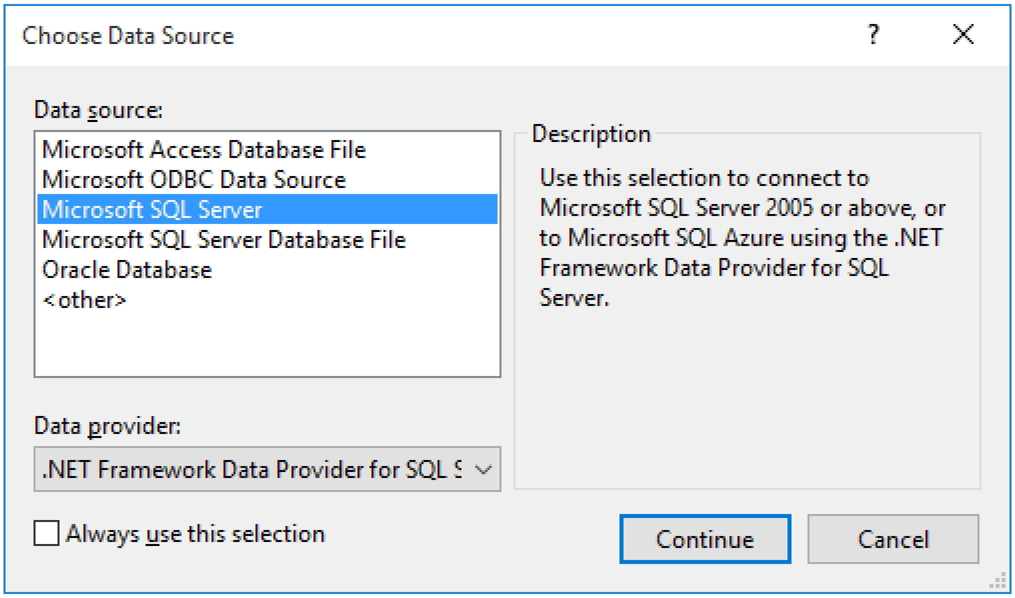
5. Click **Connect**.

4.2.1.4.1 Creating a Connection String in Visual Studio

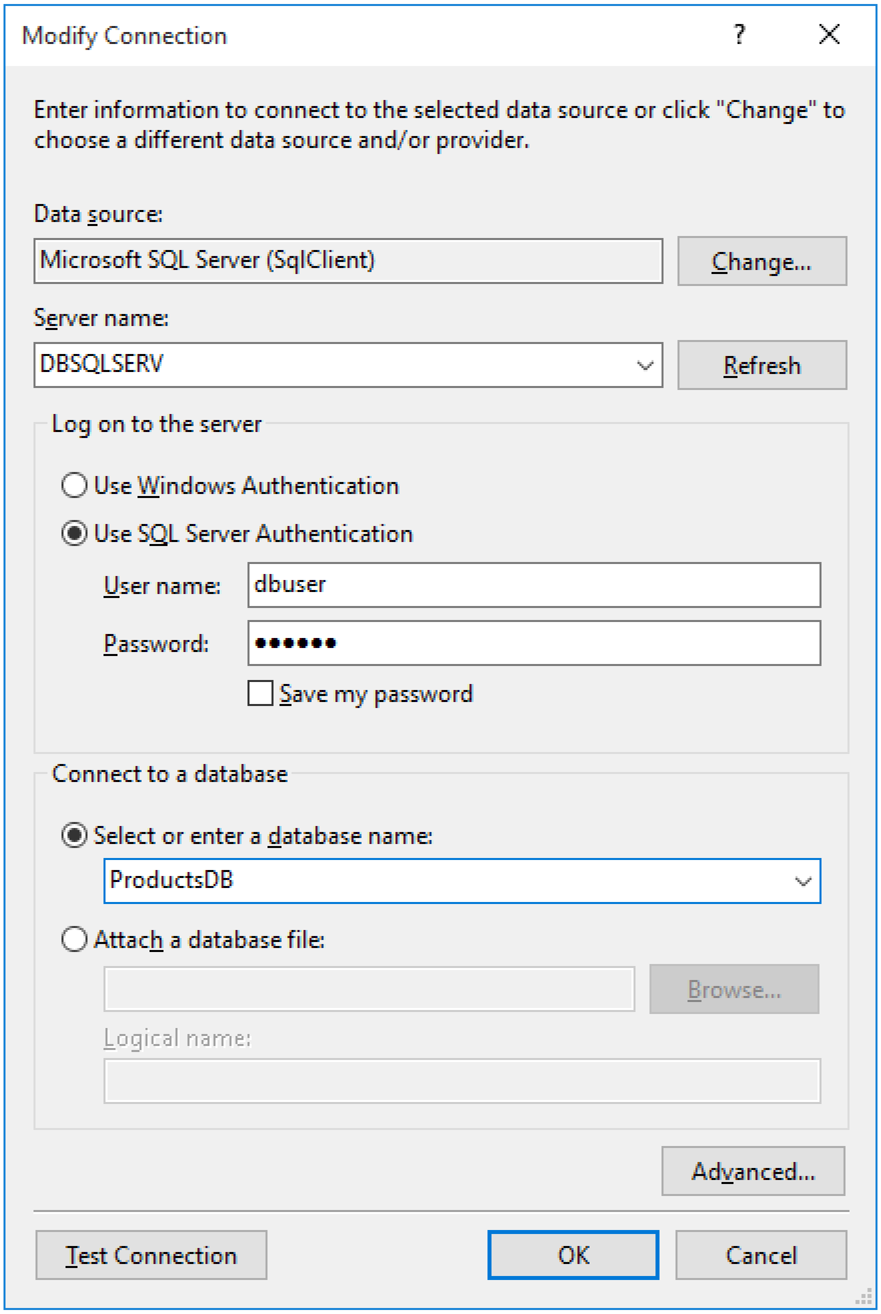
In order to connect to a data source using ADO.NET, a valid database connection string is required. The following instructions show you how to create a connection string from Visual Studio.

To create a connection string in Visual Studio:

1. On the **Tools** menu, click **Connect to Database**.
2. Select a data source from the list (in this example, Microsoft SQL Server). The Data Provider is filled automatically based on your choice.



3. Click **Continue**.



4. Enter the server host name and the user name and password to the database. In this example, we are connecting to the database `ProductsDB` on server `DBSQLSERV`, using SQL Server authentication.
5. Click **OK**.

If the database connection is successful, it appears in the Server Explorer window. You can display the Server Explorer window using the menu command **View | Server Explorer**. To obtain the database connection string, right-click the connection in the Server Explorer window, and select **Properties**. The connection string is now displayed in the Properties window of Visual Studio. Note that, before pasting the string into the "Connection String" box of MapForce, you will need to replace the asterisk (*) characters with the actual password.

4.2.1.4.2 Sample ADO.NET Connection Strings

To set up an ADO.NET connection, you need to select an ADO.NET provider from the database connection dialog box and enter a connection string (see also Setting up an ADO.NET Connection 175). Sample ADO.NET connection strings for various databases are listed below under the .NET provider where they apply.

.NET Data Provider for Teradata

This provider can be downloaded from Teradata website (https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata). A sample connection string looks as follows:

```
Data Source=ServerAddress;User Id=user;Password=password;
```

.NET Framework Data Provider for IBM i

This provider is installed as part of *IBM i Access Client Solutions - Windows Application Package*. A sample connection string looks as follows:

```
DataSource=ServerAddress;UserID=user;Password=password;DataCompression=True;
```

For more information, see the ".NET Provider Technical Reference" help file included in the installation package above.

.NET Framework Data Provider for MySQL

This provider can be downloaded from MySQL website (https://dev.mysql.com/downloads/connector/net/). A sample connection string looks as follows:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

See also: https://dev.mysql.com/doc/connector-net/en/connector-net-programming-connecting-connection-string.html

.NET Framework Data Provider for SQL Server

A sample connection string looks as follows:

```
Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass
```

See also: https://msdn.microsoft.com/en-us/library/ms254500(v=vs.110).aspx

IBM DB2 Data Provider 10.1.2 for .NET Framework 4.0

```
Database=PRODUCTS;UID=user;Password=password;Server=localhost:50000;
```

Note: This provider is typically installed with the IBM DB2 Data Server Client package. If the provider is missing from the list of ADO.NET providers after installing IBM DB2 Data Server Client package, refer to the following technical note: https://www-01.ibm.com/support/docview.wss?uid=swg21429586.

See also:
https://www.ibm.com/support/knowledgecenter/en/SSEPGG_10.1.0/com.ibm.swg.im.dbclient.adonet.ref.doc/doc/DB2ConnectionClassConnectionStringProperty.html

Oracle Data Provider for .NET (ODP.NET)

The installation package which includes the ODP.NET provider can be downloaded from the Oracle website (see http://www.oracle.com/technetwork/topics/dotnet/downloads/index.html). A sample connection string looks as follows:

```
Data Source=DSORCL;User Id=user;Password=password;
```

Where `DSORCL` is the name of the data source which points to an Oracle service name defined in the **tnsnames.ora** file, as described in Connecting to Oracle (ODBC) 229.

To connect without configuring a service name in the **tnsnames.ora** file, use a string such as:

```
Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=host)(PORT=port)))
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=MyOracleSID)));User
Id=user;Password=password;
```

See also: https://docs.oracle.com/cd/B28359_01/win.111/b28375/featConnecting.htm

4.2.1.4.3 ADO.NET Support Notes

The following table lists known ADO.NET database drivers that are currently not supported or have limited support in MapForce.

Database	Driver	Support notes
All databases	**.Net Framework Data Provider for ODBC**	Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ODBC direct connections instead.
	.Net Framework Data Provider for OleDb	Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ADO direct connections instead.
Firebird	**Firebird ADO.NET Data Provider**	Limited support. It is recommended to use ODBC or JDBC instead.
Informix	**IBM Informix Data Provider for**	Not supported. Use **DB2 Data Server**

Database	Driver	Support notes
	.NET Framework 4.0	**Provider** instead.
IBM DB2 for i (iSeries)	**.Net Framework Data Provider for i5/OS**	Not supported. Use **.Net Framework Data Provider for IBM i** instead, installed as part of the *IBM i Access Client Solutions - Windows Application Package*.
Oracle	**.Net Framework Data Provider for Oracle**	Limited support. Although this driver is provided with the .NET Framework, its usage is discouraged by Microsoft, because it is deprecated.
PostgreSQL	**-**	No ADO.NET drivers for this vendor are supported. Use a native connection instead.
Sybase	**-**	No ADO.NET drivers for this vendor are supported.

4.2.1.5 ODBC Connection

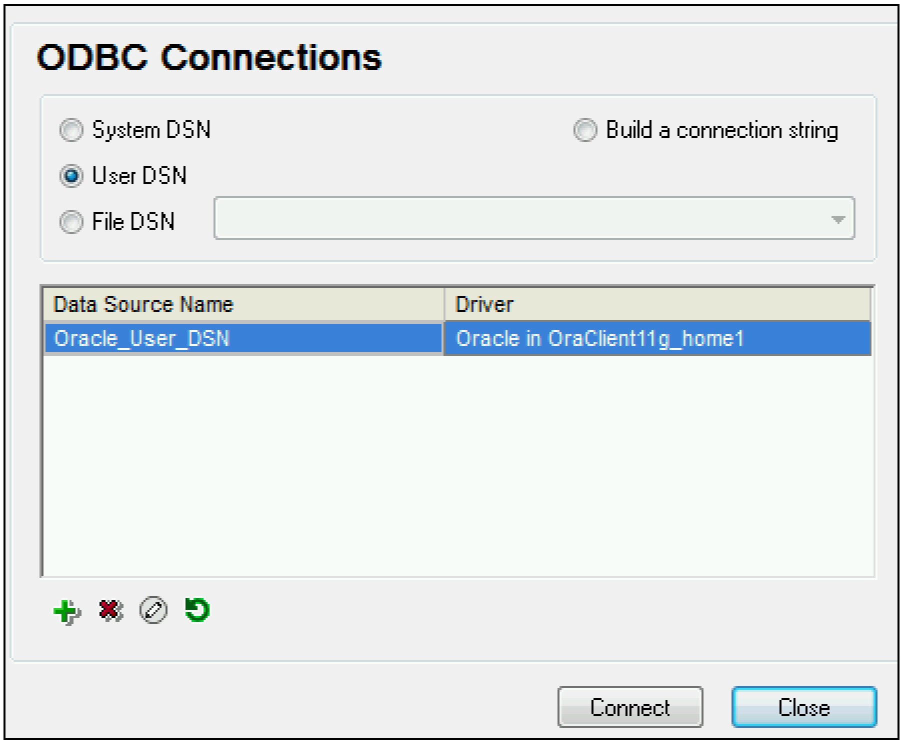
ODBC (Open Database Connectivity) is a widely used data access technology that enables you to connect to a database from MapForce. It can be used either as primary means to connect to a database, or as an alternative to native, OLE DB, or JDBC-driven connections.

To connect to a database through ODBC, first you need to create an ODBC data source name (DSN) on the operating system. This step is not required if the DSN has already been created, perhaps by another user of the operating system. The DSN represents a uniform way to describe the database connection to any ODBC-aware client application on the operating system, including MapForce. DSNs can be of the following types:

- System DSN
- User DSN
- File DSN

A *System* data source is accessible by all users with privileges on the operating system. A *User* data source is available to the user who created it. Finally, if you create a *File DSN*, the data source will be created as a file with the .dsn extension which you can share with other users, provided that they have installed the drivers used by the data source.

Any DSNs already available on your machine are listed by the database connection dialog box when you click **ODBC connections** on the ODBC connections dialog box.



ODBC Connections dialog box

If a DSN to the required database is not available, the MapForce database connection wizard will assist you to create it; however, you can also create it directly on your Windows operating system. In either case, before you proceed, ensure that the ODBC driver applicable for your database is in the list of ODBC drivers available to the operating system (see Viewing the Available ODBC Drivers 184).

To connect by using a new DSN:

1. Start the database connection wizard 165.
2. On the database connection dialog box, click **ODBC Connections**.
3. Select a data source type (User DSN, System DSN, File DSN).

> To create a System DSN, you need administrative rights on the operating system, and MapForce must be run as administrator.

4. Click **Add** .
5. Select a driver, and then click **User DSN** or **System DSN** (depending on the type of the DSN you want to create). If the driver applicable to your database is not listed, download it from the database vendor and install it (see Database Drivers Overview 167).
6. On the dialog box that pops up, fill in any driver specific connection information to complete the setup.

For the connection to be successful, you will need to provide the host name (or IP address) of the database server, as well as the database username and password. There may be other optional connection parameters—these parameters vary between database providers. For detailed information about the parameters specific to each connection method, consult the documentation of the driver provider. Once created, the DSN becomes available in the list of data source names. This enables you to reuse the database connection details any time you want to connect to the database. Note that User DSNs are added to the list of User DSNs whereas System DSNs are added to the list of System DSNs.

To connect by using an existing DSN:

1. Start the database connection wizard 165.
2. Click **ODBC Connections**.
3. Choose the type of the existing data source (User DSN, System DSN, File DSN).
4. Click the existing DSN record, and then click **Connect**.

To build a connection string based on an existing .dsn file:

1. Start the database connection wizard 165.
2. Click **ODBC Connections**.
3. Select **Build a connection string**, and then click **Build**.
4. If you want to build the connection string using a File DSN, click the **File Data Source** tab. Otherwise, click the **Machine Data Source** tab. (System DSNs and User DSNs are known as "Machine" data sources.)
5. Select the required .dsn file, and then click **OK**.

To connect by using a prepared connection string:

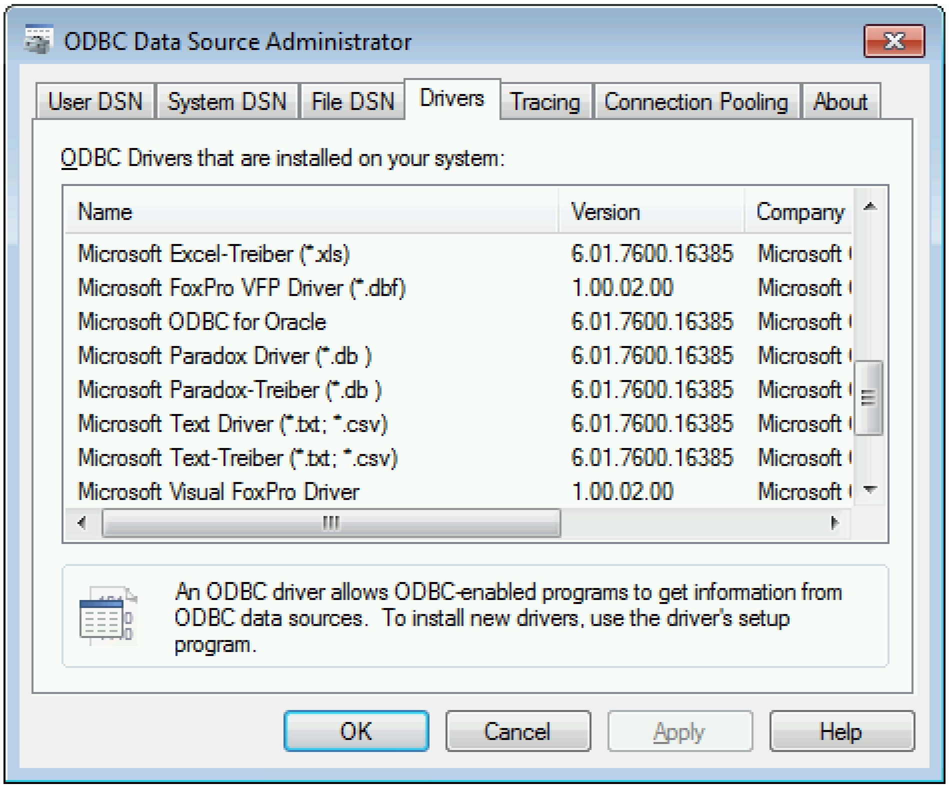
1. Start the database connection wizard 165.
2. Click **ODBC Connections**.
3. Select **Build a connection string**.
4. Paste the connection string into the provided box, and then click **Connect**.

4.2.1.5.1 Available ODBC Drivers

You can view the ODBC drivers available on your operating system in the ODBC Data Source Administrator. You can access the ODBC Data Source Administrator (**Odbcad32.exe**) from the Windows Control Panel, under **Administrative Tools**. On 64-bit operating systems, there are two versions of this executable:

- The 32-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\SysWoW64** directory (assuming that **C:** is your system drive).
- The 64-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\System32** directory.

Any installed 32-bit database drivers are visible in the 32-bit version of ODBC Data Source Administrator, while 64-bit drivers—in the 64-bit version. Therefore, ensure that you check the database drivers from the relevant version of ODBC Data Source Administrator.



ODBC Data Source Administrator

If the driver to your target database does not exist in the list, or if you want to add an alternative driver, you will need to download it from the database vendor (see Database Drivers Overview [167]). Once the ODBC driver is available on your system, you are ready to create ODBC connections with it (see Setting up an ODBC Connection [182]).

4.2.1.6 JDBC Connection

JDBC (Java Database Connectivity) is a database access interface which is part of the Java software platform from Oracle. JDBC connections are generally more resource-intensive than ODBC connections but may provide features not available through ODBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java Settings [1476]; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC drivers from the database vendor must be installed. These may be JDBC drivers installed as part of a database client installation, or JDBC libraries (.jar files) downloaded separately, if available and supported by the database, see also Database Connection Examples [195].

- The `CLASSPATH` environment variable must include the path to the JDBC driver (one or several .jar files) on your Windows operating system. When you install some database clients, the installer may configure this variable automatically. See also Configuring the CLASSPATH 188.

Connecting to SQL Server via JDBC with Windows credentials

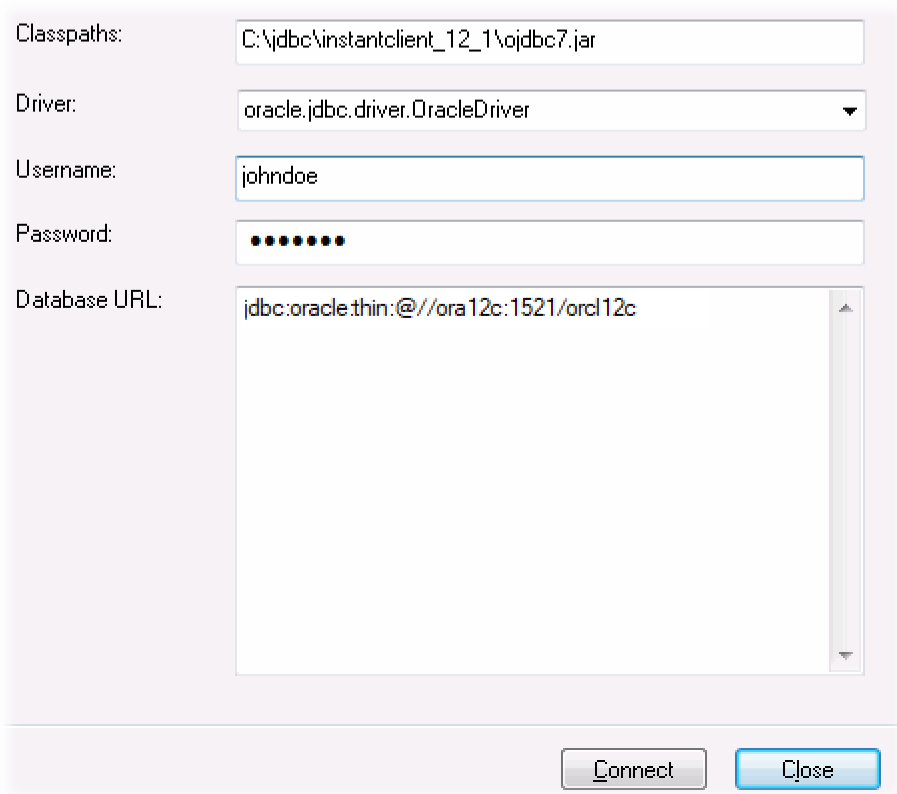
If you connect to SQL Server through JDBC with Windows credentials (integrated security), note the following:

- The **sqljdbc_auth.dll** file included in the JDBC driver package must be copied to a directory that is on the system PATH environment variable. There are two such files, one for the x86 and one for x64 platform. Make sure that you add to the PATH the one that corresponds to your JDK platform. Also, make sure that you restart MapForce (or the program that runs the mapping, if applicable) after changing the environment variable.
- The JDBC connection string must include the property **`integratedSecurity=true`**. You can add this property from various places:
  - o from the database connection wizard, see below
  - o from the database component settings 279
  - o if applicable, by editing the database connection string in generated Java code 1508.

For further information, refer to *Microsoft JDBC driver for SQL Server* documentation, https://docs.microsoft.com/en-us/sql/connect/jdbc/building-the-connection-url.

Setting up a JDBC connection

1. Start the database connection wizard 165.
2. Click **JDBC Connections**.
3. Optionally, enter a semicolon-separated list of .jar file paths in the "Classpaths" text box. The .jar libraries entered here will be loaded into the environment in addition to those already defined in the `CLASSPATH` environment variable. When you finish editing the "Classpaths" text box, any JDBC drivers found in the source .jar libraries are automatically added to the "Driver" list (see the next step).



4. Next to "Driver", select a JDBC driver from the list, or enter a Java class name. Note that this list contains any JDBC drivers configured through the `CLASSPATH` environment variable (see Configuring the CLASSPATH 188), as well as those found in the "Classpaths" text box.

> The JDBC driver paths defined in the CLASSPATH variable, as well as any .jar file paths entered directly in the database connection dialog box are all supplied to the Java Virtual Machine (JVM). The JVM then decides which drivers to use in order to establish a connection. It is recommended to keep track of Java classes loaded into the JVM so as not to create potential JDBC driver conflicts and avoid unexpected results when connecting to the database.

5. Enter the username and password to the database in the corresponding boxes.
6. In the Database URL text box, enter the JDBC connection URL (string) in the format specific to your database type. The following table describes the syntax of JDBC connection URLs (strings) for common database types.

Database	JDBC Connection URL
Firebird	`jdbc:firebirdsql://<host>[:<port>]/<database path or alias>`
IBM DB2	`jdbc:db2://hostName:port/databaseName`
IBM DB2 for i	`jdbc:as400://[host]`

Database	JDBC Connection URL
IBM Informix	`jdbc:informix-sqli://`**`hostName`**`:`**`port`**`/`**`databaseName`**`:INFORMIXSERVER=`**`myserver`**
MariaDB	`jdbc:mariadb://`**`hostName`**`:`**`port`**`/`**`databaseName`**
Microsoft SQL Server	`jdbc:sqlserver://`**`hostName`**`:`**`port`**`;`**`databaseName`**`=name`
MySQL	`jdbc:mysql://`**`hostName`**`:`**`port`**`/`**`databaseName`**
Oracle	`jdbc:oracle:thin:@`**`hostName`**`:`**`port`**`:`**`SID`** `jdbc:oracle:thin:@//`**`hostName`**`:`**`port`**`/`**`service`**
Oracle XML DB	`jdbc:oracle:oci:@//`**`hostName`**`:`**`port`**`:`**`service`**
PostgreSQL	`jdbc:postgresql://`**`hostName`**`:`**`port`**`/`**`databaseName`**
Progress OpenEdge	`jdbc:datadirect:openedge://`**`host`**`:`**`port`**`;databaseName=`**`db_name`**
Sybase	`jdbc:sybase:Tds:`**`hostName`**`:`**`port`**`/`**`databaseName`**
Teradata	`jdbc:teradata://`**`databaseServerName`**

Note: Syntax variations to the formats listed above are also possible (for example, the database URL may exclude the port or may include the username and password to the database). Check the documentation of the database vendor for further details.

7. Click **Connect**.

4.2.1.6.1 Configuring the CLASSPATH

The `CLASSPATH` environment variable is used by the Java Runtime Environment (JRE) or the Java Development Kit (JDK) to locate Java classes and other resource files on your operating system. When you connect to a database through JDBC, this variable must be configured to include the path to the JDBC driver on your operating system, and, in some cases, the path to additional library files specific to the database type you are using.

The following table lists sample file paths that must be typically included in the `CLASSPATH` variable. Importantly, you may need to adjust this information based on the location of the JDBC driver on your system, the JDBC driver name, as well as the JRE/JDK version present on your operating system. To avoid connectivity problems, check the installation instructions and any pre-installation or post-installation configuration steps applicable to the JDBC driver installed on your operating system.

Database	Sample CLASSPATH entries
Firebird	`C:\Program Files\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar`
IBM DB2	`C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;`

Database	Sample CLASSPATH entries
IBM DB2 for i	`C:\jt400\jt400.jar;`
IBM Informix	`C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;`
Microsoft SQL Server	`C:\Program Files\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\enu\sqljdbc.jar`
MariaDB	**<installation directory>**\mariadb-java-client-2.2.0.jar
MySQL	**<installation directory>**\mysql-connector-java-*version*-bin.jar;
Oracle	**ORACLE_HOME**\jdbc\lib\ojdbc6.jar;
Oracle (with XML DB)	**ORACLE_HOME**\jdbc\lib\ojdbc6.jar;**ORACLE_HOME**\LIB\xmlparserv2.jar; **ORACLE_HOME**\RDBMS\jlib\xdb.jar;
PostgreSQL	**<installation directory>**\postgresql.jar
Progress OpenEdge	`%DLC%\java\openedge.jar;%DLC%\java\pool.jar;` Note: Assuming the Progress OpenEdge SDK is installed on the machine, `%DLC%` is the directory where OpenEdge is installed.
Sybase	`C:\sybase\jConnect-7_0\classes\jconn4.jar`
Teradata	**<installation directory>**\tdgssconfig.jar;**<installation directory>**\terajdbc4.jar

- Changing the CLASSPATH variable may affect the behavior of Java applications on your machine. To understand possible implications before you proceed, refer to the Java documentation.
- Environment variables can be user or system. To change system environment variables, you need administrative rights on the operating system.
- After you change the environment variable, restart any running programs for settings to take effect. Alternatively, log off or restart your operating system.

To configure the CLASSPATH on Windows 7:

1. Open the **Start** menu and right-click **Computer**.
2. Click **Properties**.
3. Click **Advanced system settings**.
4. In the **Advanced** tab, click **Environment Variables**,
5. Locate the CLASSPATH variable under user or system environment variables, and then click Edit. If the CLASSPATH variable does not exist, click **New** to create it.
6. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator (;).

To configure the CLASSPATH on Windows 10:

1. Press the Windows key and start typing "environment variables".
2. Click the suggestion **Edit the system environment variables**.
3. Click **Environment Variables**.
4. Locate the CLASSPATH variable under user or system environment variables, and then click **Edit**. If the CLASSPATH variable does not exist, click **New** to create it.
5. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator (;).

4.2.1.7 PostgreSQL Connection

Connections to PostgreSQL databases can be set up either as native connections, or connections via ODBC, JDBC, and other drivers. The advantage of setting up a native connection is that no drivers are required to be installed on your system. Also, if you intend to deploy files for execution on a Linux or macOS server, no drivers are required to be installed on the target server as well.

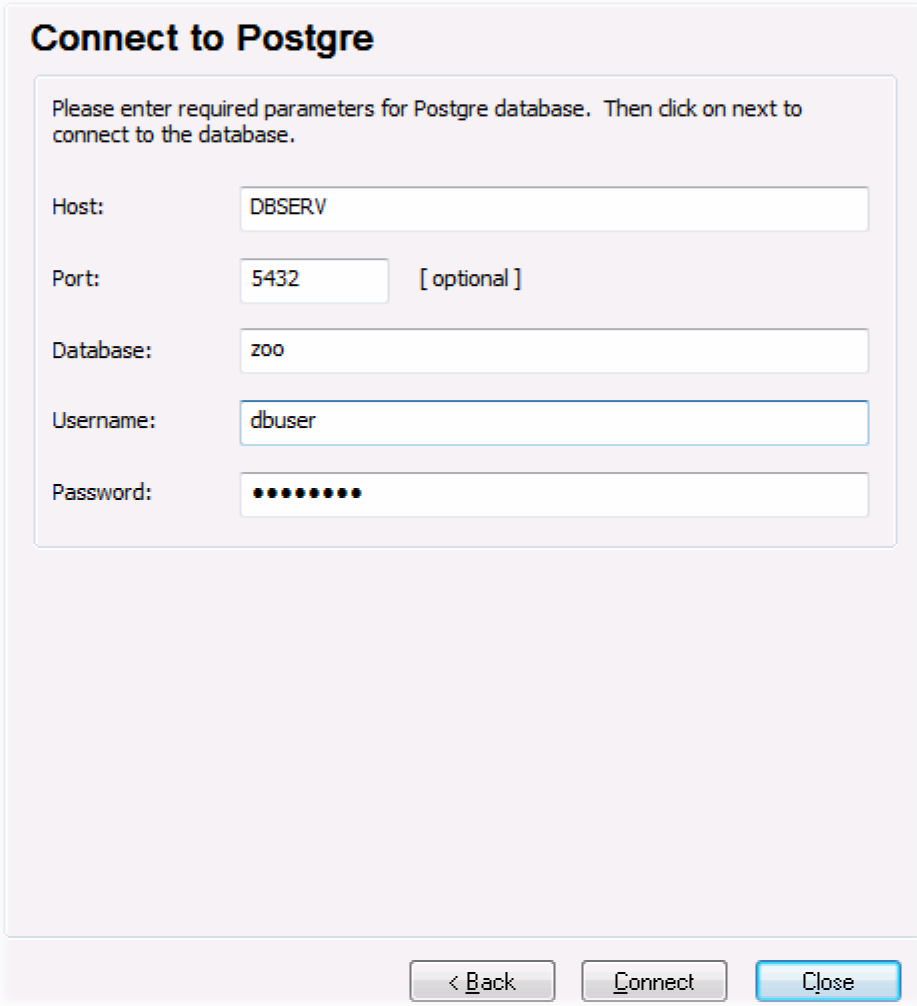
If you prefer to establish a connection by means of a non-native driver, see the following topics:

- Setting up a JDBC Connection 185
- Connecting to PostgreSQL (ODBC) 233

Otherwise, if you want to set up a native connection to PostgreSQL, follow the steps below. To proceed, you need the following prerequisites: host name, port, database name, username, and password.

To set up a native PostgreSQL connection:

1. Start the database connection wizard 165.
2. Click **PostgreSQL Connections**.
3. Enter the host (*localhost*, if PostgreSQL runs on the same machine), port (typically 5432, this is optional), the database name, username, and password in the corresponding boxes.



4. Click **Connect**.

If the PostgreSQL database server is on a different machine, note the following:

- The PostgreSQL database server must be configured to accept connections from clients. Specifically, the **pg_hba.conf** file must be configured to allow non-local connections. Secondly, the **postgresql.conf** file must be configured to listen on specified IP address(es) and port. For more information, check the PostgreSQL documentation (https://www.postgresql.org/docs/9.5/static/client-authentication-problems.html).
- The server machine must be configured to accept connections on the designated port (typically, 5432) through the firewall. For example, on a database server running on Windows, a rule may need to be created to allow connections on port 5432 through the firewall, from **Control Panel > Windows Firewall > Advanced Settings > Inbound Rules**.

4.2.1.8 SQLite Connection

SQLite is a file-based, self-contained database type, which makes it ideal in scenarios where portability and ease of configuration is important. Since SQLite databases are natively supported by MapForce, you do not need to install any drivers to connect to them.

SQLite database support notes

- On Linux, statement execution timeout for SQLite databases is not supported.
- Full text search tables are not supported.
- SQLite allows values of different data types in each row of a given table. All processed values must be compatible with the declared column type; therefore, unexpected values can be retrieved and run-time errors may occur if your SQLite database has row values which are not the same as the declared column type.
- If your mapping should write data to a SQLite database, and if you don't have the target database file already, you will need to create it separately. In this case, you can create it with a tool such as DatabaseSpy or download the SQLite command-line shell from the official website, and create the database file from the command line (see also Example: Mapping data from XML to SQLite 329). For complete reference to SQLite command syntax, refer to the official SQLite documentation.
- SQLite databases are supported in the MapForce BUILT-IN transformation language (either when you preview the mapping or when you run a MapForce Server execution file).
- SQLite databases are not supported in user-defined functions (UDF).

> **Important**
>
> It is recommended to create tables with the `STRICT` keyword to ensure more predictable behavior of your data. Otherwise, the data may not be read or written correctly when values of different types are mixed in one column. To find out more about STRICT tables, see the SQLite documentation.

4.2.1.8.1 Connect to an Existing SQLite Database

To connect to an existing SQLite database:

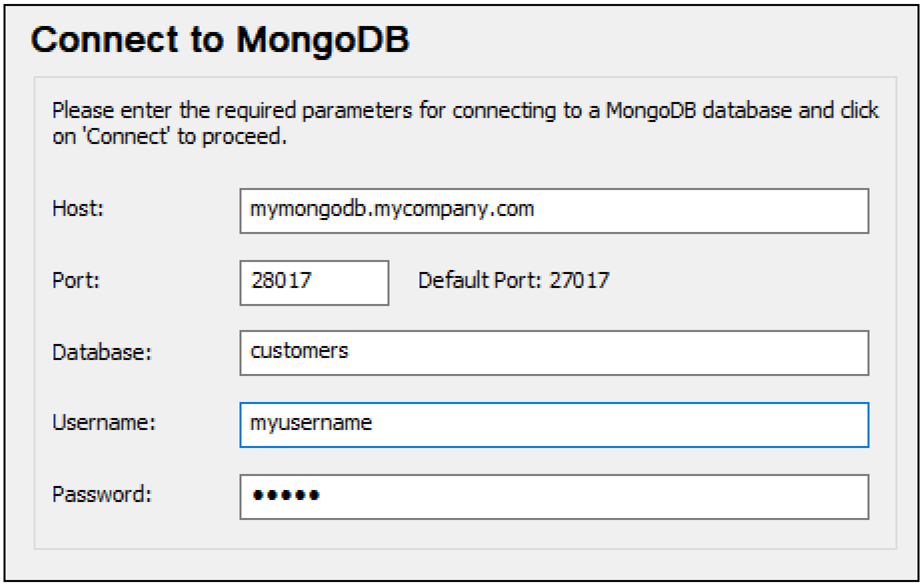
1. Run the database connection wizard (see Starting the Database Connection Wizard 165).
2. Select **SQLite**, and then click **Next**.
3. Browse for the SQLite database file, or enter the path (either relative or absolute) to the database. The **Connect** button becomes enabled once you enter the path to a SQLite database file.
4. Click **Connect**.

4.2.1.9 MongoDB Connection

To connect to an existing MongoDB, do the following.

1. Start the database connection wizard (*see* *Starting the Connection Wizard* 165).

2. Select MongoDB and click **Next**.
3. In the *Connect to MongoDB* screen (*screenshot below*), enter the connection details of the MongoDB.



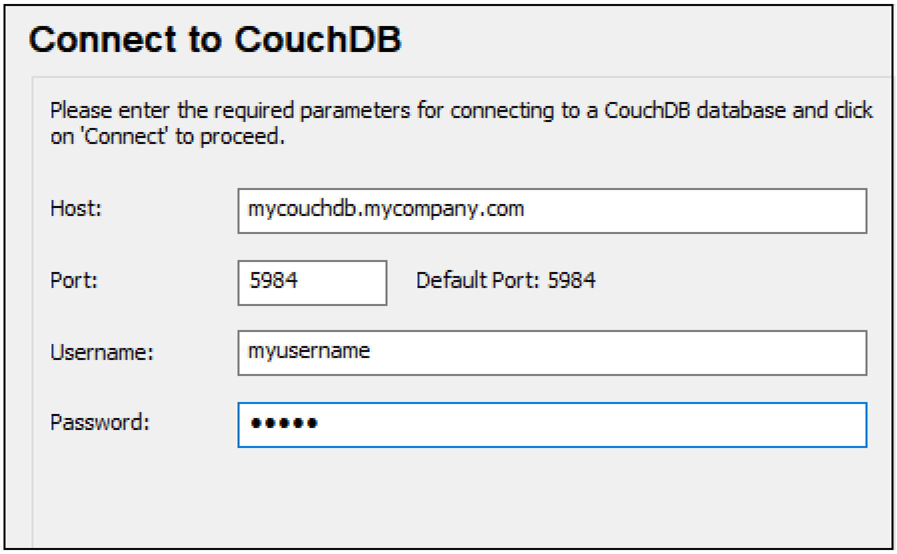
4. Click **Connect**.
5. The connection will be made and the *Insert Database Objects* screen appears, in which the document (record) collections of the selected database will be listed.
6. Select the collection/s that you want and click **OK**.

Note: If you use MongoDB Realm, you can access your database by means of Realm Sync. To find out more about how to get the connection URL, see the MongoDB Atlas documentation.

4.2.1.10 CouchDB Connection

To connect to an existing CouchDB, do the following.

1. Start the database connection wizard (*see Starting the Connection Wizard* 165).
2. Select **CouchDB**, and then click **Next**.
3. In the *Connect to CouchDB* screen that appears (*screenshot below*), enter the connection details of the CouchDB.



4. Click **Connect**.
5. The connection will be made and the *Insert Database Objects* screen appears, in which the tables of the database will be listed.
6. Select the table/s that you want and click **OK**.

4.2.1.11 Global Resources

After you have created a database as a global resource, its connection details are stored and can be used across all Altova products installed on your machine.

Create a database as a global resource

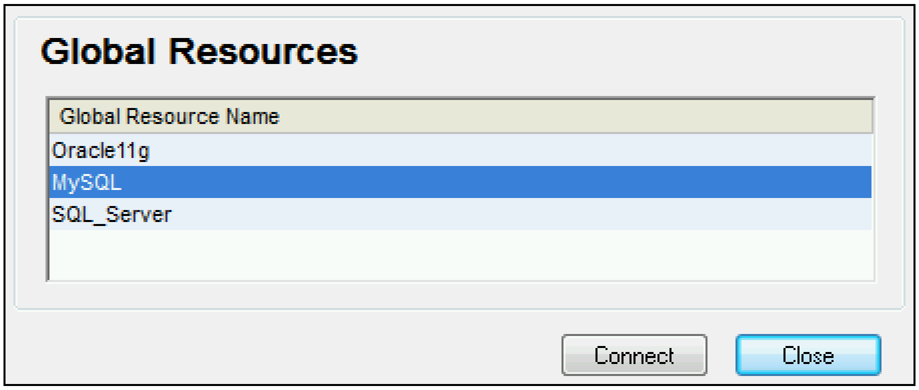
To create a database as a global resource, do the following

1. On the **Tools** menu of MapForce, click **Global Resources**.
2. Click **Add**, and then click Database.
3. Type in a name for the global resource in the *Resource Alias* field.
4. Click **Choose Database**. The Connection Wizard 165 appears.
5. Use the Connection Wizard to add a database connection as described above.

Use a global-resource database

To use a database that has been created as a global resource (*see above*), do the following:

1. Start the Connection Wizard as described above.
2. Select Global Resources. All the databases that have been created as global resources will be listed by their names in the Global Resources pane (*see screenshot below*).



3. Select the global resource that you want. Tip: Move the mouse cursor over a global resource in the list to see information about the database.

4.2.1.12 Database Connection Examples

This section includes examples for connecting to a database from MapForce through ADO, ODBC, or JDBC. The ADO.NET connection examples are listed separately, see Sample ADO.NET Connection Strings180. For instructions about establishing a native connection to PostgreSQL and SQLite, see Setting up a PostgreSQL Connection190 and Setting up a SQLite Connection192, respectively.

Note the following:

- The instructions may differ if your Windows configuration, network environment and the database client or server software are not the same as the ones described in each example.
- For most database types, it is possible to connect using more than one data access technology (ADO, ADO.NET, ODBC, JDBC) or driver. The performance of the database connection, as well as its features and limitations will depend on the selected driver, database client software (if applicable), and any additional connectivity parameters that you may have configured outside MapForce.

4.2.1.12.1 Firebird (JDBC)

This example illustrates how to connect to a Firebird database server through JDBC.

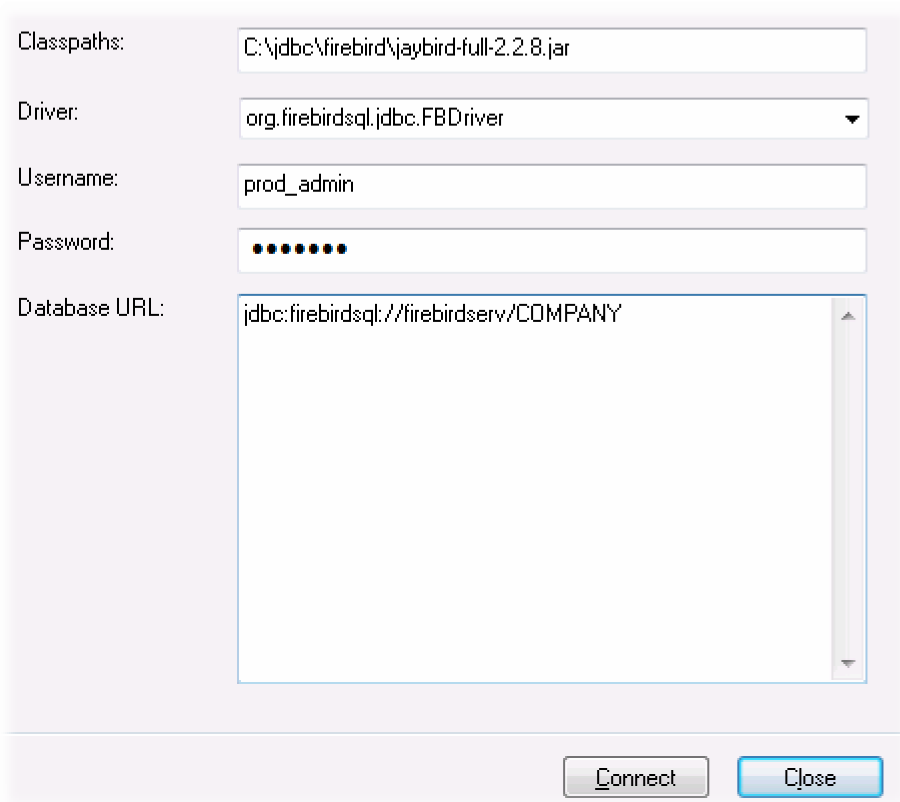
Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java Settings1476; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The Firebird JDBC driver must be available on your operating system (it takes the form of a .jar file which provides connectivity to the database). The driver can be downloaded from the Firebird website (https://www.firebirdsql.org/). This example uses *Jaybird 2.2.8*.

- You have the following database connection details: host, database path or alias, username, and password.

To connect to Firebird through JDBC:

1. Start the database connection wizard[165].
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: **C:\jdbc\firebird\jaybird-full-2.2.8.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH[188]).
4. In the "Driver" box, select **org.firebirdsql.jdbc.FBDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:firebirdsql://<host>[:<port>]/<database path or alias>
```

7. Click **Connect**.

4.2.1.12.2 Firebird (ODBC)

This example illustrates how to connect to a Firebird 2.5.4 database running on a Linux server.

Prerequisites:

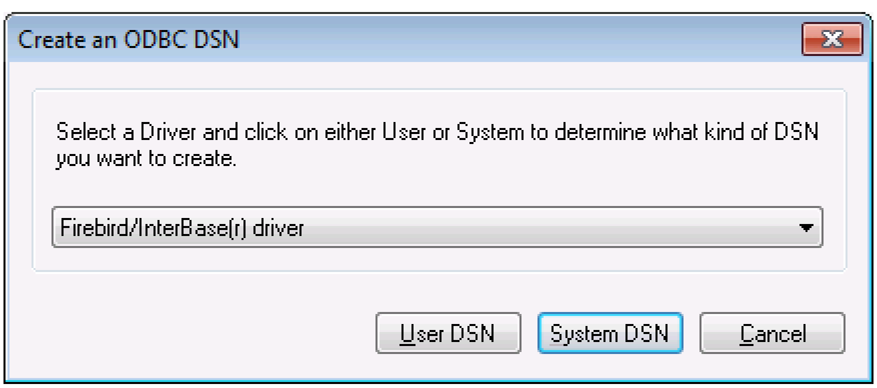
- The Firebird database server is configured to accept TCP/IP connections from clients.
- The Firebird ODBC driver must be installed on your operating system. This example uses the Firebird ODBC driver version 2.0.3.154 downloaded from the Firebird website (https://www.firebirdsql.org/).
- The Firebird client must be installed on your operating system. Note that there is no standalone installer available for the Firebird 2.5.4 client; the client is part of the Firebird server installation package. You can download the Firebird server installation package from the Firebird website (https://www.firebirdsql.org/), look for "Windows executable installer for full Superclassic/Classic or Superserver". To install only the client files, choose "**Minimum client install - no server, no tools**" when going through the wizard steps.

> **Important:**
>
> - The platform of both the Firebird ODBC driver and client (32-bit or 64-bit) must correspond to that of MapForce.
> - The version of the Firebird client must correspond to the version of Firebird server to which you are connecting.

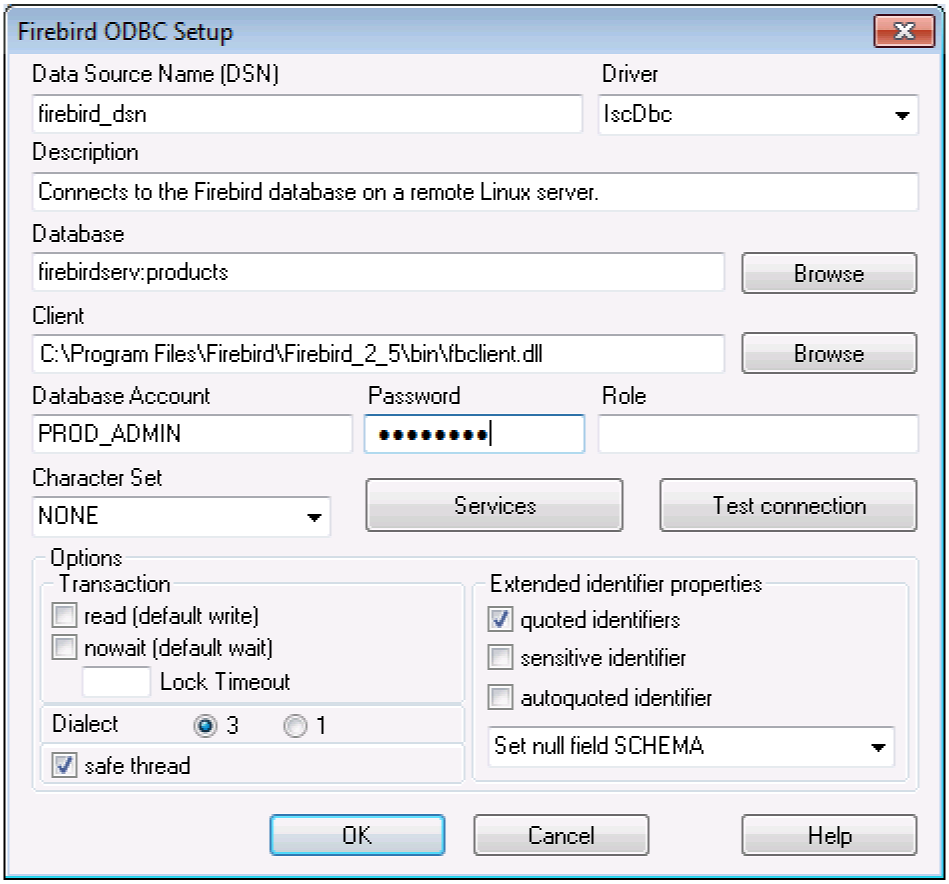
- You have the following database connection details: server host name or IP address, database path (or alias) on the server, user name, and password.

To connect to Firebird via ODBC:

1. Start the database connection wizard [165].
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .



4. Select the Firebird driver, and then click **User DSN** (or **System DSN**, depending on what you selected in the previous step). If the Firebird driver is not available in the list, make sure that it is installed on your operating system (see also Viewing the Available ODBC Drivers [184]).



5. Enter the database connection details as follows:

Data Source Name (DSN)	Enter a descriptive name for the data source you are creating.

Database	Enter the server host name or IP address, followed by a colon, followed by the database alias (or path). In this example, the host name is `firebirdserv`, and the database alias is `products`, as follows:
	
	`firebirdserv:products`
	
	Using a database alias assumes that, on the server side, the database administrator has configured the alias *products* to point to the actual Firebird (.fdb) database file on the server (see the Firebird documentation for more details).
	
	You can also use the server IP address instead of the host name, and a path instead of an alias; therefore, any of the following sample connection strings are valid:
	
	`firebirdserver:/var/Firebird/databases/butterflies.fdb`
	`127.0.0.1:D:\Misc\Lenders.fdb`

	If the database is on the local Windows machine, click **Browse** and select the Firebird (.fdb) database file directly.
Client	Enter the path to the **fbclient.dll** file. By default, this is the `bin` subdirectory of the Firebird installation directory.
Database Account	Enter the database user name supplied by the database administrator (in this example, `PROD_ADMIN`).
Password	Enter the database password supplied by the database administrator.

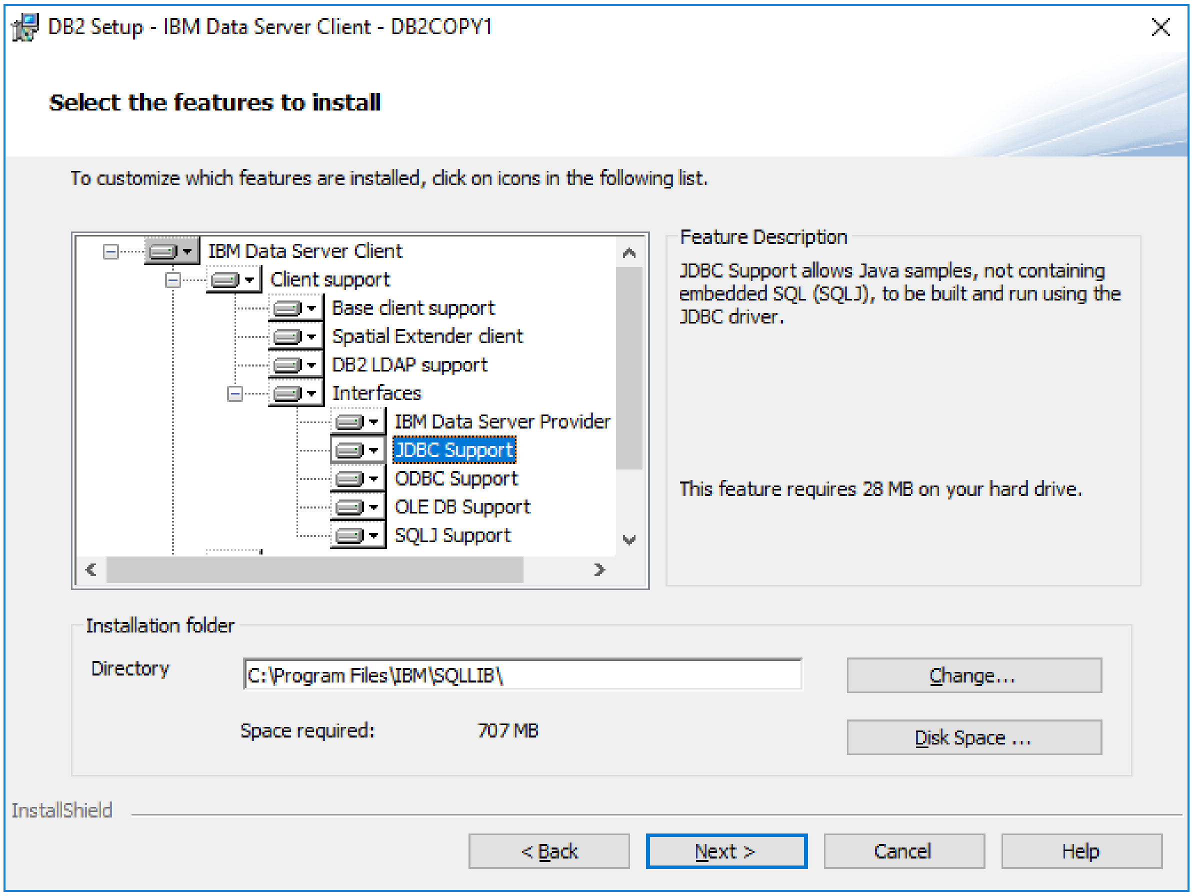
6. Click **OK**.

4.2.1.12.3 IBM DB2 (JDBC)

This example illustrates how to connect to an **IBM DB2** database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java Settings 1476; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK. This example uses Oracle's OpenJDK 11.0 64-bit, and, consequently, the 64-bit version of MapForce.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. This example uses the JDBC driver available after installing the **IBM Data Server Client** version 10.1 (64-bit). For the JDBC drivers to be installed, choose a **Typical** installation, or select this option explicitly on the installation wizard.

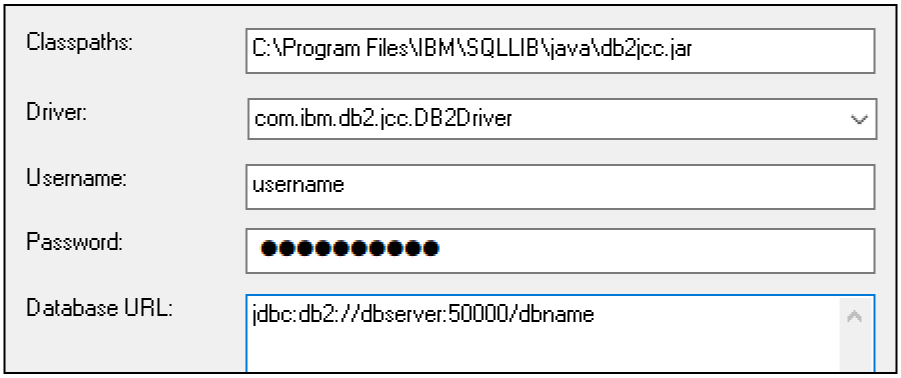


If you did not change the default installation path, the required .jar files will be in the **C:\Program Files\IBM\SQLLIB\java** directory after installation.

- You need the following database connection details: host, port, database name, username, and password.

To connect to IBM DB2 through JDBC:

1. Start the database connection wizard 165.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. This examples refers to **C:\Program Files\IBM\SQLLIB\java\db2jcc.jar**. You may need to refer to the **db2jcc4.jar** driver, depending on the database server version. For driver compatibility, refer to IBM documentation (http://www-01.ibm.com/support/docview.wss?uid=swg21363866). Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH 188).
4. In the "Driver" box, select **com.ibm.db2.jcc.DB2Driver**. This entry becomes available only if a valid .jar file path was found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



5. Enter the username and password of the database user in the corresponding text boxes.
6. Enter the JDBC connection string in the **Database URL** text box. Make sure to replace the connection details with the ones applicable to your database server.

```
jdbc:db2://hostName:port/databaseName
```

7. Click **Connect**.

4.2.1.12.4 IBM DB2 (ODBC)

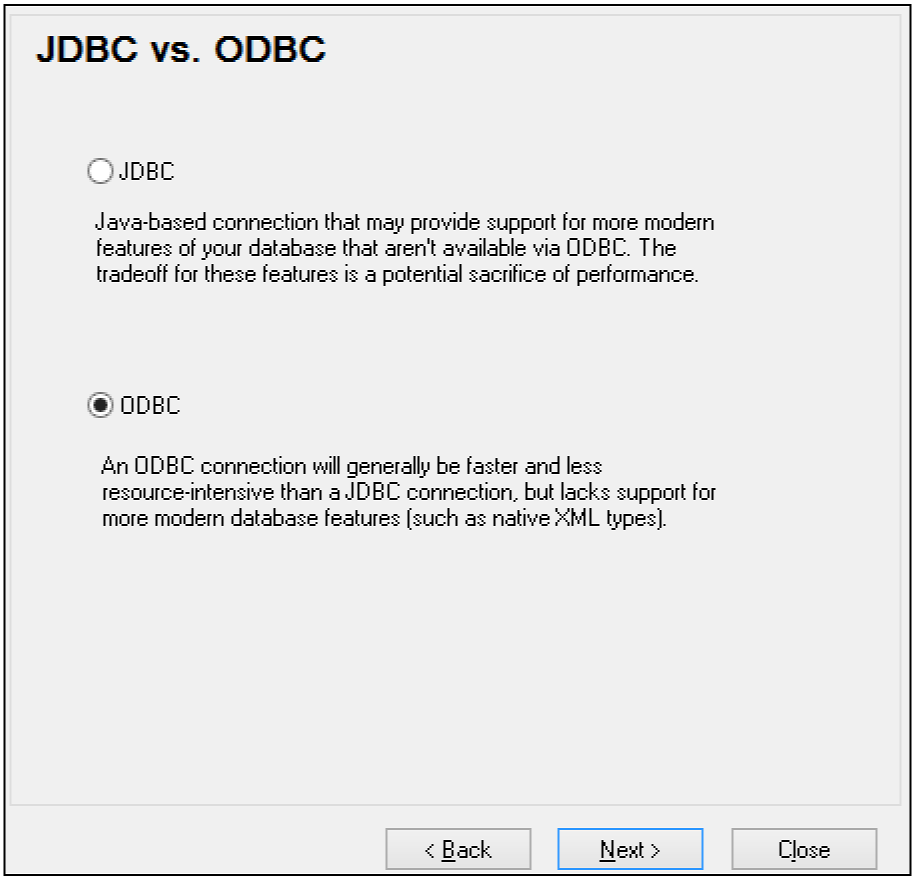
This example illustrates how to connect to an IBM DB2 database through ODBC.

Prerequisites:

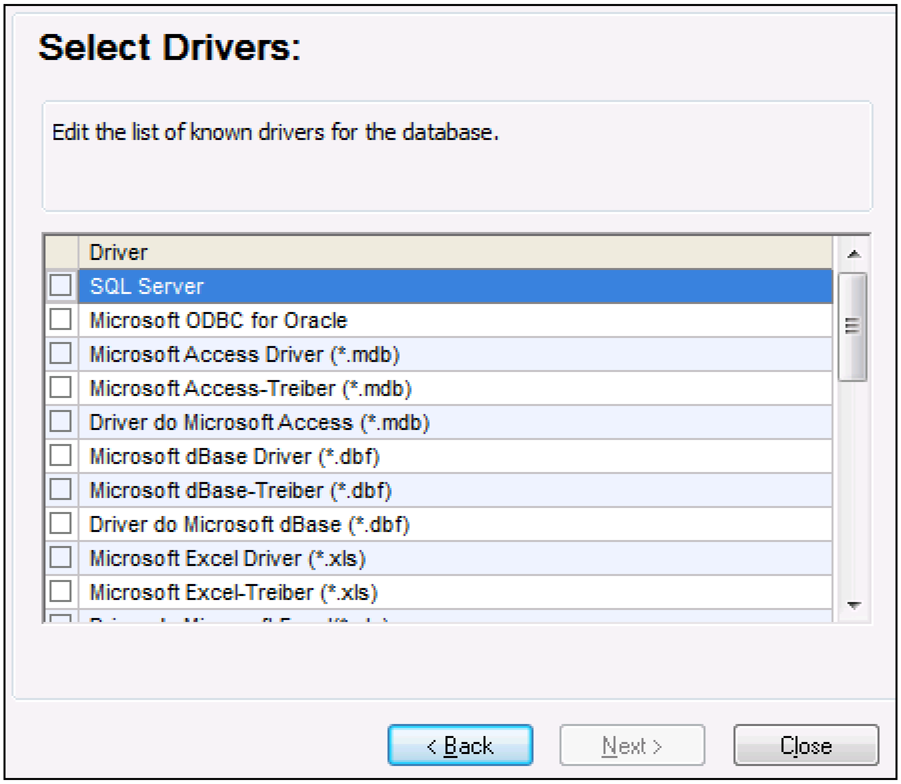
- IBM Data Server Client must be installed and configured on your operating system (this example uses IBM Data Server Client 9.7). For installation instructions, check the documentation supplied with your IBM DB2 software. After installing the IBM Data Server Client, check if the ODBC drivers are available on your machine (see Viewing the Available ODBC Drivers [184]).
- Create a database alias. There are several ways to do this:
  - o From IBM DB2 Configuration Assistant
  - o From IBM DB2 Command Line Processor
  - o From the ODBC data source wizard (for this case, the instructions are shown below)
- You have the following database connection details: host, database, port, username, and password.

To connect to IBM DB2:

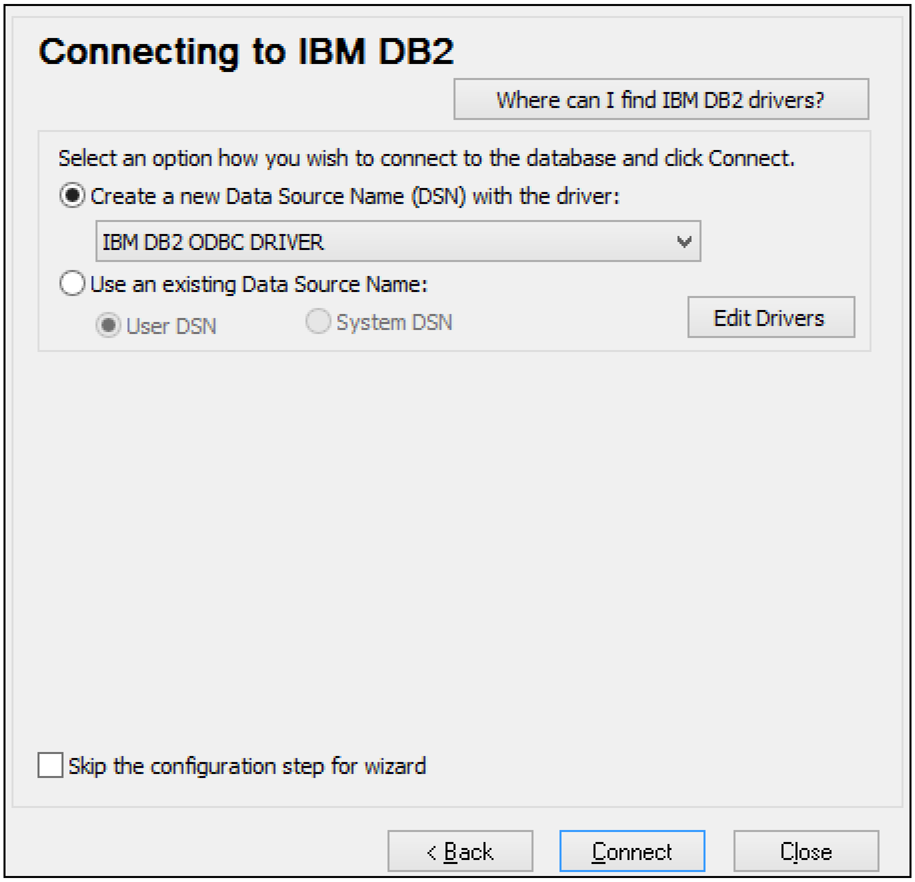
1. Start the database connection wizard [165] and select **IBM DB2 (ODBC/JDBC)**.
2. Click **Next**.



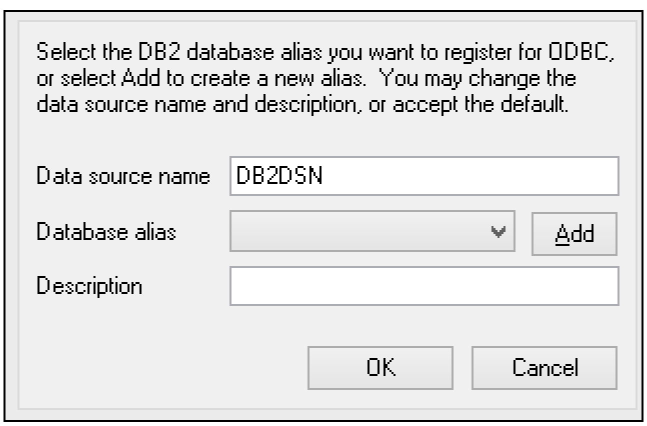
3. Select **ODBC**, and click **Next**. If prompted to edit the list of known drivers for the database, select the database drivers applicable to IBM DB2 (see Prerequisites 201), and click **Next**.



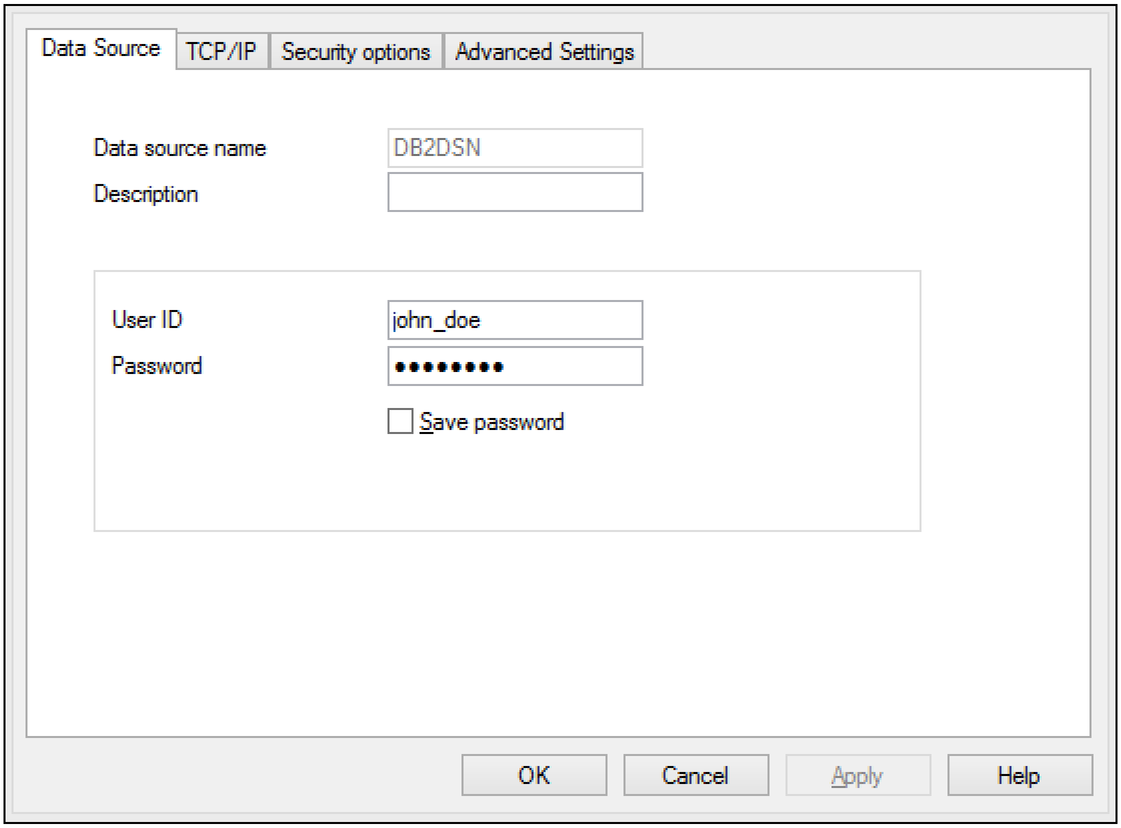
4. Select the IBM DB2 driver from the list, and then click **Connect**. (To edit the list of available drivers, click **Edit Drivers**, and then check or uncheck the IBM DB2 drivers you wish to add or remove, respectively.)



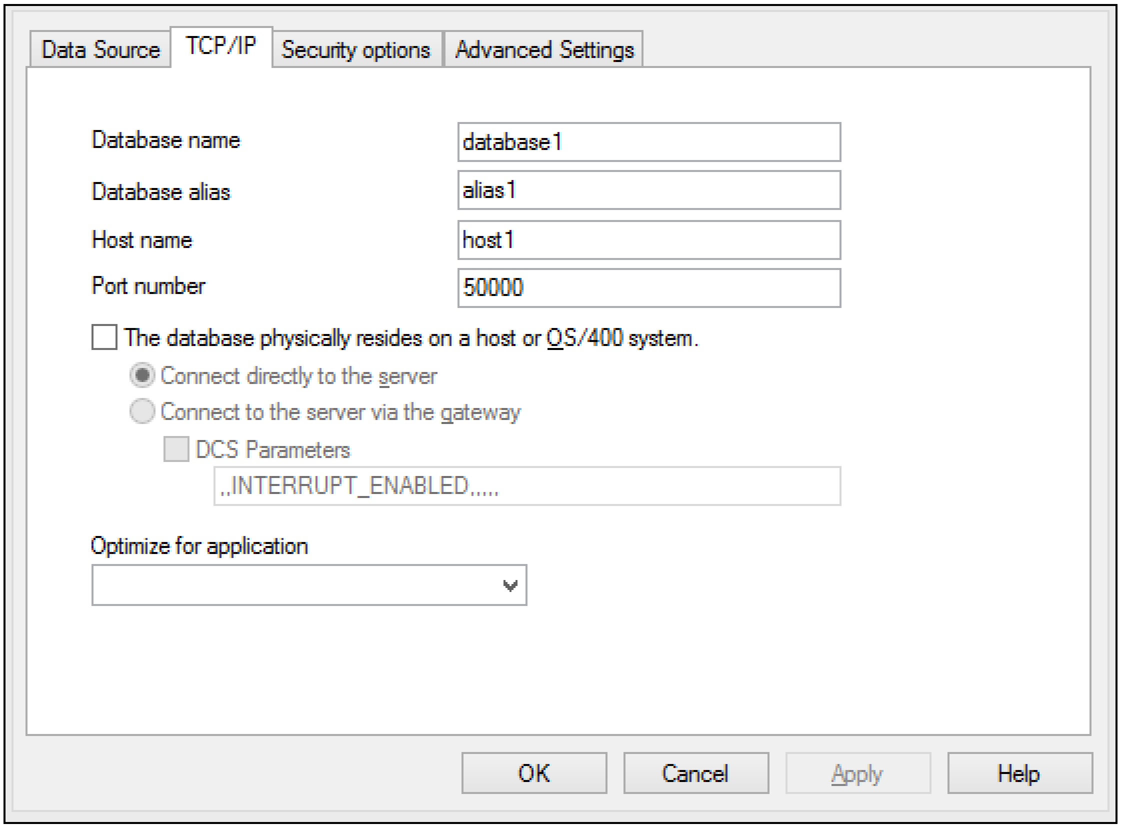
5. Enter a data source name (in this example, **DB2DSN**), and then click **Add**.



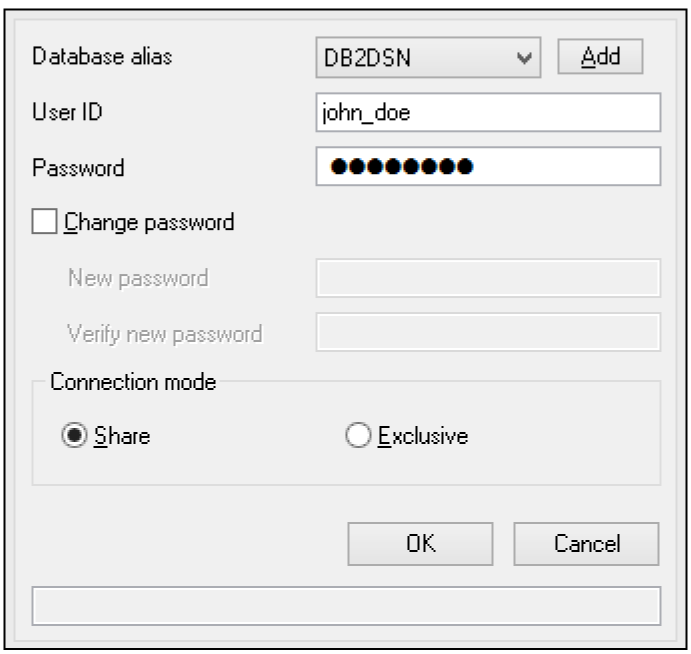
6. On the **Data Source** tab, enter the user name and password to the database.



7. On the **TCP/IP** tab, enter the database name, a name for the alias, the host name and the port number, and then click OK.



8. Enter again the username and password, and then click **OK**.



4.2.1.12.5 IBM DB2 for i (JDBC)

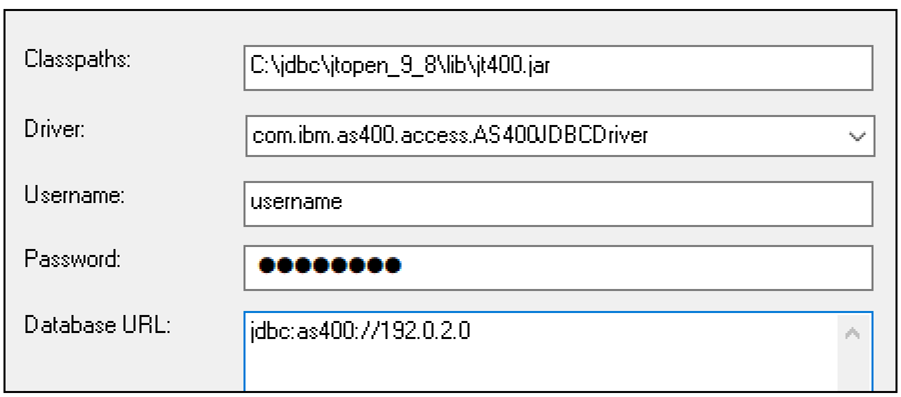
This example illustrates how to connect to an **IBM DB2 for i** database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see [Java Settings](#)[1476]; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK. This example uses Oracle's OpenJDK 11.0 64-bit, and, consequently, the 64-bit version of MapForce.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. This example uses the open source **Toolbox for Java/JTOpen** version 9.8 (http://jt400.sourceforge.net/). After you download the package and unpack to a local directory, the required .jar files will be available in the **lib** subdirectory.
- You need the following database connection details: host, username, and password.

To connect to IBM DB2 for i through JDBC:

1. Start the database connection wizard[165].
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. In this example, the required .jar file is at the following path: **C:\jdbc\jtopen_9_8\jt400.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH[188]).
4. In the "Driver" box, select **com.ibm.as400.access.AS400JDBCDriver**. This entry becomes available only if a valid .jar file path was found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



5. Enter the username and password of the database user in the corresponding text boxes.
6. Enter the JDBC connection string in the **Database URL** text box. Make sure to replace **`host`** with the host name or IP address of your database server.

```
jdbc:as400://host
```

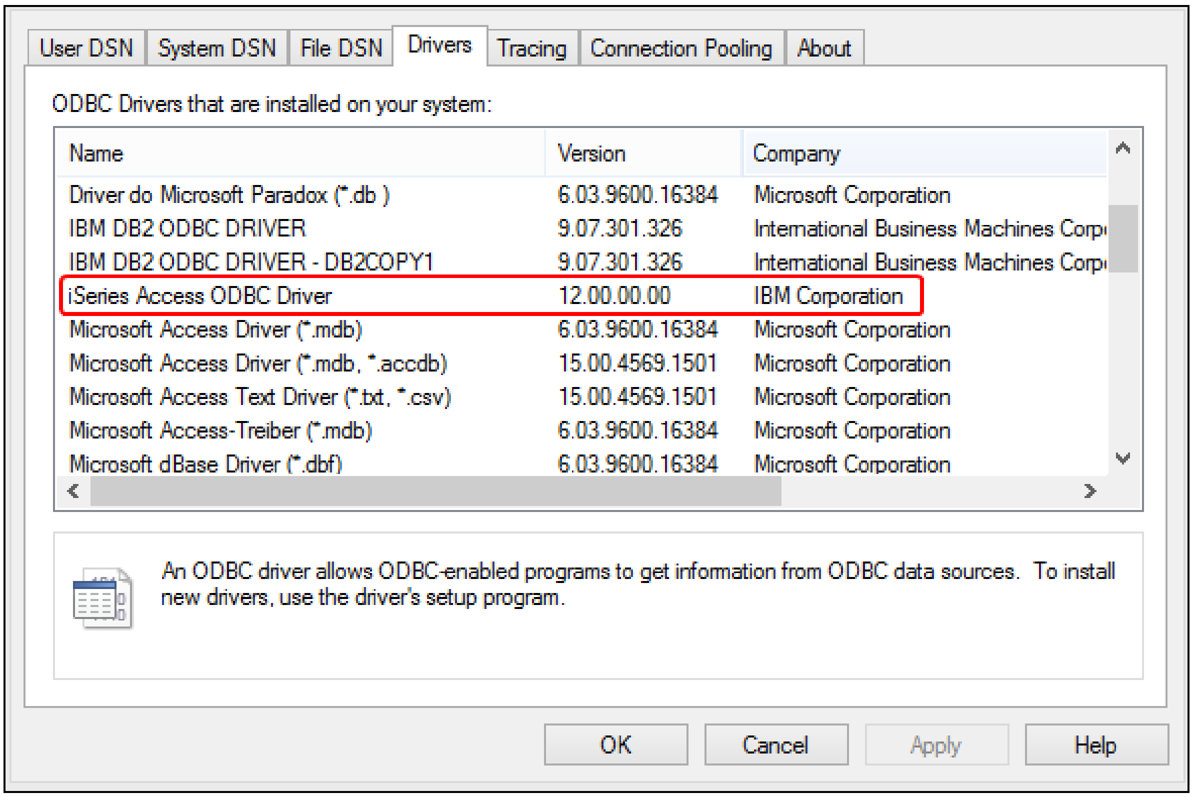
7. Click **Connect**.

4.2.1.12.6 IBM DB2 for i (ODBC)

This example illustrates how to connect to an *IBM DB2 for i* database through ODBC.

Prerequisites:

- *IBM System i Access for Windows* must be installed on your operating system (this example uses *IBM System i Access for Windows V6R1M0*). For installation instructions, check the documentation supplied with your *IBM DB2 for i* software. After installation, check if the ODBC driver is available on your machine (see Viewing the Available ODBC Drivers[184]).

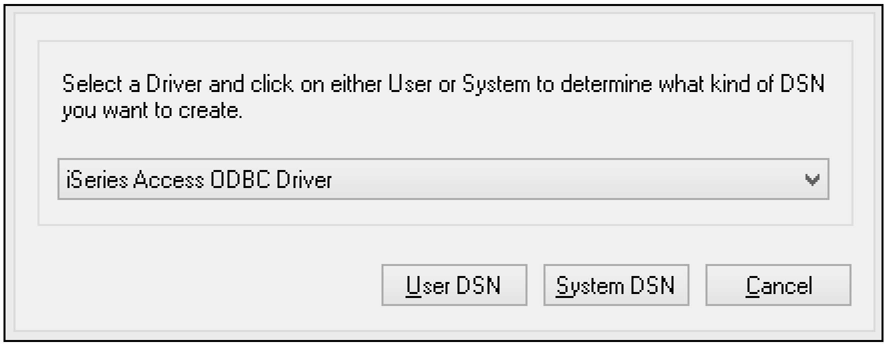


- You have the following database connection details: the I.P. address of the database server, database user name, and password.
- Run *System i Navigator* and follow the wizard to create a new connection. When prompted to specify a system, enter the I.P. address of the database server. After creating the connection, it is recommended to verify it (click on the connection, and select **File** > **Diagnostics** > **Verify Connection**). If you get connectivity errors, contact the database server administrator.

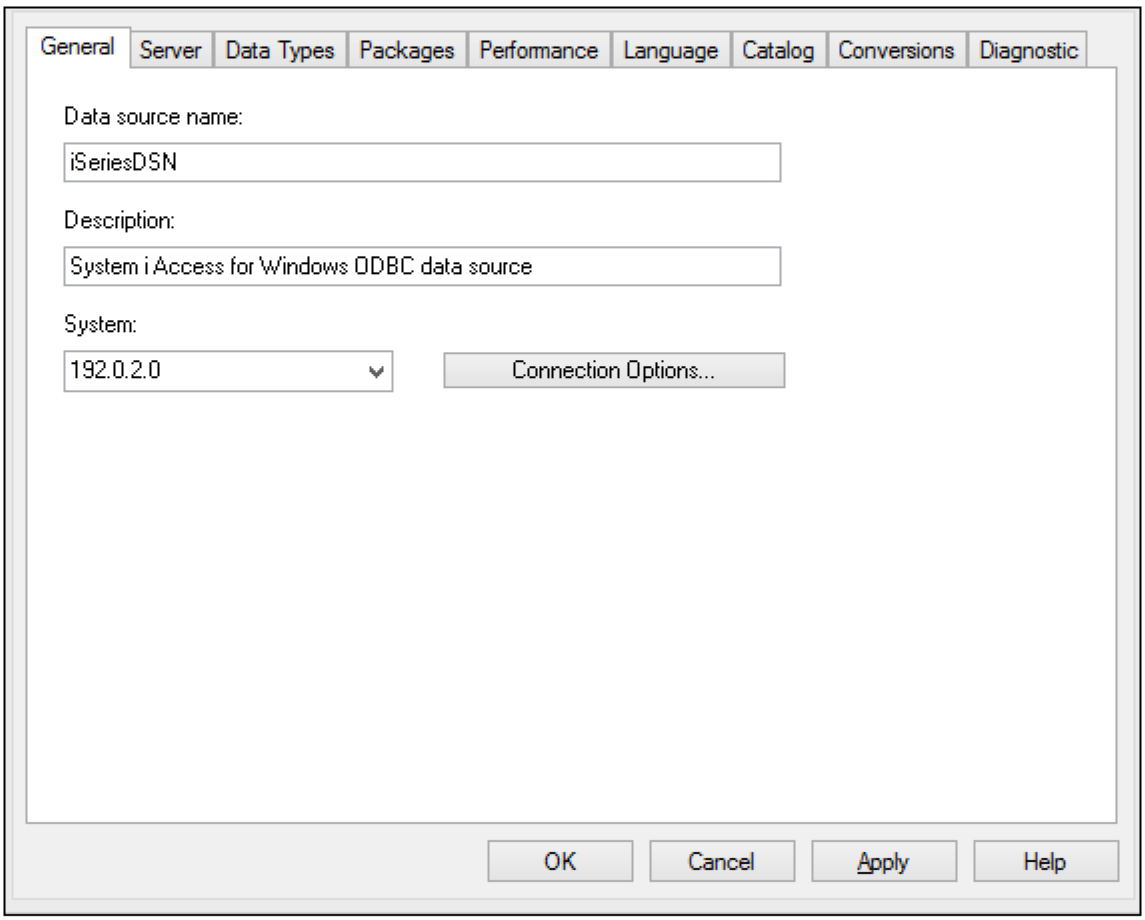
To connect to IBM DB2 for i:

1. Start the database connection wizard[165].
2. Click **ODBC connections**.

3. Click **User DSN** (alternatively, click **System DSN**, or **File DSN**, in which case the subsequent instructions will be similar).
4. Click **Add**  .
5. Select the **iSeries Access ODBC Driver** from the list, and click **User DSN** (or **System DSN**, if applicable).

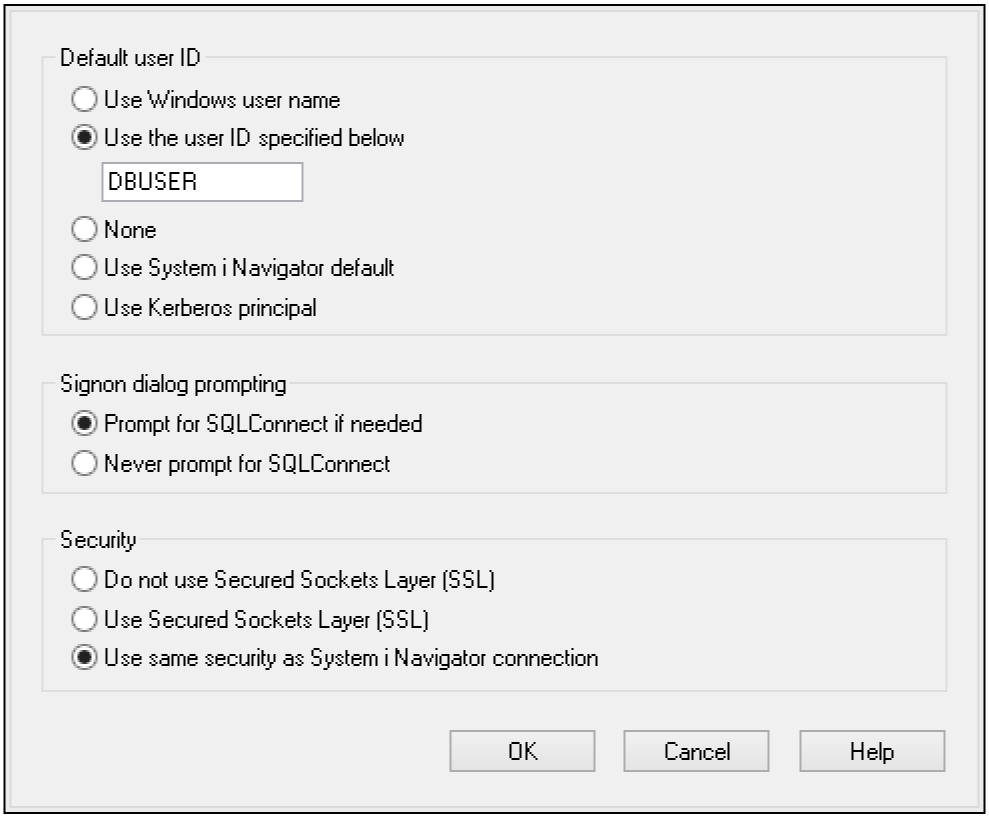


6. Enter a data source name and select the connection from the System combo box. In this example, the data source name is **iSeriesDSN** and the System is **192.0.2.0**.



Note: When adding an ODBC data source for an *IBM DB2 for i* database, a default flag is set which enables query timeouts. This setting must be disabled for MapForce to correctly load mapping files. To disable the setting, select the **Performance** tab, click **Advanced**, and clear the **Allow query timeout** check box.

7. Click Connection Options, select **Use the User ID specified below** and enter the name of the database user (in this example, **DBUSER**).



8. Click **OK**. The new data source becomes available in the list of DSNs.
9. Click **Connect**.
10. Enter the user name and password to the database when prompted, and then click **OK**.

4.2.1.12.7 IBM Informix (JDBC)

This example illustrates how to connect to an IBM Informix database server through JDBC.

Prerequisites:

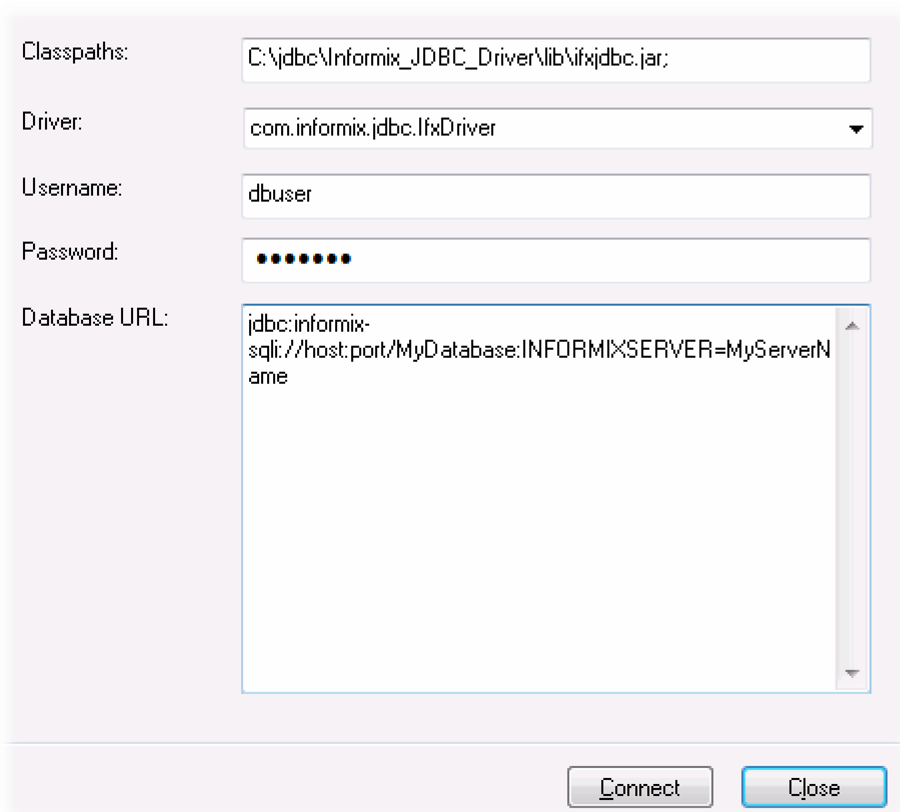
- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java Settings 1476; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. In this example, IBM Informix JDBC driver version 3.70 is used. For the driver's

installation instructions, see the documentation accompanying the driver or the "IBM Informix JDBC Driver Programmer's Guide").

- You have the following database connection details: host, name of the Informix server, database, port, username, and password.

To connect to IBM Informix through JDBC:

1. Start the database connection wizard 165.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: **C:\Informix_JDBC_Driver\lib\ifxjdbc.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH 188).
4. In the "Driver" box, select **com.informix.jdbc.IfxDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:informix-sqli://hostName:port/databaseName:INFORMIXSERVER=myserver;
```

7. Click **Connect**.

4.2.1.12.8 MariaDB (ODBC)

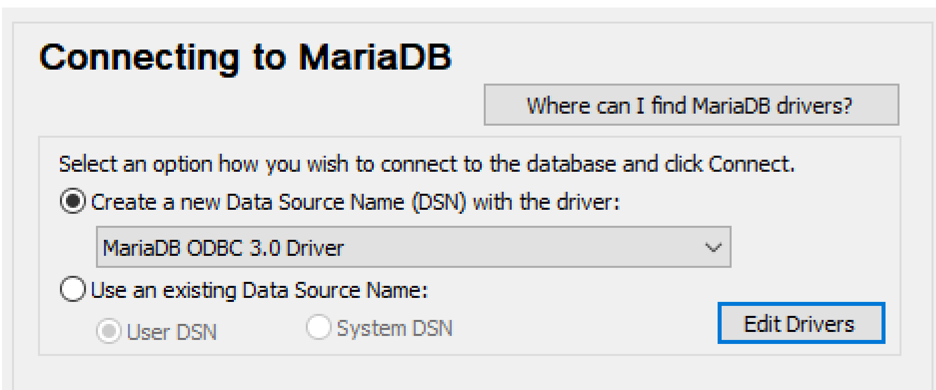
This example illustrates how to connect to a MariaDB database server through ODBC.

Prerequisites:

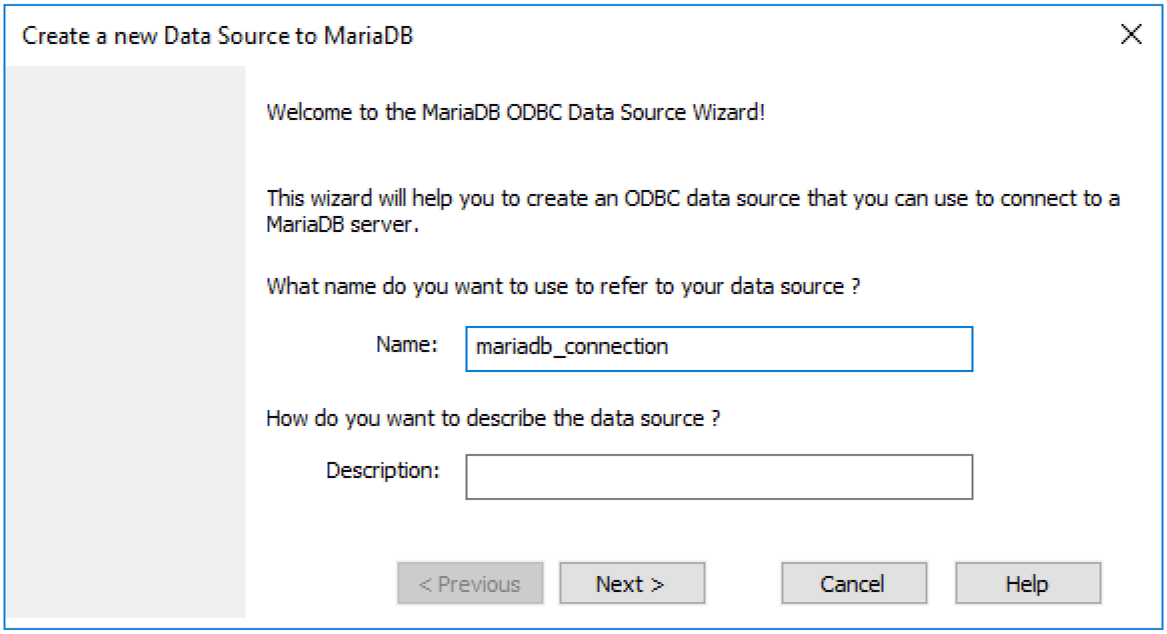
- The MariaDB Connector/ODBC (https://downloads.mariadb.org/connector-odbc/) must be installed.
- You have the following database connection details: host, database, port, username, and password.

To connect to MariaDB through ODBC:

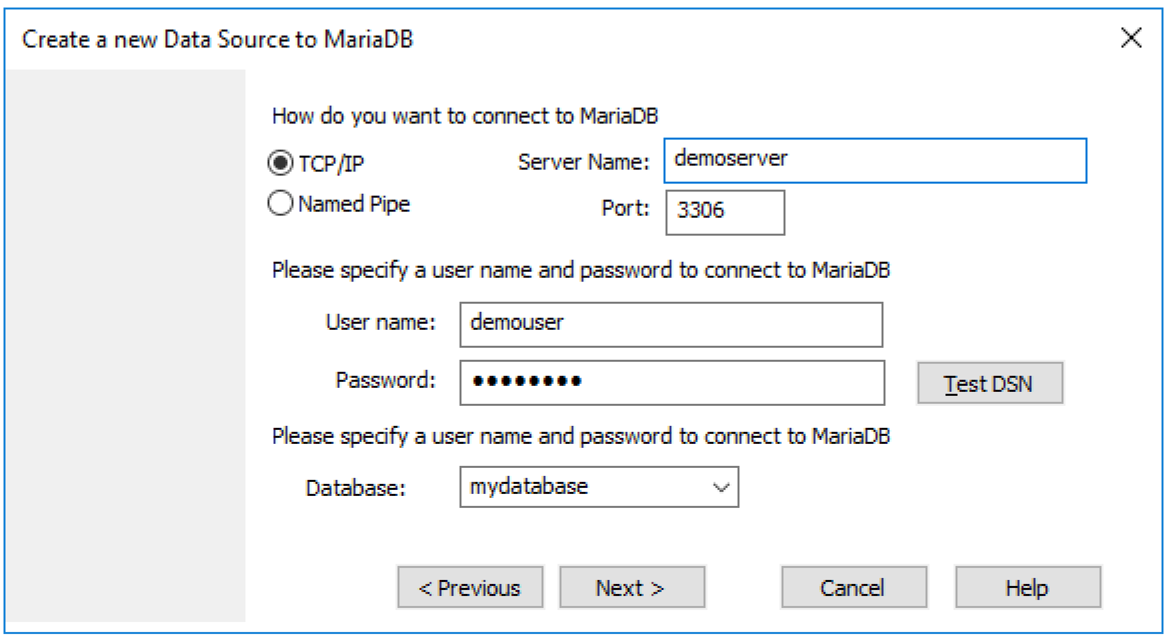
1. Start the database connection wizard 165.
2. Select **MariaDB (ODBC)**, and then click **Next**.



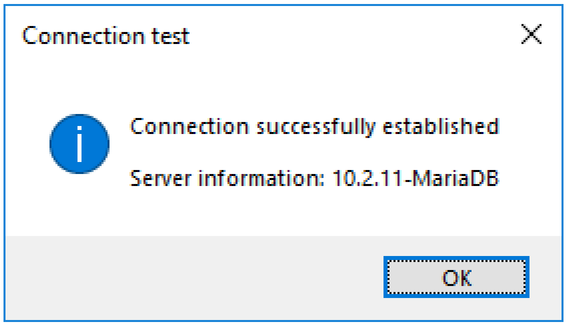
3. Select **Create a new Data Source Name (DSN) with the driver**, and choose **MariaDB ODBC 3.0 Driver**. If no such driver is available in the list, click **Edit Drivers**, and select any available MariaDB drivers (the list contains all ODBC drivers installed on your operating system).
4. Click **Connect**.



5. Enter name and, optionally, a description that will help you identify this ODBC data source in future.



6. Fill in the database connection credentials (TCP/IP Server, User, Password), select a database, and then click **Test DSN**. Upon successful connection, a message box appears:



7. Click **Next** and complete the wizard. Other parameters may be required, depending on the case (for example, SSL certificates if you are connecting to MariaDB through a secure connection).

Note: If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address.

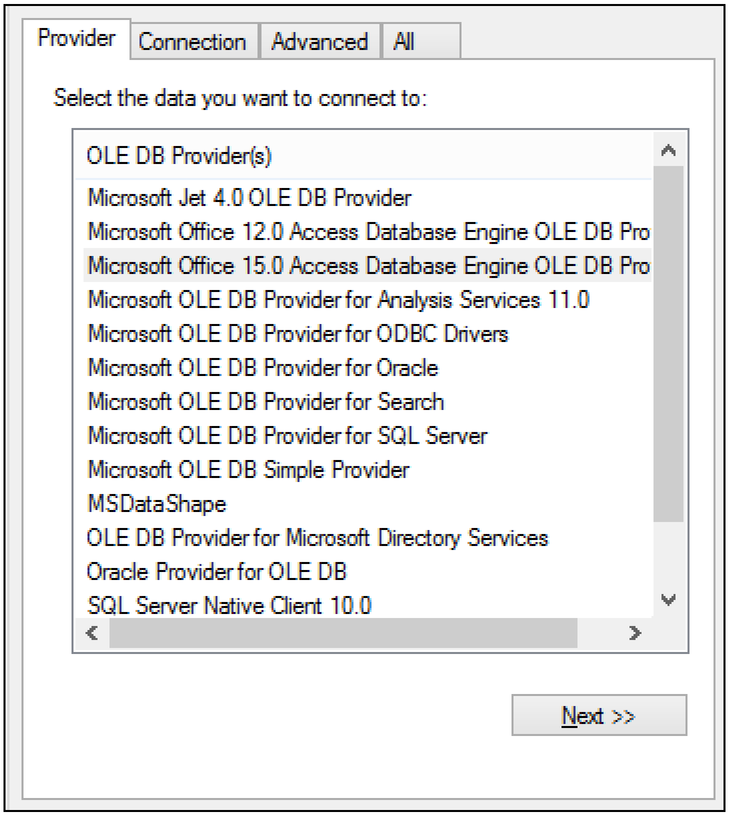
4.2.1.12.9 Microsoft Access (ADO)

A simple way to connect to a Microsoft Access database is to follow the wizard and browse for the database file, as shown in Connecting to an Existing Microsoft Access Database 173. An alternative approach is to set up an ADO connection explicitly, as shown in this topic. This approach is useful if your database is password-protected.

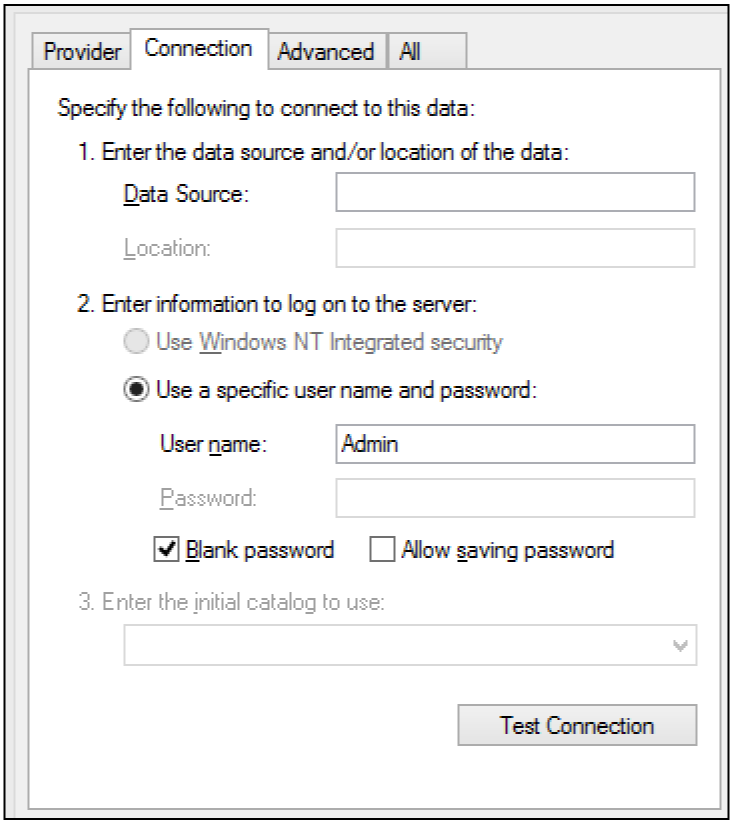
It is also possible to connect to Microsoft Access through an ODBC connection, but it has limitations, so it is best to avoid it.

To connect to a password-protected Microsoft Access database:

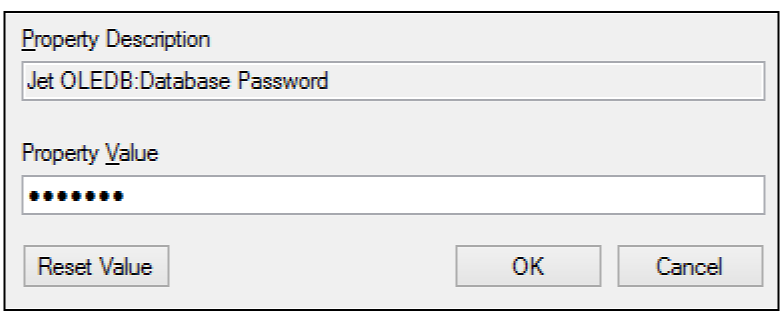
1. Start the database connection wizard 165.
2. Click **ADO Connections**.
3. Click **Build**.



4. Select the **Microsoft Office 15.0 Access Database Engine OLE DB Provider**, and then click **Next**.



5. In the Data Source box, enter the path to the Microsoft Access file in UNC format, for example, **\\myserver\\mynetworkshare\Reports\Revenue.accdb**, where **myserver** is the name of the server and **mynetworkshare** is the name of the network share.
6. On the **All** tab, double click the **Jet OLEDB:Database Password** property and enter the database password as property value.



Note: If you are still unable to connect, locate the workgroup information file (**System.MDW**) applicable to your user profile, and set the value of the **Jet OLEDB: System database** property to the path of the **System.MDW** file.

4.2.1.12.10 Microsoft SQL Server (ADO)

This example illustrates how to connect to a SQL Server database through ADO. These instructions are applicable when you use the recommended **Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL)**, which is available for download at https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15.

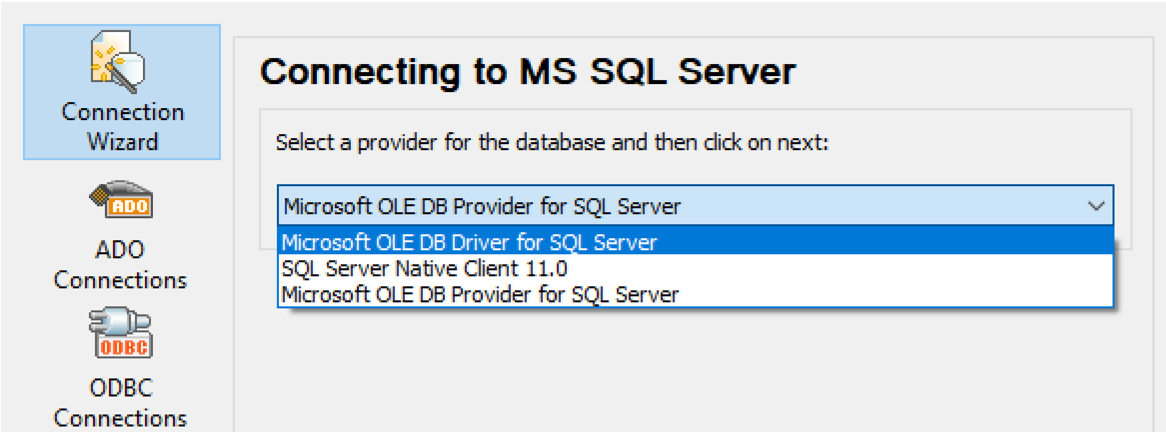
> Before following these instructions, make sure that you have downloaded and installed the provider above on your workstation. The ADO provider must match the platform of MapForce (32-bit or 64-bit).

If you would like to use other ADO providers such as **SQL Server Native Client (SQLNCLI)** or **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)**, the instructions are similar, but these providers are deprecated and thus not recommended. Also, for the connection to be successful with a deprecated provider, you may need to set additional connection properties as described in Setting up the SQL Server Data Link Properties [173].

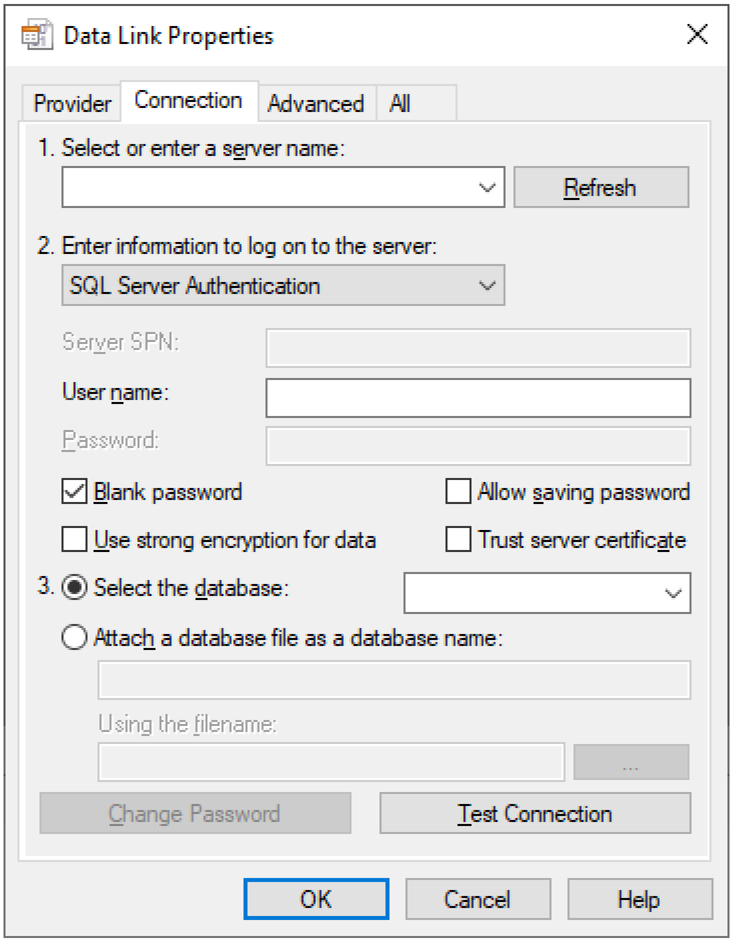
> The **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)** is known to have issues with parameter binding of complex queries like Common Table Expressions (CTE) and nested SELECT statements.

To connect to SQL Server:

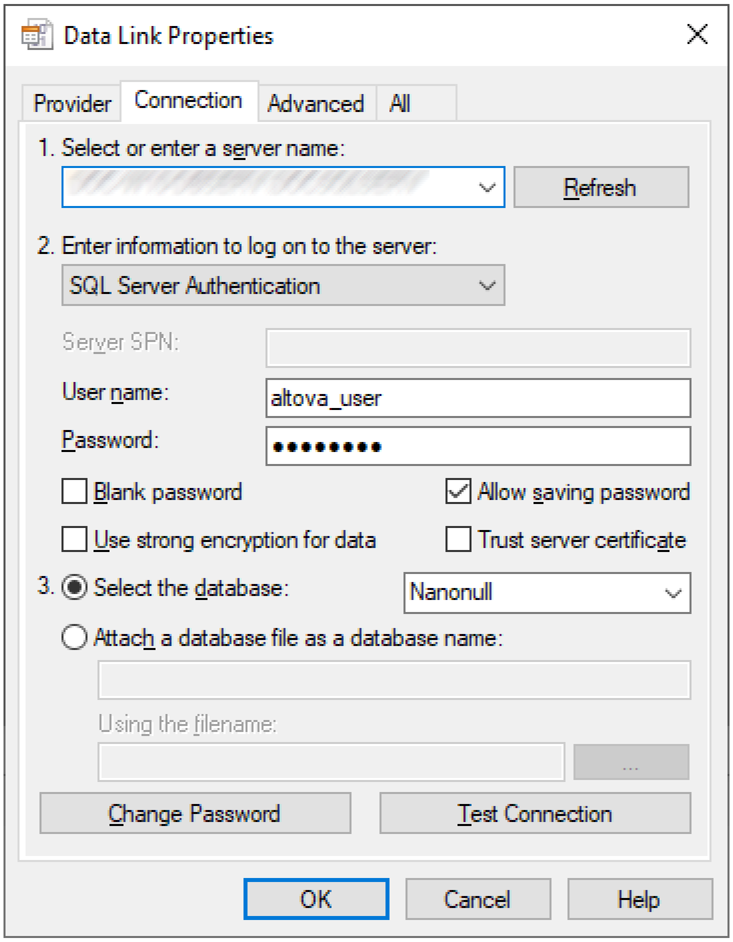
1. Start the database connection wizard [165].
2. Select **Microsoft SQL Server (ADO)**, and then click **Next**. The list of available ADO providers is displayed. In this example, the **Microsoft OLE DB Driver for SQL Server** is used. If it's not in the list, make sure that it is installed on your computer, as mentioned above.



3. Click **Next**. The Data Link Properties dialog box appears.



4. Select or enter the name of the database server, for example, **SQLSERV01**. If you are connecting to a named SQL Server instance, the server name looks like **SQLSERV01\SOMEINSTANCE**.
5. If the database server was configured to allow connections from users authenticated on the Windows domain, select **Windows Authentication**. Otherwise, select **SQL Server Authentication**, clear the **Blank password** check box, and enter the database credentials in the relevant boxes.
6. Select the **Allow saving password** check box and the database to which you are connecting (in this example, "Nanonull").



7. To test the connection at this time, click **Test Connection**. This is an optional, recommended step.
8. Click **OK**.

4.2.1.12.11 Microsoft SQL Server (ODBC)

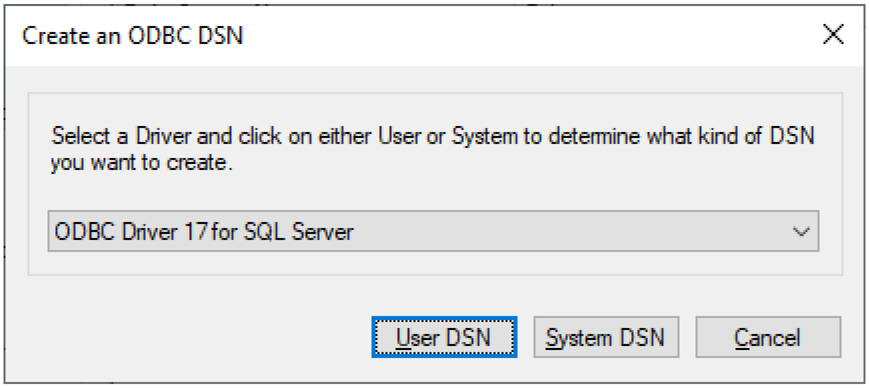
This example illustrates how to connect to a SQL Server database through ODBC.

Prerequisites:

- Download and install the **Microsoft ODBC Driver for SQL Server** from the Microsoft website, see https://docs.microsoft.com/en-us/SQL/connect/odbc/download-odbc-driver-for-sql-server. This example uses **Microsoft ODBC Driver 17 for SQL Server** to connect to a **SQL Server 2016** database. You might want to download a different ODBC driver version, depending on the version of SQL Server where you want to connect. For information about ODBC driver versions supported by your SQL Server database, refer to the driver's system requirements.

To connect to SQL Server using ODBC:

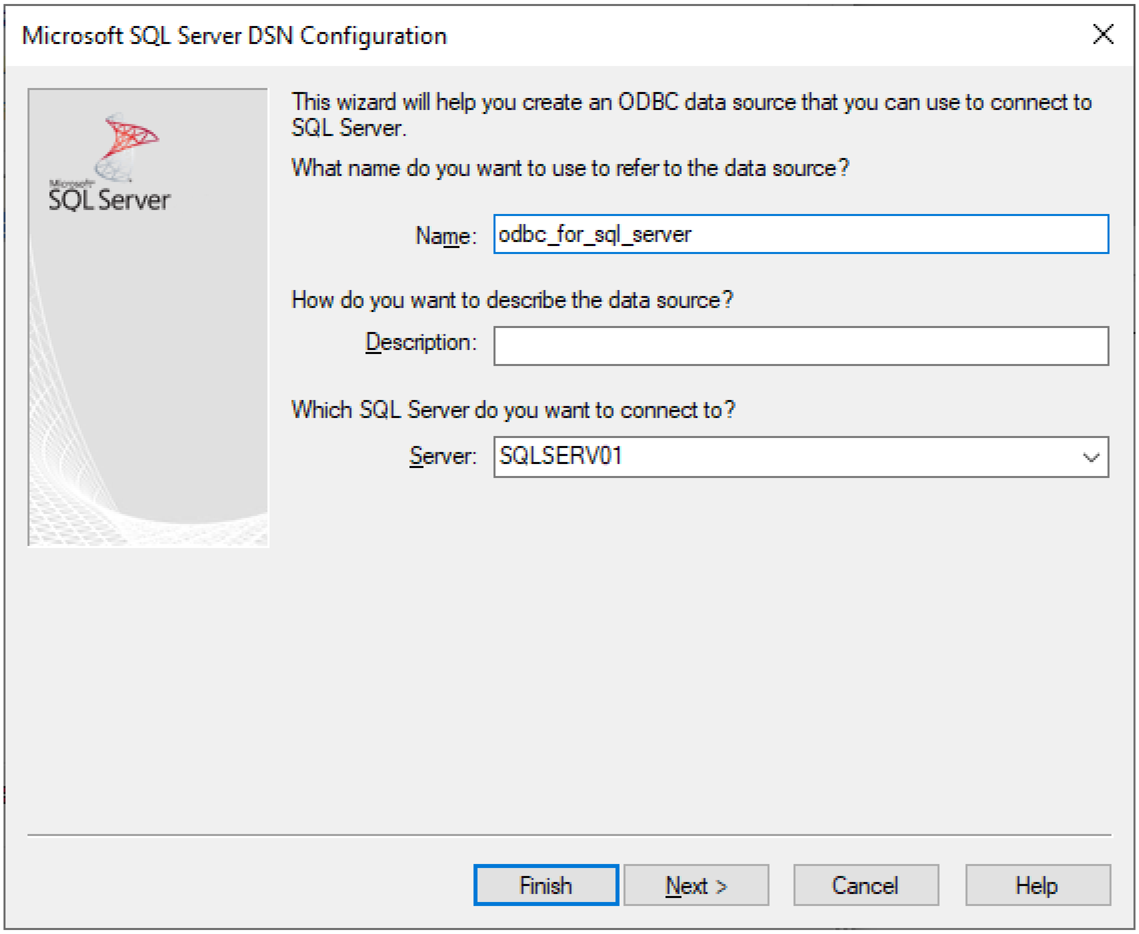
1. Start the database connection wizard 165.
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .
4. Select the driver from the list. Note that the driver appears in the list only after it has been installed.



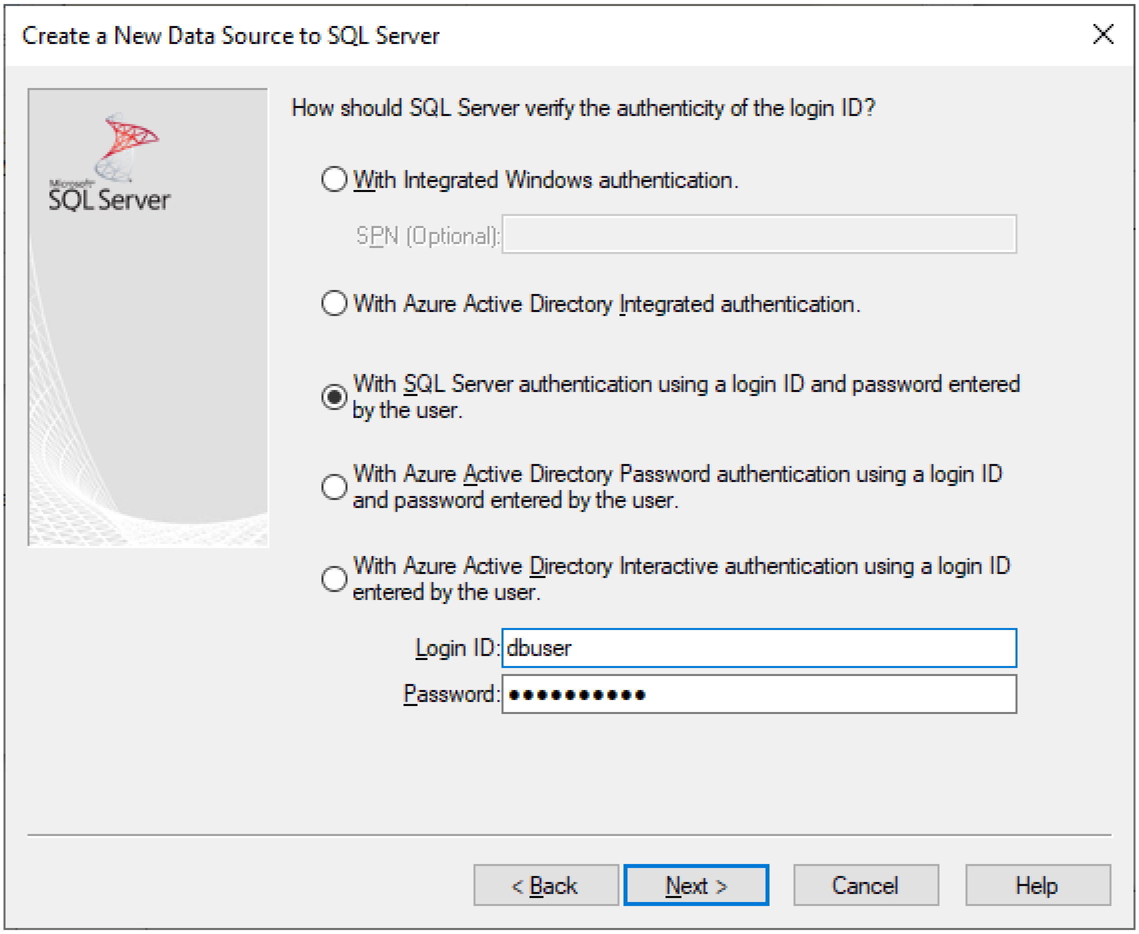
5. Click **User DSN** (or **System DSN** if you are creating a System DSN).

> Creating a **System DSN** requires that MapForce be run as an administrator. Therefore, in order to create a **System DSN**, cancel the wizard, make sure that you run MapForce as an administrator, and perform the steps above again.

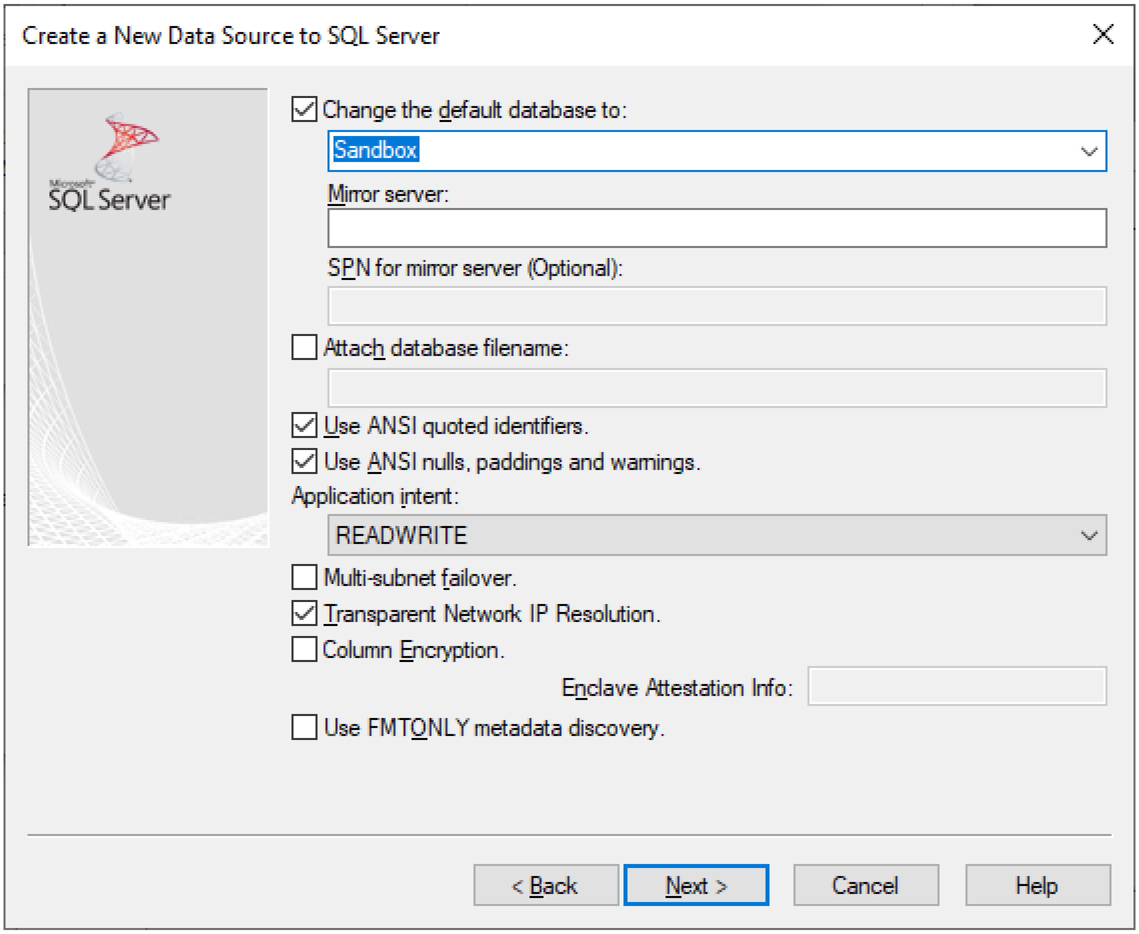
6. Enter a name and, optionally, a description to identify this connection, and then select from the list the SQL Server to which you are connecting (**SQLSERV01** in this example).



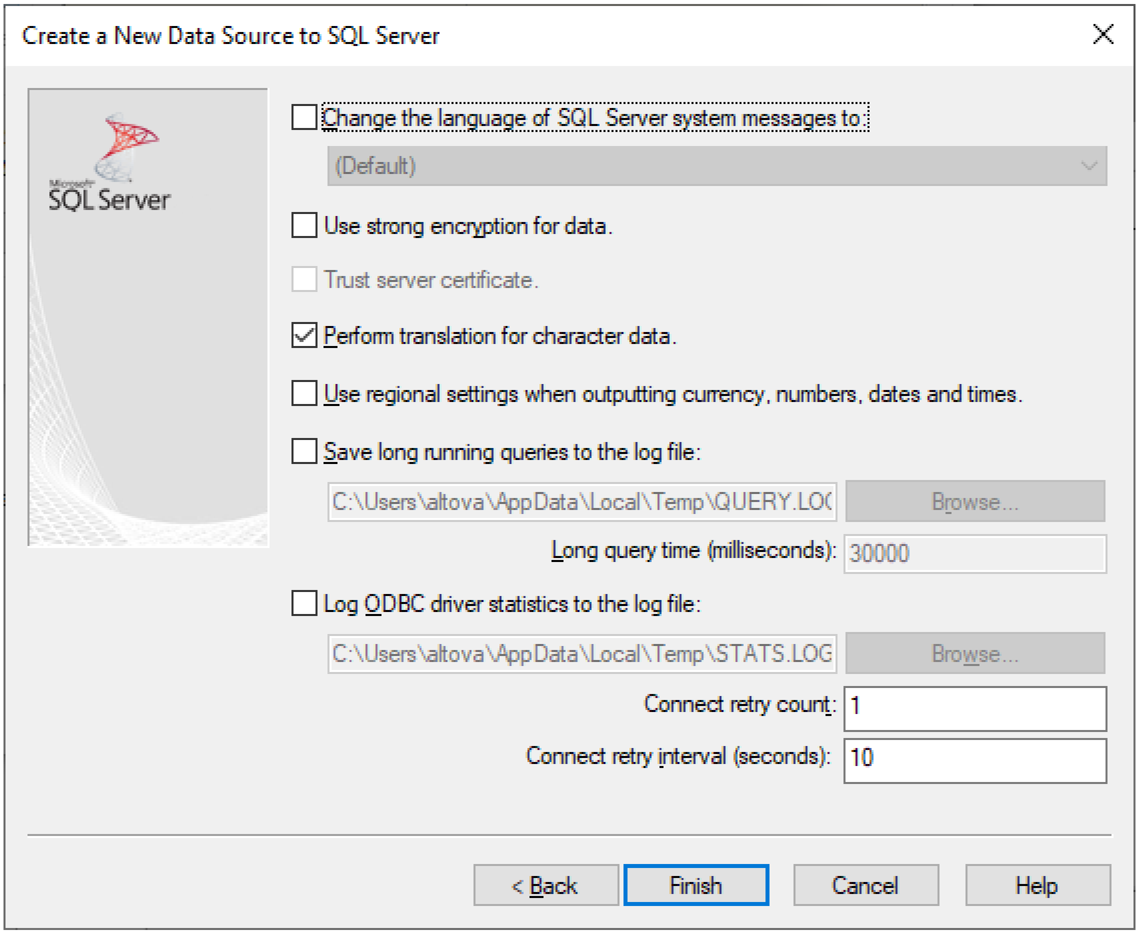
7. If the database server was configured to allow connections from users authenticated on the Windows domain, select **With Integrated Windows authentication**. Otherwise, select one of the other options, as applicable. This example uses **With SQL Server authentication...** , which requires that the user name and password be entered in the relevant boxes.



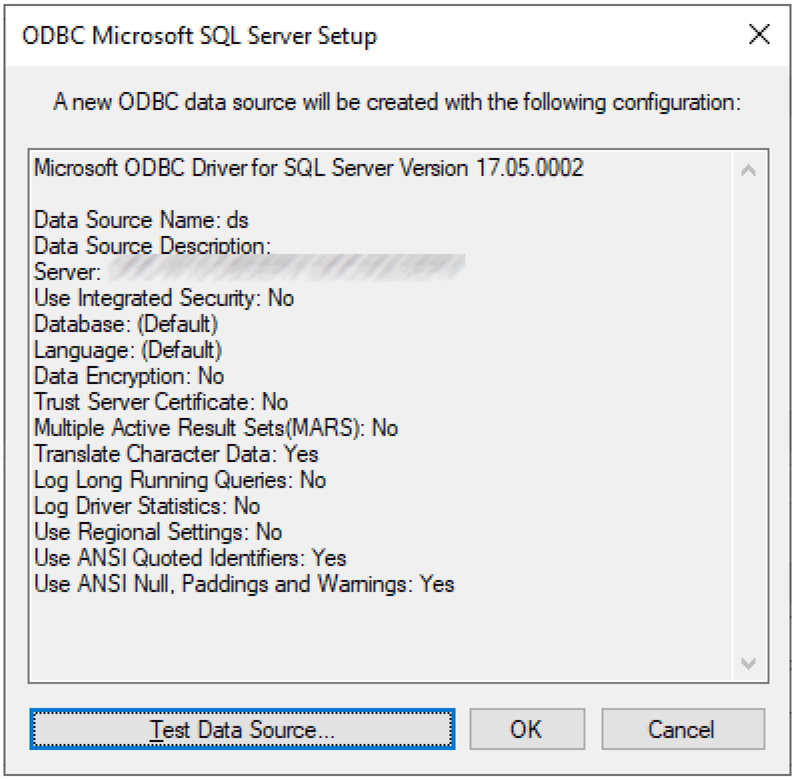
8. Optionally, select the **Change the default database to** check box and enter the name of the database to which you are connecting (in this example, **Sandbox**).



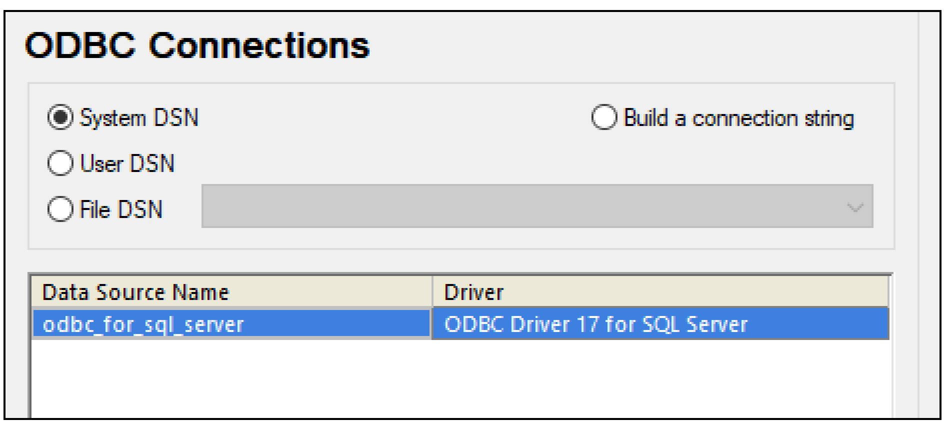
9. Click **Next** and, optionally, configure additional parameters for this connection.



10. Click **Finish**. A confirmation dialog box listing the connection details opens.



11. Click **OK**. The data source now appears in the list of **User** or **System** data sources, as configured, for example:



4.2.1.12.12 MySQL (ODBC)

This example illustrates how to connect to a MySQL database server from a Windows machine through the ODBC driver. The MySQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses MySQL Connector/ODBC 8.0.

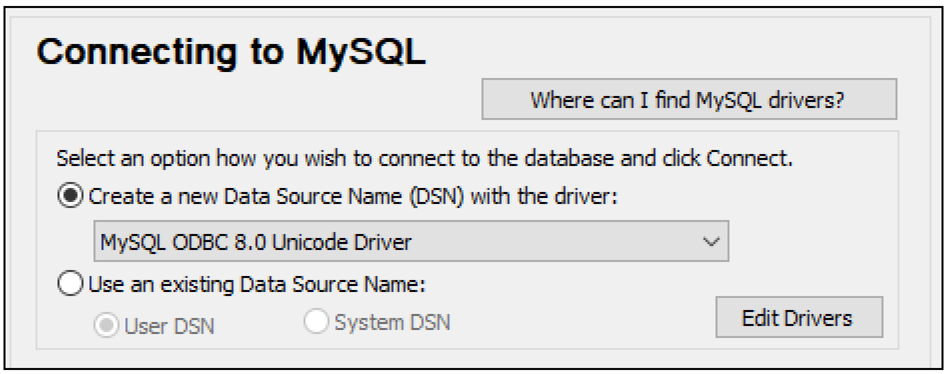
Prerequisites:

- MySQL ODBC driver must be installed on your operating system. Check the MySQL documentation for the driver version recommended for your database server version (see https://dev.mysql.com/downloads/connector/odbc/).
- You have the following database connection details: host, database, port, username, and password.

> If you installed MySQL Connector/ODBC for 64-bit platform, make sure to install MapForce for 64-bit platform as well.

To connect to MySQL via ODBC:

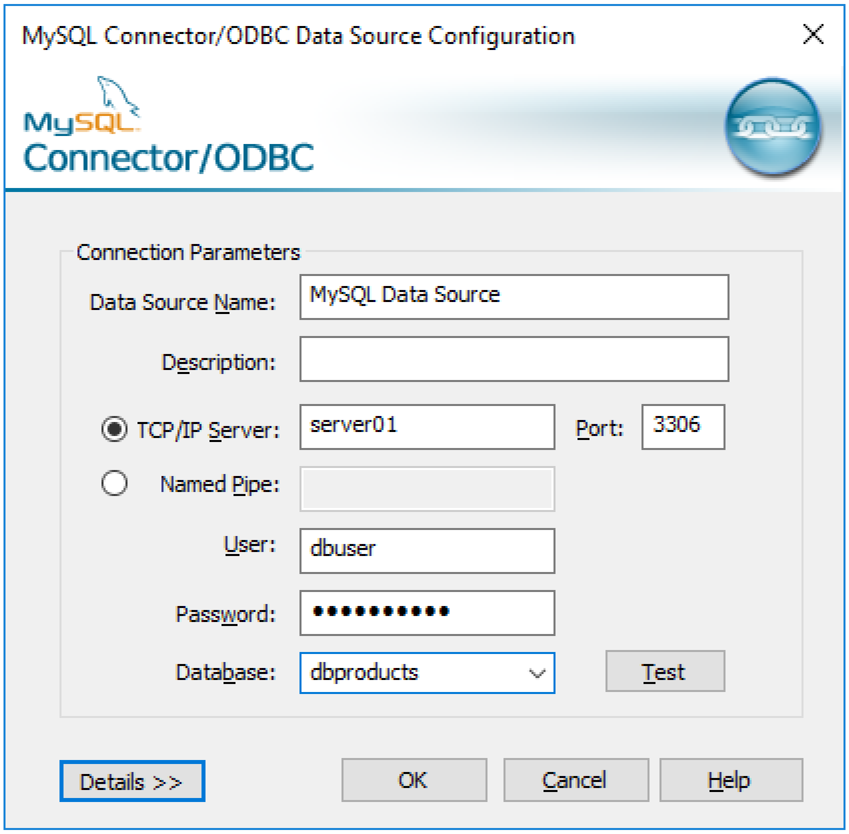
1. Start the database connection wizard 165.
2. Select **MySQL (ODBC)**, and then click **Next**.



3. Select **Create a new Data Source Name (DSN) with the driver**, and select a MySQL driver. If no MySQL driver is available in the list, click **Edit Drivers**, and select any available MySQL drivers (the list contains all ODBC drivers installed on your operating system).

> If you installed MapForce 64-bit, then the 64-bit ODBC drivers are shown in the list. Otherwise, the 32-bit ODBC drivers are shown. See also Viewing the Available ODBC Drivers 184.

4. Click **Connect**.



5. In the Data Source Name box, enter a descriptive name that will help you identify this ODBC data source in future.
6. Fill in the database connection credentials (TCP/IP Server, User, Password), select a database, and then click **OK**.

Note: If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address. Also, if you click **Details>>**, there are several additional parameters available for configuration. Check the driver's documentation before changing their default values.

4.2.1.12.13 Oracle (JDBC)

This example shows you how to connect to an Oracle database server from a client machine, using the JDBC interface. The connection is created as a pure Java connection, using the **Oracle Instant Client Package (Basic)** available from the Oracle website. The advantage of this connection type is that it requires only the Java environment and the .jar libraries supplied by the Oracle Instant Client Package, saving you the effort to install and configure a more complex database client.

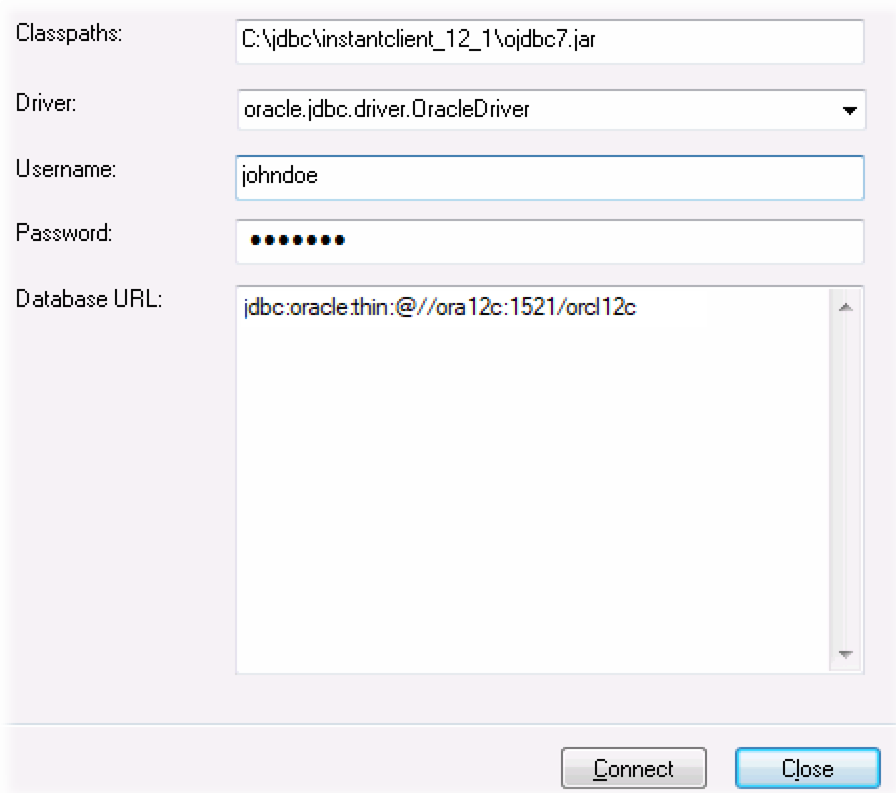
Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java Settings[1476]; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.

- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The **Oracle Instant Client Package (Basic)** must be available on your operating system. The package can be downloaded from the official Oracle website. This example uses Oracle Instant Client Package version 12.1.0.2.0, for Windows 32-bit and, consequently, Oracle JDK 32-bit.
- You have the following database connection details: host, port, service name, username, and password.

To connect to Oracle through the Instant Client Package:

1. Start the database connection wizard[165].
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file is located at the following path: **C:\jdbc\instantclient_12_1\ojdbc7.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH[188]).
4. In the "Driver" box, select either **oracle.jdbc.OracleDriver** or **oracle.jdbc.driver.OracleDriver**. Note that these entries are available if a valid .jar file path is found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).
5. Enter the username and password to the database in the corresponding text boxes.



6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:oracle:thin:@//host:port:service
```

7. Click **Connect**.

4.2.1.12.14 Oracle (ODBC)

This example illustrates a common scenario where you connect from MapForce to an Oracle database server on a network machine, through an Oracle database client installed on the local operating system.

The example includes instructions for setting up an ODBC data source (DSN) using the database connection wizard in MapForce. If you have already created a DSN, or if you prefer to create it directly from the **ODBC Data Source administrator** in Windows, you can do so, and then select it when prompted by the wizard. For more information about ODBC data sources, see Setting up an ODBC Connection 182.

Prerequisites:

- The Oracle database client (which includes the ODBC Oracle driver) must be installed and configured on your operating system. For instructions on how to install and configure an Oracle database client, refer to the documentation supplied with your Oracle software.
- The **tnsnames.ora** file located in Oracle home directory contains an entry that describes the database connection parameters, in a format similar to this:

```
ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = server01)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = orcl)
      (SERVER = DEDICATED)
    )
  )
```

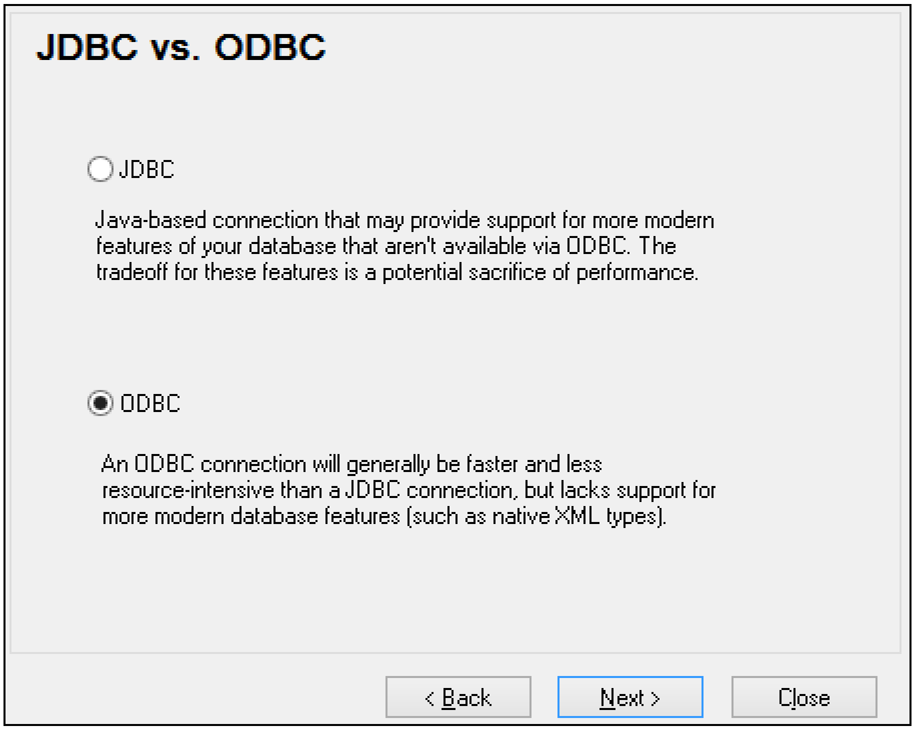
The path to the **tnsnames.ora** file depends on the location where Oracle home directory was installed. For Oracle database client 11.2.0, the default Oracle home directory path could be as follows:

```
C:\app\username\product\11.2.0\client_1\network\admin\tnsnames.ora
```

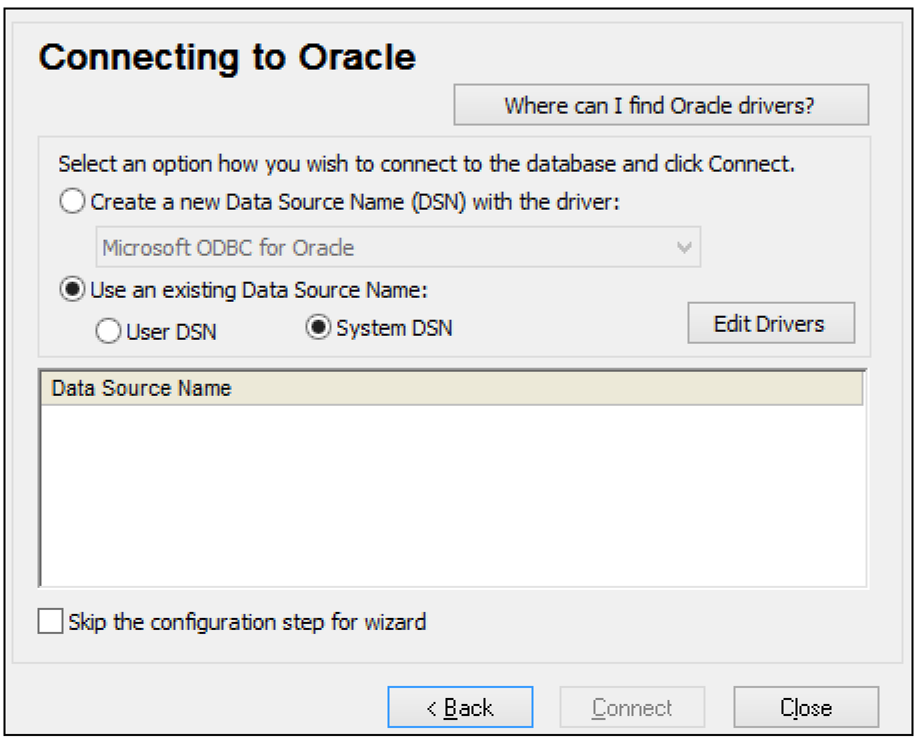
You can add new entries to the **tnsnames.ora** file either by pasting the connection details and saving the file, or by running the Oracle *Net Configuration Assistant* wizard (if available). If you want these values to appear in dropdown lists during the configuration process, then you may need to add the path to the admin folder as a **TNS_ADMIN** environment variable.

To connect to Oracle using ODBC:

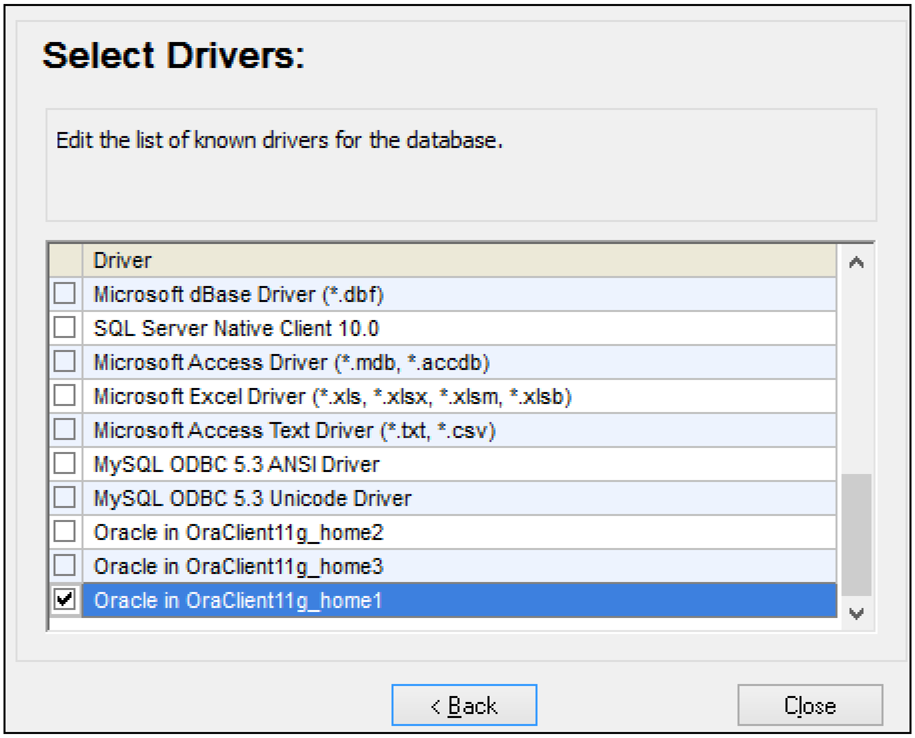
1. Start the database connection wizard 165.
2. Select **Oracle (ODBC / JDBC)**, and then click **Next**.



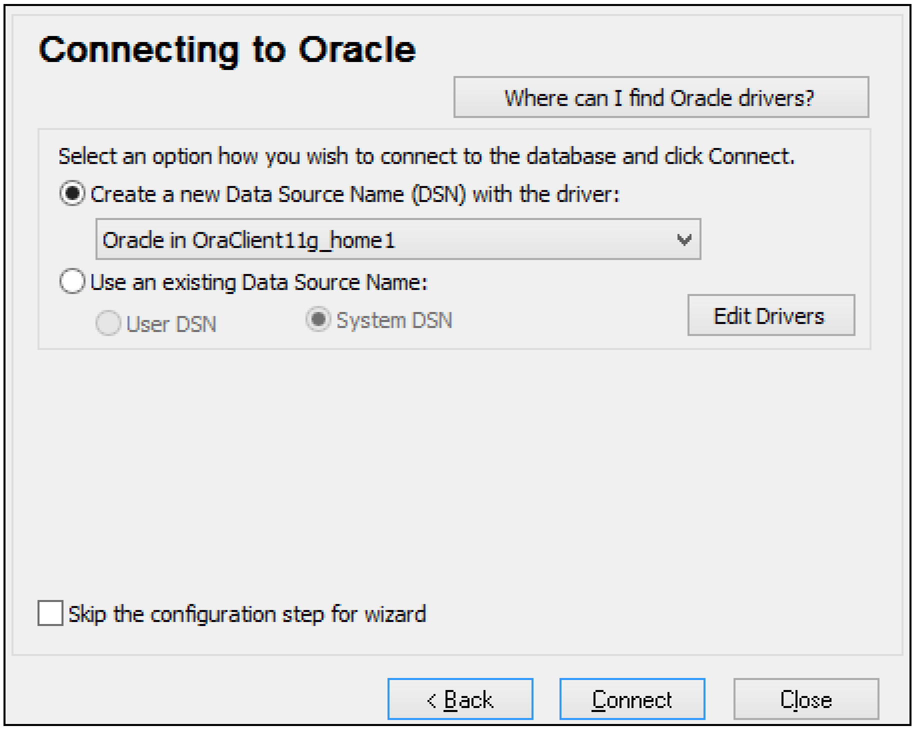
3. Select **ODBC**.



4. Click **Edit Drivers**.

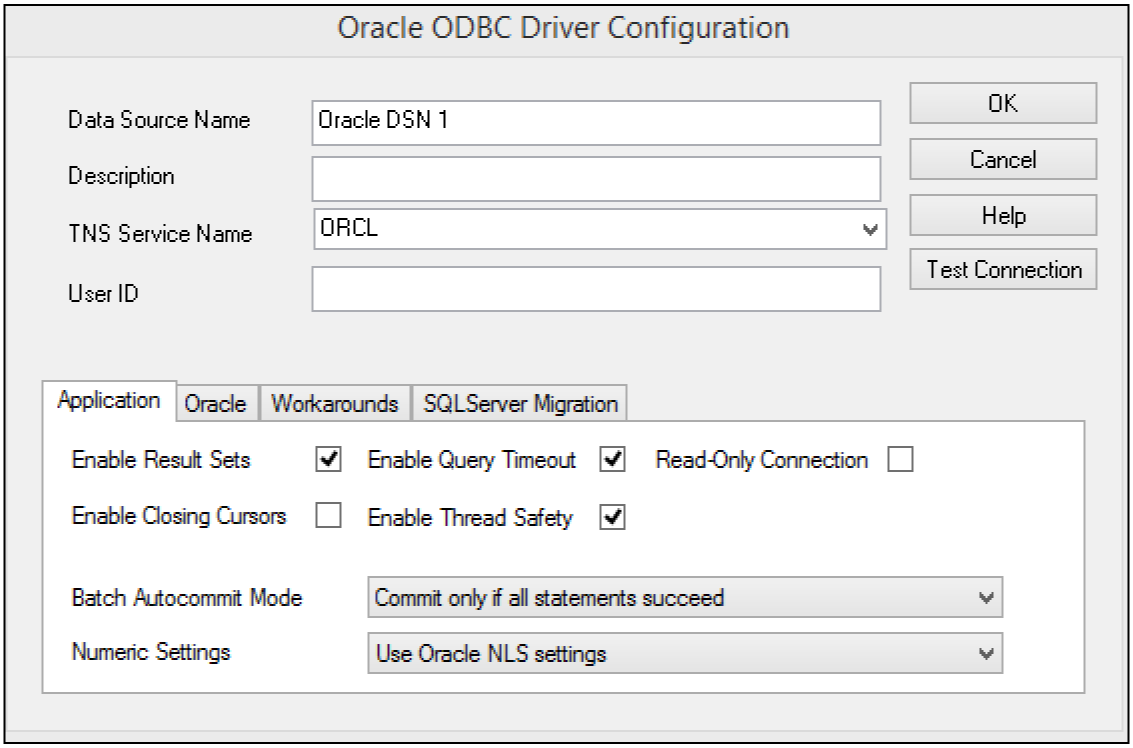


5. Select the Oracle drivers you wish to use (in this example, **Oracle in OraClient11g_home1**). The list displays the Oracle drivers available on your system after installation of Oracle client.
6. Click **Back**.
7. Select **Create a new data source name (DSN) with the driver**, and then select the Oracle driver chosen in step 4.

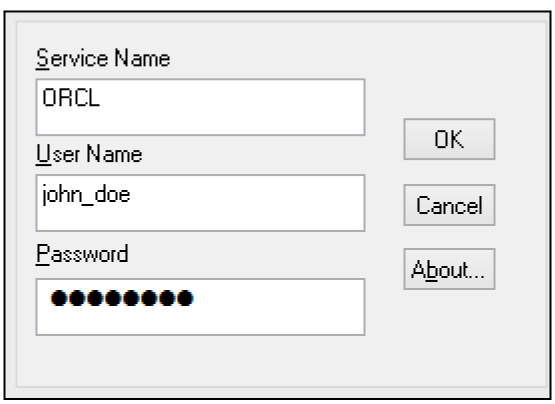


> Avoid using the Microsoft-supplied driver called **Microsoft ODBC for Oracle** driver. Microsoft recommends using the ODBC driver provided by Oracle (see http://msdn.microsoft.com/en-us/library/ms714756%28v=vs.85%29.aspx)

8. Click **Connect**.



9. In the Data Source Name text box, enter a name to identify the data source (in this example, **Oracle DSN 1**).
10. In the TNS Service Name box, enter the connection name as it is defined in the **tnsnames.ora** file (see prerequisites 229). In this example, the connection name is **ORCL**. *Note:* If you wish to have the dropdown list of the combo box populated with the values of the **tnsnames.ora** file, then you may need to add the path to the admin folder as a **TNS_ADMIN** environment variable.
11. Click **OK**.



12. Enter the username and password to the database, and then click OK.

4.2.1.12.15 PostgreSQL (ODBC)

This example illustrates how to connect to a PostgreSQL database server from a Windows machine through the ODBC driver. The PostgreSQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses the psqlODBC driver (version 11.0) downloaded from the official website (see also Database Drivers Overview[167]).

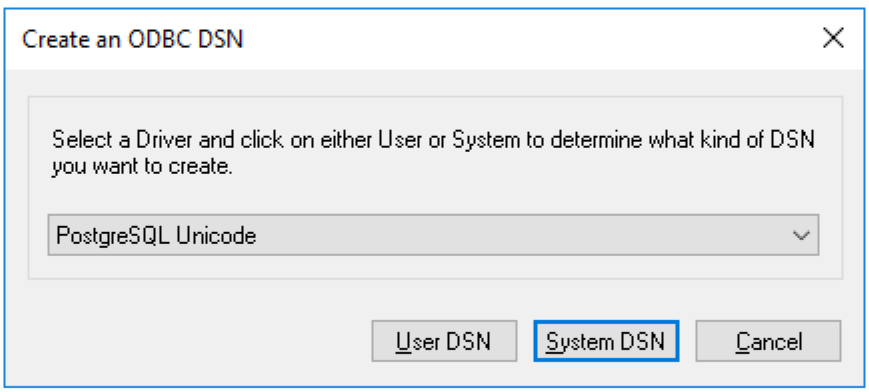
Note: You can also connect to a PostgreSQL database server directly (without the ODBC driver), see Setting up a PostgreSQL Connection[190].

Prerequisites:

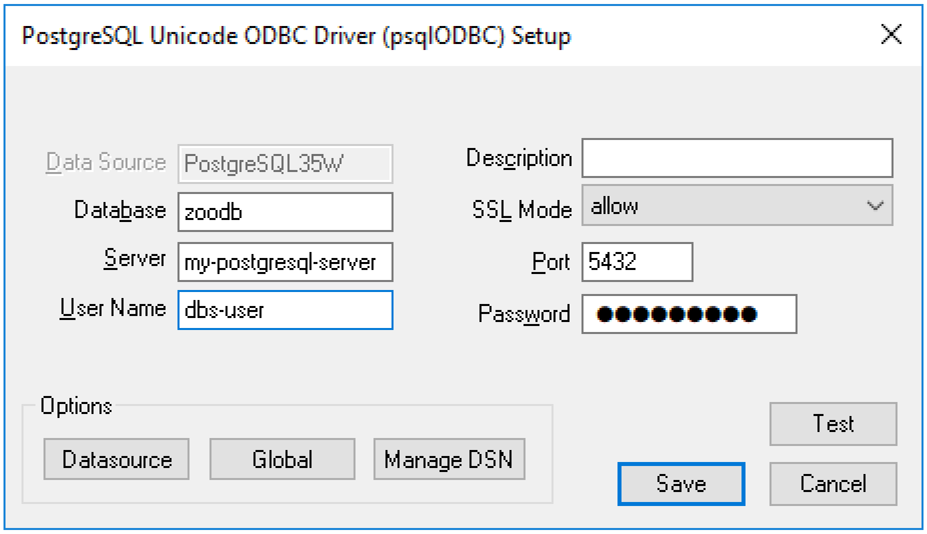
- *psqlODBC* driver must be installed on your operating system.
- You have the following database connection details: server, port, database, user name, and password.

To set up a connection to PostgreSQL using ODBC:

1. Start the database connection wizard[165].
2. Click **ODBC Connections**.
3. Select the **User DSN** option.
4. Click **Create a new DSN** and select the driver from the drop-down list. If no PostgreSQL driver is available in the list, make sure that the PostgreSQL ODBC driver is installed on your operating system, as mentioned in the prerequisites above.

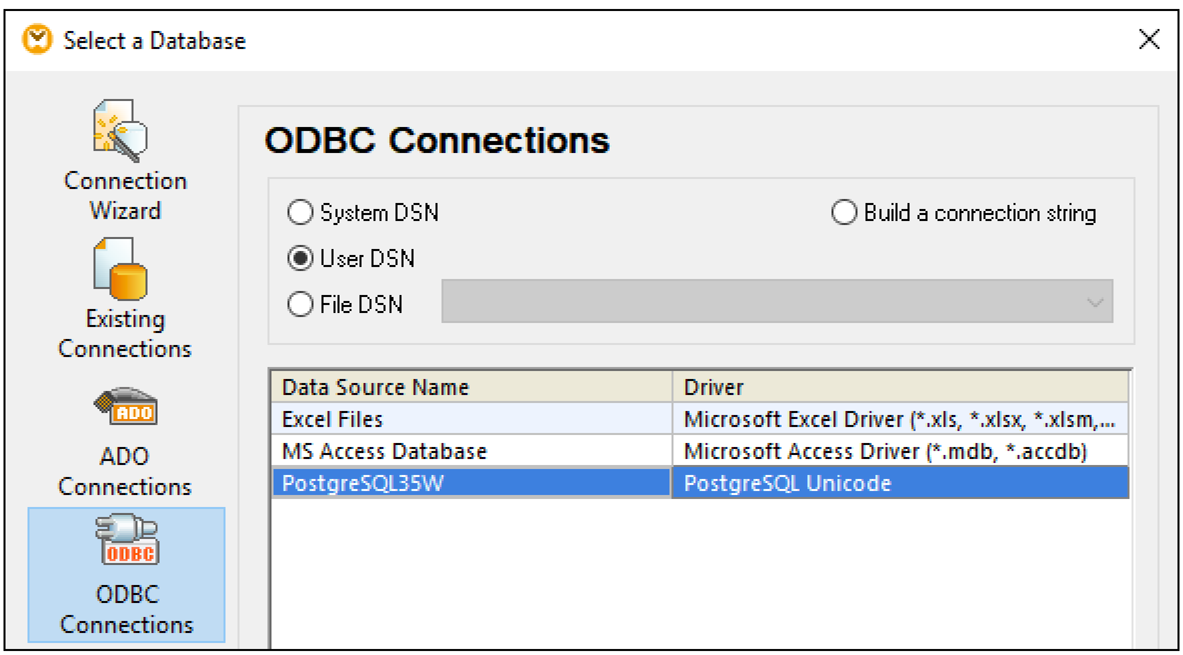


5. Click **User DSN**.



6. Fill in the database connection credentials (these must be supplied by the database owner), and then click **Save**.

The connection is now available in the list of ODBC connections. To connect to the database, you can either double-click the connection or select it, and then click **Connect**.



4.2.1.12.16 Progress OpenEdge (JDBC)

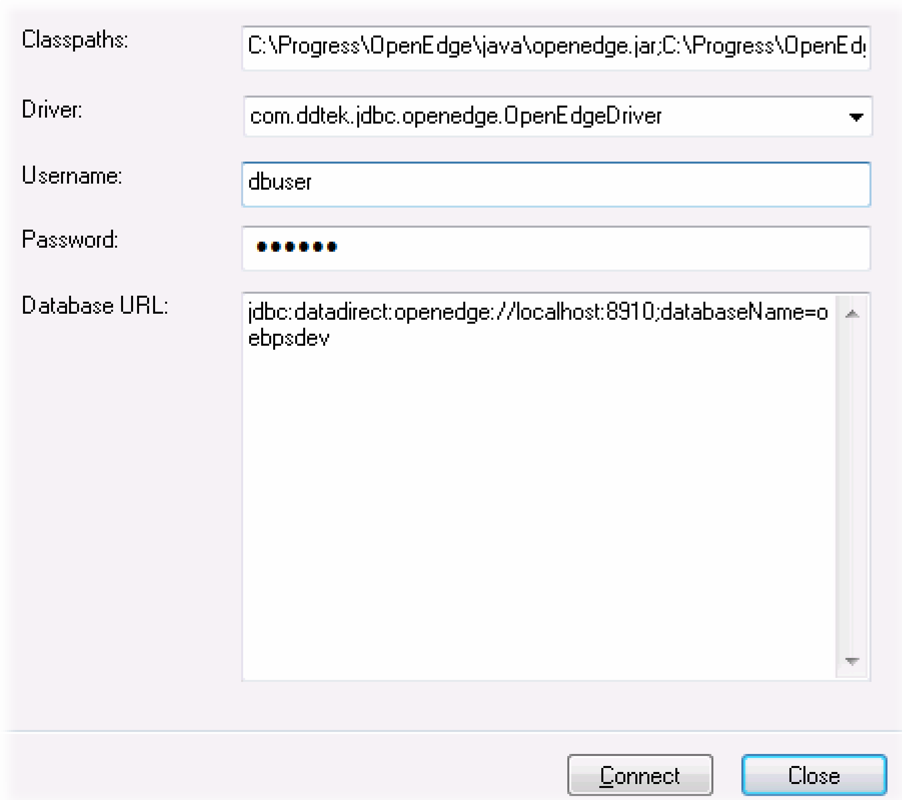
This example illustrates how to connect to a Progress OpenEdge 11.6 database server through JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java Settings 1476; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The operating system's `PATH` environment variable must include the path to the `bin` directory of the JRE or JDK installation directory, for example `C:\Program Files (x86)\Java\jre1.8.0_51\bin`.
- The Progress OpenEdge JDBC driver must be available on your operating system. In this example, JDBC connectivity is provided by the **openedge.jar** and **pool.jar** driver component files available in **C:\Progress\OpenEdge\java** as part of the OpenEdge SDK installation.
- You have the following database connection details: host, port, database name, username, and password.

Connecting to OpenEdge through JDBC

1. Start the database connection wizard 165.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file paths are: **`C:\Progress\OpenEdge\java\openedge.jar;C:\Progress\OpenEdge\java\pool.jar;`**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH 188).
4. In the "Driver" box, select **com.ddtek.jdbc.openedge.OpenEdgeDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:datadirect:openedge://host:port;databaseName=db_name
```

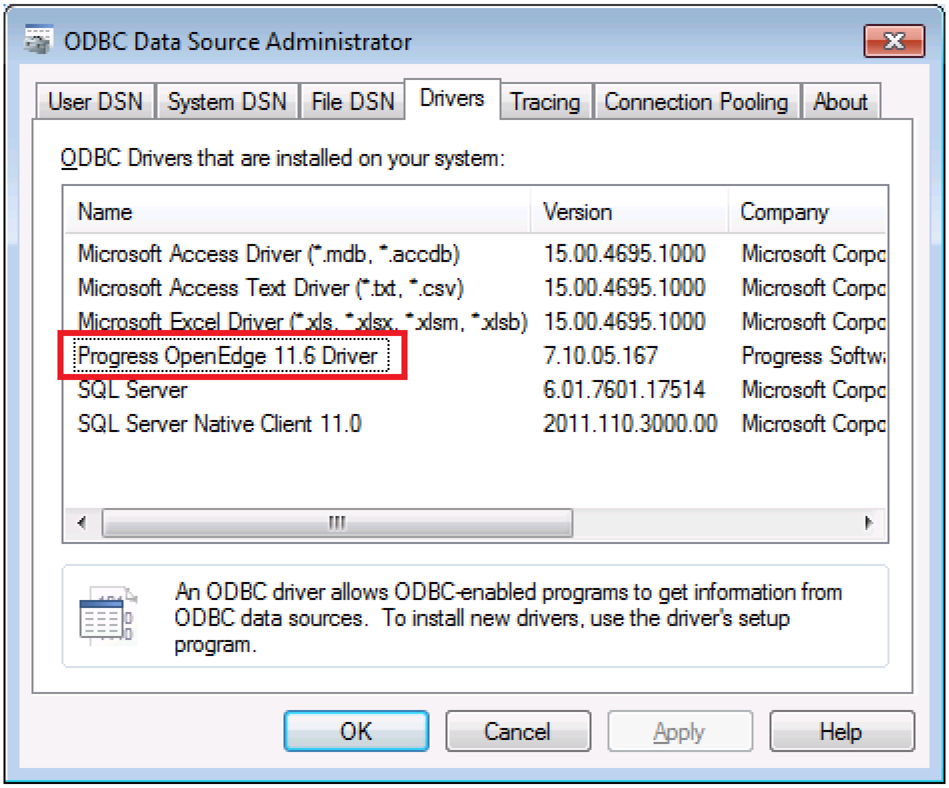
7. Click **Connect**.

4.2.1.12.17 Progress OpenEdge (ODBC)

This example illustrates how to connect to a Progress OpenEdge database server through the Progress OpenEdge 11.6 ODBC driver.

Prerequisites:

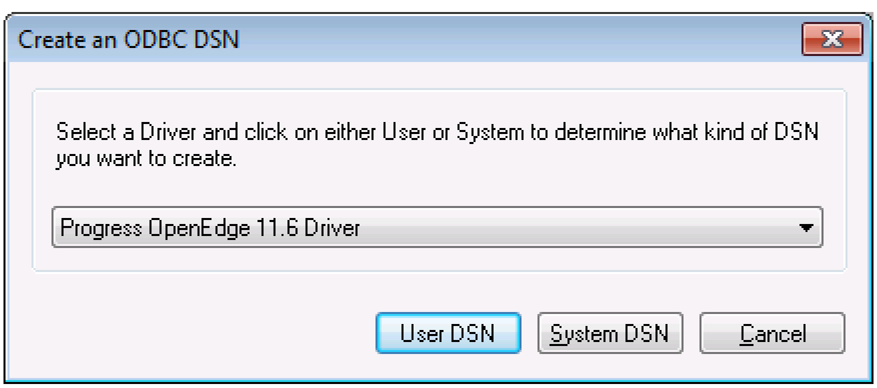
- The *ODBC Connector for Progress OpenEdge* driver must be installed on your operating system. The Progress OpenEdge ODBC driver can be downloaded from the vendor's website (see also Database Drivers Overview[167]). Make sure to download the 32-bit driver when running the 32-bit version of MapForce, and the 64-bit driver when running the 64-bit version. After installation, check if the ODBC driver is available on your machine (see also Viewing the Available ODBC Drivers[184]).



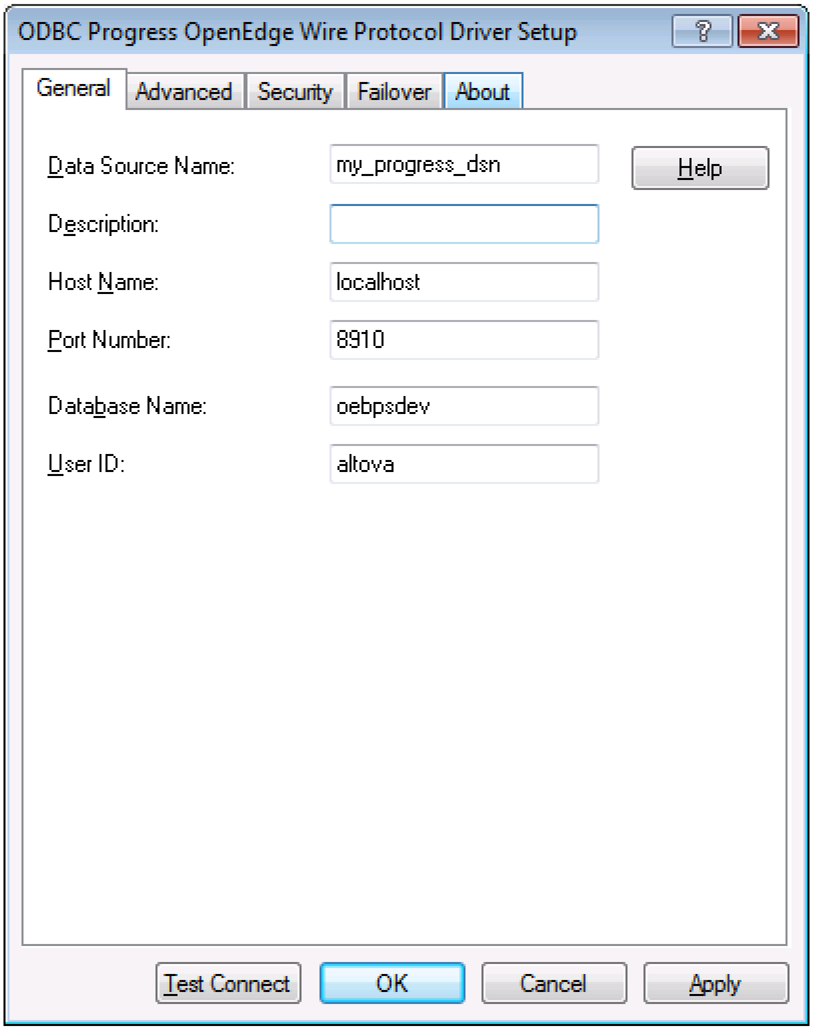
- You have the following database connection details: host name, port number, database name, user ID, and password.

Connecting to Progress OpenEdge through ODBC

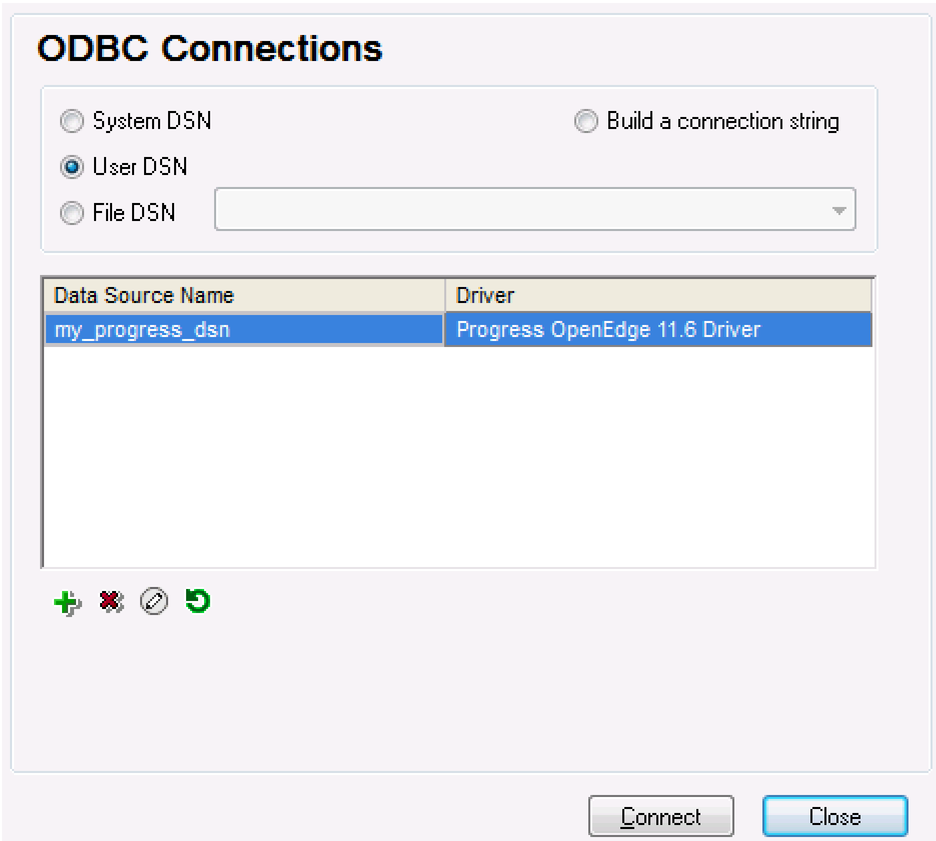
1. Start the database connection wizard 165.
2. Click **ODBC Connections**.
3. Click **User DSN** (alternatively, click **System DSN**, or **File DSN**, in which case the subsequent instructions will be similar).
4. Click **Add** .
5. Select the **Progress OpenEdge Driver** from the list, and click **User DSN** (or **System DSN**, if applicable).



6. Fill in the database connection credentials (Database, Server, Port, User Name, Password), and then click **OK**. To verify connectivity before saving the entered data, click **Test Connect**.



7. Click OK. The new data source now appears in the list of ODBC data sources.



8. Click **Connect**.

4.2.1.12.18 Sybase (JDBC)

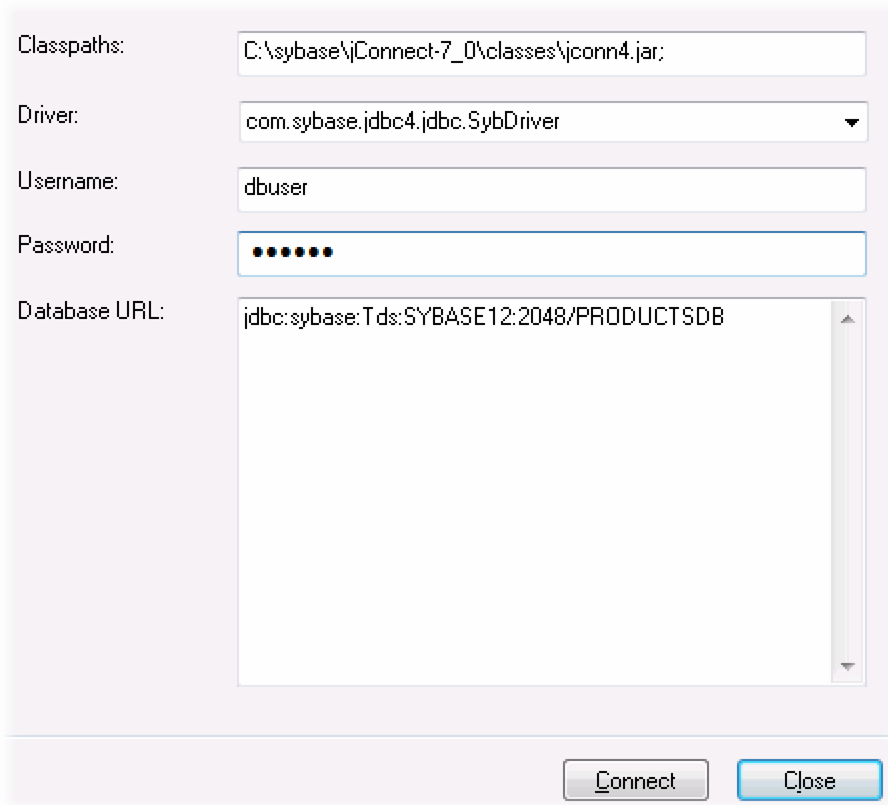
This example illustrates how to connect to a Sybase database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java Settings 1476; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- Sybase *jConnect* component must be installed on your operating system (in this example, *jConnect 7.0* is used, installed as part of the *Sybase Adaptive Server Enterprise PC Client* installation). For the installation instructions of the database client, refer to Sybase documentation.
- You have the following database connection details: host, port, database name, username, and password.

To connect to Sybase through JDBC:

1. [Start the database connection wizard](165).
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the required .jar file path is: **C:\sybase\jConnect-7_0\classes\jconn4.jar**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also [Configuring the CLASSPATH](188)).
4. In the "Driver" box, select **com.sybase.jdbc4.jdbc.SybDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpaths" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:sybase:Tds:hostName:port/databaseName
```

7. Click **Connect**.

4.2.1.12.19 Teradata (JDBC)

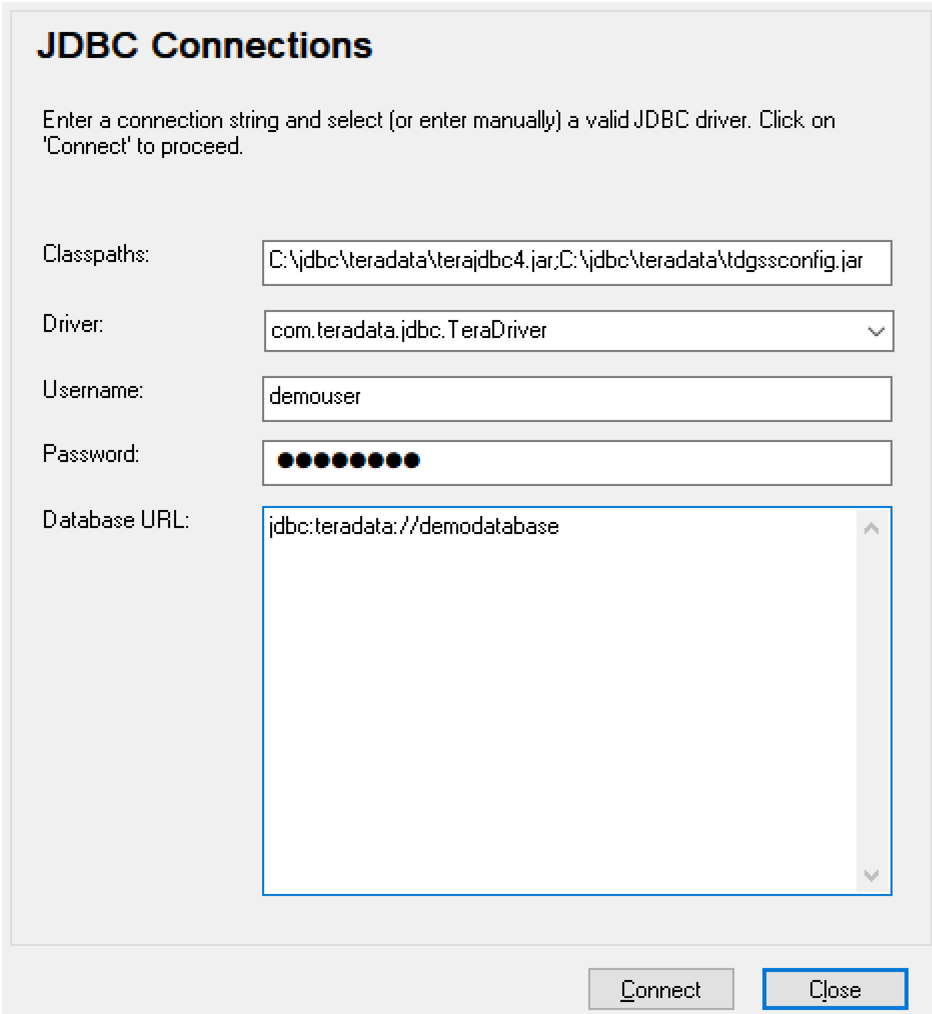
This example illustrates how to connect to a Teradata database server through JDBC.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: a) The custom JVM path you may have set in application **Options**, see Java Settings 1476; b) The JVM path found in the Windows registry; c) The `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or more .jar files that provide connectivity to the database) must be available on your operating system. In this example, Teradata JDBC Driver 16.20.00.02 is used. For more information, see https://downloads.teradata.com/download/connectivity/jdbc-driver.
- You have the following database connection details: host, database, port, username, and password.

To connect to Teradata through JDBC:

1. Start the database connection wizard 165.
2. Click **JDBC Connections**.
3. Next to "Classpaths", enter the path to the .jar file which provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. In this example, the .jar files are located at the following path: **C:\jdbc\teradata**. Note that you can leave the "Classpaths" text box empty if you have added the .jar file path(s) to the CLASSPATH environment variable of the operating system (see also Configuring the CLASSPATH 188).
4. In the "Driver" box, select **com.teradata.jdbc.TeraDriver**. Note that this entry is available if a valid .jar file path is found either in the "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



5. Enter the username and password to the database in the corresponding text boxes.
6. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted value with the one applicable to your database server.

```
jdbc:teradata://databaseServerName
```

7. Click **Connect**.

4.2.1.12.20 Teradata (ODBC)

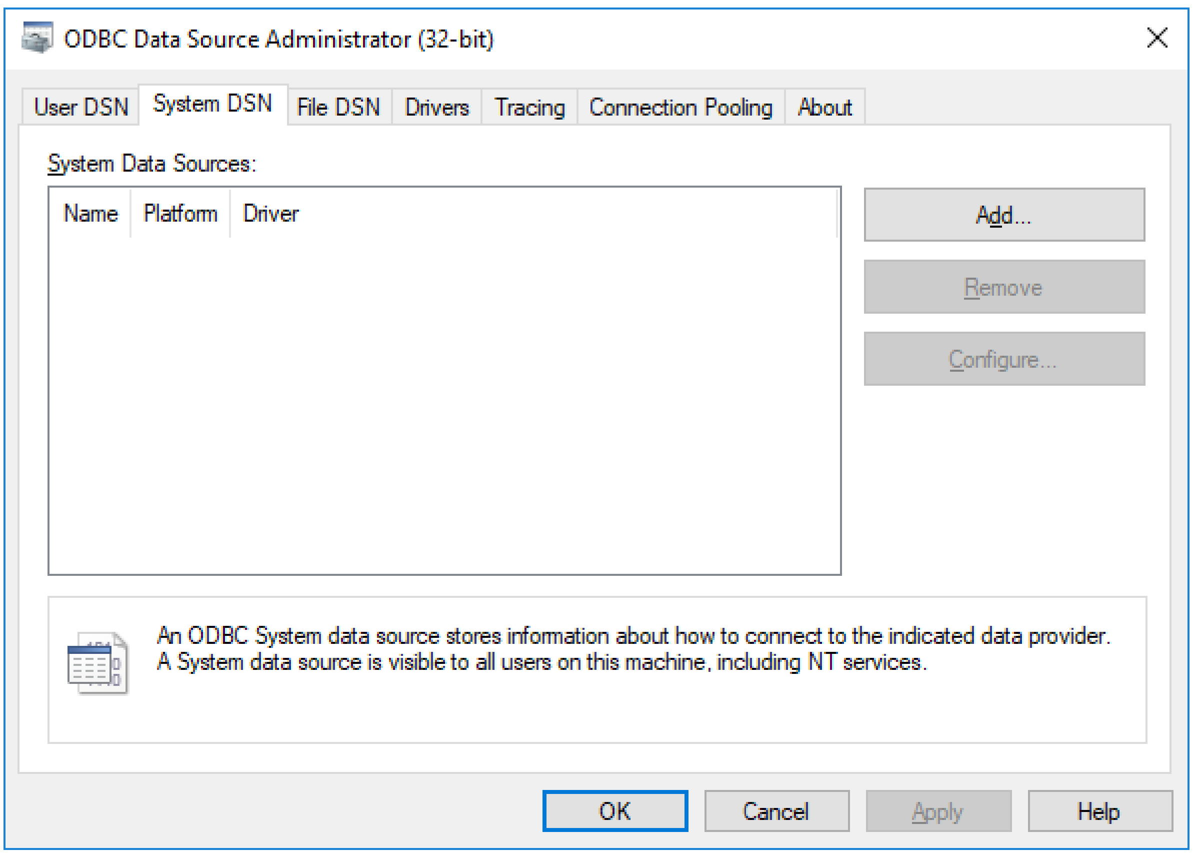
This example illustrates how to connect to a Teradata database server through ODBC.

Prerequisites:

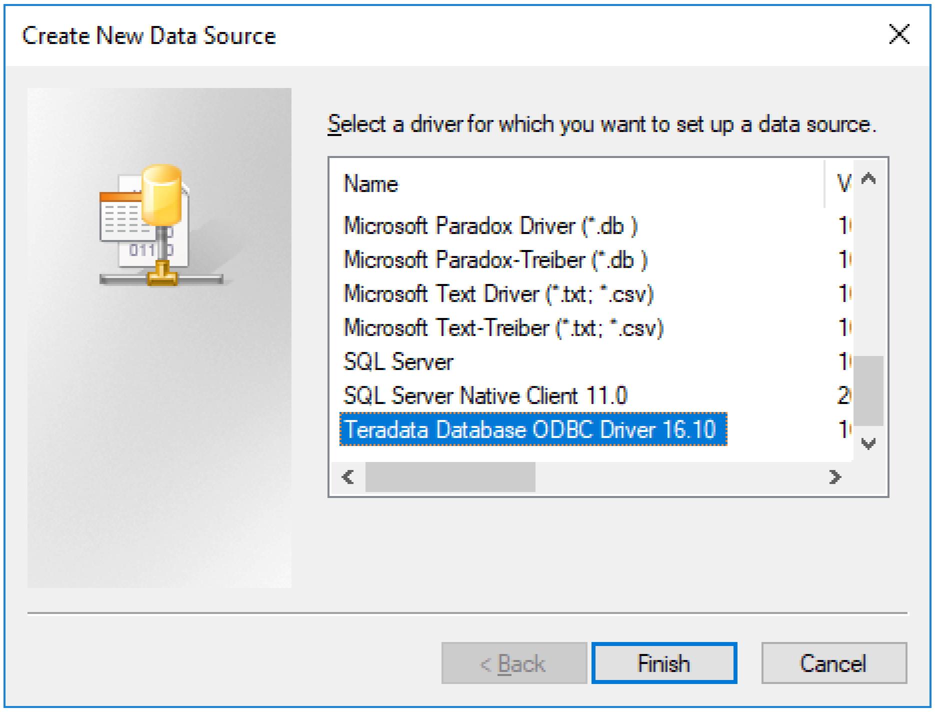
- The Teradata ODBC driver must be installed (see https://downloads.teradata.com/download/connectivity/odbc-driver/windows. This example uses Teradata ODBC Driver for Windows version 16.20.00.
- You have the following database connection details: host, username, and password.

To connect to Teradata through ODBC:

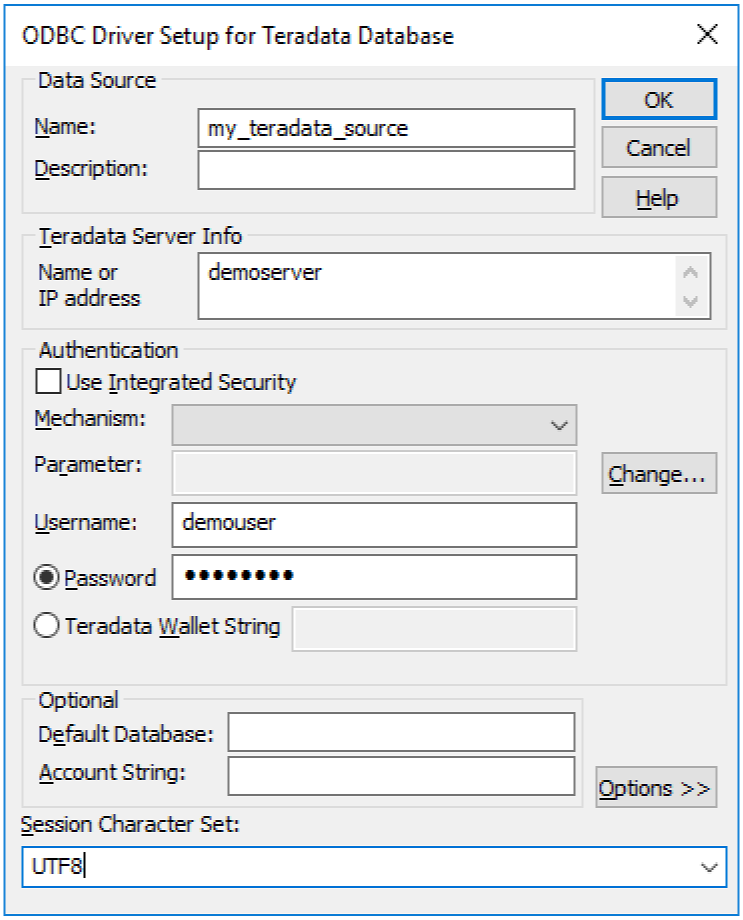
1. Press the **Windows** key, start typing "ODBC", and select **Set up ODBC data sources (32-bit)** from the list of suggestions. If you have a 64-bit ODBC driver, select **Set up ODBC data sources (64-bit)** and use 64-bit MapForce in the subsequent steps.



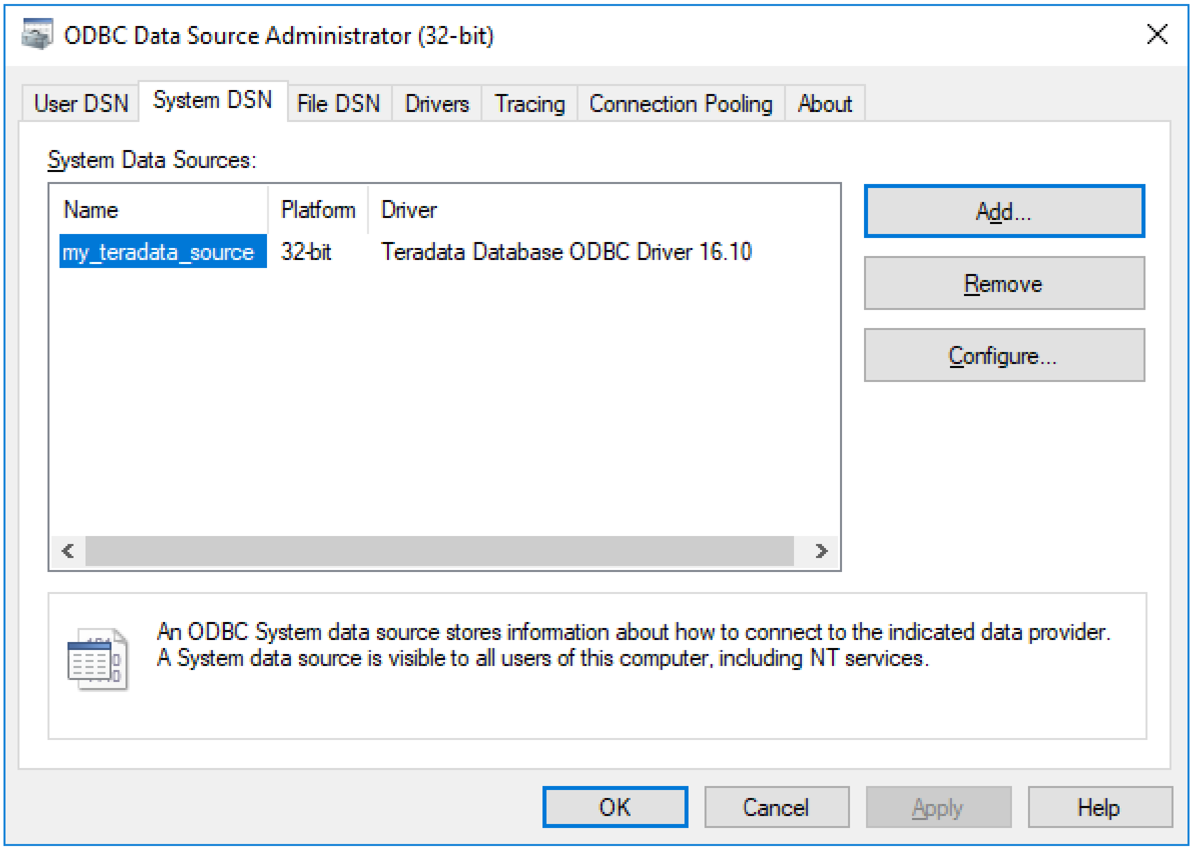
2. Click the **System DSN** tab, and then click **Add**.



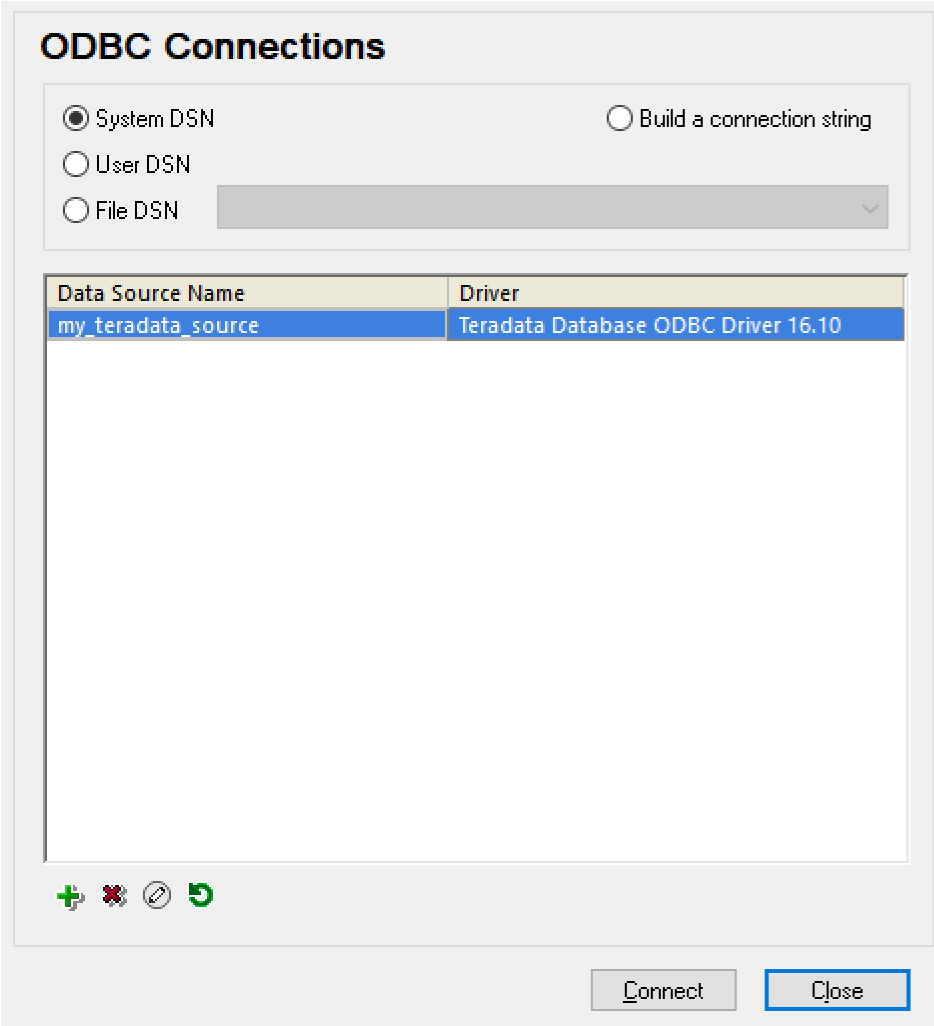
3. Select **Teradata Database ODBC Driver** and click **Finish**.



4. Enter name and, optionally, a description that will help you identify this ODBC data source in future. Also, enter the database connection credentials (Database server, User, Password), and, optionally, select a database.
5. Click **OK**. The data source now appears in the list.



6. Run MapForce and start the database connection wizard 165.
7. Click **ODBC Connections**.



8. Click **System DSN**, select the data source created previously, and then click **Connect**.

Note: If you get the following error: "The driver returned invalid (or failed to return) SQL_DRIVER_ODBC_VER: 03.80", make sure that the path to the ODBC client (for example, **C:\Program Files\Teradata\Client\16.10\bin**, if you installed it to this location) exists in your system's PATH environment variable. If this path is missing, add it manually.

4.2.2 Introduction to Database Mappings

This section is an introduction to working with databases in MapForce. It shows you how to work with a database after the connection to it is successfully established (see Connecting to a Database 163). This includes selecting the database objects that you want to appear on the mapping, handling database relationships, and configuring the database settings applicable to the mapping process. Examples of how to achieve specific goals when mapping data to or from database components are also included.

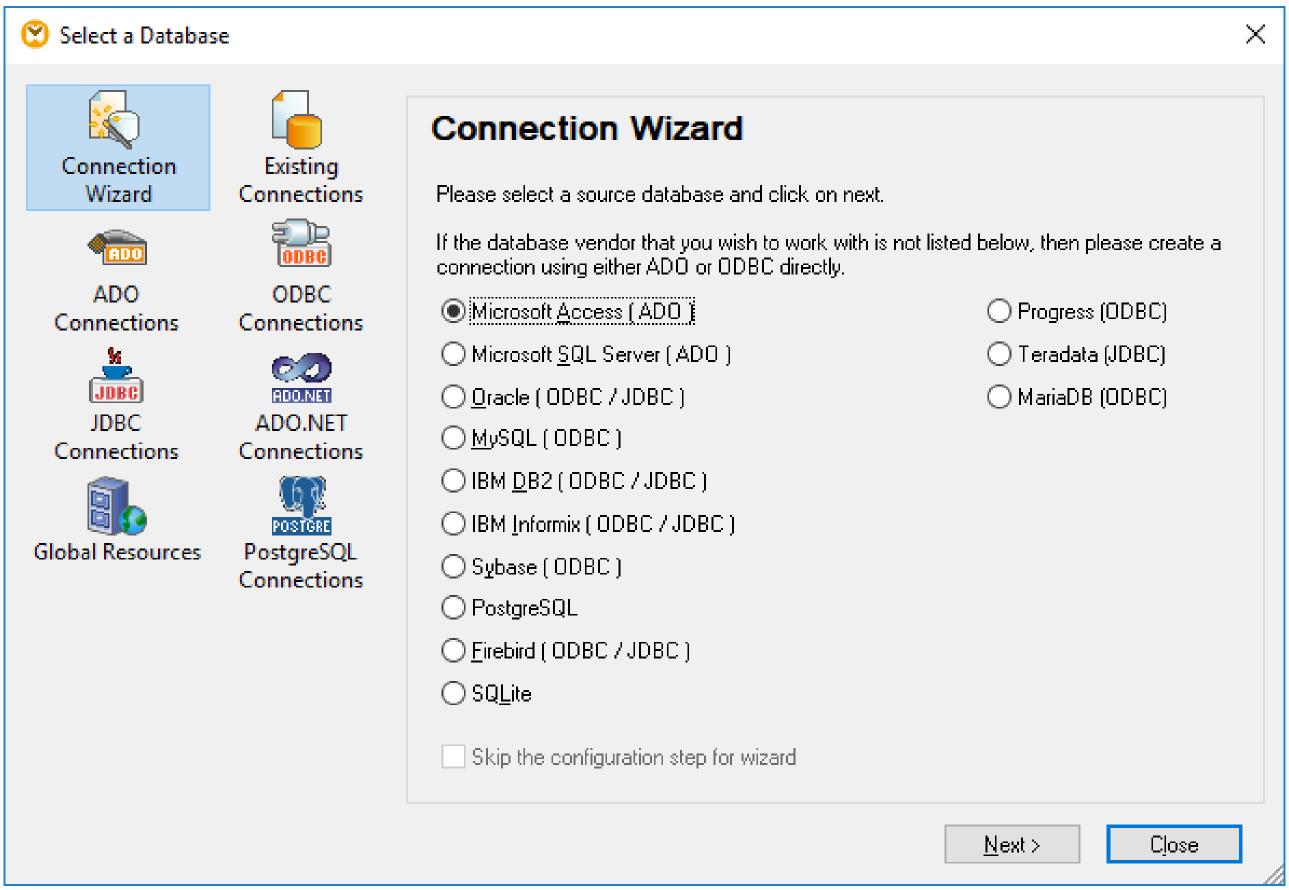
4.2.2.1 Adding Databases to the Mapping

Before adding a database to the mapping, make sure to select a transformation language where database mappings are supported. This can be either the BUILT-IN transformation language, or any of the following languages: C++, C#, Java (see also Transformation Languages [27]). Note that, if you intend to deploy the mapping to FlowForce Server or execute it with MapForce Server, or use features such as Bulk Transfer and stored procedures, BUILT-IN must be selected as transformation language.

Once the desired transformation language is selected, you can add a database to the mapping in one of the following ways:

- On the **Insert** menu, click **Database**.
- Click the **Insert Database** ( ) toolbar button.

When you take any of these actions, a database connection wizard appears, guiding you through the steps required to connect to the database.



Database Connection Wizard

Note: In some advanced scenarios, databases can also be added to the mapping as variables (see Using Variables [847]). When you choose to add a database structure as a variable, the same database connection wizard mentioned above appears.

For instructions about setting up a connection to any of the databases supported by MapForce, see Connecting to a Database [163].

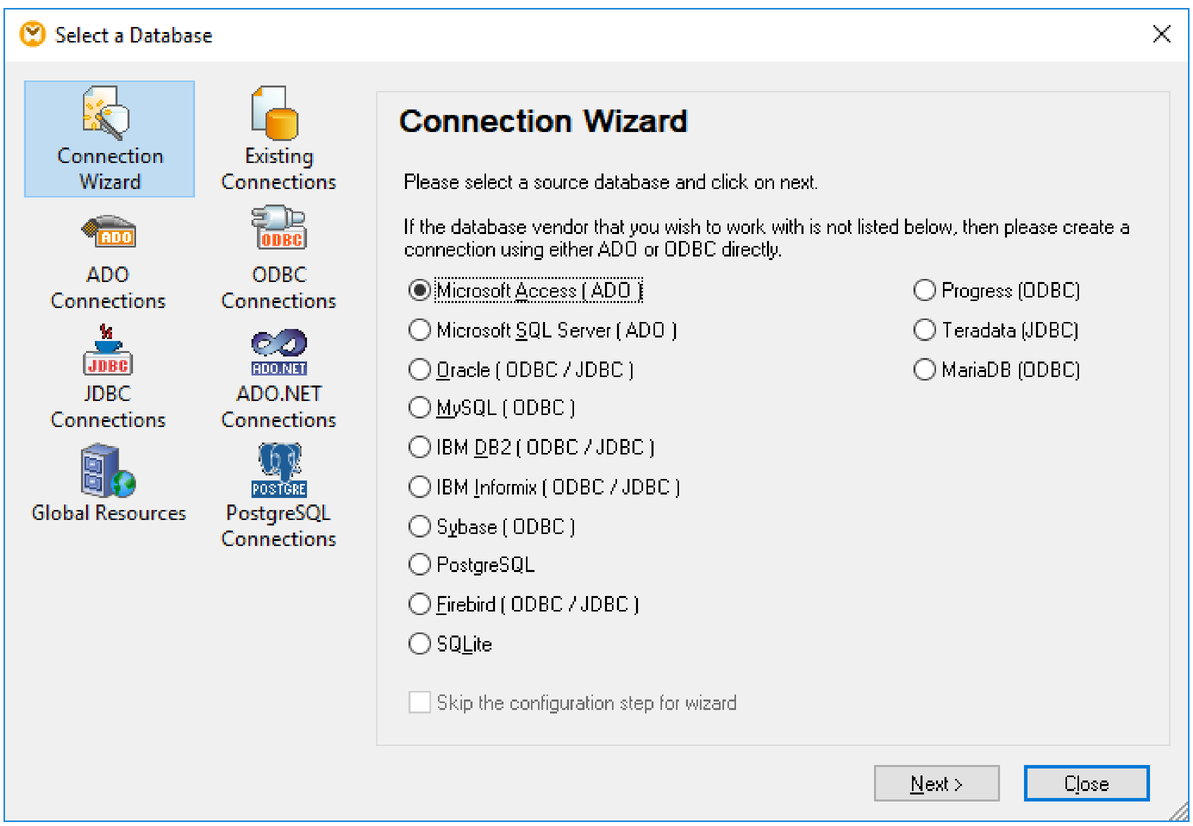
Once the database connection is successfully established, you are prompted to select the database objects that should appear on the mapping. See Adding, Editing, and Removing Database Objects [252] for further information.

4.2.2.2 Example: Adding the "altova.mdb" Database to the Mapping

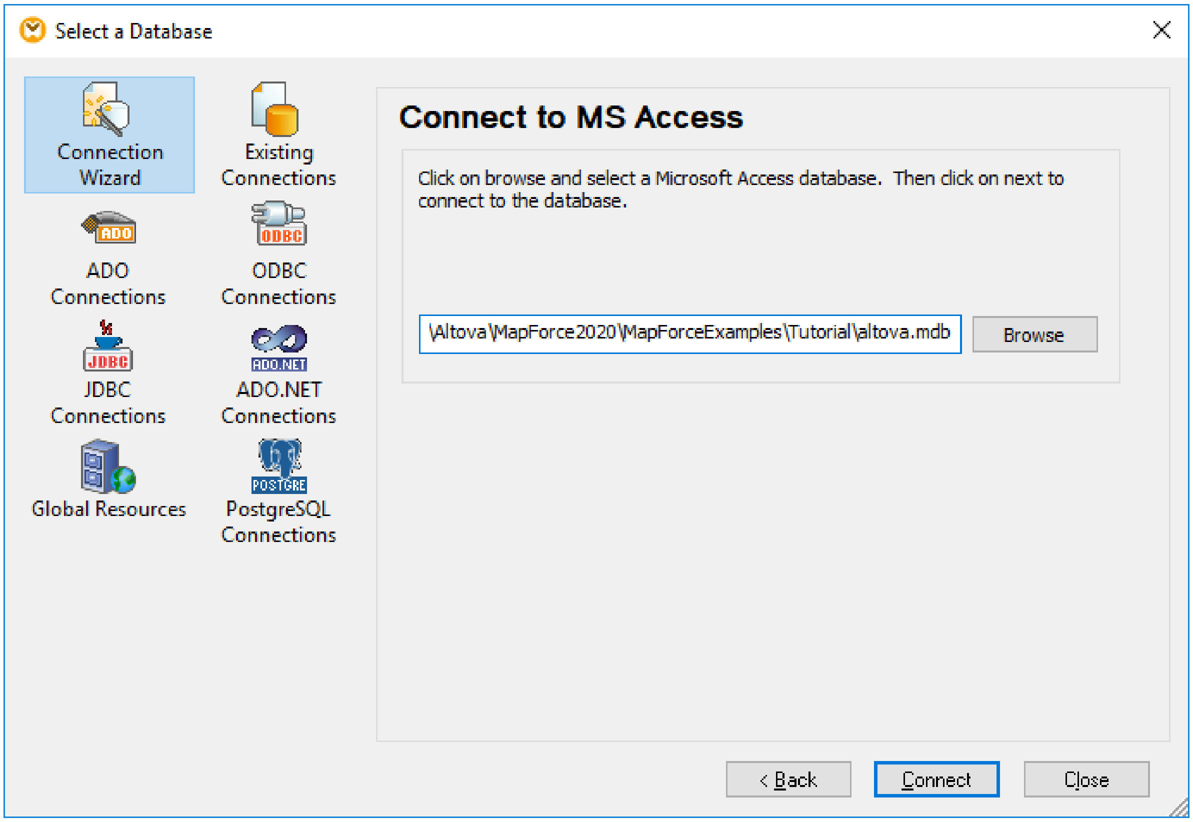
This example shows you how to add a sample Microsoft Access database to a mapping. The sample database is called **altova.mdb** and can be found in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. The **altova.mdb** database supports various database-related actions and concepts described in this documentation.

To add the altova.mdb database to the mapping:

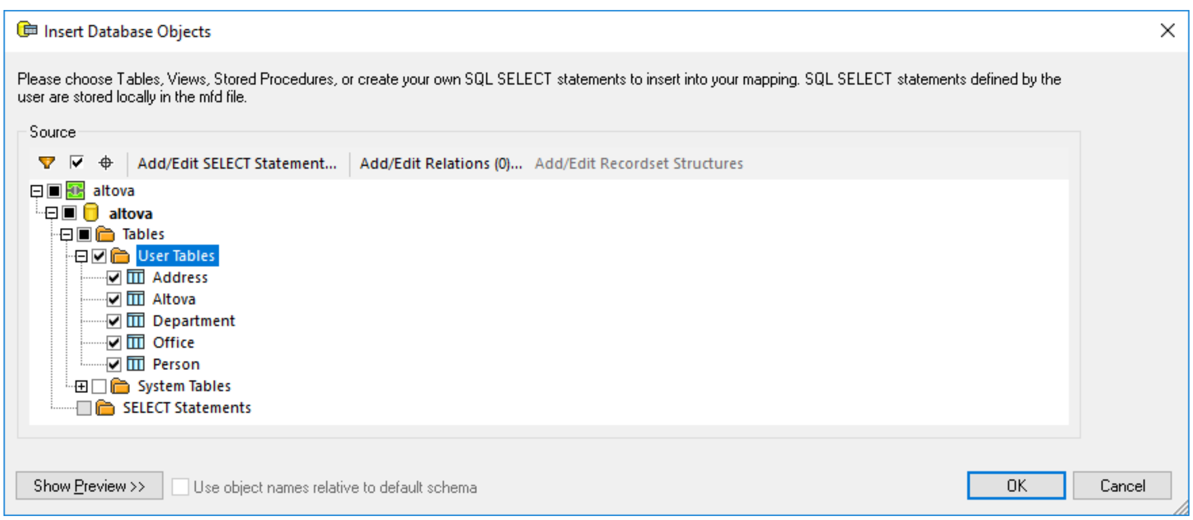
1. On the **Insert** menu, click **Database**. Alternatively, click the **Insert Database** ( ) toolbar button.



2. Click **Connection Wizard**, select **Microsoft Access (ADO)**, and click **Next**.



3. Browse for the **altova.mdb** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder, and then click **Connect**.
4. When prompted to select the database objects, select **User Tables**.

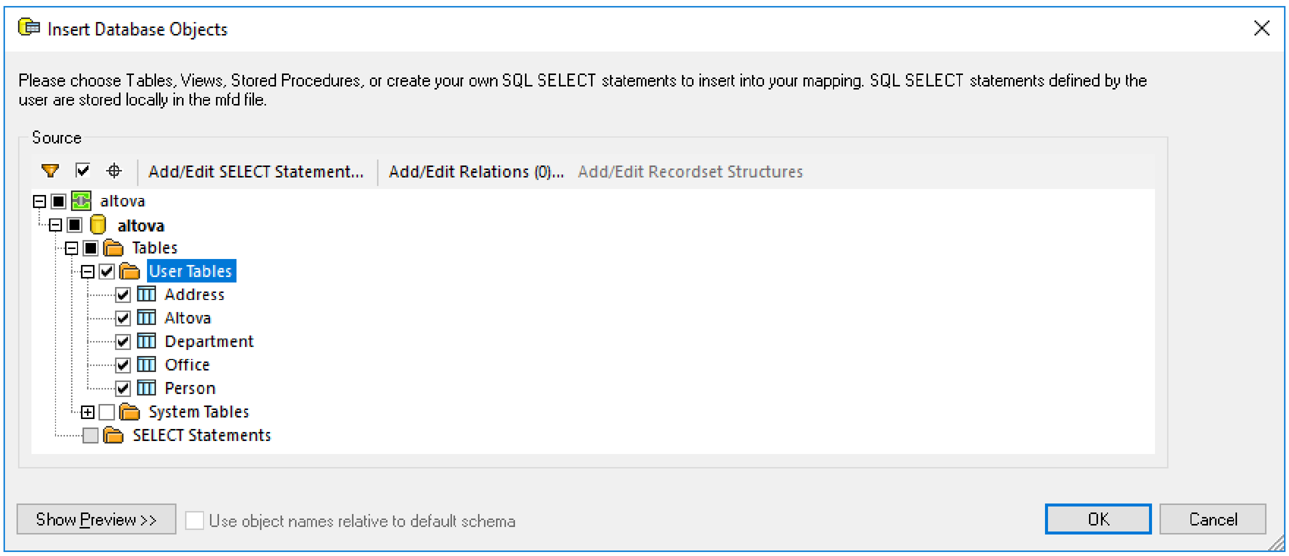


4.2.2.3 Adding, Editing, and Removing Database Objects

Some databases can have a large number of objects (such as schemas, tables, views, and so on). This topic shows you how to get on the mapping only those database objects that are required for mapping purposes. Below, we will be using a sample Access database; the instructions are similar for other database types.

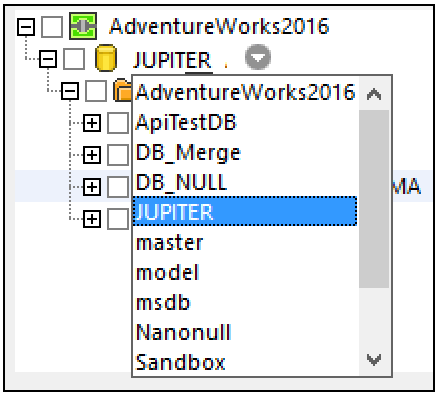
1. On the **Insert** menu, click **Database**.
2. Click **Connection Wizard**, and then click **Microsoft Access (ADO)**.
3. Click **Next**, and browse for the **altova.mdb** available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder.

A dialog box appears, enabling you to select the database objects to be included in the mapping.



The top node in the structure indicates the database connection. The subsequent structure varies depending on the database kind. For example, Oracle and IBM DB2 databases have a schema node under the connection node, while other database kinds have a catalog (database) node. Bold font indicates the default catalog (database) or schema, as applicable.

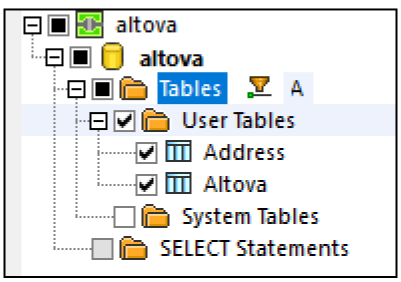
If your database user account has access to multiple databases or schemas on the server, you can switch to the required one by clicking the icon. The example below is illustrative of SQL Server; the structure may vary in other database kinds.



To include a database object (for example, a table) in the mapping, select the check box next to it. For the purpose of this example, select the check box next to **User Tables**.

The **Object Locator** button allows you to find specific database items. Select a particular object (or type its name) in the combo box which appears in the lower area of the dialog box.

The **Filter** button allows you to filter objects by name. Once you click the Filter button, a filter icon is available next to objects which support filtering (in this example, "Tables"). Click the filter icon to choose whether the object name should begin with, end with, be equal with, or contain the search text. Now you can enter the search text next to the filter (in this example, "A"):



The **Show checked objects only** button displays those items where a check box is active.

The **Add/Edit SELECT Statement** button enables you to add or edit custom SELECT statements for the current database. The data returned by such statements then becomes available as mapping source. For more information, see SQL SELECT Statements as Virtual Tables [357].

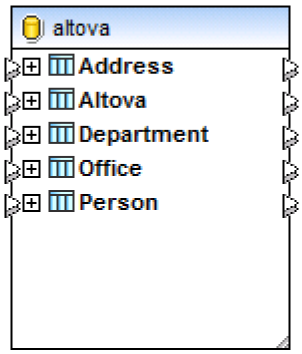
The **Add/Edit Relations** button enables you to define local primary and foreign key relationships between fields in the database, in addition to those that may already be present. For more information, see Defining Local Relationships [267].

The **Add/Edit Recordset Structures** button applies for databases that support stored procedures. It is only enabled if a stored procedure is currently selected from the database tree. For more information, see Stored Procedures [390].

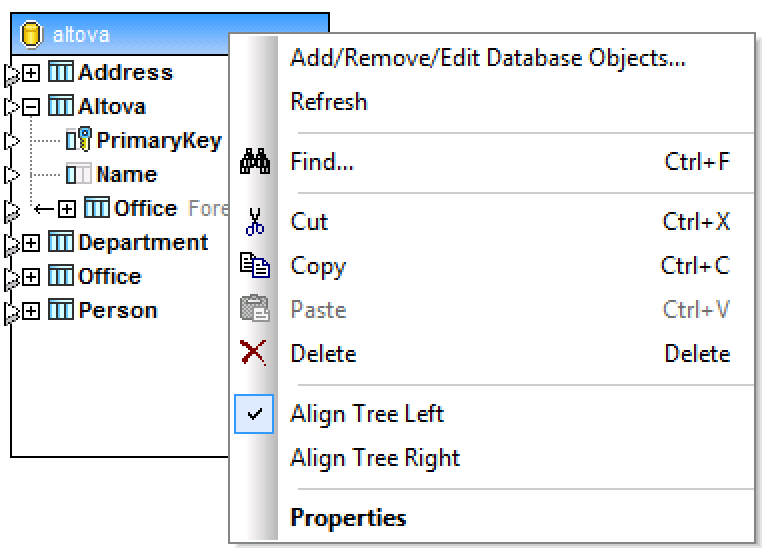
The **Show Preview** button enables you to quickly preview the data of the currently selected table or view. Note that you can also browse and query a database independently of the mapping process, by using the Database Browser. For more information, see Browsing and Querying Databases 376.

The **Strip schema from table name** check box is enabled only for databases that support qualifying objects with a relative name. This is useful if you plan to switch to a different database later. For more information, see Switching Databases and Schemas 255.

When you are ready to add the database objects to the mapping, click **OK**. Only the selected tables, views, etc. will appear on the database component, and you can draw mapping connections to or from them in the standard MapForce way.



To change at any time the database objects, right-click the component, and select **Add/Remove/Edit Database Objects**.



4.2.2.4 Switching Databases and Schemas

To change a database component on the mapping so that it uses a new database connection, open the database component settings [279], click **Change**, and follow the wizard steps to connect to the new database. If you need to change only the database objects that are displayed on the mapping, right-click the title bar of the database component, and select **Add/Edit/Remove Database Objects** from the context menu, see also Adding, Editing, and Removing Database Objects [252].

Be aware that, by default, changing the database does not remap all existing mapping connections to or from the database component. With most database kinds, however, it is possible to preserve the mapping connections after switching a database, if you take some preparation steps as described in this topic.

When selecting database objects as described in Adding, Editing, and Removing Database Objects [252], you can optionally choose whether database object names should be considered as relative to a default schema. In the context of databases, a "schema" is a logical grouping of database objects, typically, for security purposes such as assigning permissions. Schemas have slightly different behavior depending on the database kind.

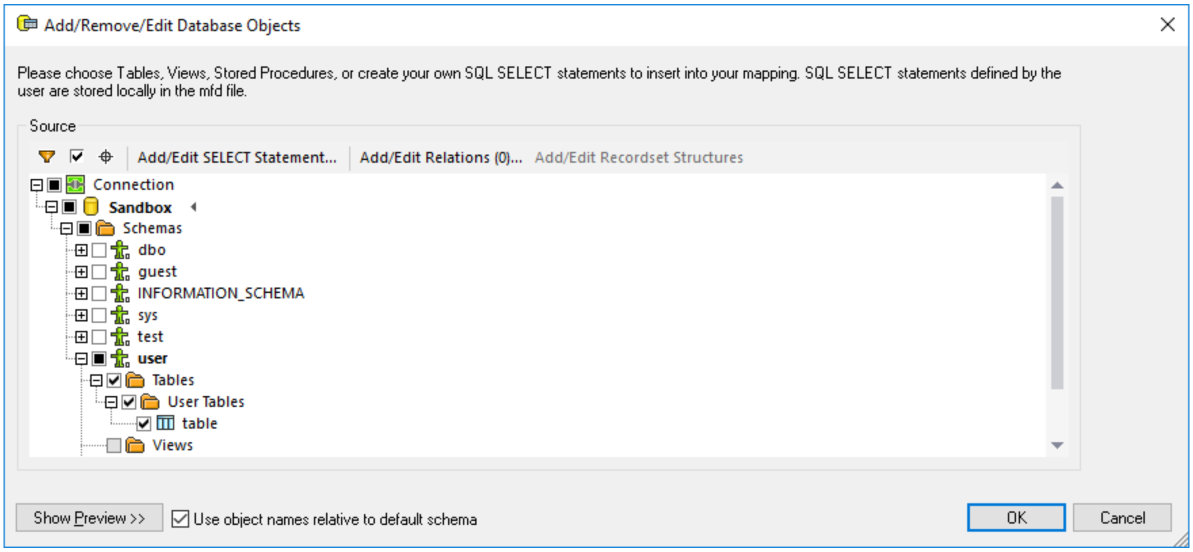
In MapForce, treating database objects names as relative to a schema is important if you plan to switch the mapping to a different database later (for example, to a production database). This is also useful if the database schema has been renamed on the server at some point, and so you need to update the mapping accordingly. If the new schema has the same structure as the one used at mapping design time, you can switch to it without having to change manually all mapping connections. This is possible only if you configure the object names to be relative to a default schema as described below.

Note the following:

- Using object names relative to a default schema is possible for those database types that support schemas: IBM DB2, IBM Informix, IBM Db2 for i (iSeries), Oracle, PostgreSQL, Progress OpenEdge, SQL Server and Sybase.
- This feature is supported starting with MapForce version 2020.
- It is not possible to use relative names if the database component includes local relationships [267] or SELECT Statements as Virtual Tables [357].

To treat database objects names as relative to the default schema:

1. Right-click the title bar of the database component on the mapping, and select **Add/Remove/Edit Database Objects** from the context menu.
2. Select one or more objects that *belong to the default schema*, or to the *default catalog (database) and schema*, as applicable. The default database and schema are shown in bold. In the example below, the default catalog is **Sandbox** and the default schema is **user**. This structure is specific for SQL Server databases and may vary in other database kinds. For more information about the displayed structure, see Adding, Editing, and Removing Database Objects [252].



3. Select the **Use object names relative to default schema** check box. Note that this check box is grayed out if the database does not support relative object names.

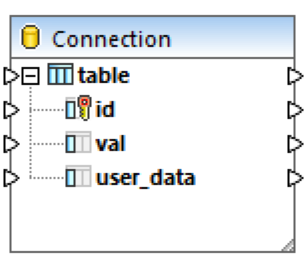
If the objects that you require on the mapping are in a different schema (not the default one), you have the following alternatives:

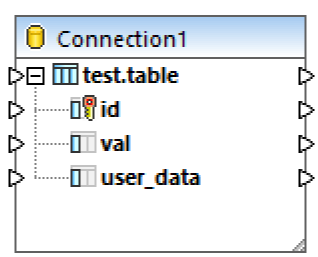
- Connect as another database user that has access to the required default schema.
- Reconfigure the database server so as to change the default schema of the existing database user, provided that you have the required privileges.

Taking SQL Server as example, a database administrator could change the default schema of a database user with a statement such as the one below (which assumes the current catalog is "Sandbox" and both the user and the schema already exist).

```
USE [Sandbox]
GO
ALTER USER [test_user] WITH DEFAULT_SCHEMA=[test_schema]
GO
```

Database objects that are relative to the default schema are shown on the mapping with a relative name. Consider differences between the following two database components:





In the left component, the **Use object names relative to default schema** check box was selected, therefore, the table is shown with a relative name. In the right component, the check box was not selected, so the table name includes the schema name.

When database objects are referenced in a relative way, it becomes possible to switch the mapping to a new database or schema *without losing the mapping connections*, as follows:

- Open the database component settings [279] and click **Change**. Follow the wizard steps to connect to the new database *as a new user*. If you have qualified the objects with a relative name, and if the new database has the same structure, then all the connections on the mapping will be updated automatically. Specifically, they will now match *the default catalog and schema of the new database user*.
- If you need to perform the switch on a recurrent basis, it is recommended to define the database connection as a Global Resource [1397]. For example, the Global Resource could have two configurations: a default one for the development database, and a production configuration. Assuming that both databases have the same structure, you can switch between databases by changing the required configuration from a drop-down list:

  For more information about this scenario, see Example: Switch Databases [1408].

> If database objects appear in red after switching, this indicates that they do not exist in the new database schema.

Code generation

The **Use object names relative to default schema** check box affects the generated C#, C++, or Java program code. When this check box is selected, all the database references become relative in the generated code also. The following is an example of a generated SQL statement (in C#) when the check box is not selected. Note that the table name includes the schema name.

```
var3_NewStatement = (Altova.Db.Dbs.NewStatement(closure.var1_Connection_Instance, 1,
"SELECT [id], [val], [user_data] FROM [user].[table]"));
```

The following is an example of a generated SQL statement (in C#) with the check box selected:

```
var3_NewStatement = (Altova.Db.Dbs.NewStatement(closure.var1_Connection_Instance, 1,
"SELECT [id], [val], [user_data] FROM [table]"));
```

Note: There is also a legacy way to strip schema names in generated code, by selecting the **Strip schema names from table names** check box from the database component settings [279]. The legacy approach is supported only for backward compatibility and should be avoided.

4.2.2.5 Handling Database Relationships

Relational databases, as their name implies, normally have relationships defined between their tables. Taking as example the **altova.mdb** database found in the folder

<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial, several relationships exist in it, for example:

- The sample company (corresponding to the table "Altova") consists of one or more offices (for example, in Brenton and in Vereno). In database terminology, there is a one-to-many relationship between the "Altova" and "Office" tables. In other words, for each **PrimaryKey** record in the "Altova" table there can be multiple **ForeignKey** records in the "Office" table. Any "Office" record where **ForeignKey** value corresponds to **PrimaryKey** value in "Altova" should therefore be considered an office of "Altova".
- Each office consists of one or several departments (for example, "Marketing", "IT", "Development"). Again, there is a one-to-may relationship between "Office" and "Department" tables.
- Finally, each department consists of one or several people. Hence, the one-to-many relationship between the "Department" and "Person" tables.

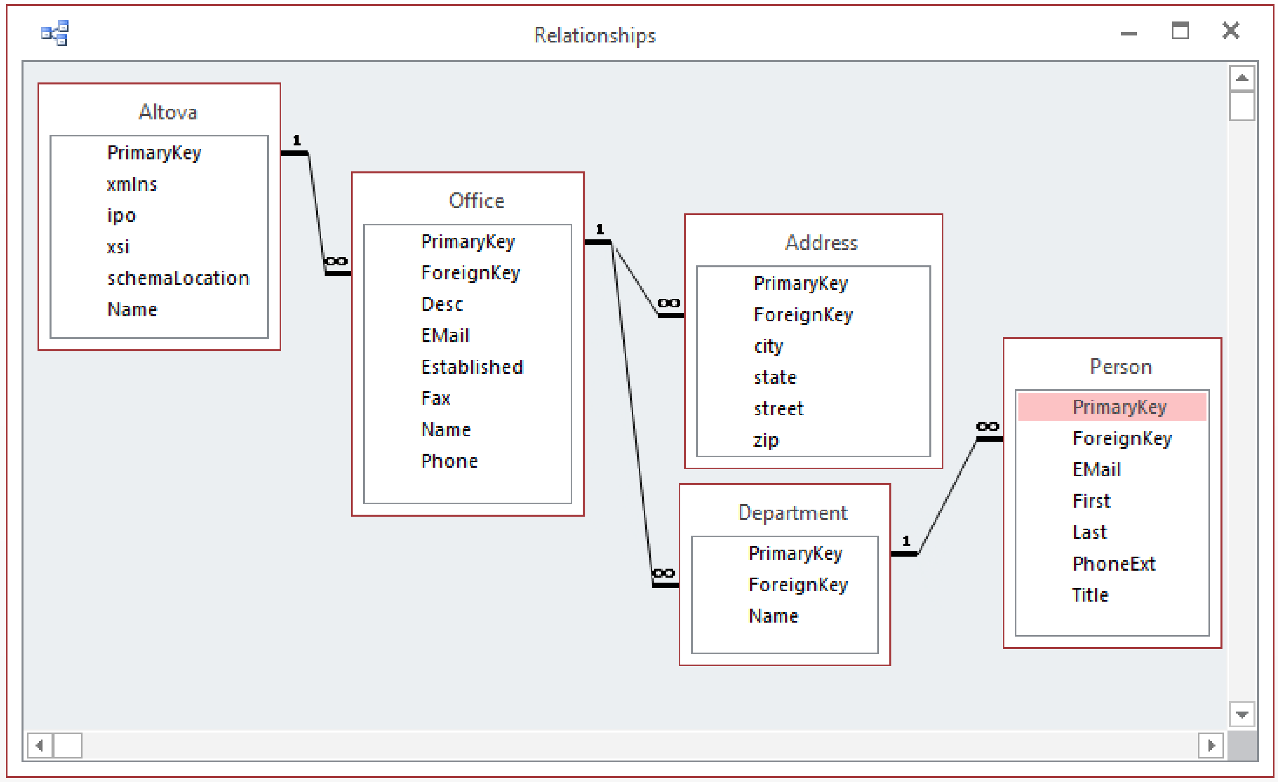
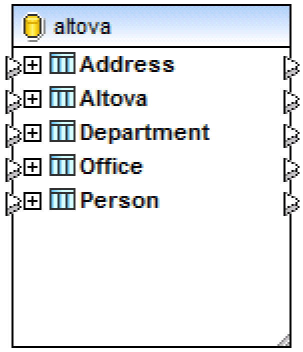


Table relationships in altova.mdb database (Microsoft Access "Relationships" view)

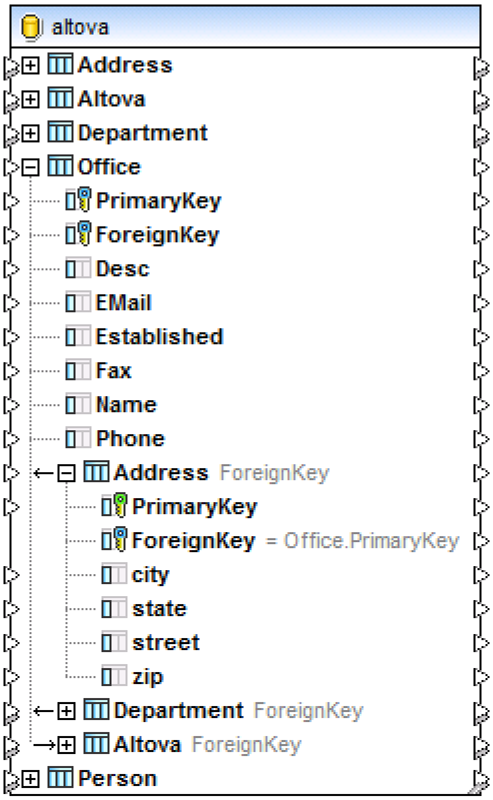
Relationships between database tables are important for mapping purposes. MapForce keeps track of such database relationships when you add a database to the mapping. This enables you to preserve the database relationships when mapping data to or from a database. To understand this concept better, add the **altova.mdb** database to the mapping (using the **Insert | Database** menu command). Let us call each of the tables below a "root" table:



"Root" tables

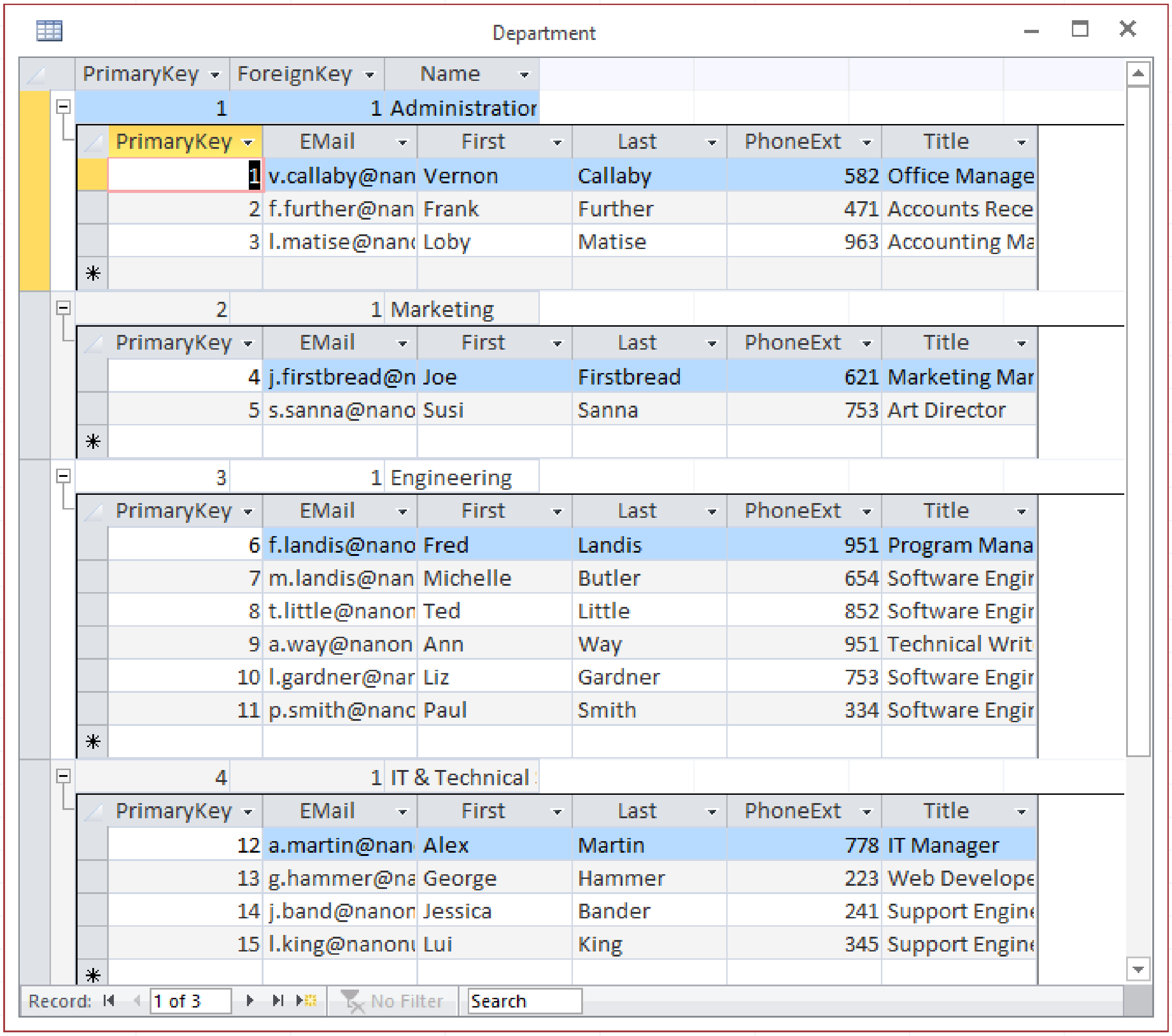
Expanding a "root" table displays all related tables beneath it in a tree view. For example, if you expand the **Office** table, notice how the related table hierarchy is displayed:

- A left arrow (←) in front of a table denotes that this is a child table. For example, **Address** is a child of **Office**. **Department** is also a child of **Office**, as well as a "sibling" table of **Address**, so both have the same indentation level. As you can see, the relationship on the mapping corresponds to the "Relationships" diagram above.
- A right arrow (→) in front of a table denotes a parent table. For example, **Altova** is a parent of **Office**.



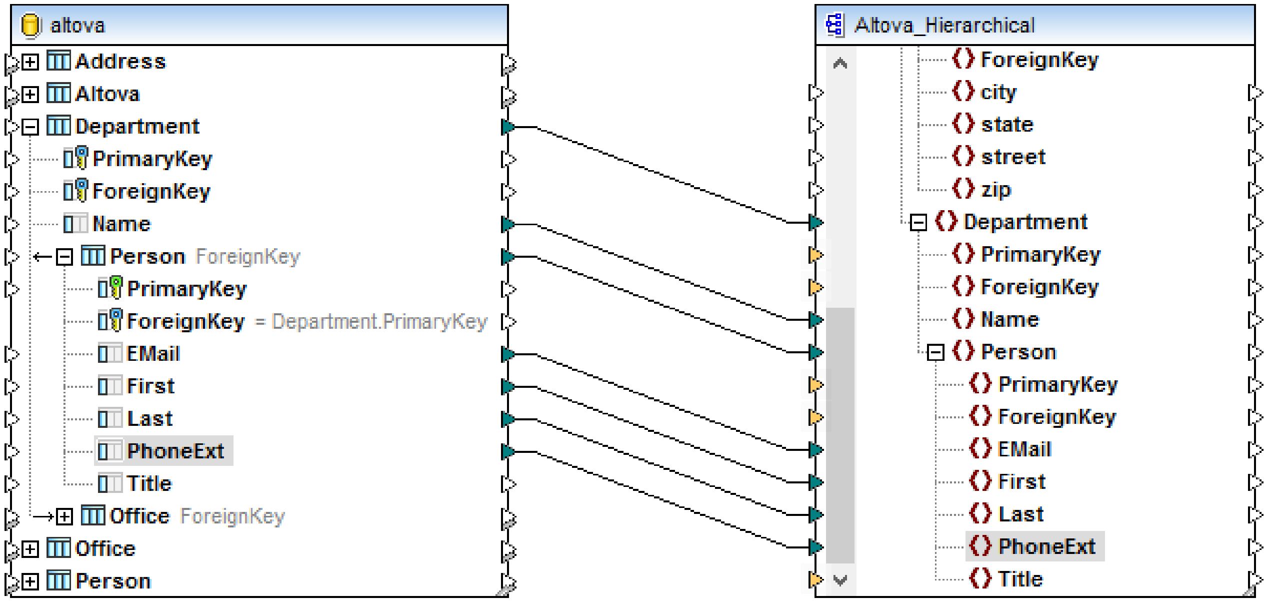
Tables relationships in MapForce (altova.mdb database)

This hierarchical representation of tables helps you preserve the existing database relationships when your mapping reads from or writes to a database. For example, let's assume you want to get all the records from the **Person** table into an XML file, grouped by their department. Specifically, your XML file should link every person to a department, similar to the **altova.mdb** database used in this example:



As illustrated above, the "Administration" department has three people, "Marketing" has two people, "Engineering" has six people, etc.

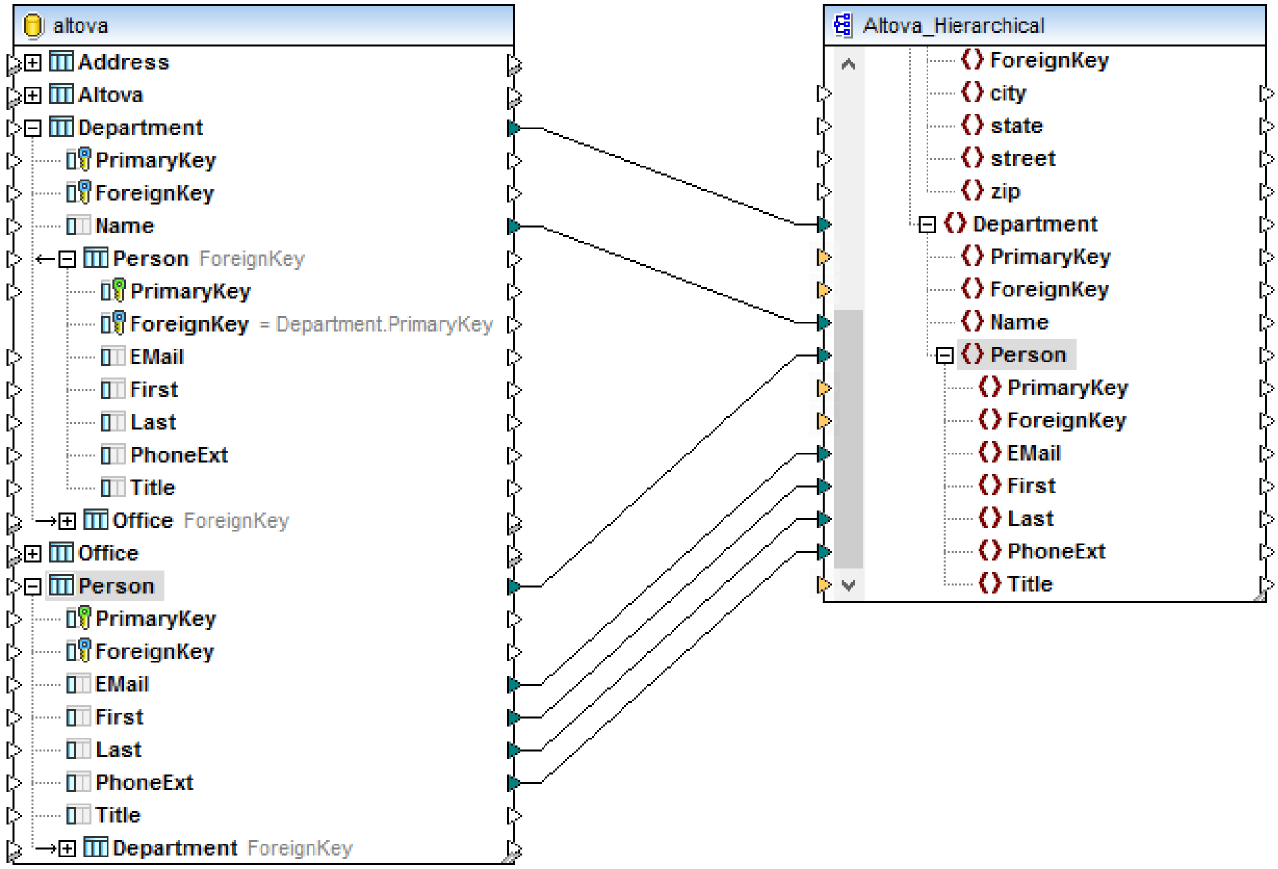
When mapping data from this database, if you want every person to be distributed to the correct department, it is important that you use **Department** as "root" table, and then map from the **Person** table which is child of **Department**:



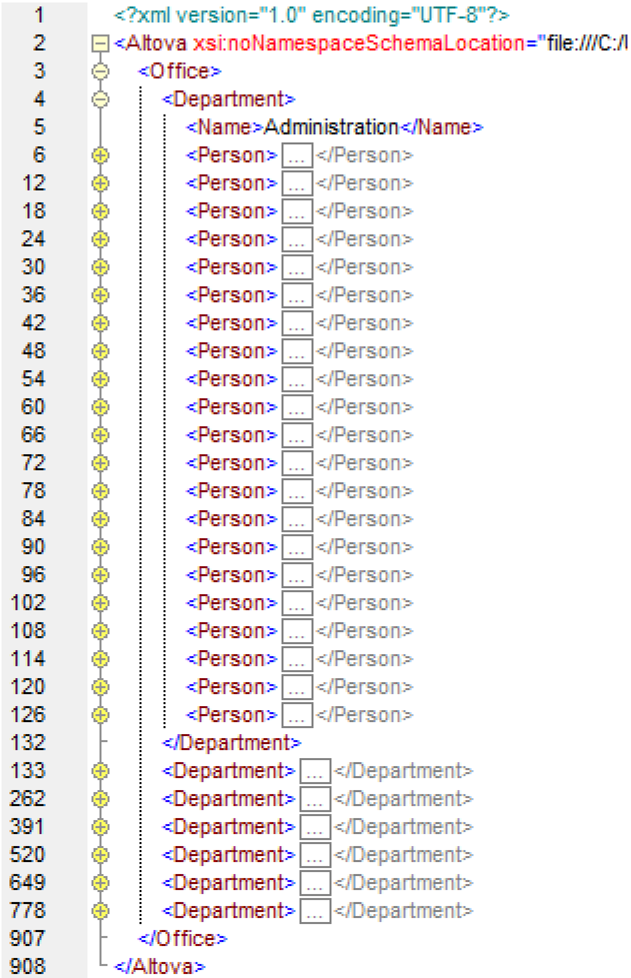
The mapping above is a modified **DB_Altova_Hierarchical.mfd** from the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. When you preview the mapping, the result is that each person is grouped by department, which was the intended behaviour. That is, "Administration" has three people, "Marketing" has two people, "Engineering" has six people, etc.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Altova xsi:noNamespaceSchemaLocation="file:///C:
  <Office>
    <Department>
      <Name>Administration</Name>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
    </Department>
    <Department>
      <Name>Marketing</Name>
      <Person> ... </Person>
      <Person> ... </Person>
    </Department>
    <Department>
      <Name>Engineering</Name>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
    </Department>
    <Department> ... </Department>
    <Department> ... </Department>
    <Department> ... </Department>
    <Department> ... </Department>
  </Office>
</Altova>
```

Now have a look at the slightly modified mapping below, where connections have been deliberately drawn so that both **Department** and **Person** are "root" tables.



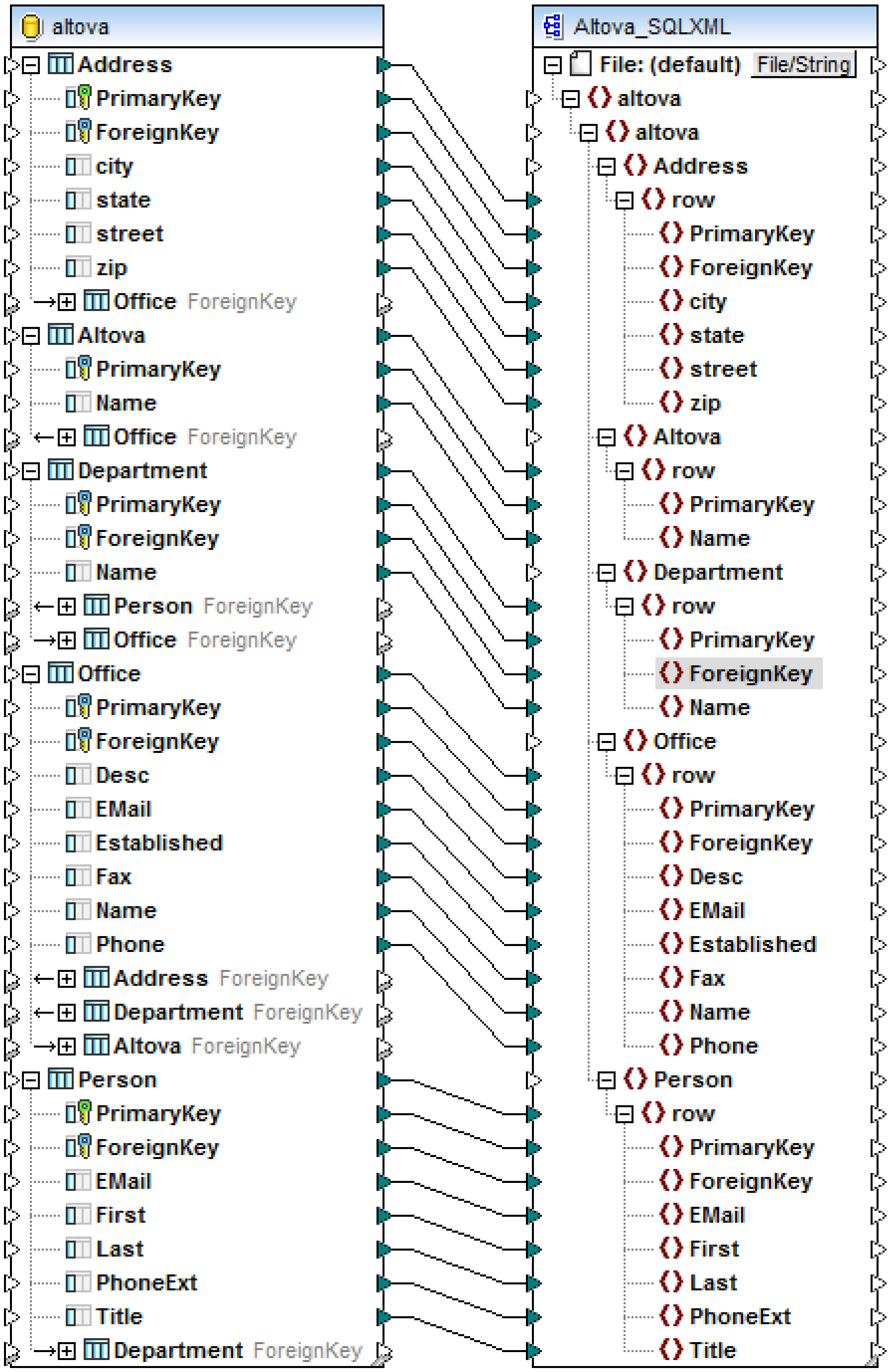
This time, when you preview the mapping, all persons (regardless of their source department) are grouped under each target department, which was not the intended behaviour. That is, "Administration" has 21 people, "Marketing" has 21 people, "Engineering" has 21 people, etc.



In the second example, the database relationships are disregarded, due to the way the connections were made.

Therefore, when you want to preserve database relationships, make sure that connections are drawn to or from the same "root" table, which contains the child tables whose relationships you want to preserve. This works in the same way for both source and target databases. For examples of database mappings which preserve relationships, see the **DB_Altova_Hierarchical.mfd** and **Altova_Hierarchical_DB.mfd** files available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder (see also Inserting Data into Multiple Linked Tables 286).

There might also be cases when you do not want to preserve database relationships. For example, let's assume that you want to export all data from the **altova.mdb** database to a flat XML file adhering to the SQL/XML specification (Part 14 of the Structured Query Language (SQL) specification). This kind of mapping is illustrated by the **DB_Altova_SQLXML.mfd** sample, available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. The goal of the mapping is to get database data as a flat XML file. The target SQL/XML schema was generated with XMLSpy, using the **Convert | Create XML Schema from DB Structure** menu command.



DB_Altova_SQLXML.mfd

As illustrated above, every database table has a corresponding element in the target XML. When you preview the mapping result, you can see that the actual database rows from each table are written to "row" elements in the target.

```
<?xml version="1.0" encoding="UTF-8"?>
<altova xsi:noNamespaceSchemaLocation="file://
  <altova>
    <Address>
      <row>...</row>
      <row>...</row>
    </Address>
    <Altova>
      <row>...</row>
    </Altova>
    <Department>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
    </Department>
    <Office>
      <row>...</row>
      <row>...</row>
    </Office>
    <Person>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
      <row>...</row>
    </Person>
  </altova>
</altova>
```

As the XML output shows, no hierarchies exist between the XML elements; it is a flat SQL/XML structure. The database relationships were ignored, because we intentionally mapped data from multiple "root" tables.

4.2.2.6 Defining Local Relationships

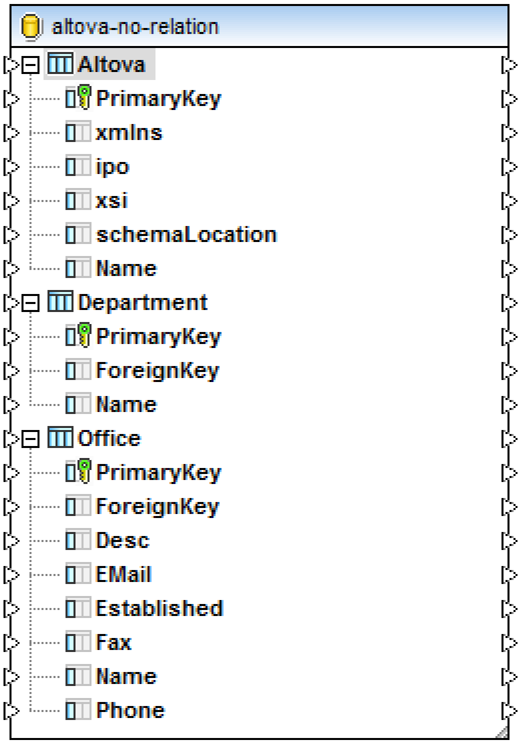
When database tables do not have explicitly defined relationships between them, you can define such relationships locally in MapForce. In particular, you can create, from MapForce, primary and foreign key relationships between columns of different tables, without affecting the database in any way. Any database columns can be used as primary or foreign keys. Also, new relations can be created, in addition to those existing in the database. Locally defined relationships are saved together with the mapping.

These "on-the-fly" relationships are called **Local Relations** in MapForce. The following table lists all the possible fields between which you can define local relations. Mixed relationships are possible (for example, mapping the output of a stored procedure to a database column). The fields taking part in the relationship must have the same, or a compatible, data type.

Primary/unique key	Foreign key
• Column of a database table or view • Output parameter or return value of a stored procedure, see also Stored Procedures [390] • Column of a recordset returned by a stored procedure* • Column of a user-defined SELECT statement, see also SQL SELECT Statements as Virtual Tables [357].	• Column of a database table or view • Input parameter of a stored procedure • Input parameter of a user-defined SELECT statement

** Applicable if the stored procedure is called either as data source (without parameters) or as a function (with input and output parameters). In order for the recordset to become available for selection, you must execute the stored procedure once, to retrieve the recordset.*

The **altova-no-relation.mdb** database used in this example is a simplified version of the **altova.mdb** database supplied with MapForce. The "Person" and "Address" tables, as well as all table relationships have been removed in Microsoft Access. As illustrated below, none of the tables visible in the **altova-no-relation** tree have any child tables; all tables are on the same "root" level. The content of each table is limited to the fields it contains.

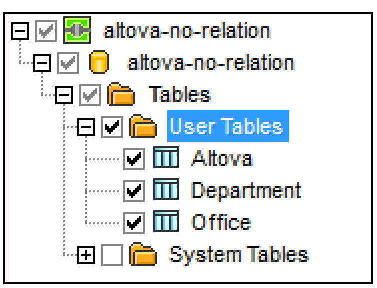


Database structure with no explicit relationships

The aim of the example is to display the offices of "Altova" and show the departments in each office. Note that, in the **altova-no-relation.mdb**, the primary and foreign key relationships do not exist explicitly, as mentioned above. They exist only logically (implicitly), so we will be re-creating them locally in MapForce to achieve the goal of the mapping.

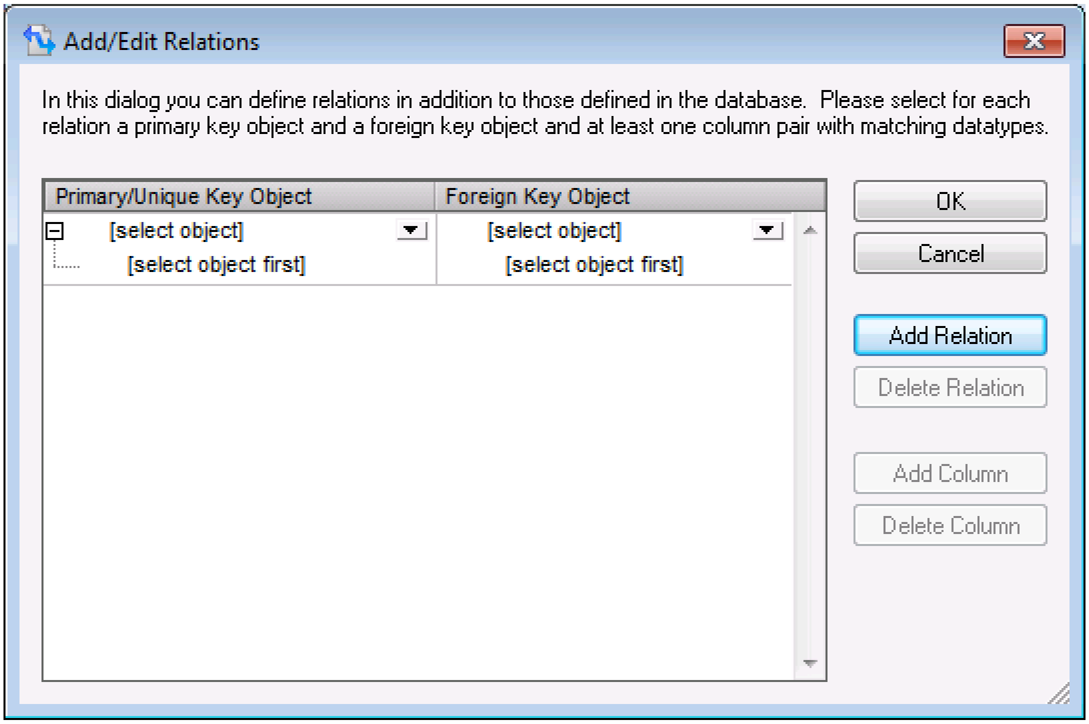
Local relations can be defined while adding a database to the mapping, or by right-clicking an existing database component and selecting **Add/Remove/Edit Database Objects** from the context menu, as illustrated in the steps below.

1. On the **Insert** menu, click **Database**.
2. In the connection wizard, click **Microsoft Access (ADO)**, and then click **Next**.
3. Browse for the **altova-no-relation.mdb** database available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder, and click **Connect**.
4. Select the **User Tables** check box.

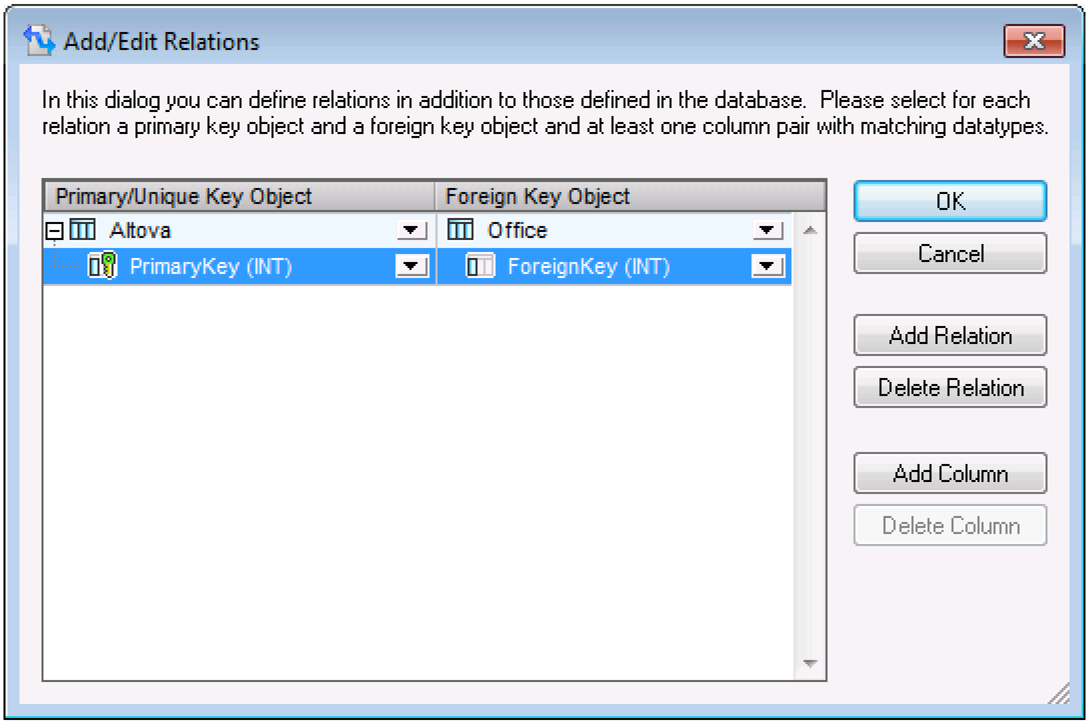


5. Click the **Add/Edit Relations** button in the icon bar.

6. The "Add/Edit Table Relations" dialog box opens. Click **Add Relation**.

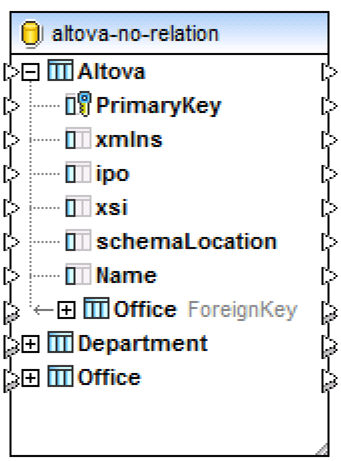


7. Select values from the two drop-down lists so as to create a primary and foreign key relationship between the "Altova" and "Office" tables, as illustrated below. The two drop-down lists allow you to select the tables or database objects you want to create relations for. The left list specifies the object which stores the primary/unique key, while the right one specifies the foreign key object. The Primary/Unique Key object will be the parent object in MapForce, and the Foreign Key object will be shown as child in the database component (see also Handling Database Relationships 257).

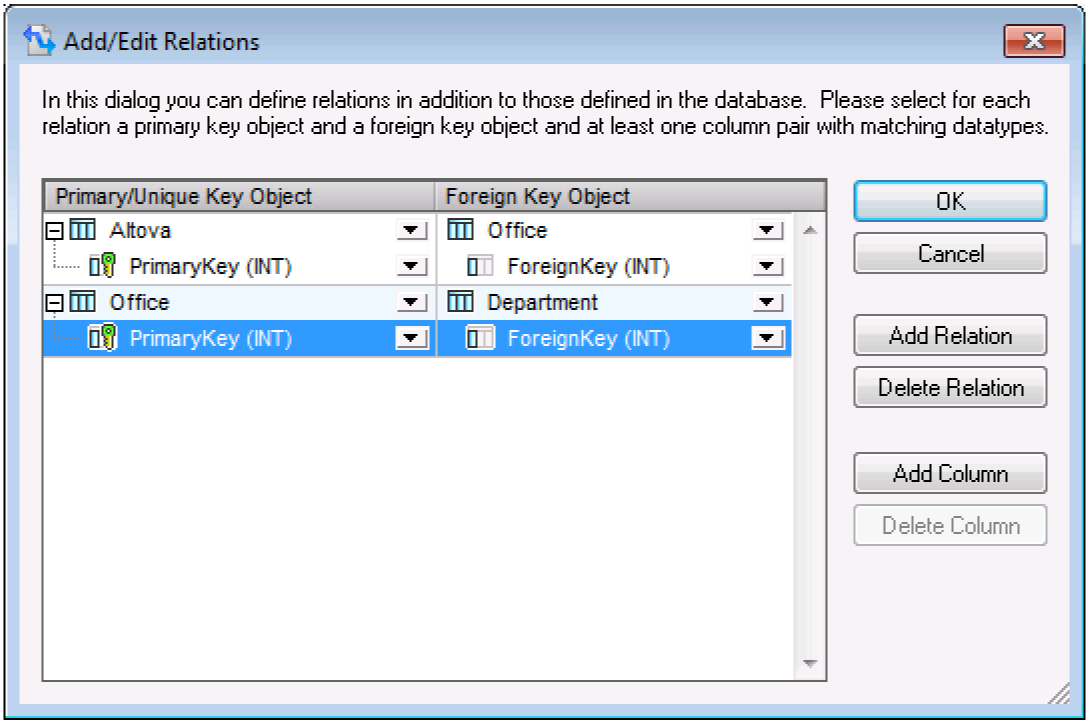


8. Click **OK** to complete the local relation definition, and then click the **Insert** button to insert the database into the mapping area.

At this stage, you have created a local relationship between the **PrimaryKey** column of the "Altova" table and the **ForeignKey** column of the "Office" table. As illustrated below, the "Altova" root table is now a parent to the "Office" table. Namely, the Office table is shown as a related table below the Altova table with its own expand icon.



However, the mapping goal is not yet complete. To complete the mapping goal, use the same method to create a relationship between the **Office** and **Department** tables, as shown below.

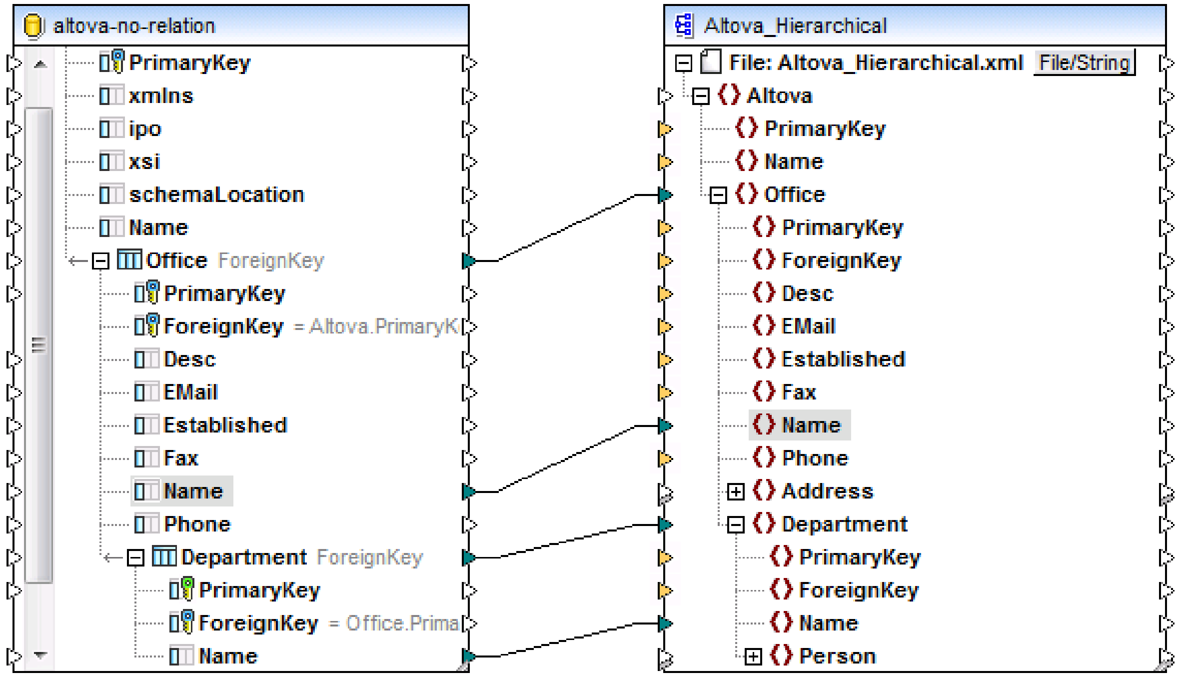


To open again the "Add/Edit Relations" dialog box, right-click the database component, and select **Add/Remove/Edit Database Objects** from the context menu.

Finally, add the target schema to the mapping as follows:

1. On the **Insert** menu, click **Insert XML Schema/File**.
2. Browse for the **Altova_Hierarchical.xsd** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. When prompted to supply a sample XML file, click **Skip**. When prompted to select a root element, select "Altova".

Notice that, in order to preserve relationships between tables in the target XML, all connections were drawn from the same "root" table, hierarchically (in this case, "Altova"). For more information, see Handling Database Relationships 257.



Having defined the mapping as shown above, click the **Output** tab, to preview the result. The mapping result shows the department elements nested under each respective office, which was the intended goal of this mapping.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Altova xsi:noNamespaceSchemaLocation="file:///C:/U
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Department>
      <Name>Administration</Name>
    </Department>
    <Department>
      <Name>Marketing</Name>
    </Department>
    <Department>
      <Name>Engineering</Name>
    </Department>
    <Department>
      <Name>IT &amp; Technical Support</Name>
    </Department>
  </Office>
  <Office>
    <Name>Nanonull Partners, Inc.</Name>
    <Department>
      <Name>Administration</Name>
    </Department>
    <Department>
      <Name>Marketing</Name>
    </Department>
    <Department>
      <Name>IT &amp; Technical Support</Name>
    </Department>
  </Office>
</Altova>
```

4.2.2.7 Executing Mappings Which Modify Databases

When a mapping modifies database data in any way (for example, by inserting, updating, or deleting records), the changes are applied to the database by the engine that executes the mapping. The engine that executes the mapping can be MapForce, MapForce Server (both standalone or under FlowForce Server management), or the execution environment of the code generated for C++, C#, or Java.

When you preview the mapping result directly in MapForce (by clicking the **Output** tab), an update script is displayed. The script is not actually executed against the database until you take this action explicitly; it is available for preview only.


```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
Data Source=C:/Users/altova/Documents/Altova/MapForce2016/MapForceExamples/AltovaTarget.mdb;Provider=Microsoft.Jet.OLEDB.4.0
*/

-- begin transaction

DELETE FROM [Address]

DELETE FROM [Person]

DELETE FROM [Department]

DELETE FROM [Office]

DELETE FROM [Altova]

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM [Altova]
-->>> %PrimaryKey1%

INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Organization Chart', '%PrimaryKey1%')
```

Output preview of a mapping which modifies a database (Altova_Hierarchical_DB.mfd)

> The script shows pseudo-SQL statements just for information purposes; it does not directly reflect the statements that are to be executed. For example, if multiple actions are defined against a table (such as "Update if... Insert Rest"), only the first action is shown in the preview, since the second action is executed conditionally.
>
> The pseudo-SQL script must not be manually applied to the database using SQL tools other than the execution engines mentioned above. The script may contain formatting of values not "understood" by external SQL editors.

If you want to apply the mapping changes to the database directly from MapForce, click the **Run SQL-Script** command available in the **Output** menu. Remember that this action will actually modify the database with immediate effect.

When the mapping is executed with MapForce Server (both standalone or under FlowForce Server management), the changes to the database are applied with immediate effect. The same happens in the generated code: the database changes are applied when you compile and run the code (for example, by clicking the **Run** command in Visual Studio).

> Your MapForce installation includes several example databases (Microsoft Access or SQLite files) available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. It is advisable not to apply database changes from MapForce, using the **Run SQL-Script** command, against any of the example databases supplied with MapForce; this may render the examples unusable. A simple way to avoid overriding original data is to back up the entire **<Documents>\Altova\MapForce2023\MapForceExamples** folder before updating any files in it.

For information about running mappings in execution environments other than MapForce, see:

- Deploying Mappings to FlowForce Server [1381]
- Compiling Mappings to MapForce Server Execution Files [1378]
- Code Generator [1492]

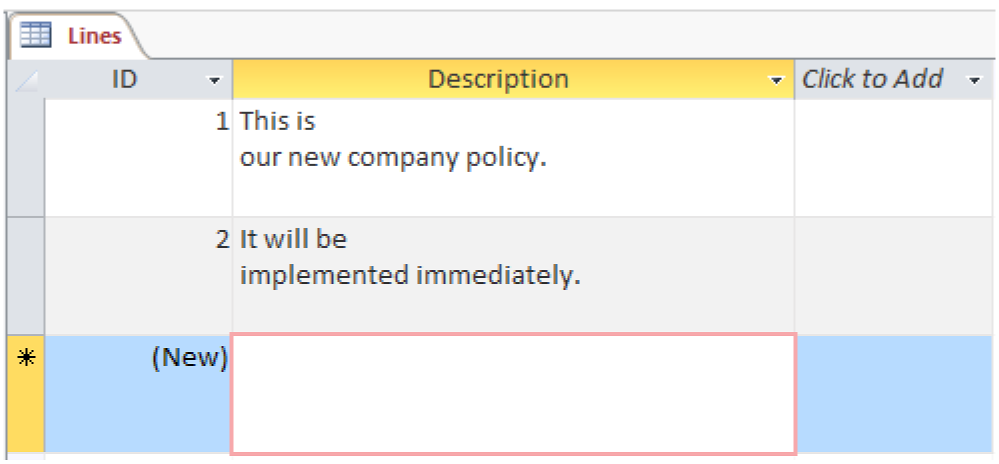
4.2.2.8 Replacing Special Characters

When transforming database data, you might need to remove specific special characters, such as the carriage return/line feed (CR/LF) characters, from the data source. To achieve this, you can use any of the following approaches:

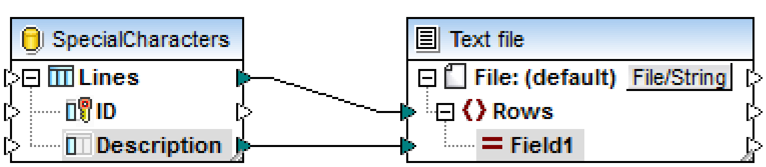
1. Define a node function for the specific database field (or even multiple fields) that you need to process. The node function will receive the value of the database field as input, apply to it some processing, and then return the outcome to the mapping. For more information about this technique, see Defaults and Node Functions [912].
2. Process database values with the help of MapForce built-in functions. For example, to identify specific characters, including control characters, you can use the **`char-from-code`** [1056] function from the "core" library. To replace values, use the **`replace`** [1126] function from the "lang" library.

This topic illustrates the second approach. The database used in this example is Microsoft Access but the same strategy can be applied with other database types as well.

Consider a Microsoft Access database consisting of a table "Lines" which has two columns: "ID" and "Description".



The goal is to extract each description to a CSV file (one description per line); therefore, a mapping to achieve this goal could look as follows:

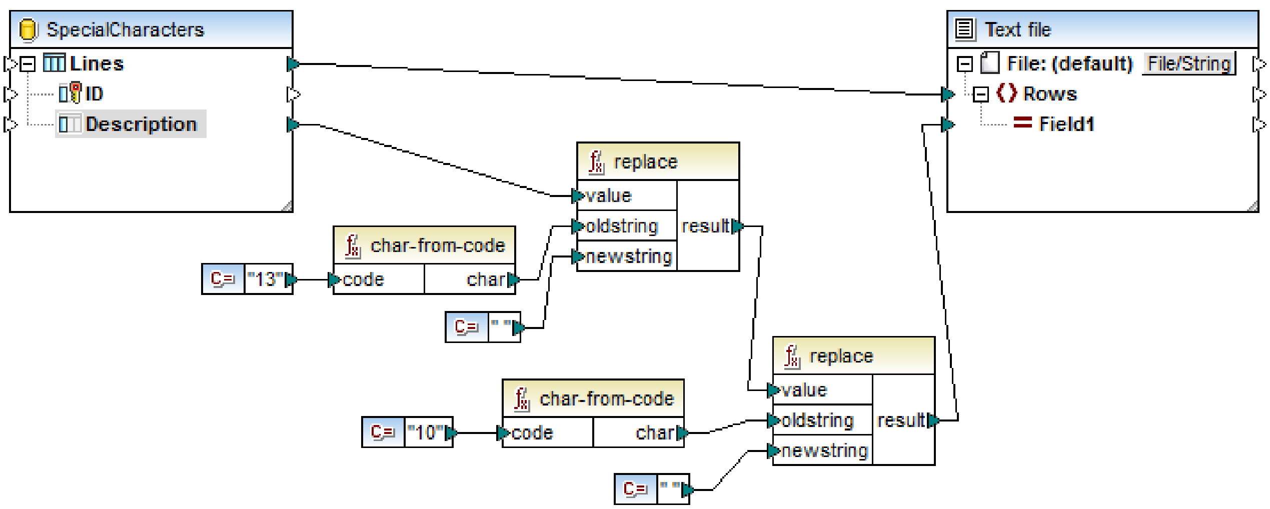


However, because each "Description" row in Access contains multiple lines separated by CR/LF characters, the mapping output includes line breaks also, which is not the intended result:

```
"This is
our new company policy."
"It will be
implemented immediately."

```

To overcome this problem, we are going to add to the mapping the **`char-from-code`** and **`replace`** functions from the MapForce built-in library (see also Add a Function to the Mapping [905]). Every description must be processed so that, whenever the characters above are encountered, they should be replaced by a space character.

In the Unicode chart (http://www.unicode.org/charts/), the LF and CR characters correspond to **hex 0A | dec 10** and **hex 0D | dec 13** characters, respectively. Therefore, the mapping has to be modified to convert the decimal Unicode values 13 and 10 to a string, so as to allow further processing by the **`replace`** function.



If you preview the mapping now, notice that the CR/LF characters within each database field have been replaced by a space.

```
This is  our new company policy.
It will be  implemented immediately.

```

4.2.2.9 Handling Null Values

To check at mapping runtime whether a database field is null, use the **`is-null`** [1075] and **`is-not-null`** [1075] MapForce library functions. To see from MapForce if a table has null fields, query it using the Database Browser (see Browsing and Querying Databases [376]).

To set a database field to null, use the **`set-null`** [1076] function.

To replace null database values with a string, use the **`substitute-null`** [1076] function. A sample mapping that illustrates this is **DB_ApplicationList.mfd** available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder.

For information about handling NULL value comparisons in mappings which update databases, see Handling Nulls in Database Table Actions [323].

For information about handling nulls when mapping database to or from XML documents, see Nil Values / Nillable [129].

4.2.2.10 Generating Sequential and Unique Values

When inserting data or updating a database, sometimes you might need to create "on-the-fly" sequential or unique values for those database fields which do not have any input from the mapped source. For such cases, use the following built-in MapForce library functions:

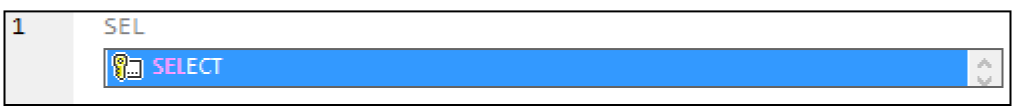
- **`auto-number`** [1008] (available in the "core | generator functions" library). This function is generally used to generate primary key values for a numeric field.
- **`create-guid`** [1103] (available in the "lang | generator functions" library). This function creates a globally-unique identifier (as a hex-encoded string) for the specific field.

Note that values for database fields can also be written using database-generated values. This option is available on the Database Table actions dialog box (see Database Table Actions Settings [326]) and is particularly useful when generating primary keys.

4.2.2.11 SQL Auto-Completion Suggestions

When you type SQL statements in certain contexts, MapForce may suggest text entries automatically. Auto-completion is available in the following contexts:

- SQL Editor (see Browsing and Querying Databases [376])
- "Custom SQL" text box in the "Database Table Actions" dialog box (see Database Table Actions Settings [326])
- "Enter a SQL SELECT statement" dialog box (see Creating SELECT Statements [357])



Auto-completion

Use the **Up** and **Down** keyboard keys to navigate through the list of suggestions. To pick a suggested entry, click it or press **Enter**.

To disable auto-completion suggestions:

1. On the **Tools** menu, click **Options** (or press **Ctrl+Alt+O**).
2. Under **Database**, click **SQL Editor**.
3. Under **Entry Helpers**, clear the **Automatically open** check box.

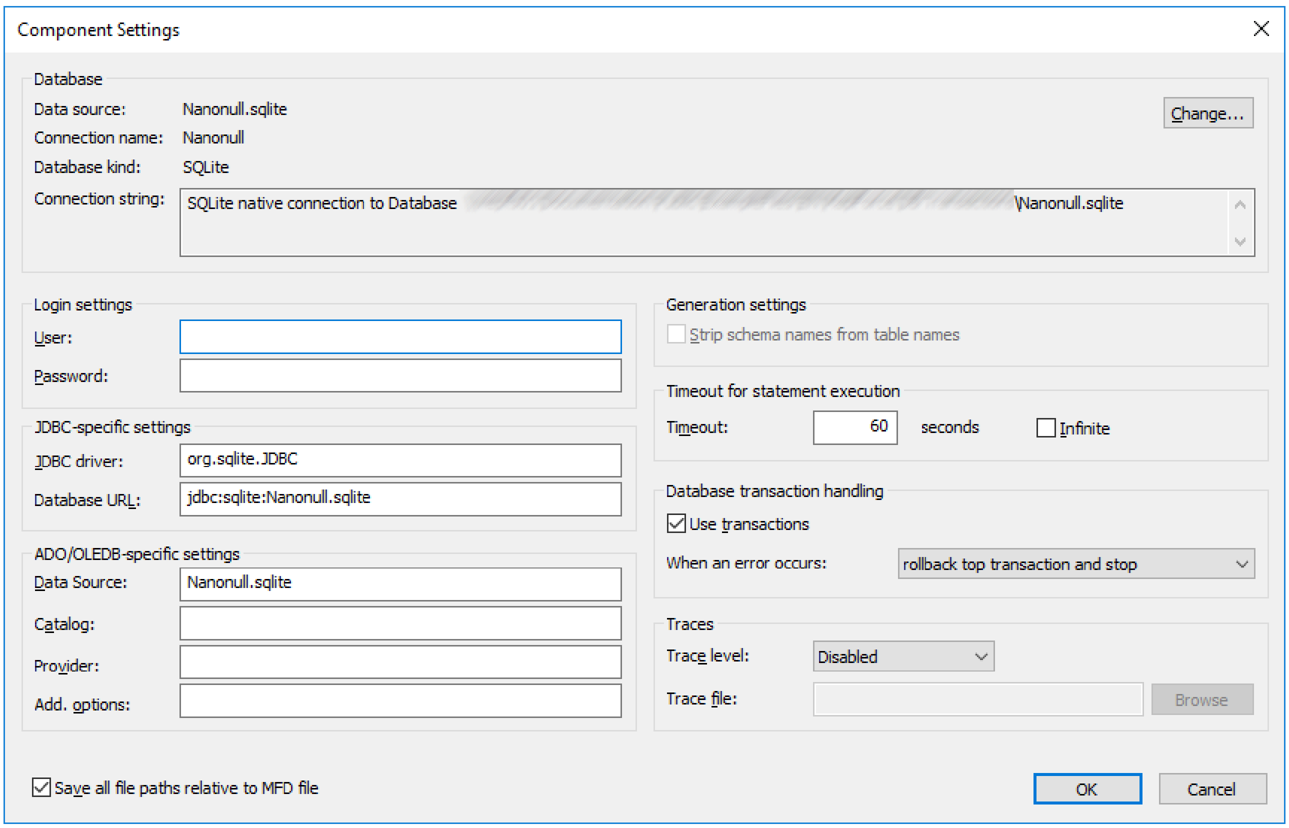
To invoke auto-completion suggestions manually as and when required:

- Press **Ctrl+Space**.

4.2.2.12 Database Component Settings

After you add a database component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component on the mapping, and then, on the **Component** menu, click **Properties**.
- Double-click the component title bar.
- Right-click the component, and then click **Properties**.



Database Component Settings dialog box

The available settings are as follows.

Database

This group displays database connection information. Click **Change** to select a different database, or to redefine the database objects in the existing database component. Connectors to tables of the same name will be retained. You can also change the tables in the component, by right clicking a database component and selecting **Add/Remove/Edit Database Objects**.

Data Source	Specifies the name of the current data source. For file-based databases, this can be a path on the file system. Use this setting to determine whether a file-based database was added to the mapping using an absolute or relative path. In case of relative paths, "Data Source" contains a path; in case of absolute paths, it contains just the database filename.
Connection Name	Specifies the name of the connection. This name is generated automatically by MapForce. Typically it is the same as the data source name, but it may also be a so-called "alias" name if you are connecting with Altova Global Resources 1397. If there are multiple database components with the same connection name on the mapping, then the name will take the form "<connection1>", "<connection2>", etc.
Database Kind	Specifies the kind of the database.
Connection String	Displays the current database connection string. This read-only field is generated based on the information you supply when creating or changing the database connection.

Login Settings

The login settings are used for all code generation targets and the built-in execution engine.

User	Enables you to change the user name for connecting to the database. Mandatory if the database requires a user name to connect.
Password	Enables you to change the password for connecting to the database. Mandatory if the database requires a password to connect.

JDBC-Specific Settings

These settings are used to connect to the database if the mapping contains a JDBC connection and is executed by generated Java code or by MapForce Server.

Note: ADO, ADO.NET, and ODBC connections are converted to JDBC (and the JDBC settings below apply) when the mapping is run on a Linux or macOS machine, see Database mappings in various execution environments 162.

JDBC Driver	Displays the currently active driver for the database component. The default driver is automatically entered when you define the database component. You can change the driver entered here to suit your needs. Make sure that the syntax of the entry in the Database URL field conforms to the specific driver you choose.

Database URL	URL of the currently selected database. Make sure that this entry conforms to the JDBC driver syntax, of the specific driver entered in the JDBC-driver field.

ADO/OLEDB-Specific Settings

These settings are used to connect to the database if the mapping contains an ADO connection and it is executed by generated C# or C++ code, or by MapForce Server running on Windows, see Database mappings in various execution environments 162. The **Data Source** and **Catalog** settings are not used by the built-in execution engine.

Data Source	Displays the name of the ADO data source.
Catalog	Displays the name of the ADO catalog.
Provider	Displays the currently active provider for the database component.
Add. Options	Displays any additional database options.

Generation settings

Generation settings apply to all code generation targets as well as the built-in execution engine.

Strip schema names from table names	Allows you to strip database schema names from generated code, only retaining the table names for added flexibility. Note that this option works only for SQL Select statements generated by MapForce. User-defined SQL-Statements, when creating virtual tables 357, will not be modified. This option is supported for backward compatibility and should be avoided. To treat database object names as relative to the default schema, use the approach described in Switching Databases and Schemas 255.

Timeout for statement execution

When a database is used as a target component, execution timeouts can occur due to server availability, traffic, long-running triggers, and other factors. This setting allows you to define how long the timeout period can be before the database connection is closed. The setting takes effect when querying database data as well as in generated C#, Java, and C++ code.

Timeout	Defines the time period, in seconds, that the execution engine must wait for a database response before aborting the execution of the database statement. The default setting for the execution timeout is 60 seconds.
Infinite	When enabled, this option instructs the execution engine to never time out.

Note: Timeout for statement execution is not applicable to SQLite databases.

Database transaction handling

Use transactions	Enables transaction processing when using a database as a target. Transaction processing is enabled for all tables of the database component when you select this option. For more information, see Handling Database Exceptions (415).

Traces

Trace level	Applicable for target database components. When tracing is enabled, the actions performed by the mapping against the database are logged in a trace file. You can choose to log all actions, only errors, or disable tracing completely. For more information, see Database Tracing and Error Logging (427).
Trace file	Specifies the file to which database trace information will be written when the mapping runs. The trace file is in XML format, see Trace File Structure (429). This path can be either absolute or relative, and is influenced by the **Save all file paths relative to MFD** file checkbox.

Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. Use relative paths if you intend to run the mapping with MapForce Server on a different operating system. See also Using Relative Paths on a Component (85).

4.2.3 Mapping Data to Databases

This section provides instructions and examples for transferring data from any mapping source supported by MapForce (for example, an XML file) to a target database. Use the following roadmap for a summary of available options.

I want to...	**Read this topic...**
Insert data into a target database table based on data supplied by the mapping...	• Inserting Data into a Table (283)
Control how primary key values are to be created...	• Inserting Data into a Table (283) • Inserting Data into Multiple Linked Tables (286)
Run a "preliminary" SQL statement to be executed before a table is modified by the mapping (for example, delete all records in the table, or a custom SQL statement)...	• Inserting Data into Multiple Linked Tables (286) • Database Table Actions Settings (326)

I want to...	Read this topic...
Preserve the hierarchical relationship of records in tables linked by foreign keys...	• Inserting Data into Multiple Linked Tables 286
Update a table conditionally...	• Updating a Table 291
Merge records into a database table (update some records, and also insert some other records into the same table), based on a condition...	• "Update if... Insert Rest" Action 294 • MERGE statements 301
Preserve database integrity when updating tables that are linked to other tables through foreign key relationships...	• Options for Child Tables When Updating a Parent Table 303
Define multiple actions against the same table (for example, delete a record if a condition is satisfied, otherwise insert a new record)...	• "Delete if..." Action 311 • "Ignore if..." Action 315
Preserve data integrity in case of failed mapping execution...	• Handling Database Exceptions 415
Insert multiple records into a database table in bulk (combine multiple INSERT statements in one query)...	• Bulk Inserts (MapForce Server) 319
Avoid undesired results when mapping data to target database tables that contain null values...	• Handling Nulls in Database Table Actions 323
View an example of how to create a mapping which updates a database.	• Example: Mapping Data from XML to SQLite 329

4.2.3.1 Inserting Data into a Table

A mapping can insert data into a database table from any of the source components supported by MapForce, including other databases. You can flexibly configure how the primary key of newly inserted records should be created. For example, the primary key can be taken from the mapping, generated by the database, or calculated based on existing key values in the database table.

This example shows you how to insert new records into an existing database table from an XML file. You will also configure how the primary key is to be generated. The example uses the following files:

- **altova-cmpy.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **altova.mdb** — the target database to be updated.

All files are available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. Below, the complete path to them will be omitted, for simplicity.

The mapping in this example modifies a sample database file. It is strongly recommended to back up the

original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases 274.

The goal of the mapping is to insert companies found in the **Altova-cmpy.xml** as new records in the "Altova" table of the **altova.mdb** database. If you open the source XML file, you will notice that it contains only one company, called "Microtech OrgChart". Therefore, the mapping must add a new record to the "Altova" table with the name "Microtech OrgChart". Also, a new primary key must be generated for it.

To achieve the mapping goal, we will take the steps below.

Step 1: Insert the source XML component

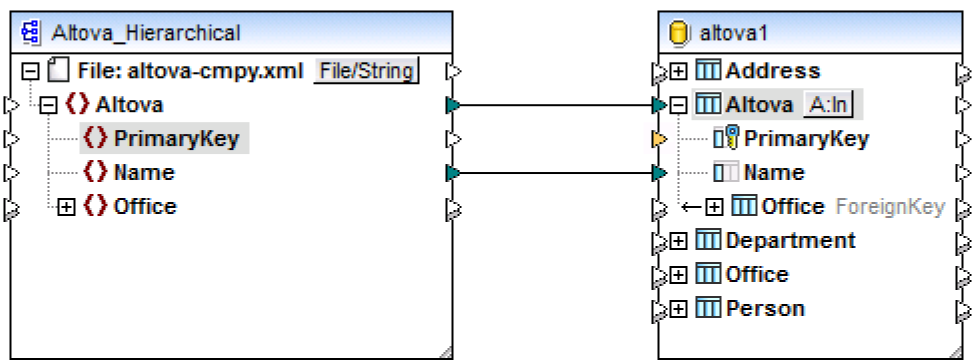
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **Altova-cmpy.xml**.

Step 2: Insert the target database

- On the **Insert** menu, click **Database**, and go through the wizard steps to connect to **altova.mdb** (see Example: Adding the "altova.mdb" Database to the Mapping 250).

Step 3: Draw the connections

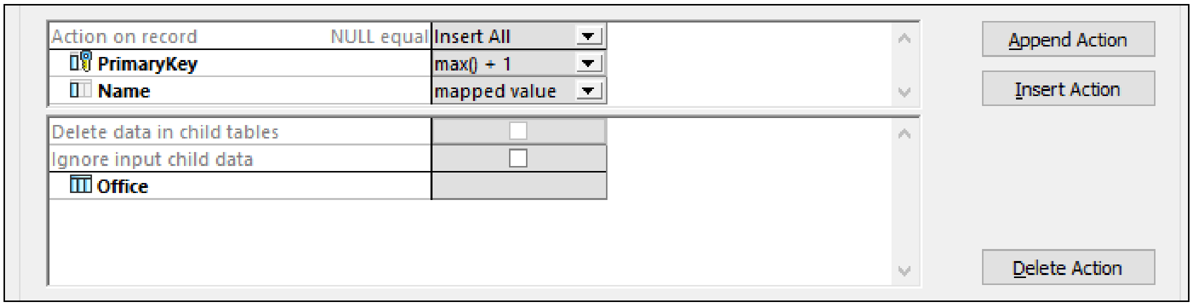
- Draw the mapping connections as shown below.



Note: If unwanted connections are automatically drawn for descending items, the option "Auto-connect children" is active. In this case, to undo the last action, select the menu option **Edit | Undo**. To disable the auto-connect option, select the menu option **Connection | Auto-connect matching children**.

Step 4: Configure the Insert action

1. On the target component, notice the **Action: Insert** (A:In) button. This button appears for each table that has a connection from the mapping (in this case, the "Altova" table). Click this button to configure in more detail the database action to be executed (in this case, the insert action). The Database Table Actions dialog box appears.
2. In the Database Table Actions dialog box, under **Insert All**, next to **PrimaryKey**, select the **max() + 1** option.



The options available in this list have the following meaning:

Option	Description
mapped value	Allows source data to be mapped to the database field directly, and is the standard setting for all database fields. It is also possible to use a stored procedure to supply a key value by defining a relation, see Using Stored Procedures to Generate Keys 410.
max() + 1	Generates the key values based on the existing keys in the database. For example, if the table has three records, with primary keys `1`, `2`, and `3`, then **max() + 1** is `4`. In this example, the "Altova" table has only one record with primary key `1`, so **max() + 1** is `2`, which is the expected value of the new primary key.
DB-generated	The database uses the **Identity function** to generate key values.

The option **mapped value** next to "Name" signifies that this column will get the value directly from the mapping. For reference to other options available on the Database Table Actions dialog box, see Database Table Actions Settings 326.

Step 5: Preview the mapping and update the database

Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.

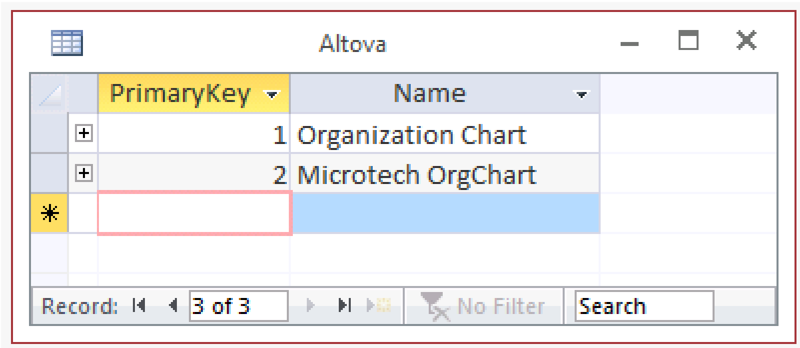
```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
Data Source=\\viepfs06\Documentation\Public\ExampleFiles\EN\MapForce\DB Insert\altova.mdb;Provider=Microsoft.Jet.OLEDB.4.0
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM [Altova]
-->>> %PrimaryKey1%

INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Microtech OrgChart', '%PrimaryKey1%')

```

To run the script against the database:

- On the **Output** menu, click **Run SQL-Script**.

Note: Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases 274.

To see the result, open the **altova.mdb** database in DatabaseSpy or Access. Notice that a new "Microtech OrgChart" record has been added to the "Altova" table with the new primary key 2. The data for this record originated in the input XML instance.



You have now finished creating a mapping which inserts data into a database table. For a mapping example which inserts data both into the current table and a dependent child table, see Inserting Data into Multiple Related Tables 286.

4.2.3.2 Inserting Data into Multiple Linked Tables

A database table may be a "parent" table; that is, it might be referred by other tables in the database through foreign key relationships. In such scenarios, you can configure the mapping to insert records not only into the parent table, but also into dependent child tables. For example, when inserting a new "company" record into a database table, you can also insert records for offices linked to this company, as well as their children departments, people, and so on.

This example shows you how to insert data into several tables while preserving the database relationships. It is a slightly more elaborate version of the previous example, Inserting Data into a Table 283. The example is accompanied by a sample mapping, and it uses the following files:

- **Altova_Hierarchical.mfd** — the actual mapping file.
- **Altova_Hierarchical.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **AltovaTarget.mdb** — the target database to be updated.

All files are available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. Below, the complete path to them will be omitted, for simplicity.

> The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases 274.

The goal of the mapping is to replace data in the target database (**AltovaTarget.mdb**) with data from a source XML file. The XML file structure roughly corresponds to the hierarchical structure of tables in the database. It is an organization chart, structured as follows: the top element is a company which contains two offices. Each office contains departments, and each department contains people. The same hierarchy exists in the **AltovaTarget.mdb**, where the "Altova" table corresponds to the company. This table is linked, through a foreign key relationship, to records in the "Office" table. Likewise, the "Office" links to "Department", and "Department" links to "Person". To view a relationship diagram of the **AltovaTarget.mdb** database, open it in the "Relationships" view of Access (see also Handling Database Relationships 257).

To achieve the mapping goal, we will take the steps below.

Step 1: Insert the source XML component

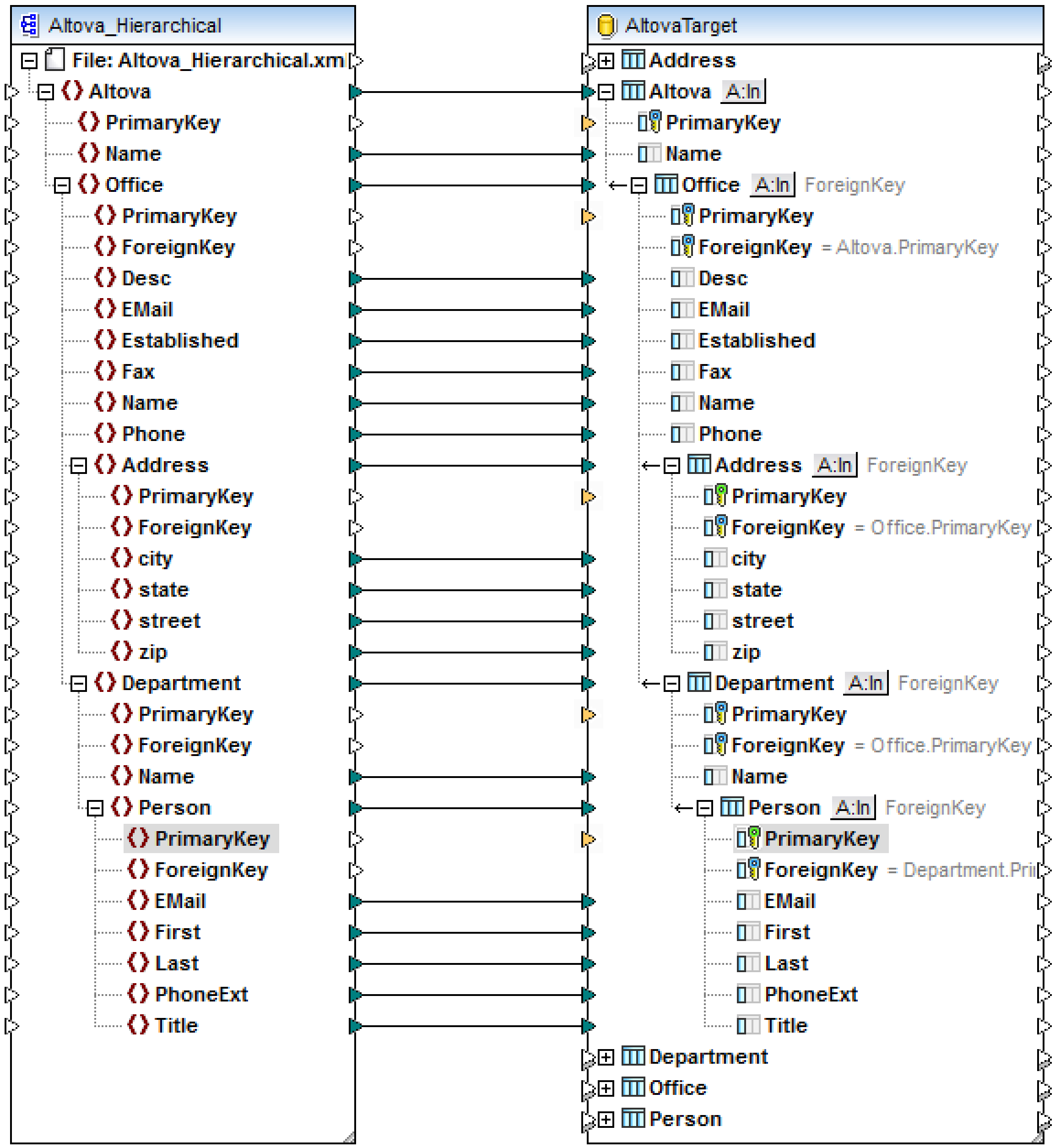
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **Altova_Hierarchical.xml**.

Step 2: Insert the target database

- On the **Insert** menu, click **Database**, and go through the wizard steps to connect to **AltovaTarget.mdb.** The instructions for connecting to this database are the same as for altova.mdb (see Example: Adding the "altova.mdb" Database to the Mapping 250).

Step 3: Draw the connections

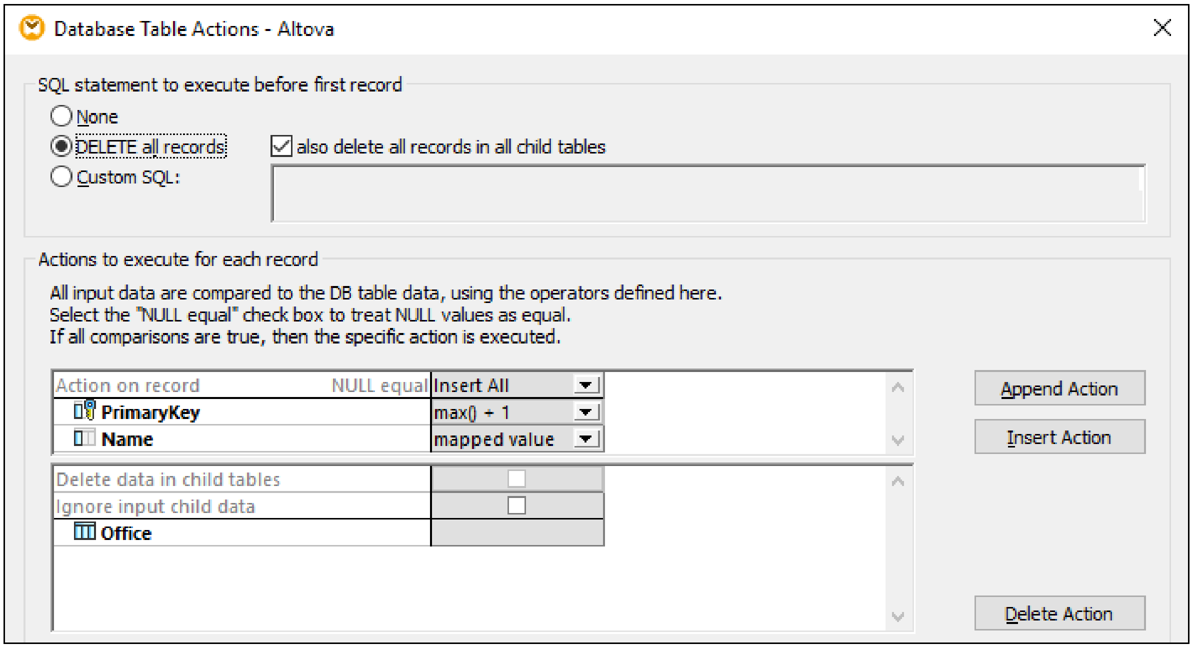
- Draw the mapping connections as shown below. Notice that the primary and foreign keys are not mapped; they will be generated on the fly, as shown below.



Note: If unwanted connections are automatically drawn for descending items, the option "Auto-connect children" is active. In this case, to undo the last action, select the menu option **Edit | Undo**. To disable the auto-connect option, select the menu option **Connection | Auto-connect matching children**.

Step 4: Configure the Insert actions

1. On the target component, click the **Action: Insert** (A:In) button next to the "Altova" table and configure the **max() + 1** setting of the primary key as shown below. This setting was explained in more detail in the previous example, see Inserting Data into a Table 283.

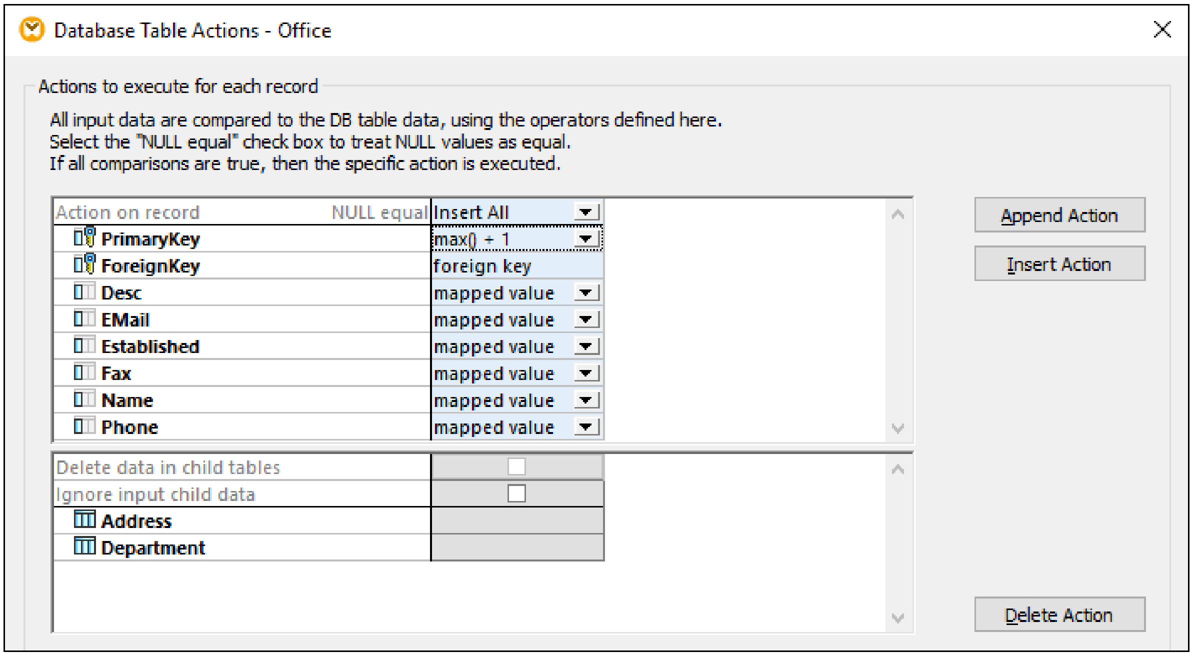


Also, notice that the **DELETE all records** option is enabled. This clears all existing records from the table, before new ones are entered, which is the desired behavior in this example. If you disable this option, new records (with a new primary key) will be added to the database in addition to existing ones, every time you run the mapping, which is not the desired behaviour.

For the scope of this example, the option **also delete all records from child tables** is also enabled. This ensures that not only records from the Altova table are deleted, but also all records in tables that are linked to the Altova table through a foreign key relationship. If the child tables have their own child tables, those will also be deleted, and so on, down to the last table in the dependency tree. If you attempted to delete only records from the root Altova table, this would violate the database integrity, and the mapping execution would fail.

For reference to other options available on the Database Table Actions dialog box, see Database Table Actions Settings 326.

2. Click **OK** to close the dialog box. Notice that, on the mapping area, the appearance of the button has now changed to DEL, A:In. This indicates that a "Delete" statement is configured to take place before the "Insert" action.
3. Click the A:In button next to the "Office" table and configure the **max() + 1** setting of the primary key.



4. Perform step 3 for each table descending from "Office", namely: "Address", "Department", and "Person". Make sure that all these tables are immediate descendants of the root "Altova" table. For an explanation of what is a "root" table and why it is necessary, see Handling Database Relationships [257].

Step 5: Preview the mapping and update the database

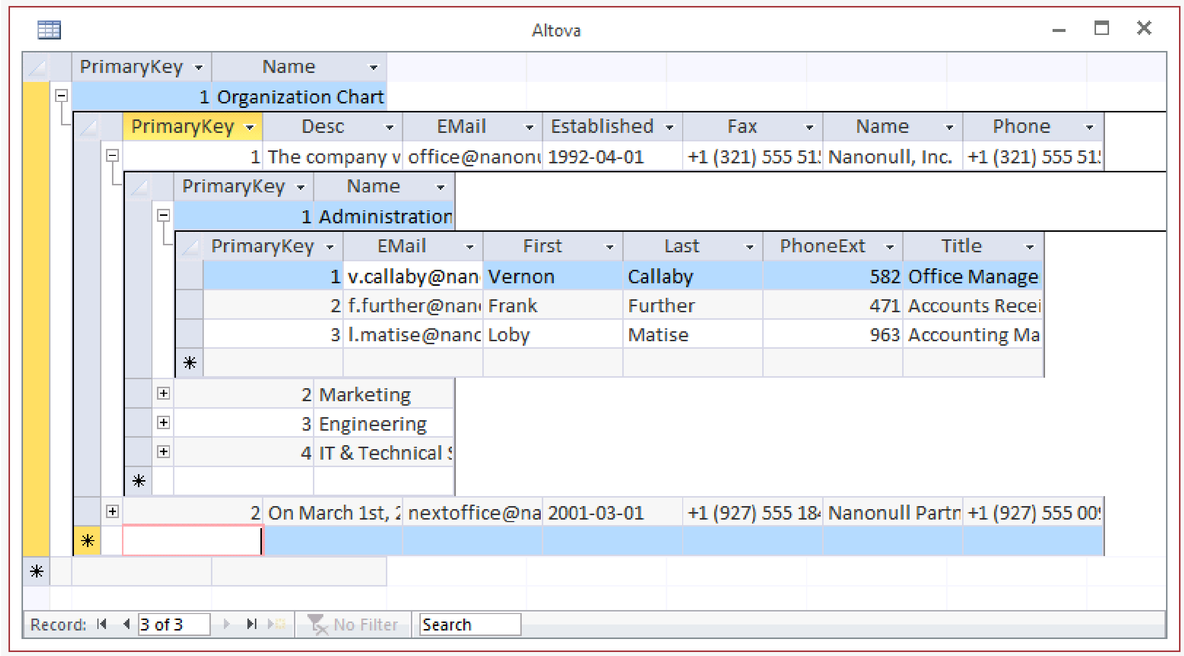
Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.

To run the script against the database:

- On the **Output** menu, click **Run SQL-Script**.

Note: Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases [274].

To see the result, open the "Altova" table in Microsoft Access, and observe how relationships from the XML file have now been propagated to the database, from the "Altova" table down to the "Person" table.



You have now finished creating a mapping which inserts data into multiple database tables, while preserving the table integrity relationships.

4.2.3.3 Updating a Table

This example shows you how to update data of an existing database table with data coming from an XML source. The example uses the following files:

- **altova-cmpy.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **altova.mdb** — the target database to be updated.

All files are available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. Below, the complete path to them will be omitted, for simplicity.

> The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases 274.

The goal of the mapping is to update all records in the "Person" table with instances of "Person" from the XML document. Each person in the XML file has a `PrimaryKey` child element. Each person in the "Person" table has a `PrimaryKey` column. Only those records where a person's `PrimaryKey` in the XML file corresponds to a person's `PrimaryKey` in the database must be updated.

To achieve the mapping goal, we will take the steps below.

Step 1: Insert the source XML component

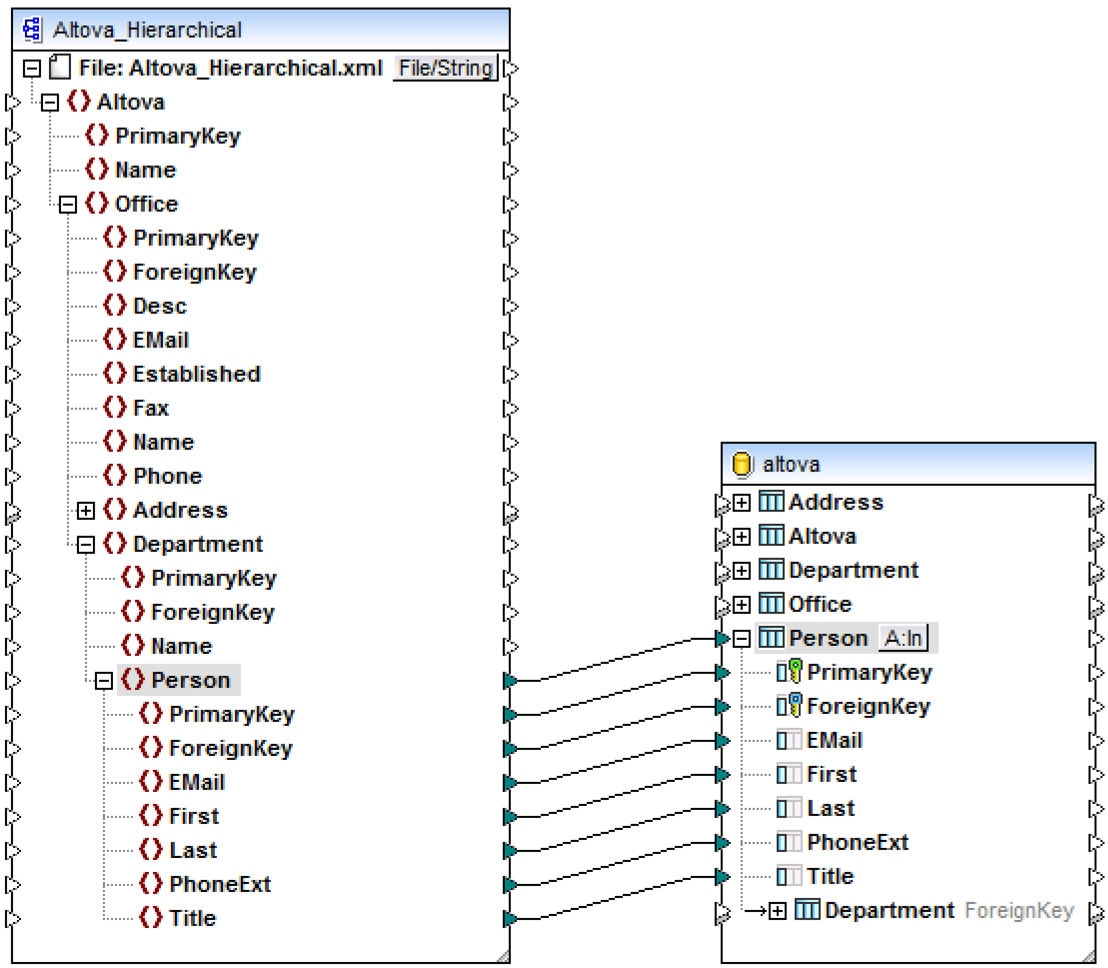
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **altova-cmpy.xml**.

Step 2: Insert the target database

- On the **Insert** menu, click **Database**, and go through the wizard steps to connect to **altova.mdb** (see Example: Adding the "altova.mdb" Database to the Mapping 250).

Step 3: Draw the connections

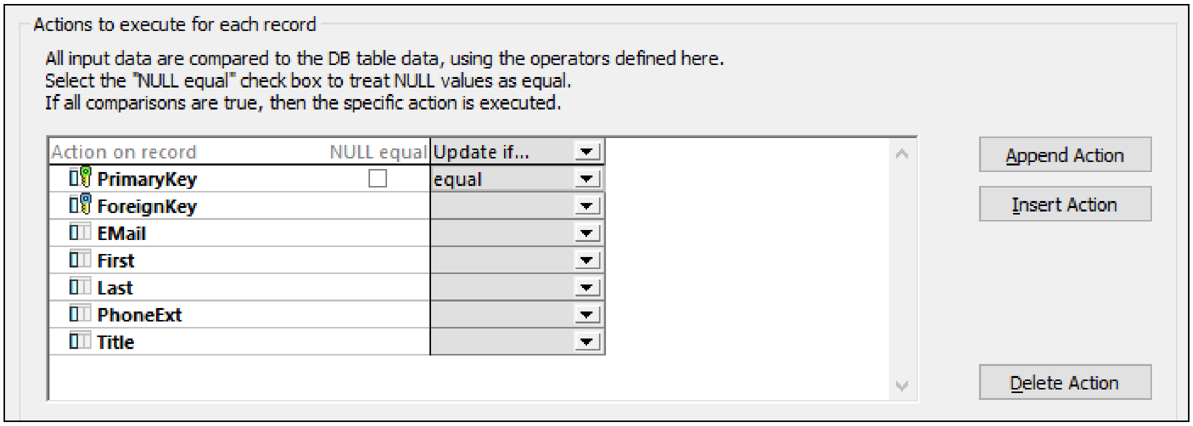
- Draw the mapping connections as shown below.



Step 4: Configure the Update action

1. On the target component, click the **Action: Insert** (A:In) button next to the "Person" table.

2. Next to **Action on record**, select **Update if...** . This changes the database table action to a conditional update action. That is, the current record will only be updated when a condition is satisfied (see next step).
3. Next to **PrimaryKey**, select the value **equal**, as shown below. This defines the update condition: that is, the database record will be updated only when its **PrimaryKey** value is equal to the **PrimaryKey** value coming from the mapping.



In this example, the equality operator is applied to the **PrimaryKey** field, which is a likely scenario when updating databases. Note that conditions can also be defined on other fields which are not necessarily primary keys. For example, by selecting **equal** next to the **First** and **Last** fields, you would update only those records where both the first and last name is equal to that in the source XML.

Optionally, select the **NULL equal** check box. In this example, this check box is irrelevant, because neither the **PrimaryKey** field in the source XML nor the **PrimaryKey** field in the database contains any null values. However, you should select this check box if your data contains null values, and if you want to treat such null values as equal; otherwise, you may get undesired results. For more information, see Handling Nulls in Database Table Actions 323.

4. Click **OK** to close the dialog box. Notice that, back on the mapping, the **Action: Insert** A:In button has now changed to an **Action: Update** (A:Up) button. This indicates that an update action is configured to take place for this table.

Step 5: Preview the mapping and update the database

Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.

```
/*
The following SQL statements were executed during "Generate output" function.
Every single result is written right to the "-->>>" string.
These statements are only for preview and may not be executed in another SQL query tool!
The database was connected using the following connection-string:
Data Source=\\viepfs06\Documentation\Public\ExampleFiles\EN\MapForce\DB Update
if\altova.mdb;Provider=Microsoft.Jet.OLEDB.4.0
*/

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'A.Aldrich@microtech.com', [First] = 'Albert', [Last] = 'Aldrich',
[PhoneExt] = 582, [Title] = 'Manager' WHERE ([Person].[PrimaryKey] = 1)
-->>> OK. 1 row(s).

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'b.bander@microtech.com', [First] = 'Bert', [Last] = 'Bander',
[PhoneExt] = 471, [Title] = 'Accounts Receivable' WHERE ([Person].[PrimaryKey] = 2)
-->>> OK. 1 row(s).

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'c.clovis@microtech.com', [First] = 'Clive', [Last] = 'Clovis',
[PhoneExt] = 963, [Title] = 'Accounting Manager' WHERE ([Person].[PrimaryKey] = 3)
-->>> OK. 1 row(s).

UPDATE [Person] SET [ForeignKey] = 2, [EMail] = 'd.Durnell@microtech.com', [First] = 'Dave', [Last] = 'Durnell',
[PhoneExt] = 621, [Title] = 'Marketing Manager Europe' WHERE ([Person].[PrimaryKey] = 4)
-->>> OK. 1 row(s).
```

To run the script against the database:

- On the **Output** menu, click **Run SQL-Script**.

Note: Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases 274.

4.2.3.4 "Update if... Insert Rest" Action

Sometimes, it is necessary not only to update existing records, but also to insert new records into the same database table. For such cases, MapForce provides an "Update if... Insert Rest" action. This works as follows:

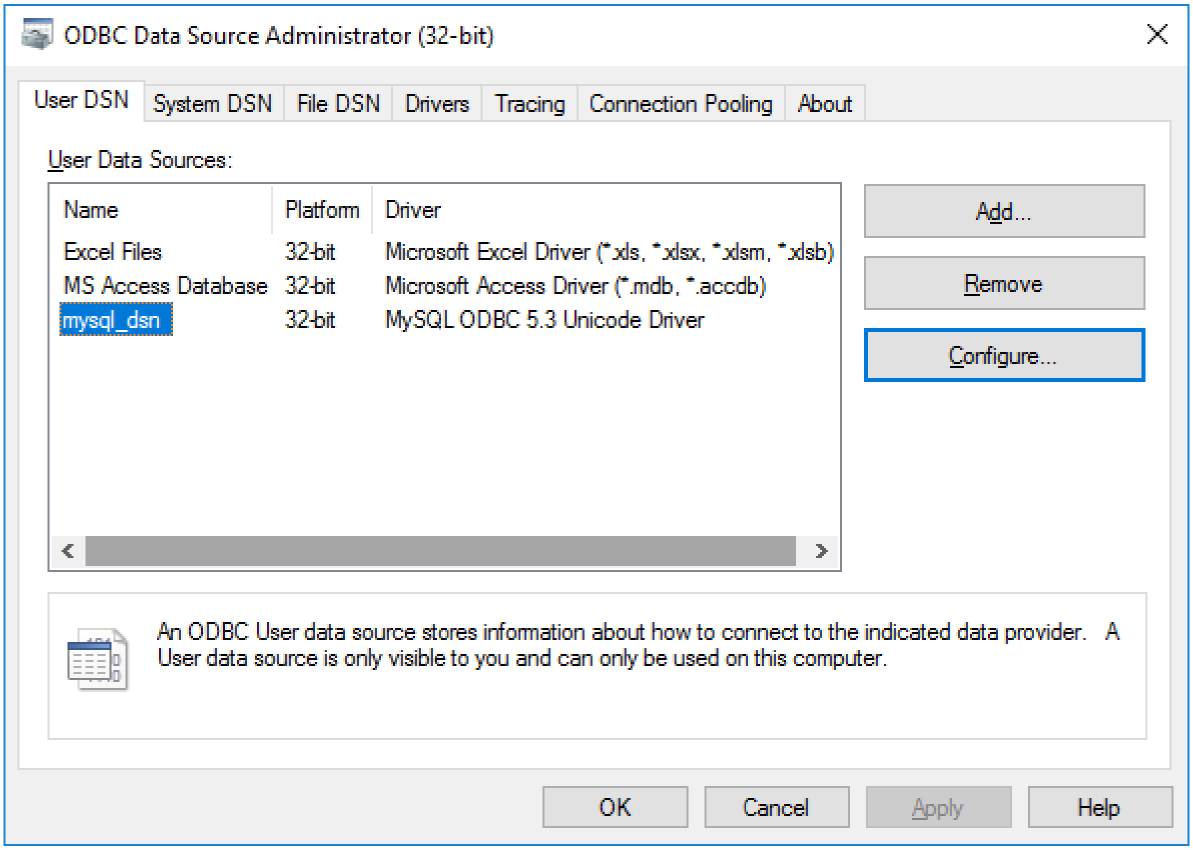
- If the **Update if** condition is true, then the existing database record is updated with data from the mapping.
- If the **Update if** condition is false, and an **Insert Rest** condition exists, then a new record is inserted.
- If records exist in the database with no counterpart in the source file, then these records remain unchanged.

MySQL / MariaDB ODBC note

If the target database is MySQL or MariaDB through ODBC, the option **Return matched rows instead of affected rows** must be enabled in the **Cursor/Results** tab of MySQL ODBC Connector. Alternatively, if you enter the connection string manually through the Database Connection wizard, add `Option=2` to the connection string , for example: `Dsn=mydsn;Option=2;`

To enable this option from the MySQL ODBC Connector:

1. Press the **Windows** key and start typing "ODBC".
2. Run the ODBC Data Sources Administrator (either 32-bit or 64-bit, depending on the platform of the installed MySQL ODBC Connector).
3. Click the Data Source Name (DSN) used by the MapForce mapping, and then click **Configure**.



4. Click **Details >>** to make the advanced options available.
5. Click the **Cursors/Results** tab, and then select the check box **Return matched rows instead of affected rows**.

Example

The following example shows you how to merge (both update and insert) data from an XML source into a database table. The example uses the following files:

- **altova-cmpy-extra.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **altova.mdb** — the target database to be updated.

All files are available in the folder **<Documents>\Altova\MapForce2023\MapForceExamples**. Below, the complete path to them will be omitted, for simplicity.

> The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see

Executing Mappings Which Modify Databases 274.

The goal of the mapping is to merge all records from a source XML document into a target "Person" table. Namely, for each record in the source XML data, the mapping must do the following:

- If the person's `PrimaryKey` in the XML file corresponds to a person's `PrimaryKey` in the database, then update the record.
- Any existing records in the Person table which do not meet the above condition must not be affected.
- If the person's `PrimaryKey` in the XML file does not have a match in the target database table, then add a new record to the database table.

To achieve the mapping goal, we will take the steps below.

Step 1: Insert the source XML component

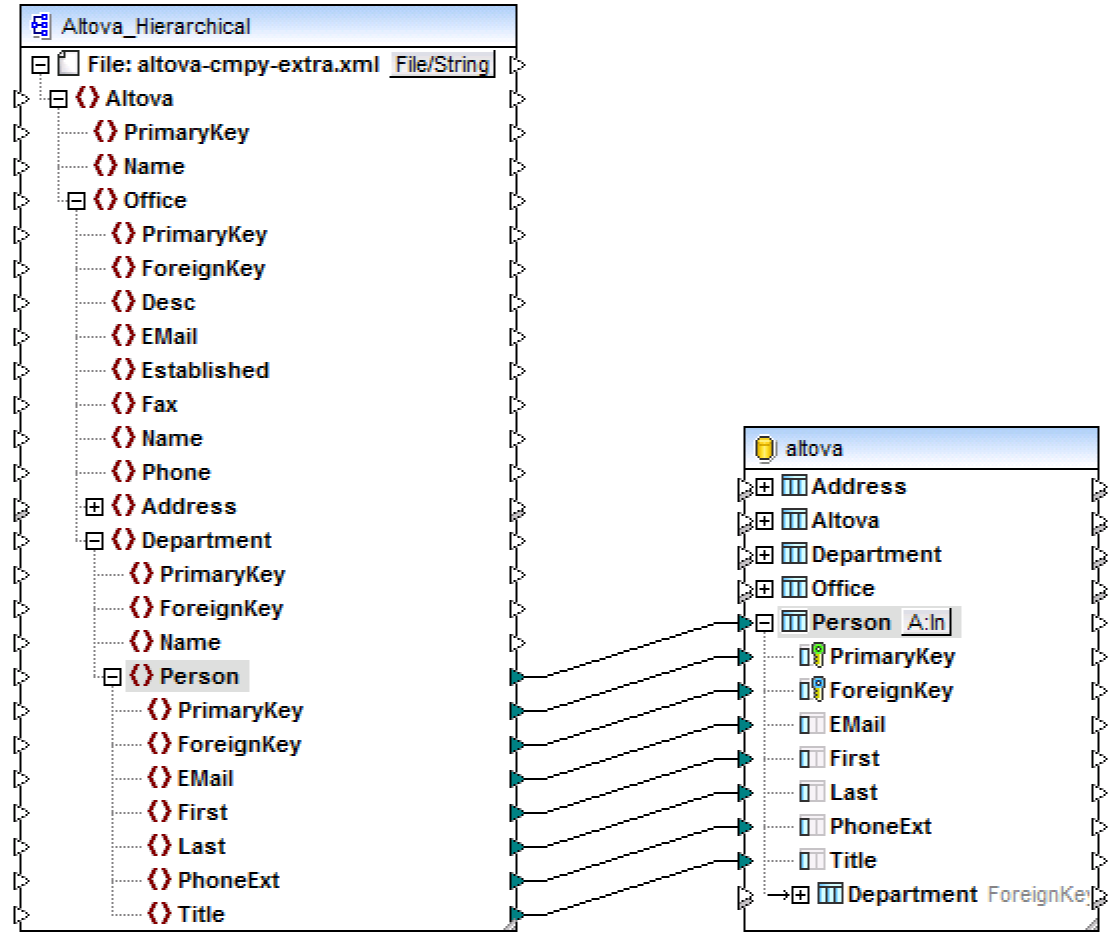
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **altova-cmpy-extra.xml**.

Step 2: Insert the target database

- On the **Insert** menu, click **Database**, and go through the wizard steps to connect to **altova.mdb** (see Example: Adding the "altova.mdb" Database to the Mapping 250).

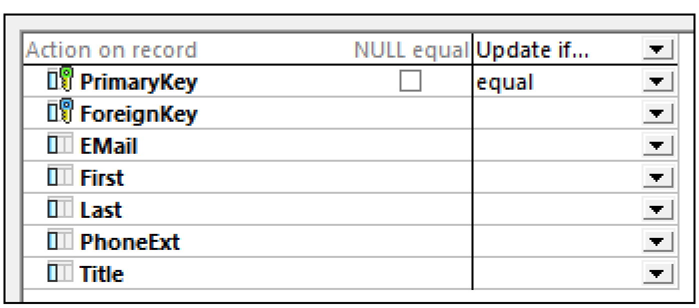
Step 3: Draw the connections

- Draw the mapping connections as shown below.

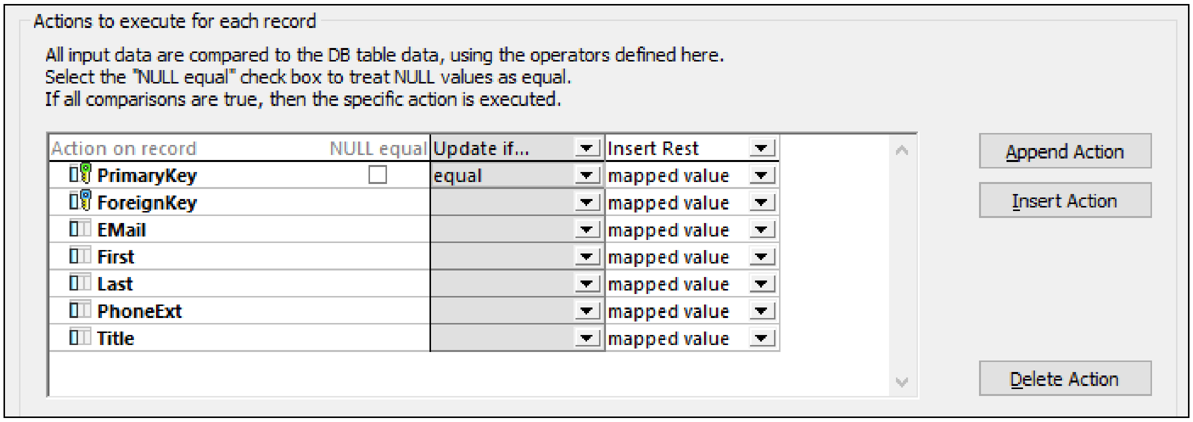


Step 4: Configure the "Update if... Insert Rest" actions

1. On the target component, click the **Action: Insert** (A:In) button next to the "Person" table.
2. Next to **Action on record**, select **Update if...** . This changes the database table action to a conditional update action. That is, the current record will only be updated when a condition is satisfied (see next step).
3. Next to **PrimaryKey**, select the value **equal**, as shown below. This defines the update condition: that is, the database record will be updated only when its **PrimaryKey** value is equal to the **PrimaryKey** value coming from the mapping.



4. Optionally, select the **NULL equal** check box. In this example, this check box is irrelevant, because neither the **PrimaryKey** field in the source XML nor the **PrimaryKey** field in the database contains any null values. However, you should select this check box if your data contains null values, and if you want to treat such null values as equal; otherwise, you may get undesired results. For more information, see Handling Nulls in Database Table Actions 323.
5. Click **Append Action**. This adds a new action to the right of the existing **Update If** action. Configure the new action as **Insert Rest**:



In the image above, the database table actions have been configured in accordance with the goals of the mapping. That is, only when the **Update If**... condition is satisfied will the record be updated; otherwise, it will be inserted. The option "mapped value" specifies that values from the mapping will be used to populate all fields of the record.

It is also possible to define more than two actions against the same database table (this is not necessary in this example, however). At mapping runtime, actions are executed from left to right. The last **Insert** action is considered final; any other actions added after it will be ignored.

Note that the **Append Action** button on the dialog box adds the new action *after* the selected one. **Insert Action** adds the new action *before* the selected one. To delete an existing action, click anywhere inside it, and then click **Delete Action**.

6. Click **OK** to close the dialog box. Notice that, back on the mapping, the **Action: Insert** A:In button has now changed to an **Action: Update; Insert** (A:Up,In) button. This indicates that both an update and an insert action is configured to take place for this table.

Step 5: Preview the mapping and update the database

Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.


```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
Data Source=\\viepfs06\Documentation\Public\ExampleFiles\EN\MapForce\DB Update if... Insert Rest\altova.mdb;Provider=Microsoft.Jet.OLEDB.4.0
*/

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'A.Aldrich@microtech.com', [First] = 'Albert', [Last] = 'Aldrich', [PhoneExt] = 582, [Title] = 'Manager' WHERE ([Person].[PrimaryKey] = 1)

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'b.bander@microtech.com', [First] = 'Bert', [Last] = 'Bander', [PhoneExt] = 471, [Title] = 'Accounts Receivable' WHERE ([Person].[PrimaryKey] = 2)

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'c.clovis@microtech.com', [First] = 'Clive', [Last] = 'Clovis', [PhoneExt] = 963, [Title] = 'Accounting Manager' WHERE ([Person].[PrimaryKey] = 3)

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'c.Cicada@microtech.com', [First] = 'Camilla ', [Last] = 'Cicada', [PhoneExt] = 765, [Title] = 'HR' WHERE ([Person].[PrimaryKey] = 30)
```

SQL script (partial view) before updating the database

You may notice that no INSERT statements are visible in the preview script. This is normal behavior, because records are inserted conditionally, and the INSERT statements depend on the result of the **Update If** action (which is not known before the mapping runs).

Note: For certain database types, MapForce creates MERGE statements instead of UPDATE statements. For further information, see MERGE statements 301.

To run the script against the database:

- On the **Output** menu, click **Run SQL-Script**.

Now that the mapping has been executed and the script applied against the database, notice that INSERT statements are visible in the **Output** tab.

```
/*
The following SQL statements were executed during "Generate output" function.
Every single result is written right to the "-->>>" string.
These statements are only for preview and may not be executed in another SQL query tool!
The database was connected using the following connection-string:
Data Source=\\viepfs06\Documentation\Public\ExampleFiles\EN\MapForce\DB Update if... Insert
Rest\altova.mdb;Provider=Microsoft.Jet.OLEDB.4.0
*/

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'A.Aldrich@microtech.com', [First] = 'Albert', [Last] = 'Aldrich',
[PhoneExt] = 582, [Title] = 'Manager' WHERE ([Person].[PrimaryKey] = 1)
-->>> OK. 1 row(s).

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'b.bander@microtech.com', [First] = 'Bert', [Last] = 'Bander', [PhoneExt]
 = 471, [Title] = 'Accounts Receivable' WHERE ([Person].[PrimaryKey] = 2)
-->>> OK. 1 row(s).

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'c.clovis@microtech.com', [First] = 'Clive', [Last] = 'Clovis', [PhoneExt] =
 963, [Title] = 'Accounting Manager' WHERE ([Person].[PrimaryKey] = 3)
-->>> OK. 1 row(s).

UPDATE [Person] SET [ForeignKey] = 1, [EMail] = 'c.Cicada@microtech.com', [First] = 'Camilla ', [Last] = 'Cicada',
[PhoneExt] = 765, [Title] = 'HR' WHERE ([Person].[PrimaryKey] = 30)
-->>> OK. 0 row(s).

INSERT INTO [Person] ([PrimaryKey], [ForeignKey], [EMail], [First], [Last], [PhoneExt], [Title]) VALUES (30, 1,
'c.Cicada@microtech.com', 'Camilla ', 'Cicada', 765, 'HR')
-->>> OK. 1 row(s).
```

SQL script (partial view) after updating the database

Note: Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases [274].

If you open the "Person" table in the DB query tab of MapForce (see Browsing and Querying Databases [376]), you can see the result of the mapping as follows:

- All database records which had corresponding primary keys in the XML file have been updated. Examples are records with primary key 1, 2, 3, 4, and 5.
- All database records which had no corresponding keys in the XML file remained unaffected. Examples are records with primary key 6, 7, 8, and 9.
- New records have been inserted to the "Person" table (where key did not already exist in the database). Examples are records with primary key 30 and 31.

	PrimaryKey	ForeignKey	EMail	First	Last	PhoneExt	Title
1	1	1	A.Aldrich@microtech.com	Albert	Aldrich	582	Manager
2	2	1	b.bander@microtech.com	Bert	Bander	471	Accounts Receivable
3	3	1	c.clovis@microtech.com	Clive	Clovis	963	Accounting Manager
4	4	2	d.Durnell@microtech.com	Dave	Durnelll	621	Marketing Manager Europe
5	5	2	e.ellas@microtech.com	Eve	Ellas	753	Art Director
6	6	3	f.landis@nanonull.com	Fred	Landis	951	Program Manager
7	7	3	m.landis@nanonull.com	Michelle	Butler	654	Software Engineer
8	8	3	t.little@nanonull.com	Ted	Little	852	Software Engineer
9	9	3	a.way@nanonull.com	Ann	Way	951	Technical Writer
10	10	3	l.gardner@nanonull.com	Liz	Gardner	753	Software Engineer
11	11	3	p.smith@nanonull.com	Paul	Smith	334	Software Engineer
12	12	4	a.martin@nanonull.com	Alex	Martin	778	IT Manager
13	13	4	g.hammer@nanonull.com	George	Hammer	223	Web Developer
14	14	4	n.newbury@microtech.com	Nira	Newbury	241	Support Engineer
15	15	4	o.origone@microtech.com	Olanda	Origone	345	Support Engineer
16	16	5	s.meier@microtech.com	Paul	Ponetti	114	Office Manager
17	17	5	t.bone@microtech.com	Quang	Qui	331	Accounts Receivable
18	18	6	m.nafta@microtech.com	Rita	Rotter	122	PR & Marketing Manager US
19	19	7	v.bass@microtech.com	Silla	Semua	716	IT Manager
20	20	7	c.franken@microtech.com	Tom	Tinker	147	Support Engineer
21	21	7	m.redgreen@microtech.com	Uri	Ugoner	152	Support Engineer
22	30	1	c.Cicada@microtech.com	Camilla	Cicada	765	HR
23	31	1	c.corrigan@microtech.com	Carol	Corrigan	629	Admin

The "Person" table after updating the database

4.2.3.5 MERGE Statements

For certain mappings which both update and insert data into a database table (see also "Update if... Insert Rest" Action 294), MapForce generates MERGE statements to be executed against the database at mapping runtime. The execution engine may not necessarily be MapForce, see Executing Mappings Which Modify Databases 274.

MERGE statements are supported for the following database types:

- SQL Server 2008 and later
- Oracle
- DB2
- Firebird

MERGE statements reduce the number of database server calls, since they combine the INSERT and UPDATE statements into one. Also, in case of MERGE statements, the consistency check is done by the database.

To see whether the mapping will execute database MERGE statements at runtime (as opposed to applying a combination of INSERT and UPDATE statements):

1. Create a mapping which uses an **Update if...** as well as an **Insert Rest** action. For an example, see "Update if... Insert Rest" Action[294].
2. Preview the mapping, by clicking the **Output** tab.

If MERGE is supported by the database type, the generated SQL script includes MERGE statements, for example:

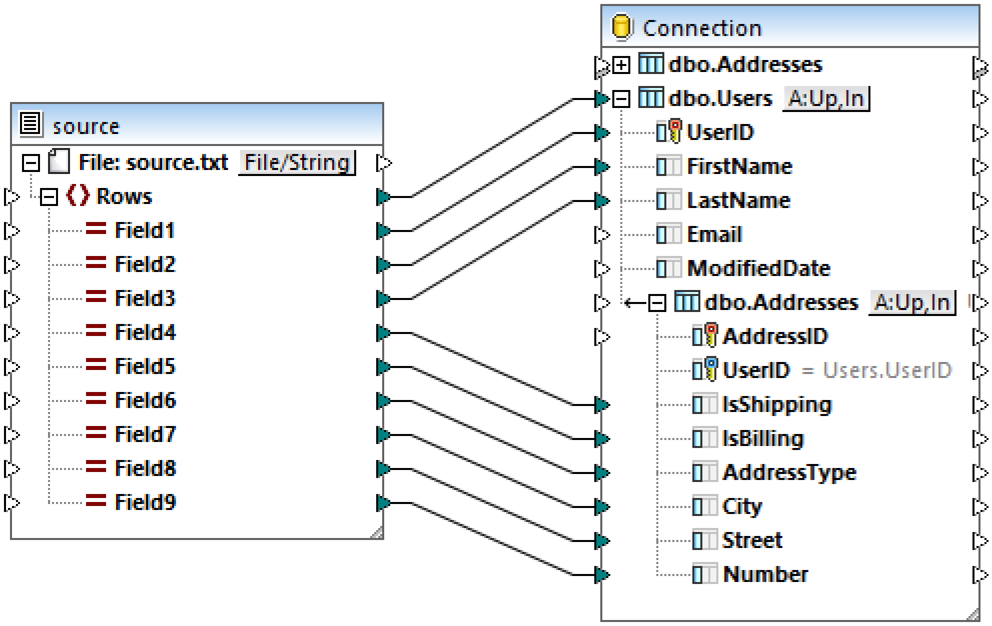
```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
Data Source=                    ;Initial
Catalog=DB_MERGE;MultipleActiveResultSets=True;Password=•••••;Persist Security Info=True;User
ID=
*/

SET QUOTED_IDENTIFIER ON

UPDATE [dbo].[Users] SET [FirstName] = (CAST('Despine' AS nvarchar(50))), [LastName] =
(CAST('Buttler' AS nvarchar(50))) WHERE ([dbo].[Users].[UserID] = 1)

SELECT [UserID] FROM [dbo].[Users] WHERE ([UserID] = 1)
-->>> %UserID1%

DELETE FROM [dbo].[Addresses] WHERE EXISTS(SELECT * FROM [dbo].[Users] WHERE [dbo].[Users].[UserID] =
[dbo].[Addresses].[UserID] AND ([dbo].[Users].[UserID] = '%UserID1%'))

MERGE INTO [dbo].[Addresses] AS T USING ( VALUES ( '%UserID1%', 1, 1, (CAST('Home' AS nvarchar(20))),
(CAST('Louisville' AS nvarchar(50))), (CAST('Elm Street' AS nvarchar(50))), (CAST('12' AS
nvarchar(20))) ) ) AS S ( [UserID], [IsShipping], [IsBilling], [AddressType], [City], [Street],
[Number] ) ON ( (S.[UserID] = T.[UserID]) ) WHEN MATCHED THEN UPDATE SET [IsShipping] = S.
[IsShipping], [IsBilling] = S.[IsBilling], [AddressType] = S.[AddressType], [City] = S.[City],
[Street] = S.[Street], [Number] = S.[Number] WHEN NOT MATCHED THEN INSERT ( [UserID], [IsShipping],
[IsBilling], [AddressType], [City], [Street], [Number] ) VALUES ( S.[UserID], S.[IsShipping], S.
[IsBilling], S.[AddressType], S.[City], S.[Street], S.[Number] );
```

Mapping | DB Query | **Output**

If MERGE is not supported by the database type, the generated SQL script includes UPDATE statements only. No INSERT statements are visible for preview, since those are to be executed only if the **Update If...** condition is not satisfied (and this is not known before the mapping execution).

Notes

- MapForce creates MERGE statements automatically when it detects a supported database type; it is not possible to manually influence whether MapForce should create a MERGE statement.
- If you are updating multiple tables having parent-child relationships, merges are created only for "leaf" tables. A "leaf" table is the deepest child table that is mapped. For example, in the mapping below, **Update If... Insert Rest...** actions have been defined for both `Users` table and `Addresses` table. However, MERGE statements will be generated only for the leaf table, `Addresses`. The parent table, `Users`, gets UPDATE or INSERT statements instead of MERGE.



- With MERGE statements, the "Bulk Transfer" option (see Bulk Inserts (MapForce Server) [319] is supported only for ODBC and JDBC database connections.

4.2.3.6 Options for Child Tables When Updating a Parent Table

When the mapping updates a table which is a "parent" table (that is, it has foreign key relationships to other tables), you can configure how the dependent records should be treated both in the source data and in the target table. For example, let's assume that you want to update the "Department" table in the **altova.mdb** database. Because every person is linked to a department by means of a foreign key, you will likely want to take action against the "Person" table as well (which could be an insert, update, or delete). Doing so would help you maintain the database integrity and avoid mapping errors.

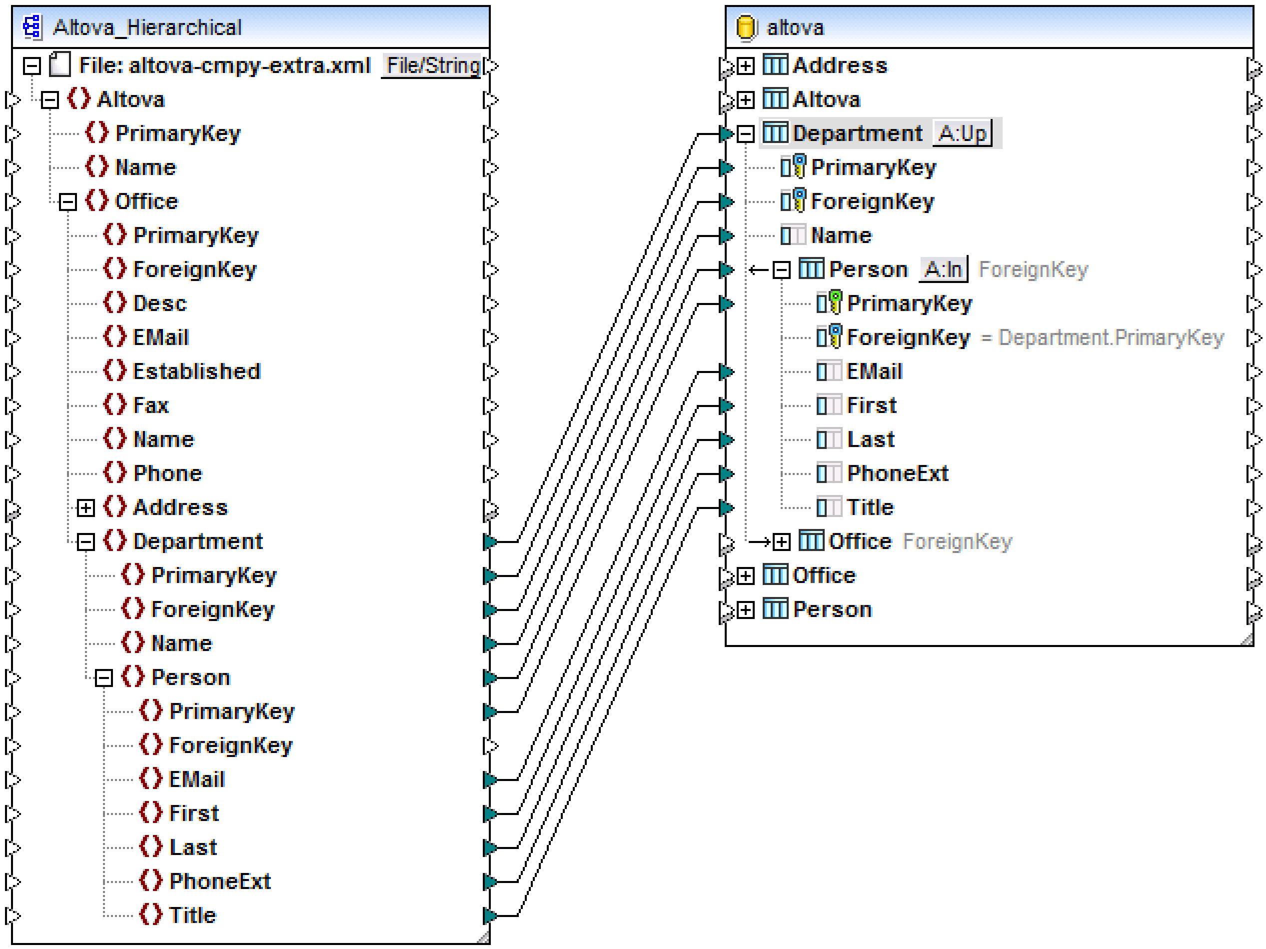
This topic discusses the options available for the "Person" table when you update the parent "Department" table. It makes use of the following example files:

- **altova-cmpy-extra.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **altova.mdb** — the target database to be updated.

All files are available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. Below, the complete path to them will be omitted, for simplicity.

> The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases [274].

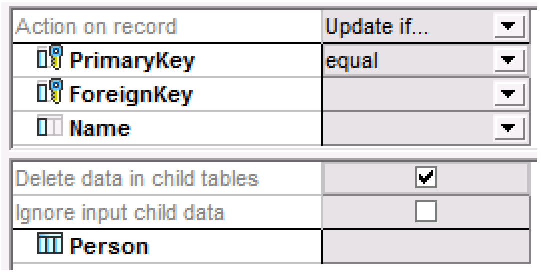
First, add the source XML schema and instance as well as the target database to the mapping (see Example: Adding the "altova.mdb" Database to the Mapping 250). Follow the same steps as in "Update if... Insert Rest" Action 294. Secondly, draw the mapping connections as shown below:



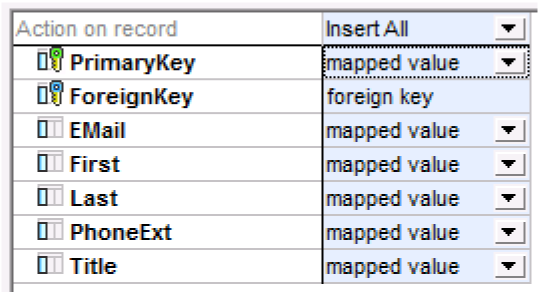
As illustrated above, the mapping updates the "Department" table in the target database. The "Department" table is chosen as "root" table. For more information about what a root table is and why it is necessary, see Handling Database Relationships 257. The action to be taken against the child "Person" table is the subject of this topic.

The following tables illustrate various configuration options and the corresponding mapping result. These options can be selected from the Database Table Actions dialog box of the parent "Department" table and the child "Person" table.

Configuration A

<table>
<tr><th>Settings</th><th>Mapping result</th></tr>
<tr><td>


"Department" table



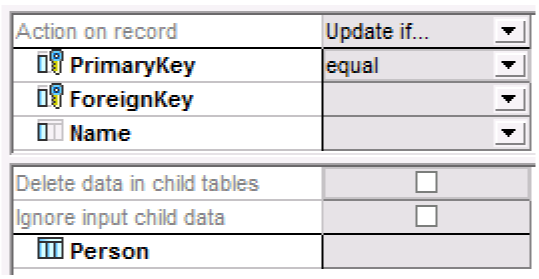
"Person" table
</td><td>

- Updates Department records where `PrimaryKey` in the source XML corresponds to the `PrimaryKey` in the database table.
- Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).
- Inserts, from the input XML instance, all Person records that do not already exist in the database.
- Deletes child data (Person records) of those Department records which satisfy the **Update if...** condition.

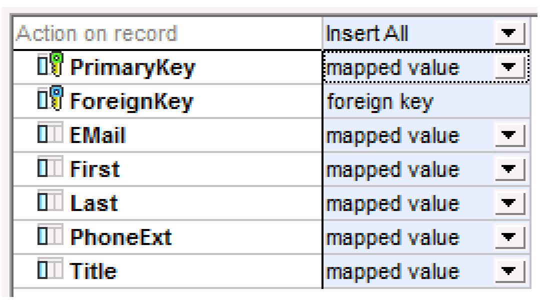
</td></tr>
</table>

Configuration B

<table>
<tr><th>Settings</th><th>Mapping result</th></tr>
<tr><td>



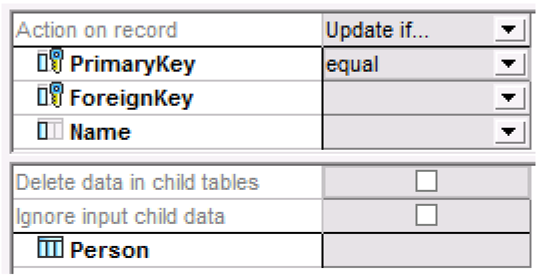
"Department" table

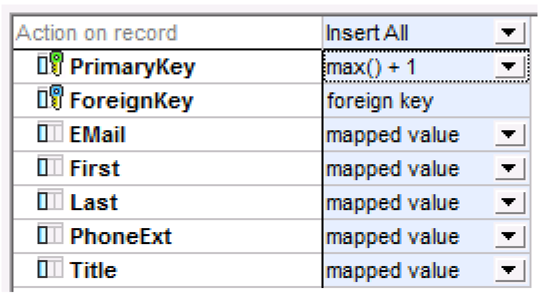


"Person" table

</td><td>The mapping fails with an SQL execution error. The reason is that the mapping attempts to insert new Person records with the same primary key as the existing Person records. If you want to insert records from the input XML in addition to those already in the database, see the next option.</td></tr>
</table>

Configuration C

<table>
<tr><th>Settings</th><th>Mapping result</th></tr>
<tr><td>


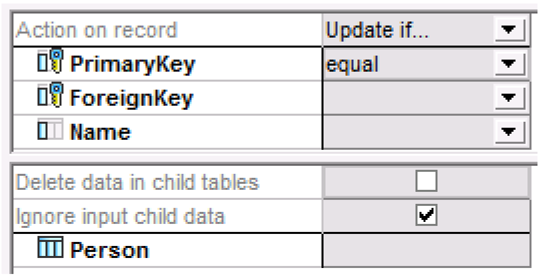
"Department" table


"Person" table
</td><td>

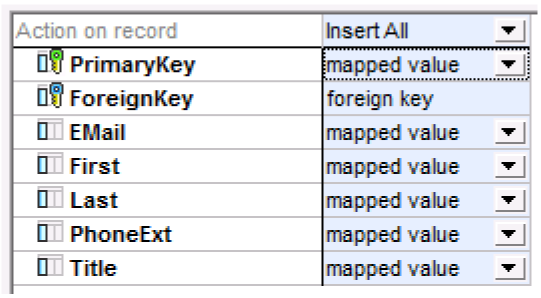
Updates Department records where <code>PrimaryKey</code> in the source XML corresponds to the <code>PrimaryKey</code> in the database table.
Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).
New Person records (with generated primary keys) are inserted into the Person table in addition to existing ones.

</td></tr>
</table>

Configuration D

<table>
<tr><th>Settings</th><th>Mapping result</th></tr>
<tr><td>


"Department" table

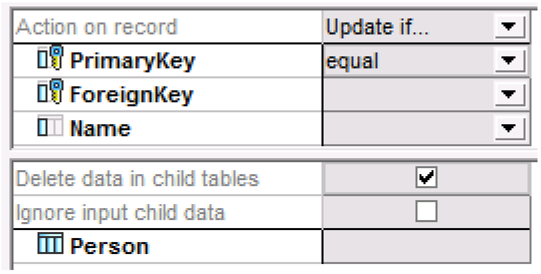
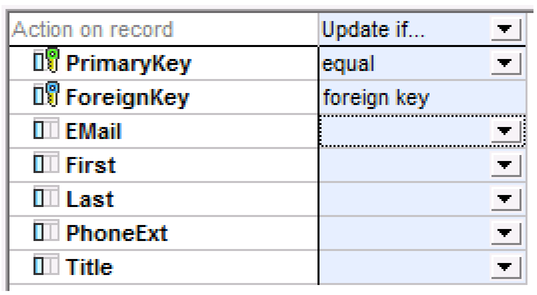


"Person" table
</td><td>

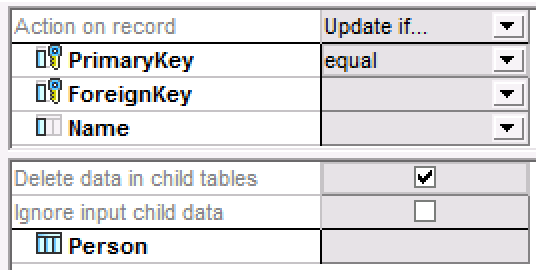
- Updates Department records where `PrimaryKey` in the source XML corresponds to the `PrimaryKey` in the database table.
- Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).
- No records are inserted in the Person table because the option **Ignore input child data** is enabled for the parent Departments table.

</td></tr>
</table>

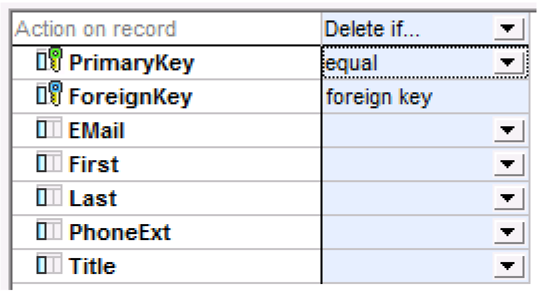
Configuration E

Settings	**Mapping result**
 *"Department" table*  *"Person" table*	• Updates Department records where `PrimaryKey` in the source XML corresponds to the `PrimaryKey` in the database table. • Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source). • Deletes all Person records linked to a Department which has a corresponding `PrimaryKey` in the source XML. The reason is that the **Delete data in child tables** option is enabled for the parent Department table. • Person records linked to a department that did not meet the **Update if...** condition remain in the database. • No records in the Person table are updated.

Configuration F

<table>
<tr><th>Settings</th><th>Mapping result</th></tr>
<tr><td>


"Department" table

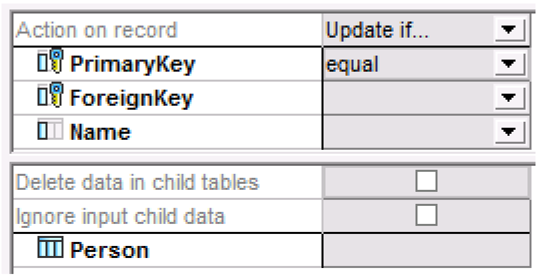


"Person" table
</td><td>

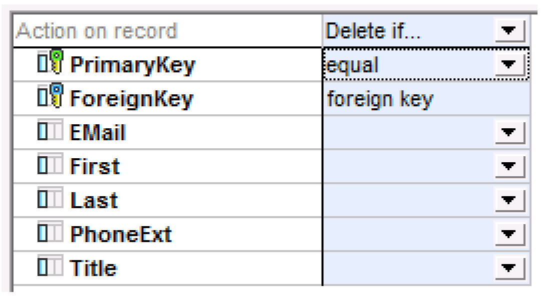
Updates Department records where <code>PrimaryKey</code> in the source XML corresponds to the <code>PrimaryKey</code> in the database table.
Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).
Deletes all Person records linked to a Department which has a corresponding <code>PrimaryKey</code> in the source XML. The reason is that the Delete data in child tables option is enabled for the parent Department table.
Person records linked to a department that did not meet the Update if... condition remain in the database.

</td></tr>
</table>

Configuration G

<table>
<tr><th>Settings</th><th>Mapping result</th></tr>
<tr><td>


"Department" table



"Person" table
</td><td>

- Updates Department records where `PrimaryKey` in the source XML corresponds to the `PrimaryKey` in the database table.
- Does not update existing Department records which do not have a counterpart in the input XML file (no such key exists in the source).
- Deletes all Person records which satisfy both of the following conditions:

  a. The Person record is linked to a Department which has a corresponding `PrimaryKey` in the source XML, and
  b. The Person record has a corresponding `PrimaryKey` in the source XML.
</td></tr>
</table>

4.2.3.7 "Delete if..." Action

The table action **Delete if..**. is used to delete data from a database table conditionally. You can define this action from the Database Table Actions dialog box (see Database Table Actions Settings 326). For example, when mapping data from a source XML to a target database, you can configure a **Delete if...** condition to check whether a certain field in the source XML is equal to a field in the target database record (typically, a primary key value). If the **Delete if...** condition is true (that is, the two fields are equal), the database record will be deleted when the mapping runs.

Note: The **Delete if...** table action should not be confused with the **Delete data in child tables** option available in the Database Table Actions dialog box. The **Delete if...** table action only affects the table for which the action is defined; no other tables are affected.

This example shows you how to delete data from a database table conditionally, and also insert records into the same database table based on the outcome of the delete condition.

This example uses the following files:

- **altova-cmpy-extra.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **altova.mdb** — the target database to be updated.

All files are available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. Below, the complete path to them will be omitted, for simplicity.

> The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases 274.

The goal of the mapping is as follows:

- If any person records with the same `PrimaryKey` exist both in the source XML and the target Person table, they must be deleted from the Person table.
- All other records from the source XML must be inserted into the Person table.

To achieve the mapping goal, we will take the steps below.

Step 1: Insert the source XML component

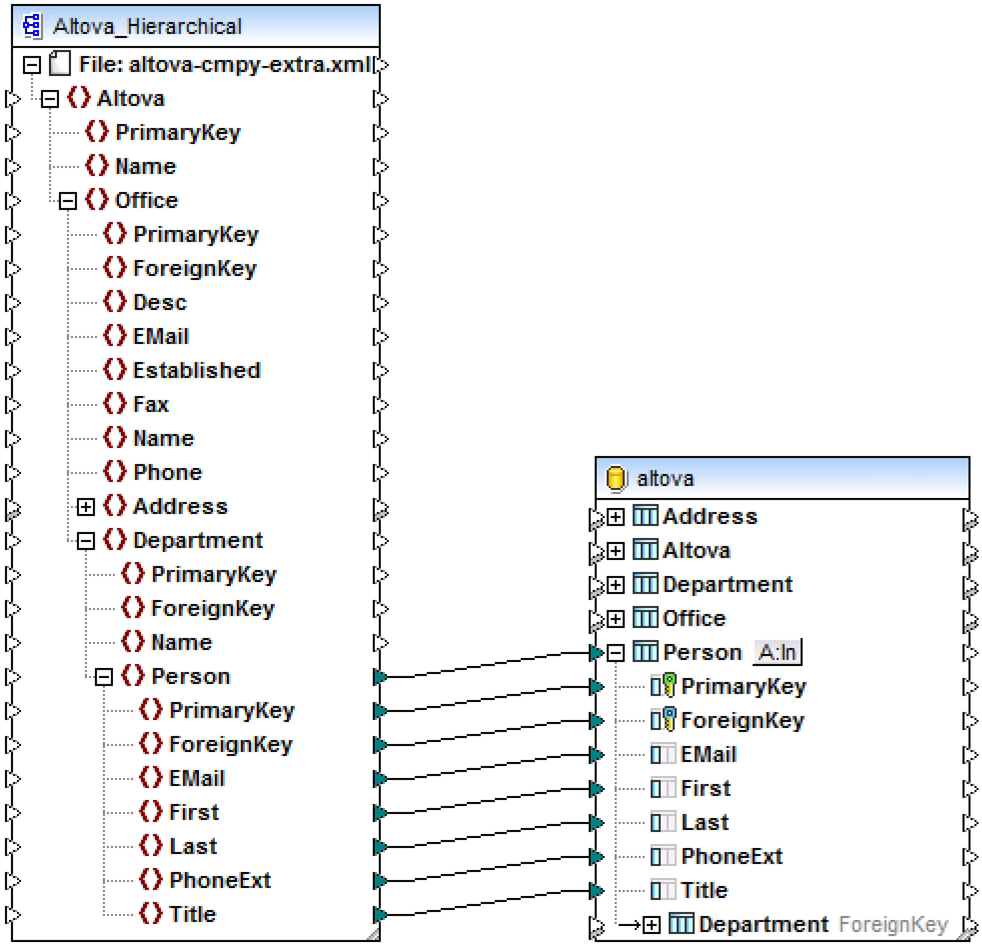
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **altova-cmpy-extra.xml**.

Step 2: Insert the target database

- On the **Insert** menu, click **Database**, and go through the wizard steps to connect to **altova.mdb** (see Example: Adding the "altova.mdb" Database to the Mapping 250).

Step 3: Draw the connections

- Draw the mapping connections as shown below.

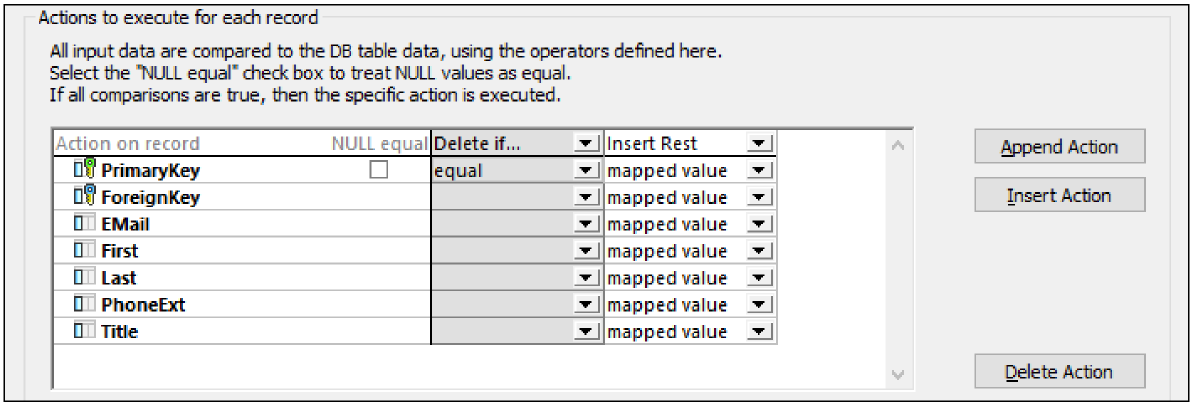


Step 4: Configure the "Delete if... Insert Rest" actions

1. On the target component, click the **Action: Insert** (A:In) button next to the "Person" table.
2. Next to **Action on record**, select **Delete if...** . This changes the database table action to a conditional delete action. That is, the current record will only be deleted when a condition is satisfied (see next step).
3. Next to **PrimaryKey**, select the value **equal**, as shown below. This defines the update condition: that is, the database record will be deleted only when its **PrimaryKey** value is equal to the **PrimaryKey** value coming from the mapping.

Action on record	NULL equal	Delete if...
PrimaryKey	☐	equal
ForeignKey		
EMail		
First		
Last		
PhoneExt		
Title		

4. Optionally, select the **NULL equal** check box. In this example, this check box is irrelevant, because neither the **PrimaryKey** field in the source XML nor the **PrimaryKey** field in the database contains any null values. However, you should select this check box if your data contains null values, and if you want to treat such null values as equal; otherwise, you may get undesired results. For more information, see Handling Nulls in Database Table Actions 323.
5. Click **Append Action**. This adds a new action to the right of the existing **Delete If** action. Configure the new action as **Insert Rest**:



In the image above, the database table actions have been configured in accordance with the goals of the mapping. That is, only when the **Delete If**... condition is satisfied will the record be deleted; otherwise, it will be inserted. The option "mapped value" specifies that values from the mapping will be used to populate all fields of the record.

6. Click **OK** to close the dialog box. Notice that, back on the mapping, the **Action: Insert** button has now changed to an **Action: Delete; Insert** ( ) button. This indicates that both a delete and an insert action is configured for this table.

Step 5: Preview the mapping and update the database

Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview. To run the script against the database:

- On the **Output** menu, click **Run SQL-Script**.

Note: Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases 274.

If you open the "Person" table in the DB query tab of MapForce (see Browsing and Querying Databases 376), you can see the result of the mapping as follows:

- All database records which had corresponding primary keys in the XML file have been deleted. Examples are records with primary key 1, 2, 3, 4, and 5.
- All database records which had no corresponding keys in the XML file remained unaffected. Examples are records with primary key 6, 7, 8, 9, 10, 11, 12, and 13.
- New records have been inserted to the "Person" table (where key did not already exist in the database). Examples are records with primary key 30 and 31.

	PrimaryKey	ForeignKey	EMail	First	Last	PhoneExt	Title
1	6	3	f.landis@nanonull.com	Fred	Landis	951	Program Manager
2	7	3	m.landis@nanonull.com	Michelle	Butler	654	Software Engineer
3	8	3	t.little@nanonull.com	Ted	Little	852	Software Engineer
4	9	3	a.way@nanonull.com	Ann	Way	951	Technical Writer
5	10	3	l.gardner@nanonull.com	Liz	Gardner	753	Software Engineer
6	11	3	p.smith@nanonull.com	Paul	Smith	334	Software Engineer
7	12	4	a.martin@nanonull.com	Alex	Martin	778	IT Manager
8	13	4	g.hammer@nanonull.com	George	Hammer	223	Web Developer
9	30	1	c.Cicada@microtech.com	Camilla	Cicada	765	HR
10	31	1	c.corrigan@microtech.com	Carol	Corrigan	629	Admin

The "Person" table after updating the database

4.2.3.8 "Ignore if..." Action

The table action **Ignore if..**. is used to prevent certain records in a database table from being updated, based on a defined condition. The **Ignore if...** action is only meaningful when used in combination with another database table action (such as the **Insert Rest** action). For example, when mapping data from a source XML to a target database, you can configure an **Ignore if...** condition to check whether a certain field in the source XML is equal to a field in the target database record (typically, a primary key value). If the **Ignore if...** condition is true (that is, the two fields are equal), the database record will be ignored when the mapping runs, and the next defined action (**Insert Rest**, for example) will be executed.

This example shows you how insert records into a database table based on the outcome of the **Ignore if...** condition. It uses the following files:

- **altova-cmpy-extra.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **altova.mdb** — the target database to be updated.

All files are available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. Below, the complete path to them will be omitted, for simplicity.

> The mapping in this example modifies a sample database file. It is strongly recommended to back up the original database and start with a new copy before following the steps below. This ensures that the original examples are not overridden and that you get the same results as below. For more information, see Executing Mappings Which Modify Databases 274.

The goal of the mapping is as follows:

- If any person records with the same `PrimaryKey` exist both in the source XML and the target Person table, no action must be taken against them (that is, they must be ignored).

- If any person records which do not meet the above condition exist in the Person table, no action must be taken against them either.
- Records from the source XML which do not have a counterpart (no primary key) in the Person table must be treated as new and inserted into the Person table with a new primary key.

To achieve the mapping goal, we will take the steps below.

Step 1: Insert the source XML component

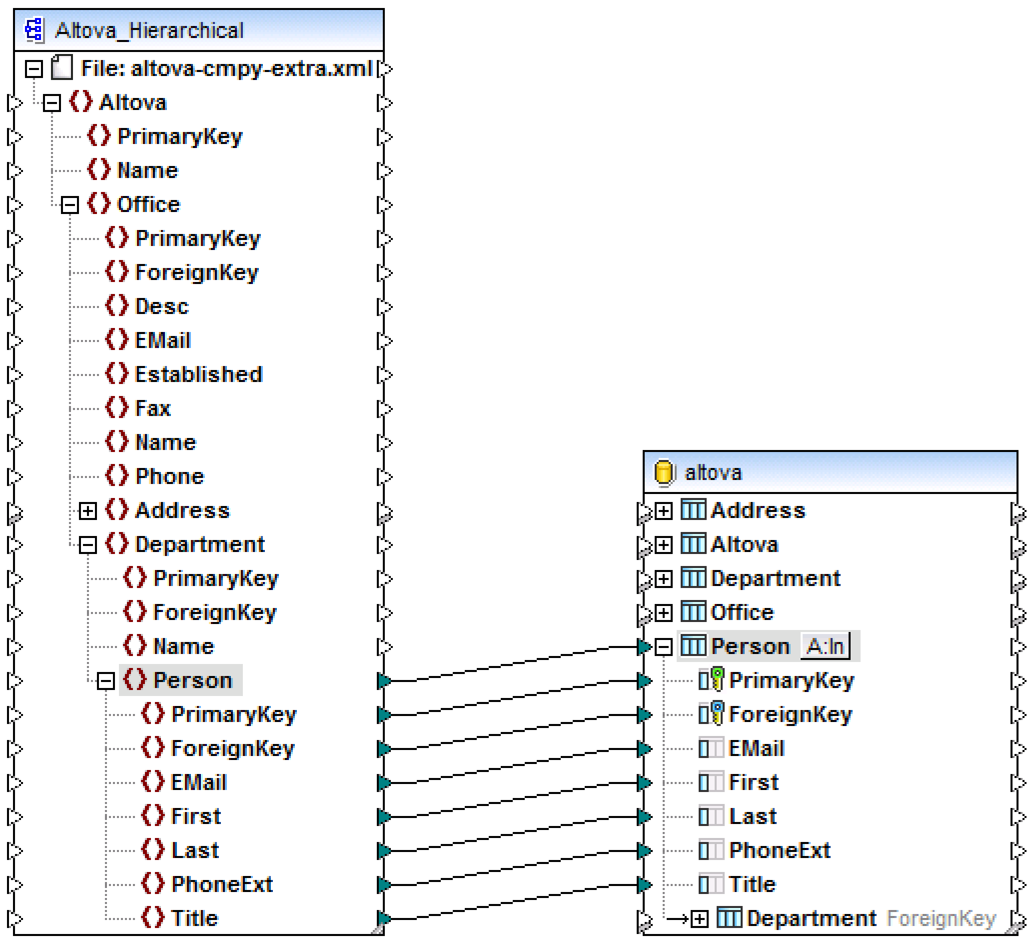
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **altova-cmpy-extra.xml**.

Step 2: Insert the target database

- On the **Insert** menu, click **Database**, and go through the wizard steps to connect to **altova.mdb** (see Example: Adding the "altova.mdb" Database to the Mapping 250).

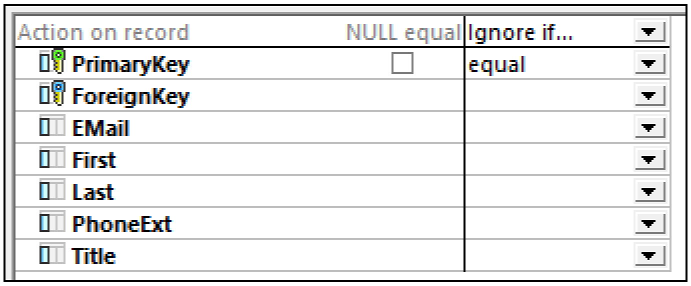
Step 3: Draw the connections

- Draw the mapping connections as shown below.

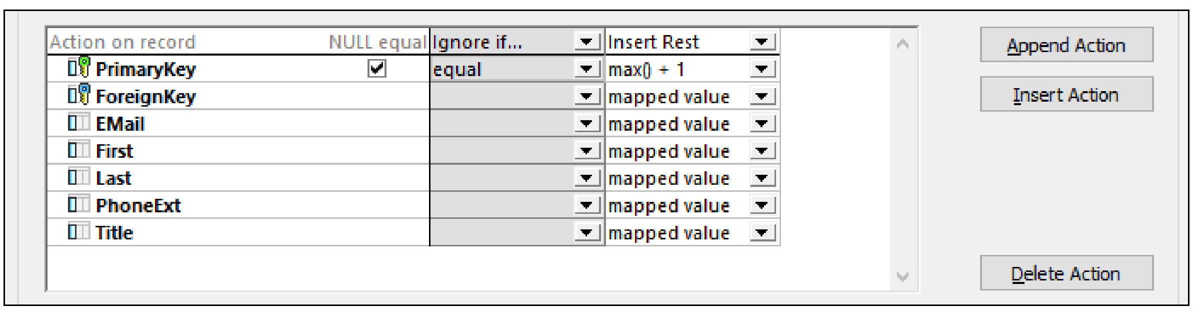


Step 4: Configure the "Ignore if... Insert Rest" actions

1. On the target component, click the **Action: Insert** (A:In) button next to the "Person" table.
2. Next to **Action on record**, select **Ignore if...** . This changes the database table action to a conditional ignore action. That is, the current record will only be ignored when a condition is satisfied (see next step).
3. Next to **PrimaryKey**, select the value **equal**, as shown below. This defines the ignore condition: that is, the database record will be ignored only when its `PrimaryKey` value is equal to the `PrimaryKey` value coming from the mapping.



4. Optionally, select the **NULL equal** check box. In this example, this check box is irrelevant, because neither the **PrimaryKey** field in the source XML nor the **PrimaryKey** field in the database contains any null values. However, you should select this check box if your data contains null values, and if you want to treat such null values as equal; otherwise, you may get undesired results. For more information, see [Handling Nulls in Database Table Actions](323).
5. Click **Append Action**. This adds a new action to the right of the existing **Ignore If** action. Configure the new action as **Insert Rest**, with the primary key set to **max() + 1**, as shown below:



In the image above, the database table actions have been configured in accordance with the goals of the mapping. That is, only when the **Ignore If**... condition is satisfied will the record be skipped; otherwise, it will be inserted. The option "mapped value" specifies that values from the mapping will be used to populate all fields of the record. The option **max() + 1** generates a unique, new primary key value for the record.

6. Click **OK** to close the dialog box. Notice that, back on the mapping, the **Action: Insert** A:In button has now changed to an **Action: Ignore; Insert** ( ) button. This indicates that both the ignore and insert actions are configured for this table.

Step 5: Preview the mapping and update the database

Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview. To run the script against the database:

- On the **Output** menu, click **Run SQL-Script**.

Note: Running the SQL script directly from MapForce is just one of the ways to update the database, see also Executing Mappings Which Modify Databases 274.

If you open the "Person" table in the DB query tab of MapForce (see Browsing and Querying Databases 376), you can see the result of the mapping as follows:

- All database records which had corresponding primary keys in the XML file satisfied the **Ignore if...** and remained unaffected. Examples are records with primary key 1, 2, 3, 4, and 5.
- All database records which had no corresponding keys in the XML file did not satisfy the **Ignore if...** condition but nevertheless remained unaffected. Examples are records with primary key 6, 7, 8, 9, 10, 11, 12, and 13.
- New records have been inserted to the "Person" table (where key did not already exist in the database). Examples are records with primary key 30 and 31 in the source XML file. These were inserted into the database with the new primary key 22 and 23, respectively.

	PrimaryKey	ForeignKey	EMail	First	Last	PhoneExt	Title
1	1	1	v.callaby@nanonull.com	Vernon	Callaby	582	Office Manager
2	2	1	f.further@nanonull.com	Frank	Further	471	Accounts Receivable
3	3	1	l.matise@nanonull.com	Loby	Matise	963	Accounting Manager
4	4	2	j.firstbread@nanonull.com	Joe	Firstbread	621	Marketing Manager Europe
5	5	2	s.sanna@nanonull.com	Susi	Sanna	753	Art Director
6	6	3	f.landis@nanonull.com	Fred	Landis	951	Program Manager
7	7	3	m.landis@nanonull.com	Michelle	Butler	654	Software Engineer
8	8	3	t.little@nanonull.com	Ted	Little	852	Software Engineer
9	9	3	a.way@nanonull.com	Ann	Way	951	Technical Writer
10	10	3	l.gardner@nanonull.com	Liz	Gardner	753	Software Engineer
11	11	3	p.smith@nanonull.com	Paul	Smith	334	Software Engineer
12	12	4	a.martin@nanonull.com	Alex	Martin	778	IT Manager
13	13	4	g.hammer@nanonull.com	George	Hammer	223	Web Developer
14	14	4	j.band@nanonull.com	Jessica	Bander	241	Support Engineer
15	15	4	l.king@nanonull.com	Lui	King	345	Support Engineer
16	16	5	s.meier@nanonull.com	Steve	Meier	114	Office Manager
17	17	5	t.bone@nanonull.com	Theo	Bone	331	Accounts Receivable
18	18	6	m.nafta@nanonull.com	Max	Nafta	122	PR & Marketing Manager US
19	19	7	v.bass@nanonull.com	Valentin	Bass	716	IT Manager
20	20	7	c.franken@nanonull.com	Carl	Franken	147	Support Engineer
21	21	7	m.redgreen@nanonull.com	Mark	Redgreen	152	Support Engineer
22	22	1	c.Cicada@microtech.com	Camilla	Cicada	765	HR
23	23	1	c.corrigan@microtech.com	Carol	Corrigan	629	Admin

The "Person" table after updating the database

4.2.3.9 Bulk Inserts (MapForce Server)

The **Use Bulk Transfer** option allows you to insert data at very high speed from a MapForce component (TXT, CSV, DAT, etc.) into a database table. Using this option dramatically speeds up the Insert process, as only one statement needs to be executed instead of many.

The **Use Bulk Transfer** option can be enabled in MapForce, at mapping design time, as shown below. A mapping where this option is enabled can be executed in MapForce, but no bulk insert applies at this stage. The actual bulk transfer of data occurs when the mapping is run by MapForce Server.

Bulk transfer is supported when the following conditions are true:

- The mapping transformation language is set to BUILT-IN. For further information, see Transformation Languages 27.
- The mapping is run by MapForce Server (either standalone or under FlowForce Server management). This means that the mapping must be either compiled to .mfx format or deployed to FlowForce Server. For further information, see Compiling Mappings to MapForce Server Execution Files 1378 and Deploying Mappings to FlowForce Server 1381.
- The MapForce Server license is not limited to "single thread execution" on a multi-core machine. That is, the **Limit to single thread execution** check box in the "Server Management" tab of Altova LicenseServer must be inactive.
- The database action is "Insert all", see also Database Table Actions Settings 326.
- The table into which the data is to be bulk loaded must be a "leaf" table, that is, on the lowest hierarchy of the database. There should not be any related tables, views, or stored procedures referencing the table in the mapping.
- The database driver supports bulk insert on WHERE conditions.

The following table summarizes support for bulk inserts depending on the database kind and the driver used.

	ADO	**ODBC**	**JDBC**	**ADO.NET**	**Native**
Access	No	No	n/a	n/a	n/a
DB2	No	Yes	Yes	Yes	n/a
Firebird	n/a	Yes	Yes	No	n/a
Informix	No	Yes	Yes	Yes	n/a
iSeries	No	Yes	Yes	Yes	n/a
MariaDB	No	Yes	Yes	No	n/a
MySQL	n/a	Yes*	Yes	No	n/a
Oracle	No	Yes	Yes	Yes	n/a
PostgreSQL	n/a	Yes	Yes	n/a	Yes
Progress	n/a	Yes	Yes	n/a	n/a
SQL Server	Yes	Yes	Yes	Yes	n/a

	ADO	ODBC	JDBC	ADO.NET	Native
SQLite	n/a	n/a	n/a	n/a	No
Sybase	No	Yes	Yes	n/a	n/a
Teradata	n/a	Yes	Yes	n/a	n/a

** MySQL version 5 or later is required.*

Note: To enable bulk insert support for MySQL and MariaDB via JDBC, use the `rewriteBatchedStatements=true` connection option.

Bulk insert example

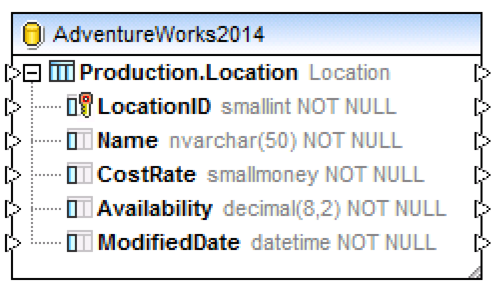
This example shows you how create a mapping which bulk loads data from a sample **source.txt** file into a target database. The example uses SQL Server 2014 and the AdventureWorks 2014 database. The latter can be downloaded from the AdventureWorks samples page on GitHub (https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks).

```
Location A,15.3,39
Location B,46,34
Location C,56.33,0
Location D,0,399
Location E,0,97.43
```

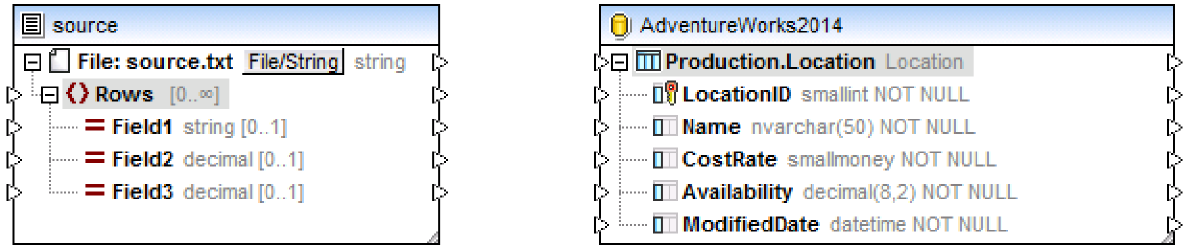
source.txt

To define a bulk insert:

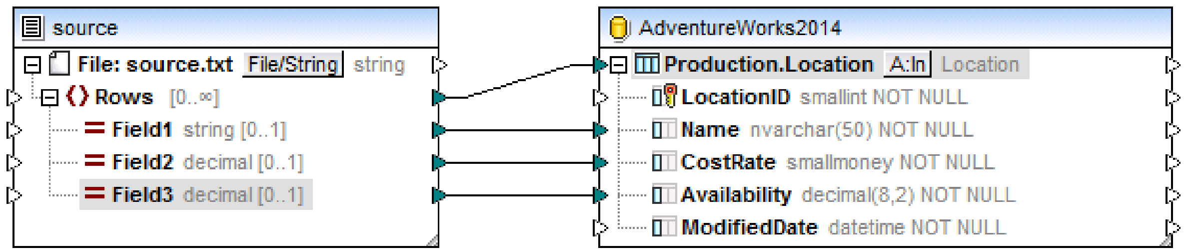
1. Set the transformation language to BUILT-IN ( ).
2. Connect to the AdventureWorks 2014 database and add the "Production.Location" table to the mapping. For more information, see Adding Databases to the Mapping 249.



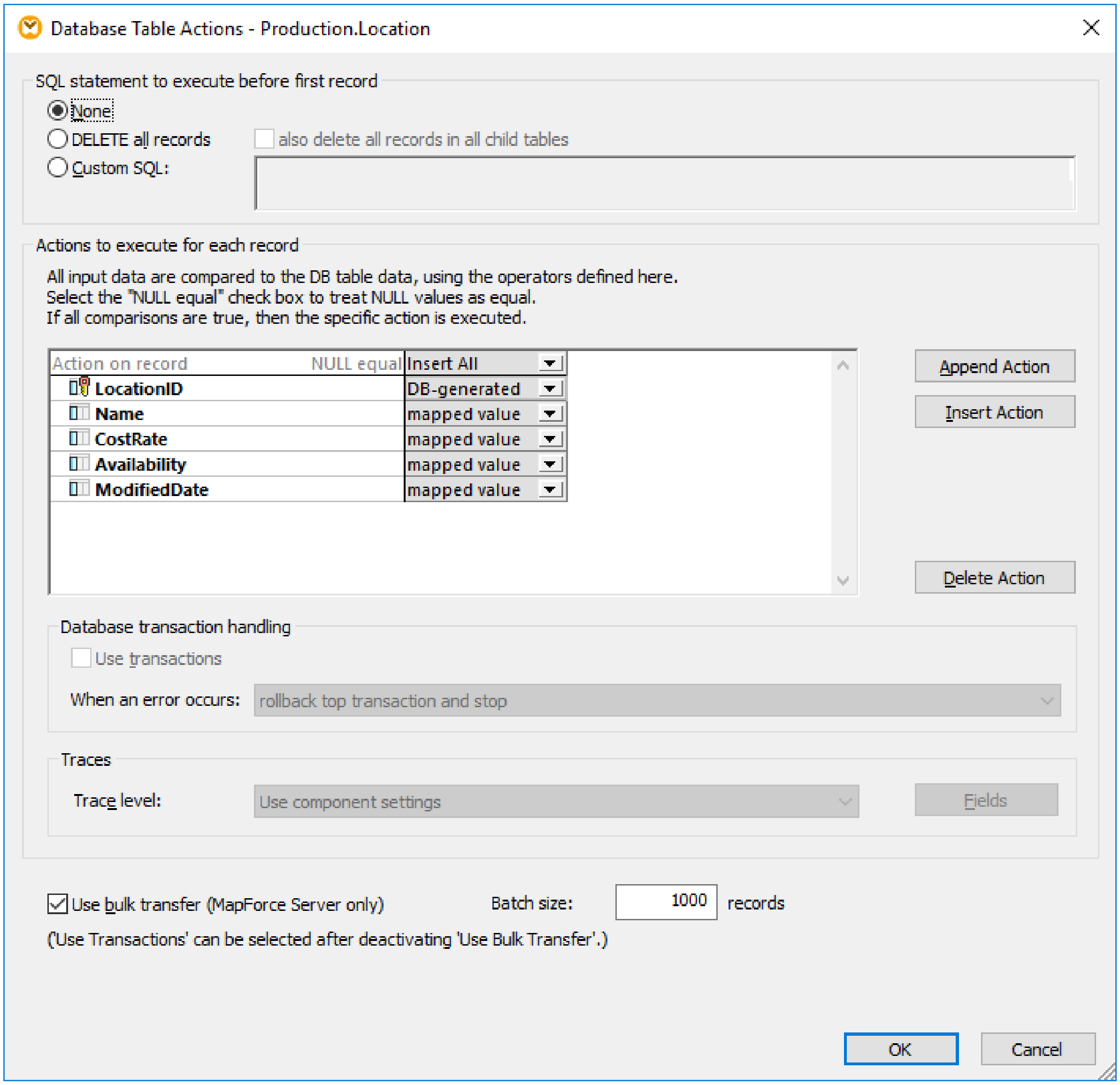
3. On the **Insert** menu, click **Text**, and add a source text file (such as the **source.txt** sample above) to the mapping. For more information, see CSV and Text Files 563. Make sure that the data types of both the source and the target components are compatible. Data types are visible in the component when the **Show Data Types** ( ) toolbar button is enabled.



4. Draw the mapping connections as shown below. Note that the **Database Actions** button A:In is now visible to the right of the table name.



5. Click the **Database Actions** button (A:In), select the **Use Bulk Transfer** check box, and click OK to confirm.



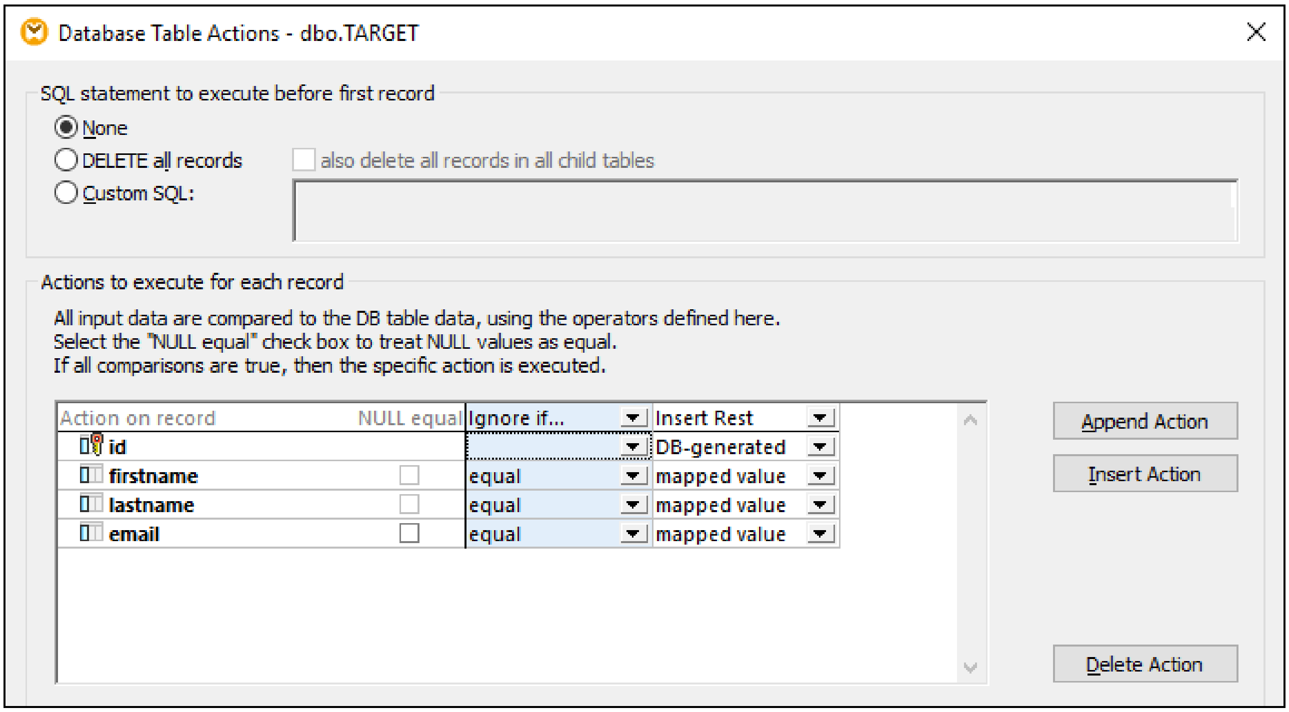
In the dialog box above, notice that the "Action on record" is "Insert All". The **Batch size** field defines the number of records to be inserted per action.

Note: When the **Use Bulk Transfer** option is enabled, the **Use Transactions** option becomes disabled, and vice versa. If you want to enable transaction processing, click to clear the **Use Bulk Transfer** check box.

Now that bulk insert is enabled, the next step is to execute the mapping in MapForce Server (either standalone or under FlowForce Server management). For further information, see Compiling Mappings to MapForce Server Execution Files [1378] and Deploying Mappings to FlowForce Server [1381].

4.2.3.10 Handling Nulls in Database Table Actions

When a mapping updates a target database by means of table actions such as "Ignore If", "Update If", "Delete If", MapForce compares the source data against the target data and generates internal database update queries as a result. These internal queries are available for preview in the **Output** pane of MapForce, see Executing Mappings Which Modify Databases[274]. The generated queries reflect the comparison conditions that were defined from the "Database Table Actions" dialog box.



Database Table Actions dialog box

If your source or the target data contains nullable fields, you have two options to compare null values from the source with those in the target data:

1. treat null values as equal
2. treat null values as not equal

For this purpose, the dialog box above displays a **NULL equal** check box next to some fields. Selecting or not selecting this check box might affect the mapping result, and is the subject of this topic. Be aware that the check box can be selected only for fields which are nullable, and when at least one table action has an "equal" or "equal (ignore case)" condition.

To avoid undesired results, you should select the **NULL equal** check box if all of the following conditions are true:

1. The "Database Table Actions" dialog box contains "Ignore if", "Update if", "Delete if" actions, and
2. These actions are taken against records that may contain null values, and
3. Your mapping requires that null values in the source must be treated as equal with null values in the target.

By default, the **NULL equal** check box is not selected. If the conditions above are true and the check box is not selected, the target database table might not be updated as expected (for example, more rows would be inserted than necessary). This happens because null values affect the data comparison. For example, in the image above, let's suppose that a record has a null **email** both in the source and target. If you select the **NULL equal** check box, that record will satisfy the **Ignore If...** condition and will be ignored (skipped). However, if you don't select the **NULL equal** check box, the record will no longer satisfy the **Ignore If...** condition and will be inserted into the database.

Example

To better understand null awareness in mappings, let's analyze an example where comparison of null data occurs. This example uses a Microsoft SQL Server database; however, it is also applicable for any other supported database type. Optionally, if you have Microsoft SQL Server, you can create the tables and data used in this example by running the following database script:
<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\CreateNullableFields.sql.

For convenience, the database tables are illustrated below. Both tables store people data and have the same columns. Also, the column `email` can contain null data in both tables.

```
+----+-----------+-----------+--------------------------+
| id | firstname | lastname  | email                    |
+----+-----------+-----------+--------------------------+
| 1  | Toby      | Hughey    | t.hughey@nanonull.com    |
| 2  | Mia       | Dahill    | NULL                     |
| 3  | Fred      | Weinstein | f.weinstein@nanonull.com |
+----+-----------+-----------+--------------------------+
```

The SOURCE table

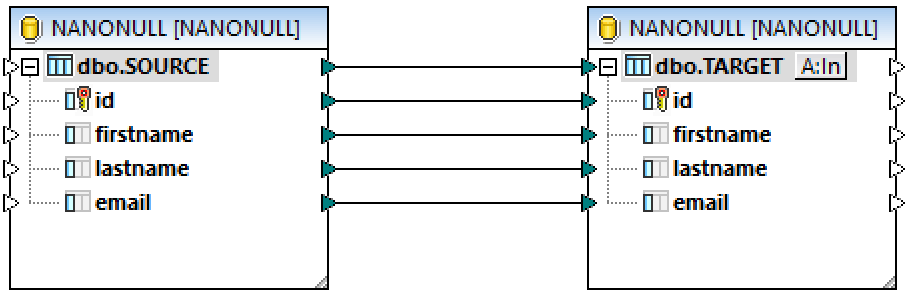
```
+----+-----------+-----------+--------------------------+
| id | firstname | lastname  | email                    |
+----+-----------+-----------+--------------------------+
| 1  | Mia       | Dahill    | NULL                     |
| 2  | Fred      | Weinstein | f.weinstein@nanonull.com |
+----+-----------+-----------+--------------------------+
```

The TARGET table

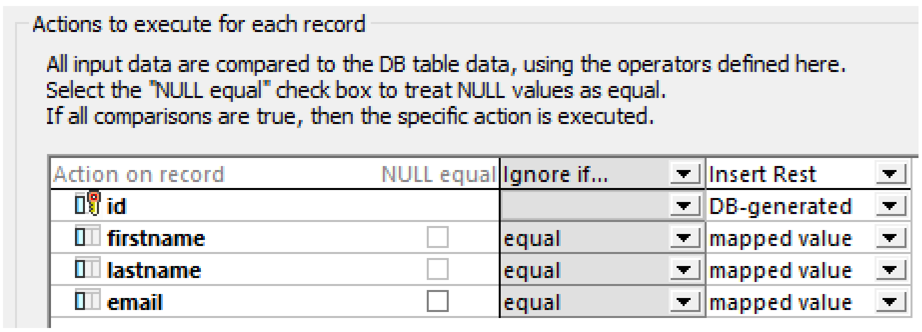
Let's suppose your task is to merge data from the `SOURCE` table into the `TARGET` table. Only the new records must be inserted into the `TARGET` table (in this example, "Tobie Hughey"). The records which exist in both tables ("Mia Dahill" and "Fred Weinstein") must be ignored.

The task can be accomplished as follows.

1. On the **Insert** menu, select **Database**. Follow the wizard steps to connect to the database (see also Connecting to a Database [163]). When prompted to add database objects, select the table `SOURCE`.
2. On the **Insert** menu, select **Database**. Connect to the database again and add the table `TARGET` to the mapping.
3. Draw the mapping connections between the source and target components.



4. Click the **Action:Insert** A:In button and configure the database table actions as follows:



As illustrated above, a combination of "Ignore if.. Insert Rest" actions are defined. This configuration means that, for each record, the mapping checks if:

- `firstname` in the source is equal to `firstname` in the target, AND
- `lastname` in the source is equal to `lastname` in the target, AND
- `email` in the source is equal to `email` in the target.

If all the conditions above are true, the record is ignored (according to the requirement). Otherwise, a new record is inserted into the target table. The `id` of the new record is generated by the database, while the other fields (`firstname`, `lastname`, `email`) are populated with values mapped from the source.

Importantly, the check box next to `email` enables or disables NULL-aware comparison for this field. This check box must be selected, because `email` can contain null values (namely, "Mia Dahill" has a null email address). To see the role played by this check box, try updating the database two times: first time, with the check box selected, and a second time with the cleared check box.

To update the database, click the **Output** tab and run the menu command **Output | Run SQL-Script**.

If the check box is selected, MapForce has explicit indication that you want to treat the null values as equal. Therefore, the record "Mia Dahill" is not inserted in the target table, which is the intended result.

If the check box is not selected, the record "Mia Dahill" is inserted in the target table (despite that fact that it exists already), which is not the intended result. The reason is that no explicit indication was given to MapForce that you want to treat null values as equal. A similar situation would occur if you ran the following query against the database (this query retrieves no records because the null value is compared with the "=" operator so it is not NULL-aware):

```
SELECT firstname, lastname, email FROM TARGET WHERE firstname = 'Mia' AND lastname =
'Dahill' AND email = NULL;
```

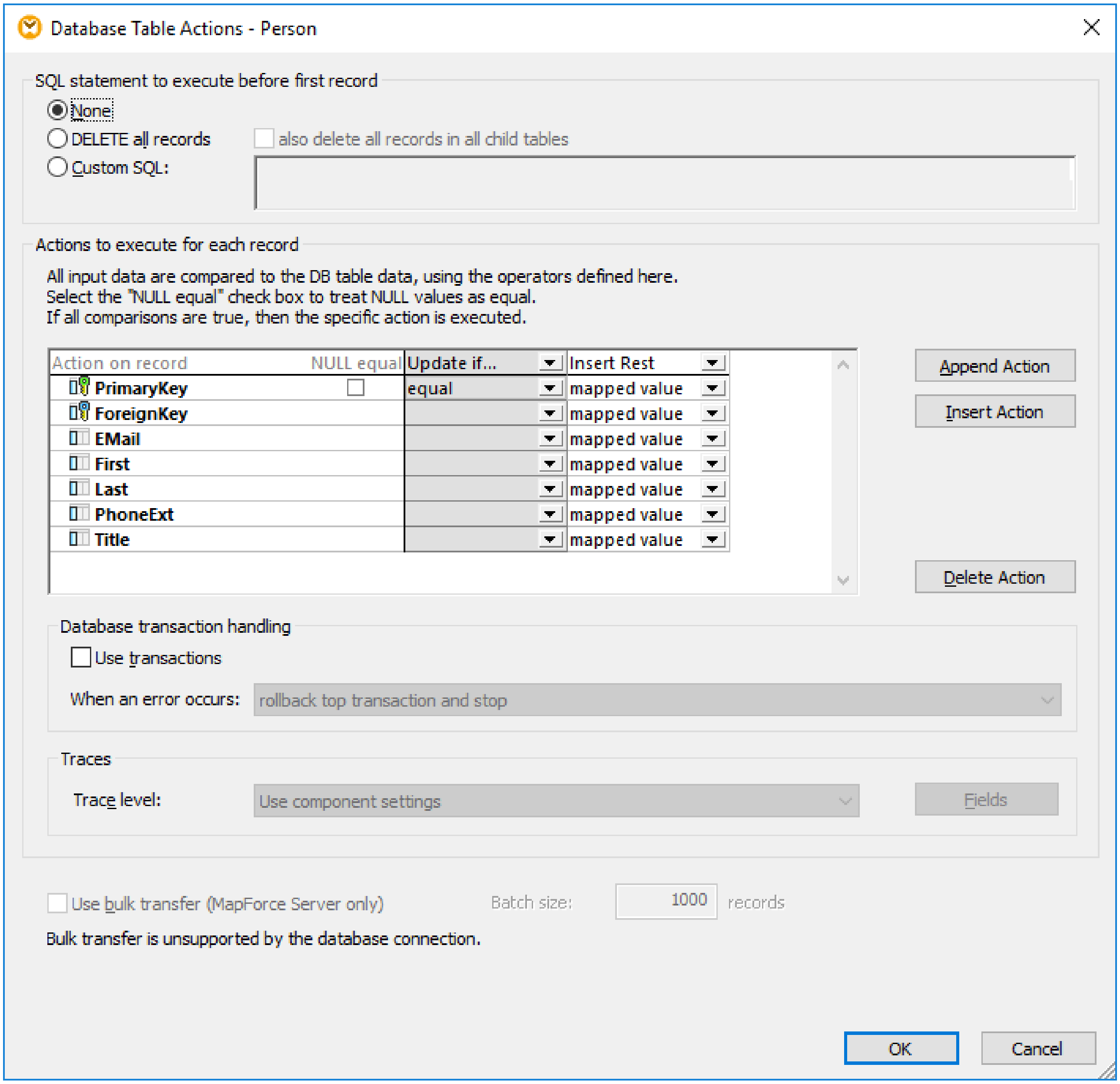
In order to be NULL-aware, the query above would have to be rewritten as follows:

```
SELECT firstname, lastname, email FROM TARGET WHERE firstname = 'Mia' AND lastname =
'Dahill' AND email IS NULL;
```

Note: The queries above are only for illustrative purposes and do not reflect the actual syntax of internal queries generated by MapForce. MapForce adapts the syntax of generated queries according to the database type (since various database vendors have different approaches to handling null comparisons).

4.2.3.11 Database Table Actions Settings

Whenever you create a mapping connection to a database table, a Database Actions button appears next to the affected table. Clicking this button opens Database Table Actions dialog box, from where you can configure the database insert, update, and delete actions, as well as other options.



Database Table Actions dialog box

Below is a description of the settings available on the Database Table Actions dialog box.

SQL statement to execute before first record

In this group box, you can define SQL statements that are executed before any actions defined under **Actions to execute for each record**. Select the desired radio button:

- **None** — No action is carried through. This is the default setting.
- **DELETE all records** — All records from the selected table are deleted before any specific table action defined in the **Actions to execute for each record** group box is performed. Activate the **also delete**

all records in all child tables check box if you also want to get rid of the data stored in child tables of the selected table. For an example, see Inserting Data into Multiple Linked Tables 286.
- **Custom SQL** — Write a custom SQL statement to affect the complete table. Note that support for multiple SQL statements in one query depends on the database, connection method, and the driver used.

Actions to execute for each record

This group of settings specify the database actions that are to take place against this table when the mapping runs. To manage table actions, click the **Append Action**, **Insert Action**, or **Delete Action** buttons. Multiple actions can be defined if necessary (for example, an "Update if..." action followed by an "Insert Rest" action.

The defined table actions are processed from left to right. In the example above, the "Update if..." action is processed first. If the update condition is not satisfied then the following action is processed (in this example, the "Insert Rest" action). Note the following:

- All the defined conditions of one action must be satisfied for the table action to be executed. When this is the case, all those fields are updated where a connection exists between the source and target items on the mapping. Any subsequent table actions (to the right of an action whose condition matched) are ignored for that record.
- If the defined condition is not satisfied, then the table action is skipped, and the next action (to the right) is processed.
- If none of the conditions are satisfied, no table action takes place.

> Any table actions defined after "Insert All" or "Insert Rest" actions will never be executed, because no column conditions exist for insert actions. A dialog box appears if this is the case, stating that the subsequent table action columns will be deleted.

The **NULL Equal** check box next to each record, where applicable, instructs MapForce that null values in the source record must be treated as equal with null values in the target record. Not selecting this check box may lead to incorrect results, see Handling Nulls in Database Table Actions 323.

When the mapping updates a table which has foreign key relationships to other tables, the following options can be used:

Delete data in child tables	This option is meaningful when you select the "Update if..." action for a parent table. It might be necessary if the number of records in the source file is different from the number of records in the target database, and you want to keep the database synchronized (no orphaned data in child tables). See also Options for Child Tables When Updating a Parent Table 303.
Ignore input child data	Use this option when you want to update a target parent table, without affecting any of the child tables/records of that table. See also Options for Child Tables When Updating a Parent Table 303.

For examples which illustrate various combinations of actions, see:

- Inserting Data into a Table 283
- Inserting Data into Multiple Linked Tables 286

- Updating a Table 291
- Options for Child Tables When Updating a Parent Table 303
- "Update if... Insert Rest" Action 294
- "Delete if..." Action 311
- "Ignore if..." Action 315

Use Transactions

Enables database transactions for this particular table action. For more information, see Handling Database Exceptions 415.

Use Bulk Transfer

Enables bulk transfer (multiple INSERT statements as one query). Bulk transfer is supported if the mapping is executed by MapForce Server and the database action is "Insert All". For more information, see Bulk Inserts (MapForce Server) 319.

4.2.3.12 Example: Mapping Data from XML to SQLite

This example walks you through the steps required to create a MapForce mapping which reads data from an XML file and writes it to a SQLite database. The example is accompanied by a sample mapping design (.mfd) file. If you want to look at the sample file before starting this example, you can open it from the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\XMLtoSQLite.mfd**.

The goal of the example is to insert data from an XML file into a SQLite database. To accomplish the goal of the example, you will need an empty SQLite database to which data will be written. As illustrated below, you can create and explore the SQLite database either with Altova DatabaseSpy, or with the command-line shell available from the official SQLite website.

To create the SQLite database:

If DatabaseSpy is installed on your computer (either standalone or as part of Altova MissionKit), you can create the new SQLite database as follows:

1. Run DatabaseSpy.
2. On the **File** menu, click **Create a Database Connection**.
3. Click **Connection Wizard**, and then click **SQLite**.
4. Click **Create a new SQLite database**, enter **c:\sqlite\articles.sqlite** as path, and then click **Connect**.
5. When prompted to set a data source name, leave the default name as is.
6. Open a new SQL editor (**Ctrl+N**) and run the following query against the database:

```
create table articles (number smallint, name varchar(10), singleprice real);
```

Otherwise, follow the steps below to create the database:

1. Download the SQLite command-line shell for Windows from the SQLite download page (http://www.sqlite.org/download.html) and unpack the .zip archive to a directory on your local machine (for the scope of this example, use **c:\sqlite**).
2. Run **c:\sqlite\sqlite3.exe** and enter the following statement:

```
create table articles (number smallint, name varchar(10), singleprice real);
```

This creates the table `articles` in the in-memory database. The table `articles` consists of three columns: *number*, *name*, and *singleprice*. The purpose of these columns is to store data from the elements with the same name defined in the source XML schema. Each column is declared with a data type suitable for the data expected to be stored in that column.

3. Run the command:

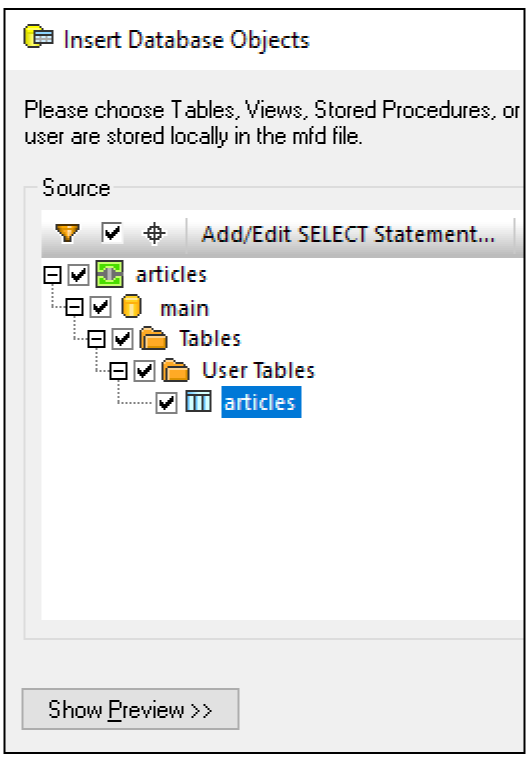
```
.save articles.sqlite
```

This saves the in-memory database to the current working path: **c:\sqlite\articles.sqlite**. Note that you will need to refer to this path in subsequent steps.

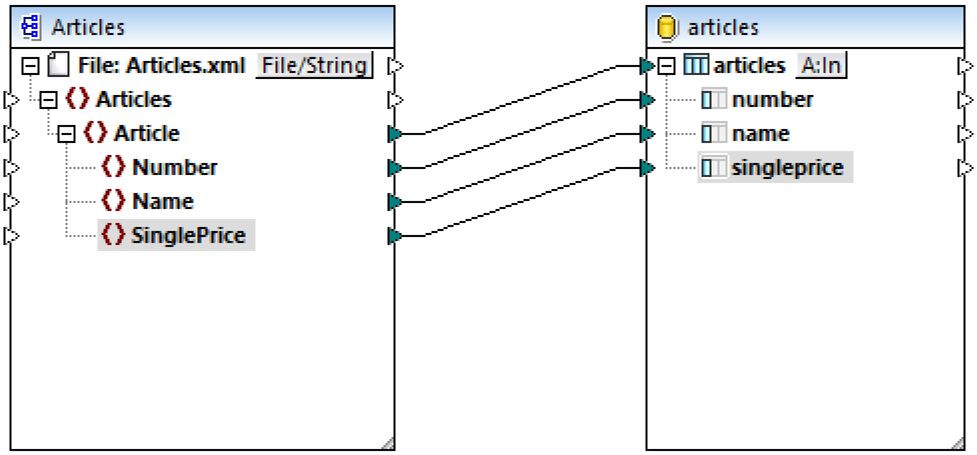
You have now finished creating the sample SQLite database required for this example.

To create the XML to SQLite mapping design:

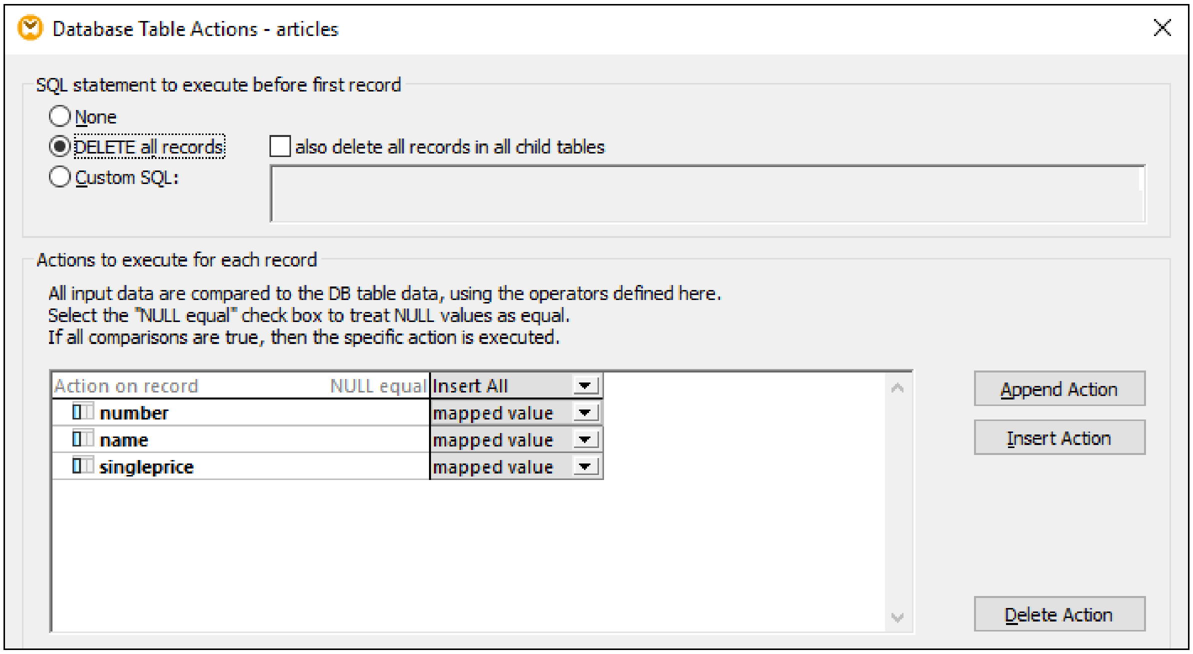
1. Run MapForce and make sure that the transformation language is set to BUILT-IN (use the menu command **Output | Built-in Execution Engine**).
2. Add to the mapping area the file **Articles.xml** located in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder (use the menu command **Insert | XML Schema/File**).
3. Add to the mapping area the database **articles.sqlite** created in previous steps (use the menu command **Insert | Database**), and then select SQLite.
4. Click **Connect**. When prompted to choose the database objects, select the `articles` table.



5. Draw the connections as shown below:



6. Click the **A:In** button on the database component and select the **Delete All records** option. This ensures that, every time the mapping is executed, all existing database rows are first deleted, in order to prevent duplication.



7. Click the **Output** tab of the main mapping window. MapForce executes the mapping using the built-in execution engine and displays the create SQL query in the Output window.

```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
C:/sqlite/articles.sqlite
*/

PRAGMA foreign_keys = ON;

DELETE FROM "articles"

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (1, 'T-Shirt', 25)

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (2, 'Socks', 2.3)

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (3, 'Pants', 34)

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (4, 'Jacket', 57.5)
```

8. Run the SQL script to populate the database (use the menu command **Output | Run SQL-Script)**. If MapForce does not encounter any runtime errors, the records are inserted into the SQLite database.

```
/*
The following SQL statements were executed during "Generate output" function.
Every single result is written right to the "-->>>" string.
These statements are only for preview and may not be executed in another SQL query tool!
The database was connected using the following connection-string:
C:/sqlite/articles.sqlite
*/

PRAGMA foreign_keys = ON;
-->>> OK. 0 row(s).

DELETE FROM "articles"
-->>> OK. 0 row(s).

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (1, 'T-Shirt', 25)
-->>> OK. 1 row(s).

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (2, 'Socks', 2.3)
-->>> OK. 1 row(s).

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (3, 'Pants', 34)
-->>> OK. 1 row(s).

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (4, 'Jacket', 57.5)
-->>> OK. 1 row(s).
```

To check whether data was correctly inserted into the SQLite database:

1. Click the **DB Query** tab of MapForce.
2. Select the **articles** entry from the drop-down list at the top of the **DB Query** pane.
3. Enter the following query in the SQL Editor:

```
select * from articles;
```

Alternatively, follows the steps below:

1. Run the file **c:\sqlite\sqlite3.exe** and open the database with the command:

```
.open articles.sqlite
```

2. Run the following select statement:

```
select * from articles;
```

Regardless of the approach you choose to select the table data (MapForce or SQLite shell), the query should returns four rows now. This corresponds to the number of records in the source XML file, which was the intended goal of this example.

4.2.4 Joining Database Data

In mappings that read data from databases, you can join database objects such as tables or views by adding a Join component to the mapping. For example, you could combine data from two or more tables bound by foreign key relationships, which is the typical way data is stored in relational databases. The result would be the same as if you ran against the database an SQL query where two or more tables are joined by means of an INNER JOIN (or LEFT JOIN, if applicable) operation.

Depending on the kind of data connected to the join component, the join operation can happen either in standard (non-SQL) mode, or in SQL mode. Joins in non-SQL mode are undertaken by MapForce, while joins in SQL mode are undertaken by the database from which the mapping reads data.

Joins in non-SQL mode are more flexible because they support more component types as input (for example, the join can be between tables from different databases, or between XML structures and database tables). For an example of a non-SQL join, see Example: Join XML Structures [866]. On the other hand, a non-SQL join causes the mapping engine to perform memory-costly comparisons (because the total number of comparisons represents the cross-join, or Cartesian product, of all joined structures). Usually this process is very fast and causes negligible load in mappings which are not data-intensive; however, if the joined data sources consist of a huge number of records, then the mapping will require significant time to execute. If your mappings must process a very large number of records, consider licensing MapForce Server Advanced Edition, which includes dedicated join optimization to speed up the mapping execution.

A join in SQL mode accepts only eligible database objects as input (such as tables or views), so it is not as flexible as a non-SQL join. However, it offers better mapping performance because it is executed natively by the database. For further information, see Joins in SQL Mode [334].

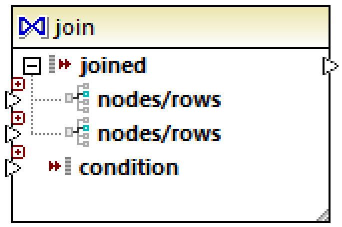
Note: Using a Join component is not the only way to join database tables or views. Joins applicable to databases can also be performed by using SQL SELECT statements, see SQL SELECT Statements as Virtual Tables [357]. A major difference between SQL SELECT statements and Join components is that the former are written by hand so they might provide more flexibility. Join components are a simpler alternative if you do not feel comfortable writing SQL statements by hand.

To add a Join component:

1. Set the mapping transformation language to BUILT-IN (to do this, either click the BUILT IN toolbar button, or

use the **Output | Built-In Execution Engine** menu command).

2. On the **Insert** menu, click **Join**. Alternatively, click the **Join** ( ) toolbar button. The Join component appears on the mapping. By default, it accepts data from two structures, so it has two `nodes/rows` inputs. If necessary, you can add new inputs to the join by clicking the **Add Input** ( ) button, see Joining Three or More Structures 865.



3. Connect the structures that are to be joined to the `nodes/rows` items of the join component.
4. Add the condition for the join (or multiple conditions). To do this, right-click the Join component and select **Properties**. Join conditions can also be added directly from the mapping, by connecting the Boolean result of some function to the `condition` item of the Join component. In certain cases when database tables are joined, the join condition (or conditions) can be created automatically by MapForce. For further information, see Adding Join Conditions 862.

Notes:

- Join components are supported when the target language of the mapping is set to BUILT-IN. Code generation in C#, C++, or Java is not supported.
- When a structure is not a valid or supported input source for the join, MapForce displays hints either immediately directly on the mapping, or in the Messages window, when you validate the mapping (see Validating Mappings 105).
- Join components should not be connected to input parameters or results of inline user-defined functions. If such connections exist, validation errors will occur during mapping validation.
- When you connect eligible database components (such as tables or views) directly to a Join component, an **SQL mode** (SQL) button automatically appears at the top-right corner of the Join component. When enabled, this button provides special SQL features applicable to the join operation (see Joins in SQL Mode 334).
- It is not possible to connect the output of the `joined` item to another Join component. If necessary, however, you can connect a partial result of one join to another one.

4.2.4.1 Joins in SQL Mode

When you connect eligible database components (such as tables or views) directly to a join component, an **SQL mode** SQL button appears at the top-right corner of the join component. When SQL mode is enabled, the join operation is undertaken by the database from where the mapping reads data. In other words, MapForce will internally send to the database a query with the appropriate SQL syntax to select and combine data from all tables that take part in the join. Importantly, you do not need to write any SQL; the required query is produced based on how you visually designed the Join component on the mapping, as you will see in subsequent examples.

For SQL mode to be possible, the following conditions must be met:

1. Both objects (tables or views) that are to be joined must be from the same database.

2. Both objects that are to be joined must originate from the same MapForce component. (Note that you can quickly add/remove database objects in a component as follows: right-click the database component, and select **Add/Remove/Edit Database Objects** from the context menu.)
3. The Join condition (or conditions) must be defined only from the component properties (by right-clicking the header of the join component, and selecting **Properties**), and not on the mapping (see also Adding Join Conditions 862).

Note: When database tables are joined in SQL mode, MapForce will create the join condition (or conditions) automatically, based on foreign key relationships detected between tables. For automatic join conditions to happen, the database tables must be in a child-parent relationship on the MapForce component (that is, one table must be "parent" or "child" of another one on the component), see Example: Join Tables in SQL Mode 337.

4. All database tables must not yet be in the current target context. When the join result is used in a target component, none of the joined tables may be connected directly or indirectly to any target parent nodes. For more information about how a mapping is executed, see Mapping Rules and Strategies 1275.

You can view or control the SQL mode through the **SQL** (SQL) button at the top-right corner of the join component, as follows:

SQL	SQL mode is disabled (join will be executed by MapForce (or, if applicable, by MapForce Server).
SQL	SQL mode is enabled (join will be executed by the database).

If the SQL button is missing, this means that SQL mode is not meaningful or not supported for the data that is being joined.

In certain cases, the SQL mode must be explicitly disabled (SQL), for example:

- When your mapping requires join conditions outside of the join component properties (that is, conditions defined on the mapping and connected to the `condition` item of the join component).
- When you want to join tables from different databases. Use a standard (non-SQL) join if you need to join tables from different databases.

Changing the Join mode

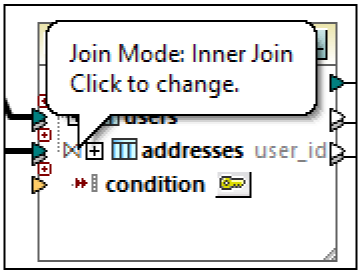
When the Join component is in **SQL mode** SQL, you can join database tables or views in one of the following ways:

- INNER JOIN - Only records which satisfy the condition in both input sets are returned by the Join component.
- LEFT OUTER JOIN - The Join component includes all records from the "leftmost" table (in MapForce, this is the topmost table of a Join component), plus those records from the subsequently joined table that satisfy the join condition.

To view the join mode of a table or view on the Join component, observe the icon shown in front of the joined table or view. One of the following icons can be shown for any joined table or view except the first one:

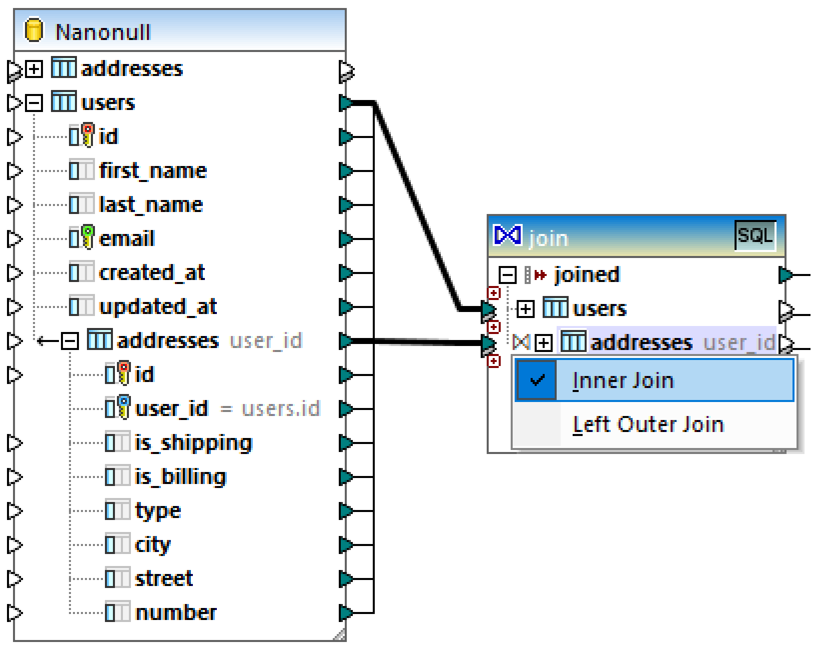
- Inner Join ⋈
- Left Join ⋈

To display a tooltip with details about the join, move the cursor over the icon:



To change the join mode, do one of the following:

- Click the **Inner Join** or **Left Join** icon in front of each joined table or view, and select **Inner Join** or **Left Outer Join** from the context menu.
- Right-click the second (or third, fourth, etc) joined table or view on the Join component, and select **Join Type | Inner Join**, or **Join Type | Left Outer Join** from the context menu.



Note the following:

- If you changed the join mode to LEFT OUTER JOIN, then the upper table or view represents the "left" side of the join.
- Changing the join mode affects the data returned by the join component in the same way that INNER JOIN or LEFT JOIN affects the result of a SQL query in a database.

Alias names

It is often the case that joined database tables or views contain identical field names in both joined structures. When SQL mode is enabled, such items appear on the component prefixed by the keyword "AS". For example, if two joined tables contain an "id" field, this field appears as "id" on the first joined table and as "id

AS id2" on the second joined table. Joined tables can also produce alias names, for example, if the same table is joined to itself.

The alias field or table names are important if you need to refer to them subsequently on a mapping. For example, imagine a case when you want to filter or sort the result of the join. To achieve this, the output of the join component can be connected to a SQL WHERE/ORDER component, where you would enter the SQL WHERE and ORDER BY clauses.

To refer to a field from the WHERE clause, write the table name, followed by a dot (.) character, followed by the field name. To refer to a table alias, use the alias name as it appears on the Join component. In the ORDER BY clause, you can either use the same technique (`table.field`), or write just the alias field name (the name that appears after "AS").

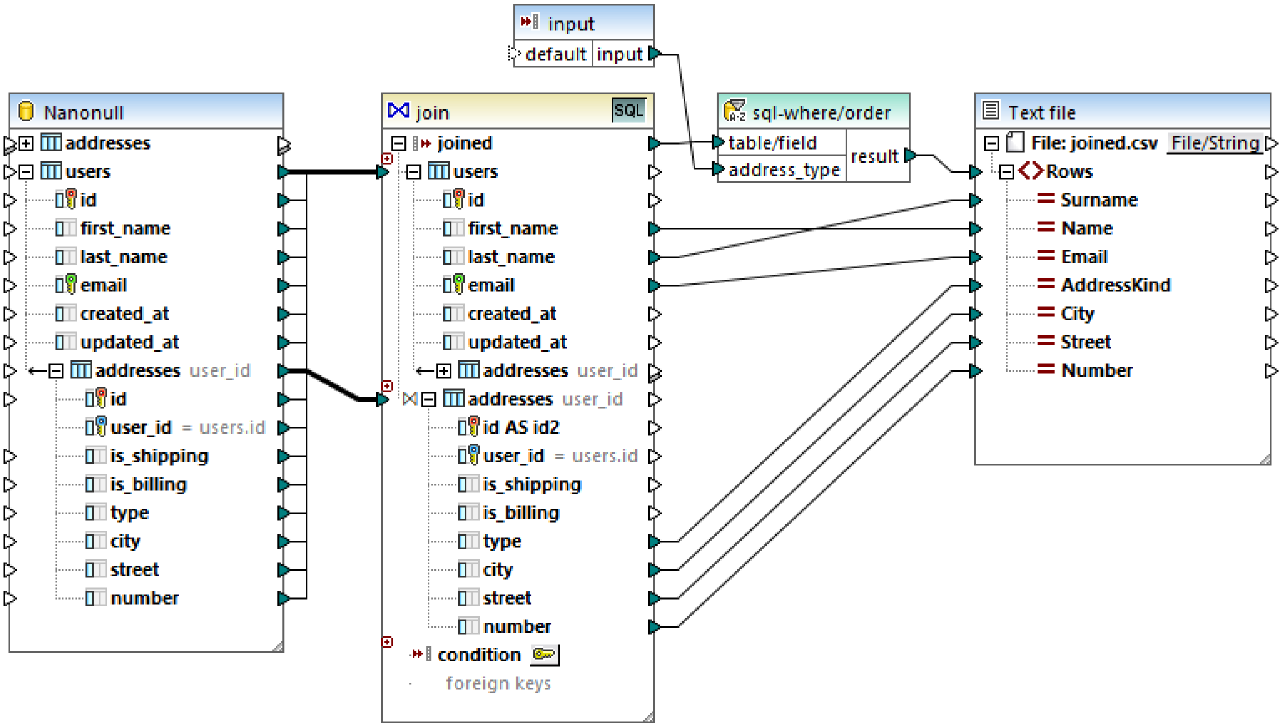
For an example mapping which uses SQL WHERE/ORDER clauses, see Example: Join Tables in SQL Mode (337).

Note: SQL WHERE/ORDER components are not allowed between a database table and the join component; they can be added only after (but not before) a join component. For more information about SQL WHERE/ORDER components, see Filtering and Sorting Database Data (SQL WHERE/ORDER) (350).

4.2.4.2 Example: Join Tables in SQL Mode

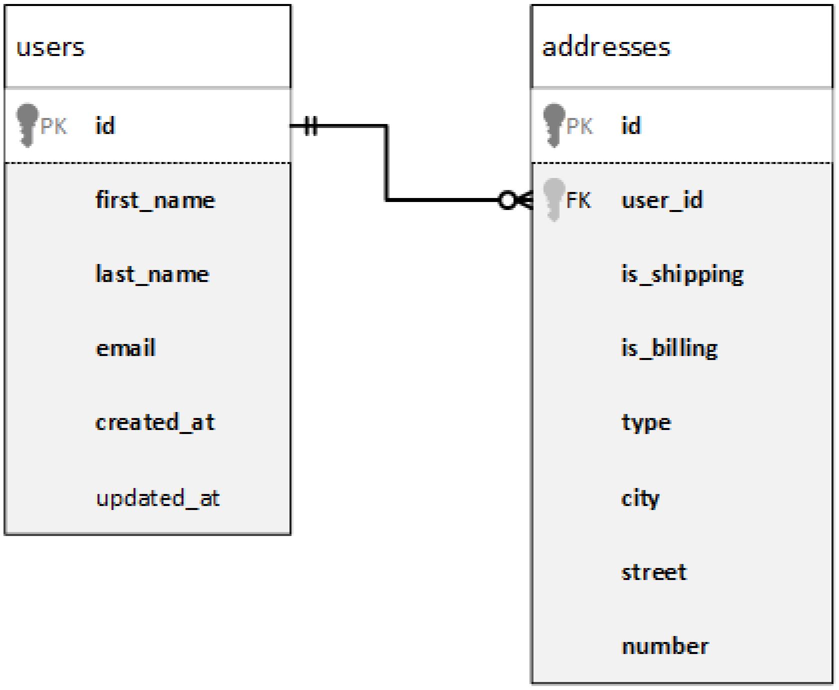
This example illustrates how to join data from two database tables, using a MapForce join component. The join operation is performed in SQL mode, as described in Joins in SQL Mode (334). Note that joining three or more tables works in a very similar way, see also Example: Create CSV Report from Multiple Tables (346).

The example is accompanied by a mapping sample which is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\JoinDatabaseTables.mfd**.



JoinDatabaseTables.mfd

The purpose of the mapping above is to combine data from two source database tables into a single target CSV file. As illustrated in the database diagram below, the first table (`users`) stores people's addresses and the second table (`addresses`) stores people names and email addresses. The two tables are linked by a common field (`id` in `users` corresponds to `user_id` in `addresses`). In database terminology, this kind of relation is also known as a "foreign key relationship".



For convenience, the image below illustrates the actual data in both tables.

users

id	first_name	last_name	email	created_at	updated_at
1	Marquita	Bailey	m.bailey@nanonull.com	2016-12-29 14:37:14	[NULL]
2	Sharda	Junker	s.junker@nanonull.com	2016-12-29 14:37:14	[NULL]
3	Totie	Rea	t.rea@nanonull.com	2016-12-29 14:37:14	[NULL]
4	Tobie	Hughey	t.hughey@nanonull.com	2016-12-29 14:37:14	[NULL]
5	Eadith	Lafreniere	e.lafreniere@nanonull.com	2016-12-29 14:37:14	[NULL]
6	Yehudi	Sponga	y.sponga@nanonull.com	2016-12-29 14:37:14	[NULL]
7	Laurianne	Huisman	l.huisman@nanonull.com	2016-12-29 14:37:14	[NULL]
8	Fred	Weinstein	f.weinstein@nanonull.com	2016-12-29 14:37:14	[NULL]
9	Mia	Dahill			
10	June	Leiker			
11	Benedick	Kocyk			
12	Andrej	Hildebrand			
13	Ariel	Phelan			
14	Matthaeus	Hulick			
15	Lotta	Mendes			
16	Jessey	Decelles			
17	Hilda	Lees			
18	Mark	Marzolla			
19	Dannie	Vignola			
20	Lanita	Krysiak			

addresses

id	user_id	is_shipping	is_billing	type	city	street	number
1	1	1	0	work	Bridgedell	Maple Lane	1
2	1	0	1	home	Bridgedell	Olive Street	6
3	3	1	1	home	Roseford	Evergreen Lane	34
4	4	1	1	work	Beardale	Route 44	9
5	6	1	1	home	Johnson City	Franklin Avenue	11
6	7	1	1	home	North Kingstown	Beach Alley	5
7	8	1	1	home	Merrowmeadow	Freybeach Street	85
8	10	1	1	work	Barrowedge	Penn Street	8
9	12	1	1	home	Elfville	Creek Road	3
10	13	1	1	home	Roseford	Bowman Ave.	853
11	14	1	1	work	Beardale	Iroquois Street	98
12	17	1	1	home	Bridgedell	Smith Road	7
13	18	1	0	home	Roseford	Wood Street	7
14	18	0	1	work	Johnson City	Thorne Lane	9677
15	20	1	1	home	Mechanicsville	Vine Street	9065

Each user record in the `users` table can have zero or more addresses in the `addresses` table. For example, a user may have one address of type "home", or two addresses (one of type "home" and another of type "work"), or no address at all.

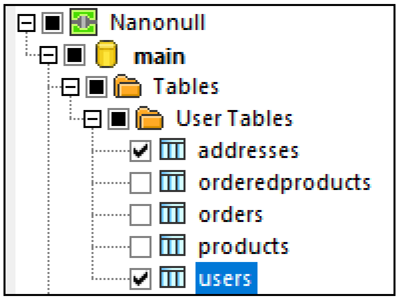
The goal of the mapping is to retrieve full data (name, surname, email, city, street, number) of all users that have at least one address in the addresses table. It should also be possible to easily retrieve only addresses of a specific kind (for example, only home addresses, or only work addresses). The kind of addresses to retrieve ("home" or "work") should be supplied as a parameter to the mapping. The retrieved people records must be sorted alphabetically by last name.

The mapping requirement will be accomplished with the help of a Join component, as illustrated in the steps below.

Note: Using a Join component is not the only way to join database tables or views. Joins applicable to databases can also be performed by using SQL SELECT statements, see SQL SELECT Statements as Virtual Tables 357. A major difference between SQL SELECT statements and Join components is that the former are written by hand so they might provide more flexibility. Join components are a simpler alternative if you do not feel comfortable writing SQL statements by hand.

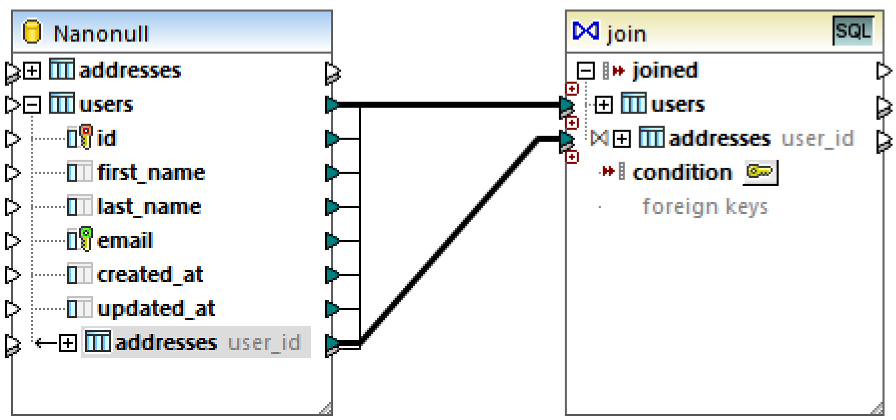
Step 1: Add the source database

1. On the **Insert** menu, click **Database**. (Alternatively, click the **Insert Database** toolbar button).
2. Select "SQLite" as database kind, and click **Next**.
3. Browse for the **Nanonull.sqlite** file available in the folder: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**, and click **Connect**.
4. When prompted, select the `addresses` and `users` tables.



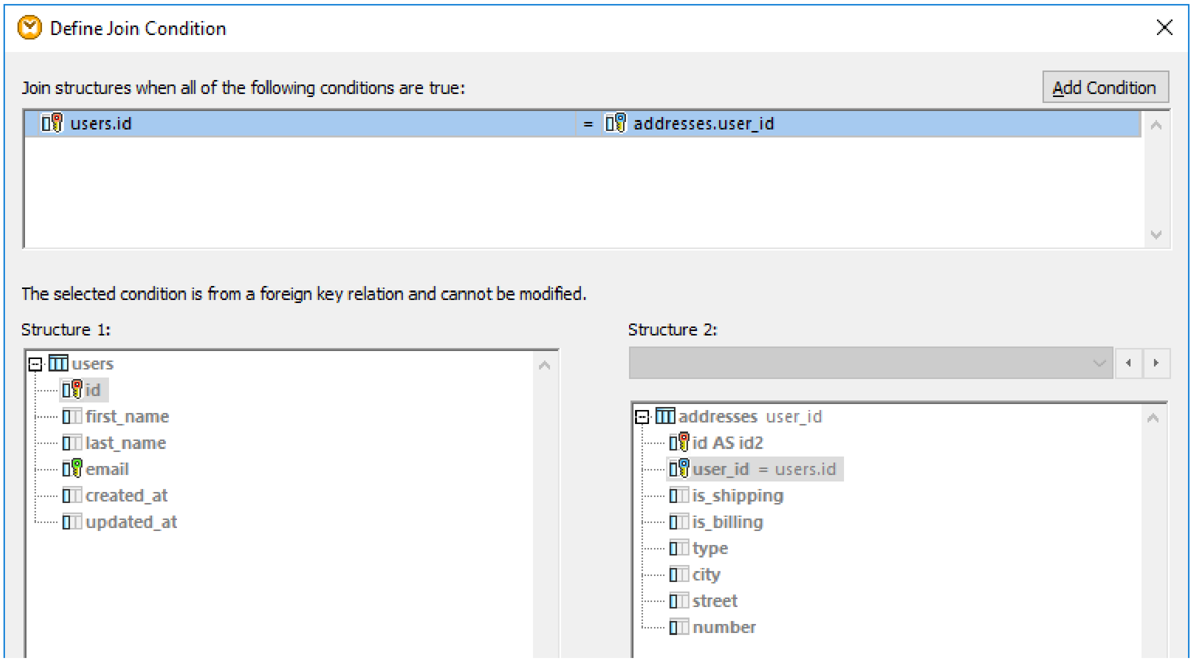
Step 2: Add the join component

1. On the **Insert** menu, click **Join**. (Alternatively, click the **Join** toolbar button).
2. Draw a connection from the `users` table to the first input of the join component.
3. Expand the `users` table and draw a connection from the `addresses` table (child of `users`) to the second input of join component. The button enables you to add more tables if necessary; however, in this example, only two tables are being joined.



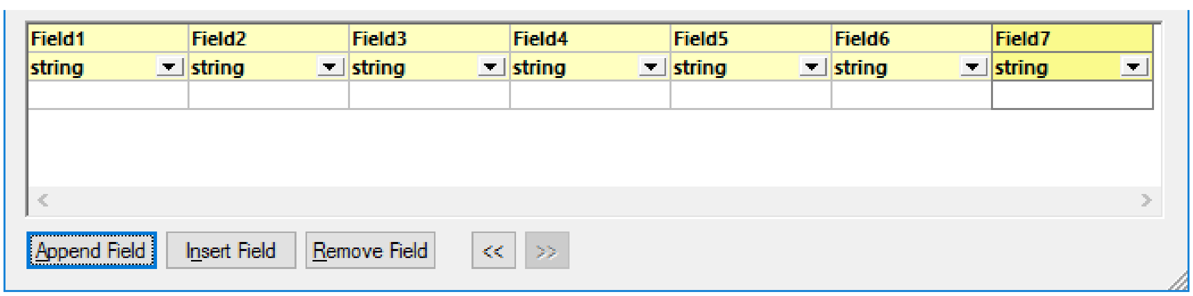
Note: It is also possible to add the connection directly from the `addresses` table (the one which is not child of `users`); however, in this case, the join conditions would have to be defined manually, as described in Adding Join Conditions 862. For the purpose of this example, make sure to create the connections as shown above. This ensures the required join condition is created automatically.

4. Click the **Define Join Condition** button available on the join component. Notice that the join condition has been created automatically (`users.id = addresses.user_id`).

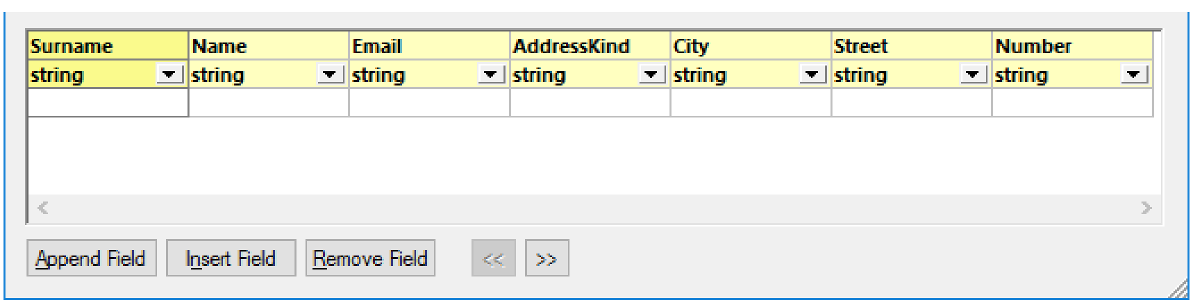


Step 3: Add the target CSV component

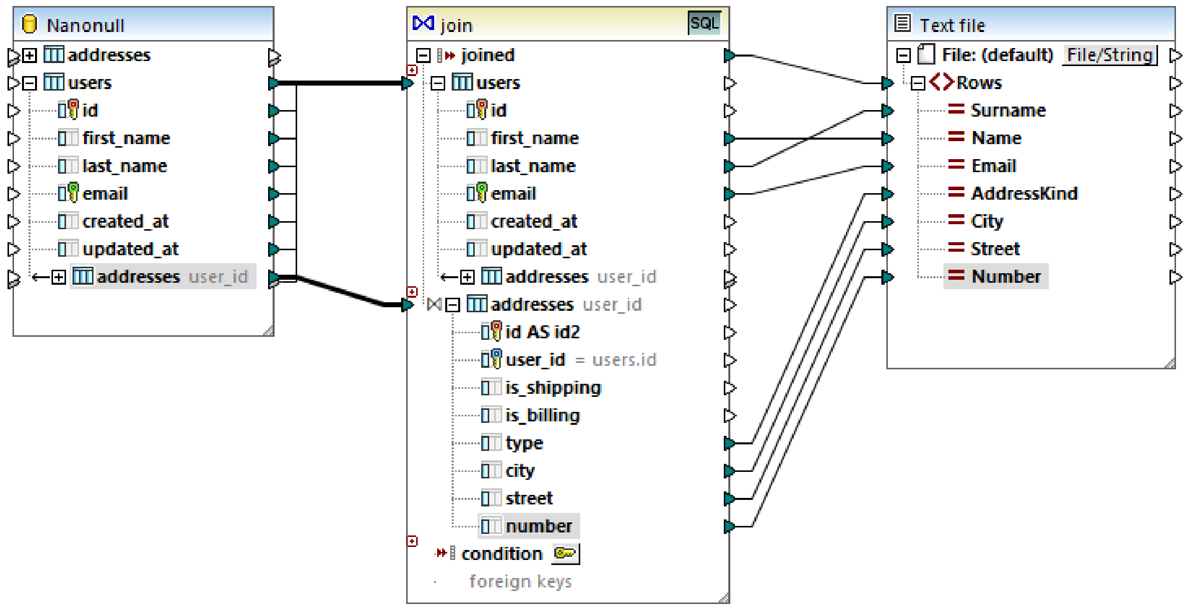
1. On the **Insert** menu, click **Text File**. (Alternatively, click the **Insert Text File** toolbar button).
2. When prompted to choose a text processing mode, select **Use simple processing for standard CSV...** .
3. Click **Append Field** several times to create seven CSV fields. Leave all other settings as is.



4. Double-click the title cell of each field to give it a descriptive name (this will make your mapping easier to read).

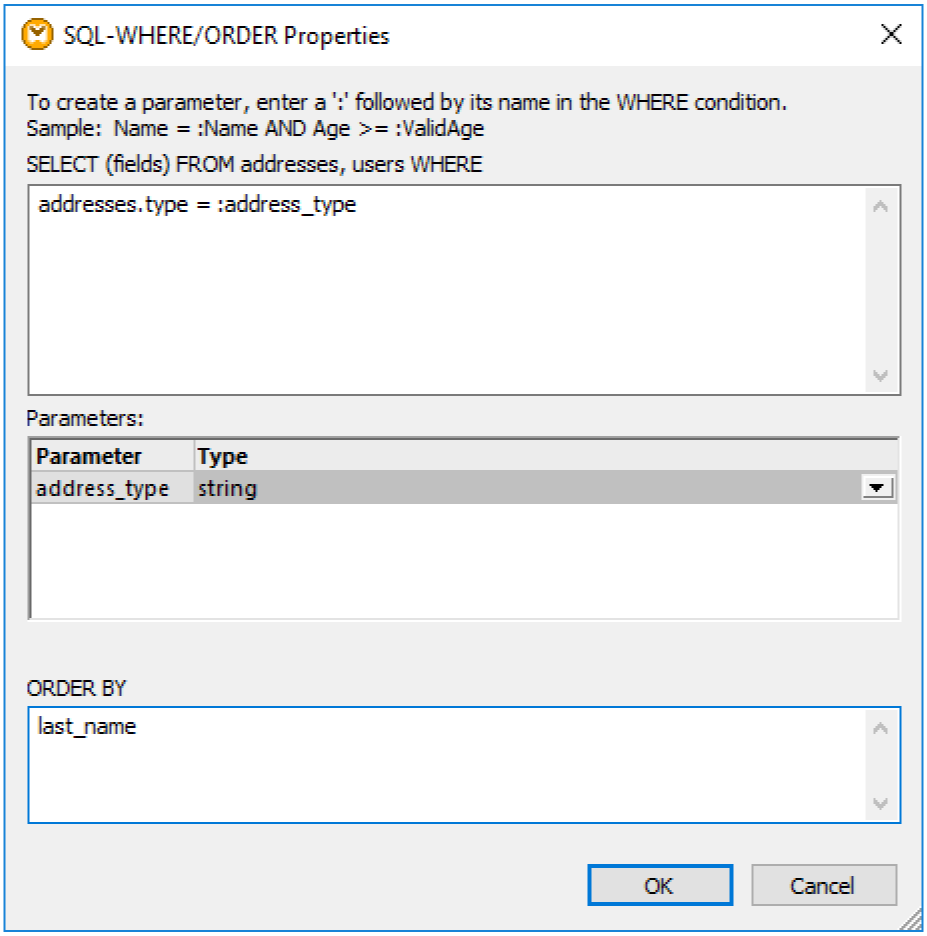


5. Draw the mapping connections between the Join component and the CSV component as shown below. The connection between the `joined` item of the join component and the `Rows` item of the target component means "create as many records (rows) in the target as there are records that meet the join condition".

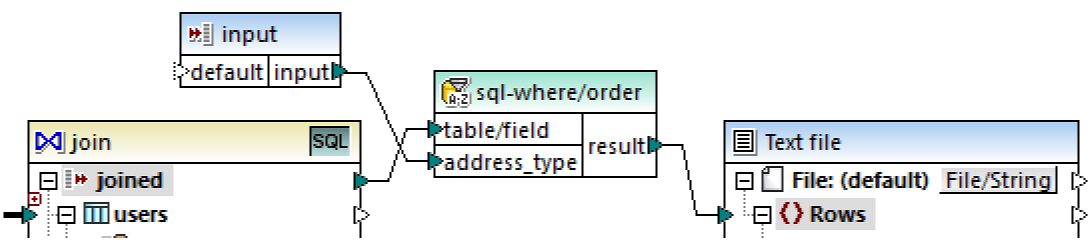


Step 4: Add the SQL WHERE/ORDER condition and input parameter

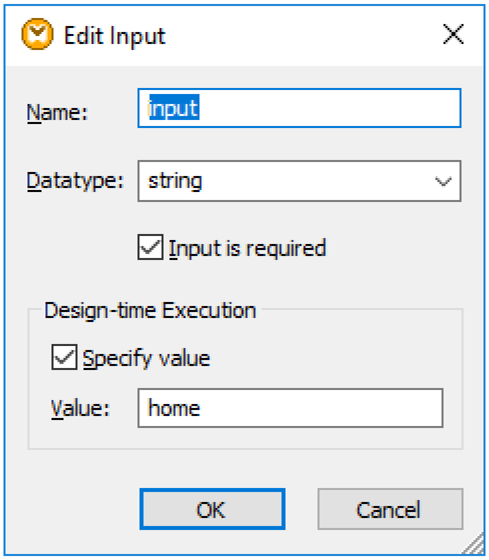
1. Right-click the connection between the `joined` item of the Join component and the `Rows` item of the target CSV component, and select **Insert SQL-WHERE/ORDER**.
2. Enter the WHERE and ORDER BY clauses as shown below.



3. On the mapping, add an input component (using the **Insert | Insert Input** menu command) and connect its output to the `address_type` parameter created in the previous step.



4. Double-click the input component and configure it as shown below. A design-time value is required (in this case, "home") to preview the mapping output in MapForce. If you want the preview to retrieve work addresses, replace this value with "work".



The mapping explained

The join condition automatically created in step 2 ensures that only records which satisfy the join condition `users.id = addresses.user_id` are copied to the target. The join condition was added automatically because the two tables are bound by a foreign key relationship and the mapping connections were drawn accordingly. For more information about table relationships, see Handling Database Relationships [257]. Because this example has made use of the already existing table relationships, you did not have to define any join conditions manually. For an example that shows you how to define join conditions manually, see Example: Create CSV Report from Multiple Tables [346].

The two source tables are from the same database and from the same component, so this join benefits from the SQL mode. Since SQL mode is enabled, the join operation is undertaken by the database, not by MapForce. In other words, an INNER JOIN statement is generated internally by MapForce and sent to the database for execution. The type of the join (INNER JOIN) is indicated by the **Inner Join** icon in front of the **addresses** table on the Join component. You can also change the join type to LEFT OUTER JOIN, as described in Changing the Join Mode [335]. Note, however, that changing the join mode does not affect the output of this example.

The SQL WHERE/ORDER component added in step 4 enables filtering (to retrieve either home or work addresses) and sorting the recordset. Notice that the WHERE clause created a parameter **`:address_type`** of type `string`. This parameter makes it possible to supply the address kind (home or work) from the mapping. For more information about SQL WHERE/ORDER, see Filtering and Sorting Database Data (SQL WHERE/ORDER) [350].

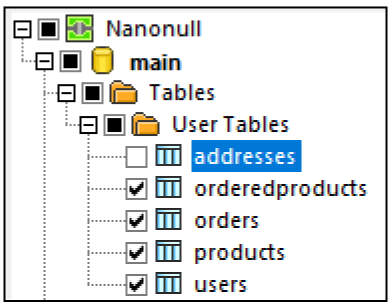
Finally, the input component makes it possible to supply the actual parameter value when the mapping runs. Note that, when the mapping runs outside MapForce (for example, when it is executed by MapForce Server on a different machine), the input must be supplied at mapping runtime as a command-line parameter, so the design-time value mentioned above is ignored. For more information, see Supplying Parameters to the Mapping [833].

4.2.4.3 Example: Create CSV Report from Multiple Tables

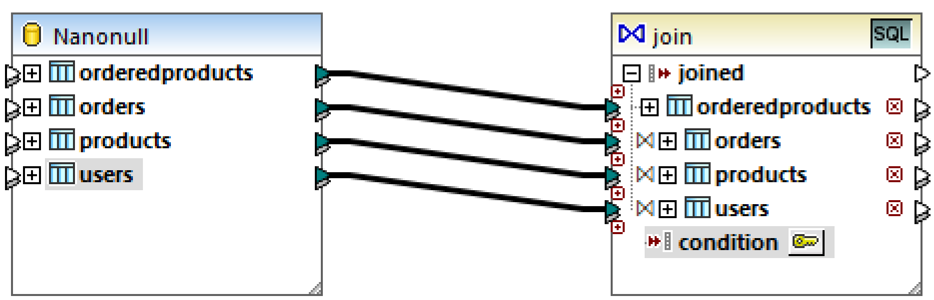
This example illustrates how to join multiple database tables for the purpose of extracting data into a single report in CSV format. The database used in this example is called **Nanonull.sqlite** and is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**. This database stores information about a fictitious business (which includes orders, products, users and their addresses). As is typically the case with relational databases, the information is normalized and spread across multiple tables. For example, the `users` table stores user personal data (which includes first name, last name, and email). The database also stores information about products ordered by users, in two different tables: `orders` (which includes the unique ID of the order, and the time when it took place) and `orderedproducts` (which includes a list of products ordered, and their quantity). Furthermore, the names of the products themselves is stored in a separate table called `products`.

The goal of the example is to produce a report based on data extracted from various tables, so as to make it clear who ordered certain products, when, and in which quantity. To achieve the mapping goal, follow the steps below:

1. On the **Insert** menu, click **Database**.
2. When prompted to select a database kind, click **SQLite**, and then click **Next**.
3. Browse for the **Nanonull.sqlite** database mentioned above, and click **Connect**.
4. When prompted, select the tables `orderedproducts`, `orders`, `products`, and `users`, and click **OK**.



5. Add a Join component to the mapping and create four nodes/rows items by clicking the **Add input** ( ) button.
6. Connect the four tables from the database component to the corresponding input items of the Join component.

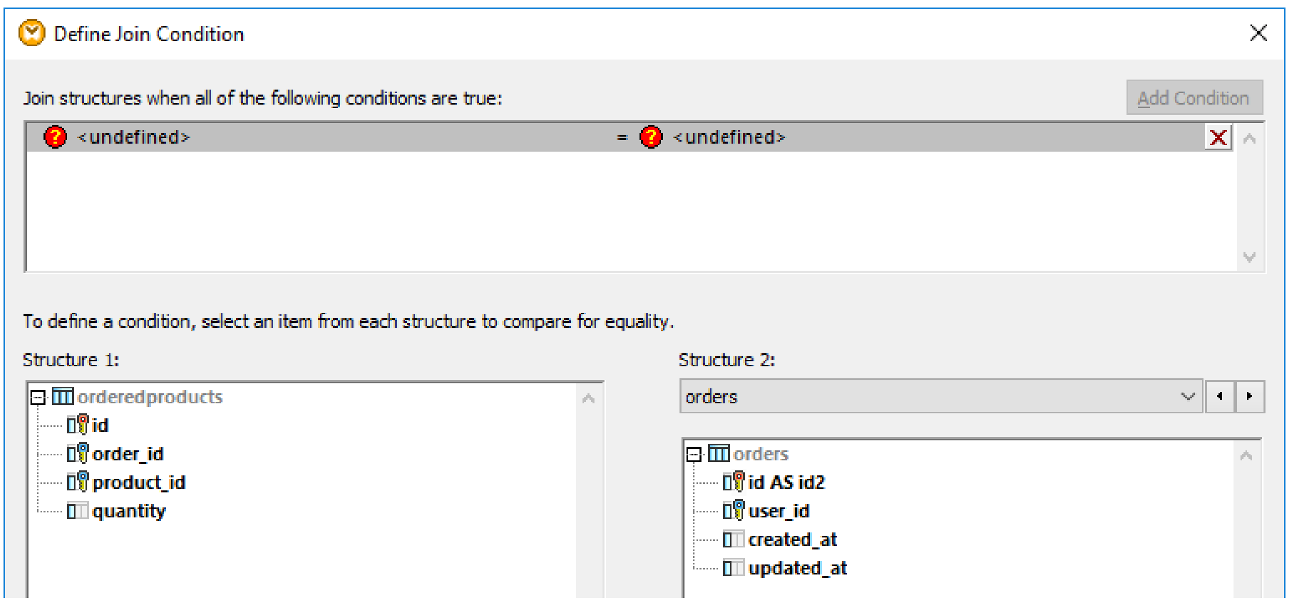


Note: In an alternative scenario, you could connect the table `orderedproducts` to the Join component, then the table `orders` (the one which is nested under it, not the one at the same level), and so on, so that all joined tables are nested under the same "root" table, see also Handling Database Relationships[257]. The mapping result would be the same if you joined tables this way. The difference is that in this example the join conditions are created manually, whereas in the alternative scenario the join conditions would be created automatically by MapForce. For an example of joining tables without having to define join conditions manually, see Example: Join Tables in SQL Mode 337. Another mapping where all joined tables are under the same "root" table is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\DB_Denormalize.mfd**.

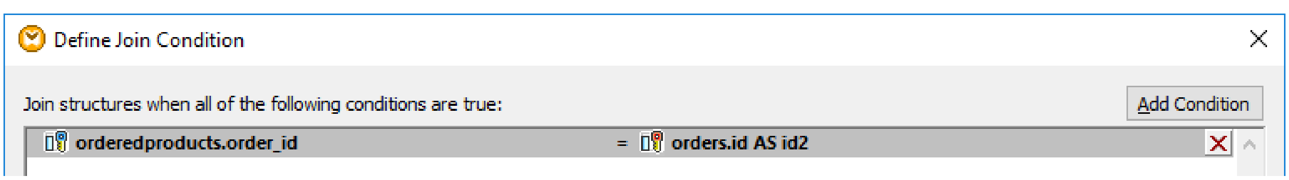
In this example, the tables connected to the Join component have the following order:

```
1. orderedproducts
2. orders
3. products
4. users
```

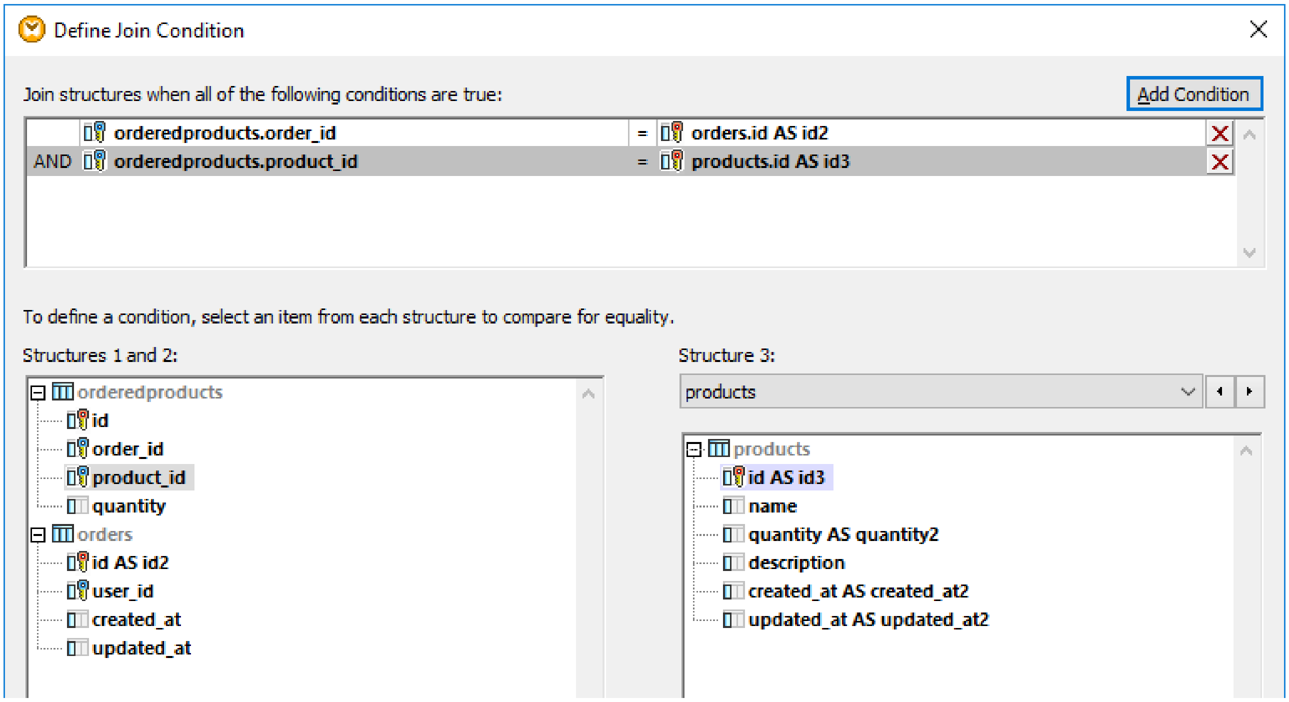
This order affects how the respective structures are displayed on the "Define Join Condition" dialog box, when you click the **Define Join Condition** ( ) button. Namely, the first table (`orderedproducts`) appears by default under **Structure 1**, and the table immediately after it (`orders`) appears under **Structure 2**.



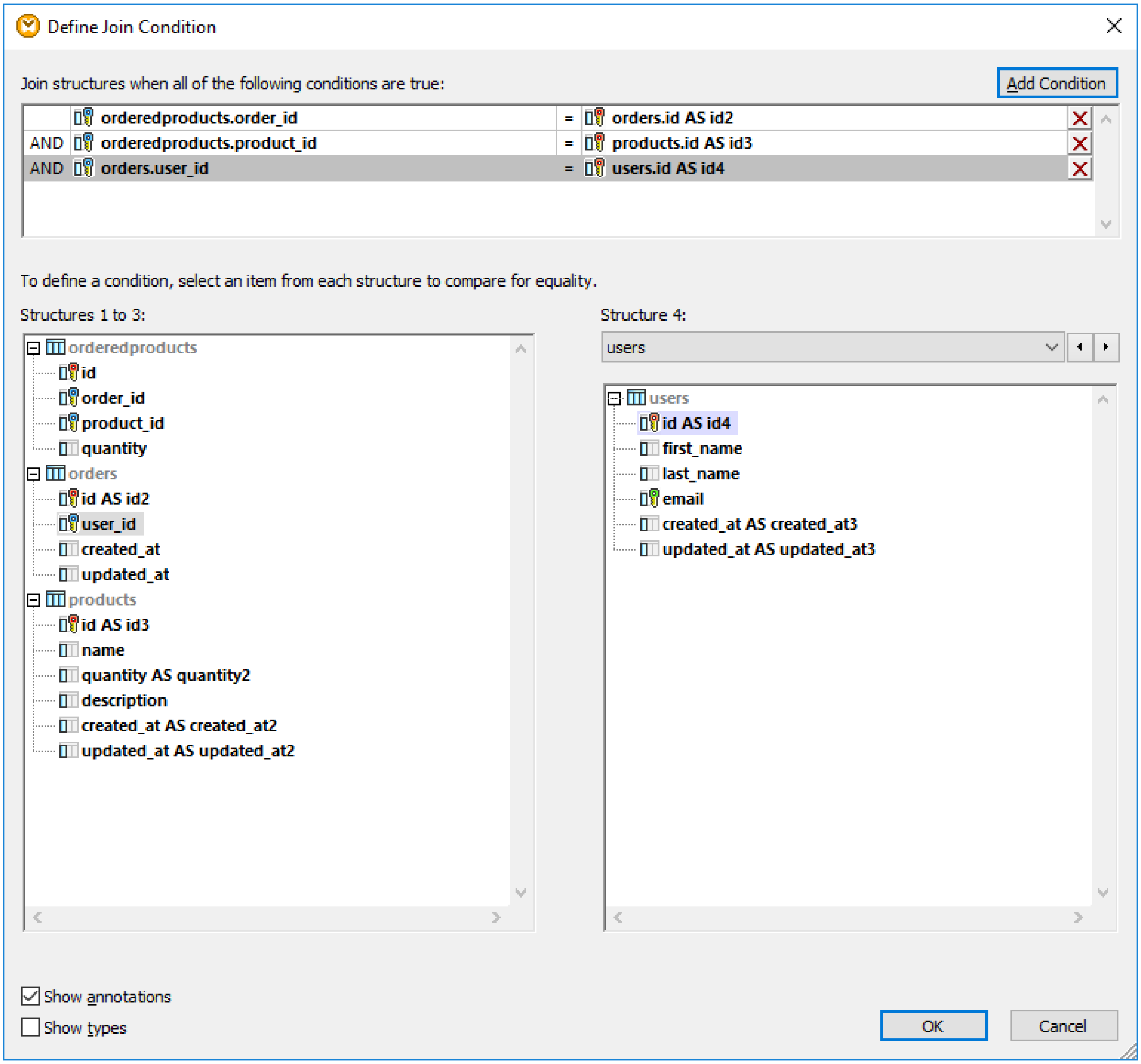
To define the first join condition, click the `order_id` item in the left pane and the `id` item in the right pane. Now the fields `orderedproducts.order_id` and `orders.id`. are paired:



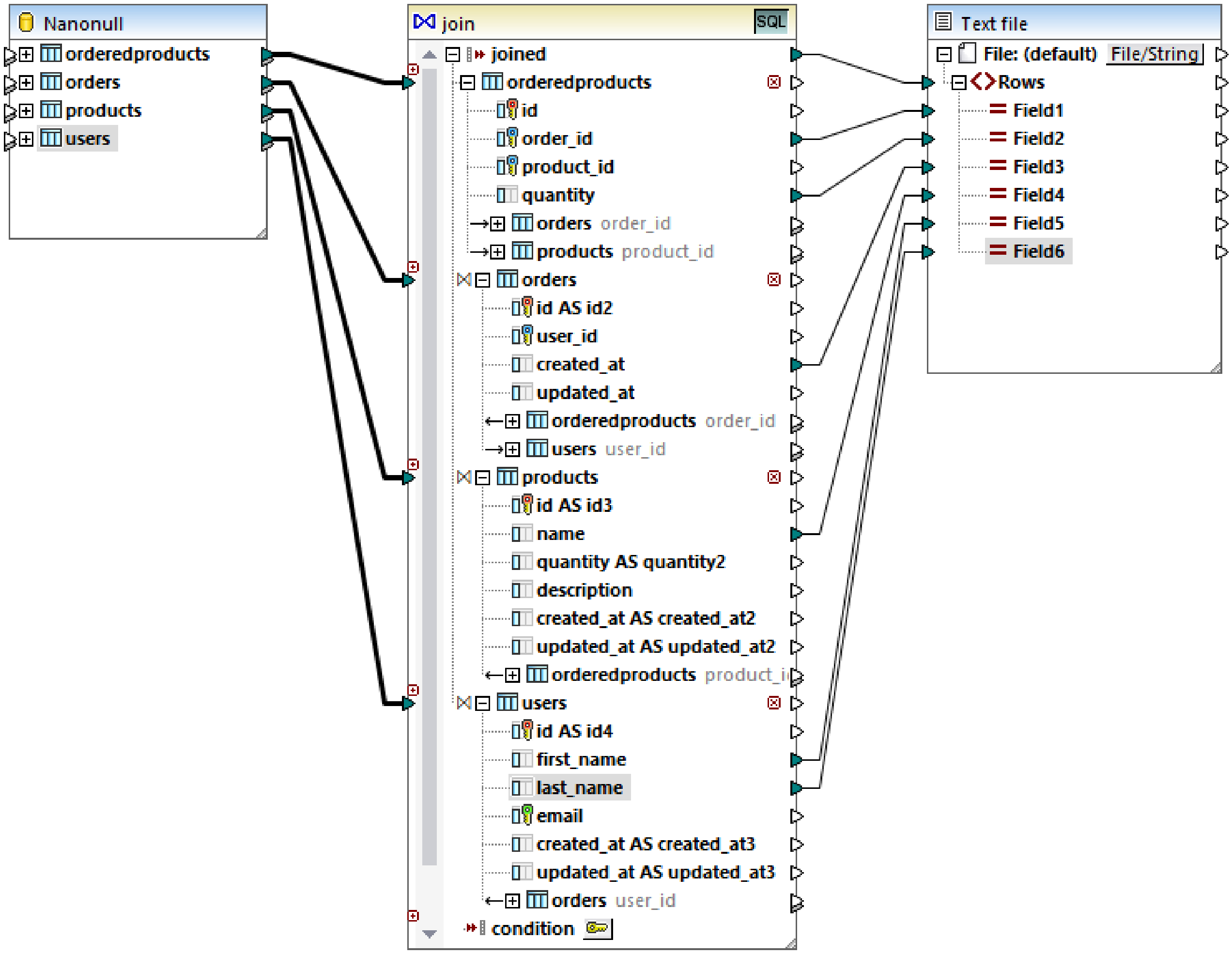
So far, only two tables have been joined. To define join conditions which involve a third table, select the desired table from the drop-down list available above the right pane. The left pane displays in this case all tables that occur *before* it on the Join component. For example, if you select `products` on the right side, then the left side displays `orderedproducts` and `orders` (since these tables occur before `products` on the Join component). You can now pair fields of table `products` with fields of tables preceding it (in this case, `orderedproducts.product_id` and `products.id`).



To join a fourth table (`users`), select the `users` table from the drop-down list. You can now pair the fields `orders.user_id` and `users.id`.



Now that all required join conditions have been defined, items of the Join component can be further mapped to a target component. To finish the mapping, add a CSV component (see CSV and Text Files 563), and connect items from the Join component to the target CSV component as illustrated below:



The mapping illustrated above produces a report (in CSV format) compiled from all four tables involved in the join, as follows:

- ID of the order (taken from the `orderedproducts` table)
- Quantity of ordered items (taken from the `orderedproducts` table)
- Time when the order took place (taken from the `orders` table)
- Name of the product ordered (taken from the `products` table)
- First name and last name of the user who ordered the product (taken from the `users` table).

All the tables in this example are joined using INNER JOIN mode. For information about changing the join mode to LEFT OUTER JOIN, see Changing the join mode 335.

4.2.5 Filter and Sort Database Data

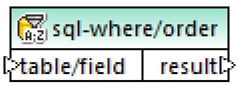
When you need to filter and sort database data, use the **SQL/NoSQL-WHERE/ORDER** component. This enables you to manually enter an SQL WHERE clause that filters data. Optionally, you can also specify an ORDER BY clause if you want to sort the record set by a particular database field, in ascending or descending order.

The **SQL/NoSQL-WHERE/ORDER** component must be connected to a table or field of a database mapping component. It is also possible to connect an **SQL/NoSQL-WHERE/ORDER** component with a **Join** component if you need to filter the joined set or records. For more information, see Joining Database Data333.

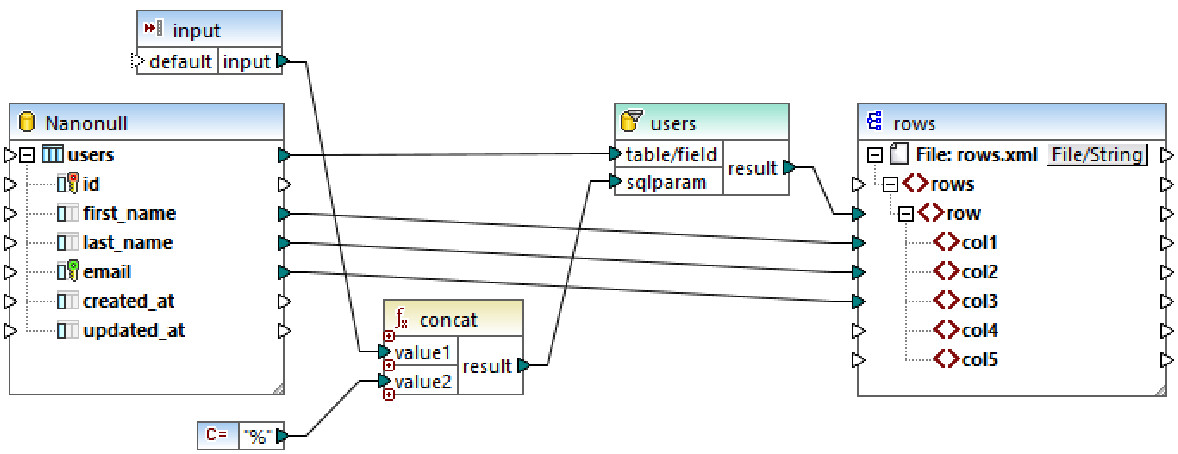
Add an SQL/NoSQL-WHERE/ORDER component

To add an **SQL/NoSQL-WHERE/ORDER** component to the mapping, follow the instructions below:

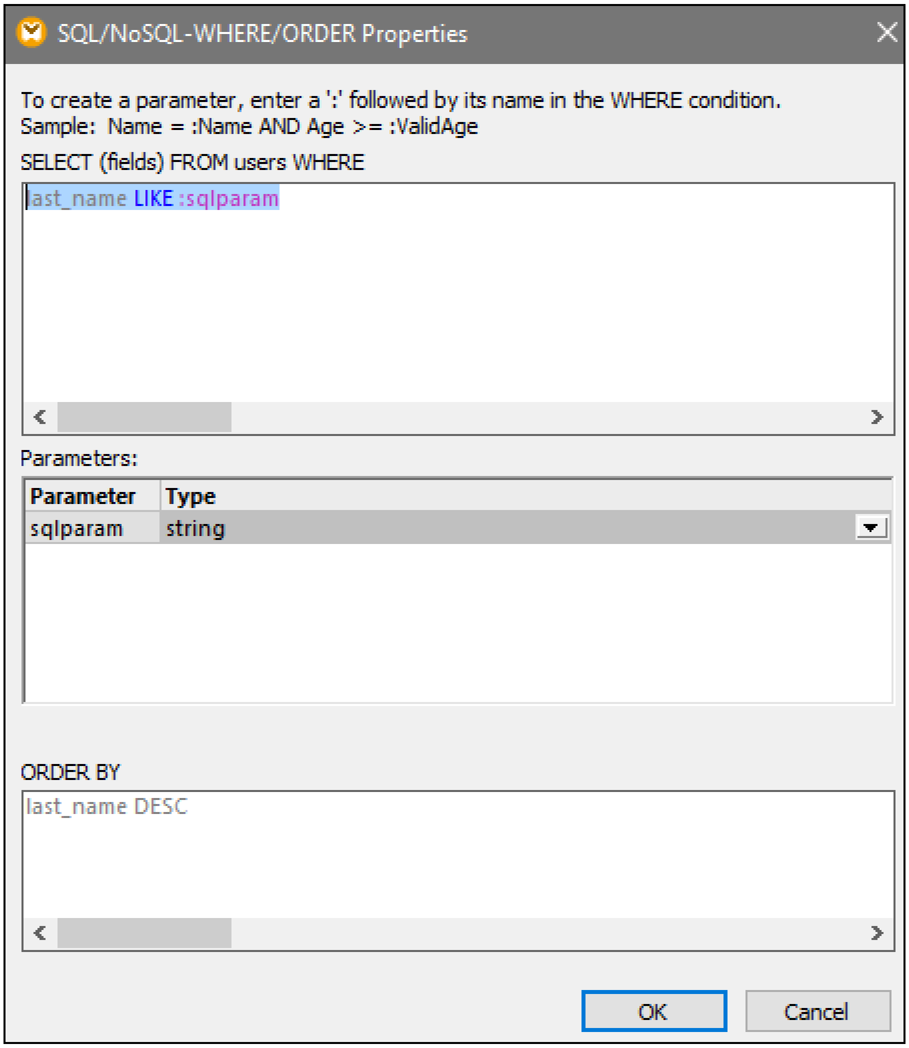
1. Go to the **Insert** menu and click **SQL/NoSQL-WHERE/ORDER**. By default, this component has the following structure:



2. Connect a source database table or field to the `table/field` item of the **SQL/NoSQL-WHERE/ORDER** component. You can find the sample mapping **`FilterDatabaseRecords.mfd`** (*see screenshot below*) in the following folder: **`<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\`**. In this mapping, the **SQL/NoSQL-WHERE/ORDER** component takes the data from the source table `users`, filters all its records and selects only those where the last name begins with the letter M (*see the explanation in the subsection below*).



3. Double-click the header of the **SQL/NoSQL-WHERE/ORDER** component. Alternatively, right-click it and select **Properties** from the context menu. This opens the dialog box **SQL/NoSQL-WHERE/ORDER Properties**.

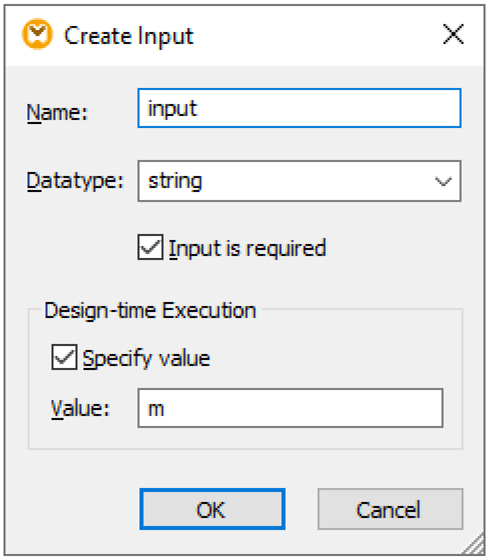


4. Type an SQL WHERE clause in the text box at the top. In our example, the SQL Where clause is as follows: **`last_name LIKE :sqlparam`**. Optionally, type an ORDER BY clause. The image above illustrates the WHERE and ORDER BY clauses defined in the **`FilterDatabaseRecords.mfd`** mapping (these settings are further explained below). For more examples, see [Creating WHERE and ORDER BY Clauses] [354].

Parameters in SQL/NoSQL-WHERE/ORDER components

The **SQL/NoSQL-WHERE/ORDER** component used in the mapping **`FilterDatabaseRecords.mfd`** has the following WHERE clause: **`last_name LIKE :sqlparam`**, where `last_name` refers to the name of the database field in the connected table; `LIKE` is an SQL operator; `:sqlparam` creates a parameter called `sqlparam` in the mapping.

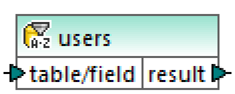
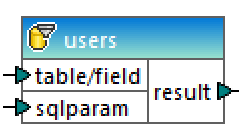
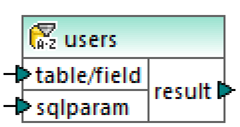
Parameters in the **SQL/NoSQL-WHERE/ORDER** component are optional. They are useful if you want to pass a value to the WHERE clause from the mapping. Without parameters, the WHERE clause above could have been written as follows: `Last LIKE "M%"`. This would retrieve all persons whose last name begins with the letter M. In order to make this query even more flexible, we have added a parameter instead of `"M%"`. This makes it possible to supply any other letter from the mapping: e.g., D, and thus retrieve people whose last name begins with D by changing a constant or a mapping input parameter. In the mapping above, the input letter comes from an input component called `input`. If you double-click the title bar of this component and open its properties, you will notice that `m` is given as a design-time execution value (*see screenshot below*).



In the mapping, the SQL wildcard character % is provided by a constant. This wildcard character is then concatenated with the parameter value with the help of the **`concat`** function. The advantage is that you do not have to type SQL wildcards in the command line if this mapping runs in another environment (e.g., MapForce Server).

Appearance of SQL/NoSQL-WHERE/ORDER components

SQL/NoSQL WHERE/ORDER components change their appearance depending on the settings defined in them. This way you can quickly view directly from the mapping what the **SQL/NoSQL WHERE/ORDER** component does (*see table below*).

	A WHERE clause has been defined.
	A WHERE clause with a parameter has been defined. The parameter name is visible under the `table/field` item.
	A WHERE clause with a parameter has been defined. Additionally, an ORDER BY clause has been defined. The sorting is indicated by the A-Z sort icon.

If you place the mouse cursor over the **SQL/NoSQL WHERE/ORDER** header, you will see a tooltip displaying the various clauses that have been defined.

4.2.5.1 Creating WHERE and ORDER BY Clauses

After an **SQL/NoSQL-WHERE/ORDER** component is added to the mapping, it needs a WHERE condition (clause) through which you specify how exactly you want to filter the data connected to the component. The WHERE condition must be entered in the dialog box **SQL/NoSQL-WHERE/ORDER Properties** (*see previous section*).

Writing a WHERE condition from MapForce is similar to writing the same SQL clause outside MapForce. Use the syntax applicable to the SQL dialect of the corresponding database. For example, you can use operators, wildcards, as well as sub-selects or aggregate functions. To create parameters that you can pass from the mapping, enter a semi-colon character (:) followed by the parameter name.

Note: When you finish writing the WHERE clause and click OK, MapForce validates the integrity of the final SQL statement. A dialog box prompts you if there are syntax errors.

The table below lists some typical operators that can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal
<	Less than
>	Greater than
>=	Greater than/equal
<=	Less than/equal
`IN`	Retrieves a known value of a column
`LIKE`	Searches for a specific pattern
`BETWEEN`	Searches between a range

Use the `%` (percentage) wildcard to represent any number of characters in a pattern. For example, to retrieve all records ending in "r" from a field called `lastname`, use the following expression:

```
lastname = "%r"
```

When querying databases that support storing and querying of XML database data (for example, IBM DB2, Oracle, SQL Server), you can use XML functions and keywords applicable to that particular database, for example:

```
xmlexists('$c/Client/Address[zip>"55116"]' passing USER.CLIENTS.CONTACTINFO AS "c")
```

See also Example: Extracting Data from IBM DB2 XML Type Columns[373].

Optionally, if you want to sort the retrieved recordset by a particular field, add an `ORDER BY` clause in the corresponding text box of the dialog box **SQL/NoSQL-WHERE/ORDER Properties**. To sort by multiple fields, separate the field names by commas. To change the sort order, use the `ASC` and `DESC` keywords. For example, the following `ORDER BY` clause retrieves records ordered by `lastname`, and then by `firstname`, in descending order:

```
lastname, firstname DESC
```

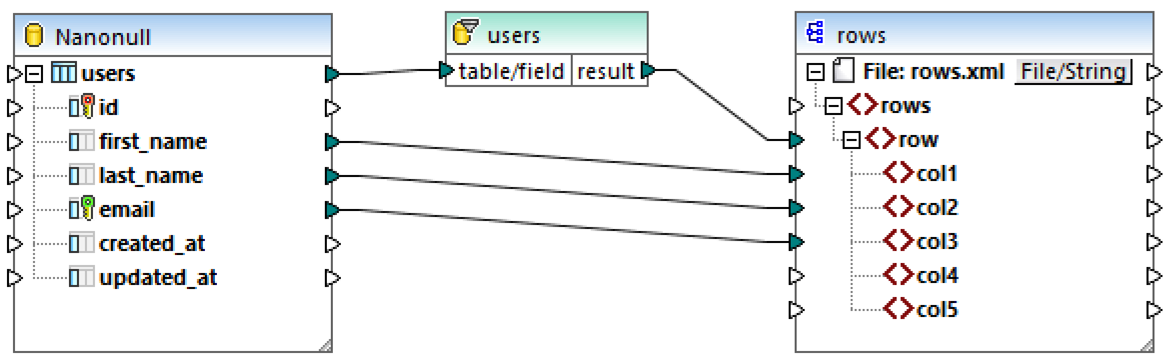
Example 1

The following WHERE condition is attached to the `users` table of the **`Nanonull.sqlite`** database component. It retrieves those records where `last_name` is greater than the letter M. In other words, it retrieves all names starting from user called Marzolla onwards.

```
last_name > "M"
```

Note how the connections are placed:

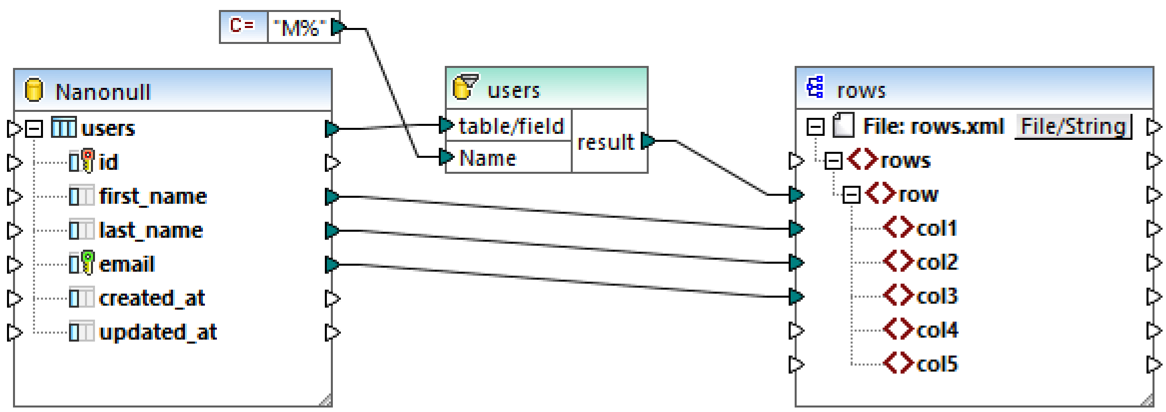
- The connection to `table/field` originates in the table that you want to query (`users` in this case).
- The `result` output is connected to a parent item of the fields that are queried/filtered (in this case the `row` item).



Example 2

The following WHERE condition creates a parameter `param` which then appears in the **SQL/NoSQL-WHERE/ORDER** component in the mapping.

```
last_name LIKE :param
```

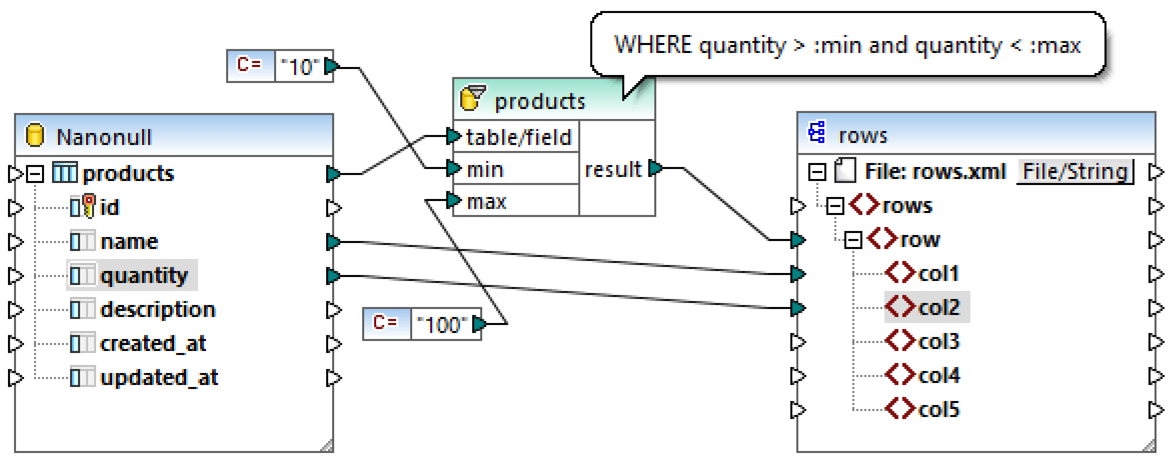


The constant component `%M` supplies the value of `param`. The wildcard % denotes any number of characters. This causes the mapping to search for a pattern in the column `last_name` (all last names starting with the letter M).

Example 3

The following WHERE condition creates two parameters, `min` and `max`, to which the current values of `quantity` are compared. The min and max values are supplied by two constant components from the mapping.

```
quantity > :min and quantity < :max
```



The WHERE condition in this example could also be written using the BETWEEN operator:

```
quantity BETWEEN :min and :max
```

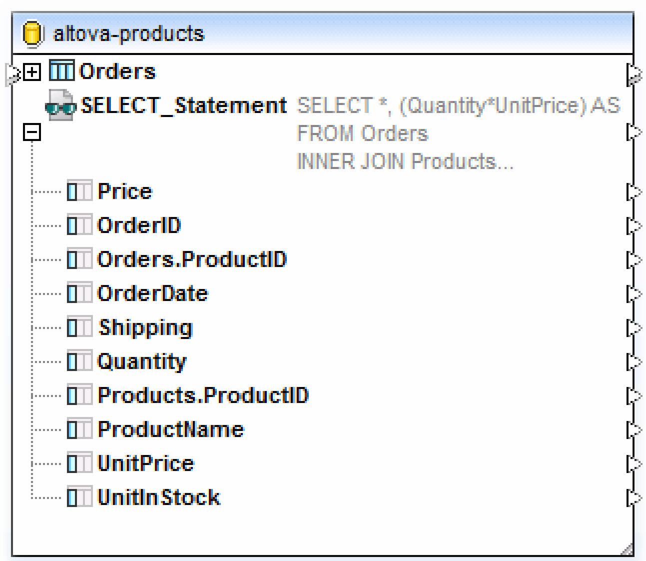
4.2.6 SQL SELECT Statements as Virtual Tables

MapForce supports the creation of SQL SELECT statements with parameters in database components. These are table-like structures that contain the fields of the result set generated by the SELECT statement. These structures can then be used as a mapping data source, like any table or view defined in the database.

- When using Inner/Outer **joins** in the SELECT statement, fields of all tables are included in the component.
- Expressions with correlation names (using the SQL "AS" keyword) also appear as a mappable items in the component.
- Database views can also be used in the FROM clause.
- SELECT statements can contain parameters which use the same syntax as the SQL WHERE/ORDER 350 component.

> SQL SELECT statements are supported in C++, C#, Java, and BUILT-IN languages, as long as they don't contain input parameters. SQL SELECT statements with input parameters are supported only in the BUILT-IN transformation language.

Once the SELECT statement has been added to a database component, the fields returned by it are available for mapping, for example:



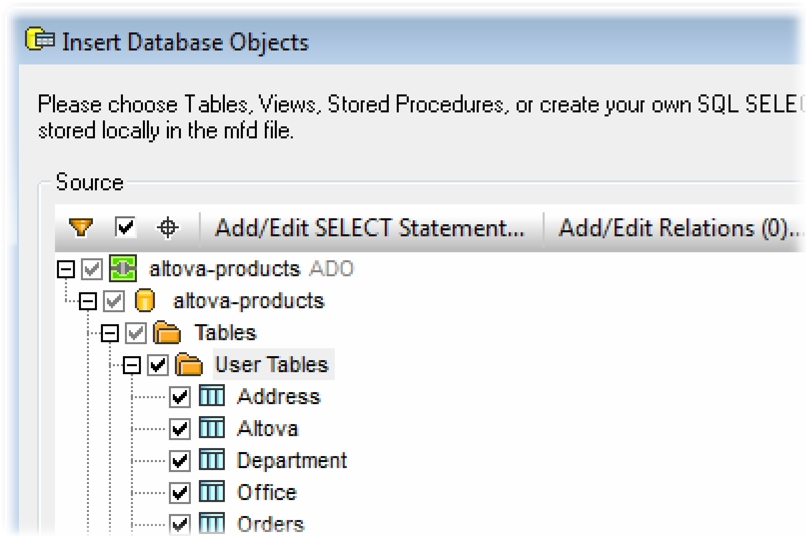
The number of visible lines of the SELECT statement is configurable. To define the number of lines you want to see on the component, select the menu command **Tools | Options**, click the **General** tab and enter the number of lines in the Mapping View group.

4.2.6.1 Creating SELECT Statements

You can create SELECT statements on any mapping which contains a database component. If your mapping does not contain a database yet, add a database first (see Connecting to a Database 163). For the scope of this example, select the menu command **Insert | Insert Database** and follow the wizard steps to connect to the **altova-products.mdb** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder.

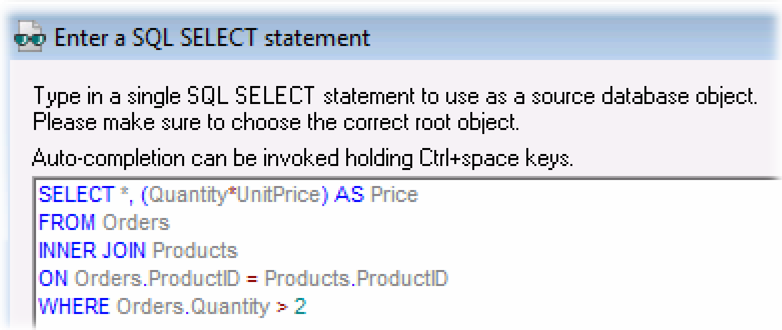
To create a SELECT statement:

1. Right-click the title of the database component, and select **Add/Remove/Edit Database Objects**. (As an alternative, select the database component, and then select the menu command **Component | Add/Remove/Edit Database Objects**).

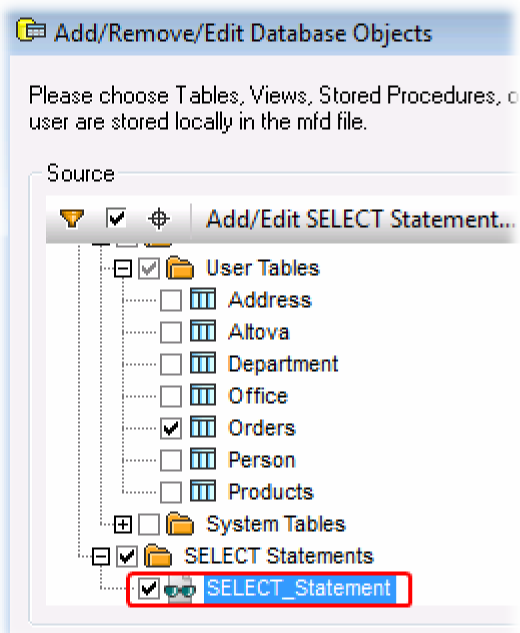


2. Do one of the following:
   - o To generate the SELECT statement from an existing table, right-click any table and select **Generate and add an SQL statement** from the context menu. You will be able to edit the generated statement afterwards.
   - o To write a custom SELECT statement, click the **Add/Edit SELECT Statement** button.
3. Edit or create the statement as required. For example, the SELECT statement below is valid for the **altova-products.mdb** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. The **Price** field is the product of the two fields, **Quantity** and **UnitPrice**, and is declared as a correlation name (`AS Price`).

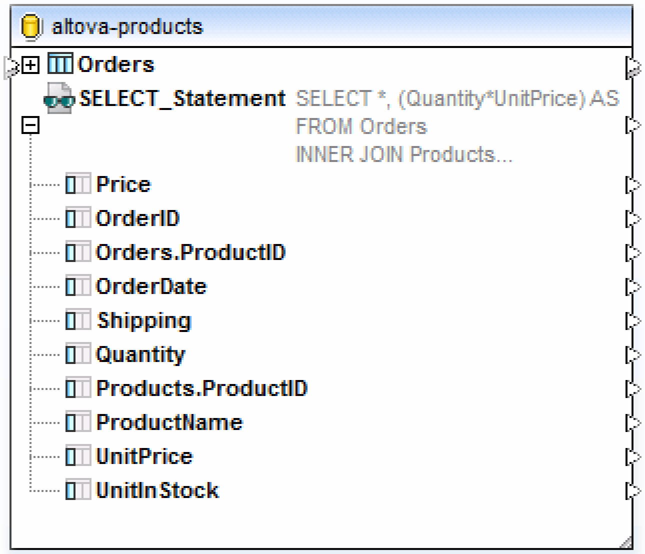
```
SELECT *, (Quantity*UnitPrice) AS Price
From Orders
INNER JOIN Products
ON Orders.ProductID = Products.ProductID
Where Orders.Quantity > 2
```



4. Click **Add SELECT Statement**. Notice that the SELECT statement is now visible as a database object, similar to how tables, views, and procedures are visible.



5. Click OK. The SELECT statement is also displayed on the database component, and you can map data from any of the fields returned by the SELECT query.



Important notes:

- All calculated expressions in the SELECT statement must have a unique correlation name (like "AS Price" in this example) to be available as a mappable item.
- If you connect to an Oracle or IBM DB2 database using JDBC, the `SELECT` statement must have no final semicolon.

To remove a previously added SELECT statement:

1. Right-click the title of the database component, and select **Add/Remove/Edit Database Objects**.
2. Right-click the SELECT statement you want to delete, and select **Remove Select Statement**.

4.2.6.2 Example: SELECT with Parameters

This example shows you how to create a MapForce mapping which reads data from a Microsoft Access database and writes it to a CSV file. In particular, the mapping described in this example uses a custom database SELECT query with a parameter. The SELECT statement combines data from multiple tables. Then, the results are supplied to the mapping for further processing.

The example is accompanied by a mapping design (.mfd) file available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\select-component.mfd**. You might want to open this sample file and analyze it first, or follow the steps below to create it from scratch.

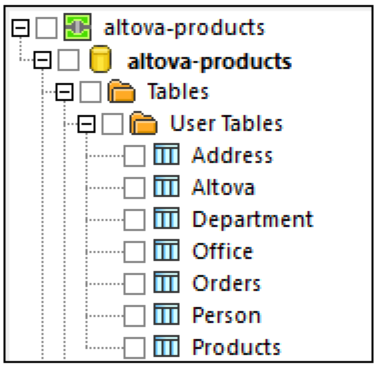
Although this example uses a Microsoft Access database, the process works in the same way for other database types. For information about connecting to other databases, see Connecting to a Database[163].

The goals are as follows:

1. We must select from the database only those orders where the number of ordered items exceeds a custom value. This custom value should be supplied as a parameter to the mapping. To achieve this goal, we will create a custom database SELECT statement that takes an input parameter.
2. In the Access database, the date format is `YYYY-MM-DD HH-MI-SS`. In the CSV file, the time part must be stripped, so the format should be `YYYY-MM-DD`. To achieve this goal, we use the date-from-datetime 1088 function available in MapForce.
3. The resulting CSV file must have the name **OrdersReport.csv**.

Step 1: Add the SELECT structure

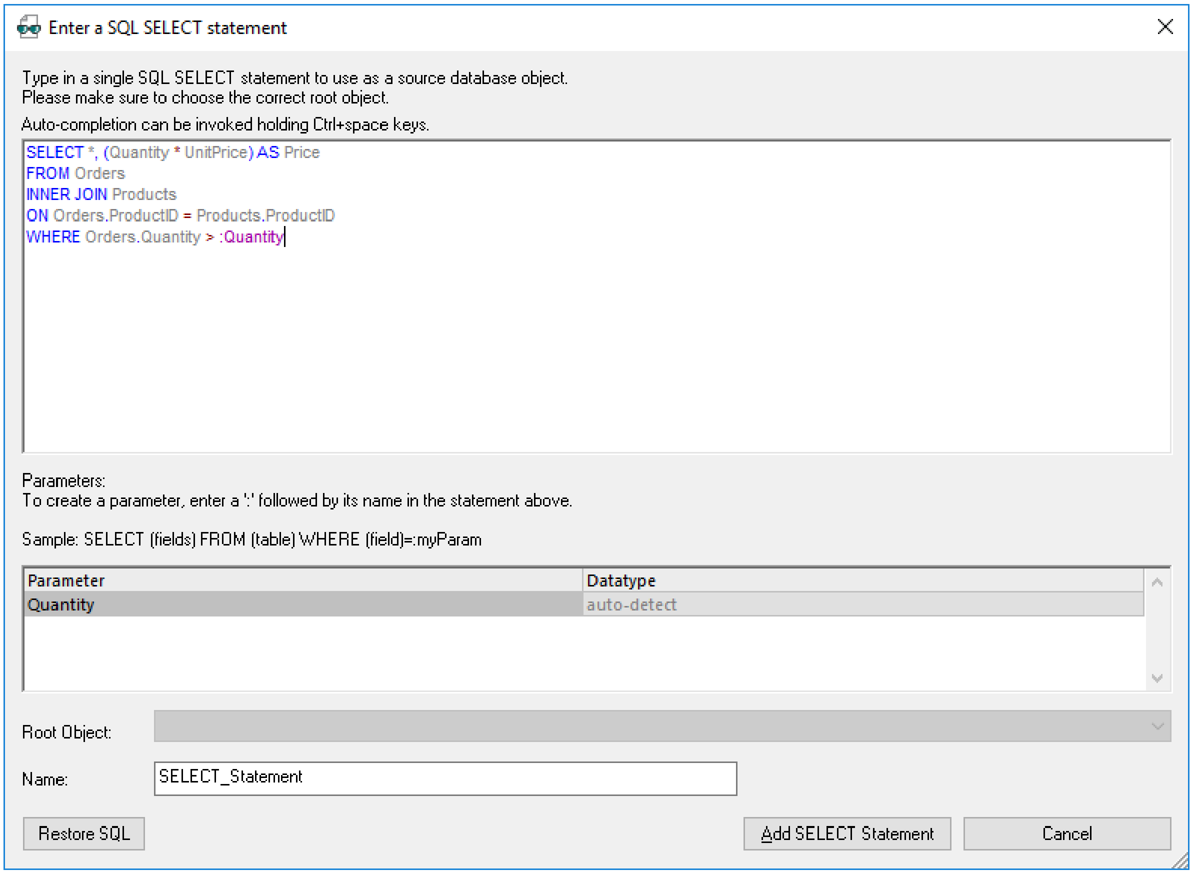
1. On the **Insert** menu, click **Database**.
2. Select **Microsoft Access (ADO)**, and follow the wizard steps to connect to the **altova-products.mdb** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder.



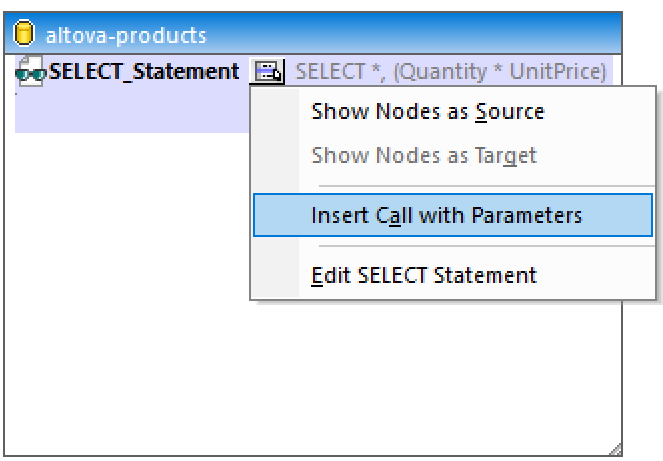
3. On the Insert Database Objects dialog box, click **Add/Edit SELECT Statement**, and enter the following query:

```
SELECT *, (Quantity * UnitPrice) AS Price
FROM Orders
INNER JOIN Products
ON Orders.ProductID = Products.ProductID
WHERE Orders.Quantity > :Quantity
```

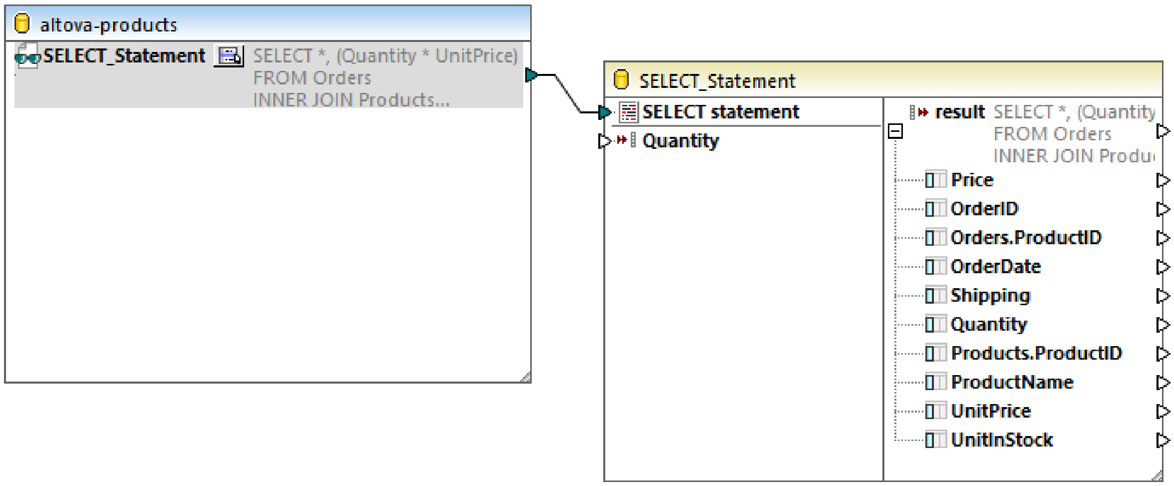
This query uses a join between the Orders and Products tables, and retrieves all fields (`*`), and a computed value (`AS Price`). The query also specifies the **:Quantity** parameter in the WHERE clause.



4. Click **Add SELECT statement**.
5. Click OK. The **altova-products** component has now been added to the mapping area.
6. On the **altova-products** component, click and select **Insert Call with Parameters**.

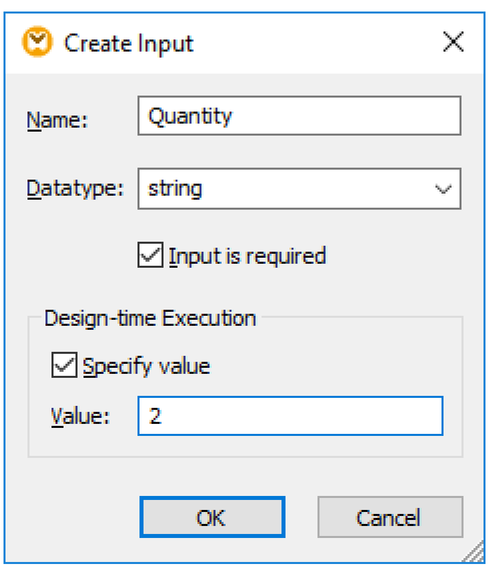


A new structure (SELECT_Statement) is now available on the mapping. It is split into two parts: the left part supplies input connectors and the right part supplies output connectors. Notice that the left part also includes the **Quantity** parameter defined previously.

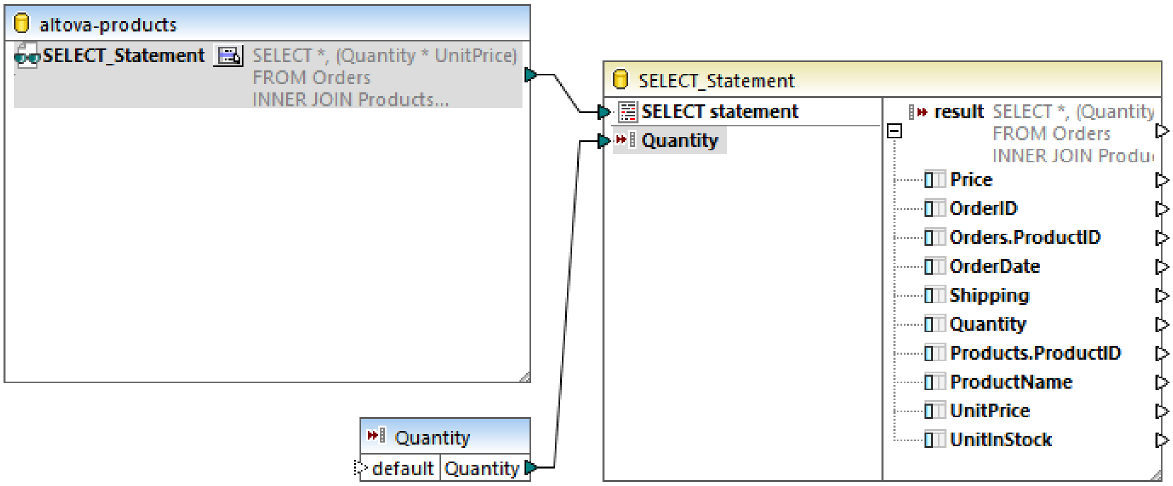


Step 2: Add the input parameter

1. On the **Insert** menu, click **Insert Input**.
2. Type "Quantity" as name.
3. Under **Design-time Execution**, enter a parameter value to be used for executing the mapping during the design phase (in this example, "2"). For more information, see Supplying Parameters to the Mapping 833.

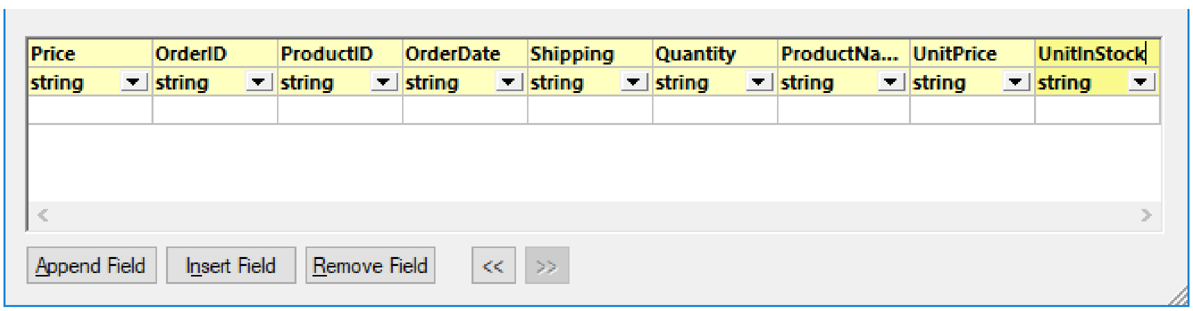


You can now connect the input parameter to the database call structure, as shown below.

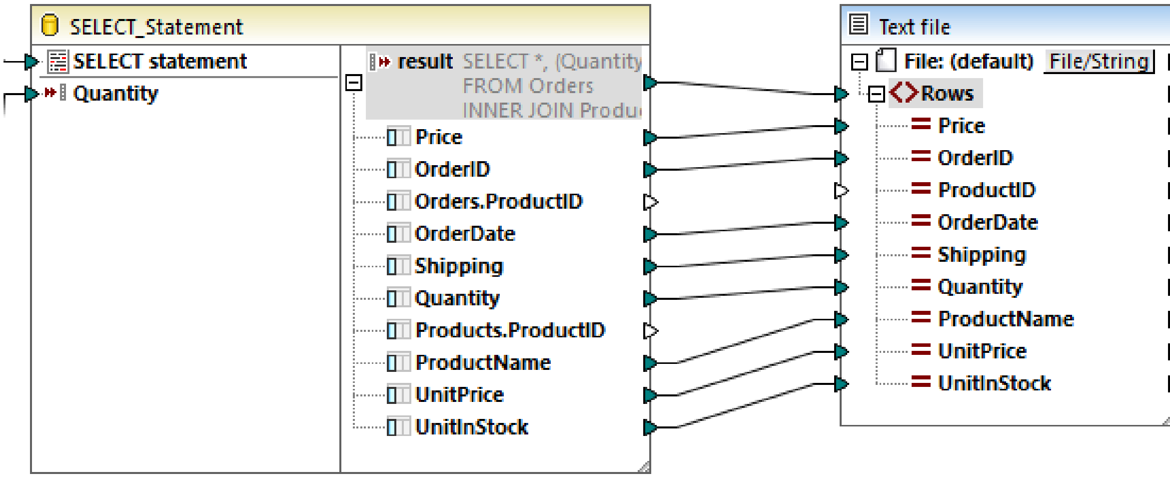


Step 3: Add the target CSV component

1. On the **Insert** menu, click **Text File**.
2. Select **Use simple processing for standard CSV...**, and then click **Continue**.
3. On the Component Settings dialog box, click **Append Field** and add nine new fields. It is recommended that you give to the CSV fields the same name as the name of the database fields, as shown below. This will help you save time later when drawing mapping connections. For more information about these settings, see Setting the CSV Options[572].



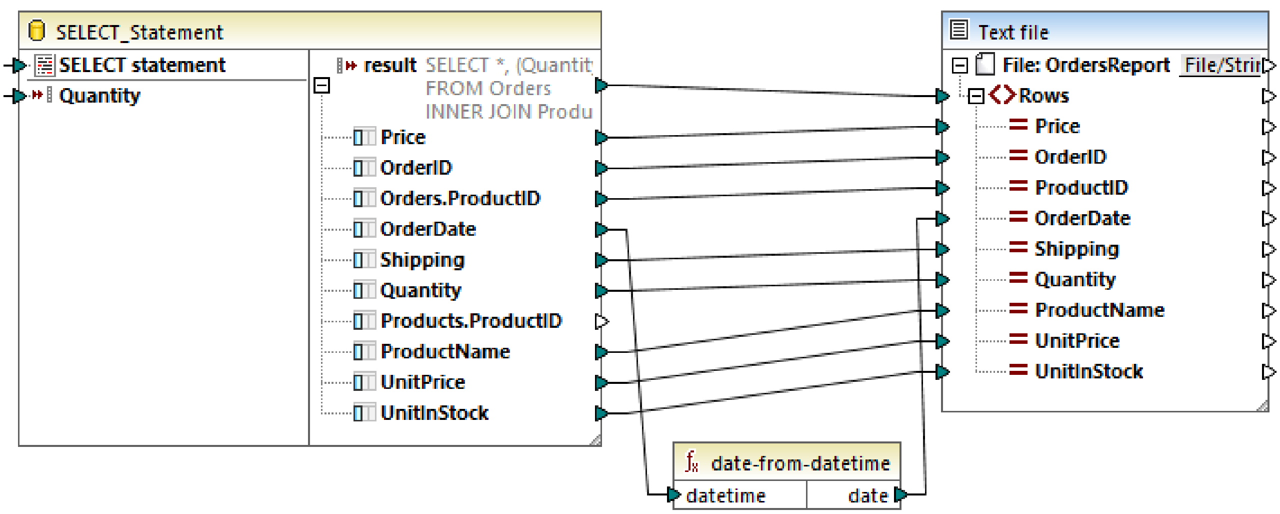
4. Create a connection between the **result** node of the SELECT structure and the **Rows** node of the CSV component.



Because most of the fields in the CSV component have the same name as their database equivalent, mapping connections will likely be drawn automatically when you connect **result** to **Rows**. If this does happen, select the **Connection** menu and make sure that the **Auto Connect Matching Children** option is enabled. The only mapping item that you have to connect manually is **ProductID**, since there is no field with this name in the SELECT structure.

Step 4: Convert the date

In the Libraries window, search for the `date-from-datetime` function and drag it to the mapping area. Then connect its input and output as shown below.



Step 5: Set the name of the output file

To set the name of the output file to **OrdersReport.csv**, double-click the title bar of the CSV component. The Component Settings dialog box opens, where you can enter the value in the **Output File** text box.

4.2.7 Mapping XML Data to / from Database Fields

MapForce enables you to map data to or from database fields (columns) that store XML content. This means that XML data stored by the database field (column) can be extracted and written to any other structure supported by MapForce, and the other way round. You can map data as follows:

1. To or from fields of a dedicated XML type (for example, `Xml` in SQL Server, `XMLType` in Oracle). Reading or writing XML to/from dedicated XML fields is applicable to databases that have native support for XML (such as IBM DB2, Oracle, and SQL Server).
2. To or from text fields storing XML content (for example, `Text, Varchar`). This applies to any database where the text field has sufficient length to store an XML document.

In either of the cases, a valid XML schema must exist for each database column to/from which you want to map data. When a database column stores XML, MapForce provides you with the choice to assign an XML schema directly from the database (if supported by the database), or select a schema from an external file. You can assign one XML schema per database column. If the schema has multiple root elements, you can select a single root element of that schema.

When XML is stored as a string field in a database, the character encoding of the XML document is that of the underlying string field. If the database field does not store text as Unicode, some characters cannot be represented.

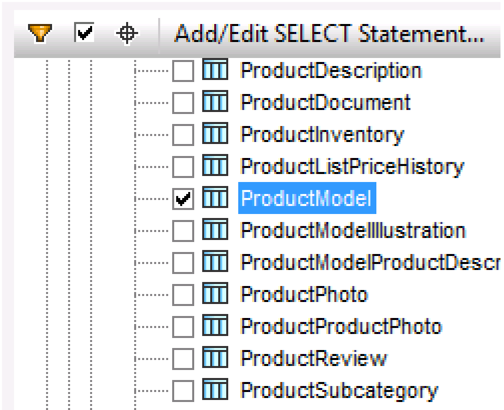
Some databases support XML encoding for XML fields (which may not necessarily be the same as that of the database character set). If supported by the database, the XML document encoding declaration is assumed to be the one declared in the XML field. For information about the XML encoding support provided by various databases, refer to their documentation.

4.2.7.1 Assigning an XML Schema to a Database Field

This topic illustrates how to assign a schema to a field that is natively defined as XML type in the database. The instructions below use SQL Server 2014 and the AdventureWorks 2014 database. The latter can be downloaded from the AdventureWorks samples page on GitHub (https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks). Note that mapping of data to or from XML fields works in the same way with other database types that support XML fields.

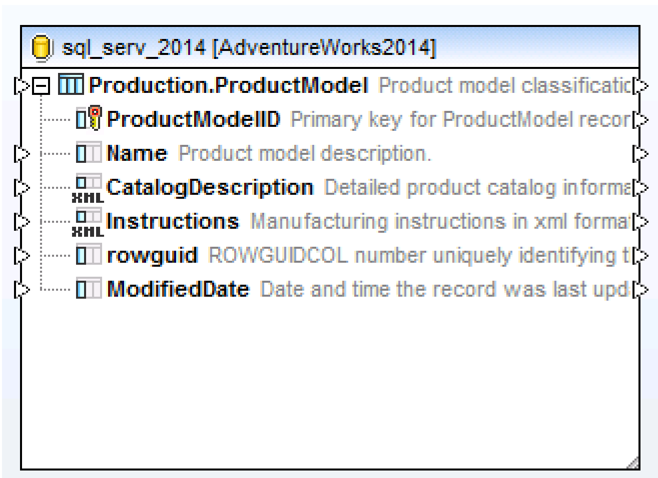
To add the Adventure Works 2014 database as a mapping component:

1. On the **Insert** menu, click **Database**, and follow the wizard to connect to the database using your preferred method (ADO or ODBC). For more information, see ADO Connection 170 and ODBC Connection 182. NOTE: If you use the **SQL Server Native Client** driver, you might need to set the **Integrated Security** property to a space character (see Setting up the SQL Server Data Link Properties 173).
2. On the **Insert Database Object** dialog box, expand the **Production** schema, and then select the **ProductModel** table.

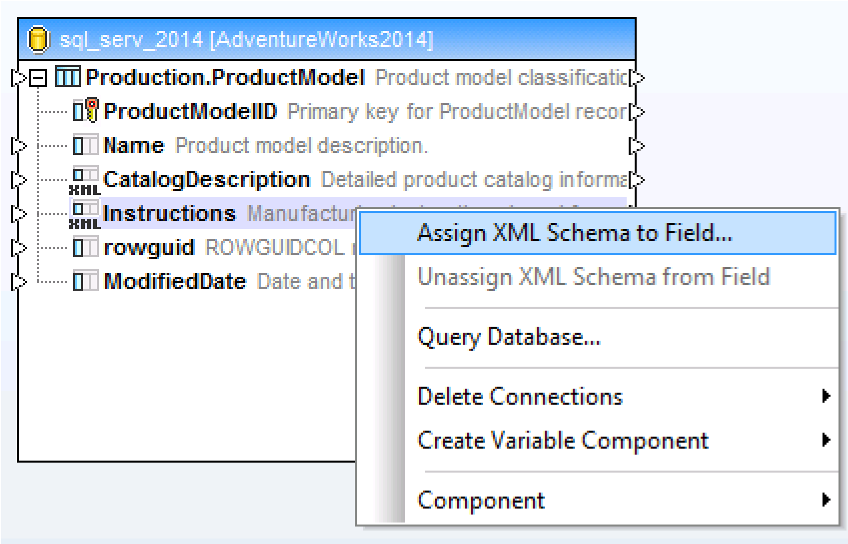


3. Click OK.

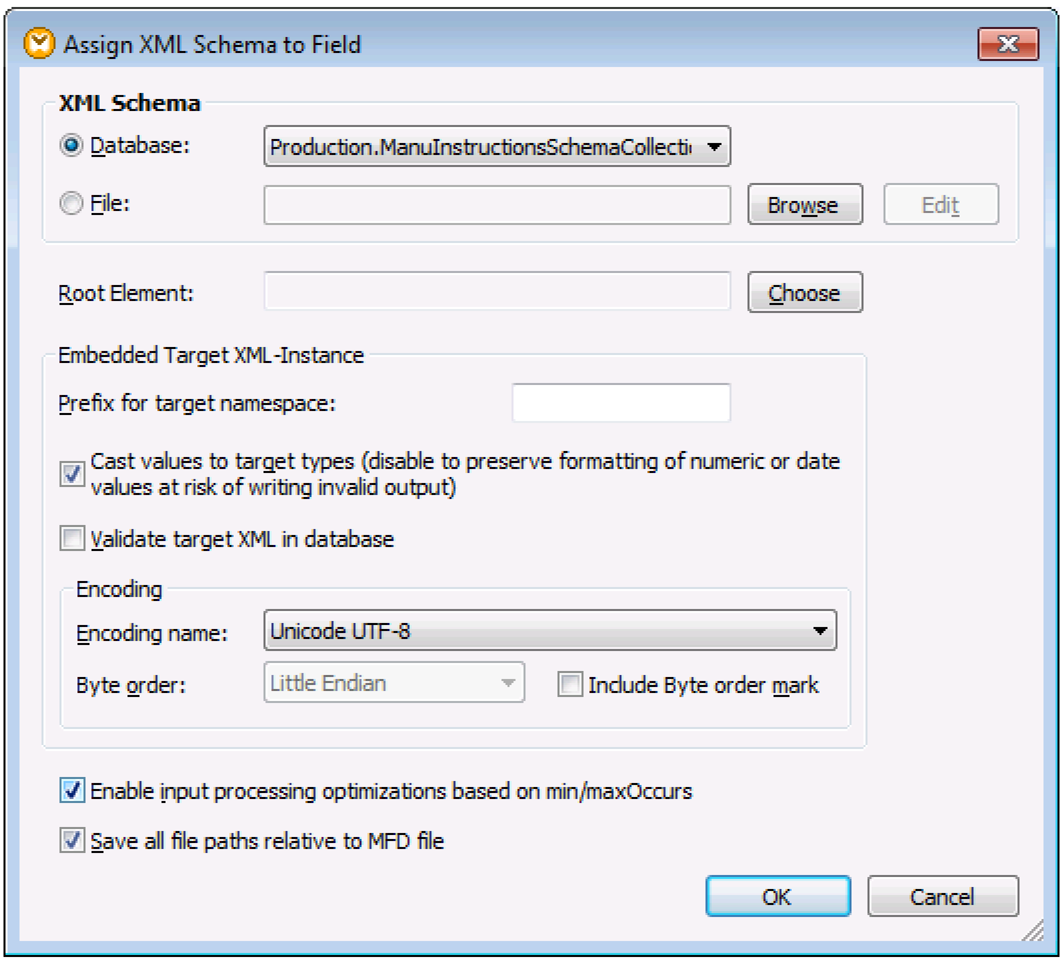
The database table has now been added to the mapping area. Notice that this table has two fields of XML type: **CatalogDescription** and **Instructions**:



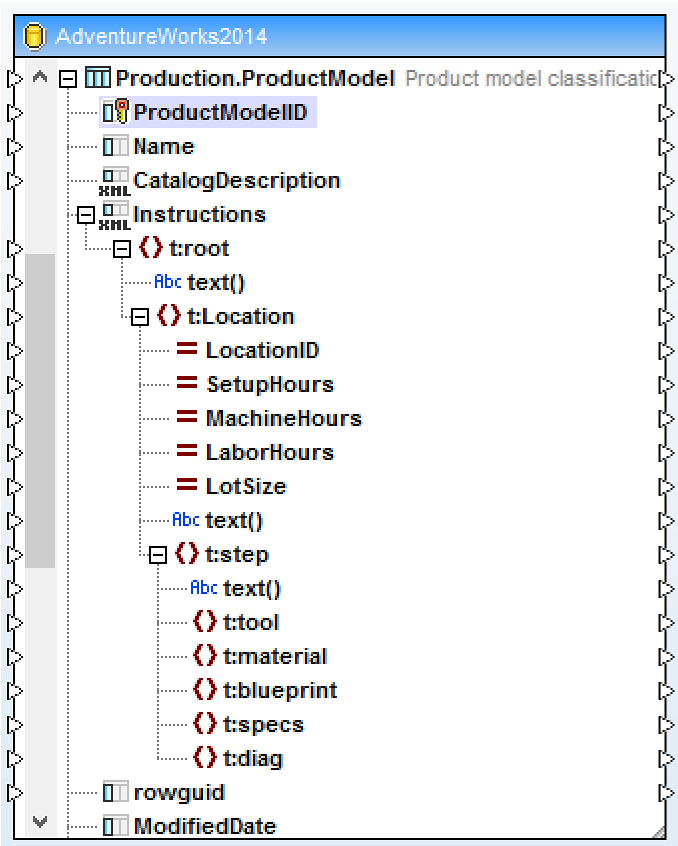
For the structure of the XML fields to appear on the mapping, the XML schema of the field content is required. Right-click the **Instructions** field and select **Assign XML Schema to Field** from the context menu.



In this particular example, you will assign a schema to the **Instructions** field directly from the database. To do this, select the **Production.ManuInstructionsSchemaCollection** item next to the **Database** option, and then click OK.



The structure of the XML field now appears on the component. You can now draw connections (and map data) to or from this field.



4.2.7.2 Example: Writing XML Data to a SQLite Field

This example walks you through the steps required to create a MapForce mapping which reads data from multiple XML files and writes it to a SQLite database. The goal of the mapping is to create, for each source XML file, a new database record in the SQLite database. Each record will store the XML document as a TEXT field.

All the files used in this example are available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**. The file names are as follows:

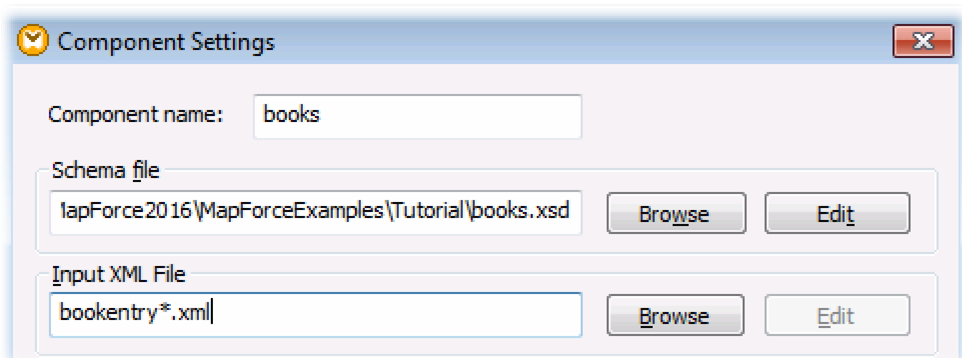
The mapping design file	• **XmlToSqliteField.mfd**
The source XML files	• **bookentry1.xml** • **bookentry2.xml** • **bookentry3.xml**
The XML schema used for validation	• **books.xsd**
The target SQLite database	• **Library.sqlite**

To achieve the goal of the mapping, the following steps will be taken:

1. Add the XML component and configure it to read from multiple files.
2. Add the SQLite database component and assign an XML schema to the target TEXT field.
3. Create the mapping connections and configure the database INSERT action.

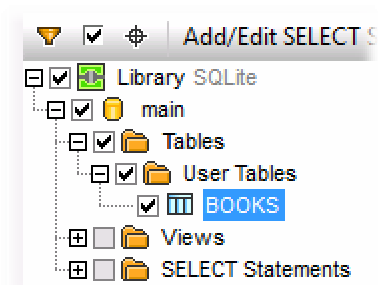
Step 1: Add the XML component

1. On the **Insert** menu, click **XML Schema/File** and browse for the **books.xsd** schema located in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** directory. When prompted to supply a sample XML file, click **Skip**. When prompted to select a root element, select **Books**.
2. Double-click the component header and type **bookentry*.xml** in the **Input XML File** box. This instructs MapForce to read all XML files whose name begins with "bookentry-" in the source directory. For more information about this technique, see Processing Multiple Input or Output Files Dynamically1313.

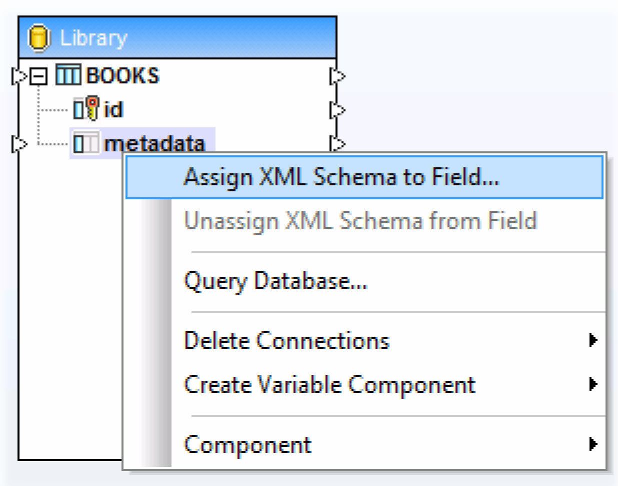


Step 2: Add the SQLite component

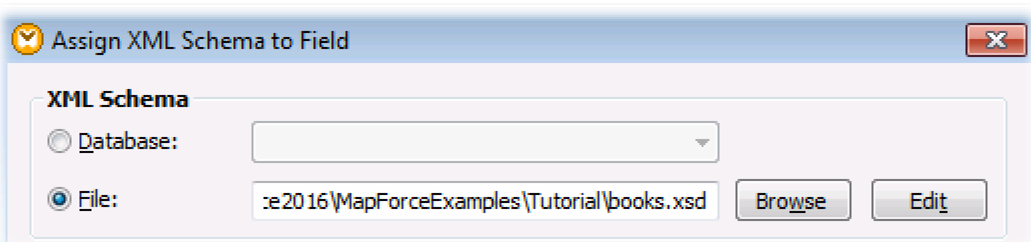
On the **Insert** menu, click **Database**, and follow the wizard to connect to the **Library.sqlite** database file from the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** directory (see also Connecting to an Existing SQLite Database192). When prompted to select the database objects, select the `BOOKS` table.



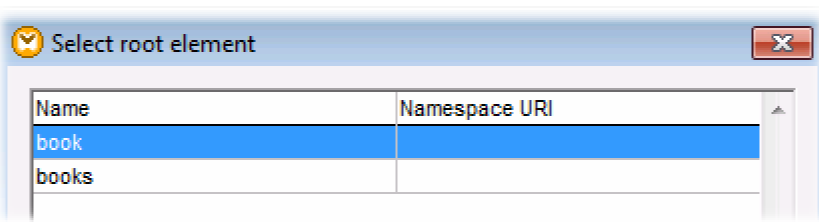
The database field where XML content will be written is called `metadata`. To assign an XML schema to this field, right-click it and select **Assign XML Schema to Field** from the context menu.



In this tutorial, the schema assigned to the `metadata` field is the same one used to validate the source XML files. Click **Browse** and select the **books.xsd** schema from the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** directory:

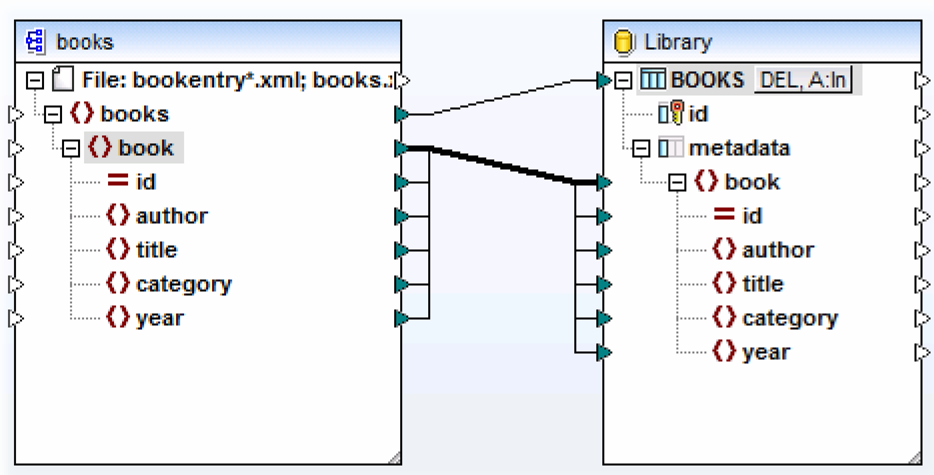


The **books.xsd** schema has two elements with global declaration: `book` and `books`. In this example, we will set `book` as the root element of the XML written to the database field. Click **Choose**, and select `book` as root element:



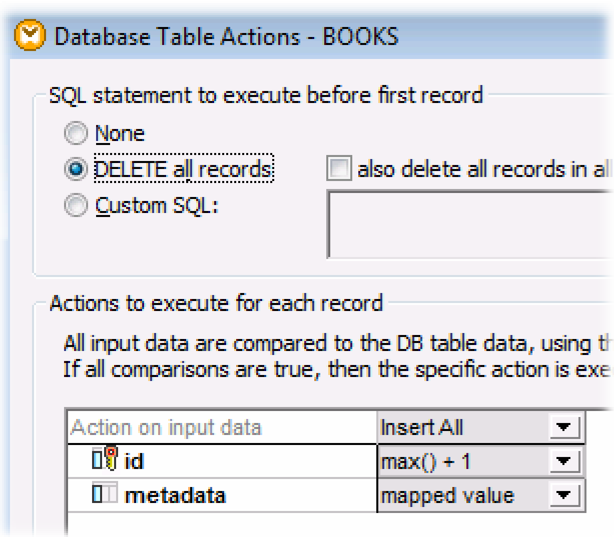
Step 3: Create the mapping connections and configure the database INSERT action

Create the mapping connections as follows:



As shown above, the connection from `book` to `book` is a "Copy-All" connection, since both the source and target use the same schema and the names of child elements are the same. For more information about such connections, see [Copy-all connections](97).

The topmost connection (`books` to `BOOKS`) iterates through each book element in the source and writes a new record in the BOOKS table. Click the `A:In` button on the database component and set the database update settings as shown below:



The **DELETE all records** option instructs MapForce to delete the contents of the `BOOKS` table before inserting any records.

The **Insert All** actions specify that a database `INSERT` query will take place. The field `id` is generated from the database itself, while the field `metadata` will be populated with the value provided by the mapping.

Make sure to save the mapping before running it.

To run the mapping and view the generated output, click the **Output** tab. Note that this action does not update the database immediately. When you are ready to run the generated database script, select the menu command **Output | Run SQL Script** (or click the toolbar button).

4.2.7.3 Example: Extracting Data from IBM DB2 XML Type Columns

This example illustrates how to extract data from IBM DB2 database columns of XML type and write it to a target CSV file. It also illustrates how to use XQuery statements embedded into SQL in order to retrieve XML content conditionally. The example requires access to an IBM DB2 database where you have permission to create and populate tables.

First, let's prepare the database so that it actually contains XML data. This can be done either in a database administration tool specific to your database, or directly in MapForce. To do this directly in MapForce, follow the steps below:

1. Create a new mapping and click the **DBQuery** tab.
2. Click **Quick Connect** ( ) and follow the wizard steps to create a new database connection (see also Database Connection Examples [195]).
3. Paste the following text into the SQL Editor. This SQL query creates a database table called `ARTICLES` and populates it with data.

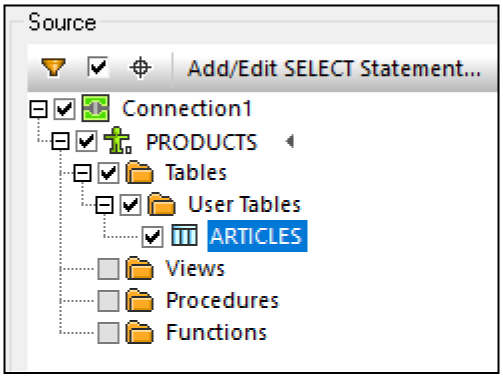
```
-- Create the table
CREATE TABLE
    ARTICLES (
        id INTEGER NOT NULL,
        article XML ) ;
-- Populate the table
INSERT INTO ARTICLES VALUES
   (1, '<Article>
      <Number>1</Number>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>'),
(2, '<Article>
      <Number>2</Number>
      <Name>Socks</Name>
      <SinglePrice>230</SinglePrice>
   </Article>'),
(3, '<Article>
      <Number>3</Number>
      <Name>Pants</Name>
      <SinglePrice>34</SinglePrice>
   </Article>'),
 (4, '<Article>
      <Number>4</Number>
      <Name>Jacket</Name>
      <SinglePrice>5750</SinglePrice>
   </Article>');
```

4. Click the **Execute** ( ) button. The query execution result is displayed in the Query Results window. If the query is executed successfully, four rows are added to the newly created table.

Next, we will create a mapping which retrieves XML data from the `ARTICLES` table created above conditionally. The goal is to retrieve from the `ARTICLES` column only articles with a price greater than 100.

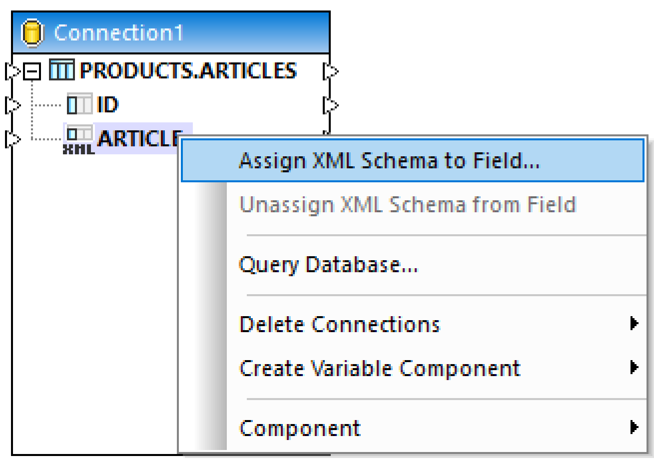
Step 1: Add the database

1. Click the **Mapping** tab to switch back to the mapping pane.
2. On the **Insert** menu, click **Database**, and follow the wizard steps to connect to the database.
3. When prompted to select the database objects, select the `ARTICLES` table created previously.



Step 2: Assign the schema to the XML type field

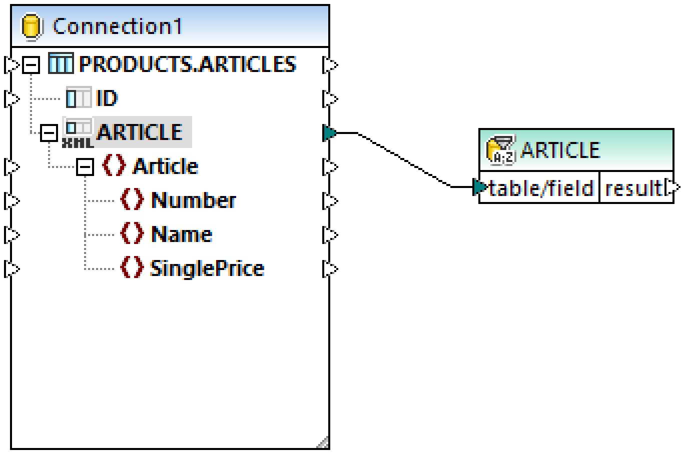
1. Right-click the `ARTICLE` item of the component, and select **Assign XML Schema to field** from the context menu.



2. Select **File**, and browse for the following schema: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\DB2xsd.xsd**.

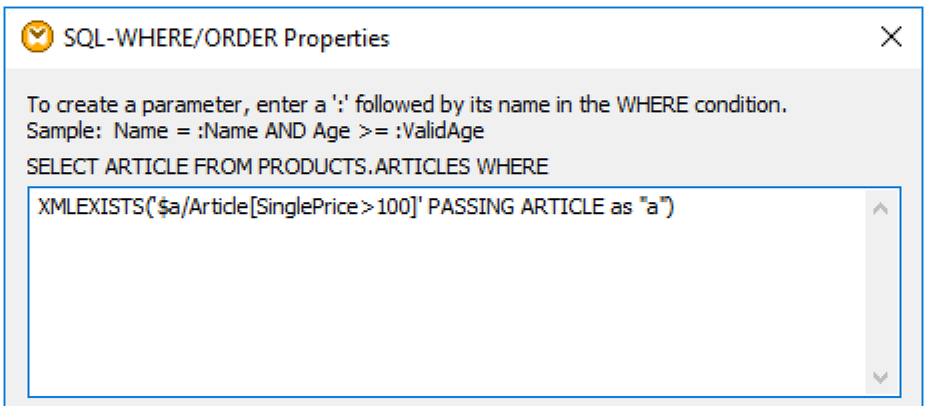
Step 3: Add the SQL WHERE/ORDER component

1. On the **Insert** menu, click **SQL WHERE/ORDER**.
2. Connect the `ARTICLE` XML type column to the input of the SQL WHERE/ORDER.



3. In the SQL-WHERE/ORDER Properties dialog box, enter the following text:

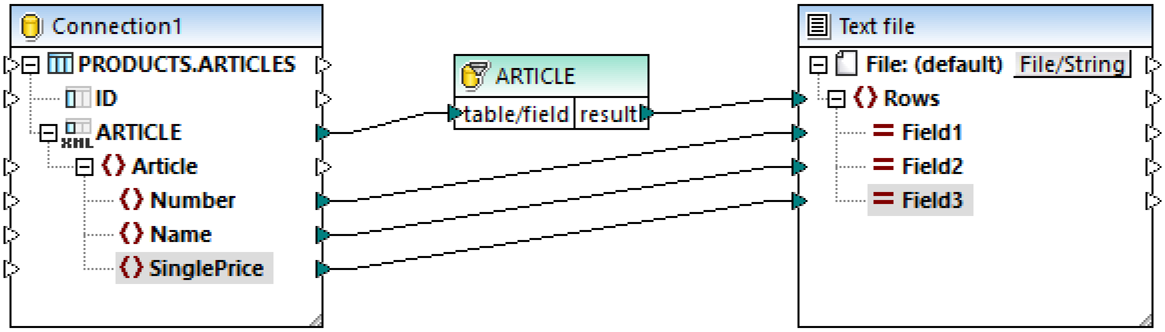
```
XMLEXISTS('$a/Article[SinglePrice>100]' PASSING ARTICLE as "a")
```



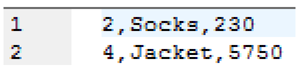
The text above represents the "WHERE" part of the SQL query. At mapping runtime, it will be combined with the "SELECT" part displayed on the dialog box. This statement uses the `XMLEXISTS` function and syntax specific to IBM DB2 databases.

Step 4: Add the target CSV file

1. On the **Insert** menu, click **Text File**.
2. When prompted, select **Use simple processing for standard CSV...** , and click **Continue**.
3. Click **Append Field** three times to add three fields which will store the article number, name, and price, respectively. Leave all other settings as is.
4. Draw the mapping connections as shown below.



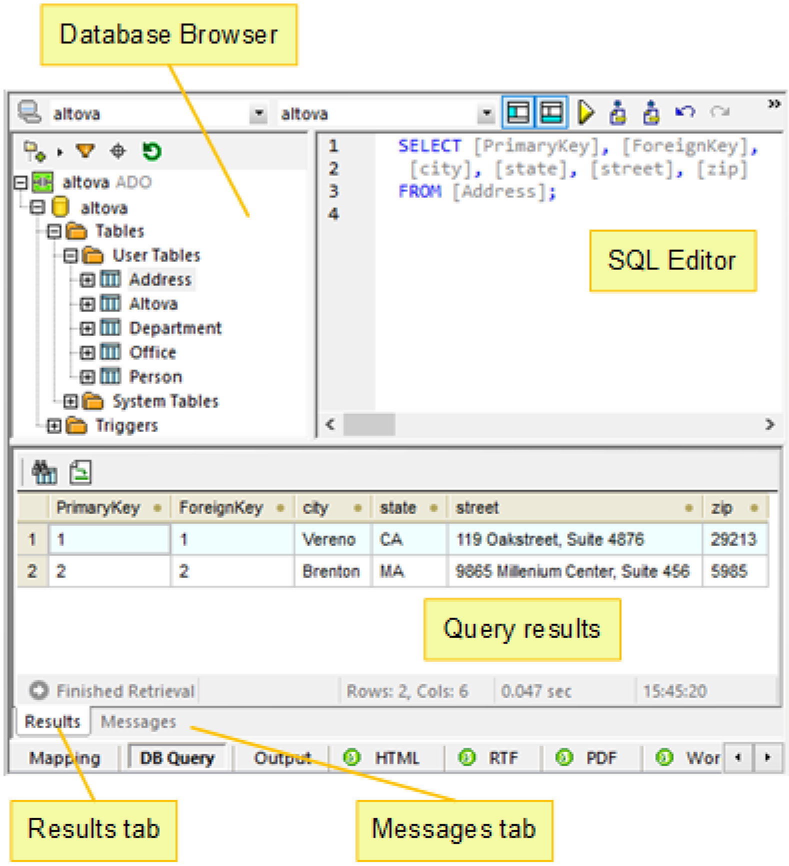
You can now preview the mapping result, by clicking the **Output** tab. As expected, only articles with price greater than 100 are shown in the output.



4.2.8 Browsing and Querying Databases

MapForce has a dedicated Database Query pane (also called **DB Query**) that allows you to query a database independently of the mapping process. Such direct queries are not saved together with the mapping *.mfd file but provide a convenient way to browse or modify the contents of a database directly from MapForce.

A separate **DB Query** pane exists for each currently active mapping. You can create multiple active connections, to different databases, within each **DB Query** pane. Note that the connections created from the **DB Query** pane are not part of the mapping and thus are not preserved after you close MapForce, unless you define them as Global Resources 1397.



The Database Query pane consists of the following parts:

- **Database Browser**, which displays connection info and database tables
- **SQL Editor**, in which you write your SQL queries
- **Results tab**, which displays the query results in tabular form
- **Messages** tab, which displays warnings or error messages.

The upper area of the Database Query pane contains the connection controls allowing you to define the working databases, as well as the connection and database schemas.

To configure database query settings, use the *Database* section in the **Options** dialog box. For more information, see Database Query Settings 1480.

4.2.8.1 Selecting or Connecting to a Database

For each database that you want to query, a database connection must be created. If your mapping already includes a database component, you can select the existing database connection from the upper area of the **DB Query** pane (by default, the connection is "Offline") and start exploring the database objects and run queries.

If your mapping does not include any database component, or if you want to connect to a new database, click **Quick Connect** ( ) and follow the wizard steps to create a new database connection (see Examples 195). You can also select an existing database connection from Global Resources, if one has been defined as such (see Global Resources 1397).

Once you are connected to the database, you can create database queries using one of the following methods:

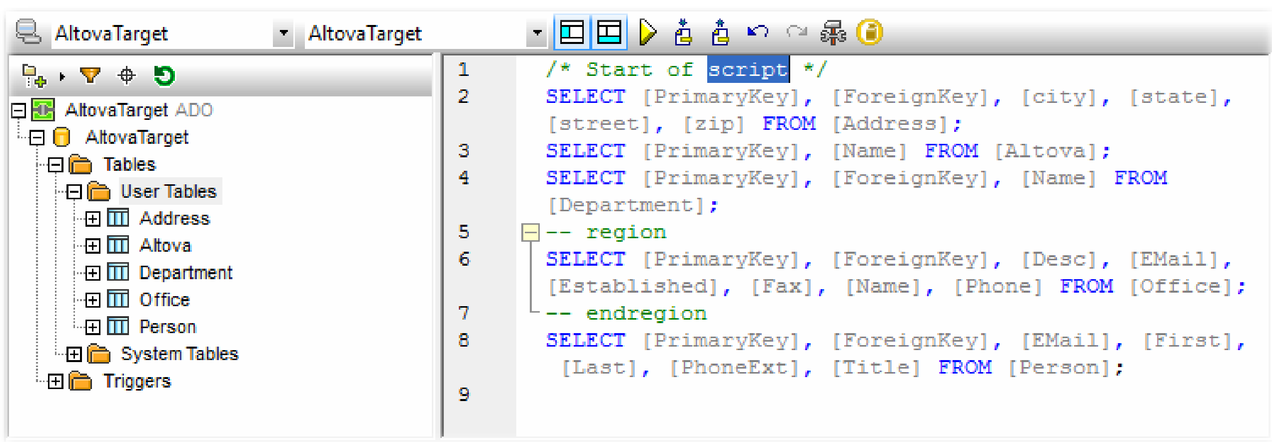
- Import the SQL query into the SQL Editor pane from an existing SQL file.
- Write the query in the SQL Editor pane.
- Right-click an object in the Database Browser pane and generate a query (typically, SELECT).

When you are ready to run the query displayed in the SQL Editor pane, click the **Execute** button. The database data is retrieved and displayed in the Results tab in tabular form. Note that the status bar displays the "Finished Retrieval" message (Finished Retrieval), and other pertinent information about the query results.

Once the "Finished Retrieval" message is displayed, you can search, sort, or copy to clipboard the search results (see Copying, Sorting, and Searching the Query Results 388).

4.2.8.2 Creating and Editing SQL Statements

The SQL Editor is used to write and execute SQL statements. It displays any SQL statements that you may have generated automatically, loaded from existing SQL scripts, or written manually. The SQL Editor supports autocompletion (see Auto-Completion 278), regions, and line or block comments.



The SQL Editor toolbar provides the following buttons:

	Toggle Browser: Toggles the Browser pane on and off.
	Toggle Result: Toggles the Result pane on and off.
	Execute (F5): Clicking this button executes the SQL statements that are currently selected. If multiple statements exist and none are selected, then all are executed.

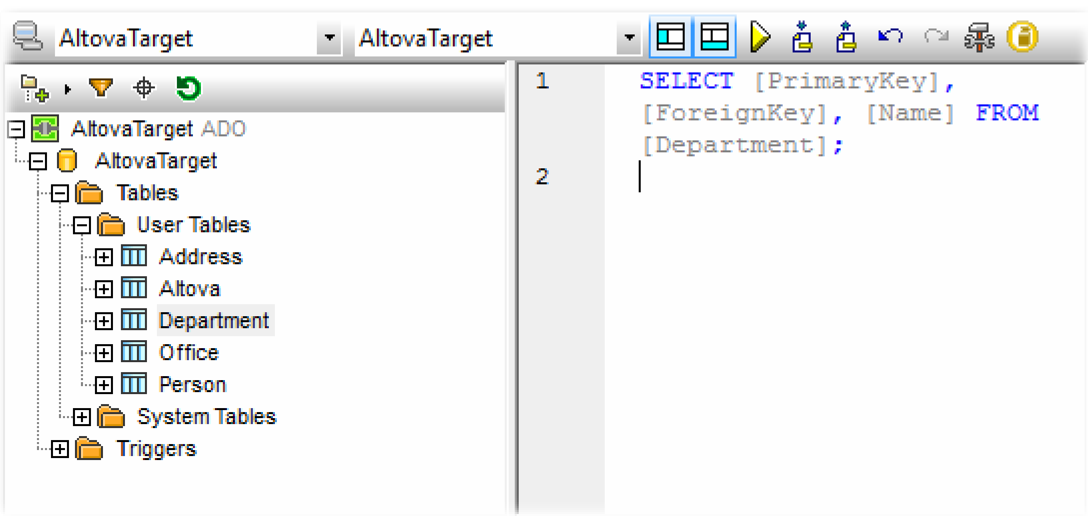
	Undo: Allows you to undo an unlimited number of edits in the SQL window.
	Redo: Allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.
	Import SQL file: Opens an SQL file in the SQL Editor, which can then be executed.
	Export SQL file: Saves SQL queries for later use.
	Open SQL script in DatabaseSpy: Starts DatabaseSpy and opens the script in the SQL Editor.
	Options: Opens the Options dialog box allowing you to define general database query settings as well as SQL Editor settings.

4.2.8.2.1 Generating SQL Statements

SQL statements can be generated automatically from the Database Browser, loaded from scripts, or entered manually.

To generate SQL SELECT statements from the Database Browser, do one of the following:

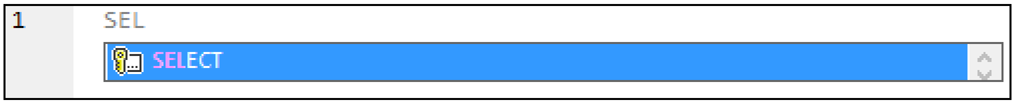
- Click a database object (such as a table or view), or a folder, in the Database Browser and drag it into the SQL Editor.



- Right-click a database object in the Database Browser and select **Show in SQL Editor | Select**.

To create SQL statements manually:

1. Start entering the SQL statement in the SQL Editor. If autocompletion is set to occur automatically, a drop-down list with suggestions appears while you enter statement.



2. Use the cursor **Up** and **Down** keys to select a suggestion, and then press **Enter** to insert the highlighted option (see also SQL Auto-Completion Suggestions[278]).

4.2.8.2.2 Executing SQL Statements

The SQL statements that appear in the SQL Editor can be executed against the database, with immediate effect. The result of the SQL query and the number of affected rows is displayed in the **Messages** pane of the **DB Query** pane.

When multiple SQL statements appear in the SQL Editor, only the selected statements will be executed. You can select individual SQL statements as follows:

- Holding the left mouse button clicked, drag the cursor over a specific statement.
- Click a line number in the SQL Editor.
- Triple-click a specific statement.

To execute a SQL statement:

1. Enter or select the SQL statement in the SQL Editor (see Generating SQL Statements[379]).
2. Click the **Execute** ( ) button.

4.2.8.2.3 Importing and Exporting SQL Scripts

You can save any SQL that appears in an SQL Editor window to a file and re-use the script file later on.

To export the contents of the SQL Editor pane to a file:

- Click **Export SQL file** ( ), and enter a name for the SQL script.

To import a previously saved SQL file:

- Click **Import SQL file** ( ), and select the SQL file you want to open.

4.2.8.2.4 Adding and Removing SQL Comments

The SQL Editor allows you to comment out statements, parts of statements, or groups of statements. These statements, or the respective parts of them, are skipped when the SQL script is being executed.

To comment out a section of text:

1. Select a statement or part of a statement.
2. Right-click the selected statement and select **Insert / Remove Block Comment**.

```

/*
SELECT [PrimaryKey], [ForeignKey], [city], [state], [street], [zip] FROM [Address];
SELECT [PrimaryKey], [Name] FROM [Altova];
SELECT [PrimaryKey], [ForeignKey], [Name] FROM [Department];*/

SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail], [Established], [Fax], [Name], [Phone] FROM [Office];
SELECT [PrimaryKey], [ForeignKey], [EMail], [First], [Last], [PhoneExt], [Title] FROM [Person];

```

To comment out text line by line:

- Right-click at the position you want to comment out the text and select **Insert / Remove Line Comment**. The statement is commented out from the current position of the cursor to the end of the statement.

To remove a block comment or a line comment:

1. Select the part of the statement that is commented out. If you want to remove a line comment, it is sufficient to select only the comment marks -- before the comment.
2. Right-click and select **Insert / Remove Block (or Line) Comment**.

4.2.8.2.5 Using Bookmarks

Bookmarks are used to mark items of interest in long scripts.

To add a bookmark:

- Right-click the line you want to have bookmarked and select **Insert/Remove Bookmark** from the context menu.

```
SELECT [PrimaryKey], [ForeignKey], [city], [state],
[street], [zip] FROM [Address];
SELECT [PrimaryKey], [Name] FROM [Altova];
SELECT [PrimaryKey], [ForeignKey], [Name] FROM
[Department];
SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail],
[Established], [Fax], [Name], [Phone] FROM [Office];
SELECT [PrimaryKey], [ForeignKey], [EMail], [First],
[Last], [PhoneExt], [Title] FROM [Person];
```

A bookmark icon is displayed in the margin at the beginning of the bookmarked line.

To remove a bookmark:

- Right-click the line from where you want to remove the bookmark and select **Insert/Remove Bookmark** from the context menu.

To navigate between bookmarks:

- To move the cursor to the next bookmark, right-click and select **Go to Next Bookmark**.
- To move the cursor to the previous bookmark, right-click and select **Go to Previous Bookmark**.

To remove all Bookmarks:

- Right-click and select **Remove all Bookmarks**.

4.2.8.2.6 Inserting Regions

Regions are sections of text that you mark and declare as a unit to structure your SQL scripts. Regions can be collapsed and expanded to display or hide parts of SQL scripts. It is also possible to nest regions within other regions.

When you insert a region, an expand/collapse icon and a `--region` comment are inserted above the selected text.

Note: You can change the name of a region by appending descriptive text to the --region comment. The word "region" must not be deleted, e.g. --region DB2query.

To create a region:

1. In the SQL Editor, select the statements you want to make into a region.
2. Right-click and select **Insert Region** from the context menu. The selected statements become a region which can be expanded or collapsed.


```
SELECT [PrimaryKey], [ForeignKey], [city], [state],
[street], [zip] FROM [Address];
-- region
SELECT [PrimaryKey], [Name] FROM [Altova];
SELECT [PrimaryKey], [ForeignKey], [Name] FROM
[Department];
-- endregion
SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail],
[Established], [Fax], [Name], [Phone] FROM [Office];
SELECT [PrimaryKey], [ForeignKey], [EMail], [First],
[Last], [PhoneExt], [Title] FROM [Person];
```

3. Click the + or - box to expand or collapse the region.

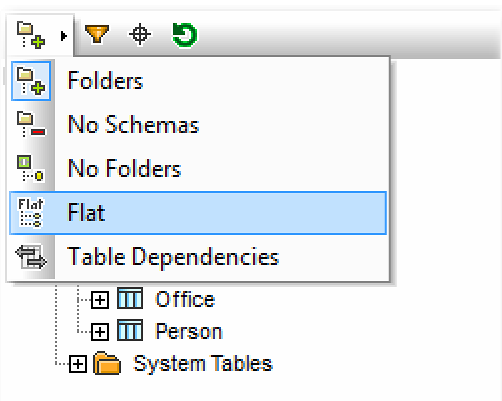
To remove a region:

- Delete the `-- region` and `-- endregion` comments.

4.2.8.3 Browsing Database Objects

When you are connected to one or several databases, the **Database Browser** pane gives a full overview of the objects in each database, including tables, views, procedures, and so on, up to the most detailed level. For databases with XML support, the **Database Browser** additionally shows registered XML schemas in a separate folder.

For custom navigation through database objects, the **Database Browser** pane includes several predefined database display layouts. The predefined layouts are available in the top area of the Database Browser.



To select a layout, click the **Folders Layout** ( ) drop-down button and select an entry from the list. Note that the button changes with the selected layout.

- The **Folders** layout organizes database objects into folders based on object type in a hierarchical tree, this is the default setting.

- The **No Schemas** layout is similar to the Folders layout, except that there are no database schema folders; tables are therefore not categorized by database schema.
- The **No Folders** layout displays database objects in a hierarchy without using folders.
- The **Flat** layout divides database objects by type in the first hierarchical level. For example, instead of columns being contained in the corresponding table, all columns are displayed in a separate Columns folder.
- The **Table Dependencies** layout categorizes tables according to their relationships with other tables. There are categories for tables with foreign keys, tables referenced by foreign keys and tables that have no relationships to other tables.

In addition to layout navigation, you can use the **Database Browser** for the following tasks:

- Generate SQL statements (see Generating SQL Statements 379).
- Filter and search the displayed database objects (see Filtering and Searching Database Objects 384).
- Sort the tables into "System" and "User" tables.
- Refresh the root object of the active data source.

To sort tables into User and System tables:

- In the **Database Browser**, right-click the "Tables" folder, and then select **Sort into User and System Tables**.

  Note: This function is available when one of the following layouts is selected: **Folders**, **No Schemas** or **Flat**.

To refresh the root object of the active data source:

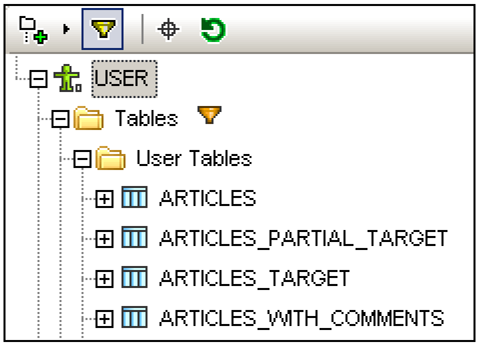
- At the top of the **Database Browser**, click **Refresh** ( ).

4.2.8.3.1 Filtering and Searching Database Objects

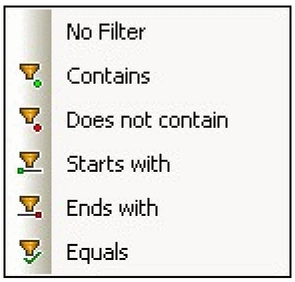
You can filter any database objects (schemas, tables, views, etc) displayed in the **Database Browser** by name or part of a name. Objects are filtered as you type in the characters. Filtering is case-insensitive by default. Filtering is not supported if you have selected the "No Folders" layout.

Filtering database objects

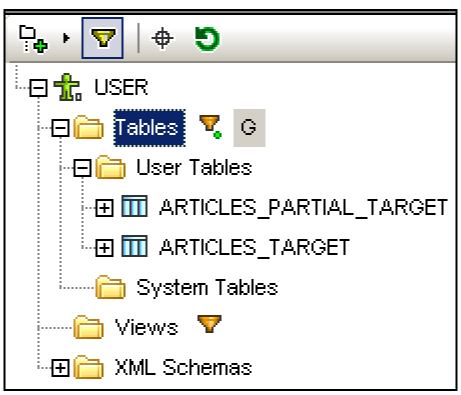
1. At the top of the Database Browser, click **Filter Folder contents** ( ). Filter icons appear next to all folders in the currently selected layout.



2. Click the filter icon next to the folder you want to filter, and select the filtering option from the context menu (for example, **Contains**).



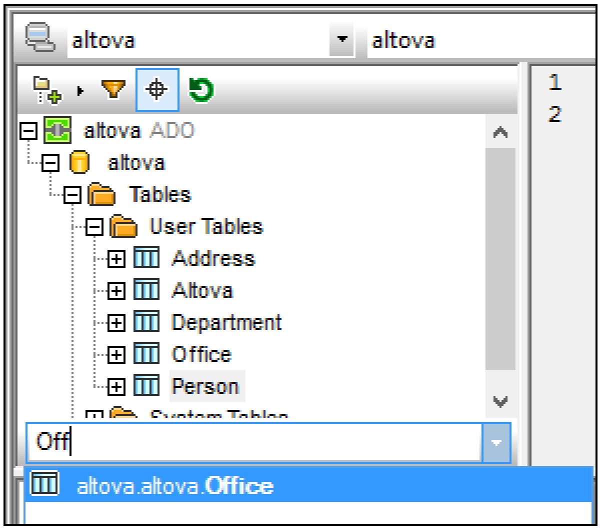
3. In the empty field which appears next to the filter icon, enter the search text (for example, "G"). The results are adjusted as you type.



Searching database objects

To find a specific database item by its name, you can either use filtering functions or the **Object Locator**. To find database elements using the Object Locator:

1. At the top of the **Database Browser**, click **Object Locator** ( ) .
2. In the drop-down list that appears, enter the search text (for example, "Off").



3. Click an object in the list to select it in the **Database Browser**.

4.2.8.3.2 Context Options in Database Browser

The context menu options available in the **Database Browser** depend on the object you have selected, for example:

- Right-clicking the "root" object allows you to **Refresh** the database.
- Right-clicking a folder always presents the same choices: **Expand | Siblings | Children** and **Collapse | Siblings | Children**.
- Right-clicking a database object reveals the **Show in SQL Editor** command and the submenu items discussed below.

To select multiple database objects, press either **Shift + Click** or **Ctrl + Click**.

Note: The syntax of the SQL statements may vary depending on the database you are using. The syntax below applies to Microsoft SQL Server 2014.

The following options are available under the **Show in SQL Editor** context menu for the root object:

- **CREATE**: Creates a CREATE statement for the selected database root object, for example:
  `CREATE DATABASE [MYDB]`
- **DROP**: Creates a DROP statement for the selected database root object, for example:
  `DROP DATABASE [MYDB]`

The following options are available under the **Show in SQL Editor** context menu for tables and views:

- **SELECT**: Creates a SELECT statement that retrieves data from all columns of the source table, for example:
  `SELECT [DepartmentID], [Name], [GroupName], [ModifiedDate] FROM [MYDB].[HumanResources].[Department]`
- **Name**: Returns the name of the table.

- **Path**: Returns the full path of the tables, in the format `DataSourceName.DatabaseName.SchemaName.TableName`.

  If you selected multiple tables, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for columns:

- **SELECT**: Creates a SELECT statement that retrieves data from the selected column(s) of the parent table, for example:
  `SELECT [DepartmentID] FROM [MYDB].[HumanResources].[Department]`
- **Name**: Returns the name of the selected column.
- **Path**: Returns the full path of the column, in the format `DataSourceName.DatabaseName.SchemaName.TableName.ColumnName`.

  If you selected multiple columns, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for constraints:

- **Name**: Returns the name of the selected constraint.
- **Path**: Returns the full path of the constraint, in the format `DataSourceName.DatabaseName.SchemaName.TableName.ConstraintName`.

  If you selected multiple constraints, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for indexes:

- **Name**: Returns the name of the selected index.
- **Path**: Returns the full path of the index, in the format `DataSourceName.DatabaseName.SchemaName.TableName.IndexName`.

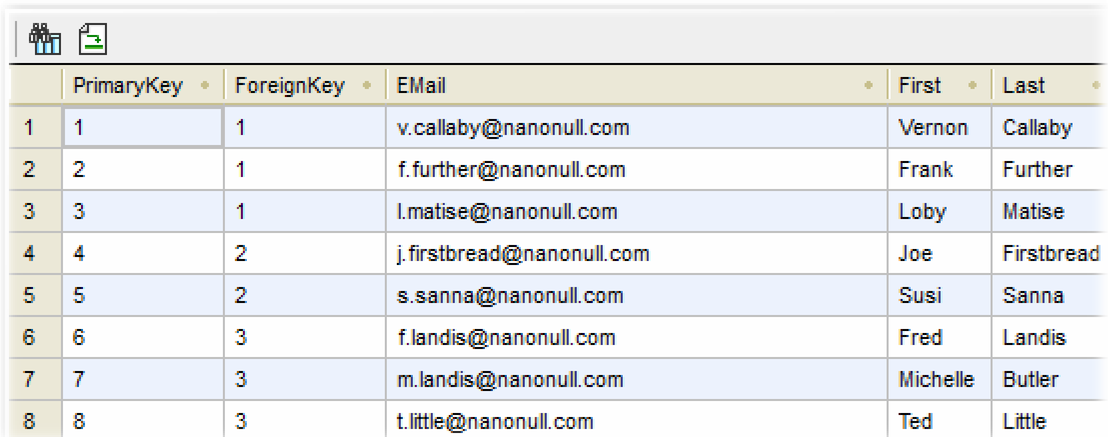
  If you selected multiple indexes, the names or paths are printed on separate lines, separated by commas.

If the database has support for XML Schemas, the following options are available for every schema displayed under the "XML Schemas" folder:

- **View in XMLSpy**: Opens the database schema in XMLSpy, provided that the latter is installed.
- **Manage XML Schemas**: Opens a dialog box where you can add new or drop existing database XML schemas.

4.2.8.4 Copying, Sorting, and Searching the Query Results

The **Results** tab of the **DB Query** pane shows the recordset retrieved as a result of a database query.



	PrimaryKey	ForeignKey	EMail	First	Last
1	1	1	v.callaby@nanonull.com	Vernon	Callaby
2	2	1	f.further@nanonull.com	Frank	Further
3	3	1	l.matise@nanonull.com	Loby	Matise
4	4	2	j.firstbread@nanonull.com	Joe	Firstbread
5	5	2	s.sanna@nanonull.com	Susi	Sanna
6	6	3	f.landis@nanonull.com	Fred	Landis
7	7	3	m.landis@nanonull.com	Michelle	Butler
8	8	3	t.little@nanonull.com	Ted	Little

The toolbar buttons enable navigation between results and SQL statements and facilitate searching within the query results.

	**Find:** Searches a specific text within the displayed results. Press **F3** to go to the next occurrence of the search term.
	**Go to statement**: Jumps to the SQL Editor and highlights the SQL statement that produced the current result. This might be particularly useful when the SQL Editor contains multiple statements.

To select cells from the query results:

- Click a column header to select the entire column
- Click a row number to select the entire row
- Click individual cells. Holding down the **Ctrl** key while clicking allows you to make multiple selections. If a column or cell contains XML data then this data can also be copied.

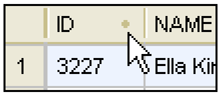
  Note: The context menu can also be used to select data, **Selection | Row | Column | All.**

To copy the selected cells to clipboard:

- Right-click and select **Copy selected cells** from the context menu.

To sort data:

- Right-click anywhere in the column to be sorted and select **Sorting | Ascending** or **Descending**
- Click the sort icon in the column header



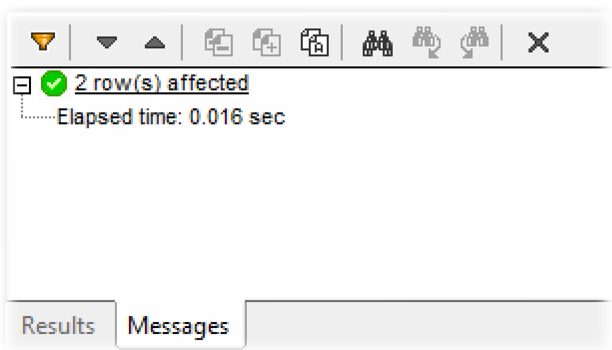
The data is sorted according to the contents of the sorted column.

To restore the default sort order:

- Right-click anywhere in the table and choose **Sorting | Restore default** from the context menu.

4.2.8.5 Viewing the Status of Executed Queries

The **Messages** tab of the **DB Query** pane provides specific information about the last executed SQL statement and reports errors or warning messages.



You can use different filters to customize the view of the **Messages** tab or use the **Next** and **Previous** buttons to browse data row by row. The buttons at the top are used to navigate the messages, copy text to clipboard, and hide certain parts of the message. These options are also available in the context menu, when you right-click anywhere inside the **Messages** tab.

	Filter: Opens a pop-up menu from where you can filter out the individual message types (**Summary**, **Success**, **Warning**, **Error, Autoinsertion, Progress**). "Autoinsertion" refers to those messages that may be triggered when SQL statements or SQL constructs are inserted automatically in SQL Editor. "Progress" messages report the database connection result, as well as the outcome of SQL parsing and data structure loading. You can check all or none of these options with a single mouse click by selecting either **Check All** or **Uncheck All** from the pop-up menu.
	Next: Jumps to and highlights the next message.
	Previous: Jumps to and highlights the previous message.
	Copy selected message to the clipboard
	Copy selected message including its children to the clipboard

	**Copy all messages to the clipboard**
	**Find**: Opens the **Find** dialog box.
	**Find previous**: Jumps to the previous occurrence of the string specified in the **Find** dialog box.
	**Find next**: Jumps to the next occurrence of the string specified in the **Find** dialog box.
	**Clear**: Removes all messages from the Message tab of the SQL Editor window.

4.2.9 Stored Procedures

Stored procedures are programs that are hosted and run on a database server. Stored procedures can be called by client applications and they are often written in some extended dialect of SQL. Some databases support also implementations in Java, .NET CLR, or other programming languages.

Typical uses of stored procedures include querying a database and returning data to the calling client, or performing modifications to the database after additional validation of input parameters. Stored procedures can also perform other actions outside the database, such as sending e-mails.

A stored procedure may have zero or more input and output parameters, and may optionally return zero or more recordsets, in addition to the default return value. Consequently, in MapForce, you can call a stored procedure in various ways:

- Call a stored procedure in order to retrieve data, as if it were a source component on the mapping. This is applicable for procedures that do not take input parameters. When the mapping runs, the procedure is called, and it returns some recordset or output parameters. You can map the recordset, or the output parameters, or both, to any other data type supported by MapForce. For an example, see Stored Procedures as Data Source [395].
- Call a stored procedure as a function-like call, with parameters. In this case, you supply all required input parameters from the mapping, and you can also map the returned recordset, or the output parameters, or both, to some other target supported by MapForce. For an example, see Stored Procedures with Input and Output [398].
- Call a stored procedure as if it were a target component on the mapping. The typical use case is calling a stored procedure with parameters in order to modify the database (for example, insert a record). This approach is suitable if you do not need any output from the stored procedure. Also, in this approach you can execute the stored procedure within a database transaction that can be rolled back in case of an error. For an example, see Stored Procedures in Target Components [402].

There are also cases where you may need to call stored procedures or perform actions on database tables in a specific order (first insert, then update, and so on). For example, you may need to pass the output parameter of a stored procedure to another stored procedure. Or you may need to combine data returned by a stored procedure with data from a table. Such actions are possible with the help of local relations defined in MapForce, even when the underlying database does not enforce primary/foreign key relationships between tables. For more information, see Stored Procedures and Local Relations [405].

Note: To illustrate how MapForce implements stored procedures, this chapter uses Microsoft SQL Server 2016 and the "AdventureWorks 2016" database. The latter can be downloaded from https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks.

Support notes

- Stored procedures can be used only in the BUILT-IN execution engine. Code generation in C++, C#, or Java is not supported.
- User-defined types, cursor types, variant types and many "exotic" database-specific data types (such as arrays, geometry, CLR types) are generally not supported as input or output parameter types.
- Procedure and function overloading (multiple definitions of routines with the same name and different parameters) is not supported.
- Some databases support default values on input parameters, this is currently not supported. You cannot omit input parameters in the mapping to use the default value.
- Stored procedures returning multiple recordsets are supported depending on the combination of driver and database API (ODBC/ADO/ADO.NET/JDBC). Only procedures that return the same number of recordsets with a fixed column structure are supported.
- Whenever possible, use the latest version of the database native driver maintained by the database vendor. Avoid using bridge drivers, such as ODBC to ADO Bridge, or ODBC to JDBC Bridge.
- You can optionally enable database transactions for stored procedures that are called as data target, see Stored Procedures in Target Components 402. Transactions are not supported for stored procedures that are called as a data source (without input parameters), or those that are called like a function (with both input and output).

The following table lists the database-specific support notes.

Database	Support notes
Access	• Stored procedures in Microsoft Access databases have very limited functionality and are not supported in MapForce.
DB2	• Supported in MapForce: stored procedures, scalar functions, table-valued functions. • Return values from DB2 stored procedures are not supported because they cannot be read via the database APIs used in MapForce. • Row-valued functions (RETURNS ROW) are not supported. • It is recommended to install at minimum "IBM_DB2 9.7 Fix Pack 3a" to avoid a confirmed JDBC driver issue when reading errors/warnings after execution. This also fixes an issue with the ADO provider that causes one missing result set row.
Firebird	• Supported in MapForce: stored procedures, table-valued functions
Informix	• Supported in MapForce: stored procedures, table-valued functions.
MariaDB	• Supported in MapForce: stored procedures, scalar functions
MySQL	• Supported in MapForce: stored procedures, scalar functions • MySQL includes complete support for stored procedures and functions starting with version 5.5. If you are using an earlier version, functionality in MapForce is limited.
Oracle	• Supported in MapForce: stored procedures, scalar functions, table-valued functions. This includes standalone stored procedures and functions as well as those defined inside an Oracle package.

Database	Support notes
	• It is recommended to use a native Oracle driver instead of the **Microsoft OLE DB Provider for Oracle**. • Oracle has a special way to return result sets to the client by using output parameters of type REF CURSOR. This is supported by MapForce for stored procedures, but not for functions. The names and number of recordsets is therefore always fixed for Oracle stored procedures.
PostgreSQL	• Supported in MapForce: scalar functions, row-valued functions, table-valued functions. • In PostgreSQL, any output parameters defined in a function describe the columns of the result set. This information is automatically used by MapForce - no detection by execution or manual input of recordsets is needed. Parameters of type `refcursor` are not supported.
Progress OpenEdge	• Supported in MapForce: stored procedures.
SQL Server	• Supported in MapForce: stored procedures, scalar functions, table-valued functions. • It is recommended to use the latest **SQL Server Native Client** driver instead of the **Microsoft OLE DB Provider for SQL Server**. • The ADO API has limited support for some data types introduced with SQL Server 2008 (`datetime2`, `datetimeoffset`). If you encounter data truncation issues with these temporal types when using ADO with the SQL Server Native Client, you can set the connection string argument `DataTypeCompatibility=80` or use ODBC. • SQL Server Procedures have an implicit return parameter of type `int null`, which is available for mapping. If the procedure omits a RETURN statement, the resulting value is 0.
SQLite	• SQLite does not use stored procedures.
Teradata	• Supported in MapForce: stored procedures, macros. • Scalar functions, aggregate functions and table functions are not supported • Known issue: The Teradata ODBC driver refuses to populate output parameter values after a procedure call.

4.2.9.1 Adding Stored Procedures to the Mapping

On the mapping area, stored procedures are shown as part of the database component where they belong. In order for stored procedures to be visible on the database component, you must explicitly select them when adding the database component to the mapping, as shown below. In this example, we connect to the "AdventureWorks" database running on SQL Server. Instructions are similar for other database types.

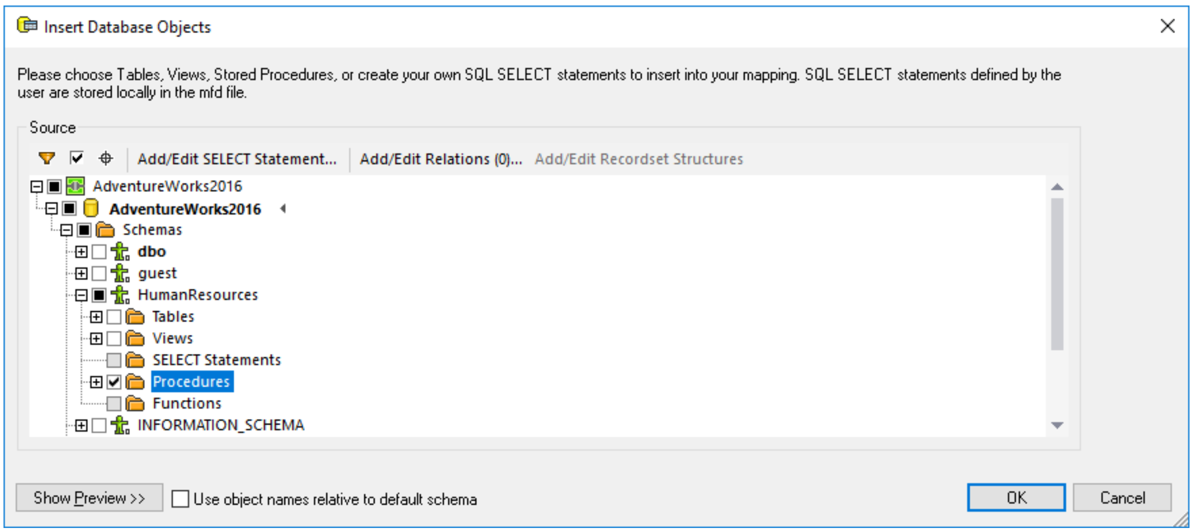
In case of Oracle databases, stored procedures or functions may be standalone or part of Oracle packages. You can add both categories to the mapping. The stored procedures or functions belonging to a package appear under the respective package name on the "Insert Database Objects" dialog box illustrated below.

To add stored procedures to the mapping:

1. Do one of the following:

   - On the **Insert** menu, click **Database**.
   - Click the **Insert Database** ( ) toolbar button.

2. Follow the database wizard steps until you get to the "Insert Database Objects" dialog box. For detailed instructions applicable to each database type, see Database Connection Examples 195.

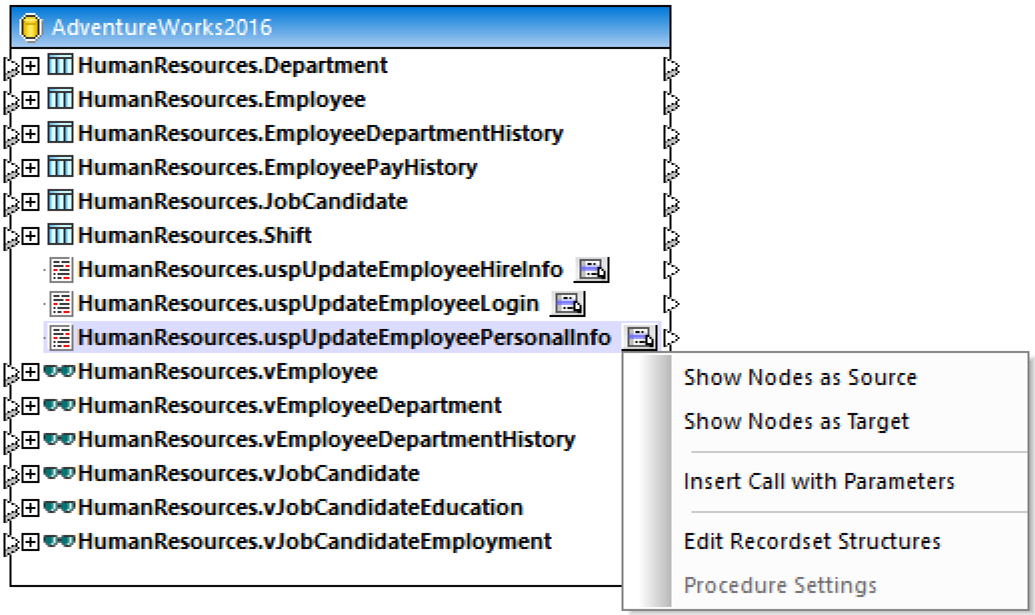


3. Select the check boxes next to the database objects that you need to be displayed on the mapping, and click **OK**. In this example, we have selected all the tables, views, and stored procedures available in the "HumanResources" schema.

Notes

- You can change the selected objects at any time later, by right-clicking the title bar of a database component, and selecting **Add/Remove/Edit Database Objects** from the context menu.
- Your database user account must have rights to view and execute stored procedures in the database.

The database component is now added to the mapping. Notice that stored procedures are identified by the icon. In addition, tables, views and procedures are sorted alphabetically in the database component.



The **Show Context Menu** button next to each stored procedure lets you configure how the stored procedure is to be called, and other procedure-related settings, as follows:

Option	Usage
Show Nodes as Source	Select this option if you want to call a stored procedure *without parameters* in order to retrieve data from a database and map it to another component supported by MapForce (XML, text, EDI, and so on). For an example, see Stored Procedures as Data Source 395.
Show Nodes as Target	Select this option if you want to call a stored procedure in order to modify the database or perform another specific action where you don't need the output of the stored procedure. For an example, see Stored Procedures in Target Components 402.
Insert Call with Parameters	Select this option if you want to call a stored procedure *with parameters* and want to map the returned data to another component supported by MapForce. For an example, see Stored Procedures with Input and Output Parameters 398.
Edit Recordset Structures	Applicable for stored procedures that return recordsets. Select this option to execute the stored procedure once, so that MapForce can determine the structure of the returned recordset and display it on the mapping. Alternatively, if you don't want to execute the stored procedure at design time, you can define the recordset structure manually.
Procedure Settings	Applicable only for stored procedures that were configured as "target" (that is, those that update the database). Select this option to configure additional procedure-related settings, such as running a custom SQL query before calling the procedure, or enabling database transactions.

4.2.9.2 Stored Procedures as Data Source

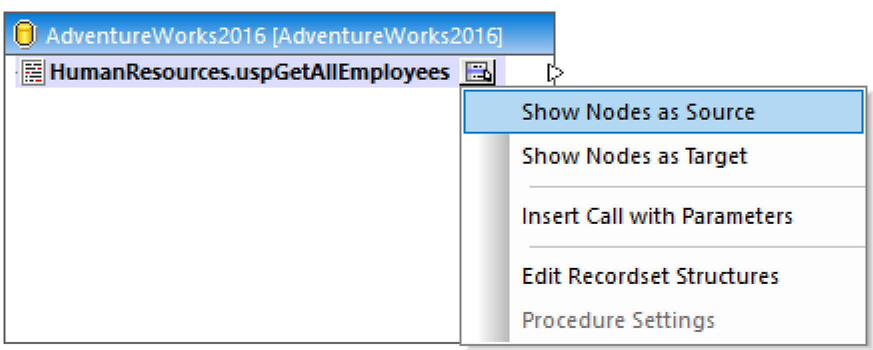
This example shows you how to call a procedure that takes no input parameters and just retrieves some data from the database. In this scenario, the stored procedure acts as a source component to the mapping, and you can map data retrieved by it to any other target component supported by MapForce. If you need to call a stored procedure with input parameters, see Stored Procedures with Input and Output [398].

Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** tab of MapForce, see Browsing and Querying Databases [376]. In either case, make sure that your database user account has permission to create stored procedures.

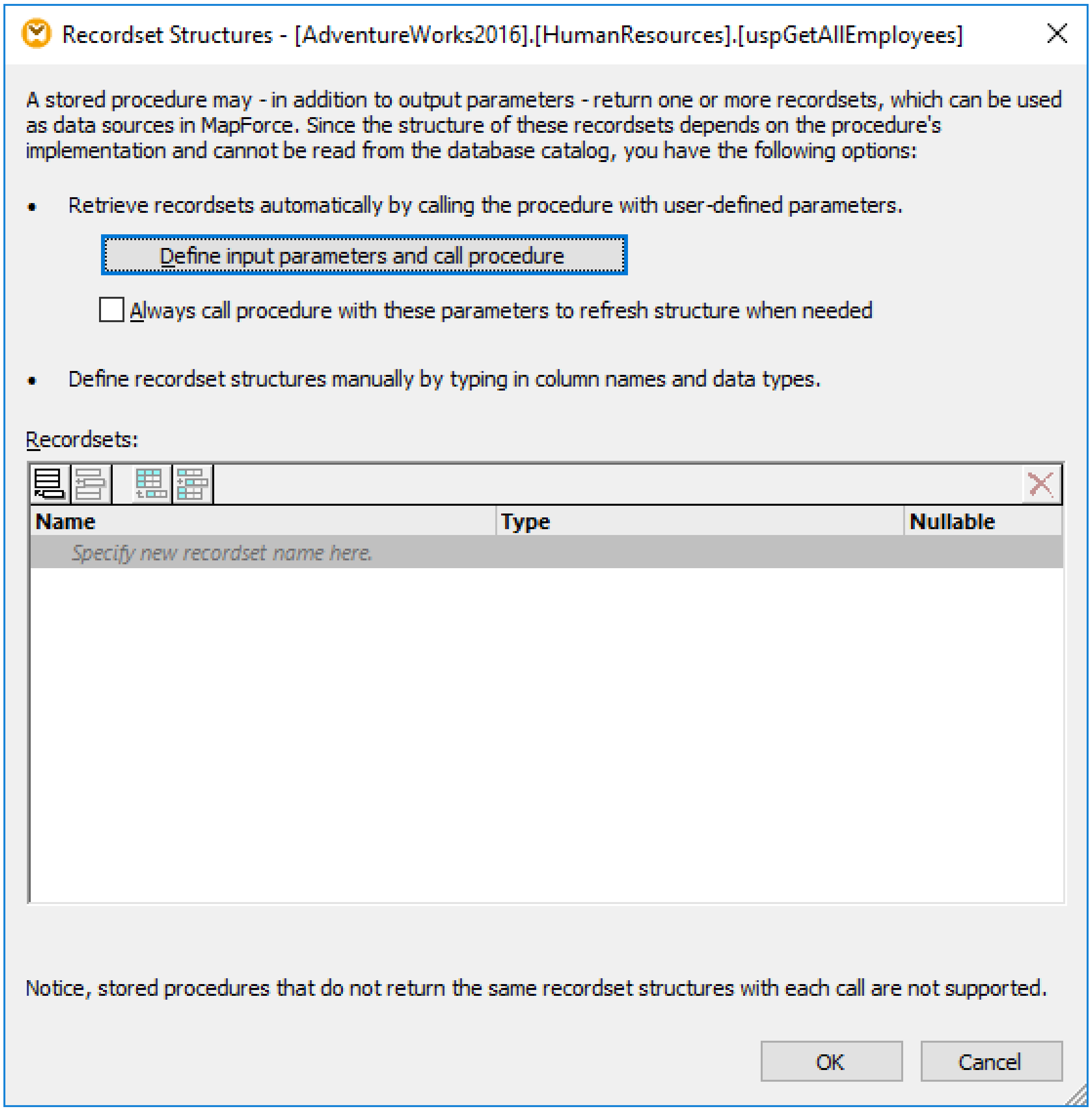
```
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment
```

The stored procedure above returns employee information from the **vEmployeeDepartment** view. The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping [392]. Make sure that your database user account has permission to view and execute stored procedures.
2. Click the **Show Context Menu** button next to the stored procedure and select **Show Nodes as Source**.

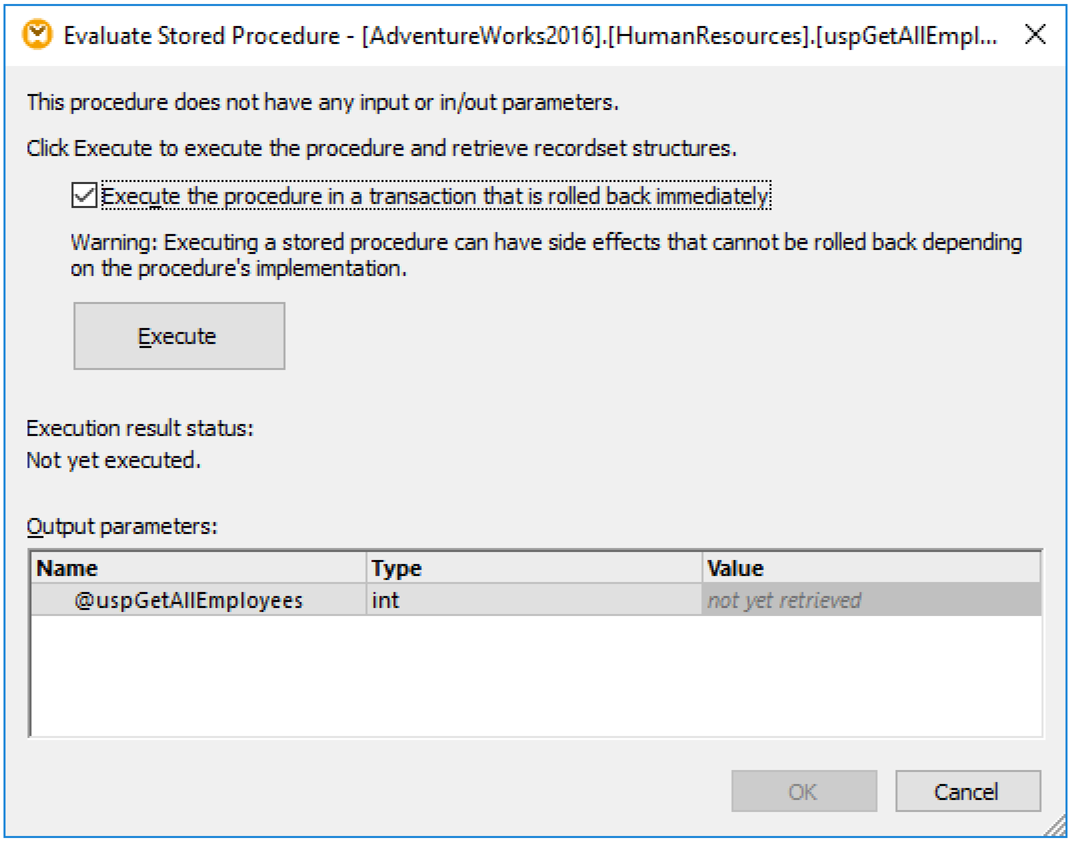


3. Click the **Show Context Menu** button again and select **Edit Recordset Structures**. The "Recordset Structures" dialog box appears.

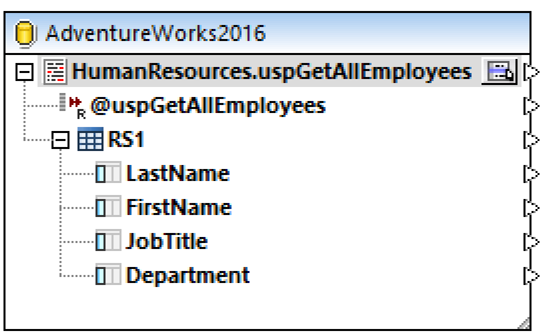


Calling a stored procedure at design time may have side effects (depending on the procedure implementation). If you do not want to execute the stored procedure at design time, do not click **Execute**, as further described in subsequent steps. Instead, define the expected recordset in the "Recordset Structures" dialog box, by adding recordsets and their associated columns manually. Use the **Add recordset** or **Add column** buttons in the "Recordset Structures" dialog box.

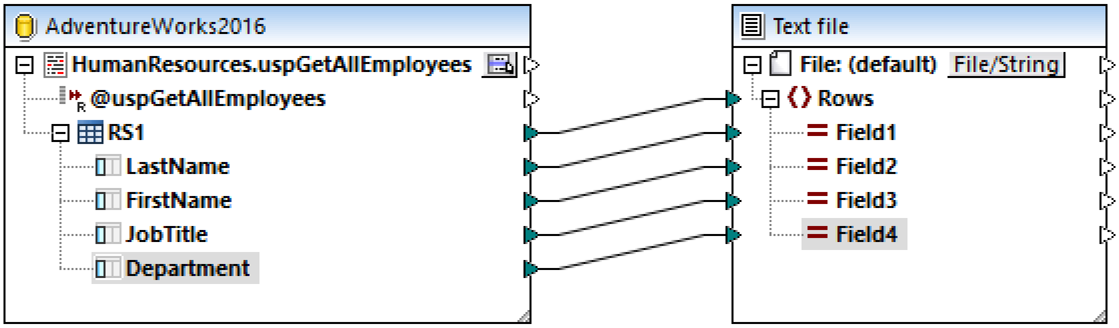
4. Click **Define input parameters and call procedure**. The "Evaluate Stored Procedure" dialog box appears.



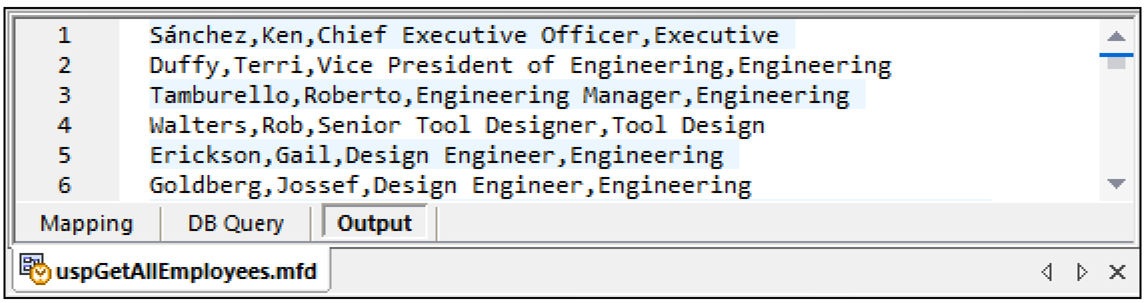
5. Click **Execute**, and then click **OK**. The recordset structure ("RS1") is now visible both on the "Recordset Structures" dialog box and on the mapping.



6. At this stage, you can add a target component where the retrieved data will be written. In this example, data will be written to a CSV file. On the **Insert** menu, click **Text File**, and add a CSV component to the mapping. For more information, see CSV and Text Files 563.



You can now preview the mapping. Click the **Output** button and observe the mapping result in the **Output** pane, for example:



4.2.9.3 Stored Procedures with Input and Output

This example shows you how to call a procedure that takes input parameters and also retrieves some output from the database. In this scenario, the stored procedure is called similar to a Web service, or a function, and you can map data retrieved by it to any other target component supported by MapForce.

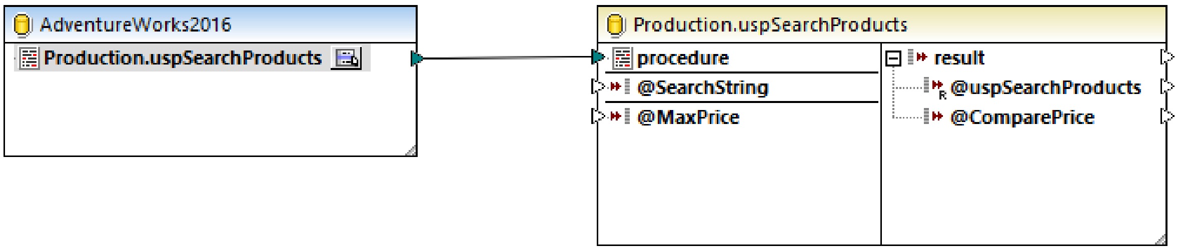
Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** tab of MapForce, see Browsing and Querying Databases[376]. In either case, make sure that your database user account has permission to create stored procedures.

```
CREATE PROCEDURE Production.uspSearchProducts
   @SearchString nvarchar(50)
   ,@MaxPrice money
   ,@ComparePrice money OUTPUT
AS
BEGIN
   SET NOCOUNT ON
   SELECT pr.[Name], pr.ListPrice FROM [Production].[Product] pr
   WHERE pr.[Name] like @SearchString AND  pr.ListPrice < @MaxPrice

   SET @ComparePrice = @MaxPrice
   RETURN @ComparePrice
END
```

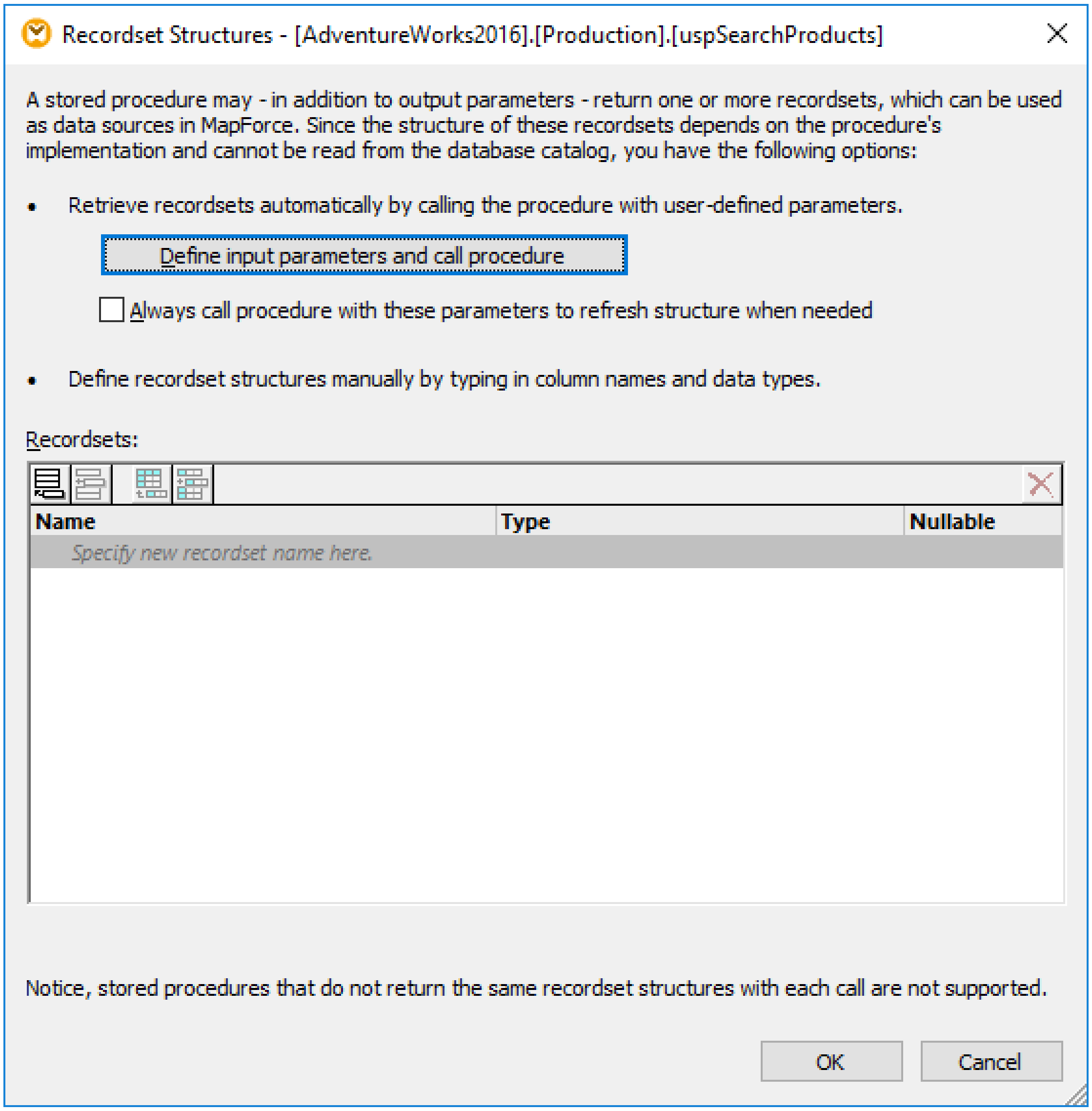

The stored procedure above retrieves a recordset containing product information. It takes as input two parameters: a string with the product name (`@SearchString`) and the maximum product price (`@MaxPrice`). In addition to the recordset and the default return parameter, it also retrieves an output parameter (`@ComparePrice`).

The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping 392. Make sure that your database user account has permission to view and execute stored procedures.
2. Click the **Show Context Menu** button next to the stored procedure and select **Insert Call with Parameters**. The stored procedure now appears in a separate component on the mapping, where the left side lists the input parameters, and the right side contains the return and the output parameters.

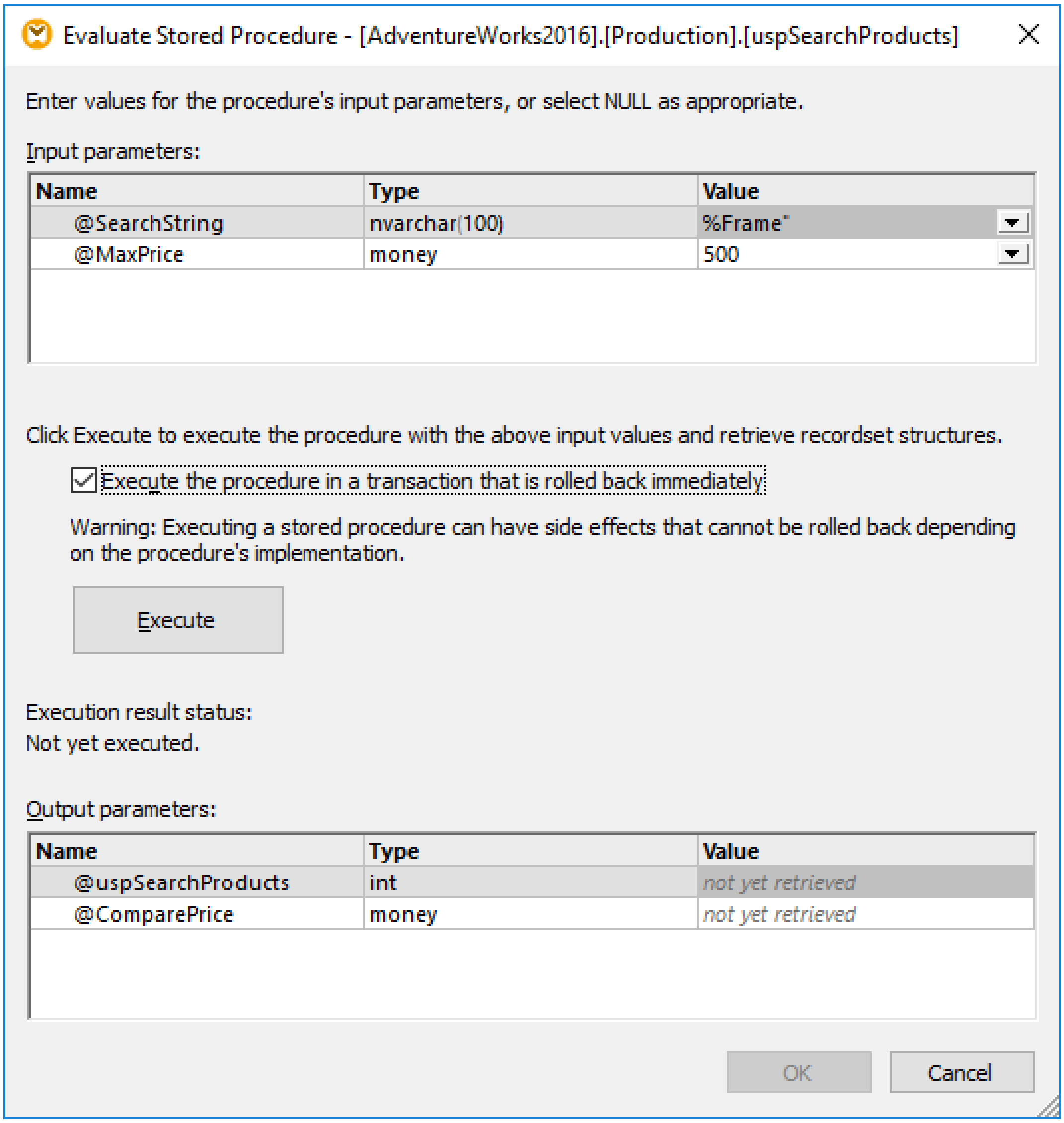


3. Click the **Show Context Menu** button again, and select **Edit Recordset Structures**. This is necessary so as to provide to MapForce information about the structure of the recordset returned by the procedure. The "Recordset Structures" dialog box appears.

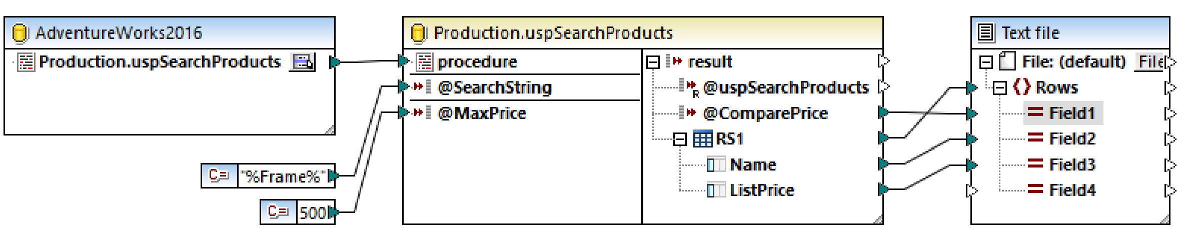


> Calling a stored procedure at design time may have side effects (depending on the procedure implementation). If you do not want to execute the stored procedure at design time, do not click **Execute**, as further described in subsequent steps. Instead, define the expected recordset in the "Recordset Structures" dialog box, by adding recordsets and their associated columns manually. Use the **Add recordset** or **Add column** buttons in the "Recordset Structures" dialog box.

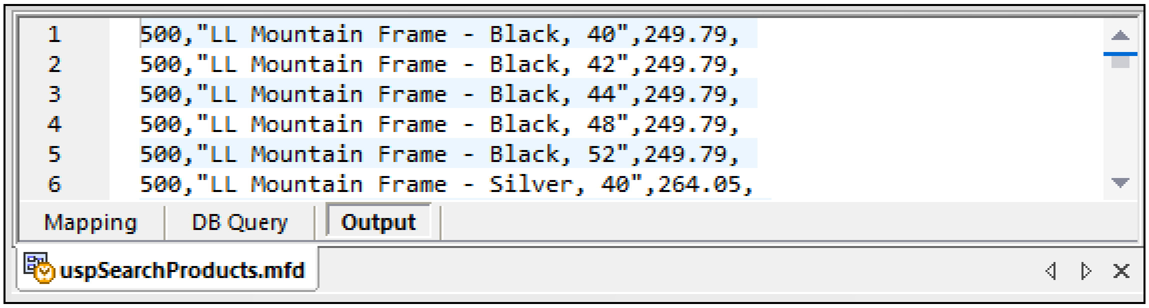
4. Click **Define input parameters and call procedure**. The "Evaluate Stored Procedure" dialog box appears.



5. Fill in the parameter values as shown above, and click **Execute**.
6. Click **OK**. The recordset structure ("RS1") is now visible both on the "Recordset Structures" dialog box and on the mapping.
7. At this stage, you can add a target component where the retrieved data will be written. In this example, data will be written to a CSV file. On the **Insert** menu, click **Text File**, and add a CSV component to the mapping, see also CSV and Text Files [563]. Next, draw the mapping connections as illustrated below. Notice that the procedure's input parameters are supplied by means of constants. For more information about constants, see Add a Constant to the Mapping [905].



You can now preview the mapping. Click the **Output** button and observe the mapping result in the **Output** pane, for example:



4.2.9.4 Stored Procedures in Target Components

This example shows you how to call a procedure that takes input parameters and updates a database. Calling a procedure this way makes it possible to enable transactions and roll the action back in case of an error, or add a custom SQL statement to be executed before the procedure is called. This scenario implies that the stored procedure acts like a target component in MapForce and you are not interested in the output returned by it. For an example that illustrates how to pass parameters and also map data returned by a stored procedure, see Stored Procedures with Input and Output [398].

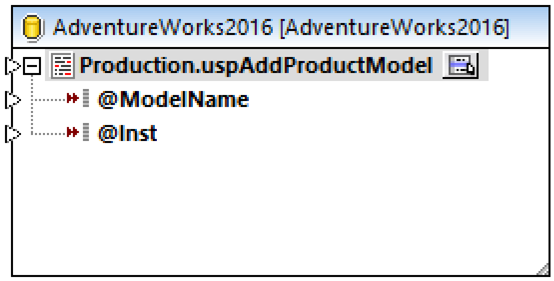
Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** tab of MapForce, see Browsing and Querying Databases [376]. In either case, make sure that your database user account has permission to create stored procedures.

```
CREATE PROCEDURE Production.uspAddProductModel
   @ModelName nvarchar(50)
   ,@Inst xml
AS
BEGIN
INSERT INTO [Production].[ProductModel]
           ([Name]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           ,@Inst
           ,NEWID()
           ,GETDATE())
END
```

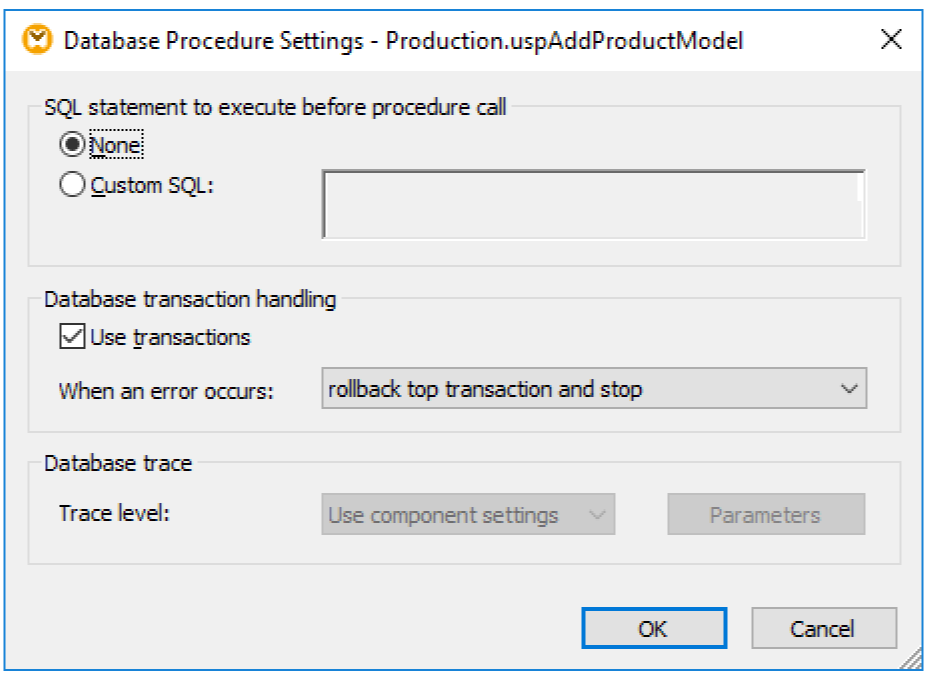
The stored procedure above takes two parameters (`@ModelName`, `@Inst`) as input and inserts the corresponding values into the `ProductModel` table of the AdventureWorks database, along with some database-generated data.

The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping [392]. Make sure that your database user account has permission to view and execute stored procedures.
2. Click the **Show Context Menu** button next to the stored procedure and select **Show Nodes As Target**. The stored procedure now appears as target component on the mapping, where the left side lists the input parameters.

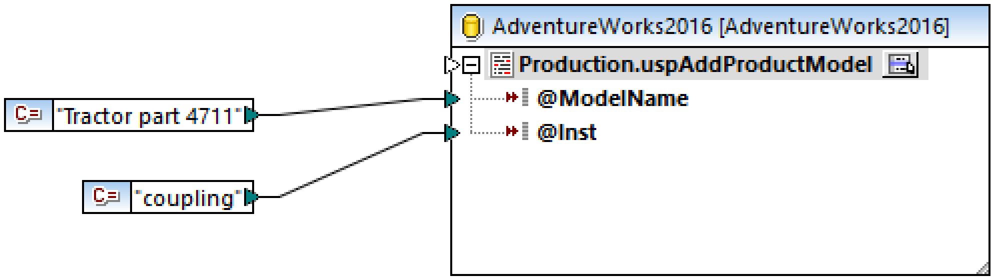


3. Click the **Show Context Menu** button again, and select **Procedure Settings**. This optional step enables you to execute the stored procedure inside a transaction that can be rolled back. You can also add a custom SQL statement to be executed before the procedure is called.
4. Select the **Use Transactions** check box.

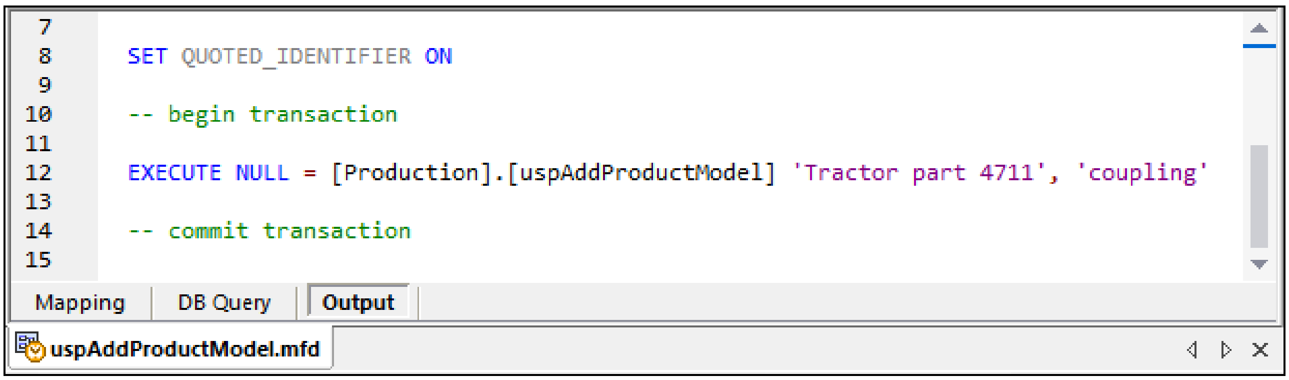


Note: In this example, database tracing is disabled at database component level and no tracing is set to take place. However, you can enable database tracing for stored procedures if necessary. For more information, see Database Tracing and Error Logging [427].

5. Add the source component that provides data to be inserted into the database. In this example, the source data is supplied by constants; however, any other source component supported by MapForce could act as input. For more information about constants, see Add a Constant to the Mapping [905].



Since this mapping updates a database, you do not preview its output directly like with other mappings. Instead, click the **Output** button to display the pseudo-SQL containing hints about how the database will be modified. If you enabled transactions, these will occur as indicated by the comments.



> The pseudo-SQL displayed in the **Output** pane does not show the actual transaction commands, only hints (as comments). The actual SQL commands are sent to the underlying database API, however.

To run the mapping against the database, do one of the following:

- On the **Output** menu, click **Run SQL-Script**.
- Click the **Run SQL-Script** toolbar button.

Stored procedures and duplicate inputs

If you need to map data from multiple sources on the mapping to the same stored procedure, you can duplicate the stored procedure so that it accepts multiple inputs. To do this, right-click the stored procedure item on the component and select **Add duplicate input** from the context menu, see also Duplicating Input 83. When the mapping runs, such duplicate stored procedures will be called once for each duplicated input.

Note that the **Add duplicate input** command is disabled for the stored procedure parameters, because each parameter is an atomic value (and could also be "nullable").

4.2.9.5 Stored Procedures and Local Relations

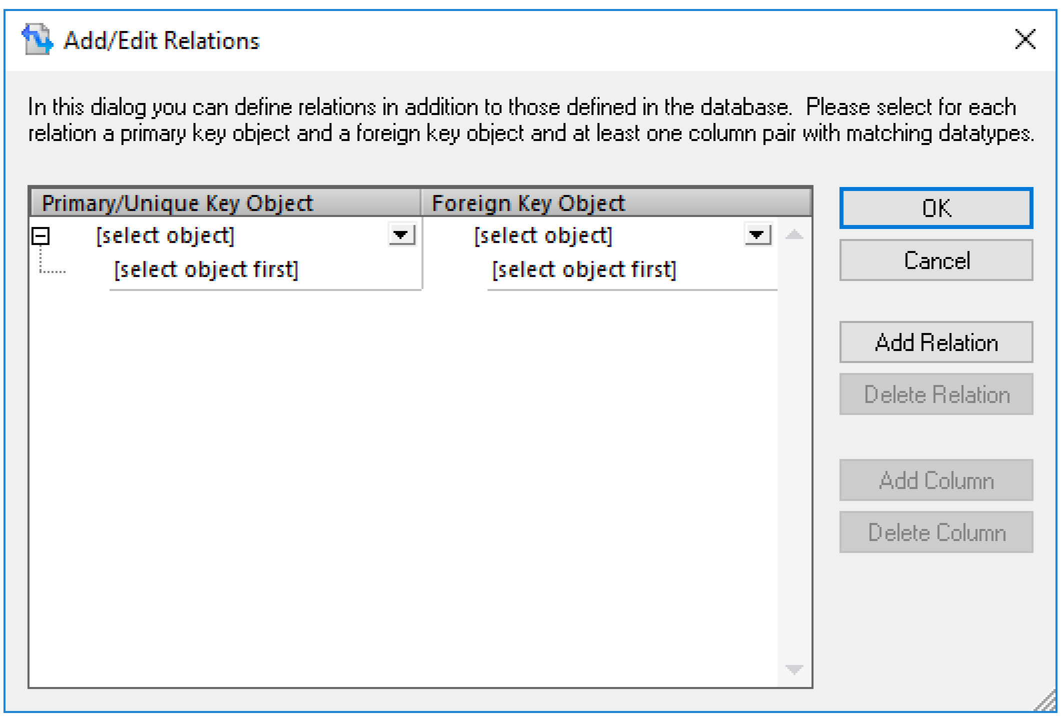
Local relations are logical relationships between database fields that you can create in MapForce, saving you the need to change the underlying database, see also Defining Local Relationships [267]. You can define local relations not only for database fields, but also for stored procedures as well, both in source and target components.

In source components, local relations make it easy to read data from related objects, for example, read IDs from a database table and call a stored procedure with each of these IDs to retrieve related information. It is also possible to call a stored procedure with data retrieved from another procedure.

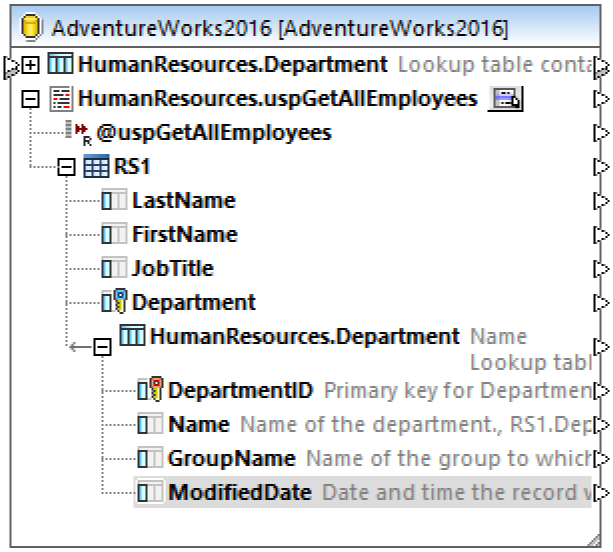
In target components, local relations enable you to define a hierarchical order in which multiple related procedures are to be called. For example, you can first call a stored procedure that creates an ID value, and another one that inserts related information into a table. It is also possible to mix stored procedures and tables in local relations. For example, you can perform the insert directly on a related table instead of calling another procedure, see Using Stored Procedures to Generate Keys [410].

To create a local relation:

1. Right-click the title bar of a database component and select **Add/Remove/Edit Database Objects** from the context menu. The "Add/Remove/Edit Database Objects" dialog box opens.
2. Click **Add/Edit Relations**.
3. Click **Add Relation** and select the objects between which you want to create the relationship.



As illustrated above, a local relation consists of a **primary/unique key** object and a **foreign key** object. Think of it as a parent-child relationship. On the mapping component, the object (table, view, procedure, and so on) where the primary/unique key is will appear as a parent while the object where the foreign key is will appear nested under it. For example, in the database component illustrated below, a local relation was defined between a recordset column (**RS1.Department**) and a table column (**Department.Name**). Consequently, the **Department** table appears as a child of the stored procedure on the mapping. This example is discussed in more detail in Local Relations in Source Components [406].



The following table lists all the possible fields between which you can define local relations. Mixed relationships are possible (for example, mapping the output of a stored procedure to a database column). The fields taking part in the relationship must have the same, or a compatible, data type.

Primary/unique key	**Foreign key**
• Column of a database table or view • Output parameter or return value of a stored procedure, see also Stored Procedures [390] • Column of a recordset returned by a stored procedure* • Column of a user-defined SELECT statement, see also SQL SELECT Statements as Virtual Tables [357].	• Column of a database table or view • Input parameter of a stored procedure • Input parameter of a user-defined SELECT statement

** Applicable if the stored procedure is called either as data source (without parameters) or as a function (with input and output parameters). In order for the recordset to become available for selection, you must execute the stored procedure once, to retrieve the recordset.*

4.2.9.6 Local Relations in Source Components

This example shows you how to combine data returned by a stored procedure with data from a table in the same database, with the help of local relations.

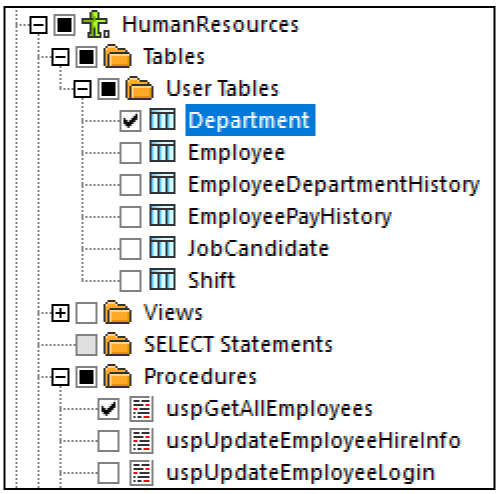
If you haven't done so already in a previous example, run the following script to create the demo stored procedure in the "AdventureWorks" database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** pane of MapForce, see Browsing and Querying

Databases[376]. In either case, make sure that your database user account has permission to create stored procedures.

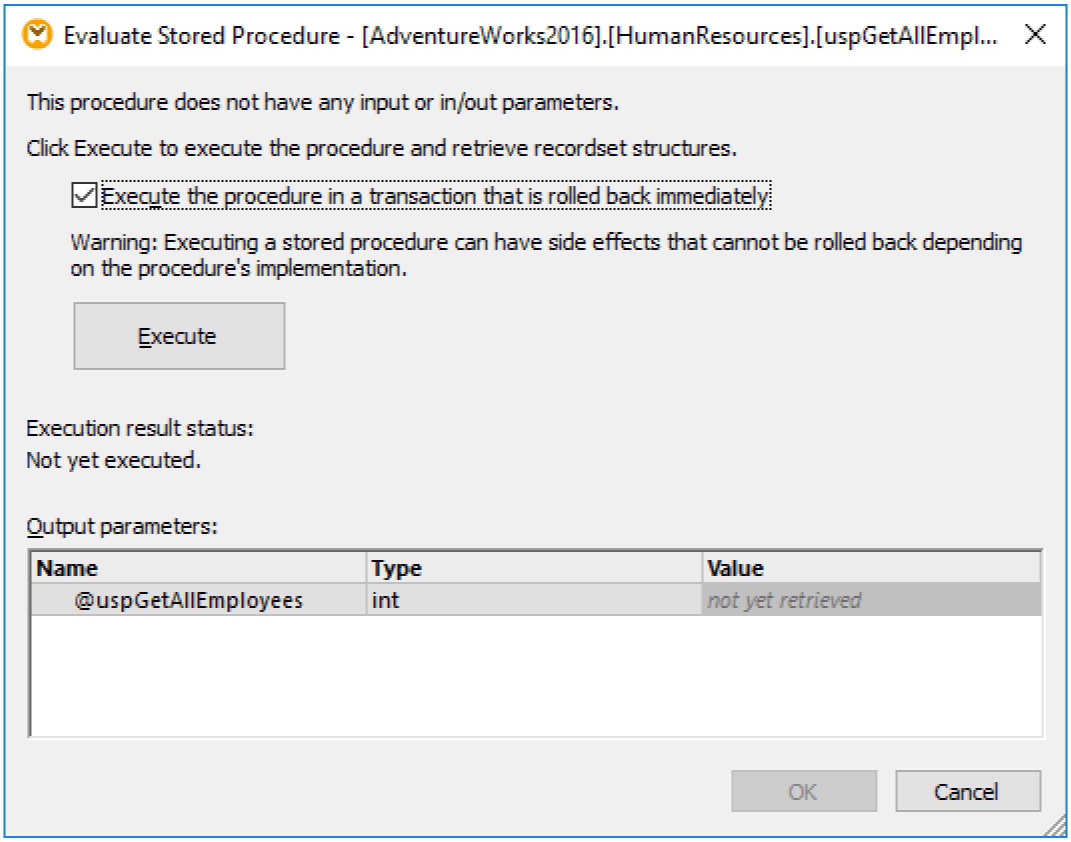
```
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment
```

The stored procedure above returns employee information from the **vEmployeeDepartment** view. The following steps show you how to create a mapping that consumes data returned by this procedure.

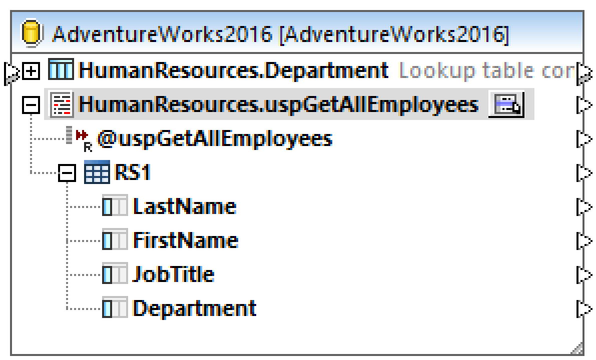
1. Connect to the "AdventureWorks" database from MapForce, as described in Adding Stored Procedures to the Mapping[392]. Make sure that your database user account has permission to view and execute stored procedures.
2. When prompted to choose database objects, select the **Department** table and the **uspGetAllEmployees** stored procedure.



3. Click the **Show Context Menu** button next to the stored procedure and select **Show Nodes as Source**.
4. Click the **Show Context Menu** button again and select **Edit Recordset Structures**. The "Recordset Structures" dialog box appears.
5. Click **Define input parameters and call procedure**. The "Evaluate Stored Procedure" dialog box appears.



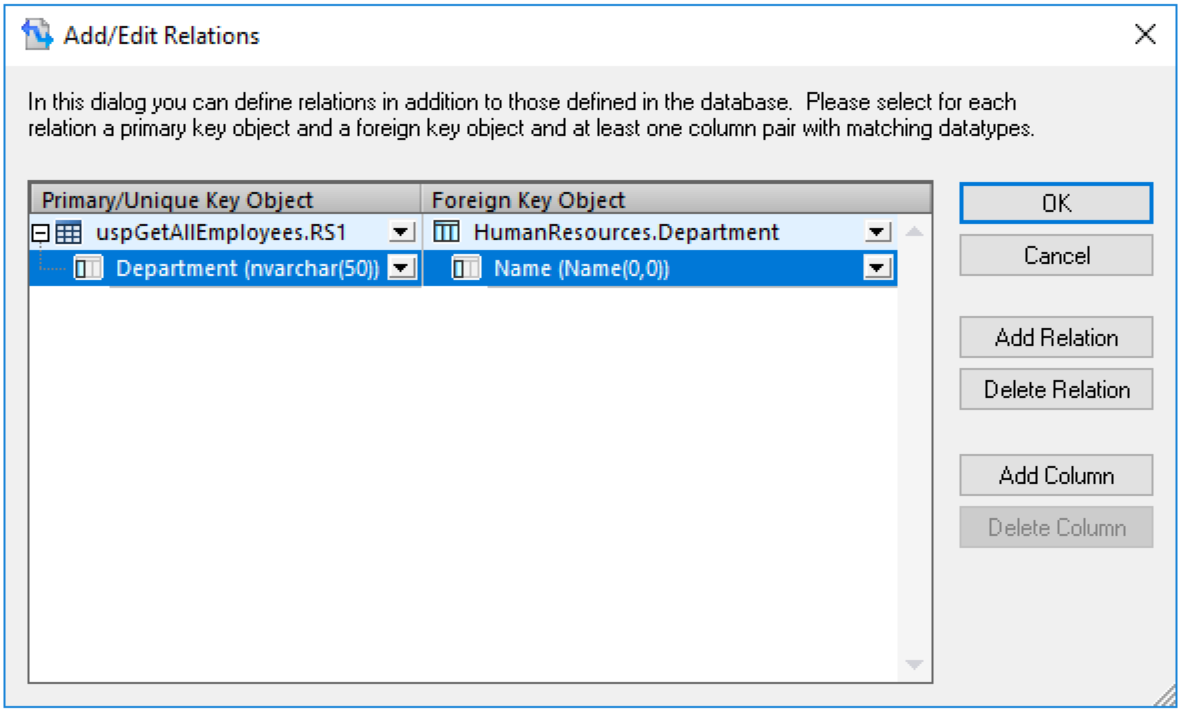
6. Click **Execute**, and then click **OK**. The recordset structure ("RS1") is now visible both on the "Recordset Structures" dialog box and on the mapping.



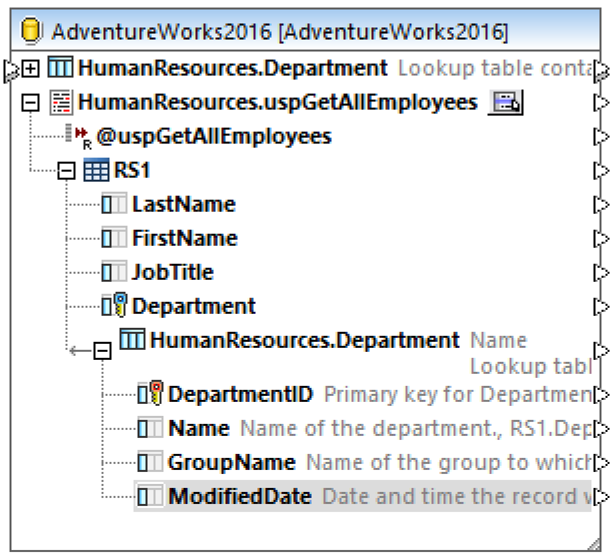
Define the local relationships

Let's now define a local relationship between the **Department** column of the returned recordset and the **Name** column of the **Department** table.

1. Right-click the title bar of the database component and select **Add/Remove/Edit Database Objects** from the context menu.
2. Click **Add/Edit Relations**, and then click **Add Relation**. Define the relationships as shown below.



3. Click **OK** to close the dialog box. Notice that the **Department** table has now become a child of the **RS1** recordset.

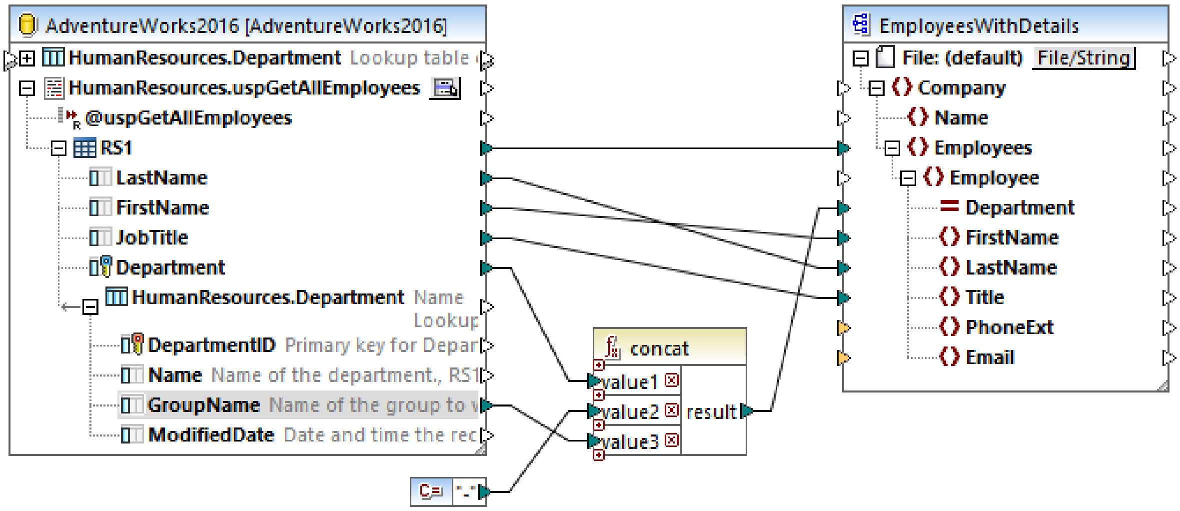


Completing the mapping

Thanks to the relationship that was just created, it is now possible to map data from the recordset combined with data from the table. For the scope of this example, let's write data to a target XML file, as follows:

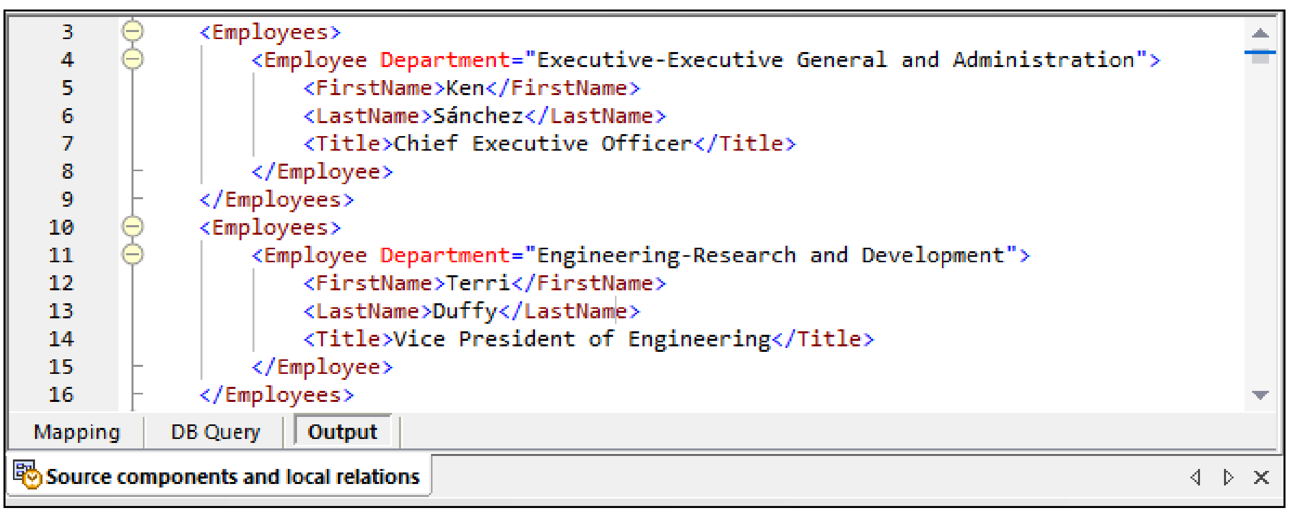
1. On the **Insert** menu, click **XML Schema/File** and select the following file: **<Documents>\Altova\MapForce2023\MapForceExamples\EmployeesWithDetails.xsd**.

2. When prompted to provide a sample XML instance file, click **Skip**.
3. Draw the mapping connections as shown below.



The mapping illustrated above writes data from the database to a target XML file. The source data is a combination of data extracted by the stored procedure with data extracted directly from a table. The mapping uses the **concat** [1057] function to produce a string that includes the department name, followed by a dash character, followed by the group name.

To preview the mapping, click the **Output** button and observe the mapping result in the **Output** pane, for example:



4.2.9.7 Using Stored Procedures to Generate Keys

This example shows you how to insert some key (ID) generated by a stored procedure into another table, with the help of local relations.

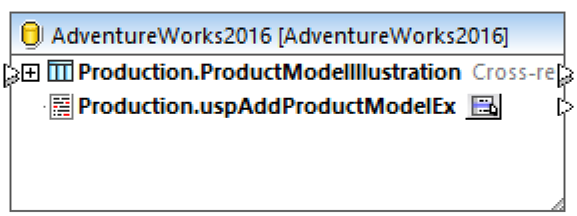
Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** tab of MapForce, see Browsing and Querying Databases 376. In either case, make sure that your database user account has permission to create stored procedures.

```
CREATE PROCEDURE Production.uspAddProductModelEx
  @ModelName nvarchar(50)
  ,@Inst xml
  ,@ProductModelID int OUTPUT
AS
BEGIN
INSERT INTO [Production].[ProductModel]
           ([Name]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           ,@Inst
           ,NEWID()
           ,GETDATE())
   SELECT @ProductModelID = SCOPE_IDENTITY()
END
```

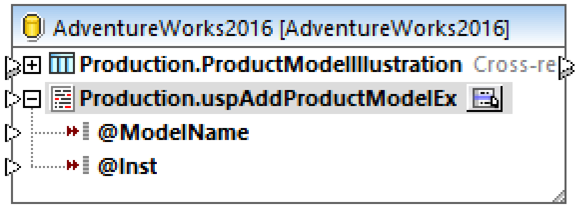
The stored procedure above takes two parameters (`@ModelName`, `@Inst`) as input and performs an `INSERT` operation into the **ProductModel** table. It then returns the generated `@ProductModelID` as output parameter. The requirement is to insert the `@ProductModelID` returned by the stored procedure into the **ProductModelIllustration** table.

The following steps show you how to create a mapping that satisfies the requirement above.

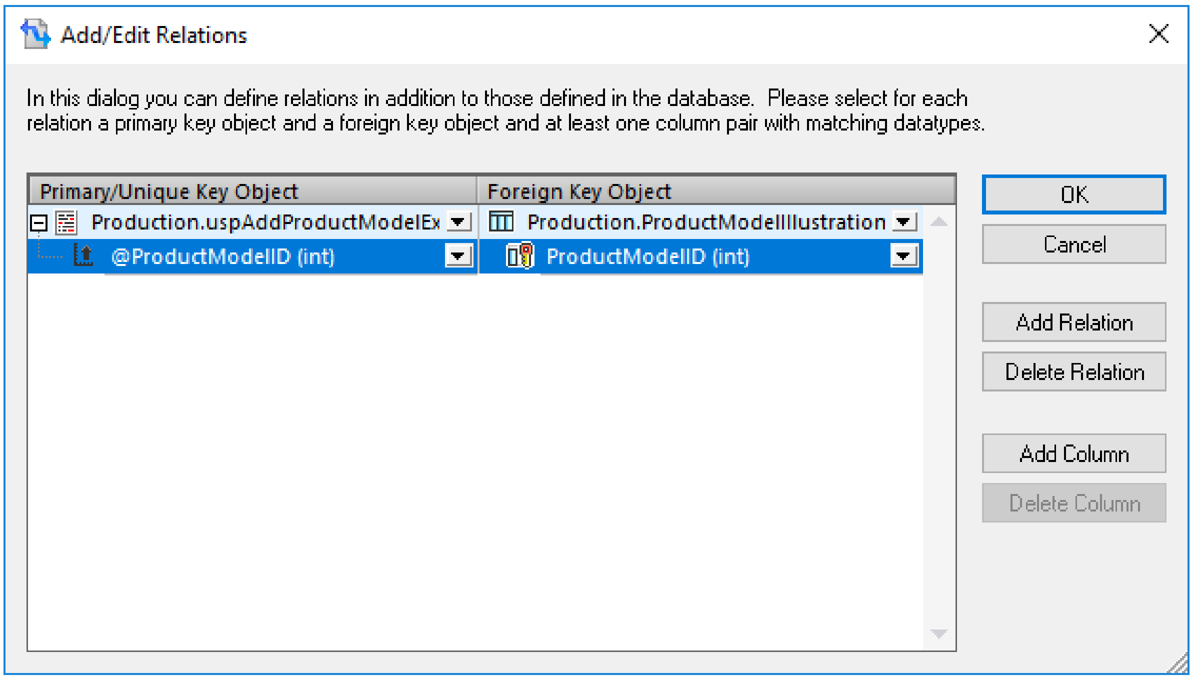
1. Connect to the "AdventureWorks" database from MapForce, as described in Adding Stored Procedures to the Mapping 392. Make sure that your database user account has permission to view and execute stored procedures.
2. When prompted to choose database objects, select the **ProductModelIllustration** table and the **uspAddProductModelEx** stored procedure.



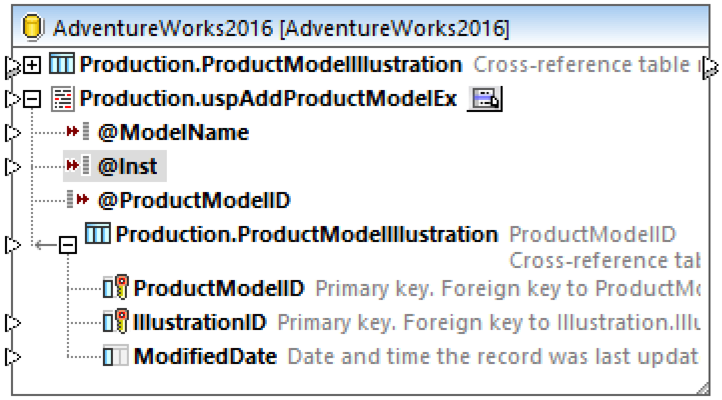
3. Click the **Show Context Menu** button next to the stored procedure and select **Show Nodes As Target**. The stored procedure now appears as target component on the mapping, where the left side lists the input parameters.



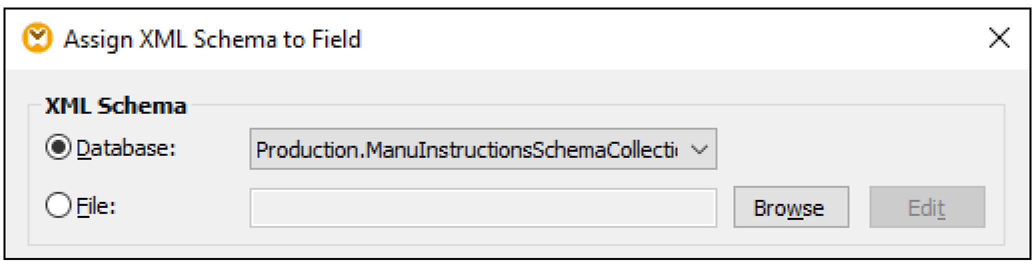
4. Optionally, if you want to execute the stored procedure inside a transaction, click the **Show Context Menu** button again, select **Procedure Settings**, and then select the **Use Transactions** check box. Defining the transaction for the stored procedure ensures that retrieving the key and inserting the record occur during the same transaction.
5. Right-click the title bar of the database component and select **Add/Remove/Edit Database Objects** from the context menu.
6. Click **Add/Edit Relations**, and then click **Add Relation**. Define the relationships as shown below.



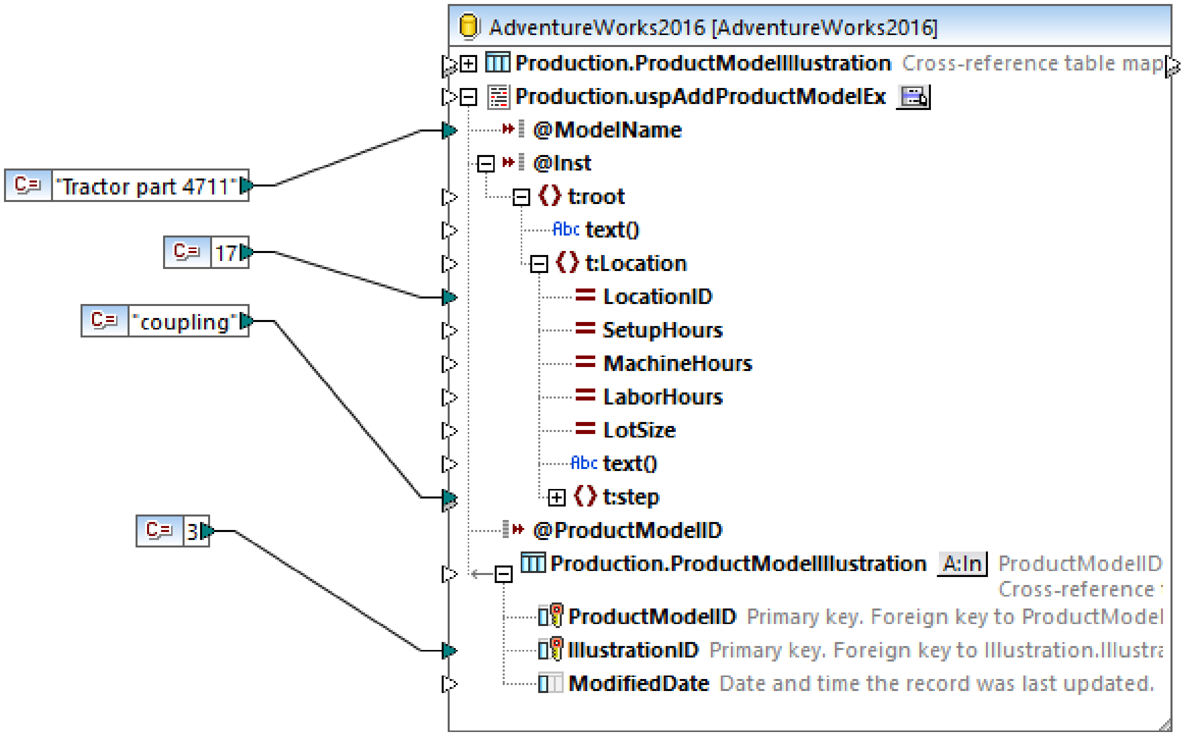
7. Click **OK** to close the dialog box. Notice that the **ProductModelIllustration** table now appears as a child of the stored procedure. The stored procedure output parameter (`@ProductModelID`) is displayed as an indicator that it will be used in the local relation, but it does not have any input or output connectors.



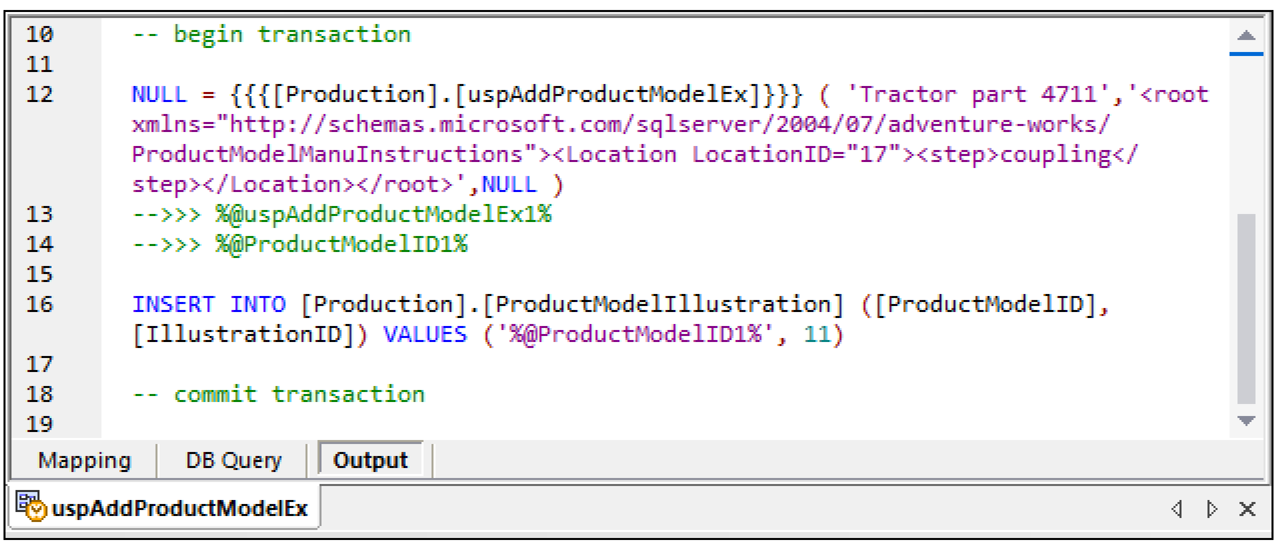
8. In this example, the `@Inst` parameter is of XML type. Right-click the `@Inst` parameter on the component and select **Assign XML Schema to Field** from the context menu. Next, select the **Production.ManuInstructionsSchemaCollection** schema from the database. When prompted to select a root element, leave the default value as is, and click **OK**. For more information about mapping data to database XML fields, see Mapping XML Data to / from Database Fields 366.



9. Add the source components that provides data to be inserted into the database. In this example, the source data is supplied by constants; however, any other source component supported by MapForce could act as input. For more information about constants, see Add a Constant to the Mapping 905.



Since this mapping updates a database, you do not preview its output directly like with other mappings. Instead, click the **Output** button to display the pseudo-SQL containing hints about how the database will be modified. If you enabled transactions, these will occur as indicated by the comments.



> The pseudo-SQL displayed in the **Output** pane does not show the actual transaction commands, only hints (as comments). The actual SQL commands are sent to the underlying database API, however.

To run the mapping against the database, do one of the following:

- On the **Output** menu, click **Run SQL-Script**.
- Click the **Run SQL-Script** toolbar button.

4.2.10 Handling Database Exceptions

When running mappings that modify a database, MapForce (or MapForce Server, or a MapForce-generated program) may potentially encounter various database-related errors. For example, the database account may not have enough privileges to perform a specific database action, or some database constraints might be in place that prevent the mapping from inserting or updating invalid data. The latter may happen, for example, when a table column is mandatory but the mapping does not supply a value for it.

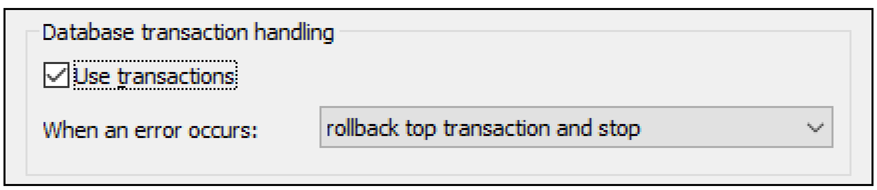
To make it possible to roll back data if database exceptions are encountered, you can configure the mapping to use database transactions. As further described below, you can enable transactions at database component level and at table action level.

The transaction rollback settings you configure in MapForce are preserved when you run the mapping either with MapForce Server or with a MapForce-generated program.

Transactions at database component level

To enable transactions at database component level, do the following:

1. Right-click the database component, and select **Properties** from the context menu (Alternatively, double-click the title bar of the database component). This opens the Database Component Settings dialog box.



2. Select the **Use transactions** check box.

Doing this encloses all changes done by the database component inside a transaction that will be rolled back in case of error. When an error occurs during mapping execution, the outcome depends on the option you select from the drop-down list:

- **Rollback top transaction and stop** — The transaction which encloses all the changes done by the database component is rolled back and execution of the mapping stops.
- **Rollback top transaction and continue** — Same as above, but the mapping continues to run after rollback (for example, in order to process a second target component, if one exists).

Some mappings may contain multiple database components. These, in their turn, may all use the same database connection, or connections to completely different databases. The outcome of such mappings in case of database error depends on the execution engine:

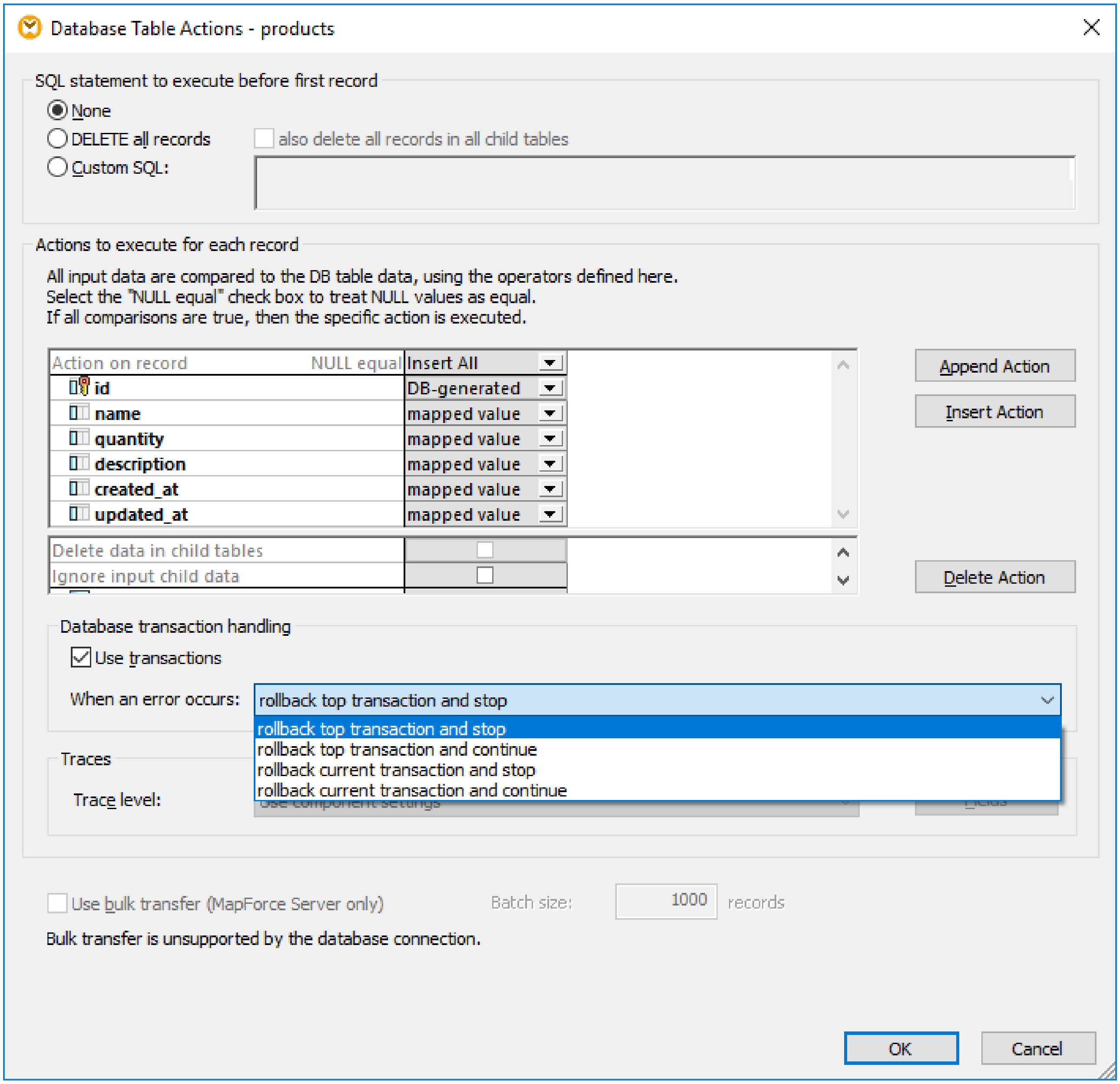
- If the mapping is run with MapForce, only one target component can be executed when the mapping runs. This is the component where the **Preview** button is enabled. If a database error occurs in that component, and if the **Use transactions** check box is enabled, all the changes done by the component will be rolled back.

- If the mapping is run with MapForce Server or a MapForce-generated program, all the target components are executed, sequentially. In this case, when a database error occurs, the rollback will take place for the database component where the error occurred. The mapping will stop or continue executing the next target component depending on the value you selected from the **When an error occurs** drop-down list, as mentioned above.

For reference to all the settings available on the Database Component Settings dialog box, see Database Component Settings 279.

Transactions at table action level

Transactions at table action level enclose each individual table action (insert, update, and so on) inside a transaction. With this setup, if some insert or update action fails on a record, it is possible to roll back only the current failed action and continue processing other records. To enable transactions at table action level, select the **Use transactions** check box from the Database Table Actions dialog box (see also Database Table Actions Settings 326).



The available options are as follows:

- **Rollback top transaction and stop** — If the mapping is going to update multiple tables having a parent-child relationship, it is possible to enable transactions not only for the parent table, but also for the child table. Since transactions can be enabled at multiple levels, "top" refers to the transaction at the highest level in the hierarchy. This is the transaction from the database component level, if you enabled transactions at that level. Otherwise, it is the transaction defined in the topmost table action, assuming that the component contains multiple nested table actions. To understand this option, recall that it is possible for a component to update multiple tables hierarchically, as described in Inserting Data into Multiple Linked Tables 286. For example, a component may have a parent "Insert" action that inserts person records and a nested "Insert" action that inserts one or more addresses for each person record into a separate table. Since you can enable transactions for both insert actions, "top" refers to the highest level in the hierarchy (in this example, the people insert action). Therefore, when an exception is encountered while inserting an address, this rolls back the top transaction (which is at person table level, in this example) and execution stops. In other words, no person record will be inserted if it wasn't possible to insert the person's address in first place.

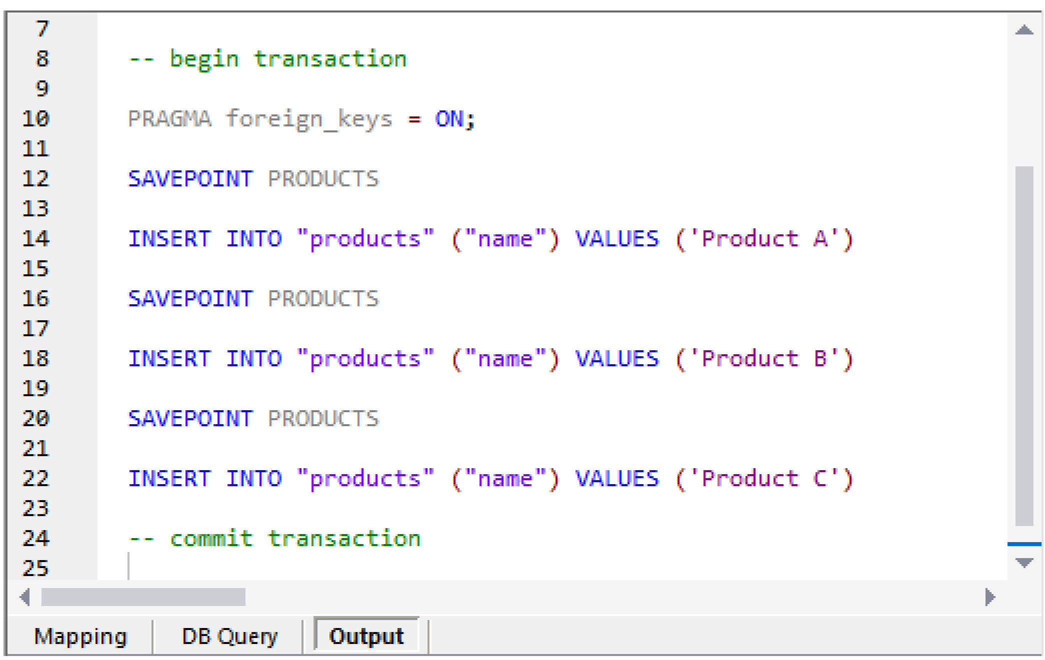
- **Rollback top transaction and continue** — Same as above, but the mapping continues to run after rollback (for example, to process another target component, if one exists).
- **Rollback current transaction and stop** — When an exception is encountered, this rolls back only the changes enclosed in the current transaction; changes done previously outside of the current transaction will be committed.
- **Rollback current transaction and continue** — Same as above, but the mapping continues to run after rollback.

Stored procedure-level transactions

You can also enable database transactions for stored procedures that are called as data target. For an example, see Stored Procedures in Target Components[402]. Transactions are not supported for stored procedures that are called as a data source (without input parameters), or those that are called like a function (with both input and output).

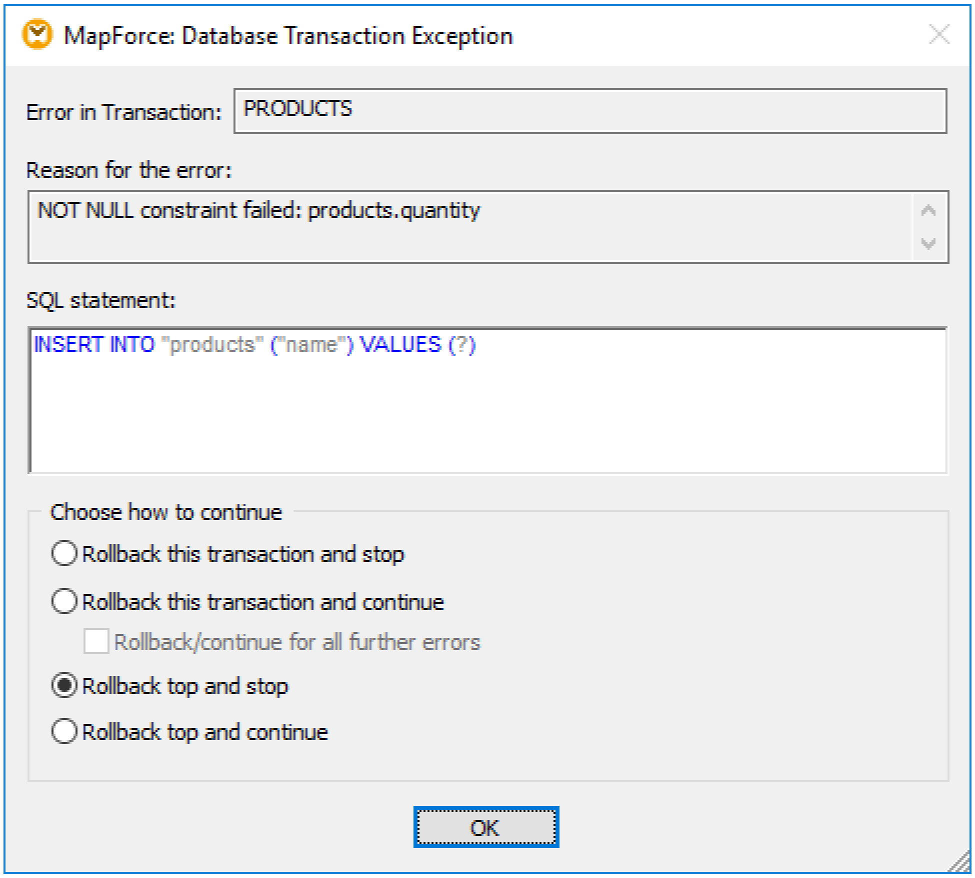
Rollback

To see if transactions are going to occur at mapping runtime, click the **Output** tab and observe the pseudo-SQL code generated by MapForce. The image below illustrates the preview of a mapping where transactions are enabled at component level and table action level. As shown below, a main transaction encloses the entire operation (identified by the comments `--begin transaction` and `--commit transaction`). Also, multiple smaller transactions occur for each insert operation (these are identified by the `SAVEPOINT` command).



When the mapping runs in MapForce Server, or from a MapForce-generated program, the rollback will take place based on the options you have configured while designing the mapping, as described above. For example, if you selected the option **Rollback top transaction and stop**, a failed insert operation will cause all the changes to be rolled back, even if some other inserts may have been successful. However, if you selected **Rollback current transaction and continue**, only the failed insert will be rolled back to the nearest savepoint, and the mapping will continue to attempt inserting other records.

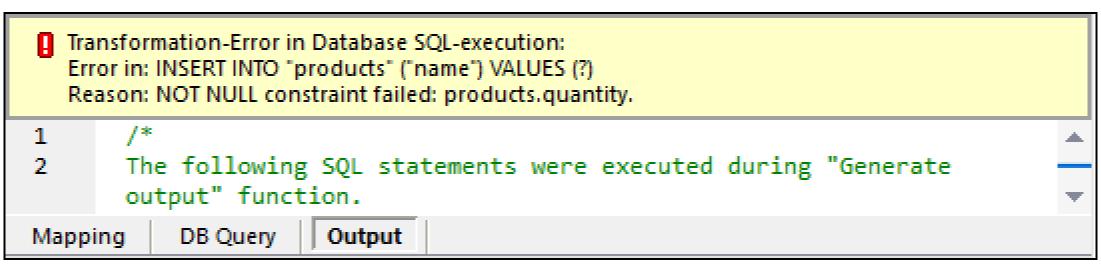
If you run a transaction-enabled mapping directly in MapForce with the menu command **Output | Run SQL-Script**, a dialog box informs you when the database exception is encountered, for example:



You can then change the subsequent behavior as follows:

Rollback this transaction and stop

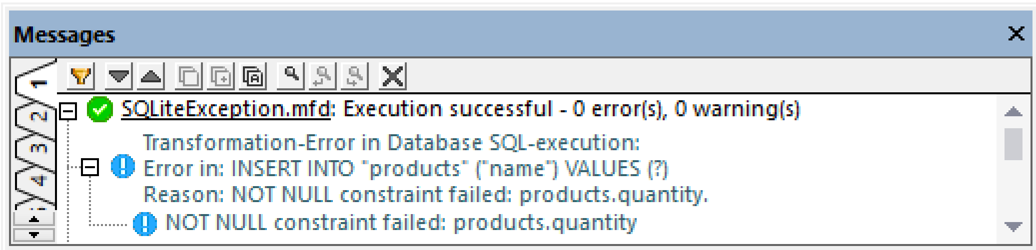
- The current transaction is already rolled back at the time the dialog comes up
- An error is shown at the top of the preview window and in the Messages window



- Any parent transaction is committed
- Mapping execution stops with an error code

Rollback this transaction and continue

- The current transaction is already rolled back at the time the dialog comes up
- The error is treated as a warning, and will be shown in the Messages window when mapping execution finishes.



- Any parent transaction is not affected
- Mapping execution continues with the next record (if this transaction is not on component level) or with the next component (MapForce Server only)

Rollback top and stop

- The current transaction is already rolled back at the time the dialog comes up
- An error is shown at the top of the preview window and in the Messages window
- Any parent transaction is rolled back
- Mapping execution stops with an error code

Note: The **Rollback top and stop** option is available on the dialog box only when there are nested transactions (savepoints).

Rollback top and continue

- The current transaction is already rolled back at the time the dialog comes up
- The error is treated as a warning, and will be shown in the Messages window when mapping execution finishes.
- Any parent transaction is rolled back
- Mapping execution continues with the next record (if this transaction is not on component level) or with the next component (MapForce Server only)

Note: The **Rollback top and continue** option is available on the dialog box only when there are nested transactions (savepoints).

4.2.10.1 Example: Transaction Rollback

This example illustrates possible ways to handle database exceptions when running a mapping that modifies a database. The database used in this example stores user records (`users` table) and addresses (`addresses` table). Each user can have zero, one, or two addresses (for example, a home address and a work address). Namely, each address in the addresses table contains a `user_id` field which points to the `id` field in the `users` table. The `users` table is therefore a "parent" table for `addresses`. The listing below illustrates the SQL creation scripts for both tables (note the syntax is applicable to a SQLite database):


```
CREATE TABLE
       users (id INTEGER NOT NULL PRIMARY KEY,
       first_name TEXT NOT NULL,
       last_name TEXT NOT NULL,
       email TEXT UNIQUE NOT NULL);

CREATE TABLE
       addresses (id INTEGER NOT NULL PRIMARY KEY,
       user_id INTEGER NOT NULL,
       is_shipping INTEGER,
       is_billing INTEGER,
       type TEXT NOT NULL,
       city TEXT NOT NULL,
       street TEXT NOT NULL,
       number INTEGER NOT NULL,
       FOREIGN KEY (user_id) REFERENCES users (id) ) ;
```

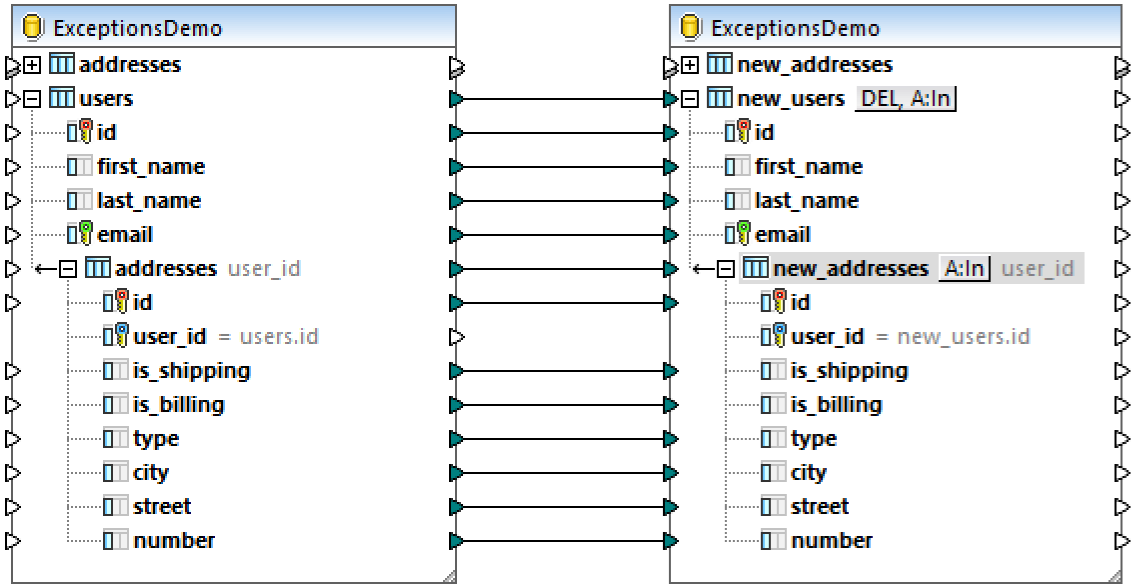
Source tables

The business requirement is to copy all data from the `users` and `addresses` table to some new tables, namely, the `new_users` and `new_addresses` tables. These are almost identical to the `users` and `addresses` tables. The only difference is that the table `new_addresses` does not allow null values for the columns `is_shipping` and `is_billing`, as highlighted in the code listing below. This is important because it means that exceptions will occur if null values are encountered.

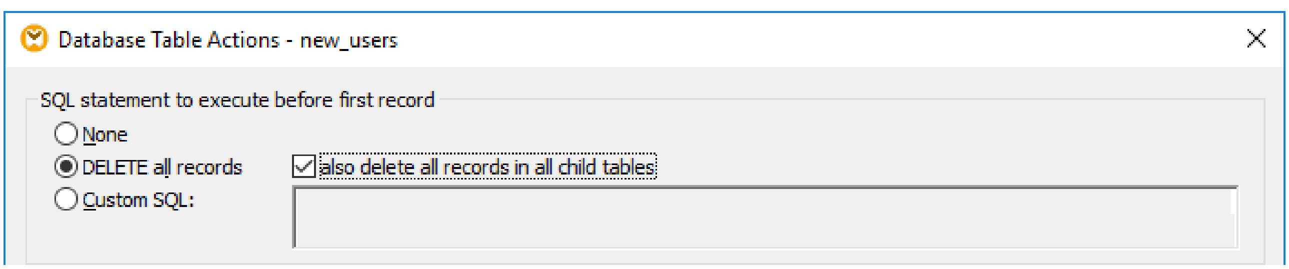
```
CREATE TABLE
       new_users (id INTEGER NOT NULL PRIMARY KEY,
       first_name TEXT NOT NULL,
       last_name TEXT NOT NULL,
       email TEXT UNIQUE NOT NULL);

CREATE TABLE
       new_addresses (id INTEGER NOT NULL PRIMARY KEY,
       user_id INTEGER NOT NULL,
       is_shipping INTEGER NOT NULL,
       is_billing INTEGER NOT NULL,
       type TEXT NOT NULL,
       city TEXT NOT NULL,
       street TEXT NOT NULL,
       number INTEGER NOT NULL,
       FOREIGN KEY (user_id) REFERENCES new_users (id) ) ;
```

Target tables

The mapping that copies all data from the old to new tables is illustrated below. You can find this mapping at the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\DatabaseExceptions.mfd.

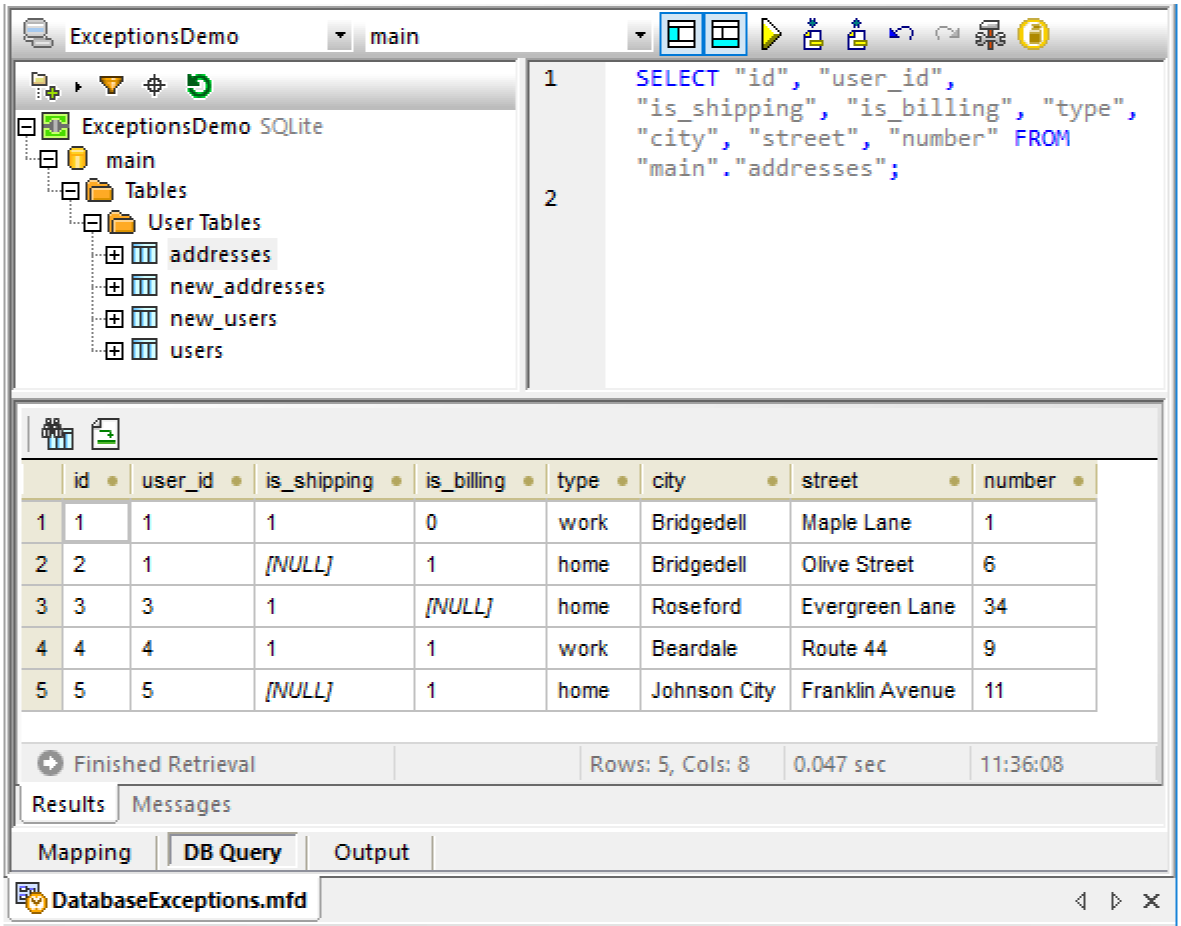


As illustrated above, this mapping is configured to copy data verbatim from a source database component to a target database component. No other transformations take place, for the sake of simplicity. In this example, both the source and target component belong to the same database, only the source and target tables are different. Click the DEL, A:In button and notice that all the records are set to be deleted before each insert action. This ensures that the target tables are always empty before each insert.



As stated before, exceptions will occur if the source `addresses` table contains null values. You can check that the source table contains null values as follows:

1. Click the **DB Query** tab.
2. Right-click the `addresses` table, and select Show in **SQL Editor | SELECT**.
3. Click **Execute Query** .



As illustrated above, there are various `NULL` fields in the source `addresses` table. Considering this, when running this mapping, you have various options to deal with exceptions, for example:

- A) If any exception is encountered, roll back all changes. In other words, do not insert any records if the source data contains invalid records.
- B) If any exception is encountered, skip the records where the exception occurs but keep inserting records that are valid.

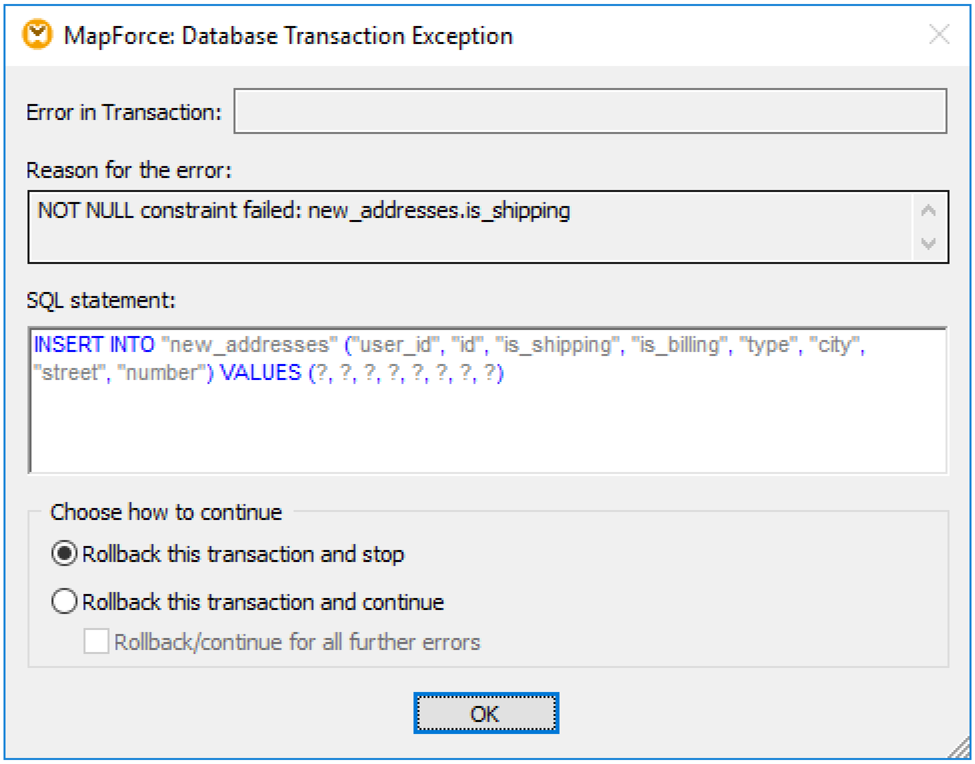
Scenario A: Rollback all changes on exception

Business requirement: I want to roll back all changes to the database if an exception occurs. To configure the mapping to accomplish this:

1. Right-click the title bar of the target database component and select **Properties** from the context menu. Alternatively, double-click the title bar. This opens the Database Component Settings dialog box.
2. Select the **Use Transactions** check box, and **choose rollback top transaction and stop**.

When the mapping is configured as shown above, any encountered exception will cause the top-level transaction at database component level to be rolled back. To run the mapping with these settings:

1. Click the **Output** tab.
2. On the Output menu, click **Run SQL-Script**. At this stage, the mapping encounters an exception because of a null value in the source addresses table. The Database Transaction Exception dialog box appears.



3. Leave the default preselected option unchanged and click **OK**.

The result is as follows:

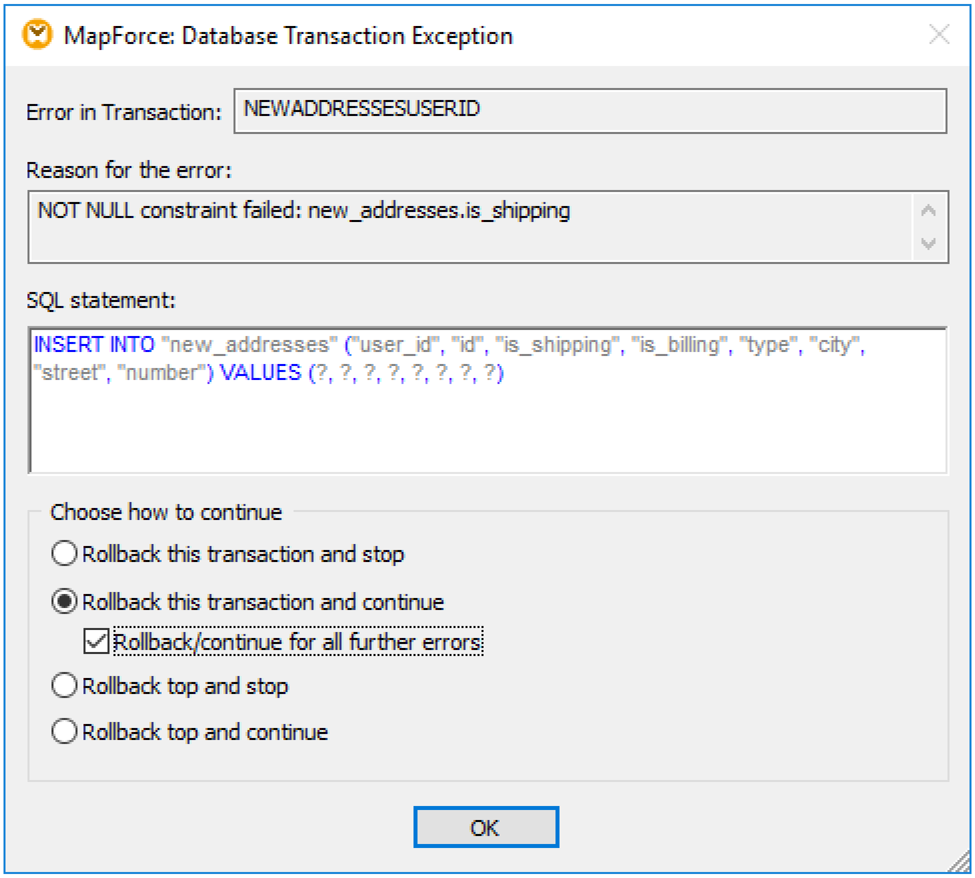
- All the changes are rolled back.
- No records are inserted in the `new_users` table
- No records are inserted in the `new_addresses` table

Scenario B) Rollback current transaction and continue execution

Business requirement: I want to skip records which generate an exception but keep inserting valid records. To configure the mapping to accomplish this:

1. Click the DEL, A:In button next to the `new_users` table, select the **Use transactions** check box and choose **rollback current transaction and continue**
2. Click the A:In button next to the `new_addresses` table, select the **Use transactions** check box and choose **rollback current transaction and continue**

When you run the mapping with these settings, the Database Transaction Exception dialog box is displayed. Select the options as shown below to roll back only the current transaction and keep running the mapping:



The result is as follows:

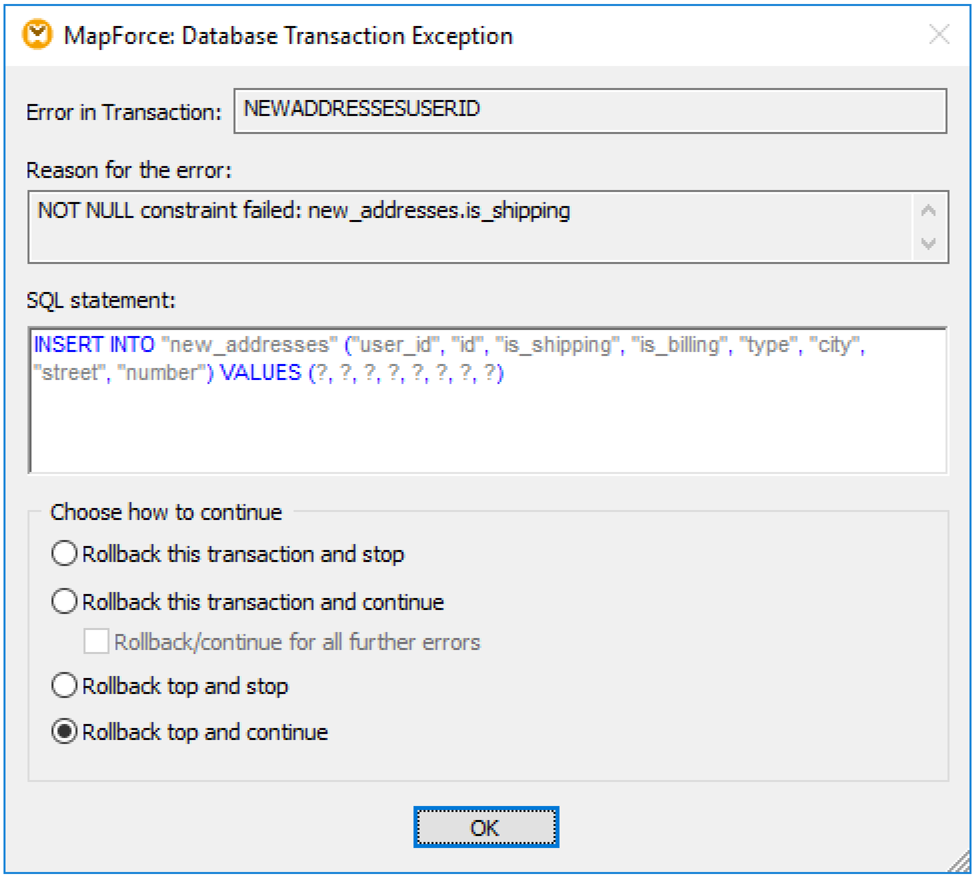
- All erroneous transactions are rolled back.
- 5 users out of 5 are inserted (because no user record generated any database exception)
- 2 addresses out of 5 are inserted (because 3 addresses contain null values and generated exceptions)

Scenario C) Rollback top transaction and continue execution

Business requirement: If an address fails to be inserted, then the parent user record should also not be inserted. To configure the mapping to accomplish this:

1. Double-click the title bar of the database component, and clear the **Use transactions** check box.
2. Click the DEL, A:In button next to the `new_users` table, select the **Use transactions** check box and choose **rollback current transaction and continue**
3. Click the A:In button next to the `new_addresses` table, select the **Use transactions** check box and choose **rollback top transaction and continue**

When you run the mapping with these settings, the Database Transaction Exception dialog box. Select the options as shown below, and repeat the choice as many times as prompted:



The result is as follows:

- 2 users are inserted (Sharda Junker, Tobie Hughey)
- 1 address is inserted

Explanation: The user "Sharda Junker" is the only user who does not have an address. No exceptions occurred for this record, so it was inserted. The second user, "Tobie Hughey", was inserted because it is the only user where no exceptions occurred at address level. All other user records were not inserted because they had at least one address where an exception occurred and the transactions were rolled back.

There are two addresses which don't have nulls and thus don't throw exceptions. These are addresses with `id=1` and `id=4`. The insert transaction for the first address, however, was rolled back because the parent user transaction had to be rolled back. Therefore, only the address with `id=4` was inserted.

Note that the same result can also be achieved as follows:

1. Click the DEL, A:In button next to the `new_users` table, select the **Use transactions** check box and choose **rollback current transaction and continue**
2. Click the A:In button next to the `new_addresses` table, and clear the **Use transactions** check box.

Running the mapping with MapForce Server

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine, as follows:

1. Compile the mapping to a MapForce Server execution file (.mfx) with the menu command **File | Compile to MapForce Server Execution File**, see also Compiling Mappings to MapForce Server Execution Files 1378.
2. Copy the .mfx file and the **ExceptionsDemo.sqlite** database to a directory on the server machine. Let's call it the "working directory".
3. Change the command prompt to the working directory and run MapForce Server with the command below:

```
mapforceserver run DatabaseExceptions.mfx
```

Notes:

- **mapforceserver** is the path to the MapForce Server executable as applicable for your operating system. Change the path to the .mfx file as applicable. For example, on a Linux machine, the path is: `/opt/Altova/MapForceServer2023/mapforceserver`.

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see Automation with MapForce Server 1371.

4.2.11 Database Tracing and Error Logging

When a mapping writes data to a database, you can optionally enable database tracing and error logging. Tracing is useful if you want to track all the changes done by the mapping to the database when the mapping runs. When tracing is on, events such as database insert or update actions or errors are logged in an XML file that you can later analyze or perhaps process further in an automated way. If you prefer the log file to be in a format other than XML, you can optionally map data from it to some other component kind supported by MapForce (for example, a text file, or even another database).

Note: In order for a mapping to support tracing and error logging, the mapping language must be set to **Built-in**. Database logging and tracing is not supported in C#, C++, or Java.

Database tracing is configurable, in that you can choose to trace all messages, only errors, or you can disable tracing completely. In addition, you can use tracing at the following levels:

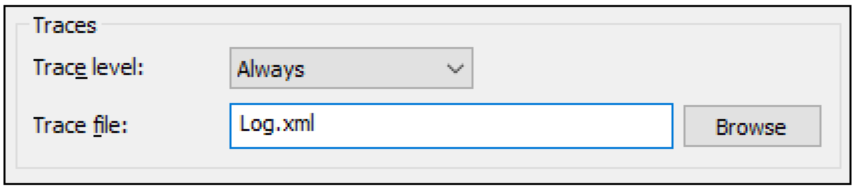
- Database component level. At this level, you effectively switch tracing on or off for that database component. This is useful for mappings that have multiple target database components, and you need tracing only for some of them.
- Table or stored procedure level. On a database component, you can decide whether tracing should take place for each table or stored procedure. When enabled, tracing at this level includes events pertaining to table actions such as "Insert All", "Update If", "Ignore If", "Delete If". In case of stored procedures, events related to the stored procedure call are traced.
- Database field level. By default, all fields are traced, but you can exclude certain fields from being traced, or you can specify that they should be included in the trace file only on error.

Importantly, the three levels above are hierarchical and tracing settings are propagated from top to bottom. Therefore, if you disable tracing at database component level, it is not possible to enable it at table or stored procedure level. Likewise, if you disable tracing at table level, it is not possible to set it at database field (column) level. The same applies when you narrow down the tracing level. For example, if you limit tracing only to errors at database component level, then it is not possible to use full tracing at table or stored procedure level.

Tracing at database component level

To enable tracing at database component level:

1. Right-click the database component, and select **Properties** from the context menu (Alternatively, double-click the title bar of the database component). This opens the Database Component Settings dialog box.



2. Select the desired trace level (**Always**, **Errors**, **Disabled**). Selecting **Errors** will create a trace file only if database errors occur when the mapping runs.
3. Next to **Trace file**, type the name or path of the XML file where all the tracing details will be written (for example, **Log.xml**). If you enter a file name instead of a full path, the trace file will be created in the same directory where the mapping runs.

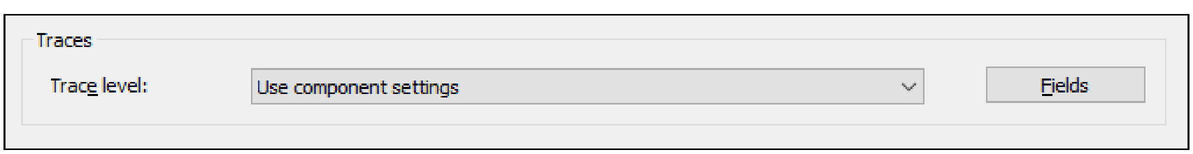
After you enable tracing, the database component changes its structure on the mapping. Namely, it displays, in addition to the database structure, a new node hierarchy of items where the top item in the hierarchy is the trace file. For more information, see Trace File Structure 429.

Enabling tracing at database component level automatically enables it for all tables or stored procedures on that component to which a mapping connection exists. If necessary, you can further tweak tracing at the table or stored procedure level, as described below. Note that tables or stored procedures that are displayed on the component but do not take part in the mapping (that is, no mapping connections exist to them) will not be traced.

Tracing at table or stored procedure level

To enable tracing at table or stored procedure level:

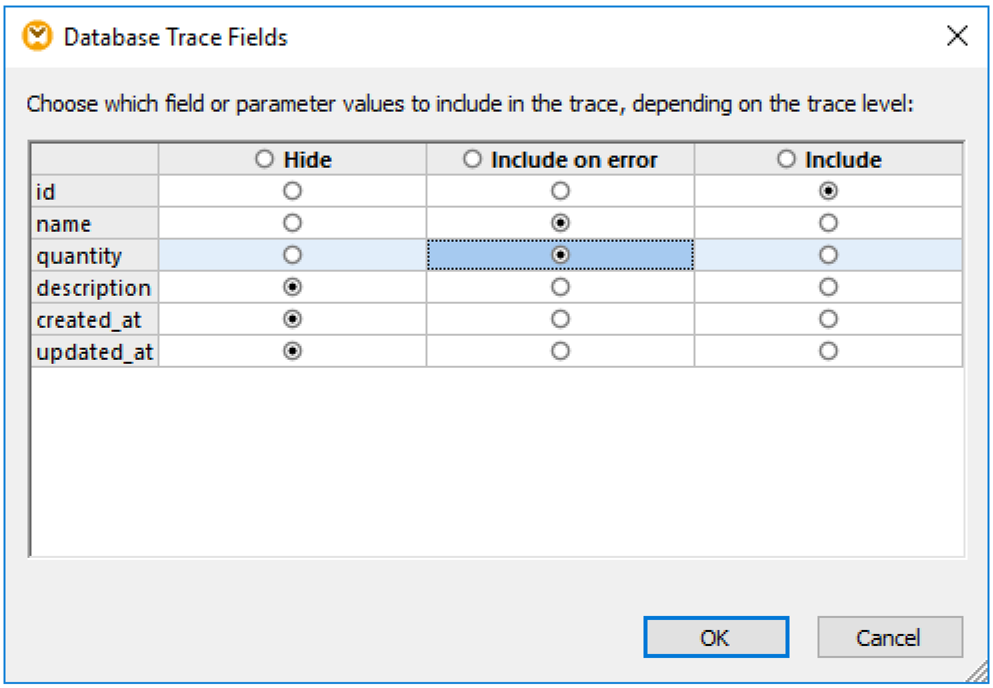
1. Make sure that the tracing level at database component level is set to either **Always** or **Error** (see above).
2. Do one of the following:
   - o Click the table action button (for example, A:In) next to the table where you want to enable tracing, or
   - o If this is a stored procedure, click the **Show Context Menu** button, and then select **Procedure Settings** from the context menu.



3. Select the trace level. The **Use component settings** option inherits the same settings that were defined at the component level. The **Limit to errors** option restricts tracing only to error events. **Always disabled** means that no tracing will occur for this table or stored procedure.

Tracing at database field level

When you enable tracing at database component level and table or stored procedure level, all the fields (database columns) are included in the tracing report by default. Optionally, you can restrict tracing only to specific database fields or stored procedure parameters. To do this, click the **Fields** button on the Database Table Actions or Database Procedure Settings dialog box, see above.

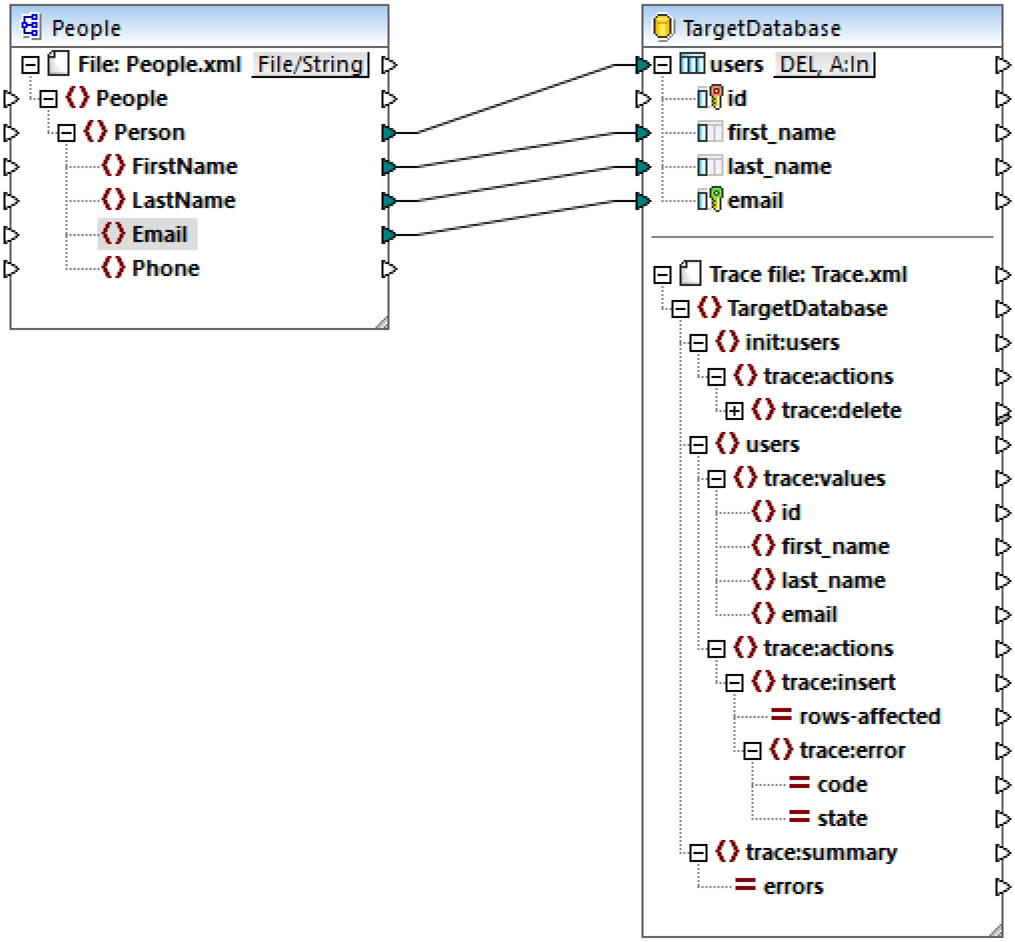


Selections made on this dialog box affect the trace file structure. For example, any fields that you opted to hide will no longer appear in the trace file. You might want to do this, for example, in order to reduce the log file size.

To hide or include multiple fields simultaneously, click the respective option in the header.

4.2.11.1 Trace File Structure

When tracing is enabled for a database component, a tracing structure becomes available in the lower half of the component, for example:



As illustrated above, the top node in the tracing structure indicates the name of the trace file that will be written when the mapping runs. The rest of the tracing structure is modeled after the structure of the database tables or stored procedures that take part in the mapping. In this example, the database name is "TargetDatabase", so a similar item exists in the trace structure.

The child item of "TargetDatabase" is **users**, which corresponds to the "users" database table. In this example, the database component has only one table, "users". However, in a component with many tables, the tracing structure includes each table present on the component. If you add or remove tables or stored procedures from the database component, then the tracing structure is updated accordingly. For instructions, see Adding, Editing, and Removing Database Objects 252.

In this example, an **init:users** structure also appears. This happens when you have configured the mapping to perform some initial action before the main insert, update, or some other action. For example, if you opted to delete all existing records from the table before inserting new ones, the **init** structure is present. It collects tracing information about the initial action (such as delete, or a custom SQL query) and any errors that may occur at this stage.

The **trace:values** structure displays all the columns of the database table (in this case, the columns of the "users" table). For stored procedures, this structure displays the parameters of the stored procedure. By default, all columns are set to be traced, but you can configure which columns should be traced, as described previously.

The **trace:actions** items indicate which actions are currently set to take place on the database table. In this example, a database update action A:Up is set to take place, so the **trace:update** item is available. If you change the component to perform a database insert action A:In, then the trace structure would change to **trace:insert**, and so on for other action kinds. In case of stored procedures, a **trace:execute** action is visible instead.

Each trace action has a **rows-affected** attribute that specifies how many rows were affected by the respective database action.

The **trace:error** item is populated only if a database error occurs during mapping execution. It has two attributes, **code** and **state**. The text of the error and the attribute values are supplied by the database driver and will therefore be different for each database kind.

The **trace:summary** item includes an **errors** attribute which reports the number of encountered errors. The attribute value is **0** if no errors were encountered.

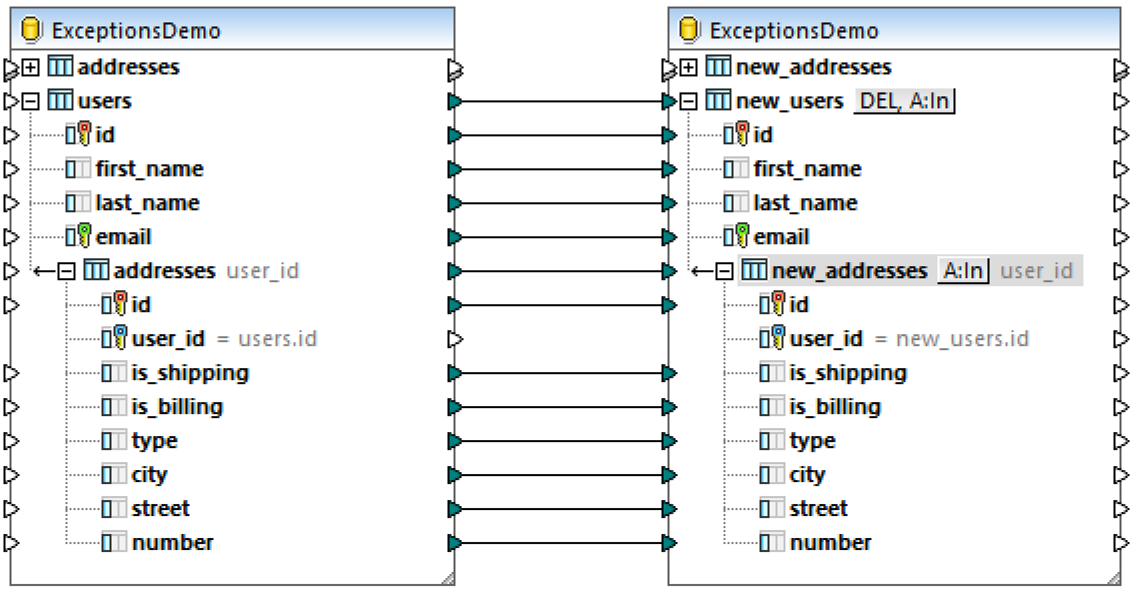
Previewing trace files

When tracing is enabled for a target database component, the trace file is also included in the preview when you click the **Output** tab to preview the mapping result. This is useful if you want to see how the file will approximately look like. Note that the actual generated trace file will look different based on the mapping execution result.

4.2.11.2 Example: Logging Database Errors

This example shows you how to configure database tracing and error logging for a mapping which inserts data into a SQLite database. The instructions are similar for all other database types supported by MapForce, as well as for other database action types (such as "update", "ignore", "delete", and so on).

You can find the demo mapping referenced by this example at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\DatabaseExceptions.mfd**. This mapping inserts data from two tables of a database into a new pair of tables of the same database, and is described in more detail in the Example: Transaction Rollback [420]. Note that some of the records in the database are deliberately null so as to generate database exceptions.



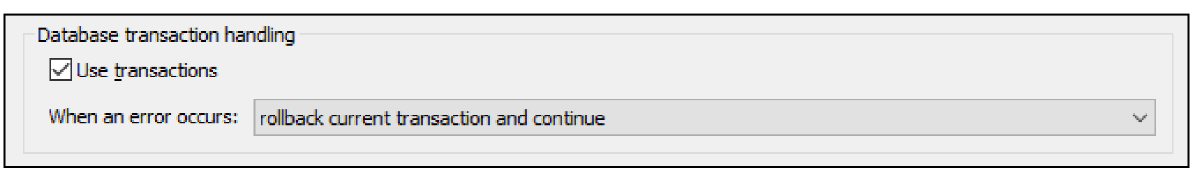
DatabaseExceptions.mfd

In this example, the business requirement is to enable tracing for the target database component in order to determine how many errors were generated, and where exactly did the errors occur. The steps below illustrate how to achieve this goal.

Step 1: Enable transaction rollback

This demo mapping is configured to use transaction rollback at database component level. This means that, when the first exception is encountered, the mapping will stop execution and roll back all data to the initial state. Let's first change the transaction rollback options so that the mapping execution continues even if exceptions are encountered for some of the records:

1. Click the DEL, A:In button next to the `new_users` table, select the **Use transactions** check box and choose **rollback current transaction and continue**.



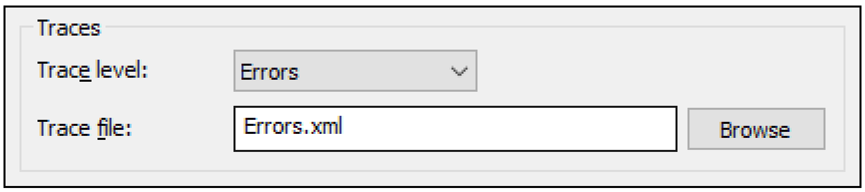
2. Click the A:In button next to the `new_addresses` table, select the **Use transactions** check box and choose **rollback current transaction and continue**.

You could also disable transaction rollback completely; however, doing this would stop mapping execution at the first encountered error and would *not* roll back changes, so this is not recommended. Tracing and logging in this case would reflect execution only up to the first encountered exception. For further information about transaction rollback, see Handling Database Exceptions 415.

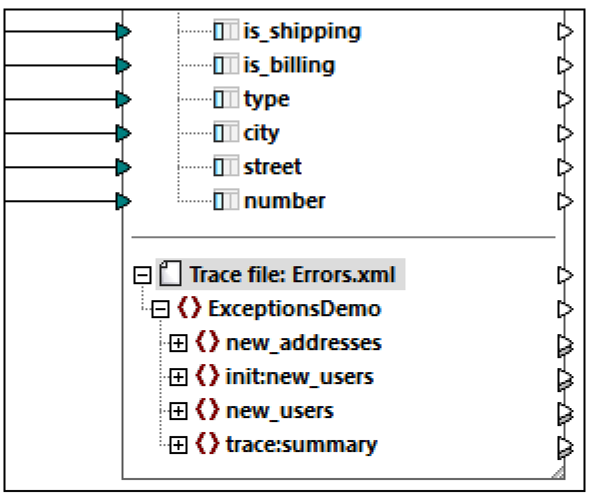
Step 2: Configure error logging

To enable tracing, do the following:

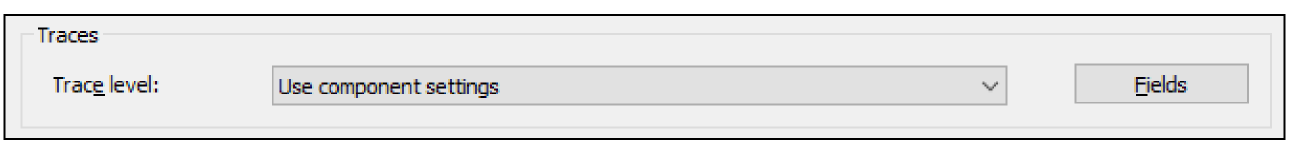
1. Right-click the title bar of the database component, and select **Properties** from the context menu (Alternatively, double-click the title bar of the database component). This opens the Database Component Settings dialog box.
2. Select **Errors** as trace level.
3. Next to **Trace file**, type the name of the tracing file (in this example, **Errors.xml**). Do not enter an absolute path if you intend to run the mapping with MapForce Server on other machines or operating systems.



The target database component has now changed its structure and displays an additional compartment with the trace file structure:



So far, you have enabled tracing at the database component level. Consequently, the tracing settings will be inherited by all table actions on this database component. In this example, they are inherited by the "Insert" actions of both `new_users` and `new_addresses` tables. If you click the action button A:In for each of these tables, the settings illustrate this fact:



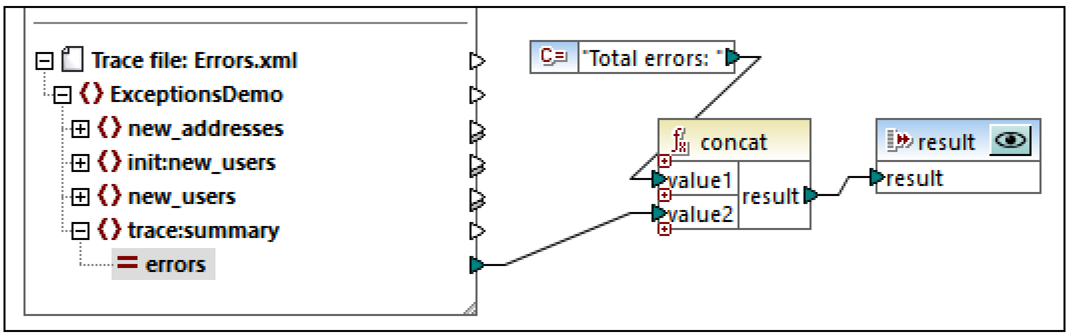
Optionally, click the **Fields** button to configure which fields should be included or hidden in the trace file. In this example, all fields are included.

Step 3: Configure the mapping output

This step shows you how to map the values from the trace file to some other component kind (in this case, a simple output component). Note that this step is optional; an error log file is produced even if you do not map the trace file to some other component. In this example, however, we would like to explicitly configure the mapping to return the total number of encountered errors in the output. To do this, let's take the following steps:

1. Add a simple output component, by selecting the **Insert | Insert Output** menu command. For more information about simple input components, see Returning String Values from a Mapping [843].
2. Add a constant, by selecting the **Insert | Constant** menu command. The constant will provide the static text "Total errors: " to make the mapping output text more clear.
3. Drag the **concat** function from the Libraries window onto the mapping. For more information about built-in functions, see Add a Function to the Mapping [905].

As illustrated below, the **concat** function concatenates the text of the constant with the **errors** value supplied by the trace file.



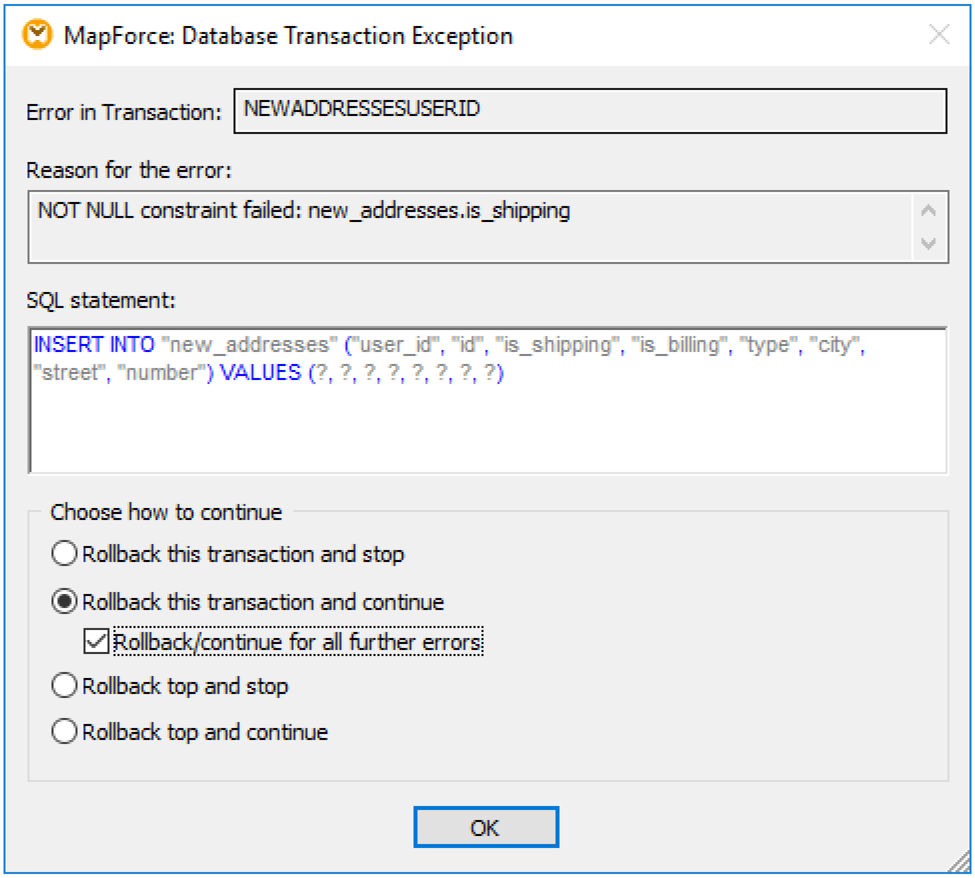
With the configuration above, when the mapping is run with MapForce Server, the output will display the text: "Total errors: {value}", where **value** represents the number of errors encountered at runtime.

> At this stage, running the mapping in MapForce with the **Preview** button pressed on the "result" component is not possible and will result in an error. A trace file must exist first, because the simple output reads data from the trace file. The next step shows you how to produce the trace file.

Step 4: Running the mapping

To run this mapping with MapForce:

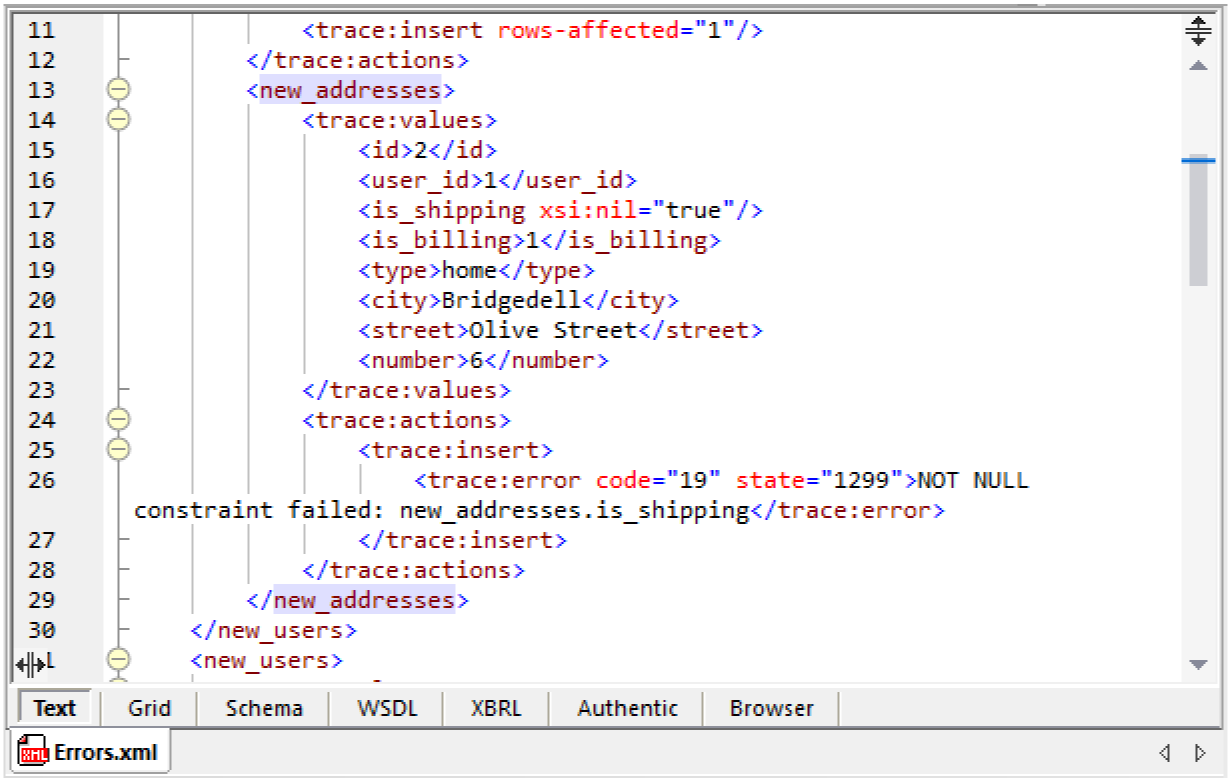
1. Click the **Preview** button on the database component.
2. Click the **Output** tab.
3. On the **Output** menu, click **Run SQL-Script**. At this stage, the mapping encounters the first exception; this is intentional. The Database Transaction Exception dialog box appears.



4. Select the rollback options as illustrated above, and click **OK**.

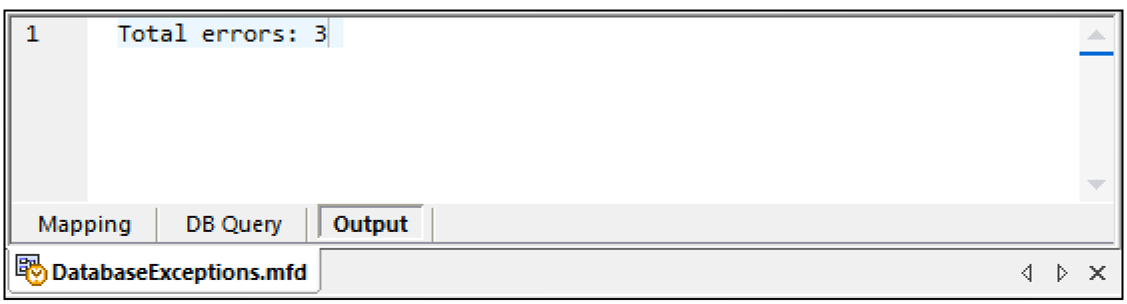
Note: If a mapping is executed by MapForce Server, no dialog box appears and the erroneous transactions are rolled back automatically, according to the rollback settings configured in MapForce.

In this example, once the mapping completes execution, all failed transactions are rolled back, and only the successful ones are committed to the database. The encountered errors are written to the **Errors.xml** file, a small fragment of which is illustrated below:



Now that the log file was produced, you can preview the output that returns the total number of errors in MapForce execution, as follows:

1. Back on the mapping, click the **Preview** button on the "result" component.
2. Click the **Output** tab and observe the result.



> This part of the mapping reads data from the log file produced previously. If the log file does not exist, an error will be generated.

4.2.12 NoSQL Databases

MapForce Enterprise Edition and MapForce Server Advanced Edition support NoSQL databases that can be used as mapping sources and targets. Currently, only two document databases are supported: MongoDB and CouchDB. To find out more about the peculiarities of NoSQL databases, see the following articles: What is NoSQL and NoSQL vs. SQL Databases. NoSQL databases do not support relationships [257] and custom SQL statements.

This section discusses the following topics:

- About NoSQL Databases [437];
- NoSQL Database Configuration [438].

4.2.12.1 About NoSQL Databases

This subsection discusses the main points of NoSQL databases.
Primary key
The primary key is always **`_id`**.

JSON Schema
MapForce uses JSON Schema to describe the structure of NoSQL database components. Note that in MapForce a data structure is an essential prerequisite for data mapping. Instead of fields in a database table, the user must embed a JSON schema to get a mappable structure. The schema can be user-created or auto-generated from the database content. See details in Assign JSON Schema [440]. MongoDB supports storing a schema in the database to validate documents.

Data in JSON and BSON
MongoDB stores documents in BSON (Binary JSON). CouchDB stores data in JSON. See JSON and BSON to find out more about their differences.

BSON function library
A new bson function library [1072] is available in the Libraries window [33], which allows creating and manipulating BSON types. To find the `bson` libarary, type `bson` in the text box located at the bottom of the **Libraries** window.

Component Settings
In the **Component Setting** dialog box, only the following sections of the dialog are available for NoSQL databases: **Database**, **Login settings**, **Timeout for statement execution**, and **Database transaction handling**. The other settings are relevant only to SQL databases. To find out more about the component settings, see this topic [279].

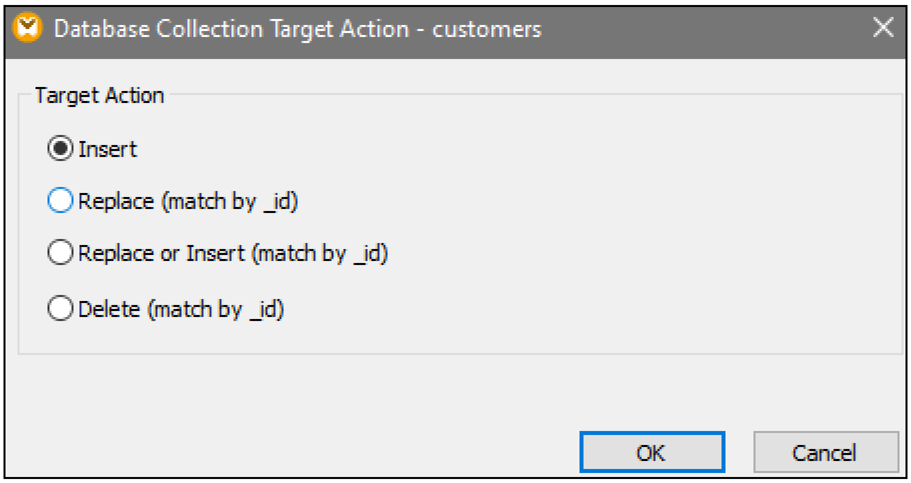
DB Query
Currently, there is limited support in the DB Query pane [38]. The **DB Query** pane allows you to see the list of available collections in your database, but query execution is not supported.

SQL/NoSQL Where/Order Component
The SQL/NoSQL-WHERE/ORDER [350] component: JSON-based syntax is used to filter and sort database data; the **`"@"`** sign is used instead of **`":"`** before a parameter in the query filter. See the sample in the dialog box

SQL/NoSQL-WHERE/ORDER Parameters. To find out more about syntax for querying documents in a collection, see the MongoDB documentation.

Join component
The Join component 333 works in non-SQL mode only.

Dialog box Database Collection Target Action
NoSQL databases support four actions for target collections. When a NoSQL database is used as a target component, the In button (the **Action** button) appears next to the collection name. The **Action** button changes depending on the chosen action. When you click on this button, the dialog box **Database Collection Target Action** opens and suggests choosing one of the four target actions: **Insert**, **Replace**, **Insert or Replace**, and **Delete** (*see screenshot below*).



The description of the target actions is summarized in the list below:

- The **Insert** action: The mapped data will be inserted into the target database in addition to the data (if any) that is already in this target database.
- The **Replace** action replaces the whole document in the target database if its `_id` matches the one in the mapped data.
- The **Replace or Insert** action replaces the whole document in the target database if its `_id` matches the one in the mapped data. Otherwise, the **Insert** action is performed.
- The **Delete** action deletes the whole document in the target database if its `_id` matches the one in the mapped data.

4.2.12.2 NoSQL Database Configuration

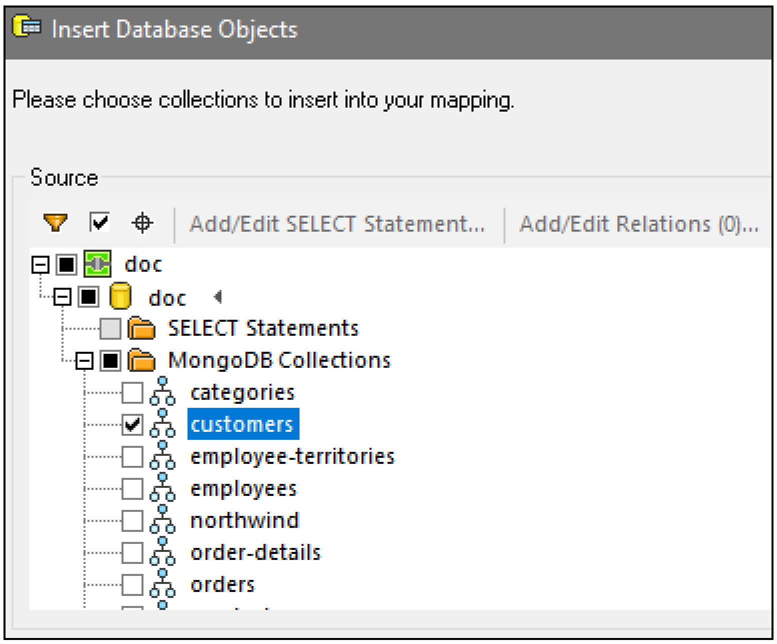
The instructions below will help you add a NoSQL database as a mapping component and assign a schema to it.

Add a NoSQL database

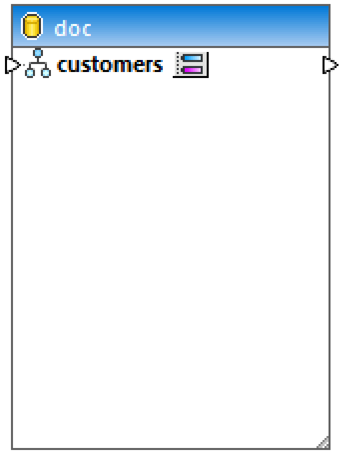
Before adding a database to the mapping, make sure to select a transformation language where database mappings are supported. For NoSQL databases, MapForce supports only Built-In 27 as a transformation language. The instructions below show how to add collections from a MongoDB database to a mapping. Our sample database is called **doc**.

To add a collection from the **doc** database to the mapping, follow the instructions below:

1. In the first step, we need to select a source database (MongoDB in our case). Go to the **Insert** menu and click **Database**. Alternatively, click the toolbar button (**Insert Database**).
2. Select **MongoDB** in the **Connection Wizard** and click **Next**.
3. The **Select a Database** dialog box asks you to provide the following parameters: **Host**, **Port**, **Database**, **Username**, and **Password**. Enter the required parameters and click **Connect**. To find out more about how to connect to a database source, see MongoDB Connection 192 and CouchDB Connection 193.
4. The dialog box **Insert Database Objects** will open and ask you to select collections you would like to use in your mapping (*see screenshot below*). Check the relevant boxes and click **OK**.



5. Now your NoSQL database component contains a collection called `customers` (*see screenshot below*).



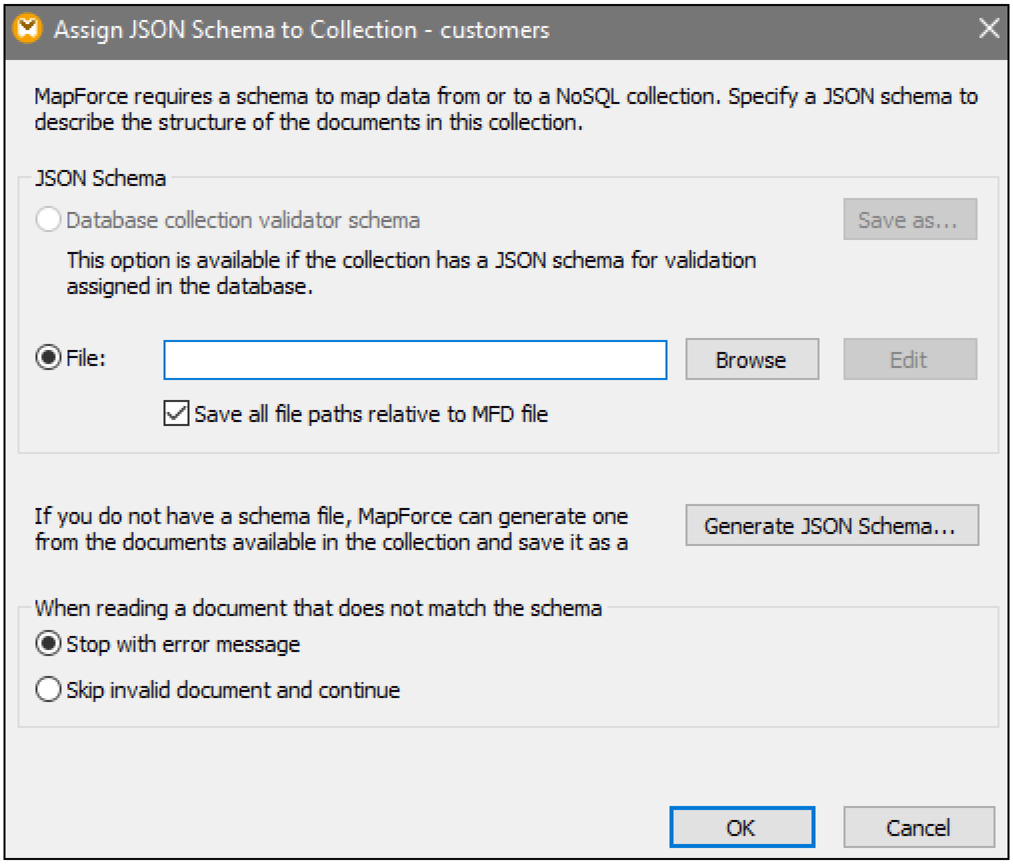
The next step will be to assign a JSON schema to define the structure of your collection.

Note: When you connect to a MongoDB cluster, you may need to contact your IT department to get firewall access to the IP addresses of your cluster and port.

Assign JSON schema

Now that we have selected the relevant documents from our **doc** collection, we need to assign a JSON schema to the NoSQL component. Note that assigning a JSON schema is a prerequisite for MapForce: otherwise, you will not be able to carry out a mapping. To assign a JSON schema to the NoSQL component, follow the instructions below.

1. Click the button (**Assign JSON Schema**) located to the right of the collection name (`customers`). This opens the dialog box **Assign JSON Schema to Collection** (*see screenshot below*).



2. To assign a JSON schema, you can choose one of the three options: (i) choosing a schema assigned in your database; (ii) browsing for the existing JSON schema on your local machine; (iii) having MapForce generate a new schema file.

   i. If you choose the first option, you will also be able to export the JSON schema from the database (click **Save as**).
   ii. If you already have a JSON schema for your source file, click **Browse** and search for the relevant schema. When you choose to upload your schema file, the **Edit** button is enabled. When you click this button, your schema file will be opened in Altova XMLSpy if it is installed on your machine.
   iii. If you do not have a schema, MapForce can generate it for you. In this case, click **Generate JSON Schema**.

   You must also specify whether document processing will stop or continue if there are documents that do not match the schema (*see the last section of the dialog box above*). Note that both of these options apply at mapping runtime.

3. If you click **Generate JSON Schema**, the dialog box **Generate JSON Schema** will open. Leave the **FILTER** and **SORT** text boxes empty if you do not want to specify any filtering and sorting criteria. Then click **OK**. Note that you can also specify the number of documents the parser needs to analyze. The **SKIP** option tells the parser to skip the first *N* documents in the collection. The **LIMIT** option instructs the parser to analyze only the first *N* documents in the collection. The schema of the database component will then be based on the structure of the documents specified by the combination of these criteria. For more information about **FILTER** and **SORT**, see the subsection below.

4. When you click **OK**, you will need to give your schema file a name and select the desired folder in which to save it. After you have done this, click **OK** again.

Now your database component has a tree structure and is ready for mapping.

FILTER and SORT
Without any query options in the **FILTER** and **SORT** text boxes (the **Generate JSON Schema** dialog box), MapForce will read all the documents in the collection to generate a schema. However, collections may contain thousands of documents, which could lead to a considerable processing time. In this case, the best strategy would be to select a subset of documents that cover all structure variants that you want to process in your mapping. In this context, the term *structure variants* specifies which properties exist in the document and which types these properties may have.

It is unlikely that documents with completely different structures will be stored in the same collection, but there may be many additional fields which only occur in some documents. The following scenarios are possible:

- If the filter selects only documents that do not have a field present in some other documents, the schema will not contain that field. If such a field is encountered later at runtime in a document, a validation error will occur or there will be no node to map to in the target component.
- If all sampled documents do contain a specific property, this property will be marked as required in the schema, and a document (which may have been added later to this collection) without this property will be invalid against the schema.

In practice, a schema generated from a limited sample of documents may be incomplete and require manual modifications: e.g., to allow additional properties, to make some properties optional, or to allow additional data types for some properties.

FILTER example
In the subsection Add a NoSQL database 438, we created a NoSQL database component that contains a document called `customers`. Now we would like MapForce to take the filtering criteria into consideration while generating a new schema. In our `customers` collection, there are documents of two types (*see below*).

doc.customers1

```
"CustomerID": "ALFKI",
"CompanyName": "Alfreds Futterkiste",
"ContactName": "Maria Anders",
"ContactTitle": "Sales Representative",
"Address": "Obere Str. 57",
"City": "Berlin",
"Region": "NULL",
"PostalCode": "12209",
"Country": "Germany",
"Phone": "030-0074321",
"Fax": "030-0076545"
```

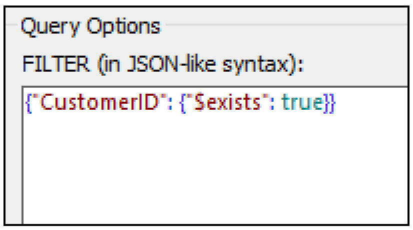
doc.customers2

```
"0": "BOTTM",
"1": "Bottom-Dollar Markets",
"2": "Elizabeth Lincoln",
"3": "Accounting Manager",
"4": "23 Tsawassen Blvd.",
```



```
"5": "Tsawassen",
"6": "BC",
"7": "T2F 8M4",
"8": "Canada",
"9": "(604) 555-4729",
"10": "(604) 555-3745"
```

Now we want our schema file to be based on the structure with the name fields (`doc.customers1`). The screenshot below illustrates our filtering criterion that runs as follows: Select only those documents in which the `CustomerID` field exists (*see screenshot below*). The parser will run through the list of all the documents in the collection and choose only those that satisfy this criterion. To find out more about syntax for querying documents in a collection, see the [MongoDB documentation](#).



After filtering your data, you could also choose to sort it, for example, based on the `City` field. You could then specify how many documents from the sorted sample to skip (**SKIP**) and/or to take into account (**LIMIT**) for your schema generation.

Important:

When you choose to filter and/or sort your data, a subset of your documents like the one in `doc.customers2` will not conform to the generated schema, which will cause validation errors. To avoid these errors, select **Skip invalid documents and continue** in the dialog box **Assign JSON Schema to Collection** (*see screenshot above*).

4.3 EDI

Altova website: EDI mapping

EDI (Electronic Data Interchange) is a family of standards which enable electronic exchange of data between organizations or businesses. You can map data to or from EDI formats in the following transformation languages: Built-In, C#, C++, Java. The EDI data can be mapped to or from any of the formats supported by MapForce, including databases, flat files, XML documents, and others. For the extent of supported features applicable to each language and format, see Support Notes 1889.

The table below summarizes the EDI-related standards supported in MapForce.

Standard	Description
ASC X12	ASC X12 is an industry standard for document interchange. MapForce supports versions: 3040, 3050, 3060, 3070, 4010, 4020, 4030, 4040, 4041, 4042, 4050, 4051, 4052, 4060, 5010, 5011, 5012, 5020, 5030, 5040, 5050, 6010, 6020, 6030, and 6040. **The default ASC X12 version in MapForce is 6040.** ASC X12 components have "virtual" nodes into which EDI parser error information/data is written depending on the settings you select in the EDI Validation Settings dialog box (see EDI component validation 454). An X12 997 Functional Acknowledgement can be generated from any X12 document. For more information about ASC X12, see http://www.x12.org/.
HIPAA X12	HIPAA is based on the X12 EDI 5010 standard, but has its own specialized versions which are natively supported by MapForce 2011 Release 3 or later. **The default HIPAA X12 version in MapForce is release A2 of the HIPAA implementation specs (TR3).** Previous releases are available for download on the MapForce Components page of the Altova website (https://www.altova.com/mapforce/download/libraries).
HL7	HL7 is an industry standard for data exchange between medical applications and is an abbreviation of "Health Level Seven". MapForce supports versions 2.2 to 2.6. **The default HL7 version in MapForce is 2.6.** A separate installer for the additional HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the MapForce Components page of the Altova website (https://www.altova.com/mapforce/download/libraries). The XML-based HL7 version 3.x is supported in MapForce 2023 using XML schema components. The **MapForceExamples** project contains a sample that maps a HL7 V2.6 to a HL7 V3 XML file (HL7V260_to_HL7V3.mfd).

Standard	Description
	For more information about HL7, see https://www.hl7.org/.
IATA PADIS	PADIS (Passenger and Airport Data Interchange Standards) is an industry standard for the exchange of passenger and airport data using EDI documents. MapForce supports versions: 00.1 to 08.1. **The default PADIS version in MapForce is 08.1.** For more information about IATA PADIS, see http://www.iata.org/Pages/default.aspx.
NCPDP SCRIPT	SCRIPT is a standard for exchanging electronic information in the US healthcare sector developed by the NCPDP (National Council for Prescription Drug Programs), see http://www.ncpdp.org. MapForce supports mapping data between files in SCRIPT format and any other format supported by MapForce, including XML, CSV, Microsoft Excel, databases, and others. The supported NCPDP SCRIPT versions are: 8.1, 10.6, 10.10. **The default NCPDP SCRIPT version in MapForce is 10.10.**
SAP IDocs	SAP IDocs (intermediate documents) documents are used to exchange business data between SAP R/3 and non-SAP applications. The documents are a form of intermediate data storage which can be exchanged between different systems. For more information about SAP IDocs, see the SAP Help Portal.
SWIFT	SWIFT (Society for Worldwide Interbank Financial Telecommunication) is a global provider of financial messaging services. SWIFT MT (Message Text) is a message type used in SWIFT network. MapForce supports mapping SWIFT MT data to or from any other MapForce-supported format, including XML, CSV, JSON, databases, and others. MapForce supports the following SWIFT MT releases: 2019 or newer. Note that the default installed version is SWIFT 2022. Support for mapping data to or from SWIFT MX / ISO 20022 formats is implicit, since these formats are XML based.
TRADACOMS	TRADACOMS (Trading Data Communications) is a UK-specific Electronic Data Interchange standard used in the retail business. MapForce implements the base TRADACOMS specification as laid out in the "TRADACOMS Manual of Standards for Electronic Data Interchange", published in January 1993 by the Article Numbering Association (ANA) UK, now known as GS1 UK. For other TRADACOMS versions, MapForce can be customized to process new message types, data elements, and code values, by means of configuration files.
UN/EDIFACT	UN/EDIFACT (the United Nations rules for Electronic Data Interchange for Administration, Commerce and Transport) is a set of standards for electronic document interchange. MapForce supports the following UN/EDIFACT Directories: D.93A or later. The latest UN/EDIFACT version is the default installed version in MapForce. A separate installer for all previous versions back to D.93A, including the configuration files required for Interactive EDI mappings, is available on the Altova MapForce Libraries

Standard	Description
	page.
ODETTE	ODETTE EDI is a set of standards used for message interchange in the automotive industry. For more information about ODETTE EDI message types, see *Messaging Resources*.

The EDI configuration versions installed by default with MapForce (formatted in bold in the table above) typically change with every new release. For example, when a newer version of the UN/EDIFACT standard appears, it becomes the default one in MapForce for the next upcoming release. Note that the default EDI versions are mainly suitable for trial and demo purposes, but not for production, because they change constantly.

To design an EDI mapping for production, make sure to download and install the required specific version of EDI configuration files from the Altova website (https://www.altova.com/mapforce/download/libraries). After installation, the installed EDI versions become available in MapForce in addition to the default versions, see Adding EDI Components to the Mapping 447.

For a single EDI component that you have added to the mapping area, MapForce can process multiple message types of any single standard EDI release. For example, this enables you to read data from an interchange file which includes more than one message type, or to generate as mapping output an interchange file which contains multiple message types. You can also process data from multiple instance files containing different message types. Note that operations such as the ones described above are only meaningful when they are supported by the underlying specification.

If your organization is using EDI specifications that are customizations of those already supported by MapForce, it is possible to adapt MapForce to your custom EDI specification by means of configuration files. For example, you can add custom messages, segments, data elements, code lists, and so on (or change the existing ones), thus creating a custom EDI collection similar to any of the EDI-related directories that are already supported by MapForce. For more information, see Customizing EDI Structure 519 and Customizing EDI Validation 552.

4.3.1 EDI Terminology

The following list describes some of the common terms used in conjunction with EDI-related standards.

Message

A message represents a business document. In ASC X12, messages are called "transaction sets". A message is composed of multiple building blocks known as "segments". MapForce supports one or multiple different message types in a single EDI component (see Adding EDI Messages as Mapping Components 447).

Segment

A segment represents a single "record" in a message. Segments are identified by a two or three character ID at the beginning of the segment. A group of related elements comprise a segment tag (or segment ID, in ASC X12). Segments of a transaction can be defined as mandatory or conditional (optional).

Data Element

A data element is an individual data field within a segment. A data element can be thought of as a field, because it usually contains one type of data (for example, a name or an address). Data elements can be simple or composite elements. In the latter case, they consist of component elements or sub-elements.

Separator

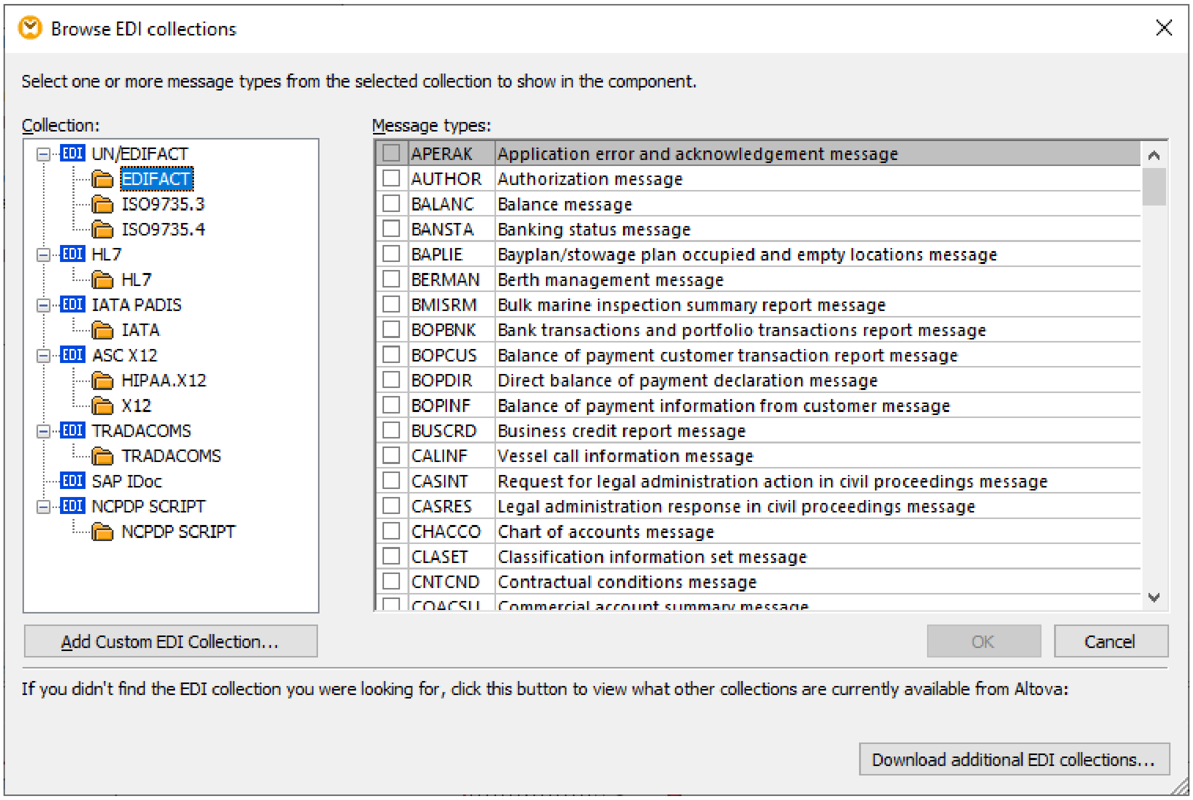
Data elements are delimited by separator characters, referred to as "separators". Separators are usually ASCII characters which are less likely to occur in text fields (for example, colons, asterisks, pipes, bars, ampersands, etc). The separators vary depending on the EDI component type. By default, MapForce recognizes the standard separators applicable for each EDI flavor. However, separators can be customized if necessary (see EDI Component Settings 450).

4.3.2 Adding EDI Components to the Mapping

To map data to or from EDI files, the EDI component must be added to the mapping first. The EDI component represents the structure of the EDI message (or messages) on the mapping. Once the EDI component is added to the mapping, you can draw mapping connections to or from it, similar to any other component types supported by MapForce (see Working with Components 78). When necessary, you may also configure the EDI component to include multiple messages. This would enable you to read data from multiple EDI files (one message per file), as shown in Processing Multiple Input or Output Files Dynamically 1313, or process different message types within the same file.

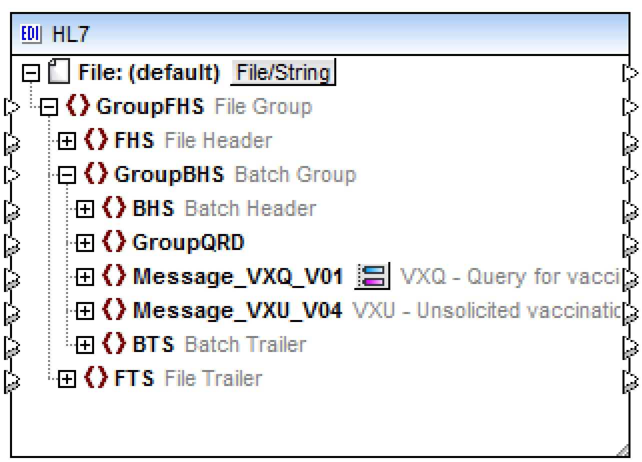
To add the EDI component to the mapping:

1. On the **Insert** menu, click **EDI**.
2. Click the EDI collection, and select the check box next to the message type that you want to include. You can also include multiple messages, by selecting their corresponding check boxes. For example, in the image below, two messages of the HL7 collection are being added to the mapping.



3. Click OK. You are now prompted to supply a sample EDI file.
4. If you want to read data from the EDI component, browse for the sample EDI file; otherwise, click **Skip**.

The component now appears on the mapping, and it includes the structure of both messages selected previously. If you want to add additional messages, or remove existing ones, click the button that appears next to the first message.



Sample HL7 mapping component which includes two messages

You can now draw mapping connections from this EDI component to a target one, or vice versa. Several MapForce mapping design (.mfd) files that illustrate mapping data to or from various EDI flavours are available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder:

Format	Mapping sample
ASC X12	• **X12_To_Xml_Order.mfd** • **Xml_To_X12_Order.mfd**
HIPAA X12	• **HIPAA_837D.mfd**
HL7	• **HL7_MultiMessageTypes.mfd** • **HL7V260_To_HL7V3.mfd**
IATA	• **IATA_FlightInformationReport.mfd**
SAP IDoc	• **IDOC_Order.mfd**
UN/EDIFACT	• **EDI_Order.mfd** • **EDI_Order_DB.mfd** • **EDI_OrderReverse.mfd**

4.3.3 Running EDI Mappings

Mappings which contain EDI components are generally supported when the mapping is set to the BUILT-IN [27] transformation language. This enables not only previewing the mapping directly in MapForce but also deploying it to a server environment for execution by MapForce Server. Mappings where *quick* (non-standard) EDI conversion to XML was enabled also require the BUILT-IN engine, see Generating XML from EDI [458].

With most EDI-related formats, it is additionally possible to generate mapping code in C++, C#, Java. For details, see the Support Notes [1889].

Once an EDI mapping has been designed in MapForce, you can run in the following ways:

- You can preview the result of the data transformation directly in MapForce, by clicking the **Output** tab. The output generated by the mapping will be saved to the desired location with the desired name, depending on how you configured both MapForce and the target component.
- You can run the mapping on a server machine (Windows, Linux, macOS) with MapForce Server installed standalone, through an API call or at the command-line interface. For further information, see Compiling Mappings to MapForce Server Execution Files [1378].
- You can deploy the mapping to a server machine (Windows, Linux, macOS) through HTTP (or HTTPS) where MapForce Server runs under FlowForce Server management. In such setups, you can run mappings as a scheduled or on demand job, or as a Web service, see Deploying Mappings to FlowForce Server [1381].
- For mappings that support C++, C#, or Java languages, you can generate program code from the mapping. The generated program performs the same result as if the mapping is previewed in MapForce; it can be run standalone or integrated into your existing code, see Code Generator [1492].

To configure only the settings applicable to an EDI component (either source or target), right-click the component title bar, and select **Properties** from the context menu, see EDI Component Settings [450]. For settings applicable to other component types, see Changing the Component Settings [82].

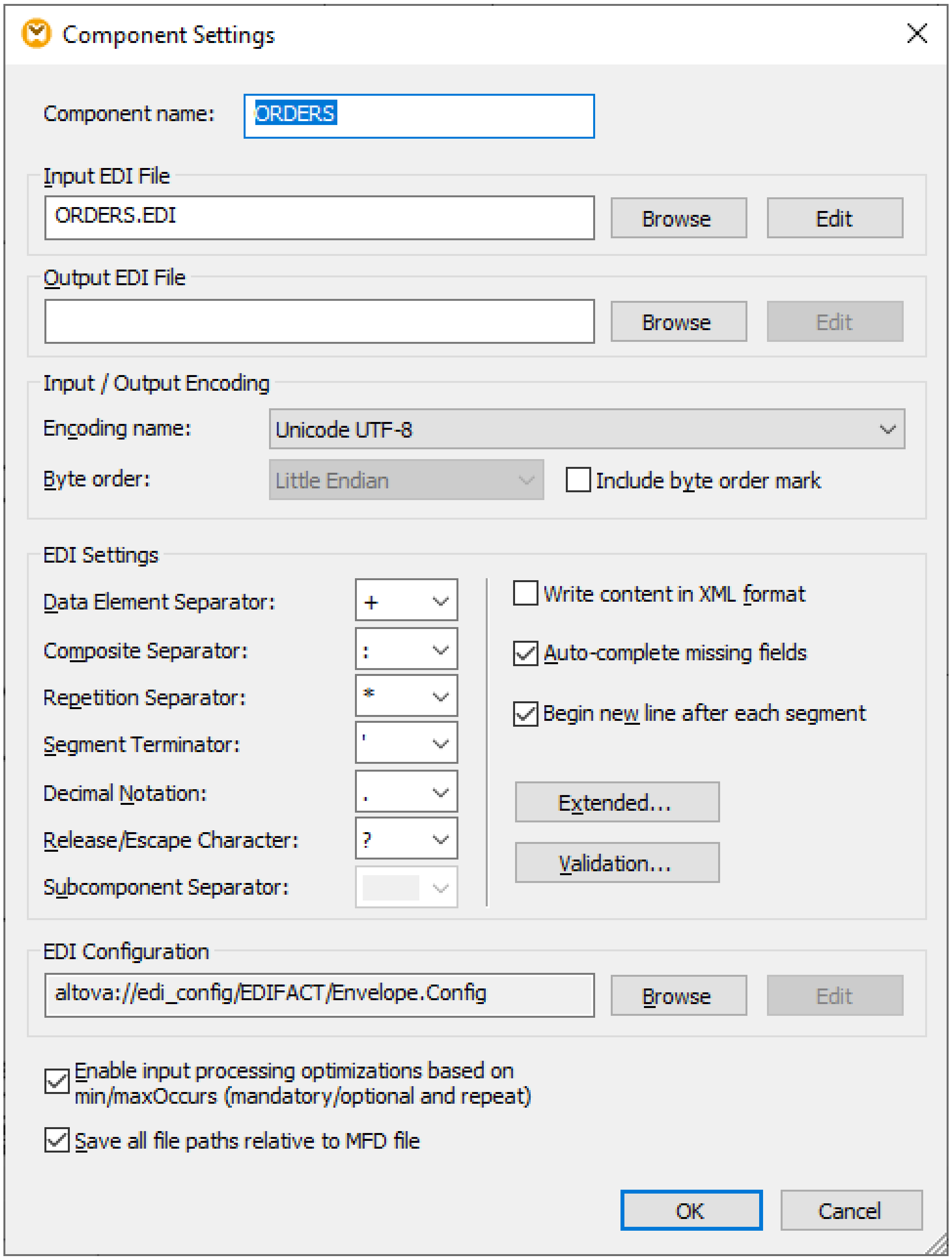
To configure the mapping settings for the entire mapping, right-click the mapping, and select **Mapping Settings** from the context menu, see Changing the Mapping Settings 114.

To access the code generation settings, go to **Tools | Options**, and then click the **Generation** tab.

4.3.4 EDI Component Settings

This topic explains how to define EDI component settings and shows how to use mappable EDI delimiters in a mapping. After you have added an EDI component to the mapping area, you can configure the component's settings in the **Component Settings** dialog box (*see screenshot below*). You can open the **Component Settings** dialog box in one of the following ways:

- Double-click the EDI component's title bar in the mapping.
- Select an EDI component in the mapping and select the **Component | Properties** menu command.
- Right-click the EDI component's title bar and then click **Properties**.



The available settings are listed below. Note that some settings are not available if the currently selected component does not support them.

⊟ Component name

The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change the name of the component, be aware of the following:

- If you intend to deploy the mapping to FlowForce Server, the component name must be unique.
- It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.

Input EDI file

Specifies the EDI source file from which MapForce will read data.

Output EDI file

Specifies the EDI target file to which MapForce will write data.

Input/Output Encoding

This section allows you to specify the following settings of the output instance file: encoding, byte order, and whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the *Default encoding for new components* option. You can access this option in **Tools | Options | General**. For details, see Options 1473.

EDI Settings

This group of settings enables you to define EDI delimiters (e.g., segment terminators, data element separators). Note that the settings are available only if they are supported by the EDI format you are using. The EDI separators entered in this dialog box always take effect when MapForce writes EDI files. When MapForce reads EDI files, the separators take effect only if the input file does not define/contain its own separators (e.g., EDIFACT files without the UNA segment).

If an EDI input component/file contains separator definitions (e.g., an X12 file with an ISA segment), the existing separators override any separators defined in the **Component Settings** dialog box for that file. You can use non-printable characters as separators, by typing `x` followed by the hexadecimal ASCII character code into one of the combo boxes: e.g., `x1e` for the RS control character (ASCII record separator, decimal code 30).

The *Subcomponent Separator* applies to the HL7 standard. The default value of this separator is the ampersand (`&`).

Write content in XML format

Selecting this check box is meaningful only for a target EDI component. This option generates XML output. For details, see Example: Switch between XML and EDI Output 461.

Auto-complete missing fields

This option applies to target EDI components. When this check box is selected, MapForce fills in the values of some data fields automatically. This applies only to those fields where this operation would not contradict the specification of the currently selected EDI format. For more information about the auto-completion option, see the following topics:

- ASC X12 Automatic Data Completion Rules 463
- UN/EDIFACT Automatic Data Completion Rules 510

Begin new line after each segment

This option applies to target EDI components. When the check box is selected, MapForce adds a CR/LF (carriage return/line feed) character after each EDI segment. The EDI standard ignores these lines if they are present in a message.

Extended

Opens a dialog box where you can define additional settings for the current EDI component. The available settings vary, depending on the type of EDI component.

Validation

Opens a dialog box where you can define the validation settings for the current EDI component. For details, see EDI Component Validation 454.

EDI Configuration

Provides information about the EDI configuration used by the current component. For details, see How MapForce resolves EDI configurations 521.

Enable input processing optimizations based on min/maxOccurs

This option allows special handling of sequences that have only one item (e.g., required attributes and child elements with `minOccurs` and `maxOccurs="1"`). In this case, the first item of the sequence is extracted, and the item is then directly processed as an atomic value (and not as a sequence).

If the input data is *not valid* against the schema, the mapping will cause an error. To allow the processing of invalid input, clear this check box.

Save all file paths relative to MFD file

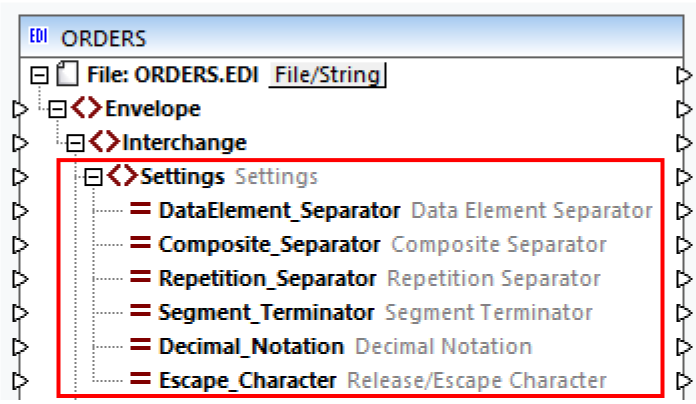
When this option is enabled, MapForce saves the file paths displayed in the **Component Settings** dialog box relative to the location of the MapForce Design (**`.mfd`**) file. This setting affects the following files:

- The input EDI file (if present)
- The output EDI file (if present)

See also Relative and Absolute Paths 85.

Mappable EDI delimiters

MapForce enables you to map EDI delimiters (e.g., data element separators, segment terminators etc.). Mappable EDI delimiters are currently supported for the following EDI standards: EDIFACT, X12, and NCPDP SCRIPT. In an EDI component, the mappable EDI delimiters are available in the `Settings` element (the child of the `Interchange` element) (*screenshot below*).



The screenshot above shows an extract from an EDIFACT message, in which each delimiter is represented by one child node, and all of these delimiters can be configured in the **Components Settings** dialog box (*subsection above*). Note that the `Settings` element shows only those delimiters that are configurable for this particular EDI standard.

Note about 997 and 999 messages

When you create a mapping for the X12/HIPAA.X12 997 and 999 messages, you must create a copy-all connection [97] between the `Settings` node of the input instance and the `Settings` node of the 997 and 999 messages of the output.

Source vs. target EDI components

In source components, the delimiter nodes provide the actually used delimiters from the source instance, which might differ from the delimiters set in the **Components Settings** dialog box. For target components, the delimiter nodes allow setting the separators at runtime, which means that mapped delimiters override those set in the **Component Settings** dialog.

Validation checks

Note the following validation-related information:

- Not connecting anything to a separator node will not change the existing behavior.
- Mapping and empty sequence or a sequence of more than one string will cause a runtime error.
- Repetition separators in EDIFACT: Mapping an empty space to a separator will mark the separator as unused. The UNA segment will have a space instead of the separator. Mapping an empty space to an actually needed separator will cause an error, because the separator has not been set.
- Mapping a string that does not contain exactly one character will cause a runtime error.
- Mapping the same character to two or more separators will result in a runtime error.

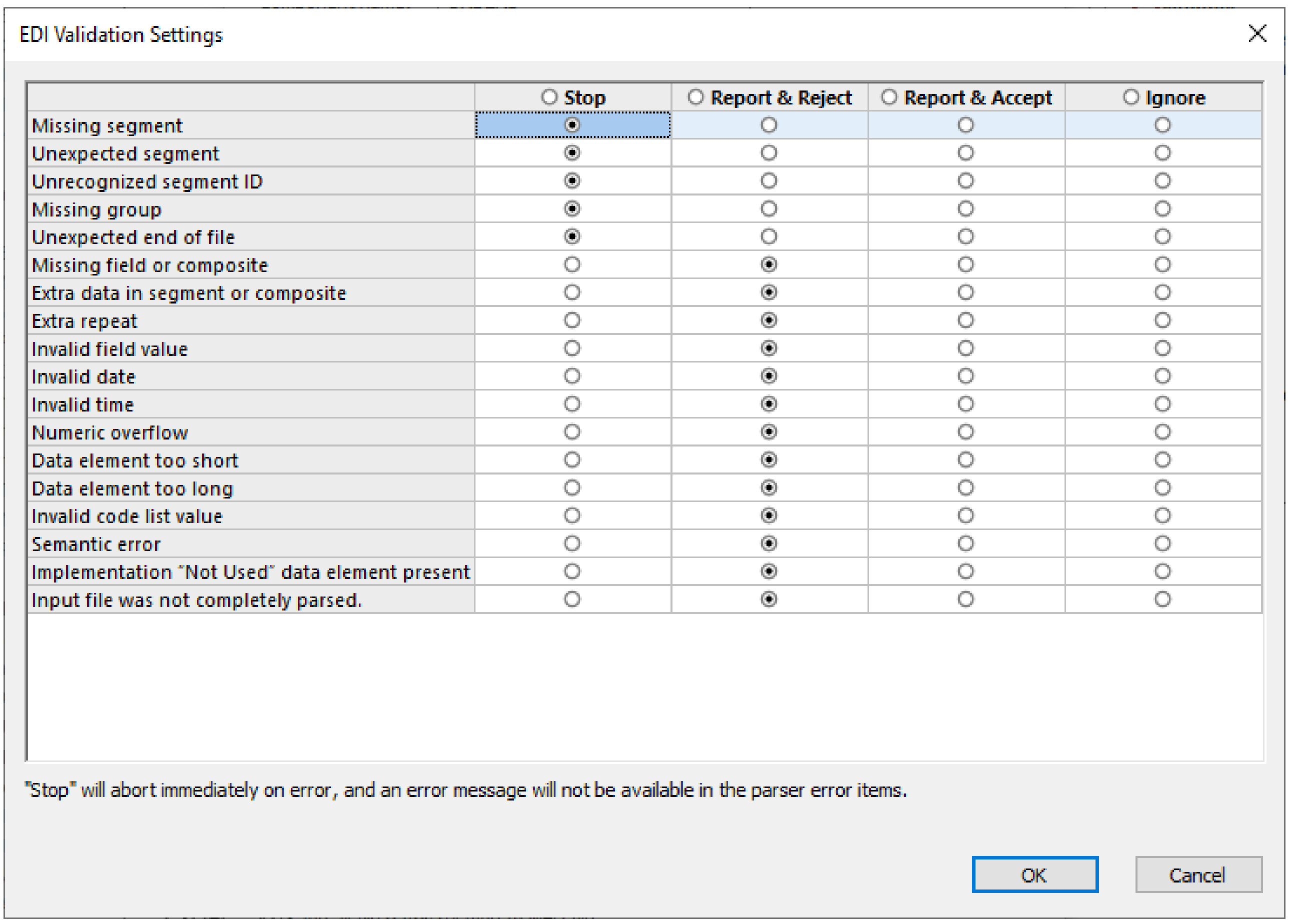
4.3.5 EDI Component Validation

MapForce validates all supported EDI source and target documents when the mapping is executed. This happens either when you preview the mapping in MapForce, by clicking the **Output** tab, or when the mapping is run by MapForce Server, or by a C++, C#, or Java program. You can also validate explicitly the EDI output generated by the mapping. To do this, first click the **Output** tab, and then click the **Validate Output**  toolbar button (see also Validating the Mapping Output [105]).

MapForce performs validation of EDI documents as specified in this documentation for each individual supported format. The validation process is, however, not a full EDI syntax or semantic validation. For example, there may exist instances where the possible values of a field rely on standards external to EDI and cannot be easily expressed as a finite set of values that would foresee all processing possibilities. To address this limitation, you can customize the validation rules on a case-by-case basis with the help of configuration files (see Customizing EDI Validation 552).

At a very basic level, some validation options can be configured from the MapForce graphical user interface. For example, you can choose to ignore specific errors, or stop the mapping execution when specific errors are encountered.

To customize the validation settings, double-click an EDI component and click the **Validation** button in the Component Settings dialog box (see also EDI Component Settings 450). The default validation settings are shown below.



EDI Validation Settings dialog box

For each validation event listed above, you can take any of the following actions: **Stop**, **Report & Reject**, **Report & Accept**, **Ignore**.

Stop is used to catch fatal errors and stops the execution of the mapping.

Report & Reject provides information about the error and rejects the current message. **Report & Accept** provides information about the error and accepts the current message. The actual error information is visible in the Messages window of MapForce. For source X12 and HIPAA components, the validation errors are also logged inside the mapping component, which enables you to generate X12 997 or X12 999 reports (see Validation Results and X12 Acknowledgement 457).

Ignore ignores the validation event. If you select this action, no information is provided about the error, so use it cautiously.

If unexpected segments are encountered while processing an EDI mapping, only MapForce and MapForce Server have the capability to recover and attempt to continue parsing. The generated C++, C#, or Java code cannot recover from unexpected segments and will stop processing. Unexpected segments include the following: incorrect segment order, missing mandatory segment, incorrect segment name, additional segments that are not present in the configuration files.

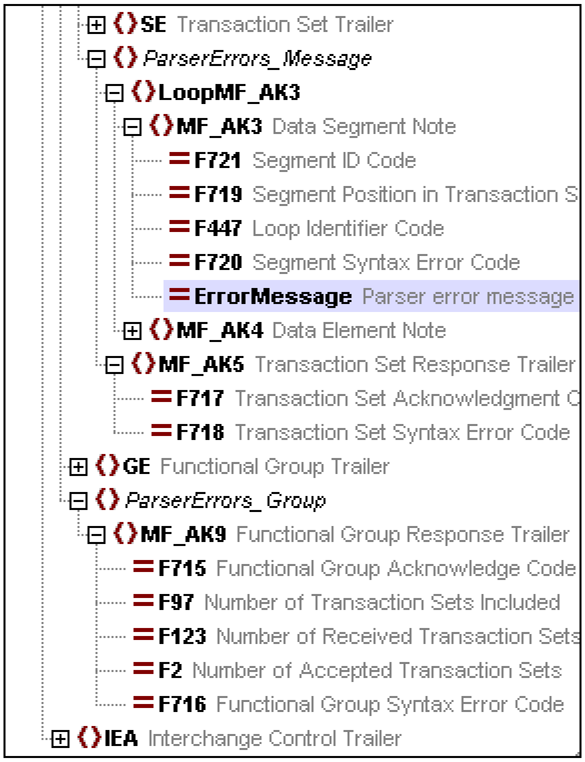
The EDI validation events for which you can configure custom actions (**Stop**, **Ignore**, etc) are as follows.

Event	Description	Error might occur for a
Missing segment	A mandatory segment is missing or the occurrence is less than a specified minimum.	source component
Unexpected segment	A segment is defined in the specification but not in this message.	source component
Unrecognized segment ID	A segment was found which is not defined in the specification.	source component
Missing group	A mandatory group is missing or the occurrence is less than the specified minimum.	source component
Unexpected end of file	The instance cannot be parsed since some data is missing.	source component
Missing field or composite	A mandatory field or composite is missing, or the occurrence is less than the specified minimum.	source or target component
Extra data in segment or composite	The input instance contains additional data that is not expected by the syntax description.	source component
Extra repeat	The actual number of fields within a segment/composite exceeds the specified maximum number.	source or target component
Invalid field value	A numeric field contains an invalid character.	source component
Invalid date	A date field contains either an invalid character or the values for the month or the day are invalid.	source component
Invalid time	A time field contains either an invalid character or the value for the hours or the minutes are invalid.	source component

Event	Description	Error might occur for a
Numeric overflow	A numeric value overflows its defined domain. This error is only supported within the generated code.	source component
Data element too short	The length of a data element is less than the specified minimum value.	source or target component
Data element too long	The length of a data element is greater than the specified maximum limit.	source or target component
Invalid code list value	The value of the data element does not match any of the preconfigured code values. See also Changing Possible Code Values of Data Elements 535.	source or target component
Semantic error	A semantic error has occurred.	source or target component
Implementation "Not Used" data element present	An element exists in the input file which is not allowed by the HIPAA configuration file (`maxOccurs="0"`).	source component
Input file was not completely parsed	The input file was not completely parsed.	source component

Validation results and X12 Acknowledgment

When MapForce validates an X12 or HIPAA component, the validation results are placed in "virtual" items at the base of the EDI component (under "ParserErrors_Message" and "ParserErrors_Group", as shown in the image below). This information can be used to generate X12 997 or X12 999 files that report the status of the interchange (see Generating an X12 997 Functional Acknowledgment 464 and Generating an X12 999 Implementation Acknowledgment 465).



With X12 or HIPAA.X12 components, the validation actions available on the EDI Validation Settings dialog box (discussed above) have the following meaning:

Stop will stop the execution of the mapping without generating a report message.

Report & Reject and **Report & Accept** will provide information in the "ParserErrors_Message" and "ParserErrors_Group" items of the EDI component which can be mapped further.

Depending on the setting **Reject** or **Accept,** the "Functional Group Acknowledge Code F715" and the "Transaction Set Acknowledgment Code F717" will contain either:

the value 'R' for *'Rejected'* or
the value 'E' for '*Accepted, but errors were noted'*. The errors also appear in the Messages window.

Ignore ignores the specific error. No information is provided within the "ParserErrors_Message" and "ParserErrors_Group" items.

4.3.6 Generating XML from EDI

As mentioned previously, you can map EDI data to or from any of the other data formats supported by MapForce. EDI mappings are not significantly different from non-EDI mappings, and involve the same level of control over the generated data.

Beyond regular mappings that require some design effort, MapForce also provides the ability to *quickly* convert EDI files to XML format, with a few button clicks—this does not require writing any XML schema or drawing mapping connections manually. This is a convenience feature that enables you to export EDI files to an alternative format—which is significantly easier to read. The XML output is well-formed and can be saved to a file, like most outputs generated by MapForce.

Making use of this feature neither requires nor provides an XML schema. The generated XML simply replicates all the data from the source EDI component, outputting the data element names and their values, and respecting the hierarchy of elements. Any fields that do not contain data are omitted. Note that the exact structure of the generated XML depends on the source EDI data and cannot be altered. If you need full control over the generated XML data, create a standard EDI to XML mapping instead, which would enable you to tackle each individual mapping item, use filters, functions, and so on. For examples of standard EDI to XML mappings, see Example: Mapping UN/EDIFACT to XML 512, Example: Mapping HIPAA X12 to XML 468, and Example: Mapping HL7 to XML 472.

Prerequisites

To *quick* convert EDI files to XML with MapForce, the transformation language of the mapping must be set to BUILT-IN; no other transformation languages are supported. This requirement specifically applies to *quick* EDI to XML conversion. Regular EDI to XML mappings support additional languages, see EDI 444.

Quick EDI to XML conversion

After you have added an EDI component to the mapping, there are two ways to generate XML from it:

1. If the EDI component reads data from an EDI instance file (in other words, if it is a source component), right-click its title bar and select **Creating Mapping to XML** from the context menu. For a step-by-step example, see Example: Quick Convert EDI to XML 459.

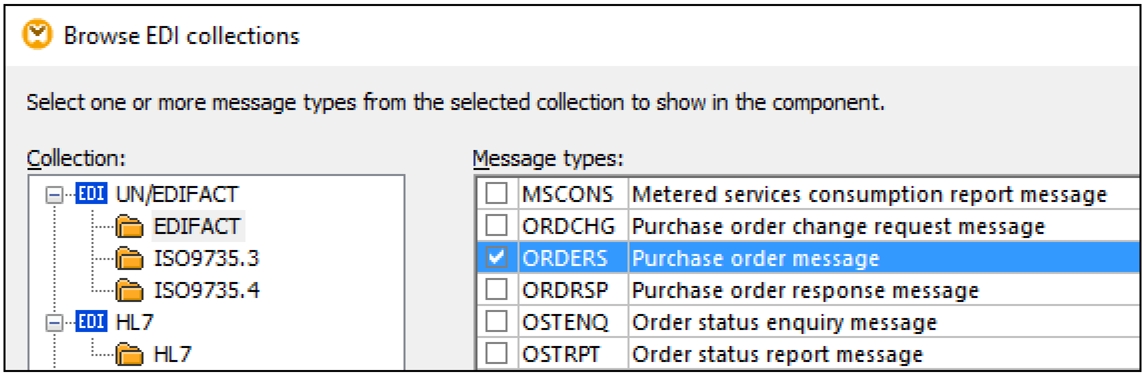
Note: You can also do this for target components, if you would like to create a new output in addition to the existing EDI output, and thus turn the mapping into a "chained" mapping. For more information, see Chained Mappings 1246.

2. If the EDI component is a target one, first double-click its title bar to open the component's settings, and then select the **Write content in XML format** check box. For a step-by-step example, see Example: Switch between XML and EDI Output 461.

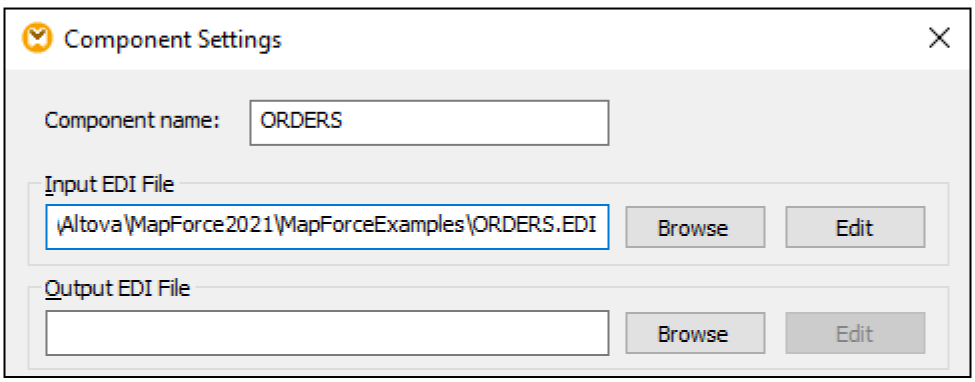
4.3.6.1 Example: Quick Convert EDI to XML

This example shows you how to convert data from an EDI file to XML format with the least amount of effort. The example specifically deals with *quick* conversion and does not involve drawing mapping connections manually. For a regular EDI to XML conversion example, see Example: Mapping UN/EDIFACT to XML 512.

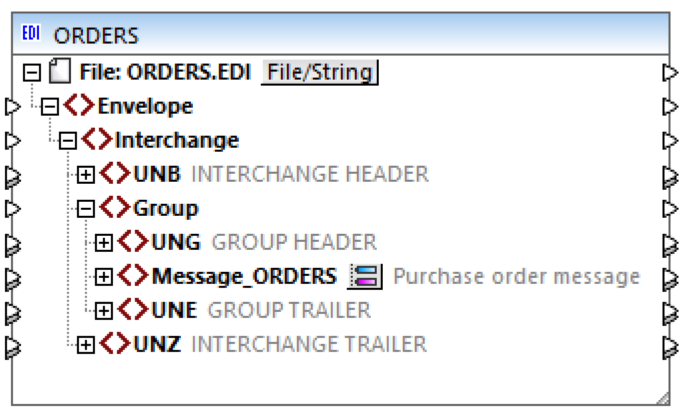
1. Create a new mapping and select BUILT-IN as transformation language. See also Transformation Languages 27.
2. On the **Insert** menu, click **EDI**.
3. Select the **EDIFACT** collection and the **ORDERS** message type, and click **OK**.



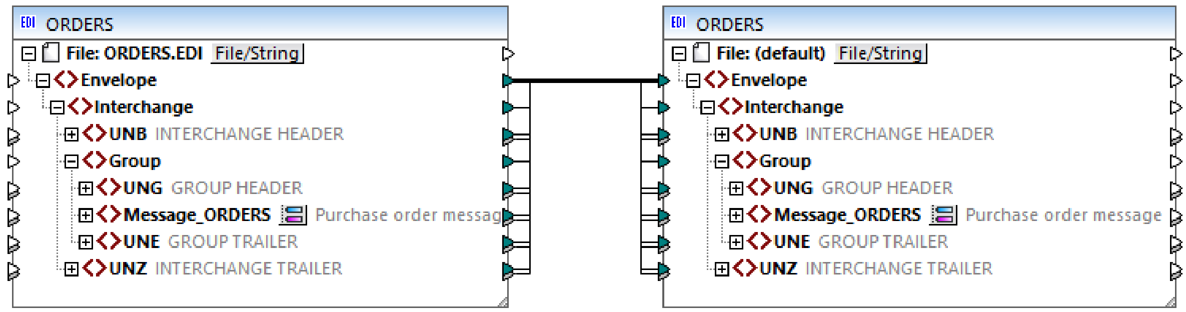
4. When prompted to supply a sample EDI file, click **Browse** and open the **ORDERS.EDI** file from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory. After you open the file, the Component Settings dialog box opens. This enables you to review the settings of the EDI component before adding it to the mapping. You can change these settings 450 at any time later if required. Notice that the **ORDERS.EDI** file appears as **Input EDI file**.



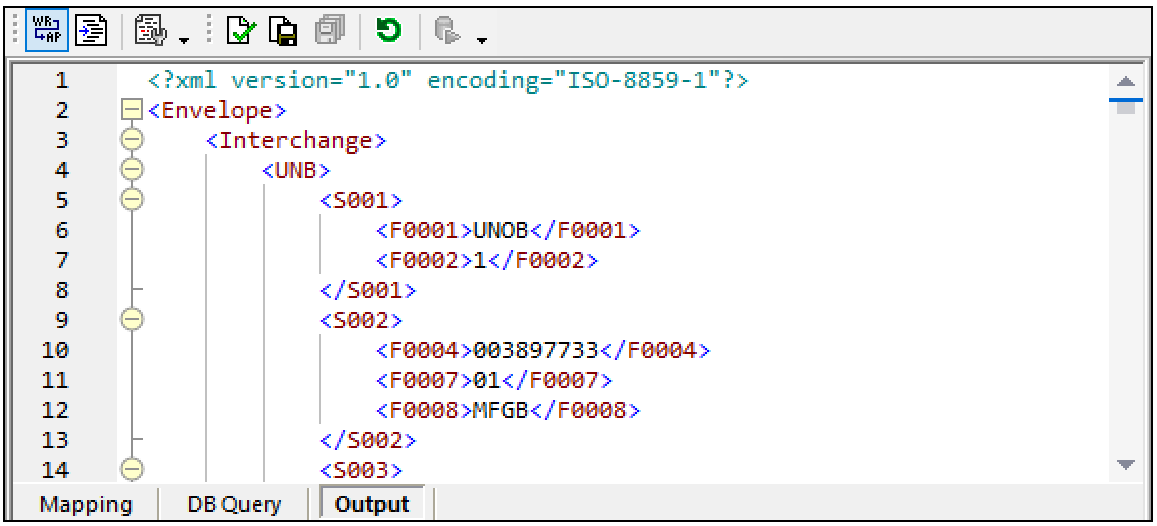
5. Click **OK**. The EDI component is now displayed in the mapping area.



6. Right-click the title bar of the ORDERS component and select **Create Mapping to XML** from the context menu. This creates a target XML component and makes all connections to it automatically through a Copy-All 97 connection.



7. Click the **Output** tab to preview the generated XML.

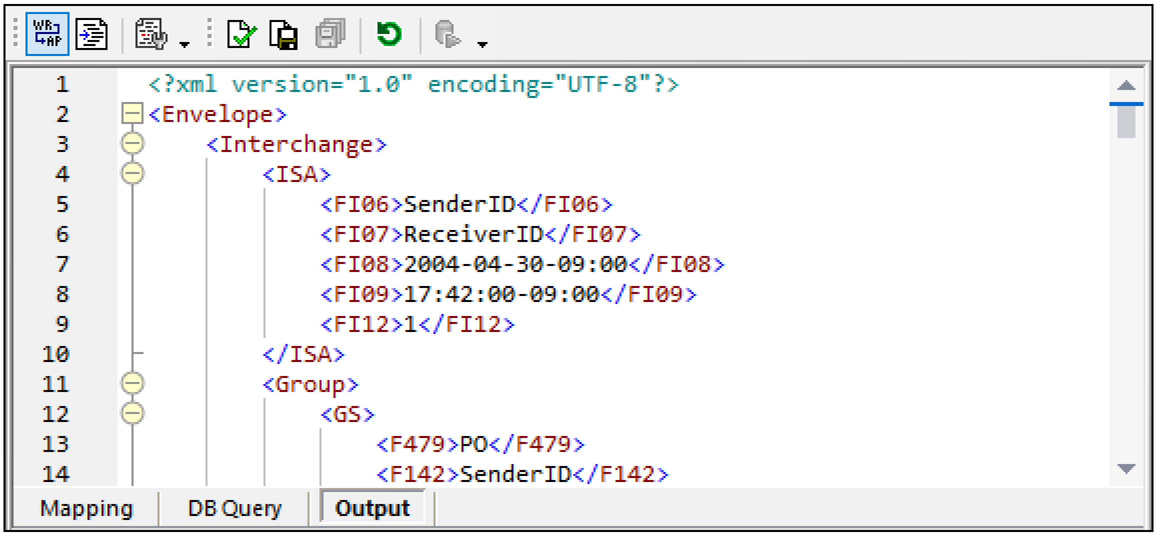


4.3.6.2 Example: Switch between XML and EDI Output

This example shows you how to configure a target EDI component so that it produces XML output instead of EDI output. The configuration is minimal and involves changing a single property, as illustrated below.

1. Open the following mapping: **<Documents>\Altova\MapForce2023\MapForceExamples\Xml_To_X12_Order.mfd**.
2. Change the transformation language to BUILT-IN (On the **Output** menu, select **Built-In Execution Engine**).
3. Double-click the title bar of the target **850** component. The EDI Component Settings dialog box opens.
4. Select the **Write content in XML format** check box.

You can now preview the mapping by clicking the **Output** tab. As illustrated below, the EDI component now generates XML output instead of EDI.



4.3.7 ASC X12

ASC X12 is an industry standard for document interchange. MapForce supports versions: 3040, 3050, 3060, 3070, 4010, 4020, 4030, 4040, 4041, 4042, 4050, 4051, 4052, 4060, 5010, 5011, 5012, 5020, 5030, 5040, 5050, 6010, 6020, 6030, and 6040.

The default ASC X12 version in MapForce is 6040.

ASC X12 components have "virtual" nodes into which EDI parser error information/data is written depending on the settings you select in the EDI Validation Settings dialog box (see EDI component validation[454]). An X12 997 Functional Acknowledgement can be generated from any X12 document.

For more information about ASC X12, see http://www.x12.org/.

4.3.7.1 ASC X12 Validation Rules

When validating ASC X12 documents, MapForce performs the following checks:

- Whether an ISA and an IEA segment exist
- Whether ISA/I01 contains a legal authorization information qualifier.
- Whether ISA/I03 contains a legal security information qualifier.
- Whether the two ISA/I05 segments contain legal interchange ID qualifiers.
- Whether ISA/I08 contains a well-formed date value.
- Whether ISA/I09 contains a well-formed time value.
- Whether ISA/I13 contains a legal Boolean value.
- Whether ISA/I14 contains a legal interchange usage indicator.
- Whether ISA/I12 and IEA/I12 contain the same value.
- Whether IEA/I16 contains the correct number of function groups in the interchange.

Each function group is checked:

- If there is a matching GS and GE pair.
- Whether GS/373 contains a well-formed date value.
- Whether GS/337 contains a well-formed time value.
- Whether GS/28 and GE/28 contain the same value.
- Whether GE/97 contains the correct number of messages in the function group.

Each message is checked:

- If there is a matching ST and SE pair.
- Whether ST/143 contains the correct message identifier.
- Whether ST/329 and SE/329 contain the same value.
- Whether SE/96 contains the correct number of segments in the message.

4.3.7.2 ASC X12 Automatic Data Completion Rules

When generating an ASC X12 file, MapForce may automatically create certain fields or values, according to the rules below (subsequently referred to as "automatic data completion", or "auto-completion" rules). To disable this behavior, clear the **Auto-complete missing fields** check box in EDI Component Settings 450.

Automatic data completion for EDI/X12 makes sure:

- That an ISA and IEA pair exist on the interchange level.
- That if either GS or GE exist, the other ID also exists.
- That there is at least one ST/SE pair on the message level.
- That ISA/I01 and ISA/I03 exist. If they do not contain data, 00 is inserted.
- That ISA/I02 and ISA/I04 exist. If they do not contain data, ten blanks are inserted.
- That both ISA/I05 segments exist. If they do not contain data, ZZ is inserted.
- That ISA/I08 exists. If it does not contain data, the current date in EDI format is inserted.
- That ISA/I09 exists. If it does not contain data, the current time in EDI format is inserted.
- That ISA/I65 exists. If it does not contain data, the repetition separator is inserted.
- That ISA/I11 exists. If it does not contain data, the interchange control version number from the user-defined settings is inserted (see the setting **Extended | Interchange control version-number** of the EDI Component Settings 450).
- That ISA/I12 exists.
- That ISA/I13 exists. If it does not contain data, the request acknowledgment setting is used (see the setting **Extended | Request Acknowledgement** of the EDI Component Settings 450).
- That ISA/I14 exists. If it does not contain data, P is inserted.
- That ISA/I15 exists. If it does not contain data, the composite separator from the user-defined settings is inserted (see the setting **Composite Separator** of the EDI Component Settings 450)..
- That IEA/I16 exists. If it does not contain data, the number of function groups in the interchange is calculated and inserted.
- That IEA/I12 exists. If it does not contain data, the value from ISA/I12 is copied.

Note: Any fields not mentioned here are not inserted or created. The correct values cannot be ascertained automatically. Given a (target) parent element A (in the target EDI component) with child items x, y, and z - where y is mandatory, parent element A will only be created in the output file if the mandatory child element "y" in the target component has been mapped.

The potentially existing function group is checked as follows:

- That GS/373 exists. If it does not contain data, the current date in EDI format is inserted.
- That GS/337 exists. If it does not contain data, the current time in EDI format is inserted.
- That GE/97 exists. If it does not contain data, the number of messages in the function group are calculated and inserted.
- That GE/28 exists. If it does not contain data, the value from GS/28 is copied.

Message checking makes sure:

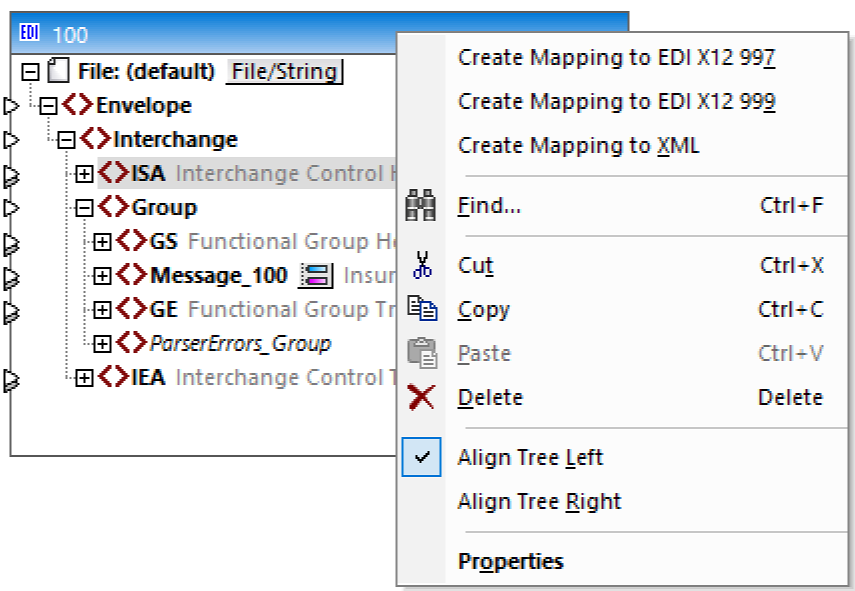
- That ST/143 exists. If it does not contain data, the name of the message is inserted.
- That SE/96 exists. If it does not contain data, the number of segments in the message is calculated and inserted.
- That ST/329 and SE/329 exist. If SE/329 does not contain data, the value from ST/329 is copied.

4.3.7.3 Generating an X12 997 Functional Acknowledgment

The X12 997 (Functional Acknowledgment) reports the status of the EDI interchange. All errors encountered during processing of the document are reported in it. MapForce can automatically generate an X12 997 component in the main mapping area, and automatically create the necessary connectors.

To generate the EDI 997 Functional Acknowledgment:

1. On the **Insert** menu, click **EDI**, and add the X12 component to the mapping. When prompted to select as instance file, browse for the file for which you want to create the X12 997 acknowledgement.
2. Right-click the title bar of the EDI component and select **Create Mapping to EDI X12 997** (note this command is enabled only for X12 components).



This creates an EDI 997 mapping component and automatically connects the items needed to generate the X12 997 acknowledgment. For an example, see the **X12_To_XML_Order.mfd** mapping design file available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. To see the acknowledgment file generated by this mapping, click the **Preview** ( ) button of the EDI 997 component, and then click the **Output** tab.

```
ISA+00+          +00+          +ZZ+ReceiverID     +ZZ+SenderID       +100128+1113+!+00505+000000000+1+P+:'
GS+01+ReceiverID+SenderID+20100128+11130544+1+X+05012'
ST+997+0001'
AK1+PO+1+05012'
AK2+850+12345'
AK5+A'
AK9+A+1+1+1'
SE+6+0001'
GE+1+1'
IEA+1+000000000'

```

To save the X12 997 Acknowledgement file:

- Click the **Output** tab, and then click the **Save generated output** ( ) toolbar button. If you intend to generate program code from the mapping, enter a file name in the "Output EDI File" field of the Component Settings dialog box (see EDI Component Settings 450).

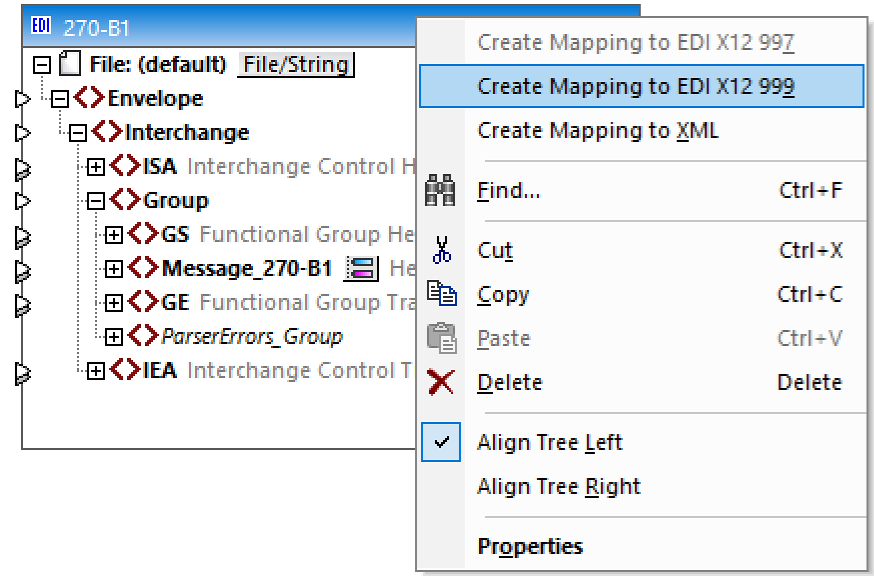
4.3.7.4 Generating an X12 999 Implementation Acknowledgment

The X12 999 (Implementation Acknowledgment) reports X12 non-compliance or application errors. Because it is a super-set of the 997 Functional Acknowledgement, 999 can be used instead of 997 to accept or reject transaction sets based on either X12 or HIPAA Implementation Guide syntax requirements.

All errors encountered during processing of the document are reported in it (for example, "Required Segment Missing", "Required Data Element Missing", "Code Value Not Used in Implementation", etc). MapForce can automatically generate the X12 999 component on the mapping, and automatically create the necessary mapping connections.

To generate the X12 999 Implementation Acknowledgment:

1. On the **Insert** menu, click **EDI**, and add the X12 or HIPAA component to the mapping. When prompted to select as instance file, browse for the file for which you want to create the X12 999 acknowledgement.
2. Right-click the title bar of the EDI component and select **Create mapping to EDI X12 999** (note this command is enabled only for X12 and HIPAA components).



This creates an EDI 999 mapping component and automatically connects the items needed to generate the X12 999 implementation acknowledgment. To preview the generated acknowledgment file, click the **Output** tab. If the mapping has multiple target components, first click the **Preview** ( ) button, and then click the **Output** tab.

```
ISA+00+          +00+          +ZZ+ReceiverID     +ZZ+SenderID       +110502+1533+!+00505+000000000+1+P+:'
GS+FA+ReceiverID+SenderID+20110502+15333751+1+X+006020'
ST+999+0001'
AK1+PO+1+006020'
AK2+850+12345'
IK5+A'
AK9+A+1+1+1'
SE+6+0001'
GE+1+1'
IEA+1+000000000'

```

To save the 999 Acknowledgement file:

- Click the **Output** tab, and then click the **Save generated output** ( ) toolbar button. If you intend to generate program code from the mapping, enter a file name in the "Output EDI File" field of the Component Settings dialog box (see EDI Component Settings 450).

4.3.8 HIPAA X12

HIPAA X12 is the latest version of the standard for electronic health care records established by the US Department of Health and Human Services for electronic medical data transactions between insurers, providers, and employers, based on EDI X12 version 5010.

MapForce supports the latest release, A2, of the HIPAA implementation specs (TR3). Older releases are downloadable as a separate ZIP file from Altova website.

HIPAA components are similar to ordinary ANSI X12 components, and MapForce supports the following transactions:

X12 name	Message name
X279A1	"Health Care Eligibility Benefit Inquiry (270)"
X279A1	"Health Care Eligibility Benefit Response (271)"
X212	"Health Care Claim Status Request (276)"
X212	"Health Care Information Status Notification (277)"
X214	"Health Care Claim Acknowledgment (277)"
X217	"Health Care Services Review - Request for Review (278)"
X217	"Health Care Services Review - Response (278)"
X218	"Payroll Deducted and Other Group Premium Payment for Insurance Products (820)"
X220A1	"Benefit Enrollment and Maintenance (834)"
X221A1	"Health Care Claim Payment/Advice (835)"
X222A1	"Health Care Claim: Professional (837)"
X224A2	"Health Care Claim: Dental (837)"
X223A2	"Health Care Claim: Institutional (837)"
X231A1	"Implementation Acknowledgment For Health Care Insurance (999)"

Differences to standard X12 message handling is that MapForce:

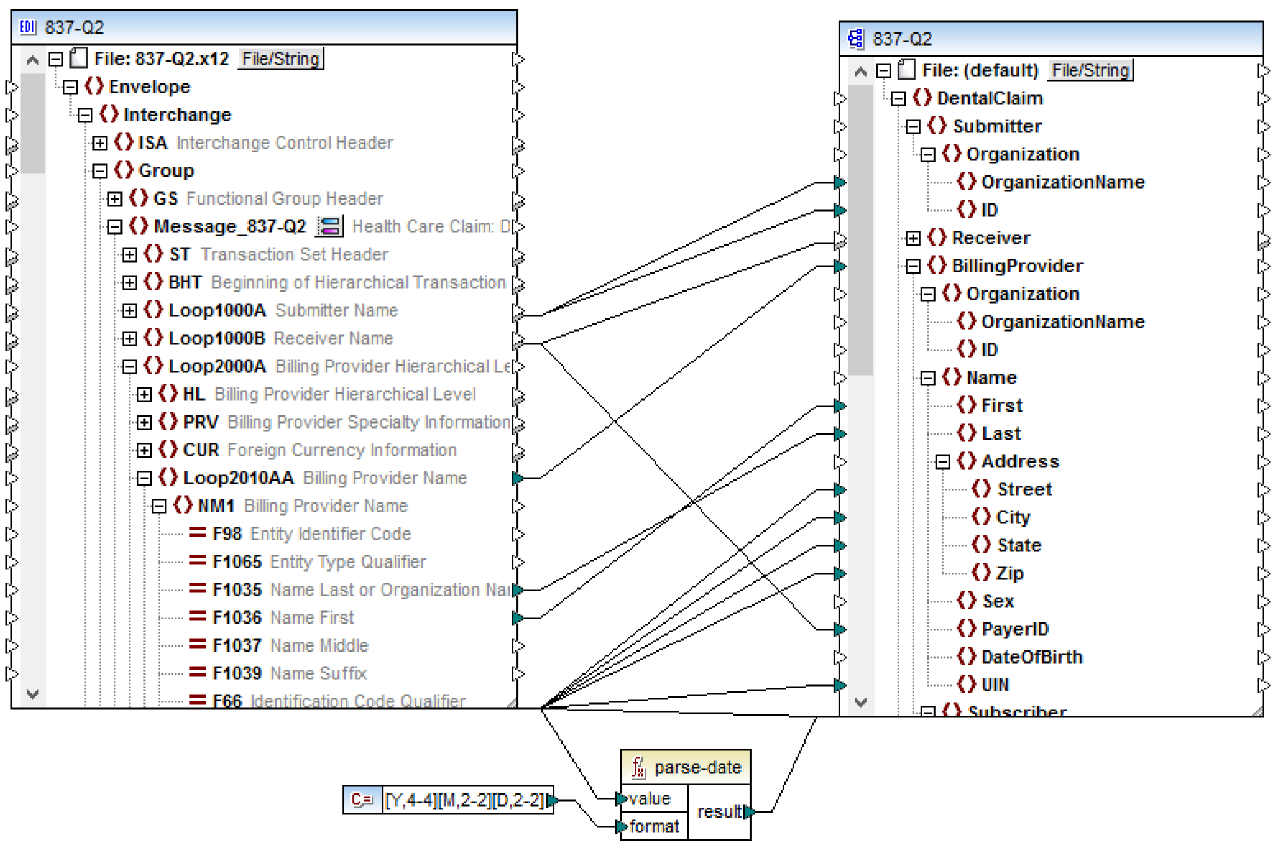
- automatically maintains the hierarchy of HL segments
- supports so called floating structures (in 837 messages)
- auto-completes and validates more fields.

MapForce also supports the automatic generation of HIPAA X12 999 Implementation Acknowledgment, which is similar to 999 Implementation Acknowledgement of the X12 standard (see Generating an X12 999 Implementation Acknowledgment 465).

Multiple messages per interchange (i.e. EDI component) is also supported for HIPAA components.

4.3.8.1 Example: Mapping HIPAA X12 to XML

This example illustrates how to map a file which contains an "837-Q2 Health Care Claim: Dental" transaction to an XML target file. It is accompanied by a mapping available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\HIPAA_837D.mfd**.



HIPAA_837D.mfd

The mapping can be created as shown in the steps below.

Step 1: Add the source HIPAA X12 file to the mapping

1. On the **Insert** menu, click **EDI**.
2. Select the collection "HIPAA.X12", and the message type "837-Q2 Health Care Claim: Dental".
3. When prompted to supply an instance file, open the source file available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\837-Q2.x12**.
4. When prompted, adjust the settings of the newly added HIPAA X12 component if necessary (for more information, see EDI Component Settings [450]).

Step 2: Add the target schema to the mapping

1. On the **Insert** menu, click the **XML Schema/File**, and open the **837-Q2.xsd** file from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.

2. When prompted to supply a sample XML file, click **Skip** and select **DentalClaim** as the root of the target document.

Step 3: Add a function to format the date and draw the mapping connections

In the source HIPAA file, the DMG segment includes a date, in the format `19490501`. To convert this date to a format suitable for the target XML file `1949-05-01`, we will add a date conversion function, as follows:

1. Drag the **parse-date** 998 function from the Libraries window into the mapping. For more information about functions, see Add a Function to the Mapping 905.
2. On the **Insert** menu, click **Constant**, and enter `[Y,4-4][M,2-2][D,2-2]` as constant value.
3. Draw the following connections:

From	To	Purpose
Constant	`format` input item of the **parse-date** function	Supplies to the function the format according to which the date should be parsed.
`F1251`	`value` input item of the **parse-date** function	Supplies to the function the actual value to be formatted.
`result` output item of the **parse-date** function	`DateOfBirth` input item of the XML target	Supplies the value returned by the function to the `DateOfBirth` XML node.

4. Draw the remaining mapping connections as shown in the example mapping (these are too many to be listed here; essentially, each connection copies values from the source HIPAA file to the target XML file. For basic information about mapping connections, see Working with Connections 89.)

4.3.8.2 Considerations when Mapping HIPAA Transactions

HL segment hierarchy

HL segments are generated automatically and do need to be mapped manually. It is however possible to map values if there is such a need.

Autocompletion

The configuration files for HIPAA transactions contain many codes and are used for auto-completion of fields whenever possible. Autocompletion is context sensitive, so fields in different places may have different values. To disable auto-completion, clear the **Auto-complete missing fields** check box in the Component Settings dialog box (see EDI Component Settings 450).

Validation

Validation is similar to auto-completion in that it is also context sensitive. Validation errors are handled in the same way as per X12 validation (see ASC X12 Validation Rules 462).

Editing of configuration files

HIPAA configuration files use a newer schema version than the X12 configuration files (see EDI Configuration Schemas 523). Therefore, HIPAA configuration files are similar to X12 configuration files, with a few additions:

- Code values at field level
- Conditions
- Completion flags
- “Not Used” Data Elements

Code values

Field instances can have a built-in code list which is used for validation and/or autocompletion. Values can be manually added or removed (see Changing Possible Code Values of Data Elements 535).

```
<Data ref="F365" info="Communication Number Qualifier" nodeName="F365_1">
<Values>
        <Value Code="EM" />
        <Value Code="FX" />
        <Value Code="TE" />
</Values>
</Data>
```

Conditions

Conditions work in conjunction with values. As there can be multiple repetitions of a segment, or group (loop), they are identified by a specific condition. A Condition requires a value to be present for it to be fulfilled, otherwise the specific group is not found.

For example, Loop1000 is repeated multiple times, but each instance has a different semantic meaning.

To tell which is which, MapForce uses a condition that specifies that Loop1000A must have a Value Code of “41”, in Field F98 of segment NM1. If this is not the case, then Group name Loop1000A is not found.

```
<Group name="Loop1000A" info="Submitter Name">
  &lt;Segment name="NM1" info="Submitter Name">
    <Condition path="F98" />
    <Data ref="F98" info="Entity Identifier Code">
        <Values>
          <Value Code="41" />
        </Values>
    </Data>
(...)
```

Condition codes are auto generated in the target component, if they contain a single value.

Configuration files are generated by a method that ensures that condition values are guaranteed to be unique across the sequence of repeating segments/loops.

If there is a need to edit conditions, or their values, the uniqueness constraint must be taken into account.

Completion flags

Autocompletion can be adjusted on the configuration file level. A new element "Completion", having three attributes, has been introduced to define this:

- singleConditions – auto-complete single conditions for all fields in the target component
- singleValues – auto-complete single values for all fields in the target component
- HL – generate appropriate fields in the HL segment in the target component (for all HL segments)

  (Where "1" means true)

```
<Message>
  <MessageType>837-Q2</MessageType>
  <Completion singleConditions="1" singleValues="1" HL="1" />
  <Description>Health Care Claim: Dental</Description>
(…)
```

"Not Used" Data Elements

HIPAA omits several optional elements that are present in standard X12 transactions. These optional elements are hidden in MapForce components.

The validation engine checks to make sure that these unused fields are not present in source files. Since these fields cannot be mapped to the target component (because they are hidden), they cannot appear in target output files either.

These fields are defined to have the maxOccurs attribute equal to 0, in the configuration files. If necessary, you can manually hide (or unhide) specific fields by using the maxOccurs attribute.

```
<Data ref="F1037" minOccurs="0" info="Submitter Middle Name or Initial" />
<Data ref="F1038" minOccurs="0" maxOccurs="0" />
<Data ref="F1039" minOccurs="0" maxOccurs="0" />
```

Scope of validation

Semantic validation is not supported in MapForce. This means that “situational” fields are simply treated as optional.

4.3.9 HL7 Version 2

HL7 is an industry standard for data exchange between medical applications and is an abbreviation of "Health Level Seven". MapForce supports versions 2.2 to 2.6.

The default HL7 version in MapForce is 2.6.

A separate installer for the additional HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the MapForce Components page of the Altova website (https://www.altova.com/mapforce/download/libraries).

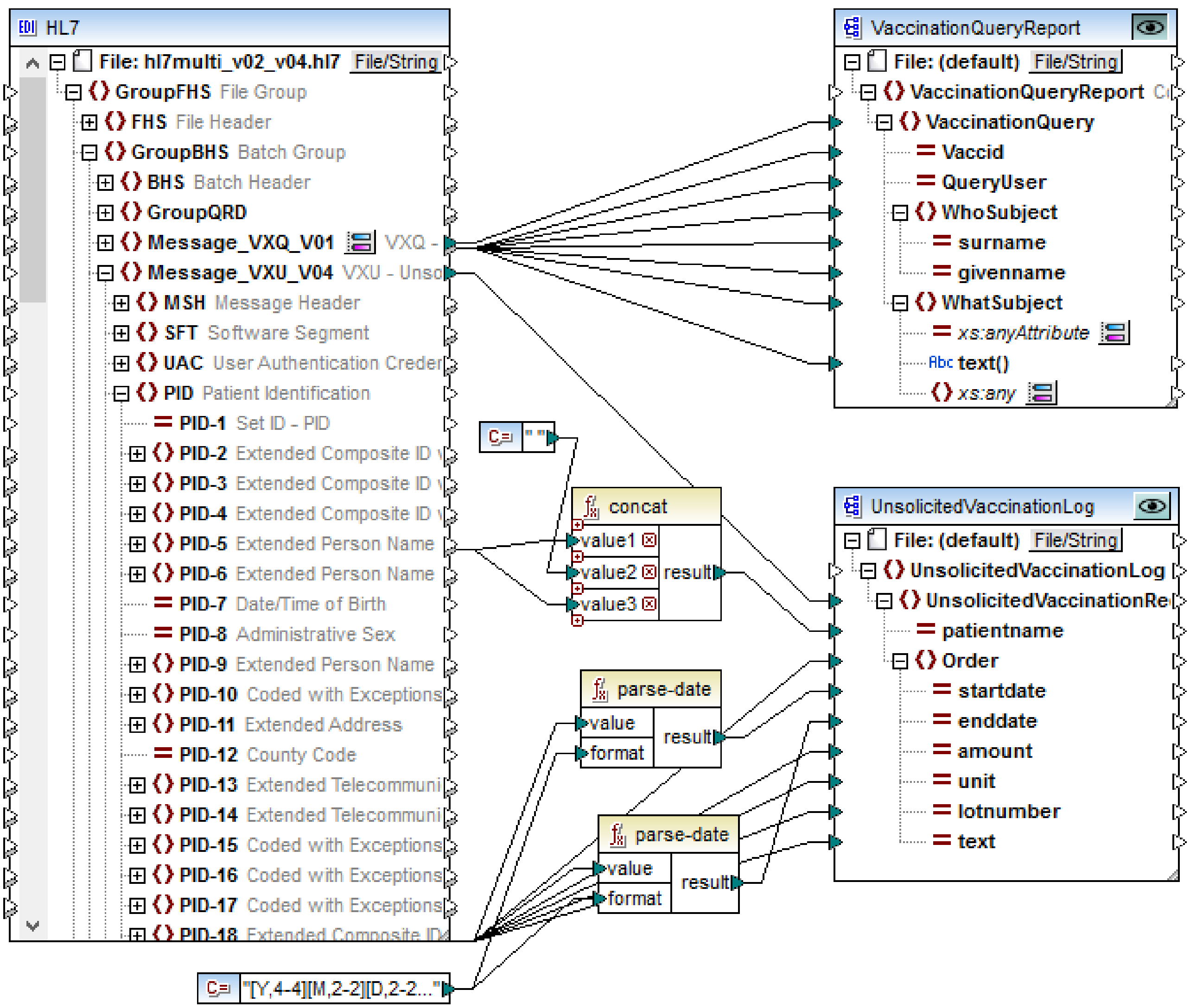
The XML-based HL7 version 3.x is supported in MapForce 2023 using XML schema components.

The **MapForceExamples** project contains a sample that maps a HL7 V2.6 to a HL7 V3 XML file (HL7V260_to_HL7V3.mfd).

For more information about HL7, see https://www.hl7.org/.

4.3.9.1 Example: Mapping HL7 to XML

This example illustrates how to map a file which contains two HL7 messages ("Query for vaccination record" and "Unsolicited vaccination record update") to two separate XML target files. It is accompanied by a mapping available at the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\HL7_MultiMessageTypes.mfd.



HL7_MultiMessageTypes.mfd

The mapping can be created as shown in the steps below.

Step 1: Add the source HL7 file to the mapping

1. On the **Insert** menu, click **EDI**.
2. Select the collection "HL7", and then select the check boxes next to the message types "VXQ - Query for vaccination record" and "VXU - Unsolicited vaccination record update".
3. When prompted to supply an instance file, open the source file available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\hl7multi_v02_v04.hl7**.
4. When prompted, adjust the settings of the newly added HL7 component if necessary (for more information, see EDI Component Settings 450).

Step 2: Add the target schema to the mapping

1. On the **Insert** menu, click the **XML Schema/File**, and open the **VaccinationQueryReport.xsd** file from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
2. When prompted to supply a sample XML file, click **Skip**.

Follow the same steps to add the **UnsolicitedVaccinationLog.xsd** as a mapping component.

Step 3: Add functions and draw the mapping connections

In the source HL7 file, dates have the format `YYYYMMDD`. To convert such dates to a format suitable for the target XML file `YYYY-MM-DD`, we will add a date conversion function, as follows:

1. Drag the **`parse-date`** 998 function from the Libraries window into the mapping. For more information about functions, see Working with Functions 905.
2. On the **Insert** menu, click **Constant**, and enter `[Y,4-4][M,2-2][D,2-2]` as constant value.
3. Draw the following connections:

From	To	Purpose
Constant	`format` input item of the **`parse-date`** function	Supplies to the function the format according to which the date should be parsed.
`RXA-3`	`value` input item of the **`parse-date`** function	Supplies to the function the actual value to be formatted.
`result` output item of the **`parse-date`** function	`startdate` input item of the XML target	Supplies the value returned by the function to the `startdate` XML node.

Follow the same steps to add a new **`parse-date`** function, so as to convert the value of `RXA-4` and copy it to the `enddate` item of the target XML.

The `patientname` item in the second target XML is created by concatenating two fields of the source HL7 file: `FN-1` (Surname) and `XPN-2` (Given Name). To achieve this, this drag the **`concat`** function from the Libraries window onto the mapping. Also, add a constant that will supply the space character to separate the two names.

Draw the remaining mapping connections as shown in the example mapping (these are too many to be listed here; essentially, each connection copies values from the source HL7 file to the target XML files. For basic information about mapping connections, see Working with Connections 89.)

To preview the output generated by any of the target XML components, first click its **Preview** ( ) button, and then click the **Output** tab.

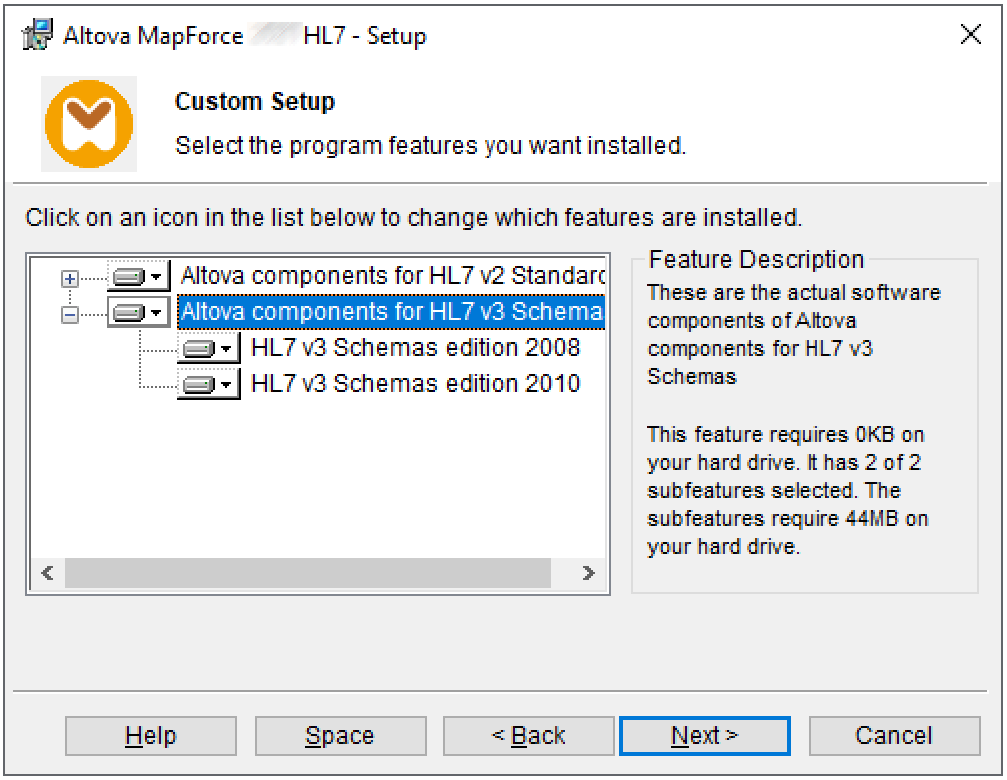
4.3.10 HL7 Version 3

MapForce includes support for mapping data to or from HL7 version 3 documents. Specifically, you can map data from any component type supported by your edition of MapForce to HL7 3.x instance files, or vice versa.

The HL7 version 3 standard is based on XML, unlike the HL7 version 2. Therefore, you can add HL7 version 3 components to the mapping in the same way as adding any other XML schema. Note that MapForce does not install any HL7 version 3 schemas by default. However, you can download and install the HL7 version 3 schemas separately, by following the instructions below.

To install the HL7 version 3 XML schemas:

1. Download the package called "HL7 EDI Configuration Files" from the "MapForce Libraries" page at the Altova download center, https://www.altova.com/mapforce/download#mapforce_libraries. Be aware that this package includes both HL7 version 2 configuration files (which are EDI-based) and HL7 version 3 XML schemas.
2. Double-click the downloaded executable file and follow the installation wizard steps.
3. When prompted to choose a setup type, select **Custom**, and then click **Next**.
4. Choose to install the HL7 version 3 schemas, for example:



After installation, the HL7 version 3 XML schemas are available in the following directory: **C:\Program Files\Altova\Common2023\Schemas\hl7v3**.

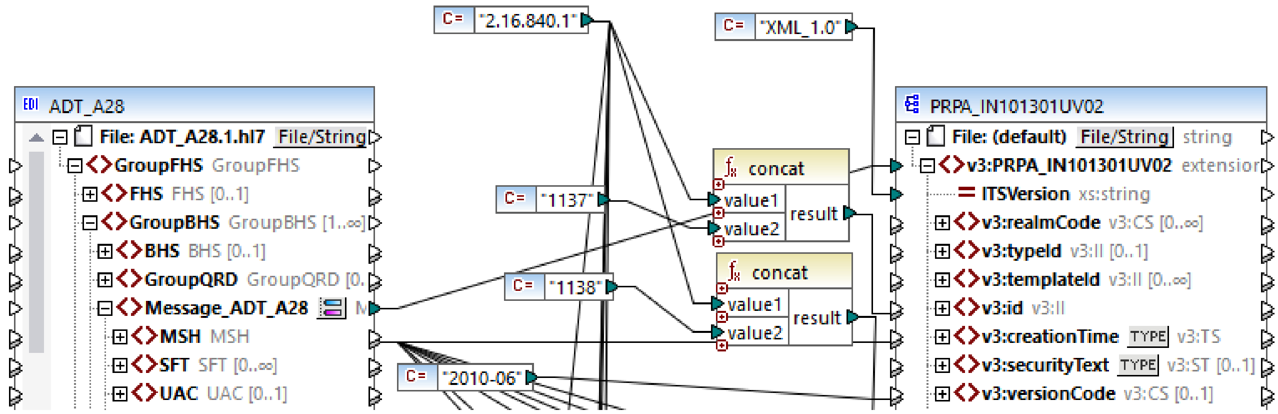
Note: If you use 32-bit MapForce on a 64-bit operating system, adjust the path above to **C:\Program Files(x86)\Altova\Common2023\Schemas\hl7v3**.

Once you have installed the HL7 version 3 XML schemas, you can start a mapping as follows:

1. Run MapForce.
2. On the **Insert** menu, click **XML Schema/File**.
3. Browse for the schema file. You can select for example, the **PRPA_IN101301UV02.xsd file** from the installation directory above.
4. When prompted to supply the instance file, select the HL7 version 3 instance file corresponding to the message type selected previously if you would like to read data from it and map it to some other format. Otherwise, if you plan to generate the instance file (not read data from it), click **Skip**.

You have now added the HL7 version 3 component to the mapping, and can proceed to designing the actual mapping connections. For an introduction to designing mappings with MapForce, see the Tutorials [42].

An example mapping that converts data from an HL7 version 2.6 file to HL7 version 3 file is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\HL7V260_To_HL7V3.mfd**. A fragment of this mapping is illustrated below.



In the mapping illustrated above, the source component is an HL7 version 2.6 instance file. To add such components to the mapping, select the menu command **Insert | EDI**, and then select the required message types from the "HL7" group. This is an EDI component; you can view or change the instance file and other settings pertaining to this component by double-clicking the component's title bar, see also EDI Component Settings 450.

The target component is an HL7 version 3 instance file. This is an XML component, as already stated above, and you can add such components to the mapping by selecting the menu command **Insert | XML Schema/File**. For reference to all the settings that you can change for XML components, see XML Component Settings 123.

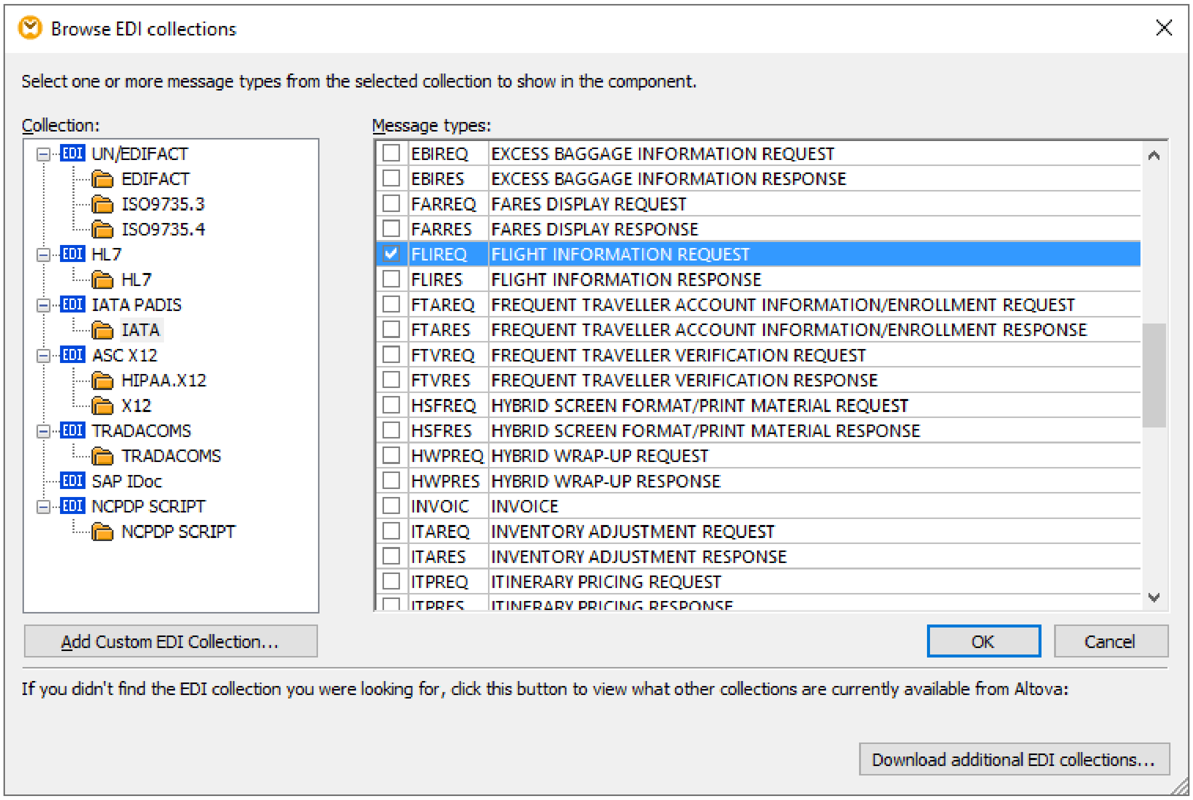
4.3.11 IATA PADIS

PADIS (Passenger and Airport Data Interchange Standards) are a set of messages using the EDIFACT (ISO 9735) syntax.

- MapForce currently supports only messages using the UNH/UNT message header and trailer segments.
- MapForce supports the collections IATA.00.1 to IATA.08.1.

To add an IATA PADIS file as mapping component:

1. On the **Insert** menu, click **EDI**.
2. Click the "IATA" collection under "IATA PADIS", and select the check box next to the required message type(s).

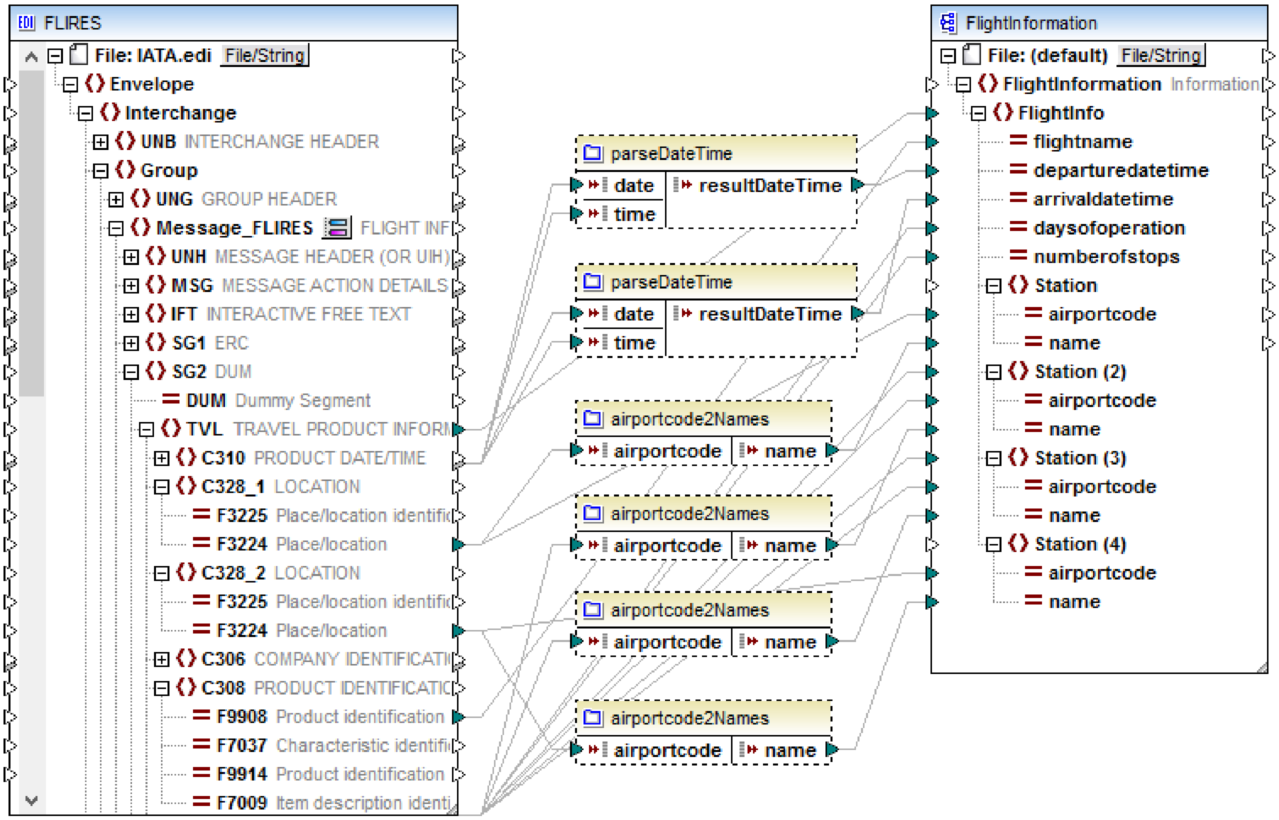


3. When prompted to supply an instance file, click **Browse**, and open the source IATA file to be used as mapping source. If you are mapping to an IATA PADIS component, click **Skip**.

The IATA component is now added to the mapping, and you can draw mapping connections to or from it. To change the settings of the IATA component, right-click it, and then select **Properties** from the context menu (see also EDI Component Settings 450).

4.3.11.1 Example: Mapping IATA to XML

The **IATA_FlightInformationReport.mfd** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder shows an example mapping of an IATA PADIS file to an XML target file.



IATA_FlightInformationReport.mfd

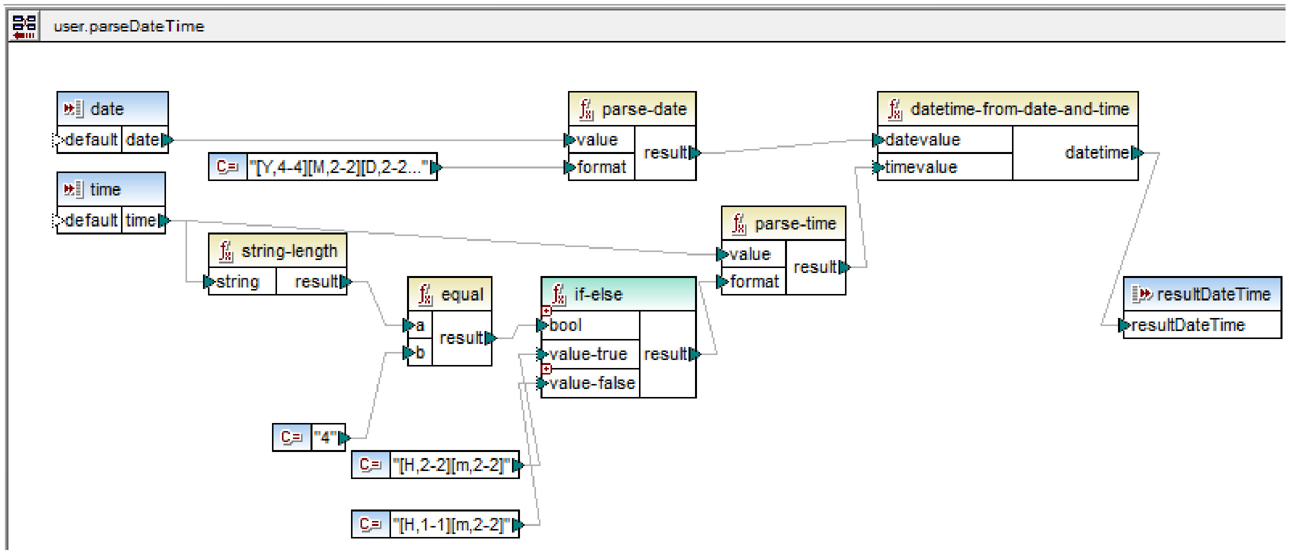
This mapping consists of a source IATA component which includes a FLIRES message, a target XML component, and two user-defined functions, **`parseDateTime`** and **`airportcode2Names`** (which are reused multiple times).

To view the instance file read or written by the source or target components, as well as other settings, right-click the component, and select **Properties** (see also EDI Component Settings [450]).

To view the structure of each user-defined function, double-click it. To go back to the main mapping, click the **Return to Main Mapping** ( ) button available in the upper-left corner of the mapping area. For more information, see User-Defined Functions [927].

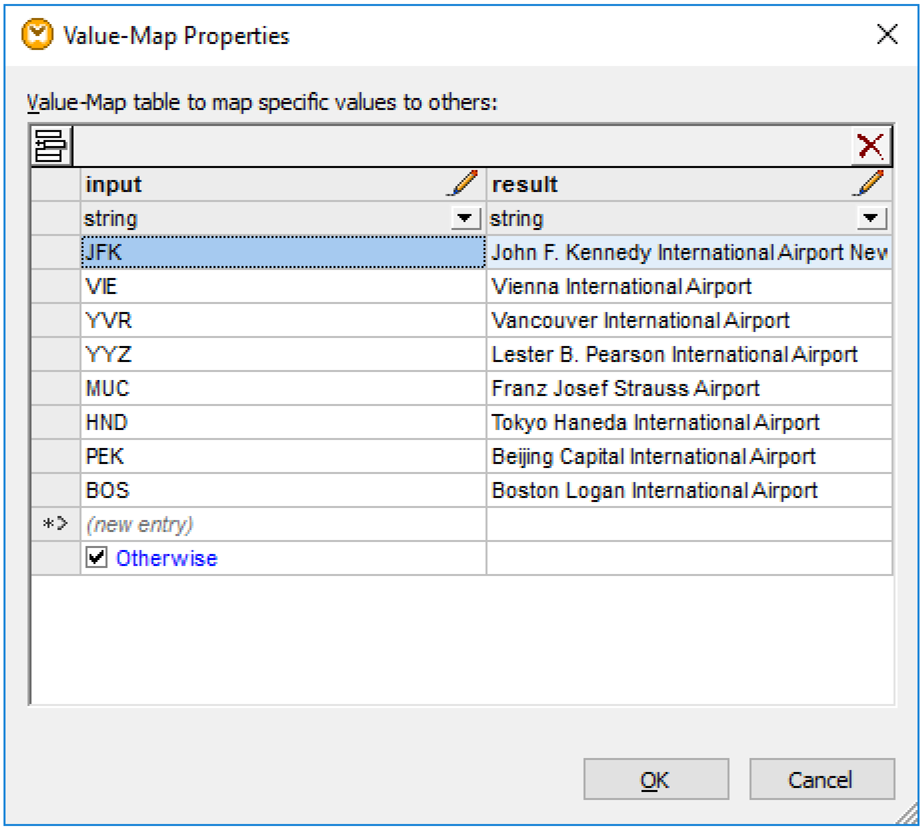
In this mapping, the user-defined functions have the following purpose:

1) The **`parseDateTime`** function receives as arguments the date and time from IATA date and time elements and processes them in order to produce an `xs:dateTime` value required in the target XML file. To achieve this goal, it makes use of various functions available in the MapForce core library, such as



Structure of the parseDateTime function

2) The **airportcode2Names** function receives as argument the F3223 or F3224 field values (the airport code) and returns the full name of the airport. To achieve this, the function contains a Value-Map component which maps each airport code to an airport name, as follows:



Structure of the Value-Map component

For more information about Value-Map components, see Using Value-Maps[884].

On the target side, the XML component uses a technique called "duplication of nodes". Namely, the node `Station` is being duplicated multiple times on the mapping component. This makes it possible to map values from multiple source items to the same target item (see Duplicating Input[83]).

To preview the output produced by the mapping, click the **Output** tab.

4.3.12 NCPDP SCRIPT

SCRIPT is a standard for exchanging electronic information in the US healthcare sector developed by the NCPDP (National Council for Prescription Drug Programs), see http://www.ncpdp.org. MapForce supports mapping data between files in SCRIPT format and any other format supported by MapForce, including XML, CSV, Microsoft Excel, databases, and others. The supported NCPDP SCRIPT versions are: 8.1, 10.6, 10.10.

The default NCPDP SCRIPT version in MapForce is 10.10.

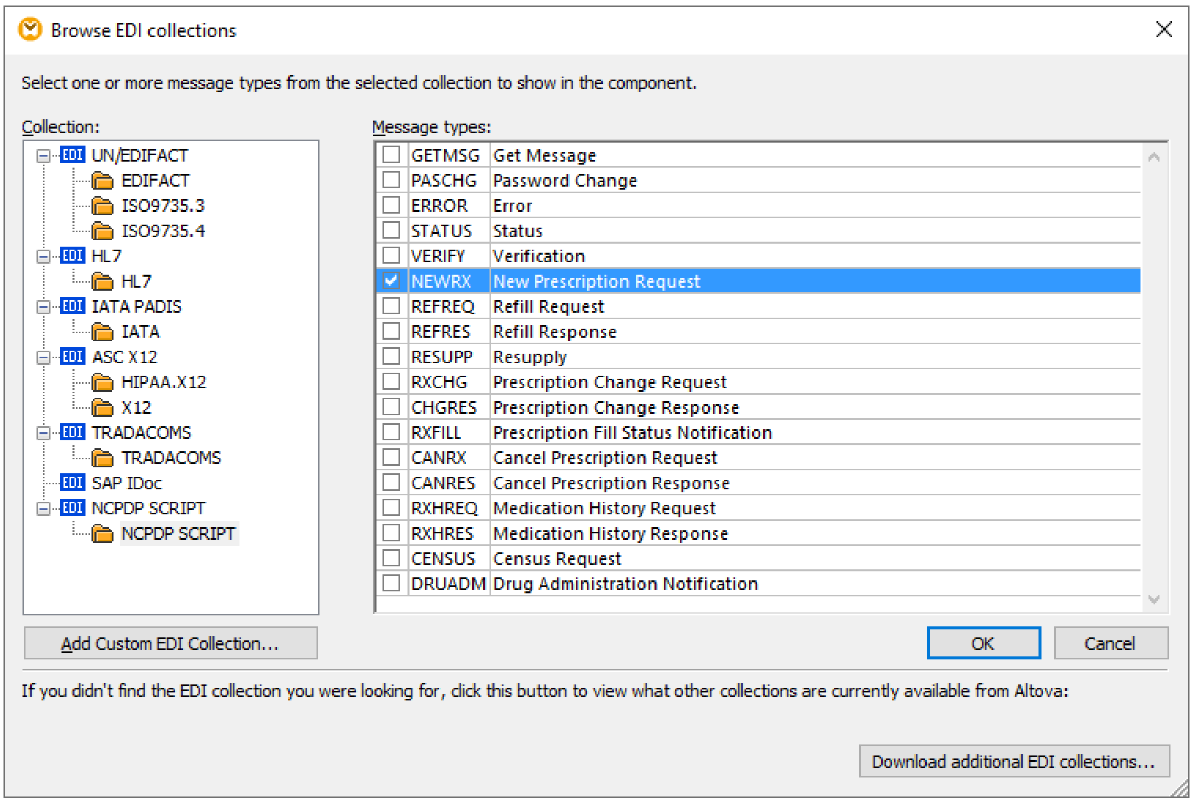
Note: To enable support for NCPDP SCRIPT versions other than the default one, download and install the "NCPDP SCRIPT Configuration Files" package from the Altova download center (https://www.altova.com/mapforce/download/libraries).

NCPDP SCRIPT can use either the XML format or the UN/EDIFACT format. This documentation is applicable to the UN/EDIFACT version of NCPDP SCRIPT. If you have the NCPDP XML Schemas, MapForce will support mapping data to or from SCRIPT implicitly, using the standard XML mapping commands and validation (to add XML files to the mapping, use the menu command **Insert | XML Schema/File**).

Note that MapForce does not support sending or accepting SCRIPT messages as part of data communications between SCRIPT parties. The role of MapForce is to read data in SCRIPT format and map it to other formats, or convert data from other formats into SCRIPT format.

NCPDP SCRIPT files in UN/EDIFACT format can be added to the mapping as follows:

1. On the **Insert** menu, click **EDI**.
2. Select the "NCPDP SCRIPT" collection, and then select the check box next to the required message type(s).



3. Click OK. You are now prompted to supply an instance file.
4. Do one the following
   - If you want to map data from an NCPDP SCRIPT message to some other format, click **Browse** and select the source SCRIPT file. It is also possible to open files from a URL, see Adding Components from a URL 80.
   - If you want to map data to an NCPDP SCRIPT message, or supply the source file at a later time, click **Skip**.

At this stage, the structure of the selected NCPDP SCRIPT message(s) is displayed on the mapping (in MapForce terminology, an "EDI Component"). You can now either draw mapping connections to this component (in order to map data to it), or from this component, in order to map data from it. See also:

- Adding EDI Components to the Mapping 447
- Running EDI Mappings 449
- EDI Component Settings 450
- EDI Component Validation 454

4.3.12.1 NCPDP SCRIPT Validation Rules

With respect to NCPDP SCRIPT messages, MapForce performs the following validation checks:

- `UIB/S001/F0001` should be "UNOA"
- `UIB/S001/F0002` should be "0"
- `UIH/S306/F0329` should contain message type "SCRIPT"
- `UIH/S306/F0316` should contain message version number that is defined in selected configuration files
- `UIH/S306/F0318` should contain message release number that is defined in selected configuration files
- `UIH/S306/F0326` should contain message function (or the message type in MapForce terminology)
- `UIT/F0062` should match to corresponding `UIH/F0062` (message header/trailer) or not present (optional field)
- `UIT/F0074` should contain segments count or not be present (optional field)
- `UIZ/F0036` should contain messages count or not be present (optional field).

See also EDI Component Validation 454.

4.3.12.2 NCPDP SCRIPT Auto-Completion Rules

When generating an NCPDP SCRIPT file, MapForce automatically generates certain fields or values, according to the rules below (referred to as "automatic data completion", or "auto-completion" rules). To disable this behaviour, clear the **Auto-complete missing fields** check box in EDI Component Settings 450.

The following fields will be automatically completed:

Interchange Header

- `UIB/S001/F0001` = controlling agency + syntax version number = "UNOA"
- `UIB/S001/F0002` = syntax version number = "0"
- `UIB/S002/F0004` = "Sender"
- `UIB/S003/F0010` = "Recipient"
- `UIB/S300/F0017` = current date
- `UIB/S300/F0114` = current time

Message Header

- `UIH/S306/F0329` = "SCRIPT", message type
- `UIH/S306/F0316` = "010" for 10.6, message version number from configuration files.
- `UIH/S306/F0318` = "006" for 10.6, message release number from configuration files.
- `UIH/S306/F0326` = message function (or message type from a MapForce perspective)
- `UIH/F0062` = "0", message reference number

Message Trailer

- `UIT/F0062` = `UIH/F0062`

- `UIT/F0074` = number of segments in the message.

Interchange Trailer

- `UIZ/F0036` = message count

4.3.13 SAP IDoc

SAP IDoc (intermediate documents) documents are used to exchange business data between SAP and non-SAP applications. The documents are a form of intermediate data storage which can be exchanged between different systems.

An IDoc is structured as follows:

- **Control Record:** contains control information about the IDoc: sender, receiver, message type, and IDoc type. The control record format is similar for all IDoc types.
- **Data Segment:** contains the actual data of the segment as well as other metadata: header, segment no. and type as well as the fields containing the data.
- **Status Records:** contain info on the current status of the document, i.e. the currently processed stages, and the stages that still need to be processed. The status format is identical for all types of IDoc.

The version number in the port definition defines the systems you are communicating with. The major differences between the versions are the various name lengths used in the various elements and the use of extensions. SAP R3 version 4.X supports long names (as well as extensions) while the previous versions do not.

Port Version 1: Releases 2.1 and 2.2.
Port Version 2: Releases 3.0, 3.1 and R/2 systems.
Port Version 3: Release 4.x (default value)

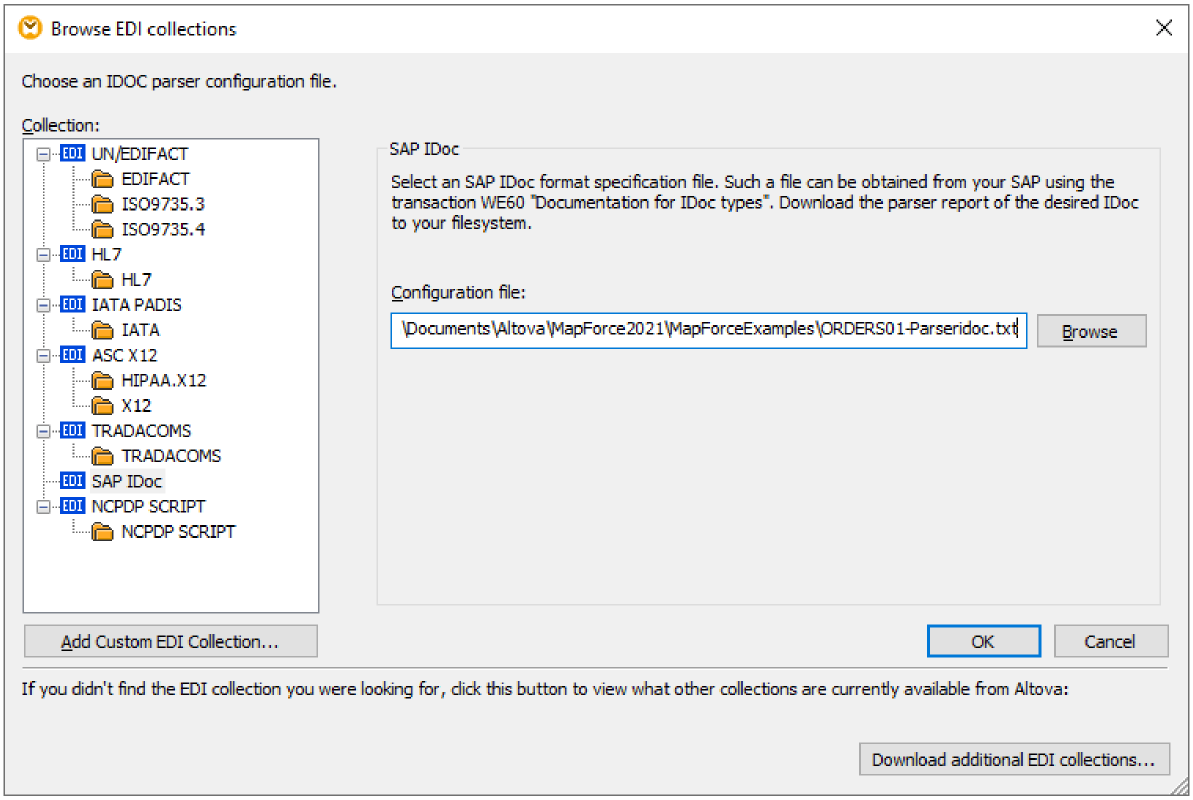
MapForce treats IDoc components as fixed-length files of length 30 char for Message type, 30 for IDoc type, and 27 for segment fields.

For an example which illustrates mapping data from SAP IDoc to XML, see **<Documents>\Altova\MapForce2023\MapForceExamples\IDoc_Order.mfd**.

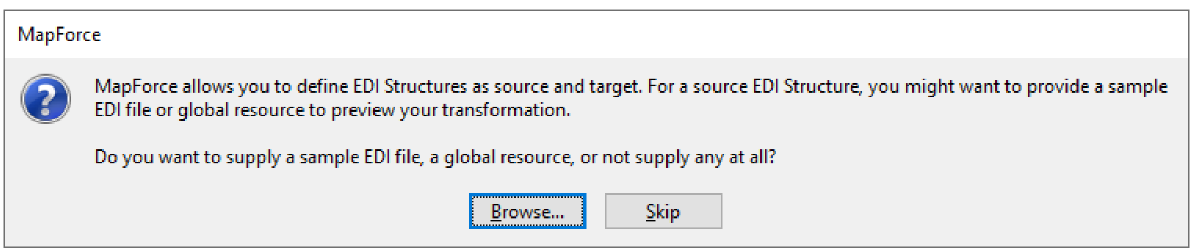
To add an SAP IDoc document as mapping component:

1. Select the menu option **Insert | EDI**. This opens the Browse EDI collections dialog box.
2. Click the **SAP IDoc** entry in the list box.
3. Click **Browse** and select the IDoc parser report (also known as "IDoc definition file").

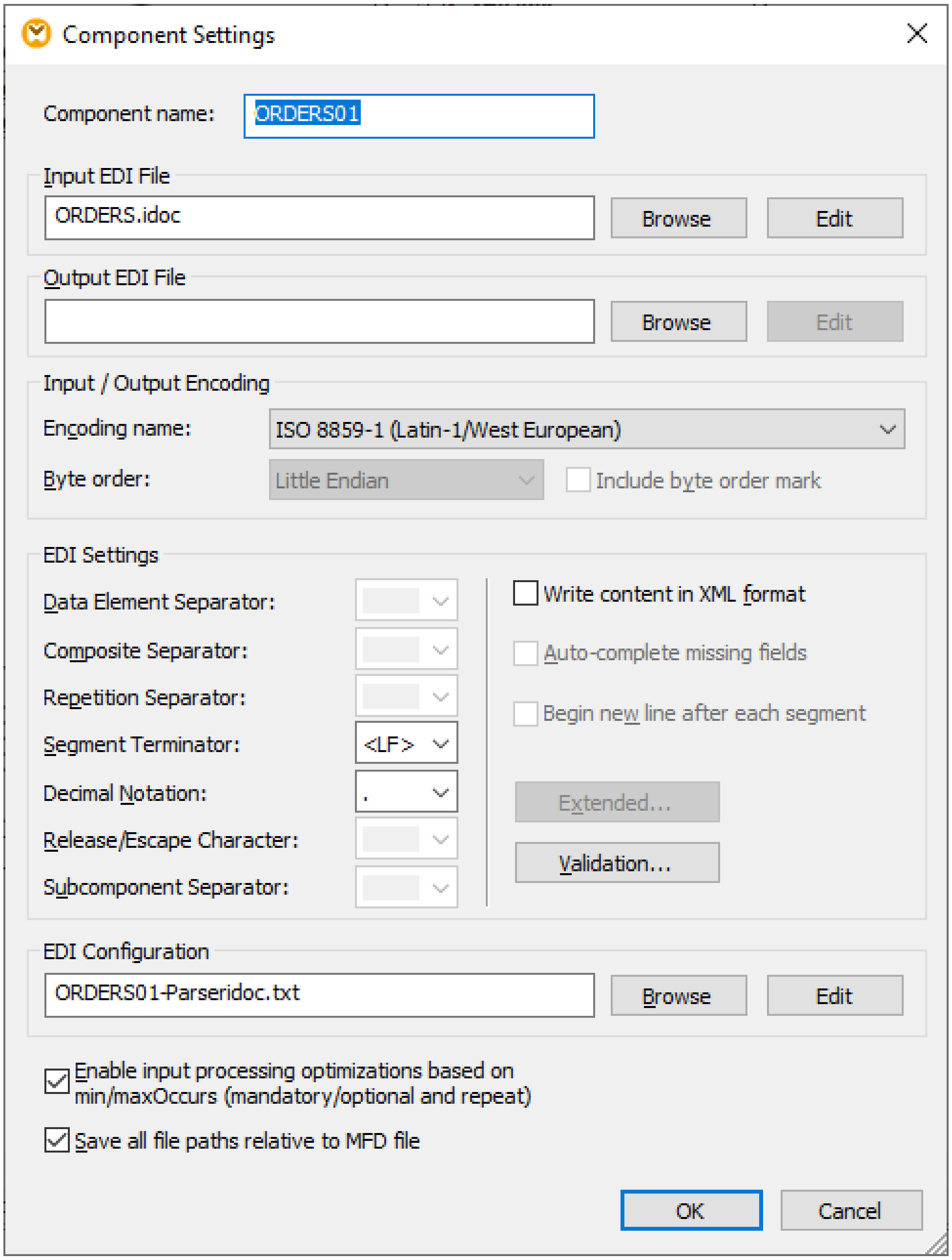
> The IDoc parser report is created from the SAP system using the transaction WE60 ("Documentation for IDoc types"). Note that the file must be exported from SAP in uncompressed format. For information on how to invoke transaction WE60 and generate the parser report, refer to the SAP IDoc documentation. The MapForce Examples folder includes a sample parser report file (see **<Documents>\Altova\MapForce2023\MapForceExamples\ORDERS01-Parseridoc.txt**).



4. Click **OK**. MapForce prompts you to optionally select a sample EDI file.



5. If you would like to map data *from* an IDoc file, click **Browse** and select the IDoc (*.idoc) file that supplies the data. A sample IDoc file is available in the MapForce Examples folder (see **<Documents>\Altova\MapForce2023\MapForceExamples\ORDERS.idoc**). Otherwise, click **Skip**.
6. The Component Settings dialog box is displayed. This enables you to review the settings of the IDoc component before adding it to the mapping. You can change these settings at any time later if required, see EDI Component Settings [450].



7. Click OK to close the Component Settings dialog box and add the IDoc component to the mapping area.

4.3.14 SWIFT MT

SWIFT (Society for Worldwide Interbank Financial Telecommunication) is a global provider of financial messaging services. SWIFT MT (Message Text) is a message type used in SWIFT network. MapForce supports mapping SWIFT MT data to or from any other MapForce-supported format, including XML, CSV, JSON, databases, and others.

MapForce supports the following SWIFT MT releases: 2019 or newer. Note that the default installed version is SWIFT 2022.

Note: Mapping data to or from SWIFT MT requires BUILT-IN[27] as a transformation language. You can preview the mapping in MapForce, save the output files, if any, or choose to execute it with MapForce Server (licensed separately) on a different computer or platform. It is not supported to generate an executable C#, C++, or Java program from mappings that read or write SWIFT MT data.

To support mapping data to or from various EDI-related formats, including SWIFT MT, MapForce uses configuration files, also known as EDI Collections, specific to each release. An EDI collection is a directory that contains machine-readable configuration files, in XML format. By default, the EDI collections are installed in the following folder: `C:\Program Files\Altova\MapForce2023\MapForceEDI`. The name of collections (directories) applicable to SWIFT begins with SWIFT.

Be aware that only the latest supported SWIFT collection is installed by default. You can download and install additional versions from the Altova website.

Limitations

The following limitations apply when mapping data to or from SWIFT MT components. These limitations are applicable regardless of the execution engine running the mapping, be it MapForce or MapForce Server.

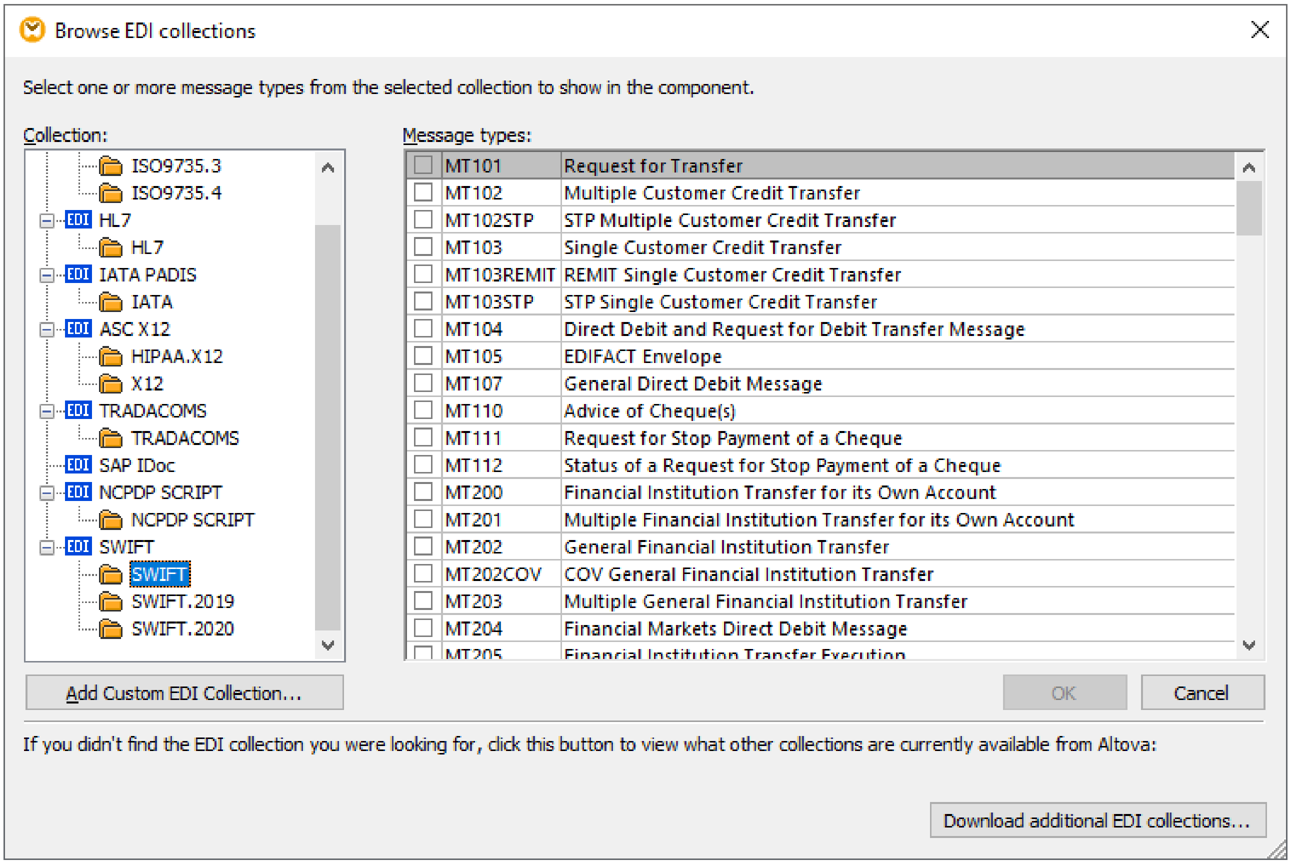
- The following message types are not supported:
  - Non-FIN messages;
  - Category 0 messages (SWIFT system messages);
  - Category n messages (MTx9x).
- To generate a SWIFT MT message, you need to supply the correct Message Type in the Application Header from the mapping, because it is not created automatically from content data. Make sure to map data for the correct message type into the message text body (block 4).
- MapForce provides no validation of SWIFT MT content. Only basic checks required to parse a message are performed.
- No checks are performed with respect to the maximum message length.
- If the input is invalid, the parser stops with an error message, and no further recovery options are available.

Adding SWIFT MT components to the mapping

SWIFT MT messages have a structure that is significantly different from other EDI messages. Nevertheless, you can map SWIFT MT structures in MapForce in a way that is common to all EDI-based formats. Specifically, you can add SWIFT MT components (either source or target) to the mapping using any of the following approaches:

- In the **Insert** menu, select the **EDI** command.
- Click the **Insert EDI** toolbar button.

In the **Browse EDI collections** dialog box, select the required SWIFT release. Then select the check boxes corresponding to the message types that you need to add to the mapping (*see screenshot below*).



Note: By default, only a generic collection called SWIFT is available in the **Browse EDI collections** dialog box illustrated above. This default SWIFT collection is not suitable for production, because its version may change in newer versions of MapForce, just like all other EDI-related standards. For production, it is recommended to download and install the release-specific SWIFT configurations from the Altova website. After installation, these release-specific configurations will appear in the **Browse EDI collections** dialog box in addition to the default SWIFT configuration.

See also Adding EDI Components to the Mapping [447]. For a mapping example, see Example: Convert SWIFT MT to SWIFT MX [488].

SWIFT MT settings

You can change the input files (or the path of the generated file) and other settings of SWIFT MT components in the **Component Settings** dialog box. To open the **Component Settings** dialog box, double-click the title bar of the component in MapForce. Alternatively, right-click the title bar and select **Properties** from the context menu.

For reference to all options available on this dialog box, see EDI Component Settings [450]. Note that the validation settings are disabled (grayed out) for SWIFT MT components because they are not supported, as mentioned above.

Handling SWIFT MT data types

MapForce handles SWIFT MT-specific simple data types as follows:

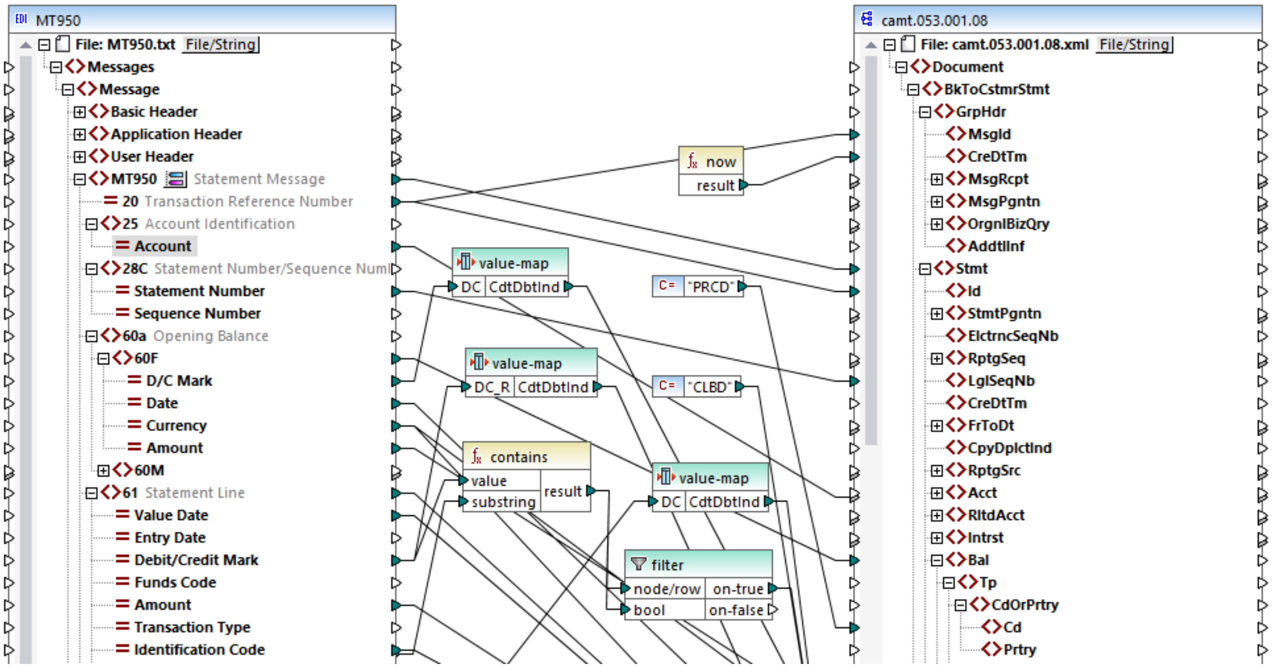
- You can map SWIFT MT date types to an item of type `xs:date` or to a function that takes `xs:date` as an input, and this will be processed correctly, without requiring any special handling. For example, if you map an item of type `swift:date6` to a string, the resulting string is, for example, `20200922`. However, if you map the same type to `xs:date`, the result is **`2020-09-22`** (of type `xs:date`).
- If the target item is a string and you would like the date to be represented as `2020-09-22`, add a variable847 of type `xs:date` to the mapping and map its result to the target string item.
- If you map the `swift:decimal` value **`42,66`** to an item of type `xs:decimal`, the result is **`42.66`**. However, if you map the same type to `xs:string`, the result is **`42,66`**.
- The same behavior is true for writing data to a target component. For example, if you map the `xs:decimal` value **`15`** to an item of type `swift:decimal`, the result is **`15,`** (of type `swift.decimal`).

Quick conversion to XML

As with all other EDI-related formats, you can easily convert SWIFT MT files to XML without drawing any mapping connections. For details, see Generating XML from EDI458.

4.3.14.1 Example: Convert SWIFT MT to SWIFT MX

This example illustrates how to map a file which contains a SWIFT MT message to an XML target file in SWIFT MX format. It is accompanied by a mapping available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\SWIFT_MT950_camt053.mfd**. Because this mapping has a large size, only a fragment of it is illustrated below.



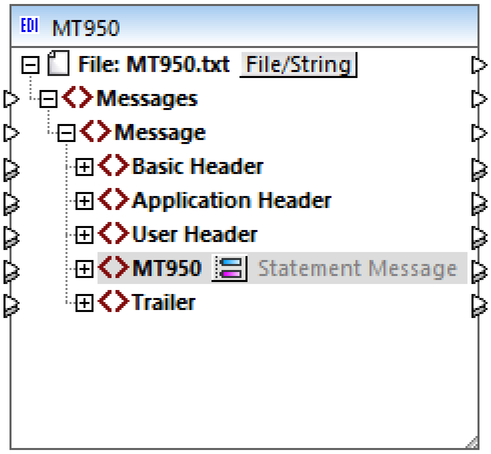
SWIFT_MT950_camt053.mfd

A mapping such as the one illustrated above can be created by following the steps below.

Step 1: Add the source SWIFT MT file to the mapping

1. On the **Insert** menu, click **EDI**.
2. Select the collection "SWIFT", and the message type "MT950 - Statement Message".
3. When prompted to supply an instance file, open the source file available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\MT950.txt**. The Component Settings dialog box appears. For the scope of this example, you can leave the settings unchanged, and click **OK**. You can change these settings at any time later, see EDI Component Settings 450.

The SWIFT MT component is now added to the mapping.

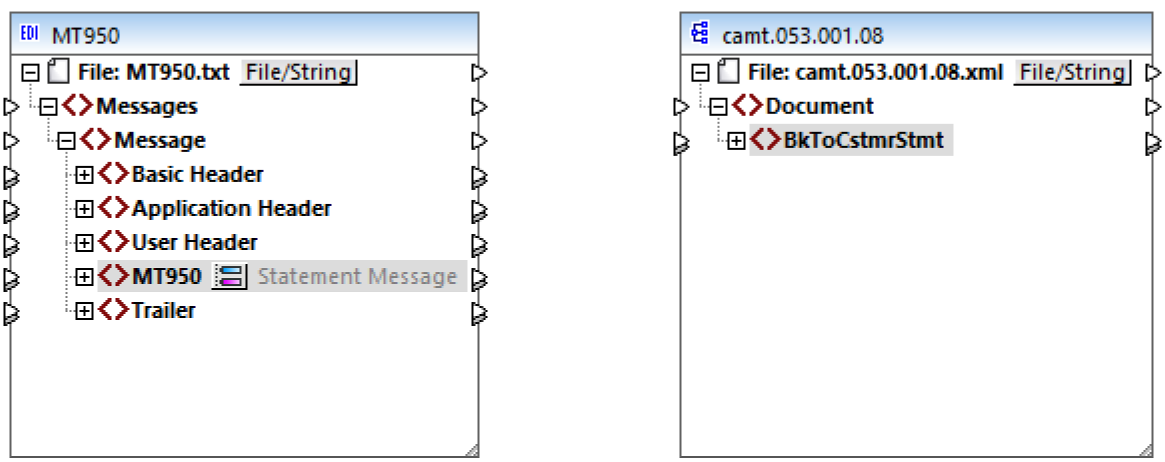


At this stage, if you haven't done that already, you might want to save the mapping to the disk. A good practice is to save the mapping design file in the same directory as its source and target files—this way, you can easily manage all the relative paths to the source or target files. In this example, you could save the mapping to **<Documents>\Altova\MapForce2023\MapForceExamples**, since this is the directory where both the source file and the target schema are located.

Step 2: Add the target schema to the mapping

1. On the **Insert** menu, click the **XML Schema/File**, and open the **camt.053.001.08.xsd** file from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
2. When prompted to supply a sample XML file, click **Skip**.

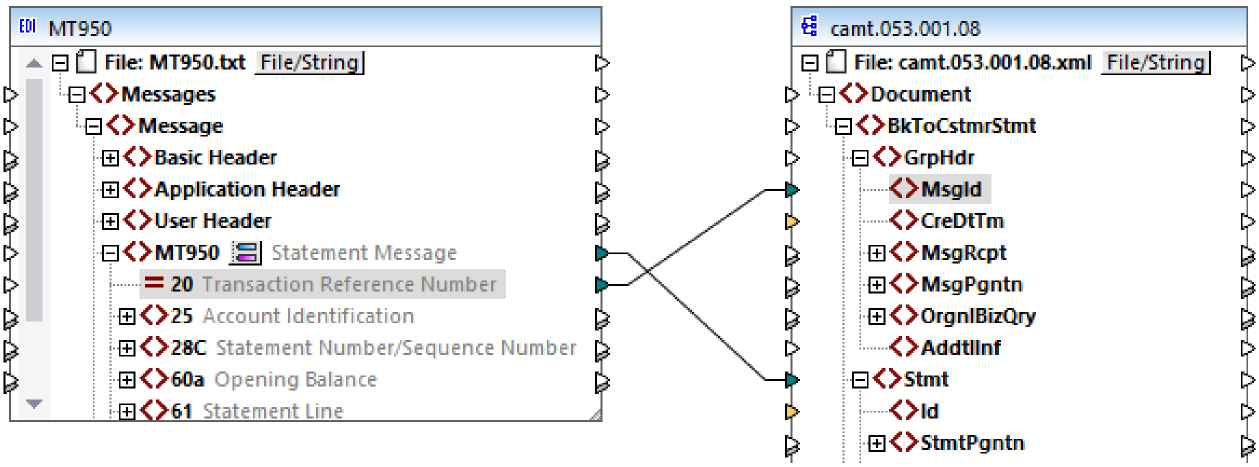
The target XML component is now added to the mapping as well:



Step 3: Add the intermediary components

The remaining mapping consists of drawing connections between the source and the target components. For example, item **MT950** in the source maps to item **Stmt** in the target, while the item **20 (Transaction**

Reference Number) maps to **MsgId**, as illustrated below. If you are new to MapForce, you may need to refer to the Working with Connections[89] and the Tutorials[42] sections for more details about drawing connections.



When at least one connection exists from source to target, you can already preview the mapping output by clicking the **Output** tab. Bear in mind, however, that the resulting XML output will not be valid according to the schema until all mandatory connectors have inputs. The mandatory connectors are highlighted in orange (for example **CreDtTm** in the image above).

Besides straightforward connections from source to target, the complete example mapping (**SWIFT_MT950_camt053.mfd**) contains a few intermediary components, namely:

1. Value-map components (look-up tables). These are useful whenever you need to convert values between two sets (for example, "C" to "CRDT", "D" to "DBIT", and so on). For more information, see Using Value-Maps[884].
2. The functions **`now`** and **`contains`**. The **`now`** function returns the current date and time, while the **`contains`** function returns a Boolean true if the first string argument contains the second string argument. For more information, see Add a Function to the Mapping[905].
3. Various string constants (for example, "PRCD", "CLBD"), see also Add a Constant to the Mapping[905].

To give you just a few examples, the following table describes how some of the connectors from the target component receive input.

Target connector	Description
`GrpHdr`	Although this target item does not receive an input explicitly, its child element **MsgId** does (see below); therefore, the parent **GrpHdr** will be created as well when you run the mapping.
`MsgId`	Receives a straightforward connection from the **Transaction Reference Number** in the source (item **20** under **MT950**).
`CreDtTm`	Receives the direct output of the function now[1096], which is a MapForce built-in function that returns the current date and time.
`Stmt`	Receives a direct connection from **MT950** source item.

Target connector	Description
`Stmt/Bal/Tp/CdOrPrtry/Cd`	Receives input from a text constant, "PRCD".

Running the mapping

To preview the generated file in MapForce, click the **Output** tab. The image below illustrates only a fragment of the resulting XML file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Document xsi:schemaLocation="urn:iso:std:iso:20022:tech:xsd:camt.053.001.08
    <BkToCstmrStmt>
        <GrpHdr>
            <MsgId>MF12345</MsgId>
            <CreDtTm>2020-09-28T14:09:53.277+02:00</CreDtTm>
        </GrpHdr>
        <Stmt>
            <Id>MF12345</Id>
            <LglSeqNb>42</LglSeqNb>
            <Acct>
                <Id>
                    <IBAN>NL58NANO3141592658</IBAN>
                </Id>
            </Acct>
            <Bal>
                <Tp>
                    <CdOrPrtry>
                        <Cd>PRCD</Cd>
                    </CdOrPrtry>
                </Tp>
                <Amt Ccy="EUR">3723495</Amt>
                <CdtDbtInd>CRDT</CdtDbtInd>
```

Mapping output

To save the mapping output to a file, do one of the following:

- On the **Output** menu, click **Save Output File**.
- Click the **Save generated output** toolbar button.

Automation with MapForce Server

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine. Note that running mappings on a server typically requires a few extra steps as outlined in [Preparing Mappings for Server Execution]1372. Here are the steps required to run this particular mapping on a server:

1. Right-click an empty area on the mapping and select **Mapping Settings** from the context menu.
2. Clear the **Make paths absolute in generated code** check box and save the mapping in the same directory.
3. On the **File** menu, click **Compile to MapForce Server Execution File** and save the .mfx file in the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
4. Copy the generated .mfx file to a directory on the server machine (let's call it the "working directory").

5. Copy the file **<Documents>\Altova\MapForce2023\MapForceExamples\MT950.txt** to the working directory on the server machine.
6. Open a terminal and change the current directory to the working directory.
7. Run MapForce Server with the command below. Adjust **mapforceserver_exec** to be the path to the MapForce Server executable as applicable for your operating system.

```
mapforceserver_exec run SWIFT_MT950_camt053.mfx
```

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see Compiling Mappings to MapForce Server Execution Files 1378.

4.3.15 SWIFT MX / ISO 20022

The SWIFT MX / ISO 20022 standards are based on XML. Therefore, you can add such components to the mapping in the same way as any other XML schema 122. There are no other special requirements to map such data.

The standard SWIFT MX schemas can be downloaded at https://www2.swift.com/knowledgecentre/products/Standards%20MX.

The standard ISO 20022 schemas can be downloaded at https://www.iso20022.org/iso-20022-message-definitions.

There are many non-standard schema variations of the ones mentioned above, such as country-specific variants. You can use any XML schema as mapping component in MapForce as long as it is a valid XML schema.

For a mapping example, see Example: Convert SWIFT MT to SWIFT MX 488.

4.3.16 TRADACOMS

TRADACOMS (Trading Data Communications) is a UK-specific Electronic Data Interchange standard used in the retail business. It was introduced in 1982 as the first EDI standard for UK trade and industry. Although it has many similarities with the UN/EDIFACT standard, TRADACOMS is one of the precursors of EDIFACT and uses different structures within messages.

MapForce implements the base TRADACOMS specification as laid out in the "TRADACOMS Manual of Standards for Electronic Data Interchange", published in January 1993 by the Article Numbering Association (ANA) UK, now known as GS1 UK (https://www.gs1uk.org). For other TRADACOMS versions, MapForce can be customized to process new message types, data elements, and code values, by means of configuration files.

You can work with the TRADACOMS format in MapForce in the same way as with other supported EDI formats, as follows:

- You can map TRADACOMS files with any other data formats supported by MapForce, in both directions (either as data source or data target).
- Mapping data to or from TRADACOMS format is available in the BUILT-IN language (used when previewing mappings, or in MapForce Server and FlowForce Server execution) and in code generation

languages (C++, C#, Java). When you deploy the mapping to a FlowForce Server running on a different machine, the deployed package includes automatically the configurations of selected TRADACOMS message types and all code lists that are used by their data elements. Likewise, in the generated code, MapForce generates classes for the configuration groups, segments, and data elements.

- You can flexibly define which validation events should stop the data conversion, which ones should result in rejected (or accepted) records, and which ones should be ignored.
- You can enable or disable data auto-completion. When "Automatic data completion" (or "auto-completion") is enabled, MapForce fills some of the values automatically when generating TRADACOMS files.
- You can customize the message types, data elements, and code lists by means of configuration files, either globally or locally (see Configuration Files 503).
- You can set the encoding of parsed or generated files.

The following TRADACOMS file types and message types are available by default in MapForce:

File Type	Message Type	Version
THE PRODUCT INFORMATION FILE	PROHDR (Product File Header)	8
	PROINF (Product Details)	8
	PROTLR (Product File Trailer)	8
THE PRICE INFORMATION FILE	PRIHDR (Price File Header)	8
	PRIINF (Price Details)	8
	PRITLR (Price File Trailer)	8
THE CUSTOMER INFORMATION FILE	CUSHDR (Customer Information Header)	8
	CUSINF (Customer Information Details)	8
	CUSTLR (Customer Information Trailer)	8
THE ORDER FILE	ORDHDR (Order File Header)	9
	ORDERS (Order Details)	9
	ORDTLR (Order File Trailer)	9
THE PICKING INSTRUCTIONS FILE	PICHDR (Picking Instructions File Header)	4
	PICKER (Picking Instructions File Details)	4
	PICTLR (Picking Instructions File Trailer)	4
THE DELIVERY NOTIFICATION FILE	DELHDR (Delivery File Header)	9
	DELIVR (Delivery Details)	9
	DELTLR (Delivery File Trailer)	9
DELIVERY CONFIRMATION FILE	DLCHDR (Delivery Confirmation Header)	5

File Type	Message Type	Version
	DLCDET (Delivery Confirmation Details)	5
	DLCTLR (Delivery Confirmation Trailer)	5
THE INVOICE FILE	INVFIL (Invoice File Header)	9
	INVOIC (Invoice Details)	9
	VATTLR (VAT Trailer)	9
	INVTLR (Invoice File Trailer)	9
THE CREDIT NOTE FILE	CREHDR (Credit Note File Header)	9
	CREDIT (Credit Note Details)	9
	VATTLR (File VAT Trailer)	9
	CRETLR (Credit Note File Trailer)	9
STATEMENT/REMITTANCE FILE	SRMHDR (Statement/Remittance Details File Header)	9
	SRMINF (Statement/Remittance Line Details)	9
	SRMTLR (Statement/Remittance Details File Trailer)	9
UPLIFT INSTRUCTION FILE	UPLHDR (Uplift File Header)	4
	UPLIFT (Uplift File Details)	4
	UPLTLR (Uplift File Trailer)	4
UPLIFT CONFIRMATION FILE	UCNHDR (Uplift Confirmation Header)	3
	UCNDET (Uplift Confirmation Details)	3
	UCNTLR (Uplift Confirmation Trailer)	3
THE STOCK SNAPSHOT FILE	SNPHDR (Stock Snapshot Header)	3
	SNPSTS (Stock Snapshot Details)	3
	SNPTLR (Stock Snapshot Trailer)	3
THE STOCK ADJUSTMENT FILE	SADHDR (Stock Adjustment Header)	3
	SADDET (Stock Adjustment Details)	3
	SADTLR (Stock Adjustment Trailer)	3
AVAILABILITY REPORT FILE	AVLHDR (Availability File Header)	4
	AVLDET (Availability Report Details)	4

File Type	**Message Type**	**Version**
	AVLTLR (Availability File Trailer)	4
GENERAL COMMUNICATIONS FILE	GENHDR (General Communications File Header)	3
	GENRAL (General Communications Text)	3
	GENTLR (General Communications Trailer)	3
COMPLEX ORDER FILE	CORHDR (Complex Order File Header)	6
	CORDER (Complex Order Details)	6
	CORTLR (Complex Order Trailer)	6
THE ACKNOWLEDGEMENT OF ORDER FILE	ACKHDR (Acknowledgement File Header)	4
	ACKMNT (Acknowledgement Details)	4
	ACKTLR (Acknowledgement Trailer)	4
PRODUCT PLANNING REPORT FILE	PPRHDR (Product Planning Report Header)	2
	PPRDET (Product Planning Report Details)	2
	PPRTLR (Product Planning Report Trailer)	2
THE PAYMENT ORDER FILE	PAYHDR (Payment Order File Header)	3
	PAYINF (Payment Order Line Details)	3
	PAYTLR (Payment Order File Trailer)	3
THE DEBIT ADVICE FILE	DRAHDR (Debit Advice File Header)	3
	DRAINF (Debit Advice Line Details)	3
	DRATLR (Debit Advice File Trailer)	3
THE CREDIT ADVICE FILE	CRAHDR (Credit Advice File Header)	3
	CRAINF (Credit Advice Line Details)	3
	CRATLR (Credit Advice File Trailer)	3
THE EXCEPTION CONDITION FILE	EXCHDR (Exception Condition File Header)	3
	EXCINF (Exception Condition Line Details)	3
	EXCTLR (Exception Condition File Trailer)	3
LOCATION PLANNING REPORT FILE	LPRHDR (Location Planning Report Header)	2
	LPRDET (Location Planning Report Details)	2

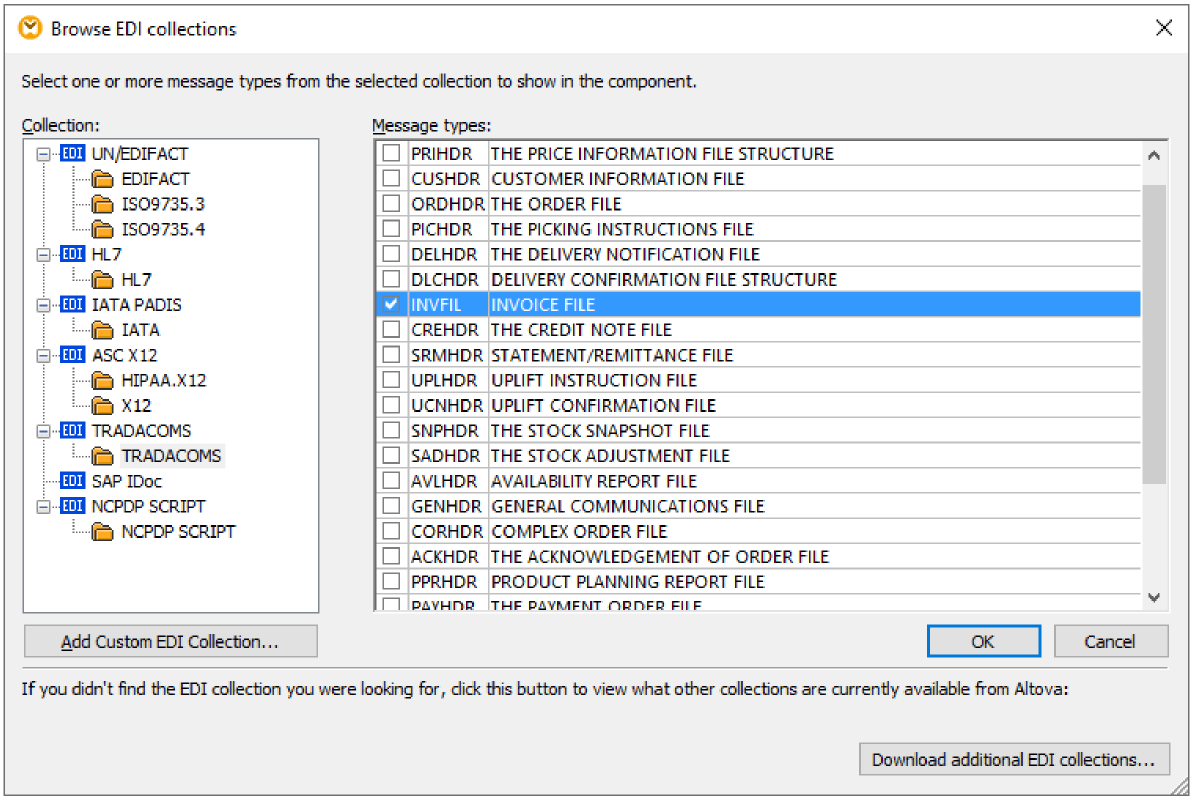
File Type	Message Type	Version
	LPRTLR (Location Planning Report Trailer)	2
THE UTILITY BILL FILE	UTLHDR (Utility Bill File Header)	3
	UTLBIL (Utility Bill File Details)	3
	UVATLR (Utility Bill VAT Trailer)	3
	UTLTLR (Utility Bill File Trailer)	3

Note: The RSGRSG (Reconciliation Message) version 2 is also supported. This message type may occur in all file types.

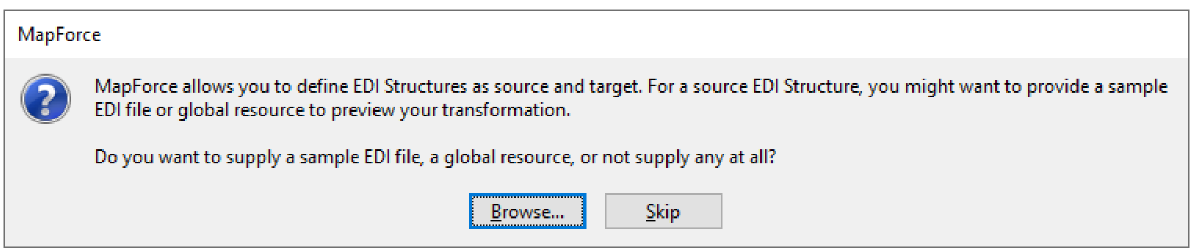
4.3.16.1 Adding TRADACOMS Files as Mapping Components

To add a TRADACOMS file as a data mapping source:

1. Do one of the following:
   - On the **Insert** menu, click **EDI**.
   - Click the **Insert EDI file** () toolbar button.
2. On the Browse EDI Collections dialog box, click **TRADACOMS**, and select the message type(s) to be included in the message.



3. Click **OK**. You are now prompted to supply an instance file.



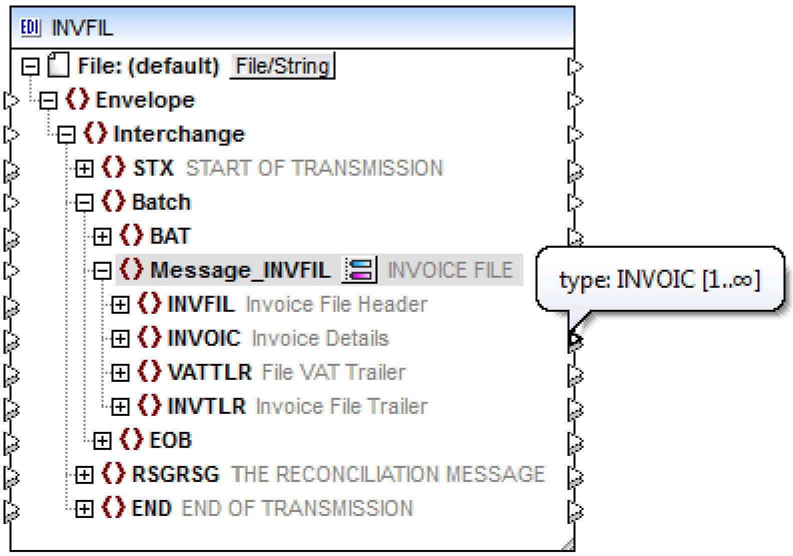
4. If you want to specify the source TRADACOMS file later, click **Skip**. Otherwise, click **Browse** and choose how you would like to open the TRADACOMS file:
   - o If you want to open the file from your local drive or network, browse for the file, and then click **Open**.
   - o If you want to open the file from a URL, click **Switch to URL** (for more information, see Adding Components from a URL 80)
   - o If you have previously defined the file as a Global Resource, and would like to open it from Global Resources, click **Global Resources** (see Altova Global Resources 1397).
5. On the Component Settings dialog box, set or change the settings if required (see EDI Component Settings 450).

To add a TRADACOMS file as a data mapping target:

1. Follow the steps 1 and 2 above.
2. When prompted to supply an instance file, click **Skip**.

4.3.16.2 The TRADACOMS Component in MapForce

In MapForce, the TRADACOMS component replicates the structure of a generic TRADACOMS transmission (with optional batching feature), while retaining all features common to other MapForce EDI-related components.



A sample TRADACOMS component in MapForce

The topmost item of the component, **File**, displays the name of the TRADACOMS interchange file being processed or generated by the component. "Default" indicates that no file has been assigned yet. The **File/String** (File/String) button displays advanced features common to all file-based MapForce components. It provides, among other options, the ability to read or process multiple files dynamically (for more information, see Processing Multiple Input or Output Files Dynamically 1313).

The "Envelope" and "Interchange" structures are generic for all MapForce EDI-related components. Where applicable, they provide the ability to process multiple interchanges within the same MapForce component.

The `STX` and `END` structures are specific to the TRADACOMS format. They denote the "Start of Transmission" and "End of Transmission" segments, respectively.

The `BAT` and `EOB` structures are specific to the TRADACOMS format. They denote the "Start of Batch" and "End of Batch" segments, respectively. Such segments are applicable if you are reading data from a TRADACOMS interchange file with batches, or if you want to generate a file with batches. Since there may exist multiple batches in the same transmission, a **Batch** sequence is available as parent of the `BAT` sequence.

The **Message_*Code*** structure (**Message_INVFIL**, in the sample above) corresponds to the TRADACOMS "file". The **Select EDI Message Types** button ( ) opens a dialog box where you can change the type of the file (for example, "Invoice File", "Orders File", and so on). By virtue of the existing MapForce EDI functionality, you can also include multiple file types in the same component. Note, however, that the TRADACOMS specification recommends one file and one type of file in each interchange transmission.

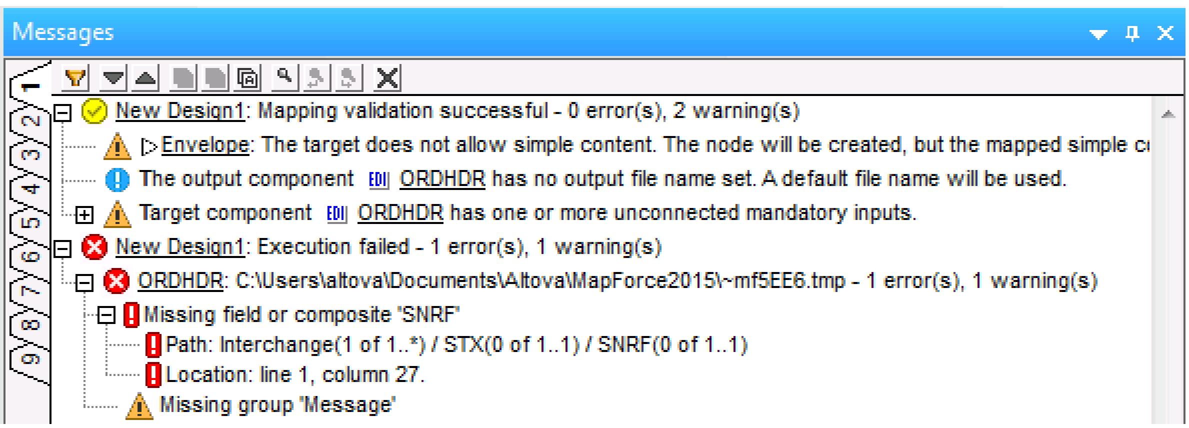
Any TRADACOMS file ("Invoice File", in this example) contains a Header Message, one or more Details Messages, and a Trailer Message. In the component above, these are the `INVFIL` (Invoice File Header), `INVTLR` (Invoice File Trailer), the `VATTLR` (File VAT Trailer) and multiple `INVOIC` (Invoice Details) messages.

As with any other MapForce component, the input and output connectors (small triangles) displayed on each side of the component provide the ability to map each individual data element or segment to or from other data types or formats supported by MapForce. Likewise, when you move the mouse over any item on the component, you can view additional information about it (such as the minimum and maximum allowed occurrences), provided that tips are enabled from the **View | Show Tips** menu.

Finally, you can change these and other settings by double-clicking the component head and opening the Component Settings dialog box (see EDI Component Settings 450). Note that this dialog box is generic for all EDI-related component types and, thus, some of the options might not be applicable to TRADACOMS.

4.3.16.3 Validation and Automatic Data Completion

When you run a mapping that reads data from or writes data to a TRADACOMS structure, MapForce performs structural data validation checks according to the TRADACOMS specification, and displays any validation errors in the Messages window.



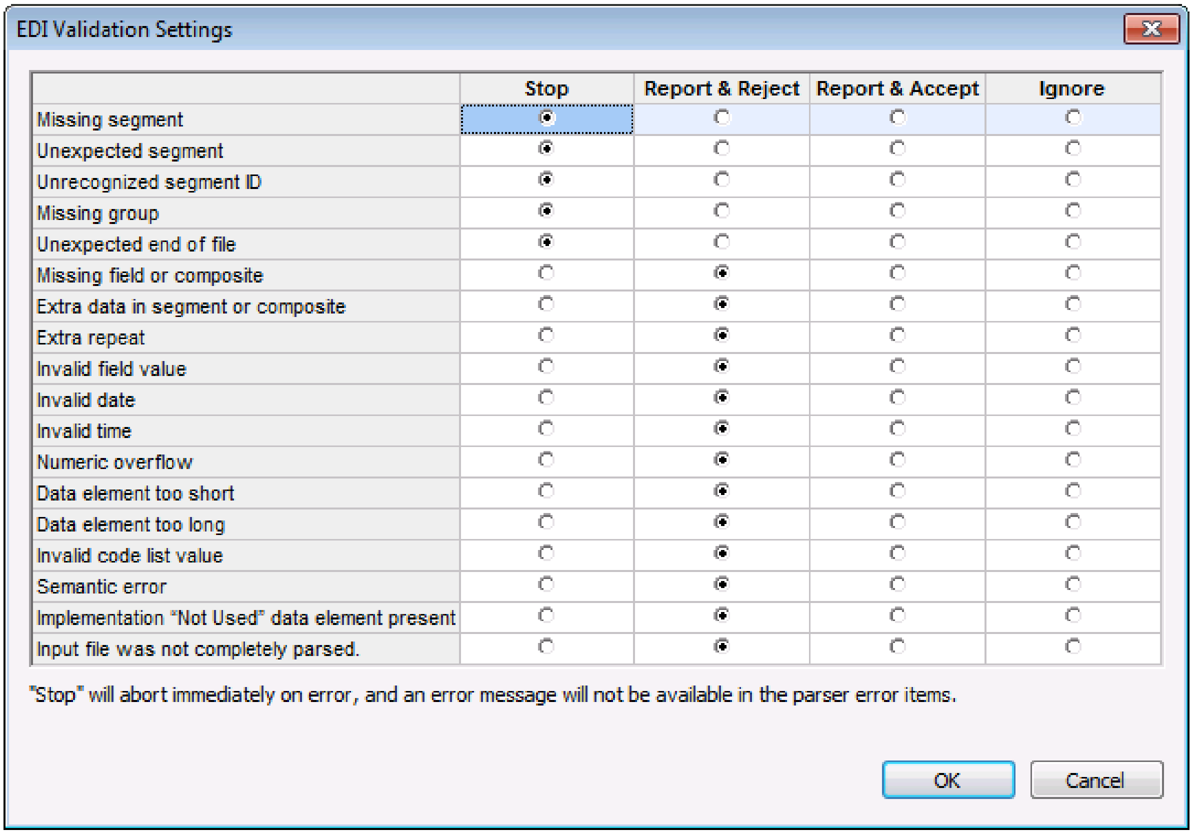
Messages window with validation errors

As shown in the sample above, the validation messages specific to TRADACOMS (information, warnings, and errors) are displayed in addition to the generic validation messages common to any MapForce mapping. For more information about MapForce validation in general, see Validating Mappings 105.

The following factors determine how MapForce validates the parsed or generated TRADACOMS files:

- The validation constraints defined in the configuration files available in the MapForce installation folder (subfolder **MapForceEDI\TRADACOMS**). These configuration files supply, on one hand, the default validation rules of the TRADACOMS specification. On the other hand, they provide the means to adapt the TRADACOMS format to custom requirements. In particular, it is possible to modify the data elements, segments, or code values defined in the configuration files, and thus influence both the outcome of validation and the mapping execution. For more information about the configuration files, see Configuration Files 503.
- The validation logic built into MapForce. This includes MapForce internal data integrity checks that may not be enforced by means of configuration files.

- Any custom validation settings that you have defined from the MapForce graphical user interface (see EDI component validation 454). To view or change the current EDI validation settings of any EDI component, including TRADACOMS, double-click the header of the component, and then click **Validation** on the Component Settings dialog box.



When writing to a TRADACOMS structure, MapForce automatically fills in the contents of those data elements for which the value can be calculated or is predefined. This is referred to as "automatic completion" (or "auto-completion"). To disable this behavior, clear the **Auto-complete missing fields** check box from the Component Settings dialog box (see EDI Component Settings 450).

The following TRADACOMS validation rules cause MapForce to either raise validation errors (during file parsing or generation) or to auto-complete missing fields (during file generation):

1. The segments `STX` (Start of Transmission) and `END` (End of Transmission) must exist.
2. If `STDS-1` has the value 'ANAA' then a Reconciliation Message (`RSGRSG`) must exist before the end of transmission (`END`). Otherwise, no Reconciliation Message (`RSGRSG`) must be present.
3. If `STDS-1` has the value 'ANAA' then:
   a. The value of `RSGA` in the Reconciliation Message must be equal to the value of `SNRF` in the `STX segment`
   b. The value of `RSGB` in the Reconciliation Message must be equal to the value of `UNTO-1` in the `STX segment.`
4. `TRDT-1` must contain the date (YYMMDD) and `TRDT-2` must contain the time (HHMMSS) of transmission (current date and time).
5. If the Batch Header (`BAT`) is present then the Batch Trailer (`EOB`) must also be present, and the number of messages in the batch must be available in the `NOLI` (Number of Messages in Batch) data element.

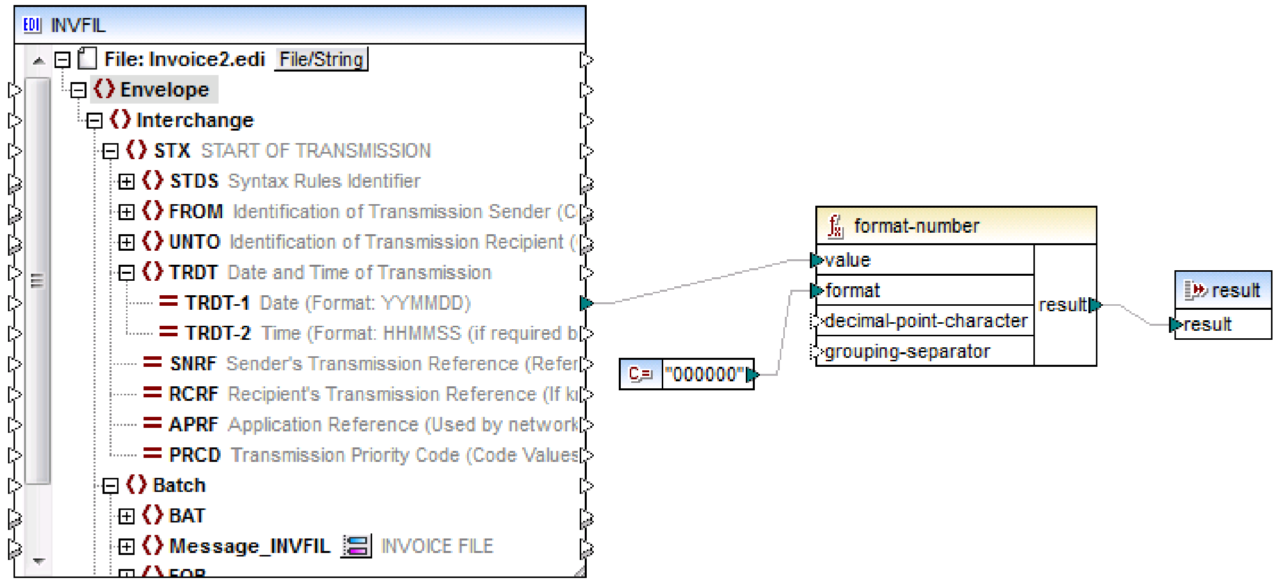
6. The MSRF (Message Reference) data element in the Message Header (MHD) must contain the consecutive count of messages within the transmission, starting with 1.
7. The NOSG (Number of Segments in Message) data element in the Message Trailer (MTR) must contain the number of segments, including MHD and MTR.
8. When present, the Reconciliation Message (RSGRSG) must consist of one segment (RSG), apart from the Message Header and the Message Trailer.
9. The NMST (Number of Messages in Transmission) data element in the END segment must contain the number of messages in interchange (count of MHD segments).
10. In general, when reading a TRADACOMS structure, MapForce expects that the interchange environment is of type "computer to computer" (or, in TRADACOMS terminology, "intelligent terminal to intelligent terminal"). Therefore, a segment such as **`MHD = 12 + ORDHDR :3`** would trigger a validation error, since it contains extra leading and trailing spaces.
11. String data must be in upper case. When generating TRADACOMS output, MapForce converts string data to upper case.

Additionally, as already mentioned, any validation rules defined in the configuration files will also affect TRADACOMS parsing and generation.

4.3.16.4 Preserving Leading Zeros During Conversion

In the TRADACOMS specification, the data type of the TRDT-1 and TRDT-2 (Date and Time of Transmission) data elements is defined as "decimal". This means that, by default, when you map data from the TRDT-1 or TRDT-2 fields, any leading zeros in these fields would be trimmed during conversion, and thus produce undesired results. This may also be the case of any other items which are defined as "decimal", but store values which are meant to be treated as string.

You can instruct MapForce to treat numeric fields as string (and thus preserve the leading zeros) by applying the **`core | format-number`**993 MapForce function. In the following example, the value of the TRDT-1 item in the source file is "020312". The normal output of this item would be "20312" (as a result of conversion to decimal), which is not the desired result. Therefore, to keep the leading zero, the **`format-number`** function has been added to the mapping.



Preserving leading zeros with the format-number function

As illustrated above, the function has the following two input arguments:

1. The value to be formatted (in this case, "020312").
2. The format mask "000000".

To test the output of the function, this example uses a simple output component (see Simple Output 843). When you click the **Output** button, the output of the mapping (which is the same as the result of the function) is "020312", as intended.

4.3.16.5 Configuration Files

By default, the MapForce configuration files and the underlying MapForce processing logic reflects the base version of the TRADACOMS standard (931). Therefore, if you use this TRADACOMS version, customization of the TRADACOMS configuration files is not necessary. However, if you want to make changes to the existing configuration for any reason, it is possible to do so. Note that any changes made to the standard TRADACOMS configuration files will take immediate effect and will affect the way MapForce parses or writes TRADACOMS structures.

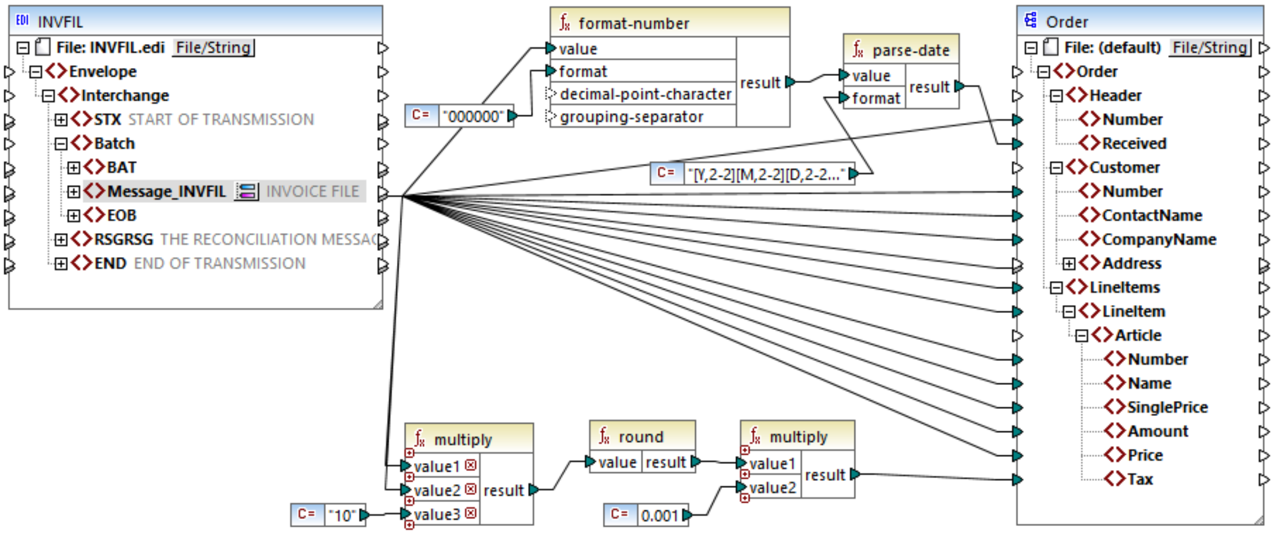
The MapForce configuration files applicable to TRADACOMS are stored in the following directory: **C:\Program Files\Altova\MapForce2023\MapForceEDI\TRADACOMS**. Should you need to make changes to the TRADACOMS configuration files, you can start by creating a sibling copy of the TRADACOMS directory, for example "TRADACOMS1", edit the directory permissions not to be "read-only", and then edit the files in it according to your custom requirements. All configuration files adhere to the XML syntax. Therefore, it is recommended that you use an XML editor to modify the files.

File	Purpose
EDI.Collection	The **EDI.Collection** file contains a list of all messages in the current TRADACOMS configuration directory (for example, **C:\Program Files\Altova\MapForce2023\MapForceEDI\TRADACOMS**). Any messages listed in the collection file become available for selection in the "Browse EDI Collections" dialog box when you add a TRADACOMS component to the mapping (**Insert | EDI**). Therefore, if you want to suppress any message type on this dialog box, comment out or remove their respective XML element from the EDI.Collection file. For example, commenting out the following line makes the `LPRHDR` message type unavailable: `<!--<Message Type="LPRHDR" File="LPRHDR.Config" Description="LOCATION PLANNING REPORT FILE"/>-->`
Envelope.Config	Defines the structure of the TRADACOMS component in MapForce (that is, elements that are surrounded by the **Envelope** sequence). This file is validated against a schema handled by MapForce internally, and it should not be edited.
Tradacoms.Codelist	This file defines the TRADACOMS codes and the values that they may contain, and is used for validation of input/output TRADACOMS files. You can add custom code lists to this file, if required.
Tradacoms.Segment	This file defines the Segment, Composite and Field names of the TRADACOMS files, and is used when parsing a TRADACOMS file. Changes made to this file are global customizations, and apply to all segments and messages.
[Message].Config	Defines the structure of a single TRADACOMS message. Changes made to this file are treated as local (inline) customizations.

For more information, see Customizing EDI Structure 519 and Customizing EDI Validation 552.

4.3.16.6 Example: Converting a TRADACOMS Invoice to XML

This example provides instructions on how to create a mapping design that converts data from a TRADACOMS invoice file to XML format. The mapping design file created in this example is also available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\TRADACOMS_Invoice.mfd**.



TRADACOMS_Invoice.mfd

The mapping accomplishes the following goals:

1. Convert a source TRADACOMS file (**INVFIL.edi**) to an XML file which is valid against an existing schema (**Order.xsd**). Both the source TRADACOMS file and the schema file (**Order.xsd**) are available in the MapForce Examples folder (**<Documents>\Altova\MapForce2023\MapForceExamples**).
2. The date in the TRADACOMS file has `decimal` format so it must be converted to `xs:dateTime` format in the target XML file.
3. For each line item in the purchase order, the tax amount must be calculated as a decimal value, rounded to three decimal places (thousandths). For the scope of this example, we will assume that the tax is calculated according to the following formula:

```
Tax = round(LEXC * VATP * 10) * 0.001
```

Where **round** rounds the value to the nearest integer.

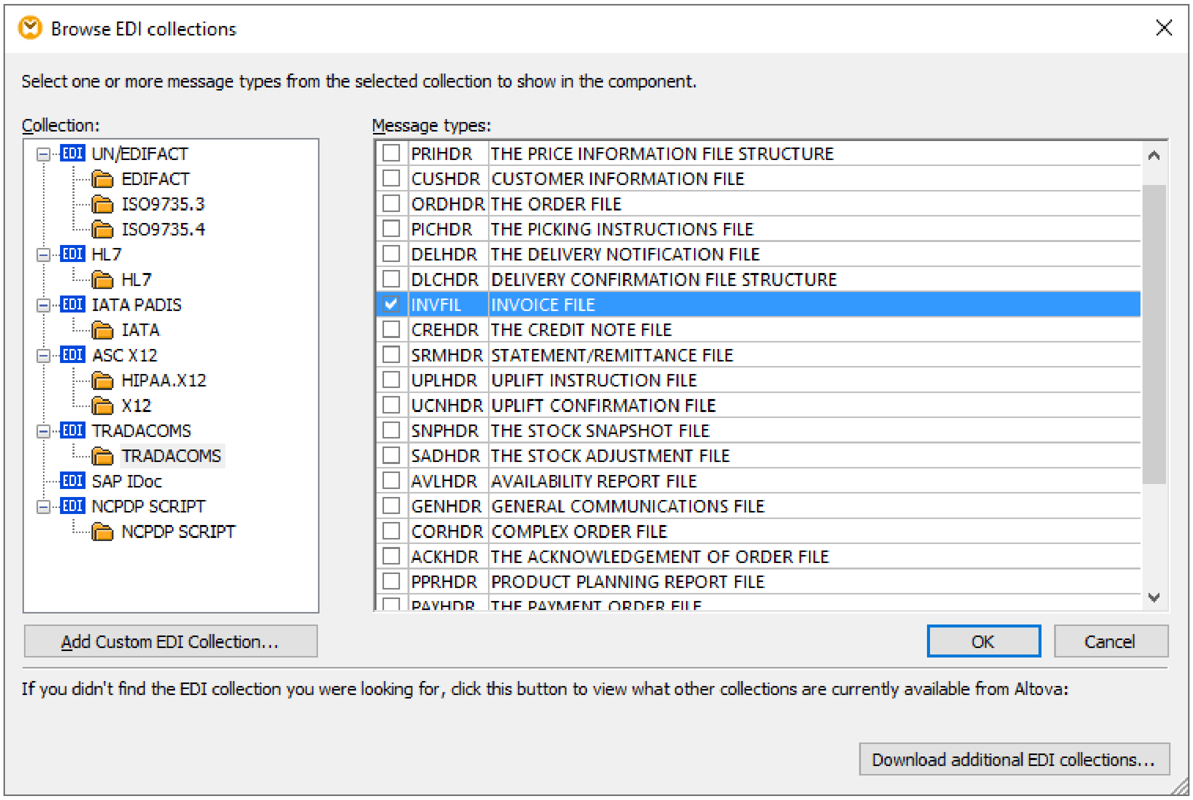
To accomplish the goals, take the following steps:

1. Add the source and target components to the mapping area and draw the mapping connections between them.
2. Use date functions to convert the date to the required format.
3. Use math functions to calculate the tax value.

Step 1: Add the components to the mapping area

To add the source TRADACOMS component:

1. On the **Insert** menu, click **EDI**.
2. Click the TRADACOMS collection folder, and, under Message Types, select the **INVOICE FILE**.



3. When prompted to supply a sample file, browse for the **INVFIL.edi** available in the MapForce Examples folder (**<Documents>\Altova\MapForce2023\MapForceExamples**).
4. When prompted to enter the component settings, leave the default values as is, and click OK.

To add the target XML component:

1. On the **Insert** menu, click **XML Schema/File**.
2. When prompted to supply a sample file, click **Skip**. (Because this is a target component, the file will be generated, so there is no need to browse for an existing one.)
3. When prompted to select a root element, select "Order".

In this example, the data will be mapped from the source TRADACOMS file to the target XML file as follows:

Source (TRADACOMS)	Target (XML)
INVN	/Order/Header/Number
IVDT*	/Order/Header/Received
CIDN-2	/Order/Customer/Number
CADD-1	/Order/Customer/ContactName
CNAM	/Order/Customer/CompanyName
CADD-2	/Order/Customer/Address/Street

Source (TRADACOMS)	Target (XML)
CADD-3	/Order/Customer/Address/City
CADD-5	/Order/Customer/Address/ZIP
CADD-4	/Order/Customer/Address/State
Group_IRF**	/Order/LineItems
Group_ILD**	/Order/LineItems/LineItem
SPRO-1	/Order/LineItems/LineItem/Article/Number
TDES-1	/Order/LineItems/LineItem/Article/Name
AUCT-1	/Order/LineItems/LineItem/Article/SinglePrice
QTYI-1	/Order/LineItems/LineItem/Article/Amount
LEXC	/Order/LineItems/LineItem/Article/Price
LEXC*, VATP*	/Order/LineItems/LineItem/Article/Tax

* Source data will be processed with functions before being written to the target
** These fields enable iteration through a group of fields of the same type

Using the table above as reference, you can now draw mapping connections from all of the above items in the source component to their destination in the target component. For now, you can omit those fields that require calculated values (that is, the ones marked with * in the table); these will be handled in the next steps. For general information about drawing connections in MapForce, see Working with Connections 89.

Tip: To search for a field by its name inside the MapForce component, press **Ctrl + F**.

Step 2: Add the date conversion functions

To convert the value of the `IVDT` item from a decimal value to `xs:dateTime` type, the mapping uses the **`format-number`** function from the **`core | conversion functions`** 988 library. For a worked example, see Preserving Leading Zeros During Conversion 502.

The role of the second function, **`parse-date`**, is to convert the YYMMDD string value returned by the **`format-number`** to `xs:dateTime` type. It has two input arguments: (a) the value to be formatted and (b) the format mask. The format mask essentially instructs MapForce to treat the year, month, and date as having a width of exactly two characters each.

To add the functions above to the mapping, drag them from the Libraries window to the mapping area. To supply an argument to a function, insert a constant (the menu command is **Insert | Constant**). For general information about working with functions in MapForce, see Functions 904.

Step 3: Add the tax calculation functions

The functions used to calculate tax according to the formula are **`multiply`** and **`round`**, available in the **`core | math functions`** 1016 library. The multiply function is extendable (that is, it can take a variable number of arguments). In this example, it takes `LEXC`, `VATP`, and the integer 10 as arguments. The first two values are connected from the source TRADACOMS component, while the integer is supplied by a constant. The result of the **`multiply`** function is then rounded to the nearest integer using the **`round`** function. Finally, the result of the **`round`** function is multiplied by 0.001 by means of a second **`multiply`** function. This result is then connected to the destination item of the XML component (`/Order/LineItems/LineItem/Article/Tax`).

To preview the output of the mapping design, click the **Output** tab in the lower part of the mapping window.

Step 4 (optional): Style the mapping output with StyleVision

Optionally, you can link the target XML component to a StyleVision Power StyleSheet (.sps) file. (For more information about StyleVision, see https://www.altova.com/stylevision.) This would enable you to generate the mapping output as HTML, RTF, PDF, or 2007+ file (provided that StyleVision is installed and the prerequisites required by each format are in place, see Styling Mapping Output with StyleVision1336). To take this optional step, double-click the header of the XML component, and, next to "StyleVision Power StyleSheet file", browse for the **Order.sps** file available in the MapForce Examples folder. Back on the mapping area, you can now click the HTML, PDF, RTF, and Word 2007+ tabs to view the mapping output in the corresponding format.

4.3.17 UN/EDIFACT

UN/EDIFACT (the United Nations rules for Electronic Data Interchange for Administration, Commerce and Transport) is a set of standards for electronic document interchange. MapForce supports the following UN/EDIFACT Directories: D.93A or later. The latest UN/EDIFACT version is the default installed version in MapForce.

A separate installer for all previous versions back to D.93A, including the configuration files required for Interactive EDI mappings, is available on the Altova MapForce Libraries page.

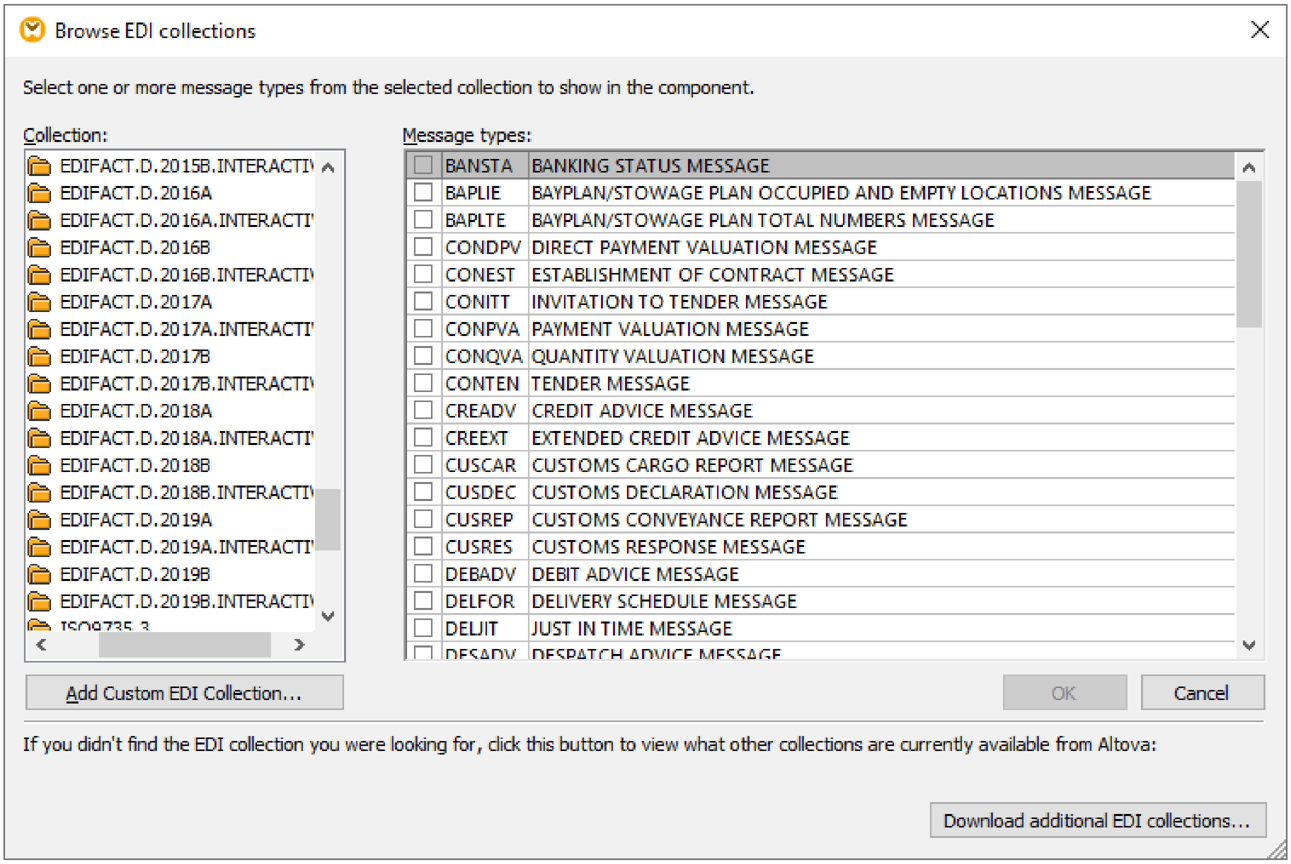
4.3.17.1 Batch Versus Interactive EDIFACT Messages

MapForce supports mapping data to or from both batch and interactive UN/EDIFACT messages. In other words, the UN/EDIFACT component in MapForce provides the necessary structure required to read or write syntactically valid UN/EDIFACT format files (meant for either batch or interactive interchange). Also, some basic validation and auto-completion rules are enforced when reading or writing UN/EDIFACT format files, as further described in this documentation. Note that MapForce does not actually transmit files between parties that take part in the UN/EDIFACT interchange process. The role of MapForce is to generate such files or convert them to other formats.

There are no special requirements in MapForce to map data to or from batch EDIFACT messages. However, in order to map data to or from Interactive EDIFACT messages, the following requirements apply:

1. MapForce 2018 or later is required.
2. The MapForce configuration files required for supporting Interactive EDI messages must be installed separately. The installation file can be freely downloaded and installed from the Altova MapForce Libraries page. Once you do this, the required collections and message types become available in the **Browse EDI collections** dialog box (*see screenshot below*).

To add a batch or an interactive UN/EDIFACT component to the mapping, select the corresponding collection from the **Browse EDI collections** dialog box. To open this dialog box, go to the menu command **Insert | EDI** or click the **Insert EDI File** toolbar button. For more information, see Adding EDI Components to the Mapping 447.



Note: In the dialog box above, the files with an INTERACTIVE component indicate a collection applicable to the interactive EDIFACT structure. Collections applicable to the batch structure only include the UN/EDIFACT directory version in their title (e.g., EDIFACT.D.2016A). The same convention applies to the configuration directories (collections) available in the EDI configuration folder of MapForce: **`C:\Program Files\Altova\MapForce2023\MapForceEDI\`**.

The structure of UN/EDIFACT batch files is different from that of files meant for interactive interchange. Therefore, an EDIFACT component in MapForce can be either batch OR interactive (and thus include either batch OR interactive messages), not both simultaneously. Likewise, the validation and auto-completion rules differ for both scenarios, as described in the following topics:

- UN/EDIFACT Validation Rules (Batch) 510
- UN/EDIFACT Auto-Completion Rules (Batch) 510
- UN/EDIFACT Validation Rules (Interactive) 511
- UN/EDIFACT Auto-Completion Rules (Interactive) 511

4.3.17.2 UN/EDIFACT Validation Rules (Batch)

When validating UN/EDIFACT documents, MapForce performs the following checks:

- Whether a UNB and a UNZ segment exist.
- Whether UNB/S004 contains a valid date/time specification.
- Whether UNB/0020 and UNZ/0020 contain the same value.
- Whether UNZ/0036 contains the correct number; which is defined as the number of functional groups, if present, or the number of messages. If there are functional groups, this should be the number of functional groups, otherwise it should be the number of messages contained in the interchange.

Each functional group is checked:

- Whether it contains a matching UNG and UNE pair.
- Whether UNG/S004 contains a valid date/time specification.
- Whether UNE/0060 contains the correct number of messages contained in the functional group.

Each message is checked:

- Whether it contains a matching UNH and UNT pair.
- Whether UNH/S009/0052 contains the same value as UNG/S008/0052 of the enclosing functional group.
- Whether UNH/0062 and UNT/0062 contain the same value.
- Whether UNH/S009/0065 contains the correct message type specifier.
- Whether UNT/0074 contains the correct number of segments contained in the message.

4.3.17.3 UN/EDIFACT Auto-Completion Rules (Batch)

When generating UN/EDIFACT files, MapForce may automatically create certain fields or values, according to the rules below (subsequently referred to as "automatic data completion", or "auto-completion" rules). To disable this behaviour, clear the **Auto-complete missing fields** check box in EDI Component Settings 450.

Automatic data completion for UN/EDIFACT makes sure:

- That a UNB and a UNZ segment exist
- If either UNG or UNE exist, that the other ID also exists
- That a UNH and a UNT segment exist
- That UNB/S001 exists. If it does not contain data, the syntax level and syntax version number from the user-defined settings are used (see the setting **Extended | Syntax version number** of the EDI Component Settings 450)
- That UNB/S002 and UNB/S003 exist.
- That UNB/S004 exists. If it does not contain data, the current date/time in EDI format is inserted.
- That UNZ/0036 exists. If it does not contain data, the number of functional groups or messages is calculated and inserted.
- That UNZ/0020 exists. If it does not contain data, the value from UNB/0020 is copied.

Note: Any fields not mentioned here are not inserted or created. The correct values cannot be ascertained automatically. Given a (target) parent element A (in the target EDI component) with child items x, y, and z - where y is mandatory, parent element A will only be created in the output file if the mandatory child element "y" in the target component has been mapped.

Functional group checking makes sure:

- That UNG/0038 exists. If it does not contain data, the name of the message is inserted.
- That UNG/S006 and UNG/S007 exist.
- That UNG/S004 exists. If it does not contain data, the current date/time in EDI format is inserted.
- That UNG/0051 exists. If it does not contain data, the first two characters of the controlling agency from the user-defined settings are inserted (see the setting **Extended | Controlling agency** of the EDI Component Settings 450)
- That UNE/0060 exists. If it does not contain data, the number of messages in the group is calculated and inserted.
- That UNE/0048 exists. If it does not contain a value, the value from UNG/0048 is copied.

Message checking makes sure:

- That UNH/S009/0065 exists. If it does not contain data, the name of the message is inserted.
- That UNH/S009/0052 and UNH/S009/0054 exist.
- That UNH/S009/0051 exists. If it does not contain data, the first two characters of the controlling agency from the user-defined settings are inserted (see the setting **Extended | Controlling agency** of the EDI Component Settings 450)
- That UNT/0074 exists. If it does not contain data, the number of segments in the message is calculated and inserted.
- That UNT/0062 exists. If it does not contain data, the value from UNH/0062 is copied.
- That UNH/0062 exists. If it does not contain data, the value from UNT/0062 is copied. (If only the trailer segment number is mapped, then the corresponding field in the header segment is supplied with the same value.)

4.3.17.4 UN/EDIFACT Validation Rules (Interactive)

The following validation rules apply for MapForce components which contain Interactive EDIFACT messages:

- If `UIB` is present, then all following `UIH/S302` should match `UIB/S302`
- `UIH/S306/F0065` should contain message type (validated by the parser)
- `UIH/S306/F0052` should contain message version number that is defined in selected configuration files
- `UIH/S306/F0054` should contain message release number that is defined in selected configuration files
- `UIT/F0340` should match to corresponding `UIH/F0340` (message header/trailer) or not be present (optional field)
- `UIT/F0074` should contain segments count or not be present (optional field)
- `UIZ/S302` should match `UIB/S302` or not be present (optional composite)
- `UIZ/F0036` should contain messages count or not be present (optional field).

4.3.17.5 UN/EDIFACT Auto-Completion Rules (Interactive)

The following auto-completion rules apply for MapForce components which contain Interactive EDIFACT messages:

- `UIB/S001/F0001` will be filled with controlling agency + syntax version number
- `UIB/S001/F0002` will be filled with syntax version number
- `UIB/S002/F0004` will be filled with "Sender"
- `UIB/S003/F0010` will be filled with "Recipient"

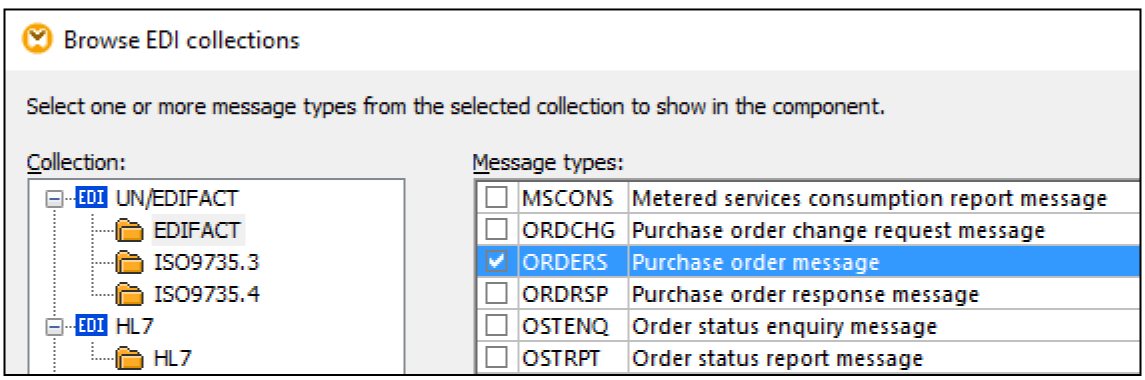
- `UIZ/S302/*` will be filled with `UIB/S302/*`
- `UIZ/F0036` will be filled with the message count
- `UIH/S302/*` will be filled with `UIB/S302/*` (if `UIB` and `UIB/S302` is present)
- `UIH/S306/F0065` will be filled with the message type
- `UIH/S306/F0052` will be filled with the message version number from configuration files
- `UIH/S306/F0054` will be filled with the message release number from configuration files
- `UIT/F0340` will be filled with `UIH/F0340` (this is an optional field but will be completed anyway)
- `UIT/F0074` will be filled with the number of segments in the message (this is an optional field but will be completed anyway).

4.3.17.6 Example: Mapping UN/EDIFACT to XML

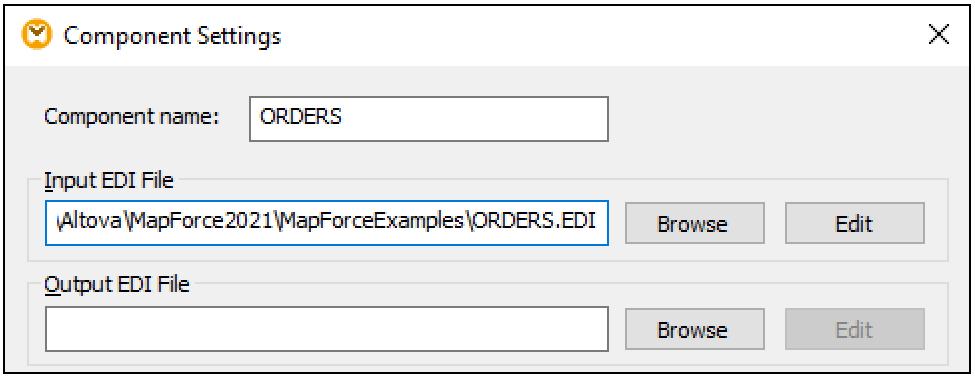
This example shows how to map data from UN/EDIFACT messages to an XML schema, in order to produce an XML instance file for further processing. The mapping created in this example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\EDI_Order.mfd**.

Step 1: Add the UN/EDIFACT component to the mapping

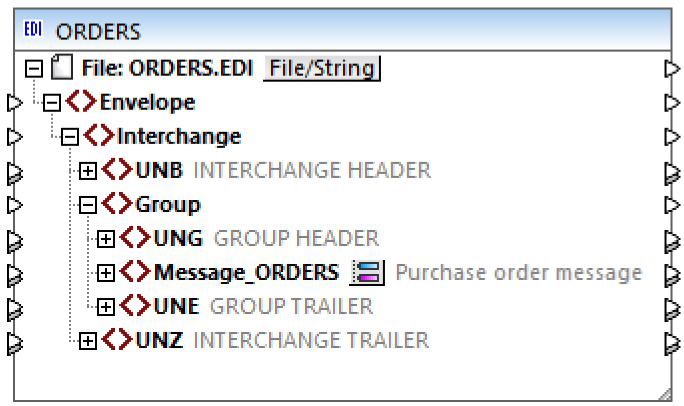
1. Create a new mapping and select one of the following transformation languages: Java, C#, C++, or BUILT-IN. In this example, Java is selected as transformation language.
2. On the **Insert** menu, click **EDI**.
3. On the **EDI collections** dialog box, select the **EDIFACT** collection, then select the **ORDERS** message type, and click **OK**.



4. When prompted to supply a sample EDI file, click **Browse** and open the **ORDERS.EDI** file from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory. After you open the file, the Component Settings dialog box opens. This enables you to review the settings of the EDI component before adding it to the mapping. You can change these settings at any time later if required (see EDI Component Settings 450). Notice that the **ORDERS.EDI** file appears as **Input EDI file**.

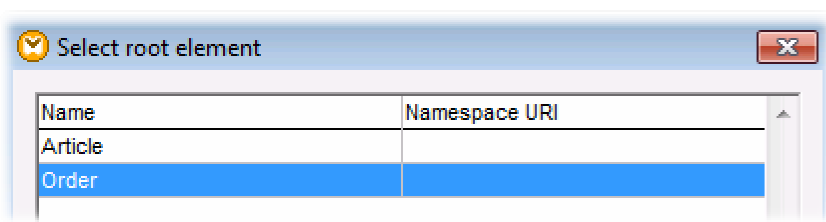


5. Click **OK**. The EDI component is now displayed in the mapping area. Double-click the **Message_ORDERS** node to view its child items. To resize the component, click and drag the lower-right corner of the component window.

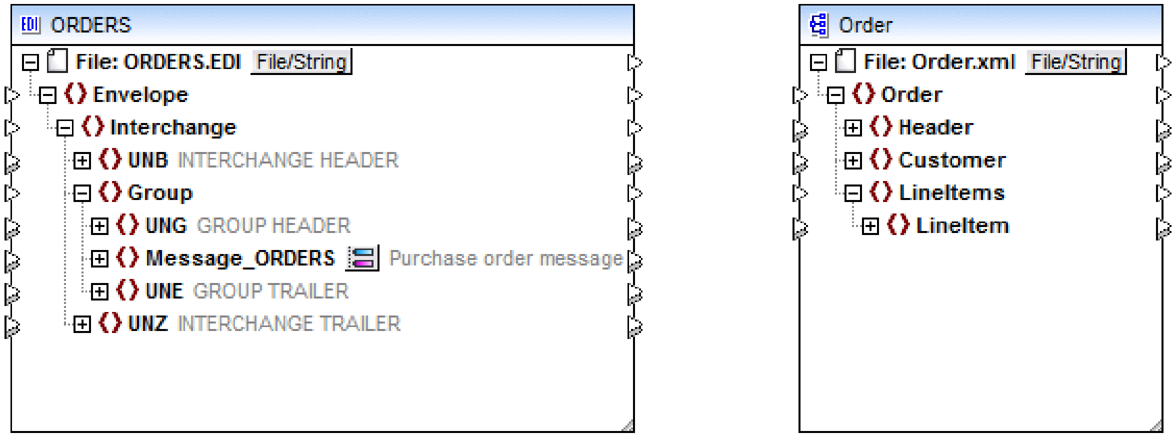


Step 2: Add the target schema component to the mapping

1. On the **Insert** menu, click **XML Schema/File**, and open the **Order.xsd** file from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
2. When prompted to supply a sample XML file, click **Skip** and select **Order** as the root of the target document.



At this point, both the source EDI component and the target XML schema are on the mapping area, so we are ready to start drawing the mapping connections.



Step 3: Map the EDI items

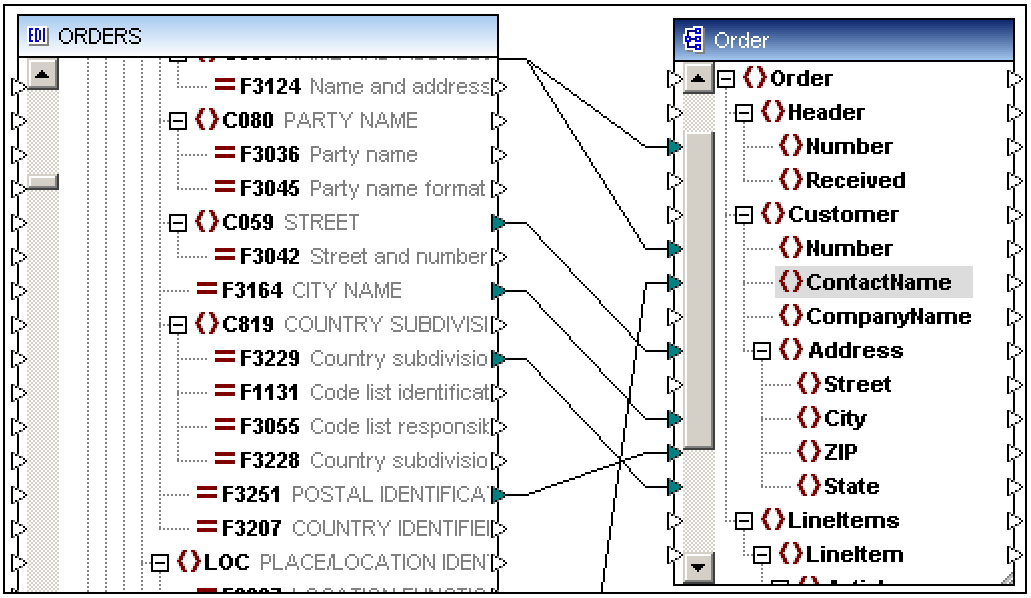
The EDI component displays the structure of a message based on the collection (ORDERS) we selected. Typically, not all of the nodes will actually contain data, so you must be sufficiently familiar with the EDI documents being worked on, to locate the relevant nodes.

In MapForce, you map a source item and a target item by drawing a connection between them. For step-by-step instructions on how to create mapping connections in MapForce, see Working with Connections 89.

In this example, the following nodes (starting from the **Group/Message_ORDERS** node) will be mapped (connected) first:

Source	Target
`BGM/C106/F1004`	`Order/Header/Number`
`SG2/NAD/C082/F3039`	`Order/Customer/Number`
`SG2/NAD/C080/F3036`	`Order/Customer/CompanyName`
`SG2/NAD/C059/F3042`	`Order/Customer/Address/Street`
`SG2/NAD/F3164`	`Order/Customer/Address/City`
`SG2/NAD/C819/F3229`	`Order/Customer/Address/State`
`SG2/NAD/F3251`	`Order/Customer/Address/ZIP`
`SG2/SG5/CTA/C056/F3412`	`Order/Customer/ContactName`

At this stage, the mapping should look similar to the image below:



Continue the mapping process and map:

Source	Target
`SG29`	`Order/LineItems`
`SG29/LIN/C212/F7140`	`Order/LineItems/LineItem/Article/Number`
`SG29/IMD/C273/F7008`	`Order/LineItems/LineItem/Article/Name`
`SG29/QTY/C186/F6060`	`Order/LineItems/LineItem/Article/Amount`
`SG29/SG33/PRI/C509/F5118`	`Order/LineItems/LineItem/Article/SinglePrice`

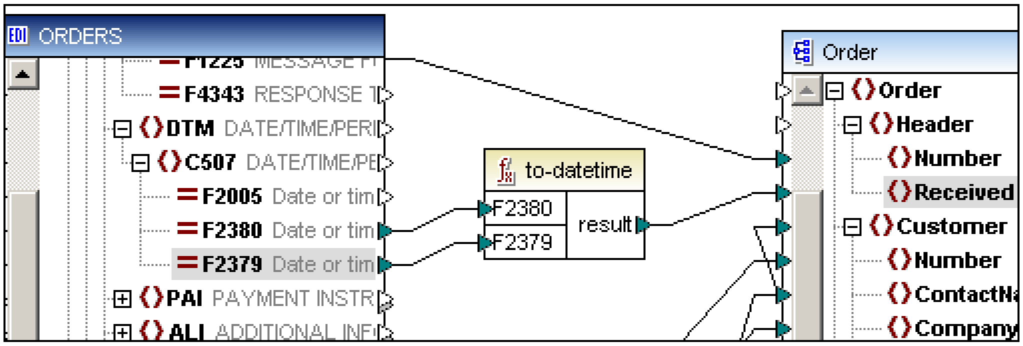
Step 4: Format the date

Drag the **`to-datetime`** function from the **edifact** library into the Mapping area, see also Add a Function to the Mapping 905.

By supplying as arguments to this function the **F2380** and **F2379** components of the **DTM/C507** element, we can create an appropriately formatted **Received** datetime.

We therefore map the following fields:

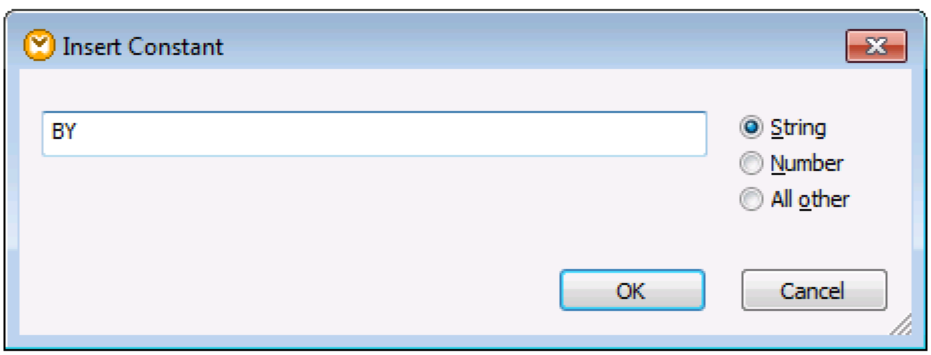
Source	Target
`DTM/C507/F2380`	The **F2380** input of the to-datetime function
`DTM/C507/F2379`	The **F2379** input of the to-datetime function
The **result** of the **`to-datetime`** function	`Order/Header/Received`



Step 5: Filter the buyer purchase orders

At this point we want to filter the "Buyer" purchase orders. These can be identified by the party function code qualifier of the NAD (Name and address) segment. In this case, the value 'BY' indicates a "Buyer" (Party to whom merchandise and/or service is sold).

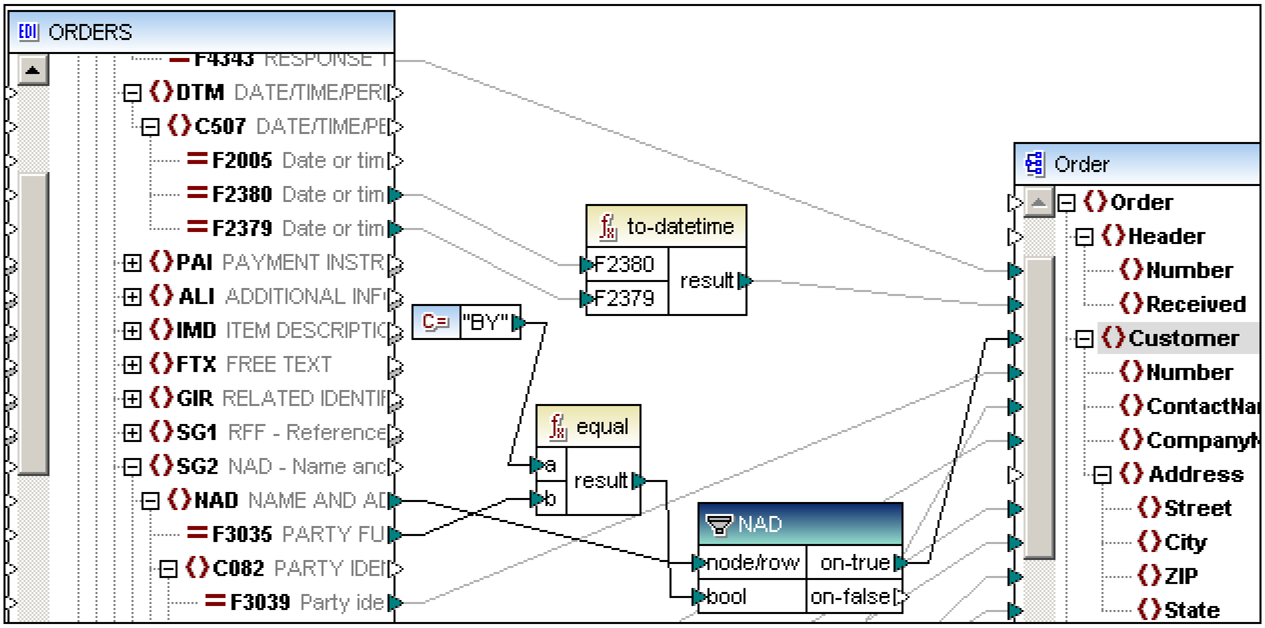
1. Drag the **`equal`** function from the **core** library into the Mapping area.
2. Add a filter to the mapping (On the **Insert** menu, click **Filter: Nodes/Rows**).
3. Add a constant to the mapping (On the **Insert** menu, click **Constant**). Assign to the constant the value "BY" by entering "BY" into the text field:



Map the following items:

Source	Target
`SG2/NAD/F3035`	The **b** input of the equal function
The constant "BY"	The **a** input of the equal function
The result of the **`equal`** function	The **bool** input of the filter component
`SG2/NAD`	The **node/row** input of the filter component
The **on-true** result of the filter component	`Order/Customer` in the schema

The aim here is to only map data if the NAD node refers to a 'Buyer', as identified by the party function code qualifier 'BY'.



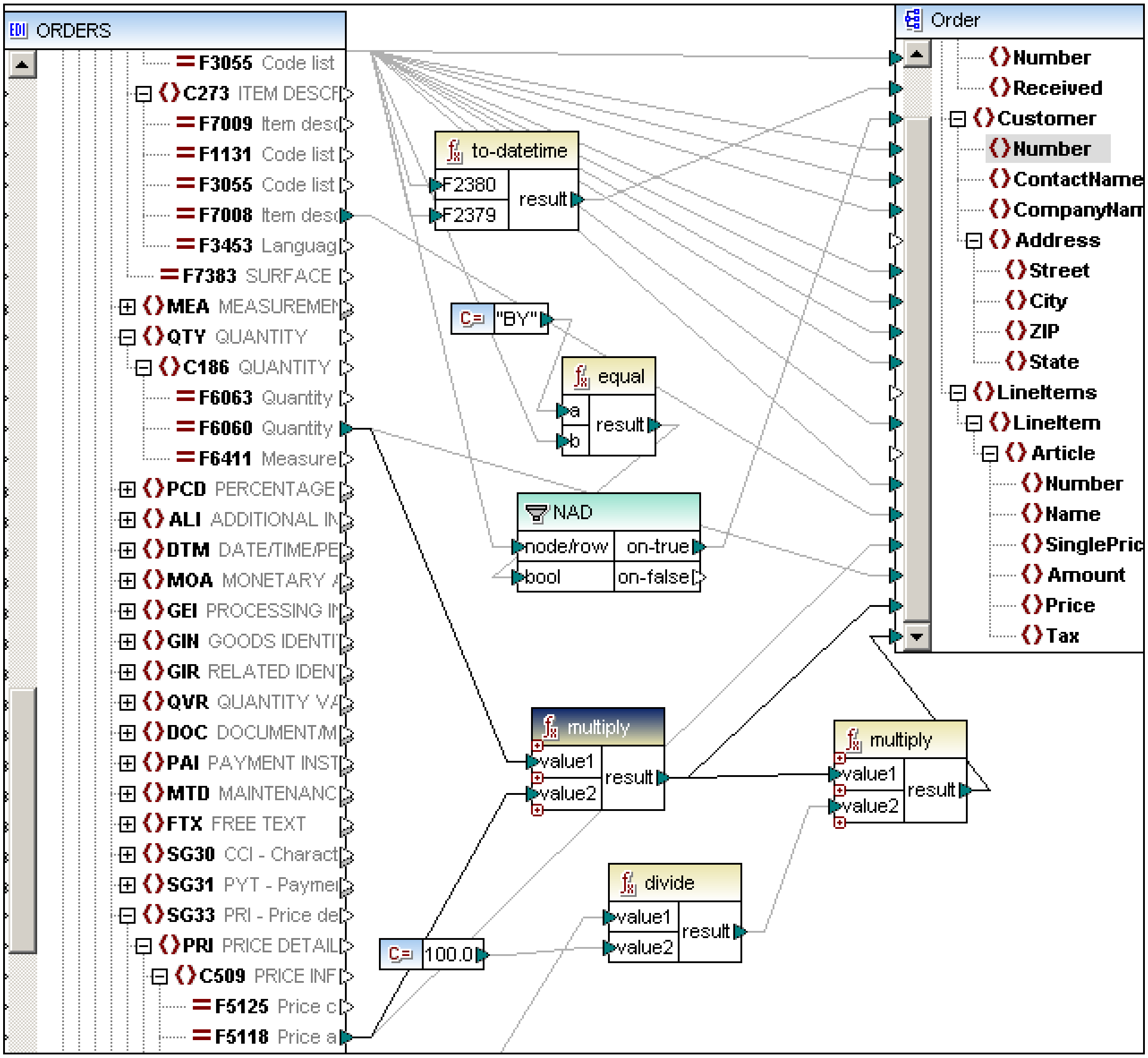
Step 6: Calculate pricing and tax

The final step in this task is to calculate the pricing and tax costs.

1. From the Libraries window, drag two **`multiply`** and one **`divide`** function (of the `core` library) into the Mapping area.
2. Insert a **Constant** component (**Insert | Constant**). Make sure "Number" is selected as type, and enter 100.0 into the text field.
3. Map the following items:

Source	Target
`SG29/QTY/C186/F6060`	**value1** of the first **`multiply`** function
`SG29/SG33/PRI/C509/F5118`	**value2** of the first **`multiply`** function
The result of the first **`multiply`** function	`Order/LineItems/LineItem/Article/Price`
`SG29/SG40/TAX/C243/F5278`	**value1** of the **`divide`** function
The constant "100.0"	**value2** of the **`divide`** function
The **result** of the first **`multiply`** function	**value1** of the second **`multiply`** function
The **result** of **`divide`** function	**value2** of the second **`multiply`** function
The **result** of the second **`multiply`** function	`Order/LineItems/LineItem/Article/Tax`

Your mapping should now look like this:



Clicking the output tab performs an "on the fly" transformation and presents you with the XML document result:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Header>
    <Number>ABC123456XYZ</Number>
    <Received>2004-04-30T17:42:00-09:00</Received>
  </Header>
  <Customer>
    <Number>123</Number>
    <ContactName>Michelle Butler</ContactName>
    <CompanyName>Nanonull, Inc.</CompanyName>
    <Address>
      <Street>119 Oakstreet Suite 4876</Street>
      <City>Vereno</City>
      <ZIP>29213</ZIP>
      <State>CA</State>
    </Address>
  </Customer>
  <LineItems>
    <LineItem>
      <Article>
        <Number>42</Number>
        <Name>Pizza Pepperoni</Name>
        <SinglePrice>7.2</SinglePrice>
        <Amount>1</Amount>
        <Price>7.2</Price>
        <Tax>0.648</Tax>
      </Article>
    </LineItem>
```

4.3.18 Customizing EDI Structure

By default, MapForce represents the structure of EDI files on the mapping according to their formal specifications. In most cases, it shouldn't be necessary to manually customize the default EDI configuration. However, if you need to process various non-standard EDI formats and variations, then customization is required. For example, you may need to slightly adapt the structure or validation rules of certain data elements. This can be achieved by adding a custom EDI collection to MapForce, and then configuring it according to your business needs, which is the subject of this chapter.

> You can create EDI variations (custom EDI collections) for the following standards already supported by MapForce: EDIFACT (ISO 9735), ASC X12, HIPAA X12, HL7, NCPDP SCRIPT, SWIFT MT, or TRADACOMS. Creating a new EDI standard from scratch is not supported.

4.3.18.1 Creating a Custom EDI Collection

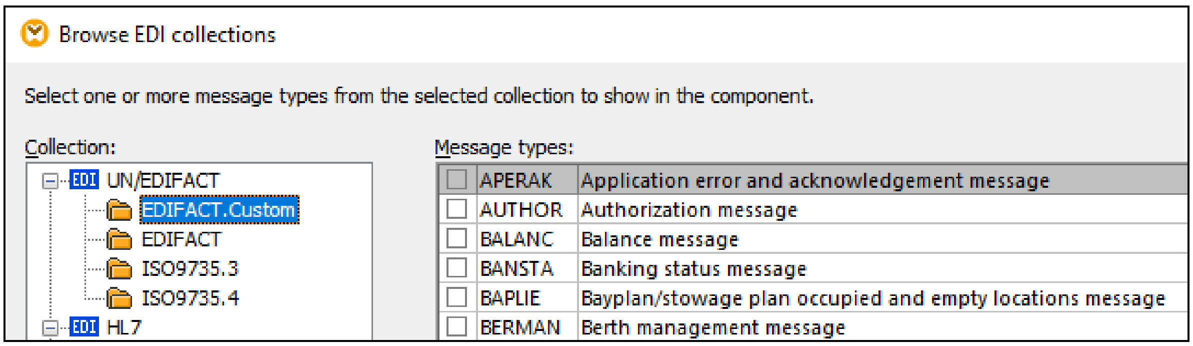
An "EDI Collection" is a directory which contains configuration files applicable to a specific EDI standard. Each EDI collection contains a file named "EDI.Collection" and various other files referenced by it, directly or indirectly. The last directory on the path is the name of the EDI collection as it appears in the graphical user interface of MapForce when you add an EDI component to the mapping [447] (for example, "UN/EDIFACT", "HL7, "ASC X12", and so on).

After installing MapForce, several built-in EDI collections exist at the following path: **C:\Program Files\Altova\MapForce2023\MapForceEDI**. If you run 32-bit MapForce on a 64-bit operating system, adjust the directory path to **C:\Program Files (x86)\Altova\MapForce2023\MapForceEDI**.

> By default, the directory above contains only the built-in EDI collections. You can download all the supported versions of each EDI collection from the Altova Download Center (https://www.altova.com/mapforce/download#mapforce_libraries).

There are three ways to create custom EDI collections in MapForce:

1. Create a custom EDI collection in the following directory: **C:\users\<name>\Documents\Altova\MapForce2023\MapForceEDI**. For example, if you copy **C:\Program Files\Altova\MapForce2023\MapForceEDI\EDIFACT** to **C:\users\<name>\Documents\Altova\MapForce2023\MapForceEDI\EDIFACT.Custom**, the "EDIFACT.Custom" collection will appear in the list of EDI collections when you add an EDI component to the mapping [447].

   Typically, your custom EDI collections should be distinguishable from the built-in EDI collections. For this purpose, they should not have the same name as the built-in collections. For example, the directory of your custom EDI Collection should not be called "EDIFACT", because this name is already used by the built-in collection from **C:\Program Files\Altova\MapForce2023\MapForceEDI\EDIFACT**.

   However, if you specifically need your custom EDI collection to take precedence over a default EDI collection from **C:\Program Files\Altova\MapForce2023\MapForceEDI**, then name your custom EDI collection in the same way as the built-in EDI collection.

2. Designate a custom directory where your EDI collection is to be stored. For example, you can copy **C:\Program Files\Altova\MapForce2023\MapForceEDI\EDIFACT** to **C:\EDIFACT.Custom**. Next, click the **Add Custom EDI Collection** button on the dialog box above and browse for **C:\EDIFACT.Custom**. To make the path of the EDI collection relative to the mapping (.mfd) file, select the **Save all file paths relative to the MFD file** check box.

   Custom EDI collections added this way affect only the current EDI component on the mapping.

3. (Not recommended) Create your custom EDI collection in the **C:\Program Files\Altova\MapForce2023\MapForceEDI** directory. Note that making changes in this directory

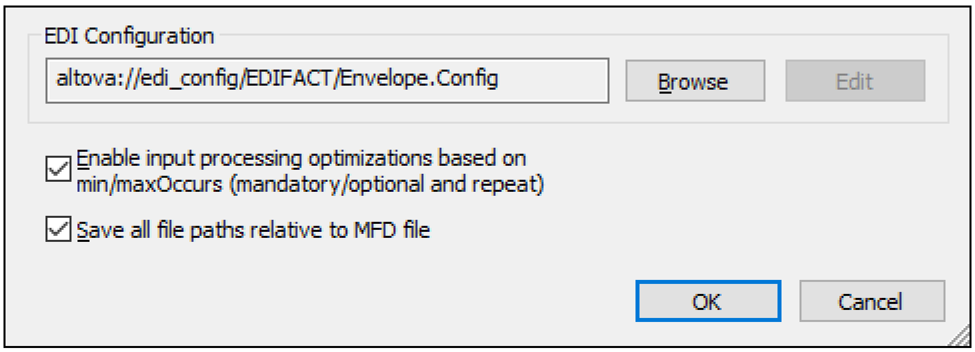
requires administrative rights. This approach is supported only for backward compatibility and is not recommended.

> It is not recommended to alter the contents of original files and folders in the **C:\Program Files\Altova\MapForce2023\MapForceEDI** directory, unless you want to override, at your own responsibility, the default EDI validation and processing in MapForce. Instead, use either the first or the second approaches described above. This will also make it easier to migrate your custom configuration to a newer version of MapForce.

How MapForce resolves EDI configurations

To view the configuration (EDI Collection) used by any EDI component:

- Double-click the title bar of the component on the mapping. (Alternatively, right-click the component title bar, and select **Properties** from the context menu.) The EDI Component Settings[450] dialog box appears.



The **EDI Configuration** property indicates the path to the configuration (EDI Collection) used by the component. If this path begins with `altova://edi_config/`, then MapForce will look for the respective configuration in the following directory (the order is important):

1. In your user's profile directory on Windows (**C:\users\<name>\Documents\Altova\MapForce2023\MapForceEDI**\). If the configuration is found at this path, then it takes precedence over the one from **C:\Program Files\...** (see the next bullet).
2. In the **C:\Program Files\Altova\MapForce2023\MapForceEDI** directory.

If the **EDI Configuration** property indicates a standard Windows path, then the EDI Collection at the respective path is used by the component.

Examples

For step-by-step EDI customization examples, see:

- Example: Adding a TA1 Segment[536]
- Example: Customizing an EDIFACT message[538]
- Example: Customizing an ANSI X12 transaction[545]

4.3.18.2 EDI Configuration Files

An EDI collection (directory) may contain any of the following configuration files:

File	Purpose
EDI.Collection	The **EDI.Collection** file defines all message types present in the custom EDI collection that you are implementing. You can edit this file to add or remove message types that should be available for selection in MapForce (on the "Browse EDI Collections" dialog box). See also Adding or Removing Message Types[526].
Envelope.Config	The **Envelope.config** file defines the structure of the EDI mapping component in MapForce (that is, elements that are surrounded by the **Envelope** sequence). Under normal circumstances, this file should not be edited manually.
<Message>.Config	The EDI collection directory must include one or several **<Message>.Config** files. One such file corresponds to an EDI message type, and contains the group and segment definitions used in that message. The **<Message>.Config** files are referenced from the **EDI.Collection** file. You can modify the **<Message>.Config files** if you want to alter the structure or validation rules applicable to fields of a particular message. Note that changes to these files are considered local customizations (that is, they apply only to that specific message). For examples, see Customizing an EDIFACT message[538] and Customizing an ANSI X12 transaction[545].
<Filename>.Segment	This file defines the data elements, segments, composites, and subcomposites of the EDI files, and is used when parsing the EDI file. You can modify the .segment files if you want to alter the structure or validation rules applicable to a field (data element) in all messages where it may occur. Changes made to this file are global customizations.
<Filename>.Codelist	This file defines EDI code values used by MapForce to validate the EDI input and output files in MapForce. In general, the code values defined here correspond to those defined by the respective EDI standard. Exceptions to this rule are some edge cases where the standard relies on external code values and does not fully enforce the list of possible values (see also EDI component validation[454]). You can modify .codelist files if you need to change the default validation rules (for example, in order to relax validation, or add new codes that are not already defined in the EDI code list). For more information, see Relaxing Field Validation[553] and Adapting Code Values from External Sources[555].
ParserErrors.Config	This file is applicable to the X12 and HIPAA X12 standards only. It defines the structure of "ParserErrors_Message" and "ParserErrors_Group" items on the EDI component in MapForce. These items contain information about parsing errors encountered during file processing and can be used to

File	Purpose
	produce X12 997 (Functional Acknowledgement) and X12 999 (Implementation Acknowledgement) messages.

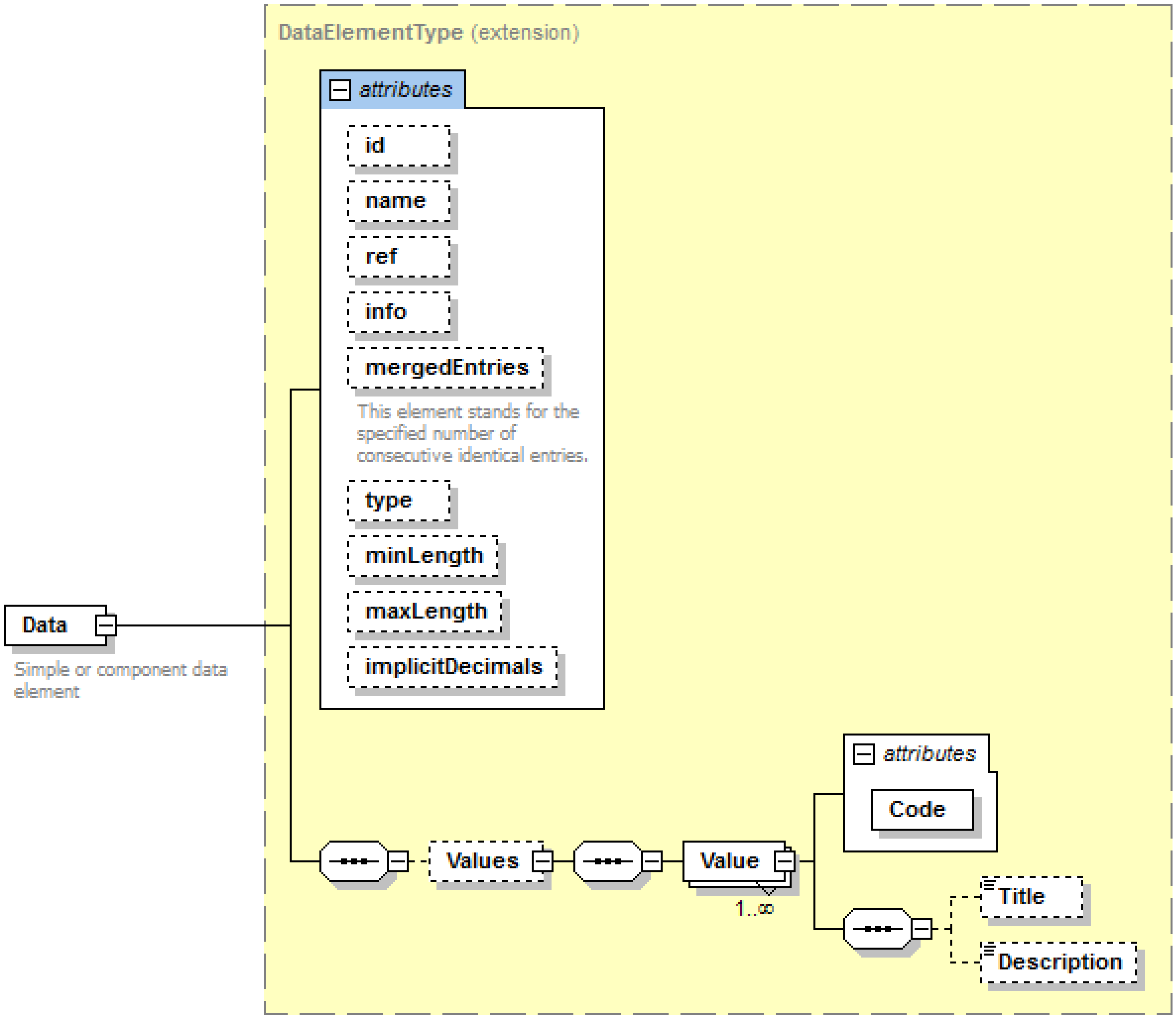
All of the files above are XML files and can be edited in an XML editor such as XMLSpy. They all refer to some XML schema (see EDI Configuration Schemas523), with the exception of the **EDI.Collection** file. The **EDI.Collection** file has no explicit schema defined but is validated internally by MapForce.

4.3.18.3 EDI Configuration Schemas

Whenever you make a change to a configuration file, it is important to validate it against its applicable XML schema. The schema is available in the directory **C:\Program Files\Altova\MapForce2023\MapForceEDI** and it can be one of the following:

File	Purpose
EDIConfig.xsd (version "3")	This configuration schema is valid starting with MapForce 2011, Release 2. If you have configuration files that are using a schema prior to version 3, they must be upgraded to use this schema (or a newer one). For more information, see Upgrading Older Configuration Files524.
EDIConfig4.xsd	This configuration schema is valid starting with MapForce 2011, Release 3. It contains multiple enhancements to the original **EDIConfig.xsd** schema, including the ability to define custom recursive hierarchies which might be necessary when you map data to or from X12 or HL7 formats.
EDIConfig5.xsd	This configuration schema is valid starting with MapForce 2017. It introduces the ability to define a data element that has incomplete validation rules. With this schema, it is possible to define advanced field validation rules at character level (see Configuring Character-Level Validation559).
EDIConfig6.xsd	This configuration schema is valid starting with MapForce 2018. It introduces the ability to map data to or from NCPDP SCRIPT format, see NCPDP SCRIPT480 .
EDIConfig7.xsd	This configuration schema is valid starting with MapForce 2021. It introduces the ability to map data to or from SWIFT MT485.

The EDI configuration schema will also offer information about the structure of any elements that may be present in the configuration files. For example, below is a representation of an EDI Data Element structure as it appears in the **EDIConfig4.xsd** schema:



Structure of the Data element (Schema view of XMLSpy)

4.3.18.4 Upgrading Older Configuration Files

EDI configuration files contain a reference to (and are validated against) one of the XML schemas available in the **MapForceEDI** directory (see EDI Configuration Schemas [523]). To view the schema and version of a configuration file, open the file in a text editor and check the following line at the beginning:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd" Version="5">
```

MapForce 2023 recognizes custom EDI configurations that are based on **EDIConfig.xsd** schema version 3 or later, introduced in MapForce 2011, Release 3. Compared to previous versions, this schema adds the ability to map data to or from multiple message types in the same EDI component. If you were using custom EDI configuration files that point to a schema version prior to version 3, then your configuration files must be upgraded. Otherwise, when you attempt to add the custom EDI collection to the mapping, the following message appears on the "Browse EDI Collections" dialog box:

> "The selected EDI collection uses an old configuration file version that supports only a single message type per component."

The following instructions show you how to upgrade the schema of an EDI configuration file from **EDIConfig.xsd** version 2, to **EDIConfig.xsd** version 3.

To upgrade the configuration files:

1. Copy **Envelope.Config** from the original configuration folder (for example, "EDIFACT") to the folder containing your customized EDI collection (for example, "EDIFACT.Custom").
2. Edit **EDI.Collection**, and change the root element's `Version` attribute from "2" to "3".
3. Add **`<Root File="Envelope.Config"/>`** after the `</Meta>` tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="3">
  <Meta>
    <Version>D</Version>
    <Release>04B</Release>
    <Agency>UN</Agency>
    </Meta>
    <Root File="Envelope.Config"/>
    <Message Type="ORDERS" File="ORDERS.Config" Description="Purchase order
message"/>
</Messages>
```

4. Edit **ORDERS.Config**, and change the root element's `Version` attribute from "2" to "3".
5. Add **`<Format standard="EDIFACT"/>`** (or, depending on the case, X12, or HL7) after the `</Meta>` tag.
6. Rename **`<Group name="Message"...>`** to **`<Group name="Message_ORDERS"...>`** (or whatever the custom message type is), and remove the outer group levels ("Envelope", "Interchange", and their segments) as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="3">
   <Meta>
      <Version>D</Version>
      <Release>04B</Release>
      <Agency>UN</Agency>
   </Meta>
   <Format standard="EDIFACT"/>
   <Include href="Admin.Segment"/>
   <Include href="EDSD.Segment"/>
   <Include href="UNCL.Codelist"/>
   <Message>
      <MessageType>ORDERS</MessageType>
      <Description>Purchase order message</Description>
      <Revision>14</Revision>
      <Date>2004-11-23</Date>

      <!-- DELETED TEXT -->

               <Group name="Message_ORDERS" maxOccurs="unbounded" info="UNH -
Message header">
                  &lt;Segment ref="UNH"/&gt;
                  &lt;Segment ref="BGM"/&gt;
                        &lt;Segment ref="UNT"/&gt;
               </Group>

     <!-- DELETED TEXT -->

   </Message>
</Config>
```

If a mapping was open while you were editing the configuration file, it should be reloaded. The connections will be automatically remapped from "Message" to "Message_ORDERS" item.

4.3.18.5 Adding or Removing Message Types

To add custom EDI message types (or transaction types, in X12 terminology), follow the steps below:

1. In the directory where your custom EDI collection is stored, create a **<Message>.Config** file for each message (transaction) type to be added to the collection.
2. Modify the **EDI.Collection** file to make a reference to each available message (transaction) type.

To remove existing message types:

- Comment out the required `Message` elements in the **EDI.Collection** file.

Adding and configuring the <Message>.Config file

Each **<Message>.config** file you create must be valid according to its configuration schema (see EDI Configuration Files 522). To avoid problems, pay special attention to the following elements in the body of each message configuration file:

- Meta
- Format
- Include
- Message

The `Meta` element indicates the specification version, release, and agency, for example:

```
<Meta>
   <Version>D</Version>
   <Release>14B</Release>
   <Agency>UN</Agency>
</Meta>
```

The `Format` element indicates the name of the base standard, for example:

```
<Format standard="EDIFACT"/>
```

The following values are valid:

- Fixed
- EDIFACT
- X12
- HL7
- TRADACOMS (this value requires the **EDIConfig4.xsd** schema or later, see EDI Configuration Files and Schemas 522)

The `Include` element in a **<Message>.Config** file specifies references to any additional configuration files. For example, the code listing below includes a reference to two .segment files and one .codelist file. A .segment file stores all Segment, Composite, and Data Element definitions at a global level (this is to avoid repeating them in each message where they may occur). Likewise, a .codelist file defines validation rules applicable to each field, at a global level.

```
<Include href="Admin.Segment"/>
<Include href="EDSD.Segment"/>
<Include href="UNCL.Codelist"/>
```

Any file referenced by the `Include` element must exist in the same directory as the **<Message>.Config** file.

The `Message` element in each **<Message>.Config** file defines the structure of the message (that is, the hierarchy of groups, segments, and data elements), as well as any code values used for validation (if applicable). For more information about the structure of a message, see Changing the Message Structure 528. For examples of altering existing messages, see the following topics:

- Customizing an EDIFACT Message 538
- Customizing an ANSI X12 Transaction 545
- Defining Code Values at Message Level (Inline) 558

Modifying the EDI.Collection file

If you add a custom **<Message>.Config** file, a `Message` element must also be added to the **EDI.Collection** file. Each `Message` element must have the following attributes:

Type	Specifies the official code of the message type (for example "BALANC", "999").
File	Specifies the file name containing the specification of this particular message type. The actual file must exist in the same directory.
Description	Provides a description of this message type as it will appear in the MapForce graphical user interface.

For samples of valid **EDI.Collection** files, check any of the MapForce built-in EDI collections in the **C:\Program Files\Altova\MapForce2023\MapForceEDI** directory.

4.3.18.6 Changing the Message Structure

In each **<Message>.config** file, the structure of a `Message` element is determined by the XML schema of the EDI standard that you are customizing. This schema is called **EDIConfig.xsd** (or a similar name, see EDI Configuration Schemas 523). To build or change the structure of the message, you will therefore need to consult both the documentation of your custom EDI standard and the XML schema.

How to add segments, data elements, and composites

To add data elements, segments, composites, and subcomposites, create <Data>, <Segment>, <Composite>, and <SubComposite> elements as follows:

- The .segment file must contain the list of all reusable, globally-defined segments and data elements.
- The .config file corresponding to each message (or transaction) type must contain the actual structure of that message (the hierarchy of segments, composites, and data elements).

This makes it possible to define a data element once (in the .segment file) and reuse it in multiple messages (in .config files).

To refer to a data element, use the `ref` attribute. For example, the code listing below shows how data element F100 is defined in the **X12.Segment** file of the HIPAA.X12 collection.

```
<Data name="F100" type="string" maxLength="3" minLength="3" info="Currency Code" />
```

X12.Segment file

All segments, in all transaction types that need this data element, can now refer to it. The code listing below shows how the "CUR" segment of the "Health Care: Professional" (837P) transaction refers to the F100 data element previously defined:

```
<Include href="X12.Segment" />

...

&lt;Segment name="CUR" minOccurs="0" info="Foreign Currency Information"&gt;
    <Data ref="F100" info="Currency Code" />
&lt;/Segment&gt;
```

837P.Config file (X12)

A reference can be in the same file, or in an external file. In the code listing above, the .config file includes a reference to an external .segment file. For this reason, an `<Include href="X12.Segment" />` statement is present at the beginning of the **837P.Config** file to add the reference to the external **X12.Segment** file.

For another example that shows how to add a custom segment, see Example: Adding the TA1 Segment[536].

How to create groups and loops

Creating or changing EDI structures often requires grouping of segments or data elements under some parent node (a group). Even the message on the EDI component is a group, since it groups together all the multiple segments that this message consists of. Groups may be nested (they can have child groups). Also, they may be repeated multiple times (in which case, they become loops). Groups and loops can be created by adding a `<Group>` element. Use the `maxOccurs` attribute to set the number of allowed occurrences of a loop. For example, the following code listing shows a loop that can occur up to 25 times, is optional, and consists of two segments:

```
<Group name="LoopSAC" maxOccurs="25" minOccurs="0">
   &lt;Segment ref="SAC"/&gt;
   &lt;Segment ref="CUR" minOccurs="0"/&gt;
</Group>
```

850.Config file (X12)

How to specify the type and size of data elements

Use the `type` attribute of a data element or segment to specify its type (string, decimal, etc.). The possible data types (the `type` attribute) are generally any of XML Schema types that are used in the configuration files delivered by default. Other XML Schema simple types can tentatively be used, but cannot be guaranteed. The following simple types are not supported: "anyType", "ENTITIES" and "QName".

Use the `minLength` and `maxLength` attributes of a data element to specify its minimum and maximum length.

How to specify mandatory and optional data elements

To specify that a data element is mandatory, do not set the value of the `minOccurs` attribute (this attribute has "1" as default value, which means the field is mandatory by default). To specify that a data element is optional, set the `minOccurs` attribute to "0". For example, the code listing below defines two mandatory data elements (F1331, F1332) and one optional data element (F1325).

```
<Composite name="C023" info="Health Care Service Location Information">
   <Data ref="F1331"/>
   <Data ref="F1332"/>
   <Data ref="F1325" minOccurs="0"/>
</Composite>
```

4.3.18.7 Handling Multiple Consecutive Data Elements

EDI segments often allow the use of multiple consecutive elements of the same name. For example, this is applicable to the N2 segment of the "850 - PURCHASE ORDER" transaction set of the X12 standard. According to the specification, the N2 segment allows for two consecutive alphanumeric "Name" fields (one mandatory and one optional):

REF	ELE ID	NAME	RPT	ATTRIBUTES	
01	93	Name	M	AN	1/60
02	93	Name	O	AN	1/60

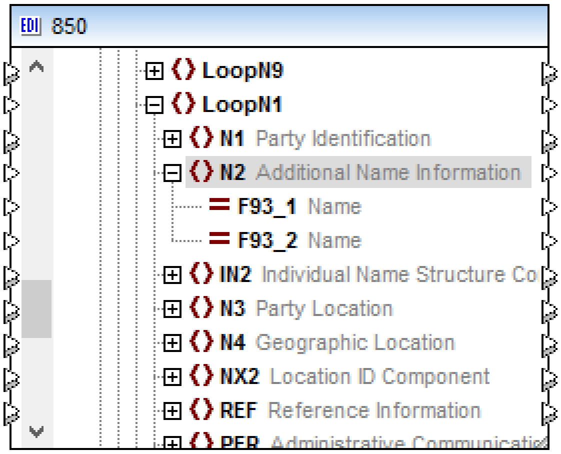
When you create or modify an EDI configuration, there are two possibilities to handle such data elements, as shown below.

Approach 1: Create separate mapping nodes for each consecutive data element

In this approach, you create multiple occurrences of the data element in the segment file. In this case, a new node will appear on the mapping for each occurrence of the data element. Consider this configuration:

```
&lt;Segment name="N2" info="Additional Name Information">
   <Data ref="F93" nodeName="F93_1"/>
   <Data ref="F93" minOccurs="0" nodeName="F93_2"/>
&lt;/Segment&gt;
```

On the mapping, this configuration creates two separate F93 nodes, each corresponding to an occurrence of F93:



X12 EDI component with separate F93 nodes

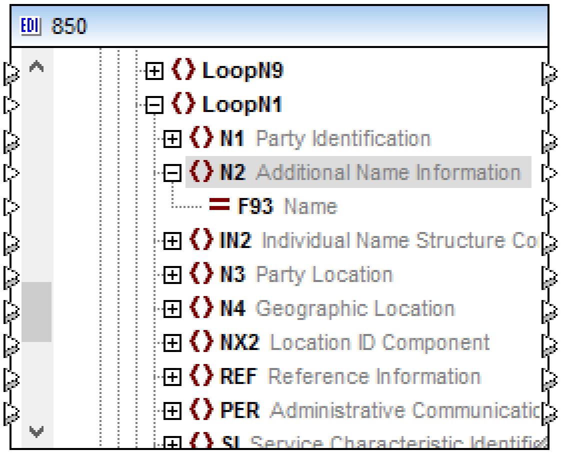
Using this approach is suitable when every occurrence of the data element is meaningful by itself and, consequently, you want a separate node to appear for it on the mapping.

Approach 2: Create a single node for all occurrences of the data element

In this approach, you treat all occurrences of the data element as one single node. To do this, add the `mergedEntries` attribute to the corresponding data element. In the code listing below, the `mergedEntries` attribute of data element F93 is set to "2". This allows for two consecutive occurrences of this field in the EDI instance file.

```
&lt;Segment name="N2" info="Additional Name Information"&gt;
   &lt;Data ref="F93" mergedEntries="2"/&gt;
&lt;/Segment&gt;
```

On the mapping, the node corresponding to the F93 data element will be displayed as a single node:



X12 EDI component with separate F93 nodes

Mapping data from a data element which has merged entries will create multiple duplicate elements on the target side, if the source EDI file contains multiple (not empty) occurrences of this element. For example, let's consider a source EDI file where the N2 segment contains two consecutive occurrences of F93. The first occurrence is "Michelle Butler" and the second is "Mrs".

```
N2+Michelle Butler+Mrs+'
```

Mapping data from such an EDI file would create, on the target side, the following output (notice the duplicate occurrences of `<ContactName>`):

```
<Customer>
      <Number>123</Number>
      <ContactName>Michelle Butler</ContactName>
      <ContactName>Mrs</ContactName>
      <CompanyName>Nanonull, Inc.</CompanyName>
</Customer>
```

Using this approach is suitable when each occurrence of the data element is not meaningful by itself, and you do not want a separate node to appear on the mapping for each occurrence. For example, a data element with multiple occurrences may store multiple line entries that make up an address. For such cases, the "merged entries" approach might be more useful than having individual nodes for each address line.

Some configuration files supplied by default with MapForce use merged entries, and therefore, only one node appears on the mapping for multiple consecutive occurrences of a data element. If you need separate nodes for each occurrence of the data element, this can be achieved by customizing the EDI configuration. For an example, see Splitting Merged Entries into Separate Nodes[532].

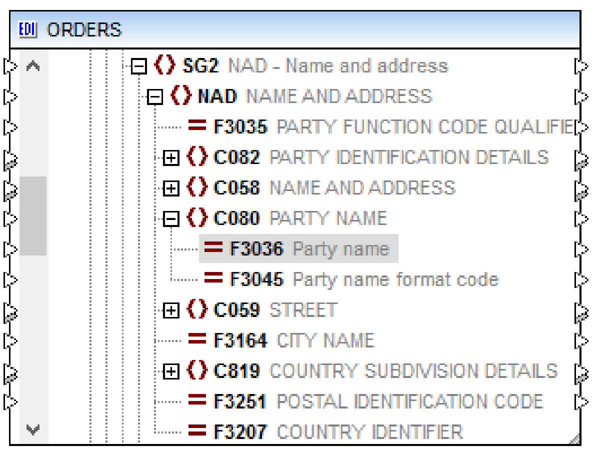
4.3.18.8 Splitting Merged Entries into Separate Nodes

Some data elements in the EDI configuration files supplied by default with MapForce are configured to display only one node on the mapping, even if the corresponding data element may occur multiple times in the source

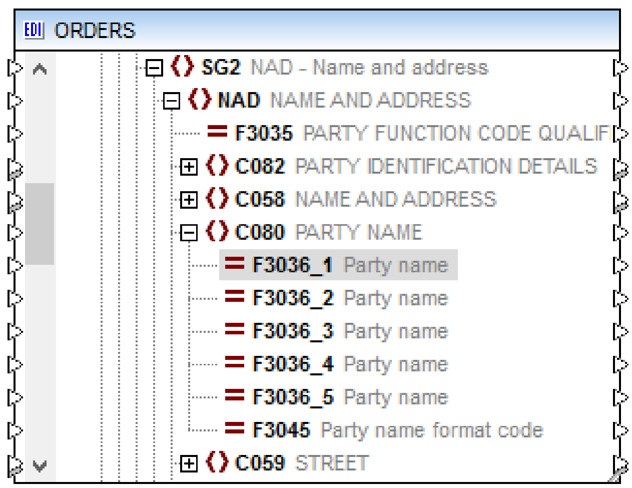
EDI file (see also Handling Multiple Consecutive Data Elements [530]). An example of such data element is F3036 of the UN/EDIFACT standard. According to the specification, F3036 can occur up to five times consecutively in the C080 ("Party name") composite (see https://www.unece.org/fileadmin/DAM/trade/untdid/d15b/trcd/trcdc080.htm).

In the default UN/EDIFACT collection of MapForce, the data element F3036 is represented through a single node on the mapping. If you draw a mapping connection from this node to a target node, then the connection would create as many items in the target as there are occurrences of F3036 in the source. (This follows the basic rule of MapForce connections, for each item in the source, create one item in the target). However, there may be situations where this is not the desired result, and you need separate nodes for each occurrence of the data element.

The goal of the example is, therefore, to create a custom UN/EDIFACT configuration which would display separate mapping nodes for each occurrence of F3036. This would enable you to map each individual occurrence of F3036 to a separate target node. The images below show the EDIFACT mapping component before and after customization.



Before customization



After customization

The steps required to split the F3036 data element into multiple mapping nodes are as follows:

1. Create a custom EDI collection using the existing UN/EDIFACT collection as starting point (see Creating a Custom EDI Collection [519]). This ensures your changes will not affect in any way the default EDI configuration.
2. Open the **EDSD.Segment** file of your custom collection in an XML editor. (If you want to customize a different EDI flavour, then edit the corresponding .segment file of that particular EDI flavour. For example, for X12, this would be the **X12.Segment** file.)
3. Find the composite C080 and remove the `mergedEntries="5"` attribute of the F3036 data element.

```
<Composite name="C080" info="PARTY NAME">
   <Data ref="F3036" mergedEntries="5"/>
   <Data ref="F3045" minOccurs="0"/>
</Composite>
```

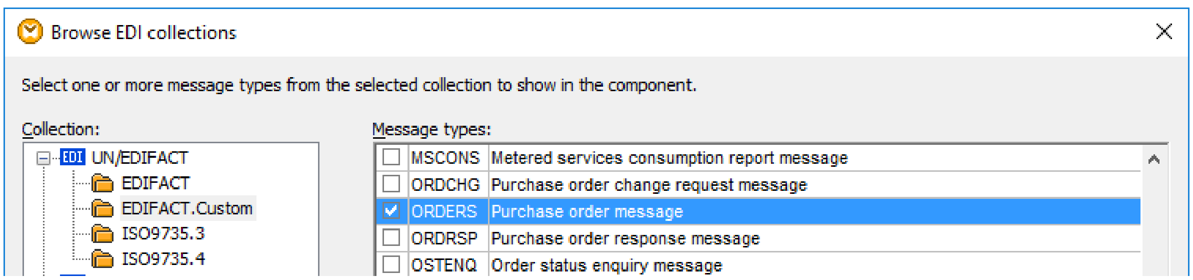
4. Enter a unique node name for each of the fields (for example, F3036_1, F3036_2, and so on). At this stage, the configuration should look as follows:

```
<Composite name="C080" info="PARTY NAME">
   <Data ref="F3036" nodeName="F3036_1"/>
   <Data ref="F3036" nodeName="F3036_2" minOccurs="0"/>
   <Data ref="F3036" nodeName="F3036_3" minOccurs="0"/>
   <Data ref="F3036" nodeName="F3036_4" minOccurs="0"/>
   <Data ref="F3036" nodeName="F3036_5" minOccurs="0"/>
   <Data ref="F3045" minOccurs="0"/>
</Composite>
```

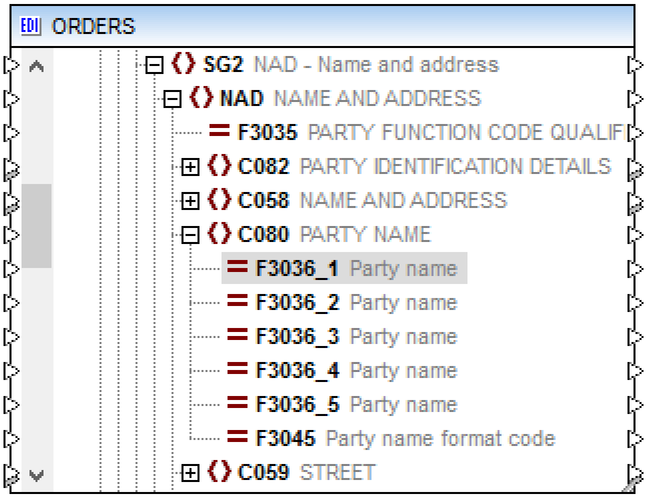
5. Add the `minOccurs="0"` attribute to all fields except for the first one. The reason is that, according to the UN/EDIFACT standard, only the first occurrence of the field is mandatory; the others are conditional.
6. Save the edited **EDSD.Segment** file.

You can now view and work with the modified EDI collection in MapForce, as follows:

1. On the **Insert** menu, click **EDI**.
2. When prompted, select the custom collection created previously (in this example, "EDIFACT.Custom").
3. Select any message type that uses the C080 segment (in this example, "ORDERS - Purchase Order Message").



The EDI component now appears modified in MapForce (each occurrence of the F3036 data element has a separate node on the mapping).



The technique described in this example can also be used to split merged entries for other data elements, regardless of the EDI flavour.

4.3.18.9 Changing Possible Code Values of Data Elements

The possible values of a data element can be changed by means of configuration files. Similar to other configuration information, you can configure the possible code values at each message level (local customization), or globally (across all messages that may use that particular data element).

> Like with any other custom configuration, to customize code values, it is recommended to create a custom EDI collection (see Creating a Custom EDI Collection 519) and apply all customization to it, rather than altering the default MapForce configuration.

The available customization options depend on the schema version used by your custom EDI collection (see EDI Configuration Schemas 523). To put it simply, a newer schema means that more advanced configuration options are available. In particular, each schema supports custom code values as follows:

- With **EDIConfig.xsd**, it is possible to change an existing .codelist file in order to add custom code values at a global level.
- With **EDIConfig4.xsd**, custom code values can be added not only at global level, but also at message level.
- With **EDIConfig5.xsd**, all of the options above are available, including the ability to define advanced configuration rules at character level (see Configuring Character-Level Validation 559). Also, data elements can be declared as having incomplete validation (see Relaxing Field Validation 553).

Global customization

To add custom code values for a data element globally, edit the .codelist file of your custom EDI collection. First, find the data element (field) whose values you need to change, and then add all the required custom values as new `<Value>` elements, as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="5"
xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
   <Include href="ISO6346.Codelist"/>
   <Codelist>
      <Field Name="F1001" Length="3" Type="Alphanumeric">
         <Title>Document name code</Title>
         <Description>Code specifying the document name.</Description>
         <Values>
            <Value Code="1">
               <Title>Certificate of analysis</Title>
               <Description>Certificate providing the values of an
analysis.</Description>
            </Value>

>>> add your custom values here...

        </Values>
      </Field>
   </Codelist>
</Config>
```

UNCL.Codelist file (UN/EDIFACT collection)

As shown in the code listing above, each `<Value>` element has a `Code` attribute which specifies the actual value code expected for this data element. The `<Title>` and `<Description>` elements provide additional descriptive information about this value code.

The code listing above also has an `Include` statement which instructs MapForce to look for additional code values defined in an external file (**ISO6346.Codelist**) which uses the same schema as the **UNCL.Codelist** file. Use this technique if you need to adapt various external files containing custom code values.

For an example, see Adapting Code Values from External Sources 555.

Local customization

To add custom code values at message level, you will need to edit the .config file corresponding to the message type. For more information, see Defining Code Values at Message Level (Inline) 558.

4.3.18.10 Example: Adding a TA1 Segment

The TA1 Segment is an optional segment and is used to acknowledge the reception of the interchange and the syntactical correctness of the envelope segments within it. The segment can be added to EDI X12 and EDI HIPAA components in MapForce by modifying the configuration files.

To include the TA1 segment in an EDI component:

1. Create a custom EDI collection using the HIPAA.X12 collection as a model (see Creating a Custom EDI Collection 519). This ensures that your changes do not affect the original default configuration.
2. Open the **Envelope.config** file of your custom EDI collection in an XML editor.

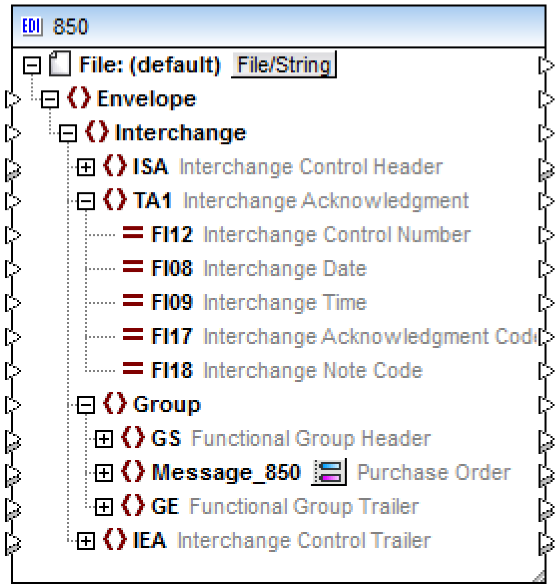
3. Paste the text **<Segment ref="TA1" minOccurs="0" maxOccurs="unbounded"/>** between the ISA and Group segments, and save the file. The file should now look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../EDIConfig.xsd" Version="3">
   <Meta>
      <Release>6020</Release>
      <Agency>X12</Agency>
   </Meta>
   <Format standard="X12"/>
   <Include href="X12.Segment"/>
   <Include href="X12.Codelist"/>
   <Include collection="EDI.Collection"/>
   <Group name="Envelope">
      <Group name="Interchange" maxOccurs="unbounded">
         &lt;Segment ref="ISA" minOccurs="0"/>
         &lt;Segment ref="TA1" minOccurs="0" maxOccurs="unbounded"/>
         <Group name="Group" maxOccurs="unbounded">
            ...
         </Group>
         &lt;Segment ref="IEA" minOccurs="0"/>
      </Group>
   </Group>
</Config>
```

The line highlighted above adds a reference to the TA1 segment which is already defined in the **X12.Segment** file. You can now view the structure of the TA1 segment in MapForce as follows:

1. On the **Insert** menu, click **EDI**.
2. On the "Browse EDI Collections" dialog box, select the custom collection created above and a message type (for example, "850 - Purchase Order").

Notice that the customized component now includes the TA1 segment. You can now draw mapping connections to and from this customized component as required.



Customized X12 EDI component with optional TA1 segment

4.3.18.11 Example: Customizing an EDIFACT message

This example shows you how to customize EDIFACT messages so that MapForce can process non-standard or changed EDIFACT formats. Note that this customization example is specifically based on version **19B** of EDIFACT; instructions may slightly vary for other EDIFACT versions.

A sample mapping which processes a custom EDIFACT file with a slightly extended CTA (Contact Information) segment of the ORDERS message is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ExtractCustomEDIFACT.mfd**. Note that, if you open this mapping before following the customization instructions below, MapForce does not validate it successfully. The reason is that the mapping references a custom EDIFACT configuration which does not exist yet on your computer (it will be created in this example).

The example mapping reads data from the **Orders-Custom.EDI** file, available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** directory. If you open this file with a text editor, notice that it uses a customized CTA segment. Specifically, line 9 contains a **Mr** entry, and line 11 contains a **Mrs** entry:

Orders-Custom.edi

```
UNB+UNOB:1+003897733:01:MFGB+PARTNER ID:ZZ:ROUTING ADDR+970101:1230+00000000000001++ORDERS++++1'
UNH+0001+ORDERS:D:19B:UN'
BGM+221+ABC123456XYZ+9'
DTM+4:202004301742PDT:303'
FTX+PUR+3++Pizza purchase order'
RFF+CT:123-456'
RFF+CR:1122'
NAD+SE+999::92++24h Pizza+Long Way+San-Francisco+CA+34424+US'
CTA+SR+:Ted Little:Mr'
NAD+BY+123::92++Nanonull, Inc.+119 Oakstreet Suite 4876+Vereno+CA+29213+US'
CTA+PD+:Michelle Butler:Mrs'
NAD+ST+123::92++Nanonull, Inc.+769 Oakstreet Suite 4876+Vereno+CA+29213+US'
TAX+9++++++1-12345-6789-0'
CUX+2:USD:9'
PCD+12:2'
```

Before you can map data from custom EDI files such as the one above, a custom EDIFACT configuration must be created. In this example, you will customize the default EDIFACT configuration so as to extend the CTA segment with a new field which would accommodate the title of the person. For convenience, the final result of the customization procedure is available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** directory, as a ZIP archive (**EDIFACT.Nanonull.zip**).

Prerequisites

To complete this example, first download the "EDIFACT Configuration Files" executable from the Altova Download Center (https://www.altova.com/mapforce/download). Make sure to choose the package applicable to your version and platform (32-bit, 64-bit) of MapForce.

The installation package contains all EDIFACT versions supported by MapForce, including version "19B" used in this example. To perform the installation, double-click the downloaded executable file and follow the on-screen instructions. After installation, new directories corresponding to the installed EDIFACT configuration versions are created in **C:\Program Files\Altova\MapForce2023\MapForceEDI**.

If you run 32-bit MapForce on a 64-bit operating system, adjust the directory path above to **C:\Program Files (x86)**.

Create the custom EDI collection

1. Create a new directory that will store your custom EDI collection, at the following path: **C:\users\<name>\Documents\Altova\MapForce2023\MapForceEDI\EDIFACT.Nanonull**. The name "EDIFACT.Nanonull" helps differentiate this custom EDI collection from the default one installed with MapForce.
2. Copy the following files from **C:\Program Files\Altova\MapForce2023\MapForceEDI\EDIFACT.2019B** to **C:\users\<name>\Documents\Altova\MapForce2023\MapForceEDI\EDIFACT.Nanonull**:

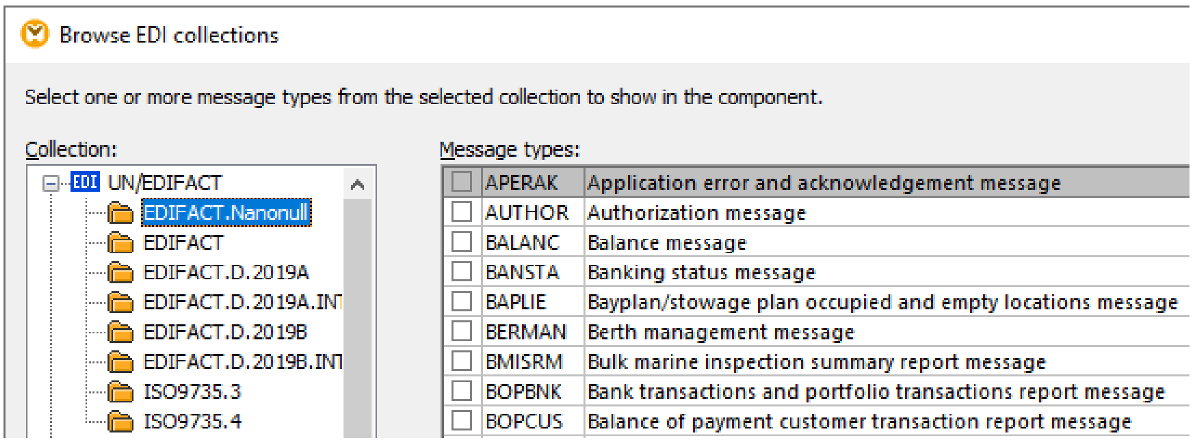
    - Admin.Segment
    - EDI.Collection
    - EDSD.Segment
    - Envelope.Config
    - ISO6346.Codelist
    - ORDERS.Config

- UNCL.Codelist

Except for **ORDERS.Config**, all these files are referenced from the **Envelope.Config** file, so they must be present in the EDI collection. For information about the purpose of each file, see EDI Configuration Files[522].

The new **EDI.Collection** file should now be accessible to MapForce. You can test this as follows:

1. Start MapForce (or restart it if it was already running).
2. On the **Insert** menu, click **EDI**. Alternatively, click the **Insert EDI** toolbar button. A dialog box opens, displaying a new collection named "EDIFACT.Nanonull".



Note: You can also create your custom EDI collection in a different directory, as described in Creating a Custom EDI Collection[519].

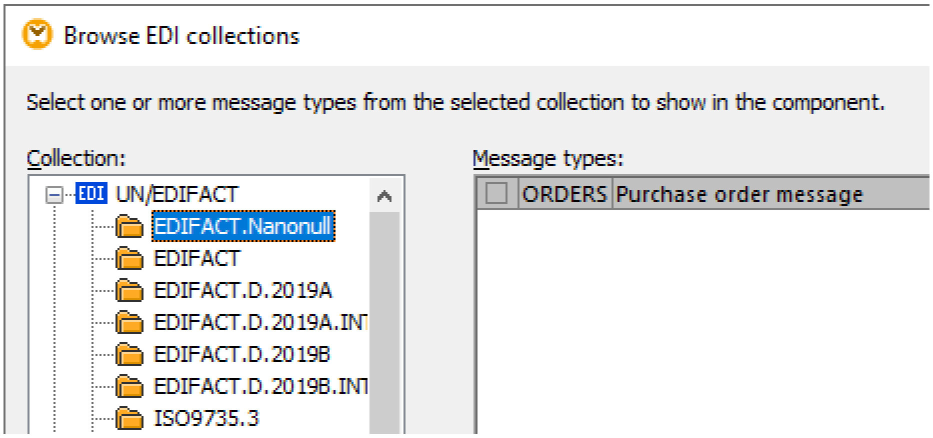
Configuring the EDI.Collection file

1. Open the **EDI.Collection** file from your custom EDI collection in an XML editor (for example, XMLSpy).
2. Remove all "Message" elements, except for the "ORDERS" message. The file should now look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages Version="3">
   <Meta>
      <Version>D</Version>
      <Release>19B</Release>
      <Agency>UN</Agency>
   </Meta>
   <Root File="Envelope.Config"/>
   <Message Type="ORDERS" File="ORDERS.Config" Description="Purchase order
message"/>
</Messages>
```

3. Save the file.

If you run the **Insert | EDI** menu command, now the collection shows only one message type: "ORDERS".



Global versus inline customization

The goal of the example is to redefine the CTA (Contact Information) segment. The CTA segment consists of one field (F3139) and one composite (C056). To store the title data (for example, "Mr" or "Mrs"), we will add to the C056 composite a new field, called X1000. There are several ways to perform the customization:

- **Globally**, by customizing the **EDSD.segment** file. All segments, in all messages that use composite C056, will contain/reference the new element.
- **Inline**, by customizing the **ORDERS.Config** file. Only the customized segment (CTA) in the current message will contain the new element.

Global customization

To make access to the new X1000 field *global*, changes have to be made only to the **EDSD.Segment** file. All segments, in all messages that use composite C056, will contain/reference the new element.

To redefine the Composite C056 globally:

1. Open the **EDSD.Segment** file in a text or XML editor, and navigate to **Config | Elements | Composite | C056**.

```
<Composite name="C056" info="CONTACT DETAILS">
  <Data ref="F3413" minOccurs="0"/>
  <Data ref="F3412" minOccurs="0"/>
</Composite>
```

2. Insert the following line in the C056 segment, under "F3412":

```
<Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
```

The `Composite` definition should now look as shown below:

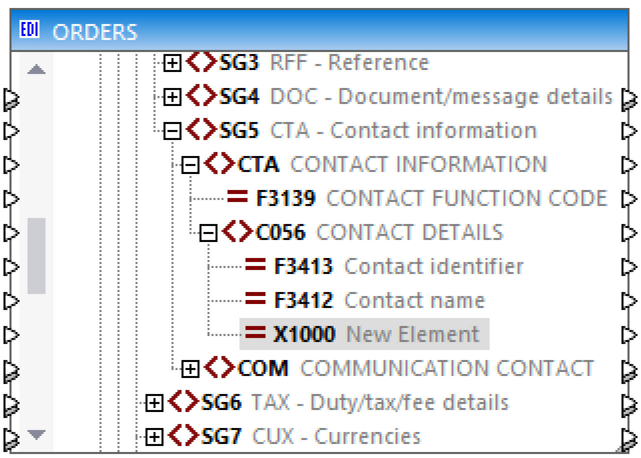
```
<Composite name="C056" info="CONTACT DETAILS">
  <Data ref="F3413" minOccurs="0"/>
```

```
  <Data ref="F3412" minOccurs="0"/>
  <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New
Element"/>
</Composite>
```

In the code listing above, the new X1000 field is defined using the "name" attribute as opposed to other fields of the segment which use the "ref" attribute. The two other fields are defined at the beginning of the file and are only referenced here.

The new X1000 field is now available to all messages that use composite C056. You can preview the new field in MapForce as follows:

1. On the **Insert** menu, click **EDI**. Alternatively, click the **Insert EDI** toolbar button.
2. Click the "EDIFACT.Nanonull" collection, and select the ORDERS message.
3. When prompted to select a source EDI file, click **Skip**. The ORDERS component is now visible in the mapping window.
4. Click the component header, press **Ctrl + F** and enter "X1000" as search text. Click **Find Next** to jump to the next occurrence of the new X1000 element (it should be under **Envelope > Interchange > Group > Message_ORDERS > SG2 > SG5 > CTA > C056**).



Inline customization

To make access to the new X1000 field *local (or inline)*, changes have to be made only to the **ORDERS.config** file. In this case, only the redefined CTA segment in the **current** message will contain/reference the new element. In other words, the CTA segment is redefined locally to contain a redefined Composite C056, with a custom new field X1000.

To redefine the Composite C056 locally:

1. Open the **ORDERS.Config** file in a text or XML editor and navigate to `Group name="SG5"` (or search for **SG5**).

```
<Group name="SG5" minOccurs="0" maxOccurs="5" info="CTA - Contact information">
   &lt;Segment ref="CTA"/&gt;
```

```
   &lt;Segment ref="COM" minOccurs="0" maxOccurs="5"/&gt;
</Group>
```

2. Replace the line `<Segment ref="CTA"/>` with the following lines:

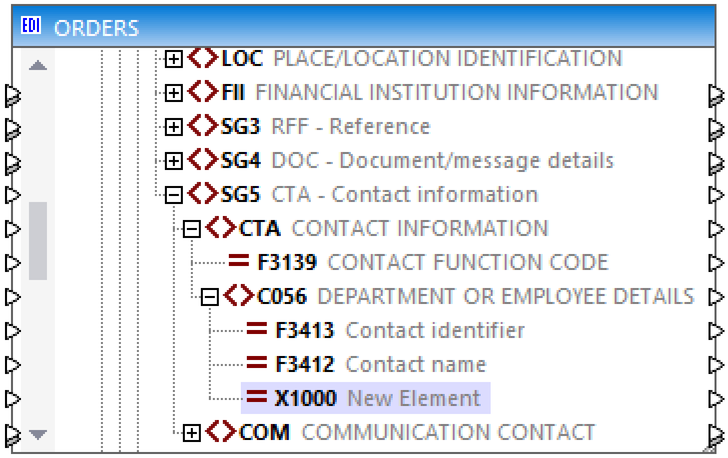
```
&lt;Segment name="CTA" id="CTA_ORDERS_SG5" info="CONTACT INFORMATION"&gt;
    <Data ref="F3139" minOccurs="0"/>
    <Composite name="C056" minOccurs="0" info="DEPARTMENT OR EMPLOYEE DETAILS">
           <Data ref="F3413" minOccurs="0"/>
           <Data ref="F3412" minOccurs="0"/>
           <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New
Element"/>
    </Composite>
&lt;/Segment&gt;
```

The `Group` definition should now look as shown below:

```
<Group name="SG5" minOccurs="0" maxOccurs="5" info="CTA - Contact information">
   &lt;Segment name="CTA" id="CTA_ORDERS_SG5" info="CONTACT INFORMATION"&gt;
    <Data ref="F3139" minOccurs="0"/>
    <Composite name="C056" minOccurs="0" info="DEPARTMENT OR EMPLOYEE DETAILS">
           <Data ref="F3413" minOccurs="0"/>
           <Data ref="F3412" minOccurs="0"/>
           <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New
Element"/>
    </Composite>
   &lt;/Segment&gt;
   &lt;Segment ref="COM" minOccurs="0" maxOccurs="5"/&gt;
</Group>
```

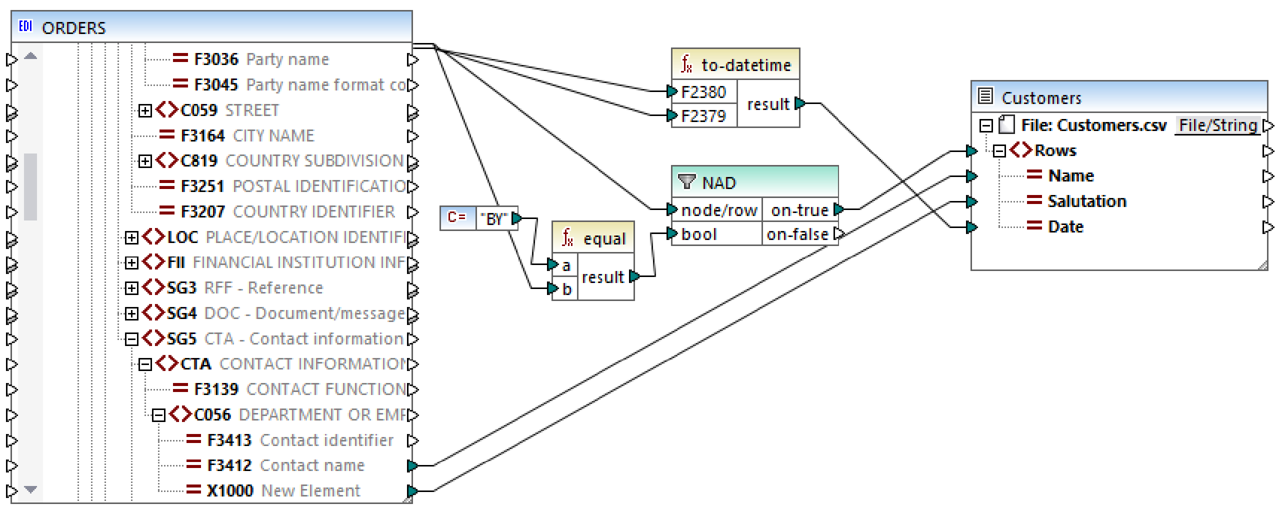
You can preview the new field in MapForce as follows:

1. On the **Insert** menu, click **EDI**. Alternatively, click the **Insert EDI** toolbar button.
2. Click the "EDIFACT.Nanonull" collection, and select the ORDERS message.
3. When prompted to select a source EDI file, click **Skip**. The ORDERS component is now visible in the mapping window.
4. Navigate to **Envelope > Interchange > Group > Message_ORDERS > SG2 > SG5 > CTA > C056**, to see the new X1000 element.

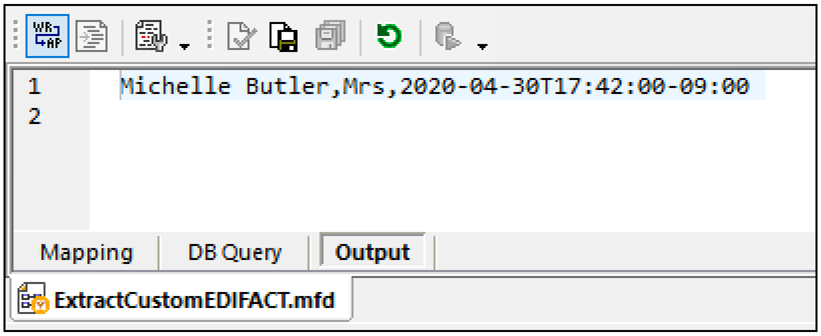


Using the customized message in mappings

If you have followed all the instructions above, you can now use the custom EDIFACT configuration in your mappings. An example mapping which uses the custom EDIFACT configuration created above is **ExtractCustomEDIFACT.mfd**, available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. The example maps the **ORDERS-Custom.EDI** file to a CSV file. The custom field that was added to the EDI structure, `X1000`, maps to the `Salutation` item in the target file.



To preview the mapping result, click the **Output** tab. As expected, the value "Mrs" extracted from the custom X1000 data element of the EDIFACT file appears in the second field of the CSV file ("Salutation").



Similar to other mapping design files in MapForce, EDI mappings can be executed outside MapForce as well. For example, you can deploy them to another server for execution with MapForce Server, or generate program code to run the mapping. For the scope of this example, if you choose to run the mapping by generating C++ code, note that a class named "CX1000Type" is generated, which is accessible from the "CC056Type" class.

4.3.18.12 Example: Customizing an ANSI X12 transaction

This example illustrates how to customize MapForce so that it can process non-standard or changed X12 formats. Note that this customization example is specifically based on version **6020** of ANSI X12; instructions may slightly vary for other X12 versions.

The example is accompanied by a sample mapping which is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ExtractCustomX12.mfd**. Note that, if you open this mapping before following the customization instructions below, MapForce does not validate it successfully. The reason is that the mapping references a custom X12 collection which does not exist yet on your computer (it will be created in this example).

The example mapping reads data from the **Orders-Custom.X12** file, available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** directory. If you open this file with a text editor, notice that it uses a customized N2 segment. Specifically, line 6 contains an additional ++**Mrs** entry:

Orders-Custom.X12

```
1  ISA+00+          +00+          +ZZ+SenderID       +ZZ+ReceiverID
2  GS+PO+SenderID+ReceiverID+20060308+182347+1+T+005011'
3  ST+850+12345'
4  BEG+00+SA+ABC123456XYZ++20200430'
5  N1+001+Nanonull, Inc.++123'
6  N2+Michelle Butler++Mrs'
7  N3+119 Oakstreet Suite 4876'
8  N4+Vereno+CA+29213'
9  PO1++1++7.2'
10 PO3+AQ+++7.2++1+US'
11 PID+F++++Pizza Pepperoni'
```

Before you can map data from custom X12 files such as the one above, a custom X12 configuration must be created. In this example, you will customize the default X12 configuration so as to extend the N2 segment with a new data element which would accommodate the title of the person. For convenience, the final result of the customization procedure is available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** directory, as a ZIP archive (**X12.Nanonull.zip**).

Prerequisites

To complete this example, first download the "X12 Configuration Files" executable from the Altova Download Center (https://www.altova.com/mapforce/download). Make sure to choose the package applicable to your version and platform (32-bit, 64-bit) of MapForce.

The installation package contains all X12 versions supported by MapForce, including version "6020" used in this example. To perform the installation, double-click the downloaded executable file and follow the on-screen instructions. After installation, new directories corresponding to the installed X12 configuration versions are created in **C:\Program Files\Altova\MapForce2023\MapForceEDI**.

If you run 32-bit MapForce on a 64-bit operating system, adjust the directory path above to **C:\Program Files (x86)**.

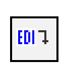
Setting up the customization files

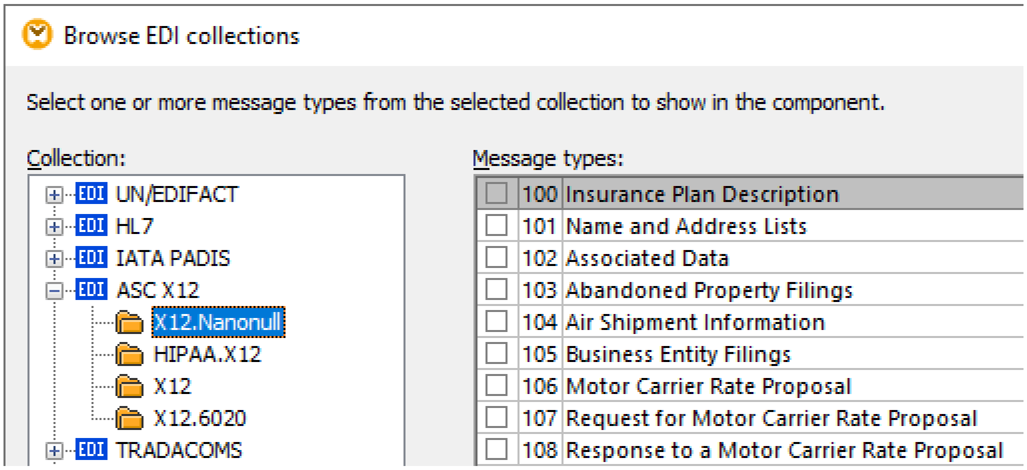
1. Create a new directory that will store your custom EDI collection, at the following path: **C:\users\<name>\Documents\Altova\MapForce2023\MapForceEDI\X12.Nanonull**. The name "X12.Nanonull" helps differentiate this custom EDI collection from the default one installed with MapForce.
2. Copy the following files from **C:\Program Files\Altova\MapForce2023\MapForceEDI\X12.6020** to **C:\users\<name>\Documents\Altova\MapForce2023\MapForceEDI\X12.Nanonull**:

    - 850.Config
    - EDI.Collection
    - Envelope.Config
    - ParserErrors.Config
    - X12.Codelist
    - X12.Segment

    For information about the purpose of each file, see EDI Configuration Files 522.

The new **EDI.Collection** file should now be accessible to MapForce. You can test this as follows:

1. Start MapForce.
2. On the **Insert** menu, click **EDI**. Alternatively, click the **Insert EDI**  toolbar button. A dialog box such as the one below opens, displaying a new collection named "X12.Nanonull".



Note: You can also create your custom EDI collection in a different directory, as described in Creating a Custom EDI Collection 519.

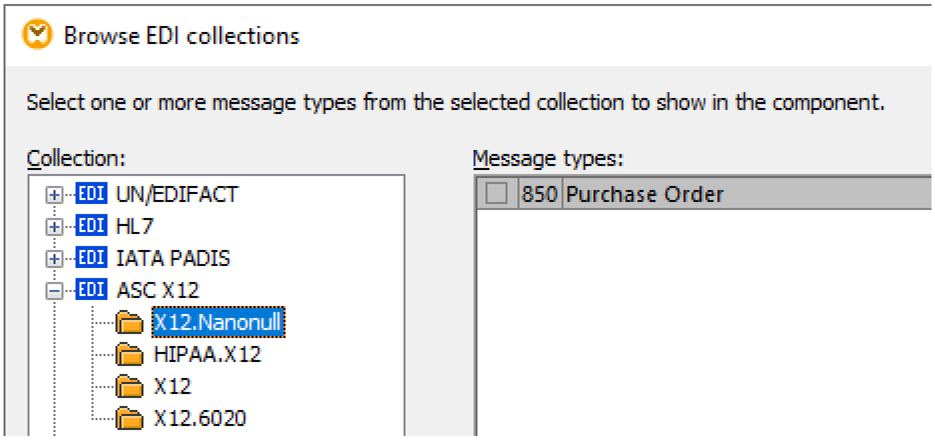
Configuring the EDI.Collection file

1. Open **EDI.Collection** file in an XML editor (for example, XMLSpy).
2. Remove all "Message" elements, except for the "Purchase Order" message. Make sure you retain the <Messages> tags, however.

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages Version="3">
   <Meta>
      <Release>6020</Release>
      <Agency>X12</Agency>
   </Meta>
   <Root File="Envelope.Config"/>
   <Message Type="850" File="850.Config" Description="Purchase Order"/>
</Messages>
```

3. Save the file.

If you run the **Insert | EDI** menu command, now the collection shows only one message type: "850 - Purchase Order".



Global versus inline customization

The goal of this example is to redefine the N2 "Additional Name Information" segment so that it contains a custom X1000 data element which would store a person's title (for example, "Mrs"). Before customization, N2 consists of one data element only, "F93 Name". There are several ways the customization can be achieved:

- **Globally**, by customizing the **X12.Segment** file. "Global" means that all segments, in all transactions that use N2, will contain/reference the new element.
- **Inline**, by customizing the **850.Config** file. "Local" means that only the customized segment (N2) in the current transaction will contain the new element.

Global customization

To make access to the new X1000 field *global*, changes have to be made only to the **X12.Segment** file. All segments, in all transactions that use N2, will contain/reference the new element.

To redefine the N2 segment globally:

1. Open the **X12.Segment** file in an XML editor and navigate to **Config | Elements | Segment name="N2"**.

```
&lt;Segment name="N2" info="Additional Name Information"&gt;
   <Data ref="F93" mergedEntries="2"/>
&lt;/Segment&gt;
```

2. Insert the following line under F93, and save the file.

```
<Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
```

The segment definition should now look as shown below:

```
&lt;Segment name="N2" info="Additional Name Information"&gt;
   <Data ref="F93" mergedEntries="2"/>
```

```
  <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New
Element"/>
&lt;/Segment&gt;
```

Notes:

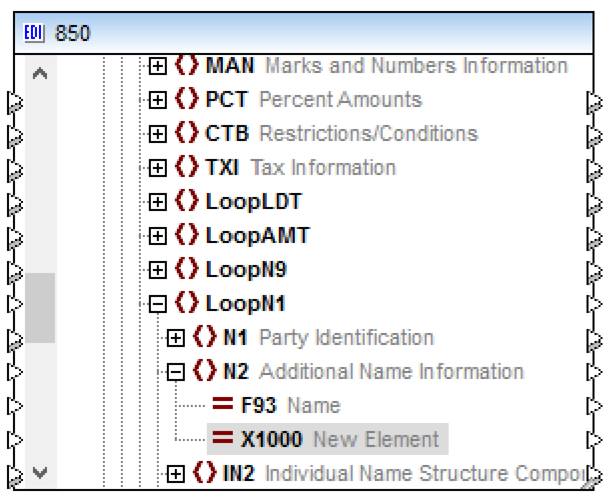
- The new X1000 field is defined using the `name` attribute as opposed to other fields of the segment which use the `ref` attribute. The F93 field is defined at the beginning of the **X12.Segment** file and is only referenced here.
- The `mergedEntries` attribute of F93 specifies that there can be two consecutive occurrences of this element with the same name. This reflects the definition of this field as it is defined in the ASC X12 specification. For this reason, in the source EDI file that is being customized here, the N2 segment looks as follows:

  ```
  N2+Michelle Butler++Mrs'
  ```

  In the line above, notice the two consecutive "+" separators between "Michelle Butler" and "Mrs". The explanation is that the N2 segment consists of a "F93" data element ("Michelle Butler"), followed by a "+" separator, followed by the second "F93" element (which is empty), followed by a "+" separator, followed by the custom "X1000" data element ("Mrs"). For more information about the `mergedEntries` attribute, see Handling Multiple Consecutive Data Elements 530.

The new X1000 field is now available to all transactions that use segment N2. You can preview the new field in MapForce as follows:

1. On the **Insert** menu, click **EDI**. Alternatively, click the **Insert EDI** toolbar button.
2. Click the "X12.Nanonull" collection, and select the "Purchase Order" message.
3. When prompted to select a source EDI file, click **Skip**. The 850 component is now visible in the mapping window.
4. Click the component header, press **Ctrl + F** and enter "X1000" as search text. Click **Find Next** to jump to the next occurrence of the new X1000 element (it should be under **Envelope > Interchange > Group > Message_850 > LoopN1 > N2**).



Local customization

If you want to make access to the new X1000 field local, then only the **850.Config** file must be edited. In this case, only the segment N2 in the "850 - Purchase Order" transaction will contain/reference the new X1000 field.

To redefine the N2 segment locally:

1. Open the **850.Config** file in a text or XML editor and navigate to `Group name="LoopN1"` (or search for **LoopN1**).

```
<Group name="LoopN1" maxOccurs="10" minOccurs="0">
   &lt;Segment ref="N1"/&gt;
   &lt;Segment ref="N2" minOccurs="0" maxOccurs="2"/&gt;

   ... ...

</Group>
```

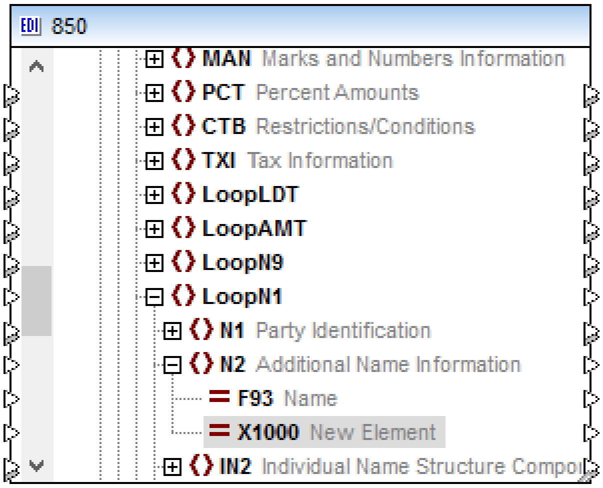
2. Replace the line `<Segment ref="N2" minOccurs="0" maxOccurs="2"/>` with the following lines:

```
&lt;Segment name="N2" info="Additional Name Information"&gt;
   <Data ref="F93" mergedEntries="2"/>
   <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New
Element"/>
&lt;/Segment&gt;
```

The `Group` definition should now look as shown below:

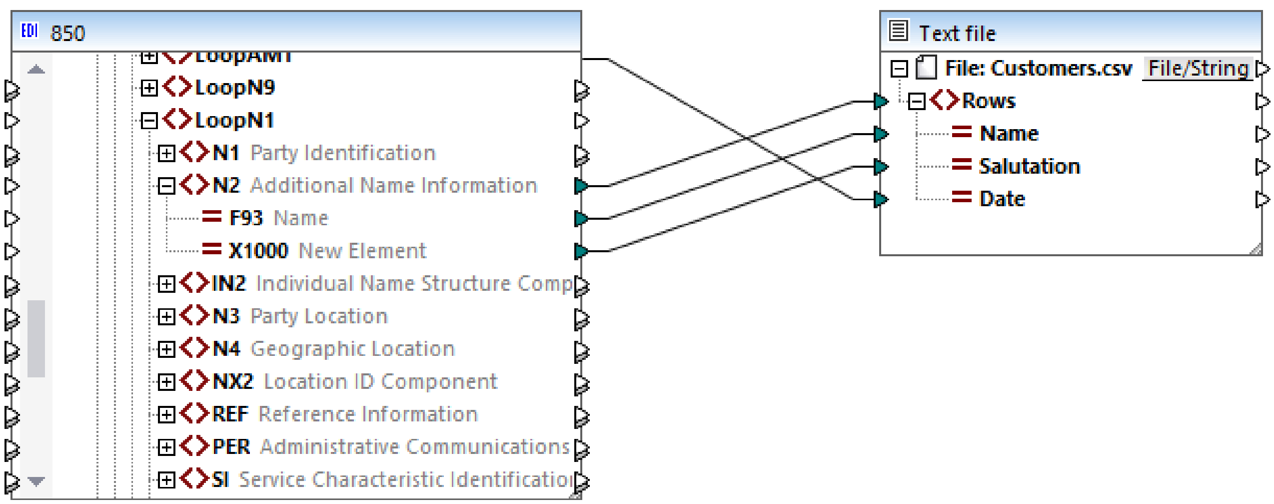
```
<Group name="LoopN1" maxOccurs="10" minOccurs="0">
   &lt;Segment ref="N1"/&gt;
   &lt;Segment name="N2" info="Additional Name Information"&gt;
      <Data ref="F93" mergedEntries="2"/>
      <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New
Element"/>
   &lt;/Segment&gt;

   ... ...

</Group>
```

You can now preview the new field in MapForce, using the same instructions as those described above for global customization.



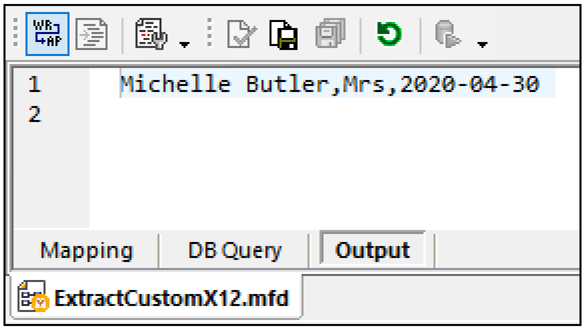
Using the customized message in mappings

If you have followed all the instructions above, you can now use the custom X12 configuration in your mappings. An example mapping which uses the custom EDIFACT configuration created above is **ExtractCustomX12.mfd**, available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. The example maps the **ORDERS-Custom.X12** file to a CSV file. The custom `X1000` field that was added in the steps above maps to the `Salutation` item in the target file.



ExtractCustomX12.mfd

To preview the mapping result, click the **Output** tab. As expected, the `Salutation` target item is populated with the value "Mrs" extracted from the custom X1000 data element of the X12 file.



4.3.19 Customizing EDI Validation

This section illustrates how validation of EDI messages can be customized in MapForce so as to achieve the following goals:

- Relax validation of fields where validation rules are not completely enforced by the standard and may be extended by means of external standards.

  For example, there are EDI data elements that may accept code values specified in external ISO standards or UN/ECE recommendations, in addition to predefined code values. In this case, when processing such EDI fields, you can configure MapForce to return a warning (not a validation error) when an external code is encountered, and resume the mapping process. For further information, see Relaxing Field Validation[553].

- Perform full validation of fields such as the ones described above, by adapting custom configuration files to MapForce.

  For example, to validate the Data Element 4405 of the UN/EDIFACT D.16A standard, you may add a custom configuration file which contains the validation rules as defined by the UN/ECE Recommendation 24 ("Trade and Transport Status Codes". The code values defined in UN/ECE Recommendation 24 are external to the EDIFACT standard (and thus are not part of the default set of EDIFACT configuration files supplied with MapForce). For further information, see Adapting Code Values from External Sources[555].

- Perform validation globally for all messages where a field is used or locally for a particular message.

  It is possible to configure validation to apply globally or locally. Global validation affects all occurrences of the field in all messages. Local (or "inline") validation applies only to a specific message. For further information, see Defining Custom Code Values at Message Level (Inline)[558].

- Define advanced validation rules for fields that require validation not only at field level, but also at character level.

  Certain messages, such as the VERMAS (Verification of Mass) message of UN/EDIFACT D.16A, contain fields that require advanced validation rules (such as Data Element 8155). This field may accept code values specified in ISO 6346. To validate fields such as ISO 6346, MapForce can be customized, by means of configuration files, to check if specific characters occur at specific positions within the field. For further information, see Configuring Character-Level Validation[559]. (Note that ISO

6346 is mentioned here only as an example, the MapForce configuration files already include validation rules for ISO 6346.)

4.3.19.1 Relaxing Field Validation

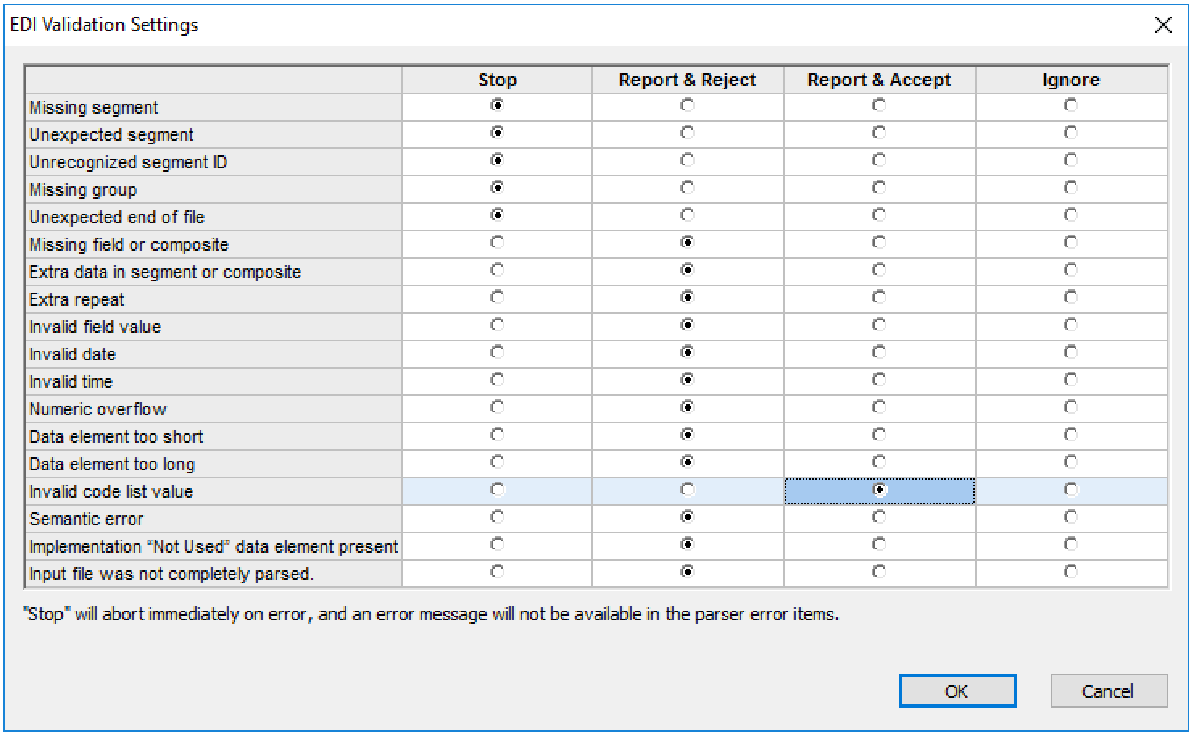
There are cases when a standard does not completely enforce the code values that a field may have, allowing code values defined by external standards or recommendations. Taking EDIFACT D16A as example, several fields allow code values defined in external ISO standards, or in UN/ECE recommendations. For example, Data Element 4405 defines 146 code values, but also allows use of external UN/ECE Recommendation 24, which specifies 346 code values.

In MapForce, the possible code values for a field are validated against a configuration file having the .codelist extension (hereafter referred to as the ".codelist" file). When MapForce processes an EDI file, it normally produces a validation error if a field does not have any of the code values specified in the .codelist file for that EDI flavour. (This assumes you have not changed the default validation options so as to silently ignore invalid field values, see EDI component validation 454.) The .codelist files available for each EDI flavour are in the **C:\Program Files\Altova\MapForce2023\MapForceEDI** directory. For example, the **UNCL.codelist** file available in **C:\Program Files\Altova\MapForce2023\MapForceEDI\EDIFACT** stores possible code values for fields in UN/EDIFACT fields.

The default .codelist file includes all possible code values for a field, according to the respective EDI standard. When code values are not explicitly defined by the standard but enforced by external standards, validation is considered to be incomplete. Namely, when the EDI standard does not fully enforce the list of code values for a particular field (such as Data Element 4405 of UN/EDIFACT D16A), processing the field during the mapping execution would normally raise validation errors. Specifically, MapForce (or the execution engine running the EDI mapping, be it MapForce Server, or a C#, C++, or Java program) reports an error in the format: **{Value} is not a legal value for field {Field}**. This error means that the field value was not found in the list of existing code list values defined in the .codelist file.

To prevent the mapping process from being stopped when errors such as above are encountered, you can do the following:

1. Change the validation behavior from EDI Validation Settings (for example, select the check box **Report & Accept** next to "Invalid code list value" on the EDI Validation Settings dialog box. (To configure EDI validation settings, click the **Validation** button on the EDI Component Settings 450 dialog box.)



2. Relax field validation from configuration files, as shown below.

MapForce provides a mechanism by which a field can be marked as having "incomplete" validation rules. This ensures that MapForce (or the execution engine running the EDI mapping, be it MapForce Server, or a C#, C++, or Java program) does not produce an error message when the field is processed; instead, a warning is displayed, and the mapping process resumes. In other words, "incomplete" validation means that only the code values defined in the MapForce default .codelist file should be considered for validation, not those defined externally.

To allow for flexibility without adding too much complexity, the default .codelist configuration files supplied with MapForce treat certain fields as having incomplete validation.

By default, MapForce produces warnings (not errors) if any of the following fields contain code values that are defined in external specifications:

- Data Element 4053 of UN/EDIFACT D.16A
- Data Element 4277 of UN/EDIFACT D.16A
- Data Element 4405 of UN/EDIFACT D.16A
- Data Element 8179 of UN/EDIFACT D.16A

The warning is produced so as to make you aware that validation of this field is incomplete. Incomplete validation may result in incorrect data being produced by the mapping. To prevent this from happening, it is recommended to perform full field validation, by means of custom .codelist files to MapForce. Custom .codelist files let you define external code values in a format understood by MapForce and perform full validation of the field. For more information, see Adapting Code Values from External Sources 555.

Note: Data Element 8155 also relies for validation on an external standard (the ISO 6346 standard); however, for this field MapForce provides full validation by means of a separate .codelist file, see Adapting Code Values from External Sources 555).

If necessary, you can additionally customize other fields to be treated as having incomplete validation rules. To instruct MapForce that a field has incomplete validation rules, do the following:

1. Create a custom EDI Collection that will store your changes to the validation rules. It is not recommended modify the EDI Collections installed by default with MapForce. For more information about this step, see Creating a Custom EDI Collection 519.
2. Locate the .codelist file in your custom EDI Collection, and open it with an XML editor such as XMLSpy.
3. Find the <Field> element corresponding to the required data element, and add the attribute `Incomplete="1"` to it (highlighted below), for example:

```
<Field Name="F8179" Length="8" Type="Alphanumeric" Incomplete="1">
   <Title>Transport means description code</Title>
   <Description>Code specifying the means of transport.</Description>
   <Note>1 Use UN/ECE recommendation 28.
</Note>
   <Values>
      <Value Code="1">
         <Title>Box tricycle</Title>
         <Description>Tricycle with a basket for delivering goods.</Description>
      </Value>
   </Values>
</Field>
```

If your processing requirements demand that an error should be raised instead of a warning for any fields that were already marked as "incomplete", change the attribute `Incomplete` to value "0" instead of "1". This way, field validation will be enforced strictly, and errors (not warnings) will occur if the field value is not part of the set of values defined in the .codelist file.

4.3.19.2 Adapting Code Values from External Sources

In the EDI configuration files supplied by default with MapForce, some data elements are configured to have incomplete validation, as explained in Relaxing Field Validation 553. This is due to the fact that the possible code values for these data elements rely on specifications which are external to the EDI standard itself.

When incomplete (or relaxed) validation of a field is not acceptable or desirable, it is possible to configure MapForce to perform full field validation. This approach requires adding custom .codelist files in addition to the default ones already available in MapForce. The role of the custom .codelist files is to supply any code values that are not defined by default in the EDI standard and consequently are not present in MapForce configuration files. This usually applies to code values that are defined in external standards or recommendations.

Note that the custom .codelist file will require an entry for every possible code value that is to be validated (which may be hundreds of records). An example is the Data Element 4405 of the UN/EDIFACT D16A. In addition to the code values defined in the UN/EDIFACT standard, this field also allows code values that are specified in the UN/ECE Recommendation 24 ("Trade and Transport Status Codes", see https://www.unece.org/fileadmin/DAM/cefact/recommendations/rec24/rec24_ecetrd258e.pdf) which amount to 346 codes. In order to accommodate all these external code values, a custom .codelist file must be created which contains an entry for each of the 346 codes present in the recommendation. Therefore, if that is acceptable by your particular business case, using incomplete validation is a simpler solution (see Relaxing Field Validation 553). Be aware that incomplete validation involves the risk of allowing invalid data to be produced in the mapping output, so full field validation is recommended instead.

To perform full field validation of fields whose code values are defined in an external standard, take the following steps:

1. Create a custom EDI Collection that will store your changes to the default validation rules. For more information about this step, see Creating a Custom EDI Collection 519.

> Although it is possible to define external code lists by editing the existing .codelist file supplied with MapForce (in this example, **UNCL.codelist**), this is not recommended. A good practice is to always define external code lists in custom files and avoid any changes to the default MapForce configuration files.

2. In your custom EDI Collection's directory, find the .codelist file to be used as template. For example, if adding a custom .codelist file to validate UN/EDIFACT messages, use as template the **UNCL.Codelist** file.
3. Rename the template .codelist file so as to obtain a new .codelist file with a short and descriptive filename. This file will be referred from other MapForce configuration files. For the scope of this example, the custom .codelist file will be called **UNECE24.codelist**.
4. Using an XML editor such as XMLSpy, delete from the new file all content included between the `<Codelist>` and `</Codelist>` tags. The file should now look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="5"
xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
   <Codelist>

...

   </Codelist>
</Config>
```

5. Add a new `<Field>` structure under `<Codelist>`, as shown below. In this example, we assume to be adding external code values for Data Element 4405 of the UN/EDIFACT. Otherwise, replace the field attributes as required by your particular case. Remember that you can find the correct name, length and type of each field in the default .codelist file supplied by MapForce (in this example, **UNCL.Codelist**).

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="5"
xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
   <Codelist>
      <Field Name="F4405" Length="3" Type="Alphanumeric">
         <Title>Trade and transport status codes</Title>
```

```
            <Description>Code specifying a status.</Description>
            <Values>

...

            <Values>
         </Field>
      </Codelist>
</Config>
```

6. Under <Values>, add a new <Value> for each external code, as shown below. In this example, we are adding the first two codes defined in the UN/ECE Recommendation 24 ("Trade and Transport Status Codes", see https://www.unece.org/fileadmin/DAM/cefact/recommendations/rec24/rec24_ecetrd258e.pdf).

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="5"
xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
   <Codelist>
      <Field Name="F4405" Length="3" Type="Alphanumeric">
         <Title>Trade and transport status codes</Title>
         <Description>Code specifying a status.</Description>
         <Values>
            <Value Code="1">
               <Title>Arrival, completed</Title>
               <Description>The goods/consignment/equipment/means of transport has
arrived.</Description>
            </Value>
            <Value Code="2">
               <Title>Loading, authorized</Title>
               <Description>Authorization to load has been given.</Description>
            </Value>
         <Values>
      </Field>
   </Codelist>
</Config>
```

At this stage, the custom .codelist file is ready but not yet bound to MapForce. Just be aware that, for the scope of the example, we have only added two external codes ("1" and "2"); however, in a real-life scenario the custom .codelist file is expected to include all code values specified by the external standard.

Finally, before the custom .codelist file can be recognized by MapForce, the following must also be done:

1. Since now the Data Element 4405 does no longer have incomplete validation, MapForce must be instructed to apply full validation. Namely, the `Incomplete="1"` attribute must be modified in the **UNCL.codelist** file for this field (the attribute must either be removed, or set to `Incomplete="0"`). Therefore, modify the **UNCL.codelist** file as shown below:

```
...
```

```
<Field Name="F4405" Length="3" Type="Alphanumeric" Incomplete="0">
         <Title>Status description code</Title>
         <Description>Code specifying a status.</Description>
         <Note>1 For transport status, use UN/ECE Recommendation 24.</Note>

...
```

2. Modify the **UNCL.codelist** file to add a reference to the newly created **UNECE24.codelist** file, as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="5"
xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
   <Include href="UNECE24.codelist"/>
   <Codelist>

....
```

You have now finished adapting MapForce configuration files to validate Field 4405 from code values defined in a custom file. This approach can also be used for other EDI flavours to achieve full field validation based on some external standard.

Note that the instructions above cause the field validation to be global, for any segment that may be using Field 4405. In some cases, you may want the custom code lists to apply only to certain messages (in other words, the configuration should not apply globally for all messages, but inline, or locally, for a particular message only). For more information about using the "inline" approach, see Defining Code Values at Message Level (Inline) 558.

4.3.19.3 Defining Code Values at Message Level (Inline)

In a previous topic, Adapting Code Values from External Sources 555, you have seen how to configure field validation through custom code lists *globally* (it applies for any message that may be using a particular field). In the event that you need validation to apply only to a specific message, it is possible to add custom code lists inline (locally for a particular message). When you need to do this, follow the steps below:

1. If you haven't done that already, create a custom EDI Collection, see Creating a Custom EDI Collection 519.
2. In the directory of your custom EDI collection, find the **{Message}.config** file corresponding to the message of interest, and open in an XML Editor.
3. Find the `<Data>` element under the required `<Segment>`, and add the custom code values as children of `<Values>`, for example:

```
<Group name="Message_270-B1" info="Health Care Eligibility Benefit Inquiry"
maxOccurs="unbounded">
      &lt;Segment name="ST" info="Transaction Set Header">
        <Condition path="F1705" value="005010X279A1" />
        <Data ref="F143" info="Transaction Set Identifier Code">
```

```
          <Values>
            <Value Code="270" />

[>>>> custom values can be added here...]

          </Values>
        </Data>
        <Data ref="F329" info="Transaction Set Control Number" />
        <Data ref="F1705" info="Implementation Convention Reference" />
      &lt;/Segment&gt;
```

Note the following:

- When a field has both locally-defined values (such as those in the code listing above) and globally-defined code values (in a .codelist file, as shown in Adapting Code Values from External Sources 555), then make sure that local code values are also defined as global code values. Otherwise, a validation error*** will be raised: **{Value} is not a legal value for field {Field}**.
- Inline (local) validation can be used if you require validation at field level (that is, the possible values that a field can take can be expressed as a list). Inline validation cannot be used for those fields that require complex validation rules at character level. When the field needs complex validation rules at character level (such as those defined in ISO 6346 applicable to Data Element 8155 of the UN/EDIFACT standard), add the code values globally, as shown in Adapting Code Values from External Sources 555. You can then configure validation as shown in Configuring Character-Level Validation 559.

*** Note that the actual validation outcome takes into account the settings configured for the rule "Invalid code list value" in the EDI Validation Settings dialog box. (To open this dialog box, click the **Validation** button on the EDI Component Settings 450 dialog box).

4.3.19.4 Configuring Character-Level Validation

In most cases, validation of EDI fields means comparing, at mapping runtime, the field value with a list of possible values defined in a MapForce configuration file. For example, if the field value matches one the possible values defined in the configuration file, it is considered valid.

However, certain fields might need more advanced validation rules, meaning that their validation logic cannot be expressed as a simple list of possible values. An example of a field which requires complex validation is Data Element 8155 used in the VERMAS (Verification of Mass) message of UN/EDIFACT D16A. This field accepts not only code values defined in the UN/EDIFACT standard, but also external codes specified by the ISO 6346 standard. The ISO 6346 standard, in its turn, defines validation rules at character-level, namely:

- First character can be one of these: 1, 2, 3, 4, A, B, C, D, E, F, G, H, K, L, M, N, P
- Second character can be one of these: 0, 2, 4, 5, 6, 8, 9, C, D, E, F, L, M, N, P
- Third and fourth character can be: G0, G1, G2, G3, etc. (values are too many to be listed here).

It is possible to adapt validation logic such as the one above to MapForce as well. In this case, MapForce will validate Data Element 8155 not only according to the UN/EDIFACT code values, but also taking into consideration the rules above applicable to the ISO 6346 standard.

From a MapForce configuration perspective, the rules expressed by the three bullets above can be regarded as three separate lists of values. The first list is one character long, the second list is one character long, and the third list is two characters long. All three lists are finite and their values are known. This information makes it possible to validate the value of the data element by means of MapForce configuration.

To see an example of how character-level validation must be configured, open the **ISO6346.Codelist** file from the **C:\Program Files\Altova\MapForce2023\MapForceEDI\EDIFACT** directory. This file supplied by default with MapForce is configured to perform validation of Data Element 8155 according to ISO 6346 rules stated above.

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="5"
xsi:noNamespaceSchemaLocation="../EDIConfig5.xsd">
   <Codelist>
      <Field Name="F8155" Type="Alphanumeric" ListName="ISO6346" Length="1" Offset="1">
         <Title>ISO 6346</Title>
         <Description>Code specifying seagoing container type.</Description>
         <Values>
            <Value Code="1"/>

            .............

            <Value Code="N"/>
         </Values>
      </Field>
      <Field Name="F8155" Type="Alphanumeric" ListName="ISO6346" Length="1" Offset="2">
         <Title>ISO 6346</Title>
         <Description>Code specifying seagoing container type.</Description>
         <Values>
            <Value Code="0"/>

            .............

            <Value Code="F"/>
         </Values>
      </Field>
      <Field Name="F8155" Type="Alphanumeric" ListName="ISO6346" Length="2" Offset="3">
         <Title>ISO 6346</Title>
         <Description>Code specifying seagoing container type.</Description>
         <Values>
            <Value Code="G0"/>

            .............

            <Value Code="A0"/>
         </Values>
      </Field>
   </Codelist>
</Config>
```

ISO6346.Codelist configuration file

Note: Some of the values from the sample code listing above were removed, for simplicity.

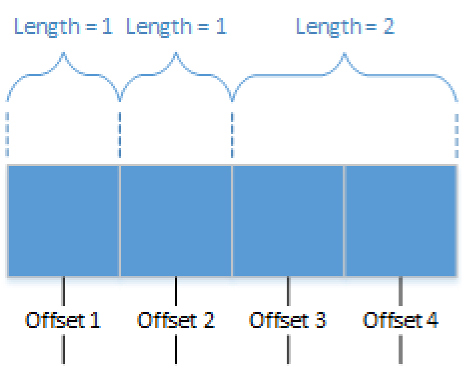
In the code listing above, notice that, the `<Field>` element is defined three times. This makes it possible to validate the field content at character level. Namely, each occurrence of `<Field>` validates one of the rules stated in the bullets above at a time. To achieve this, each `<Field>` has several attributes (highlighted in the code listing above):

- `Name` - This attribute indicates that the list of values being validated belongs to the same field. In other words, it binds all three lists to the same field (in this case, "F8155").
- `ListName` - This attribute gives a name to a list of values. Lists of values with the same name are grouped into one code list. Notice that in the example above all three lists have the same `ListName="ISO6346"`. This is needed specifically for character level validation. The actual characters to be validated are specified by the `Length` and `Offset` attributes of each list in the group, as explained below.
- `Length` - This attribute specifies the number of characters to be validated (the starting position of characters in the field is defined by the `Offset` attribute). Notice that character length varies for each `<Field>`: the first and the second list of values is one character long, while the third list of values contains two characters.
- `Offset` - This attribute specifies the position at which MapForce (or the external execution engine) must start reading the characters from the data field which is being validated. The first character has offset 1, the second character has offset 2, and so on. The actual number of characters is defined by the `Length` attribute.

> If the `Length` attribute is defined, the `Offset` attribute must be defined as well. Also, the value of both `Length` and `Offset` must be greater than zero.

In the code listing above, the following Length-Offset combinations were defined:

- `Length="1"`, `Offset="1"` - this combination will validate one character of the field, starting at offset 1
- `Length="1"`, `Offset="2"` - this combination will validate one character of the field, starting at offset 2
- `Length="2"`, `Offset="3"` - this combination will validate two characters of the field, starting at offset 3



Each Length-Offset combination, combined with the code values defined under `<Values>`, represents a validation condition at character level. When the mapping runs, all character-level conditions must be satisfied (because all have the same `ListName` attribute value) for field validation to be successful. In other words, when validating a message that includes Data Element 8155, MapForce will apply the following logic:

- The field characters are validated against the list of code values defined in the configuration file (in this case, **ISO6346.Codelist**) for a matching Length-Offset combination. For example, the first character of the field must be one of the following: 1, 2, 3, 4, B, C, G, H, L, M, N.
- The field value must satisfy all conditions at character-level (in this case, three conditions must be satisfied) in order for validation to be successful.
- If any of the conditions is not satisfied, the field value is not valid, and a validation error is produced in MapForce (or in the execution engine running the EDI mapping (such as MapForce Server, or a C#, C++, or Java program).
- A field may have more than one list of possible code values. In ordered to be considered valid, the field value must be present in at least one of the lists.

4.4 CSV and Text Files

MapForce includes support for mapping data to or from text-based file formats such as CSV (comma-separated values) and FLF (Fixed-Length Field) text files. For example, you can create data transformations such as:

- XML schema to/from flat file formats
- Database to/from flat file formats
- UN/EDIFACT and ANSI X12 to/from flat file formats or databases

An FLF is a common text format where data is conventionally separated into fields which have a fixed length (for example, the first 5 characters of every row represent a transaction ID, and the next 20 characters represent a transaction description).

Note that, in case of CSV, your files can have as delimiter not only commas, but also tabs, semicolons, spaces, or any other custom values.

In addition to CSV and FLF files, mapping to or from text files with more complex or custom structures is possible using MapForce FlexText (this module is available in MapForce Enterprise Edition). FlexText essentially enables you to define the structure of your custom text data (using a so-called "FlexText template"), for the purpose of mapping it to other formats. For more information, see MapForce FlexText 658.

Mapping data to or from text files is supported in any one of the following languages: Java, C#, C++, or BUILT-IN.

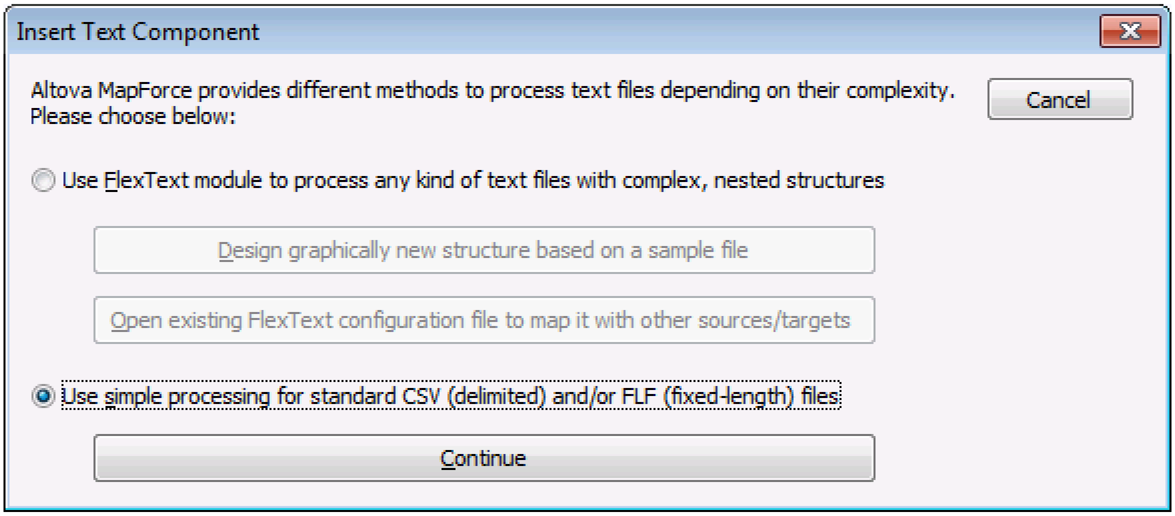
There are two ways that mapped flat file data can be generated:

- By clicking the Output tab which generates a preview using the Built-in execution engine. You can also save the mapping result by selecting the menu option **Output | Save output file**, or clicking the icon.
- By selecting **File | Generate code in | Java, C#, or C++** , and then compiling and executing the generated code.

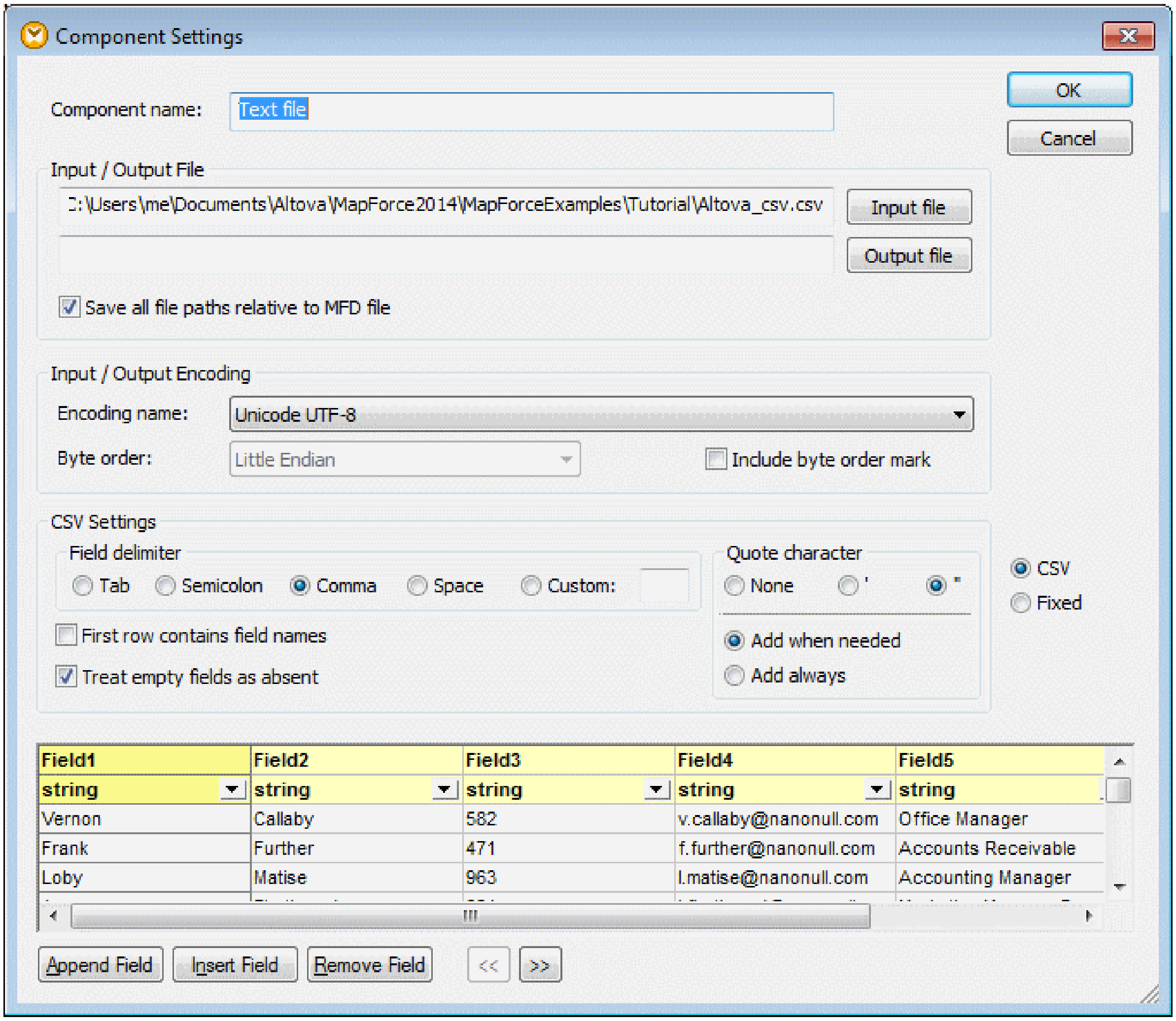
4.4.1 Example: Mapping CSV Files to XML

The goal of this example is to create a mapping which reads data from a simple CSV file and writes it to an XML file. The files used in the example are available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder.

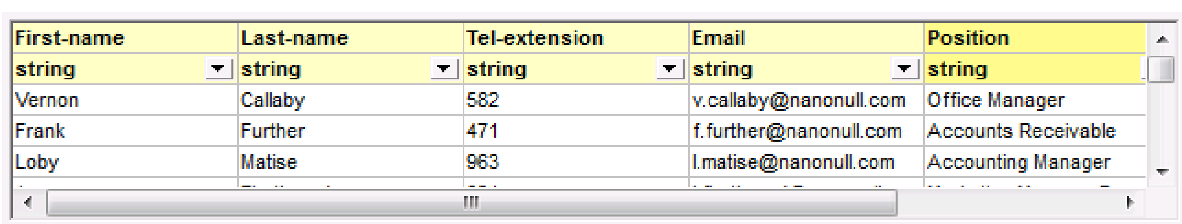
1. Select one of the following as transformation language: Java, C#, C++, or BUILT-IN.
2. Add a Text file component to the mapping area (on the **Insert** menu, click **Text File**, or click the **Insert Text file** toolbar button ( ).



3. Select the **Use simple processing** ... option, and then click **Continue**.
4. On the Component Settings dialog box, click **Input file** and browse for the **Altova_csv.csv** file. The file contents are now visible in the lower part of the dialog box. Note that only the first 20 rows of the text file are displayed when in preview mode.

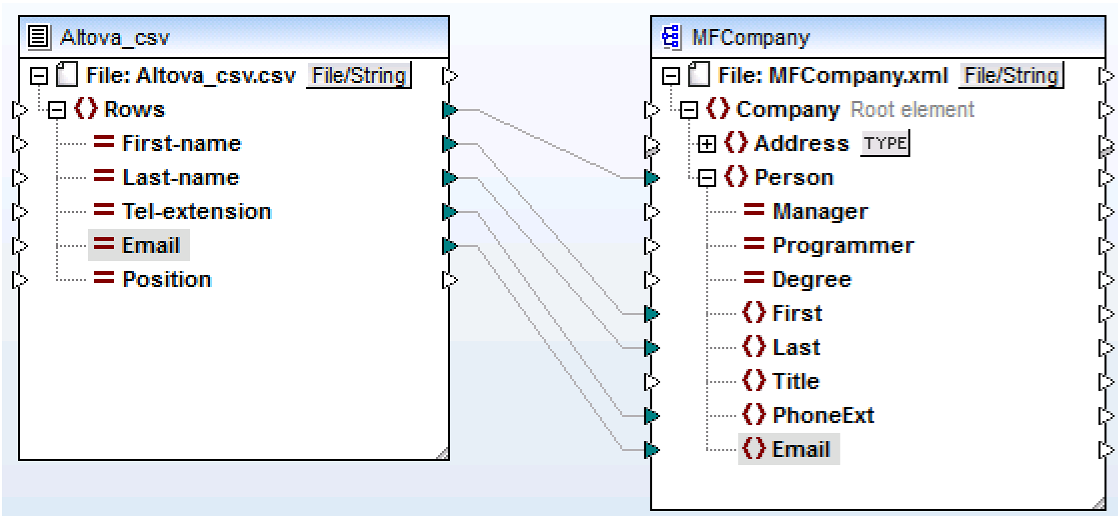


5. Click inside the **Field1** header and change the text to First-name. Do the same for all the other fields, as follows: Field 2 => Last-name, Field 3 =>Tel-extension, Field 4 => Email, Field 5 => Position. TIP: Press the **Tab** key to cycle through all the fields: header1, header2 etc.



6. Click OK.
7. When prompted to change the component name, click "Change component name". The CSV component is now visible in the mapping.
8. Add **MFCompany.xsd** as the target XML component of the mapping (on the **Insert** menu, click **XML/Schema file**).
9. Click **Skip** when prompted to supply a sample XML file, and select `Company` as the root element.
10. Map the corresponding items of both components, making sure to map the **Rows** item to the **Person** item in the schema target.

The connector from the `Rows` item in the CSV component to the `Person` item in the schema is essential, as it defines which elements will be iterated through. That is, for each row in the CSV file, a new `Person` element will be created in the XML output file.



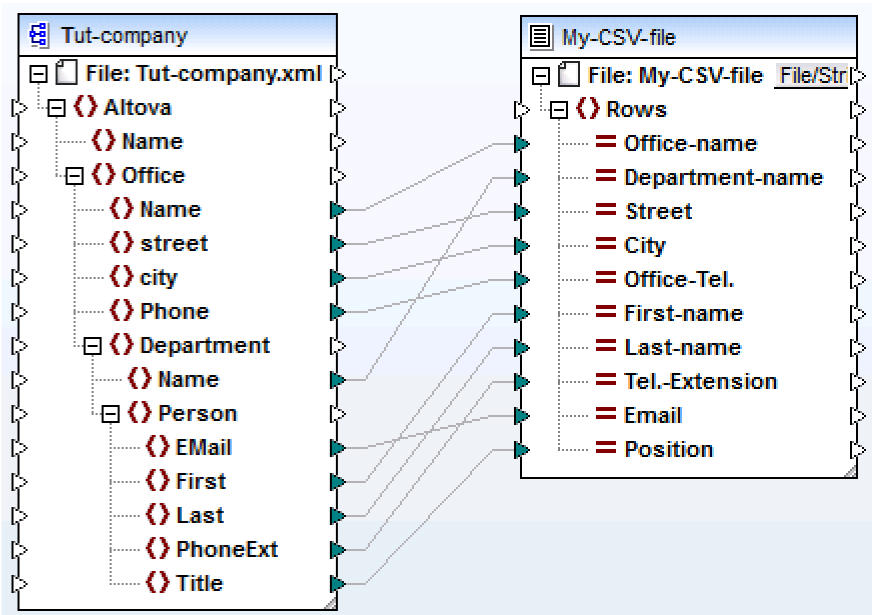
11. Click the Output tab to see the result.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
  <Person Manager="true">
    <First>Vernon</First>
    <Last>Callaby</Last>
    <PhoneExt>582</PhoneExt>
    <Email>v.callaby@nanonull.com</Email>
  </Person>
  <Person Manager="true">
    <First>Frank</First>
    <Last>Further</Last>
    <PhoneExt>471</PhoneExt>
    <Email>f.further@nanonull.com</Email>
  </Person>
  <Person Manager="true">
```

The data from the CSV file is now successfully mapped to an XML file.

4.4.2 Example: Iterating Through Items

This example illustrates how to create iterations (multiple rows) in a target CSV file. The mapping design file accompanying this example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Tut-xml2csv.mfd**.

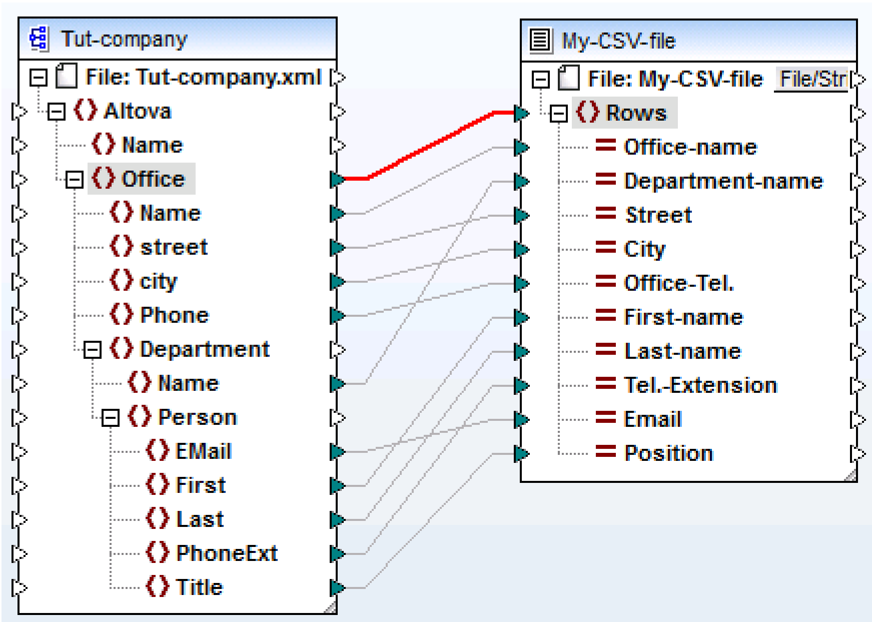


Tut-xml2csv.mfd

This mapping has been intentionally created as incomplete. If you attempt to validate the example file using the menu command **File | Validate Mapping**, you will notice that validation warnings occur. Also, if you preview the mapping output, a single row is produced, which may or may not be your intended goal.

Let's assume that your goal is to create multiple rows in the CSV file from a sequence of items in the XML file. You can achieve this by drawing a connection to the `Rows` item of the target CSV file.

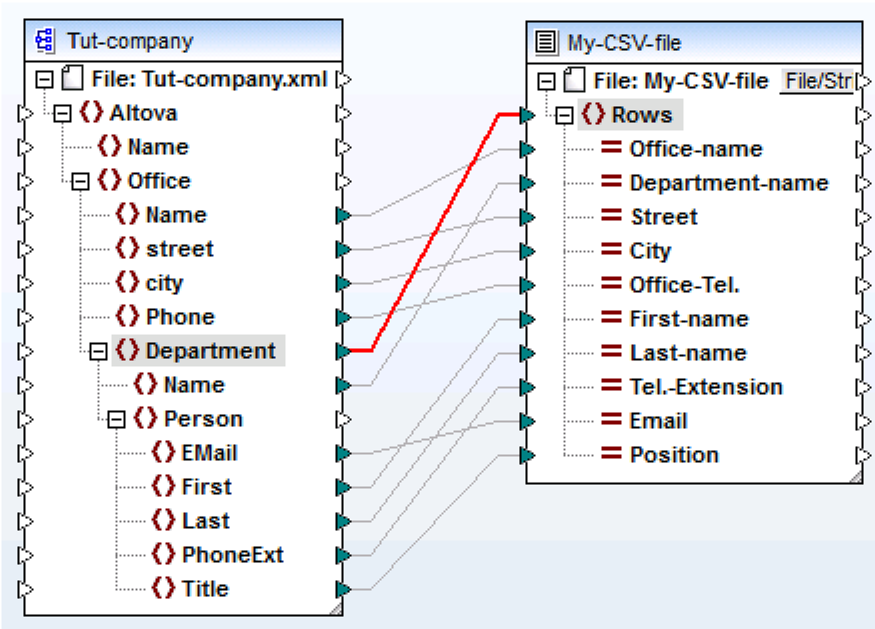
For example, to iterate through all offices and have the output appear in the CSV file, it is necessary to connect `Office` to `Rows`. By doing this, you are instructing MapForce: for each `Office` item of the source XML, create a row in the target CSV file.



The `Rows` item in the CSV component acts as an iterator for the sequence of items connected to it. Therefore, if you connect the Office item, the output creates a row for each office found in the source XML.

```
"Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,558833
"Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Otto
```

In a similar fashion, if you connect **Department** to the **Rows** item, a row will be produced for each department found in the source XML.



The output would then look as follows:

```
"Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Clive,Clo
"Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,558
"Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,K
"Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,5588339
"Microtech Partners, Inc.",Admin,Perro Bvd 1324,Ottowa,3549202,
"Microtech Partners, Inc.",Sales and Marketing,Perro Bvd 1324,O
"Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Ottow

```

Finally, mapping `Person` to the `Rows` item results in all the Persons being output. In this case, MapForce will iterate through the records as follows: each Person within each Department, within each Office.

4.4.3 Example: Creating Hierarchies from CSV and Fixed-Length Text Files

This example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Tut-headerDetail.mfd**. The example uses a CSV file (Orders.csv) which has the following format:

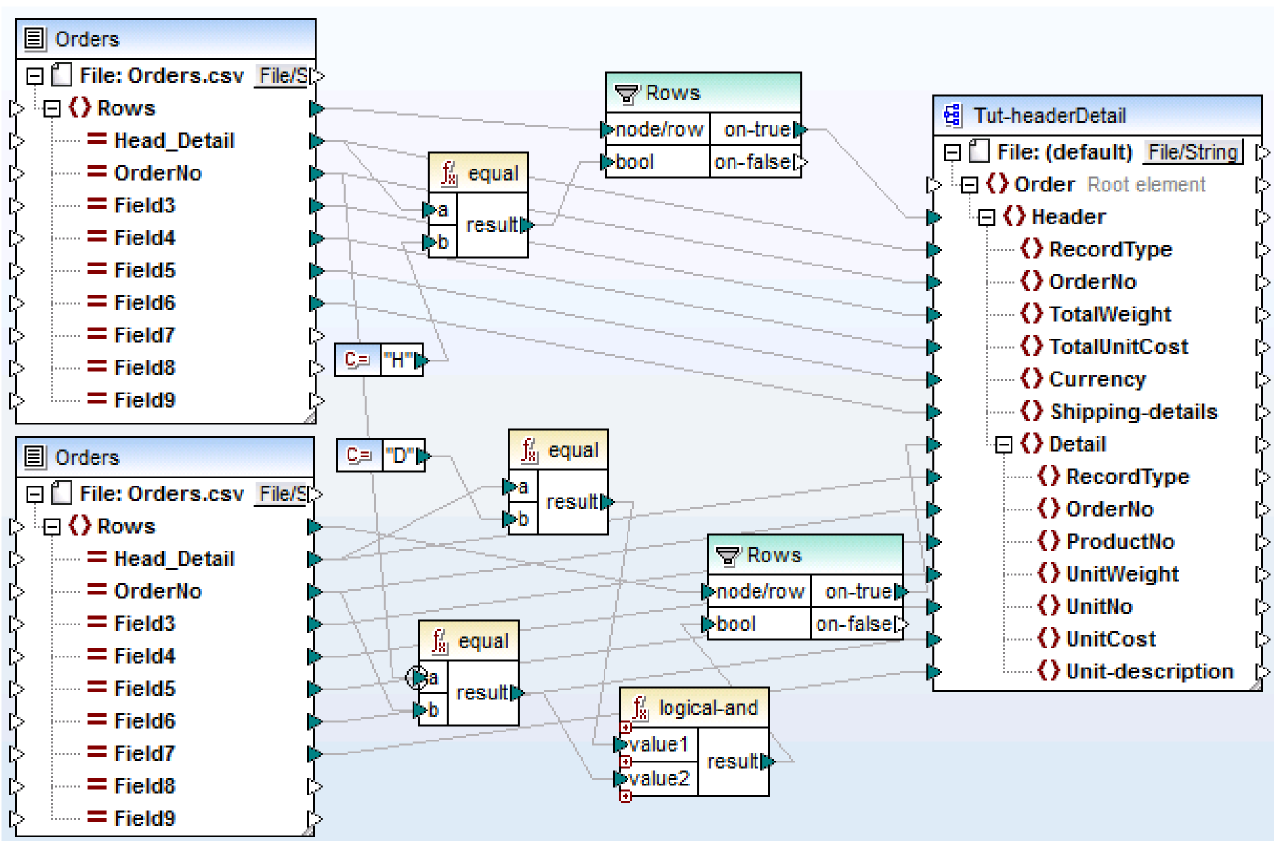
- Field 1: H defines a header record and D a detail record.
- Field 2: A common key for both header and detail records.
- Each Header or Detail record is on a separate line.

The contents of the Orders.csv file are shown below.

```
H,111,332.1,22537.7,,Container ship,,,
D,111,A-1579-227,10,3,400,Microtome,,
D,111,B-152-427,7,6,1200,Miscellaneous,,
H,222,978.4,7563.1,,Air freight,,,
D,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,,
```

The aim of the mapping is as follows:

- Map the flat file CSV to an hierarchical XML file
- Filter the Header records, designated with an H
- Associate the respective detail records, designated with a D, with each of the header records.



tut-headerDetail.mfd

For this to be achieved, the header and detail records must have one common field. In this case the common field, or key, is the second field of the CSV file, i.e. OrderNo. In the CSV file, both the first header record and the following two detail records contain the common value 111.

The Orders.csv file has been inserted twice to make the mapping more intuitive.

The **Tut-headerDetail.xsd** schema file has a hierarchical structure: Order is the root element, with Header as its child element, and Detail being a child element of Header.

The first Orders.csv file supplies the **Header** records (and all mapped fields) to the Header item in the schema target file. The filter component is used to filter out all records other than those starting with H. The **Rows** item supplies these filtered records to the Header item in the schema file.

The second Orders.csv file supplies the **Detail** records (and all mapped fields) by filtering out the Detail records that match the OrderNo key of the Header record. This is achieved by:

- Comparing the **OrderNo** field of the Header record with the same field of the Detail records, using the **equal** function (the priority context[1288] is set on the **a** parameter for enhanced performance).
- Using the **Logical-and** function to only supply those Detail records containing the same OrderNo field, as the Header record.

The **Rows** item supplies these filtered records to the Header and Detail items in the schema file, through the on-true parameter of the filter component.

Clicking the Output tab produces the XML file displayed below. Each Header record contains its data, and all associated Detail records that have the same Order No.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
  <Header>
    <RecordType>H</RecordType>
    <OrderNo>111</OrderNo>
    <TotalWeight>332.1</TotalWeight>
    <TotalUnitCost>22537.7</TotalUnitCost>
    <Currency/>
    <Shipping-details>Container ship</Shipping-details>
    <Detail>
      <RecordType>D</RecordType>
      <OrderNo>111</OrderNo>
      <ProductNo>A-1579-227</ProductNo>
      <UnitWeight>10</UnitWeight>
      <UnitNo>3</UnitNo>
      <UnitCost>400</UnitCost>
      <Unit-description>Microtome</Unit-description>
    </Detail>
    <Detail>
      <RecordType>D</RecordType>
      <OrderNo>111</OrderNo>
      <ProductNo>B-152-427</ProductNo>
      <UnitWeight>7</UnitWeight>
      <UnitNo>6</UnitNo>
      <UnitCost>1200</UnitCost>
      <Unit-description>Miscellaneous</Unit-description>
    </Detail>
  </Header>
```

Let's now have a look at another example, which uses a slightly different CSV file and is available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder as **Head-detail-inline.mfd**. The difference is that:

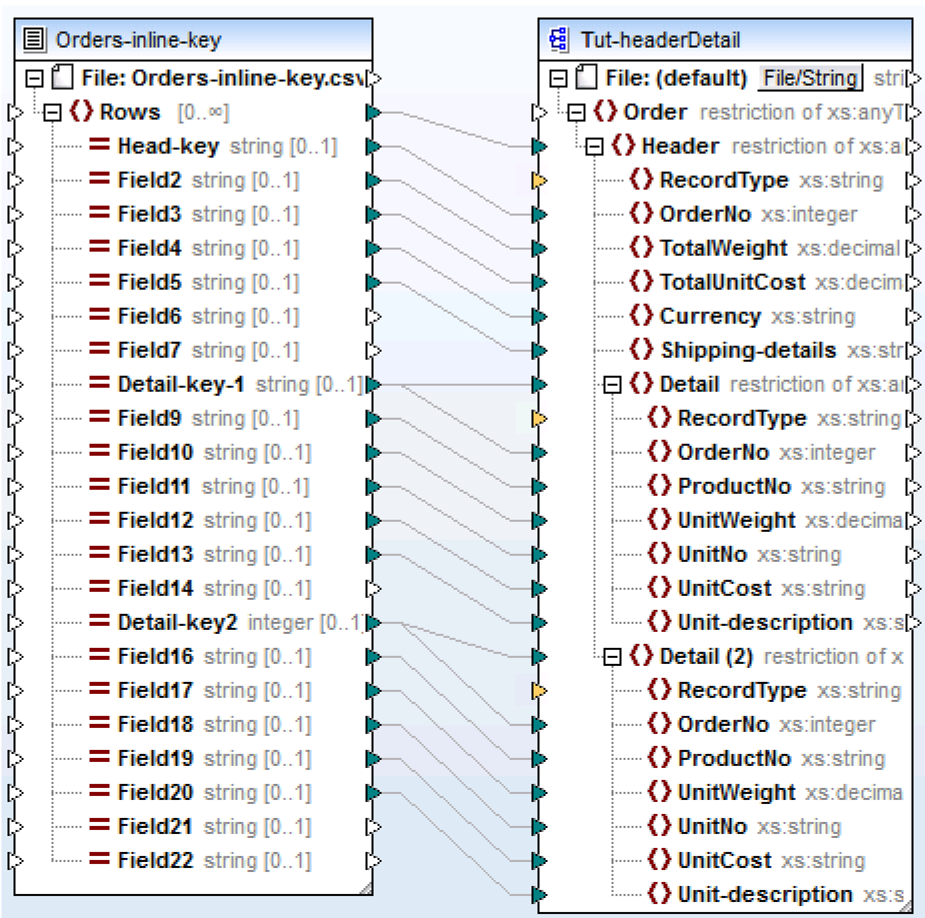
- No record designator (H, or D) is available

- A common key field, the first field of the CSV file, still exists for both header and detail records (Head-key, Detail-key...). The field is mapped to OrderNo in the schema target
- Header and all respective Detail fields are all on the same line.

```
111,332.1,22537.7,,Container ship,,,111,A-1579-227,10,3,400,Microtome,,111,B-15
222,978.4,7563.1,,Air freight,,,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,,
```

The mapping has been designed as follows:

- The key fields are mapped to the respective OrderNo items in the schema target.
- The Detail item in the schema target file has been duplicated, and is displayed as **Detail (2)**. This allows you to map the second set of detail records to the correct item.
- The result of this mapping is basically the same XML file that was produced in the first example.

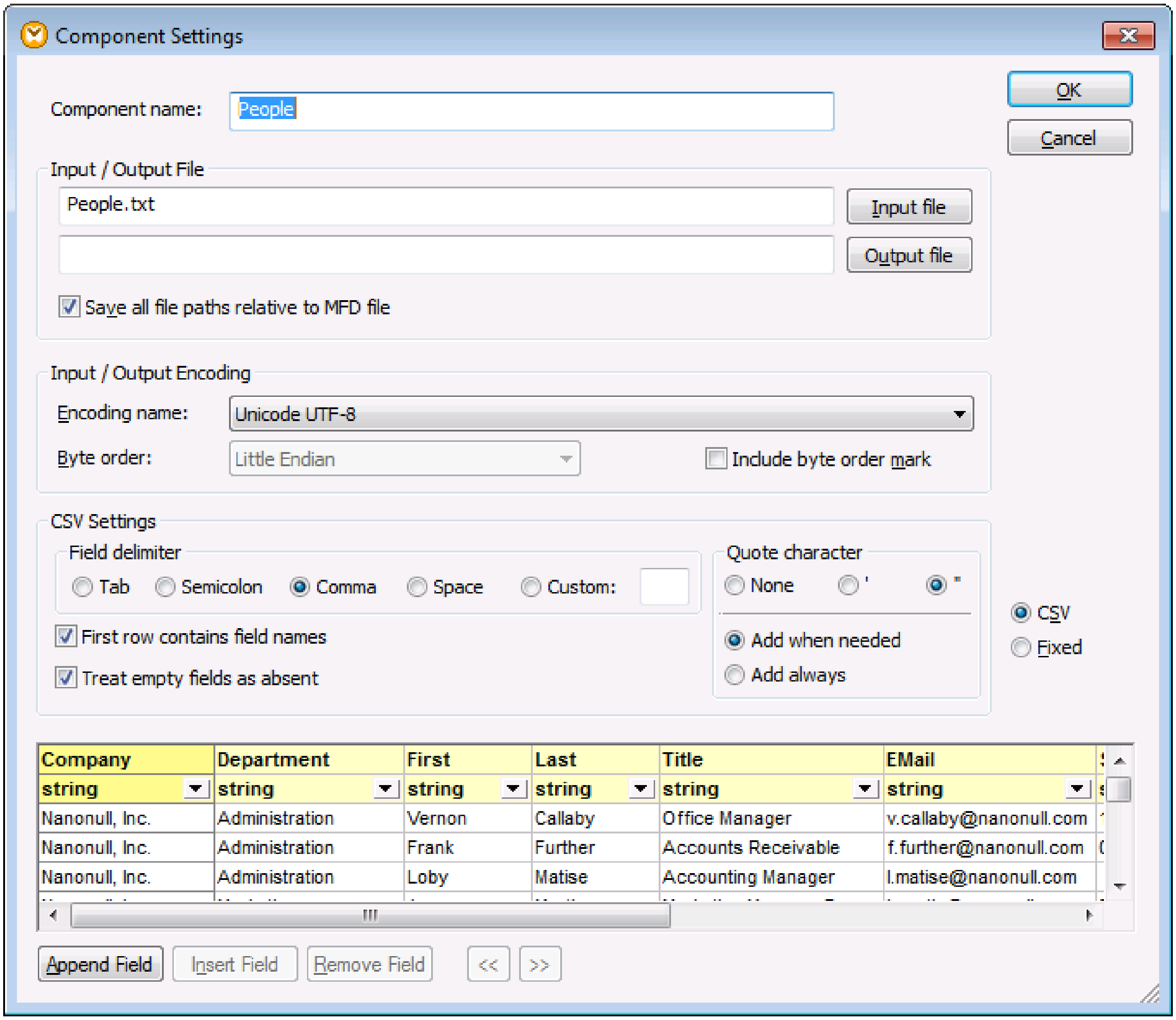


Head-detail-inline.mfd

4.4.4 Setting the CSV Options

After you add a text component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.



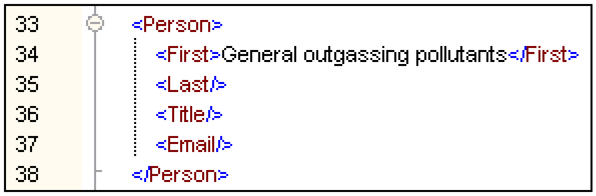
Text Component Settings dialog box (in CSV mode)

The available settings are as follows.

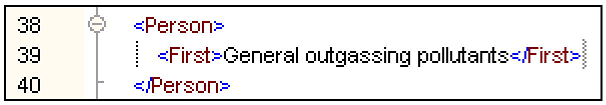
Component name	The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change the name of the component, be aware of the following:

	• If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.
Input file	Specifies the file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The field can remain empty if you are using the text file component as a target for your mapping. In a source component, MapForce uses the value of this field to read column names and preview the contents of the instance text file. To select a new file, click **Input File**.
Output file	Specifies the file to which MapForce will write data. This field is meaningful for a target component. To select a new file, click **Output File**.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the input and output files used by the text component. See also Using Relative Paths on a Component 85.
Input / Output Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab.
Field delimiter	CSV files are comma delimited "," by default. This option enables you to select the **Tab**, **Semicolon**, or **Space** characters as delimiters. You can also enter a custom delimiter in the **Custom** field.
First row contains field names	Select this option to instruct MapForce to treat the values in the first record of the text file as column headers. The column headers then appear as item names on the mapping.
Treat empty fields as absent	When this option is enabled, empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file. For example, the CSV record "`General outgassing pollutants,,,`" consists of four fields, the last three of which are empty.

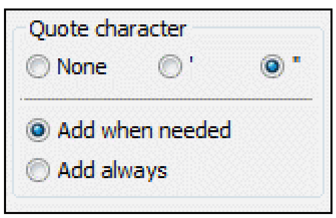
<table>
<tr><td></td><td>
Assuming that the output is an XML file, when this option is disabled, the empty fields will be created in the output with an empty value (in this example, the elements <code>Last</code>, <code>Title</code>, and <code>Email</code>):



When this option is enabled, the empty fields will not be created in the output:



</td></tr>
<tr><td>Quote character</td><td>
If your input file contains quotes around field values, select the quote character that exists in the source file. The same setting will also be used for output files.



For output files, you can specify additional settings:

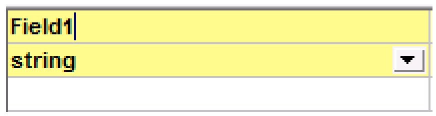
<table>
<tr><td>Add when needed</td><td>Adds the selected quote character to only thos fields where the text contains the field delimite line breaks.</td></tr>
<tr><td>Add always</td><td>Adds the selected quote character to all fields the generated CSV file.</td></tr>
</table>
</td></tr>
<tr><td>CSV / Fixed</td><td>Changes the component type to either CSV or FLF (fixed-length field).</td></tr>
<tr><td>Preview area</td><td>
The lower part of the dialog box displays a preview of up to 20 rows of the file selected as input or output.

If necessary, you can create the structure of the file (or change the structure of the existing one), as follows.

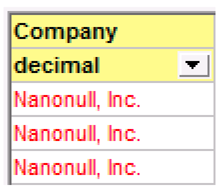
<table>
<tr><td>Append field</td><td>Creates a new field after the last CSV record.</td></tr>
</table>
</td></tr>
</table>

Insert field	Creates a new field immediately before the currently selected CSV record.
Remove field	Deletes the currently selected field.
<<	Moves the currently selected field one position the left.
>>	Moves the currently selected field one position the right.

To change the name of a field, click the header (for example, **Field1**), and type the new value. Note that the field names are not editable when the **First row contains field names** option is enabled.



To change the data type of a field, select the required value from the drop-down list. MapForce checks the data type, so if the input data and the field format to do not agree, then the data is highlighted in red.



The field types are based on the default XML schema data types. For example, the Date type is in the form **YYYY-MM-DD**.

4.4.5 FLF to Database

This topic explains how to map data of a fixed-length text file (FLF) to a Microsoft Access database. The files used in this topic are available in the `Tutorial` folder. The source text file and the target database store a list of employees. In the source file, the records are delimited by their size as follows:

Field position and name	Size (in characters)
Field 1 (First name)	8
Field 2 (Last name)	10
Field 3 (Phone extension)	3
Field 4 (Email)	25

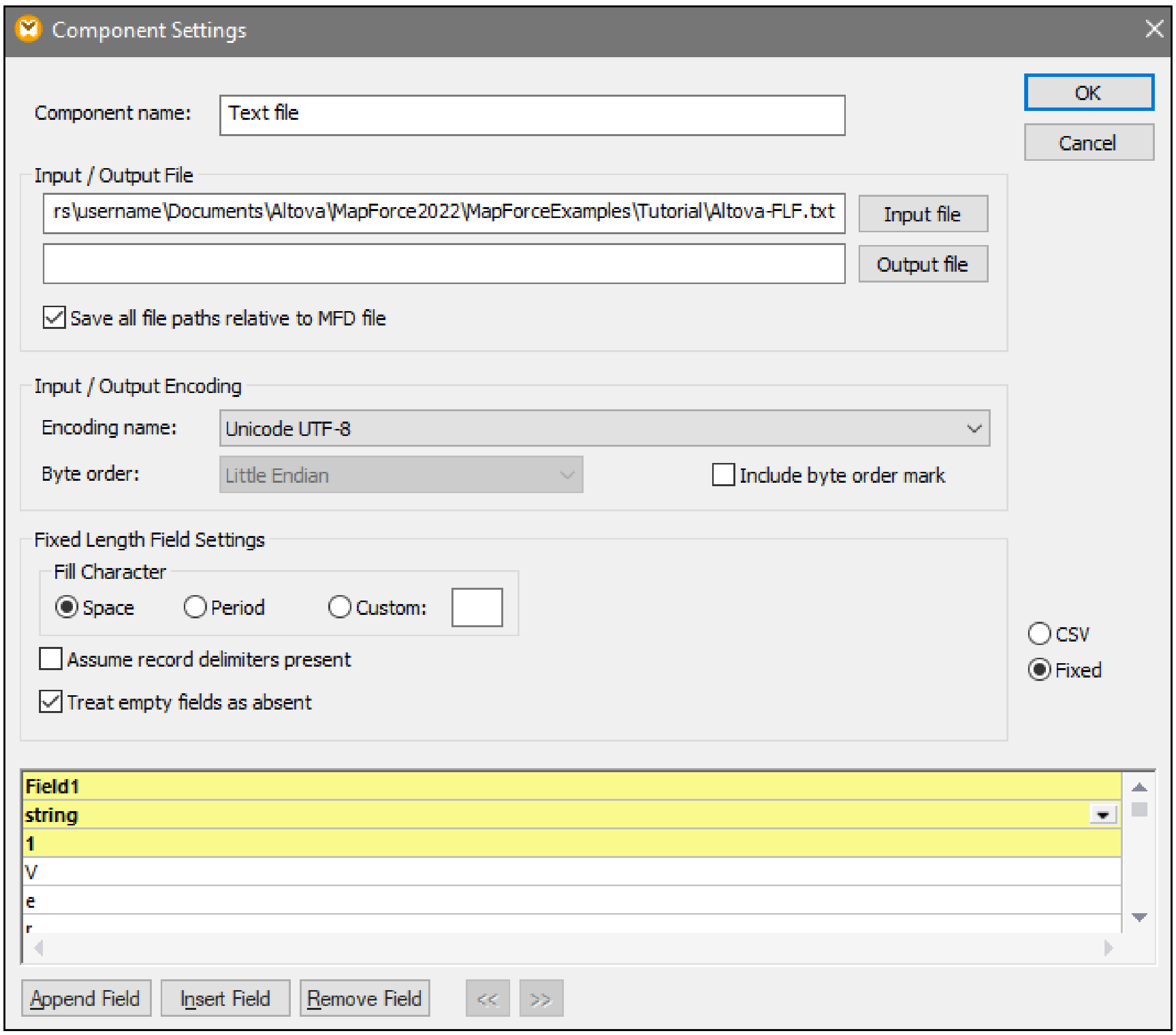
Field position and name	Size (in characters)
Field 5 (Position)	25

The goal of the mapping is to map the FLF data to the database component. We also want to map the phone extensions with a new prefix. To achieve this, take the steps below.

Step 1: Insert a text component

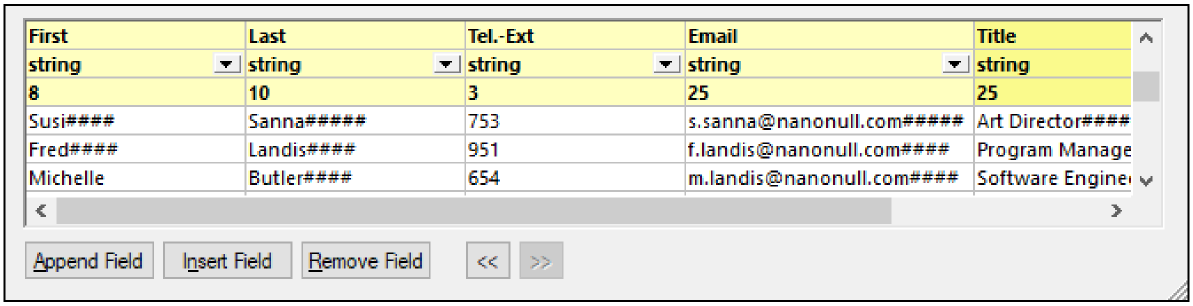
The first step is to add and configure a text component. Follow the instructions below:

1. Select the menu item **Insert | Text file** or click the  toolbar button (**Insert Text file**). This opens the **Insert Text Component** dialog box. Select **Use simple processing...** and click **Continue**.
2. Click **Input file** in the **Component Settings** dialog box (*see below*) and select **`Altova-FLF.txt`**.



3. Select **Fixed**.
4. Clear the **Assume record delimiters present** check box.

5. The yellow rows are editable and enable you to specify i) the field name, ii) the data type, and iii) the field size. Set the field size to 8 (the third yellow line from the top) and press **Enter**. More data is now visible in the first column, which is now 8 characters wide.
6. Click **Append Field** to add a new field and set the length of the second field to 10 characters.
7. Use the same method to create three more fields of the following lengths: 3, 25, and 25 characters. Then change the field headers, as shown in the screenshot below.

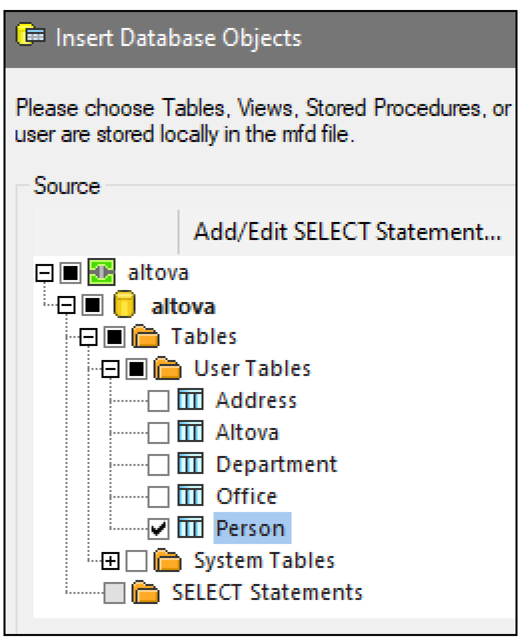


8. In **Fixed Length Field Settings**, select **Custom** and type the hash (#) character. This instructs MapForce to treat the # character as a fill character.
9. Click **OK**.
10. MapForce will ask you if you would like to change the component name to match the instance files. Click **Change component name**. The `Altova-FLF` component appears in the mapping window.

Step 2: Insert a database component

The next step is to add a database component. Follow the steps below:

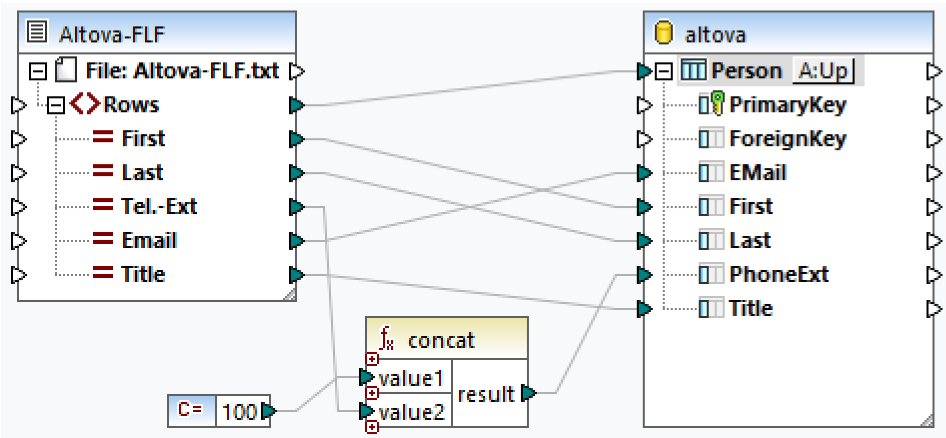
1. Go to **Insert | Database**, select **Microsoft Access**, and click **Next**.
2. Select the **`altova.mdb`** database and click **Connect**.
3. Select the **Person** table (*see below*) and click **OK**.



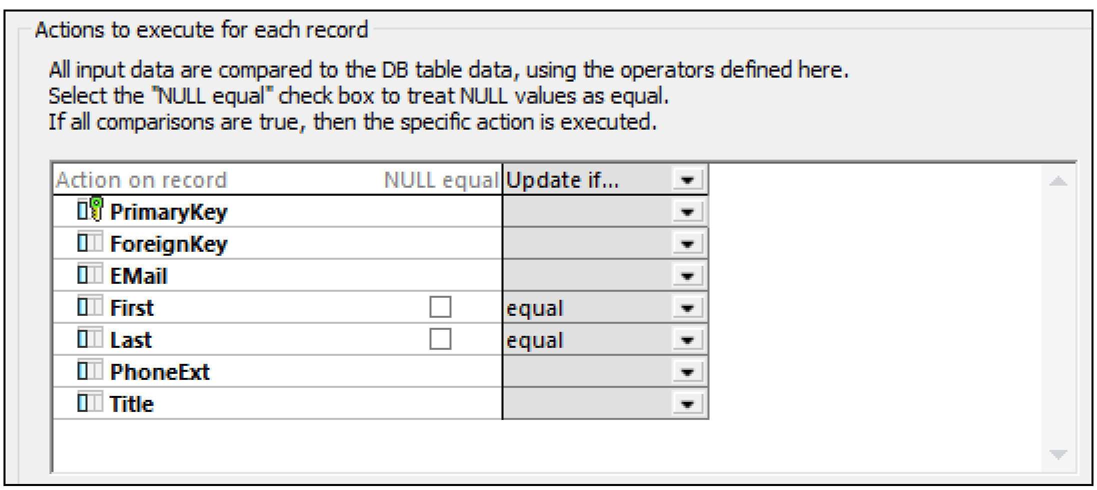
Step 3: Design the mapping

The next step is to create a mapping:

1. Drag the **core | concat** 1057 function from the **Libraries** window into the mapping.
2. Go to **Insert | Constant**, select *Number* as a type, and enter 100 in the text field. This constant stores the new telephone extension prefix.
3. Create connections as shown below.



4. In the database component, click the **Table Action** button next to **Person**. This opens the **Database Table Actions** dialog box (*see screenshot below*).
5. Next to *Action on record data*, select **Update if**. Set the `equal` action for the **First** and **Last** fields. Click **OK**. MapForce will be instructed to update the `Person` table only if the first and last names in the source file are equal to the corresponding database fields. When this condition is true, the telephone extension will get a prefix 100 and will be copied to the `PhoneExt` field of the `Person` table.

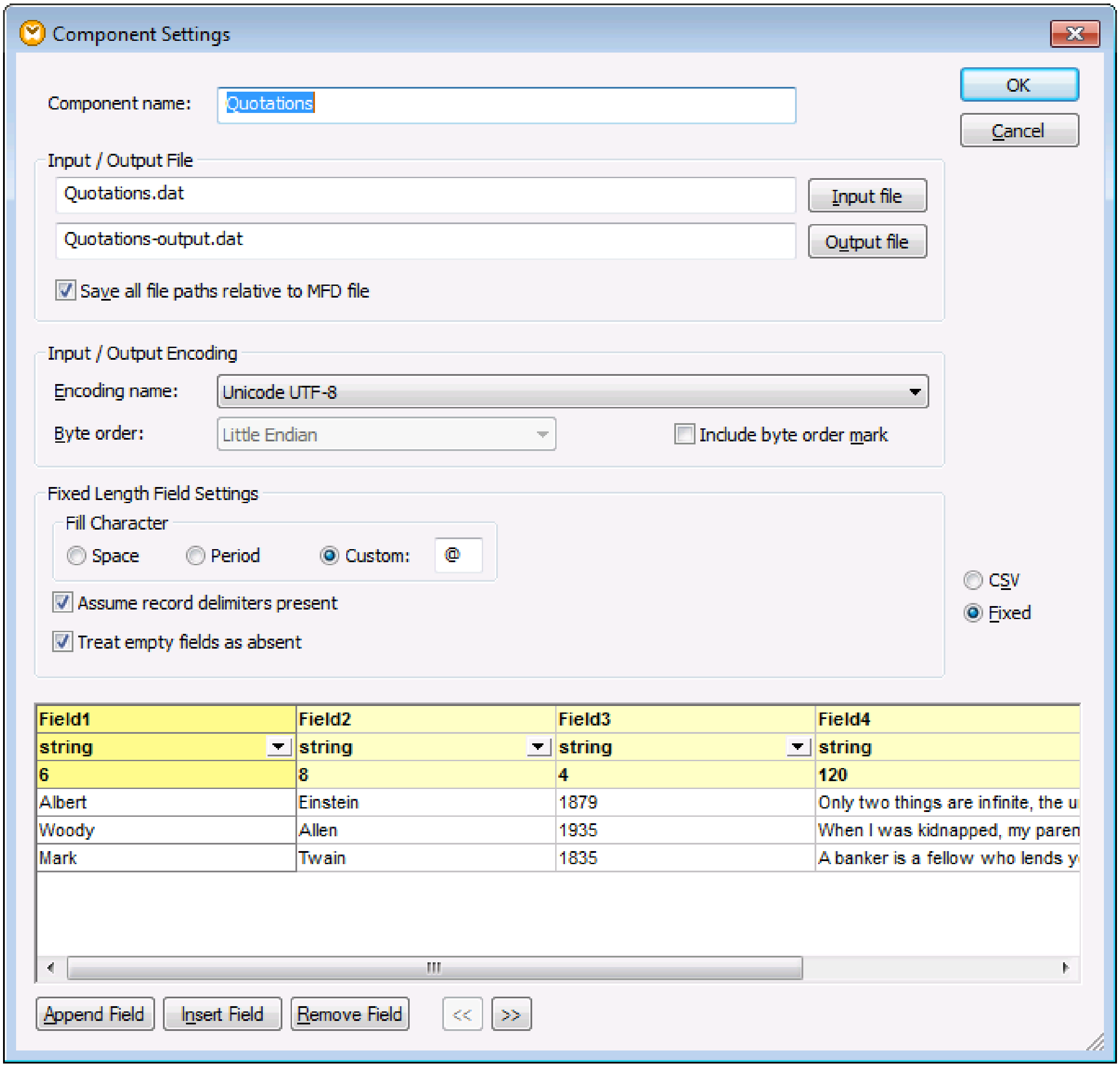


6. To generate the SQL statements (for preview in MapForce), click the **Output** pane. To run the SQL statements against the database, click the **Run SQL-script** button.

4.4.6 Setting the FLF Options

After you add a text component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.



Text Component Settings dialog box (in fixed-length field mode)

The available settings are as follows.

Component name	The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change the name of the component, be aware of the following: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.
Input file	Specifies the file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The field can remain empty if you are using the text file component as a target for your mapping. In a source component, MapForce uses the value of this field to read column names and preview the contents of the instance text file. To select a new file, click **Input File**.
Output file	Specifies the file to which MapForce will write data. This field is meaningful for a target component. To select a new file, click **Output File**.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the input and output files used by the text component. See also Using Relative Paths on a Component 85.
Input / Output Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab.
Fill Character	This option allows you to define the characters that are to be used to complete, or fill in, the rest of the (fixed) field when the incoming data is shorter than the respective field definitions. The custom field allows you to define your own fill character.

	If the incoming data already contains specific fill characters, and you enter the same fill character in the Custom field, then the incoming data will be stripped of those fill characters!
Assume record delimiters present	This option is useful when you want to read data from a source flat file that does not contain record delimiters such as CR/LF, or when you want to produce a target flat FLF file without record delimiters. See the Understanding the "Assume record delimiters present" option 583 section below.

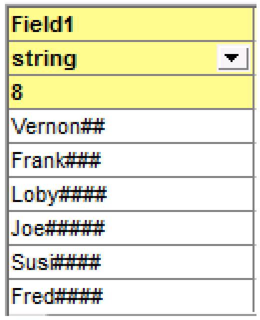
Treat empty fields as absent	When this option is enabled, empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file.
	
	Assuming that the output is an XML file, when this option is disabled, the empty fields will be created in the output with an empty value (in this example, the elements `Last`, `Title`, and `Email`):
	
	`<Person>`
	`  <First>General outgassing pollutants</First>`
	`  <Last/>`
	`  <Title/>`
	`  <Email/>`
	`</Person>`
	
	When this option is enabled, the empty fields will not be created in the output:
	
	`<Person>`
	`  <First>General outgassing pollutants</First>`
	`</Person>`

CSV / Fixed	Changes the component type to either CSV or FLF (fixed-length field).
Preview area	The lower part of the dialog box displays a preview of up to 20 rows of the file selected as input or output. If necessary, you can create the structure of the file (or change the structure of the existing one), as follows. **Append field** — Creates a new field after the last record. **Insert field** — Creates a new field immediately before the currently selected record. **Remove field** — Deletes the currently selected field. **<<** — Moves the currently selected field one position the left.

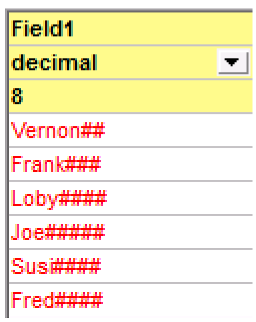
<table>
<tr>
<td></td>
<td>

>> Moves the currently selected field one position the right.

To change the name of a field, click the header (in this example, **Field1**), and type the new value.



To change the data type of a field, select the required value from the drop-down list. MapForce checks the data type, so if the input data and the field format to do not agree, then the data is highlighted in red.

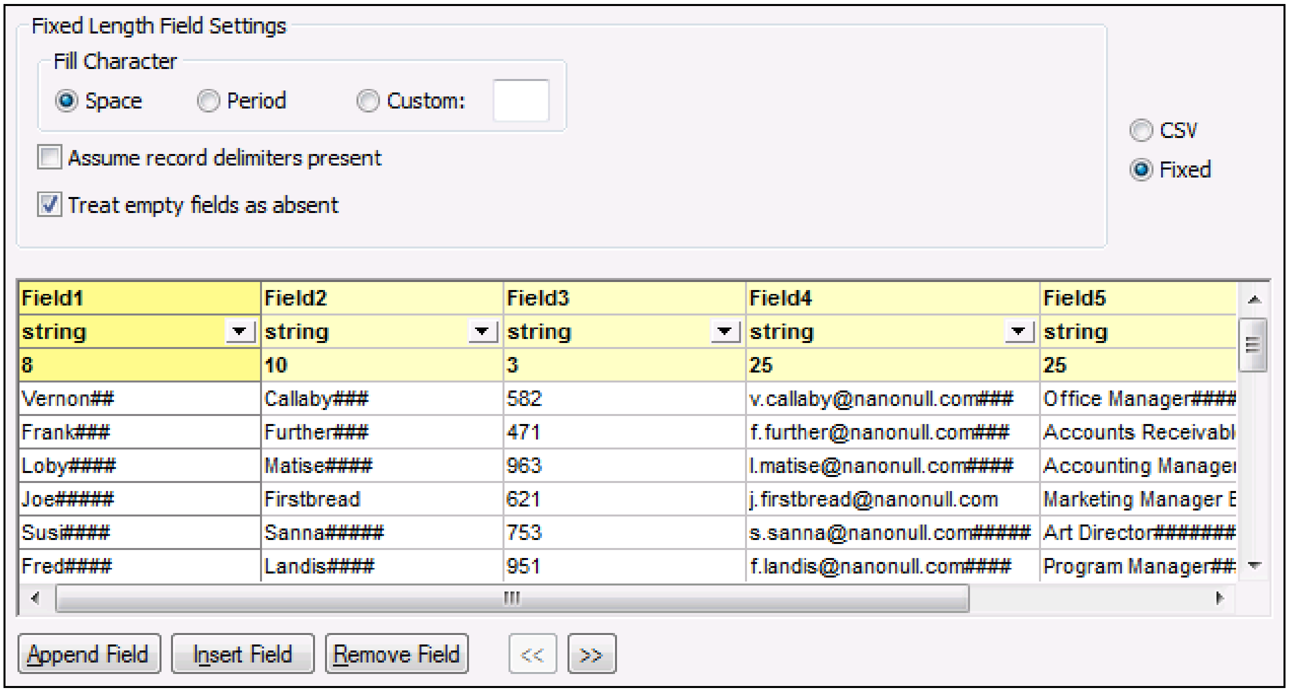


To set the size of the field in characters, enter the field size in the third row from the top.

</td>
</tr>
</table>

Understanding the "Assume record delimiters present" option

To better understand this option, open the **Altova-FLF.txt** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. Notice that the file consists of 71-character long records, without any delimiters such as CR/LF. If you would need to read data from this particular file, first you would need to split this file into records. That is, create several fields whose total size sums up to 71 characters (as shown below), and then disable **Assume record delimiters present**. For a step-by-step example, see Example: Mapping Fixed-Length Text Files to Databases 576.



If you would need to write data from this file to a destination file which uses the same structure, then enabling **Assume record delimiters present** creates a new record after every 71 characters.

```
Vernon##Callaby###582v.callaby@nanonull.com###Office Manager###########
Frank###Further###471f.further@nanonull.com###Accounts Receivable######
Loby####Matise####963l.matise@nanonull.com####Accounting Manager#######
Joe#####Firstbread621j.firstbread@nanonull.comMarketing Manager Europe#
Susi####Sanna#####753s.sanna@nanonull.com#####Art Director#############
Fred####Landis####951f.landis@nanonull.com####Program Manager##########
MichelleButler####654m.landis@nanonull.com####Software Engineer########
Ted#####Little####852t.little@nanonull.com####Software Engineer########
Ann#####Way#######951a.way@nanonull.com#######Technical Writer#########
Liz#####Gardner###753l.gardner@nanonull.com###Software Engineer########
Paul####Smith#####334p.smith@nanonull.com#####Software Engineer########
Alex####Martin####778a.martin@nanonull.com####IT Manager###############
George##Hammer####223g.hammer@nanonull.com####Web Developer############
Jessica#Bander####241j.band@nanonull.com######Support Engineer#########
Lui#####King######345l.king@nanonull.com######Support Engineer#########
Steve###Meier#####114s.meier@nanonull.com#####Office Manager###########
Theo####Bone######331t.bone@nanonull.com######Accounts Receivable######
Max#####Nafta#####122m.nafta@nanonull.com#####PR & Marketing Manager US
ValentinBass######716v.bass@nanonull.com######IT Manager###############
Carl####Franken###147c.franken@nanonull.com###Support Engineer#########
Mark####Redgreen##152m.redgreen@nanonull.com##Support Engineer#########
```

The mapping result when "Assume record delimiters present "is enabled

If **Assume record delimiters present** is disabled, the mapping result appears as one long string.

```
Vernon##Callaby###582v.callaby@nanonull.com###Office
Manager###########Frank###Further###471f.further@nanonull.com###Accounts
Receivable######Loby####Matise####963l.matise@nanonull.com####Accounting
Manager#######Joe#####Firstbread621j.firstbread@nanonull.comMarketing Manager
Europe#Susi####Sanna#####753s.sanna@nanonull.com#####Art
Director#############Fred####Landis####951f.landis@nanonull.com####Program
Manager##########MichelleButler####654m.landis@nanonull.com####Software
Engineer########Ted#####Little####852t.little@nanonull.com####Software
Engineer########Ann#####Way#######951a.way@nanonull.com#######Technical
Writer#########Liz#####Gardner###753l.gardner@nanonull.com###Software
Engineer########Paul####Smith#####334p.smith@nanonull.com#####Software
Engineer########Alex####Martin####778a.martin@nanonull.com####IT
Manager###############George##Hammer####223g.hammer@nanonull.com####Web
Developer############Jessica#Bander####241j.band@nanonull.com######Support
Engineer#########Lui#####King######345l.king@nanonull.com######Support
Engineer#########Steve###Meier#####114s.meier@nanonull.com#####Office
Manager###########Theo####Bone######331t.bone@nanonull.com######Accounts
Receivable######Max#####Nafta#####122m.nafta@nanonull.com#####PR & Marketing Manager
USValentinBass######716v.bass@nanonull.com######IT
Manager###############Carl####Franken###147c.franken@nanonull.com###Support
Engineer#########Mark####Redgreen##152m.redgreen@nanonull.com##Support
Engineer#########
```

The mapping result when "Assume record delimiters present "is disabled

4.5 Web Services

In MapForce, you can design a mapping that calls a Web service as part of its execution. Namely, you can insert a Web service call into a mapping similar to a function, connect input and output components to it, and consume the result as required (e.g., you can preview it in the MapForce output window, pass it to another component, or write it to a file).

A Web service call created with MapForce can be executed by MapForce itself or on a different machine or even platform, by MapForce Server. MapForce Server can run mappings through the command-line interface, as an API call or as a FlowForce Server job. For details, see Automation with MapForce Server1371.

Supported Web services

MapForce supports calling WSDL-based and REST Web services (*see details below*).

- For WSDL-based Web services, you need to provide the WSDL file of the Web service. MapForce uses the WSDL file to communicate with the Web server. If the WSDL file implements multiple services, endpoints, and operations, you can select or update them directly in MapForce.
- For REST Web services, you need to enter Web service details (i) manually or (i) automatically (by importing a WADL file or a URL). The Web service details include a URL, a request method (e.g., GET, POST, PUT), request and response structures (e.g., XML, JSON, custom MIME types), and parameters.

REST vs. WSDL-based Web services

The table below summarizes support details of REST and WSDL-based Web services in MapForce.

Feature	REST	WSDL-based
Mapping language	Built-In	Built-In, C#, Java
Automation with MapForce Server	Yes	Yes, when language is Built-In
Protocols	HTTP (GET, POST, PUT, DELETE, custom verbs)	• SOAP 1.1, 1.2 over HTTP • Non-SOAP services over HTTP
Request/response structures	• XML • JSON • Protocol Buffers • Custom MIME types	SOAP message
Security	• HTTP/HTTPS • Server certificates • Client certificates • HTTP authentication • Preemptive authentication • OAuth authorization	• HTTP/HTTPS • Server certificates • Client certificates* • HTTP authentication • Preemptive authentication* • WS-Security* • OAuth authorization*

Feature	**REST**	**WSDL-based**
		** Supported only in Built-In.*
Import service definition	From WADL file (optional)	From WSDL file (required)
Import parameters from URL	Yes	Not applicable
Dynamic Web service URL	You can provide the full URL of the Web service as a parameter to the mapping or define it inside the mapping. You can also supply specific URL parts as parameters to the mapping while the base URL is defined in the mapping.	You can provide the URL of the Web service as a parameter to the mapping or make it fixed (saved together with the mapping).
Timeout (the interval after which the call should time out if the Web server is not responding)	Yes	Yes
Dynamic authentication (Authentication details supplied by the mapping or as a parameter to the mapping. For details, see Dynamic Authentication 1309.)	Yes	Yes

WSDL-based Web serivces

When you select a WSDL-based Web service, you can use the following protocols:

- SOAP 1.1, 1.2 over HTTP: Both the `RPC/Encoded` and `Document/Literal` styles are supported. If the Web server returns a WSDL fault, the mapping execution stops. In such cases, you can optionally insert an exception component on the mapping area to handle the error. If the Web server returns a non-WSDL error, the mapping execution stops, and an error message is returned (or displayed on the screen if you are previewing the mapping in MapForce).
- Non-SOAP over HTTP: This refers to less common, non-SOAP HTTP services. With HTTP GET, the `url-encoded` style is supported. With HTTP POST, the `url-encoded` and `text/xml` styles are supported.

4.5.1 REST Web Services

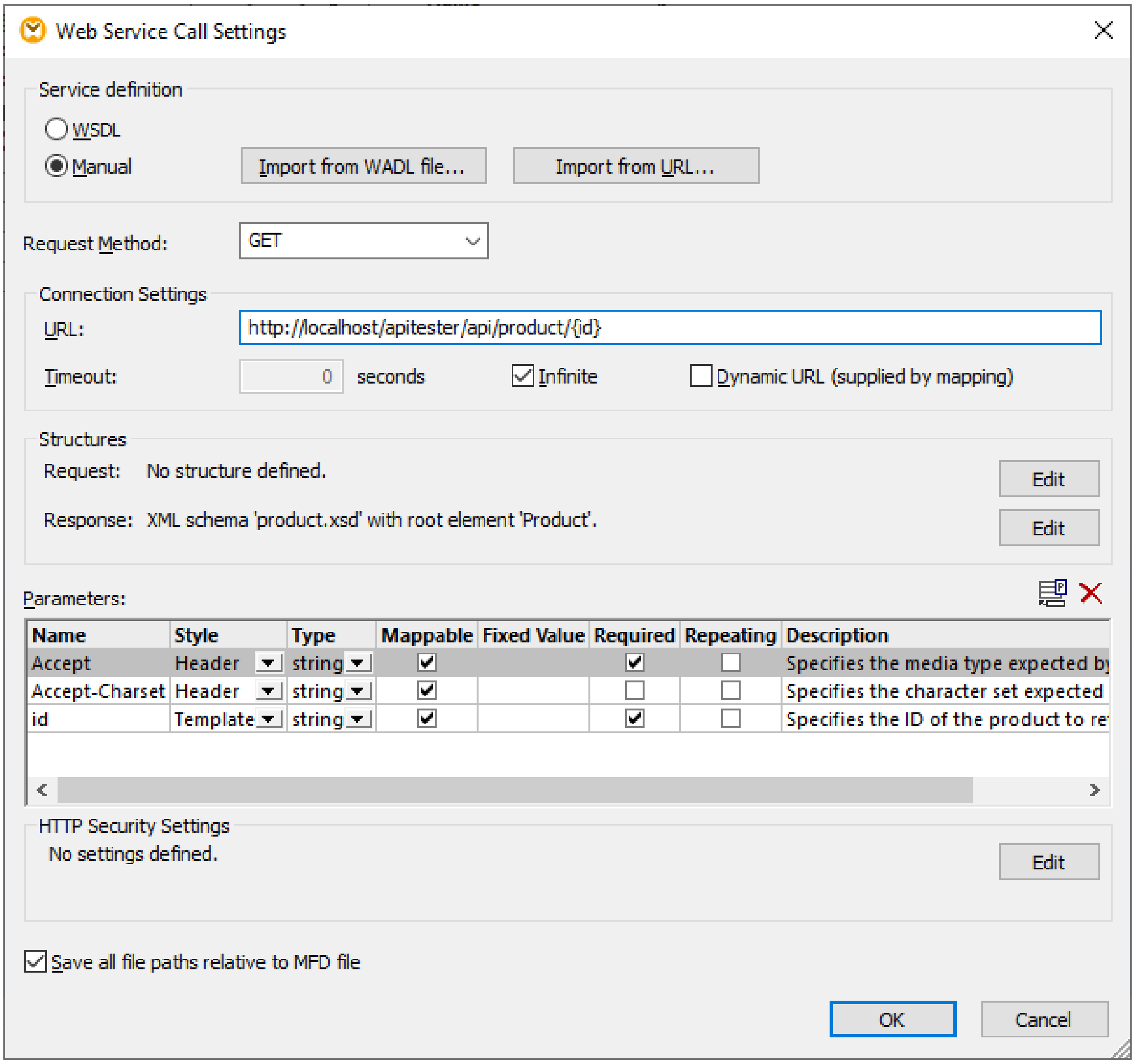
Generic (not WSDL-style) Web services include a large category of Web services that follow or partially follow an architectural style referred to as "REST" (and are typically called "RESTful", or REST-style Web services). Generic HTTP Web services typically carry custom request or response structures in the message body part. MapForce supports the following kinds of request or response bodies: JSON, XML, Protocol Buffers, and unstructured bodies carrying custom MIME types. Therefore, from a MapForce perspective, the prerequisites to call a Web service depend on the type of the request:

- In case of XML or JSON requests or responses, MapForce needs the request or response structure as a JSON, XML, or DTD schema. MapForce will also accept as structure an XML file with a valid schema reference. The structure of the Web service could be published by the service provider as XML or JSON schema, or through a formal language such as WADL, or even be a human-readable specification. If you already have the XML or JSON schema of the request/response, you will need the least amount of effort to create the Web service call. If you have the Web service definition as a WADL file, you can import it from the WADL file, and make any potential adjustments manually. Note, however, that WADL provides no standard way to define JSON structures, only XML structures. Finally, if you have a sample XML or JSON instance file but don't have a schema file, you can either create or generate the schema with XMLSpy (https://www.altova.com/xmlspy.html). If necessary, XMLSpy can also convert your instance file from XML to JSON, or vice versa.

- In case of Protocol Buffers requests or responses, you need the .proto file that describes the Protocol Buffers binary file. In this scenario, the body of the Web service can be mapped to or from a Protocol Buffers component. For more information, see Example: Read Data from Protocol Buffers 816 and Example: Write Data to Protocol Buffers 821.
- You can also call Web services where the request or response structure is flexible and not tied to a particular schema. For such cases, you can use MapForce built-in mime 1131 functions to either create the raw message body sent to the Web service (the MIME entity) or process from the mapping the MIME entity returned by the Web service.

Adding a call to a generic Web service

1. On the **Insert** menu, click **Web Service Function**. (Alternatively, click the **Insert Web service function** toolbar button).
2. Under **Service definition**, click **Manual**.



3. Optionally, if you have the WADL file describing the service, click **Import from WADL file** and select the file, see also [Importing Web Service Information from WADL](598)[598].
4. Select the HTTP request method that MapForce should use to call the Web service. You can either select a value from the existing list, or type the name of the request method. The HTTP method names are case-sensitive.
5. Do one of the following:

    a. Enter the URL of the Web service in the **URL** text box. In this approach, you can optionally turn specific parts of the URL into parameters and supply them from the main mapping. For URLs defined this way, note the following:
        i. If you are calling a Web service with "template" or "matrix" style parameters, enclose the parameters within curly braces, for example: `http://example.org/api/products/{id}`. Then define the actual settings of each parameter in the "Parameters" table. At runtime, MapForce replaces the placeholders in curly braces with the actual parameter values and produces the final URL.

   ii. If you are calling a Web service with "query" URL parameters (for example, `http://example.org/api/products?sort=asc&category=1&page=1`), do not enter the query part in the URL text box. Instead, define the parameters only in the "Parameters" table, and make sure to designate them as "Query"-style parameters.
   iii. For examples, see Defining Web Service Parameters 592.

b. If you would like to supply the *full* URL of the Web service from the main mapping (or perhaps as parameter to the mapping), select the **Dynamic URL (supplied by mapping)** check box. This disables the **URL** text box and in this case you must create the full URL of the Web service (including any URL parameters) from the main mapping and connect it to the respective input that appears on the Web service component. Note that with this approach it is no longer meaningful to specify any parameters in the "Parameters" table, except for "Header"-style parameters. If you attempt to do this, a dialog box will warn you to this effect.

Dynamic URLs
With either of the two approaches described above (fully or partially dynamic URLs), you can flexibly adjust the URL based on your needs. For example, you could run the mapping with a specific URL during development and use a different URL in production, without modifying the mapping. A URL like `https://{host}/some/path/to/service` would make this possible, assuming that the host name is the only difference between the production and testing URLs, and you supply it as a parameter to the mapping. Note that partially dynamic URLs enforce stricter validation, since only the designated URL parts are replaced with mappable or runtime values.

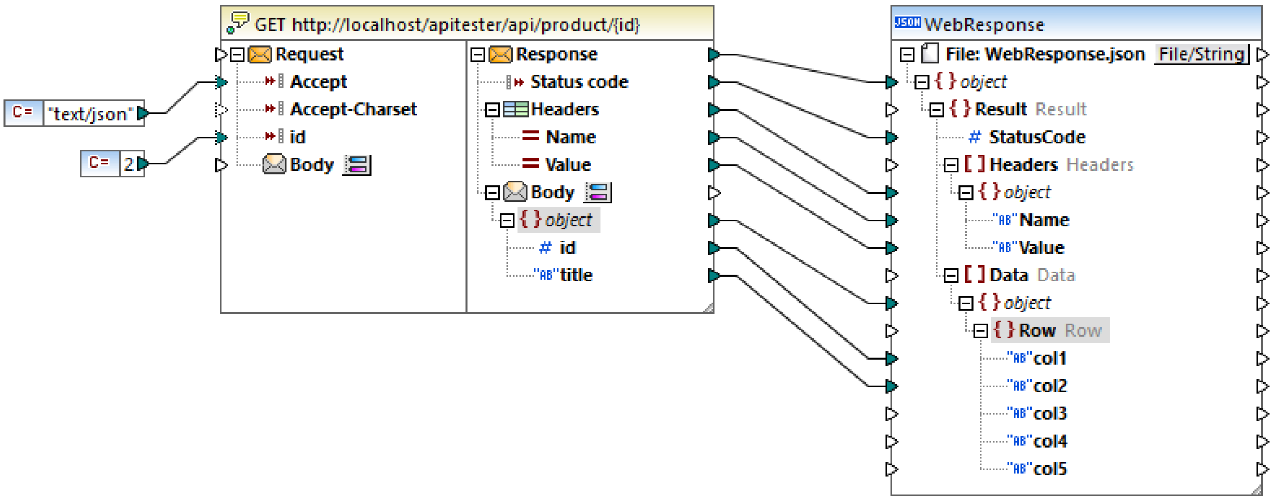
With fully dynamic URLs, the entire URL is mappable and you have complete control over it; the only requirement is that the URL must begin with **http://** or **https://** and it must be a valid URI. Dynamic URLs also make it possible to supply the URL returned from a Web service call as input to another component which may call the same (or some other) Web service.

6. Optionally, under **Timeout**, enter a period in seconds after which the connection should time out if the server is not responding.
7. If the HTTP method requires or returns a body part (such as XML, JSON, and others), click the **Edit** button under **Structure** and browse for the schema of the body part. For more information, see Defining the Request Structure 599 and Defining the Response Structure 601.
8. Under **Parameters**, define the parameters of the Web service. Optionally, click **Import from URL** to import the parameters from a sample URL of the Web service and populate the "Parameters" table automatically, as shown in Importing Web Service Parameters from URL 597. After importing parameters from a URL, you can modify the contents of the "Parameters" table if necessary.

Note: To specify custom request headers, add a parameter with style "Header", where the parameter name corresponds to the header name, and the parameter value corresponds to the header value. Furthermore, if you need to provide the value of the request header from the mapping itself, set the parameter type to "Mappable". For further information, see Defining Web Service Parameters 592.

9. If the Web service requires basic HTTP authentication, OAuth 2.0 authorization, or certificate-based security, click the **Edit** button under **HTTP Security Settings** and fill in the required fields, see also Setting HTTP Security 610.

After you click **OK**, a new Web service component is added to the mapping area. For example, the mapping below calls a Web service in order to retrieve a product by its ID using a GET request. In this example, the ID supplied in the HTTP request has the constant value "2". However, it could also be a parameter to the mapping, or be supplied by any component supported by MapForce. In addition to the `id` parameter, the request contains the header `Accept: text/json`. You can set the request headers by means of "header" parameters, as shown in Defining Web Service Parameters 592.



Notice that the Web service component consists of two parts: **Request** and **Response**. The **Request** part enables you to supply data from the mapping to the Web service, while the **Response** part enables you to access the data returned by the Web service and map it to other formats. The structure of the request and response depends on the parameters, as well as the request or response structure you have defined by clicking the button. In the Web service call above, the structure of the request body is not set, while the structure of the response body is set as JSON—which makes it possible to map the result of the Web service to an actual JSON file.

To call the Web service with request parameters (if applicable), draw mapping connections between any component supported by MapForce (for example, an XML or JSON file) and the **Request** part. Likewise, to map data returned by the Web service to another format, draw mapping connections between the **Response** part and any other component type supported by MapForce. If you are new to MapForce and need instructions about drawing mapping connections, see Working with Connections 89.

The response headers returned by the Web service are also mappable, if they are additional headers (the ones which do not begin with "Content"). The header values are available on the Web service component through an item called "Headers" ( ) which contains two child items: "Name" and "Value". This structure acts as sequence, and enables you to map data from any number of headers returned by the Web service. To map the data from the response headers to any other format supported by MapForce, connect the "Headers" structure node to a target sequence in the mapping. For example, if you connect the **Headers** sequence (and its children) to a **Rows** sequence (and its children) of a CSV component, one header would correspond to one row in the CSV file.

The **Body** represents the entity body of the HTTP message. Data at this level is binary encoded, so interacting with it directly requires MapForce mime 1131 functions. Note that you don't typically need to interact with the **Body** at all if the Web service expects or returns structured data such as XML or JSON. Mapping data directly to or from the **Body** is only necessary when calling Web services that expect or return unstructured content.

> By default, the **Body** item is configured to parse the result if the HTTP status code ranges from 200

through 299. This means that the mapping will return an error for status codes greater than 299. Also, an error will be returned if the response cannot be parsed or the Web service cannot be called due to connection failure or DNS resolution problems.

In some cases, you might want the mapping not to throw an error even if the HTTP status code is greater than 299. To do this, click the button next to the response **Body** and change the HTTP status code range. Alternatively, you can create multiple response **Body** items. Creating multiple **Body** items is useful when you want to handle the mapping conditionally depending on the HTTP status code returned by the Web service. To do this, right-click the **Body** of the response part, and select **Add Body Node Before/After** from the context menu. For example, you could create two **Body** items:

1. A **Body** for all status codes that are in the success range (200 to 299). You can configure the status codes range by clicking the button. For this scenario, you will typically want to map the **Body** structure to some "success" output, like a JSON or XML file.
2. A **Body** for all other status codes, in the range between 300 and 599. Some Web services may provide additional details about error status codes in the body. To find out the reason of the error, you can therefore map the "erroneous" **Body** to an output other than the "success" one (for example, a string).

When multiple **Body** items exist, MapForce evaluates them from top to bottom. To change the priority, right-click the **Body** item, and select **Move Up** or **Move Down** from the context menu. For more information about handling the HTTP response conditionally, see Handling the HTTP Response Conditionally 603.

Mappings containing generic HTTP Web service calls can be executed like most other mappings, namely:

- Manually, with MapForce, by clicking the **Output** tab. In this case, the result of the mapping call is immediately available in the **Output** pane. If the mapping has multiple output components like in the example above, press the **Preview** button on the desired component before running the mapping.
- Through command line or API calls, with MapForce Server (https://www.altova.com/mapforce-server). This requires compiling the mapping to a mapping execution file first (see Compiling Mappings to MapForce Server Execution Files 1378).
- As a recurring job, with MapForce Server running under FlowForce Server control (https://www.altova.com/flowforceserver). See also Deploying Mappings to FlowForce Server 1381.

For step-by-step examples of how to create a generic Web service call, see:

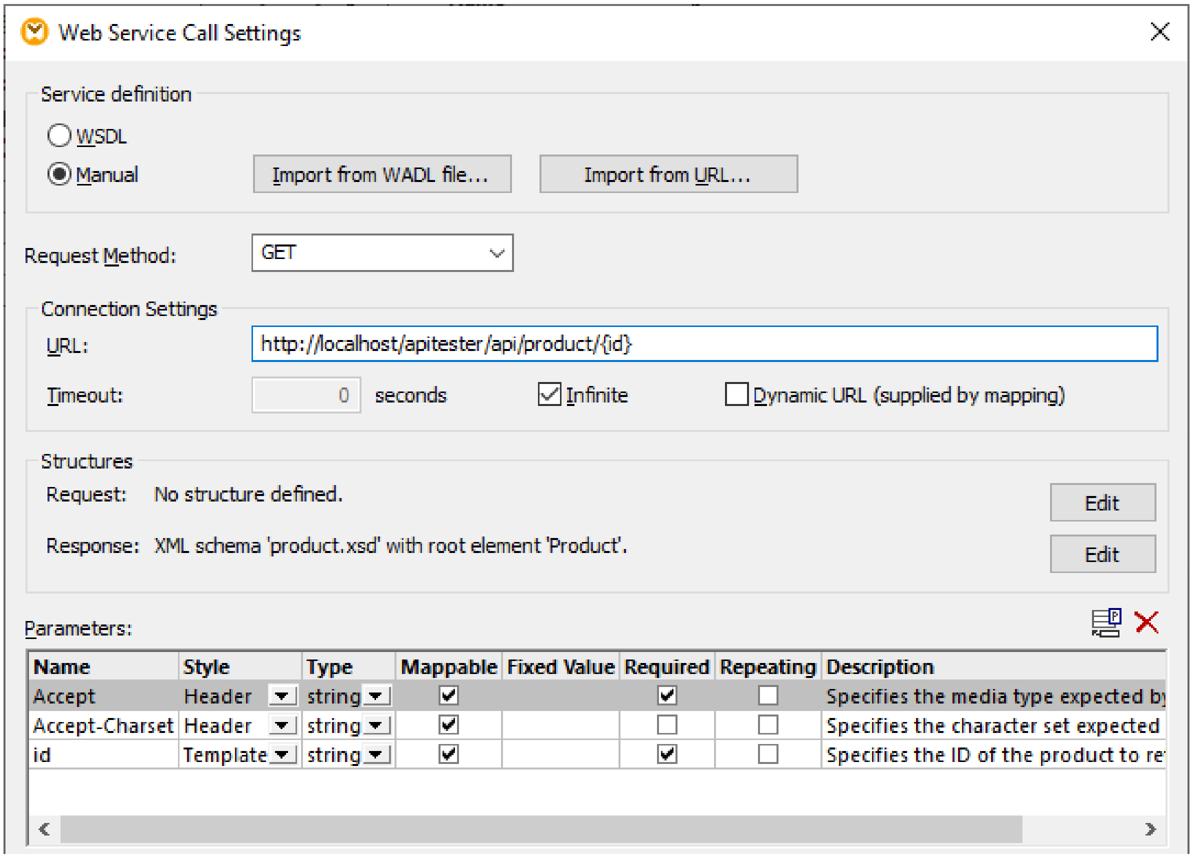
- Example: Calling a REST-Style Web Service 614
- Example: Mapping Data from an RSS Feed 619

4.5.1.1 REST Web Service Parameters

When you need to call a Web service with URL parameters, the parameters must be defined in the **Web Service Call Settings** dialog box. This dialog box also lets you configure the Web service request with custom headers. You can configure the parameter values to be *mappable* (supplied by the mapping) or *fixed* (defined on the dialog box).

To define Web service parameters, take the following steps:

1. Double-click the title bar of the Web service component in the mapping or right-click it and select **Properties**. The **Web Service Call Settings** dialog box opens.



2. Under **Service Definition**, select **Manual**. You can enter the parameters manually or automatically (by importing them from an existing URL [597] or from a WADL file [598]). The imported parameters become available in the **Parameters** table (*see subsection below*), where you can further modify them if required.
3. Use the **Add Parameter** ( ) and **Delete Parameter** ( ) buttons, as required.

Parameters

The subsections below summarize the available parameters in the **Web Service Call Settings** dialog.

Name

Specifies the name of the URL parameter. The parameter name must be unique and may consist of letters, digits, periods (.), hyphens (-) and underscores (_). No spaces are allowed in the parameter name.

Style

Specifies the syntax (style) of the URL parameter.

- Use the *Header* style to add a parameter to the HTTP header when calling the Web service. For example, adding a parameter `Accept` with value `text/xml` is equivalent to specifying the `Accept: text/xml` header, which informs the Web server that MapForce expects the response to be in XML format. For more information about HTTP headers, see the Message Headers Specification.

- Use the *Query* style for URL parameters that define key-value pairs using the **`?key=value&key=value`** format (e.g., `http://example.org/api/products?sort=asc&category=1&page=1`)
- Use the *Template* style for URL parameters enclosed within curly braces (e.g., **`http://example.org/api/products/{id}`**). For such parameters, MapForce escapes the values according to the https://tools.ietf.org/html/rfc6570 RFC 6570 rules.
- Use the *Matrix* style for URL parameters that define key-value pairs in the **`;key=value;key=value;`** format (e.g., `http://example.org/api/products;sort=asc;category=1;page=1;`). To use Boolean matrix parameters, set the style to *Matrix* and the type to `boolean`.
- Select *Body* to send an `application/x-www-form-urlencoded` request structure to a REST service (the request method must be POST). To find out more about the `application/x-www-form-urlencoded` format, see the URL Specification.

⊟ Type

Specifies the data type of the parameter (string, integer, date, etc). This can be any XML Schema type. Note that any value that is not a string is converted to a string when the Web service call takes place. Setting a type is meaningful if you want MapForce to show conversion error messages when you attempt to call a Web service with wrong values.

⊟ Mappable

Select this check box if you want to pass values to this parameter from the mapping. This option is mutually exclusive with the *Fixed Value* option.

⊟ Fixed value

Specifies the value of the parameter. Applicable only if the parameter has a constant value. Not applicable if the parameter is mappable (*see previous option*).

⊟ Required

Select this check box if the parameter is required by the Web service. For parameters that are required and also mappable, MapForce enforces validation checks: An error message is displayed if the parameter does not have a value.

⊟ Repeating

Specifies whether the parameter is single-valued or may have multiple values. This setting is applicable only for mappable parameters. This option enables you to pass multiple values in the same Web service call by means of a single parameter. When you select the check box, you can connect a sequence of values to the parameter structure node in the mapping. MapForce will then handle the sequence of values depending on the style of the parameter:

- For *Template* parameters, the values will be supplied to the Web service as comma-separated (e.g., `http://example.org/api/products/1,2,3`).
- For *Query* parameters, the parameter name will be repeated for each value (e.g., `http://example.org/api/products?color=red&color=green&color=blue`).
- If the style is *Matrix*, multiple values will be separated by comma (e.g., `http://example.org/api/products;color=red;color=green;color=blue;size=big;size=small;`).
- If the style is *Header*, the HTTP header will be repeated for each value.

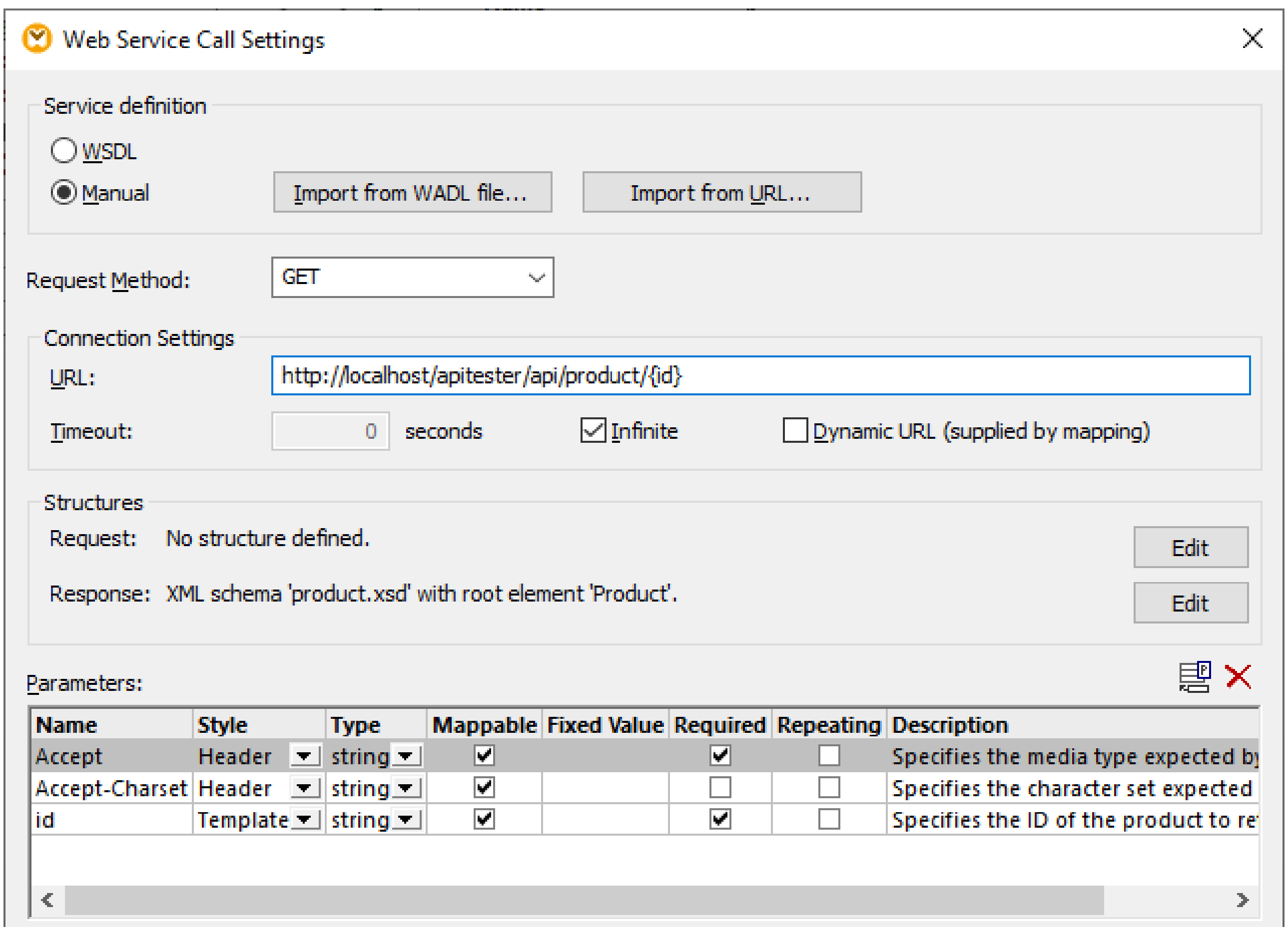
⊟ Description

Specifies the optional description of the parameter. If the parameter is mappable, the description entered here appears in the mapping component as an annotation next to the mapping item.

Example 1: Header and template parameters

The Web service illustrated below retrieves a product by its identifier (`id`) using the HTTP GET method. The URL of the Web service specifies the `id` parameter in the curly brackets. Notice that the `id` parameter exists in the Parameters table and has the style *Template*. It is also **mappable**: this causes the parameter to appear on the mapping as a structure node to which you can connect the actual value of `id` (which could be, for example, taken from a database, a file, or a constant). At mapping execution runtime, this parameter would be replaced with the actual value; so, if the value is "1", the URL becomes `http://example.org/api/products/1` .

To supply a constant `id` value instead of a value from the mapping, clear the check box "Mappable" and enter the value in the "Fixed value" column.

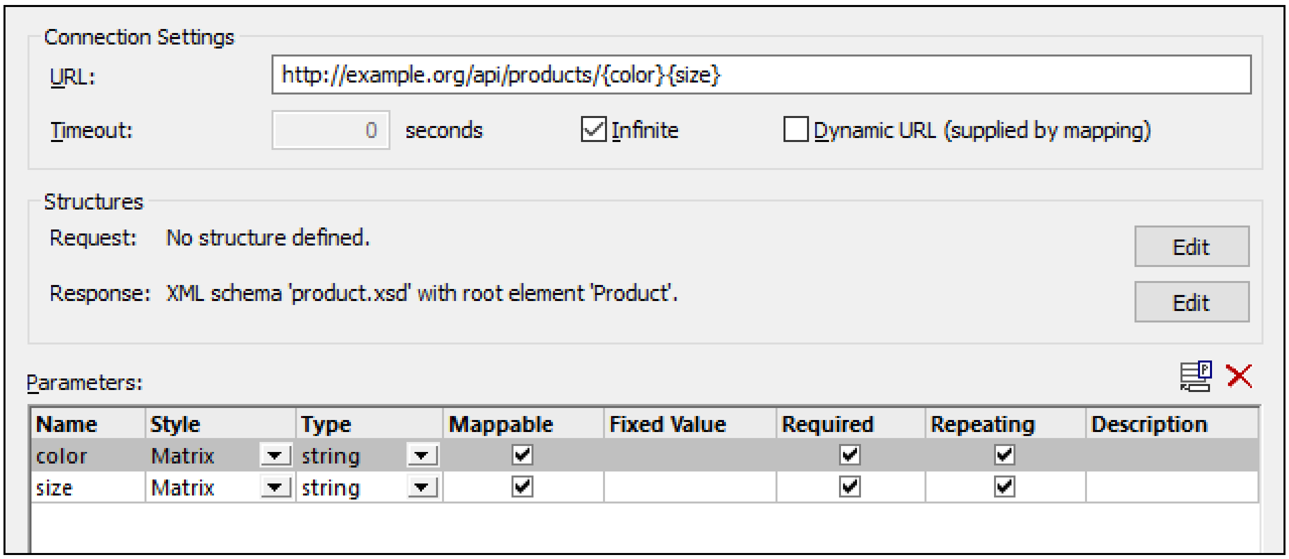


The parameters `Accept` and `Accept-Charset` have the *Header* style. These parameters are used to call the Web service with custom request headers. There are two ways to supply the header value:

- Leave the option **Mappable** checked and supply the custom header value from the mapping, or
- Clear the option **Mappable** and enter the value directly in the "Fixed value" column.

Example 2: Matrix parameters

The Web service illustrated below retrieves a list of products that match the color and size supplied as arguments. The style of the parameters is "Matrix", so they are defined both as placeholders inside the URL and in the mapping table. Notice that the parameters are mappable and the "Repeating" option is checked. This means that their value will be read from some sequence of values on the mapping (for example, a list of rows inside a text file, an XML node, or a database column) and supplied to the Web service at runtime. Thus, a URL such as the one below would become `http://example.org/api/products/;color=red;color=blue;size=big;size=small`, provided that the mapping supplies `red` and `blue` as colors, and `big` and `small` as size.



If you need the URL to be `http://example.org/api/products;color=red,blue;size=big,small`, do the following:

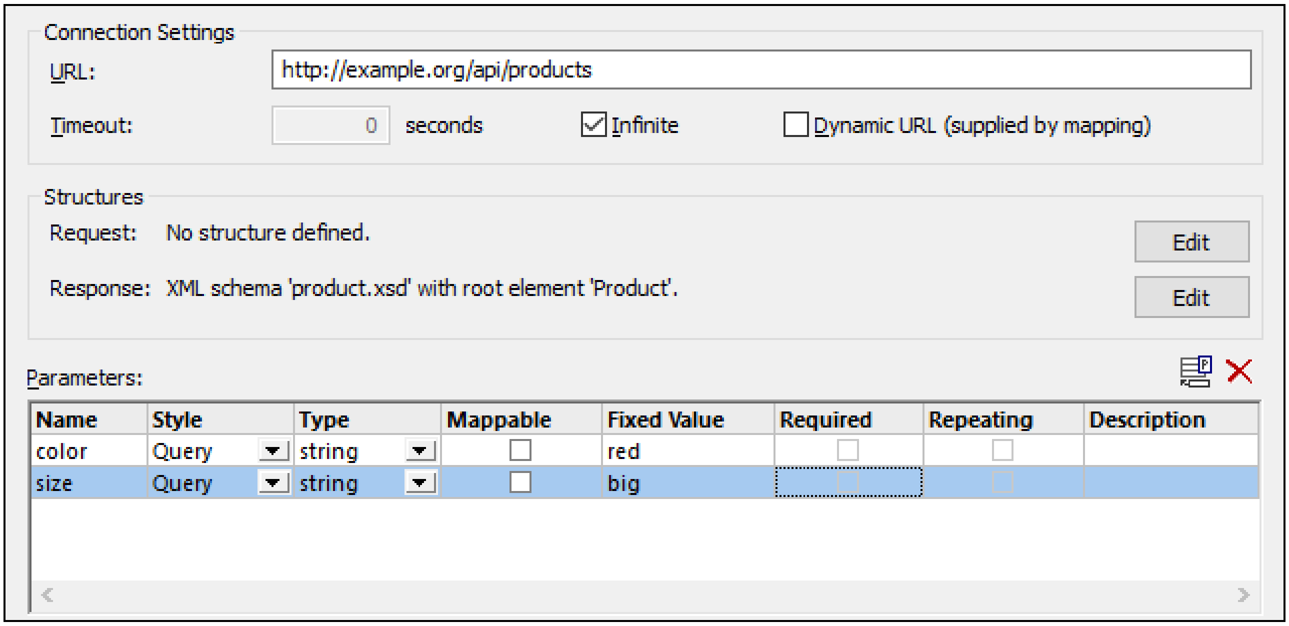
1. Enter the URL as `http://example.org/api/products;color={color};size={size}`
2. In the Parameters table, change the parameter style to *Matrix*.

If you need the URL to be `http://example.org/api/products?color=red,blue&size=big,small`, do the following:

1. Enter the URL as `http://example.org/api/products?color={color}&size={size}`
2. In the Parameters table, change the parameter style to *Matrix*.

Example 3: Query parameters

The Web service illustrated below also retrieves a list of products that match the color and size supplied as arguments, this time using the style "Query". For this style, it is not necessary to define the parameters as placeholders in the URL, so they are defined only in the "Parameters" table. In this example, let's assume that the parameter values are fixed and must not be supplied from the mapping. To achieve this, the parameter values were entered under "Fixed Values", and the "Mappable" option is unchecked. Thus, at mapping runtime, the URL below would become **`http://example.org/api/products?color=red&size=big`**.



4.5.1.2 Importing Web Service Parameters from URL

If the URL to the Web service uses parameters, you can instruct MapForce to parse the URL and import any parameters automatically. When you use this option, any template, matrix, or query parameters extracted from the URL become available in the Parameters table of the Web Service Call Settings dialog box, where you can further manipulate them as necessary.

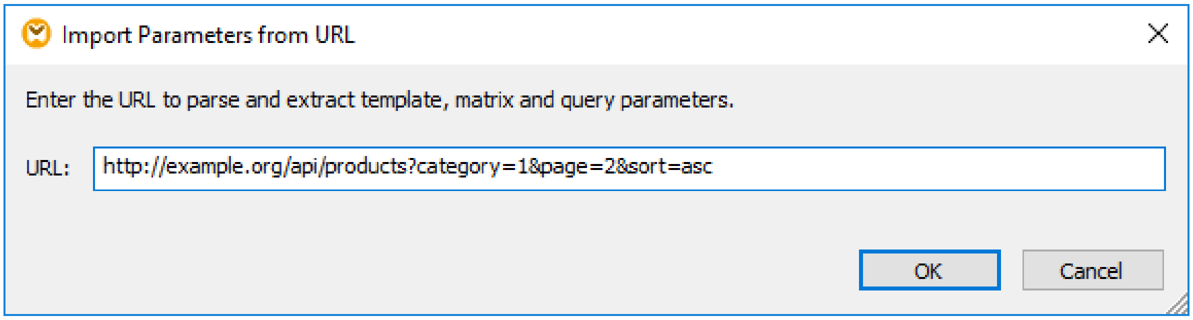
The following are examples of various URL styles:

URL with template parameters	`http://example.org/api/products/{id}`
URL with matrix parameters	`http://example.org/api/products;sort=asc;category=1;page=1`
URL with query parameters	`http://example.org/api/products?category=1&page=2&sort=asc`

Note: A URL must begin with either "http://" or "https://" to be parseable.

To import Web service parameters from a URL:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings 606).
2. Under **Service Definition**, select **Manual**.
3. Click **Import from URL**.



4. Enter or paste the URL to the Web service in the text box, and click OK.

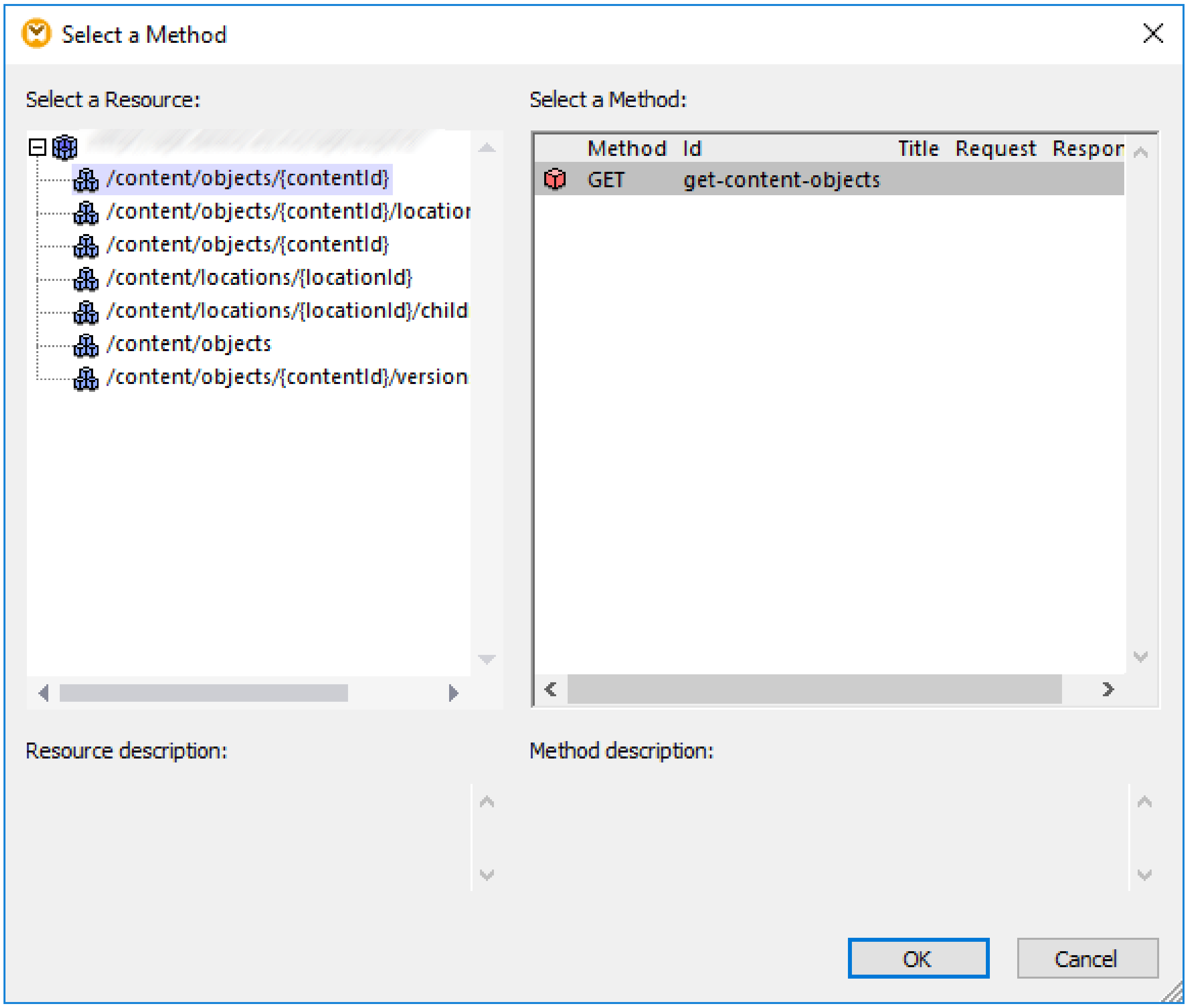
4.5.1.3 Importing Web Service Information from WADL

The Web Application Description Language (WADL) is one of the ways to represent the contents of the responses and requests used in a web application, including RESTful Web services. MapForce supports importing Web service information from WADL version 2009 (https://www.w3.org/Submission/2009/SUBM-wadl-20090831/).

If the Web service has a WADL file, you can import the Web service definition from the WADL file, rather than entering it manually. When you import information from a WADL file, the parameters and the XML message body structure are populated automatically on the Web Service Call Settings dialog box. After the import, you can modify the parameters and the response/request structure manually, if necessary. This will not affect the underlying WADL file.

To import Web service information from a WADL file:

1. Open the Web Service Call Settings 606 dialog box.
2. Under Service Definition, select **Manual**.
3. Click **Import from WADL file** and browse for the WADL file.



4. On the dialog box, select the resource and method, and click **OK**.

4.5.1.4 Defining the Request Structure

When calling Web services that expect content in the body part of the request message, you can set the request structure to be one of the following:

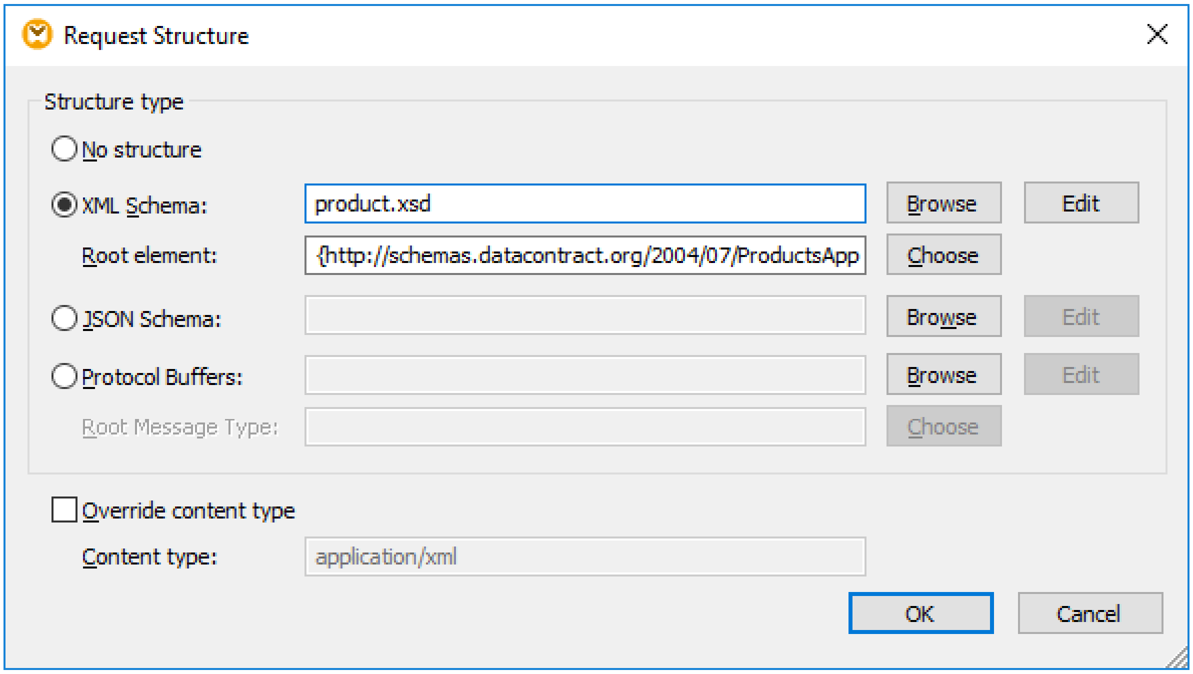
- No structure
- XML
- JSON
- Protocol Buffers files, see Protocol Buffers 811

Note: In Web service calls created with MapForce, you can use standard JSON as request or response structure; JSON Lines or JSON5 structures are not supported.

The structure selected here determines the mappable items that appear on the mapping as children of the **Body** item.

To define the request structure:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings 606).
2. Under Service Definition, select **Manual**.
3. Under **Structures**, click **Edit** next to **Request**.



Tip: As a shortcut to display this dialog box, click the button in the request part of the Web service component on the mapping.

4. Select one of the following options:

a) If the Web service expects no content in the body part, or if you would like to provide the raw body (MIME entity) directly from the mapping, select **No structure**.

b) For XML, obtain the XML or DTD schema of the request from the provider of the Web service. If you select an XML instance file, it must have a valid schema reference. Some XML schemas define elements with global declaration (that is, elements whose parent is the `schema` element). For such schemas, you can choose what element in the schema should be the root element of the mapping structure in MapForce. To do this, click **Choose**, and then, in the dialog box that appears, select the desired root element.

c) For JSON, obtain the JSON schema of the request from the provider of the Web service, or generate it from a sample JSON file with a tool such as XMLSpy.

Tip: If you have a sample XML or JSON request file but don't have a schema file, you can use XMLSpy to generate the schema file. If necessary, XMLSpy can also convert your instance file from XML to JSON, or vice versa.

b) For Protocol Buffers files, obtain from the Web service provider the `.proto` file describing the binary file, and then click **Browse** to select the file. The **Root Message Type** field is applicable if the Protocol Buffers file contains multiple message types. Click **Choose** to select the message type that should be at the structure's root.

5. Optionally, select the **Override content type** check box if you need to override the **Content-Type** header set implicitly by MapForce. You can also enter parameters after the content type, for example: `Content-Type: text/html; charset=utf-8`.

> When you save the mapping, any absolute paths on this dialog box will become relative to the mapping design file (.mfd) if the check box **Save all file paths relative to mfd** is selected in component settings, see Web Service Call Settings 606.

4.5.1.5 Defining the Response Structure

When calling Web services that return content in the body part of the response message, you can set the response structure to be one of the following:

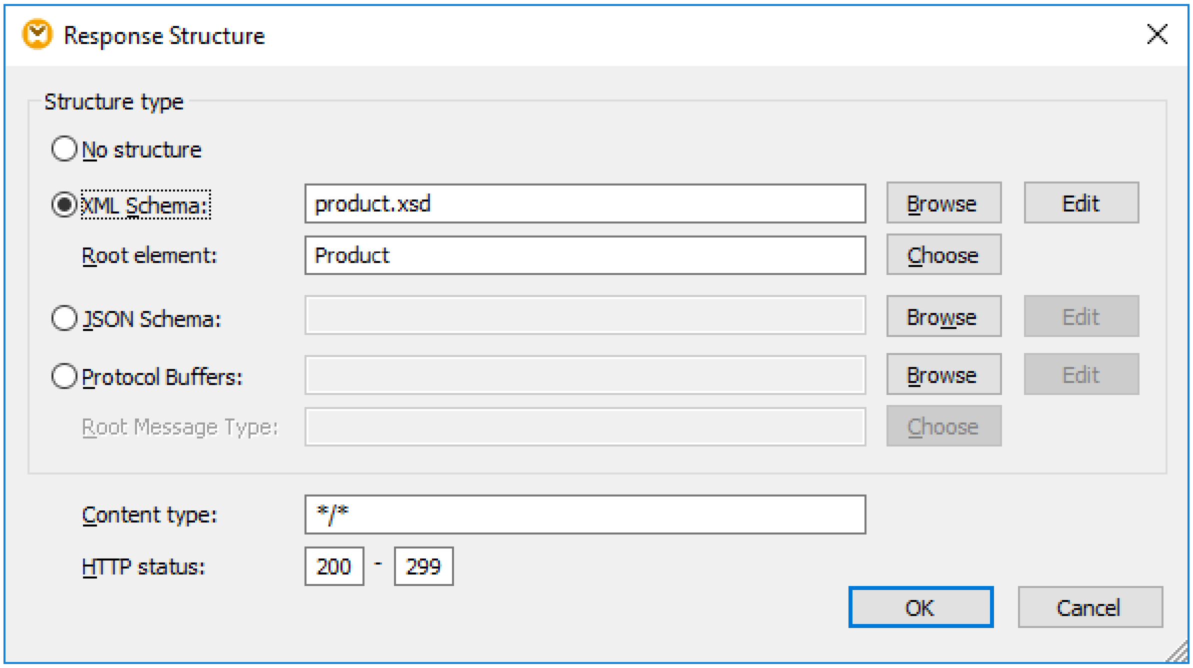
- No structure
- XML
- JSON
- Protocol Buffers files, see Protocol Buffers 811

Note: In Web service calls created with MapForce, you can use standard JSON as request or response structure; JSON Lines or JSON5 structures are not supported.

The structure selected here determines the mappable items that appear on the mapping as children of the **Body** item.

To define the response structure:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings 606).
2. Under Service Definition, select **Manual**.
3. Under **Structures**, click **Edit** next to **Response**.



Tip: As a shortcut to display this dialog box, click the button in the request part of the Web service component on the mapping.

4. Select one of the following options:

a) If the Web service returns no content in the body part, or if you would like to process the raw body (MIME entity) directly from the mapping, select **No structure**.

b) For XML, obtain the XML or DTD schema of the request from the provider of the Web service. If you select an XML instance file, it must have a valid schema reference. Some XML schemas define elements with global declaration (that is, elements whose parent is the `schema` element). For such schemas, you can choose what element in the schema should be the root element of the mapping structure in MapForce. To do this, click **Choose**, and then, in the dialog box that appears, select the desired root element.

c) For JSON, obtain the JSON schema of the request from the provider of the Web service, or generate it from a sample JSON file with a tool such as XMLSpy.

Tip: If you have a sample XML or JSON request file but don't have a schema file, you can use XMLSpy to generate the schema file. If necessary, XMLSpy can also convert your instance file from XML to JSON, or vice versa.

b) For Protocol Buffers files, obtain from the Web service provider the `.proto` file describing the binary file, and then click **Browse** to select the file. The **Root Message Type** field is applicable if the Protocol Buffers file contains multiple message types. Click **Choose** to select the message type that should be at the structure's root.

Optionally, fill the **Content type** and **HTTP status** text boxes if you need to filter a response MIME entity by media (content) type and HTTP status code. This is useful if you need conditional handling depending on the

HTTP status code and the content type returned by the Web service call. For an example, see [Handling the HTTP Response Conditionally](603).

By default, the **Content type** filter is set to `*/*`, which matches all media types. The default HTTP status codes are **200** through **299**. If the Web service call returns some other status code that isn't handled explicitly, the mapping execution will fail with an error.

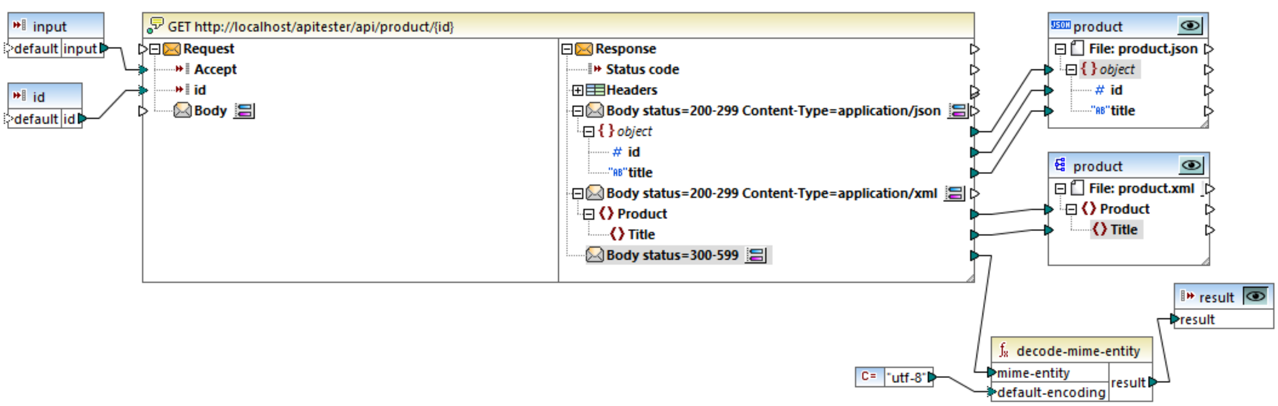
> When you save the mapping, any absolute paths on this dialog box will become relative to the mapping design file (.mfd) if the check box **Save all file paths relative to mfd** is selected in component settings, see [Web Service Call Settings](606).

4.5.1.6 Handling the HTTP Response Conditionally

Let's assume that a Web service returns either JSON or XML content type in the response body, depending on the value of the **Accept** HTTP request header. Additionally, in case of an error (status code > 299), the Web service may return an undefined content type. This is a case where it might be useful to handle the response conditionally based on the HTTP status code and content type.

To handle both the JSON and the XML response from the Web service in the same mapping, you could create multiple body items, by right-clicking the body item on the component and selecting **Add Body Node After** from the context menu.

On the mapping, a Web service component with multiple **body** items configured for the scenario above could look as follows:



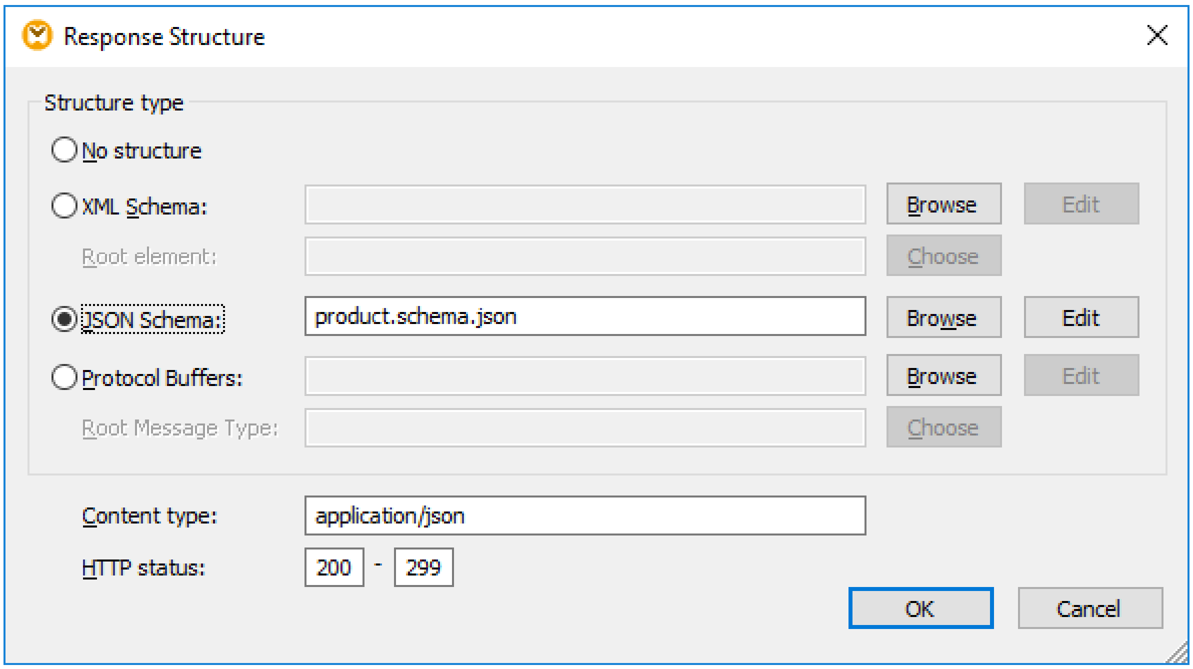
The mapping illustrated above takes two input parameters. The first parameter provides the expected content type and its value can be `application/json`, or `application/xml`. The second parameter provides the numeric ID of the product to be retrieved.

If you provide the value `application/json` as input parameter, click the **Preview** button of the JSON target component. Otherwise, if the input parameter value is `application/xml`, click the **Preview** button of the XML target component. The respective output file (JSON or XML) will be generated if the Web service call is successful.

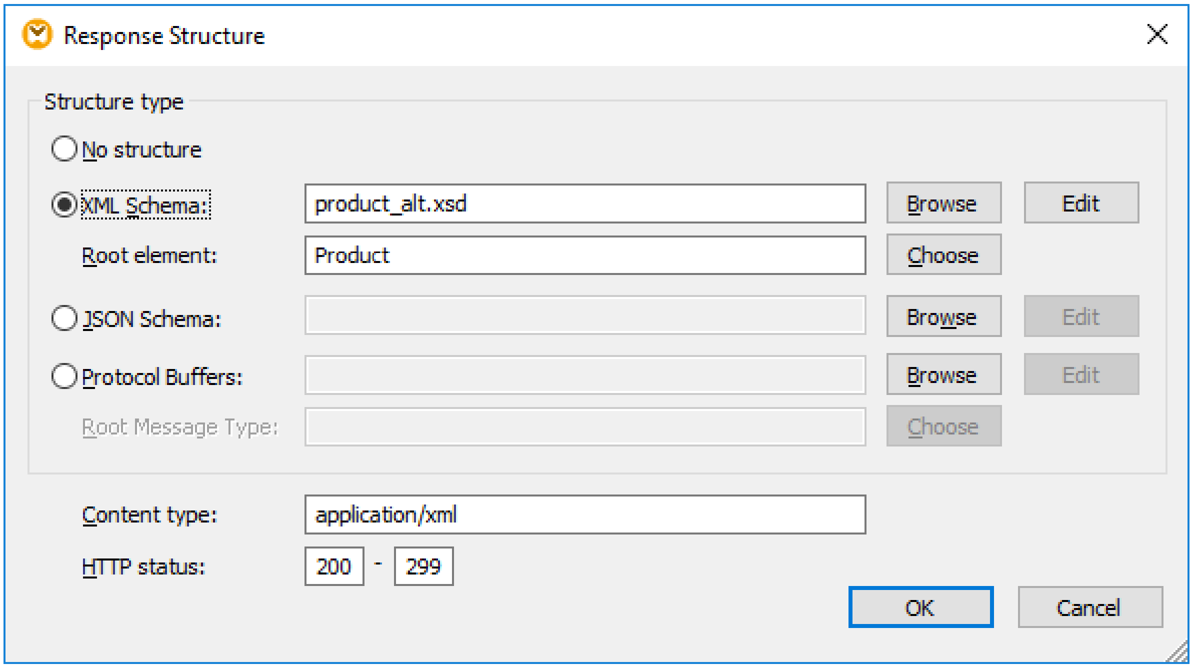
If the Web service call returns a status code greater than 300 (error), the HTTP response reaches the third **body** item. Since we don't know the exact content type, we choose to decode it with the help of the decode-mime-entity 1133 function. The Web service does not supply the character set value in the response header, which is why the constant value **utf-8** is connected to the decoding function.

Each of the three response body items on the component above are configurable by clicking the button next to each body:

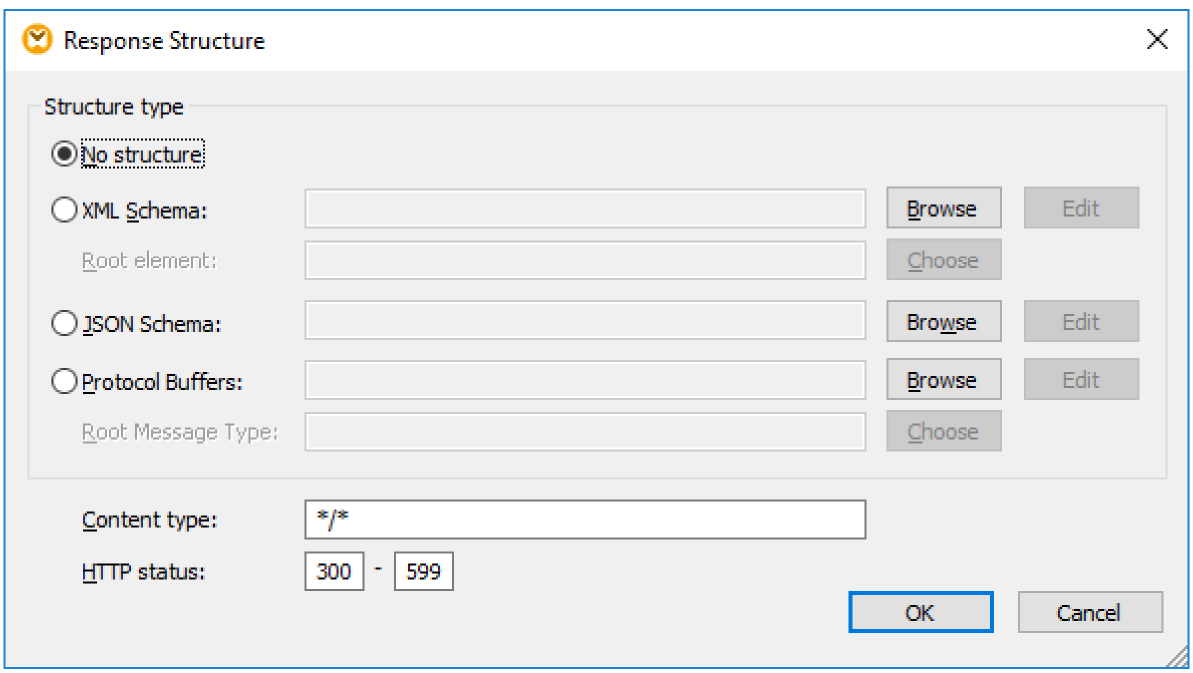
1. The first body handles content returned as `application/json` if the status code is between 200 and 299. Also, the schema of the expected JSON response structure is set.



2. The second body handles content returned as `application/xml` if the status code is between 200 and 299. Also, the schema of the expected XML response structure is set.



3. The third body handles content returned as `*/*` if the Web service returns an erroneous status code (between 300 and 599).



4.5.2 WSDL-Based Web Services

This topic describes how to add to a mapping a call to a WSDL-style Web service. For instructions about adding a generic Web service to the mapping, see Adding a Web Service Call (REST-Style) 587.

Before adding a WSDL-style Web service call to the mapping area, make sure that you have the Web Service Definition Language (.wsdl) that describes the Web service to which you are attempting to connect.

To add a call to a WSDL-style Web service:

1. On the **Insert** menu, click **Web Service Function**. (Alternatively, click the **Insert Web service function** toolbar button).
2. Under Service Definition, click **WSDL** if this option is not already selected.
3. Under WSDL Settings, click Browse and select the Web Service Definition Language (.wsdl) file. Two sample .wsdl files are available in the MapForce Example folder, one for each version of WSDL (1.0 and 2.0). The corresponding paths are as follows: <Documents>\Altova\MapForce2023\MapForceExamples\TimeService\TimeService.wsdl and <Documents>\Altova\MapForce2023\MapForceExamples\TimeServiceWsdl2\TimeService20.wsdl.
4. If the .wsdl file defines multiple services, operations, and endpoints, click **Choose** and select the required options.
5. If necessary, define other settings as required (see Web Service Call Settings 606).
6. Click **OK**.

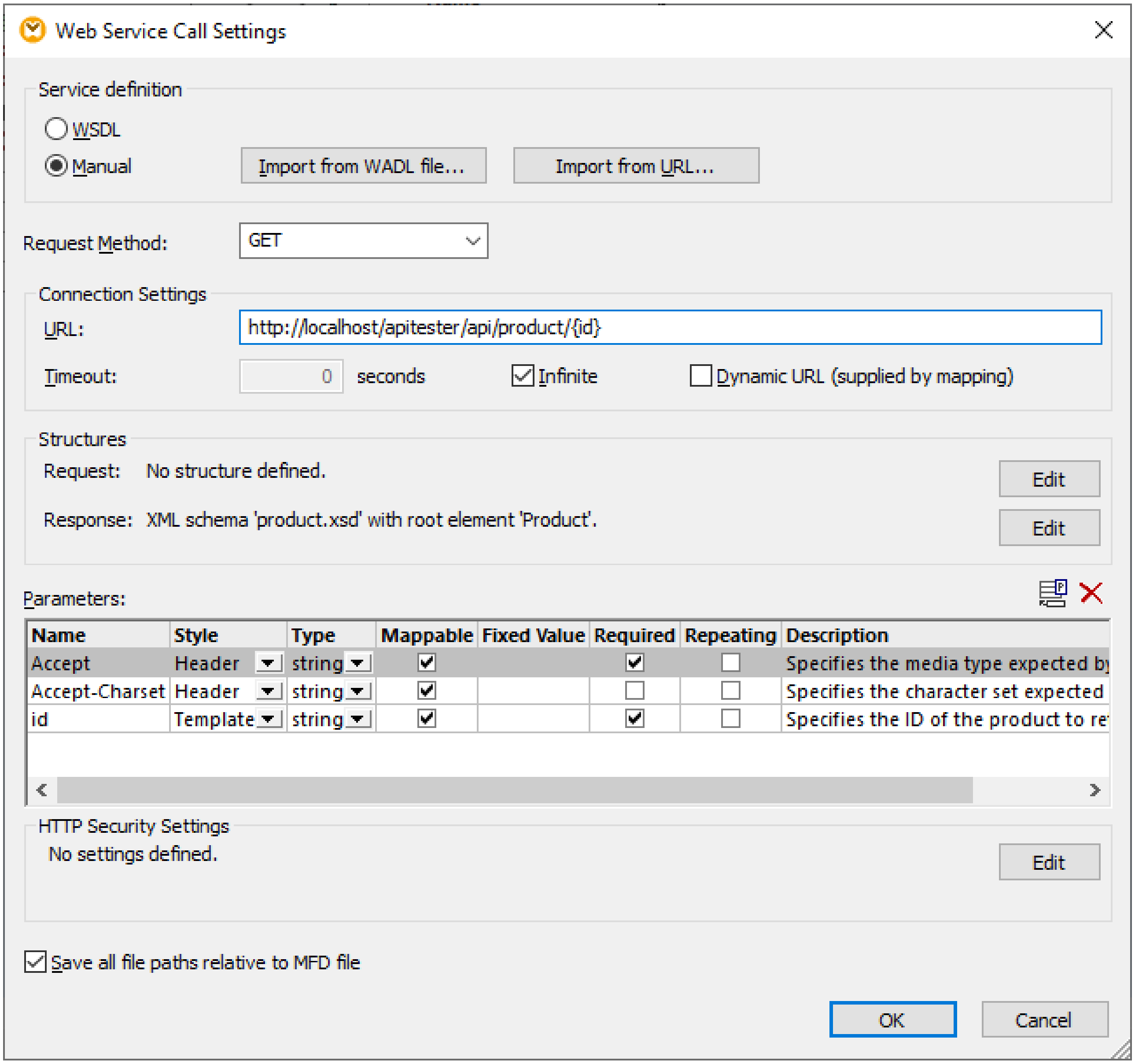
See also Example: Calling a SOAP Web Service 624.

4.5.3 Web Service Call Settings

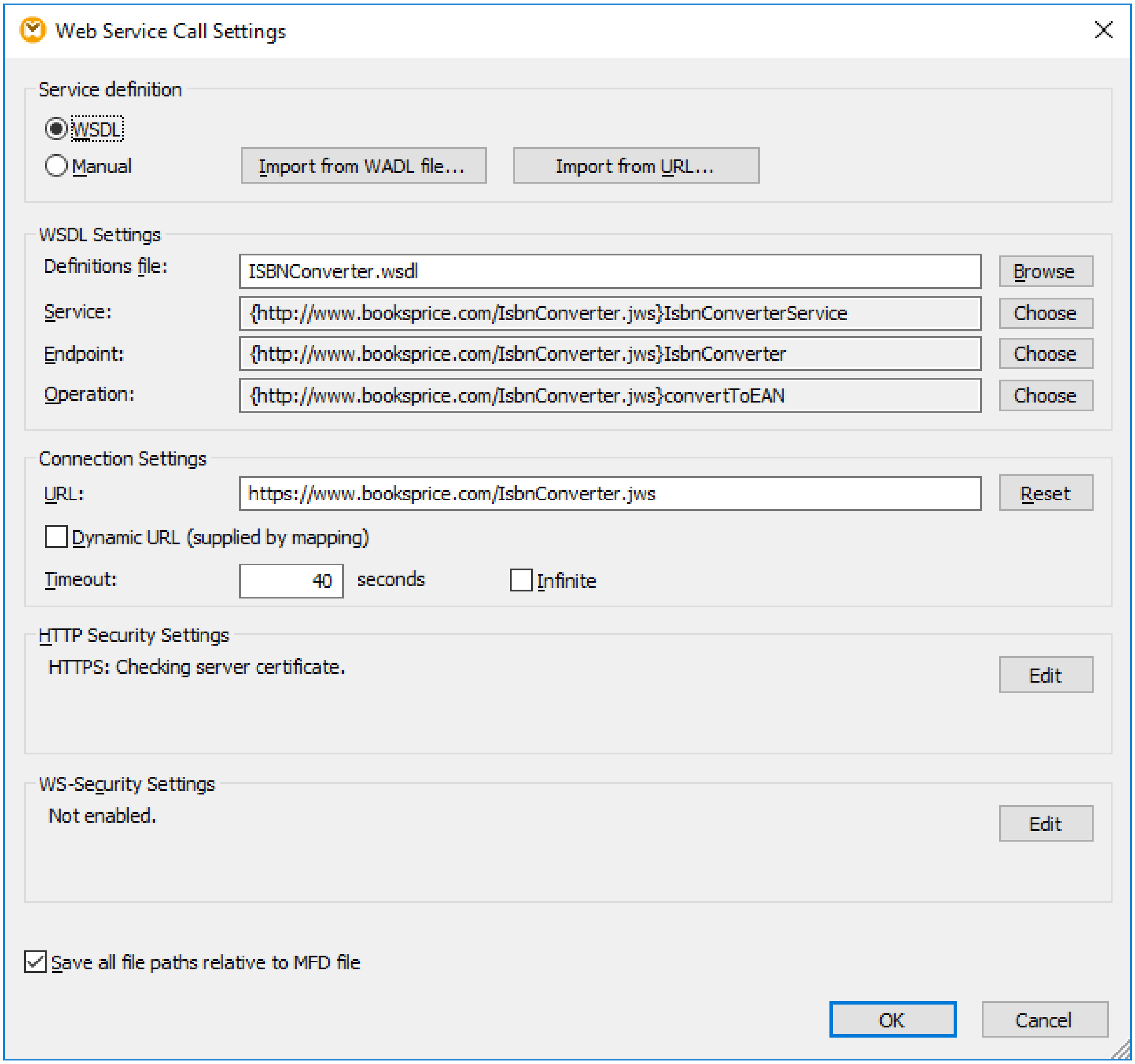
You can change the settings applicable to Web services from the Web Service Call Settings dialog box. To open this dialog box, do one of the following:

- Click the Web service component on the mapping, and, then, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

The list of available settings depends on whether the **Service definition** option at the top of the dialog box is set to **WSDL** or **Manual**. The following image illustrates the settings applicable for a generic HTTP Web service:



The following image illustrates the settings applicable for a WSDL-style Web service:



The available settings are described below.

Service Definition

If the settings you define apply to a WSDL-style Web service, select **WSDL**. Otherwise, select **Manual**.

When **Manual** is selected, you must enter the Web service settings manually into the dialog box. Optionally, if you have the WADL file of the Web service, you can import the Web service settings by clicking **Import from WADL** (see Importing Web Service Information from WADL 598). Also, if you want to extract the Web service parameters from a URL, click the **Import from URL** button (see Importing Web Service Parameters from URL 597).

WSDL Settings

This group of settings is applicable only for WSDL-style Web services.

WSDL Definitions	Specifies the Web Service Definition Language (WSDL) file of the web service to be called.
Service	Specifies the name of the Web service to be called. If the WSDL file defines multiple web services, click **Choose** to select the required one.
Endpoint	Specifies the endpoint (or port) of the Web service to be called. If the selected Web service defines multiple web service endpoints, click **Choose** to select the required one.
Operation	Specifies the operation of the Web service to be called. If the selected endpoint defines multiple web service operations, click **Choose** to select the required one.

Request Method

This setting is applicable only for generic (not WSDL) Web services. It defines the HTTP method (verb) used by the service (for example, GET, POST, PUT, and so on).

Connection Settings

The **URL** text box specifies the address (URL) of the Web service. **Timeout** defines the time interval after which the Web service call will time out if there is no response from the server. Select **Infinite** if the call should wait for a response for an indefinite amount of time.

If you would like to supply the URL of the Web service from the mapping, select the **Dynamic URL (supplied by mapping)** check box. This creates an additional input connector on the Web service component on the mapping, to which you can connect some input which provides the URL of the Web service.

Note: When the **Dynamic URL (supplied by mapping)** check box is selected, the **URL** text box becomes disabled (grayed out), and you have to supply the URL from the main mapping. For REST-style Web services, this also means that you can define only Header-style parameters in the "Parameters" table.

Parameters

This group of settings is applicable only for generic (not WSDL) Web services. The **Parameters** table specifies the URL parameters with which MapForce will call the Web service (see Defining Web Service Parameters 592).

Structures

This group of settings is applicable only for generic (not WSDL) Web services. For more information, see Defining the Request Structure 599 and Defining the Response Structure 601.

HTTP Security Settings

If the Web service requires authentication (either HTTP or HTTPS), click **Edit** to specify the required authentication settings (see Setting HTTP Security 610).

WS-Security Settings

These settings are applicable only for WSDL-style Web services. If the Web service uses SOAP security, click **Edit** to specify further settings (see Setting WS-Security 612).

Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative and Absolute Paths 84.

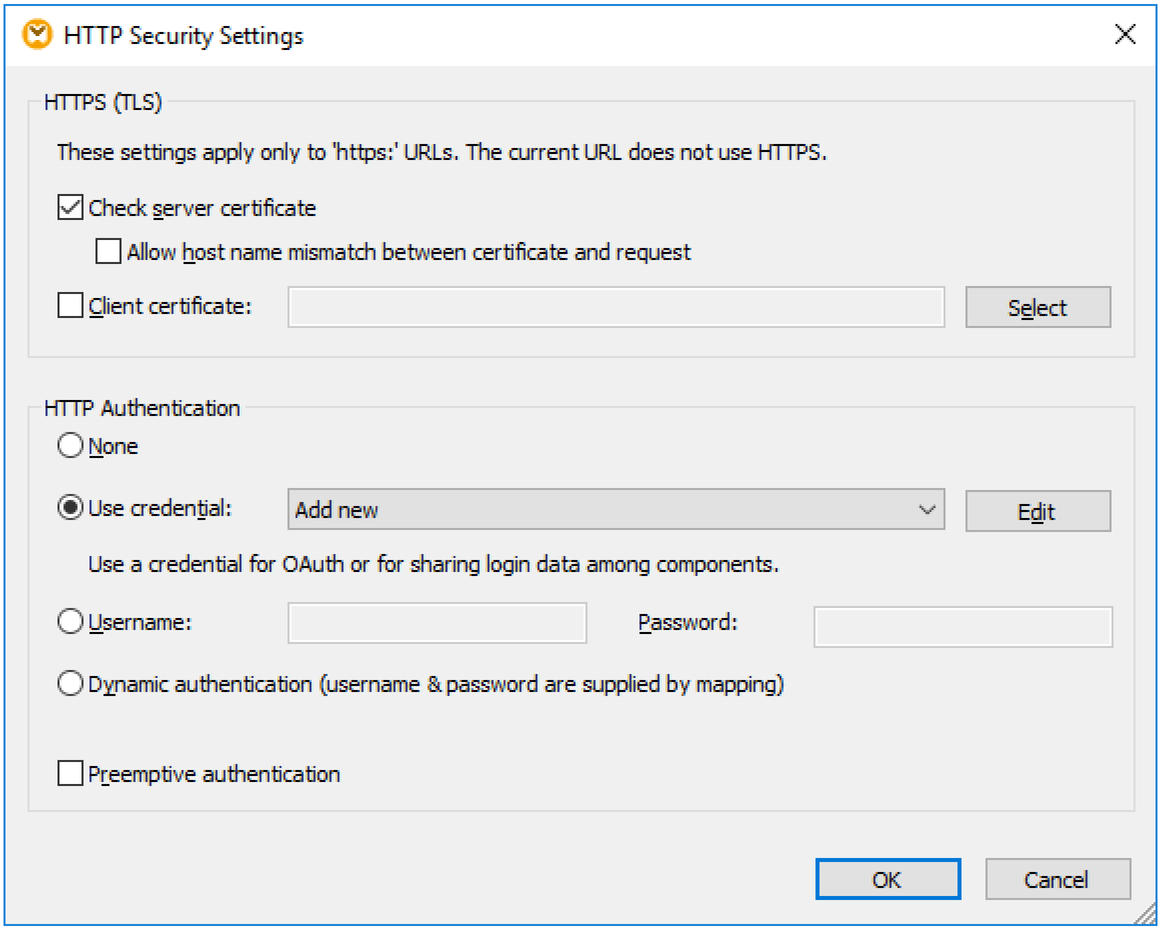
4.5.4 Setting HTTP Security

You can configure any Web service component with respect to various scenarios related to HTTP security, for example:

- The Web service is called through HTTPS and requires a client certificate
- The server uses an incorrect certificate and you would like to allow a host name mismatch between certificate and request
- The Web service requires basic HTTP authentication or OAuth 2.0 authorization.

To set HTTP security:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings 606).
2. Click the **Edit** button next to HTTP Security Settings.



HTTP Security Settings dialog box

The "HTTPS (TLS)" group of options applies for Web services called through HTTPS.

Check Server Certificate	This check box is selected by default, meaning the MapForce is configured to check the certificate of the server before proceeding with the request. When this option is enabled, the Web service request (and the mapping) will fail if the server is not trusted, or if your operating system is not configured to trust the Web server. It is not recommended to switch this option off unless you have a good reason to do so. See also Digital Certificate Management 640.
Allow host name mismatch between certificate and request	Sometimes a server certificate issued for a particular host name (for example, *www.example.com*) is installed on a different host name (for example, *example.com*). Select this check box to proceed with authentication even if the host name of the certificate does not match the host name called by the Web service.

Client certificate	Click **Select** to choose a client certificate from the **Current User\Personal** certificate store. This assumes the client certificate already exists in the **Current User\Personal** certificate store; otherwise, you can import it using the Certificates snap-in (see Accessing the Certificate Stores on Windows 646). If the mapping will be deployed for execution to another operating system, the same certificate must be installed on the target operating system as well. For further information, see Digital Certificate Management 640.

The "HTTP Authentication" group of options applies if the Web service requires HTTP authentication. For more information about these options, see Running Mappings with Authentication 1297.

None	Select this option if the Web service does not require any kind of authentication.
Use credential	Select this option to configure HTTP or OAuth 2.0 credentials. For more information, see Credentials 1297.
Username, Password	Select this option if you would like to store the username and password in encrypted form in the mapping file. This option is supported only for backward compatibility, see Stored Username and Password 1311.
Dynamic authentication	Select this check box if you would like to supply the username and password from the mapping instead of entering them into this dialog box. For more information, see Dynamic Authentication 1309.
Preemptive authentication	Select this check box if the Web service is configured to expect authentication data in the first call. Otherwise, MapForce attempts access without username and password and will use them if the server requires authorization (HTTP status 401).

4.5.5 Setting WS-Security

The WS-Security settings must be configured if the Web service is protected by WS-Security and requires that you provide the **UsernameToken** security token.

Conventions

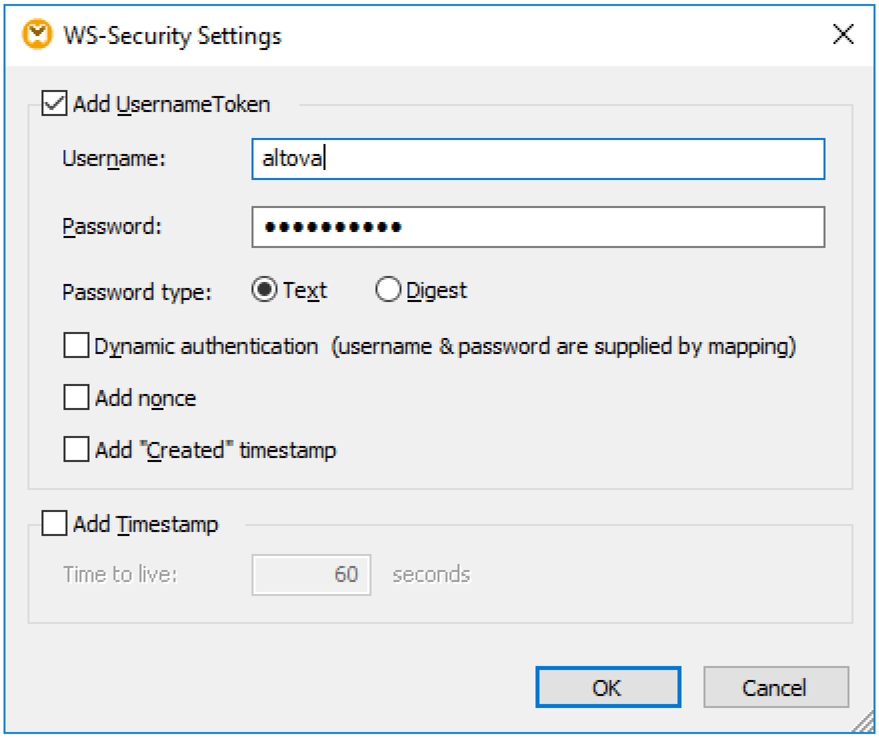
The following abbreviations for the namespaces applicable to Web services are used in this topic:

Prefix	Namespace
wsse	`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd`

wsu	`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`

To set WS-Security:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings [606]).
2. Click the **Edit** button next to WS-Security Settings.



WS-Security Settings dialog box

Add UsernameToken	A UsernameToken is an optional WS-security element present in the header of the SOAP message. The UsernameToken is used by the Web server to authenticate the caller of the Web service. In the **wsse** specification (see Conventions [612]), this field corresponds to: `/wsse:UsernameToken`
Username	Enter the username included in the UsernameToken. In the **wsse** specification (see Conventions [612]), this field corresponds to: `/wsse:UsernameToken/wsse:Username`
Password	Enter the text of the password included in the UsernameToken.

	In the **wsse** specification (see Conventions 612), this field corresponds to: `/wsse:UsernameToken/wsse:Password`
Password type	Select the type of password included in the UsernameToken. Select **Digest** if the Web server expects the password in this mode; otherwise select **Text**. In the **wsse** specification (see Conventions 612), this field corresponds to: `/wsse:UsernameToken/wsse:Password/@Type`
Dynamic authentication	Select this check box if you would like to supply the username and password from the mapping instead of entering them into this dialog box.
Add nonce	Select this check box if you want to add a nonce to the Username token. A nonce is a random value which uniquely identifies each UsenameToken to provide additional security. If you enable this option, it is recommended to enable the **Add "Created" timestamp** option as well. In the **wsse** specification (see Conventions 612), this field corresponds to: `/wsse:UsernameToken/wsse:Nonce`
Add "Created" timestamp	Select this option to add a timestamp to each nonce. In the **wsse** specification (see Conventions 612), this field corresponds to: `/wsse:UsernameToken/wsu:Created`
Add Timestamp	Select this check box if you want to enable the time-to-live (TTL) value for the SOAP message (see the next option). In the **wsu** specification (see Conventions 612), this field corresponds to: `/wsu:Timestamp`
Time to live	Enter the time-to-live (TTL) for the SOAP message to diminish the chance of someone intercepting the message and replaying it.

4.5.6 Example: Calling a REST-Style Web Service

This example shows you how to call a generic HTTP Web service from MapForce. The description of the Web service called in this example can be found at https://app.swaggerhub.com/apis/FAA/ASWS/1.1.0. This Web

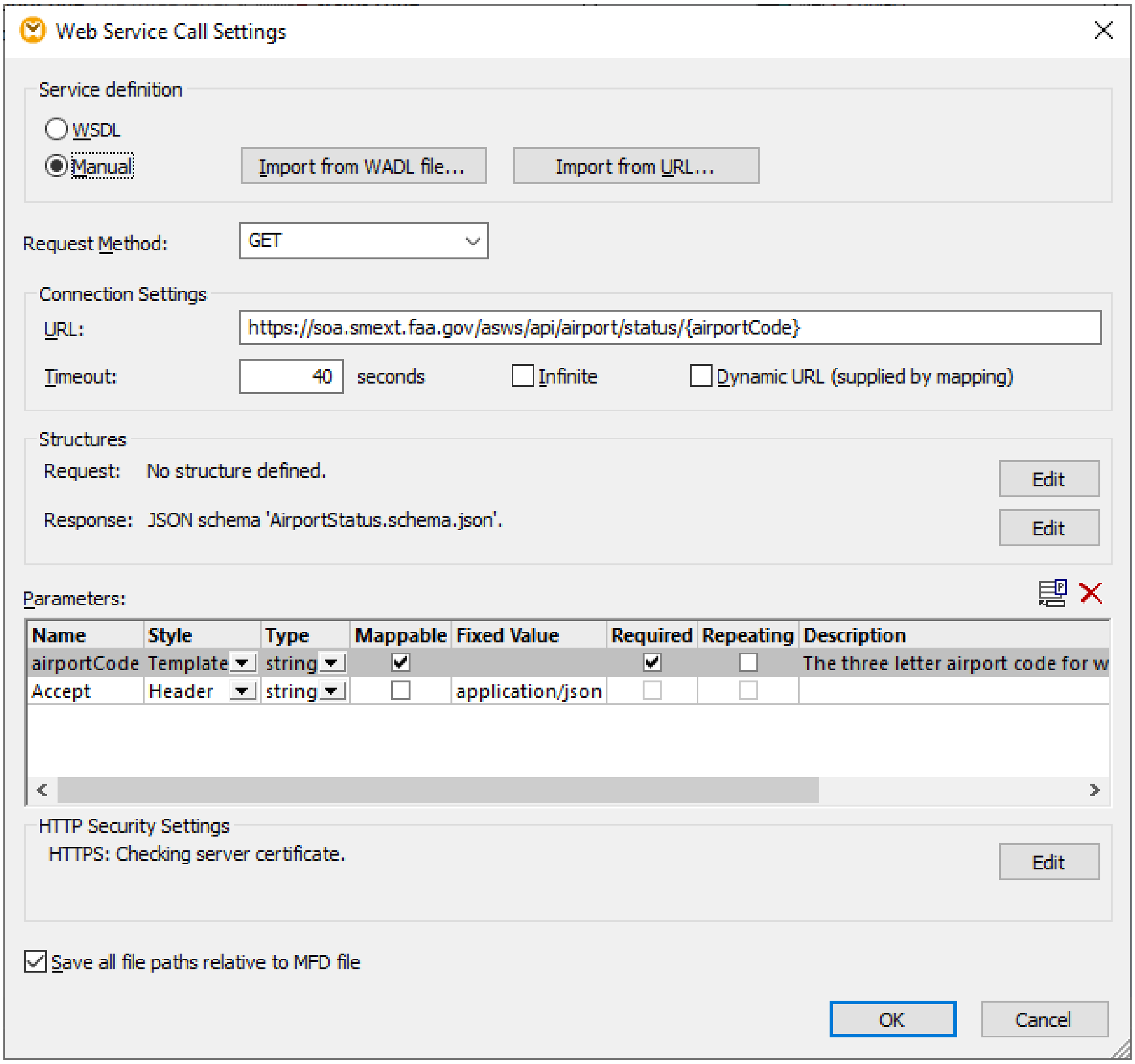
service returns the current status of any major US airport, as an XML or JSON structure, accepting the three-letter airport code as argument (for example, "SFO", "IAD", "ABE", "DFW", etc). The example is accompanied by a mapping design file, which is available at the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\AirportStatus.mfd .

In this example, the response of the Web service is in JSON format, and is mapped to a JSON file. Therefore, the JSON schema of the Web service response structure will be required. For convenience, the JSON schema was generated with XMLSpy from a sample response supplied by the Web service provider. It is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\AirportStatus.schema.json**.

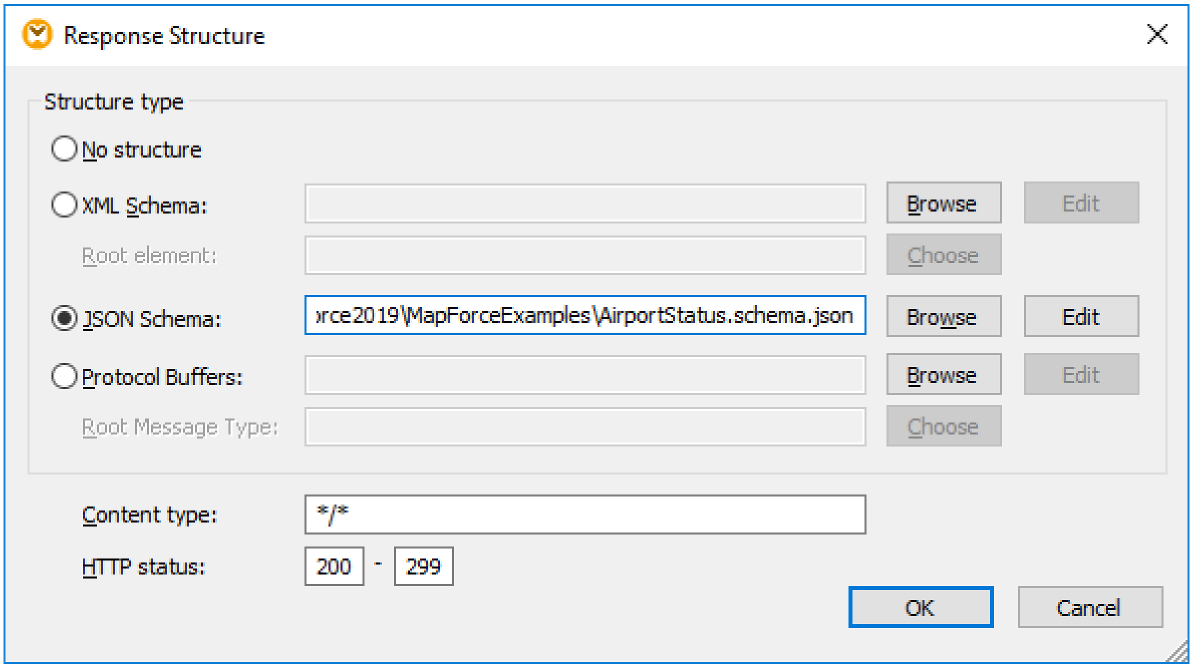
For the request part, we are going to call the Web service with a parameter which will supply the value "SFO" in the request. For the response part, we will map the response data to a JSON file. Therefore, the mapping will consist of three main components: the Web service call, the input parameter, and the JSON output file.

Step 1: Add the Web service component

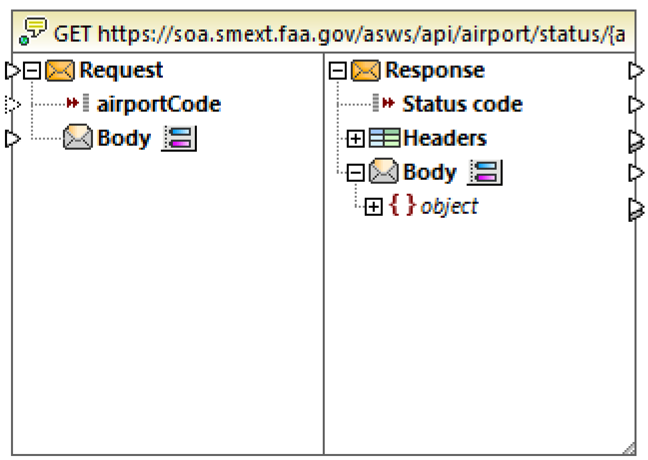
1. Make sure the transformation language of the mapping is BUILT-IN[27].
2. On the **Insert** menu, click **Web Service Function**.
3. Under **Service Definition**, click **Manual**.
4. Set the request method to GET and the URL to `https://soa.smext.faa.gov/asws/api/airport/status/{airportCode}`. The value within curly braces is a template parameter which will be replaced with the actual airport code at runtime (see Adding a Web Service Call (REST-Style[587]).
5. Click the **Add Parameter** ( ) button and add a new parameter to the "Parameters" table. Notice the name of the parameter must be the same as that of the parameter specified within curly braces in the URL. Set the style to "Template", type to "String", check the "Mappable" and the "Required" option. The "Description" field is optional.
6. Add a header to tell the Web server that the client (that is, MapForce) expects JSON in the response. To do this, click again the **Add Parameter** ( ) button, name the parameter "Accept", set the style to "Header" and enter **application/json** as fixed value.



7. Under **Response**, click the **Edit** button and browse for the schema of the Web service response. The schema can be found at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\AirportStatus.schema.json**.

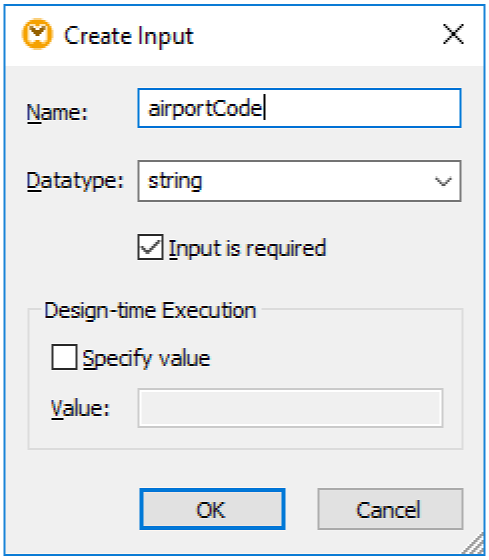


8. Click **OK**. The Web service is now available on the mapping area.

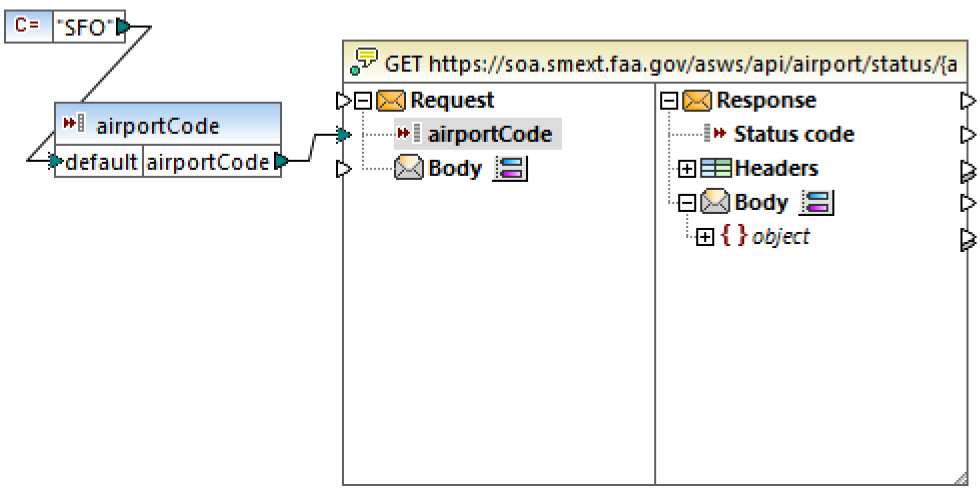


Step 2: Add the input parameter

1. On the **Insert** menu, click **Insert Input**.
2. Enter "airportCode" as parameter name and click to clear the **Input is required** option (since the input will be supplied by a constant, as shown in the next step).



3. Add a constant (using the menu command **Insert | Constant**) with value "SFO" and connect it to the parameter input.
4. Connect the output of the aiportCode parameter to the input of the request, as shown below.

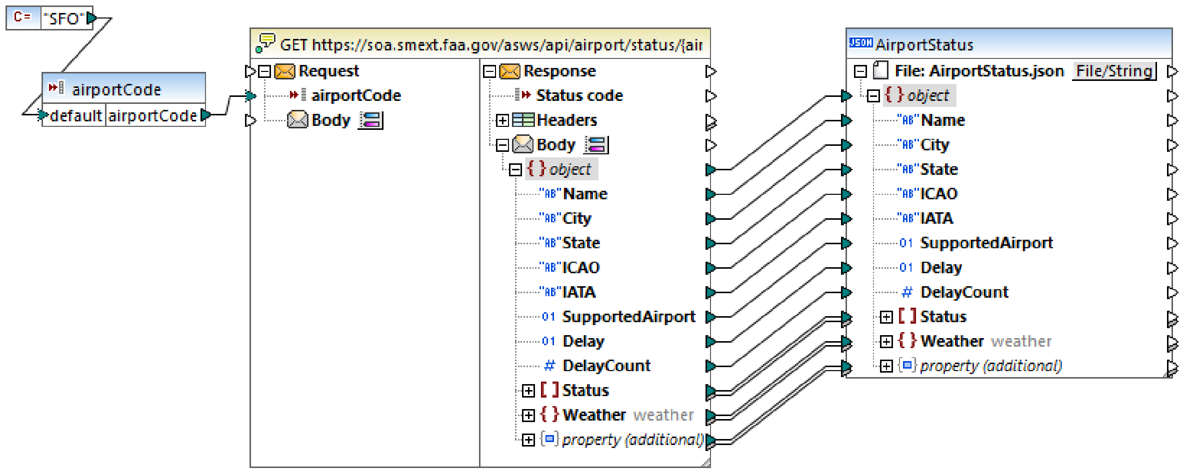


At this stage, the request part of the Web service is ready. At mapping runtime, the Web service will be called with the value provided by the constant (in this case, "SFO").

Step 3: Add the JSON output

1. On the **Insert** menu, click **JSON Schema/File**.
2. Browse for the **<Documents>\Altova\MapForce2023\MapForceExamples\AirportStatus.schema.json**, and click **Open**.
3. When prompted to supply a sample JSON file, click **Skip** (there is no need for a sample JSON file since it will be generated).
4. On the **Connection** menu, make sure that the **Auto Connect Matching Children** menu item is enabled. Enabling this option saves you time with the next step.

5. Connect the `object` node of the response body of the Web service to the `object` node of the JSON component, as shown below. Since the **Auto Connect Matching Children** was enabled in the previous step, all descendent nodes are connected automatically, so you don't need to draw individual connections for each.



Step 4: Execute the mapping

You are now ready to call the Web service. Click the **Output** button to execute the mapping and preview the generated output. If the Web service call is executed successfully, the **Output** tab displays the returned JSON structure. If the call is not successful, MapForce will return the error accordingly. As stated in the description of the Web service, the HTTP code 500 may be returned by the Web service in the case of error response. You may also get the HTTP code 404 if the requested information has not been found.

If you have MapForce Server, you can also compile the mapping to a mapping execution file (*.mfx) and execute it from the command-line or from the MapForce Server API on the server machine where MapForce Server runs (see Compiling Mappings to MapForce Server Execution Files1378).

You have now finished creating a generic HTTP Web service call that uses a GET method to retrieve airport status data in real time. For more information about working with generic Web services, see also Adding a Web Service Call (REST-Style)587.

4.5.7 Example: Mapping Data from an RSS Feed

This example shows you how to call a generic HTTP Web service from MapForce in order to map data from an RSS (Rich Site Summary) feed to Microsoft Excel. The example is accompanied by a mapping design file, which is available at the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\RssReader.mfd .

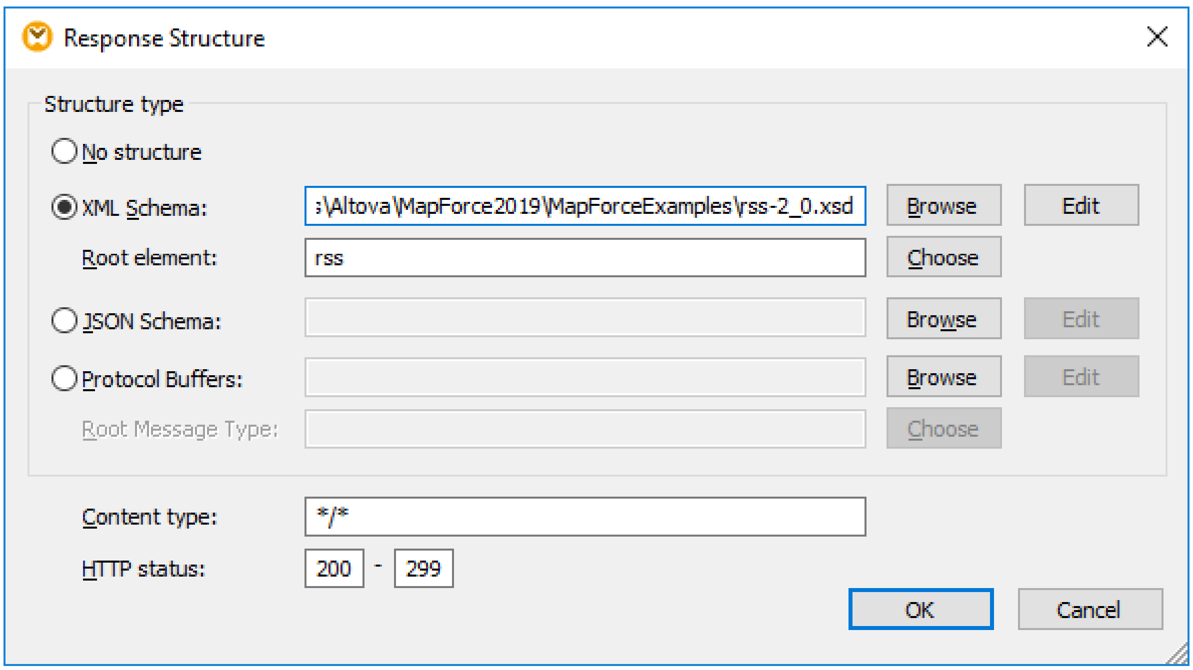
In this example, the response of the Web service is mapped from the RSS feed of the Altova blog (https://blog.altova.com/feed). The schema of the Web service response structure will be required, so that MapForce can create the structure of the data returned by the feed. For convenience, the required schema is available at the following path relative to your "(My) Documents" folder:**<Documents>\Altova\MapForce2023\MapForceExamples\rss-2-0.xsd**.

Calling the RSS feed does not require any request parameter, so the request part of the Web service call will be empty. As for the response, it will be mapped to a Microsoft Excel file. Thirdly, in order to make the publication date of each RSS entry easily readable, it will be formatted as YYYY-MM-DD in the Excel file. To achieve this goal, date processing functions will be used.

Therefore, the mapping will consist of three main parts: the Web service call, the target Excel component, and the date processing functions.

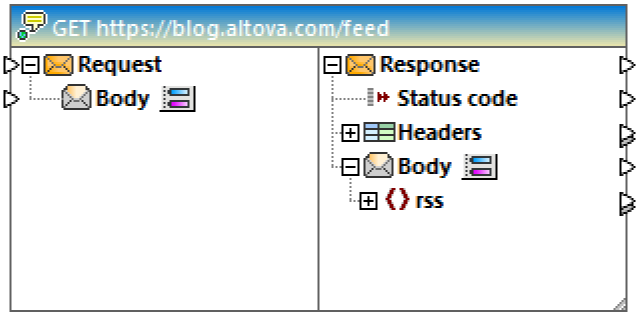
Step 1: Add the Web service component

1. Make sure the transformation language of the mapping is BUILT-IN[27].
2. On the **Insert** menu, click **Web Service Function**.
3. Under **Service Definition**, click **Manual**.
4. Set the request method to GET and the URL to `https://blog.altova.com/feed`.
5. Under **Response**, click the **Edit** button and browse for the schema of the Web service response. The schema can be found at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\rss-2-0.xsd**.



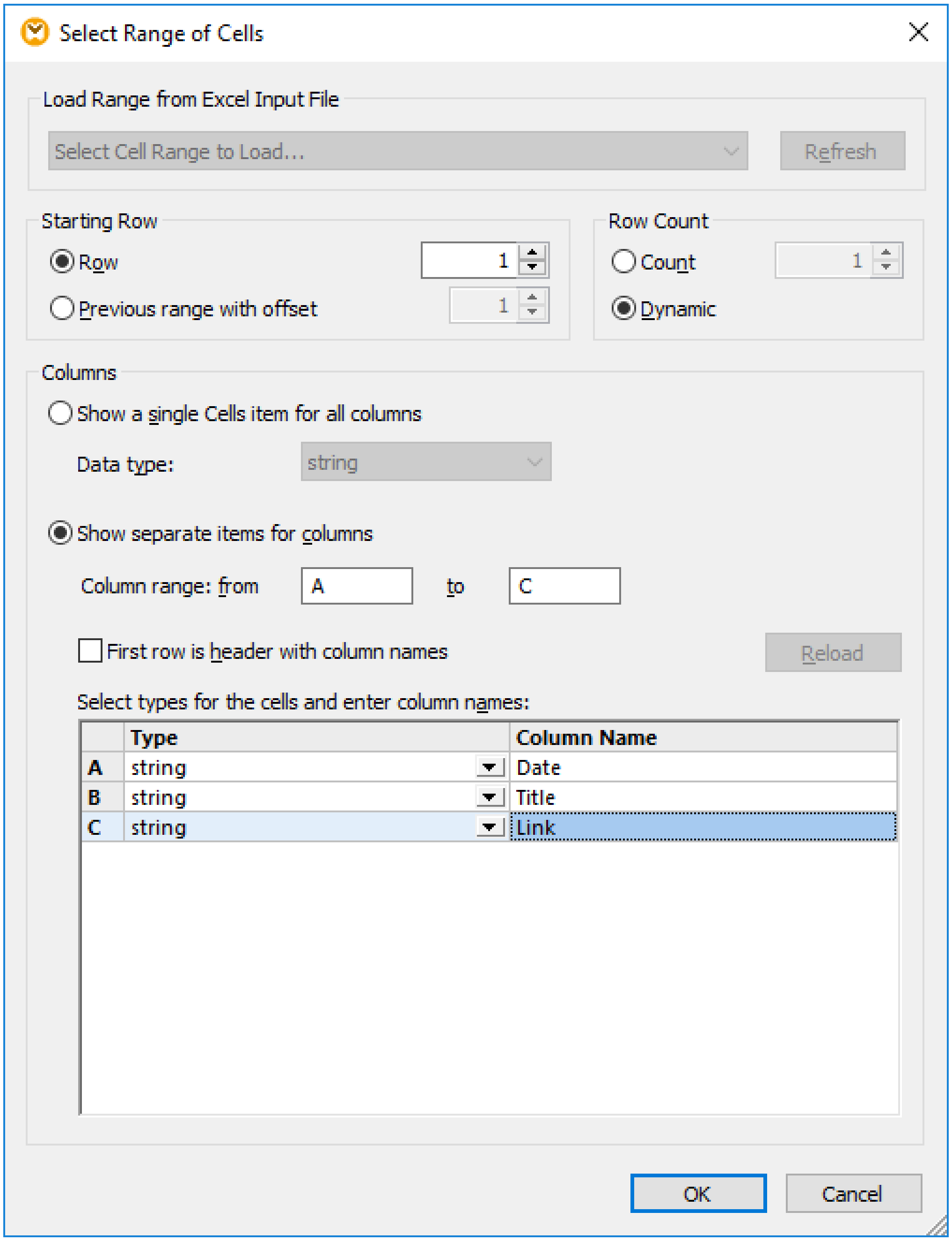
6. Click **OK**.

At this stage, the Web service is available on the mapping area and you are ready to map data from the response part.



Step 2: Add the target Excel component

1. On the **Insert** menu, click **Excel 2007+ File**.
2. When prompted to supply a sample file, click **Skip**. The Excel component is now available on the mapping area.
3. Click the button next to **Sheet 1**, and click to clear the **Show Worksheets by name** option.
4. Click the button next to **Row 1**, and change the options as shown below.



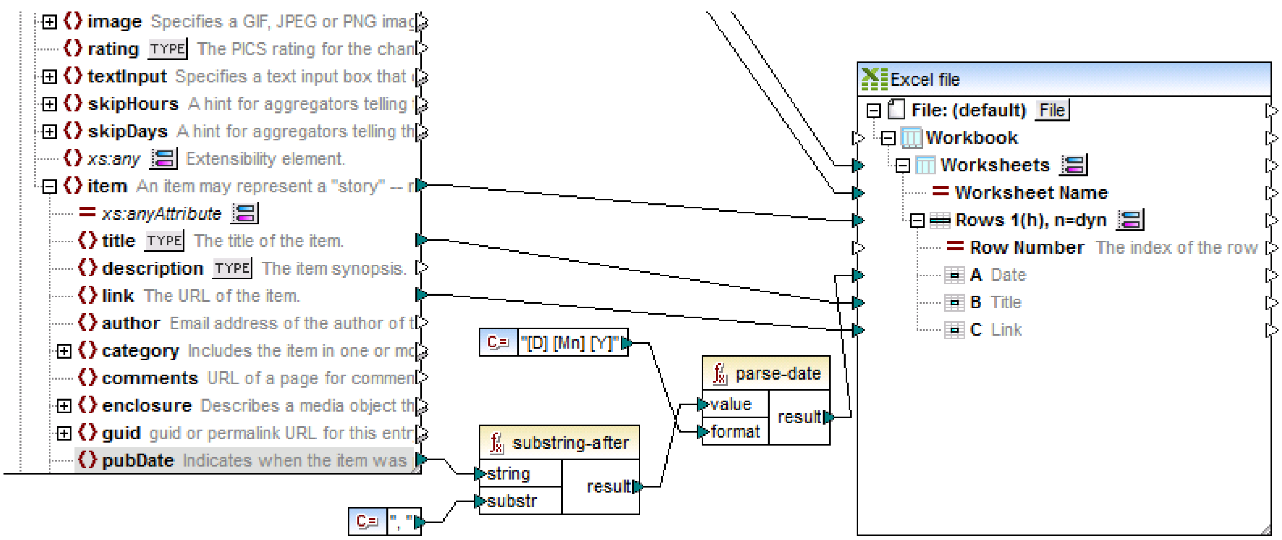
5. Connect the following items of the response body to the following items of the Excel component:

Response body	**Excel file**
`channel`	`Worksheets`
`title`	`Worksheet Name`
`item`	`Rows`
`item/title`	`Row B (Title)`
`item/link`	`Row C (Link)`

At this stage, the **Row A (Date)** is still missing an input value. This value will be added in the next step.

Step 3: Add the date processing functions

The date in the response of the RSS feed appears formatted in RFC-822 format, for example: `Wed, 20 Jan 2016 14:49:35 +0000`. The goal is, however, to see the date formatted in the Excel file as `YYYY-MM-DD`. To achieve this, add the **`substring-after`**[1062] and **`parse-date`**[998] functions of the MapForce core library to the mapping, see also Add a Function to the Mapping[905].



As shown above, the **`substring-after`**[1062] function takes the value from the **pubDate** node and returns only the text after `, `. The resulting value is then passed to another MapForce core function, **`parse-date`**[998], which parses it using the mask **`[D] [Mn] [Y]`**, and returns the value as `xs:date`. The format mask means "numeric day of the month, followed by a space, followed by the month name in title case, followed by a space, followed by the four-digit year". For date and time formatting examples, see the reference to the `parse-dateTime`[999] function.

Finally, the parsed value is supplied to the **Date** row of the Excel component.

Step 4: Executing the mapping

You are now ready to call the Web service. Click the **Output** button to execute the mapping and preview the generated output. If the Web service call is executed successfully, the **Output** tab displays the returned data structure. If the call was not successful, MapForce returns the error accordingly.

If you have MapForce Server (https://www.altova.com/mapforce-server), you can also compile the mapping to a mapping execution file (*.mfx) and execute it from the command-line or from the MapForce Server API on the server machine where MapForce Server runs (see Compiling Mappings to MapForce Server Execution Files 1378).

You have now finished creating a generic HTTP Web service call that uses a GET method to read data from an RSS feed. For general information about working with generic Web services, see also Adding a Web Service Call (REST-Style) 587.

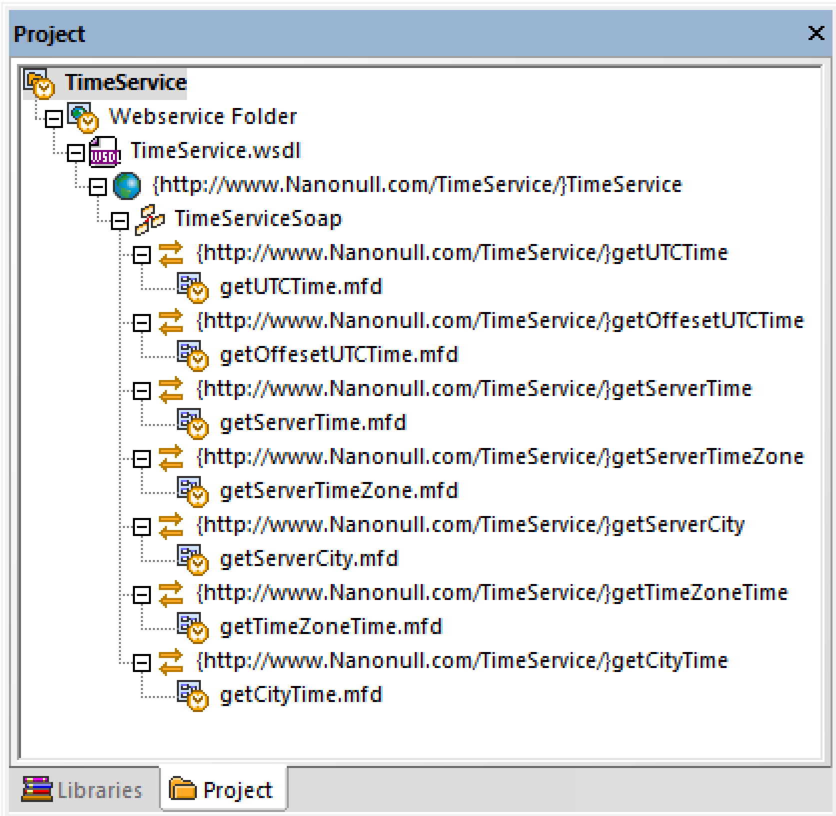
4.5.8 Example: Calling a SOAP Web Service

This example shows you how to query a Web time service using a constant as an input. The Web service itself was implemented using MapForce. This is for demonstration purposes—you can implement the Web service with any other technology that supports a compatible protocol.

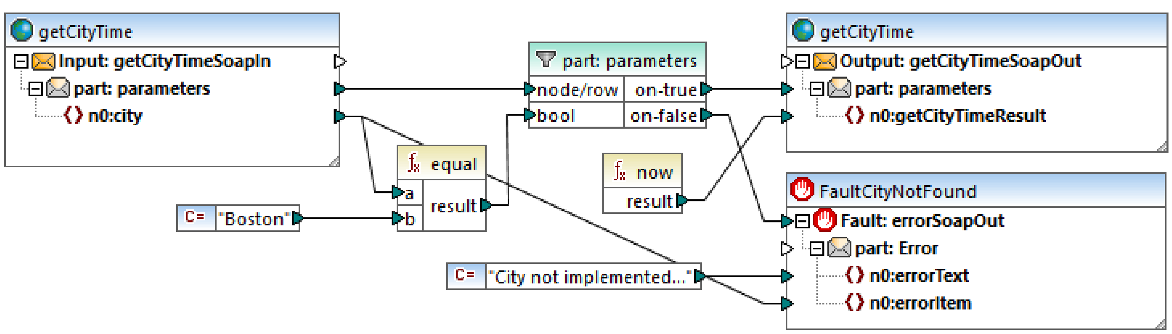
The mapping shown below is part of the **TimeService.mfp** mapping project, available in the **<Documents>\Altova\MapForce2023\MapForceExamples\TimeService** folder. The **TimeService2.mfp** project file available in the **<Documents>\Altova\MapForce2023\MapForceExamples\TimeserviceWsdl2** folder supports WSDL 2.0.

To view how the Web service is implemented:

1. Select **File | Open** and select the **TimeService.mfp** file in the **<Documents>\Altova\MapForce2023\MapForceExamples\Timeservice** folder. The files associated with the project are loaded in the Project window.



2. Double-click the **getCityTime.mfd** entry in the project window.



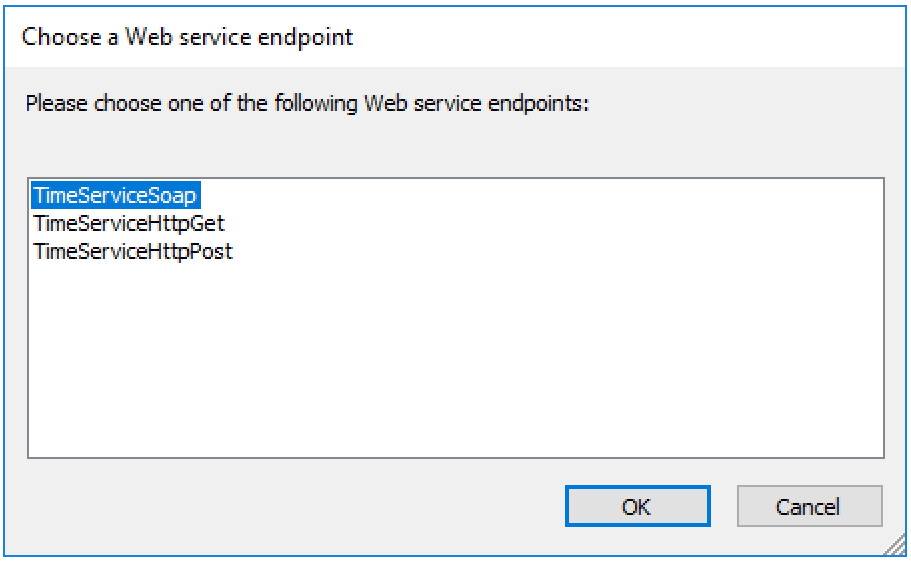
The **getCityTime.mfd** mapping accepts a city name as input and returns the current time in the output. If the city is not "Boston", a WSDL fault is returned. The mapping takes the input data from the **getCityTimeRequest.xml** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples\TimeService** directory. If you double-click the **getCityTimeSoapIn** component, you can see that this file is set as data source.

For the scope of this example, we will now assume that this particular Web service project has already been compiled and deployed to an actual Web server which you are going to call in the following steps of this example. A demo online service is available for that purpose at http://nanonull.com/TimeService/TimeService.asmx. For examples of how to create such Web services with

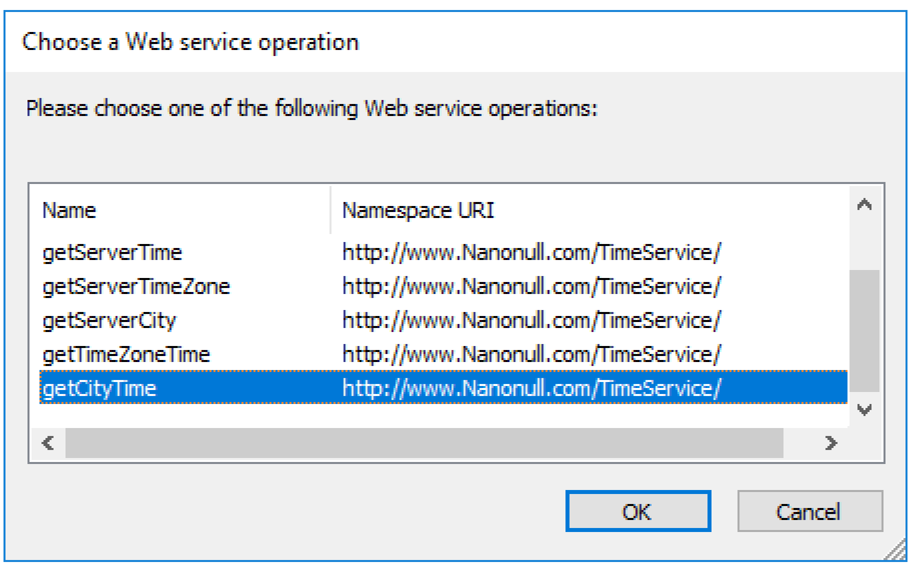
MapForce, see Example: Generating SOAP Web Services (C#)[1223] and Example: Generating SOAP Web Services (Java)[1237].

To add the Web service to the mapping:

1. Select **File | New**, click the Mapping icon and confirm with OK.
2. Select the menu option **Insert | Web service function...** or click the toolbar button.
3. Click Browse to select the WSDL definition file; select **TimeService.wsdl** from the TimeService directory, then click the Open button.
4. When prompted to choose a Web service endpoint, click **TimeServiceSoap**.

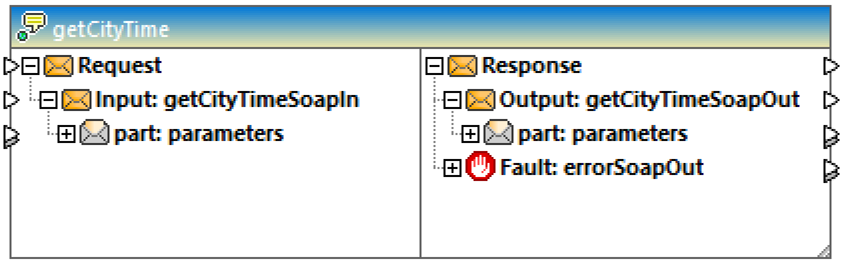


5. When prompted to choose a Web service operation, click **getCityTime**.



6. Leave the WSDL Call Settings unchanged, and click OK. For more information, see Web Service Call Settings[606].

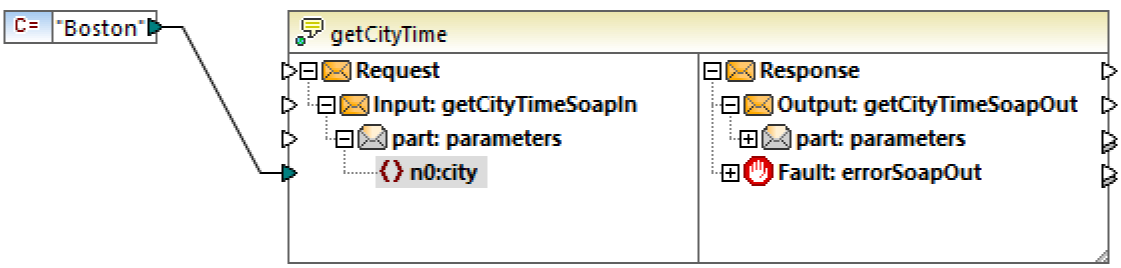
The **getCityTime** Web service function is inserted as a single component. Note that it actually represents all eight components that make up the **getCityTime.mfd** file as saved in the WSDL project.



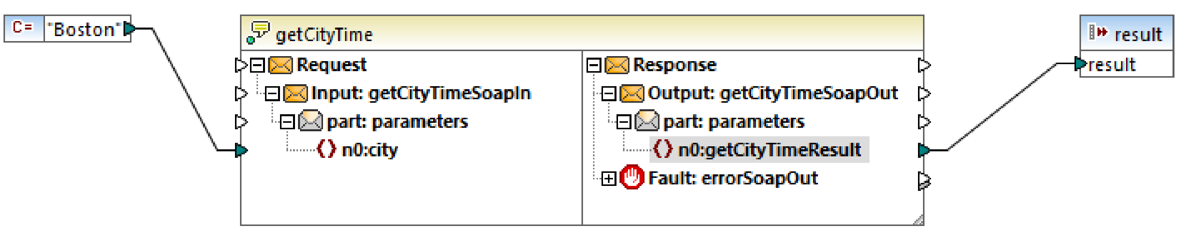
The left section of the component defines the data input (SoapIn), while the right side defines the data output (SoapOut), which may also include a fault section, if one has been defined in the .wsdl file.

To call the Web service:

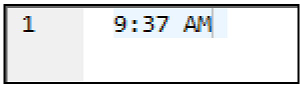
1. Insert the component that is to supply the input data, e.g. a constant, text, or schema component. For the scope of this example, insert a constant component, and enter "Boston" as the input string.



2. Connect the constant to the **n0:city** item.
3. Insert a simple output component (on the **Function** menu, click **Insert Output**).
4. Connect the **n0:getCityTimeResult** to the output component.

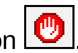


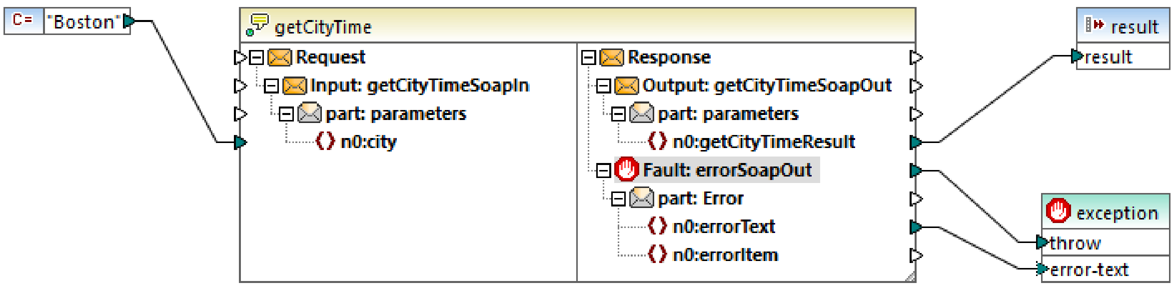
5. Right-click the "getCityTime" component on the mapping, and select **Properties**. The Web Service Call Settings dialog box opens. Enter `http://nanonull.com/TimeService/TimeService.asmx` in the **URL** box (this is a demo online Web service created with MapForce, see Implementing SOAP Web Services [1204]).
6. Click the **Output** tab. As illustrated below, the current time in Boston is displayed in the **Output** pane.



Note: The input value of the Web service function takes precedence over the data source of the original mapping. For example, the constant "Boston" takes precedence over the **getCityTimeRequest.xml** data source file in the original mapping.

To map Web service faults:

1. Select **Insert | Exception**, or click the Exception toolbar button .
2. Map the **Fault:** item to the **throw** item of the exception component.
3. Map the **n0:errorText** item to the **error-text** item of the exception component.



4.5.9 Example: OAuth 2.0 Authorization

This example shows you how to call a REST-style Web service that requires OAuth 2.0 Authorization. The client application is a MapForce mapping that will retrieve calendar events using the Google Calendar API (https://developers.google.com/calendar/). To keep things simple, the mapping will retrieve the calendar information "as is" and will just output the raw JSON result without any other processing.

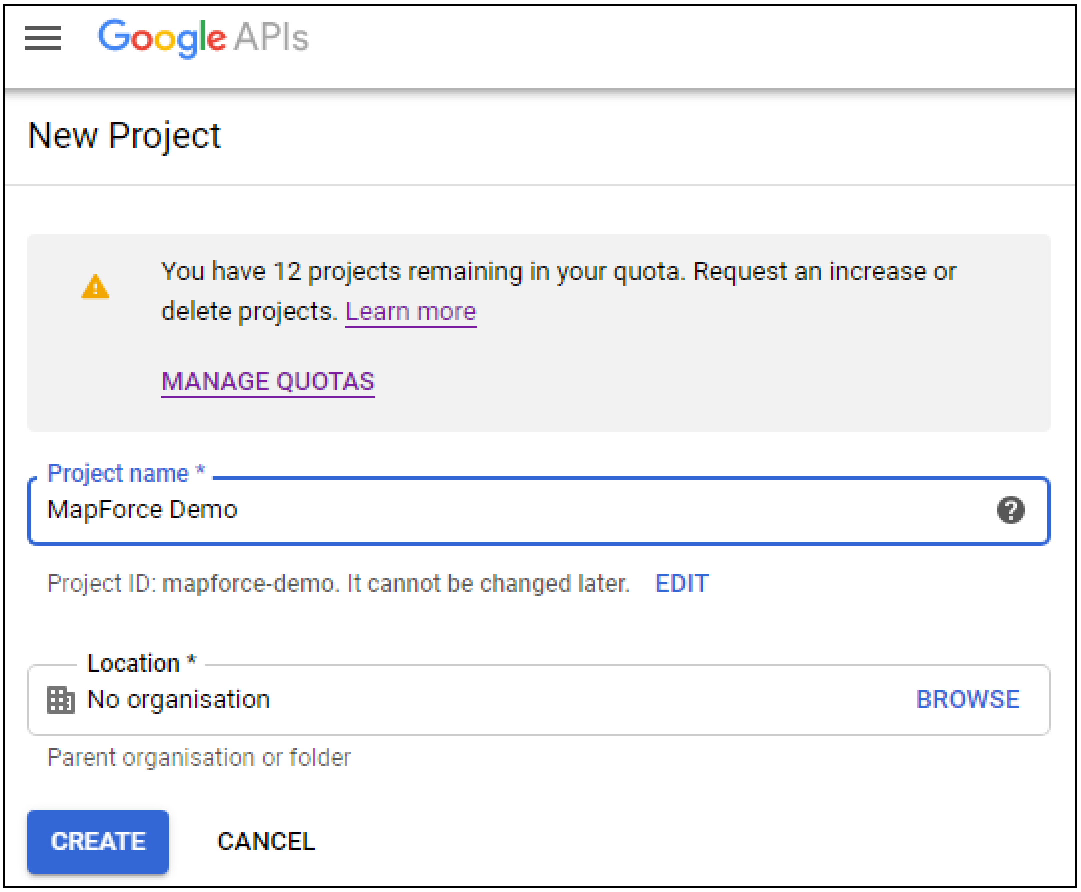
Prerequisites:

- MapForce Enterprise Edition
- To follow this example step-by-step, you must have a Google account. If you would like to call another Web service, obtain OAuth 2.0 credentials from your Web service provider and use them in the instructions below instead.

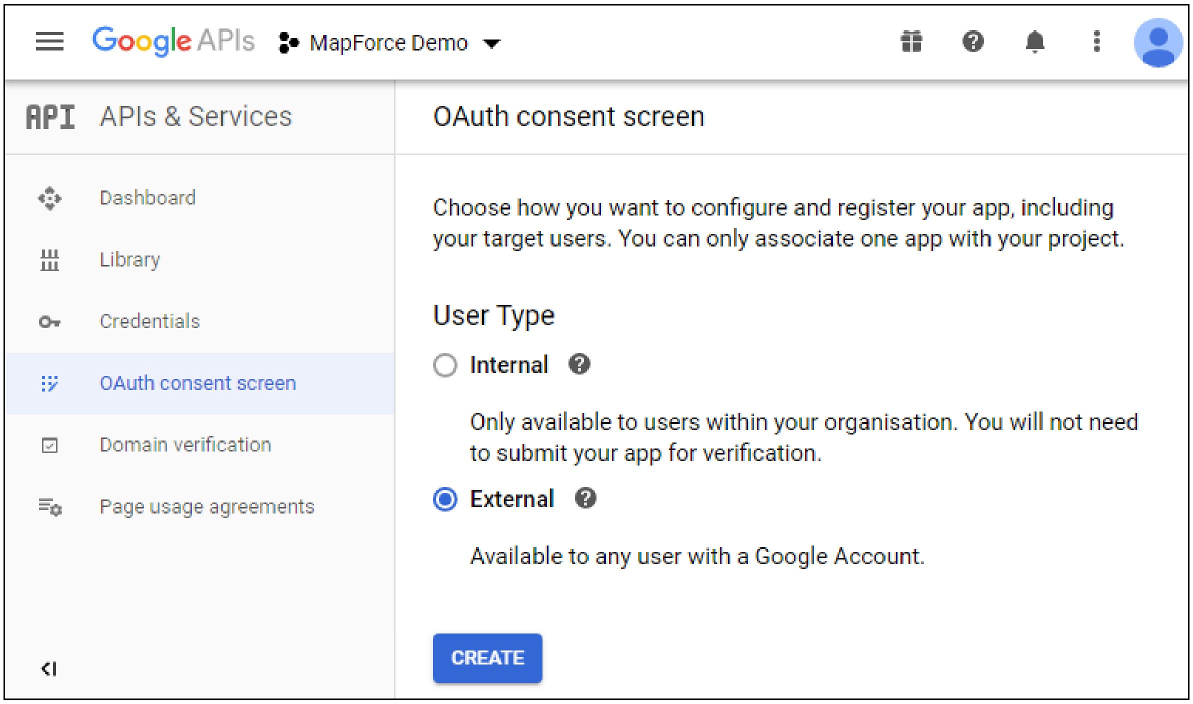
Obtain the OAuth 2.0 credentials

If you already have the OAuth 2.0 credentials required to access the Web service, you can skip this step. Otherwise, the exact instructions to obtain them depend on the provider of the Web service that your mapping will call. To call the Google Calendar API like in this example, follow these steps:

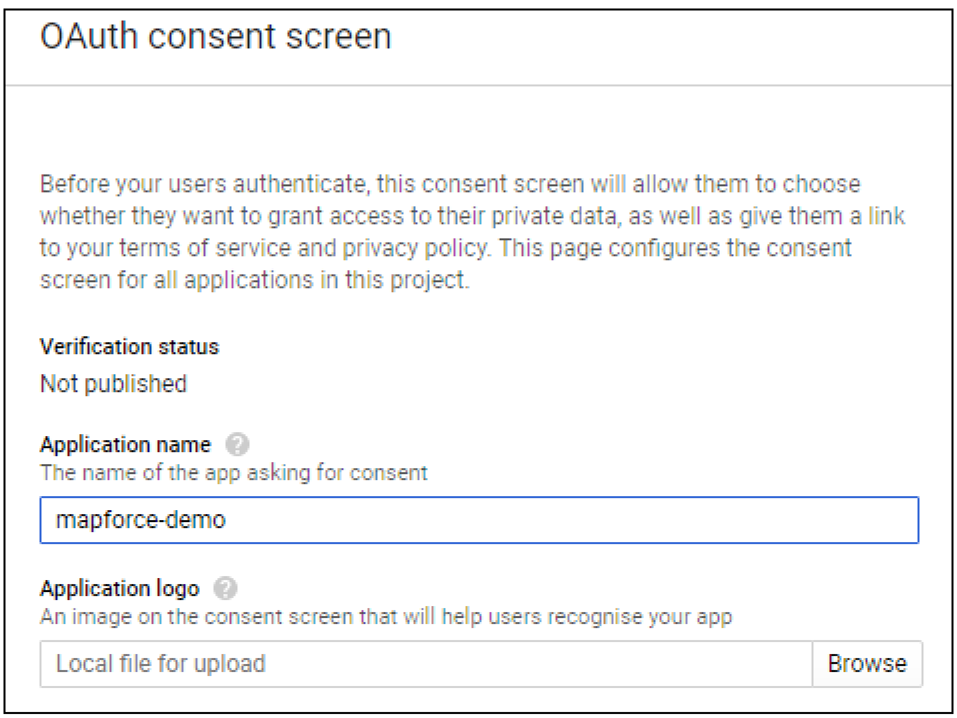
1. Login to the Google API Console (https://console.developers.google.com/).
2. Create a new project.



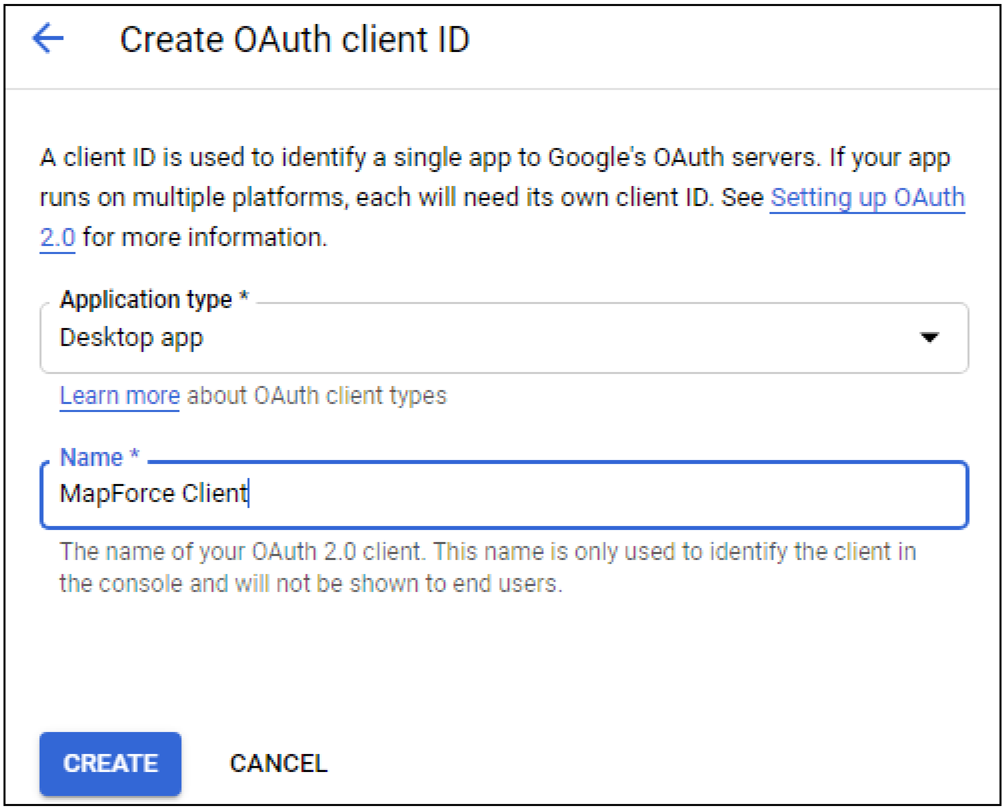
3. Click **OAuth consent screen**.
4. Select **External** as user type, unless you have a G Suite account which would enable you to grant API access only to users in your organization.



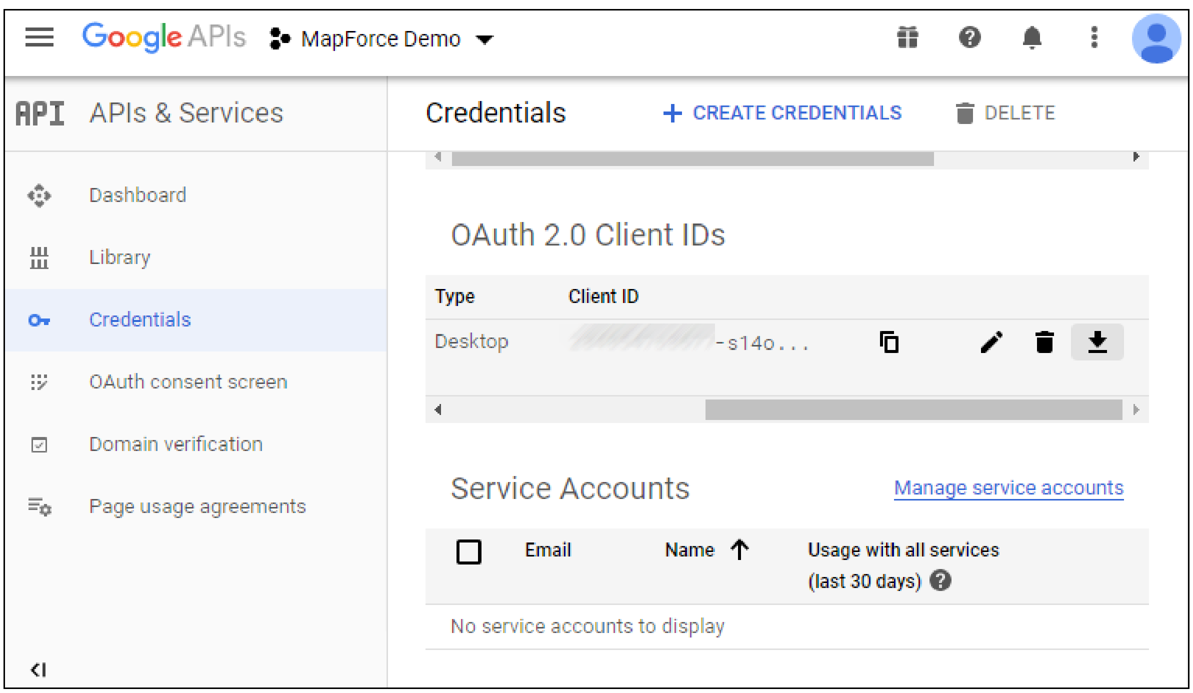
5. Enter "mapforce-demo" as application name and save the settings.



6. Click **Create credentials** and then select **OAuth Client ID**.
7. Enter **Desktop app** as application type and "MapForce Client" as the client name.



8. Click **Create**. The client ID is created and becomes available in the **Credentials** page.



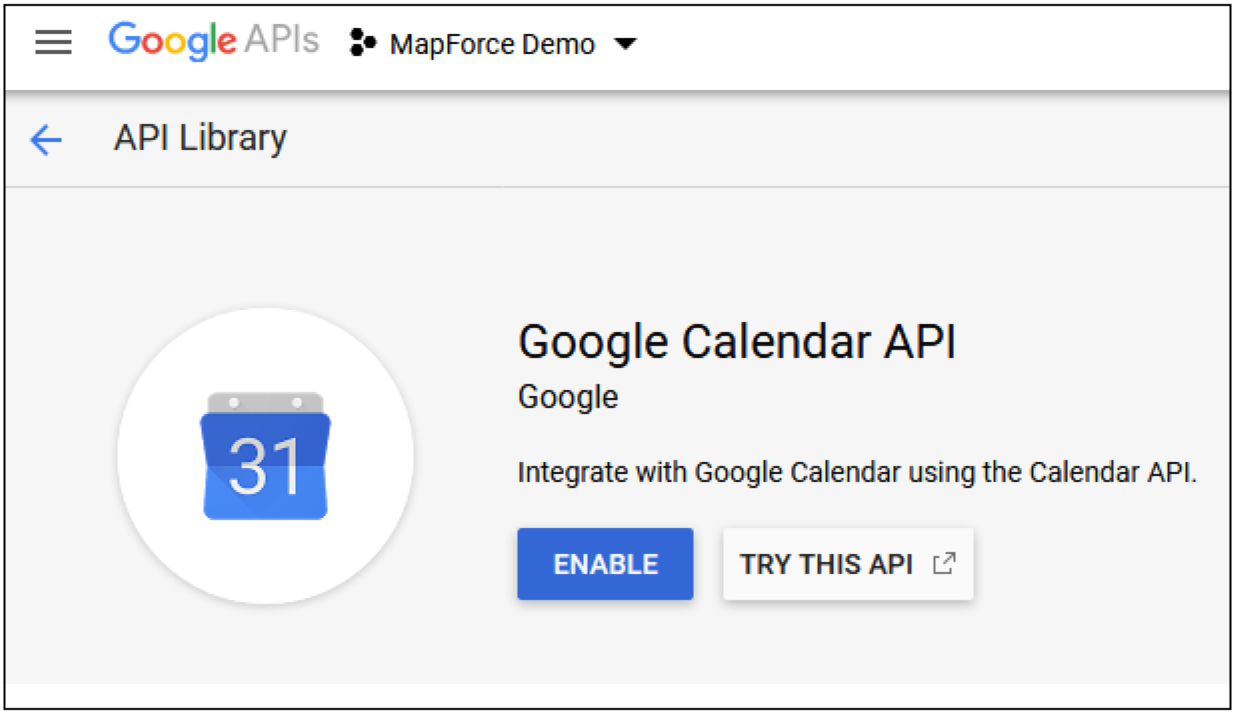
9. Click  to download the OAuth 2.0 authorization details as a JSON file.

You have now obtained the OAuth 2.0 authorization details from Google Console API, namely:

1. Authorization Endpoint
2. Token Endpoint
3. Client ID
4. Client Secret

Enable the Google Calendar API

To accept calls from clients, the Google Calendar API used in this example must be enabled. In the Google API Console, click **Library**, search for the Google Calendar API and enable it:



In this example, we are going to call the **list** method of the **Events** entity. You can find detailed reference to this API method at https://developers.google.com/calendar/v3/reference/events/list. For now, note the following important points:

1. As pointed out in documentation, the method must be called by sending a GET request to `https://www.googleapis.com/calendar/v3/calendars/calendarId/events`, where `calendarId` is the identifier of a Google Calendar. The **calendarId** request parameter will be configured from MapForce in a subsequent step.
2. Calling the API method requires at least one of the following scopes:

   - `https://www.googleapis.com/auth/calendar.readonly`
   - `https://www.googleapis.com/auth/calendar`
   - `https://www.googleapis.com/auth/calendar.events.readonly`
   - `https://www.googleapis.com/auth/calendar.events`

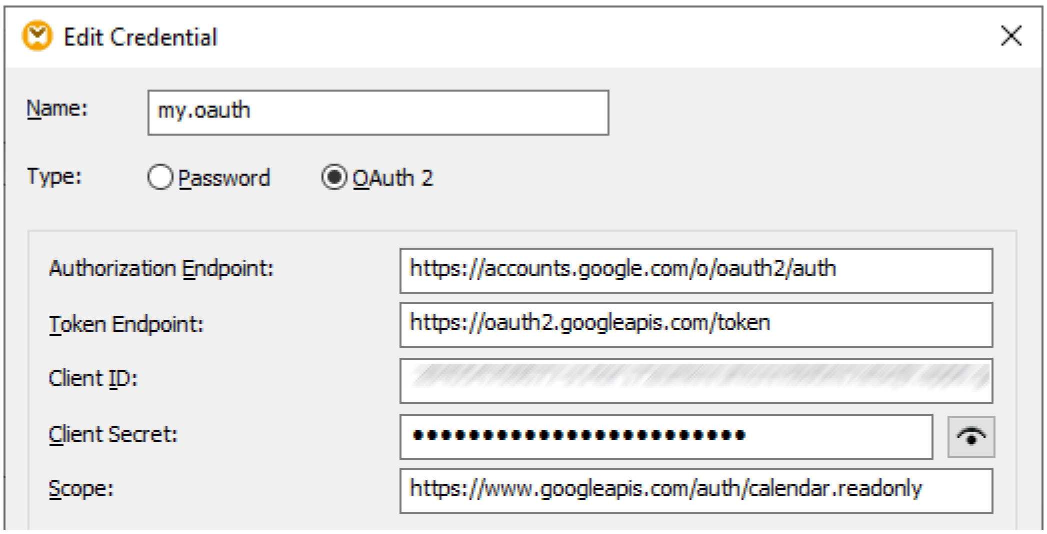
   During the OAuth 2 authorization process, your mapping will have to provide one of the scopes above—

this will also be configured in a subsequent step. For the purpose of this example, the first "read-only" scope is sufficient.

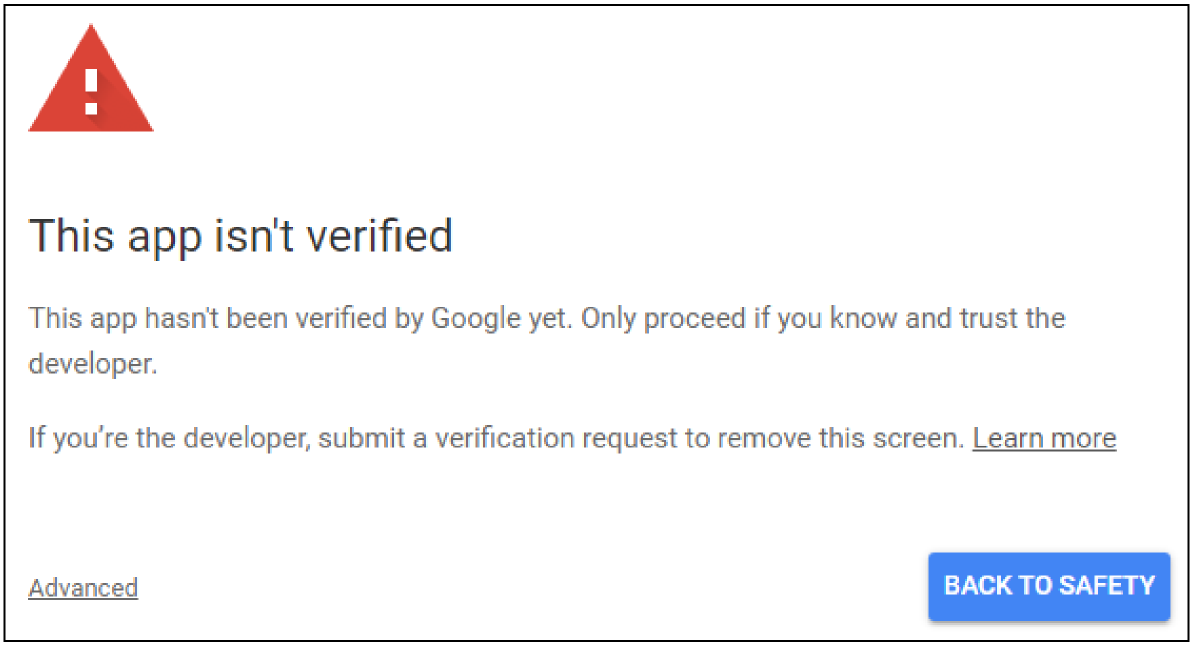
Request an authorization token

In order to preview the mapping in MapForce, you will need to add the OAuth 2.0 authorization details to the mapping and request an authorization token, as illustrated below.

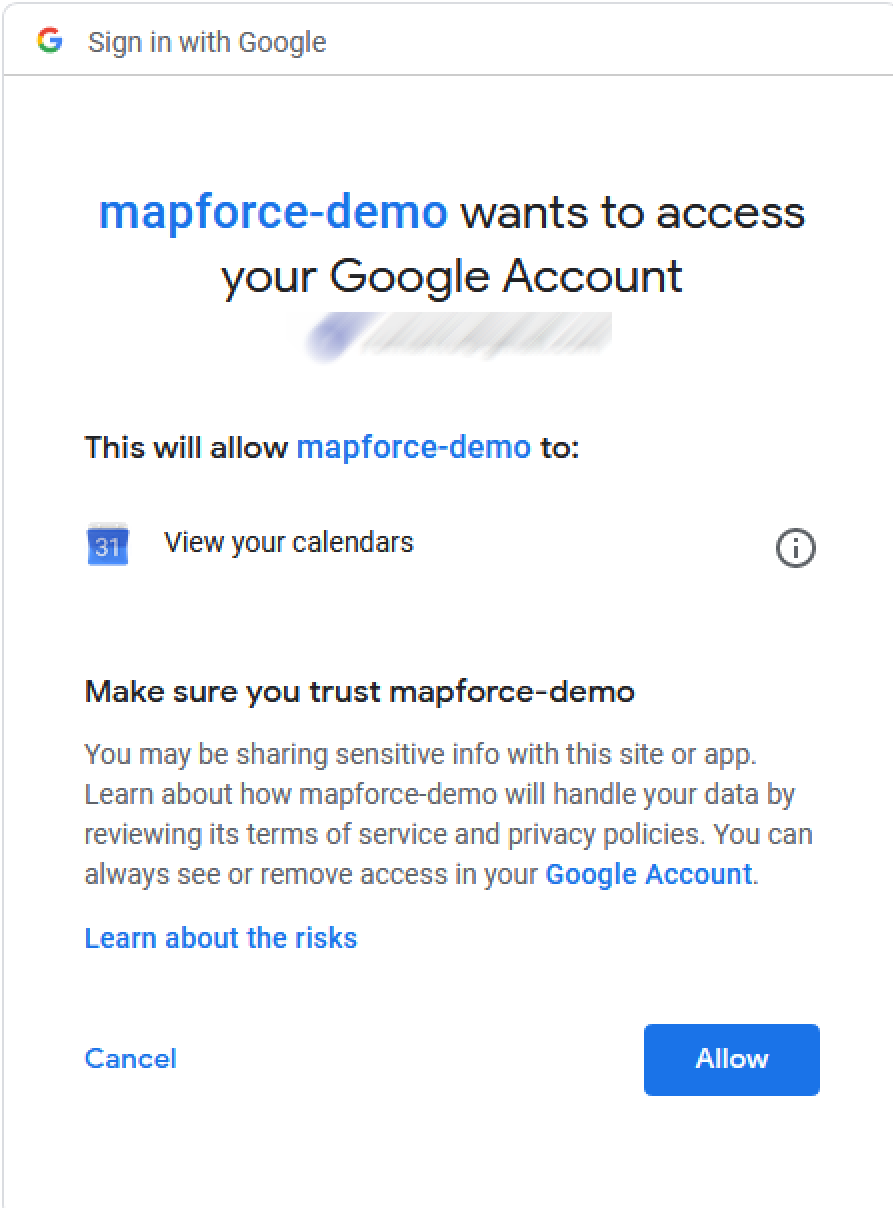
1. In MapForce, right-click an empty area on the mapping, and select **Open Credentials Manager** from the context menu.
2. Click **Add Credential**.
3. Enter a name ("my.oauth", in this example), and select **OAuth 2** as type.
4. Fill in the **Authorization Endpoint**, **Token Endpoint**, **Client ID**, **Client Secret** text boxes with the corresponding values from the JSON file downloaded previously.
5. Enter `https://www.googleapis.com/auth/calendar.readonly` in the **Scope** text box.
6. Leave all other settings as is.



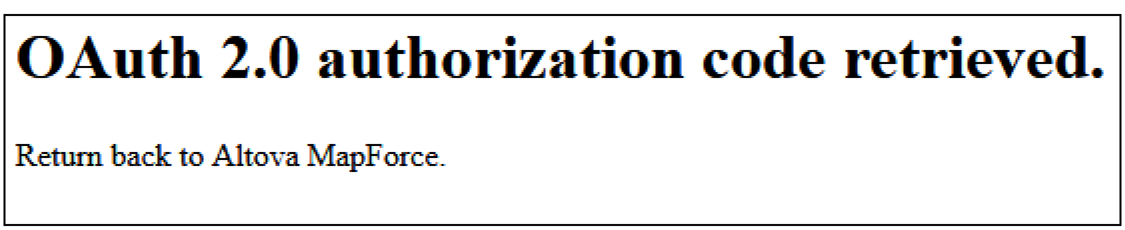
7. Click **Request Access Token** to obtain the token from the authorization server (in this example, Google). A browser window opens asking you to connect to your Google account.
8. Login to your Google account. Since you haven't submitted any app verification requests to Google yet, the following page appears.



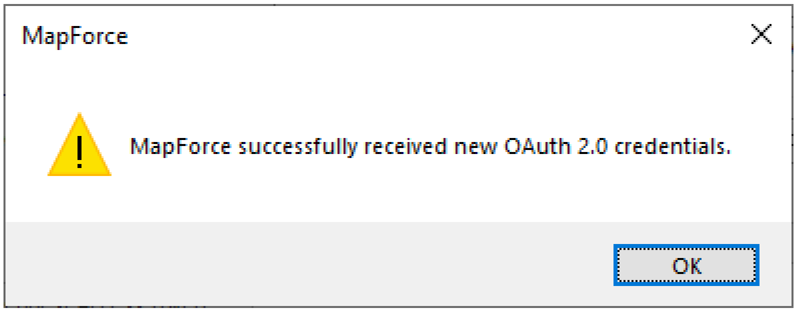
9. Click **Advanced**, and then click **Go to mapforce-demo (unsafe)**.



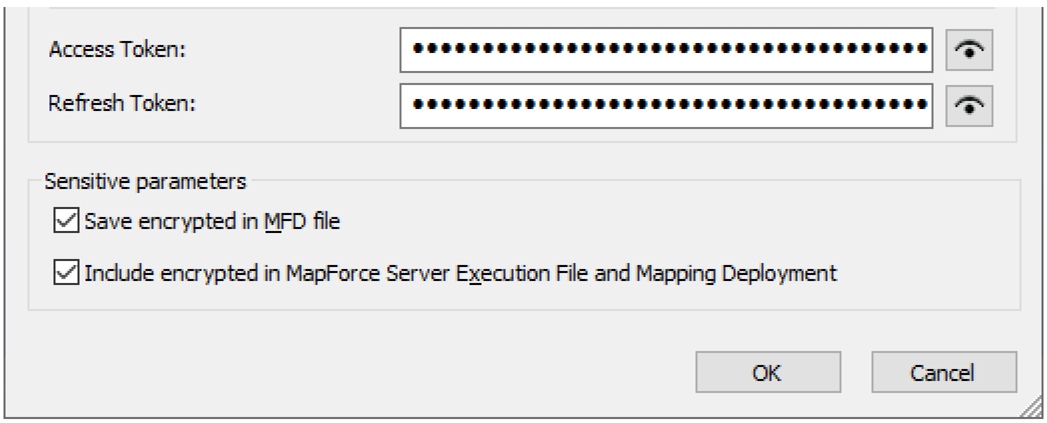
10. Click **Allow**. A confirmation is now displayed in the browser.



MapForce also notifies you that the OAuth 2.0 authorization code has been retrieved successfully.



11. Click **OK**. Notice that the **Access Token** and **Refresh Token** fields have now been populated with data.



12. Save the mapping as **GetCalendarEvents.mfd**.

> In this tutorial, the **Save encrypted in MFD file** check box is selected on the Edit Credentials dialog box. Therefore, the sensitive fields **Client Secret** , **Authorization Token**, and **Refresh Token** will be saved in encrypted form in the mapping design file (.mfd) when you save the mapping.

Be aware that the authorization token will eventually expire after a period. When that happens, you will no longer be able to run the mapping (at this stage, no mapping has been designed, but this will happen in a subsequent step). Whenever you need to obtain a new authorization code manually, click **Request Access Token**, and follows the steps described above.

Design the Web service call

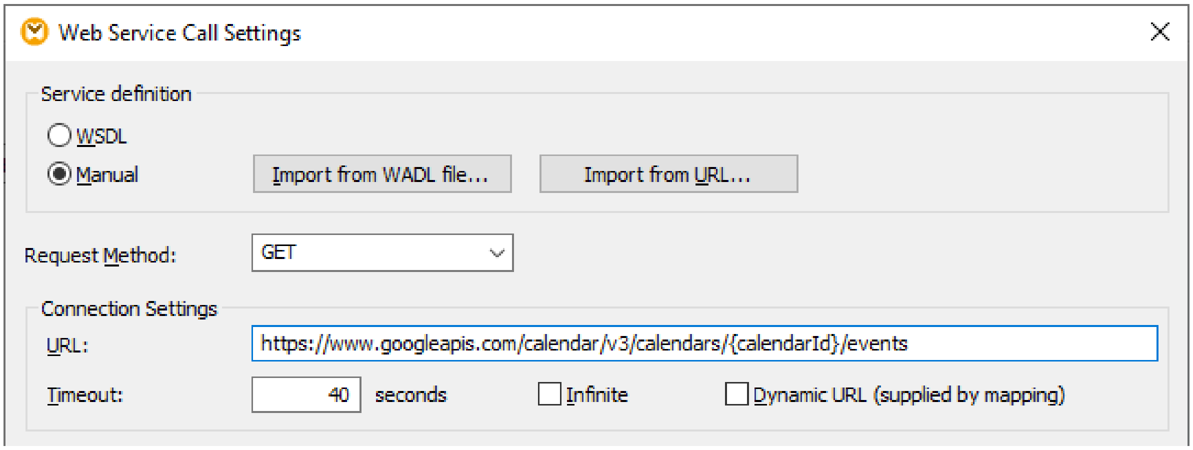
The mapping **GetCalendarEvents.mfd** created so far does not do anything yet. The only thing it contains are OAuth 2.0 credentials that enable access to the Google Calendar API.

Let's now design the Web service call in MapForce, as follows:

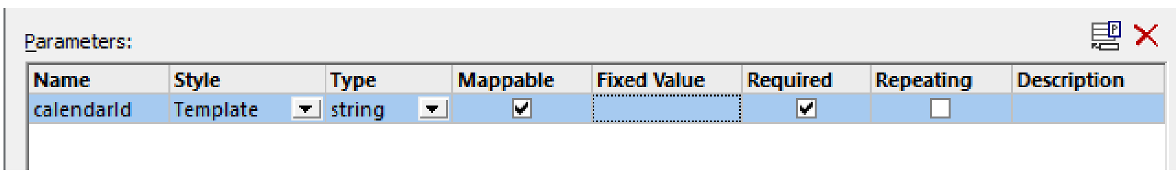
1. Open the **GetCalendarEvents.mfd** mapping.
2. On the **Insert** menu, click **Web Service Function**. The "Web Service Call Settings" dialog box appears.
3. Click **Manual**.
4. Select **GET** as request method and enter the URL to the Web service mentioned in a previous step:

`https://www.googleapis.com/calendar/v3/calendars/calendarId/events`.

5. Because **calendarId** is a placeholder that must be provided as a parameter, enclose it within curly braces as shown below.

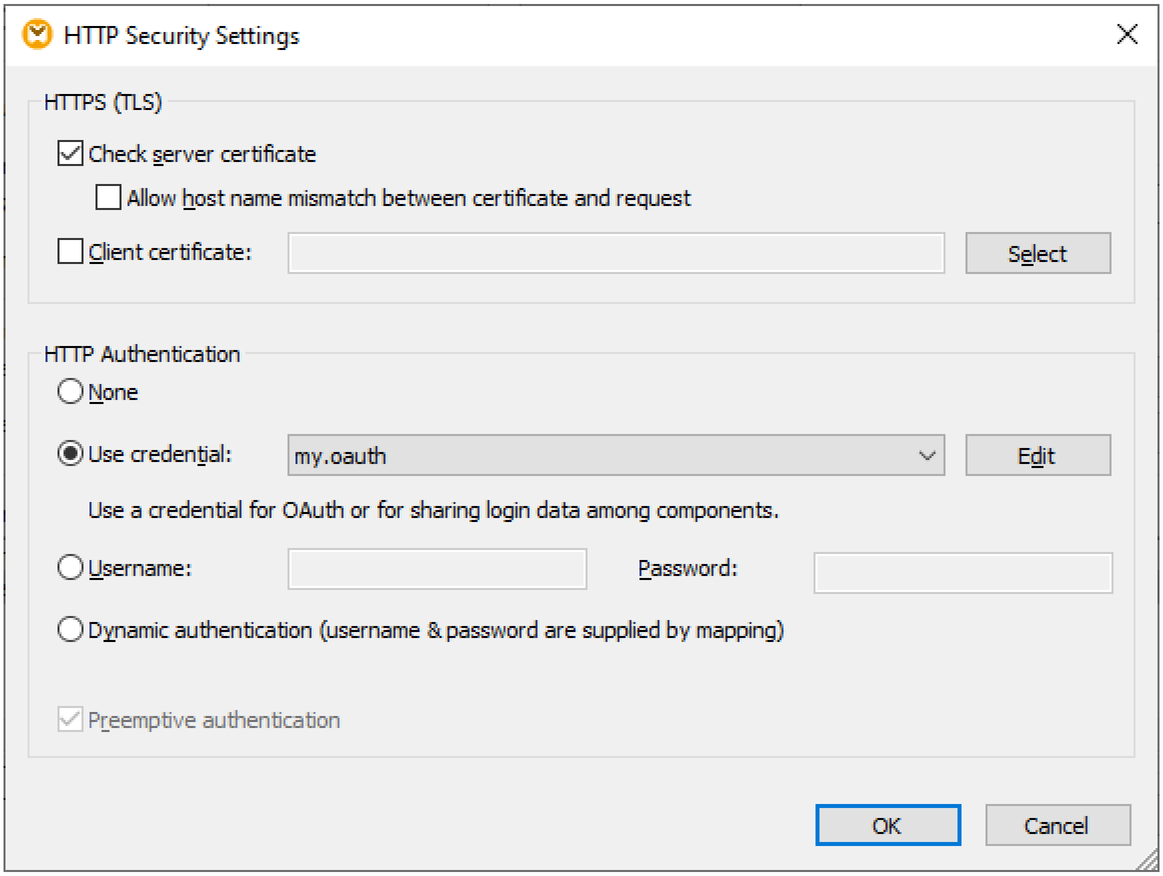


6. Click the **Add Parameter** button and define the parameter details as follows:

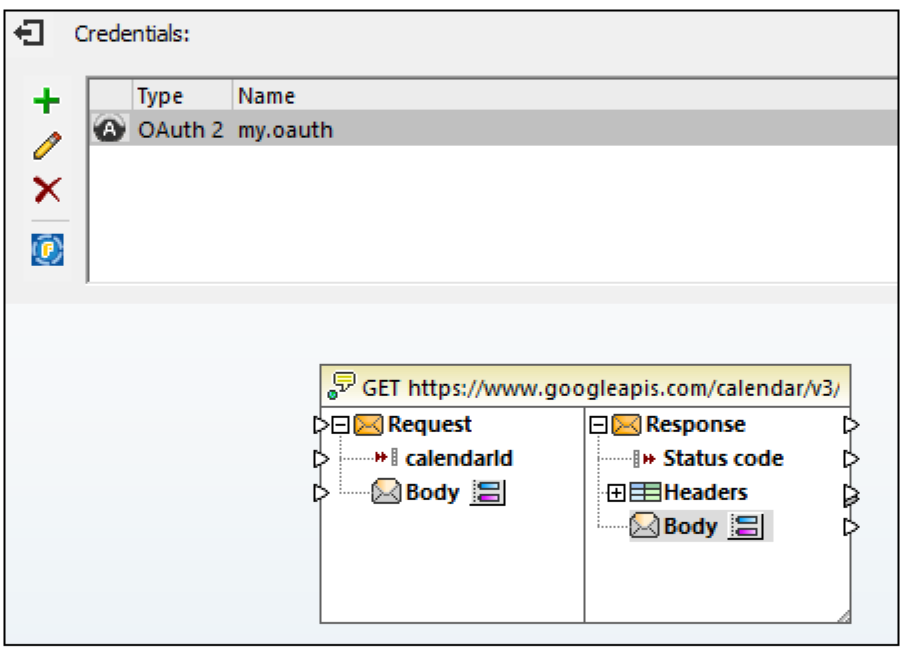


In the configuration above, the "Template" style makes it possible to replace the URL part enclosed within curly braces with the parameter value at runtime. "Mappable" means that you can supply the value from the mapping (for example, from a constant, or perhaps from an input parameter). Finally, the parameter has been marked as "Required" because the API call cannot take place without it. For further information, see Defining Web Service Parameters 592.

7. Click the **Edit** button adjacent to **HTTP Security Settings**.
8. On the "HTTP Security Settings" dialog box, select **Use Credential** and choose the "my.oauth" credential record configured previously.

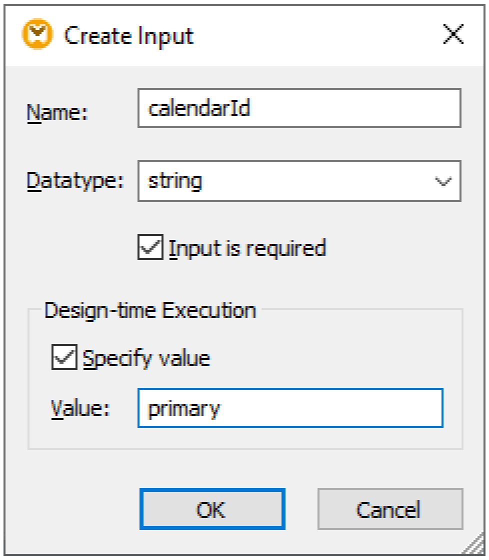


The Web service configured so far has the following appearance on the mapping:



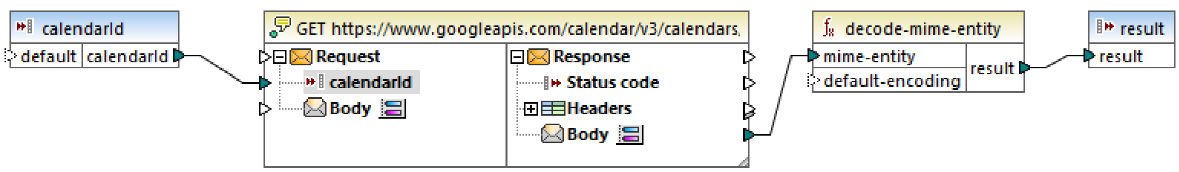
You can now complete the design by taking the following steps:

1. On the **Insert** menu, click **Insert Input**, and configure the component as follows:



As illustrated above, the input component has the design-time value "primary". According to the API's documentation, the value "primary" instructs the API server to access the primary Google calendar of the currently logged in user. Note that this value is a design-time value and is applicable only when you preview the mapping in MapForce. When the mapping runs in a server environment, you will need to provide the desired value at runtime.

2. Drag the decode-mime-entity [1133] function from the Libraries window into the mapping area. This function converts the raw MIME body received from the server into a string.
3. On the **Insert** menu, click **Insert Output**, and add a simple output component whose role is to output the result as a plain string.
4. Make the connections between components as illustrated below.



This concludes the design part in MapForce.

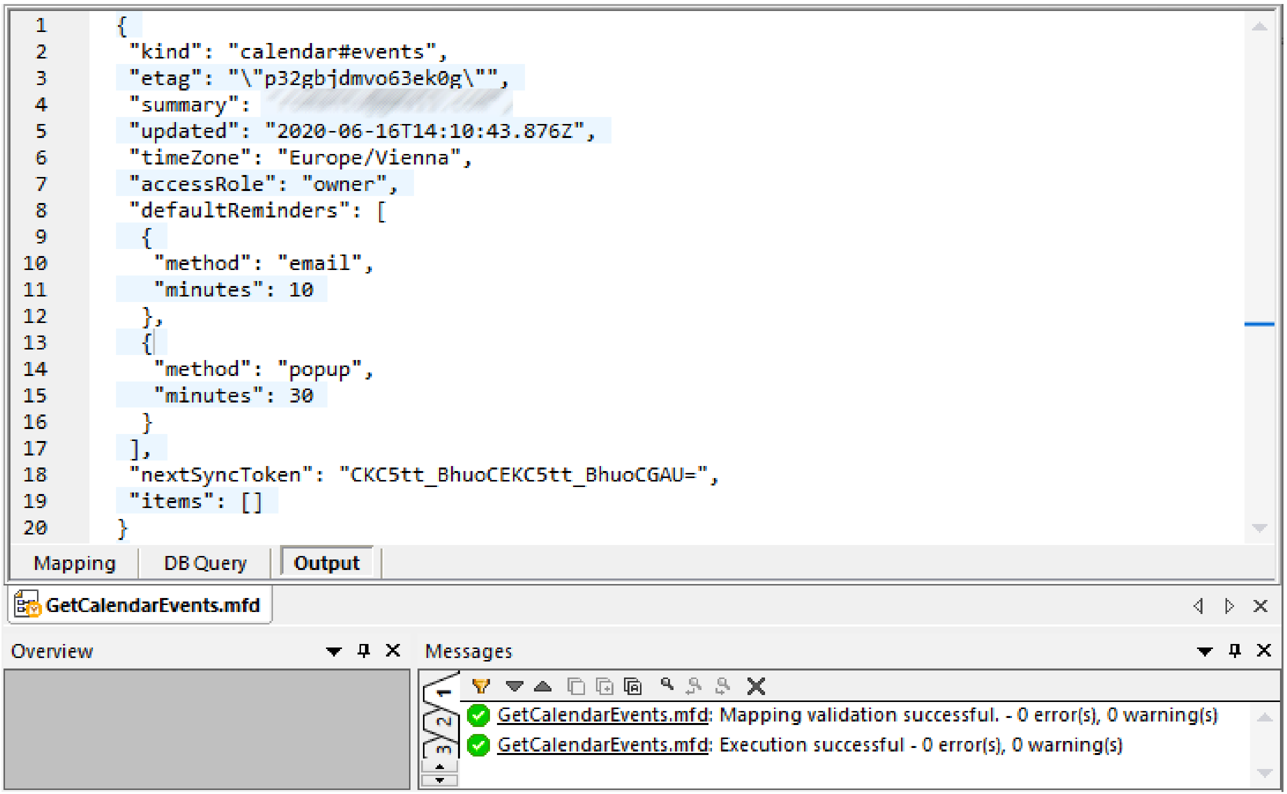
Test the mapping execution

To test the mapping execution in MapForce, click the **Output** tab and observe the result displayed in the Messages window.

If you get an authorization error such as "Unauthorized (401)", note the following troubleshooting tips:

1. Make sure that the Google Calendar API is enabled, see Enabling the Google Calendar API [632].
2. Request a new authorization token [633], in the event that the access token obtained previously has already expired.
3. Double-check that all OAuth 2.0 details were entered correctly in MapForce.

On successful execution and OAuth 2.0 authorization from MapForce, the mapping output is expected to look similar to the one below:



If you used a Google account that does not have any calendar events like in this example, the "items" array is empty in the response. However, if you add an event to your Google Calendar and run the mapping again, the output will reflect that. As a side note, you could also retrieve events from a calendar other than the default one. For example, you could retrieve data from a public calendar like "Holidays in United States". To do this, set the value of **calendarId** parameter to `en.usa#holiday@group.v.calendar.google.com` instead of `primary`.

For information about other parameters that you can add to the API call, refer to the API method's documentation at https://developers.google.com/calendar/v3/reference/events/list.

4.5.10 Digital Certificate Management

Digital certificate management is an integral part of secure data exchange between a client computer and a Web server. Since mappings can be executed not only on Windows by MapForce, but also on a Windows, Linux or macOS server by MapForce Server (either standalone or in FlowForce Server execution), this section deals with managing HTTPS certificates on various platforms.

In the context of secure HyperText Transport Protocol (HTTPS), it is important to distinguish between server and client certificates.

Server certificates

A server certificate is what identifies a server as a trusted entity to a client application such as MapForce. The server certificate may be digitally signed by a commercial Certificate Authority, or it may be self-signed by your organization. In either case, while designing the mapping in MapForce, you can specify the following settings:

- Whether the server certificate must be checked.
- Whether the request must proceed if a mismatch has been detected between the name certificate and the name of the host.

These settings are available on the HTTP Security Settings dialog box of MapForce (see Setting HTTP Security 610). When you enable server certificate checks, consider the following:

- If you are calling a Web server whose certificate is signed by a trusted Certificate Authority, your operating system will likely be already configured to trust the server certificate, and no additional configuration is necessary.
- If you are calling a Web server which provides a self-signed certificate (for example, a local network server within your organization), you will need to configure your operating system as well to trust that certificate.

In most cases, you can check the level of trust between your operating system and the Web server by typing the URL of the Web service in the browser's address bar. If the server is not trusted, or if your operating system is not configured to trust the server, your browser will display a message such as "This connection is untrusted", or "There is a problem with this website's certificate". Note that you cannot use the browser to check the level of trust with a Web server if the browser uses a certificate database other than that of the operating system (for example, Firefox on Ubuntu).

On Windows, you can establish trust with the server by following the browser's instructions and importing or installing the required certificates into your system's Trusted Root Authorities store (see Trusting Server Certificates on Windows 645). On macOS, you can do the equivalent operation in Keychain Access (see Trusting Server Certificates on macOS 644). For instructions applicable to Linux, see Trusting Server Certificates on Linux 643.

Client certificates

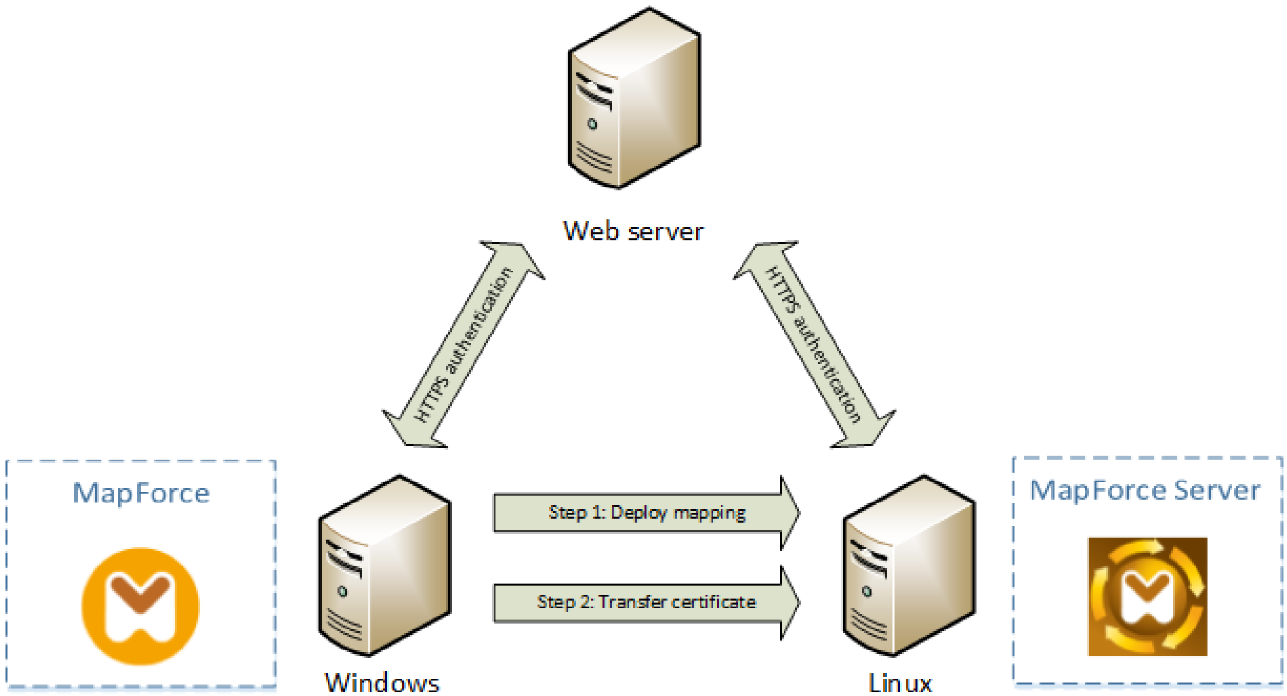
While server certificates are used to identify a server as a trusted entity, client certificates are primarily used to authenticate the caller against the Web server. If you intend to call a Web server which requires client certificates, you may need to contact the administrator of the Web server for the client configuration instructions. Taking IIS (Internet Information Services) as an example, the Web server may be configured to handle HTTPS and client certificates in one of the following ways:

- Require HTTPS and ignore client certificate
- Require HTTPS and accept client certificate
- Require HTTPS and require client certificate

The success or failure of the Web service request depends both on the configuration of the Web server and the client application. For example, if the Web server is configured to require a client certificate, then, for the call to be successful, the calling application must present a valid client certificate.

From a MapForce perspective, the same is true for mappings which include Web service calls through HTTPS. In particular, to run such mappings successfully, it is assumed that the Web server has been configured to accept or require the client certificate, and that the operating system where the mapping runs provides the correct client certificate to the Web server.

The diagram below illustrates a scenario where a client certificate used in MapForce is transferred to a Linux server running MapForce Server. Once the certificate has been transferred to the target operating system, MapForce Server can use it to authenticate itself against the Web server and execute the mapping successfully.



Deploying mappings with client certificates to another computer

For HTTPS authentication in Web service calls, MapForce is capable of using Transport Layer Security (TLS) on top of HTTP, which is the successor of Secure Sockets Layer (SSL) protocol. Note that fallback to SSL may occur if either the client implementation or the server does not support TLS.

To support Web calls with client certificate authentication on multiple platforms, MapForce (and MapForce Server) relies on the certificate management implementation of each platform, thus ensuring that certificate management is always in the scope of the underlying operating system. Each operating system provides different support for certificate management, as shown in the table below.

Platform	**Certificate management and implementation**
Windows	On Windows, you can manage certificates using the Certificate snap-in (see [Accessing the Certificate Stores on Windows](646)). TLS support is available through the *Secure Channel* (also known as *SChannel*) library.
Linux	On Linux, you can manage certificates using the OpenSSL (`openssl`) command line tool and library. If OpenSSL support is not already available on the Linux machine where MapForce Server is installed, you will need to download and install it before you can manage certificates. TLS support is available through the OpenSSL library (https://www.openssl.org/).

Platform	Certificate management and implementation
macOS	On macOS, you can manage certificates using the *Keychain Access Manager*, located under **Finder > Applications > Utilities**. TLS support is provided by the *Secure Transport* library native to the operating system.

If you execute the mapping on a Windows operating system where you can already successfully consume the same Web service that you intend to call from MapForce, no additional certificate configuration is normally required (for the conditions to run the mapping successfully on Windows, see Client Certificates on Windows 656). However, if you design mappings with MapForce on a Windows computer, and then deploy them to another computer (which may run a different operating system), the client certificate is not stored or copied together with the deployed package. For the Web service call (and the mapping) to execute successfully, the client certificate must exist on the target operating system as well.

To transfer a certificate from a Windows system to another Windows-based computer, export the required certificate (with private key) from the source system (see Exporting Certificates from Windows 649). Then import the same certificate to the **Current User\Personal** store on the target operation system (see Client Certificates on Windows 656).

For instructions on how to transfer client certificates to the Linux and macOS platforms, see Client Certificates on Linux 654 and Client Certificates on macOS 655, respectively.

4.5.10.1 Trusting Server Certificates on Linux

On Linux, you can import a trusted certificate into the system's certificate store as shown below.

> Perform the following steps only if you are sure of the authenticity of the certificate you want to trust.

On Debian and Ubuntu, follow the steps below:

1. Copy the certificate file of the Web server to the following directory.

```
sudo cp /home/downloads/server_cert.crt /usr/local/share/ca-certificates/
```

2. Update the certificate store as follows:

```
sudo update-ca-certificates
```

On CentOS, follow the steps below:

1. Install the `ca-certificates` package:

```
yum install ca-certificates
```

2. Enable the dynamic certificate authority configuration feature:

```
update-ca-trust enable
```

3. Copy the server certificate to the following directory:

```
cp server_cert.crt /etc/pki/ca-trust/source/anchors/
```

4. Use the command:

```
update-ca-trust extract
```

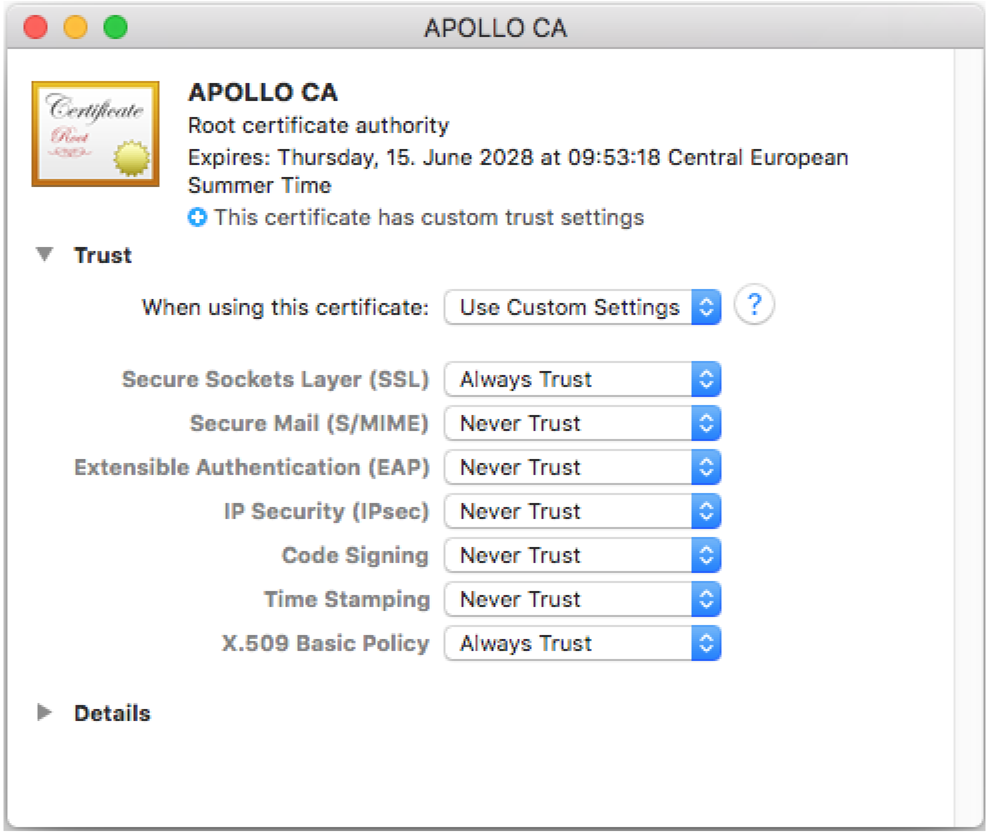
For cases where you need to access the server only through the browser, it is sufficient to import the certificate into the browser certificate store. The exact instructions will vary for each browser. For example, in Firefox 59.0.2, you can do this as follows:

1. Under **Options | Privacy & Security**, click **View Certificates**.
2. On **Authorities** tab, click **Import** and browse for the root certificate file created previously.
3. When prompted, select **Trust this CA to identify websites**.

4.5.10.2 Trusting Server Certificates on macOS

On macOS, you can import a trusted certificate into Keychain Access as follows.

1. Run Keychain Access.
2. Click **System**, and then click **Certificates**.
3. On the **File** menu, click **Import Items**.
4. Browse for the trusted certificate, and click **Open**.
5. Enter the Keychain Access password when prompted, and then click **Modify Keychain**.
6. Double-click the certificate, expand the **Trust** section, and select **Always Trust**.



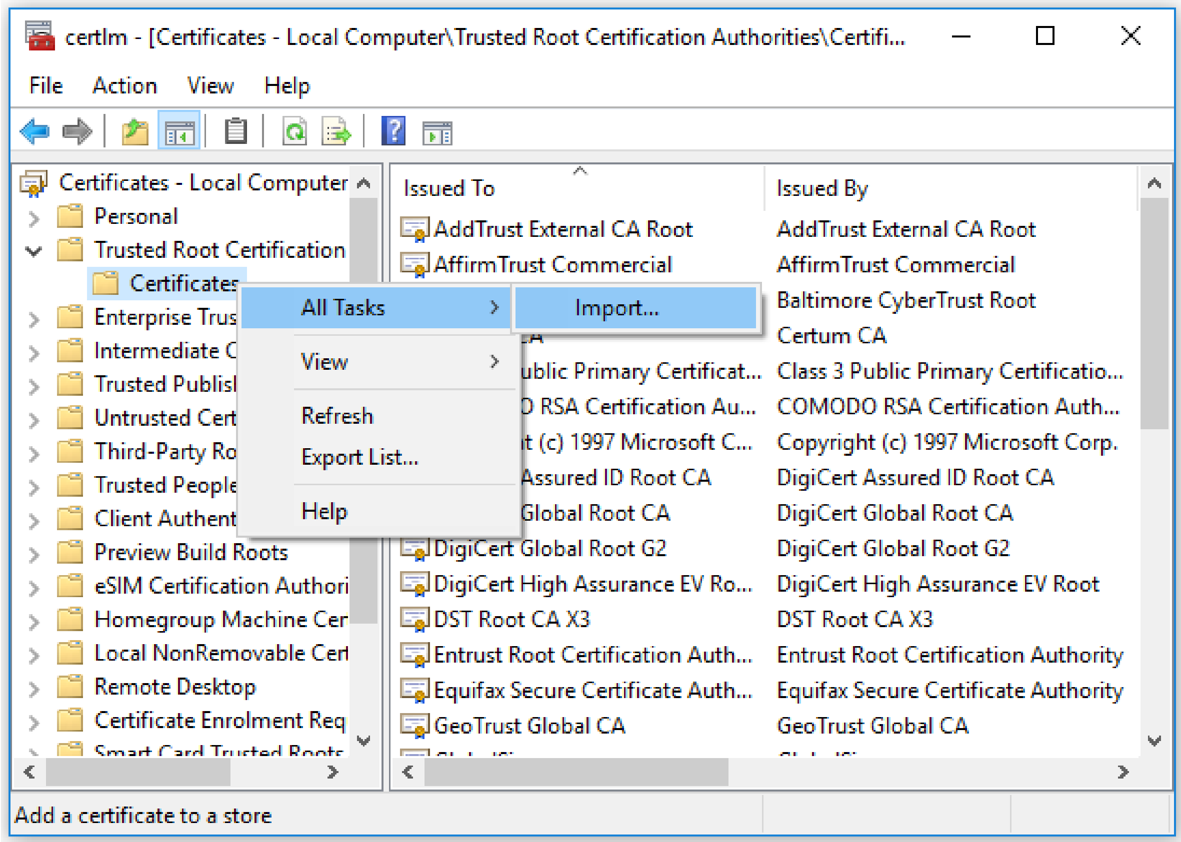
4.5.10.3 Trusting Server Certificates on Windows

On Windows, you can import a trusted certificate into the system certificates store as follows:

1. Open the Windows certificate store *for the computer account*, see Accessing Windows Certificate Store 646 .

Perform the following steps only if you are sure of the authenticity of the Web server certificate.

2. Under "Trusted Root Certification Authorities", right-click **Certificates**, and select **All Tasks | Import**, and follow the certificate import wizard.



For more information, see https://technet.microsoft.com/en-us/library/cc754489(v=ws.11).aspx.

4.5.10.4 Accessing the Certificate Stores on Windows

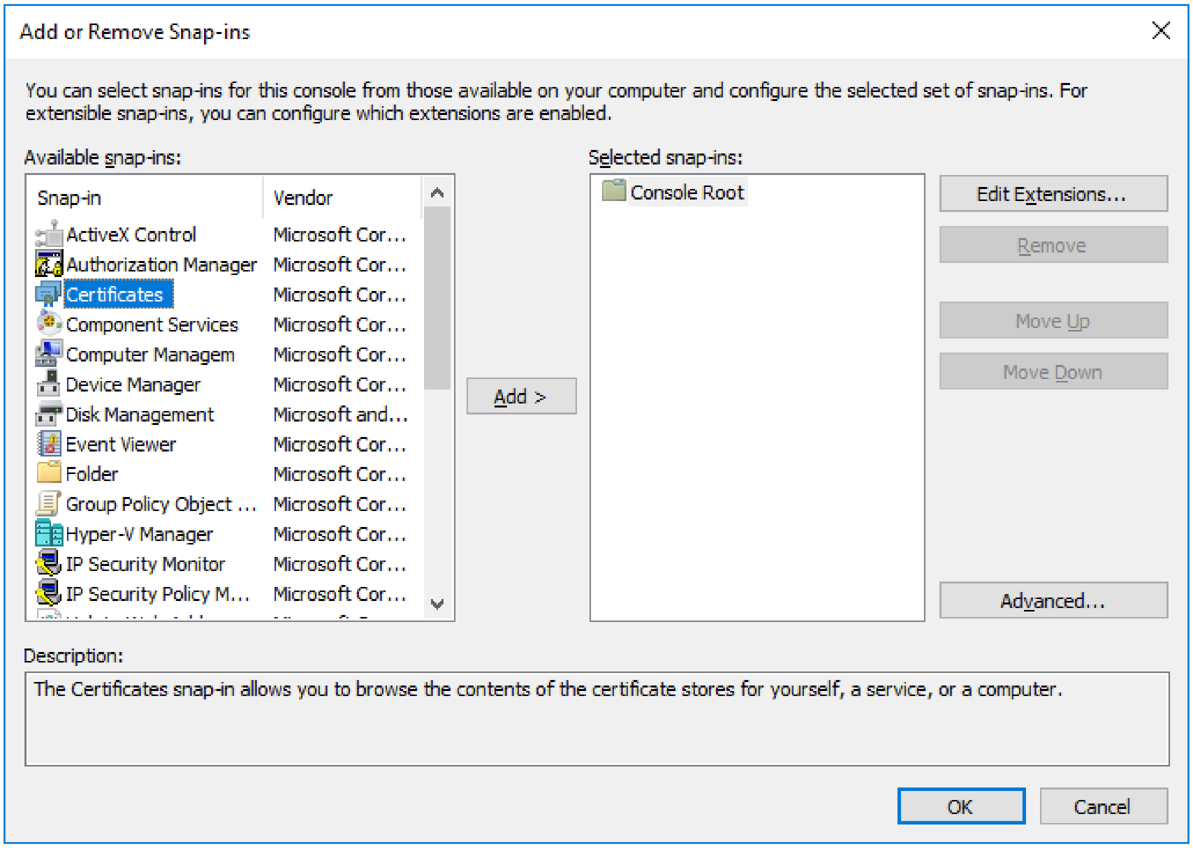
On Windows, you can manage certificates from the Microsoft Management Console (MMC) snap-in, either for your user account, or for the computer account.

To open the Certificates snap-in (for the current Windows user):

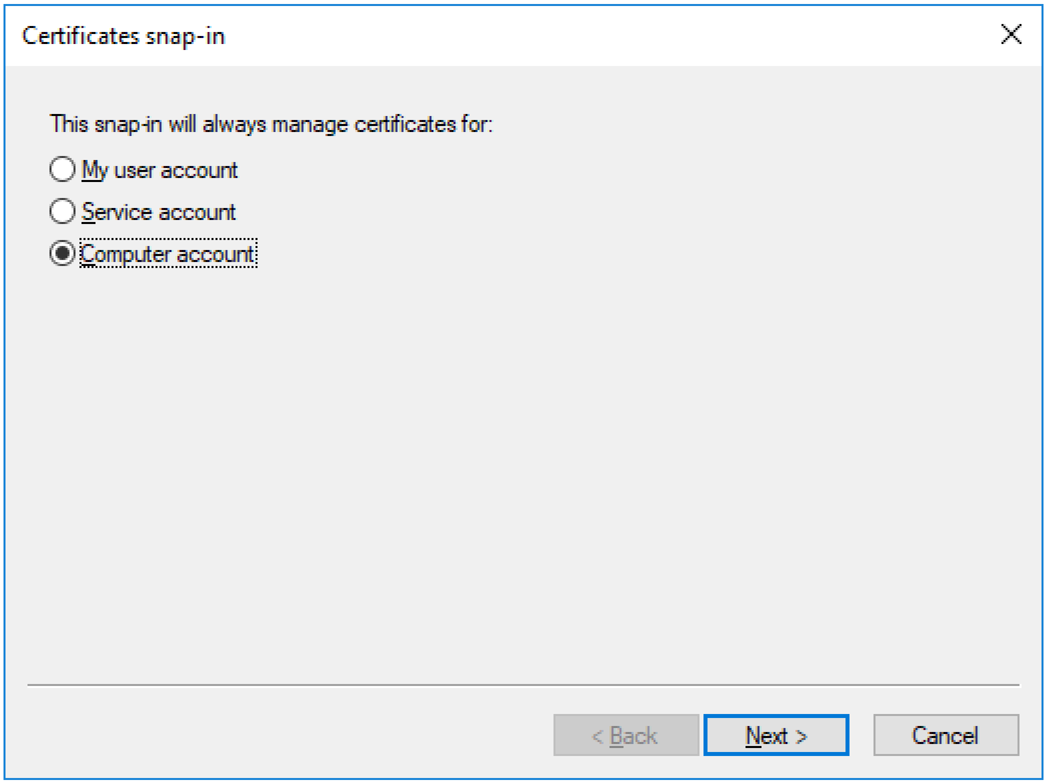
- Run `certmgr.msc` at the command line.

To open the Certificates snap-in (for the computer account):

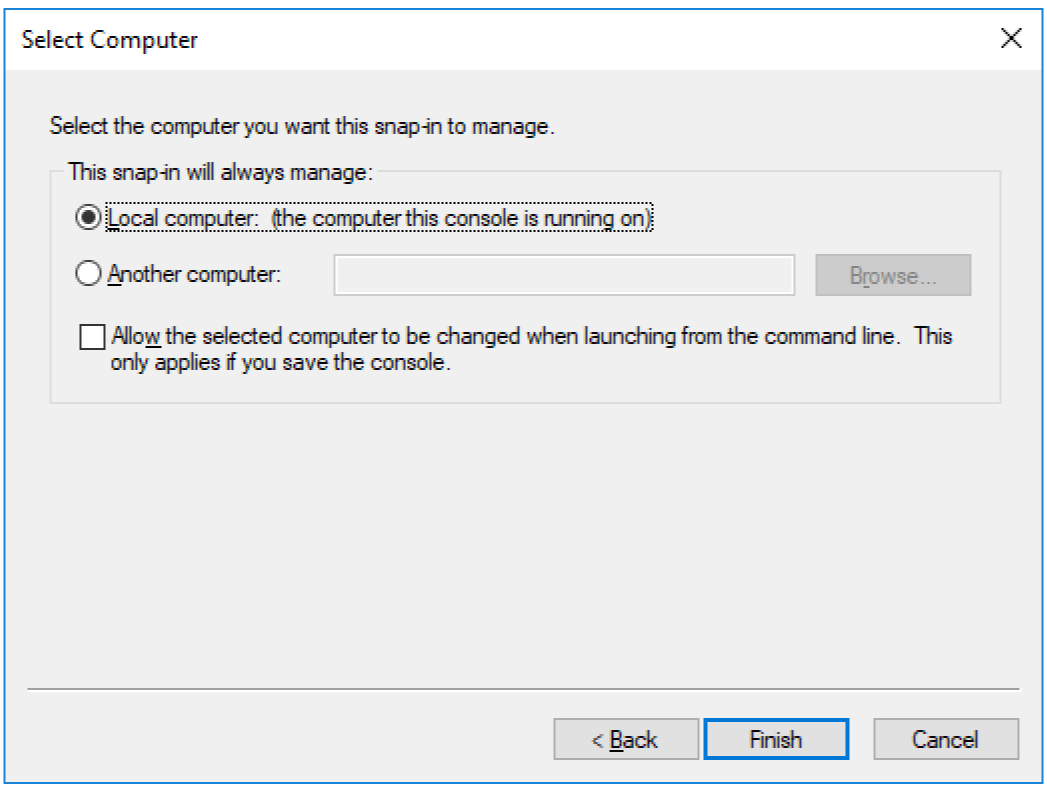
1. Run `mmc` at the command line.
2. On the **File** menu, click **Add/Remove Snap-in**.
3. Click **Certificates**, and then click **Add**.



4. Click **Computer account**, and click **Next**.



5. Click **Local computer**, and then click **Finish**.



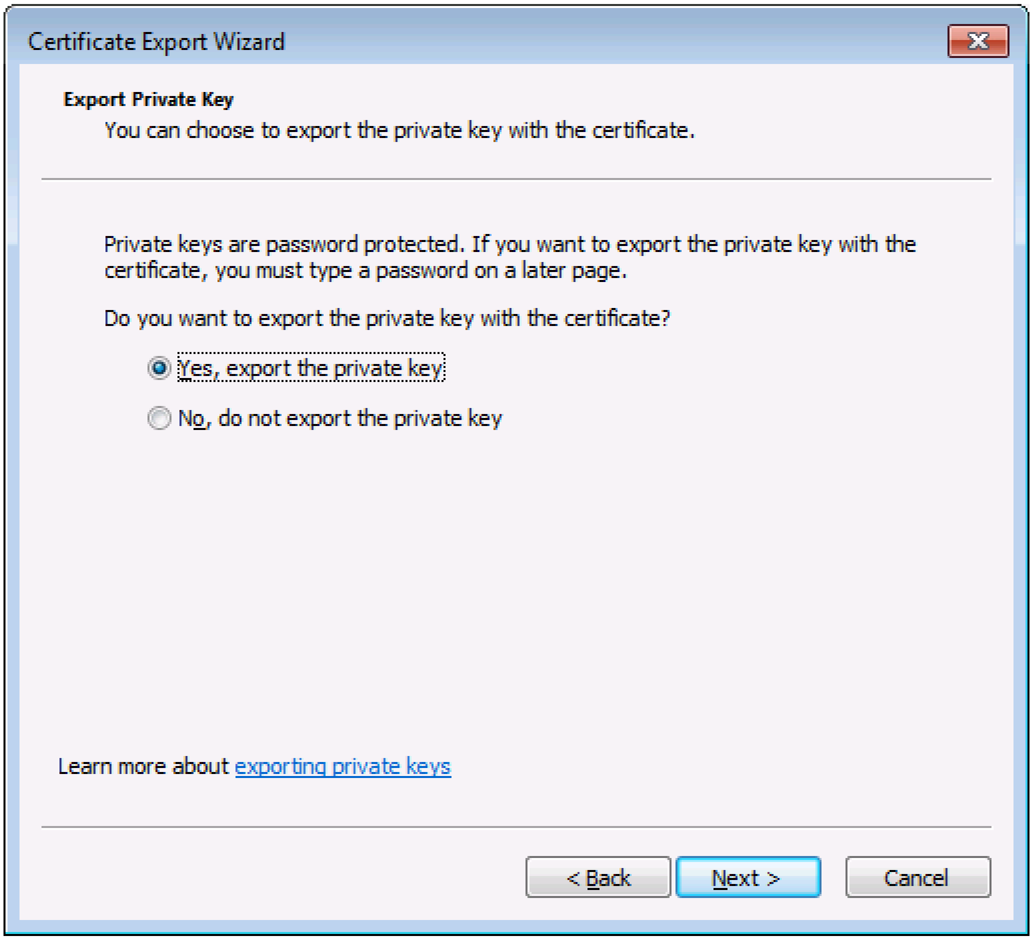
4.5.10.5 Exporting Certificates from Windows

For mappings that call Web services through HTTPS and are deployed to a macOS or Linux server running MapForce Server or FlowForce Server, the same client certificate must be available on the non-Windows operating system as the one used on Windows to design and test the mapping. To execute such mappings on a non-Windows operating system with MapForce Server, export the required certificate with private key from Windows and then import it into the target operating system.

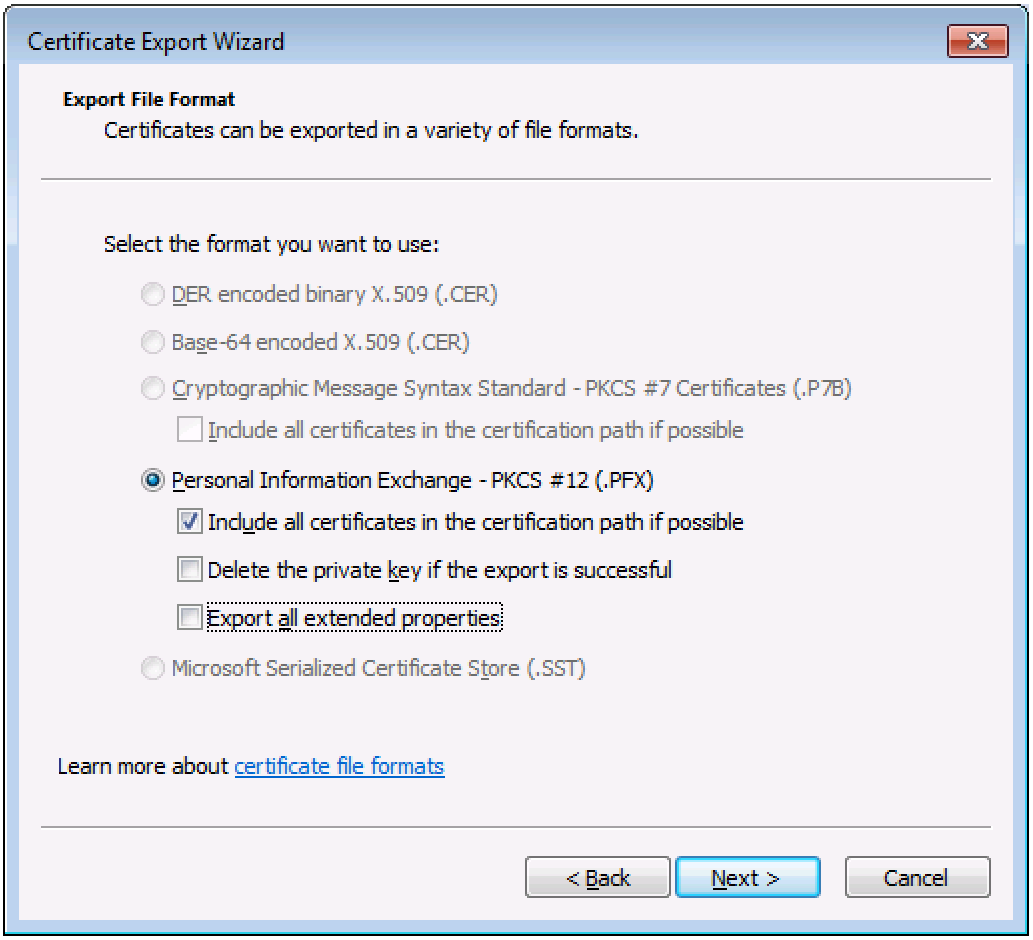
To export a certificate with private key from Windows:

1. On Windows, open the Certificates snap-in (see Accessing the Certificate Stores on Windows 646).
2. Right-click the certificate that you want to export, point to **All Tasks**, and then click **Export**.
3. Click **Next**.

4. Choose to export from Windows the certificate together with its private key, and then click **Next**.

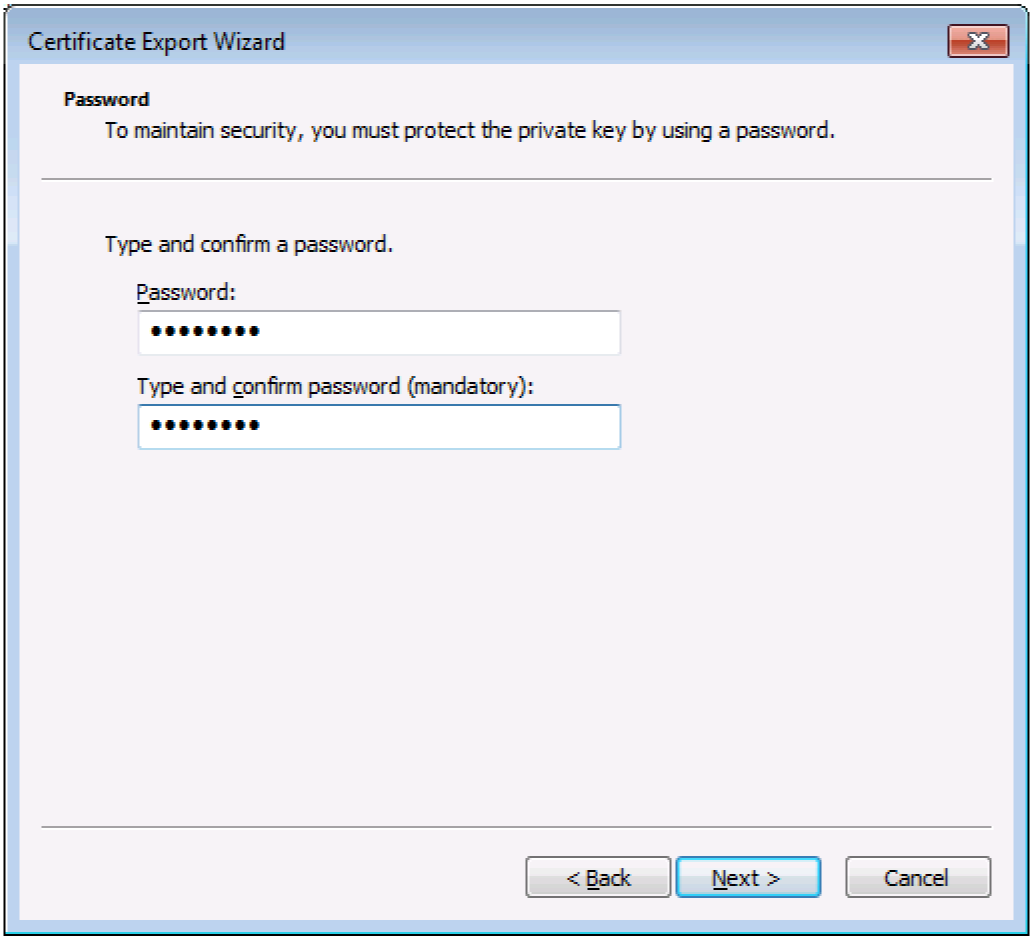


5. Choose the *Personal Information Exchange - PKCS #12 (.pfx)* file format, and then click **Next**.

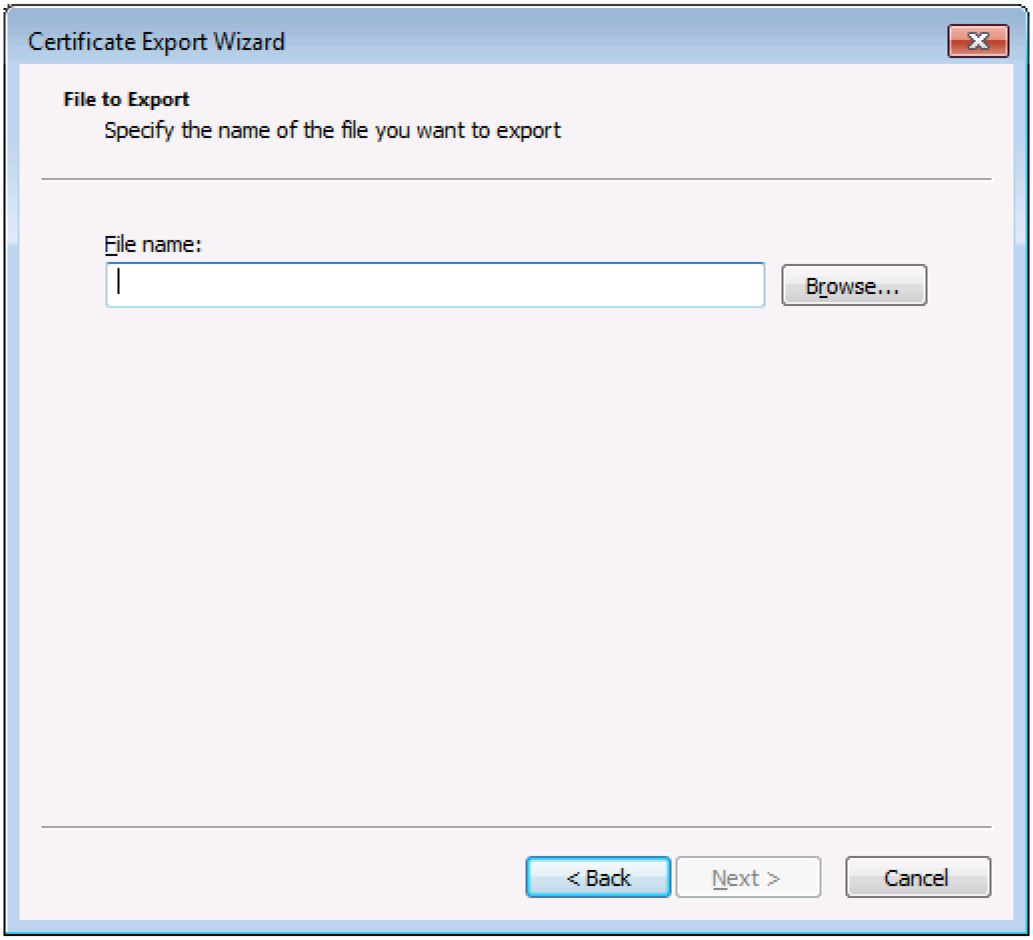


Note: Make sure not to select the option **Delete the private key if the export is successful**, otherwise you will not be able to make use of the certificate after it is exported.

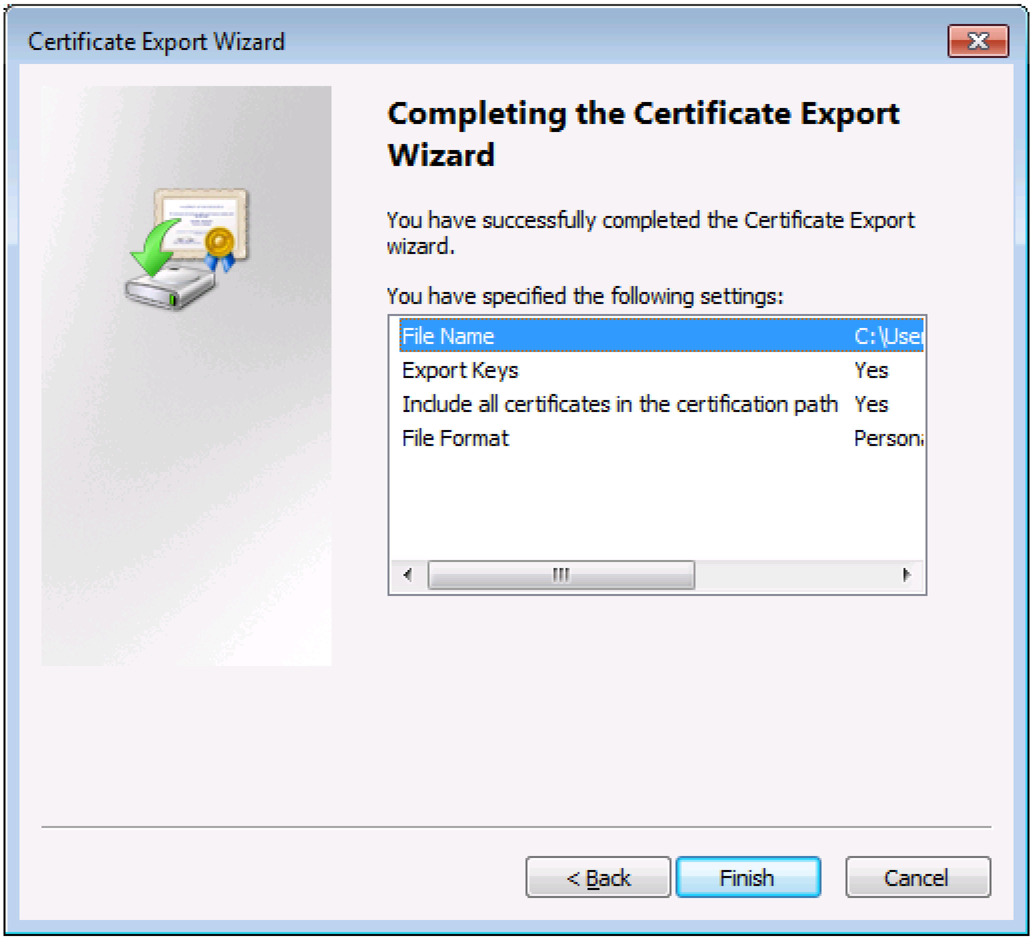
6. Enter a password, and then click **Next**. You will need this password after you copy the certificate to the target operating system.



7. Browse for the location of the file to export, and then click **Next**.



8. Click **Finish**.



4.5.10.6 Client Certificates on Linux

If your mappings include Web service authentication through HTTPS by means of client certificates, follow these steps to deploy such mappings to a Linux machine running MapForce Server:

1. Open the mapping which calls the Web service.
2. Double-click the header of the Web Service component. The Component Settings dialog box appears.
3. Click **Edit** next to HTTP Security Settings.
4. In the HTTP Security Settings dialog box, click **Client Certificate**, and then select the required certificate from the **Current User\Personal** store on Windows (see Setting HTTP Security 610).
5. Save the mapping and compile it to a mapping execution file or deploy it to FlowForce Server (see Compiling Mappings to Server Execution Files 1378 and Deploying Mappings to FlowForce Server 1381).
6. Transfer the client certificate required by the Web service call to the target operating system. Make sure that the certificate has a private key, and that the **Enhanced Key Usage** property of the certificate includes "Client authentication" as purpose.

To transfer the client certificate to Linux:

1. Export the client certificate with private key from Windows, in the *Personal Information Exchange -*

PKCS #12 (.pfx) file format (see Exporting Certificates from Windows 649).
2. Copy the certificate file to the Linux machine.
3. Convert the .pfx file to .pem format using the command:

```
openssl pkcs12 -in cert.pfx -out "John Doe.pem" -nodes
```

This command parses the .pfx file and outputs a .pem file, without encrypting the private key. Certificates with an encrypted private key prompt for password and are not supported in server execution.

Executing the mapping

To instruct MapForce Server to use the .pem file as client certificate, set the `--certificatespath` parameter when running the mapping. The `--certificatespath` parameter defines the path of the directory where all certificates required by the current mapping are stored. For example, if the certificate file path is `/home/John/John Doe.pem`, then `--certificatespath` must be set to `/home/John`.

By default, if the `--certificatespath` parameter is not provided, MapForce Server looks for certificates in the directory `$HOME/.config/altova/certificates` of the current user.

> For the mapping to execute successfully, the certificate file is expected to have the .pem extension and the file name must match the Common Name (CN) of the certificate, including spaces (for example, **John Doe.pem**). If the CN contains a forward slash (/), it must be replaced with an underscore (_) character.

If you intend to execute the mapping as a FlowForce Server job, copy the certificate file to the `$HOME/.config/altova/certificates` directory. When running the job, FlowForce Server will use this directory to look for any certificate files required by the mapping.

> For security considerations, make sure that certificate files are not readable by other users, since they contain sensitive information.

4.5.10.7 Client Certificates on macOS

If your mappings include Web service authentication through HTTPS client certificates, follow these steps to deploy such mappings to a macOS running MapForce Server:

1. Open the mapping which calls the Web service.
2. Double-click the header of the Web Service component. The Component Settings dialog box appears.
3. Click **Edit** next to HTTP Security Settings.
4. In the HTTP Security Settings dialog box, click **Client Certificate**, and then select the required certificate (see Setting HTTP Security 610).
5. If the certificate name does not match exactly the host name of the server, select **Allow name mismatch between certificate and request**.
6. Save and deploy the mapping to the target operating system (see Deploying a MapForce mapping 1381).
7. Transfer the client certificate required by the Web service call to the target operating system. Make sure that the certificate has a private key, and that the **Enhanced Key Usage** property of the certificate includes "Client authentication" as purpose.

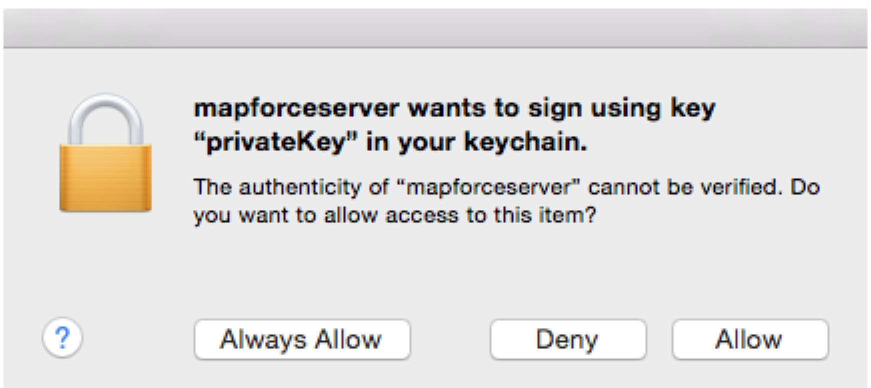
To transfer the client certificate to macOS:

1. Export the client certificate with private key from Windows, in the *Personal Information Exchange - PKCS #12 (.pfx)* file format (see Exporting Certificates from Windows 649) and copy the .pfx file to the macOS.
2. If this hasn't been done already, make sure that the operating system trusts the server certificate (see Trusting Server Certificates on Mac OS 644).
3. Run Keychain Access from **Finder > Applications > Utilities**.
4. On the **File** menu, click **Import Items**.
5. Browse for the client certificate exported from Windows in step 1 and select a destination keychain.
6. Click **Open** and enter the password with which the certificate was encrypted.

Executing the mapping

You are now ready to run the mapping using the MapForce Server `run` command. Note the following:

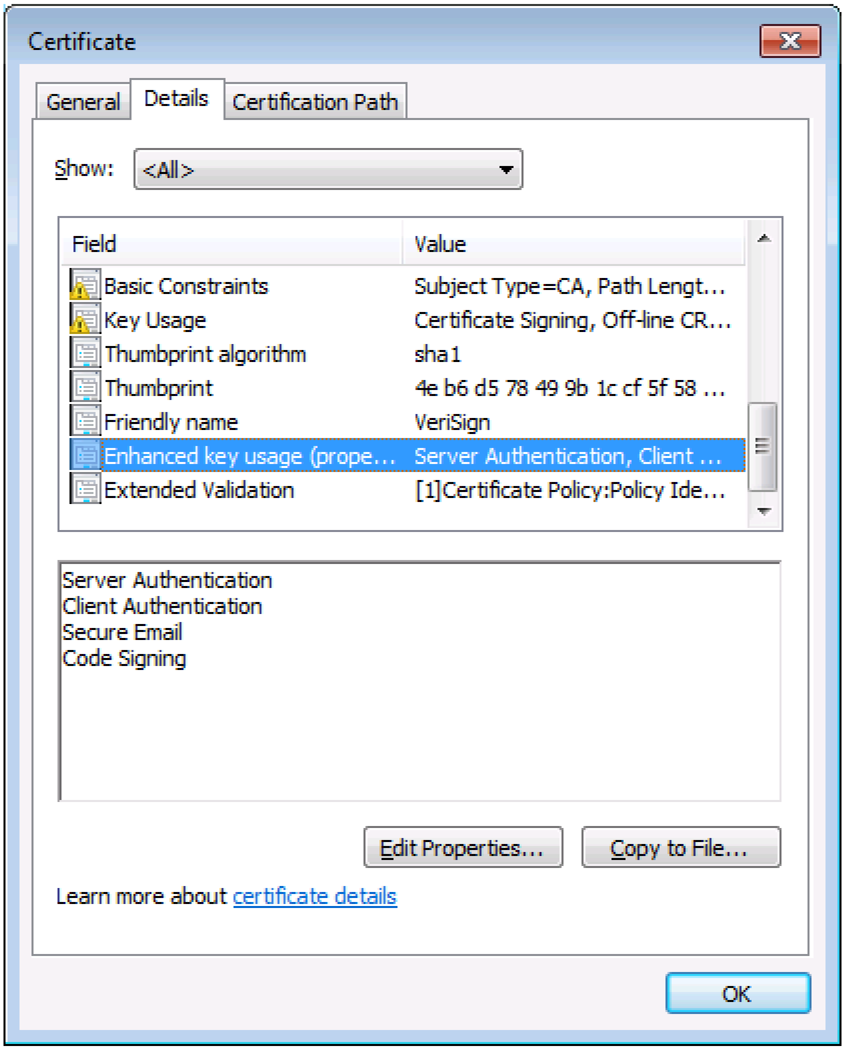
- If you execute the mapping remotely through SSH, first unlock the keychain with the `security unlock-keychain` command.
- If you execute the mapping through the macOS graphical user interface, when prompted to allow MapForce Server access to the keychain, click **Allow**.



4.5.10.8 Client Certificates on Windows

When you run on Windows a mapping which requires client certificates, the conditions to run the mapping successfully are as follows:

- The client certificate must exist in the **Current User\Personal** certificate store (also referred to as the **My** store). For the certificate to exist in this store, it must be imported through the Certificate Import Wizard. For instructions, see https://technet.microsoft.com/en-us/library/cc754489(v=ws.11).aspx.
- The certificate must have a private key.
- The **Enhanced Key Usage** property of the certificate must include "Client authentication" as purpose.



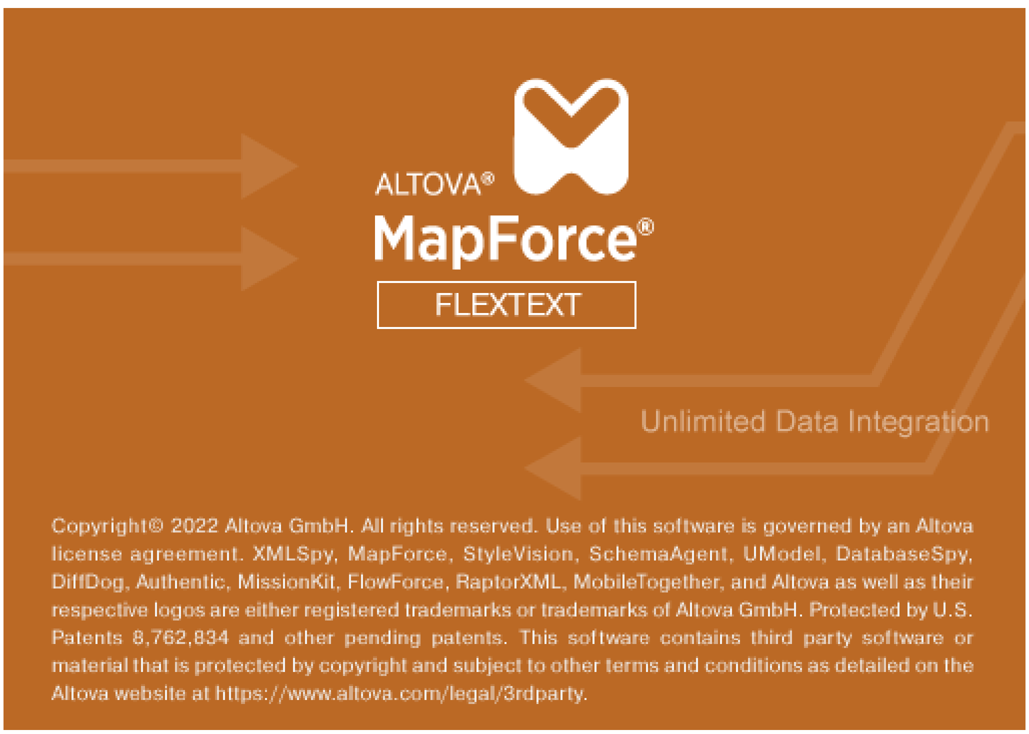
In the current version of MapForce, due to a limitation of the library used by MapForce, Windows will select the required certificate automatically from the certificate store when you run the mapping. The mapping will execute successfully if, after filtering the **Current User\Personal** certificate store, the server finds a suitable certificate. Note that the HTTPS authentication (and the certificate selection operation) is managed by Windows and is not controlled by MapForce or MapForce Server. In some cases, if multiple certificates exist in the **Current User\Personal** store, an unsuitable certificate may be selected automatically by the operating system, which causes the mapping execution to fail. This situation can be avoided by limiting the number of certificates available in the **Current User\Personal** store.

4.6 MapForce FlexText

Altova web site: [Text and Flat File Mapping]

FlexText is a MapForce module which enables you to convert data from non-standard or legacy text files of high complexity to other formats supported by MapForce. While XML files have a schema or a structure from which MapForce can derive the schema, this is not the case of text files, especially when they have a complex and unique structure that does not consistently fit into CSV or fixed-length field patterns. Moreover, sometimes you need to extract only portions of useful data from a legacy text file. FlexText solves these problems by helping you define and test visually, in real-time, the rules, or the template, according to which text data must be split down into mappable items. A FlexText template essentially defines the structural model of your custom text data, according to criteria you specify, for the purpose of mapping it to other formats.

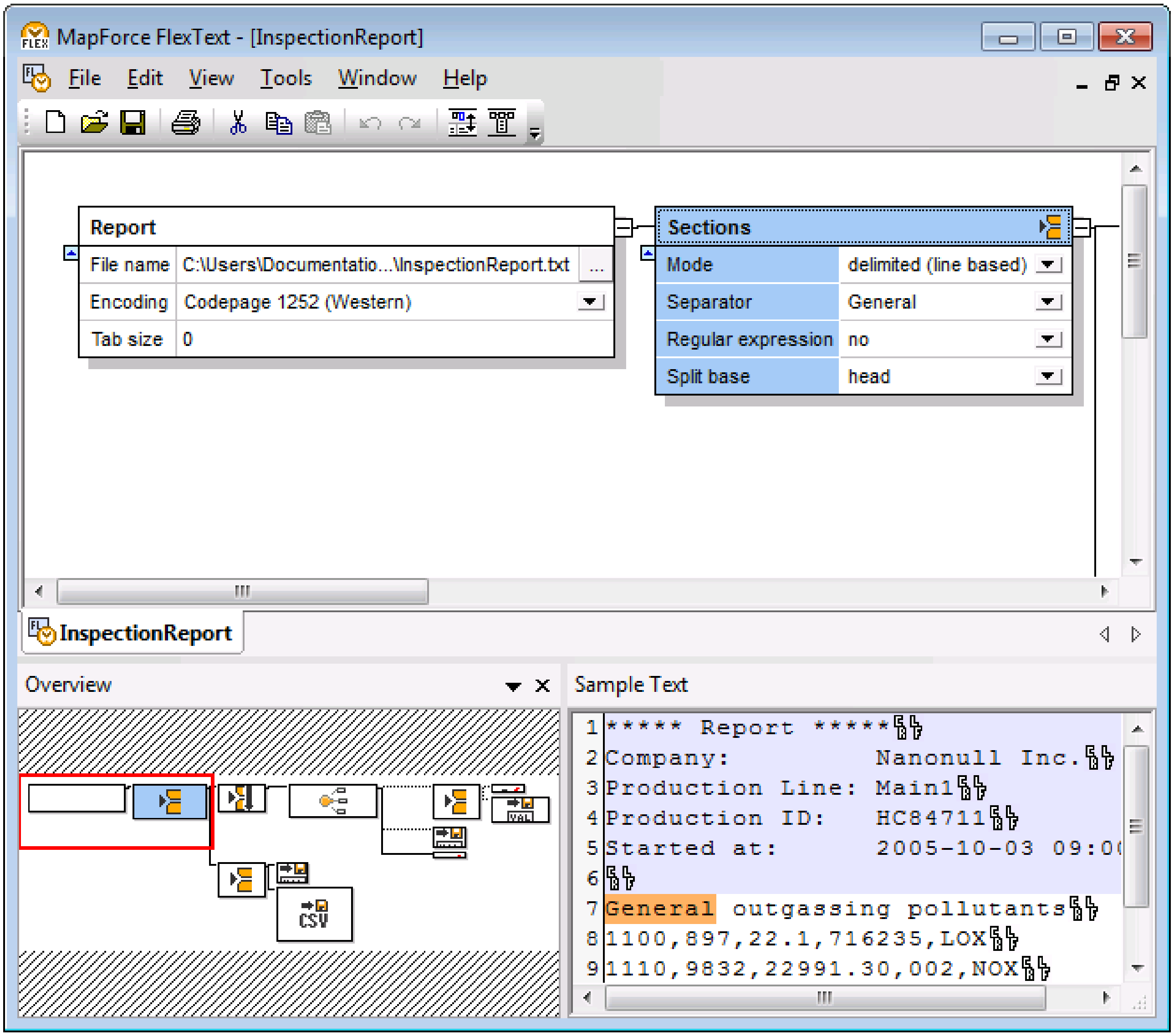
Once you define a FlexText template, you can add it to the MapForce mapping area as a source or target component, and thus map complex and non-standard text data with any other format supported by MapForce. FlexText template files have the .mft (MapForce FlexText) extension. You can reuse the same FlexText template in multiple mappings, to process any number of text files.



4.6.1 Overview

You can start FlexText directly from MapForce, when adding text files as mapping components (use the **Insert | Text File** menu command). You can also start FlexText as a standalone program, by running the *Altova MapForce FlexText* executable available in the MapForce installation directory.

The screen shot below illustrates a sample FlexText template (**InspectionReport.mft**) which is available, along with several other FlexText samples, in the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.



The FlexText graphical Interface

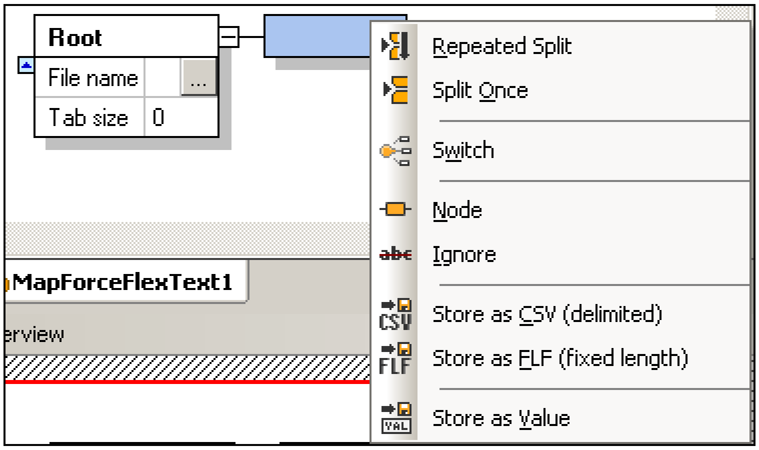
The FlexText interface consists of the following panes: Design pane, Overview pane and Sample Text pane. As you will see next, the position and appearance of these panes can be customized according to your preferences.

Design pane

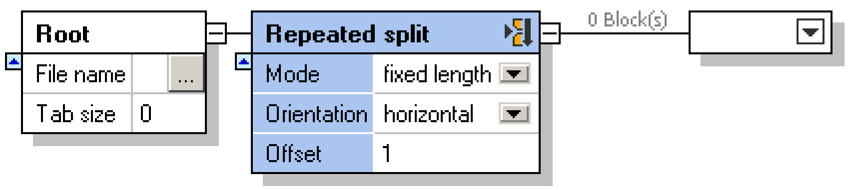
The Design pane is the working area where you define the structure of your text, by means of containers. Containers define the rules for handling text data according to criteria you specify. The first container of any FlexText template is the *Root* container, which represents the entire text file. All other containers essentially define the subsequent text processing logic (typically, splitting text into meaningful units from which you can map data). For example, the *Split Once* container splits a fragment of text into exactly two fragments. By default, the name of any container describes its function (for example, *Repeated Split*); however, you can change it if necessary. You typically define as many containers as demanded by the structure of the text file with which you are working.

The FlexText Reference 676 section of this documentation covers containers in more detail. However, to begin with, note the following about containers:

- Clicking the icon in the top-right corner of a container opens a pop-up menu from which you can select the container type.



- Each container has a number of options which dictate what happens to text data at that particular place in the structure. These options determine the content of the container, and enable you to refine it before providing it to the MapForce component.



- To show or hide container contents, click the **Expand** or **Collapse** icon, respectively.
- To collapse containers as a group, press the **Shift** key. Two chevrons appear. Clicking the handle collapses the section of the container tree to the right of the one clicked.
- To preview the text created by a particular container directly in the design pane, enable the **Node Text in Design view** toolbar button.
- When the **Auto-collapse unselected node text** toolbar button is enabled, the full contents of a container is expanded when you select it. All other containers are collapsed.

Overview pane

The Overview pane gives a bird's-eye view of the Design pane. To navigate the Design pane, click and drag the red rectangle. To detach the Overview pane and reposition it elsewhere in the interface, click its title bar and drag the pane to the desired location.

Sample Text pane

The Sample Text pane displays the contents of the currently selected container. (Note that this pane is not shown if the **Node Text in Design view** toolbar button is enabled. Instead, the contents of the currently selected container is shown in the Design pane.)

To help you quickly identify the **Tab** and **Carriage Return/Line Feed** characters in the preview text, FlexText displays such characters using the and symbols, respectively.

4.6.2 FlexText Tutorial

The tutorial will show you how to use the most common, and most powerful, features of FlexText to process a text file and map its output in various ways in MapForce.

The example uses the **Flex-tutorial.txt** file available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. The .txt file has the following format:

```
111,332.1,22537.7,5,Container ship,Mega,
111,A1579227,10,3,400,Microtome,
111,B152427,7,6,1200,Miscellaneous,
222,978.4,7563.1,69,Air freight,Mini,
222,ZZAW561,10,5,10000,Gas Chromatograph,,

General outgassing pollutants
1100,897,22.1,716235,LOX
1110,9832,22991.30,002,NOX
1120,1213,33.01,008,SOX
```

Aim of the tutorial:

- To separate out the records containing 111, and 222 keys, into separately mappable items.
- To discard the plain text record.
- To create a CSV file of the remaining records.

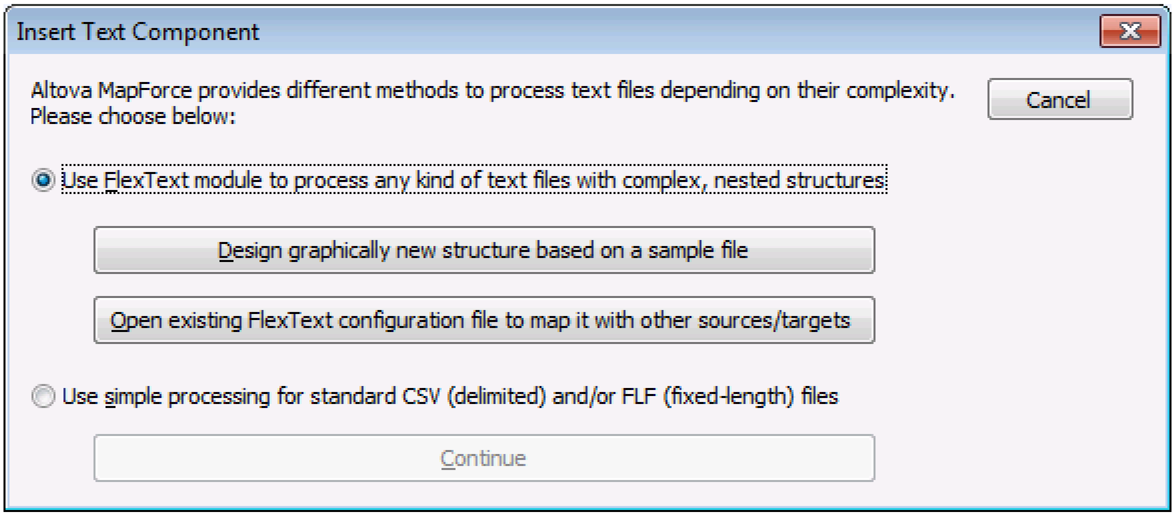
The tutorial is organized into the following parts:

4.6.2.1 Step 1: Create the FlexText Template

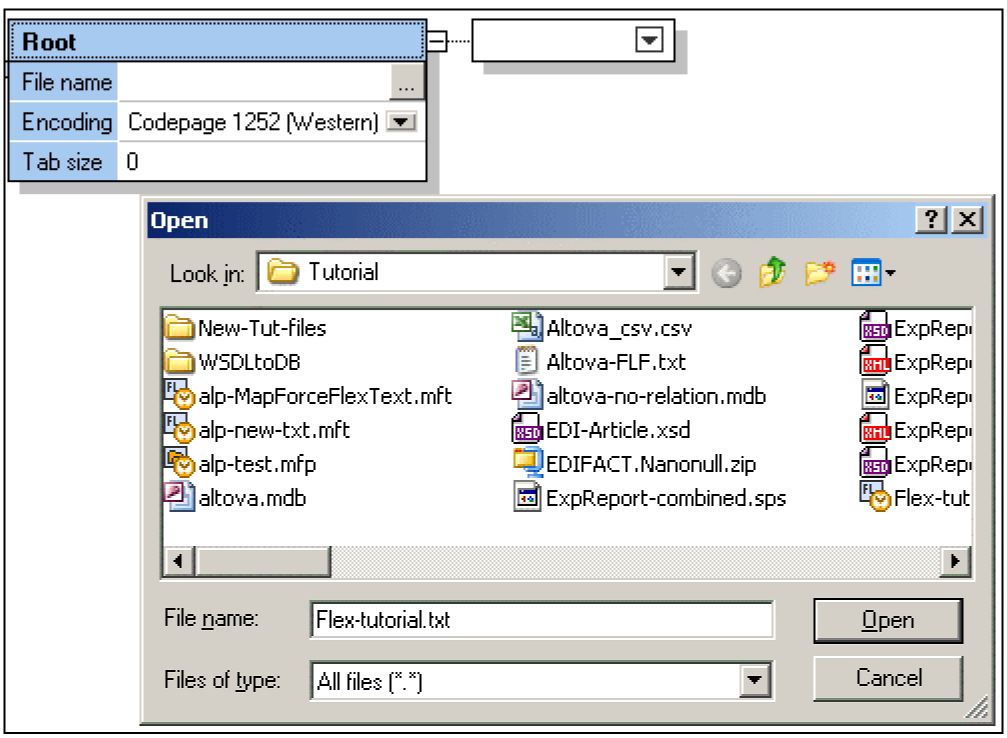
To create the FlexText template:

1. Start MapForce, and open a new mapping file.

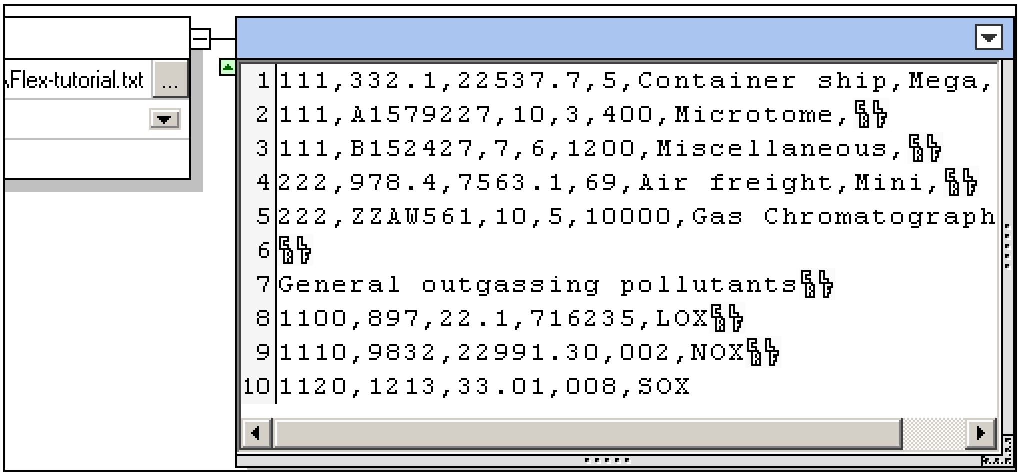
2. Select **Insert | Text file**, or click the Insert Text file icon .
3. Click the "**Design graphically new structure ...**" button.



4. Enter a name for your FlexText template, and click Save to continue (e.g. Flex-tutorial.mft). An empty design, along with the "Open" dialog box are displayed.



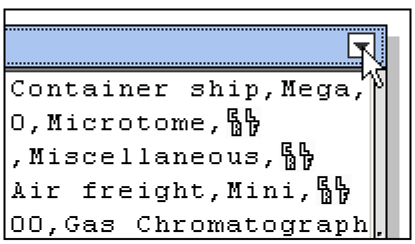
5. Select the **Flex-tutorial.txt** file in the **...\MapForceExamples\Tutorial** folder, and confirm by clicking Open.



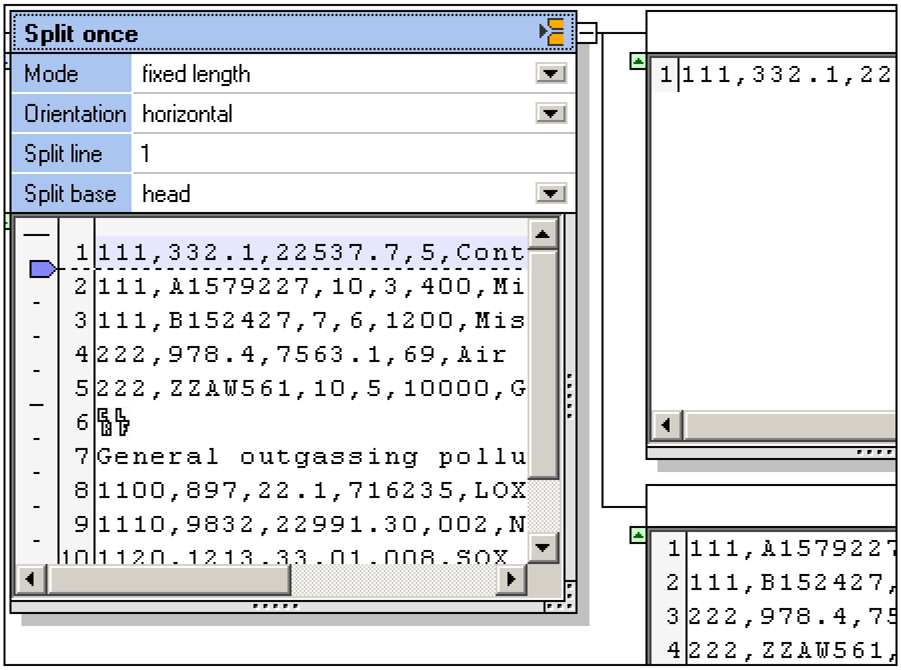
The text file contents are now visible.

Clicking the "Node Text in Design view" icon , displays the active container contents, in the Sample Text pane.

Activating "Auto-collapse unselected node text"  , displays the content in the active container, all other containers which contain content, are collapsed.



6. Click the container icon at the top right, and select **Split once** from the pop-up menu. Two new containers appear next to the Split once container. For more information on the Split once condition, please see: Split once 684.



The default settings of the Split once container are visible: fixed length, horizontal and split line=1.

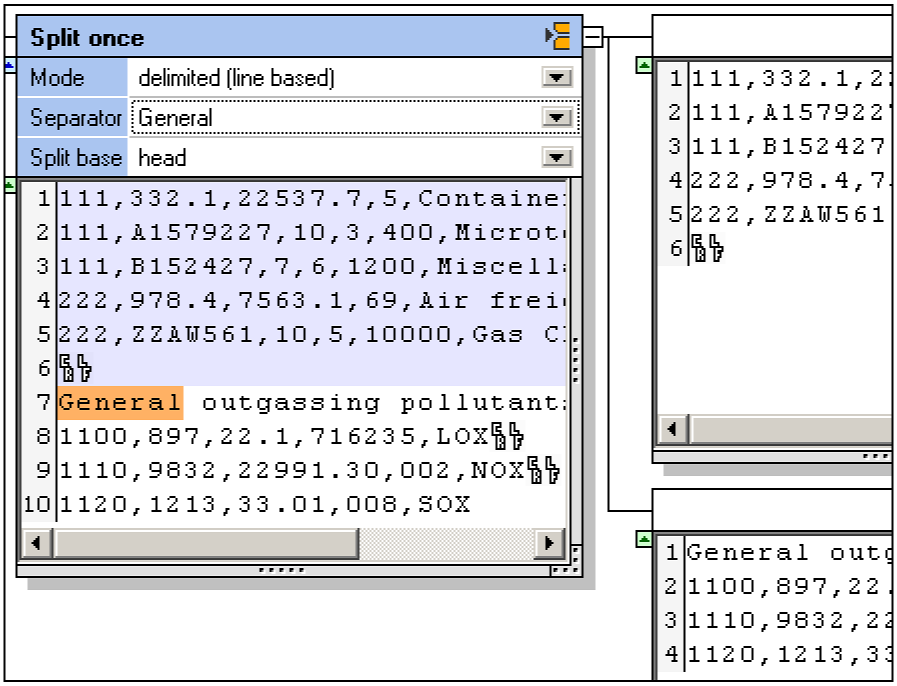
The result of these default settings are also visible:

- The top container contains the first line of the text file, highlighted in the **Split once** container.
- The lower container contains the rest of the text file.

4.6.2.2 Step 2: Define Split Conditions

Split conditions allow you to segment text fragments in various ways. To define split conditions:

1. Click the **Mode** combo box and select "delimited (line based)".
2. Double-click the Separator field and enter "General".

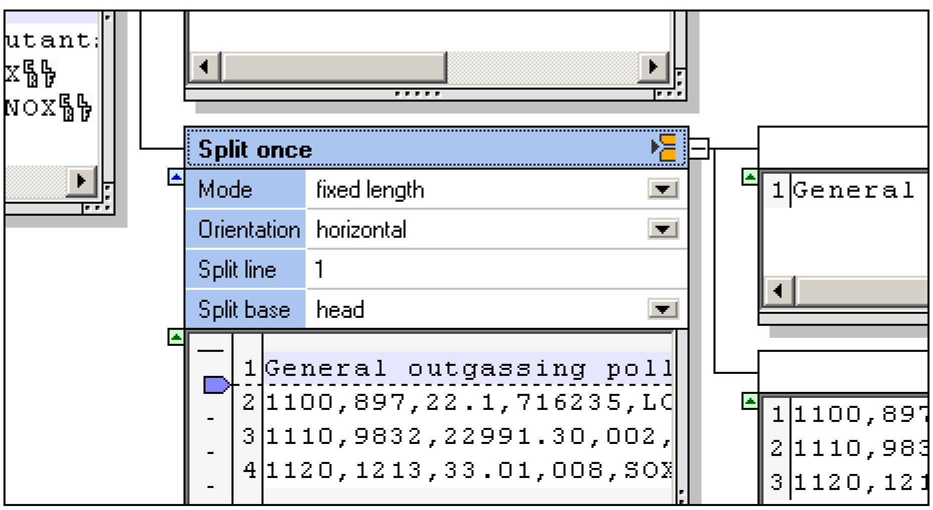


The text fragments in the respective containers have now changed.

Entering "General" and using delimited (line based), allows you to split off that section of text that contains the string "General", into the lower container. The text fragment up to the separator, is placed in the top container.

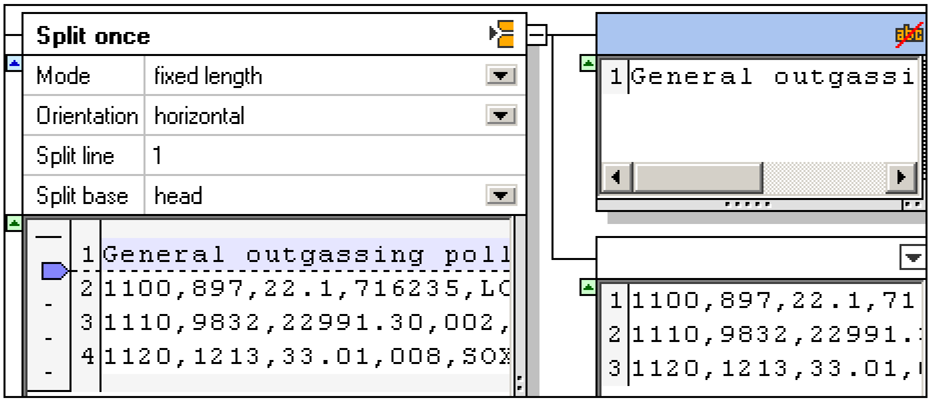
What we want to do now, is work on the lower container to produce a CSV file containing the records with 1100 and up.

3. Click the lower container and change it to **Split once**.



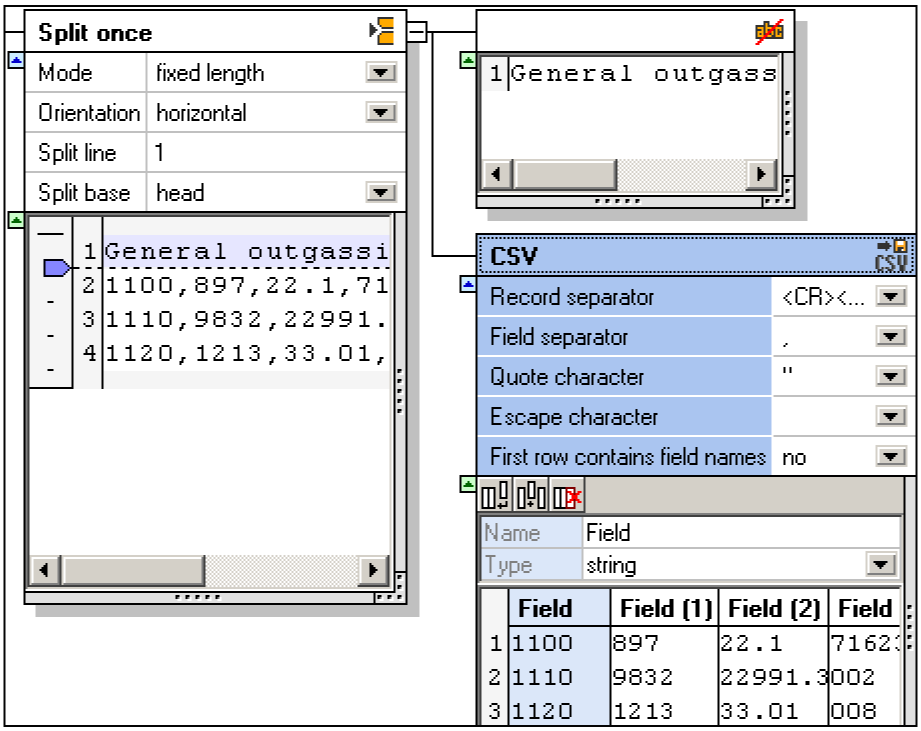
Two new containers are created. The default settings can remain as they are, because we now want to split off the first line of this text fragment, and ignore it. The remaining fragment in the lower container will be made into a CSV file.

4. Click the top container and change it to **Ignore**.



The text fragment, and thus mapping item, of this container has now been made unavailable for mapping in MapForce.

5. Click the lower container icon and change it to **Store as CSV**.



The container now shows the text fragment in a tabular form. The default settings can be retained.

Configuring the CSV file

If you want to change the field names, click the field, in the table, and then change the entry in the **Name** field. Columns can also be appended, inserted and deleted in this container, please see "Store as CSV" 696 for more information.

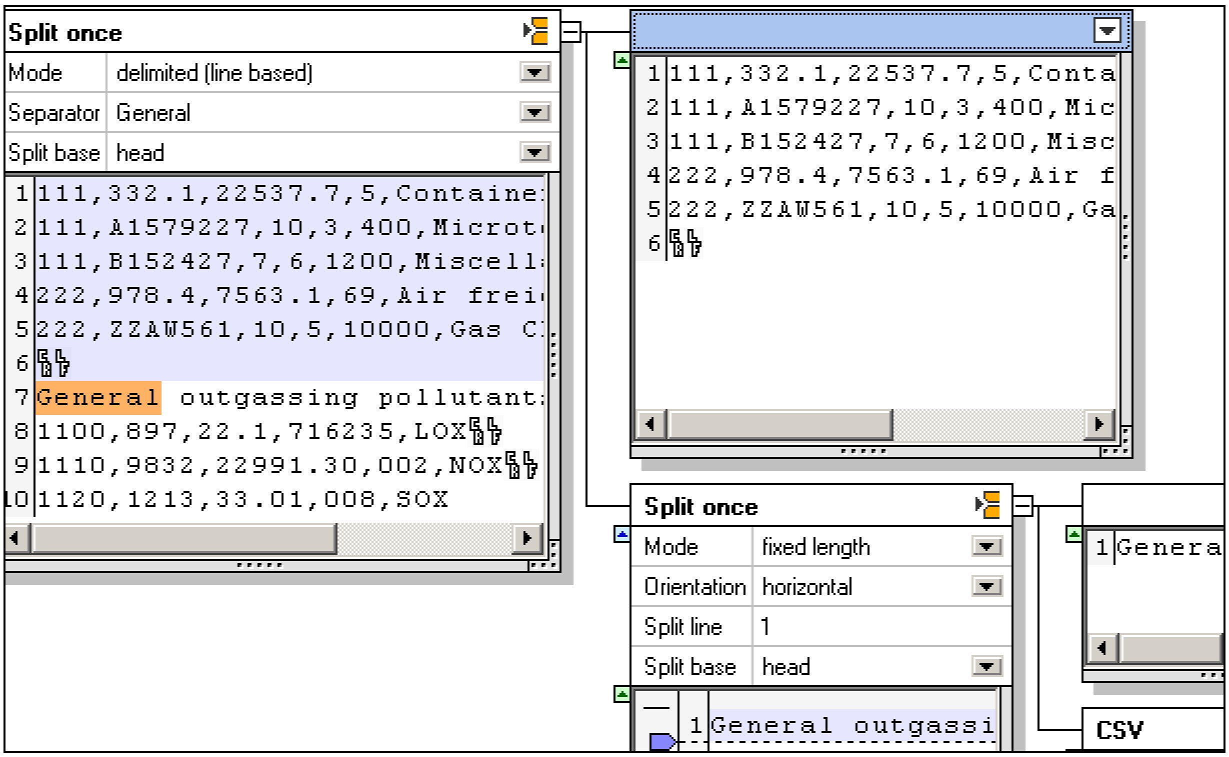
We can now continue with defining the remaining text fragment.

4.6.2.3 Step 3: Define Multiple Conditions per Container

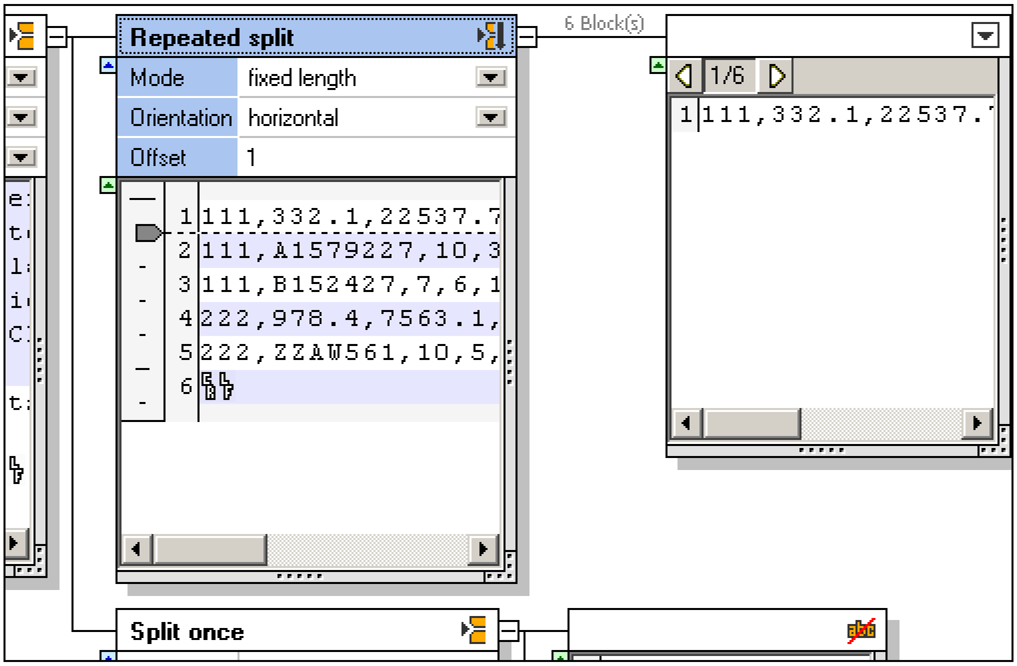
FlexText allows you to define multiple conditions per text fragment, using the Switch container. An associated container is automatically allocated to each condition that you define.

The current state of the tutorial at this point is that lower text fragment, of the first **Split once** container, has been defined:

- A "Split once" container splits off the first line into an Ignore container.
- The remaining segment is defined/stored as a CSV file.



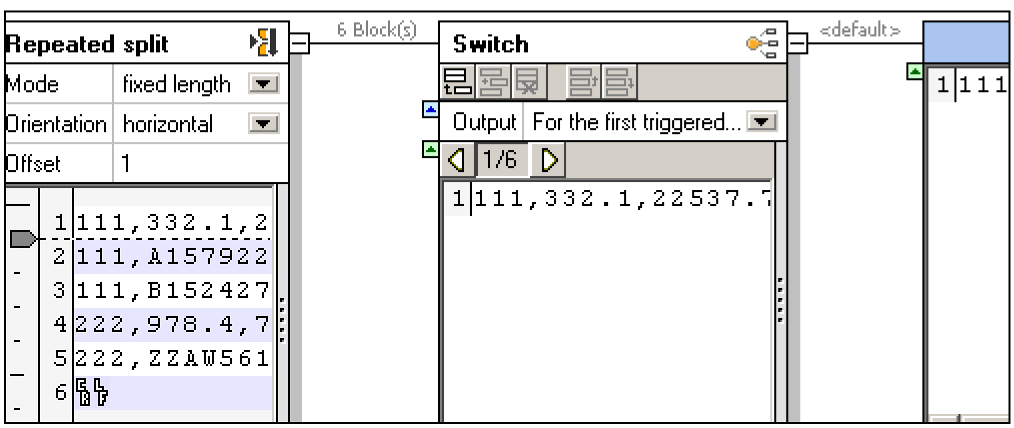
1. Click the top container icon and change it to **Repeated split**.



The default settings are what we need at this point. The text fragment is split into multiple text blocks of a single line each. The associated container shows a preview of each of the text blocks.

Clicking the Next text block icon , allows you to cycle through all the text fragments, of which there are 6.

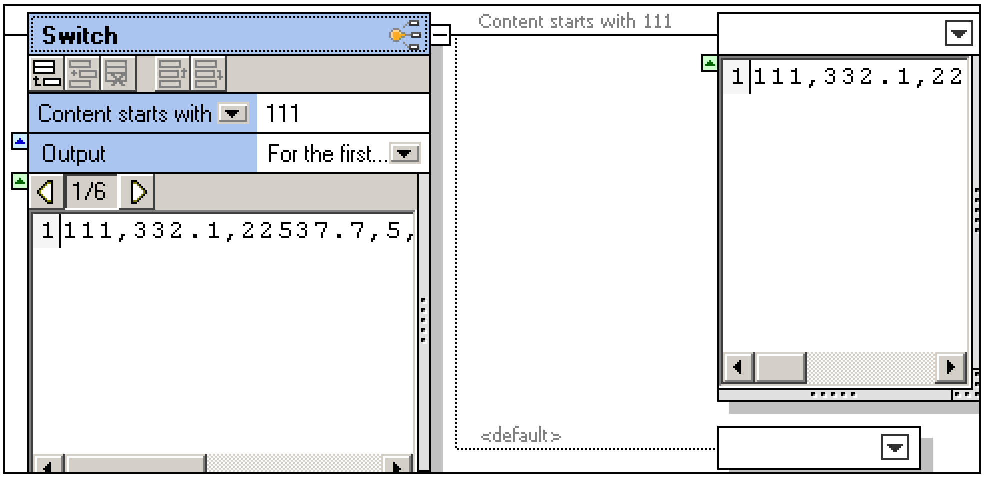
2. Click the new container and change it to **Switch**.



The initial state of the Switch container is shown above.

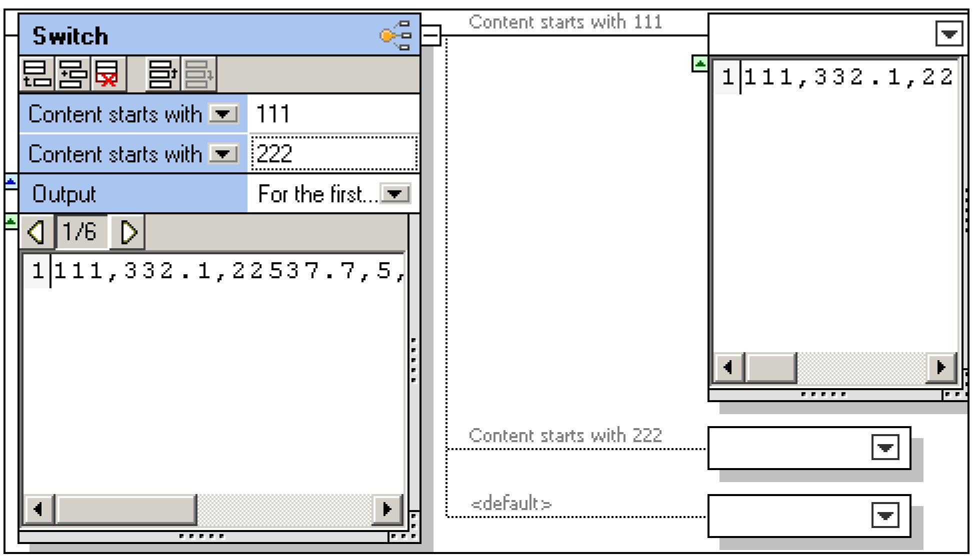
- An associated container "default", has been added.
- The content of the first record 1/6, is displayed in the default container.

3. Click the Append condition icon  in the "Switch" title bar, to add a new condition.
4. Double click in the field "Content starts with", and enter 111.



This defines the first condition. An associated container (Content starts with 111) has been added above the "default" container.

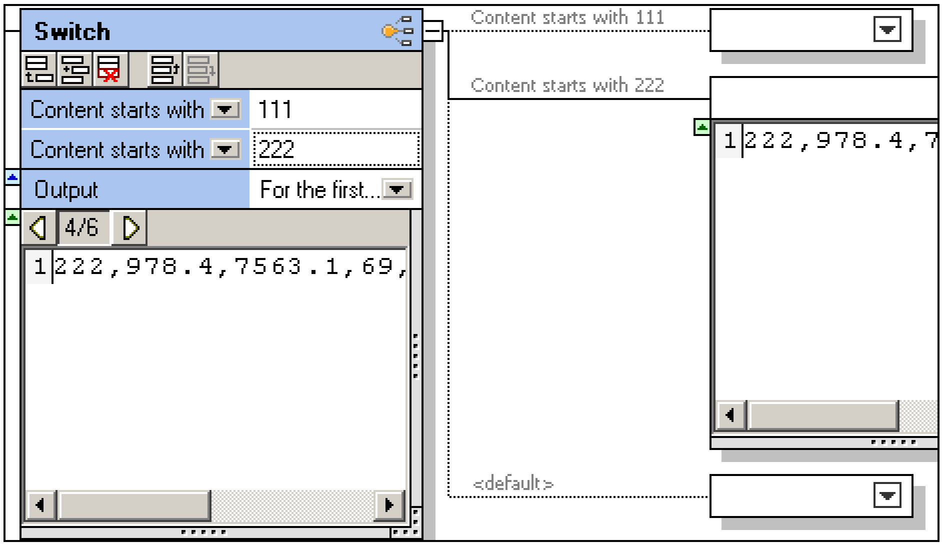
5. Click the append icon again, and enter "222" in the Content starts with field.



A third container has been added (Content starts with 222).

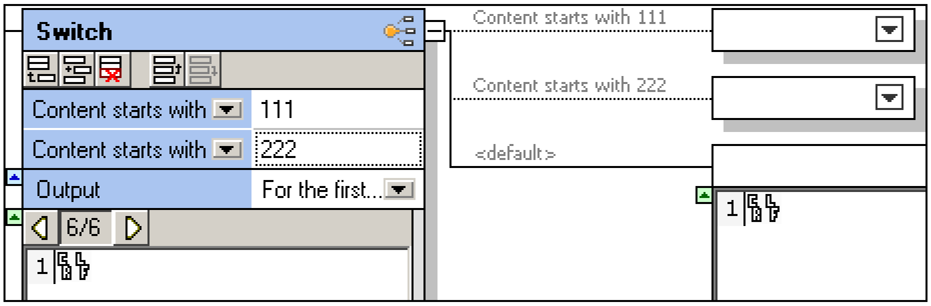
Note: Clicking the "Contents starts with" combo box, allows you to select the "Contains" option. This allows you to specify a "string" which can occur anywhere in the text fragment.

6. Click the Next text block icon , several times to see the effect.



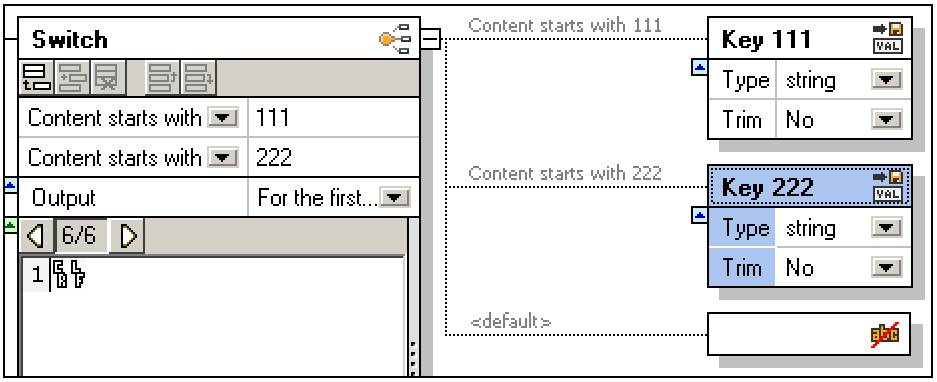
Upon reaching record 4 of 6, container 222 opens up, and displays its content.

7. Continue clicking, till you reach record 6 of 6. A single CR / LF character is displayed in the default container.



If a data fragment in the current block satisfies a condition, then the **complete data** of that block is passed on to the associated container. Data is not split up in any way, it is just routed to the associated container, or to the default container if it does not satisfy any of the defined conditions.

8. Click the first two containers and change them to **Store as value**. Click the last container and change it to **Ignore**.
9. Double-click the "Store" text, and add descriptive text e.g Key 111 and Key 222.

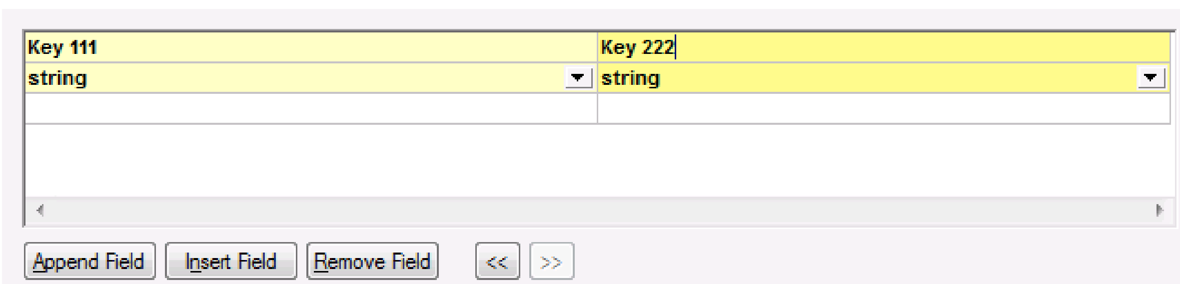


10. Save the FlexText template, e.g. Flex-Tutorial.mft.

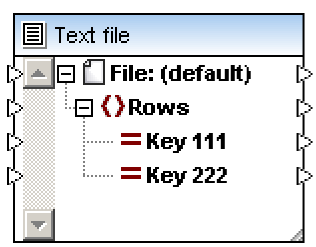
4.6.2.4 Step 4: Create the Target MapForce Component

To create the target component for use with the FlexText source:

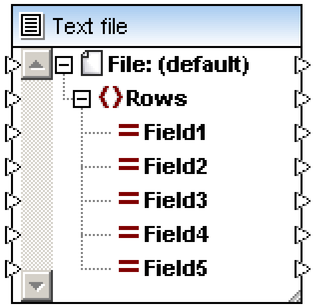
1. Select **Insert | Text file**, or click the Insert Text file icon .
2. Click the "**Use simple processing...**" radio button and click **Continue**. This opens the Component Settings dialog box.
3. Click the **Append Field** button to add a new field.
4. Double click the Field1 field name and change it to **Key 111**.



5. Do the same for the other field, and name it **Key 222** and click OK to confirm. A text component with two fields has now been created.



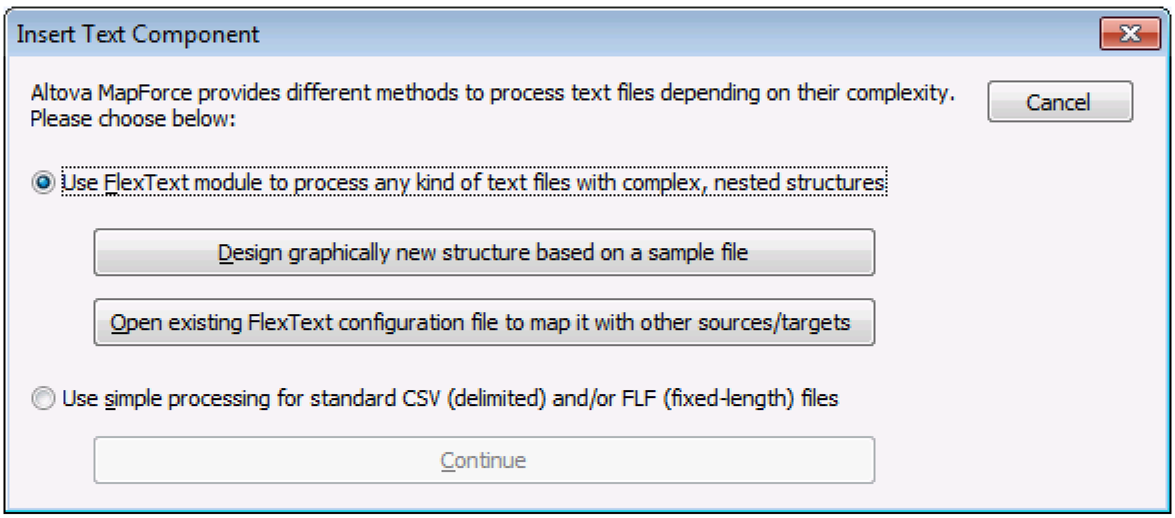
6. Use the same method to create a second text component that consists of five fields.



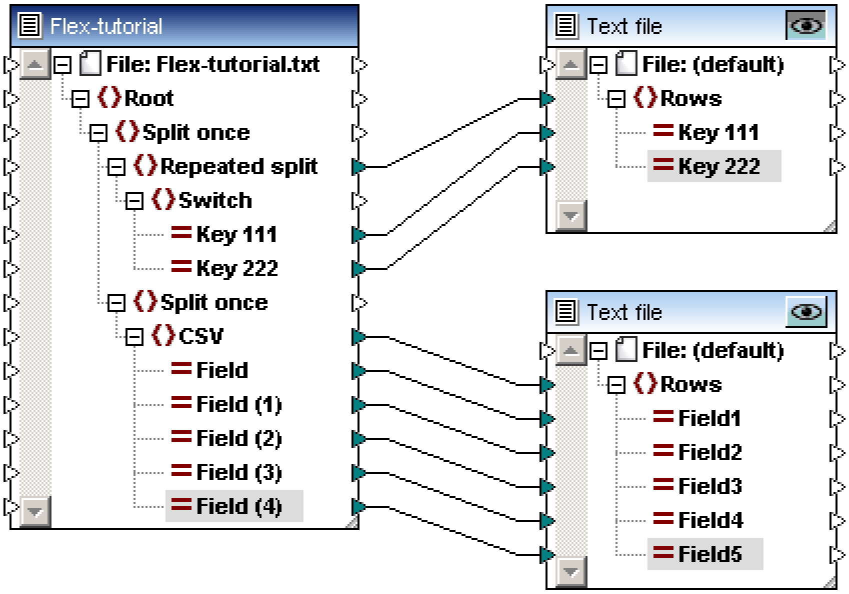
4.6.2.5 Step 5: Use the FlexText Template in MapForce

To use the FlexText template in MapForce:

1. Start, or switch back to MapForce, and select **Insert | Text file**.



2. Click the **Open existing FlexText configuration file...** button and select the previously defined FlexText template (**Flex-tutorial.mft**). The structure of the MapForce component mirrors that of the containers in Design view in FlexText.



3. Map the various items to the previously defined target components, and click the Output tab to preview the results.

Mapping preview of the top text component:

```
"111,332.1,22537.7,5,Container ship,Mega,
",
"111,A1579227,10,3,400,Microtome,
",
"111,B152427,7,6,1200,Miscellaneous,
",
,"222,978.4,7563.1,69,Air freight,Mini,
"
,"222,ZZAW561,10,5,10000,Gas Chromatograph,,
"

```

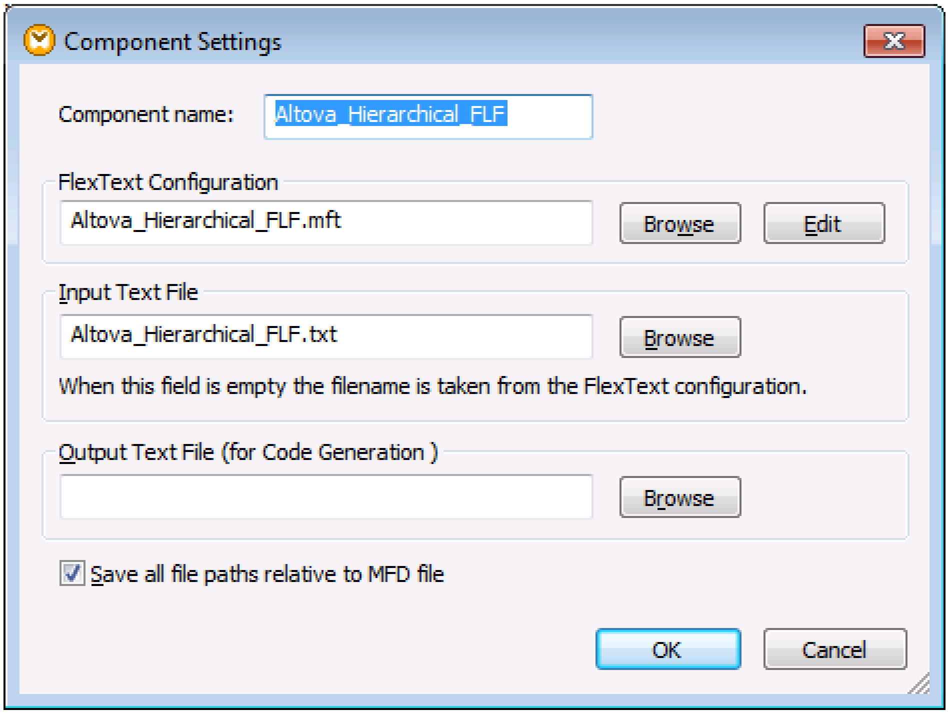
Mapping preview of the lower text component:

```
1100,897,22.1,716235,LOX
1110,9832,22991.30,002,NOX
1120,1213,33.01,008,SOX

```

4.6.3 FlexText Component Settings

After you add a FlexText component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- On the **Component** menu, click **Properties** (this menu item becomes enabled when you select a component).
- Double-click the component header.
- Right-click the component header, and then click **Properties**.



FlexText Component Settings dialog box

The available settings are as follows.

Component name	The component name is automatically generated when you create the component. You can however change the name at any time. If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well. The component name can contain spaces and full stop characters. The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications when changing the name of the component: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have different encodings in Windows and at the command line.
FlexText Configuration	Specifies the name or path of the FlexText template (.mft) used by the component.

	To select a new template, click **Browse**. To edit the template in FlexText, click **Edit**.
Input Text File	Specifies the text file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The source text file specified here takes precedence over the one defined in the FlexText template. If this field is empty, the filename defined in the FlexText template is used as source.
Output Text File (for Code Generation)	Specifies the name or path of the text file to which MapForce will write data. This field is meaningful for a target component when generating code. Entering a full path allows you to specifically define the target directory, for example, **c:\myfiles\sequence.txt**.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file.

4.6.4 Using FlexText as a Target Component

The main use of FlexText components is to reduce, or split off sections of text files and map the relevant items of the FlexText component to other target components. FlexText can, however, also be used as target component to assemble or reconstitute separate files into a single file, although this can entail a fair amount of trial and error to achieve the desired results.

Using a FlexText component as a **target** reverses the operations previously defined in it. Instead of splitting a file into various subsections, you assemble/reconstitute a file.

In general, the inverse of each operation defined in the FlexText template is carried out (in a bottom-up fashion) when using it as a target:

- "Split" becomes "merge", e.g. mapping to a repeated split delimited by "," becomes a merge items separated by ",".
- "Store" becomes "load".
- "Switch" becomes "choose the first match".

Note the following when using FlexText as target components:

- As soon as a connection is made between a data source component and one of the input items of a FlexText component, the FlexText component data source is ignored. The data provided by the newly mapped source component now takes precedence.
- If text is mapped to a "Store as..." (Store as CSV and FLF) container, then the separator is retained. However, text might be truncated if a fixed length split occurs in a node above the "Store as..." node.
- Fixed Width Splits truncate the left/top section if Split Base=Head, or the right/bottom section if Split Base=tail, to the predefined length. The truncated section is then as long as the defined length in characters. If the text is too short, then space characters are inserted to pad the section.
- FlexText would normally insert separators (or white space for fixed splits) between the items of a split operation, but this is not the case for 'delimited (line based)' splits. The 'delimited (line based)'

operation is not a perfectly reversible operation. The "Delimited" text may occur anywhere in the first line and is included in the text, and therefore an automatic process cannot reliably add it.
  - o Delimited (line based), will not add a separator to the first line if it is missing.
  - o Delimited (floating), will add a separator between two sections.
- The switch operation cannot be inverted in a meaningful way except for simple cases. The switch scans its branches for the first branch that contains data, and uses/inserts this data. Only the first connection of a switch operation is mapped. To transfer data to the remaining switch containers, filters have to be defined for the remaining connectors and the duplication of the switch parent item is necessary, so that each switch item returns a single item which is then fed to a repeated split item to merge all of them.
- Mapping to a child of a single split container discards all mapping results except for the last item. Only a single result is retained, even if multiple results were generated.

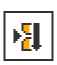
The following analogy to the XML Schema content model gives some idea of FlexText's behavior when used as a target:

- A repeated split is a repeatable element.
- A single split forms part of a 'sequence' content model group.
- A switch forms a 'choice' content model group, each case being a possible child element.
- A store creates an element of simple type.

4.6.5 FlexText Reference

The reference section describes the various features of FlexText and shows how to use them to achieve specific results.

4.6.5.1 Repeated split

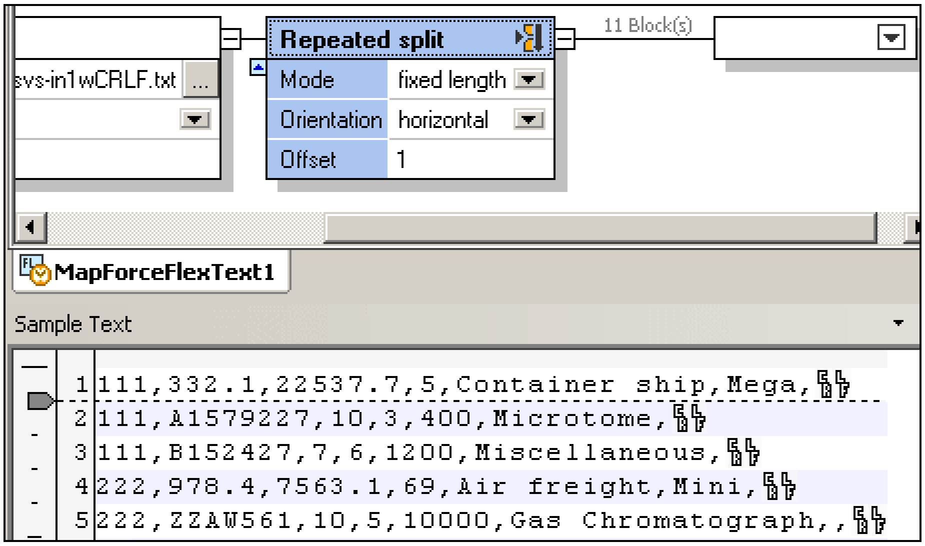


Using this option initially creates a single container. The container contains the text defined by the condition set in **Repeated Split**. There are several versions of the Repeated split option: Fixed [677] length, Delimited Floating [678], Delimited Line based [680], and Delimited Starts with... [682].

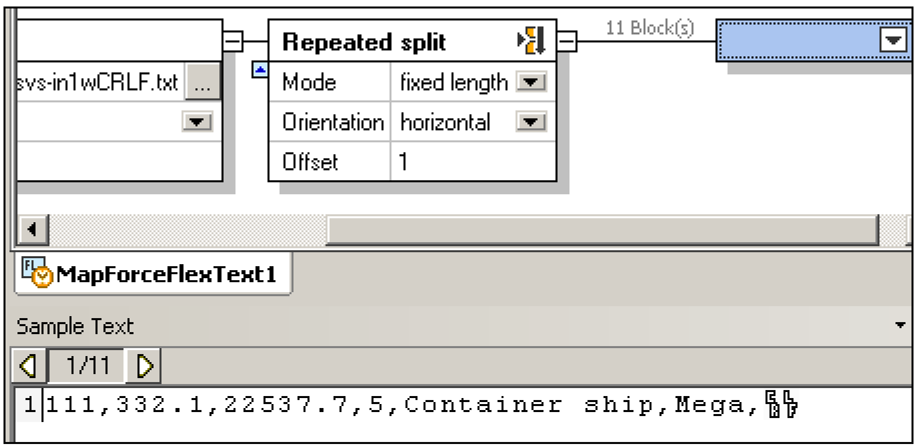
When you first select this option, default parameters are automatically set and the resultant fragments appear in the associated container. Note that the Repeated Split container is currently active, and the preview displays all current records/lines, in the Sample Text pane.

Container **default settings**:

Mode	fixed length
Orientation	horizontal
Offset	1



Default result:
Each line of text appears as a line/record in the new container, as the Offset is 1. Click the new container to preview its contents. The Sample Text **scroll arrows**, let you scroll through each of the 11 blocks/fragments produced by these settings.



4.6.5.1.1 Mode - Fixed length

Use the **Repeated split (fixed length)** mode when you want to split text into multiple horizontal or vertical fragments of fixed length. The settings applicable to the **Repeated split (fixed length)** mode are described below.

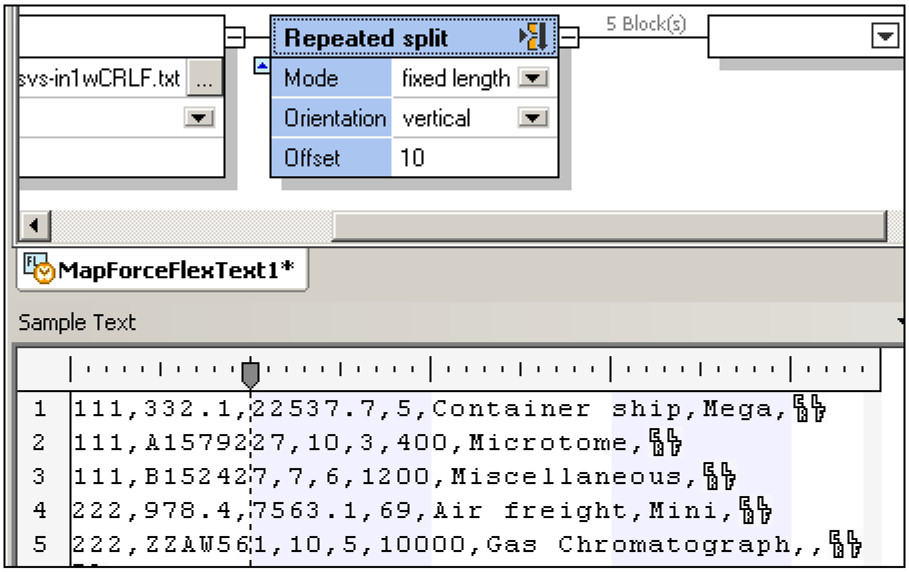
Orientation

Specifies if the text fragment is to be split horizontally or vertically. Choose "Horizontal" to split the fragment into multiple horizontal sections. Choose "Vertical" to split the fragment into multiple vertical columns. The default orientation is "Horizontal".

Offset

If orientation is set to "Horizontal", this setting specifies the number of lines that each fragment should contain. If orientation is set to "Vertical", this setting specifies the width in characters of each fragment. The default offset is "1". To modify the offset, do one of the following:

- Enter a value into the Offset field
- Drag the tab on the vertical or horizontal ruler.



4.6.5.1.2 Mode - Delimited (floating)

Use the **Repeated split with delimited (floating)** mode in the following situations:

- To split text where the separator characters that you specify must be stripped out from the resulting fragments
- To split text where the separators are in-line (for example, text that doesn't contain CR/LF characters)

Note: A fragment is defined as the text between the first character after the separator, up to the last character before the next instance of the same separator. An exception to this rule are the first and last fragments, as shown in the example below.

The settings applicable to the **Repeated split with delimited (floating)** mode are described below.

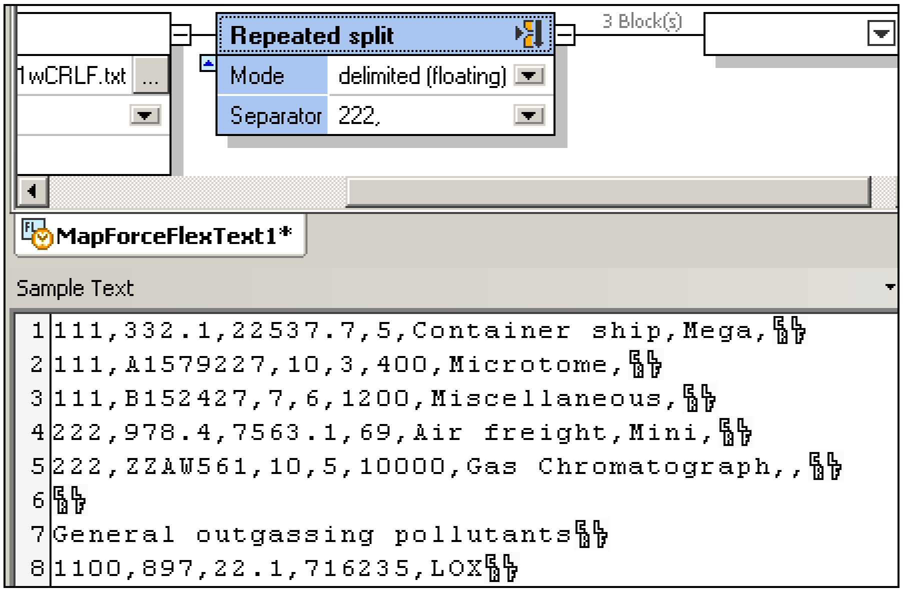
Regular expression

This is an optional setting which splits text into fragments whenever there is a regular expression match (see Splitting Text with Regular Expressions 708). The default value is "no".

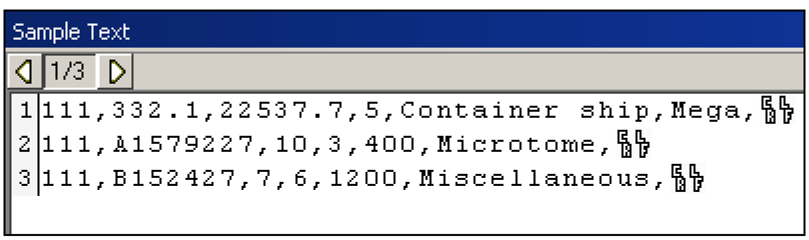
Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, using the separator "**222,**" against the text shown below produces three separate fragments.



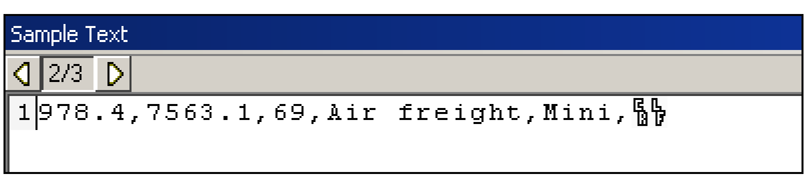
The first fragment contains all characters from the start of the fragment to the start of the first separator ("222,"), that is, from "111" to "Miscellaneous,".



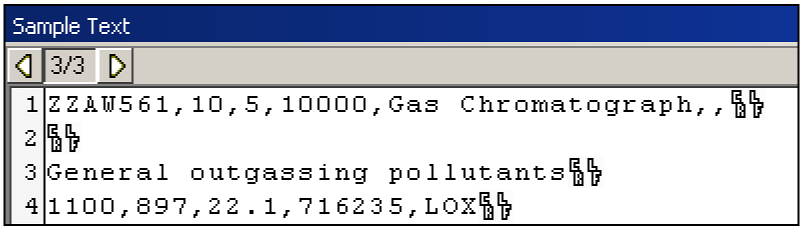
If the separator is not the first set of characters of the first line in the fragment, as in this example, then the first fragment includes all the text up to the first instance of the separator (for example, "222").

If "111" were the separator, then the first fragment would be a zero-length string, as the separator appears at the beginning of the first line of the source fragment.

The second fragment contains the first line containing the separator 222, without the separator.



The third fragment contains the next line containing the separator 222, without the separator itself, up to the end of the text file/fragment.



4.6.5.1.3 Mode - Delimited (line based)

Use the **Repeated split with delimited (line based)** mode to split text into multiple fragments, with the following behaviour:

- This mode creates multiple fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- A fragment is defined as the entire line containing the separator, up to the next line containing the same separator.
- If the separator does not appear in the first line, then the first fragment contains the line(s) up to the first line containing the separator.

The settings applicable to the **Repeated split with delimited (line based)** mode are listed below.

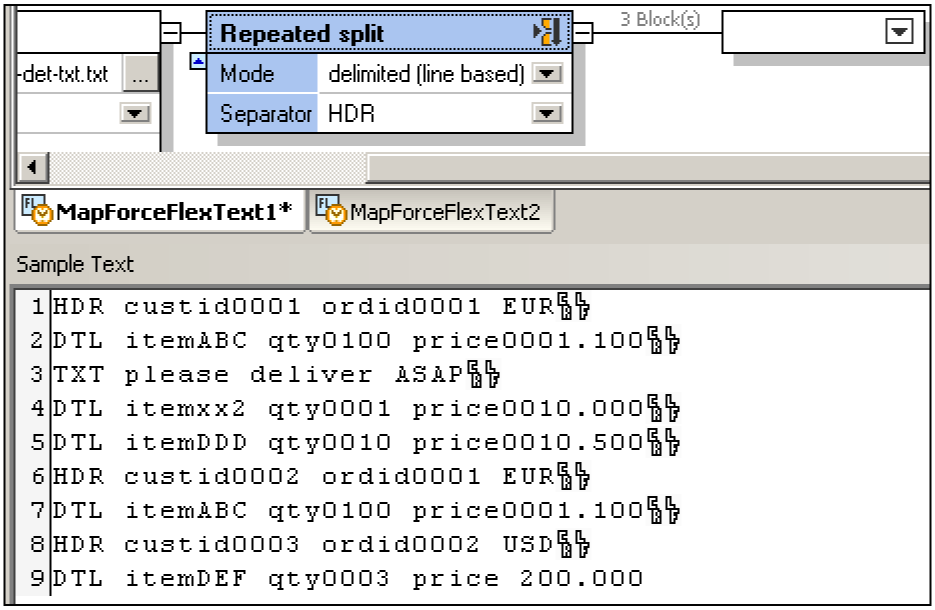
Regular expression

This is an optional setting which splits text into fragments whenever there is a regular expression match (see Splitting Text with Regular Expressions 708). The default value is "no".

Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, using the separator "**HDR**" against the text shown below produces three separate fragments.



The first fragment contains all characters from the start of the file/fragment, including all lines up to the next line containing the same separator.

```
HDR custid0001 ordid0001 EUR
DTL itemABC qty0100 price0001.100
TXT please deliver ASAP
DTL itemxx2 qty0001 price0010.000
DTL itemDDD qty0010 price0010.500
```

Note that this option allows you access to any number of lines between two separators. This is useful in files with record types that are optional or not in sequence (such as "DTL" or "TXT" in this example).

The second fragment contains all text from the second occurrence of "HDR" up to the next occurrence of "HDR".

```
HDR custid0002 ordid0001 EUR
DTL itemABC qty0100 price0001.100
```

The third fragment contains all text from the third occurrence of "HDR" up to the end.

```
HDR custid0003 ordid0002 USD
DTL itemDEF qty0003 price 200.000
```

4.6.5.1.4 Mode - Delimited (line starts with)

Use the **Repeated split with delimited (line starts with)** mode to split text into multiple fragments, with the following behaviour:

- This mode creates multiple fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- A fragment is defined as the entire line, starting with the separator, up to the next line containing the same separator at the start of the line.
- If the separator does not appear in the first line, then the first fragment contains the line(s) up to the first line containing the separator.

The settings applicable to the **Repeated split with delimited (line starts with)** mode are listed below.

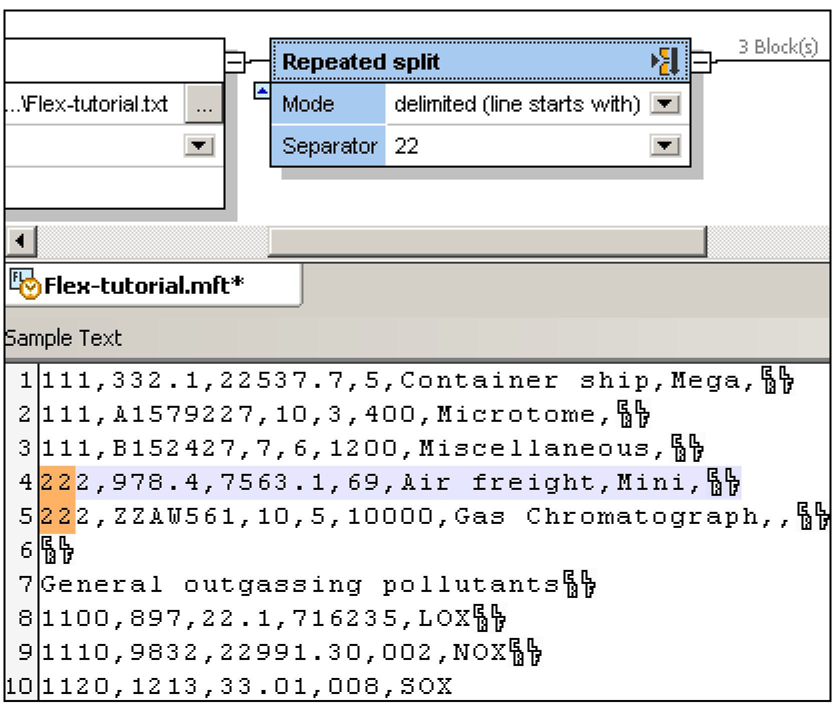
Regular expression

This is an optional setting which splits text into fragments whenever there is a regular expression match (see Splitting Text with Regular Expressions 708). The default value is "no".

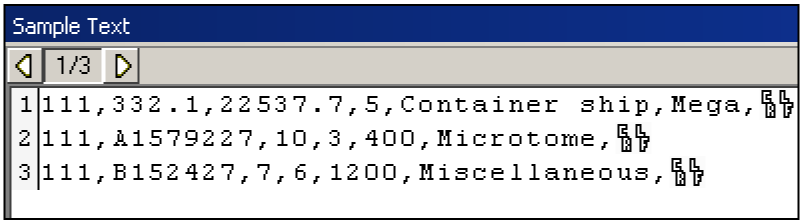
Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

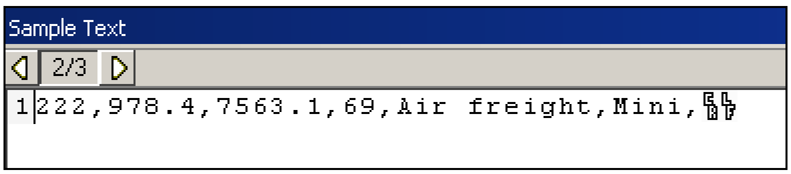
For example, using the separator "**22**" against the text below produces three separate fragments:



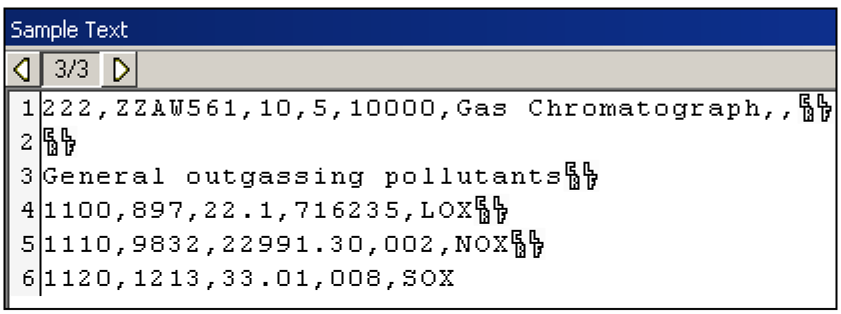
The first fragment contains all characters from the start of the file/fragment, including all lines up to the line containing the separator "22".



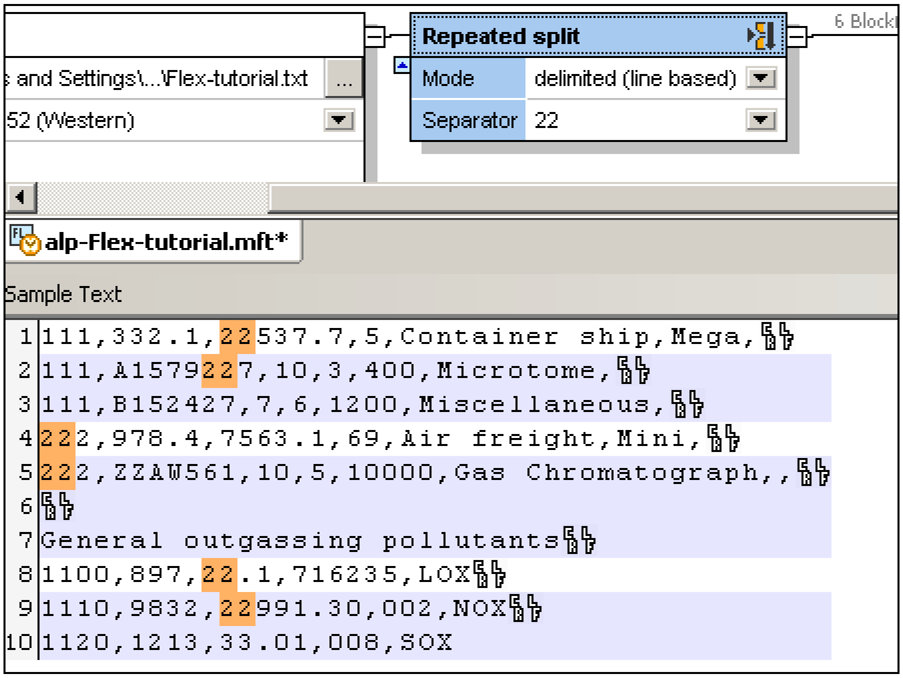
The second fragment contains all characters/lines from the first occurrence of "22", up to the next occurrence of "22", which in this case is only one line.



The third fragment contains all characters/lines from the second occurrence of "22", up to the end.



By contrast, here is what would happen if we used the **delimited (line based)** mode and separator as "22":



There would be six fragments, composed of lines that contained 22 anywhere in that line.

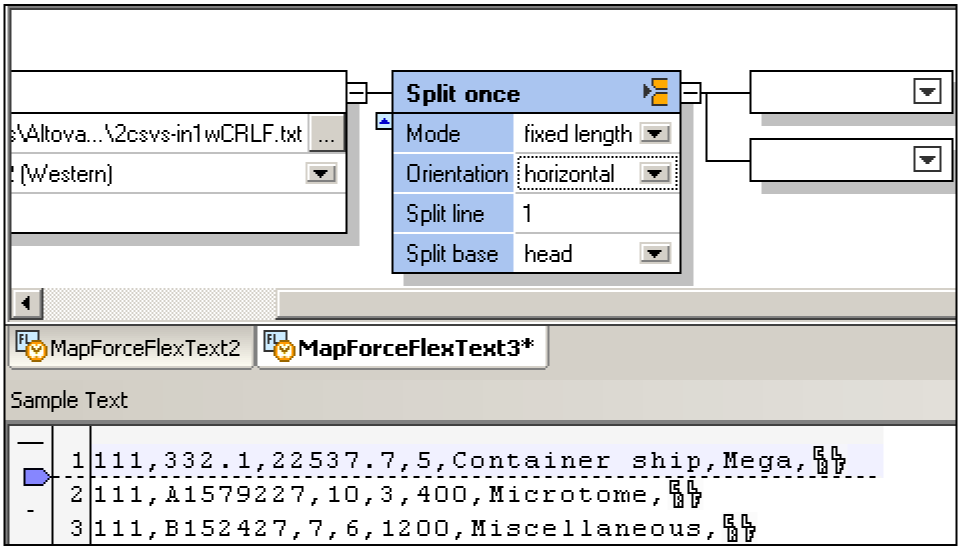
4.6.5.2 Split once

Using this option creates two vertically aligned containers. The top container contains the text defined by the condition set in the **Split once** container. The bottom container contains the rest of the text file/fragment. There are several versions of the Split once option: Fixed 685 length, Delimited Floating 686, and Delimited Line Based 687.

When you first select this option, default parameters are automatically set, and the resultant fragments appears in both containers. Note that the **Split once** container is currently active, and displays a preview of all current records/lines, in the Sample Text pane.

Container **default settings** are:

Mode	fixed length
Orientation	horizontal
Split line	1
Split base	head



Default result:
The first line of text appears in the top container. The bottom container contains the rest of the text file/fragment.

4.6.5.2.1 Mode - Fixed length

Use the **Split once (fixed length)** mode when you want to split text into two horizontal or vertical fragments, at a particular line or column relative to the beginning or end of the text. The settings applicable to the **Split once (fixed length)** mode are described below.

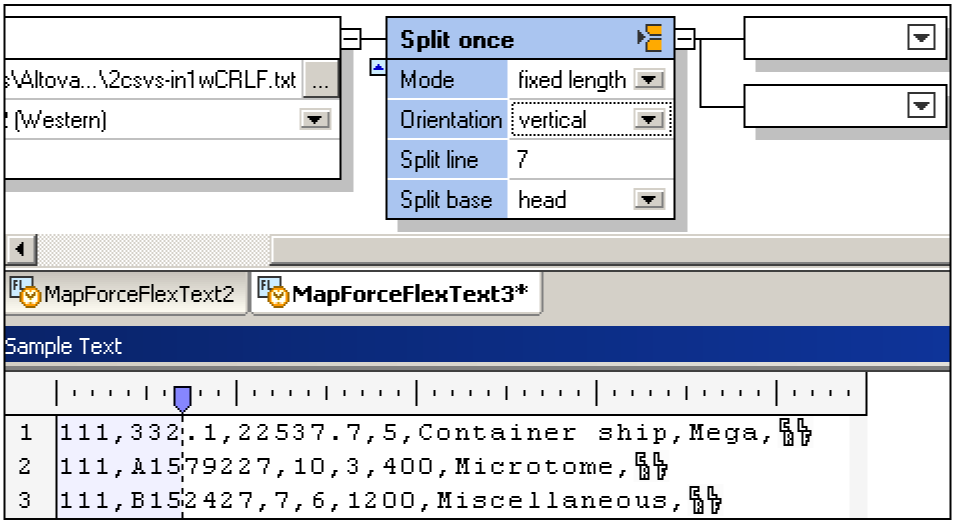
Orientation

Specifies if the text fragment is to be split horizontally or vertically. Choose "Horizontal" to split the fragment into two horizontal sections. Choose "Vertical" to split the fragment into two vertical sections. The default orientation is "Horizontal".

Split Line

Specifies the number of lines (or columns) after which the fragment should be divided into two. The default offset is "1". To modify this setting, do one of the following:

- Enter a value into the **Split line** field
- Drag the tab on the vertical or horizontal ruler.



Split base

Head	Look for the first occurrence of the separator starting from the beginning of text.
Tail	Look for the first occurrence of the separator starting from the end of text.

4.6.5.2.2 Mode - Delimited (floating)

Use the **Split once with delimited (floating)** mode to split text into two fragments, using a custom separator that is anywhere in the text. This is generally useful in files that do not contain CR, or LF characters, and you want to split the fragment into two, at some specific in-line location. Note the following:

- This mode creates two fragments defined by separator characters that you enter in the Separator field.
- The separator characters are not included in the fragment.
- The first fragment is defined as the text between the first character of the file/fragment, up to the last character before the separator.
- The second fragment is defined as the first character after the separator, up to the last character in the file/fragment.
- If the separator appears in the first/last position of the file/fragment, then the first of the two resulting containers remains empty.

The settings applicable to the **Split once with delimited (floating)** mode are listed below.

Regular expression

This is an optional setting which splits text into two fragments when there is a regular expression match (see Splitting Text with Regular Expressions 708). The default value is "no".

Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

Split base

Head	Look for the first occurrence of the separator starting from the beginning of text.
Tail	Look for the first occurrence of the separator starting from the end of text.

4.6.5.2.3 Mode - Delimited (line based)

Use the **Split once with delimited (line based)** mode to split text into two fragments, where the separator is anywhere in one of the lines. The line containing the separator is not split, but is retained whole. This is generally useful in files containing record delimiters (CR/LF), and you want to split the fragment into two separate fragments. Note the following:

- This mode creates two fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- The first fragment is defined as all the text up to the line containing the separator.
- The second fragment is defined as the text, and line, including the separator up to the end of the file/fragment.
- If the separator appears in the first/last line, of the file/fragment, then the top container remains empty.

The settings applicable to the **Split once with delimited (line based)** mode are listed below.

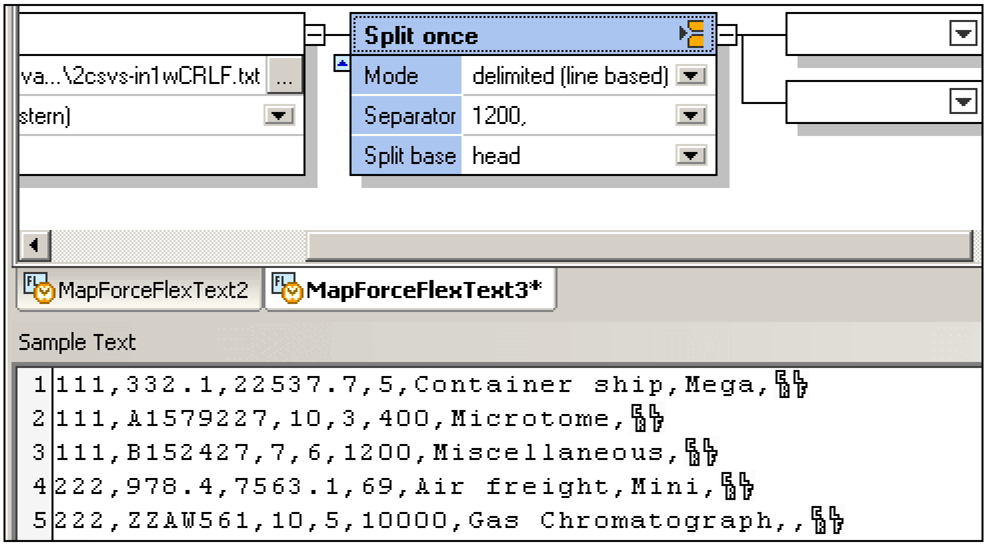
Regular expression

This is an optional setting which splits text into two fragments when there is a regular expression match (see Splitting Text with Regular Expressions 708). The default value is "no".

Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, if you use the separator "1200," against the text below, two fragments are created.



The first fragment contains the text up to the line containing the separator.

```
1 111,332.1,22537.7,5,Container ship,Mega,
2 111,A1579227,10,3,400,Microtome,
```

The second fragment contains the entire line containing the separator, and all remaining lines up to the end.

```
1 111,B152427,7,6,1200,Miscellaneous,
2 222,978.4,7563.1,69,Air freight,Mini,
3 222,ZZAW561,10,5,10000,Gas Chromatograph,,
```

Split base

Head	Look for the first occurrence of the separator starting from the beginning of text.
Tail	Look for the first occurrence of the separator starting from the end of text.

4.6.5.2.4 Mode - Delimited (line starts with)

Use the **Split once with delimited (line starts with)** mode if the split should occur at the first line which begins with the specified separator. When you select this mode, two fragments of text are created, as follows:

- The first fragment contains all the text up to the line where the separator is.
- The second fragment contains the remaining text, including the line where the separator is.
- If the separator is at the beginning of the first line, no split occurs (that is, the first resulting fragment remains empty).

The settings applicable to the are listed below.

Regular expression

This is an optional setting which splits text into two fragments when there is a regular expression match (see [Splitting Text with Regular Expressions](708)). The default value is "no".

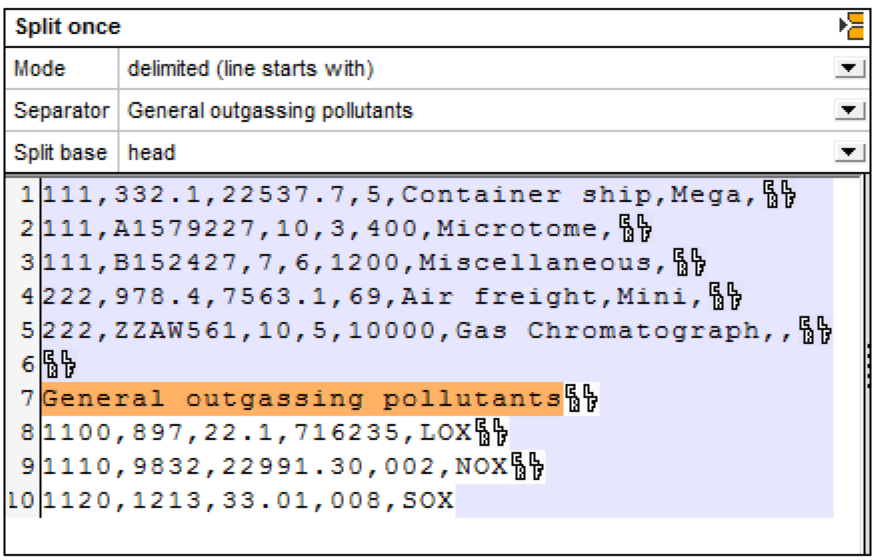
Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

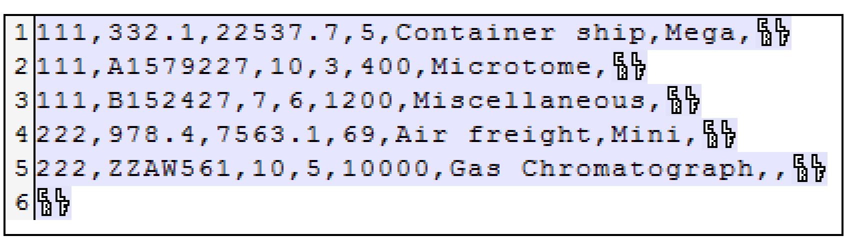
Split base

Head	Look for the first occurrence of the separator starting from the beginning of text.
Tail	Look for the first occurrence of the separator starting from the end of text.

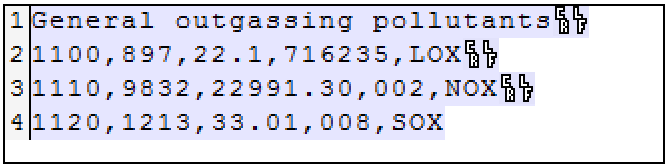
For example, in the image below, the split occurs at the first line which begins with the specified separator ("General outgassing pollutants"). If the Split base option was set to **tail**, then the split would have occurred at the first line which begins with the specified separator, starting from the end (which in this example happens to be the same line).



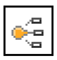
The first resulting fragment is as follows:



The second resulting fragment is as follows:



4.6.5.3 Switch

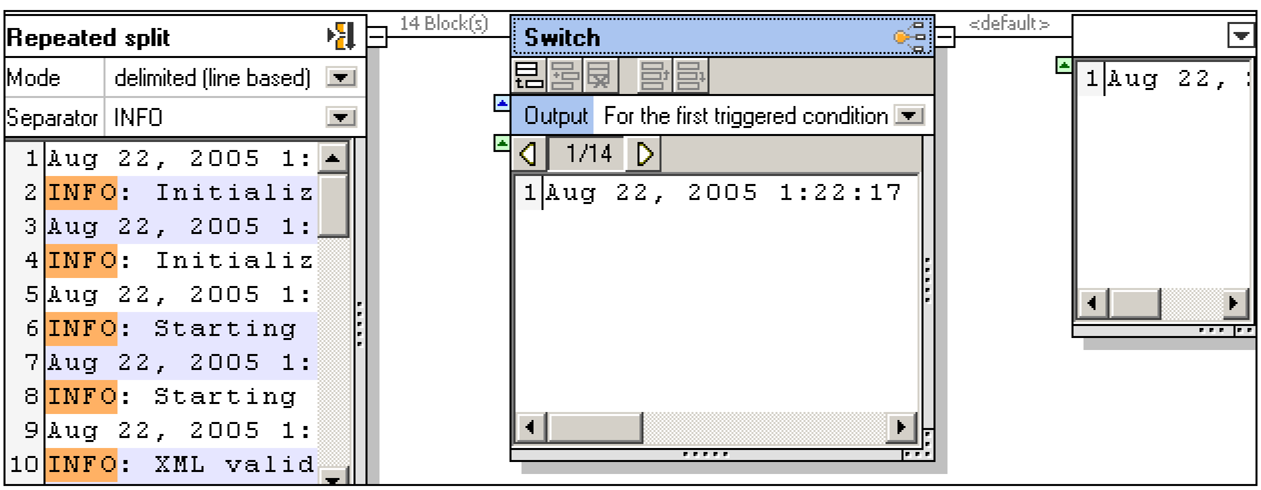


Using the Switch option allows you to define multiple keywords, or conditions, for a single text fragment. Each keyword you define, has its own associated container which receives data only if the specific condition is satisfied, i.e. true. If none of the conditions are satisfied, then the specific fragment is mapped to a "default" container.

Container **default settings** are:

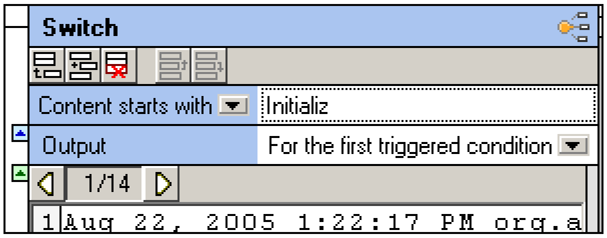
Output For the first triggered condition.

The example below processes a Tomcat log file, where the individual processes are to be separated out, and made mappable. When you first define a Switch container, only the **default** container appears to the right of the Switch container. All data is automatically passed on to it.



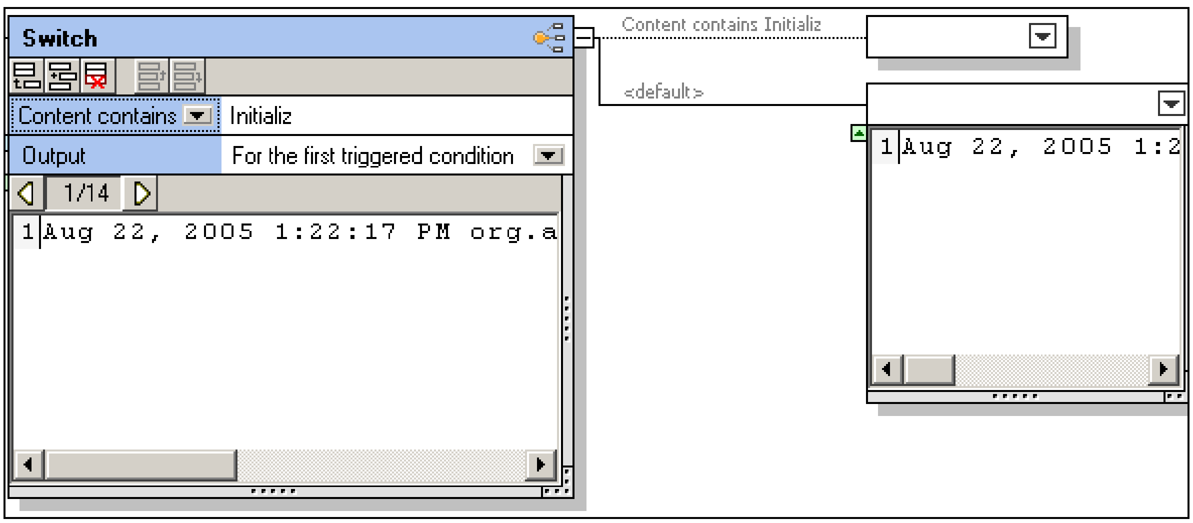
The repeated split container, using delimited (line based), separates all INFO sections out of the log file and passes them on to the Switch container.

1. Click the append icon  to add a new condition to the Switch container.
2. Double click in the "**Content starts with**" field, enter "**Initializ**" and hit Return.

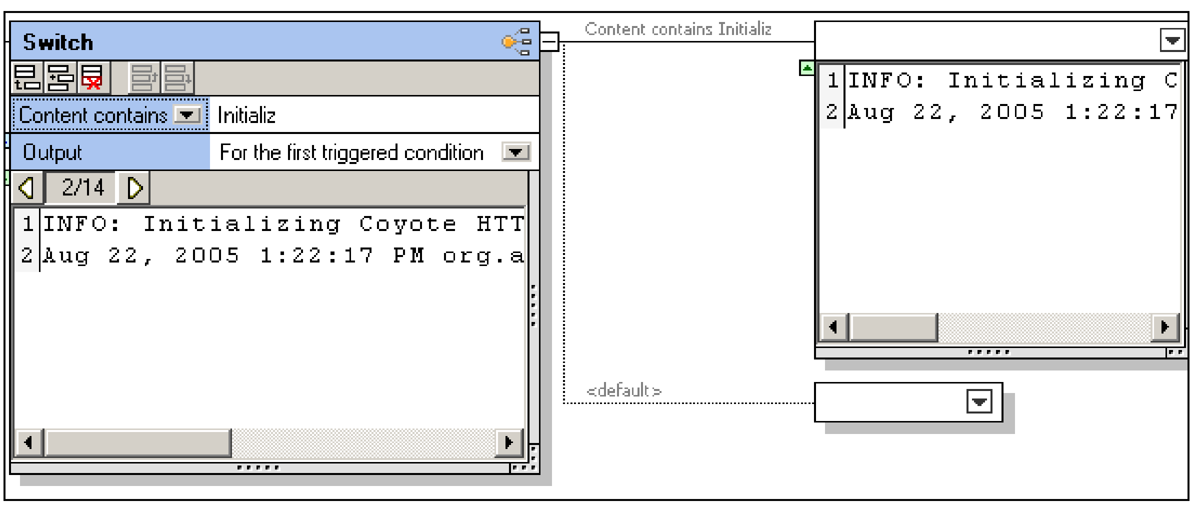


A new container is added. Data will be forwarded to this container if the condition is true. If not, the data is forwarded to the default container.

3. Click the "**Content starts with**" combo box, and change it to "Content contains". The first condition has now been defined and you can see the result below. The first fragment does not contain "Initializ", and its contents are therefore forwarded to the **default** container.

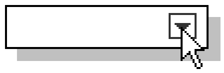


4. Click the Display next block icon , to see the next text fragment.

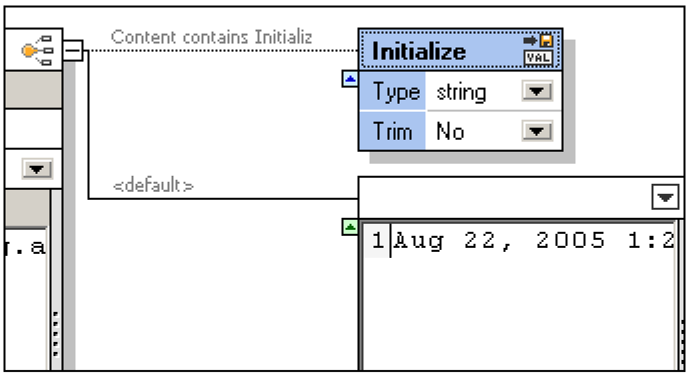


The Initializing... fragment now appears in its associated container, and the default container is empty. Stepping through the fragments gives you a preview of what the individual containers hold.

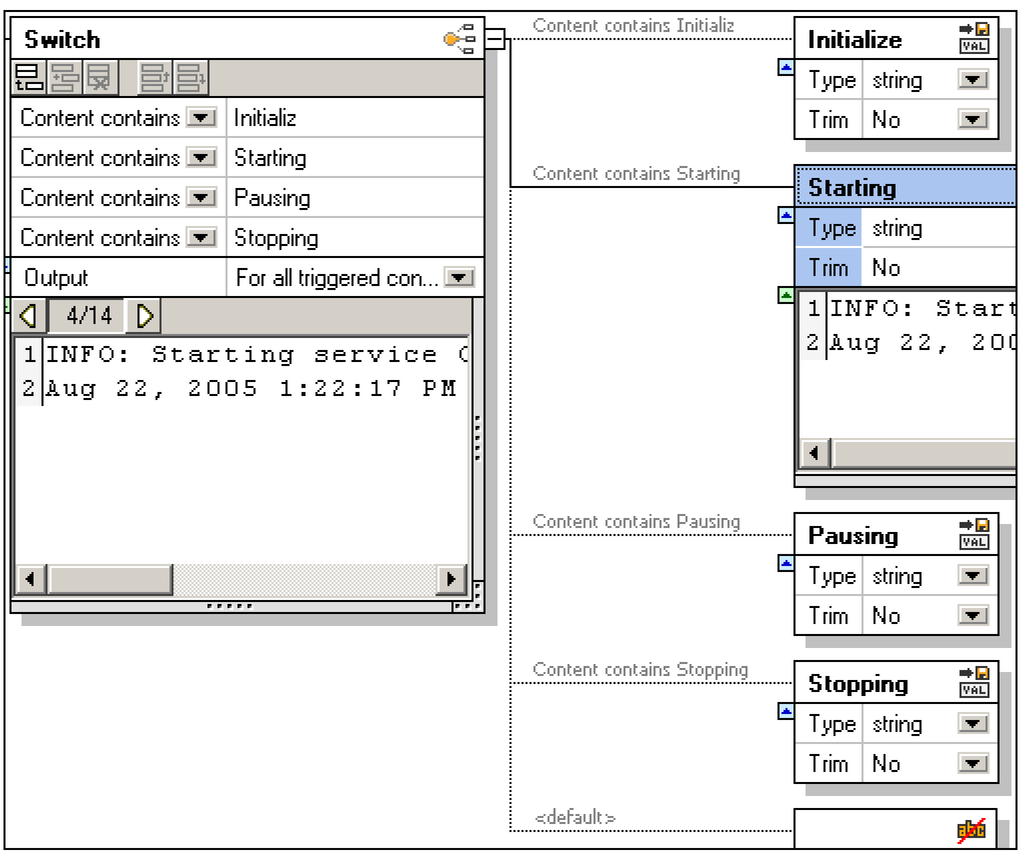
5. Click the container icon button, and select **Store as value**.



6. Double click in the "Store" title bar and change the text e.g. Initialize.



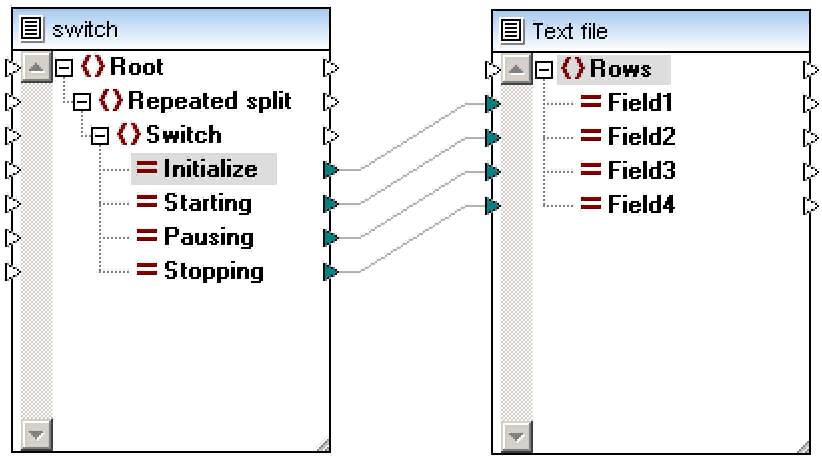
7. Click the append icon  to add a new condition to the Switch container.
8. Double click in the "**Content starts with**" field, enter "Starting" and hit Return. You can add as many conditions as you need e.g. Pausing, and Stopping. Give each of the associated containers a name, to make recognition in MapForce easier.



The screenshot above shows all four conditions, and the contents of the "Starting" container at block/fragment no 4. The associated containers have all been renamed to make identification in the MapForce component easier.

Note that conditions can be moved up and down in the condition list, using the respective Move Up/Down buttons , or .

9. Save the template and insert it in MapForce.



Note: If a text fragment in the current fragment satisfies a condition, then the **complete data** of that fragment is passed on to the associated container. Data is not split up in any way, it is just routed to the associated containers, or to the default container if it does not satisfy any of the defined conditions.

The associated containers produced by Switch, can be used for further processing. You can change such a container to Split once, Repeated split, or anything else if you wish.

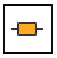
Content starts with:
Data is only passed to the associated container, if the condition string appears at the start of the text fragment.

Content contains:
Data is passed on to the associated container, if the condition string appears anywhere in the text fragment.

For the first triggered condition:
Data is passed on when **one** of the **conditions** in the condition list is **true**. Any other conditions that are true are ignored, and no data is passed on to any of the associated containers.

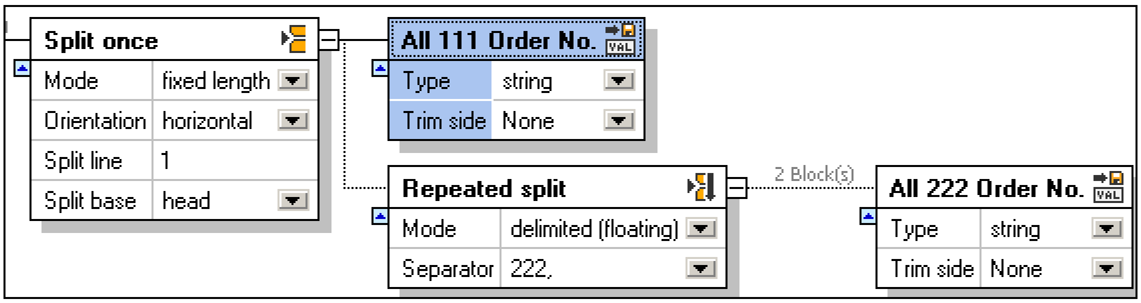
For all triggered conditions:
Outputs data for **every** condition that is true in the condition list. This makes it possible to have multiple occurrences of the same data/fragment in multiple associated containers simultaneously. This might occur if a text fragment contains text that satisfies two conditions simultaneously e.g. "initializing starting sequence" in the example above.

4.6.5.4 Node

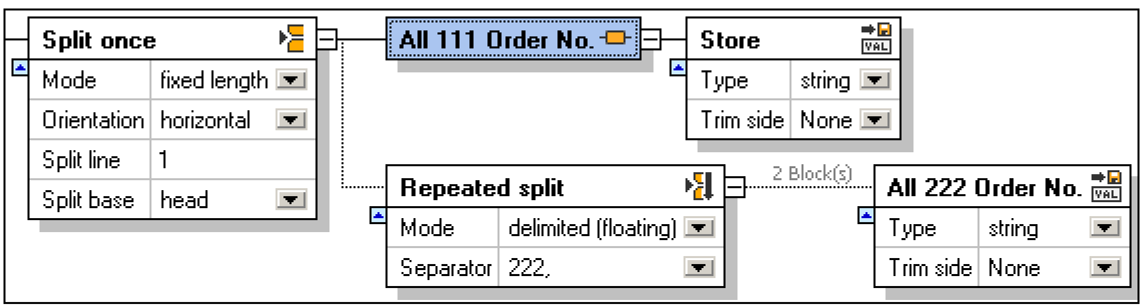


Allows you to add a new hierarchical level to the FlexText, and MapForce tree structures. The data that the following node/container contains, is passed on as is.

In the screenshot below, the "All 111 Order No." container is the last container in the top branch.



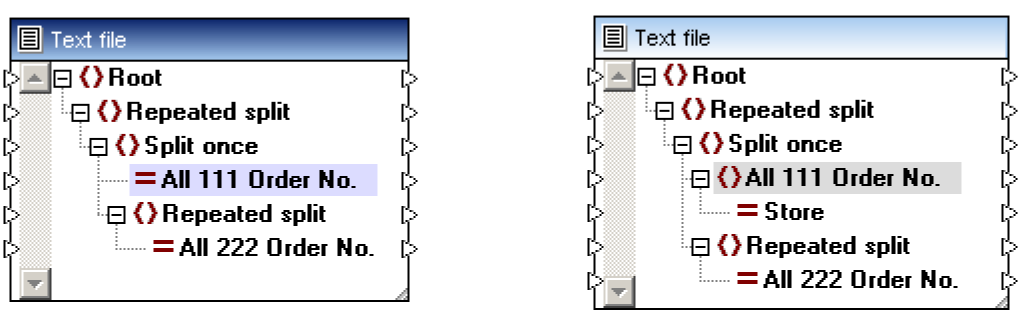
Click the top-right icon of the container, and select **Node** from the context menu.



A new container has been added to the right of the current one.

Note: The automatically appended container was then manually defined as "**Store as value**".

The screenshot below shows both template structures as they appear when inserted into MapForce.

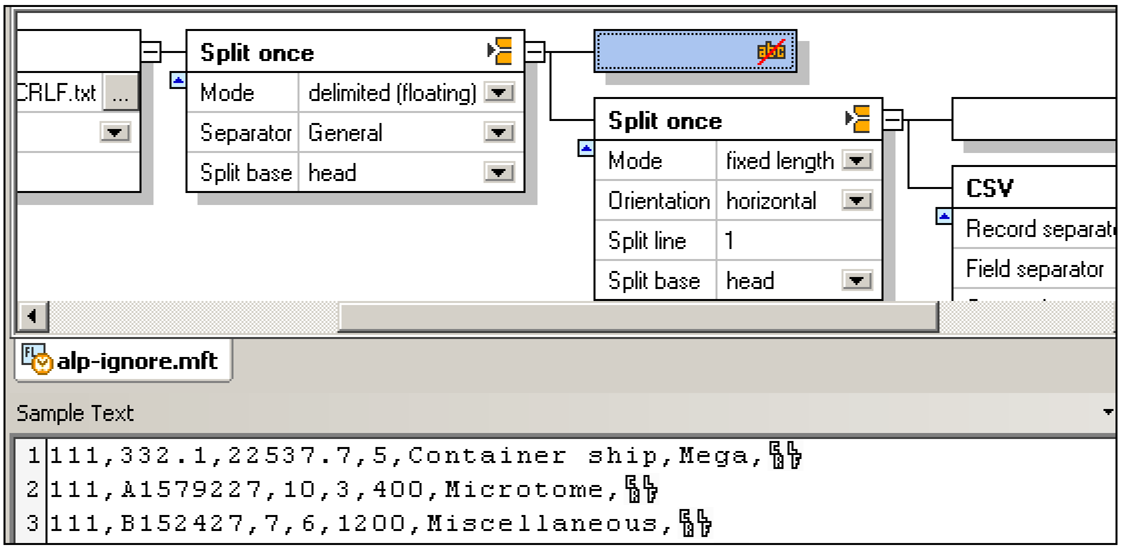


The left component shows the initial structure before adding the new Node.

The right component shows how the component structure has changed. "All 111..." is now a parent item, and a new child item "Store" has been added below it.

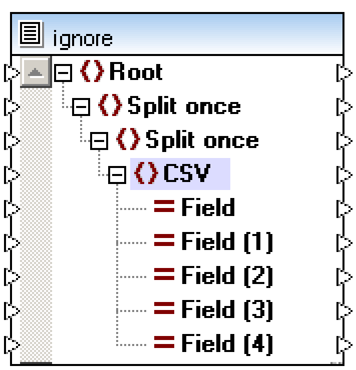
4.6.5.5 Ignore

Allows you to suppress the output of a specific text fragment. What this means, is that the container and any data it may contain, will not be made available as a mappable item in the FlexText component in MapForce.

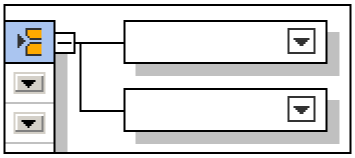


In the example shown above, the active container has been set to "Ignore". The Sample text that it contains will therefore not appear as a mappable item in MapForce.

The text template when inserted into MapForce, has the structure shown below. There is no mappable item between the two "Split once" items.



Note: Default "ignore" containers also exist. These are the new containers that are automatically appended when selecting "Split once" and "Repeated split" etc.



The contents of these containers are not initially mappable/available to MapForce when the template is inserted. You have to select one of the container options in FlexText: Store as value, Store as CSV etc., to be able to map them.

4.6.5.6 Store as CSV (delimited)

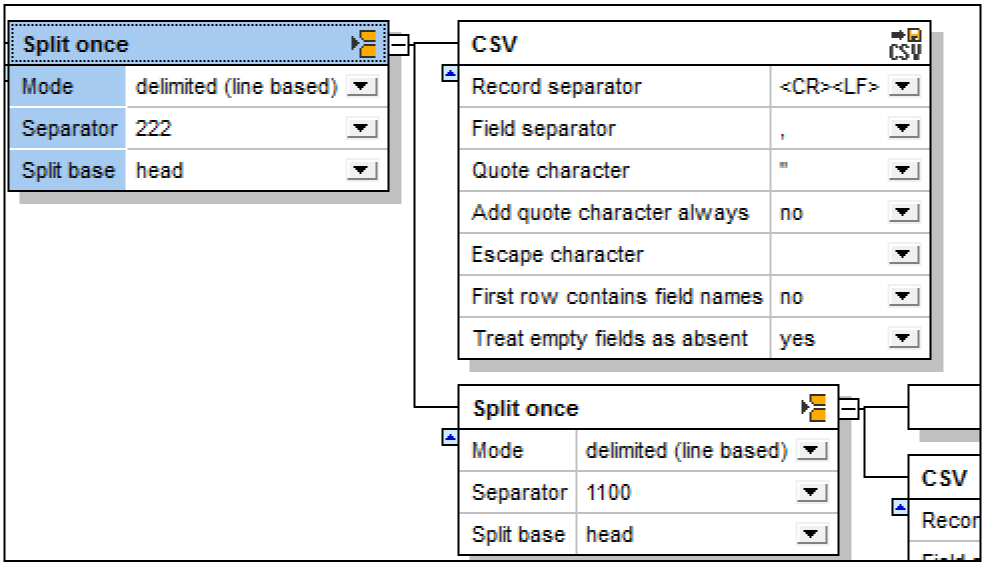
Store CSV allows you to store text fragments as CSV text, and map individual columns to MapForce. Any number of CSV containers/components can be created in FlexText, and each of the CSV containers may have different separators.

The Sample Text pane provides an overview of the current CSV fragment, and also allows you to specify individual field names, and field types. Each column appears as a mappable item in the FlexText component in MapForce.

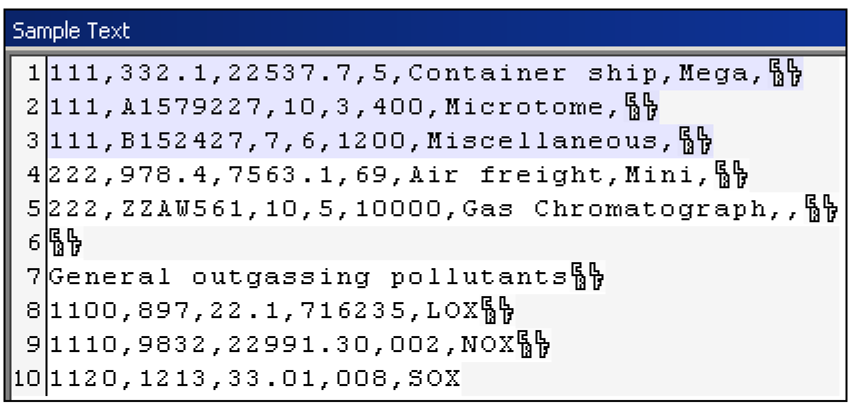
Container **default settings** are:

Record separator	CR LF
Field separator	,
Quote character	"
Add quote character always	no
Escape character	(none)
First row contains field names	no
Treat empty fields as absent	yes

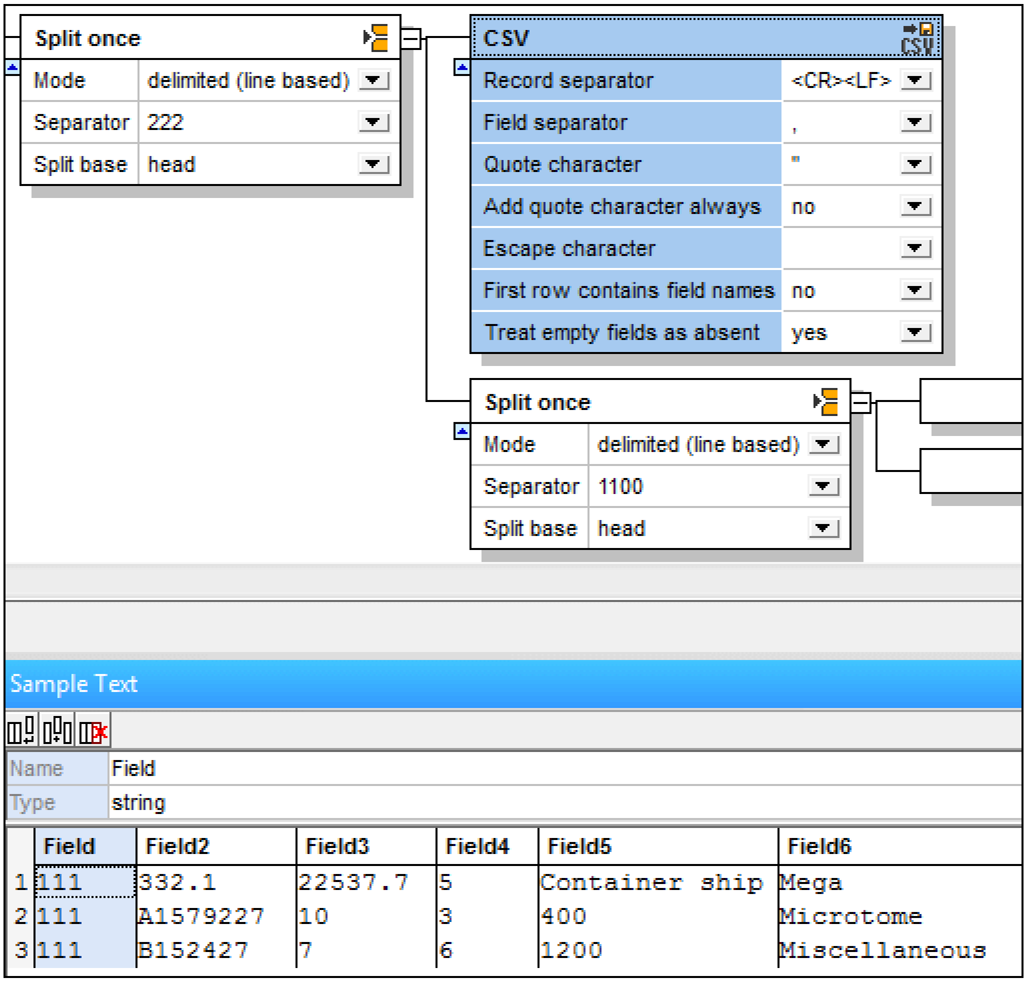
The following example shows how data in a small text file is split up into two CSV files, and mapped to separate XML files in MapForce.



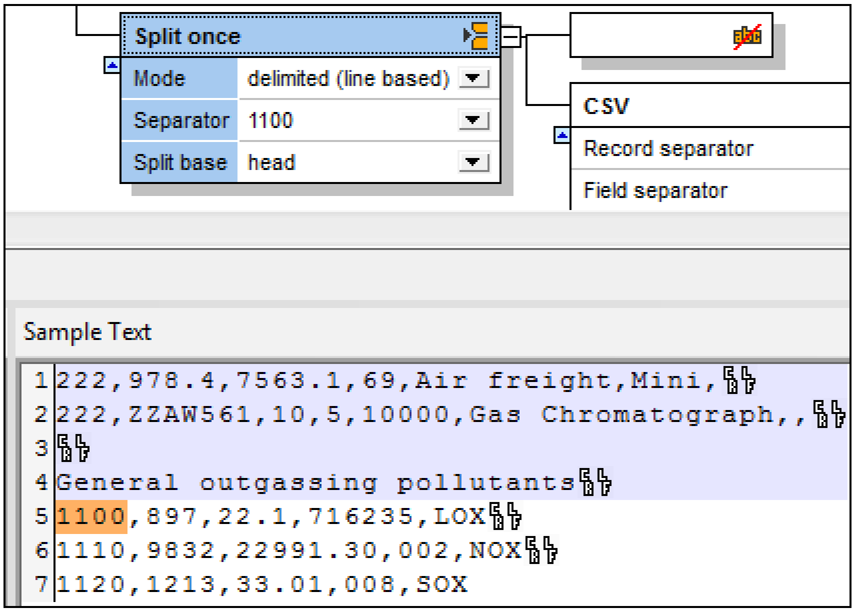
The **Split once** container shown above, is used to create two containers. The **delimited (line based)** function with the separator 222, is used to achieve this. All records up to the first occurrence of 222, are passed to the CSV container. The first, consisting of all records containing 111, is then defined as a CSV container. The Sample Text pane shows the contents of the currently active container "Split once".



The default CSV settings have not been changed. Clicking the CSV container shows its contents in tabular form.

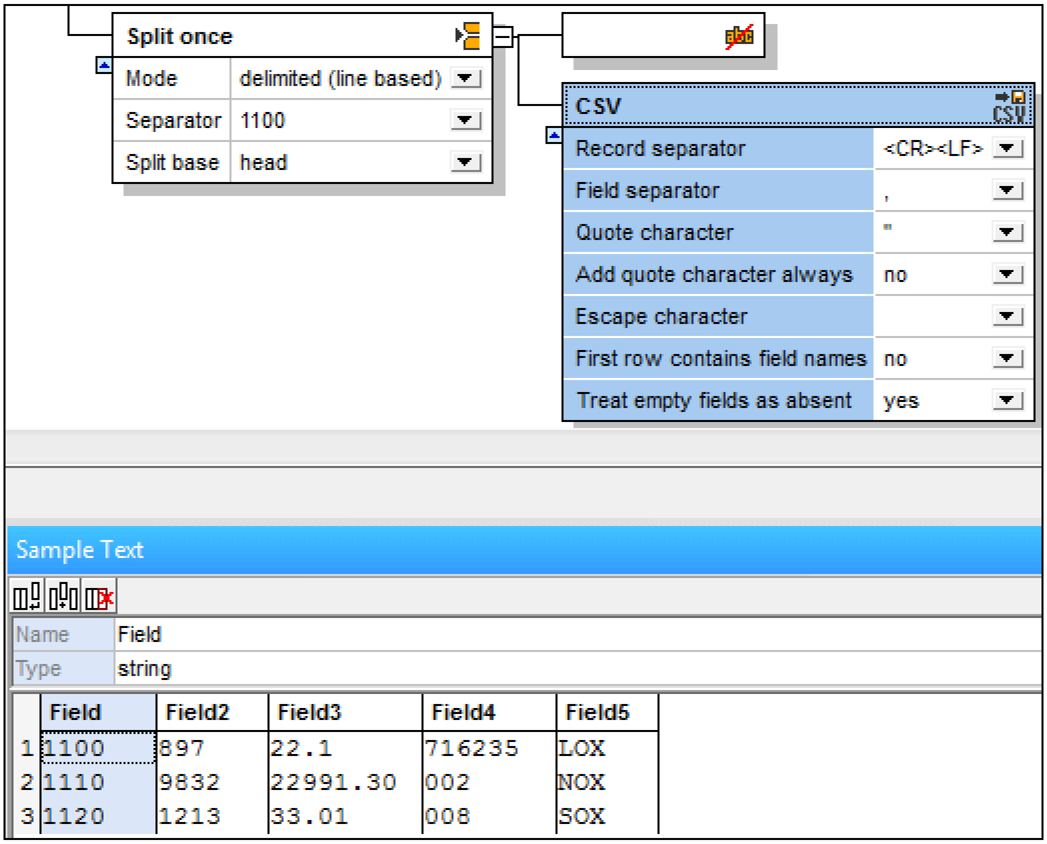


The second container holds the remaining data, and is made into another **Split once** container. This creates two more containers, one of which will be the second CSV. Clicking the Split once container, shows the current contents.

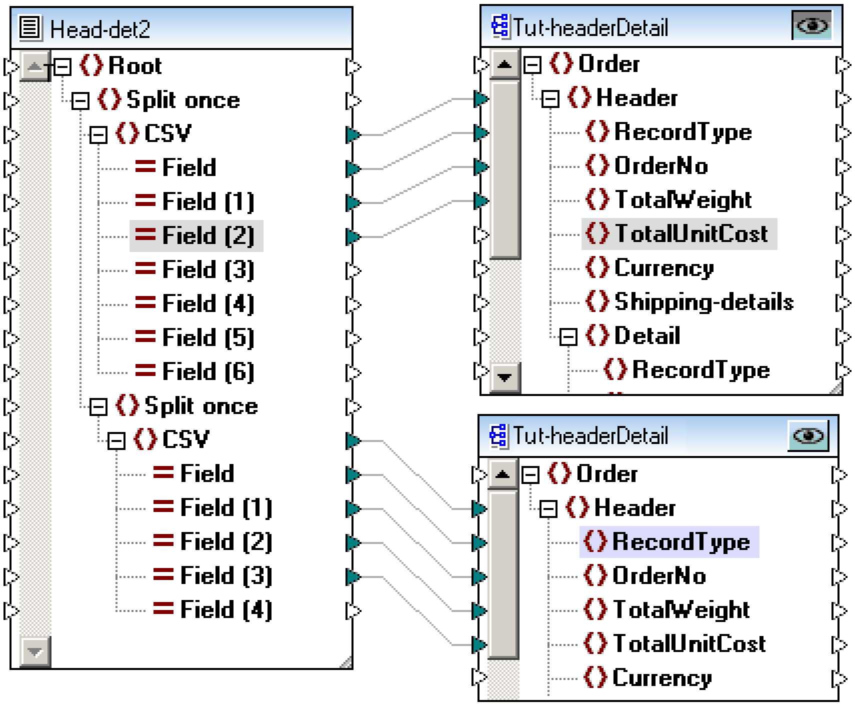


The delimited (line based) function, using 1100 as the separator, is used to split the remaining data into two sections.

- All records up to the first occurrence of 1100, are passed to the first container which is made non-mappable, by defining it as "Ignore" .
- The second container is then defined as CSV. The default settings have not been changed. Clicking the CSV container shows the contents in tabular form.



Inserting the FlexText template into MapForce allows you to map the data to any of the supported target files. In this example, each of the CSV items are mapped to two separate XML files.



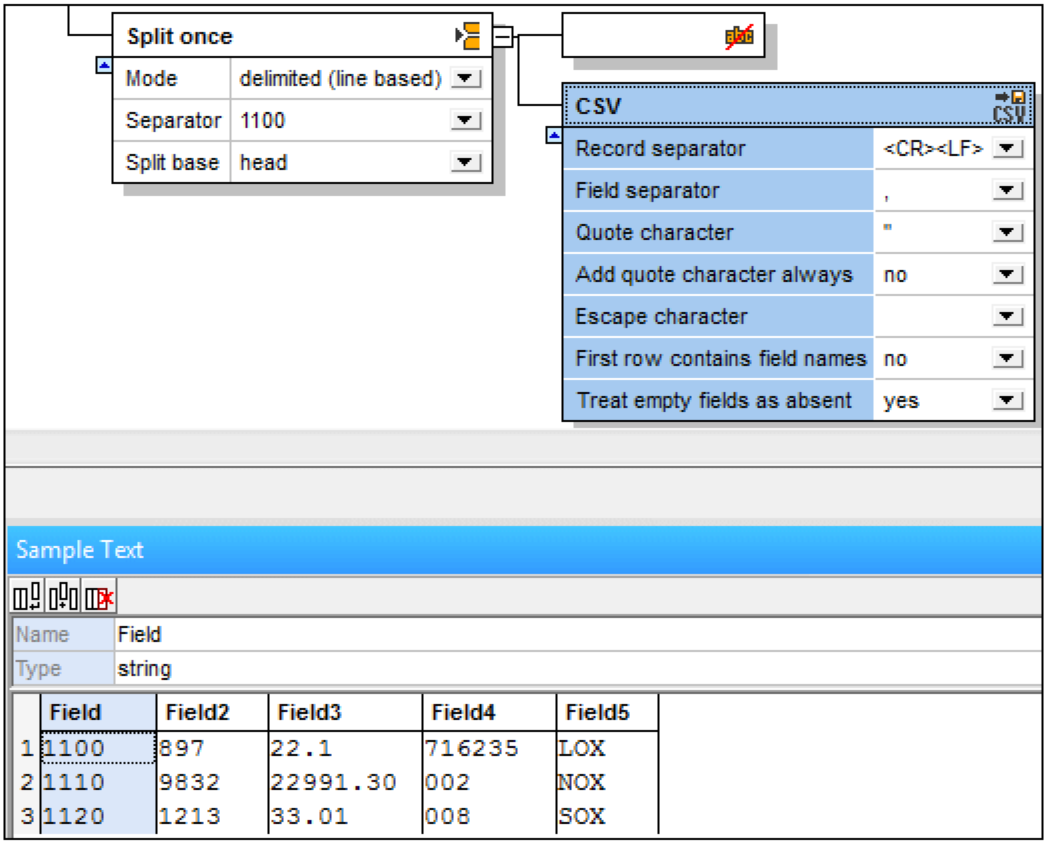
Note that not all of the items in the CSV sections are mapped to the target files. The first XML file contains all 111 record types.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
	<Header>
		<RecordType>111</RecordType>
		<OrderNo>332</OrderNo>
		<TotalWeight>22537.7</TotalWeight>
	</Header>
	<Header>
		<RecordType>111</RecordType>
		<OrderNo>0</OrderNo>
		<TotalWeight>10</TotalWeight>
	</Header>
	<Header>
		<RecordType>111</RecordType>
		<OrderNo>0</OrderNo>
		<TotalWeight>7</TotalWeight>
	</Header>
</Order>
```

The second XML file contains all records starting with 1100.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	<Header>
		<RecordType>1100</RecordType>
		<OrderNo>897</OrderNo>
		<TotalWeight>22.1</TotalWeight>
		<TotalUnitCost>716235</TotalUnitCost>
	</Header>
	<Header>
		<RecordType>1110</RecordType>
		<OrderNo>9832</OrderNo>
		<TotalWeight>22991.3</TotalWeight>
		<TotalUnitCost>2</TotalUnitCost>
	</Header>
	<Header>
		<RecordType>1120</RecordType>
		<OrderNo>1213</OrderNo>
		<TotalWeight>33.01</TotalWeight>
		<TotalUnitCost>8</TotalUnitCost>
	</Header>
</Order>
```

Configuring the CSV container/data



Clicking a field in the Sample Text pane highlights it, allowing you to configure it further.

- Click in the **Name** field to edit the default text that is presented.
- Click in the **Type** field to define the field data type: string, boolean, decimal etc.
- Click the append icon to append a new field.
- Click the insert icon to insert a field before the currently active field.
- Click the delete icon to delete the currently active field.

Note: The field boundaries can be dragged by the mouse to display the data.

Add quote character always

Allows you define if the specified quote character is to be added to all fields of the generated CSV file.

Treat empty fields as absent

Allows you to define that empty fields in the source file, will not produce a corresponding empty item (element or attribute) in the target file.

Note that the delimiters for the empty fields in the source file must exist however, e.g. "General outgassing pollutants,,,,".

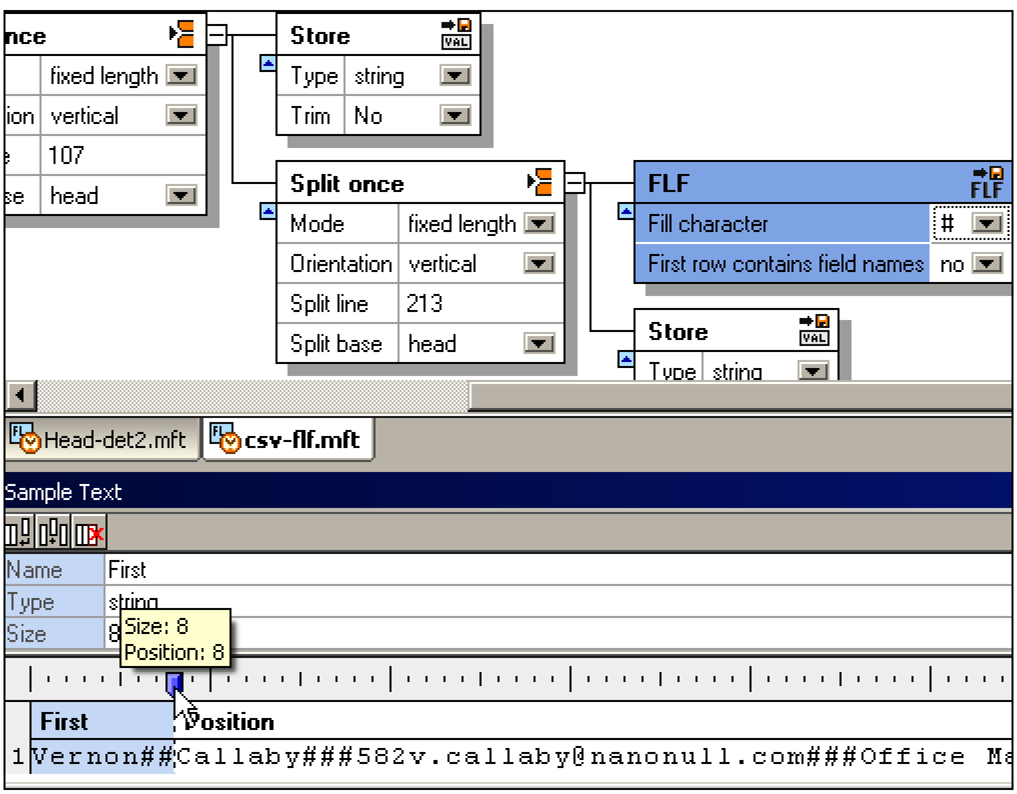
4.6.5.7 Store as FLF (fixed length)

Store FLF allows you to store text fragments as fixed length text, and map individual columns to MapForce. Any number of FLF containers/components can be created in FlexText, and each of the FLF containers may have different fill characters.

The Sample Text pane provides an overview of the current FLF fragment, and also allows you to specify field names, lengths, and widths. Each column appears as a mappable item in the text component in MapForce.

Container **default settings** are:

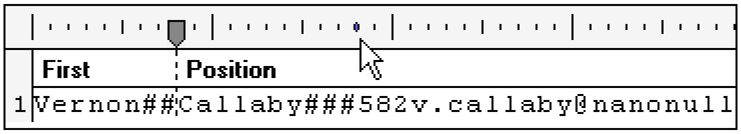
Fill character (none)
First row contains field names no
Treat empty fields as absent yes



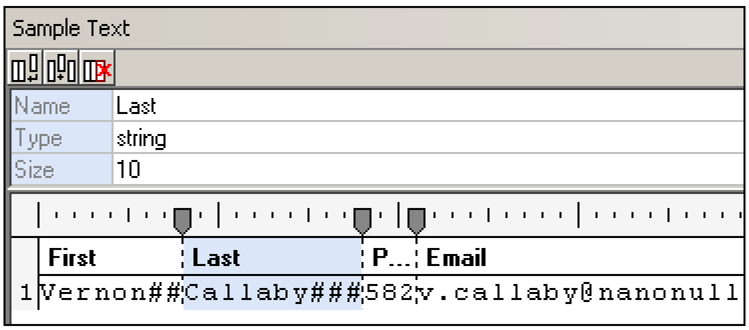
Configuring the FLF container/data

Having defined a container as "Store FLF", the Sample Text pane appears as shown in the screenshot above. A default field of width 10 is automatically inserted.

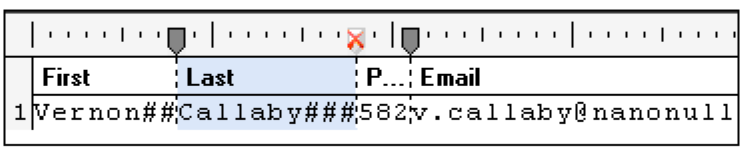
- Click the tab icon on the ruler and drag to reposition it. A tip appears showing you the current position.
- Positioning the cursor over the ruler displays a "dot"; clicking places a new tab at the click position.



- Having defined the new position, click the field to select it, and edit the name in the Name field.

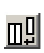
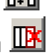


- To **remove** a field, click the tab icon and drag it off the ruler. The tab icon changes when this action can be successfully completed.

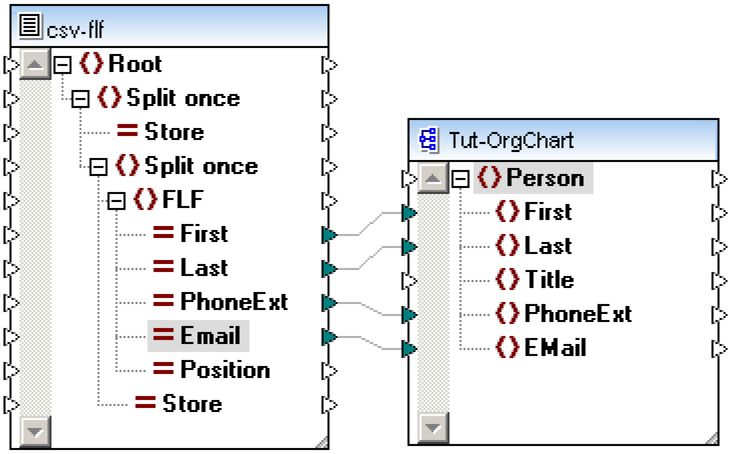


Note: When you drag a tab on the ruler, all tabs to the right of it will be automatically repositioned. To retain the other tab positions, hold down the **Shift** key before moving the tab.

Clicking a field in the Sample Text pane highlights it, allowing you to further configure it.

- Click the append icon  to append a new field, of length 10.
- Click the insert icon  to insert a field before the currently active field, length 10.
- Click the delete icon  to delete the currently active field.
- Click in the **Name** field to edit the default text that is presented.
- Click in the **Type** field to define the field data type: string, boolean, decimal etc.

Inserting the FlexText template into MapForce allows you to map the data to any of the supported target files. In this example, FLF items are mapped to XML items.



If the option **Treat empty fields as absent** is **yes**, then any empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file. A field is considered as absent if there is no data between two subsequent fill characters.

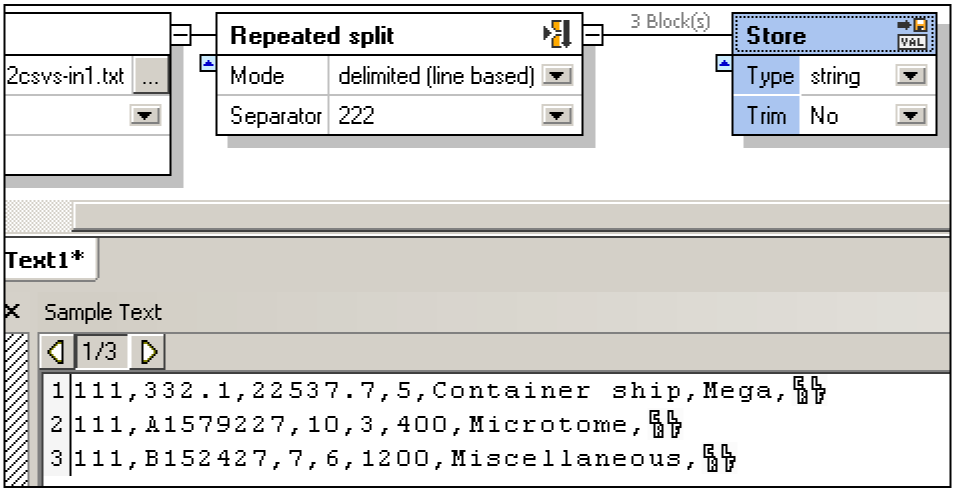
4.6.5.8 Store value

Allows you to define a container, which makes its data available as a mappable item, in MapForce. If you do not change the container name in FlexText, then the mappable item appears with the name "Store".

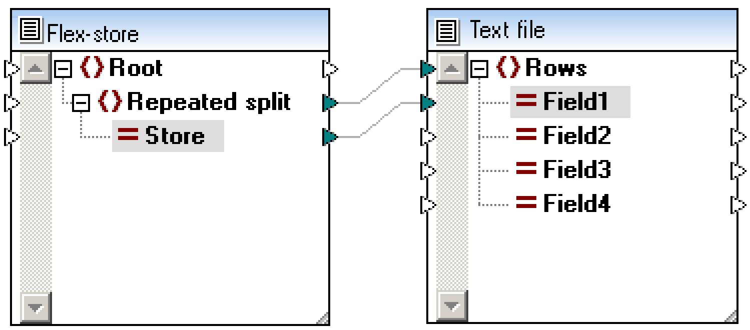
Container **default settings** are:

Type	string
Trim	no

The screenshot below shows the "Store" container with its contents visible in the Sample Text pane.



Saving this template and opening it in MapForce, allows you to map the Store item to other items in a target component.



Note: The field1 item in the target text file, will contain all 3 fragments supplied by the Store item, when you click the Output tab to preview the result.

Type

Allows you to define the data type of the text fragments.

Trim side

Defines the side from which the characters will be trimmed: left, right or both. Selecting left, right, or both activates the "Trim character set" option.

Trim character set

Defines the characters you want to trim from this text fragment. You can enter any number of characters here, by double clicking in the field. The characters you enter are removed from the Trim side(s) of the fragment.

4.6.6 FlexText and Regular Expressions

In MapForce FlexText, you can use regular expressions as follows:

1. To split text containers whenever a match is found (the matched text acts as separator). In this case, regular expressions are implicitly anchored; therefore, the caret (^) and the dollar sign ($) characters are not used. For example, to match any three consecutive digits, use `[0-9]{3}` instead of `^[0-9]{3}$`.
2. To redirect text from a Switch container if the text contains a regular expression match.

You can use regular expressions in FlexText components in any of the following MapForce target languages:

- Built-in (when previewing the mapping)
- Built-in (when running the MapForce Server execution file)
- Code generation languages (C++, C#, Java). Note that, in these languages, some advanced features of regular expressions may depend on the regular expressions implementation in that specific language.

The regular expression syntax and semantics in FlexText is based on http://www.w3.org/TR/xmlschema-2/#regexs, similar to the MapForce core function **`tokenize-regexp`**. Note the following:

- If the split condition matches two sequences following each other, FlexText creates an empty result in between (this behavior is the same when you are not using regular expressions).
- No regular expression flags (http://www.w3.org/TR/xquery-operators/#flags) are supported.

This section includes the following topics:

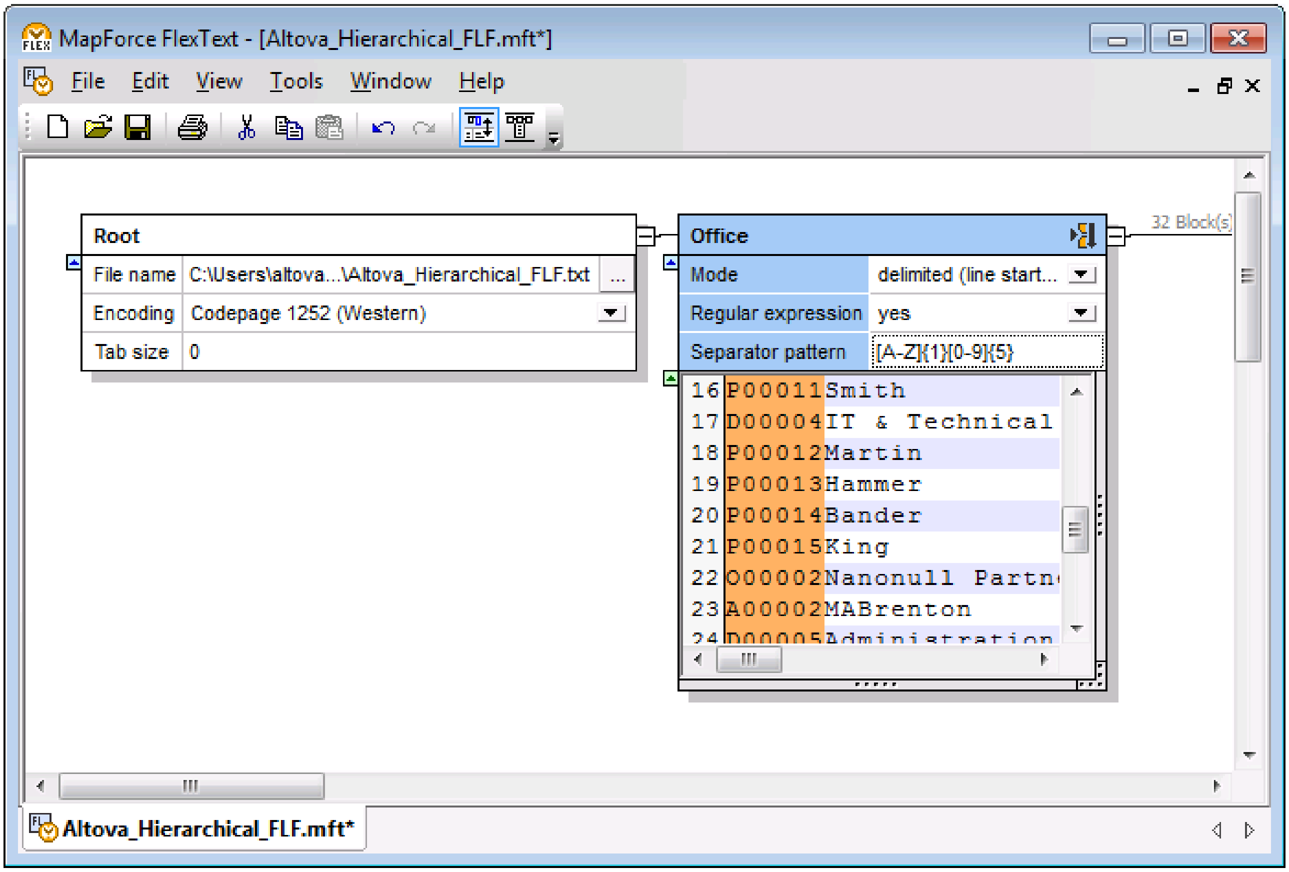
- Splitting Text with Regular Expressions [708].
- Using Regular Expressions in Switch Conditions [710]

4.6.6.1 Splitting Text with Regular Expressions

When you need to split text into two or more fragments, you can optionally use a regular expression as separator. This is an alternative, more advanced option as compared to separating text by means of single or multiple consecutive characters.

The option to split text by means of regular expressions becomes available in FlexText when the following conditions are true:

- The container is of type **Split Once** [684] or **Repeated Split** [676].
- The *Mode* option is set to **delimited (floating)**, **delimited (line based)**, or **delimited (line starts with)**.



Sample FlexText template which uses regular expressions to separate text

The options applicable to regular expressions are as follows.

Regular expression	To use a regular expression as text separator, switch this option to **yes**. To use simple text as separator, switch this option to **no** (this is the default value).
Separator pattern	If the *Regular expression* option is switched to **yes**, a *Separator Pattern* text box becomes available where you can enter the regular expression that must act as separator. In the FlexText template illustrated above, the regular expression `[A-Z]{1}[0-9]{5}` matches exactly one alphanumeric character, followed by exactly five numeric characters (for example, "P00011"). All such occurrences are highlighted in the preview and act as text splitting separator.
Separator for writing	The *Separator for writing* option is meaningful when the following conditions are true: • The FlexText component is a target component • *Mode* is set to **delimited (floating)**.

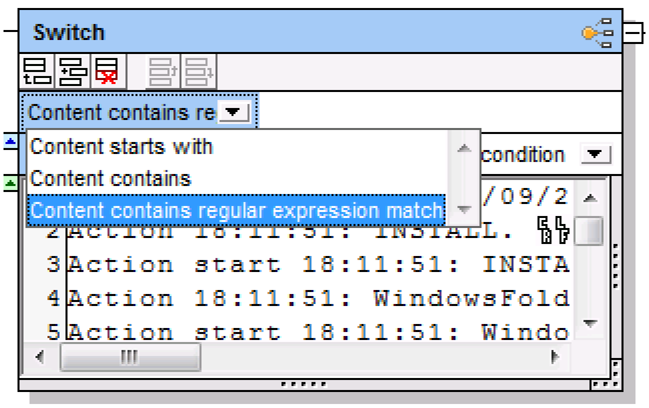
	Enter in this field the string to be written to the target component at the occurrence where a regular expression match was found.

4.6.6.2 Using Regular Expressions in Switch Conditions

When working with Switch containers (see also Switch690), you can optionally create a condition within the Switch container to redirect the text fragment if it contains a regular expression match. In such cases, you can use the caret (^) and the dollar sign ($) characters to match the beginning or end of the text to be searched, except when C++ is set as target transformation language. (In C++, the caret and dollar sign characters are interpreted as the beginning or end of a line, not as the beginning or end of the whole text).

To use a regular expression in a switch condition:

1. Define the container type as Switch690 (click on the top-right corner of the container, and then click **Switch**).
2. Click the **Append Condition** ( ) button to add a new condition.
3. Set the condition type to **Content contains regular expression match** and enter the regular expression in the adjacent text box.



Example

Let's assume that you need to map data from the database log file represented below (you can also find this file at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\SampleDatabaseLog.txt**).

```
Action 18:11:51: INSTALL.
Action start 18:11:51: INSTALL.
Action 18:11:51: WindowsFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Action start 18:11:51: WindowsFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: WindowsFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E. Return value
1.
Action 18:11:51: SystemFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Action start 18:11:51: SystemFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: SystemFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E. Return value 0.
```



```
Action 18:11:51: WindowsFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Action start 18:11:51: WindowsFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: WindowsFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E. Return value
0.
Action 18:11:51: SystemFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Action start 18:11:51: SystemFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: SystemFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E. Return value 1.
Action 18:11:51: WindowsFolder.63E949F6_03BC_5C40_FF1F_C8B3B9A1E18E.
Action start 18:11:51: WindowsFolder.63E949F6_03BC_5C40_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: WindowsFolder.63E949F6_03BC_5C40_FF1F_C8B3B9A1E18E. Return value
1.
```

Your goals are as follows:

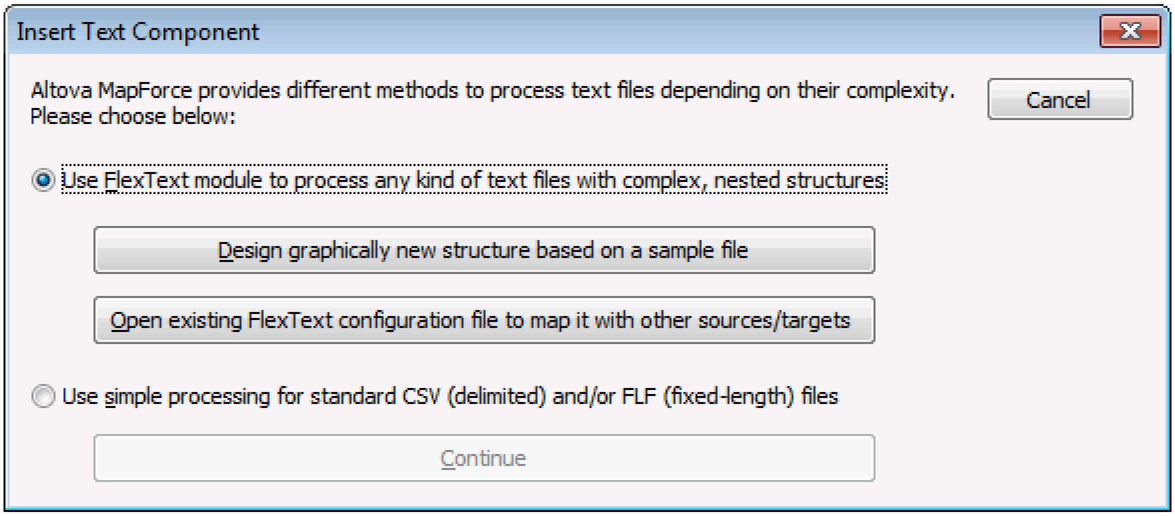
1. Collect in a list all rows where a return value is present. That is, the list must include every row which contains the value "Return value 1" or the value "Return value 0", or any other return value expressed as a digit.
2. Collect in another list all rows where the text contains the value "Action start".

To achieve these goals, you can use a *Repeated Split* container to split down the text into individual rows. After that, you can use a *Switch* container to redirect each row as required. Namely, the *Switch* container will consist of three conditions, as follows:

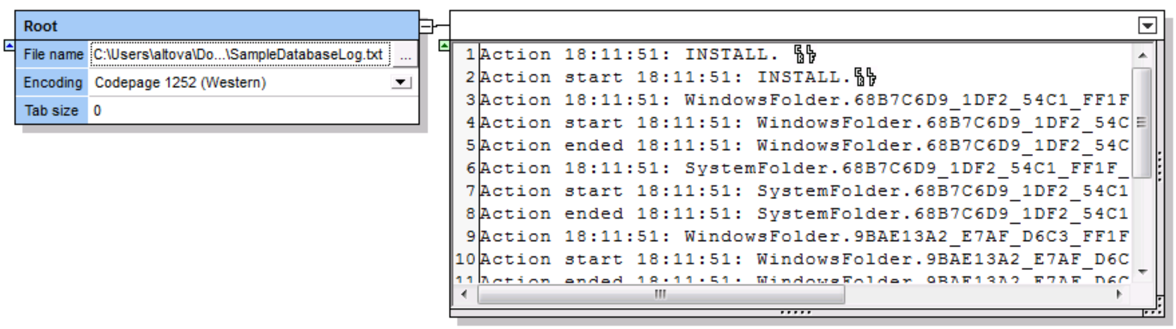
1. Redirect the current row to output A if it contains the value "Action started". You can find such rows by using a condition of type "Content starts with", and enter "Action started" as value.
2. Redirect the current row to output B if it contains a return value. You can find such rows by using the regular expression `Return value [0-9]\.` . This regular expression will return a match if the row contains the text "Return value", followed by a single digit, followed by a full stop. The backslash (\ ) before the full stop acts as an escape character, to denote that the full stop must be treated as normal character, not as a metacharacter.
3. Redirect the current row to output C (<default>) if the row satisfies neither of the conditions above.

To create the FlexText template which performs the tasks above:

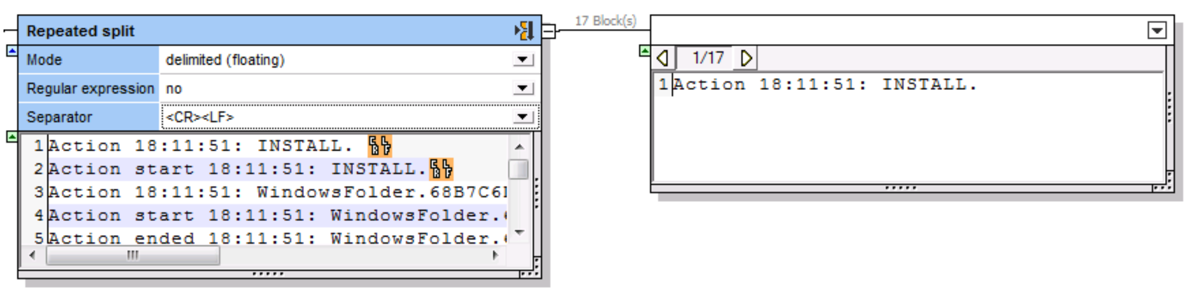
1. On the **Insert** menu, click **Text File**.



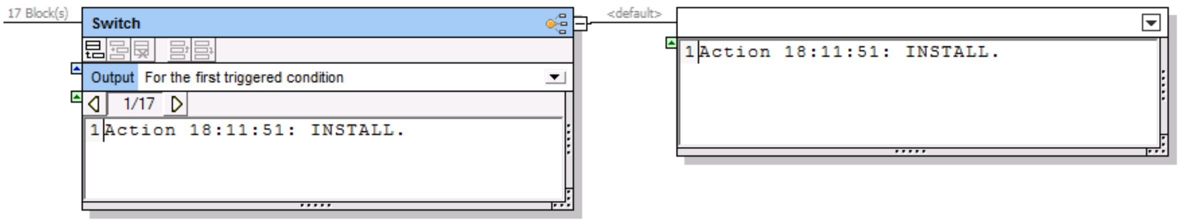
2. Click **Design graphically new structure based on a sample file** and save the FlexText .mft template to a directory of your choice.
3. When prompted to open a text file, browse for the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\SampleDatabaseLog.txt** file.



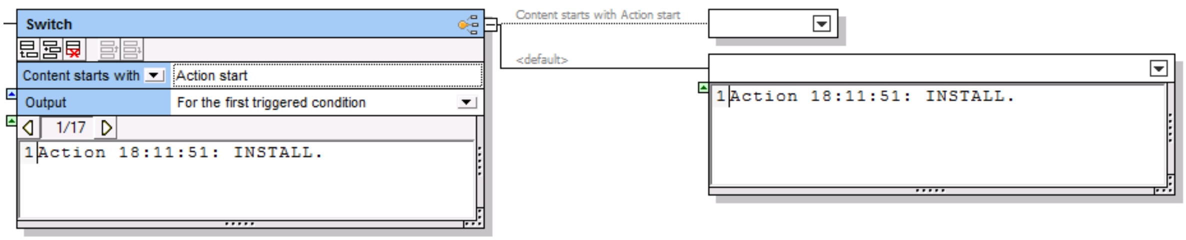
4. Click the top-right corner of the output container and select **Repeated Split**. Because we are using carriage return as split character, choose **delimited (floating)** mode, and **<CR><LF>** as separator. This creates a new output container which consists of 17 blocks (one for each row).



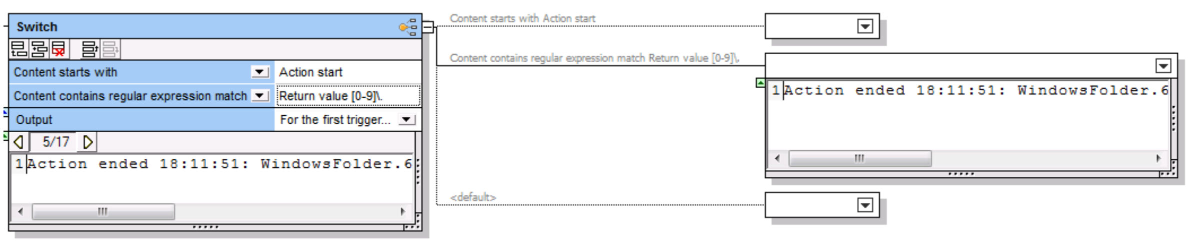
5. Click the top-right corner of the new output container and select **Switch**. Now FlexText will treat the contents of the container as a switch. As shown below, one <default> switch condition was created automatically—this condition redirects to a new container any text that does not match other conditions. At this stage, there are no other conditions defined, therefore all text is currently being redirected to the **<default>** output.



6. Click the **Append Condition** ( ) button and add the condition of type **Content starts with** with the value "Action start", as shown below. This condition redirects to a new container any text that begins with "Action start".

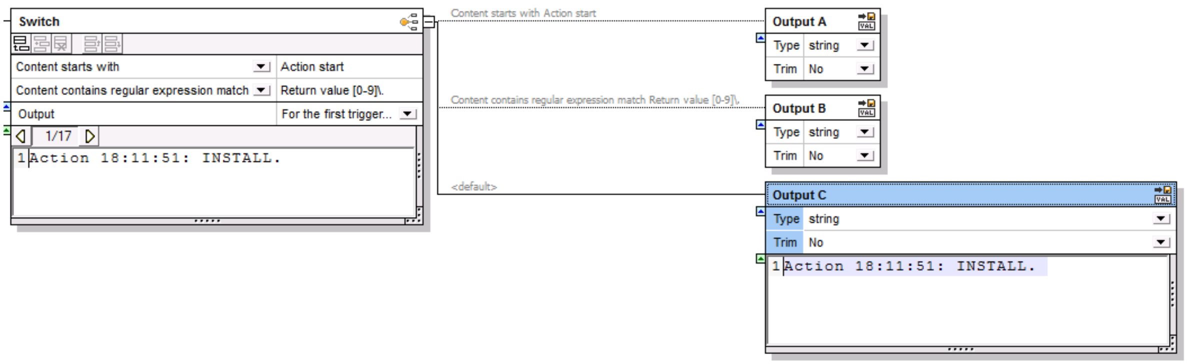


7. Click the **Append Condition** ( ) button and add the condition of type **Content contains regular expression match** with the value `Return value [0-9]\.` . If you now navigate to block 5 out of 17, you can see that this condition redirects that block to a new container, since the block contains a match for the regular expression.



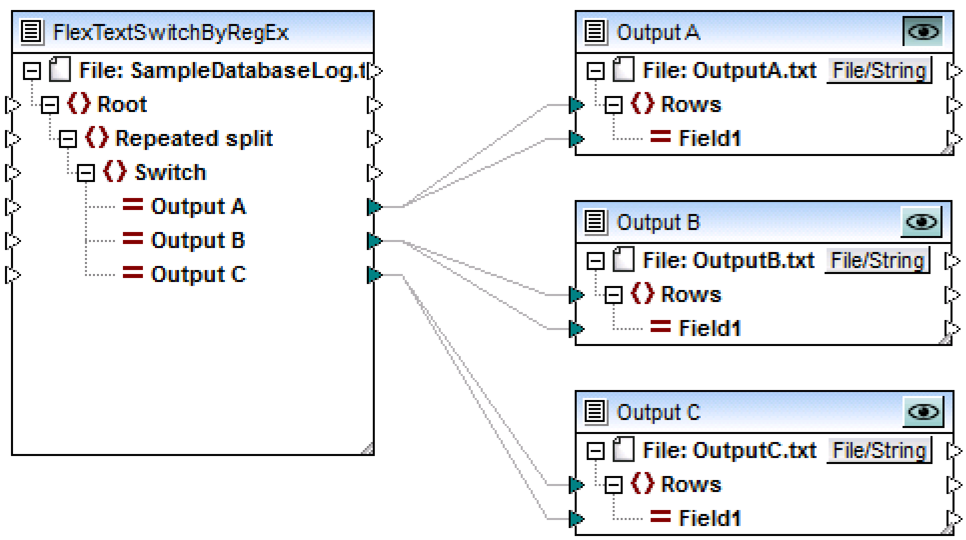
You have now configured the Switch container so that it redirects text to a different output based on a conditions. There are three switch conditions and three possible outputs (one output for each condition). The remaining steps of this tutorial show how you can write each of output to a separate text file.

8. Click the top-right corner of each output container and choose **Store as Value**. Additionally, double-click the title bar of each output and enter a descriptive name: **Output A**, **Output B**, and **Output C**, respectively.



9. Save and close the FlexText template.

The required FlexText template has now been created. You can see how this template works by opening the following tutorial file:
<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\FlexTextSwitchByRegEx.mfd.



FlexTextSwitchByRegEx.mfd

The **FlexTextSwitchByRegEx.mfd** mapping illustrated in the screen shot reads text data from the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\SampleDatabaseLog.txt** file and converts it into three separate text files: Output A, Output B, and Output C. Each of the target text files contains only the rows that satisfy one of the three conditions defined previously in the FlexText template. To view the output of a particular target component, click the **Preview Component** ( ) button in the top-right corner of the component, and then click the **Output** tab.

4.7 Microsoft OOXML Excel 2007+

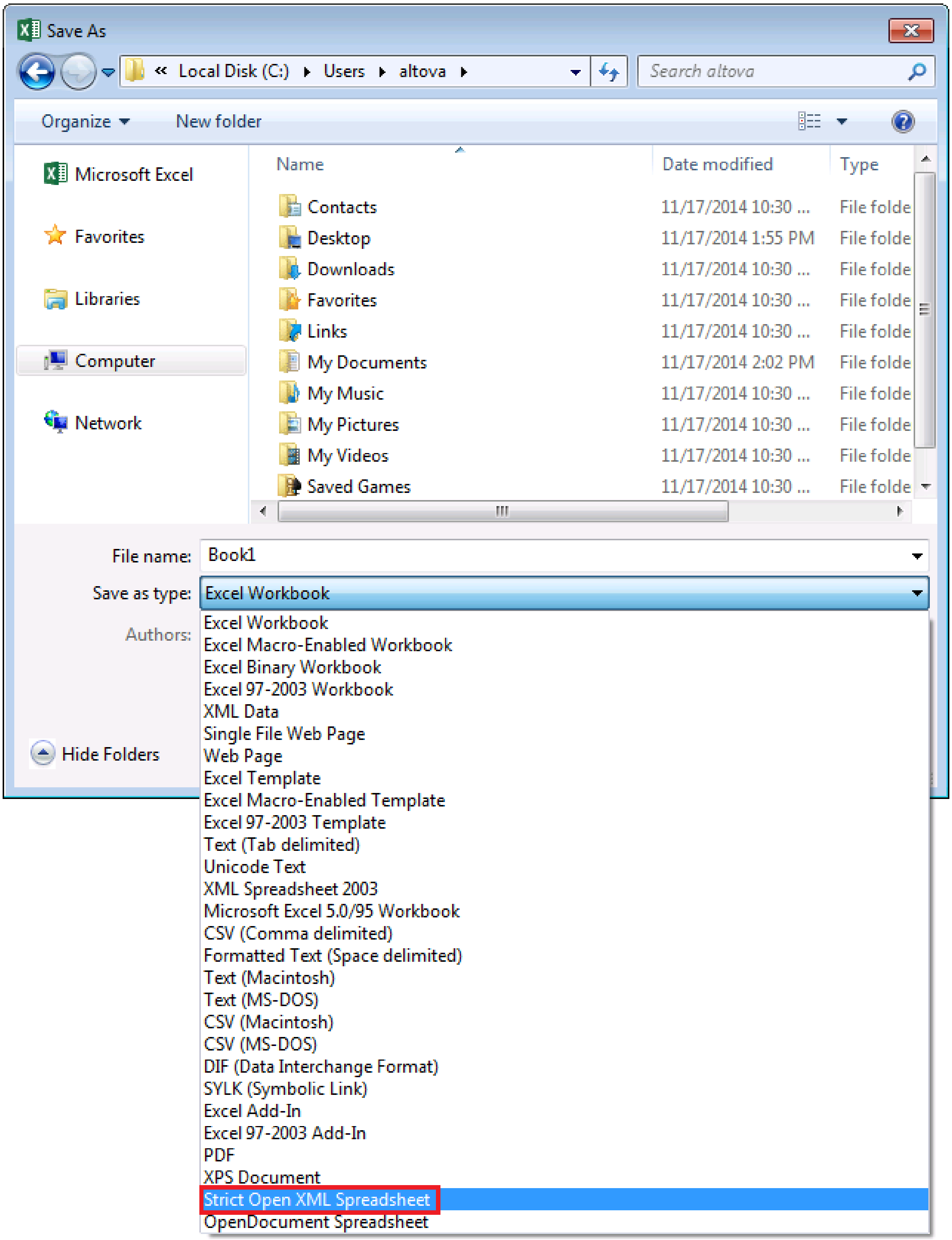
MapForce can read data from and write data to Microsoft Excel 2007+ workbooks, in the default *Office Open XML (OOXML)* format. This format was first introduced in Microsoft Office 2007 and, in case of Excel workbooks, is associated with the default .xlsx extension. In the MapForce interface, and in this documentation, Microsoft Excel 2007 and later files are generically referred to as "Excel 2007+" files.

Microsoft Office 2013 added support for *Strict Open XML Spreadsheet* format (ISO/IEC 29500 Strict). The *Strict Open XML Spreadsheet* format also has the .xlsx extension; however, technically this is a distinct format which adheres to stricter validation rules.

The following table illustrates how reading and writing data from/to Excel 2007+ workbooks is supported across MapForce transformation languages. Notice the differences between *Office Open XML* and *Strict Open XML Spreadsheet* formats.

Microsoft Excel Format	**MapForce Language**	**Reading Support**	**Writing Support**
Office Open XML	BUILT-IN	Yes	Yes
	C#	Yes	Yes
	Java	Yes	Yes
	XSLT 2.0, XSLT 3.0	Yes	No
Strict Open XML Spreadsheet	BUILT-IN	Yes	No
	C#	Yes	No
	Java	Yes	No
	XSLT 2.0. XSLT 3.0	Yes	No

If you need to convert any *Office Open XML* files generated by MapForce to *Strict Open XML Spreadsheet* format, open the workbook in Excel 2013, and then save it as *Strict Open XML Spreadsheet*.


Save As
« Local Disk (C:) ▸ Users ▸ altova ▸
Search altova
Organize ▾
New folder
Microsoft Excel
Favorites
Libraries
Computer
Network
Name
Date modified
Type
Contacts 11/17/2014 10:30 ... File folde
Desktop 11/17/2014 1:55 PM File folde
Downloads 11/17/2014 10:30 ... File folde
Favorites 11/17/2014 10:30 ... File folde
Links 11/17/2014 10:30 ... File folde
My Documents 11/17/2014 2:02 PM File folde
My Music 11/17/2014 10:30 ... File folde
My Pictures 11/17/2014 10:30 ... File folde
My Videos 11/17/2014 10:30 ... File folde
Saved Games 11/17/2014 10:30 ... File folde
File name: Book1
Save as type: Excel Workbook
Authors:
Hide Folders
Excel Workbook
Excel Macro-Enabled Workbook
Excel Binary Workbook
Excel 97-2003 Workbook
XML Data
Single File Web Page
Web Page
Excel Template
Excel Macro-Enabled Template
Excel 97-2003 Template
Text (Tab delimited)
Unicode Text
XML Spreadsheet 2003
Microsoft Excel 5.0/95 Workbook
CSV (Comma delimited)
Formatted Text (Space delimited)
Text (Macintosh)
Text (MS-DOS)
CSV (Macintosh)
CSV (MS-DOS)
DIF (Data Interchange Format)
SYLK (Symbolic Link)
Excel Add-In
Excel 97-2003 Add-In
PDF
XPS Document
Strict Open XML Spreadsheet
OpenDocument Spreadsheet

Saving to Strict Open XML Spreadsheet format

As mapping components in MapForce, Excel 2007+ files have the following general behavior:

- You can map data from Excel 2007+ to any component supported in MapForce, and vice versa, including XBRL taxonomy files (see Excel to XBRL example [790]).
- If Microsoft Excel 2007 or later is installed on your computer, you can preview the transformation output immediately in the **Output** tab of the mapping window, and you can save it to a file. If you don't have Excel 2007 or later, you can still map to or from Excel 2007+ files. In this case, you cannot preview the result in the **Output** tab, but you can still save it, by clicking **Save Output File** on the **Output** menu.

> If you are using multiple high-DPI monitors, and if they have different DPI-awareness levels, it is not possible to preview the Excel 2007+ output directly in MapForce. Instead, a message similar to the following is displayed: "Embedding of Excel is not possible with the current system configuration".

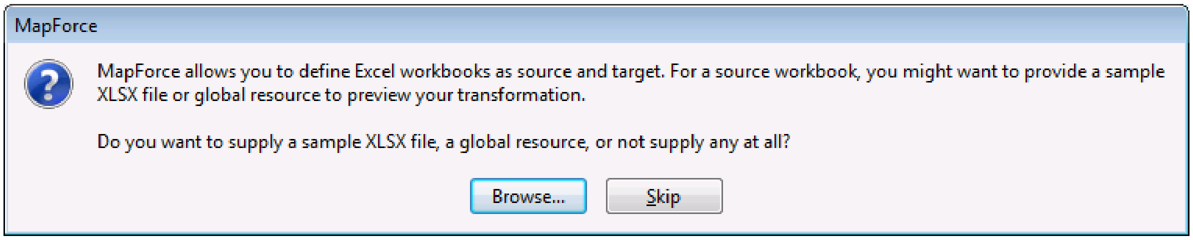
- You can choose from the component settings [729] whether to create new Excel files from a mapping, or update existing Excel files.
- If you choose to create new Excel files, the generated Excel 2007+ output does not contain any formatting, only data.
- If you choose to update existing Excel files, the formatting information of your existing Excel file is preserved as it is. Note that updating cells that contain formulas is not supported. Instead, you can configure from the component settings [729] to stop the mapping execution with an error when such cells are encountered, or ignore them (while still displaying warnings in the Messages window).
- As an alternative to generating and saving the output manually, you can compile the mapping design to a MapForce Server execution file, or deploy it to a FlowForce Server, and execute it as and when required through FlowForce Server jobs. For further information, see Compiling a MapForce mapping [1378] and Deploying a MapForce mapping [1381], respectively.
- If supported by the mapping components, you can generate program code from a mapping (see the supported languages above) and execute it externally from your custom application. For more information, see Code Generator [1492]. Note that the **Update existing Excel files** feature is supported only when the transformation language is BUILT-IN.

4.7.1 Adding Excel 2007+ Files as Mapping Components

To add a Microsoft Excel 2007+ (*.xlsx) component to the mapping area:

1. Do one of the following:

    o On the **Insert** menu, click **Excel 2007+ File**.
    o Click the **Insert Excel 2007+ File** ( ) toolbar button .

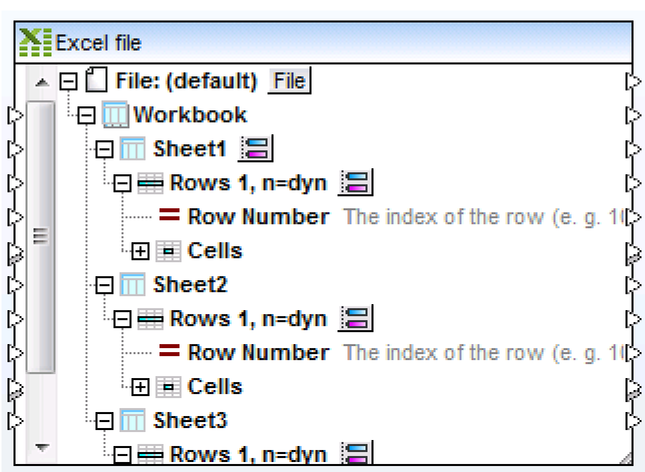


2. Do one of the following:

   - If you want to map data *from* a Microsoft Excel workbook, click **Browse** to select the .xlsx file from which you are mapping data. MapForce uses the sample Excel file to read worksheet names and cell ranges from it. If you have defined any Excel files as global resources, you can also select them from the Global Resources dialog box (see Global Resources 1397). Click **Skip** if you would like to provide a sample file later.
   - If you want to map data *to* a Microsoft Excel workbook, click **Skip**. By default, when the mapping transformation runs, MapForce will generate an output Excel file named **xlsx-mapforce.xlsx** in the mapping folder. If required, you can change the name of the output file from the Excel 2007+ Component Settings (see About the Excel 2007+ Component 718).

4.7.2 About the Excel 2007+ Component

When you add an Excel 2007+ file to the mapping area without specifying a sample file (see Adding Microsoft Excel Files as Mapping Components 717), MapForce creates a default component which includes three worksheets (illustrated below). If you provide a sample file, MapForce reads the sample file and creates only the required worksheets.



Default Excel component

The structure of the Excel 2007+ component in MapForce reflects the structure of data in the Excel workbook, with the difference that in MapForce it is expressed in a tree structure which makes it possible to map each individual cell.

Before you can connect the Excel 2007+ component to any other component type, you will need to instruct MapForce precisely what are the columns and rows to be used in the mapping. Unlike other MapForce components such as XML or JSON, Excel 2007+ files do not have an explicit schema that MapForce can use to infer the structure of your data. Instead, MapForce provides you with settings from where you can define:

- What data precisely (such as worksheets, named ranges and tables, columns, rows) must be selected from your workbook (if you are reading from a workbook);
- To which worksheets, columns, and rows must MapForce write data (if you are writing to a workbook).

You can also configure the component to read from multiple locations within a workbook, or write to multiple locations, in the same mapping operation.

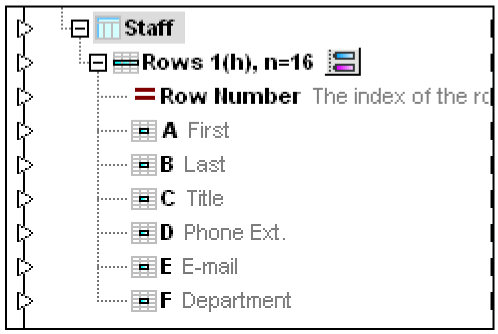
The required component configuration settings are available directly on the component. Use the following table to get started.

<table>
<tr><th>Component Item</th><th>Icon</th><th>Description</th></tr>
<tr><td>Workbook</td><td></td><td>Represents an Excel workbook.</td></tr>
<tr><td>Worksheet</td><td></td><td>Represents an Excel worksheet. The button displayed next to the first worksheet lets you specify worksheet-related settings, see Adding and Removing Worksheets 720.</td></tr>
<tr><td>Rows</td><td></td><td>Represents a range of Excel rows. You can add multiple ranges of rows within a worksheet, see Adding and Removing Row Ranges 722. This enables you read from (or write to) multiple ranges of cells in the same mapping operation.

For each defined range, you can specify individual data selection options. For example, one range may begin at row 1 and include all columns of that row, while another range may begin at row 3, and consist of a dynamic number of rows, depending on the amount of data in the source Excel file.

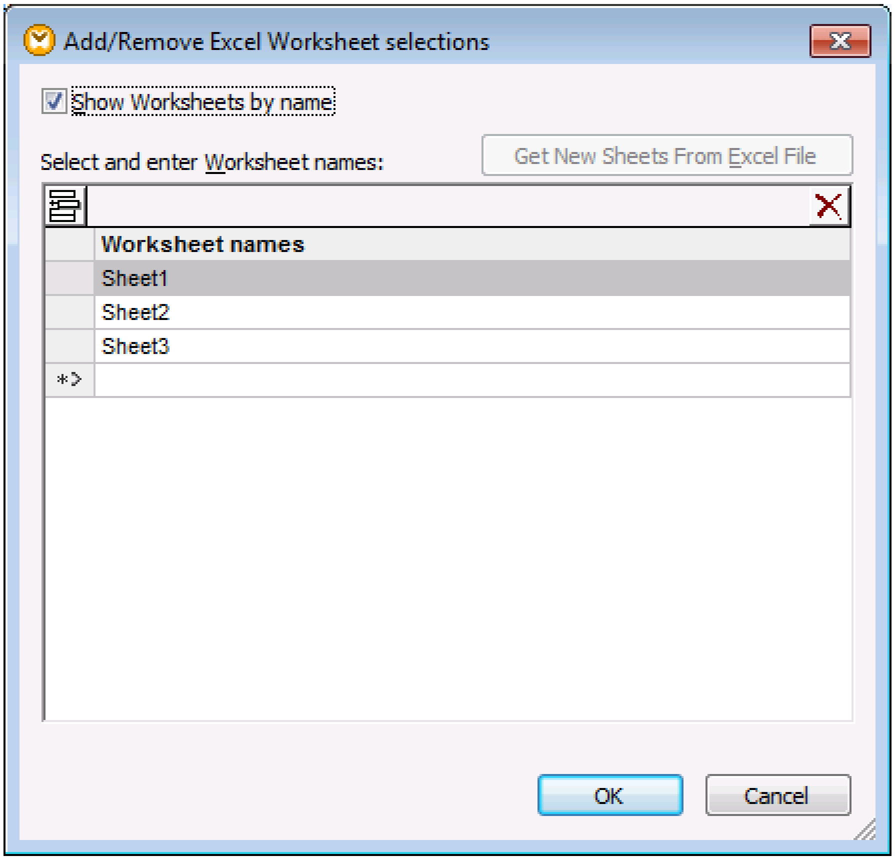
To help you see all range settings at a glance, the component provides visual clues about them, as shown below.

<table>
<tr><td>Rows n</td><td>Indicates a range which begins at row n.</td></tr>
<tr><td>Row n</td><td>Indicates a single-line range of row n.</td></tr>
<tr><td>Rows prev+n</td><td>Indicates a range which begins n rows after the previous range.</td></tr>
<tr><td>Rows n(h)</td><td>Indicates a range which begins at row n, and the first row is designated as a header row.</td></tr>
<tr><td>n=n</td><td>Indicates a range which consists of exactly n rows.</td></tr>
<tr><td>n=dyn</td><td>Indicates a dynamic range. Dynamic ranges may have an unlimited number of rows.</td></tr>
</table>
The button displayed next to each row range lets you specify advanced data selection settings for that range.</td></tr>
</table>

Component Item	Icon	Description
Cells		Represents all the cells (columns) of a particular row. This item appears if the component is configured to show a single cell for all columns (this is the default MapForce behavior). Alternatively, you can configure a component to display each column separately, in which case it would look as shown in the following sample. 
Change Selection		The button displayed next to each worksheet or row lets you specify settings meaningful in that context. Using this button, you can modify the mapping structure of the Excel component as required, see Excel 2007+ Component Settings 729. For example, if you are reading data from an Excel file, you can specify the worksheet, row and column from where MapForce should read data. If you are writing to an Excel file, you can specify the worksheet, row and column to which MapForce should write data.

4.7.3 Adding and Removing Worksheets

You can add or remove worksheets on the Excel 2007+ component, either manually, or by reloading them from an input .xlsx file. To do so, click the button next to a worksheet node.



Add/Remove Excel Worksheet selections dialog box

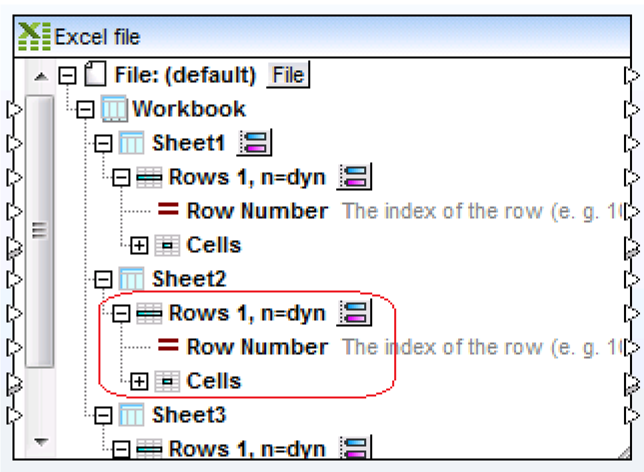
The options applicable to worksheets are as follows.

Show Worksheets by name	This option must be selected if each worksheet in your workbook has a different layout and therefore must appear as a separate item in MapForce. If all worksheets in your workbook have an identical structure, you can make this option inactive. This way, MapForce will collapse the worksheet items to a single item representing the ordered collection of all worksheets. If a workbook has multiple worksheets and the **Show Worksheets by name** option is inactive, then MapForce treats the workbook like a single worksheet. This allows a mapping to process any number of worksheets at once, but requires that all worksheets have the same structure. See Example: Convert Excel Rows to XML Files 734.
Insert ( )	Inserts a new worksheet before the currently selected one.

Append ( )	Appends a new worksheet. Type a name into the text field to the right of the icon.
Delete ( )	Deletes the currently selected worksheet.
Get New Sheets From Excel File	Gets into the component the new worksheets found in the input Excel file. This setting is relevant and enabled in source components which have an input Excel file.

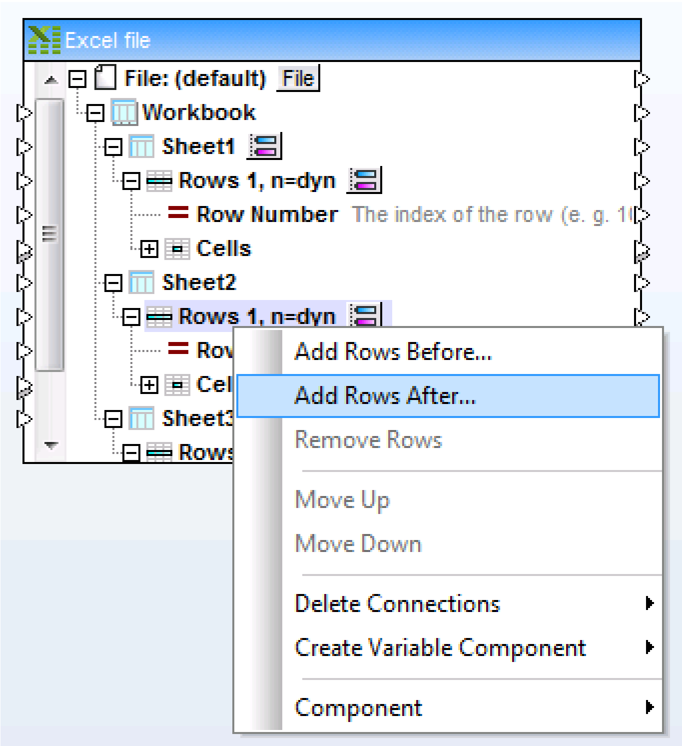
4.7.4 Adding and Removing Row Ranges

You can instruct MapForce to read (or write) a particular range of cells at a particular location within a worksheet. The following sample component illustrates a range which is available in Sheet2 of the workbook.



By default, any range is set to begin at Row 1 and iterate dynamically for n rows (**n=dyn**). However, you can change these and other settings if required (see Selecting Ranges of Cells 723).

You can create as many ranges of rows as required within the same worksheet, and remove the ones you do not need. To add a new range of rows, right-click any Rows ( ) node, and then select **Add Rows Before** or **Add Rows After**, respectively.



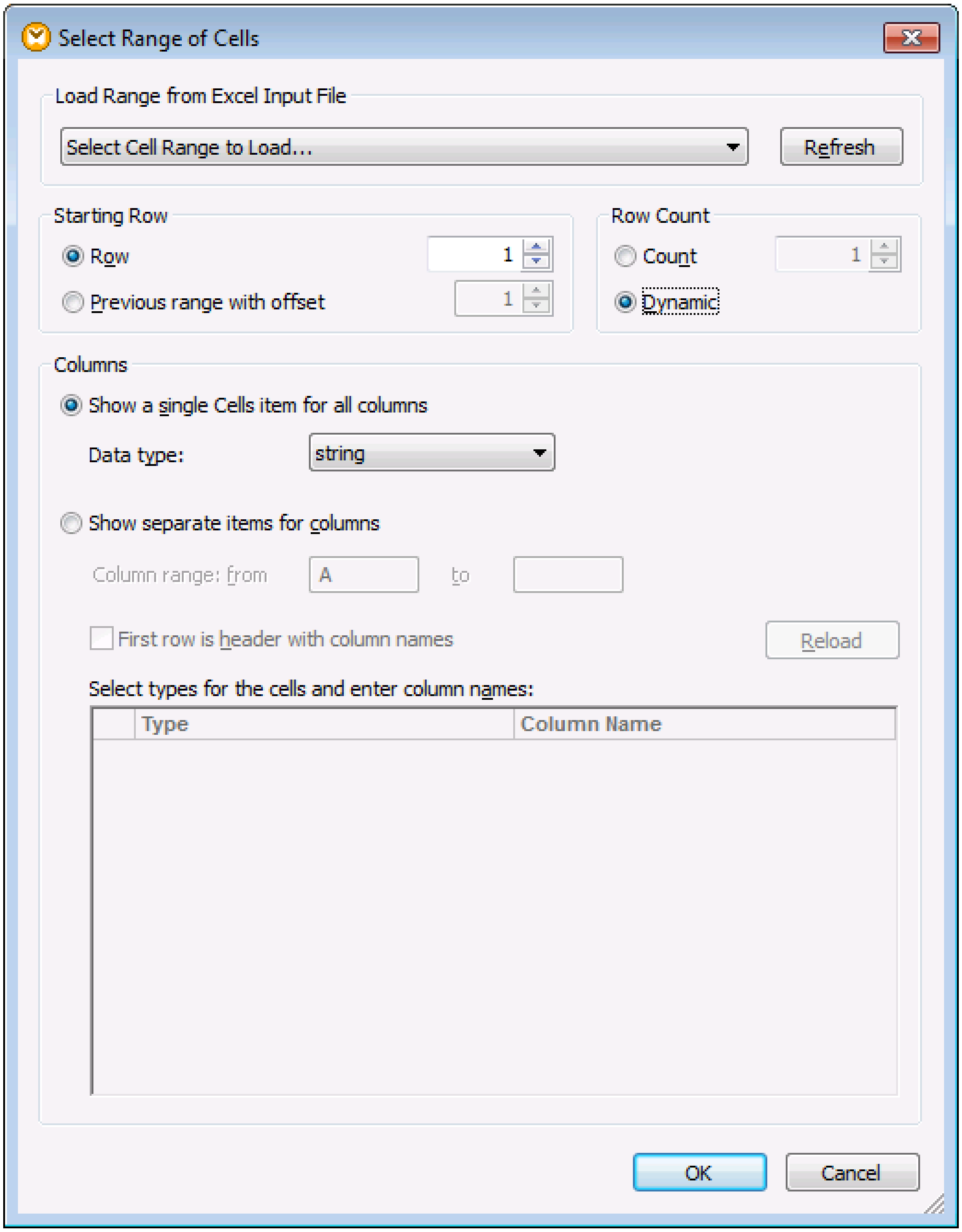
Adding cell ranges

To delete a range, right-click it, and then select **Remove Rows**.

To move a range up or down in the component, right-click it, and then select **Move Up** (or **Move Down**, respectively).

4.7.5 Selecting Ranges of Cells

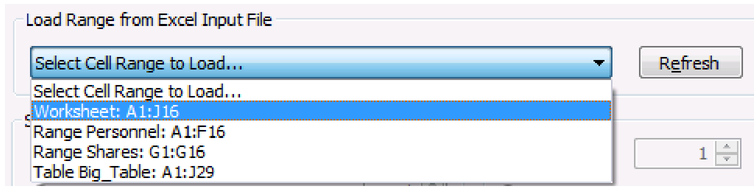
You can define what range of cells must be read by MapForce (when reading from a workbook) or written to (when writing to a workbook) from the "Select Range of Cells" dialog box. To open this dialog box, click the button next to a cell range on the component.



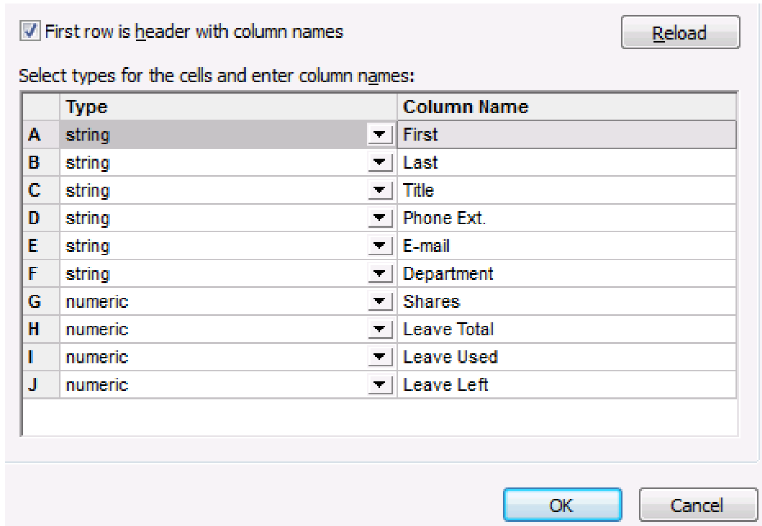
Select Range of Cells dialog box

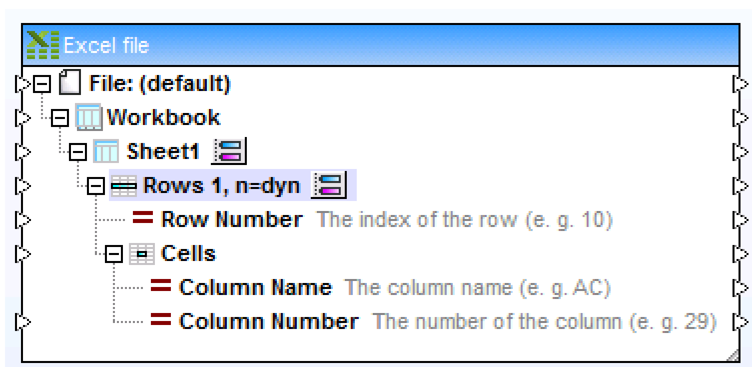
The available settings are as follows.

Load Range from Excel Input File	If you are reading from a workbook, use this option to select a particular worksheet range, named range or table.

	 If the **Show Worksheets by name** option is disabled (see Adding and Removing Worksheets 720), data from all worksheets is visible in the list. The **Refresh** button updates the cell ranges from the input Excel file. Note that only rectangular ranges are currently supported. If you are writing to a workbook, this option is not available.
Starting Row	The **Row** option lets you define the first row of data for the specific range. For example, if you enter "5" as starting row, MapForce will read (or write) beginning with the fifth row of the workbook. The **Previous range with offset** option is meaningful if there is another range in the same worksheet. It instructs MapForce to move the current range *N* rows down from the previously defined range. The minimum offset value is 1.
Row Count	*If you are reading from a workbook* **Count** defines the exact number of rows from which you want to read data, starting from the position defined in the **Starting Row** (see previous option). This value is automatically populated if you selected an Excel named range or table. The option **Dynamic** instructs MapForce to read all rows found in the source data beginning with **Starting Row**. Use this option only if your range is the last defined range of the worksheet, otherwise any subsequent range will not select data from the source Excel file. *If you are writing to a workbook* **Count** defines the exact number of rows to which data should be written, starting from the position defined in the **Starting Row** (see previous option). Note that if your input instance contains more rows than allowed by **Count**, MapForce writes only the number of rows defined by **Count**, and ignores the rest of data without any warning. The option **Dynamic** instructs MapForce to write all rows found in the input instance, beginning with **Starting Row**.

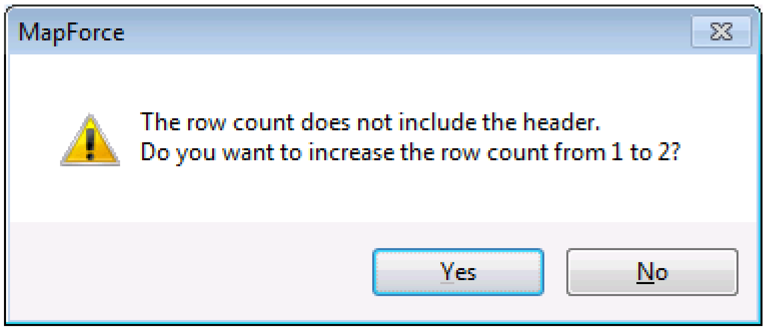
<table>
<tr>
<td></td>
<td>If you defined a header row using the First row is header with column names option, Count does not take the header row into account (see the Excel_Company_to_XML.mfd sample).


</td>
</tr>
<tr>
<td>Show a single Cells item for all columns</td>
<td>Collapses all cell items into a single mappable Cells item as shown below.



If you are reading from a workbook
Use this option if you want to read all the cells of a particular row. For examples, see the ExcelColumnsToRecords.mfd and ExcelWith2Dimensions.mfd samples available in the MapForceExamples project, OOXML Excel 2007+ folder.

If you are writing to a workbook
Use this option to write data to one or multiple cells in the same row. For an example, see the Altova_Hierarchical_Excel.mfd sample.</td>
</tr>
<tr>
<td>Show separate items for columns</td>
<td>This option enables you to access individual columns of the given row range.</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
If you selected a worksheet range, named range or table, the column names are automatically populated. Otherwise, you can select specific column names by typing their corresponding alphabetic letter in the from and to text boxes.
If the ranges in the input Excel file have changed, click Reload to update the component with the changes.
To instruct MapForce to consider the first row of a range as the column header for that range, select the First row is header with column names option. When you activate or deactivate this option, and Row Count has been set, MapForce prompts you to optionally adjust the Row Count value. This prevents the Row Count from being one row too large, or too small.


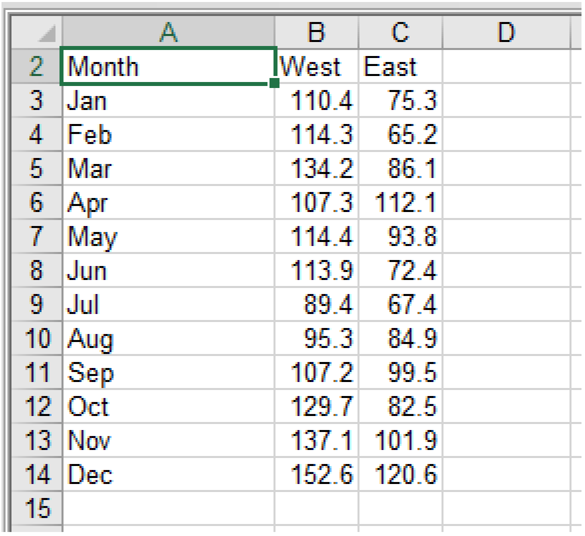
Note that the Row Count setting does not take the header row into account.
</td>
</tr>
</table>

4.7.6 Inserting Columns Between Existing Ones

When you map data to an Excel spreadsheet, it is possible to easily add new columns in between any of the existing columns, without redoing the existing mapping connections. To do this, right-click a cell in the Excel component, and select **Add Cell Before** from the context menu.

Note: The **Add Cell Before** command is available if the cell belongs to a row range (that is, it must be child of a "Row" or "Rows" item of an Excel component). Also, the option **Show separate items for columns** must be enabled in the "Select Range of Cells" dialog box (see Selecting Ranges of Cells 723).

For example, let's suppose you have created a mapping which writes data from an XML file to an Excel spreadsheet (such as **Sales_to_Excel.mfd** from the folder **<Documents>\Altova\MapForce2023\MapForceExamples**). This mapping outputs an Excel spreadsheet which consists of three columns (Month, West, and East), as illustrated below.

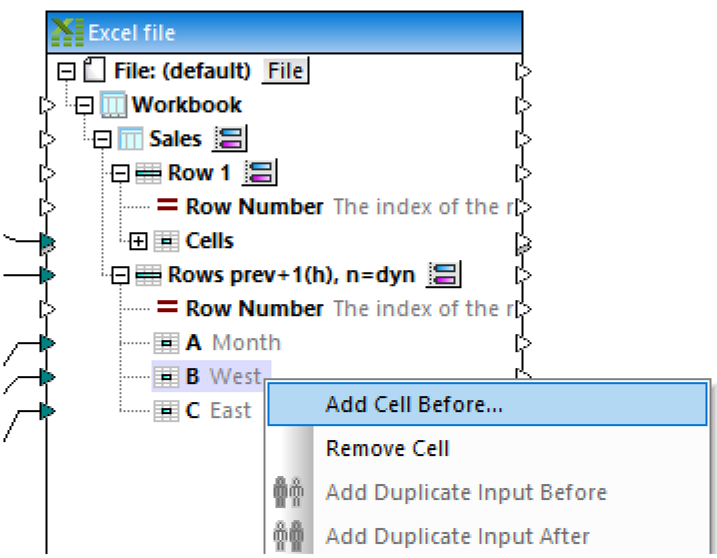


	A	B	C	D
2	Month	West	East	
3	Jan	110.4	75.3	
4	Feb	114.3	65.2	
5	Mar	134.2	86.1	
6	Apr	107.3	112.1	
7	May	114.4	93.8	
8	Jun	113.9	72.4	
9	Jul	89.4	67.4	
10	Aug	95.3	84.9	
11	Sep	107.2	99.5	
12	Oct	129.7	82.5	
13	Nov	137.1	101.9	
14	Dec	152.6	120.6	
15				

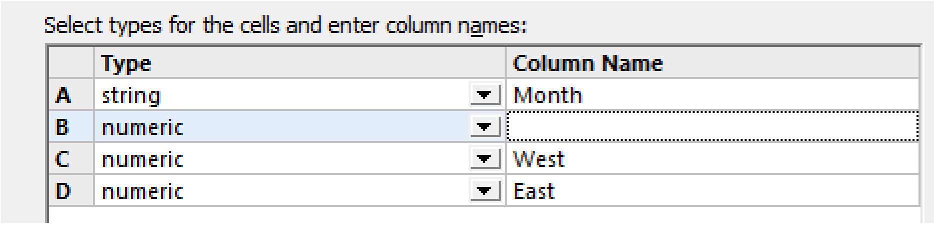
Output of mapping Sales_to_Excel.mfd

Assuming that you need to add a new column immediately before "West" (for example, "North"), this is possible without redoing any of the existing connections, as follows:

1. Right-click the cell B on the mapping, and select **Add Cell Before** from the context menu.

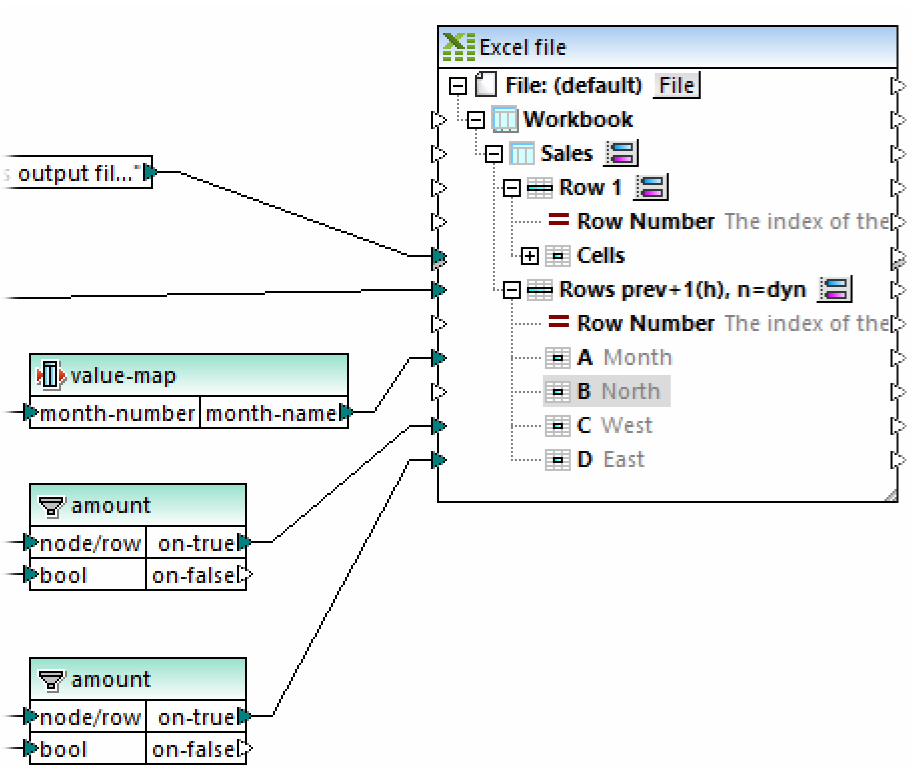


2. Optionally enter a column name (for example, "North").



3. Click OK.

The new column ("North", in this example) has now been added to the mapping, and it is now possible to draw a mapping connection to this column. All the existing connections are not affected by the change.



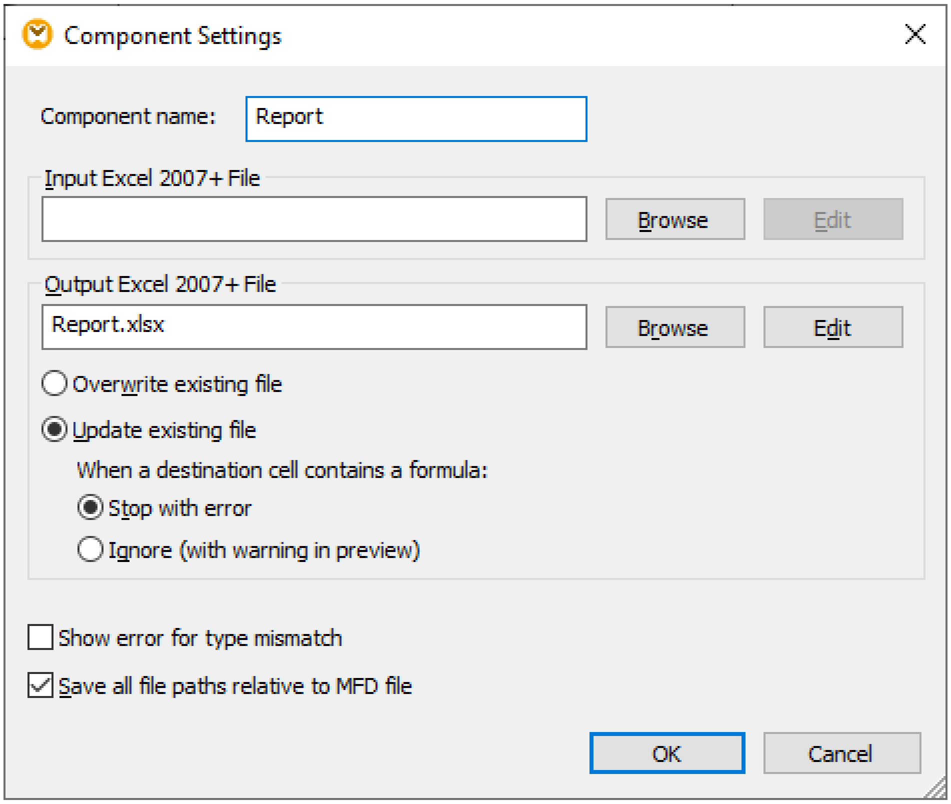
Sales_to_Excel.mfd mapping (modified)

Existing columns can also be removed in a similar fashion: right-click a cell on the mapping, and select **Remove Cell** from the context menu.

4.7.7 Excel 2007+ Component Settings

After you add an "Excel 2007+" component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component Settings dialog box in one of the following ways:

- On the **Component** menu, click **Properties** (this menu item becomes enabled when you select a component).
- Double-click the component header.
- Right-click the component header, and then click **Properties**.



Component Settings dialog box

The available settings are as follows.

Component name	The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change the name of the component, be aware of the following: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.
Input Excel 2007+ file	Specifies the Microsoft Excel file from which MapForce will read data. This field is meaningful in a source component, when

	MapForce uses it to read the Excel worksheet names and columns. To change the location of the file, click **Browse** and select the new file. To edit the file in Microsoft Excel, click **Edit**.
Output Excel 2007+ file	Specifies the name or path of the Microsoft Excel file to which MapForce will write data. This field is meaningful in a target component. To change the location of the file, click **Browse** and select the new file. To edit the file in Microsoft Excel, click **Edit**. If you do not specify an output file, MapForce generates the output to a file named **xlsx-mapforce.xlsx**. By default, this file is generated in the same folder as the .mfd file, unless you configured a different path.
Overwrite existing file	When this option is selected, the file specified in the **Output Excel 2007+** text box (see above) will be generated when the mapping runs. The generated Excel file does not contain any formatting. If a file with the same name already exists in the output directory, it will be overwritten when the mapping transformation runs. This option is supported when the transformation language is C#, Java or BUILT-IN.
Update existing file	When this option is selected, the file specified in the **Output Excel 2007+** text box (see above) will be updated selectively (not overwritten) when the mapping runs. Specifically, any designated worksheets 720, rows 722 and cells 723 from the existing file will be replaced with data from the mapping. Any formatting in the existing file will be preserved as is. Note that updating existing files is possible only for cells that do not contain formulas. Select the **Stop on error option** to stop mapping execution whenever a cell with formulas is encountered. Select the **Ignore (with warning in preview)** option to ignore such cells while still displaying a warning in the Messages window. This option is supported only when the transformation language is set to BUILT-IN.
Show error for type mismatch	Enables error messages in the Messages window, when mismatches occur between a data type declared in the component and data in the input .xlsx file. For example, let's assume that this setting is enabled, and you declared a column as numeric in the *Select range of cells* dialog

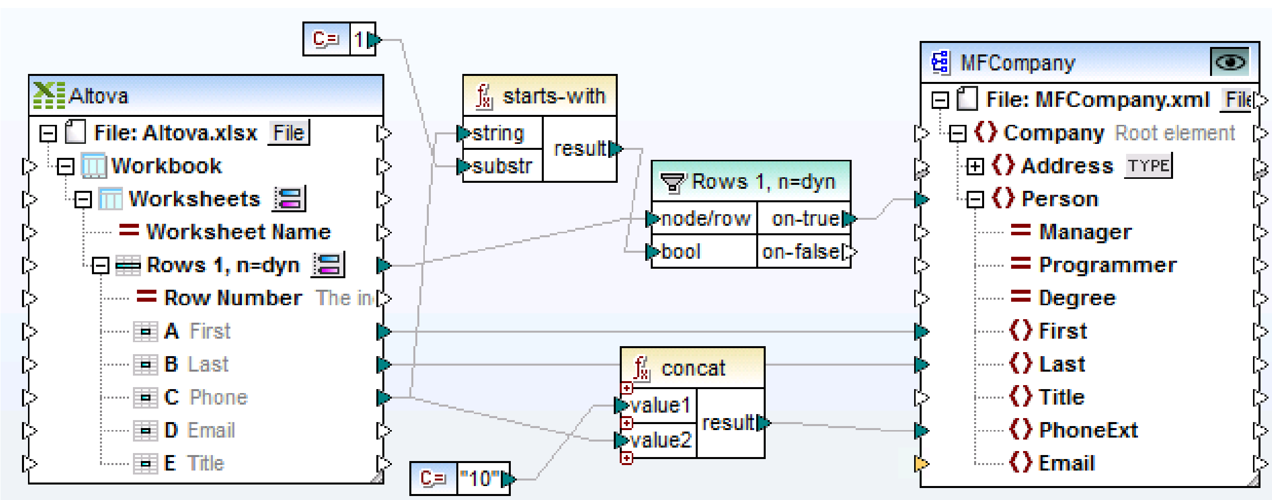
	box. If the Excel file contains text data in this column, an error message is shown in the Messages window after you connect this column to some output and preview the output.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative Paths on a Component 85.

4.7.8 Example: Mapping Excel 2007+ to XML

The mapping file used in the following example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Excel-mapping.mfd**. When you open the sample file, you will notice that it contains three distinct mapping transformations. The top two transformations are discussed in this section.

The aim of the first mapping is to do the following:

- Select from the source Excel workbook only people whose phone extension (column C of the workbook) starts with a "1".
- Add the prefix "10" to the original number, and write it to a target XML file, along with the First and Last names of the respective persons.



Excel-mapping.mfd (sample 1)

The mapping is configured as follows:

- **Altova.xlsx** is the source Excel 2007+ workbook. Columns A and B supply the First and Last names respectively. Column C supplies the phone extension number.
- Both worksheets of the workbook are shown as one node in the component (in other words, the **Show worksheets by name** option is disabled). This is indicated by the "Worksheets" node under the Workbook item.
- The `starts-with` function checks if the phone extension (col. C) starts with a "1", and if the result is true then those records are forwarded by the filter component.

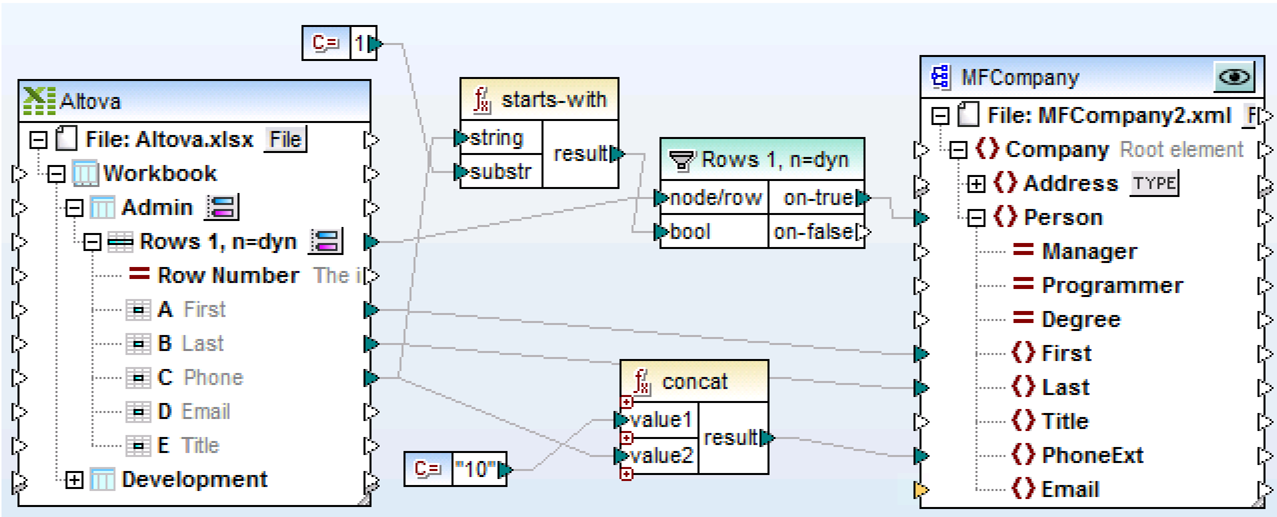
- The **concat** filter adds the prefix "10" to each of the telephone extensions and writes it to the PhoneExt item.
- MFCompany.xsd is the target component and contains the filtered person details when data is output.

The result of the mapping is that four persons have been mapped to the XML file with their details.

```
<?xml version="1.0" encoding="UTF-8"?>
<Company xsi:schemaLocation="http://my-company.com/namespace
  <Person>
    <First>Steve</First>
    <Last>Meier</Last>
    <PhoneExt>10114</PhoneExt>
  </Person>
  <Person>
    <First>Max</First>
    <Last>Nafta</Last>
    <PhoneExt>10122</PhoneExt>
  </Person>
  <Person>
    <First>Carl</First>
    <Last>Franken</Last>
    <PhoneExt>10147</PhoneExt>
  </Person>
  <Person>
    <First>Mark</First>
    <Last>Redgreen</Last>
    <PhoneExt>10152</PhoneExt>
  </Person>
</Company>
```

The second mapping is identical with the first one, except that worksheets have been individually enabled using the **Show worksheets by name** option. This mapping is configured as follows:

- The **Admin** and **Development** worksheets are both visible under the Workbook item.
- Connectors have only been defined from the **Admin** worksheet to the target component.



Excel-mapping.mfd (Sample 2)

The result of the mapping is that only two persons have been mapped to the XML file with their details.

```
<?xml version="1.0" encoding="UTF-8"?>
<Company xsi:schemaLocation="http://my-company.com/namespace
  <Person>
    <First>Steve</First>
    <Last>Meier</Last>
    <PhoneExt>10114</PhoneExt>
  </Person>
  <Person>
    <First>Max</First>
    <Last>Nafta</Last>
    <PhoneExt>10122</PhoneExt>
  </Person>
</Company>
```

4.7.9 Example: Convert Excel Rows to XML Files

Excel files may consist of multiple worksheets that have a different structure each, meaning that the number and order of columns could be different. On the other hand, there are also Excel files where all worksheets have identical structure and only the name and data of the worksheet is different.

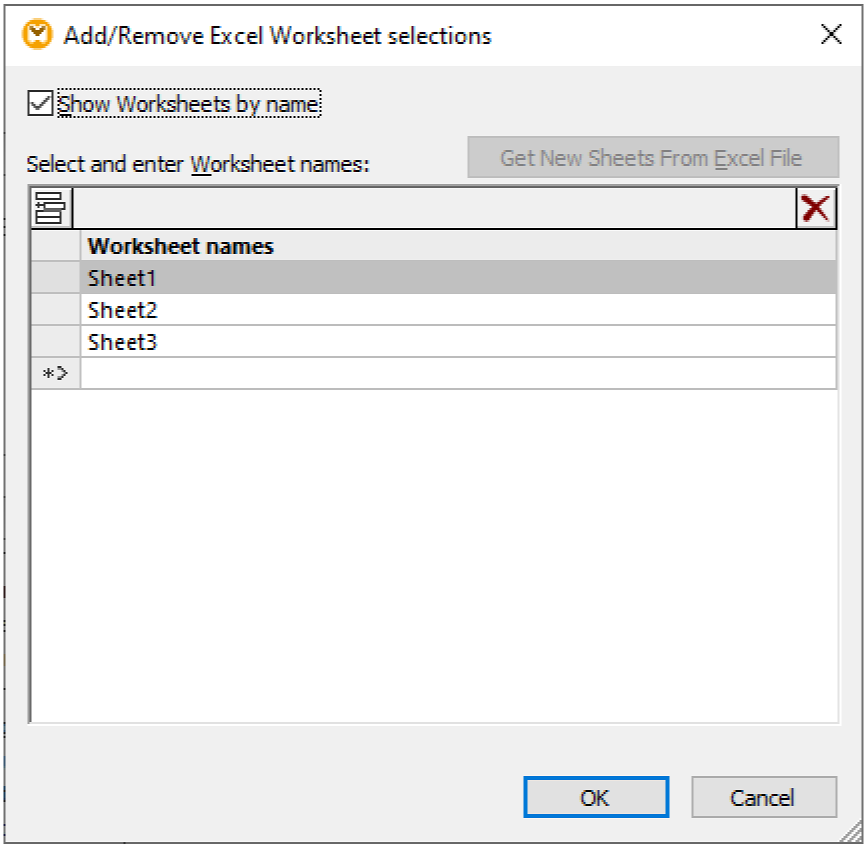
With MapForce, you can configure a mapping to flexibly extract data from Excel worksheets in either of the scenarios above, namely:

1. You can show each individual Excel worksheet as a distinct structure on the mapping. This approach is useful when each Excel worksheet has a different structure. You have the flexibility to map data originating from any cell of any worksheet in your source Excel file.
2. You can treat all the Excel worksheets in the file as if they were a single worksheet. This approach is useful only if all worksheets have the same structure; it has the advantage that you draw fewer

connections on the mapping. Since the structure of each worksheet is assumed to be the same, you draw connections from each column only once, regardless of the worksheet where it appears.

To configure an Excel component for either of the two approaches described above:

1. Add the Excel file to the mapping using the **Insert | Excel 2007+ File** menu command, see also Adding Excel 2007+ Files as Mapping Components 717.
2. Click the button adjacent to the first worksheet. The following dialog box appears:

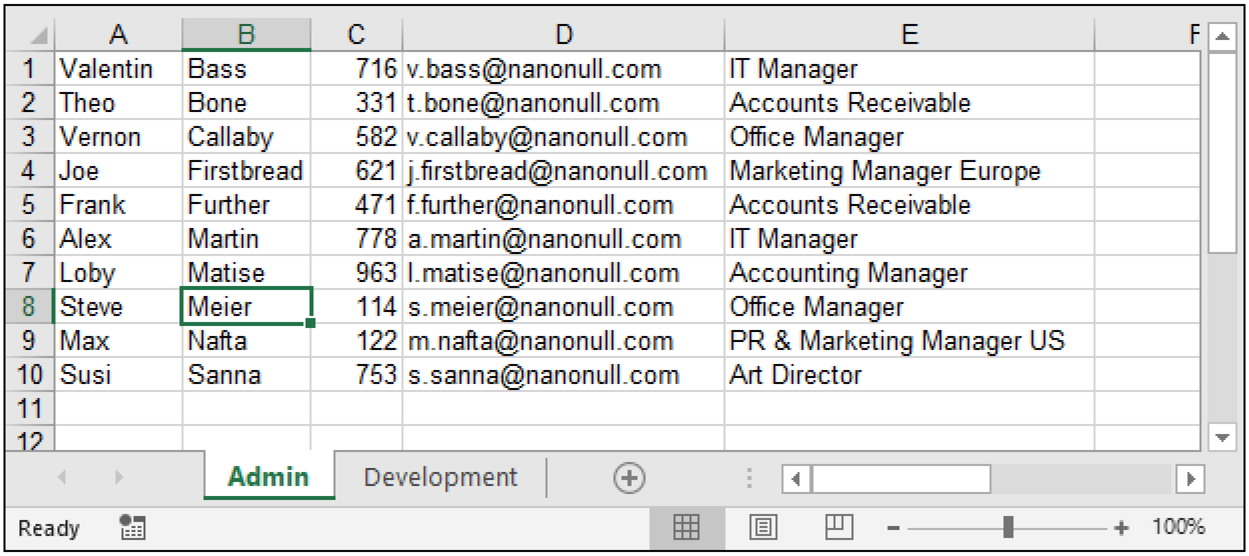


3. Do one of the following:

   a. To show the structure of each worksheet separately, make sure that the check box **Show Worksheets by name** is selected. This is the default option.
   b. To treat all worksheets as a single mappable component, clear the check box **Show Worksheets by name**.

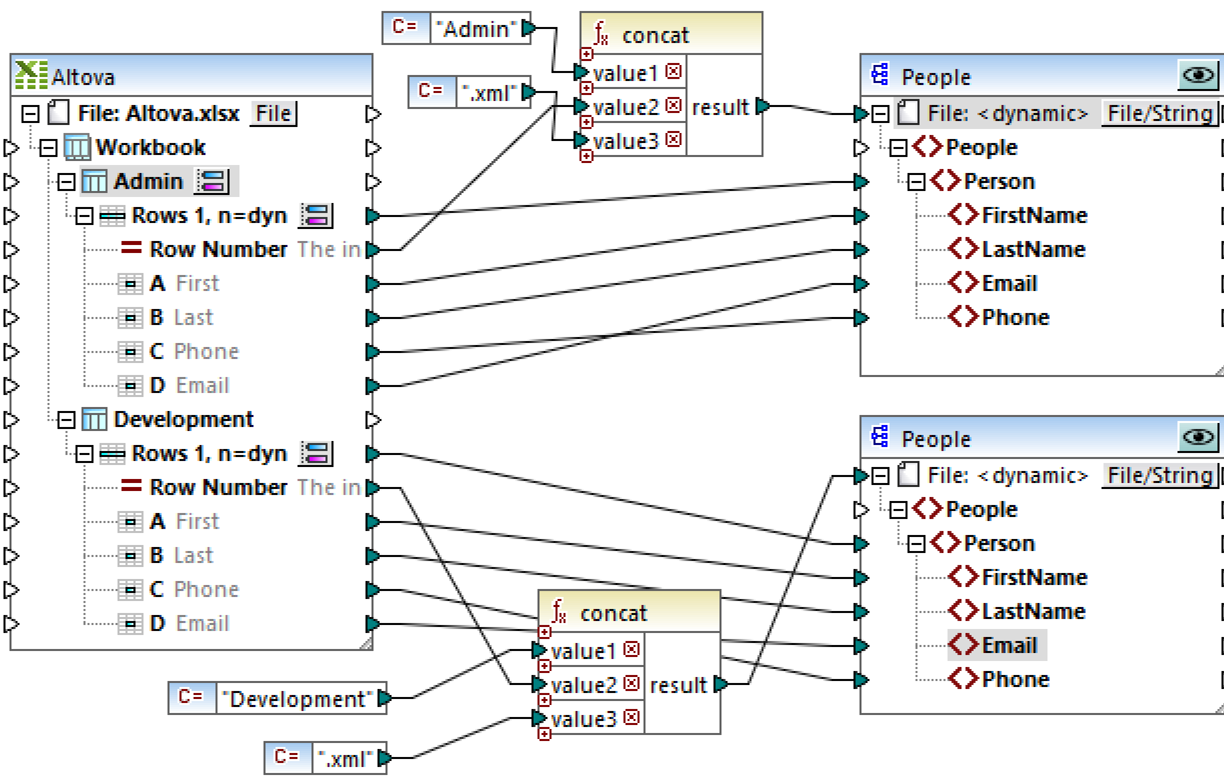
A demo mapping design file that illustrates each of the two approaches described above is available at the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ConvertExcelRows.mfd. This mapping illustrates two different strategies of reading data from an Excel file. Since this mapping produces multiple unrelated outputs, the **Preview** button appears in the title bar of each target component. Remember to click this button before clicking the **Output** tab, in order to preview the respective output file.

The source Excel file is called **Altova.xlsx** and it consists of two worksheets: "Admin" and "Development". Note that the structure of each worksheet is identical, only the data in each worksheet differs (which is why more than one data access strategy is possible).



	A	B	C	D	E	F
1	Valentin	Bass	716	v.bass@nanonull.com	IT Manager	
2	Theo	Bone	331	t.bone@nanonull.com	Accounts Receivable	
3	Vernon	Callaby	582	v.callaby@nanonull.com	Office Manager	
4	Joe	Firstbread	621	j.firstbread@nanonull.com	Marketing Manager Europe	
5	Frank	Further	471	f.further@nanonull.com	Accounts Receivable	
6	Alex	Martin	778	a.martin@nanonull.com	IT Manager	
7	Loby	Matise	963	l.matise@nanonull.com	Accounting Manager	
8	Steve	Meier	114	s.meier@nanonull.com	Office Manager	
9	Max	Nafta	122	m.nafta@nanonull.com	PR & Marketing Manager US	
10	Susi	Sanna	753	s.sanna@nanonull.com	Art Director	
11						
12						

The upper part of the mapping illustrates reading data from the Excel file by treating each worksheet as a separate structure. If you click the button adjacent to the **Admin** item, you will notice that the check box **Show Worksheets by name** was selected. This has the effect that both worksheets, "Admin" and "Development", appear explicitly on the source component, as children of **Workbook**.



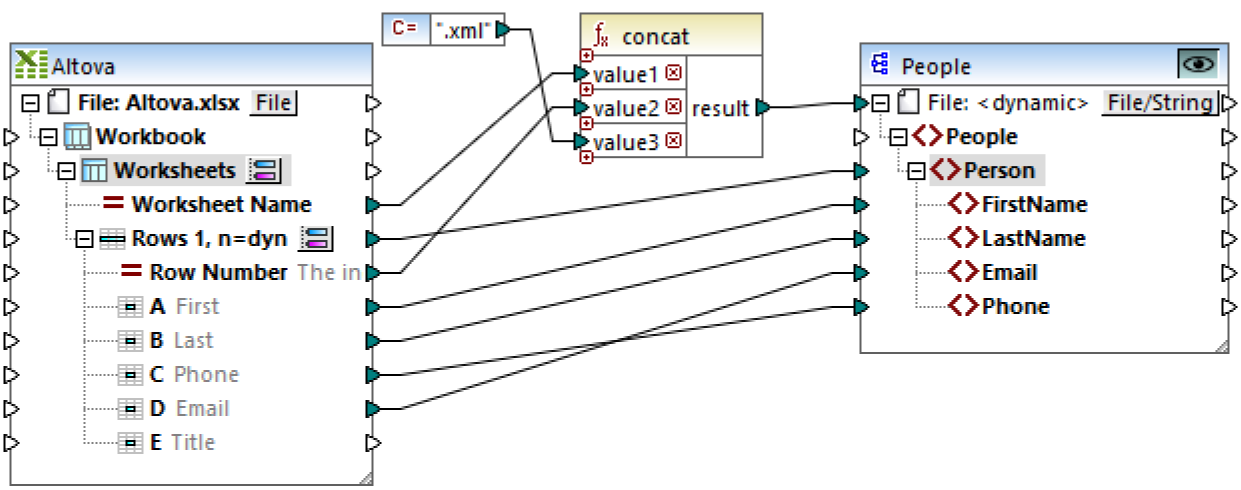
ConvertExcelRows.mfd (part 1)

The mapping illustrated above converts data from both Excel worksheets to XML files. A separate XML file must be generated for each person record from each row in each source worksheet. The file name must identify the department name and have a numeric ID. For example, the file **Admin1.xml** will store the details of the first person (row) in the "Admin" worksheet, and so on. This mapping goal was accomplished as follows:

1. The **concat** function produces the file name for each target XML. It joins (in this order) the following: the name of each department, the Excel row number, and the string ".xml".
2. The connection from the result of the **concat** function to the **File: <dynamic>** target item has the effect that a new file is generated at mapping runtime for each item in the incoming sequence of items. In this example, the incoming sequence contains all the rows of the source Excel worksheet; therefore, a new XML file will be generated for each Excel row.

As a side note, to configure a target component to generate file names dynamically, click the **File/String** button, and select the option **Use Dynamic File Names Supplied by the Mapping**. For more information, see Processing Multiple Input or Output Files Dynamically 1313.

The second part of the **ConvertExcelRows.mfd** mapping is illustrated below. This part of the mapping achieves exactly the same result as part 1, but in a more concise way. Notice that the mapping connections are significantly reduced.



ConvertExcelRows.mfd (part 2)

The mapping logic is almost identical with the one described previously, namely:

1. For each row in each source worksheet, one **Person** item in the target will be created.
2. The **concat** function generates the name of each target file.
3. The target component was switched to generate file names dynamically.

The only difference is that all the worksheets in the source Excel file are iterated over implicitly—there is no need to show each worksheet's structure separately and draw mapping connections for each. The reason is that the source Excel component has been configured to treat all worksheets as a single one—you can click the button adjacent to the **Worksheets** item and observe that the **Show Worksheets by name check** box is not selected.

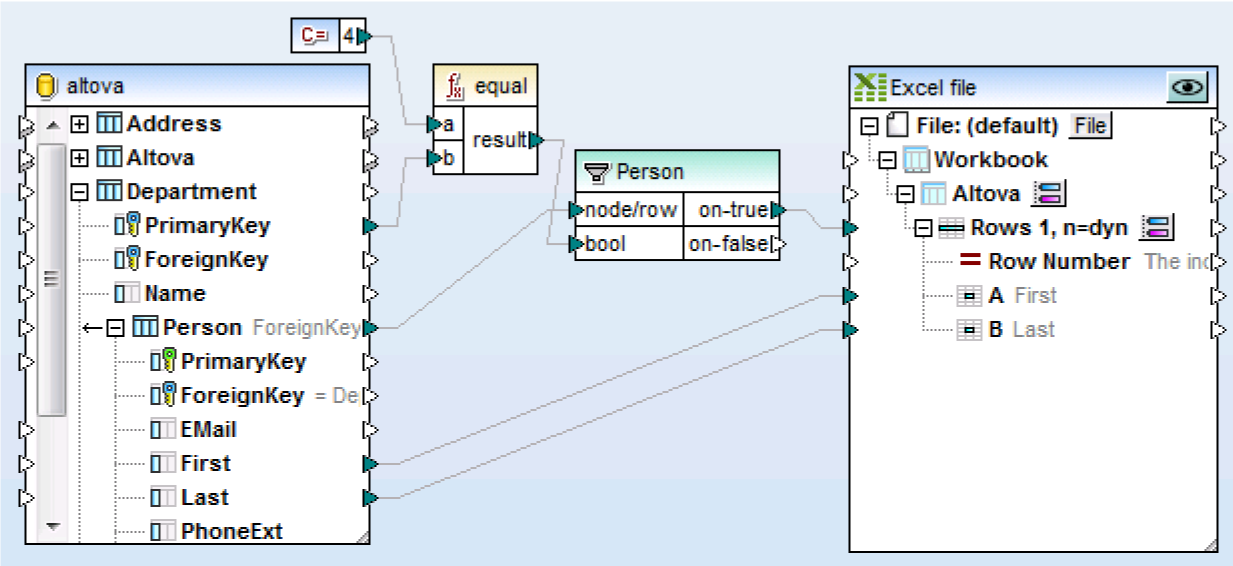
In conclusion, in this topic you have seen the differences between mapping data from individual Excel worksheets as opposed to treating all worksheets as one. Just keep in mind that the second approach requires and implies that the structure of all worksheets is the same. In this example, the assumption is that each first column in each worksheet stores the person's first name, each second column stores the person's last name, and so on.

4.7.10 Example: Mapping Database Data to Excel 2007+

The mapping file used in this example is available as **Excel-mapping.mfd** in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. The third mapping of the three is discussed here.

The aim of the mapping is as follows:

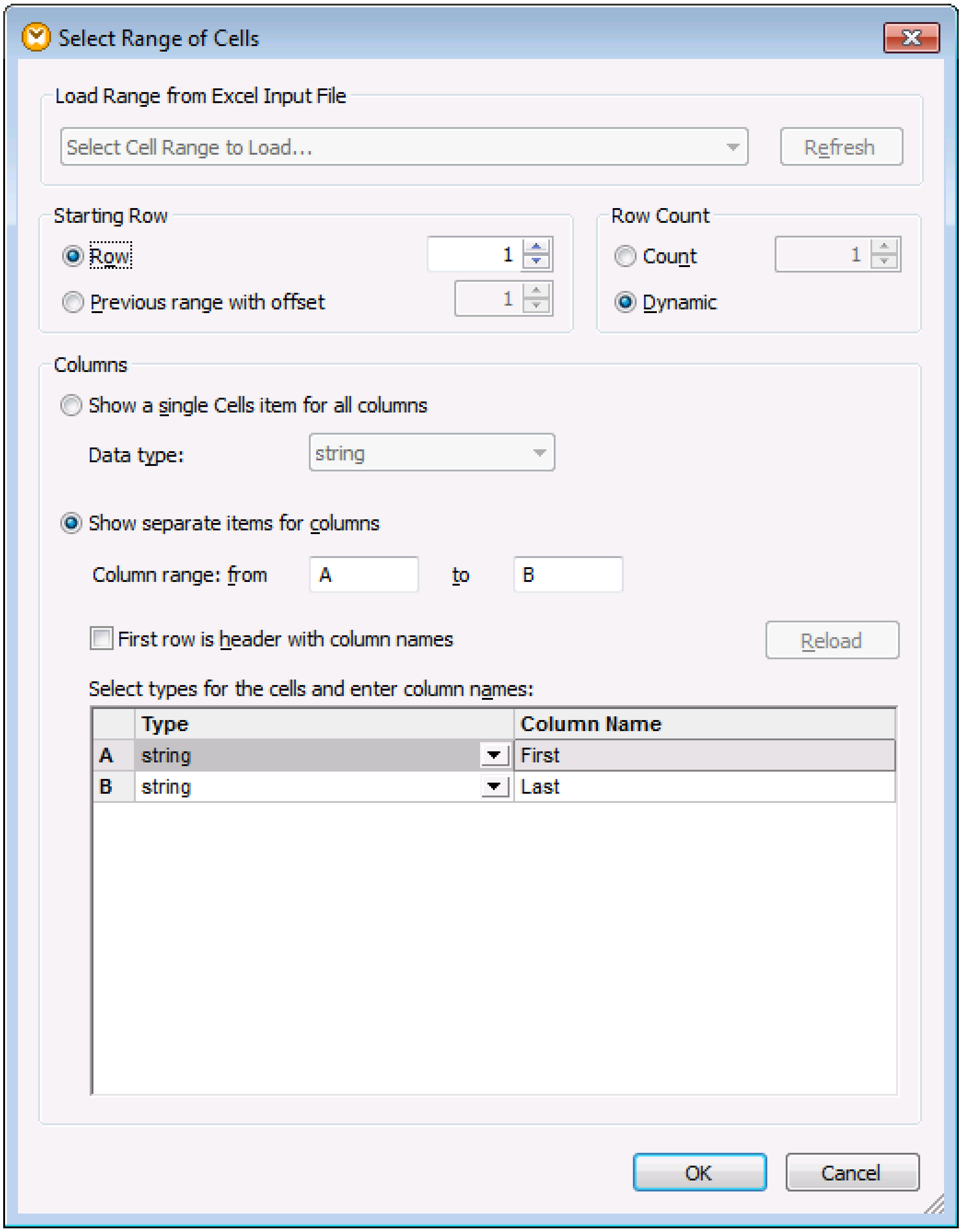
- Extract from the "altova" database only persons whose department primary key is equal to 4 (that is, those who are in the IT department).
- Write the extracted records to a default Excel 2007+ component.



Excel-mapping.mfd (sample 3)

The mapping is configured as follows:

1. The database "altova" was added to the mapping area from the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder, using the **Insert | Database** menu command, and following the wizard for a Microsoft Access database.
2. The default Excel 2007+ component was added using the **Insert | Excel 2007+ file** menu command, and then skipping the option to supply a sample file.
3. The first worksheet (Sheet1) was renamed by clicking the  button adjacent to it and then entering "Altova" as worksheet name.
4. The **Rows 1, n=dyn** range was configured by clicking the  button adjacent to it. The cell range options were defined as follows:



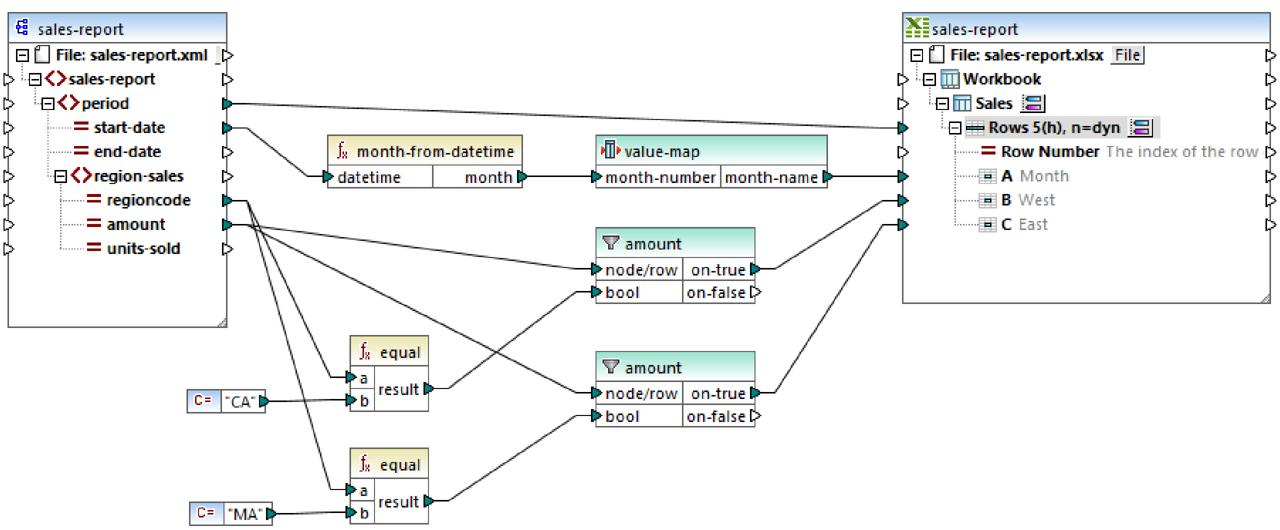
5. Other options were defined as follows:

- o The value of the **PrimaryKey** is compared to the value "4", supplied by the **Constant** component, using the **`equal`** function.
- o The **filter** component passes on the First and Last fields if the Boolean condition is true (that is, if the department primary key is "4").
- o The **on-true** item is connected to the **Rows 1, n=dyn** item in the Excel file.

The result of the mapping is that four persons of the IT department are shown in the Excel workbook.

4.7.11 Example: Updating Existing Excel Sheets

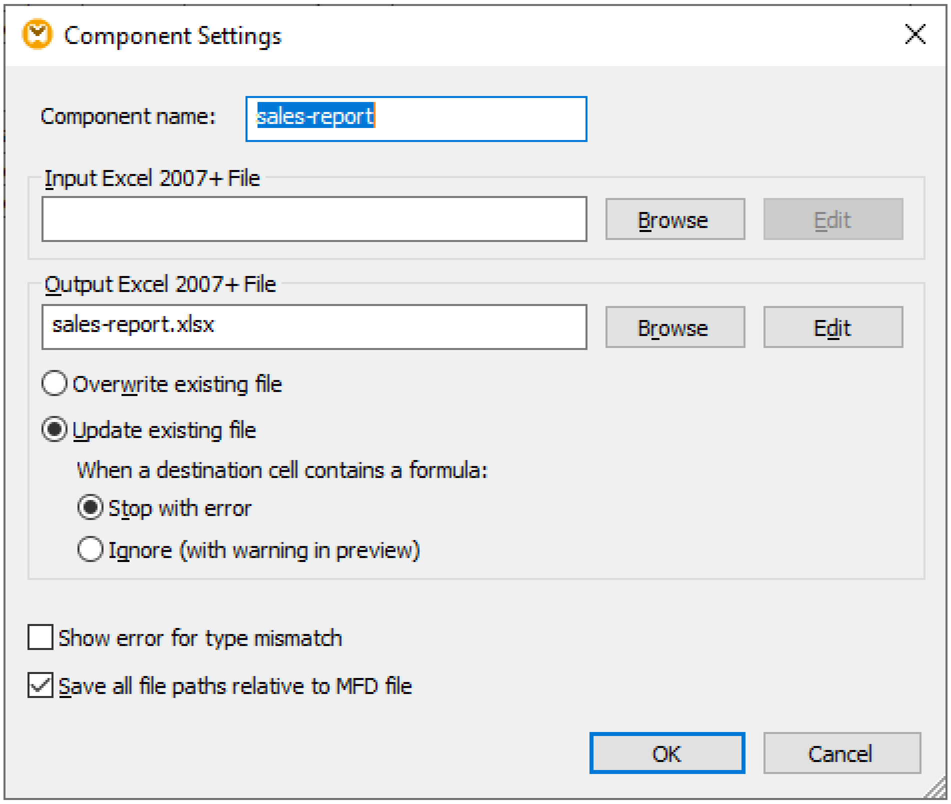
If you choose to *generate* Excel sheets with MapForce, the generated sheets do not contain any formatting, only data. However, it is also possible to *update* existing Excel files and preserve their formatting (as opposed to *generating* them), which is the goal of this example. You can find the mapping described in this example at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Sales_to_Excel.mfd**. As illustrated below, this mapping reads data from a file in XML format and writes data to an existing Excel file (**sales-report.xlsx**). Specifically, it updates a sheet called "Sales", starting from the fifth row, in columns A, B, and C.



Sales_to_Excel.mfd

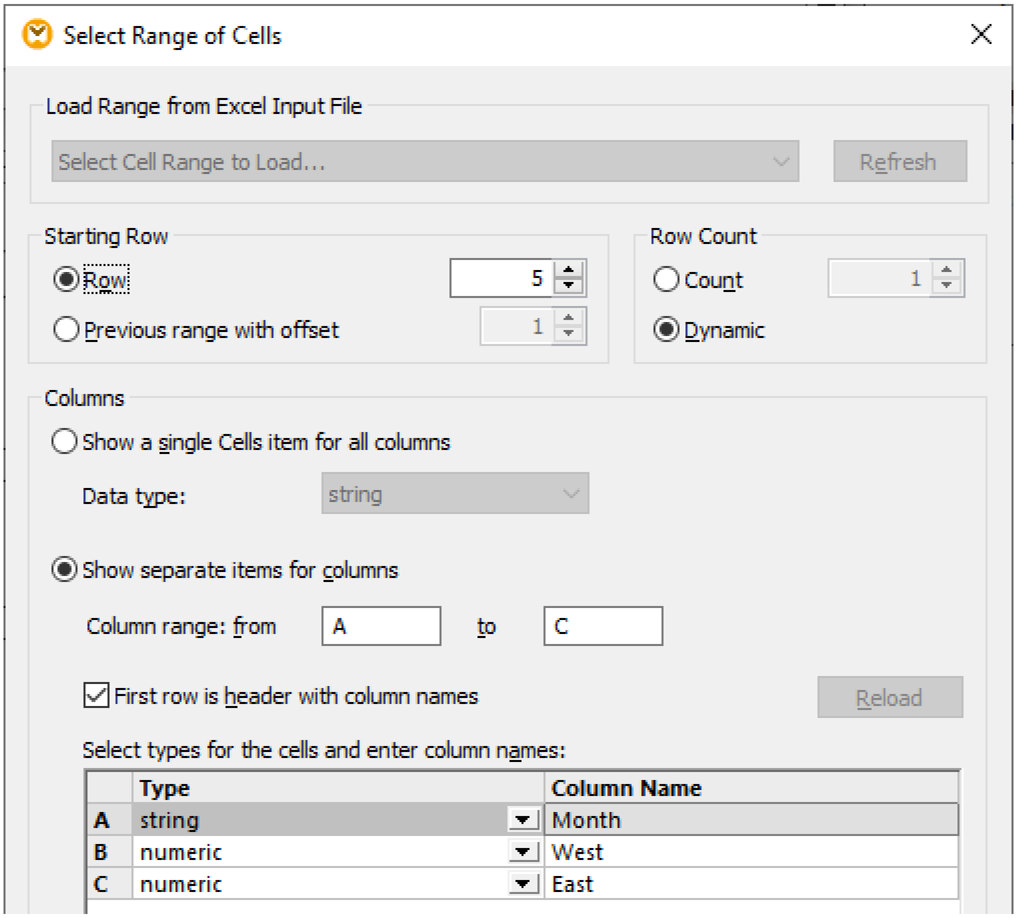
The target component was configured to update the existing file as follows:

1. Set the transformation language to BUILT-IN.
2. Right-click the title bar of the target component, and select **Properties**. The **Component Settings** dialog box opens.
3. In the **Output Excel File** text box, enter the path to the existing Excel file. In this example, the file is in the same folder, so the path is relative, and the check box **Save all file paths relative to MFD** file is selected.
4. Select the **Update existing file** option.

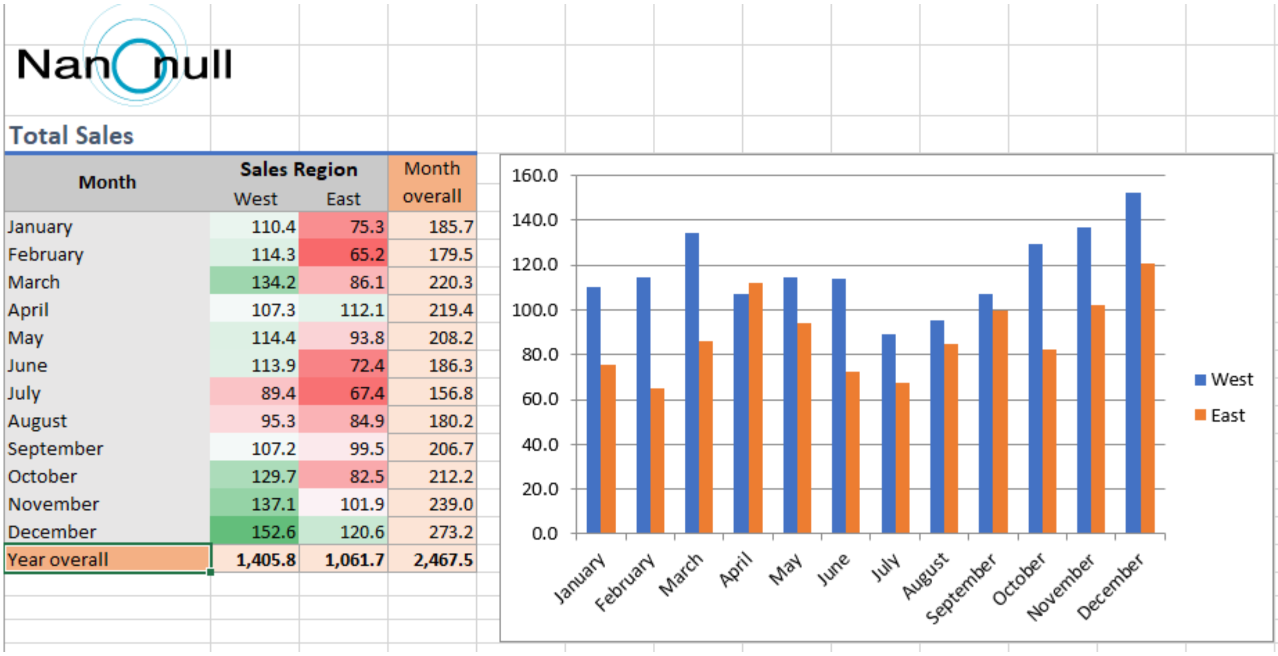


The target worksheet was selected by clicking the button adjacent to the "Sales" node ("Sales" here is the name of the worksheet).

Finally, one can select the range of cells/rows to write to by clicking the button adjacent to the "Rows" node. According to the configuration below, the mapping will update the worksheet starting with row 5, and create as many rows as there are items in the mapped source data. The columns A, B, and C were given distinct names so that they can be easily mapped to.



When you click the **Output** tab to preview the mapping result, the existing worksheet is updated and all existing formatting is preserved. The existing chart is also updated as a consequence (because the source cells used by the chart changed their values).



Month	Sales Region West	Sales Region East	Month overall
January	110.4	75.3	185.7
February	114.3	65.2	179.5
March	134.2	86.1	220.3
April	107.3	112.1	219.4
May	114.4	93.8	208.2
June	113.9	72.4	186.3
July	89.4	67.4	156.8
August	95.3	84.9	180.2
September	107.2	99.5	206.7
October	129.7	82.5	212.2
November	137.1	101.9	239.0
December	152.6	120.6	273.2
Year overall	1,405.8	1,061.7	2,467.5

Mapping result

Note that previewing the mapping does not actually save changes to the Excel file. To save the file displayed in the **Output** pane to the disk, select the menu command **Output | Save Output File**, or click the **Save generated output** toolbar button. Alternatively, you might want to enable the option **Write directly to final output files** from **Tools | Options** 1473 **| General**. The latter option should be exercised with caution, because it overwrites any existing files without requesting further confirmation.

To automate running mappings such as this one and eliminate the need for button clicks, you can prepare and deploy the mapping to MapForce Server, as described in Automating Mappings and MapForce 1369.

4.8 XBRL

XBRL (Extensible Business Reporting Language) is an open international standard that enables exchange of business information. XBRL is used by governments, financial institutions, and businesses, primarily for financial and regulatory reporting. XBRL is managed by XBRL International.

A company or organization may have different business needs in conjunction with XBRL (for example, XBRL validation, creation of XBRL data, creation of new XBRL taxonomies, export of XBRL data to other formats). Various Altova products make it possible to accomplish XBRL-related tasks as follows:

- *XMLSpy* edits or creates new taxonomies and generates XBRL reports based on XSLT transformation files (XSLT transformation files can be created in StyleVision).
- *StyleVision* creates taxonomy stylesheets/templates, allowing you to generate XBRL reports.
- *MapForce* maps data to or from XBRL instance files. You can map XBRL data to or from other formats supported by MapForce (such as Microsoft Excel 2007 and later, databases, or CSV files). For example, you can filter specific data from existing XBRL instance documents, or create XBRL instances from data extracted from a database.
- *RaptorXML+XBRL Server* is a cross-platform XBRL processor which validates XBRL instances and taxonomies.
- *Altova XBRL add-ins for Excel* enable businesses to prepare and validate XBRL reports using Microsoft Excel.

While a full introduction to XBRL is beyond the scope of this documentation, the following terms are frequently used in conjunction with XBRL:

- *XBRL Taxonomy*. A taxonomy is like a dictionary of all concepts that may occur in the XBRL report. It also define rules for organizing the concepts hierarchically.
- *XBRL Instance*. The XBRL instance is a document which contains the so-called "facts" to be reported. The actual definition of concepts and hierarchical relationships between them is provided by the taxonomy.

In order to map data to or from XBRL instance files in MapForce, the underlying XBRL taxonomy is required. The taxonomy consists of a file archive (usually .zip) which is provided by the organization or entity to which XBRL data is submitted. The following are examples of taxonomies that can be used for data mapping with MapForce:

- US-GAAP (Generally Accepted Accounting Principles), published by the Financial Accounting Standards Board (http://www.fasb.org)
- IFRS (International Financial Reporting Standards), published by the IFRS (International Financial Reporting Standards) Foundation (http://www.ifrs.org)
- COREP / FINREP (Common Reporting and Financial Reporting), published by the European Banking Authority (http://www.eba.europa.eu)

You can install or uninstall XBRL taxonomies across all XBRL-enabled Altova products from the XBRL Taxonomy Manager 751.

To resolve URIs in XBRL documents, MapForce uses a catalog mechanism that maps URIs to files on the local system. This is in large part due to the size of the taxonomies and the fact that they contain thousands of files. Accessing the taxonomy files over the Internet would result in extremely slow performance, even if their issuing organizations served them that way. Note that the catalog mechanism relies on XBRL taxonomy packages installed locally on your computer. By default, no XBRL taxonomy packages are installed. Consequently, when you open a mapping that contains XBRL components, MapForce may display a dialog box asking you to install the required XBRL taxonomy packages. In addition, you can always view and manage all the installed XBRL taxonomies from XBRL Taxonomy Manager [751].

Supported XBRL standards

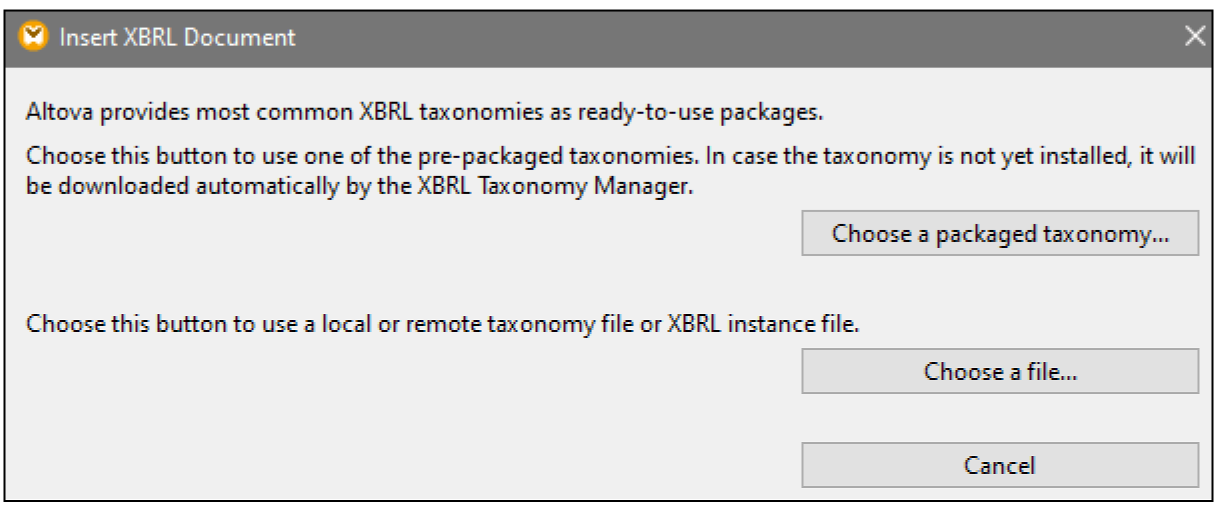
MapForce supports the following XBRL-related standards:

- XBRL Specification 2.1
- XBRL Dimensions Specification 1.0
- Table Linkbase Specification 1.0 (see Working with XBRL Tables [782])
- Taxonomy Packages 1.0 (see XBRL Taxonomy Packages [1478])
- Inline XBRL 1.0 and 1.1, including Transformation Registry 3 (only for reading data). See also Reading Data from Inline XBRL [750].
- XII Transformation Registry 5 Specification

4.8.1 Add XBRL Files

This topic explains how to add an XBRL document to a mapping. To insert an XBRL document, select the menu command **Insert | XBRL Document** or the toolbar command. The dialog box (*see screenshot below*) will prompt you to choose between a packaged taxonomy and a local or remote file.

If you choose to add a prepackaged taxonomy, you will be prompted to select an entry point. If the taxonomy you wish to use is not yet installed, it will automatically be downloaded by the XBRL Taxonomy Manager [751]. If you want to import a local or remote file, you can choose a taxonomy file or an XBRL instance file.



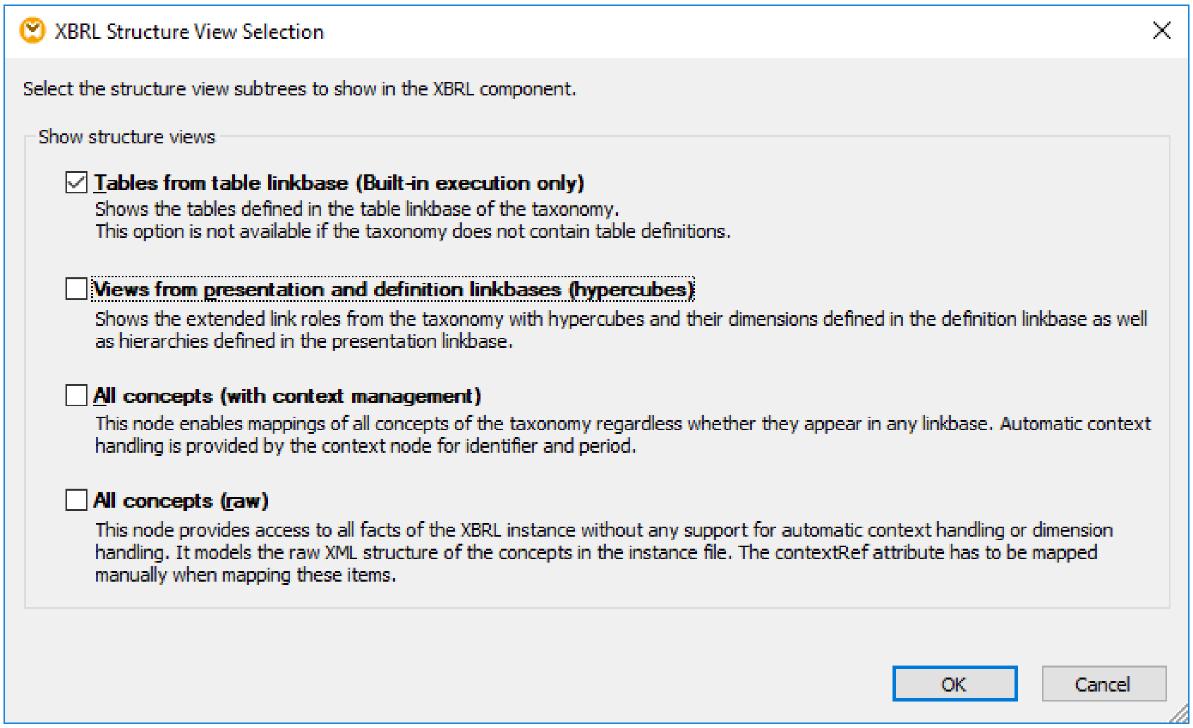
For information about items that can appear in an XBRL component, including their conventional graphical representation, see XBRL Component Items [747].

4.8.2 Structure Views

MapForce provides various ways to display the XBRL structure inside a component, by means of so-called "structure views". You can select the XBRL structure views either when you add an XBRL document to the mapping area (see Adding XBRL Files as Mapping Components[745]), or at any time later.

To select the structure views to be shown on the XBRL component:

1. On the root element of the XBRL component, click the **Show Context Menu** ( ) button, and then **Select Structure Views**.



2. Select one or more structure views, and then click **OK**.

The available structure views are as follows.

Structure View	Description
Tables from table linkbase	Shows the tables defined in the table linkbase of the taxonomy. This option is disabled if the taxonomy does not contain table definitions. This view is conditional. To proceed, select either this check box, or the **Views from presentation and definition linkbases** check box.
Views from presentation and definition linkbases	Shows data from the presentation and definition linkbases. The presentation linkbase includes hierarchies and the definition linkbase includes extended link roles with hypercubes and dimensions.

	This view is conditional. To proceed, select either this check box, or the **Table from table linkbase** check box.
All concepts (with context management)	Shows the additional node "All concepts" in the hierarchical structure. This node contains the hypercube "Dimensionless" which enables mappings of all concepts of the taxonomy regardless whether they are reported within hypercubes by means of the two default dimensions **identifier** and **period**. Automatic context handling is provided by the **context** node which contains all XBRL concepts of the taxonomy as children. Abstract items are not shown. This view is optional to display.
All concepts (raw)	Shows the additional node "All concepts (raw)" which provides access to all facts of the XBRL instance without any support for automatic context handling or dimension handling. It models the raw XML structure of the concepts in the instance file. The `contextRef` attribute has to be mapped manually when mapping these items. This view is optional to display.

Notes

- When reading data from an XBRL component (that is, if the XBRL component is a source component), you can choose any combination of structure views from the context menu . However, when writing data to XBRL, if the **Tables from table linkbase** view is selected, it is not possible, for technical reasons, to write data to any other view except this one.
- The **Tables from table linkbase** view requires that BUILT-IN[27] be set as a transformation language.

4.8.3 XBRL Component Items

The following table gives an overview of the component items (nodes) which are typically part of an XBRL component in MapForce. Note that some of the nodes are only available when you selected the relevant structure view (see Selecting Structure Views[746]). For example, the nodes specific to the table linkbase are not visible if you choose to display XBRL data only from the presentation and definition linkbases.

Component Item	Icon	Description
Abstract item		Abstract items are used to organize related facts. They are either defined within the presentation linkbase or within domain member networks of definition linkbases. They do not actually exist in the XBRL instance file; they allow grouping of facts in an intuitive way.
Breakdown		As defined in the Table Linkbase Specification 1.0, §5.4.

Component Item	Icon	Description
Context		A context node is a container for all related business facts. Context nodes inside hypercubes manage their context elements and dimensions automatically. The `xbrli:context` node which is a child of the XBRL root element is used for manual dimension handling.
Dimension		Dimensions (XBRL Dimensions Specification 1.0, §2.5) are used to structure contextual information for business facts. Dimensions are defined in the taxonomy within a hypercube.
Explicit member		An explicit member is a member of a dimension which is defined by an enumeration of `QName` values (see also Showing Dimensions in a Component[776]).
Explicit member value		The value of an explicit member.
Fact		Facts (XBRL Specification 2.1, §1.4) are the values of the XBRL items. They can be of the following types: • Monetary items ( ) • String items ( ) • Numeric items ( ) • General items ( ) • Shares items ( )
Filing indicators	Filing Indicators	The `Filing Indicators` node appears only in target XBRL components with EBA-related taxonomies. The **Filing Indicators** button allows you to choose how filing indicators are to be generated in the output. For details, see XBRL Tables[783].
Footnote		Footnotes allow you to assign additional text information to facts.
Hypercube		Hypercubes (XBRL Dimensions Specification 1.0, §2.2) use information from the definition linkbase and the presentation linkbase to hierarchically structure dimensions, contexts and related XBRL concepts. See also Working with XBRL Hypercubes[774].
Root element		The `xbrli:xbrl` element is instance root element of every XBRL component.

Component Item	Icon	Description
Rule node		As defined in the Table Linkbase Specification 1.0, §6.6.
Structural node		As defined in the Table Linkbase Specification 1.0, §5.5.
Table		As defined in the Table Linkbase Specification 1.0, §5.1. MapForce displays the table structures in a hierarchy (see Working with XBRL Tables[782]).
Table set		As defined in the Table Linkbase Specification 1.0, §5.2.
Tuple		Tuples (XBRL Specification 2.1, §1.4) are complex elements containing facts or other tuples as members.
Typed member		Typed members are members that are defined by an XML schema element (see https://www.xbrl.org/specification/dimensions/rec-2012-01-25/dimensions-rec-2006-09-18+corrected-errata-2012-01-25-clean.html#sec-typed-dimensions).
Unit		The unit element **xbrli:unit** contains units to which XBRL items refer. It is mandatory to define a value for the `xbrl:unit` element when mapping data. For example, for dollars, `UnitID` is `usd` and `Measure` is `iso4217:USD`. See also Working with XBRL Defaults[770].
View		Views represent extended link roles from the definition and presentation linkbase of an XBRL taxonomy.

Note that the icon is generic and is used with a variety of node types, including the XBRL root element.

Additionally, the following graphical user interface elements can appear on XBRL components if required by the context.

GUI element	Icon	Description
Additional information		This icon accompanies items that have an associated message (such as errors). Click on the icon to display additional information in the Messages pane.
Error		This icon indicates an error message.

GUI element	Icon	Description
Show Context menu		This icon accompanies items that have additional context menu options available for selection.
Unknown		This icon accompanies items that could not be resolved by MapForce due to invalid namespace references, or when the referenced item does not exist in the taxonomy.

4.8.4 Read Data from Inline XBRL

Some regulators that work with XBRL may provide (or accept for submission) information in inline XBRL format, as an alternative (or in addition) to plain XBRL. Inline XBRL, or iXBRL, refers to machine-readable XBRL content embedded into HTML documents. This makes it possible to display XBRL data in a human-readable format in a Web browser. For the list of specifications related to iXBRL, see https://specifications.xbrl.org/spec-group-index-inline-xbrl.html.

In MapForce, files that contain inline XBRL (such as .htm, .html, or .xhtml files) can be added to the mapping as a data source. That is, MapForce can read data from Inline XBRL instance files for the purpose of mapping it to any supported target format. Note that writing data to inline XBRL instances is not supported.

Prerequisites

- The mapping transformation language must be set to BUILT-IN [27]; other languages are not supported. With the BUILT-IN transformation language, mappings can be previewed either directly in MapForce, or they can be compiled to server execution files and run by MapForce Server (see Compiling Mappings to MapForce Server Execution Files [1378]).
- The taxonomy of the Inline XBRL file might be either a URL address or a reference to a local file. In the latter case, if the mapping is run by MapForce Server, note the following additional requirements:
  - The taxonomy file must be in the same directory as the .mfx (mapping execution) file.
  - The Inline XBRL file must point to the taxonomy file using a relative path, for example `<link:schemaRef xlink:type="simple" xlink:href="taxonomy.xsd" />`.
  - The Component Settings dialog box (see XBRL Component Settings [767]) must also refer to the taxonomy file using a relative path (for example, "taxonomy.xsd").
- The source Inline XBRL file must contain a single XBRL instance; multiple instances are not supported.

Adding Inline XBRL instance files to the mapping

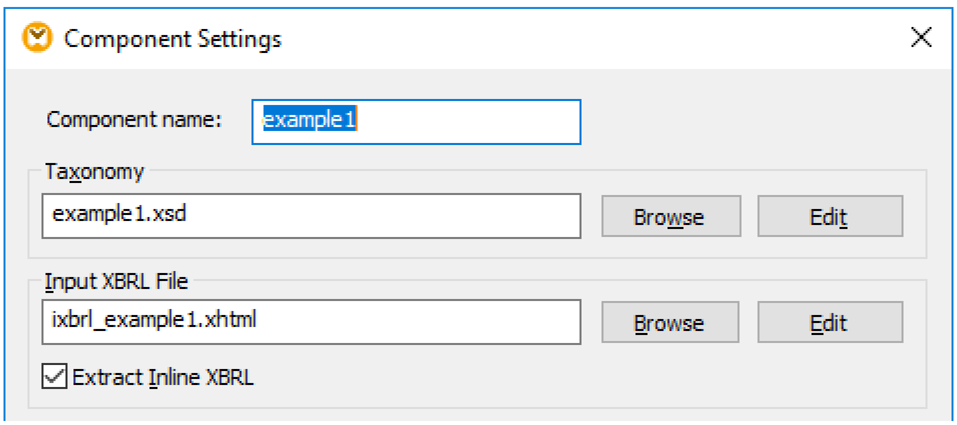
1. On the **Insert** menu, click **XBRL Document**.
2. Do one of the following:
   a. If you have the XBRL taxonomy, browse for the XBRL taxonomy (typically, a file with .xsd extension). This must be a valid taxonomy for the Inline XBRL instance file. You can also specify a URL as taxonomy; to do this, click the **Switch to URL** button on the Browse dialog box (see also Adding Components from a URL [80]).
   b. If you don't have the XBRL taxonomy file or URL, browse for the Inline XBRL instance file instead. In this case, MapForce will attempt to determine and load the corresponding XBRL taxonomy automatically.
3. When prompted to supply an instance, browse for the HTML file that contains inline XBRL (typically, a file with .html, .htm or .xhtml extension).

4. When prompted to select a structured view, leave the default option as is. You can change the structure view(s) at any time later (see Selecting Structure Views746).
5. Click **OK**.

At this stage, the structure of the Inline XBRL content becomes available in the component. In case there are validation errors, they are displayed in the Messages window. Otherwise, you can draw connections from the inline XBRL component to a target structure such as XML, text, database, and so on, in a similar fashion as with standard XBRL components.

The "Extract Inline XBRL" check box

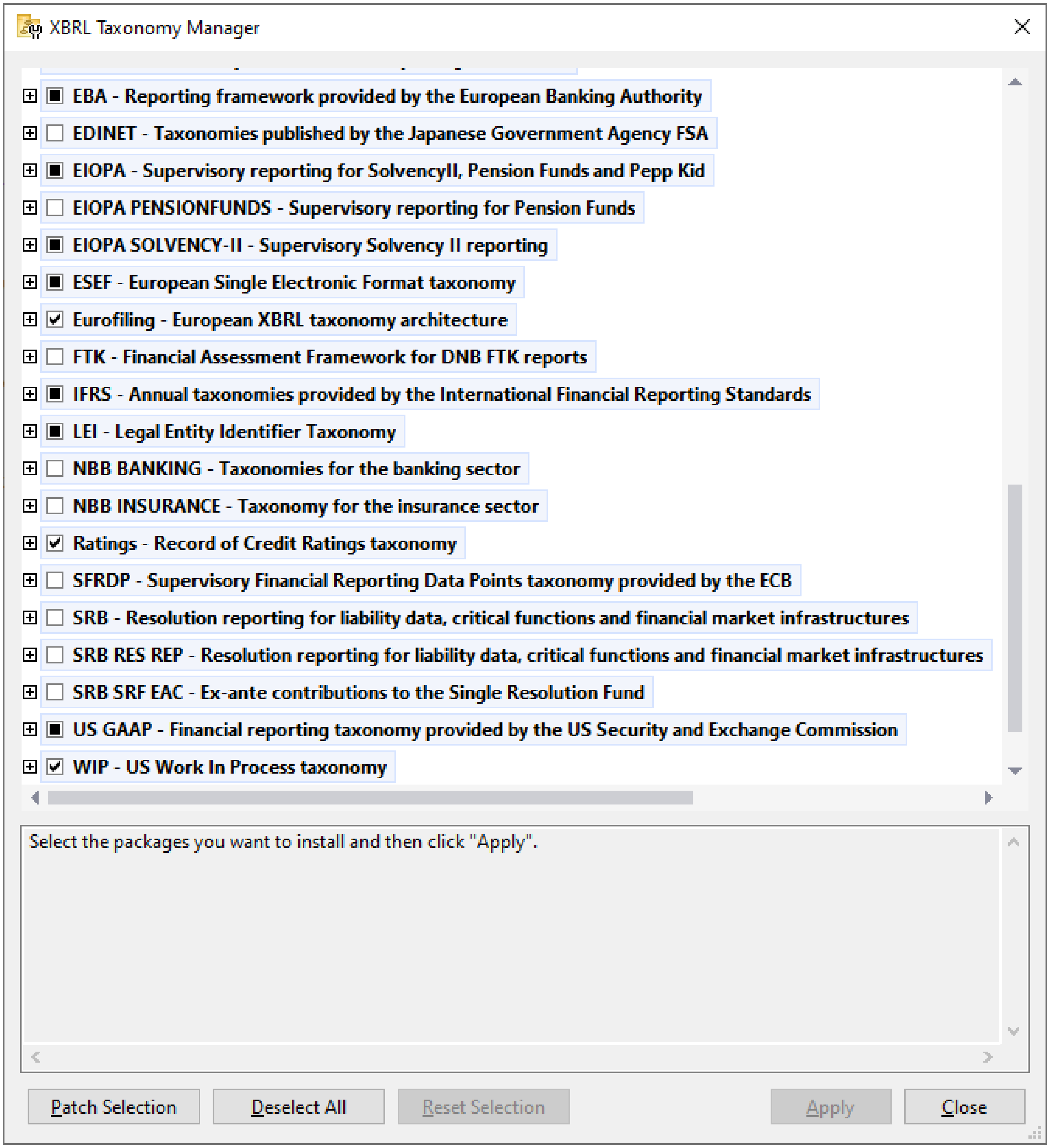
When the **Extract Inline XBRL** check box on the XBRL Component Settings767 dialog box is selected, the .html or .xhtml file specified as input will be treated as Inline XBRL. Consequently, MapForce will extract the Inline XBRL content from the input file and validate it when you close the dialog box. This check box is typically selected automatically when you add an Inline XBRL instance file to the mapping. To check whether this option is enabled, double-click the XBRL component header on the mapping (alternatively, right-click the XBRL component header, and select **Properties**).



4.8.5 Taxonomy Manager

XBRL Taxonomy Manager is an Altova tool that provides a centralized way to install and manage XBRL taxonomies for use across all Altova's XBRL-enabled applications, including MapForce.

- On Windows, Taxonomy Manager has a graphical user interface (*screenshot below*) and is also available at the command line. (Altova's desktop applications are available on Windows only; *see list below*.)
- On Linux and macOS, Taxonomy Manager is available at the command line only. (Altova's server applications are available on Windows, Linux, and macOS; *see list below*.)



Altova's XBRL-enabled applications

Desktop applications (Windows only)	Server applications (Windows, Linux, macOS)
Altova XBRL Add-ins for Excel (EBA, ESEF, Solvency II)	MapForce Server (Standard and Advanced Editions)

MapForce Enterprise Edition	RaptorXML+XBRL Server
StyleVision Enterprise Edition	StyleVision Server
XMLSpy Enterprise Edition	

Installation and de-installation of Taxonomy Manager

Taxonomy Manager is installed automatically when you first install a new version of Altova Mission Kit Enterprise Edition or of any of Altova's XBRL-enabled applications (*see table above*).

Likewise, it is removed automatically when you uninstall the last Altova XBRL-enabled application from your computer.

Taxonomy Manager features

Taxonomy Manager provides the following features:

- Shows XBRL taxonomies installed on your computer and checks whether new versions are available for download.
- Downloads newer versions of XBRL taxonomies independently of the Altova product release cycle. (Altova stores taxonomies online, and you can download them via Taxonomy Manager.)
- Install or uninstall any of the multiple versions of a given taxonomy (or all versions if necessary).
- An XBRL taxonomy may have dependencies on other taxonomies. When you install or uninstall a particular taxonomy, Taxonomy Manager informs you about dependent taxonomies and will automatically install or remove them as well.
- Taxonomy Manager uses the XML catalog mechanism to map schema references to local files. In the case of large XBRL taxonomies, processing will therefore be faster than if the taxonomies were at a remote location.
- All major taxonomies are available via Taxonomy Manager and are regularly updated for the latest versions. This provides you with a convenient single resource for managing all your taxonomies and making them readily available to all of Altova's XBRL-enabled applications.
- Changes made in Taxonomy Manager take effect for all Altova products installed on that machine.

Custom XBRL Taxonomies

If you need to work with custom XBRL taxonomies that are not included with Taxonomy Manager, you can add these taxonomies to Taxonomy Manager for management as follows:

- *In Altova desktop applications:* Select the **Tools | Options** menu command, and go to the *XBRL | Taxonomy Packages* section. Browse for the ZIP package of your custom XBRL taxonomy. For more information, see the description of this command in your desktop product documentation.
- *In Altova server applications:* When running commands from the command line that support custom taxonomies, provide the **`--taxonomy-package`** or **`--taxonomy-package-config-file`** option. For example: In RaptorXML+XBRL Server, these options are supported by XBRL validation commands such as valxbrl or valxbrltaxonomy; in MapForce, they are supported by run command.

How it works

Altova stores all XBRL taxonomies used in Altova products online. This repository is updated when new versions of the taxonomies are released. Taxonomy Manager displays information about the latest available taxonomies when invoked in both its GUI form as well as on the CLI. You can then install, upgrade or uninstall taxonomies via Taxonomy Manager.

Taxonomy Manager also installs taxonomies in one other way. At the Altova website (https://www.altova.com/taxonomy-manager) you can select a taxonomy and its dependent taxonomies that you want to install. The website will prepare a file of type **`.altova_taxonomies`** for download that contains information about your taxonomy selection. When you double-click this file or pass it to Taxonomy Manager via the CLI as an argument of the **`install`** 762 command, Taxonomy Manager will install the taxonomies you selected.

Local cache: tracking your taxonomies

All information about installed taxonomies is tracked in a centralized cache directory on your computer, located here:

Windows	`C:\ProgramData\Altova\pkgs\.cache`
Linux	`/var/opt/Altova/pkgs\.cache`
macOS	`/var/Altova/pkgs`

This cache directory is updated regularly with the latest status of taxonomies at Altova's online storage. These updates are carried out at the following times:

- Every time you start Taxonomy Manager.
- When you start MapForce for the first time on a given calendar day.
- If MapForce is open for more than 24 hours, the cache is updated every 24 hours.
- You can also update the cache by running the `update` 765 command at the command line interface.

The cache therefore enables Taxonomy Manager to continuously track your installed taxonomies against the taxonomies available online at the Altova website.

> **Do not modify the cache manually!**
>
> The local cache directory is maintained automatically based on the taxonomies you install and uninstall. It should not be altered or deleted manually. If you ever need to reset Taxonomy Manager to its original "pristine" state, then, on the command line interface (CLI): (i) run the `reset` 764 command, and (ii) run the `initialize` 762 command. (Alternatively, run the **`reset`** command with the **`--i`** option.)

4.8.5.1 Run Taxonomy Manager

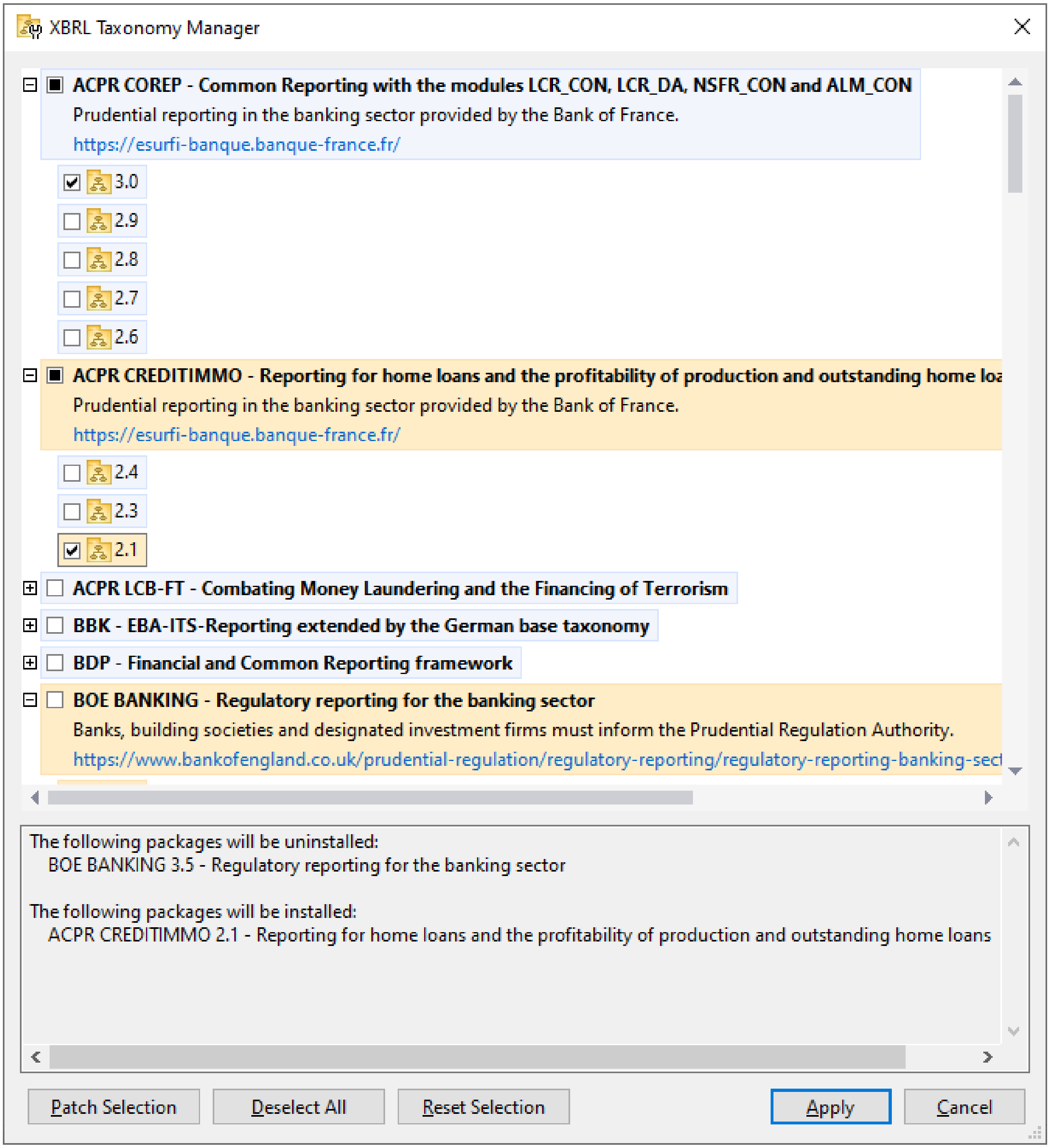
Graphical User Interface

You can access the GUI of Taxonomy Manager in any of the following ways:

- *During the installation of MapForce:* Towards the end of the installation procedure, select the check box *Invoke Altova Taxonomy Manager* to access the XBRL Taxonomy Manager GUI straight away. This will enable you to install taxonomies during the installation process of your Altova application.
- *After the installation of MapForce:* After your application has been installed, you can access the Taxonomy Manager GUI at any time, via the menu command **Tools | XBRL Taxonomy Manager**.

- Via the **`.altova_taxonomies`** file downloaded from the Altova's XBRL Taxonomy Download Center: Double-click the downloaded file to run the Taxonomy Manager GUI, which will be set up to install the taxonomies you selected (at the website) for installation.

After the Taxonomy Manager GUI (*screenshot below*) has been opened, already installed taxonomies will be shown selected. If you want to install an additional taxonomy, select it. If you want to uninstall an already installed taxonomy, deselect it. After you have made your selections and/or deselections, you are ready to apply your changes. The taxonomies that will be installed or uninstalled will be highlighted and a message about the upcoming changes will be posted to the Messages pane at the bottom of the Taxonomy Manager window (*see screenshot*).



Command line interface

You can run Taxonomy Manager from a command line interface by sending commands to its executable file, `taxonomymanager.exe`.

The **`taxonomymanager.exe`** file is located in the following folder:

- *On Windows:* `C:\ProgramData\Altova\SharedBetweenVersions`
- *On Linux or macOS (server applications only):* `%INSTALLDIR%/bin`, where `%INSTALLDIR%` is the program's installation directory.

You can then use any of the commands listed in the CLI command reference section.

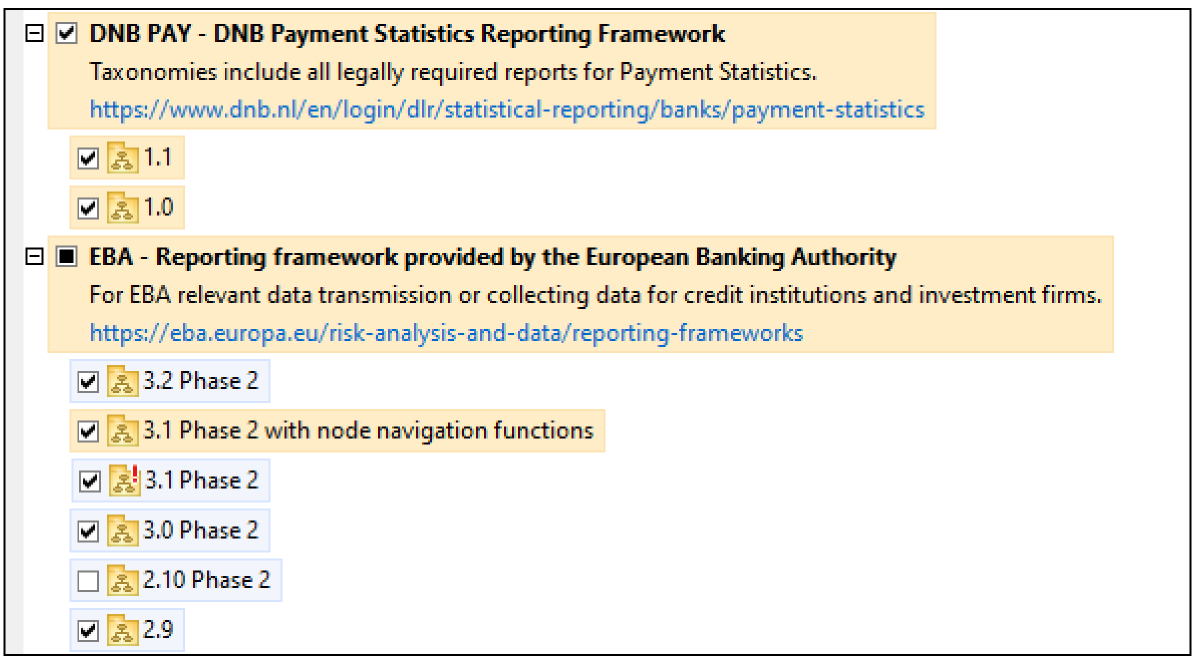
To display help for the commands, run the following:

- *On Windows:* `taxonomymanager.exe --help`
- *On Linux or macOS (server applications only):* `sudo ./taxonomymanager --help`

4.8.5.2 Status Categories

Taxonomy Manager categorizes the taxonomies under its management as follows:

- *Installed taxonomies.* These are shown in the GUI with their check boxes selected (*in the screenshot below the checked versions of the DNB and EBA taxonomies are installed taxonomies*). If all the versions of a taxonomy are selected, then the selection mark is a tick. If at least one version is unselected, then the selection mark is a solid colored square. You can deselect an installed taxonomy to **uninstall** it.
- *Uninstalled available taxonomies.* These are shown in the GUI with their check boxes unselected. You can select the taxonomies you want to **install**.



- *Upgradeable taxonomies* are those which have been revised by their issuers since they were installed. They are indicated in the GUI by a icon (*see screenshot above*). You can **patch** an installed taxonomy with an available revision.

Points to note

- In the screenshot above, both DNB taxonomies and some of the EBA taxonomies are checked. Those with the blue background are already installed. Those with the yellow background are uninstalled and have been selected for installation. Note that (i) the EBA 2.10 Phase 2 taxonomy is not installed and has not been selected for installation, (ii) the EBA 3.1 Phase 2 taxonomy has been installed, but it has been patched by its issuer since it was installed and the patch has not yet been installed.
- When running Taxonomy Manager from the command line, the `list` 763 command is used with different options to list different categories of taxonomies:

`taxonomymanager.exe list`	Lists all installed and available taxonomies; upgradeables are also indicated
`taxonomymanager.exe list -i`	Lists installed taxonomies only; upgradeables are also indicated
`taxonomymanager.exe list -u`	Lists upgradeable taxonomies

Note: On Linux and macOS, **use `sudo ./taxonomymanager list`**

4.8.5.3 Patch or Install a Taxonomy

Patch an installed taxonomy

Occasionally, XBRL taxonomies may receive patches (upgrades or revisions) from their issuers. When Taxonomy Manager detects that patches are available, these are indicated in the taxonomy listings of Taxonomy Manager and you can install the patches quickly.

In the GUI

Patches are indicated by the icon. (*Also see the previous topic about status categories* 757.) If patches are available, the **Patch Selection** button will be enabled. Click it to select and prepare all patches for installation. In the GUI, the icon of each taxonomy that will be patched changes from to , and the Messages pane at the bottom of the dialog lists the patches that will be applied. When you are ready to install the selected patches, click **Apply**. All patches will be applied together. Note that if you deselect a taxonomy marked for patching, you will actually be uninstalling that taxonomy.

On the CLI

To apply a patch at the command line interface:

1. Run the `list -u` 763 command. This lists any taxonomies where patch upgrades are available.
2. Run the `upgrade` 766 command to install all the patches.

Install an available taxonomy

You can install taxonomies using either the Taxonomy Manager GUI or by sending Taxonomy Manager the install instructions via the command line.

Note: If the current taxonomy references other taxonomies, the referenced taxonomies are also installed.

In the GUI
To install taxonomies using the Taxonomy Manager GUI, select the taxonomies you want to install and click **Apply**.

You can also select the taxonomies you want to install at the Altova website and generate a downloadable `.altova_taxonomies` file. When you double-click this file, it will open Taxonomy Manager with the taxonomies you wanted pre-selected. All you will now have to do is click **Apply**.

On the CLI
To install taxonomies via the command line, run the `install` 762 command:

```
taxonomymanager.exe install [options] Taxonomy+
```

where **Taxonomy** is the taxonomy (or taxonomies) you want to install or a **.altova_taxonomies** file. A taxonomy is referenced by an identifier of format **<name>-<version>**. (The identifiers of taxonomies are displayed when you run the **list** 763 command.) You can enter as many taxonomies as you like. For details, see the description of the `install` 762 command.

Note: On Linux or macOS, use the **sudo ./taxonomymanager** command.

Installing a required taxonomy
When you run an XBRL-related command in MapForce and MapForce discovers that a taxonomy it needs for executing the command is not present or is incomplete, Taxonomy Manager will display information about the missing taxonomy. You can then directly install any missing taxonomy via Taxonomy Manager.

In the Taxonomy Manager GUI, you can view all previously installed taxonomies at any time by running Taxonomy Manager from **Tools | Taxonomy Manager**.

4.8.5.4 Uninstall a Taxonomy, Reset

Uninstall a taxonomy

You can uninstall taxonomies using either the Taxonomy Manager GUI or by sending Taxonomy Manager the uninstall instructions via the command line.

Note: If the taxonomy you want to uninstall references other taxonomies, then the referenced taxonomies are also uninstalled.

In the GUI
To uninstall taxonomies in the Taxonomy Manager GUI, clear their check boxes and click **Apply**. The selected taxonomies and their referenced taxonomies will be uninstalled.

To uninstall all taxonomies, click **Deselect All** and click **Apply**.

On the CLI
To uninstall taxonomies via the command line, run the **`uninstall`** command:

```
taxonomymanager.exe uninstall [options] Taxonomy+
```

where each **`Taxonomy`** argument is a taxonomy you want to uninstall or a **`.altova_taxonomies`** file. A taxonomy is specified by an identifier that has a format of **`<name>-<version>`**. (The identifiers of taxonomies are displayed when you run the **`list`** 763 command.) You can enter as many taxonomies as you like. For details, see the description of the `uninstall` 764 command.

Note: On Linux or macOS, use the **`sudo ./taxonomymanager`** command.

Reset Taxonomy Manager

You can reset Taxonomy Manager. This removes all installed taxonomies and the cache directory.

- In the GUI, click **Reset Selection**.
- On the CLI, run the `reset` 764 command.

After running this command, make sure to run the `initialize` 762 command in order to recreate the cache directory. Alternatively, run the `reset` 764 command with the `-i` option.

Note that `reset -i` 764 restores the original installation of the product, so it is recommended that you run the `update` 765 command after performing a reset. Alternatively, run the `reset` 764 command with the `-i` and `-u` options.

4.8.5.5 Command Line Interface (CLI)

To call Taxonomy Manager at the command line, you need to know the path of the executable. By default, the Taxonomy Manager executable is installed here:

```
C:\ProgramData\Altova\SharedBetweenVersions\TaxonomyManager.exe
```

Note: On Linux and macOS systems, once you have changed the directory to that containing the executable, you can call the executable with **`sudo ./taxonomymanager`**. The prefix **`./`** indicates that the executable is in the current directory. The prefix **`sudo`** indicates that the command must be run with root privileges.

Command line syntax

The general syntax for using the command line is as follows:

```
<exec> -h | --help | --version | <command> [options] [arguments]
```

In the listing above, the vertical bar **`|`** separates a set of mutually exclusive items. The square brackets **`[]`** indicate optional items. Essentially, you can type the executable path followed by either `--h`, `--help`, or `--version` options, or by a command. Each command may have options and arguments. The list of commands is described in the following sections.

4.8.5.5.1 help

This command provides contextual help about commands pertaining to Taxonomy Manager executable.

Syntax

```
<exec> help [command]
```

Where `[command]` is an optional argument which specifies any valid command name.

Note the following:

- You can invoke help for a command by typing the command followed by **`-h`** or **`--help`**, for example: **`<exec> list -h`**
- If you type **`-h`** or **`--help`** directly after the executable and before a command, you will get general help (not help for the command), for example: **`<exec> -h list`**

Example

The following command displays help about the `list` command:

```
taxonomymanager help list
```

4.8.5.5.2 info

This command displays detailed information for each of the taxonomies supplied as a `Taxonomy` argument. This information for each submitted taxonomy includes the title, version, description, publisher, and any dependent taxonomies, as well as whether the taxonomy has been installed or not.

Syntax

```
<exec> info [options] Taxonomy+
```

- The **`Taxonomy`** argument is the name of a taxonomy or a part of a taxonomy's name. (To display a taxonomy's package ID and detailed information about its installation status, you should use the `list` 763 command.)
- Use **`<exec> info -h`** to display help for the command.

Example

The following command displays information about the **`eba-2.10`** and **`us-gaap-2020.0`** taxonomies:

```
taxonomymanager info eba-2.1.0 us-gaap-2020.0
```

4.8.5.5.3 initialize

This command initializes the Taxonomy Manager environment. It creates a cache directory where information about all taxonomies is stored. Initialization is performed automatically the first time an XBRL-enabled Altova application is installed. You would not need to run this command under normal circumstances, but you would typically need to run it after executing the **`reset`** command.

Syntax

```
<exec> initialize | init [options]
```

Options
The **`initialize`** command takes the following options:

`--silent, --s`	Display only error messages. The default is **`false`**.
`--verbose, --v`	Display detailed information during execution. The default is **`false`**.
`--help, --h`	Display help for the command.

Example

The following command initializes Taxonomy Manager:

```
taxonomymanager initialize
```

4.8.5.5.4 install

This command installs one or more taxonomies.

Syntax

```
<exec> install [options] Taxonomy+
```

To install multiple taxonomies, add the **`Taxonomy`** argument multiple times.

The **`Taxonomy`** argument is one of the following:

- A taxonomy identifier (having a format of **`<name>-<version>`**, for example: **`eba-2.10`**). To find out the taxonomy identifiers of the taxonomies you want, run the `list` 763 command. You can also use an abbreviated identifier if it is unique, for example **`eba`**. If you use an abbreviated identifier, then the latest version of that taxonomy will be installed.
- The path to a **`.altova_taxonomies`** file downloaded from the Altova website. For information about these files, see *Introduction to TaxonomyManager: How It Works* 751.

Options
The **`install`** command takes the following options:

`--silent, --s`	Display only error messages. The default is **`false`**.
`--verbose, --v`	Display detailed information during execution. The default is **`false`**.
`--help, --h`	Display help for the command.

Example

The following command installs the latest **`eba`** (European Banking Authority) and **`us-gaap`** (US Generally Accepted Accounting Principles) taxonomies:

```
taxonomymanager install eba us-gaap
```

4.8.5.5.5 list

This command lists taxonomies under the management of Taxonomy Manager. The list displays one of the following

- All available taxonomies
- Taxonomies containing in their name the string submitted as a **`Taxonomy`** argument
- Only installed taxonomies
- Only taxonomies that can be upgraded

Syntax

```
<exec> list | ls [options] Taxonomy?
```

If no **`Taxonomy`** argument is submitted, then all available taxonomies are listed. Otherwise, taxonomies are listed as specified by the submitted options (*see example below*). Note that you can submit the **`Taxonomy`** argument multiple times.

Options
The **`list`** command takes the following options:

`--installed, --i`	List only installed taxonomies. The default is **`false`**.
`--upgradeable, --u`	List only taxonomies where upgrades (patches) are available. The default is **`false`**.
`--help, --h`	Display help for the command.

Examples

- To list all available taxonomies, run: **`taxonomymanager list`**
- To list installed taxonomies only, run: **`taxonomymanager list -i`**
- To list taxonomies that contain either "eba" or "us-gaap" in their name, run: **`taxonomymanager list eba us-gaap`**

4.8.5.5.6 reset

This command removes all installed taxonomies and the cache directory. You will be completely resetting your taxonomy environment. After running this command, be sure to run the `initialize`[762] command to recreate the cache directory. Alternatively, run the `reset` command with the `-i` option. Since `reset -i` restores the original installation of the product, we recommend that you run the `update`[765] command after performing a reset and initialization. Alternatively, run the reset command with both the `-i` and `-u` options.

Syntax

```
<exec> reset [options]
```

Options

The **`reset`** command takes the following options:

`--init, --i`	Initialize Taxonomy Manager after reset. The default is **`false`**.
`--update, --u`	Updates the list of available taxonomies in the cache. The default is **`false`**.
`--silent, --s`	Display only error messages. The default is **`false`**.
`--verbose, --v`	Display detailed information during execution. The default is **`false`**.
`--help, --h`	Display help for the command.

Examples

- To reset Taxonomy Manager, run: **`taxonomymanager reset`**
- To reset Taxonomy Manager and initialize it, run: **`taxonomymanager reset -i`**
- To reset Taxonomy Manager, initialize it,and update its taxonomy list, run: **`taxonomymanager reset -i -u`**

4.8.5.5.7 uninstall

This command uninstalls one or more taxonomies. By default, any taxonomies referenced by the current one are uninstalled as well. To uninstall just the current taxonomy and keep the referenced taxonomies, set the option `--k`.

Syntax

```
<exec> uninstall [options] Taxonomy+
```

To uninstall multiple taxonomies, add the **`Taxonomy`** argument multiple times.

The **`Taxonomy`** argument is one of the following:

- A taxonomy identifier (having a format of **<name>-<version>**, for example: **eba-2.10**). To find out the taxonomy identifiers of the taxonomies that are installed, run the `list -i` (763) command. You can also use an abbreviated taxonomy name if it is unique, for example **eba**. If you use an abbreviated name, then all taxonomies that contain the abbreviation in its name will be uninstalled.
- The path to a **.altova_taxonomies** file downloaded from the Altova website. For information about these files, see *Introduction to TaxonomyManager: How It Works* (751).

Options
The **uninstall** command takes the following options:

`--keep-references, --k`	Set this option to keep referenced taxonomies. The default is **false**.
`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.
`--help, --h`	Display help for the command.

Example
The following command uninstalls the **eba-2.10** and **us-gaap-2020.0** taxonomies and their dependencies:

```
taxonomymanager uninstall eba-2.10 us-gaap-2020.0
```

The following command uninstalls the **eba-2.10** taxonomy but not the taxonomies it references:

```
taxonomymanager uninstall --k eba-2.10
```

4.8.5.5.8 update

This command queries the list of taxonomies available from the online storage and updates the local cache directory. You should not need to run this command unless you have performed a `reset` (764) and `initialize` (762).

Syntax

```
<exec> update [options]
```

Options
The **update** command takes the following options:

`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.
`--help, --h`	Display help for the command.

Example
The following command updates the local cache with the list of latest taxonomies:

```
taxonomymanager update
```

4.8.5.5.9 upgrade

This command upgrades all installed taxonomies that can be upgraded to the latest available *patched* version. You can identify upgradeable taxonomies by running the `list -u` 763 command.

Note: The **`upgrade`** command removes a deprecated taxonomy if no newer version is available.

Syntax

```
<exec> upgrade [options]
```

Options
The **`upgrade`** command takes the following options:

`--silent, --s`	Display only error messages. The default is **`false`**.
`--verbose, --v`	Display detailed information during execution. The default is **`false`**.
`--help, --h`	Display help for the command.

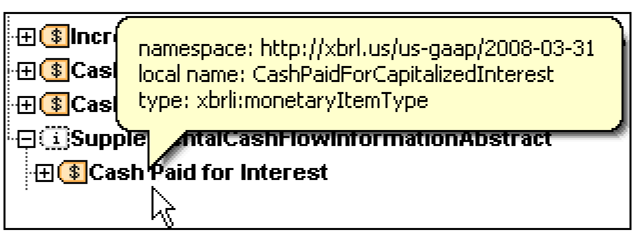
4.8.6 XBRL Settings and Preferences

In MapForce, you can configure the XBRL-specific settings as follows:

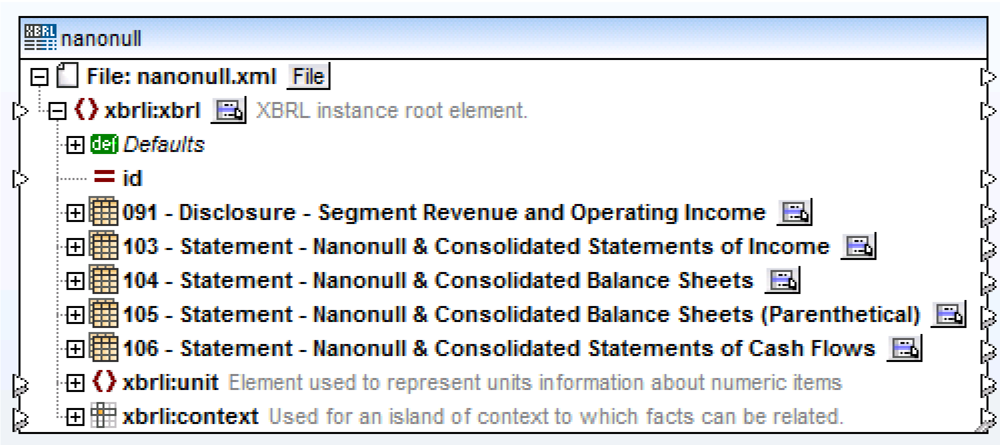
- At the mapping level, you can view additional information about component items, see Enabling Tips and Annotations 766
- For each XBRL component on the mapping, you can change its applicable settings, see XBRL Component Settings 767
- You can also configure application-wide XBRL settings 1477.

4.8.6.1 Enabling Tips and Annotations

When tips are enabled, you can view additional information about each component item if you place the mouse cursor over it, as shown below.



You can also switch on or off the annotation text (if it exists). When enabled, annotations are displayed to the right of the item inside the component. For example, in the screen shot below, the annotation of the `xbrli:xbrl` root element is "XBRL instance root element".



To enable or disable tips, do one of the following:

- On the **View** menu, click **Show Tips**.
- Click the **Show Tips** () toolbar button.

To enable or disable annotations, do one of the following:

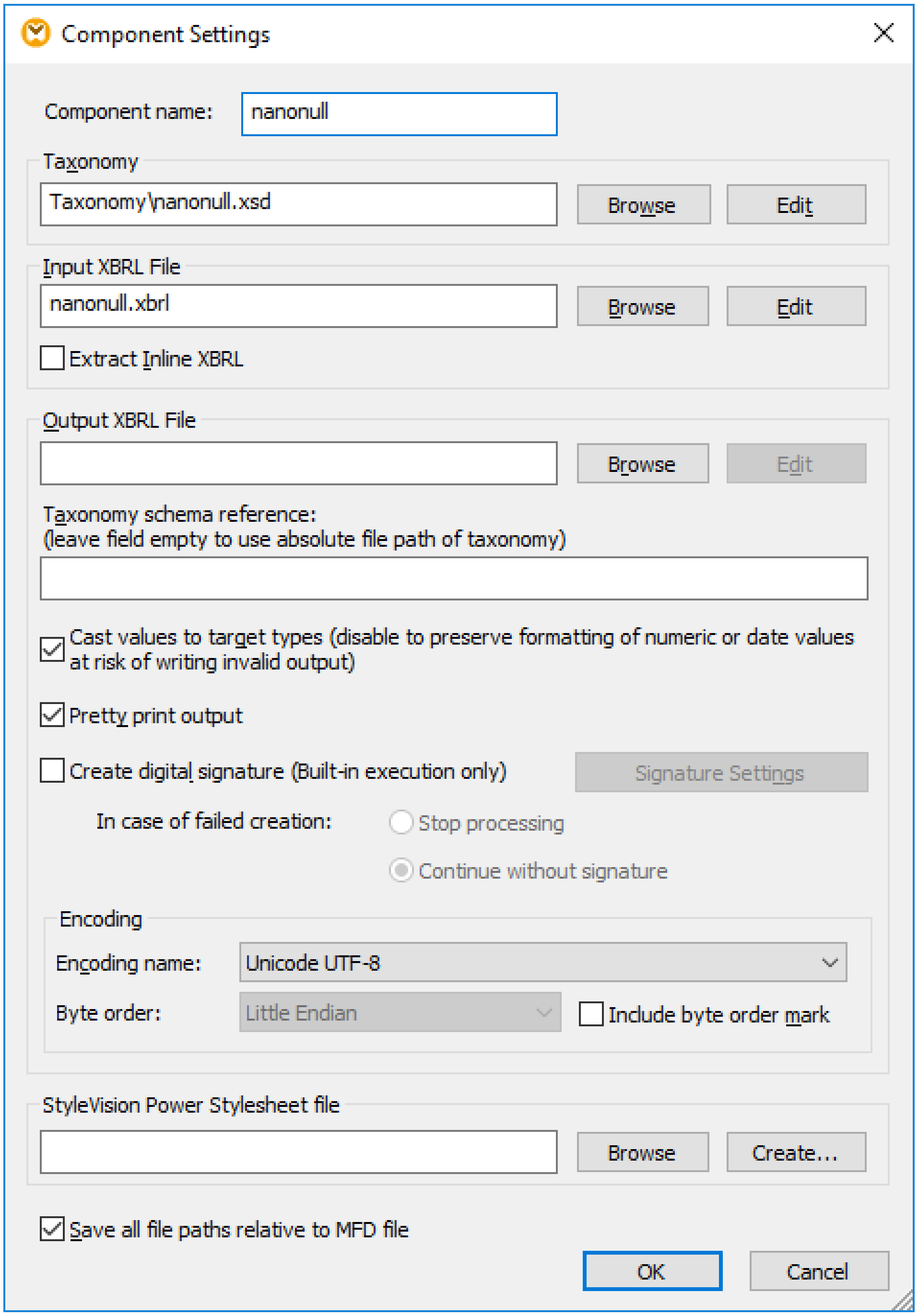
- On the **View** menu, click **Show Annotations**.
- Click the **Show Annotations** () toolbar button.

4.8.6.2 XBRL Component Settings

Once you added an XBRL component to the mapping area, you can configure its settings from the "Component Settings" dialog box.

To open the Component Settings dialog box:

- Double-click the title bar of the XBRL component, or right-click it and select **Properties**.



The Component Settings dialog box includes the following settings.

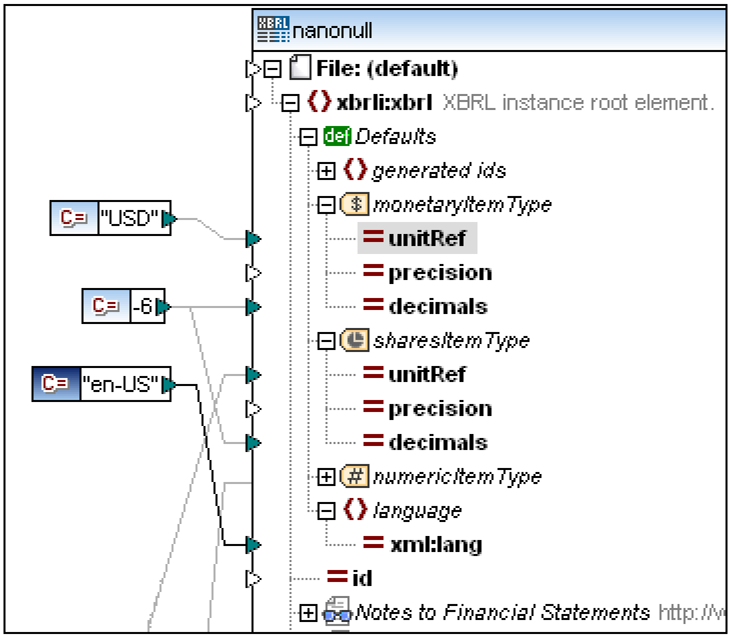
Component name	The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change

	the name of the component, be aware of the following: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.
Taxonomy	Specifies the file name and path of the main taxonomy file. To change the location of the file, click **Browse** and select the new file. To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.
Input XBRL File	Specifies the file name of the input XBRL instance for the currently selected XBRL component. This field is filled automatically when you first insert the XBRL component and assign an XBRL instance file. To change the location of the file, click **Browse** and select the new file. To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.
Extract Inline XBRL	When this check box is selected, the Inline XBRL content will be extracted from the .html, .htm, or .xhtml file specified as input.
Output XBRL File	Specifies the file name and path where the XBRL target instance file is placed, if the component is used as a mapping target. The entry from the Input XBRL File field is automatically copied to this field when you assign the input XBRL instance file. If you do not assign an input XBRL instance file to the component, then this field contains the file name and path of the taxonomy file and the extension "xml". To change the location of the file, click **Browse** and select the new file. To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.
Taxonomy schema reference	The path of the referenced/associated taxonomy schema file relative to the MFD file. Use this field if you want to specify a different taxonomy location for validation. The taxonomy reference is written into the `href` attribute of the `link:schemaRef` element, for example:

	`<link:schemaRef xlink:type="simple" xlink:href="..\..\nanonull.xsd"/>`
Cast values to target types	Allows you to define if the target XML schema types should be used when mapping, or if all data mapped to the target component should be treated as **string** values. By default, this setting is enabled.
Pretty print output	Reformats the output XBRL document to give it a structured look. Each child node is offset from its parent by a single tab character.
Create digital signature	Allows you to add a digital signature to the XBRL output instance file. Adding a digital signature is possible when you select "Built-in" as transformation language, see also Digital Signatures 139.
Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included.
StyleVision Power Stylesheet file	This option allows you to select or create an Altova StyleVision stylesheet file. Such files enable you to output data from the XBRL instance file to a variety of formats suitable for reporting, such as HTML, RTF, and others.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the following files: • The XBRL taxonomy file • The XBRL input file • The XBRL output file • The StyleVision stylesheet file See also Using Relative Paths on a Component 85.

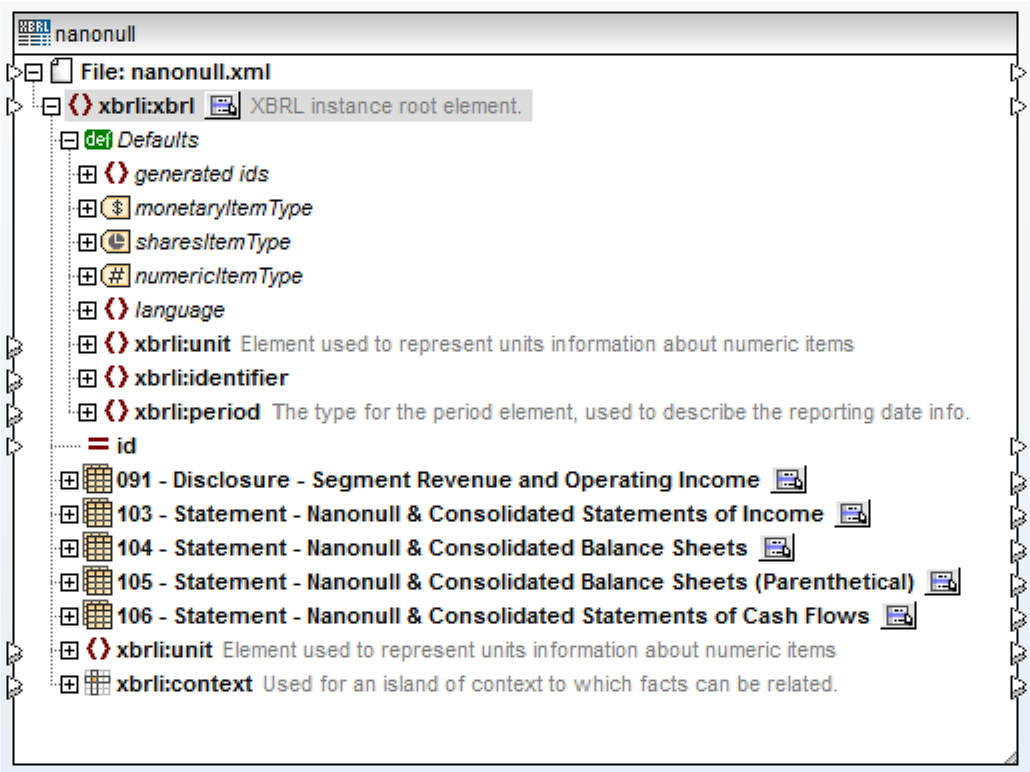
4.8.7 XBRL Defaults

Defaults are a powerful way to assign values to attributes in the XBRL component without having explicit mapping connections to all of them. For example, if you assign a constant unit identifier to the attribute `unitRef` of the `monetaryItemType` item within the Defaults hierarchy (as shown in the following screen shot), this assigns the default to every monetary XBRL item unit identifier, except where its input is mapped explicitly by some other value. The screen shot below also illustrates the use of the default value for the `xml:lang` attribute which defines the language of a footnote.



By default, the "Defaults" node is visible in a new XBRL component; however, you can hide it if you do not need to map to Defaults. You can display the "Defaults" node at the root level (for the whole XBRL document), or for individual nodes at any hierarchical level of the XBRL component. Note that, since Defaults can be defined at any level in the XBRL structure, different subtrees can have different default values.

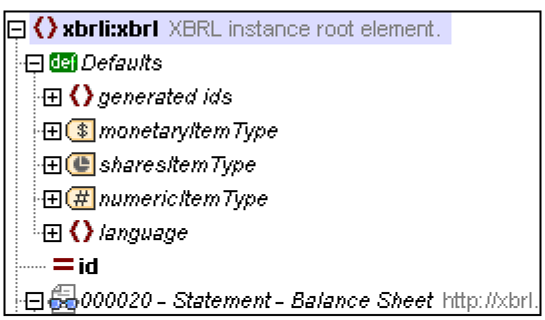
You can also map to aspect value defaults from the table linkbase, if your XBRL component uses the table linkbase view instead of the presentation/definition linkbase views. In the screen shot below, the `xbrli:unit`, `xbrli:identifier` and `xbrli:period` elements are aspect value defaults that you can use when mapping to XBRL tables.



To display the "Defaults" node for a particular item:

1. Right-click the item for which you want to display the default units, and select **XBRL | Show defaults**. (Alternatively, click the **Show Context Menu** ( ) button if available for the node, and then click **Show Defaults**.)

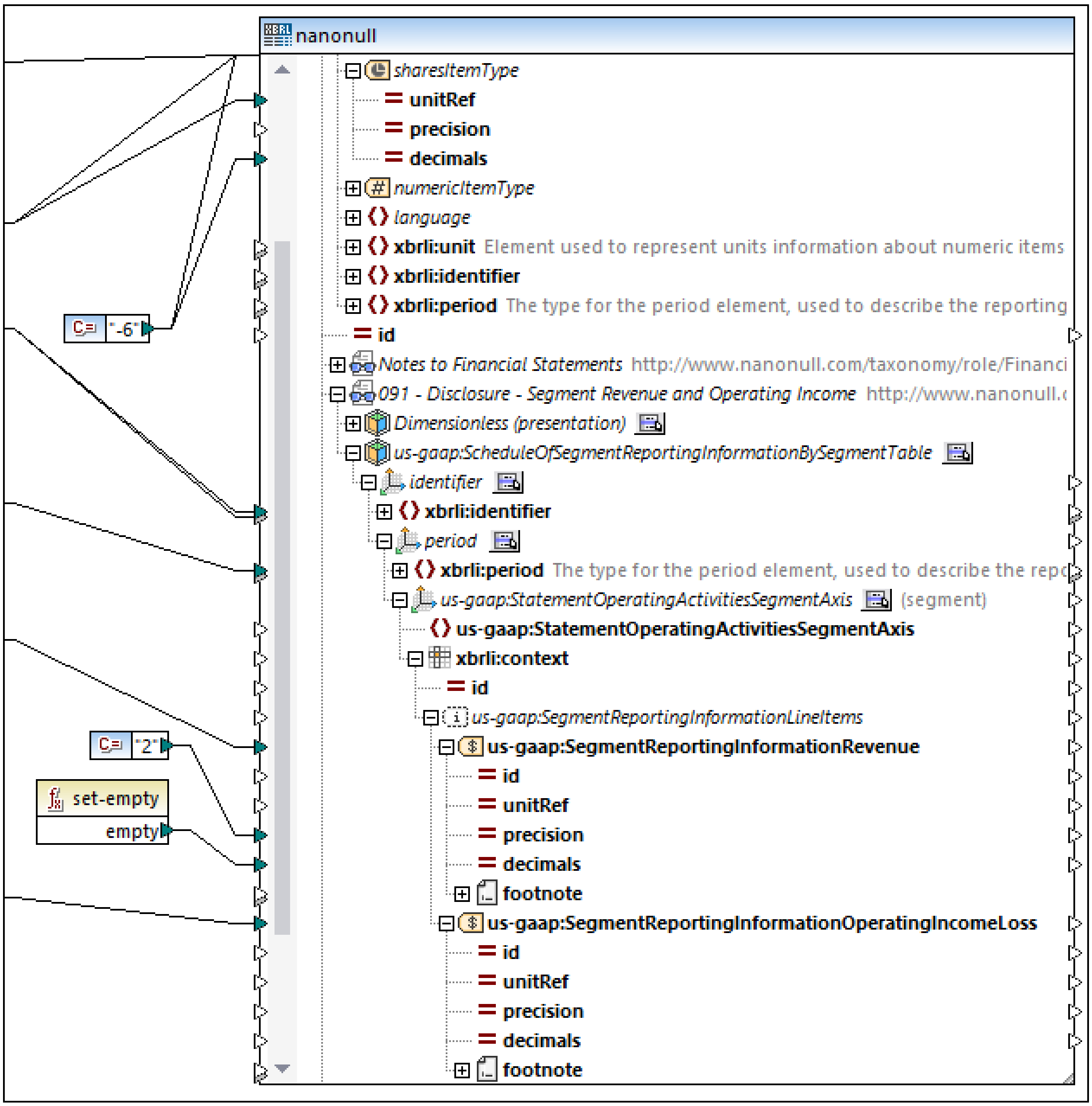
   This inserts a Defaults item to which you can connect your own default values for the various item types.

Replacing or de-activating a default value

If a default value has been defined for some concept attribute, e. g. **decimals**, it is possible to remove this setting locally for each concept, by using the function **set-empty**. For more information about functions, see Functions 904.

In the example mapping below, the **set-empty** function "deactivates" the default value "-6" for the monetary item "Segment Reporting Information, Revenue". This item will now be reported with the precision of "2" mapped to the "precision" attribute, while the other item, "Segment Reporting Information, Operating Income Loss" will be reported with the default decimals value of "-6".



Consequently, in the mapping output, these two items would look as illustrated below:

```
<us-gaap:SegmentReportingInformationRevenue contextRef="ctx1" unitRef="USD"
precision="2">23000988</us-gaap:SegmentReportingInformationRevenue>
<us-gaap:SegmentReportingInformationOperatingIncomeLoss contextRef="ctx1" unitRef="USD"
decimals="-6">5605900</us-gaap:SegmentReportingInformationOperatingIncomeLoss>
```

Context handling

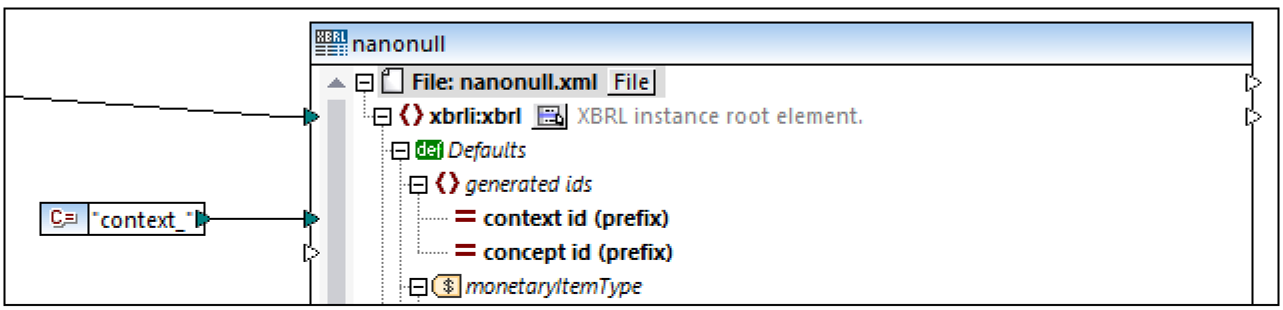
The hierarchical structure within XBRL components allows automatic context handling. The generation of the `xbrli:context` in XBRL output instances is done automatically when reporting related concepts.

The value of the attribute `id` of a specific context in an XBRL instance is the value of the attribute `contextRef` in each related XBRL concept. MapForce automatically numbers all created contexts in an output instance.

Customization is possible by assigning text as a prefix into the node "context id (prefix)" under "generated ids".

For example, mapping the constant value "**context_**" as a default prefix creates consecutive context-ids in the output instance having the values "context_1", "context_2", "context_3", and so on.

If no default value is defined, MapForce will create all context-ids with the prefix "**ctx_**".



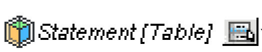
If the output XBRL instance contains **footnotes**, the related concepts must have concept IDs to link to the automatically generated footnote links. These attributes are automatically generated. The item "**concept id (prefix)**" can be mapped to determine such a prefix.

If the prefix is not mapped within the XBRL component, MapForce will create all concept IDs with the prefix "**fact_**".

4.8.8 XBRL Hypercubes

XBRL hypercubes enable automatic context handling. For more information about XBRL hypercubes, see [the XBRL Dimensions 1.0 Specification](). MapForce shows all hypercubes which have reportable concepts. If one of the related hypercube dimensions has no domain, it is not shown in the XBRL component.

Types of XBRL hypercubes
XBRL hypercubes can be of the following types:

- Defined by the taxonomy (e.g., 

).

- Generated by MapForce to simplify the default dimensions identifier and period, derived from the `Presentation` linkbase (e.g., *Dimensionless (presentation)*).
- Generated by MapForce within the **All concept node** *Dimensionless* .

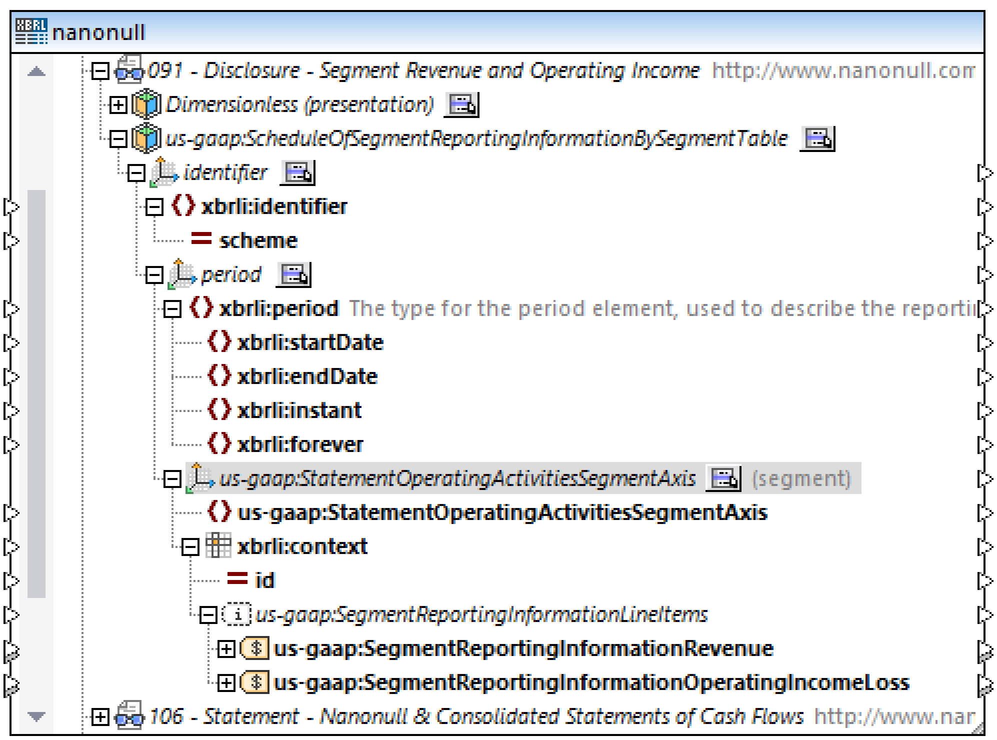
Every hypercube contains two default dimensions, identifier and period, that support the easy reading/writing of these two elements for each context. Additionally defined dimensions in the taxonomy are automatically related to the context elements `xbrli:segment` and `xbrli:scenario`.

Hypercubes denoted as `Dimensionless (presentation)` use both default dimensions. The hierarchical order of concepts shown within its context node is taken from the presentation linkbase.

`Dimensionless` hypercube items also use both default dimensions, but do not have any hierarchical concept order and show only the raw list of all concepts defined in the taxonomy.

All other hypercubes are defined within the taxonomy and are designated according to the name defined in the Label linkbase of the taxonomy.

Hypercubes and their dimensions (or Axes) have an icon which allows you to define the presentation of each of the dimensions in the component. The screen shot below shows a sample taxonomy file which contains both generated hypercubes and hypercubes defined by the taxonomy.



In addition to concepts, any tuples which contain only items of the same hypercube are shown as children of the context node.

4.8.8.1 Showing Dimensions in a Component

Dimension items in XBRL refer either to **explicit** or **typed** dimension values in the instance. The annotation of each dimension item shows in brackets whether the dimension is reported in the context elements `xbrli:segment` or `xbrli:scenario`.

Typed dimension items show the elements of their XML Schema type as children. Their values can be directly mapped.

Explicit dimensions in an XBRL taxonomy have a value of type `xs:QName` from a certain domain. This comprises the XBRL domain member values and the value of the XBRL domain item itself. Explicit dimensions can be displayed in two different modes, depending on the mapping requirements and the other component/structure you are mapping to or from XBRL.

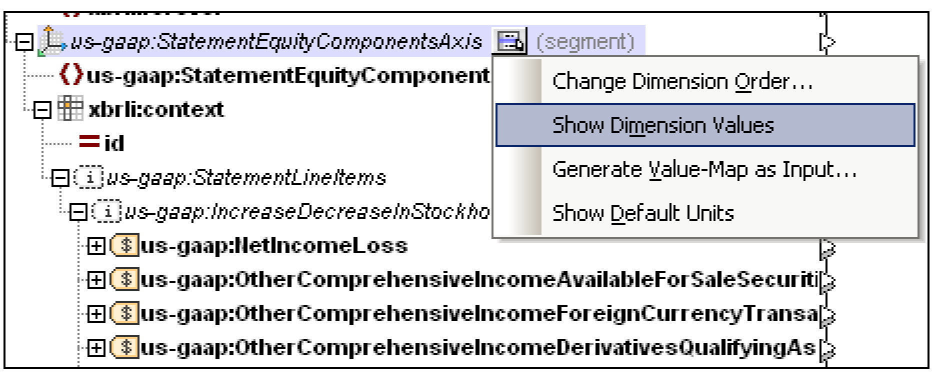
Initially, the explicit dimension is displayed with a **single** child node and can be mapped directly using this child, e. g. "Statement, Equity Components [Axis]".

This is useful (for an XBRL target component) when the dimension values can be derived from a field in the source data, e.g. a database field, or a column in an Excel table. As the source data will generally not contain the required `QName` datatype, MapForce can automatically create them using the value-map function (see Generating Value-Maps for Hypercube Dimensions 780).

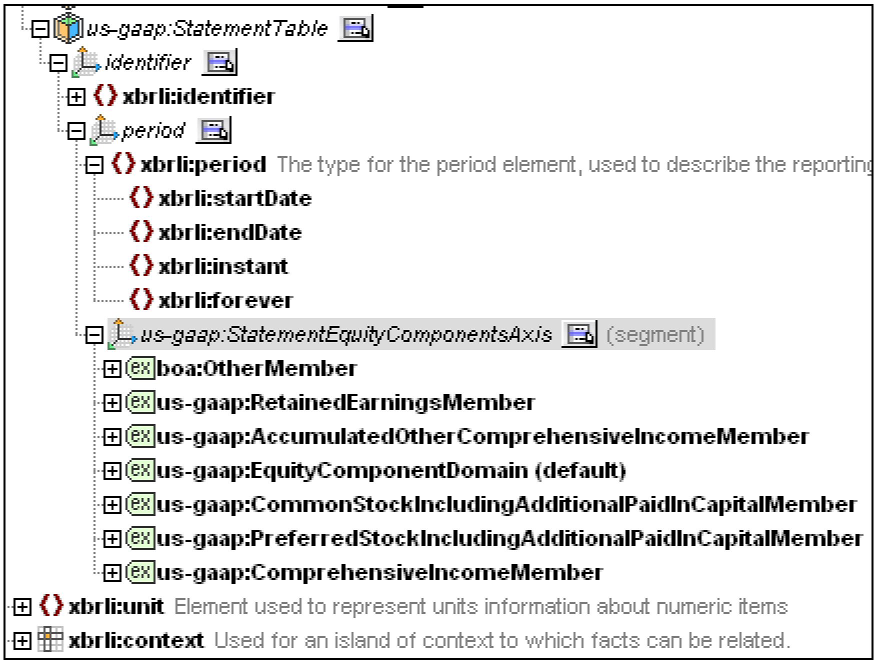
To allow different mappings for the facts related to each dimension member, you can display separate nodes for every single value of the dimension domain.

To show the dimension values in the component:

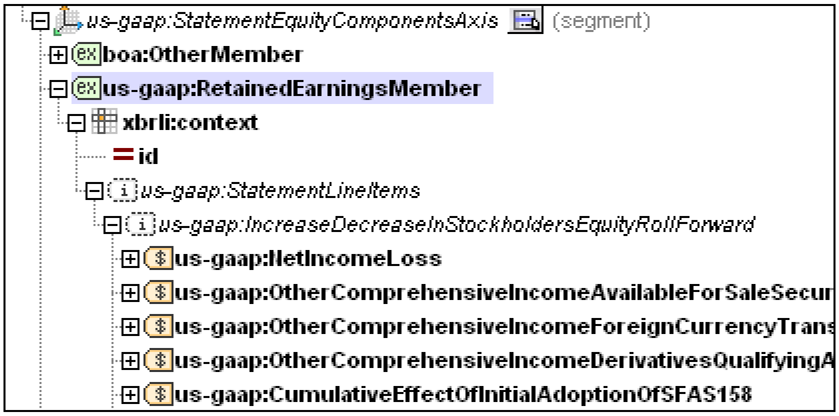
1. Click the icon of the dimension you want to see the values for, and select "Show Dimension Values".



This changes the items visible below the dimension name. The dimension **domain** and **member** items are now visible, each with a light green icon. These are all explicit members of a domain which is shown by the "ex" prefix in the item icon.



Each explicit member will now contain the same substructure, allowing different mappings for each.



When the output of a concept is mapped, only those values are used for which the related context element has the appropriate dimension value. For example, the value of "Net Income (Loss)" in the instance is mapped only for contexts which contain the dimension value "Comprehensive Income [Member]" for the dimension "Statement, Equity Components [Axis]". There is no additional filtering required.

When writing XBRL output instances, the automatic generation of proper dimension values within the context is supported. For example, for every reported monetary item "Net Income (Loss)", the context node **xbrli:context** acquires within its context element (**xbrli:segment**) an element for the explicit dimension "Statement, Equity Components [Axis]" containing the value "Comprehensive Income [Member]".

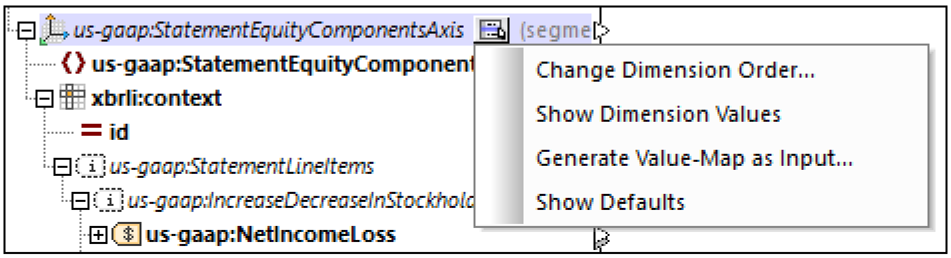
4.8.8.2 Changing the Order of Dimensions

Initially, MapForce displays all dimensions of a hypercube as nested child nodes, automatically creating a hierarchy. The hierarchical order of dimensions within the hypercube can be changed to match the other (non-XBRL) side of the mapping.

Furthermore, where dimension values have to be set specifically for some concepts, MapForce is able to display a dimension, without a hierarchy, and show it as a child element of the context node.

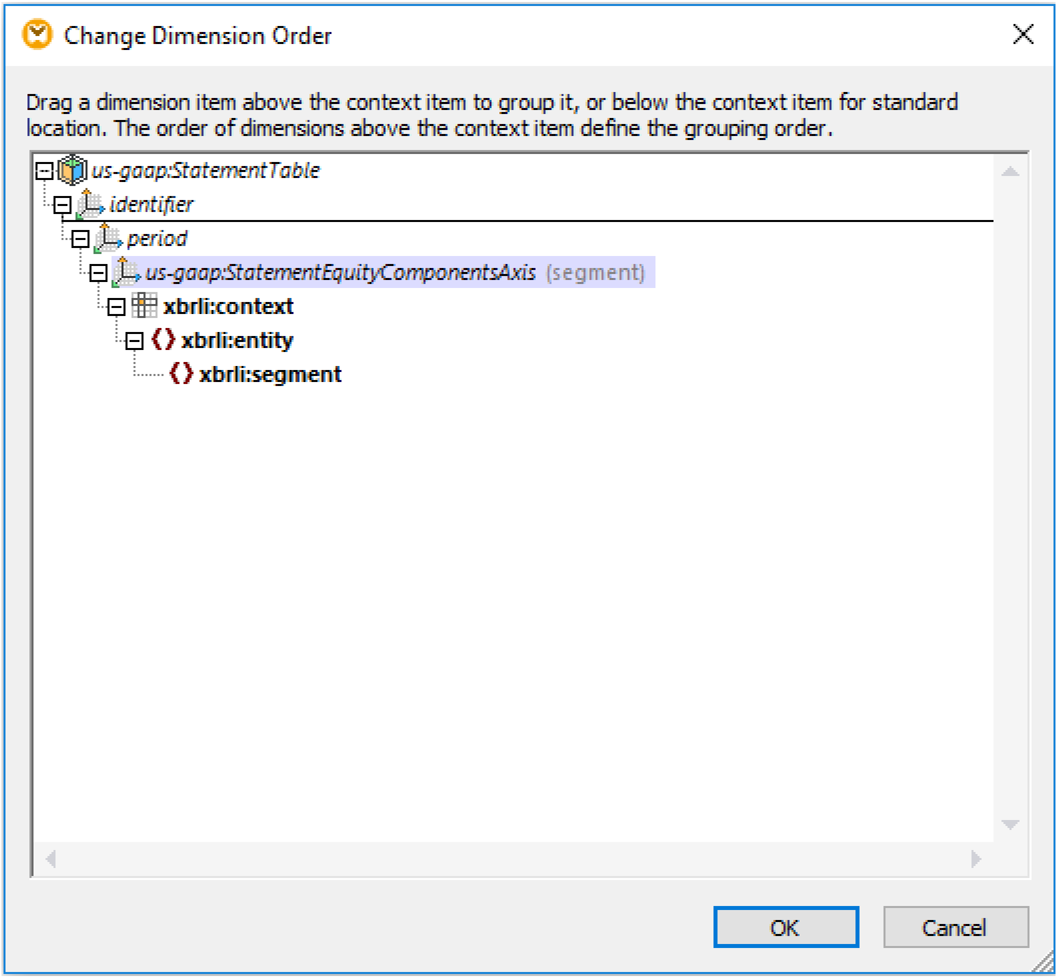
To change the order of dimension items:

1. Click any one of the  icons of the respective hypercube, and select "Change Dimension Order" entry in the popup menu.

   This opens a dialog box allowing you to reposition the various dimensions of a hypercube. Note that a hypercube has two **default dimensions**: **identifier** and **period** whose order in the hypercube can also be changed.

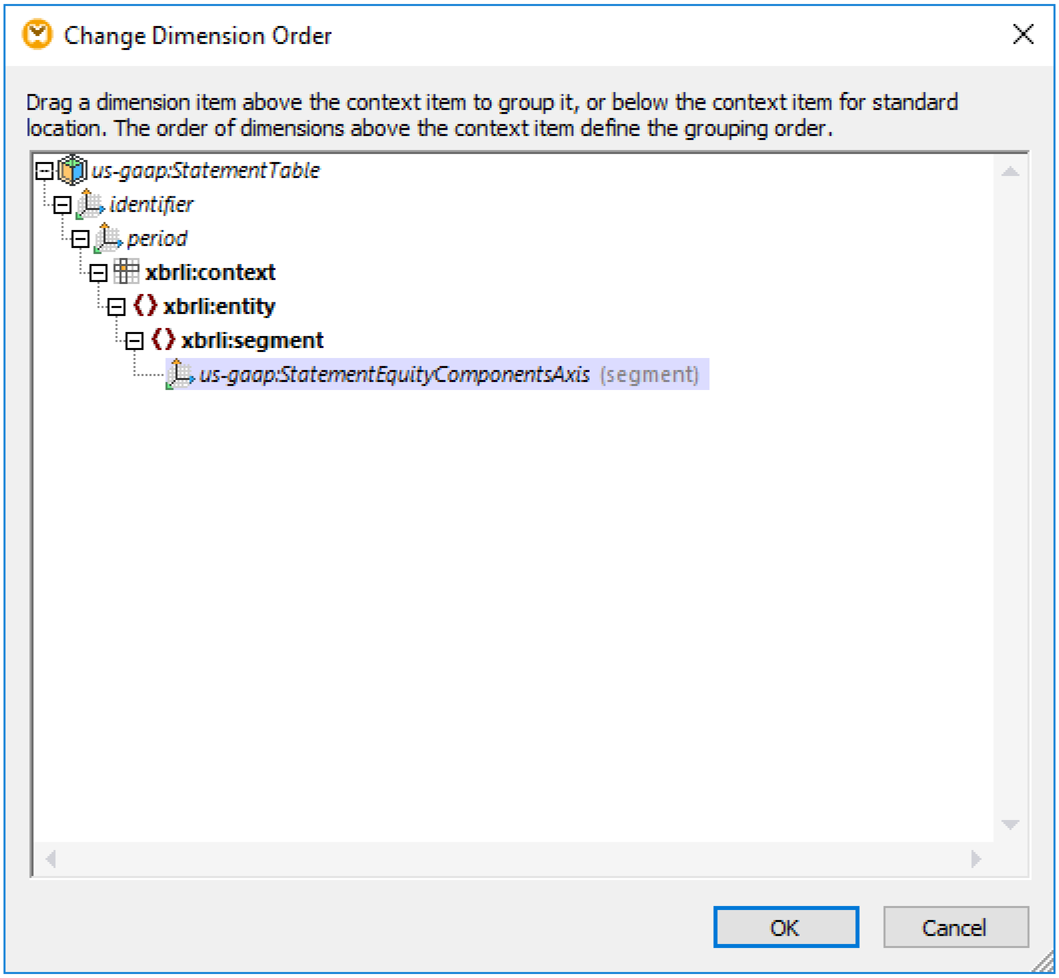
2. Click the hypercube dimension and use drag-and-drop to reposition it in the dialog box. A line appears at a position where the dimension can be dropped.



3. Click OK to close the dialog and have the dimension repositioned in the component.

To exclude a dimension from the hierarchy:

- Drag the dimension below the **xbrli:context** line, which will insert it into its context item.

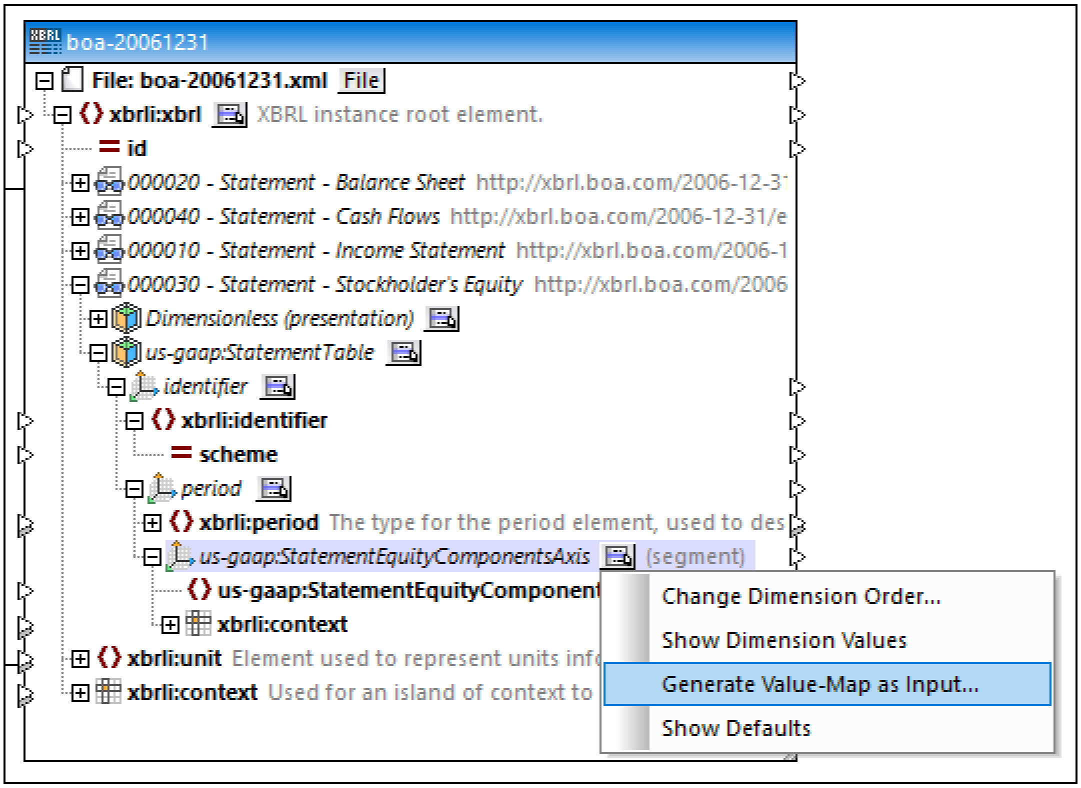


4.8.8.3 Generating Value-Maps for Hypercube Explicit Dimensions

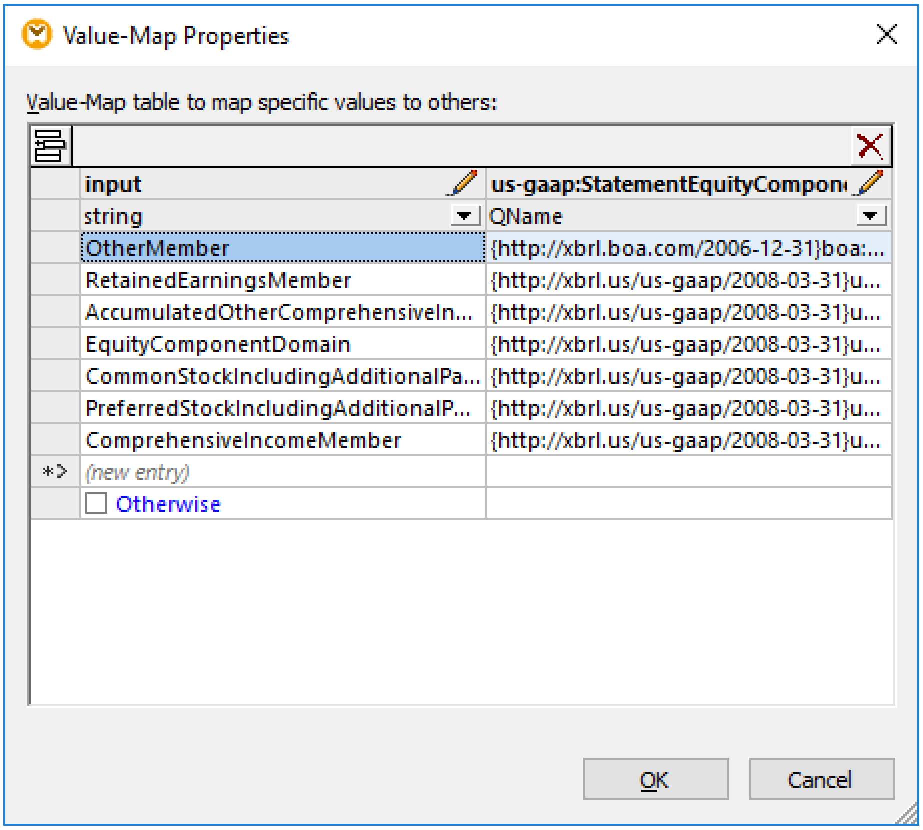
This option transforms input data of any type into a valid `QName` in the target component. In other words, the input string is converted into the prefixed name (`QName`).

To generate a value-map lookup table for the selected hypercube dimension:

1. Make sure that the dimension values are not visible for the specific hypercube dimension.
2. Click the icon of the specific **dimension** and select the "Generate Value-Map as input" option in the popup menu.

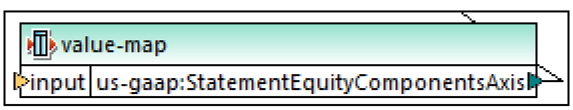


This opens the Value-Map Properties dialog box containing automatically generated input and output values based on the dimension default domain and domain members, as defined in the taxonomy.



3. Edit the input values if necessary, and click OK to insert the value-map component. Note that you can edit the column header text (by double-clicking the header) to make them shorter, or more descriptive if you wish.

   This inserts the value-map function, showing the input and output parameter names. The output connector is automatically connected to the domain element of the target.



4. Connect the source item that contains the input data to be transformed, to the input parameter of the function.

For more information about Value-Map components, see Using Value-Maps[884].

4.8.9 XBRL Tables

If your XBRL taxonomy references the table linkbase, MapForce can be configured to display the node types applicable to the table linkbase (such as tables or breakdowns). This enables you to map data to or from facts displayed in rendered tables. To show the *Tables from table linkbase* view in the XBRL component, take the steps below:

1. Go to the **Insert** menu and click **XBRL Document**.
2. Browse for the XBRL instance file. See Adding XBRL Files as Mapping Components[745].

3. Click the (**Show Context Menu**) button next to the root element. Then click **Select Structure Views**[746].
4. Select *Tables from table linkbase* and click **OK**.

For information about other views, see Selecting Structure Views[746].

Notes

- When reading data from an XBRL component (that is, if the XBRL component is a source component), you can choose any combination of structure views from the context menu . However, when writing data to XBRL, if the **Tables from table linkbase** view is selected, it is not possible, for technical reasons, to write data to any other view except this one.
- The **Tables from table linkbase** view requires that BUILT-IN[27] be set as a transformation language.

To render information from the table linkbase, MapForce normally uses the structural model, with the following exceptions:

- Merged rule nodes are visible.
- Roll-up nodes without siblings do not form a separate hierarchy level.

The structural model is one of the three data models defined by the Table Linkbase Specification 1.0, §4. To identify the parts of the structural model referenced by the mapping, MapForce also uses information from the definition model.

Since a MapForce component structure is one-dimensional, whereas tables can have up to three dimensions (`x`, `y`, `z`), MapForce displays table dimensions by nesting all breakdowns within each other, in the following default order: `z`, `y`, `x`. As such, the purpose of breakdowns nodes in MapForce is to inform you where each breakdown starts.

Filing indicators (EBA)

According to the EBA Filing Rules, filing indicators show the user's intention to report or not to report a specific reporting item. EBA-related taxonomies use filing indicators when tables have a label with the linkrole `http://www.eurofiling.info/xbrl/role/filing-indicator-code` (EBA Filing Rules §1.6.3). In addition, the concept for the filing indicator facts must exist: `find:fIndicators` (EBA-compatible style).

The `Filing Indicators` node is shown in the target XBRL component if (i) the *Tables from table linkbase* view is enabled in an XBRL component, and (ii) the taxonomy uses filing indicators as specified above. The `Filing Indicators` node is shown as a child of the `xbrli:xbrl` root element, between the `id` attribute and the first table node.

MapForce allows you to auto-generate filing indicators for target XBRL components. The `Filing Indicators` node has no mappable datapoints, only a button to change the setting[747]. Clicking the button allows choosing how filing indicators are to be generated: (i) *None*, which means no filing indicators will be generated in the output (default option), and (ii) *Auto-generate*, which means filing indicators will be generated for all relevant tables in the component. Tables that have any input connections in the subtree will generate a positive filing indicator `filed="true"`; unconnected nodes will generate a negative filing indicator `filed="false"`.

In this section

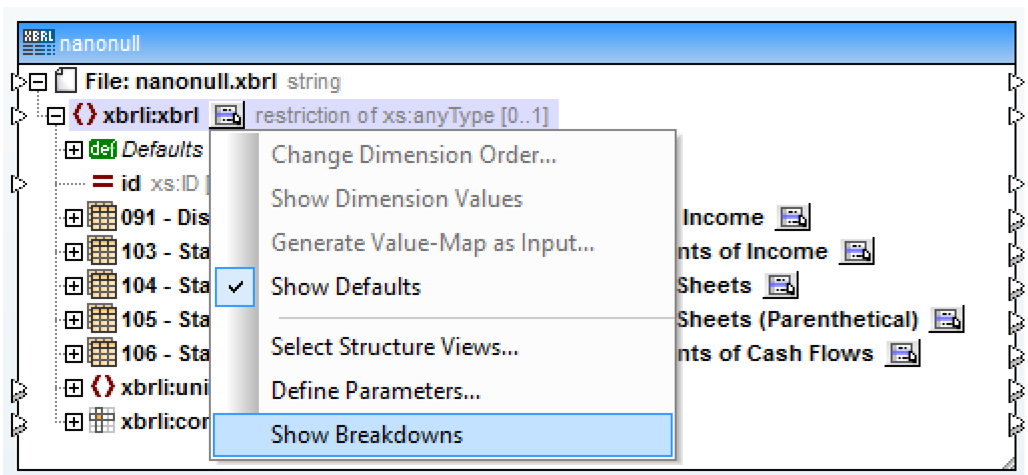
This section shows how to work with entities from the table linkbase and is divided into the following topics:

4.8.9.1 Showing or Hiding Breakdowns

In XBRL documents that reference the table linkbase, you can select whether to view or hide breakdowns in the component.

To show or hide breakdowns:

1. In the XBRL component, locate a node that contains the **Show Context Menu** ( ) button to the right (for example, the root node).



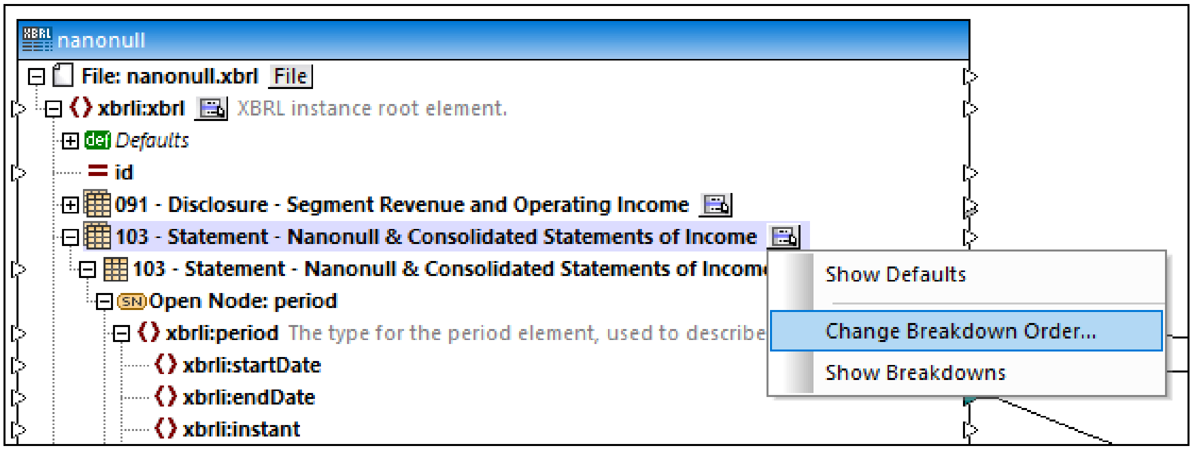
2. Click the **Show Context Menu** ( ) button, and then click **Show Breakdowns**.

4.8.9.2 Changing the Order of Breakdowns

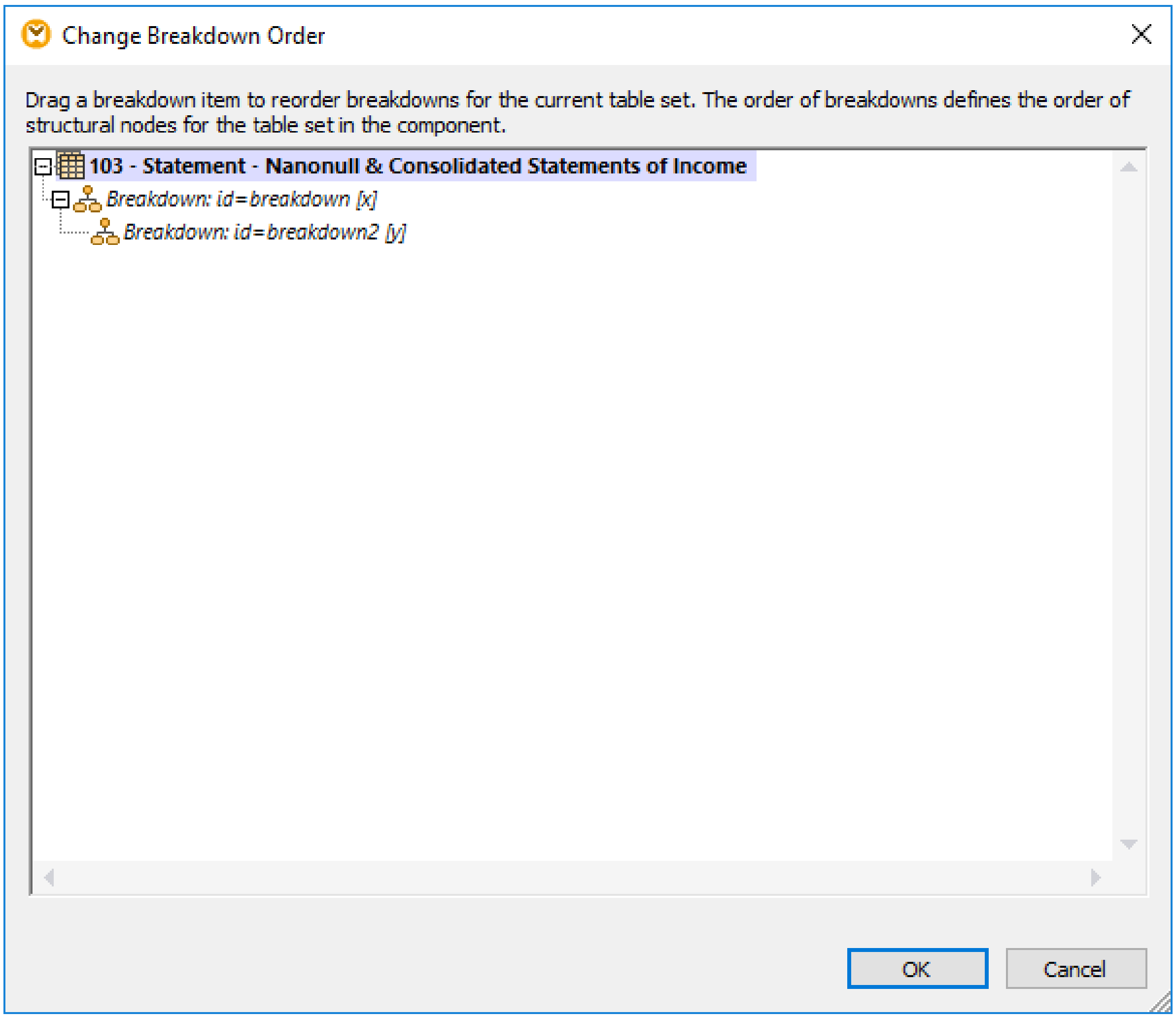
In XBRL documents that reference the table linkbase, you can change the order of breakdowns.

To change the order of breakdowns:

1. On the XBRL component, locate the node for which you want to change the breakdown order.
2. Click the **Show Context Menu** ( ) button to the right of the node, and then click **Change Breakdown Order**. (If the **Change Breakdown Order** menu option is not available, this means that this option is not meaningful for the selected node.)



3. Drag the breakdowns to the desired location (for example, in the following screen shot, you can drag the *y* breakdown on top of the *x* breakdown).



4. Click **OK**.

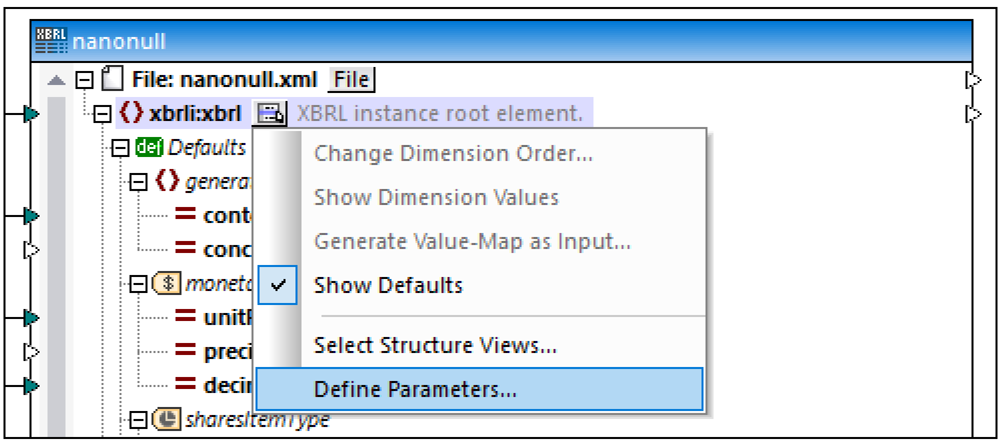
4.8.9.3 Working with Parameters

In XBRL documents that reference the table linkbase and contain parameters (Table Linkbase Specification 1.0, §5.3), you can change the parameter values. If supported by the context, you can also append new parameter values to existing parameters.

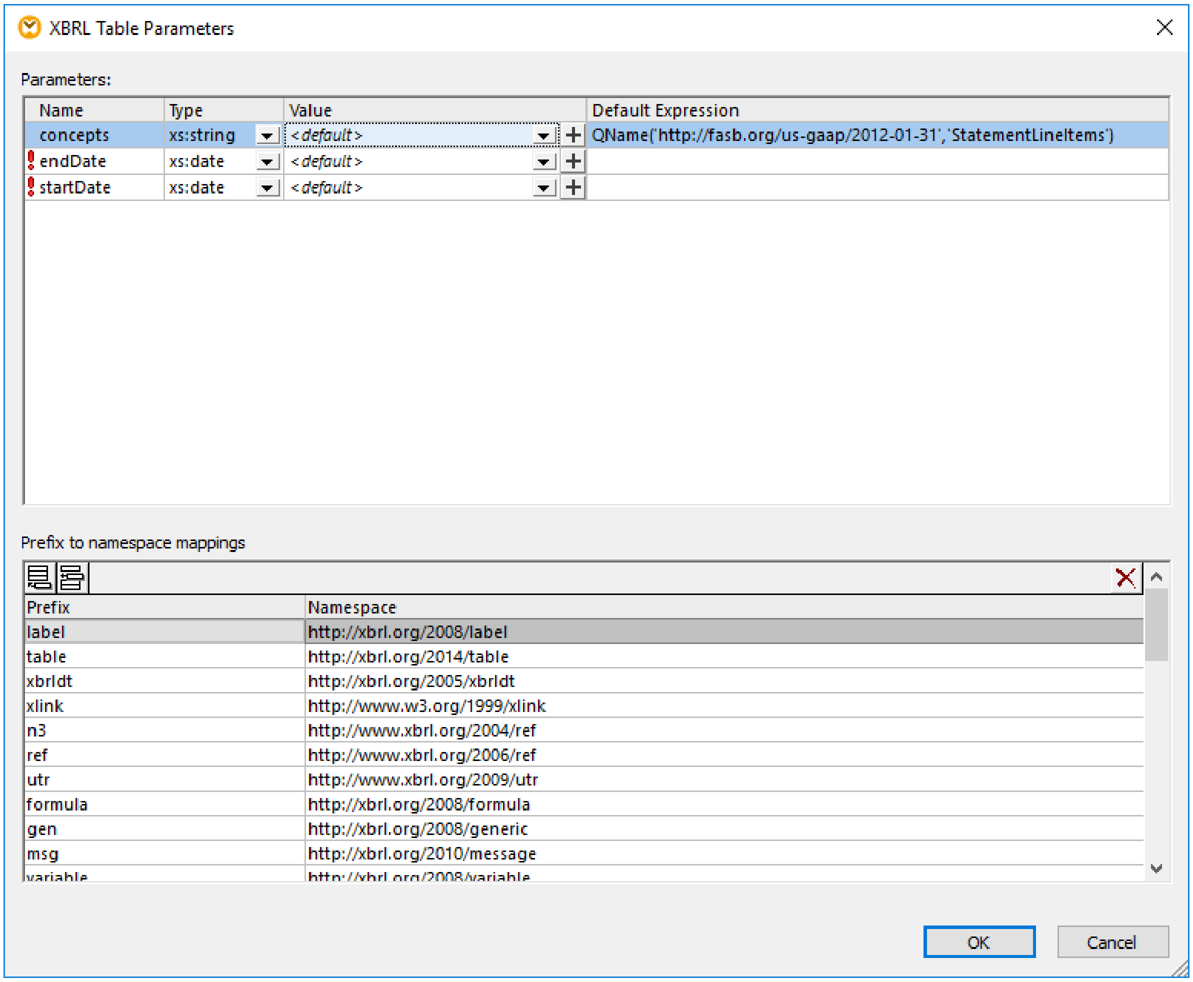
You can change parameter values from the XBRL Table Parameters dialog box (see instructions below). This dialog box displays parameters defined both at global level (anywhere in the Discoverable Taxonomy Set) and in individual tables. When you change any parameter value and close the dialog box, MapForce attempts to regenerate all tables in the component and merge the structure tree nodes. If the structure cannot be regenerated due to missing parameters, an error node is displayed in the relevant node of the component. For more detailed information about the error, check the Messages window 36.

To change parameter values:

1. Click the **Show Context Menu** ( ) button to the right of the root node, and then click **Define Parameters**. If the **Define Parameters** menu option is not available, the table linkbase does not contain parameters.



2. Select the parameter record, and type the new value in the Value column. Mandatory parameters have a red exclamation sign ( ) icon.



To append parameter values to existing parameters:

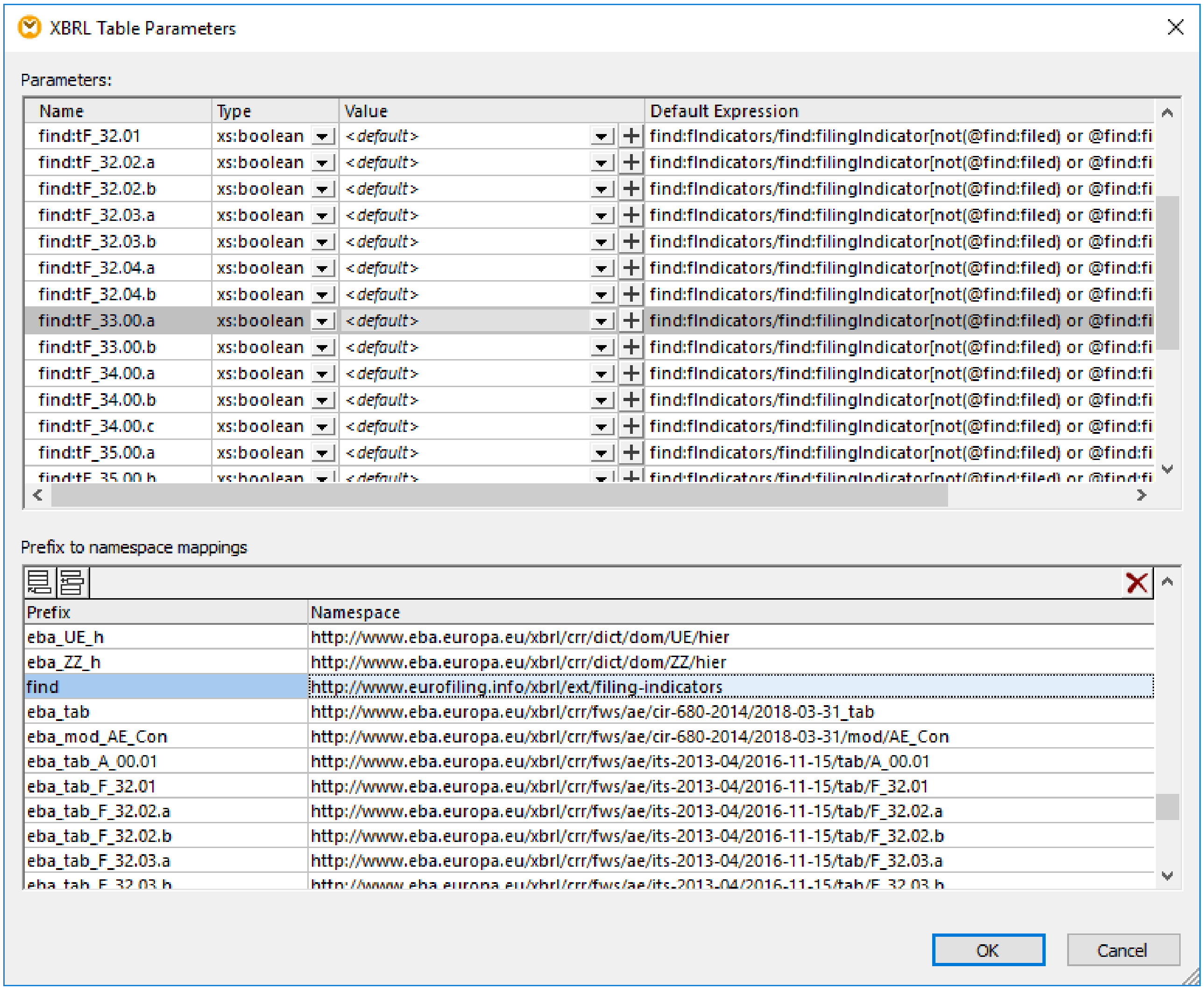
- Select the parameter record, and then click the **Add parameter value** ( ) button under the Value column. Note that appending a value is possible only if the parameter type supports adding multiple values.

To remove an appended value:

- Click the **Remove Value** ( ) button next to the value you want to remove.

Prefix to namespace mappings

Table parameter names consist of an optional namespace and a local name (that is, they are QNames and take the form *namespace_prefix:localname*). MapForce detects any such mappings automatically and displays them in the lower area of the XBRL Table Parameters dialog box. For example, in the image below, prefix **find** maps to **http://eurofiling.info/xbrl/ext/filling-indicators**.



If you need MapForce to parse a parameter prefix according to some custom namespace, you can change the existing prefix-namespace pair, or add a custom one. To create name-prefix maps, click the **Insert** or **Append** buttons. Note that this does not influence prefix names of XBRL instance items if your mapping writes data to an XBRL instance. This merely serves as an indication to MapForce how to parse and regenerate the XBRL structure after you close the dialog box.

4.8.10 XBRL Mapping Examples

This section includes the following XBRL mapping examples:

- Microsoft Access to XBRL 789 (shows how data from a Microsoft Access database is mapped to an XBRL taxonomy, producing a valid XBRL instance file)
- Microsoft Excel to XBRL 790 (shows how Microsoft Excel spreadsheet data is mapped to a taxonomy, producing a valid XBRL instance file).

4.8.10.1 Microsoft Access to XBRL

This example is available as **DB_to_XBRL.mfd** in the following folder: **`C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples`**. The example uses various filters and functions to extract the database data.

The taxonomy **nanonull.xsd** is derived from US:GAAP. The mapping creates an XBRL output instance which contains all contexts, concepts, units and footnotes for one Disclosure and three Statements.

The report **"091 - Disclosure - Segment Revenue and Operating Income"** shows how MapForce can map dimension values. The hypercube "us-gaap:ScheduleOfSegmentReportingInformationBySegmentTable" contains an explicit dimension "us-gaap:StatementOperatingActivitiesSegmentAxis".

Its domain has been extended in the taxonomy by the three dimension values "nanonull:USA", "nanonull:Europe" and "nanonull:Asia" The mapping shows how a value-map maps the values of the database column "Name" of the table "Region" to the required dimension values of type QName.

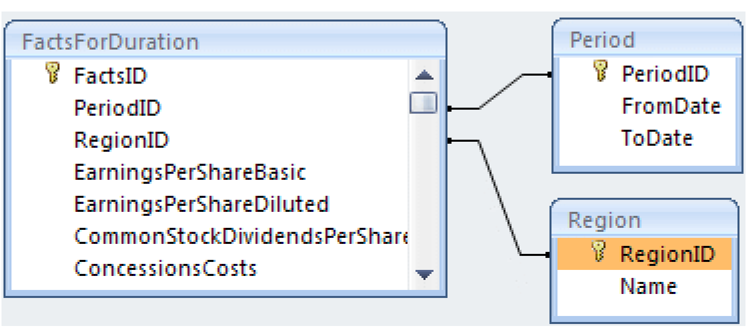
The report **"106 - Statement - Nanonull and Consolidated Statement of Cash Flows"** illustrates how MapForce can be used to write facts into the output instance which relates to both duration and instant periods.

As the mapping shows, proper reporting of facts such as "Cash and cash equivalents at beginning (end) of period" can be achieved by duplicating the period item in the hierarchy structure.

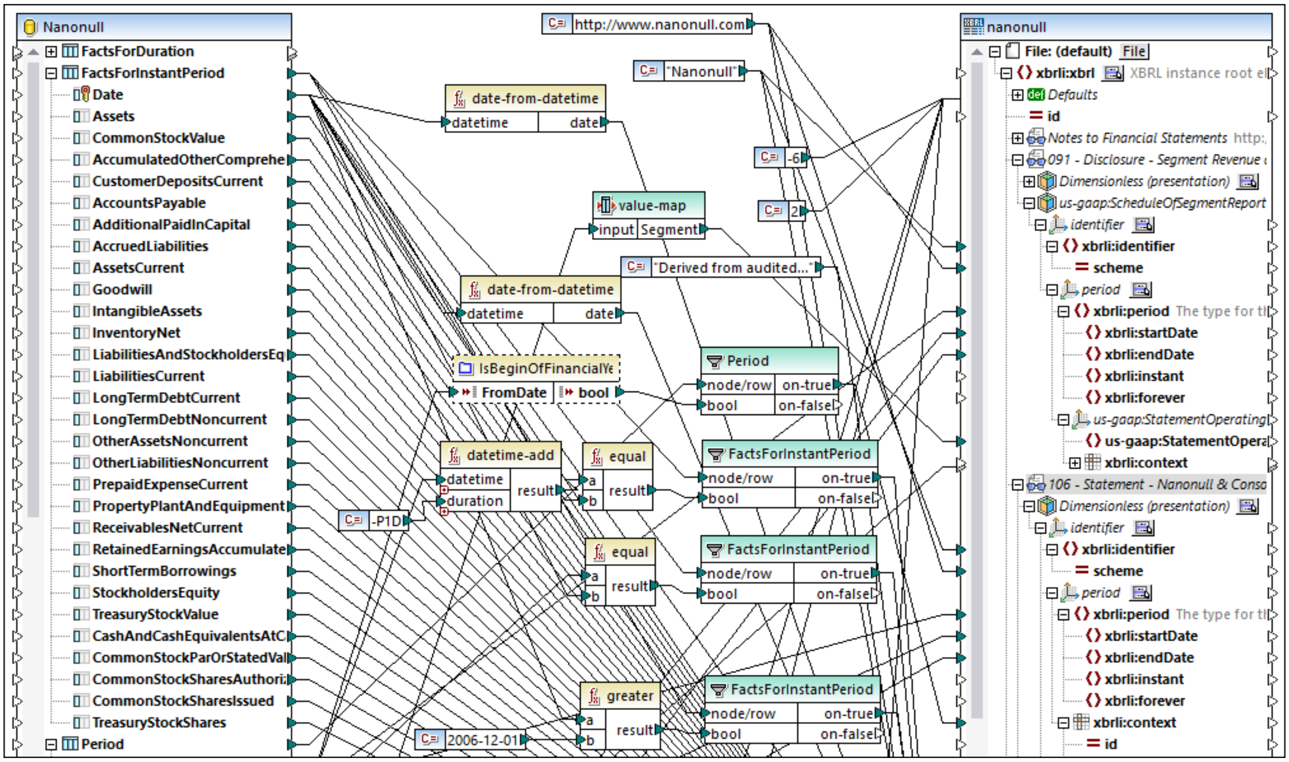
The mapping defines two units in the output instance, "USD" and "perShareItemType". The xbrli:unit element must be duplicated to do this. The related measure elements are created using the functions "xbrl-measure-currency" and "xbrl-measure-shares" from the XBRL library.

The facts in the database tables have been split up depending on whether they relate to an instant or duration period.

- The table **FactsForInstantPeriod** is a flat table of values.
- The table **FactsForDuration** is hierarchical and each fact it contains, relates to a specific PeriodID as well as a RegionID.



The Period table uses FromDate and ToDate fields to define the start and end period dates; while the Region table relates each of the facts to a specific region, i.e. Asia, Europe or USA.



DB_to_XBRL.mfd mapping

Mandatory XBRL items needed in a XBRL instance file:

- **unitRef** and either **decimals** or **precision** in monetary concepts
- xbrli:**identifier** and **scheme** of the identifier dimension
- xbrli:**period** and either xbrli:**instant** or xbrli:**startDate**/xbrli:**endDate** elements
- xbrli:**id** and xbrli:**measure** in the xbrli:**unit** element

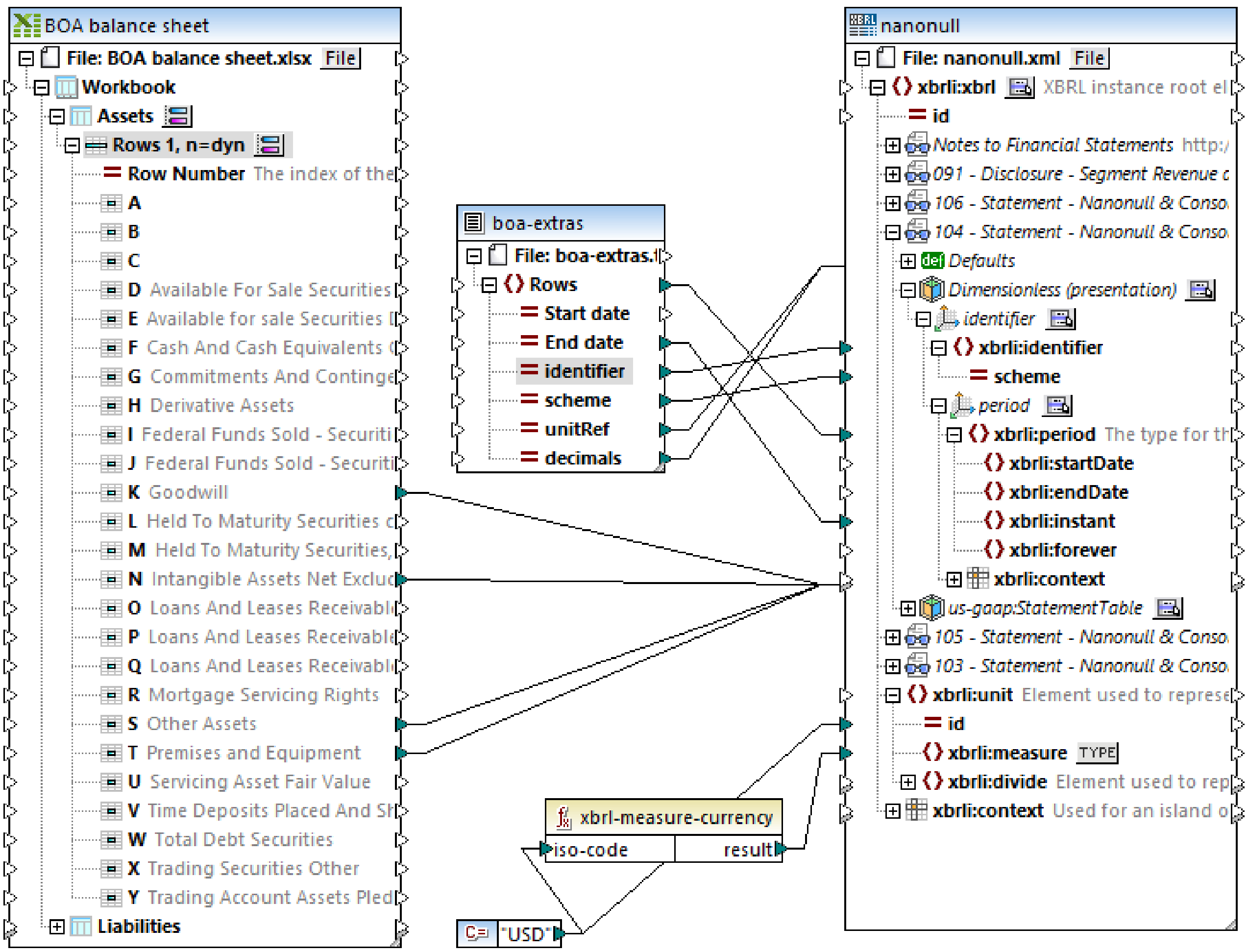
See also Microsoft Excel to XBRL 790.

4.8.10.2 Microsoft Excel to XBRL

This example shows how data from an Excel sheet are mapped to an XBRL taxonomy to generate an XBRL instance file. The actual mapping design file is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\boa-balance-sheet.mfd**.

To keep the example simple, only certain columns from the "Assets" worksheet have been mapped. The result of the mapping is a valid XBRL instance document containing some Assets data for a particular instant, Dec. 31st 2012.

The XBRL taxonomy for the target component is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\\Tutorial\Taxonomy\Nanonull.xsd**. This is a demo taxonomy intentionally simplified; unlike a real taxonomy, it contains only a few mappable XBRL items. The target XBRL component is set to display the presentation and definition linkbases (hypercubes), see Selecting Structure Views 746.



boa-balance-sheet.mfd mapping

The Excel columns that are mapped directly to the XBRL items/facts in the taxonomy component are as follows: "Goodwill", "Intangible Assets Net Excluding Goodwill", "Other Assets", and "Premises and Equipment".

Note that the item names in the source and target components are not, and do not need to be, identical. The name of the target taxonomy item determines the name used in the resulting XBRL instance. However, having identical source and target item names does have the advantage that you can use the **Autoconnect Matching Children** option (see Connecting Matching Children 95).

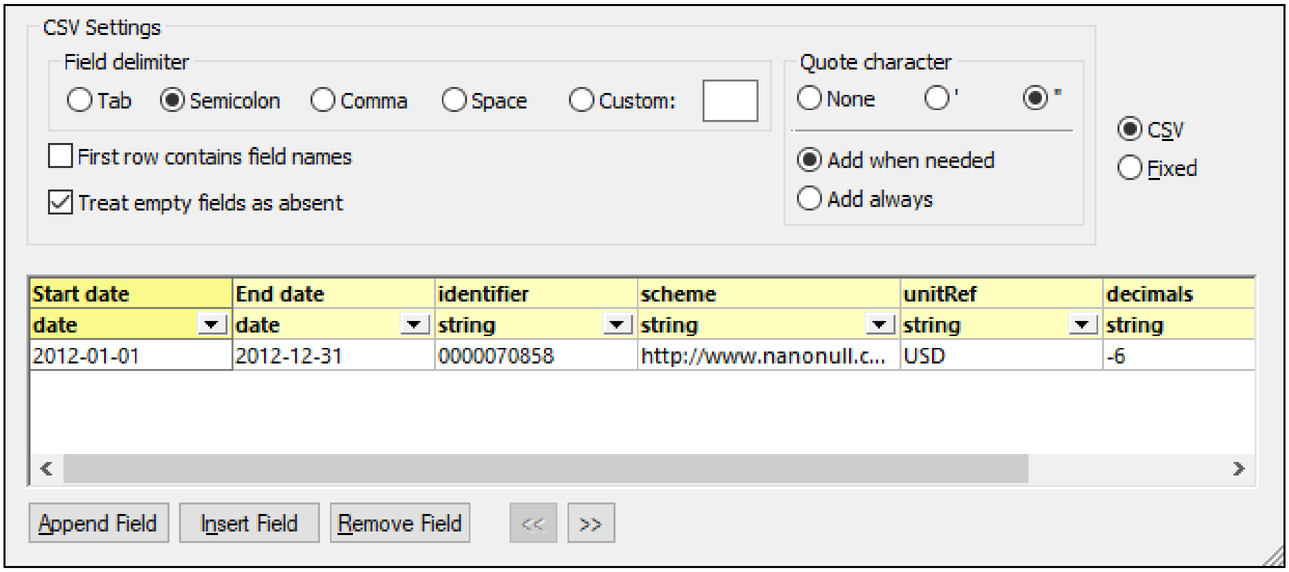
To generate a valid XBRL instance file, several mandatory items must be mapped. Some mandatory items are supplied by a text file (**boa-extras.txt**, the middle component on the mapping) whose fields are mapped to child elements of the **Defaults** item in the **Dimensionless (presentation)** hypercube.

The **Defaults** item is inserted by right-clicking a hypercube dimension and selecting **XBRL | Show defaults**, or by clicking the icon and selecting the option there. For more information, see Working with XBRL Defaults 770.

The following XBRL items are mapped in order to generate a valid XBRL instance file:

- **unitRef** and either **decimals** or **precision** in monetary concepts
- xbrli:**identifier** and **scheme** of the identifier dimension
- xbrli:**period** and either xbrli:**instant** or xbrli:**startDate**/xbrli:**endDate** elements
- xbrli:**id** and xbrli:**measure** in the xbrli:**unit** element

A text file supplies the data for some of the mandatory items; you can view or change these values by right-clicking the "boa-extras" component and selecting **Properties** from the context menu.



A constant supplies data for the **id** and **measure** items, which are at the base of the taxonomy component. The `xbrl-measure-currency` function converts the currency code into a value suitable for the **xbrli:measure** field. For more information about MapForce functions, see the Functions 904 chapter.

Click the **Output** button to run the mapping and preview the resulting XBRL instance file.

- To check the validity of the XBRL instance, click the **Validate Output** button of the Output toolbar. Messages or warnings are displayed in the Messages window.
- To define various settings applicable to the generated output, click the **Text view settings** button .

4.9 JSON

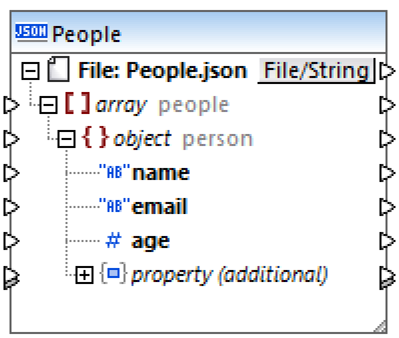
Altova website: JSON mapping

JSON (JavaScript Object Notation) is a very popular data format for storing and transporting data. In MapForce Enterprise Edition, you can map data from any formats supported by MapForce to JSON, and vice versa. You can also serialize JSON data as a string (for example, for writing to a database) or parse a JSON string into a JSON structure with the help of Parse/Serialize components. For more information, see Parsing and Serializing Strings[1324].

> To use JSON files in mappings, the mapping language must be set to one of the following: Built-in, C#, Java. For more information, see Transformation Languages[27].

Similar to other component types supported by MapForce, a schema file is required before you can read or write JSON data. If you don't have a schema file, MapForce can generate one automatically. See Adding JSON Files as Mapping Components[795]. If you supply a JSON schema file not generated by MapForce, then MapForce expects it to be of version Draft 04, Draft 06, Draft 07, Draft 2019-09, Draft 2020-12 or have no version at all. For more information about JSON Schema versions, see the JSON Schema Specification.

The following is an example of how a JSON structure appears in MapForce:



JSON Component in MapForce

As illustrated above, the structure of nodes in the JSON component reflects the structure of the JSON instance file. The basic JSON types are conventionally represented in MapForce as shown below.

Type	MapForce representation
Array	[ ]
Boolean	01
Null	Ø
Number or integer	#
Object	{ }

Type	MapForce representation
String	"AB"

There are also special cases, described below:

Case	Description
additionalProperties, patternProperties	The node appears on the JSON component under any object whose `additionalProperties` property is `true` or not present in the schema. It allows you to map to or from properties not explicitly listed in the schema (see also Example: Mapping from JSON to CSV 802). This node can also appear for objects having the `patternProperties` property.
Subtypes	JSON schema allows defining subtypes for objects and arrays (`anyOf`, `allOf`, `oneOf`). MapForce displays such subtypes using special structure nodes ( ) that do not have a direct representation in the JSON instance file. If your mapping is reading from a JSON file, such subtype nodes provide a value only if the current input value is valid according to the subtype schema. If your mapping is writing to a JSON file, make sure that you choose the correct subtype to fill. Filling multiple subtypes may lead to duplicate object properties and, thus, may result in invalid output JSON files.
Multiple types at the same location	JSON schema allows multiple types to occur at the same location. In such cases, the MapForce component displays separate structure nodes for all basic types that can occur at that location.
Type names	MapForce displays the `title` and `description` properties of types in the JSON schema in the "type" and "annotation" fields, if available. If `title` is absent, MapForce may also use part of the URI from the `$ref` property as a type name.
Arrays containing mixed item types	If an array has different types of items in the JSON schema (for example, both strings and numbers), MapForce displays an "item" node for each item type. When writing to a JSON file, this enables you to create arrays which contain items of different types.
Arrays defined as tuples	If an array has items whose type is assigned by position in the JSON schema, MapForce displays the zero-based index of the item as separate structure nodes (for example **item[0]**, **item[1]**, and so on. When writing to a JSON file, this enables you to specify the type of each individual item in the array by its zero-based index.

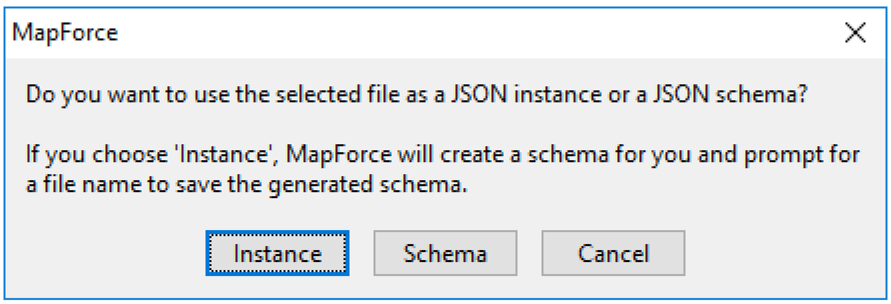
4.9.1 Adding JSON Files as Mapping Components

Reading or writing JSON instance files with MapForce requires a JSON schema file. You can either generate the JSON schema file with MapForce, or supply it as a custom file, as shown below.

At the time when this documentation is written, there is no formal convention for naming JSON schema files. Therefore, to distinguish JSON schema files from instance ones, you can use the **schema.json** extension for your JSON schema files (for example, **Example.schema.json**), and MapForce will recognize them as such.

To add JSON files as mapping components:

1. On the **Insert** menu, click **JSON Schema/File**. A dialog box may prompt you to select the type of the JSON file (schema or instance). This dialog box does not appear if MapForce positively identifies the file as a JSON schema.

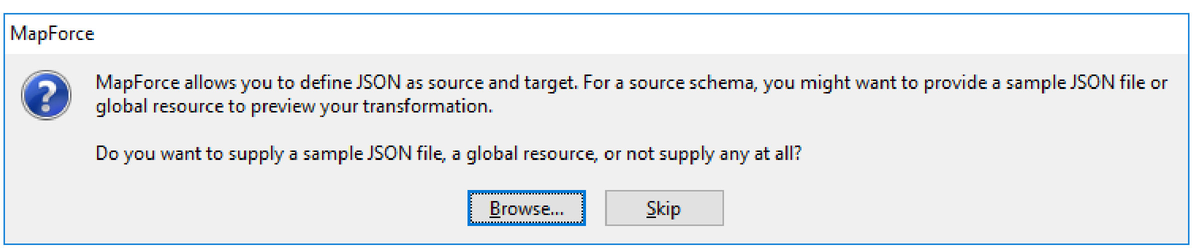


2. Select **Schema** or **Instance**, as required.

> The size of the JSON instance file must not exceed your system's available memory.

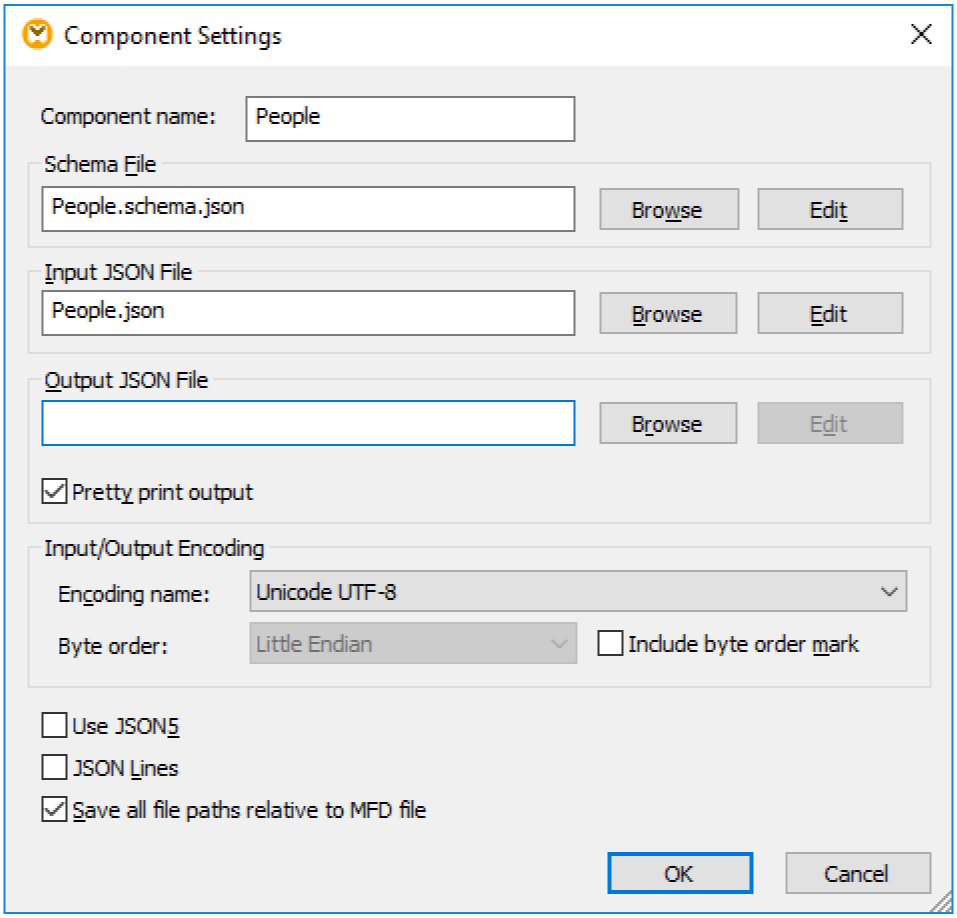
If you select **Instance**, MapForce generates automatically the schema based on the instance file, and prompts you for a location to save the schema.

If you select **Schema**, MapForce prompts you to specify a sample JSON file or a global resource (see Global Resources[1397]). This is required to preview the transformation and is meaningful only when you want to read data from a JSON file. Click **Skip** if your mapping is going to write data to JSON.



4.9.2 JSON Component Settings

To change the settings of a JSON component, right-click the JSON component header, and then click **Properties** (alternatively, double-click the JSON component header).



JSON Component Settings dialog box

The available settings are as follows.

Component name	The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change the name of the component, be aware of the following: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.

Schema file	Specifies the file name and path of the schema file. To change the location of the file, click **Browse** and select the new file. To edit the file in your JSON editor (for example, XMLSpy), click **Edit**.
Input JSON File	This text box is meaningful only if the mapping reads data from a JSON file. It specifies the file name and path of the input JSON instance. To set or change the location of the file, click **Browse** and select the new file. You can also enter a relative path if the **Save all file paths relative to MFD file** check box is selected (see below). To edit the file in your JSON editor (for example, XMLSpy), click **Edit**.
Output JSON File	This text box is meaningful only if the mapping writes data to a JSON file. It specifies the file name and path where the target JSON instance will be created when the mapping is executed by MapForce Server. This is also the default location when you save the output from the **Output** tab. To change the location of the file, click **Browse** and select the new file. You can also enter a relative path if the **Save all file paths relative to MFD file** check box is selected (see below). To edit the file in your JSON editor (for example, XMLSpy), click Edit.
Pretty print output	Reformats your JSON document when the mapping is executed, in order to give it a structured display. Each child node is offset from its parent by a single tab character.
Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. JSON files are expected to have UTF encoding (see https://tools.ietf.org/html/rfc7159#section-8.1). Other encodings are considered non-standard.
Use JSON5	Select this check box if the input JSON file from which the component reads data uses JSON5 syntax. In a target component, selecting the check box instructs MapForce to create a .json5 file where object keys are unquoted if they are valid ECMAScript5 identifiers. See also JSON5 Support 798.
JSON Lines	Select this check box if you would like to read data from a source file in JSON Lines format, or if you want to write data to a target file in JSON Lines format. For further information, see JSON Lines 799.

Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the following files: • The JSON schema file • The JSON input file • The JSON output file See also Using Relative Paths on a Component[85].

4.9.3 JSON5 Support

JSON5 is a proposed extension to JSON that aims to make it easier for people to write JSON files by hand. Importantly, JSON5 is not an official successor to JSON (for more information, see https://json5.org/).

JSON5 files use the .json5 extension; however, at the time when this documentation is written, there is no official MIME type for .json5 files. The .json5 extension is nevertheless recognized by MapForce when you add such files to the mapping. If you deploy the mapping to FlowForce Server (see Deploying Mappings to FlowForce Server[1381]), the MIME type of .json5 files will be reported as "`application/json5`".

Prerequisites

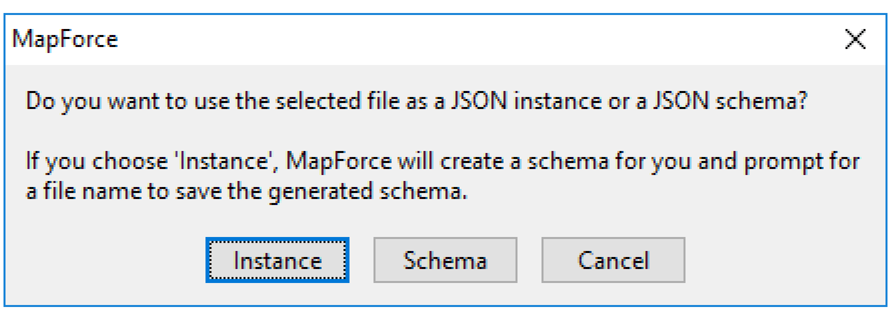
- In order to map data to or from JSON5 files, a valid JSON schema (which should not use JSON5 syntax) must be supplied to MapForce. If you don't have a JSON schema for your file, MapForce will generate it automatically, as shown below. The schema generated automatically by MapForce will be a JSON (not a JSON5) schema. It is also possible to generate a JSON schema for a .json5 file with XMLSpy.

Limitations

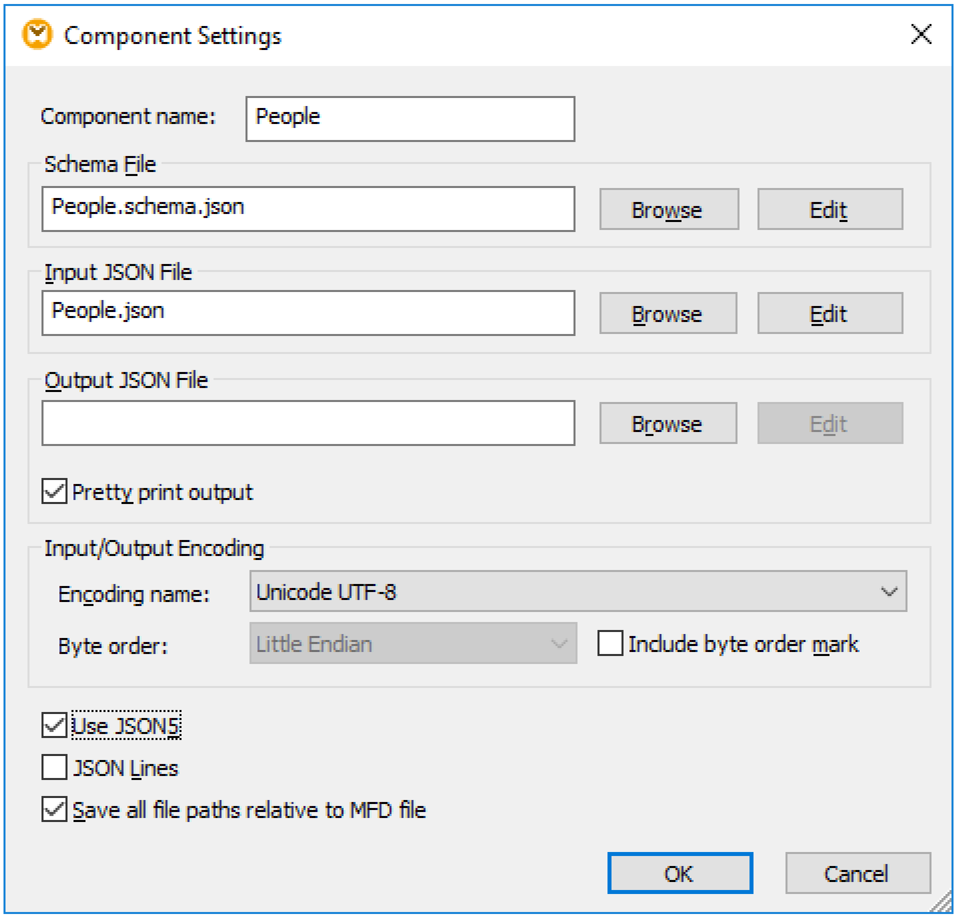
- Mapping comments to or from .json5 files is not supported.
- Because MapForce works with strict data types, it is not possible to qualify and map values such as `Infinity`, `-Infinity`, `NaN`, or `-NaN`.

How to map data to or from .json5 files

1. On the **Insert** menu, click **JSON**, and browse for the .json5 instance file (Alternatively, click the toolbar button). If you have a JSON schema file and prefer to supply the .json5 instance file later, browse for a valid JSON (not a JSON5) schema. This works in the same way as with plain JSON files (see Adding JSON Files as Mapping Components[795]).



2. After the JSON component has been added to the mapping, right-click its header, select **Properties**, and then select the **Use JSON5** check box from component properties (see also JSON Component Settings[796]). This explicitly instructs MapForce to treat the instance data as such.



To generate a JSON5 file with MapForce, select the **Use JSON5** check box as shown above, and connect the required input data to the JSON component on the mapping. When the **Use JSON5** check box is selected, MapForce will not surround the object keys by quotes, except where this would make the generated JSON5 instance invalid. The desired file extension (for example, "json5") can also be controlled from the dialog box illustrated above, by entering the appropriate value in the **Output JSON File** field.

4.9.4 JSON Lines

JSON Lines (http://jsonlines.org/) is a text format that extends JSON. You may find it convenient for storing or processing JSON data where new line delimiters are important, such as log files. Each line in a JSON Lines file stores a JSON value, for example:

```
{"name":"Alethia Alonso","email":"alethia@example.org"}
{"name":"Klaus Mauer","email":"klaus@example.org"}
{"name":"Natsuo Shinohara","email":"natsuo@example.org"}
```

With MapForce Enterprise Edition, you can read data in JSON Lines format and convert it to any other formats supported by MapForce. Likewise, you can convert data from any format supported by MapForce to JSON Lines format.

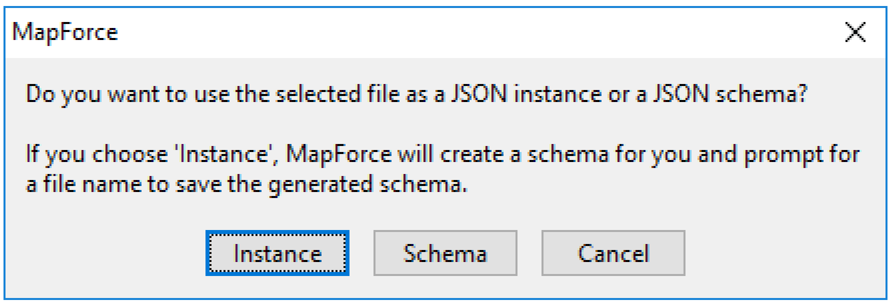
Note: In Web service calls created with MapForce, you can use standard JSON as request or response structure; JSON Lines or JSON5 structures are not supported.

To read or write data in JSON Lines format, MapForce requires a JSON schema, like with standard JSON files. The difference is that, in case of JSON Lines, each line is treated as a separate JSON file for validation purposes. The schema you provide to MapForce, therefore, validates each individual line as a JSON document.

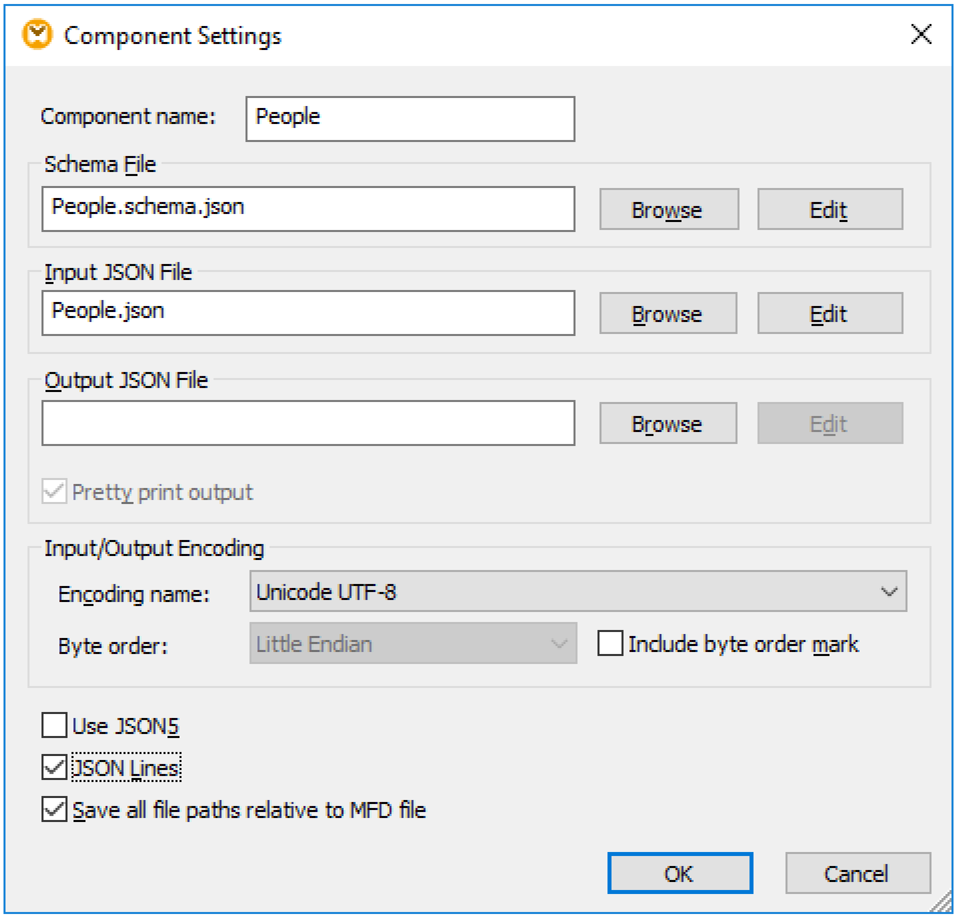
The file extension suggested by the authors of JSON Lines format is **.jsonl**.

How to map data to or from JSON Lines format

1. On the **Insert** menu, click **JSON**, and browse for the **.jsonl** instance file. Alternatively, click the toolbar button. If you already have a JSON schema file and prefer to supply the instance file later, browse for a valid JSON schema. This works in the same way as with plain JSON files (see Adding JSON Files as Mapping Components 795). Just have in mind that, in case of JSON Lines format, the JSON schema must describe the contents of *each line* in the instance file, not the file as a whole.



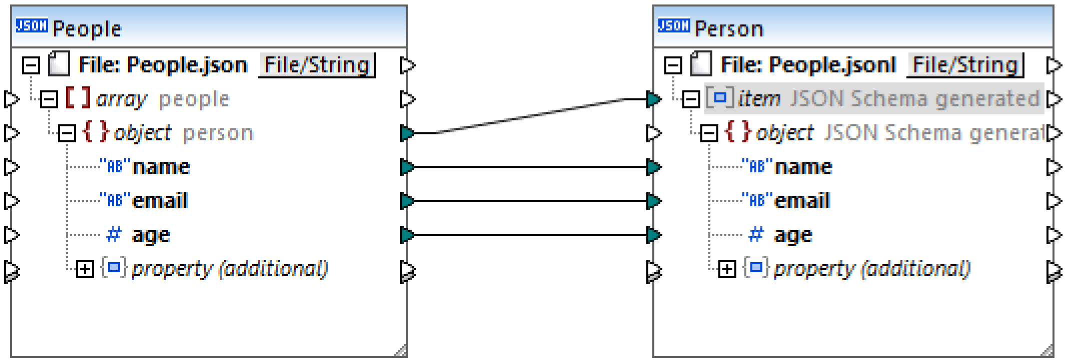
2. After the JSON component has been added to the mapping, right-click its title bar, select **Properties**, and then select the **JSON Lines** check box.



To set or change the JSON instance file from which the mapping reads data, enter the path in the **Input JSON File** text box. Filling this text box is not required if the mapping writes data to a JSON file.

To set or change the generated file name, enter the desired file name (for example, "somefile.jsonl") in the **Output JSON File** text box of the dialog box above. Filling this text box is not required if the mapping reads data from a JSON file. For reference to other settings, see JSON Component Settings 796.

When the JSON Lines check box is selected, the JSON component has a new item node on the mapping. By connecting any sequence of values to it (for example, the **Rows** item of a CSV source component), you can create multiple lines in the JSON Lines file (one line for each item in the sequence). In the mapping below, the source data is a standard JSON file and the target is a JSON Lines file. Thanks to the topmost connection, a new JSON line will be created in the target **People.jsonl** file for each object in the source **People.json** file when the mapping runs:



Likewise, if your mapping should read data from a JSON Lines file, you can connect the item node to the desired target item (for example, an XML item). This would create as many XML elements in the target as there are JSON lines in the source.

4.9.5 Example: Convert JSON to CSV

This example shows you how to read data from a JSON file and convert it to a comma-separated text file with MapForce. You will also learn how to map data from additional properties that might be present in JSON instance files like the one illustrated in this example. Additional properties refers to data that is present in the JSON instance file even though it is not defined by the schema. As a rule, you should avoid relying on additional properties and instead define all the properties in your JSON schema. However, in the event that the JSON instance contains additional properties, you can use the technique illustrated in this example.

This example is accompanied by a demo mapping file available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ReadJSON.mfd**. If you follow the steps below, you will learn how to create such a mapping file from scratch.

The source JSON file contains people records that must be converted to CSV. Notice the highlighted text: the first and the second person have an additional property that is not defined in the schema, namely `birthday`.

```
[
	{
		"name": "Alethia Alonso",
		"email": "altethia@example.com",
		"age": 35,
		"birthday": "4 July"
	}, {
		"name": "Klaus Mauer",
		"email": "klaus@example.com",
		"age": 57,
		"birthday": "31 August"
	}, {
		"name": "Natsuo Shinohara",
		"email": "natsuo@example.com",
		"age": 29
	}
]
```

People.json

The JSON schema used in the example is represented below. As the `$schema` keyword indicates, the schema validates JSON instances with respect to Draft 04 JSON Schema. It describes a `people` array which consists of multiple `person` objects. The `people` object must contain at least one `person` object to be valid. Each `person` object has a `name`, `age`, and `email` address as properties. Note that `name` and `email` are of type `string`, while `age` is of type `integer`. Also, the `name` and `email` properties are required, while `age` is optional.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "people",
  "type": "array",
  "items": {
       "title": "person",
       "type": "object",
    "required": [
      "name",
      "email"
    ],
    "properties": {
      "name": {
        "type": "string"
      },
      "email": {
        "type": "string",
        "format": "email"
      },
      "age": {
        "type": "integer"
      }
    }
  },
  "minItems": 1
}
```

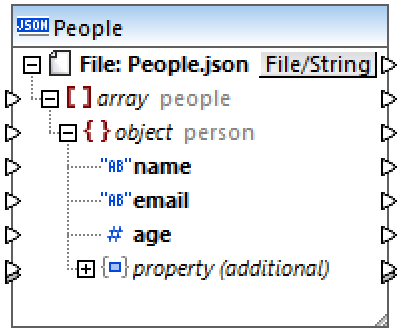
People.schema.json

Importantly, the JSON schema above does not have an `additionalProperties` property for the `person` object. When not present in the JSON schema, the `additionalProperties` property has the default value of **true**, which means that the object in the JSON instance can have as many additional properties as required, and still be valid. Therefore, a JSON instance file like **People.json** will validate successfully against this schema.

To achieve the mapping goals of this example, take the steps below.

Step 1: Add the source JSON file to the mapping

1. On the **Insert** menu, click **JSON Schema/File** and browse for the following file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\People.schema.json**. When prompted to specify an instance, select the **People.json** file in the same folder. At this point, the MapForce component looks as follows:

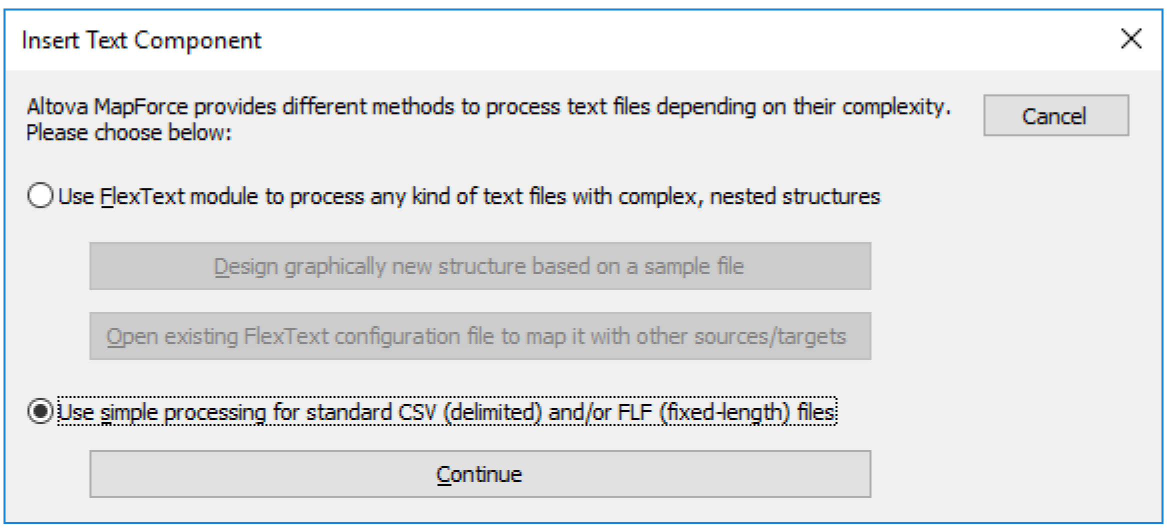


The structure of the MapForce component resembles that of the JSON file itself, with the exception of **property (additional)** node. This node indicates that the `additionalProperties` property of the `person` object is either missing or set to **true** in the schema. This means that the schema can contain custom additional properties, so MapForce displays the node in case you want to map from any additional properties of the object (the next steps show how to do this).

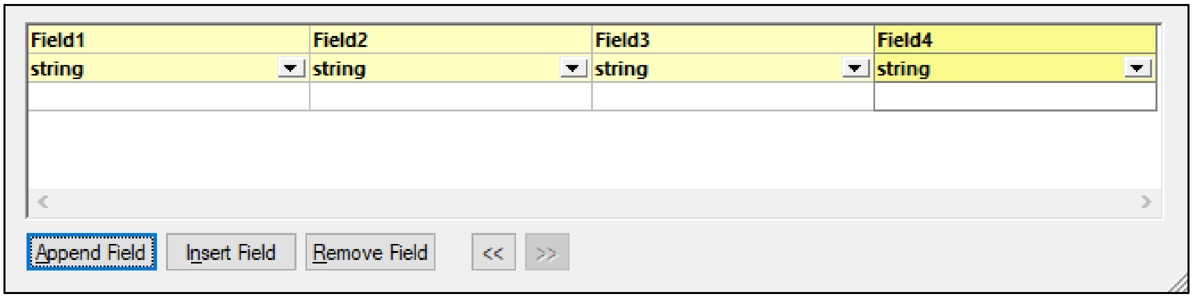
To change the source JSON file or the JSON schema file at any time, double-click the title bar of the JSON component and tweak the settings, see JSON Component Settings 796.

Step 2. Add the target CSV to the mapping

1. On the **Insert** menu, click **Text File**.



2. Select **Use simple processing for standard CSV**, and then click **Continue**.

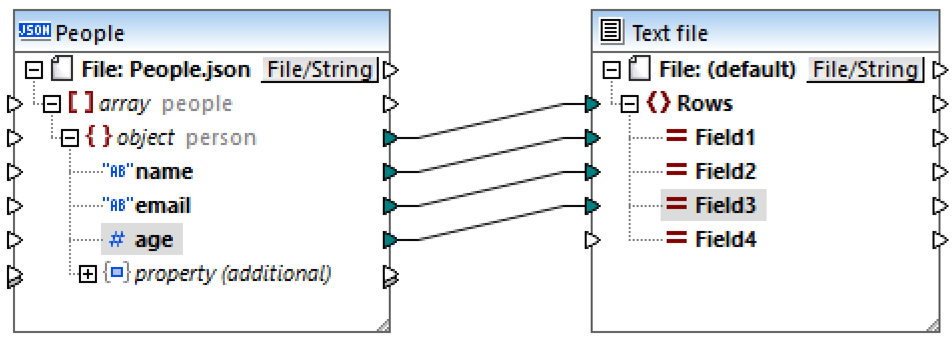


3. Click **Append Field** several times to add four CSV fields as shown above, and then click **OK**.

For more information about text components, see CSV and Text Files563.

Step 3: Draw the mapping connections

1. On the mapping, draw the connections between the JSON component and the CSV component as shown below.



For more information, see Working with Connections89.

Step 4: Preview the mapping

You can now quickly preview the transformation output by clicking on the **Output** tab. The result is:

```
Alethia Alonso,altethia@example.com,35,
Klaus Mauer,klaus@example.com,57,
Natsuo Shinohara,natsuo@example.com,29,
```

As the output shows, for each person object in the JSON file, a new row is created, and object properties are comma-separated, which is the intended behavior.

However, we have not mapped yet the `birthday` field which exists in the JSON instance file, even if it does not exist in the schema.

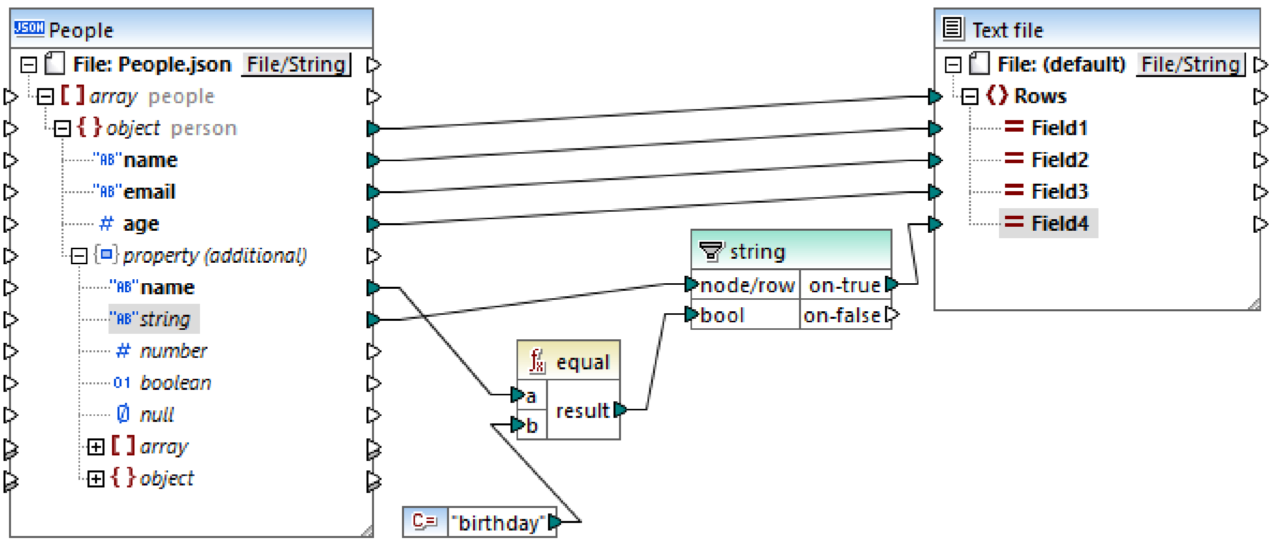
Step 5: Map additional properties

Recall that the field `birthday` is an additional property not defined in the schema. To read data from such fields, MapForce provides additional items on the JSON component.

In this example, in order to read data from the `birthday` field, add the following to the mapping:

1. A constant with the value "birthday", see Add a Constant to the Mapping [905].
2. A filter component, see Filters and Conditions [878].
3. The logical function **`equal`**, see Add a Function to the Mapping [905].

Next, connect the components as illustrated below. Connecting components this way instructs MapForce to look for an additional property with the name **birthday**, and, if a string value is found, copy it to **Field4** of the text component.



In the mapping above, the **name** item under **property (additional)** allows you to access the additional property by its name. The subsequent items (**string**, **number**, **boolean**, and so on) correspond to JSON data types. They can be used to access the actual JSON value of that type. In our example, the name of the additional property is "birthday" and the type is "string", which is why these two items are connected to the filter.

If you preview the transformation output at this time, the result is:

```
Alethia Alonso,altethia@example.com,35,4 July
Klaus Mauer,klaus@example.com,57,31 August
Natsuo Shinohara,natsuo@example.com,29,
```

As the output shows, the fourth CSV field now includes the value of the only additional property of type "string", which is, in this case, `birthday`. Also, since the third person does not have a birthday, no value is available in the CSV at the corresponding position.

4.9.6 Example: Convert Excel to JSON

This example shows you how to create a JSON document from an Excel file with MapForce. The source Excel workbook consists of two worksheets: **Office** and **Staff**. The **Office** worksheet provides information about the company's address, while the **Staff** worksheet lists all the employees.

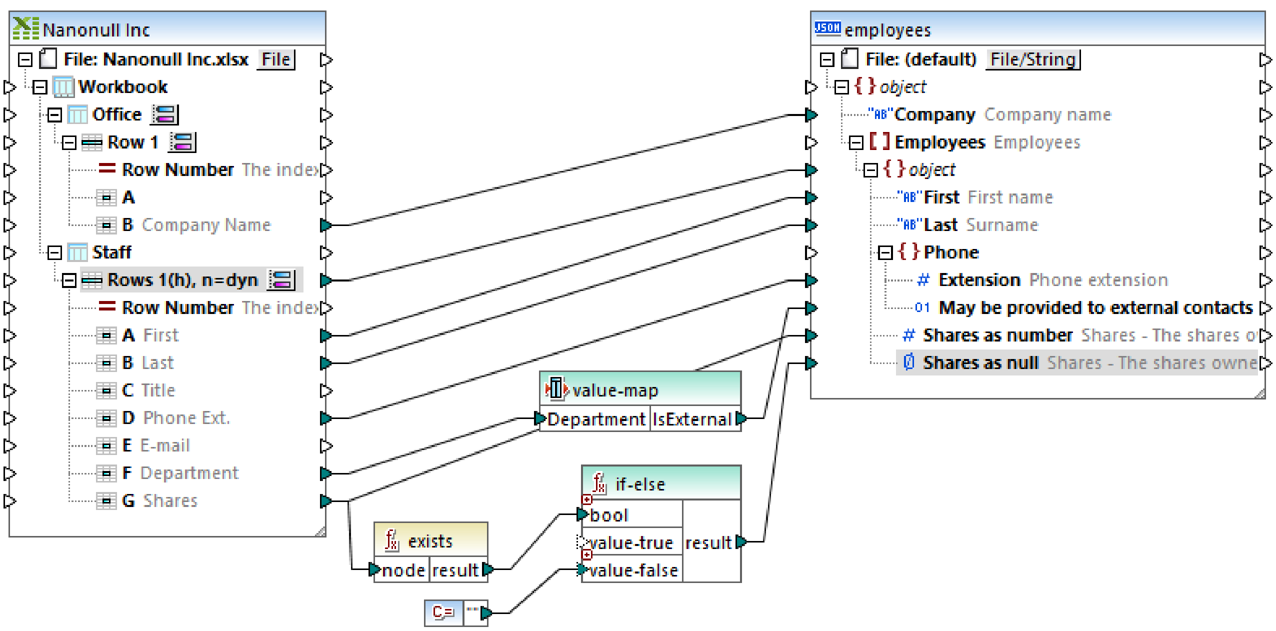
	A	B	C	D	E	F	G
1	First	Last	Title	Phone Ext.	E-mail	Department	Shares
2	Jessica	Bander	Support Engineer	241	j.band@nanonull.com	IT & Technical Support	0
3	Michelle	Butler	Software Engineer	654	m.butler@nanonull.com	Engineering	1500
4	Vernon	Callaby	Office Manager	582	v.callaby@nanonull.com	Administration	1500
5	Frank	Further	Accounts Receivable	471	f.further@nanonull.com	Administration	0
6	Liz	Gardner	Software Engineer	753	l.gardner@nanonull.com	Engineering	
7	George	Hammer	Web Developer	223	g.hammer@nanonull.com	IT & Technical Support	0

Staff worksheet

The business requirements are as follows:

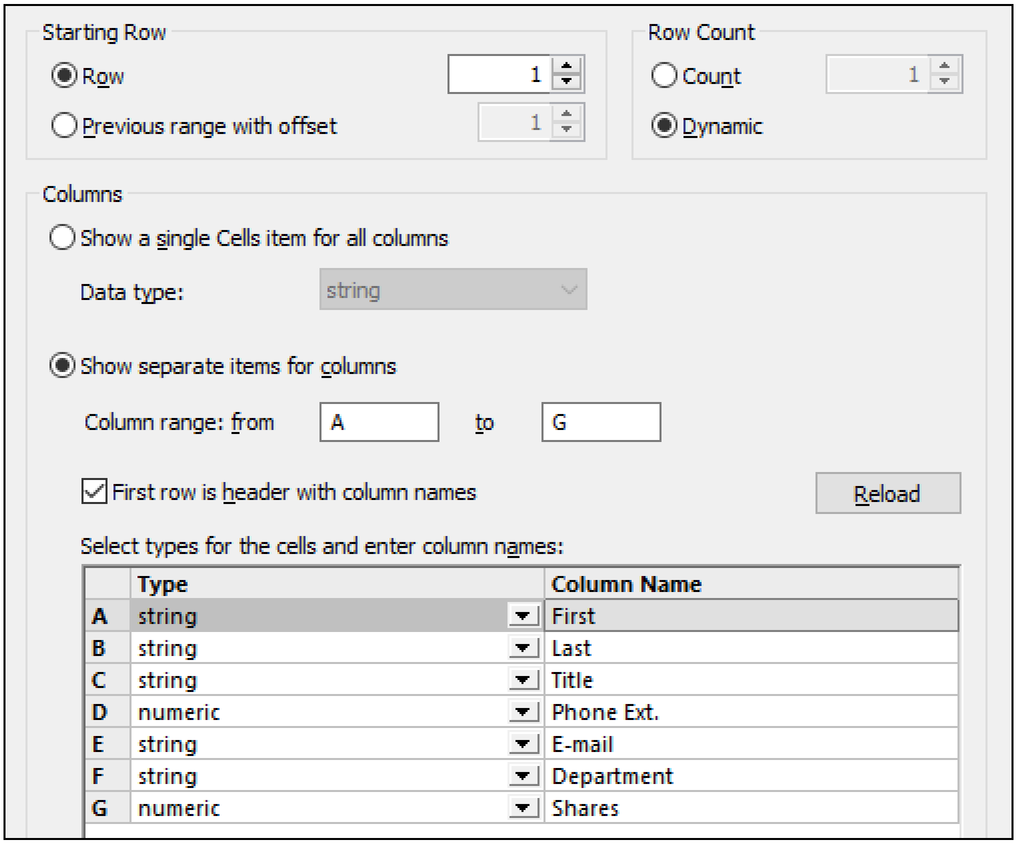
1. Extract data from the Excel file and convert it into a JSON document.
2. The JSON file must specify, for each employee, whether their phone extension can be shared for external contracts. Only phone extensions of employees from the Marketing and Administration departments can be shared externally.
3. The shares of each employee must be indicated in the JSON file as a numeric value, which may include zero. If not known, shares must be indicated as `null` in the JSON file.

The mapping that accomplishes the goal above is illustrated below. This mapping is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\\Excel_Company_To_JSON.mfd**.



Excel_Company_To_JSON.mfd

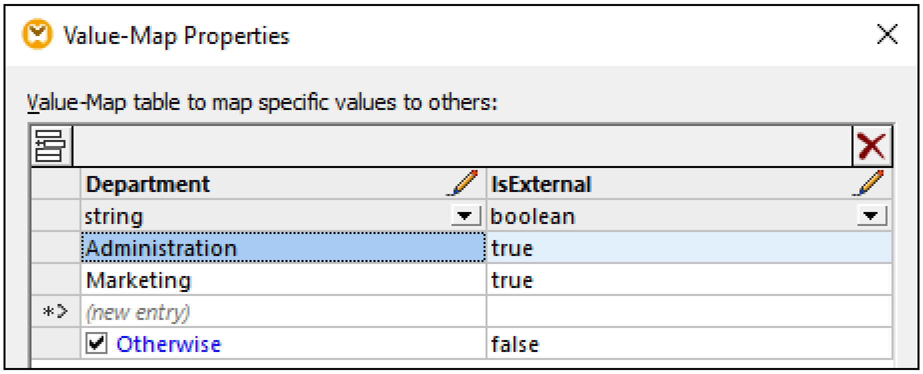
As shown above, the mapping consists of a source Excel component and a target JSON component. Notice that the two worksheets of the Excel file appear as on the source Excel component. The company name is mapped from the **Office** worksheet, while the rest of the fields are mapped from the **Staff** worksheet. To view how the mapping is configured to read row data from each worksheet, click the corresponding button. For the **Staff** worksheet, the settings are illustrated below.



The settings above instruct the mapping to read rows starting from 1 up to the end (dynamically). The first row is a header with column names, so the **First row is header...** check box is selected. The dialog box also specifies the type of columns A through G, and the name with which each column appears on the mapping. For more information about Excel components, see Microsoft OOXML Excel 2007+ 715.

The target component is a JSON structure. You can add JSON components to the mapping as described in Adding JSON Files as Mapping Components 795. The structure reflects the JSON schema, **employees.schema.json**, which is available in the same folder as the mapping. For more information about configuring the JSON component, see JSON Component Settings 796.

There also several intermediate components on the mapping. The first one is a value-map component. The value-map decides, based on the department, whether the phone number can be shared externally. This corresponds to business requirement #2 above. You can view the settings of the value-map component by double-clicking its title bar:



The configuration above can be summarized as follows: If **Department** is "Administration" or "Marketing", return `true`. Otherwise, return `false`. The `true` or `false` value computed this way is then written to the target JSON property. For more information about value maps, see Using Value-Maps[884].

Finally, the mapping makes use of the exists[1032] function whose purpose is to determine, in the context of each row, if a **Shares** value exists in the source Excel file. This function returns a Boolean `true` if the value exists, or `false` otherwise. The resulting `true` or `false` value is then supplied as input to an **If-Else** condition. On `false`, an empty string provided by a constant is passed on to the **Shares as null** target item. This empty value will become a `null` in the resulting JSON file, because this is the data type of the **Shares as null** item, as indicated by the icon.

If the employee's share value is numeric, it will no longer meet the **If-Else** condition. Instead, it will be written to the **Shares as number** target item (which is of numeric type #), through a separate connection.

For more information adding functions to a mapping, see Add a Function to the Mapping[905]. **If-Else** conditions are covered in more detail under Filters and Conditions[878].

Running the mapping

To preview the generated file in MapForce, click the **Output** tab. The image below illustrates only a fragment of the resulting JSON file.

```
{
    "Company": "Nanonull, Inc.",
    "Employees": [
        {
            "First": "Jessica",
            "Last": "Bander",
            "Phone": {
                "Extension": 241,
                "May be provided to external contacts": false
            },
            "Shares": 0
        }, {
            "First": "Michelle",
            "Last": "Butler",
            "Phone": {
                "Extension": 654,
                "May be provided to external contacts": false
            },
            "Shares": 1500
```


Mapping output

To save the mapping output to a file, do one of the following:

- On the **Output** menu, click **Save Output File**.
- Click the **Save generated output** toolbar button.

Automation with MapForce Server

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine. Note that running mappings on a server typically requires a few extra steps as outlined in Preparing Mappings for Server Execution [1372]. Here are the steps required to run this particular mapping on a server:

1. Right-click the mapping and select **Mapping Settings** from the context menu. Make sure to clear the **Make paths absolute in generated code** check box and save the mapping.
2. On the **File** menu, click **Compile to MapForce Server Execution File**. This generates an executable .mfx file in the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
3. Copy the generated .mfx file to a directory on the server machine (let's call it the "working directory").
4. Copy the Excel file from **<Documents>\Altova\MapForce2023\MapForceExamples\\Nanonull Inc.xlsx** to the working directory on the server machine.
5. Open a terminal and change the current directory to the working directory.
6. Run MapForce Server with the command below. Adjust **mapforceserver_exec** to be the path to the MapForce Server executable as applicable for your operating system.

```
mapforceserver_exec run Excel_Company_To_JSON.mfx
```

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see Compiling Mappings to MapForce Server Execution Files [1378].

4.10 Protocol Buffers

MapForce supports mapping data to or from binary files encoded in Protocol Buffers format (https://developers.google.com/protocol-buffers/). Specifically, MapForce (or MapForce Server) can read data from such binary files and convert it to any other format supported by MapForce (for example, XML, CSV, database, and so on). Likewise, you can read data from any format supported by MapForce and write it as a Protocol Buffers binary file. In addition, you can use Protocol Buffers files in Web service requests and responses, see Adding a Web Service Call (REST-Style) 587.

Note the following:

- To read or write data to binary files in Protocol Buffers format, the transformation language of the mapping must be set to "Built-In". Code generation in C#, C++, Java, XSLT, or XQuery is not supported. For more information, see Transformation Languages 27.
- To make data exchange possible, binary files in Protocol Buffers format are accompanied by so-called `.proto` files. MapForce supports Protocol Buffers `.proto` files version 2 and 3.

A `.proto` file defines the structure of the encoded binary data, similar to how an XML schema describes the structure of an XML instance file. For example, the following code listing illustrates a `.proto` file that describes a person object:

```
syntax="proto3";

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;
}
```

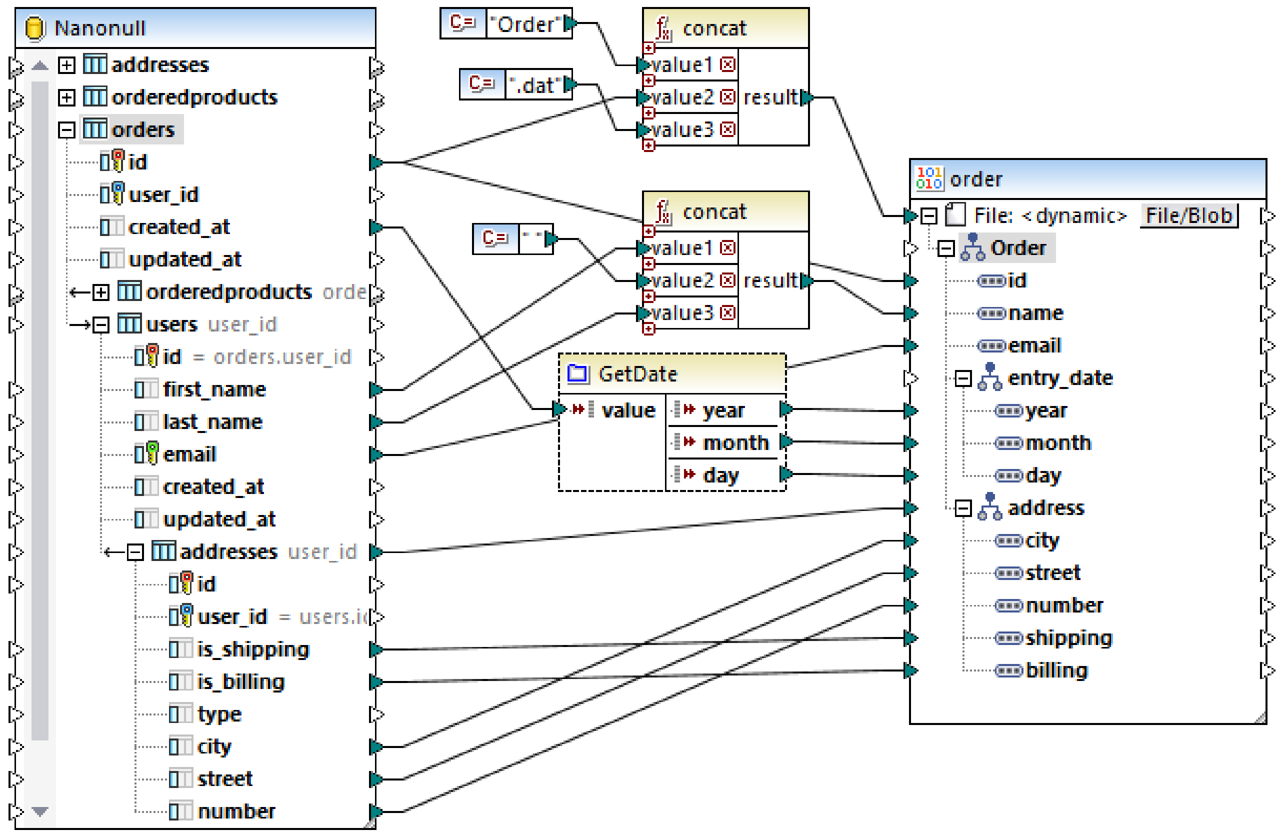
Example .proto file

The `.proto` file makes it possible to interpret and process the corresponding binary file. Therefore, in order for MapForce to read data in Protocol Buffers format, you need the following:

1. The source binary file. This file may have an extension like .bin or .dat, or any other extension.
2. The `.proto` file.

To write data in Protocol Buffers format with MapForce, all you need is the `.proto` file. You can either write your .proto files manually, or get them from whoever expects you to create data in the respective format. In either case, when you run the mapping, MapForce (or MapForce Server) generates the binary file based on the `.proto` file.

In order to create or read data from Protocol Buffers files outside MapForce, you would need to generate and write program code. With MapForce, however, you do not need to write code or generate it from .proto files—just add the .proto file to the mapping, and draw the required connections visually. Once you run the mapping, MapForce (or MapForce Server, if applicable) will read data from the source binary file(s) or generate binary files, according to the mapping design. For example, the image below illustrates a mapping that reads data from a database and writes it as multiple binary files in Protocol Buffers format.



GenerateOrders.mfd

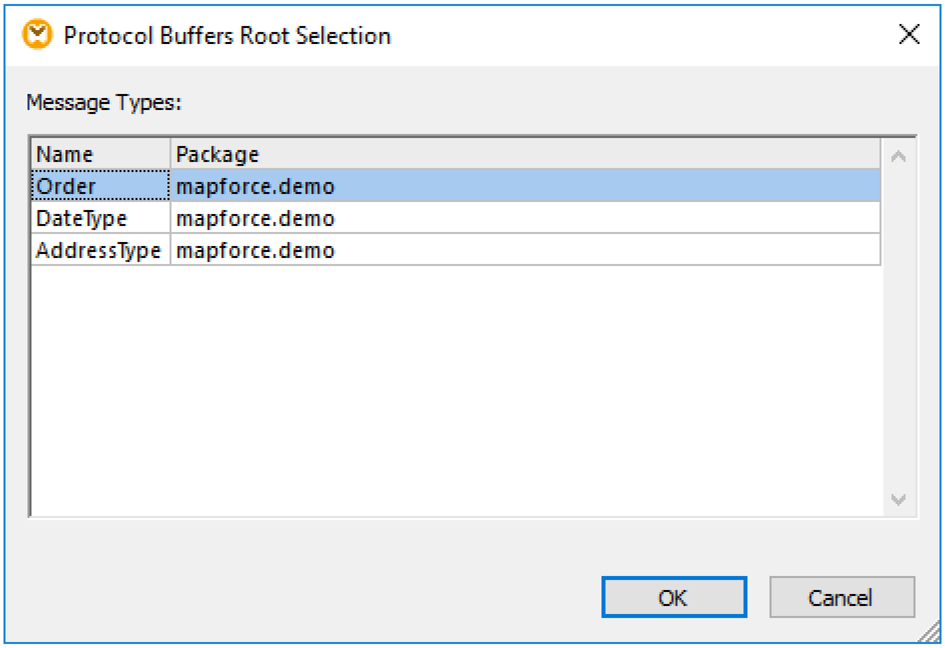
The mapping above is discussed in more detail in Example: Write Data to Protocol Buffers [821].

4.10.1 Adding Binary Files to the Mapping

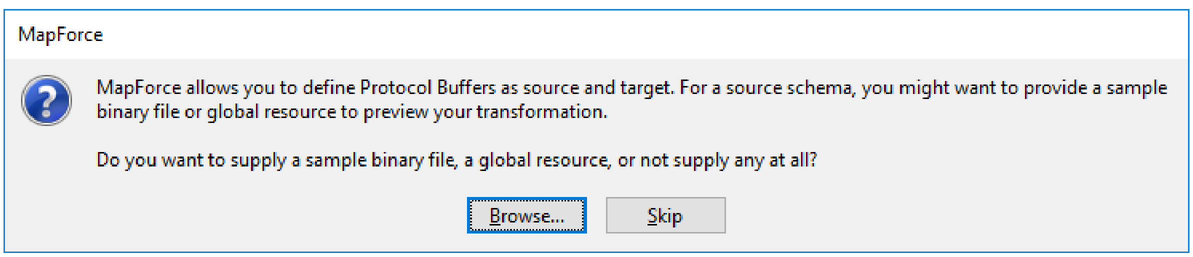
To map data to or from binary files encoded in Protocol Buffers format, you must first add their accompanying .proto file to the MapForce mapping, as shown below.

To add a .proto file as mapping component:

1. On the **Insert** menu, click **Protocol Buffers File**. Alternatively, click the **Insert Protocol Buffers File**  toolbar button.
2. Do one of the following:
   a. To open a .proto file stored on the local disk, browse for the file, and click **Open**.
   b. To open a .proto file from a URL, click **Switch to URL** and enter the URL of the file in the dialog box, see Adding Components from a URL [80].
   c. To open a .proto file previously defined as a global resource, click **Switch to Global Resource**. For information about global resources, see Altova Global Resources [1397].
3. A .proto file typically contains several message types. When prompted, select the message type that is the top object in the hierarchy (the so-called "root").

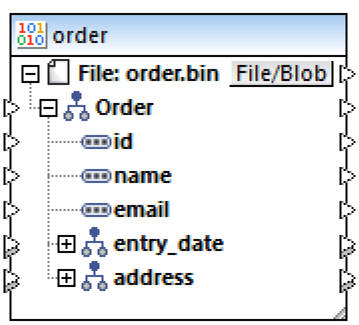


4. Click **OK**. A dialog box appears.



5. If you intend to read data from a Protocol Buffers binary file, click **Browse** and select the binary instance from which you want to read data. If you intend to write data to a Protocol Buffers file, or select an instance at a later time, click **Skip**.

If the .proto file can be read successfully, MapForce displays its structure on the mapping similar to how it does for other component types, for example:



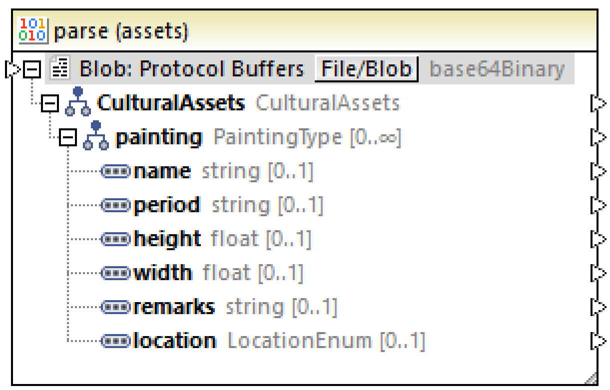
So far, you have added a binary component to the mapping; however, the mapping is not yet complete, because it does not read or write any data. For examples that illustrate complete mappings, see:

- Example: Read Data from Protocol Buffers[816]
- Example: Write Data to Protocol Buffers[821]

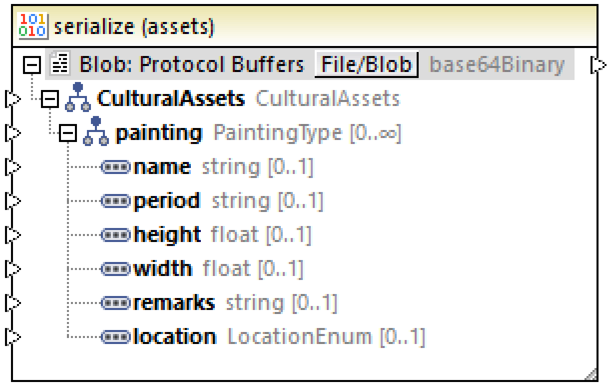
"File/Blob" button

The **File/Blob** button on the binary component opens a context menu with several commands that are useful in the following situations:

- **Use File Names from Component Settings -** This is the option selected by default and it is meaningful when your mapping reads or generates a single binary file. When this option is set, you can change the name of the file from the "Component Settings" dialog box, see Binary Component Settings[815].
- **Use Dynamic File Names Supplied by the Mapping -** This command is meaningful when your mapping should read or generate multiple binary files. When this option is set, you can no longer change the file name from the "Component Settings" dialog box. Instead, you set the file names based on some information that is coming from the mapping itself. For an example of use, see Example: Write Data to Protocol Buffers[821].
- **Parse Blobs to Protocol Buffers -** This command turns the component into a parsing component. This helps you parse a binary value in Protocol Buffers format so that it becomes a mappable tree structure. This works in a similar way to string parsing and serialization (see Parsing and Serializing Strings[1324]), only in this case it works not with character strings but with binary byte strings. For example, in the component below, you can connect an item of binary type to the "Blob" input on the left side. At mapping runtime, this binary value will be parsed and it will pass data onto the outputs of the mappable structure on the right side.



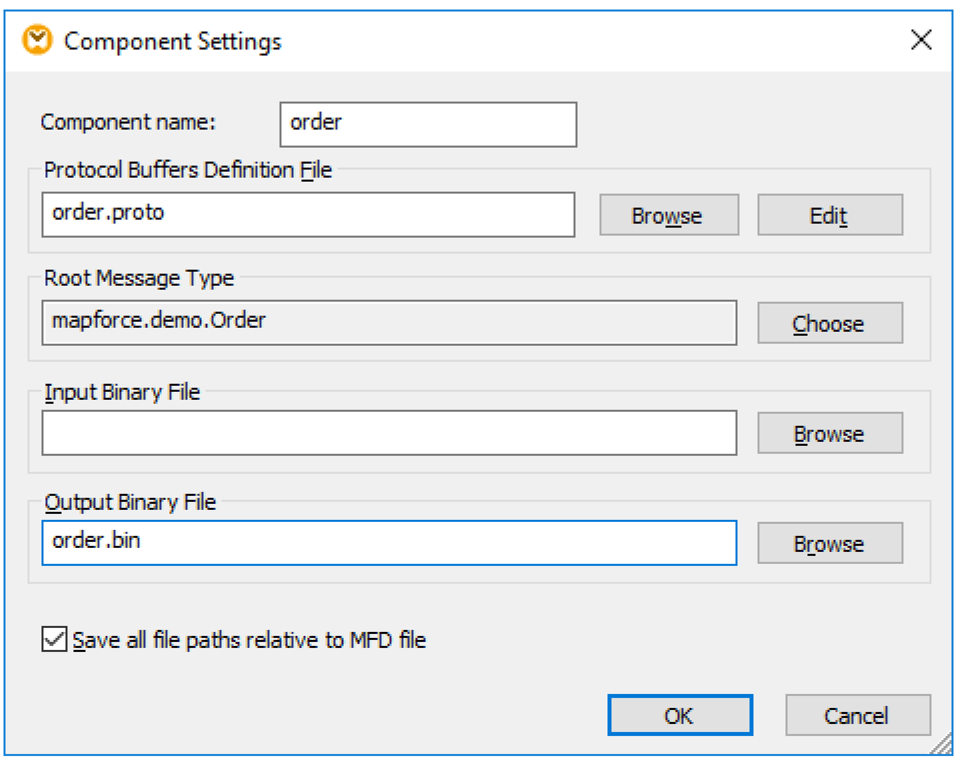
- **Serialize Protocol Buffers to Blobs -** This command turns the component into a serialization component. This helps you serialize a mappable tree structure to a binary value in Protocol buffers format. For example, in the component below, you can connect items of appropriate types to the mappable structure on the left side. At mapping runtime, the structure will be serialized and it will populate the single output (of binary type) on the right side, so that you can connect it further to a target component. This may be useful when your mapping should write data as a BLOB (Binary Large Object) type to some target database. Another use is to write data to binary fields (of type "bytes") of a target Protocol Buffer component.



4.10.2 Protocol Buffers Component Settings

After you add a binary component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component on the mapping, and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.



Component Settings dialog box

The available settings are as follows.

Component name	The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stop characters. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change the name of the component, be aware of the following: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.
Protocol Buffers Definition File	Specifies the name or path of the structure definition (.proto) file used by MapForce to map the actual binary data. To change the structure definition file, click **Browse** and select the new file.
Root Message Type	A .proto file typically contains several message types. Click **Choose** to select the message type that should be displayed as the top item in the hierarchy (the "root").
Input Binary File	Specifies the binary instance file from which MapForce will read data. This field is meaningful for a source component. To change the location of the file, click **Browse** and select the new file.
Output Binary File	Specifies the binary instance file to which MapForce will write data. This field is meaningful for a target component. To change the location of the file, click **Browse** and select the new file.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative Paths on a Component 85.

4.10.3 Example: Read Data from Protocol Buffers

This example shows you how to read data from a binary file encoded in Protocol Buffers format. The source binary file is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\assets.bin**. The `.proto` file describing the binary file is available in the same directory and it looks as follows:

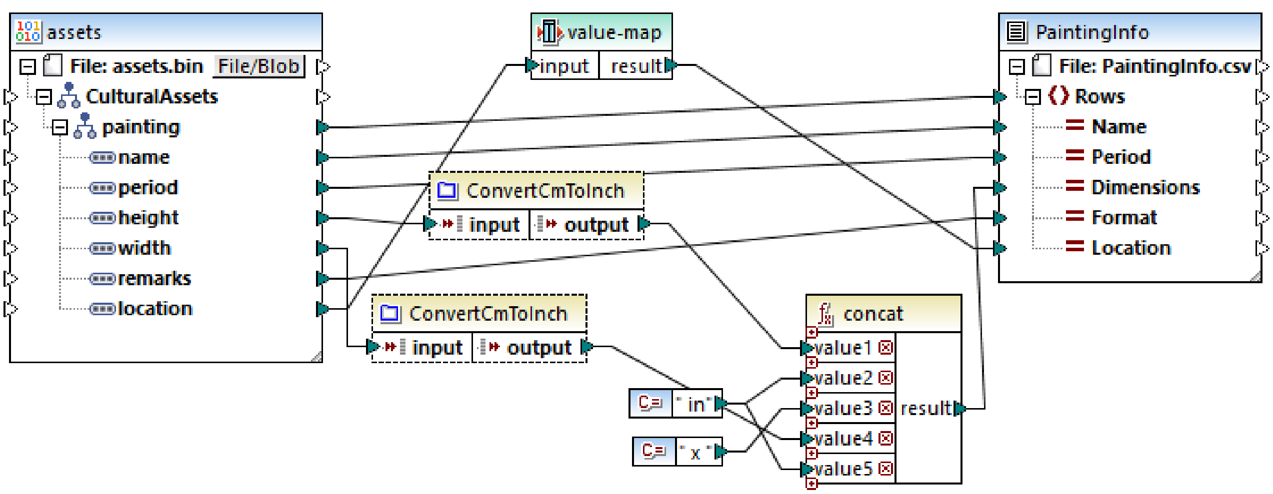
```
syntax="proto3";

package mapforce.demo;
```

```
message CulturalAssets {
  repeated PaintingType painting = 1;
}

message PaintingType {
       string name = 1;
       string period = 2;
       float height = 3; // in cm
       float width = 4; // in cm
       string remarks = 5;
       LocationEnum location = 6;

       enum LocationEnum {
              UNKNOWN = 0;
              MUSEUM = 1;
              TEMPLE = 2;
              PRIVATE = 3;
       }
}
```

assets.proto

The source binary file contains information about various cultural assets (in this example, a collection of paintings). As illustrated by the `.proto` file above, each painting has a **height** and **width** expressed in centimeters. There are various other fields that describe a painting, including an enumeration that specifies its location (museum, private collection, or temple). The business requirement is to extract this information to a comma-separated values (CSV) file. Also, all painting dimensions must be converted from centimeters to inch and be represented as a single string, in a format like "24 in x 56.8 in".

The mapping that extracts data according to the requirements above and produces the desired output is illustrated below. This mapping is available in the following directory: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**. You can open the mapping and run it directly, or follow the steps below to create it from scratch.



ReadProtocolBuffers.mfd

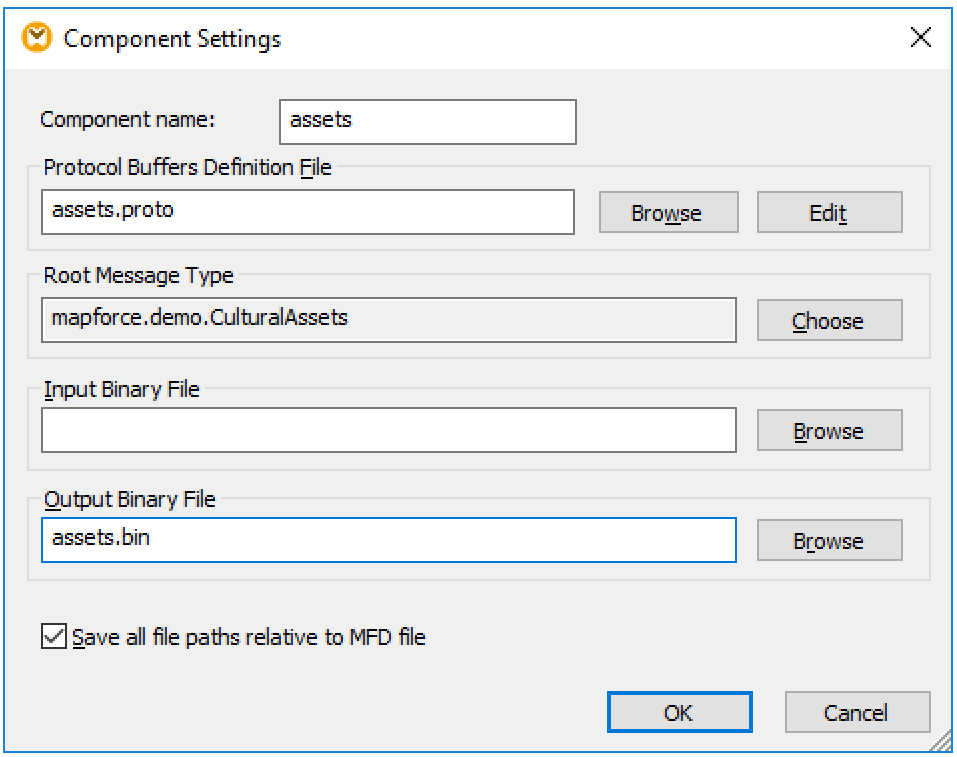
As illustrated above, the mapping consists of a source binary component (**assets**) and a target CSV component (**PaintingInfo**). Additionally, it contains the following intermediary components:

- A value-map table responsible for converting enumeration values like 0, 1, 2, 3 to a human-readable string representation (for example, `0 = UNKNOWN`, `1 = MUSEUM`, and so on).
- The **`ConvertCmToInch`** function. As suggested by the title, this is a user-defined function that converts centimeters to inches. This function is called twice on the mapping (for **height** and **width**, respectively).
- The **`concat`** function. This function concatenates several values and returns a string in the format "# in x # in", where # represents the actual height and width of the painting, in inches.

The following steps illustrate how to create the mapping above from scratch.

Step 1: Add the source binary file

1. On the **Insert** menu, click **Protocol Buffers File**, and browse for the following file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\assets.proto**.
2. When prompted, select "CulturalAssets" as root message, and click **OK**.
3. Double-click the title bar of the new component. The "Binary Component Settings" dialog box opens.
4. Click **Browse** next to **Input Binary File**, and select the following file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\assets.bin**.



See also Adding Binary Files to the Mapping 812.

Step 2: Add the target CSV component

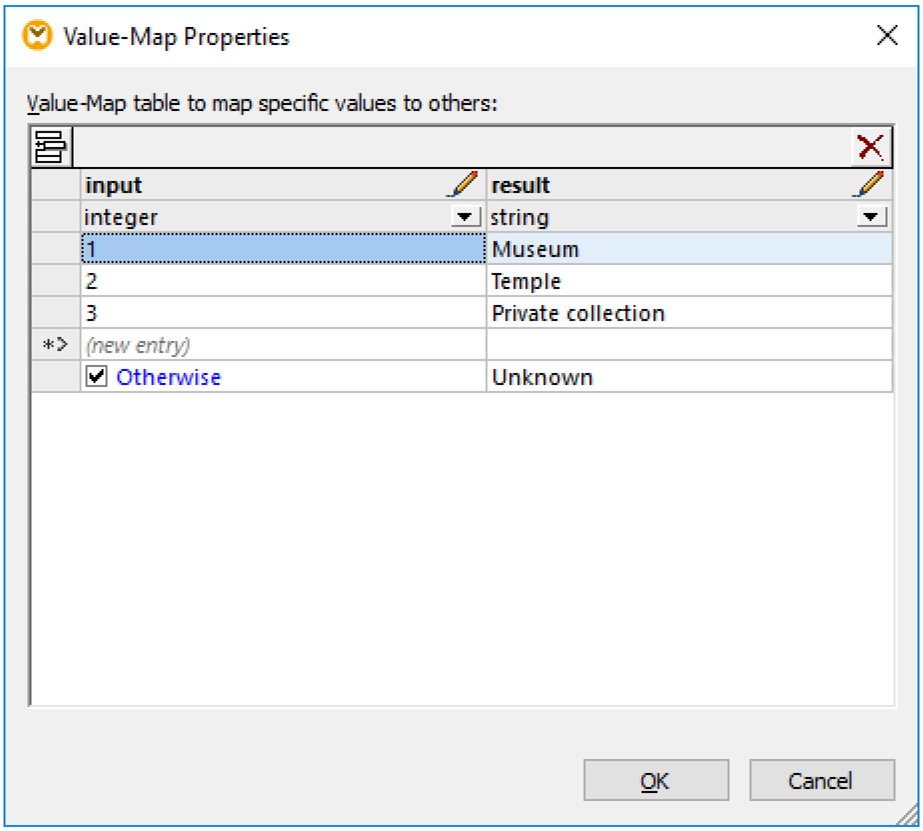
1. On the **Insert** menu, click **Text File**.
2. Select the **Use simple processing for standard CSV...** check box, and click **Continue**. The "CSV Component Settings" dialog box opens.

3. Click the **Append Field** button several times to create five target fields required to store the expected data.
4. Optionally, double-click the header of each field to give it a more suggestive name.

For more information, see Setting the CSV Options[572].

Step 3: Add the value-map

1. On the **Insert** menu, click **Value-Map**.
2. Create an entry for each mapped record, as shown below.

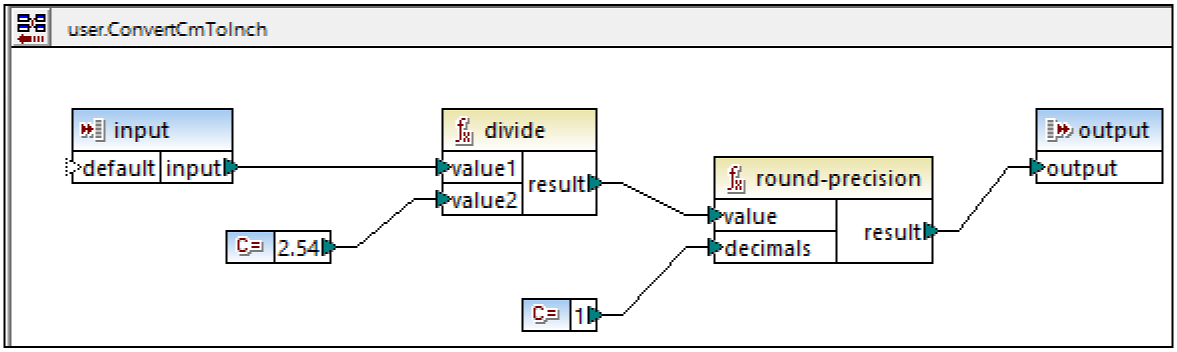


The "Otherwise" condition handles the case when the binary file contains no value for the `Location` field. In the event this happens, the mapping will write the value "Unknown" to the target instead of no result. For more information, see Using Value-Maps[884].

Step 4: Create the user-defined function

1. On the **Function** menu, click **Create User-Defined Function** and call it "ConvertCmToInch". Leave all other options unchanged.
2. Drag the **`divide`**[1017] and **`round-precision`**[1020] built-in functions from the Libraries window into the function's mapping window. See also Add a Function to the Mapping[905].
3. Add two constants and make all connections as illustrated below, see also Add a Constant to the Mapping[905]. The first constant supplies the decimal constant required to convert centimeters to inches according to formula, and the second one supplies the rounding precision.

Essentially, the function divides the input value by 2.54, and then rounds the result to one decimal digit, as illustrated below:

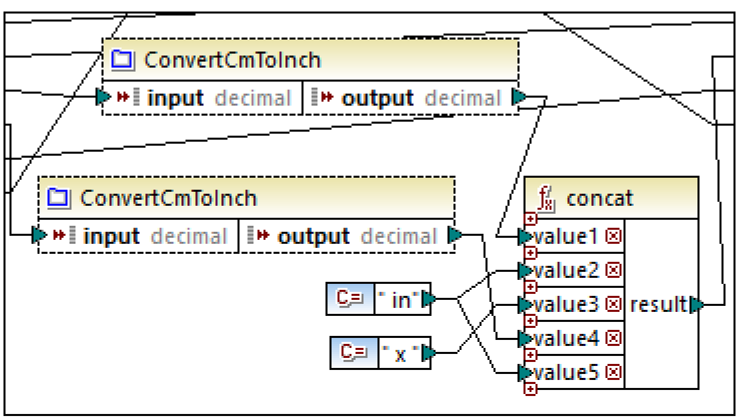


To avoid potential errors, set the data type of the input and output components to "decimal". To view or change the data type, double-click the title bar of the component, or right-click the title bar, and select **Properties** from the context menu.

Once the user-defined function is ready, click **Return to main mapping** , and drag the function from the Libraries window into the mapping to call it. In this example, it is called two times. For more information, see User-Defined Functions 927.

Step 5: Add the concatenation function

Start typing "concat" in the Libraries window, and then drag the **concat** 1057 function into the mapping. To add all the required input arguments, click **Add parameter** ( ) several times, see also Add or Delete Function Arguments 906. The constants " in" and " x " are string values. Essentially, the function returns the painting's **height**, followed by the string " in ", followed by " x ", followed by the painting's **width**, and followed by " in" again.



Running the mapping

To preview the generated file in MapForce, click the **Output** tab. Notice that the third CSV column contains painting dimensions in required format.

```
Red and White Plum Blossoms,18th century,61.4 in x 67.8 in,Ink and color on paper,Museum
Red and white hibiscuses,1197,10 in x 9.9 in,Color on silk,Museum
Pigeon on a peach branch,1108,11.2 in x 10.3 in,Color on silk,Private collection
Pine trees in snow,1773,142.5 in x 61.2 in,Light color on paper,Museum
Fishing village in the evening glow,13th century,13 in x 44.3 in,Ink on paper,Museum
Five Guardian Kings,Kamakura period,76.3 in x 49.7 in,Color on silk,Temple
Snowclad houses in the night,Edo period,11 in x 51.2 in,Ink and light color on paper,Private
```

Mapping output

To save the mapping output to the disk:

- On the **Output** menu, click **Save Output File**.

Automation with MapForce Server

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine, as follows:

1. Compile the mapping to a MapForce Server execution file (.mfx) with the menu command **File | Compile to MapForce Server Execution File**, see also Compiling Mappings to MapForce Server Execution Files[1378].
2. Copy the .mfx file to the server machine.
3. Run MapForce Server with the command below.

```
mapforceserver run ReadProtocolBuffers.mfx
```

Notes:

- **mapforceserver** is the path to the MapForce Server executable as applicable for your operating system.
- Change the path to the .mfx file as applicable, or copy the .mfx to the same folder as the executable.
- The .mfx file is self-contained; the .proto file is not needed for server execution.

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see Automation with MapForce Server[1371].

4.10.4 Example: Write Data to Protocol Buffers

This example shows you how to write data to binary files encoded in Protocol Buffers format. In this demo, the source data comes from a SQLite database (**Nanonull.sqlite**); however, you can use a similar approach to extract data from various other formats supported by MapForce, such as XML, EDI, JSON, and others.

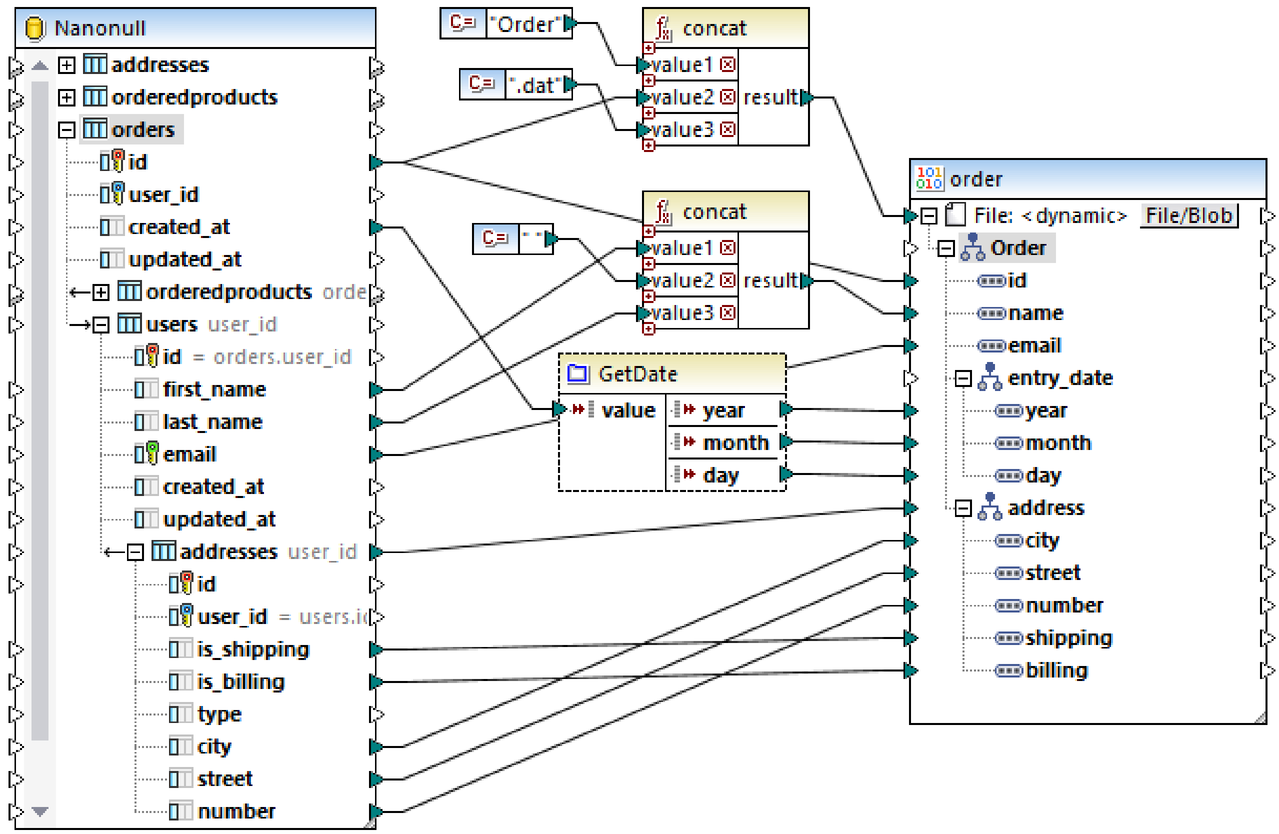
The source demo database stores data about persons, their addresses, and products they ordered from a store. The business requirement is to extract order information from the database and to generate multiple binary files in Protocol Buffers format. One binary file with `.dat` extension must be generated for each order. The generated file name must contain the order's unique ID, for example, **Order1.dat**, **Order2.dat**, and so on.

Also, each binary file must include the order ID, the person's name and email address, date, and shipping address information. More specifically, the `.proto` file that describes the structure of an order looks as follows:

```
syntax="proto3";

package mapforce.demo;

message Order {
  int32 id = 1;
  string name = 2;
  string email = 3;
  DateType entry_date = 4;
  // An order can have multiple addresses (shipping, billing)
  repeated AddressType address = 5;
}

message DateType {
  // Must be from 1 to 9999
  int32 year = 1;
  // Must be from 1 to 12
  int32 month = 2;
  // Must be from 1 to 31, and valid for the year and month
  int32 day = 3;
}

message AddressType {
  string city = 1;
  string street = 2;
  int32 number = 3;
  bool shipping = 5;
  bool billing = 4;
}
```

orders.proto

Both the **Nanonull.sqlite** database and the `orders.proto` file are available in the following directory: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**. The mapping design file that performs the transformation is called **GenerateOrders.mfd** and is available in the same directory. You can open the mapping and run it directly, or follow the steps below to create it from scratch.



GenerateOrders.mfd

As illustrated above, the mapping consists of a source component (the SQLite database) and a target component (the binary component). In the source database, the main table relevant for this mapping is **orders**.

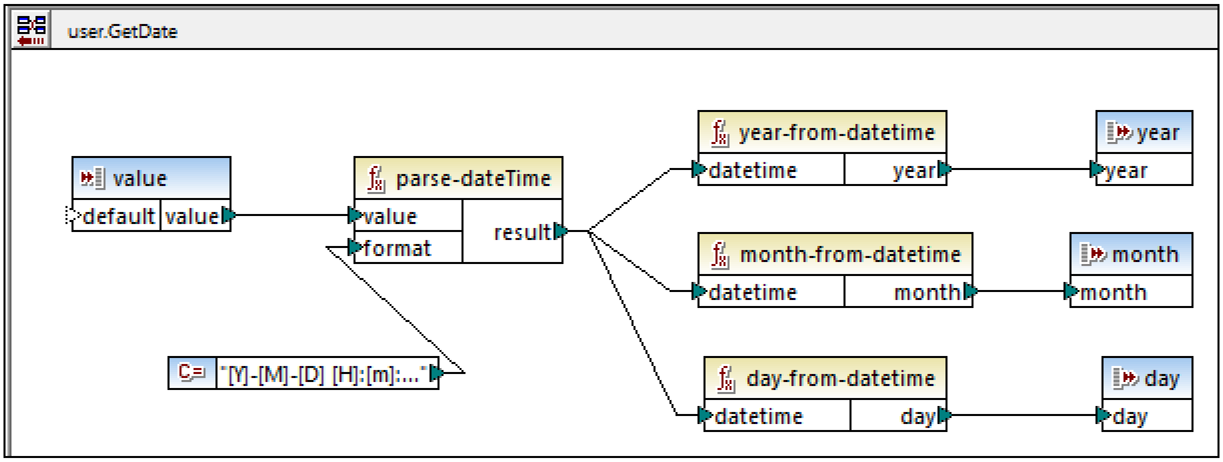
Notice that table **users** is nested under table **orders**, because MapForce detected automatically a foreign key relationship between these two tables. Likewise, table **addresses** is nested under **users**. All the connections drawn from such nested tables ensure that key relationships are preserved by the mapping. For more information, see Handling Database Relationships 257.

On the target side, the **File<dynamic>** item illustrates that the binary component is set to generate instance files dynamically, based on information received from the mapping. To have a target component generate files dynamically, click the File/Blob button and select **Use Dynamic File Names Supplied by the Mapping** from the context menu. For more information about dynamic file names, see Processing Multiple Input or Output Files Dynamically 1313.

In this mapping, for each **id** in the **orders** database table, a new file will be generated, as shown by the connection to the **File: <dynamic>** item. The name of each generated file is created with the help of the first **concat** 1057 function. This function joins the string "Order" with the unique ID of the order from the database, and with the extension ".dat". Therefore, the generated files will get names like **Order1.dat**, **Order2.dat**, and so on.

The **concat** function is also called a second time by the mapping. This time it returns the name of the person, by concatenating the first and last name, and inserting a space character in between.

Finally, the "GetDate" component in the middle is a user-defined function that populates the date fields in the binary file. This function takes as input a date expressed as string, parses it, and returns the year, month, and day separately as integer values. This conversion is required to accommodate the date format as it was defined in the `.proto` file. If you double-click the function's title bar on the mapping, you can view or change the definition of this function.



"GetDate" user-defined function

As illustrated above, the "GetDate" user-defined function parses the input value with the help of the [**`parse-dateTime`**](999) built-in function. The mask **`[Y]-[M]-[D] [H]:[m]:[s]`** matches the date format as it is stored in the database, for example, **`2017-10-19 08:09:54`**. For more information about format masks, refer to the description of the [**`parse-dateTime`**](999) function. Next, various built-in functions are called to extract and return the significant date parts from the parsed date.

Creating the mapping

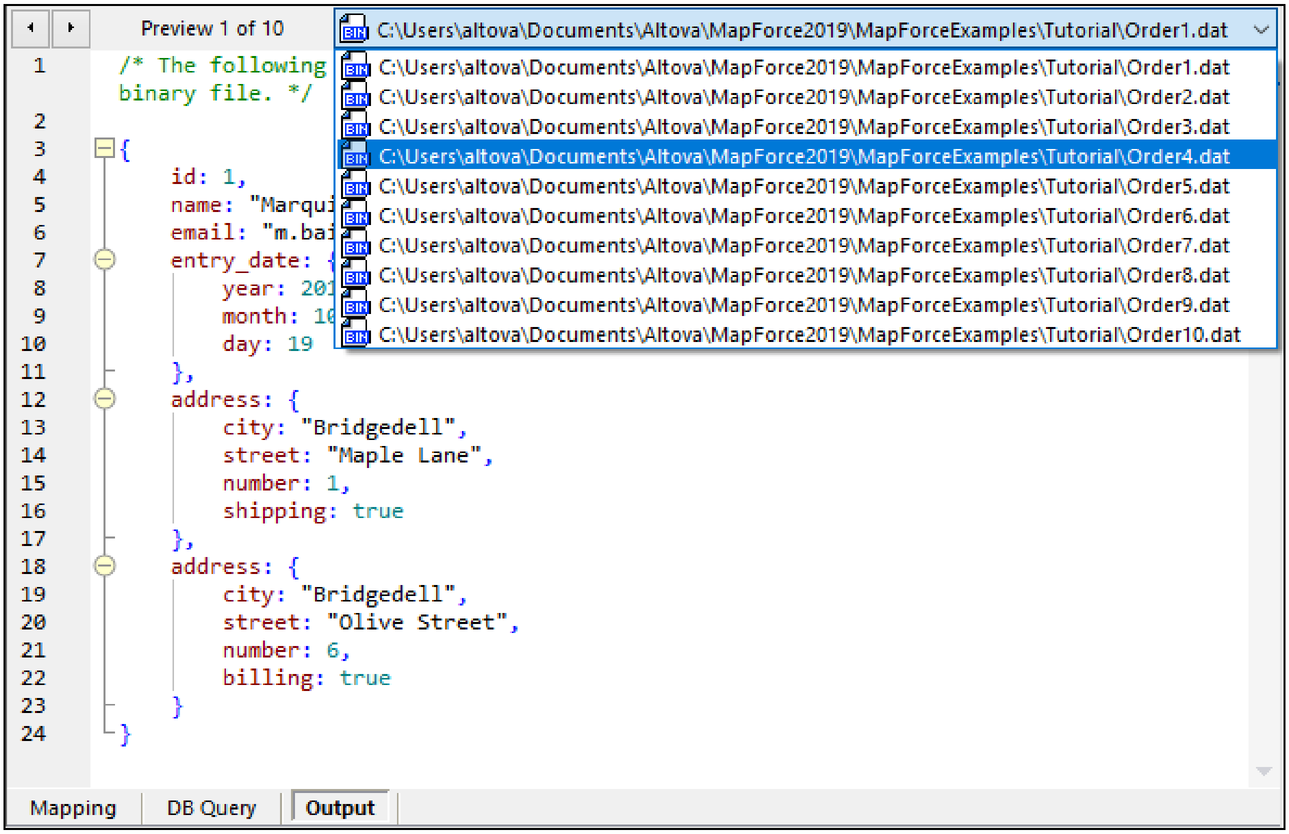
To create the mapping above (or a similar mapping) from scratch, take the following steps:

1. Insert the source database, as described in Adding Databases to the Mapping[249].
2. Insert the target `.proto` file, see Adding Binary Files to the Mapping[812].
3. Configure the target component to generate file names dynamically, when the mapping runs. To do this, click the File/Blob button and select **Use Dynamic File Names Supplied by the Mapping** from the context menu. Next, connect the **File<dynamic>** item to some input that provides the file name. In this example, the file name is produced with the help of the **`concat`** function, as mentioned above. For more information about dynamic file names, see Processing Multiple Input or Output Files Dynamically[1313].
4. Optionally, add any MapForce built-in functions and connect their inputs. In this example, the **`concat`** function is used in two occurrences. For more information, see Add a Function to the Mapping[905]. Note that, when you call built-in functions such as **`concat`**, it is likely that some of the input data will come from string constants. For information about how to add such constants, see Add a Constant to the Mapping[905].
5. Optionally, add any user-defined functions if required. These help you abstract away some of the mapping details and focus on the big picture. In this example, the "GetDate" user-defined function converts a string into date parts, as mentioned, and its implementation logic is hidden from the main mapping. For more information about creating custom functions, see User-Defined Functions[927].

If you are completely new to MapForce, see the Tutorials[42] chapter for simple step-by-step examples.

Running the mapping

As outlined in the business requirement at the beginning of this example, the mapping is expected to produce multiple .dat files. To preview the generated files, click the **Output** tab.



As illustrated above, the **Output** pane displays all the generated files, and you can navigate through them by clicking the directional buttons, or by picking an item from the list. To save the generated output, do one of the following:

- On the **Output** menu, click **Save All Output Files** ( ).
- Click the **Save all generated outputs** ( ) toolbar button.

> When you preview output binary files generated with MapForce, their structure is displayed as JSON since this format is human-readable as opposed to binary format. Viewing the output as JSON is to help you test the mapping. When you save the output file, it will be saved as a binary file, however.

Automation with MapForce Server

If you have licensed MapForce Server, you can also run the mapping at the command line, on a Linux, macOS, or Windows machine, as follows:

1. Compile the mapping to a MapForce Server execution file (.mfx) with the menu command **File | Compile to MapForce Server Execution File**, see also Compiling Mappings to MapForce Server Execution Files 1378.
2. Copy the .mfx file to the server machine.
3. Since this mapping reads data from a SQLite database file, copy the **Nanonull.sqlite** database file to the same directory as the .mfx file on the server. For other database kinds, see Database mappings in various execution environments 162.
4. Run MapForce Server with the command below.

```
mapforceserver run GenerateOrders.mfx
```

Notes:

- **mapforceserver** is the path to the MapForce Server executable as applicable for your operating system.
- Change the path to the .mfx file as applicable, or copy the .mfx to the same folder as the executable.

In server execution, you can also run mappings as an API call, or as FlowForce Server jobs, either on demand or on recurring basis. For more information, see Automation with MapForce Server 1371.

4.11 Binary Files

MapForce provides the ability to read BLOB (binary large object) data from binary files into a mapping, and then consume it without changing the internal structure of the binary data (raw). For example, you can save binary data to a database BLOB field, to a field of type `xs:base64Binary` in an XML file, or send it to a Web service*.

** Web service calls are supported in MapForce Enterprise Edition only.*

You can also create mappings that read binary data from some source (such as a BLOB field in a database, a field of type `xs:base64Binary` in an XML file, or a Web service) and then write binary files to the disk.

The following are just some example scenarios that benefit from the ability to read or write binary files:

- Extract binary content encoded as base-64 data from an XML file and save it to the disk (for example, as a PDF file)
- Process image files stored on the disk and send them as base-64 encoded binary content to a Web service
- Extract BLOB content from a database table and save it as image files to the disk (one image file for each row in the database table)
- Read image files from the disk and save them to a database table as BLOB data fields.

Note: Mapping data to or from binary files requires BUILT-IN 27 as a transformation language. You can preview the mapping in MapForce (and save the output files, if any) or choose to execute it with MapForce Server (licensed separately) on a different computer or platform. It is not supported to generate an executable C#, C++, or Java program from mappings that read or write binary files.

Adding binary files to the mapping

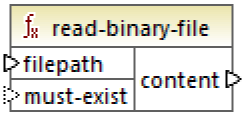
As such, there is no component kind associated with binary files in MapForce, like it is the case, for example, with XML, text, or JSON files. Instead, to help you accomplish goals such as the ones above, the following MapForce built-in functions are available:

- read-binary-file 1101
- write-binary-file 1102

You can find these functions in the Libraries window (more specifically, in the "lang" library). To use either of the functions in a mapping, drag them from the Libraries window into the mapping, or double-click inside an empty area of the mapping and start typing the function name to select it. For further information, see Adding Functions to the Mapping 905.

read-binary-file

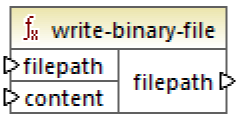
This function returns the content of the specified file as a BLOB (binary large object) of type `xs:base64Binary`. Note that even though the data type is called "base64Binary", the internal representation is just a BLOB. Only when you map the function's result to an XML node of type `xs:base64Binary` will it actually be base64-encoded. You could also map the function's result to `xs:hexBinary`, to a database blob, or to a binary field in a Protocol Buffers structure.



To read a binary file into a mapping, supply its path as input to the **filepath** argument. If the **filepath** is relative, then MapForce will look for the file in the same directory as the mapping. The **must-exist** argument is optional; if the file cannot be opened and this parameter is **true**, the mapping throws an error. If the file cannot be opened and this parameter is **false**, an empty binary is returned.

write-binary-file

This function writes binary content to the specified file path and returns the path of the written file. If a binary file is the only desired output, connect the function's result to a simple output[844] component. Because this function writes a file whenever its output is used in the mapping, it is recommended to connect the function's result directly to a target component, without using other processing in between.



To write binary files, supply their path as input to the **filepath** argument. If **filepath** is relative, then MapForce will generate the file in the same directory as the mapping. The **content** argument must be connected to the actual binary content (for example, a BLOB field in a database).

When you preview the mapping in MapForce, the function generates temporary files by default, instead of writing files directly to the disk. To save the temporary files to disk, first click the **Output** tab, and then click the **Save generated output** or **Save all generated outputs** toolbar button, as applicable.

To configure MapForce to write output directly to final files instead of temporary, select the **Tools | Options** menu command, click **General**, and then select the **Write directly to final output files** option. Be aware that this option overwrites any existing files with the same name.

The function always returns the final (not temporary) file name, even when the final file is not saved to the disk yet (that is the case when you preview the mapping and the **Write directly to final output files** option is disabled).

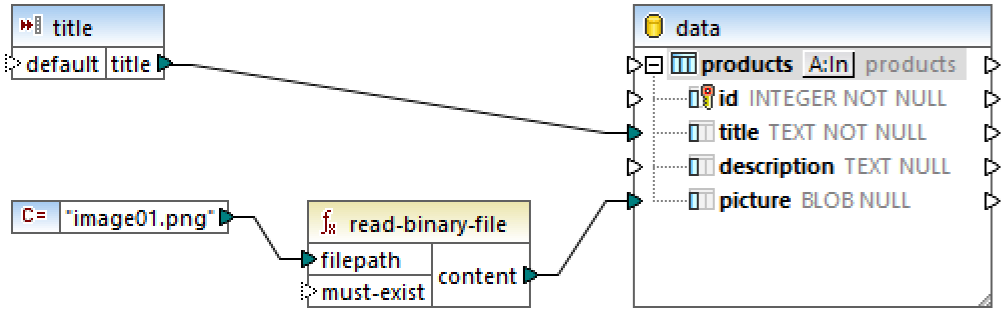
> Note that it is not supported for a mapping to read back its own output file.

Examples

See Example: Read Binary Files[828] and Example: Write Binary Files[830].

4.11.1 Example: Read Binary Files

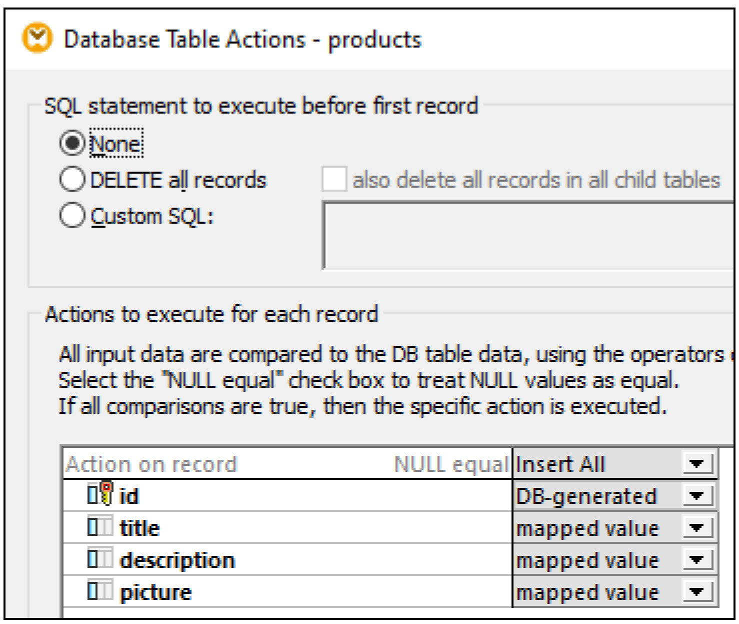
The mapping illustrated below reads data from an image file and writes it to a database table. The target database is SQLite. Notice that the data type of the **picture** database field is BLOB.



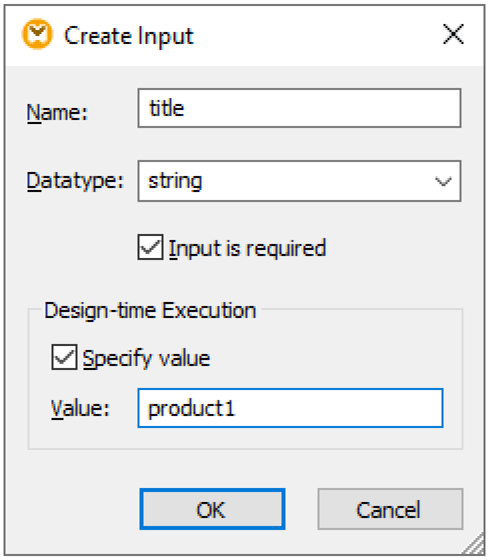
To extract binary content from the file, the `read-binary-file` function was used. In this example, the first argument, **filepath**, is supplied by a constant. Note that, because the path is relative, MapForce will look for the image file in the same directory as the mapping.

The mapping populates the following fields in the target database:

- **id** - In this example, the database component is configured so that **id** is database-generated rather than being supplied by the mapping. For more information, see Inserting Data into a Table[283].



- **title** - This value is provided by a simple input component with the same name. Note that a design-time execution value is set ("product1") in order to make it possible to preview the mapping. For more information, see Supplying Parameters to the Mapping[833].



- **picture** - This field receives the direct output of the `read-binary-file` function.

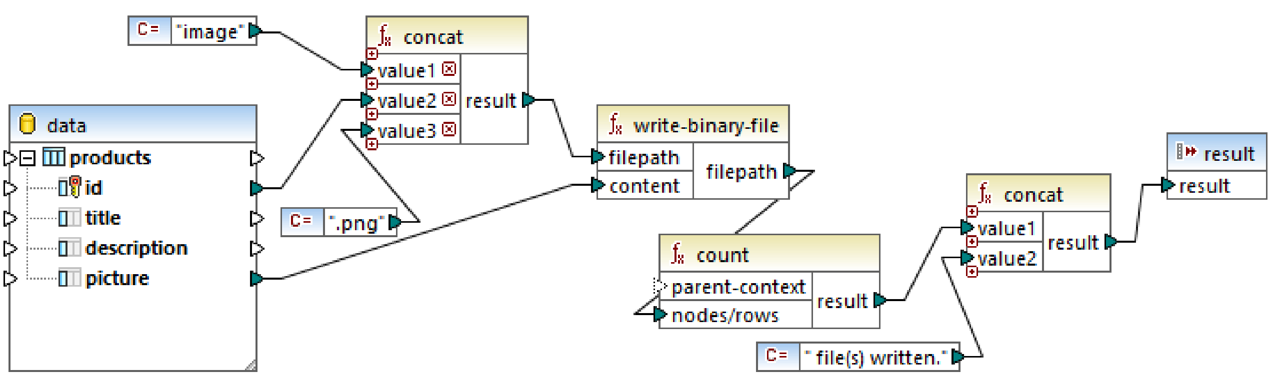
Because the target component is a database, previewing the mapping generates a pseudo-SQL script that you can review, but does not send any changes to the database. To run the actual script against the database, select the menu command **Output | Run SQL-Script**.

4.11.2 Example: Write Binary Files

The mapping illustrated below reads BLOB values from a SQLite database and writes image files to the disk. The database has a table called **products**, which has the following columns (fields):

- **id** (*integer*, the unique permanent serial number of the record)
- **title** (*text*, the title of the product)
- **description** (*text*, the product's description)
- **picture** (*blob*, the product's image)

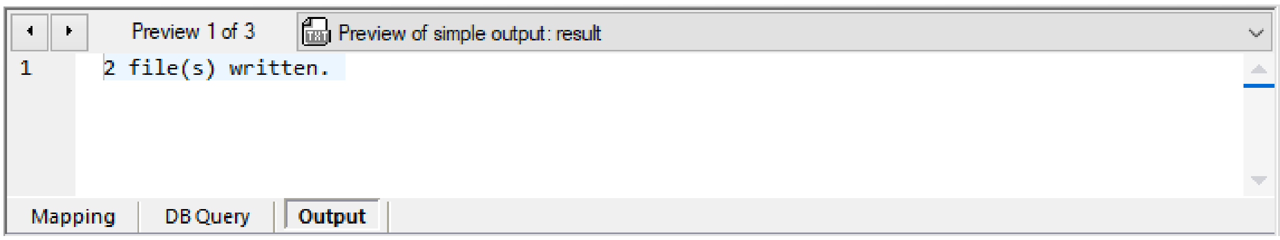
For the scope of this example, only the **id** and **picture** fields are relevant.



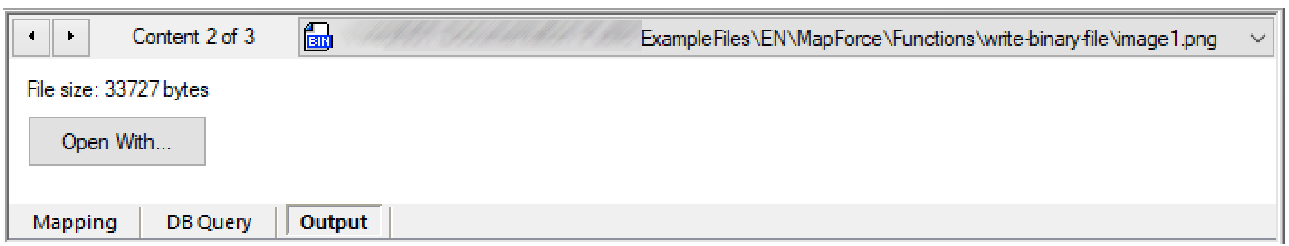
The goal of the mapping is to extract all pictures from the **products** table and write them as files to the disk. As illustrated above, the **`write-binary-file`** function is used for that purpose. The first argument, **filepath**, receives the file path for each image. The path must be unique to avoid overriding any files, so it is generated by concatenating the unique database **id** of each record with the word "image" and then adding the ".png" file extension (it is assumed that all pictures are in PNG format). Note that the path is relative, so the files will be written to the same directory as the mapping.

The second argument, **content**, receives the binary content stored in the database.

The `count` function returns a count of all generated files, and combines this number with the string "file(s) written". This provides a report as to how many files were actually generated by the mapping. In this example, the database contains only two product records, so the mapping output reflects this:



As stated previously, this function generates temporary files when the mapping runs in preview execution. To preview each individual file, use the arrow buttons to navigate to the record of interest, click the **Open with** button, and select an image editor.



To save all files, click the **Save generated output** or **Save all generated outputs** toolbar button, as applicable.

5 Transformation Components

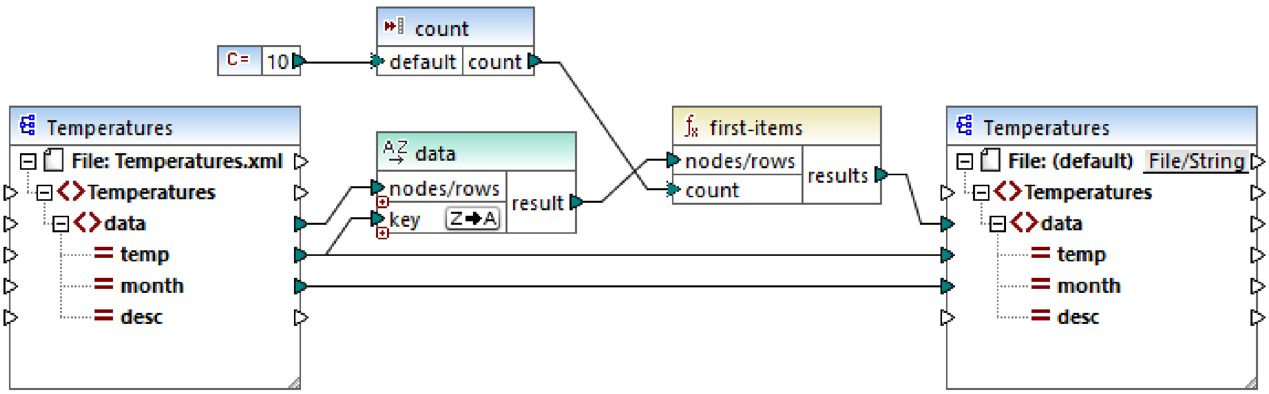
This section describes transformation components that can be used to transform data or to store data temporarily for further processing. The list of transformation components is given below:

- Simple input 833
- Simple output 843
- Variables 847
- Join Components 860
- Sort Components 872
- Filters and Conditions 878
- Value-Maps 884
- Group Functions 894
- Exceptions 900

Note that functions also belong to transformation components. However, functions 904 are organized as a standalone section.

5.1 Simple Input

If you need to create a mapping that takes parameters as input, you can do so by adding a special component type called "simple input component". Simple input components always have a simple data type (for example, string, integer, and so on) instead of a structure of items and sequences. For example, in the mapping illustrated below, there is a simple input component **count**. Its role is to supply as parameter the maximum number of rows that should be retrieved from the source XML file (with value **10** as default). Importantly, the nodes supplied as input to the first-items [1033] function are sorted with the help of a sort component, so the mapping outputs the highest *N* temperatures only, where *N* is the parameter's value.



FindHighestTemperatures.mfd

Another fairly common usage of simple input components is to supply a file name to the mapping. This is useful in mappings that read input files or write output files dynamically, see Processing Multiple Input or Output Files Dynamically [1313].

You can use simple input components in any the following MapForce transformation languages:

- BUILT-IN (when you preview the mapping transformation directly in MapForce, from the **Preview** tab)
- BUILT-IN (when you run a compiled MapForce Server execution file)
- XSLT 1.0, XSLT 2.0, XSLT 3.0
- XQuery
- C++
- C#
- Java

In case of mappings executed with MapForce Server or by means of generated code, simple input components become command line parameters. In case of mappings generated as XSLT transformations, simple input components correspond to stylesheet parameters in the generated XSLT file.

You can create each simple input component (or parameter) as optional or mandatory, see Adding Simple Input Components [834]. If necessary, you can also create default values for the mapping input parameters, see Creating a Default Input Value [836]. This enables you to safely run the mapping even if you do not explicitly supply a parameter value at mapping execution time. For an example, see Example: Using File Names as Mapping Parameters [837].

Input parameters added on the main mapping area should not be confused with input parameters in user-defined functions 927. There are some similarities and differences between the two, as follows.

Input parameters on the mapping	Input parameters of user-defined functions
Added from **Function | Insert Input** menu.	Added from **Function | Insert Input** menu.
Can have simple data types (string, integer, and so on).	Can have simple as well as complex data types.
Applicable to the entire mapping.	Applicable only in the context of the function in which they were defined.

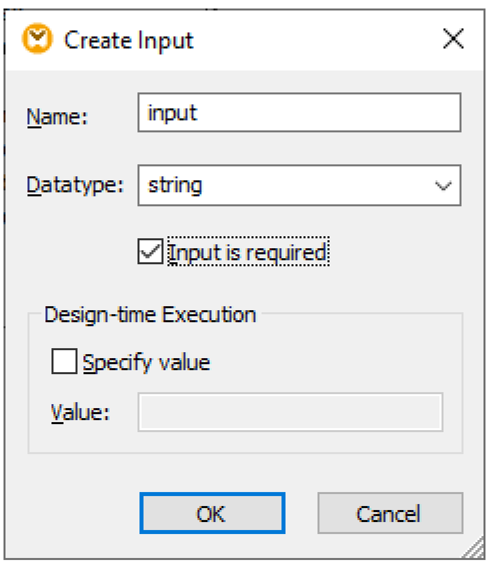
When you create a reversed mapping (using the menu command **Tools | Create Reversed Mapping**), a simple input component becomes a simple output component.

5.1.1 Adding Simple Input Components

To add a simple input to the mapping:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. Do one of the following:

   - On the **Function** menu, click **Insert Input**.
   - On the **Insert** menu, click **Insert Input**.
   - Click the **Insert Input** toolbar button.



3. Enter a name and select the data type required for this input. If the input should be treated as a mandatory mapping parameter, select the **Input is required** check box. For a complete list of settings, see Simple Input Component Settings 835.

Note: The parameter name can contain only letters, digits, and underscores; no other characters are allowed. This makes it possible for a mapping to work across all code generation languages.

4. Click **OK**.

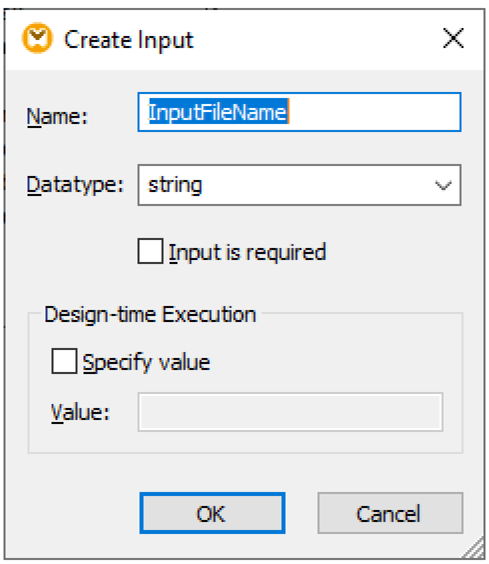
You can change later any of the settings defined here (see Simple Input Component Settings [835]).

5.1.2 Simple Input Component Settings

You can define the settings applicable to a simple input component when adding it to the mapping area. You can also change the settings at a later time, from the Edit Input dialog box.

To open the Edit Input dialog box, do one of the following:

- Select the component, and, on the **Component** menu, click **Properties**.
- Double-click the component.
- Right-click the component, and then click **Properties**.



Edit Input dialog box

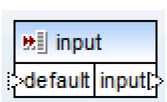
The available settings are as follows.

Name	Enter a descriptive name for the input parameter corresponding to this component. At mapping execution time, the value entered in this text box becomes the name of the parameter supplied to the mapping; therefore, no spaces or special characters are allowed.
Datatype	By default, any input parameter is treated as string data type. If the parameter should have a different data type, select the respective value from the list. When the mapping is executed, MapForce casts the input parameter to the data type selected here.

Input is required	When enabled, this setting makes the input parameter mandatory (that is, the mapping cannot be executed unless you supply a parameter value). Clear this check box if you want to specify a default value for the input parameter (see Creating a Default Input Value 836).
Specify value	This setting is applicable only if you execute the mapping during design time, by clicking the **Preview** tab. It allows you to enter directly in the component the value to use as mapping input.
Value	This setting is applicable only if you execute the mapping during design time, by clicking the **Preview** tab. To enter a value to be used by MapForce as mapping input, select the **Specify Value** check box, and then type the required value. **Note:** If you click the **Specify value** check box and enter a value in the adjacent box, the entered value takes precedence over the default value when you preview the mapping (that is, at design-time execution). However, the design-time value has no effect in the generated XSLT, XQuery, or program code, in execution by MapForce Server, or deployment to FlowForce Server.

5.1.3 Creating a Default Input Value

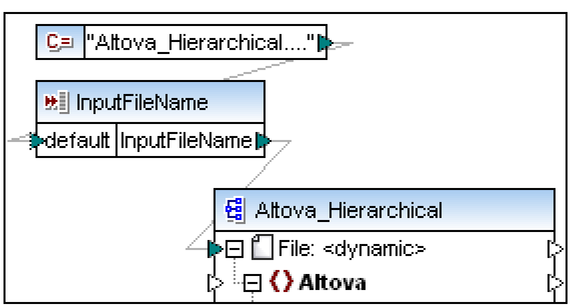
After you add an Input component to the mapping area, notice the **default** item to the left of the component.



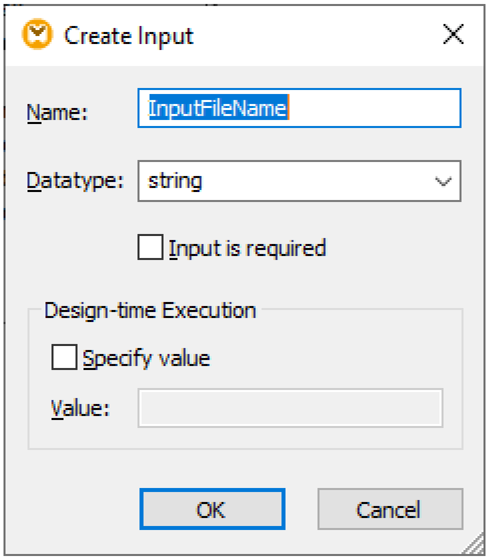
Simple input component

The **default** item enables you to connect an optional default value to this input component, as follows:

1. Add a constant component (on the **Insert** menu, click **Constant**), and then connect it to the **default** item of the input component.



2. Double-click the input component and clear the **Input is required** check box. When you create a default input value, this setting is not meaningful and causes mapping validation warnings.



3. Click **OK**.

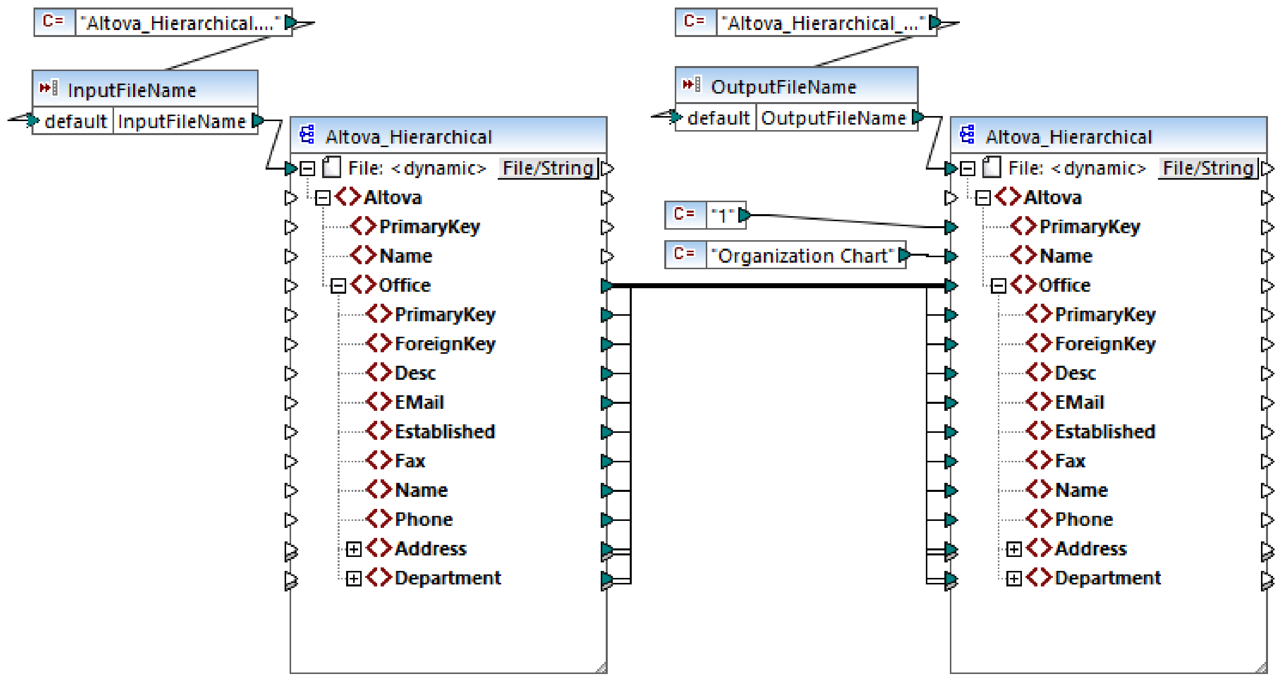
Note: If you click the **Specify value** check box and enter a value in the adjacent box, the entered value takes precedence over the default value when you preview the mapping (that is, at design-time execution). However, the design-time value has no effect in the generated XSLT, XQuery, or program code, in execution by MapForce Server, or deployment to FlowForce Server.

5.1.4 Example: Using File Names as Mapping Parameters

This example walks you through the steps required to execute a mapping that takes input parameters at runtime. The mapping design file used in this example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\FileNamesAsParameters.mfd**.

This mapping reads data from a source XML file and writes it to a target XML file. The data is written to the target file almost unchanged; only the attributes **PrimaryKey** and **Name** are populated with some constant values from the mapping. The main goal of the mapping is to enable the caller to specify the name of the input file and the name of the output file, as mapping parameters, at mapping runtime.

To achieve this, the mapping has two input components: **InputFileName** and **OutputFileName**. These supply the input file name (and the output file name, respectively) of the source and target XML file. For this reason, they are connected to the **File: <dynamic>** item. You can switch a component to this mode by clicking the **File/String** (File/String) button, and selecting **Use Dynamic File Names Supplied by Mapping**.



FileNamesAsParameters.mfd (MapForce Enterprise Edition)

If you double-click the title bar of either of the **InputFileName** and **OutputFileName** components, you can view or edit their properties. For example, you can specify the data type of the input parameter or change the input parameter name, as described in Simple Input Component Settings [835]. In this example, the input and output parameters are configured as follows:

- The **InputFileName** parameter is of type "string" and it has a default value supplied by a constant defined in the same mapping. The constant is of type "string" and its value is "Altova_Hierarchical.xml". Therefore, when this mapping runs, it will attempt to read data from a file called "Altova_Hierarchical.xml", assuming that you do not supply some other value as parameter.
- The **OutputFileName** parameter is of type "string" and it also has a default value supplied by a constant defined in the same mapping. The constant is of type "string" and its value is "Altova_Hierarchical_output.xml". Therefore, the mapping will create an XML output file called "Altova_Hierarchical_output.xml" when it runs, assuming that you do not supply some other value as parameter.

The following sections illustrate how to run the mapping and supply parameters in the following transformation languages:

- XSLT 2.0 [839], using RaptorXML Server
- Built-in (MapForce Server Execution File) [839], using MapForce Server
- Java [840]
- C# [841]
- C++ [841]

XSLT 2.0

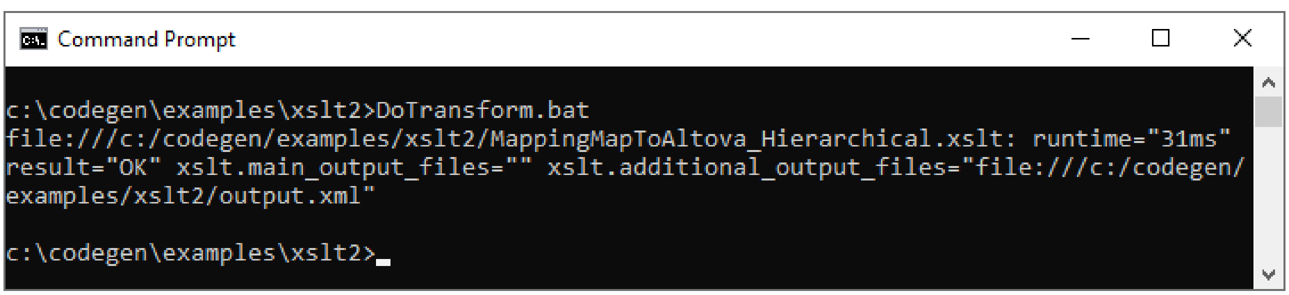
If you generate code in XSLT 1.0, XSLT 2.0, or XSLT 3.0, a **DoTransform.bat** batch file is generated in the chosen target directory, in addition to the XSLT file. The **DoTransform.bat** lets you execute the mapping with RaptorXML Server, see Automation with RaptorXML Server 1370.

To use a different input (or output) file, edit the **DoTransform.bat** file to include the required parameters, as follows:

1. First, generate the XSLT code. For example, to generate XSLT 2.0, select the menu command **File | Generate Code In | XSLT 2.0**.
2. Copy the **Altova_Hierarchical.xml** file from **<Documents>\Altova\MapForce2023\MapForceExamples** to the directory where you generated the XSLT 2.0 code (in this example, **c:\codegen\examples\xslt2**). As stated previously, the mapping will attempt to read this file if you do not supply a custom value to the **InputFileName** parameter.
3. Edit **DoTransform.bat** to include the custom input parameter either before or after `%*`. Note that the parameter value is enclosed with single quotes. The available input parameters are listed in the **rem** (Remark) section. Let's suppose that you would like to generate an output file called **output.xml**. To achieve this, change the **DoTransform.bat** file as follows:

```
@echo off

RaptorXML xslt --xslt-version=2
  --input="MappingMapToAltova_Hierarchical.xslt"
  --param=OutputFileName:'output.xml' %* "MappingMapToAltova_Hierarchical.xslt"
rem --param=InputFileName:
rem --param=OutputFileName:
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

When you run the **DoTransform.bat** file, RaptorXML Server completes the transformation using **Altova_Hierarchical.xml** as input. If you followed the steps above, the name of the generated output file will be **output.xml**.



MapForce Server Execution File

To supply custom input parameters to a MapForce Server execution file:

1. If you haven't done that already, open the **FileNamesAsParameters.mfd** example from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.

2. On the **File** menu, click **Compile to MapForce Server Execution File**, see also Compiling Mappings to MapForce Server Execution Files 1378. When prompted, save the .mfx execution file to a directory on your computer (in this example, **c:\codegen\examples\mfx**).
3. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory to the directory where you saved the .mfx file.
4. Run MapForce Server with the following command:

```
MapForceServer.exe run "C:\codegen\examples\mfx\FileNamesAsParameters.mfx"
  -p=InputFileName:"C:\codegen\examples\mfx\Altova_Hierarchical.xml"
  -p=OutputFileName:"C:\codegen\examples\mfx\OutputFile.xml"
```

In the MapForce Server command above, `-p=InputFileName` and `-p=OutputFileName` are the input parameters to the mapping. You can use any file name as the value of **-OutputFileName**. However, the file name supplied in **-InputFileName** parameter must exist as a physical file; otherwise, the mapping will fail.

Note: If you see the message "MapForceServer.exe is not recognized as an internal or external command, operable program, or batch file", change the current directory to the one where the MapForce Server executable is installed. To avoid changing the path every time when you run a mapping, add to your operating system's PATH environment variable the path of the directory where the MapForce Server executable is installed (for example, **C:\Program Files (x86)\Altova\MapForceServer2023\bin**).

With MapForce Server, running a mapping is also possible by calling the MapForce Server API (which is invokable from languages such as C++, C#, or Java). For further information about this scenario, refer to the MapForce Server documentation (https://www.altova.com/documentation).

Java

To supply a custom input parameter to a Java .jar application:

1. If you haven't done that already, open the **FileNamesAsParameters.mfd** example from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
2. On the **File** menu, click **Generate Code In | Java**.
3. Compile the Java code into an executable JAR file. For an example of how to do this in Eclipse, see Example: Generate and Run Java Code 1514.
4. Copy the **Altova_Hierarchical.xml** file from **<Documents>\Altova\MapForce2023\MapForceExamples** to the directory where the .jar file is. As stated previously, the mapping will attempt to read this file if you do not supply a custom value to the **InputFileName** parameter.
5. Run the Java application with the following command:

```
java -jar Mapping.jar /OutputFileName "output.xml"
```

In the command above, the input parameter `/OutputFileName` supplies the name of the output file to be generated.

Note: If you use wildcards when passing parameters to .jar files, enclose the wildcard parameters within quotes, for example:

```
java -jar Mapping.jar /InputFileName "altova-*.xml"
```


C#

To supply a custom input parameter to a C# command line application generated by MapForce:

1. If you haven't done that already, open the **FileNamesAsParameters.mfd** example from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
2. On the **File** menu, click **Generate Code In | C#**, and select a target directory (**C:\codegen\examples\cs**, in this example).
3. Open the solution in Visual Studio and build it (**Ctrl + Shift + B**).
4. Copy the **Altova_Hierarchical.xml** file from **<Documents>\Altova\MapForce2023\MapForceExamples** to the directory where **Mapping.exe** was generated (in this example, **C:\codegen\examples\cs\Mapping\bin\Debug**). As stated previously, the mapping will attempt to read this file if you do not supply a custom value to the **InputFileName** parameter.
5. Open a Command Prompt window and change to the directory where **Mapping.exe** is.

```
cd C:\codegen\examples\cs\Mapping\bin\Debug
```

6. Run the application with the following command:

```
Mapping.exe /OutputFileName output.xml
```

In the command above, the input parameter `/OutputFileName` supplies the name of the output file to be generated.

C++

To supply a custom input parameter to a C++ command line application generated by MapForce:

1. If you haven't done that already, open the **FileNamesAsParameters.mfd** example from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
2. On the **File** menu, click **Generate Code In | C++**, and select a target directory (**C:\codegen\examples\cpp**, in this example).
3. Open the solution in Visual Studio and build it (**Ctrl + Shift + B**).
4. Copy the **Altova_Hierarchical.xml** file from **<Documents>\Altova\MapForce2023\MapForceExamples** to the directory where **Mapping.exe** was generated (in this example, **C:\codegen\examples\cpp\Mapping\Debug**). As stated previously, the mapping will attempt to read this file if you do not supply a custom value to the **InputFileName** parameter.
5. Open a Command Prompt window and change to the directory where **Mapping.exe** is.

```
cd C:\codegen\examples\cpp\Mapping\Debug
```

6. Run the application with the following command:

```
Mapping.exe /OutputFileName output.xml
```

In the command above, the input parameter `/OutputFileName` supplies the name of the output file to be generated.

5.2 Simple Output

An output component (or "simple output") is a MapForce component which enables you to return a string value from the mapping. Output components represent one possible type of target components[76], but should not be confused with the latter. Use a simple output component when you need to return a string value from the mapping. On the mapping area, simple output components play the role of a target component which has a string data type instead of a structure of items and sequences. Consequently, you can create a simple output component instead of (or in addition to) a file-based target component. For example, you can use a simple output component to quickly test and preview the output of a function (see Example: Testing Function Output[845]). This technique is also useful for mappings which use string serialization (see String Parsing and Serialization[1324]). The main purpose of a simple output component is, however, to get back a string when calling the MapForce Server API, without writing any files.

Simple output components should not be confused with output parameters of user-defined functions (see User-Defined Functions[927]). There are some similarities and differences between the two, as follows.

Output components	Output parameters of user-defined functions
Added from **Function | Insert Output** menu.	Added from **Function | Insert Output** menu.
Have "string" as data type.	Can have simple as well as complex data types.
Applicable to the entire mapping.	Applicable only in the context of the function in which they were defined.

If necessary, you can add multiple simple output components to a mapping. You can also use simple output components in combination with file-based and database target components. When your mapping contains multiple target components, you can preview the data returned by a particular component by clicking the **Preview** () button in the component title bar, and then clicking the **Output** tab on the Mapping window.

You can use simple output components as follows in MapForce transformation languages:

Language	How it works
BUILT-IN (when previewing the mapping transformation)	You can preview Output components in the same way as you would preview a file-based mapping output—by clicking the **Output** tab on the Mapping window.
BUILT-IN (when running the MapForce Server execution file)	When you run a compiled MapForce Server execution file (see Compiling a MapForce mapping[1378]), the mapping output is returned in the standard output stream (stdout), so you can view it or redirect to a file. For example, assuming that the name of the MapForce server execution file is **MyMapping.mfx**, use the following syntax to redirect the mapping output to the **output.txt** file and any errors to the **log.txt** file: `MapForceServer.exe run MyMapping.mfx >output.txt 2>log.txt`
XSLT 1.0, XSLT 2.0, XSLT 3.0	In the generated XSLT files, a simple output component defined in the mapping becomes the output of the XSLT transformation.

<table>
<tr><th>Language</th><th>How it works</th></tr>
<tr><td></td><td>
If you are using RaptorXML Server, you can instruct RaptorXML Server to write the mapping output to the file passed as value to the <code>--output</code> parameter.
To write the output to a file, add or edit the <code>--output</code> parameter in the DoTransform.bat file. For example, the following DoTransform.bat file has been edited to write the mapping output to the Output.txt file (see highlighted text).
<pre>RaptorXML xslt --xslt-version=2 --
input="MappingMapToResult1.xslt" --output="Output.txt" %*
"MappingMapToResult1.xslt"</pre>
If an <code>--output</code> parameter is not defined, the mapping output will be written to the standard output stream (stdout) when the mapping is executed.
</td></tr>
<tr><td>C++, C#, Java</td><td>
In the generated C++, C#, and Java code, the mapping output is written to the standard output of the generated application.
If the mapping contains multiple target components, the generated application concatenates the standard output of each target component and returns it as one unified standard output.
</td></tr>
</table>

When you create a reversed mapping (using the menu command **Tools | Create Reversed Mapping**), the simple output component becomes a simple input component.

5.2.1 Adding Simple Output Components

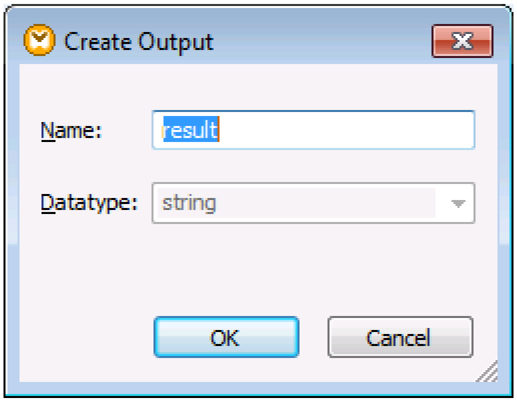
To add an output component to the mapping area:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. Do one of the following:

   a. On the **Function** menu, click **Insert Output**.

   b. Click the **Insert output** toolbar button.

3. Enter a name for the component.
4. Click **OK**.



Create Output dialog box

You can change the component name at any time later, in one of the following ways:

- Select the component, and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

5.2.2 Example: Previewing Function Output

This example illustrates how to preview the output returned by MapForce functions with the help of simple output components. You will make the most of this example if you already have a basic understanding of functions in general, and of MapForce functions in particular. If you are new to MapForce functions, you may want to refer to Using Functions[904] before continuing.

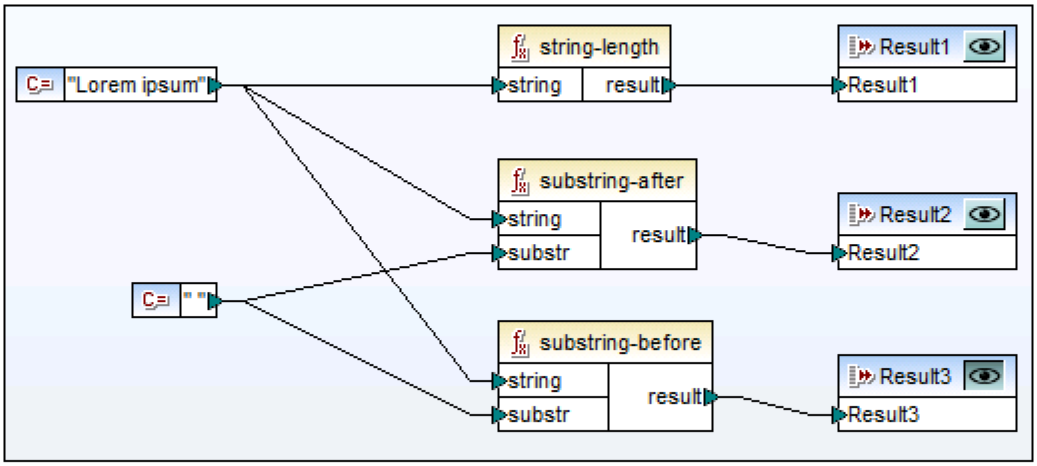
Our aim is to add a number of functions to the mapping area, and learn how to preview their output with the help of simple output components. In particular, the example uses a few simple functions available in the core library. Here is a summary of their usage:

`string-length`[1061]	Returns the number of characters in the string provided as argument. For example, if you pass to this function the value "Lorem ipsum", the result is "11", since this is the number of characters that the text "Lorem ipsum" takes.
`substring-after`[1062]	Returns the part of the string that occurs after the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "ipsum".
`substring-before`[1063]	Returns the part of the string that occurs before the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "Lorem".

To test each of these functions against a custom text value ("Lorem ipsum", in this example), follow the steps below:

1. Add a constant with the value "Lorem ipsum" to the mapping area (use the menu command **Insert | Constant**). The constant will be the input parameter for each of the functions to be tested.

2. Add the **string-length**, **substring-after**, and **substring-before** functions to the mapping area, by dragging them to the mapping area from the core library, **string functions** section.
3. Add a constant with an empty space (" ") as value. This will be the separator parameter required by the **substring-after** and **substring-before** functions.
4. Add three simple output components (use the menu command **Function | Insert Output**). In this example, they have been named *Result1*, *Result2*, and *Result3*, although you can give them another title.
5. Connect the components as illustrated below.



Testing function output with simple output components

As shown in the sample above, the "Lorem ipsum" string acts as input parameter to each of the **string-length**, **substring-after**, and **substring-before** functions. In addition to this, the **substring-after** and **substring-before** functions take a space value as second input parameter. The **Result1**, **Result2**, and **Result3** components can be used to preview the result of each function.

To preview the output of any function:

- Click the **Preview** () button in the component title bar, and then click the **Output** tab on the Mapping window.

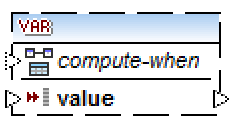
5.3 Variables

A variable is a special type of components used to store an intermediate mapping result for further processing. Variables can be of simple type (e.g., string, integer, boolean, etc) and complex type (a tree structure). See the examples of both types in the subtopics below.

One of the most important aspects of variables is that they are sequences and can be used to create sequences. The term *sequence* means a list of zero or more items. This makes it possible for a variable to process multiple items for the duration of the mapping lifetime. For more information, see also Mapping Rules and Strategies[1275]. However, it is also possible to assign a value to a variable once and keep this value the same for the rest of the mapping. For details, see Changing the Context and Scope of Variables[853].

Simple variables

A simple variable is built to represent atomic types such as strings, numbers, and booleans (*see screenshot below*).



Complex variables

A complex variable has a tree structure. The structures on which a complex variable can be based are summarized in the list below.

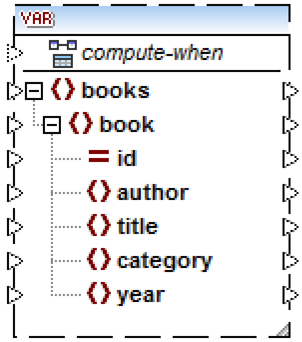
MapForce Basic Edition:
- XML Schema Structure

MapForce Professional Edition:
- XML Schema Structure
- Database Structure

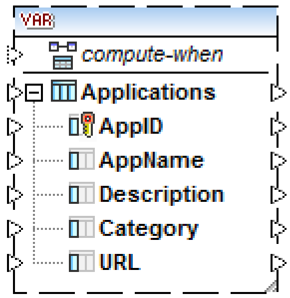
MapForce Enterprise Edition:
- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure
- JSON Schema Structure

Example 1: Variable based on XML Schema

You can create a complex variable by supplying an XML schema which defines the structure of the variable (*see screenshot below*). If the schema defines any elements globally, you can choose which one should become the root node of the variable structure. Note that a variable does not have an associated instance XML file. The data of the variable is computed at mapping runtime.



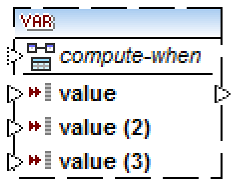
Example 2: Variable based on a database (MapForce Professional and Enterprise editions)
If you choose a database structure for your variable (*see screenshot below*), you can choose a specific database table as the root item for the variable structure. MapForce allows you to create DB-based variables with a tree of related tables. The tree of related tables represents an in-memory structure that has no connection to the database at runtime. This also means that there is no automatic handling of foreign keys and no table actions in parameters or variables.



Compute-when
In both examples above, each variable has an item called `compute-when`. Connecting this item is optional: This enables you to control how the variable value should be computed in the mapping. For more information, see Changing the Context and Scope of Variables 853.

Variables with duplicated inputs

When necessary, items of a variable structure can be duplicated to accept data from more than one source connection. This is similar to duplicating inputs 83 in standard components. This does not apply, however, to variables created from database tables. The screenshot below illustrates a simple variable with duplicated inputs.



Chained mappings vs. variables

Variables can be compared to intermediate components of a chained mapping[1246]. However, variables are more flexible and convenient if you do not need to produce intermediary files at each stage of the mapping. The table below outlines differences between variables and chained mappings.

Chained mappings	**Variables**
Chained mappings involve two independent steps. For example, a mapping has three components, namely A, B, and C. Step 1: mapping data A to B. Step 2: mapping data from B to C.	You can control when and how often the variable value is computed when the mapping is carried out. For details, see Changing the Context and Scope of Variables[853].
When the mapping is carried out, intermediate results are stored externally in files.	When the mapping is carried out, intermediate results are stored internally. No external files containing the results of a variable are produced.
The intermediate result can be previewed using the preview button.	The result of a variable cannot be previewed, since it is computed at mapping runtime. You can, however, use the debugger[1349] to see the data produced by the variable.

Note: Variables are not supported if the mapping transformation language is set to XSLT 1.0.

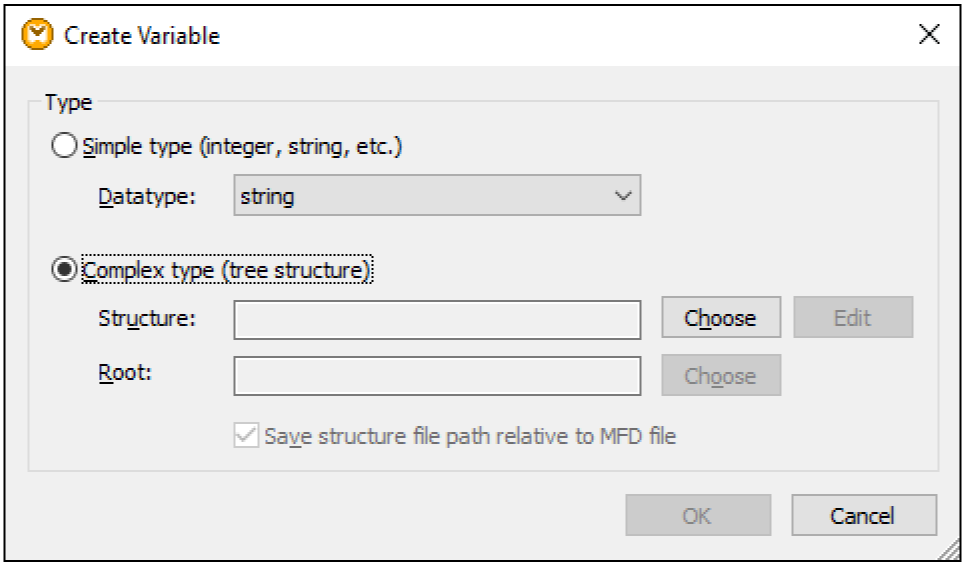
5.3.1 Add a Variable

This topic explains how to add a variable to a mapping. The first option is to add a variable via the menu or toolbar command. The second option allows you to add a variable via the context menu.

Option 1: via the menu or toolbar command

This option enables you to add a variable via the menu or toolbar command. Take the steps below:

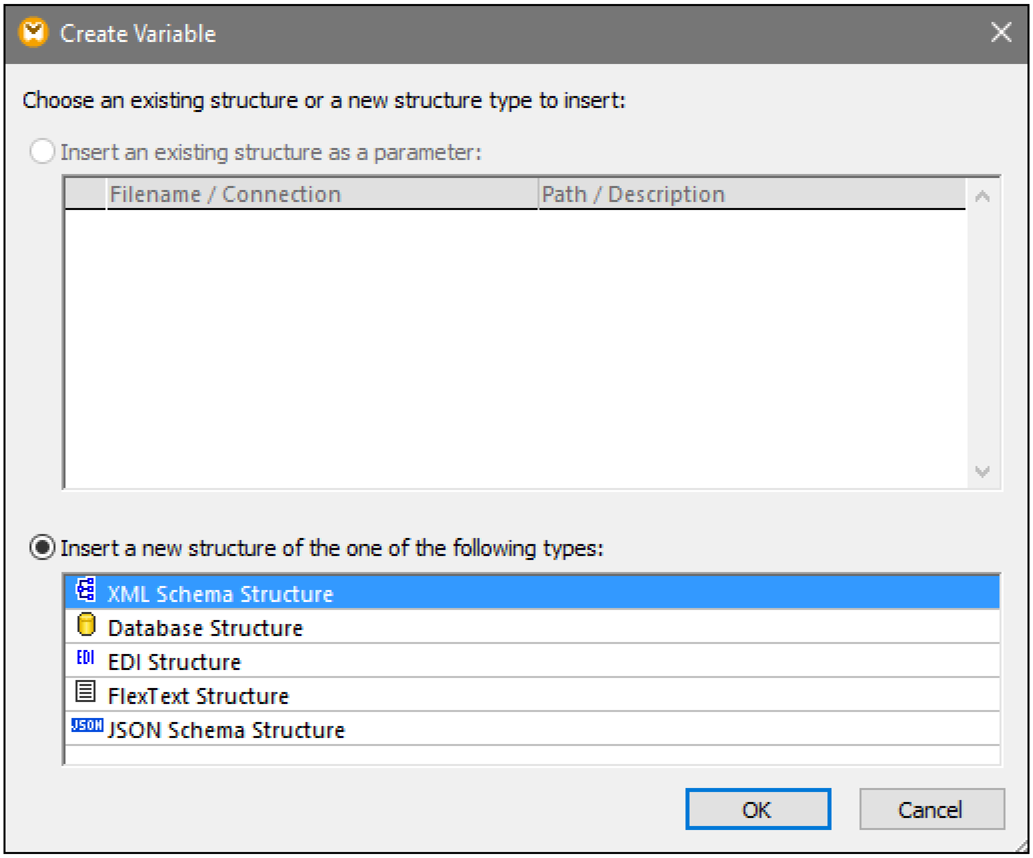
1. Go to the **Insert** menu and click **Variable**. Alternatively, click the toolbar button (**Variable**).



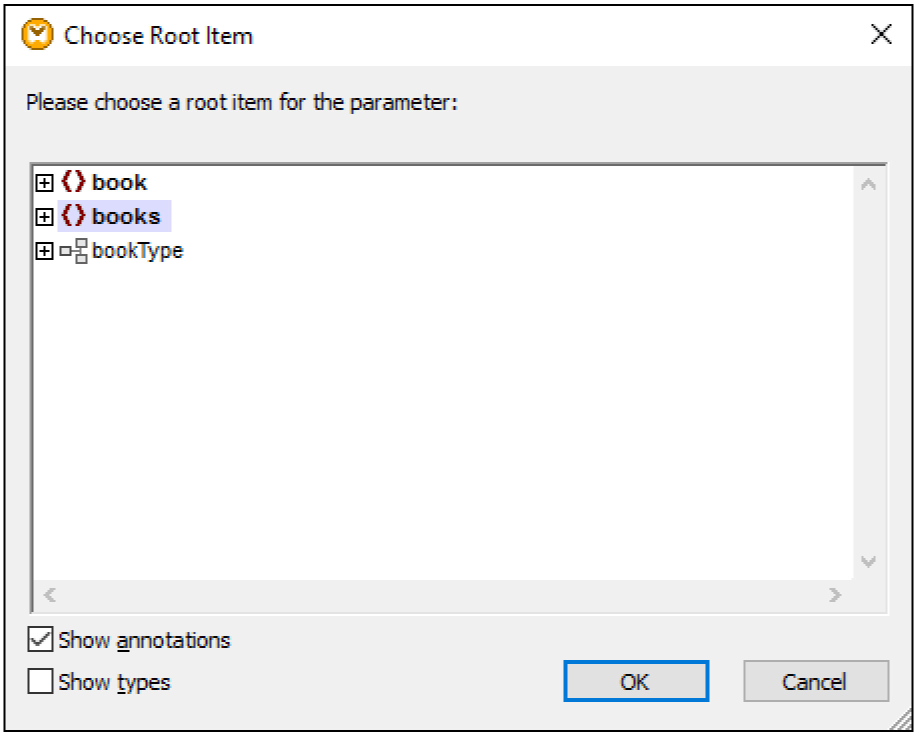
2. Select the type of variable you want to insert (simple or complex type).

If you select **Complex type**, there are a few additional steps:

3. Click **Choose** to select the source which should provide the structure of the variable [847]. The structures illustrated in the screenshot below only apply to MapForce Enterprise Edition. See the list of structures relevant to other MapForce editions in the previous topic [847].



4. When prompted, specify the root item of the structure of the variable. For example, in XML schemas, you can select any element or type from the selected source (*see screenshot below*).

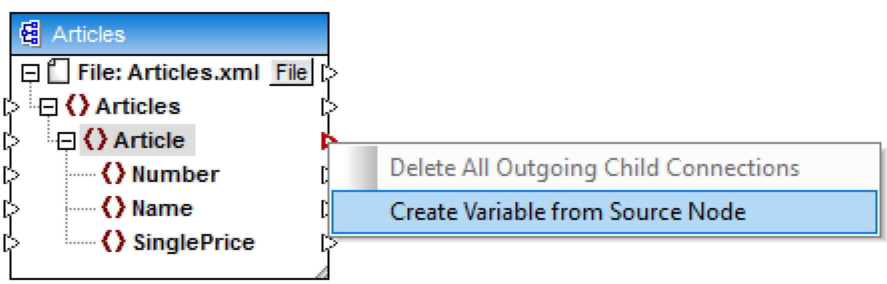


Option 2: via the context menu

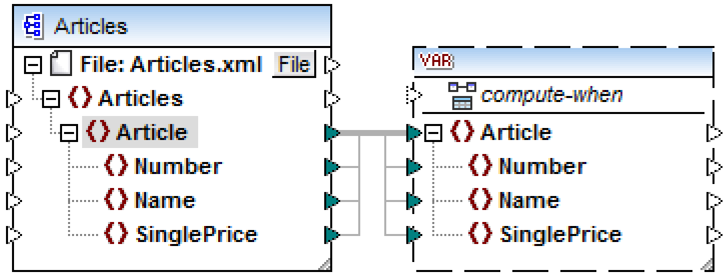
The second option allows you to create a variable using the context menu. The possible options are listed below.

Variable from a source node

To create a variable from a source node, right-click the output connector of a component (in this example, the output connector of the `<Article>` element) and select **Create Variable from Source Node** (*see screenshot below*).



This creates a complex variable with the source schema of the `Articles` component. All the items are automatically connected with a copy-all connection 97 (*see screenshot below*).

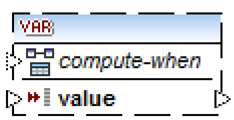


Variable from a target node
To create a variable from a target node, right-click the input connector of a target component and select **Create Variable for Target Node**. This creates a complex variable with the same schema as in the target. All the items are automatically connected with a copy-all connection.

Variable from a filter:
To create a variable using a filter, right-click the output connector of a filter component (`on-true/on-false`) and select **Create Variable from Source Node**. This creates a complex component with the source schema and automatically uses the item linked to the filter input as the root element of the intermediate component.

5.3.2 Scope and Context of Variables

Every variable has a `compute-when` input item (*see screenshot below*), which allows you to control the scope of the variable. This means that you can control when and how often the variable value is computed when the mapping is executed. You do not have to connect this input in many cases, but it can be essential to override the default context or to optimize the mapping performance.



The following terms are relevant to the discussion of the scope and context of variables: *subtree* and *variable value*. A subtree is a set of an item/node in a target component and all of its descendants: for example, a `<Person>` element with its `<FirstName>` and `<LastName>` child elements.

A variable value is the data that is available at the output side of the variable component.

- For simple variables, it is a sequence of atomic values that have the datatype specified in the component properties.
- For complex variables, it is a sequence of root nodes (of the type specified in the component properties), each one including all its descendant nodes.

The sequence of atomic values (or nodes) may contain one or even zero elements. This depends on what is connected to the input side of the variable, and to any parent items in the source and target components.

Compute-when is not connected (default)

If the `compute-when` input item is not connected to an output node of a source component, the variable value is computed *whenever it is first used in a target subtree* directly via a connector from the variable component to a node in the target component or indirectly via functions. The same variable value is also used for all target child nodes inside the subtree.

The actual variable value depends on any connections between parent items of the source and target components. This default behavior is the same as that of complex outputs of regular user-defined functions 930 and Web service function calls. If the variable output is connected to multiple unrelated target nodes, the variable value is computed *separately for each of them*. This can produce different results in each case, because different parent connections influence the context in which the variable's value is evaluated.

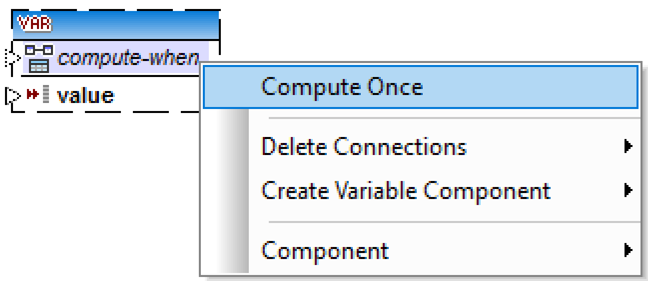
Compute-when is connected

By connecting an output connector of a source component to `compute-when`, the variable is computed *whenever that source item is first used in a target subtree*.

The variable actually acts as if it were a child item of the item connected to `compute-when`. This makes it possible to bind the variable to a specific source item. That is, at runtime the variable is re-evaluated whenever a new item is read from the sequence in the source component. This is related to the general rule governing connections in MapForce: For each source item, one target item is created. In this case, `compute-when` instructs MapForce to compute the variable value for each source item. For more information, see Mapping Rules and Strategies 1275.

Compute-once

If necessary, you can choose to compute the variable value *once before each of the target components*, making the variable essentially a global constant for the rest of the mapping. To do this, right-click the `compute-when` item and select **Compute Once** from the context menu:



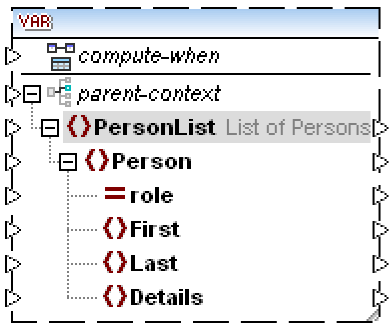
When you change the scope of a variable to `compute-when=once`, the input connector is removed from the `compute-when` item, since such a variable is only evaluated once. In a user-defined function, the `compute-when=once` variable is evaluated each time the function is called before the actual function result is evaluated.

Parent-context

The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g.,

min, **max**, **avg**, **count**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.

Adding a `parent-context` item may be necessary, for example, if your mapping uses multiple filters and you need an additional parent node to iterate over. For details, see Example: Changing the Parent Context 1284. To add a parent-context to a variable, right-click the root node (in this example, `PersonList`) and select **Add Parent Context** from the context menu. This adds a new node, `parent-context`, to the existing hierarchy.

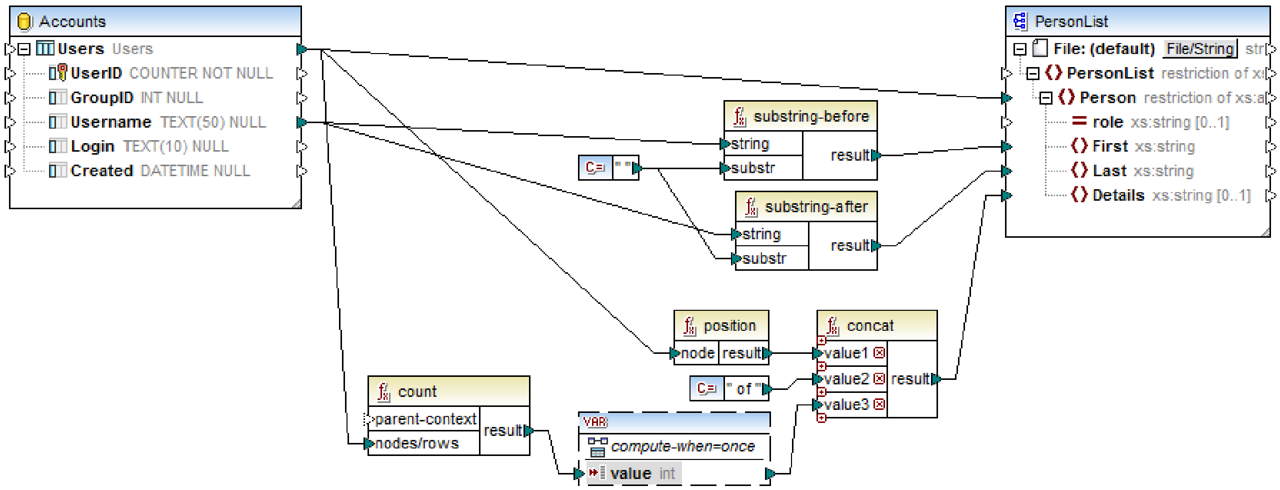


The parent context adds a virtual parent node to the hierarchy within the component. This allows you to iterate over an additional node in the same or in a different source component.

5.3.3 Example: Counting Database Table Rows

The mapping illustrated in this example is available as **DB_UserList.mfd** in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. This mapping extracts user records from a database table called "Users" and writes them to an XML file. The database column "Username" contains both the first name and the surname of a person (for example, "Vernon Callaby"). This mapping has the following goals:

1. For each record in the "Users" table, create a new `Person` element in the XML file.
2. Split the value extracted from the database field "Username" into two separate fields in the XML file ("First" and "Last").
3. For each record, find its sequential number compared to the number of total records present in the database (for example, "Record 1 of 4") and write this information to the `Details` element.



DB_UserList.mfd

As illustrated above, in order to achieve the first goal, a connection is drawn between the source "Users" table and the `Person` element of the target XML file. This ensures that, for each record in the source table, a new `Person` element will be created in the target.

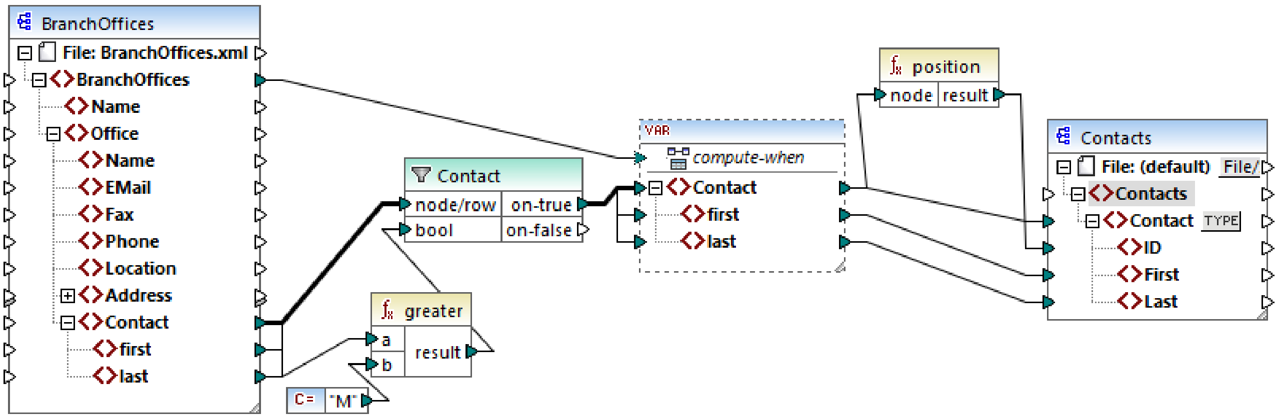
The value of the field "Username" is supplied to the **substring-before** 1063 and **substring-after** 1062 functions. These two functions extract the text before and after the space character (" "), respectively, which takes care of the second mapping goal.

Finally, to achieve the third goal, the mapping uses the **count** function. The result of the count function is passed on to a variable. The variable ensures that this result is stored on the mapping and available when writing the "Details" element of each person to the target XML. Note that, for efficiency reasons, database records should be counted only once, so the variable scope is set to `compute-when=once` (see Changing the Context and Scope of Variables 853)

5.3.4 Example: Filtering and Numbering Nodes

The mapping illustrated in this example is available as **PositionInFilteredSequence.mfd** in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder.

This mapping reads an XML file which contains contact data of several people, filters them, and writes them to a target XML file. The goal of the mapping is to filter from the source XML file only those people whose last name begins with letter "M" or a subsequent letter. Secondly, the extracted contacts must be numbered. The number is going to act as the unique identifier of each contact in the target XML file.



PositionInFilteredSequence.mfd

To achieve the goal above, the following component types were added to the mapping:

- A filter (see Filters and Conditions [878])
- A complex variable (see Adding Variables [849])
- The functions **greater** [1012] and **position** [1049] (see Add a Function to the Mapping [905])
- A constant (To add a constant, select the menu command **Insert | Constant**).

The variable uses the same schema as the source component. If you right-click the variable and select **Properties** from the context menu, notice that the node **BranchOffices/Office/Contact** is selected as root node for this variable structure.

First, data of the source component is passed on to the filter. The filter passes onwards to the variable only those records that meet the filter condition. Namely, the filter is configured to get only those `Contact` nodes where the last name is equal or greater than M". To achieve this, the function **greater** [1012] compares each `last` item with the constant value "M".

The variable has the `compute-when` input connected to the root item of the source component (`BranchOffices`). At runtime, this causes the variable to be re-evaluated whenever a new item is read from the sequence in the source component. In this mapping, however, connecting or not connecting the `compute-when` item does not make a difference. The reason is that the variable is connected to the `Contact` source item (indirectly through the filter), and it would compute as many times as there are instances of `Contact` which meet the filter condition.

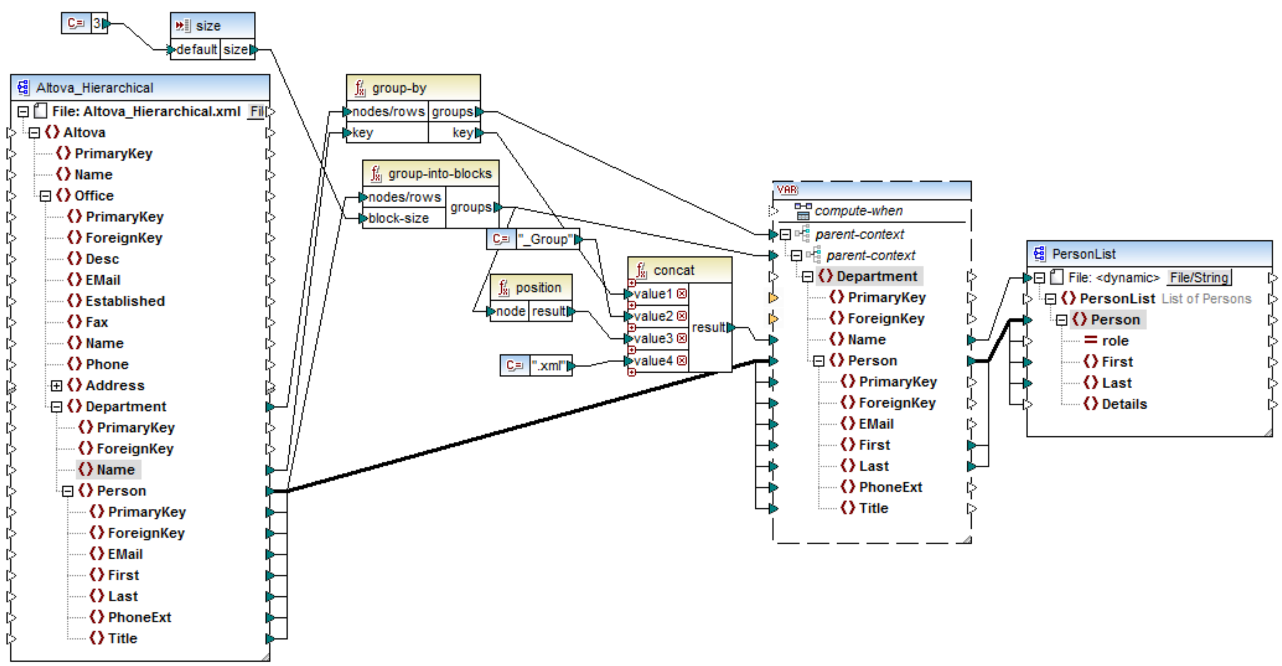
The **position** [1049] functions returns, for each iteration of the variable, the number of the current sequence. Only eight contacts meet the filter condition; therefore, if you preview the mapping and look at the output, notice how IDs 1 through 8 were written to the `ID` element of the target component.

In case you were wondering why the variable was necessary at all, it is because of the requirement to number all records. Had we connected the filter result directly to the target component, there would have been no way to number each occurrence of `Contact`. The purpose of the variable in this mapping is, therefore, to store each instance of `Contact` temporarily on the mapping, so that it can be numbered before it is written to the target.

5.3.5 Example: Grouping and Subgrouping Records

The mapping illustrated in this example is available as **DividePersonsByDepartmentIntoGroups.mfd** in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder.

This mapping processes an XML file that contains employee records of a fictitious company. The company has two offices: "Nanonull, Inc." and "Nanonull Partners, Inc". Each office has several departments (for example, "IT", "Marketing", and so on), and each department has one or more employees. The goal of the mapping is to create groups of maximum three people from each department, regardless of the office. The size of each group is three by default; however, it should be easy to change if necessary. Each group must be saved as a separate XML file, with the name having the format "<Department Name>_GroupN" (for example, **Marketing_Group1.xml**, **Marketing_Group2.xml**, and so on).



DividePersonsByDepartmentIntoGroups.mfd

As illustrated above, in order to achieve the mapping goal, a complex variable was added to the mapping, and a few other component types (primarily functions). The variable has the same structure as a `Department` item in the source XML. If you right-click the variable in order to view its properties, you will notice that it uses the same XML schema as the source component, and has `Department` as root element. Importantly, the variable has two nested `parent-context` items, which ensure that the variable is computed first in the context of each department, and then in the context of each group within each department (see also Changing the Context and Scope of Variables 853).

Initially, the mapping iterates through all departments in order to obtain the name of each department (this will be subsequently required to create the file name corresponding to each group). This is achieved by connecting the **`group-by`** 1037 function to the `Department` source item, and by supplying the department name as grouping key.

Next, within the context of each department, a second grouping takes place. Namely, the mapping calls the **group-into-blocks**[1041] function in order to create the required groups of people. The size of each group is supplied by a simple input component which has a default value of "3". The default value is supplied by a constant. In this example, in order to change the size of each group, one can easily modify the constant value as required. However, the "size" input component can also be modified so that, if the mapping is run by generated code or with MapForce Server, the size of each group could be conveniently supplied as a parameter to the mapping. For more information, see Supplying Parameters to the Mapping[833].

Next, the value of the variable is supplied to the target PersonList XML component. The file name for each created group was computed by concatenating the following parts, with the help of the **concat**[1057] function:

1. The name of each department
2. The string "_Group"
3. The number of the group in the current sequence (for example, "1" if this is the first group for this department)
4. The string ".xml"

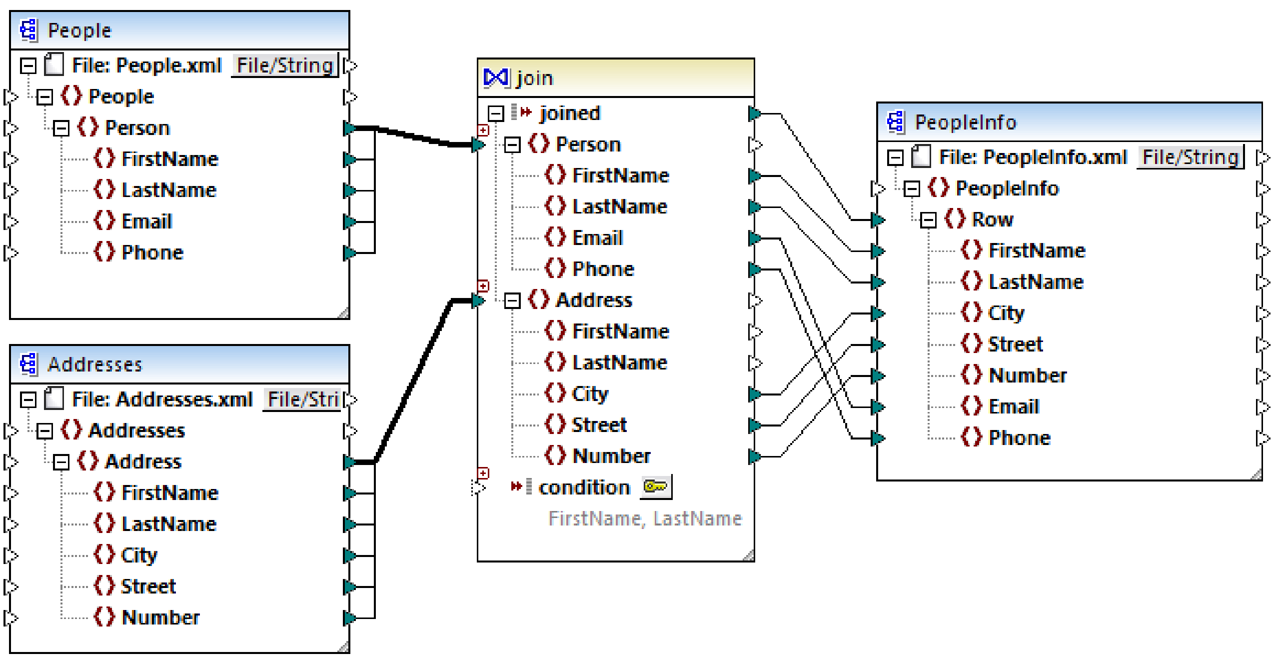
The result of this concatenation is stored in the `Name` item of the variable, and then supplied as a dynamic file name to the target component. This causes a new file name to be created for each received value. In this example, the variable computes eight groups in total, so eight output files are created when the mapping runs, as required. For more information about this technique, see Processing Multiple Input or Output Files Dynamically[1313].

5.4 Join Components

Sometimes, you may need to combine data from two or more structures based on some condition (for example, if field A in the first structure has the same value as field B in the second structure). For such mapping requirements, a Join component can be used.

A Join component is a MapForce component which enables joining two or more structures on the mapping based on custom-defined conditions. It returns the association (joined set) of items that satisfy the condition. Joins are particularly useful to combine data from two structures which share a common field (such as an identity).

For example, on the mapping illustrated below, the middle component is a "Join" component. In this mapping, two XML structures (a list of people and a list of addresses) are being joined. The goal here is to get the full details of each person into a target XML file. The `FirstName` and `LastName` fields act as joining keys. Namely, if value of `FirstName` and `LastName` (under `Person`) is the same as that of `FirstName` and `LastName` (under `Address`), the address details belong to one and the same person and they become "joined". Any items from the joined structure can further be mapped to a subsequent target (in this case, an XML file). The join condition itself is defined in the properties of the Join component, by clicking the **Define Join Condition** () button. This example is accompanied by a mapping sample and is explained in more detail in Example: Join XML Structures 866.



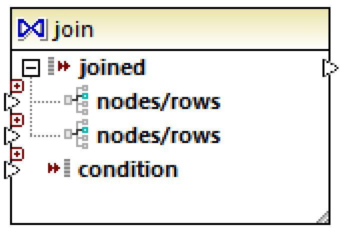
JoinPeopleInfo.mfd

As illustrated above, the source structures and the Join component are connected by means of "Copy-All" connection, which reduces the mapping clutter. In general, such connections are created automatically by MapForce when the context is relevant (for more information, see Copy-All Connections 97).

The structures that are to be joined may either be from separate components (as in the mapping above), or belong to the same component. The structures to be joined may also be of different kinds (for example, an XML structure and a database table). For more information about database-related joins, see Joining Database Data[333].

To add a Join component:

1. Set the mapping transformation language to BUILT-IN (to do this, either click the toolbar button, or use the **Output | Built-In Execution Engine** menu command).
2. On the **Insert** menu, click **Join**. Alternatively, click the **Join** ( ) toolbar button. The Join component appears on the mapping. By default, it accepts data from two structures, so it has two `nodes/rows` inputs. If necessary, you can add new inputs to the join by clicking the **Add Input** ( ) button, see Joining Three or More Structures[865].



3. Connect the structures that are to be joined to the `nodes/rows` items of the join component.
4. Add the condition for the join (or multiple conditions). To do this, right-click the Join component and select **Properties**. Join conditions can also be added directly from the mapping, by connecting the Boolean result of some function to the `condition` item of the Join component. In certain cases when database tables are joined, the join condition (or conditions) can be created automatically by MapForce. For further information, see Adding Join Conditions[862].

Notes:

- Join components are supported when the target language of the mapping is set to BUILT-IN. Code generation in C#, C++, or Java is not supported.
- When a structure is not a valid or supported input source for the join, MapForce displays hints either immediately directly on the mapping, or in the Messages window, when you validate the mapping (see Validating Mappings[105]).
- Join components should not be connected to input parameters or results of inline user-defined functions. If such connections exist, validation errors will occur during mapping validation.
- When you connect eligible database components (such as tables or views) directly to a Join component, an **SQL mode** ( ) button automatically appears at the top-right corner of the Join component. When enabled, this button provides special SQL features applicable to the join operation (see Joins in SQL Mode[334]).
- It is not possible to connect the output of the `joined` item to another Join component. If necessary, however, you can connect a partial result of one join to another one.

Join components compared to other component types

In some cases, complex variables or filters can be used instead of Join components to achieve the same results (see Using Variables[847] and Filters and Conditions[878], respectively). However, unlike other component types, Join components make the mapping easier to understand, because you can see at a glance the data that is being joined. Additionally, if SQL mode is enabled on the join component, the mapping performance improves significantly (this applies to database joins, see Joining Database Tables[333]).

Adding a parent context

In some special cases, in order to achieve a specific mapping result, you can explicitly provide a mapping context (a so-called "parent context") for data connected to the Join component. To add a parent context, right-click the `joined` item of the Join component, and select **Add Parent Context** from the context menu. The Join component changes appearance to include an additional `parent-context` input where you can connect the required source item. For more information, see Example: Changing the Parent Context [1284].

The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.

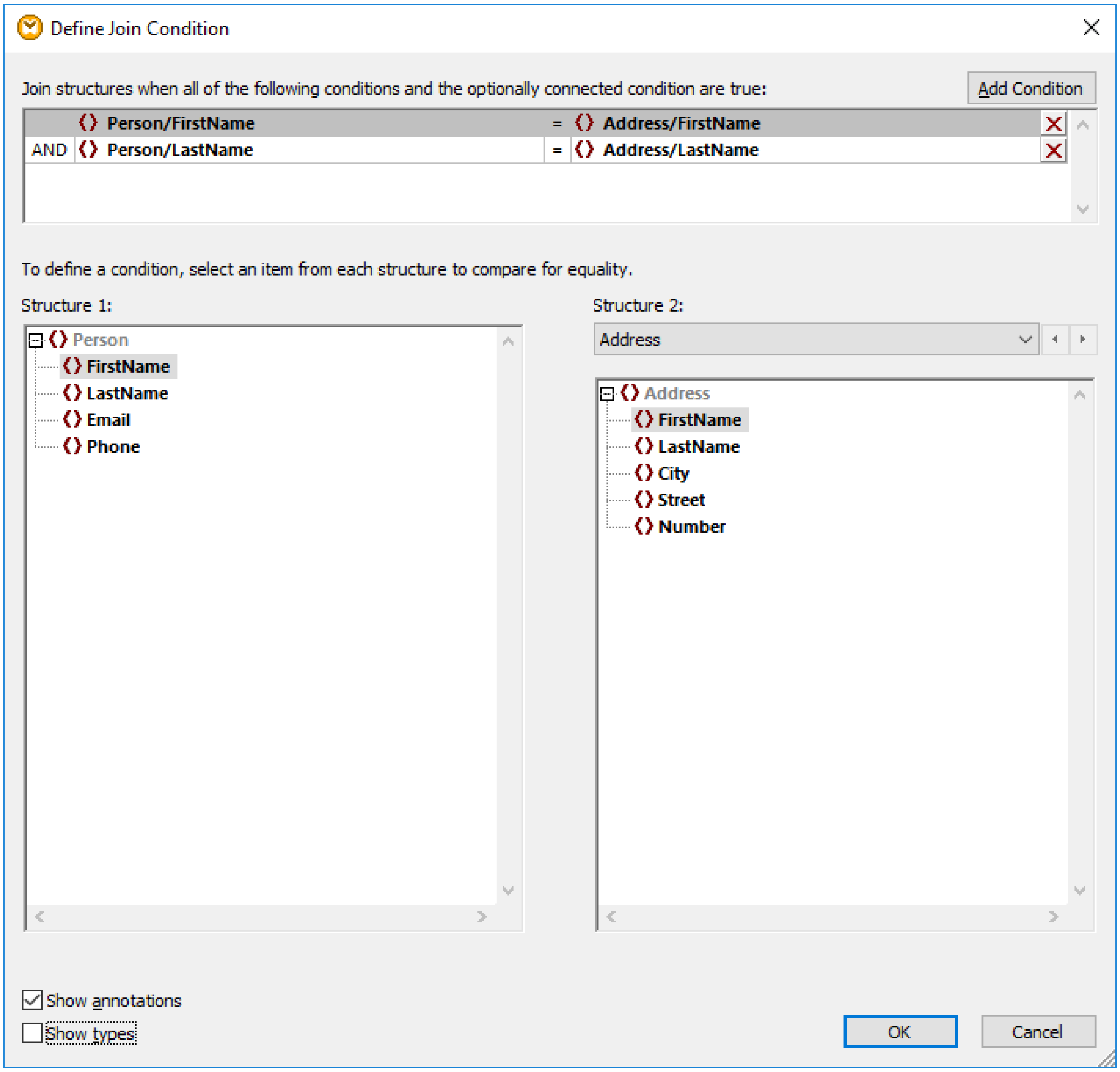
5.4.1 Adding Join Conditions

A join works by combining items of two or more structures according to a condition, so a join always requires at least one condition. There are several ways to add join conditions, as shown below.

Note: When database tables are joined in SQL mode, MapForce will create the join condition (or conditions) automatically, based on foreign key relationships detected between tables. For automatic join conditions to happen, the database tables must be in a child-parent relationship on the MapForce component (that is, one table must be "parent" or "child" of another one on the component), see Example: Join Tables in SQL Mode [337].

Approach 1: Add a join condition from the component properties

1. On the mapping, make sure that at least two structures (or database tables) are connected to the Join component. The Join component illustrated in this example is part of the **JoinPeopleInfo.mfd** mapping available in the folder **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**. This mapping is discussed in more detail in Example: Join XML Structures [866].
2. On the Join component, click the **Define Join Condition** ( ) button (or right-click the header of the component, and select **Properties** from the context menu).
3. Select an item from the left structure and another one from the right structure (that is, whenever the comparison of this pair returns true, the left and right structures become joined).



If you need to add multiple conditions, click **Add Condition**, and then select a new pair of items. For example, in the image above, two join conditions are defined:

1. `FirstName` in the Structure 1 must be equal to `FirstName` in Structure 2, and
2. `LastName` in Structure 1 must be equal to `LastName` in Structure 2.

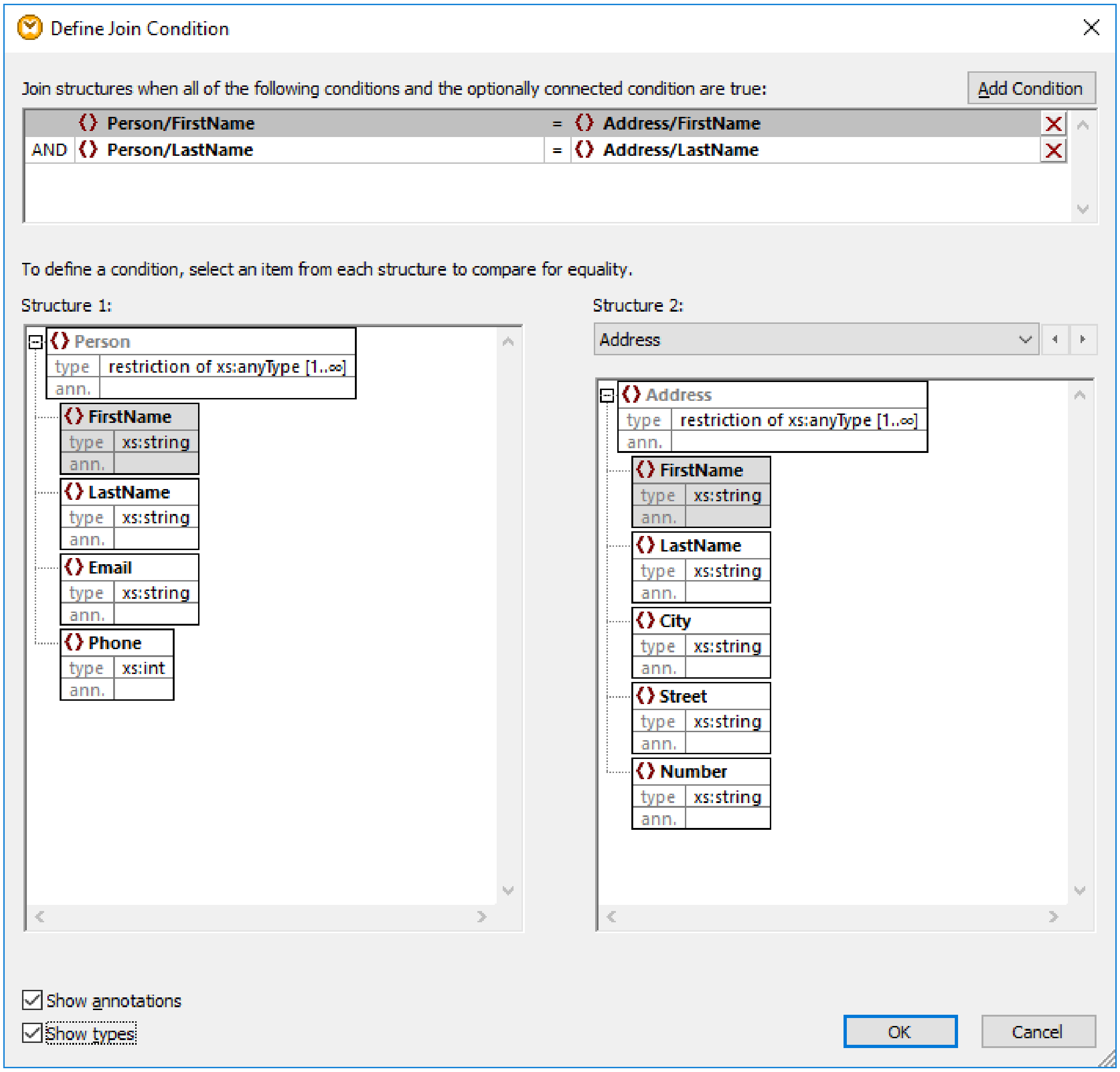
To remove a join condition, click the **Delete** button next to it.

Notes:

- When multiple join conditions exist, all of them must be satisfied in order for the two structures to be joined. In other words, multiple conditions are joined by a logical AND operation. This also includes optional conditions that were added from the mapping (see Approach 2 below).
- If more than two structures are connected to the Join component, such additional structures appear in the drop-down list below "Structure 2". When you select such an additional structure from the drop-

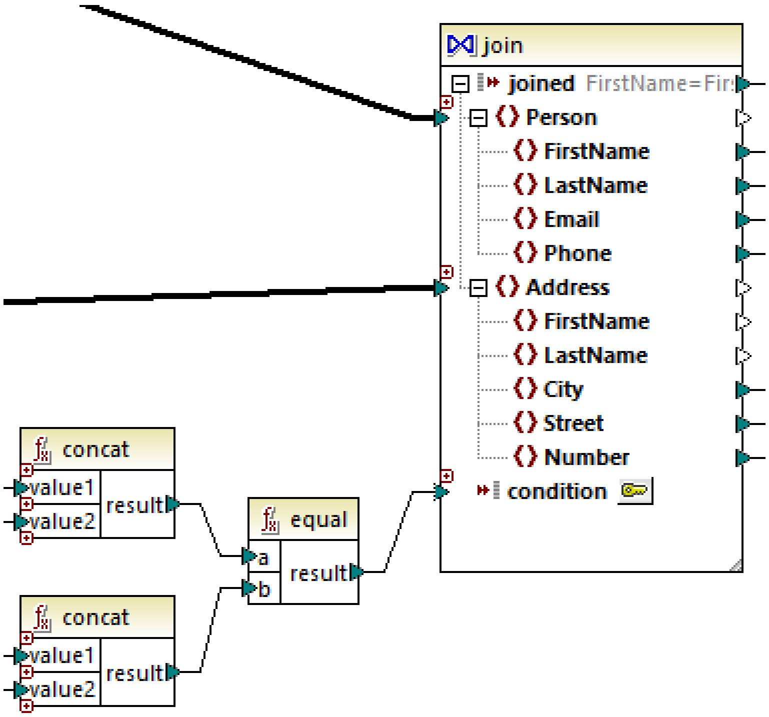
down list, the left pane displays all structures that occur *before* it on the Join component. This way you can define join conditions between any of the multiple structures. For an example, see [Example: Create CSV Report from Multiple Tables](346).

- To view the data type of items in each structure, select the **Show types** check box. The **Show annotations** option displays additional information about items, provided that such information exists in the underlying schema (or database). If both check boxes are selected, the layout changes to accommodate the display of both annotations and types, for example:



Approach 2: Add a join condition from the mapping

- On the mapping, add components which produce a Boolean value, and then connect the Boolean output to the input of the `condition` item. For example, the **`equal`** function may compare a value with some mapping item, and supply the Boolean result as input to the `condition` item of the join component.



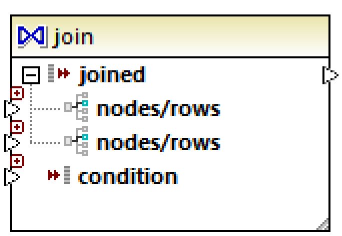
Note: If no condition is defined from the join component properties (Approach 1), the `condition` item of the join component must be connected (Approach 2).

Approach 3: Mixed approach

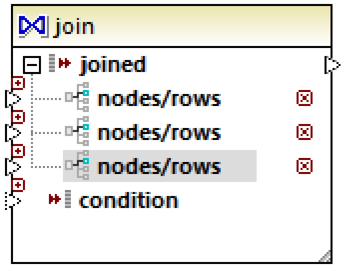
In the same mapping, it is possible to define some join conditions in component properties (Approach 1) and combine them with the one from the mapping (Approach 2). However, if you intend to join database tables in SQL mode, the conditions must be defined strictly using Approach 1, see also Joins in SQL Mode[334].

5.4.2 Joining Three or More Structures

When you add a Join component to the mapping using the menu command **Insert | Join**, it accepts two structures by default (that is, the component contains only two `nodes/rows` inputs).



If you need to join more than two structures, click the **Add input** (⊞) button and create as many `nodes/rows` as necessary. If you need to remove a `nodes/rows` input, click the **Delete input** (⊠) button. Note that a join requires at least two structures, so the ⊠ button is only available when more than two inputs exist.



When a join has multiple inputs, the join conditions must accordingly take into consideration each of the inputs that you want to be joined, see Adding Join Conditions 862. For a step-by-step example of how to join multiple database tables, see Example: Create CSV Report from Multiple Tables 346.

5.4.3 Example: Join XML Structures

This example shows you how to combine data from two XML structures conditionally, by using a join component. The example is accompanied by a mapping sample which is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\JoinPeopleInfo.mfd**.

The purpose of this mapping is to collect people data (name, surname, address, email, and phone) from two source XML files into a single target XML file.

The first XML file stores the name and surname of each person, as well as their email and phone, as shown in the sample code listing below (note that the XML declaration, namespaces, and some records have been omitted, for simplicity):

```
<People>
   <Person>
      <FirstName>Marquita</FirstName>
      <LastName>Bailey</LastName>
      <Email>m.bailey@nanonull.com</Email>
      <Phone>555323698</Phone>
   </Person>
   <Person>
      <FirstName>Totie</FirstName>
      <LastName>Rea</LastName>
      <Email>t.rea@nanonull.com</Email>
      <Phone>555598653</Phone>
   </Person>
</People>
```

People.xml

The second XML file stores the name and surname of each person, as well as their address details:

```
<Addresses>
   <Address>
      <FirstName>Marquita</FirstName>
      <LastName>Bailey</LastName>
```

```
      <City>Bridgedell</City>
      <Street>Olive Street</Street>
      <Number>4</Number>
   </Address>
   <Address>
      <FirstName>Totie</FirstName>
      <LastName>Rea</LastName>
      <City>Roseford</City>
      <Street>Evergreen Lane</Street>
      <Number>34</Number>
   </Address>
</Addresses>
```

Addresses.xml

The goal of the mapping is to combine the <Person> information from the first file with <Address> information from the second file, wherever the first and last names match. Specifically, for each <Person> in the first file, and for each <Address> in the second file, the `FirstName` and `LastName` must be compared. If both values are the same, then the corresponding <Person> and <Address> records refer to the same person, and must be joined. The target XML structure should look like this:

```
<PeopleInfo>
   <Row>
      <FirstName>Marquita</FirstName>
      <LastName>Bailey</LastName>
      <City>Bridgedell</City>
      <Street>Olive Street</Street>
      <Number>4</Number>
      <Email>m.bailey@nanonull.com</Email>
      <Phone>555323698</Phone>
   </Row>
   <Row>
      <FirstName>Totie</FirstName>
      <LastName>Rea</LastName>
      <City>Roseford</City>
      <Street>Evergreen Lane</Street>
      <Number>34</Number>
      <Email>t.rea@nanonull.com</Email>
      <Phone>555598653</Phone>
   </Row>
</PeopleInfo>
```

PeopleInfo.xml

This mapping goal can be easily achieved by adding a Join component to the mapping. Note that it is also possible to achieve the same result using other component types; however, in the steps below, you will be using a Join component, which is the subject of this example.

To create the required mapping, follow the steps below.

Step 1: Add the source XML files to the mapping

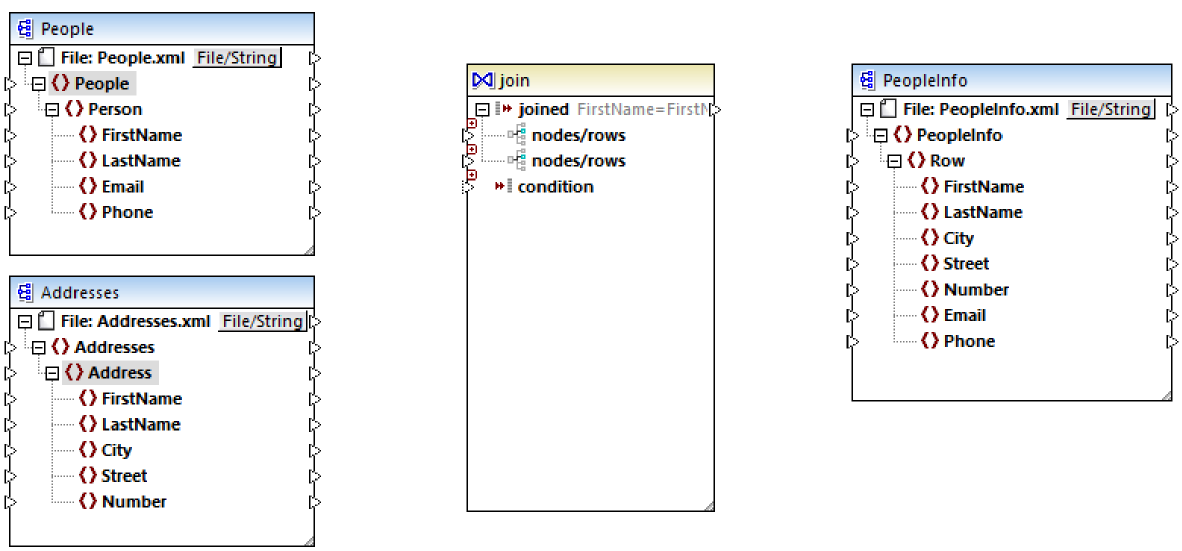
1. On the **Insert** menu, click **XML Schema/File**, and browse for the following source file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\People.xml**.
2. Repeat the step above for **Addresses.xml** (the second source file).

Step 2: Add the target schema file to the mapping

- On the **Insert** menu, click **XML Schema/File**, and browse for **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\PeopleInfo.xsd** (the target XSD schema file). When prompted to supply a sample XML file, click **Skip**. When prompted to select a root element, select `PeopleInfo` as root element.

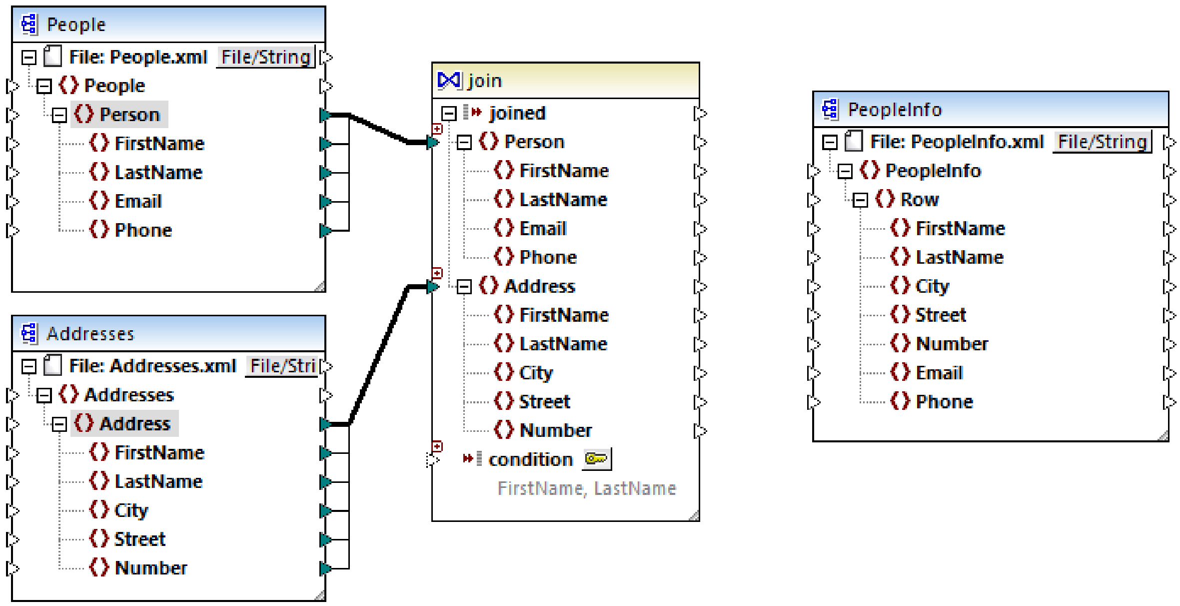
Step 3: Add the Join component

1. On the **Insert** menu, click **Join**. (Alternatively, click the **Join** toolbar button). At this stage, the mapping should look as follows (you will need to drag and resize the components in order to make them look as illustrated below):

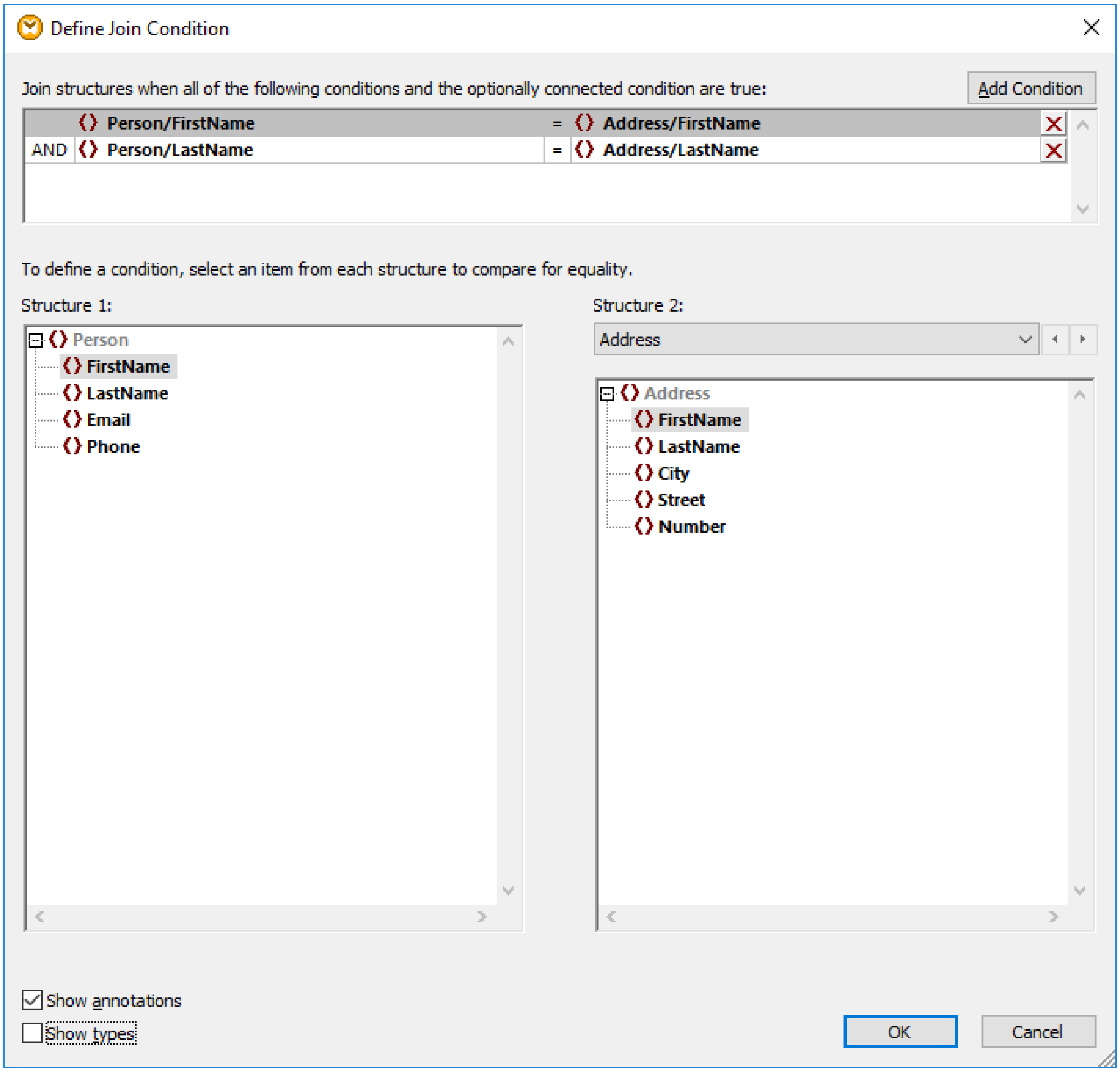


Observe the structure of the Join component. It has two `nodes/rows` items, which makes it possible to connect to it the two structures that need to be compared (in this case, the `Person` and the `Address` structures).

2. Draw a connection from `Person` to the first `nodes/rows` item of the Join component. Likewise, connect `Address` to the second `nodes/rows` item.



3. As mentioned earlier, the join should take place only if the `FirstName` and `LastName` values are equal in both structures. To define this condition, click the **Define Join Condition** button.
4. Select the pair of items that define the first join condition (`FirstName` under Structure 1, and `FirstName` under Structure 2).
5. Click **Add Condition**, and repeat the step above for `LastName`.



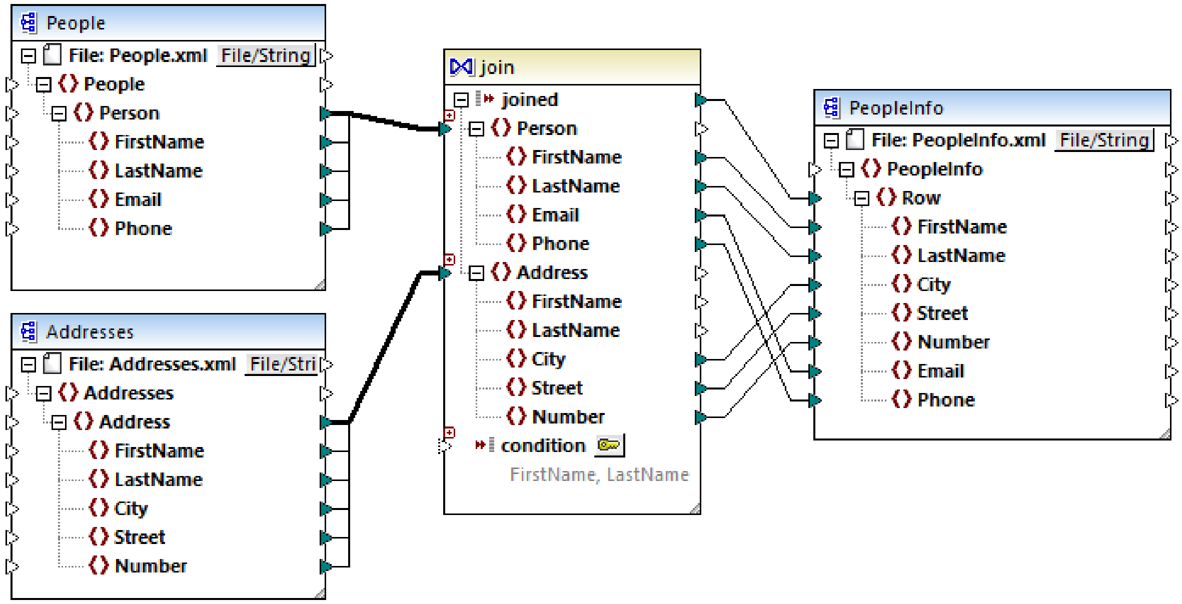
In some mappings, a condition consisting of one comparison may be sufficient to perform the join. However, in this example, it is important that two comparisons are created:

1) `FirstName` in Structure1 = `FirstName` in Structure 2
2) `LastName` in Structure 1 = `LastName` in Structure 2.

When multiple conditions are defined, *all of them must be true in order for the join to take place*. Therefore, in this example, a join will happen only when both comparisons are true (which is the intended behaviour). Otherwise, if only one of the comparisons above were defined, a join could happen for persons that have the same first name but different last names.

Step 4: Map the Join component to the target schema

Now that the two structures are joined, you can define which items of the joined structure should be mapped to the target. To do this, create connections from items of both joined structures to the target component, as shown below. The connection between `joined` and `Row` has the following purpose: whenever the join condition is satisfied, it creates a new `Row` item in the target.



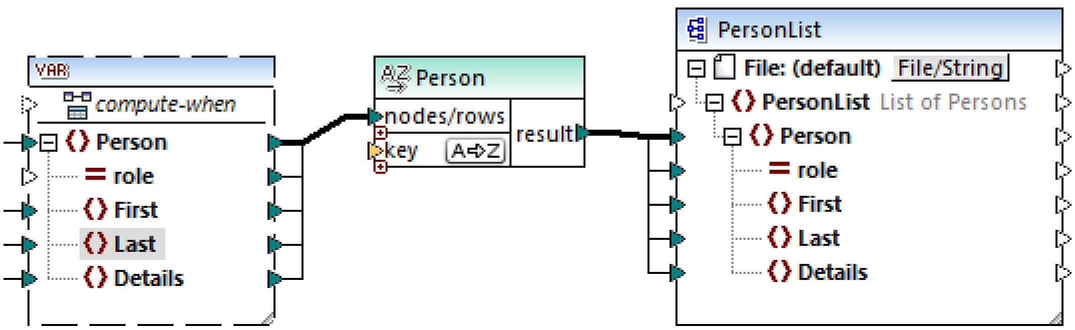
To preview the mapping output, click the **Output** tab. As expected, each person record (<Row>) now includes the full address details, joined from two different sources.

5.5 Sort Components

To sort input data based on a specific sort key, use a Sort component. The Sort component supports the following target languages: XSLT2, XQuery, and Built-in. When the transformation language is "Built-in", the Sort component can be used to sort database table data. Better performance is, however, achieved using an SQL-WHERE/ORDER component. For more details, see Filtering and Sorting Database Data (SQL WHERE/ORDER) 350.

To add a sort component to the mapping, do one of the following:

- Right-click an existing connection, and select **Insert Sort: Nodes/Rows** from the context menu. This inserts the Sort component and automatically connects it to the source and target components. For example, in the mapping below, the Sort component was inserted between a variable and an XML component. The only thing that remains to be connected manually is the sorting key (the field by which you want to sort).

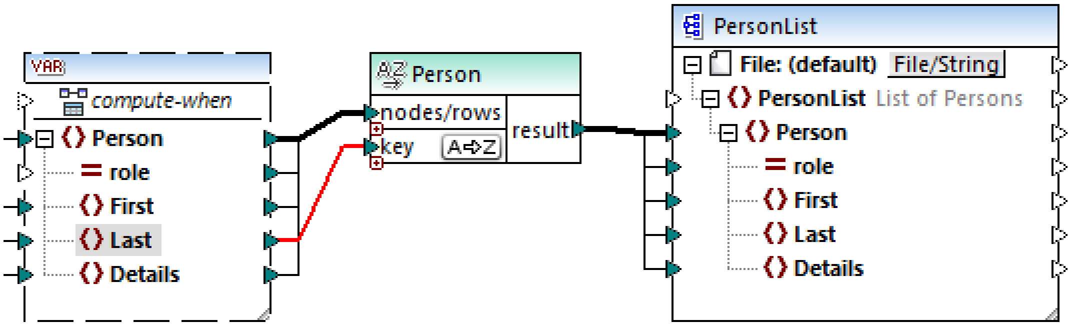


- On the **Insert** menu, click **Sort** (alternatively, click the **Sort**  toolbar button). This inserts the Sort component in its "unconnected" form.

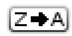
As soon as a connection is made to the source component, the title bar name changes to that of the item connected to the `nodes/rows` item.

To define the item by which you want to sort:

- Connect the item by which you want to sort to the `key` parameter of the Sort component. For example, in the mapping below, the `Person` nodes/rows are sorted by the field `Last`.

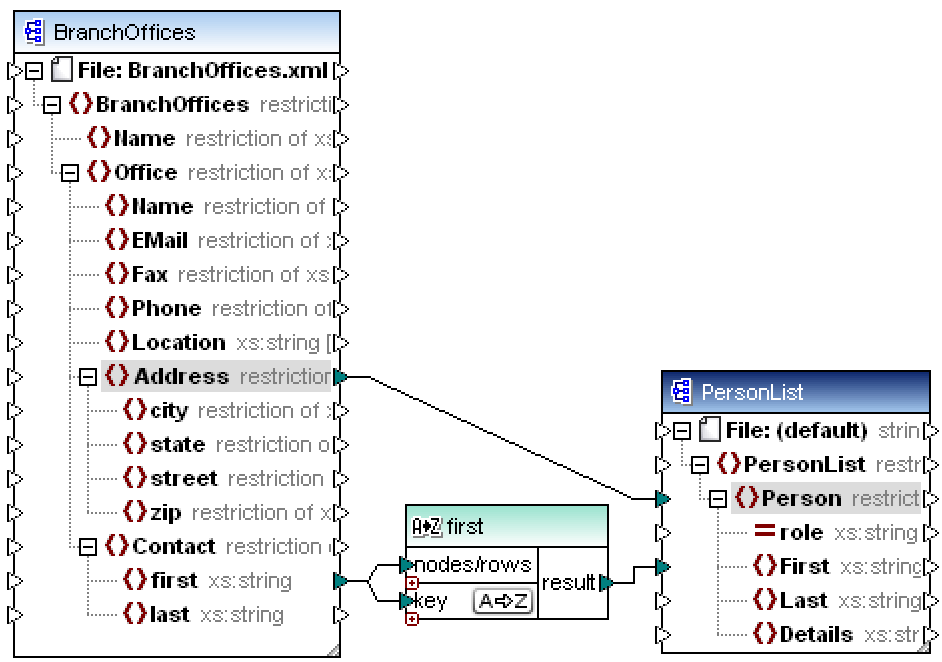


To change the sort order:

- Click the  icon in the Sort component. It changes to  to show that the sort order has been reversed.

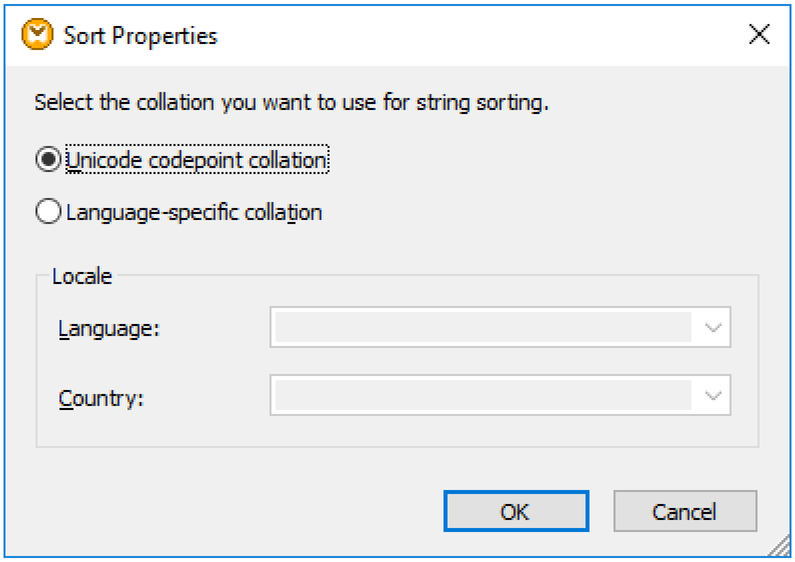
To sort input data consisting of simple type items:

- Connect the item to both the `nodes/rows` and `key` parameters of the sort component. In the mapping below, the element of simple type `first` is being sorted.



To sort strings using language-specific rules:

- Double-click the header of the Sort component to open the Sort Properties dialog box.



Unicode codepoint collation: This (default) option compares/orders strings based on code point values. Code point values are integers that have been assigned to abstract characters in the Universal Character Set adopted by the Unicode Consortium. This option allows sorting across many languages and scripts.

Language-specific collation: This option allows you to define the specific language and country variant you want to sort by. This option is supported when using the BUILT-IN execution engine. For XSLT, support depends on the specific engine used to execute the code.

5.5.1 Sorting by Multiple Keys

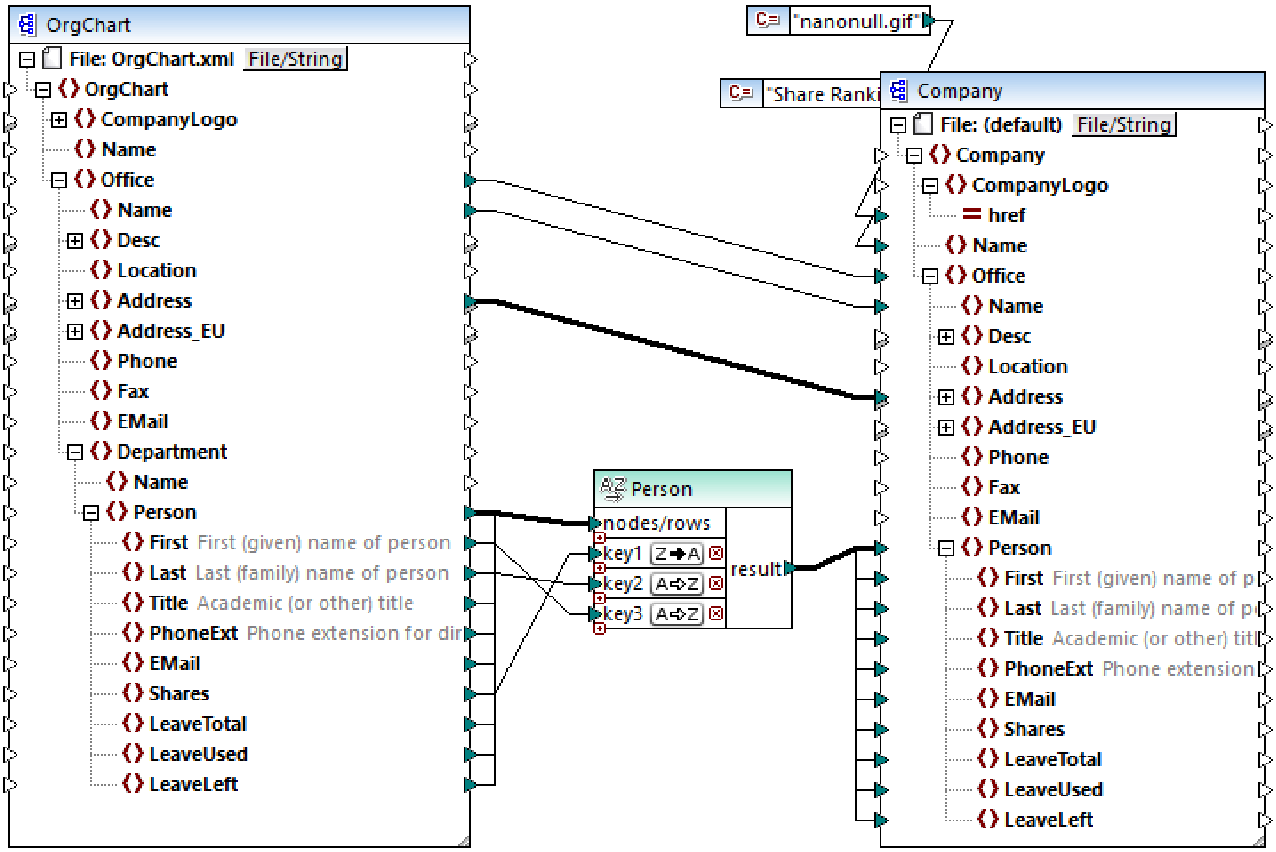
After you add a Sort component to the mapping, one sorting key called `key` is created by default.

Default Sort component

If you want to sort by multiple keys, adjust the Sort component as follows:

- Click the **Add Key** ( ) icon to add a new key (for example, `key2` in the mapping below).
- Click the **Delete Key** ( ) icon to delete a key.
- Drop a connection onto the icon to add a key and also connect to it.

A mapping which illustrates sorting by multiple key is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\SortByMultipleKeys.mfd**.



SortByMultipleKeys.mfd

In the mapping above, `Person` records are sorted by three sorting keys:

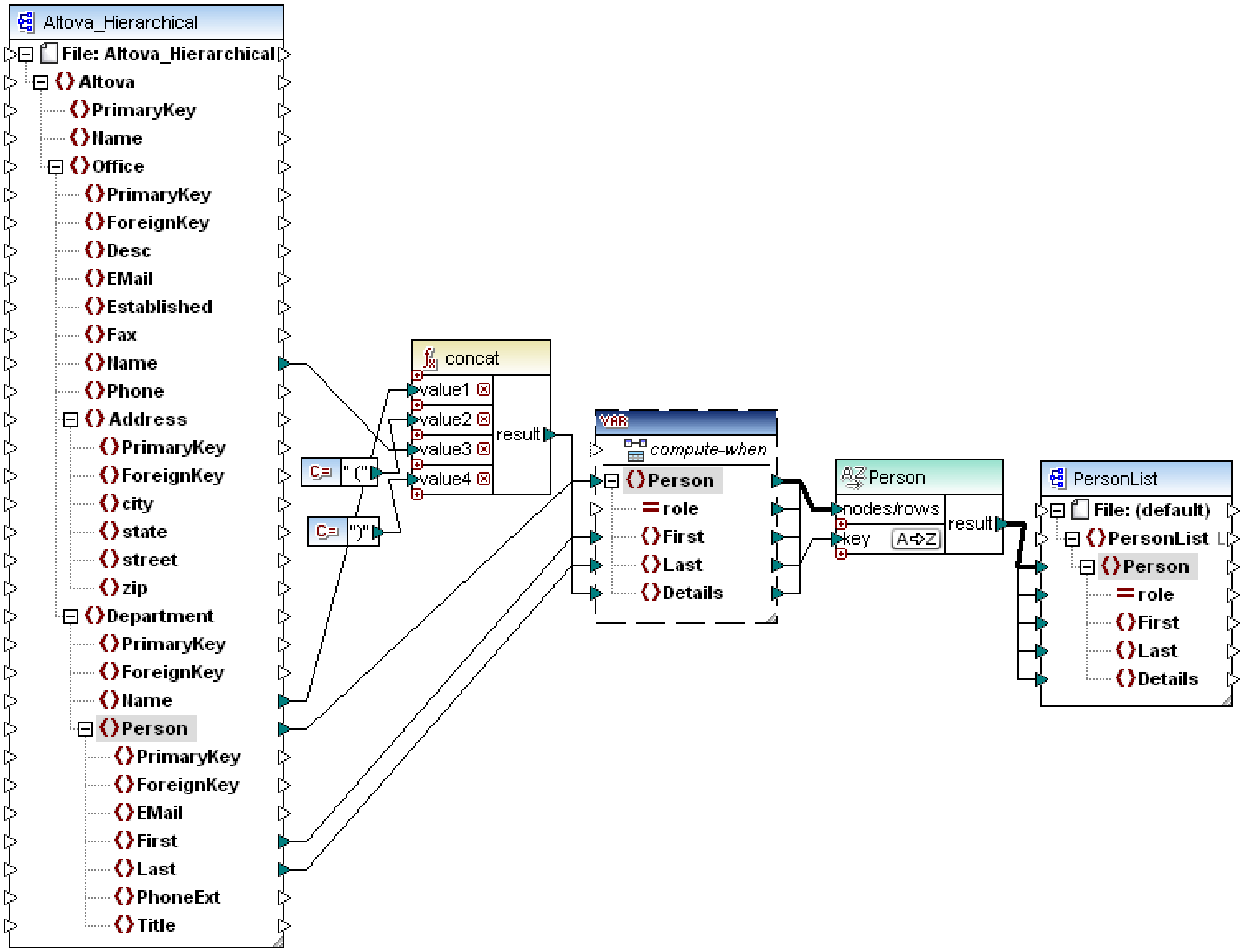
1. `Shares` (number of shares a person holds)
2. `Last` (last name)
3. `First` (first name)

Note that the position of the sorting key in the Sort component determines its sort priority. For example, in the mapping above, records are initially sorted by the number of shares. This is the sorting key with the highest priority. If the number of shares is the same, people are then sorted by their last name. Finally, when multiple people have the same number of shares and the same last name, the person's first name is taken into account.

The sort order of each key can be different. In the mapping above, the key `Shares` has a descending sort order (Z-A), while the other two keys have ascending sort order (A-Z).

5.5.2 Sorting with Variables

In some cases, it may be necessary to add intermediate variables to the mapping in order to achieve the desired result. This example illustrates how to extract records from an XML file, and sort them, with the help of intermediate variables. The example is accompanied by a mapping sample located at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Altova_Hierarchical_Sort.mfd**.



Altova_Hierarchical_Sort.mfd

This mapping reads data from a source XML file called **Altova_Hierarchical.xml** and writes it to a target XML file. As shown above, the source XML contains information about a fictitious company. The company is divided into offices. Offices are sub-divided into departments, and departments are further divided into people.

The target XML component, `PersonList`, contains a list of `Person` records. The `Details` item is meant to store information about the office and department where the person belongs.

The aim is to extract all persons from the source XML and sort them alphabetically by last name. Also, the office and department name where each person belongs must be written to the `Details` item.

To achieve this goal, this example makes use of the following component types:

1. The **concat** function. In this mapping, this function returns a string in the format `Office(Department)`. It takes as input the office name, the department name, and two constants which supply the start and end brackets. See also Add a Function to the Mapping 905.
2. An intermediate variable. The role of the variable is to bring all data relevant to a person into the same mapping context. The variable causes the mapping to look up the department and office of each person, in the context of each person. To put it differently, the variable "remembers" the office and

department name to which a person belongs. Without the variable, the context would be incorrect, and the mapping would produce unwanted output (for more information about how a mapping is executed, see Mapping Rules and Strategies[1275]). Notice that the variable replicates the structure of the target XML file (it uses the same XML schema). This makes it possible to connect the sort result to the target, through a Copy-All connection. See also Using Variables[847] and Copy-All Connections[97].

3. A Sort component, which performs the actual sorting. Notice that the key input of the `Sort` component is connected to the `Last` item of the variable, which sorts all person records by their last name.

5.6 Filters and Conditions

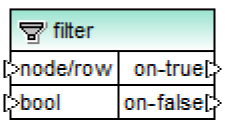
When you need to filter data, or get a value conditionally, you can use one of the following component types:

- Filter: Nodes/Rows ( )
- SQL WHERE/ORDER ( )
- If-Else Condition ( )

You can add these components to the mapping either from the **Insert** menu, or from the **Insert Component** toolbar. Importantly, each of the components above has specific behavior and requirements. The differences are explained in the following sections.

Filtering nodes or rows

When you need to filter data, including XML nodes or CSV rows, use a **Filter Nodes/Rows** component. The **Filter Nodes/Rows** component enables you to retrieve a subset of nodes from a larger set of data, based on a true or false condition. Its structure on the mapping area reflects this:



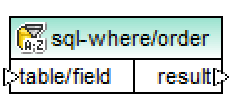
In the structure above, the condition connected to **bool** determines whether the connected **node/row** goes to the **on-true** or **on-false** output. Namely, if the condition is true, the **node/row** will be redirected to the **on-true** output. Conversely, if the condition is false, the **node/row** will be redirected to the **on-false** output.

When your mapping needs to consume only items that *meet* the filter condition, you can leave the **on-false** output unconnected. If you need to process the items that *do not meet* the filter condition, connect the **on-false** output to a target where such items should be redirected. If you want to add an exception when the filter condition is not met, connecting the **on-false** output is mandatory (see Adding Exceptions [900]).

For a step-by-step mapping example, see Example: Filtering Nodes [880].

Filtering database data

Filter Nodes/Rows components can filter data from any other component structure supported by MapForce, including databases. However, if you want to filter data from a database, it is recommended to use a **SQL WHERE/ORDER** component instead. The **SQL WHERE/ORDER** component is optimized for working with databases and provides better performance than a **Filter Nodes/Rows** component.



For more information about such components, see SQL WHERE / ORDER Component [350].

Returning a value conditionally

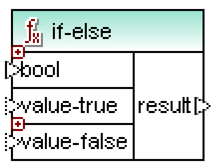
If you need to get a single value (not a node or row) conditionally, use an **If-Else Condition**. Note that If-Else conditions are not suitable for filtering nodes or rows. Unlike **Filter Nodes/Rows** components, an **If-Else**

Condition returns a value of simple type (such as a string or integer). Therefore, **If-Else Conditions** are only suitable for scenarios where you need to process a simple value conditionally. For example, let's assume you have a list of average temperatures per month, in the format:

```
<Temperatures>
   <data temp="19.2" month="2010-06" />
   <data temp="22.3" month="2010-07" />
   <data temp="19.5" month="2010-08" />
   <data temp="14.2" month="2010-09" />
   <data temp="7.8" month="2010-10" />
   <data temp="6.9" month="2010-11" />
   <data temp="-1.0" month="2010-12" />
</Temperatures>
```

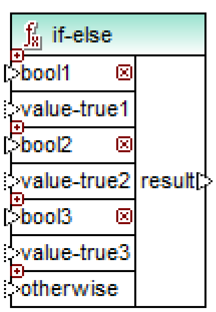
An **If-Else Condition** would enable you to return, for each item in the list, the value "high" if temperature exceeds 20 degrees Celsius, and value "low" if temperature is lower than 5 degrees Celsius.

On the mapping, the structure of the **If-Else Condition** looks as follows:



If the condition connected to **bool** is true, then the value connected to **value-true** is output as **result**. If the condition is false, the value connected to **value-false** is output as **result**. The data type of **result** is not known in advance; it depends on the data type of the value connected to **value-true** or **value-false**. The important thing is that it should always be a simple type (string, integer, and so on). Connecting input values of complex type (such as nodes or rows) is not supported by **If-Else Conditions**.

If-Else Conditions are extendable. This means that you can add multiple conditions to the component, by clicking the **Add** (⊞) button. To delete a previously added condition, click the **Delete** (⊠) the button. This feature enables you to check for multiple conditions and return a different value for each condition, if it is true.



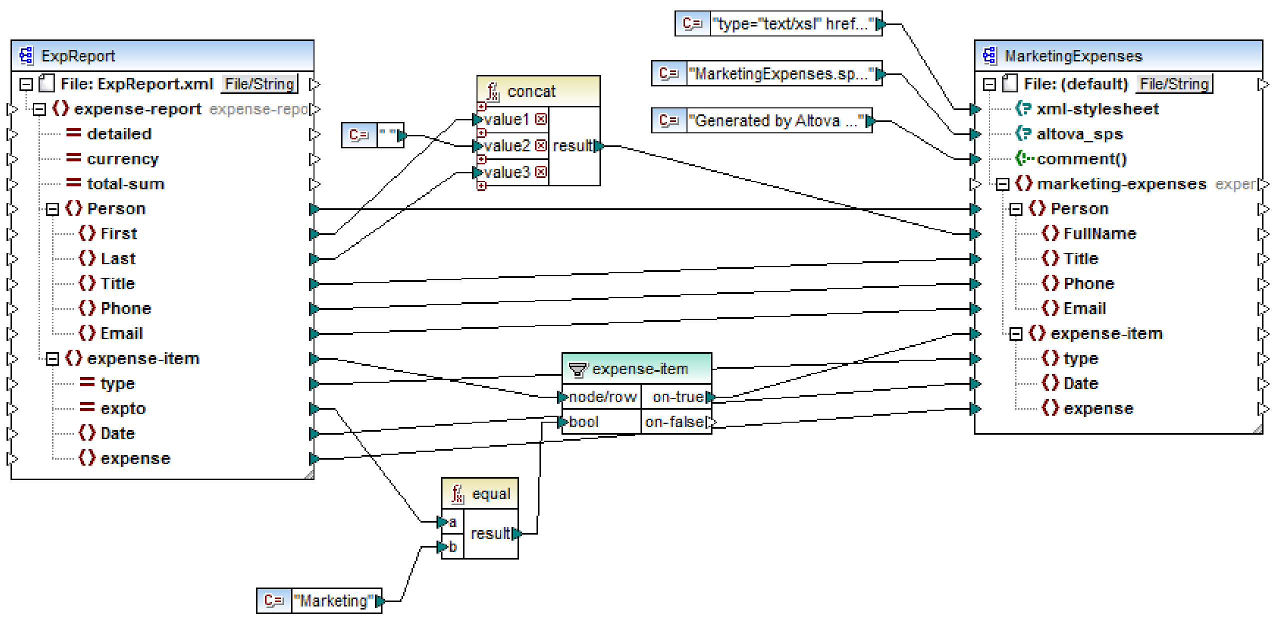
Expanded **If-Else Conditions** are evaluated from top to bottom (first conditions is checked first, then the second one, and so on). If you want to return a value when none of the conditions are true, connect it to **otherwise**.

For a step-by-step mapping example, see Example: Returning a Value Conditionally 882.

5.6.1 Example: Filtering Nodes

This example shows you how to filter nodes based on a true/false condition. A **Filter: Nodes/Rows** ( ) component is used to achieve this goal. The technique illustrated in this example works in the same way not only for XML, but also for other component types, such as CSV or text. In case of databases, although you can use a filter, it is recommended to use a SQL WHERE/ORDER component instead, for better performance (see SQL WHERE / ORDER Component 350).

The mapping described in this example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\MarketingExpenses.mfd**.



As shown above, the mapping reads data from a source XML which contains an expense report ("ExpReport") and writes data to a target XML ("MarketingExpenses"). There are several other components between the target and source. The most relevant component is the **expense-item** filter ( ), which represents the subject of this topic.

The goal of the mapping is to filter out only those expense items that belong to the Marketing department. To achieve this goal, a filter component has been added to the mapping. (To add a filter, click the **Insert** menu, and then click **Filter: Nodes/Rows**.)

To identify whether each expense item belongs to Marketing, this mapping looks at the value of the "expto" attribute in the source. This attribute has the value "Marketing" whenever the expense is a marketing expense. For example, in the code listing below, the first and third expense item belongs to Marketing, the second belongs to Development, and the fourth belongs to Sales:

```
...
   <expense-item type="Meal" expto="Marketing">
      <Date>2003-01-01</Date>
```



```
      <expense>122.11</expense>
   </expense-item>
   <expense-item type="Lodging" expto="Development">
      <Date>2003-01-02</Date>
      <expense>122.12</expense>
   </expense-item>
   <expense-item type="Lodging" expto="Marketing">
      <Date>2003-01-02</Date>
      <expense>299.45</expense>
   </expense-item>
   <expense-item type="Entertainment" expto="Sales">
      <Date>2003-01-02</Date>
      <expense>13.22</expense>
   </expense-item>
...
```

XML input before the mapping is executed

On the mapping area, the **node/row** input of the filter is connected to the **expense-item** node in the source component. This ensures that the filter component gets the list of nodes that it must process.

To add the condition based on which filtering should occur, we have added the **equal** function from the MapForce core library, see also Add a Function to the Mapping[905]. The **equal** function compares the value of the `expto` attribute to a constant which has the value `Marketing`. (To add a constant, click the **Insert** menu, and then click **Constant**.)

Since we need to filter only those items that satisfy the condition, we connected only the **on-true** output of the filter to the target component.

When you preview the mapping result, by clicking the **Output** tab, MapForce evaluates, for each expense-item node, the condition connected to the **bool** input of the filter. When the condition is true, the expense-item node is passed on to the target; otherwise, it is ignored. Consequently, only the expense items matching the criteria are displayed in the output:

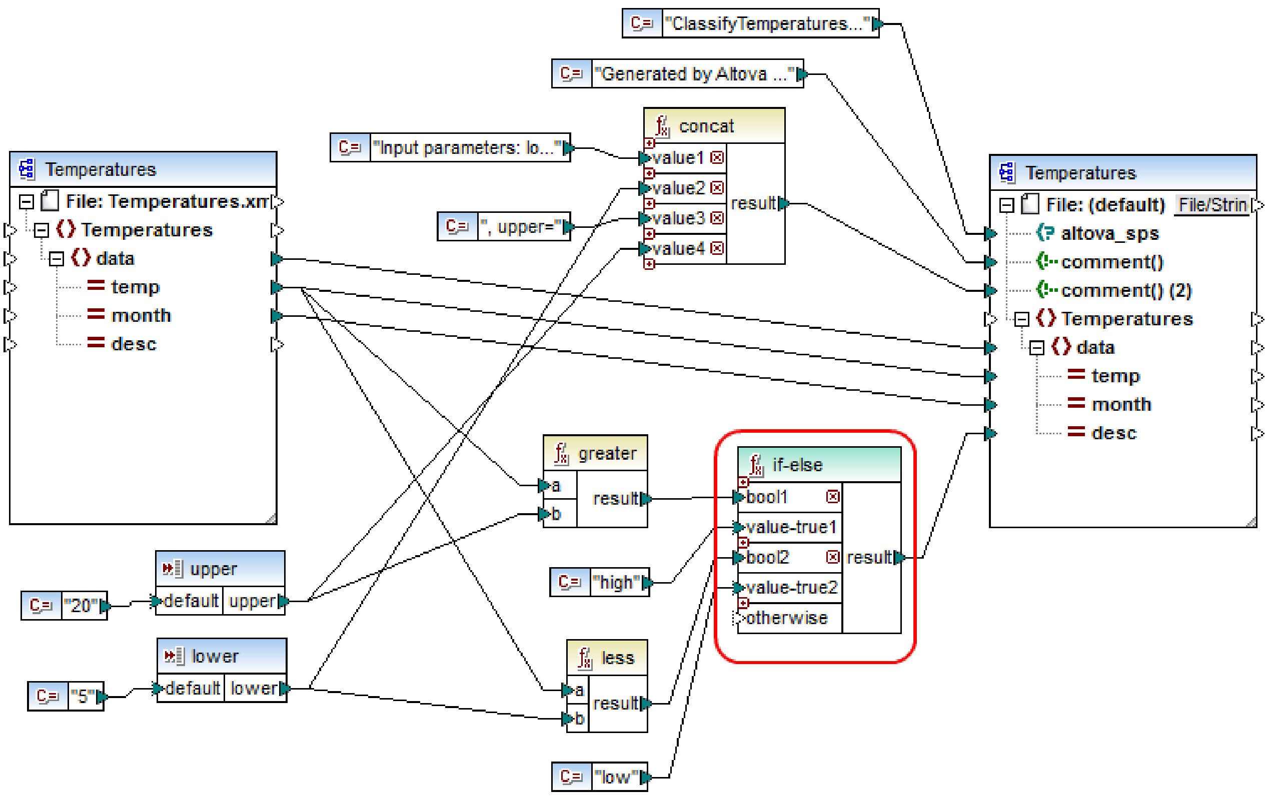
```
...
   <expense-item>
      <type>Meal</type>
      <Date>2003-01-01</Date>
      <expense>122.11</expense>
   </expense-item>
   <expense-item>
      <type>Lodging</type>
      <Date>2003-01-02</Date>
      <expense>299.45</expense>
   </expense-item>
...
```

XML output after the mapping is executed

5.6.2 Example: Returning a Value Conditionally

This example shows you how to return a simple value from a component, based on a true/false condition. An **If-Else Condition** ( ) is used to achieve the goal. Note that **If-Else Conditions** should not be confused with filter components. **If-Else Conditions** are only suitable when you need to process simple values conditionally (string, integer, etc.). If you need to filter complex values such as nodes, use a filter instead (see Example: Filtering Nodes 880).

The mapping described in this example is available at the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\ClassifyTemperatures.mfd.



This mapping reads data from a source XML which contains temperature data ("Temperatures") and writes data to a target XML which conforms to the same schema. There are several other components between the target and source, one of them being the **if-else** condition (highlighted in red), which is also the subject of this topic.

The goal of the mapping is to add short description to each temperature record in the target. Specifically, if temperature is above 20 degrees Celsius, the description should be "high". If the temperature is below 5 degrees Celsius, the description should be "low". For all other cases, no description should be written.

To achieve this goal, conditional processing is required; therefore, an If-Else Condition has been added to the mapping. (To add an If-Else Condition, click the **Insert** menu, and then click **If-Else Condition**.) In this mapping, the If-Else Condition has been extended (with the help of the button) to accept two conditions: **bool1** and **bool2**.

The conditions themselves are supplied by the **greater** and **less** functions, which have been added from the MapForce core library, see also Add a Function to the Mapping905. These functions evaluate the values provided by two input components, called "upper" and "lower". (To add an input component, click the **Insert** menu, and then click **Insert Input**. For more information about input components, see Supplying Parameters to the Mapping833.)

The **greater** and **less** functions return either true or false. The function result determines what is written to the target instance. Namely, if the value of the "temp" attribute in the source is greater than 20, the constant value "high" is passed to the **if-else** condition. If the value of the "temp" attribute in the source is less than 5, the constant value "low" is passed on to the **if-else** condition. The **otherwise** input is not connected. Therefore, if none of the above conditions is met, nothing is passed to the **result** output connector.

Finally, the **result** output connector supplies this value (once for each temperature record) to the "desc" attribute in the target.

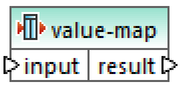
When you are ready to preview the mapping result, click the **Output** tab. Notice that the resulting XML output now includes the "desc" attribute, whenever the temperature is either greater than 20 or lower than 5.

```
...
   <data temp="-3.6" month="2006-01" desc="low"/>
   <data temp="-0.7" month="2006-02" desc="low"/>
   <data temp="7.5" month="2006-03"/>
   <data temp="12.4" month="2006-04"/>
   <data temp="16.2" month="2006-05"/>
   <data temp="19" month="2006-06"/>
   <data temp="22.7" month="2006-07" desc="high"/>
   <data temp="23.2" month="2006-08" desc="high"/>
...
```

XML output after the mapping is executed

5.7 Value-Maps

The Value-Map component enables you to replace a value by another value with the help of a predefined look-up table. Such a component processes only one value at a time; therefore, it has one **input** and one **result** on the mapping.



A Value-Map is very useful when you would like to map individual items within two sets in order to replace items. For example, you could map the days of the week expressed as numbers (1, 2, 3, 4, 5, 6, and 7) to the name of each day of the week ("Monday", "Tuesday", and so on). Likewise, you could map the month names ("January", "February", "March", etc) to the numeric representation of each month (1, 2, 3, etc). At mapping run time, the matching values will be replaced according to your custom look-up table. The values in both sets can be of different type, but each set must store values of the same data type.

Value-Map components are suitable for simple look-ups, where each value in the first set corresponds to a single value in the second set. If a value is not found in the look-up table, you can either replace it with a custom value or an empty value, or pass it on as is. If you need to look up or filter values based on more complex criteria, use one of the filtering components[878] instead.

Importantly, when you generate code or compile a MapForce Server Execution file from the mapping, the look-up table data is embedded into the generated code or file. Consequently, defining a look-up table directly on the mapping is a good choice only if your data does not change frequently and is not very big (less than maybe a few hundred entries). If the look-up data changes regularly, you may find it difficult to maintain both the mapping and the generated code regularly—it is easier to maintain the look-up data as text, XML, database, or perhaps Excel.

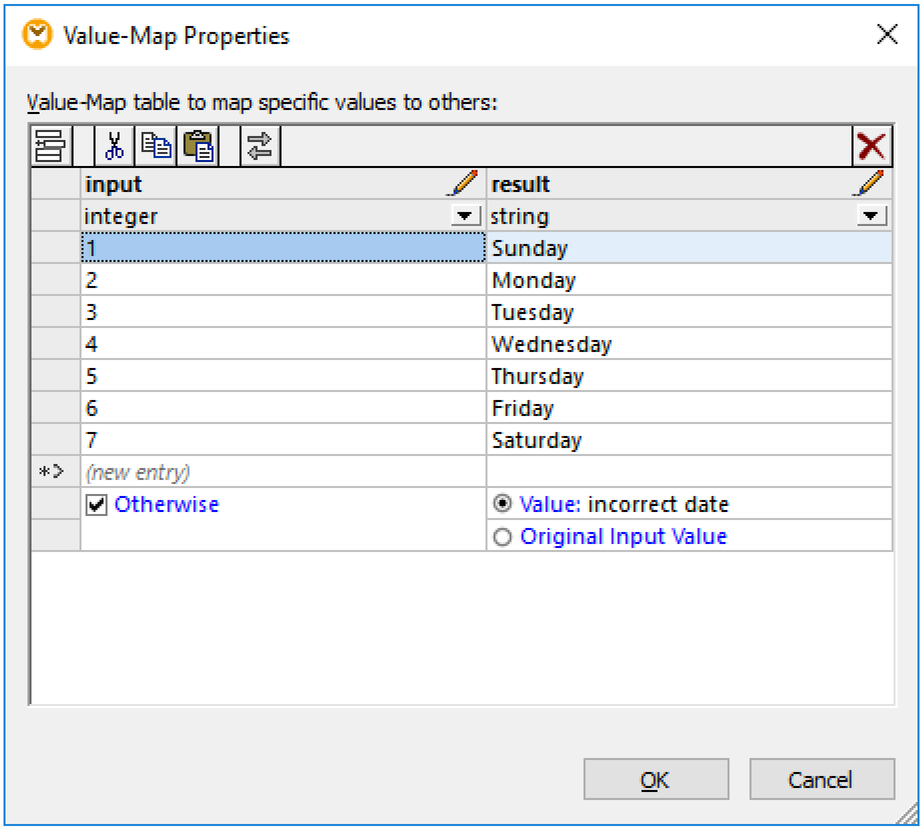
If the look-up table is huge, the mapping execution will be slowed down by the look-up table. In this case, it is recommended to use a database component with SQL-Where[350] instead. SQLite databases are good candidates for this, given their portability. On the server side, you can improve the performance of look-up tables by running a mapping with MapForce Server or MapForce Server Advanced Edition.

Creating Value-Maps

To add a Value-Map component to the mapping, do one of the following:

- Click the **Insert Value-Map**  toolbar button.
- On the **Insert** menu, click **Value-Map**.
- Right-click a connection, and select **Insert Value-Map** from the context menu.

This adds a new Value-Map component to the mapping. You can now start adding pairs of items to the look-up table. To do this, double-click the component's title bar or right-click it and select **Properties** from the context menu.



At mapping run time, MapForce checks each value that reaches the **input** of the Value-Map. If there is a matching value *in the left column* of the look-up table, then it replaces the original input value with the value *from the right column*. Otherwise, you can optionally configure it to return one of the following:

- A replacement value. In the example above, the replacement value is the text "incorrect date". You can also set the replacement value to be empty, by not entering any text at all.
- The original input value. This means that, if no match is found in the look-up table, the original input value will be passed further on to the mapping, unchanged.

> If you do not configure an "Otherwise" condition, the Value-Map returns an **empty node** whenever a match is not found. In this case, nothing will be passed to the target component and the output will contain missing fields. To prevent this from happening, you should either configure the "Otherwise" condition, or use the **`substitute-missing`** 1056 function.

There is a difference between setting an empty replacement value and not specifying the "Otherwise" condition. In the first case, the field will be generated in the output, but it will have an empty value. In the latter case, the field (or XML element) enclosing the value will not be created at all. For more information, see Example: Replacing Job Titles 890.

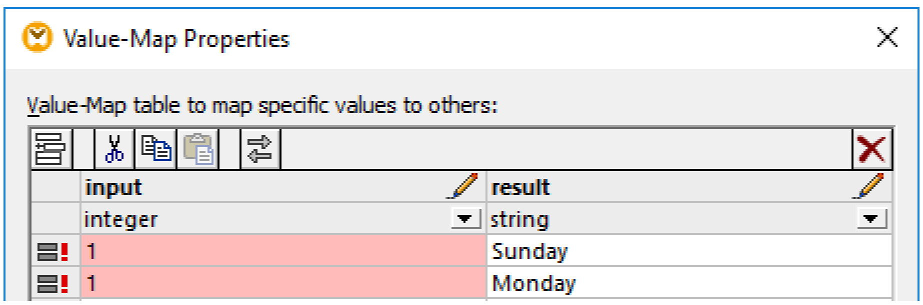
Populating a Value-Map

In a look-up table, you can define as many pairs of values as needed. You can enter the values manually, or copy-paste tabular data from text, CSV, or Excel files. Copy-pasting tables from an HTML page using a common browser will also work in most cases. You can also paste data from the database grid in the DB Query pane 376. If you copy data from text files, the fields must be separated by tab characters. In addition, MapForce will recognize text separated by commas or semicolons in most cases.

Keep in mind the following when creating look-up tables:

1. All items in the left column must be unique. Otherwise, it would not be possible to determine which item you want to match specifically.
2. Items that belong to the same column must be of the same data type. You can choose the data type from the drop-down list at the top of each column in the look-up table. If you need to convert Boolean types, enter the text "true" or "false" literally. For an illustration of this case, see Example: Replacing Weekdays 888.

If MapForce encounters invalid data in the look-up table, it displays an error message and highlights the invalid rows in pink color, for example:



To import data from an external source into the Value-Map component:

1. Select the cells of interest in the source program (for example, Excel). This can be either a single column of data or two adjacent columns.
2. Copy data to clipboard using the **Copy** command of the external program.
3. On the Value-Map component, click the row before which you would like to paste the data .
4. Click the **Paste table from clipboard** button on the Value-Map component. Alternatively, press **Ctrl+V** or **Shift+Insert**.

Note: The **Paste table from clipboard** button is enabled only if you have copied data from some source first (that is, if there is data on the clipboard).

When your clipboard data contains multiple columns, then only data from the first two columns are inserted into the look-up table; any other subsequent columns will be ignored. If you paste data from a single column on top of any existing values, a context menu appears, asking whether the clipboard data should be inserted as new rows or the existing rows should be overwritten. Therefore, if you need to overwrite existing values in the look-up table as opposed to inserting new rows, ensure that the clipboard contains only one column, not multiple.

To insert rows manually before an existing row, first click the row of interest, and then click the **Insert** button.

To move an existing row to some other position, drag the row to the new position (upwards or downwards) while holding the left mouse button pressed.

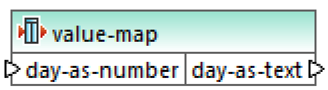
To copy or cut rows for subsequent pasting at some other position, first select the row, and then click the **Copy** button (or **Cut** button, respectively). You can also copy or cut multiple rows that are not necessarily consecutive. To select multiple rows, hold the **Ctrl** key pressed while clicking the rows. Note that the cut or copied text always contains values from both columns; you cannot cut or copy values from one column only.

To remove a row, click it, and then click the **Delete** button.

To swap the left and right columns, click the **Swap** button.

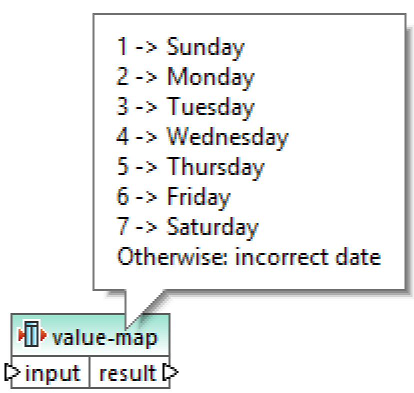
Renaming Value-Map parameters

By default, the input parameter of a Value-Map component is called "input" and the output parameter is called "result". To make the mapping clearer, you can optionally rename any of these parameters by clicking the **Edit** button next to the respective name. The following is an example of a Value-Map with custom parameter names:



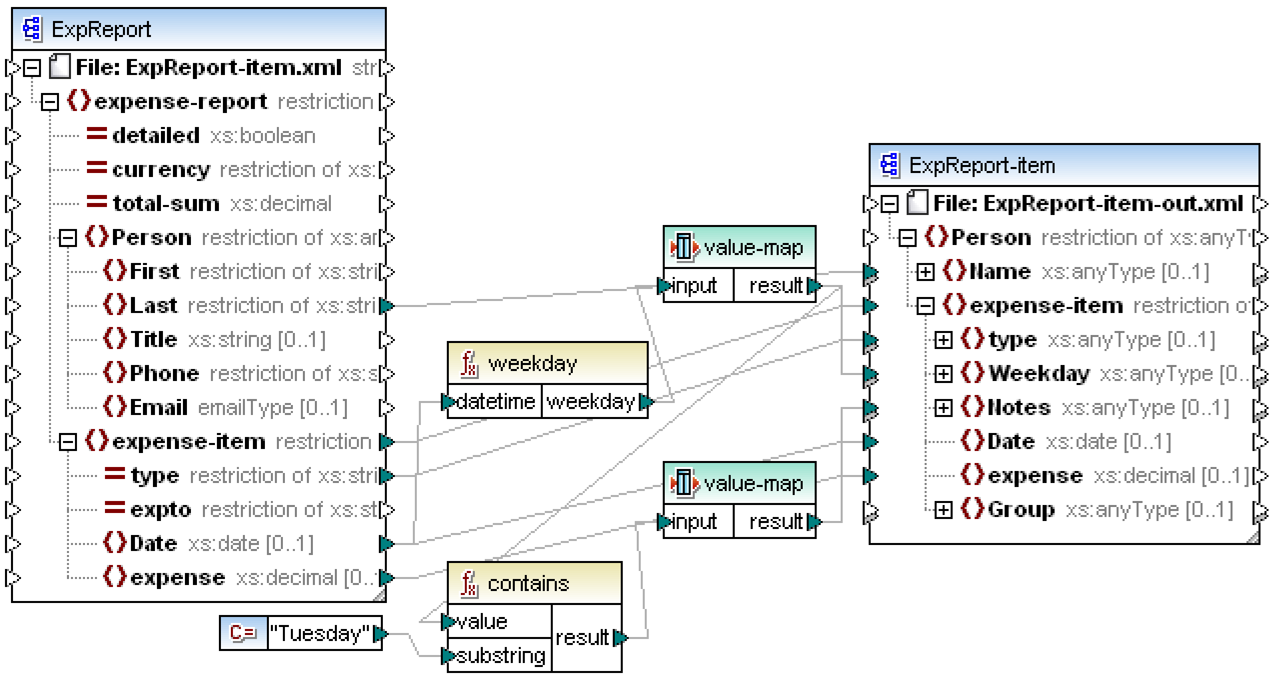
Previewing a Value-Map

After you have finished creating a Value-Map, you can quickly preview its implementation directly from the mapping by holding the mouse over the component's title bar:



5.7.1 Example: Replacing Weekdays

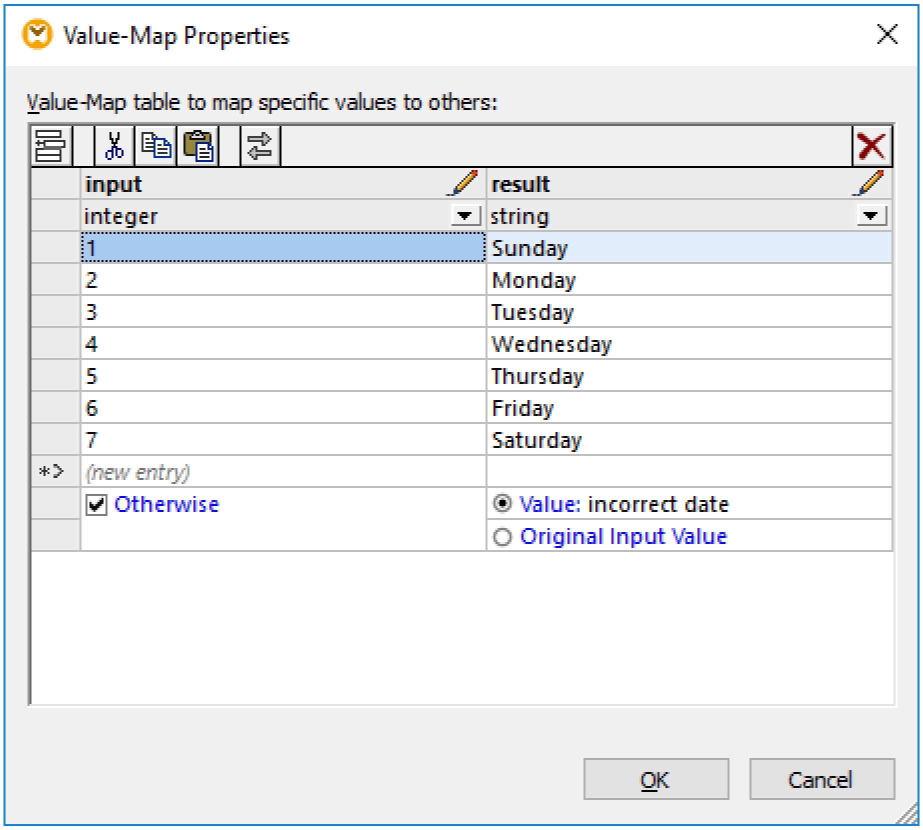
This example illustrates a Value-Map that replaces integer values with weekday names (1 = Sunday, 2 = Monday, and so on). This example is accompanied by a mapping which is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Expense-valmap.mfd**.



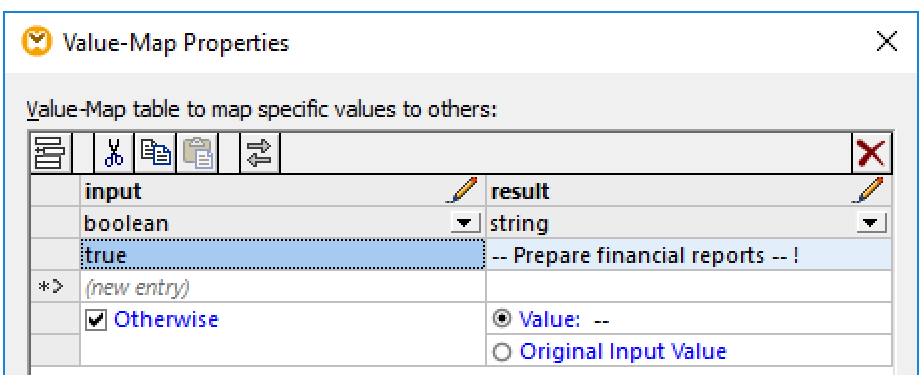
Expense-valmap.mfd

This mapping extracts the day of the week from the **Date** item in the source file, converts the numerical value into text, and writes it to the **Weekday** item of the target component. More specifically, the following happens:

- The **weekday** function extracts the weekday number from the **Date** item in the source file. The result of this function are integers ranging from 1 to 7.
- The first Value-Map component transforms the integers into weekdays (1 = Sunday, 2 = Monday, and so on). If the component encounters an invalid integer outside of the 1-7 range, then it will return the text "incorrect date".



- If the weekday contains "Tuesday", then the text "Prepare Financial Reports" is written to the **Notes** item in the target component. This is achieved with the help of the **`contains`** function, which passes a Boolean **true** or **false** value to a second Value-Map component. The second Value-Map has the following configuration:



The Value-Map illustrated above should be understood as follows:

- Whenever a Boolean **true** is encountered, convert it to the text "-- Prepare financial reports -- ! ". For all other cases, return the text "--".

Notice that the data type of the first column is set to "boolean". This ensures that the input Boolean value **true** is recognized as such.

5.7.2 Example: Replacing Job Titles

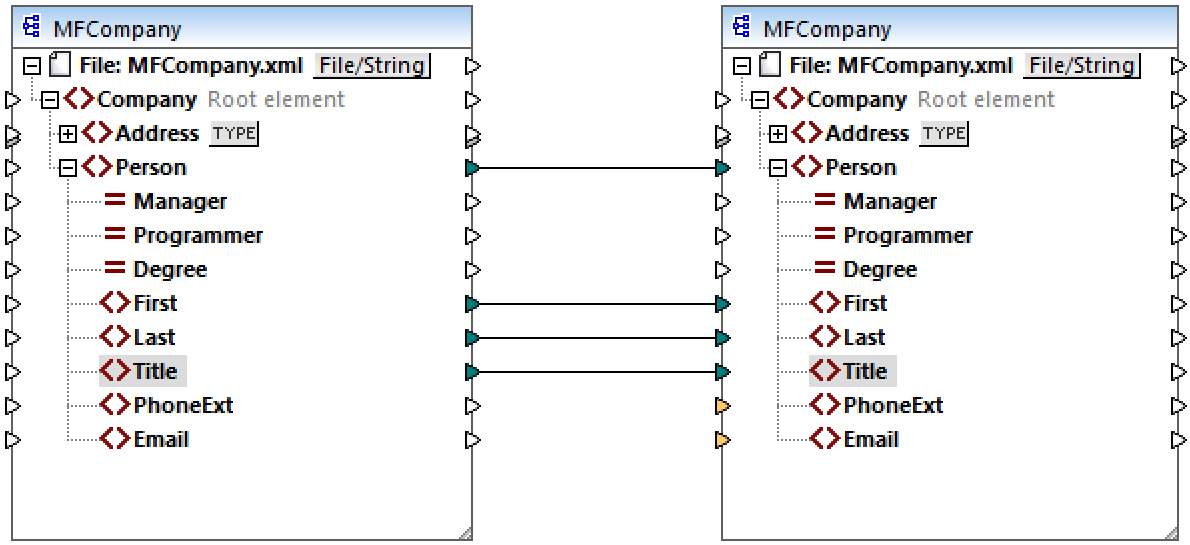
This example shows you how to replace values of specific elements in an XML file with the help of Value-Map components (that is, using a predefined look-up table).

The XML file required for this example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\MFCompany.xml**. It stores, among other data, information about company employees and their job titles, for example:

```
<Person>
   <First>Michelle</First>
   <Last>Butler</Last>
   <Title>Software Engineer</Title>
</Person>
<Person>
   <First>Lui</First>
   <Last>King</Last>
   <Title>Support Engineer</Title>
</Person>
<Person>
   <First>Steve</First>
   <Last>Meier</Last>
   <Title>Office Manager</Title>
</Person>
```

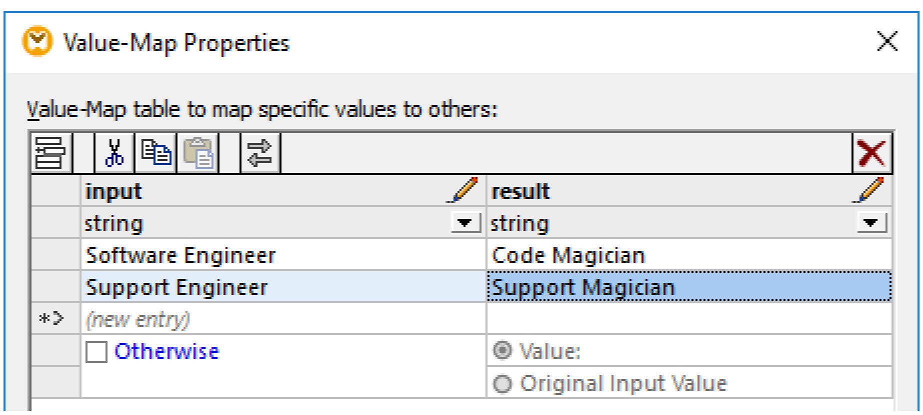
Let's assume that you need to replace some of the job titles in the XML file above. Specifically, the title "Software Engineer" must be replaced with "Code Magician". Also, the title "Support Engineer" must be replaced with "Support Magician". All the other job titles must remain unchanged.

To achieve the goal, add the XML file to the mapping area, by clicking the **Insert XML Schema/File** toolbar button or by running the **Insert | XML Schema/File** menu command. Next, copy-paste the XML component on the mapping and create the connections as shown below. Note that you might need to turn off the **Toggle auto-connect of children** toolbar option first, in order to prevent unnecessary connections from being created automatically.



The mapping created so far simply copies the **Person** elements to the target XML file, without making any changes to the **First**, **Last**, and **Title** elements.

To replace the required job titles, let's add a Value-Map component. Right-click the connection between the two **Title** elements, and select **Insert Value-Map** from the context menu. Set up the Value-Map properties as shown below:

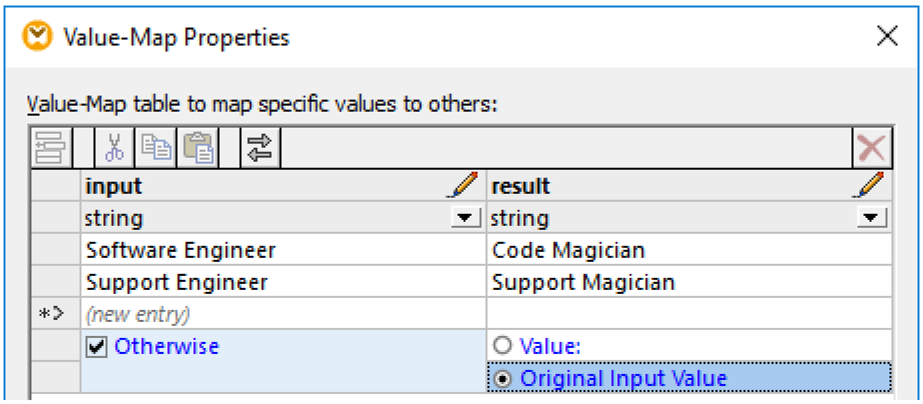


According to the setup above, each occurrence of "Software Engineer" will be replaced with "Code Magician", and each occurrence of "Support Engineer" will be replaced with "Support Magician". Notice that the **Otherwise** condition was not specified yet. For this reason, the Value-Map returns an *empty node* whenever the job title is other than "Software Engineer" and "Support Engineer". Consequently, if you click the **Output** tab and preview the mapping, some of the **Person** elements will have a missing a **Title**, for example:

```
<Person>
   <First>Vernon</First>
   <Last>Callaby</Last>
</Person>
<Person>
```

```
   <First>Frank</First>
   <Last>Further</Last>
</Person>
<Person>
   <First>Michelle</First>
   <Last>Butler</Last>
   <Title>Code Magician</Title>
</Person>
```

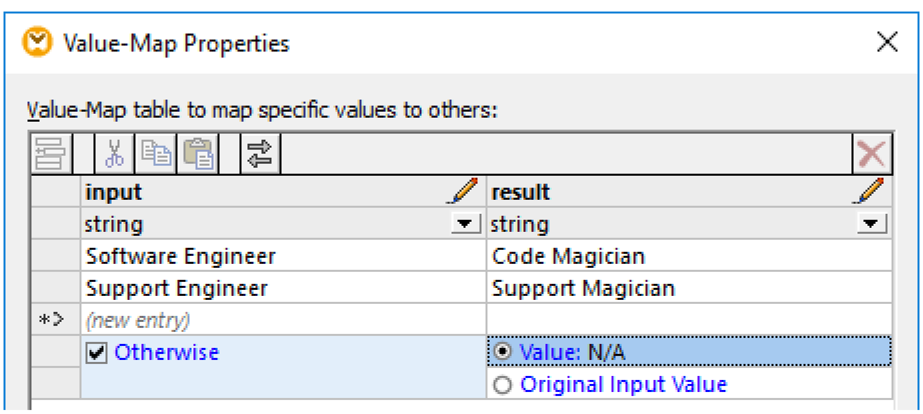
As stated before, empty nodes cause missing entries in the generated output; therefore, in the XML fragment above, only Michelle Butler had the title replaced, because her title was present in the look-up table. The configuration created so far still does not fulfill the original requirement. The correct setup is as follows:



With the configuration above, the following happens at mapping run time:

- Each occurrence of "Software Engineer" will be replaced with "Code Magician"
- Each occurrence of "Support Engineer" will be replaced with "Support Magician"
- If the original title is not found in the look-up table, the Value-Map will return it unchanged.

For illustrative purposes only, we can also change all the job titles other than "Software Engineer" and "Support Engineer" to a custom value, for example "N/A". To achieve this, set the Value-Map properties as shown below:



When you preview the mapping this time, each job title is present in the output, but those that were not matched have the "N/A" value, for example:

```
<Person>
   <First>Vernon</First>
   <Last>Callaby</Last>
   <Title>N/A</Title>
</Person>
<Person>
   <First>Frank</First>
   <Last>Further</Last>
   <Title>N/A</Title>
</Person>
<Person>
   <First>Michelle</First>
   <Last>Butler</Last>
   <Title>Code Magician</Title>
</Person>
```

This concludes the Value-Map example. By applying the logic above, you can now achieve the desired result in other mappings.

5.8 Group Functions

When your mapping must group nodes or rows, you can achieve this with the help of the following MapForce built-in functions:

- `group-by`
- `group-adjacent`
- `group-into-blocks`
- `group-starting-with`
- `group-ending-with`

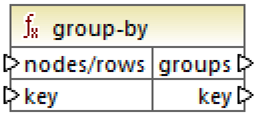
To use any of these functions on the mapping, drag them from the Libraries window onto the mapping area. See also Add a Function to the Mapping 905.

Note: Grouping functions are available in the following languages: XSLT 2.0, XSLT 3.0, C++, C#, Java, Built-In.

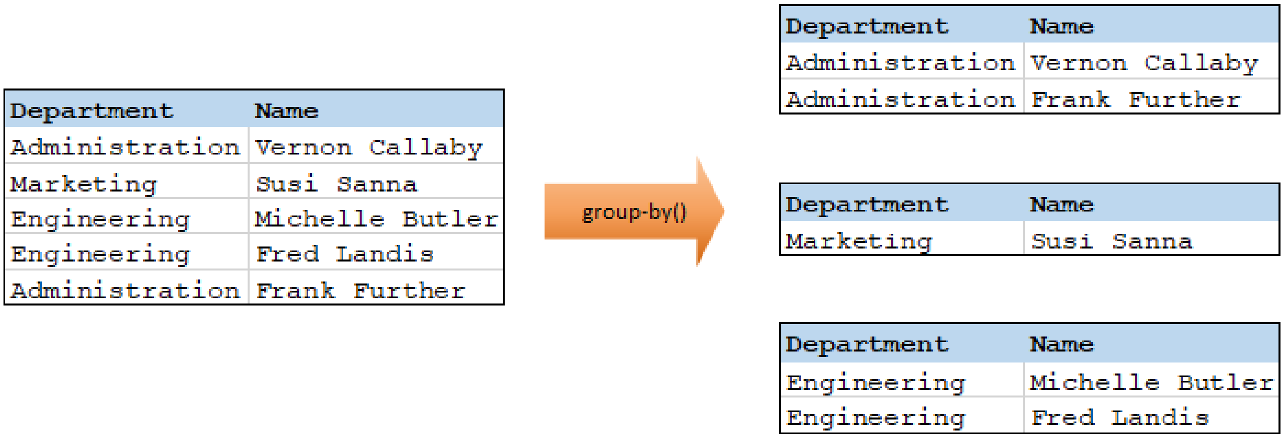
The following sections provide typical examples of use for grouping functions. These examples are accompanied by the following demo mapping: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Note that the demo mapping contains multiple transformations, one for each function. Since only one output can be previewed at a time, remember to click the **Preview** button applicable to the desired transformation before clicking the **Output** tab.

group-by

The **`group-by`** function creates groups of records according to some grouping key that you specify.



For example, in the abstract transformation illustrated below, the grouping key is "Department". Since there are three unique departments in total, applying the group-by function would create three groups:



Department	Name
Administration	Vernon Callaby
Marketing	Susi Sanna
Engineering	Michelle Butler
Engineering	Fred Landis
Administration	Frank Further

Department	Name
Administration	Vernon Callaby
Administration	Frank Further

Department	Name
Marketing	Susi Sanna

Department	Name
Engineering	Michelle Butler
Engineering	Fred Landis

For more information, see the reference to the group-by [1037] function.

group-adjacent

The **group-adjacent** function requires a grouping key as argument, similar to **group-by** function. Unlike **group-by**, this function creates a new group whenever the next key is different. If two adjacent records have the same key, they will be placed into the same group.

For example, in the abstract transformation illustrated below, the grouping key is "Department". The left side of the diagram shows the input data while the right side shows the output data after grouping. The following takes place when the transformation runs:

- Initially, the first key, "Administration", creates a new group.
- The next key is different, so a second group is created, "Marketing".
- The third key is also different, so another group is created, "Engineering".
- The fourth key is the same as the third; therefore, this record is placed in the already existing group.
- Finally, the fifth key is different from the fourth, and this creates the last group.

As illustrated below, "Michelle Butler" and "Fred Landis" were grouped together because they have the same key and are adjacent. However, "Vernon Callaby" and "Frank Further" are in separate groups because they are not adjacent, even though they have the same key.

Department	Name
Administration	Vernon Callaby
Marketing	Susi Sanna
Engineering	Michelle Butler
Engineering	Fred Landis
Administration	Frank Further

Department	Name
Administration	Vernon Callaby

Department	Name
Marketing	Susi Sanna

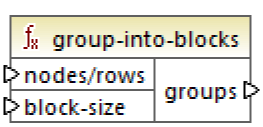
Department	Name
Engineering	Michelle Butler
Engineering	Fred Landis

Department	Name
Administration	Frank Further

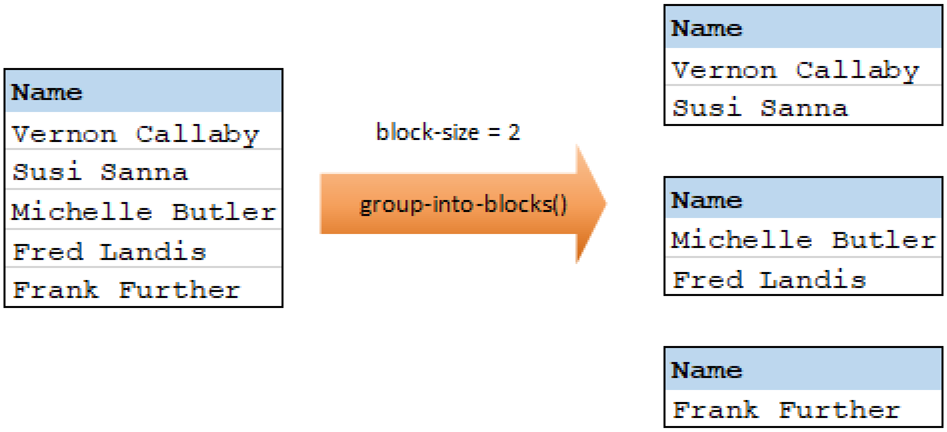
For more information, see the reference to the group-adjacent [1035] function.

group-into-blocks

The **group-into-blocks** function creates equal groups that contain exactly N items, where N is the value you supply to the `block-size` argument. Note that the last group may contain N items or less, depending on the number of items in the source.



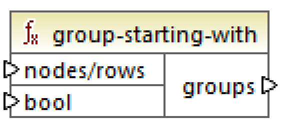
In the example below, `block-size` is 2. Since there are five items in total, each group contains exactly two items, except for the last one.



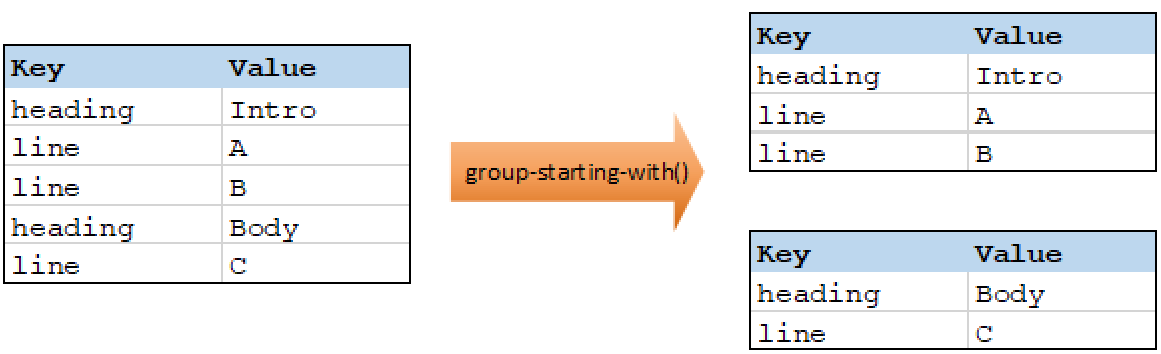
For more information, see the reference to the group-into-blocks 1041 function.

group-starting-with

The **`group-starting-with`** function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, starting with the record that satisfies the condition.



In the example below, the condition is that "Key" must be equal to "heading". This condition is true for the first and fourth records, so two groups are created as a result:

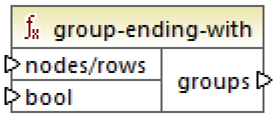


Note: One additional group is created if records exist before the first one that satisfies the condition. For example, if there were more "line" records before the first "heading" record, these would all be placed into a new group.

For more information, see the reference to the group-starting-with 1043 function.

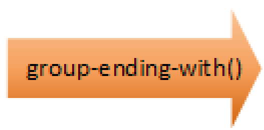
group-ending-with

The `group-ending-with` function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, ending with the record that satisfies the condition.



In the example below, the condition is that "Key" must be equal to "trailing". This condition is true for the third and fifth records, so two groups are created as a result:

Key	Value
line	A
line	B
trailing	Total 1
line	C
trailing	Total 2



Key	Value
line	A
line	B
trailing	Total 1

Key	Value
line	C
trailing	Total 2

Note: One additional group is created if records exist after the last one that satisfies the condition. For example, if there were more "line" records after the last "trailing" record, these would all be placed into a new group.

For more information, see the reference to the group-ending-with[1039] function.

5.8.1 Example: Grouping Records by Key

This example shows you how to group records with the help of the `group-by` function, and also illustrates how to aggregate data. This example is accompanied by a demo mapping available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\GroupTemperaturesByYear.mfd**. This mapping reads data from an XML file that contains a log of monthly temperatures, as illustrated in the code listing below:

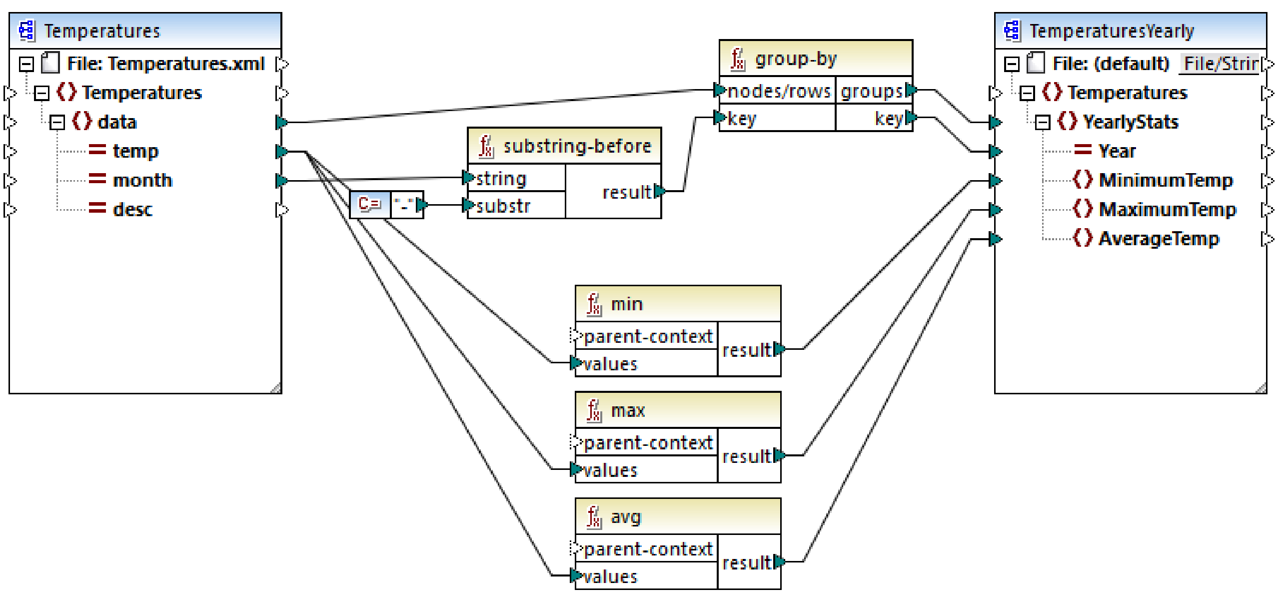
```
<Temperatures>
   <data temp="-3.6" month="2006-01" />
   <data temp="-0.7" month="2006-02" />
   <data temp="7.5" month="2006-03" />
   <data temp="12.4" month="2006-04" />
   <data temp="16.2" month="2006-05" />
   <data temp="19" month="2006-06" />
   <data temp="22.7" month="2006-07" />
   <data temp="23.2" month="2006-08" />
   <data temp="18.7" month="2006-09" />
```

```
    <data temp="11.2" month="2006-10" />
    <data temp="9.1" month="2006-11" />
    <data temp="0.8" month="2006-12" />
    <data temp="-3.2" month="2007-01" />
    <data temp="-0.3" month="2007-02" />
    <data temp="6.5" month="2007-03" />
    <data temp="10.6" month="2007-04" />
    <data temp="19" month="2007-05" />
    <data temp="20.3" month="2007-06" />
    <data temp="22.3" month="2007-07" />
    <data temp="20.7" month="2007-08" />
    <data temp="19.2" month="2007-09" />
    <data temp="12.9" month="2007-10" />
    <data temp="8.1" month="2007-11" />
    <data temp="1.9" month="2007-12" />
</Temperatures>
```

The business requirement of this mapping is two-fold:

1. Group temperatures of each year together.
2. Find out the minimum, maximum, and the average temperature of each year.

To achieve the first goal, the mapping calls the **group-by** [1037] function. To achieve the second goal, it calls the **min** [985], **max** [984], and **avg** [982] aggregation functions. All of these functions are MapForce built-in functions, and you can add them to any mapping by dragging them from the Libraries window, see How to Add a Function to the Mapping [905].



GroupTemperaturesByYear.mfd

The way MapForce executes a mapping (and the recommended approach to start reading one) is by looking at the topmost item of the target component. In this example, an **YearlyStats** item will be created for each group returned by the **group-by** function. The **group-by** function takes as first argument all **data** items from the source and groups them by whatever is connected to the **key** input. Since the requirement is to group temperatures by year, the year must be obtained first. To achieve this, the **substring-before**[1063] function extracts the year part from the **month** attribute of each **data** element. Namely, it takes as argument the value of **month** and returns the part before the first occurrence of **substr**. As illustrated above, in this example, **substr** is set to the dash character; therefore, if given the value "2006-01", the function will return "2006".

Finally, the values of **MinimumTemp**, **MaximumTemp**, and **AverageTemp** are obtained by connecting these items with the respective aggregate functions: **min**, **max**, and **avg**. All three functions take as input the sequence of temperatures read from the source component. These functions do not need a **parent-context** argument, because they already work in the context of each group. In other words, there is a parent connection —from **data** to **YearlyStats**— which provides the context for each aggregation function to work on.

To preview the mapping output, click the **Output** tab. Notice that the number of groups coincides with the number of years obtained by reading the source file, for example:

```
<Temperatures>
   <YearlyStats Year="2006">
      <MinimumTemp>-3.6</MinimumTemp>
      <MaximumTemp>23.2</MaximumTemp>
      <AverageTemp>11.375</AverageTemp>
   </YearlyStats>
   <YearlyStats Year="2007">
      <MinimumTemp>-3.2</MinimumTemp>
      <MaximumTemp>22.3</MaximumTemp>
      <AverageTemp>11.5</AverageTemp>
   </YearlyStats>
</Temperatures>
```

Note: For simplicity, the code listings above contain less data than the actual input and output used by the demo mapping.

5.9 Exceptions

An exception is a special component type that enables you to stop the mapping process and display an error when a condition returned by a filter occurs. You can add an exception when your mapping includes a filter that checks for a true/false condition (see Filters and Conditions 878). For example, you may want to throw an exception if the value of some mapping item is greater than some custom threshold.

Exceptions can also be added when you design WSDL Web services with MapForce (see Web Service Faults 1218).

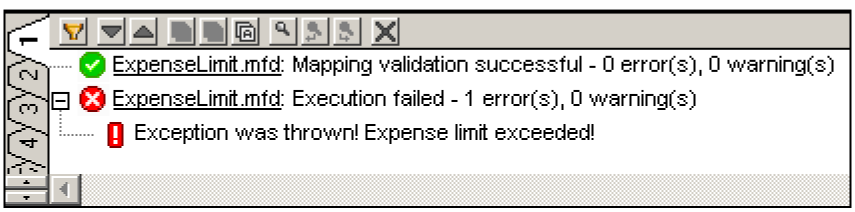
To add an exception to the mapping:

1. On the **Insert** menu, click **Exception**.
2. Click the **Insert Exception** ( ) toolbar button.
3. Connect the **throw** input of the exception either to an **on-true** or **on-false** output of a filter.
4. Optionally, connect the **error-text** input of the exception to another component (typically, a constant) that supplies the text of the error when this exception is thrown.

Note: Both the **on-true** and **on-false** outputs of the filter must be connected. Specifically, one of these outputs must be connected directly to the exception (without any intermediary functions or components). The other output must be connected to the target component, either directly, or through other intermediary components.

When the mapping encounters an exception, you are notified about it as follows:

- In MapForce, the Messages window displays an error, and the exception text (in this case, "Expense limit exceeded").

  If the mapping language is XSLT 2.0 or XQuery, an "Execution failed" error appears in the Messages window, and the respective XSLT2 or XQuery tab is opened. The error line is highlighted in the Messages window.

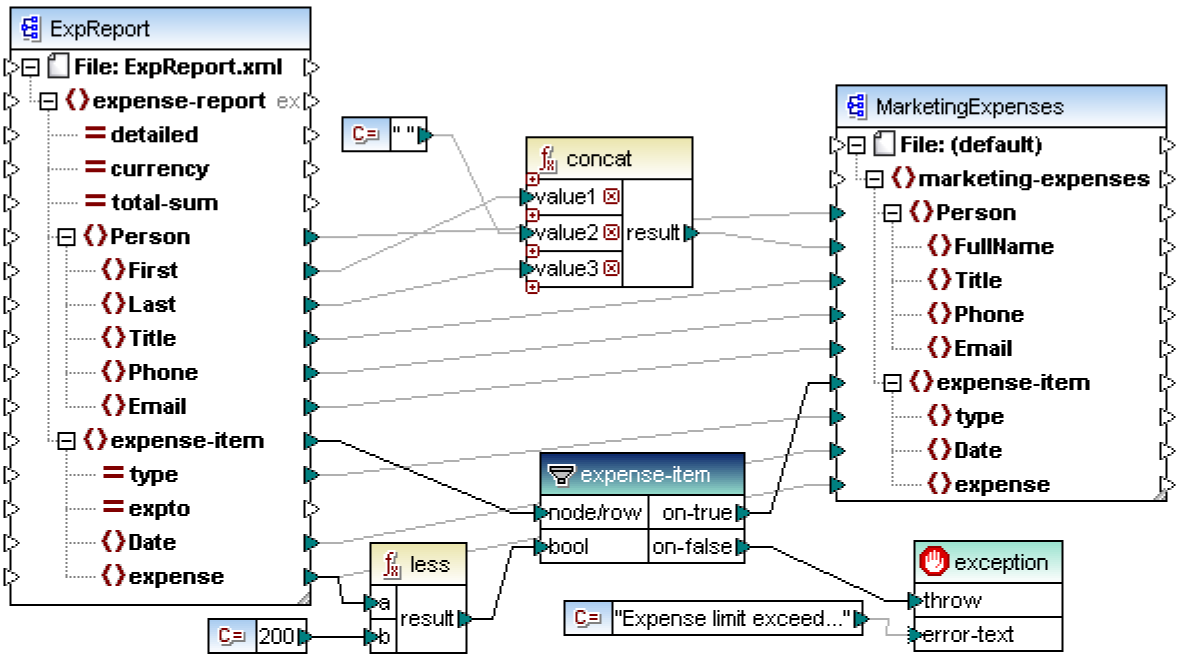
- If you run the mapping with MapForce Server, the error "Exception was thrown!" is returned, followed by the custom exception text you have defined in MapForce.

```
c:\codegen\mfx\ExpenseLimit>"C:\Program Files (x86)\Altova\MapForceServer2016\bi
n\MapForceServer.exe" run ExpenseLimit.mfx
Exception was thrown! Expense limit exceeded!
```

- If you run the mapping from the generated C#, C++, or Java code, the error "USER EXCEPTION" is returned, followed by the custom exception text you have defined in MapForce.

5.9.1 Example: Exception on "Greater Than" Condition

This example illustrates a mapping that throws an exception when a "Greater Than" condition occurs. The sample mapping accompanying this example can be found at: **<Documents>\Altova\MapForce2023\MapForceExamples\ExpenseLimit.mfd**.



This mapping throws an exception whenever the **expense** item in the source XML instance has a value greater than 200. The value "200" is provided by a constant. The **less** function is then used to compare the two values. If the value of **expense** is less than 200, then its parent, the **expense-item**, is passed on to the filter, and no exception is thrown. Otherwise, an exception is thrown, with the custom text "Expense limit exceed".

As shown above, the exception is identified by the  icon and it consists of two items: **throw** and **error-text**. The **throw** item must be connected to the **on-false** or **on-true** output of a filter. The **error-text** is connected to a constant which provides the custom text of the exception.

Importantly, both outputs of the filter are connected; otherwise, the exception would not be thrown. In this particular example, the **on-false** output is connected to the exception, while the **on-true** output is connected to the target component.

5.9.2 Example: Exception When Node Does Not Exist

This example illustrates how to throw an exception when a node in the source XML schema does not exist. For the sake of simplicity, this example uses the same XML schema both as source and target component.

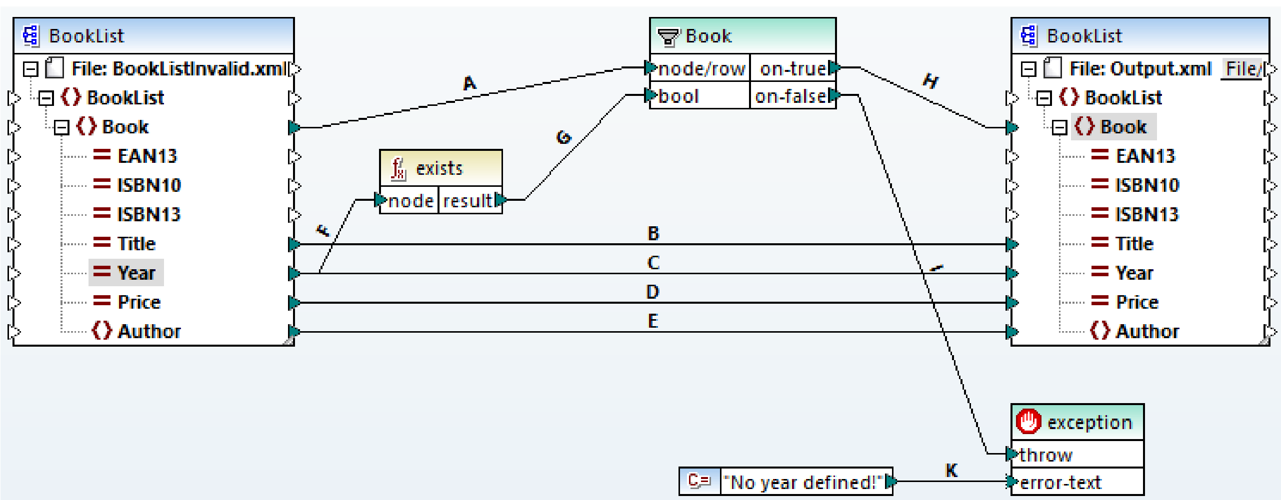
To add the source schema to the mapping:

1. On the **Insert** menu, click **XML Schema/File**, and browse for **<Documents>\Altova\MapForce2023\MapForceExamples\BookList.xsd**.
2. When prompted to provide an instance file, click **Skip**.
3. When prompted to select a schema root element, select **BookList** as root element.

To add the target schema, follow the same steps. Then, using the corresponding commands from **Insert** menu (or the corresponding toolbar buttons), add the following:

- A **Filter: Nodes/Rows** component (see also Filters and Conditions 878)
- A constant with the text "No year defined!"
- An exception

Finally, drag the **exists** function from the **Libraries** window into the mapping area, and make the connections as illustrated below.



According to the XML schema, all attributes of the **Book** element are optional, except the book title. Therefore, the "Year" attribute may or may not exist in a valid XML instance. The goal of the mapping is to process successfully an XML instance where the "Year" attribute exists for each book. Otherwise, the mapping must throw an exception.

To test the successful execution of the mapping:

1. Double-click the header of the source component and, next to **Input XML file**, browse for the following file: **<Documents>\Altova\MapForce2023\MapForceExamples\BookList.xml**.
2. Click the **Output** button to run the mapping.

To test the exception:

1. Create, in the same directory, a copy of the **BookList.xml** file called **BookListInvalid.xml**.

2. Modify it so as to remove the "Year" attribute from a Book element.
3. Double-click the header of the source component, and, next to **Input XML file**, browse for the **BookListInvalid.xml** file.
4. Click the **Output** button to run the mapping.

Let's now have a closer look at how the mapping works.

Connection **A** ensures that a book in the target instance is created for each book in the source instance. Connections **B, C, D, E** ensure that the "Title", "Year", "Price", and "Author" are copied from the source to the target, for each book.

Connection **F** triggers the **`exists`** function to check for the existence of the "Year" attribute. Connection **G** passes the function result (`true` or `false`) to the filter. If the result is `true`, the "Year" attribute exists, and the book is passed on to the filter, and subsequently to the target through connection **H**.

> Notice that the filter was not connected directly to the **Year** output of the source component. Had we done so, the filter would filter the **Year** by its own existence, which is not meaningful, and the exception would never be thrown.

Connection **I** is there because the exception must be connected either to an **on-false** or **on-true** output of a filter, according to the rules. Finally, connection **K** passes the custom error text from the constant to the exception component.

6 Functions

In MapForce, you can use the following categories of functions to transform data according to your needs:

- **MapForce built-in functions** — these functions are predefined in MapForce and you can use them in your mappings to perform a wide range of processing tasks that involve strings, numbers, dates, and other types of data. You can also use them to perform grouping, aggregation, auto-numbering, and various other tasks. For reference to all available built-in functions, see Function Library Reference 979.
- **Node functions and defaults** — these are more specialized functions that let you create and apply custom processing logic to one or multiple descendant nodes on a mapping component. They enable you to process data either *before* it reaches a node of a mapping structure, or *immediately after* it leaves a node. For more details, see Defaults and Node Functions 912.
- **User-defined functions (UDFs)** — these are MapForce functions that you can create yourself, using as basis the native component kinds and built-in functions already available in MapForce, see User-Defined Functions 927.
- **Custom functions** — these are functions that you can import from external sources such as XSLT libraries, XQuery library modules, Java .class files, .NET .dll files, and adapt to MapForce. Note that, in order to be reusable in MapForce, your custom functions must return data of simple type (such as string or integer) and they must also take parameters of simple type. For more information, see Importing Custom XSLT Functions 942, Importing Custom XQuery 1.0 Functions 949, and Importing Custom Java and .NET Libraries 953.

Note: You can import custom external libraries of functions either directly (no configuration required) or by configuring a MFF (MapForce Function File) recognized by MapForce. If you use the latter approach, you can also import C++ libraries, in addition to Java classes and .NET assemblies. Note that libraries imported via .mff files must meet the prerequisites mentioned in Referencing Java, C# and C++ Libraries Manually 960.

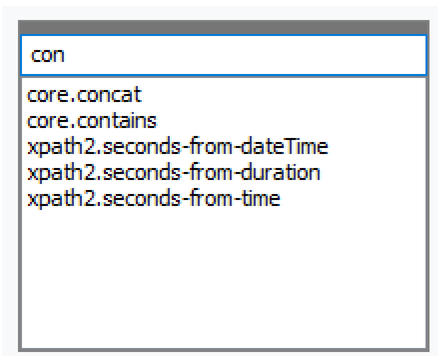
6.1 Functions Basics

The subsections below give an overview of basic function-related actions. The functions you see in the **Libraries** window depend on the transformation language you have selected. For more information, see Transformation Languages[27].

Add function

MapForce includes a large number of built-in functions that you can add to a mapping. For more information about all available built-in functions, see Function Library Reference[979]. To add a function to a mapping, you can choose one of the following methods:

- Press and hold the required function in the **Libraries** window and drag it to the mapping area. To filter functions by name, start typing the function name in the text box at the bottom of the window.
- Double-click anywhere in the empty area of the mapping and start typing the function name (*see screenshot below*). To see a tooltip with more details about a function, click this function in the list. To add a function to your mapping, double-click the relevant function in the combo box.



Add a UDF

You can also add user-defined functions (UDFs) to your mapping using the same approaches as described above if (i) a UDF has already been created in the same mapping or (ii) you have imported a mapping that contains a UDF as a local or global library.

Add a constant

Constants enable you to supply custom text and numbers to a mapping. To add a constant to your mapping, you can choose one of the following options:

- Right-click anywhere in the empty mapping area and select **Insert Constant** from the context menu. Enter the value and select one of the data types: *String*, *Number*, or *All other*.
- Select the **Insert | Constant** menu command. Enter the value and select one of the data types: *String*, *Number*, or *All other*.
- Click the **Constant** toolbar command. Enter the value and select one of the data types: *String*, *Number*, or *All other*.
- Double-click anywhere in the empty mapping area. Type the double quotation mark followed by the constant value. The closing double quotation mark is optional. To add a numeric constant, just type the number.

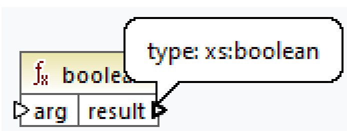
Search for a function

To search for a function in the **Libraries** window, start typing the function name in the text field at the bottom of the window. By default, MapForce searches by function name and description text. If you want to exclude the function description from the search, click the down-arrow and disable the *Search in function descriptions* option. To cancel the search, press the **Esc** key or click ✖.

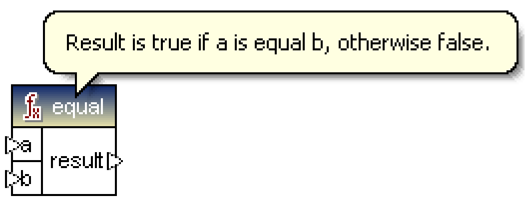
To find all occurrences of a function within the currently active mapping, right-click the function name in the **Libraries** window and select **Find All Calls** from the context menu. The search results are displayed in the **Messages** window.

View a function's type and description

To view the data type of a function's input/output argument, move your mouse over the argument part of a function (*see screenshot below*). Make sure that the (**Show tips**) toolbar button is enabled.

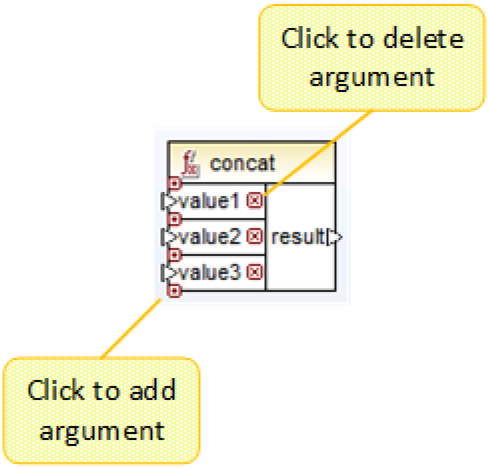


To view the description of a function, move the mouse over the function's header (*see screenshot below*). Make sure that the (**Show tips**) toolbar button is enabled.



Add/delete function arguments

For some MapForce built-in functions, it is possible to add as many parameters as you need for your mapping purposes. One such example is the **concat** [1057] function. To add or delete function arguments (for functions that support that), click **Add parameter** ( ) or **Delete parameter** ( ) next to the parameter you want to add or delete, respectively (*see below*). Moving a connection to the symbol adds another parameter and connects this parameter.


Click to delete argument
concat
value1
value2
value3
result
Click to add argument

6.2 Manage Function Libraries

In MapForce, you can import and use the following kinds of libraries in a mapping:

- Any mapping design files (*.mfd) that contain user-defined functions (UDFs). This specifically refers to mapping files that contain UDFs created with MapForce, using the MapForce built-in functions and components as building blocks. For further information, see Creating User-Defined Functions 928.
- Custom XSLT files that contain functions. This refers to XSLT functions written outside of MapForce that qualify for import into MapForce as described in Importing Custom XSLT Functions 942.
- Custom XQuery 1.0 files that contain functions. This refers to XQuery functions written outside of MapForce that qualify for import into MapForce as described in Importing Custom XQuery 1.0 Functions 949.
- Java .class files and .NET .dll libraries which qualify for import into MapForce as described in Importing Custom Java and .NET Libraries 953.

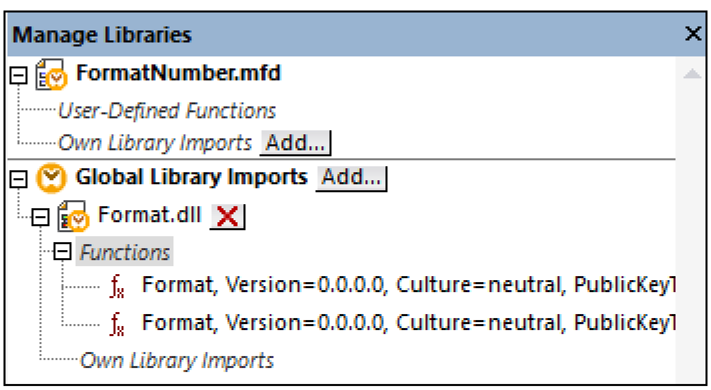
Note: You can import custom external libraries of functions either directly (no configuration required) or by configuring a MFF (MapForce Function File) recognized by MapForce. If you use the latter approach, you can also import C++ libraries, in addition to Java classes and .NET assemblies. Note that libraries imported via .mff files must meet the prerequisites mentioned in Referencing Java, C# and C++ Libraries Manually 960.

Manage Libraries window

You can view and manage all libraries used by a mapping file from the Manage Libraries window. This includes UDFs and custom libraries.

By default, the **Manage Libraries** window is not visible. To display it, do one of the following:

- In the **View** menu, click **Manage Libraries**.
- Click **Add/Remove Libraries** at the bottom of the **Libraries** window.



You can choose to view UDFs and libraries only for the mapping document that is currently active or for all open mapping documents. To view imported functions and libraries for all of the currently open mapping documents, right-click inside the window and select **Show Open Documents** from the context menu.

To display the path of the open mapping document instead of the name, right-click inside the window and select **Show File Paths** from the context menu.

Data displayed in the Manage Libraries window is organized as a tree hierarchy as follows:

- Any currently open mapping documents are displayed as top-level entries. Each entry has two branches: **User-Defined Functions** and **Own Library Imports**.
    - o The **User-Defined Functions** branch displays any UDFs contained in that document.
    - o The **Own Library Imports** branch displays libraries imported *locally* into the current mapping document. The term "libraries" means other mapping documents (.mfd files that contain user-defined functions) or custom external libraries written in XSLT 1.0, XSLT 2.0, XQuery 1.0*, Java*, C#*, or .mff files mentioned previously. Note that the **Own Library Imports** structure could be several levels deep, since any mapping document may import any other mapping document as a library.
- The **Global Library Imports** entry encloses any custom libraries that you have imported *globally* at application level. Again, in case of .mfd files, the structure could be several levels deep, for the reasons mentioned above.

** These languages are supported only in MapForce Professional or Enterprise edition.*

Note: The XSLT, XQuery, C#, and Java libraries may have dependencies of their own. Such dependencies are not displayed in the Libraries window.

Context menu commands

You can quickly perform various operations against objects in the Manage Libraries window by right-clicking an object and selecting one of the following context menu options:

Command	Description	Applicable for
Open	Opens the mapping.	Mappings
Add	Opens a dialog box where you can browse for a custom library of functions.	Own Library Imports
Locate Function in Libraries Window	Changes focus to the Libraries window, and selects the function.	Functions
Cut, Copy, Delete	These standard Windows commands are applicable only to MapForce user-defined functions. You cannot copy-paste functions from external XSLT files or other library kinds.	User-defined functions
Paste	Lets you paste a user-defined function that was previously copied to clipboard into the current library.	Libraries (UDF)
Options	Opens a dialog box where you can set or change options for the current library.	Libraries
Show All Open Documents	When this option is switched on, the Manage Libraries window will display all currently open mappings. This is typically useful if you need to copy-paste functions between mappings.	Always

Command	Description	Applicable for
	Otherwise, only the mapping that is currently in focus is shown.	
Show File Paths	When this option is switched on, objects in the Manage Libraries window are displayed with their full path. Otherwise, only the object name is shown.	Always

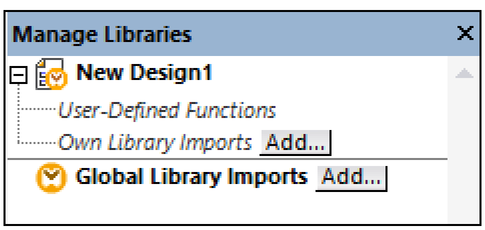
6.2.1 Local and Global Libraries

You can import libraries *locally* or *globally*. Global imports are at application level. If a library was imported globally, you can use its functions from any mapping.

Local imports are at mapping file level. For example, let's suppose that, while working on mapping **A.mfd**, you decide to import all user-defined functions from mapping **B.mfd**. In this case, mapping **B.mfd** is considered to be imported as a local library into **A.mfd** and you can use functions from **B.mfd** in **A.mfd** as well. Likewise, if you import functions from an XSLT file into **A.mfd**, this is also a local import.

You can view and manage all local and global imports from the Manage Libraries window. To import a library, do one of the following:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[33]. The **Manage Libraries** window opens (*see screenshot below*).

2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

Conflicting function names

You may come across situations where the same function name is defined at any of the following levels:

- in the main mapping
- in a library that was imported locally
- in a library that was imported globally

When it encounters such cases, MapForce will attempt to call the function exactly in the order above, to prevent ambiguity. That is, the function defined directly in the mapping takes precedence if the same function name exists in a locally imported library. Also, the function imported locally takes precedence over the function imported globally (assuming that both functions have the same name).

If multiple functions with the same name exist, only the "winning" function will be called, according to the rule above; any other ambiguous function names will be blocked. Such blocked functions appear as grayed out in the Libraries window, and it is not be possible to use them in the mapping.

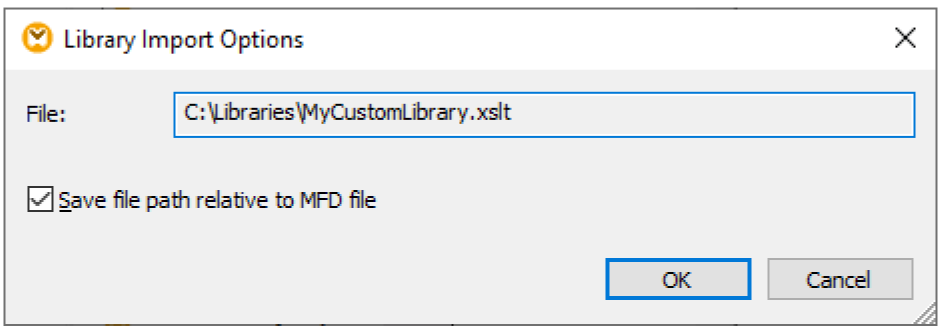
6.2.2 Relative Library Paths

You can set the path of any imported library file to be relative to the mapping design file (.mfd), provided that the library was imported locally (not globally), as described in Local and Global Libraries [910].

> Setting a relative library path is applicable only for those libraries that were imported *locally* at document level. If a mapping was imported *globally* at program level, its path is always absolute.

To set a library path as relative to the mapping design file:

1. Click **Add/Remove Libraries** at the base of the Libraries window. The Manage Libraries window [35] opens.
2. Click **Options** next to the library of interest. (Alternatively, right-click the library, and select **Options** from the context menu.)



3. Select the **Save file path as relative to MFD file** check box.

Note: If the check box is grayed out, make sure that the library was indeed imported locally, and not globally.

When the check box is selected, MapForce will keep track and update the path to any referenced library files when you save the mapping file to a new directory using the **Save as** menu command. Also, if the library files are in the same directory as the mapping file, the path reference will not be broken when you move the entire directory to a new location on the disk, see also Using Relative Paths on a Component [85].

Note that the **Save file path as relative to MFD file** check box specifies that paths are *relative to the mapping file*, and it does not affect paths in generated code. For information about how library references are handled in generated code, see Paths in Various Execution Environments [87].

6.3 Defaults and Node Functions

Defaults and node functions are particularly useful when you want to apply the same processing logic to multiple descendant items in a structure. Usually you would need to copy-paste the same function multiple times in the mapping. This could clutter the mapping and make it more difficult to understand. Defaults and node functions can apply to a single item or to multiple items at once. Defaults replace empty sequences. If the connection transports a value, the default will be ignored.

Defaults and node functions are suitable for most components that have a tree with nodes (e.g., XML, EDI, Join components, variables).

Defaults and node functions are compatible only with the Built-In transformation language. Running such mappings from generated C#, C++, Java program code or with generated XSLT/XQuery transformations is not supported. On the server side, you can execute such mappings with MapForce Server Advanced Edition.

Advantages of node functions and defaults

Creating node functions and defaults means defining a *rule*. Rules have the following important characteristics that make them flexible and easy to use:

- *Inheritance.* When you define a rule on an item that has descendants, the rule will be inherited by descendants by default unless you choose to disable this option. If the item for which you define the function has multiple levels of child items nested under it, you can choose to apply the rule only to direct child items or to all descendant items.
- *Filtering.* MapForce applies rules conditionally, based on the data type of each item. This makes it possible, for example, to apply a certain default value or function to all items of type `string` and a different default or function to all items of type `decimal`. For details, see *Scenario 2* in Use-Case Scenarios 917. You can also define more advanced filtering options: For example, you can specify a data type that your function must match (this could be a category of data types such as `numeric`) and then filter the nodes of this data type based on the node name or node type (e.g., `integer`). For details, see *Scenario 5* 921.

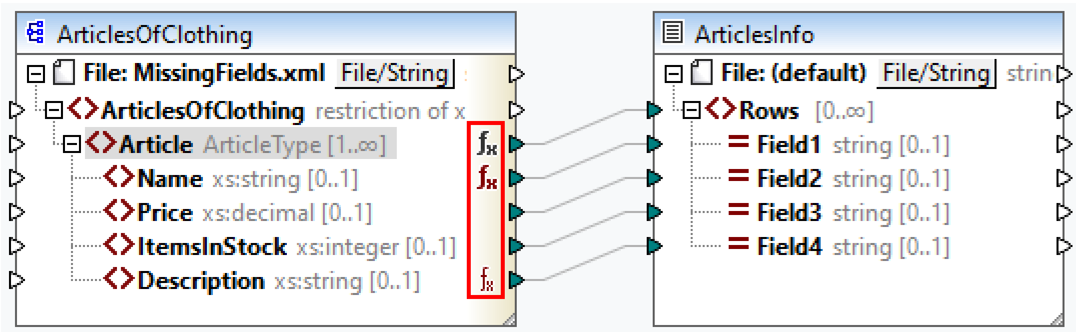
For example, you can instruct MapForce to do the following:

- Every time an empty or null value is encountered, replace it with some other value and do this recursively for all descendant items.
- Every time a specific value is encountered, replace it with some other value (or with an empty string) and do this recursively for all descendant items.
- Replace all database null values with empty strings or custom text.
- Append a custom prefix or suffix to all values that are written to a target file or database.

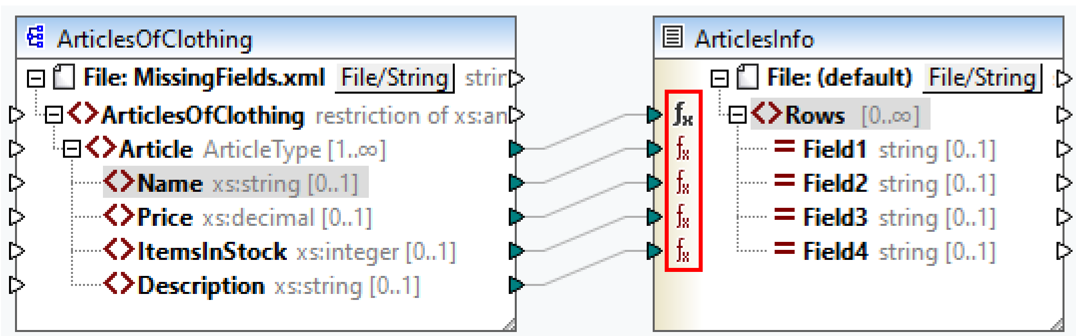
Output vs. input side

You can define node functions and defaults on the input side, output side, or on both sides of a component, depending on your needs. In MapForce, a mapping works in the following way: (i) it first reads data from a source component (e.g., an XML file), (ii) then optionally processes the data in some way (e.g., using a function), and (iii) finally writes data to some target component (e.g., a database). Considering this basic principle, you can set node functions and defaults at various stages:

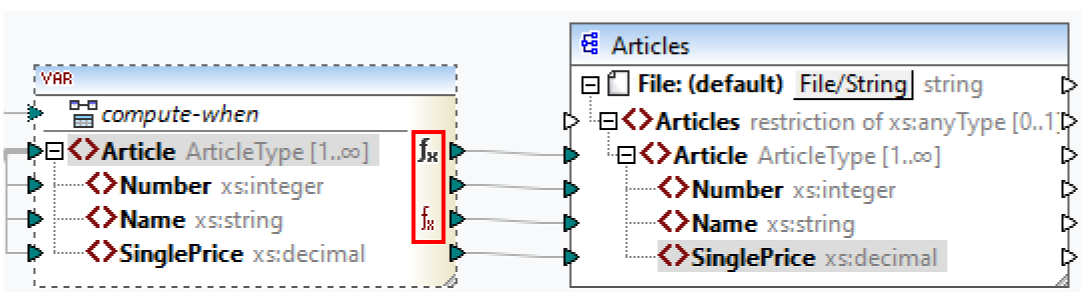
- Immediately *after* data is read from the source, but before it is further processed by your mapping, which means the function/default is defined on the output side of the source component (*see example below*).



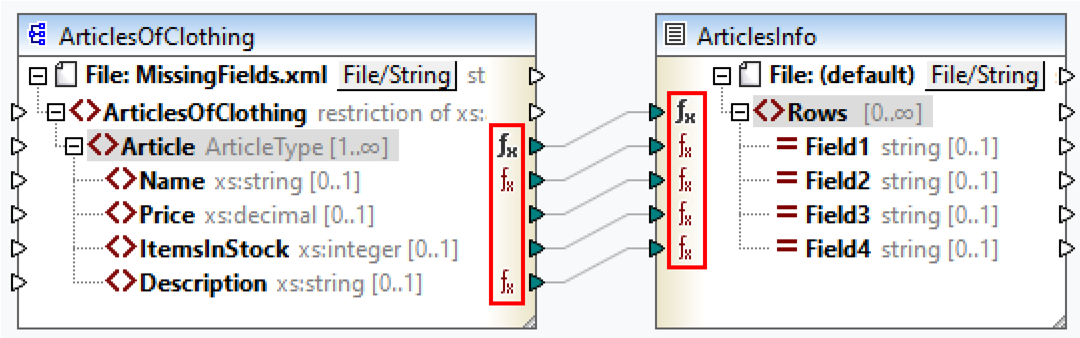
- Immediately *before* data is written to the target component (and after it has finished all intermediary processing), which means the function/default is defined on the input side of the target component (*see example below*).



- At an intermediary stage in the mapping process. For example, if the mapping contains an intermediary variable of complex type (e.g., an XML structure), you can trim all values before they are supplied to the XML structure or immediately after they are returned by the XML structure (*see example below*).



- On the output side of a source component and on the input side of a target component. In the example below, a default has been defined for all the nodes of type `string` in the `ArticlesOfClothing` component. In the `ArticlesInfo` component, we have defined a node function that will transform all values of string nodes into uppercase characters.



In this section

This section explains how to configure a rule, describes real-life scenarios in which defaults and node functions can be useful, and shows how to add node metadata to node functions. The section is organized into the following topics:

- Rule Configuration 914
- Use-Case Scenarios 917
- Node Metadata in Node Functions 923

6.3.1 Rule Configuration

You can create node functions and defaults for almost any item in the mapping. Creating node functions and defaults means defining a *rule*. To create a rule, select an item (node or field) for which you want to define a rule. This can be a single node or a node with child nodes. If you choose to create a rule for a node with children, the rule will apply to all the children unless you disable this option explicitly.

Important aspects of defaults and node functions

Defaults and node functions have the following important aspects:

- You can create defaults and node functions on the input side of a target component or on the output side of a source component. To establish which side is right for your needs, see Input vs. Output Side 912.

- When multiple rules exist for one and the same item, MapForce will apply the rule that is closer to that item. To find out how to override rules, see *Scenario 4* in Use-Case Scenarios 920.

- Defaults and node functions can be created if the connection type between the source and target is source-driven or target-driven. However, copy-all connections are not supported. Specifically, node functions and defaults do not apply to descendants of copy-all connections. The parent node with the copy-all connection can have node functions and defaults but only if this parent node has a simple value, for example, an XML element with simple-type content and attributes. For more information about connection types, see Connection Types 92.

- Creating defaults and node functions is not supported for the `File` node.

- Inside a node function, only certain components meaningful in this context are supported (e.g., built-in functions, variables, if-else conditions). Complex structures such as XML, JSON, EDI, or databases are not supported. Adding inline user-defined functions and join components to a node function is not supported.

- A node function can have one input parameter or no parameter at all. The input parameter is always called `raw_value`. Do not delete the input parameter even if you do not need an input for your function; otherwise, validation errors will occur when you run the mapping. The same applies to the function's output. Should you need to restore an accidentally deleted input component, run the menu command **Function | Insert Input**.

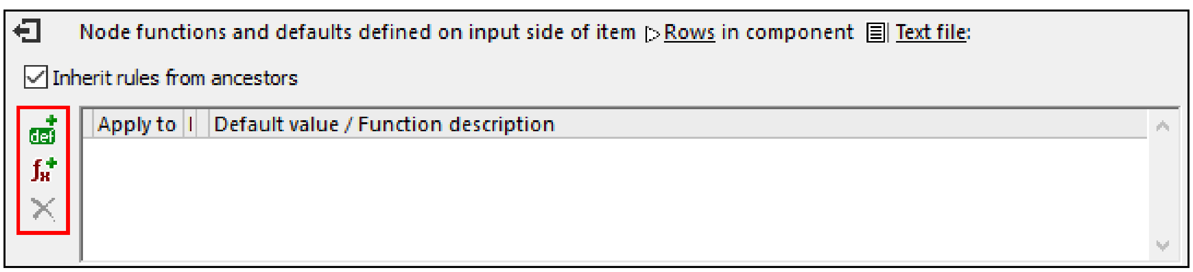
- MapForce allows you to supply node metadata (e.g., a node name, an annotation) to a node function. For details, see Node Metadata in Node Functions 923.

- The **Filter Node Functions and Defaults** dialog box allows you to choose from an extensive range of data types, some of which are categories of types. This means that they will match a larger selection of types. For example, the type `string` matches various other data types derived from `string`, such as `normalizedString`, `token`, `NCName`, `NMTOKEN`, `IDREF`, `ENTITY`, and others. Likewise, the type `decimal` will match the derived types `integer`, `long`, `short`, and others. The hierarchy of types is specified in the XML Schema W3C Recommendation.

Create a rule

To create a rule, take the following steps:

1. Right-click the node of interest and select **Node Functions and Defaults | Input/Output Node Functions and Defaults** from the context menu. Whether it is *Input* or *Output Node Functions and Defaults* depends on the side of the component on which you need to create a node function or a default. Alternatively, right-click a connector and select the node function command for that side. The **Mapping** pane will display the Node Functions window, in which you can define defaults and node functions (*red rectangle in screenshot below*).



   If a parent node has rules defined for it, and you want to transfer these rules from the parent to a child, select the *Inherit rules from ancestors* check box (*screenshot above*). For more information about inheritance, see Use-Case Scenarios 917.

2. The next step is to define whether you want to add a default ( ) or a function ( ). As soon as you click the relevant option, a new rule will be created (a row in the grid in the Node Functions window). For information about rule configuration, see *Rule Configuration* below. If you are defining a function, the mapping area will display the function's input and output parameters.

Edit/delete rules

To view, modify, or delete a rule, click the icon (black or red) next to a node of interest. The Node Functions window will display all the rules defined for this node. If you have added a node function, you will see the Edit button in the grid, which allows you to change the implementation of the function. If the Edit button is not present, the function is most likely defined on some ancestor. In this case, click the icon near the item for which the rule has been defined.

To delete a rule, select it from the grid in the Node Functions window and then click the button.

Configure a rule

As soon as you select **Node Functions and Defaults | Input/Output Node Functions and Defaults** from the context menu (*see instructions above*), you will see the Node Functions window in which you can configure node functions and defaults. The available settings are described below.

- *Apply to:* A rule can apply to the current item, its direct child items, or to all descendant items. If the item you have selected has no descendants, only the *Current item* option will be available.

- *Data type:* Click the ... button and select a data type from the dialog box. The rule will apply only to items of this data type (or a derived data type). If the item you have selected has no descendants, the item's data type is the only choice.

- *Default Value/Function Description:* If you are defining a default, type the default value that you wish to set for the selected item (and all descendants, if applicable). To set an empty string as a default, leave this field empty. If you are defining a function, this field is for information only: It displays a summary of the function.

To exit the Node Functions window, click the button in the upper-left corner of this window or press **Escape**.

Visual clues

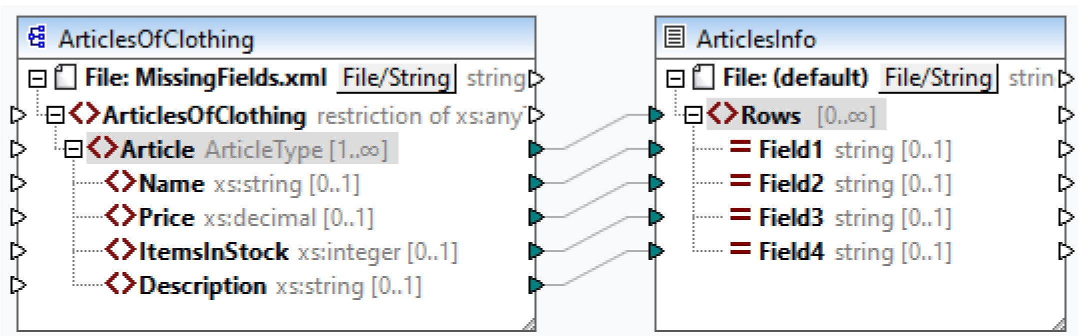
MapForce displays different visual clues to help you understand which rules are defined and whether they apply to a specific node (*see table below*).

Icon	Description
	This icon indicates that a rule has been defined for this item and may affect all its descendants. Click the icon to modify or delete the rule.
	This icon indicates that the item inherits the rule defined at ancestor level.
	This icon indicates that a rule is defined for and applies to this item. This icon usually appears when a function or a default is defined for a single node.
	This icon indicates that even though a rule applies to this item, it is deliberately blocked. This icon is displayed only if inheritance is blocked and no other rules are defined for this node. If a rule from an ancestor does apply, the icon has priority.
	This icon indicates that a rule defined for this item is inactive. For example, this icon may appear for child items that are not connected yet.
	This icon indicates that the rule is currently inactive, because no matching and connected nodes have been identified. By *matching* the following is meant: Before applying a function or a default, MapForce looks for the data type you have defined; if you have also specified a filter

Icon	Description
	that must match a specific node name, for example, MapForce will also look for the relevant node name.

6.3.2 Use-Case Scenarios

This topic describes different scenarios in which defaults and node functions can be useful. To test these scenarios, you will need the following sample mapping: **`Tutorial\MissingFields.mfd`**. Our starting point is the mapping illustrated below.



Some of the nodes in the source file are empty. At this stage, the output file looks as follows:

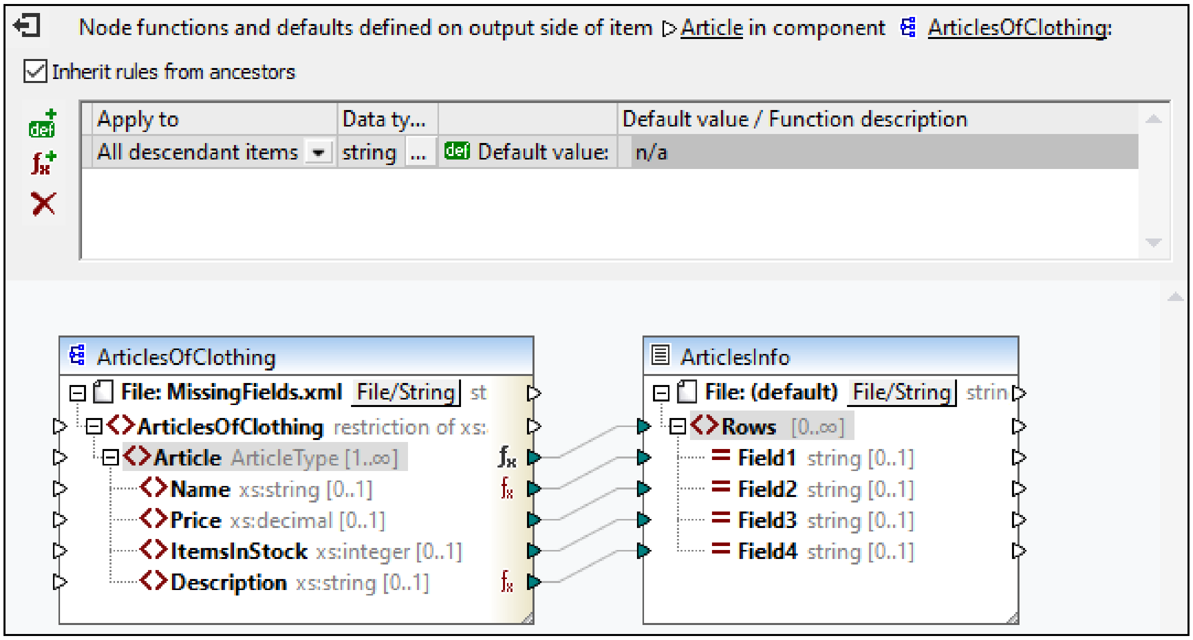
`T-Shirt`	`25.3`	`20`	`Available in black, blue, and red`
`Shirt`	`70.3`	`60`	
`Pants`			`Limited stock`
`Jacket`	`57.5`	`40`	

The scenarios described below will show you how to set a default for all nodes of type `string`, set defaults for different data types, block rules for specific nodes, override inherited rules, set a node function and conditionally apply this function to relevant nodes, and apply defaults to unconnected nodes.

Scenario 1: Set a default for all string nodes

In our mapping, some of the nodes of type `string` are empty. Our goal is to set a default value `n/a` instead of the empty strings. Take the following steps:

1. Right-click the `Article` element and select **Node Functions and Defaults | Output Node Functions and Defaults** from the context menu.
2. Click the option in the Node Functions window and type `n/a` in the *Default value* field (*see screenshot below*).



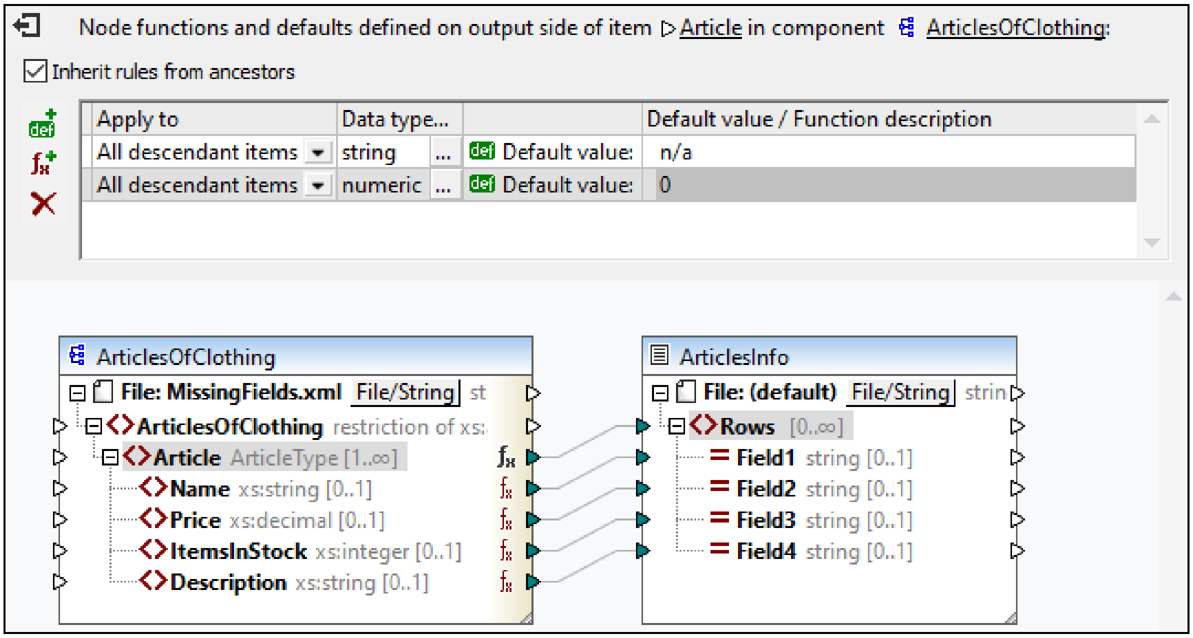
In the `ArticlesOfClothing` component, the `Name` and `Description` elements are of type `string`, which makes the default value suitable for both of them. Therefore, the icon appears next to these nodes. Since default values apply only to empty values, only two `Field4` nodes have received the `n/a` values in the output (*highlighted yellow below*).

T-Shirt	25.3	20	Available in black, blue, and red
Shirt	70.3	60	n/a
Pants			Limited stock
Jacket	57.5	40	n/a

Scenario 2: Set defaults for different data types

Besides setting a default for nodes of one specific type, you can additionally set another default for some other data type. For example, one of the articles has empty values in `Price` and `ItemsInStock`, both of which are of numeric type. Now our goal is to replace these empty values with `0`. Take the steps below:

1. Repeat the steps from *Scenario 1*. This will add a default value for all nodes of type `string` that have empty values.
2. Add another default and select `numeric` as a data type.
3. Type `0` in the *Default value* field (*see screenshot below*).



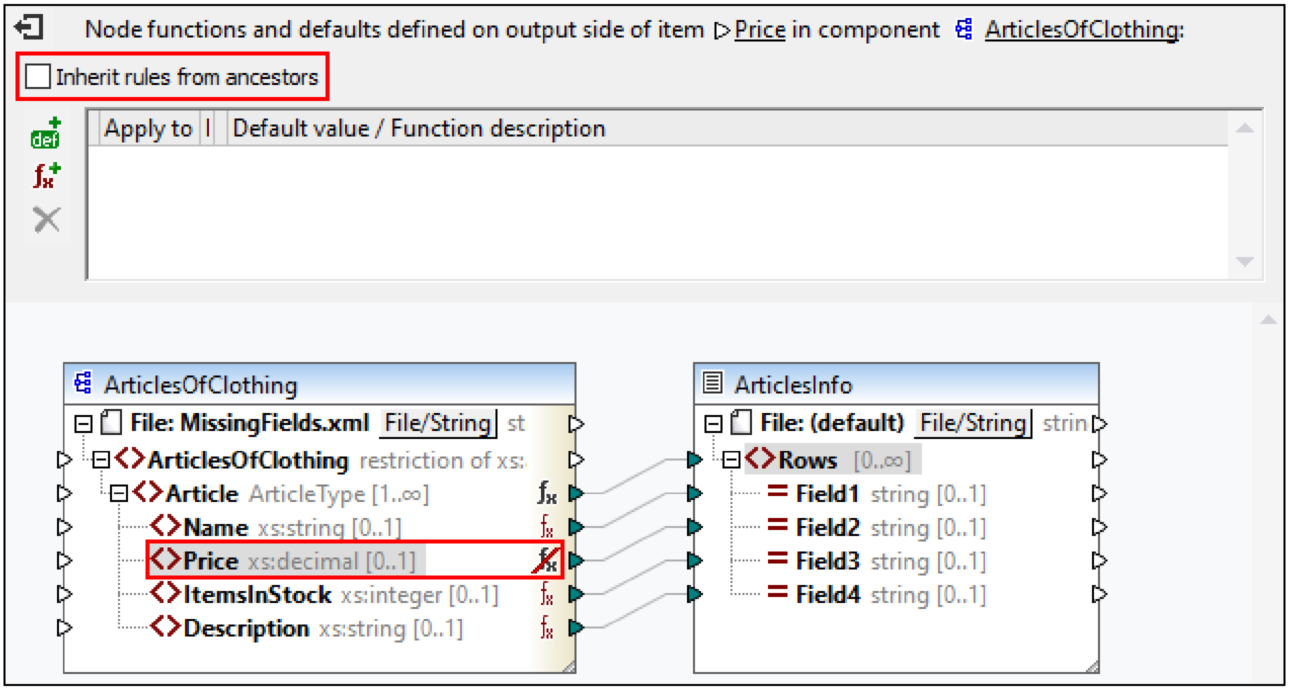
The output now looks as follows (*note the highlighted parts*):

`T-Shirt`
`Shirt`
`Pants`
`Jacket`
	`25.3`
	`70.3`
	`0`
	`57.5`
		`20`
		`60`
		`0`
		`40`
			`Available in black, blue, and red`
			`n/a`
			`Limited stock`
			`n/a`

Note: According to the [XML Schema Specification](), `integer` is derived from `decimal`, and both of them belong to the numeric data type. In our example, the rule will apply to both `ItemsInStock` and `Price` elements if you select `numeric` as a data type in the Node Functions window. If you select `decimal` as a data type, the rule will still apply to both elements. However, if you select `integer` as a data type, the rule will apply only to the `ItemsInStock` element.

Scenario 3: Block rules for specific nodes

In this scenario, we want to apply defaults for all string and numeric nodes, but we do not want to set a default for the `Price` item. To achieve the goal, reproduce the steps from *Scenario 2*, click the `Price` element, and clear the check box *Inherit rules from ancestors*. In the mapping below, the `Price` element no longer inherits rules from its parent, `Article`. Therefore, the icon appears next to the `Price` element (*see screenshot below*).

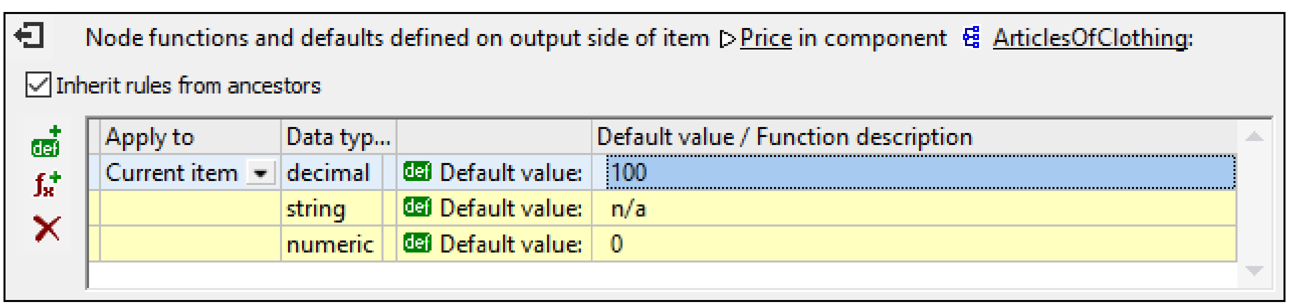


The output below shows that the empty value of the `Price` element, for which the rule has been blocked, has been mapped to `Field2` (*note the highlighted parts*).

T-Shirt Shirt Pants Jacket	25.3 70.3 57.5	20 60 0 40	Available in black, blue, and red n/a Limited stock n/a

Scenario 4: Override inherited rules

In this scenario, we want to set defaults for all string and numeric nodes; however, for the `Price` element exclusively, we want to set `100` as a default value. To achieve this, repeat the steps outlined in *Scenario 2*, click `Price` in the `ArticlesOfClothing` component, add a new default and type `100`, as shown in the screenshot below. The inherited rules have a yellow background.



The rule defined directly on a node has priority over the rules created at parent level. Therefore, the output now looks as follows (*note the highlighted part*):

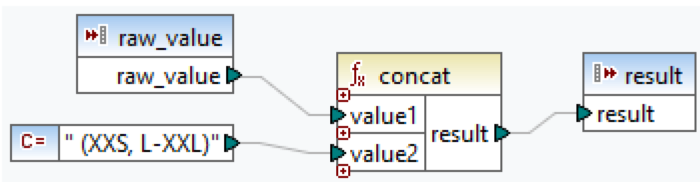
T-Shirt Shirt Pants Jacket	25.3 70.3 100 57.5	20 60 0 40	Available in black, blue, and red n/a Limited stock n/a

If you have defined more than one rules for the same node, the rule at the top of the grid will apply to the current node. You can swap the order of these rules by manually dragging the rules in the grid.

Scenario 5: Create a node function and apply it conditionally

This scenario will explain how to create a node function and apply it conditionally. We will use the **concat** function to add some text to nodes of type `string`. Follow the instructions below.

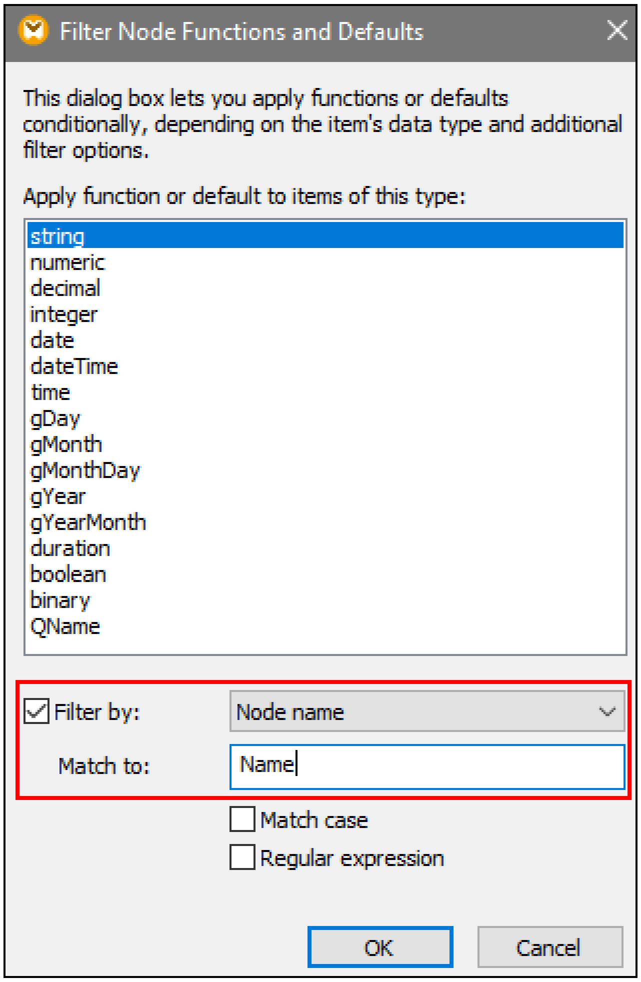
1. Right-click the `Article` element and select **Node Functions and Defaults | Output Node Functions and Defaults** from the context menu.
2. Add a node function by clicking the icon and create the function shown below.



At this stage, we have not filtered the nodes yet, and the output looks as follows (*note the highlighted parts*):

T-Shirt (XXS, L-XXL) Shirt (XXS, L-XXL) Pants (XXS, L-XXL) Jacket (XXS, L-XXL)	25.3 70.3 57.5	20 60 40	Available in black, blue, and red (XXS, L-XXL) Limited stock (XXS, L-XXL)

The output shows that our rule has been applied to both elements of type `string`: `Name` and `Description`. Now we want the rule to apply only to the `Name` element. To achieve this, go to the Node Functions window and click the ... button in the *Data type and filter* column. This opens the **Filter Node Functions and Defaults** dialog box, in which you can change data types and specify filtering options. Define the filter as shown in the screenshot below.



The output now looks as follows:

`T-Shirt (XXS, L-XXL)`
`Shirt (XXS, L-XXL)`
`Pants (XXS, L-XXL)`
`Jacket (XXS, L-XXL)`
	`25.3`
	`70.3`
	
	`57.5`
		`20`
		`60`
		
		`40`
			`Available in black, blue, and red`
			
			`Limited stock`

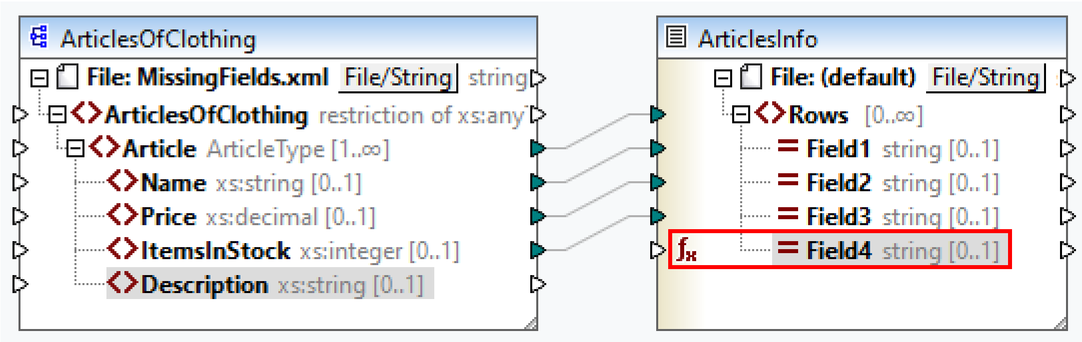
Filter by node type
This option allows you to narrow down the data type you have selected in the **Filter Node Functions and Defaults** dialog. For example, you have chosen to apply your node function to numeric nodes. Then you can specify a subtype (e.g., *Filter by/Node name:* `xs:decimal`).

Filter with regular expressions
For more advanced filtering options, you can use regular expressions (e.g., to match multiple node names or type names). For more information, see Regular Expressions 975. Note the following points:

- The anchors `^` and `$` are implicit and must not be entered in the *Match to* box.
- Case sensitivity can be specified in the *Match Case* check box. Therefore, the `i` flag is not supported.
- Matching on multiple lines is not meaningful for node filtering. Therefore, the `m` flag is not supported.

Scenario 6: Create defaults for unconnected nodes

MapForce allows you to apply defaults to unconnected nodes but only if the parent node and/or a sibling are connected. Consider the following example:



In the mapping above, we have defined a default value for the `Field4` node in the `ArticlesInfo` component. At this stage, `Field4` does not have an input connection. The parent and sibling nodes are connected. If `Field4` remains unconnected, the default value will overwrite all the `Description` values in the output (*see below*). If you connect `Description` and `Field4`, the default value will apply only to empty sequences.

T-Shirt	25.3	20	n/a
Shirt	70.3	60	n/a
Pants			n/a
Jacket	57.5	40	n/a

Note: Defaults do not apply to unconnected nodes in JSON components.

6.3.3 Node Metadata in Node Functions

There might be cases in which you want a node function to do something based on *node metadata*. By node metadata, we understand miscellaneous information about the node on which the function applies (e.g., a node name, value length, and precision), which is stored in the schema. The table below lists all possible metadata parameters that you can use in a node function. Note that some metadata parameters apply only to specific data types. MapForce will display a warning when you attempt to use metadata that is incompatible with the current node.

Metadata parameters	Description
`node_name`	Supplies the name of the current node as it is shown in the component. This metadata is supported by all nodes. In XML files, `node_name` refers to the name of the current element or attribute. In CSV components,

Metadata parameters	Description
	node_name refers to the name of a CSV field. In databases, node_name refers to the name of a table column.
node_annotation	Displays annotation text next to the node value in the output. This metadata is supported by all nodes.
node_minLength	Supplies the value of the minLength facet of the node's data type. Applicable to XML and text nodes with appropriate types. For more information about length-constraining facets, see the XML Schema Recommendation.
node_maxLength	Supplies the value of the maxLength facet of the node's data type. Applicable to XML and text nodes with appropriate types. For more information about length-constraining facets, see the XML Schema Recommendation.
node_totalDigits	Supplies the value of the totalDigits facet of the node's data type. Applicable to XML nodes with appropriate types. For details about the totalDigits facet, see the XML Schema Recommendation.
node_fractionDigits	Supplies the value of the fractionDigits facet of the node's data type. Applicable to XML nodes with appropriate types. For details about the fractionDigits facet, see the XML Schema Recommendation.
node_length	Supplies the length (in bytes) of a number. Applicable to database fields with appropriate types. To find out more about length, see the Microsoft documentation.
node_precision	Supplies the number of all digits of a number. Applicable to database fields with appropriate types. For more information, see the Microsoft documentation.
node_scale	Supplies the number of digits to the right of the decimal point of a number. Applicable to database fields with appropriate types. For more information, see the Microsoft documentation.

Note: The data type of a matadata parameter must coincide with the data type of the relevant node. Otherwise, the function execution will fail.

Add metadata to a node function

To add a metadata parameter to a node function, you need to create a node function 915 and then click the **Add Node Specifics** option, which is located under the Node Functions window. This will open the **Insert Input with Node Specific** dialog box, in which you can select a relevant metadata parameter. Instead of the **Add Node Specifics** option, you can also choose one of the following options:

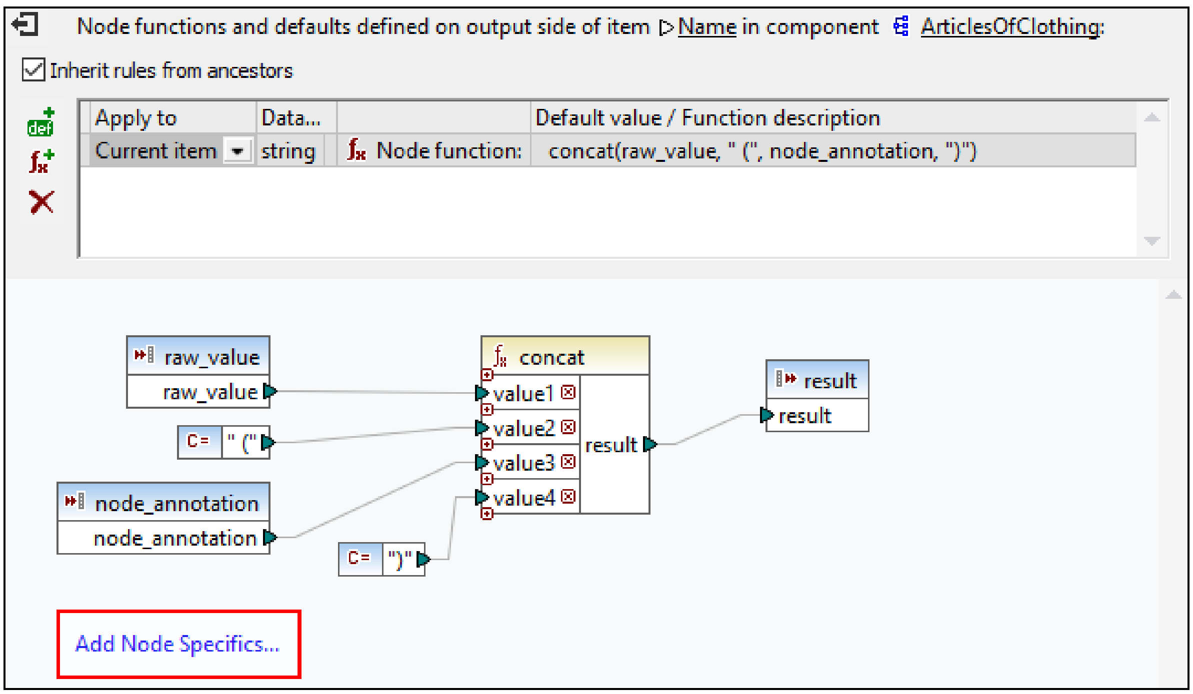
- Right-clicking an empty area in the mapping and selecting **Insert Input** from the context menu.
- Clicking the toolbar command.
- Clicking **Insert Input** in the **Function** menu.

When matadata is not supported by a node
When a metadata parameter is not supported by a node, you can instruct MapForce to return an empty sequence or not to apply the node function (both options are available in the **Insert Input with Node Specific** dialog box). The empty sequence must be handled; otherwise, the node function might not return a value at all. You would typically need to use sequence functions (e.g., `substitute-missing`[1056] and `exists`[1032]) or other component types to process the empty sequence further.

Example

The example below explains how to supply annotation text to a node function. For this example, you will need the following mapping: **`<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\MissingFields.mfd`**. In **`MissingFields.xsd`** (on which **`MissingFields.xml`** is based), we have added the following annotation for the `Name` element: `Article name from catalog Spring 2022`. Our goal is to supply this annotation to a node function and to see the annotation in the output. Follow the instructions below.

1. Create a node function[915] for the `Name` element in the `ArticlesOfClothing` component.
2. Replicate the function shown below. To add a `node_annotation` parameter, click **Add Node Specifics** (*see red rectangle below*) and select this parameter from the list. To find out how to add and manipulate functions, see Functions Basics[905].



Note: By default, deeply nested structures are not fully scanned so as to preserve memory and improve the user experience. If the component where you apply the node function has such deeply nested structures, you can expand the relevant nodes in the mapping to make MapForce aware of them. In this case, MapForce will take the expanded nodes into account when you add a new metadata parameter, and the warning may disappear. The node function must be connected in order to take effect; expanding unconnected items is not relevant.

Output
The listing below shows that the annotation has been added to the values of the `Name` elements.

```
T-Shirt (Article name from catalog Spring 2022)   25.5  20    Available in black, blue,
and red
Shirt (Article name from catalog Spring 2022)     70.3  60

Pants (Article name from catalog Spring 2022)                 Limited
stock
Jacket (Article name from catalog Spring 2022)    57.5  40
```

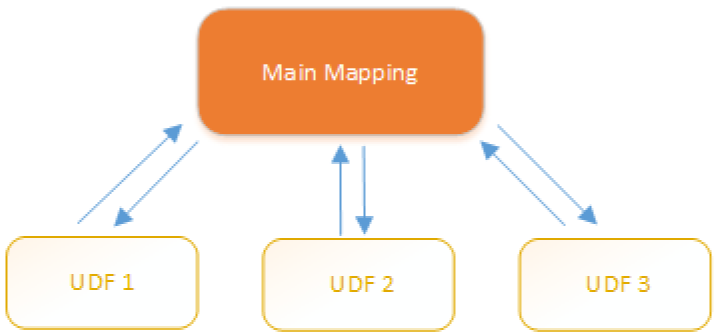
6.4 User-Defined Functions

User-Defined Functions (UDFs for short) are custom functions that are defined once and can be reused multiple times within the same mapping or across multiple mappings. UDFs are like mini-mappings themselves: They typically consist of one or more input parameters, some intermediary components to process data, and an output to return data to the caller. The caller is the main mapping or another UDF.

Advantages of UDFs

UDFs have the following advantages:

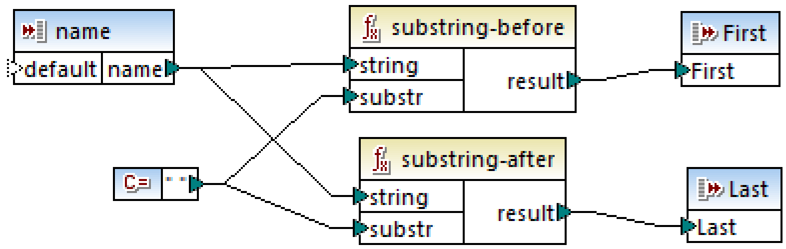
- UDFs are reusable within one mapping or across multiple mappings.
- UDFs can make your mapping easier to read: For example, you can package parts of the mapping into smaller components and abstract away the implementation details. The diagram below illustrates this principle.



- UDFs are flexible functions that enable you to process strings, numbers, dates, and other data in a custom way that extends the built-in MapForce functions. For example, you might want to concatenate or split text in a particular way, perform some advanced calculations, manipulate dates and times, and so on.
- Another common use of UDFs is to look up a field in an XML file, database or some other data format supported by your MapForce edition and present this data in a convenient way. For details, see Look-up Implementation 939.
- UDFs can be called recursively (i.e., the UDF calls itself). This requires that the UDF be defined as a regular (not inline) function 930. Recursive UDFs 936 can fulfill various advanced mapping requirements, such as iterating over data structures having a depth of *N* children, where *N* is not known in advance.

Example

Below is an example of a simple UDF that splits a string into two separate strings. This UDF is part of a larger mapping called **`MapForceExamples\ContactsFromPO.mfd`**. The UDF takes the name as a parameter (e.g., `Helen Smith`), applies the built-in functions **`substring-before`** and **`substring-after`**, and then returns two values: `Helen` and `Smith`.



In this section

This section explains how to work with UDFs and is organized into the following topics:

6.4.1 UDF Basics

This topic explains how to create, import, edit, copy-paste, and delete user-defined functions (UDFs for short).

Create a UDF

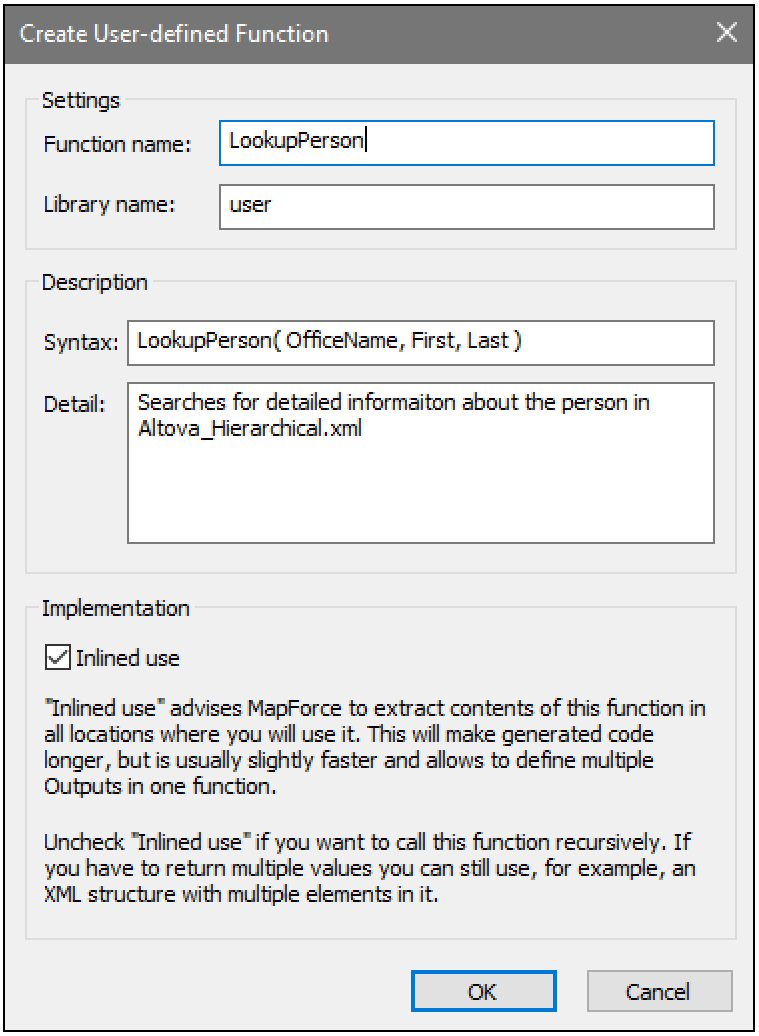
This subsection explains how to create a UDF from scratch and from already existing components. The minimum requirement is one output component to which some data is connected. For input parameters, a function can have zero, one or more inputs. The input and output parameters can be of simple type (e.g., a string) or complex type (a structure). For more information about simple and complex parameters, see UDF Parameters 933.

UDF from scratch

To create a UDF from scratch, follow the instructions below:

1. Select **Function | Create User-Defined Function**. Alternatively, click the toolbar button.

2. Enter the relevant information into the **Create User-defined Function** dialog (*see screenshot below*).



The available options are listed below.

- *Function Name:* Mandatory field. For the name of your UDF, you can use the following characters: alphanumeric characters (a-z, A-Z, 0-9), an underscore (_), a hyphen/dash (-), and a colon (:).
- *Library Name:* Mandatory field. This is the name of a function library (in the Libraries window[33]) in which your function will be saved. If you do not specify the name of a library, the function will be placed into a default library called `user`.
- *Syntax:* Optional field. Enter some text that concisely describes the syntax of the function (e.g., expected parameters). This text will be displayed next to the function in the **Libraries** window and does not affect the implementation of the function.
- *Detail:* Optional field. This description will be displayed when you move the cursor over the function in the **Libraries** window or in other contexts.
- *Inlined use:* Select this check box if the function should be created as inline. For more information, see *Regular vs. Inline UDFs* below.

3. Click **OK**. The function becomes immediately visible in the **Libraries** window under the library name specified above. The mapping window is now redrawn to allow you to create a new function (this is a standalone mapping referred to as *the function's mapping*). The function's mapping includes an output component by default.

4. Add all the required components to the function's mapping. You can do this in the same way as for a standard mapping.

To use the UDF in a mapping, drag the UDF from the **Libraries** window onto the main mapping area. See also *Call and import UDFs* below.

UDF from existing components
To create a UDF from existing components, take the steps below:

1. Select the relevant components on the mapping by making a rectangular selection with the mouse. You can also select multiple components by clicking each one while holding the **Ctrl** key pressed.
2. Select the menu command **Function | Create User-Defined Function from Selection**. Alternatively, click the toolbar button.
3. Follow Steps 2-4 from *UDF From Scratch*.

Regular vs. Inline UDFs
There are two types of UDFs: *regular* and *inline*. You can specify the type of your UDF when you create it. Inline and regular functions behave differently in terms of code generation, recursiveness, and the ability to have multiple output parameters. The table below summarizes the main differences between regular and inline UDFs.

Inline functions (dashed border)	Regular functions (solid border)
With inline functions, the UDF code is inserted at all locations where the function is called. If the UDF is called several times, the generated program code would be significantly longer.	The code for the UDF is generated once, and inputs to it are passed as parameter values. If the UDF is called several times, the UDF is evaluated each time with the corresponding parameter values.
Inline functions can have multiple outputs and thus return multiple values.	Regular functions can have only one output. To return multiple values, you can declare the output to be of complex type (e.g., an XML structure), which would allow you to pass multiple values to the caller.
Inline functions cannot be called recursively.	Regular functions can be called recursively.
Inline functions do not support setting a priority context [1288] on a parameter.	Regular functions support setting a priority context on a parameter.

Note: Switching a UDF from inline to regular, and vice versa, may affect the mapping context [1277], and this may cause the mapping to produce a different result.

Call and import UDFs

After you have created a UDF, you can call it from the same mapping where you created it or from any other mapping.

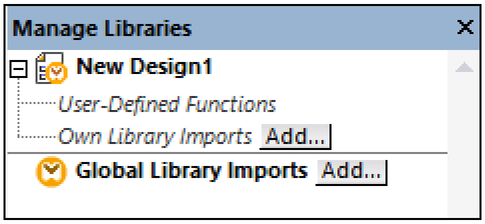
Call UDF from the same mapping
To call a UDF from the same mapping, take steps below:

1. Find the relevant function in the **Libraries** window under the library that you specified when you created the function. To do that, start typing the name in the **Libraries** window.

2. Drag the function from the **Libraries** window into the mapping. You can now connect all the required parameters to the function.

Import UDF from a different mapping
To import a UDF from another mapping, follow the instructions below:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[33]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for the **`.mfd`** file that contains the UDF and click **Open**. A message box will inform you that a new library has been added and can be accessed in the **Libraries** window.

You can now use any of the imported functions in the current mapping by dragging them from the **Libraries** window into the mapping. For more information about viewing and organizing function libraries, see Manage Function Libraries[908].

Mapping with credentials (Enterprise Edition)
If the imported **`.mfd`** file contains credentials, these are shown as imported (with a yellow background) in Credentials Manager. By default, imported credentials are not saved with the mapping, but you can optionally create a local copy and save them in the mapping. See Credentials in UDFs[1301].

Edit UDFs

To edit a UDF, take the steps below:

1. Open the mapping that contains a UDF.
2. Double-click the title bar of the UDF in the mapping to see the function's contents where you can add, edit, or remove components as required.
3. To change the function's properties (e.g., the name or description), right-click an empty area in the mapping and select **Function Settings** from the context menu. Alternatively, click the toolbar button.

You can also edit a function by double-clicking its name in the **Libraries** window. However, only functions in the currently active document can be opened this way. Double-clicking a UDF that was created in another mapping opens that mapping in a new window. If you edit or delete a UDF that was imported into multiple mappings, all these mappings will be affected by the change.

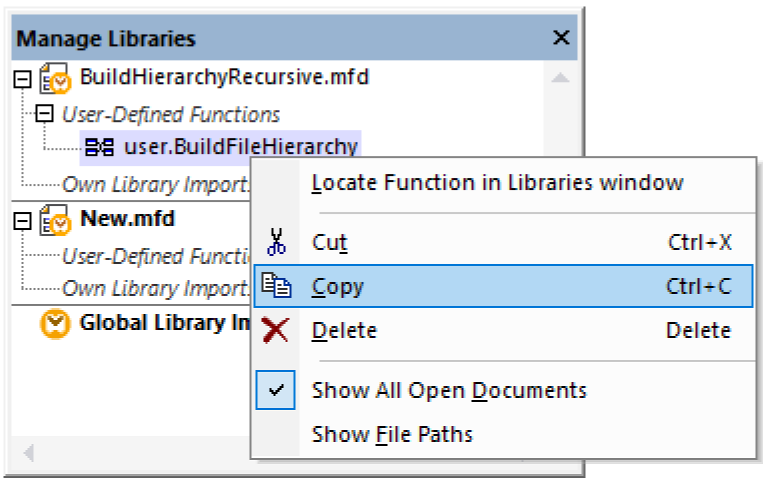
To go back to the main mapping, click the button in the top-left corner of the mapping window.

MapForce allows you to navigate through different mappings and UDFs by using the and toolbar buttons. The corresponding keyboard shortcuts for these buttons are **Alt+Left** and **Alt+Right**, respectively.

Copy-paste UDFs

To copy a UDF and paste it into another mapping, follow the steps below:

1. Open the Manage Libraries window[908].
2. Right-click inside an empty area in the **Libraries** window and select the option **Show All Open Documents**.
3. Open both the source and destination mappings. Make sure that both mappings are saved to disk. This ensures correct resolution of paths. See also Copy-Paste and Relative Paths[85].
4. Right-click the relevant UDF from the source mapping in the **Manage Libraries** window and select **Copy** from the context menu (*see screenshot below*) or press **Ctrl+C**. Leave the **Manage Libraries** window open.



5. Switch to the destination mapping (and the **Manage Libraries** window will change accordingly), right-click *User-Defined Functions*, and select **Paste** from the context menu.

Delete UDFs

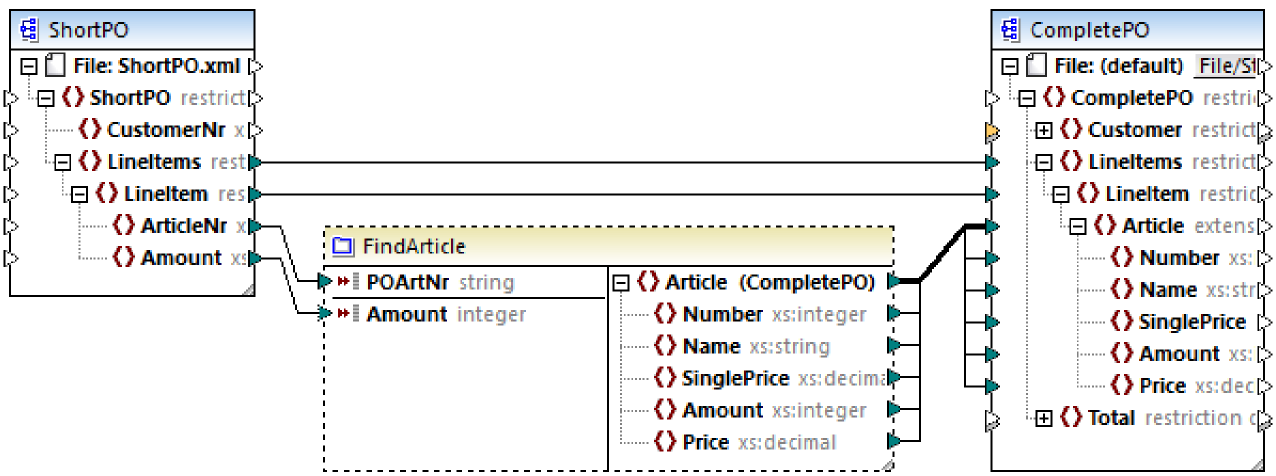
To delete a UDF, take the steps below:

1. Double-click the title bar of the UDF in the mapping.
2. Click the button in the top-right corner of the Mapping window.
3. If the function is used in the currently open mapping, MapForce will ask whether you want to replace all instances with internal components. Click **Yes** if you want to delete the function and replace all instances where it is called with the function's components. This lets you keep the main mapping valid even if the function is deleted. However, if the deleted function is used in any other external mappings, those will not be valid. Click **No** if you want to delete the function and all its internal components permanently. In this case, all the mappings where the function is used will not be valid.

6.4.2 UDF Parameters

When you create a UDF, you must specify what input parameters it should take (if any) and what output it should return. While input parameters are sometimes not necessary, an output parameter is mandatory in all cases. Function parameters can be of simple type (e.g., `string` or `integer`) or a complex structure 933. For example, the `FindArticle` UDF illustrated below has two input and one output parameters:

- `POArtNr` is an input parameter of type `string`.
- `Amount` is an input parameter of type `integer`.
- `CompletePO` is an output parameter that has a complex XML structure.



Parameter order
When a UDF has multiple input or output parameters, you can change the order in which parameters will appear to the callers of this function. The order of parameters in the function's mapping (starting from the top) dictates the order in which they appear to the callers of this function.

Important

- Input and output parameters are sorted by their position from top to bottom. Therefore, if you move the `input3` parameter to the top in the function's mapping, it will become the first parameter of this function.
- If two parameters have the same vertical position, the leftmost takes precedence.
- In the unusual case that two parameters have exactly the same position, the internal component ID is automatically used.

Complex-type structures

The structures on which a parameter in a UDF can be based are summarized in the list below.

MapForce Basic Edition
- XML Schema Structure

MapForce Professional Edition
- XML Schema Structure
- Database Structure

MapForce Enterprise Edition

- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure
- JSON Schema Structure

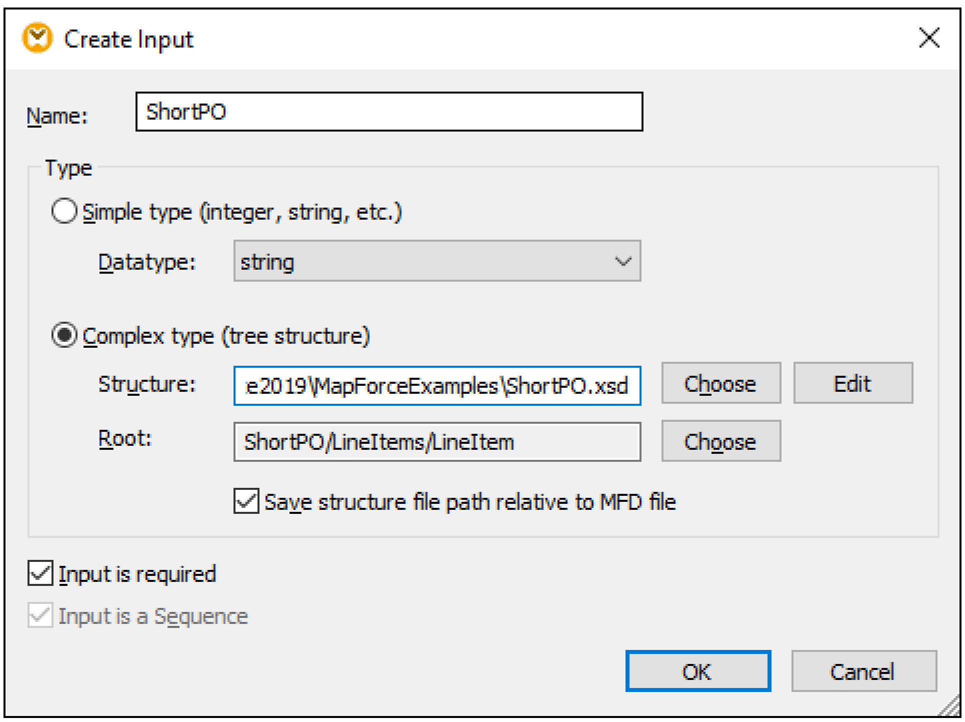
UDFs based on database structures (Professional and Enterprise editions)

MapForce allows you to create DB-based UDF parameters with a tree of related tables. The tree of related tables represents an in-memory structure that has no connection to the database at runtime. This also means that there is no automatic handling of foreign keys and no table actions in parameters or variables.

Add Parameters

To add an input or output parameter, take the following steps:

1. Create a UDF [928] or open an existing one [930].
2. Run the menu command **Function | Insert Input** or **Function | Insert Output** (*see screenshot below*). Alternatively, click (**Insert Input**) or (**Insert Output**) in the toolbar.



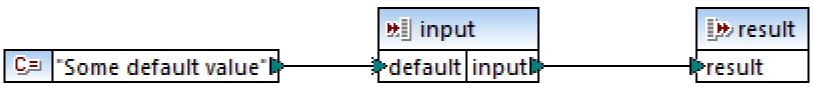
3. Choose whether input or output parameters should be of simple or complex type (*see dialog box above*). See the list of available complex structures above. For example, to create a parameter that is a complex XML type, click **Choose** next to *Structure* and browse for the XML schema that describes the required structure.

If the function's mapping already includes XML schemas, they become available for selection as structures. Otherwise, you can select a new schema that will provide the structure of the parameter. The same is true for databases and other complex structures if they are supported by your MapForce edition. With XML structures, it is possible to select the root element for your structure if the XML schema allows it. To specify the root element, click **Choose** next to *Root* and select the root item from the dialog box that opens.

If selected, the check box *Save structure file path relative to MFD file* will change the absolute path of the structure into a path relative to the current mapping, when you save the mapping. For more information, see Relative and Absolute Paths 85. The check boxes *Input is required* and *Input is a Sequence* are explained below.

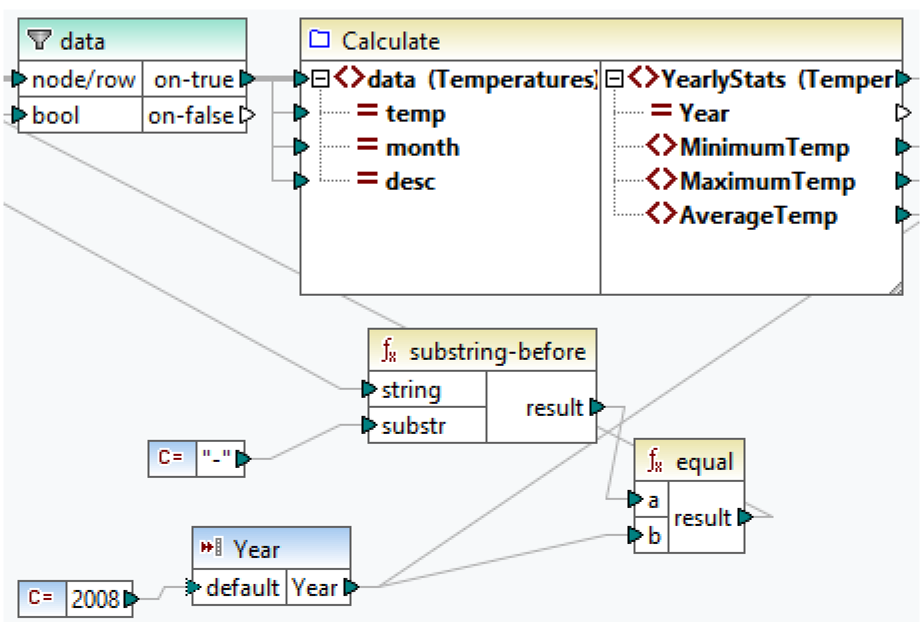
Input is required
To make a parameter mandatory in a UDF, select the *Input is required* check box (*see dialog box above*). If you clear the *Input is required* check box, the parameter will become optional and have a dashed border in the mapping.

You can also specify a default parameter value by connecting it to the `default` input of a parameter (*see example below*). The default value will apply only if there is no other value. If the optional parameter receives a value when the function is called, that value takes precedence over the default.

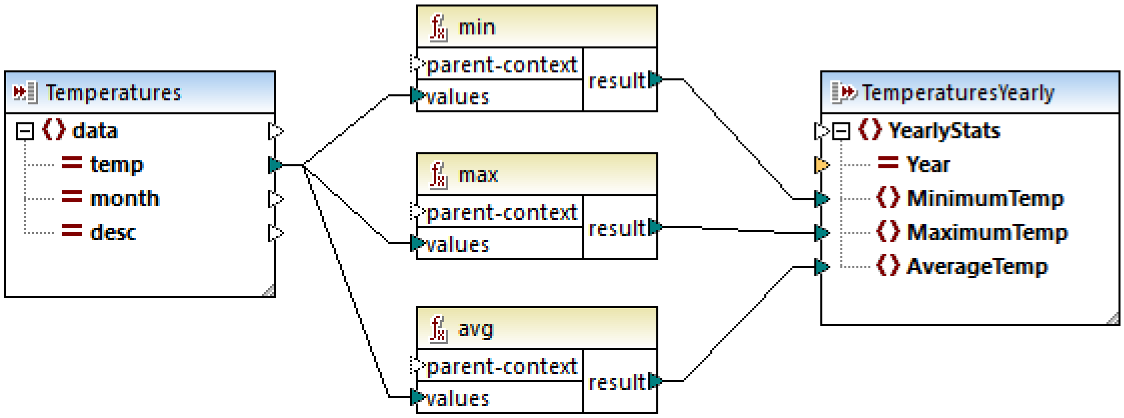


Input is a sequence
You can optionally define whether a function's parameter should be treated as a single value (default option) or as a sequence. A sequence is a range of zero or more values. A sequence might be useful when your UDF expects input data as a sequence in order to calculate values in that sequence, for example, by calling functions such as **avg**, **min**, **max**. To treat the input of the parameter as a sequence, select the *Input is sequence* check box. Note that this check box is enabled only if the UDF is regular 930.

The usage of a sequence is illustrated in the following mapping: **MapForceExamples\InputIsSequence.mfd**. In the extract of this mapping (*see screenshot below*), the `data` filter is connected to the UDF called `Calculate`. The filter's output is a sequence of items. Therefore, the input parameter of the function is set to be a sequence.



Illustrated below is the implementation of the `Calculate` function that aggregates all the sequence values: It runs the **`avg`**, **`min`**, and **`max`** functions on the input sequence. To see the internal structure of the `Calculate` function, double-click the header of the `Calculate` component in the mapping above.



As a rule of thumb, the input data (sequence or non-sequence) determines how often the function is called:

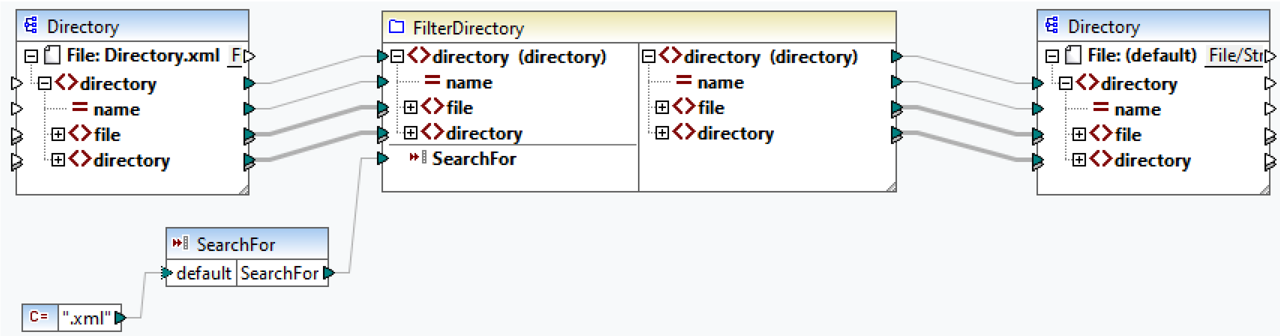
- When input data is connected to a *sequence* parameter, the UDF is called only *once*, and the complete sequence is passed into the UDF.
- When input data is connected to a *non-sequence* parameter, the UDF is called *once for each single item* in the sequence.
- If you connect an empty sequence to a non-sequence parameter, the function is not called at all. This can happen if the source structure has optional items or when a filter condition returns no matching items. To avoid this, use the `substitute-missing` 1056 function before the function input to ensure that the sequence is never empty. Alternatively, set the parameter to a sequence and add handling for the empty sequence inside the function.

The *Output is a Sequence* check box may be required for output parameters, too. When a function passes a sequence of multiple values to its output component, and the output component is not set to sequence, the function will return only the first item in the sequence.

6.4.3 Recursive UDFs

This topic explains how to search for data in a source XML file with the help of a recursive UDF. To test the recursive UDF, you will need the following mapping: **`MapForceExamples\RecursiveDirectoryFilter.mfd`**. In the mapping below, the `FilterDirectory` UDF receives data from the source file `Directory.xml` and the simple input component `SearchFor` that supplies the `.xml` extension. After the data has been processed by the UDF, it is mapped to the target file.

The main mapping (*see screenshot below*) describes the general mapping layout. The way the UDF processes the data is defined separately, in the function's mapping (*See UDF Implementation below*).



Goal
Our goal is to list files with a **`.xml`** extension in the output while preserving the entire directory structure. The subsections below explain the mapping process in detail.

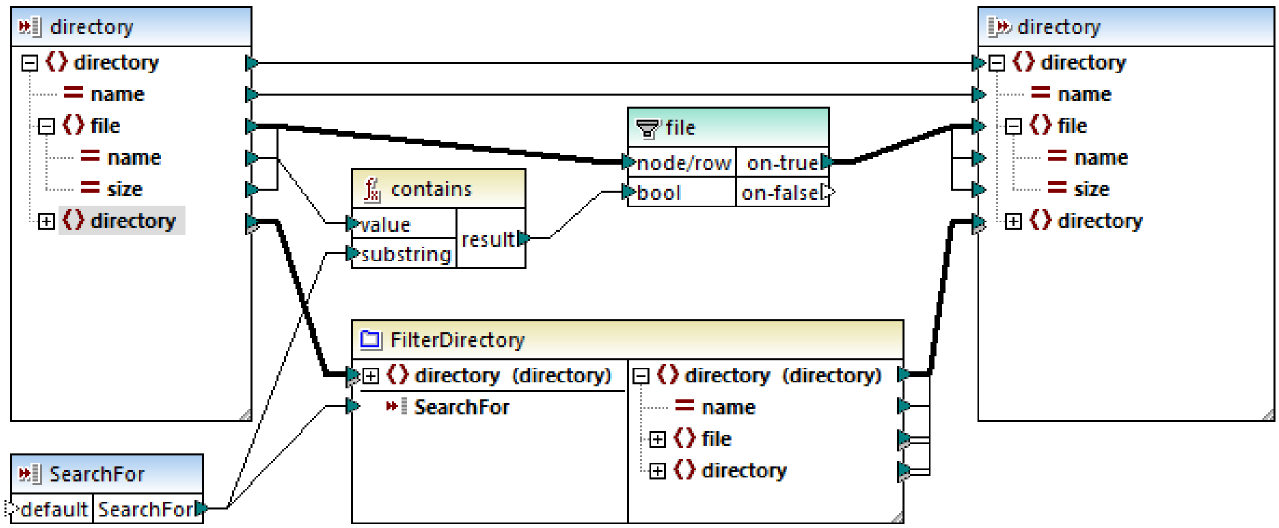
Source file

Below is an extract from the source XML file (**`Directory.xml`**) that contains information about files and directories. Note that the files in this listing have different extensions (e.g., **`.xml`**, **`.dtd`**, **`.sps`**). According to the schema (**`Directory.xsd`**), the `directory` element can have `file` children and `directory` children. All `directory` elements are recursive.

```
<directory name="ExampleSite">
	<file name="blocks.sps" size="7473"/>
	<file name="blocks.xml" size="670"/>
	<file name="block_file.xml" size="992"/>
	<file name="block_schema.xml" size="1170"/>
	<file name="contact.xml" size="453"/>
	<file name="dictionaries.xml" size="206"/>
	<file name="examplesite.dtd" size="230"/>
	<file name="examplesite.spp" size="1270"/>
	<file name="examplesite.sps" size="20968"/>
	...
	<directory name="output">
		<file name="examplesite1.css" size="3174"/>
		<directory name="images">
			<file name="blank.gif" size="88"/>
			<file name="block_file.gif" size="13179"/>
			<file name="block_schema.gif" size="9211"/>
			<file name="nav_file.gif" size="60868"/>
			<file name="nav_schema.gif" size="6002"/>
		</directory>
	</directory>
</directory>
```

UDF implementation

To see the internal implementation of the UDF, double-click its header in the main mapping. The UDF is recursive, that is, it includes a call to itself. Because it is connected to the recursive element `directory`, this function will be called as many times as there are nested `directory` elements in the source XML instance. To support recursive calls, the function must be regular 930.



The implementation of the UDF consists of two parts: (i) defining the files and (ii) defining the directory to be searched.

Defining files
The UDF processes the files as follows: The **`contains`** function checks whether the first string (the file name) contains the substring `.xml` (supplied by the simple input component `SearchFor`). If the function returns `true`, the file name with a `.xml` extension is written to the output.

Processing child directories
Child directories of the current directory are sent as input to the current UDF. The UDF thus iterates through all `directory` elements and check whether files with the **`.xml`** extension exist.

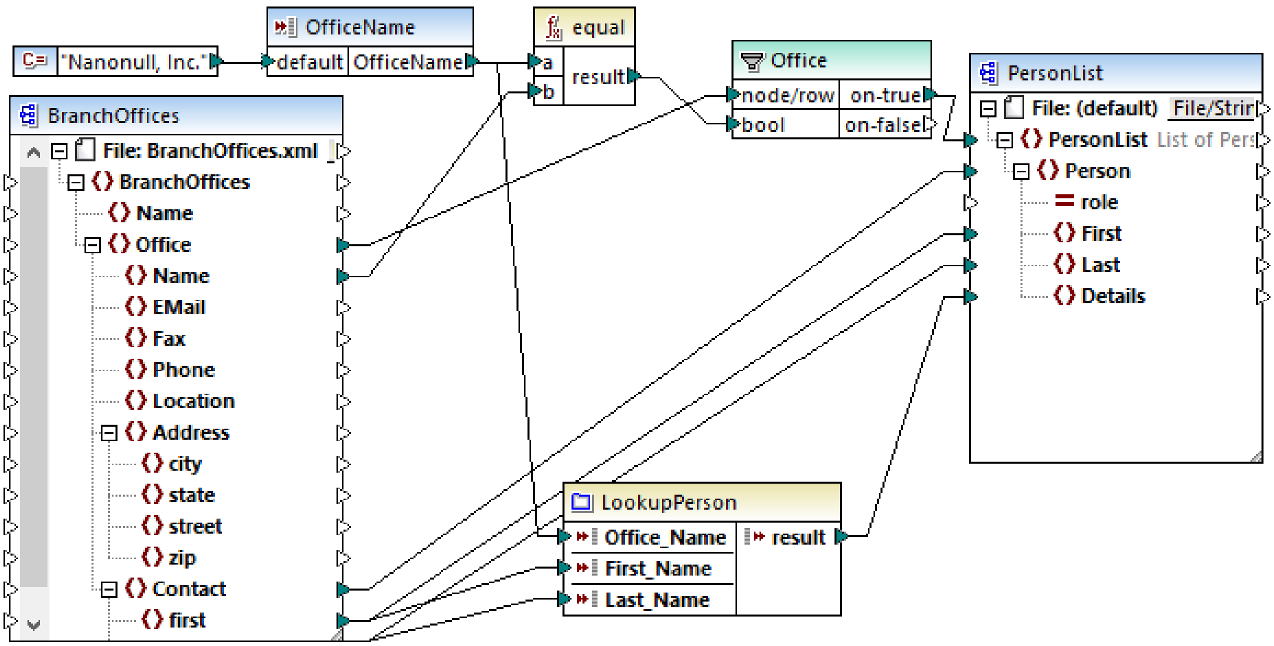
Output

When you click the **Output** pane, MapForce will display only files with the **`.xml`** extension (*see extract below*).

```
<directory name="ExampleSite">
        <file name="blocks.xml" size="670"/>
        <file name="block_file.xml" size="992"/>
        <file name="block_schema.xml" size="1170"/>
        <file name="contact.xml" size="453"/>
        ...
        <directory name="output">
                <directory name="images"/>
        </directory>
</directory>
```


6.4.4 Look-up Implementation

This topic explains how to look up data about employees and present this information in a suitable way. To test the look-up implementation, you will need the following mapping: **MapForceExamples\PersonListByBranchOffice.mfd**.



Goals
Our goals are as follows:

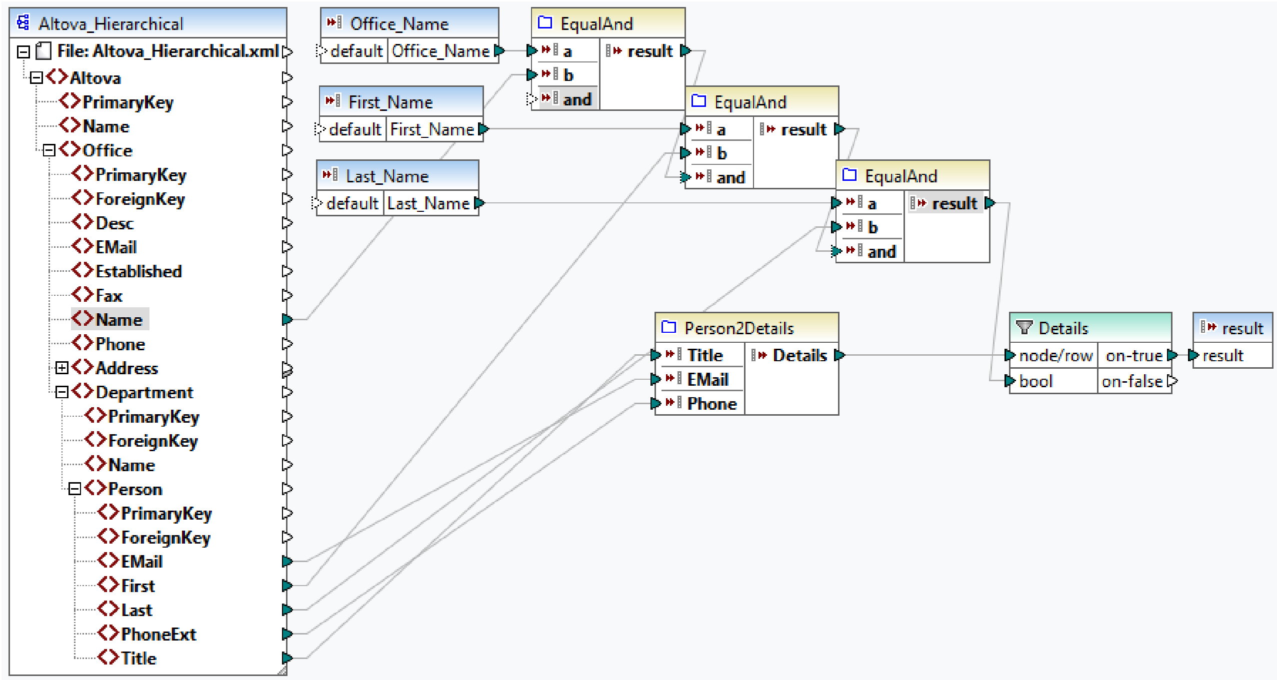
- To look up data about each employee (their phone extension, email address, and title) in a separate XML file.
- To present this data as a comma-separated list and map this list to the `Details` element of the target XML file.
- To extract information about employees only from one branch office called Nanonull, Inc.

To achieve these goals, we have designed our mapping in the following way:

- To filter only employees from Nanonull, Inc., the mapping uses the `Office` filter.
- To look up information about employees in a different XML file, the mapping calls the `LookupPerson` UDF. The implementation of this UDF is described in the subsection below.
- To process employee data, the `LookupPerson` function internally calls other functions that retrieve and concatenate information about each employee. All these operations are in the function's mapping and not visible in the main mapping. The `LookupPerson` function then maps the employee data to the `Details` element in `PersonList`.

LookupPerson implementation

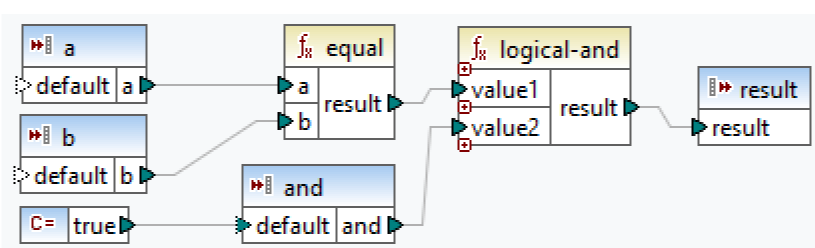
The look-up functionality is provided by the `LookupPerson` function, whose definition is illustrated below. To see the internal implementation of the UDF, double-click its header in the main mapping.



The UDF is defined as follows:

- The data is retrieved from the XML file **`Altova_Hierarchical.xml`**: (i) the name of the office and first and last names of employees, which are used to select employees only from Nanonull, Inc., and (ii) the email, title, and phone extension that are concatenated into one string. The definitions of the `EqualAnd` and `Person2Detail` functions are described below.
- The UDF also has three input parameters that provide the look-up values `Office_Name`, `First_Name`, and `Last_Name`. The value of the `Office_Name` parameter is retrieved from the `OfficeName` input from the main mapping, and the values of `First_Name` and `Last_Name` are supplied by the `BranchOffices` component from the main mapping.
- The value of the `EqaulAnd` function (`true` or `false`) is passed to the `Details` filter each time a new employee's details (title, email, phone) are processed. When the `Details` filter gets the value `true`, the look-up operation is successful and the employee's details are retrieved and returned to the main mapping. Otherwise, the next item in the context is examined, and this procedure continues until the loop finishes.

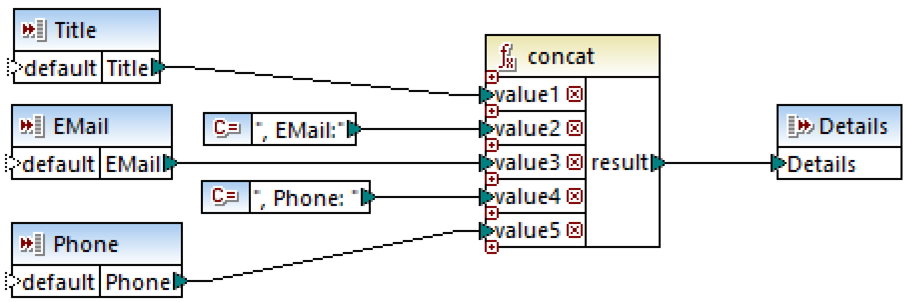
EqualAnd implementation
The `EqualAnd` function (*see below*) is a separate UDF defined inside the `LookupPerson` UDF. To see the internal structure of the `EqualAnd` UDF, double-click the function's header.



The `EqualAnd` UDF first checks whether `a` equals `b`; if the result is `true`, it is passed as the first parameter of the **`logical-and`** function. If both values are true in the **`logical-and`** function, the result is also true and is passed on to the next `EqualAnd` function. The result of the third `EqaulAnd` function (*see LookupPerson UDF above*) is passed on to the `Details` filter.

Person2Detail implementation

The `Person2Details` UDF is another function inside the `LookupPerson` UDF. The `Person2Details` UDF (*see below*) concatenates three values (retrieved from **`Altova_Hierarchical.xml`**) and two text constants.



Output

When you click the **Output** pane, MapForce will display first and last names, and details of employees only from Nanonull, Inc (*see extract below*).

```
<PersonList>
       <Person>
              <First>Vernon</First>
              <Last>Callaby</Last>
              <Details>Office Manager, EMail:v.callaby@nanonull.com, Phone:
582</Details>
       </Person>
       <Person>
              <First>Frank</First>
              <Last>Further</Last>
              <Details>Accounts Receivable, EMail:f.further@nanonull.com, Phone:
471</Details>
       </Person>
       ...
</PersonList>
```

6.5 Custom Functions

This section explains how to import custom XSLT[942], XQuery[949], Java and .NET[953] functions. The section also shows how to reference C#, C++ and Java libraries manually[960].

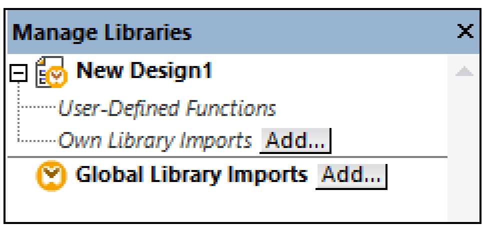
6.5.1 Import Custom XSLT Functions

You can extend the XSLT 1.0, XSLT 2.0, and XSLT 3.0 function libraries available in MapForce with your own custom functions, provided that your custom functions return simple types.

> Only custom functions that return simple data types (for example, strings) are supported.

To import functions from an XSLT file:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[33]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for the .xsl file that contains the functions, and click **Open**. A message box appears informing you that a new library has been added.

Imported XSLT files appear as libraries in the Libraries window, and display all named templates as functions below the library name. If you do not see the imported library, ensure you have selected XSLT as a transformation language[27]. See also Managing Function Libraries[908].

Note the following:

- To be eligible for import into MapForce, functions must be declared as named templates conforming to the XSLT specification in the XSLT file. You can also import functions that occur in an XSLT 2.0 document in the form `<xsl:function name="MyFunction">`. If the imported XSLT file imports or includes other XSLT files, then these XSLT files and functions will be imported as well.
- The mappable input connectors of imported custom functions depends on the number of parameters used in the template call; optional parameters are also supported.
- Namespaces are supported.

- If you make updates to XSLT files that you have already imported into MapForce, changes are detected automatically and MapForce prompts you to reload the files.
- When writing named templates, make sure that the XPath statements used in the template are bound to the correct namespace(s). To see the namespace bindings of the mapping, preview the generated XSLT code 107.

Data types in XPath 2.0

If your XML document references an XML Schema and is valid according to it, you must explicitly construct or cast datatypes that are not implicitly converted to the required datatype by an operation.

In the XPath 2.0 Data Model used by the Altova XSLT 2.0 Engine, all **atomized** node values from the XML document are assigned the `xs:untypedAtomic` datatype. The `xs:untypedAtomic` type works well with implicit type conversions.

For example,

- the expression `xs:untypedAtomic("1") + 1` results in a value of 2 because the `xdt:untypedAtomic` value is *implicitly* promoted to `xs:double` by the addition operator.
- Arithmetic operators implicitly promote operands to `xs:double`.
- Value comparison operators promote operands to `xs:string` before comparing.

See also:

Example: Adding Custom XSLT Functions 943
Example: Summing Node Values 946
XSLT 1.0 engine implementation 1894
XSLT 2.0 engine implementation 1894

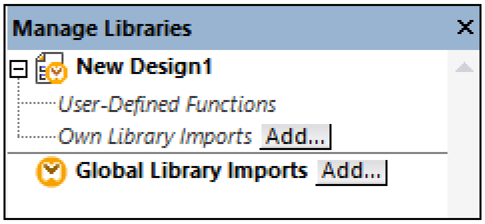
6.5.1.1 Example: Adding Custom XSLT Functions

This example illustrates how to import custom XSLT 1.0 functions into MapForce. The files needed for this example are available in the following folder: **`C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples`**.

- `Name-splitter.xslt`. This XSLT file defines a named template called **tokenize** with a single parameter `string`. The template works through an input string and separates capitalized characters with a space for each occurrence.
- `Name-splitter.xml` (the source XML instance file to be processed)
- `Customers.xsd` (the source XML schema)
- `CompletePO.xsd` (the target XML schema)

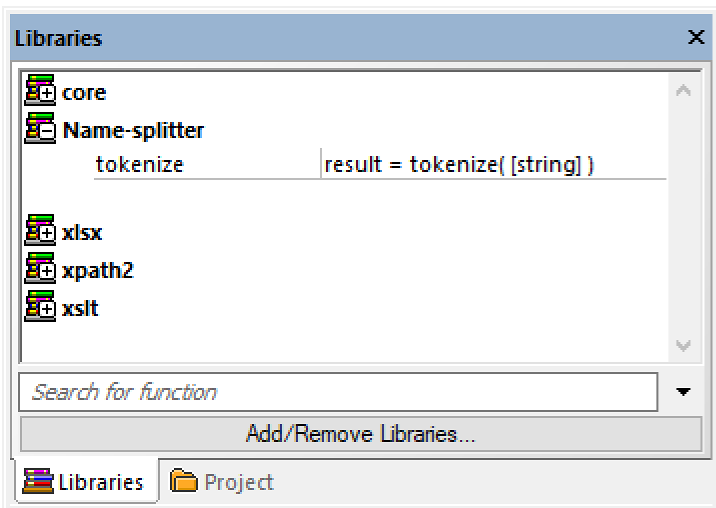
To add a custom XSLT function:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window 33. The **Manage Libraries** window opens (*see screenshot below*).



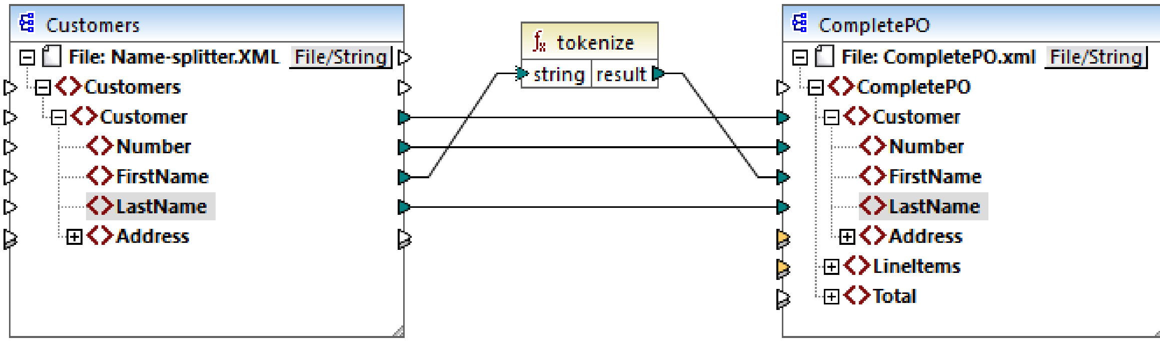
2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for the `.xsl` or `.xslt` file that contains the named template you want to act as a function, in this case `Name-splitter.xslt`, and click **Open**. A message box appears informing you that a new library has been added, and the XSLT file name appears in the **Libraries** window, along with the functions defined as named templates (in this example, **Name-splitter** with the **`tokenize`** function).



To use the XSLT function in your mapping:

1. Drag the **`tokenize`** function into the Mapping window and map the items as show below.



2. Click the **XSLT** tab to see the generated XSLT code.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--...-->
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xs="http
    <xsl:include href="file:///C:/Users/altova/Documents/Altova/MapForce2020/MapForceExamples
    <xsl:output method="xml" encoding="UTF-8" byte-order-mark="no" indent="yes"/>
    <xsl:template match="/">
        <CompletePO>
            <xsl:attribute name="xsi:noNamespaceSchemaLocation" namespace="http://www.w3.org/
CompletePO.xsd'"/>
            <xsl:for-each select="Customers/Customer">
                <Customer>
                    <Number>
                        <xsl:sequence select="xs:string(xs:integer(fn:string(Number)))"/>
                    </Number>
                    <FirstName>
                        <xsl:call-template name="tokenize">
                            <xsl:with-param name="string" select="FirstName" as="item()"/>
                        </xsl:call-template>
                    </FirstName>
                    <LastName>
                        <xsl:sequence select="fn:string(LastName)"/>
                    </LastName>
                </Customer>
            </xsl:for-each>
        </CompletePO>
    </xsl:template>
</xsl:stylesheet>
```

Note: As soon as a named template is used in a mapping, the XSLT file containing the named template is **included** in the generated XSLT code (**xsl:include href...**), and is **called** using the command **xsl:call-template**.

3. Click the **Output** tab to see the result of the mapping.

To remove custom XSLT libraries from MapForce:

1. Click **Add/Remove Libraries** at the base of the Libraries window. The Manage Libraries window opens.
2. Click **Delete Library** next to the library that is to be deleted.

6.5.1.2 Example: Summing Node Values

This example shows you how to process multiple nodes of an XML document and have the result mapped as a single value to a target XML document. Specifically, the goal of the mapping is to calculate the price of all products in a source XML file and write it as a single value to an output XML file. The files used in this example are available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder:

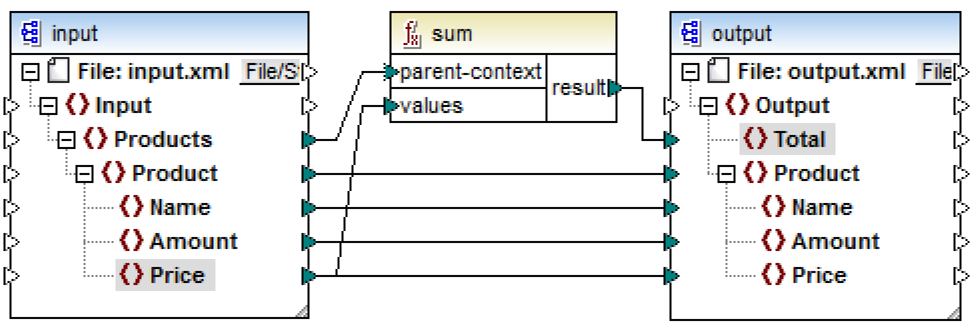
- **Summing-nodes.mfd** — the mapping file
- **input.xml** — the source XML file
- **input.xsd** — the source XML schema
- **output.xsd** — the target XML schema
- **Summing-nodes.xslt** — A custom XSLT stylesheet containing a named template to sum the individual nodes.

There are two different ways to achieve the goal of the mapping:

- By using the sum 988 function. This MapForce built-in function is available in the Libraries window.
- By importing a custom XSLT stylesheet into MapForce.

Solution 1: Using the "sum" aggregate function

To use the **sum** function in the mapping, drag it from the Libraries window into the mapping. Note that the functions available in the Libraries window depend on the XSLT language version you selected (XSLT 1 or XSLT 2). Next, create the mapping connections as shown below.



For more information about aggregate functions of the core library, see also core | aggregate functions 981.

Solution 2: Using a custom XSLT Stylesheet

As mentioned above, the aim of the example is to sum the `Price` fields of products in the source XML file, in this case products A and B.

```
<?xml version="1.0" encoding="UTF-8"?>
<Input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="input.xsd">
   <Products>
      <Product>
         <Name>ProductA</Name>
         <Amount>10</Amount>
```

```
            <Price>5</Price>
         </Product>
         <Product>
            <Name>ProductB</Name>
            <Amount>5</Amount>
            <Price>20</Price>
         </Product>
      </Products>
</Input>
```

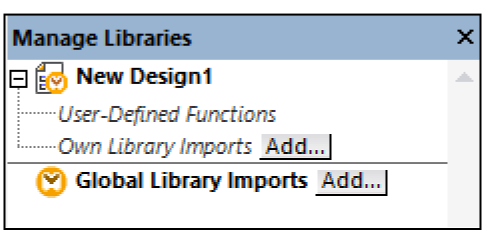
The code listing below shows a custom XSLT stylesheet which uses the named template "Total" and a single parameter `string`. The template works through the XML input file and sums all the values obtained by the XPath expression `/Product/Price`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

   <xsl:template match="*">
      <xsl:for-each select=".">
      <xsl:call-template name="Total">
         <xsl:with-param name="string" select="."/>
      </xsl:call-template>
      </xsl:for-each>
   </xsl:template>

   <xsl:template name="Total">
   <xsl:param name="string"/>
      <xsl:value-of select="sum($string/Product/Price)"/>
   </xsl:template>
</xsl:stylesheet>
```

Note: To sum the nodes in XSLT 2.0, change the stylesheet declaration to `version="2.0"`.

Before importing the XSLT stylesheet into MapForce, select XSLT 1.0 as a transformation language [27]. You are now ready to import the custom function, as follows:

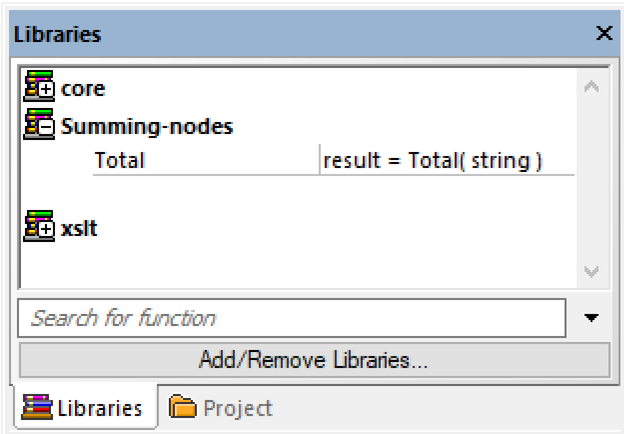
1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window [33]. The **Manage Libraries** window opens (*see screenshot below*).



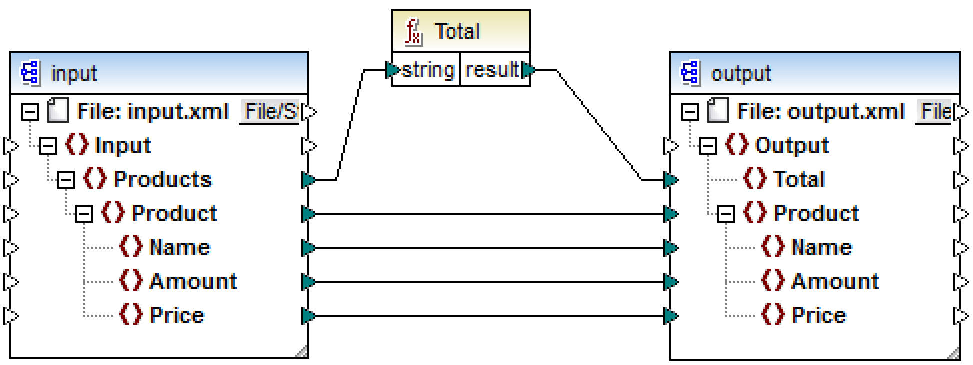
2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always

absolute.

3. Browse for **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Summing-nodes.xslt**, and click **Open**. A message box appears informing you that a new library has been added, and the new library appears in the Libraries window.



4. Drag the `Total` function from the Libraries into the mapping, and create the mapping connections as shown below.



To preview the mapping result, click the **Output** tab. The sum of the two `Price` fields is now displayed in the `Total` field.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Output xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="output.xsd">
   <Total>25</Total>
   <Product>
      <Name>ProductA</Name>
      <Amount>10</Amount>
      <Price>5</Price>
   </Product>
   <Product>
      <Name>ProductB</Name>
      <Amount>5</Amount>
      <Price>20</Price>
```

```
   </Product>
</Output>
```

6.5.2 Import Custom XQuery 1.0 Functions

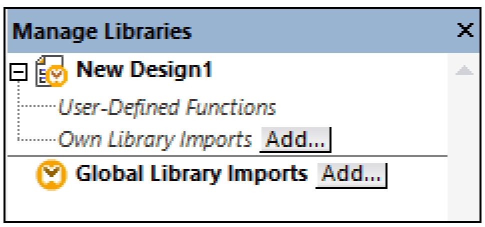
When XQuery is selected as mapping transformation language, MapForce displays the built-in function libraries available for XQuery in the Libraries window. If necessary, you can extend this list with custom XQuery functions, by importing custom XQuery 1.0 library modules into MapForce.

To be eligible for import into MapForce, an XQuery file must satisfy the following requirements:

- It must be a valid library module according to XQuery specification. In other words, it must start with a module declaration such as `module namespace <prefix>="<namespace name"`
- All functions declared in the imported library module must return atomic data types (for example, `xs:string`, `xs:boolean`, `xs:integer`, etc). Function parameters must also have atomic types.

To import an XQuery library module:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window 33. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for the .xq or .xquery library file, and click **Open**.

The imported library modules appear in the Libraries window, and then you can drag specific functions into the mapping area and use them like any other MapForce function component, see also Add a Function to the Mapping 905.

If you do not see the imported XQuery library module, make sure that XQuery is selected as a transformation language 27.

See also:

XQuery engine implementation 1896

6.5.2.1 Example: Import Custom XQuery Function

This example shows you how to import a demo XQuery library module into MapForce and call its functions from a mapping. The demo module in this example consists of only one function which calculates tax on decimal amounts as 20% of the amount. In a production scenario, an XQuery module may contain multiple functions.

> All functions declared in the XQuery module must return atomic types and their parameters must also be of atomic data types. Otherwise, the module is not eligible for import into MapForce.

You can find the demo XQuery module file at the following path relative to your personal "Documents" folder on the computer where MapForce is installed: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\module.xq**.

```
xquery version "1.0";

module namespace demo="http://www.altova.com/mapforce/demo";

declare function demo:calculatetax($val as xs:decimal) as xs:decimal {
   $val*0.2
};
```

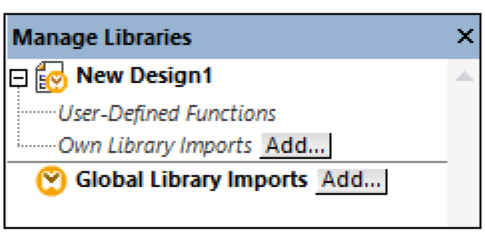
module.xq

After you import the XQuery module file into MapForce, you will be able to call the `demo:calculatetax` function from a mapping. Note that calculating the tax amount with the help of an XQuery function is just for demo purposes—you can achieve the same result by using MapForce built-in functions.

A demo mapping which calls the `demo:calculatetax` function above is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\CalculateTax_XQuery.mfd**. When you open this mapping initially, MapForce displays a warning that it contains one or more components that are not available in XQuery. This warning is normal and it occurs because the mapping references a function from a custom XQuery library module that was not imported yet. To remove the warning and run the mapping, we will import the missing XQuery module into MapForce as shown below.

To import the XQuery module into MapForce:

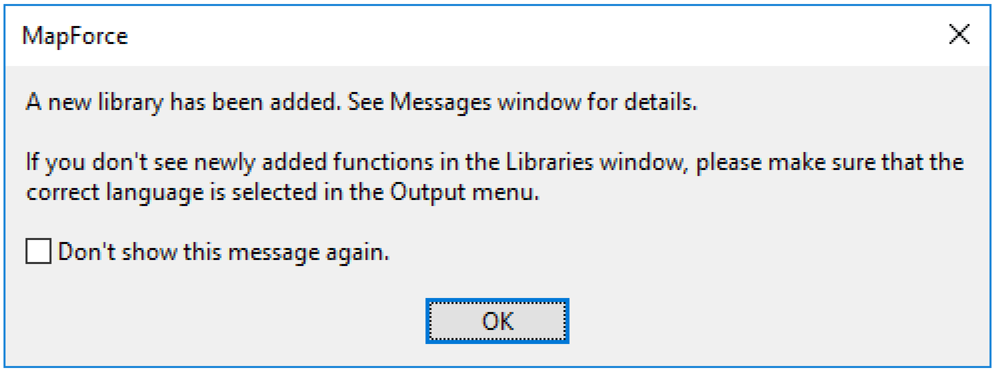
1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[33]. The **Manage Libraries** window opens (*see screenshot below*).



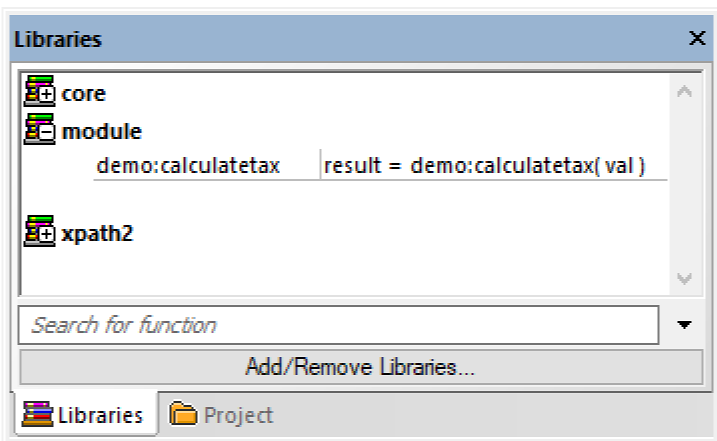
2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to

Global Library Imports. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

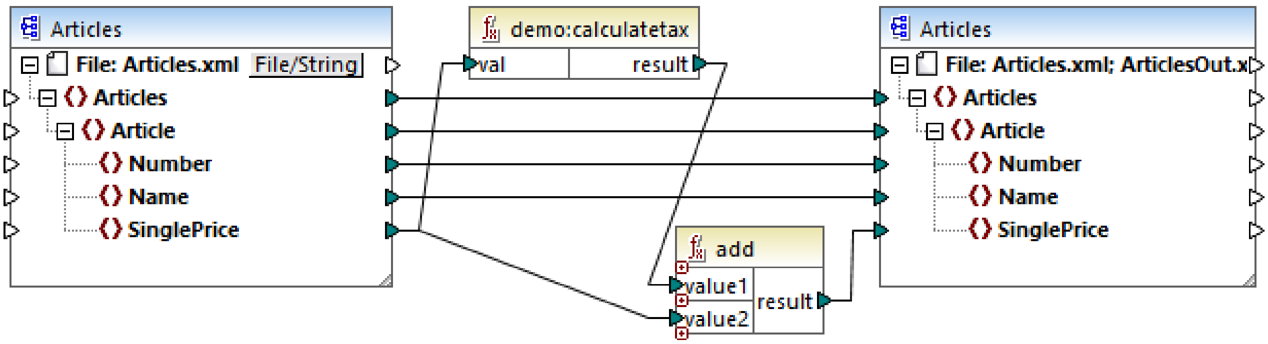
3. Browse for **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\module.xq** and click **Open**. A message box appears informing you that a new library has been added.



The imported library and the **demo:calculatetax** function are now visible in the Libraries window.



Also, the mapping can now be validated and run without warnings. The **demo:calculatetax** function illustrated in the image below originates from the imported XQuery module, and it can be added to the mapping just like any other built-in function, see Add a Function to the Mapping[905].



CalculateTax_XQuery.mfd

The mapping explained

The **CalculateTax_XQuery.mfd** mapping illustrated above takes as input an XML file that stores articles. Each article has a single price expressed as a decimal value, for example:

```
<Articles>
   <Article>
      <Number>1</Number>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>
   <Article>
      <Number>2</Number>
      <Name>Socks</Name>
      <SinglePrice>2.30</SinglePrice>
   </Article>
   <Article>
      <Number>3</Number>
      <Name>Pants</Name>
      <SinglePrice>34</SinglePrice>
   </Article>
   <Article>
      <Number>4</Number>
      <Name>Jacket</Name>
      <SinglePrice>57.50</SinglePrice>
   </Article>
</Articles>
```

Articles.xml

The mapping produces an XML file which abides by the same schema as the source XML file. Therefore, both the source and target components have the same structure on the mapping. As the mapping connections suggest, nearly all elements are mapped in a straightforward way from target to source—for example, for each **Article** in the source there will be an **Article** in the target. The values of all items are copied verbatim from the source XML, except for **SinglePrice**. The value of **SinglePrice** is calculated with the help of two functions:

- The **demo:calculatetax** XQuery function calculates the tax amount by taking the original **SinglePrice** as input.
- The MapForce built-in **add** function adds the tax amount to the original **SinglePrice** amount and returns the final amount.

Importantly, the data type of the **SinglePrice** item is `xs:decimal`—which corresponds to the input parameter type and the return type of the XQuery function.

The output produced by the mapping when you click the **Output** tab is illustrated below. Notice the increase of 20% applied to each price compared to the source XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<Articles xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    <Article>
        <Number>1</Number>
        <Name>T-Shirt</Name>
        <SinglePrice>30</SinglePrice>
    </Article>
    <Article>
        <Number>2</Number>
        <Name>Socks</Name>
        <SinglePrice>2.76</SinglePrice>
    </Article>
    <Article>
        <Number>3</Number>
        <Name>Pants</Name>
        <SinglePrice>40.8</SinglePrice>
    </Article>
    <Article>
        <Number>4</Number>
        <Name>Jacket</Name>
        <SinglePrice>69</SinglePrice>
    </Article>
</Articles>
```

6.5.3 Import Custom Java and .NET Libraries

This section explains how to import compiled Java class files and .NET DLL assemblies (including .NET 4.0 assemblies) into MapForce. If the imported libraries contain functions that use basic data types as parameters and return simple types, such functions appear in the **Libraries** window and can be used in mappings as any other function available in MapForce. The mapping output of imported Java and .NET functions can be previewed in the **Output** pane, and the functions are available in the generated code. To find out more about importing custom libraries, see the examples provided in Import Custom Java Class 957 and Import Custom .NET DLL Assembly 958.

Important:

- To import custom Java or .NET functions, you need compiled Java classes (`.class`) or .NET.dll assembly files. The import of Java `.jar` files or `.dll` files that are not a .NET assembly is not supported.

- .NET assembly files are supported when the mapping language is set to C#. The .NET assemblies may be written in .NET languages other than C# (e.g., C++.NET or VB.NET) if they use only the basic data types from the System Assembly as parameters and return types. For details, see .NET Function Support955.
- If you want to use custom .NET functions in the built-in output preview (in the **Output** pane), these functions need to be compiled for .NET Framework 4.x or .NET Standard 2.0.
- Compiled Java class (`.class`) files are supported when the mapping language is set to Java. Java Runtime Environment 7 or later must be installed on your computer. Only specific types and members are supported (see Java function support954).
- You cannot set the mapping language to C++ if the mapping uses imported Java `.class` or .NET DLL assemblies.
- You cannot set the mapping language to XSLT if the mapping uses imported Java `.class` or .NET DLL assemblies (a custom XSLT function that acts as an adapter would have to be written).
- The import of functions from native C++ DLLs is limited and requires a special approach. For more information, see Reference Java, C# and C++ Libraries Manually960.
- All functions called from a MapForce mapping should return the same value each time the function is called with the same input parameters. The exact order and the number of times a function is called by MapForce is undefined.
- In the case of Java, the imported class files and their packages do not need to be added to the `CLASSPATH` variable since the Built-in execution engine and generated Java code will automatically add imported packages to the Java engine's classpath or to Ant, respectively. However, any dependencies of the imported class files and packages will not be handled automatically. Therefore, if imported Java class files or packages depend on other class files, make sure to add the parent directories of all dependent packages to the `CLASSPATH` environment variable.

Java function support

Top-level classes, static member classes and non-static member classes are supported:

- `new <classname>(<arg1>, <arg2>, ...)`
- `<object>.new <member-class>(<arg1>, <arg2>, ...)`

Member functions and static functions are supported:

- `<function>(<arg1>, <arg2>, ...)`
- `<object>.<method>(<arg1>, ...)`

Supported connections between XML Schema and Java types:

Schema type	Java type
xs:string	String
xs:byte	byte
xs:short	short

Schema type	Java type
xs:int	int
xs:long	long
xs:boolean	boolean
xs:float	float
xs:double	double
xs:decimal	java.math.BigDecimal
xs:integer	java.math.BigInteger

Connections in both directions are possible. Other Java types (including array types) are not supported. Methods using such parameters or return values, will be ignored. Object types are supported by calling their constructor or as a return value of a method. They can be mapped to other Java methods. Manipulating the object using MapForce means is not possible.

.NET function support

Top-level classes and member classes are supported:

- `new <classname>(<arg1>, <arg2>, ...)`

Member functions and static functions are supported:

- `<function>(<arg1>, <arg2>, ...)`
- `<object>.<method>(<arg1>, ...)`

Supported connections between XML Schema and .NET/C# types:

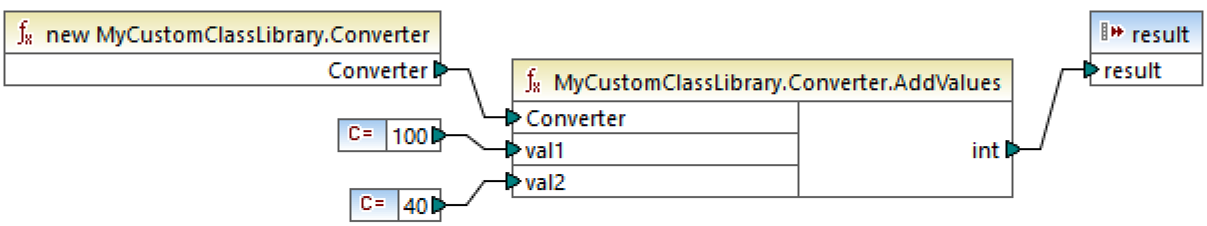
Schema type	.NET type	C# type
xs:string	System.String	string
xs:byte	System.SByte	sbyte
xs:short	System.Int16	short
xs:int	System.Int32	int
xs:long	System.Int64	long
xs:unsignedByte	System.Byte	byte
xs:unsignedShort	System.UInt16	ushort
xs:unsignedInt	System.UInt32	uint
xs:unsignedLong	System.UInt64	ulong
xs:boolean	System.Boolean	bool

Schema type	.NET type	C# type
`xs:float`	`System.Single`	`float`
`xs:double`	`System.Double`	`double`
`xs:decimal`	`System.Decimal`	`decimal`

Connections in both directions are possible. Other .NET/C# types (including array types) are not supported. Methods using such parameters or return values will be ignored. Object types are supported by calling their constructor or as a return value of a method. They can be mapped to other .NET methods. Manipulating the object using MapForce means is not possible.

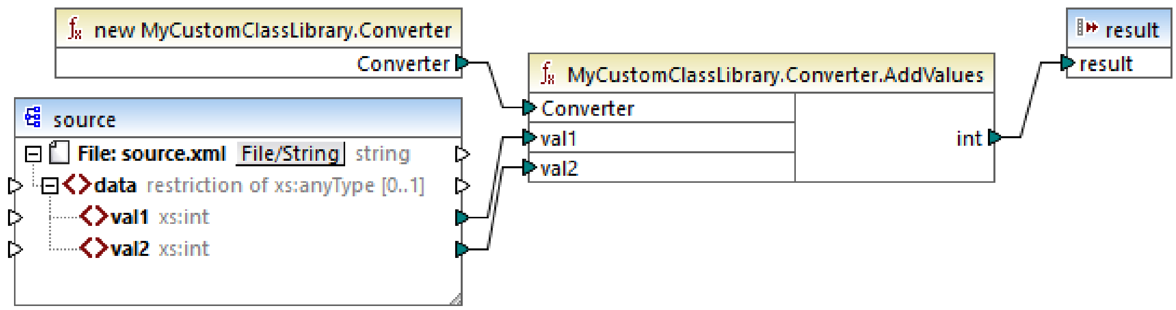
Data type issues and workarounds

When a function in your custom library expects integer types, connecting constants of type `Number` to the function's arguments may cause a type mismatch error similar to this one: `No match for MyCustomClassLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null.MyCustomClassLibrary.Converter.AddValues(MyCustomClassLibrary.Converter, xs:decimal, xs:decimal). Check argument types.` This issue is specific to constants of type `Number` only. A sample mapping that could generate this error is shown below. In this mapping, two constants of type `Number` are connected to the function's arguments of type `Integer`.



The possible workarounds are described below:

1. Change the constant type from **Number** to **All other**. You can do this after double-clicking the title bar of the constant component.
2. Instead of a constant, use a source component (for example, an XML file) that provides values of the data type expected by the function.



3. In your external code, create a wrapper function that accepts a decimal value and returns an integer value. The wrapper solution may be imported as a separate library. Therefore, you do not need to change the original source code of the target function to use this approach.

6.5.3.1 Example: Import Custom Java Class

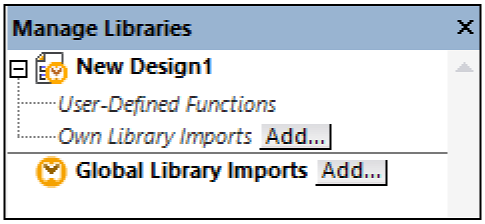
This example shows how to import a custom Java `.class` file into MapForce.

Note: Java SE 8 Runtime Environment or later is required to complete this example.

Java .class import

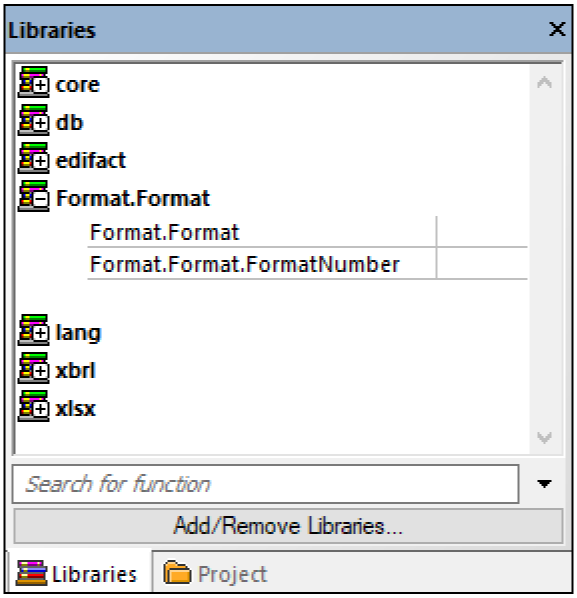
To add a Java `.class` file as a MapForce library, take the following steps:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[33]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Search for the following file: **`<Documents>\Altova\MapForce2023\MapForceExamples\Java\Format\Format.class`**. A message appears informing you that a new library has been added. The imported library is now visible in the **Libraries** window (*see screenshot below*).



If you do not see the newly imported library in the **Libraries** window, make sure that the transformation language[27] is set to Java. To add the function to the mapping, drag it from the **Libraries** window into the mapping area. For details, see Add a Function to the Mapping[905].

Mapping output
To preview the mapping output in MapForce, take the following steps:

1. Open the following mapping: **`<Documents>\Altova\MapForce2023\MapForceExamples\Java\FormatNumber.mfd`**. This is a complete mapping that already imports the Java `.class` library mentioned above.
2. Click the **Output** button to see the result of the mapping (*see screenshot below*).

```
Start date,End date,Region,Amount
2008-01-01,2008-01-31,CA,"110.400,00"
2008-01-01,2008-01-31,MA,"75.300,00"
2008-02-01,2008-02-29,CA,"114.300,00"
2008-02-01,2008-02-29,MA,"65.200,00"
2008-03-01,2008-03-31,CA,"134.200,00"
2008-03-01,2008-03-31,MA,"86.100,00"
2008-04-01,2008-04-30,CA,"107.300,00"
2008-04-01,2008-04-30,MA,"112.100,00"
2008-05-01,2008-05-31,CA,"114.400,00"
2008-05-01,2008-05-31,MA,"93.800,00"
```

Mapping in Java
To run the mapping in Java, follow the instructions below:

1. Click **Generate Code In | Java** in the **File** menu.
2. Select a target directory where the code should be generated and click **OK**.
3. Import the generated libraries into your Java project and build the Java application. For more information, see Example: Generate and Run Java Code[1514].

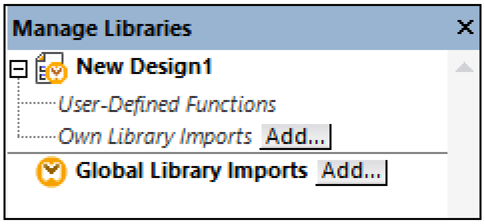
6.5.3.2 Example: Import Custom .NET DLL Assembly

This example shows how to import a custom .NET DLL assembly created in C# into MapForce. The source code of this sample is available at the following path: **`<Documents>\Altova\MapForce2023\MapForceExamples\C#\Format`**. The `.dll` assembly file that will be imported into MapForce is in the **`..\bin\Debug`** directory. You can also open the `.sln` solution file in Visual Studio and compile a new `.dll` file.

Note: If you want to use custom .NET functions in the built-in output preview (in the **Output** pane), these functions need to be compiled for .NET Framework 4.x or .NET Standard 2.0.

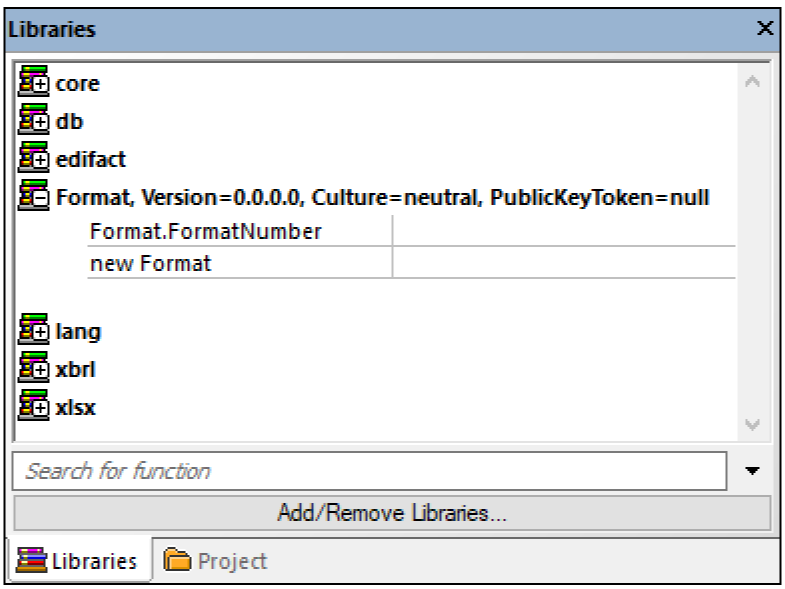
.NET assembly import
To import a .NET assembly file, take the following steps:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[33]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Search for `Format.dll` in the following directory: `...\MapForceExamples\C#\Format\bin\Debug\`. A message appears informing you that a new library has been added. The imported library is now visible in the **Libraries** window.



If you do not see the newly imported library in the **Libraries** window, make sure that the transformation language[27] is set to C#. To add the function to the mapping, drag it from the **Libraries** window into the mapping area. For more information, see Add a Function to the Mapping[905].

Mapping output
To preview the mapping output, take the following steps:

1. Open the `FormatNumber.mfd` file available in the following folder: `...\MapForceExamples\C#`. This is a sample mapping that has an imported `.dll` library mentioned above.
2. Click the **Output** button to see the result of the mapping (*see screenshot below*).

```
Start date,End date,Region,Amount
2008-01-01,2008-01-31,CA,"110.400,00"
2008-01-01,2008-01-31,MA,"75.300,00"
2008-02-01,2008-02-29,CA,"114.300,00"
2008-02-01,2008-02-29,MA,"65.200,00"
2008-03-01,2008-03-31,CA,"134.200,00"
2008-03-01,2008-03-31,MA,"86.100,00"
2008-04-01,2008-04-30,CA,"107.300,00"
2008-04-01,2008-04-30,MA,"112.100,00"
2008-05-01,2008-05-31,CA,"114.400,00"
2008-05-01,2008-05-31,MA,"93.800,00"
```

Mapping in C#
To run the mapping from a custom C# application, follow the instructions below:

1. Click **Generate Code In | C#** in the **File** menu.
2. Select a target directory where the code should be generated and click **OK**.
3. Build the application with Visual Studio and run the generated console application. For more details, see Generating C# code [1501].

6.5.4 Reference C#, C++ and Java Libraries Manually

This section explains how to reference custom libraries in a `.mff` file (MapForce Function File). The `.mff` file containing the reference can then be imported as a MapForce library. A `.mff` file is an XML file in which you manually define the link between class definitions in your custom code and MapForce. Once you create a custom `.mff` file, you can import it into MapForce, which is similar to importing a .NET DLL or Java class file.

Important:

- If you want to use custom .NET functions in the built-in output preview (in the **Output** pane), these functions need to be compiled for .NET Framework 4.x or .NET Standard 2.0.

- You can import a function into MapForce only if its return type and parameters are of simple type. To find out more about a list of data types available for each language, see Data Type Mapping [966].

- When you import function libraries from custom `.mff` files, the preview of the mapping directly in MapForce (in the **Output** pane) is limited. For libraries written in C++, the preview of the mapping in MapForce is not supported. For Java and C#, the preview is available when your library uses native language types, but it is not available if your library imports the Altova generated classes. However, you can generate code in the specific language targeted by your library. The custom functions will be available in the generated code, enabling you to run the mapping from the generated code.

- The exact order in which functions are called by the generated mapping code is undefined. MapForce may need to cache calculated results for reuse or evaluate expressions in any order. Therefore, it is recommended to use only custom functions that have no side effects.

- It is important to distinguish between user-defined functions and custom function libraries. User-defined functions are created graphically in a mapping; they cannot and need not be saved to a `.mff` file, because they are saved together with the `.mfd` file in which they are created. For more information, see Call and Import UDFs 930.

- If you are upgrading from a MapForce version earlier than 2010, you may need to update the data types used in your custom functions. For details, see Data Type Mapping 966.

To find out more about how to create and configure a custom `.mff` file, see Configure .mff File 961. The examples are provided in the following topics:

- Reference C# in .mff 969
- Reference C++ in .mff 970
- Reference Java in .mff 972

6.5.4.1 Configure .mff File

This topic provides instructions on how to configure a MapForce Function File file (`.mff`). A `.mff` file is a configuration file in XML format that allows importing functions from custom Java, C#, or C++ libraries into MapForce so that they appear in the **Libraries** window. A `.mff` file is an intermediary between your custom libraries and MapForce. The `.mff` file must be configured to specify i) the interfaces for the custom functions and ii) where the implementation can be found in the generated code.

Important:

- The `*.mff` library files must be valid against the following schema: **`C:\Program Files\MapForceLibraries\mff.xsd`**. The `mff.xsd` schema defines the custom library configuration and is for internal use only. Altova GmbH retains the right to change this file format with new releases.
- It is only possible to define one C#, C++, or Java class per `.mff` file.

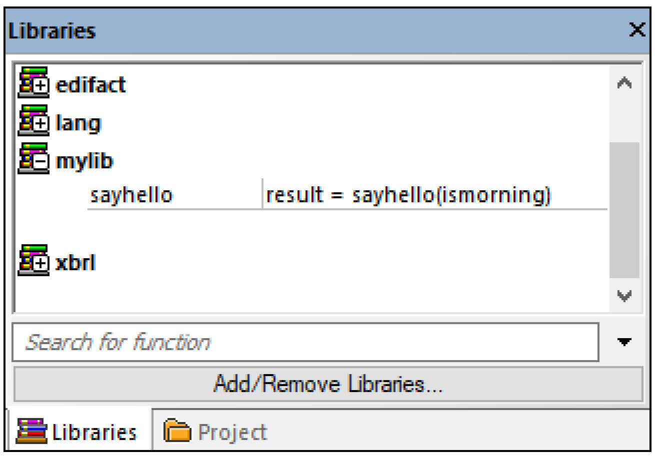
Sample .mff for C#
The following code listing illustrates a sample `.mff` file for C++:

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
   <implementations>
      <implementation language="cpp">
         <setting name="namespace" value="mylib"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Libraries\cpp"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
      </implementation>
   </implementations>
   <group name="greetings">
      <component name="sayhello">
         <sources>
            <datapoint name="ismorning" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string"/>
```

```
            </targets>
            <implementations>
               <implementation language="cpp">
                  <function name="SayHello"/>
               </implementation>
            </implementations>
            <description>
               <short>result = sayhello(ismorning)</short>
               <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
            </description>
         </component>
      </group>
   </mapping>
```

Imported custom library

The image below shows how a custom `.mff` file may look after being imported into MapForce. Notice that the custom library `mylib` appears as a library entry (sorted alphabetically), containing the `sayhello` string function.



Configuration steps

To configure the `.mff` file, follow the instructions below.

Step 1. Configure the library name

The library name can be found in the `.mff` file (*see below*). By convention, library names are written in lowercase in MapForce; however, you can also use uppercase letters.

```
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
```

In the sample above, the entry that will appear in the **Libraries** window is called `mylib`.

Step 2. Configure the language implementations

The `<implementations>` element is a mandatory element which specifies which languages your library should support, and it must be added as a child element of `<mapping>` (*see example below*).


```
<!-- ... -->
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
   <implementations>
      <implementation language="cpp">
         <setting name="namespace" value="mylib"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Libraries\cpp"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
      </implementation>
   </implementations>
<!-- ... -->
```

The settings within each `<implementation>` element allow the generated code to call the specific functions defined in Java, C++ or C#. A `.mff` file can be written so that it targets more than one programming language. In this case, every additional language must contain an additional `<implementation>` element. The specific settings for each programming language are discussed below.

Java library reference

```
<!-- ... -->
<implementation language="java">
   <setting name="package" value="com.hello.functions"/>
   <setting name="class" value="Greetings"/>
</implementation>
<!-- ... -->
```

It is important for the generated code to be able to find your **`Greetings.class`** file. Therefore, make sure to add a reference to your class to the Java class path.

C# library reference

```
<!-- ... -->
      <implementation language="cs">
         <setting name="namespace" value="MyLibrary" />
         <setting name="class" value="Greetings" />
         <setting name="reference" value="C:
\Libraries\cs\MyLibrary\bin\debug\MyLibrary.dll" />
      </implementation>
<!-- ... -->
```

For C#, it is important that the namespace in the code should correspond to the namespace defined in the `.mff` file (in the code listing above, the namespace is **`MyLibrary`**). The same is true for the class name (in the code listing above, the class name is **`Greetings`**). The third setting, `reference`, provides the path of the `dll` that is to be linked to the generated code.

C++ library reference

```
<!-- ... -->
   <implementation language="cpp">
         <setting name="namespace" value="MyLibrary"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Libraries\cpp"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
```

```
   </implementation>
<!-- ... -->
```

For C++, note the following:

- `namespace` is the namespace in which your `Greetings` class will be defined. It must be equal to the `library` attribute in `mapping` element.
- `path` is the path in which the include and the source files are to be found.
- When code for a mapping is generated, the include and source files will be copied to the directory **`targetdir/libraryname`**, which is defined when you select the menu command **File | Generate code in | C++**, and included in the project file.

All the include files you supply will be included in the generated algorithm.

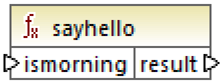
Step 3. Add a component
In the **Libraries** window, each function appears nested under a function group, for example, string functions. In the `.mff` file, a function corresponds to a `<component>` element. Conversely, each `<component>` must be nested under a `<group>` element, for example:

```
<!-- ... -->
<group name="string functions">
   <component name="sayhello">
   <!-- ... -->
   </component>
</group>
<!-- ... -->
```

The code shown below defines a sample function (component) called **`sayhello`**.

```
<!-- ... -->
<component name="sayhello">
   <sources>
      <datapoint name="ismorning" type="xs:boolean"/>
   </sources>
   <targets>
      <datapoint name="result" type="xs:string"/>
   </targets>
   <implementations>
   <!-- ... -->
   </implementations>
   <description>
     <short>result = sayhello(ismorning)</short>
     <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
   </description>
</component>
<!-- ... -->
```

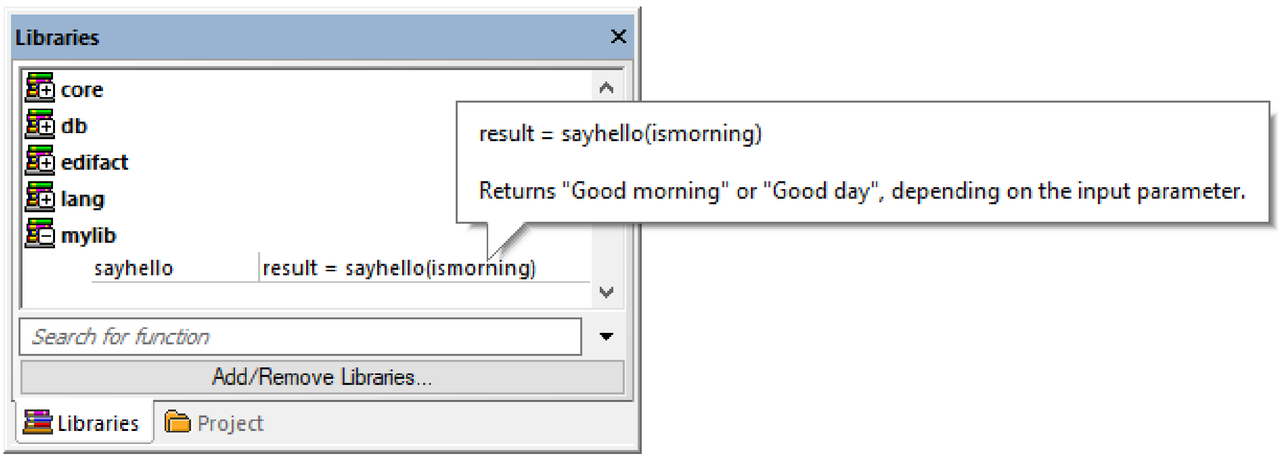
This is how the component above would look in MapForce:



In the code listing above, the `<datapoint>` element can be loosely defined as the input or output parameter of a function (also known as an input or output connector). The `type` argument of the `<datapoint>` element specifies the data type of the parameter or the data type of the return value. Only one target datapoint is allowed for each function. The number of source datapoints you can define is not limited.

The data type of each datapoint must be one of the XML Schema types (e.g., `xs:string`, `xs:integer`, etc.) These data types must correspond to the data types of the function's parameters you defined in your Java, C++ or C# library. To find out more about mapping of XML Schema datatypes to language types, see Data Type Mapping 966.

Functions are accompanied by short and long descriptions in the **Libraries** window. The short description is always shown to the right of the function name, while the long description is displayed as a tooltip when you place the mouse cursor over the short description (*see screenshot below*).



Step 4. Define language implementations

We can now connect the function in the **Libraries** window with the function in the custom Java, C# or C++ classes. This is achieved through the `<implementation>` element. One function may have multiple `<implementation>` elements—one for each supported programming language. A function may be called `Hello` in Java or `SayHello` in C++. This is why you need to specify a separate function name for each programming language. A function for each of the three programming languages might look as follows:

```
<!-- ... -->
<component name="sayhello">
<!-- ... -->
   <implementations>
      <implementation language="cs">
         <function name="HelloFunction"/>
      </implementation>
      <implementation language="java">
         <function name="Hello"/>
      </implementation>
      <implementation language="cpp">
         <function name="SayHello"/>
      </implementation>
   </implementations>
<!-- ... -->
```

```
</component>
<!-- ... -->
```

The value you supply as a function name must match the name of the method in the Java, C# or C++ class.

6.5.4.2 Import .mff Libraries

After you have created[961] a custom `.mff` file, you can import it into MapForce as follows:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[33]. The **Manage Libraries** window opens (*see screenshot below*).

2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Search for the custom `.mff` file and click **Open**.

The imported library becomes visible in the **Libraries** window after you set the mapping language to a language targeted by the custom library.

If you save the `*.mff` file in **`...\Altova\MapForce2023\MapForceLibraries`**, which is relative to the **Program Files** (or **Program Files (x86)** folder), the library is automatically loaded into the **Libraries** window when you start MapForce. Libraries and their functions can be toggled on or off, by deleting or adding the respective library file (`*.mff`).

6.5.4.3 Data Type Mapping

The table below lists the data types supported as function return types and parameter types when you create custom `.mff` files that reference your Java, C#, and C++ libraries. The table lists both native and non-native data types. If you need support for non-native data types such as Altova date, time and duration types, your custom Java and C# libraries must include a reference to the Altova libraries. In the case of C++, the Altova libraries must always be imported. For information about how to generate the Altova libraries, see Code Generator[1492].

XML Schema Type	Java Type	C# Type	C++ Type
`anyAtomicType`	`String`	`string`	`string_type`

XML Schema Type	Java Type	C# Type	C++ Type
anySimpleType	String	string	string_type
anyURI	String	string	string_type
base64Binary	byte[]	byte[]	altova::mapforce::blob
boolean	boolean	bool	bool
byte	int	int	int
date	com.altova.types.DateTime 1609	Altova.Types.DateTime 1594	altova::DateTime 1579
dateTime	com.altova.types.DateTime 1609	Altova.Types.DateTime 1594	altova::DateTime 1579
dayTimeDuration	com.altova.types.Duration 1613	Altova.Types.Duration 1598	altova::Duration 1582
decimal	java.math.BigDecimal	decimal	double
double	double	double	double
duration	com.altova.types.Duration 1613	Altova.Types.Duration 1598	altova::Duration 1582
ENTITIES	String	string	string_type
ENTITY	String	string	string_type
float	double	double	double
gDay	com.altova.types.DateTime 1609	Altova.Types.DateTime 1594	altova::DateTime 1579
gMonth	com.altova.types.DateTime 1609	Altova.Types.DateTime 1594	altova::DateTime 1579
gMonthDay	com.altova.types.DateTime 1609	Altova.Types.DateTime 1594	altova::DateTime 1579
gYear	com.altova.types.DateTime 1609	Altova.Types.DateTime 1594	altova::DateTime 1579
gYearMonth	com.altova.types.DateTime 1609	Altova.Types.DateTime 1594	altova::DateTime 1579
hexBinary	byte[]	byte[]	altova::mapforce::blob
ID	String	string	string_type
IDREF	String	string	string_type

XML Schema Type	Java Type	C# Type	C++ Type
IDREFS	String	string	string_type
int	int	int	int
integer	java.math.BigInteger	decimal	__int64
language	String	string	string_type
long	long	long	__int64
Name	String	string	string_type
NCName	String	string	string_type
negativeInteger	java.math.BigInteger	decimal	__int64
NMTOKEN	String	string	string_type
NMTOKENS	String	string	string_type
nonNegativeInteger	java.math.BigInteger	decimal	unsigned __int64
nonPositiveInteger	java.math.BigInteger	decimal	__int64
normalizedString	String	string	string_type
NOTATION	String	string	string_type
positiveInteger	java.math.BigInteger	decimal	unsigned __int64
QName	javax.xml.namespace.QName	Altova.Types.QName	altova::QName
short	int	int	int
string	String	string	string_type
time	com.altova.types.DateTime 1609	Altova.Types.DateTime 1594	altova::DateTime 1579
token	String	string	string_type
unsignedByte	long	ulong	unsigned __int64
unsignedInt	long	ulong	unsigned __int64
unsignedLong	java.math.BigInteger	ulong	unsigned __int64
unsignedShort	long	ulong	unsigned __int64
untypedAtomic	String	string	string_type
yearMonthDuration	com.altova.types.Duration 1613	Altova.Types.Duration 1598	altova::Duration 1582

6.5.4.4 Reference C# Library in .mff

This example shows how to create a sample C# library and reference it in a MapForce Function File (`.mff`). The `.mff` file can then be imported as a MapForce library. Referencing a C# library in a `.mff` file is one of the ways to import C# libraries into MapForce. A simpler alternative is to import .NET assemblies directly. For more information, see Example: Import Custom .NET DLL Assembly958.

Configuration steps

To reference a C# library in a `.mff` file, follow the instruction below.

Note: If you want to use custom .NET functions in the built-in output preview (in the **Output** pane), these functions need to be compiled for .NET Framework 4.x or .NET Standard 2.0.

Step 1. Create a new class library in VS

Create a new class library project in Visual Studio. Notice that the function has been defined as `public static`.

```
namespace MyLibrary
{
    public class Greetings
    {
        public static string SayHello(bool isMorning)
        {
            if (isMorning)
                return "Good morning!";
            return "Good Day!";
        }
    }
}
```

Step 2. Add a reference to Altova.dll

If you need special XML Schema types (such as date and duration), add a reference from your Visual Studio project to the `Altova.dll` library. To obtain this library, generate C# code from a mapping without custom functions. The `Altova.dll` file will be located in the **`..\Altova\bin\debug`** directory relative to the directory where the code was generated. To add the reference to `Altova.dll` in Visual Studio, click **Add Reference** in the **Project** menu and search for `Altova.dll`. Then add the line **`using Altova.Types;`** to your code. For information about how XML Schema types map to C# types, see Data Type Mapping966.

Step 3. Build your VS project

Build your Visual Studio project. The `MyLibrary.dll` file is generated in your project output directory.

Step 4.Create .mff and reference your C# library

Using an XML editor, create a new `.mff` file and validate it against the following schema: **`C:\Program Files\MapForceLibraries\mff.xsd`**. Make sure that all references under **`implementation language="cs"`** point to the correct C# members and paths created previously. The line **`function name="SayHello"`** must refer to the function name exactly as it was defined in C#. For details, see Configure .mff File961.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
```

```
    <implementations>
       <implementation language="cs">
          <setting name="namespace" value="MyLibrary" />
          <setting name="class" value="Greetings" />
          <setting name="reference" value="C:
\Libraries\cs\MyLibrary\bin\debug\MyLibrary.dll" />
       </implementation>
    </implementations>
    <group name="string functions">
       <component name="sayhello">
          <sources>
             <datapoint name="ismorning" type="xs:boolean"/>
          </sources>
          <targets>
             <datapoint name="result" type="xs:string"/>
          </targets>
          <implementations>
             <implementation language="cs">
                <function name="SayHello"/>
             </implementation>
          </implementations>
          <description>
             <short>result = sayhello(ismorning)</short>
             <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
          </description>
       </component>
    </group>
</mapping>
```

Step 5. Import .mff as a library
Now that your custom library is referenced in the `.mff` file, you can import the `.mff` file into MapForce as a library. For more information, see Import.mff Libraries 966.

6.5.4.5 Reference C++ in .mff

This example shows how to create a sample C++ library and reference it in a MapForce Function File (`.mff`). The `.mff` file can then be imported as a MapForce library.

Configuration steps

To reference a C++ library in a `.mff` file, follow the instructions below.

Step 1. Create a header file
Create a header (`.h`) file for your class library. The following code listing illustrates a sample header file called **`Greetings.h`**.

```
#ifndef MYLIBRARY_GREETINGS_H_INCLUDED
#define MYLIBRARY_GREETINGS_H_INCLUDED

#if _MSC_VER > 1000
#pragma once
```

```
#endif // _MSC_VER > 1000

using namespace altova;

namespace mylib {

class ALTOVA_DECLSPECIFIER Greetings
{
   public:
    static string_type SayHello(bool isMorning);
};

} // namespace mylib

#endif // MYLIBRARY_GREETINGS_H_INCLUDED
```

Notice that the function is declared as static and that the namespace `altova` is imported. Remember to write `ALTOVA_DECLSPECIFIER` in front of the class name: this ensures that your classes will be compiled correctly—whether you use dynamic or static linkage in the generated code.

Step 2. Create a .cpp file
Create a `.cpp` file with the same name as the header file. The `.cpp` file must be in the same directory as the `.h` file. The following code listing illustrates a sample `.cpp` file called **`Greetings.cpp`** that includes the **`Greetings.h`** file created previously:

```
#include "StdAfx.h"
#include "../Altova/Altova.h"
#include "../Altova/AltovaException.h"
#include "../Altova/SchemaTypes.h"

#include "Greetings.h"

namespace mylib {

   string_type Greetings::SayHello(bool isMorning)
   {
      if( isMorning )
         return _T("Good morning!");
      return _T("Good day!");
   }
}
```

Notice the lines that import **`StdAfx.h`** and several Altova libraries. These lines must be left unchanged. If the paths to the Altova libraries are correct in the generated code, these paths will point to the respective files. In contrast to Java or C#, you do not need to compile your source C++ files. They will be copied to the generated code and compiled with the rest of the generated mapping code.

Step 3. Create .mff and reference your C++ library
Using an XML editor, create a new `.mff` file and validate it against the following schema: **`C:\Program Files\MapForceLibraries\mff.xsd`**. Make sure that the namespace, function names and data types defined here correspond to those in the C++ code, as described in Configure .mff File 961. For information about data type support, see Data Type Mapping 966.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
   <implementations>
      <implementation language="cpp">
         <setting name="namespace" value="mylib"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Libraries\cpp"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
      </implementation>
   </implementations>
   <group name="greetings">
      <component name="sayhello">
         <sources>
            <datapoint name="ismorning" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="cpp">
               <function name="SayHello"/>
            </implementation>
         </implementations>
         <description>
            <short>result = sayhello(ismorning)</short>
            <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
         </description>
      </component>
   </group>
</mapping>
```

Step 4. Import .mff as a library
Now that your custom library is referenced in the `.mff` file, you can import the `.mff` file into MapForce as a library. For more information, see Import .mff File966.

C++ compiler errors
In order to execute mappings that use native C++ libraries, you will need to generate C++ code and run the mapping from your C++ code or application, as described in Generating C++ code1496. If you get a compiler error in **`#import "msado15.dll" rename("EOF", "EndOfFile")`**, modify the project properties to include a reference to **`msado15.dll`** in **`C:\Program Files\Common Files\System\ADO`**.

6.5.4.6 Reference Java in .mff

This example shows how to create a sample Java library and reference it in a MapForce Function File (`.mff`). The `.mff` file can then be imported as a MapForce library. Referencing a Java library in a `.mff` file is one of the ways to import Java libraries into MapForce. A simpler alternative is to import Java `.class` files directly. For more information, see Example: Import Custom Java Class957.

Configuration steps

To reference a C# library in a `.mff` file, follow the instruction below.

Step 1. Create a new project

Create a new Java project in your preferred development environment (for example, Eclipse).

Step 2. Add the com.mylib package

Add to the project a new package called **`com.mylib`** which consists of a class called **`Greetings`**. In the code listing below, notice that the `SayHello` function has been defined as `public static`.

```
package com.mylib;

public class Greetings {

   public static String SayHello ( boolean isMorning ) {
      if( isMorning )
         return "Good Morning!";
      return "Good Day!";
   }

}
```

Step 3. Import com.altova.types

Optionally, if your project needs support for special schema types such as date, time, and duration, import the `com.altova.types` package. To obtain this package, generate Java code from a mapping without custom functions: **`import com.altova.types.*;`**.

Step 4. Compile your custom library

Compile your custom library to a class file and add it to the Java classpath.

Step 5. Create .mff and reference your Java library

Using an XML editor, create a new `.mff` file and validate it against the following schema: **`C:\Program Files\MapForceLibraries\mff.xsd`**. Make sure that all references under **`implementation language="java"`** point to the correct Java members created previously. Also, the line **`function name="SayHello"`** must refer to the function name exactly as it was defined in Java. For more detail, see Configure .mff File[961].

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="custom" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
   <implementations>
      <implementation language="java">
         <setting name="package" value="com.mylib"/>
         <setting name="class" value="Greetings"/>
      </implementation>
   </implementations>
   <group name="greetings">
      <component name="sayhello">
         <sources>
            <datapoint name="ismorning" type="xs:boolean"/>
         </sources>
```

```
         <targets>
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="java">
               <function name="SayHello"/>
            </implementation>
         </implementations>
         <description>
            <short>result = sayhello(ismorning)</short>
            <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
         </description>
      </component>
   </group>
</mapping>
```

Step 6. Import .mff as a library

Now that your custom library is referenced in the `.mff` file, you can import the `.mff` file into MapForce as a library. For more information, see Import .mff Libraries[966].

6.6 Regular Expressions

When designing a MapForce mapping, you can use regular expressions ("regex") in the following contexts:

- In the **pattern** parameter of the **match-pattern** 1124 and **tokenize-regexp** 1069 functions
- To filter the nodes on which a node function should apply. For more information, see Applying Node Functions and Defaults Conditionally 921.
- To split text based on a pattern, when creating MapForce FlexText templates, see FlexText and Regular Expressions 708.

The regular expression syntax and semantics for XSLT and XQuery are as defined in Appendix F of "XML Schema Part 2: Datatypes Second Edition".

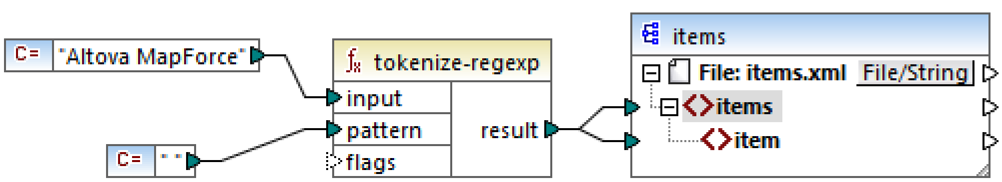
Note: When generating C++, C#, or Java code, the advanced features of the regular expression syntax might differ slightly. See the regex documentation of each language for more information.

Terminology

Let's examine the basic regular expression terminology by analyzing the **tokenize-regexp** function as an example. This function splits text into a sequence of strings, with the help of regular expressions. To achieve this, the function takes the following input parameters:

input	The input string to be processed by the function. The regular expression will operate on this string.
pattern	The actual regular expression pattern to be applied.
flags	This is an optional parameter that defines additional options (flags) that determine how the regular expression is interpreted, see "Flags" below.

In the mapping below, the input string is "Altova MapForce". The **pattern** parameter is a space character, and no regular expression flags are used.



This causes the text to be split whenever the space character occurs, so the mapping output is:

```
<items>
   <item>Altova</item>
   <item>MapForce</item>
</items>
```

Note that the **tokenize-regexp** function excludes the matched characters from the result. In other words, the space character in this example is omitted from the output.

The example above is very basic and the same result can be achieved without regular expressions, with the tokenize [1064] function. In a more practical scenario, the **pattern** parameter would contain a more complex regular expression. The regular expression can consist of any of the following:

- Literals
- Character classes
- Character ranges
- Negated classes
- Meta characters
- Quantifiers

Literals

Use literals to match characters exactly as they are written (literally). For example, if input string is `abracadabra`, and **pattern** is the literal `br`, the output is:

```
<items>
   <item>a</item>
   <item>acada</item>
   <item>a</item>
</items>
```

The explanation is that the literal `br` had two matches in the input string `abracadabra`. After removing the matched characters from the output, the sequence of three strings illustrated above is produced.

Character classes

If you enclose a set of characters in square brackets (`[` and `]`), this creates a character class. One and only one of the characters inside the character class is matched, for example:

- The pattern `[aeiou]` matches any lowercase vowel.
- The pattern `[mj]ust` matches "must" and "just".

Note: The pattern is case sensitive, so a lowercase "a" does not match the uppercase "A". To make the matching case insensitive, use the **i** flag, see below.

Character ranges

Use `[a-z]` to create a range between the two characters. Only one of the characters will be matched at one time. For example, the pattern `[a-z]` matches any lowercase character between "a" and "z".

Negated classes

Using the caret (`^`) as the first character after the opening bracket negates the character class. For example, the pattern `[^a-z]` matches any character not in the character class, including newline characters.

Matching any character

Use the dot (`.`) meta character to match any single character, except for newline character. For example, the pattern `.` matches any single character.

Quantifiers

Within a regular expression, quantifiers define how many times the preceding character or sub-expression is allowed to occur in order for the match to take place.

?	Matches zero or one occurrences of the immediately preceding item. For example, the pattern `mo?` will match "m" and "mo".
+	Matches one or more occurrences of the immediately preceding item. For example, the pattern `mo+` will match "mo", "moo", "mooo", and so on.
*****	Matches zero or more occurrences of the immediately preceding item.
{min,max}	Matches any number of occurrences between *min* and *max*. For example, the pattern `mo{1,3}` matches "mo", "moo", and "mooo".

Parentheses

Parentheses `(` and `)` are used to group parts of a regex together. They can be used to apply quantifiers to a sub-expression (as opposed to just one character), or with alternation (see below).

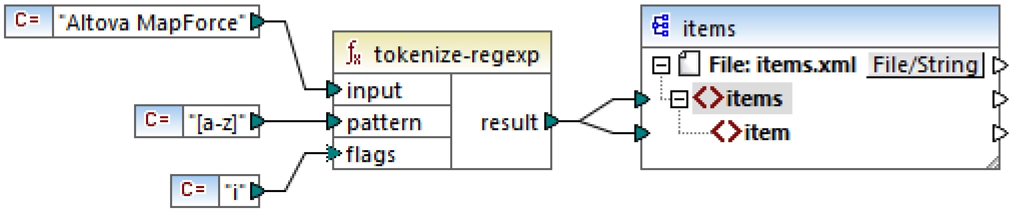
Alternation

The vertical bar (pipe) character `|` means "or". It can be used to match any of the several sub-expressions separated by `|`. For example, the pattern `(horse|make) sense` will match both "horse sense" and "make sense".

Flags

These are optional parameters that define how the regular expression is to be interpreted. Each flag corresponds to a letter. Letters may be in any order and can be repeated.

s	If this flag is present, the matching process operates in the "dot-all" mode. If the input string contains "hello" and "world" on two *different* lines, the regular expression `hello*world` will only match if the **s** flag is set.
m	If this flag is present, the matching process operates in multi-line mode. In multi-line mode, the caret `^` matches the start of any line, i.e. the start of the entire string and the first character after a newline character. The dollar character `$` matches the end of any line, i.e. the end of the entire string and the character immediately before a newline character. Newline is the character **#x0A**.
i	If this flag is present, the matching process operates in case-insensitive mode. For example, the regular expression `[a-z]` plus the **i** flag matches all letters a-z and A-Z.

	
x	If this flag is present, whitespace characters are removed from the regular expression prior to the matching process. Whitespace characters are **#x09**, **#x0A**, **#x0D** and **#x20**. **Note:** Whitespace characters within a character class are not removed, for example, `[#x20]`.

6.7 Function Library Reference

This reference section describes the MapForce built-in functions available in the Libraries window [33]. The functions are organized by library. The availability of function libraries in the **Libraries** window depends on the transformation language you choose for your mapping. To find out more about the list of available transformation languages, see this topic [27].

The information about the compatibility of functions and transformation languages is provided in the subsections below.

core functions

The lists below summarize the compatibility of core functions with transformation languages.
core | aggregate functions

- **avg**, **max**, **max-string**, **min**, **min-string**: XSLT 2.0, XSLT 3.0, XQuery 1.0, C#, C++, Java, Built-In;
- **count**, **sum**: all transformation languages.

core | conversion functions

- **boolean**, string, **number**: all transformation languages;
- **format-date**, **format-dateTime**, **format-time**: XSLT 2.0, XSLT 3.0, C#, C++, Java, Built-In;
- **format-number**: XSLT 1.0, XSLT 2.0, XSLT 3.0, C#, C++, Java, Built-In;
- **parse-date**, **parse-dateTime**, **parse-number**, **parse-time**: C#, C++, Java, Built-In.

core | file path functions
All the file path functions are compatible with all the transformation languages.

core | generator functions
The **auto-number** function is available for all the transformation languages.

core | logical functions
The logical functions are compatible with all the transformation languages.

core | math functions

- **add**, **ceiling**, **divide**, **floor**, **modulus**, **multiply**, **round**, **subtract**: all transformation languages;
- **round-precision**: C#, C++, Java, Built-In.

core | node functions

- **is-xsi-nil**, **local-name**, **static-node-annotation**, **static-node-name**: all transformation languages;
- **node-name**, **set-xsi-nil**, **substitute-missing-with-xsi-nil**: XSLT 2.0, XSLT 3.0, XQuery 1.0, C#, C++, Java, Built-In.

core | QName functions
The QName functions are compatible with all the transformation languages except for XSLT1.0.

core | sequence functions

- **`exists`**, **`not-exists`**, **`position`**, **`substitute-missing`**: all transformation languages;
- **`distinct-values`**, **`first-items`**, **`generate-sequence`**, **`item-at`**, **`items-from-till`**, **`last-items`**, **`replicate-item`**, **`replicate-sequence`**, **`set-empty`**, **`skip-first-items`**: XSLT 2.0, XSLT 3.0, XQuery 1.0, C#, C++, Java, Built-In;
- **`group-adjacent`**, **`group-by`**, **`group-ending-with`**, **`group-into-blocks`**, **`group-starting-with`**: XSLT 2.0, XSLT 3.0, C#, C++, Java, Built-In.

core | string functions

- **`concat`**, **`contains`**, **`normalize-space`**, **`starts-with`**, **`string-length`**, **`substring`**, **`substring-after`**, **`substring-before`**, **`translate`**: all transformation languages;
- **`char-from-code`**, **`code-from-char`**, **`tokenize`**, **`tokenize-by-length`**, **`tokenize-regexp`**: XSLT 2.0, XSLT 3.0, XQuery 1.0, C#, C++, Java, Built-In.

bson functions (MapForce Enterprise Edition only)

All the BSON functions are compatible only with Built-In.

db functions (MapForce Professional and Enterprise editions)

The `db` functions are compatible with C#, C++, Java, Built-In.

edifact functions (MapForce Enterprise Edition only)

The `edifact` functions are compatible with C#, C++, Java, Built-In.

lang functions (MapForce Professional and Enterprise editions)

The lists below summarize the compatibility of `lang` functions with transformation languages.
lang | datetime functions
The `lang | datetime` functions are compatible with C#, C++, Java, Built-In.

lang | file functions
The functions **`read-binary-file`** and **`write-binary-file`** are compatible only with Built-In.

lang | generator functions
The **`create-guid`** function is available for C#, C++, Java, Built-In.

lang | logical functions
The `lang | logical` functions are available for C#, C++, Java, Built-In.

lang | math functions
The `lang | math` functions are available for C#, C++, Java, Built-In.

lang | QName functions
The `lang | QName` functions are compatible with C#, C++, Java, Built-In.

lang | string functions

- **`charset-decode`**, **`charset-encode`**: Built-In;
- **`match-pattern`**: C#, Java, Built-In.

- **`capitalize`**, **`count-substring`**, **`empty`**, **`find-substring`**, **`format-guid-string`**, **`left`**, **`left-trim`**, **`lowercase`**, **`pad-string-left`**, **`pad-string-right`**, **`repeat-string`**, **`replace`**, **`reversefind-substring`**, **`right`**, **`right-trim`**, **`string-compare`**, **`string-compare-ignore-case`**, **`uppercase`**: C#, C++, Java, Built-In.

mime functions (MapForce Enterprise Edition only)

The `mime` functions are available for Built-In only.

xbrl functions (MapForce Enterprise Edition only)

The `xbrl` functions are compatible with C#, C++, Java, Built-In.

xlsx functions (MapForce Enterprise Edition only)

The `xlsx` functions are compatible with XSLT 2.0, XSLT 3.0, C#, Java, and Built-In.

xpath2 functions

All the `xpath2` functions are compatible with XSLT 2.0, XSLT 3.0, and XQuery 1.0.

xpath3 functions

All the `xpath3` functions are compatible only with XSLT 3.0.

xslt10 functions

The lists below summarize the compatibility of `xslt10` functions with transformation languages.
xslt10 | xpath functions

- **`local-name`**, **`name`**, **`namespace-uri`**: XSLT 1.0, XSLT 2.0, and XSLT 3.0.
- **`lang`**, **`last`**, **`position`**: XSLT 1.0.

xslt10 | xslt functions

- **`generate-id`**, **`system-property`**: XSLT 1.0, XSLT 2.0, and XSLT 3.0.
- **`current`**, **`document`**, **`element-available`**, **`function-available`**, **`unparsed-entity-uri`**: XSLT 1.0.

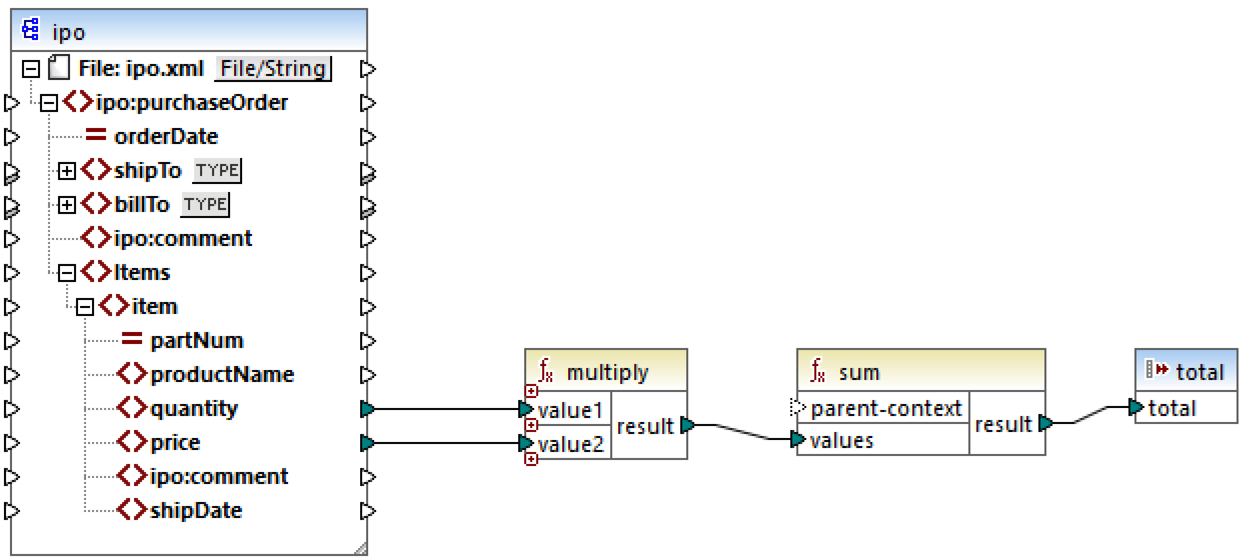
6.7.1 core | aggregate functions

"Aggregating" means processing multiple values of the same type so as to obtain a single result, such as a sum, a count, or an average. You can perform data aggregation in MapForce with the help of aggregation functions, such as `avg`, `count`, `max`, and others.

The following two arguments are common to all aggregation functions:

1. **parent-context**. This argument is optional; it lets you override the default mapping context (and thus change the scope of the function, or the values that the function must iterate over). For a worked example, see Example: Changing the Parent Context 1284.

2. **values**. This argument must be connected to a source item that supplies the values to be processed. For example, in the mapping illustrated below, the **sum** function takes as input a sequence of numeric values that originates from a source XML file. For each item in the source XML file, the **multiply** function gets the item's price times quantity, and passes the result to the **sum** function. The **sum** function will aggregate all input values and produce a total result that is also the output of the mapping. You can find this mapping in the **MapForceExamples** folder.

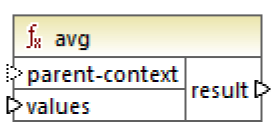


SimpleTotal.mfd

Some aggregate functions, such as min, max, sum, and avg, work exclusively with numeric values. The input data of these functions is converted to the **decimal** data type for processing.

6.7.1.1 avg

Returns the average value of all values within the input sequence. The average of an empty set is an empty set.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

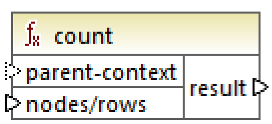
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context[1284]. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

Example

See Example: Grouping Records by Key[897].

6.7.1.2 count

Returns the number of individual items making up the input sequence. The count of an empty set is zero.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Note that this function has limited functionality in XSLT 1.0.

Parameters

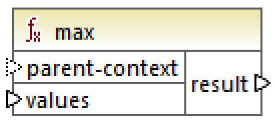
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context[1284]. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
nodes/rows	This argument must be connected to the source item to be counted.

Example

See Example: Changing the Parent Context[1284], Example: Counting Database Table Rows[855].

6.7.1.3 max

Returns the maximum value of all numeric values in the input sequence. The maximum of an empty set is an empty set.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

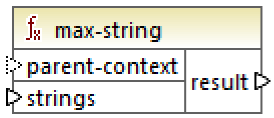
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context[1284]. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the maximum from a sequence of strings, use the max-string[984] function.

Example

See Example: Grouping Records by Key[897].

6.7.1.4 max-string

Returns the maximum value of all string values in the input sequence. For example, **`max-string("a", "b", "c")`** returns **`"c"`**. The function returns an empty set if the **strings** argument is an empty set.



Languages

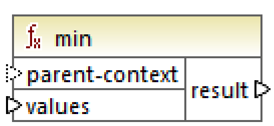
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context [1284]. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

6.7.1.5 min

Returns the minimum value of all numeric values in the input sequence. The minimum of an empty set is an empty set.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context [1284]. The `parent-context` argument is an optional argument in some MapForce core

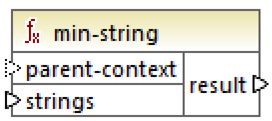
Argument	Description
	aggregation functions (e.g., **min**, **max**, **avg**, **count**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the minimum from a sequence of strings, use the min-string [986] function.

Example

See Example: Grouping Records by Key [897].

6.7.1.6 min-string

Returns the minimum value of all string values in the input sequence. For example, `min-string("a", "b", "c")` returns `"a"`. The function returns an empty set if the **strings** argument is an empty set.



Languages

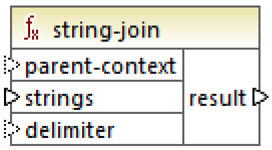
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context [1284]. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **min**, **max**, **avg**, **count**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

6.7.1.7 string-join

Concatenates all the values of the input sequence into one string delimited by whatever string you choose to use as the delimiter. The function returns an empty string if the **strings** argument is an empty set.



Languages

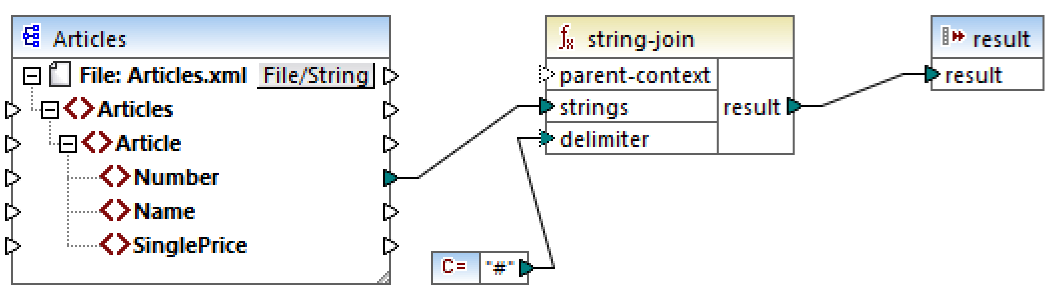
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also [Example: Changing the Parent Context] [1284]. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.
delimiter	Optional argument. Specifies the delimiter to be inserted between any two consecutive strings.

Example

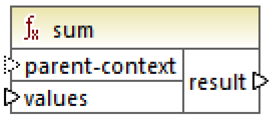
In the example below, the source XML file contains four **Article** items, with the following numbers: 1, 2, 3, and 4.



The constant supplies the character "#" as the delimiter. The mapping result is, therefore, **`1#2#3#4`**. If you do not supply a delimiter, then the result becomes **`1234`**.

6.7.1.8 sum

Returns the arithmetic sum of all values in the input sequence. The sum of an empty set is zero.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context[1284]. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

Example

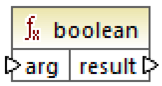
See Example: Summing Node Values[946].

6.7.2 core | conversion functions

To support explicit data type conversion, several type conversion functions are available in the **conversion** library. Note that the conversion functions are not always necessary because, in most cases, MapForce creates the necessary conversions automatically. Conversion functions are typically useful to format date and time values, or to compare values. For example, if some mapping items are of differing types (such as integer and string), you can use the number[997] conversion function to force a numeric comparison.

6.7.2.1 boolean

Converts the value of **arg** to a Boolean value. This may be useful for working with logical functions (such as **`equal`**, **`greater`**, and so on), as well as filters and if-else conditions[878]. To get a Boolean **false**, supply an empty string or numeric 0 as argument. To get a Boolean **true**, supply a non-empty string or numeric 1 as argument.



Languages

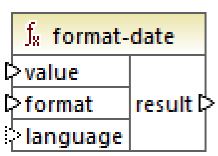
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameter

Argument	Description
arg	Mandatory argument. Supplies the value to be converted.

6.7.2.2 format-date

Converts a date value of type `xs:date` to a string and formats it according to specified options.



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

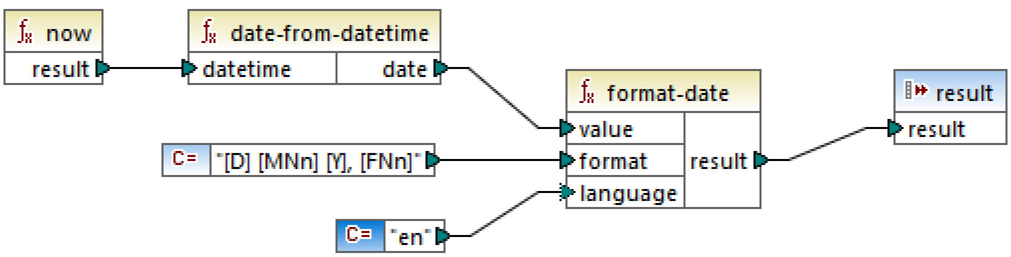
Parameters

Argument	Description
value	The `xs:date` value to be formatted.
format	A format string identifying the way in which the date is to be formatted. This argument is used in the same way as the **format** argument in the format-dateTime[990] function.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values:

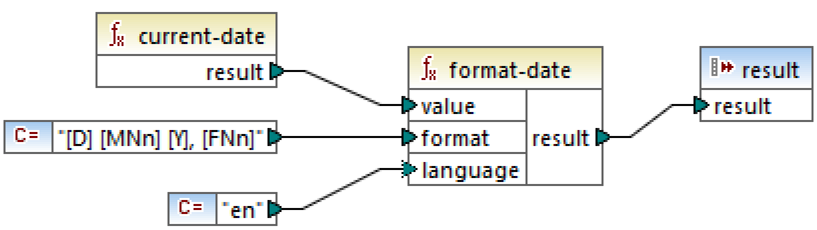
Argument	Description
	de German **en (default)** English **es** Spanish **fr** French **ja** Japanese

Example

The following mapping outputs the current date in a format like: "25 March 2020, Wednesday". To translate this value to Spanish, set the value of the **language** argument to **es**.

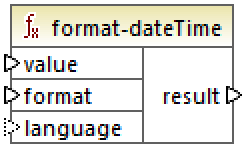


Note that the mapping above is designed for the Built-in, C++, C#, or Java transformation languages. In XSLT 2.0, the same result can be achieved by the following mapping:



6.7.2.3 format-dateTime

Converts a value of type `xs:dateTime` to a string. The string representation of date and time is formatted according to the value of the **format** argument.



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value	The `xs:dateTime` value to be formatted.
format	A format string identifying the way in which **value** is to be formatted. See "Remarks" below.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values: **de** German **en (default)** English **es** Spanish **fr** French **ja** Japanese

Note: If the function's output (result) is connected to an item of type other than string, the formatting may be lost as the value is cast to the target type. To disable this automatic cast, clear the **Cast target values to target types** check box in the Component Settings 82 of the target component.

Remarks

The **format** argument consists of a string containing so-called variable markers enclosed in square brackets, for example `[Y]/[M]/[D]`. Characters outside the square brackets are literal characters. If square brackets are needed as literal characters in the result, then they should be doubled.

Each variable marker consists of a component specifier identifying which component of the date or time is to be displayed, an optional formatting modifier, another optional presentation modifier and an optional width modifier, preceded by a comma if it is present.

```
format := (literal | argument)*
argument := [component(format)?(presentation)?(width)?]
width := , min-width ("-" max-width)?
```

The components are as follows:

Specifier	Description	Default Presentation
Y	year (absolute value)	four digits (2010)
M	month of the year	1-12
D	day of month	1-31
d	day of year	1-366
F	day of week	name of the day (language dependent)
W	week of the year	1-53
w	week of month	1-5
H	hour (24 hours)	0-23
h	hour (12 hour)	1-12
P	A.M. or P.M.	alphabetic (language dependent)
m	minutes in hour	00-59
s	seconds in minute	00-59
f	fractional seconds	numeric, one decimal place
Z	timezone as a time offset from UTC	+08:00
z	timezone as a time offset using GMT	GMT+n

The formatting modifier can be one of the following:

Character	Description	Example
1	Decimal numeric format with no leading zeros	1, 2, 3
01	Decimal format, two digits	01, 02, 03
N	Name of component, upper case[1]	MONDAY, TUESDAY
n	Name of component, lower case[1]	monday, tuesday
Nn	Name of component, title case[1]	Monday, Tuesday

Footnotes:

1. The **N**, **n**, and **Nn** modifiers are supported by the following components only: **M**, **d**, **D**.

The width modifier, if necessary, is introduced by a comma, followed by a digit that expresses the minimum width. Optionally, you can add a dash followed by another digit that expresses the maximum width. For example:

- **`[D,2]`** is the day of the month, with leading zeros (two digits).
- **`[MNn,3-3]`** is the name of the month, written as three characters, e.g. *Jan*, *Feb*, *Mar*, and so on.

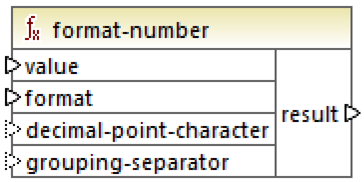
Examples

The table below illustrates some examples of formatting `xs:dateTime` values with the help of the **`format-dateTime`** function. The "Value" column specifies the value supplied to the **value** argument. The "Format" column specifies the value of the **format** argument. The "Result" column illustrates what is returned by the function.

Value	Format	Result
2003-11-03T00:00:00	`[D]/[M]/[Y]`	3/11/2003
2003-11-03T00:00:00	`[Y]-[M,2]-[D,2]`	2003-11-03
2003-11-03T00:00:00	`[Y]-[M,2]-[D,2] [H,2]:[m]:[s]`	2003-11-03 00:00:00
2010-06-02T08:02	`[Y] [MNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f]`	2010 June 02 Wed 153 8:02:12.054
2010-06-02T08:02	`[Y] [MNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f] [z]`	2010 June 02 Wed 153 8:02:12.054 GMT+02:00
2010-06-02T08:02	`[Y] [MNn] [D1] [F] [H]:[m]:[s].[f] [Z]`	2010 June 2 Wednesday 8:02:12.054 +02:00
2010-06-02T08:02	`[Y] [MNn] [D] [F,3-3] [H01]:[m]:[s]`	2010 June 2 Wed 08:02:12

6.7.2.4 format-number

Converts a number into a string and formats it according to the specified options.



Languages

Built-in, C++, C#, Java, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value	Mandatory argument. Supplies the number to be formatted.
format	Mandatory argument. Supplies a format string that identifies the way in which the number is to be formatted. See "Remarks" below.
decimal-point-format	Optional argument. Supplies the character to be used as the decimal point character. The default value is the full stop (.) character.
grouping-separator	Optional argument. Supplies the character used to separate groups of numbers. The default value is the comma (,) character.

Note: If the function's output (result) is connected to an item of type other than string, the formatting may be lost as the value is cast to the target type. To disable this automatic cast, clear the **Cast target values to target types** check box in the Component Settings 82 of the target component.

Remarks

The **format** argument takes the following form:

```
format := subformat (;subformat)?
  subformat := (prefix)? integer (.fraction)? (suffix)?
  prefix := any characters except special characters
  suffix := any characters except special characters
  integer := (#)* (0)* ( allowing ',' to appear)
  fraction := (0)* (#)* (allowing ',' to appear)
```

The first *subformat* is used for formatting positive numbers, and the second subformat for negative numbers. If only one *subformat* is specified, then the same subformat will be used for negative numbers, but with a minus sign added before the *prefix*.

Special Character	Default	Description
zero-digit	0	A digit will always appear at this point in the result
digit	#	A digit will appear at this point in the result string unless it is a redundant leading or trailing zero
decimal-point	.	Separates the integer and the fraction part of the number.
grouping-separator	,	Separates groups of digits.
percent-sign	%	Multiplies the number by 100 and shows it as a percentage.
per-mille	‰	Multiplies the number by 1000 and shows it as per-mille.

The table below illustrates examples of format strings and their result.

Note: The rounding method used by the `format-number` function is "half up", which means that the value gets rounded up if the fraction is greater than or equal to 0.5. The value gets rounded down if the fraction is less than 0.5. This method of rounding applies only to generated program code and the built-in execution engine. In XSLT 1.0, the rounding mode is undefined. In XSLT 2.0, the rounding mode is "round-half-to-even".

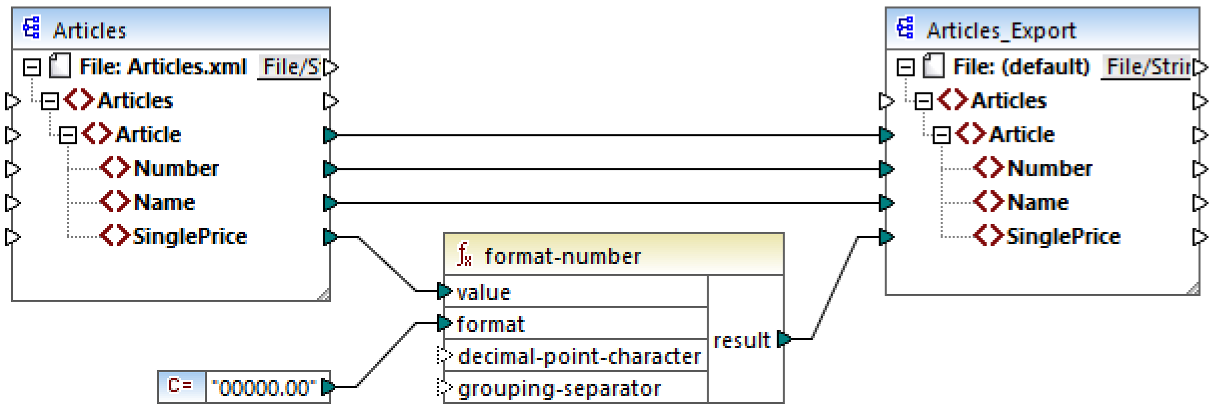
Number	Format String	Result
1234.5	#,##0.00	1,234.50
123.456	#,##0.00	123.46
1000000	#,##0.00	1,000,000.00
-59	#,##0.00	-59.00
1234	###0.0###	1234.0
1234.5	###0.0###	1234.5
.00025	###0.0###	0.0003
.00035	###0.0###	0.0004
0.25	#00%	25%
0.736	#00%	74%
1	#00%	100%
-42	#00%	-4200%
-3.12	#.00;(#.00)	(3.12)
-3.12	#.00;#.00CR	3.12CR

Example

The mapping illustrated below reads data from source XML and writes it to a target XML. There are multiple **SinglePrice** elements in the source that contain the following decimal values: **25**, **2.30**, **34**, **57.50**. The mapping has two goals:

1. Pad all values with zeros to the left so that the significant part takes 5 digits exactly
2. Pad all values with zeros to the right so that the decimal part takes 2 digits exactly

To achieve this, the format string `00000.00` was supplied as argument to the `format-number` function.



PreserveFormatting.mfd

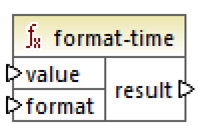
Consequently, the values in the target have become:

```
00025.00
00002.30
00034.00
00057.50
```

You can find the mapping design file at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\PreserveFormatting.mfd**.

6.7.2.5 format-time

Converts an `xs:time` input value into a string.



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

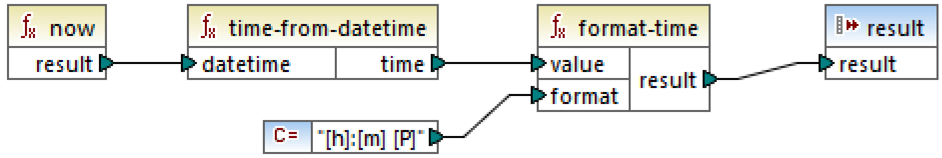
Parameters

Argument	Description
value	Mandatory argument. Supplies the `xs:time` value to be formatted.
format	Mandatory argument. Supplies a format string. This argument is used in the same way as the **format** argument in the format-dateTime 990 function.

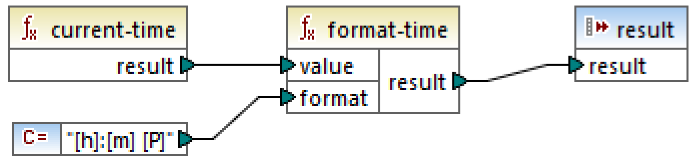
Example

The following mapping outputs the current time in a format like `2:15 p.m.` . To achieve this, it uses the format string `[h]:[m] [P]`, where:

- **[h]** is the current hour in 12-hour format
- **[m]** is the current minute
- **[P]** is the "a.m." or "p.m." part

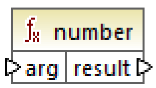


Note that the mapping above is designed for the Built-in, C++, C#, or Java transformation languages. In XSLT 2.0, the same result can be achieved by the following mapping:



6.7.2.6 number

Converts the value of **arg** into a number, where **arg** is a string or Boolean value. If **arg** is a string, MapForce will attempt to parse it as a number. For example, a string like `"12.56"` is converted to the decimal value `12.56`. If **arg** is Boolean **true**, it is converted to numeric **1**. If **arg** is Boolean **false**, it is converted to numeric **0**.



Languages

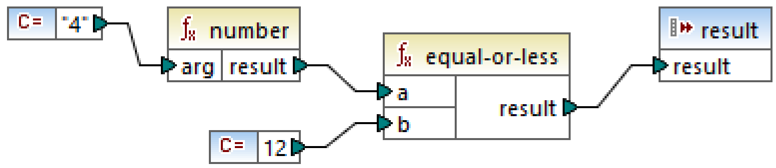
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
arg	Mandatory argument. Supplies the value to be converted.

Example

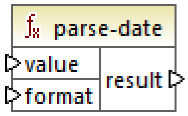
In the example below, the first constant is of type `string` and it contains the string "4". The second constant contains the numeric constant 12. In order for the two values to be compared as numbers, the types must agree.



Adding a **`number`** function to the first constant converts the string "4" to the numeric value of 4. The result of the comparison is then "true". If the **`number`** function were not used (that is, if "4" was connected directly to **a**), a string comparison would occur, with the result being "false".

6.7.2.7 parse-date

Converts a string into a date. This function uses the parse-dateTime999 function as a basis, while ignoring the time component. The result is of type `xs:date`.



Languages

Built-in, C++, C#, Java.

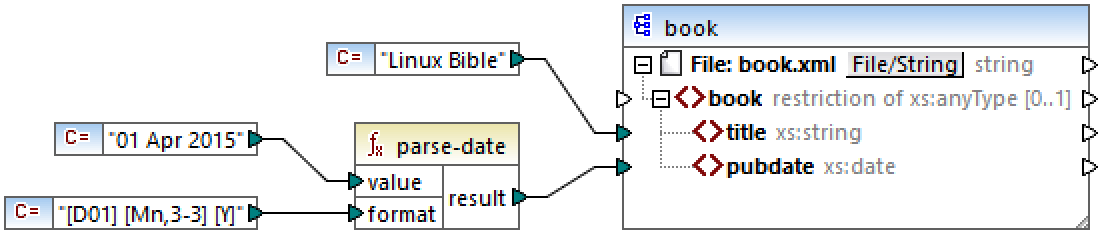
Parameters

Argument	Description
value	Mandatory argument. Supplies the string value to be converted.
format	Mandatory argument. Supplies a format string. This argument is used in the same way as the **format** argument in the parse-dateTime 999 function.

Example

The mapping below parses the string **`"01 Apr 2015"`**, converts it to a date and writes the result to a target item (**pubdate**) of type `xs:date`. This was achieved by using the format **`[D01] [MNn,3-3] [Y]`**, where:

- **`[D01]`** is the date of the month, expressed as two digits
- **`[MNn,3-3]`** is the month name, with a minim and maximum width of 3 characters
- **`[Y]`** is the year



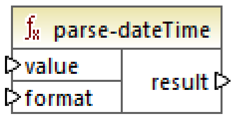
The result is as follows (excluding the XML and namespace declarations):

```
<book>
   <title>Linux Bible</title>
   <pubdate>2015-04-01</pubdate>
</book>
```

See also Example: Mapping Data from an RSS Feed 619.

6.7.2.8 parse-dateTime

Converts a date and time value expressed as a string into a value of type `xs:dateTime`.



Languages

Built-in, C++, C#, Java.

Parameters

Argument	Description
value	The string value to be converted.
format	Specifies the format mask to apply to **value**.

Remarks

A format mask can consist of the following components:

Component	Description	Default Presentation
Y	year (absolute value)	four digits (2010)

Component	Description	Default Presentation
M	month of the year	1-12
D	day of month	1-31
d	day of year	1-366
H	hour (24 hours)	0-23
h	hour (12 hour)	1-12
P	A.M. or P.M.	alphabetic (language dependent)
m	minutes in hour	00-59
s	seconds in minute	00-59
f	fractional seconds	numeric, one decimal place
Z	timezone as a time offset from UTC	+08:00
z	timezone as a time offset using GMT	GMT+n

Some of the components above take modifiers (for example, they can be used to interpret a date either as a single digit or as two digits):

Modifier	Description	Example
1	decimal numeric format with no leading zeros: 1, 2, 3, ...	1, 2, 3
01	decimal format, two digits: 01, 02, 03, ...	01, 02, 03
N	name of component, upper case	FEBRUARY, MARCH
n	name of component, lower case	february, march
Nn	name of component, title case	February, March

Note: *N*, *n*, and *Nn* modifiers support only the component *M* (month).

The width modifier, if necessary, is introduced by a comma, followed by a digit that expresses the minimum width. Optionally, you can add a dash followed by another digit that expresses the maximum width. For example:

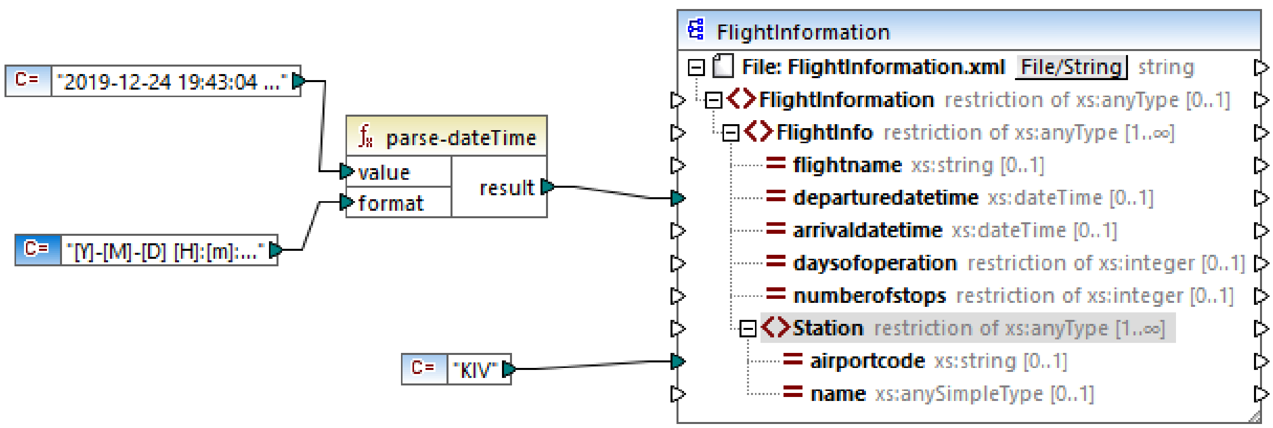
- `[D,2]` is the day of the month, with leading zeros (two digits).
- `[MNn,3-3]` is the name of the month, written as three characters, e.g. *Jan*, *Feb*, *Mar*, and so on.

The table below lists some format examples:

Value	Format	Result
21-03-2002 16:21:12.492 GMT+02:00	`[D]-[M]-[Y] [H]:[m]:[s].[f] [z]`	2002-03-21T16:21:12.492+02:00
315 2004 +01:00	`[d] [Y] [Z]`	2004-11-10T00:00:00+01:00
1.December.10 03:2:39 p.m. +01:00	`[D].[MNn].[Y,2-2] [h]:[m]:[s] [P] [Z]`	2010-12-01T15:02:39+01:00
20110620	`[Y,4-4][M,2-2][D,2-2]`	2011-06-20T00:00:00

Example

In the mapping below, the string value **`2019-12-24 19:43:04 +02:00`** is converted into its `dateTime` equivalent, by applying the format mask **`[Y]-[M]-[D] [H]:[m]:[s] [Z]`**.



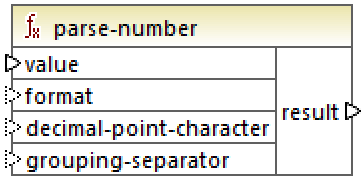
The result is as follows (excluding the XML and namespace declarations):

```
<FlightInformation>
   <FlightInfo departuredatetime="2019-12-24T19:43:04+02:00">
      <Station airportcode="KIV"/>
   </FlightInfo>
</FlightInformation>
```

See also Example: Write Data to Protocol Buffers 821.

6.7.2.9 parse-number

The **`parse-number`** function (*see below*) converts a string into a decimal number according to a specified pattern. The function uses the pattern to determine only the prefix, suffix and digit grouping. The actual length of the number is not checked against the pattern. If the input value is longer or shorter than specified by the pattern, this will be ignored in the checking.



Languages

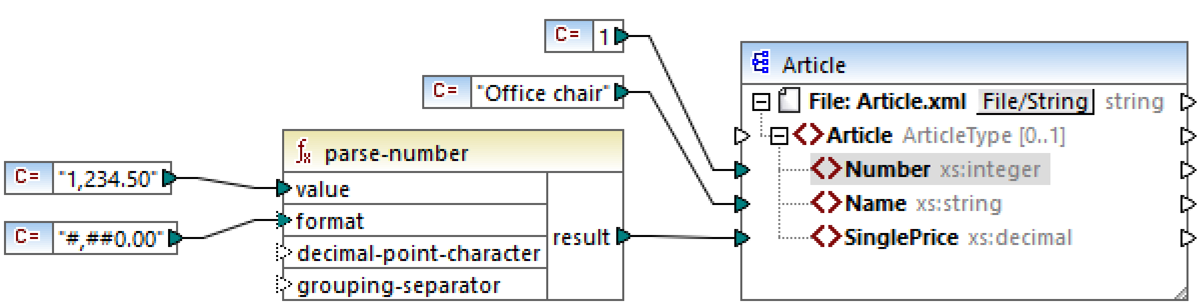
Built-in, C++, C#, Java.

Parameters

Argument	Description
`value`	The string to be converted to a number.
`format`	Optional argument. A format string that identifies the way in which the number is currently formatted. The format string is the same as that used in format-number 993. Default is "#,##0.#"
`decimal-point-character`	Optional argument. Specifies the character to be used as the decimal point character. Default is the '.' character.
`grouping-separator`	Optional argument. Specifies the separator/delimiter used to separate groups of numbers. Default is the "," character.

Example

The following mapping parses the string value **`"1,234.50"`** to a decimal equivalent, by using the format mask **`#,##0.00`**. In this mapping , there is no need to connect the `decimal-point-character` and `grouping-separator` arguments, since their default values match the format of the input string.



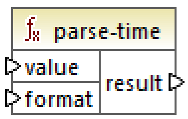
The output (excluding the XML and namespace declarations) is shown below:

```
<Article>
```

```
  <Number>1</Number>
  <Name>Office chair</Name>
  <SinglePrice>1234.5</SinglePrice>
</Article>
```

6.7.2.10 parse-time

Converts a string into an `xs:time` value. This function uses the parse-dateTime 999 function as a basis, while ignoring the date component.



Languages

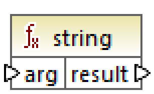
Built-in, C++, C#, Java.

Parameters

Argument	Description
value	Mandatory argument. Supplies the string value to be converted.
format	Mandatory argument. Supplies a format string. This argument is used in the same way as the **format** argument in the parse-dateTime 999 function.

6.7.2.11 string

Converts an input value into a string. The function can also be used to retrieve the text content of a node. If the input node is an XML complex type, then all descendants are also output as a single string.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

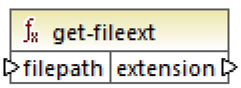
Argument	Description
arg	Mandatory argument. Supplies the value to be converted.

6.7.3 core | file path functions

The **file path** functions allow you to directly access and manipulate file path data, such as folders, file names, and extensions for further processing in your mappings. They can be used in all languages supported by MapForce.

6.7.3.1 get-fileext

Returns the extension of the file path including the dot "." character.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

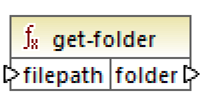
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.

Example

If you supply "c:\data\Sample.mfd" as argument, the result is `.mfd`.

6.7.3.2 get-folder

Returns the folder name of the file path including the trailing slash, or backslash character.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

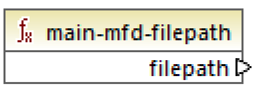
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.

Example

If you supply "c:\data\Sample.mfd" as argument, the result is `c:\data\`.

6.7.3.3 main-mfd-filepath

Returns the full path of the mapping design file (.mfd) containing the main mapping. An empty string is returned if the .mfd is currently not saved.

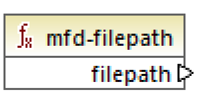


Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

6.7.3.4 mfd-filepath

If the function is called in the main mapping, it returns the same as the main-mfd-filepath 1005 function, i.e. the full path of the .mfd file containing the main mapping. An empty string is returned if the .mfd file is currently not saved. If called within a user-defined function which is *imported* by an .mfd file, it returns the full path of the *imported* .mfd file that contains the definition of the user-defined function.

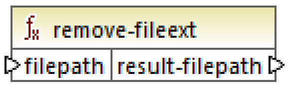


Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

6.7.3.5 remove-fileext

Removes the extension of the file path, including the dot character.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

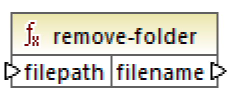
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.

Example

If you supply "c:\data\Sample.mfd" as argument, the result is `c:\data\Sample`.

6.7.3.6 remove-folder

Removes the directory of the file path, including the trailing slash, or backslash character.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

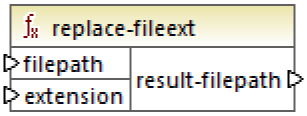
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.

Example

If you supply "c:\data\Sample.mfd" as argument, the result is `Sample.mfd`.

6.7.3.7 replace-fileext

Replaces the extension of the file path supplied by the **filepath** parameter with the one supplied by the connection to the **extension** parameter.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

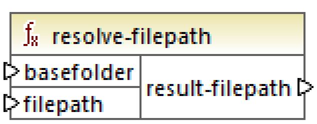
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.
extension	Mandatory argument. Supplies the new extension to use.

Example

If you supply "c:\data\Sample.log" as **filepath**, and ".txt" as **extension**, the result is `c:\data\Sample.txt`.

6.7.3.8 resolve-filepath

Resolves a relative file path against a base folder. The function supports '.' (current directory) and '..' (parent directory).



Languages

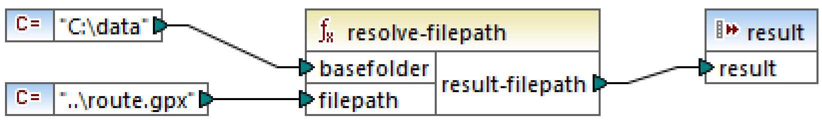
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
basefolder	Mandatory argument. Supplies the base directory relative to which the path should be resolved. This can be an absolute or relative path.
filepath	Mandatory argument. Supplies the relative file path to be resolved.

Examples

In the mapping below, the relative file path **..\route.gpx** is resolved against the **C:\data** directory.



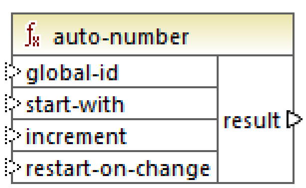
The mapping result is `C:\route.gpx`.

6.7.4 core | generator functions

The **core / generator** functions library includes functions which generate values.

6.7.4.1 auto-number

Generates integer numbers in a sequence (for example, 1,2,3,4, ...). It is possible to set the starting integer, the increment value, and other options by means of parameters.



> The exact order in which functions are called by the generated mapping code is undefined. MapForce may need to cache calculated results for reuse, or evaluate expressions in any order. Also, unlike other functions, the **`auto-number`** function returns a different result when called multiple times with the same input parameters. Therefore, it is strongly recommended to use the **`auto-number`** function cautiously. In some cases, it is possible to achieve the same result by using the position 1049 function instead.

Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

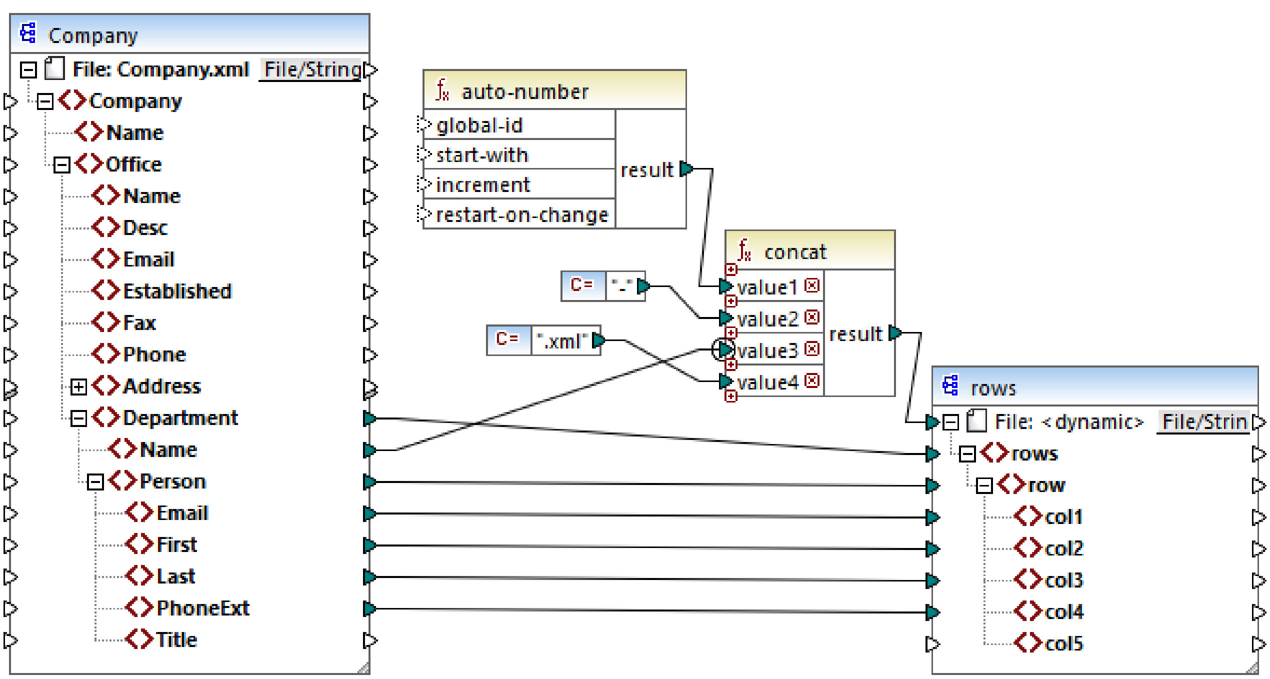
Parameters

Argument	Description
global-id	Optional parameter. If a mapping design contains multiple **auto-number** functions, they will generate sequences with duplicate (overlapping) numbers. To make all **auto-number** functions aware of each other, and thus generate sequences that do not overlap, connect a common string (for example, a constant) to the **global-id** input of each **auto-number** function.
start-with	Optional parameter. Specifies the integer with which the generated sequence begins. The default value is **1**.
increment	Optional parameter. Specifies the increment value. The default value is **1**.
restart-on-change	Optional parameter. Resets the counter to **start-with**, when the content of the connected item changes.

Example

The following mapping is a variation of the **ParentContext.mfd** mapping discussed in the Example: Changing the Parent Context 1284.

The goal of the mapping illustrated below is to generate multiple XML files, one for each department in the source XML file. There are some departments with the same name (that's because they belong to different parent offices). For this reason, each generated file name must begin with a sequential number, for example **1-Administration.xml**, **2-Marketing.xml**, and so on.



To achieve the mapping goal, the **`auto-number`** function was used. The result of this function is concatenated with a dash character, followed by the department name, followed by the ".xml" string in order to create the unique name of the generated file. Importantly, the third parameter of the **`concat`** function (the department name) has a priority context[1288] applied. This has the effect that the **`auto-number`** function is called in the context of each department, and produces the required sequential values. If priority context were not used, the **`auto-number`** function would keep generating number 1 (in the absence of any context), and duplicate file names would be generated as a consequence.

6.7.5 core | logical functions

Logical functions are (generally) used to compare input data and return a Boolean `true` or `false`. They are generally used to test data before passing on a subset to the target component using a filter[878]. Nearly all logical functions have the following structure:

- input parameters: `a | b` or `value1 | value2`
- output parameter: `result`

The evaluation result depends on the input values as well as the data types used for the comparison. For example, the less than comparison of the integer values **`4`** and **`12`** yields the boolean value `true`, since 4 is less than 12. If the two input parameters contain string values **`4`** and **`12`**, the lexical analysis results in the output value `false`, since **`4`** is alphabetically greater than the first character **`1`** of the second operand (**`12`**).

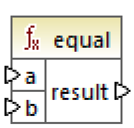
If all input values are of the same data type, then the comparison is done for the common type. If input values are of different types (for example, `integer` and `string`, or `string` and `date`), then the data type used for the comparison is the most general (least restrictive) of the two.

Before comparing two values of different types, all input values are converted to a common data type. Using the previous example, the data type `string` is less restrictive than `integer`. Comparing the integer value **`4`** with the string **`12`** converts the integer value **`4`** to the string **`4`**, which is then compared with the string **`12`**.

Note: Logical functions cannot be used to test the existence of null values. If you supply a null value as an argument to a logical function, it returns a null value. For more information about handling null values, see Nil Values / Nillable[129].

6.7.5.1 equal

The **`equal`** function (*see screenshot below*) returns Boolean `true` if **`a`** is the same as **`b`**; `false` otherwise. The comparison is case-sensitive.



Example:

```
a = hi
b = hi
```

In this example, both values are the same. Therefore, the result is `true`. If, for instance, `b` equaled `Hi`, the function would return `false`.

Languages

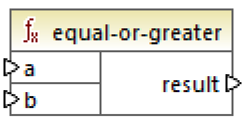
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	**Description**
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

6.7.5.2 equal-or-greater

Returns Boolean **true** if *a* is equal to or greater than *b*; **false** otherwise.



Languages

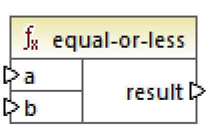
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	**Description**
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

6.7.5.3 equal-or-less

Returns Boolean **true** if *a* is equal to or less than *b*; **false** otherwise.



Languages

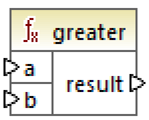
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

6.7.5.4 greater

Returns Boolean **true** if *a* is greater than *b*; **false** otherwise.



Languages

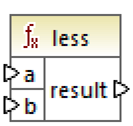
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

6.7.5.5 less

Returns Boolean **true** if *a* is less than *b*; **false** otherwise.



Languages

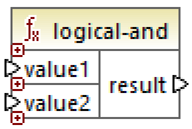
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

6.7.5.6 logical-and

Returns Boolean **true** only if each input value is true; **false** otherwise. You can connect the result to another **`logical-and`** function and thus join an arbitrary number of conditions with logical AND, in order to test that they all return **true**. Also, this function can be extended to take additional arguments, see Add or Delete Function Arguments 906.



Languages

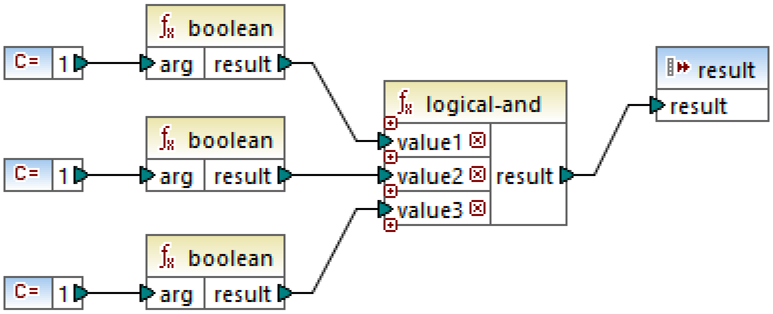
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first value to compare.
value2	Mandatory parameter. Provides the second value to compare.

Example

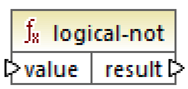
The mapping illustrated below returns **true** because all input values to the **`logical-and`** function are **true** as well. If any of the input values were **false**, then the mapping's result would be **false** as well.



See also [Example: Look-up and Concatenation](939)939.

6.7.5.7 logical-not

Inverts or flips the logical result of the input value. For example, if *value* is **true**, the function's result is false. If *value* is **false**, then result is true.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

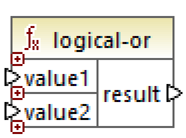
Parameters

Argument	Description
value	Mandatory parameter. Provides the input value.

6.7.5.8 logical-or

This function requires both input values to be Boolean. If at least one of the input values is **true**, then the result is **true**. Otherwise, the result is **false**.

This function can be extended to take additional arguments, see [Add or Delete Function Arguments](906)906.



Languages

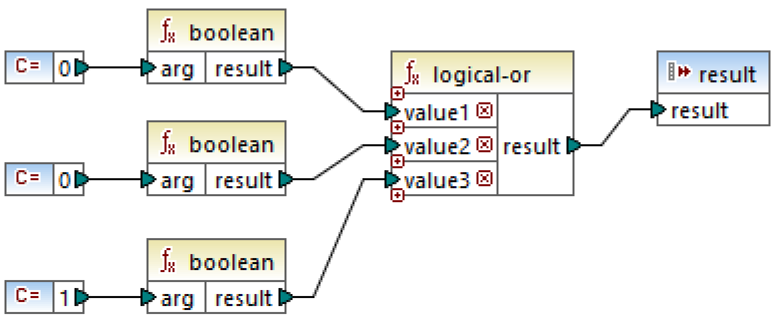
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first value to compare.
value2	Mandatory parameter. Provides the second value to compare.

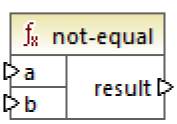
Example

The result of the mapping below is **true**, because at least one of the function's arguments is **true**.



6.7.5.9 not-equal

Returns Boolean **true** if *a* is not equal to *b*; **false** otherwise.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

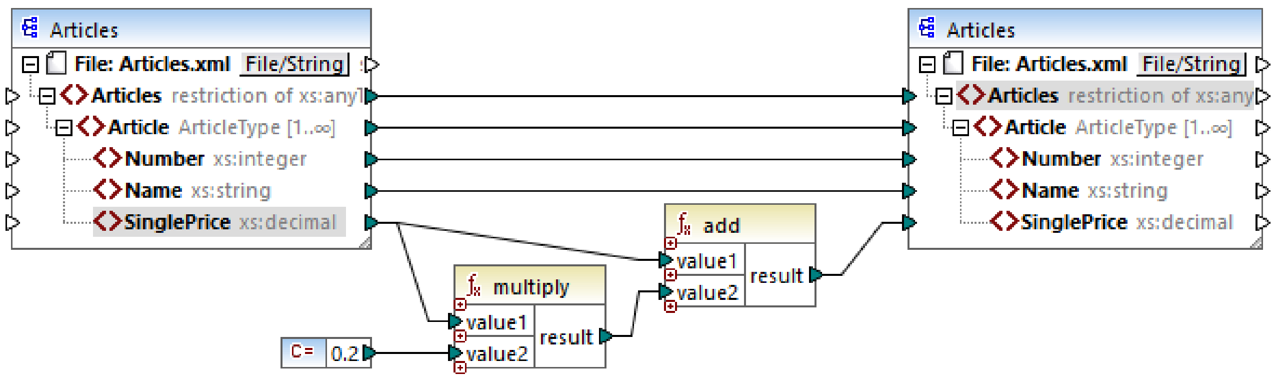
Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

6.7.6 core | math functions

Math functions are used to perform basic mathematical operations on data. Note that they cannot be used to perform computations on durations or `datetime` values.

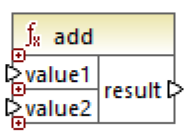
Most math functions take two input parameters (**value1**, **value2**) that are operands of the mathematical operation. The input values are automatically converted to `decimal` type for further processing. The result of math functions is also of `decimal` type.



The example shown above adds 20% sales tax to each of the articles mapped to the target component.

6.7.6.1 add

Adds **value1** to **value2** and returns the result as a decimal value. This function can be extended to take additional arguments, see Add or Delete Function Arguments 906.



Languages

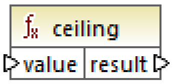
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

6.7.6.2 ceiling

Returns the smallest integer that is greater than or equal to **value**.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

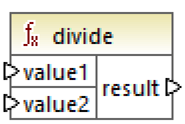
Argument	Description
value	Mandatory parameter. Provides the function's input value.

Example

If the input value is **11.2**, then applying the **ceiling** function to it makes the result **12**, i.e. the smallest integer that is greater than **11.2**.

6.7.6.3 divide

Divides **value1** by **value2** and returns the result as decimal value. The result precision depends on the target language. Use the round-precision [1020] function to define the precision of result.



Languages

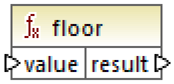
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

6.7.6.4 floor

Returns the greatest integer that is less than or equal to **value**.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

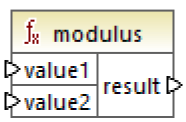
Argument	Description
value	Mandatory parameter. Provides the function's input value.

Example

If the input value is **11.7**, then applying the **`floor`** function to it makes the result **11**, i.e. the greatest integer than is less than **11.7**.

6.7.6.5 modulus

Returns the remainder of dividing **value1** by **value2**.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

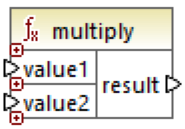
Example

If the input values are **1.5** and **1**, then the result of the `modulus` function is **0.5**. The explanation is that `1.5 / 1` leaves a remainder of **0.5**.

If the input values are **9** and **3**, then the result is **0**, since `9 / 3` leaves no remainder.

6.7.6.6 multiply

Multiplies **value1** by **value2** and returns the result as a decimal value.



Languages

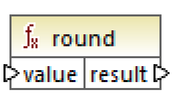
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

6.7.6.7 round

Returns the value rounded to the nearest integer. When the value is exactly in between two integers, the "Round Half Towards Positive Infinity" algorithm is used. For example, the value "10.5" gets rounded to "11", and the value "-10.5" gets rounded to "-10".



Languages

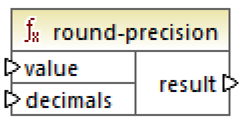
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value	Mandatory parameter. Provides the function's input value.

6.7.6.8 round-precision

Rounds the input value to *N* decimal places, where *N* is the **decimals** argument.



Languages

Built-in, C++, C#, Java.

Parameters

Argument	Description
value	Mandatory parameter. Provides the function's input value.
decimals	Mandatory parameter. Specifies the number of decimals to round to.

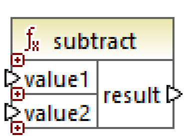
Example

Rounding the value **2.777777** to 2 decimals yields **2.78**. Rounding the value **0.1234** to 3 decimals yields **0.123**.

See also Example: Read Data from Protocol Buffers [816].

6.7.6.9 subtract

Subtracts **value2** from **value1** and returns the result as decimal value.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

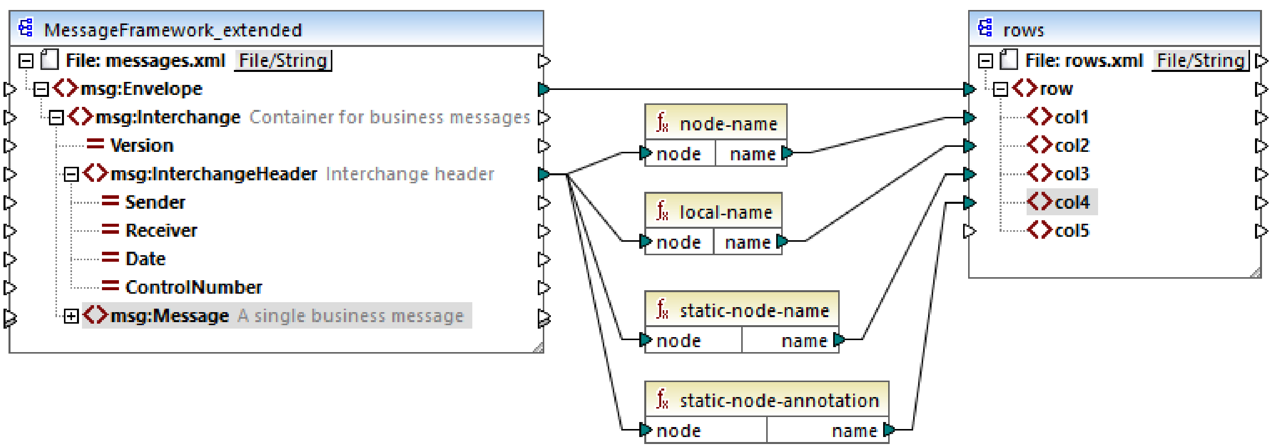
6.7.7 core | node functions

The functions from the **core | node functions** library allow you to access information about nodes on a mapping component (such as the node name or annotation), or to process nillable elements, see also Nil Values / Nillable[129].

Be aware that there is an alternative way to access node names, which does not require node functions at all, see Mapping Node Names[1255].

The mapping illustrated below shows a few node functions that get information from the **msg:InterchangeHeader** node of the source XML file. More specifically, the following information is extracted:

1. The **node-name** function returns the qualified name of the node, which includes the node prefix.
2. The **local-name** function returns just the local part.
3. The **static-node-name** function is similar to the **node-name** function, but is available in XSLT 1.0 as well.
4. The **static-node-annotation** function gets the element's annotation as it was defined in the XML schema.



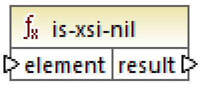
The output of the mapping is as follows (excluding the XML and namespace declarations):

```
<row>
   <col1>msg:InterchangeHeader</col1>
   <col2>InterchangeHeader</col2>
   <col3>msg:InterchangeHeader</col3>
```

```
    <col4>Interchange header</col4>
</row>
```

6.7.7.1 is-xsi-nil

Returns **true** if the **element** node has the `xsi:nil` attribute set to **true**.



Languages

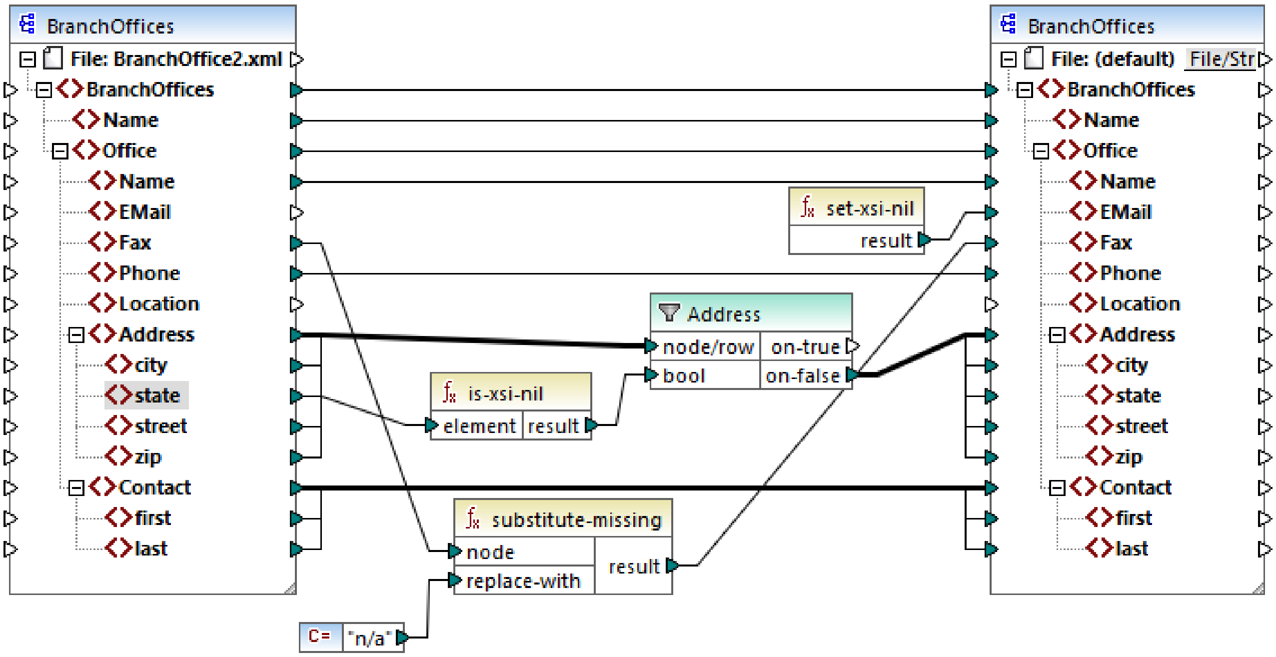
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
element	Mandatory parameter. Must be connected to the source node that is to be checked.

Example

The mapping design illustrated below copies data from a source to a target XML file conditionally, and also illustrates the usage of several functions, including **`is-xsi-nil`**. This mapping is called **HandlingXsiNil.mfd** and can be found in the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.



As illustrated above, the **`is-xsi-nil`** function checks whether the `xsi:nil` attribute is "true" for the **state** item in the source file. If this attribute is "false", the filter will copy the parent **Address** element to the target. The source XML file looks as follows (excluding the XML and namespace declarations):

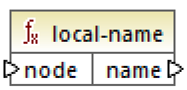
```
<BranchOffices>
   <Name>Nanonull</Name>
   <Office>
      <Name>Nanonull Research Outpost</Name>
      <EMail>sp@nanonull.com</EMail>
      <Fax xsi:nil="true"/>
      <Phone>+8817 3141 5926</Phone>
      <Address>
         <city>South Pole</city>
         <state xsi:nil="true"/>
         <street xsi:nil="true"/>
         <zip xsi:nil="true"/>
      </Address>
      <Contact>
         <first>Scott</first>
         <last>Amundsen</last>
      </Contact>
   </Office>
</BranchOffices>
```

The result of the mapping is that no **Address** is copied to the target at all, because there is only one **Address** in the source, and the `xsi:nil` attribute is set to "true" for the **state** element. Consequently, the mapping output is as follows:

```xml
<BranchOffices>
   <Name>Nanonull</Name>
   <Office>
      <Name>Nanonull Research Outpost</Name>
      <EMail xsi:nil="true"/>
      <Fax>n/a</Fax>
      <Phone>+8817 3141 5926</Phone>
      <Contact>
         <first>Scott</first>
         <last>Amundsen</last>
      </Contact>
   </Office>
</BranchOffices>
```

6.7.7.2 local-name

Returns the local name of the node. Unlike the node-name[1024] function, **local-name** does not return the node's prefix. If the node does not have a prefix, then **local-name** and **node-name** return the same value.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

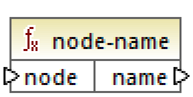
Parameters

Argument	Description
node	Mandatory parameter. Connect this input to the node whose name you want to get.

6.7.7.3 node-name

Returns the qualified name (QName) of the connected node. If the node is an XML **text()** node, an empty QName is returned. This function works only on those nodes that have a name. If XSLT 2.0 is the target language (which calls **fn:node-name**), the function returns an empty sequence for nodes which have no names.

Note: Getting the node name is not supported for "File input" nodes, database tables or fields, XBRL, Excel, JSON, or Protocol Buffers fields.



Languages

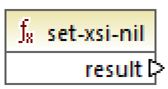
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
node	Mandatory parameter. Connect this input to the node whose name you want to get.

6.7.7.4 set-xsi-nil

Sets the target node to xsi:nil.



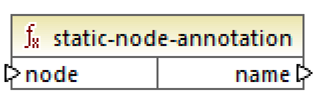
Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

6.7.7.5 static-node-annotation

Returns the string with annotation of the connected node. The input must be: (i) a source component node, or (ii) a user-defined function of type "inline[930]" that is directly connected to a parameter[933], which in turn is directly connected to a node in the calling mapping.

The connection must be direct. It cannot pass through a filter or a regular (not "inline") user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

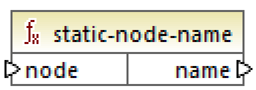
Parameters

Argument	Description
node	Mandatory parameter. Connect this input to the node whose annotation you want to get.

6.7.7.6 static-node-name

Returns the string with the name of the connected node. The input must be: (i) a source component node, or (ii) a user-defined function of type "inline[930]" that is directly connected to a parameter[933], which in turn is directly connected to a node in the calling mapping.

The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.



Languages

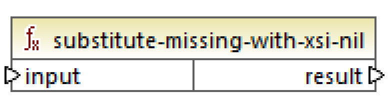
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
node	Mandatory parameter. Connect this input to the node whose name you want to get.

6.7.7.7 substitute-missing-with-xsi-nil

For nodes with simple content, this function substitutes any missing (or null values) of the source component, with the `xsi:nil` attribute in the target node.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

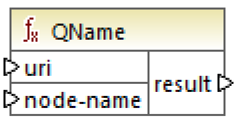
Argument	Description
input	Mandatory parameter. Connect this input to the node whose name you want to get.

6.7.8 core | QName functions

QName functions provide ways to manipulate the Qualified Names (QName) in XML documents.

6.7.8.1 QName

Constructs a QName from a namespace URI and a local part. Use this function to create a QName in a target component. The **uri** and **node-name** parameters can be supplied by a constant function.



Languages

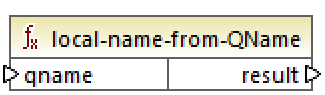
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
uri	Mandatory. Provides the URI.
node-name	Mandatory. Provides the name of the node.

6.7.8.2 local-name-from-QName

Extracts the local name part from a value of type `xs:QName`. Note that, unlike the local-name [1024] function which returns the local name of the *node*, this function processes the *content* of the item connected to the **qname** input.



Languages

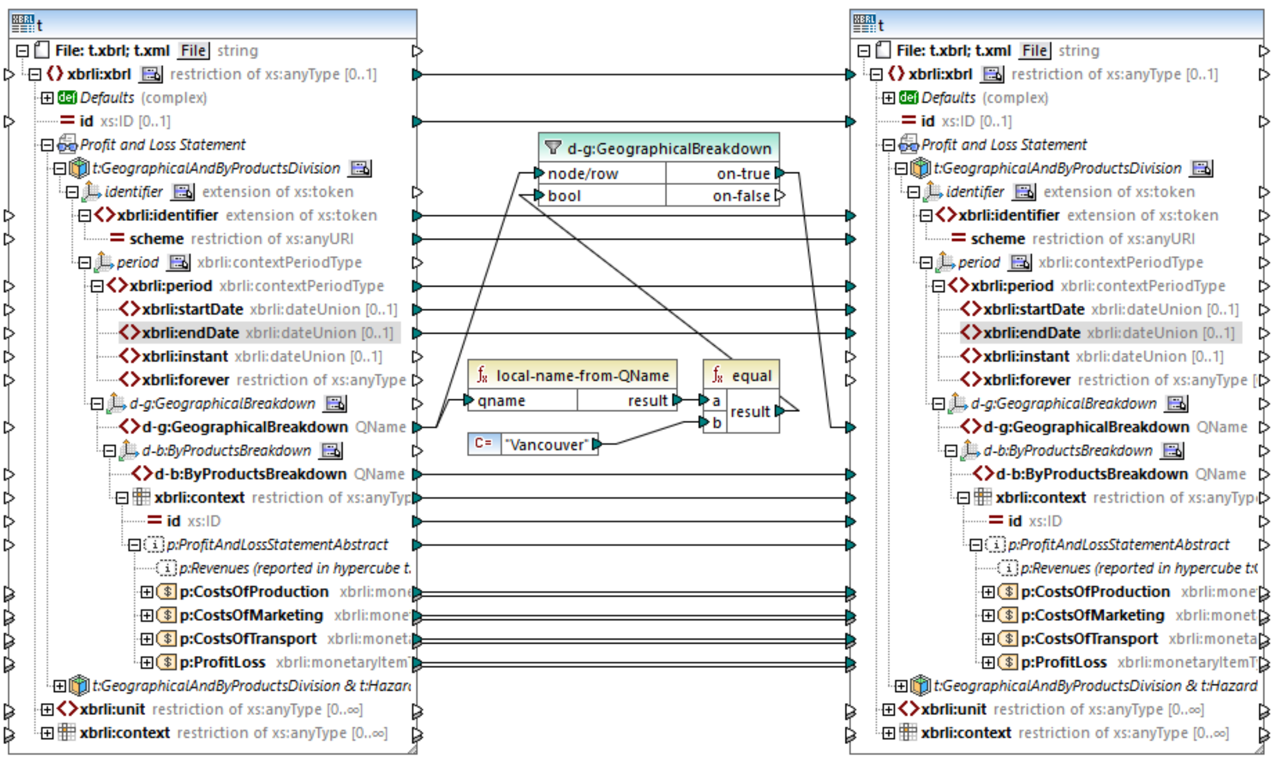
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
qname	Mandatory. Provides the function's input value, of type `xs:QName`.

Example

This function is useful when mapping XBRL instance documents containing hypercubes.



The mapping above filters those facts where the local name of the *content* of the explicit member **d-g:Vancouver** is equal to "Vancouver". Note that the content of the member is itself a QName, as illustrated in the code listing below:

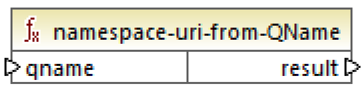
```
<xbrli:entity>
    <xbrli:identifier scheme="http://nasdaq.com/ticker">ACME2</xbrli:identifier>
    &lt;xbrli:segment&gt;
        <xbrldi:explicitMember dimension="d-b:ByProductsBreakdown">d-
b:Water</xbrldi:explicitMember>
        <xbrldi:explicitMember dimension="d-g:GeographicalBreakdown">d-
```

```
g:Vancouver</xbrldi:explicitMember>
    &lt;/xbrli:segment&gt;
</xbrli:entity>
```

After running the mapping, all the facts that belong to the dimension **GeographicalBreakdown** are filtered and passed to the target component.

6.7.8.3 namespace-uri-from-QName

Returns the namespace URI part of the QName value supplied as argument.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

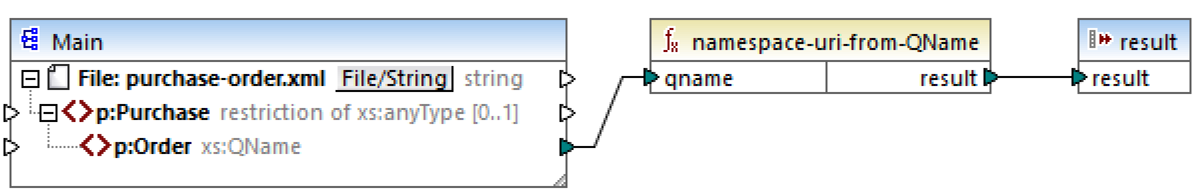
Name	Description
qname	Mandatory. Provides the function's input value.

Example

The following XML file contains a QName value, **o:name**. Note that the prefix "o" is mapped to the namespace **`http://NamespaceTest.com/Order`**.

```
<?xml version="1.0" encoding="utf-8"?>
<p:Purchase xsi:schemaLocation="http://NamespaceTest.com/Purchase Main.xsd"
          xmlns:p="http://NamespaceTest.com/Purchase"
          xmlns:o="http://NamespaceTest.com/Order"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <p:Order>o:name</p:Order>
</p:Purchase>
```

A mapping that processes the QName value and gets the namespace URI is illustrated below:



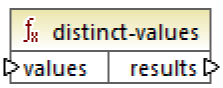
The output of this mapping is `http://NamespaceTest.com/Order`.

6.7.9 core | sequence functions

Sequence functions allow processing of input sequences[1276] and grouping of their content.

6.7.9.1 distinct-values

Processes the sequence of values connected to the **values** input and returns only the distinct values, as a sequence. This is useful when you need to remove duplicate values from a sequence and copy only the unique items to the target component.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
values	This input must receive a connection from a mapping item that provides a sequence[1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.

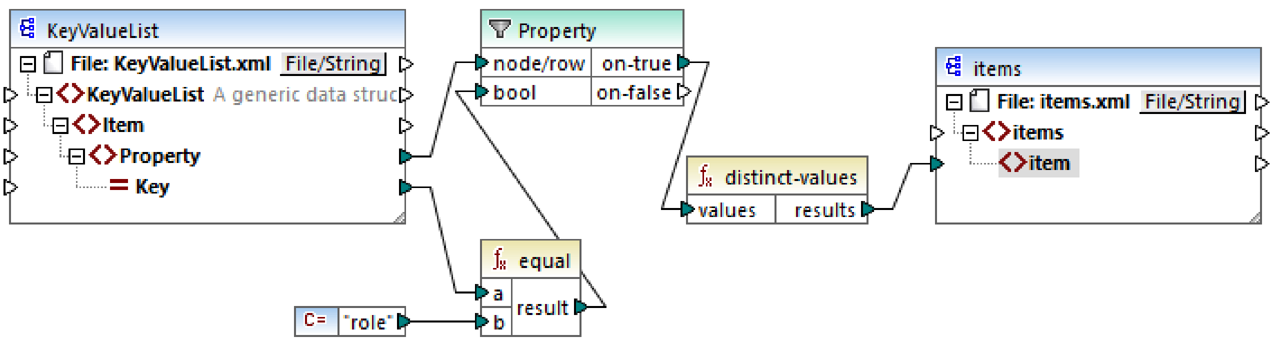
Example

The following XML file contains information about employees of a demo company. Some employees have the same role; therefore, the "role" attribute role contains duplicate values. For example, both "Loby Matise" and "Susi Sanna" have the role "Support".

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyValueList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="KeyValueList.xsd">
   <Item>
      <Property Key="role">Manager</Property>
      <Property Key="First">Vernon</Property>
      <Property Key="Last">Callaby</Property>
   </Item>
   <Item>
      <Property Key="role">Programmer</Property>
      <Property Key="First">Frank</Property>
```

```
      <Property Key="Last">Further</Property>
   </Item>
   <Item>
      <Property Key="role">Support</Property>
      <Property Key="First">Loby</Property>
      <Property Key="Last">Matise</Property>
   </Item>
   <Item>
      <Property Key="role">Support</Property>
      <Property Key="First">Susi</Property>
      <Property Key="Last">Sanna</Property>
   </Item>
</KeyValueList>
```

Let's suppose that you need to extract a list of all *unique* role names that occur in this XML file. This can be achieved with a mapping like the one below:



In the mapping above, the following happens:

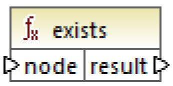
- Each **Property** element from the source XML file is processed by a filter.
- The connection to the filter's **bool** input ensures that only **Property** elements where the **Key** attribute is equal to "role" are supplied to the target component. The string "role" is provided by a constant. Note that the filter's output still produces duplicates at this stage (since there are two "Support" properties that meet the filter's condition).
- The sequence produced by the filter is processed by the `distinct-values` function, which excludes any duplicate values.

As a result, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>Manager</item>
   <item>Programmer</item>
   <item>Support</item>
</items>
```

6.7.9.2 exists

Returns **true** if the connected node exists; **false** otherwise. Since it returns a Boolean value, this function is typically used with filters 878, to filter out only records which have (or perhaps do not have) a child element or attribute.



Languages

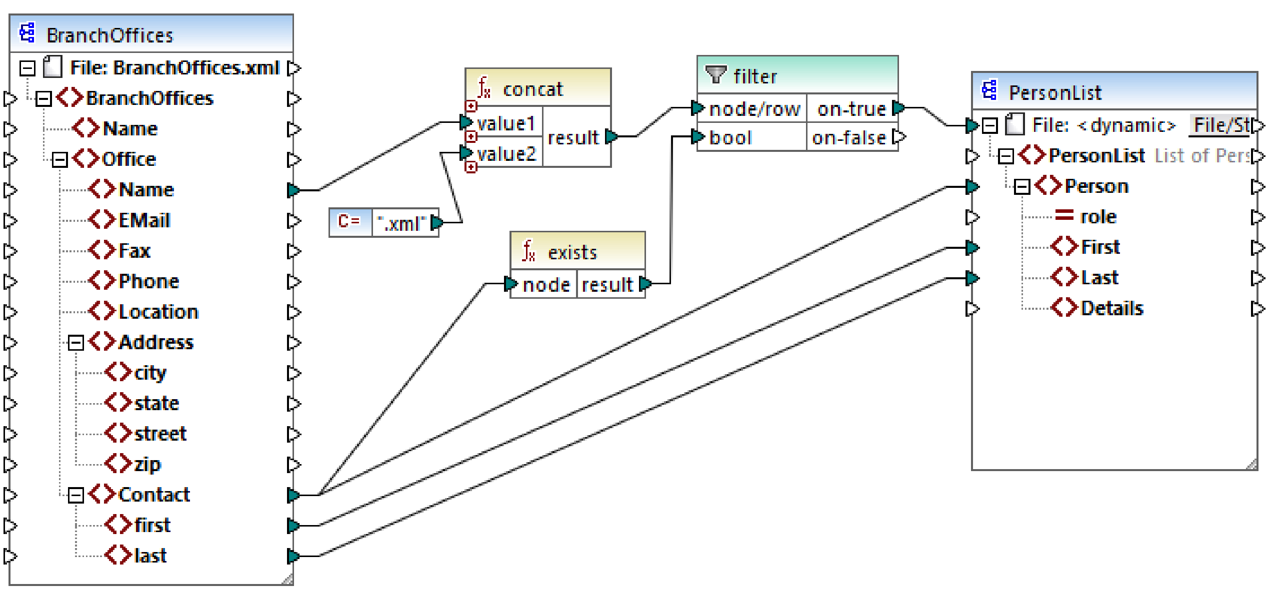
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node	The node to be tested for existence.

Examples

The following mapping illustrates how to filter data with the help of the `exists` function. This mapping is called **PersonListsForAllBranchOffices.mfd** and it can be found in the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.



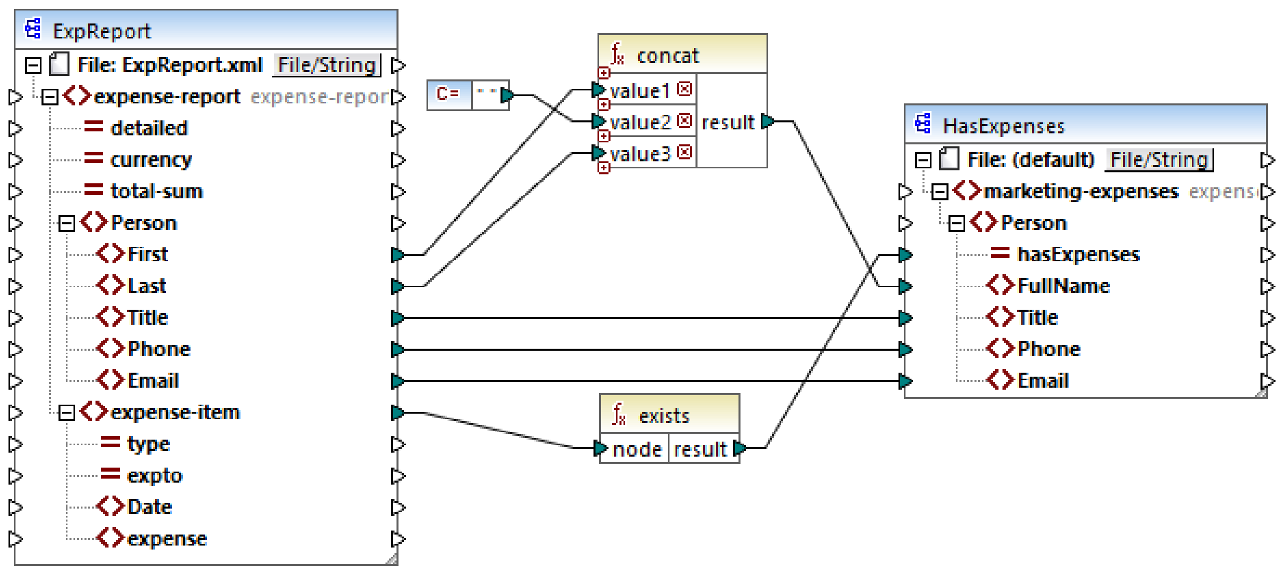
PersonListsForAllBranchOffices.mfd

In the source file **BranchOffices.xml**, there are three **Office** elements. Notably, one of the offices does not have any **Contact** child elements. The goal of the mapping is many-fold:

a) for each office, extract a list of contacts that exist in that office
b) for each office, create a separate XML file with the same name as the office
c) do not generate the XML file if the office has no contacts.

To achieve these goals, a filter was added to the mapping. The filter passes on to the target only those **Office** items where at least one **Contact** item exists. This Boolean condition is provided by the **`exists`** function. If the function's result is true, then the name of the office is concatenated with the string **`.xml`** in order to produce the target file name. For more information about generating file names from the mapping, see Processing Multiple Input or Output Files Dynamically [1313] .

Another example is the following mapping:
<Documents>\Altova\MapForce2023\MapForceExamples\HasMarketingExpenses.mfd. Here, if an **expense-item** exists in the source XML, then the **hasExpenses** attribute is set to **true** in the target XML file.

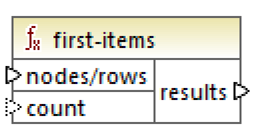


HasMarketingExpenses.mfd

See also Example: Exception When Node Does Not Exist [901].

6.7.9.3 first-items

Returns the first *N* items of the input sequence, where *N* is supplied by the **count** parameter.



Languages

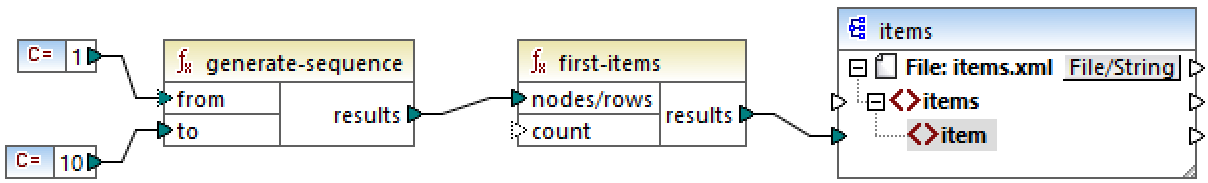
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Optional parameter. Specifies how many items should be retrieved from the input sequence. The default value is **1**.

Example

The following mock-up mapping generates a sequence of 10 values. The sequence is processed by the `first-items` function and the result is written to a target XML file.



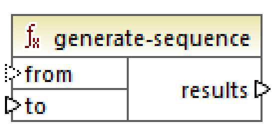
Because the **count** argument has no value, the default value of **1** applies. As a result, only the first value from the sequence is generated in the mapping output:

```
<items>
   <item>1</item>
</items>
```

For a more realistic example, see the **FindHighestTemperatures.mfd** mapping discussed in Supplying Parameters to the Mapping[833].

6.7.9.4 generate-sequence

Creates a sequence of integers using the "from" and "to" parameters as the boundaries.



Languages

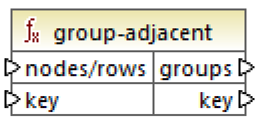
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
from	Optional parameter. Specifies the integer that the sequence should start with (lower boundary). The default value is **1**.
to	Mandatory parameter. Specifies the integer that the sequence should end with (upper boundary).

6.7.9.5 group-adjacent

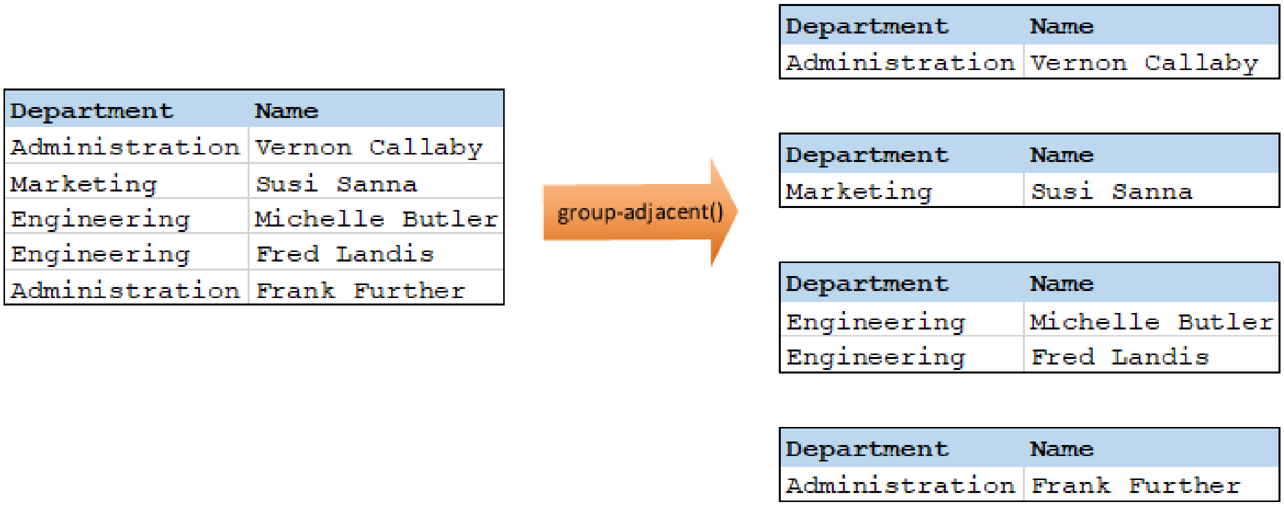
The **`group-adjacent`** function groups the items connected to the **nodes/rows** input by the key connected to the **key** input. Note that this function places items that share the same key into separate groups if they are not adjacent. If multiple consecutive (adjacent) items share the same key, they are placed into the same group.



For example, in the abstract transformation illustrated below, the grouping key is "Department". The left side of the diagram shows the input data while the right side shows the output data after grouping. The following takes place when the transformation runs:

- Initially, the first key, "Administration", creates a new group.
- The next key is different, so a second group is created, "Marketing".
- The third key is also different, so another group is created, "Engineering".
- The fourth key is the same as the third; therefore, this record is placed in the already existing group.
- Finally, the fifth key is different from the fourth, and this creates the last group.

As illustrated below, "Michelle Butler" and "Fred Landis" were grouped together because they have the same key and are adjacent. However, "Vernon Callaby" and "Frank Further" are in separate groups because they are not adjacent, even though they have the same key.



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

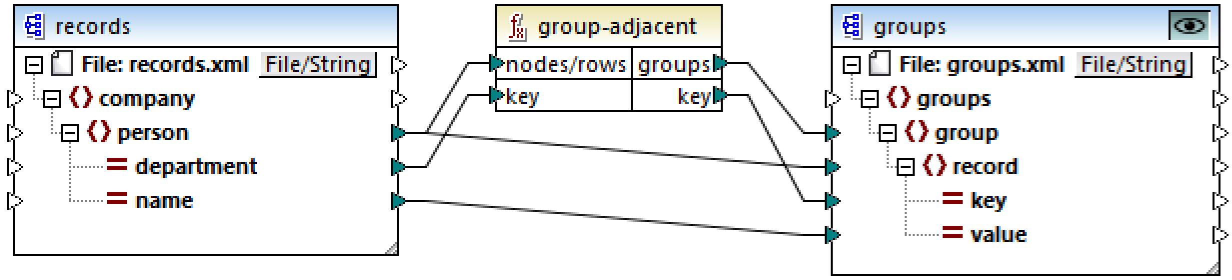
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence 1276 of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
key	The key by which to group items.

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records by department, provided they are adjacent. To achieve this, the following mapping invokes the `group-adjacent` function, and supplies **department** as **key**.



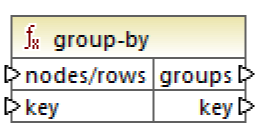
The mapping result is as follows:

```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
   </group>
   <group>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
   </group>
   <group>
      <record key="Administration" value="Frank Further"/>
   </group>
</groups>
```

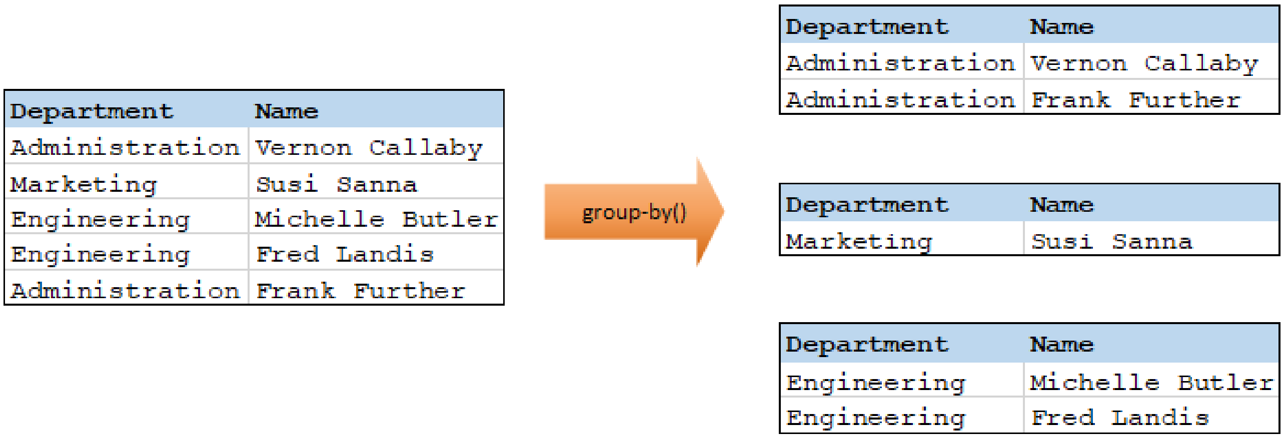
This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview** button applicable to the function you want to preview, before clicking the **Output** tab.

6.7.9.6 group-by

The **group-by** function creates groups of records according to some grouping key that you specify.



For example, in the abstract transformation illustrated below, the grouping key is "Department". Since there are three unique departments in total, applying the group-by function would create three groups:



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

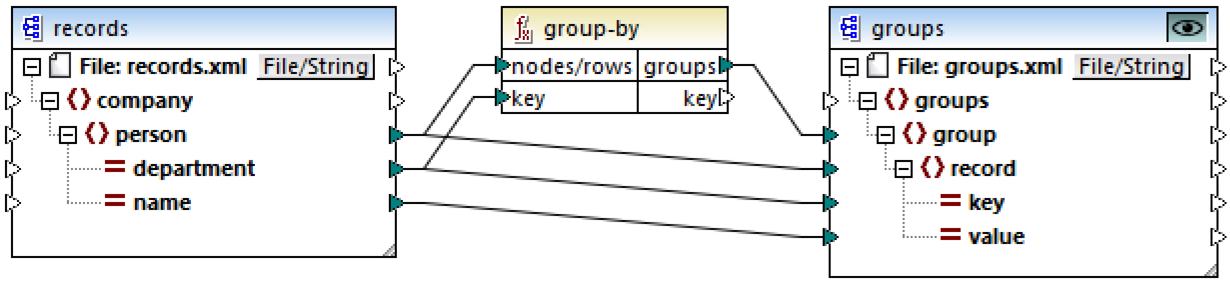
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence [1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
key	The key by which to group items.

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records by department. To achieve this, the following mapping invokes the `group-by` function, and supplies **department** as key.



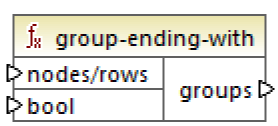
The mapping result is as follows:

```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
      <record key="Administration" value="Frank Further"/>
   </group>
   <group>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
   </group>
</groups>
```

This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview**  button applicable to the function you want to preview, before clicking the **Output** tab.

6.7.9.7 group-ending-with

The **group-ending-with** function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, ending with the record that satisfies the condition.



In the example below, the condition is that "Key" must be equal to "trailing". This condition is true for the third and fifth records, so two groups are created as a result:

Key	Value
line	A
line	B
trailing	Total 1
line	C
trailing	Total 2

group-ending-with()

Key	Value
line	A
line	B
trailing	Total 1

Key	Value
line	C
trailing	Total 2

Note: One additional group is created if records exist after the last one that satisfies the condition. For example, if there were more "line" records after the last "trailing" record, these would all be placed into a new group.

Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

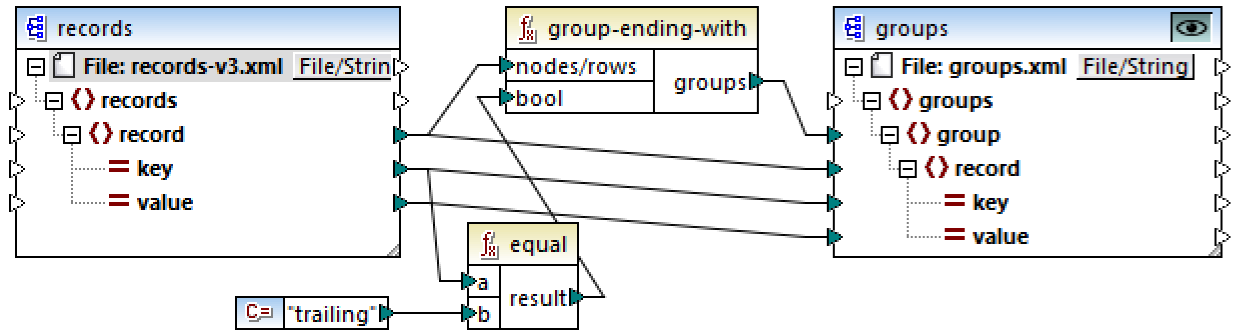
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence 1276 of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
bool	Provides the Boolean condition that starts a new group when **true**.

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<records>
   <record key="line" value="A"/>
   <record key="line" value="B"/>
   <record key="trailing" value="Total 1"/>
   <record key="line" value="C"/>
   <record key="trailing" value="Total 2"/>
</records>
```

The business requirement is to create groups for each "trailing" record. Each group must also include any "line" records that precede the "trailing" record. To achieve this, the following mapping invokes the **group-ending-with** function. In the mapping below, whenever the **key** name is equal to "trailing", the argument supplied to **bool** becomes **true**, and a new group is created.



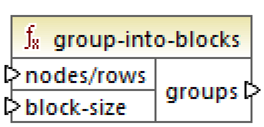
The mapping result is as follows:

```
<groups>
   <group>
      <record key="line" value="A"/>
      <record key="line" value="B"/>
      <record key="trailing" value="Total 1"/>
   </group>
   <group>
      <record key="line" value="C"/>
      <record key="trailing" value="Total 2"/>
   </group>
</groups>
```

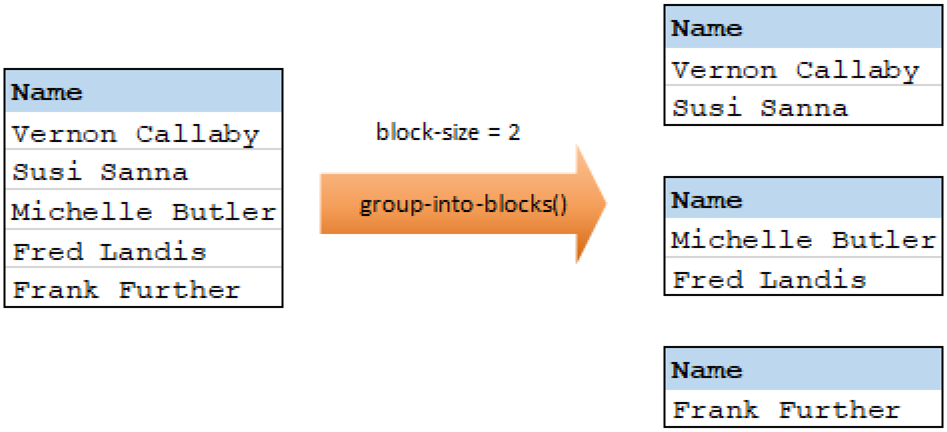
This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview** button applicable to the function you want to preview, before clicking the **Output** tab.

6.7.9.8 group-into-blocks

The **group-into-blocks** function creates equal groups that contain exactly N items, where N is the value you supply to the `block-size` argument. Note that the last group may contain N items or less, depending on the number of items in the source.



In the example below, `block-size` is 2. Since there are five items in total, each group contains exactly two items, except for the last one.



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

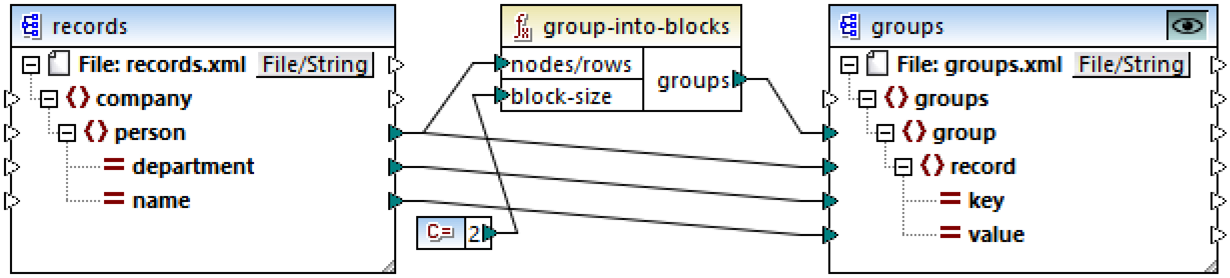
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
block-size	Specifies the size of each group

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records into blocks of two items each. To achieve this, the following mapping invokes the `group-into-blocks` function, and supplies the integer value "2" as **block-size**.



The mapping result is as follows:

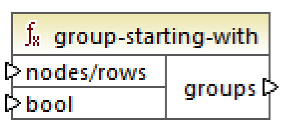
```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
   </group>
   <group>
      <record key="Administration" value="Frank Further"/>
   </group>
</groups>
```

Note that the last group contains only one item, since the total number of items (5) cannot be divided evenly by 2.

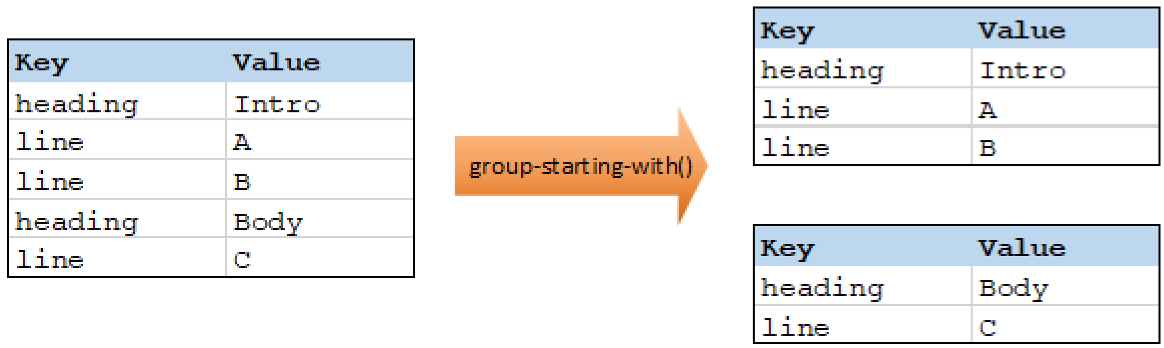
This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview**  button applicable to the function you want to preview, before clicking the **Output** tab.

6.7.9.9 group-starting-with

The **`group-starting-with`** function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, starting with the record that satisfies the condition.



In the example below, the condition is that "Key" must be equal to "heading". This condition is true for the first and fourth records, so two groups are created as a result:



Note: One additional group is created if records exist before the first one that satisfies the condition. For example, if there were more "line" records before the first "heading" record, these would all be placed into a new group.

Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

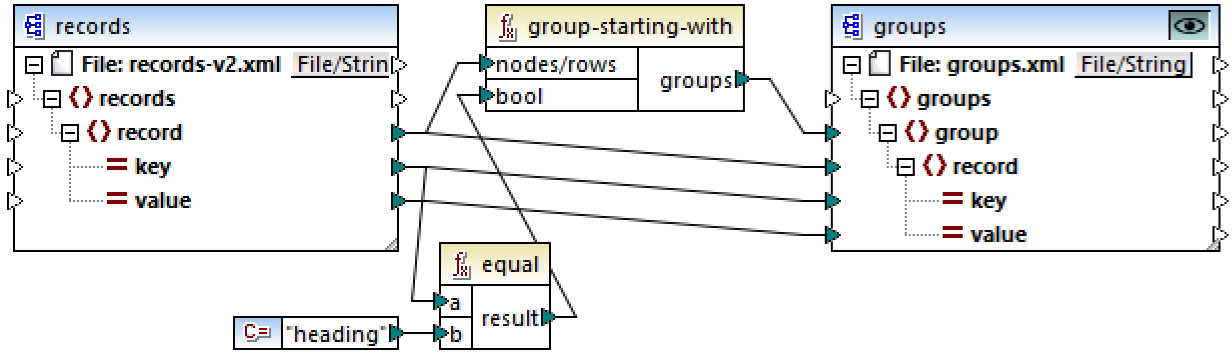
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence [1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
bool	Provides the Boolean condition that starts a new group when **true**.

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<records>
   <record key="heading" value="Intro"/>
   <record key="line" value="A"/>
   <record key="line" value="B"/>
   <record key="heading" value="Body"/>
   <record key="line" value="C"/>
</records>
```

The business requirement is to create groups for each "heading" record. Each group must also include any "line" records that follow the "heading" record. To achieve this, the following mapping invokes the **group-starting-with** function. In the mapping below, whenever the **key** name is equal to "heading", the argument supplied to **bool** becomes **true**, and a new group is created.



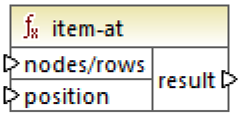
The mapping result is as follows:

```
<groups>
   <group>
      <record key="heading" value="Intro"/>
      <record key="line" value="A"/>
      <record key="line" value="B"/>
   </group>
   <group>
      <record key="heading" value="Body"/>
      <record key="line" value="C"/>
   </group>
</groups>
```

This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview**  button applicable to the function you want to preview, before clicking the **Output** tab.

6.7.9.10 item-at

Returns an item from the sequence of **nodes/rows** supplied as argument, at the position supplied by the **position** argument. The first item is at position **1**.



Languages

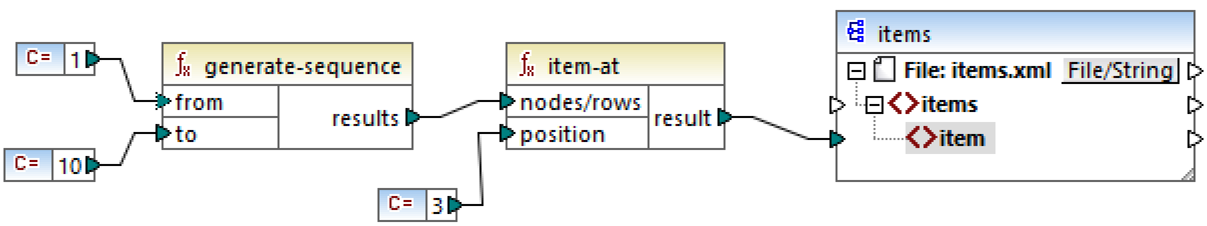
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence [1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
position	This integer specifies which item from the sequence of items is to be returned.

Example

The following mock-up mapping generates a sequence of 10 values. The sequence is processed by the **`item-at`** function and the result is written to a target XML file.

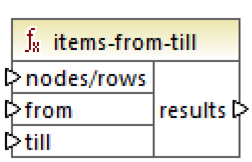


Because the **position** argument is set to **3**, only the third value from the sequence is passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>3</item>
</items>
```

6.7.9.11 items-from-till

Returns a sequence of **nodes/rows** using the "from" and "till" parameters as the boundaries. The first item is at position **1**.



Languages

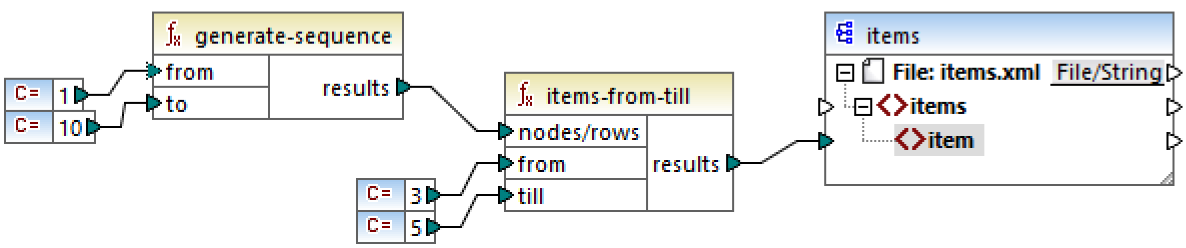
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence1276 of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
from	This integer specifies the starting position from which items must be retrieved.
till	This integer specifies the position up to which items must be retrieved.

Example

The following mock-up mapping generates a sequence of 10 values. The sequence is processed by the `items-from-till` function and the result is written to a target XML file.

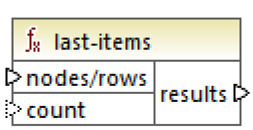


Because the **from** and **till** arguments are set to **3** and **5**, respectively, only the subset of values from **3** through **5** are passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>3</item>
   <item>4</item>
   <item>5</item>
</items>
```

6.7.9.12 last-items

Returns the last *N* items of the input sequence, where *N* is supplied by the **count** parameter. The first item is at position "1".



Languages

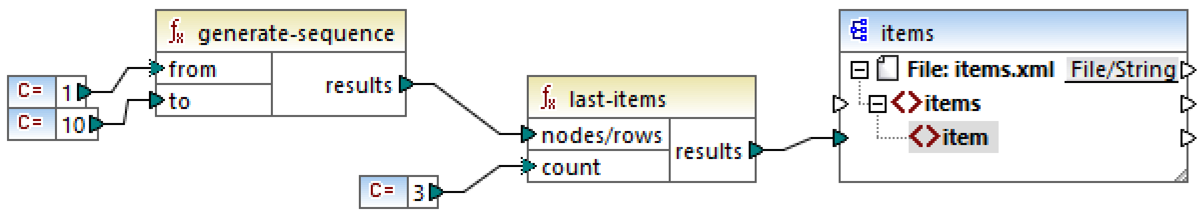
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence [1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Optional parameter. Specifies how many items should be retrieved from the input sequence. The default value is **1**.

Example

The following mock-up mapping generates a sequence of 10 values. The sequence is processed by the **last-items** function and the result is written to a target XML file.

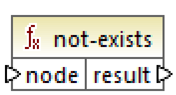


Because the count argument is set to **3**, only the last three values from the sequence are passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>8</item>
   <item>9</item>
   <item>10</item>
</items>
```

6.7.9.13 not-exists

Returns **false** if the connected node exists; **true** otherwise. This function is the opposite of exists [1032] function, but, otherwise, it has the same use.



Languages

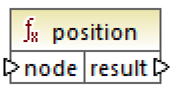
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node	The node to be tested for non-existence.

6.7.9.14 position

Returns the position of an item within the sequence of items currently being processed. This can be used, for example, to auto-number items sequentially.



Languages

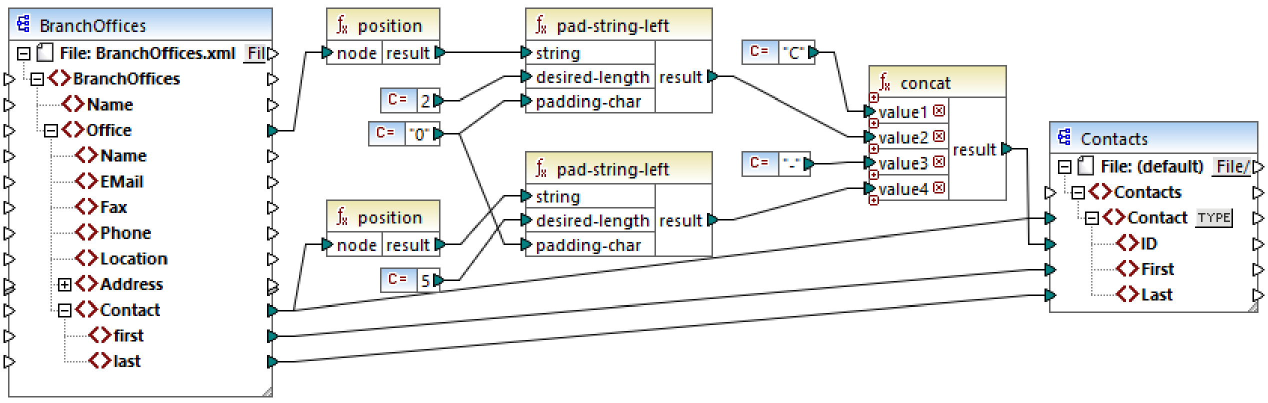
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node	This input must receive a connection from a mapping item that provides a sequence[1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.

Example

The following mapping illustrates using the **`position`** function in order to generate unique identification values in data generated by the mapping. This mapping is accompanied by a mapping design file that is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\ContactsFromBranchOffices.mfd**.



ContactsFromBranchOffices.mfd

In the mapping above, the source XML file contains three branch offices. A branch office may contain an arbitrary number of **Contact** child items. The goals of the mapping are as follows:

- Extract all **Contact** items from the source XML file and write them to the target XML file.
- Each contact must be assigned a unique identification number (the **ID** item in the target XML).
- The ID of each contact must take the form **`CXX-YYYYY`**, where X identifies the office number, and Y identifies the contact number. If the office number is less than two characters, it must be left-padded with zeros. Likewise, if the contact number takes less than five characters, it must be left-padded with zeros. Consequently, a valid identification number of the first contact from the first office should look like **`C01-00001`**.

To achieve the mapping goals, several MapForce functions have been used, including the **`position`** function. The upper **`position`** function gets the position of each office. The lower one gets the position of each contact, in the context of each office.

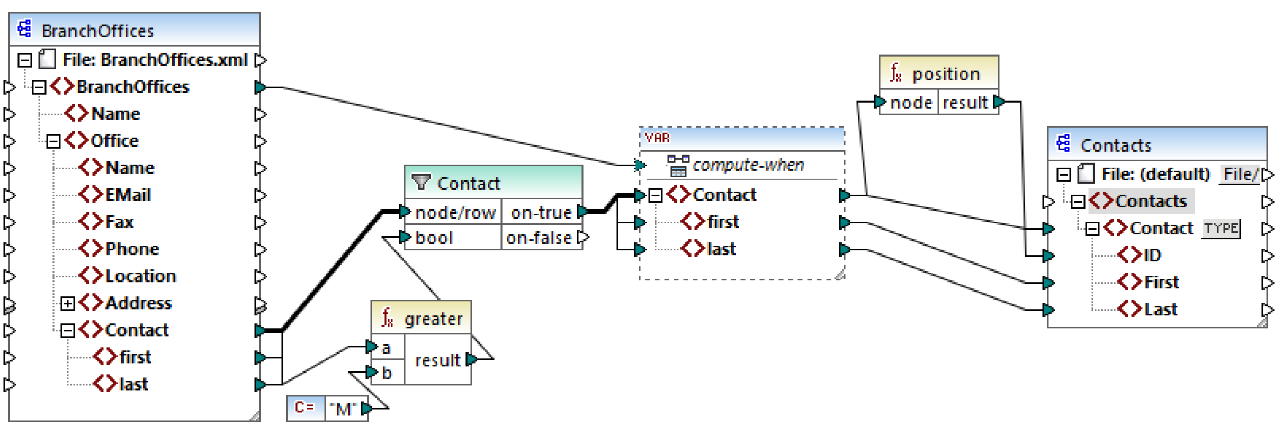
When using the **`position`** function, it is important to consider the current mapping context 1277. More specifically, when the mapping runs, the initial mapping context is established from the root item of the target component to the source item connected to it (even indirectly via functions). In this example, the upper **`position`** function processes *the sequence of all offices* and it initially generates the value 1, corresponding to the first office in the sequence. The lower **`position`** function generates sequential numbers corresponding to the contact's position *in the context of that office* (1, 2, 3, and so on). Note that this "inner" sequence will be reset (and thus start from 1 again) when the next office gets processed. Both **`pad-string-left`** functions apply padding to the generated numbers, according to the requirements stated previously. The `concat` function operates *in the context of each contact* (because of the parent connection from the source to the target **Contact**). It joins all the computed values and returns the unique identification number of each contact.

The output generated from the mapping above is shown below (note that some of the records were removed for readability):

```
<Contacts>
   <Contact>
      <ID>C01-00001</ID>
      <First>Vernon</First>
      <Last>Callaby</Last>
   </Contact>
   <Contact>
```

```
      <ID>C01-00002</ID>
      <First>Frank</First>
      <Last>Further</Last>
   </Contact>
   <!-- ... -->
   <Contact>
      <ID>C02-00001</ID>
      <First>Steve</First>
      <Last>Meier</Last>
   </Contact>
   <Contact>
      <ID>C02-00002</ID>
      <First>Theo</First>
      <Last>Bone</Last>
   </Contact>
   <!-- ... -->
</Contacts>
```

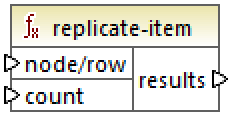
There may also be cases where you need to get the position of items resulting after applying a [filter](878). Note that the filter component is not a sequence function, and it cannot be used *directly* in conjunction with the **position** function to find the position of filtered items. Indirectly, this is possible by adding a [variable](847) component to the mapping. For example, the mapping below is a simplified version of the previous one. Its mapping design file is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\PositionInFilteredSequence.mfd**.



The result of variables in MapForce are always sequences. Therefore, in the mapping above, the **position** function iterates through the sequence created by the variable and returns the position of each item in that sequence. This mapping is discussed in more detail in [Example: Filtering and Numbering Nodes](856).

6.7.9.15 replicate-item

Repeats every item in the input sequence the number of times specified in the **count** argument. If you connect a single item to the **node/row** input, the function returns *N* items, where *N* is the value of the **count** argument. If you connect a sequence of items to the **node/row** input, the function repeats each individual item in the sequence **count** times, processing one item at a time. For example, if count is **2**, then the sequence `1,2,3` produces `1,1,2,2,3,3`. It is also possible to supply a different **count** value for each item in the input sequence, as illustrated in the example below.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

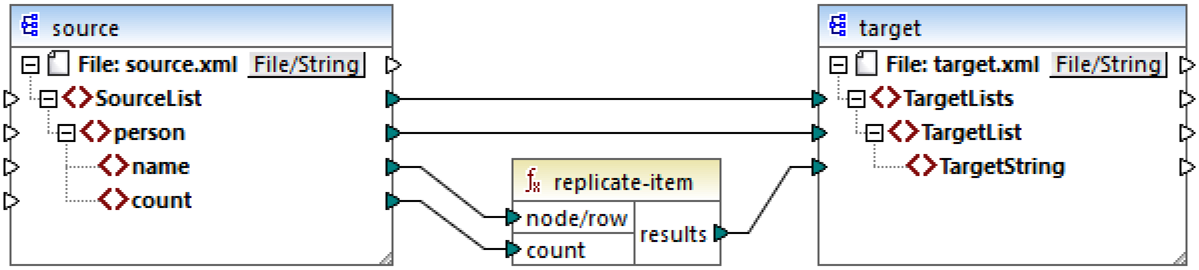
Name	Description
node/row	This input must receive a connection from a mapping item that provides a sequence[1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Specifies the number of times to replicate each item or sequence connected to **node/row**.

Example

Let's assume that you have a source XML file with the following structure:

```
<SourceList>
   <person>
      <name>Michelle</name>
      <count>2</count>
   </person>
   <person>
      <name>Ted</name>
      <count>4</count>
   </person>
   <person>
      <name>Ann</name>
      <count>3</count>
   </person>
</SourceList>
```

With the help of the `replicate-item` function, you can repeat each person name a different number of times in a target component. To achieve this, connect the **count** node of each person to the **count** input of the `replicate-item` function:

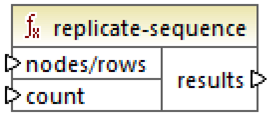


The output is as follows:

```
<TargetLists>
   <TargetList>
      <TargetString>Michelle</TargetString>
      <TargetString>Michelle</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
   </TargetList>
</TargetLists>
```

6.7.9.16 replicate-sequence

Repeats all items in the input sequence the number of times specified in the **count** argument. For example, if count is **2**, then the sequence `1,2,3` produces `1,2,3,1,2,3`.



Languages

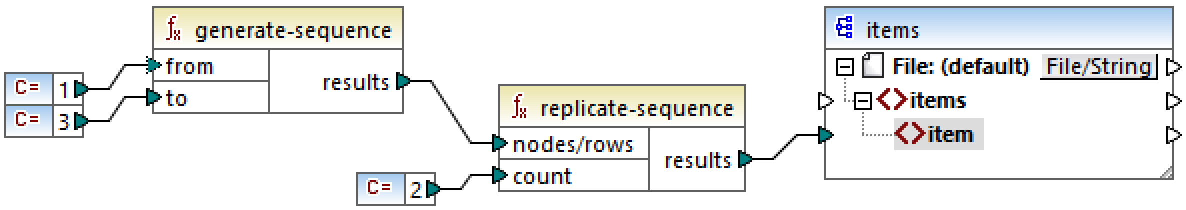
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node/rows	This input must receive a connection from a mapping item that provides a sequence[1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Specifies the number of times to replicate the connected sequence.

Example

The following mock-up mapping generates the sequence `1,2,3`. The sequence is processed by the `replicate-sequence` function and the result is written to a target XML file.

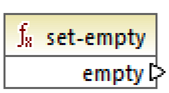


Because the **count** argument is set to **2**, the sequence is replicated twice and then passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>1</item>
   <item>2</item>
   <item>3</item>
   <item>1</item>
   <item>2</item>
   <item>3</item>
</items>
```

6.7.9.17 set-empty

Returns an empty sequence. For example, you can use this function to cancel default values[773] of an XBRL document that were defined higher up the XBRL component/taxonomy.

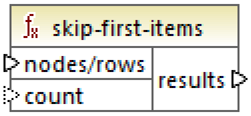


Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

6.7.9.18 skip-first-items

Skips the first N items of the input sequence, where N is supplied by the **count** argument, and returns the rest of the sequence.



Languages

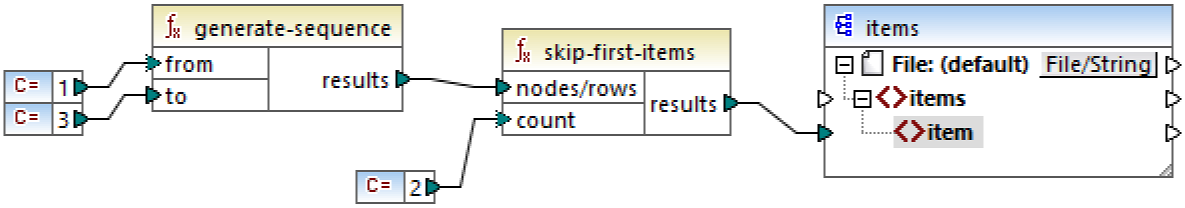
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node/rows	This input must receive a connection from a mapping item that provides a sequence 1276 of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Optional argument. Specifies the number of items to skip. The default value is **1**.

Example

The following mock-up mapping generates the sequence `1,2,3`. The sequence is processed by the `skip-first-items` function and the result is written to a target XML file.

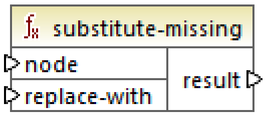


Because the **count** argument is set to **2**, the first two items are skipped and the remaining items are passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>3</item>
</items>
```

6.7.9.19 substitute-missing

This function is a convenient combination of exists[1032] function and if-else condition[882]. If the item connected to the **node** input exists, its content will be copied to the target. Otherwise, the content of the item connected to the **replace-with** input will be copied to the target.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

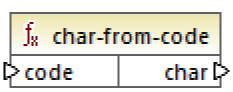
Name	Description
node	This input must receive a connection from a mapping item that provides a sequence[1276] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
replace-with	This input must receive a connection from a mapping item that provides the replacement value.

6.7.10 core | string functions

The string functions allow you to manipulate string data so as to extract parts of strings, test for sub-strings, retrieve information from strings, split strings, and others.

6.7.10.1 char-from-code

Returns the character representation of the decimal Unicode value (code) supplied as argument. **Tip:** To find the Unicode decimal code of a character, you can use the `code-from-char`[1057] function.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

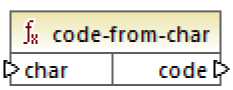
Name	Description
code	The Unicode value, as a decimal number.

Example

According to the charts available on the Unicode website (https://www.unicode.org/charts/), the exclamation mark character has the hexadecimal value of `0021`. The corresponding value in decimal format is **33**. Therefore, supplying `33` as argument to the `char-from-code` function will return the `!` character. See also Replacing Special Characters 276.

6.7.10.2 code-from-char

Returns the decimal Unicode value (code) of the character supplied as argument. If the string supplied as argument has multiple characters, then the code of the first character is returned.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

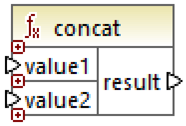
Name	Description
char	The input string value.

Example

If the input **char** is the `$` (dollar sign) character, the function returns **36** (which is the decimal Unicode value for this character).

6.7.10.3 concat

Concatenates (appends) two or more values into a single result string. All input values are automatically converted to type "string". By default, this function has only two parameters, but you can add more. Click **Add parameter** ( ) or **Delete parameter** ( ) to add or remove parameters.



Note: All the inputs to the `concat` function must have a value. If any of the inputs does not have a value, the function is not called and an error occurs. Be aware that an empty string is a valid input value; however, an empty sequence (such as the result of the **`set-empty`** function) is not a valid value and the function will fail as a result. To prevent this from happening, you can first process values with the substitute-missing 1056 function and then supply the result as input to the **`concat`** function.

Languages

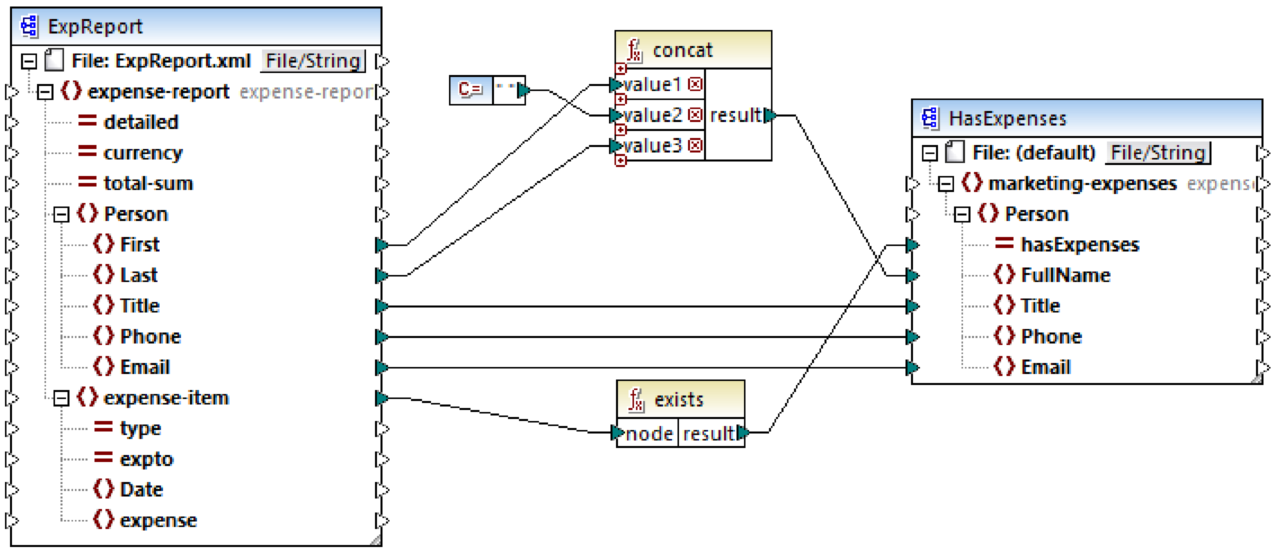
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
value1	The first input value.
value2	The second input value.
valueN	The *N* input value.

Example

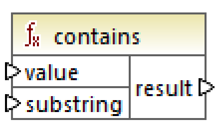
In the mapping illustrated below, the **`concat`** function joins the first name, the constant " ", and the last name. The returning value is then written to the **FullName** target item. The mapping of this function is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\HasMarketingExpenses.mfd**.



HasMarketingExpenses.mfd

6.7.10.4 contains

Returns Boolean **true** if the string value supplied as argument contains the sub-string supplied as argument.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

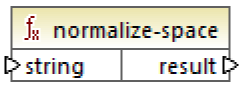
Name	Description
value	The input value (that is, the "haystack").
substring	The sub-string to look for (that is, the "needle").

Example

If the input **value** is "category" and **substring** is "cat", the function returns **true**.

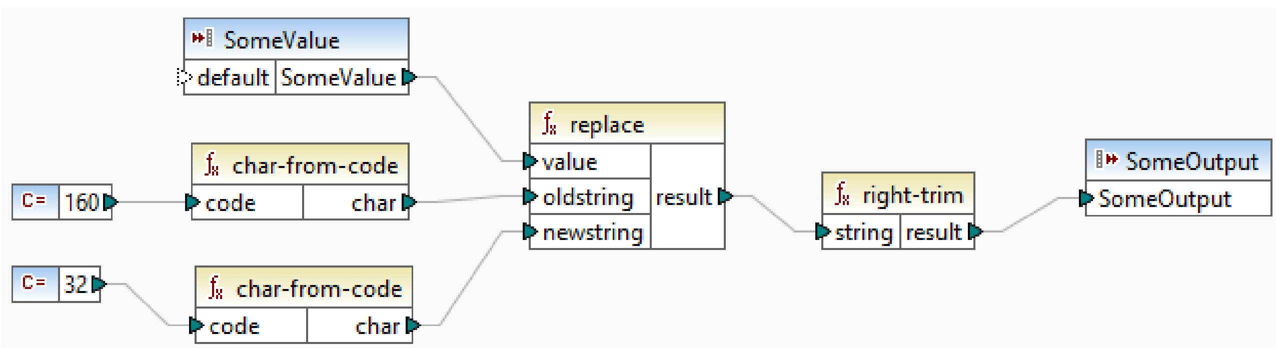
6.7.10.5 normalize-space

The **`normalize-space`** function (*see screenshot below*) removes leading and trailing spaces of a string and replaces internal whitespaces with a single whitespace character. Whitespace includes space (U+0020), tab (U+0009), carriage return (U+000D), and line feed (U+000A) characters. For details about whitespaces, see the [XML Recommendation](#).



About non-breaking spaces

The **`left-trim`**, **`right-trim`**, and **`normalize-space`** functions do not remove non-breaking spaces. One of the possible solutions could be to replace the non-breaking space character, whose decimal representation is 160, with the space character, whose decimal representation is 32. The mapping below shows that after the non-breaking space has been replaced, the trimmed `SomeValue` value will be mapped to the target.



If your source component is an Excel file, you can remove extra spaces in Excel using a combination of TRIM, CLEAN, and SUBSTITUTE functions. For details, see *[Removing Spaces and Nonprinting Characters from Text](#)*.

Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

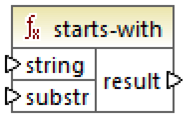
Name	Description
string	The input string to normalize.

Example

If the input string is "`  The quick  brown fox  `", the function returns "`The quick brown fox`".

6.7.10.6 starts-with

Returns Boolean **true** if the string supplied as argument starts with the sub-string supplied as argument; **false** otherwise.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

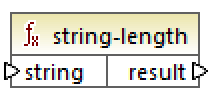
Name	Description
string	The input string.
substr	The sub-string to check for.

Example

If the input **string** is `category` and **substr** is `cat`, the function returns **true**.

6.7.10.7 string-length

Returns the number of characters in the string supplied as argument.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

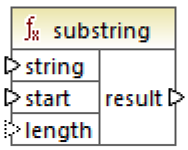
Name	Description
string	The input string.

Example

If the input string is **`car`**, the function returns **3**. If the input string is an empty string, the function returns **0**.

6.7.10.8 substring

Returns the portion of the string specified by the **start** and **length** parameters.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

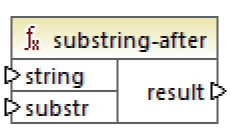
Name	Description
string	The input string.
start	Specifies the starting position (index) from which the sub-string should be retrieved. The first index is **1**.
length	Optional. Specifies the number of characters to retrieve. If the **length** parameter is not specified, the result is a fragment starting from **start** until the end of the string.

Example

If the input string is **`MapForce`**, start is **1**, and length is **3**, the function returns **`Map`**. If the input string is **`MapForce`**, start is **4**, and length is not provided, the function returns **`Force`**.

6.7.10.9 substring-after

Returns the portion of the string that occurs after the first occurrence of **substr**. If **substr** does not occur in **string**, the function returns an empty string.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

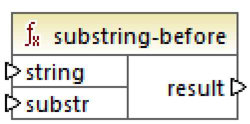
Name	Description
string	The input string.
substr	The sub-string. Any characters after the first occurrence of **substr** are the result of the function.

Example

If the input string is `MapForce`, and **substr** is `Map`, the function returns `Force`. If the input string is `2020/01/04` and **substr** is `/`, the function returns `01/04`. See also Example: Mapping Data from an RSS Feed[619].

6.7.10.10 substring-before

Returns the portion of the string that occurs before the first occurrence of **substr**. If **substr** does not occur in **string**, the function returns an empty string.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

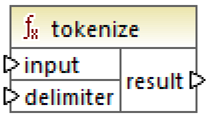
Name	Description
string	The input string.
substr	The sub-string. Any characters before the first occurrence of **substr** are the result of the function.

Example

If the input string is `MapForce`, and **substr** is `Force`, the function returns `Map`. If the input string is `2020/01/04` and **substr** is `/`, the function returns `2020`.

6.7.10.11 tokenize

Splits the input string into a sequence of strings using the delimiter supplied as argument.



Languages

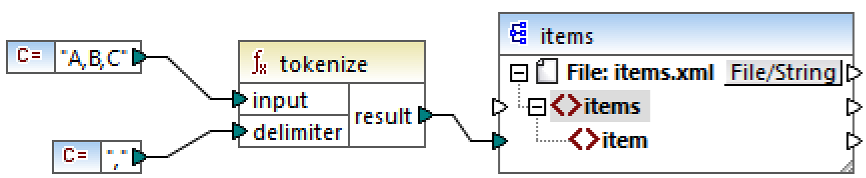
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
input	The input string.
delimiter	The delimiter to use.

Example

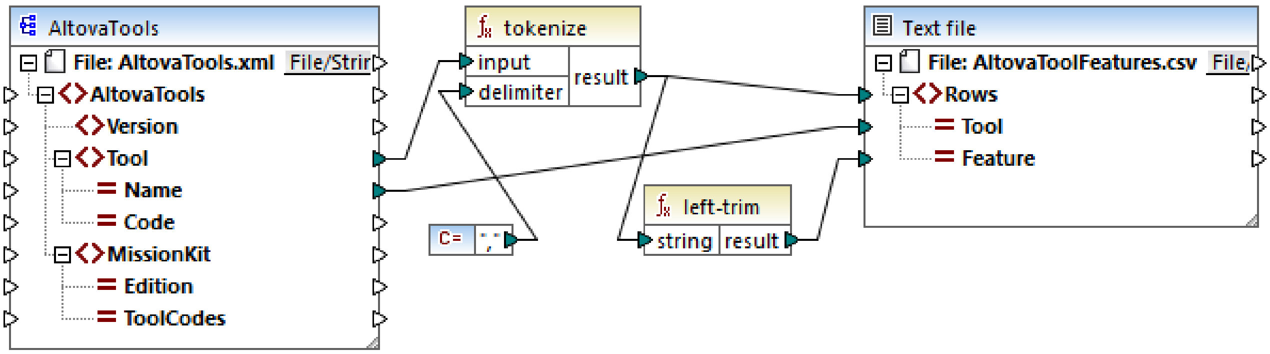
If the input string is `A,B,C` and the delimiter is `,` then the function returns a sequence of three strings: `A`, `B`, and `C`.



In the mock-up mapping illustrated above, the function's result is a sequence of strings. According to the general mapping rules 1276, for each item in the source sequence, a new **item** is created in the target component. Consequently, the mapping output looks as follows:

```
<items>
   <item>A</item>
   <item>B</item>
   <item>C</item>
</items>
```

For a more elaborate example, see the **tokenizeString1.mfd** mapping available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder.



tokenizeString1.mfd

A fragment from the source XML file is shown below. The **Tool** element has two attributes: **Name** and **Code**. The **Tool** element data consists of comma-delimited text.

```
<?xml version="1.0" encoding="UTF-8"?>
<AltovaTools xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AltovaTools.xsd">
   <Version>2010</Version>
   <Tool Name="XMLSpy" Code="XS">XML editor, XSLT editor, XSLT debugger, XQuery editor,
XQuery debugger, XML Schema / DTD editor, WSDL editor, SOAP debugger</Tool>
   <Tool Name="MapForce" Code="MF">Data integration, XML mapping, database mapping, text
conversion, EDI translator, Excel mapping, XBRL mapping, Web services</Tool>
   <Tool Name="StyleVision" Code="SV">Stylesheet designer, electronic forms, XSLT design,
XSL:FO design, database reporting, XBRL rendering</Tool>
   <Tool Name="UModel" Code="UM">UML modeling tool, code generation, reverse engineering,
UML, BPMN, SysML, project documentation, XMI interchange</Tool>
   <Tool Name="DatabaseSpy" Code="DS">Multi-database tool, SQL auto-completion, graphical
database design, table browser, content editor, database comparison tool</Tool>
   <!-- ... -->
</AltovaTools>
```

The mapping does the following:

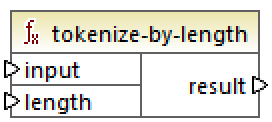
- The **tokenize** function receives data from the **Tool** source item and uses the comma `,` delimiter to split that data into separate chunks. The first chunk is "XML editor", the second one is "XSLT editor", and so on.
- For each chuck resulting from the **tokenize** function, a new row is generated in the target. This happens thanks to the connection between the function's result and the **Rows** item in the target component.
- The result of the **tokenize** function is also mapped to the **left-trim** function, which removes the leading white space of each chunk.
- The result of the **left-trim** function (each chunk) is written to the **Feature** item of the target component.
- The target component output file has been defined as a CSV file (**AltovaToolFeatures.csv**) with the field delimiter being a semicolon (double click component to see settings).

The result of the mapping is that, for each chunk created by the **tokenize** function, a new row is created in the target CSV file. A fragment of the mapping output looks as follows:

```
Tool;Feature
XMLSpy;XML editor
XMLSpy;XSLT editor
XMLSpy;XSLT debugger
XMLSpy;XQuery editor
XMLSpy;XQuery debugger
XMLSpy;XML Schema / DTD editor
XMLSpy;WSDL editor
XMLSpy;SOAP debugger
MapForce;Data integration
MapForce;XML mapping
MapForce;database mapping
MapForce;text conversion
MapForce;EDI translator
MapForce;Excel mapping
```

6.7.10.12 tokenize-by-length

Splits the input string into a sequence of strings. The size of each resulting string is determined by the **length** parameter.



Languages

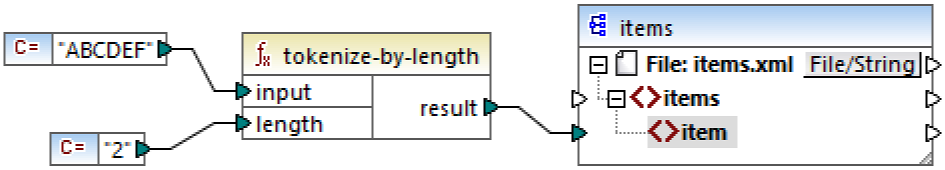
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
input	The input string.
length	Determines the length of each string in the generated sequence of strings.

Example

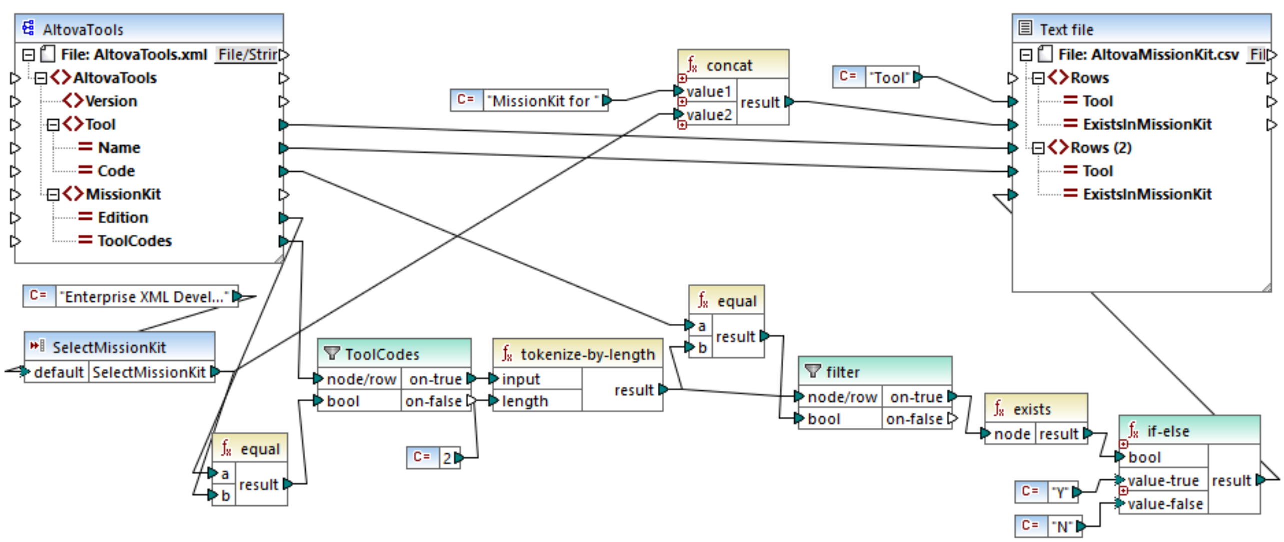
If the input string is **ABCDEF** and the length is **2**, then the function returns a sequence of three strings: **AB**, **CD**, and **EF**.



In the mock-up mapping illustrated above, the function's result is a sequence of strings. According to the general mapping rules 1276, for each item in the source sequence, a new **item** is created in the target component. Consequently, the mapping output looks as follows:

```
<items>
   <item>AB</item>
   <item>CD</item>
   <item>EF</item>
</items>
```

For a more elaborate example, see the **tokenizeString2.mfd** mapping available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder.



tokenizeString2.mfd

The XML source file is shown below, and is the same as the one used in the previous example. The **MissionKit** element has two attributes: **Edition** and **ToolCodes**, but no MissionKit element content. Note that some of the XML content not relevant to this example has been removed.

```
<?xml version="1.0" encoding="UTF-8"?>
<AltovaTools xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AltovaTools.xsd">
   <Version>2010</Version>
   <Tool Name="XMLSpy" Code="XS"><!--...--></Tool>
   <Tool Name="MapForce" Code="MF"><!--...--></Tool>
```

```
  <Tool Name="StyleVision" Code="SV"><!--...--></Tool>
  <Tool Name="UModel" Code="UM"><!--...--></Tool>
  <Tool Name="DatabaseSpy" Code="DS"><!--...--></Tool>
  <Tool Name="DiffDog" Code="DD"><!--...--></Tool>
  <Tool Name="SchemaAgent" Code="SA"><!--...--></Tool>
  <Tool Name="SemanticWorks" Code="SW"><!--...--></Tool>
  <Tool Name="Authentic" Code="AU"><!--...--></Tool>
  <MissionKit Edition="Enterprise Software Architects" ToolCodes="XSMFSVUMDSDDSASW"/>
  <MissionKit Edition="Professional Software Architects" ToolCodes="XSMFSVUMDS"/>
  <MissionKit Edition="Enterprise XML Developers" ToolCodes="XSMFSVDDSASW"/>
  <MissionKit Edition="Professional XML Developers" ToolCodes="XSMFSV"/>
</AltovaTools>
```

The aim of the mapping is to generate a list showing which Altova tools are part of the respective MissionKit editions.

How the mapping works:

- The **SelectMissionKit** input component acts as a parameter to the mapping; it receives its default value from a constant, in this case "Enterprise XML Developers".
- The **equal** function compares the edition supplied as parameter with the **Edition** item from the source XML file and passes on the result to the **bool** parameter of the **ToolCodes** filter.
- The **node/row** input of the **ToolCodes** filter is supplied by the **ToolCodes** item of the source file. The value for the "Enterprise XML Developers" edition is: **XSMFSVDDSASW**.
- The **XSMFSVDDSASW** value is passed to the **on-true** parameter, and further to the **input** parameter of the **tokenize-by-length** function.
- The **tokenize-by-length** function splits the value **XSMFSVDDSASW** into multiple chunks of two characters each. The **length** parameter is **2**; therefore 6 chunks are created as a result.
- Each chunk is compared to the 2-character **Code** value from the source file (of which there are 9 items in total). The result of the comparison (true/false) is passed on to the **bool** parameter of the filter. Note that *all* chunks produced by the **tokenize-by-length** function are passed on to the **node/row** parameter of the filter.
- The **exists** functions now checks for existing/non-existing nodes passed on to it by the **on-true** parameter of the filter component. Existing nodes are those where *there is a match* between the **ToolCodes** chunk and the **Code** value. Non-existing nodes are those where there was no **ToolCodes** chunk to match a **Code** value.
- Each **bool** result of the **exists** function is passed on to the **if-else** component, which generates a "Y" in the target if the node exists, or an "N" if the node does not exist.

The result of the mapping is as follows:

```
Tool;MissionKit for Enterprise XML Developers
XMLSpy;Y
MapForce;Y
StyleVision;Y
UModel;N
DatabaseSpy;N
DiffDog;Y
SchemaAgent;Y
```

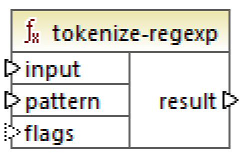


```
SemanticWorks;Y
Authentic;N
```

6.7.10.13 tokenize-regexp

Splits the input string into a sequence of strings. Any substring that matches the regular expression **pattern** supplied as argument defines the separator. The matched (separator) strings are not included in the result returned by the function.

Note: When generating C++, C#, or Java code, the advanced features of the regular expression syntax might differ slightly. See the regex documentation of each language for more information.



Languages

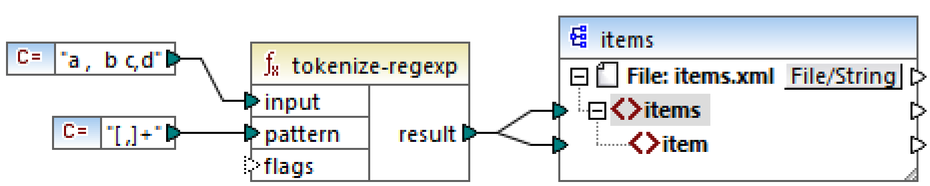
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
input	The input string.
pattern	Provides a regular expression pattern. Any substring that matches the pattern will be treated as delimiter. For more information, see Regular expressions [975].
flags	Optional parameter. Provides the regular expression flags [977] to be used. For example, the flag "i" instructs the mapping process to operate in case-insensitive mode.

Example

The goal of the mapping illustrated below is to split the string `a ,  b c,d` into a sequence of strings, where each alphabetic character is an item in the sequence. Any redundant whitespace or commas must be removed.



To achieve this goal, the regular expression pattern `[ ,]+` was supplied as parameter to the `tokenize-regexp` function. This pattern has the following meaning:

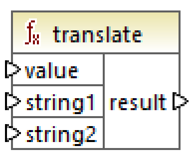
- It matches any of the characters inside the character class `[ ,]`. Therefore, a split will occur whenever a comma or a space is encountered in the input string.
- The quantifier `+` specifies that one or more occurrences of the preceding character class are to be matched. Without this quantifier, each occurrence of space or comma would create a separate item in the resulting sequence of strings, which is not the intended result.

The mapping output is as follows:

```
<items>
   <item>a</item>
   <item>b</item>
   <item>c</item>
   <item>d</item>
</items>
```

6.7.10.14 translate

Performs a character by character replacement. It looks in the **value** for characters contained in **string1**, and replaces each character with the one in the same position in the **string2**. When there are no corresponding characters in **string2**, the character is removed.



Languages

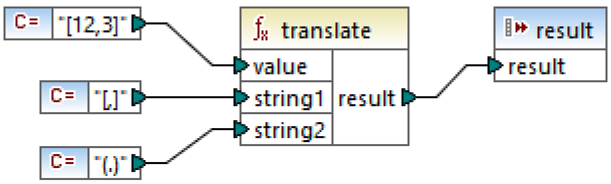
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
value	The input string.
string1	Provides a list of search characters. The position of each character inside the string is important.
string2	Provides a list of replacement characters. The position of each replacement character must correspond to the one in **string1**.

Example

Let's suppose you want to convert the string `[12,3]` to `(12.3)`. Namely, the square brackets must be replaced by round brackets, and any comma must be replaced by the dot character. To achieve this, you can call the `translate` function as follows:



In the mapping above, the first constant supplies the input string to be processed. The second and the third constant provide a list of characters as **string1** and **string2**, respectively.

string1	`[,]`
string2	`(.)`

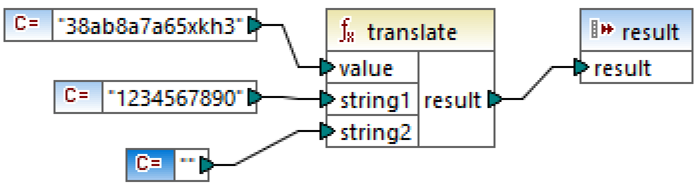
Notice that both **string1** and **string2** have the same number of characters. For each character in **string1**, the equivalent character at the same position from **string2** will be used as a replacement. Consequently, the following replacements will take place:

- Each `[` will be replaced by a `(`
- Each `,` will be replaced by a `.`
- Each `]` will be replaced by a `)`

The mapping output is as follows:

```
(12.3)
```

This function can also be used to strip certain characters selectively from a string. To achieve this, set the **string1** parameter to the characters you want to remove, and **string2** to an empty string. For example, the mapping below removes all digits from the string `38ab8a7a65xkh3`.



The mapping output is as follows:

```
abaaxkh
```

6.7.11 bson

This section provides information about `bson` functions. To find out more about these functions, see the respective topic below:

- bson | binary functions 1072
- bson | constructors 1073
- bson | regex functions 1074

For more information about BSON types, see the BSON Specification.

Important: The `bson` functions are compatible only with the Built-In 27 transformation language.

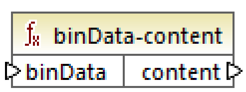
6.7.11.1 bson | binary functions

This subsection describes binary functions. BSON binary values are two-part values: They have a subtype, which is defined in the BSON Specification, and binary content. The functions described below enable you to extract the content from a BSON binary value (**`binData-content`**), find out about the subtype of this binary value (**`binData-subtype`**), and create a binary value based on the `subtype` and `content` parameters (**`to-binData`**).

Important: The `bson` functions are compatible only with the Built-In 27 transformation language.

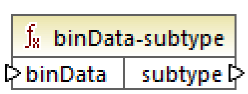
⊟ binData-content

The **`binData-content`** function (*see screenshot below*) enables you to extract content from a BSON binary value.



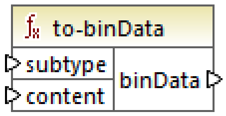
⊟ binData-subtype

BSON binary values have a subtype that defines a type of data stored in the byte array. The list of BSON subtypes is given in the BSON Specification. The **`binData-subtype`** function (*see screenshot below*) extracts the subtype from a BSON binary value.



⊟ to-binData

The **`to-binData`** function (*see screenshot below*) enables you to create a BSON binary value by supplying the `subtype` and `content` parameters.



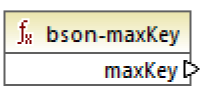
6.7.11.2 bson | constructors

This subsection describes BSON constructor functions that are concerned with `maxKey`, `minKey`, `null` and `undefined` values, all of which are defined in the BSON Specification.

Important: The `bson` functions are compatible only with the Built-In 27 transformation language.

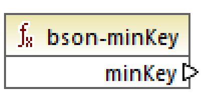
bson-maxKey

The **`bson-maxKey`** function (*see screenshot below*) enables you to construct a `maxKey` value on the selected node of the database component.



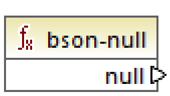
bson-minKey

The **`bson-minKey`** function (*see screenshot below*) enables you to construct a `minKey` value on the selected node of the database component.



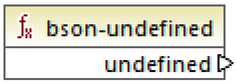
bson-null

The **`bson-null`** function (*see screenshot below*) constructs a BSON null value on the selected node of your database component.



bson-undifined

The **`bson-undefined`** function (*see screenshot below*) enables you to construct an `undefined` value on the selected node of the database component.



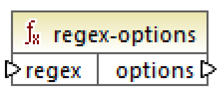
6.7.11.3 bson | regex functions

This subsection describes regex functions that enable you to perform pattern matching. According to the BSON Specification, regular expressions are two-part values: They have a pattern and options. The functions listed below enable you separate these two parts and extract either of them (the **regex-options** and **regex-pattern** functions). You can also create a regex expression based on the `pattern` and `options` parameters (**to-regex**).

Important: The `bson` functions are compatible only with the Built-In [27] transformation language.

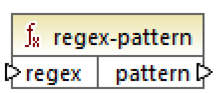
⊟ regex-options

The **regex-options** function (*see screenshot below*) extracts the options from a BSON regex value.



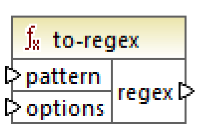
⊟ regex-pattern

The **regex-pattern** function (*see screenshot below*) extracts the pattern from a BSON regex value.



⊟ to-regex

The **to-regex** function (*see screenshot below*) enables you to create a BSON regex value based on the `pattern` and `options` parameters.

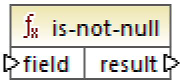


6.7.12 db

The **db** library contains functions that allow you to define the mapping results when encountering null fields in databases.

6.7.12.1 is-not-null

Returns **false** if the field is null; **true** otherwise.



Languages

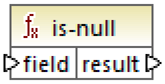
Built-in, C++, C#, Java.

Parameters

Name	Description
field	The database field.

6.7.12.2 is-null

Returns **true** if the field is null; **false** otherwise.



Languages

Built-in, C++, C#, Java.

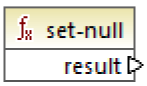
Parameters

Name	Description
field	The database field.

6.7.12.3 set-null

Sets a database field to null. This function will also overwrite a default value with null. If connected to something that is not a database field, it will behave like an empty sequence. Note the following:

- Connecting **`set-null`** to a different function will usually result in the other function not being called at all. Connecting **`set-null`** to a sequence function such as **`count`** will call the function with an empty sequence.
- Connecting **`set-null`** to filters and if-else conditions works as expected; fields are set to null. For filters, this means the "node/row" input.
- Using **`set-null`** as an input for a `simpleType` element will not create that element in the target component.

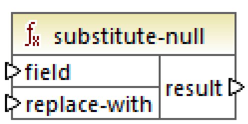


Languages

Built-in, C++, C#, Java.

6.7.12.4 substitute-null

Returns the field itself if it is not null; otherwise, **replace-with** is returned.



Languages

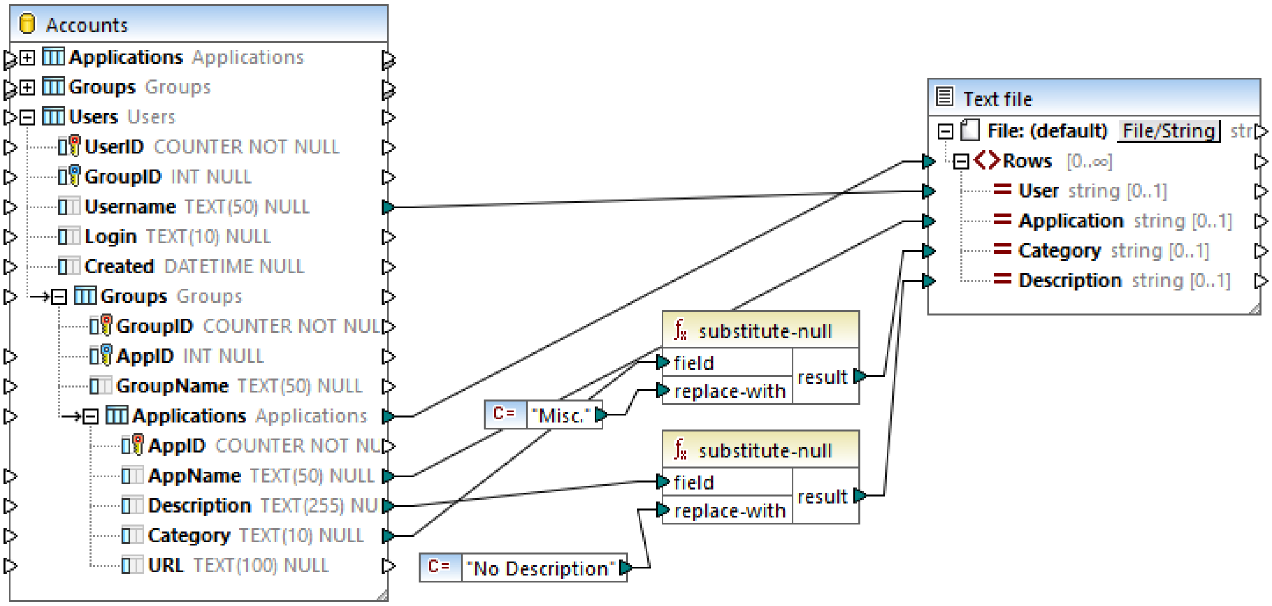
Built-in, C++, C#, Java.

Parameters

Name	Description
field	The database field.
replace-with	The replacement value.

Example

The mapping below shows an example of the **`substitute-null`** function in use. This mapping is called **DB-ApplicationList.mfd** and is available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder.



The mapping reads data from an Access database which contains an "Applications" table.

	AppID	AppName	Description	Category	URL
1	1	Altova MapForce	The premier data mapping tool.	IDE	www.altova.com/xmlspy
2	2	Notepad	*[NULL]*	*[NULL]*	*[NULL]*

The first function checks if the **Category** field is null in the "Applications" table. Since this field is null for the Notepad application, the substitute value "Misc" is mapped to the **Category** item of the target text file.

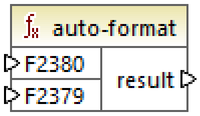
The second function checks if the **Description** field is null. Again, this field is null for the Notepad application, so the substitute value "No description" is mapped to the **Description** item of the target file.

6.7.13 edifact

Functions in the **edifact** library are applicable to mappings that involve UN/EDIFACT components, see UN/EDIFACT[508]. The functions can be used to convert EDI date, time, and period values into `xs:date` formats.

6.7.13.1 auto-format

Returns the `date`, `time`, or `datetime` value extracted from Data Element **F2380** ("Date or time or period value"), formatted using Data Element **F2379** ("Date or time or period format code").



Languages

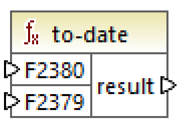
Built-in, C++, C#, Java.

Parameters

Name	Description
F2380	This input must be connected to an **F2380** item of a UN/EDIFACT component.
F2379	This input must be connected to an **F2379** item of a UN/EDIFACT component.

6.7.13.2 to-date

Returns the `date` value extracted from Data Element **F2380** ("Date or time or period value"), formatted using Data Element **F2379** ("Date or time or period format code").



Languages

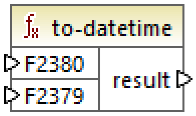
Built-in, C++, C#, Java.

Parameters

Name	Description
F2380	This input must be connected to an **F2380** item of a UN/EDIFACT component.
F2379	This input must be connected to an **F2379** item of a UN/EDIFACT component.

6.7.13.3 to-datetime

Returns the `datetime` value extracted from Data Element **F2380** ("Date or time or period value"), formatted using Data Element **F2379** ("Date or time or period format code").



Languages

Built-in, C++, C#, Java.

Parameters

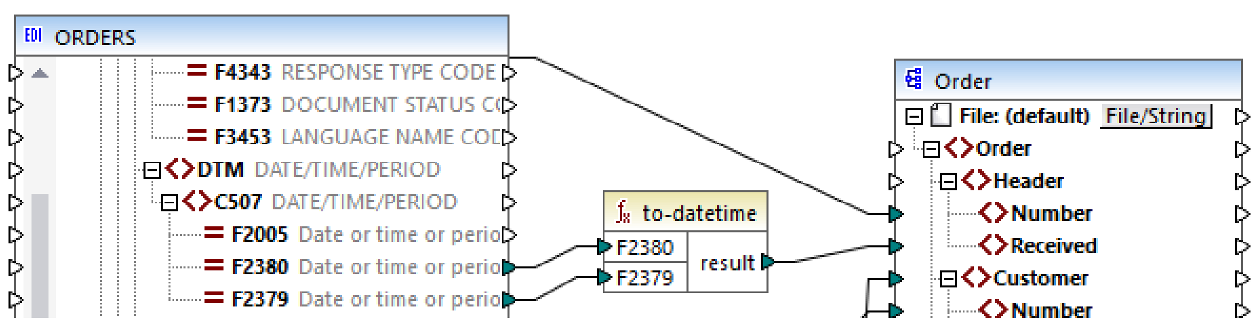
Name	Description
F2380	This input must be connected to an **F2380** item of a UN/EDIFACT component.
F2379	This input must be connected to an **F2379** item of a UN/EDIFACT component.

Example

The code listing below shows a fragment of a UN/EDIFACT file where Data Elements **F2380** and **F2379** appear in the line that begins with "DTM". Specifically, the value of **F2380** is `200404301742PDT` and the value of **F2379** is `303`.

```
UNB+UNOB:1+003897733:01:MFGB+PARTNER ID:ZZ:ROUTING ADDR+970101:1230+00000000000001+
+ORDERS++++1'
UNH+0001+ORDERS:D:19B:UN'
BGM+221+ABC123456XYZ+9'
DTM+4:200404301742PDT:303'
FTX+PUR+3++Pizza purchase order'
RFF+CT:123-456'
RFF+CR:1122'
NAD+SE+999::92++24h Pizza+Long Way+San-Francisco+CA+34424+US'
CTA+SR+:Ted Little'
```

The mapping illustrated below extracts the value of **F2380** and formats it using the code from **F2379**, with the help of `to-datetime` function. This mapping is called **EDI-Order.mfd** and is available in the **<Documents>\Altova\MapForce2023\MapForceExamples** folder.

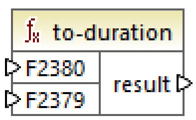


The relevant part of the mapping output is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order>
   <Header>
      <Number>ABC123456XYZ</Number>
      <Received>2004-04-30T17:42:00-09:00</Received>
   </Header>
   <!-- ... -->
</Order>
```

6.7.13.4 to-duration

Returns the `duration` value extracted from Data Element **F2380** ("Date or time or period value"), formatted using Data Element **F2379** ("Date or time or period format code").



Languages

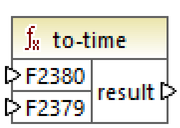
Built-in, C++, C#, Java.

Parameters

Name	Description
F2380	This input must be connected to an **F2380** item of a UN/EDIFACT component.
F2379	This input must be connected to an **F2379** item of a UN/EDIFACT component.

6.7.13.5 to-time

Returns the `time` value extracted from Data Element **F2380** ("Date or time or period value"), formatted using Data Element **F2379** ("Date or time or period format code").



Languages

Built-in, C++, C#, Java.

Parameters

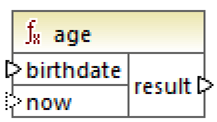
Name	Description
F2380	This input must be connected to an **F2380** item of a UN/EDIFACT component.
F2379	This input must be connected to an **F2379** item of a UN/EDIFACT component.

6.7.14 lang | datetime functions

The date and time functions from the **lang** library can be used to manipulate dates, times, and durations. Unlike the date and time functions from the **core** library, these functions are available only when selecting Built-in, Java, C#, or C++ languages.

6.7.14.1 age

Returns the number of full years elapsed between the birth date supplied as argument and now.



Languages

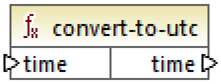
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
birthdate	`xs:date`	Mandatory. Provides the birth date as an `xs:date` value.
now	`xs:date`	Optional parameter. The default is the current system date. If a value is mapped to the now argument, the function returns the difference between the birth date and now, in full years.

6.7.14.2 convert-to-utc

Converts the time value supplied as argument to UTC (Coordinated Universal Time). The function takes the timezone component (for example, "+5:00") into account.



Languages

Built-in, C++, C#, Java.

Parameters

Name	Type	Description
time	`xs:dateTime`	Provides the `xs:dateTime` value to be converted.

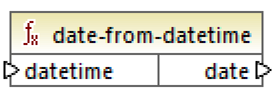
Example

If the input value is **`2001-12-17T09:30:02+05:00`**, the function's result is **`2001-12-17T04:30:02`**.

If the input value is **`2001-12-17T09:30:02Z`**, the function's result is **`2001-12-17T09:30:02`**. In this case, no conversion has taken place, because the trailing "Z" already defines this time to be "Zero" (or "Zulu") time, which is the same as UTC.

6.7.14.3 date-from-datetime

Returns the *date* part from the `xs:dateTime` value supplied as argument. The *time* part is set to zero. The timezone is not changed.



Languages

Built-in, C++, C#, Java.

Parameters

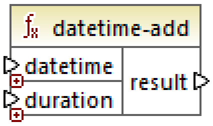
Name	Type	Description
datetime	`xs:dateTime`	Provides the `xs:dateTime` value to be processed.

Example

If the input value is **`2001-12-17T09:30:02+05:00`**, the function's result is **`2001-12-17+05:00`**.

6.7.14.4 datetime-add

Returns an `xs:dateTime` value obtained by adding a duration (the second argument) to a datetime (the first argument).



Languages

Built-in, C++, C#, Java.

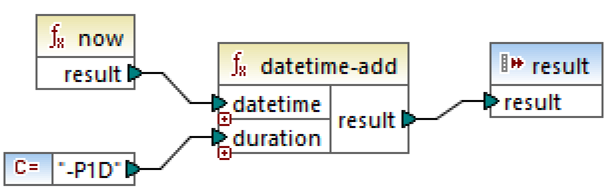
Parameters

Name	Type	Description
datetime	**xs:dateTime**	Provides the `xs:dateTime` value to be used as input.
duration	**xs:duration**	Provides the `xs:duration` value. An example duration is **`P1Y2M3DT04H05M59S`**, where: • "P" is the period designator, and is mandatory; • The rest of the characters denote, in this order: 1 Year, 2 Months, 3 Days, T (Time designator), 04 Hours, 05 Minutes, 59 Seconds. If the minus character appears before the "P" designator, this indicates a negative duration, for example: **`-P1D`**.

Example

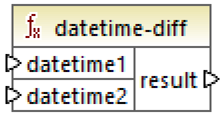
Let's assume that the input **datetime** value is **`2001-12-17T09:30:02+05:00`**. If the **duration** is **`P10D`** (10 days), the function's result is **`2001-12-27T09:30:02+05:00`**.

To obtain yesterday's date, connect the **now** function to the **datetime** input. In the mapping below, the period **`-P1D`** means "minus 1 day", so the mapping returns yesterday's date.



6.7.14.5 datetime-diff

Returns the duration obtained by subtracting **datetime2** (second argument) from **datetime1** (first argument). The result can be mapped to a string or duration data type.



Languages

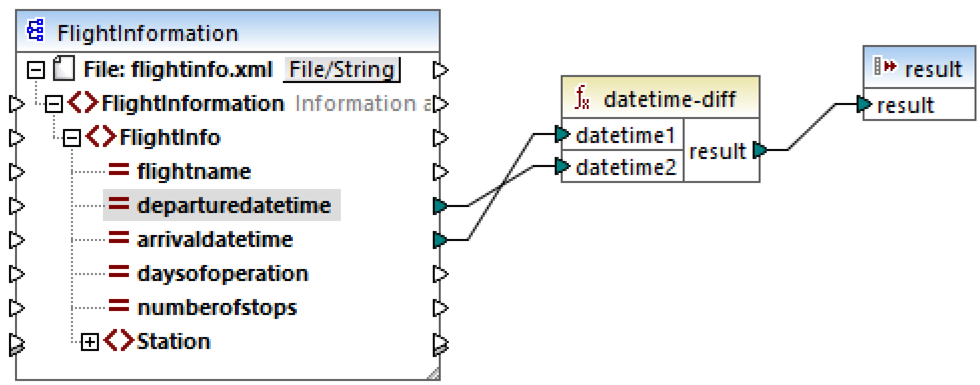
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
datetime1	`xs:dateTime`	Provides the first `xs:dateTime` value.
datetime2	`xs:duration`	Provides the second `xs:dateTime` value.

Example

In the mapping illustrated below, the **`datetime-diff`** function subtracts the flight departure datetime **`2001-12-17T09:30:02+05:00`** from the arrival datetime **`2001-12-17T19:30:02+05:00`**. Note that the arrival datetime is the greater value, so it is connected to the first input of the function.

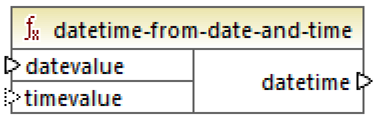


The mapping output is the difference between the two (a period of 10 hours):

```
PT10H
```


6.7.14.6 datetime-from-date-and-time

Returns an `xs:dateTime` value built from an `xs:date` value (first argument) and an `xs:time` value (second argument). The result can be mapped to a string or `xs:dateTime` data type.



Languages

Built-in, C++, C#, Java.

Parameters

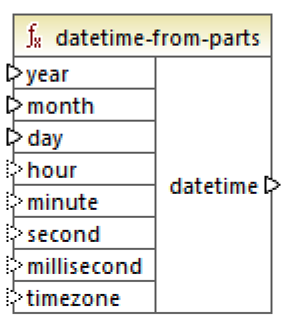
Name	Type	Description
datevalue	**`xs:date`**	Provides a value of type `xs:date`
timevalue	**`xs:time`**	Provides a value of type `xs:time`

Example

If the first argument is **`2012-06-29`** and the second argument is **`11:59:55`**, the function returns **`2012-06-29T11:59:55`**.

6.7.14.7 datetime-from-parts

Returns a value of type `xs:dateTime` built from any combination of the following parts as arguments: year, month, day, hour, minute, second, millisecond, and timezone. This function automatically normalizes the supplied parameters. For example, 32nd of January will automatically be changed to 1st of February.



Languages

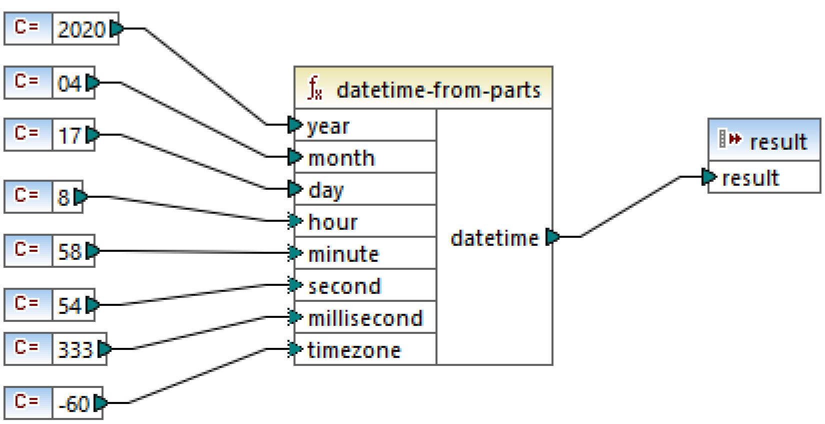
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
year	`xs:int`	Provides the year.
month	`xs:int`	Provides the month.
day	`xs:int`	Provides the day of the month.
hour	`xs:int`	Optional. Provides the hour.
minute	`xs:int`	Optional. Provides the minute.
second	`xs:int`	Optional. Provides the second.
millisecond	`xs:decimal`	Optional. Provides the millisecond.
timezone	`xs:int`	Optional. Provides the timezone, in minutes. This value can be negative.

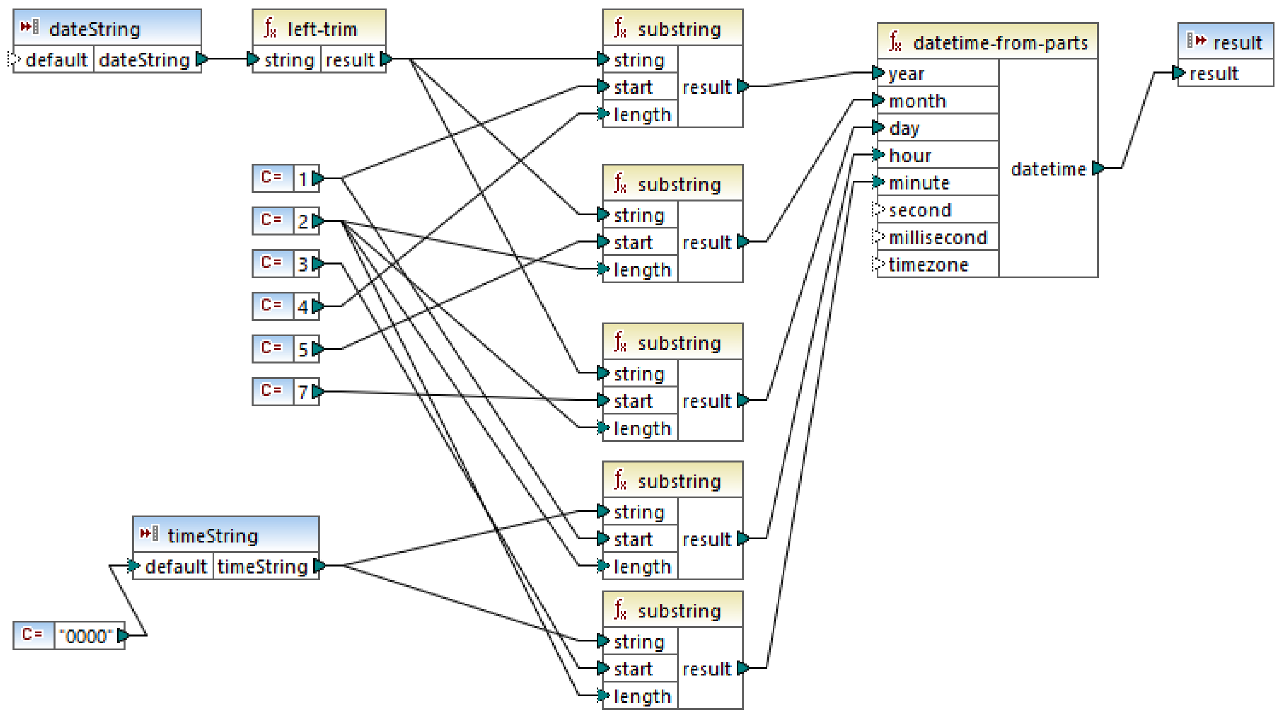
Example

The following mapping constructs an `xs:dateTime` value from parts that are supplied by constants.

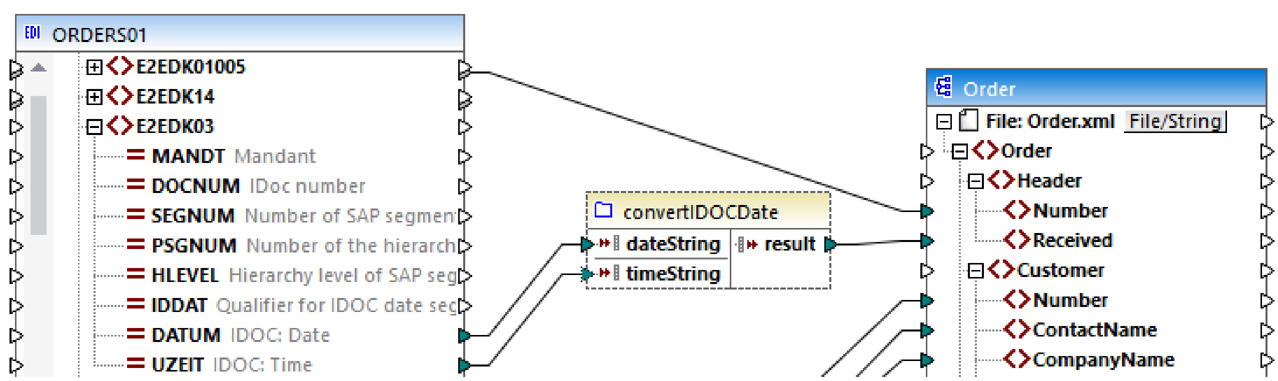


The mapping output is **`2020-04-17T08:58:54.333-01:00`**.

For another example, see the **IDoc_Order.mfd** mapping from the **<Documents>\Altova\MapForce2023\MapForceExamples** folder. This mapping illustrates how to read data from SAP IDoc 483 instance files. It contains a user-defined function "convertIDOCDate" which assembles the `xs:dateTime` value from two input strings, using the **`left-trim`** and **`substring`** functions.



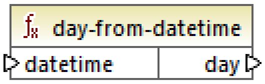
The **dateString** and **timeString** components are parameters to the user-defined function. They are supplied by the main mapping, from the SAP IDoc instance file:



In this mapping, the date value as a string is `19990621`. The time value is `0930`. Consequently, the `datetime-from-parts` function returns the value `1999-06-21T09:30:00`. This value is also the result of the user-defined function and is written to the **Received** item of the target XML document.

6.7.14.8 day-from-datetime

Returns the day, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

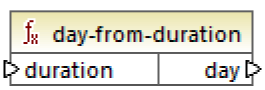
Name	Type	Description
datetime	**xs:dateTime**	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T10:30:03+01:00`**, then the function returns **`17`**.

6.7.14.9 day-from-duration

Returns the day, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

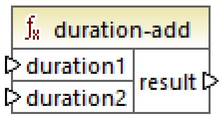
Name	Type	Description
duration	**xs:duration**	Provides the input value of type `xs:duration`.

Example

If **duration** is **`P1Y2M3DT10H30M`**, then the **`day-from-duration`** function returns **`3`**.

6.7.14.10 duration-add

Returns the duration obtained by adding two durations.



Languages

Built-in, C++, C#, Java.

Parameters

Name	Type	Description
duration1	`xs:duration`	Provides the first input value of type `xs:duration`.
duration2	`xs:duration`	Provides the second input value of type `xs:duration`.

Example

If the first duration is **`P0Y0M3DT03H0M`** (3 days and 3 hours) and the second duration is **`P0Y0M3DT01H0M`** (3 days and 1 hour), then the function returns **`P6DT4H`** (6 days and 4 hours).

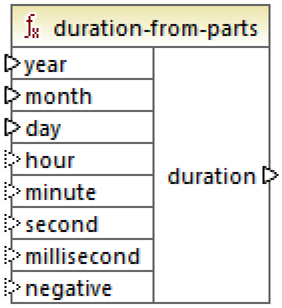
6.7.14.11 duration-from-parts

Returns a value of type `xs:duration` calculated by combining the following parts supplied as arguments: year, month, day, hour, minute, second, millisecond, negative.

An example duration is **`P1Y2M3DT04H05M59S`**, where:

- "P" is the period designator, and is mandatory;
- The rest of the characters denote, in this order: 1 Year, 2 Months, 3 Days, T (Time designator), 04 Hours, 05 Minutes, 59 Seconds.

If the minus character appears before the "P" designator, this indicates a negative duration, for example: **`-P1D`**.



Languages

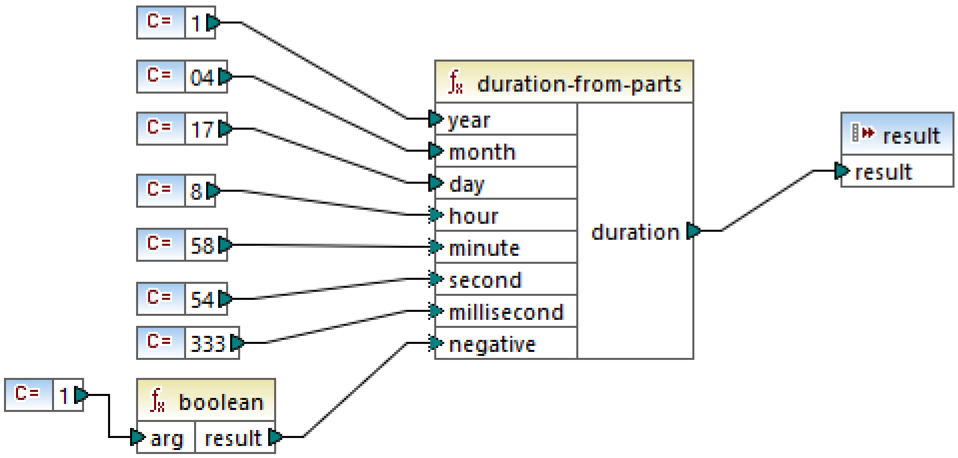
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
year	`xs:int`	Provides the year.
month	`xs:int`	Provides the month.
day	`xs:int`	Provides the day of the month.
hour	`xs:int`	Optional. Provides the hour.
minute	`xs:int`	Optional. Provides the minute.
second	`xs:int`	Optional. Provides the second.
millisecond	`xs:decimal`	Optional. Provides the millisecond.
negative	`xs:boolean`	Optional. Must be **true** for a negative duration; **false** otherwise.

Example

The following mapping generates a negative duration of 1 year, 4 months, 17 days, 8 hours, 58 minutes, and 54.333 seconds.



The mapping output is **`-P1Y4M17DT8H58M54.333S`**.

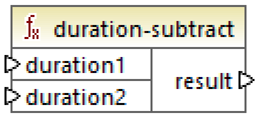
6.7.14.12 duration-subtract

Returns the `xs:duration` value obtained by subtracting **duration2** from **duration1**.

An example duration is **`P1Y2M3DT04H05M59S`**, where:

- "P" is the period designator, and is mandatory;
- The rest of the characters denote, in this order: 1 Year, 2 Months, 3 Days, T (Time designator), 04 Hours, 05 Minutes, 59 Seconds.

If the minus character appears before the "P" designator, this indicates a negative duration, for example: **`-P1D`**.



Languages

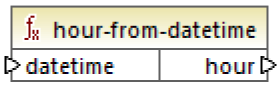
Built-in, C++, C#, Java.

Example

If **duration1** is **`P0Y0M0DT05H07M`** (5 hours and 7 minutes) and **duration2** is **`PT1H`** (1 hour), the function returns **`PT4H7M`** (4 hours and 7 minutes).

6.7.14.13 hour-from-datetime

Returns the hour, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

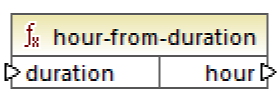
Name	Type	Description
datetime	**`xs:dateTime`**	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T09:30:02+05:00`**, then the function returns **`9`**.

6.7.14.14 hour-from-duration

Returns the hour, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

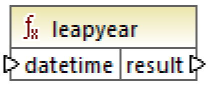
Name	Type	Description
duration	**`xs:duration`**	Provides the input value of type `xs:duration`.

Example

If **duration** is **`P0Y0M0DT05H07M`**, the function returns **`5`**.

6.7.14.15 leapyear

Returns Boolean **true** if the year of the `xs:dateTime` value supplied as argument is a leap year; **false** otherwise.



Languages

Built-in, C++, C#, Java.

Parameters

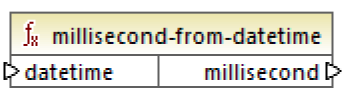
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2020-04-17T09:30:02+02:00`**, the function returns **true**, since the year 2020 is a leap year.

6.7.14.16 millisecond-from-datetime

Returns the milliseconds, as an `xs:decimal` value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

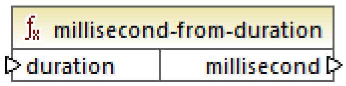
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T09:30:02.544+05:00`**, the function returns **`544`**.

6.7.14.17 millisecond-from-duration

Returns the milliseconds, as an `xs:decimal` value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

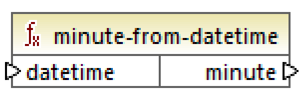
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is `P0Y0M0DT05H07M02.227S`, the function returns `227`.

6.7.14.18 minute-from-datetime

Returns the minutes, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

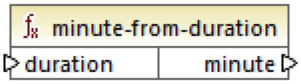
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T09:30:02.544+05:00`, the function returns `30`.

6.7.14.19 minute-from-duration

Returns the minutes, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

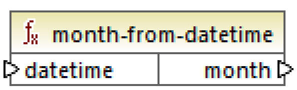
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is `P0Y0M0DT05H07M02.227S`, the function returns `7`.

6.7.14.20 month-from-datetime

Returns the month, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

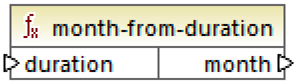
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T09:30:02.544+05:00`, the function returns `12`.

6.7.14.21 month-from-duration

Returns the month, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

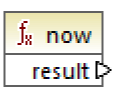
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is **`P0Y04M0DT05H07M02.227S`**, the function returns **`4`**.

6.7.14.22 now

Returns the current date and time (including timezone), as an `xs:dateTime` value.

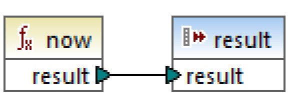


Languages

Built-in, C++, C#, Java.

Example

The following mapping outputs the current date and time. The output changes each time when the mapping runs.

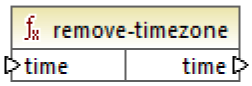


An example output would be **`2020-04-17T11:42:34.684+02:00`**.

For an example on how to extract yesterday's date, see the core | lang | datetime-add[1083] function.

6.7.14.23 remove-timezone

Removes the timezone component from the **time** (of type `xs:dateTime`) input parameter.



Languages

Built-in, C++, C#, Java.

Parameters

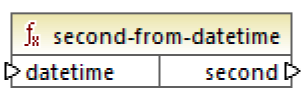
Name	Type	Description
time	**xs:dateTime**	Provides the input value of type `xs:dateTime`.

Example

If **time** is **`2001-12-17T09:30:02+05:00`**, the function returns **`2001-12-17T09:30:02`**.

6.7.14.24 second-from-datetime

Returns the seconds, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

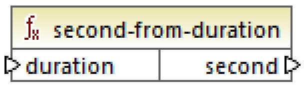
Name	Type	Description
datetime	**xs:dateTime**	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T09:30:02.544+05:00`**, the function returns **`2`**.

6.7.14.25 second-from-duration

Returns the seconds, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

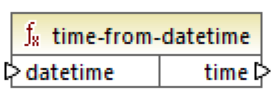
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is **`P0Y04M0DT05H07M02.227S`**, the function returns **`2`**.

6.7.14.26 time-from-datetime

Returns the time component, as an `xs:time` value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

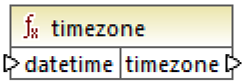
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T09:30:02+05:00`**, the function returns **`09:30:02+05:00`**.

6.7.14.27 timezone

Returns the timezone offset, in minutes, from the `xs:dateTime` value supplied as argument. Returns 0 for UTC.



Languages

Built-in, C++, C#, Java.

Parameters

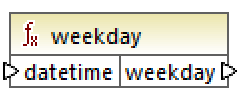
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T09:30:02.544+05:00`**, the function returns **`300`**.

6.7.14.28 weekday

Returns the day of the week from the `xs:dateTime` value supplied as argument. The function will return value **1** for Monday, value **2** for Tuesday, and so on.



Languages

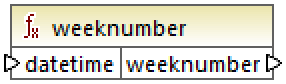
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

6.7.14.29 weeknumber

Returns the week number within the year from the `xs:dateTime` value supplied as argument. The function will return value **1** for the first week of the year, value **2** for the second week, and so on.



Languages

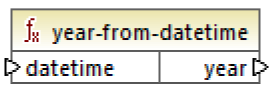
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
datetime	**`xs:dateTime`**	Provides the input value of type `xs:dateTime`.

6.7.14.30 year-from-datetime

Returns the year, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

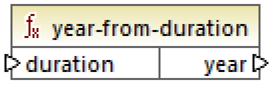
Name	Type	Description
datetime	**`xs:dateTime`**	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T09:30:02.544+05:00`**, the function returns **`2001`**.

6.7.14.31 year-from-duration

Returns the year, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

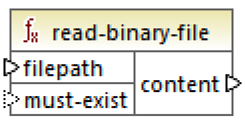
If **duration** is **`P01Y04M0DT05H07M02.227S`**, the function returns **`1`**.

6.7.15 lang | file functions

Use the **file** functions from the **lang** library to read binary content from files, or write binary content to files.

6.7.15.1 read-binary-file

This function returns the content of the specified file as a BLOB (binary large object) of type `xs:base64Binary`. Note that even though the data type is called "base64Binary", the internal representation is just a BLOB. Only when you map the function's result to an XML node of type `xs:base64Binary` will it actually be base64-encoded. You could also map the function's result to `xs:hexBinary`, to a database blob, or to a binary field in a Protocol Buffers structure.



To read a binary file into a mapping, supply its path as input to the **filepath** argument. If the **filepath** is relative, then MapForce will look for the file in the same directory as the mapping. The **must-exist** argument is optional; if the file cannot be opened and this parameter is **true**, the mapping throws an error. If the file cannot be opened and this parameter is **false**, an empty binary is returned.

Languages

Built-in.

Parameters

Name	Type	Description
filepath	`xs:string`	The file path.

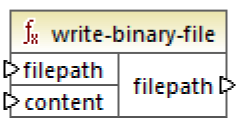
Name	Type	Description
must-exist	`xs:boolean`	Optional parameter. If the file cannot be opened and this parameter is **true**, the mapping throws an error. If the file cannot be opened and this parameter is **false**, an empty binary is returned. The default value is **true**.

Example

See Example: Read Binary Files[828].

6.7.15.2 write-binary-file

This function writes binary content to the specified file path and returns the path of the written file. If a binary file is the only desired output, connect the function's result to a simple output[844] component. Because this function writes a file whenever its output is used in the mapping, it is recommended to connect the function's result directly to a target component, without using other processing in between.



To write binary files, supply their path as input to the **filepath** argument. If **filepath** is relative, then MapForce will generate the file in the same directory as the mapping. The **content** argument must be connected to the actual binary content (for example, a BLOB field in a database).

When you preview the mapping in MapForce, the function generates temporary files by default, instead of writing files directly to the disk. To save the temporary files to disk, first click the **Output** tab, and then click the **Save generated output** or **Save all generated outputs** toolbar button, as applicable.

To configure MapForce to write output directly to final files instead of temporary, select the **Tools | Options** menu command, click **General**, and then select the **Write directly to final output files** option. Be aware that this option overwrites any existing files with the same name.

The function always returns the final (not temporary) file name, even when the final file is not saved to the disk yet (that is the case when you preview the mapping and the **Write directly to final output files** option is disabled).

> Note that it is not supported for a mapping to read back its own output file.

Languages

Built-in.

Parameters

Name	Type	Description
filepath	`xs:string`	The input file path.
content	`xs:base64Binary`	The binary content of type `xs:base64Binary`.

Example

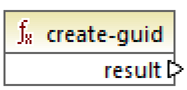
See Example: Write Binary Files [830].

6.7.16 lang | generator functions

The generator functions from the **lang** library are functions that generate values (currently, **`create-guid`** is the only such function).

6.7.16.1 create-guid

Creates a globally unique identifier (GUID), as a hex-encoded string. This function can be used to generate unique values, directly from the mapping, for database fields or other component types. See also the function format-guid-string [1121].



Languages

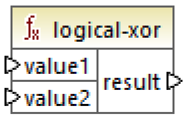
Built-in, C++, C#, Java.

6.7.17 lang | logical functions

Logical functions from the **lang** library include functions that evaluate miscellaneous value types using Boolean logic.

6.7.17.1 logical-xor

Returns **true** if **value1** is different from **value2**; **false** otherwise.



Languages

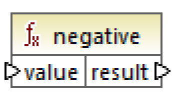
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value1	`xs:boolean`	The first input value.
value2	`xs:boolean`	The second input value.

6.7.17.2 negative

Returns **true** if the input value is negative (less than zero); **false** otherwise.



Languages

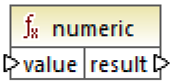
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:decimal`	The input value.

6.7.17.3 numeric

Returns **true** if the input value is a number or a string that can be parsed as a number; **false** otherwise.



Languages

Built-in, C++, C#, Java.

Parameters

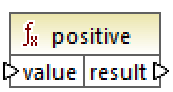
Name	Type	Description
value	xs:decimal	The input value.

Example

If the input value is the string `"4.33"`, the function returns **true**. If the input value is the string `"4.33 USD"`, the function returns **false**.

6.7.17.4 positive

Returns **true** if the input value is positive (equal to or greater than zero); **false** otherwise.



Languages

Built-in, C++, C#, Java.

Parameters

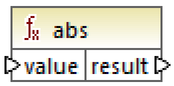
Name	Type	Description
value	xs:decimal	The input value.

6.7.18 lang | math functions

The math functions from the **lang** library can be used to perform various mathematical operations in the mapping.

6.7.18.1 abs

Returns the absolute value of the numeric value supplied as argument. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.



Languages

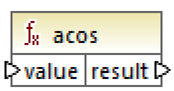
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:decimal`	The input value.

6.7.18.2 acos

Returns the arc cosine of **value**, in the range of `-pi/2` through `pi/2`.



Languages

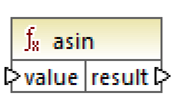
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.18.3 asin

Returns the arc sine of **value**, in the range of `-pi/2` through `pi/2`.



Languages

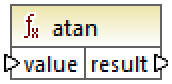
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.18.4 atan

Returns the arc tangent of **value**, in the range of `-pi/2` through `pi/2`.



Languages

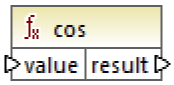
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.18.5 cos

Returns the trigonometric cosine of the angle given by **value**. The unit of value is radian.



Languages

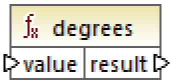
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.18.6 degrees

Converts an angle measured in radians to an approximately equivalent angle measured in degrees.



Languages

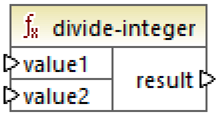
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.18.7 divide-integer

Returns the integer result of dividing **value1** by **value2**.



Languages

Built-in, C++, C#, Java.

Parameters

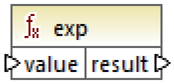
Name	Type	Description
value1	xs:decimal	The first input value.
value2	xs:decimal	The second input value.

Example

If the first value is **15** and the second value is **2**, the function returns **7**.

6.7.18.8 exp

Returns Euler's number **e** raised to the power of **value**.



Languages

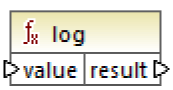
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.18.9 log

Returns the natural logarithm (base e) of **value**.



Languages

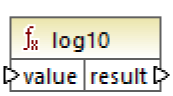
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.18.10 log10

Returns the decimal logarithm (base 10) of **value**.



Languages

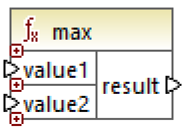
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.18.11 max

Returns the numeric value of the largest value supplied as argument. By default, this function has only two parameters, but you can add more. Click **Add parameter** ( ) or **Delete parameter** ( ) to add or remove parameters, see also Add or Delete Function Arguments 906.



Languages

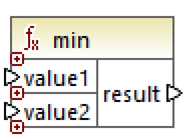
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value1	xs:decimal	The first input value.
value2	xs:decimal	The second input value.
valueN	xs:decimal	The *n*th input value.

6.7.18.12 min

Returns the numeric value of the smallest value supplied as argument. By default, this function has only two parameters, but you can add more. Click **Add parameter** ( ) or **Delete parameter** ( ) to add or remove parameters, see also Add or Delete Function Arguments 906.



Languages

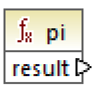
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value1	**xs:decimal**	The first input value.
value2	**xs:decimal**	The second input value.
valueN	**xs:decimal**	The *n*th input value.

6.7.18.13 pi

Returns the value of mathematical constant *pi*.

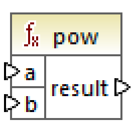


Languages

Built-in, C++, C#, Java.

6.7.18.14 pow

Returns the value of **a** raised to the power of **b**.



Languages

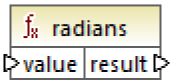
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
a	**xs:double**	Supplies value **a** (the base).
b	**xs:double**	Supplies value **b** (the power).

6.7.18.15 radians

Converts an angle measured in degrees to an approximately equivalent angle measured in radians.



Languages

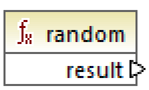
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.18.16 random

Returns a value with a positive sign, greater than or equal to 0.0 and less than 1.0. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

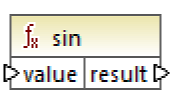


Languages

Built-in, C++, C#, Java.

6.7.18.17 sin

Returns the sine of **value**.



Languages

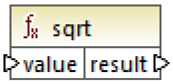
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.18.18 sqrt

Returns the correctly rounded positive square root of **value**.



Languages

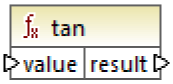
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.18.19 tan

Returns the tangent of **value**.



Languages

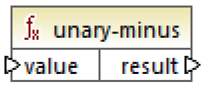
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.18.20 unary-minus

Returns the negation of the signed input value.



Languages

Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:decimal`	The input value.

Example

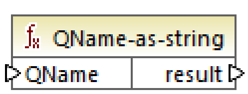
If the input value is **3**, the function returns **-3**. If the input value is **-3**, the function returns **3**.

6.7.19 lang | QName functions

The QName functions from the **lang** library convert Qualified Name (QName) values to strings, and vice versa. Unlike the functions from the **core** library, these functions are available only in the Built-in, Java, C#, or C++ languages.

6.7.19.1 QName-as-string

Returns the string representation of the QName value supplied as argument.



Languages

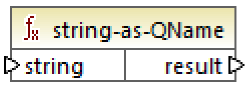
Built-in, C++, C#, Java.

Parameters

Name	Description
QName	The input `xs:QName` value.

6.7.19.2 string-as-QName

Converts the string representation of a QName back to a QName.



Languages

Built-in, C++, C#, Java.

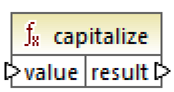
Name	Description
string	The input `string` value.

6.7.20 lang | string functions

The string functions from the **lang** library enable you to process strings (for example, trim, pad, replace, convert strings to upper- or lower- case, and so on).

6.7.20.1 capitalize

Returns the input string **value,** where the first letter of each word is capitalized.



Languages

Built-in, C++, C#, Java.

Parameters

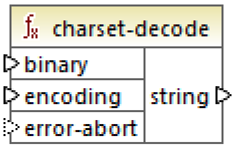
Name	Type	Description
value	`xs:string`	The input value.

Example

If the input value is **`the quick brown fox`**, the function returns **`The Quick Brown Fox`**.

6.7.20.2 charset-decode

The **charset-decode** function takes as input binary data encoded as Base64 text. It decodes data according to the specified character set (for example, "utf-8") and returns the resulting string value. If you need to encode binary data as Base64 text, use the charset-encode [1117] function.



Languages

Built-in.

Parameters

Name	Type	Description
binary-data	`xs:base64Binary`	The binary data as Base64 text.
encoding	`xs:string`	The character set used for encoding (for example, "utf-8").
error-abort	`xs:boolean`	Optional argument that specifies how processing should continue when errors are encountered. Valid values: • **true** - End processing with an exception on the invalid character. • **false** - Continue processing, and replace invalid characters with the replacement character �. The default value is **true**.

Example

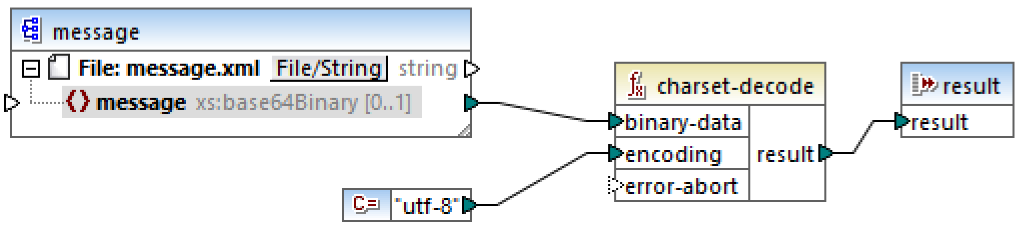
Let's suppose that you would like to decode binary data originating from the following source XML file. Notice that the **message** element contains binary data encoded as Base64 text.

```
<?xml version="1.0" encoding="UTF-8"?>
<message xsi:noNamespaceSchemaLocation="message.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">TG9yZW0gaXBzdW0=</message>
```

The data type of the **message** element is `xs:base64Binary`, as illustrated by the schema:

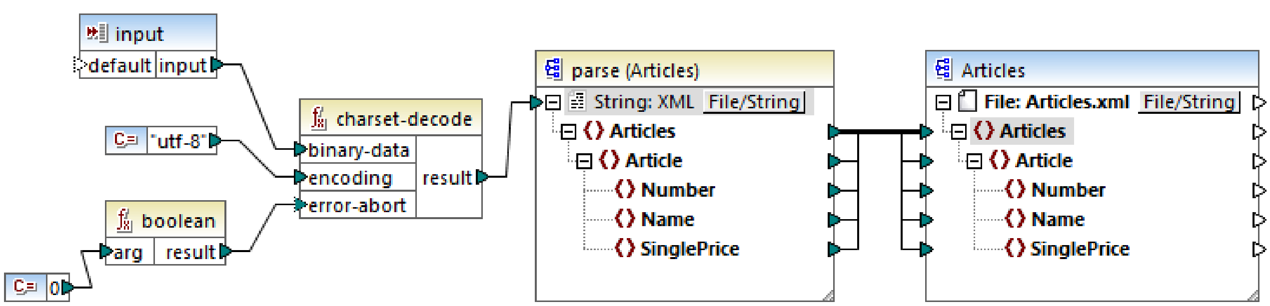
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="message" type="xs:base64Binary"/>
</xs:schema>
```

A mapping that decodes the message above looks as follows:



The mapping in this example outputs the text "Lorem ipsum".

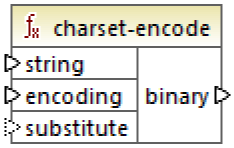
A mapping can also process text or XML files encoded as Base64 data, with the help of a MapForce serialization component. For example, the mapping illustrated below has an **input** parameter which expects Base64 text data. Assuming that the Base64 data was created from an XML file as shown in the [charset-encode](1117) example, you can recreate the original XML file as shown in the mapping below:



In this mapping, the **error-abort** argument gets a **false** value, which was produced with the help of the **boolean** built-in function. This ensures that processing will continue even if invalid characters are encountered. The string result of the function is then passed to an XML parsing component which converts it to an XML file. Note that, in order for XML parsing to be possible, you must have the XSD schema file. For more information, see [Parsing and Serializing Strings](1324).

6.7.20.3 charset-encode

The **charset-encode** function takes as input string data and encodes it as Base64 text. Data is encoded in the specified character set (for example, "utf-8") and returned as `xs:base64Binary` type. If you need to decode binary data previously encoded as Base64 text, use the [charset-decode](1116) function.



Languages

Built-in.

Parameters

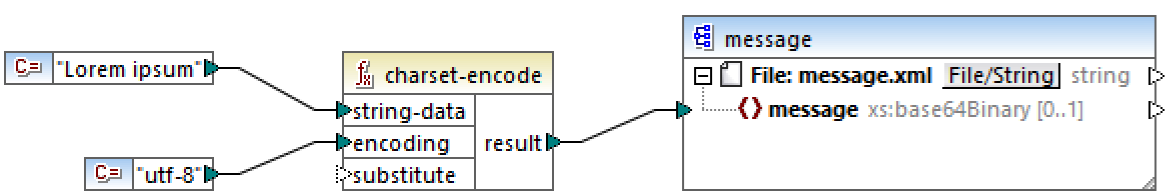
Name	Type	Description
string-data	**xs:string**	The string data to be encoded.
encoding	**xs:string**	The character set used for encoding (for example, "utf-8").
substitute	**xs:string**	Optional argument that specifies a replacement character when invalid characters are encountered. This argument is applicable if you use a non-Unicode encoding. For Unicode encodings, the replacement character is �.

Example

Let's suppose that you would like to encode the text "Lorem ipsum" as Base64 data, using the UTF-8 character set, and write it to a target XML file. The target XML file has a **message** element of `xs:base64Binary` type, as illustrated by the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="message" type="xs:base64Binary"/>
</xs:schema>
```

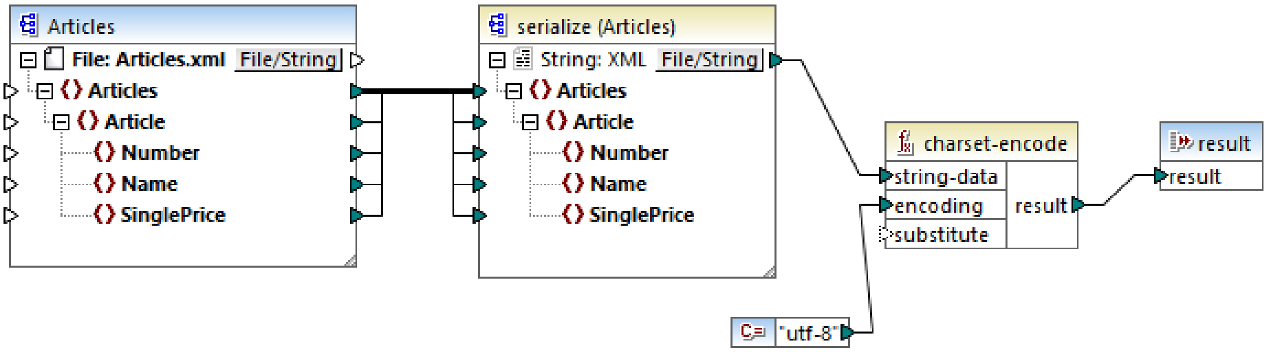
A mapping that performs the Base64 encoding looks as follows:



This mapping produces XML output like the one below (the schema references and XML declaration were skipped):

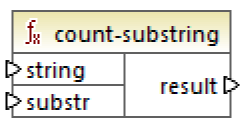
```
<message>TG9yZW0gaXBzdW0=</message>
```

You can also encode text or XML files as Base64, with the help of a MapForce serialization component. For example, the mapping illustrated below serializes a source XML file to a string. The resulting string is then supplied as argument to the **charset-encode** function. Finally, the function result is returned as mapping output, with the help of a simple output component, see Returning String Values from a Mapping 843. For more information about serialization, see Parsing and Serializing Strings 1324.



6.7.20.4 count-substring

Returns an integer value expressing the number of times that **substr** occurs in **string**.



Languages

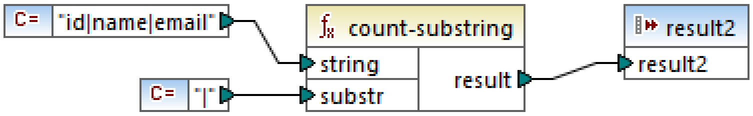
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	xs:string	The input string.
substr	xs:string	The sub-string to test for.

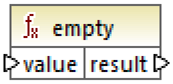
Example

The following mapping returns **2**. This is the number of times that the pipe separator occurs within the input string `id|name|email`.



6.7.20.5 empty

Returns **true** if the input string value is empty; **false** otherwise.



Languages

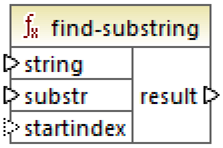
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:string`	The input value.

6.7.20.6 find-substring

Returns the position of the first occurrence of **substr** within **string**. By default, the function starts the search from the first character, which has position (index) 1, but you can optionally specify a specific starting index. If **substr** cannot be found, then the function returns **0**.



Languages

Built-in, C++, C#, Java.

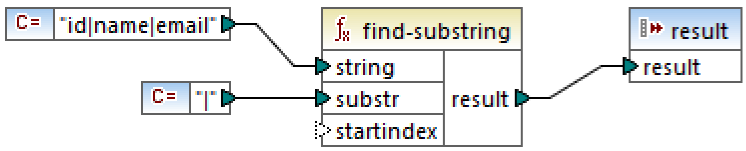
Parameters

Name	Type	Description
string	`xs:string`	The input string.

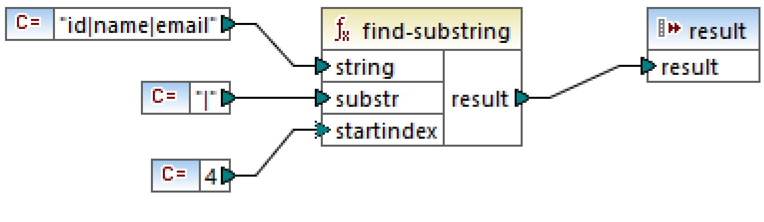
Name	Type	Description
substr	`xs:string`	The sub-string to search for.
startindex	`xs:int`	Optional. Specifies the starting position (index) of the search. If this parameter is not specified, the search starts at position 1.

Example

The following mapping outputs **3**, which is the position of the first occurrence of the pipe character in the input string `id|name|email`.

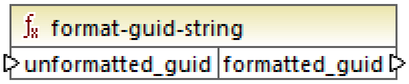


If you specify **4** as starting index, then the function starts searching from the fourth character. Consequently, the mapping below outputs **8**, which is the first occurrence of the pipe character after searching from the fourth character onwards.



6.7.20.7 format-guid-string

Returns a correctly formatted globally unique identifier (GUID) string, typically for use in database fields. See also the create-guid 1103 function.



Languages

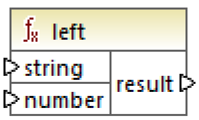
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
unformatted_guid	`xs:string`	The input HEX-encoded string to be formatted.

6.7.20.8 left

Returns a string containing the first **number** characters of the input string.



Languages

Built-in, C++, C#, Java.

Parameters

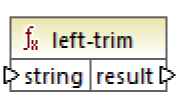
Name	Type	Description
string	`xs:string`	The input string.
number	`xs:int`	Specifies how many characters to return, starting from the beginning of the string.

Example

If the input string is **`This is a sentence`** and number is **4**, the function returns **`This`**.

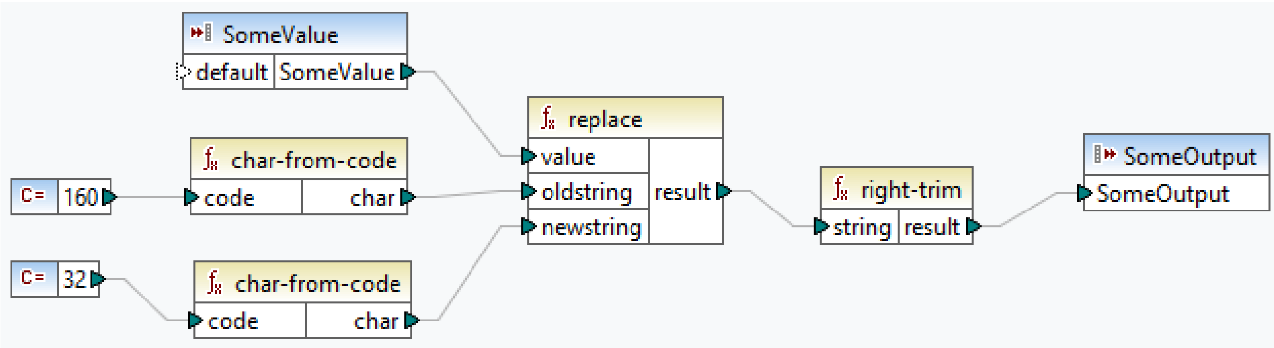
6.7.20.9 left-trim

The **`left-trim`** function (*see screenshot below*) removes leading whitespace characters of a string. Whitespace includes space (U+0020), tab (U+0009), carriage return (U+000D), and line feed (U+000A) characters. For details about whitespaces, see the XML Recommendation.



About non-breaking spaces

The **`left-trim`**, **`right-trim`**, and **`normalize-space`** functions do not remove non-breaking spaces. One of the possible solutions could be to replace the non-breaking space character, whose decimal representation is 160, with the space character, whose decimal representation is 32. The mapping below shows that after the non-breaking space has been replaced, the trimmed `SomeValue` value will be mapped to the target.



If your source component is an Excel file, you can remove extra spaces in Excel using a combination of TRIM, CLEAN, and SUBSTITUTE functions. For details, see *Removing Spaces and Nonprinting Characters from Text*.

Languages

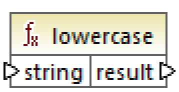
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	The input string.

6.7.20.10 lowercase

Converts the input **string** to lowercase. For Unicode characters, the corresponding lower-case characters (defined by the Unicode consortium) are used.



Languages

Built-in, C++, C#, Java.

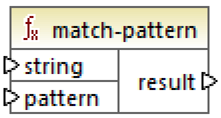
Parameters

Name	Type	Description
string	`xs:string`	The input string.

6.7.20.11 match-pattern

Returns Boolean **true** if the input string matches the regular expression defined by **pattern**; **false** otherwise. See also Regular expressions 975.

Note: When generating C++, C#, or Java code, the advanced features of the regular expression syntax might differ slightly. See the regex documentation of each language for more information.



Languages

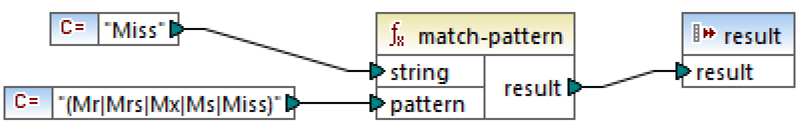
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	The input string.
pattern	`xs:string`	The regular expression to match.

Example

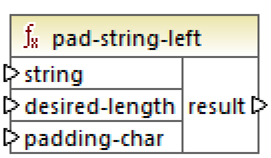
The following mapping validates various person titles. Specifically, the mapping will output **true** for any of the following titles: Mr, Mrs, Mx, Ms, Miss.



If the input string is other than any of the titles listed above, the mapping outputs **false**.

6.7.20.12 pad-string-left

Returns a string which is padded to the left by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.



Languages

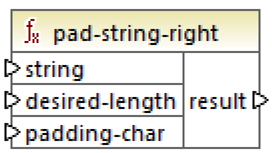
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	Specifies the input string.
desired-length	`xs:int`	Defines the desired length of the string after padding.
padding-char	`xs:string`	Defines the character to use as padding character.

6.7.20.13 pad-string-right

Returns a string which is padded to the right by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.



Languages

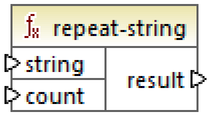
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	Specifies the input string.
desired-length	`xs:int`	Defines the desired length of the string after padding.
padding-char	`xs:string`	Defines the character to use as padding character.

6.7.20.14 repeat-string

Repeats the string supplied as argument *n* times. The **count** argument specifies the number of times to repeat the string.



Languages

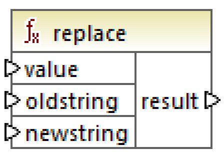
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	The input string.
count	`xs:int`	The number of times to repeat the string.

6.7.20.15 replace

Result is a new string where each instance of **oldstring**, in the input string **value**, is replaced by **newstring**.



Languages

Built-in, C++, C#, Java.

Parameters

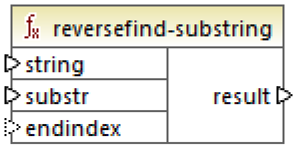
Name	Type	Description
value	`xs:string`	The input value.
oldstring	`xs:string`	The old string to be replaced.
newstring	`xs:string`	The new string to act as replacement.

Example

See Replacing Special Characters 276.

6.7.20.16 reversefind-substring

Returns the position of the last occurrence of **substr** within **string**. By default, the function starts the search from the first character, which has position (index) 1, and ends the search at the last character, but you can optionally specify an ending index. If **substr** cannot be found, then the function returns **0**.



Languages

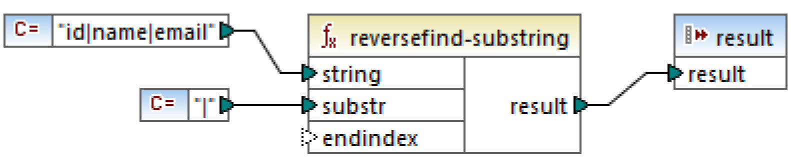
Built-in, C++, C#, Java.

Parameters

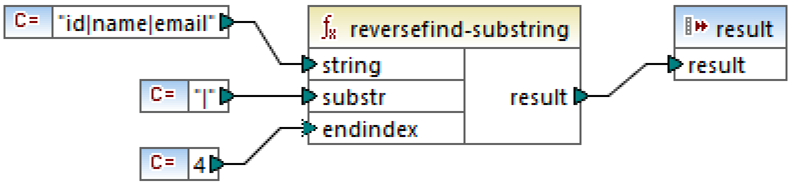
Name	Type	Description
string	xs:string	The input string.
substr	xs:string	The sub-string to search for.
endindex	xs:int	Optional. Specifies the ending position (index) of the search. If this parameter is not specified, the search ends after the last character in **string**.

Example

The following mapping outputs **8**, which is the position of the last occurrence of the pipe character in the input string `id|name|email`.

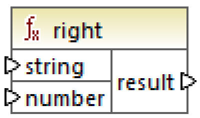


If you specify **4** as ending index, then the function searches up to the fourth character. Consequently, the mapping below outputs **3**.



6.7.20.17 right

Returns a string containing the last **number** characters of the input string.



Languages

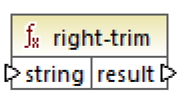
Built-in, C++, C#, Java.

Example

If the input string is **`The brown red fox`** and number is **3**, the function returns **`fox`**.

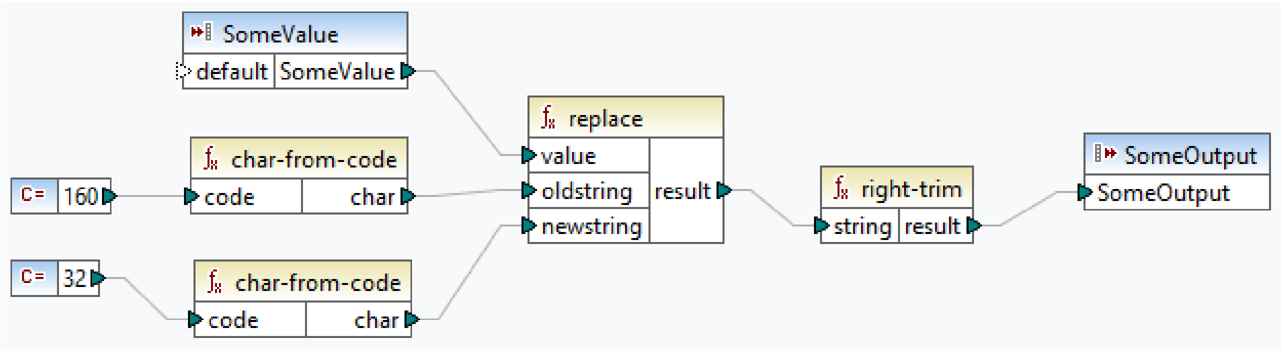
6.7.20.18 right-trim

The **`right-trim`** function (*see screenshot below*) removes trailing whitespace characters of a string. Whitespace includes space (U+0020), tab (U+0009), carriage return (U+000D), and line feed (U+000A) characters. For details about whitespaces, see the XML Recommendation.



About non-breaking spaces

The **`left-trim`**, **`right-trim`**, and **`normalize-space`** functions do not remove non-breaking spaces. One of the possible solutions could be to replace the non-breaking space character, whose decimal representation is 160, with the space character, whose decimal representation is 32. The mapping below shows that after the non-breaking space has been replaced, the trimmed `SomeValue` value will be mapped to the target.



If your source component is an Excel file, you can remove extra spaces in Excel using a combination of TRIM, CLEAN, and SUBSTITUTE functions. For details, see *Removing Spaces and Nonprinting Characters from Text*.

Languages

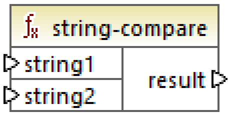
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	The input string.

6.7.20.19 string-compare

The **`string-compare`** function (*see screenshot below*) returns the result of a character by character comparison of two input strings: `string1` and `string2`. The comparison is based on the ASCII codes. Both `string1` and `string2` are of type `xs:string`. The function is case-sensitive. If the strings are equal, the result is 0. If `string1` is less than `string2`, the result is -1. If `string1` is greater than `string2`, the result is 1.



Example:

```
string1: hi
string2: Hit
```

The **`string-compare`** function compares the strings character by character. The comparison stops after the function has detected that the first character of `string1` and the first character of `string2` are different. The result is based on the comparison of the first character of each string. Since `h` is represented as a bigger ASCII code number (104 in the decimal system) than `H` (72 in the decimal system), `string1` is greater than `string2`, and the result of the string comparison is 1. If the first character of `string1` and the first character of `string2` were the same, the function would proceed to analyze the second character and so on.

For simple string comparison with a boolean result, see core | logical functions | equal[1010].

Languages

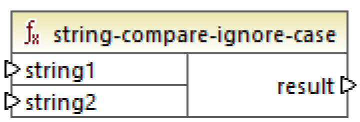
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string1	xs:string	The first input string.
string2	xs:string	The second input string.

6.7.20.20 string-compare-ignore-case

The **string-compare-ignore-case** function (*see screenshot below*) returns the result of a character by character comparison of two input strings: `string1` and `string2`. Both `string1` and `string2` are of type `xs:string`. The function ignores case. The comparison is based on the ASCII codes. If the strings are equal, the result is 0. If `string1` is less than `string2`, the result is -1. If `string1` is greater than `string2`, the result is 1.



Example:

```
string1: hi
string2: Hit
```

The **string-compare-ignore-case** function compares the strings character by character. Even though `h` is represented as a bigger ASCII code number than `H`, these two characters are treated as equal in this function. The second character in both strings is the same. However, `string2` has a third character, whereas `string1` does not. The third character in `string1` has an empty value. The `t` value in `string2` is greater than the empty value in `string1`. Therefore, `string1` is less than `string2`, and the result equals -1.

Languages

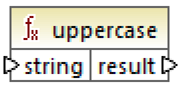
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string1	xs:string	The first input string.
string2	xs:string	The second input string.

6.7.20.21 uppercase

Converts the input **string** to uppercase. For Unicode characters, the corresponding upper-case characters (defined by the Unicode consortium) are used.



Languages

Built-in, C++, C#, Java.

Parameters

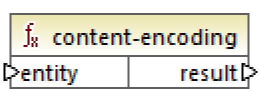
Name	Type	Description
string	`xs:string`	The input string.

6.7.21 mime

The **mime** library contains functions that let MapForce interact with MIME (Multipurpose Internet Mail Extensions) content. You typically need **mime** functions to create Web service calls that send custom MIME content (such as binary data). With **mime** functions, you can also handle the response MIME entity returned by a Web service call.

6.7.21.1 content-encoding

The `content-encoding` function returns the character encoding of the MIME entity supplied as argument. In case of an HTTP message, this is the value of the **charset** parameter from the **Content-Type** header.



Languages

Built-in.

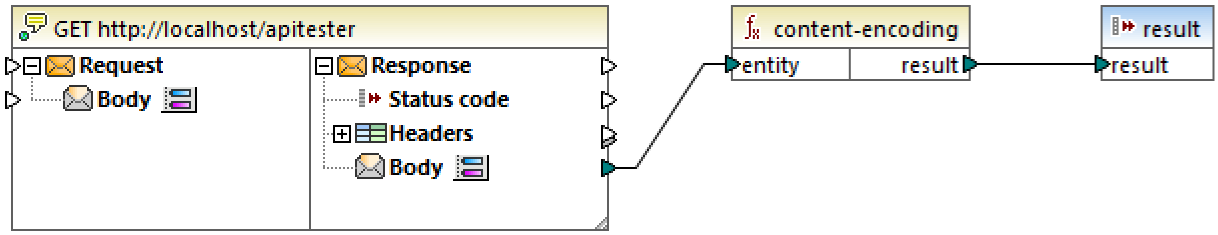
Parameters

Name	Type	Description

entity	mf:MimeEntity	The MIME entity content.

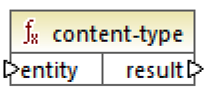
Example

If a mapping calls a Web service that returns a **Content-Type** header as `text/json; charset=utf-8`, then the function output is `utf-8`.



6.7.21.2 content-type

The `content-type` function returns the media type (**Content-Type**) of the MIME entity supplied as argument.



Languages

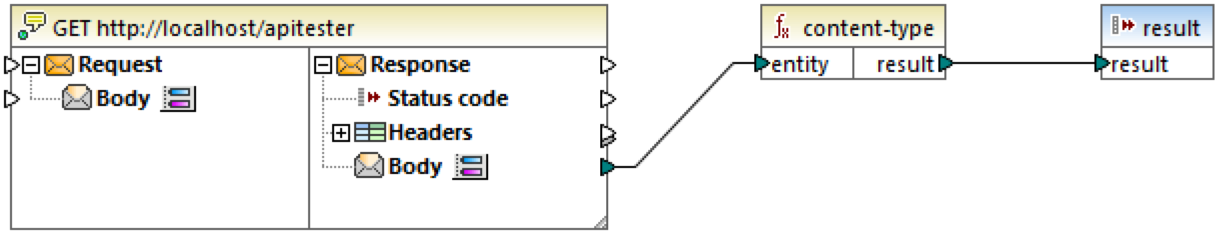
Built-in.

Parameters

Name	Type	Description
entity	mf:MimeEntity	The MIME entity content.

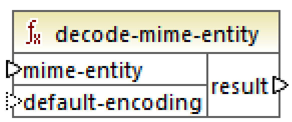
Example

If a mapping calls a Web service that returns a **Content-Type** header as `text/json; charset=utf-8`, then the function output is also `text/json; charset=utf-8`.



6.7.21.3 decode-mime-entity

The **decode-mime-entity** function returns a string that represents the MIME entity decoded using the **charset** parameter included in the **Content-Type** header of the entity. If no **charset** was specified, you can supply a default encoding, by connecting the desired string value to the **default-encoding** parameter.



Languages

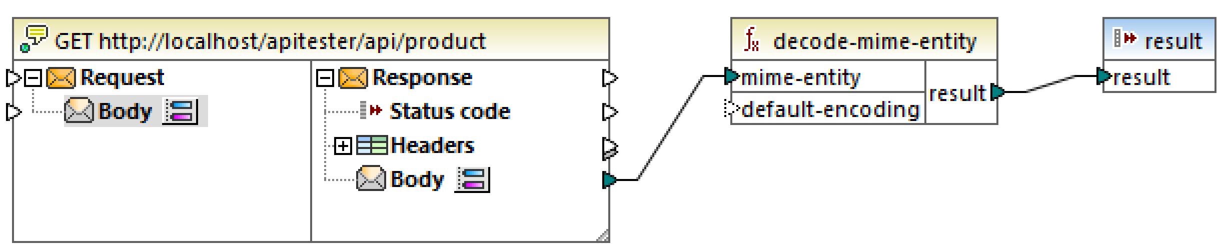
Built-in.

Parameters

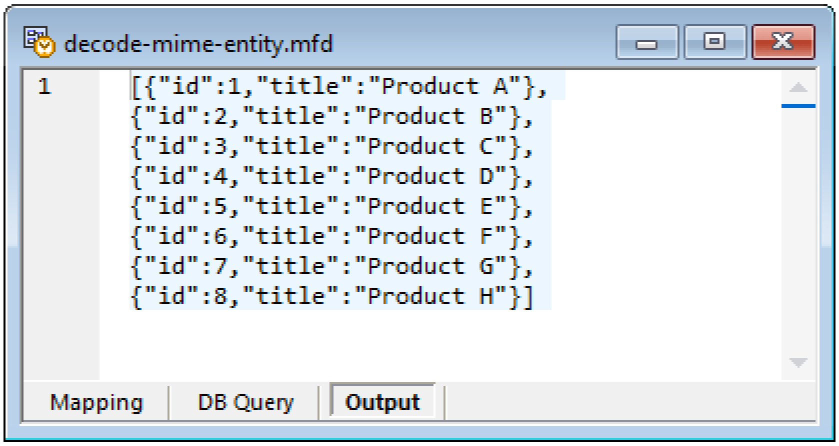
Name	Type	Description
mime-entity	`mf:MimeEntity`	The MIME entity to be decoded.
default-encoding	`xs:string`	The default encoding to use when no encoding was specified in the **charset** parameter.

Example

The following mapping calls a Web service that returns JSON content.

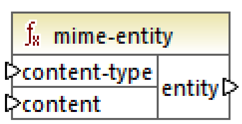


When this mapping runs, the result of the Web service call is supplied as input to the **decode-mime-entity** function. The function decodes the "raw" body returned by the call according to the **charset** parameter of the **Content-Type** header. The result is as follows:



6.7.21.4 mime-entity

The **mime-entity** function returns a MIME entity. The MIME entity is created from the content supplied to the **content** argument, and has the media type supplied to the **type** argument.



Languages

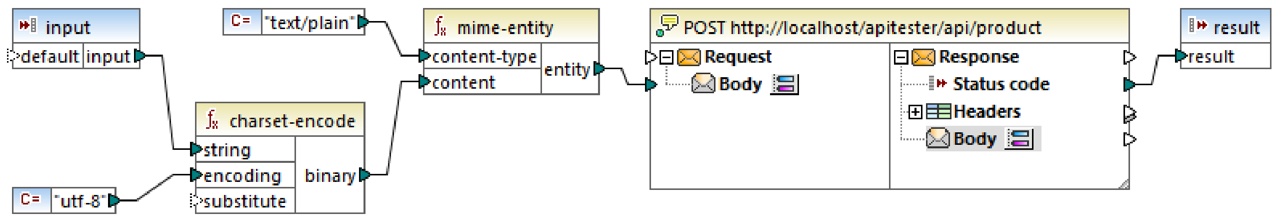
Built-in.

Parameters

Name	Type	Description
content-type	**mf:MediaType**	The media type that describes this MIME entity, for example: **image/png**.
content	**xs:base64Binary**	The actual data encoded using the Base64 encoding scheme. You can encode such data by using the charset-encode 1117 function.

Example

The following mapping calls a Web service that expects unstructured text submitted via POST requests.

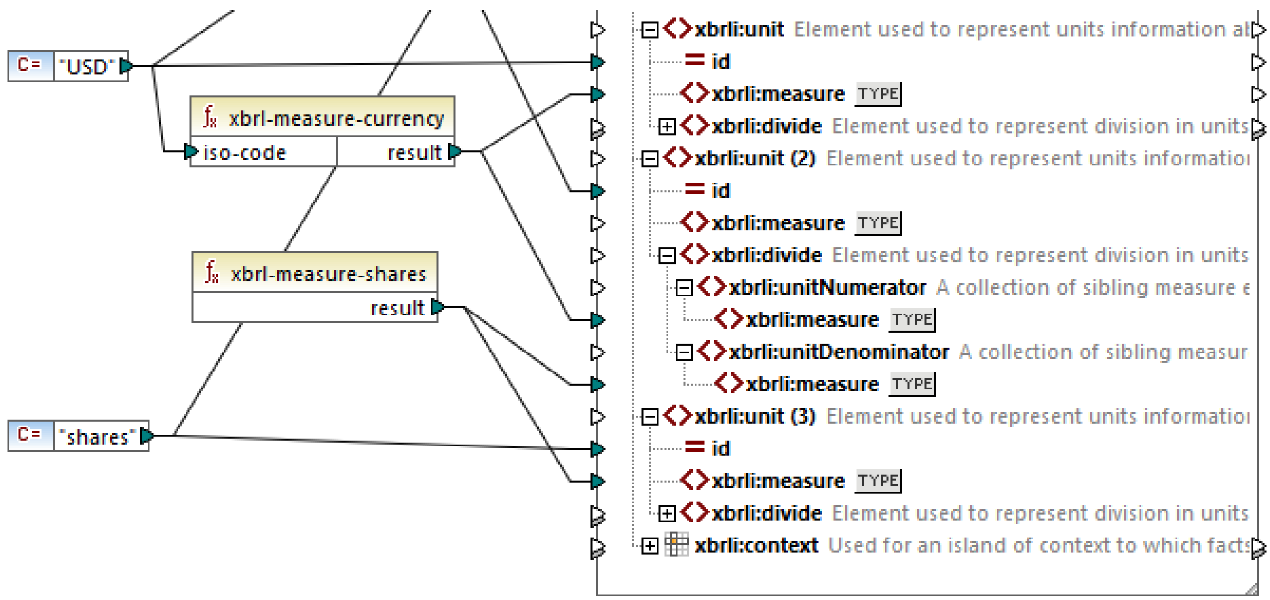


In the mapping above, the input text is supplied as parameter to the mapping, with the help of a simple input component, see [Supplying Parameters to the Mapping](833). The string received as input is converted to binary form with the help of the [charset-encode](1117) function, using the **utf-8** encoding. The **mime-entity** function takes the binary content as argument and returns the body of the request (the MIME entity), which is mapped to the **Body** input of the Web service call.

6.7.22 xbrl

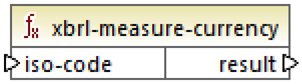
The XBRL library contains functions that convert data into the QName format necessary for XBRL instance or taxonomy files.

The **xbrl:ID** and **xbrl:measure** child elements of the **xbrl:unit** element are mandatory in an XBRL instance file, and must be mapped for the mapping to be valid.



6.7.22.1 xbrl-measure-currency

Returns the QName for items of `monetaryItemType` from an ISO 4217 currency code.



Languages

Built-in, C++, C#, Java.

Parameters

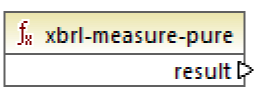
Name	Type	Description
iso-code	`xs:string`	The ISO 4217 currency code.

Example

See the Microsoft Excel to XBRL [790] example.

6.7.22.2 xbrl-measure-pure

Returns the QName for items of rates, percentages or ratios.

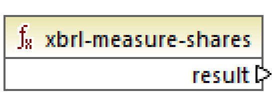


Languages

Built-in, C++, C#, Java.

6.7.22.3 xbrl-measure-shares

Returns the QName for items of `sharesItemType`.



Languages

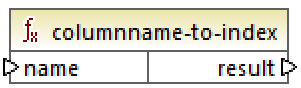
Built-in, C++, C#, Java.

6.7.23 xlsx

The **xlsx** library contains functions that primarily convert data to or from Excel date and time formats. The **`date-to-xlsx`**, **`datetime-to-xlsx`**, and **`time-to-xlsx`** functions are typically useful to write date and time values to a target Excel sheet. The **`xlsx-to-date`**, **`xlsx-to-datetime`**, and **`xslx-to-time`** functions are typically useful to convert date and time values from a source Excel sheet into values of type `xs:date`, `xs:time`, and `xs:dateTime`.

6.7.23.1 columnname-to-index

Returns the index of the column with the given name. The index of column "A" is 1.



Languages

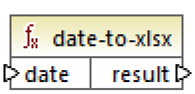
Built-in, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
name	`xs:string`	The column name as it appears in Excel, starting from "A", "B", "C", and so on.

6.7.23.2 date-to-xlsx

Returns the Excel representation of the *date* value extracted from the source, as a serial number that represents the date. By default, in Microsoft Excel for Windows, January the 1st, 1900 is serial number **1**.



Languages

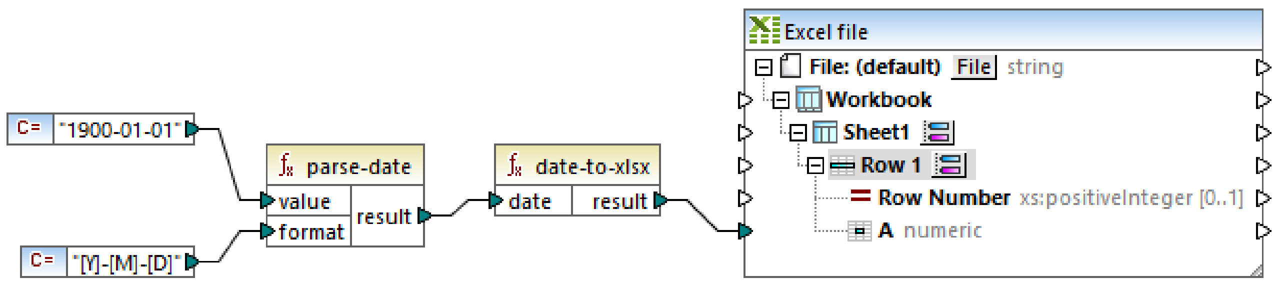
Built-in, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	`xs:date`	The input date value.

Example

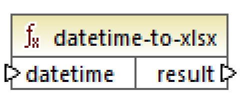
The following mapping converts the string "1900-01-01" to a date, with the help of the **parse-date** function. The obtained `xs:date` value (January the 1st, 1900) is then supplied as argument to the **date-to-xlsx** function, which returns its numerical representation and writes it to an Excel cell. The resulting cell value is 1, as expected.



Note that the format of the target Excel cell is "Numeric". If you right-click the cell in Excel and format it as a date using the **Format Cells** command, the cell value becomes **`1/1/1900`**.

6.7.23.3 datetime-to-xslx

Returns the Excel representation of the *date and time* value extracted from the source.



Languages

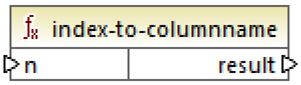
Built-in, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
datetime	`xs:dateTime`	The input value, of type `xs:dateTime`.

6.7.23.4 index-to-columnname

Returns the name of the Excel column (for example, "A", "B", "C") given the column index (for example, 1, 2, 3). The name of column 1 is "A".



Languages

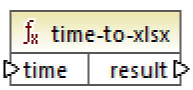
Built-in, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
n	`xs:int`	The 1-based index of the column.

6.7.23.5 time-to-xlsx

Returns the Excel representation of the *time* value extracted from the source.



Languages

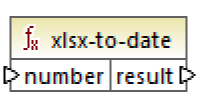
Built-in, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	`xs:time`	The input value, of type `xs:time`.

6.7.23.6 xlsx-to-date

Returns an `xs:date` value given the numeric representation of the date in Excel.



Languages

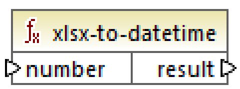
Built-in, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
number	`xs:double`	The serial number of the date in Excel.

6.7.23.7 xlsx-to-datetime

Returns an `xs:dateTime` value given the numeric representation of the date and time value in Excel.



Languages

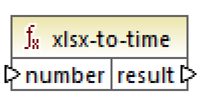
Built-in, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
number	`xs:double`	The representation of the date and time value in Excel.

6.7.23.8 xlsx-to-time

Returns an `xs:time` value given the numeric representation of the time value in Excel.



Languages

Built-in, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

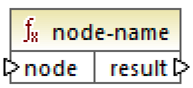
Name	Type	Description
number	`xs:double`	The representation of the time in Excel.

6.7.24 xpath2 | accessors

Functions from the **xpath2 | accessors** sub-library retrieve information about XML nodes or items. These functions are available when either the XSLT2 or XQuery languages are selected.

6.7.24.1 base-uri

The **`base-uri`** function takes a node as input, and returns the URI of the XML resource containing the node. The output is of type `xs:string`.



Languages

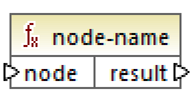
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
node	`mf:node`	The input node.

6.7.24.2 node-name

The **`node-name`** function takes a node as its input argument and returns its QName. When the QName is represented as a string, it takes the form of `prefix:localname` if the node has a prefix, or `localname` if the node has no prefix. To obtain the namespace URI of a node, use the **`namespace-uri-from-QName`** 1029 function.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

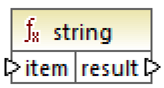
Parameters

Name	Type	Description
node	`mf:node`	The input node.

6.7.24.3 string

The **`string`** function works like the `xs:string` constructor: it converts its argument to `xs:string`.

When the input argument is a value of an atomic type (for example `xs:decimal`), this atomic value is converted to a value of `xs:string` type. If the input argument is a node, the string value of the node is extracted. (The string value of a node is a concatenation of the values of the node's descendant nodes.)



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
item	`mf:item`	The input value.

6.7.25 xpath2 | anyURI functions

The **xpath2 | anyURI** sub-library contains the **`resolve-uri`** function. This function is available when either the XSLT2 or XQuery languages are selected.

6.7.25.1 resolve-uri

The **`resolve-uri`** function takes a relative URI as its first argument and resolves it against the base URI in the second argument. The result is of data type `xs:string`. The function's implementation treats both inputs as strings; no checks are performed as to whether the resources identified by these URIs actually exist.

Languages

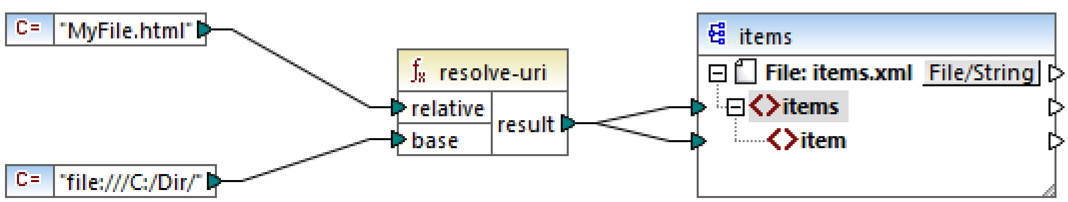
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
relative	`xs:string`	The relative URI to be resolved against the base.
base	`xs:string`	The base URI.

Example

In the mapping illustrated below, the first argument provides the relative URI `MyFile.html`, and the second argument provides the base URI `file:///C:/Dir/`. The resolved URI will be a concatenation of both, so `file:///C:/Dir/MyFile.html`.

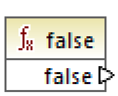


6.7.26 xpath2 | boolean functions

The Boolean functions `true` and `false` take no argument and return the boolean constant values **true** and **false**, respectively. They can be used where a constant boolean value is required.

6.7.26.1 false

Returns the Boolean value **false**.

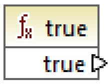


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.26.2 true

Returns the Boolean value **true**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.27 xpath2 | constructors

The functions in the "constructors" sub-library of the XPath 2.0 library construct specific data types from the input text. The following table lists the available constructor functions.

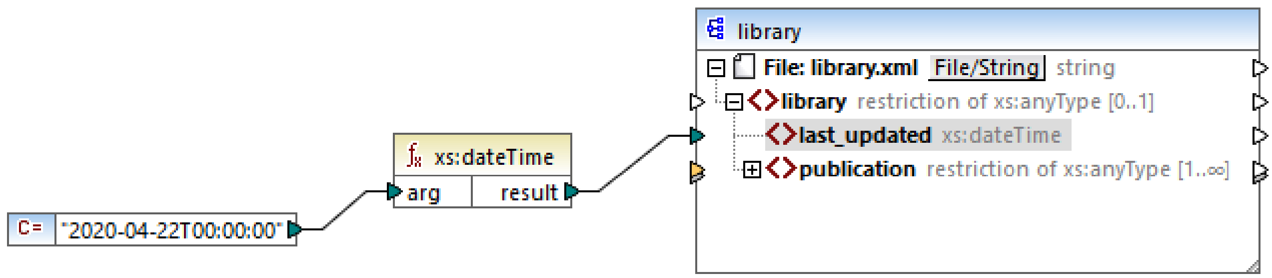
xs:ENTITY	xs:double	xs:nonPositiveInteger
xs:ID	xs:duration	xs:normalizedString
xs:IDREF	xs:float	xs:positiveInteger
xs:NCName	xs:gDay	xs:short
xs:NMTOKEN	xs:gMonth	xs:string
xs:Name	xs:gMonthDay	xs:time
xs:QName	xs:gYear	xs:token
xs:anyURI	xs:gYearMonth	xs:unsignedByte
xs:base64Binary	xs:hexBinary	xs:unsignedInt
xs:boolean	xs:int	xs:unsignedLong
xs:byte	xs:integer	xs:unsignedShort
xs:date	xs:language	xs:untypedAtomic
xs:dateTime	xs:long	xs:yearMonthDuration
xs:dayTimeDuration	xs:negativeInteger	
xs:decimal	xs:nonNegativeInteger	

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Example

Typically, the lexical format of the input text must be the one expected of the data type to be constructed. Otherwise, the transformation will not be successful. For example, to construct an `xs:dateTime` value using the **`xs:dateTime`** constructor function, the input text must have the lexical format of the `xs:dateTime` data type, which is **`YYYY-MM-DDTHH:mm:ss`**.



In the mapping illustrated above, a string constant (**`"2020-04-28T00:00:00"`**) has been used to provide the input argument of the function. The input could also have been obtained from an item in the source document. The **`xs:dateTime`** function returns the value **`2020-04-28T00:00:00`** of type `xs:dateTime`.

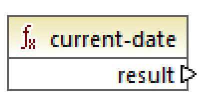
To view the expected data type of a mapping item (including the data type of function arguments), move the mouse cursor over the respective input or output connector.

6.7.28 xpath2 | context functions

The context functions from the **`xpath2`** library provide miscellaneous information about the current date and time, the default collation used by the processor, the size of the current sequence, and the position of the current node.

6.7.28.1 current-date

Returns the current date (`xs:date`) from the system clock.

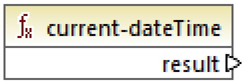


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.28.2 current-dateTime

Returns the current date and time (`xs:dateTime`) from the system clock.

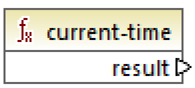


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.28.3 current-time

Returns the current time (`xs:time`) from the system clock.



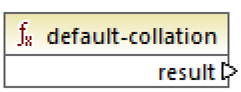
Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.28.4 default-collation

The **`default-collation`** function takes no argument and returns the default collation, that is, the collation that is used when no collation is specified for a function where one can be specified.

Comparisons, including for the **`max-string`** and **`min-string`** functions, are based on the default collation.

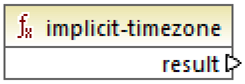


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.28.5 implicit-timezone

Returns the value of the "implicit timezone" property from the evaluation context.

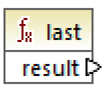


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.28.6 last

Returns the number of items in the sequence of items currently being processed. Importantly, the sequence of items is determined by the current mapping context[1277], as described in the example below.



Languages

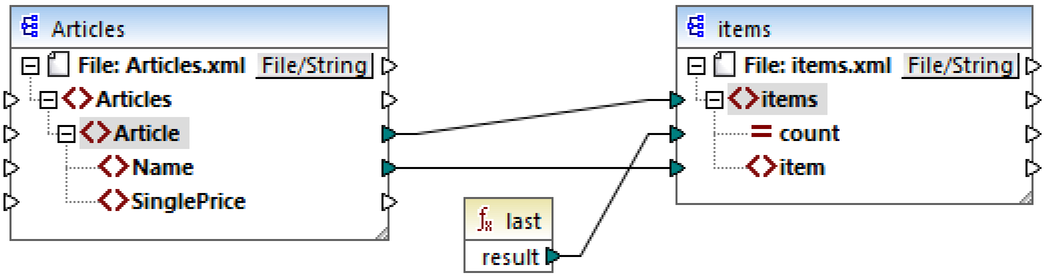
XQuery, XSLT 2.0, XSLT 3.0.

Example

Let's suppose that you have the following source XML file:

```
<Articles>
   <Article>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>
   <Article>
      <Name>Socks</Name>
      <SinglePrice>2.30</SinglePrice>
   </Article>
   <Article>
      <Name>Jacket</Name>
      <SinglePrice>57.50</SinglePrice>
   </Article>
</Articles>
```

Your goal is to copy data to an XML file with a different schema. Also, the count of all items must be saved to the target XML file. This can be achieved by a mapping like the one below:



In the example above, the **last** function returns the position of the last node in the current parent context and populates the **count** attribute with value **3**.

```
<items count="3">
   <item>T-Shirt</item>
   <item>Pants</item>
   <item>Jacket</item>
</items>
```

Note that value **3** is the position of the last item (and thus the count of all items) in the mapping context created by the connection between **Article** and **items**. If this connection did not exist, items would still be copied to the target, but the **last** function would return value **1** incorrectly, because it would have no parent context[1284] to iterate over. (More precisely, it would use the default implicit context created between the root items of both components, which produces a sequence of 1 item, not 3 as expected).

It is generally advisable to use the **count**[983] function from the **core** library instead of the **last** function, because the former has a **parent-context** argument, which enables you to alter the mapping context explicitly.

6.7.29 xpath2 | durations, date and time functions

The duration, date and time functions from the **xpath2** library enable you to adjust the time zone in date and time values, extract particular components from date, time, and duration values, and subtract date and time values.

Adjusting the time zone

To adjust the time zone in date and time values, the following functions are available:

- **adjust-date-to-timezone**
- **adjust-date-to-timezone** (with timezone argument)
- **adjust-dateTime-to-timezone**
- **adjust-dateTime-to-timezone** (with timezone argument)
- **adjust-time-to-timezone**
- **adjust-time-to-timezone** (with timezone argument)

Each of these related functions takes an `xs:date`, `xs:time`, or `xs:dateTime` value as the first argument and adjusts the input by adding, removing, or modifying the time zone component depending on the value of the second argument (if one is present).

The following situations are possible when the first argument contains no time zone (for example, the date `2020-01` or the time `14:00:00`).

- If the **timezone** argument is present, the result will contain the time zone specified in the second argument. The time zone in the second argument is added.
- If the **timezone** argument is absent, the result will contain the implicit timezone, which is the system's time zone. The system's time zone is added.
- If the **timezone** argument is empty, the result will contain no time zone.

The following situations are possible when the first argument contains a time zone (for example, the date `2020-01-01+01:00` or the time `14:00:00+01:00`).

- If the **timezone** argument is present, the result will contain the time zone specified in the second argument. The original time zone is replaced by the timezone in the second argument.
- If the **timezone** argument is absent, the result will contain the implicit time zone, which is the system's time zone. The original time zone is replaced by the system's time zone.
- If the **timezone** argument is empty, the result will contain no time zone.

Extracting components of dates and times

To extract numeric values such as hours, minutes, days, months, and so on from date and time values, the following functions are available:

- **day-from-date**
- **day-from-dateTime**
- **hours-from-dateTime**
- **hours-from-time**
- **minutes-from-dateTime**
- **minutes-from-time**
- **month-from-date**
- **month-from-dateTime**
- **seconds-from-dateTime**
- **seconds-from-time**
- **timezone-from-date**
- **timezone-from-dateTime**
- **timezone-from-time**
- **year-from-date**
- **year-from-dateTime**

Each of these functions extracts a particular component from `xs:date`, `xs:time`, `xs:dateTime`, and `xs:duration` values. The result will be either `xs:integer` or `xs:decimal`.

Extracting components of durations

To extract time components from durations, the following functions are available:

- **days-from-duration**
- **hours-from-duration**
- **minutes-from-duration**
- **months-from-duration**
- **seconds-from-duration**
- **years-from-duration**

The duration must be specified either as `xs:yearMonthDuration` (for extracting years and months) or `xs:dayTimeDuration` (for extracting days, hours, minutes, and seconds). All functions returns a result of type `xs:integer`, with the exception of the **`seconds-from-duration`** function, which returns `xs:decimal`.

Subtracting date and time values

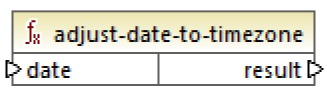
To subtract date and time values, the following functions are available:

- **`subtract-dateTimes`**
- **`subtract-dates`**
- **`subtract-times`**

Each of the subtraction functions enables you to subtract one time value from another and return a duration value.

6.7.29.1 adjust-date-to-timezone

Adjusts an `xs:date` value to the implicit time zone in the evaluation context (the system's time zone).



Languages

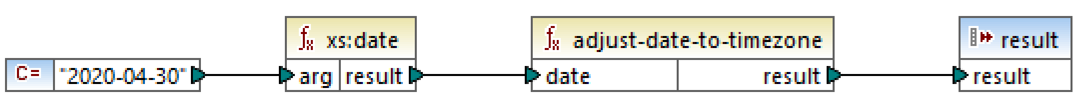
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**`xs:date`**	The input value of type `xs:date`.

Example

The following mapping constructs an `xs:date` from a string and supplies it as argument to the **`adjust-date-to-timezone`** function.

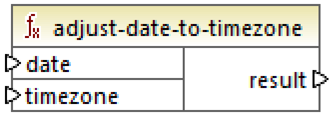


XSLT 2.0 mapping

If the mapping runs on a computer where the system time zone is +02:00, the function adjusts the date value to include the system's time zone. Consequently, the mapping output is **`2020-04-30+02:00`**.

6.7.29.2 adjust-date-to-timezone

Adjusts an `xs:date` value to a specific time zone, or to no time zone at all. If the **timezone** argument is an empty sequence, the function returns an `xs:date` without a time zone. Otherwise, it returns an `xs:date` with a time zone.



Languages

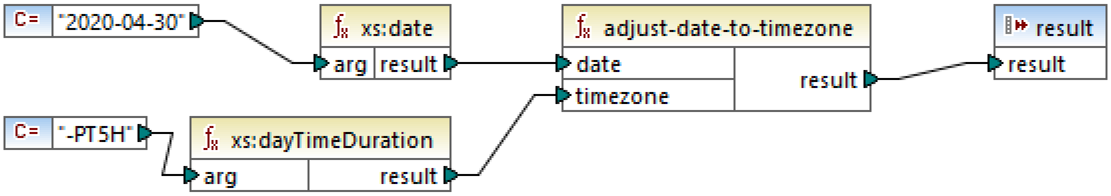
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**xs:date**	The input value of type `xs:date`.
timezone	**xs:dayTimeDuration**	The time zone expressed as an `xs:dayTimeDuration` value. The value can be negative. For example, a time zone value of -5 hours can be expressed as **`-PT5H`**.

Example

The following mapping constructs both parameters to the **`adjust-date-to-timezone`** function from strings, using the corresponding XPath 2 constructor 1144 functions. The goal of the mapping is to adjust the time zone to -5 hours. This time zone can be expressed as **`-PT5H`**.

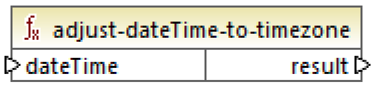


XSLT 2.0 mapping

The function adjusts the date value to the time zone supplied as argument. Consequently, the mapping output is **`2020-04-30-05:00`**.

6.7.29.3 adjust-dateTime-to-timezone

Adjusts an `xs:dateTime` value to the implicit time zone in the evaluation context (the system's time zone).



Languages

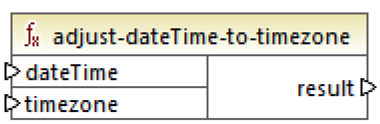
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**xs:dateTime**	The input value of type `xs:dateTime`.

6.7.29.4 adjust-dateTime-to-timezone

Adjusts an `xs:dateTime` value to a specific time zone, or to no time zone at all. If the **timezone** argument is an empty sequence, the function returns an `xs:dateTime` without a time zone. Otherwise, it returns an `xs:dateTime` with a time zone.



Languages

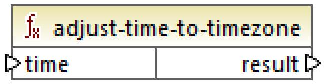
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**xs:dateTime**	The input value of type `xs:dateTime`.
timezone	**xs:dayTimeDuration**	The time zone expressed as an `xs:dayTimeDuration` value. The value can be negative. For example, a time zone value of -5 hours can be expressed as **`-PT5H`**.

6.7.29.5 adjust-time-to-timezone

Adjusts an `xs:time` value to the implicit time zone in the evaluation context (the system's time zone).



Languages

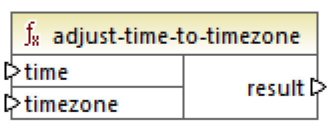
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	**`xs:time`**	The input value of type `xs:time`.

6.7.29.6 adjust-time-to-timezone

Adjusts an `xs:time` value to a specific time zone, or to no time zone at all. If the **timezone** argument is an empty sequence, the function returns an `xs:time` without a time zone. Otherwise, it returns an `xs:time` with a time zone.



Languages

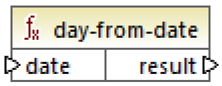
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	**`xs:time`**	The input value of type `xs:time`.
timezone	**`xs:dayTimeDuration`**	The time zone expressed as an `xs:dayTimeDuration` value. The value can be negative. For example, a time zone value of -5 hours can be expressed as **`-PT5H`**.

6.7.29.7 day-from-date

Returns an `xs:integer` representing the day part of the `xs:date` value supplied as argument.



Languages

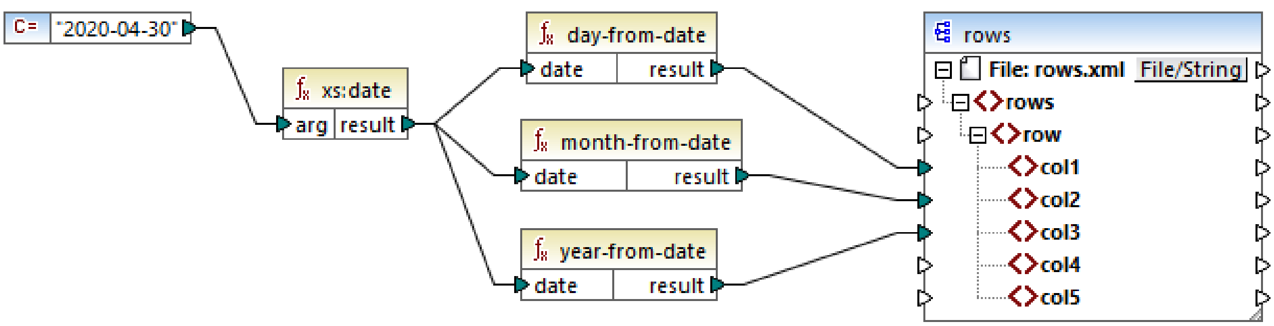
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**`xs:date`**	The input value of type `xs:date`.

Example

The following mapping converts a string to `xs:date` using the **`xs:date`** constructor function. The **`day-from-date`**, **`month-from-date`**, and **`year-from-date`** functions each extract the respective part of the date and write it to a separate item in the target XML file.



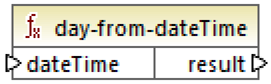
XQuery 1.0 mapping

The mapping output is as follows:

```
<rows>
   <row>
      <col1>30</col1>
      <col2>4</col2>
      <col3>2020</col3>
   </row>
</rows>
```

6.7.29.8 day-from-dateTime

Returns an `xs:integer` representing the day part of the `xs:dateTime` value supplied as argument.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**xs:dateTime**	The input value of type `xs:dateTime`.

6.7.29.9 days-from-duration

Returns an `xs:integer` representing the "days" component of the canonical representation of the duration value supplied as argument.

Languages

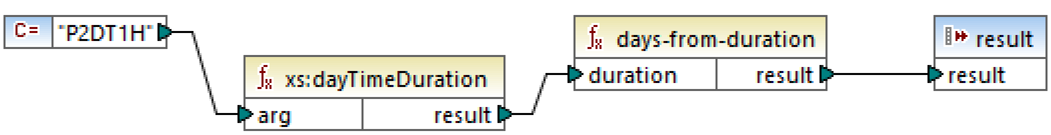
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	**xs:duration**	The input value of type `xs:duration`.

Example

The mapping illustrated below constructs the `xs:dayTimeDuration` of **`P2DT1H`** (2 days and 1 hours) and supplies it as input to the **`days-from-duration`** function. The result is **2**.



XSLT 2.0 mapping

Note: If the duration is **`P1DT24H`** (1 day and 24 hours), the function returns **2**, not **1**. This is because the canonical representation of **`P1DT24H`** is actually **`P2D`** (2 days).

6.7.29.10 hours-from-dateTime

Returns an `xs:integer` representing the hours part of the `xs:dateTime` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**`xs:dateTime`**	The input value of type `xs:dateTime`.

6.7.29.11 hours-from-duration

Returns an `xs:integer` representing the hours component of the canonical representation of the duration value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	**`xs:duration`**	The input value of type `xs:duration`.

Example

If the duration is **`PT1H60M`** (1 hour and 60 minutes), the function returns **2**, not **1**. This is because the canonical representation of **`PT1H60M`** is actually **`PT2H`** (2 hours).

6.7.29.12 hours-from-time

Returns an `xs:integer` representing the hours part of the `xs:time` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	**`xs:time`**	The input value of type `xs:time`.

6.7.29.13 minutes-from-dateTime

Returns an `xs:integer` representing the minutes part of the `xs:dateTime` supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**`xs:dateTime`**	The input value of type `xs:dateTime`.

6.7.29.14 minutes-from-duration

Returns an `xs:integer` representing the minutes component of the canonical representation of the duration supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	**`xs:duration`**	The input value of type `xs:duration`.

Example

If the duration is **`PT1M60S`** (1 minute and 60 seconds), the function returns **2**, not **1**. This is because the canonical representation of **`PT1M60S`** is actually **`PT2M`** (2 minutes).

6.7.29.15 minutes-from-time

Returns an `xs:integer` representing the minutes part of the `xs:time` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	**`xs:time`**	The input value of type `xs:time`.

6.7.29.16 month-from-date

Returns an `xs:integer` representing the month part of the `xs:date` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**`xs:date`**	The input value of type `xs:date`.

6.7.29.17 month-from-dateTime

Returns an `xs:integer` representing the month part of the `xs:dateTime` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**`xs:dateTime`**	The input value of type `xs:dateTime`.

6.7.29.18 months-from-duration

Returns an `xs:integer` representing the months component in the canonical representation of the duration value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	**xs:duration**	The input value of type `xs:duration`.

6.7.29.19 seconds-from-dateTime

Returns an xs:integer representing the seconds component in the localized value of dateTime.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**xs:dateTime**	

6.7.29.20 seconds-from-duration

Returns an `xs:integer` representing the seconds component in the canonical representation of the duration value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	**xs:duration**	The input value of type `xs:duration`.

6.7.29.21 seconds-from-time

Returns an `xs:integer` representing the seconds part of the `xs:time` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	`xs:time`	The input value of type `xs:time`.

6.7.29.22 subtract-dateTimes

Returns the `xs:dayTimeDuration` that corresponds to the difference between the normalized value of **dateTime1** and the normalized value of **dateTime2**.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime1	`xs:dateTime`	The first input value.
dateTime2	`xs:dateTime`	The second input value.

6.7.29.23 subtract-dates

Returns the `xs:dayTimeDuration` that corresponds to the difference between the normalized value of **date1** and the normalized value of **date2**.

Languages

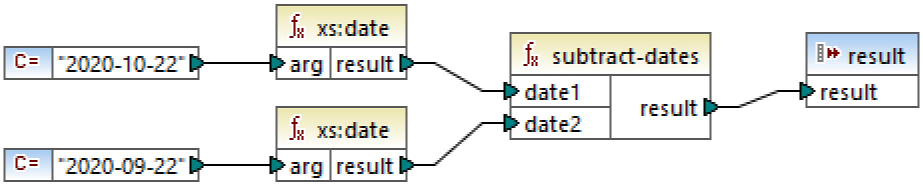
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date1	`xs:date`	The first input value.
date2	`xs:date`	The second input value.

Example

The mapping illustrated below subtracts two dates (2020-10-22 minus 2020-09-22). The result is the value **`P30D`** of type `xs:dayTimeDuration`., which represents a duration of 30 days.



6.7.29.24 subtract-times

Returns the `xs:dayTimeDuration` that corresponds to the difference between the normalized value of **time1** and the normalized value of **time2**.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time1	`xs:time`	The first input value.
time2	`xs:time`	The second input value.

6.7.29.25 timezone-from-date

Returns the timezone component of the date supplied as argument. The result is an `xs:dayTimeDuration` that indicates deviation from UTC; its value may range from +14:00 to -14:00 hours, both inclusive.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	`xs:date`	The input value of type `xs:date`.

6.7.29.26 timezone-from-dateTime

Returns the timezone component of the `xs:dateTime` value supplied as argument. The result is an `xs:dayTimeDuration` that indicates deviation from UTC; its value may range from +14:00 to -14:00 hours, both inclusive.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**xs:dateTime**	The input value of type `xs:dateTime`.

6.7.29.27 timezone-from-time

Returns the timezone component of the `xs:time` value supplied as argument. The result is an `xs:dayTimeDuration` that indicates deviation from UTC; its value may range from +14:00 to -14:00 hours, both inclusive.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	**xs:time**	The input value of type `xs:time`.

6.7.29.28 year-from-date

Returns an `xs:integer` representing the year part of the `xs:date` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**xs:date**	The input value of type `xs:date`.

6.7.29.29 year-from-dateTime

Returns an `xs:integer` representing the year part of the `xs:dateTime` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	`xs:dateTime`	The input value of type `xs:dateTime`.

6.7.29.30 years-from-duration

Returns an `xs:integer` representing the years component in the canonical lexical representation of the duration value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	`xs:duration`	The input value of type `xs:duration`.

6.7.30 xpath2 | node functions

The node functions from the **xpath2** library provide information about nodes (items) on a mapping component.

The **`lang`** function takes a string argument that identifies a language code (such as "en"). The function returns **true** or **false** depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function.

The **`local-name`**, **`name`**, and **`namespace-uri`** functions, return, respectively, the local name, name, and namespace URI of the input node. For example, for the node **altova:Products**, the local name is **`Products`**, the name is **`altova:Products`**, and the namespace URI is the URI of the namespace to which the altova: prefix is bound (see the example given for the **`local-name`** 1166 function). Each of these three functions has two variants:

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the **`lang`** 1164 function).

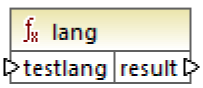
- With an argument that must be a node: the function is applied to the connected node.

The `number` function takes a node as input, atomizes the node (that is, extracts its contents), and converts the value to a decimal and returns the converted value. There are two variants of the `number` function:

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the `lang`[1164] function).
- With an argument that must be a node: the function is applied to the connected node.

6.7.30.1 lang

Returns **true** if the context node has an `xml:lang` attribute with a value that either matches exactly the **testlang** argument, or is a subset of it. Otherwise, the function returns **false**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

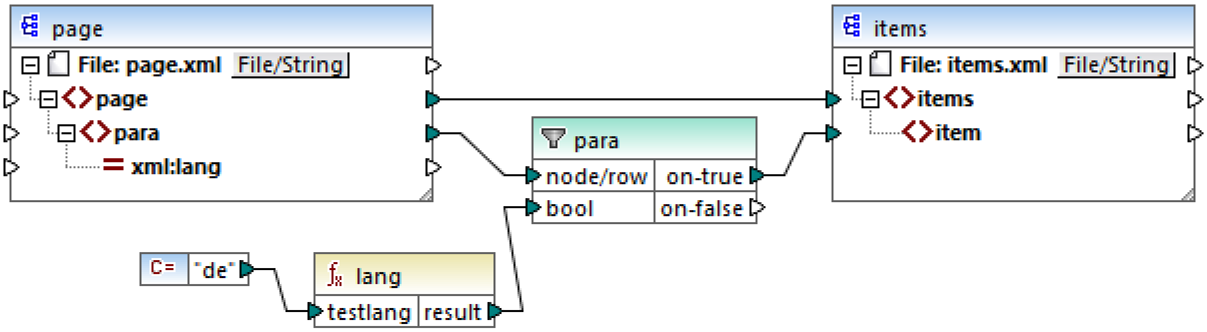
Name	Type	Description
testlang	`xs:string`	The language code to check, for example, "en".

Example

The following XML contains **para** elements with different values for the `xml:lang` attribute.

```
<page>
   <para xml:lang="en">Good day!</para>
   <para xml:lang="fr">Bonjour!</para>
   <para xml:lang="de-AT">Grüss Gott!</para>
   <para xml:lang="de-DE">Guten Tag!</para>
   <para xml:lang="de-CH">Grüezi!</para>
</page>
```

The mapping illustrated below filters only the German paragraphs, regardless of the country variant, with the help of the `lang` function.



XSLT 2.0 mapping

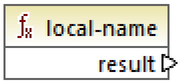
In the mapping above, for each **para** in the source, an **item** is created in the target, conditionally. The condition is provided by a filter which passes on to the target only those nodes where the **lang** function returns **true**. That is, only those nodes that have the `xml:lang` attribute set to "de" (or a subset of "de") will satisfy the filter's condition. Consequently, the mapping output is as follows:

```
<items>
   <item>Grüss Gott!</item>
   <item>Guten Tag!</item>
   <item>Grüezi!</item>
</items>
```

Note that the **lang** function operates in the context of each **para**, because of the parent connection between **para** and **item**, see also The Mapping Context[1277].

6.7.30.2 local-name

Returns the local part of the name of the context node as an `xs:string`. This is a parameterless variant of the **local-name** function where the context node is determined by the connections in your mapping. To specify a node explicitly, use the **local-name**[1166] function that takes an input node as parameter.

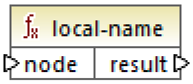


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.30.3 local-name

Returns the local part of the name of **node** as an `xs:string`.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

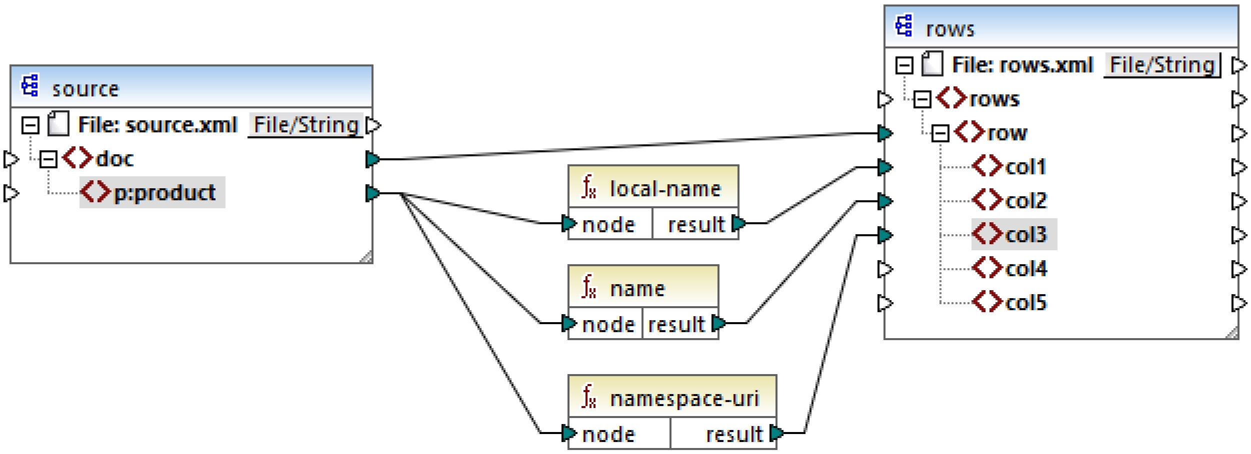
Name	Type	Description
node	**node()**	The input node.

Example

In the following XML file, the name of the `p:product` element is a prefixed qualified name (QName). The prefix "p" is mapped to the namespace "http://mycompany.com".

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:p="http://mycompany.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="source.xsd">
   <p:product/>
</doc>
```

The following mapping extracts the local name, the name, and the namespace URI of the node and writes these values to a target file:



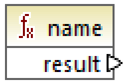
XSLT 2.0 mapping

The mapping output is displayed below. Each **col** item lists the result of the **local-name**, **name**, and **namespace-uri** functions, respectively.

```
<rows>
   <row>
      <col1>product</col1>
      <col2>p:product</col2>
      <col3>http://mycompany.com</col3>
   </row>
</rows>
```

6.7.30.4 name

Returns the name of the context node. This is a parameterless variant of the **name** function where the context node is determined by the connections in your mapping. To specify a node explicitly, use the **name** 1168 function that takes an input node as parameter.

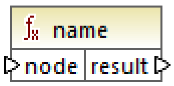


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.30.5 name

Returns the name of a node.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

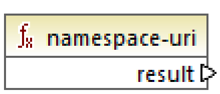
Name	Type	Description
node	`node()`	The input node.

Example

See the example given for the **`local-name`**[1166] function.

6.7.30.6 namespace-uri

Returns the namespace URI of the QName of the context node, as an `xs:string`. This is a parameterless variant of the **`namespace-uri`** function where the context node is determined by the connections in your mapping. To specify a node explicitly, use the **`namespace-uri`**[1168] function that takes an input node as parameter.

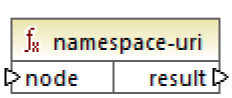


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.30.7 namespace-uri

Returns the namespace URI of the QName of **node**, as an `xs:string`.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
node	`node()`	The input node.

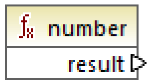
Example

See the example given for the `local-name` 1166 function.

6.7.30.8 number

Returns the value of the context node, converted to an `xs:double`. This is a parameterless variant of the **`number`** function where the context node is determined by the connections in your mapping. To specify a node explicitly, use the `number` 1169 function that takes an input node as parameter.

The only types that can be converted to numbers are Booleans, numeric strings, and other numeric types. Non-numeric input values (such as a non-numeric string) result in NaN (Not a Number).

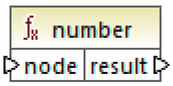


Languages

XQuery, XSLT 2.0, XSLT 3.0.

6.7.30.9 number

Returns the value of **node**, converted to an `xs:double`. The only types that can be converted to numbers are Booleans, numeric strings, and other numeric types. Non-numeric input values (such as a non-numeric string) result in NaN (Not a Number).



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

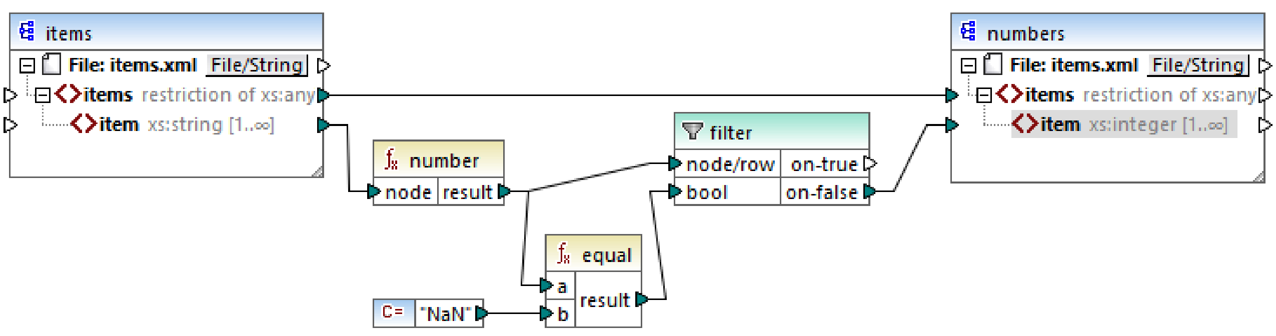
Name	Type	Description
node	`mf:atomic`	The input node.

Example

The following XML contains items of type `string`:

```
<items>
   <item>1</item>
   <item>2</item>
   <item>Jingle Bells</item>
</items>
```

The mapping illustrated below attempts to convert all these strings to numeric values and write them to a target XML file. Notice that the data type of **item** in the target XML component is `xs:integer` while the source **item** is of `xs:string` data type. If the conversion is not successful, the item must be skipped and not copied to the target file.



XSLT 2.0 mapping

To achieve the mapping goal, a filter was used. The **`equal`** function checks if the result of the conversion is "NaN". If this is false, this indicates a successful conversion, so the item is copied to the target. The output of the mapping is as follows:

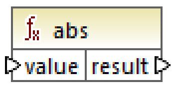
```
<items>
   <item>1</item>
   <item>2</item>
</items>
```

6.7.31 xpath2 | numeric functions

The numeric functions of the **xpath2** library include the **`abs`** and **`round-half-to-even`** functions.

6.7.31.1 abs

Returns the absolute value of the argument. For example, if the input argument is **-2** or **2**, the function returns **2**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

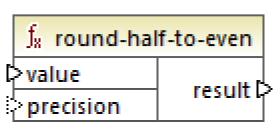
Parameters

Name	Type	Description
value	`xs:decimal`	The input value.

6.7.31.2 round-half-to-even

The **`round-half-to-even`** function rounds the supplied number (first argument) to the degree of precision (number of decimal places) supplied in the optional second argument. For example, if the first argument is **2.141567** and the second argument is **3**, then the first argument (the number) is rounded to three decimal places, so the result will be **2.142**. If no precision (second argument) is supplied, the number is rounded to zero decimal places, that is, to an integer.

The "even" in the name of the function refers to the rounding to an even number when a digit in the supplied number is midway between two values. For example, `round-half-to-even(3.475, 2)` would return **3.48**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:decimal`	Mandatory argument which provides the input value to be rounded.

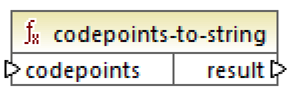
Name	Type	Description
precision	`xs:integer`	Optional argument which specifies the number of decimal places to round to. The default value is **0**.

6.7.32 xpath2 | string functions

The string functions of the **xpath2** library enable you to process strings (this includes comparing strings, converting strings to upper or lower case, extracting substrings from strings, and others).

6.7.32.1 codepoints-to-string

Creates a string from a sequence of Unicode code points. This function is the opposite of the `string-to-codepoints`1180 function.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
codepoints	`ZeroOrMore xs:integer`	This input must be connected to a sequence of items of integer type, where each integer specifies a Unicode code point.

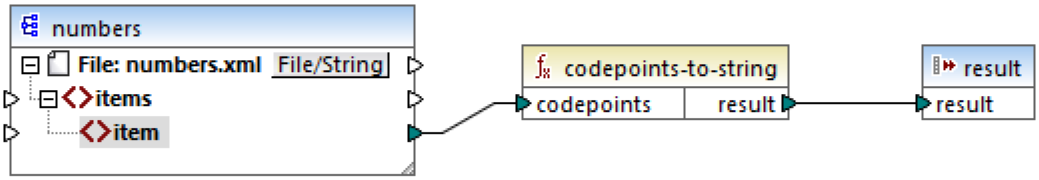
Example

The following XML contains multiple **item** elements that store each Unicode code point values.

```
<items>
   <item>77</item>
   <item>97</item>
   <item>112</item>
   <item>70</item>
   <item>111</item>
   <item>114</item>
   <item>99</item>
```

```
   <item>101</item>
</items>
```

The mapping illustrated below supplies the sequence of items as argument to the **codepoint-to-string** function.



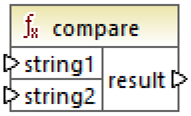
XSLT 2.0 mapping

The mapping output is **MapForce**.

6.7.32.2 compare

The **compare** function takes two strings as arguments and compares them for equality and alphabetically. If **string1** is alphabetically less than **string2** (for example the two string are "A" and "B"), then the function returns **-1**. If the two strings are equal (for example, "A" and "A"), the function returns **0**. If **string1** is greater than **string2** (for example, "B" and "A"), then the function returns **1**.

This variant of the function uses the default collation, which is Unicode. Another variant 1174 of this function exists where you can supply the collation as argument.



Languages

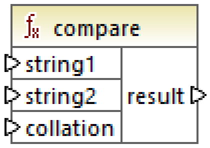
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string1	xs:string	The first input string.
string2	xs:string	The second input string.

6.7.32.3 compare

The **compare** function takes two strings as arguments and compares them for equality and alphabetically, using the collation supplied as argument. If **string1** is alphabetically less than **string2** (for example the two string are "A" and "B"), then the function returns **-1**. If the two strings are equal (for example, "A" and "A"), the function returns **0**. If **string1** is greater than **string2** (for example, "B" and "A"), then the function returns **1**.



Languages

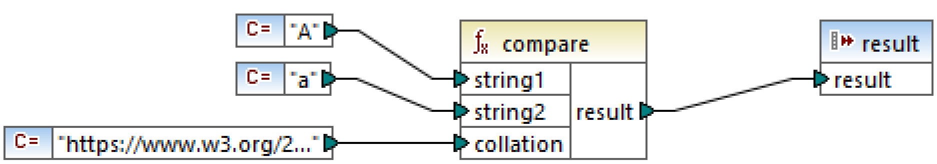
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string1	`xs:string`	The first input string.
string2	`xs:string`	The second input string.
collation	`xs:string`	Specifies the collation to use for string comparison. This input may originate from the output of the **default-collation** [1146] function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

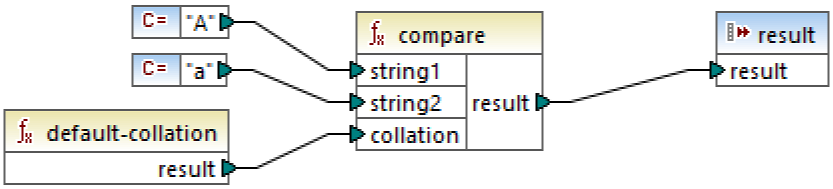
Example

The following mapping compares the strings "A" and "a" using the case insensitive collation `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`, which is supplied by a constant.



XSLT 2.0 Mapping

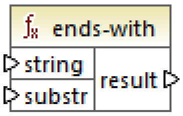
The result of the mapping above is **0**, meaning that both strings are treated as equal. However, if you replace the collation with the one provided by the **`default-collation`** function, the collation changes to the default Unicode code point collation, and the mapping result becomes **-1** ("A" is alphabetically less than "a").



6.7.32.4 ends-with

Returns **true** if **string** ends with **substr**; **false** otherwise. The returned value is of type `xs:boolean`.

This variant of the function uses the default collation, which is Unicode. Another variant1176 of this function exists where you can supply the collation as argument.



Languages

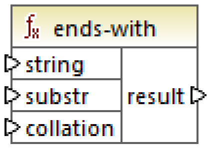
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	The input string (that is, the "haystack").
substr	`xs:string`	The substring (that is, the "needle").

6.7.32.5 ends-with

Returns **true** if **string** ends with **substr**; **false** otherwise. The returned value is of type `xs:boolean`.



Languages

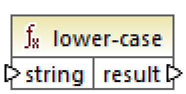
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	The input string (that is, the "haystack").
substr	`xs:string`	The substring (that is, the "needle").
collation	`xs:string`	Specifies the collation to use for string comparison. This input may originate from the output of the `default-collation`[1146] function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

6.7.32.6 lower-case

Returns the value of **string** after translating every character to its lower-case correspondent.



Languages

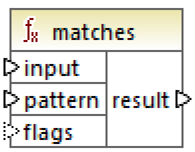
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	xs:string	The input value.

6.7.32.7 matches

The **matches** function tests whether a supplied string (the first argument) matches a regular expression (the second argument). The syntax of regular expressions must be that defined for the `pattern` facet of XML Schema. The function returns **true** if the string matches the regular expression, **false** otherwise.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

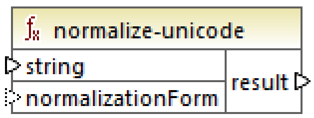
Parameters

Name	Type	Description
input	xs:string	The input string.
pattern	xs:string	The regular expression to match, see Regular Expressions (975).
flags	xs:string	Optional argument that influences the matching. This argument may supply any combination of the following flags: `i`, `m`, `s`, `x`. Multiple flags can be used, for example, `imx`. If no flag is used, the default values of all four flags are used. The four flags are as follows: `i` Use case-insensitive mode. The default is case-sensitive. `m` Use multi-line mode, in which the input string is considered to have multiple lines, each separated by a newline character (`x0a`). The meta characters **^** and **$** indicate the beginning and end of each line. The default is string mode, in which the string starts and ends with the meta characters **^** and **$**. `s` Use dot-all mode. The default is not-dot-all mode, in which the meta character **.** matches all characters except the newline character (`x0a`). In dot-all mode, the dot also matches the newline character.

Name	Type	Description
		x Ignore whitespace. By default, whitespace characters are not ignored.

6.7.32.8 normalize-unicode

Returns the value of **string** normalized according to the rules of the normalization form specified (the second argument). For more information about Unicode normalization, see §2.2 of https://www.w3.org/TR/charmod-norm/.



Languages

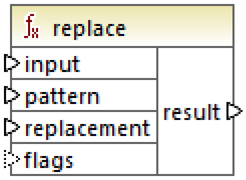
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	The string value to be normalized.
normalizationForm	`xs:string`	Optional argument which supplies the normalization form. The default is Unicode Normalization Form C (NFC). The normalization forms NFC, NFD, NFKC, and NFKD are supported.

6.7.32.9 replace

This function takes an input string, a regular expression, and a replacement string as arguments. It replaces all matches of the regular expression in the input string with the replacement string. If the regular expression matches two overlapping strings in the input string, only the first match is replaced.



Languages

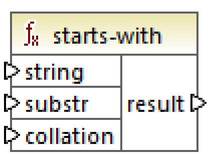
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
input	`xs:string`	The input string.
pattern	`xs:string`	The regular expression to match, see Regular Expressions [975].
replacement	`xs:string`	The replacement string.
flags	`xs:string`	Optional argument that influences the matching. This argument is used in the same way as the **flags** argument of the **`matches`** [1177] function.

6.7.32.10 starts-with

Returns **true** if **string** starts with **substr**; **false** otherwise. The returned value is of type `xs:boolean`. String comparison takes place according to the specified collation.



Languages

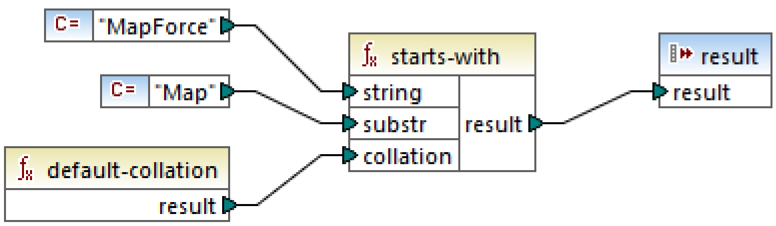
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	The input string (that is, the "haystack").
substr	`xs:string`	The substring (that is, the "needle").
collation	`xs:string`	Specifies the collation to use for string comparison. This input may originate from the output of the **default-collation** [1146] function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

Example

The following mapping returns the value **`true`**, because the input string "MapForce" begins with the substring "Map", assuming that the default Unicode collation is used.



6.7.32.11 string-to-codepoints

Returns the sequence of Unicode code points (integer values) that constitute the string supplied as argument. This function is the opposite of the **`codepoints-to-string`** [1172] function.

Languages

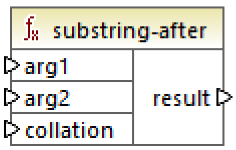
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
input	xs:string	The input string

6.7.32.12 substring-after

Returns the part of string **arg1** that occurs after the string **arg2**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

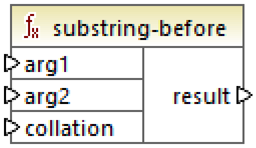
Name	Type	Description
arg1	xs:string	The input string (that is, the "haystack").
arg2	xs:string	The substring (that is, the "needle").
collation	xs:string	Specifies the collation to use for string comparison. This input may originate from the output of the **default-collation** [1146] function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

Example

If **arg1** is "MapForce", **arg2** is "Map", and **collation** is **default-collation** [1146], the function returns "Force".

6.7.32.13 substring-before

Returns the part of string **arg1** that occurs before the string **arg2**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

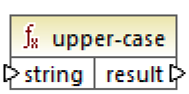
Name	Type	Description
arg1	`xs:string`	The input string (that is, the "haystack").
arg2	`xs:string`	The substring (that is, the "needle").
collation	`xs:string`	Specifies the collation to use for string comparison. This input may originate from the output of the **default-collation**[1146] function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

Example

If **arg1** is "MapForce", **arg2** is "Force", and **collation** is **default-collation**[1146], the function returns "Map".

6.7.32.14 upper-case

Returns the value of **string** after translating every character to its upper-case correspondent.



Languages
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

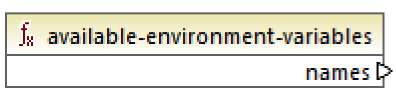
Name	Type	Description
string	`xs:string`	The input string.

6.7.33 xpath3 | external information functions

The external information functions of the **xpath3** library enable you to obtain information about the XSLT execution environment or retrieve data from external resources.

6.7.33.1 available-environment-variables

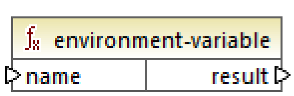
Returns a list of environment variable names that are suitable for passing to the `environment-variable` function, as a (possibly empty) sequence of strings.



Languages
XSLT 3.0.

6.7.33.2 environment-variable

Returns the value of a system environment variable, if it exists. The return type is `xs:string`.



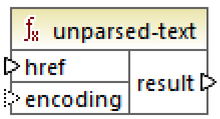
Languages
XSLT 3.0.

Parameters

Name	Type	Description
name	`xs:string`	The name of the environment variable.

6.7.33.3 unparsed-text

Reads an external resource (for example, a file) and returns a string representation of the resource.



Languages

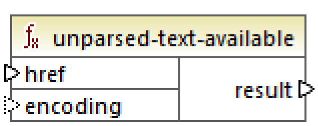
XSLT 3.0.

Parameters

Name	Type	Description
href	`xs:string`	A string in the form of a URI reference.
encoding	`xs:string`	Optional argument. Specifies the name of the encoding, for example "UTF-8", "UTF-16". If the encoding cannot be determined automatically, then UTF-8 is assumed.

6.7.33.4 unparsed-text-available

Determines whether a call to **`unparsed-text`** with particular arguments would succeed. The return type is `xs:boolean`.



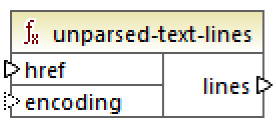
Languages
XSLT 3.0.

Parameters

Name	Type	Description
href	xs:string	A string in the form of a URI reference.
encoding	xs:string	Optional argument. Specifies the name of the encoding, for example "UTF-8", "UTF-16". If the encoding cannot be determined automatically, then UTF-8 is assumed.

6.7.33.5 unparsed-text-lines

Reads an external resource (for example, a file) and returns its contents as a sequence of strings, one for each line of text in the string representation of the resource.



Languages
XSLT 3.0.

Parameters

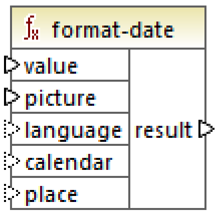
Name	Type	Description
href	xs:string	A string in the form of a URI reference.
encoding	xs:string	Optional argument. Specifies the name of the encoding, for example "UTF-8", "UTF-16". If the encoding cannot be determined automatically, then UTF-8 is assumed.

6.7.34 xpath3 | formatting functions

The formatting functions available of the **xpath3** library are used to format date, time and integer values.

6.7.34.1 format-date

Returns a string containing an `xs:date` value formatted for display.



Languages

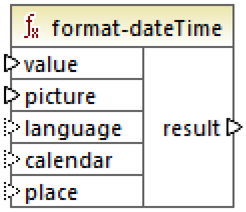
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:date`	The input `xs:date` value to be formatted. Mandatory parameter.
picture	`xs:string`	Mandatory parameter. See section 9.8.4.1 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
language	`xs:string`	Optional parameter. See section 9.8.4.8 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
calendar	`xs:string`	Same as above.
place	`xs:string`	Same as above.

6.7.34.2 format-dateTime

Returns a string containing an `xs:dateTime` value formatted for display.



Languages

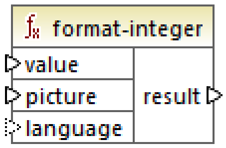
XSLT 3.0.

Parameters

Name	Type	Description
value	**xs:dateTime**	The input `xs:dateTime` value to be formatted.
picture	**xs:string**	Mandatory parameter. See section 9.8.4.1 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
language	**xs:string**	Optional parameter. See section 9.8.4.8 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
calendar	**xs:string**	Same as above.
place	**xs:string**	Same as above.

6.7.34.3 format-integer

Formats an integer according to a given picture string, using the conventions of a given natural language if specified.



Languages

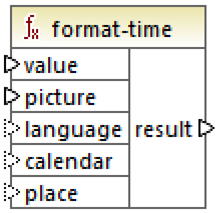
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:integer`	The input integer value to be formatted.
picture	`xs:string`	Mandatory parameter. See section 4.6.1 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
language	`xs:string`	Optional parameter. Specifies the natural language according to which the value should be formatted. If specified, this value must be either an empty string or any value that would be allowed for the `xml:lang` attribute according to the "Extensible Markup Language (XML) 1.0 W3C Recommendation (https://www.w3.org/TR/xml).

6.7.34.4 format-time

Returns a string containing an `xs:time` value formatted for display.



Languages

XSLT 3.0.

Parameters

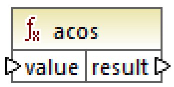
Name	Type	Description
value	**xs:time**	The input `xs:time` value to be formatted.
picture	**xs:string**	Mandatory parameter. See section 9.8.4.1 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
language	**xs:string**	Optional parameter. See section 9.8.4.8 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
calendar	**xs:string**	Same as above.
place	**xs:string**	Same as above.

6.7.35 xpath3 | math functions

The math functions of the **xpath3** library are used to perform trigonometric and other mathematical calculations.

6.7.35.1 acos

Returns the arc cosine of an angle, in the range of **0** through **pi**.



Languages

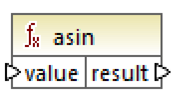
XSLT 3.0.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.35.2 asin

Returns the arc sine of an angle, in the range of **-pi/2** through **pi/2**.



Languages

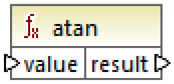
XSLT 3.0.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.35.3 atan

Returns the arc tangent of an angle, in the range of **-pi/2** through **pi/2**.



Languages

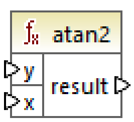
XSLT 3.0.

Parameters

Name	**Type**	**Description**
value	`xs:double`	The input value.

6.7.35.4 atan2

Returns the angle in radians subtended at the origin by the point on a plane with coordinates (x, y) and the positive x-axis.



Languages

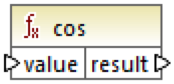
XSLT 3.0.

Parameters

Name	**Type**	**Description**
y	`xs:double`	The x coordinate.
x	`xs:double`	The y coordinate.

6.7.35.5 cos

Returns the trigonometric cosine of the angle given by value. The unit of value is radian.



Languages

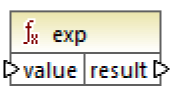
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.35.6 exp

Returns Euler's number *e* raised to the power of the value.



Languages

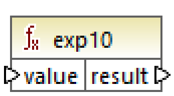
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.35.7 exp10

Returns 10 raised to the power of the value.



Languages

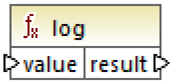
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.35.8 log

Returns the natural logarithm (base e) of a value.



Languages

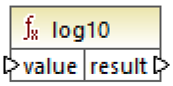
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.35.9 log10

Returns the decimal logarithm (base 10) of a value.



Languages

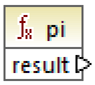
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

6.7.35.10 pi

Returns an approximation to the mathematical constant **pi**.

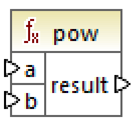


Languages

XSLT 3.0.

6.7.35.11 pow

Returns the value of **a** raised to the power of **b**.



Languages

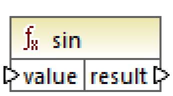
XSLT 3.0.

Parameters

Name	Type	Description
a	`xs:double`	The input value **a**.
b	`xs:double`	The input value **b**.

6.7.35.12 sin

Returns the trigonometric sine of the angle given by value. The unit of value is radian.



Languages

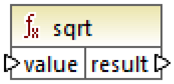
XSLT 3.0.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.35.13 sqrt

Returns the non-negative square root of the argument.



Languages

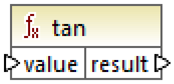
XSLT 3.0.

Parameters

Name	Type	Description
value	xs:double	The input value.

6.7.35.14 tan

Returns the trigonometric tangent of the angle given by value. The unit of value is radian.



Languages

XSLT 3.0.

Parameters

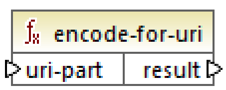
Name	Type	Description
value	xs:double	The input value.

6.7.36 xpath3 | URI functions

The URI functions in the **xpath3** library perform encoding, escaping, and conversion of values intended for use in URIs.

6.7.36.1 encode-for-uri

Encodes reserved characters in a string that is intended to be used in the path segment of a URI. For further information about this function, see section 6.2 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).



Languages

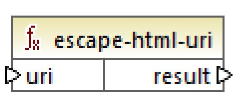
XSLT 3.0.

Parameters

Name	Type	Description
uri-part	`xs:string`	The input URI value to encode.

6.7.36.2 escape-html-uri

Escapes a URI in the same way that HTML user agents handle attribute values expected to contain URIs. For further information about this function, see section 6.4 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).



Languages

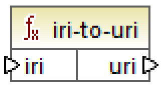
XSLT 3.0.

Parameters

Name	Type	Description
uri	`xs:string`	The input URI value to escape.

6.7.36.3 iri-to-uri

Converts a string containing an IRI (Internationalized Resource Identifier) into a URI (Uniform Resource Identifier). For further information about this function, see section 6.3 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).



Languages

XSLT 3.0.

Parameters

Name	Type	Description
iri	`xs:string`	The input IRI value.

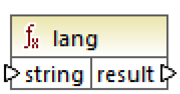
6.7.37 xslt | xpath functions

The functions in this sub-group are XPath 1.0 functions that retrieve information about mapping items (or nodes). Most of these functions take a node as argument and return information about that node. The `last` and `position` functions operate in the current mapping context[1277], which is determined by the connections on your mapping.

Note: Additional XPath 1.0 functions can be found in the **core** library.

6.7.37.1 lang

Returns **true** if the context node has an `xml:lang` attribute with a value that either matches exactly the **string** argument, or is a subset of it. Otherwise, the function returns **false**.



Languages

XSLT 1.0.

Parameters

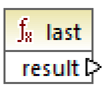
Name	Type	Description
string	`xs:string`	The language code to check, for example, "en".

Example

See the example given for the `lang`[1164] function of the **xpath2** library.

6.7.37.2 last

Returns the position number of the last node in the processed node list.



Languages

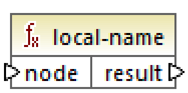
XSLT 1.0.

Example

See the example given for the `last`[1147] function of the **xpath2** library.

6.7.37.3 local-name

Returns the local part of the name of the node supplied as argument.



Languages

XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

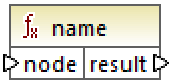
Name	Type	Description
node	`node()`	The input node.

Example

See the example given for the `local-name`[1166] function of the **`xpath2`** library.

6.7.37.4 name

Returns the name of the node supplied as argument.



Languages

XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

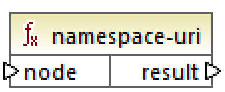
Name	Type	Description
node	`node()`	The input node.

Example

See the example given for the `local-name`[1166] function of the **`xpath2`** library.

6.7.37.5 namespace-uri

Returns the namespace URI of the node supplied as argument.



Languages

XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

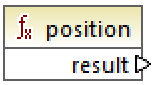
Name	Type	Description
node	`node()`	The input node.

Example

See the example given for the **`local-name`**[1166] function of the **`xpath2`** library.

6.7.37.6 position

Returns the position of the current node in the node set that is being processed.



Languages

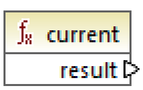
XSLT 1.0.

6.7.38 xslt | xslt functions

The functions in this group are miscellaneous XSLT 1.0 functions.

6.7.38.1 current

The **`current`** function takes no argument and returns the current node.

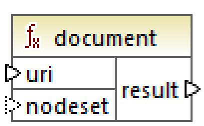


Languages

XSLT 1.0.

6.7.38.2 document

Accesses nodes from an external XML document. The result is output to a node in the output document.



Languages

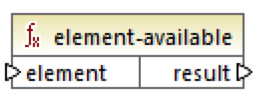
XSLT 1.0.

Parameters

Name	Type	Description
uri	`xs:string`	Mandatory. Specifies the path to the XML document. The XML document must be valid and parseable.
nodeset	`node()`	Optional. Specifies a node, the base URI of which is used to resolve the URI supplied as the first argument if it is relative.

6.7.38.3 element-available

The **`element-available`** function tests whether an element, entered as the only string argument of the function, is supported by the XSLT processor. The argument string is evaluated as a QName. Therefore, XSLT elements must have an `xsl:` prefix and XML Schema elements must have an `xs:` prefix—since these are the prefixes declared for these namespaces in the underlying XSLT that will be generated for the mapping. The function returns a Boolean.



Languages

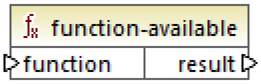
XSLT 1.0.

Parameters

Name	Type	Description
element	`xs:string`	The element name.

6.7.38.4 function-available

The **`function-available`** function is similar to the **`element-available`** function and tests whether the function name supplied as the function's argument is supported by the XSLT processor. The input string is evaluated as a QName. The function returns a Boolean.



Languages

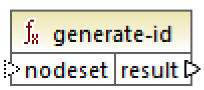
XSLT 1.0.

Parameters

Name	Type	Description
function	`xs:string`	The function name.

6.7.38.5 generate-id

The **`generate-id`** function generates a unique string that identifies the first node in the node set identified by the optional input argument. If no argument is supplied, the ID is generated on the context node. The result can be directed to any node in the output document.



Languages

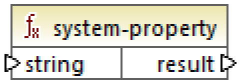
XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
nodeset	`node()`	Optional argument that supplies the input node.

6.7.38.6 system-property

The **`system-property`** function returns properties of the XSLT processor (the system). Three system properties, all in the XSLT namespace, are mandatory for XSLT processors. These are `xsl:version`, `xsl:vendor`, and `xsl:vendor-url`. The input string is evaluated as a QName and so must have the `xsl:` prefix, since this is the prefix associated with the XSLT namespace in the underlying XSLT stylesheet.



Languages

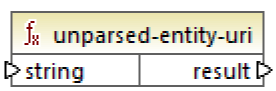
XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	Specifies the property name, which can be any of the following: `xsl:version`, `xsl:vendor`, `xsl:vendor-url`.

6.7.38.7 unparsed-entity-uri

If you are using a DTD, you can declare an unparsed entity in it. This unparsed entity (for example, an image) will have a URI that locates the unparsed entity. The input string of the function must match the name of the unparsed entity that has been declared in the DTD. The function then returns the URI of the unparsed entity, which can then be directed to a node in the output document, for example, to an **href** node.



Languages

XSLT 1.0.

Parameters

Name	Type	Description
string	`xs:string`	The name of the unparsed entity whose URI should be retrieved.

7 SOAP Web Services

With MapForce Enterprise edition you can generate Java or C# program code that implements SOAP Web services, from existing Web Services Description Language (WSDL) files. You can map data to WSDL operations as follows:

- From the input of the WSDL operation to any data sources supported by MapForce, including flat files, XML, XBRL, EDI, Microsoft Excel, and databases.
- From data sources supported by MapForce to the output of the WSDL operation.

MapForce supports WSDL 1.1 and WSDL 2.0 (for additional support information and limitations, see SOAP/WSDL Support Notes 1207 and Java SOAP Web Services Specifics 1220).

Note: As an alternative to developing SOAP Web services from existing WSDL files, you can also deploy MapForce mappings to MapForce Server, where you can run them through an API or command line call. Additionally, with FlowForce Server, you can further expose the mapping as a generic (not SOAP) Web service on the machine where FlowForce Server runs. Any mapping designed for the BUILT-IN 27 transformation language qualifies for deployment to MapForce Server or FlowForce Server. FlowForce Server is a dedicated server solution that is capable not only to convert MapForce mappings into generic Web services, but also run them as scheduled or on demand jobs. For more information, see Deploying Mappings to FlowForce Server 1381.

Prerequisites

To create a Web service with MapForce, the WSDL file of the Web service is required. Note that you can design WSDL files and test SOAP requests with XMLSpy, for example. Additionally, you need platform-specific software required to build, deploy, and run a SOAP Web service, for example:

Java

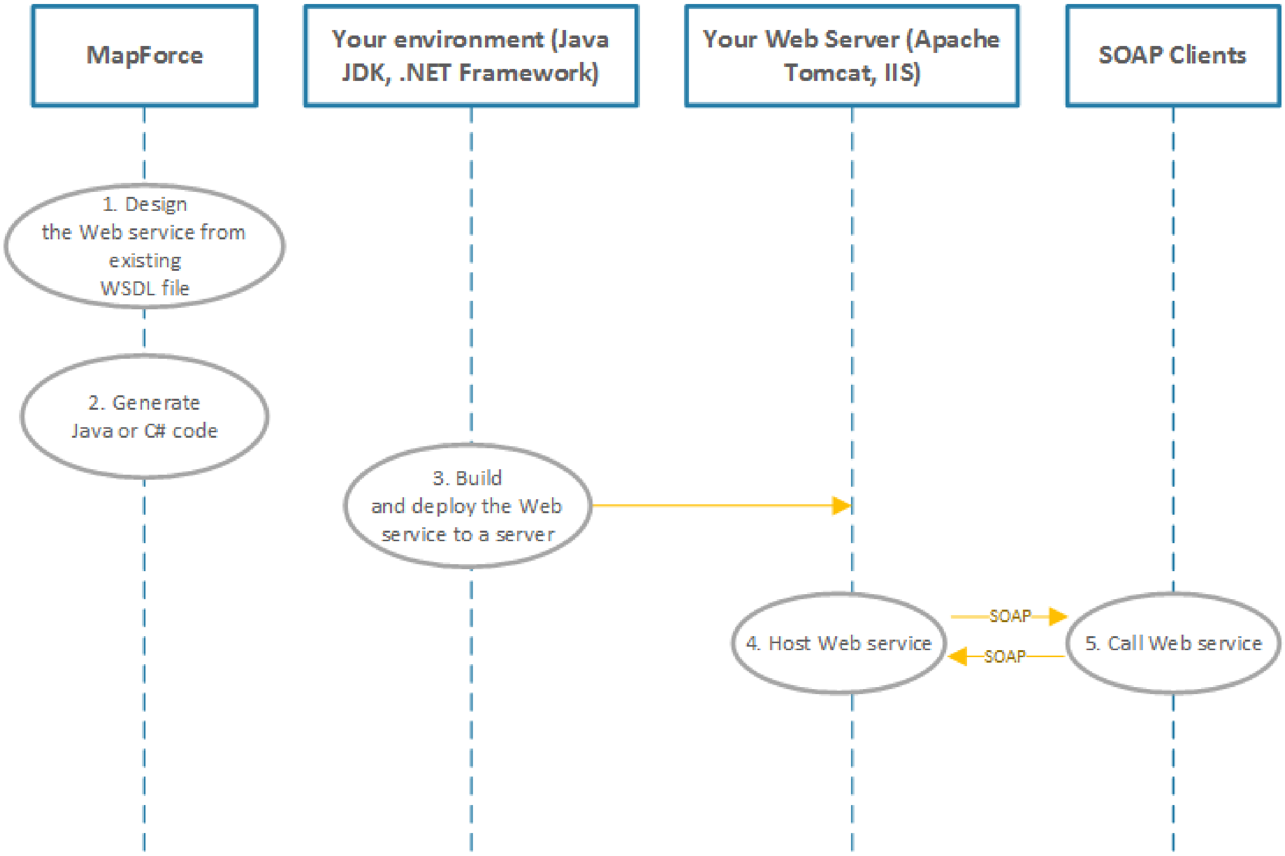
- Java SE JDK 8 or 11 (including OpenJDK)
- Apache Tomcat: https://tomcat.apache.org/
- Apache Axis2: https://axis.apache.org/axis2/java/core/, a SOAP framework running within Tomcat
- Apache Ant: https://ant.apache.org/

C#

- Microsoft Visual Studio 2013, 2015, 2017, 2019, 2022
- Microsoft Internet Information Services (IIS) version 5.0 or later.

How it works

The following diagram illustrates the entire process, starting from implementing a Web service from an existing WSDL file up to the stage when it can be called by SOAP client applications. Note that only the first two steps take place in MapForce. Steps 3 and 4 depend on the specifics of the third-party tools and environments used by your organization.



The process illustrated above works as follows:

Step #	Description
1. Design the Web Service from an existing WSDL file	Once you have the WSDL file, you can start a new MapForce Web Service Project. A MapForce Web Service Project has a predefined structure that enables you to quickly access a particular WSDL operation or Web service in it. When you start a MapForce Web Service Project, you can optionally instruct MapForce to generate mapping files automatically for all WSDL operations found in the WSDL file. As an alternative, you can select individually the WSDL Services and WSDL Endpoints to be included in the MapForce Web Service Project. In the graphical user interface of MapForce, WSDL operations correspond to mappings, and the operation's input and output parameters appear as components in the mapping. While working with mappings that include input or output of WSDL operations, you can preview the execution result as you would do for any other mapping. To do this, you need a sample SOAP request file (XML format) that conforms to the WSDL grammar of the Web service (such instance files can be generated with XMLSpy).

2. Generate Java or C# Code	When your Web Service Project is ready in MapForce, you can generate the C# or Java code either for individual mappings (for testing purposes), or for the entire project. In the latter case, MapForce creates a complete Web service server in the language of choice (C# or Java).
3. Build and deploy the Web service to a server	In this step, you compile the generated C# or Java code (outside MapForce) and deploy it to your custom Web server. Your Web server must be capable to host SOAP services and respond to SOAP client calls, see the "Prerequisites" section above.
4. Host Web service	The compiled Web service can be hosted by the server of your choice, for example: • Apache Tomcat • Internet Information Services (IIS)
5. Call Web service	The clients of the Web service include any application that is capable of generating SOAP calls and processing the response, including XMLSpy or MapForce itself.

For more information about step 1, see Creating SOAP Web Service Projects 1210. For all other steps, see Example: Generating SOAP Web Services (C#) 1223 and Example: Generating SOAP Web Services (Java) 1237

7.1 SOAP/WSDL Support Notes

The following table summarizes the WSDL support details in MapForce.

WSDL support	Version 1.1, W3C Note from http://www.w3.org/TR/wsdl Version 2.0, W3C Recommendation from http://www.w3.org/TR/wsdl20/
WSDL type system	XML Schema 2001
SOAP support	Version 1.1: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ Version 1.2: http://www.w3.org/TR/soap12-part0/
Protocols	SOAP over HTTP (HTTP POST, HTTP GET protocols are not supported).
C#	The SOAPAction must be different for each operation in C#.
Bindings	Multiple operations with same name are currently not supported (Section 2.5 of the WSDL 1.1 specification).
style/use	• Document/literal: supported. • RPC/literal: supported in C# • RPC/encoded: limited support • One style/use per Web service (Java), or operation (C#) is currently supported.
SOAP headers	Depends on underlying platform.
SOAP encodingStyle	If use="encoded", encoding style **"http://schemas.xmlsoap.org/soap/encoding/"** for complete `soap:Body` is assumed. There is no support for other encoding styles. The `encodingStyle` attribute is ignored in messages (Section 4.1.1 of the SOAP 1.1 specification).
References	• References to external resources are currently not supported (Section 5.4.1 of the SOAP 1.1 specification). • References to independent elements are supported.
SOAP-ENC:Array	Linear access is supported. Partial arrays and sparse arrays are currently not supported.
Custom SOAP enhancements	Not supported.
Default or fixed values in schemas	Not supported.
Non SOAP message validation	Not validated; passed on to underlying framework.
Namespaces	Non namespace entries are invalid WSDL, and are therefore not supported (WSDL and XML 1.0).

WSDL 1.1

portType
A <portType> element defines a Web service interface, namely:

- the **operations** that can be performed.
- the **messages** involved in each operation as inputs and outputs.

types
The <types> element defines the datatypes that are used by the Web service. MapForce supports XML Schemas in WSDL files, as this is the most common type system for WSDL files. MapForce displays these elements (datatypes) as items in a (message) component, allowing you to map them to other item/constructs directly.

message
The <message> element defines the **parts** of each message and the **data elements** of an operation's input and output parameters. These are the messages exchanged by the client and server. There are three types of messages: Input, Output and Fault. In MapForce, each **message** is a **component** from or to which you can map other items. Messages can consist of one or more message parts.

When using the document / literal combination in MapForce, it is necessary that the **message / part element** refer to a global element as opposed to a **type**. For example, in the following code, the `element` attribute refers to a global element defined in a schema (`ns2:Vendor`):

```
<message name="processRequest">
      <part name="inputData" element="ns2:Vendor"/>
</message>
```

Whereas the following code references a type in the schema:

```
<message name="processRequest">
      <part name="inputData" type="ns2:VendorType"/>
</message>
```

operation
Operations use messages as input and output parameters. An operation can have:

- one Input message
- zero or more Output messages
- zero or more Fault messages

Input messages can only be used as source components. Output and Fault messages can only be used as target components.

WSDL 2.0

WSDL 2.0 is substantially different from WSDL 1.1, the main differences being:

- PortTypes have been renamed to interfaces.
- Messages and parts are now defined using the XML Schema type system in the **types** element.
- Ports have been renamed to endpoints.
- WSDL 2.0 operation inputs and outputs are defined by the XML schema.

In MapForce, the Component Settings dialog box of a WSDL component displays "Endpoint" for both WSDL 1.1 Ports and WSDL 2.0 endpoints.

7.2 Creating SOAP Web Service Projects

This example shows you how to create a SOAP Web service in MapForce. The mapping project created in this example is also available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Query Person database.mfp**. The Web service in this example has the following goals:

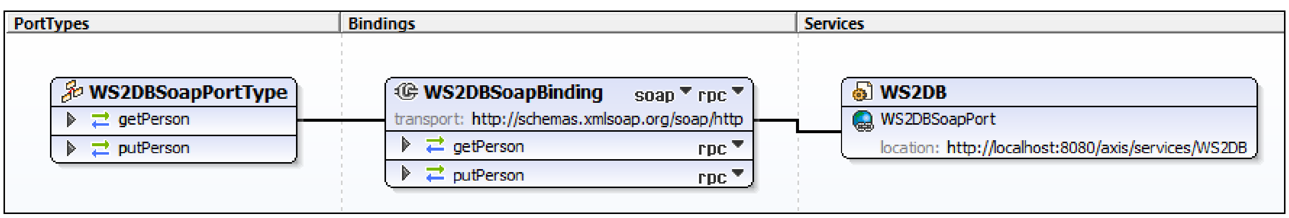
- Retrieve from a Microsoft Access database on a server a set of person records that satisfy some search criteria, through a SOAP Request. (The query is entered at runtime on the client, and is then sent to the server. The server then returns an XML response with the search results).
- Insert new records into the Microsoft Access database, also through SOAP requests. (The person details are provided in the SOAP Request. On success, the server returns a confirmation that the person details have been successfully added to the database.)

Note the following:

- The process of creating the Web service in MapForce does not depend on the target programming language; it is identical when generating Java, or C# Web services. The differences arise only when you compile and deploy the Web service on the web server, as further described in this documentation.
- In order to generate sample SOAP requests required to preview the mapping output, this example uses XMLSpy Enterprise Edition. The source WSDL file of this MapForce project was also created with XMLSpy (and can be validated with it). Designing Web services in MapForce is also possible without XMLSpy; however, in this case you will need a SOAP client capable to generate SOAP requests from WSDL files.

Creating a Web service project

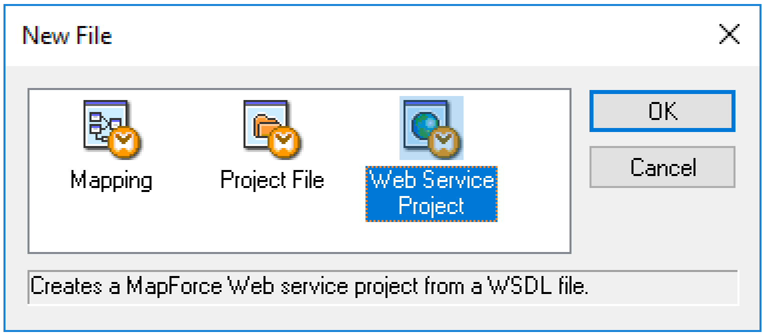
As stated before, a WSDL file is required to create a SOAP Web service with MapForce. In this example, we will use a WSDL file that was created with XMLSpy Enterprise edition. The image below illustrates the WSDL structure as it is displayed in the WSDL view of XMLSpy. The important thing to notice is that it defines two operations (`getPerson` and `putPerson`) that correspond to the goals outlined above.



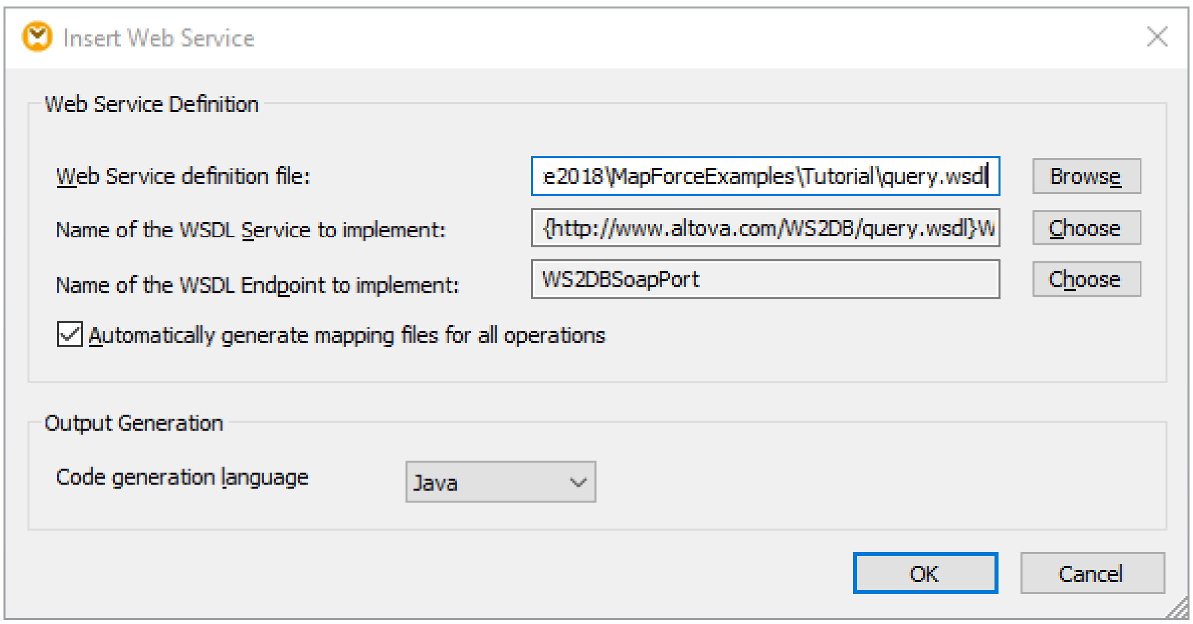
Note: The WSDL file must be valid before you can create a MapForce Web Service project from it.

To create a Web service project from this web service, follow the steps below:

1. On the **File** menu, click **New**, and select "Web Service Project".

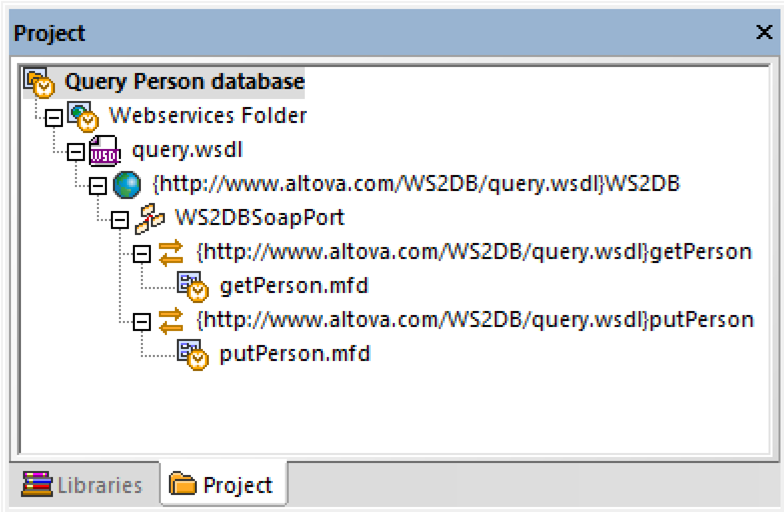


2. Browse for the **query.wsdl** file available in the folder **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** . After you select the WSDL file, MapForce automatically fills in the remaining fields.



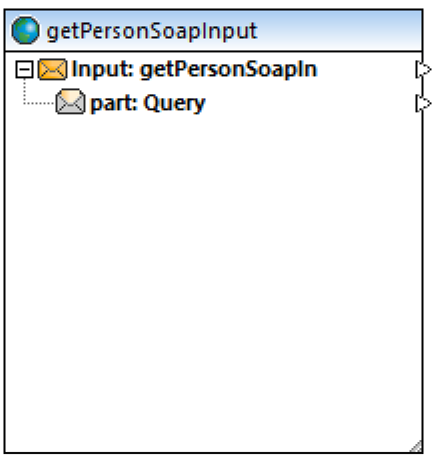
3. Click OK.
4. When prompted, enter the name of the new WSDL project, and click **Save**.

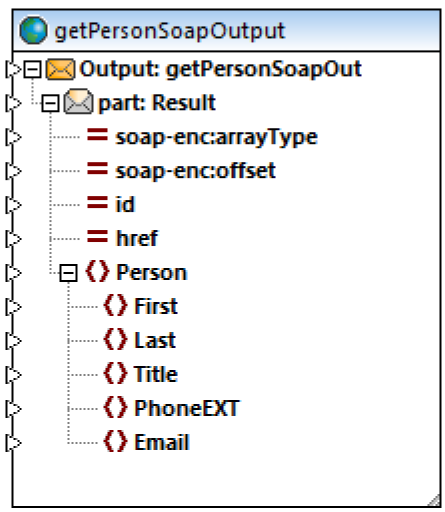
The Project window shows the project and WSDL name, as well as each of the operations defined in the WSDL file. The two operations are **getPerson** and **putPerson**.



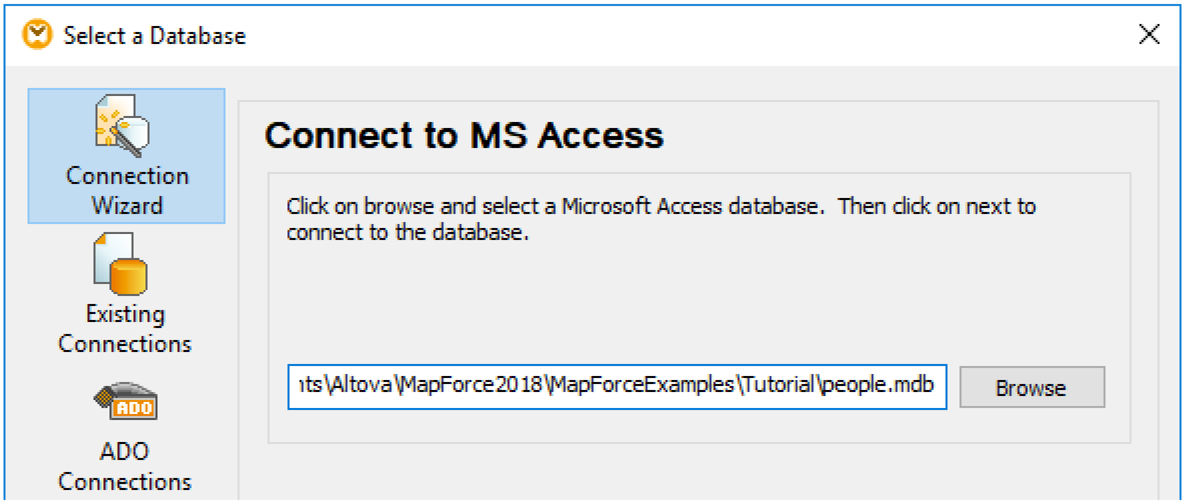
Designing the "getPerson" operation

Double-click the **getPerson.mfd** file in the Project window to load its contents in the main pane. The **getPersonSoapIn** component contains the query (item) which will be used to query the database through the Web service. The **getPersonSoapOut** component contains the `Person` structure defined in the WSDL file.

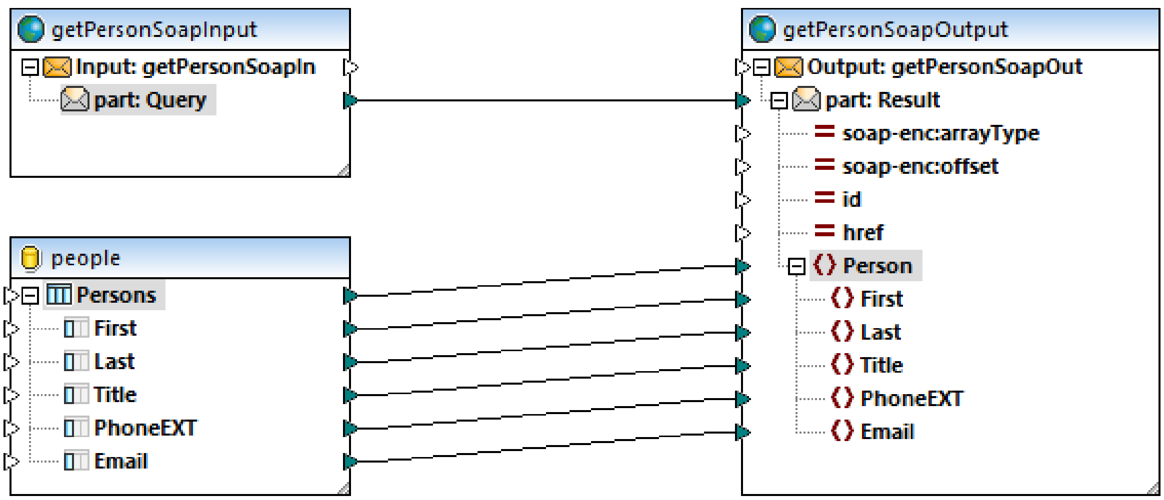




On the **Insert** menu, click **Database**, and add to the mapping area the following Access database: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\people.mdb**. The database contains only one table, `Persons`, whose fields match those in the `getPersonSoapOut` component. For more information about adding databases as mapping components, see Databases and MapForce 161.



Draw connections between components as shown below:

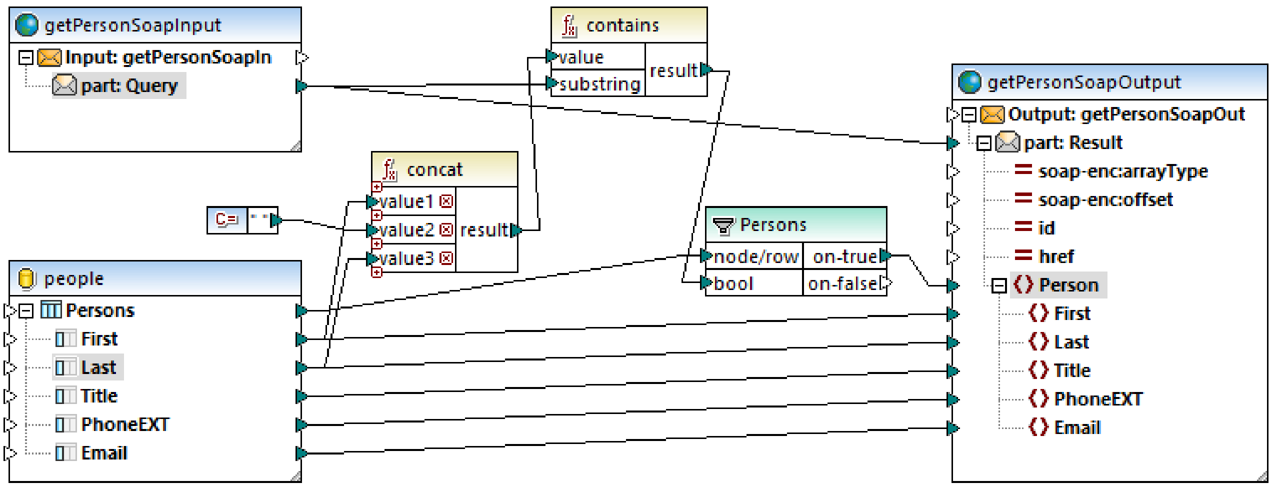


The mapping design above is not yet complete but clearly illustrates what will be happening at runtime. The connection between **part:Query** and **part:Result** represents the Web service call. The **part:Query** item of the `getPersonSoapIn` component is the query placeholder (it is where the query string is entered in the SOAP client). The connections between the database table and the **Person** item of the **getPersonSoapOutput** component extract values from the corresponding database fields and pass them to the corresponding target items. You can already quickly preview the output as follows:

1. Right-click the **getPersonSoapInput** component, and select **Properties** from the context menu.
2. Click **Browse** and select the following example request: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\getPersonRequest.xml**.
3. Click **OK**.
4. Click the **Output** tab.

At this stage, the mapping extracts all the records from the database table, which is not very useful. Therefore, we need to add a filter and extract only items that match the request string.

1. Add the **concat** function, by dragging it from the Libraries window[33]. This function will be used to concatenate the `First` and `Last` names of each person.
2. On the **Insert** menu, click **Constant**, and add a constant which contains one space character. The constant will supply the space character between the first and the last name extracted from the database.
3. Add the **contains** function, by dragging it from the Libraries window.
4. Right-click the connection between the source **Persons** table and **Person** target node, and select **Insert Filter: Nodes/Rows** from the context menu.
5. Draw connections between components as shown below.

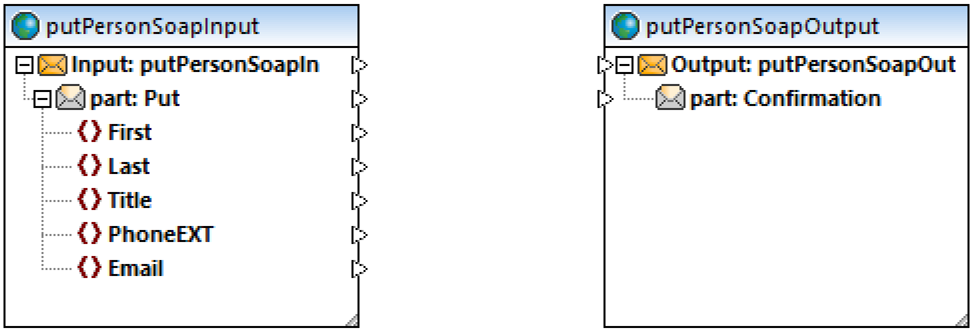


"getPerson" mapping

As illustrated above, the **contains** function returns **true** when the query string matches the full or partial name of any person in the database (where "name" is provided by the **concat** function, and it consists of the first name, followed by a space, followed by the last name). When there is a match, the details of the matching person are included in the response message. Therefore, in this example, the search value "Ro" would return all the following records: "Martin **Ro**pe", "**Ro**nald Superstring", "**Ro**bert Darkmatter", and "**Ro**ger Gravity".

Designing the "putPerson" operation

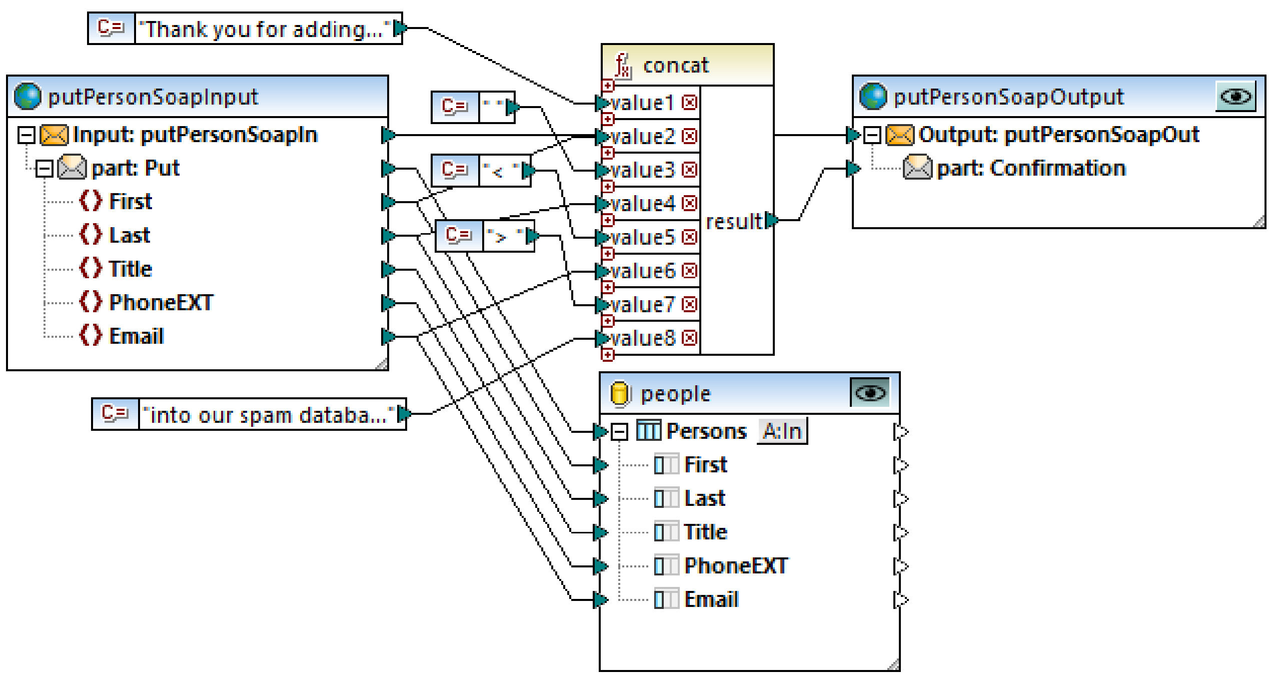
Double-click the **putPerson.mfd** file in the Project window to load its contents in the main pane. The **putPersonSoapInput** component supplies the request structure (items that will be inserted into the database through the Web service). The **putPersonSoapOutput** component represents the result of the Web service call. It contains a confirmation part to display the outcome of the call to the Web service caller.



Follow the same instructions as above to add the following to the mapping:

- The **people.mdb** Access database (this is where data supplied by the Web service caller will be inserted)
- The **concat** function and five constants (these will be used to build the confirmation string shown to the caller of the Web service)

Draw the connections as shown below:



putPerson mapping

In the mapping above, the connection between **part: Put** and the database will cause the database to be updated at runtime. On the target side, the **part: Confirmation** response is obtained by concatenating various fields from the request, as well as some text supplied by constants. For more information about mappings which update databases, see Mapping Data to Databases 282.

To preview the output of the mapping:

1. Right-click the **putPersonSoapInput** component.
2. Click **Browse** and select the following example request: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\putPersonRequest.xml**.
3. Click the **Output** tab.

When you preview the mapping, MapForce will generate the INSERT query required to update the database and display it in the Output window, but will not perform the actual update. To run the query against the database, select the menu command **Output | Run SQL-Script**.

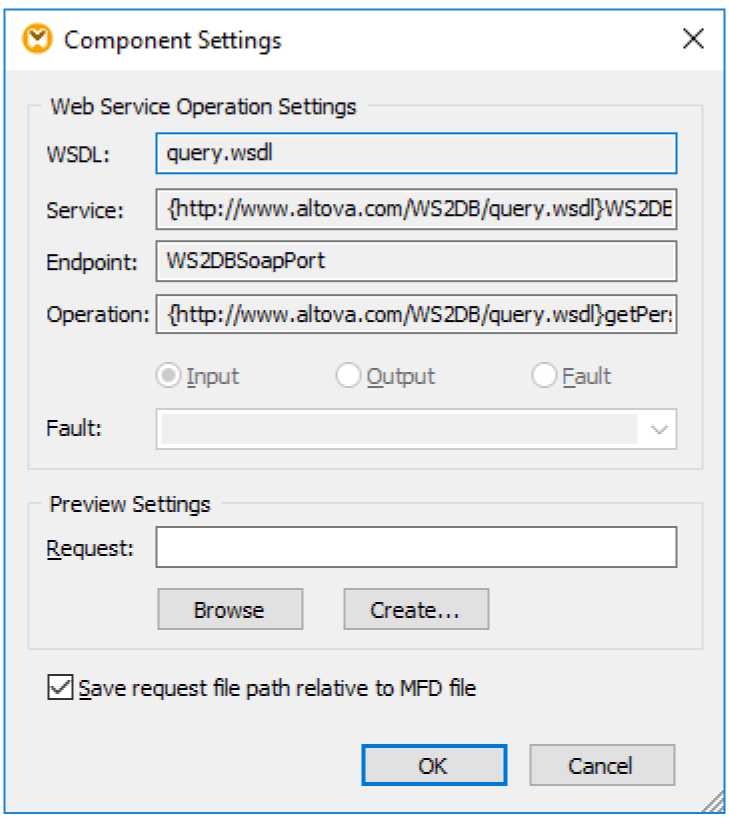
```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
Data Source=C:\Users\altova\Documents\Altova\MapForce2018\MapForceExamples\Tutorial\people.mdb;Provider=Microsoft.Jet.OLEDB.4.0
*/

INSERT INTO [Persons] ([First], [Last], [Title], [PhoneEXT], [Email]) VALUES ('John', 'Doe', 'Mr', '123', 'john.doe@example.org')

```

By contrast, when the compiled Web service is deployed to a server and called by a client, the database will be updated with immediate effect. See also Executing Mappings Which Modify Databases 274.

Previewing the mapping output

Mappings created from WSDL files can be previewed in MapForce before they are deployed to an actual Web server. To preview such mappings, a sample SOAP request (XML file) is necessary which conforms to the WSDL file. You can create a sample request for each operation (mapping) inside your Web service project either with XMLSpy or another SOAP client of your choice. If you use XMLSpy, run the menu command **SOAP | Create New SOAP Request** to generate the request. If XMLSpy is installed alongside MapForce, you can generate a sample request directly from MapForce, for example:

1. Right-click the **getPersonSoapInput** component on the mapping, and select **Properties** from the context menu.



2. Click **Create**.
3. When prompted, save the XML file to a location on the disk.

The request file is generated with some dummy input values (for example, "String") instead of input parameters. Before using the request file, make sure to replace the dummy values with the actual request values that you want to use. For this project, two sample requests are available:

1. **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\getPersonRequest.xml.** Use this file to preview the output of the **getPerson** mapping. By default, this request file supplies the value "Ro" as input to the Web service. This will retrieve from the database all people records where the last or first name contains the text "Ro".
2. **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\putPersonRequest.xml.** Use this file to preview the output of the **putPerson** mapping. By default, this request file supplies the details of a fictitious person as input to the Web service.

To obtain a different output while previewing the two mappings, therefore, edit the files above so that they contain the input values you wish to test.

Next steps

You are now ready to generate code that implements the Web service (either Java or C#), compile it, and deploy it to a server, as further described in the following examples:

- Example: Generating SOAP Web Services (C#) 1223
- Example: Generating SOAP Web Services (Java) 1237

7.3 Defining SOAP Web Service Faults

A WSDL file can contain a `fault` element for an operation and a message attribute that contains the fault message. Accordingly, when designing a SOAP Web service in MapForce, you can add a "Fault" component to your mapping. This will cause the Web service to throw an exception if a certain condition is satisfied.

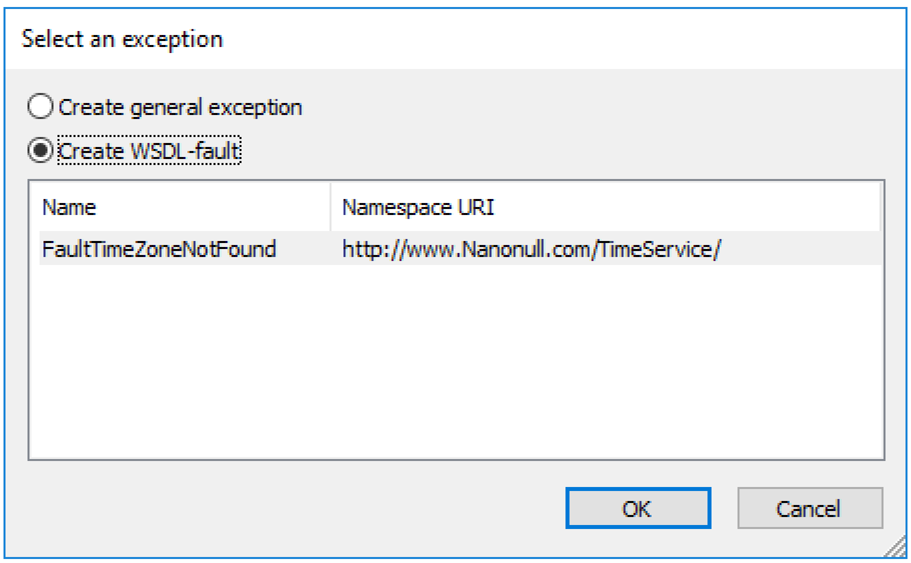
> To insert a Fault component in a mapping, a `fault` element must be present in the WSDL file.

You can define the failure condition inside the mapping, and you can also set the text of the error message. When the condition is satisfied, the Web service will fail as follows:

- In a live environment (at execution time, after the Web service is deployed to a server), the Web service will fail and return the custom-defined message to the caller.
- In MapForce (at design time), the mapping execution stops, and you can view the message in the **Message** window.

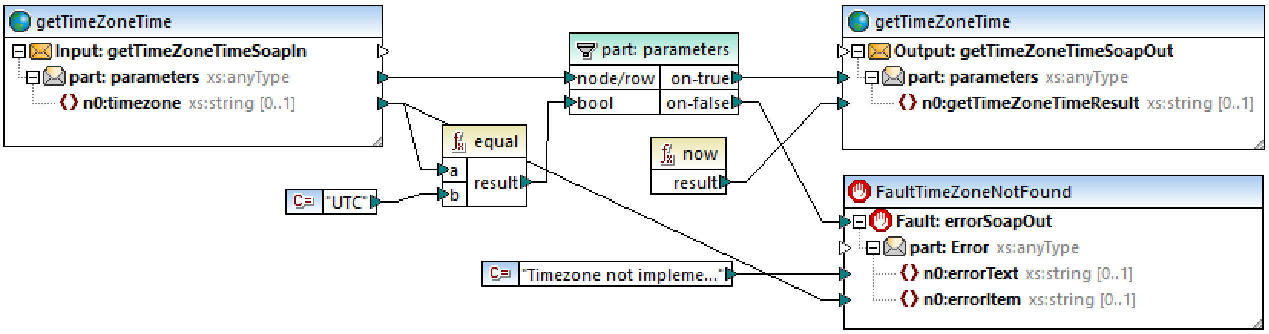
To add a fault component to a MapForce mapping:

1. On the **Insert** menu, click **Exception**. (Alternatively, click the **Exception** toolbar button.)
2. Select **Create WSDL-fault**.



Note: If the **Create WSDL-fault** option is disabled and you see the message "No faults for this operation", this means that no faults exist in the WSDL file, and, consequently, none can be added in MapForce.

For an example of a mapping which contains a Web service fault, open the following sample MapForce project: **<Documents>\Altova\MapForce2023\MapForceExamples\TimeService\TimeService.mfp**. Double-click the mapping (operation) **getTimeZoneTime.mfd**.



In the mapping above, the exception should be triggered when **n0:timezone** is not equal to UTC.

- The **equal** function checks to see if timezone equals UTC, with the bool result being passed on to the filter.
- If the condition is false, i.e. something other than UTC, the **on-false** parameter of the filter activates the **Fault:errorSoapOut** exception and the mapping process is halted. (Note that you can also connect the exception to the on-true parameter, if that is what you need.)
- Two sets of error text are supplied by the SoapFault message.

It is very important to note the filter placement in the example:

- Both parameters of the filter component, **on-true** and **on-false**, must be mapped. One of them needs to be mapped to the fault component, and the other, to the target component that receives the filtered source data. Otherwise, the fault component will never be triggered.
- The exception and target components must be directly connected to the filter component. Functions or other components may not be placed between the filter and either the exception, or target components.

7.4 Java SOAP Web Services Specifics

MapForce generates all necessary code and scripts needed to create a SOAP Web service. However, for the Web service to be available to consumers, the generated code must be built and deployed to the Apache Tomcat server with Axis2 installed. Additionally, Apache Ant is required to build the Java code.

Generating and building Java code

1. Open the Web service project from which you want to generate code. See the previous section, Creating Web Service Projects from WSDL files 1210, for an example of how to create such a project.
2. On the **Project** menu, click **Generate code in | Java**, and select the target directory. When code generation completes, several folders and files are created in the target directory, including a **com** directory which contains Altova generic classes, as well as the actual classes of the Web service project.
3. Build the generated Java code (by supplying to Apache Ant the **build.xml** file generated by MapForce). As a result, an Axis Archive File (*.aar) file is created, which you can then deploy to Axis2.

Note for Eclipse users: when you open the generated project with Eclipse, you may see an error like "The import org.apache cannot be resolved". In this case, make sure that the Axis2 libraries are added to the Java build path. To add the Axis2 libraries to the Java build path in Eclipse 4.4.2, do the following:

1. Right-click the project in the Package Explorer, and select **Properties**.
2. Click **Java Build Path**.
3. On the **Libraries** tab, click **Add External JARs**, and add the Axis2 libraries from the <AXIS2_HOME>\lib folder.

Deploying the Web service

To deploy the Web service, do one of the following:

- Open the "Upload Services" Web administration page of Axis2 and upload the .aar file created in the previous step
- Do a manual upload. For example, if your Tomcat server was installed to the folder <TOMCAT_HOME>, you can manually copy the .aar file to <TOMCAT_HOME>\webapps\axis2\WEB-INF\services.

For a step-by-step example, see Example: Generating SOAP Web Services (Java) 1237.

Undeployment

Delete the *.aar file from the <TOMCAT_HOME>\webapps\axis2\WEB-INF\services folder.

Axis2 limitations

Axis2 support for **RPC/encoded** is limited. MapForce can, however, generate RPC/encoded Web services (both SOAP 1.1 and SOAP 1.2). The limitation is that the original WSDL is not retrieved from the Web server.

This means that, for example, http://127.0.0.1/axis2/services/WS2DB?wsdl would **not** return a usable .wsdl file.

For **document/literal** Web services, the URL above will provide a usable and correct .wsdl file. It will differ from the original, however: comments will be stripped out, and namespaces will be changed. It will, however, still have the same semantics as the original .wsdl file with which the service was generated.

Although Axis2 does not support **RPC/encoded**, it is able to generate WSDL from deployed Java code (compiled code), and thus MapForce-generated code can process **RPC/encoded** messages; Axis2 is just used for transport.

Known issue: namespaces in the SOAP response message

The code generated by MapForce instantiates the **`javax.xml.transform.TransformerFactory`** class. When the class implementation is loaded, it might be read from the **`javax.xml.transform.TransformerFactory`** system property (for details, refer to the Java documentation of this class).

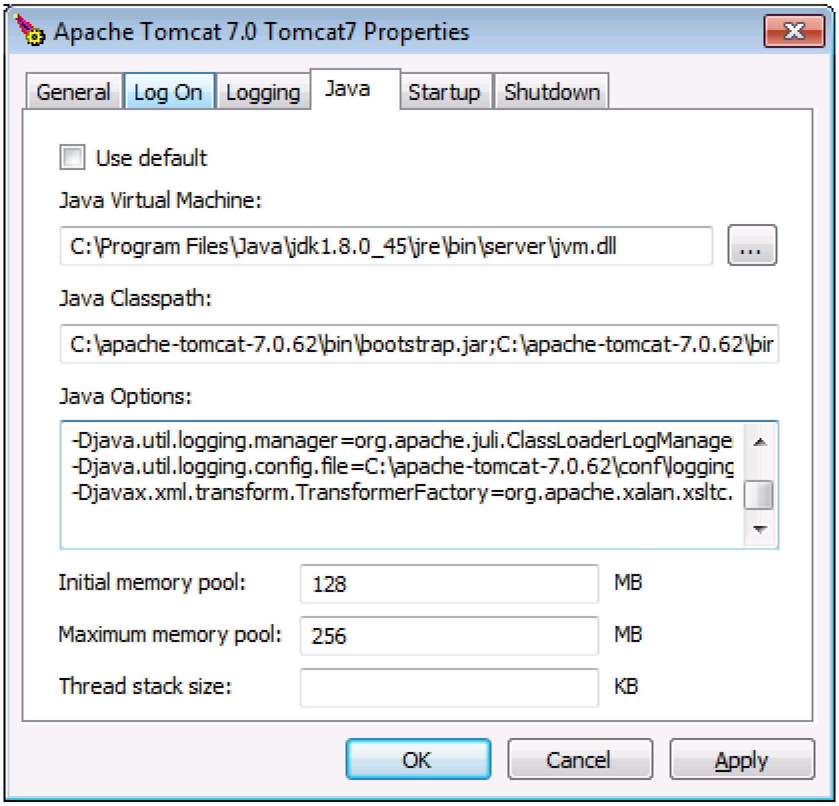
If the system property **`javax.xml.transform.TransformerFactory`** in your Java environment defines the implementation **`org.apache.xalan.transformer.TransformerIdentityImpl`**, the SOAP response message may be generated with incorrect namespaces.

To prevent this issue, it is recommended to use the **`org.apache.xalan.xsltc.trax.TransformerFactoryImpl`** implementation. You can do this by adding the following parameter to the Java Virtual Machine (JVM):

```
-Djavax.xml.transform.TransformerFactory=org.apache.xalan.xsltc.trax.TransformerFactoryImpl
```

The instructions for adding the parameter to the JVM depend on the operating system and server you are using. The following instructions are applicable to Apache Tomcat 7.0 configured to run as a service application on Windows 7:

1. Run **Tomcat7w.exe** (this file is located in the **\bin** subfolder of the Tomcat distribution root folder, `CATALINA_HOME`).
2. Click the **Java** tab.
3. At the end of "Java Options" box, enter: `-Djavax.xml.transform.TransformerFactory=org.apache.xalan.xsltc.trax.TransformerFactoryImpl`


Apache Tomcat 7.0 Tomcat7 Properties
General
Log On
Logging
Java
Startup
Shutdown
Use default
Java Virtual Machine:
C:\Program Files\Java\jdk1.8.0_45\jre\bin\server\jvm.dll
Java Classpath:
C:\apache-tomcat-7.0.62\bin\bootstrap.jar;C:\apache-tomcat-7.0.62\bir
Java Options:
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManage
-Djava.util.logging.config.file=C:\apache-tomcat-7.0.62\conf\logging
-Djavax.xml.transform.TransformerFactory=org.apache.xalan.xsltc.
Initial memory pool:
128
MB
Maximum memory pool:
256
MB
Thread stack size:
KB
OK
Cancel
Apply

7.5 Example: Generating SOAP Web Services (C#)

This example illustrates how to generate a SOAP Web service with MapForce. You will generate C# program code from a sample MapForce project available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Query Person database.mfp**. This project implements a SOAP Web service that exposes two operations: `getPerson` and `putPerson`. This Web service communicates in the background with a Microsoft Access database to retrieve or add person details, respectively, through the operations above. For an example of how to create such projects in MapForce, see Creating Web Service Projects from WSDL Files 1210.

This example uses Visual Studio 2015 to compile the generated C# code. The solution will be deployed to Internet Information Services (IIS) 10 with ASP.NET enabled.

For simplicity, this example uses a basic Web server configuration and should not be taken as prescriptive. In a production environment, you may need to use a different approach as demanded by the specifics of your Web server, the security policies in your organization, and other factors.

To call the SOAP Web service, this example makes use of XMLSpy Enterprise Edition. If you do not have XMLSpy, you will need a SOAP client to test the Web service, or, alternatively, write program code that can call the Web service.

Note: If you are using IIS 7.x, you may first need to install the Windows feature "IIS Metabase and IIS 6 configuration compatibility".

Step 1: Prepare the query.wsdl file

This example Web service originates from the following WSDL file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\query.wsdl**. By default, the **query.wsdl** file defines the service details for both C# and Java in two separate sections, one of which is commented out. Therefore, before generating C# code, locate the following section in the **query.wsdl** file and make sure that it is not commented out:

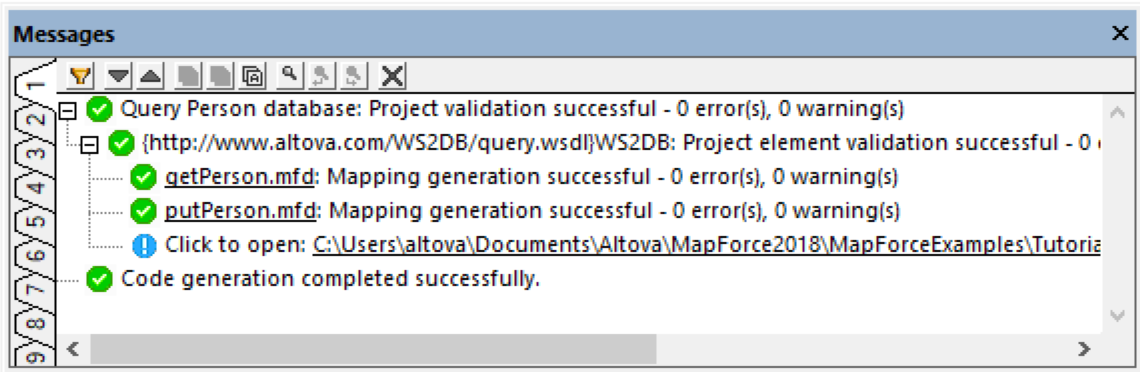
```
<service name="WS2DB">
   <port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
      <soap:address location="http://localhost/WS2DB/WS2DB.asmx"/>
   </port>
</service>
```

Also, make sure that the following section (only applicable to Java) is commented out:

```
<!--<service name="WS2DB">
   <port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
      <soap:address location="http://localhost:8080/axis/services/WS2DB"/>
   </port>
</service>-->
```

Step 2: Generate C# code and build it

Run MapForce and open the "Query Person database.mfp" project from the folder **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**. Right-click the project in the Project window, and select **Generate code in | C#**. MapForce generates program code and displays the outcome in the Messages window:



By default, code will be generated in a subdirectory called "output", relative to the project's directory. For the scope of this tutorial, we will leave the default settings as is. However, if you want to change the output directory, see Setting the Code Generation Settings[118] .

Run Visual Studio and open the generated solution.

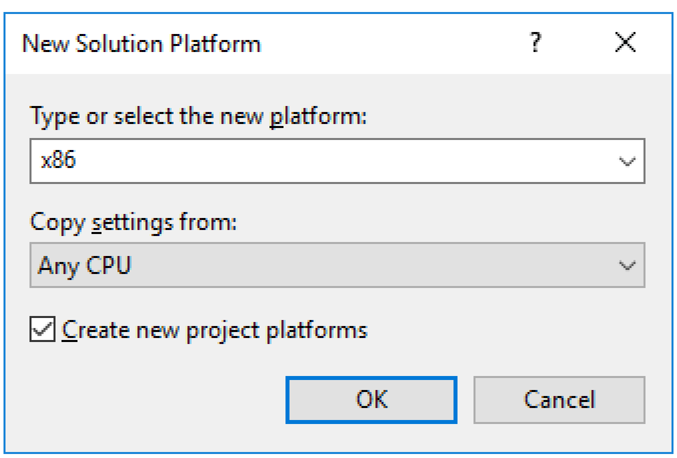
> You may need to run Visual Studio as administrator. Also, this example solution must be compiled as a 32-bit application, as shown below.

You can build the solution as follows:

- On the **Build** menu, click **Build Solution (Ctrl+Shift+B)**.

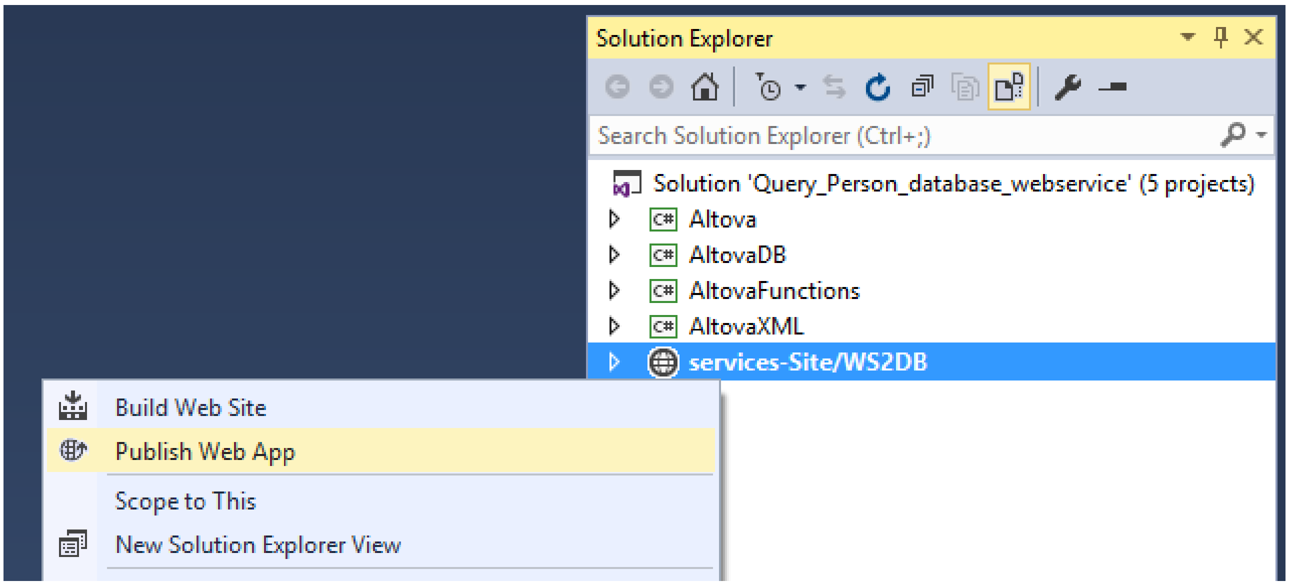
Note that this example queries a Microsoft Access database using the ADO provider for Access, which only works with 32-bit applications. Therefore, the solution must be compiled as a 32-bit application, as follows:

- On the **Build** menu, click **Configuration Manager**, create a new solution platform for x86, and build again.

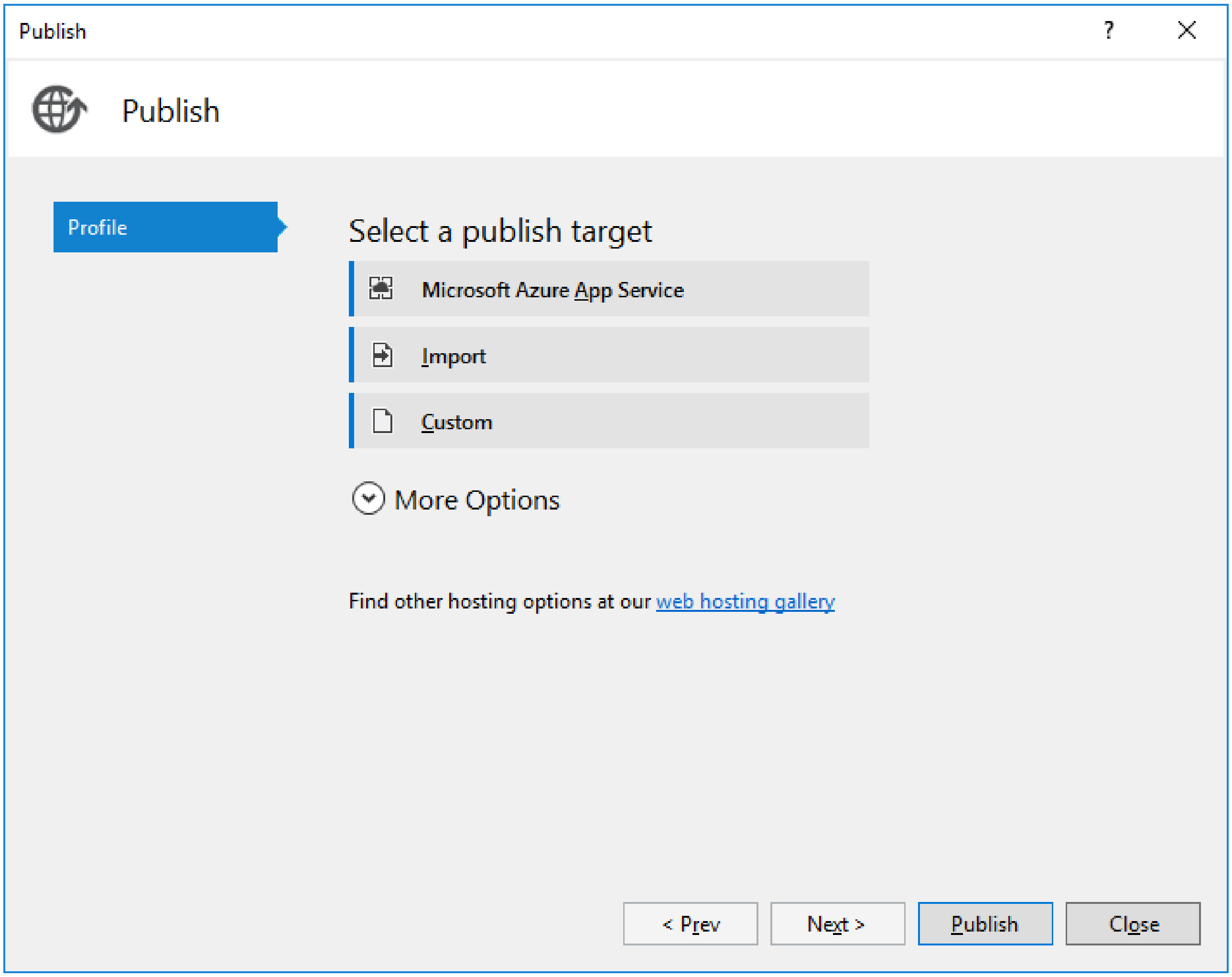


Step 3: Deploy the Web service to IIS

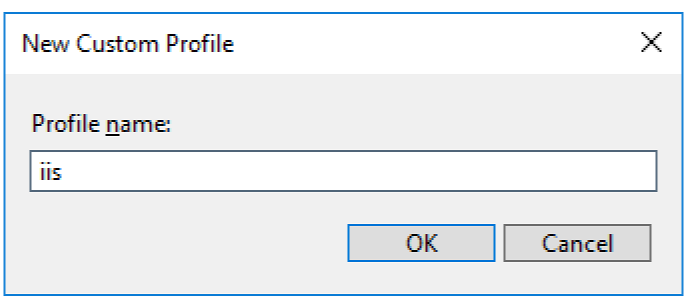
In the Solution Explorer window of Visual Studio, right-click the project **services-Site/WS2DB** and select "Publish Web App" from the content menu.



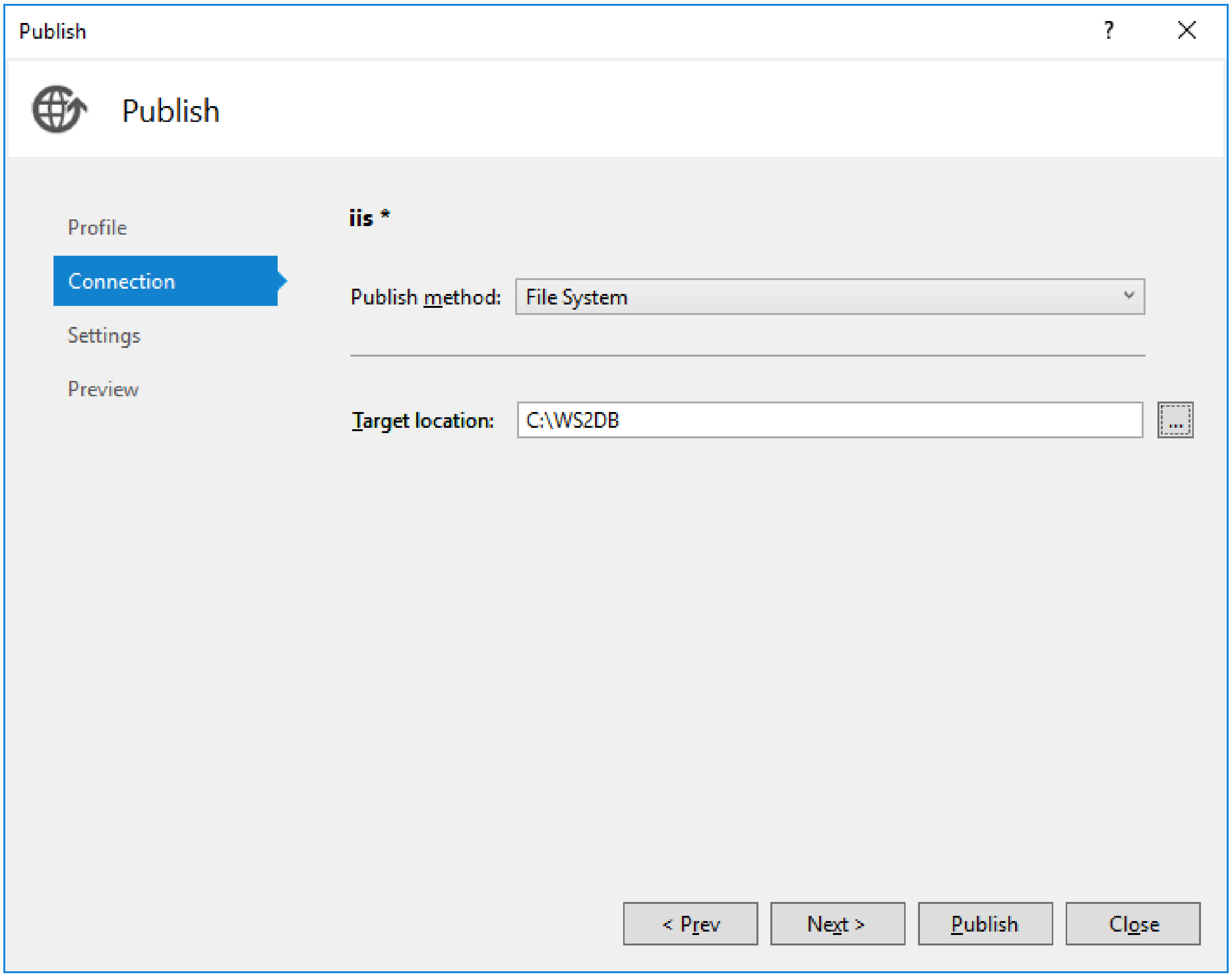
When prompted to select a publish target, select "Custom".



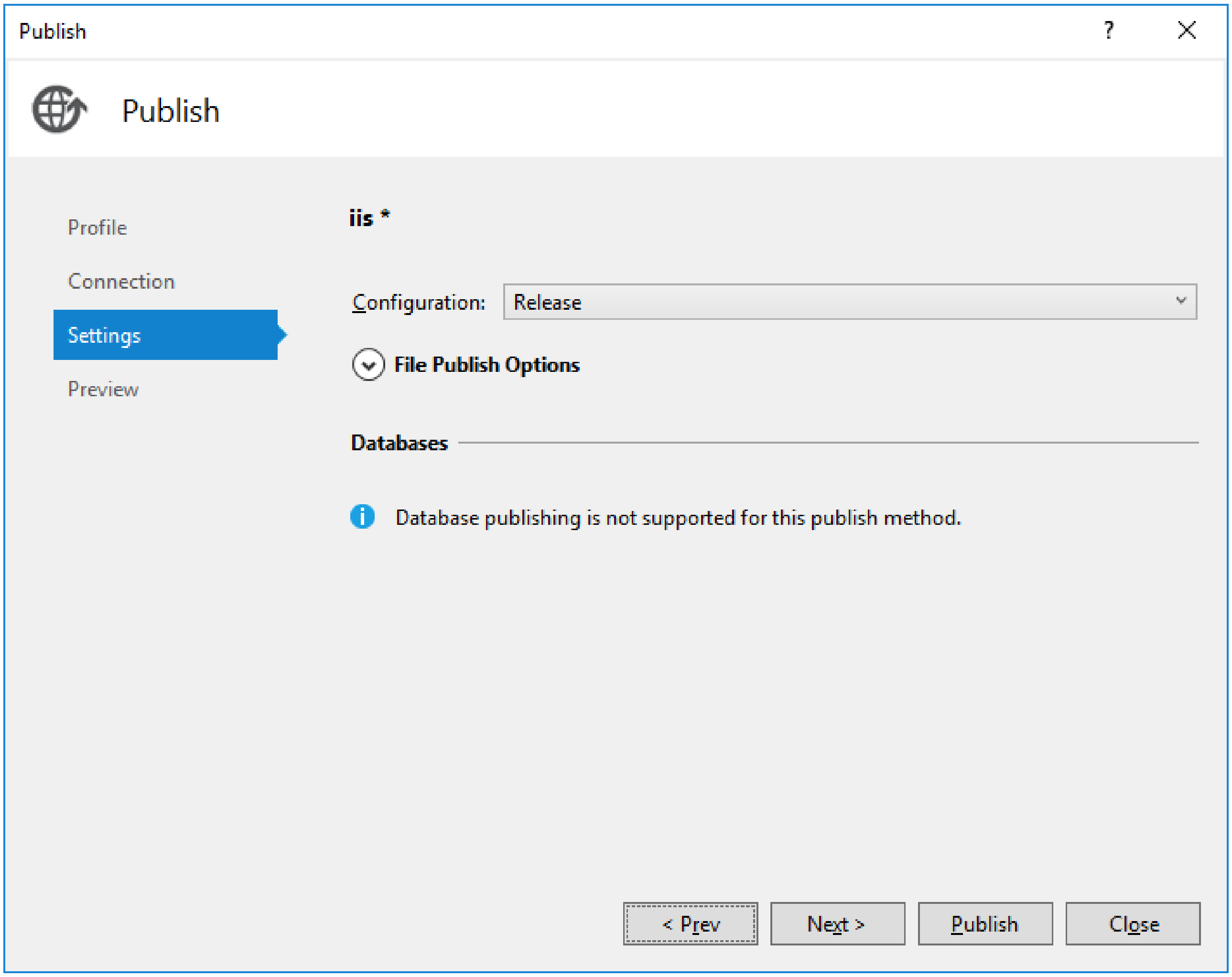
When prompted to enter a profile name, enter a value that will be easy to recognize later, for example "iis".



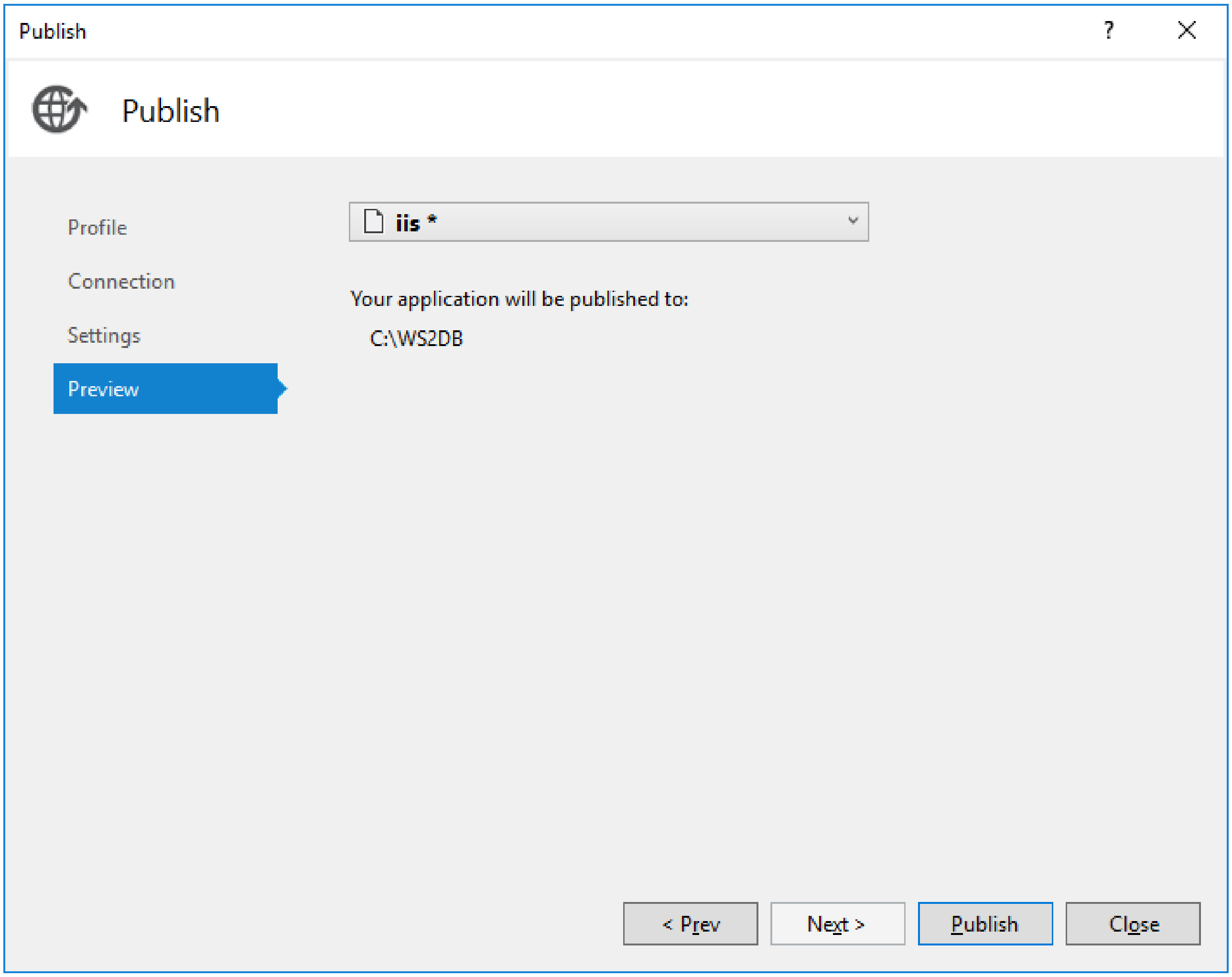
Select "File System" as publish method and choose a destination folder (in this example, "C:\WS2DB").



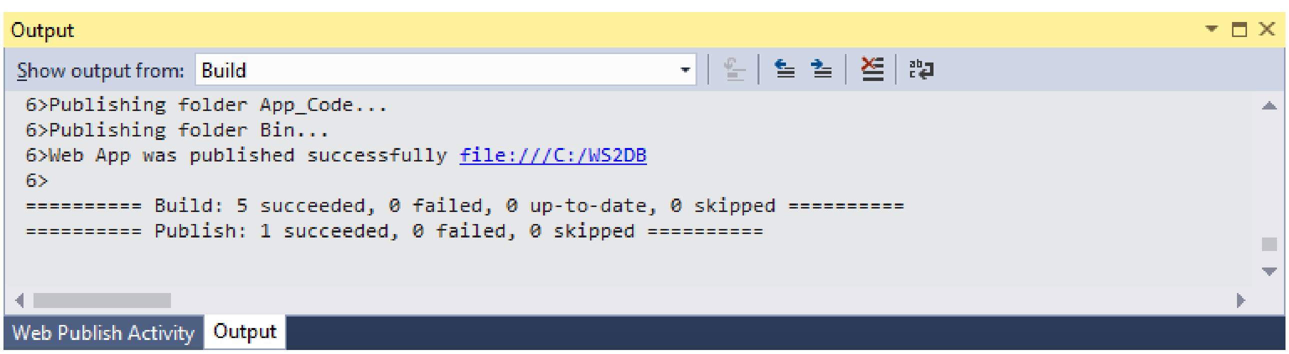
Click **Next**. Leave the default settings as is.



Click **Next**, and then click **Publish**.



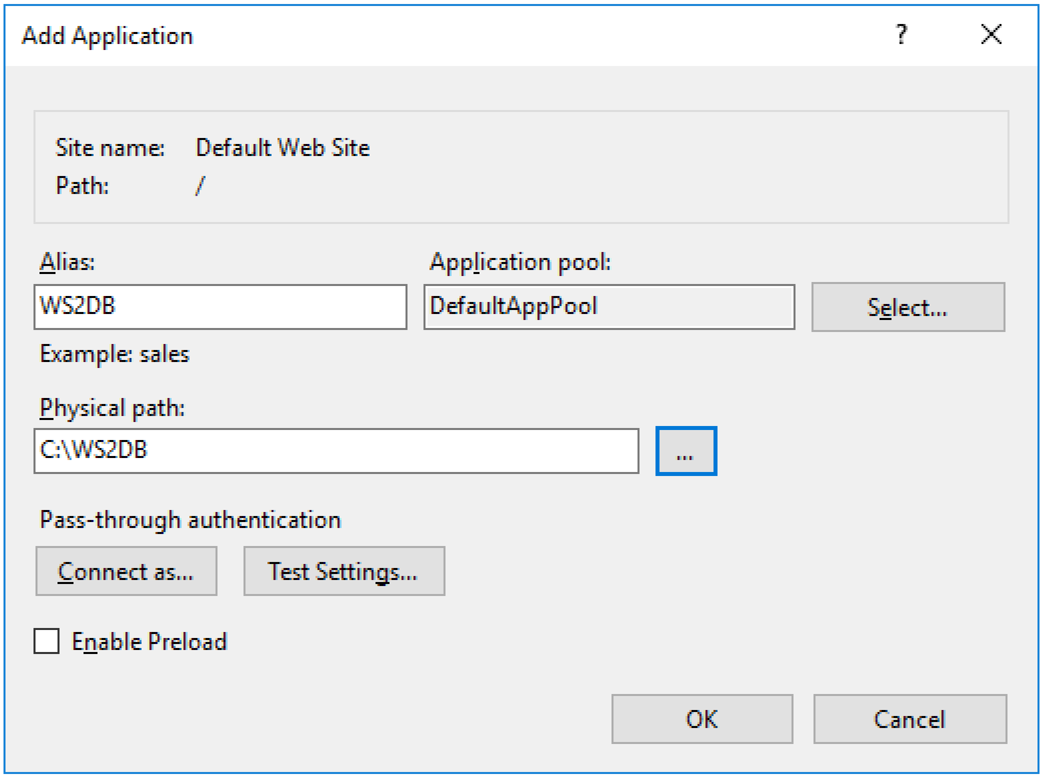
The Output window displays the result.



Step 4: Configure the application in IIS Manager

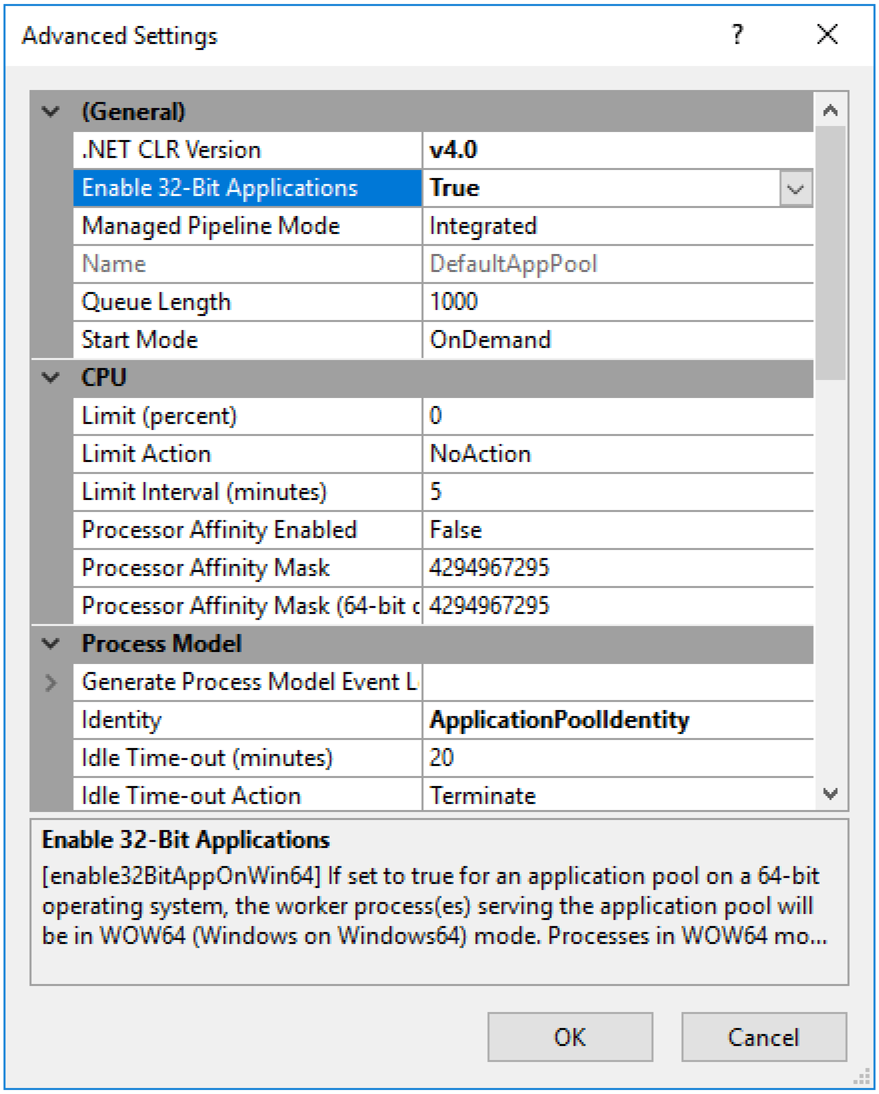
You can add the new application to IIS as follows:

1. In IIS, right-click **Sites | Default Web Site**, and select **Add Application** from the context menu.
2. Under "Alias name", enter a name to identify your web application ("services", in this example).
3. Under "Physical path", enter the path where the application was previously published ("C:\WS2DB", in this example).



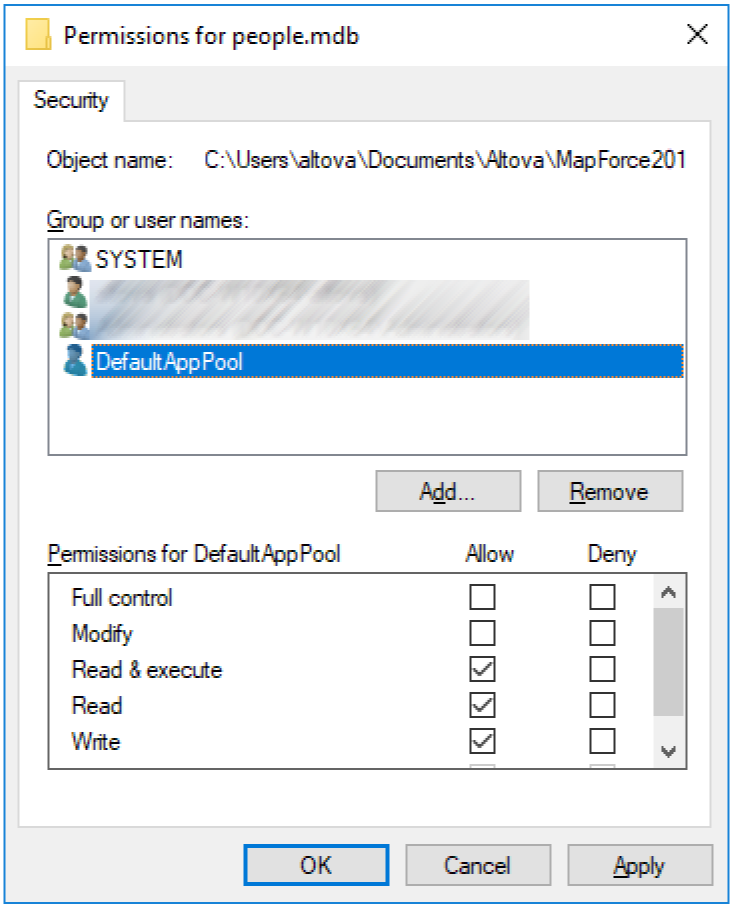
Additionally, for this particular example, it is necessary that 32-bit applications are enabled for the pool where the app is running.

1. In IIS, right-click the application pool where the app was deployed, and select **Advanced Settings** from the context menu.
2. Under **Enable 32-bit Applications**, select the option **True**.

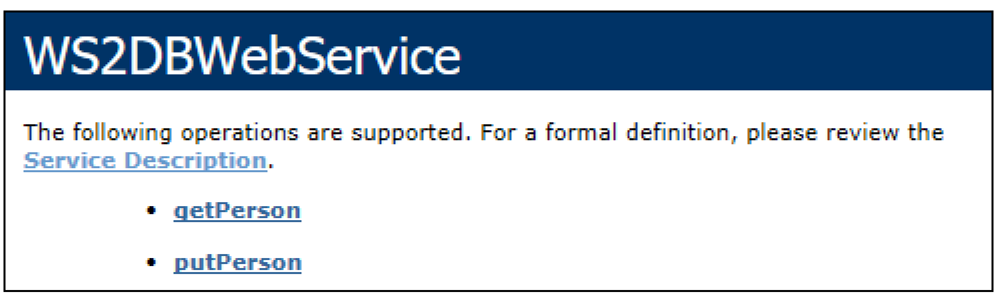


Finally, since this example reads data from a Microsoft Access database, the IIS application pool must have permissions to access the database file. Unless you modified the MapForce project, the database is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\people.mdb**.

1. In Windows Explorer, right-click the **people.mdb** file, and select **Properties**.
2. On the **Security** tab, click **Edit**.
3. Click **Add**, and enter the object name as follows: **IIS AppPool\DefaultAppPool**. Change "DefaultAppPool" to a different value if you deployed the app to a pool other than the default one.
4. Grant the required permissions, and click **Apply**. (In this example, permission to read and write the database file is required.)



To test if the Web service is up and running, access the following URL: `http://localhost/WS2DB`.



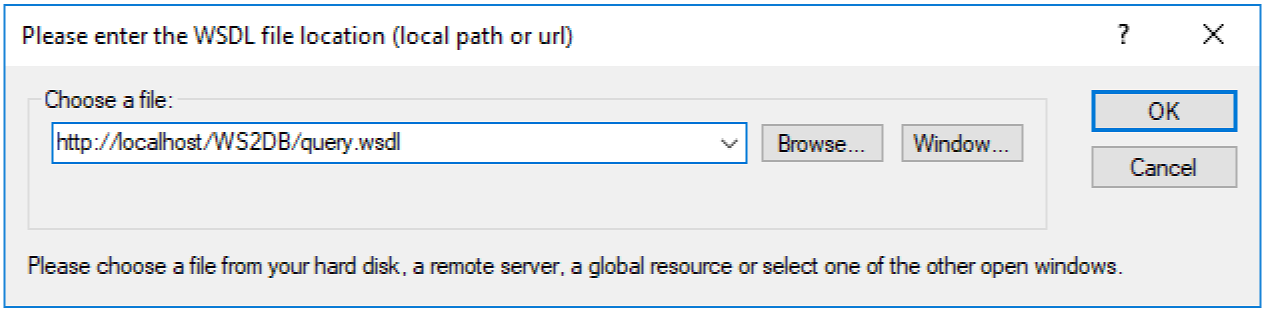
If the Web service details load in the browser, you can proceed to calling the Web service from a client application, as shown below.

Calling the Web service

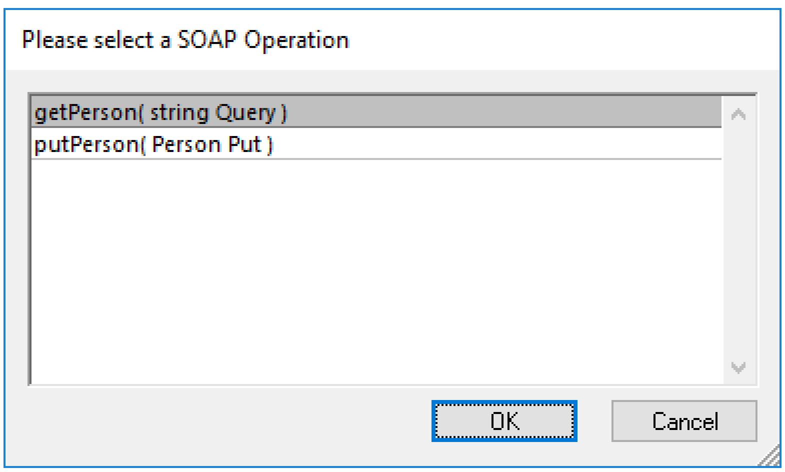
If you followed the steps above literally, then the WSDL of the Web service is available at `http://localhost/WS2DB/query.wsdl`. To call the Web service we just deployed, we will use XMLSpy, although you can use, of course, other SOAP clients as well.

Note: It is recommended to access the genuine WSDL at `http://localhost/WS2DB/query.wsdl` rather than using the "?wsdl" query method. The latter may return a WSDL file lacking some features of the original or one which does not work at all.

Run XMLSpy Enterprise Edition. On the **SOAP** menu, click **Create New SOAP Request**. When prompted, enter the WSDL of the Web service (in this example, `http://localhost/WS2DB/query.wsdl`).



Click OK. When prompted to select a SOAP Operation, select **getPerson**.



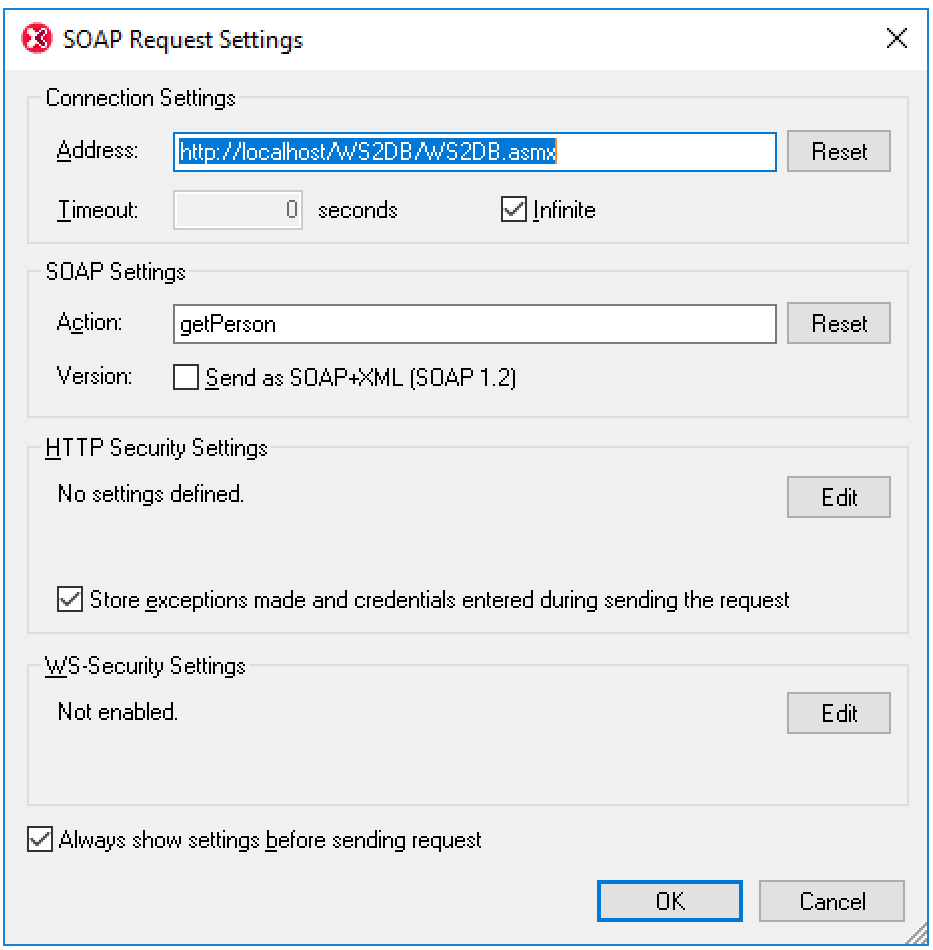
XMLSpy generates a sample request like the one below.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <getPerson SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <Query xsi:type="xsd:string">String</Query>
      </getPerson>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

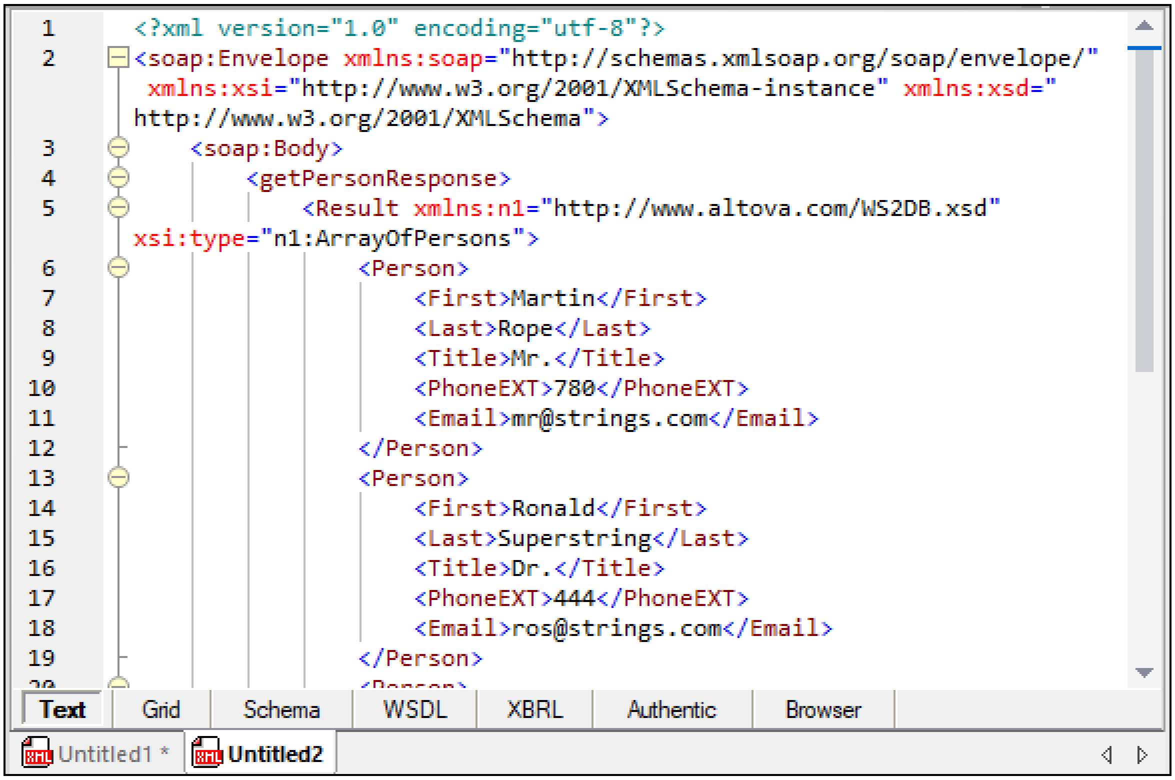
Locate the <Query> element of the request, and replace "String" with the value you want to supply as input to the Web service. As stated before, this Web service retrieves people details from an Access database. For the scope of this example, in order for the call to actually retrieve some data, "String" must be replaced with "Ro" (in other words, only people whose first or last name contains "Ro" would be retrieved). The new request now looks as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <getPerson SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <Query xsi:type="xsd:string">Ro</Query>
      </getPerson>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

You can now send the request above to the server. On the **SOAP** menu, click **Send Request to Server**. The SOAP Request Settings dialog box opens.



Click **OK**. XMLSpy initiates the call, and returns the response in the editor. The image below illustrates a successful response.



Using the same approach as described above, you can also call the **putPerson** operation exposed by this Web service. For step-by-step instructions, see "Calling the Web Service[1241]" section in the Java tutorial. Only the WSDL URL is different in the Java tutorial; all other instructions for calling the Web service are identical.

Troubleshooting

The following table lists common problems that you might encounter when building or invoking the Web service, and their solution.

Problem	Solution
Calling the Web service returns the following error: `The 'Microsoft.Jet.OLEDB.4.0' provider is not registered on the local machine."`	1. Build the C# solution in Visual Studio as a 32-bit application. 2. Deploy to IIS. 3. Right-click the IIS application pool where you deployed the application, and select **Advanced Settings**. 4. Under **Enable 32-bit applications**, select **True**.
Calling the Web service returns the following error: `The Microsoft Jet database engine cannot open the file '<path>\people.mdb'.  It is already opened exclusively by another`	• Grant to the IIS application pool permission to read the Access database file, see "Step 4: Configure the application in IIS Manager" above.

Problem	Solution
`user, or you need permission to view its data.`	
Calling the Web service returns the following error: `System.Data.OleDb.OleDbException: Operation must use an updateable query.`	• Grant to the IIS application pool permission to write the Access database file, see "Step 4: Configure the application in IIS Manager" above.

7.6 Example: Generating SOAP Web Services (Java)

This example illustrates how to generate a SOAP Web service with MapForce. You will generate Java program code from a sample MapForce project available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Query Person database.mfp**. This project implements a SOAP Web service that exposes two operations: `getPerson` and `putPerson`. This Web service communicates in the background with a Microsoft Access database to retrieve or add person details, respectively, through the operations above. For an example of how to create such projects in MapForce, see Creating Web Service Projects from WSDL Files 1210.

This example requires Java 7 and Apache Ant to compile the generated Java code. The .aar file will be deployed to a running Apache Tomcat server with Axis2 installed.

For simplicity, this example uses a basic Web server configuration and should not be taken as prescriptive. In a production environment, you may need to use a different approach as demanded by the specifics of your Web server, the security policies in your organization, and other factors.

To call the SOAP Web service, this example makes use of XMLSpy Enterprise Edition. If you do not have XMLSpy, you will need a SOAP client to test the Web service, or, alternatively, write program code that can call the Web service.

Note: As stated before, this Web service example accesses a Microsoft Access database. With Java 7, connectivity to Access databases is possible through the ODBC-JDBC bridge included in Java. However, the ODBC-JDBC bridge has been removed in Java 8, and Oracle recommends using JDBC drivers provided by the vendor, see http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/bridge.html. Therefore, to run this example in a Java 8 or newer environment, you will need to find and install third-party JDBC drivers that can connect to a Microsoft Access database.

Step 1: Prepare the query.wsdl file

This example Web service originates from the following WSDL file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\query.wsdl**. By default, the **query.wsdl** file defines the service details for C# and Java in two separate sections, one of which is commented out. Therefore, before generating Java code, locate the following section in the **query.wsdl** file and make sure that it is commented out:

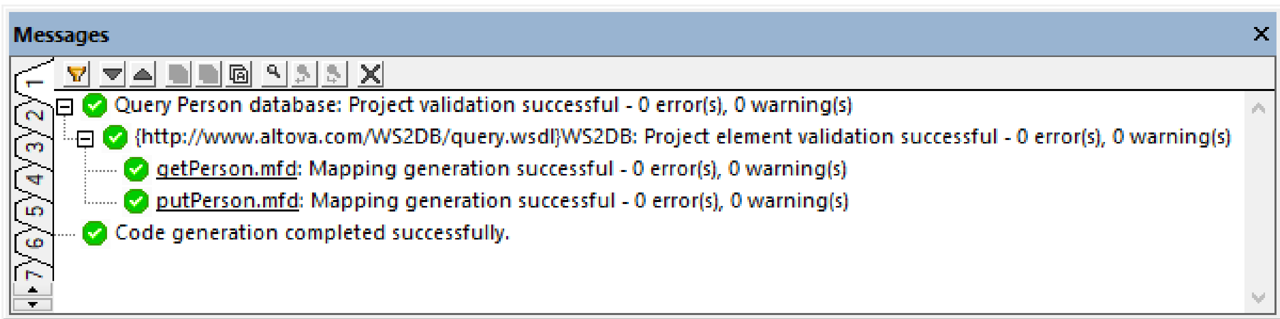
```
<!--service name="WS2DB">
   <port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
      <soap:address location="http://localhost/WS2DB/WS2DB.asmx"/>
   </port>
</service-->
```

Also, make sure that the following section (only applicable to Java) is NOT commented out:

```
<service name="WS2DB">
   <port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
      <soap:address location="http://localhost:8080/axis/services/WS2DB"/>
   </port>
</service>
```

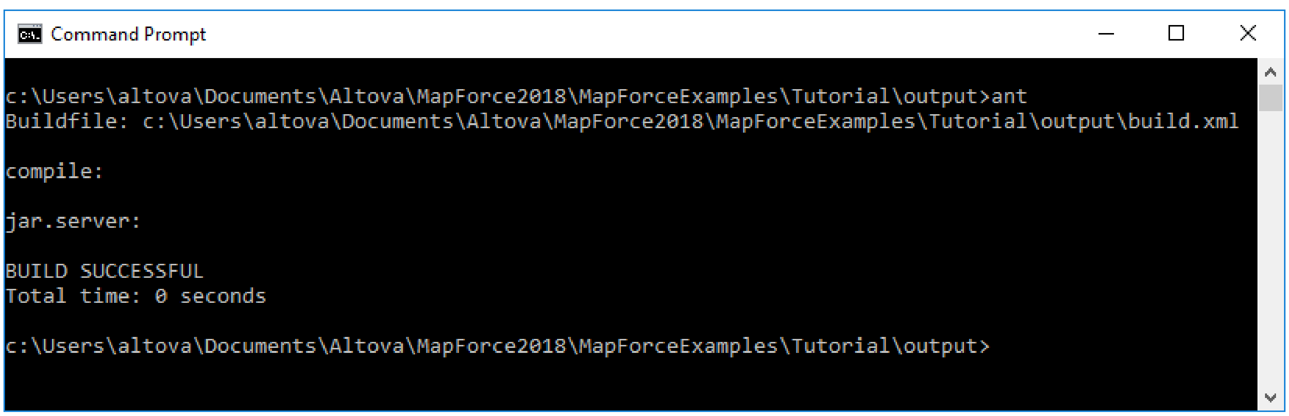
Step 2: Generate Java code and build it

Run MapForce and open the "Query Person database.mfp" project from the folder **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**. Right-click the project in the Project window, and select **Generate code in | Java**. MapForce generates program code and displays the outcome in the Messages window:



By default, code will be generated in a subdirectory called "output", relative to the project's directory. For the scope of this tutorial, we will leave the default settings as is. However, if you want to change the output directory, see Setting the Code Generation Settings 118.

To build the Java code, run Ant at the command line in the same directory as the **build.xml** file.



As a result, an Axis Archive (.aar) file called **Query_Person_database.aar** is generated in the output directory.

Before you can call Ant from any directory as illustrated above, it must be installed and configured correctly. Here is a quick summary (for more details, check the Ant manual from http://ant.apache.org):

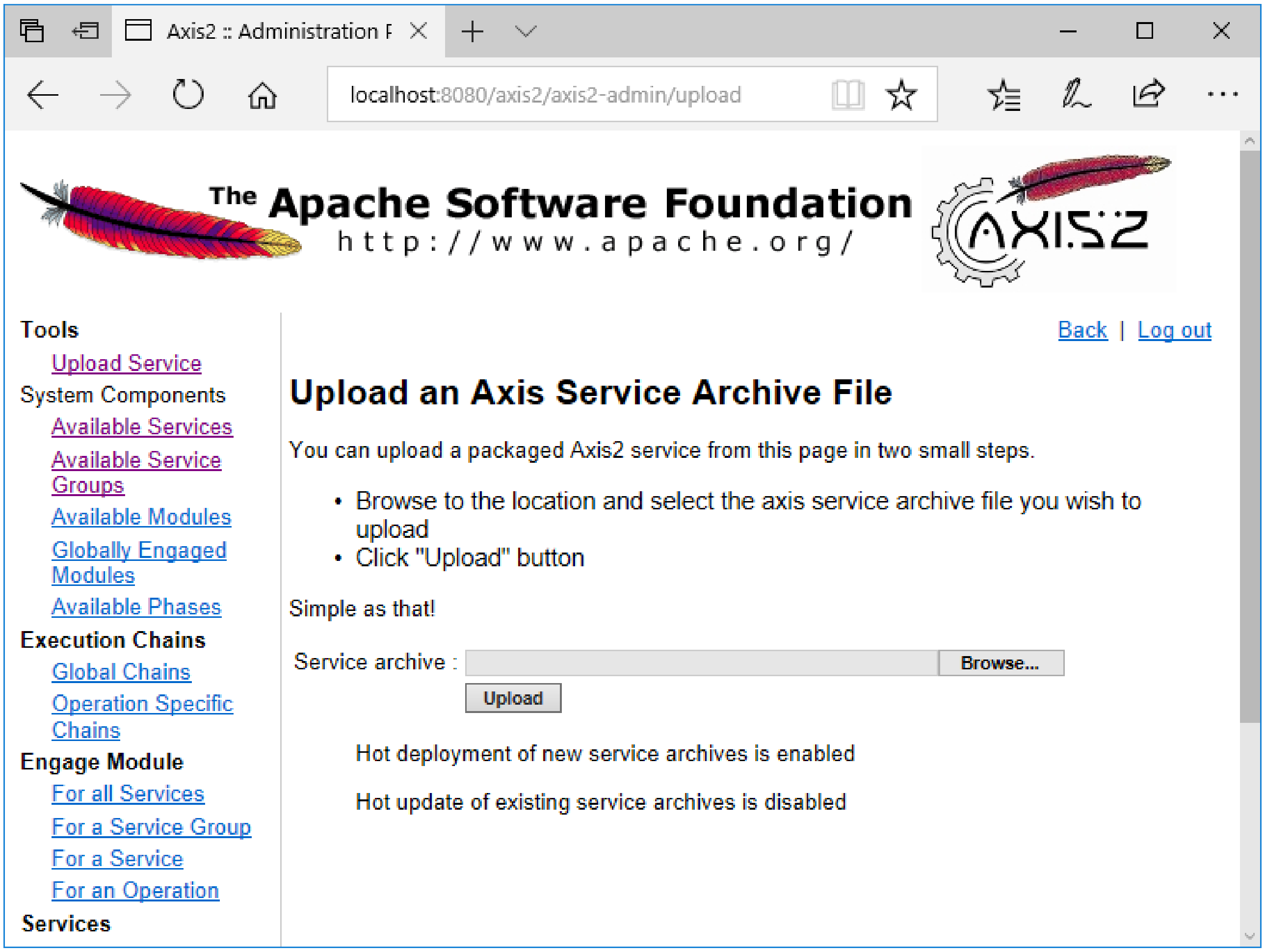
- The `PATH` environment variable must include the path to the "bin" subdirectory of Ant root directory.
- The `ANT_HOME` environment variable must be set to the directory where Ant is installed.
- The `AXIS2_HOME` environment variable must be set to the directory where the standard distribution of Axis2 is installed.
- The `JAVA_HOME` environment variable must be set to the directory where Java JDK is installed.
- Check that Ant is running by typing "ant" at the command line. The message "Buildfile: build.xml does not exist!" means that Ant is configured and is looking for a build file.

Step 3: Deploy the Web service to Axis 2

In this example, the web service is uploaded to Tomcat from the Axis2 administration page. The latter can be installed and deployed to a running Tomcat server as follows (for more details, refer to Axis2 documentation):

1. Download the Axis2 WAR (Web Archive) distribution from http://axis.apache.org/axis2/java/core/download.html.
2. Unpack the downloaded package to a local directory.
3. Copy the **axis2.war** file to Tomcat's directory **%CATALINA_HOME%\webapps**.
4. Check that Axis2 is running, by opening: `http://localhost:8080/axis2`

Open the Axis2 Web administration page and upload the .aar file generated in the previous step.

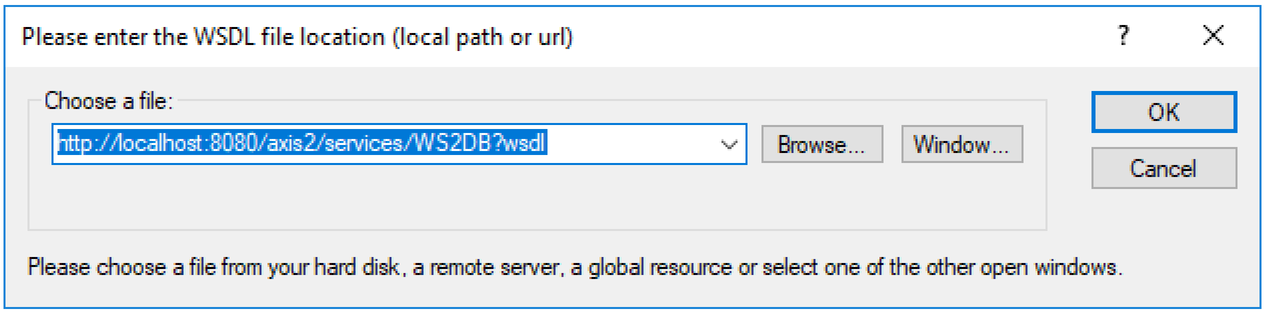


A new Web service called "WS2DB" should now be available and active in the Axis2 administration page.

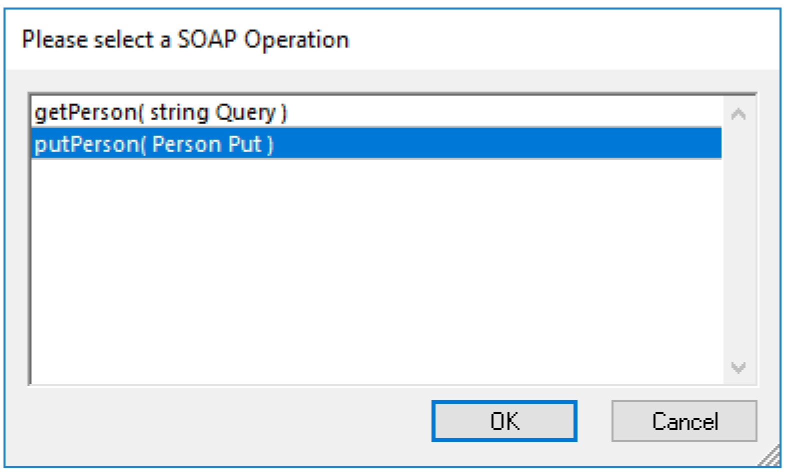
Calling the Web service

If you followed the steps above literally, then the WSDL of the Web service is available at `http://localhost:8080/axis2/services/WS2DB?wsdl`. To call the Web service we just deployed, we will use XMLSpy, although you can use, of course, other SOAP clients as well.

Run XMLSpy Enterprise Edition. On the **SOAP** menu, click **Create New SOAP Request**. When prompted, enter the WSDL of the Web service (in this example, `http://localhost:8080/axis2/services/WS2DB?wsdl`).



Click OK. When prompted to select a SOAP Operation, select **putPerson**.



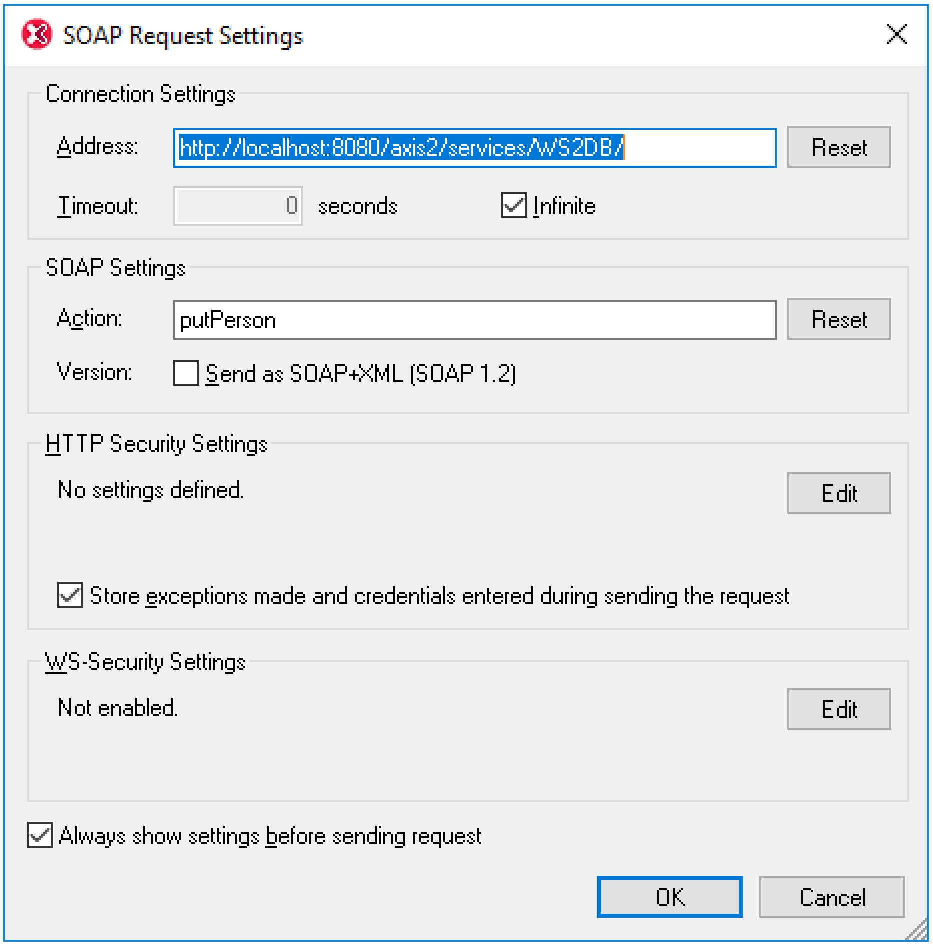
XMLSpy generates a sample request like the one below.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:m0="http://www.altova.com/WS2DB.xsd">
   <SOAP-ENV:Body>
      <putPerson SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <Put xsi:type="m0:Person">
            <First xsi:type="xsd:string">String</First>
            <Last xsi:type="xsd:string">String</Last>
            <Title xsi:type="xsd:string">String</Title>
            <PhoneEXT xsi:type="xsd:string">String</PhoneEXT>
            <Email xsi:type="xsd:string">String</Email>
         </Put>
      </putPerson>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

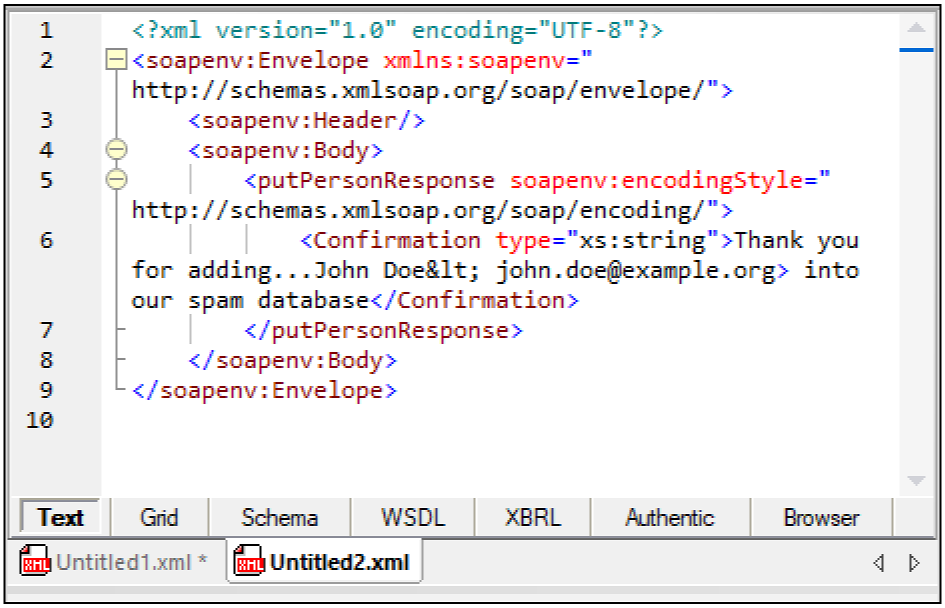
Replace "String" with the person details you want to supply as input to the Web service. For the scope of this example, you could use a request like the one below:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:m0="http://www.altova.com/WS2DB.xsd">
   <SOAP-ENV:Body>
      <putPerson SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <Put xsi:type="m0:Person">
            <First xsi:type="xsd:string">John</First>
            <Last xsi:type="xsd:string">Doe</Last>
            <Title xsi:type="xsd:string">Mr</Title>
            <PhoneEXT xsi:type="xsd:string">123</PhoneEXT>
            <Email xsi:type="xsd:string">john.doe@example.org</Email>
         </Put>
      </putPerson>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

You can now send the request above to the server. On the **SOAP** menu, click **Send Request to Server**. The SOAP Request Settings dialog box opens.



Click **OK**. XMLSpy initiates the call, and returns the response in the editor. The image below illustrates a successful response.



Using the same approach as described above, you can also call the **getPerson** operation exposed by this Web service. For step-by-step instructions, see "Calling the Web Service[1232]" section in the C# tutorial. Only the WSDL URL is different in the C# tutorial; all other instructions for calling the Web service are identical.

8 Advanced Mapping Scenarios

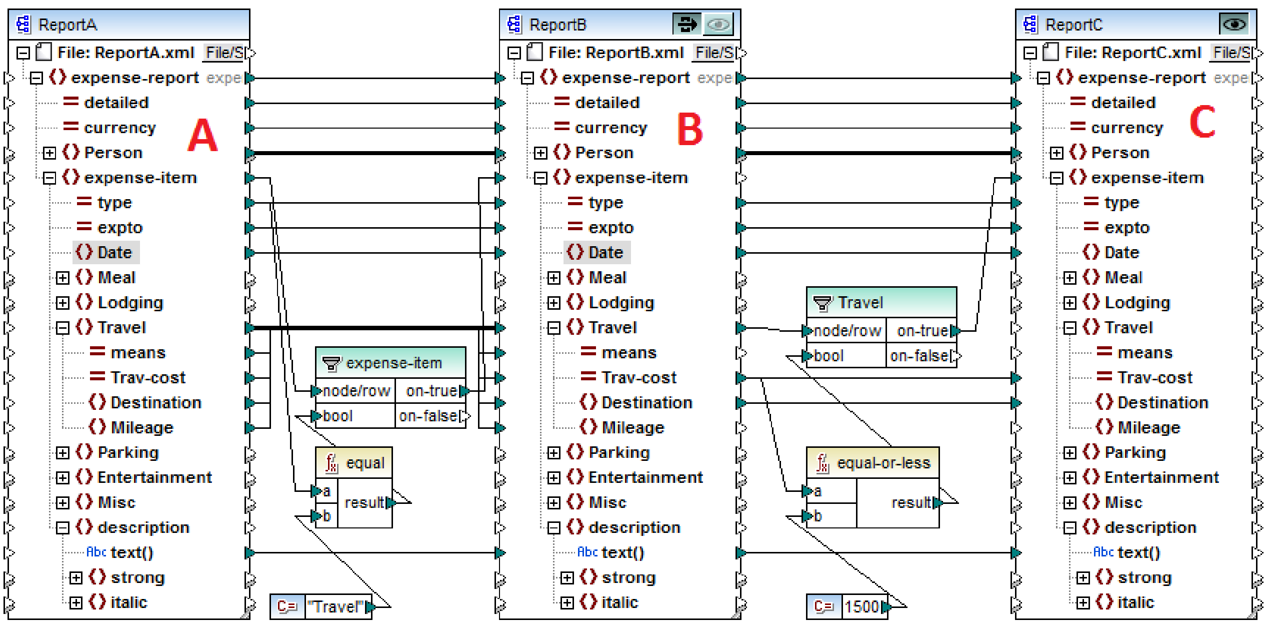
Altova website: Data integration tool

This section describes advanced mapping scenarios and includes the following topics:

8.1 Chained Mappings

MapForce supports mappings that consist of multiple components in a mapping chain. Chained mappings are mappings where at least one component acts both as a source and a target. Such a component creates output which is later used as input for a following mapping step in the chain. Such a component is called an "intermediate" component.

For example, the mapping illustrated below shows an expense report (in XML format) that is being processed in two stages. The part of the mapping from A to B filters out only those expenses that are marked as "Travel". The mapping from B to C filters out only those "Travel" expenses that have a travel cost less than 1500. Component B is the "intermediate" component, as it has both input and output connections. This mapping is available at the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ChainedReports.mfd.



ChainedReports.mfd

Chained mappings introduce a feature called "pass-through". "Pass-through" is a preview capability allowing you to view the output produced at each stage of a chained mapping in the Output window. For example, in the mapping above, you can preview and save the XML output resulting from A to B, as well as the XML output resulting from B to C.

Note: The "pass-through" feature is available only for file-based components (for example, XML, CSV, and text). Database components can be intermediate, but the pass-through button is not shown. The intermediate component is always regenerated from scratch when previewing or generating code. This would not be feasible with a database as it would have to be deleted prior to each regeneration.

If the mapping is executed by MapForce Server, or by generated code, then the full mapping chain is executed. The mapping generates the necessary output files at each step in the chain, and the output of a step of a mapping chain is forwarded as input to the following mapping step.

It is also possible for intermediate components to generate dynamic file names. That is, they can accept connections to the "File:" item from the mapping, provided that the component is configured correspondingly. For more information, see Processing Multiple Input or Output Files Dynamically 1313.

Preview button

Both the component B and the component C have preview buttons. This allows you to preview in MapForce the intermediate mapping result of B, as well as the final result of the chained mapping. Click the preview button of the respective component, then click Output to see the mapping result.

"Intermediate" components with the pass-through button active cannot be previewed. Their preview button is automatically disabled, because it is not meaningful to preview and let data pass through at the same time. To see the output of such a component, first click the "pass-through" button to deactivate it, and then click the preview button.

Pass-through button

The intermediate component B has an extra button in the component title bar called "pass-through".

If the pass-through button is **active** , MapForce maps all data into the preview window in one go; from component A to component B, then on to component C. Two result files will be created:

- the result of mapping component A to intermediate component B
- the result of the mapping from the intermediate component B, to target component C.

If the pass-through button is **inactive** , MapForce will execute only parts of the full mapping chain. Data is generated depending on which preview buttons are active:

- If the preview button of component B is active, then the result of mapping component A to component B is generated. The mapping chain actually stops at component B. Component C is not involved in the preview at all.
- If the preview button of component C is active, then the result of mapping intermediate component B to the component C is generated. Because pass-through is inactive, automatic chaining has been interrupted for component B. Only the right part of the mapping chain is executed. Component A is not used.

> When the "pass-through" button is inactive, it is important that the intermediate component has identical file names in the "Input XML File" and "Output XML File" fields. This ensures that the file generated as output when you preview the portion of the mapping between A and B is used as input when you preview the portion of the mapping between B and C. Also, in generated code, or in MapForce Server execution, this ensures that the mapping chain is not broken.

As previously mentioned, if the mapping is executed by MapForce Server, or by generated code, then the output of all components is generated. In this case, the settings of the pass-through button of component B, as well as the currently selected preview component, are disregarded. Taking the mapping above as example, two result files will be generated, as follows:

1. The output file resulting from mapping component A to B
2. The output file resulting from mapping component B to C.

The following sections, Example: Pass-Through Active[1248] and Example: Pass-Through Inactive[1252], illustrate in more detail how the source data is transferred differently when the pass-through button is active or inactive.

8.1.1 Example: Pass-Through Active

The mapping used in this example (**ChainedReports.mfd**) is available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. This mapping processes an XML file called **ReportA.xml** that contains travel expenses and looks as shown below. For simplicity, the namespace declaration and some `expense-item` elements have been omitted:

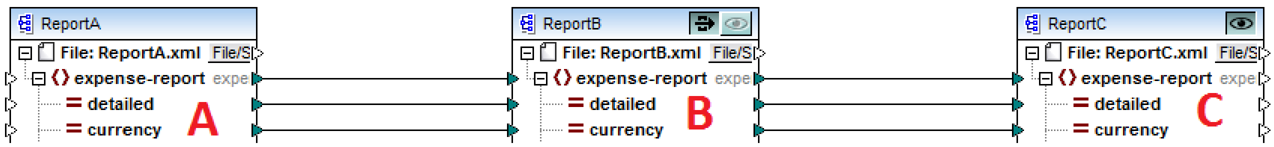
```
<?xml version="1.0" encoding="UTF-8"?>
<expense-report currency="USD" detailed="true">
   <Person>
      <First>Fred</First>
      <Last>Landis</Last>
      <Title>Project Manager</Title>
      <Phone>123-456-78</Phone>
      <Email>f.landis@nanonull.com</Email>
   </Person>
   <expense-item type="Travel" expto="Development">
      <Date>2003-01-02</Date>
      <Travel Trav-cost="337.88">
         <Destination/>
      </Travel>
      <description>Biz jet</description>
   </expense-item>
   <expense-item type="Lodging" expto="Sales">
      <Date>2003-01-01</Date>
      <Lodging Lodge-cost="121.2">
         <Location/>
      </Lodging>
      <description>Motel mania</description>
   </expense-item>
   <expense-item type="Travel" expto="Marketing">
      <Date>2003-02-02</Date>
      <Travel Trav-cost="2000">
         <Destination/>
      </Travel>
      <description>Hong Kong</description>
   </expense-item>
</expense-report>
```

ReportA.xml

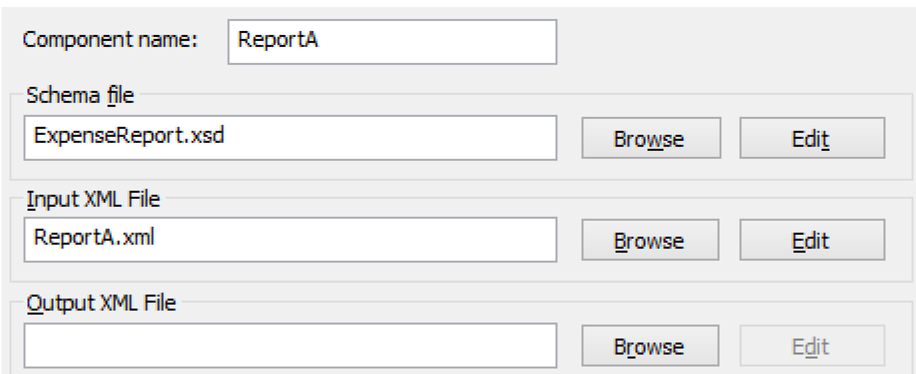
The goal of the mapping it to produce, based on the file above, two further reports:

- **ReportB.xml** - this report should contain only those travel expenses that are of type "Travel".
- **ReportC.xml** - this report should contain only those travel expenses that are of type "Travel" and do not exceed 1500.

To achieve this goal, the intermediate component of the mapping (component B) has the pass-through button  active, as shown below. This causes the mapping to be executed in stages: from A to B, and then from B to C. The output created by the intermediate component will be used as input for the mapping between B and C.

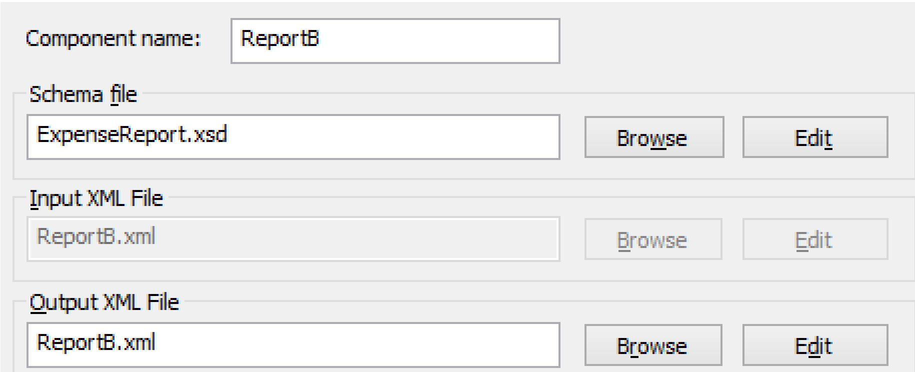


The names of generated output files at each stage in the mapping chain is determined by the settings of each component. (To open the component settings, right-click it, and then select **Properties** from the context menu). Namely, the first component is configured to read data from an XML file called **ReportA.xml**. Because this is a source component, the **Output XML File** field is irrelevant and it was left empty.



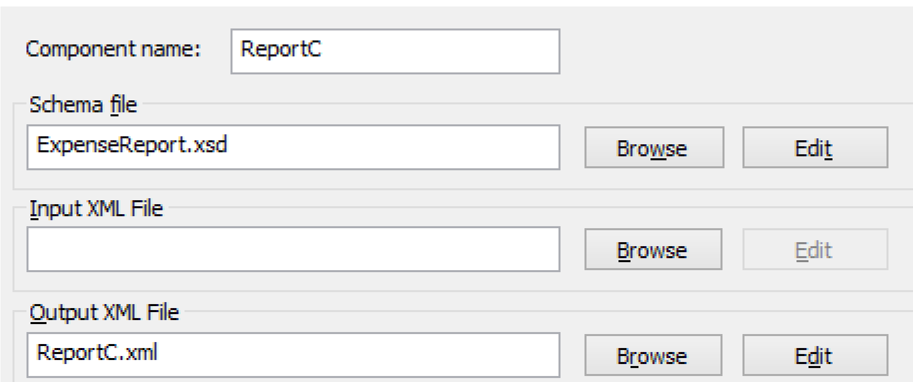
Settings of the source component

As shown below, the second component (**ReportB**) is configured to create an output file called **ReportB.xml**. Notice that the **Input XML File** field is grayed out. When pass-through is active (as in this example), the **Input XML File** field of the intermediate component is automatically deactivated. An input file name need not exist for the mapping to execute, because the output created at this stage in the mapping is stored in a temporary file and reused further in the mapping. Also, if an **Output XML File** is defined (as illustrated below), then it is used for the file name of the intermediate output file. If no **Output XML File** is defined, a default file name will be automatically used.



Settings of the intermediate component

Finally, the third component is configured to produce an output file called **ReportC.xml**. The **Input XML File field** is irrelevant here, because this is a target component.

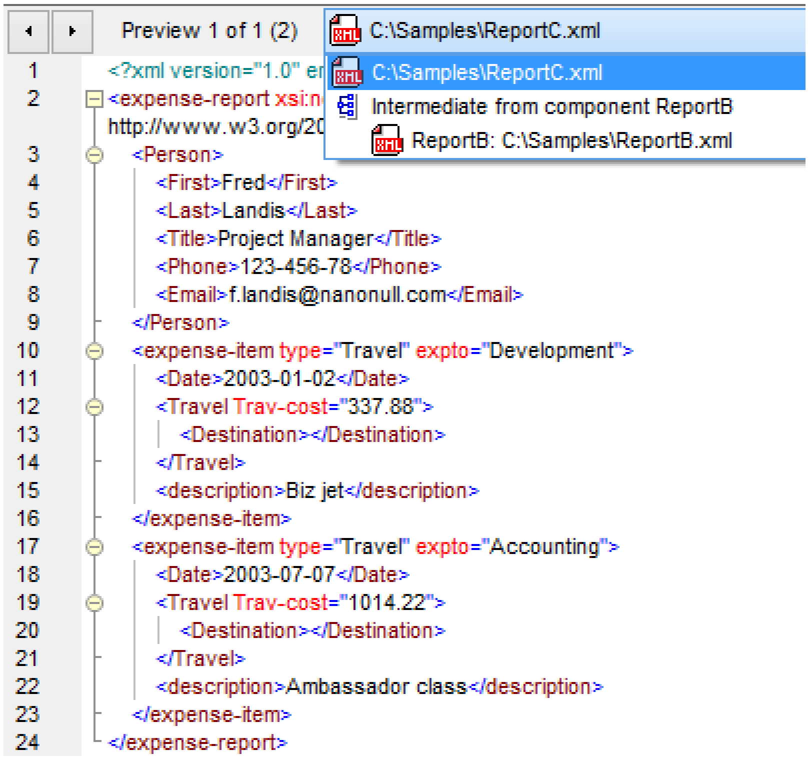


Settings of the target component

If you preview the mapping by clicking the **Output** tab in the mapping window, two files are shown in the output, as expected:

1. **ReportB.xml**, which represents the result of the mapping A to B
2. **ReportC.xml**, which represents the result of mapping B to C.

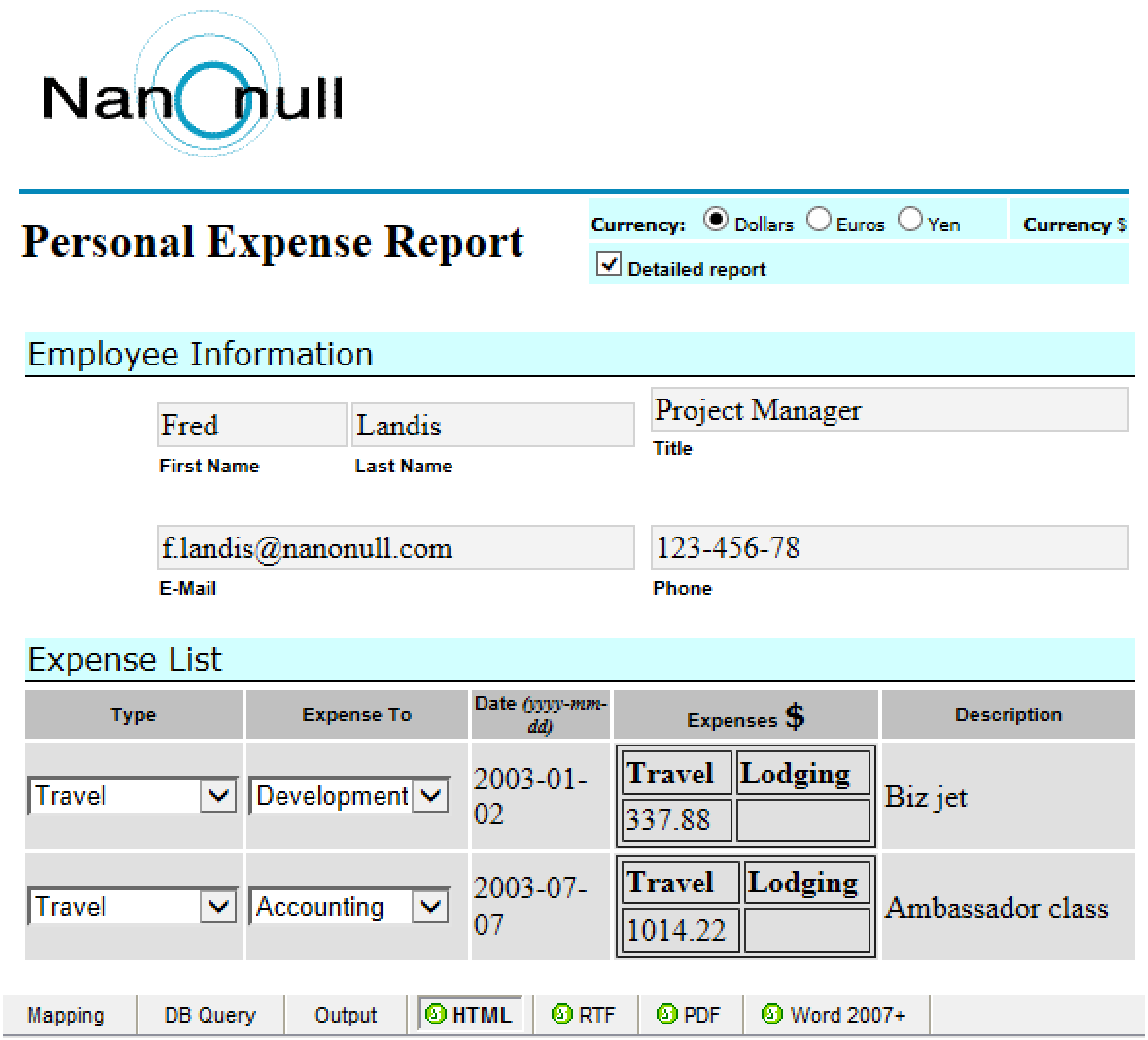
To select which of the two generated output files should be displayed in the Output window, either click the arrow buttons, or select the desired entry from the dropdown list.



Generated output files

When the mapping is executed by MapForce, the setting "Write directly to final output file" (configured from **Tools | Options | General**) determines whether the intermediate files are saved as temporary files or as physical files. Note that this is only valid when the mapping is previewed directly in MapForce. Had this mapping been executed by MapForce Server or by generated code, actual files would be produced at each stage in the mapping chain.

If StyleVision is installed, and if a StyleVision Power Stylesheet (SPS) file has been assigned to the target component (as in this example), then the final mapping output can be viewed (and saved as) HTML, RTF, PDF, or Word file. To generate and view this output in MapForce, click the tab with the corresponding name.

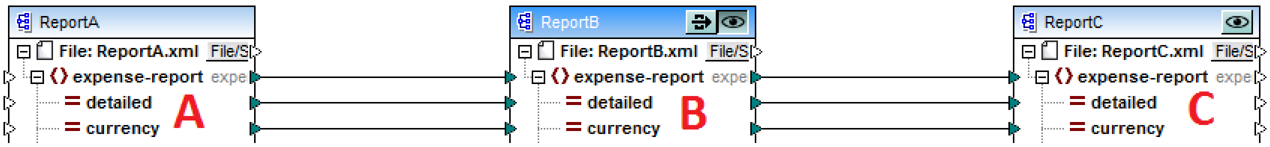


Generated HTML output

Note that only the output of the final target component in the mapping chain is displayed. To display StyleVision output of intermediary components, you would need to deactivate the pass-through button, and preview the intermediate component (as shown in Example: Pass-Through Inactive 1252).

8.1.2 Example: Pass-Through Inactive

The mapping used in this example (**ChainedReports.mfd**) is available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. This example illustrates how output is generated differently when the pass-through button is deactivated on the intermediate component.

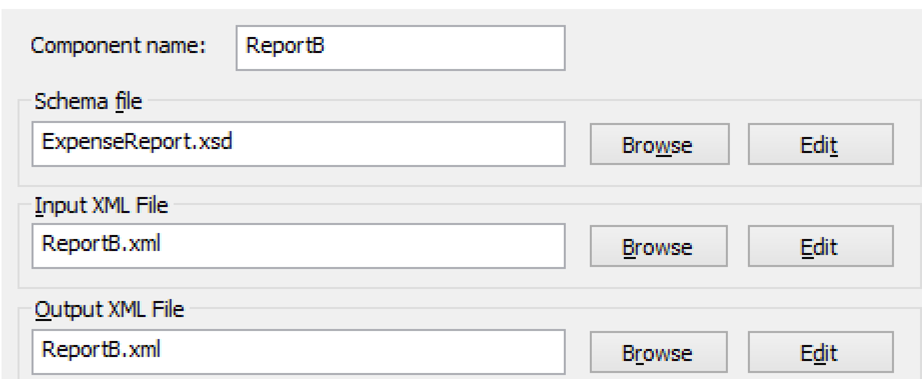


As explained in [Example: Pass-Through Active](1248), the goal of the mapping is to produce two separate reports. In the previous example, the pass-through button was active , and both reports were generated as expected and could be viewed in the **Output** tab. However, if you want to preview only one of the reports (either **ReportB.xml** or **ReportC.xml**), then the pass-through button must be deactivated ( ). More precisely, deactivating the pass-through button may be useful if you want to achieve the following:

- Preview only output generated from A to B, and disregard the portion of the mapping from B to C
- Preview only output generated from B to C, and disregard the portion of the mapping from A to B.

When you deactivate the pass-through button as shown above, you can choose whether to preview either **ReportB** or **ReportC** (notice that both have preview buttons).

Deactivating the pass-through button also lets you to choose what input file should be read by the intermediate component. In most cases, this should be the same file as defined in **Output XML File** field (as in this example).



Settings of the intermediate component

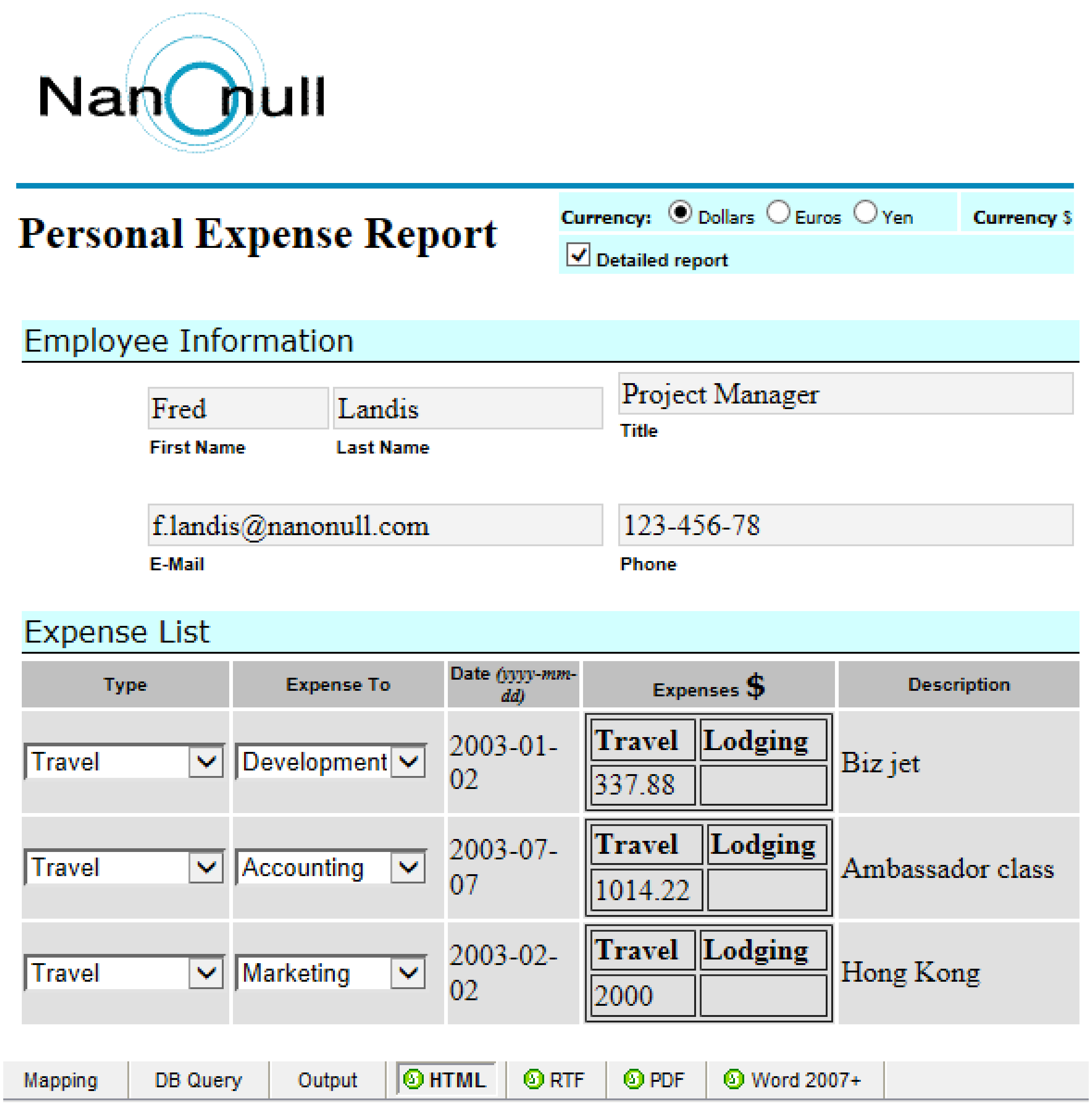
Having the same input and output file on the intermediate component is particularly important if you intend to generate code from the mapping, or run the mapping with MapForce Server. As previously mentioned, in these environments, all outputs created by each component in the mapping chain are generated. So, it usually makes sense for the intermediate component to receive one file for processing (in this case **ReportB.xml**) and forward the same file to the subsequent mapping, rather than look for a different file name. Be aware that, not having the same input and output file names on the intermediate component (when the pass-through button is inactive) might cause errors such as "The system cannot find the file specified" in generated code or in MapForce Server execution.

If you click the preview button on the third component (**ReportC**), and attempt to preview the mapping in MapForce, you will notice that an execution error occurs. This is expected, since, according to the settings above, a file called **ReportB.xml** is expected as input. However, the mapping did not produce yet such a file (because the pass-through button is not active, and only the portion of the mapping from B to C is executed). You can easily fix this problem as follows:

1. Click the preview button on the intermediate component.
2. Click the **Output** tab to preview the mapping.
3. Save the resulting output file as **ReportB.xml**, in the same folder as the mapping (**<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial**).

Now, if you click again the preview button on the third component (**ReportC**), the error is no longer shown.

When the pass-through button is inactive, you can also preview the StyleVision-generated output for each component that has an associated StyleVision Power StyleSheet (SPS) file. In particular, you can view the HTML version of the intermediate report as well (in addition to that of the final report):



HTML output of the intermediate component

8.2 Mapping Node Names

Most of the time when you create a mapping with MapForce, the goal is to read *values* from a source and write *values* to a target. However, there might be cases when you want to access not only the node *values* from the source, but also the node names. For example, you might want to create a mapping which reads the element or attribute names (not values) from a source XML and converts them to element or attribute values (not names) in a target XML.

Consider the following example: you have an XML file that contains a list of products. Each product has the following format:

```
<product>
   <id>1</id>
   <color>red</color>
   <size>10</size>
</product>
```

Your goal is to convert information about each product into name-value pairs, for example:

```
<product>
   <attribute name="id" value="1" />
   <attribute name="color" value="red" />
   <attribute name="size" value="10" />
</product>
```

For such scenarios, you would need access to the node name from the mapping. With *dynamic* access to node names, you can perform data conversions such as the one above.

Note: You can also perform the transformation above by using the `node-name` 1024 and `static-node-name` 1026 core library functions. However, in this case, you need to know exactly what element names you expect from the source, and you need to connect every single such element manually to the target. Also, these functions might not be sufficient, for example, when you need to filter or group nodes by name, or when you need to manipulate the data type of the node from the mapping.

Accessing node names dynamically is possible not only when you need to read node names, but also when you need to write them. In a standard mapping, the name of attributes or elements in a target is always known before the mapping runs; it comes from the underlying schema of the component. With dynamic node names, however, you can create new attributes or elements whose name is not known before the mapping runs. Specifically, the name of the attribute or element is supplied by the mapping itself, from any source supported by MapForce.

For dynamic access to a node's children elements or attributes to be possible, the node must actually have children elements or attributes, and it must not be the XML root node.

Dynamic node names are supported when you map to or from the following component types:

- XML
- CSV/FLF*

* Requires MapForce Professional or Enterprise Edition.

Note: In case of CSV/FLF, dynamic access implies access to "fields" instead of "nodes", since CSV/FLF structures do not have "nodes".

> When the mapping target is a CSV or FLF (fixed-length field) file, the fields must be defined in the component settings (and it is not possible to change the name, order, or number of the target fields). Unlike XML, the format of text files is fixed, so only the actual field value can be manipulated, not the field name, number or order.

Dynamic node names are supported in any of the following mapping languages: Built-In*, XSLT 2.0, XSLT 3.0, XQuery*, C#*, C++*, Java*.

* These languages require MapForce Professional or Enterprise Edition.

For information about how dynamic node names work, see Getting Access to Node Names 1256. For a step-by-step mapping example, see Example: Map Element Names to Attribute Values 1267.

8.2.1 Getting Access to Node Names

When a node in an XML component (or a field in a CSV/FLF component) has children nodes, you can get both the name and value of each child node directly on the mapping. This technique is called "dynamic node names". "Dynamic" refers to the fact that processing takes place "on the fly", during mapping runtime, and not based on the static schema information which is known before the mapping runs. This topic provides details on how to enable dynamic access to node names and what you can do with it.

When you read data from a source, "dynamic node names" means that you can do the following:

- Get a list of all children nodes (or attributes) of a node, as a sequence. In MapForce, "sequence" is a list of zero or more items which you can connect to a target and create as many items in the target as there are items in the source. So, for example, if a node has five attributes in the source, you could create five new elements in the target, each corresponding to an attribute.
- Read not only the children node values (as a standard mapping does), but also their names.

When you write data to a target, "dynamic node names" means that you can do the following:

- Create new nodes using names supplied by the mapping (so-called "dynamic" names), as opposed to names supplied by the component settings (so-called "static" names).

To illustrate dynamic node names, this topic makes use of the following XML schema: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Products.xsd**. This schema is accompanied by a sample instance document, **Products.xml**. To add both the schema and the instance file to the mapping area, select the **Insert | XML Schema/File** menu command and browse for **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Products.xml**. When prompted to select a root element, choose `products`.

To enable dynamic node names for the `product` node, right-click it and select one of the following context menu commands:

- **Show Attributes with Dynamic Name**, if you want to get access to the node's attributes
- **Show Child Elements with Dynamic Name**, if you want to get access to the node's child elements

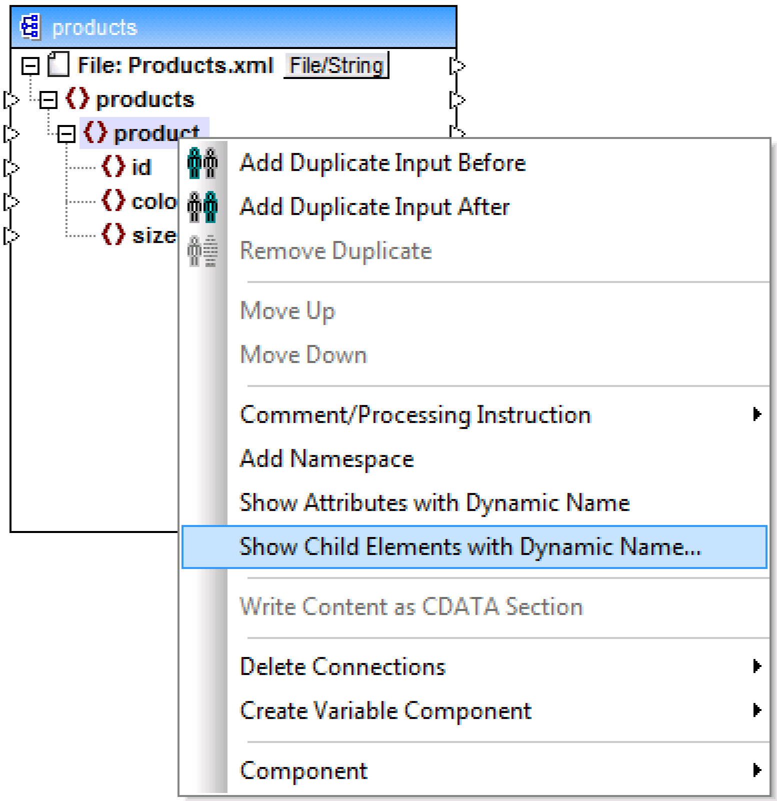


Fig. 1 Enabling dynamic node names (for child elements)

Note: The commands above are available only for nodes that have children nodes. Also, the commands are not available for root nodes.

When you switch a node into dynamic mode, a dialog box such as the one below appears. For the purpose of this topic, set the options as shown below; these options are further discussed in Accessing Nodes of Specific Type 1264.

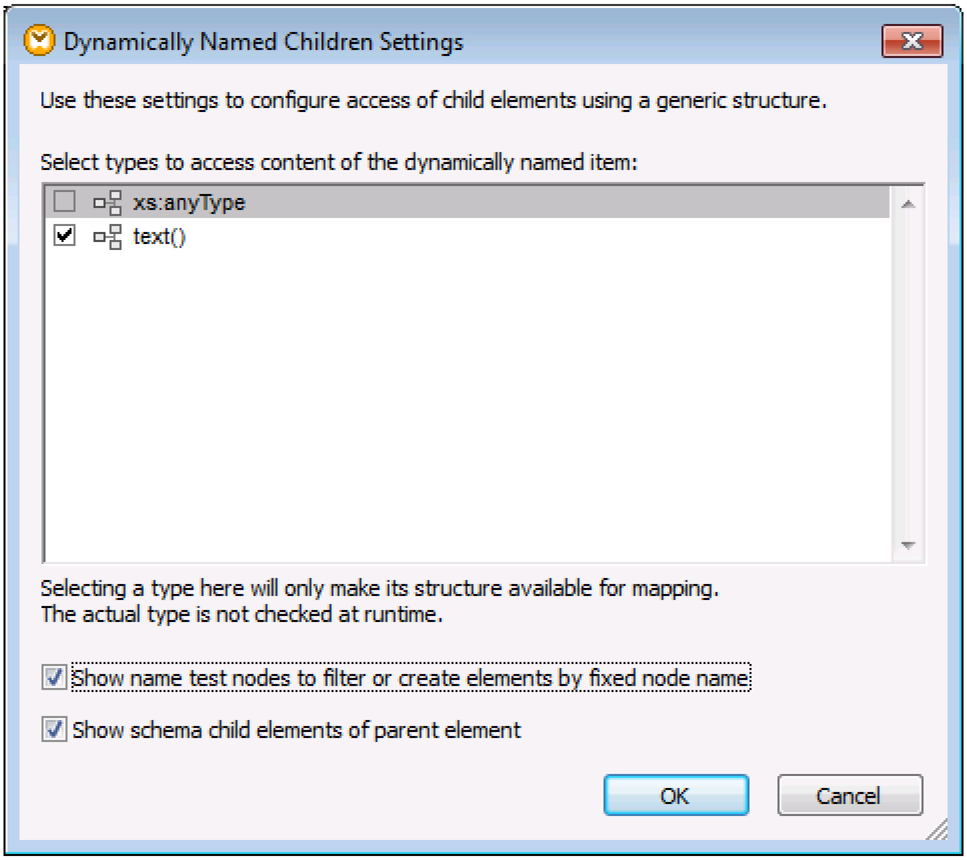


Fig. 2 "Dynamically Named Children Settings" dialog box

Fig. 3 illustrates how the component looks when dynamic node names are enabled for the `product` node. Notice how the appearance of the component has now significantly changed.

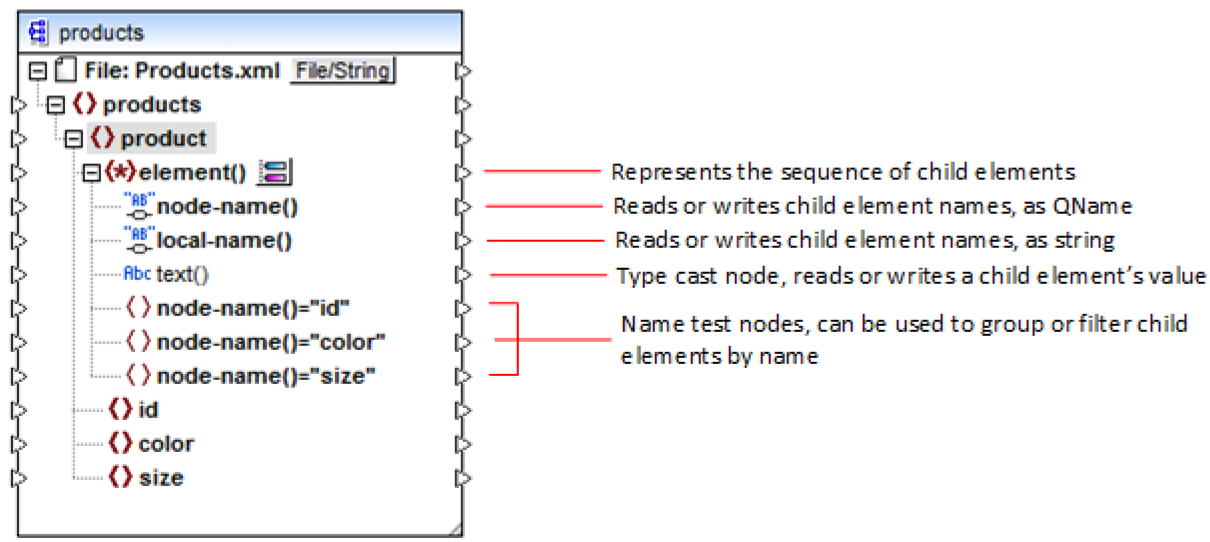


Fig.3 Enabled dynamic node names (for elements)

To switch the component back to standard mode, right-click the `product` node, and disable the option **Show Child Elements with Dynamic Name** from the context menu.

The image below shows how the same component looks when dynamic access to attributes of a node is enabled. The component was obtained by right-clicking the `product` element, and selecting **Show Attributes with Dynamic Name** from the context menu.

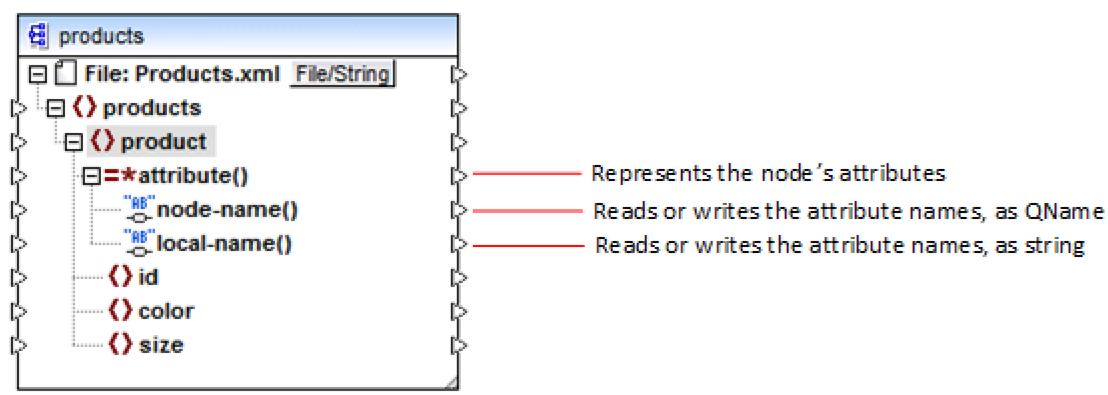


Fig. 4 Enabled dynamic node names (for attributes)

To switch the component back to standard mode, right-click the `product` node, and disable the option **Show Attributes with Dynamic Name** from the context menu.

As illustrated in Fig. 3 and Fig. 4, the component changes appearance when any node (in this case, `product`) is switched into "dynamic node name" mode. The new appearance opens possibilities for the following actions:

- Read or write a list of all children elements or attributes of a node. These are provided by the `element()` or `attribute()` item, respectively.

- Read or write the name of each child element or attribute. The name is provided by the `node-name()` and `local-name()` items.
- In case of elements, read or write the value of each child element, as specific data type. This value is provided by the type cast node (in this case, the `text()` item). Note that only elements can have type cast nodes. Attributes are treated always as "string" type.
- Group or filter child elements by name. For an example, see Example: Group and Filter Nodes by Name[1271].

The node types that you can work with in "dynamic node name" mode are described below.

element()

This node has different behaviour in a source component compared to a target component. In a source component, it supplies the child elements of the node, as a sequence. In Fig.3, `element()` provides a list (sequence) of all children elements of `product`. For example, the sequence created from the following XML would contain three items (since there are three child elements of `product`):

```
<product>
   <id>1</id>
   <color>red</color>
   <size>10</size>
</product>
```

Note that the actual name and type of each item in the sequence is provided by the `node-name()` node and the type cast node, respectively (discussed below). To understand this, imagine that you need to transform data from a source XML into a target XML as follows:

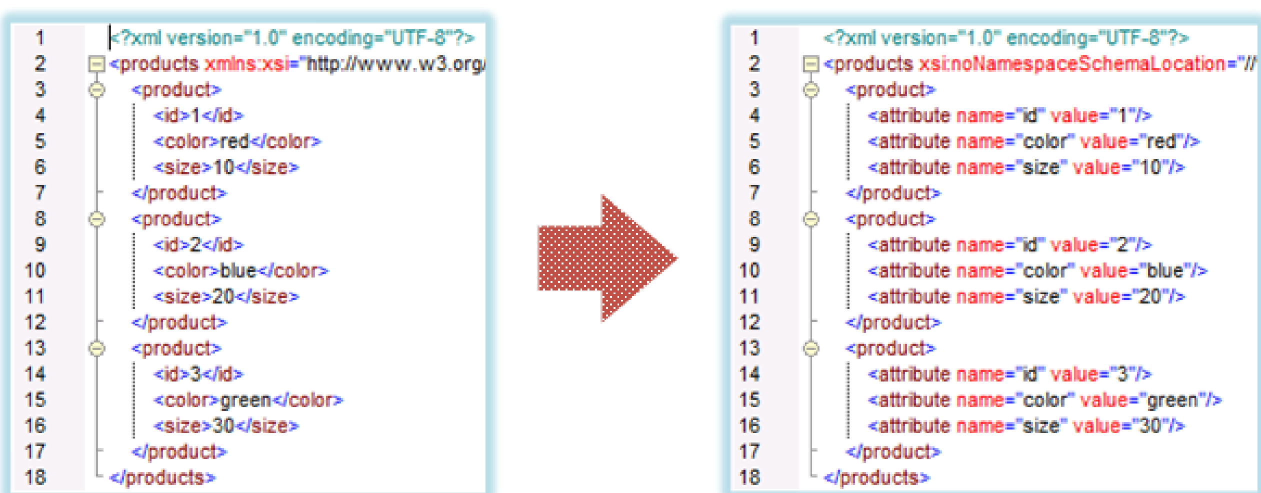


Fig. 6 Mapping XML element names to attribute values (requirement)

The mapping which would achieve this goal looks as follows:

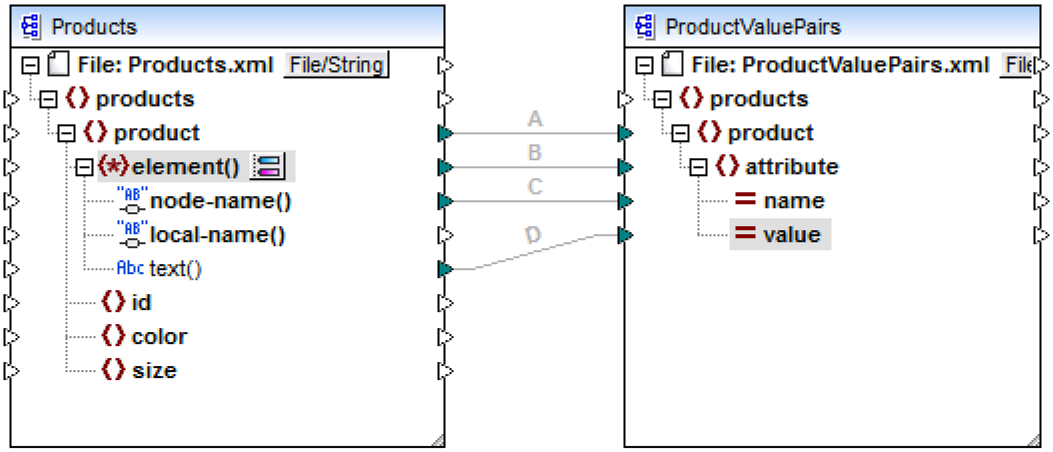


Fig. 7 Mapping XML element names to attribute values (in MapForce)

The role of `element()` here is to supply the sequence of child elements of `product`, while `node-name()` and `text()` supply the actual name and value of each item in the sequence. This mapping is accompanied by a tutorial sample and is discussed in more detail in [Example: Map Element Names to Attribute Values](1267).

In a target component, `element()` does not create anything by itself, which is an exception to the basic rule of mapping "for each item in the source, create one target item". The actual elements are created by the type cast nodes (using the value of `node-name()`) and by name test nodes (using their own name).

attribute()

As shown in Fig. 4, this item enables access to all attributes of the node, at mapping runtime. In a source component, it supplies the attributes of the connected source node, as a sequence. For example, in the following XML, the sequence would contain two items (since `product` has two attributes):

```
<product id="1" color="red" />
```

Note that the `attribute()` node supplies only the value of each attribute in the sequence, always as string type. The name of each attribute is supplied by the `node-name()` node.

In a target component, this node processes a connected sequence and creates an attribute value for each item in the sequence. The attribute name is supplied by the `node-name()`. For example, imagine that you need to transform data from a source XML into a target XML as follows:

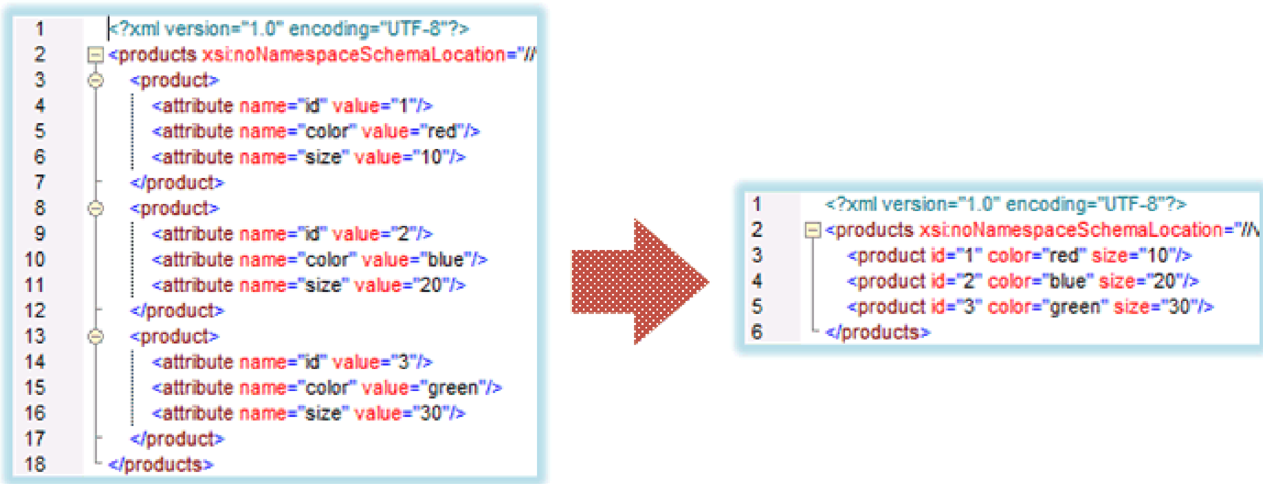


Fig. 8 Mapping attribute values to attribute names (requirement)

The mapping which would achieve this goal looks as follows:

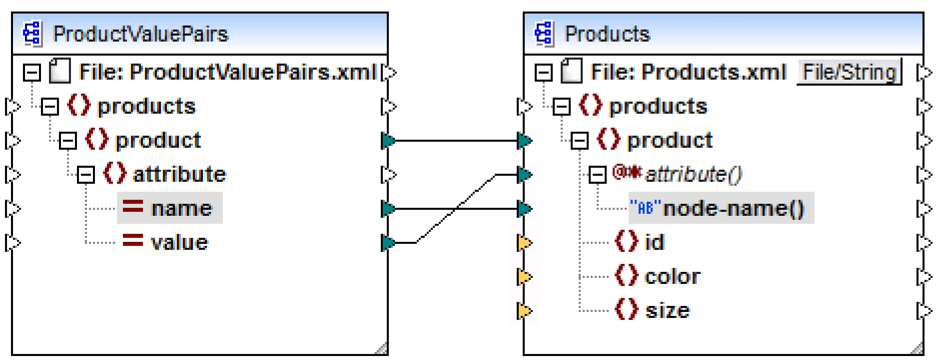


Fig. 9 Mapping attribute values to attribute names (in MapForce)

Note: This transformation is also possible without enabling dynamic access to a node's attributes. Here it just illustrates how `attribute()` works in a target component.

If you want to reconstruct this mapping, it uses the same XML components as the **ConvertProducts.mfd** mapping available in the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder. The only difference is that the target has now become the source, and the source has become the target. As input data for the source component, you will need an XML instance that actually contains attribute values, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
   <product>
      <attribute name="id" value="1"/>
      <attribute name="color" value="red"/>
      <attribute name="size" value="big"/>
```

```
   </product>
</products>
```

Note that, in the code listing above, the namespace and schema declaration have been omitted, for simplicity.

node-name()

In a source component, `node-name()` supplies the name of each child element of `element()`, or the name of each attribute of `attribute()`, respectively. By default, the supplied name is of type `xs:QName`. To get the name as string, use the `local-name()` node (see Fig. 3), or use the function QName-as-string[1114].

In a target component, `node-name()` writes the name of each element or attribute contained in `element()` or `attribute()`.

local-name()

This node works in the same way as `node-name()`, with the difference that the type is `xs:string` instead of `xs:QName`.

Type cast node

In a source component, the type cast node supplies the value of each child element contained in `element()`. The name and structure of this node depends on the type selected from the "Dynamically Named Children Settings" dialog box (Fig. 2).

To change the type of the node, click the **Change Selection** ( ) button and select a type from the list of available types, including a schema wildcard (`xs:any`). For more information, see Accessing nodes of specific type[1264].

In a target component, the type cast node writes the value of each child element contained in `element()`, as specific data type. Again, the desired data type can be selected by clicking the **Change Selection** ( ) button.

Name test nodes

In a source component, name test nodes provide a way to group or filter child elements from a source instance by name. You may need to filter child elements by name in order to ensure that the mapping accesses the instance data using the correct type (see Accessing Nodes of Specific Type[1264]). For an example, see Example: Group and Filter Nodes by Name[1271].

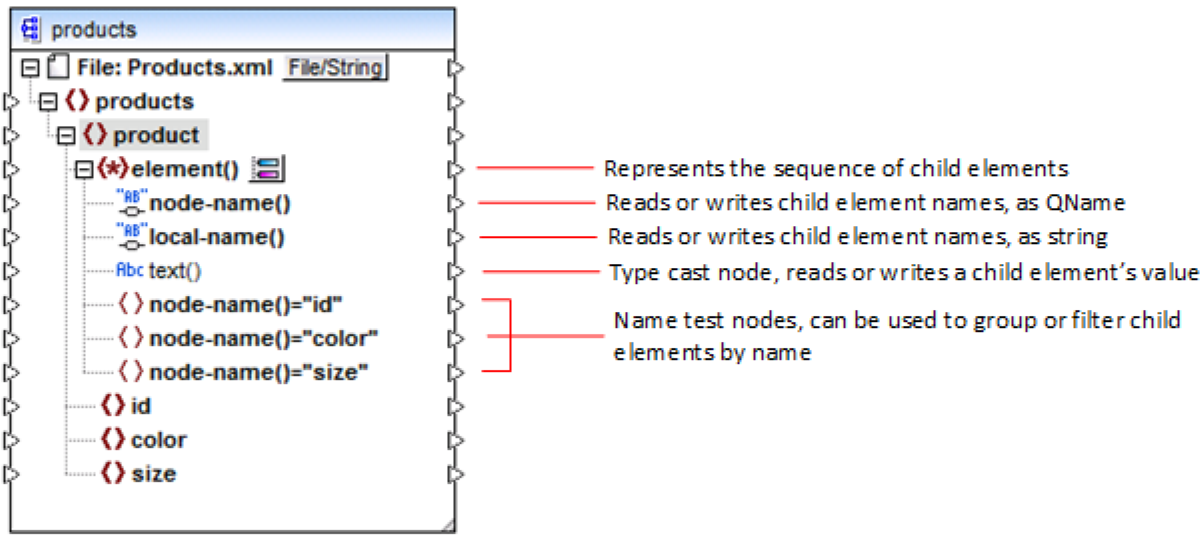
In general, the name test nodes work almost like normal element nodes for reading and writing values and subtree structures. However, because the mapping semantics is different when dynamic access is enabled, there are some limitations. For example, you cannot concatenate the value of two name test nodes.

On the target side, name test nodes create as many elements in the output as there are items in the connected source sequence. Their name overrides the value mapped to `node-name()`.

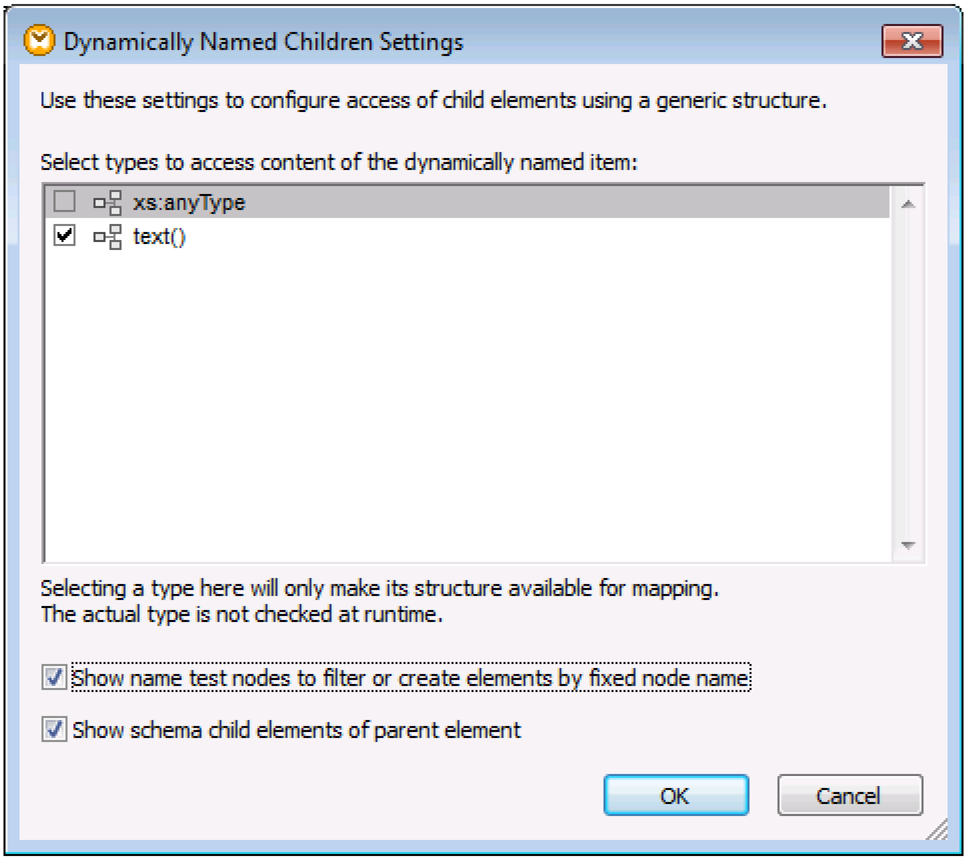
If necessary, you can hide the name test nodes from the component. To do this, click the **Change Selection** ( ) button next to the `element()` node. Then, in the "*Dynamically Named Children Settings*" dialog box (Fig. 2), clear the **Show name test nodes...** check box.

8.2.2 Accessing Nodes of Specific Type

As mentioned in the previous section, Getting Access to Node Names [1256], you can get access to all child elements of a node by right-clicking the node and selecting the **Show Child Elements with Dynamic Name** context menu command. At mapping runtime, this causes the name of each child element to be accessible through the `node-name()` node, while the value—through a special type cast node. In the image below, the type cast node is the `text()` node.



Importantly, the data type of each child element is not known before the mapping runtime. Besides, it may be different for each child element. For example, a `product` node in the XML instance file may have a child element `id` of type `xs:integer` and a child element `size` of type `xs:string`. To let you access the node content of a specific type, the dialog box shown below opens every time when you enable dynamic access to a node's child elements. You can also open this dialog box at any time later, by clicking the **Change Selection** ( ) button next to the `element()` node.



"Dynamically Named Children Settings" dialog box

To access the content of each child element at mapping runtime, you have several options:

1. Access the content as string. To do this, select the **text()** check box on the dialog box above. In this case, a `text()` node is created on the component when you close the dialog box. This option is suitable if the content is of simple type (`xs:int`, `xs:string`, etc.) and is illustrated in the Example: Map Element Names to Attribute Values 1267. Note that a **text()** node is displayed only if a child node of the current node can contain text.
2. Access the content as a particular complex type allowed by the schema. When custom complex types defined globally are allowed by the schema for the selected node, they are also available in the dialog box above, and you can select the check box next to them. In the image above, there are no complex types defined globally by the schema, so none are available for selection.
3. Access the content as any type. This may be useful in advanced mapping scenarios (see "Accessing deeper structures" below). To do this, select the check box next to **xs:anyType**.

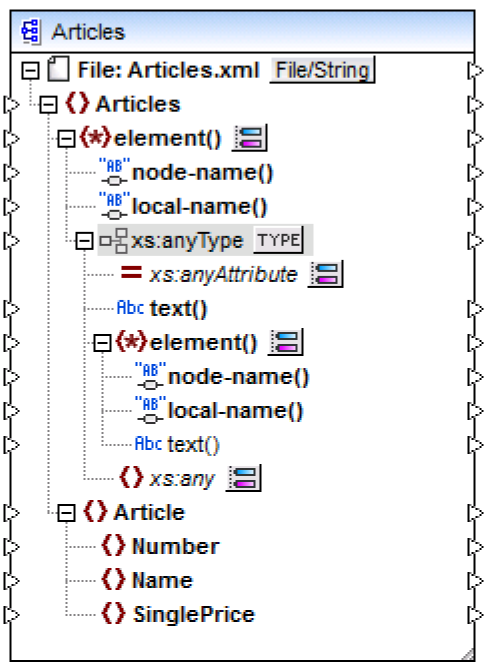
Be aware that, at mapping runtime, MapForce (through the type cast node) has no information as to what the actual type of the instance node is. Therefore, your mapping must access the node content using the correct type. For example, if you expect that the node of a source XML instance may have children nodes of various complex types, do the following:
a) Set the type cast node to be of the complex type that you need to match (see item 2 in the list above).

b) Add a filter to read from the instance only the complex type that you need to match. This technique is illustrated in Example: Group and Filter Nodes by Name[1271].

Accessing deeper structures

It is possible to access nodes at deeper levels in the schema than the immediate children of a node. It is useful for advanced mapping scenarios. In simple mappings such as Example: Map Element Names to Attribute Values[1267], you don't need this technique because the mapping accesses only the immediate children of an XML node. However, if you need to access deeper structures dynamically, such as "grandchildren", "great-grandchildren", and so on, this is possible as shown below.

1. Create a new mapping.
2. On the Insert menu, click **Insert XML Schema/File** and browse for the XML instance file (in this example, the **Articles.xml** file from the **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial** folder).
3. Right-click the `Articles` node and select the **Show Child Elements with Dynamic Name** context command.
4. Select **xs:anyType** from the "Dynamically Named Children Settings" dialog box.
5. Right-click the `xs:anyType` node and select again the **Show Child Elements with Dynamic Name** context command.
6. Select **text()** from the "Dynamically Named Children Settings" dialog box.



In the component above, notice there are two `element()` nodes. The second `element()` node provides dynamic access to grandchildren of the <Articles> node in the **Articles.xml** instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<Articles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Articles.xsd">
```



```
   <Article>
      <Number>1</Number>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>
   <Article>
      <Number>2</Number>
      <Name>Socks</Name>
      <SinglePrice>2.30</SinglePrice>
   </Article>
   <Article>
      <Number>3</Number>
      <Name>Pants</Name>
      <SinglePrice>34</SinglePrice>
   </Article>
   <Article>
      <Number>4</Number>
      <Name>Jacket</Name>
      <SinglePrice>57.50</SinglePrice>
   </Article>
</Articles>
```

Articles.xml

For example, to get "grandchildren" element names (`Number`, `Name`, `SinglePrice`), you would draw a connection from the `local-name()` node under the second `element()` node to a target item. Likewise, to get "grandchildren" element values (`1`, `T-Shirt`, `25`), you would draw a connection from the `text()` node.

Although not applicable to this example, in real-life situations, you can further enable dynamic node names for any subsequent `xs:anyType` node, so as to reach even deeper levels.

Note the following:

- The TYPE button allows you to select any derived type from the current schema and display it in a separate node. This may only be useful if you need to map to or from derived schema types (see Derived XML Schema Types [127]).
- The **Change Selection** ( ) button next to an `element()` node opens the "Dynamically Named Children Settings" dialog box discussed in this topic.
- The **Change Selection** ( ) button next to `xs:anyAttribute` allows you to select any attribute defined globally in the schema. Likewise, the **Change Selection** ( ) button next to `xs:any` element allows you to select any element defined globally in the schema. This works in the same way as mapping to or from schema wildcards (see also Wildcards - xs:any / xs:anyAttribute [134]). If using this option, make sure that the selected attribute or element can actually exist at that particular level according to the schema.

8.2.3 Example: Map Element Names to Attribute Values

This example shows you how to map element names from an XML document to attribute values in a target XML document. The example is accompanied by a sample mapping, which is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ConvertProducts.mfd**.

To understand what the example does, let's assume you have an XML file that contains a list of products. Each product has the following format:

```
<product>
   <id>1</id>
   <color>red</color>
   <size>10</size>
</product>
```

Your goal is to convert information about each product into name-value pairs, for example:

```
<product>
   <attribute name="id" value="1" />
   <attribute name="color" value="red" />
   <attribute name="size" value="10" />
</product>
```

To perform a data mapping such as the one above with minimum effort, this example uses a MapForce feature known as "dynamic access to node names". "Dynamic" means that, when the mapping runs, it can read the node names (not just values) and use these names as values. You can create the required mapping in a few simple steps, as shown below.

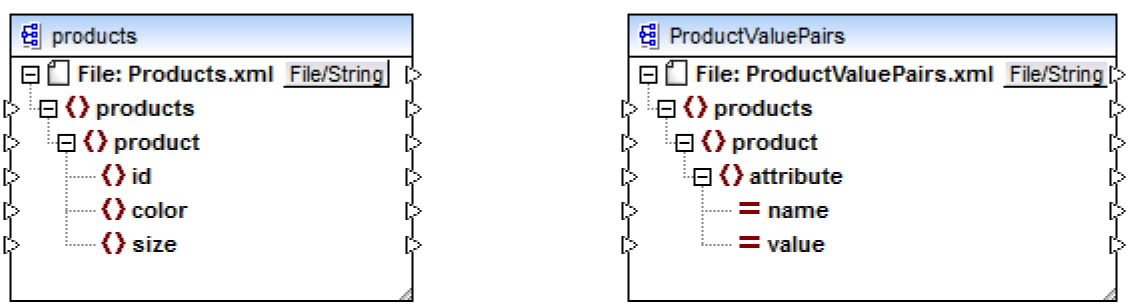
Step 1: Add the source XML component to the mapping

- On the **Insert** menu, click **XML Schema/File**, and browse for the following file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Products.xml**. This XML file points to the **Products.xsd** schema located in the same folder.

Step 2: Add the target XML component to the mapping

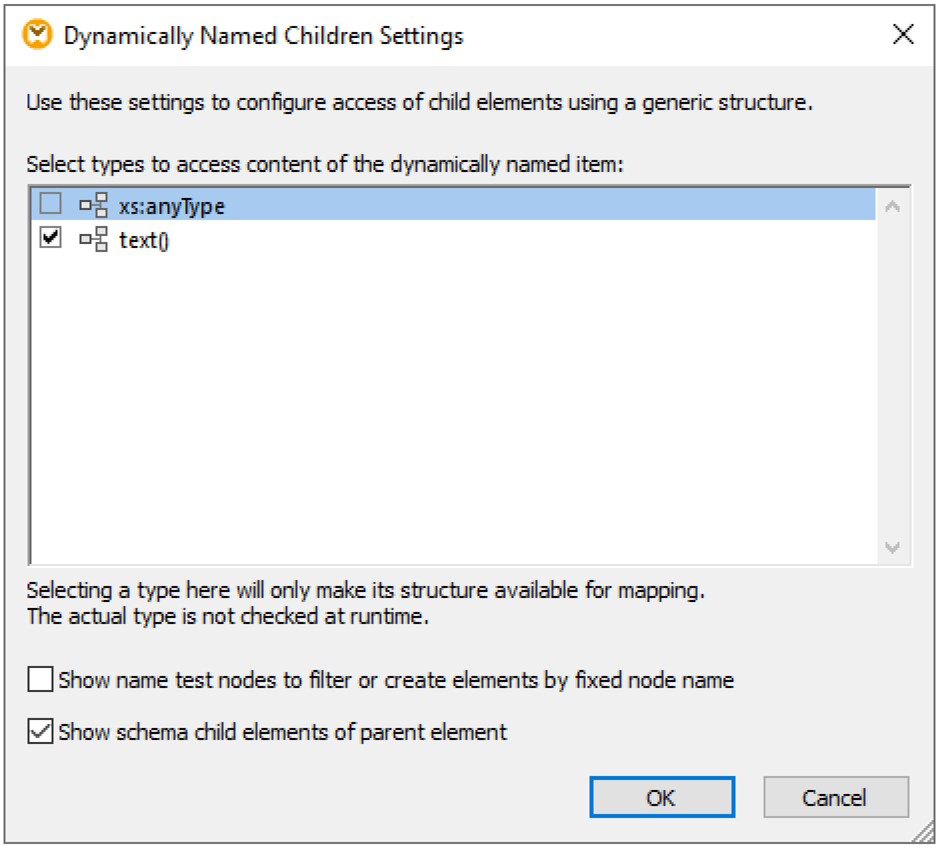
- On the **Insert** menu, click **XML Schema/File**, and browse for the following schema file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ProductValuePairs.xsd**. When prompted to supply an instance file, click **Skip**. When prompted to select a root element, select `products` as root element.

At this stage, the mapping should look as follows:

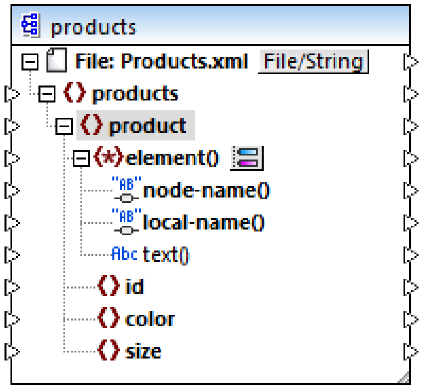


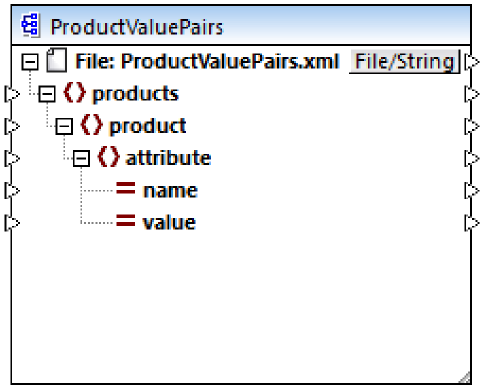
Step 3: Enable dynamic access to child nodes

1. Right-click the `product` node on the source component, and select **Show Child Elements with Dynamic Name** from the context menu.
2. In the dialog box which opens, select **text()** as type. Leave other options as is.



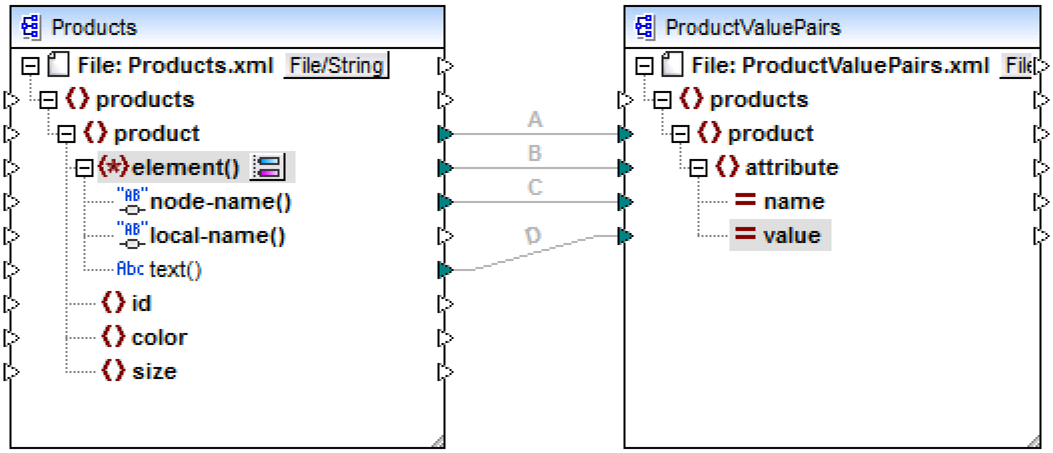
Notice that a `text()` node has been added on the source component. This node will supply the content of each child item to the mapping (in this case, the value of "id", "color", and "size").





Step 4: Draw the mapping connections

Finally, draw the mapping connections A, B, C, D as illustrated below. Optionally, double-click each connection, starting from the top one, and enter the text "A", "B", "C", and "D", respectively, into the Description box.



ConvertProducts.mfd

In the mapping illustrated above, connection A creates, for each product in the source, a product in the target. So far, this is a standard MapForce connection that does not address the node names in any way. The connection B, however, creates, for each encountered child element of `product`, a new element in the target called `attribute`.

> Connection B is a crucial connection in the mapping. To reiterate the goal of this connection, it carries a *sequence* of child elements of `product` from the source to the target. It does not carry the actual *names* or *values*. Therefore, it must be understood as follows: if the source **element()** has N child elements, create N instances of that item in the target. In this particular case, `product` in the source has three children elements (`id`, `color` and `size`). This means that each `product` in the target will have three child elements with the name `attribute`.
>
> Although not illustrated in this example, the same rule is used to map child elements of **attribute()**: if the source **attribute()** item has N child attributes, create N instances of that item in the target.

Next, connection C copies the actual name of each child element of `product` to the target (literally, "id", "color", and "size").

Finally, connection D copies the value of each child element of product, as string type, to the target.

To preview the mapping output, click the **Output** tab and observe the generated XML. As expected, the output contains several products whose data is stored as name-value pairs, which was the intended goal of this mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
<products xsi:noNamespaceSchemaLocation="ProductValuePairs.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <product>
      <attribute name="id" value="1"/>
      <attribute name="color" value="red"/>
      <attribute name="size" value="10"/>
   </product>
   <product>
      <attribute name="id" value="2"/>
      <attribute name="color" value="blue"/>
      <attribute name="size" value="20"/>
   </product>
   <product>
      <attribute name="id" value="3"/>
      <attribute name="color" value="green"/>
      <attribute name="size" value="30"/>
   </product>
</products>
```

Generated mapping output

8.2.4 Example: Group and Filter Nodes by Name

This example shows you how to design a mapping that reads key-value pairs from an XML property list (or XML plist) and writes them to a CSV file. (XML property lists represent a way of storing macOS and iOS object information in XML format, see https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/PropertyLists/UnderstandXMLPlist/UnderstandXMLPlist.html.) The example is accompanied by a mapping sample which is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ReadPropertyList.mfd**.

The code listing below represents the source XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "https://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
   <dict>
      <key>First Name</key>
      <string>William</string>
      <key>Last Name</key>
      <string>Shakespeare</string>
      <key>Birthdate</key>
      <integer>1564</integer>
      <key>Profession</key>
      <string>Playwright</string>
      <key>Lines</key>
      <array>
         <string>It is a tale told by an idiot,</string>
         <string>Full of sound and fury, signifying nothing.</string>
      </array>
   </dict>
</plist>
```

The goal of the mapping is to create a new line in the CSV file from certain key-value pairs found under `<dict>` node in the property list file. Specifically, the mapping must filter only **<key>** - **<string>** pairs. Other key-value pairs (for example, **<key>** - **<integer>**) must be ignored. In the CSV file, the line must store the name of the property, separated from the value of the property by a comma. In other words, the output must look as follows:

```
First Name,William
Last Name,Shakespeare
Profession,Playwright
```

To achieve this goal, the mapping uses dynamic access to all children nodes of the `dict` node. Secondly, the mapping uses the group-starting-with [1043] function to group the key-value pairs retrieved from the XML file. Finally, the mapping uses a filter to filter only those nodes where the node name is "string".

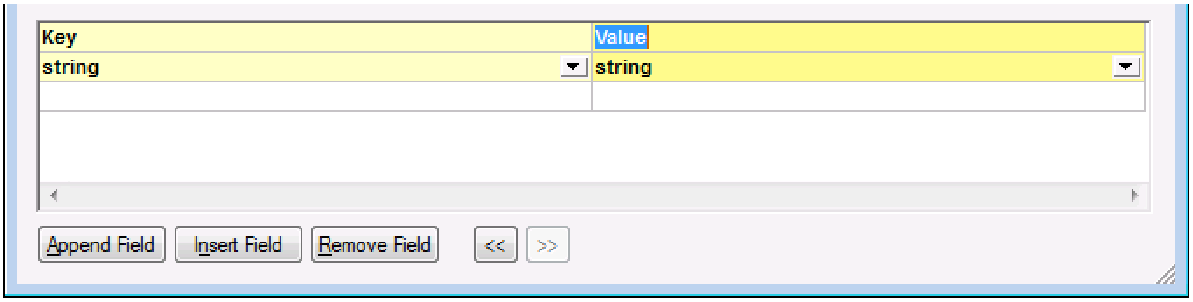
The following steps show how the required mapping can be created.

Step 1: Add the source XML component to the mapping

1. Set the mapping transformation language to BUILT-IN [27].
2. On the **Insert** menu, click **XML Schema/File**, and browse for the following file: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\plist.xml**. This XML file points to the **plist.dtd** schema located in the same folder.

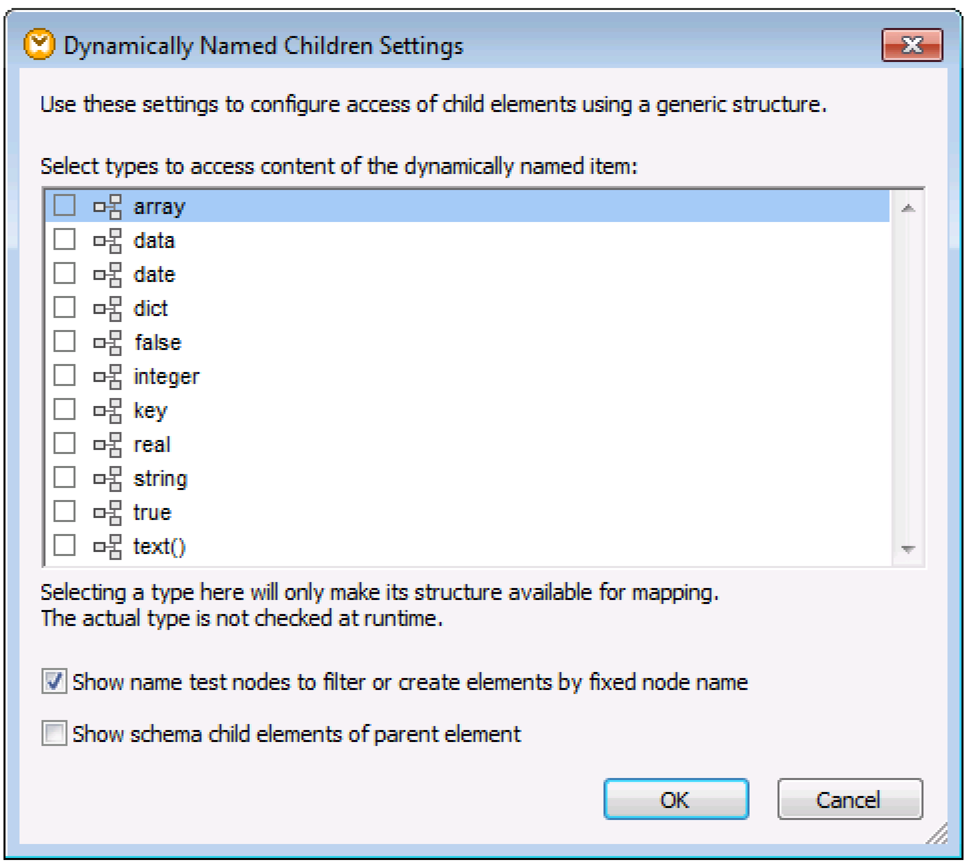
Step 2: Add the target CSV component to the mapping

1. On the **Insert** menu, click **Text File**. When prompted, select the **Use simple processing for standard CSV...** option.
2. Add a CSV field to the component, by clicking **Append field**.
3. Double-click the name of each field, and enter "Key" as name of the first field, and "Value" as name of the second field. The "Key" field will store the name of the property, while the "Value" field will store the property value. For more information about CSV components, see CSV and Text Files [563].

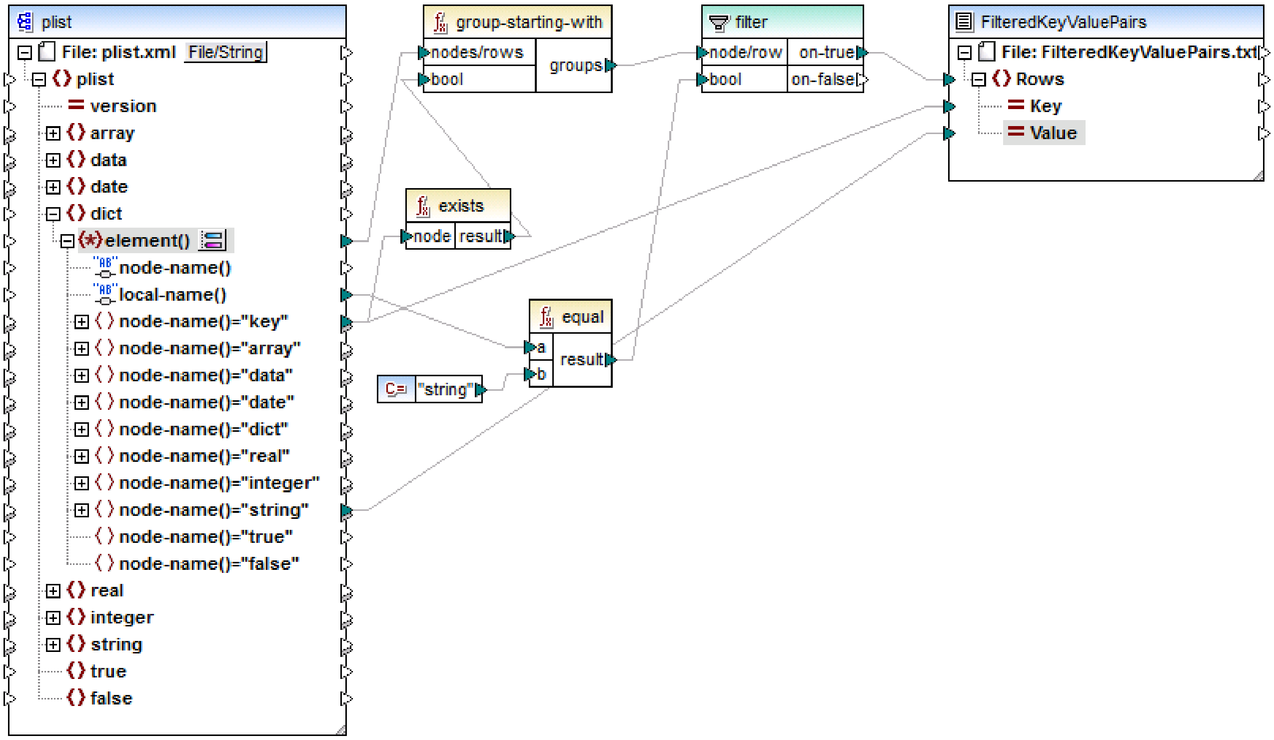


Step 3: Add the filter and functions

1. Drag the **`equal`**[1010], **`exists`**[1032] and **`group-starting-with`**[1043] functions from the Libraries window into the mapping. For general information about functions, see Functions[904].
2. To add the filter, click the **Insert** menu, and then click **Filter: Nodes/Rows**. For general information about filters, see Filters and Conditions[878].
3. On the **Insert** menu, click **Constant**, and then enter the text "string".
4. In the source component, right-click the `dict` node and select **Show Child Elements with Dynamic Name** from the context menu. On the "Dynamically Named Children Settings" dialog box, make sure that the check box **Show name test nodes to filter or create elements by fixed node name** is selected.



5. Draw the connections as shown below.



ReadPropertyList.mfd

The mapping explained

The `element()` item on the source component provides all children of the `dict` node, as a sequence, to the **group-starting-with** function. The **group-starting-with** function creates a new group whenever a node with the name `key` is encountered. The **exists** function checks for this condition and returns the result as Boolean true/false to the grouping function.

For each group, the filter checks if the name of the current node is equal to "string", with the help of the **equal** function. The name itself is read from the `local-name()`, which supplies the node's name as a string.

The connections to the target component have the following role:

- Only when the filter condition is true, a new row is created in the target CSV.
- `Key` (property name) is taken from the value of the `key` element in the source.
- `Value` (property value) is taken from the `string` name test node.

8.3 Mapping Rules and Strategies

In general, MapForce maps data in an intuitive way, but you may come across situations where the output contains too many or too few items. This chapter is meant to help you avoid situations when the mapping produces undesired output due to incorrect connections or mapping context.

Mapping rules

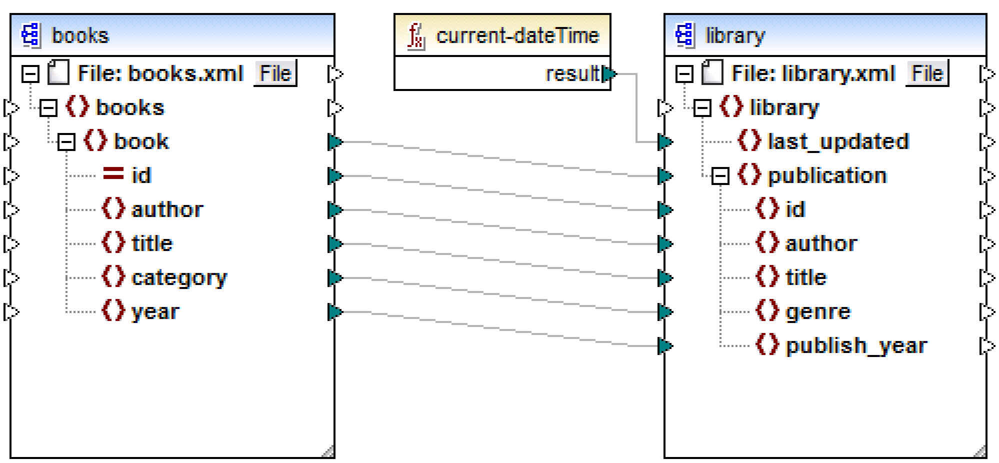
In order to be valid, a mapping must include at least one source and at least one target component. A source component is one that reads data, typically from a file or database. A target component is one that writes data, typically to a file or database. If you attempt to save a mapping where the above is not true, an error appears in the Message window: "A mapping requires at least two connected structures, for a example, a schema or a database structure".

To create a data mapping, you draw mapping connections between items in the source and target components.

All mapping connections that you draw make together a mapping algorithm. At mapping runtime, MapForce evaluates the algorithm and processes data based on it. The integrity and the efficiency of the mapping algorithm depends primarily on the connections. You can also tweak some settings at mapping [114] level, at component [82] level, or even at connection [92] level, but, essentially, the mapping *connections* determine how your data is processed.

Keep in mind the following rules when creating connections:

1. When you draw a connection *from* a source item, the mapping reads data associated with that item from the source file or database. The data may have zero, one, or multiple occurrences (in other words, it may be a sequence, as further described below). For example, if the mapping reads data from an XML file containing books, the source XML file may contain zero, one, or multiple **book** elements. In the mapping below, notice that the **book** item appears only once on the mapping component, even though the source (instance) file may contain multiple **book** elements, or none.



2. When you draw a connection *to* a target item, the mapping generates instance data of that kind. If the source item contains simple content (for example, string or integer) and if the target item accepts simple content, MapForce copies the content to the target item and, if necessary, converts the data type. Zero, one, or multiple values can be generated, depending on the incoming source data, see the next bullet.

3. For each (instance) item in the source, one (instance) item is created in the target. **This is the general mapping rule in MapForce.** Taking the mapping above as example, if the source XML contains three **book** elements, then three **publication** elements will be created on the target side. Note that there are also a few special cases, see Sequences 1276.
4. Each connection creates a *current mapping context*. The context determines which data is available *at the current moment, for the current target node*. The context, therefore, determines which source items are actually copied from the source to the target component. By drawing or omitting a connection, you may inadvertently change the current context and thus affect the output of the mapping. For example, your mapping might unnecessarily call a database or a Web service multiple times in the same mapping execution. This concept is further described below, see The mapping context 1277.

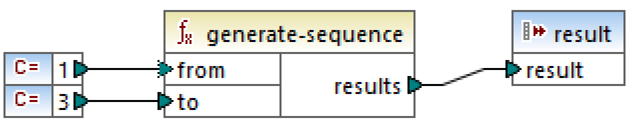
8.3.1 Sequences

As mentioned before, the general mapping rule is "for each item in the source, create one in the target". Here, "item" means one of the following:

- a single instance node of the input file or database
- a sequence of zero to multiple instance nodes of the input file or database

During mapping execution, if a sequence reaches a target item, this creates a loop that generates as many target nodes as there are source nodes. There are some exceptions to this rule, however:

- If the target item is an XML root element, it is created once and only once. If you connect a sequence to it, the result might not be schema valid. If attributes of the root element are also connected, the XML serialization will fail at mapping runtime. Therefore, avoid connecting a sequence to the root XML element.
- If the target item accepts only one value, it is created only once. Examples of items that accept only one value: XML attributes, database fields, simple output components. For example, the mapping below generates a sequence of three integers (1, 2, 3) with the help of the **generate-sequence** function. Nevertheless, the output will contain only one integer, because the target is a simple output component that accepts a single value. The other two values are ignored.



- If the source schema specifies that a specific item occurs only once, but the instance file contains many, MapForce may extract the first item from the source (which must exist according to the schema) and create only one item in the target. To disable this behavior, clear the check box **Enable input processing optimizations based on min/maxOccurs** from the component settings, see also XML Component Settings 123.

If the sequence is empty, nothing is generated on the target side. For example, if the target is an XML document and the source sequence is empty, no XML elements would be created in the target at all.

Functions work in a similar way: if they get a sequence as input, then they are called as many times as (and produce as many results as) there are items in the sequence.

If a function gets an empty sequence as input, it returns an empty result as well, and consequently produces no output at all.

However, there are some categories of functions that, by virtue of their design, return a value even if they get an empty sequence as input:

- **exists**, **not-exists**, **substitute-missing**
- **is-null**, **is-not-null**, **substitute-null** (these three functions are aliases of the previous three)
- aggregate functions (**sum**, **count**, and so on)
- user-defined functions that accept sequences and are regular (not inlined) functions

If you need to replace an empty value, add the **substitute-missing** function to the mapping and replace the empty value with a substitute value of choice. Alternatively, you can achieve the same result by using Defaults and Node Functions 912.

Functions may have multiple inputs. If a sequence is connected to each input, this produces a Cartesian product of all inputs, which is typically not the desired outcome. To avoid this, connect only one sequence to a function with multiple parameters; all other parameters must be connected to "singular" items from parents or other components.

8.3.2 The Mapping Context

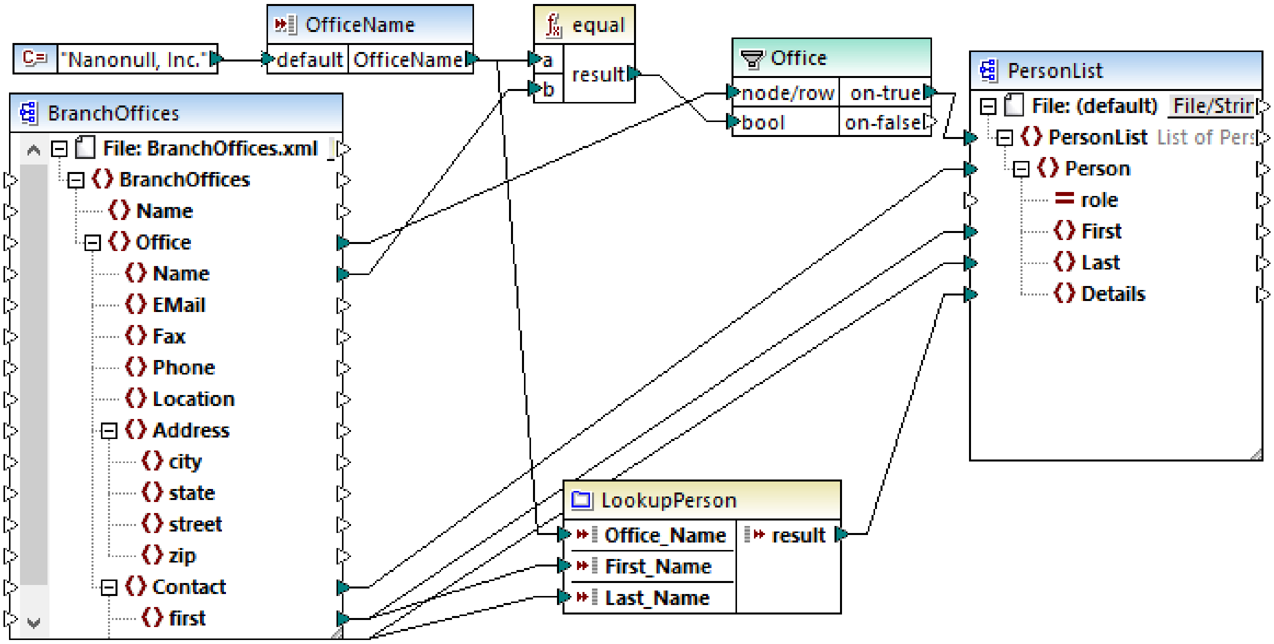
Mapping components are hierarchical structures that may contain many levels of depth. On the other hand, a mapping may have multiple source and components, plus any intermediary components such as functions, filters, value-maps, and so on. This adds complexity to the mapping algorithm, especially when multiple unrelated components are connected. To make it possible to execute the mapping in portions, one step at a time, a current context must be established for each connection.

We could also say that multiple "current contexts" are established for the duration of the mapping execution, since the current context changes with each processed connection.

MapForce always establishes the current context starting from the *target root item (node)*. This is where the mapping execution actually begins. The connection to the target root item is traced back to all source items that are directly or indirectly connected to it, including via functions or other intermediary components. All the source items and results produced by functions are added to the current context.

After it finishes processing the target node, MapForce works down the hierarchy. Namely, it processes all *mapped items* of the target component from top to bottom. For each new item, a new context is established that initially contains all items of the parent context. Thus, all mapped sibling items in a target component are independent of each other, but have access to all source data of their parent items.

Let's see how the above applies in practice, based on an example mapping, **PersonListByBranchOffice.mfd**. You can find this mapping in the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.



In the mapping above, both the source and the target component are XML. The source XML file contains two **Office** elements.

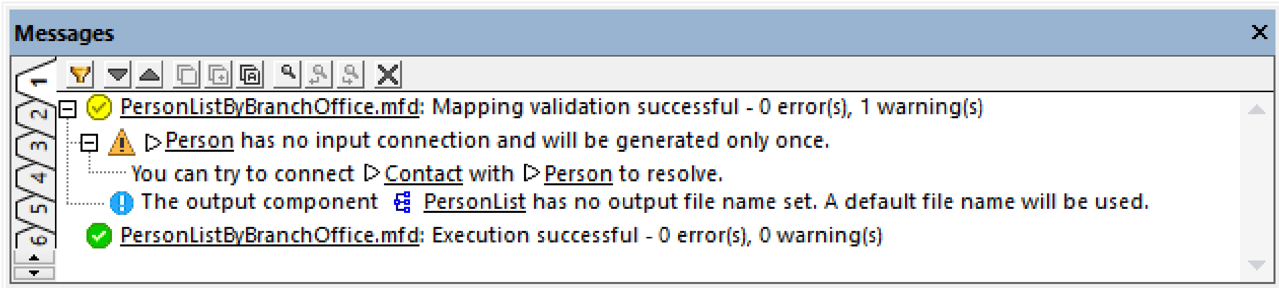
As mentioned previously, the mapping execution always begins from the target root node (**PersonList**, in this example). By tracing back the connection (via the filter and the function) to a source item, you can conclude that the source item is **Office**. (The other connection path leads to an input parameter and its purpose is further explained below).

Had there been a straightforward connection between **Office** and **PersonList**, then, according to the general mapping rule, the mapping would have created as many **PersonList** instance items as there are **Office** items in the source file. However, this does not happen here, because there is a filter in between. The filter supplies to the target component only data that satisfies the Boolean condition connected to the **bool** input of the filter. The **equal** function returns **true** if the office name is equal to "Nanonull, Inc.". This condition is satisfied only once, because there is only one such office name in the source XML file.

Consequently, the connection between **Office** and **PersonList** defines a single office as the context for the entire target document. This means that all descendants of the **PersonList** item have access to data of the office "Nanonull, Inc." office, and no other office exists in the current context.

The next connection is between **Contact** and **Person**. According to the general mapping rule, it will create one target **Person** for each source **Contact**. On each iteration, this connection establishes a new current context; therefore, the child connections (**first** to **First**, **last** to **Last**) supply data from the source to the target item in the context of each **Person**.

If you left out the connection between **Contact** and **Person**, then the mapping would create only one **Person** with multiple **First**, **Last**, and **Details** nodes. In such cases, MapForce issues a warning and a suggestion in the Messages window, for example:



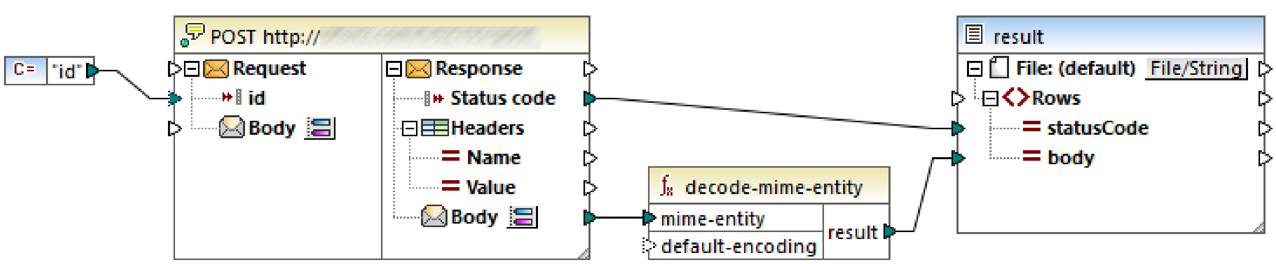
Finally, the mapping includes a user-defined function, **LookupPerson**. The user-defined function is also executed in the context of each **Person**, because of the parent connection between **Contact** and **Person**. Specifically, each time when a new Person item is created on the target side, the function is called to populate the **Details** element of the person. This function takes three input parameters. The first one (**OfficeName**) is set to read data from the input parameter of the mapping. The source data for this parameter could as well be provided by the **Name** source item, without changing in any way the mapping output. In either case, the source value is the same and it is taken from a parent context. Internally, the look-up function concatenates the values received as arguments and produces a single value. For more information about how the **LookupPerson** function works, see the Example: Look-up and Concatenation 939.

8.3.2.1 Web Services

In mappings that contain Web service calls, if the context allows it, the same Web service may be called multiple times, which you will normally want to avoid. Consider the following cases:

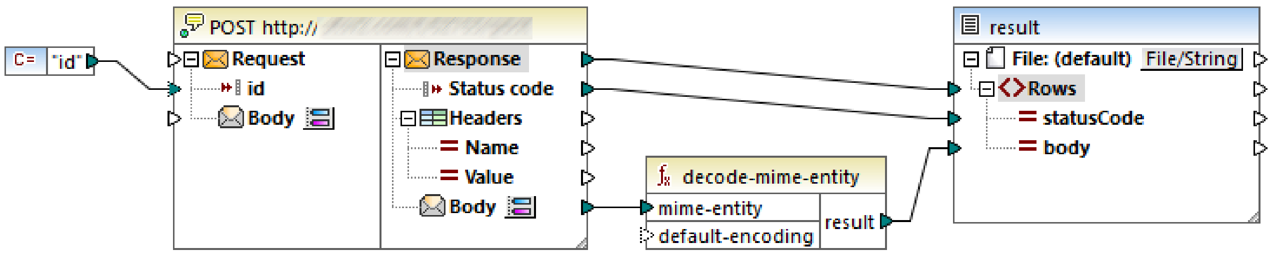
- If the connection to the **Request** item of the Web service component carries a *sequence of values*, then the Web service will be called multiple times, once for each item in the sequence.
- If a Web service is inside a user-defined function, and if the UDF gets called multiple times for the duration of the mapping execution, then the Web service will be called at least as many times. This can be avoided by configuring the UDF settings. For details, see User-Defined Functions 1281.
- You will typically want to connect the **Response** item of a Web service to a target item that:
  - o is expected to be created only once (for example, a simple output component, or the top item of some complex target component)
  - o is not in a parent context that demands multiple iterations of the item (and thus multiple Web service calls, potentially).

With respect to the third case above, it could lead to a Web service being called multiple times, even though the mapping output would be the one expected. For example, the Web service illustrated below is called twice because of the missing connection from **Response** to **Rows**.



As usual, mapping execution begins with the target root item (**Rows**, in this case). **Rows** does not have a connection, so MapForce proceeds down the hierarchy. To populate the **statusCode** target item, the first call to the Web service is made. Next, the mapping must populate the **body** item, so it makes the second call to the Web service. If there were more children of **Rows** connected to the Web service, a new call would occur for each.

The reason is that, in the mapping above, neither **statusCode** nor **body** have an explicit parent context (in other words, their parent is not connected to a source item). As a consequence, they cannot reuse any data from their parent context and must retrieve it individually from the source component, which in this case happens to be a Web service. This problem is easily solved by drawing a connection from **Response** to **Rows**, as shown below:



Thanks to the connection between **Response** and **Rows**, the Web service is now called only once. After the first call occurs, all the Web service data is in the current context and available to children of **Rows**, so no more redundant Web service calls take place.

8.3.2.2 Databases

To improve efficiency and decrease usage of hardware or network resources, you will typically want to avoid calling the same database multiple times in the same mapping unnecessarily. There may still be situations where you simply cannot avoid calling a database multiple times because of the nature of the mapping, but here are some general considerations:

- If you need only one database call, avoid placing the database component in a parent context that would demand calling the database multiple times. This could happen, for example, if you add a database component inside a user-defined function that receives a sequence of values as input and thus gets called for each item in the sequence, see also User-Defined Functions 1281 below. Variables are typically helpful to gather data into the same context before you pass it on to the target component.
- If you need to aggregate values from a database (for example, to count the number of records using the `count` function), it is recommended to connect the output of the aggregate function to a variable where **compute-when=once**. This prevents repetitive calls to the database, as described in the Example: Counting Database Table Rows 855.
- Try to extract all database data in one call (for example, a SQL-SELECT statement, or a stored procedure), as opposed to adding the same database component multiple times on the mapping.
- If you need to extract data from multiple tables or views from the same database, it's advisable to use either a Join component (in SQL mode), or a SQL-SELECT statement. The latter is more convenient if you prefer to write the SQL SELECT statement yourself. If you need to join database data to some non-database data, or data from different databases, use non-SQL joins. To optimize execution of non-SQL joins in data-intensive mappings, run mappings with MapForce Server Advanced Edition.

- If you need to filter data from a database, it's more efficient to use a SQL-WHERE component instead of a standard filter, since the former component is optimized for working with databases specifically, in the grammar of the respective database.

8.3.2.3 User-Defined Functions

User-defined functions (UDFs) are custom functions embedded into the mapping, where you define the inputs, outputs, and processing logic. Each user-defined function may contain the same component kinds as a main mapping, including Web services and databases.

> By default, if a UDF contains a database or a Web service component, and if the input data to the UDF is a sequence of multiple values, then each input value will call the UDF and consequently will result in a database or Web service call.

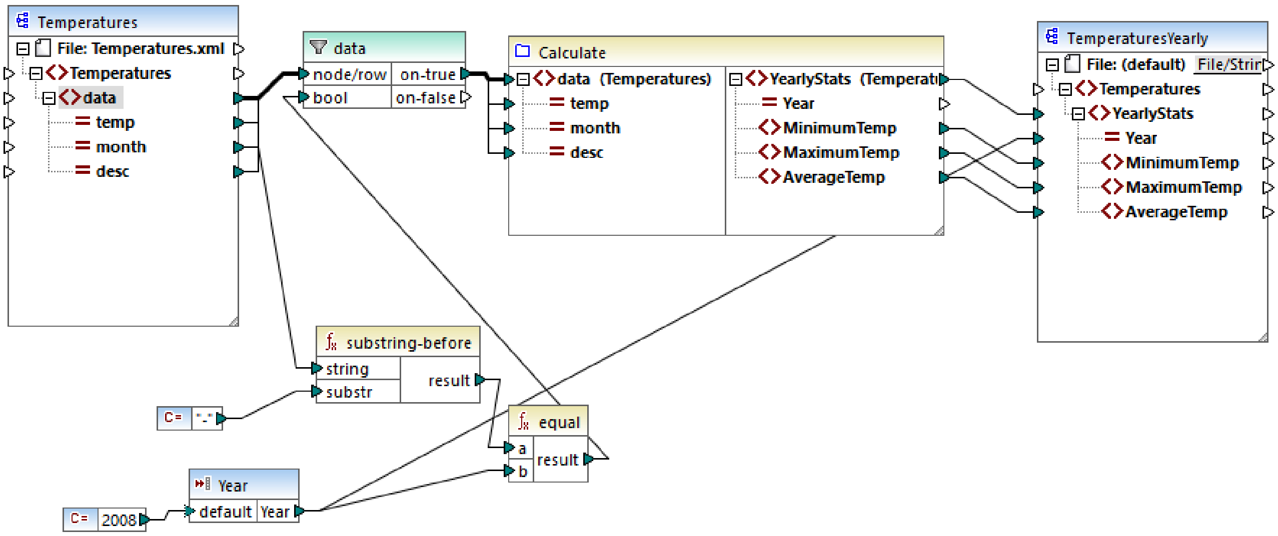
The behavior above may be acceptable for those mappings where you really need the UDF to be called *as many times as there are input values* and there is simply no other alternative way.

If you do not want the above to happen, you can configure the UDF so that it is called only once even if gets a sequence of values as input. You will typically want to do this for those UDFs that operate on a set of values before they can return (such as functions that calculate averages or totals).

Configuring a UDF to accept multiple input values in the same function call is possible if the UDF is of type "regular", not "inlined". (For details, see the User-Defined Functions 927 chapter.) With regular functions, you can specify that the input parameter is a sequence by selecting the **Input is a sequence** check box. This check box is visible on the component settings, after you double-click the title bar of an input parameter. The check box affects how often the function is called, as follows:

- When input data is connected to a **sequence** parameter, the user-defined function is called *only once* and the complete sequence is passed into the user-defined function.
- When input data is connected to a **non-sequence** parameter, the user-defined function is called *once for each single item in the sequence*.

For an example, open the following demo mapping:
<Documents>\Altova\MapForce2023\MapForceExamples\InputIsSequence.mfd.

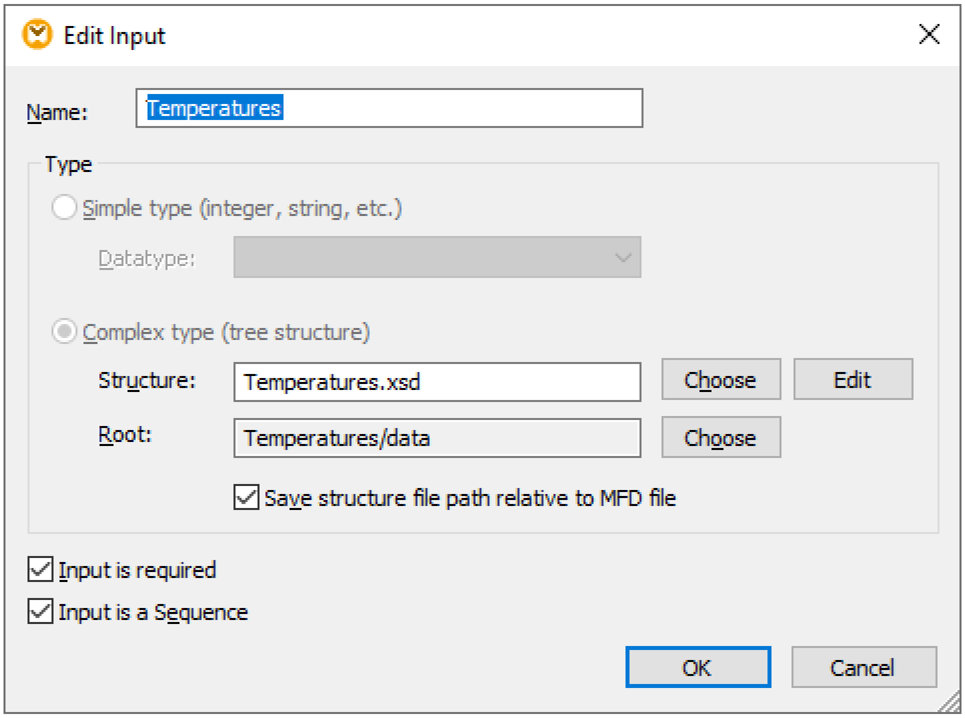


The mapping above illustrates a typical case of a UDF that operates on a set of values and thus requires all the input values in one call. Specifically, the **Calculate** user-defined function returns the minimum, maximum and average temperatures, taking as input data from a source XML file. The expected mapping output is as follows:

```
<Temperatures>
    <YearlyStats Year="2008">
        <MinimumTemp>-0.5</MinimumTemp>
        <MaximumTemp>24</MaximumTemp>
        <AverageTemp>11.6</AverageTemp>
    </YearlyStats>
</Temperatures>
```

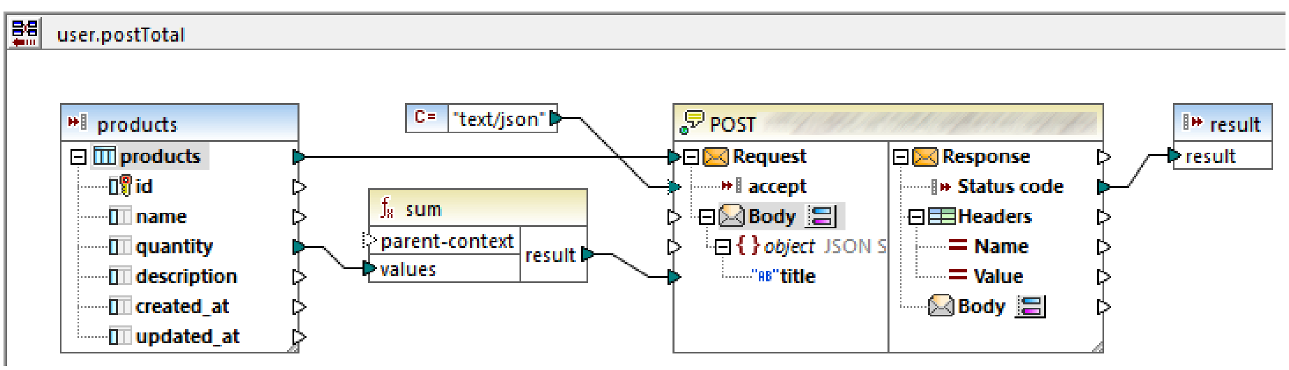
As usual, the mapping execution begins with the top item of the target component (**YearlyStats**, in this example). To populate this node, the mapping attempts to obtain source data from the UDF, which in its turn, triggers the filter. The filter's role in this mapping is to pass onto the UDF only temperatures from year 2008.

The check box **Input is sequence** was selected for the input parameter of the UDF (To view this check box, double-click the title bar of the **Calculate** function to enter the function's mapping; then double-click the title bar of the input parameter). As mentioned before, the **Input is sequence** option causes the complete sequence of values to be supplied as input to the function and the function is called only once.



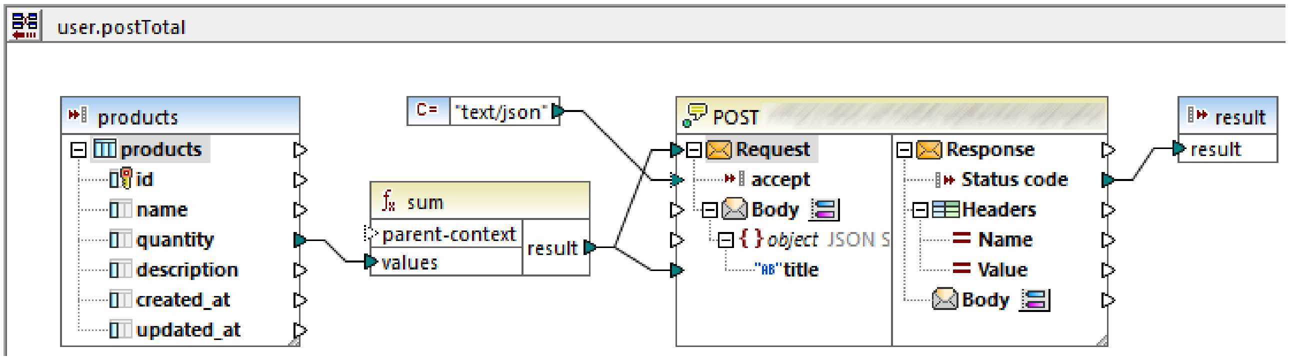
Had the **Input is sequence** check box not been selected, the UDF would have been called for each value in the source. As a result, the minimum, maximum and average would be calculated for each single value individually and incorrect output would be produced.

By applying the same logic in more complex UDFs that include database or Web service calls, it may be possible to optimize the execution and avoid unnecessary calls to the database or Web service. Nevertheless, be aware that the **Input is sequence** check box does not control what happens to the sequence of values *after* it enters the function. In other words, there is nothing to prevent you from connecting the incoming sequence of values to the input of a Web service and thus call it multiple times. Consider the following example:



The UDF illustrated above receives a sequence of values from the external mapping. Specifically, the data supplied to the input parameter originates from a database. The input parameter has the option **Input is sequence** selected, so the entire sequence is supplied to the function in one call. The function is supposed to add up multiple **quantity** values and post the result to a Web service. Exactly one Web service call is expected. However, the Web service will be incorrectly called multiple times when the mapping runs. The reason is that the **Request** input of the Web service receives *a sequence of values*, not a single value.

To fix this problem, connect the **Request** input of the Web service to the result of the **sum** function. The function produces one single value, so the Web service will also be called once:



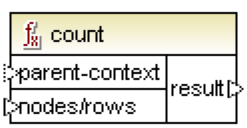
> Normally, aggregate functions like **sum**, **count**, etc produce a single value. Nevertheless, if there is a parent connection that allows it, they may produce a sequence of values as well, as described further in the Example: Changing the Parent Context[1284].

8.3.2.4 Example: Changing the Parent Context

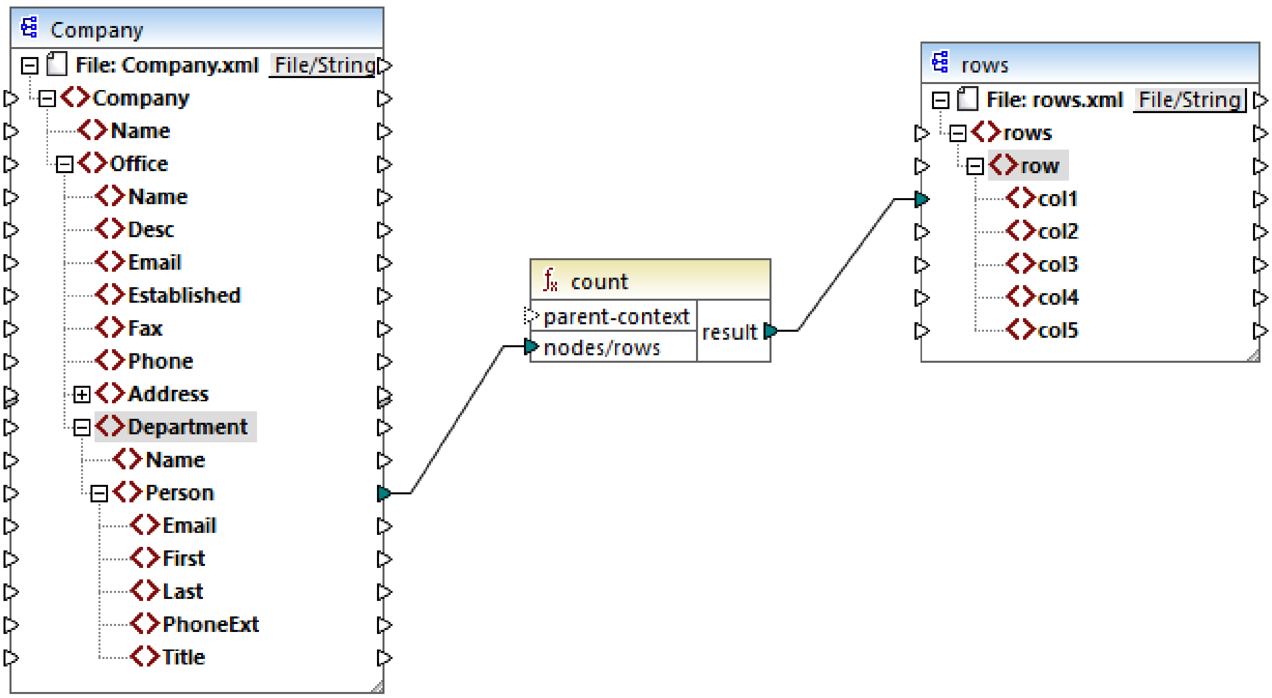
Some mapping components have an optional **parent-context** item.

The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **min**, **max**, **avg**, **count**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.

With the help of this item you can influence the mapping context in which that component should operate and consequently change the mapping output. The components that have an optional **parent-context** are: aggregate functions, variables, and Join components.



For a demo of how changing the parent context is useful, open the following mapping: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ParentContext.mfd**.



In the source XML of the mapping above, there is a single **Company** node which contains two **Office** nodes. Each **Office** node contains multiple **Department** nodes, and each **Department** contains multiple **Person** nodes. If you open the XML file in an XML editor, you can see that the distribution of people by office and department is as follows:

Office	Department	Number of people
Nanonull, Inc.	Administration	3
	Marketing	2
	Engineering	6
	IT & Technical Support	4
Nanonull Partners, Inc.	Administration	2
	Marketing	1
	IT & Technical Support	3

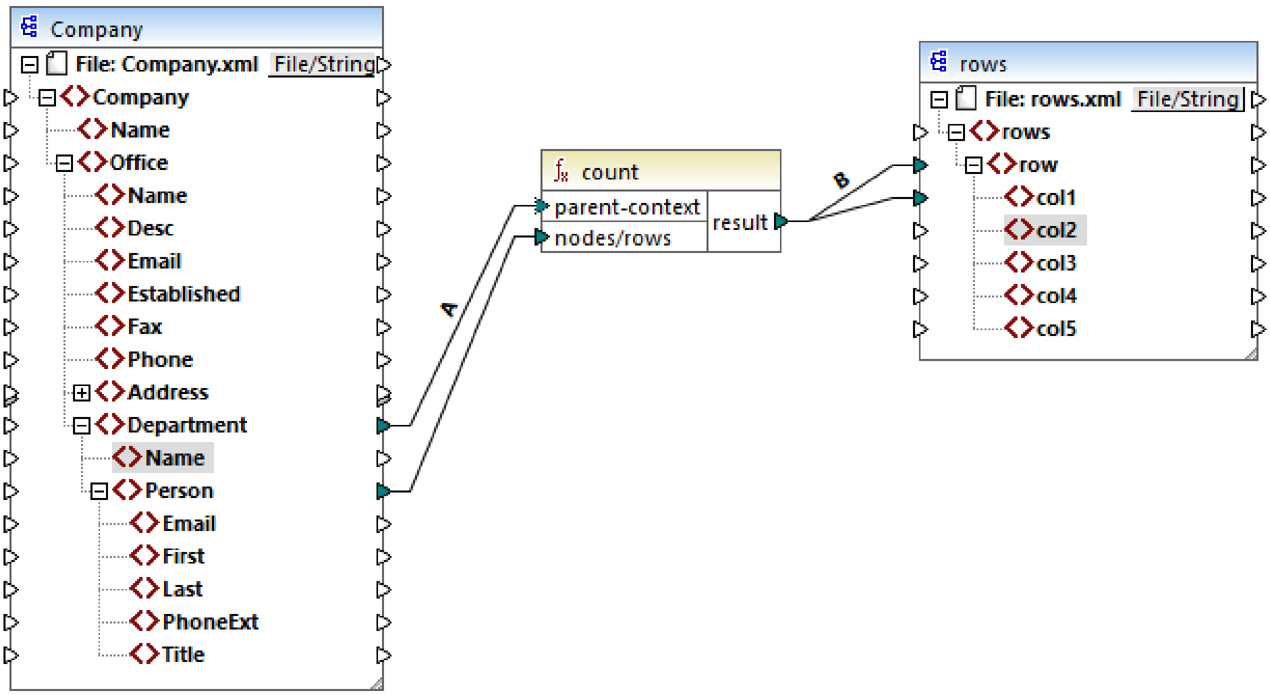
The mapping counts all people in all departments. For this purpose, it uses the **count** function from the `core` library. If you click the **Output** tab to preview the mapping, you will notice that it produces a single value, **21**, which corresponds to the total number of people in the source XML file.

The mapping works as follows:

- As usual, the mapping execution starts from the top node of the target component (**rows**, in this example). There is no incoming connection to **rows**. As a result, an implicit mapping context is established between **Company** (top item of the source component) and **rows** (top item of the target component).

- The function's result is a single value, because there is only one company in the source file.
- To populate the **col1** target item, MapForce executes the **count** function in the *implicit parent context* mentioned above, so it will count all **Person** nodes from all offices and from all departments.

The **parent-context** argument of the function lets you change the mapping context. This enables you, for example, to count the number of people in each department. To do this, draw two more connections as shown below:

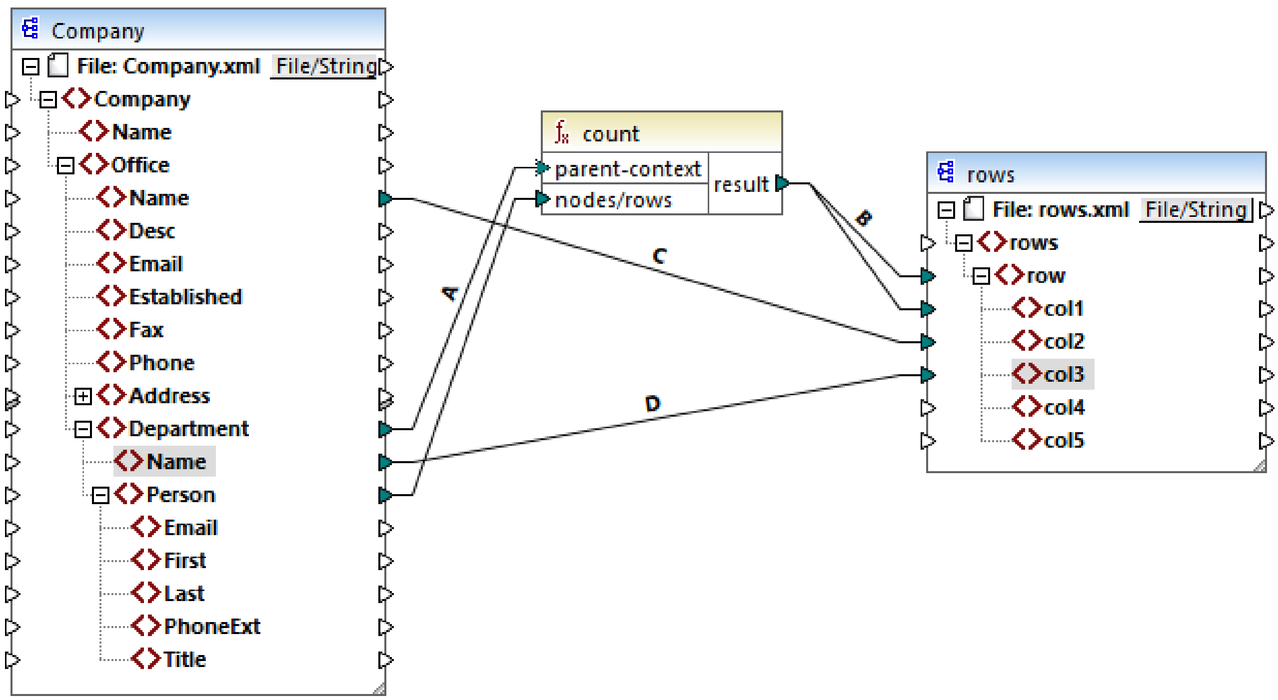


In the mapping above, connection A changes the parent context of the **count** function to **Department**. As a result, the function will count the number of people in each department. Very importantly, the function will now return a *sequence* of results instead of a single result, because multiple departments exist in the source. This is the reason why connection B exists: for each item in the resulting sequence it creates a new row in the target file. The mapping output has now changed accordingly (notice the numbers correspond exactly to the count of people in each department):

```
<rows>
    <row>
        <col1>3</col1>
    </row>
    <row>
        <col1>2</col1>
    </row>
    <row>
        <col1>6</col1>
    </row>
    <row>
```

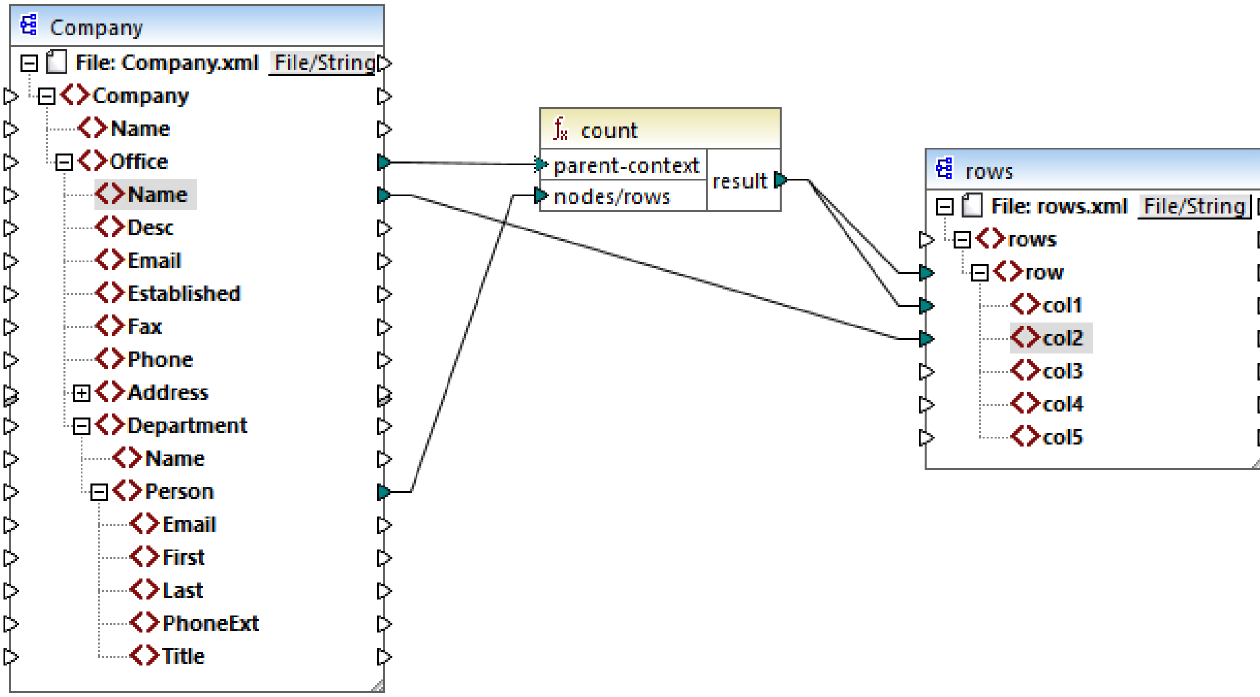
```
            <col1>4</col1>
        </row>
        <row>
            <col1>2</col1>
        </row>
        <row>
            <col1>1</col1>
        </row>
        <row>
            <col1>3</col1>
        </row>
</rows>
```

Given that the current mapping creates a row for each department, you can optionally copy the office name and the department name as well into the target file, by drawing connections C and D:



This way, the output will display not only the count of people but also the corresponding office and department name.

If you would like to count the number of people in each office, connect the parent context of **count** function to the **Office** item in the source.



With the connections shown above, the **count** function returns one result for each office. There are two offices in the source file, so the function will now return two sequences. Consequently, there will be two rows in the output, where each row is the number of people in that office:

```
<rows>
    <row>
        <col1>15</col1>
        <col2>Nanonull, Inc.</col2>
    </row>
    <row>
        <col1>6</col1>
        <col2>Nanonull Partners, Inc.</col2>
    </row>
</rows>
```

8.3.3 Priority context

Priority context is a way to influence the order in which input parameters of a function are evaluated. Setting a priority context may be necessary if your mapping joins data from two unrelated sources.

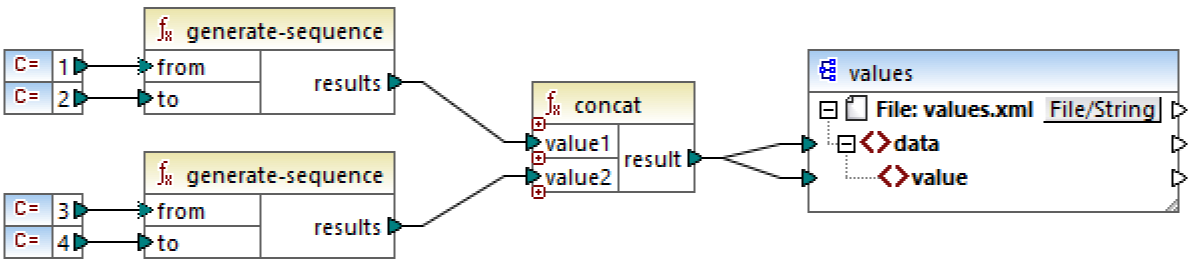
To understand how priority context works, recall that, when a mapping runs, the connection to an input item may carry a *sequence* of multiple values. For functions with two input parameters, this means that MapForce must create two loops, one of which must be processed first. The loop that is processed first is the "outer" loop. For example, the **equal** function receives two parameters: *a* and *b*. If both *a* and *b* get a sequence of values, then MapForce processes as follows:

- For each occurrence of *a*
  - For each occurrence of *b*
    - Is *a* equal to *b*?

As you can see from above, each *b* is evaluated in the context of each *a*. Priority context lets you alter the processing logic so that each *a* is evaluated in the context of each *b*. In other words, it lets you swap the inner loop with the outer loop, for example:

- For each occurrence of *b*
  - For each occurrence of *a*
    - Is *a* equal to *b*?

Let's now examine a mapping where priority context affects the mapping output. In the mapping below, the **`concat`** function has two input parameters. Each input parameter is a sequence that was generated with the help of the **`generate-sequence`** function. The first sequence is "1,2" and the second sequence is "3,4".



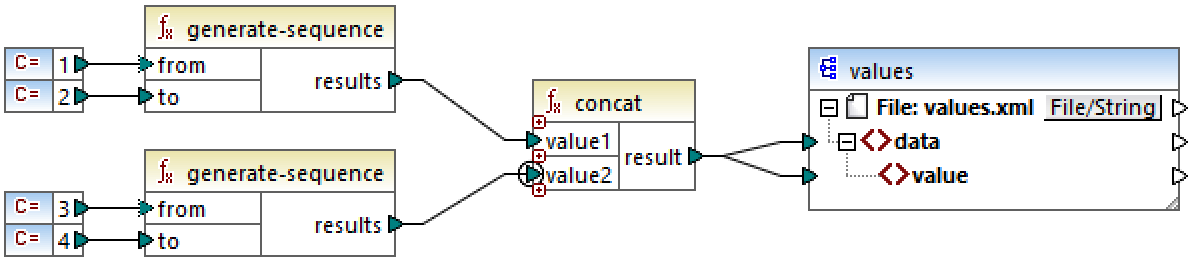
First, let's run the mapping without setting a priority context. The **`concat`** function starts evaluating the top sequence first, so it combines values in the following order:

- 1 with 3
- 1 with 4
- 2 with 3
- 2 with 4

This is reflected in the mapping output as well:

```
<data>
    <value>13</value>
    <value>14</value>
    <value>23</value>
    <value>24</value>
</data>
```

If you right-click the second input parameter and select **Priority Context** from the context menu, it will become the priority context. As illustrated below, the priority context input is encircled.



This time, the second input parameter will be evaluated first. The `concat` function will still concatenate the same values, but this time it will process the sequence `3,4` first. Consequently, the output becomes:

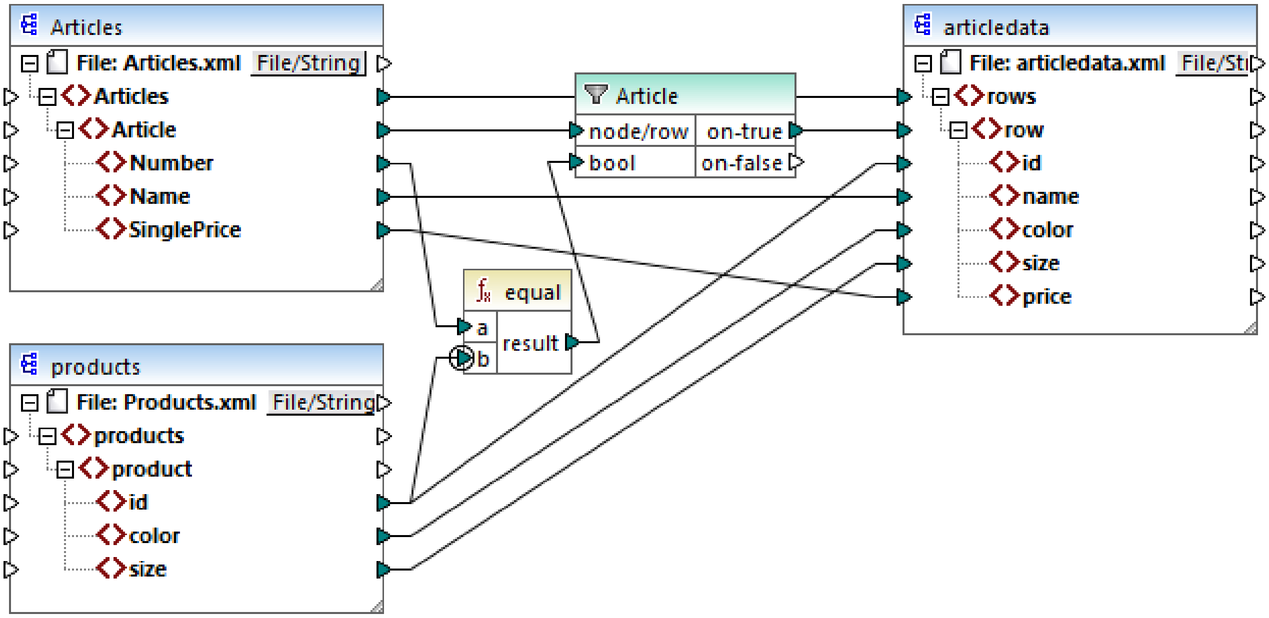
```
<data>
    <value>13</value>
    <value>23</value>
    <value>14</value>
    <value>24</value>
</data>
```

So far, you have seen only the theoretical part behind priority context. For a more practical scenario, see Example: Filter with priority context 1290.

8.3.3.1 Example: Filter with priority context

When a function is connected to a filter, priority context affects not only the function itself, but also the evaluation of the filter. The mapping below illustrates a typical case when it's required to set a priority context in order to get the correct output. You can find this mapping at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\FilterWithPriority.mfd**.

Note: This mapping uses XML components, but the same logic as described below applies for all other component types in MapForce, including EDI, JSON, and so on. For databases, it's advisable to perform filtering using SQL WHERE 350 components rather than standard filters.



The aim of the mapping above is to copy people data from **Articles.xml** into a new XML file with a different schema, **articledata.xml**. At the same time, the mapping should look up the details of each article in the **Products.xml** file and join them to the respective article record. Note that each record in **Articles.xml** has a **Number** and each record in **Products.xml** has an **id**. If these two values are equal, then all the other values (**Name**, **SinglePrice**, **color**, **size**) should be copied to the same **row** in the target.

This goal has been accomplished by adding a filter. Each filter requires a Boolean condition as input; only those nodes/rows that satisfy the condition will be copied over to the target. For this purpose, there is an **`equal`** function on the mapping. The **`equal`** function checks if the article number and product ID are equal in both sources. The result is then supplied as input to the filter. If **true**, then the **Article** item is copied to the target.

Notice that a priority context has been defined on the second input parameter of the second **`equal`** function. In this mapping, the priority context makes a big difference, and not setting it will result in incorrect mapping output.

Initial mapping: No priority context

Here is the mapping logic without priority context:

- According to the general mapping rule, for each **Article** that satisfies the filter condition, a new **row** is created in the target. The connection between **Article** and **row** (via the function and filter) takes care of this part.
- The filter checks the condition for each article. To do this, it iterates through all products, and brings multiple products in the current context.
- To populate the **id** on the target side, MapForce follows the general rule (for each item in the source, create an item in the target). However, as explained above, all products from **Products.xml** are in the current context. There is no connection between **product** to anywhere else in the target so as to read the **id** of a specific product only. As a consequence, multiple **id** elements will be created for each **Article** in the target. The same happens with **color** and **size**.

To summarize: items from **Products.xml** have the filter's context (which must iterate through each product); therefore, the **id**, **color**, and **size** values will be copied to each target **row** as many times as there are products in the source file, and generate incorrect output like the one below:

```
<rows>
    <row>
        <id>1</id>
        <id>2</id>
        <id>3</id>
        <name>T-Shirt</name>
        <color>red</color>
        <color>blue</color>
        <color>green</color>
        <size>10</size>
        <size>20</size>
        <size>30</size>
        <price>25</price>
    </row>
</rows>
```

Solution A: Use priority context

The problem above was solved by adding a priority context to the function that computes the filter's Boolean condition.

Specifically, if the second input parameter of the `equal` function is designated as priority context, the sequence incoming from **Products.xml** is prioritized. This translates to the following mapping logic:

- For each product, populate input **b** of the `equal` function (in other words, prioritize **b**). At this stage, the details of the current product are in context.
- For each article, populate input **a** of the `equal` function and check if the filter condition is true. If yes, then put the article details as well into the current context.
- Next, copy the article and product details from the current context to the respective items in the target.

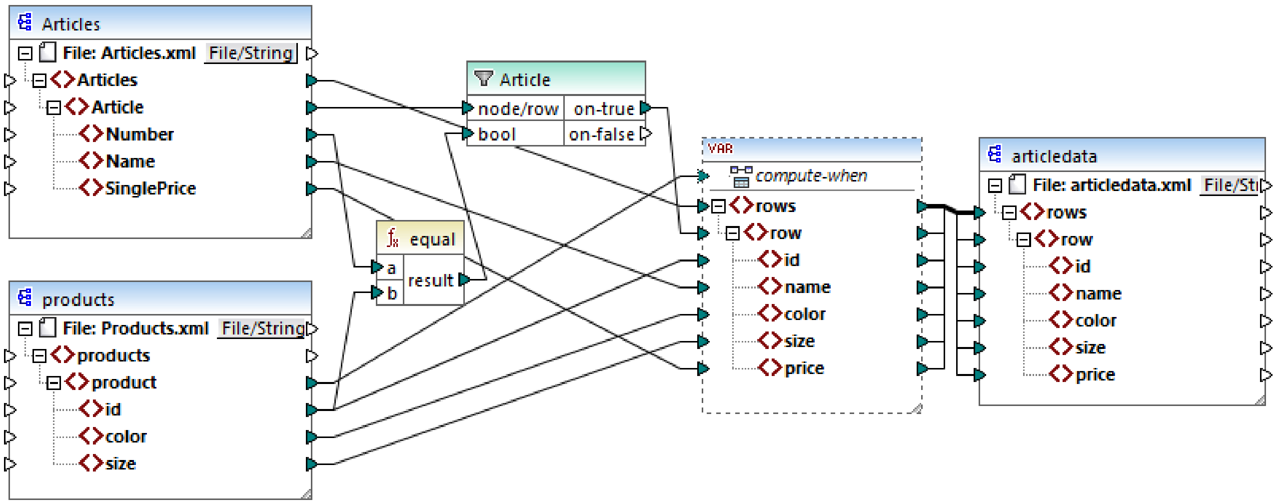
The mapping logic above produces correct output, for example:

```
<rows>
    <row>
        <id>1</id>
        <name>T-Shirt</name>
        <color>red</color>
        <size>10</size>
        <price>25</price>
    </row>
</rows>
```

Solution B: Use a variable

As an alternative solution, you could bring each article and product that matches the filter's condition into the same context with the help of an intermediate variable. Variables are suitable for scenarios like this one because they let you store data temporarily on the mapping, and thus help you change the context as necessary.

For scenarios like this one, you can add to the mapping a variable that has the same schema as the target component. On the **Insert** menu, click **Variable**, and supply the **articledata.xsd** schema as structure when prompted.



In the mapping above, the following happens:

- Priority context is not used any longer. There is a variable instead, which has the same structure as the target component.
- As usual, the mapping execution starts from the target root node. Before populating the target, the mapping collects data into the variable.
- The variable is computed in the context of each product. This happens because there is a connection from **product** to the **compute-when** input of the variable.
- The filter condition is thus checked in the context of each product. Only if the condition is true will the variable's structure be populated and passed on to the target.

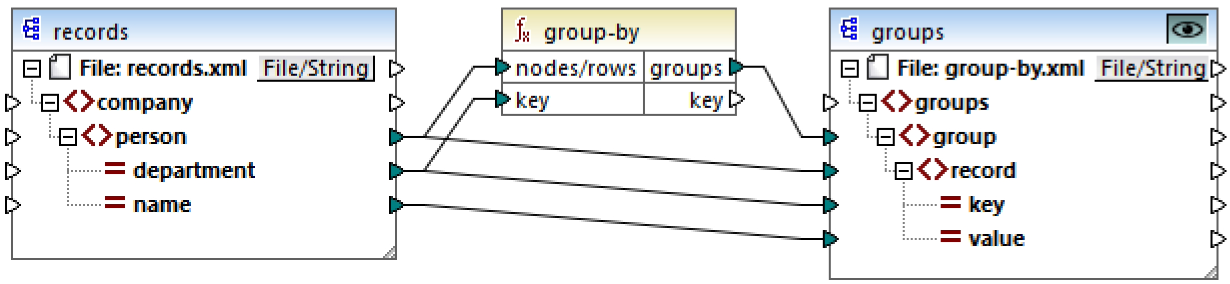
8.3.4 Multiple target components

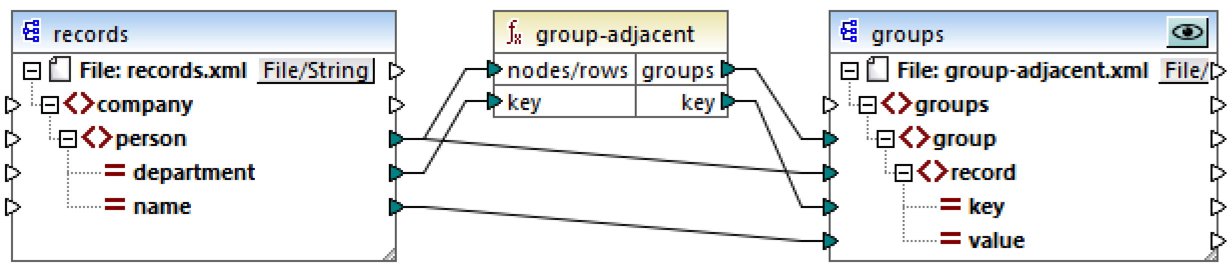
A mapping may have multiple source and target components. When there are multiple target components, you can preview only one component output at a time in MapForce, the one that you indicate by clicking the **Preview** button. In other execution environments (MapForce Server or generated code), all of the target components will be executed sequentially, and the respective output of each component will be produced.

By default, target components are processed from top to bottom and from left to right. If necessary, you can influence this order by changing the position of target components in the mapping window. The point of reference is each component's top left corner. Note the following:

- If two components have the same vertical position, then the leftmost takes precedence.
- If two components have the same horizontal position, then the highest takes precedence.
- In the unlikely event that components have the exact same position, then an unique internal component ID is automatically used, which guarantees a well-defined order but which cannot be changed.

For an example of how this works, open the following demo mapping: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\GroupingFunctions.mfd**. This mapping consists of multiple source and multiple target components; only a fragment is shown below.





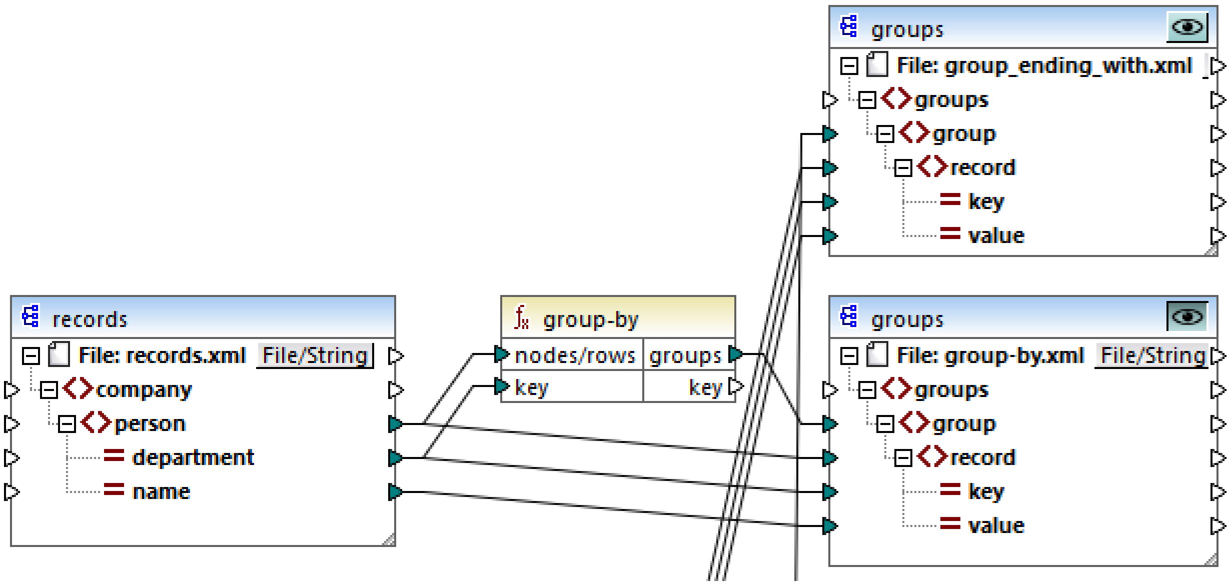
According to the rules, the default processing order of this mapping in MapForce Server and in generated code is from top to bottom. You can check that this is the case by generating XSLT 2.0 code, for example.

1. On the **File** menu, click **Generate code in | XSLT 2.0**.
2. When prompted, select a target directory for the generated code.

After generation, the target directory includes several XSLT files and a **DoTransform.bat** file. The latter can be executed by RaptorXML Server (requires a separate license). The **DoTransform.bat** file processes components in the same order as they were defined on the mapping, from top to bottom. This can be verified by looking at the `--output` parameter of each transformation.

```
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-by.xml" --xml-
validation-error-as-warning=true %* "MappingMapTogroups.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-adjacent.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-into-blocks.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups3.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v2.xml" --output="group-starting-
with.xml" --xml-validation-error-as-warning=true %* "MappingMapTogroups4.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v3.xml" --output="group_ending_with.xml"
--xml-validation-error-as-warning=true %* "MappingMapTogroups5.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```


The last transformation produces an output file called **group-ending-with.xml**. Let's now move this target component on the mapping to the very top:



If you now generate the XSLT 2.0 code again, the processing order changes accordingly:

```
RaptorXML xslt --xslt-version=2 --input="records-v3.xml" --output="group_ending_with.xml"
--xml-validation-error-as-warning=true %* "MappingMapTogroups.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-by.xml" --xml-
validation-error-as-warning=true %* "MappingMapTogroups2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-adjacent.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups3.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-into-blocks.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups4.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v2.xml" --output="group-starting-
with.xml" --xml-validation-error-as-warning=true %* "MappingMapTogroups5.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

In the code listing above, the first call now produces **group-ending-with.xml**.

You can change the processing order in a similar way in other code languages and in compiled MapForceServer execution files (.mfx).

Chained mappings

The same processing sequence as described above is followed for chained mappings. The chained mapping group is taken as one unit, however. Repositioning the intermediate or final target component of a single chained mapping has no effect on the processing sequence. Only if multiple "chains" or multiple target components exist in a mapping does the position of the final target components of each group determine which is processed first.

- If two final target components have the same vertical position, then the leftmost takes precedence.
- If two final target components have the same horizontal position, then the highest takes precedence.
- In the unlikely event that components have the exact same position, then an unique internal component ID is automatically used, which guarantees a well-defined order but which cannot be changed.

8.4 Running Mappings with Authentication

Sometimes, mappings created with MapForce need to access external resources that require authentication (for example, a Web service). To address this need, MapForce (and, on the server side, MapForce Server and FlowForce Server) provides support for the following authentication means:

- Username and password. These are typically necessary for calling Web services that require basic HTTP authentication.
- OAuth 2.0 credentials. These are necessary to call Web services where OAuth 2.0 verification is required. Note that, on the server side, OAuth 2.0 credentials are supported in MapForce Server Advanced Edition and FlowForce Server Advanced Edition.

In order to authenticate with the Web service and run the mapping successfully, you have the following options when designing a mapping in MapForce:

- Define a *credential*. This is the recommended and most flexible approach. A credential object may contain full authentication details or be just a named reference without carrying sensitive data. In the latter case, the actual details (for example, password) will have to be provided at mapping runtime. Optionally, you can save the sensitive details associated with the credential in encrypted form as part of the mapping design file (*.mfd). If you need to run the mapping with MapForce Server in other environments without any manual intervention, you can make the credential portable. The portable credential does not necessarily have to contain the actual username and password (although it can, depending on your choice). You can always set or override these details manually at mapping runtime. For more information, see Credentials 1297.
- Use *dynamic authentication*, that is, supply the username and password as input parameters to the mapping. In this approach, you will need to supply the parameter values to the mapping each time the mapping runs. Optionally, to be able to preview the mapping in MapForce, you can store the username and password as "design-time" execution values; however, they will be stored in clear text. Note that OAuth credentials cannot be supplied as input parameters to the mapping; this is possible only for the username and password. For more information, see Dynamic Authentication 1309.
- (Legacy approach) Define the username and password in MapForce not as a named credential object, but as standalone username and password. When you save the mapping, the username and password are saved in encrypted form as part of the mapping design file (*.mfd). This approach is supported for backward compatibility of mappings designed in earlier versions of MapForce. For more information, see Stored Username and Password 1311.

The following sections show you how to configure the mapping for each of the scenarios mentioned above.

8.4.1 Credentials

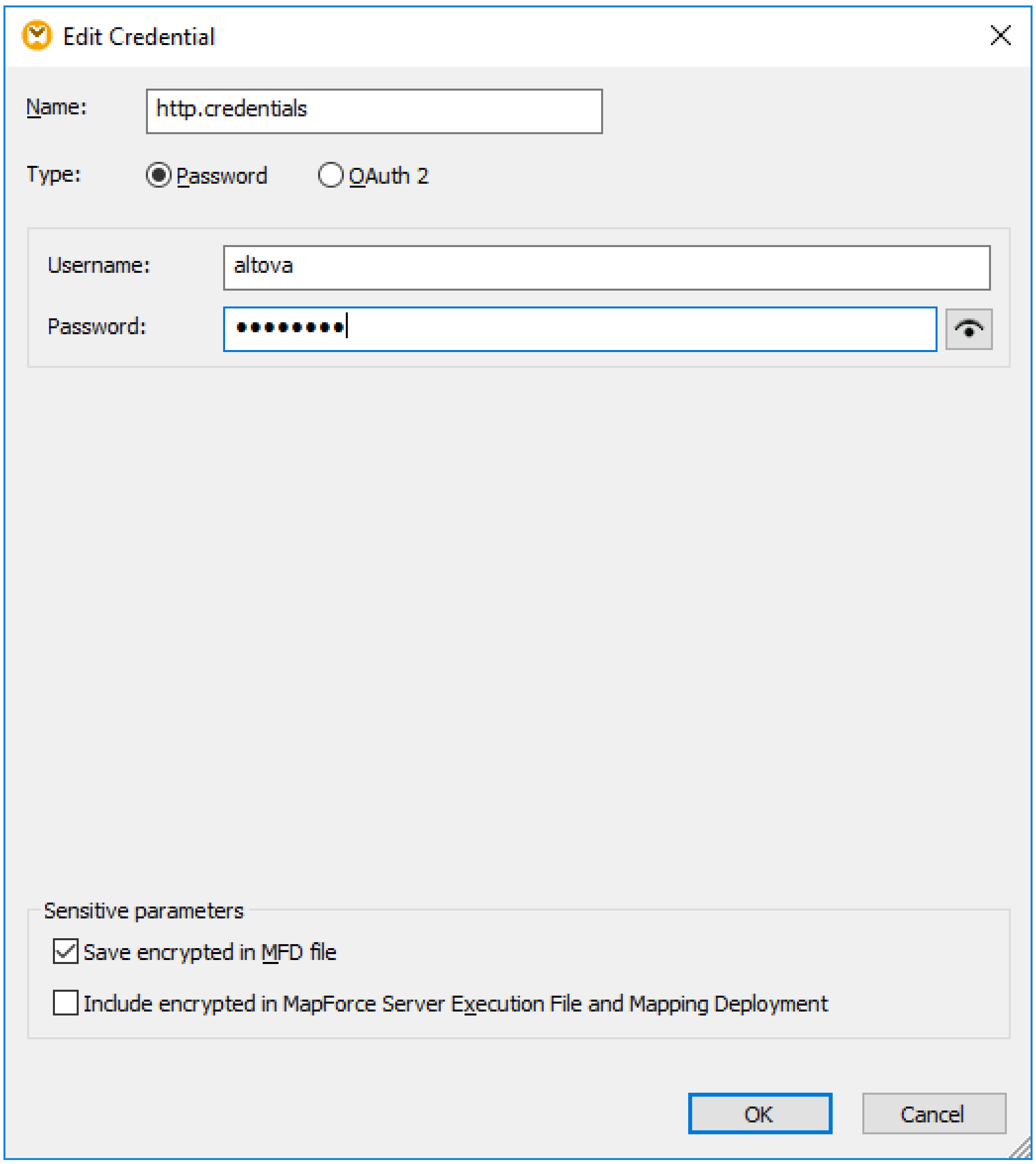
Credential objects provide a way to make authentication data (such as usernames, passwords, and OAuth authentication details) portable across various mapping execution environments, in a secure way. Credentials are useful in mappings that require basic HTTP authentication or OAuth 2.0 authorization. You can define credentials in MapForce and also in FlowForce Server. If credentials were defined in MapForce, you can optionally deploy them to FlowForce Server, similar to how mappings are deployed.

A credential must have a name and a type; all other details are optional. The name is just an indication to the execution engine (be it MapForce or MapForce Server) that the mapping requires credentials to run. You can choose whether the sensitive information associated with the credential should be stored encrypted inside the mapping, or be supplied only at mapping runtime. If you stored the sensitive information inside the mapping, you can still override it when executing the mapping in a server environment.

The type of credential can be either **Password** or **OAuth 2.0**.

To create a credential:

1. Right-click an empty area on the mapping, and select **Open Credentials Manager** from the context menu. The Credentials Manager[1300] grid appears at the top of the mapping pane.
2. Click **Add**. A dialog box appears where you can enter the credential information. (The same dialog box appears when you create or edit the HTTP security settings for a Web service component.)
3. Enter a credential name.
4. Select a credential type (**Password** or **OAuth 2.0**).



5. To authenticate the mapping at design time, fill in the following fields:
   - In case of "password" credentials, fill in the username and password required to authenticate to the Web service. Confirm the password when prompted to do so.
   - For details concerning **OAuth 2.0** credentials, see OAuth 2.0 Credentials [1301].

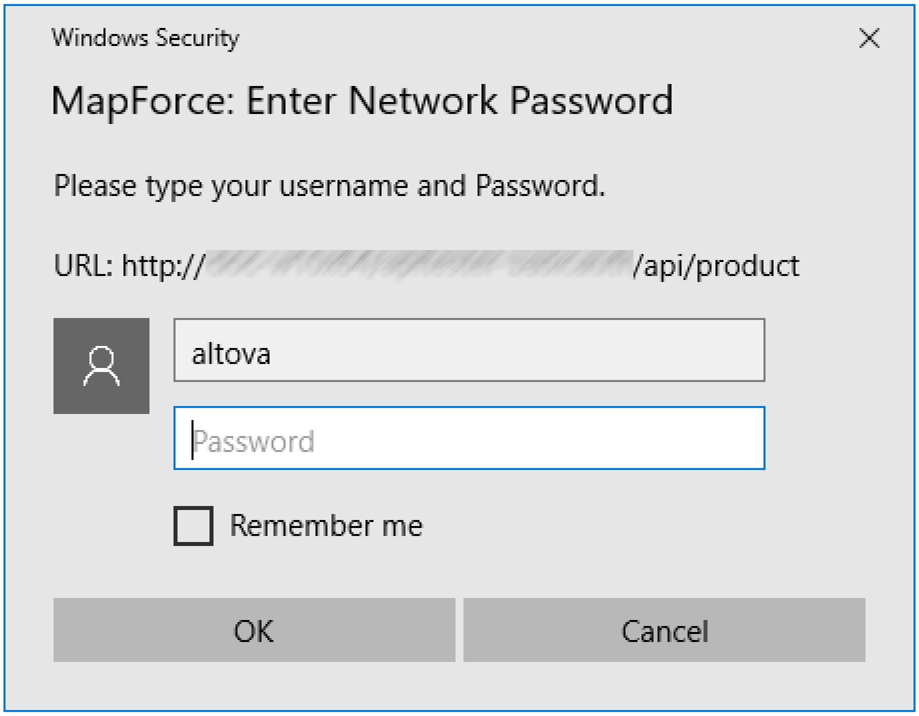
You may optionally save sensitive data in the mapping design and deploy it to other environments in encrypted form, or not store it at all. If you stored sensitive data, you can still view after closing the mapping, by clicking the **Reveal** button.

> In case of password credentials, the sensitive field is **Password**. In case of OAuth 2 credentials, the sensitive fields are **Client Secret**, **Access Token**, and **Refresh Token**.

The following check boxes let you control the desired behavior with respect to storing sensitive data.

Save encrypted in MFD file	Select this check box to save the sensitive data in the mapping design file (.mfd), in encrypted form.
Include in MapForce Server Execution File and Mapping Deployment	Select this check box if you would like the sensitive data to be included in the MapForce Server execution file (.mfx) when you compile one as described in Compiling Mappings to MapForce Server Execution Files [1378]. When this check box is selected, the sensitive data will also be included in the package deployed to FlowForce Server, see Deploying Mappings to FlowForce Server [1381].

The sensitive fields are stored in reversible encrypted form, since they must be decrypted and used at mapping execution time. If you prefer not to store sensitive data, clear the check boxes above. In this case, MapForce may prompt you for the password when you attempt to run the mapping, for example:



On the server side, you can supply the password dynamically at mapping execution time. For more details, see Credentials in MapForce Server [1303] and Credentials in FlowForce Server [1305].

8.4.1.1 Credentials Manager

Credentials Manager is the graphical user interface where you can manage credentials in any mapping. When a mapping contains credentials, these are displayed in a grid at the top of the mapping window. For each credential object, the grid displays the credential type (**OAuth** or **Password**), the name, and the number of references that exist in the current mapping for this credential.

Credentials Manager

To hide **Credentials Manager**, click **Close** or press the **Esc** key.

To show **Credentials Manager**, right-click a mapping, and select **Open Credentials Manager** from the context menu.

From **Credentials Manager**, you can take the following actions:

Icon	Description
	Create a new credential object, see Credentials[1297].
	Edit an existing credential object.
	Deploy the credential object to FlowForce Server, see Deploying Credentials to FlowForce Server[1307].
	Delete the credential object from the mapping.

You can also cut, copy and paste credential records by right-clicking a record and selecting the respective command from the context menu, or by using the standard Windows shortcuts (**Ctrl+X**, **Ctrl+C**, and **Ctrl+V** for **Cut**, **Copy**, and **Paste**, respectively). To find if any credentials are referenced by components on the mapping, right-click anywhere inside the Credentials Manager, and select **Find All Referencing Components** from the context menu. The outcome is reported in the Messages window.

8.4.1.2 Credentials in UDFs

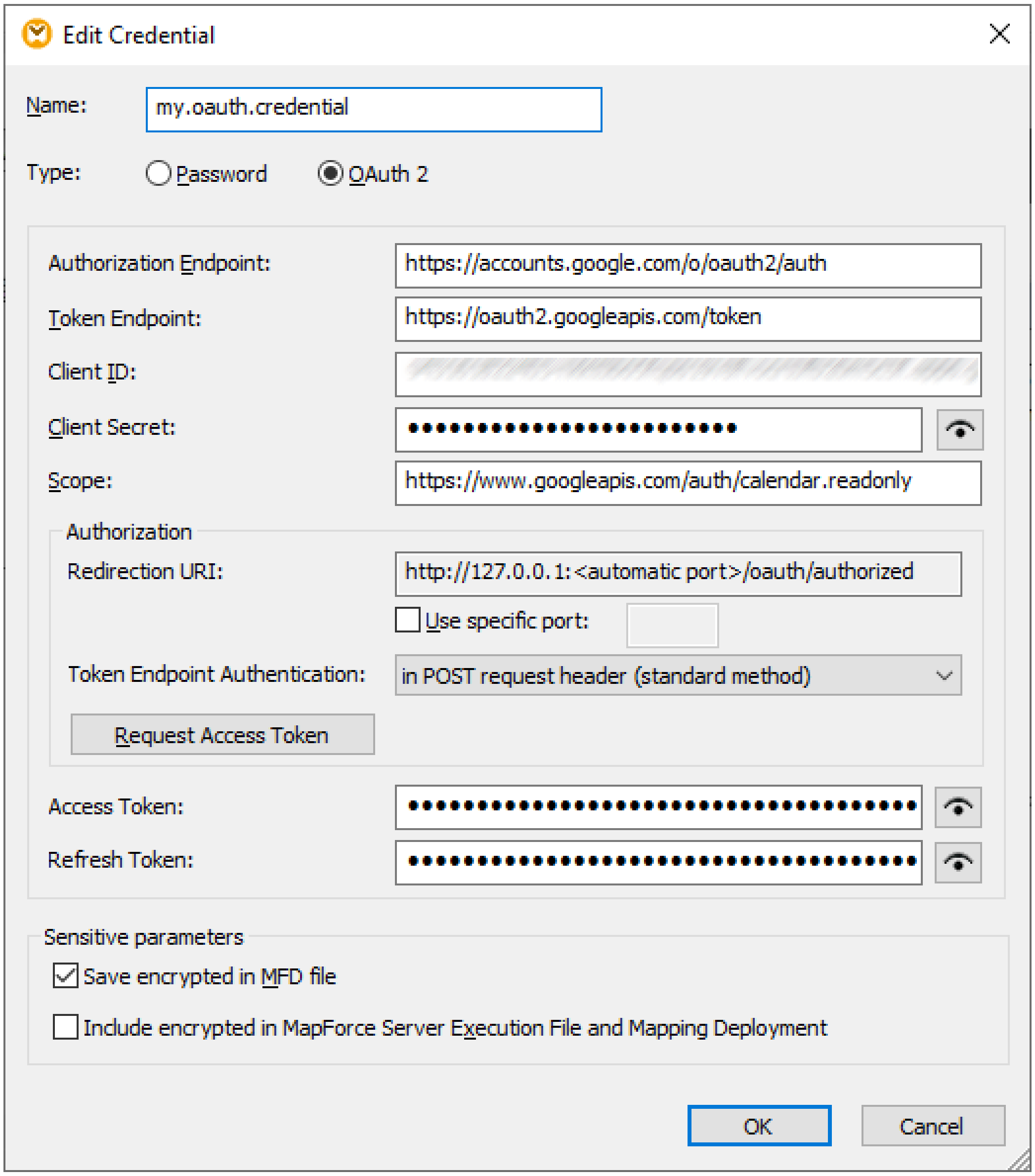
Mappings that contain user-defined functions (UDFs) can be imported into other mappings, see Calling and Importing User-Defined Functions[930]. If the imported mapping contains credentials, then credentials appear as imported in Credentials Manager. Imported credentials are shown with a yellow background.

By default, imported credentials are not saved with the main mapping, but you can optionally create a local copy and save them in the main mapping. A warning dialog box appears to that effect whenever you attempt to edit an imported credential from the main mapping.

Note: If both the imported mapping and the main contain credentials with the same name, collisions may occur. If this happens, the credential from the main mapping takes precedence. To avoid name collisions, ensure that credential names are unique within the scope of a mapping and all its imported UDFs.

8.4.1.3 OAuth 2.0 Credentials

To run a mapping that requires OAuth 2.0 authorization in MapForce, you will need to create a credential of type **OAuth 2.0** using the Credentials Manager[1300], fill in the OAuth 2.0 details, and go through a manual authorization process as described below.



The fields associated with an OAuth 2.0 credential object are listed below. To obtain these values, you must first register with the Web service provider (for example, Google API Console, Facebook API, Bitbucket API, and so on).

Authorization Endpoint	Specifies the URI from where MapForce initiates authorization flows. You can obtain this value after registering with the Web service provider.
Token Endpoint	Specifies the URI from where MapForce initiates token flows. You can obtain this value after registering with the Web service provider.
Client ID	The identifier of the client application (MapForce, in this case). You can obtain this value after registering with the Web service provider.

Client secret	The secret associated with the client application. You can obtain this value after registering with the Web service provider.
Scope	The scope of the client application, if required by the provider. You can obtain this value after registering with the Web service provider.
Redirection URI	The OAuth 2.0 authorization server needs a callback URL where it will send responses to MapForce. MapForce will designate automatically the URL and port for communicating with the authorization server. If the authorization server mandates that communication should take place over a specific port, select **Specific port** and enter the desired port name.
Token Endpoint Authentication	Most OAuth 2.0 authorization servers require that the authorization details be submitted **in the POST request header**. This is also the value selected by default from the drop-down list. Some OAuth 2.0 authorization servers accept the authentication details only in the body of the POST request. For such authorization servers, select the value **in POST request body** from the drop-down list.
Access token	This is the access token returned by the authorization server. The mapping will execute successfully only if the resource server determines that the access token is correct and valid.
Refresh token	This is the refresh token returned by the authorization server. It is required when the **Access token** expires (see above). In rare cases when the access token never expires, this is not necessary.

After you have filled in the fields above, click **Request Access Token** to obtain the access token required to run the mapping. The exact authorization process depends on the Web service provider; it typically requires that you manually confirm in a browser window that you grant access to the Web service to determine your identity (for example, your Google account if the mapping calls a Google API). At the end of the process, MapForce displays a confirmation message that the access token has been obtained from the server. Note that the access token will be saved only if you selected the **Save encrypted in MFD file** check box, as described in Credentials1297. If you did not select the **Save encrypted in MFD** file check box, you will need to manually authorize each time when you run the mapping.

The access token may expire after some time; the period after which a token expires depends on the provider of the Web service. Moreover, it may also be the case that the access token was explicitly revoked by the Web service provider. When this happens, the Messages window will display authorization errors when you attempt to preview the mapping. In this case, you will need to click **Request Access Token** and go through the authorization process again before you can run the mapping.

8.4.1.4 Credentials in MapForce Server

After you compile the mapping to a MapForce Server execution file (.mfx), MapForce Server will run the .mfx file depending on your choices at mapping design time, as explained in Creating Credentials1297.

If you selected the **Include in MapForce Server Execution File and Mapping Deployment** check box when creating the credential in MapForce, MapForce Server will use at mapping runtime any credentials that were stored in the .mfx file. This means that you can run the mapping with a command such as:

```
<exec> run mapping.mfx
```

Where **<exec>** is the path to the MapForce Server executable. This path can be either absolute or, if the current directory is the same as the executable, you can enter just the executable name.

If you entered only the credential name (without username and password) in MapForce, then you must explicitly provide these details at mapping runtime, with the help of the **--credential** command line option available for the `run` command. This way, you can use, for example, a different set of credentials in production, as opposed to those used when you designed the mapping. The `--credential` option has the form `--credential=KEY:VALUE`., where

- **KEY** is the name of the credential as it was defined in MapForce.
- **VALUE** is a credential property, or a list of properties separated by ampersand (&). For credentials of type "password", the possible properties are **username** and **password**. For credentials of type OAuth 2.0, the only supported property is **oauth:token**.
- The actual property values are supplied just like query parameters in a URL, using the "=" sign.

For example:

```
<exec> run mapping.mfx --credential="mycredential:username=admin&password=4xJ38dnx7"
```

In the code listing above, the value of the `--credential` option was enclosed within quotes in order to treat the value literally, since the username and password are separated by an ampersand character.

If your mapping needs multiple sets of named credentials to run, you can specify the **--credential** option multiple times.

> The credentials supplied as command line options take precedence over stored credentials.

If you did not select the **Include in MapForce Server Execution File and Mapping Deployment** check box, the sensitive fields are missing. This means that you must supply the password at the command line while still referring the credential by its name, for example:

```
<exec> run mapping.mfx --credential=mycredential:password=4xJ38dnx7
```

> The following fields are considered sensitive data:
>
> - **Password** (for credentials of type "Password")
> - **Client Secret**, **Access Token**, and **Refresh Token** (for credentials of type "OAuth 2.0")

For mappings that require OAuth 2.0 authorization, the MapForce Server command line accepts an OAuth 2.0 access token as input at the mapping runtime. Note that the MapForce Server command line does not provide an interactive GUI by design, so you will need to obtain the OAuth 2.0 access token by external means (for example, by requesting it with MapForce) when using the command line specifically. This is, however, not necessary if MapForce Server runs under FlowForce Server management, since the latter is capable of acquiring a new OAuth 2.0 access token at runtime by itself.

At the command line, running the mapping with stored credentials is possible as long as the stored OAuth 2.0 token has not expired or has not been revoked by the Web service provider. To address this, supply a new OAuth 2.0 access token (obtained by some external means) by using the **`--credential`** option, for example:

```
<exec> run mapping.mfx --
credential=my_oauth_credential:oauth:token=jdsaflkajlkewsaiurthczv904215-jhd
```

Where:

- **`my_oauth_credential`** is the name of the OAuth 2.0 credential created from MapForce.
- **`oauth:token`** is the way to indicate to MapForce Server that a new OAuth 2.0 access token is being supplied at runtime.

MapForce Server API

The MapForce Server API provides methods to create credentials, add properties to credentials, and close credentials after you finished declaring them. For more details, refer to the MapForce Server documentation (https://www.altova.com/documentation)

8.4.1.5 Credentials in FlowForce Server

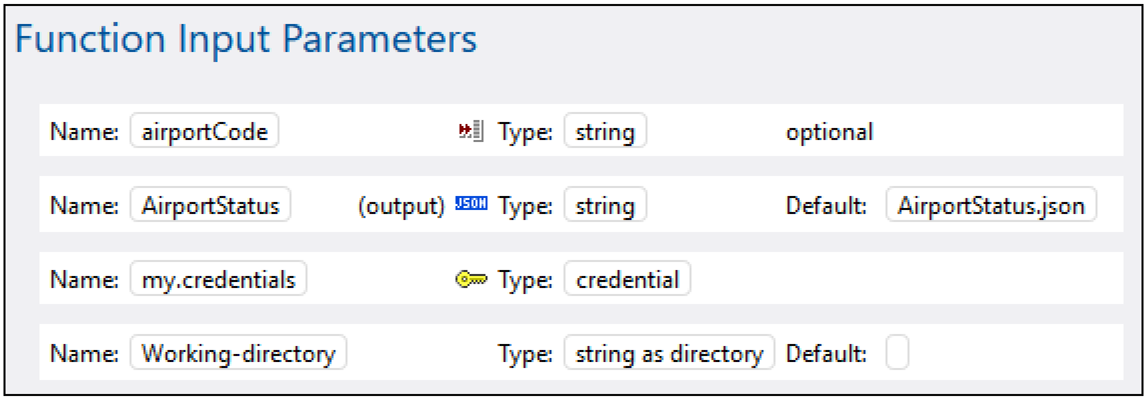
When you deploy a mapping containing credentials from MapForce to FlowForce Server, the credentials are deployed to the server as well. The deployed information will contain only the fields that you filled in when creating the credential record. For example, this may be an empty credential (if you chose to store only the credential name) or a credential object that contains both the username and password.

You can also deploy credential objects from MapForce to FlowForce Server as standalone objects, separately from the main mapping, see Deploying Credentials to FlowForce Server 1307.

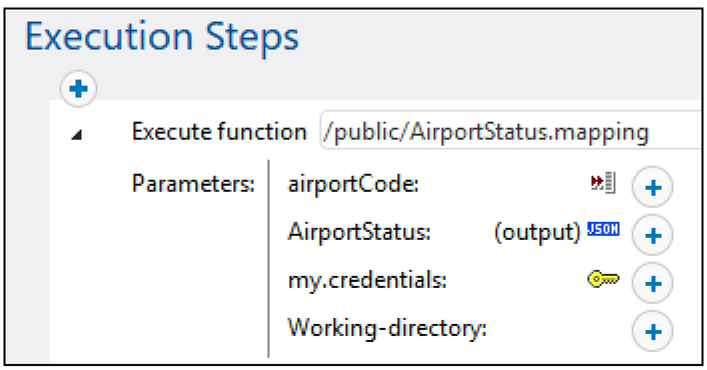
> The following fields are considered sensitive data:
>
> - **Password** (for credentials of type "Password")
> - **Client Secret**, **Access Token**, and **Refresh Token** (for credentials of type "OAuth 2.0")

The sensitive data will be deployed only if you selected the **Include in MapForce Server Execution File and Mapping Deployment** check box, as described in Creating Credentials 1297. This applies both when you deploy the mapping and when you deploy the standalone credentials.

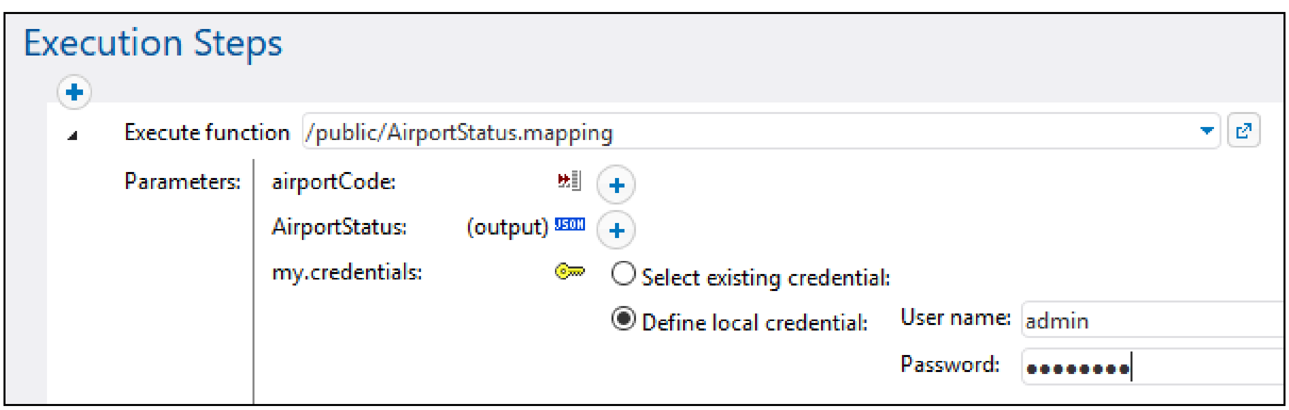
In FlowForce Server, you can see whether a mapping function needs credentials by opening the page of the respective mapping function, for example:



If you selected the **Include in MapForce Server Execution File and Mapping Deployment** check box when creating the credential, then the job will use the credentials deployed together with the mapping. In this case, you don't need to specify them from the job configuration page. For example, the following execution step will run the mapping function with the stored credentials if such exist (notice that the "my.credentials" parameter is not expanded):



You can always override the stored credentials with any other credential object that was defined directly in FlowForce Server, or with some local credentials. To do this, click the "+" button and either select a credential object that already exists in FlowForce Server, or enter the username and password directly, for example:



The credentials supplied as parameter to the execution step take precedence over credentials stored inside the mapping function.

If you did not select the **Include in MapForce Server Execution File and Mapping Deployment** check box when creating the credential in MapForce, it is mandatory to supply credentials as parameters to the execution step; otherwise, the job execution will fail.

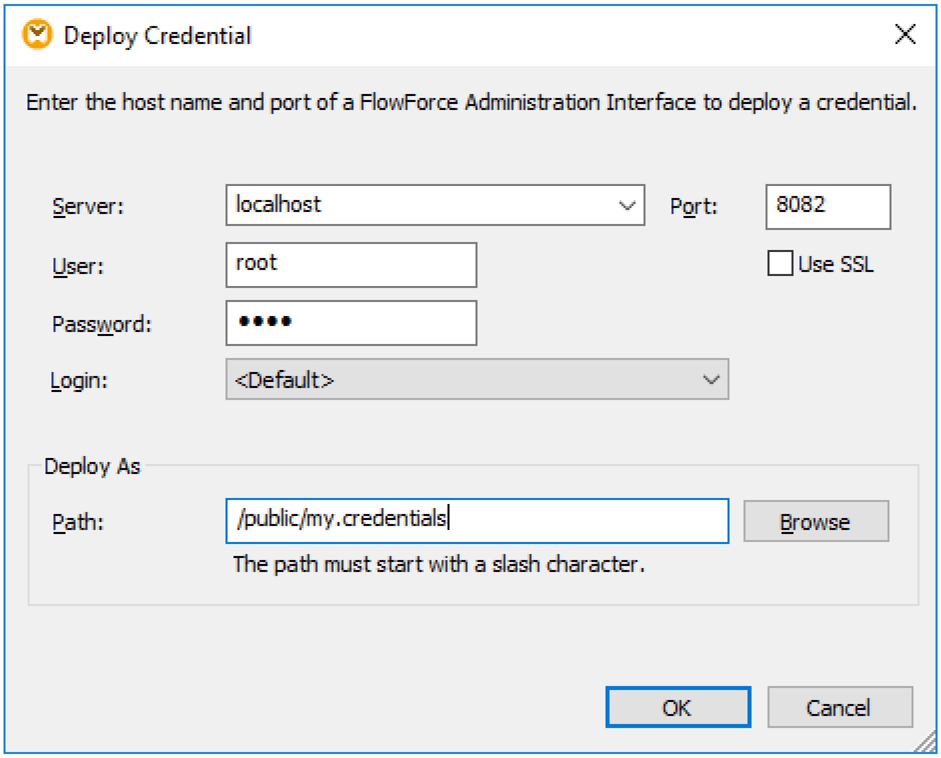
8.4.1.6 Deploying Credentials to FlowForce Server

Credentials may change frequently. This doesn't mean, however, that you need to change and redeploy the mapping each time the credentials change. Instead, you can conveniently deploy to FlowForce Server at any time just a credential object, without making any changes to the mapping. This also enables you to reuse a credential object in multiple FlowForce Server jobs.

Alternatively, you can create a credential object directly in FlowForce Server and then supply it as parameter to the execution step, just like any other FlowForce credential.

To deploy a MapForce credential to FlowForce Server:

1. Open the mapping that contains credentials.
2. Right-click an empty area on the mapping and select **Open Credentials Manager** from the context menu.
3. Select a record from the Credentials Manager grid, and then click the **Deploy Credential to FlowForce Server** button. A dialog box opens where you can enter the FlowForce Server connection details, and also specify the target path on the server:



4. Click **OK** and observe the outcome displayed in the Messages window.

The deployment options are listed below.

Setting	Description
Server, Port, Use SSL	Enter the server host name (or IP address) and port of FlowForce Server. These could be **localhost** and **8082** if FlowForce Server is running on the same machine at the default port. When in doubt, log on to FlowForce Server Web administration interface and check the I.P. address and port displayed in the Web browser's address bar. If you encounter connectivity errors, ensure that the machine on which FlowForce Server runs is configured to allow incoming connections on the designated address and port. To deploy the credential through a SSL-encrypted connection, select the **Use SSL** check box. This assumes that FlowForce Server is already configured to accept SSL connections. For more information, refer to FlowForce Server documentation (https://www.altova.com/documentation).
User and Password	The user name and password to be entered depends on the value of the Login drop-down list (see next option). If the Login drop-down list is set to **<Default>** or **Directly**, enter your FlowForce Server user name and password. Otherwise, enter your domain user name and password, and select the domain name from the Login drop-down list.

Setting	Description
Login	If Directory Service integration is enabled in FlowForce Server, select the domain name from this drop-down list, and enter your domain credentials in the User and Password fields (see previous option).
Path	Click **Browse**, and select the path where the credential should be saved in the FlowForce Server container hierarchy. By default, the path is set to the **/public** container of FlowForce Server. You can also create new containers or delete existing containers and mappings directly from this dialog box, provided that you have the required FlowForce Server permissions and privileges.

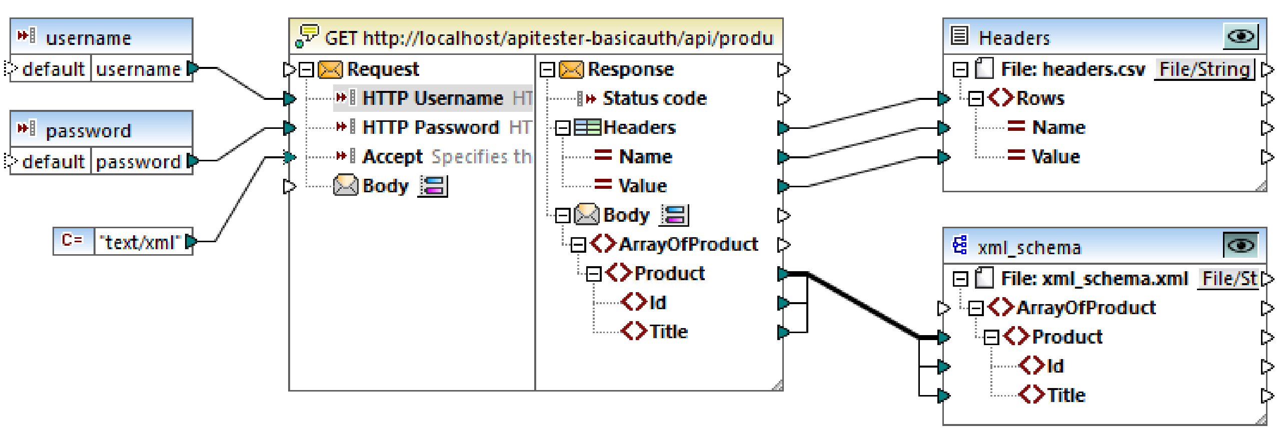
8.4.2 Dynamic Authentication

Dynamic authentication is the ability to supply the username and password to the mapping at runtime, as input parameters. Dynamic authentication is supported for mappings that call Web services where basic HTTP authentication is necessary.

To set up dynamic authentication:

1. Open a mapping that contains a Web service call, or add a Web service to a new mapping. For details, see Adding a Web Service Call (REST-Style) [587] and Adding a Web Service Call (SOAP) [605].
2. Double-click the title bar of the Web service component to open the Settings dialog box.
3. Next to "HTTP Security Settings", click **Edit**.
4. Select **Dynamic Authentication**.
5. Click **OK**.

Once dynamic authentication is enabled, the "Request" side of the Web service component gets two new input connectors: **HTTP username** and **HTTP Password**. To supply the actual values, add two new input components to the mapping, using the **Insert | Insert Input** menu command, and connect them to the **HTTP Username** and **HTTP Password**, respectively.



To preview the mapping in MapForce, you can optionally supply the username and password as "design-execution" values. To do this, double-click the title bar of each input component, and enter the username and password, respectively, in the **Value** text box. For more information, see Supplying Parameters to the Mapping 833.

> Username and password are treated just like any other design-time execution values, so they are saved in the mapping design file as clear text. To circumvent this, do not enter any design-time execution values at all, or use the Credentials 1297 approach.

8.4.2.1 Dynamic Authentication in MapForce Server

If you configured the mapping for dynamic authentication, then the respective username and password must be supplied as parameters at mapping runtime. This is not different from supplying any other parameter kinds to the mapping. For example, when calling MapForce Server at the command line, the syntax for a mapping like the one above is:

```
<exec> run mapping.mfx --p=username:admin --p=password:dj9JaVax
```

Where:

- **<exec>** is the path to the MapForce Server executable. This path can be either absolute or, if the current directory is the same as the executable, you can enter just the executable name.
- **username** and **password** are the names of the respective input parameters on the MapForce mapping.

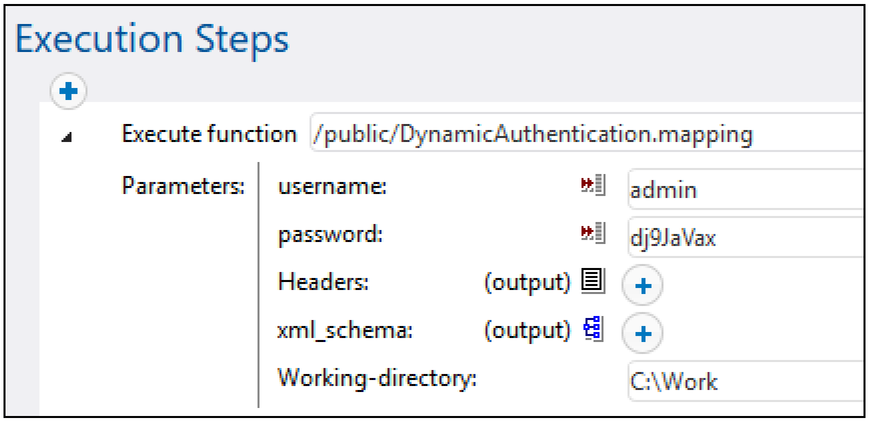
When calling the MapForce Server API, you can authenticate the mapping by calling the **AddParameter** method before calling the **Run** method. For example, in C#, the code to achieve this could look as follows:

```
try
{
    Altova.MapForceServer.Server mfs = new Altova.MapForceServer.Server();
    mfs.AddParameter("username", "admin");
    mfs.AddParameter("password", "dj9JaVax");
    mfs.WorkingDirectory = "C:\\Work";
    if(mfs.Run("C:\\Work\\mapping.mfx"))
    {
      Console.WriteLine("Success");
    }
    else
    {
      Console.WriteLine(mfs.LastExecutionMessage);
    }
    Console.ReadLine();
}
catch(Exception ex)
{
  Console.WriteLine(ex);
}
```

Again, the **username** and **password** (first argument to the **AddParameter** method) must be the same as the names of the respective input parameters on the MapForce mapping.

8.4.2.2 Dynamic Authentication in FlowForce Server

When you deploy a mapping containing dynamic authentication to FlowForce Server, the username and password become input parameters to the mapping function. Any FlowForce Server job that calls such a mapping function will require the username and password before it can run successfully, for example:



In the example illustrated above, the username and password are simply entered in the respective text boxes. However, you can also supply them as input parameters to the job. For more information, refer to FlowForce Server documentation (https://www.altova.com/documentation).

8.4.3 Stored Username and Password

Storing username and password in encrypted form in the mapping is one of the ways to run mappings with authentication. This approach makes it simple to run the mapping regardless of the execution environment because the sensitive data is stored encrypted as part of the mapping, or, in server execution, as part of the server execution file. However, despite reduced complexity, it has the disadvantage that, whenever the username and password should change, you need to update and recompile the mapping to a new server execution .mfx file, or redeploy it to FlowForce Server. Also, it is not possible to change the username and password at mapping runtime, like it is possible with Credentials[1297].

> This approach is supported only for backward compatibility with earlier versions of MapForce. It is recommended to use either Credentials[1297] or Dynamic Authentication[1309] instead.

To set up authentication with stored username and password:

1. Open a mapping that contains a Web service call, or add a Web service to a new mapping.
2. Double-click the title bar of the Web service component to open the Settings dialog box.
3. Next to "HTTP Security Settings", click **Edit**.
4. Select the radio button next to **Username** and fill in the username and password to be stored. Confirm the password when prompted to do so.
5. Click **OK**.

You may now preview the mapping in MapForce without any other further prerequisites. In MapForce Server, you can run the mapping as you would run any mapping that doesn't require credentials. For example, when calling MapForce Server at the command line, the syntax to run the mapping could be:

```
<exec> run mapping.mfx
```

If you deploy the mapping to FlowForce Server in order to run it as a mapping function, there is no special indication in FlowForce Server that the mapping includes stored credentials. You can just create a job from the mapping function and run it without special prerequisites.

8.5 Processing Multiple Input or Output Files

You can configure MapForce to process multiple files (for example, all files in a directory) when the mapping runs. Using this feature, you can solve tasks such as:

- Supply to the mapping a list of input files to be processed
- Generate as mapping output a list of files instead of a single output file
- Generate a mapping application where both the input and output file names are defined at runtime
- Convert a set of files to another format
- Split a large file (or database) into smaller parts
- Merge multiple files into one large file (or load them into a database)

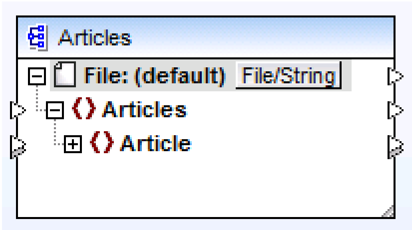
You can configure a MapForce component to process multiple files in one of the following ways:

- Supply the path to the required input or output file(s) using wildcard characters instead of a fixed file name, in the component settings (see Changing the Component Settings 82). Namely, you can enter the wildcards * and ? in the Component Settings dialog box, so that MapForce resolves the corresponding path when the mapping runs.
- Connect to the root node of a component a sequence which supplies the path dynamically (for example, the result of the `replace-fileext` function). When the mapping runs, MapForce will read dynamically all the input files or generate dynamically all the output files.

Depending on what you want to achieve, you can use either one or both of these approaches on the same mapping. However, it is not meaningful to use both approaches at the same time on the same component. To instruct MapForce which approach you want to use for a particular component, click the **File** (File) or **File/String** (File/String) button available next to the root node of a component. This button enables you to specify the following behavior:

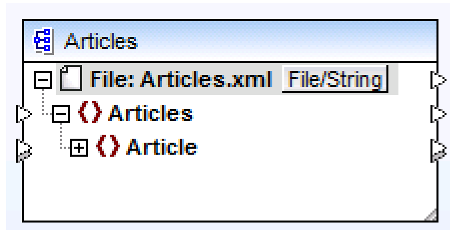
<table>
<tr>
<td>Use File Names from Component Settings</td>
<td>If the component should process one or several instance files, this option instructs MapForce to process the file name(s) defined in the Component Settings dialog box.

If you select this option, the root node does not have an input connector, as it is not meaningful.



If you did not specify yet any input or output files in the Component Settings dialog box, the name of the root node is File: (default). Otherwise, the root node displays the name of the input file, followed by a semi-colon (;), followed by the name of the output file.</td>
</tr>
</table>

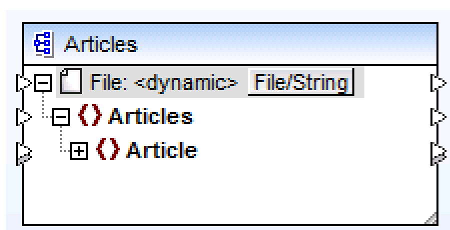
<table>
<tr>
<td></td>
<td>If the name of the input is the same with that of the output file, it is displayed as name of the root node.



Note that you can select either this option or the Use Dynamic File Names Supplied by Mapping option.</td>
</tr>
<tr>
<td>Use Dynamic File Names Supplied by Mapping</td>
<td>This option instructs MapForce to process the file name(s) that you define on the mapping area, by connecting values to the root node of the component.

If you select this option, the root node gets an input connector to which you can connect values that supply dynamically the file names to be processed during mapping execution. If you have defined file names in the Component Settings dialog box as well, those values are ignored.

When this option is selected, the name of the root node is displayed as File: <dynamic>.



This option is mutually exclusive with the Use File Names from Component Settings option.</td>
</tr>
<tr>
<td>Parse Strings to XML, Parse Strings to JSON, Parse Strings to CSV, Parse Strings to FLF, Parse Strings to EDI</td>
<td>When switched on, this option enables the component to accept a string value as input to the root node, and convert it to an XML, JSON, CSV, FLF, or EDI structure, respectively. For more information, see Parsing and Serializing Strings [1324].</td>
</tr>
<tr>
<td>Serialize XML to Strings, Serialize JSON to Strings, Serialize CSV to Strings, Serialize FLF to Strings, Serialize EDI to Strings</td>
<td>When switched on, this option enables the component to accept a structure as input, and convert it to string. The input structure can be XML, JSON, CSV, Fixed-length Field, or EDI, respectively. For more information, see Parsing and Serializing Strings [1324].</td>
</tr>
</table>

Multiple input or output files can be defined for the following components:

- XML files
- Text files (CSV*, FLF* files and FlexText** files)
- EDI documents**
- Excel spreadsheets**
- XBRL documents**
- JSON files**
- Protocol Buffers files**

** Requires MapForce Professional Edition*
*** Requires MapForce Enterprise Edition*

The following table illustrates support for dynamic input and output file and wildcards in MapForce languages.

Target language	**Dynamic input file name**	**Wildcard support for input file name**	**Dynamic output file name**
XSLT 1.0	*	Not supported by XSLT 1.0	Not supported by XSLT 1.0
XSLT 2.0	*	*(1)	*
XSLT 3.0	*	*(1)	*
XQuery	*	*(1)	Not supported by XQuery
C++	*	*	*
C#	*	*	*
Java	*	*	*
BUILT-IN	*	*	*

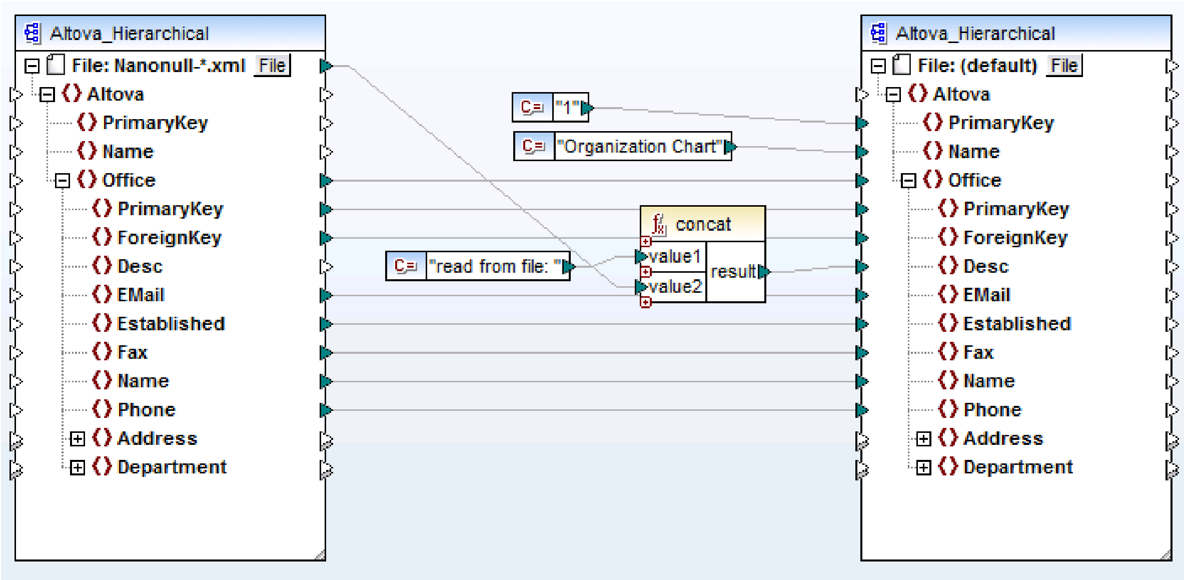
Legend:

*	Supported
(1)	XSLT 2.0, XSLT 3.0, and XQuery use the `fn:collection` function. The implementation in the Altova XSLT 2.0, XSLT 3.0, and XQuery engines resolves wildcards. Other engines may behave differently.

8.5.1 Mapping Multiple Input Files to a Single Output File

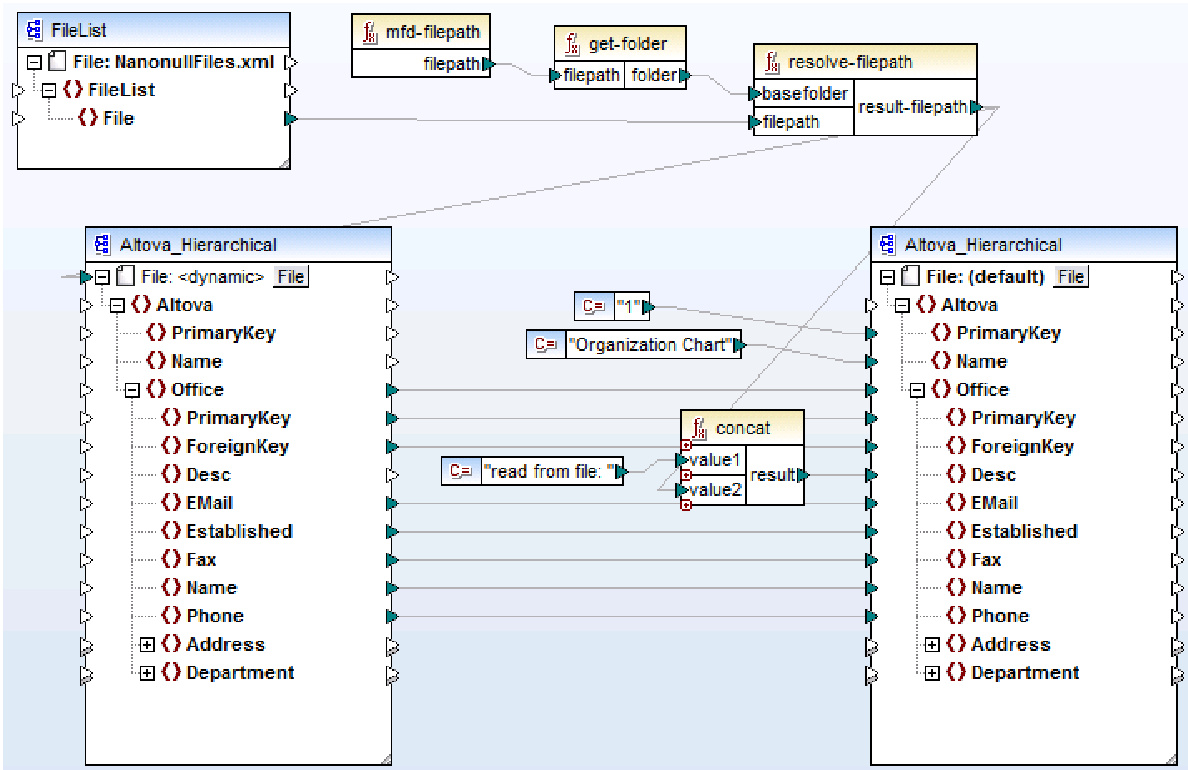
To process multiple input files, do one of the following:

- Enter a file path with wildcards (* or ?) as input file in the Component Settings dialog box. All matching files will be processed. The example below uses the ***** wildcard character in the Input XML file field to supply as mapping input all files whose name begins with "Nanonull-". Multiple input files are being merged into a **single** output file because there is no dynamic connector to the target component, while the source component accesses multiple files using the wildcard *. Notice that the name of the root node in the target component is **File: <default>**, indicating that no output file path has been defined in the Component Settings dialog box. The multiple source files are thus appended in the target document.



MergeMultipleFiles.mfd (MapForce Basic Edition)

- Map a **sequence** of strings to the *File* node of the source component. Each string in the sequence represents one file name. The strings may also contain wildcards, which are automatically resolved. A sequence of file names can be supplied by components such as an XML file , database text fields, text files (CSV or fixed length), or an Excel sheet.

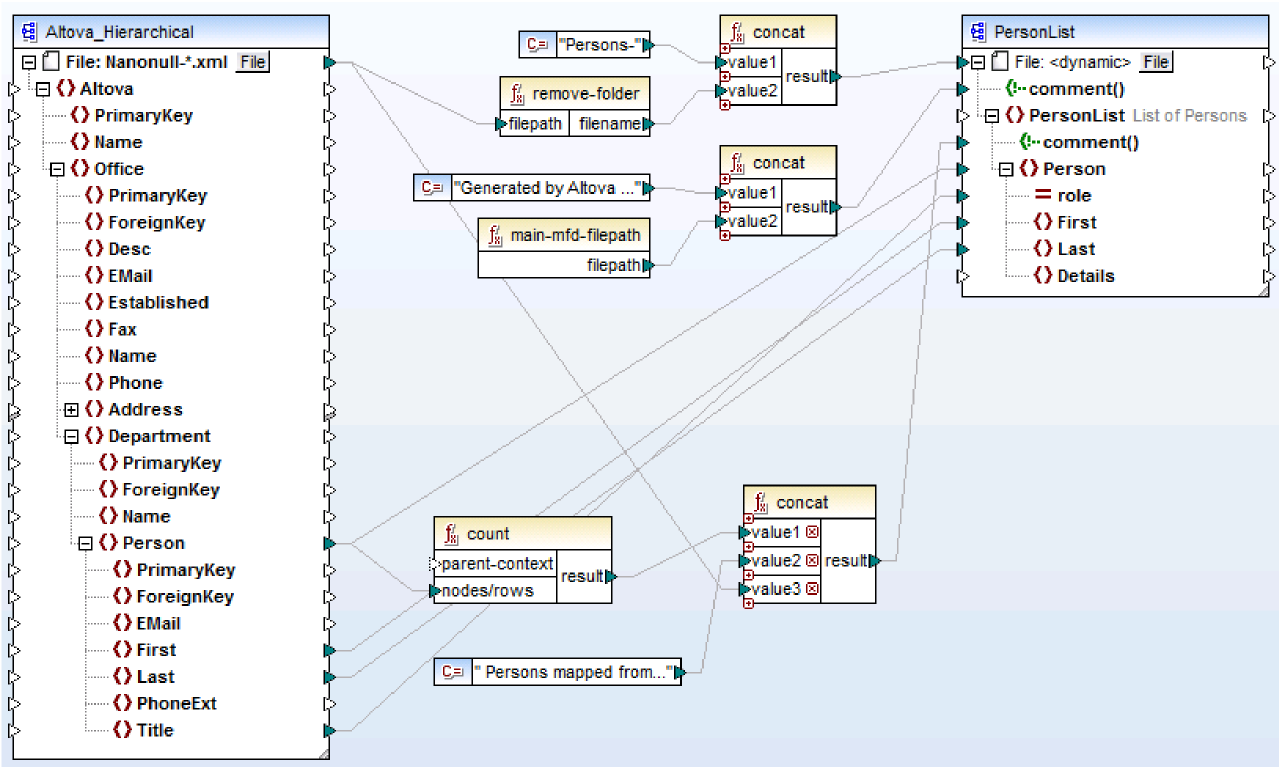


MergeMultipleFiles_List.mfd (MapForce Basic Edition)

8.5.2 Mapping Multiple Input Files to Multiple Output Files

To map multiple files to multiple target files, you need to generate unique output file names. In some cases, the output file names can be derived from strings in the input data, and in other cases it is useful to derive the output file name from the input file name, e.g. by changing the file extension.

In the following mapping, the output file name is derived from the input file name, by adding the prefix "Persons-" with the help of the **concat** function.



MultipleInputToMultipleOutputFiles.mfd (MapForce Basic Edition)

Note: Avoid simply connecting the input and output root nodes directly, without using any processing functions. Doing this will overwrite your input files when you run the mapping. You can change the output file names using functions such as the **concat** function, as shown above.

The menu option **File | Mapping Settings** allows you to define globally the file path settings used by the mapping (see Changing the mapping settings [114]).

8.5.3 Supplying File Names as Mapping Parameters

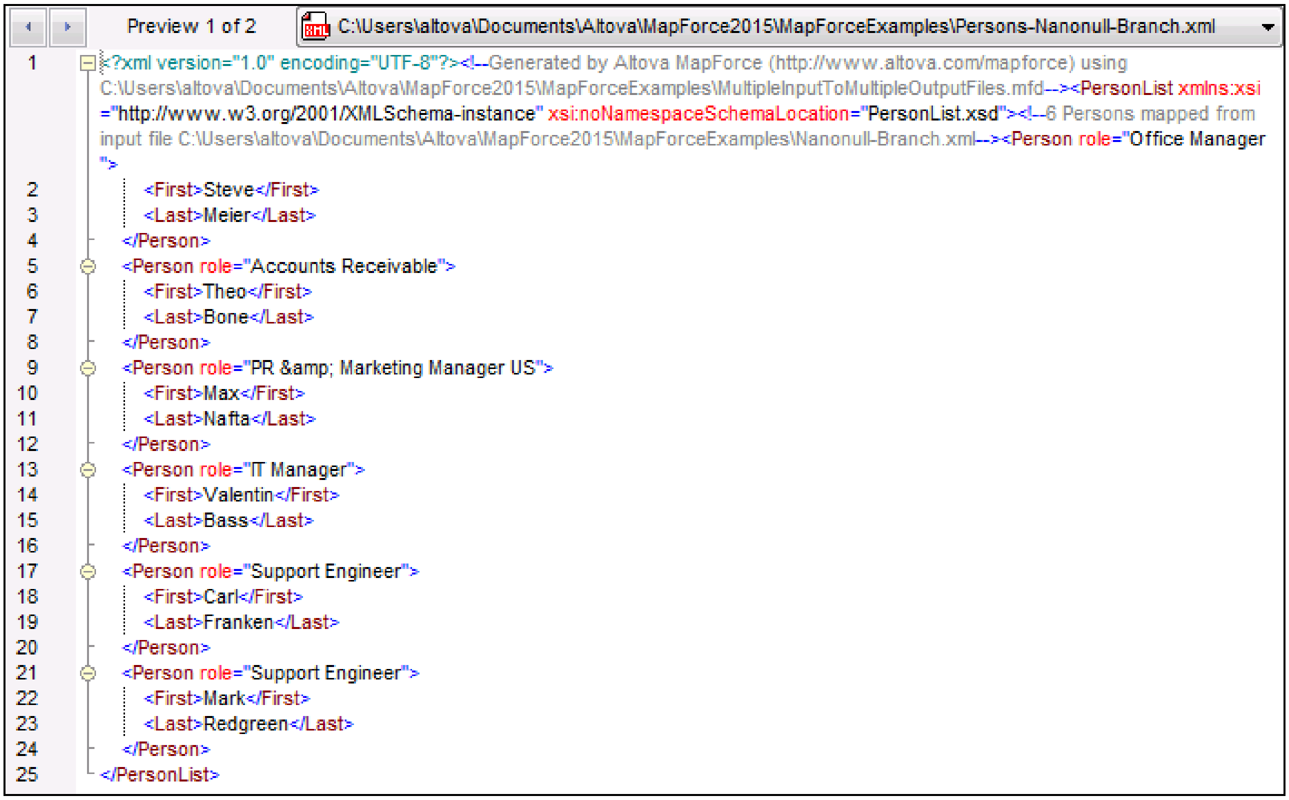
To supply custom file names as input parameters to the mapping, do the following:

1. Add a simple input component to the mapping (On the **Function** menu, click **Insert Input**). For more information about such components, see Supplying Parameters to the Mapping [833].
2. Click the **File** (File) or **File/String** (File/String) button of the source component and select **Use Dynamic File Names Supplied by Mapping**.
3. Connect the simple input component to the root node of the component which acts as mapping source.

For a worked example, see Example: Using File Names as Mapping Parameters [837].

8.5.4 Previewing Multiple Output Files

Click the Output tab to display the mapping result in a preview window. If the mapping produces multiple output files, each file has its own numbered pane in the Output tab. Click the arrow buttons to see the individual output files.




```xml
<?xml version="1.0" encoding="UTF-8"?><!--Generated by Altova MapForce (http://www.altova.com/mapforce) using C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\MultipleInputToMultipleOutputFiles.mfd--><PersonList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="PersonList.xsd"><!--6 Persons mapped from input file C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Nanonull-Branch.xml--><Person role="Office Manager">
        <First>Steve</First>
        <Last>Meier</Last>
    </Person>
    <Person role="Accounts Receivable">
        <First>Theo</First>
        <Last>Bone</Last>
    </Person>
    <Person role="PR &amp; Marketing Manager US">
        <First>Max</First>
        <Last>Nafta</Last>
    </Person>
    <Person role="IT Manager">
        <First>Valentin</First>
        <Last>Bass</Last>
    </Person>
    <Person role="Support Engineer">
        <First>Carl</First>
        <Last>Franken</Last>
    </Person>
    <Person role="Support Engineer">
        <First>Mark</First>
        <Last>Redgreen</Last>
    </Person>
</PersonList>
```

MultipleInputToMultipleOutputFiles.mfd

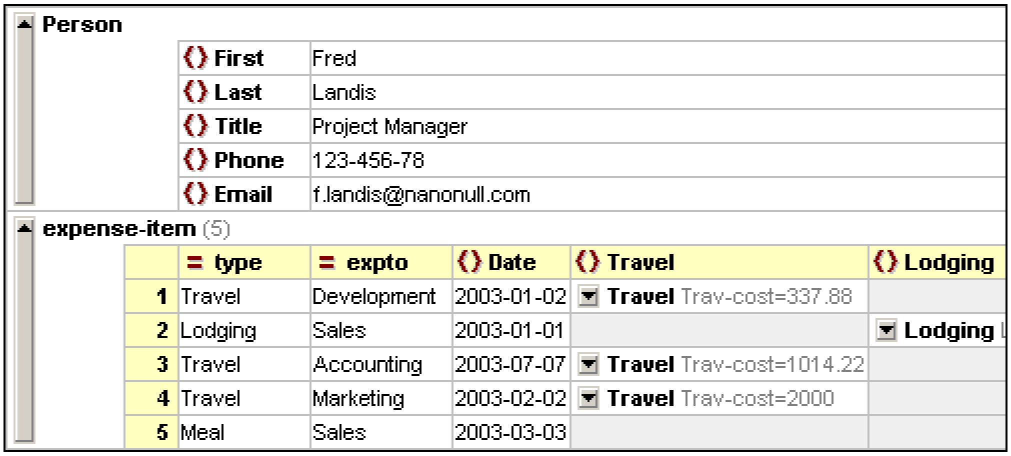
To save the generated output files, do one of the following:

- On the **Output** menu, click **Save All Output Files** ( ).
- Click the **Save all generated outputs** ( ) toolbar button.

8.5.5 Example: Split One XML File into Many

This example shows you how to generate dynamically multiple XML files from a single source XML file. The accompanying mapping for this example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\Tut-ExpReport-dyn.mfd**.

The source XML file (available in the same folder as the mapping) consists of the expense report for a person called "Fred Landis" and contains five expense items of different types. The aim of the example is to generate a separate XML file for each of the expense items listed below.



Person

First	Fred
Last	Landis
Title	Project Manager
Phone	123-456-78
Email	f.landis@nanonull.com

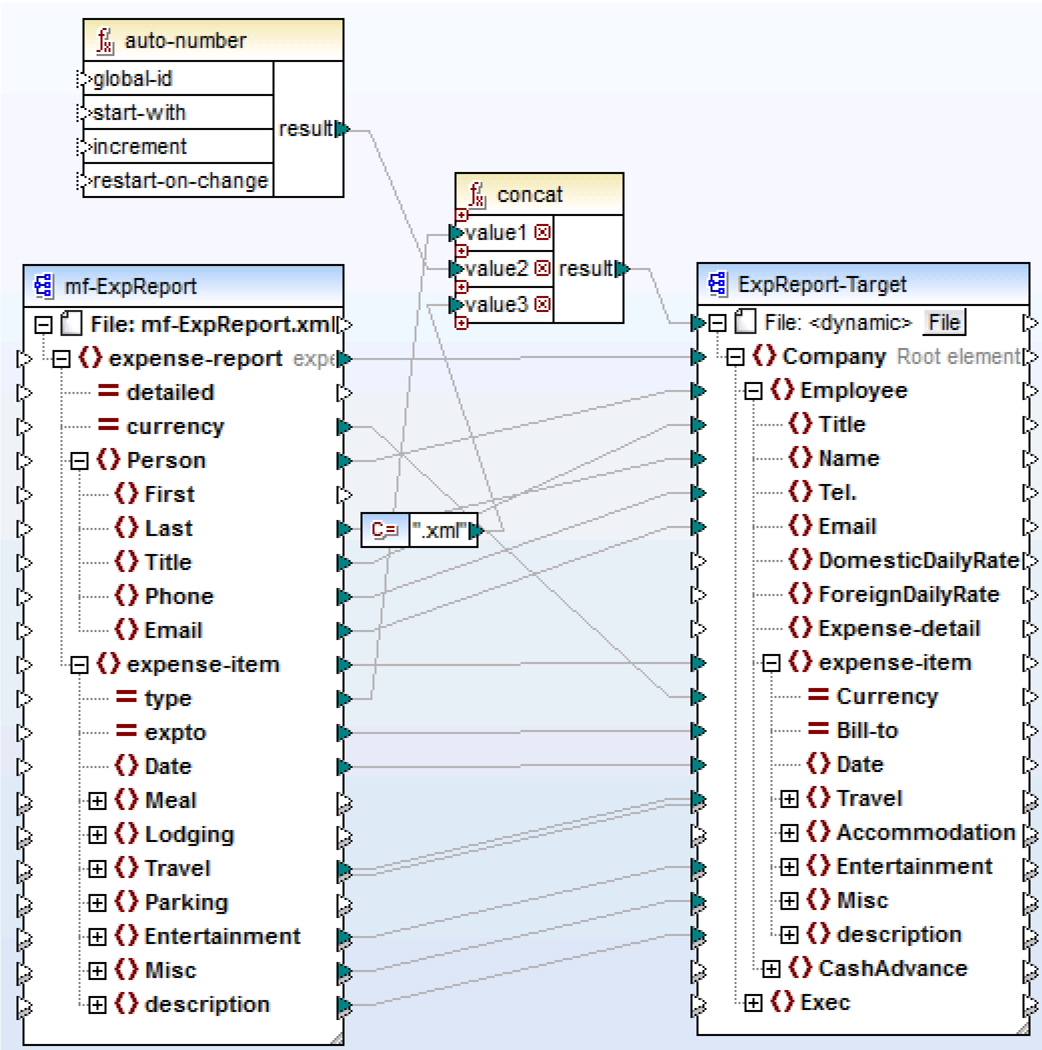
expense-item (5)

	= type	= expto	Date	Travel	Lodging
1	Travel	Development	2003-01-02	Travel Trav-cost=337.88	
2	Lodging	Sales	2003-01-01		Lodging L
3	Travel	Accounting	2003-07-07	Travel Trav-cost=1014.22	
4	Travel	Marketing	2003-02-02	Travel Trav-cost=2000	
5	Meal	Sales	2003-03-03		

mf-ExpReport.xml (as shown in XMLSpy Grid view)

As the `type` attribute defines the specific expense item type, this is the item we will use to split up the source file. To achieve the goal of this example, do the following:

1. Insert a **`concat`** function (you can drag it from the **core | string functions** library of the Libraries pane).
2. Insert a constant (on the **Insert** menu, click **Constant**) and enter ".xml" as its value.
3. Insert the **`auto-number`** function (you can drag it from the **core | generator functions** library of the Libraries pane).
4. Click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.
5. Create the connections as shown below and then click the **Output** tab to see the result of the mapping.



Tut-ExpReport-dyn.mfd (MapForce Basic Edition)

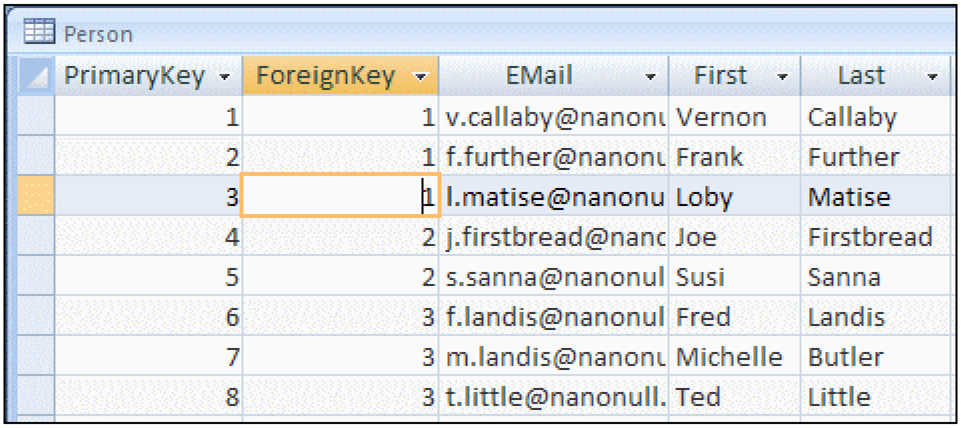
Note that the resulting output files are named dynamically as follows:

- The `type` attribute supplies the first part of the file name (for example, "Travel").
- The **`auto-number`** function supplies the sequential number of the file (for example, "Travel1", "Travel2", and so on).
- The constant supplies the file extension, which is ".xml", thus "Travel1.xml" is the file name of the first file.

8.5.6 Example: Split Database Table into Many XML Files

This example shows you how to generate dynamically multiple XML files, one for each record of a database table. The accompanying mapping for this example is available at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\PersonDB-dyn.mfd**.

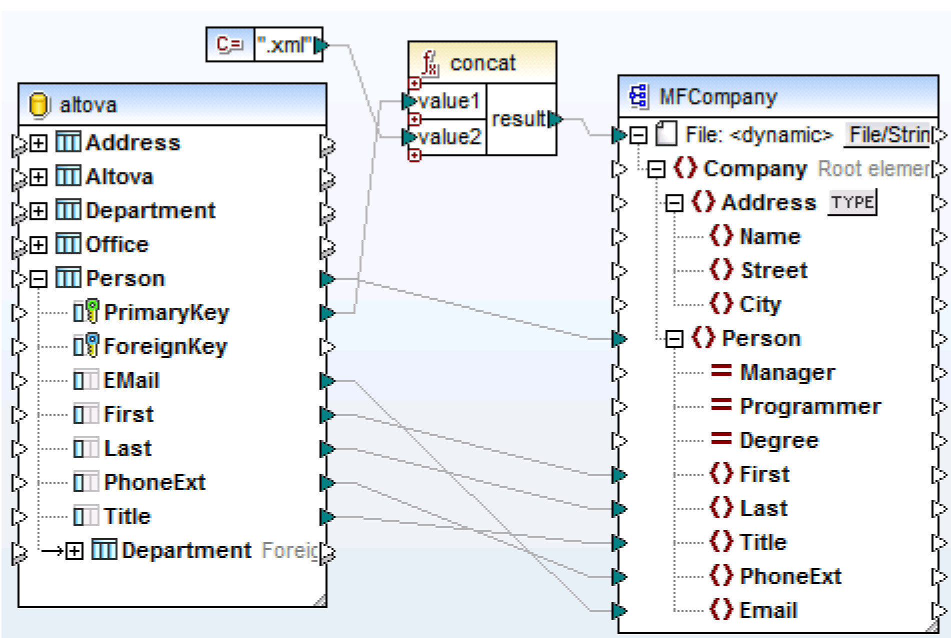
The source database file (available in the same folder as the mapping) includes a Person table which contains 21 records. The aim of the example is to generate a separate XML file for each record in the Person table.


Person

PrimaryKey	ForeignKey	EMail	First	Last
1	1	v.callaby@nanonu	Vernon	Callaby
2	1	f.further@nanonu	Frank	Further
3	1	l.matise@nanonu	Loby	Matise
4	2	j.firstbread@nanc	Joe	Firstbread
5	2	s.sanna@nanonul	Susi	Sanna
6	3	f.landis@nanonul	Fred	Landis
7	3	m.landis@nanonu	Michelle	Butler
8	3	t.little@nanonull.	Ted	Little

As the "PrimaryKey" field uniquely identifies each person in the table, this is the item we will use to split up the source database into separate files. To achieve the goal of this example, do the following:

1. Insert a **concat** function (you can drag it from the **core | string functions** library of the Libraries pane).
2. Insert a constant (on the **Insert** menu, click **Constant**) and enter ".xml" as its value.
3. Click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.
4. Create the connections as shown below and then click the **Output** tab to see the result of the mapping.



PersonDB-dyn.mfd (MapForce Professional Edition)

Note that the resulting output files are named dynamically as follows:

- The **PrimaryKey** field supplies the first part of the file name (for example, "1").
- The constant supplies the file extension (".xml"), thus "1.xml" is the file name of the first file.

8.6 Parsing and Serializing Strings

String parsing and serialization is an advanced mapping technique that enables you to configure the component to either parse data from a string, or serialize data to a string. This technique can be regarded as an alternative to reading data from (or writing data to) files. MapForce components which parse strings or serialize data to strings can be useful in a variety of situations, for example:

- You need to insert structures such as XML, text, or JSON files into database fields or Excel spreadsheet cells.
- You need to convert XML fragments stored in database fields into standalone XML files.
- You have legacy data stored as text (for example, fixed-length content in a single database field), and you would like to convert this data into a fully sortable, field-based structure (using FlexText, for example)

String parsing and serialization is available for the following MapForce component types:

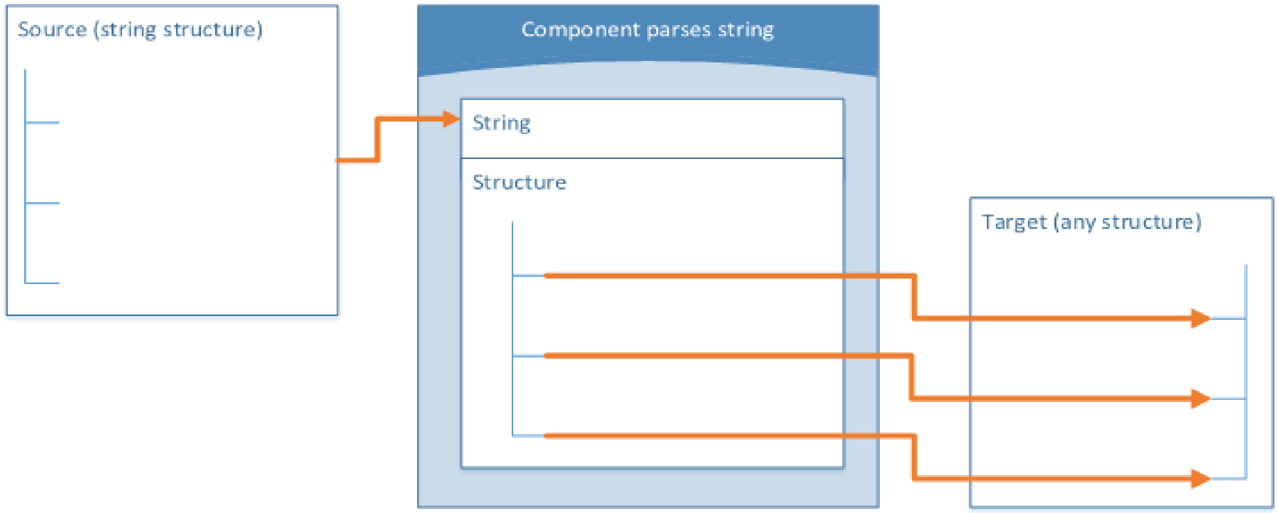
- Text (CSV, fixed-length field text, EDI, and MapForce FlexText templates)
- JSON schema files
- XML schema files

For all component types above, string parsing and serialization is supported in the BUILT-IN target language. In addition, parsing strings to JSON or serializing JSON from strings is supported in BUILT-IN, C#, and Java.

8.6.1 About the Parse/Serialize Component

A Parse/Serialize component in MapForce is a hybrid component which is neither a source nor a target component. Given the role they play in the mapping design, such components must be placed in between other source and target components.

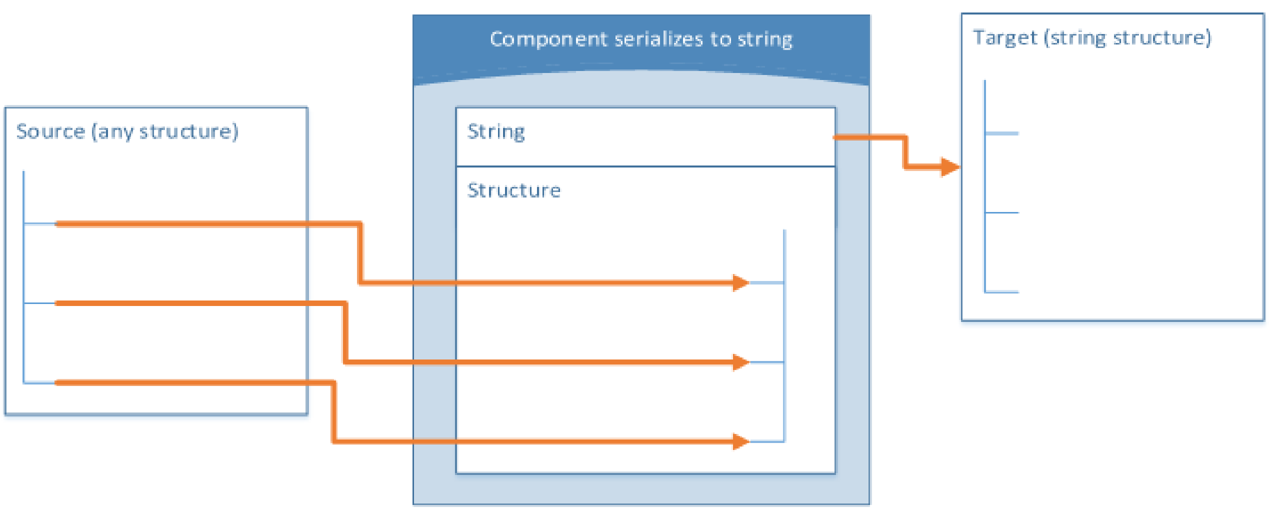
You can use a "Parse/Serialize String" component for string parsing when, for some reason, you need to convert a string that has structure (for example, some XML stored as string in a database) into another format. Parsing data from the source string to the "Parse/Serialize" component means that the source string is turned into a MapForce structure, and, thus, you get access to any element or attribute of the source XML stored as string.



Generic "Parse String" component

The diagram above illustrates the typical structure of a MapForce component which parses a string. Note that the "Parse/Serialize String" component is placed in between the source and target of the mapping. What this component does is accept some string structure as input, by means of a single MapForce connector which is connected to its top **String** node. The output structure can be any of the data targets supported by MapForce. For an example, see Example: Parse String (Fixed-Length Text to Excel) 1326.

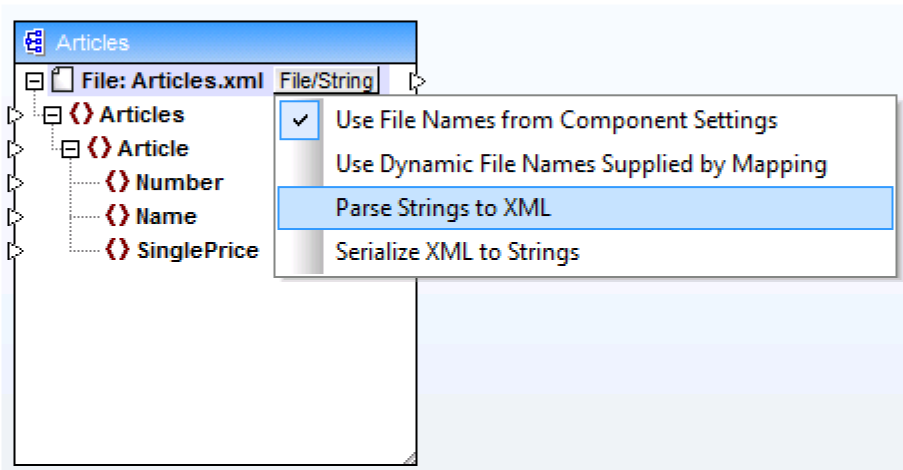
When you serialize data from a component to string, the reverse happens. Specifically, the entire structure of the MapForce component becomes a string structure which you can further manipulate as necessary. For example, this enables you to write an XML file (or XML fragment) to a database field or to a single cell of an Excel spreadsheet.



Generic "Serialize to String" component

The diagram above illustrates a generic MapForce "Serialize to String" component. What this component does is accept as input any data source supported by MapForce (by means of standard MapForce connectors). The output structure is a string which you can pass further by means of a single MapForce connector drawn from the top **String** node of the component to a target component item (for example, a spreadsheet cell). For an example, see Example: Serialize to String (XML to Database)[1331].

You can designate a component for string parsing or serialization at any time from the mapping window. To do so, click the **File/String** (File/String) button adjacent to the root node, and then select the desired option.



Changing the component mode

Note: A "Parse/Serialize String" component cannot read data from a string and write to a string simultaneously. Therefore, the root node can have either an incoming connector or an outgoing connector (not both). An error will be generated if you attempt to use the same component for both operations.

When you designate a component for string parsing or serialization, the appearance of component changes as follows:

- The component gets the **parse** or **serialize** prefix in the title.
- The title bar has yellow background color, similar to function components.
- The top node begins with the **String:** prefix and is identified by the icon.
- If the component parses a string, the output connector from the root node is not meaningful and thus it is not available.
- If the component serializes to a string, the input connector to the root node is not meaningful and thus it is not available.

When a component is in "Parse/Serialize String" mode, you can change its settings in a similar way as if it were in a file-based mode (see Changing the Component Settings[82]). Note that not all component settings are available when a component is in either "Parse" or "Serialize" mode.

8.6.2 Example: Parse String (Fixed-Length Text to Excel)

This example walks you through the steps required to create a mapping design which parses string data. The example is accompanied by a sample file. If you want to look at the sample file before starting this example, you can open it from the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\ParseString.mfd.

Let's assume a scenario where you have some legacy text data stored as a single database field. The text data is a list of employees, stored in the `RESOURCE` column, and formatted as fixed-length fields, as follows:

```
|----|-------------------------------------------------------------------------------------|------------|
| ID | RESOURCE                                                                            | TYPE       |
|----|-------------------------------------------------------------------------------------|------------|
| 1  | P00001Callaby        Vernon         582Office Manager          v.callaby@nanonull.com   | Employees  |
|    | P00002Further        Frank          471Accounts Receivable     f.further@nanonull.com   |            |
|    | P00003Matise         Loby           963Accounting Manager      l.matise@nanonull.com    |            |
|    | P00004Firstbread     Joe            621Marketing Manager Europe j.firstbread@nanonull.com |            |
|    | P00005Sanna          Susi           753Art Director            s.sanna@nanonull.com     |            |
|    | P00006Landis         Fred           951Program Manager         f.landis@nanonull.com    |            |
|    | P00007Butler         Michelle       654Software Engineer       m.landis@nanonull.com    |            |
|    | P00008Little         Ted            852Software Engineer       t.little@nanonull.com    |            |
|    | P00009Way            Ann            951Technical Writer        a.way@nanonull.com       |            |
|    | P00010Gardner        Liz            753Software Engineer       l.gardner@nanonull.com   |            |
|    | P00011Smith          Paul           334Software Engineer       p.smith@nanonull.com     |            |
|----|-------------------------------------------------------------------------------------|------------|
```

Because the text data is stored as a single database field, you cannot easily access and manipulate each individual employee record. This makes it difficult to add or remove new employees, or to sort data. For the purpose of this example, your goal is to extract the text data from the `RESOURCE` database field and split it into a structure so that you can easily process the records.

This task can be accomplished by using a "Parse/Serialize String" component. First, the "Parse/Serialize String" component will take the text data as input. Then it will parse it and convert it into a structure. Finally, it will write the structure to a target format. In this example, the target format is an Excel spreadsheet; however, in general, it can be any other output format supported by MapForce.

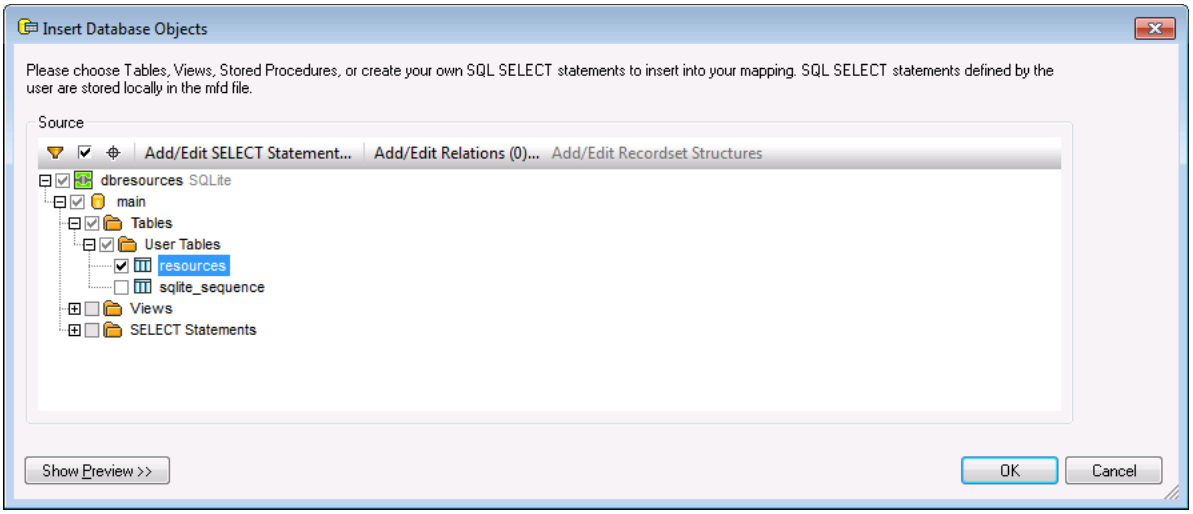
To summarize, the mapping described in this example will convert the contents of the `RESOURCE` database field to a table. After the mapping is executed, each table row will correspond to an employee and each column will correspond to one of the fixed-length fields, in this order: ID, Last Name, First Name, Extension, Job Title, Email.

	A	B	C	D	E	F	G
1	P00001	Callaby	Vernon	582	Office Manager	v.callaby@nanonull.com	
2	P00002	Further	Frank	471	Accounts Receivable	f.further@nanonull.com	
3	P00003	Matise	Loby	963	Accounting Manager	l.matise@nanonull.com	
4	P00004	Firstbread	Joe	621	Marketing Manager Europe	j.firstbread@nanonull.com	
5	P00005	Sanna	Susi	753	Art Director	s.sanna@nanonull.com	
6	P00006	Landis	Fred	951	Program Manager	f.landis@nanonull.com	
7	P00007	Butler	Michelle	654	Software Engineer	m.landis@nanonull.com	
8	P00008	Little	Ted	852	Software Engineer	t.little@nanonull.com	
9	P00009	Way	Ann	951	Technical Writer	a.way@nanonull.com	
10	P00010	Gardner	Liz	753	Software Engineer	l.gardner@nanonull.com	
11	P00011	Smith	Paul	334	Software Engineer	p.smith@nanonull.com	
12							
13							

Expected output after parsing the string

To accomplish the goal, follow the steps below:

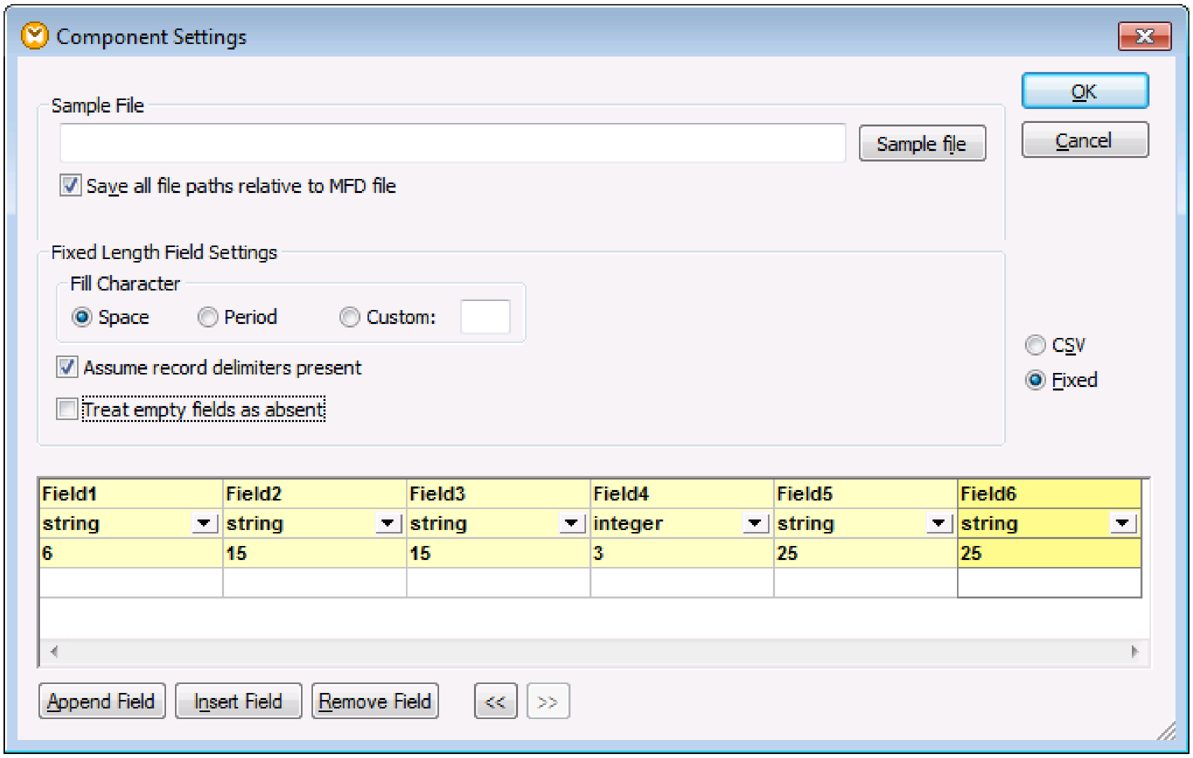
1. Add to the mapping area the source database. The database is available as a standalone SQLite database file at the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\\dbresources.db**. (To add the database component, use the **Insert | Database** menu command, see also Connecting to a Database 163).
2. When prompted to insert the database objects, select the **resources** table.



3. Add to the mapping area a Text component (use the **Insert | Text File** menu command). Since the source data is field-delimited text, choose **Use simple processing...** when prompted by MapForce.
4. Configure the structure of the Text component to map to the structure of the source text data stored in the database. As you may have already noticed, in this example, the source text consists of six fixed-length fields of fixed size, as follows:

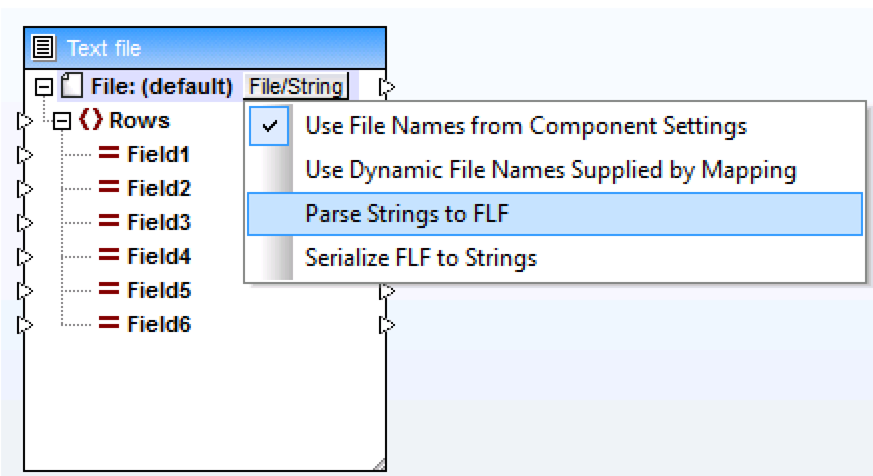
Size in characters	Field
6	ID
15	Surname
15	First Name
3	Extension
25	Job Title
25	Email

To achieve this, declare the Text component as **Fixed**, and add to it six fields that correspond to positions above. (To open the Component Settings dialog box, right-click the component, and then select **Properties** from the context menu.)

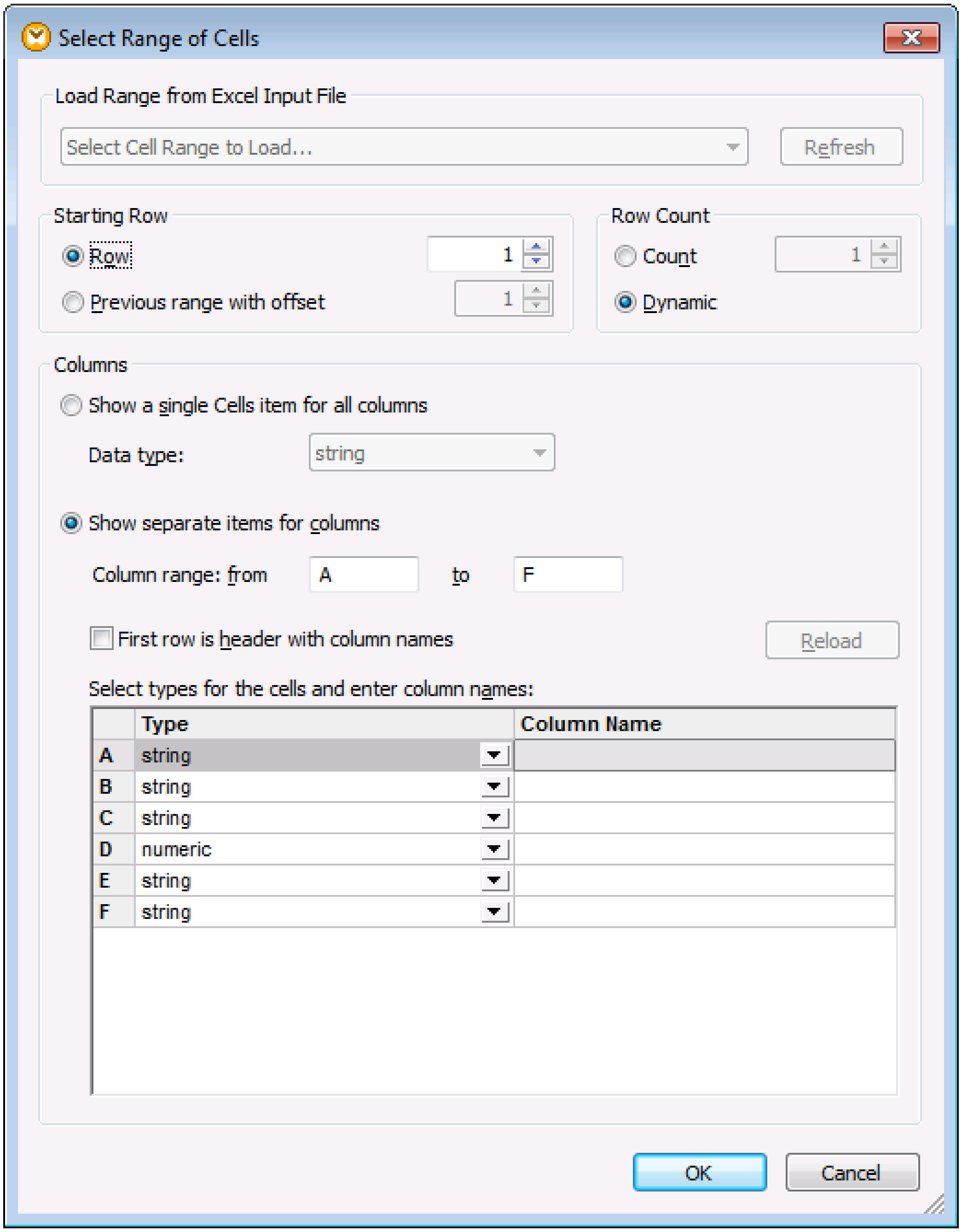


Observe the **Field4** field, which uses **integer** as data type. Although declaring the **Field4** as numeric type is optional for the scope of the current example, this ensures that the phone extension (**Field4**) extracted from the source text is validated as a numeric value.

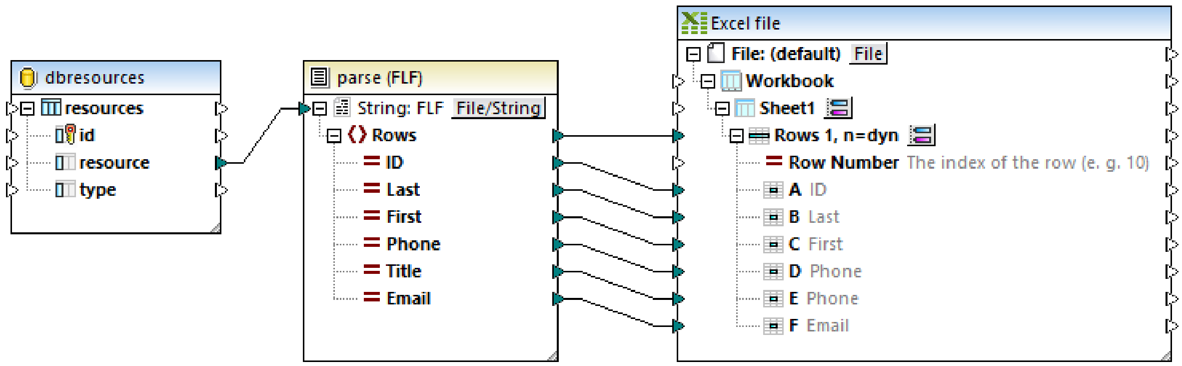
5. Click File/String the, and then select **Parse Strings to FLF** from the context menu. This instructs MapForce that this component will parse a string to fixed-length field (FLF) format.



6. Add to the mapping area the target Excel 2007+ component. When prompted to select a sample file, click **Skip**. (You can add the Excel component using the **Insert | Excel 2007+** menu command, see also Adding Excel 2007+ Files as Mapping Components [717]).
7. Click the button next to **Row 1, n=dyn** node, and configure the Excel component to write a row for each text field, starting with the first row, as shown below. (For more information about Excel 2007+ component types and their configuration, see About the Excel 2007+ Component [718]).



8. Draw the connections between component items, as shown below.



On the left side of the mapping, the contents of the resource database column is being converted (parsed) from a string value to a MapForce structure. On the right side of the mapping, the items of the *Parse (Text file)* component are connected to individual Excel columns, thus splitting the source string into individual sortable cells.

You have now finished creating a MapForce design file which parses string data and creates a structure from it. If you click the **Output** tab, the legacy text data is now converted to individual rows and columns of the Excel spreadsheet, which was the intended goal of this mapping.

8.6.3 Example: Serialize to String (XML to Database)

This example walks you through the steps required to create a mapping design which serializes data to a string. The example is accompanied by a sample file. If you want to look at the sample file before starting this example, you can open it from the following path:
<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\SerializeToString.mfd.

Let's assume you have an XML file (and its related schema) which consists of multiple `<Person>` elements. Each `<Person>` element describes a person's first name, last name, job title, phone extension, and email address, as follows:

```
<Person>
      <First>Joe</First>
      <Last>Firstbread</Last>
      <Title>Marketing Manager Europe</Title>
      <PhoneExt>621</PhoneExt>
      <Email>j.firstbread@nanonull.com</Email>
</Person>
```

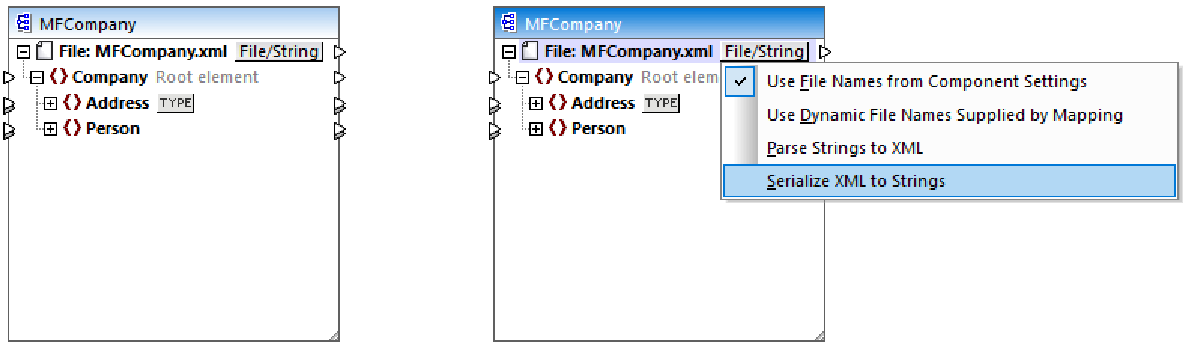
Your goal is to extract each `<Person>` element from the XML file and insert it literally (including XML tags) as a new database record in the `PEOPLE` table of a SQLite database. The `PEOPLE` table contains only two columns: `ID` and `PERSON`. Its full definition is as follows:

```
CREATE TABLE PEOPLE (ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, PERSON TEXT);
```

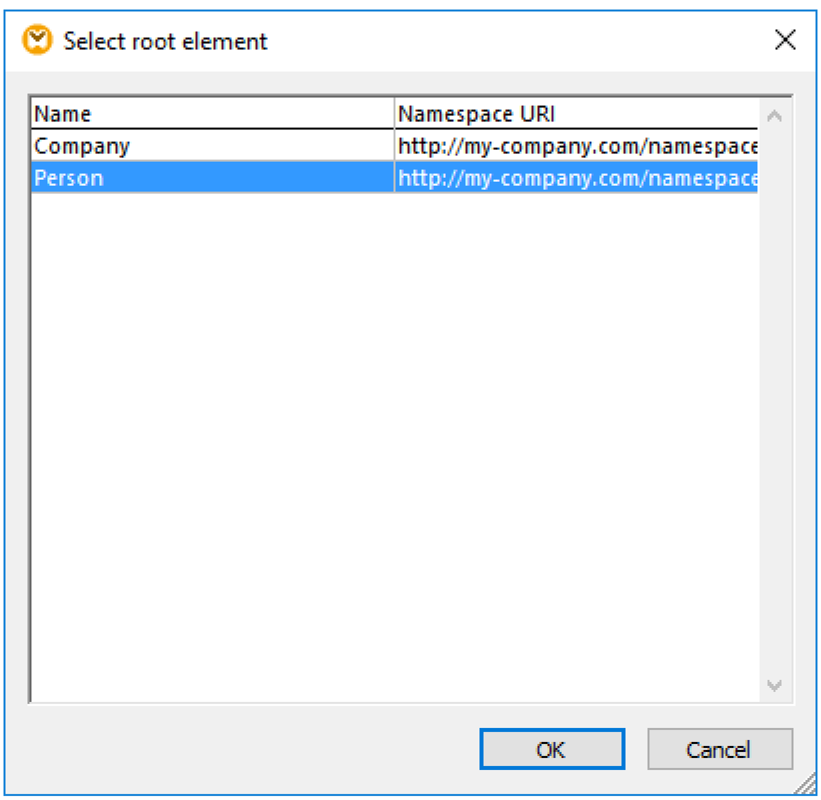
After the mapping is executed, the expected result is that the `PEOPLE` table will have the same number of rows as the number of <Person> elements in the XML file.

To achieve the goal, do the following:

1. Add to the mapping area the source XML component (use the **Insert | XML Schema/File** menu command). The sample file is available at: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\MFCompany.xml**.
2. Duplicate (copy-paste) the XML component.
3. On the duplicated XML component, click File/String, and then select **Serialize XML to Strings**.

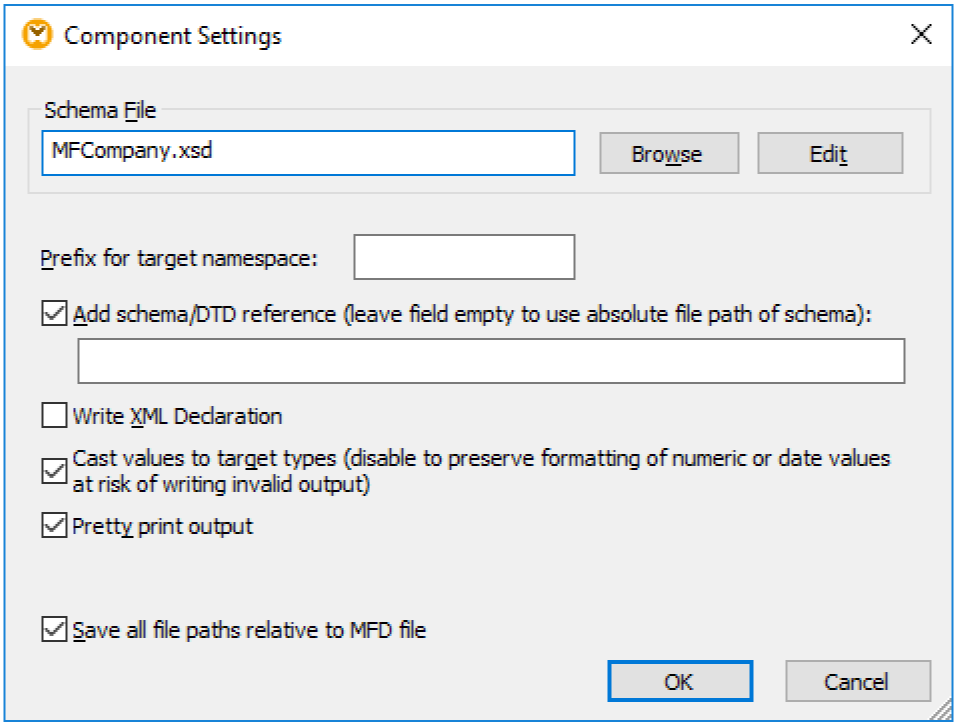


4. Right-click the duplicated component and select **Change Root Element** from the context menu. Then change the root element to `<Person>`.

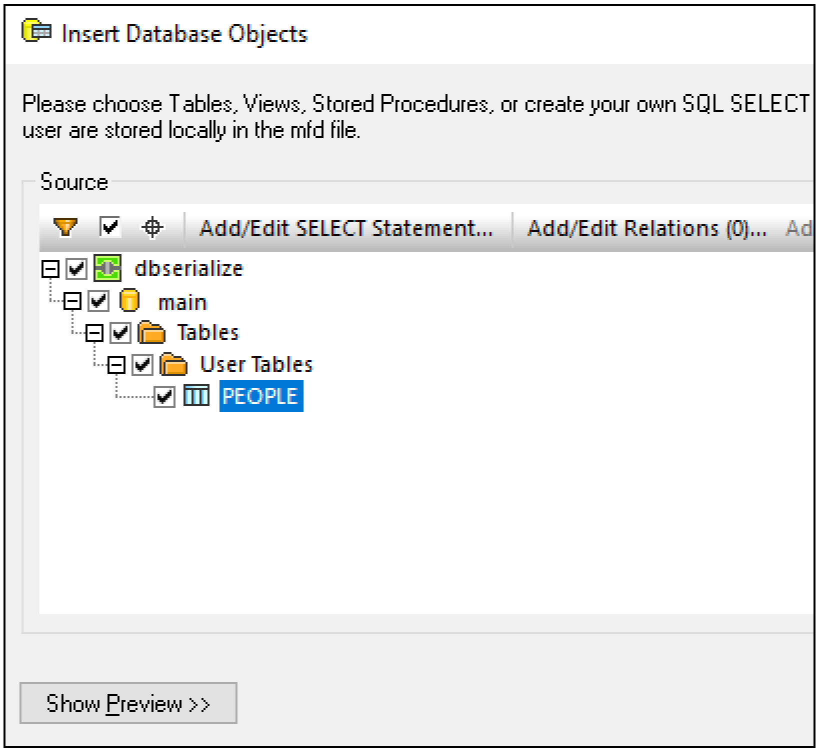


In general, you can change the root element to any element that has a global (not local) declaration in the XML schema. Any elements that are not defined globally in your schema are not listed in the "Select Root Element" dialog box.

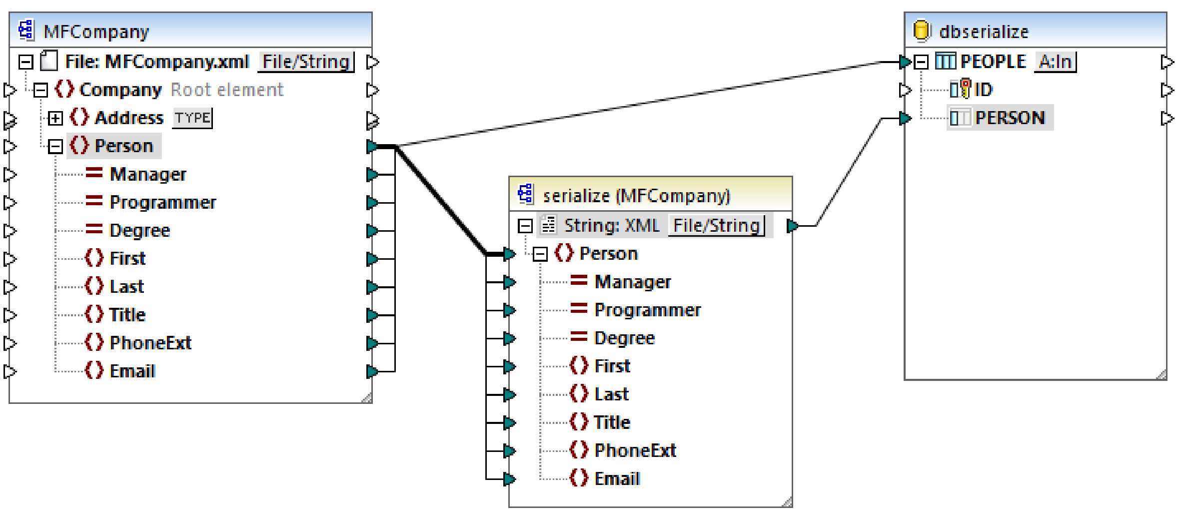
5. Double-click the component and clear the **Write XML Declaration** check box. This prevents the XML declaration from being written for each `<Person>` element.



6. Add to the mapping area the target SQLite database component, from the following path: **<Documents>\Altova\MapForce2023\MapForceExamples\Tutorial\\dbserialize.db**. (To add the database component, use the **Insert | Database** menu command, see also Connecting to a Database 163). When prompted to insert a database object, select the `PEOPLE` table.

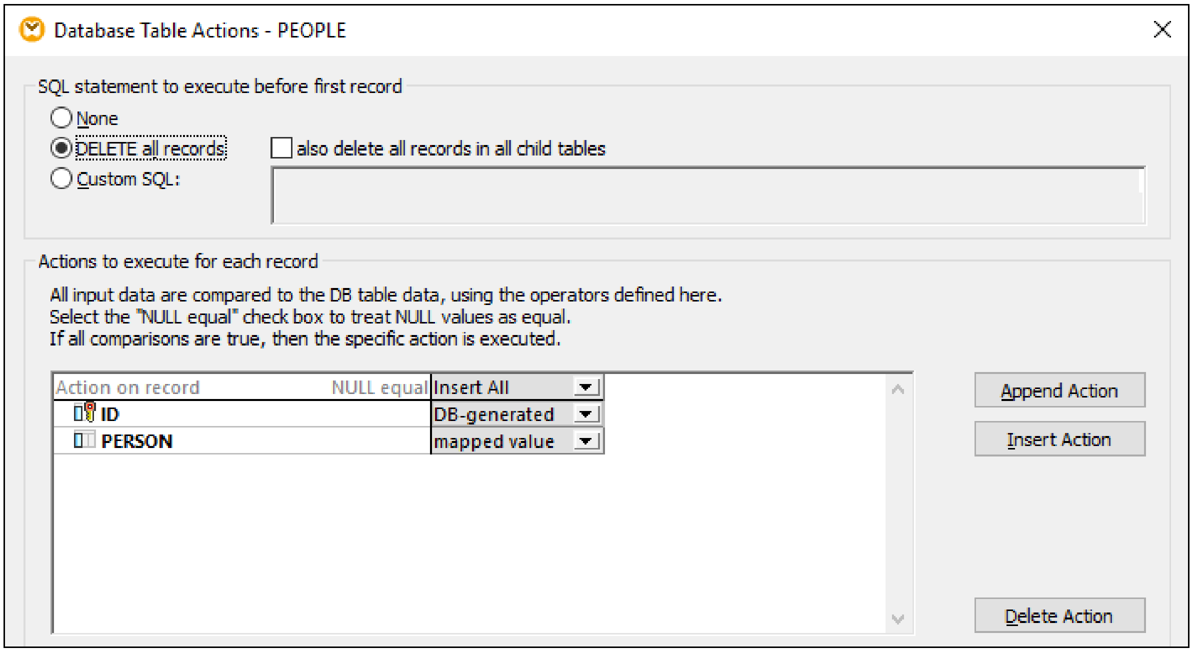


7. Link the components as shown below. On the left side of the mapping, the `<Person>` element maps to the serialization component. On the right side of the mapping, the serialized string value is inserted into the `PERSON` column of the `PEOPLE` database table. Finally, the connector drawn from `<Person>` to the `PEOPLE` table instructs MapForce to create a new record for each `<Person>` element encountered.



8. Click the **A:In** button on the database component, and do the following:
   a. Select the **Delete all records** option. At mapping runtime, this will delete any existing records from the database before new ones are inserted.
   b. Select the **DB-generated** option next to the **ID** column. This ensures that the ID of the record will be generated by the database. Note that the **DB-generated** option appears only if the column supports this option. For columns that are not an identity or auto-incremented field, the **max+1**

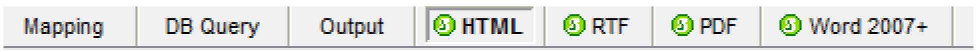
option is available instead—this option will check what is the maximum value already existing in that column, and insert the next available integer, incremented by 1.



You have now created a mapping design which serializes data to string. If you click the **Output** tab, the preview SQL query indicates that separate records will be inserted into the database for each `<Person>` element in the XML file, which was the goal of this mapping.

8.7 StyleVision Output Panes

In mappings where the target component is XML or XBRL, it is possible to preview and save the mapping output as HTML, RTF, PDF, Word 2007+, and Text documents if Altova StyleVision is installed on your computer. If you are using StyleVision Enterprise Edition, charts will also be rendered in these previews. When a mapping supports preview in any of these formats, additional panes become available next to the **Output** pane (*see screenshot below*).



Important

- When StyleVision Professional is installed, it is possible to preview **HTML**, **RTF**, and **Text** outputs.
- With StyleVision Enterprise, it is possible to preview **HTML**, **RTF**, **PDF**, **Word 2007+**, and **Text** outputs.
- Previewing the mapping output as PDF requires Java, Acrobat Reader, and FOP (Formatting Objects Processor) version 0.93 or 1.0. FOP is installed together with StyleVision unless you opted not to install it when installing StyleVision.
- In the 64-bit edition of MapForce, the Word 2007+ and RTF previews are opened as non-embedded applications.
- If your mapping contains components that act both as sources and targets (pass-through components), the StyleVision preview will be possible only for those components where the **Preview** button of the component has been activated. For more information about such mappings, see Chained Mappings 1246.

To preview your mapping data in the StyleVision output panes, the following is required:

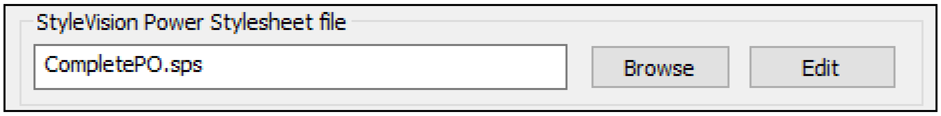
- Altova StyleVision must be installed on your computer either as a standalone installation or as part of Altova MissionKit.
- The target component must have a StyleVision Power Stylesheet (SPS) file associated with it. The stylesheet file can be created or edited with StyleVision. You cannot edit or change the stylesheet in MapForce directly, but you can open it via MapForce in StyleVision. Once the stylesheet is ready, you can assign it to a target MapForce component, as shown below.

StyleVision output panes configuration

The instructions below will help you set up StyleVision output panes.

Assign a StyleVision Power Stylesheet to a target component
To assign an SPS file to a target component, take the following steps:

1. In StyleVision, create the required stylesheet file. Make sure to use the same XML schema as a source as that of the MapForce component.
2. In MapForce, right-click the target XML or XBRL component and select **Properties**.
3. In the **Component Settings** dialog box, next to **StyleVision Power Stylesheet file**, browse for the stylesheet file created previously (*see screenshot below*).



Note: The path to the StyleVision Power Stylesheet file can be absolute or relative. For details, see Relative and Absolute Paths 85.

Save the StyleVision-generated output
You can save the StyleVision-generated output to a file in a similar way as saving the result of any other mapping: Click the toolbar button (**Save generated output**) or go to the **Output** menu and click **Save Output File**.

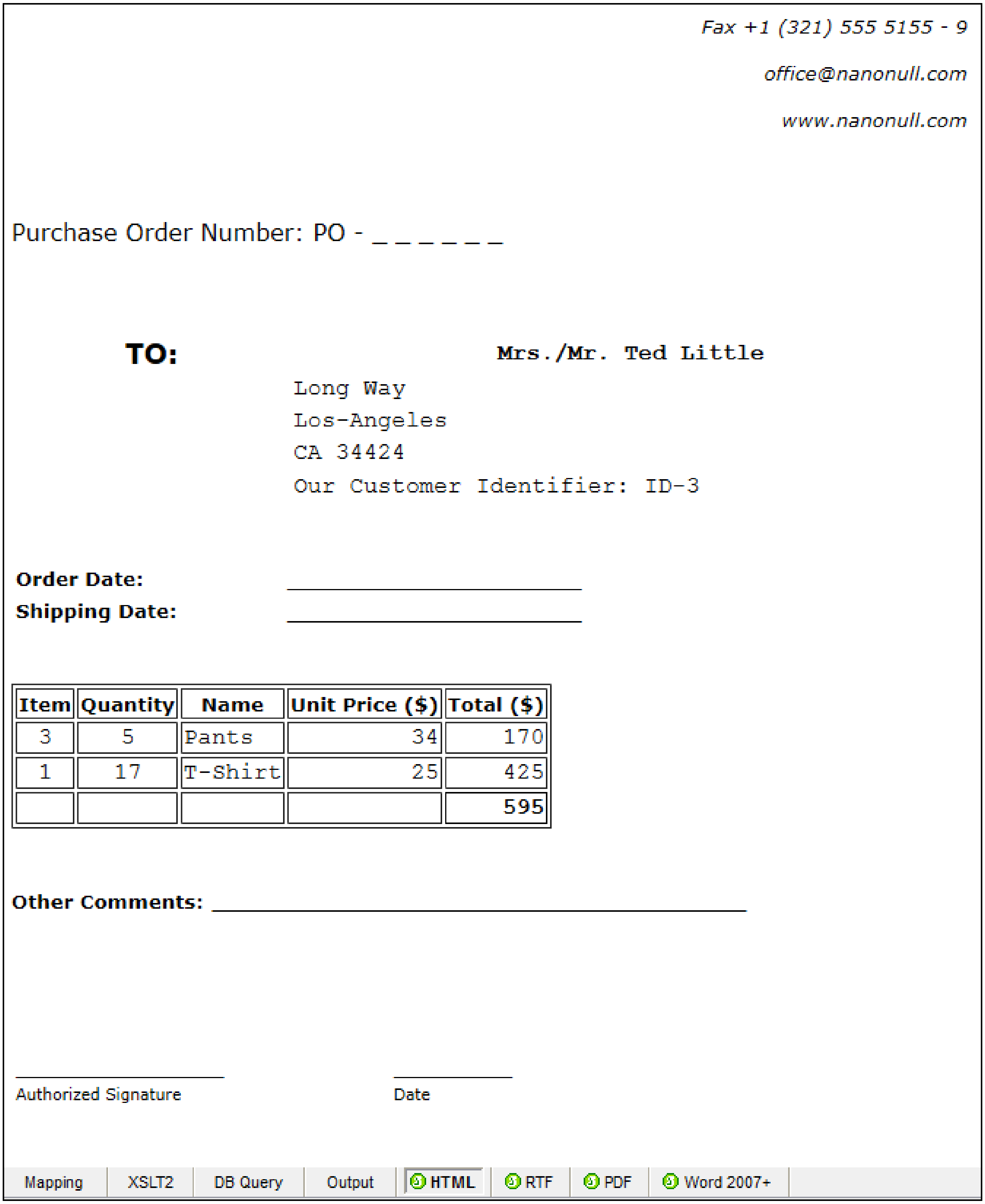
Automate generation of different formats with Altova products
If you need your mapping to generate HTML, PDF, RTF, Word 2007+, and Text files automatically (either on the same or on a different computer or even platform), you can use MapForce Server or StyleVision Server. These are separately licensed server products that extend the functionality of MapForce and StyleVision, respectively. In this scenario, each application plays its own specific role:

- MapForce enables you to design a mapping (**.mfd**) which defines the data transformation inputs and outputs (for example, database to XML)
- MapForceServer runs the executable mapping (**.mfx**) from the command line or an API (on the same or a different operating system)
- StyleVision enables you to design the stylesheet (**.sps**) required to transform the mapping output to HTML, PDF, RTF, Word 2007+, and Text files.
- StyleVision Server runs the **.sps** stylesheet which transforms the mapping output to a desired format. This happens in the command line or from an API (on the same or a different operating system).
- Both StyleVision Server and MapForce Server can optionally run under the management of FlowForce Server (licensed separately). In this scenario, MapForce mappings and StyleVision transformations can run as scheduled, triggered, or on-demand jobs. This means that these MapForce mappings and StyleVision transformations can be fully automated.

Examples

The example below (**MapForceExamples\CompletePO.mfd**) shows the output in the StyleVision output pane called **HTML**. This mapping produces a purchase order in XML format. Right-click the target component, select **Properties**, and notice that it has a **.sps** file assigned to it.

If you click the **HTML** pane, you will see the following output:


Fax +1 (321) 555 5155 - 9

office@nanonull.com

www.nanonull.com

Purchase Order Number: PO - _ _ _ _ _ _

TO: **Mrs./Mr. Ted Little**

Long Way
Los-Angeles
CA 34424
Our Customer Identifier: ID-3

Order Date: ____________________

Shipping Date: ____________________

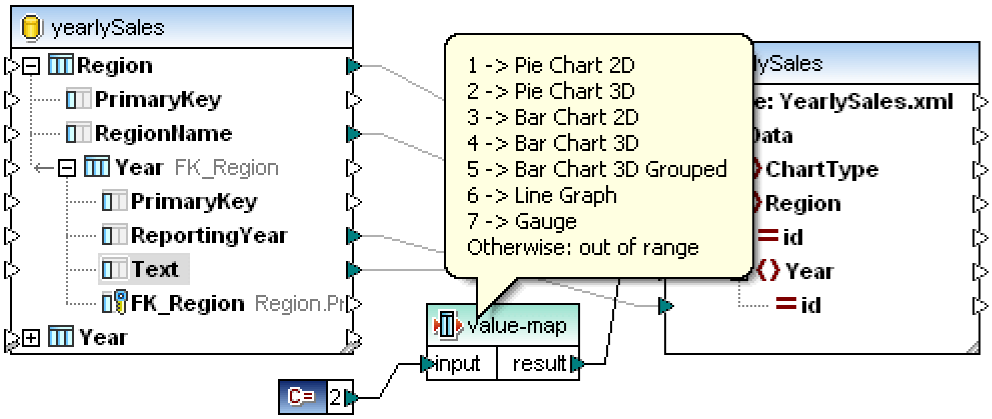
Item	Quantity	Name	Unit Price ($)	Total ($)
3	5	Pants	34	170
1	17	T-Shirt	25	425
				595

Other Comments: ____________________________________

Authorized Signature

Date

Another example is **`Tutorial\YearlySales.mfd`**. The stylesheet assigned to this mapping was designed in StyleVision in such a way that it is possible to control the type of the chart by changing the value of the `ChartType` element. This makes it possible to change the chart type directly from the mapping: You can change the default value of the constant to any value from 1 to 7. If you place the mouse cursor over the `value-map` component, you will see the possible values (*see screenshot below*).



The default value of the constant is 2, which generates a 3D pie chart in the output. To display other chart types, change this value to any other allowed value and click the **Output** pane to see the changes.

8.8 Generating Mapping Documentation

You can generate detailed documentation about any mapping in HTML, Microsoft Word (.doc), or RTF format. If StyleVision is installed, you can additionally generate documentation in PDF format.

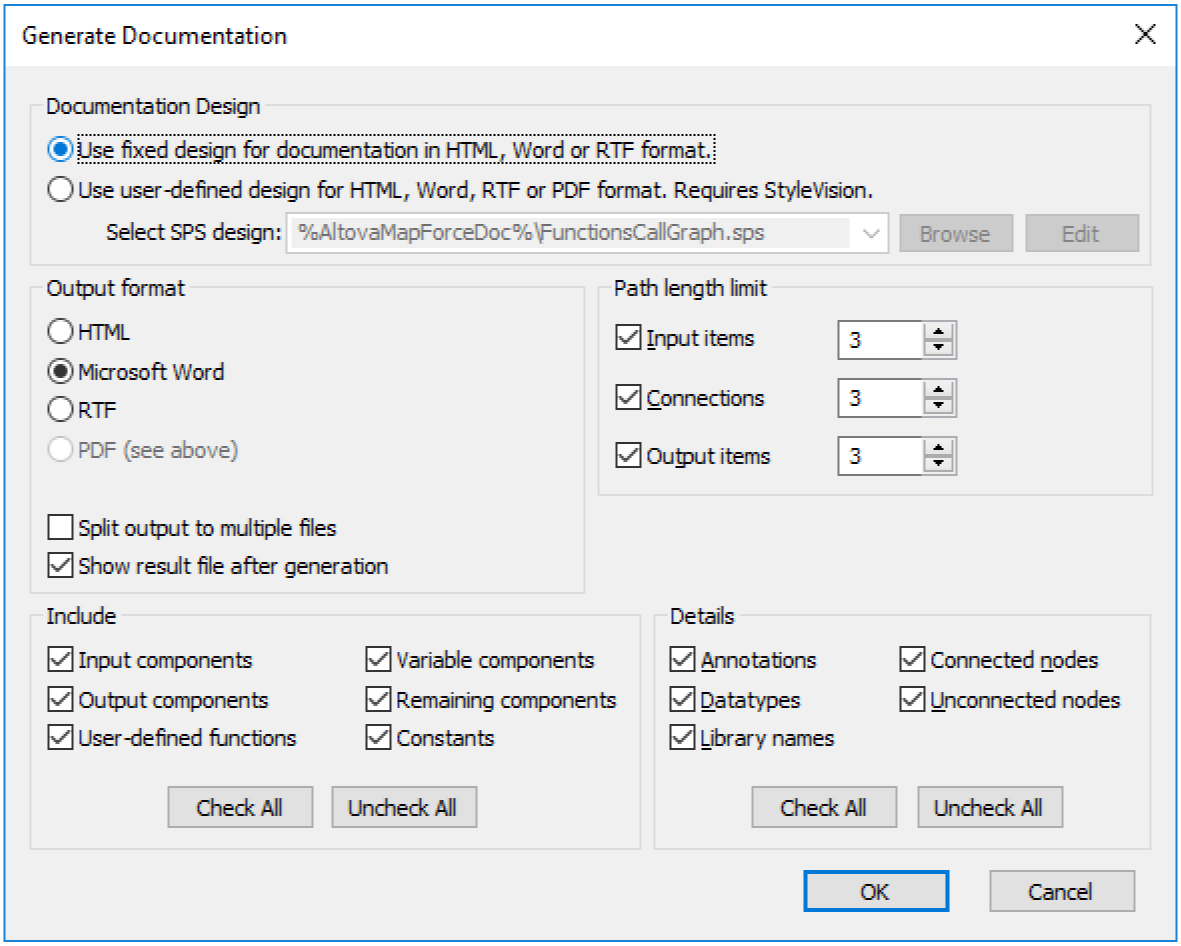
Prerequisites

- Microsoft Word 2000 or later must be installed if you would like to generate documentation in Microsoft Word format.
- StyleVision must be installed if you would like to generate documentation in PDF format or customize the design of the generated documentation.

By default, documentation is generated with a fixed design, where you can configure basic options such as the components to include, the depth of displayed paths, and other settings. If StyleVision is installed, you can additionally benefit from several included StyleVision Power Stylesheets (SPS) files, or even create your own design in StyleVision.

To generate mapping documentation:

1. On the **File** menu, click **Generate Documentation**. This opens the "Generate documentation" dialog box.



2. Select the required settings and click **OK**.

The settings you can configure are described below.

Documentation Design

- Select "Use fixed design..." to use the built-in documentation template.
- Select "Use user-defined..." to use a predefined StyleVision Power Stylesheet created in StyleVision. The SPS files are available in the **...\Documents\Altova\MapForce2023\Documentation\MapForce** folder. For details, see Predefined StyleVision Power Stylesheets [1342].
- Click **Browse** to browse for a predefined SPS file.
- Click **Edit** to launch StyleVision and open the selected SPS in a StyleVision window.

Output Format

- Select one of the following output formats: HTML, Microsoft Word, RTF, or PDF. Microsoft Word documents are created with the **.doc** file extension when generated using a fixed design, and with a **.docx** file extension when generated using a StyleVision SPS. The PDF output format requires StyleVision and is available only if you selected a StyleVision SPS.
- Select **Split output to multiple files** if you would like to generate multiple documentation files, one file for each individual component such as input or output component. If using a fixed design, links between multiple documents are created automatically.

- If the **Show result file after generation** option is selected, MapForce will open the generated files in the default browser or application, as applicable.

Path length limit

Use these options to define the maximum path length to be shown for input or output items or connections. For example, with the default length **3**, an item path would be shown as **.../ShortPO/LineItems/LineItem**.

Include

Select here the specific components that should be included in the generated documentation.

Details

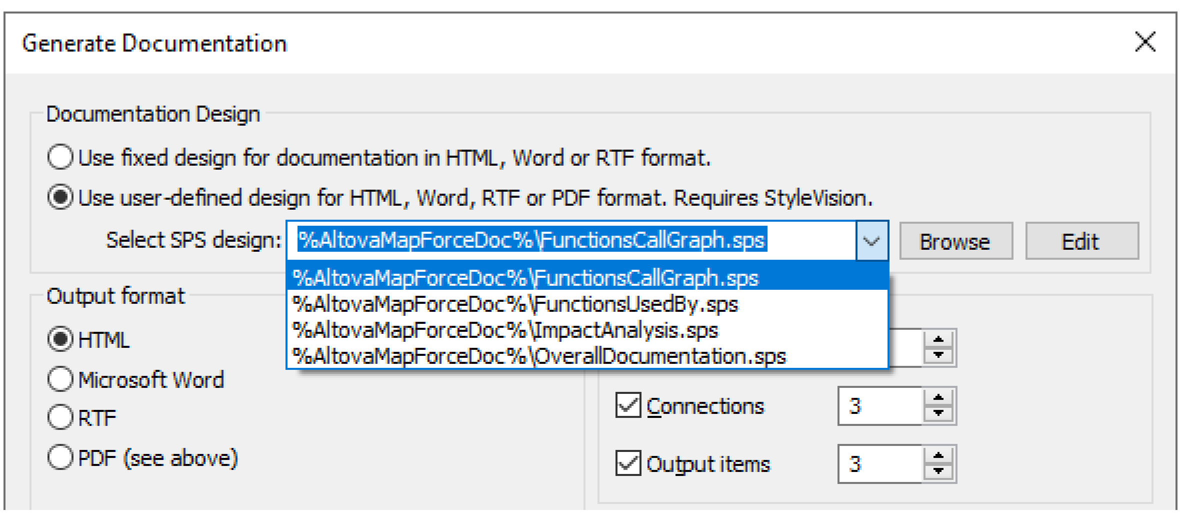
Use these options to customize the level of detail in the generated documentation. The **Library Names** option inserts the "core" prefix for functions.

8.8.1 Predefined StyleVision Power Stylesheets

When StyleVision is installed on your computer, you can generate mapping documentation by selecting one of the predefined StyleVision Power Stylesheet (SPS) files as template, instead of the built-in fixed design. The following predefined SPS stylesheets are available:

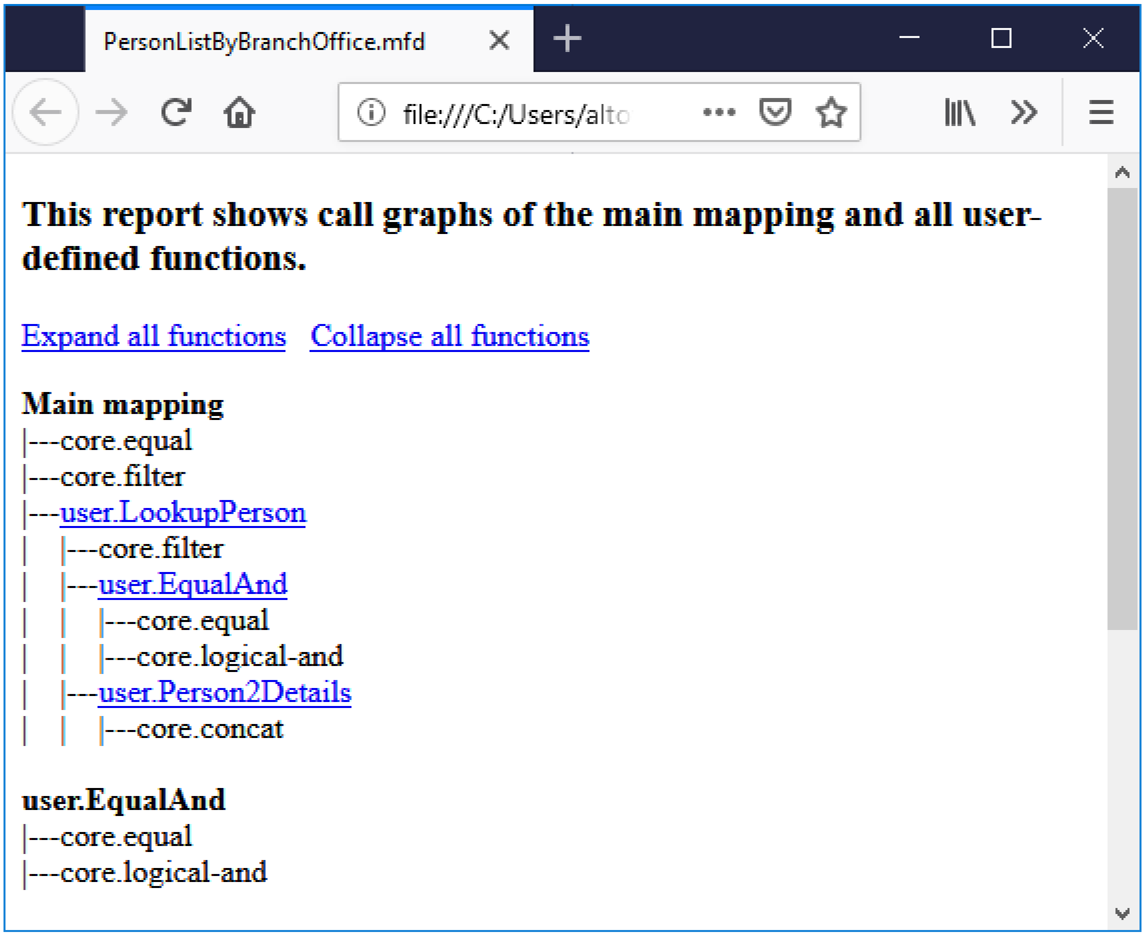
- **FunctionCallGraph.sps** - shows the call graph of the main mapping and any user-defined functions.
- **FunctionsUsedBy.sps** - shows which functions are used directly or indirectly in the mapping.
- **ImpactAnalysis.sps** - lists every source and target node, and the route taken via various functions to the target node.
- **OverallDocumentation.sps** - shows all nodes, connections, functions, and target nodes. This template outputs the maximum detail and is identical to the built-in "fixed design" output.

You can select the required stylesheet each time before generating documentation, as shown below. The files are located in the **...\MapForce2023\Documentation\MapForce** folder.

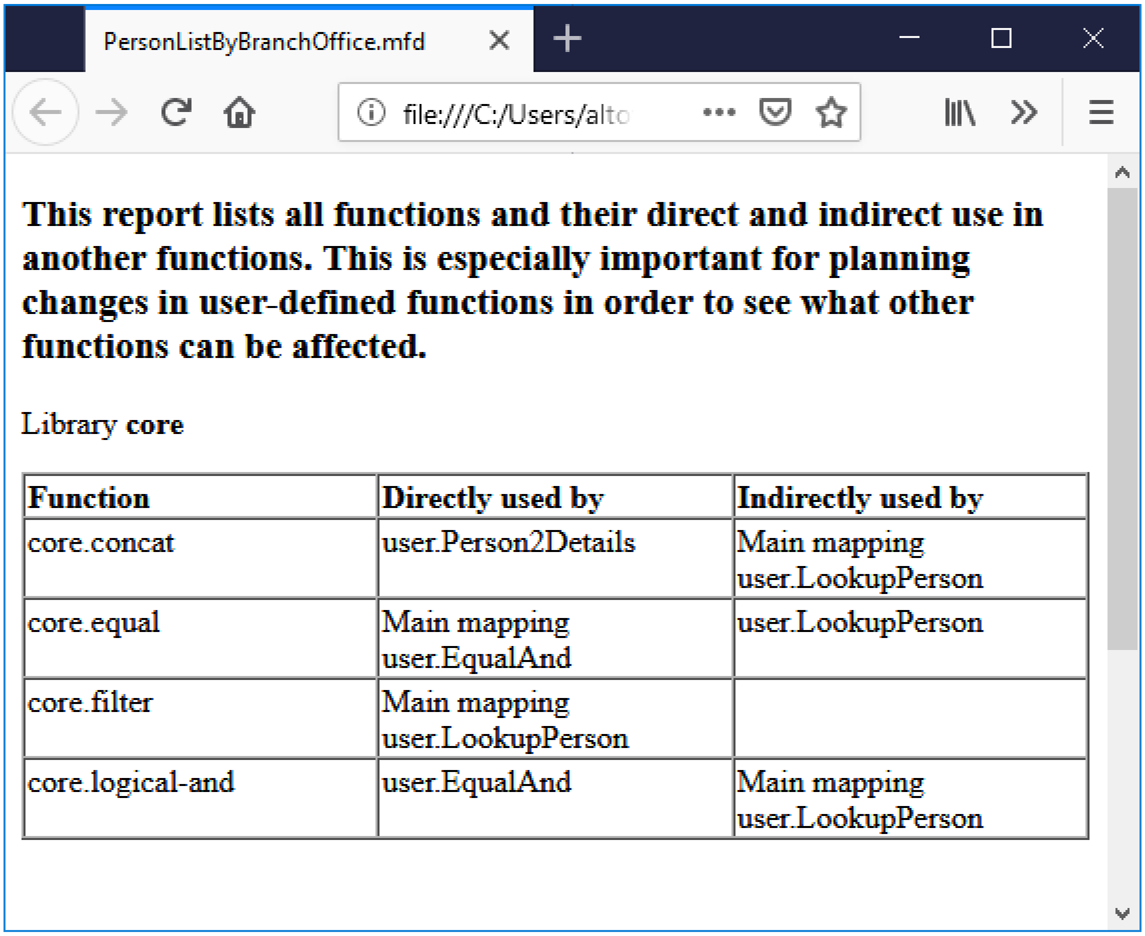


The examples below illustrate output produced by each of these stylesheets. The examples were generated from one of the demo mappings installed with MapForce, **PersonListByBranchOffice.mfd**. Although these examples illustrate HTML output specifically, the layout is similar with other formats. For information about creating or customizing SPS files, see Custom Stylesheets 1346.

Stylesheet "FunctionCallGraph.sps"



Stylesheet "FunctionsUsedBy.sps"

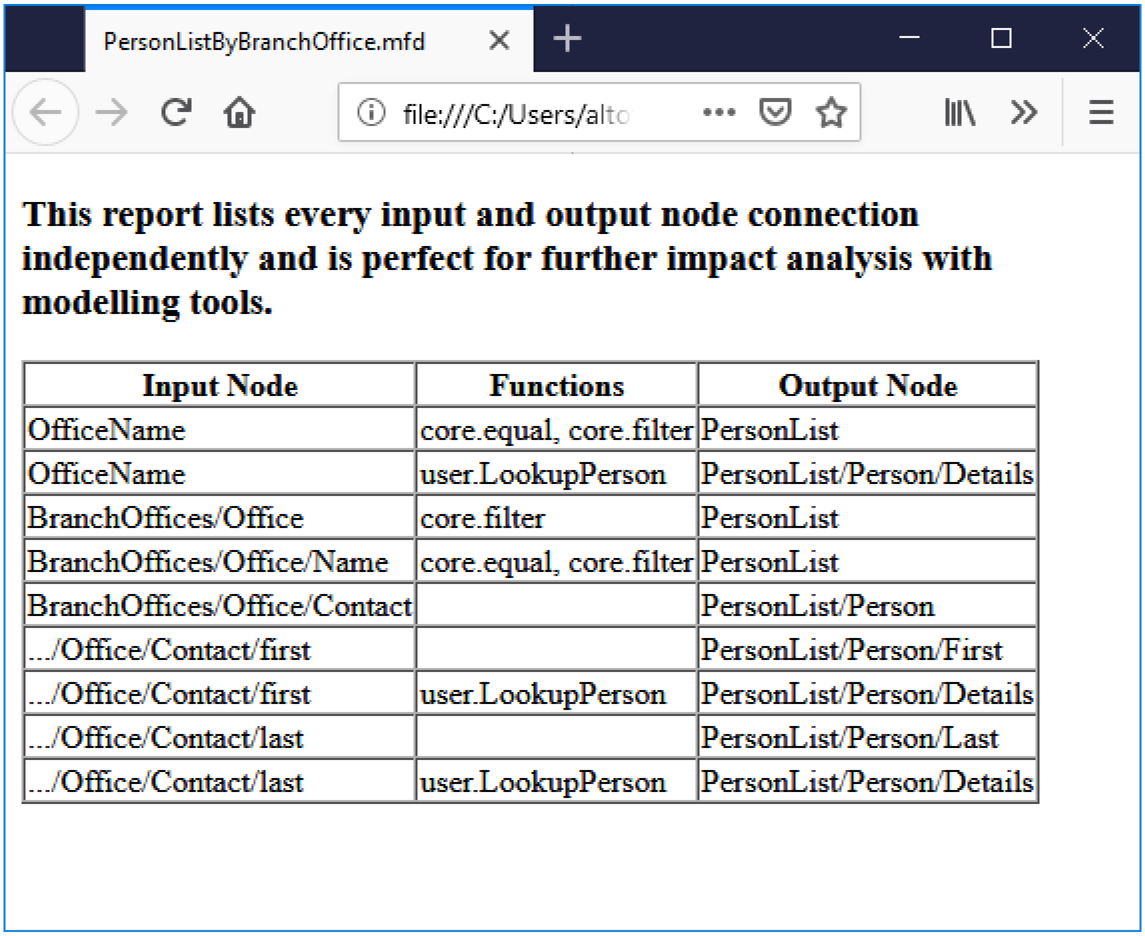


This report lists all functions and their direct and indirect use in another functions. This is especially important for planning changes in user-defined functions in order to see what other functions can be affected.

Library **core**

Function	Directly used by	Indirectly used by
core.concat	user.Person2Details	Main mapping user.LookupPerson
core.equal	Main mapping user.EqualAnd	user.LookupPerson
core.filter	Main mapping user.LookupPerson	
core.logical-and	user.EqualAnd	Main mapping user.LookupPerson

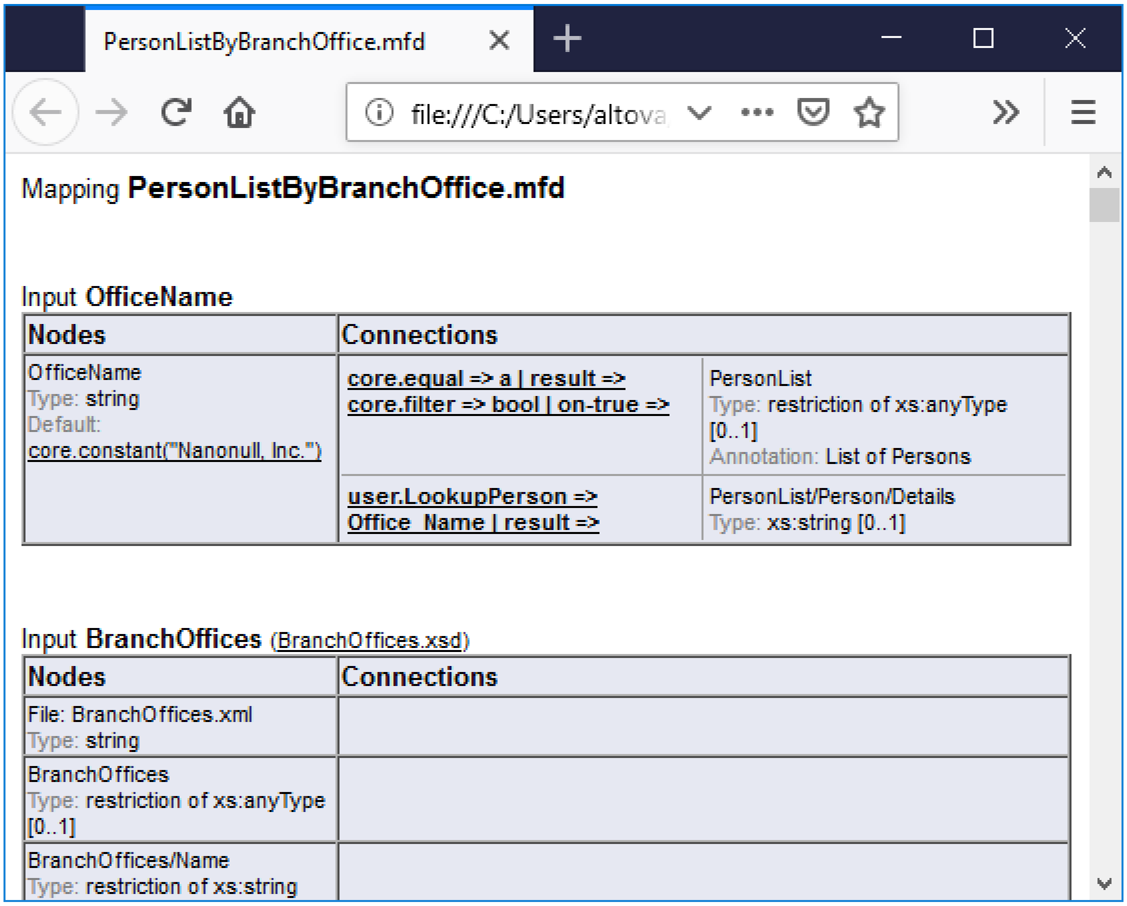
Stylesheet "ImpactAnalysis.sps"



This report lists every input and output node connection independently and is perfect for further impact analysis with modelling tools.

Input Node	Functions	Output Node
OfficeName	core.equal, core.filter	PersonList
OfficeName	user.LookupPerson	PersonList/Person/Details
BranchOffices/Office	core.filter	PersonList
BranchOffices/Office/Name	core.equal, core.filter	PersonList
BranchOffices/Office/Contact		PersonList/Person
.../Office/Contact/first		PersonList/Person/First
.../Office/Contact/first	user.LookupPerson	PersonList/Person/Details
.../Office/Contact/last		PersonList/Person/Last
.../Office/Contact/last	user.LookupPerson	PersonList/Person/Details

Stylesheet "OverallDocumentation.sps"



8.8.2 Custom Stylesheets

In addition to the built-in fixed design, you can create custom stylesheets for the generated mapping documentation with StyleVision (https://www.altova.com/stylevision). You can also change any of the predefined stylesheets 1342, for example, by adjusting the fonts and other styles.

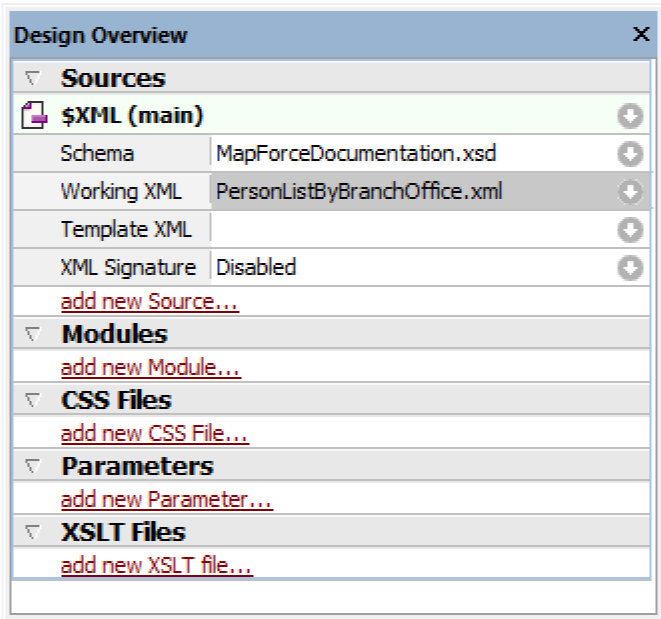
A custom design is a StyleVision Power StyleSheet (SPS). The advantage of using an SPS for generating mapping documentation is that you have complete control over the design of the documentation.

To create a custom SPS file, the following is required:

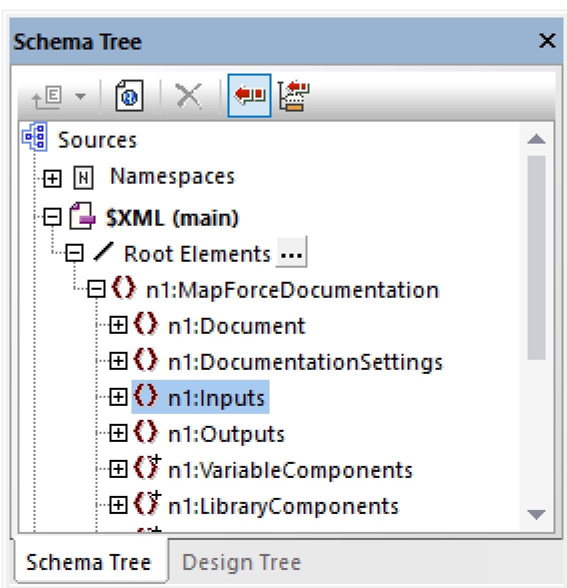
1. The XML Schema that provides the structure of the generated MapForce documentation. This schema is called **MapForceDocumentation.xsd** and is delivered with your MapForce installation package. It is stored in the **...\Documents\Altova\MapForce2023\Documentation\MapForce** folder. Note that the **MapForceDocumentation.xsd** includes the **Documentation.xsd** file located in the folder above it.

2. Some sample data to test and preview the custom design. You can use the following XML file as sample data : **...\Documents\Altova\MapForce2023\Documentation\MapForce\SampleData\PersonListByBranchOffice.xml**.

The files mentioned above must be referenced in the Design Overview window in StyleVision, for example:



In StyleVision, you create a design by dragging nodes from the Schema Tree window onto the design area and assigning styles and properties to them.



You can also add additional components such as links and images to the SPS design. To preview the design in a specific format, click any of the following tabs: **HTML**, **RTF**, **PDF**, or **Word 2007+**. For more details, refer to StyleVision documentation (https://www.altova.com/documentation).

9 Debugger

MapForce includes a mapping debugger available for the MapForce BUILT-IN transformation language. The mapping debugger helps you achieve the following goals:

- View and analyze the values produced by the mapping at each individual connector level.
- Highlight on the mapping the context (set of nodes) responsible for producing a particular value.
- Execute a mapping step-by-step, in order to see how MapForce processes or computes each value in real time, and preview the mapping output as it is being generated.
- Set milestones (breakpoints) at which the mapping execution should stop and display the value(s) currently being processed.
- View the history of values processed by a connector since mapping execution began up until the current execution position.

The mapping debugger is available when the transformation language of the mapping is BUILT-IN. If you start debugging a mapping designed for a different language, you will be prompted to change the mapping language to BUILT-IN. You can also convert a mapping to BUILT-IN by selecting the menu command **Output | Built-in Execution Engine**. In either case, the conversion to BUILT-IN will be successful if the mapping does not include components that are not available in the BUILT-IN language (for example, XSLT functions).

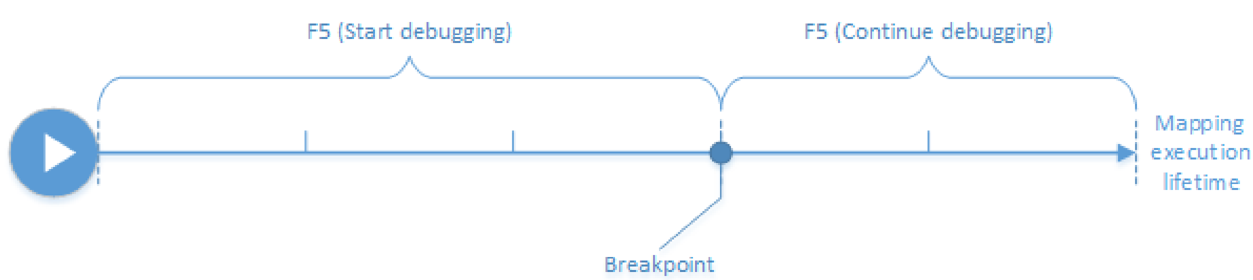
The MapForce debugger is unlike a traditional debugger in that it does not traverse your program code line by line (since you do not write any code with MapForce). Instead, the debugger exposes the results of MapForce-generated code produced from the mappings you design. More specifically, the debugger logs values that are passed from and to mapping components through their input and output connectors. The logged values are then available for your analysis directly on the mapping or through dedicated windows.

The following sections highlight various ways in which you can use the mapping debugger.

Debugger settings are available in the Options[1475] dialog box. The list of available debugging commands is available in Debug[1466].

Debug with breakpoints

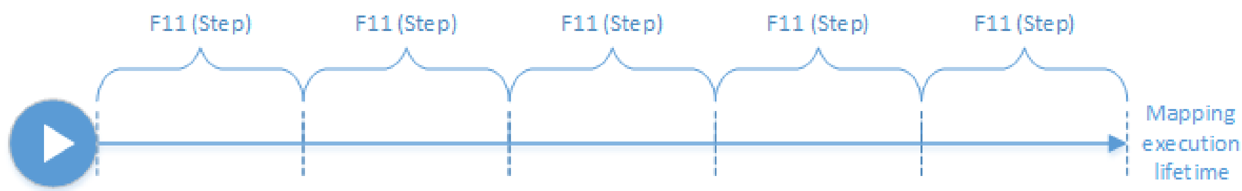
When you need to stop the debugging execution at a particular place in the mapping, you can set breakpoints, similar to how you would do that in a traditional development environment. The difference is that breakpoints are added not to a line of code, but to an input or output connector of a mapping component. You can also add conditions to breakpoints (this can be useful if you want to stop the execution only if the set condition is satisfied).



You can define breakpoints on the desired connectors and execute the mapping up to the first encountered breakpoint, then go to the next one, and so on. This way you can analyze the mapping context and values associated with chosen connectors. You can also speed up or slow down the execution by means of the **Step Into**, **Step Out**, **Step Over**, and **Minimal Step** commands provided by the debugger. These commands enable you to skip portions of the mapping, or, on the contrary, execute portions of the mapping in a more granular way if necessary.

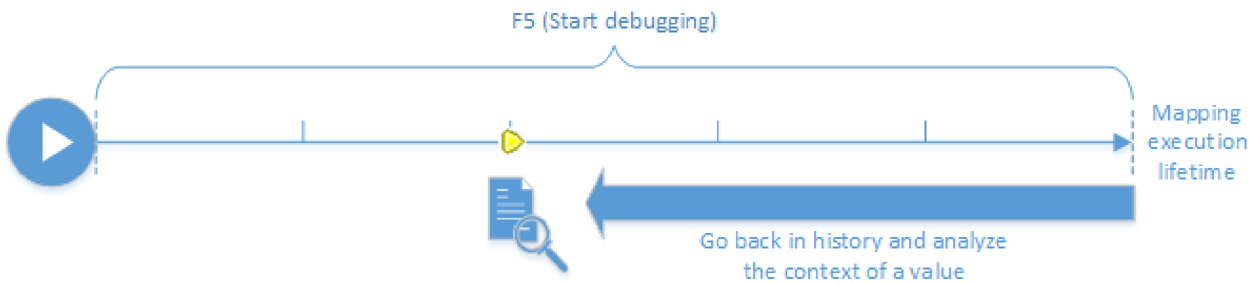
Debug step-by-step

You can debug a mapping step-by-step, and analyze the mapping context and values associated with each step. This scenario is similar to the previous one, in that you can speed up or slow down execution using the **Step Into**, **Step Out**, **Step Over**, and **Minimal Step** commands.



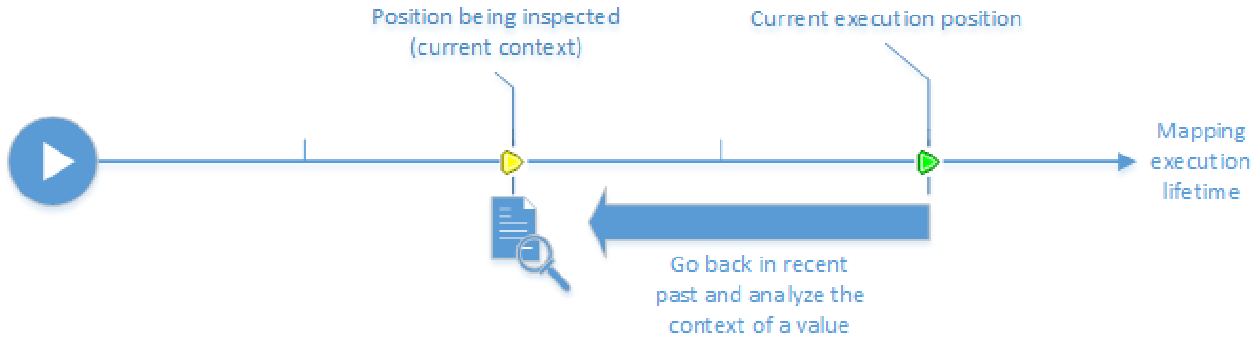
Analyze the log of values

You can configure MapForce to remember the log of all values (trace history) that were processed by all connectors while you debug a mapping. Keeping the full trace history may not be suitable for mappings that are data-intensive, so this option can be disabled if necessary. When the option is enabled, you can analyze the full log of values processed by each connector up until the current execution position. You can also instruct MapForce to recreate the mapping context associated with any particular value, which would help you understand why that value was produced.



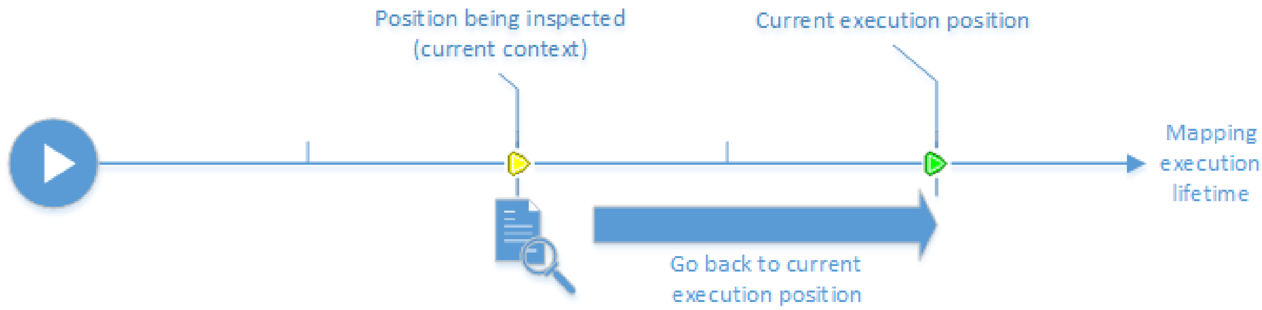
Set the context to a value related to the current execution position

When the debugger is at a particular execution position on the mapping, it is possible to analyze the context of a past value relative to the current execution position (this can be compared to stepping slightly back in time):



A context is meant to explain why a value is computed; in other words, it describes how a particular value on the mapping came to be generated. The context is normally the current execution position, although it can also be a context in the recent past that MapForce enables you to set. When the context is set to a particular value, MapForce highlights directly on the mapping the nodes that are relevant to it, provides tips next to mapping connectors, and exposes additional information in debugger-related windows (the **Values**, **Context**, and **Breakpoints** windows).

After you have inspected a mapping context that is not the same as the current execution position, you can reset the context back to the current execution position:



Limitations

- When MapForce executes a mapping, it may internally optimize code (for example, by caching data, or by calculating intermediate results at arbitrary points). This may cause certain connectors (and thus breakpoints) to be unreachable for debugging, in which case MapForce displays a notification. Note that the MapForce code optimizations (and, consequently, the behavior exposed by the debugger) may be different from one MapForce release to the other, even though the mapping output is the same for a given mapping.
- The debugger can debug the output generation for one target component at a time. If there are multiple target components on the mapping, you will need to select which one should be executed by the debugger.
- Currently, debugging is not supported for the database table actions (such as "Insert All", "Update If", etc.) of database components.
- Breakpoints cannot be added on any of the following entities: constants, the **`core | position`** function, descendent items of "Copy-all" connections, parameters of "inline" user-defined functions.

9.1 Debugger Preparation

Debugging preparation is primarily required for big data mappings that are likely to need a lot of system memory to execute. This is the case of mappings that either process very big input or output files, or repeatedly iterate through large collections of data.

To make debugging faster and reduce memory requirements, it is recommended to do the following before you start debugging:

- If the mapping is complex, remove or disconnect parts of the mapping that need not be debugged.
- If the mapping uses big input files, replace them with files of smaller size.
- Ensure that the **Keep full trace history** option is disabled (see Debugger Settings1475)

Also, to ensure you are debugging the right output, check the following if applicable:

- If the mapping has multiple target components, select the target component to be debugged by clicking the **Preview** button ( ).
- If the mapping is a chained mapping (see Chained Mappings1246), release the **Pass-Through** ( ) button on the intermediary component. Debugging Pass-Through components is currently not supported.

Optionally, if you want the debugger to stop at some important connectors whose value you want to analyze, add breakpoints to these connectors (see Adding and Removing Breakpoints1356).

9.2 About the Debug Mode

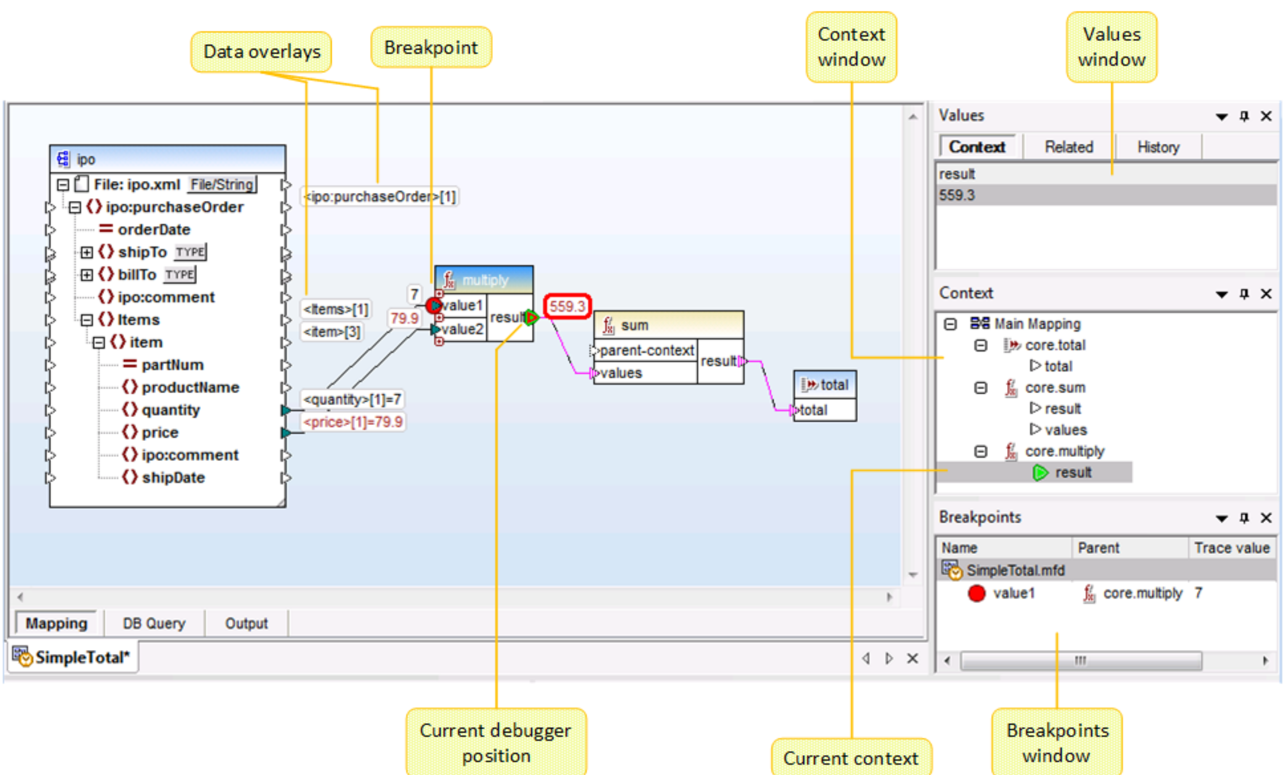
When you start debugging (by pressing **F5**, or **F11**, or **Ctrl + F11**), MapForce executes the mapping in debug mode.

> While MapForce is in debug mode, the mapping is read-only. Although you can move components on the mapping area, most commands are not available. This includes commands such as mapping validation and deployment, code generation, documenting mappings, adding new components to the mapping area or reloading existing ones, and others.

The debug mode enables you to analyze the context responsible for producing a particular value. This information is available directly on the mapping, as well as in the Values, Context, and Breakpoints windows. By default, these windows are displayed when you start debugging and are hidden when you stop debugging.

MapForce is in debug mode (and the mapping is read-only) until you stop debugging, by pressing **Shift + F5** (or by clicking the **Stop debugging** toolbar button).

The following image illustrates a sample mapping (**SimpleTotal.mfd**, from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory) that is debugged in steps (by pressing **F11** to advance a step).



The MapForce development environment in debug mode

The visual clues and other information provided by MapForce while in debug mode are described below.

The mapping pane

While debugging, the mapping pane displays additional information:

- Data overlays (see below) show the current value and related values near their connectors.
- The current context (shown as a structure in the Context window) is highlighted as follows:
  - Connectors in the context are striped magenta ( ).
  - Connectors in ambiguous context are dotted magenta ( ).
  - Connections in the context are striped magenta.
  - Connections in ambiguous context are striped magenta but lighter.
- The current execution location is displayed with a green connector icon ( ).

Data overlays

The values processed by each connector are displayed as data overlays (small rectangles) near their corresponding connector. A currently selected data overlay is displayed with thick red border. Values changed from the last step are displayed in dark red. For nodes with simple content, the data overlay combines two values - the node name and the value. If the node name has been iterated multiple times before the current execution position, the index of the current iteration is indicated by the number in square brackets.

Data overlays have the following behavior:

- Pointing the mouse to a data overlay brings it temporarily to the foreground, clicking it does it permanently. Clicking also selects the corresponding connector.
- Data overlays can be moved by dragging.
- Data overlays move when a component is moved. Therefore, if the data overlays appear stacked because the components are too close to each other, drag the components around the mapping area to make more space, and the data overlays will move together with the component.
- Clicking a data overlay shows its value in the Values window.
- Clicking a connector also selects its data overlay.

Breakpoints

Breakpoints are designated milestones at which the mapping should break during execution in debug mode. This term may be already familiar to you by analogy with other integrated development environments. Unlike other development environments where you add breakpoints to a line of code, a breakpoint in MapForce can be added to an input or output connector (small triangle to the left or right of the connection). On the mapping pane, breakpoints are represented as red circles. Any defined breakpoints are also displayed in the Breakpoints window. See also Adding and Removing Breakpoints 1356.

Current debugger position

The green triangle ( ) indicates the position of the debugger. This position is either an input or an output connector of any given component.

The value currently being processed is also displayed in the Values window, on the **Context** tab.

The set of connections and/or connectors colored in striped magenta indicate the current mapping context. The same information is also displayed as a hierarchical structure in the Context window (see Using the Context Window 1360).

When you set manually the context of a value, the current debugger position is in a position in the past relative to the most current execution position. To help you distinguish between the most current execution position and the one in the past, the "current position" connector may appear with the following colors in the debugger interface.

	Green is "the present"; it indicates the current execution position (see Viewing the Current Value of a Connector[1365]).
	Yellow is "the past"; it indicates that you are looking at some connector in the past, relative to the current execution position. This may happen after you set a context manually (see Setting the Context to a Value[1368]).

Values window

The Values window provides information about the values processed by the mapping. It enables you to see what the mapping processes at the current execution position, or in a particular context that you can set yourself. See also Using the Values Window[1358].

Context window

The Context window provides a hierarchical view of the set of nodes and functions that are relevant for the current debugger position. See also Using the Context Window[1360].

Breakpoints window

The Breakpoints window displays the list of debugging breakpoints created since MapForce was started. If you have defined breakpoints on multiple mappings, all of them appear in the Breakpoints window. See also Using the Breakpoints Window[1362].

9.3 Adding and Removing Breakpoints

Breakpoints are designated milestones at which the mapping should break during execution in debug mode. Any breakpoints you create are stored globally for all mappings and are displayed in the Breakpoints window. Breakpoints are valid until you either explicitly delete them, or close MapForce.

Note: Breakpoints cannot be added on any of the following entities: constants, the `core | position` function, descendent items of "Copy-all" connections, parameters of "inline" user-defined functions.

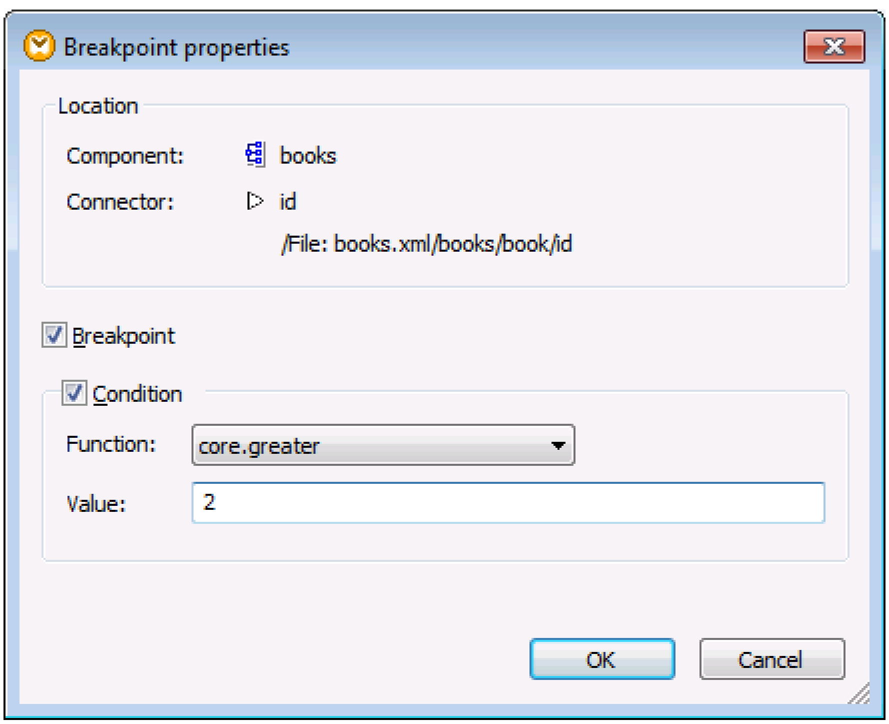
Breakpoints can be simple or conditional. Simple breakpoints stop the mapping execution unconditionally. Conditional breakpoints stop the mapping execution only when the condition assigned to them is satisfied. Conditions take the form of MapForce built-in library functions to which you supply custom values. In other words, if the condition returns true, the breakpoint will stop the mapping execution.

To create a simple breakpoint, do one of the following:

- Right-click an input or output connector (the small triangles to the left or right of a component), and select **Debugger Breakpoint**.
- Click an input or output connector, and then press **F9**.

To create a conditional breakpoint:

1. Right-click a connector, and select **Breakpoint properties**.



2. Click to select both the **Breakpoint** and **Condition** check boxes.

3. Select the required function from the list, and enter the function value (if applicable). For example, in the example above, the breakpoint will stop the mapping execution if the value passing through it is greater than 2.

> If the data type of the connector where you add the conditional breakpoint does not match the type(s) expected by the function, MapForce will attempt to convert the data type automatically. If automatic conversion is not possible, mapping execution will fail. To avoid this, make sure to use compatible data types. For example, the function `core.starts-with` expects a string value, so the breakpoint's connector must have the same type.

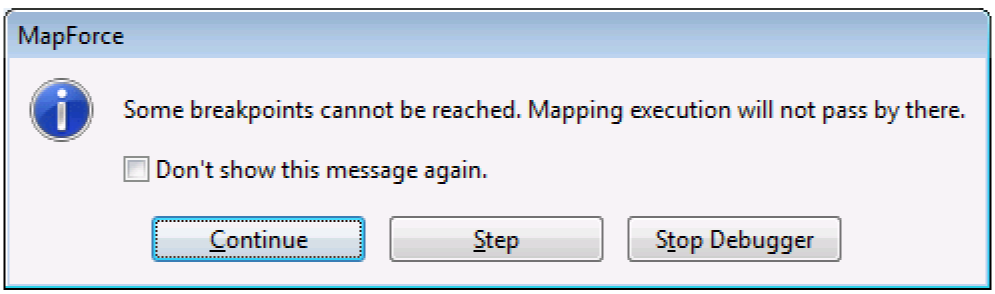
Removing breakpoints

To remove a breakpoint, right-click the connector on which the breakpoint exists, and select **Debugger Breakpoint**. Alternatively, click the input or output connector on which the breakpoint exists, and then press **F9**.

You can also remove breakpoints from the Breakpoints window (see Using the Breakpoints Window [1362]).

Unreachable breakpoints

There may be cases when MapForce displays a "Breakpoints cannot be reached" message:



This indicates that breakpoints cannot be reached by the debugger, because of one of the following reasons:

- A breakpoint has been defined on a connector that does not take part in the mapping.
- The breakpoint cannot be reached by MapForce because of execution optimizations (see Limitations [1351]).

Click **Continue** to advance to the next defined breakpoint (or go to the end of debugging execution). Click **Step** to start debugging in steps.

You can disable notifications about unreachable breakpoint encountered by the debugger, either by clicking **Don't show this message again**, or as follows:

1. On the **Tools** menu, click **Options**.
2. Click **Messages**.
3. Click to clear the **Inform about unreachable breakpoints** check box.

9.4 Using the Values Window

The Values window displays information about the values processed by the mapping when in debug mode. The information displayed in the Values window depends on the current debugger position, and on the user interface elements that you clicked. The Values window contains the following tabs:

The "Context" tab

The **Context** tab displays the value currently being processed (the same value whose context is shown in the Context window). This is either the value at the current execution position of the debugger, or the value of a connector processed in the past. MapForce helps you distinguish between the two using colors:

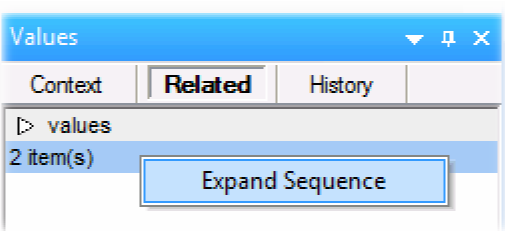
	Green is "the present"; it indicates the current execution position (see Viewing the Current Value of a Connector [1365]).
	Yellow is "the past"; it indicates that you are looking at some connector in the past, relative to the current execution position. This may happen after you set a context manually (see Setting the Context to a Value [1368]).

The "Related" tab

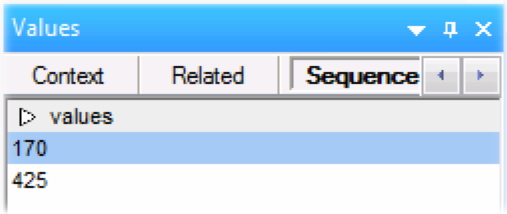
The **Related** tab displays values that are related to (or represent the "near past" of) the currently processed value. Normally, you do need to explicitly click this tab; MapForce switches to it automatically when you click the data overlay of a connector that is related to the current execution position of the debugger. See Stepping back into Recent Past [1366].

The "Sequence" tab

When present, the **Sequence** tab enables you to get access to the values of a connector that processes a sequence. This tab is visible only when a connector has processed a sequence of items (for example, an aggregate function such as `sum` or `count` does that). When you click the data overlay of a connector that processed a sequence of items, the Values window displays an entry in the format "**n items**", where **n** is the number of items processed by the connector. To get access to each value, double-click this entry (or right-click it, and select **Expand Sequence** from the context menu).



The values are then displayed in the **Sequence** tab.



The "History" tab

The **History** tab displays values have been processed by a particular node since debugging started and up to the current execution position. See Viewing the History of Values Processed by a Connector[1367].

9.5 Using the Context Window

While MapForce is in debug mode, the Context window displays a structure of connectors that are relevant to the current position of the debugger. In other words, it provides the mapping context responsible for producing the current mapping value.

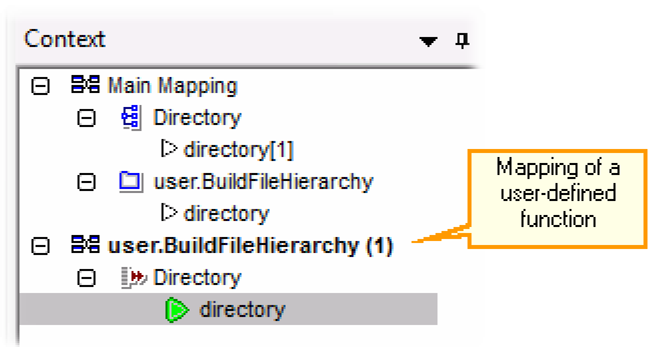
MapForce builds the current context as follows:

1. Start with the root node of the target structure.
2. Descend to the current target node.
3. From the current target node, move left inside the mapping through any components that lead to the current position. These components may be filter or sort components, built-in or user-defined functions, variables, and so on.

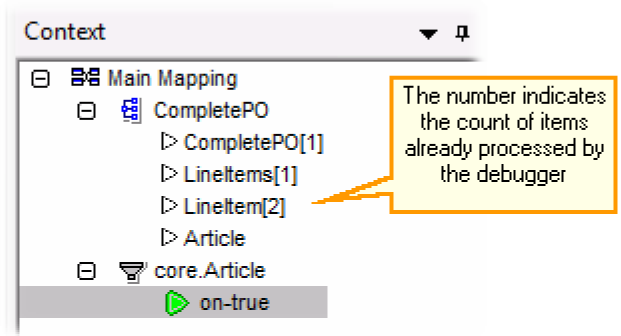
The Context window serves both as informational and as a navigational aid. To select a particular node in the mapping directly from the current context, right-click the node in the Context window, and click **Select in mapping**. This might be especially useful when the mapping is large, so as to avoid extensive scrolling.

The Context window may display the following special icons and notation:

<table>
<tr><th>Icon</th><th>Description</th></tr>
<tr><td></td><td>Represents the mapping to which the context belongs. This can be either the main mapping or the mapping of a user-defined function.



</td></tr>
<tr><td></td><td>Represents a connector. The target nodes processed so far have their position displayed in square brackets.</td></tr>
</table>

Icon	Description
	
	Represents the current connector (the most recent execution position). This is the source of the current value in the Values window. In some rare situations, it is possible that a computed value is used for multiple connectors. In this case, multiple green icons may appear.
	Represents the current connector when the debugger is at some position in the past relative to the most recent execution post. This may happen after you set the context to a value (see Setting the Context to a Value [1368]).

In addition to the icons above, the Context window includes the standard icons of any component types that are present in the mapping.

Context window and user-defined functions

If the current context includes any user-defined functions, they are displayed in the Context window as well. Note that if the current context is for computing an input value of a user-defined function, the context is determined as follows:

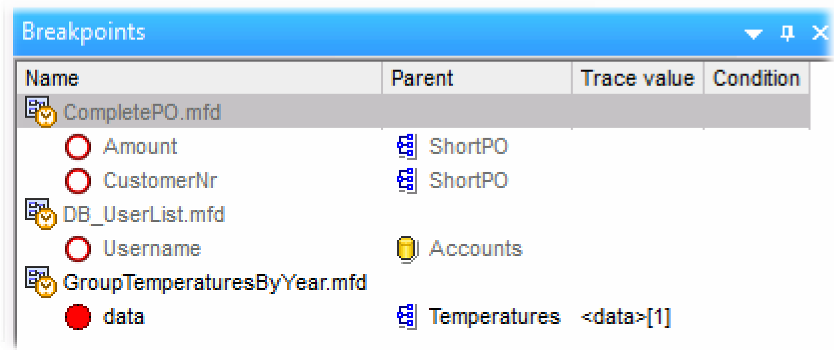
1. From the target to the output connector of the user-defined function to the input connector of the user-defined function
2. From there further to the left.

Note: A user-defined function may occur multiple times in the context. This happens either because several function calls are chained or because the user-defined function is defined as recursive.

9.6 Using the Breakpoints Window

The Breakpoints window enables you to view and manage breakpoints globally. By default, the Breakpoints window is displayed when MapForce is in debug mode. If you want to make the Breakpoints window visible at all times, select the menu command **View | Debug Windows | Breakpoints** when MapForce is not in debug mode.

The Breakpoints window displays all breakpoints created since you started MapForce, grouped by the mapping file to which they belong. While MapForce is open, any breakpoints associated with any mapping are "remembered" by MapForce and displayed in the Breakpoints window, even if you closed the mapping file in the meanwhile. The mapping that is currently being debugged is represented with standard text color in the Breakpoints window, while other mappings (the ones that are closed or not active) are grayed out.



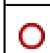
You can quickly open any mapping by double-clicking it (or any of its breakpoints) in the Breakpoints window.

Note: Once you close or restart MapForce, all breakpoints are removed.

Information about breakpoints is displayed as a grid with the following columns:

Column	Description
Name	The name of the node where the breakpoint belongs.
Parent	The name of the mapping component where the breakpoint belongs.
Trace value	The value that passes through the connector on which the breakpoint is. The trace value is displayed during debugging execution.
Condition	If the breakpoint is conditional, this column displays the condition of the breakpoint.

Breakpoints may be associated with any of the following icons.

Icon	Description
	Active breakpoint. Denotes a breakpoint from the mapping that is currently being debugged.
	Inactive breakpoint. Denotes a breakpoint from a mapping that is open, but is not currently being debugged.

Icon	Description
	Inaccessible breakpoint. Denotes a breakpoint that cannot be reached by the debugger.
	Conditional breakpoint. Denotes a breakpoint with a condition attached to it.

To view or change the properties of a breakpoint:

- Right-click it, and select **Breakpoint Properties** from the context menu.

To delete a breakpoint:

- Right-click the breakpoint you want to delete, and then select **Delete Breakpoint** from the context menu.
- Click a breakpoint, and then press **Delete**.

The context command **Delete All Breakpoints** removes all breakpoints displayed in the Breakpoints window, regardless of the mapping where they belong.

See also: Adding and Removing Breakpoints [1356]

9.7 Previewing Partially Generated Output

When you are debugging in steps or using breakpoints, you can view the mapping output generated up to the current debugger position. Previewing partially generated output is supported by XML, flat text, and EDI target components.

By default, when you press **F5** (without having defined any breakpoints), MapForce executes the entire mapping in debug mode, and then switches to the **Output** tab, displaying the final generated output. However, if you have defined breakpoints, or if you are debugging in steps (**F11**, or **Ctrl + F11**), the debugger execution stops while the mapping output is still being generated. Even if the mapping output is partially written at this stage, you can still click to the **Output** tab, and preview it.

```
Generating result C:\Users\altova\Documents\Altova\MapForce2016\MapForceExamples\CompletePO.xml...

<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xsi:noNamespaceSchemaLocation="
file:///C:/Users/altova/Documents/Altova/MapForce2016/MapForceExamples/CompletePO.xsd" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance">
  <Customer>
    <Number>3</Number>
    <FirstName>Ted</FirstName>
    <LastName>Little</LastName>
    <Address>
      <Street>Long Way</Street>
      <City>Los-Angeles</City>
      <ZIP>34424</ZIP>
      <State>CA</State>
    </Address>
  </Customer>
  <LineItems>
    <LineItem
```

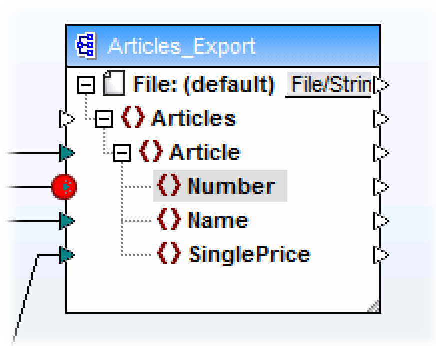
Limitations

- The currently computed target node is not always displayed in the Output tab. For example, XML attributes are collected internally and written at once.
- If the output produces multiple files, only the currently written file can be displayed; switching to another output file is disabled.

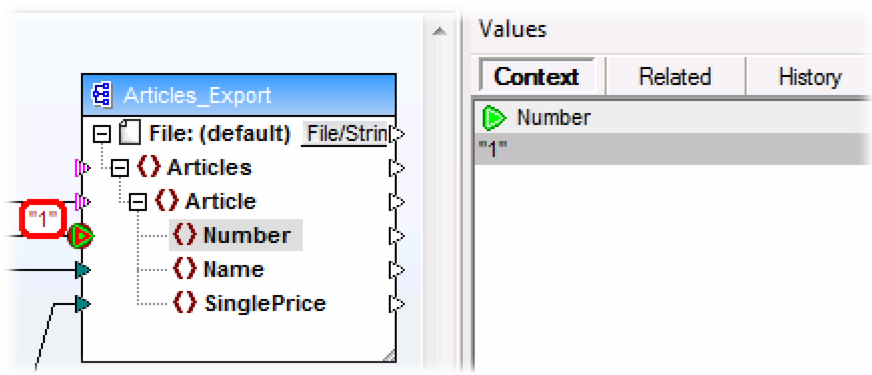
9.8 Viewing the Current Value of a Connector

When the current execution position of the debugger ( ) is on a particular connector (either because you are debugging in steps, or because there is a breakpoint defined on the connector), the current value processed by the connector is displayed in the **Context** tab of the Values window. This is the value that is about to be written to the output, that is, "the present". It is also the value whose context is displayed in the Context window (see Using the Context Window 1360).

To understand this case, open the **PreserveFormatting.mfd** sample from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory. Click the input connector of the `Number` node on the target component, and press **F9** to add a breakpoint on it.



Then press **F5** to start debugging and observe the results.



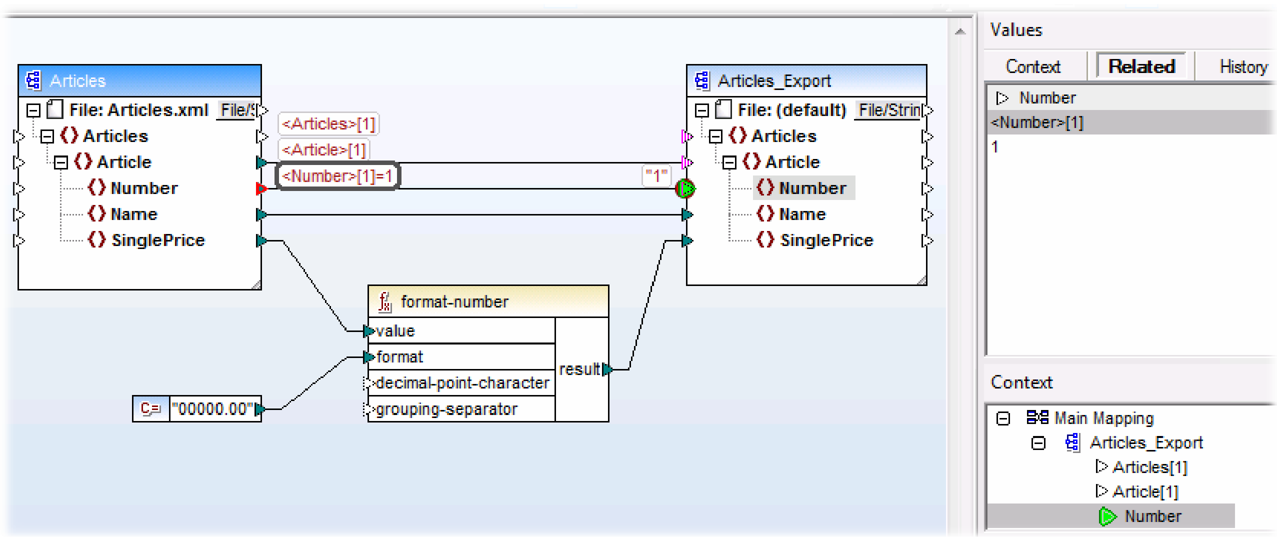
As shown in the image, the current debugger position (and the breakpoint) is on the `Number` node of the target component. The Values window indicates that this node processes the value "1" (this value is also highlighted with a thick red border on the mapping).

9.9 Stepping back into Recent Past

When you click a data overlay (small rectangular box) next to a mapping connector, the **Values** window displays the name and, optionally, the value associated with the selected connector. The focus now is no longer on the current debugger position, but on the selected data overlay. You can consider this view as stepping slightly back in the debugging history. This is the "near" past, since the mapping displays data overlays only for the last few connectors related to the current debugger position. When you click such a "related" data overlay, the Values window switches automatically to the **Related** tab.

For an illustration of this scenario, open the mapping **PreserveFormatting.mfd** from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.

After opening the mapping, click the connector next to the `Number` node on the target component, and press **F9** to add a breakpoint on it. Press **F5** to start debugging, and then click the data overlay (small rectangular box) next to the `Number` node of the source component.



Because a connector is typically iterated multiple times for the lifetime of a mapping, the current index of the iteration is displayed enclosed with square brackets: **<Number>[1]**. Also, because the connector carries a value, its value is also represented after the equal sign: **<Number>[1]=1**. The same value is displayed on a new row in the Values window, as shown below.

If you need additional information about a particular value, remember that you can recreate the context that produced it (see Setting the Context to a Value 1368).

9.10 Viewing the History of Values Processed by a Connector

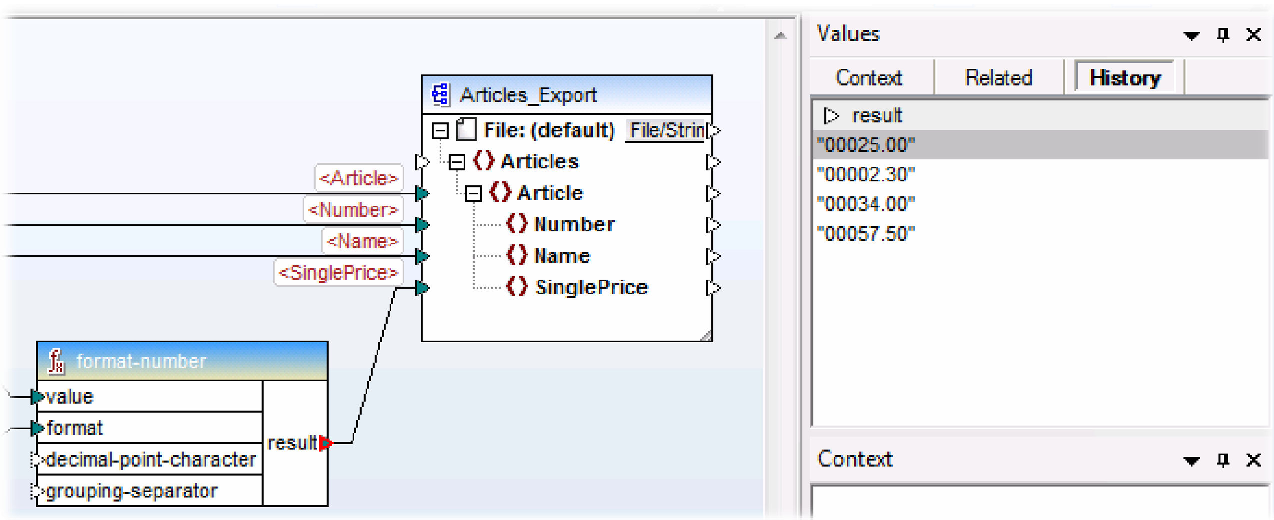
If the option **Keep full trace history** is enabled (see Debugger Settings[1475]), you can view the history of all values that were processed by that connector (up to the current execution position).

The history is displayed when you click a connector, and then click the **History** tab of the Values window. Note that this operation is meaningful only for connectors that have processed values since the beginning of mapping execution until the current debugger position.

To illustrate this case, let's debug a mapping from begging till end without using any breakpoints, and then watch the history of all values that were processed by a particular connector. First, open the mapping **PreserveFormatting.mfd** from the **<Documents>\Altova\MapForce2023\MapForceExamples** directory. If it is already open, make sure to do the following:

- Clear any breakpoints, if such exist (see Adding and Removing Breakpoints[1356])
- Stop debugging if it is currently in progress, by pressing **Shift + F5**.

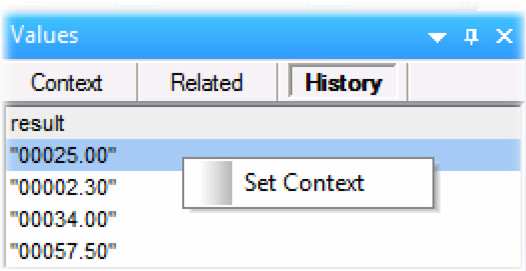
When ready, press **F5** start a new debugging operation. When you press **F5**, MapForce executes the mapping in debug mode, and switches to the **Output** tab. Click the **Mapping** tab to go back to the main mapping window, and then click the `result` node of the `format-number` function (highlighted in red in the image below). Finally, click the **History** tab of the Values window, and notice the displayed values.



As shown in the image above, this particular node (`result`) has processed four values in total. If you need additional information about a particular value, remember that you can recreate the context that produced it (see Setting the Context to a Value[1368]).

9.11 Setting the Context to a Value

Setting the context to a value is an action that can be compared to stepping into the past, in order to view more details about the mapping context that produced that value. You can set the context to any value displayed in the Values window (in the **Related** tab, **Sequence** tab, or **History** tab). If you have enabled the **Keep full trace history** option (see Debugger Settings [1475]), the **History** tab displays all values processed by the currently selected connector; therefore, in this case, you can additionally set the context to any value in the past for that connector.

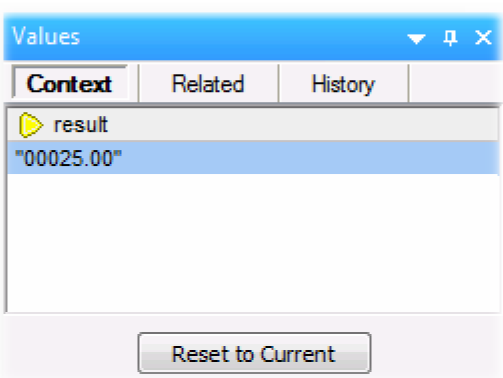


To set the context to a value, do one of the following:

- Right-click the value, and select **Set Context** from the context menu.
- Double-click the value.

When you set the context to a value, MapForce highlights the mapping area so as to recreate the situation that produced that value, and populates the **Values** window and the **Context** window according to the selected context. For a legend to visual clues used on the mapping area while in a context, see About the Debug Mode [1353]. For information about the context itself, see Using the Context Window [1360].

The connector of a manually-set context is yellow ( ), which indicates that you are no longer at the most recent execution position. To switch back to the most recent execution position (when applicable), click the **Reset to Current** button on the **Context** tab of the Values window.



10 Automation with Altova Products

Mappings designed with MapForce can be executed in a server environment (including Linux and macOS servers), and with server-level performance, by the following Altova transformation engines (licensed separately):

- *RaptorXML Server*. Running a mapping with this engine is suitable if the transformation language of the mapping is XSLT 1.0, XSLT 2.0, XSLT 3.0, or XQuery. See Automation with RaptorXML Server 1370.
- *MapForce Server (or MapForce Server Advanced Edition).* This engine is suitable for any mapping where the transformation language is BUILT-IN*. The BUILT-IN language supports the most mapping features in MapForce, while MapForce Server (and, in particular, MapForce Server Advanced Edition) provides best performance for running a mapping. See Automation with MapForce Server 1371.

** The BUILT-IN transformation language requires MapForce Professional or Enterprise Edition.*

In addition to this, MapForce provides the ability to automate generation of XSLT, XQuery, C#, C++, and Java code from the command line interface. This includes the ability to compile server execution files (.mfx) intended for MapForce Server execution. For more information, see MapForce Command Line Interface 1391.

10.1 Automation with RaptorXML Server

RaptorXML Server (hereafter also called RaptorXML for short) is Altova's third-generation, super-fast XML and XBRL processor. It is optimized for the latest standards and parallel computing environments. Designed to be highly cross-platform capable, the engine takes advantage of today's ubiquitous multi-core computers to deliver lightning-fast processing of XML and XBRL data.

RaptorXML is available in two editions which can be downloaded from the Altova download page (https://www.altova.com/download-trial-server.html):

- RaptorXML Server is a very fast XML processing engine with support for XML, XML Schema, XSLT, XPath, XQuery, and more.
- RaptorXML+XBRL Server supports all the features of RaptorXML Server with the additional capability of processing and validating the XBRL family of standards.

If you generate code in XSLT or in XQuery, MapForce creates a batch file called **DoTransform.bat** which is placed in the output folder that you choose upon generation. Executing the batch file calls RaptorXML Server and executes the XSLT (or XQuery) transformation on the server.

If you intend to execute or automate MapForce mappings for other outputs on a server, see Automation with MapForce Server1371.

Note: You can also preview the XSLT107 and XQuery code using the built-in engine.

10.2 Automation with MapForce Server

MapForce Server is an enterprise server software solution for Windows, Linux and macOS operating systems. The role of MapForce Server is to execute mappings in a server environment (including on non-Windows platforms) and with server-level performance. Any MapForce mapping where the target execution language is BUILT-IN qualifies for server execution (see also Selecting a Transformation Language 27). MapForce Server can operate either standalone (invoked from command line or API), or under the management of FlowForce Server.

If MapForce Server is used as a standalone product then the MapForce mapping has to be compiled and copied to the machine where MapForce Server runs. The mapping is then run using the MapForce Server command line command `run`. You can also run the mapping by invoking the `run` method of the MapForce Server API. For further information, see Compiling Mappings to MapForce Server Execution Files 1378.

If MapForce Server runs under FlowForce Server management, the mapping can be deployed to a target machine through an HTTP (or SSL/HTTPS) connection directly from MapForce. On the server, the mapping can then be executed as a triggered or scheduled job, or through a Web service call defined from the FlowForce Server administration interface. For further information, see Deploying Mappings to FlowForce Server 1381.

There are two editions of MapForce Server:

- MapForce Server
- MapForce Server Advanced Edition

MapForce Server Advanced Edition provides the same features as MapForce Server, and additionally includes optimization features for mappings which qualify for optimization. This is the case of mappings which join or filter large amounts of data, and where it is possible to apply join optimization so as to increase the execution speed. Unlike MapForce Server, MapForce Server Advanced Edition can execute mappings where node functions are present, see Defaults and Node Functions 912.

Limitations:

- XML digital signatures are not supported
- ADO, ADO.NET, and ODBC database connections are supported only on Windows (for other operating systems, see Preparing Mappings for Server Execution 1372).

For more information about MapForce Server, refer to its accompanying documentation (https://www.altova.com/documentation).

10.3 Preparing Mappings for Server Execution

A mapping designed and previewed with MapForce may refer to resources which are outside of the current machine and operating system (such as databases). In addition to this, in MapForce, all mapping paths follow Windows-style conventions by default. Thirdly, the machine where MapForce Server runs might not support the same database connections as the machine where the mapping was designed. For this reason, running mappings in a server environment typically requires some preparation, especially if the target machine is not the same as the source machine.

Note: The term "source machine" refers to the computer where the MapForce is installed and the term "target machine" refers to the computer where MapForce Server or FlowForce Server is installed. In the most simple scenario, this is the same computer. In a more advanced scenario, MapForce runs on a Windows machine whereas MapForce Server or FlowForce Server runs on a Linux or macOS machine.

As best practice, always make sure that the mapping validates successfully in MapForce before deploying it to FlowForce Server or compiling it to a MapForce Server execution file (see Validating Mappings 105).

If MapForce Server runs standalone (without FlowForce Server), the required licenses are as follows:

- On the source machine, MapForce Enterprise or Professional edition is required to design the mapping and compile it to a server execution file (.mfx), see Compiling Mappings to MapForce Server Execution Files 1378.
- On the target machine, MapForce Server or MapForce Server Advanced Edition is required to run the mapping.

If MapForce Server runs under FlowForce Server management, the following requirements apply:

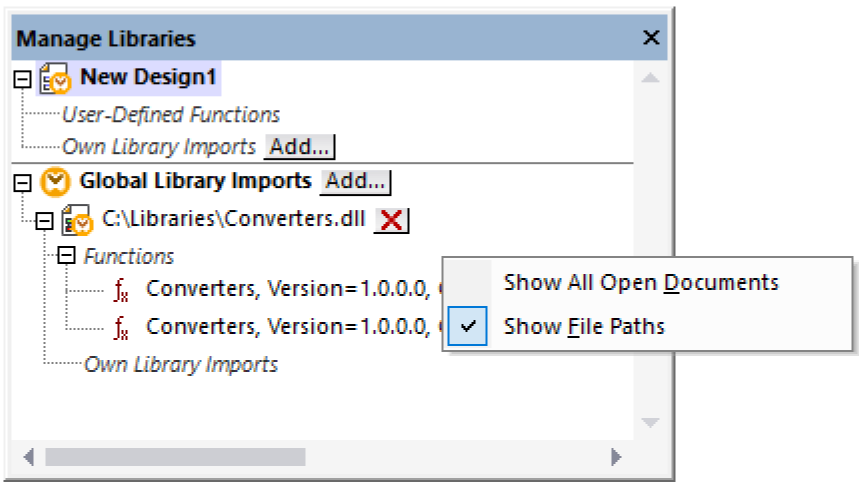
- On the source machine, MapForce Enterprise or Professional edition is required to design the mapping and deploy it to a target machine, see Deploying Mappings to FlowForce Server 1416.
- Both MapForce Server and FlowForce Server must be licensed on the target machine. The role of MapForce Server is to run the mapping; the role of FlowForce is to make the mapping available as a job which benefits from features such as scheduled or on demand execution, execution as a Web service, error handling, conditional processing, email notifications, and others.
- FlowForce Server must be up and running at the configured network address and port. Namely, the "FlowForce Web Server" service must be started and configured to accept connections from HTTP clients (or HTTPS if configured) and must not be blocked by the firewall. The "FlowForce Server" service must also be started and running at the designated address and port.
- You have a FlowForce Server user account with permissions to one of the containers (by default, the **/public** container is accessible to any authenticated user).

General considerations

- If you intend to run the mapping on a target machine with standalone MapForce Server, all input files referenced by the mapping must be copied to the target machine as well. If MapForce Server runs under FlowForce Server management, there is no need to copy files manually. In this case, the instance and schema files are included in the package deployed to the target machine, see Deploying Mappings to FlowForce Server 1416.
- If the mapping includes database components which require specific database drivers, such drivers must be installed on the target machine as well. For example, if your mapping reads data from a Microsoft Access database, then Microsoft Access or Microsoft Access Runtime

(https://www.microsoft.com/en-us/download/details.aspx?id=50040) must be installed on the target machine as well.

- When you deploy a mapping to non-Windows platforms, ADO, ADO.NET and ODBC database connections are automatically changed to JDBC. Native SQLite and native PostgreSQL connections are preserved as such and require no additional configuration. See also "Database connections" below.
- If the mapping contains custom function calls (for example, to .dll or .class files), such dependencies are not deployed together with the mapping, since they are not known before runtime. In this case, copy them manually to the target machine. The path of the .dll or .class file on the server must be the same as in the "Manage Libraries" window in MapForce, for example:



- Some mappings read multiple input files using a wildcard path (see Processing Multiple Input or Output Files Dynamically[1313]). In this case, the input file names are not known before runtime and so they are not deployed. For the mapping to execute successfully, the input files must exist on the target machine.
- If the mapping output path includes directories, those directories must exist on the target machine. Otherwise, an error will be generated when you execute the mapping. This behavior is unlike MapForce, where non-existing directories are generated automatically if the option **Generate output to temporary files** is enabled (see Changing the MapForce Options[1473]).
- If the mapping calls a Web service that requires HTTPS authentication with a client certificate, the certificate must be transferred to the target machine as well, see Digital Certificate Management[640].
- If the mapping connects to file-based databases such as Microsoft Access and SQLite, the database file must be manually transferred to the target machine or saved to a shared directory which is accessible to both the source and the target machine and referenced from there, see "File-based databases" below.

Making paths portable

If you intend to run the mapping on a server, ensure that the mapping follows the applicable path conventions and uses a supported database connection.

To make paths portable to non-Windows operating systems, use relative instead of absolute paths when designing the mapping in MapForce:

1. Open the desired mapping design file (.mfd) with MapForce on Windows.
2. On the **File** menu, select **Mapping Settings**, and clear the **Make paths absolute in generated code** check box if it is selected.
3. For each mapping component, open the **Properties** dialog box (by double-clicking the component's

title bar, for example), and change all file paths from absolute to relative. Also, select the **Save all file paths relative to MFD file** check box. For convenience, you can copy all input files and schemas into the same folder as the mapping itself, and reference them just by the file name.

For more information about dealing with relative and absolute paths while designing mappings, , see Using Relative and Absolute Paths 84.

Importantly, both MapForce Server and FlowForce Server support a so-called "working directory" against which all relative paths will be resolved, see also Paths in Various Execution Environments 87. The working directory is specified at mapping runtime, as follows:

- In FlowForce Server, by editing the "Working-directory" parameter of any job.
- In MapForce Server API, through the `WorkingDirectory` property of the COM and .NET API, or through the `setWorkingDirectory` method of the Java API.
- In MapForce Server command line, the working directory is the current directory of the command shell.

Database connections

Be aware that ADO, ADO.NET, and ODBC connections are not supported on Linux and macOS machines. Therefore, if the target machine is Linux or macOS, such connections are converted to JDBC when you deploy the mapping to FlowForce or when you compile the mapping to a MapForce Server execution file. In this case, you have the following options before deploying the mapping or compiling it to a server execution file:

- In MapForce, create a JDBC connection to the database (see Setting up a JDBC Connection 185)
- In MapForce, fill the JDBC database connection details in the "JDBC-specific Settings" section of the database component (see Database Component Settings 279).

If the mapping uses a native connection to a PostgreSQL or SQLite database, the native connection is preserved and no JDBC conversion takes place, see Database mappings in various execution environments 162. If the mapping connects to a file-based database, such as Microsoft Access and SQLite, additional configuration is required, see "File-based databases" below.

Running mappings with JDBC connections requires that the Java Runtime Environment or Java Development Kit be installed on the server machine. This may be either Oracle JDK or an open source build such as Oracle OpenJDK.

- The `JAVA_HOME` environment variable must point to the JDK installation directory.
- On Windows, a Java Virtual Machine path found in the Windows registry will take priority over the `JAVA_HOME` variable.
- The JDK platform (64-bit, 32-bit) must be the same as that of MapForce Server. Otherwise, you may get an error with the reason: "JVM is inaccessible".

To set up a JDBC connection on Linux or macOS:

1. Download the JDBC driver supplied by the database vendor and install it on the operating system. Make sure to select the 32-bit version if your operating system runs on 32-bit, and the 64-bit version if your operating system runs on 64-bit.
2. Set the environment variables to the location where the JDBC driver is installed. Typically, you will need to set the CLASSPATH variable, and possibly a few others. To find out which specific environment variables must be configured, check the documentation supplied with the JDBC driver.

Note: On macOS, the system expects any installed JDBC libraries to be in the **/Library/Java/Extensions** directory. Therefore, it is recommended that you unpack the JDBC driver to this location; otherwise, you will need to configure the system to look for the JDBC library at the path where you installed the JDBC driver.

Oracle Instant Client connections on macOS

These instructions are applicable if you connect to an Oracle database through the **Oracle Database Instant Client**, on macOS. Prerequisites:

- Java 8.0 or later must be installed. If the Mac machine runs a Java version prior to Java 8, you can also connect through the **JDBC Thin for All Platforms** library, and disregard the instructions below.
- Oracle Instant Client must be installed. You can download the Oracle Instant Client from the Oracle official download page. Note that there are several Instant Client packages available on the Oracle download page. Make sure to select a package with Oracle Call Interface (OCI) support, (for example, Instant Client Basic). Also, make sure to select the 32-bit version if your operating system runs on 32-bit, and the 64-bit version if your operating system runs on 64-bit.

Once you have downloaded and unpacked the Oracle Instant Client, edit the property list (.plist) file shipped with the installer so that the following environment variables point to the location of the corresponding driver paths, for example:

Variable	Sample Value
`CLASSPATH`	`/opt/oracle/instantclient_11_2/ojdbc6.jar:/opt/oracle/instantclient_11_2/ojdbc5.jar`
`TNS_ADMIN`	`/opt/oracle/NETWORK_ADMIN`
`ORACLE_HOME`	`/opt/oracle/instantclient_11_2`
`DYLD_LIBRARY_PATH`	`/opt/oracle/instantclient_11_2`
`PATH`	`$PATH:/opt/oracle/instantclient_11_2`

Note: Edit the sample values above to fit the paths where Oracle Instant Client files are installed on your operating system.

File-based databases

File-based databases such as Microsoft Access and SQLite are not included in the package deployed to FlowForce Server or in the compiled MapForce Server execution file. Therefore, if the source and target machine are not the same, take the following steps:

1. In MapForce, right-click the mapping and clear the check box **Make paths absolute in generated code** (see Changing the Mapping Settings 114).
2. Right-click the database component on the mapping and add a connection to the database file using a relative path, see Setting the Path to File-Based Databases 84. A simple way to avoid path-related issues is to save the mapping design (.mfd file) in the same directory as the database file and to refer to the latter from the mapping just by file name (thus using a relative path).
3. Copy the database file to a directory on the target machine (let's call it "working directory"). Keep this directory in mind since it will be required to run the mapping on the server, as shown below.

To run such mappings on the server, do one of the following:

- If the mapping will be run by MapForce Server under FlowForce Server control, configure the FlowForce Server job to point to the working directory created previously. The database file must reside in the working directory. For an example, see "Exposing a Job as a Web Service" (https://www.altova.com/manual/FlowForceServer/FlowForceServerAdvanced/index.html?fs_example_web_service.htm).
- If the mapping will be run by standalone MapForce Server at the command line, change the current directory to the working directory (for example, `cd path\to\working\directory`) before calling the `run` command of MapForce Server.
- If the mapping will be run by the MapForce Server API, set the working directory programmatically before running the mapping. To facilitate this, the property `WorkingDirectory` is available for the MapForce Server object in the COM and .NET API. In the Java API, the method `setWorkingDirectory` is available.

If both the source and the target machines are Windows machines running on the local network, an alternative approach is to configure the mapping to read the database file from a common shared directory, as follows:

1. Store the database file in a common shared directory which is accessible by both the source and the target machine.
2. Right-click the database component on the mapping and add a connection to the database file using an absolute path (see Setting the Path to File-Based Databases 84).

Global Resources

If a mapping includes references to Global Resources instead of direct paths or database connections, you will be able to use Global Resources on the server side as well. When you compile a mapping to a MapForce Server execution file (.mfx), the references to Global Resources will be kept intact, so that you can provide these on the server side, at mapping runtime. When deploying a mapping to FlowForce Server, you can optionally choose whether it should use resources on the server.

For mappings (or mapping functions, in case of FlowForce Server) to run successfully, the actual file, folder, or database connection details that you supply as Global Resources must be compatible with the server environment. For example, files and folders paths must use the Linux convention for paths if the mapping will run on a Linux server. Likewise, Global Resources defined as database connections must be possible on the server machine.

For further information, see Global Resources in MapForce Server 1415 and Global Resources in FlowForce Server 1415..

XBRL Taxonomy Packages

When you deploy a mapping that references XBRL Taxonomy Packages 1478 to FlowForce Server, MapForce collects all external references from the mapping and then resolves them using the current configuration and currently installed taxonomy packages. If there are resolved external references that point to a taxonomy package, then the taxonomy package is deployed together with the mapping. FlowForce Server will use that package—as it was during deployment—to execute the mapping. To refresh the taxonomy package used by FlowForce Server, you will need to change it in MapForce and redeploy the mapping.

Note that the root catalog of MapForce Server influences the way taxonomies are resolved on the target machine. The root catalog is found at the following path relative to the MapForce Server installation directory: **etc/RootCatalog.xml**.

Taxonomy packages that were deployed with a mapping will be used if the root catalog of MapForce Server does not already contain such a package or does not contain a package that is defined for the same URL prefix. The root catalog of MapForce Server has priority over the deployed taxonomy.

If MapForce Server runs standalone (without FlowForce Server), it is possible to specify the root catalog that should be used by the mapping as follows:

- At the command line, this is possible by adding the option `-catalog` to the `run` command.
- In the MapForce Server API, call the method `SetOption`, and supply the string `"catalog"` as first argument, and the path to the root catalog as second argument.

If a mapping uses XBRL components with table linkbases, the taxonomy package or the taxonomy package configuration file must be supplied to the mapping at runtime, as follows:

- At the MapForce Server command line, add the option `--taxonomy-package` or `--taxonomy-packages-config-file` to the `run` command.
- In the MapForce Server API, call the method `SetOption`. The first argument must be either `"taxonomy-package"` or `"taxonomy-packages-config-file"`. The second argument must be the actual path to the taxonomy package (or taxonomy package configuration) file.

10.4 Compiling Mappings to MapForce Server Execution Files

When the target language of a mapping created in MapForce is set to BUILT-IN, it can be executed not only by MapForce, but also by MapForce Server (see About MapForce Server1371). There are two ways to execute a mapping with MapForce Server:

- If MapForce Server runs in standalone mode (that is, no FlowForce Server is installed), the mapping must be compiled to a server execution file (.mfx), as shown below. You can then run the .mfx file at the command line, using the command `run`. You can also run the mapping by invoking the `run` method of the MapForce Server API. For further information, see the MapForce Server documentation (https://www.altova.com/documentation).
- If MapForce Server runs under FlowForce Server management, you can deploy the mapping to a machine where both MapForce Server and FlowForce Server run. For more information about this scenario, see Deploying Mappings to FlowForce Server1381.

Prerequisites

See Preparing Mappings for Server Execution1372.

To compile a mapping to a MapForce Server Execution (.mfx) file:

1. Open a mapping in MapForce (for example, **myMapping.mfd**).
2. On the **File** menu, click **Compile to MapForce Server Execution File**.
3. Select the folder you want to place the .mfx file in and change the file name if necessary.
4. Click **Save**. The MapForce Server Execution file **myMapping.mfx** is generated in the selected folder.

To compile a mapping to a MapForce Server Execution (.mfx) file, using the command line:

- Run MapForce at the command line, and specify the mapping file and the `/COMPILE` command line option.

For example, the following command compiles the mapping **C:\Users\altova\Documents\Altova\MapForce2023\MapForceExamples\SimpleTotal.mfd** to a MapForce Server execution file that will be created in the target output directory **C:\Users\altova\Desktop**.

```
"C:\Program Files (x86)\Altova\MapForce2023\MapForce.exe" "C:
\Users\altova\Documents\Altova\MapForce2023\MapForceExamples\SimpleTotal.mfd" /COMPILE
"C:\Users\altova\Desktop"
```

See also the MapForce Command Line Interface1391.

What's included in the .mfx file

The .mfx file includes the following data:

- The mapping algorithm, which includes all user-defined functions (UDFs) imported from other mappings.

- Input and output file names referenced from components. Paths are absolute or relative depending on the mapping settings, see Paths in Various Execution Environments 87.
- If the mapping contains XML components, information about the XML schema needed to execute the mapping is encoded into the mapping algorithm.
- The database connection details, if the mapping includes database connections. Passwords are encrypted.
- Information[1] derived from the following, as applicable to each mapping:
  - EDI and FlexText configurations
  - JSON schemas
  - .proto files
- If the mapping includes credentials, the sensitive fields are included, in encrypted form, if you selected the **Include in MapForce Server Execution File and Mapping Deployment** check box. Otherwise, the sensitive fields are skipped. For details, see Credentials 1297.

Footnotes:

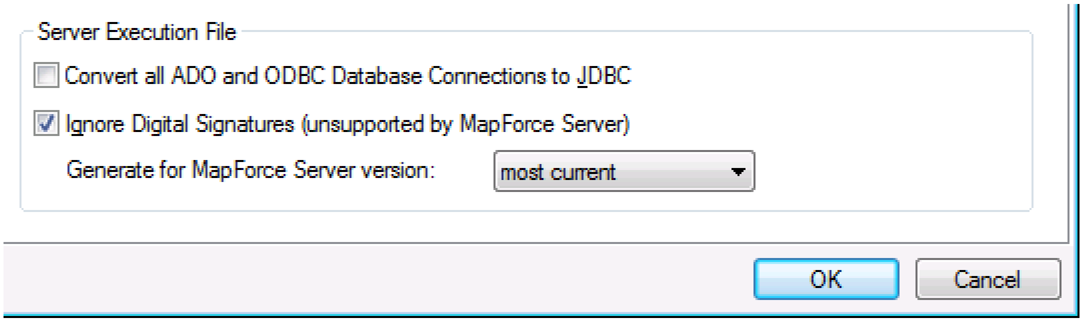
1. Information derived from EDI or FlexText configurations, JSON schemas, or .proto files is embedded in the .mfx file in a compiled and processed form. It is not possible to obtain the original source files from a compiled .mfx file. Data is compiled in the state it exists at the moment of compilation; if you change it subsequently, the .mfx file must be re-compiled.

The input instance files (XML, CSV, Text, EDI, Excel, JSON, XBRL) that are used by the mapping are not included in the compiled .mfx file. The same is true for file-based databases such as Access or SQLite. If the mapping references XBRL taxonomies related to XBRL table support, the taxonomy information is not compiled into the .mfx file. For details, see Preparing Mappings for Server Execution 1372.

Compiling mappings for a specific MapForce Server version

If your MapForce Server has an older version than MapForce, the former might not be able to execute .mfx files created with a newer version of MapForce, since new features will likely have been added in the meanwhile. In such cases, you can compile the .mfx file for a specific version of MapForce Server, as follows:

1. On the **Tools** menu, click **Options**, and then click **Generation**.
2. Under **Server Execution File**, next to **Generate for MapForce Server version**, select the required MapForce Server version from the drop-down list.



Once you have a newer MapForce Server version, remember to change this option accordingly. If you have no particular reason to compile for a specific version of MapForce Server, select the "most current" option (this is the default option). When this option is selected, the .mfx file is compiled for the most recent version of MapForce Server and could benefit from latest features and improvements which might otherwise not be available in previous versions.

To specify a target MapForce Server version at the command line, run the `/COMPILE` command with the `/MFXVERSION` switch, for example:

```
"C:\Program Files (x86)\Altova\MapForce2023\MapForce.exe" /COMPILE /MFXVERSION:2023
```

See also the MapForce Command Line Interface 1391.

Other options

Compilation of MapForce Server Execution Files is also affected by the following options:

Convert all ADO and ODBC Database Connections to JDBC	If the option is enabled, ADO, ADO.NET, and ODBC database connections are transformed to JDBC using the JDBC driver and the database URL defined in the Database Component Settings dialog box (see Database Component Settings 279). The JDBC connection will be used implicitly if the target machine is a Linux or macOS server.
Ignore Digital Signatures (unsupported by MapForce Server)	This option is applicable only to MapForce Enterprise. It is enabled by default. If the mapping uses XML digital signatures, it skips the digital signature information, since MapForce Server does not support XML digital signatures.

To view or change these options:

- On the **Tools** menu, click **Options**, and then click **Generation**.

These options are also available from the command line interface. See also the MapForce Command Line Interface 1391.

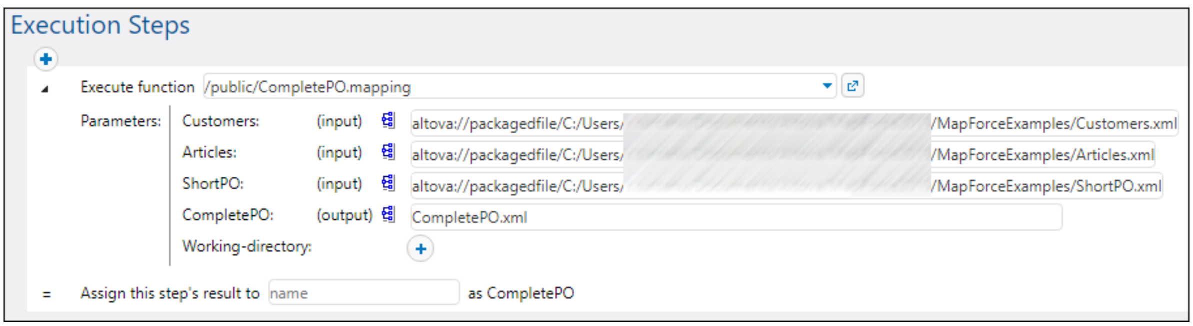
10.5 Deploying Mappings to FlowForce Server

Deploying a mapping to FlowForce Server means that MapForce organizes the resources used by the specific mapping into an object and passes it through HTTP (or HTTPS if configured) to the machine where FlowForce Server runs. MapForce mappings are typically deployed to FlowForce Server in order to automate their execution by means of FlowForce Server jobs. Once a mapping is deployed, you can create a full-featured FlowForce Server job from it, and benefit from all job-specific functionality (for example, define custom triggering conditions for the job, expose it as a Web service, and so on).

Note: The term "source machine" refers to the computer where the MapForce is installed and the term "target machine" refers to the computer where FlowForce Server is installed. In the most simple scenario, this is the same computer. In a more advanced scenario, MapForce runs on a Windows machine whereas FlowForce Server runs on a Linux or macOS machine.

The package deployed to FlowForce includes the following:

- The mapping itself. After deployment, the mapping becomes available in the FlowForce Server administration interface as a mapping function (.mapping), at the path you specify. Any source components become input arguments, and any target components become output arguments of this function.



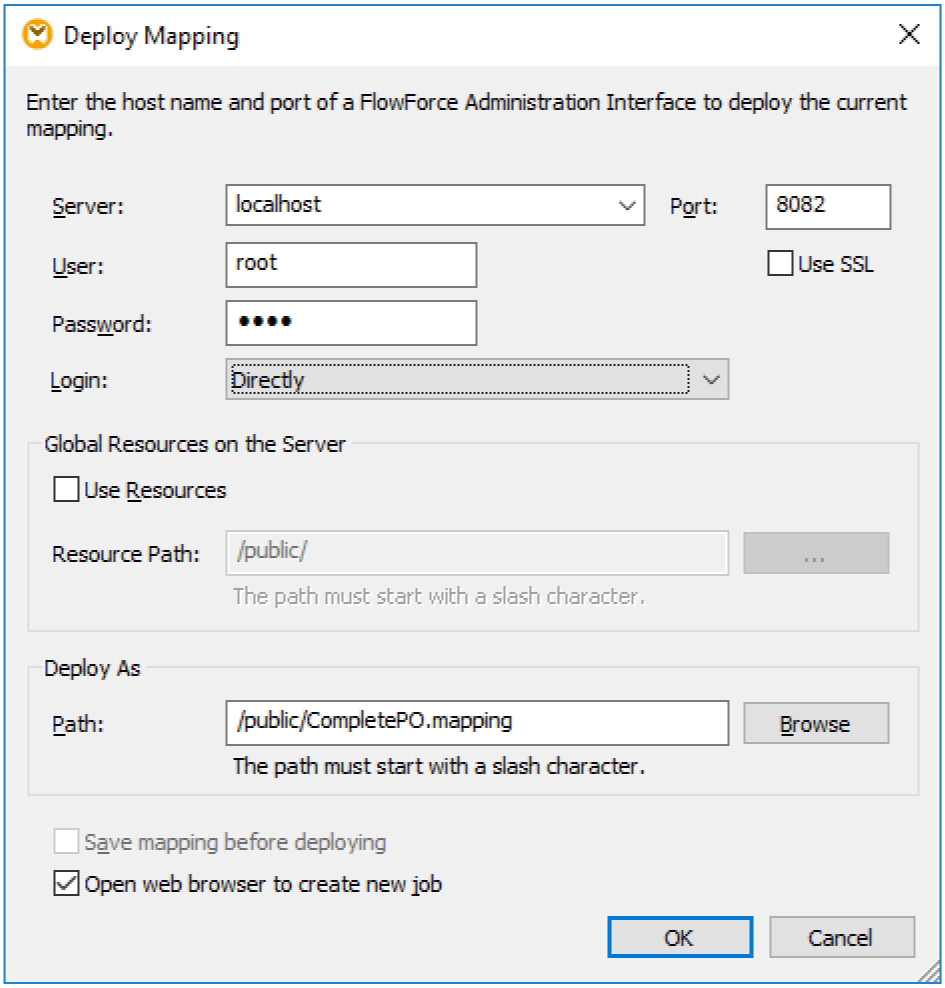
- All kinds of input instance files (XML, CSV, Text, EDI, Excel, JSON, XBRL) that are used by the mapping.
- If the mapping uses EDI and FlexText configurations, these are deployed in the state in which they exist on the source machine at the moment of deployment. If you change the EDI or FlexText configurations after deployment, the mapping must be re-deployed to FlowForce Server.
- If the mapping uses XBRL taxonomies or XBRL taxonomy packages, these are deployed in the state in which they exist on the source machine at the moment of deployment.

Prerequisites

See Preparing Mappings for Server Execution 1372.

Deploying the mapping to FlowForce Server

1. Run MapForce and ensure that the transformation language is set to BUILT-IN (either click the **Built-in** toolbar button or select the **Output | Built-in Execution Engine** menu command).
2. On the **File** menu, click Deploy to **FlowForce Server**. The Deploy Mapping dialog box opens.



3. Enter your deployment settings (as described below), and click OK. If you selected the **Open web browser to create new job** check box, the FlowForce Server administration interface opens in the browser, and you can start creating a FlowForce Server job immediately.

The following table lists the mapping deployment settings available on the Deploy Mapping dialog box.

Setting	Description
Server, Port, Use SSL	Enter the server host name (or IP address) and port of FlowForce Server. These could be **localhost** and **8082** if FlowForce Server is running on the same machine at the default port. When in doubt, log on to FlowForce Server Web administration interface and check the I.P. address and port displayed in the Web browser's address bar. If you encounter connectivity errors, ensure that the machine on which FlowForce Server runs is configured to allow incoming connections on the designated address and port. To deploy the mapping through a SSL-encrypted connection, select the **Use SSL** check box. This assumes that FlowForce Server is already

Setting	Description
	configured to accept SSL connections. For more information, refer to FlowForce Server documentation (https://www.altova.com/documentation).
User and Password	The user name and password to be entered depends on the value of the Login drop-down list (see next option). If the Login drop-down list is set to **<Default>** or **Directly**, enter your FlowForce Server user name and password. Otherwise, enter your domain user name and password, and select the domain name from the Login drop-down list.
Login	If Directory Service integration is enabled in FlowForce Server, select the domain name from this drop-down list, and enter your domain credentials in the User and Password fields (see previous option).
Use Resources, Resource Path	Select the **Use Resources** check box if the mapping function should use Resources 1397 after it is deployed to the server. If you select the check box, you must also enter the path of the respective resource on the server in the **Resource Path** text box. To select the resource, click the **Ellipsis** button. If there are no resources on the server yet to choose from, click **Deploy Global Resources** and deploy the required Global Resource to the server. For more information, see Deploying Global Resources to FlowForce Server 1416. If you do not select the **Use Resources** check box, any Global Resources will be resolved, based on the currently selected configuration. On the server, the mapping function will no longer require Global Resources, but will use the resolved value instead.
Path	Click **Browse**, and select the path where the mapping function should be saved in the FlowForce Server container hierarchy. By default, the path is set to the **/public** container of FlowForce Server. From the dialog box, you can also create new containers or delete existing containers and mappings, provided that you have the required FlowForce Server permissions and privileges.
Save mapping before deploying	This option is available if you are deploying an unsaved mapping. Select this check box to save the mapping before deployment.
Open browser to create new job	If you select this check box, the FlowForce Server Web administration interface opens in the browser after deployment, and you can start creating a FlowForce Server job immediately.

Troubleshooting

The following table lists problems that you might encounter when deploying a mapping, and their solution.

Problem	Solution
Deploying the mapping returns the following error:	Make sure that, on the target machine, the *FlowForce*

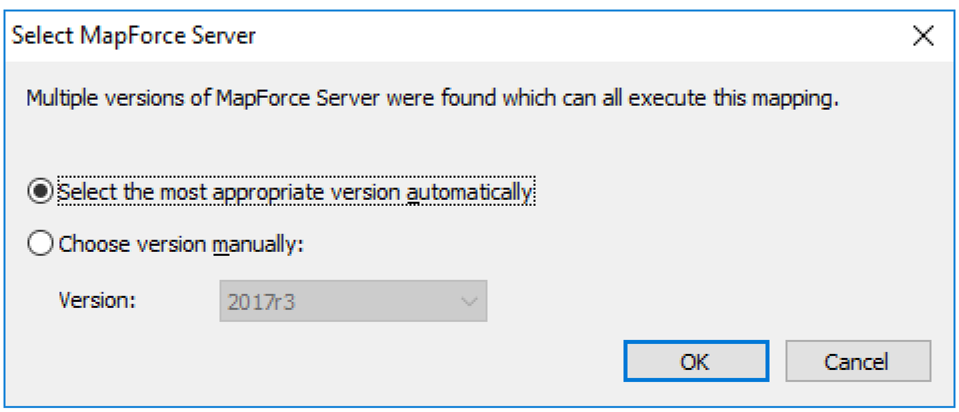
Problem	Solution

`I/O operation on file ... failed.`
`I/O Error 28: Failed to connect to`
`<server> port 8082. Timed out`
`System error 10060: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.`
	Web Server service is running and configured to listen for connections on the specified port (**8082**, by default). Also, make sure that the firewall does not block incoming connections through this port.
	
	The *FlowForce Server* service must be running as well in order for the deployment to be possible.

Deploying the mapping returns the following error:
 	
`I/O operation on file ... failed.`
`I/O Error 413: Payload Too Large`
	This error may occur if an input file of the deployed mapping exceeds the maximum size limit of HTTP requests allowed by FlowForce Server (roughly 100 MB). You can increase the limit by setting the `max_request_body_size` option (in bytes) in the **flowforceweb.ini** and **flowforce.ini** files. For details, see the FlowForce Server documentation.

Selecting the server version (Windows only)

If the server where you deploy the mapping has multiple versions of MapForce Server running under FlowForce Server management (applicable to Windows servers only), then you are additionally prompted to specify the version of MapForce Server with which you want this mapping to be executed.



Note: The dialog box appears when the FlowForce Server installation directory contains .tool files for each MapForce Server version which runs under FlowForce Server management. By default, a MapForce Server .tool file is added automatically to this directory when you install MapForce Server as part of FlowForce Server installation. The path where the .tool files are stored in FlowForce is: **C:\Program Files\Altova\FlowForceServer2023\tools**. If you have additional versions of MapForce Server which you want to run under FlowForce Server management, their .tool files may need to be copied manually to the directory above. The .tool file of MapForce Server can be found at: **C:\Program Files\Altova\MapForceServer2023\etc**.

10.6 AS2 Integration

AS2 (Applicability Statement 2) is a specification that enables exchanging files securely over the Internet. AS2 is used by businesses to exchange primarily EDIINT (EDI over Internet) and XML files through either HTTP or HTTPS.

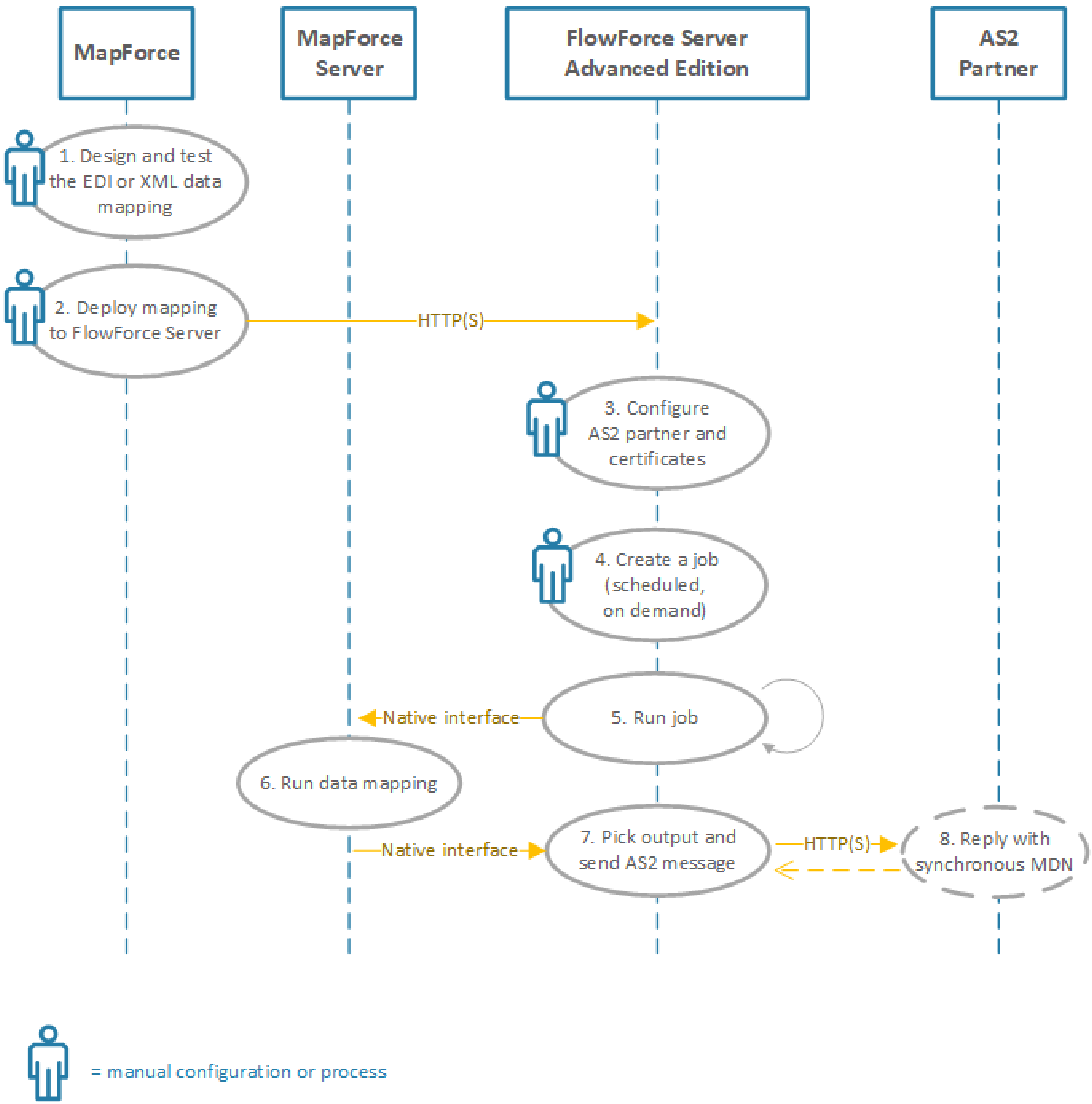
FlowForce Server Advanced Edition provides the functionality required to send AS2 messages to trading partners, or receive AS2 from trading partners. In addition, FlowForce Server is capable of processing AS2 data and storing it locally, with the help of its built-in set of functions. For even more advanced needs, if you need to prepare AS2 data from some existing source (for example, a database), or convert it to other formats, or send it to some Web service, you can also include MapForce and MapForce Server into the AS2 process.

There are two main scenarios where MapForce and MapForce Server are necessary:

1. To map or generate data in any format supported by MapForce (such as XML, XBRL, Excel, databases, Web services), before sending it to AS2 partners.
2. To transform data received from AS2 partners in a variety of ways (for example, convert it to Excel, convert it to a different XML schema, store it in a database, send it to a Web service, and so on).

Generating and sending AS2 data

In a scenario where you need to prepare or generate AS2 data with MapForce before sending it to partners, the high-level process looks as follows:



Generating and sending AS data

In the diagram above, both MapForce Server and FlowForce Server must be installed on the same machine (it can be a Windows, Linux, or macOS operating system). MapForce may run on the same machine as MapForce Server and FlowForce Server (provided that it's a Windows machine), or on a different machine that can connect to FlowForce Server via HTTP or HTTPS. The AS2 partner is a remote server with which FlowForce Server communicates through HTTP(S).

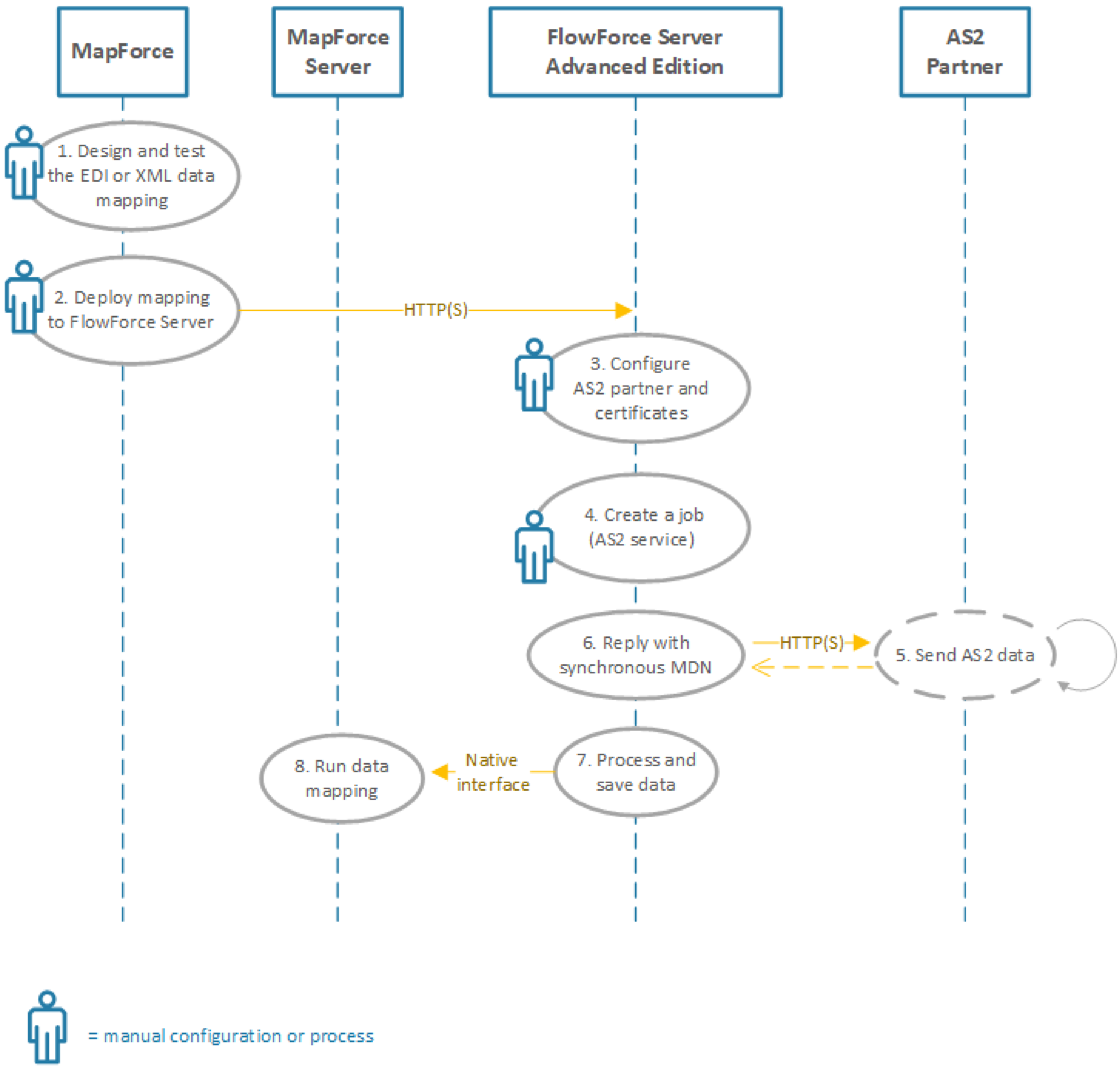
The AS2 process illustrated above works as follows:

Step #	Description
1. Design and test the EDI/XML data mapping	With MapForce, you can design a data mapping transformation that takes as input data in various formats (including plain text, CSV, JSON, XML, various EDI flavors, databases, Web services) and outputs one or several files in a destination format (for example, UN/EDIFACT). Designing a mapping for EDI purposes is not different to other mappings, and various such examples are included in MapForce documentation, see the EDI chapter. While you design the mapping, you can validate and preview the mapping output directly in MapForce, by clicking the Output tab. To ensure that the mapping is suitable for execution in a server environment, you will need to design and test it for the BUILT-IN transformation language, see also Transformation Languages 27.
2. Deploy mapping to FlowForce Server	FlowForce Server automates various tasks by means of on demand or scheduled jobs that can be defined from a Web interface. FlowForce Server can also automate the execution of a mapping designed with MapForce, provided that MapForce Server runs under FlowForce Server management. Once the MapForce mapping produces the required output, you are ready to automate its execution, by deploying it to FlowForce Server. For instructions, see Deploying Mappings to FlowForce Server 1381.
3. Configure AS2 partner and certificates	To set up the communication with AS2 partners, you will need to obtain their AS2 connectivity details (such as URI and AS2 name), and exchange certificates. The certificates must be imported (and partner details must be entered) into FlowForce Server.
4. Create a job	A FlowForce job must be created in order to (a) run the mapping and produce the required output, and (b) send the AS2 message (see also step 7). These two actions may be either execution steps of the same job, or two different jobs altogether.
5. Run job	The FlowForce Server job created in the previous step may be configured to run in various ways, depending on your business needs. For example, it can run as a Web service call, or whenever a file changes on the file system, or it could be scheduled to occur at a specific time and date. This step is fully automated.
6. Run data mapping	This step also takes place automatically and is executed by MapForce Server. If a job is configured to execute a data mapping (be it scheduled or on demand), an internal call to MapForce Server takes place. As a result, MapForce Server runs the mapping and returns the output to FlowForce Server.
7. Pick output and send AS2 message	In order to send the AS2 message, your job (or execution step within a job) must call the FlowForce Server built-in function `/system/as2/send`. This function takes a number of parameters required to send the AS2 message, including the partner object configured in step 3, the partner's URI, and the AS2 message content that you want to send. Your job may also need to call various

Step #	Description
	FlowForce Server AS2 expression functions in order to convert the mapping output to the required form (for example, from a file to a stream).
8. Partner replies with synchronous MDN	When you create the AS2 partner object in step 3, you may optionally request that the partner send a Message Disposition Notification (MDN) in reply to the AS2 message sent by FlowForce Server. The partner must send the MDN in the same session as the HTTP call outgoing from FlowForce Server (that is, it must be configured as "synchronous").

Receiving and processing AS2 data

If your organization receives AS2 data from trading partners, you can additionally configure a data receiving workflow. In this scenario, your organization would be able to not only receive and store AS2 data, but also transform it to other formats, save it to a database, or send it to another Web service. For example, you could receive files in EDI or XML format from AS2 trading partners and then supply them as input to some mapping that runs as a recurrent FlowForce Server job. In this scenario, an example AS2 process looks as follows:



Receiving and processing AS2 data

The example AS2 process illustrated above works as follows:

Step #	Description
1, 2, 3	These are the same steps as in the previous table. The only difference is that this time the mapping is expected to take as input some file that your organization expects to receive from an AS2 trading partner (for example, an EDI or XML file).

Step #	Description
4. Create a job (AS2 service)	This is a one-time step. In this step, you create a FlowForce Server job that exposes an AS2 service. The AS2 service listens for requests from your AS2 partners at a configured HTTP(S) address and port.
5. Send AS2 data	In this step, a trading partner submits AS2 messages to the AS2 service. For communication to be successful, the partner's AS2 name and certificates must already be defined in FlowForce Server.
6. Reply with synchronous MDN	FlowForce Server replies to the AS2 partner with a synchronous MDN that indicates the outcome of the operation (success or error).
7. Process and save data	As soon as there is an incoming message, a FlowForce Server job converts the received data to a string or a file, and then stores it in some directory, or passes it to another job as argument. The exact processing logic is configurable with the help of FlowForce Server built-in and expression functions.
8. Run data mapping	The FlowForce Server job that receives AS2 data may optionally invoke the data mapping job that was created in the first step. The mapping job takes as input the AS2 data received from the partner and then processes it in any of the ways supported by MapForce: for example, transforms it to another format, saves it to a database, sends it to another Web service, and so on.

For the AS2 configuration details specific to FlowForce Server, refer to the documentation of FlowForce Server Advanced Edition (see https://www.altova.com/documentation).

10.7 MapForce Command Line Interface

The general syntax of a MapForce command at the command line is:

```
MapForce.exe <filename> [/{target} [[<outputdir>] [/options]]]
```

Legend

The following notation is used to indicate command line syntax:

Notation	Description
Text without brackets or braces	Items you must type as shown
<Text inside angle brackets>	Placeholder for which you must supply a value
[Text inside square brackets]	Optional items
{Text inside braces}	Set of required items; choose one
Vertical bar (|)	Separator for mutually exclusive items; choose one
Ellipsis (...)	Items that can be repeated

<filename>

The mapping design (.mfd) or mapping project (.mfp) file from which code is to be generated. To generate code for the whole project, set the target `/GENERATE` (see description below) and enter the project path as `<filename>`, for example, **MapForceExamples.mfp**.

/{target}

Specifies the target language or environment for which code is to be generated. The following code generation targets are supported.

Target	Description
`/COMPILE[:compileoptions]`	Compiles a mapping to a MapForce Server execution file (.mfx). Optionally, the following options can be supplied, delimited by a comma: `JDBC` — Transforms all database connections to JDBC using the JDBC driver and the database URL defined in the Database Component Settings dialog box, see also Database Component Settings[279]. `NOXMLSIGNATURES` — Suppresses the generation of digital signatures in the MapForce Server Execution file (note that digital signatures are not supported by MapForce Server).

Target	Description
`/GENERATE`	Generates project code for all mappings in the project file using the current folder settings, see Managing Project Folders [119]. If you select this target, make sure to supply a MapForce project (.mfp file) as `<filename>`.
`/XSLT`	Generates XSLT 1.0 code.
`/XSLT2`	Generates XSLT 2.0 code.
`/XSLT3`	Generates XSLT 3.0 code.
`/XQuery`	Generates XQuery code.
`/JAVA`	Generates Java code.
`/CS`	Generates C# code. This command also optionally allows setting specific options for code generation, namely: `/CS[:{VS2010|VS2013|VS2015|VS2017|VS2019}]` `VS2010` Visual Studio 2010 `VS2013` Visual Studio 2013 `VS2015` Visual Studio 2015 `VS2017` Visual Studio 2017 `VS2019` Visual Studio 2019 If no Visual Studio version is specified, code will be generated using the Visual Studio version defined in the code generation options, see Code Generator Options [1625].
`/CPP`	Generates C++ code. This command also optionally allows setting specific code generation options, namely: `/CPP[:{VS2010|VS2013|VS2015|VS2017},{MSXML|XERCES3},{LIB|DLL},{MFC|NoMFC}]` The first option group sets the target Visual Studio version. Valid values: `VS2010` Visual Studio 2010 `VS2013` Visual Studio 2013 `VS2015` Visual Studio 2015 `VS2017` Visual Studio 2017

Target	Description
	`VS2019` Visual Studio 2019 The second option group specifies the XML library targeted by the generated code. Valid values: `MSXML` Generate code for MSXML 6.0 `XERCES3` Generate code for Xerces 3 The third option group specifies whether static as opposed to dynamic libraries should be generated. Valid values: `LIB` Generate static LIB libraries `DLL` Generate DLL libraries The fourth option group specifies whether code should be generated with or without MFC support. Valid values: `MFC` Enable MFC support `NoMFC` Disable MFC support If the options above are not specified, code will be generated using the Visual Studio version defined in the code generation options, see Code Generator Options [1625].

<outputdir>

Optional parameter which specifies the output directory. If an output path is not supplied, the current working directory will be used. Note that any relative file paths are relative to the current working directory.

When target is `/GENERATE` and the `<outputdir>` parameter is not set, the code generation language, as well as the output path of each mapping, are supplied by the settings defined for each folder inside the project, see Managing Project Folders [119].

When target is `/GENERATE` and the `<outputdir>` parameter is set, the `<outputdir>` value supplied at the command line takes precedence over the output directory defined at the root project level. It does not take precedence, however, over the code generation settings defined at each folder inside the project.

/options

The `/options` are not mutually exclusive. One or more of the following options can be set.

Option	Description
`[/MFXVERSION[:<version>]]`	This option is applicable if the target is `/COMPILE`. It compiles the MapForce Server Execution (.mfx) file for a particular version of MapForce Server. You can supply as value any version of MapForce Server, starting with **2013r2** onwards, up to the current MapForce version. See also Compiling mappings for a specific MapForce Server version 1379.
`/GLOBALRESOURCEFILE <filename>`	This option is applicable if the mapping uses Global Resources to resolve input or output file or folder paths, or databases. For more information, see Altova Global Resources 1397. The option `/GLOBALRESOURCEFILE` specifies the path to a Global Resource .xml file. Note that, if `/GLOBALRESOURCEFILE` is set, then `/GLOBALRESOURCECONFIG` must also be set.
`/GLOBALRESOURCECONFIG <config>`	This option specifies the name of the Global Resource configuration (see also the previous option). Note that, if `/GLOBALRESOURCEFILE` is set, then `/GLOBALRESOURCECONFIG` must also be set.
`/LIBRARY <libname> (...)`	Use together with a code generation target language to specify additional function libraries. This option can be specified more than once to load multiple libraries. See also Managing Function Libraries 908.
`/LOG <logfilename>`	Generates a log file at the specified path. `<logfilename>` can be a full path name, for example, it can include both a directory and a file name. However, if a full path is supplied, the directory must exist for the log file to be generated. If you specify only the file name, then the file will be placed in the current directory of the Windows command prompt.

Remarks

- Relative paths are relative to the working directory, which is the current directory of the application calling MapForce. This applies to the path of the .mfd filename, .mfp filename, output directory, log filename, and global resource filename.
- Do not use the end backslash and closing quote at the command line (for example, "C:\My directory\"). These two characters are interpreted by the command line parser as a literal double quotation mark. Use the double backslash \\ if spaces occur in the command line and you need the quotes ("c:\My Directory\\"), or try to avoid using spaces and therefore quotes at all.

Examples

1) To start MapForce and open the mapping `<filename>.mfd`, use:

```
MapForce.exe <filename>.mfd
```

2) To generate XSLT 2.0 code and also create a log file with the name `<logfilename>`, use:

```
MapForce.exe <filename>.mfd /XSLT2 <outputdir> /LOG <logfilename>
```

3) To generate XSLT 2.0 code taking into account the global resource configuration `<grconfigname>` from the global resource file `<grfilename>`, use:

```
Mapforce.exe <filename>.mfd /XSLT2 <outputdir> /GLOBALRESOURCEFILE
<grfilename> /GLOBALRESOURCECONFIG <grconfigname>
```

4) To generate a C# application for Visual Studio 2015 and output a log file, use:

```
MapForce.exe <filename>.mfd /CS:VS2015 <outputdir> /LOG <logfilename>
```

5) To generate a C++ application using the code generation settings defined in **Tools | Options**, and output a log file, use:

```
MapForce.exe <filename>.mfd /CPP <outputdir> /LOG <logfilename>
```

6) To generate a C++ application for Visual Studio 2015, MSXML, with static libraries, MFC support, and no log file, use:

```
MapForce.exe <filename>.mfd /CPP:VS2015,MSXML,LIB,MFC
```

7) To generate a C++ application for Visual Studio 2015, Xerces, with dynamic libraries, no MFC support, and a log file, use:

```
MapForce.exe <filename>.mfd /CPP:VS2015,XERCES,DLL,NoMFC <outputdir> /LOG <logfilename>
```

8) To generate a Java application and also output a log file, use:

```
MapForce.exe <filename>.mfd /JAVA <outputdir> /LOG <logfilename>
```

9) To generate code for all mappings in the project, using the language and output directory defined in the folder settings (of each folder inside the project), use:

```
MapForce.exe <filename>.mfp /GENERATE /LOG <logfilename>
```

10) To generate Java code for all mappings in the project file, use:

```
MapForce.exe <filename>.mfp /JAVA /LOG <logfilename>
```

Note that the code generation language defined in the folder settings are ignored, and Java is used for all mappings.

11) To supply input and output files at the command line for a previously compiled Java mapping, use:

```
java -jar <mappingfile>.jar /InputFileName <inputfilename> /OutputFileName
<outputfilename>
```

The `/InputFileName` and `/OutputFileName` parameters are the names of special input components in the MapForce mapping that allow you to use parameters in command line execution (see Supplying Parameters to the Mapping 833).

12) To compile a mapping to a MapForce Server execution file, for MapForce Server version 2023, and suppress XML signatures:

```
MapForce.exe <filename>.mfd /COMPILE:NOXMLSIGNATURES <outputdir> /MFXVERSION:2023 /LOG
<logfilename>
```

11 Altova Global Resources

Altova Global Resources are aliases for file, folder, and database resources. Each alias can have multiple configurations, and each configuration maps to a single resource. Therefore, when you use a global resource, you can switch between its configurations. For example, you could create a database resource with two configurations: `development` and `production`. Depending on your goals, you can switch between these configurations. In MapForce, you can retrieve data from the `development` or `production` database by choosing the desired configuration from the drop-down list before previewing the mapping.

Global resources can be used across different Altova applications (*see subsection below*).

Global resources in other Altova products

When stored as global resources, files, folders, and database connection details become reusable across multiple Altova applications. For example, if you often need to open the same file in multiple Altova desktop applications, you can define this file as a global resource. If you need to change the file path, you will need to change it only in one place. Currently, global resources can be defined and used in the following Altova products:

- Altova Authentic
- DatabaseSpy
- MobileTogether Designer
- MapForce
- StyleVision
- XMLSpy
- FlowForce Server
- MapForce Server
- RaptorXML Server/RaptorXML+XBRL Server

In this section

This section explains how to create and configure different types of global resources. The section is organized into the following topics:

11.1 Global Resource Setup Part 1

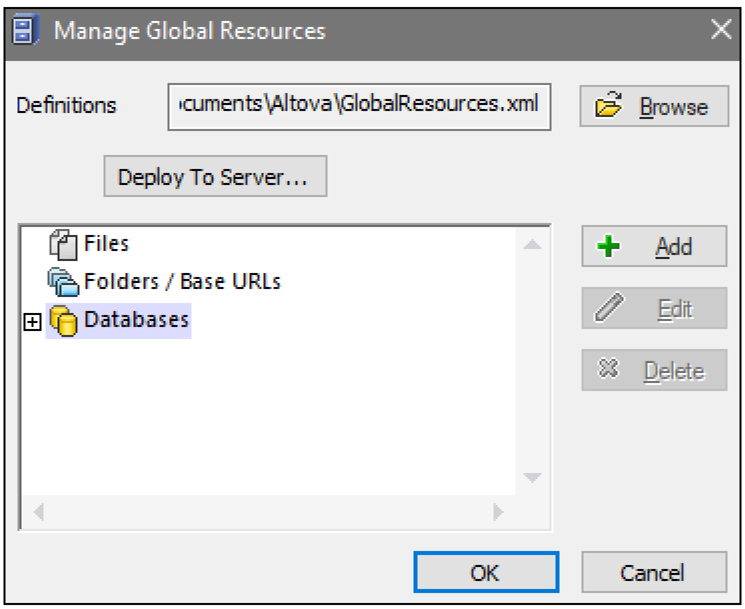
The global resource setup consists of two parts: (i) creating a global resource in the **Manage Global Resources** dialog box (*see below*) and (ii) defining the properties of this global resource in the **Global Resource** dialog. The second part is discussed in the next topic 1400.

Altova Global Resources are defined in the **Manage Global Resources** dialog, which can be accessed in two ways:

- Click the menu command **Tools | Global Resources**.
- Click the **Manage Global Resources** icon in the Global Resources toolbar (*screenshot below*).

Global Resources Definitions file

Information about global resources is stored in an XML file called the Global Resources Definitions file. This file is created when the first global resource is defined in the **Manage Global Resources** dialog box (*screenshot below*) and saved.



When you open the **Manage Global Resources** dialog box for the first time, the default location and name of the Global Resources Definitions file is specified in the *Definitions* text box (*see screenshot above*):

```
C:\Users\<username>\Documents\Altova\GlobalResources.xml
```

This file is set as the default Global Resources Definitions file for all Altova applications. A global resource can be saved from any Altova application to this file and will immediately be available to all other Altova applications as a global resource. To define and save a global resource to the Global Resources Definitions file, add the global resource in the **Manage Global Resources** dialog and click **OK** to save.

To select an already existing Global Resources Definitions file to be the active definitions file of a particular Altova application, browse for it via the **Browse** button of the *Definitions* text box (*see screenshot above*).

The **Manage Global Resources** dialog box also allows you to edit and delete existing global resources.

Notes:

- You can give any name to the Global Resources Definitions file and save it to any location accessible to your Altova applications. All you need to do in each application, is specify this file as the Global Resources Definitions file for that application (in the *Definitions* text box). The resources become global across Altova products when you use a single definitions file across all Altova products.

- You can also create multiple Global Resources Definitions files. However, only one of these can be active at any time in a given Altova application, and only the definitions contained in this file will be available to the application. The availability of resources can therefore be restricted or made to overlap across products as required.

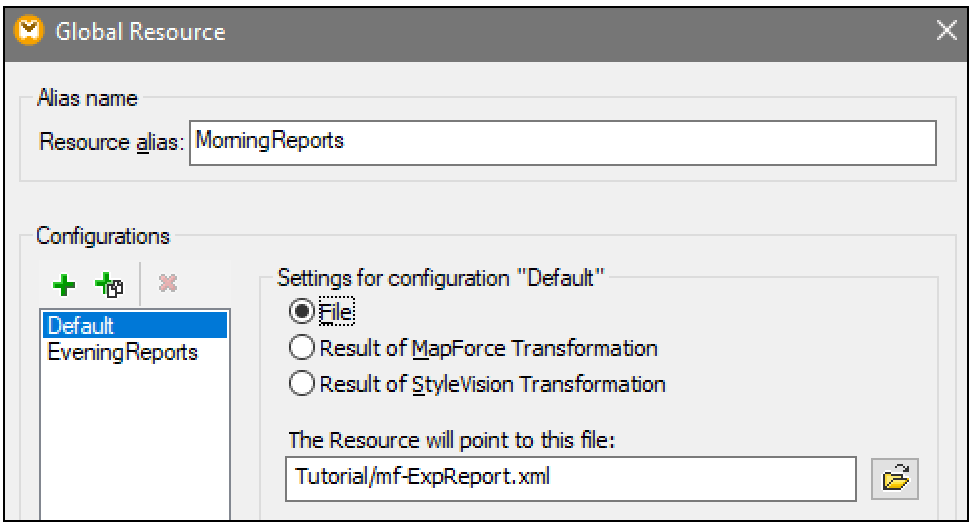
11.2 Global Resource Setup Part 2

The second part of the global resource setup consists in defining properties of a global resource in the **Global Resource** dialog box. The properties depend on the type of a global resource (*see subsections below*). You can access the **Global Resource** dialog box by clicking the **Add** button in the Manage Global Resources dialog box [1398].

To find out more about setting up different types of global resources, see the following examples: XML Files as Global Resources [1404], Folders as Global Resources [1406], Databases as Global Resources [1408].

Files

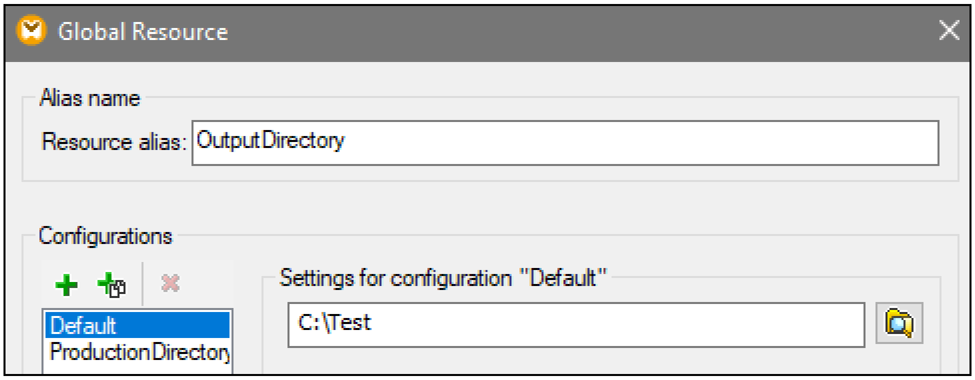
The file-specific properties are shown in the **Global Resource** dialog box below. The setup has three major parts: (i) the name of the file, (ii) the location of this file, and (iii) the list of configurations defined for this file alias.



The settings *Result of MapForce Transformation* and *Result of StyleVision Transformation* are discussed in Transformation Results as Global Resources [1410].

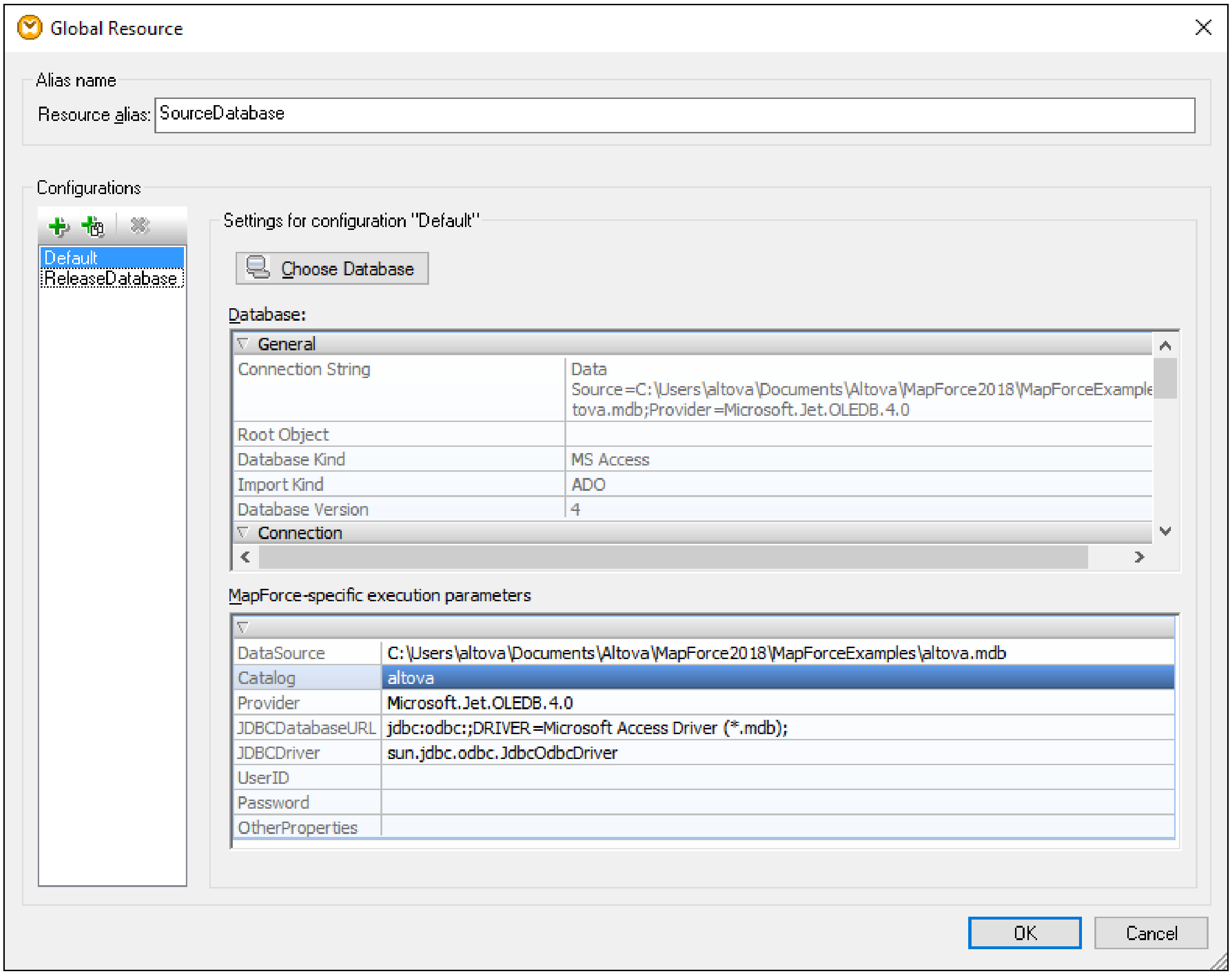
Folders

The folder-specific properties are shown in the **Global Resource** dialog box below. The setup has three major parts: (i) the name of the folder, (ii) the location of this folder, and (iii) the list of configurations defined for this folder alias.



Databases

When you add a database connection as a global resource, a connection wizard guides you through the steps required to set up the connection. For more information, see Start Database Connection Wizard 165. Once you complete the wizard, the database connection parameters are displayed in the **Global Resource** dialog box (*see screenshot below*).



In the **Global Resource** dialog box, it is possible to edit some of the database connection parameters. The parameters are grouped into two categories: database parameters and MapForce-specific execution parameters (*see below*).

Database

These parameters are shared among Altova applications. In MapForce, they are used at design time, that is, when the mapping is loaded, or when you click the **Output** pane in MapForce to preview the mapping.

MapForce-specific execution parameters

These parameters are applicable when you generate program code or compile a mapping to MapForce Server execution file (`.mfx`). They are used at mapping runtime as follows:

- In generated C++, C#, or Java program code.
- If you compile the mapping to a MapForce Server execution file, and automatic JDBC conversion takes place. For more information about automatic JDBC conversion, see Database mappings in various execution environments 162.

If a mapping uses a Global Resource to connect to a database, the database connection details in the **Global Resource** dialog box take precedence over those defined in the mapping. The Component Settings dialog box informs you that the connectivity parameters are defined as a Global Resource. To change the database component to connect to the database directly (without using Global Resources), click **Change**, and follow the wizard steps to reconnect to the database.

Global Resource dialog icons

	Add Configuration: Pops up the Add Configuration dialog in which you enter the name of the configuration to be added.
	Add Configuration as Copy: Pops up the Add Configuration dialog in which you can enter the name of the configuration to be created as a copy of the selected configuration.
	Delete: Deletes the selected configuration.
	Open: Browse for the file to be created as the global resource.
	Open: Browse for the folder to be created as the global resource.

Global resource setup: General procedures

The broad procedure of creating and configuring global resources is described below:

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add** and select the resource type you wish to create (file, folder, database). The **Global Resource** dialog box will appear.
3. Enter a descriptive name in the **Resource alias** text box (e.g., `InputFile`).
4. Setting up the default configuration depends on the type of the global resource: (i) For a file or folder, browse for the file or folder to which this resource should point by default; (ii) for a database connection, click **Choose Database** and follow the Database Connection Wizard to connect to the database (see Connecting to a Database 163). This database connection will be used by default when you run the mapping.
5. If you need an additional configuration (e.g., an additional output folder), click the button in the **Global Resource** dialog box, enter the name of this configuration, and specify the path to this configuration.
6. Repeat the previous step for each additional configuration required.

Note: Database connections are supported as global resources only in MapForce Professional and Enterprise editions.

11.3 XML Files as Global Resources

This topic explains how to use XML files as global resources. There are situations in which you may need to change an input XML file multiple times per day. For example, every morning you need to run a particular mapping and generate a report by using one XML file as a mapping input, and every evening the same report must be generated from another XML file. Instead of editing the mapping multiple times per day (or keeping multiple copies of it), you could configure the mapping to read from a file defined as a global resource (the so-called *file alias*). In this example, our file alias will have two configurations:

1. The `Default` configuration will supply a morning XML file as a mapping input.
2. The `EveningReports` configuration will supply an evening XML file as a mapping input.

To create and configure the file alias, take the steps below.

Step 1: Create a global resource

First, we need to create a file alias. Follow the instructions below:

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add | File** and enter a name in the **Resource alias** text box. In this example, we call our default configuration `MorningReports`.
3. Click the **Browse** button near the text field **The Resource will point to this file** and select **`Tutorial\mf-ExpReport.xml`**.
4. Click in the **Configurations** section and name the second configuration `EveningReports`.
5. Click **Browse** and select **`Tutorial\mf-ExpReport2.xml`**.

Step 2: Use the global resource in the mapping

Now we can use the newly created global resource in our mapping. To make the mapping read data from the global resource, take the steps below:

1. Open the **`Tutorial\Tut-ExpReport.mfd`** mapping.
2. Double-click the header of the source component to open the **Component Settings** dialog box.
3. Next to **Input XML file**, click **Browse**, then click **Global Resources** and select the file alias `MorningReports`. Click **Open**.
4. Open the **Component Settings** dialog box again: The input XML file path has now become **`altova://file_resource/MorningReports`**, which indicates that the path uses a global resource.

Step 3: Run the mapping with the desired configuration

You can now switch between the input XML files before running the mapping:

- To use **`mf-ExpReport.xml`** as an input, select the menu item **Tools | Active Configuration | Default**.
- To use **`mf-ExpReport2.xml`** as an input, select the menu item **Tools | Active Configuration | EveningReports**.

Alternatively, select the required configuration from the **Global Resources** drop-down list in the toolbar (*see screenshot below*).

To preview the mapping result with either configuration, click the **Output** pane.

11.4 Folders as Global Resources

This topic explains how to use folders as global resources. There are situations in which you may need to generate the same output in different directories. To make this possible, we will create a folder alias with two configurations:

1. The `Default` configuration will generate the output in **`C:\Test`**.
2. The `Production` configuration will generate the output in **`C:\Production`**.

To create and configure the folder alias, take the steps below.

Step 1: Create a global resource

First, we need to create a folder alias. Follow the instructions below:

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add | Folder** and enter a name in the **Resource alias** text box. In this example, we call our default configuration `OutputDirectory`.
3. Click the **Browse** button near the text field **Settings for configuration "Default"** and select **`C:\Test`**. Make sure this folder already exists on your operating system.
4. Click and enter a name of the second configuration. In this example, we call our second configuration `ProductionDirectory`.
5. Click **Browse** and select the **`C:\Production`** folder. Make sure that this folder already exists on your operating system.

Step 2: Use the global resource in the mapping

The next step is to make the mapping use the folder alias we have just created. Take the steps below:

1. Open the **`Tutorial\Tut-ExpReport.mfd`** mapping.
2. Double-click the header of the target component to open the **Component Settings** dialog box.
3. Click **Global Resources** and then click **Save**.
4. Save the output XML file as **`Output.xml`**. The output XML file path has now become **`altova://folder_resource/OutputDirectory/Output.xml`**, which indicates that the path is defined as a global resource.

Step 3: Run the mapping with the desired configuration

You can now switch between the output folders before running the mapping:

- To use **`C:\Test`** as the output directory, select the menu item **Tools | Active Configuration | Default**.
- To use **`C:\Production`** as the output directory, select the menu item **Tools | Active Configuration | ProductionDirectory**.

By default, the mapping output is written as a temporary file unless you explicitly configure MapForce to write output to permanent files. To configure MapForce so that it generates permanent files, do the following:

1. Go to the **Tools** menu and click **Options**.
2. In the **General** section, select the option **Write directly to final output files**.

11.5 Databases as Global Resources

This topic explains how to use databases as global resources. There are situations in which you may need to map data from databases with the same structure but different data. Using a database resource will allow you to switch between databases without editing your mapping. To achieve this, we need to create a database alias with two configurations:

1. The `Default` configuration will point to the `DevelopmentDatabase`: **`MapForceExamples\altova.mdb`**.
2. The `ReleaseDatabase` configuration will point to **`Tutorial\altova.mdb`**.

To create and configure the database alias, take the steps below.

Step 1: Create a global resource

The fist step is to create a database alias. Take the steps bellow:

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add | Database** and enter a descriptive name in the **Resource alias** text box. In this example, we call the default configuration `DevelopmentDatabase`.
3. Click **Choose Database**, select **Microsoft Access (ADO)**, and browse for **`MapForceExamples\altova.mdb`**.
4. Click and name the second configuration `ReleaseDatabase`.
5. Click **Choose Database**, select **Microsoft Access (ADO)**, and browse for **`Tutorial\altova.mdb`**.

Step 2: Use the global resource in the mapping

The next step is to configure the mapping so that it can use the database alias:

1. Open the **`Tutorial\PersonDB.mfd`** mapping.
2. Double-click the database component to open the **Component Settings** dialog box. Click **Change**.
3. Select **Global Resources** in the **Select a Database** dialog box and select the `DevelopmentDatabase` alias. Click **Connect**.
4. When you are prompted to select database objects, leave the default selection as is and click **OK**.

The connectivity parameters can be changed by clicking the toolbar button.

Step 3: Run the mapping with the desired configuration

You can now easily switch between the databases before running the mapping:

- To use the `DevelopmentDatabase` configuration, select the menu item **Tools | Active Configuration | Default**.
- To use the `ReleaseDatabase` configuration, select the menu item **Tools | Active Configuration | ReleaseDatabase**.

Alternatively, select the required configuration from the **Global Resources** drop-down list (*see screenshot below*).

When you switch between configurations, the **Configuration switch** dialog box informs you that the resource has been modified. Click **Reload**.

Note: The databases used in this example contain the same data. Therefore, there are no differences in the generated output.

11.6 Transformation Results as Global Resources

You can use the result of a MapForce mapping or StyleVision transformation as a global resource. This topic shows how to create a global resource from the transformation result and use this global resource across different Altova applications.

In order to make a mapping output available as a global resource, the transformation language of the mapping must be set to Built-In, or the mapping must contain only components which are supported by the Built-In language.

Important:

- The workflows mentioned above are meaningful between Altova desktop applications installed on the same computer.
- It is **not** possible to use the result of MapForce and StyleVision transformations as global resources in Altova server products and in MapForce Basic Edition.

The example below shows how to use the result of a MapForce transformation as a global resource.

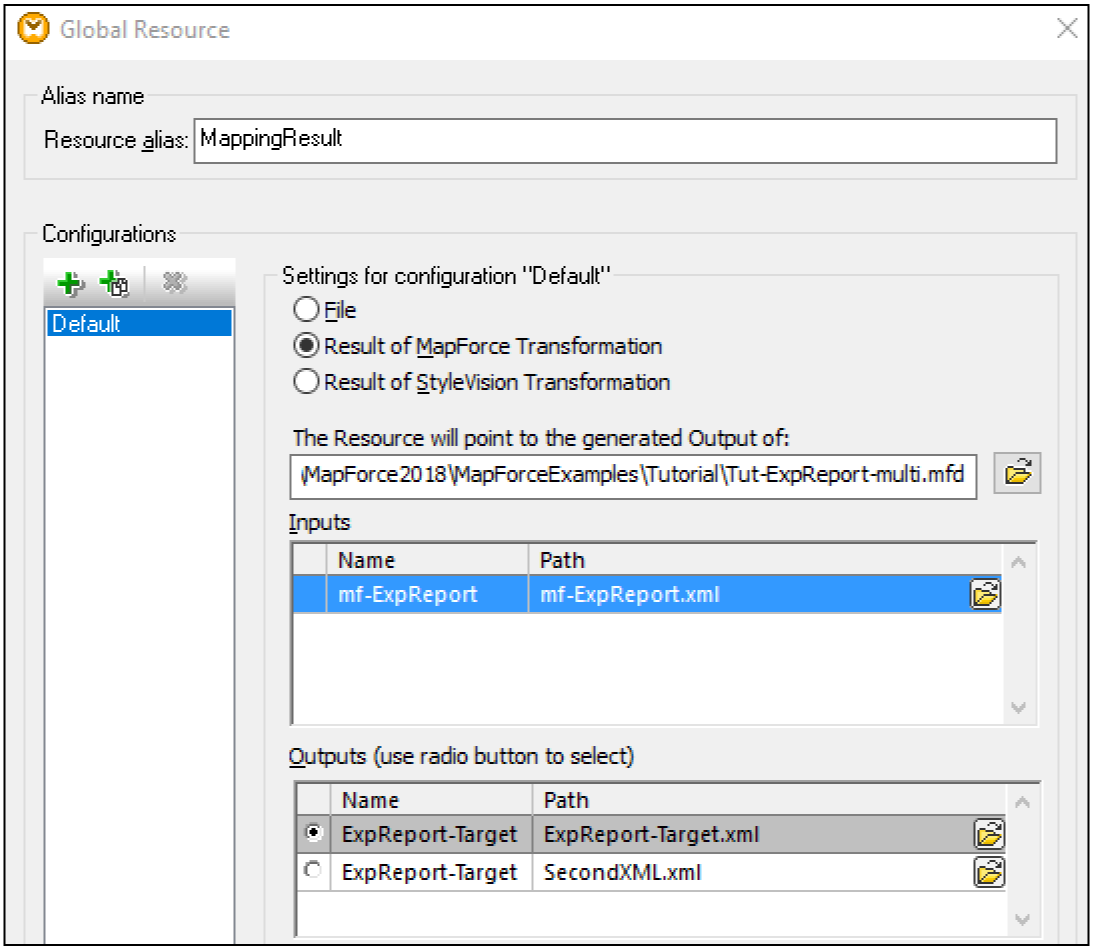
Example: Result of MapForce Transformation

This example illustrates how to create a workflow between Altova MapForce and Altova XMLSpy. More specifically, the example shows how to create a global resource from a MapForce mapping, trigger the execution of this mapping in XMLSpy, and view the outputs in XMLSpy, which were generated by MapForce.

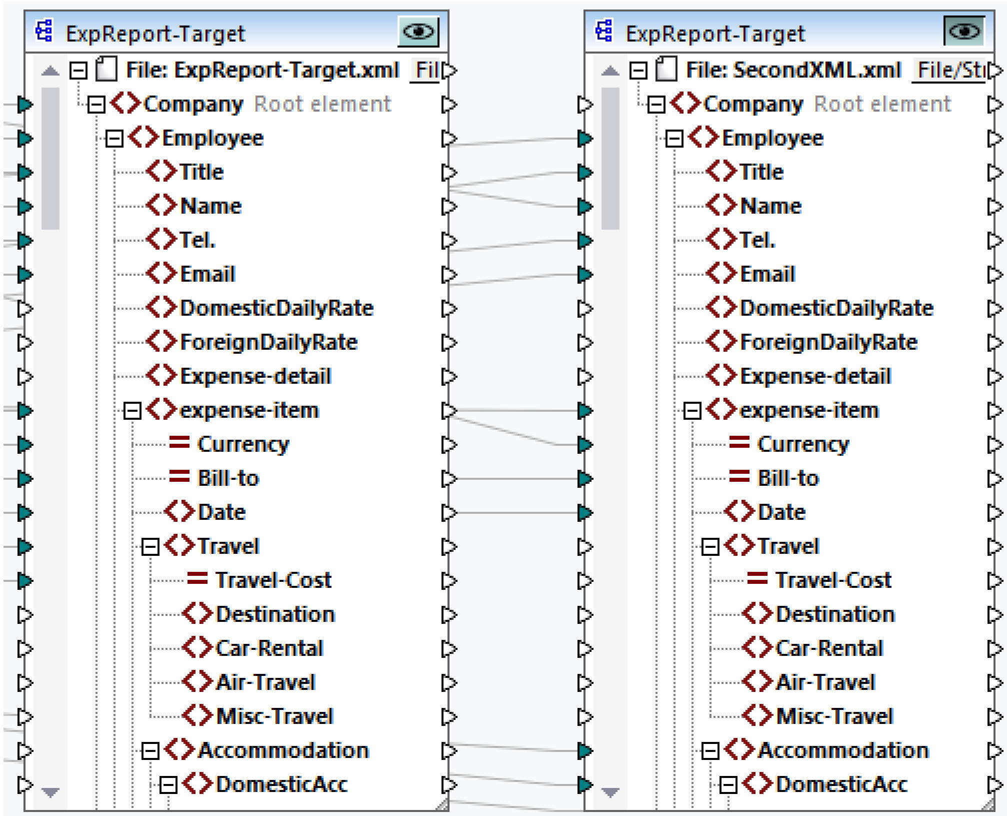
Step 1: Create a global resource

You can take this step using MapForce or XMLSpy.

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add | File** and enter a descriptive name in the **Resource alias** text box. In this example, we call our default configuration `MappingResult`.
3. Select the option **Result of MapForce Transformation**.
4. Click **Browse** and select the mapping **`Tutorial\Tut-ExpReport-multi.mfd`**. As shown below, this mapping has one input and two outputs.



The screenshot below illustrates the two outputs listed in the **Global Resource** dialog box. We will generate each output file separately in the **`C:\temp`** folder (*see Step 2 below*).



Step 2: Generate output files

At this stage, we would like to generate each of the two output files (*see screenshot above*) in the **`C:\temp`** folder and change the file names. To achieve this, we will create a configuration for each output. Take the steps below:

1. In the *Outputs* section of the **Global Resource** dialog box, click **Browse** next to the first output and enter **`C:\temp\file1.xml`** as a destination file name. This is the default configuration which will produce the first output file.
2. Click under *Configurations* and enter a name for the new configuration (in this example, `Output2`). In the *Outputs* section, click the radio button near the second file (**`SecondXML.xml`**).
3. In the *Outputs* section, click **Browse** next to the second output and enter **`C:\temp\file2.xml`** as a destination file name. This is the second configuration which produces the second output file.
4. Click **OK**.

Step 3: Use the global resource

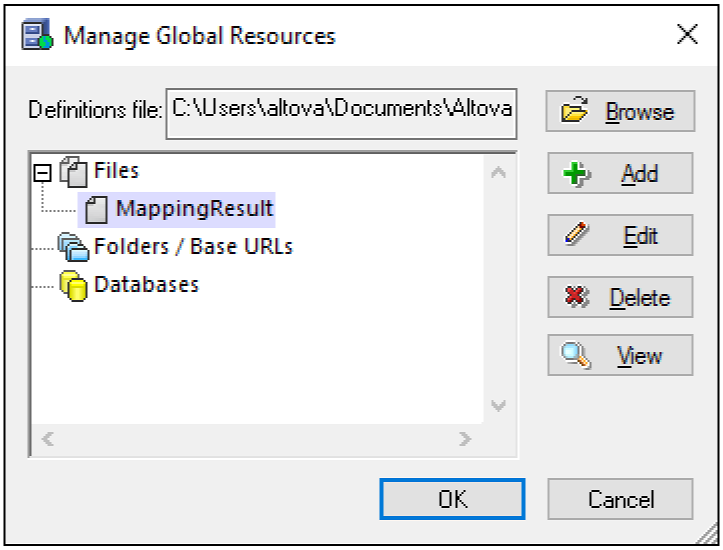
The instructions below show how to use the global resource we created in the previous step.

Default configuration

To use the default configuration in XMLSpy, take the steps below.

1. Run XMLSpy.

2. Go to the **Tools** menu and click **Global Resources**.
3. In the *Files* section, click the `MappingResult` global resource and then click **View** (*see screenshot below*). This executes the mapping, produces the default output (**`file1.xml`**) and loads it into the main pane of XMLSpy. The file is saved as **`C:\temp\file1.xml`**.



Second configuration
To trigger the mapping execution with the second configuration, do the following:

1. Go to the **Tools** menu in XMLSpy and click **Active Configuration | Output2**.
2. Click **Reload** when prompted.

As a result, the second output file is loaded into the main window of XMLSpy. The file is saved as **`C:\temp\file2.xml`**.

11.7 Global Resources in Execution Environments

This subsection explains how to work with global resources in different execution environments. The subsection is organized into the following topics:

- Global Resources in Generated Code 1414
- Global Resources in MapForce Server 1415
- Global Resources in FlowForce Server 1415

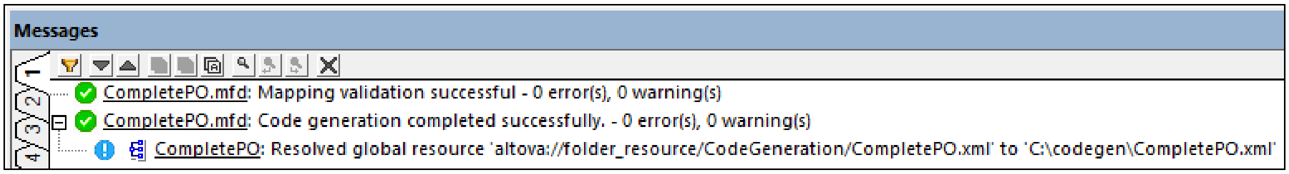
11.7.1 Global Resources in Generated Code

This topic explains how global resources are used in generated code. For more information, see the subsections below.

Global Resources in XSLT, XSLT 2, XQuery
When you generate XSLT or XSLT2 code and the mapping uses global resources, this does not affect the generated XSLT stylesheet in any way. With or without global resources, you can flexibly specify the input and output files when you run the XSLT stylesheet in your XSLT processor. The same applies to generated XQuery code.

An exception is the **DoTransform.bat** file generated for RaptorXML execution. Global resources used by the mapping will be resolved in actual paths in **DoTransform.bat**. The configuration which is currently selected from the global resources drop-down list will be taken into account. For information about supplying global resources to RaptorXML, see the RaptorXML documentation.

Global Resources in C++, C#, Java
When you generate C#, C++, or Java program code, global resources used by the mapping will be resolved. For example, a file or folder alias defined as a global resource will be converted to a file or folder path. If a particular global resource configuration is selected from the global resources drop-down list, the code will be generated for the selected configuration. The **Messages** window shows information about how exactly a global resource was resolved (*see screenshot below*).



To generate code for a particular global resource configuration, select it from the global resources drop-down list before generating code. Alternatively, if you generate code from the command line, supply the **GLOBALRESOURCEFILE** and **GLOBALRESOURCECONFIG** parameters at the command line. For details, see MapForce Command Line Interface 1391.

It is not possible to switch or refer to global resources from generated code. Instead, you can modify the code to change the input or output file path.

Note: In C# or Java, you can change the path and the data type of input or output 1528.

11.7.2 Global Resources in MapForce Server

When you compile a mapping to a MapForce Server execution file (**.mfx**), any global resource references used by the mapping are preserved, not resolved. This means that you will need to provide these references on the server side in order to run the mapping successfully. In MapForce Server, the following is required to run a **.mfx** file which uses global resources:

1. *The Global Resources Definitions file*. On the machine where MapForce is installed, the file is called **GlobalResources.xml**. You can find this file in the **Documents\Altova** folder. You can copy this file to the machine where MapForce Server runs and create multiple such files if necessary. See also Global Resources Setup Part 1 1398.
2. *The Global Resource configuration name*. Each Global Resource has a default configuration. You can also create additional configurations. For more information, see Global Resources Setup Part 2 1400.

In MapForce, the Global Resource Definitions file and the Global Resource configuration name are set or changed from the graphical user interface. In MapForce Server, these are specified at mapping runtime (*see below*).

- If you run the mapping through the command line interface, set the options **--globalresourceconfig** and **--globalresourcefile** after the run command, for example:

```
C:\Program Files (x86)\Altova\MapForceServer2023\bin\MapForceServer.exe run
SomeMapping.mfx --globalresourcefile="C:
\Users\me\Documents\Altova\GlobalResources.xml" --globalresourceconfig="Default"
```

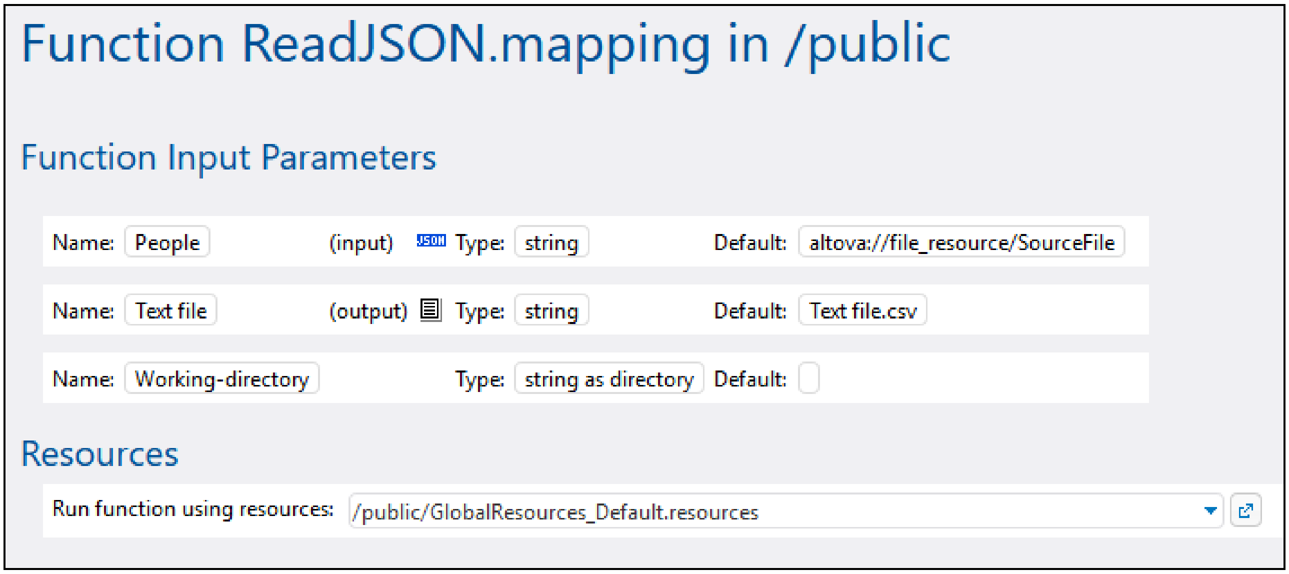
- If you run the mapping through the MapForce Server API, call the method **SetOptions** twice before calling the **Run** method. The first call is required to supply the Global Resource Definitions file path as an option, and the second call is required to supply the Global Resource configuration name.

For more information, see the MapForce Server documentation.

11.7.3 Global Resources in FlowForce Server

In FlowForce Server, global resources are not stored in one XML file as in desktop applications. In FlowForce, each resource is a reusable object that may contain file or folder paths or database connection details. Resources can be copied, exported, and imported, and are subject to the same user access mechanism as other FlowForce Server objects. This means that any FlowForce user can use any resource in their mapping functions if they have the required permissions.

Once you have created a mapping with global resources in MapForce, you can deploy it to FlowForce Server. At deployment time, if you want your mapping to use global resources, select the **Use Resources** check box in the deployment dialog box. If you do not select the check box, any global resources used by the mapping will be resolved, based on the currently selected configuration. If you have selected the check box, the mapping function will require resources in FlowForce Server as well. The screenshot below is an example of a mapping function deployed to FlowForce that requires resources to run. Notice that the first parameter gets the default file path from a resource.



In FlowForce Server, it is the mapping function that uses the global resources, not the job. The mapping function reads the path of the first input file from the resource. This means that all jobs using this function will use the same path unless you override the path from the job configuration page.

You can also deploy global resources to FlowForce Server as standalone objects. This means there is no need to deploy a mapping first in order to be able to deploy a global resource. For more information, see *Deploying Resources to FlowForce Server* below.

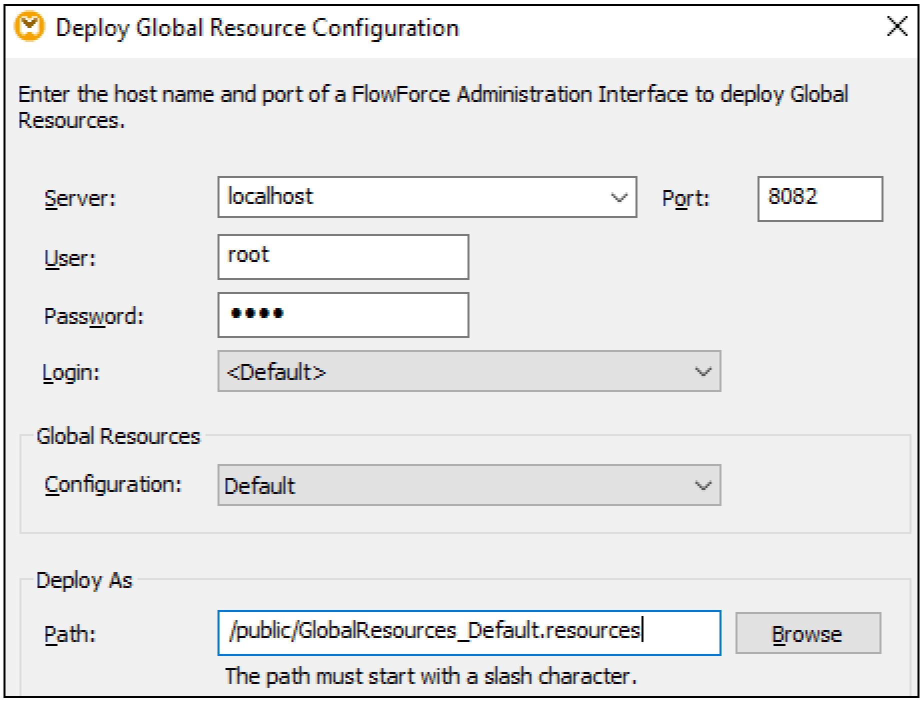
For information about consuming resources in FlowForce Server, see the FlowForce Server documentation.

Deploying resources to FlowForce Server

You can deploy global resources created with MapForce to FlowForce Server. Upon deployment, you must choose the configuration with which the resource should be deployed to the server. If you need all configurations of the same global resource on the server, you can deploy this global resource multiple times and select the desired configuration each time upon deployment. You can also change the name of each global resource on the server and choose the destination container on the server.

You can deploy global resources to FlowForce Server at the same time when you deploy the mapping or separately. To deploy global resources to FlowForce Server, take the steps below:

1. Run MapForce.
2. Click the **Manage Global Resources** toolbar button. Alternatively, go to the **Tools** menu and click **Global Resources**.
3. Click **Deploy To Server**. This opens the **Deploy Global Resource Configuration** dialog box (*see screenshot below*).



4. Enter the connection details to FlowForce Server (server, port, user, password, login method). These parameters are the same as the ones required when you deploy mappings to FlowForce Server[1381].
5. Select a configuration from the **Configuration** list. This list includes all configurations from the current Global Resources Definitions file[1398]. Note that only one global resource configuration can be deployed at a time. You can deploy the same resource multiple times, with a different name, if you need all the configurations on the server.
6. Select a target path where the resource should be saved on the server. Click **Browse** to select a target FlowForce container or create a new one if required.
7. Click **OK**.

You can see information about deploying the global resource to FlowForce Server in the **Messages** window.

Note: Global resources that run other Altova applications are *not* supported in a server environment. For more information, see Transformation Results as Global Resource[1410].

12 Catalogs in MapForce

MapForce supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables MapForce to retrieve commonly used schemas (as well as other files) from local user folders. This increases the overall processing speed, enables users to work offline (that is, not connected to a network), and improves the portability of documents (because URIs would then need to be changed only in the catalog files.)

The catalog mechanism in MapForce works as outlined in this section:

For more information on catalogs, see the XML Catalogs specification.

12.1 How Catalogs Work

Catalogs can be used to redirect both DTDs and XML Schemas. While the concept behind the mechanisms of both cases is the same, the details are different and are explained below.

DTDs

Catalogs are commonly used to redirect a call to a DTD to a local URI. This is achieved by mapping, in the catalog file, public or system identifiers to the required local URI. So when the **DOCTYPE** declaration in an XML file is read, its public or system identifier locates the required local resource via the catalog file mapping.

For popular schemas, the **PUBLIC** identifier is usually pre-defined, thus requiring only that the URI in the catalog file map the **PUBLIC** identifier to the correct local copy. When the XML document is parsed, the **PUBLIC** identifier in it is read. If this identifier is found in a catalog file, then the corresponding URL in the catalog file will be looked up and the schema will be read from this location. So, for example, if the following SVG file is opened in MapForce:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="20" height="20" xml:space="preserve">
  <g style="fill:red; stroke:#000000">
     <rect x="0" y="0" width="15" height="15"/>
     <rect x="5" y="5" width="15" height="15"/>
  </g>
</svg>
```

The catalog is searched for the **PUBLIC** identifier of this SVG file. Let's say the catalog file contains the following entry:

```
<catalog>
  ...
    <public publicId="-//W3C//DTD SVG 1.1//EN" uri="schemas/svg/svg11.dtd"/>
  ...
</catalog>
```

In this case, there is a match for the **PUBLIC** identifier. As a result, the lookup for the SVG DTD is redirected to the URL **schemas/svg/svg11.dtd** (which is relative to the catalog file). This is a local file that will be used as the DTD for the SVG file. If there is no mapping for the **Public** ID in the catalog, then the URL in the XML document will be used (in the SVG fie example above, this is the Internet URL: **http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd**).

XML Schemas

In MapForce, you can also use catalogs with **XML Schemas**. In the XML instance file, the reference to the schema will occur in the **xsi:schemaLocation** attribute of the XML document's top-level element. For example,

```
xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart OrgChart.xsd"
```

The value of the **xsi:schemaLocation** attribute has two parts: a namespace part (green above) and a URI part (highlighted). The namespace part is used in the catalog to map to the alternative resource. For example, the following catalog entry redirects the schema reference above to a schema at an alternative location.

```
<uri name="http://www.xmlspy.com/schemas/orgchart" uri="C:\MySchemas\OrgChart.xsd"/>
```

Normally, the URI part of the **xsi:schemaLocation** attribute's value is a path to the actual schema location. However, if the schema is referenced via a catalog, the URI part need not point to an actual XML Schema but must exist so that the lexical validity of the **xsi:schemaLocation** attribute is maintained. A value of `foo`, for example, would be sufficient for the URI part of the attribute's value to be valid.

12.2 Catalog Structure in MapForce

When MapForce starts, it loads a file called **`RootCatalog.xml`** (*structure shown in listing below*), which contains a list of catalog files that will be looked up. You can modify this file and enter as many catalog files to look up as you like, each of which is referenced in a **`nextCatalog`** element. These catalog files are looked up and the URIs in them are resolved according to their mappings.

Listing of RootCatalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
   xmlns:spy="http://www.altova.com/catalog_ext"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog Catalog.xsd">
  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/CustomCatalog.xml"/>
  <!-- Include all catalogs under common schemas folder on the first directory level -->
  <nextCatalog spy:recurseFrom="%CommonSchemasFolder%" catalog="catalog.xml"
spy:depth="1"/>
  <nextCatalog spy:recurseFrom="%ApplicationWritableDataFolder%/pkgs/.cache"
catalog="remapping.xml" spy:depth="0"/>
  <nextCatalog catalog="CoreCatalog.xml"/>
</catalog>
```

The listing above references a custom catalog (named **`CustomCatalog.xml`**) and a set of catalogs that locate commonly used schemas (such as W3C XML Schemas and the SVG schema).

- **`CustomCatalog.xml`** is located in your Personal Folder (located via the variable **`%PersonalFolder%`**). It is a skeleton file in which you can create your own mappings. You can add mappings to **`CustomCatalog.xml`** for any schema you require that is not addressed by the catalog files in the Common Schemas Folder. Do this by using the supported elements of the OASIS catalog mechanism (*see next section*).
- The Common Schemas Folder (located via the variable **`%CommonSchemasFolder%`**) contains a set of commonly used schemas. Inside each of these schema folders is a **`catalog.xml`** file that maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.
- **`CoreCatalog.xml`** is located in the MapForce application folder, and is used to locate schemas and stylesheets used by MapForce-specific processes, such as StyleVision Power Stylesheets which are stylesheets used to generate Altova's Authentic View of XML documents.

Location variables

The variables that are used in **`RootCatalog.xml`** (*listing above*) have the following values:

`%PersonalFolder%`	Personal folder of the current user, for example `C:\Users\<name>\Documents`
`%CommonSchemasFolder%`	`C:\ProgramData\Altova\Common2023\Schemas`
`%ApplicationWritableDataFolder%`	`C:\ProgramData\Altova`

Location of catalog files and schemas

Note the locations of the various catalog files.

- **`RootCatalog.xml`** and **`CoreCatalog.xml`** are in the MapForce application folder.
- **`CustomCatalog.xml`** is located in your **`MyDocuments\Altova\MapForce`** folder.
- The **`catalog.xml`** files are each in a specific schema folder, these schema folders being inside the Common Schemas Folder.

12.3 Customizing Your Catalogs

When creating entries in **`CustomCatalog.xml`** (or any other catalog file that is to be read by MapForce), use only the following elements of the OASIS catalog specification. Each of the elements below is listed with an explanation of their attribute values. For a more detailed explanation, see the XML Catalogs specification. Note that each element can take the **`xml:base`** attribute, which is used to specify the base URI of that element.

- `<public publicId="PublicID of Resource" uri="URL of local file"/>`
- `<system systemId="SystemID of Resource" uri="URL of local file"/>`
- `<uri name="filename" uri="URL of file identified by filename"/>`
- `<rewriteURI uriStartString="StartString of URI to rewrite" rewritePrefix="String to replace StartString"/>`
- `<rewriteSystem systemIdStartString="StartString of SystemID" rewritePrefix="Replacement string to locate resource locally"/>`

Note the following points:

- In cases where there is no public identifier, as with most stylesheets, the system identifier can be directly mapped to a URL via the **`system`** element.
- A URI can be mapped to another URI using the `uri` element.
- The `rewriteURI` and `rewriteSystem` elements enable the rewriting of the starting part of a URI or system identifier, respectively. This allows the start of a filepath to be replaced and consequently enables the targeting of another directory. For more information on these elements, see the XML Catalogs specification.

From release 2014 onwards, MapForce adheres closely to the XML Catalogs specification (OASIS Standard V1.1, 7 October 2005) specification. This specification strictly separates external-identifier look-ups (those with a Public ID or System ID) from URI look-ups (URIs that are not Public IDs or System IDs). Namespace URIs must therefore be considered simply URIs—not Public IDs or System IDs—and must be used as URI look-ups rather than external-identifier look-ups. In MapForce versions prior to version 2014, schema namespace URIs were translated through **`<public>`** mappings. From version 2014 onwards, **`<uri>`** mappings have to be used.

Prior to v2014: `<public publicID="http://www.MyMapping.com/ref" uri="file:///C:/MyDocs/Catalog/test.xsd"/>`
V-2014 onwards: `<uri name="http://www.MyMapping.com/ref" uri="file:///C:/MyDocs/Catalog/test.xsd"/>`

How MapForce finds a referenced schema

A schema is referenced in an XML document via the **`xsi:scemaLocation`** attribute (*shown below*). The value of the **`xsi:schemaLocation`** attribute has two parts: a namespace part (green) and a URI part (highlighted).

```
xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart OrgChart.xsd"
```

Given below are the steps, followed sequentially by MapForce, to find a referenced schema. The schema is loaded at the first successful step.

1. Look up the catalog for the URI part of the **`xsi:schemaLocation`** value. If a mapping is found, including in **`rewriteURI`** mappings, use the resulting URI for schema loading.
2. Look up the catalog for the namespace part of the **`xsi:schemaLocation`** value. If a mapping is found,

including in **`rewriteURI`** mappings, use the resulting URI for schema loading.
3. Use the URI part of the **`xsi:schemaLocation`** value for schema loading.

XML Schema specifications

XML Schema specification information is built into MapForce and the validity of XML Schema (`.xsd`) documents is checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema that defines the XML Schema specification.

The **`catalog.xml`** file in the `%AltovaCommonSchemasFolder%\Schemas\schema` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against these schemas. The referenced files are included solely to provide MapForce with entry helper info for editing purposes should you wish to create documents according to these older recommendations.

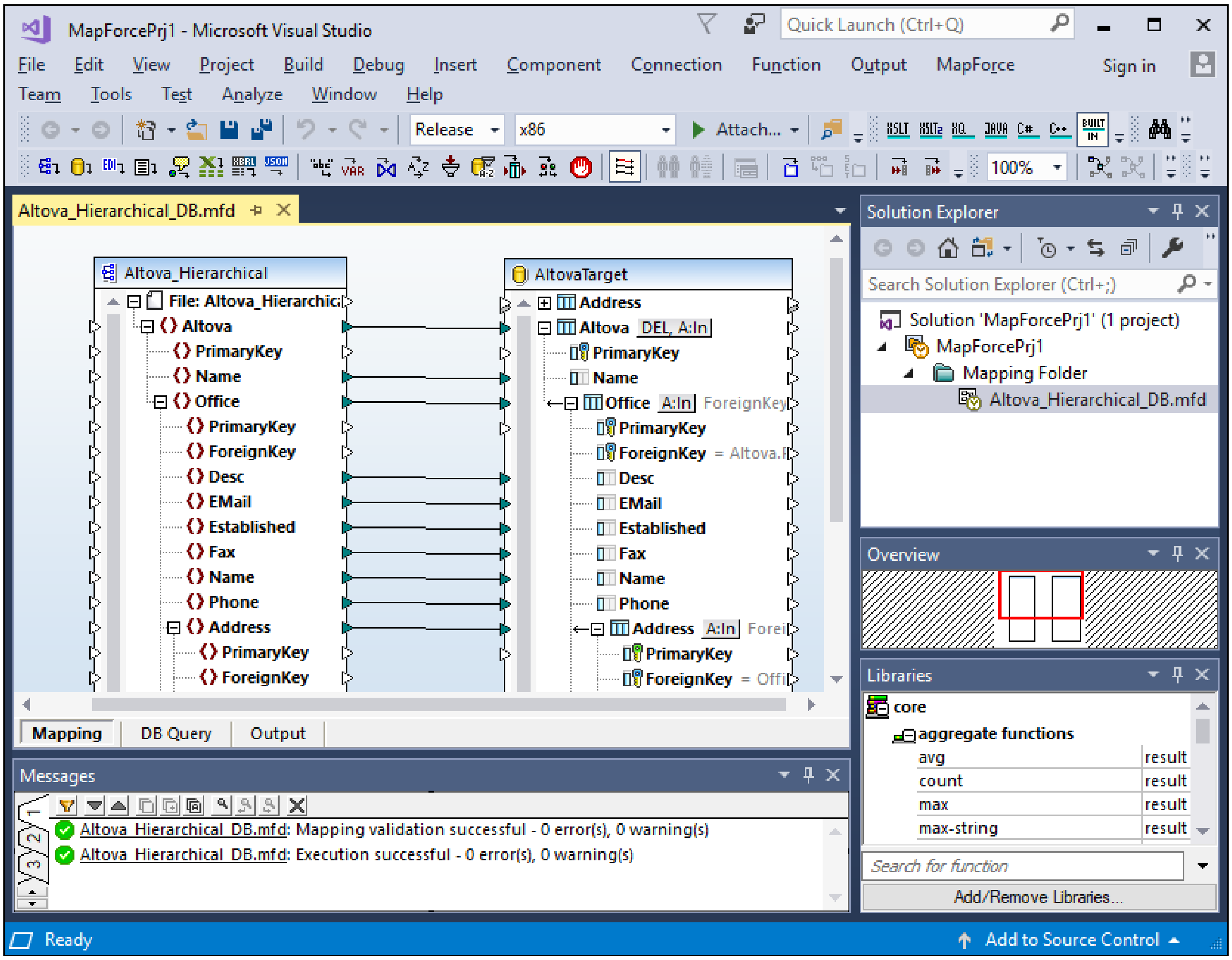
12.4 Environment Variables

Shell environment variables can be used in the `nextCatalog` element to specify the path to various system locations (*see RootCatalog.xml listing above*). The following shell environment variables are supported:

`%PersonalFolder%`	Full path to the Personal folder of the current user, for example `C:\Users\<name>\Documents`
`%CommonSchemasFolder%`	`C:\ProgramData\Altova\Common2023\Schemas`
`%ApplicationWritableDataFolder%`	`C:\ProgramData\Altova`
`%AltovaCommonFolder%`	C:\Program Files\Altova\Common2023
`%DesktopFolder%`	Full path to the Desktop folder of the current user.
`%ProgramMenuFolder%`	Full path to the Program Menu folder of the current user.
`%StartMenuFolder%`	Full path to Start Menu folder of the current user.
`%StartUpFolder%`	Full path to Start Up folder of the current user.
`%TemplateFolder%`	Full path to the Template folder of the current user.
`%AdminToolsFolder%`	Full path to the file system directory that stores administrative tools of the current user.
`%AppDataFolder%`	Full path to the Application Data folder of the current user.
`%CommonAppDataFolder%`	Full path to the file directory containing application data of all users.
`%FavoritesFolder%`	Full path of the Favorites folder of the current user.
`%PersonalFolder%`	Full path to the Personal folder of the current user.
`%SendToFolder%`	Full path to the SendTo folder of the current user.
`%FontsFolder%`	Full path to the System Fonts folder.
`%ProgramFilesFolder%`	Full path to the Program Files folder of the current user.
`%CommonFilesFolder%`	Full path to the Common Files folder of the current user.
`%WindowsFolder%`	Full path to the Windows folder of the current user.
`%SystemFolder%`	Full path to the System folder of the current user.
`%LocalAppDataFolder%`	Full path to the file system directory that serves as the data repository for local (nonroaming) applications.
`%MyPicturesFolder%`	Full path to the MyPictures folder.

13 MapForce Plug-in for Visual Studio

You can integrate MapForce 2023 into the Microsoft Visual Studio versions 2012/2013/2015/2017/2019/2022. This integration helps combine the mapping capabilities of MapForce with the development environment of Visual Studio. When the MapForce plug-in is enabled, you can create mappings and mapping projects directly in Visual Studio (*see screenshot below*).



Installation

To install the MapForce Plug-in for Visual Studio, take the steps below:

1. Install Microsoft Visual Studio 2012/2013/2015/2017/2019/2022. Note that from Visual Studio 2022 onwards, Visual Studio is being made available only as a 64-bit application.
2. Install MapForce (Enterprise or Professional Edition). If you have installed Visual Studio 2022+, then you must install the 64-bit version of MapForce.
3. Download and run the MapForce integration package for Microsoft Visual Studio. This package is available on the MapForce (Enterprise and Professional Editions) download page at www.altova.com.

Once the integration package has been installed, you will be able to use MapForce in the Visual Studio environment.

> **Important**
>
> You must use the integration package corresponding to your MapForce version (current version is 2023). The integration package is not edition-specific and can therefore be used for both Enterprise and Professional editions.

Information about menus and functions

When the MapForce plug-in for Visual Studio is enabled, you can access different MapForce menus and functions (*see below*). You can customize MapForce menus and toolbars from the **Tools | Customize** menu of Visual Studio.

Note: In Visual Studio 2019 and later, MapForce functionality can be accessed in the **Extensions** menu of Visual Studio. In earlier versions of Visual Studio, MapForce features are available in the top-level menus of Visual Studio.

⊟ Themes

You can select MapForce themes in the **MapForce** menu of Visual Studio. The options are Classic, Light, and Dark themes.

⊟ Create and open files/projects

When the MapForce plug-in for Visual Studio is enabled, you can create, open, and work with mappings and mapping projects directly in Visual Studio. To create a new mapping design file in Visual Studio, use the **File | New** menu command. To create a new project, use the **File | New Project** menu command. To open existing mapping files or projects, you can use the following Visual Studio menus: **File | Open | File** or **File | Open | Project/Solution**. Then you can look for the MapForce-related file types.

⊟ Global Resources

MapForce Global Resources[1397] are available in the **MapForce | Manage Global Resources** menu of Visual Studio. In Visual Studio 2019 onwards, the corresponding menu is **Extensions | MapForce | Manage Global Resources**.

⊟ Debugging

After you have opened a mapping file, the mapping debugging commands become available in the Debug menu[1466] and in the **Debug** toolbar. In Visual Studio 2019 onwards, the corresponding menu is **Extensions | MapForce | Debug**.

⊟ MapForce options

MapForce options are available in the **Tools | MapForce Options** menu of Visual Studio.

⊟ Mapping pane customization

When you open a mapping in the main pane of Visual Studio, the **View | MapForce** menu becomes available. It includes the same options as the **View** menu of the standalone version of MapForce.

Libraries window

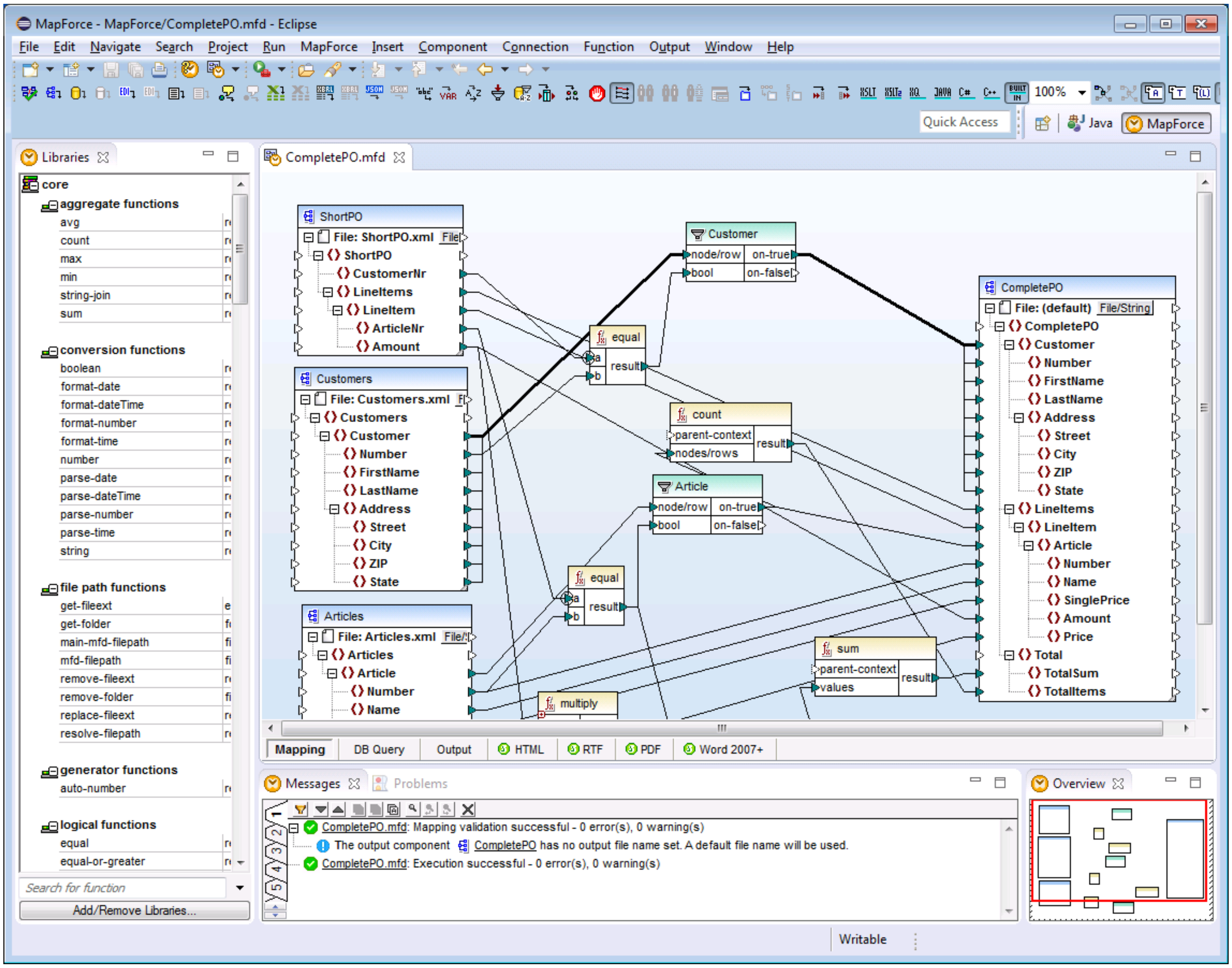
If the MapForce **Libraries** window is not visible in Visual Studio, you can enable it from the **View | MapForce | Libraries Window** menu (this menu becomes available in Visual Studio when a mapping file is open). Once the **Libraries** window is enabled, you can dock it to a particular position in the interface.

Help and Support

The MapForce menus **Help**, **Support Center**, **Check for Updates** and **About** are available in the **Help | MapForce Help** menu of Visual Studio.

14 MapForce Plug-in for Eclipse

Eclipse is an open source framework that integrates different types of applications delivered in form of plug-ins. You can integrate MapForce Enterprise and Professional Edition into Eclipse versions 2022-09, 2022-06, 2022-03, 2021-12 and access MapForce functionality directly from Eclipse.



MapForce Enterprise Edition plug-in for Eclipse

The following topics provide help on installing and using the MapForce plug-in for Eclipse.

- [Installing the MapForce Plug-in for Eclipse] 1430
- [The MapForce Perspective] 1432
- [Accessing Common Menus and Functions] 1435
- [Working with Mapping and Projects] 1439
- [Extending MapForce Plug-in for Eclipse] 1449

14.1 Installing the MapForce Plug-in for Eclipse

Prerequisites

- Eclipse 2022-09, 2022-06, 2022-03, 2021-12 (http://www.eclipse.org), 64-bit.
- A Java Runtime Environment (JRE) or Java Development Kit (JDK) for the 64-bit platform.
- MapForce Enterprise or Professional Edition 64-bit.

Note: All the prerequisites listed above must have the 64-bit platform. Integration with older Eclipse 32-bit platforms is no longer supported, although it may still work.

After the prerequisites listed above are in place, you can install the MapForce Integration Package (64-bit) to integrate MapForce in Eclipse. The integration can be carried out either during the installation of the Integration Package or manually from Eclipse after the Integration Package has been installed. The MapForce Integration Package is available for download at https://www.altova.com/components/download.

Note: Eclipse must be closed while you install or uninstall the MapForce Integration Package.

Integrate MapForce during installation of the Integration Package

You can integrate MapForce in Eclipse during the installation of the MapForce Integration Package. Do this as follows:

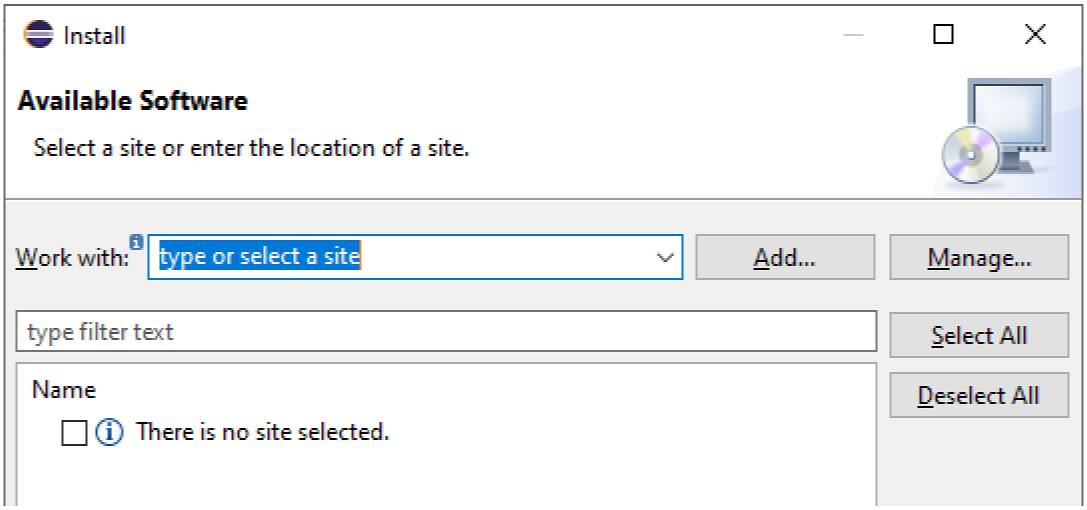
1. Run the MapForce Integration Package to start the installation wizard.
2. Go through the initial steps of the installation with eth wizard.
3. In the Integration step, select *Let this wizard integrate Altova MapForce plug-in into Eclipse*, and browse for the directory where the Eclipse executable (`eclipse.exe`) is located.
4. Click **Next** and complete the installation.

The MapForce perspective and menus will be available in Eclipse the next time you start it.

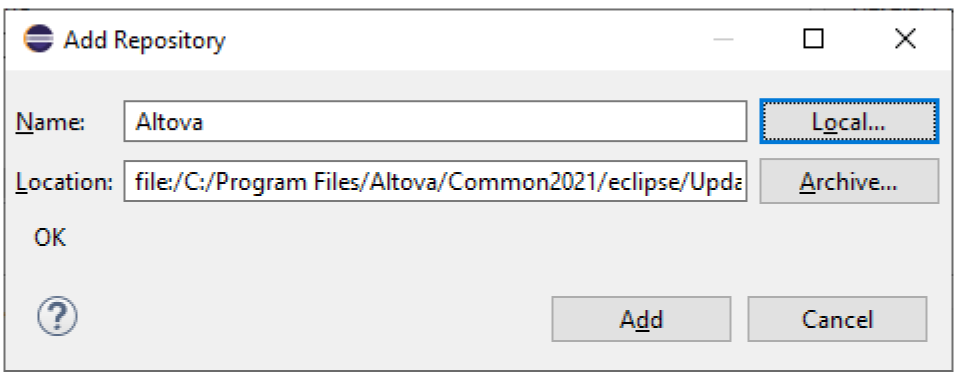
Integrate MapForce in Eclipse manually

After you have installed the MapForce Integration Package, you can manually integrate MapForce in Eclipse as follows:

1. In Eclipse, select the menu command **Help | Install New Software**.
2. In the Install dialog box, click **Add**.



3. In the Add Repository dialog box, click **Local**. Browse for the folder `C:\Program Files\Altova\Common2023\eclipse\UpdateSite`, and select it. Provide a name for the site (such as "Altova").



4. Repeat the steps 2-3 above, this time selecting the folder `C:\Program Files\Altova\<%APPNAMESHORT%>\eclipse\UpdateSite` and providing a name such as "Altova MapForce".
5. On the Install dialog box, select *Only Local Sites*. Next, select the "Altova category" folder and click **Next**.
6. Review the items to be installed and click **Next** to proceed.
7. To accept the license agreement, select the respective check box.
8. Click **Finish** to complete the installation.

Note: If there are problems with the plug-in (missing icons, for example), start Eclipse from the command line with the `-clean` flag.

14.2 The MapForce Perspective

In Eclipse, a perspective is a GUI view that is configured with the functionality of a specific application. After MapForce has been integrated in Eclipse, a new perspective, named MapForce, becomes available in Eclipse. This perspective is a GUI that resembles the MapForce GUI and includes a number of its components.

When a file having a filetype associated with MapForce is opened (`.mfd`), this file can be edited in the MapForce perspective. Similarly, a file of another filetype can be opened in another perspective in Eclipse. Additionally, for any active file, you can switch the perspective (*see below*), thus allowing you to edit or process that file in another environment.

There are therefore two main advantage of perspectives:

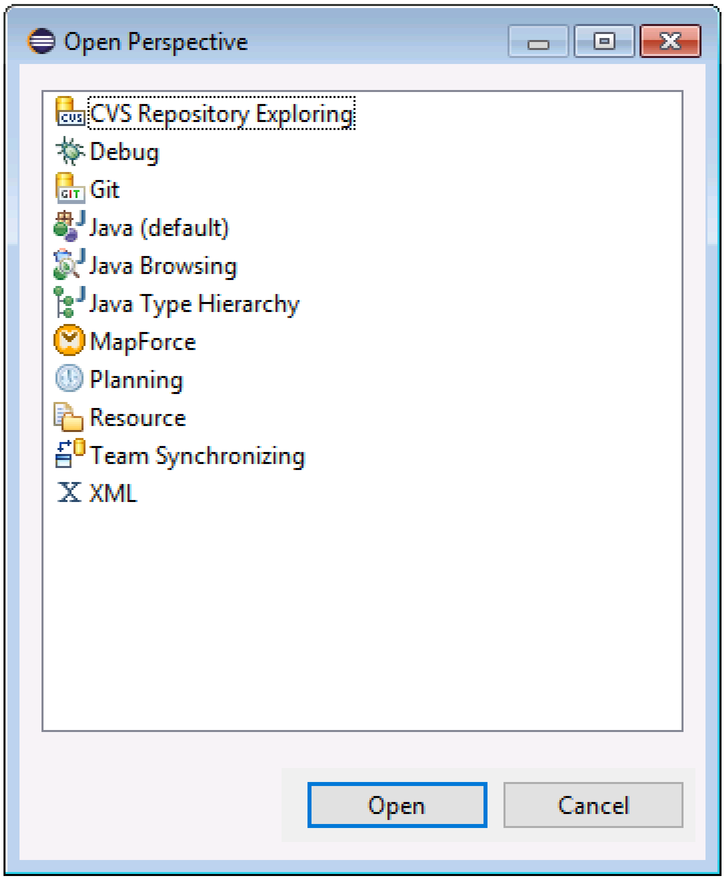
1. Being able to quickly change the working environment of the active file, and
2. Being able to switch between files without having to open a new development environment (the associated environment is available in a perspective)

Working with the MapForce perspective involves the following key procedures, which are described further below:

- Switching to the MapForce perspective.
- Setting preferences for the MapForce perspective.
- Customizing the MapForce perspective.

Switch to the MapForce perspective

In Eclipse, select the command **Window | Perspective | Open Perspective | Other**. In the dialog that appears (*screenshot below*), select **MapForce**, and click **Open**.



The empty window or the active document will now have the MapForce perspective. This is how the user switches the perspective via the menu. To access a perspective faster from another perspective, you can set the required perspective to be listed in the **Open Perspective** submenu, above the **Other** item. This setting is in the customization dialog (*see further below*).

Perspectives can also be switched when a file is opened or made active. The perspective of the application associated with a file's filetype will be automatically opened when that file is opened for the first time. Before the perspective is switched, a dialog appears asking whether you wish to have the default perspective automatically associated with this filetype. Check the *Do Not Ask Again* option if you wish to associate the perspective with the filetype without having to be prompted each time a file of this filetype is opened and then click **OK**.

Preferences for the MapForce perspective

To access the Preferences of a perspective, select the command **Window | Preferences**. In the list of perspectives in the left pane, select MapForce, then select the required preferences. Finish by clicking **OK**.

The preferences of a perspective include:

- To automatically switch to the MapForce perspective when a file of an associated filetype is opened (*see above*)
- Options for including or excluding individual MapForce toolbars
- Access to MapForce options.

Customize the MapForce perspective

The customization options enable you to determine what shortcuts and commands are included in the perspective. To access the Customize Perspective dialog of a perspective, make that perspective the active perspective and select the command **Window | Perspective | Customize Perspective**.

- In the *Toolbar Visibility* and *Menu Visibility* tabs, you can specify which toolbars and menus are to be displayed.
- In the *Action Set Availability* tab, you can add action sets to their parent menus and to the toolbar. If you wish to enable an action group, check its check box.
- In the *Shortcuts* tab of the Customize Perspective dialog, you can set shortcuts for submenus. Select the required submenu in the Submenus combo box. Then select a shortcut category, and check the shortcuts you wish to include for the perspective.

Click **Apply and Close** to complete the customization and for the changes to take effect.

Overview of the MapForce perspective

By default, the MapForce perspective in Eclipse is organized as follows:

- The mapping design window is available as an Eclipse editor. It has the same tabs and functionality as in the standalone edition of MapForce.
- The Libraries window is available as an Eclipse view, to the left of the main mapping editor. If this view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Libraries**. The Libraries view enables you to work with predefined or custom-defined functions and function libraries.
- The Messages pane is available as an Eclipse view, under the main mapping editor. If the Message view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Messages**. The messages view displays validation messages, errors, and warnings.
- The Overview pane is available as an Eclipse view. If the Overview view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Overview**. This view enables you to quickly navigate to a particular region on the mapping design area when it is very big.

14.3 Accessing Common Menus and Functions

In Eclipse, you can access most MapForce functionality from the same menus as in the standalone version, except for some Eclipse-specific variations which are listed below. This is the default setup; however, you can further customize the interface preferences from Eclipse, if desired (see The MapForce Perspective 1432).

Note: In Eclipse, some MapForce menu groups or commands are disabled (or not available) if the context is not relevant. For example, the **Insert** menu becomes available only when a mapping design file (.mfd) is active in Eclipse.

For information about the MapForce standard menus, see Menu Reference 1450.

General MapForce commands

In the standalone edition of MapForce, the commands applicable to mapping design files (such as **Validate**, **Deploy to FlowForce Server**, **Generate Code**, and others) are available in the **File** menu. In Eclipse, these commands are available in the **MapForce** menu, or in the MapForce toolbar. Note that the commands for opening or saving files (including MapForce project files) are available in the **File** menu of Eclipse.

MapForce themes can be selected in the **MapForce** menu.

The MapForce toolbar in Eclipse

The  toolbar button opens the MapForce help file.

The  toolbar button displays commands specific to MapForce files. When you expand this button, the available commands depend on the kind of file currently active in the Eclipse editor. For example, the commands specific to mapping design (.mfd) files are available when such a file is active (in focus) in the Eclipse editor.

Mapping debugging

The mapping debugging commands are available in the **MapForce | Debug** menu and in the **Debug** toolbar. Note that the mapping debugging shortcut keys (such as F5, F10, etc) are already reserved by Eclipse and are not available.

Global Resources

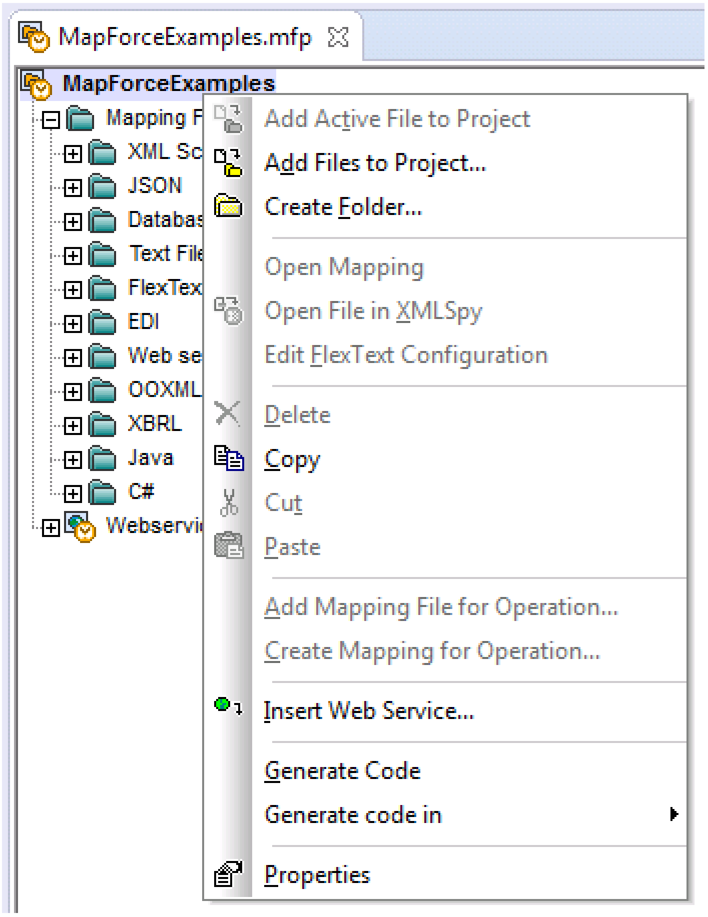
To access or manage Global Resources, do one of the following:

- Click to expand the MapForce  toolbar button, and then click **Global Resources**.
- On the MapForce menu, click **Global Resources**.

MapForce Projects

In the standard edition of MapForce, the **Project** menu contains various commands applicable to mapping project (.mfp) files. In Eclipse, these commands exist as follows:

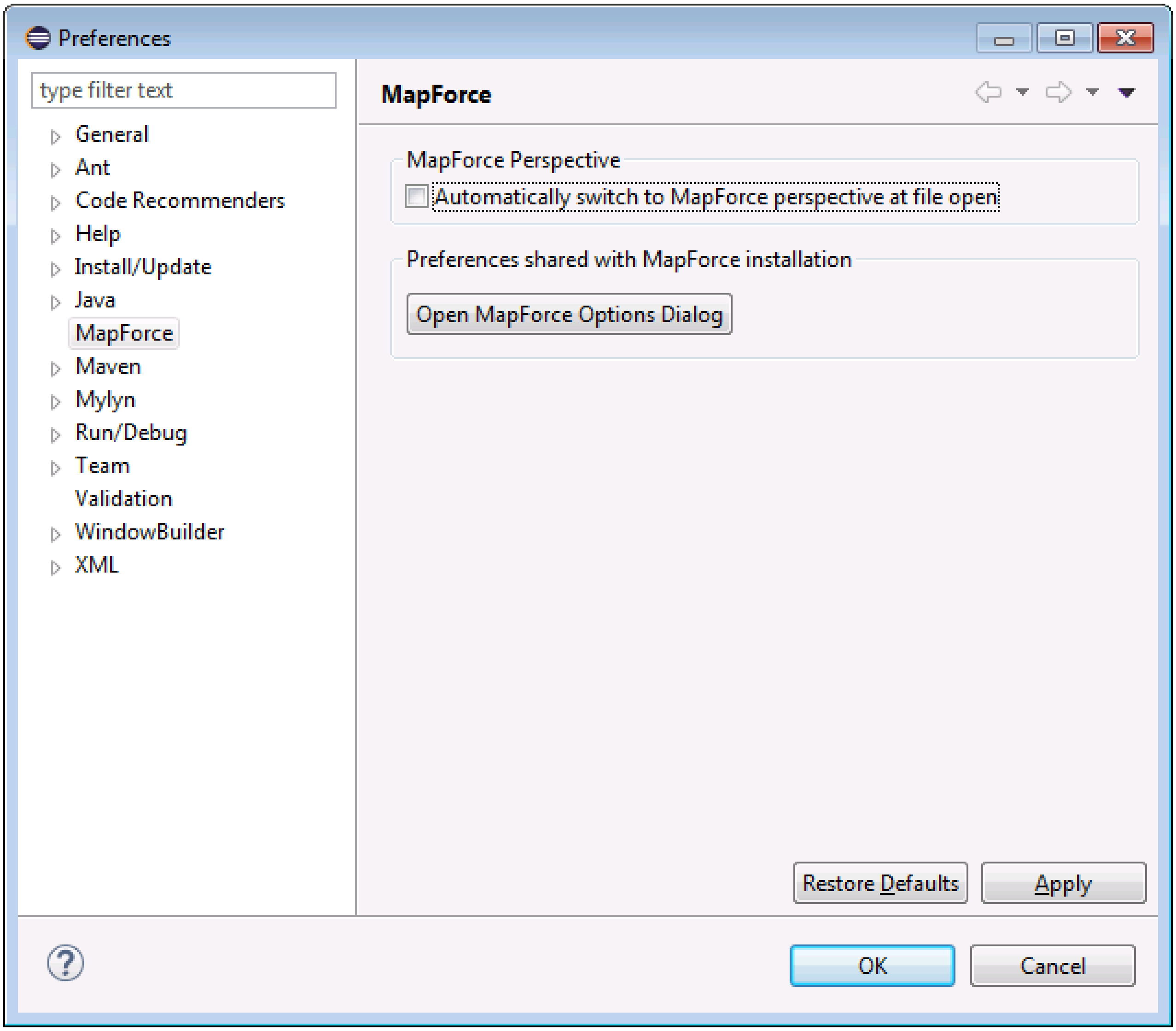
- The commands to open or save a project are available from the Eclipse **File** menu.
- Other project-specific commands are available as context commands. To display the context commands, create or open a MapForce project (.mfp) file in Eclipse, and then right-click the project.



Note that, in addition to standard MapForce projects (.mfp), in Eclipse you can also create projects of type "MapForce/Eclipse". Such projects have a dual nature, and can be configured for automatic build and generation of MapForce code. See Working with Mappings and Projects 1439.

MapForce Options

MapForce options are available from the **Window | Preferences** menu. On the Preferences dialog box, select **MapForce**, and then click **Open MapForce Options Dialog**.



Preferences dialog box

Libraries window

In Eclipse, the MapForce Libraries window is available as a view. This view is by default located to the left of the main editor window. (All MapForce-related views become visible in the Eclipse interface when the MapForce perspective is switched on, see also The MapForce Perspective 1432).

MapForce plug-in version

To see the currently installed version of the MapForce Plug-in for Eclipse, select the Eclipse menu option **Help | About Eclipse**. Then select the MapForce icon.

Help and Support

MapForce Help, Support Center, Check for Updates and About menus are available in the **Help | MapForce Help** menu of Eclipse.

14.4 Working with Mappings and Projects

When MapForce plug-in for Eclipse is installed, you can create from Eclipse the same mappings and mapping project types as in the standalone edition of MapForce, from within an Eclipse project. To design, test, compile, and deploy mappings, and to generate mapping code, you can either create a new Eclipse project or use an existing Eclipse project (for example, a Java project to which you want to add MapForce mappings).

In addition to this, you can work with all your mappings within a special project type that becomes available in Eclipse after you install the MapForce plug-in—the **MapForce/Eclipse Project**. Unless you choose to customize it, a MapForce/Eclipse project is by default assigned both a Java Builder and a MapForce Code Generation builder. Additionally, it has two Eclipse natures: MapForce nature and the JDT (Java Development tools) nature. As a result, a MapForce/Eclipse project behaves as follows when you save or change any of its resources (such as a mapping design file):

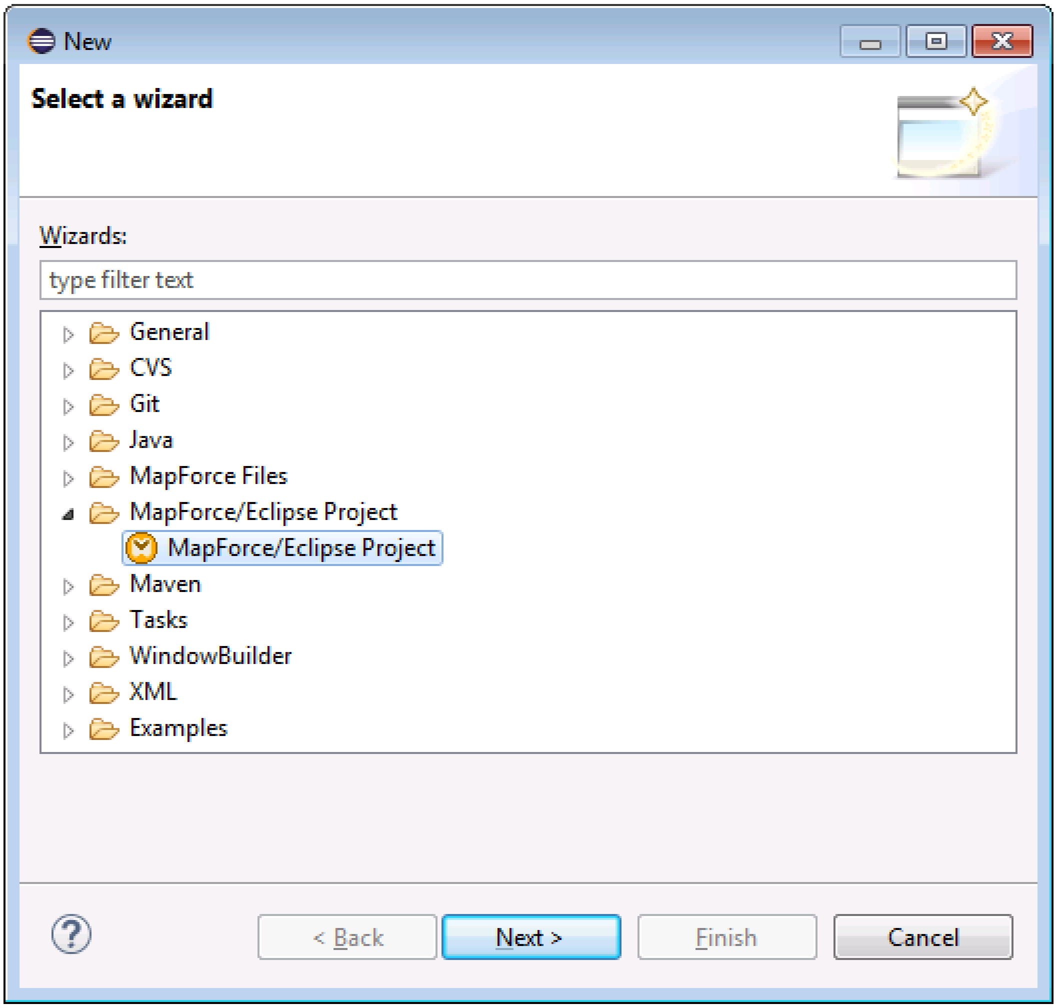
- If the **Project > Build automatically** menu option is enabled, the mapping code is generated automatically. When one or more MapForce project files exist in the MapForce/Eclipse project, the code generation language and output target folders are determined by the settings in each project file. Otherwise, Eclipse prompts you to choose a location.
- Any errors and output messages are shown in the Messages and Problems views.

This section contains the following topics:

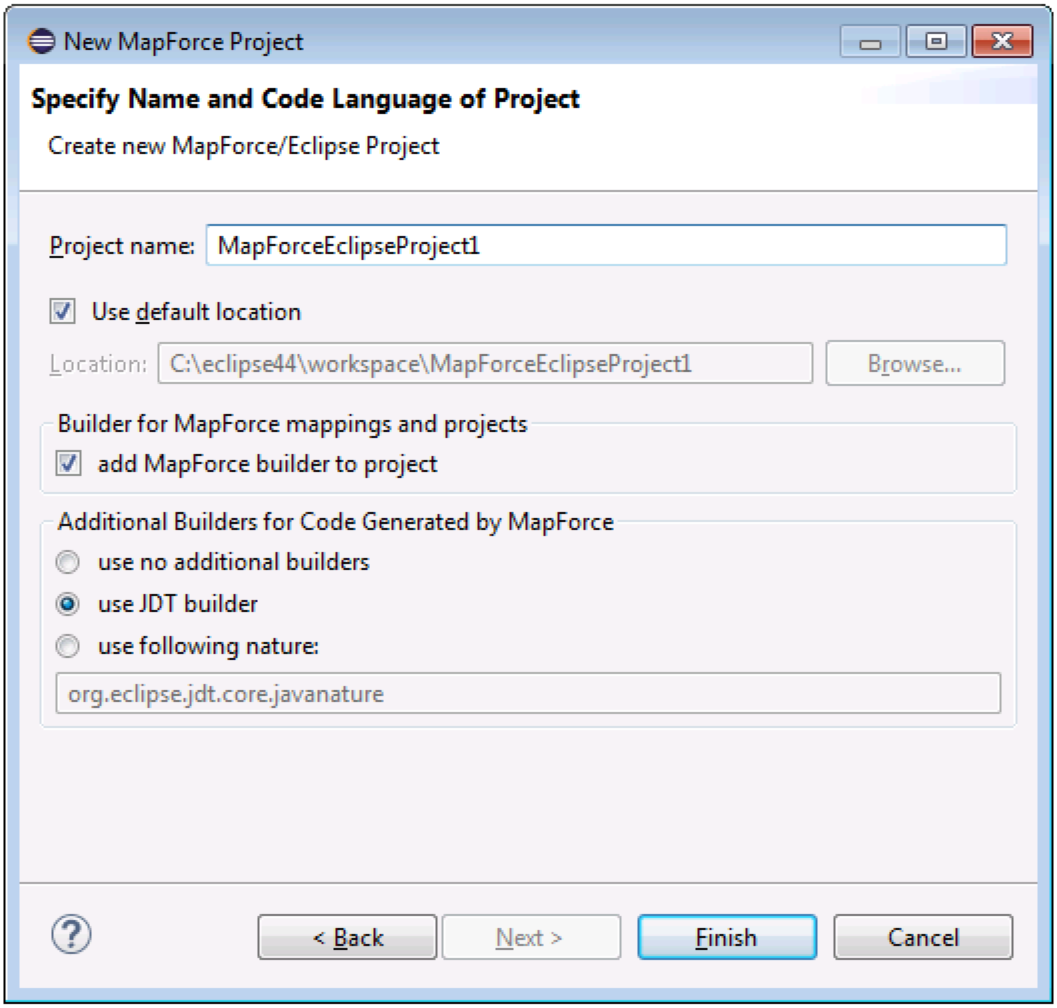
14.4.1 Creating a MapForce/Eclipse Project

To create a MapForce/Eclipse project, take the steps below:

1. On the **File** menu, click **New | Other**.
2. Select the **MapForce/Eclipse Project** category.


New
Select a wizard
Wizards:
type filter text
General
CVS
Git
Java
MapForce Files
MapForce/Eclipse Project
MapForce/Eclipse Project
Maven
Tasks
WindowBuilder
XML
Examples
< Back
Next >
Finish
Cancel

3. Click **Next**.



4. Enter a project name and choose a location where to save the project. Leave the **add MapForce builder to project** and **use JDT builder** options as is.
5. Click **Finish**.

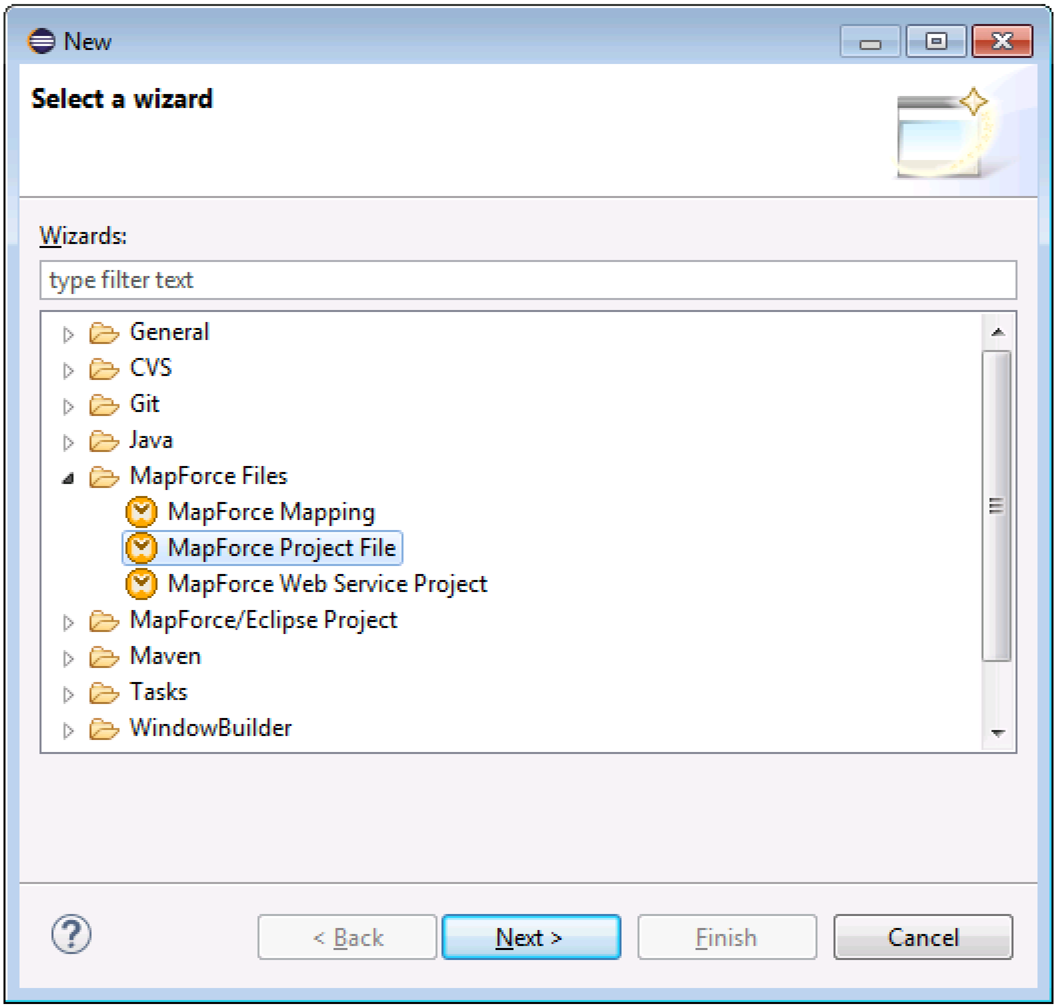
14.4.2 Creating New Mappings

You can create the following MapForce file types within an Eclipse project:

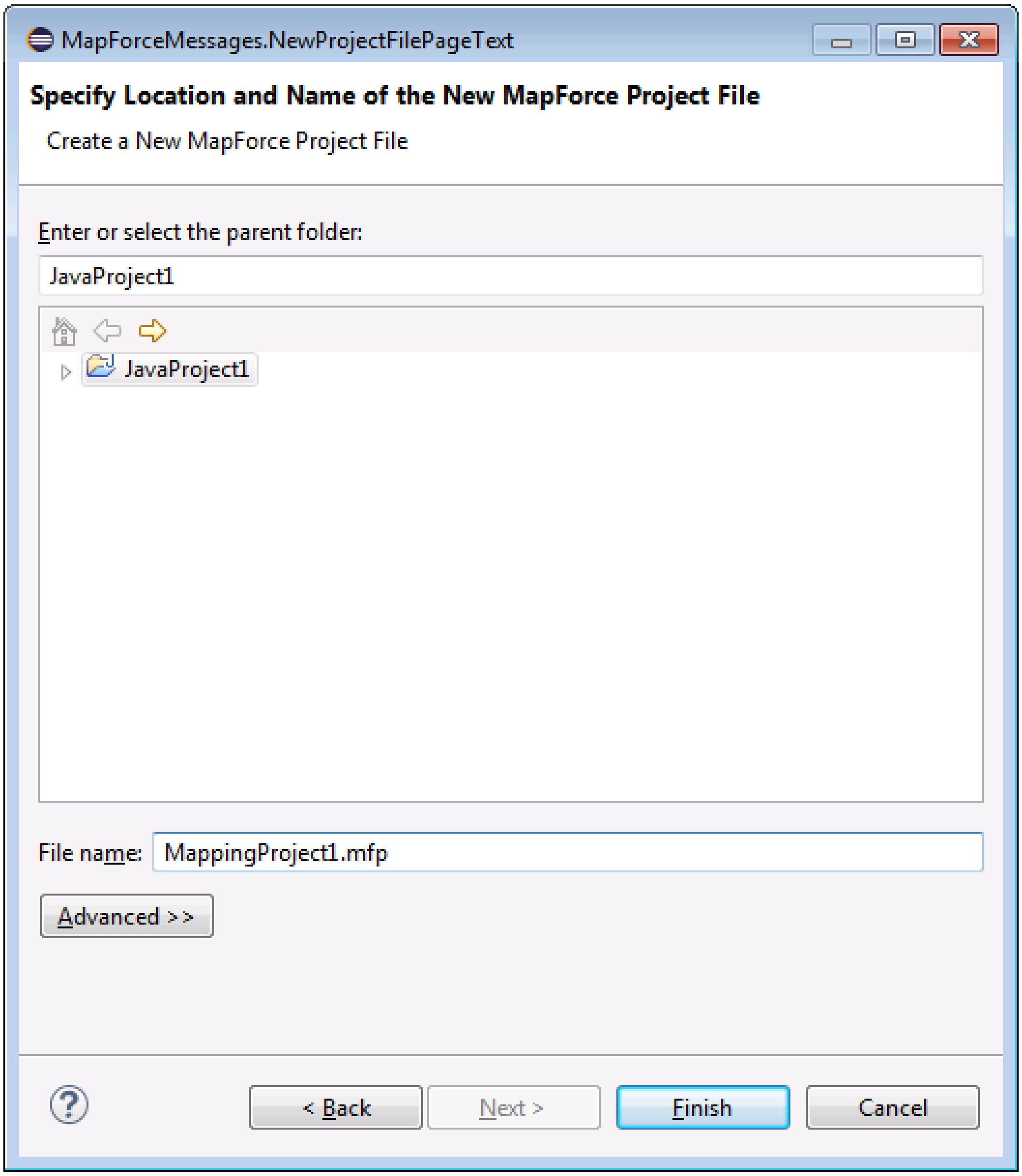
- MapForce mappings
- MapForce project files
- MapForce Web Service projects (available in MapForce Enterprise Edition)

To create any of these file types within an Eclipse project:

1. Create a new Eclipse project or open an existing one.
2. On the **File** menu, click **New**, and then click **Other**.



3. Select the required file type from the wizard dialog box, and then click **Next**.

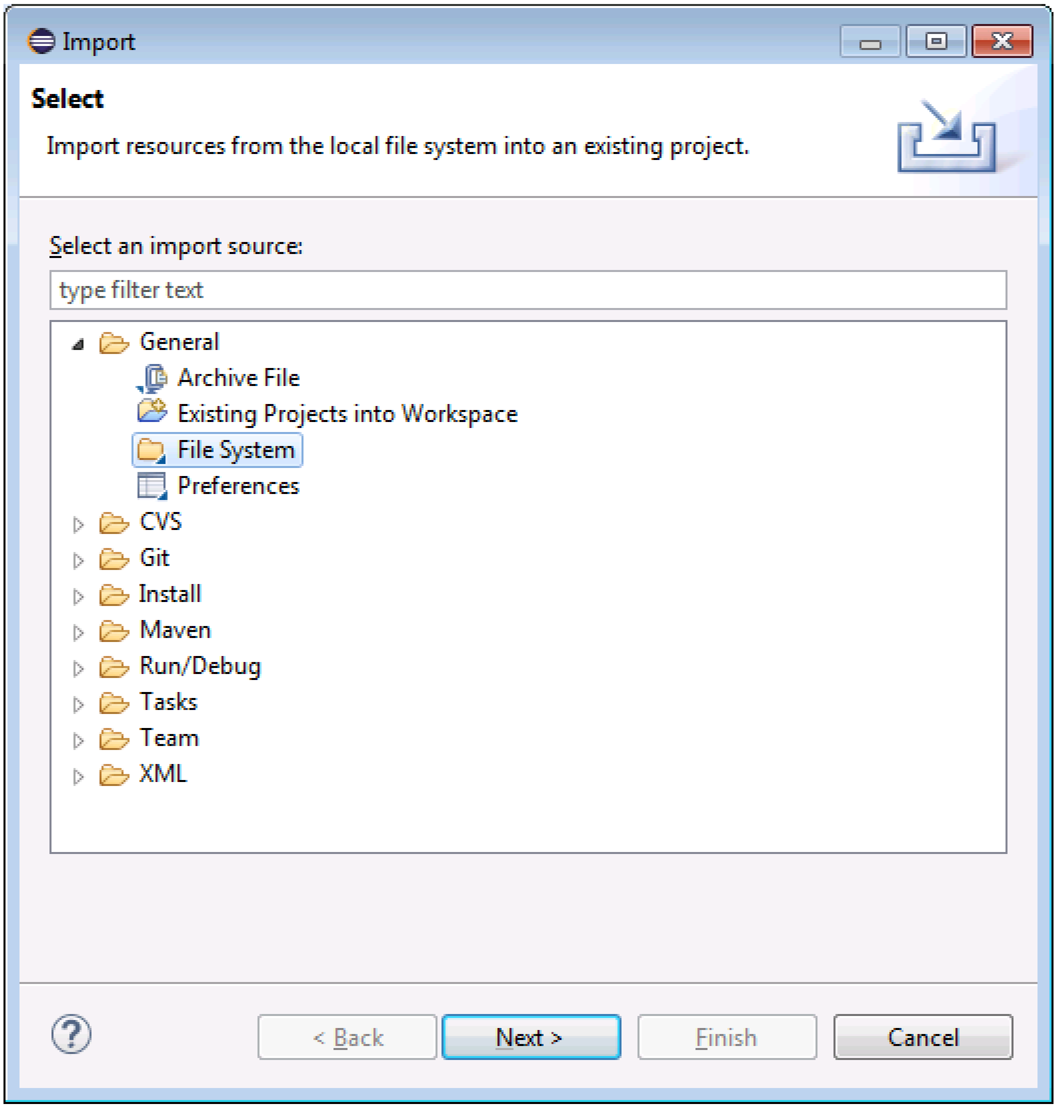


4. Select a parent folder in your existing project, and then click **Finish**.

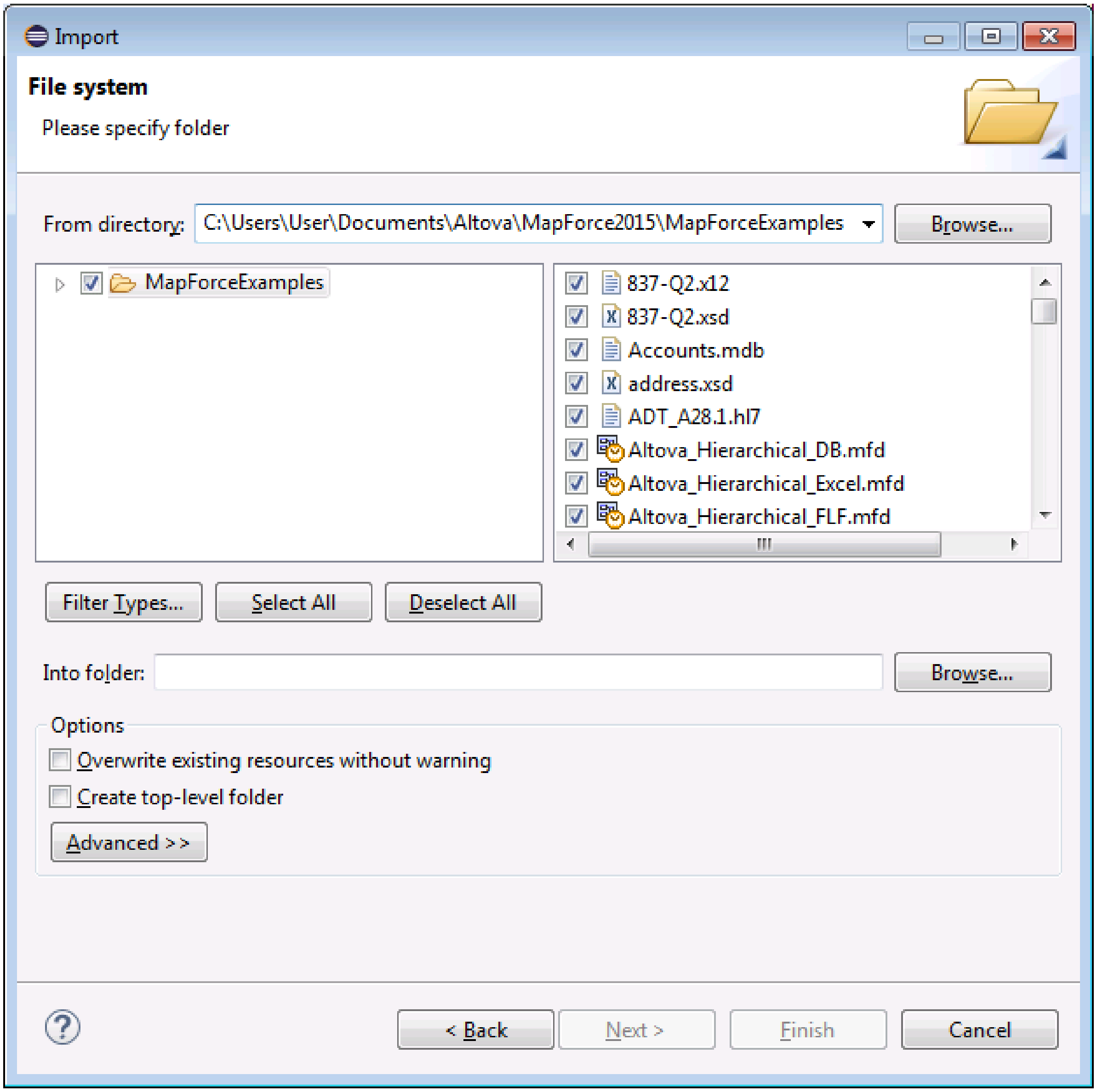
14.4.3 Importing Existing Mappings into an Eclipse Project

To import MapForce mappings and their dependent files into an existing Eclipse project:

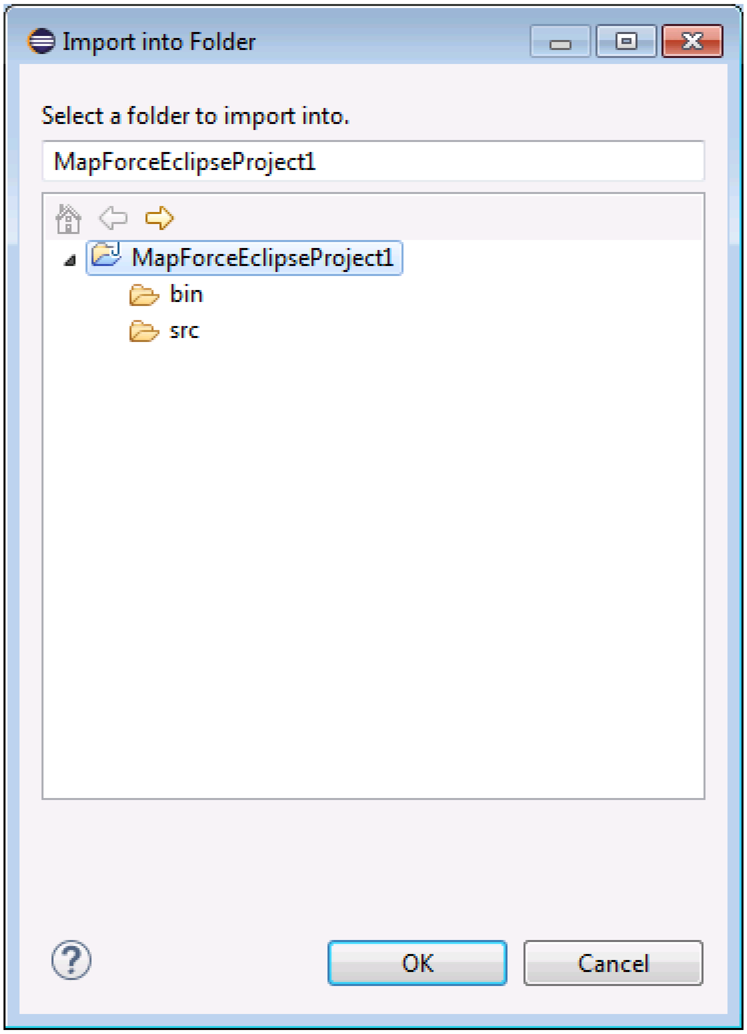
1. Open the project into which you want to import the files.
2. On the **File** menu, click **Import**.



3. Select **File System**, and then click **Next**.



4. Next to **From directory**, browse for the location of the files you want to import, and then select the required files.
5. Next to **Into folder**, click **Browse**, and select the project into which you are adding the files (in this example, *MapForceEclipseProject1*).



6. Click **OK**, and then click **Finish**.

14.4.4 Configuring Automatic Build and Generation of MapForce Code

Automatic MapForce code building and generation is enabled by default in any MapForce/Eclipse project (see Creating a MapForce/Eclipse Project 1439). If you want to enable automatic build and generation of MapForce code in an existing project which is not of type *MapForce/Eclipse*, you can do this by manually adding to it the *MapForce Code Generation* builder and the *MapForce* nature.

To add the MapForce Code Generation builder to a project:

- Add to the Eclipse **.project** file the lines highlighted below:

```
<buildSpec>
    <buildCommand>
```



```
          <name>org.eclipse.jdt.core.javabuilder</name>
          <arguments>
          </arguments>
       </buildCommand>
       <buildCommand>
          <name>com.altova.mapforceeclipseplugin.MapForceBuilder</name>
          <arguments>
          </arguments>
       </buildCommand>
    </buildSpec>
```

To add the MapForce nature to a project:

- Add to the Eclipse **.project** file the lines highlighted below:

```
    <natures>
       <nature>org.eclipse.jdt.core.javanature</nature>
       <nature>com.altova.mapforceeclipseplugin.MapForceNature</nature>
    </natures>
```

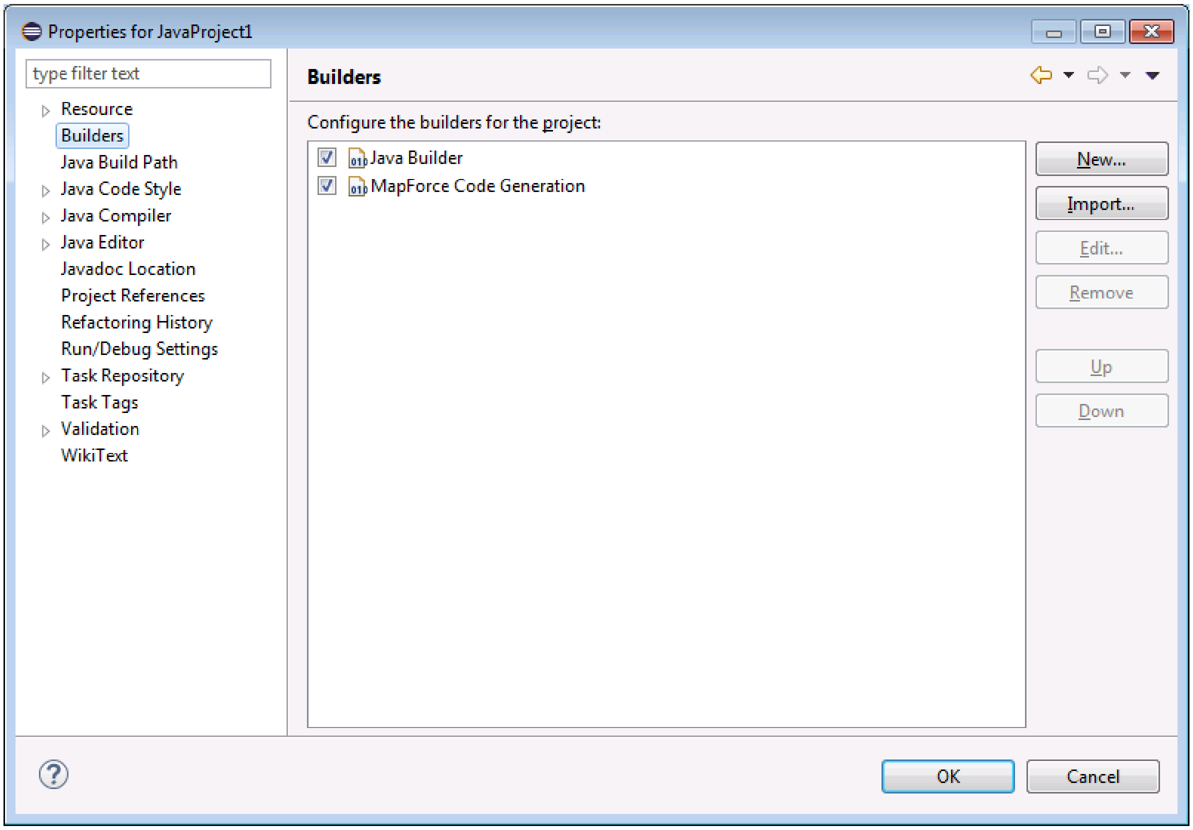
Tip: You can quickly open the **.project** file from the Navigator view of Eclipse (To enable this view, select the menu command **Window | Show View | Navigator**).

To switch automatic MapForce code generation on/off:

- On the **Project** menu, click **Build automatically**.

To disable the MapForce Code Generation builder:

1. On the **Project** menu, click **Properties**.
2. Click **Builders**.



3. Click to clear the **MapForce Code Generation** check box.

14.5 Extending MapForce Plug-in for Eclipse

The MapForce plug-in for Eclipse provides an Eclipse extension point with the ID **com.altova.mapforceeclipseplugin.MapForceAPI**. You can use this extension point to adapt, or extend the functionality of the MapForce plug-in. The extension point gives you access to the COM-Interface of the MapForce control and the MapForce API1645.

The MapForce Eclipse installation package contains a simple example of a plug-in that uses this extension point. It checks for any file open events of any new MapForce mappings, and sets the zoom level of the mapping view to 70%.

The JavaDoc documentation of the extension point is available in the MapForce plug-in installation directory (**C:\Program Files\Altova\MapForce2023\eclipse\docs**).

Before you install and run the sample MapForce plug-in, ensure that the following prerequisites are met:

- You are using 64-bit Java, 64-bit Eclipse, 64-bit MapForce and 64-bit MapForce Integration Package.
- The JDT (Java Development Tools) plug-in is installed.
- The Eclipse PDE (plug-in development environment) is installed.

To import the sample MapForce plug-in project into your workspace:

1. Start Eclipse.
2. On the **File** menu, click **Import**.
3. Select **General | Existing projects into Workspace**, and click **Next**.
4. Click the Browse... button next to the "'Select root directory" field and choose the sample project directory e.g. **C:\Program Files\Altova\MapForce2023\eclipse\workspace\MapForceExtension**.
5. Select the **Copy projects into workspace** option, and then click **Finish**. A new project named "MapForceExtension" has been created in your workspace.

To run the sample extension plug-in:

1. Switch to the Java perspective.
2. In the **Run** menu, click **Run Configurations**.
3. Right click **Eclipse Application** and select **New**. (If you cannot see "Eclipse application" in the list, the Eclipse Plug-In Development Tools are not installed in your Eclipse environment. To install Eclipse Plug-in Development Tools, click **Install New Software** in the **Help** menu. and install "Eclipse Plugin Development Tools" from "The Eclipse Project Updates" download site.)
4. Enter a name for your new configuration (in this example, SampleMapForcePlugin), and then click **Apply**.
5. Check that the MapForceClient workspace plug-in is selected in the 'Plug-ins' tab.
6. Click **Run**. A new Eclipse Workbench opens.
7. Open any MapForce mapping in the new Workbench. It will now open with a zoom level of 70%.

15 Menu Commands

This reference section contains a description of the MapForce menu commands. The following menu commands are available:

15.1 File

This topic lists all the menu commands available in the **File** menu.

⊟ New

Creates a new mapping document. In Professional and Enterprise editions, you can also create a mapping project (`.mfp`). For details, see Projects[116].

⊟ Open

Opens the previously saved mapping design (`.mfd`). In Professional and Enterprise editions, you can also open a mapping project (`.mfp`). For details, see Projects[116].

⊟ Save/Save As/Save All

The **Save** option saves the currently active mapping with its current name. The **Save As** option allows you to save the currently open mapping with a different name. The **Save All** command saves all currently open mapping files.

⊟ Reload

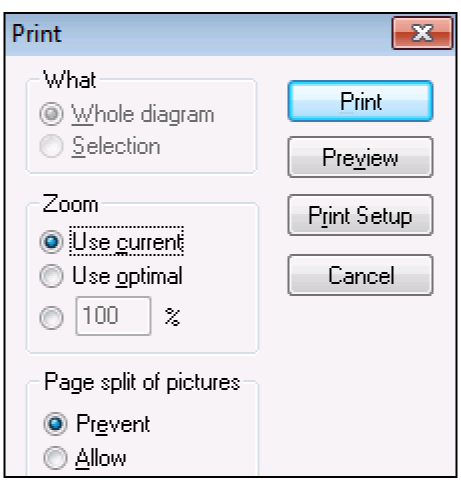
Reloading the currently active mapping reverts your last changes.

⊟ Close/Close All

The **Close** command closes the currently active mapping. The **Close All** command closes all currently open mappings. You are asked if you want to save any of the unsaved files.

⊟ Print/Print Preview/Print Setup

The **Print** command opens the **Print** dialog box (*see below*) that enables you to print out your mapping. **Use current** retains the currently defined zoom factor of the mapping. **Use optimal** scales the mapping to fit the page size. You can also specify the zoom factor numerically. Component scrollbars are not printed. You can also specify if you want to allow graphics to be split over several pages or not.



The **Preview** command opens the same **Print** dialog box with the same settings as described above. The **Print Setup** command opens the **Print Setup** dialog box in which you can select a printer and configure paper settings.

Validate Mapping

The **Validate Mapping** command checks whether all mappings are valid and displays relevant information messages, warnings, and errors. For details, see Validation105.

Mapping Settings

Opens the Mapping Settings dialog box114 where you can define document-specific settings.

Open Credentials Manager (*Enterprise Edition*)

Opens **Credentials Manager** that allows you to manage credentials required in mappings that perform basic HTTP authentication or OAuth 2.0 authorization. For more details, see Credentials Manager1300.

Generate Code in Selected Language/Generate Code in

Generates code in the currently selected language. For more information about the available transformation languages, see Transformation Languages27.

Generate Code in | XSLT (XSLT2, XSLT3)
This command generates the XSLT file(s) from the source file(s). Selecting this option opens the **Browse for Folder** dialog box in which you need to select the location of the XSLT file. The name of the generated XSLT file(s) is defined in the Mapping Settings dialog box114.

Generate Code in | XQuery | Java | C# | C++ (Professional and Enterprise editions)
This command generates source code in one of the selected transformation languages. Selecting this option opens the **Browse for Folder** dialog box in which you need to select the location of the generated files. The names of the generated application files (as well as the `*.csproj` C# project file, `*.sln` solution file, and `*.vcproj` visual C++ project file) are defined in the Mapping Settings114 dialog box.

The file name created by the executed code appears in the **Output XML File** box of the Component Settings123 dialog box if the target is an XML/Schema document.

Compile to MapForce Server Execution File (*Professional and Enterprise editions*)

Generates a file that can be executed by MapForce Server to run the mapping transformation. For details, see Compiling a MapForce mapping1378.

Deploy to FlowForce Server (*Professional and Enterprise editions*)

Deploys the currently active mapping to FlowForce Server. For details, see Deploying a MapForce mapping1381.

Generate Documentation (*Professional and Enterprise editions*)

Generates documentation of your mapping projects in great detail in various output formats. For more information, see Generating Mapping Documentation1340.

Recent files

Displays the list of the most recently opened files.

Exit

Exits the application. You are asked if you want to save any unsaved files.

15.2 Edit

This topic lists all the menu commands available in the **Edit** menu. Most of the commands in this menu become active when you view the result of a mapping in the **Output** pane or preview code, for example, in the **XSLT** pane.

Undo

MapForce has an unlimited number of undo steps that you can use to retrace your mapping steps. You can also use the toolbar command to undo actions.

Redo

The redo command allows you to redo previously undone actions. You can step backward and forward through the undo history using both these commands. You can also use the toolbar command to redo actions.

Find

Allows you to search for specific text in any of the **XQuery** (*Professional and Enterprise editions*), **XSLT**, **XSLT2**, **XSLT3**, and **Output** panes. You can also do a search using the toolbar command.

Find Next

Searches for the next occurrence of the same search string. You can also search for the next occurrence using the toolbar button.

Find Previous

Searches for the previous occurrence of the same search string. Searching for the previous occurrence is also possible with the toolbar command.

Cut/Copy/Paste/Delete

The standard windows Edit commands that allow you to cut, copy, paste, and delete any components or functions visible in the mapping window.

Select all

Selects all components in the **Mapping** pane or the text/code in the **XQuery** (*Professional and Enterprise editions*), **XSLT**, **XSLT2**, **XSLT3**, and **Output** panes.

15.3 Insert

This topic lists all the menu commands available in the **Insert** menu.

⊟ XML Schema/File

Adds an XML schema or instance file to the mapping. If you select an XML file without a schema reference, MapForce can generate a matching XML schema [122]. If you select an XML schema file, you are prompted to include an XML instance file which supplies the data for preview. You can also add an XML/XSD file via the toolbar command.

⊟ Database (*Professional and Enterprise editions*)

Adds a database component. See Databases [161]. You can also add a database component via the toolbar command. In MapForce Enterprise Edition, you can also add NoSQL databases as components. For more information, see NoSQL Databases [437].

⊟ EDI (*Enterprise Edition*)

Adds an EDI document. See EDI [444]. You can also add an EDI component via the toolbar command.

⊟ Text File (*Professional and Enterprise editions*)

Adds a flat file document such as CSV or a fixed-length text file. For details, see CSV and Text Files [563]. You can also add a text file via the toolbar command. MapForce Enterprise Edition also allows you to process text files with FlexText. See MapForce FlexText [658].

⊟ Web Service Function (*Enterprise Edition*)

Adds a call to a Web service. See Web Services [586]. You can also add a Web service via the toolbar button.

⊟ Excel 2007+ File (*Enterprise Edition*)

Adds a Microsoft Excel 2007+ (`.xlsx`) file. See Microsoft OOXML Excel 2007+ [715]. If you do not have Excel 2007+ installed on your machine, you can still map to or from Excel 2007+ files. In this case, you cannot preview the result in the **Output** pane, but you can still save the result. You can also add an Excel file via the toolbar command.

⊟ XBRL Document (*Enterprise Edition*)

Adds an XBRL instance or taxonomy document. For details, see XBRL [745]. You can also add an XBRL component via the toolbar command.

⊟ JSON Schema/File (*Enterprise Edition*)

Adds a JSON schema or file. For details, see JSON [795]. You can also add a JSON component via the toolbar command.

Protocol Buffers File (*Enterprise Edition*)

Adds a binary file encoded in Protocol Buffers format. For more information, see Protocol Buffers[811]. You can also add a file in Protocol Buffers format via the toolbar command.

Insert Input

Simple-Input components can be used as input parameters that are relevant to the entire mapping or only in the context of user-defined functions. For more information, see Simple Input[833] and Parameters in UDFs[933]. You can also insert a Simple-Input component using the toolbar command.

Insert Output

Simple-Output components can be used as output components in mappings and as output parameters of user-defined functions. For more information, see Simple Output[843] and Parameters in UDFs[933]. You can also insert a Simple-Output component using the toolbar command.

Constant

Inserts a constant which supplies fixed data to an input connector. You can select the following types of data: `String`, `Number` and `All other`. You can also insert a constant using the toolbar command.

Variable

Inserts a variable[847], which is equivalent to a regular (non-inline) user-defined function. A variable is a special type of components used to store an intermediate mapping result for further processing. You can also add a variable using the toolbar command.

Join (*Professional and Enterprise editions*)

The Join component allows you to join data in SQL and non-SQL modes. You can also add a Join component using the toolbar command. For details, see Joining Data[860].

Sort: Nodes/Rows

Inserts a component which allows you to sort nodes (see Sort Nodes/Rows[872]). You can also add a Sort component using the toolbar command.

Filter: Nodes/Rows

Inserts a Filter component that can filter data from any other component structure supported by MapForce, including databases. For more information, see Filters and Conditions[878]. You can also add a filter using the toolbar command.

SQL/NoSQL-WHERE/ORDER (*Professional and Enterprise editions*)

Inserts a component which allows you to filter database data conditionally. For more information, see SQL/NoSQL-WHERE/ORDER Component[350]. You can also access the SQL/NoSQL-WHERE/ORDER component via the toolbar command.

Value-Map

Inserts a component that transforms an input value to an output value using a lookup table. This is useful when you need to map a set of values to another set of values (e.g., month numbers to month names).

For more information, see Value-Maps[884]. You can also insert a Value-Map using the toolbar command.

IF-Else Condition

Inserts an If-Else Condition that is suitable for scenarios where you need to process a simple value conditionally. For more information, see Filters and Conditions[878]. You can also add an If-Else Condition using the toolbar command.

Exception (*Professional and Enterprise editions*)

The exception component allows you to interrupt a mapping process when a specific condition is met.

You can also add an Exception component using the toolbar command. In MapForce Enterprise Edition, this component also allows you to define Fault messages in WSDL mapping projects. For details, see Web Service Faults[1218].For more information about Exception components, see Exceptions[900].

15.4 Project

MapForce allows you to group your mappings into mapping projects 116. This topic lists all the menu commands available in the **Project** menu.

⊟ Reload Project

Reloads the currently active project and switches to the **Project** window.

⊟ Close Project

Closes the currently active project.

⊟ Save Project

Saves the currently active project.

⊟ Add Files to Project

Allows you to add mappings to the current project.

⊟ Add Active File to Project

Adds the currently active file to the currently open project.

⊟ Create Folder

This option adds a new folder to the current project. See Project Folders 119.

⊟ Open Mapping (*Enterprise Edition*)

Opens the currently highlighted/selected mapping in the **Project** window.

⊟ Create Mapping for Operation (*Enterprise Edition*)

Creates a mapping file for the currently selected operation of the WSDL project. For details, see SOAP Web Services 1204.

⊟ Add Mapping File for Operation (*Enterprise Edition*)

Allows you to add a previously saved mapping file to the currently active WSDL operation. For details, see SOAP Web Services 1204.

⊟ Insert Web Service (*Enterprise Edition*)

Allows you to insert a Web Service based on an existing WSDL file. For details, see Web Services 586.

⊟ Open in XMLSpy

Opens the selected WSDL file in Altova XMLSpy.

⊟ Generate Code for Entire Project

Generates project code for the entire project currently visible in the **Project** window. Code is generated in the selected language for all the **.mfd** files in each folder.

Generate Code in

Generates project code in the language you select from the context menu.

Properties

Opens a dialog box where you can define project settings 118.

Recent projects

Displays a list of the recently opened projects.

15.5 Component

This topic lists all the menu commands available in the **Component** menu.

Change Root Element

Allows you to change the root element of an XML instance document.

Edit Schema Definition in XMLSpy

To be able to edit a schema in Altova XMLSpy, you need to click an XML component and then select the option **Edit Schema Definition in XMLSpy**.

Edit FlexText Configuration (*Enterprise Edition*)

This command enables you to edit a FlexText file. For details, see MapForce FlexText[658].

Add/Remove/Edit Database Objects (*Professional and Enterprise editions*)

Allows you to add, remove, and change database objects in a database component. See Databases[161].

Create Mapping to EDI X12 997 (*Enterprise Edition*)

The X12 997 Functional Acknowledgment reports the status of the EDI interchange. All errors encountered during processing of the document are reported in it. MapForce can automatically generate a X12 997 document that you will be able to send to the recipient. See Generating an X12 997 Functional Acknowledgment[464].

Create Mapping to EDI X12 999 (*Enterprise Edition*)

The X12 999 Implementation Acknowledgment Transaction Set reports HIPAA implementation guide non-compliance or application errors. MapForce can automatically generate an X12 999 component and automatically create the necessary mapping connections. See Generating an X12 999 Implementation Acknowledgment[465].

Refresh (*Professional and Enterprise editions*)

Reloads the structure of the currently active database component.

Add Duplicate Input Before/After

Inserts a copy of the selected item before/after this item. Duplicated input cannot be used as a data source. For more information, see Duplicate Input[83].

Remove Duplicate

Removes a duplicated item.

Comment/Processing Instructions

This option enables you to insert comments and processing instructions[132] into XML components.

Write Content as CDATA Section

This command creates a CDATA section [133] that is used to represent parts of a document as character data which would normally be interpreted as markup.

Database Table Actions (*Professional and Enterprise editions*)

Allows you to configure database insert, update, and delete actions, and other options for database records. See Database Table Actions Settings [326] for more information.

Query Database (*Professional and Enterprise editions*)

Creates a SELECT statement based on the table/field you clicked in the database component. Clicking a table/field makes this command active, and the SELECT statement is automatically placed into the **Select** window.

Align Tree Left

Makes the tree of a component left-justified.

Align Tree Right

Makes the tree of a component right-justified.

Properties

Displays the settings of the currently selected component. See Change Component Settings [82].

15.6 Connection

This topic lists all the menu commands available in the **Connection** menu.

Auto Connect Matching Children

Activates/deactivates the **Auto Connect Matching Children** option. For more information about connections and their types, see Connections [89].

Settings for Connect Matching Children

Helps you define matching-children connections. For details, see Matching-Children Connections [95].

Connect Matching Children

This command allows you to create multiple connections for items with the same names in source and target components. The settings you define in this dialog box apply if the (**Auto connect child items**) toolbar command has been enabled. For more information, see Matching-Children Connections [95].

Target Driven (Standard)

Changes the connector type to a standard mapping. For further information, see Target-driven vs. source-driven connections [92].

Copy-all (Copy Child Items)

Creates connections for all matching child items. The main benefit of copy-all connections is that they visually simplify the mapping workspace: One connection, represented by a thick line, is created instead of multiple connections. For details, see Copy-All Connections [97].

Source Driven (Mixed Content)

Changes the connection type to a source-driven connection that enables you to automatically map mixed content (text and child nodes) in the same order as in the XML *source* file. For more information, see Source-Driven Connections [93].

Properties

Opens the **Connection Settings** dialog box which allows you to define connection types and annotation settings. For more information, see Connection Settings [98].

15.7 Function

This topic lists all the menu commands available in the **Function** menu.

Create User-Defined Function

Creates a user-defined function[927] (UDF). You can also create a UDF using the toolbar command.

Create User-Defined Function from Selection

Creates a user-defined function based on the currently selected elements in the mapping window. For details, see Create UDFs[930]. You can also create a UDF from selection using the toolbar command.

Function Settings

Opens the **Edit User-defined Function** dialog box that allows you to change a UDF's settings. For details, see Edit UDFs[931].

Remove Function

Deletes the currently active user-defined function if you are working in a context which allows this.

Insert Input

Simple-Input components can be used as input parameters that are relevant to the entire mapping or only in the context of user-defined functions. For more information, see Simple Input[833] and Parameters in UDFs[933]. You can also insert a Simple-Input component using the toolbar command.

Insert Output

Simple-Output components can be used as output components in mappings and as output parameters of user-defined functions. For more information, see Simple Output[843] and Parameters in UDFs[933]. You can also insert a Simple-Output component using the toolbar button.

15.8 Output

This topic lists all the menu commands available in the **Output** menu.

XSLT 1.0/XSLT 2.0/XSLT 3.0/XQuery/Java/C#/C++/Built-In

Sets the transformation language in which the mapping should be executed. The selection of transformation languages depends on your MapForce edition. For details, see Transformation Languages [27]. You can also select transformation languages in the toolbar.

Validate Output File

Validates the output XML file against the referenced schema. See Validation [105].

Save Output File

Saves the data in the **Output** pane to a file.

Save All Output Files

Saves all the generated output files of dynamic mappings [1313]. See also Tutorial 4 [67].

Regenerate Output

Reloads the data in the **Output** pane.

Run SQL/NoSQL-Script (*Professional and Enterprise editions*)

If an SQL/NoSQL script is currently visible in the **Output** pane, the script executes the mapping to the target database, taking the defined table actions into account. To find out more about databases, see Databases [161].

Insert/Remove Bookmark

Inserts/removes a bookmark at the cursor position in the **Output** pane.

Next/Previous Bookmark

Navigates to the next/previous bookmark in the **Output** pane.

Remove All Bookmarks

Removes all currently defined bookmarks in the **Output** pane.

Pretty-Print XML Text

Reformats your XML document in the **Output** pane so that the document has a structured display: Each child node is offset from its parent by a single tab character. In the **Output** pane, the Tab size settings defined in the Text View Settings [107] dialog (Tabs group) take effect.

Text View Settings

Displays the **Text View Settings** dialog box that allows you to customize text view settings in the **XQuery** pane (*Professional and Enterprise editions*), the **Output** pane, and the **XSLT** pane. The dialog also shows the currently defined hotkeys. For more information, see Text View Features[107].

15.9 Debug

This topic lists all the menu commands available in the **Debug** menu.

Start Debugging

Starts or continues debugging until a breakpoint is hit or the mapping finishes. You can also start debugging using the toolbar command. For more information about debugging mode, see Debugger 1349.

Stop Debugging

Stops debugging. This command exits the debug mode and switches MapForce back to standard mode. You can also stop debugging using the toolbar command. For more information about debugging mode, see Debugger 1349.

Step Into

Executes the mapping until a single step is finished anywhere in the mapping. In the mapping debugger, a step is a logical group of dependent computations which normally produce a single item of a sequence. Depending on the mapping context, this command roughly says the following: *go to the left/go to target child/go to source parent*. You can also access this command using the toolbar button. For more information about debugging mode, see Debugger 1349.

Step Over

Continues execution until the current step finishes (or finishes again for another item of the sequence) or an unrelated step finishes. This command steps over computations that are inputs of the current step. You can also access this command using the toolbar button. For more information about debugging mode, see Debugger 1349.

Step Out

Continues execution until the result of the current step is consumed or a step is executed that is not an input or child of the consumption. This command steps out of the current computation. Depending on the mapping context, this command roughly says the following: *go to the right/go to target parent/go to source child*. You can also access this command using the toolbar button. For more information about debugging mode, see Debugger 1349.

Minimal Step

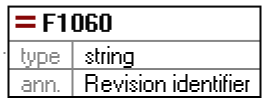
Continues execution until a value is produced or consumed. This command subdivides a step and will typically stop twice for each connection: (i) once when its source produces a value and (ii) once when its target consumes it. MapForce does not necessarily compute values in the order the mapping would suggest, so production and consumption events do not always follow each other. You can also access this command using the toolbar button. For more information about debugging mode, see Debugger 1349.

15.10 View

This topic lists all the menu commands available in the **View** menu.

Show Annotations

Displays annotations in the component. You can also enable this option by clicking the toolbar button. If the **Show Types** icon is also active, both sets of information are shown in grid form (*see screenshot below*). You can also use annotations to label connections. For details, see Connection Settings[100].



Show Types

Displays data types in a component. You can also enable this option by clicking the toolbar button. If the **Show Annotations** icon is also active, then both sets of information are shown in grid form (*see Show Annotations above*).

Show Library in Function Header

Displays the library name in the function's header. You can also enable this option by clicking the toolbar button.

Show Tips

When you place the cursor over a function's header, you will see a tooltip summarizing what this function does. With the **Show Tips** option enabled, you can also see information about datatypes in a component.

XBRL Display Options (*Enterprise Edition*)

MapForce enables you to configure the following XBRL settings:

- The label language of XBRL items and their annotations
- The preferred label roles for XBRL item names
- The specific type of label roles of annotations for XBRL items
- Custom XBRL Taxonomy Packages

For more information, see XBRL Options[1477].

Show Selected Components Connectors/Connections from Source to Target

These options allow you to highlight connections selectively. To find out how these options work, see Connections[91].

Zoom

Opens the **Zoom** dialog box. You can enter the zoom factor numerically or drag the slider to change the zoom factor interactively.

Back/Forward

The **Back** and **Forward** commands allow you to switch between the previous and next mappings you have been working on, relative to the currently open mapping.

Status Bar

Switches on/off the **Status Bar** visible below the **Messages** window.

Libraries/Manage Libraries

Click **Libraries** to switch on/off the **Libraries** window. Click **Manage Libraries** to switch on/off the **Manage Libraries** window.

Messages

Switches on/off the Messages window 36. When code is generated, the **Messages** window is automatically activated to show the validation result.

Overview

Switches on/off the Overview window 36. Drag the rectangle to navigate your way through the mapping.

Project Window (*Professional and Enterprise editions*)

Switches on/off the **Project** window. To find out more about projects, see Projects 116.

Debug Windows (*Professional and Enterprise editions*)

The debug mode enables you to analyze the context in which a particular value is produced. This information is available directly in the mapping and in the **Values**, **Context**, and **Breakpoints** windows. For more information, see About Debug Mode 1353.

15.11 Tools

This topic lists all the menu commands available in the **Tools** menu.

⊟ Global Resources

Opens the **Manage Global Resources** dialog box that enables you to add, edit, and delete settings applicable across multiple Altova applications. For more information, see Altova Global Resources 1397.

⊟ Active Configuration

Allows you to select the currently active global resource configuration from a list of configurations. To create and configure different types of global resources, see Altova Global Resources 1397.

⊟ Create Reversed Mapping

Creates a reversed mapping from the currently active mapping, which means the source component becomes the target component, and the target component becomes the source. Note that only direct connections between components are retained in the reversed mapping. It is likely that the new mapping will not be valid or suitable for preview in the **Output** pane. Therefore, the new mapping would require manual editing.

The following data is retained:

- Direct connections between components
- Direct connections between components in a chained mapping
- The type of connection 92: Standard, Mixed-Content, Copy-All
- Pass-through component settings
- Database components (*Professional and Enterprise editions*)

The following data is *not* retained:

- Connections via functions, filters, etc.
- User-defined functions
- Web service components (*Enterprise Edition*)

⊟ XBRL Taxonomy Manager (*Enterprise Edition*)

XBRL Taxonomy Manager is a tool that allows you to install and manage XBRL taxonomies. For more information, see XBRL Taxonomy Manager 751.

⊟ XML Schema Manager

XML Schema Manager is an Altova tool that provides a centralized way to install and manage XML schemas (DTDs for XML and XML Schemas) for use across all Altova's XBRL-enabled applications. For more information, see Schema Manager 145.

⊟ Customize

This option allows you to customize the MapForce graphical user interface. This includes showing/hiding toolbars as well as customizing menus 1470 and keyboard shortcuts 1471.

⊟ Restore Toolbars and Windows

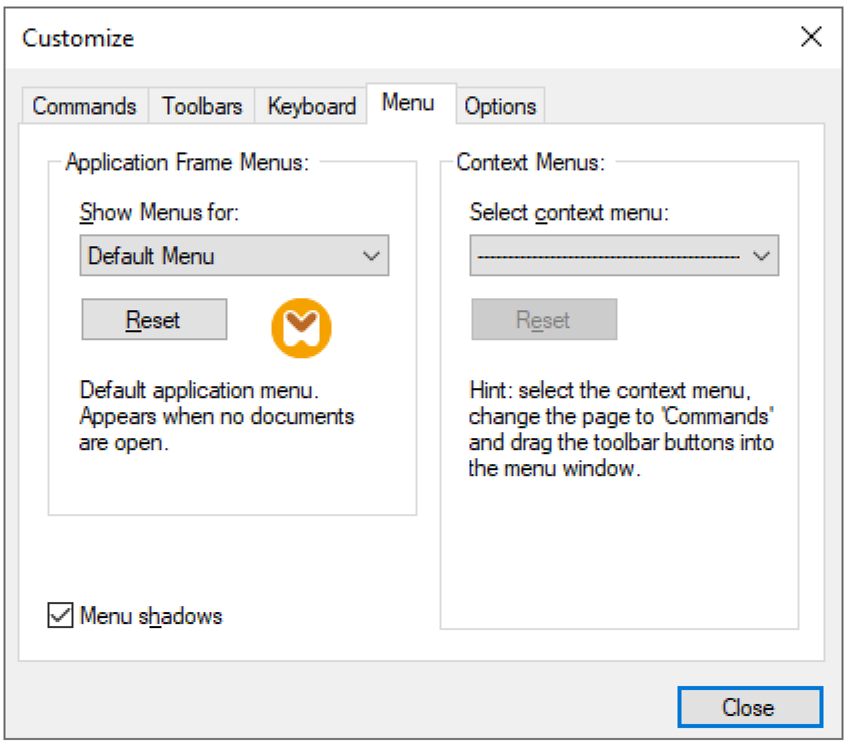
Resets the toolbars, entry helper windows, docked windows etc. to their defaults. You need to restart MapForce so that the changes take effect.

Options

Opens the **Options** dialog box that enables you to change the default MapForce settings. For more information, see Options1473.

15.11.1 Customize Menus

You can customize standard MapForce menus and context menus (e.g., to add, change, or remove commands). You can also revert your changes to the default state (**Reset**). To customize menus, go to **Tools | Customize** and click the **Menu** tab (*see screenshot below*).



Default Menu vs. MapForce Design
The *Default Menu* bar is displayed when no document is open in the main window. The *MapForce Design* menu bar is displayed when one or more mappings are open. Each menu bar can be customized separately. Customization changes made to one menu bar do not affect the other.

To customize a menu bar, select it from the *Show Menus For* drop-down list. Then click the **Commands** tab and drag commands from the *Commands* list box to the menu bar or into any of the menus.

Delete commands from menus
To delete an entire menu or a command inside a menu, do the following:

1. Select *Default Menu* or *MapForce Design* from the *Show Menus for* drop-down list.
2. With the **Customize** dialog open, select a toolbar command you want to delete or a command you want to delete from one of the menus.
3. Drag the toolbar command from the toolbar or the command from the menu. Alternatively, right-click the toolbar command or menu command and select **Delete**.

You can reset any menu bar to its default state by selecting it from the *Show Menus For* drop-down list and clicking the **Reset** button.

Customize context menus
Context menus appear when you right-click certain objects in the application's interface. Each of these context menus can be customized in the following way:

1. Select a context menu from the *Select context menu* drop-down list. This opens the respective context menu.
2. Open the **Commands** tab and drag a command from the *Commands* list box into the context menu.
3. To delete a command from the context menu, right-click that command and select **Delete**. Alternatively, drag the command out of the context menu.

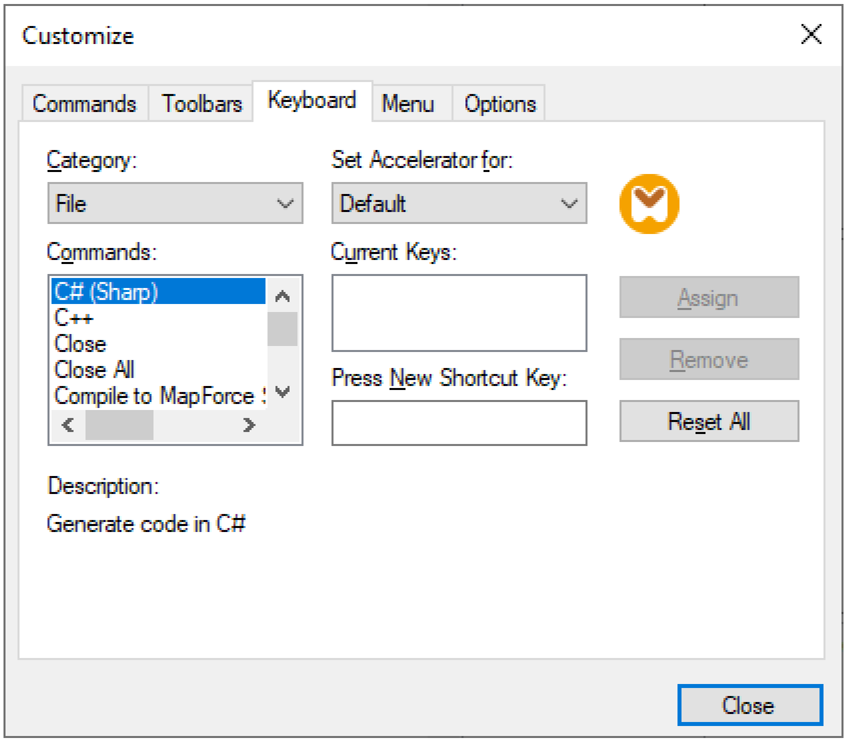
You can reset any context menu to its default state by selecting it in the *Select context menu* drop-down list and clicking the **Reset** button.

Menu shadows
Select the *Menu shadows* check box to give all menus shadows.

15.11.2 Customize Shortcuts

You can define or change keyboard shortcuts in MapForce as follows: Select the **Tools | Customize** and click the **Keyboard** tab. To assign a new shortcut to a command, take the following steps:

1. Select the **Tools | Customize** command and click the **Keyboard** tab (*see screenshot below*).
2. Click the *Category* combo box to select the menu name.
3. In the *Commands* list box, select the command you want to assign a new shortcut to.
4. Type in new shortcut keys in the *Press New Shortcut Key* text box and click **Assign**.



To clear the entry in the *Press New Shortcut Key* text box, press any of the control keys: **Ctrl**, **Alt** or **Shift**. To delete a shortcut, click the shortcut you want to delete in the *Current Keys* list box and click **Remove**.

Note: The *Set accelerator for* does not currently have any function.

Keyboard shortcuts

By default, MapForce provides the following keyboard shortcuts:

F1	Help Menu
F2	Next bookmark (in output window)
F3	Find Next
F10	Activate menu bar
Num +	Expand current item node
Num -	Collapse item node
Num *	Expand all from current item node
CTRL + TAB	Switches between open mappings
CTRL + F6	Cycle through open windows
CTRL + F4	Closes the active mapping document
Alt + F4	Closes MapForce
Alt + F, F, 1	Opens the last file
Alt + F, T, 1	Opens the last project
CTRL + N	File New
CTRL + O	File Open
CTRL + S	File Save

CTRL + P	File Print
CTRL + A	Select All
CTRL + X	Cut
CTRL + C	Copy
CTRL + V	Paste
CTRL + Z	Undo
CTRL + Y	Redo
Del	Delete component (with prompt)
Shift + Del	Delete component (no prompt)
CTRL + F	Find
F3	Find Next
Shift + F3	Find Previous
Arrow keys	
(up / down)	Select next item of component
Esc	Abandon edits/close dialog box
Return	Confirms a selection
Output window hotkeys	
CTRL + F2	Insert Remove/Bookmark
F2	Next Bookmark
SHIFT + F2	Previous Bookmark
CTRL + SHIFT + F2	Remove All Bookmarks
Zooming hotkeys	
CTRL + mouse wheel forward	Zoom In
CTRL + mouse wheel back	Zoom Out
CTRL + 0 (Zero)	Reset Zoom

15.11.3 Options

You can change general and other preferences in MapForce by selecting the **Tools | Options** command. The available options are described below.

General

In the *General* section, you can define the following options:

- *Show logo* | *Show on start:* Shows or hides an image (splash screen) when MapForce starts.
- *Mapping view.* You can enable/disable the gradient background in the **Mapping** pane (*Show gradient background*). You can also choose to limit annotation display to N lines. If the annotation text contains multiple lines, enabling this option shows only the first N lines (which is the value you specify) in the component. This setting also applies to SELECT statements visible in a component.
- *Default encoding for new components.*

Encoding name: The default encoding for new XML files can be set by selecting an option from the dropdown list. If a two- or four-byte encoding is selected as the default encoding (i.e. UTF-16, UCS-2, or UCS-4), you can also choose between little-endian and big-endian byte-ordering. This setting can also be changed individually for each component (see Change Component Settings 82).

Byte order: When a document with two-byte or four-byte character encoding is saved, the document can be saved either with little-endian or big-endian byte-ordering. You can also specify whether a byte order mark should be included.

- *Preview Settings:* The *Use execution timeout* option sets an execution timeout when you preview the mapping result in the **Output** pane.

- *On activating Output tab:* You can generate output to temporary files or write the output directly to an output file (*see below*).

  Generate output to temporary files: This is the default option. If the output file path contains folders that do not exist yet, MapForce will create these folders. For Professional and Enterprise editions: If you intend to deploy the mapping to a server for execution, any directories in the path must exist on the server; otherwise, an execution error will occur. See also Preparing Mappings for Server Execution 1372.

  Write directly to final output files: If the output file path contains folders that do not exist yet, an error will occur. This option overwrites any existing output files without requesting further confirmation.

- *Display text in steps of N million characters:* Specifies the maximum size of the text displayed in the **Output** pane when you preview mappings that generate large XML and text files. If the output text exceeds this value, you will need to click the **Load more** button to load the next chunk. For more information, see Preview and validate output 105.

Editing

In the *Editing* section, you can define mapping view options:

- *Align components on mouse dragging:* Specify whether components or functions should be aligned with other components, while you drag them in the Mapping window. For more information, see Align Components 83.

- *Smart component deletion:* MapForce allows you to keep connections even after deleting some transformation components 78. Keeping connections might be particularly useful with multiple child connections, because you will not have to restore every single child connection manually after deleting a transformation component. For details, see Keep Connections after Deleting Components 103.

Messages

The *Messages* section allows you to switch on message notifications such as suggesting connecting ancestor items, informing about the creation of multiple target components, and so on.

Generation (*Professional and Enterprise editions*)

The *Generation* section allows you to define settings for program code generation and MapForce Server Execution files. For more information, see Code Generator Options [1625] and Compiling Mappings to MapForce Server Execution Files [1378].

⊟ Java

You may need to add a custom Java VM path, for example, if you are using a Java virtual machine which does not have an installer and does not create registry entries (e.g., Oracle's OpenJDK). You might also want to set this path if you need to override any Java VM path detected automatically by MapForce. For details, see Java [1476].

⊟ XBRL (*Enterprise Edition*)

MapForce enables you to configure the following general (application-wide) XBRL settings:

- The label language of XBRL items and their annotations
- The preferred label roles for XBRL item names
- The specific type of label roles of annotations for XBRL items
- Custom XBRL Taxonomy Packages (see XBRL Taxonomy Packages [1478])

For details, see General XBRL Options [1477].

⊟ Debugger (*Professional and Enterprise editions*)

In the *Debugger* section, you can define the following debugging settings:

- *Maximum storage length of values:* Defines the string length of values displayed in the **Values** window (at least 15 characters). Note that setting the storage length to a high value may deplete available system memory.

- *Keep full trace history:* Instructs MapForce to keep the history of all values processed by all connectors of all components in the mapping for the duration of debugging. If this option is enabled, all values processed since the beginning of debug execution will be stored in memory and available for your analysis in the **Values** window until you stop debugging. It is not recommended to enable this option if you are debugging data-intensive mappings, since it may slow down debugging execution and deplete available system memory. If this option is disabled, MapForce keeps only the most recent trace history for nodes related to the current execution position.

⊟ Database (*Professional and Enterprise editions*)

In the *Database* section, you can define database query settings. For details, see Database Query Settings [1480].

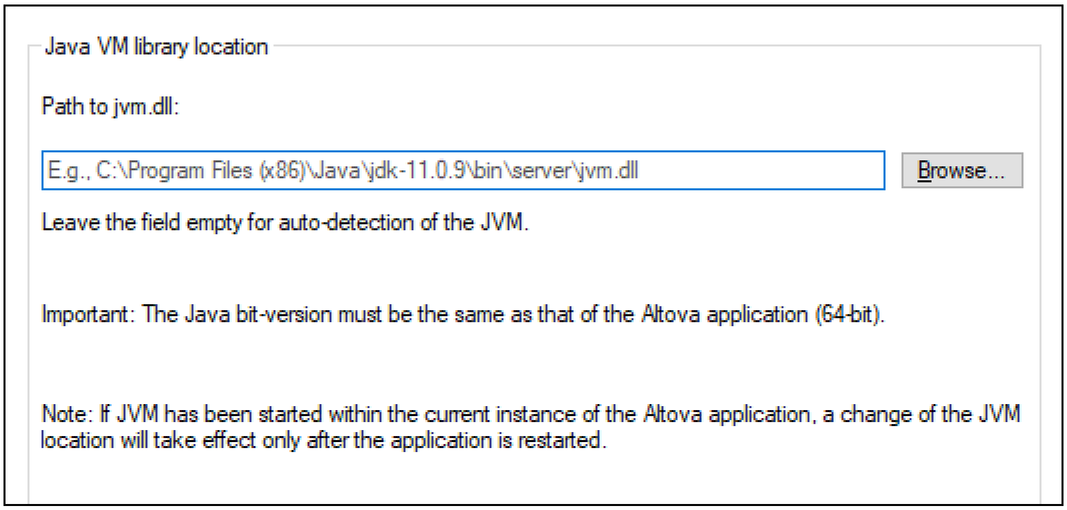
⊟ Network Proxy

The *Network Proxy* section enables you to configure custom proxy settings. These settings affect how the application connects to the Internet. By default, the application uses the system's proxy settings, so you should not need to change the proxy settings in most cases. For more details, see Network Proxy [1483].

15.11.3.1 Java

In the *Java* section (*see screenshot below*), you can optionally enter the path to a Java VM (Virtual Machine) on your file system. Note that adding a custom Java VM path is not always necessary. By default, MapForce attempts to detect the Java VM path automatically by reading (in this order) the Windows registry and the JAVA_HOME environment variable. The custom path added in this dialog box will take priority over any other Java VM path detected automatically.

You may need to add a custom Java VM path, for example, if you are using a Java virtual machine which does not have an installer and does not create registry entries (e.g., Oracle's OpenJDK). You might also want to set this path if you need to override, for whatever reason, any Java VM path detected automatically by MapForce.



Note the following:

- The Java VM path is shared between Altova desktop (not server) applications. Consequently, if you change it in one application, it will automatically apply to all other Altova applications.
- The path must point to the `jvm.dll` file from the `\bin\server` or `\bin\client` directory, relative to the directory where the JDK was installed.
- The MapForce platform (32-bit, 64-bit) must be the same as that of the JDK.
- After changing the Java VM path, you may need to restart MapForce for the new settings to take effect.

Changing the Java VM path affects the following areas:

- JDBC connectivity
- Java extension functions for XSLT/XPath

This setting does not affect Java code generation.

15.11.3.2 XBRL

MapForce enables you to configure the following general (application-wide) XBRL settings:

- The label language of XBRL items and their annotations
- The preferred label roles for XBRL item names
- The specific type of label roles of annotations for XBRL items
- Custom XBRL Taxonomy Packages

Labels are defined in the XBRL Glossary. To find out more about label roles, see *When to Use Multiple Label Roles and Types*.

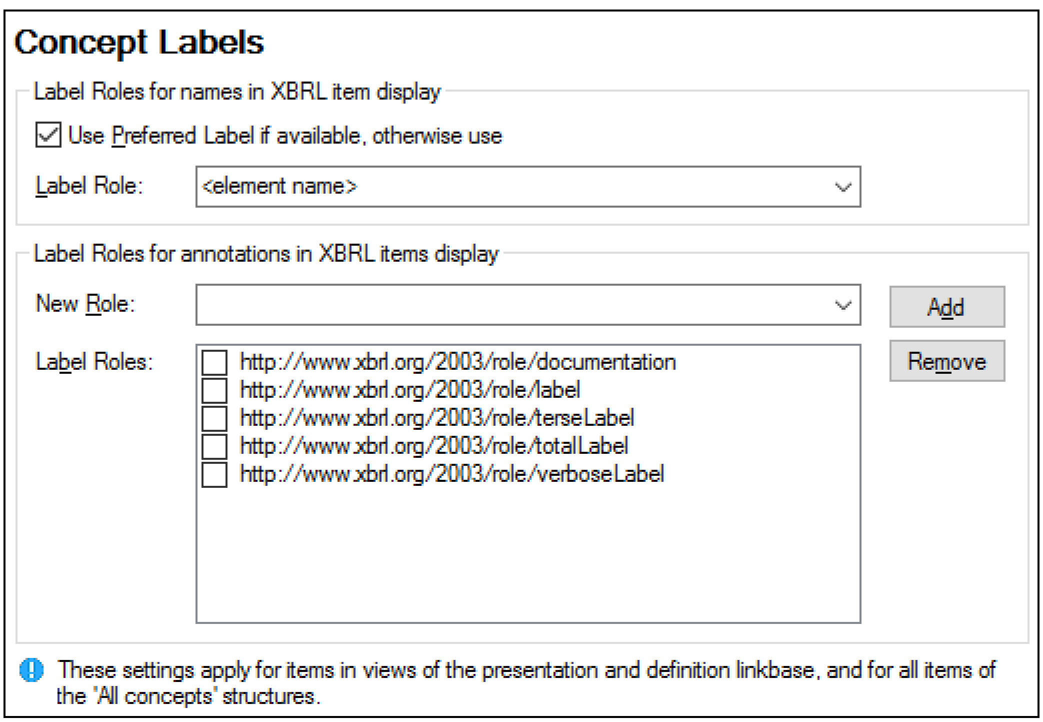
To configure XBRL options, select **Tools | Options** or **View | XBRL Display Options** and open the *XBRL* section. For details, see the subsections below.

General settings, concept labels, generic labels

This subsection describes various general settings you can define in the *XBRL* section, such as the label language of XBRL items and annotations, label roles and annotations in different Structure Views 746.

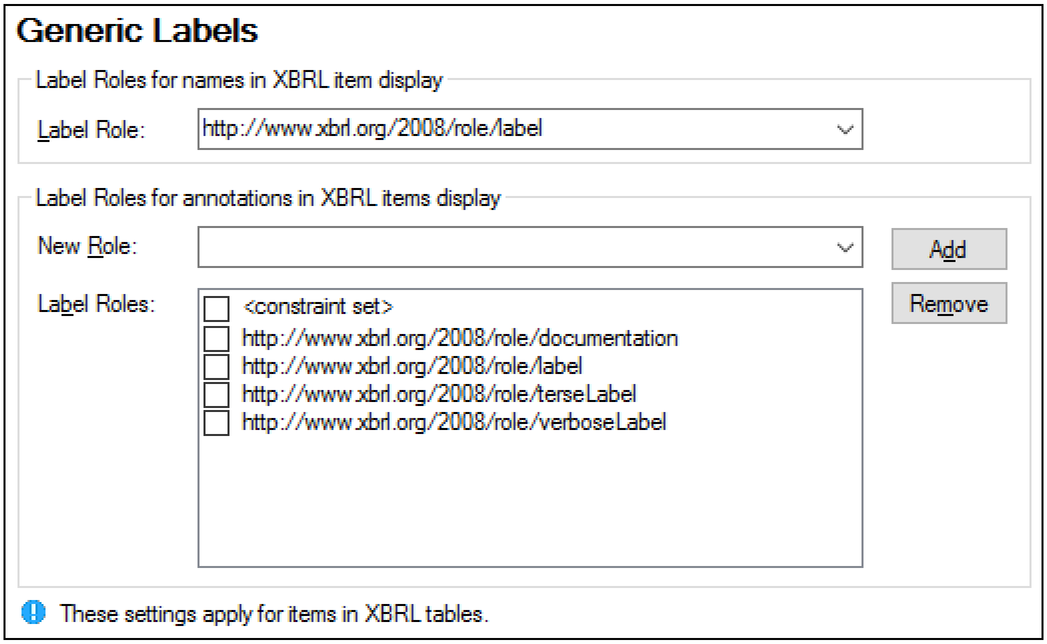
General
The *General* subsection allows you to change the label language of XBRL items and their annotations.

Concept Labels
The *Concept Labels* subsection (*see below*) allows you to change label roles and annotations in the following Structure Views: *Views from presentation and definition linkbases*, *All concepts (with context management)*, and *All concepts (raw)*. For more information about Structure Views, see Selecting Structure Views 746.



Generic Labels

The *Generic Labels* subsection allows you to change label roles and annotations of items in the *Tables from table linkbase* view. See Selecting Structure Views[746] for more information.



Taxonomy packages

In the *Taxonomy Packages* subsection, you can add and change standard and custom XBRL taxonomy packages. An XBRL taxonomy package is a ZIP archive that contains an offline copy of an XBRL taxonomy. The advantages of an XBRL taxonomy package is that XBRL resources such as schemas are resolved from local (offline) taxonomy files rather than being retrieved from the Internet. The structure and format of XBRL taxonomy packages is laid out in the Taxonomy Packages 1.0 Recommendation.

If your XBRL taxonomy provider offers XBRL taxonomy packages conforming to the Recommendation, you can add such packages to MapForce. This affects XBRL validation and processing as certain URLs such as XBRL schema references will be redirected to local files included in the taxonomy package. Taxonomy packages also enable you to view all entry points of the XBRL taxonomy directly from MapForce and improve the performance of XBRL validation.

Many XBRL taxonomies are hosted by Altova, and you can easily manage them using XBRL Taxonomy Manager[751]. If the XBRL taxonomy that you are looking to use is already managed by the XBRL Taxonomy Manager, there is no need to download a custom XBRL taxonomy package for it.

Altova applications with XBRL support

The following Altova desktop applications support XBRL taxonomy packages:

- Altova XBRL Add-ins for Excel (EBA, Solvency II, ESEF)
- MapForce Enterprise Edition
- MapForce Server
- RaptorXML+XBRL Server
- StyleVision Server

- [StyleVision Enterprise Edition]
- [XMLSpy Enterprise Edition]

The XBRL taxonomy package list is common to all these applications. Changes in XBRL taxonomies using XBRL Taxonomy Manager [751] take effect for all the applications listed below if they are installed on the same computer.

How taxonomy packages work

Each XBRL taxonomy package contains an internal catalog file (**`catalog.xml`**). This catalog file defines the URIs that should be redirected (remapped) to local resources. Catalog files conform to a restricted subset of the XML Catalogs specification. A similar mechanism also exists in MapForce to remap URIs to local resources (see Catalog Files) [1418].

After you add one or more XBRL taxonomy packages to MapForce, these packages will influence, with immediate effect, how MapForce resolves resources for XBRL validation. For example, if you add an XBRL instance to the mapping, its schema will be loaded from the local taxonomy package instead of being retrieved from the Internet (assuming that the taxonomy package is active and its **`catalog.xml`** file remaps the schema URI to a local resource). As a result, this will affect all further XBRL validation and processing.

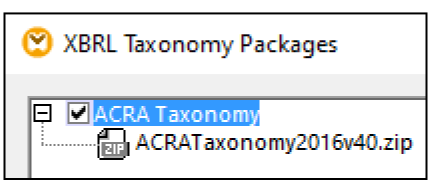
> **Note:** A resource pointed to by the catalog file of an active package will be used for all MapForce operations that require that resource. An example of such a resource would be an XML schema, which is used for XML validation as well as XBRL validation.
>
> If the offline resource located in the package's catalog file is incompatible with your existing environment, this may cause errors. In this case, deactivate the taxonomy package and contact the creators of the package.

Add a taxonomy package

To install or change standard taxonomy packages, use XBRL Taxonomy Manager [751]. To add a custom taxonomy package, follow the instructions below:

1. Download the XBRL taxonomy package from your provider and save the package to a local directory that is not likely to change. Do *not* unzip the downloaded archive. The downloaded **`.zip`** archive must conform to the Taxonomy Packages 1.0 Recommendation.
2. Select **Tools | Options | XBRL | Taxonomy Packages** and click **Edit**.
3. Click **Add Taxonomy** and browse for the **`.zip`** taxonomy package. To select multiple packages, hold the **Shift** key pressed.

The list of added XBRL taxonomy packages is displayed in the dialog box as a tree with two levels: (i) the first level indicates the taxonomy, and (ii) the second level shows the packages of that taxonomy (*see screenshot below*). When a taxonomy package is selected in the **XBRL Taxonomy Packages** dialog, the package details are displayed in the dialog's lower pane.



The check box to the left of a taxonomy entry indicates whether that taxonomy is active. By default, a newly added taxonomy is active. If you wish to add an additional package to an existing taxonomy, select the taxonomy entry and then click **Add Packages**. To remove a package, select it and click **Remove** (note this does not delete the actual taxonomy files from the disk).

15.11.3.3 Database

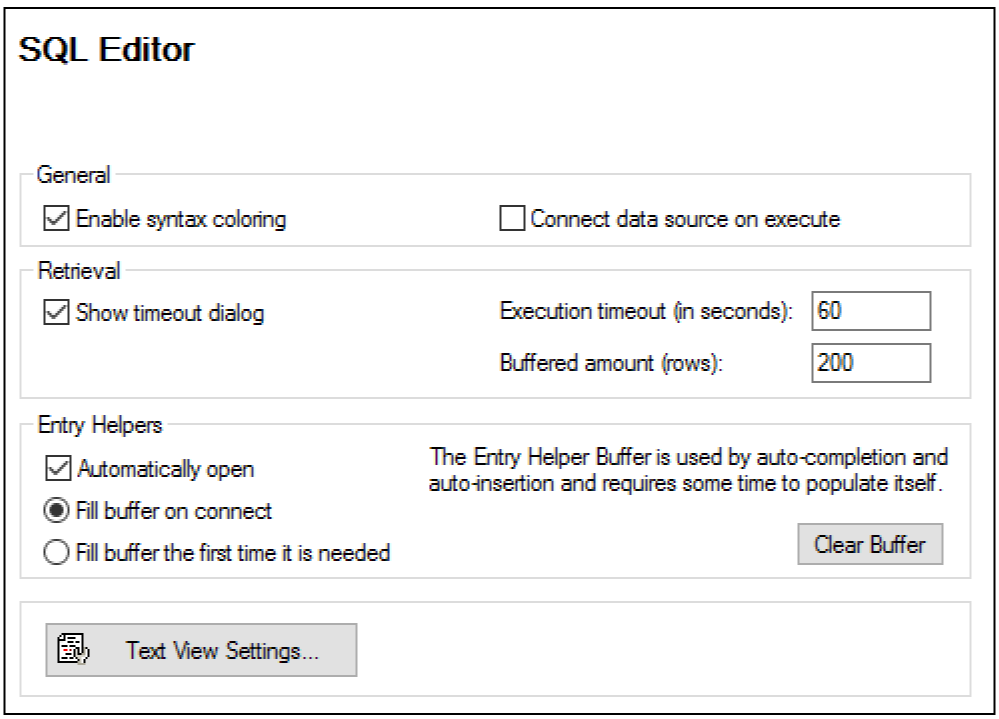
This section explains how to configure various SQL editing settings. You can access the settings in one of the following ways:

- By opening the **DB Query** pane and clicking (**Options**).
- By selecting **Tools | Options | Database** and then the relevant section.

In the *Database* section, you can define general SQL editing settings, encoding and result view options, SQL generation parameters, and fonts. For details, see the subsections below.

SQL Editor

The *SQL Editor* section allows you to change general SQL editing settings (*see screenshot below*). The available settings are described below.



- *General.* To see different elements of SQL syntax in distinct colors, enable syntax coloring. Select the *Connect data source on execute* check box to connect to the relevant data source automatically whenever an SQL statement is executed.

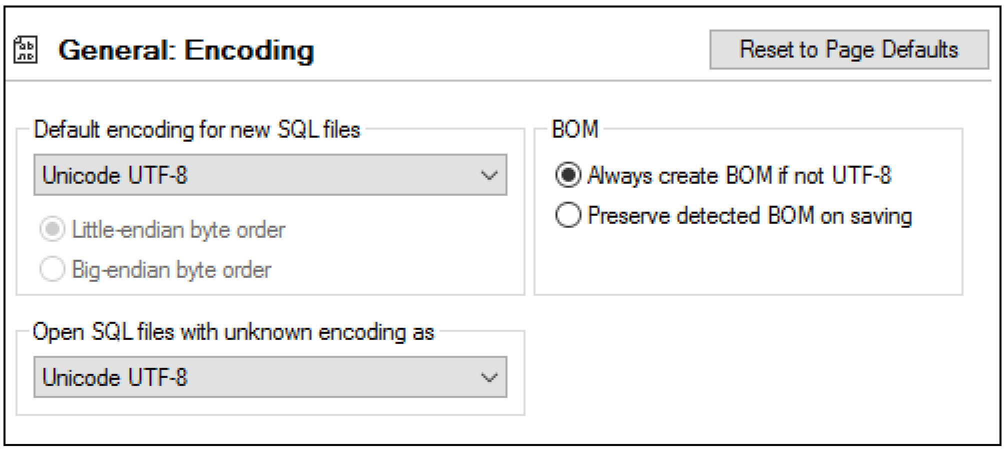
- *Retrieval.* Activating the *Show timeout dialog* check box allows you to change the timeout settings when the permissible execution period is exceeded. You can specify the maximum amount of time allowed for SQL execution (*Execution timeout*) in seconds. You can also define the number of rows that will be put into a buffer.

- *Entry Helpers.* To enable auto-completion suggestions as you start typing SQL statements, select the *Automatically open* check box (see also Auto-Completion[278]). You can choose when to fill in the entry helper buffer: when you connect to a data source or when the buffer is used for the first time. Note that filling the buffer may take some time. Use the **Clear Buffer** button to reset the buffer.

- *Text View Settings:* Allows you to define different Text View settings such as margins, tabs, visual aids, auto-highlighting options, and Text View navigation hotkeys. For more information, see Text View Features[107].

Encoding

In the *Encoding* section, you can specify encoding options for SQL files created or opened with SQL Editor (*see screenshot below*).



- *Default encoding for new SQL files:* Define default encoding for new files so that each new document includes the same encoding. If a two- or four-byte encoding is selected as the default encoding (e.g., UTF-16, UCS-2, or UCS-4), you can also choose between little-endian and big-endian byte ordering for SQL files. The encoding of existing files is not affected by this setting.

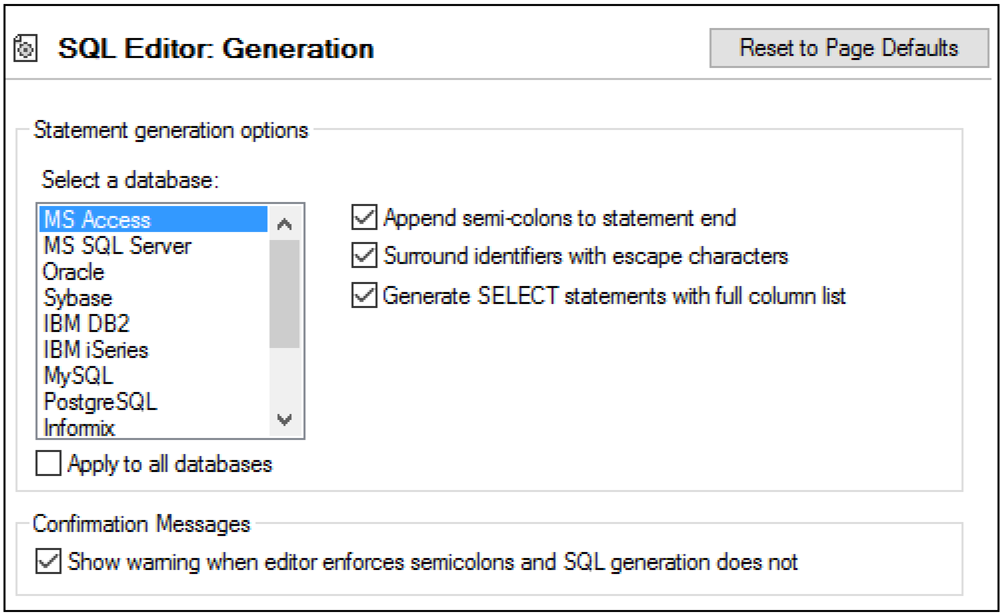
- *Open SQL files with unknown encoding as:* You can select the encoding with which to open an SQL file with unknown encoding.

- *BOM:* Choose whether to create the byte order mark (BOM) character for encoding other than UTF-8 or whether to keep the detected BOM.

Note: SQL files which have no encoding specification are saved with UTF-8 encoding.

SQL Generation

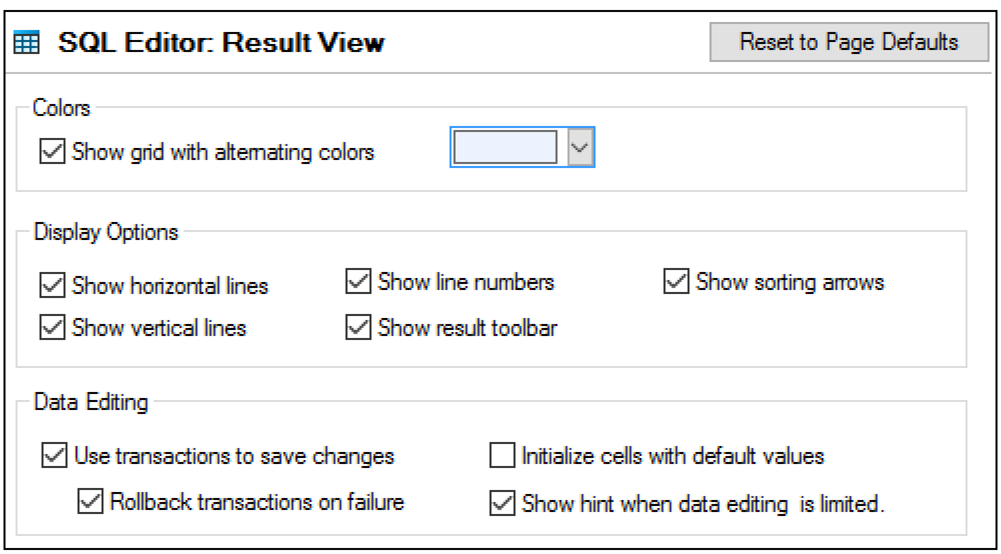
The *Generation* section enables you to specify SQL statement generation syntax for various database kinds (*see screenshot below*).



To define the syntax preferences for a specific database, select it from the list and then enable or disable options to the right. You can also choose to apply the same set of settings to all databases (the *Apply to all databases* check box). Note that using common settings for all databases may prevent you from editing data in Oracle, IBM DB2, and iSeries databases via a JDBC connection.

Result View

You can configure the appearance of the **Results** tab of the **DB Query** pane in the *Result View* section (*see screenshot below*).

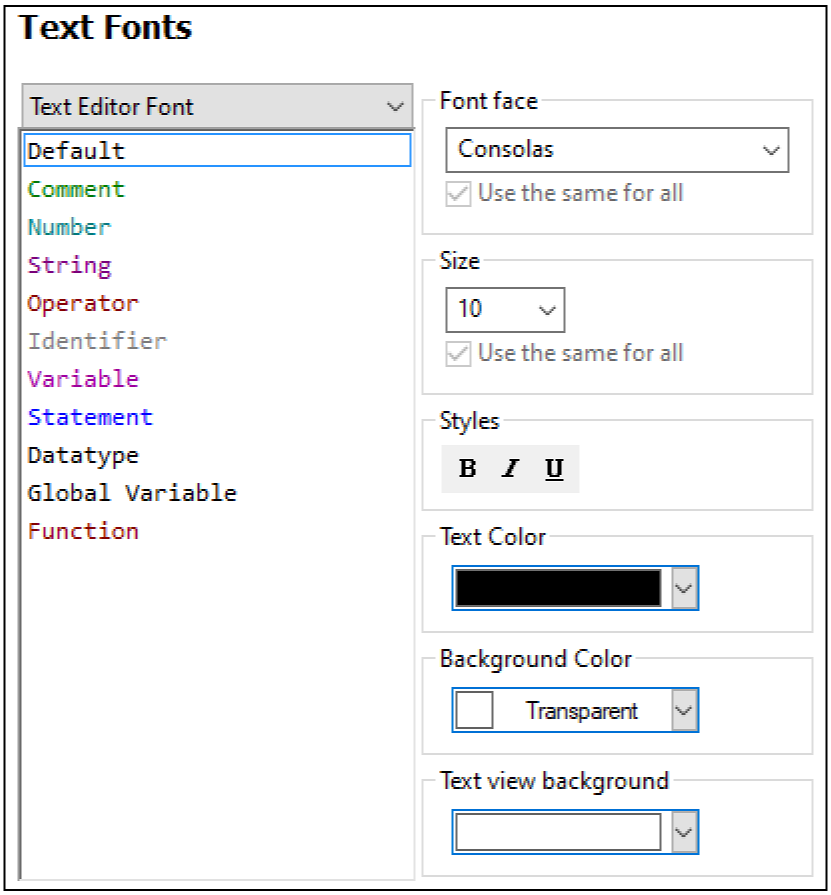


Select the *Show grid with alternating colors* check box to display rows in **Result** tabs as a simple grid or a grid with alternating white and colored rows. The alternating color is configurable. The *Display Options* group allows you to define how to display horizontal and vertical grid lines, line numbers, and the **Result** toolbar.

The *Data Editing* group allows you to define transaction settings if the cells are to be filled with default values, and if a hint is to be displayed when data editing is limited.

Text Fonts

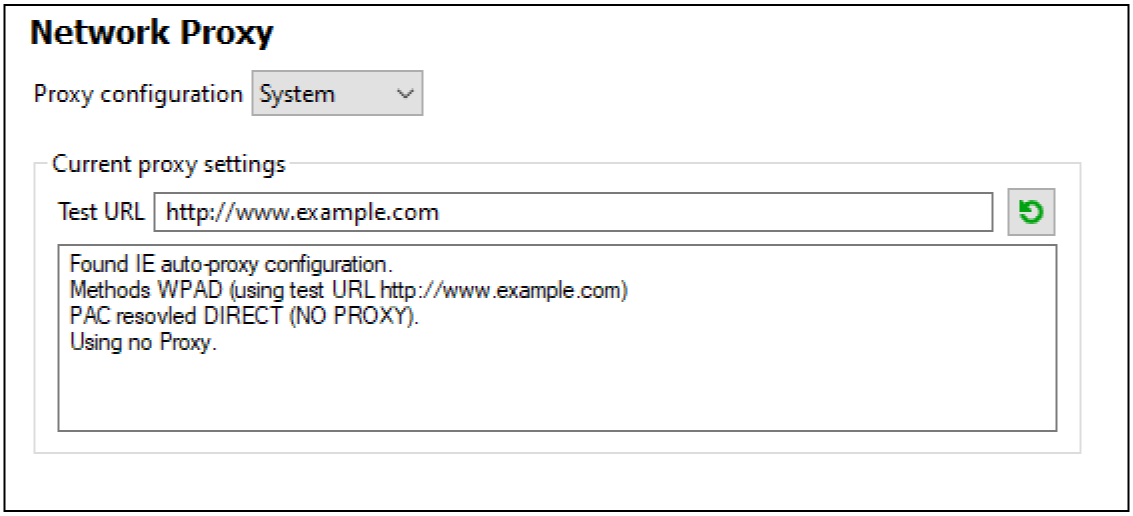
In the *Text Fonts* section, you can configure color and font settings of SQL statements (*see screenshot below*). You can choose the common font face, style, size, and syntax coloring of various text types that appear in SQL Editor. Note that the same font and size are used for all text types. Only the style can be changed for individual text types. If you access the **Text Fonts** dialog box from the **DB Query** pane, you can click the **Reset to Page Defaults** button to restore the original settings.



15.11.3.4 Network Proxy

The *Network Proxy* section enables you to configure custom proxy settings. These settings affect how the application connects to the Internet (for XML validation purposes, for example). By default, the application uses the system's proxy settings, so you should not need to change the proxy settings in most cases. If necessary, however, you can set an alternative network proxy by selecting, in the *Proxy Configuration* combo box, either *Automatic* or *Manual* to configure the settings accordingly.

Note: The network proxy settings are shared among all Altova MissionKit applications. So, if you change the settings in one application, all MissionKit applications will be affected.

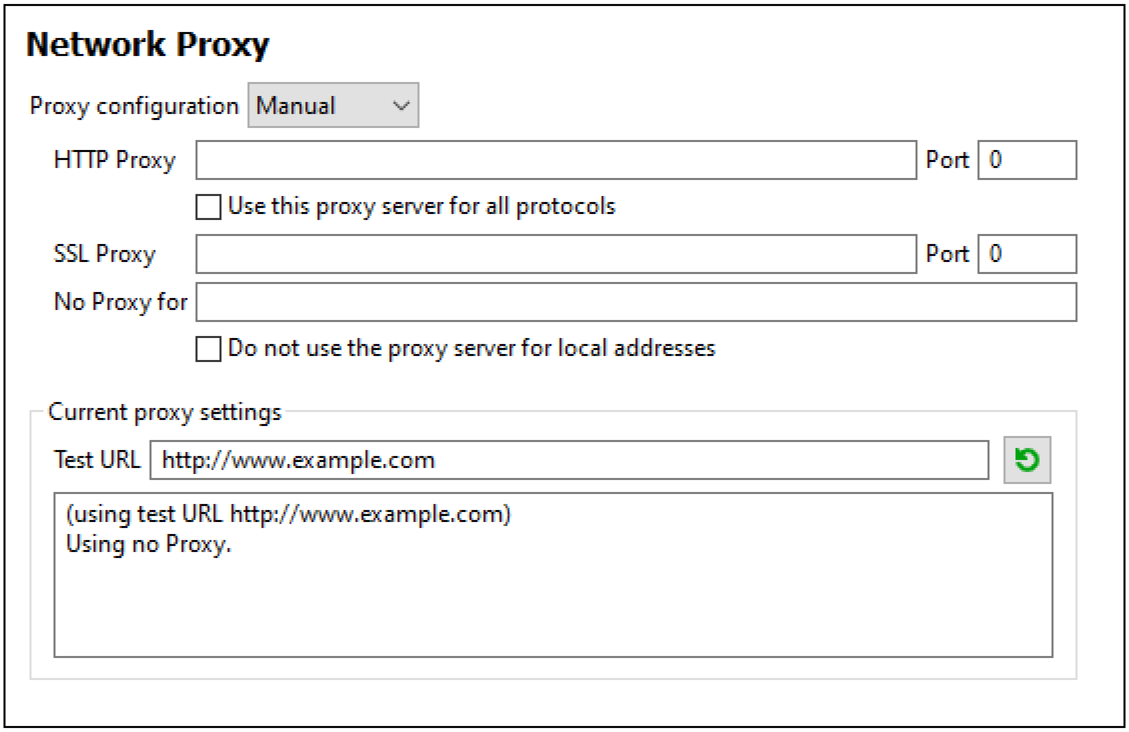


Use system proxy settings
Uses the Internet Explorer (IE) settings configurable via the system proxy settings. It also queries the settings configured with **`netsh.exe winhttp`**.

Automatic proxy configuration
The following options are provided:

- *Auto-detect settings:* Looks up a WPAD script (**`http://wpad.LOCALDOMAIN/wpad.dat`**) via DHCP or DNS, and uses this script for proxy setup.
- *Script URL:* Specify an HTTP URL to a proxy-auto-configuration (**`.pac`**) script that is to be used for proxy setup.
- *Reload:* Resets and reloads the current auto-proxy-configuration. This action requires Windows 8 or newer, and may need up to 30s to take effect.

Manual proxy configuration
Manually specify the fully qualified host name and port for the proxies of the respective protocols. A supported scheme may be included in the host name (for example: **`http://hostname`**). It is not required that the scheme is the same as the respective protocol if the proxy supports the scheme.



The following options are provided:

- *HTTP Proxy:* Uses the specified host name and port for the HTTP protocol. If *Use this proxy server for all protocols* is selected, then the specified HTTP proxy is used for all protocols.
- *SSL Proxy:* Uses the specified host name and port for the SSL protocol.
- *No Proxy for:* A semi-colon (`;`) separated list of fully qualified host names, domain names, or IP addresses for hosts that should be used without a proxy. IP addresses may not be truncated and IPv6 addresses have to be enclosed by square brackets (for example: `[2606:2800:220:1:248:1893:25c8:1946]`). Domain names must start with a leading dot (for example: `.example.com`).
- *Do not use the proxy server for local addresses:* If checked, adds **`<local>`** to the *No Proxy for* list. If this option is selected, then the following will not use the proxy: (i) **`127.0.0.1`**, (ii) **`[::1]`**, (iii) all host names not containing a dot character (`.`).

Note: If a proxy server has been set and you want to deploy a mapping to Altova FlowForce Server, you must select the option *Do not use the proxy server for local addresses*.

Current proxy settings
Provides a verbose log of the proxy detection. It can be refreshed with the **Refresh** button to the right of the *Test URL* field (for example, when changing the test URL, or when the proxy settings have been changed).

- *Test URL:* A test URL can be used to see which proxy is used for that specific URL. No I/O is done with this URL. This field must not be empty if proxy-auto-configuration is used (either through *Use system proxy settings* or *Authomatic proxy configuration*).

15.12 Window

This topic lists all the menu commands available in the **Window** menu.

Cascade

This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

Tile Horizontal

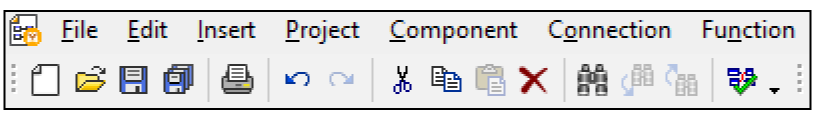
This command rearranges all open document windows as horizontal tiles, making them all visible at the same time.

Tile Vertical

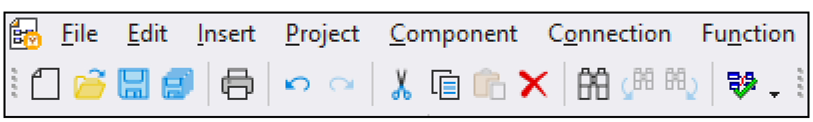
This command rearranges all open document windows as vertical tiles, making them all visible at the same time.

Classic/Light/Dark Theme

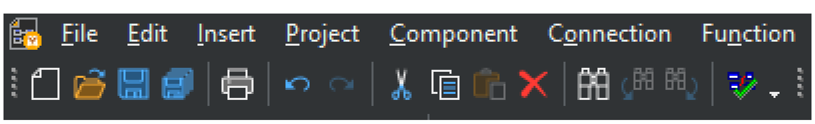
MapForce allows you to choose from the following themes: *Classic*, *Light*, and *Dark*. The examples of these themes are illustrated in the screenshots below. The default option is the Classic theme.



Classic Theme



Light Theme



Dark Theme

1 <MappingName>

Refers to the first open mapping design. If there are more mappings opened at the same time, they will be listed in the context menu, too.

Windows

This list shows all currently open windows and enables you to quickly switch between them. You can also use the **Ctrl-TAB** or **CTRL F6** keyboard shortcuts to switch between the open windows.

15.13 Help

This topic lists all the menu commands available in the **Help** menu.

⊟ Table of Contents

Opens the onscreen help manual of MapForce with the Table of Contents displayed in the left-hand-side pane of the Help window. The Table of Contents provides an overview of the entire Help document. Clicking an entry in the Table of Contents takes you to that topic.

⊟ Index

Opens the onscreen help manual of MapForce with the Keyword Index displayed in the left-hand-side pane of the Help window. The index lists keywords and lets you navigate to a topic by double-clicking the keyword. If a keyword is linked to more than one topic, a list of these topics is displayed.

⊟ Search

Opens the onscreen help manual of MapForce with the Search dialog displayed in the left-hand-side pane of the Help window. To search for a term, enter the term in the input field and press Enter or List Topics. The Help system performs a full-text search on the entire Help documentation and returns a list of hits. Double-click any item to display that item.

⊟ Software Activation

License your product
After you download your Altova product software, you can license—or activate—it using either a free evaluation key or a purchased permanent license key.

- ***Free evaluation license.*** When you first start the software after downloading and installing it, the **Software Activation** dialog will pop up. In it is a button to request a free evaluation license. Enter your name, company, and e-mail address in the dialog and click **Request**. A license file is sent to the e-mail address you entered and should reach you in a few minutes. Save the license file to a suitable location.

  When you clicked **Request**, an entry field appeared at the bottom of the Request dialog. This field takes the path to the license file. Browse for or enter the path to the license file and click **OK**. (In the **Software Activation** dialog, you can also click **Upload a New License** to access a dialog in which the path to the license file is entered.) The software will be unlocked for a period of 30 days.

- ***Permanent license key.*** The **Software Activation** dialog allows you to purchase a permanent license key. Clicking this button takes you to Altova's online shop, where you can purchase a permanent license key for your product. Your license will be sent to you by e-mail in the form of a license file, which contains your license-data.

  There are three types of permanent license: *installed*, *concurrent user*, and *named user*. An installed license unlocks the software on a single computer. If you buy an installed license for *N* computers, then the license allows use of the software on up to *N* computers. A concurrent-user license for *N* concurrent users allows *N* users to run the software concurrently. (The software may be installed on 10N computers.) A named-user license authorizes a specific user to use the software on up to 5 different computers. To activate your software, click **Upload a New License**, and, in the dialog that appears, enter the path to the license file, and click **OK**.

Note: For multi-user licenses, each user will be prompted to enter his or her own name.

Your license email and the different ways to license (activate) your Altova product
The license email that you receive from Altova will contain your license file as an attachment. The license file has a **`.altova_licenses`** file extension.

To activate your Altova product, you can do one of the following:

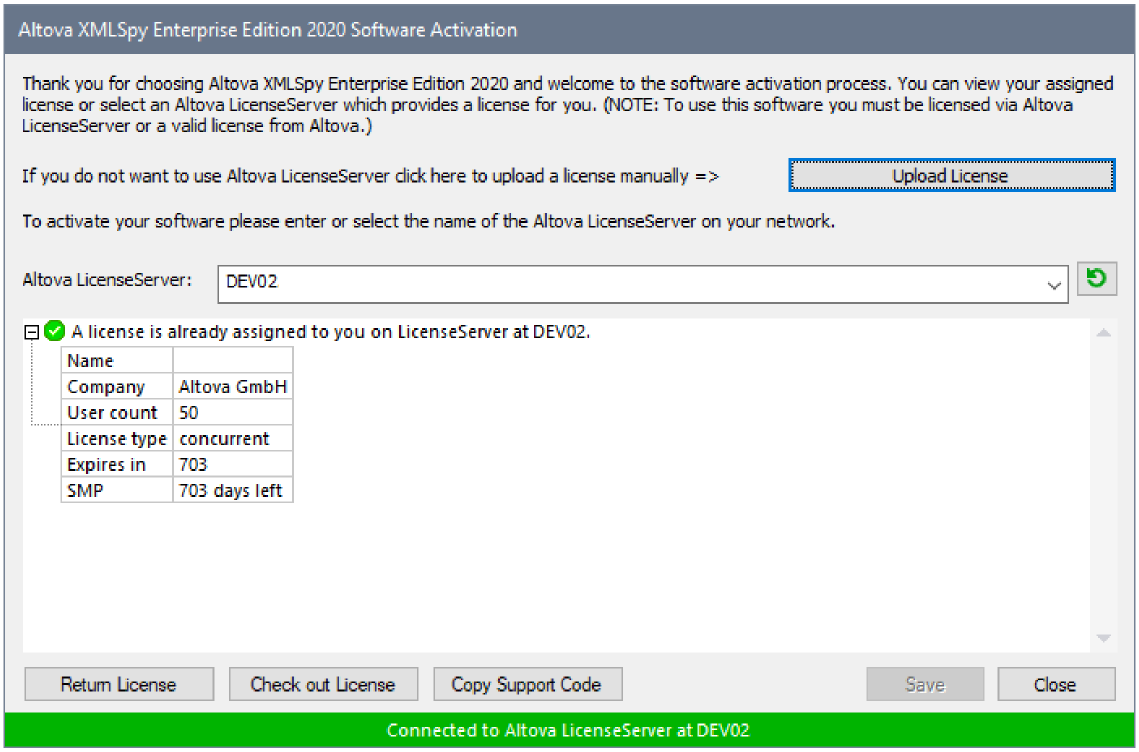
- Save the license file (**`.altova_licenses`**) to a suitable location, double-click the license file, enter any requested details in the dialog that appears, and finish by clicking **Apply Keys**.
- Save the license file (**`.altova_licenses`**) to a suitable location. In your Altova product, select the menu command **Help | Software Activation**, and then **Upload a New License**. Browse for or enter the path to the license file, and click **OK**.
- Save the license file (**`.altova_licenses`**) to any suitable location, and upload it from this location to the license pool of your Altova LicenseServer. You can then either: (i) acquire the license from your Altova product via the product's Software Activation dialog *(see below)* or (ii) assign the license to the product from Altova LicenseServer. *For more information about licensing via LicenseServer, read the rest of this topic.*

You can access the **Software Activation** dialog (*screenshot below*) at any time by clicking the **Help | Software Activation** command.

Activate your software
You can activate the software by registering the license in the Software Activation dialog or by licensing via Altova LicenseServer (*see details below*).

- *Registering the license in the Software Activation dialog*. In the dialog, click **Upload a New License** and browse for the license file. Click **OK** to confirm the path to the license file and to confirm any data you entered (your name in the case of multi-user licenses). Finish by clicking **Save**.

- *Licensing via Altova LicenseServer on your network:* To acquire a license via an Altova LicenseServer on your network, click **Use Altova LicenseServer**, located at the bottom of the **Software Activation** dialog. Select the machine on which the LicenseServer you want to use has been installed. Note that the auto-discovery of License Servers works by means of a broadcast sent out on the LAN. As these broadcasts are limited to a subnet, License Server must be on the same subnet as the client machine for auto-discovery to work. If auto-discovery does not work, then type in the name of the server. The Altova LicenseServer must have a license for your Altova product in its license pool. If a license is available in the LicenseServer pool, this is indicated in the **Software Activation** dialog (*see screenshot below showing the dialog in Altova XMLSpy*). Click **Save** to acquire the license.



After a machine-specific (aka installed) license has been acquired from LicenseServer, it cannot be returned to LicenseServer for a period of seven days. After that time, you can return the machine license to LicenseServer (click **Return License**) so that this license can be acquired from LicenseServer by another client. (A LicenseServer administrator, however, can unassign an acquired license at any time via the administrator's Web UI of LicenseServer.) Note that the returning of licenses applies only to machine-specific licenses, not to concurrent licenses.

Check out license

You can check out a license from the license pool for a period of up to 30 days so that the license is stored on the product machine. This enables you to work offline, which is useful, for example, if you wish to work in an environment where there is no access to your Altova LicenseServer (such as when your Altova product is installed on a laptop and you are traveling). While the license is checked out, LicenseServer displays the license as being in use, and the license cannot be used by any other machine. The license automatically reverts to the checked-in state when the check-out period ends. Alternatively, a checked-out license can be checked in at any time via the **Check in** button of the **Software Activation** dialog.

To check out a license, do the following: (i) In the **Software Activation** dialog, click **Check out License** (*see screenshot above*); (ii) In the **License Check-out** dialog that appears, select the check-out period you want and click **Check out**. The license will be checked out. After checking out a license, two things happen: (i) The **Software Activation** dialog will display the check-out information, including the time when the check-out period ends; (ii) The **Check out License** button in the dialog changes to a **Check In** button. You can check the license in again at any time by clicking **Check In**. Because the license automatically reverts to the checked-in status after the check-out period elapses, make sure that the check-out period you select adequately covers the period during which you will be working offline.

License check-ins must be to the same major version of the Altova product from which the license was checked out. So make sure to check in a license before you upgrade your Altova product to the next major version.

Note: For license check-outs to be possible, the check-out functionality must be enabled on LicenseServer. If this functionality has not been enabled, you will get an error message to this effect when you try to check out. In this event, contact your LicenseServer administrator.

Copy Support Code
Click **Copy Support Code** to copy license details to the clipboard. This is the data that you will need to provide when requesting support via the online support form.

Altova LicenseServer provides IT administrators with a real-time overview of all Altova licenses on a network, together with the details of each license as well as client assignments and client usage of licenses. The advantage of using LicenseServer therefore lies in administrative features it offers for large-volume Altova license management. Altova LicenseServer is available free of cost from the Altova website. For more information about Altova LicenseServer and licensing via Altova LicenseServer, see the Altova LicenseServer documentation.

Order Form

When you are ready to order a licensed version of the software product, you can use either the **Purchase a Permanent License Key** button in the **Software Activation** dialog (*see previous section*) or the **Order Form** command to proceed to the secure Altova Online Shop.

Registration

Opens the Altova Product Registration page in a tab of your browser. Registering your Altova software will help ensure that you are always kept up to date with the latest product information.

Check for Updates

Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly.

Support Center

A link to the Altova Support Center on the Internet. The Support Center provides FAQs, discussion forums where problems are discussed, and access to Altova's technical support staff.

FAQ on the Web

A link to Altova's FAQ database on the Internet. The FAQ database is constantly updated as Altova support staff encounter new issues raised by customers.

Download Components and Free Tools

A link to Altova's Component Download Center on the Internet. From here you can download a variety of companion software to use with Altova products. Such software ranges from XSLT and XSL-FO processors to Application Server Platforms. The software available at the Component Download Center is typically free of charge.

MapForce on the Internet

A link to the Altova website on the Internet. You can learn more about MapForce, related technologies and products on the Altova website.

MapForce Training

A link to the Online Training page on the Altova website. Here you can select from online courses conducted by Altova's expert trainers.

About MapForce

Displays the splash window and version number of your product. If you are using the 64-bit version of MapForce, this is indicated with the suffix (x64) after the application name. There is no suffix for the 32-bit version.

16 Code Generator

Code Generator is a MapForce built-in feature which enables you to generate Java, C++ or C# code from mapping files designed with MapForce. You can generate code not only from simple mappings with a single data source and target, but also from mappings with multiple sources and multiple targets. The result is a fully-featured and complete application which performs the mapping operation for you. Once you generate the code, you can execute the mapping by running the application directly as generated. You can also import the generated code into your own application, or extend it with your own functionality.

The primary goal of the generated code is to execute a MapForce mapping. If your mapping uses XML schemas or DTDs, you can optionally generate schema wrapper libraries for XML/DTD schemas used by the mapping, which enables you to read or write data to or from XML instances. The schema wrapper libraries enable you to work with XML data programmatically, using types generated from the schema, without too much concern for the underlying XML API. For details, see Generating Code from XML Schemas or DTDs 1535.

The generated code is expressed in C++, Java or C# programming languages.

Target Language	C++	C#	Java
Development environments	Microsoft Visual Studio 2013, 2015, 2017, 2019, 2022	Microsoft Visual Studio 2013, 2015, 2017, 2019, 2022 Target frameworks: • .NET Framework • .NET Core 3.1 • .NET 5.0 • .NET 6.0	Java SE JDK 8 or 11 (including OpenJDK) Eclipse 4.4 or later Apache Ant (build.xml file)
XML DOM implementations	MSXML 6.0 Apache Xerces 3	System.Xml	JAXP
Database API	ADO	ADO.NET	JDBC

Note: The MapForce-generated code can be considered thread-safe only if the underlying third-party XML DOM and database API libraries are. Although the thread safety of the generated code cannot be realistically proven or guaranteed, it is likely that multiple concurrent instances of the mapping code will run successfully in most cases.

C++

You can configure whether the C++ generated output should use MSXML 6.0 or Apache Xerces 3. MapForce generates complete project (.vcproj) and solution (.sln) files for all supported versions of Visual Studio (see table above). The generated code optionally supports MFC.

Prerequisites:

1. To compile the generated C++ code, Windows SDK must be installed on your computer.

2. To use Xerces 3 for C++, you will need to install and build it by following instructions from http://xerces.apache.org/xerces-c/. Make sure to add the XERCES3 environment variable that points to the directory where Xerces was installed, for example: `C:\xerces-c-3.2.2`. Also, the PATH environment variable must include the path where the Xerces binaries are, for example: `%XERCES3%\bin`.
3. When building C++ code for Visual Studio and using a Xerces library precompiled for Visual C++, a compiler setting has to be changed in all projects of the solution:

   a) Select all projects in the Solution Explorer.
   b) On the **Project** menu, click **Properties**.
   c) Click **Configuration Properties | C/C++ | Language**.
   d) In the list of configurations, select *All Configurations*.
   e) Change **Treat wchar_t as Built-in Type** to **No (/Zc:wchar_t-)**.

C#

The generated C# code can be used from any .NET capable programming language, such as VB.NET, Managed C++, or J#. Project files can be generated for all supported versions of Visual Studio (see table above).

Java

The generated Java output is written against the Java API for XML Processing (JAXP) and includes an Ant build file and project files for supported versions of Java and Eclipse (see table above).

Generated output

The designated destination folder will include all the libraries and files required to execute the mapping, namely:

- A variable number of Altova libraries required by the mapping (for example, Altova function libraries, database libraries, EDI libraries)
- A complete mapping application. When compiled and run, the application performs the mapping transformation.

Code generator templates

The generated code supports customization via a template language 1627 called SPL (Spy Programming Language). This can be useful, for example, when you want to customize code according to your company's writing conventions, or replace specific libraries in the generated code.

16.1 What's new ...

Version 2022 R2

- Support for Visual Studio 2022.
- Support for .NET 6.0.

Version 2021 R2

- When generating C# code, you can select .NET Core 3.1, .NET 5.0, or .NET 6.0 as target frameworks from code generation options 1625 (this adds to existing support for .NET Framework projects).

Version 2020 R2

- Code generated for XML schema wrapper libraries now provides more control over element namespaces and prefixes. New methods are available to declare or override namespaces for an element, or to append an element with a prefixed namespace. See Example: Purchase Order 1571.

Version 2020

- Added support for Visual Studio 2019
- End of support for Visual Studio 2008

Version 2018

- Added support for Microsoft Visual Studio 2013, 2015, 2017
- End of support for Visual Studio 2005 and Xerces 2.x

Version 2014

- Removal of compatibility mode option for code generation

Version 2011

- Contains bug fixes and enhancements

Version 2010 R3

- Support for Microsoft Visual Studio 2010
- Support for MSXML 6.0 in generated C++ code
- Support for 64-bit targets for C++ and C# projects

Version 2010

- Enumeration facets from XML schemas are now available as symbolic constants in the generated classes (using 2007r3 templates)

Version 2009 sp1

- Apache Xerces version 3.x support added (older versions starting from Xerces 2.6.x are still supported)

Version 2009

- The generated mapping implementation was redesigned to support sequences and grouping. The API has not changed.

Version 2008 R2

- Support for generation of Visual Studio 2008 project files for C# and C++ has been added
- Generated MapForce mapping code in C# and Java can use readers/writers, streams, strings or DOM documents as sources and targets

Version 2008

- The new 2007 R3-style SPL templates have been further enhanced:
    - It is now possible to remove single elements
    - Access to schema metadata (e.g. element names, facets, enumerations, occurrence, etc.) is provided
    - Complex types derived by extension are now generated as derived classes

Version 2007 R3

Code Generator has been redesigned for version 2007 release 3 to simplify usage of the generated code, reduce code volume and increase performance.

- Handling of XML documents and nodes with explicit ownership, to avoid memory leaks and to enable multi-threading
- New syntax to avoid name collisions
- New data types for simpler usage and higher performance (native types where possible, new null handling, ...)
- Attributes are no longer generated as collections
- Simple element content is now also treated like a special attribute, for consistency
- New internal object model (important for customized SPL templates)
- Compatibility mode to generate code in the style of older releases
- Type wrapper classes are now only generated on demand for smaller code

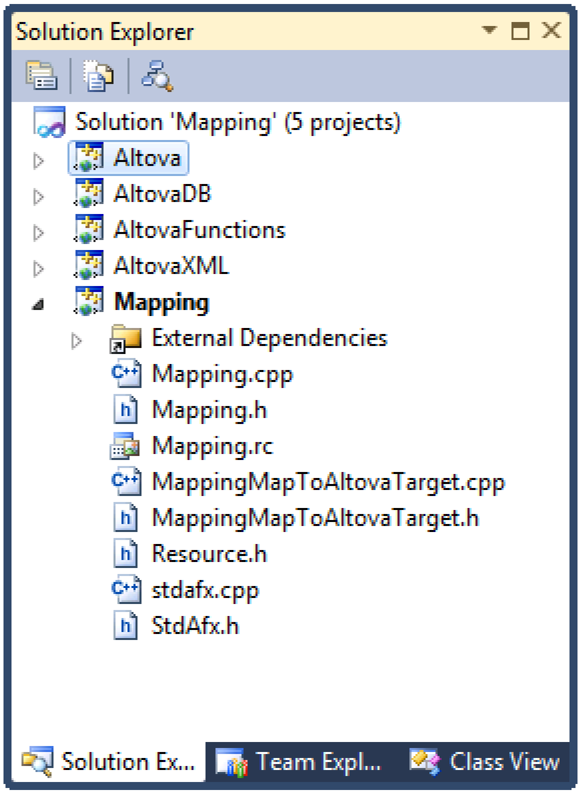
16.2 Generating C++ code

You can generate C++ code for Visual Studio 2013, 2015, 2017, 2019, 2022. The generated code includes **.sln** and .**vcproj** files for Visual Studio. Note the following when generating code:

- Certain MapForce features are not supported in generated program code. For details, see Supported features in generated code 1891.
- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the **Tools | Options** menu, **Generation** tab. See Code Generator Options 1625.
- You can change the name of the generated mapping application and other settings from the **File | Mapping settings** menu. The default application name is **Mapping**. See also Changing the Mapping Settings 114.
- For information about handling paths in generated code, see Paths in Various Execution Environments 87.
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option **Component | Properties**. See Database Component Settings 279.
- Support for database connections varies by platform and there are connection kinds that are not supported on all platforms. If your mapping connects to a database, choose a database connection that is compatible with the target environment for which you generate code. For details, see Database mappings in various execution environments 162.

A typical C++ solution generated by MapForce includes the following:

- Several Altova-signed libraries required by the mapping (all prefixed with **Altova**).
- The main mapping project (in this example, **Mapping**), which includes the mapping application and dependent files.



Sample C++ solution generated with MapForce

This section includes the following topics:

- Generating code from a mapping [1497]
- Generating code from a mapping project [1498]
- Building the project [1498]
- Running the application [1499]

16.2.1 Generating code from a mapping

To generate C++ code from a mapping design file (.mfd):

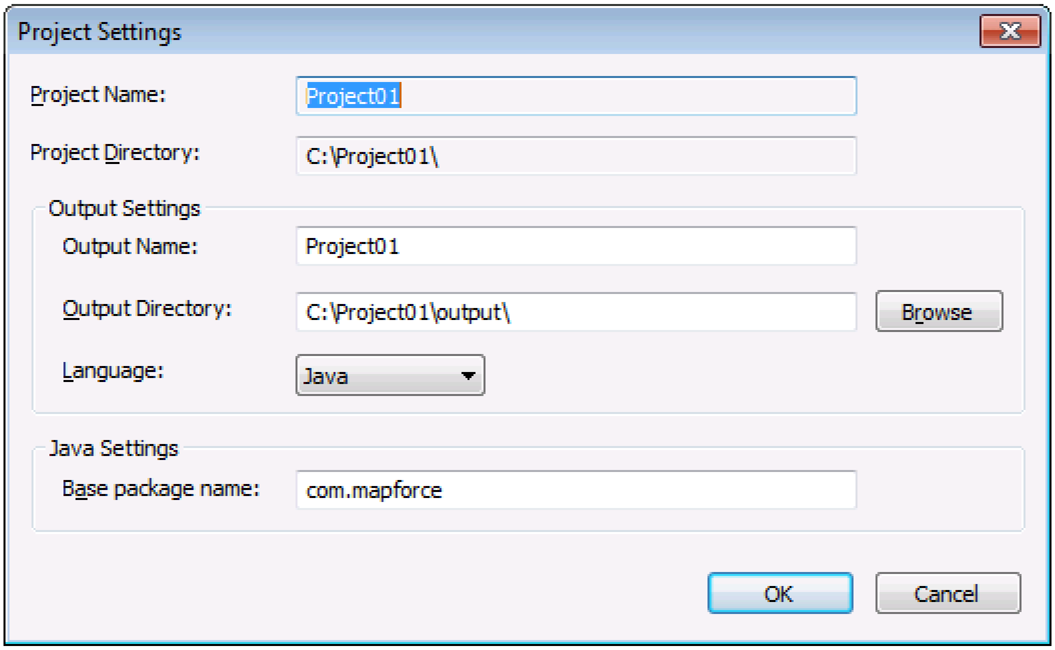
1. Review and select the code generator options [1625].
2. On the **File** menu, click **Generate code in | C++**.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed in the Messages window.

The default name of the generated application is **Mapping**. If required, you can change this, and other settings, from the Mapping Settings [114] dialog box.

16.2.2 Generating code from a mapping project

To generate code from a mapping project (.mfp):

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click **Properties**.



3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click **OK**.
4. On the **Project menu, click Generate code for the Entire Project**.

The progress and result of the code generation process (error or success message) is displayed in the Messages window.

By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the **output** sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select **Project | Properties** from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select **Properties**). Otherwise, all project folders inherit the settings from the MapForce project.

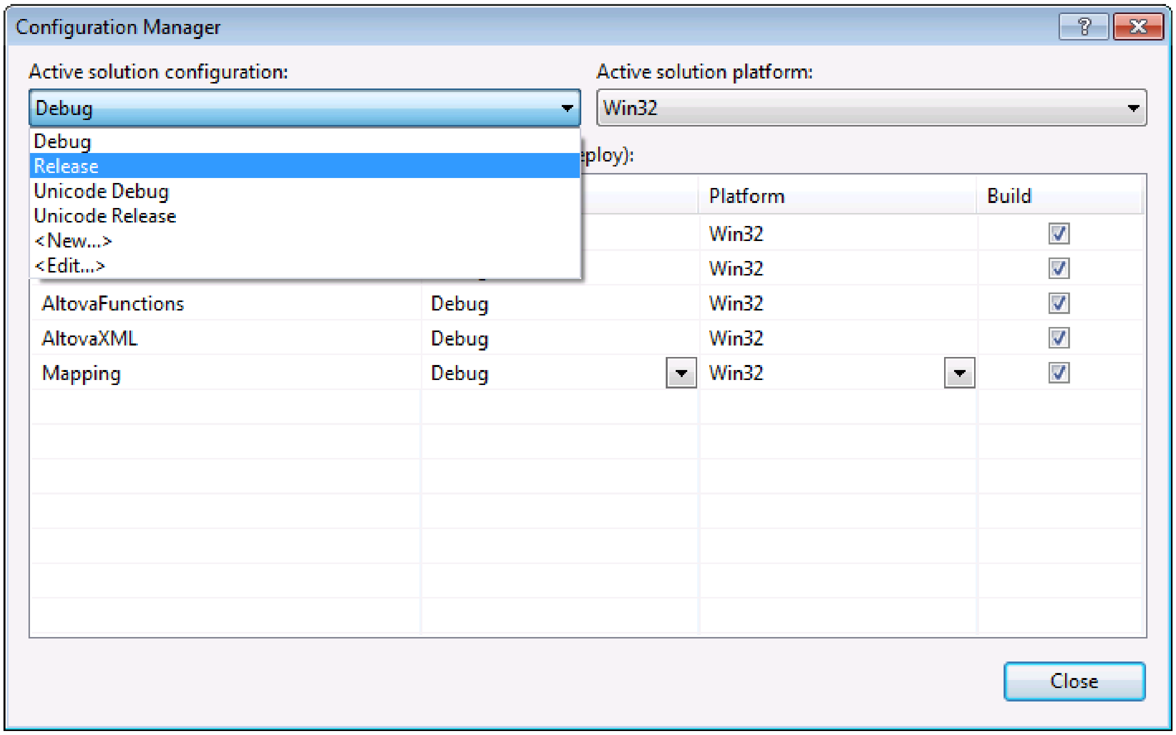
16.2.3 Building the project

Once you generated the C++ code, building it in Visual Studio is the next step. To build the generated code:

1. Open the generated solution (.sln) file in Visual Studio.

By default, the name of the solution file is **Mapping.sln**, and it is located in the **Mapping** subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to **MyApplication**, then the solution file is called **MyApplication.sln**, and it is located in the **MyApplication** subdirectory.

2. On the **Build** menu, click **Configuration Manager**.



3. Select the required build configuration (Debug, Release, Unicode Debug, Unicode Release). Note that only Unicode builds support the full Unicode character set in XML and other files. The non-Unicode builds work with the local codepage of your Windows installation.
4. On the **Build** menu, click **Build Solution**.

16.2.4 Running the application

Once you compile the Visual Studio project, a command-line application is produced, called **Mapping.exe**. (Note that if you changed the application name from the mapping settings, then the executable name is changed accordingly.)

You can locate the mapping application in one of the following subdirectories relative to the .sln file, depending on the build option you chose:

- Debug
- Release
- Unicode Debug
- Unicode Release

To run the application, open a command prompt, change the current directory to the path of the executable, and run it, for example:

```
C:\codegen\DB_CompletePOcpp\Mapping\Debug>Mapping.exe
Mapping Application
Finished

C:\codegen\DB_CompletePOcpp\Mapping\Debug>
```

16.3 Generating C# code

You can generate .NET C# code for any mapping or mapping project where all the mapping components are supported in C#. For an overview of components supported in each language, including C#, see Supported features in generated code 1891. Note that MapForce typically issues notification or warning messages whenever you attempt to add components that are not supported in a specific language.

When generating C# code, you can target not only the .NET Framework platform but also the .NET Core 3.1, .NET 5.0, and .NET 6.0 platforms. If you choose the latter, the result is a .NET (Core) cross-platform desktop application that can run on Windows, macOS, and Linux.

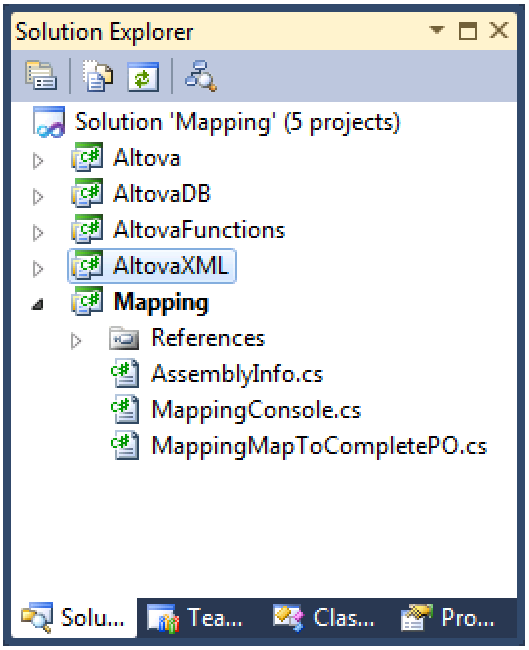
Note the following when generating code:

- Certain MapForce features are not supported in generated program code. For details, see Supported features in generated code 1891.
- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the **Tools | Options** menu, **Generation** tab. See Code Generator Options 1625.
- You can change the name of the generated mapping application and other settings from the **File | Mapping settings** menu. The default application name is **Mapping**. See also Changing the Mapping Settings 114.
- For information about handling paths in generated code, see Paths in Various Execution Environments 87.
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option **Component | Properties**. See Database Component Settings 279.
- Support for database connections varies by platform and there are connection kinds that are not supported on all platforms. If your mapping connects to a database, choose a database connection that is compatible with the target environment for which you generate code. For details, see Database mappings in various execution environments 162.

> If your mapping uses database components, make sure to create the database connection using database drivers compatible with the platform on which you intend to run the generated program. For example, ADO database connections and some ODBC drivers are not supported on Linux.

A typical C# solution generated by MapForce includes the following:

- Solution (.sln) and project (.csproj) files that can be opened in Visual Studio.
- Several Altova-signed libraries required by the mapping (all prefixed with **Altova**).
- The main mapping project (in this example, **Mapping**), which includes the mapping application and dependent files.

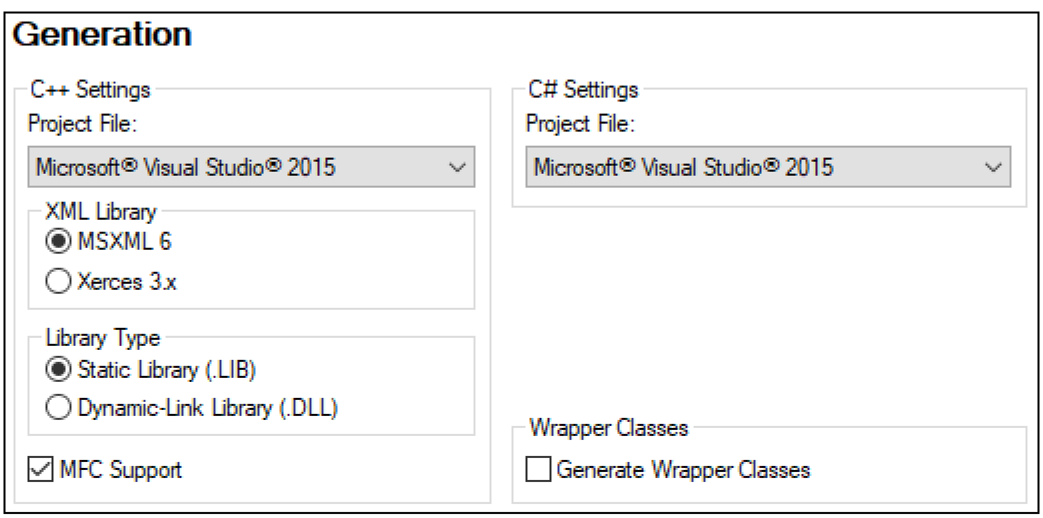


Sample C# solution generated with MapForce

16.3.1 Selecting the target platform

Before generating C# code, you can set the target platform at program level, as follows:

1. On the **Tools** menu, click **Options**.
2. Click the **Generation** tab.
3. In the "C# settings" group, select a value from the drop-down list, as applicable.



Select the option **Microsoft .NET Core 3.1**, **Microsoft .NET 5.0**, or **Microsoft .NET 6.0** to generate a Visual Studio solution targeting the respective platforms. If you need to target the **.NET Framework** platform for a specific Visual Studio version, select any of the **Microsoft Visual Studio 2010-2019** options—in this case, the generated solution will target the .NET Framework version corresponding to the respective Visual Studio version.

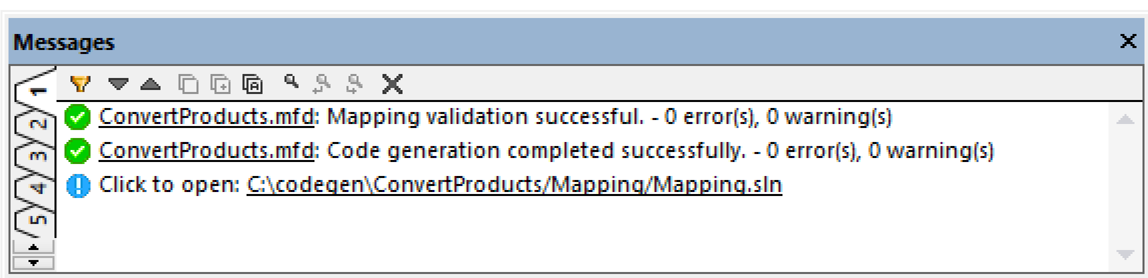
The **Generate Wrapper Classes** check box, if selected, creates additional projects in the generated code, one for each XML schema or DTD component present on the mapping. The generated wrapper projects enable you to manipulate programmatically the respective XML instances outside of the mapping. The wrapper libraries are optional and independent of the mapping program, although you can use them from the mapping program if necessary. For more information about wrapper classes, see Generating Code from XML Schemas or DTDs[1535].

16.3.2 Generating code from a mapping

To generate C# code from a mapping design file (.mfd):

1. If you haven't done so already, create a new mapping or open an existing one in MapForce.
2. Select C# as mapping transformation language. To do this, either click the C# toolbar button or select the menu command **Output | C# (Sharp)**.
3. Set or change the C# code generation options as described in Selecting the target platform[1502].
4. Select the menu command **File | Generate code in | C# (Sharp)**. You are now prompted to select a target directory.
5. Browse for the directory of your choice (for example, "C:\codegen\ConvertProducts") and click **OK**.

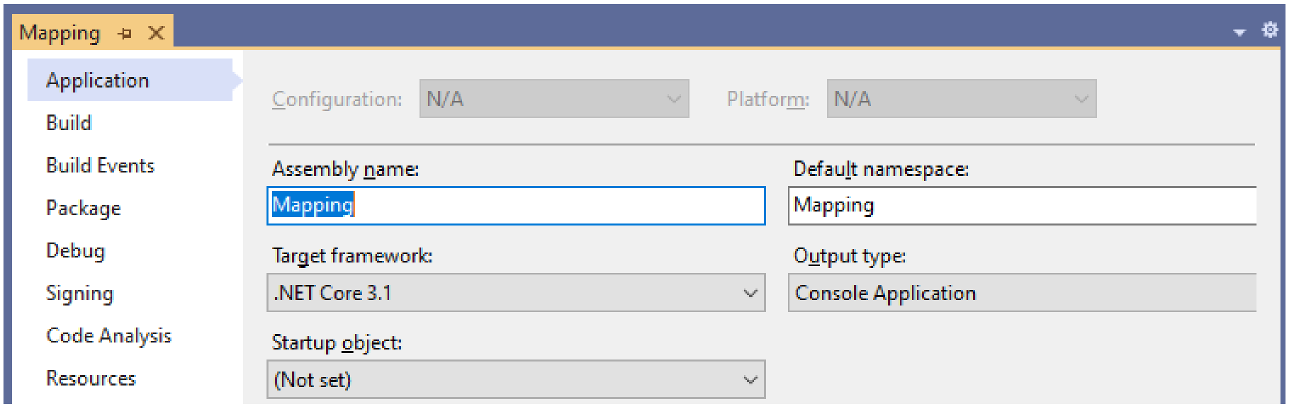
The code generation outcome is displayed in the Messages window, for example:



Optionally, click the link in the Messages window to open the generated solution in Visual Studio.

Note that, if you chose to generate code targeting .NET 5.0, .NET 6.0, or .NET Core platforms, you need the respective SDKs and a compatible Visual Studio version. For the download package applicable to your operating system and platform, refer to the Microsoft website (https://dotnet.microsoft.com/download).

To double-check the project's target framework from Visual Studio, right-click the "Mapping" project in the Solution Explorer, select **Properties**, and observe the value displayed in the **Target Framework** drop-down list, for example:



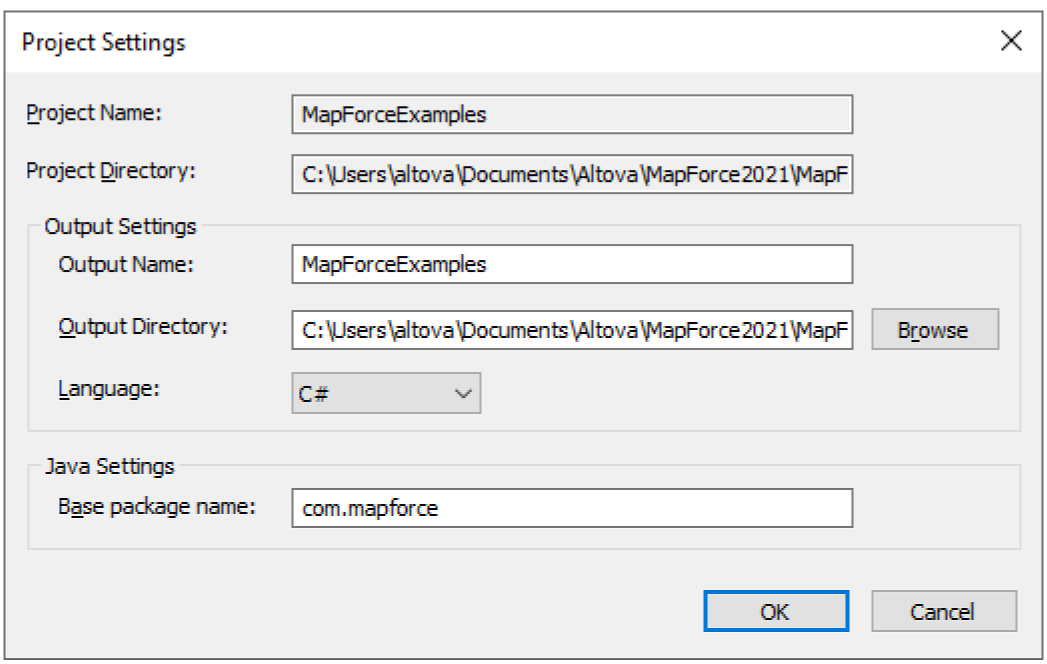
The default name of the generated application is **Mapping**. Optionally, you can change this and other settings before generating code, from the Mapping Settings (114) dialog box.

16.3.3 Generating code from a mapping project

You can generate C# code from a mapping project (.mfp) that consists of multiple mapping design (.mfd) files. Note that all mapping design files in the project must qualify for C# generation, that is, all their components must be supported in C#, as described in Supported features in generated code (1891).

To generate code from a mapping project (.mfp):

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click **Properties**.



3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click **OK**.
4. On the **Project** menu, click **Generate code for the Entire Project**.

The progress and result of the code generation process (error or success message) is displayed in the Messages window.

By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the **output** sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select **Project | Properties** from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select **Properties**). Otherwise, all project folders inherit the settings from the MapForce project.

16.3.4 Building .NET Framework projects

If your generated C# targets the .NET Framework platform, the easiest way to build the project is as follows:

1. Open the generated solution (.sln) file in Visual Studio.

By default, the name of the solution file is **Mapping.sln**, and it is located in the **Mapping** subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to **MyApplication**, then the solution file is called **MyApplication.sln**, and it is located in the **MyApplication** subdirectory.

2. On the **Build** menu, click **Configuration Manager**.
3. Select the required build configuration (Debug, Release).
4. On the **Build** menu, click **Build Solution**.

Running the application

Once you compile the Visual Studio project, a command-line application is produced, called **Mapping.exe**. Note that if you changed the application name from the Mapping Settings 114 dialog box, then the executable name is changed accordingly.

You can find the mapping application in one of the following subdirectories relative to the .sln file, depending on the build option you chose earlier:

- bin\Debug
- bin\Release

To run the application, open a command prompt, change the current directory to the path of the executable, and run it, for example:

```
C:\codegen\DB_CompletePOcs\Mapping\bin\Release>Mapping.exe
Mapping Application
Connecting to CustomersAndArticles database...

Finished

C:\codegen\DB_CompletePOcs\Mapping\bin\Release>
```

16.3.5 Building .NET and .NET Core projects

To build and run the generated .NET Core 3.1, .NET 5.0, or .NET 6.0 solution, you must have the respective SDK installed. For the download package applicable to your operating system and platform, refer to the Microsoft website (https://dotnet.microsoft.com/download). After downloading and installing the required package, yon can easily view the installed SDKs by entering the following command at the command prompt:

```
dotnet --list-sdks
```

If you have the .NET Core 3.1, .NET 5.0, or .NET 6.0 SDK, you can proceed to building the generated code, either at the command line or in Visual Studio. The .NET Core 3.1, .NET 5.0, .NET 6.0 SDKs are compatible with Visual Studio 2019 (v16.7).

Building at the command line

To build the generated code at the command line:

1. Open a command prompt window and change directory to the one where the mapping code was generated. For example, if you selected **C:\codegen\ConvertProducts** as target directory and the default mapping settings, run:

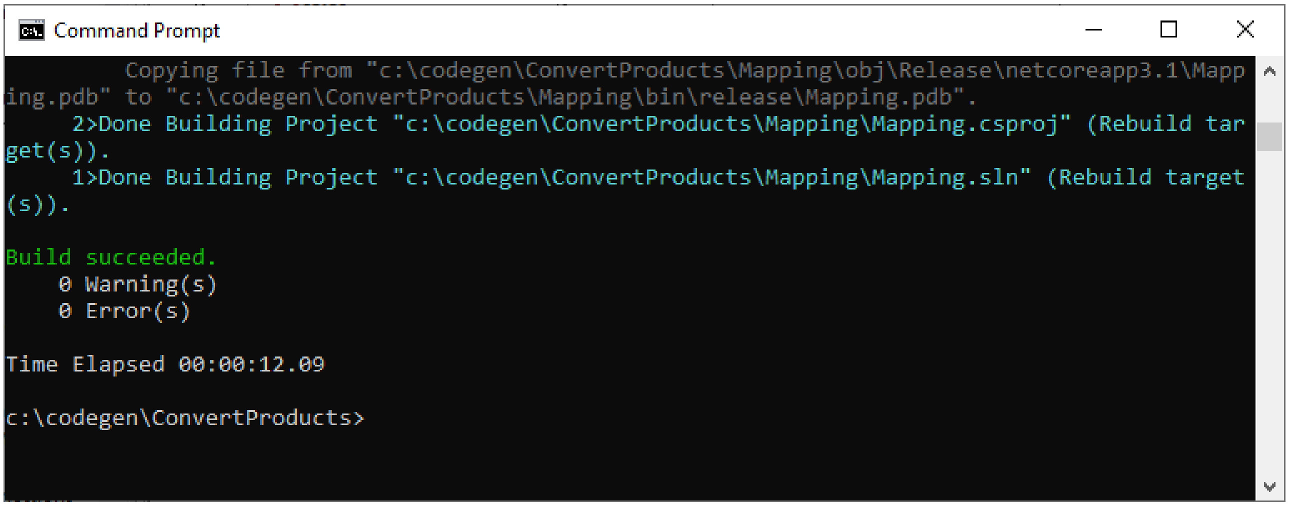
   ```
   cd C:\codegen\ConvertProducts
   ```

2. Run the following command:

   ```
   dotnet build Mapping\Mapping.sln --no-incremental --configuration Release --
   verbosity normal --output Mapping\bin\release
   ```

Note: The path to the mapping solution and its name may be different if you have changed the **Application Name** property from the Mapping Settings 114 dialog box.

The outcome for a successful build could be similar to the one below:



Building with Visual Studio

To build the generated code in Visual Studio:

1. Double-click **Mapping.sln** (or the respective custom solution name) to open it in Visual Studio.
2. On the **Build** menu, click **Configuration Manager**.
3. Select the required build configuration (Debug, Release) and solution platform (x64, x86, Any CPU).
4. On the **Build** menu, select **Build Solution (Ctrl+Shift+B)**.

Running the generated program

After you build the generated code, a **Mapping.exe** (or a program with a custom name if configured) is produced in the project's output directory. The path of the output directory depends on the build configuration you have chosen earlier (Debug, Release).

If you build the generated code on Linux, the generated executable will be called simply **Mapping**, without any extension.

On Windows, you can execute the program by double-clicking the executable in Windows Explorer or by calling the executable from a command prompt window. On other platforms, you will need to call the executable from the terminal with a command like:

```
./Mapping
```

Note that, if the mapping has references to any input or output files on the disk, the program will execute successfully only if all the paths can be resolved. For example, if you configured the mapping so that a component should look for some input file in the same directory using a relative path, then the respective file must be present in the program's execution directory. For information about handling paths at mapping design time, see Using Relative and Absolute Paths [84].

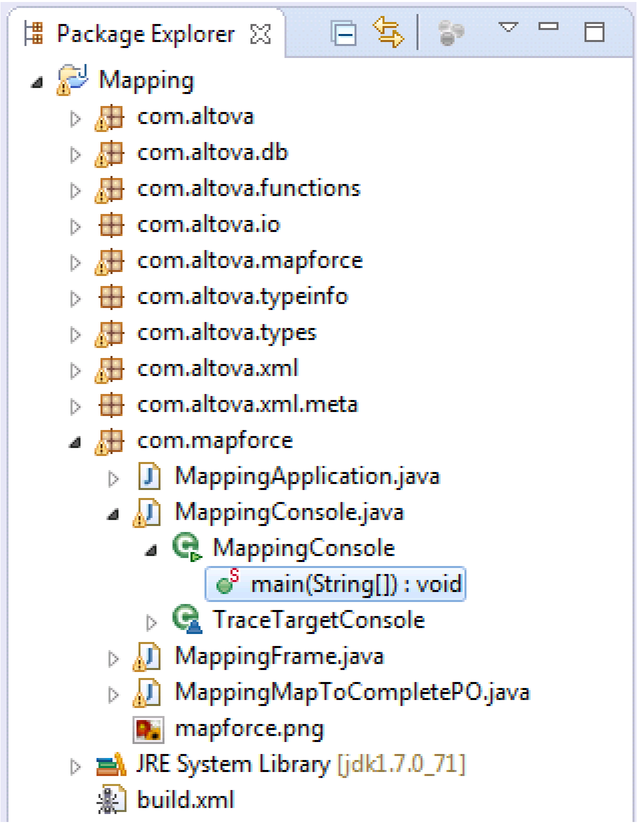
16.4 Generating Java code

You can generate program code for Java SE JDK 8 or 11 (including OpenJDK). Note the following when generating code:

- Certain MapForce features are not supported in generated program code. For details, see Supported features in generated code [1891].
- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the **Tools | Options** menu, **Generation** tab. See Code Generator Options [1625].
- You can change the name of the generated mapping application and other settings from the **File | Mapping settings** menu. The default application name is **Mapping**. See also Changing the Mapping Settings [114].
- For information about handling paths in generated code, see Paths in Various Execution Environments [87].
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option **Component | Properties**. See Database Component Settings [279].
- Support for database connections varies by platform and there are connection kinds that are not supported on all platforms. If your mapping connects to a database, choose a database connection that is compatible with the target environment for which you generate code. For details, see Database mappings in various execution environments [162].

A typical Java project generated by MapForce includes the following:

- Several Altova-signed Java packages required by the mapping (all prefixed with **com.altova**).
- The **com.mapforce** package, which includes the mapping application and dependent files (as shown below, it is possible to change the name of this package). The two most important files in this package are as follows:
  - o The Java mapping application as a dialog application (**MappingApplication.java**).
  - o The Java mapping application as a console application (**MappingConsole.java**).
- A **build.xml** file which you can execute with Apache Ant to compile the project and generate JAR files.



Sample MapForce-generated Java application (Eclipse IDE)

This section includes the following topics:

- Generating code from a mapping[1509]
- Generating code from a mapping project[1510]
- Handling JDBC references[1512]
- Building the project with Ant[1510]
- Example: Generate and Run Java Code[1514]

16.4.1 Generating code from a mapping

To generate Java code from a mapping design file (.mfd):

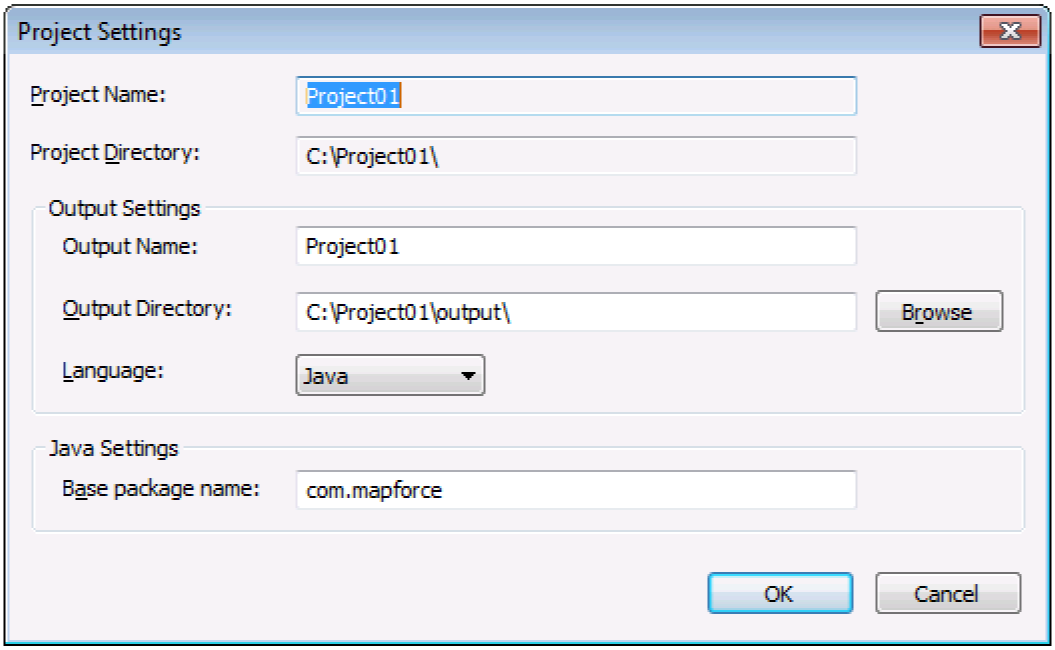
1. Review and select the code generator options[1625].
2. On the **File** menu, click **Generate code in | Java**.
3. Select a destination directory for the generated files, and then click **OK** to confirm. The result of code generation (error or success message) is displayed in the Messages window.

The default name of the generated application is **Mapping**, and the default name of the base package is **com.mapforce**. If required, you can change these from the Mapping Settings[114] dialog box.

16.4.2 Generating code from a mapping project

To generate code from a mapping project (.mfp):

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click **Properties**.



3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click **OK**.
4. On the **Project menu, click Generate code for the Entire Project**.

The progress and result of the code generation process (error or success message) is displayed in the Messages window.

By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the **output** sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select **Project | Properties** from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select **Properties**). Otherwise, all project folders inherit the settings from the MapForce project.

16.4.3 Building the project with Ant

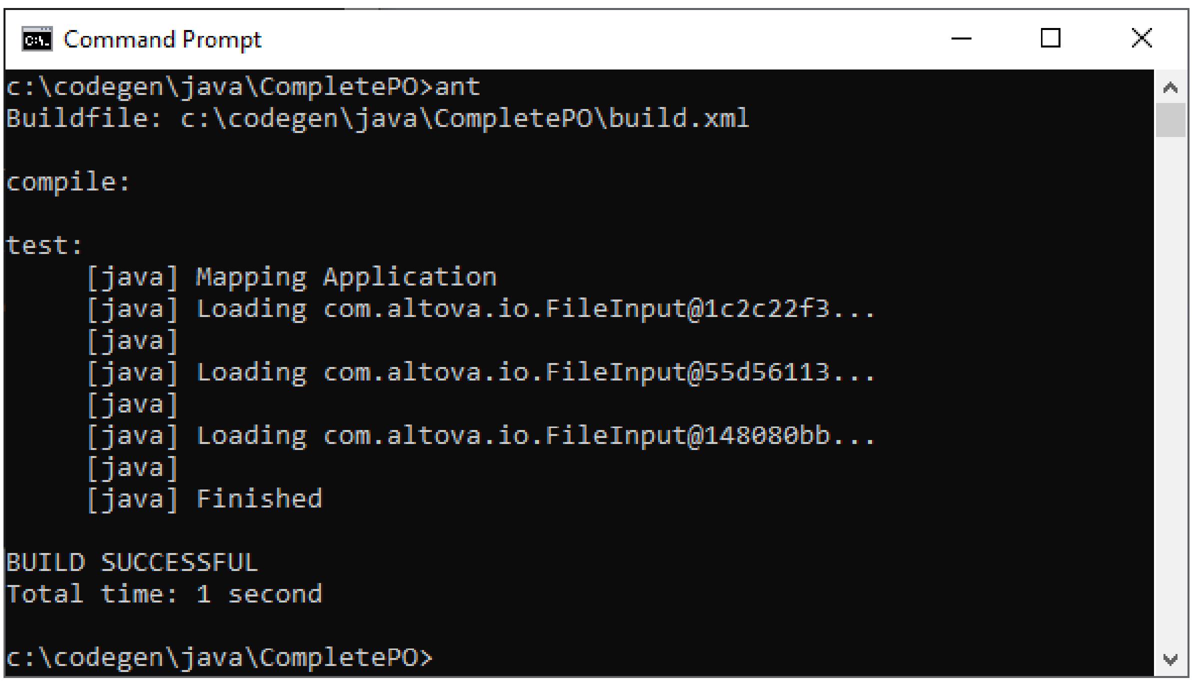
Apache Ant is an open source tool which automates building and compilation of Java projects (see http://ant.apache.org/manual/). Ant works with build files (such files define the sources and targets from which code must be compiled, as well as any specific build options). Since any MapForce-generated project includes a **build.xml** file recognized by Ant, you can easily build MapForce-generated projects with Ant.

Ant may be available on your system either as a standalone installation, or bundled within Eclipse (or other Java IDEs). For instructions on how to install Ant on your system, see http://ant.apache.org/manual/. For instructions on how to use Ant in Eclipse, refer to the Eclipse documentation.

You can quickly check whether the standalone version of Ant (not the one bundled with Eclipse) is available on your system by opening a command prompt and typing `ant` at the command line. When Ant is already available, the resulting message will be similar to: `Buildfile: buildxml does not exist!` This message indicates that Ant is installed and it is attempting to build a **build.xml** file, but the latter does not exist in the current directory. If you run Ant from a directory which includes a **build.xml** file, Ant executes the **build.xml** file instead, with whatever build options are defined in it.

To build a MapForce-generated Java project with Ant:

1. Open a command prompt and navigate to the directory where the Java project was generated (note that the directory must contain the **build.xml** file).
2. At the command prompt, enter `ant`. This will compile and execute the Java code according to the default options defined in the **build.xml** file, for example:



To generate a JAR file with Ant:

- At the command prompt, enter `ant jar`.

For help with Ant command syntax and options, enter `ant -help` at the command line. For a demo, see [Example: Generate and Run Java Code](1514).

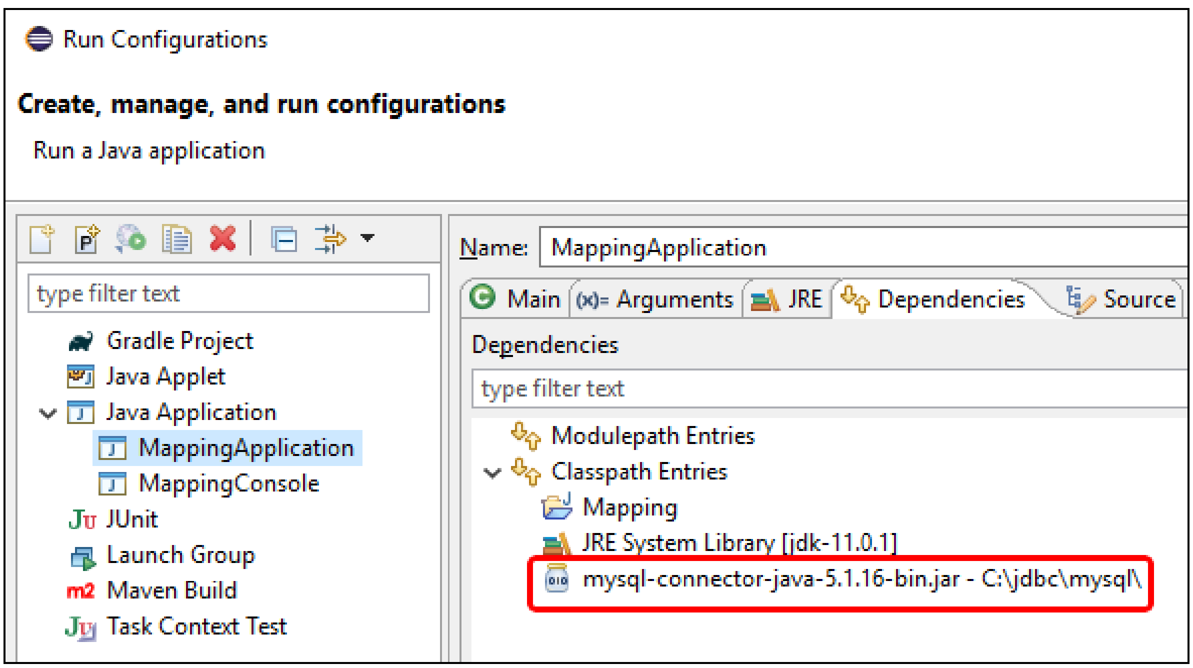
16.4.4 Handling JDBC references

If the mapping connects to a database through JDBC, ensure that the JDBC drivers used by the mapping are installed on your system. To view the current JDBC settings of any database component in MapForce, click it, and then select **Component | Properties** from the menu. For more information, see Creating a JDBC connection[185].

> If the mapping uses a non-JDBC database connection, the connection may be converted to JDBC during Java code generation, to provide compatibility in a Java environment. For example, ADO, ADO.NET, and ODBC connections are converted to JDBC connections when you generate Java code from the mapping. For details, see Database mappings in various execution environments[162].

If you test run the generated Java application in Eclipse, the JDBC driver may need to be added as a "classpath" entry to the current run configuration. Otherwise, test running the application in Eclipse could fail with an error message such as: `java.lang.ClassNotFoundException: com.mysql.jdbc.Driver`. Taking MySQL database as an example, you can add the JDBC driver as a dependency in Eclipse as follows:

1. Generate the Java code from MapForce and import the project into Eclipse using the menu command **File | Import | Existing Projects into Workspace**.
2. On the **Run** menu, click **Run Configurations**. On the left side of the dialog box, under "Java Application", there are two configurations called **MappingApplication** and **MappingConsole**.
3. Click the desired configuration (for example, **MappingApplication**).
4. On the **Dependencies** tab, click **Classpath entries**, and then click **Add External JARs**.
5. Browse for the .jar file of the JDBC driver, for example:



6. Click **Run** to run the program with the database JDBC driver added as a dependency.

If you get the JDBC driver error above when test running the Ant **build.xml** file, add the JDBC driver to the classpath of the "test" task. The following is an example of an Ant "test" task that includes a reference to the .jar file of the JDBC driver.

```
<target name="test" depends="compile">
   <java classpath="C:\codegen\java\mysql_mapping"
classname="com.mapforce.MappingConsole" fork="true" failonerror="true">
      <classpath>
         <pathelement path="${classpath}"/>
         <pathelement location="C:\jdbc\mysql\mysql-connector-java-5.1.16-bin.jar"/>
      </classpath>
      <arg line="${cmdline}"/>
   </java>
</target>
```

Note: The example above assumes that the .jar file exists at the following path: **C:\jdbc\mysql\mysql-connector-java-5.1.16-bin.jar**. Make sure to change the .jar file path as applicable to your environment.

Including JDBC drivers in the application's manifest

If you build JAR files from the generated Java application, add a reference to the database driver in the "manifest" section of the **build.xml** file. This ensures that the reference to the database driver is available in the manifest (**MANIFEST.MF**) file after you build the project.

To add the database reference to the manifest file:

1. Locate the **manifest** element in the **build.xml** file,
2. Add a new element called **attribute** where the "name" attribute is "Class-Path" and the "value" attribute is the name of the .jar file. For example, for MySQL 5.1.16, the new element could look as follows:

```
<attribute name="Class-Path" value="mysql-connector-java-5.1.16-bin.jar"/>
```

The **manifest** element of the **build.xml** file now looks as follows:

```
<manifest file="C:\codegen\java\mysql_mapping/META-INF/MANIFEST.MF"
mode="replace">
   <attribute name="Created-By" value="MapForce 2023"/>
   <attribute name="Main-Class" value="com.mapforce.MappingConsole"/>
   <attribute name="Class-Path" value="mysql-connector-java-5.1.16-bin.jar"/>
</manifest>
```

3. Copy the JAR file of the JDBC driver to the folder that contains the JAR file of the generated application.

16.4.5 Example: Generate and run Java code

This example walks you through the steps required to generate a Java application with MapForce, and compile it outside of MapForce using the Eclipse Integrated Development Environment (IDE) and Apache Ant. After completing this example, you will have created and compiled a complete Java application which executes one of the mapping samples available by default in MapForce.

If you can already compile successfully other Java applications with Eclipse and Ant, there are no special requirements to run this example. Otherwise, note the following prerequisites:

- Java Development Kit (JDK), Eclipse, and Ant must be installed on your system. Eclipse typically includes a bundled version of Ant but you can also install Ant separately, see also Building the project with Ant 1510.
- To run Eclipse with OpenJDK, the PATH environment variable typically needs to include the path to the JDK **bin** directory (for example, **C:\Java\jdk-11.0.1\bin**).
- If you run Apache Ant standalone like in this example, the PATH environment variable should also include the path to Ant **bin** directory (for example, **C:\apache-ant-1.10.5\bin**), so that you can conveniently run Ant without having to type the full path to the executable at the command line.

This example uses the following configuration:

- MapForce 64-bit
- OpenJDK 11.0.1 64-bit
- Eclipse IDE for Java Developers, version 2019-03 (4.11.0), 64-bit
- Apache Ant 1.10.6 (installed as standalone) and Apache Ant 1.10.5 (bundled with Eclipse)

Step 1: Generate Java code

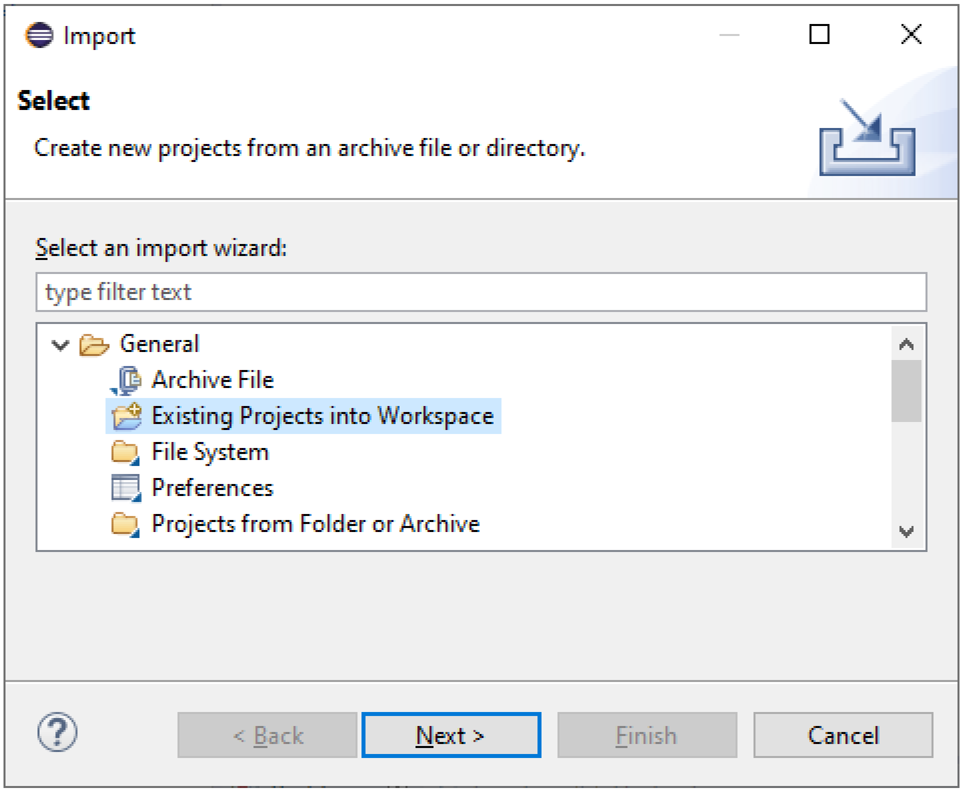
To generate the Java code from MapForce:

1. On the **File** menu, click **Open**, and browse for the **CompletePO.mfd** mapping available in the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.
2. On the **Output** menu, click **Java**. This changes the transformation language to Java.
3. On the **File** menu, click **Generate code in | Java**. When prompted, browse for the directory where the Java project should be saved. For the purpose of this example, you might want to generate Java code to **C:\codegen\java\CompletePO**. You will need to refer to this path in subsequent steps.

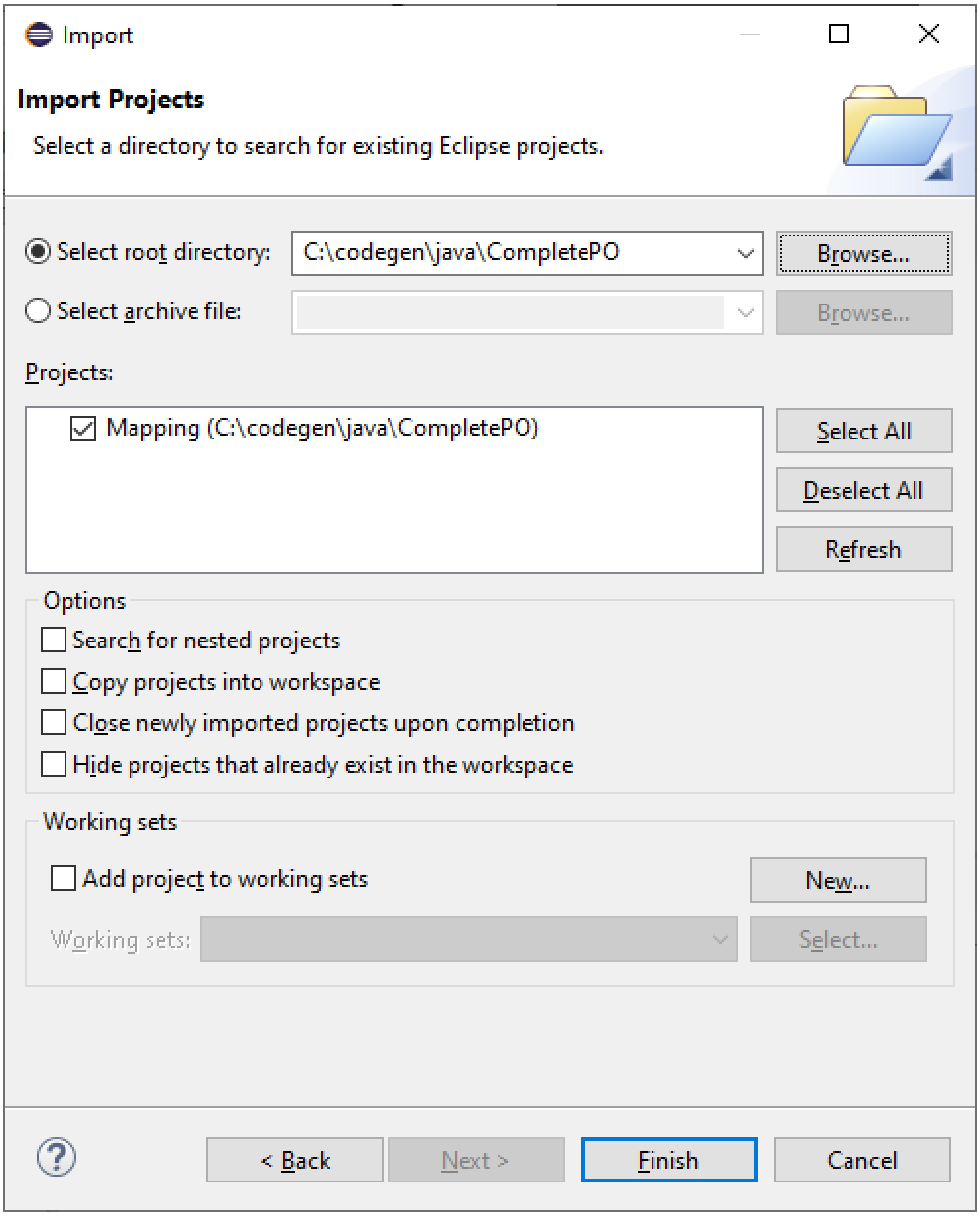
Step 2: Import the project into Eclipse

To import the project into Eclipse:

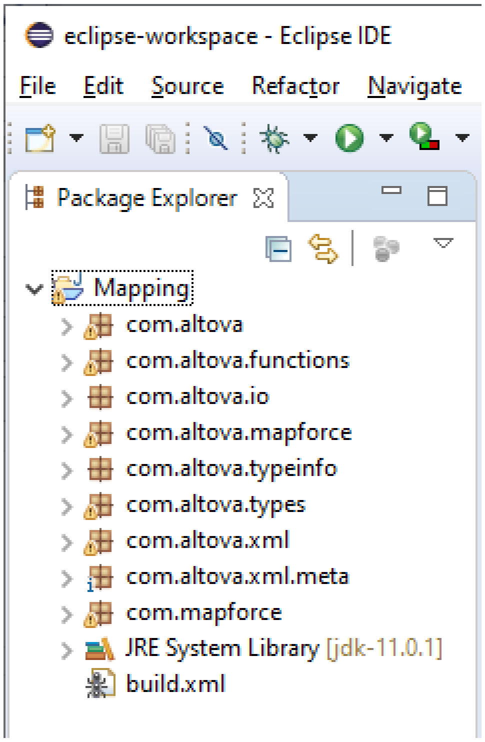
1. If you haven't done so already, run Eclipse and switch to the default Java perspective using the menu command **Window | Open Perspective**.
2. On the **File** menu, click **Import**, and then select **General | Existing Projects into Workspace**.


Import
Select
Create new projects from an archive file or directory.
Select an import wizard:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
Projects from Folder or Archive
< Back
Next >
Finish
Cancel

3. Click **Next**.



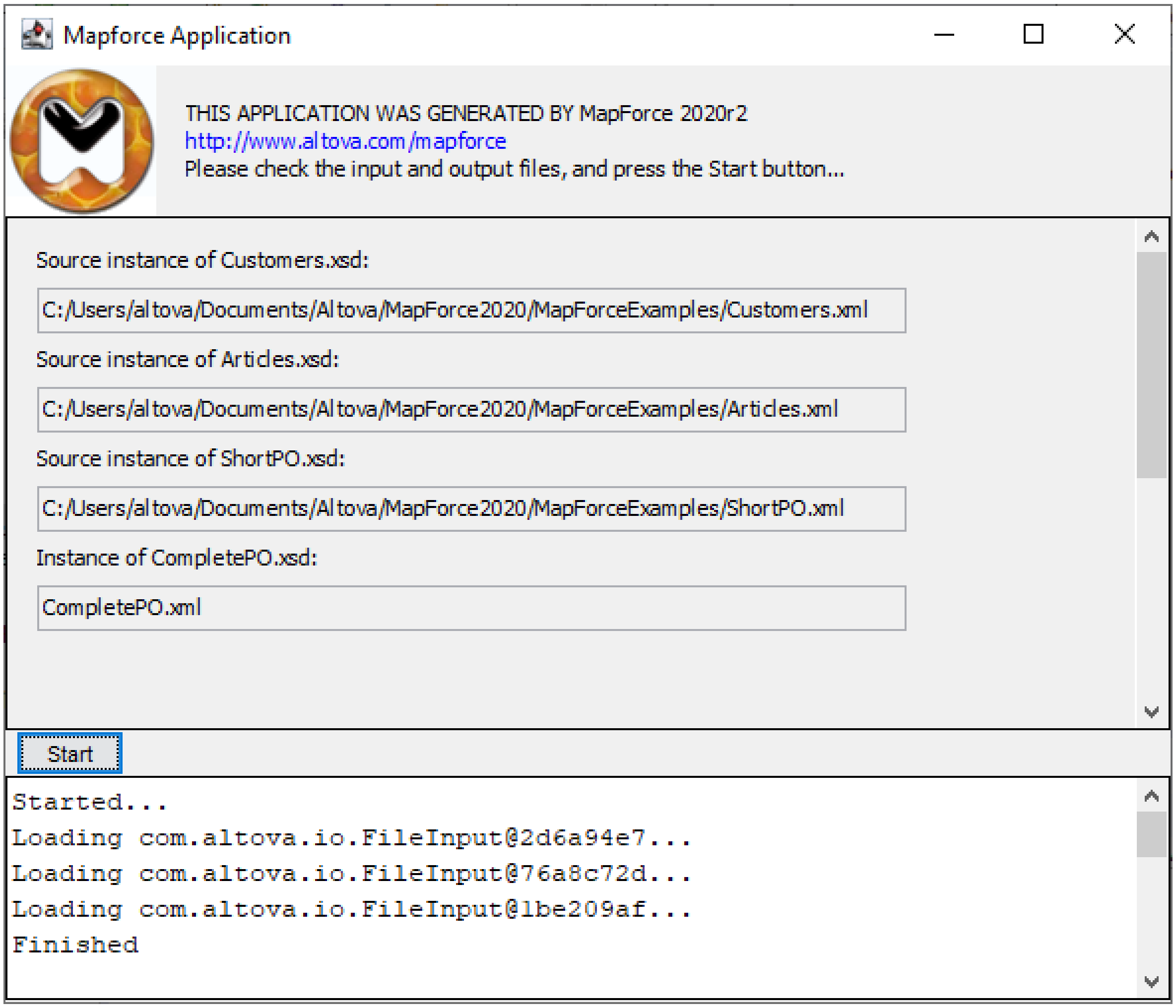
4. Browse for the folder where you have previously saved the generated code, and then click **Finish**. The Java project created by MapForce is now available in the Package Explorer view. If you cannot see the Package Explorer view, display it using the menu command **Window | Show View | Package Explorer**.



Step 3: Run the project as dialog application

To run the Java project as a GUI application:

1. In the Package Explorer view of Eclipse, click the **MappingApplication.java** file available in the **com.mapforce** package.
2. On the **Run** menu, click **Run As | Java application**.
3. On the MapForce application window, click **Start** to execute the mapping.

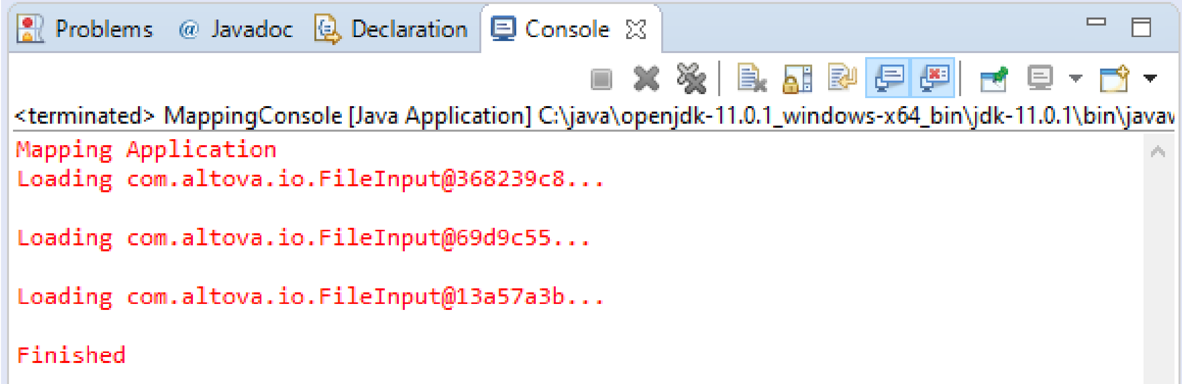


If Eclipse encounters system configuration or run-time errors, you will be prompted, see also Handling JDBC references1512. Otherwise, the Java application executes the mapping transformation and generates the **CompletePO.xml** at the output path (in this example, the output path is **C:\codegen\Java\CompletePO**).

Step 4: Run the project as console application

To run the Java project as a console application:

1. In the Package Explorer view of Eclipse, click the **MappingConsole.java** file available in the **com.mapforce** package.
2. On the **Run** menu, click **Run As | Java application**.



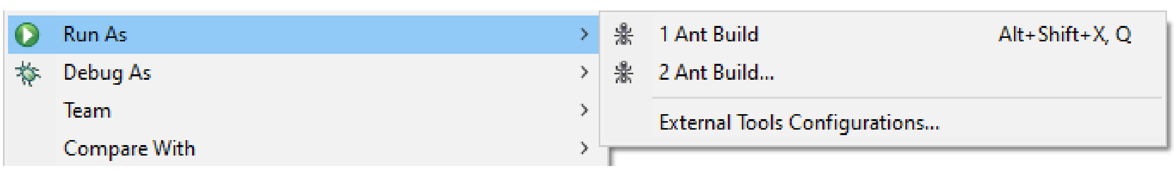
If Eclipse detects system configuration or run-time errors, you will be prompted, see also Handling JDBC references 1512. Otherwise, the Java application executes the mapping transformation and generates the **CompletePO.xml** at the output path (in this example, the output path is **C:\codegen\Java\CompletePO**).

Step 5: Build the JAR file

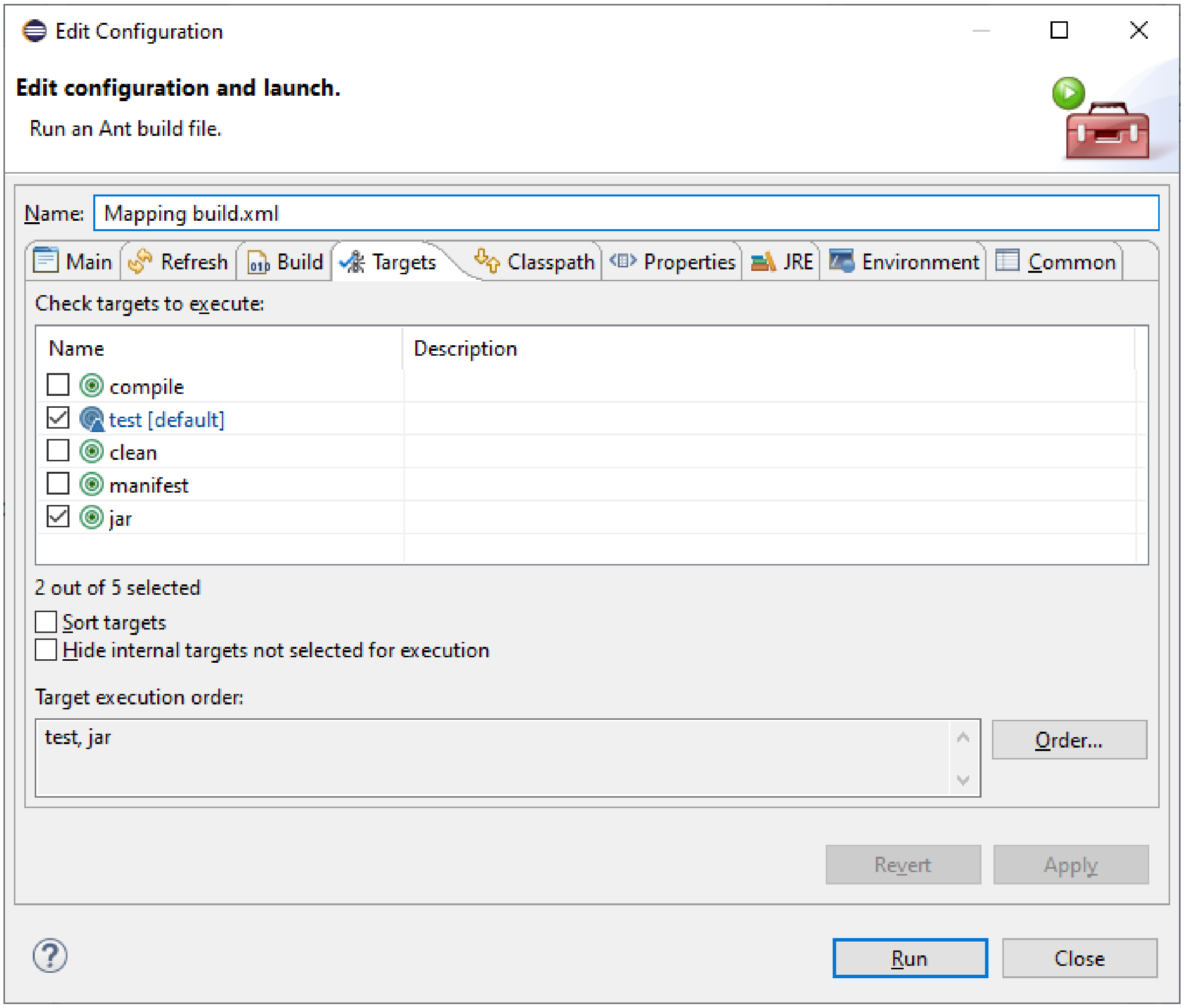
If Apache Ant is bundled with Eclipse, you can build the JAR file directly from the Eclipse IDE. Otherwise, to build the JAR file at the command line outside of Eclipse, Apache Ant must be installed separately.

To build the .jar file with Ant in Eclipse:

1. In the Package Explorer view of Eclipse, right-click the **build.xml** file, and point to the **Run As** menu. In the **Run As** menu, two possible options to run the Ant build file are displayed. If you choose the first option, Eclipse launches the Ant build with the default settings. If you choose the second option, you can change the settings of the Ant build before launching it. Select the second option.



2. Click to enable the targets that you wish to include in the Ant build. In this example, the targets **test** and **jar** are selected.



3. Click **Run**. Eclipse executes the Ant build file and displays the result in the Console view.

To build the JAR file at the command line:

1. Make sure that the directory where Apache Ant is installed (for example, **C:\apache-ant-1.10.5\bin**) is added to the PATH environment variable.
2. Open a Command Prompt window and change the current directory to the one where you generated Java code (in this example, **C:\codegen\java\CompletePO**). This is also the directory where the **build.xml** file was generated.
3. At the command line, enter:

```
ant clean jar
```

Ant runs the "clean" and "jar" targets from the **build.xml** file (including any dependent targets), and displays the outcome at the command line:

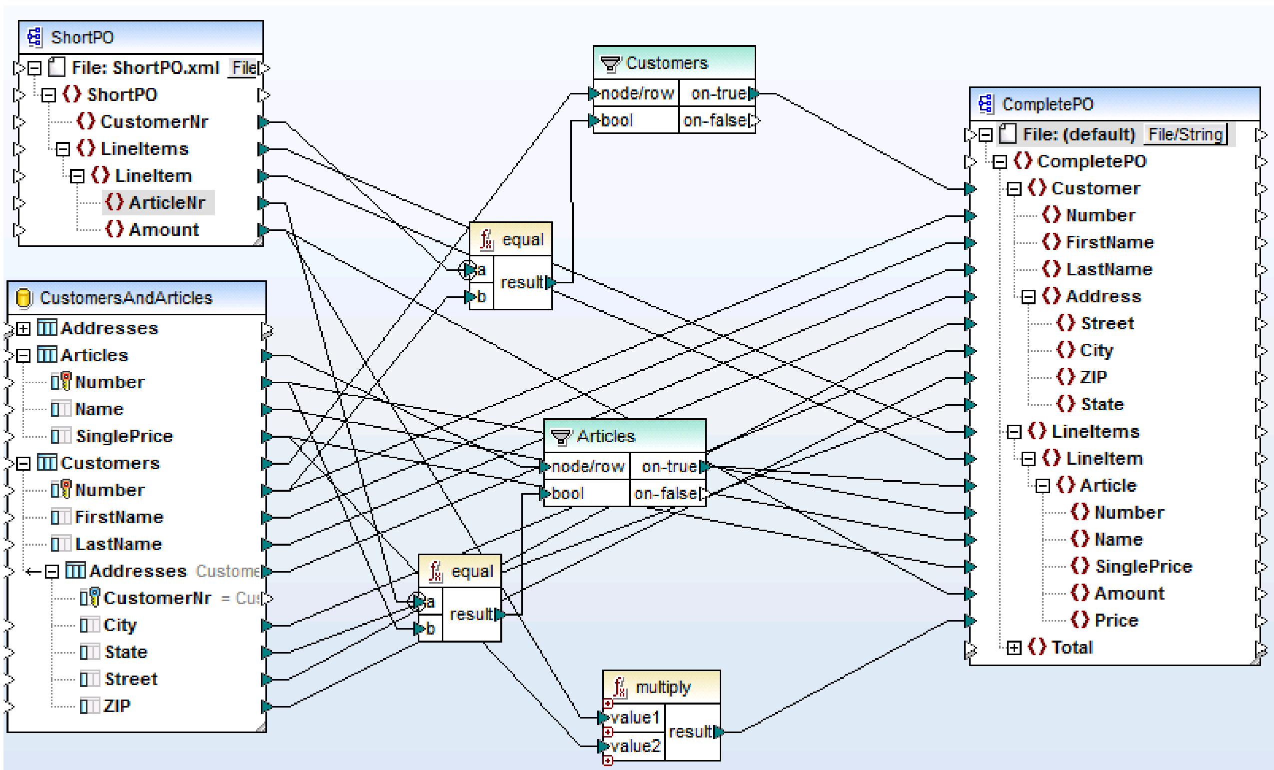
```
c:\codegen\java\CompletePO>ant clean jar
Buildfile: c:\codegen\java\CompletePO\build.xml

clean:

compile:
    [javac] Compiling 94 source files

manifest:

jar:
      [jar] Building jar: c:\codegen\java\CompletePO\Mapping.jar

BUILD SUCCESSFUL
Total time: 3 seconds

c:\codegen\java\CompletePO>
```

16.5 Integrating MapForce-Generated Code

MapForce-generated code can be integrated, or adapted to your specific application, even though the result of code generation is a complete and fully-functioning application. Some typical scenarios where you might want to change the generated code are as follows:

- Define custom source or target files for the mapping application
- Add custom error handling code
- In C# or Java generated code, you can also change the data type of the mapping input programmatically (for example, from string to stream).

This section provides instructions on how to achieve these goals, based on the **DB_CompletePO.mfd** sample mapping available in the **<Documents>\Altova\MapForce2023\MapForceExamples** directory.



DB_CompletePO.mfd mapping sample in MapForce

As illustrated above, the sample mapping consists of two sources and one target:

- **ShortPO.xml** is a source XML file
- **CustomersAndArticles.mdb** is a source database
- **CompletePO.xml** is the target XML file.

In the generated code, these sources and targets will translate to two input and one output parameters supplied to the run method which executes the mapping (as described in the subsequent topics). For now, note the following basic points about code generation:

- The number of source and targets in the mapping design corresponds to the number of mapping parameters to the **run** method in the generated code.
- If you change the number of sources or targets of the mapping, then you will need to re-generate the code accordingly.
- If you make changes to the generated code, and then re-generate the code at the same location, all changes will be overwritten.

If a mapping includes database components, the generated **run** method includes the database connection object at the appropriate location. For example, if the mapping uses three sources (text content, XML content and a database) to map to a single output file, MapForce generates the following **run** method:

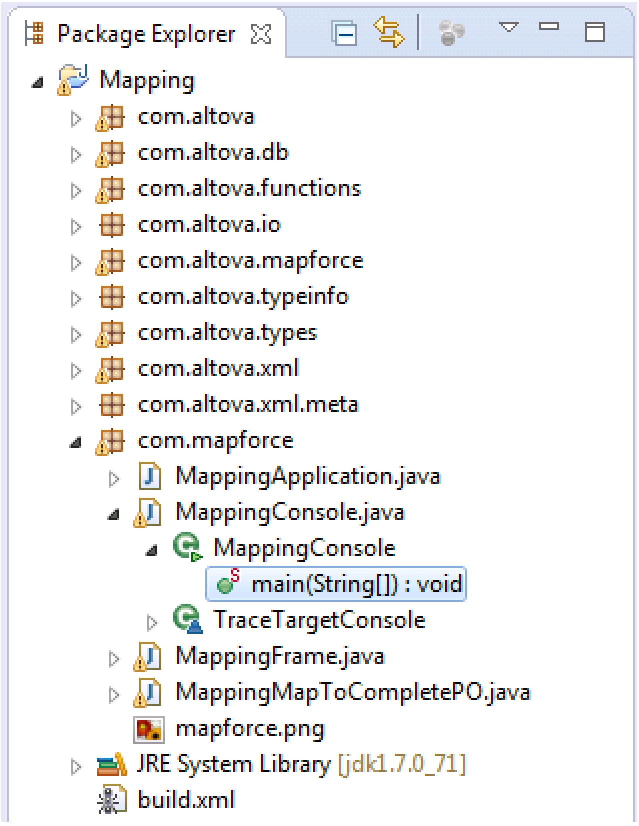
Java

```
void run(Input in1, Input in2, java.sql.Connection dbConn, Output out1);
```

The argument order is important. As you will see in the subsequent examples, you can modify `dbConn` parameters, or use the default parameters generated by MapForce when integrating your code.

16.5.1 Java example

This example uses Eclipse as Java IDE. To begin, generate Java code from the **DB_CompletePO** sample mapping available in the <Documents>\Altova\MapForce2023\MapForceExamples\ directory, and then import the project into Eclipse.



Sample MapForce-generated Java application (Eclipse IDE)

To edit the generated Java console application, locate the `main(String[] args)` method of your generated application (see the screen shot above). If you did not change the default base package name before generating code, this method is in the `MappingConsole` class of the `com.mapforce` package. Otherwise, it is in the `MappingConsole` class of your custom defined package.

To edit the generated Java dialog application, locate the place in the code where the `run` method is invoked from your generated application. If you did not change the default base package name before generating code, the run method is invoked from the class called `MappingFrame.java` of the `com.mapforce` package.

The following code sample illustrates an extract from the `main` method in the generated Java console application. The mapping sources and targets are highlighted in yellow and are defined as parameters to the run method. Since this mapping uses a database connection, the corresponding parameter has a special structure. Namely, the connection consists of the connection string (in this case, `jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);DBQ=CustomersAndArticles.mdb`), as well as two empty arguments intended for the **Username** and **Password** (in clear text) for those databases where this data is necessary.

Note that the file paths in the code below have been changed from absolute to relative.

```
com.altova.io.Input ShortPO2Source =
com.altova.io.StreamInput.createInput("ShortPO.xml");
com.altova.io.Output CompletePO2Target = new com.altova.io.FileOutput("CompletePO.xml");
```



```
MappingMapToCompletePOObject.run(
                  com.altova.db.Dbs.newConnection(
                     "jdbc:odbc:;DRIVER=Microsoft Access Driver
(*.mdb);DBQ=CustomersAndArticles.mdb",
                     "",
                     ""),
                  ShortPO2Source,
                  CompletePO2Target);
```

To define custom mapping source or target files:

- Locate the parameters passed to the `run` method and edit them as required. In the sample above, `com.altova.db.Dbs.newConnection` and `ShortPO2Source` is the mapping input and `CompletePO2Target` is the mapping output.

To add extra error handling code:

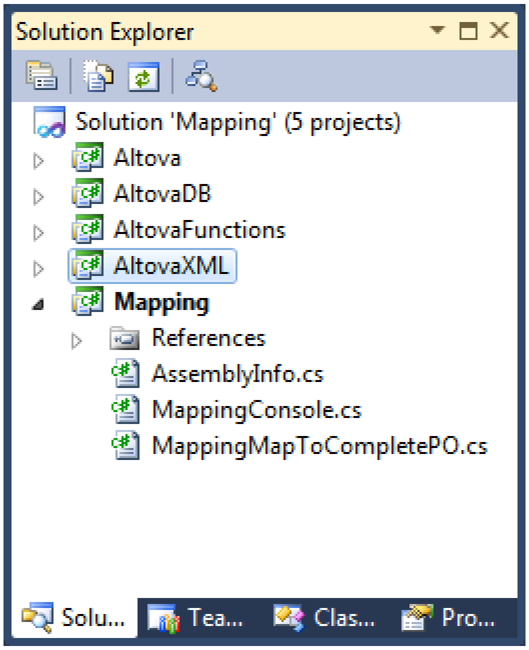
- Edit the code below the `catch (Exception e)` code (in case of a Java console application)
- Edit the code below the `catch (Exception ex)` code (in case of a Java dialog application)

For instructions on how to change the data type of parameters supplied as mapping input/output, see Changing Input and Output Programmatically 1528.

16.5.2 C# example

This example uses the Visual Studio 2010 IDE. To begin, generate C# code from the **DB_CompletePO** sample mapping available in the <Documents>\Altova\MapForce2023\MapForceExamples\ directory, and then open the solution in Visual Studio.

By default, the name of the solution file is **Mapping.sln**, and it is located in the **Mapping** subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to **MyApplication**, then the solution file is called **MyApplication.sln**, and it is located in the **MyApplication** subdirectory.



Sample C# solution generated with MapForce

Open the MappingConsole.cs file, and locate the `main(String[] args)` method. The following code sample illustrates an extract from the `main` method. The mapping sources and targets are defined as parameters to the **Run** method. Since this mapping reads data from a database, there is also an input parameter which is a database connection string. If necessary, you can modify the connection string of the database.

Note that the file paths in the code below have been changed from absolute to relative.

```
Altova.IO.Input ShortPO2Source = Altova.IO.StreamInput.createInput("ShortPO.xml");
Altova.IO.Output CompletePO2Target = new Altova.IO.FileOutput("CompletePO.xml");

MappingMapToCompletePOObject.Run(
               "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=CustomersAndArticles.mdb;
",
               ShortPO2Source,
               CompletePO2Target);
```

To define custom mapping source or target files:

- Locate the parameters passed to the Run method and edit them as required. In the sample above, the mapping input is a connection string to the **CustomersAndArticles.mdb** and `ShortPO2Source` . The mapping output is `CompletePO2Target`.

To add extra error handling code:

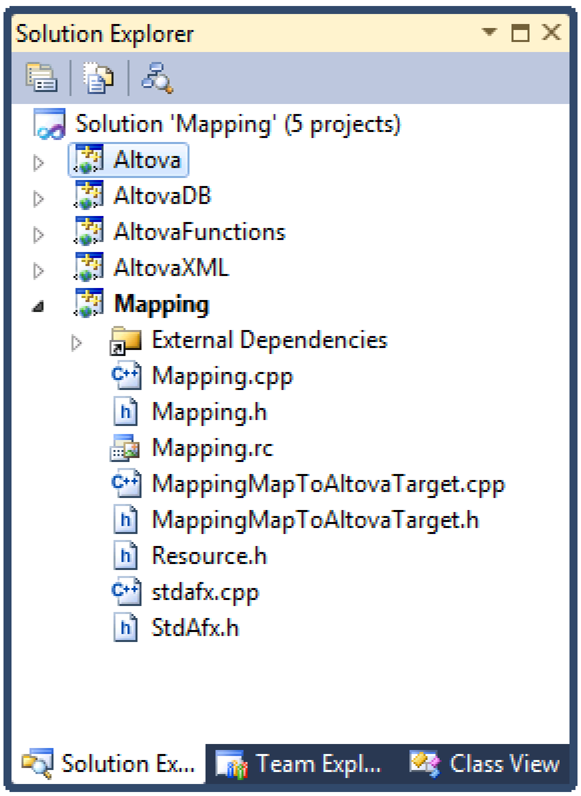
- Edit the code below the `catch (Exception e)` code

For instructions on how to change the data type of parameters supplied as mapping input/output, see [Changing Input and Output Programmatically](1528).

16.5.3 C++ example

This example uses the Visual Studio 2010 IDE. To begin, generate C++ code from the **DB_CompletePO** sample mapping available in the <Documents>\Altova\MapForce2023\MapForceExamples\ directory, and then open the solution in Visual Studio.

By default, the name of the solution file is **Mapping.sln**, and it is located in the **Mapping** subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to **MyApplication**, then the solution file is called **MyApplication.sln**, and it is located in the **MyApplication** subdirectory.



Sample C++ solution generated with MapForce

Open the **Mapping.cpp** file, and locate the `_tmain` method. The following code sample illustrates an extract from this method. The mapping sources and targets are defined as parameters to the Run method. Since this mapping reads data from a database, there is also an input parameter which is a database connection string. If necessary, you can modify the connection string of the database.

Note that the file paths in the code below have been changed from absolute to relative.

```
MappingMapToCompletePO MappingMapToCompletePOObject;
         MappingMapToCompletePOObject.Run(
            _T("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=CustomersAndArticles.mdb;
"),
            _T("ShortPO.xml"),
            _T("CompletePO.xml"));
```

To define custom mapping source or target files:

- Locate the parameters passed to the Run method and edit them as required. In the code sample above, the mapping input is a connection string to the **CustomersAndArticles.mdb** database and _T("ShortPO.xml"). The mapping output is _T("CompletePO.xml").

To add extra error handling code:

- Edit the code below the `catch (CAltovaException& e)` code.

16.5.4 Changing Input and Output Programmatically

After generating Java or C# code with MapForce, you can optionally change the data type of the mapping input or output by editing the generated code. More specifically, you can use as mapping parameters objects of types other than those generated by default. For example, instead of having the mapping read the input from a file on the disk, you can provide a string or a stream object as input. Note that this feature is specific to code generated in C# or Java only.

The object types supported as input or output are listed in the first column of the table below. Each subsequent column specifies data formats where that specific type is supported. For a more precise definition of each type, see the "Type definitions" section below.

	XML	**JSON***	**Microsoft Excel***	**EDI (includes X12, HL7)***	**FlexText***	**CSV/Text**
Files	Yes	Yes	Yes	Yes	Yes	Yes
Streams	Yes	Yes	Yes	Yes	Yes	Yes
Strings	Yes	Yes	–	Yes	Yes	Yes
Reader/Writer	Yes	Yes	–	Yes	Yes	Yes
DOM documents	Yes	–	–	–	–	–

** Formats supported only in MapForce Enterprise Edition*

To change the data type of the mapping input or output:

1. Generate C# or Java code from a mapping.
2. In the generated code, find the call to the **run** method (in Java) or **Run** method (in C#), as follows:

   a. If using C#, open the **MappingConsole.cs** file.
   b. If using Java, open the **MappingConsole.java** (the console program) or the **MappingFrame.java** file (the GUI program).

Note: The name of the file may be different if you have changed the application name in the mapping settings [114]. For example, if you changed it to "MyApp", then name of the generated file becomes **MyAppConsole.js** and **MyAppConsole.java**, and **MyAppFrame.java**, respectively.

3. Create an instance of the required type (see the "Type definitions" section).
4. Supply the declared objects as parameters to the **run** method (in Java) or **Run** method (in C#), as shown in the examples below.

The **run** method is the most important method of generated mapping classes. It has one parameter for each *static* source or input component in the mapping, and a final parameter for the output component. If your mapping contains components that process multiple files dynamically [1313], the respective parameters do not appear in generated code, because in this case the file names are processed dynamically inside the mapping.

Type definitions

In C#, the types that you can provide as parameters to the **Run** method are classes defined in the `Altova.IO` namespace. The base classes are `Altova.IO.Input` and `Altova.IO.Output`, respectively.

C# types

Files	`Altova.IO.FileInput(string filename)` `Altova.IO.FileOutput(string filename)`
Streams	`Altova.IO.StreamInput(System.IO.Stream stream)` `Altova.IO.StreamOutput(System.IO.Stream stream)`
Strings	`Altova.IO.StringInput(string content)` `Altova.IO.StringOutput(System.Text.StringBuilder sbuilder)`
Reader/Writer	`Altova.IO.ReaderInput(System.IO.TextReader reader)` `Altova.IO.WriterOutput(System.IO.TextWriter writer)`
DOM documents	`Altova.IO.DocumentInput(System.Xml.XmlDocument document)` `Altova.IO.DocumentOutput(System.Xml.XmlDocument document)`

In Java, the types that you can provide as parameters to the **run** method are classes defined in the `com.altova.io` package. The base classes are `com.altova.io.Input` and `com.altova.io.Output`, respectively.

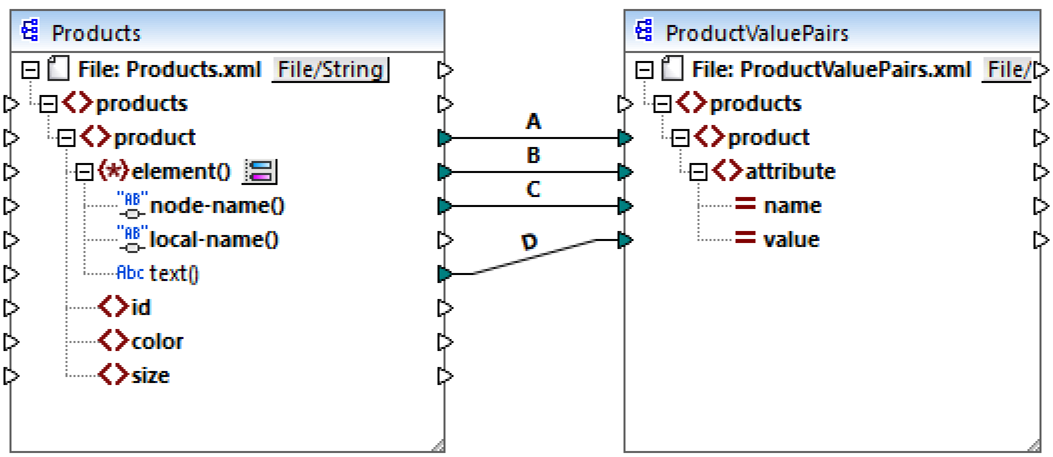
Java types

Files	`com.altova.io.FileInput(String filename)`

	`com.altova.io.FileOutput(String filename)`
Streams	`com.altova.io.StreamInput(java.io.InputStream stream)` `com.altova.io.StreamOutput(String filename)`
Strings	`com.altova.io.StringInput(String content)` `com.altova.io.StringOutput()`
Reader/Writer	`com.altova.io.ReaderInput(java.io.Reader reader)` `com.altova.io.WriterOutput(java.io.Writer writer)`
DOM documents	`com.altova.io.DocumentInput(org.w3c.dom.Document document)` `com.altova.io.DocumentOutput(org.w3c.dom.Document document)`

Example

To illustrate changing the input and output programmatically, we will use the **ConvertProducts.mfd** mapping as a model. After installing MapForce and running it at least once, you can find this mapping in the following directory: **C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\Tutorials**.

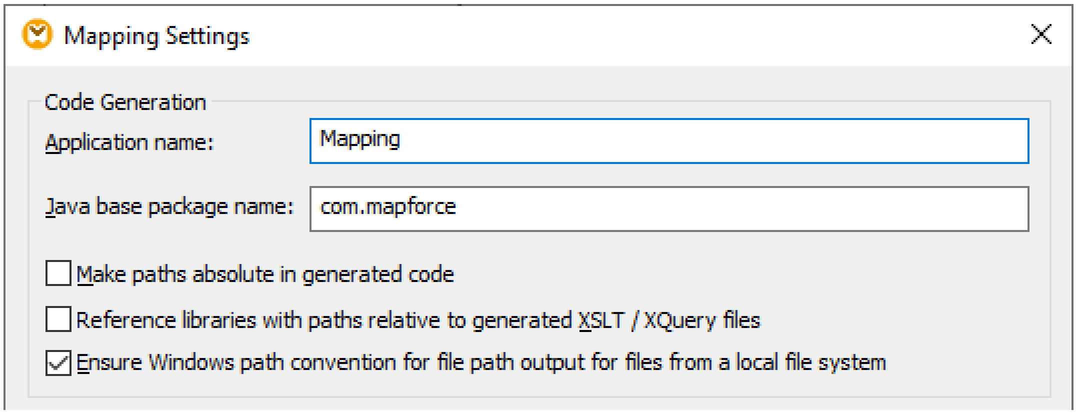


ConvertProducts.mfd

As illustrated above, the mapping converts data from a source XML document to another XML document. Our goals are as follows:

1. Generate Java and C# program code from this mapping.
2. Change the data type of the source component to a string type.
3. Change the data type of the target component to a string writer type.

To generate the program code, open the **ConvertProducts.mfd** mapping and run the **File | Generate code in | C#** (or **Java**) command. For the scope of this example, we will assume that the mapping settings of **ConvertProducts.mfd** are the default ones.



This example uses the following target directories for the generated code (feel free to change the path if necessary):

- **C:\codegen\cs\ConvertProducts**, for C#
- **C:\codegen\java\ConvertProducts**, for Java

Having generated the program code, open the **MappingConsole.cs** (in C#) or **MappingConsole.java** (in Java) and find the following lines:

C#

```
Altova.IO.Input Products2Source = Altova.IO.StreamInput.createInput("Products.xml");
Altova.IO.Output ProductValuePairs2Target = new
Altova.IO.FileOutput("ProductValuePairs.xml");
```

Java

```
com.altova.io.Input Products2Source =
com.altova.io.StreamInput.createInput("Products.xml");
com.altova.io.Output ProductValuePairs2Target = new
com.altova.io.FileOutput("ProductValuePairs.xml");
```

Comment out the lines above and change the code as follows:

C#

```
//Altova.IO.Input Products2Source = Altova.IO.StreamInput.createInput("Products.xml");
//Altova.IO.Output ProductValuePairs2Target = new
Altova.IO.FileOutput("ProductValuePairs.xml");

Altova.IO.Input Products2Source = new Altova.IO.StringInput("<?xml version=\"1.0\"
encoding=\"UTF-8\"?>\r\n" +
                                       "<products
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\"products.xsd\">\r\n" +
                                       "   <product>\r\n" +
                                       "      <id>100</id>\r\n" +
                                       "      <color>blue</color>\r\n" +
```

```
                                       "      <size>XXL</size>\r\n" +
                                       "   </product>\r\n" +
                                       "</products>\r\n");

System.IO.StringWriter writer = new System.IO.StringWriter(new
System.Text.StringBuilder());
Altova.IO.Output ProductValuePairs2Target = new Altova.IO.WriterOutput(writer);

try
{
    MappingMapToProductValuePairsObject.Run(Products2Source, ProductValuePairs2Target);

    // Print out the writer object
    Console.Write(writer.ToString());
}
finally
{
    Products2Source.Close();
    ProductValuePairs2Target.Close();
}
```

Java

```
//com.altova.io.Input Products2Source =
com.altova.io.StreamInput.createInput("Products.xml");
//com.altova.io.Output ProductValuePairs2Target = new
com.altova.io.FileOutput("ProductValuePairs.xml");

com.altova.io.Input Products2Source = new com.altova.io.StringInput("<?xml
version=\"1.0\" encoding=\"UTF-8\"?>\r\n" +
               "<products xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\"products.xsd\">\r\n" +
               "   <product>\r\n" +
               "      <id>100</id>\r\n" +
               "      <color>blue</color>\r\n" +
               "      <size>XXL</size>\r\n" +
               "   </product>\r\n" +
               "</products>\r\n");

java.io.StringWriter writer = new java.io.StringWriter();
com.altova.io.Output ProductValuePairs2Target = new com.altova.io.WriterOutput(writer);

try {
   MappingMapToProductValuePairsObject.run(Products2Source, ProductValuePairs2Target);

   // Print out the writer object
   System.out.print(writer.toString());

   } finally {
      (Products2Source).close();
      ProductValuePairs2Target.close();
}
```

In the C# and Java code listings above, the following happens:

- The two original lines that provide the input and output to the **run** method were commented out. Consequently, the mapping application no longer reads data from **Products.xml**. In fact, we did not even need to copy this file to the program's working directory.
- The type `Products2Source` has been declared as a `StringInput` that provides the content of the input XML file to be processed.
- The type `ProductValuePairs2Target` has been declared as a `WriterOutput` type that takes a string writer as argument.
- After the mapping completes running, the contents of the string writer is printed out to the console.

Usage guidelines for streams and Reader/Writer objects

When using binary streams or Reader/Writer objects as input or output to the mapping, note the following:

- Binary stream objects and Reader/Writer objects are expected to be opened and ready to use before calling the **run** method.
- By default, the **run** method closes the stream when finished. To prevent this behavior, insert (or uncomment) the following line before calling the **run** method:

C#

```
MappingMapToSomething.CloseObjectsAfterRun = false;
```

Java

```
MappingMapToSomething.setCloseObjectsAfterRun(false);
```

Note: Make sure to change `MappingMapToSomething` to the name of the mapping object as applicable to your generated code.

Usage guidelines for strings

In Java, the constructor of `StringOutput` does not take an argument. The string content produced by the mapping can be accessed by calling the `getString()` method, for example:

Java

```
com.altova.io.Input Products2Source =
com.altova.io.StreamInput.createInput("Products.xml");
com.altova.io.StringOutput ProductValuePairs2Target = new com.altova.io.StringOutput();

try {
   // Run the mapping
   MappingMapToProductValuePairsObject.run(Products2Source, ProductValuePairs2Target);
   // Get the string object
   String str = ProductValuePairs2Target.getString().toString();
}
```

In C#, the constructor of `StringOutput` takes a parameter of type `StringBuilder` which you need to declare beforehand. If the `StringBuilder` object already contains data, the mapping output will be appended to it.

C#

```
Altova.IO.Input Products2Source = Altova.IO.StreamInput.createInput("Products.xml");
System.Text.StringBuilder sb = new System.Text.StringBuilder();
Altova.IO.Output ProductValuePairs2Target = new Altova.IO.StringOutput(sb);

try
{
   // Run the mapping
   MappingMapToProductValuePairsObject.Run(Products2Source, ProductValuePairs2Target);
   // Get the string output
   String str = sb.ToString();
}
```

To run these code listings, you can use the same generated project as in the previous example. Make sure, however, to copy the file **Products.xml** from **C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\Tutorials** to your program's working directory, since the mapping code reads data from this file.

Usage guidelines for DOM documents

When using DOM documents as mapping input or output, note the following:

- The document instance supplied as parameter to the `DocumentOutput` constructor must be empty.
- After calling **run**, the DOM Document generated by the constructor of `DocumentOutput` already contains the mapping output, and you can manipulate the document as necessary.

16.6 Generating Code from XML Schemas or DTDs

When you generate code from a mapping, MapForce generates a complete application that executes all steps of the mapping automatically. Optionally, you can generate libraries for all the XML schemas used in the mapping. These allow your code to easily create or read XML instances that are used or created by the mapping code.

To generate libraries for all the XML schemas used in the mapping, select the **Generate Wrapper Classes** check box from the code generator options 1625. Next time when you generate code, MapForce will create not only the mapping application, but also wrapper classes for all schemas used in the mapping, as follows:

C++ or C#	Java	Purpose
Altova	`com.altova`	Base library containing common runtime support, identical for every schema.
AltovaXML	`com.altova.xml`	Base library containing runtime support for XML, identical for every schema.
[YourSchema]	`com.YourSchema`	A library containing declarations generated from the input schema, named as the schema file or DTD. This library is a DOM (W3C Document Object Model) wrapper that allows you to read, modify and create XML documents easily and safely. All data is held inside the DOM, and there are methods for extracting data from the DOM, and to update and create data into the DOM. The generated C++ code supports either Microsoft MSXML or Apache Xerces 3. The syntax for using the generated code is generally similar for both DOM implementations, except for a few slight differences (for example, Xerces supports more overloaded functions). The generated C# code uses the .NET standard System.XML library as the underlying DOM implementation. The generated Java code uses JAXP (Java API for XML Processing) as the underlying DOM interface.

While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore, do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see Integrating Schema Wrapper Libraries 1544).

In addition to the base libraries listed above, some supporting libraries are also generated. The supporting libraries are used by the Altova base libraries and are not meant for custom integrations, since they are subject to change.

Name generation and namespaces

MapForce generates classes corresponding to all declared elements or complex types which redefine any complex type in your XML Schema, preserving the class derivation as defined by extensions of complex types in your XML Schema. In the case of complex schemas which import schema components from multiple namespaces, MapForce preserves this information by generating the appropriate C# or C++ namespaces or Java packages.

Generally, the code generator tries to preserve the names for generated namespaces, classes and members from the original XML Schema. Characters that are not valid in identifiers in the target language are replaced by a "_". Names that would collide with other names or reserved words are made unique by appending a number. Name generation can be influenced by changing the default settings in the SPL1627 template.

The namespaces from the XML Schema are converted to packages in Java or namespaces in C# or C++ code, using the namespace prefix from the schema as code namespace. The complete library is enclosed in a package or namespace derived from the schema file name, so you can use multiple generated libraries in one program without name conflicts.

Data Types

XML Schema has a more elaborate data type model than Java, C# or C++. Code Generator converts the built-in XML Schema types to language-specific primitive types, or to classes delivered with the Altova library. Complex types and derived types defined in the schema are converted to classes in the generated library. Enumeration facets from simple types are converted to symbolic constants.

The mapping of simple types can be configured in the SPL template, see SPL Reference1627.

If your XML instance files use schema types related to time and duration, these are converted to Altova native classes in the generated code. For information about the Altova library classes, see:

- Reference to Generated Classes (C++)1579
- Reference to Generated Classes (C#)1594
- Reference to Generated Classes (Java)1609

For information about type conversion and other details applicable to each language, see:

- About Schema Wrapper Libraries (C++)1537
- About Schema Wrapper Libraries (C#)1540
- About Schema Wrapper Libraries (Java)1542

Memory management

A DOM tree is comprised of nodes, which are always owned by a specific DOM document - even if the node is not currently part of the document's content. All generated classes are references to the DOM nodes they represent, not values. This means that assigning an instance of a generated class does not copy the value, it only creates an additional reference to the same data.

XML Schema support

The following XML Schema constructs are translated into code:

a) XML namespaces

b) Simple types:

- Built-in XML schema types
- Simple types derived by extension
- Simple types derived by restriction
- Facets
- Enumerations
- Patterns

c) Complex types:

- Built-in anyType node
- User-defined complex types
- Derived by extension: Mapped to derived classes
- Derived by restriction
- Complex content
- Simple content
- Mixed content

The following advanced XML Schema features are not supported (or not fully supported) in generated wrapper classes:

- Wildcards: `xs:any` and `xs:anyAttribute`
- Content models (sequence, choice, all). Top-level compositor is available in SPL 1627, but is not enforced by generated classes.
- Default and fixed values for attributes. These are available in SPL 1627, but are not set or enforced by generated classes.
- The attributes `xsi:type`, abstract types. When you need to write the xsi:type attribute, use the `SetXsiType()` method of the generated classes.
- Union types: not all combinations are supported.
- Substitution groups are partially supported (resolved like "choice").
- Attribute `nillable="true"` and `xsi:nil`
- Uniqueness constraints
- Identity constraints (`key` and `keyref`)

16.6.1 About Schema Wrapper Libraries (C++)

Character Types

The generated C++ code can be compiled with or without Unicode support. Depending on this setting, the types `string_type` and `tstring` will both be defined as `std::string` or `std::wstring`, consisting of narrow or wide characters. To use Unicode characters in your XML file that are not representable with the current 8-bit character set, Unicode support must be enabled. Pay special attention to the `_T()` macros. This macro ensures that string constants are stored correctly, whether you're compiling for Unicode or non-Unicode programs.

Data Types

The default mapping of XML Schema types to C++ data types is:

XML Schema	C++	Remarks
xs:string	string_type	string_type is defined as std::string or std:wstring
xs:boolean	bool	
xs:decimal	double	C++ does not have a decimal type, so double is used.
xs:float, xs:double	double	
xs:integer	__int64	xs:integer has unlimited range, mapped to __int64 for efficiency reasons.
xs:nonNegativeInteger	unsigned __int64	see above
xs:int	int	
xs:unsignedInt	unsigned int	
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	altova::DateTime [1579]	
xs:duration	altova::Duration [1582]	
xs:hexBinary and xs:base64Binary	std::vector<unsigned char>	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string_type	

All XML Schema types not contained in this list are derived types, and mapped to the same C++ type as their respective base type.

Generated Classes

For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple types, assignment and conversion operators are generated. For simple types with enumeration facets, the methods `GetEnumerationValue()` and `SetEnumerationValue(int)` can be used together with generated constants for each enumeration value. In addition, the method `StaticInfo()` allows accessing schema information as one of the following types:

altova::meta::SimpleType [1587]
altova::meta::ComplexType [1586]

Classes generated from complex types include the method `SetXsiType()`, which enables you to set the `xsi:type` attribute of the type. This method is useful when you want to create XML instance elements of a derived type.

In addition to the classes for the types declared in the XML Schema, a document class (identified with "CDoc" below) is generated. It contains all possible root elements as members, and various other methods. For more information about the class, see [YourSchema]::[CDoc] [1589].

Note: The actual class name depends on the name of the .xsd schema.

For each member attribute or element of a schema type, a new class is generated. For more information about such classes, see:

`[YourSchema]::MemberAttribute` [1592]
`[YourSchema]::MemberElement` [1593]

Note: The actual class names depend on the name of the schema attribute or element.

See also Example: Using the Schema Wrapper Libraries [1547].

Error Handling

Errors are reported by exceptions. The following exception classes are defined in the namespace `altova`:

Class	Base Class	Description
`Error`	`std::logic_error`	Internal program logic error (independent of input data).
`Exception`	`std::runtime_error`	Base class for runtime errors.
`InvalidArgumentsException`	`Exception`	A method was called with invalid argument values.
`ConversionException`	`Exception`	Exception thrown when a type conversion fails.
`StringParseException`	`ConversionException`	A value in the lexical space cannot be converted to value space.
`ValueNotRepresentableException`	`ConversionException`	A value in the value space cannot be converted to lexical space.
`OutOfRangeException`	`ConversionException`	A source value cannot be represented in target domain.
`InvalidOperationException`	`Exception`	An operation was attempted that is not valid in the given context.
`DataSourceUnavailableException`	`Exception`	A problem occurred while loading an XML instance.
`DataTargetUnavailableException`	`Exception`	A problem occurred while saving an XML instance.

All exception classes contain a message text and a pointer to a possible inner exception.

Method	Purpose
`string_type message()`	Returns a textual description of the exception.

Method	Purpose
`std::exception inner()`	Returns the exception that caused this exception, if available, or NULL.

Accessing schema information

The generated library allows accessing static schema information via the following classes. All methods are declared as `const`. The methods that return one of the metadata classes return a NULL object if the respective property does not exist.

altova::meta::Attribute 1585
altova::meta::ComplexType 1586
altova::meta::Element 1587
altova::meta::SimpleType 1587

16.6.2 About Schema Wrapper Libraries (C#)

The default mapping of XML Schema types to C# data types is as follows.

XML Schema	C#	Remarks
xs:string	string	
xs:boolean	bool	
xs:decimal	decimal	xs:decimal has unlimited range and precision, mapped to decimal for efficiency reasons.
xs:float, xs:double	double	
xs:long	long	
xs:unsignedLong	ulong	
xs:int	int	
xs:unsignedInt	uint	
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	Altova.Types.DateTime 1594	
xs:duration	Altova.Types.Duration 1598	
xs:hexBinary and xs:base64Binary	byte[]	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string	

All XML Schema types not contained in this list are derived types, and mapped to the same C# type as their respective base type.

Generated Classes

For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple types, assignment and conversion operators are generated. For simple types with enumeration facets, the methods `GetEnumerationValue()` and `SetEnumerationValue(int)` can be used together with generated constants for each enumeration value. In addition, the method `StaticInfo()` allows accessing schema information as one of the following types:

`Altova.Xml.Meta.SimpleType` 1603
`Altova.Xml.Meta.ComplexType` 1601

Classes generated from complex types include the method `SetXsiType()`, which enables you to set the `xsi:type` attribute of the type. This method is useful when you want to create XML instance elements of a derived type.

In addition to the classes for the types declared in the XML Schema, a document class (identified with "Doc" below) is generated. It contains all possible root elements as members, and various other methods. For more information about the class, see `[YourSchema].[Doc]` 1604.

Note: The actual class name depends on the name of the .xsd schema.

For each member attribute or element of a schema type, a new class is generated. For more information about such classes, see:

`[YourSchemaType].MemberAttribute` 1607
`[YourSchemaType].MemberElement` 1607

Note: The actual class names depend on the name of the schema attribute or element.

Error Handling

Errors are reported by exceptions. The following exception classes are defined in the namespace Altova:

Class	Base Class	Description
`ConversionException`	`Exception`	Exception thrown when a type conversion fails
`StringParseException`	`ConversionException`	A value in the lexical space cannot be converted to value space.
`DataSourceUnavailableException`	`System.Exception`	A problem occurred while loading an XML instance.
`DataTargetUnavailableException`	`System.Exception`	A problem occurred while saving an XML instance.

In addition, the following .NET exceptions are commonly used:

Class	Description
`System.Exception`	Base class for runtime errors
`System.ArgumentException`	A method was called with invalid argument values, or a type conversion failed.
`System.FormatException`	A value in the lexical space cannot be converted to value space.
`System.InvalidCastException`	A value cannot be converted to another type.
`System.OverflowException`	A source value cannot be represented in target domain.

Accessing schema information

The generated library allows accessing static schema information via the following classes:

Altova.Xml.Meta.Attribute 1601
Altova.Xml.Meta.ComplexType 1601
Altova.Xml.Meta.Element 1602
Altova.Xml.Meta.SimpleType 1603

The properties that return one of the metadata classes return null if the respective property does not exist.

16.6.3 About Schema Wrapper Libraries (Java)

The default mapping of XML Schema types to Java data types is as follows:

XML Schema	Java	Remarks
`xs:string`	`String`	
`xs:boolean`	`boolean`	
`xs:decimal`	`java.math.BigDecimal`	
`xs:float, xs:double`	`double`	
`xs:integer`	`java.math.BigInteger`	
`xs:long`	`long`	
`xs:unsignedLong`	`java.math.BigInteger`	Java does not have unsigned types.
`xs:int`	`int`	
`xs:unsignedInt`	`long`	Java does not have unsigned types.
`xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth`	`com.altova.types.DateTime` 1609	

XML Schema	Java	Remarks
xs:duration	com.altova.types.Duration [1613]	
xs:hexBinary and xs:base64Binary	byte[]	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string	

All XML Schema types not contained in this list are derived types, and mapped to the same Java type as their respective base type.

Generated Classes

For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple types, assignment and conversion operators are generated. For simple types with enumeration facets, the methods `GetEnumerationValue()` and `SetEnumerationValue(int)` can be used together with generated constants for each enumeration value. In addition, the method `StaticInfo()` allows accessing schema information as one of the following types:

com.altova.xml.meta.SimpleType [1618]
com.altova.xml.meta.ComplexType [1617]

Classes generated from complex types include the method `SetXsiType()`, which enables you to set the `xsi:type` attribute of the type. This method is useful when you want to create XML instance elements of a derived type.

In addition to the classes for the types declared in the XML Schema, a document class (identified with "Doc" below) is generated. It contains all possible root elements as members, and various other methods. For more information about the class, see com.[YourSchema].[Doc] [1619].

Note: The actual class name depends on the name of the .xsd schema.

For each member attribute or element of a schema type, a new class is generated. For more information about such classes, see:

com.[YourSchema].[YourSchemaType].MemberAttribute [1622]
com.[YourSchema].[YourSchemaType].MemberElement [1622]

Note: The actual class names depend on the name of the schema attribute or element.

Error Handling

Errors are reported by exceptions. The following exception classes are defined in the namespace com.altova:

Class	Base Class	Description
SourceInstanceUnvailableException	Exception	A problem occurred while loading an XML instance.

Class	Base Class	Description
`TargetInstanceUnavailableException`	Exception	A problem occurred while saving an XML instance.

In addition, the following Java exceptions are commonly used:

Class	Description
`java.lang.Error`	Internal program logic error (independent of input data)
`java.lang.Exception`	Base class for runtime errors
`java.lang.IllegalArgumentsException`	A method was called with invalid argument values, or a type conversion failed.
`java.lang.ArithmeticException`	Exception thrown when a numeric type conversion fails.

Accessing schema information

The generated library allows accessing static schema information via the following classes:

`com.altova.xml.meta.Attribute` 1617
`com.altova.xml.meta.ComplexType` 1617
`com.altova.xml.meta.Element` 1618
`com.altova.xml.meta.SimpleType` 1618

The properties that return one of the metadata classes return null if the respective property does not exist.

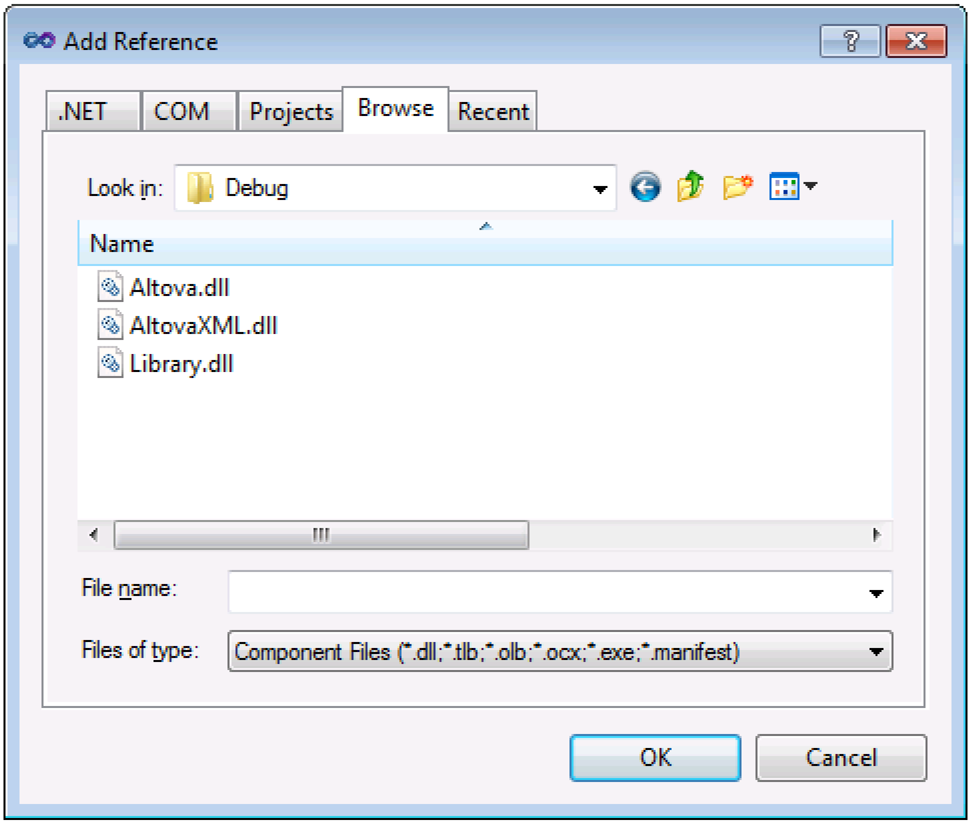
16.6.4 Integrating Schema Wrapper Libraries

To use the Altova libraries in your custom project, refer to the libraries from your project or include them in your project, as shown below for each language.

C#

To integrate the Altova libraries into an existing C# project:

1. After MapForce generates code from a schema (for example, **YourSchema.xsd**), build the generated **YourSchema.sln** solution in Visual Studio. This solution is in a project folder with the same name as the schema.
2. Right-click your existing project in Visual Studio, and select **Add Reference**.
3. On the Browse tab, browse for the following libraries: **Altova.dll**, **AltovaXML.dll**, and **YourSchema.dll** located in the output directory of the generated projects (for example, **bin\Debug**).



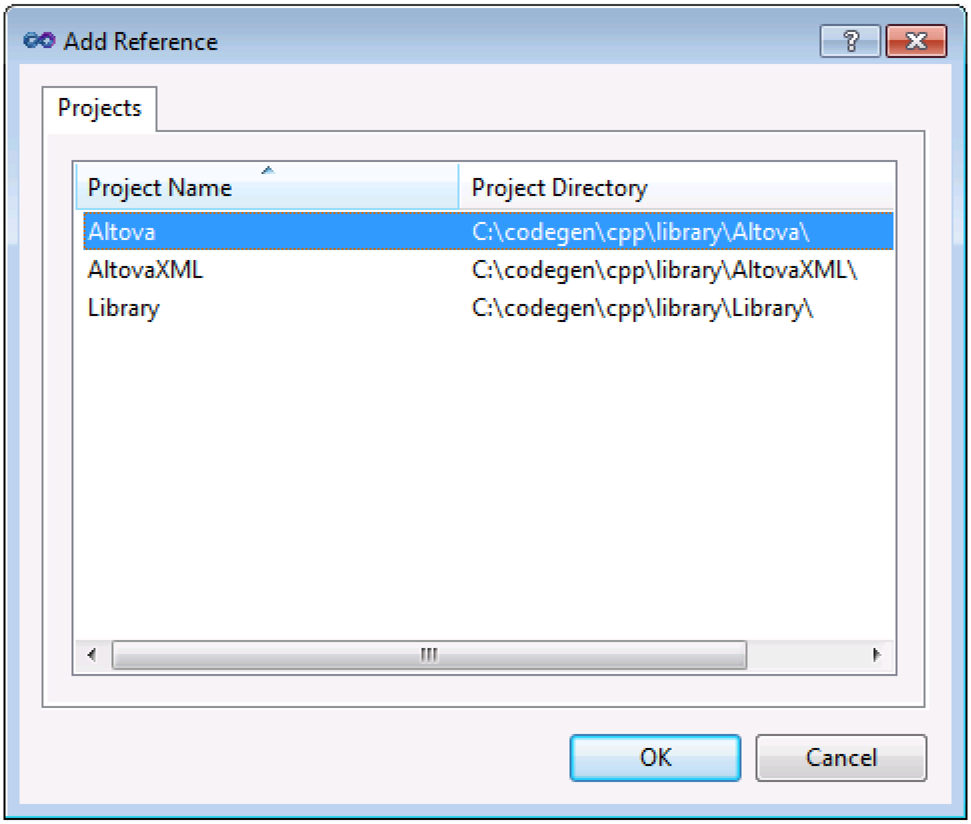
C++

The easiest way to integrate the libraries into an existing C++ project is to add the generated project files to your solution. For example, let's assume that you generated code from a schema called **Library.xsd** and selected **c:\codegen\cpp\library** as target directory. The generated libraries in this case are available at:

- c:\codegen\cpp\library\Altova.vcxproj
- c:\codegen\cpp\library\AltovaXML\AltovaXML.vcxproj
- c:\codegen\cpp\library\Library.vcxproj

First, open the generated **c:\codegen\cpp\library\Library.sln** solution and build it in Visual Studio.

Next, open your existing Visual Studio solution (in Visual Studio 2010, in this example), right-click it, select **Add | Existing Project**, and add the project files listed above, one by one. Be patient while Visual Studio parses the files. Next, right-click your project and select **Properties**. In the Property Pages dialog box, select **Common Properties | Framework and References**, and then click **Add New Reference**. Next, select and add each of the following projects: *Altova*, *AltovaXML*, and *Library*.



See also the MSDN documentation for using functionality from a custom library, as applicable to your version of Visual Studio, for example:

- If you chose to generate static libraries, see https://msdn.microsoft.com/en-us/library/ms235627(v=vs.100).aspx
- If you chose to generate dynamic libraries, see https://msdn.microsoft.com/en-us/library/ms235636(v=vs.100).aspx

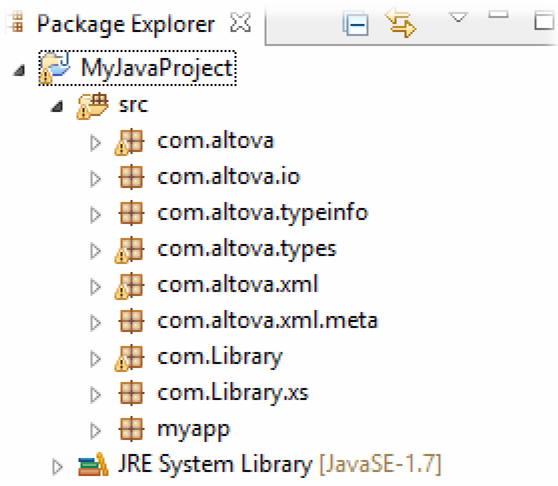
The option to generate static or dynamic libraries is available in code generation options (see Code generator options 1625).

Java

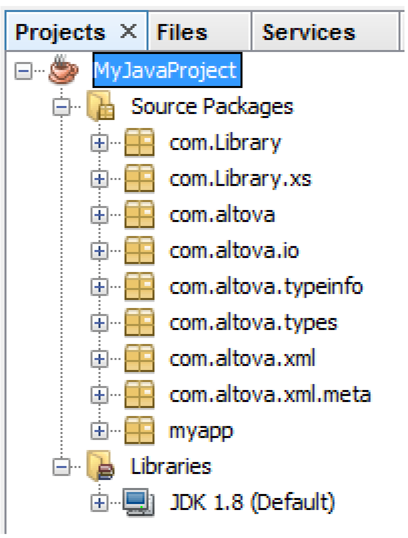
One of the ways to integrate the Altova packages into your Java project is to copy the **com** directory of the generated code to the directory which stores the source packages of your Java project (for example, **C:\Workspace\MyJavaProject\src**). For example, let's assume that you generated code in **c:\codegen\java\library**. The generated Altova classes in this case are available at **c:\codegen\java\library\com**.

After copying the libraries, refresh the project. To refresh the project in Eclipse, select it in the Package Explorer, and press **F5**. To refresh the project in NetBeans IDE 8.0, select the menu command **Source | Scan for External Changes**.

Once you perform the copy operation, the Altova packages are available in the Package Explorer (in case of Eclipse), or under "Source Packages" in the Projects pane (in case of NetBeans IDE).



Altova packages in Eclipse 4.4



Altova packages in NetBeans IDE 8.0.2

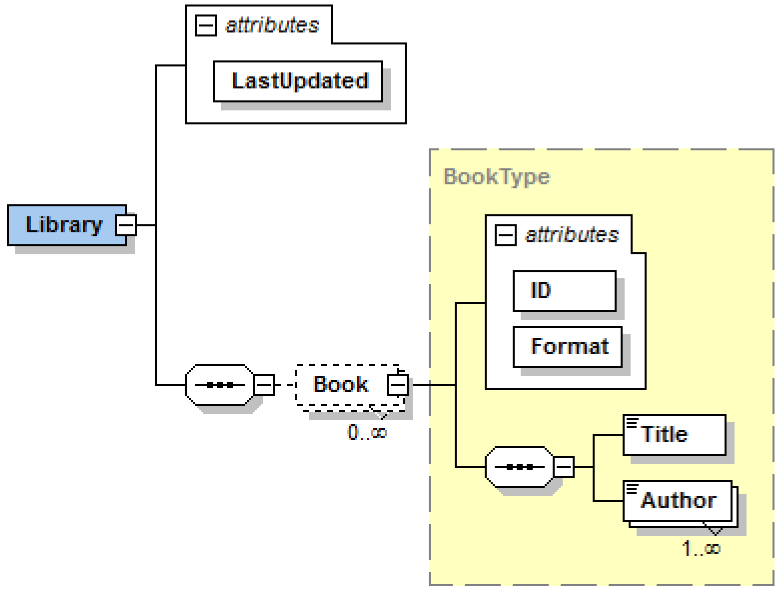
16.6.5 Example: Book Library

This example illustrates how to use the generated schema wrapper libraries in order to write or read programmatically XML documents conformant to the schema. Before using the sample code, take some time to understand the structure of the schema below.

The schema used in this example describes a library of books. The complete definition of the schema is shown below. Save this code listing as **`Library.xsd`** if you want to get the same results as this example. You will need this schema to generate the code libraries used in this example.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.nanonull.com/LibrarySample"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.nanonull.com/LibrarySample" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:element name="Library">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Book" type="BookType" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="LastUpdated" type="xs:dateTime"/>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="BookType">
      <xs:sequence>
         <xs:element name="Title" type="xs:string"/>
         <xs:element name="Author" type="xs:string" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="ID" type="xs:integer" use="required"/>
      <xs:attribute name="Format" type="BookFormatType" use="required"/>
   </xs:complexType>
   <xs:complexType name="DictionaryType">
      <xs:complexContent>
         <xs:extension base="BookType">
            <xs:sequence>
               <xs:element name="FromLang" type="xs:string"/>
               <xs:element name="ToLang" type="xs:string"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:simpleType name="BookFormatType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Hardcover"/>
         <xs:enumeration value="Paperback"/>
         <xs:enumeration value="Audiobook"/>
         <xs:enumeration value="E-book"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

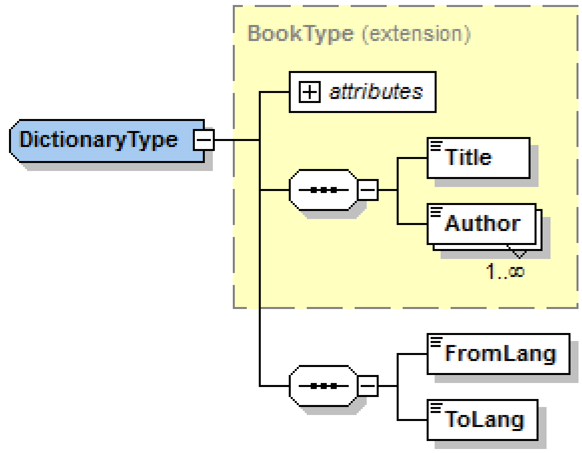
Library is a root element of a `complexType` which can be graphically represented as follows in the schema view of XMLSpy:



As shown above, the library has a **LastUpdated** attribute (defined as `xs:dateTime`), and stores a sequence of books. Each book is an `xs:complexType` and has two attributes: an **ID** (defined as `xs:integer`), and a **Format**. The format of any book can be hardcover, paperback, audiobook, or e-book. In the schema, **Format** is defined as `xs:simpleType` which uses an enumeration of the above-mentioned values.

Each book also has a **Title** element (defined as `xs:string`), as well as one or several **Author** elements (defined as `xs:string`).

The library may also contain books that are dictionaries. Dictionaries have the type `DictionaryType`, which is derived by extension from the `BookType`. In other words, a dictionary inherits all attributes and elements of a Book, plus two additional elements: **FromLang** and **ToLang**, as illustrated below.



The **FromLang** and **ToLang** elements store the source and destination language of the dictionary.

An XML instance file valid according to the schema above could therefore look as shown in the listing below (provided that it is in the same directory as the schema file):

```
<?xml version="1.0" encoding="utf-8"?>
<Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.nanonull.com/LibrarySample"
xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" LastUpdated="2016-
02-03T17:10:08.4977404">
   <Book ID="1" Format="E-book">
      <Title>The XMLSpy Handbook</Title>
      <Author>Altova</Author>
   </Book>
   <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample"
xsi:type="n1:DictionaryType">
      <Title>English-German Dictionary</Title>
      <Author>John Doe</Author>
      <FromLang>English</FromLang>
      <ToLang>German</ToLang>
   </Book>
</Library>
```

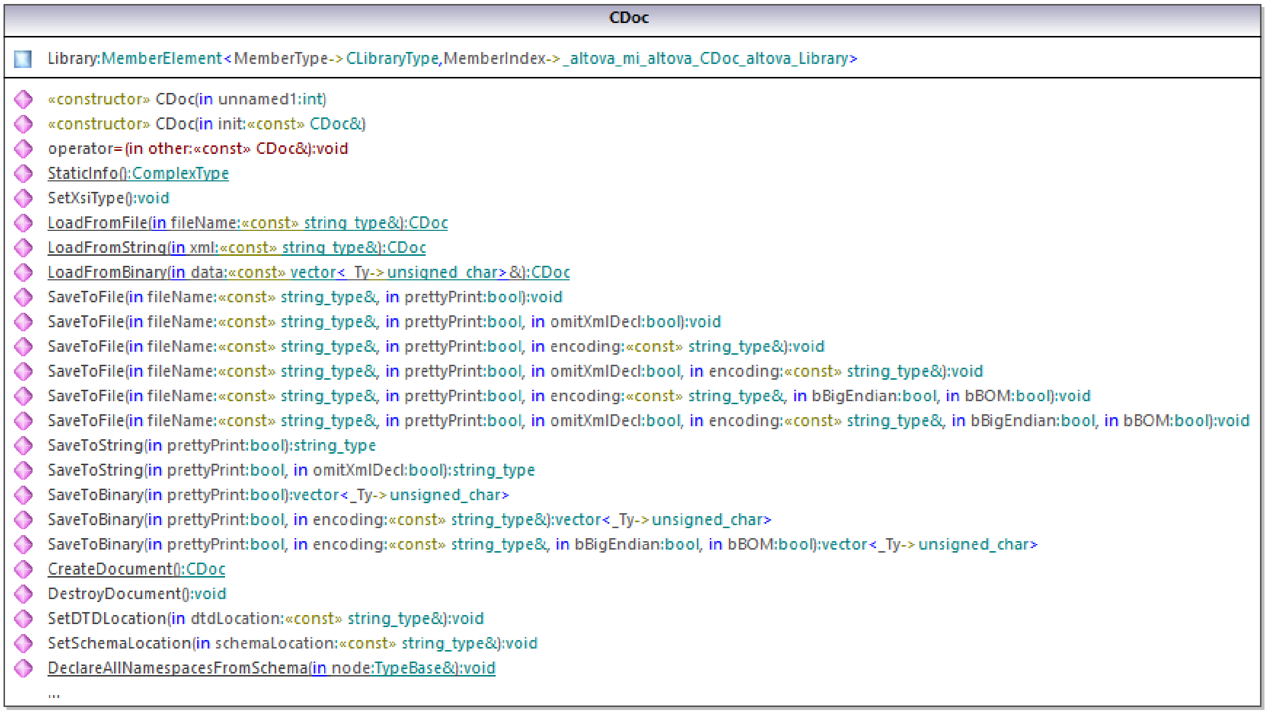
The next topics illustrate how to read from such a file programmatically, or write to such a file programmatically. To begin, generate the schema wrapper code from the schema above, using the steps described in Generating Code from XML Schemas or DTD[1535].

16.6.5.1 Reading and Writing XML Documents (C++)

After you generate code from the example schema[1547], a test C++ application is created, along with several supporting Altova libraries.

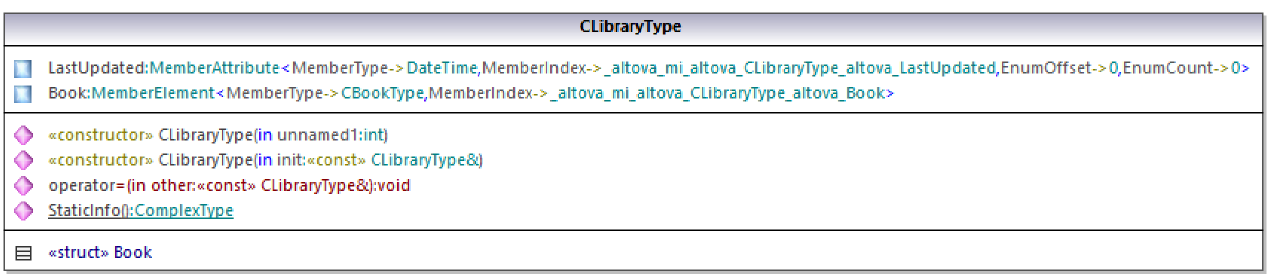
About the generated C++ libraries

The central class of the generated code is the `CDoc` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name. As shown in the diagram, this class provides methods for loading documents from files, binary streams, or strings (or saving documents to files, streams, strings). For a description of all members exposed by this class, see the class reference (`[YourSchema]::[CDoc]`[1589]).

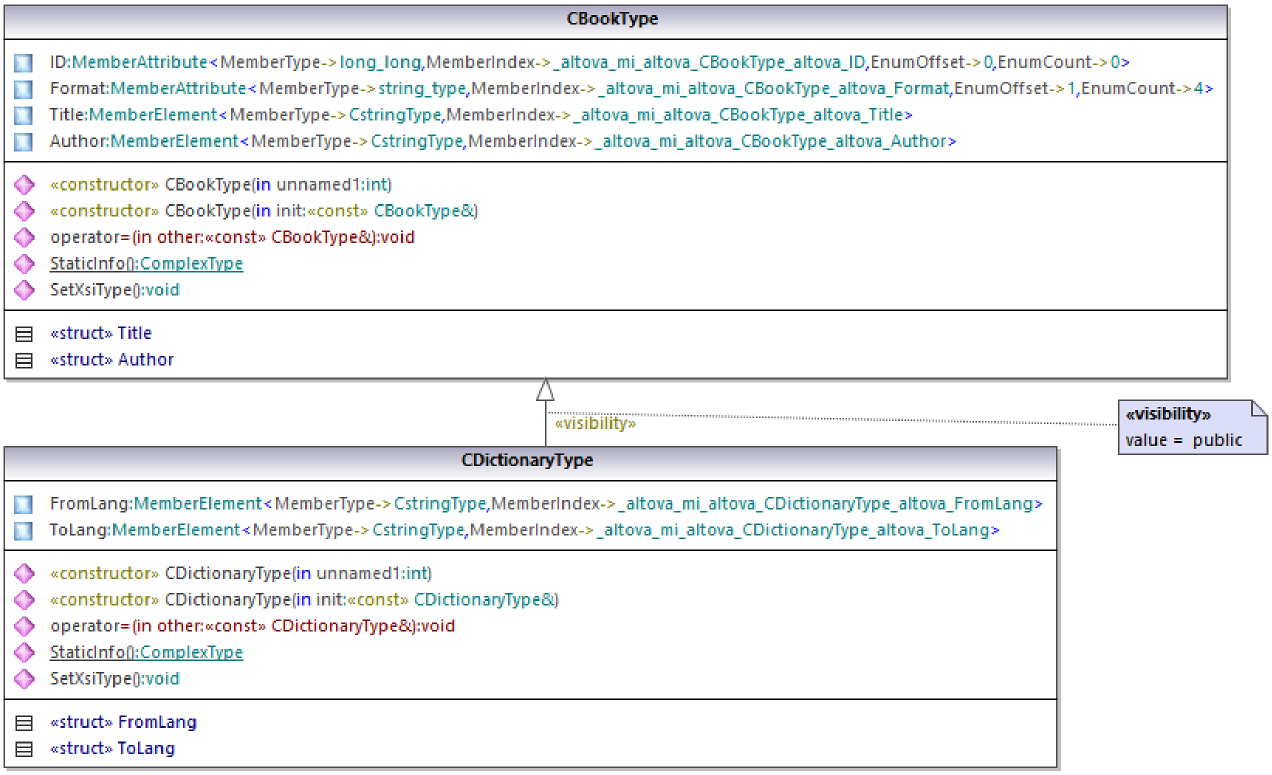


The `Library` field of the `CDoc` class represents the actual root of the document. **Library** is an element in the XML file, so in the C++ code it has a template class as type (`MemberElement`). The template class exposes methods and properties for interacting with the **Library** element. In general, each attribute and each element of a type in the schema is typed in the generated code with the `MemberAttribute` and `MemberElement` template classes, respectively. For more information, see [`[YourSchema]::MemberAttribute`] 1592 and [`[YourSchema]::MemberElement`] 1593 class reference.

The class `CLibraryType` is generated from the **LibraryType** complex type in the schema. Notice that the `CLibraryType` class contains two fields: `Book` and `LastUpdated`. According to the logic already mentioned above, these correspond to the **Book** element and **LastUpdated** attribute in the schema, and enable you to manipulate programmatically (append, remove, etc) elements and attributes in the instance XML document.



The `DictionaryType` is a complex type derived from **BookType** in the schema, so this relationship is also reflected in the generated classes. As illustrated in the diagram, the class `CDictionaryType` inherits the `CBookType` class.



If your XML schema defines simple types as enumerations, the enumerated values become available as `enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `enum` that is a member of the `CBookFormatType` class.

Writing an XML document

1. Open the **LibraryTest.sln** solution in Visual Studio generated from the Library schema mentioned earlier in this example.

> While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore, do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see Integrating Schema Wrapper Libraries 1544).

2. In Solution Explorer, open the **LibraryTest.cpp** file, and edit the `Example()` method as shown below.

```
#include <ctime> // required to get current time
using namespace Doc; // required to work with Altova libraries

void Example()
{
```

```
   // Create a new, empty XML document
   CDoc libDoc = CDoc::CreateDocument();

   // Create the root element <Library> and add it to the document
   CLibraryType lib = libDoc.Library.append();

   // Get current time and set the "LastUpdated" attribute using Altova classes
   time_t t = time(NULL);
   struct tm * now = localtime( & t );
   altova::DateTime dt = altova::DateTime(now->tm_year + 1900, now->tm_mon + 1, now-
>tm_mday, now->tm_hour, now->tm_min, now->tm_sec);
   lib.LastUpdated = dt;

   // Create a new <Book> and add it to the library
   CBookType book = lib.Book.append();

   // Set the "ID" attribute of the book
   book.ID = 1;

   // Set the "Format" attribute of the <Book> using an enumeration constant
   book.Format.SetEnumerationValue( CBookFormatType::k_Paperback );

   // Add the <Title> and <Author> elements, and set values
   book.Title.append() = _T("The XML Spy Handbook");
   book.Author.append() = _T("Altova");

   // Append a dictionary (book of derived type) and populate its attributes and elements
   CDictionaryType dictionary = CDictionaryType(lib.Book.append().GetNode());
   dictionary.ID = 2;
   dictionary.Format.SetEnumerationValue( CBookFormatType::k_E_book);
   dictionary.Title.append() = _T("English-German Dictionary");
   dictionary.Author.append() = _T("John Doe");
   dictionary.FromLang.append() = _T("English");
   dictionary.ToLang.append() = _T("German");

   // Since dictionary a derived type, set the xsi:type attribute of the book element
   dictionary.SetXsiType();

   // Optionally, set the schema location
   libDoc.SetSchemaLocation(_T("Library.xsd"));

   // Save the XML document to a file with default encoding (UTF-8),
   // "true" causes the file to be pretty-printed.
   libDoc.SaveToFile(_T("GeneratedLibrary.xml"), true);

   // Destroy the document
   libDoc.DestroyDocument();
}
```

3. Press **F5** to start debugging. If the code was executed successfully, a **GeneratedLibrary.xml** file is created in the solution output directory.

Reading an XML document

1. Open the **LibraryTest.sln** solution in Visual Studio.
2. Save the code below as **Library1.xml** to a directory that can be read by the program code (for example, the same directory as **LibraryTest.sln**).

```xml
<?xml version="1.0" encoding="utf-8"?>
<Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.nanonull.com/LibrarySample"
xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" LastUpdated="2016-
02-03T17:10:08.4977404">
   <Book ID="1" Format="E-book">
      <Title>The XMLSpy Handbook</Title>
      <Author>Altova</Author>
   </Book>
   <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample"
xsi:type="n1:DictionaryType">
      <Title>English-German Dictionary</Title>
      <Author>John Doe</Author>
      <FromLang>English</FromLang>
      <ToLang>German</ToLang>
   </Book>
</Library>
```

3. In Solution Explorer, open the **LibraryTest.cpp** file, and edit the `Example()` method as shown below.

```cpp
using namespace Doc;
void Example()
{
   // Load XML document
   CDoc libDoc = CDoc::LoadFromFile(_T("Library1.xml"));

   // Get the first (and only) root element <Library>
   CLibraryType lib = libDoc.Library.first();

   // Check whether an element exists:
   if (!lib.Book.exists())
   {
      tcout << "This library is empty." << std::endl;
      return;
   }

   // iteration: for each <Book>...
   for (Iterator<CBookType> itBook = lib.Book.all(); itBook; ++itBook)
   {
      // output values of ISBN attribute and (first and only) title element
      tcout << "ID: " << itBook->ID << std::endl;
      tcout << "Title: " << tstring(itBook->Title.first()) << std::endl;

      // read and compare an enumeration value
      if (itBook->Format.GetEnumerationValue() == CBookFormatType::k_Paperback)
```



```
         tcout << "This is a paperback book." << std::endl;

      // for each <Author>...
      for (CBookType::Author::iterator itAuthor = itBook->Author.all(); itAuthor; +
+itAuthor)
         tcout << "Author: " << tstring(itAuthor) << std::endl;

      // alternative: use count and index
      for (unsigned int j = 0; j < itBook->Author.count(); ++j)
         tcout << "Author: " << tstring(itBook->Author[j]) << std::endl;
   }

   // Destroy the document
   libDoc.DestroyDocument();
}
```

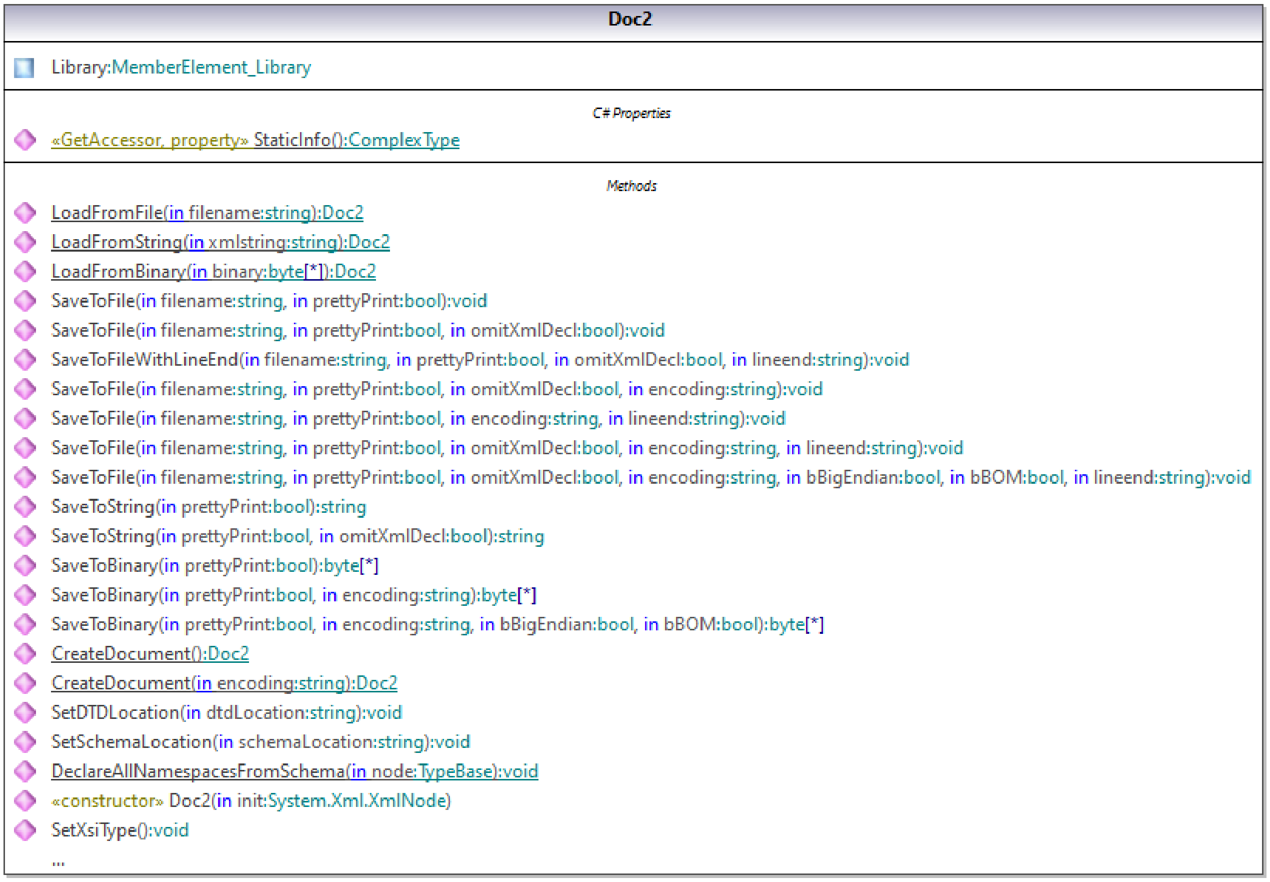
4. Press **F5** to start debugging.

16.6.5.2 Reading and Writing XML Documents (C#)

After you generate code from the example schema1547, a test C# application is created, along with several supporting Altova libraries.

About the generated C# libraries

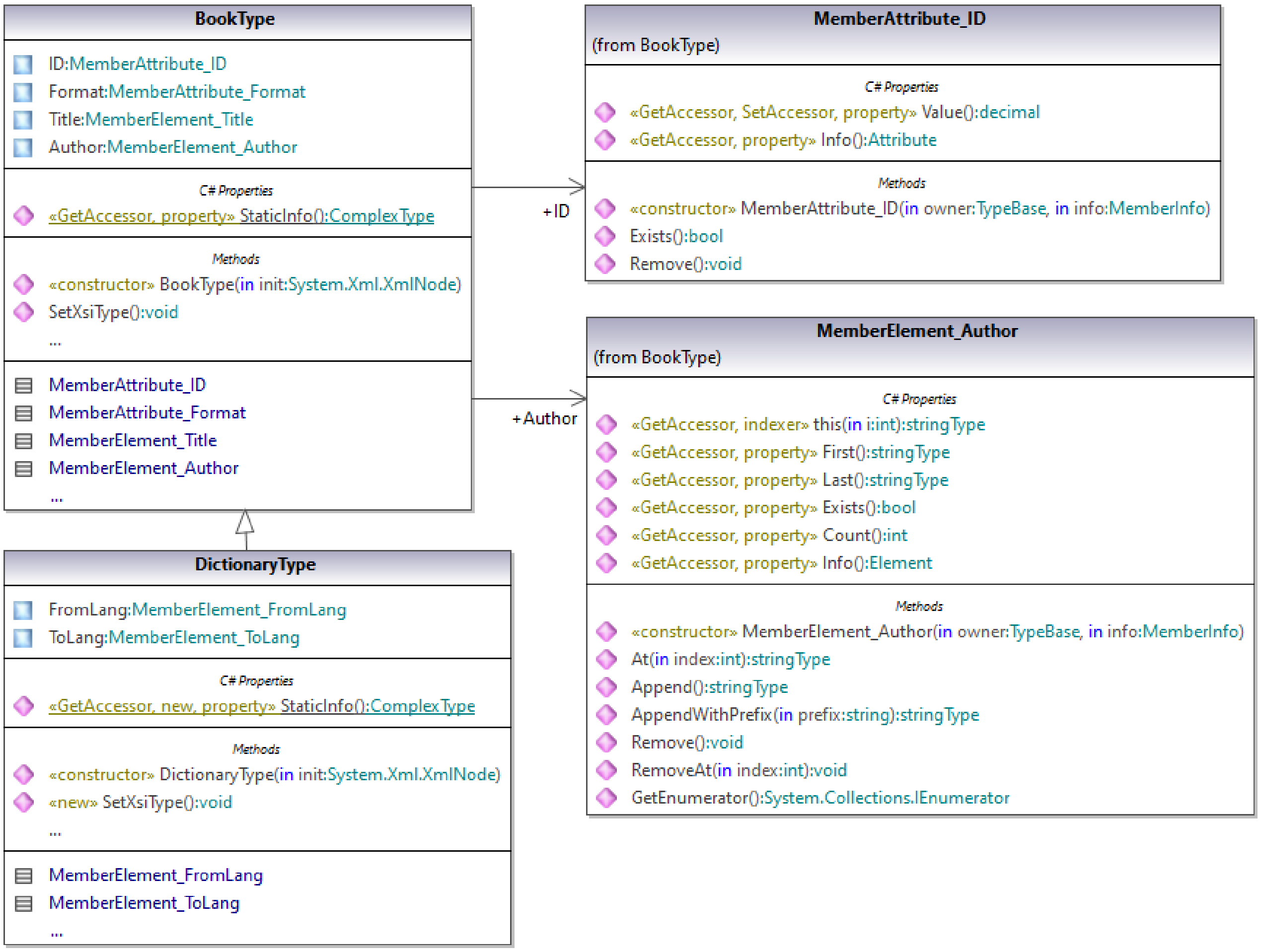
The central class of the generated code is the `Doc2` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name. Note that this class is called `Doc2` to avoid a possible conflict with the namespace name. As shown in the diagram, this class provides methods for loading documents from files, binary streams, or strings (or saving documents to files, streams, strings). For a description of this class, see the class reference ([YourSchema].[Doc]1604).



The `Library` member of the `Doc2` class represents the actual root of the document.

According to the code generation rules mentioned in About Schema Wrapper Libraries (C#) 1540, member classes are generated for each attribute and for each element of a type. In the generated code, the name of such member classes is prefixed with `MemberAttribute_` and `MemberElement_`, respectively. Examples of such classes are `MemberAttribute_ID` and `MemberElement_Author`, generated from the **Author** element and **ID** attribute of a book, respectively (in the diagram below, they are classes nested under `BookType`). Such classes enable you to manipulate programmatically the corresponding elements and attributes in the instance XML document (for example, append, remove, set value, etc). For more information, see the `[YourSchemaType].MemberAttribute` 1607 and `[YourSchemaType].MemberElement` 1607 class reference.

Since the **DictionaryType** is a complex type derived from **BookType** in the schema, this relationship is also reflected in the generated classes. As illustrated in the diagram below, the class `DictionaryType` inherits the `BookType` class.



If your XML schema defines simple types as enumerations, the enumerated values become available as `Enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `Enum` that is a member of the `BookFormatType` class.

Writing an XML document

1. Open the **LibraryTest.sln** solution in Visual Studio generated from the Library schema mentioned earlier in this example.

> While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore, do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see Integrating Schema Wrapper Libraries 1544).

2. In Solution Explorer, open the **LibraryTest.cs** file, and edit the `Example()` method as shown below.

```
protected static void Example()
{
    // Create a new XML document
    Doc2 doc = Doc2.CreateDocument();
    // Append the root element
    LibraryType root = doc.Library.Append();

    // Create the generation date using Altova DateTime class
    Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);
    // Append the date to the root
    root.LastUpdated.Value = dt;

    // Add a new book
    BookType book = root.Book.Append();
    // Set the value of the ID attribute
    book.ID.Value = 1;
    // Set the format of the book (enumeration)
    book.Format.EnumerationValue = BookFormatType.EnumValues.eHardcover;
    // Set the Title and Author elements
    book.Title.Append().Value = "The XMLSpy Handbook";
    book.Author.Append().Value = "Altova";

    // Append a dictionary (book of derived type) and populate its attributes and
elements
    DictionaryType dictionary = new DictionaryType(root.Book.Append().Node);
    dictionary.ID.Value = 2;
    dictionary.Title.Append().Value = "English-German Dictionary";
    dictionary.Format.EnumerationValue = BookFormatType.EnumValues.eE_book;
    dictionary.Author.Append().Value = "John Doe";
    dictionary.FromLang.Append().Value = "English";
    dictionary.ToLang.Append().Value = "German";
    // Since it's a derived type, make sure to set the xsi:type attribute of the
book element
    dictionary.SetXsiType();

    // Optionally, set the schema location (adjust the path if
    // your schema is not in the same folder as the generated instance file)
    doc.SetSchemaLocation("Library.xsd");

    // Save the XML document with the "pretty print" option enabled
    doc.SaveToFile("GeneratedLibrary.xml", true);
}
```

3. Press **F5** to start debugging. If the code was executed successfully, a **GeneratedLibrary.xml** file is created in the solution output directory (typically, **bin/Debug**).

Reading an XML document

1. Open the **LibraryTest.sln** solution in Visual Studio.
2. Save the code below as **Library.xml** to the output directory of the project (by default, **bin/Debug**). This is the file that will be read by the program code.

```
<?xml version="1.0" encoding="utf-8"?>
```



```
<Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.nanonull.com/LibrarySample"
xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" LastUpdated="2016-
02-03T17:10:08.4977404">
   <Book ID="1" Format="E-book">
      <Title>The XMLSpy Handbook</Title>
      <Author>Altova</Author>
   </Book>
   <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample"
xsi:type="n1:DictionaryType">
      <Title>English-German Dictionary</Title>
      <Author>John Doe</Author>
      <FromLang>English</FromLang>
      <ToLang>German</ToLang>
   </Book>
</Library>
```

3. In Solution Explorer, open the **LibraryTest.cs** file, and edit the `Example()` method as shown below.

```
protected static void Example()
{
    // Load the XML file
    Doc2 doc = Doc2.LoadFromFile("Library.xml");
    // Get the root element
    LibraryType root = doc.Library.First;

    // Read the library generation date
    Altova.Types.DateTime dt = root.LastUpdated.Value;
    string dt_as_string = dt.ToString(DateTimeFormat.W3_dateTime);
    Console.WriteLine("The library generation date is: " + dt_as_string);

    // Iteration: for each <Book>...
    foreach (BookType book in root.Book)
    {
      // Output values of ID attribute and (first and only) title element
      Console.WriteLine("ID:    " + book.ID.Value);
      Console.WriteLine("Title: " + book.Title.First.Value);

      // Read and compare an enumeration value
      if (book.Format.EnumerationValue == BookFormatType.EnumValues.ePaperback)
         Console.WriteLine("This is a paperback book.");

      // Iteration: for each <Author>
      foreach (xs.stringType author in book.Author)
         Console.WriteLine("Author: " + author.Value);

      // Determine if this book is of derived type
      if (book.Node.Attributes.GetNamedItem("xsi:type") != null)
      {
         // Find the value of the xsi:type attribute
         string xsiTypeValue =
book.Node.Attributes.GetNamedItem("xsi:type").Value;
```

```
         // Get the namespace URI and the lookup prefix of this namespace
         string namespaceUri = book.Node.NamespaceURI;
         string prefix = book.Node.GetPrefixOfNamespace(namespaceUri);

         // if this book has DictionaryType
         if (namespaceUri == "http://www.nanonull.com/LibrarySample" &&
xsiTypeValue.Equals(prefix + ":DictionaryType"))
         {
            // output additional fields
            DictionaryType dictionary = new DictionaryType(book.Node);
            Console.WriteLine("Language from: " +
dictionary.FromLang.First.Value);
            Console.WriteLine("Language to: " + dictionary.ToLang.First.Value);
         }
         else
         {
            throw new Exception("Unexpected book type");
         }
      }
    }

    Console.ReadLine();
}
```

4. Press **F5** to start debugging. If the code was executed successfully, **Library.xml** will be read by the program code, and its contents displayed as console output.

Reading and writing elements and attributes

Values of attributes and elements can be accessed using the `Value` property of the generated member element or attribute class, for example:

```
// Output values of ID attribute and (first and only) title element
Console.WriteLine("ID:    " + book.ID.Value);
Console.WriteLine("Title: " + book.Title.First.Value);
```

To get the value of the **Title** element in this particular example, we also used the `First()` method, since this is the first (and only) **Title** element of a book. For cases when you need to pick a specific element from a list by index, use the `At()` method.

The class generated for each member element of a type implements the standard `System.Collections.IEnumerable` interface. This makes it possible to loop through multiple elements of the same type. In this particular example, you can loop through all books of a Library object as follows:

```
// Iteration: for each <Book>...
foreach (BookType book in root.Book)
{
    // your code here...
}
```


To add a new element, use the `Append()` method. For example, the following code appends the root element to the document:

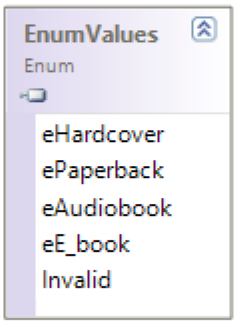
```
// Append the root element to the library
LibraryType root = doc.Library.Append();
```

You can set the value of an attribute (like ID in this example) as follows:

```
// Set the value of the ID attribute
book.ID.Value = 1;
```

Reading and writing enumeration values

If your XML schema defines simple types as enumerations, the enumerated values become available as `Enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `Enum`:



To assign enumeration values to an object, use code such as the one below:

```
// Set the format of the book (enumeration)
book.Format.EnumerationValue = BookFormatType.EnumValues.eHardcover;
```

You can read such enumeration values from XML instance documents as follows:

```
// Read and compare an enumeration value
if (book.Format.EnumerationValue == BookFormatType.EnumValues.ePaperback)
Console.WriteLine("This is a paperback book.");
```

When an "if" condition is not enough, create a switch to determine each enumeration value and process it as required.

Working with xs:dateTime and xs:duration types

If the schema from which you generated code uses time and duration types such as `xs:dateTime`, or `xs:duration`, these are converted to Altova native classes in generated code. Therefore, to write a date or duration value to the XML document, do the following:

1. Construct an Altova.Types.DateTime 1594 or Altova.Types.Duration 1598 object (either from System.DateTime, or by using parts such as hours and minutes, see Altova.Types.DateTime 1594 and Altova.Types.Duration 1598 for more information).
2. Set the object as value of the required element or attribute, for example:

```
// Create the library generation date using Altova DateTime class
Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);
// Append the date to the root
root.LastUpdated.Value = dt;
```

To read a date or duration from an XML document, do the following:

1. Declare the element value (or attribute) as Altova.Types.DateTime 1594 or Altova.Types.Duration 1598 object.
2. Format the required element or attribute, for example:

```
// Read the library generation date
Altova.Types.DateTime dt = root.LastUpdated.Value;
string dt_as_string = dt.ToString(DateTimeFormat.W3_dateTime);
Console.WriteLine("The library generation date is: " + dt_as_string);
```

For more information, see Altova.Types.DateTime 1594 and Altova.Types.Duration 1598 class reference.

Working with derived types

If your XML schema defines derived types, you can preserve type derivation in XML documents that you create or load programmatically. Taking the schema used in this example, the following code listing illustrates how to create a new book of derived type DictionaryType:

```
// Append a dictionary (book of derived type) and populate its attributes and elements
DictionaryType dictionary = new DictionaryType(root.Book.Append().Node);
dictionary.ID.Value = 2;
dictionary.Title.Append().Value = "English-German Dictionary";
dictionary.Author.Append().Value = "John Doe";
dictionary.FromLanguage.Append().Value = "English";
dictionary.ToLanguage.Append().Value = "German";

// Since it's a derived type, make sure to set the xsi:type attribute of the book element
dictionary.SetXsiType();
```

Note that it is important to set the xsi:type attribute of the newly created book. This ensures that the book type will be interpreted correctly by the schema when the XML document is validated.

When you load data from an XML document, the following code listing shows how to identify a book of derived type DictionaryType in the loaded XML instance. First, the code finds the value of the xsi:type attribute of the book node. If the namespace URI of this node is http://www.nanonull.com/LibrarySample, and if the URI lookup prefix and type matches the value of the xsi:type attribute, then this is a dictionary:

```
      // Determine if this book is of derived type
      if (book.Node.Attributes.GetNamedItem("xsi:type") != null)
```

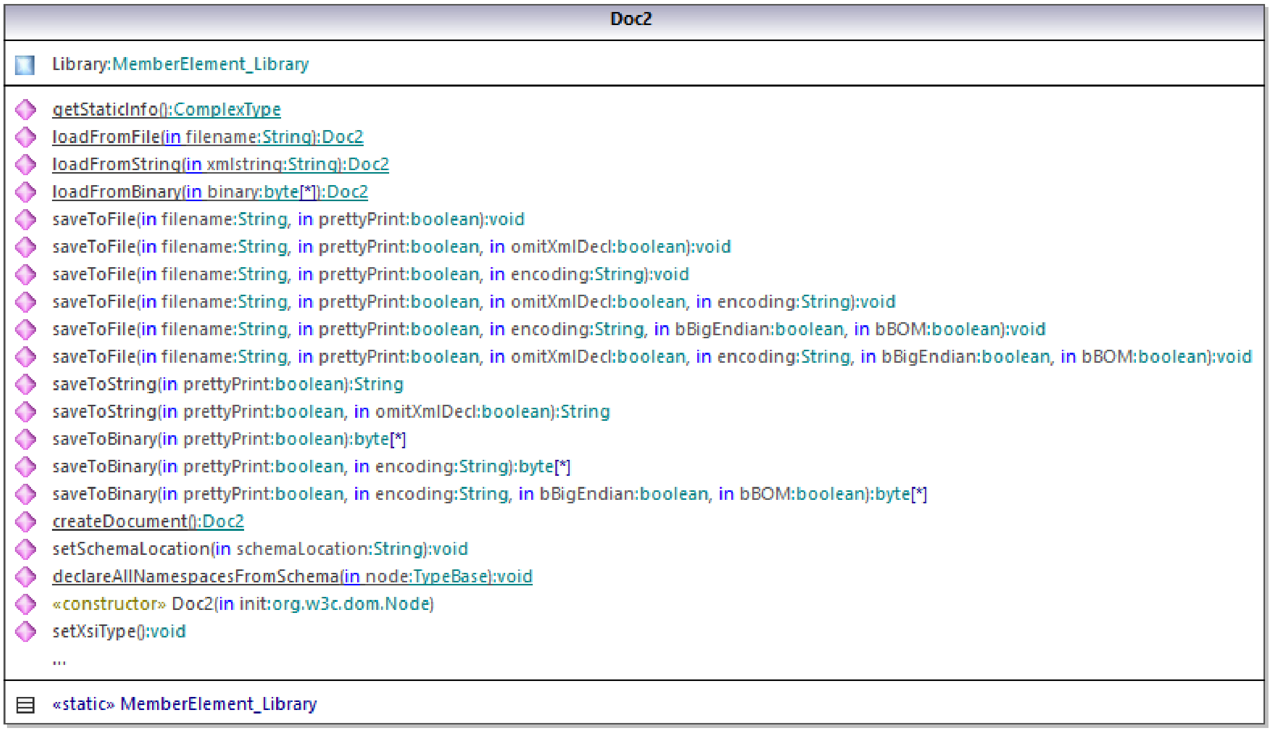
```
      {
          // Find the value of the xsi:type attribute
          string xsiTypeValue = book.Node.Attributes.GetNamedItem("xsi:type").Value;
          // Get the namespace URI and the lookup prefix of this namespace
          string namespaceUri = book.Node.NamespaceURI;
          string prefix = book.Node.GetPrefixOfNamespace(namespaceUri);

          // if this book has DictionaryType
          if (namespaceUri == "http://www.nanonull.com/LibrarySample" &&
xsiTypeValue.Equals(prefix + ":DictionaryType"))
          {
             // output additional fields
             DictionaryType dictionary = new DictionaryType(book.Node);
             Console.WriteLine("Language from: " + dictionary.FromLang.First.Value);
             Console.WriteLine("Language to: " + dictionary.ToLang.First.Value);
          }
          else
          {
              throw new Exception("Unexpected book type");
          }
      }
```

16.6.5.3 Reading and Writing XML Documents (Java)

After you generate code from the example schema 1547, a test Java project is created, along with several supporting Altova libraries.

About the generated Java libraries

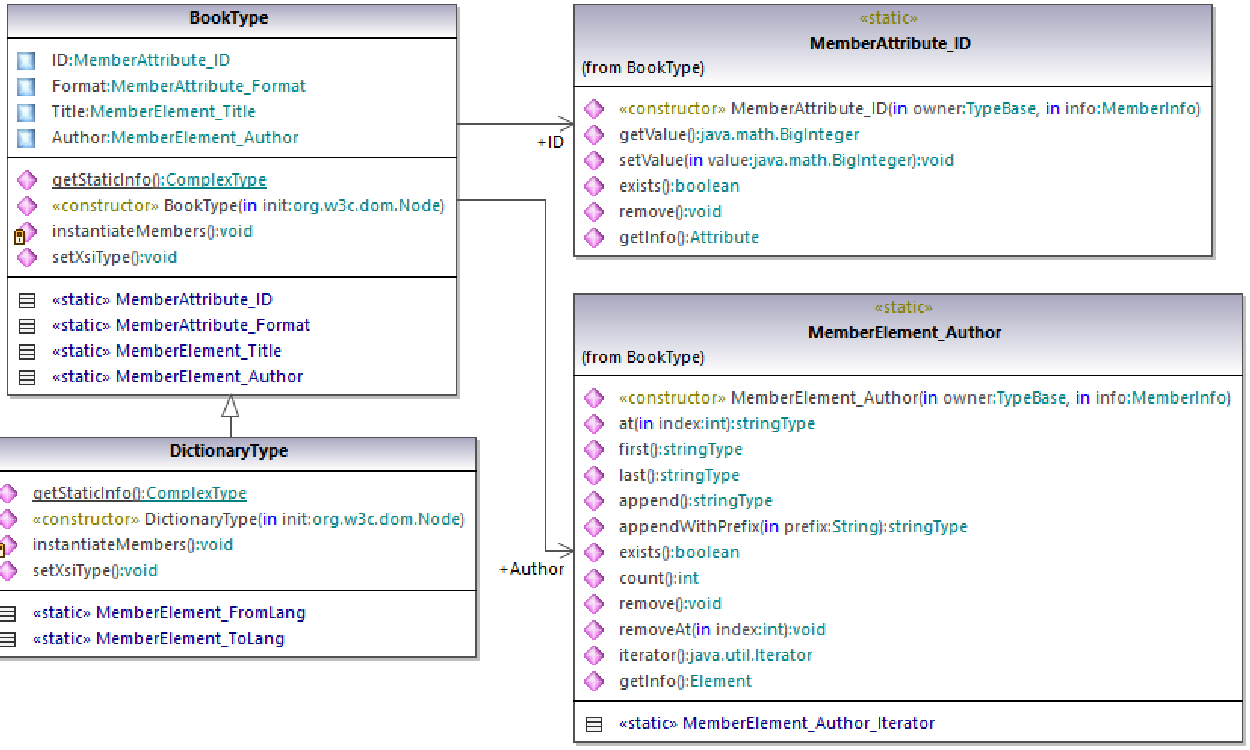
The central class of the generated code is the `Doc2` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name. Note that this class is called `Doc2` to avoid a possible conflict with the namespace name. As shown in the diagram, this class provides methods for loading documents from files, binary streams, or strings (or saving documents to files, streams, strings). For a description of this class, see the `com.[YourSchema].[Doc]` 1619 class reference.



The `Library` member of the `Doc2` class represents the actual root of the document.

According to the code generation rules mentioned in [About Generated Java Code](1542), member classes are generated for each attribute and for each element of a type. In the generated code, the name of such member classes is prefixed with `MemberAttribute_` and `MemberElement_`, respectively. In the diagram below, examples of such classes are `MemberAttribute_ID` and `MemberElement_Author`, generated from the **Author** element and **ID** attribute of a book, respectively. Such classes enable you to manipulate programmatically the corresponding elements and attributes in the instance XML document (for example, append, remove, set value, etc). For more information, see the `com.[YourSchema].[YourSchemaType].MemberAttribute` [1622] and `com.[YourSchema].[YourSchemaType].MemberElement` [1622] class reference.

Since the **DictionaryType** is a complex type derived from **BookType** in the schema, this relationship is also reflected in the generated classes. As illustrated in the diagram below, the class `DictionaryType` inherits the `BookType` class.



If your XML schema defines simple types as enumerations, the enumerated values become available as `Enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `Enum` that is a member of the `BookFormatType` class.

Writing an XML document

1. On the **File** menu of Eclipse, click **Import**, select **Existing Projects into Workspace**, and click **Next**.
2. Next to **Select root directory**, click **Browse**, select the directory to which you generated the Java code, and then click **Finish**.
3. In the Eclipse Package Explorer, expand the **com.LibraryTest** package and open the **LibraryTest.java** file.

While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore, do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see Integrating Schema Wrapper Libraries [1544]).

4. Edit the `Example()` method as shown below.

```
protected static void example() throws Exception {
      // create a new, empty XML document
```

```
      Doc2 libDoc = Doc2.createDocument();

      // create the root element <Library> and add it to the document
      LibraryType lib = libDoc.Library.append();

      // set the "LastUpdated" attribute
      com.altova.types.DateTime dt = new com.altova.types.DateTime(DateTime.now());
      lib.LastUpdated.setValue(dt);

      // create a new <Book> and populate its elements and attributes
      BookType book = lib.Book.append();
      book.ID.setValue(java.math.BigInteger.valueOf(1));
      book.Format.setEnumerationValue( BookFormatType.EPAPERBACK );
      book.Title.append().setValue("The XML Spy Handbook");
      book.Author.append().setValue("Altova");

      // create a dictionary (book of derived type)  and populate its elements and
attributes
      DictionaryType dict = new DictionaryType(lib.Book.append().getNode());
      dict.ID.setValue(java.math.BigInteger.valueOf(2));
      dict.Title.append().setValue("English-German Dictionary");
      dict.Format.setEnumerationValue(BookFormatType.EE_BOOK);
      dict.Author.append().setValue("John Doe");
      dict.FromLang.append().setValue("English");
      dict.ToLang.append().setValue("German");
      dict.setXsiType();

      // set the schema location (this is optional)
      libDoc.setSchemaLocation("Library.xsd");

      // save the XML document to a file with default encoding (UTF-8). "true" causes the
file to be pretty-printed.
      libDoc.saveToFile("Library1.xml", true);
   }
```

5. Build the Java project and run it. If the code is executed successfully, a **Library1.xml** file is created in the project directory.

Reading an XML document

1. On the **File** menu of Eclipse, click **Import**, select **Existing Projects into Workspace**, and click **Next**.
2. Next to **Select root directory**, click **Browse**, select the directory to which you generated the Java code, and then click **Finish**.
3. Save the code below as **Library1.xml** to a local directory (you will need to refer to the path of the **Library1.xml** file from the sample code below).

```
<?xml version="1.0" encoding="utf-8"?>
<Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.nanonull.com/LibrarySample"
xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" LastUpdated="2016-
02-03T17:10:08.4977404">
```



```
   <Book ID="1" Format="E-book">
      <Title>The XMLSpy Handbook</Title>
      <Author>Altova</Author>
   </Book>
   <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample"
xsi:type="n1:DictionaryType">
      <Title>English-German Dictionary</Title>
      <Author>John Doe</Author>
      <FromLang>English</FromLang>
      <ToLang>German</ToLang>
   </Book>
</Library>
```

4. In the Eclipse Package Explorer, expand the **com.LibraryTest** package and open the **LibraryTest.java** file.
5. Edit the `Example()` method as shown below.

```
protected static void example() throws Exception {
     // load XML document from a path, make sure to adjust the path as necessary
     Doc2 libDoc = Doc2.loadFromFile("Library1.xml");

     // get the first (and only) root element <Library>
     LibraryType lib = libDoc.Library.first();

     // check whether an element exists:
     if (!lib.Book.exists()) {
        System.out.println("This library is empty.");
        return;
     }

     // read a DateTime schema type
     com.altova.types.DateTime dt = lib.LastUpdated.getValue();
     System.out.println("The library was last updated on: " + dt.toDateString());

     // iteration: for each <Book>...
     for (java.util.Iterator itBook = lib.Book.iterator(); itBook.hasNext();) {
        BookType book = (BookType) itBook.next();
        // output values of ID attribute and (first and only) title element
        System.out.println("ID: " + book.ID.getValue());
        System.out.println("Title: " + book.Title.first().getValue());

        // read and compare an enumeration value
        if (book.Format.getEnumerationValue() == BookFormatType.EPAPERBACK)
           System.out.println("This is a paperback book.");

        // for each <Author>...
        for (java.util.Iterator itAuthor = book.Author.iterator(); itAuthor
              .hasNext();)
           System.out.println("Author: " + ((com.Doc.xs.stringType)
itAuthor.next()).getValue());

        // find the derived type of this book
```

```
        // by looking at the value of the xsi:type attribute, using DOM
        org.w3c.dom.Node bookNode = book.getNode();
        if (bookNode.getAttributes().getNamedItem("xsi:type") != null) {
           // Get the value of the xsi:type attribute
           String xsiTypeValue =
bookNode.getAttributes().getNamedItem("xsi:type").getNodeValue();

           // Get the namespace URI and lookup prefix of this namespace
           String namespaceUri = bookNode.getNamespaceURI();
           String lookupPrefix = bookNode.lookupPrefix(namespaceUri);

           // If xsi:type matches the namespace URI and type of the book node
           if (namespaceUri == "http://www.nanonull.com/LibrarySample"
                 && (  xsiTypeValue.equals(lookupPrefix + ":DictionaryType" )))    {
              // ...then this is a book of derived type (dictionary)
              DictionaryType dictionary = new DictionaryType(   book.getNode());
              // output the value of the "FromLang" and "ToLang" elements
              System.out.println("From language: " +
dictionary.FromLang.first().getValue());
              System.out.println("To language: " + dictionary.ToLang.first().getValue());
           }
           else
           {
              // throw an error
              throw new java.lang.Error("This book has an unknown type.");
           }
        }
     }
   }
```

6. Build the Java project and run it. If the code is executed successfully, **Library1.xml** will be read by the program code, and its contents displayed in the Console view.

Reading and writing elements and attributes

Values of attributes and elements can be accessed using the `getValue()` method of the generated member element or attribute class, for example:

```
// output values of ID attribute and (first and only) title element
System.out.println("ID: " + book.ID.getValue());
System.out.println("Title: " + book.Title.first().getValue());
```

To get the value of the **Title** element in this particular example, we also used the `first()` method, since this is the first (and only) **Title** element of a book. For cases when you need to pick a specific element from a list by index, use the `at()` method.

To iterate through multiple elements, use either index-based iteration or `java.util.Iterator`. For example, you can iterate through the books of a library as follows:

```
// index-based iteration
for (int j = 0; j < lib.Book.count(); ++j ) {
```



```
   // your code here
}

// alternative iteration using java.util.Iterator
for (java.util.Iterator itBook = lib.Book.iterator(); itBook.hasNext();) {
      // your code here
   }
```

To add a new element, use the `append()` method. For example, the following code appends an empty root **Library** element to the document:

```
// create the root element <Library> and add it to the document
LibraryType lib = libDoc.Library.append();
```

Once an element is appended, you can set the value of any of its elements or an attributes by using the setValue() method.

```
// set the value of the Title element
book.Title.append().setValue("The XML Spy Handbook");
// set the value of the ID attribute
book.ID.setValue(java.math.BigInteger.valueOf(1));
```

Reading and writing enumeration values

If your XML schema defines simple types as enumerations, the enumerated values become available as `Enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `Enum` (see the `BookFormatType` class diagram above). To assign enumeration values to an object, use code such as the one below:

```
// set an enumeration value
book.Format.setEnumerationValue( BookFormatType.EPAPERBACK );
```

You can read such enumeration values from XML instance documents as follows:

```
// read an enumeration value
if (book.Format.getEnumerationValue() == BookFormatType.EPAPERBACK)
         System.out.println("This is a paperback book."
```

When an "if" condition is not enough, create a switch to determine each enumeration value and process it as required.

Working with xs:dateTime and xs:duration types

If the schema from which you generated code uses time and duration types such as `xs:dateTime`, or `xs:duration`, these are converted to Altova native classes in generated code. Therefore, to write a date or duration value to the XML document, do the following:

1. Construct a `com.altova.types.DateTime` 1609 or `com.altova.types.Duration` 1613 object.

2. Set the object as value of the required element or attribute, for example:

```
// set the value of an attribute of DateTime type
com.altova.types.DateTime dt = new com.altova.types.DateTime(DateTime.now());
lib.LastUpdated.setValue(dt);
```

To read a date or duration from an XML document:

1. Declare the element value (or attribute) as `com.altova.types.DateTime` 1609 or `com.altova.types.Duration` 1613 object.
2. Format the required element or attribute, for example:

```
// read a DateTime type
com.altova.types.DateTime dt = lib.LastUpdated.getValue();
   System.out.println("The library was last updated on: " + dt.toDateString());
```

For more information, see `com.altova.types.DateTime` 1609 and `com.altova.types.Duration` 1613 class reference.

Working with derived types

If your XML schema defines derived types, you can preserve type derivation in XML documents that you create or load programmatically. Taking the schema used in this example, the following code listing illustrates how to create a new book of derived type `DictionaryType`:

```
// create a dictionary (book of derived type)  and populate its elements and attributes
DictionaryType dict = new DictionaryType(lib.Book.append().getNode());
dict.ID.setValue(java.math.BigInteger.valueOf(2));
dict.Title.append().setValue("English-German Dictionary");
dict.Format.setEnumerationValue(BookFormatType.EE_BOOK);
dict.Author.append().setValue("John Doe");
dict.FromLang.append().setValue("English");
dict.ToLang.append().setValue("German");
dict.setXsiType();
```

Note that it is important to set the `xsi:type` attribute of the newly created book. This ensures that the book type will be interpreted correctly by the schema when the XML document is validated.

When you load data from an XML document, the following code listing shows how to identify a book of derived type `DictionaryType` in the loaded XML instance. First, the code finds the value of the `xsi:type` attribute of the book node. If the namespace URI of this node is `http://www.nanonull.com/LibrarySample`, and if the URI lookup prefix and type matches the value of the `xsi:type` attribute, then this is a dictionary:

```
         // find the derived type of this book
         // by looking at the value of the xsi:type attribute, using DOM
         org.w3c.dom.Node bookNode = book.getNode();
         if (bookNode.getAttributes().getNamedItem("xsi:type") != null) {
            // Get the value of the xsi:type attribute
            String xsiTypeValue =
bookNode.getAttributes().getNamedItem("xsi:type").getNodeValue();
```

```
            // Get the namespace URI and lookup prefix of the book node
            String namespaceUri = bookNode.getNamespaceURI();
            String lookupPrefix = bookNode.lookupPrefix(namespaceUri);

            // If xsi:type matches the namespace URI and type of the book node
            if (namespaceUri == "http://www.nanonull.com/LibrarySample"
                  && (  xsiTypeValue.equals(lookupPrefix + ":DictionaryType" )))    {
               // ...then this is a book of derived type (dictionary)
               DictionaryType dictionary = new DictionaryType(   book.getNode());
               // output the value of the "FromLang" and "ToLang" elements
               System.out.println("From language: " +
dictionary.FromLang.first().getValue());
               System.out.println("To language: " +
dictionary.ToLang.first().getValue());
            }
            else
            {
               // throw an error
               throw new java.lang.Error("This book has an unknown type.");
            }
         }
```

16.6.6 Example: Purchase Order

This example illustrates how to work with program code generated from a "main" XML schema that imports other schemas. Each of the imported schema has a different target namespace. The goal here is to create programmatically an XML document where all elements are prefixed according to their namespace. More specifically, the XML document created from your C++, C#, or Java code should look like the one below:

```
<?xml version="1.0" encoding="utf-8"?>
<p:Purchase xsi:schemaLocation="http://NamespaceTest.com/Purchase Main.xsd"
            xmlns:p="http://NamespaceTest.com/Purchase"
            xmlns:o="http://NamespaceTest.com/OrderTypes"
            xmlns:c="http://NamespaceTest.com/CustomerTypes"
            xmlns:cmn="http://NamespaceTest.com/CommonTypes"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <p:OrderDetail>
        <o:Item>
            <o:ProductName>Lawnmower</o:ProductName>
            <o:Quantity>1</o:Quantity>
            <o:UnitPrice>148.42</o:UnitPrice>
        </o:Item>
    </p:OrderDetail>
    <p:PaymentMethod>VISA</p:PaymentMethod>
    <p:CustomerDetails>
        <c:Name>Alice Smith</c:Name>
        <c:DeliveryAddress>
            <cmn:Line1>123 Maple Street</cmn:Line1>
            <cmn:Line2>Mill Valley</cmn:Line2>
```

```
        </c:DeliveryAddress>
        <c:BillingAddress>
            <cmn:Line1>8 Oak Avenue</cmn:Line1>
            <cmn:Line2>Old Town</cmn:Line2>
        </c:BillingAddress>
    </p:CustomerDetails>
</p:Purchase>
```

The main schema used in this example is called **Main.xsd**. As illustrated in the code listing below, it imports three other schemas: **CommonTypes.xsd**, **CustomerTypes.xsd**, and **OrderTypes.xsd**. To get the same results as in this example, save all the code listings below to files, and use the same file names as above. Notice that the schema maps each of the prefixes `ord`, `pur`, `cmn`, and `cust` to some namespace (Order types, Purchase types, Common types, and Customer types, respectively). This means that, in the generated code, the classes corresponding to Orders, Purchases, Customers, and so on, will be available under their respective namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://NamespaceTest.com/Purchase"
      xmlns:ord="http://NamespaceTest.com/OrderTypes"
      xmlns:pur="http://NamespaceTest.com/Purchase"
      xmlns:cmn="http://NamespaceTest.com/CommonTypes"
      xmlns:cust="http://NamespaceTest.com/CustomerTypes"
      elementFormDefault="qualified">
    <xs:import schemaLocation="CommonTypes.xsd"
namespace="http://NamespaceTest.com/CommonTypes" />
    <xs:import schemaLocation="CustomerTypes.xsd"
namespace="http://NamespaceTest.com/CustomerTypes" />
    <xs:import schemaLocation="OrderTypes.xsd"
namespace="http://NamespaceTest.com/OrderTypes" />
    <xs:element name="Purchase">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="OrderDetail" type="ord:OrderType" />
                <xs:element name="PaymentMethod" type="cmn:PaymentMethodType" />
                <xs:element ref="pur:CustomerDetails" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="CustomerDetails" type="cust:CustomerType" />
</xs:schema>
```

Main.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://NamespaceTest.com/CommonTypes"
      elementFormDefault="qualified">
   <xs:complexType name="AddressType">
      <xs:sequence>
         <xs:element name="Line1" type="xs:string"/>
```

```
         <xs:element name="Line2" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:simpleType name="PriceType">
      <xs:restriction base="xs:decimal">
         <xs:fractionDigits value="2"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="PaymentMethodType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="VISA"/>
         <xs:enumeration value="MasterCard"/>
         <xs:enumeration value="Cash"/>
         <xs:enumeration value="AMEX"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

CommonTypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://NamespaceTest.com/CustomerTypes"
      xmlns:cmn="http://NamespaceTest.com/CommonTypes"
      elementFormDefault="qualified">
    <xs:import schemaLocation="CommonTypes.xsd"
namespace="http://NamespaceTest.com/CommonTypes" />
    <xs:complexType name="CustomerType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" />
            <xs:element name="DeliveryAddress" type="cmn:AddressType" />
            <xs:element name="BillingAddress" type="cmn:AddressType" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

CustomerTypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://NamespaceTest.com/OrderTypes"
      xmlns:cmn="http://NamespaceTest.com/CommonTypes"
      elementFormDefault="qualified">
    <xs:import schemaLocation="CommonTypes.xsd"
namespace="http://NamespaceTest.com/CommonTypes" />
    <xs:complexType name="OrderType">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" name="Item">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="ProductName" type="xs:string" />
```

```
                        <xs:element name="Quantity" type="xs:int" />
                        <xs:element name="UnitPrice" type="cmn:PriceType" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

OrderTypes.xsd

To complete this example, take the following steps:

1. Save all schemas from the code listings above to files on the disk, making sure that you preserve the indicated file names.
2. Generate the schema wrapper code from the **Main.xsd** schema above, using the steps described in Generating Code from XML Schemas or DTD1535. After completing this step, you should have generated a compilable program in the language of your choice (C++, C#, or Java).
3. Add code to your C++, C#, or Java program from one the following example code listings, as required:

   - XML Namespaces and Prefixes (C++)1574
   - XML Namespaces and Prefixes (C#)1575
   - XML Namespaces and Prefixes (Java)1577

16.6.6.1 XML Namespaces and Prefixes (C++)

After you generate code from the example schema1571, a test C++ application is created, along with several supporting Altova libraries. Recall that the example schema (**Main.xsd**) has multiple namespace declarations. Consequently, the generated code includes namespaces that correspond to namespace aliases (prefixes) from the schema, namely: `Main::ord`, `Main::pur`, `Main::cmn`, and `Main::cust`.

In general, in order to control XML namespaces and prefixes with the help of the schema wrapper libraries, you have the following methods at your disposal:

- DeclareAllNamespacesFromSchema()1589. Call this method if you want to declare the same namespaces in your XML instance as in the schema. Otherwise, if you need different namespaces as in this example, then **`DeclareNamespace()`** should be used. The method **`DeclareAllNamespacesFromSchema()`** is not used in this example because we specifically want to create XML elements with prefixes that are slightly different from those declared in the schema.
- DeclareNamespace()1591. Call this method to create or override the existing namespace prefix attribute on an element. The element must already be created using either the **`append()`** or **`appendWithPrefix()`** methods, as further illustrated below.
- appendWithPrefix().1593 Use this method to append an instance element with a specific prefix. To create the XML instance illustrated in this example, it was sufficient to call this method for the root element only. All other elements were appended using just append()1593, and their prefixes were added automatically based on their namespaces, according to the rules above.

The code listing below shows you how to create an XML document with multiple namespace declarations and prefixed element names. Specifically, it generates a Purchase Order instance as illustrated in the Example: Purchase Order1571. Importantly, for illustrative purposes, some prefixes are overridden in the XML instance (that is, they are not exactly the same as the ones declared in the schema).


```
void Example()
{
   // Create the XML document and append the root element
   Main::pur::CMain doc = Main::pur::CMain::CreateDocument();
   Main::pur::CPurchaseType purchase = doc.Purchase.appendWithPrefix(_T("p"));

   // Set schema location
   doc.SetSchemaLocation(_T("Main.xsd"));

   // Declare namespaces on root element
   purchase.DeclareNamespace(_T("o"), _T("http://NamespaceTest.com/OrderTypes"));
   purchase.DeclareNamespace(_T("c"), _T("http://NamespaceTest.com/CustomerTypes"));
   purchase.DeclareNamespace(_T("cmn"), _T("http://NamespaceTest.com/CommonTypes"));

   // Append the OrderDetail element
   Main::ord::COrderType order = purchase.OrderDetail.append();
   Main::ord::CItemType item = order.Item.append();
   item.ProductName.append() = _T("Lawnmower");
   item.Quantity.append() = 1;
   item.UnitPrice.append() = 148.42;

   // Append the PaymentMethod element
   Main::cmn::CPaymentMethodTypeType paymentMethod = purchase.PaymentMethod.append();
   paymentMethod.SetEnumerationValue(Main::cmn::CPaymentMethodTypeType::k_VISA);

   // Append the CustomerDetails element
   Main::cust::CCustomerType customer = purchase.CustomerDetails.append();
   customer.Name.append() = _T("Alice Smith");
   Main::cmn::CAddressType deliveryAddress = customer.DeliveryAddress.append();
   deliveryAddress.Line1.append() = _T("123 Maple Street");
   deliveryAddress.Line2.append() = _T("Mill Valley");
   Main::cmn::CAddressType billingAddress = customer.BillingAddress.append();
   billingAddress.Line1.append() = _T("8 Oak Avenue");
   billingAddress.Line2.append() = _T("Old Town");

   // Save to file and release object from memory
   doc.SaveToFile(_T("Main1.xml"), true);
   doc.DestroyDocument();
}
```

16.6.6.2 XML Namespaces and Prefixes (C#)

After you generate code from the example schema 1571, a test C# application is created, along with several supporting Altova libraries. Recall that the example schema (**Main.xsd**) has multiple namespace declarations. Consequently, the generated code includes namespaces that correspond to namespace aliases (prefixes) from the schema, namely: `Main.ord`, `Main.pur`, `Main.cmn`, and `Main.cust`.

In general, in order to control XML namespaces and prefixes with the help of the schema wrapper libraries, you have the following methods at your disposal:

- DeclareAllNamespacesFromSchema()[1604]. Call this method if you want to declare the same namespaces in your XML instance as in the schema. Otherwise, if you need different namespaces as in this example, then **DeclareNamespace()** should be used. The method **DeclareAllNamespacesFromSchema()** is not used in this example because we specifically want to create XML elements with prefixes that are slightly different from those declared in the schema.
- DeclareNamespace()[1606]. Call this method to create or override the existing namespace prefix attribute on an element. The element must already be created using either the **Append()** or **AppendWithPrefix()** methods, as further illustrated below.
- AppendWithPrefix().[1607] Use this method to append an instance element with a specific prefix. To create the XML instance illustrated in this example, it was sufficient to call this method for the root element only. All other elements were appended using just Append()[1607], and their prefixes were added automatically based on their namespaces, according to the rules above.

The code listing below shows you how to create an XML document with multiple namespace declarations and prefixed element names. Specifically, it generates a Purchase Order instance as illustrated in the Example: Purchase Order[1571]. Importantly, for illustrative purposes, some prefixes are overridden in the XML instance (that is, they are not exactly the same as the ones declared in the schema).

```
protected static void Example()
{
   // Create the XML document and append the root element
   pur.Main2 doc = pur.Main2.CreateDocument();
   pur.PurchaseType purchase = doc.Purchase.AppendWithPrefix("p");

   // Set schema location
   doc.SetSchemaLocation(@"Main.xsd");

   // Declare namespaces on root element
   purchase.DeclareNamespace("o", "http://NamespaceTest.com/OrderTypes");
   purchase.DeclareNamespace("c", "http://NamespaceTest.com/CustomerTypes");
   purchase.DeclareNamespace("cmn", "http://NamespaceTest.com/CommonTypes");

   // Append the OrderDetail element
   ord.OrderType order = purchase.OrderDetail.Append();
   ord.ItemType item = order.Item.Append();
   item.ProductName.Append().Value = "Lawnmower";
   item.Quantity.Append().Value = 1;
   item.UnitPrice.Append().Value = 148.42M;

   // Append the PaymentMethod element
   cmn.PaymentMethodTypeType paymentMethod = purchase.PaymentMethod.Append();
   paymentMethod.EnumerationValue = cmn.PaymentMethodTypeType.EnumValues.eVISA;

   // Append the CustomerDetails element
   cust.CustomerType customer = purchase.CustomerDetails.Append();
   customer.Name.Append().Value = "Alice Smith";
   cmn.AddressType deliveryAddress = customer.DeliveryAddress.Append();
   deliveryAddress.Line1.Append().Value = "123 Maple Street";
   deliveryAddress.Line2.Append().Value = "Mill Valley";
   cmn.AddressType billingAddress = customer.BillingAddress.Append();
   billingAddress.Line1.Append().Value = "8 Oak Avenue";
   billingAddress.Line2.Append().Value = "Old Town";

   // Save to file
```



```
   doc.SaveToFile("PurchaseOrder.xml", true);
}
```

16.6.6.3 XML Namespaces and Prefixes (Java)

After you generate code from the example schema[1571], a test Java application is created, along with several supporting Altova libraries. Recall that the example schema (**Main.xsd**) has multiple namespace declarations. Consequently, the generated code includes namespaces that correspond to namespace aliases (prefixes) from the schema, namely: **com.Main.ord**, **com.Main.pur**, **com.Main.cmn**, and **com.Main.cust**.

In general, in order to control XML namespaces and prefixes with the help of the schema wrapper libraries, you have the following methods at your disposal:

- declareAllNamespacesFromSchema()[1619]. Call this method if you want to declare the same namespaces in your XML instance as in the schema. Otherwise, if you need different namespaces as in this example, then **declareNamespace()** should be used. The method **declareAllNamespacesFromSchema()** is not used in this example because we specifically want to create XML elements with prefixes that are slightly different from those declared in the schema.
- declareNamespace()[1621]. Call this method to create or override the existing namespace prefix attribute on an element. The element must already be created using either the **append()** or **appendWithPrefix()** methods, as further illustrated below.
- appendWithPrefix().[1622] Use this method to append an instance element with a specific prefix. To create the XML instance illustrated in this example, it was sufficient to call this method for the root element only. All other elements were appended using just append()[1622], and their prefixes were added automatically based on their namespaces, according to the rules above.

The code listing below shows you how to create an XML document with multiple namespace declarations and prefixed element names. Specifically, it generates a Purchase Order instance as illustrated in the Example: Purchase Order[1571]. Importantly, for illustrative purposes, some prefixes are overridden in the XML instance (that is, they are not exactly the same as the ones declared in the schema).

```
protected static void example() throws Exception {
   // Create the XML document and append the root element
   com.Main.pur.Main2 doc = com.Main.pur.Main2.createDocument();
   com.Main.pur.PurchaseType purchase = doc.Purchase.appendWithPrefix("p");

   // Set schema location
   doc.setSchemaLocation("Main.xsd");

   // Declare namespaces on root element
   purchase.declareNamespace("o", "http://NamespaceTest.com/OrderTypes");
   purchase.declareNamespace("c", "http://NamespaceTest.com/CustomerTypes");
   purchase.declareNamespace("cmn", "http://NamespaceTest.com/CommonTypes");

   // Append the OrderDetail element
   com.Main.ord.OrderType order = purchase.OrderDetail.append();
   com.Main.ord.ItemType item = order.Item.append();
   item.ProductName.append().setValue("Lawnmower");
   item.Quantity.append().setValue(1);
   java.math.BigDecimal price = new java.math.BigDecimal("148.42");
```

```
   item.UnitPrice.append().setValue(price);

   // Append the PaymentMethod element
   com.Main.cmn.PaymentMethodTypeType paymentMethod = purchase.PaymentMethod.append();
   paymentMethod.setEnumerationValue(com.Main.cmn.PaymentMethodTypeType.EVISA);

   // Append the CustomerDetails element
   com.Main.cust.CustomerType customer = purchase.CustomerDetails.append();
   customer.Name.append().setValue("Alice Smith");
   com.Main.cmn.AddressType deliveryAddress = customer.DeliveryAddress.append();
   deliveryAddress.Line1.append().setValue("123 Maple Street");
   deliveryAddress.Line2.append().setValue("Mill Valley");
   com.Main.cmn.AddressType billingAddress = customer.BillingAddress.append();
   billingAddress.Line1.append().setValue("8 Oak Avenue");
   billingAddress.Line2.append().setValue("Old Town");

   // Save to file
   doc.saveToFile("PurchaseOrder.xml", true);
}
```

16.7 Reference to Generated Classes (C++)

This chapter includes a description of C++ classes generated with MapForce from a DTD or XML schema (see Generating Code from XML Schemas or DTDs 1535). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code does include other supporting classes, which are not listed here and are subject to modification.

16.7.1 altova::DateTime

This class enables you to process XML attributes or elements that have date and time types, such as `xs:dateTime`.

Constructors

Name	Description
`DateTime()`	Initializes a new instance of the `DateTime` class to 12:00:00 midnight, January 1, 0001.
`DateTime(__int64 value, short timezone)`	Initializes a new instance of the `DateTime` class. The `value` parameter represents the number of ticks (100-nanosecond intervals) that have elapsed since 12:00:00 midnight, January 1, 0001.
`DateTime(int year, unsigned char month, unsigned char day, unsigned char hour, unsigned char minute, double second)`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, and second supplied as argument.
`DateTime(int year, unsigned char month, unsigned char day, unsigned char hour, unsigned char minute, double second, short timezone)`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, second and timezone supplied as argument. The timezone is expressed in minutes and can be positive or negative. For example, the timezone "UTC-01:00" is expressed as "-60".

Methods

Name	Description
`unsigned char Day() const`	Returns the day of month of the current `DateTime` object. The return values range from 1 through 31.
`int DayOfYear() const`	Returns the day of year of the current `DateTime` object. The return values range from 1 through 366.
`bool HasTimezone() const`	Returns Boolean **true** if the current `DateTime` object has a timezone defined; **false** otherwise.

Name	Description
`unsigned char Hour() const`	Returns the hour of the current `DateTime` object. The return values range from 0 through 23.
`static bool IsLeapYear(int year)`	Returns Boolean **true** if the year of the `DateTime` class is a leap year; **false** otherwise.
`unsigned char Minute() const`	Returns the minute of the current `DateTime` object. The return values range from 0 through 59.
`unsigned char Month() const`	Returns the month of the current `DateTime` object. The return values range from 1 through 12.
`__int64 NormalizedValue() const`	Returns the value of the `DateTime` object expressed as the Coordinated Universal Time (UTC).
`double Second() const`	Returns the second of the current `DateTime` object. The return values range from 0 through 59.
`void SetTimezone(short tz)`	Sets the timezone of the current `DateTime` object to the timezone value supplied as argument. The **tz** argument is expressed in minutes and can be positive or negative.
`short Timezone() const`	Returns the timezone, in minutes, of the current `DateTime` object. Before using this method, make sure that the object actually has a timezone, by calling the `HasTimezone()` method.
`__int64 Value() const`	Returns the value of the `DateTime` object, expressed in the number of ticks (100-nanosecond intervals) that have elapsed since 12:00:00 midnight, January 1, 0001.
`int Weekday() const`	Returns the day of week of the current `DateTime` object, as an integer. Values range from 0 through 6, where 0 is Monday (ISO-8601).
`int Weeknumber() const`	Returns the number of week in the year of the current `DateTime` object. The return values are according to ISO-8601.
`int WeekOfMonth() const`	Returns the number of week in the month of the current `DateTime` object. The return values are according to ISO-8601.
`int Year() const`	Returns the year of the current `DateTime` object.

Example

```
void Example()
{
   // initialize a new DateTime instance to 12:00:00 midnight, January 1st, 0001
   altova::DateTime dt1 = altova::DateTime();

   // initialize a new DateTime instance using the year, month, day, hour, minute, and
second
   altova::DateTime dt2 = altova::DateTime(2015, 11, 10, 9, 8, 7);
```

```
   // initialize a new DateTime instance using the year, month, day, hour, minute,
second, and UTC +01:00 timezone
    altova::DateTime dt = altova::DateTime(2015, 11, 22, 13, 53, 7, 60);

   // Get the value of this DateTime object
   std::cout << "The number of ticks of the DateTime object is: " << dt.Value() <<
std::endl;

   // Get the year
   cout << "The year is: " << dt.Year() << endl;
   // Get the month
   cout << "The month is: " << (int)dt.Month() << endl;
   // Get the day of the month
   cout << "The day of the month is: " << (int) dt.Day() << endl;
   // Get the day of the year
   cout << "The day of the year is: " << dt.DayOfYear() << endl;
   // Get the hour
   cout << "The hour is: " << (int) dt.Hour() << endl;
   // Get the minute
   cout << "The minute is: " << (int) dt.Minute() << endl;
   // Get the second
   cout << "The second is: " << dt.Second() << endl;
   // Get the weekday
   cout << "The weekday is: " << dt.Weekday() << endl;
   // Get the week number
   cout << "The week of year is: " << dt.Weeknumber() << endl;
   // Get the week in month
   cout << "The week of month is: " << dt.WeekOfMonth() << endl;

   // Check whether a DateTime instance has a timezone
   if (dt.HasTimezone() == TRUE)
   {
      // output the value of the Timezone
      cout <<  "The timezone is: " << dt.Timezone() << endl;
   }
   else
   {
      cout <<  "No timezone has been defined." << endl;
   }

   // Construct a DateTime object with a timezone UTC+01:00 (Vienna)
   altova::DateTime vienna_dt = DateTime(2015, 11, 23, 14, 30, 59, +60);
   // Output the result in readable format
   cout << "The Vienna time: "
       << (int) vienna_dt.Month()
       << "-" << (int) vienna_dt.Day()
       << " " << (int) vienna_dt.Hour()
       << ":" << (int) vienna_dt.Minute()
       << ":" << (int) vienna_dt.Second()
       << endl;

   // Convert the value to UTC time
```

```
   DateTime utc_dt = DateTime(vienna_dt.NormalizedValue());
   // Output the result in readable format
   cout << "The UTC time:    "
      << (int) utc_dt.Month()
      << "-" << (int) utc_dt.Day()
      << " " << (int) utc_dt.Hour()
      << ":" << (int) utc_dt.Minute()
      << ":" << (int) utc_dt.Second()
      << endl;

   // Check if a year is a leap year
   int year = 2016;
   if( altova::DateTime::IsLeapYear(year) )
   { cout << year << " is a leap year" << endl; }
   else
   { cout << year << " is not a leap year" << endl; }
}
```

16.7.2 altova::Duration

This class enables you to process XML attributes or elements of type `xs:duration`.

Constructors

Name	Description
`Duration()`	Initializes a new instance of the `Duration` class to an empty value.
`Duration(const DayTimeDuration& dt)`	Initializes a new instance of the `Duration` class to a duration defined by the `dt` argument (see altova::DayTimeDuration 1584).
`Duration(const YearMonthDuration& ym)`	Initializes a new instance of the `Duration` class to the duration defined by the `ym` argument (see altova::YearMonthDuration 1585).
`Duration(const YearMonthDuration& ym, const DayTimeDuration& dt)`	Initializes a new instance of the `Duration` class to the duration defined by both the `dt` and the `ym` arguments (see altova::YearMonthDuration 1585 and altova::DayTimeDuration 1584).

Methods

Name	Description
`int Days() const`	Returns the number of days in the current `Duration` instance.
`DayTimeDuration DayTime() const`	Returns the day and time duration in the current `Duration` instance, expressed as a `DayTimeDuration` object (see altova::DayTimeDuration 1584).
`int Hours() const`	Returns the number of hours in the current `Duration` instance.
`bool IsNegative() const`	Returns Boolean **true** if the current `Duration` instance is negative.

Name	Description
`bool IsPositive() const`	Returns Boolean **true** if the current `Duration` instance is positive.
`int Minutes() const`	Returns the number of minutes in the current `Duration` instance.
`int Months() const`	Returns the number of months in the current `Duration` instance.
`double Seconds() const`	Returns the number of seconds in the current `Duration` instance.
`YearMonthDuration YearMonth() const`	Returns the year and month duration in the current `Duration` instance, expressed as a `YearMonthDuration` object (see altova::YearMonthDuration 1585).
`int Years() const`	Returns the number of years in the current `Duration` instance.

Example

The following code listing illustrates creating a new `Duration` object, as well as reading values from it.

```
void ExampleDuration()
{
   // Create an empty Duration object
   altova::Duration empty_duration = altova::Duration();

   // Create a Duration object using an existing duration value
   altova::Duration duration1 = altova::Duration(empty_duration);

   // Create a YearMonth duration of six years and five months
   altova::YearMonthDuration yrduration = altova::YearMonthDuration(6, 5);

   // Create a DayTime duration of four days, three hours, two minutes, and one second
   altova::DayTimeDuration dtduration = altova::DayTimeDuration(4, 3, 2, 1);

   // Create a Duration object by combining the two previously created durations
   altova::Duration duration = altova::Duration(yrduration, dtduration);

   // Get the number of years in this Duration instance
   cout << "Years:  " << duration.Years() << endl;

   // Get the number of months in this Duration instance
   cout << "Months: " << duration.Months() << endl;

   // Get the number of days in this Duration instance
   cout << "Days:   " << duration.Days() << endl;

   // Get the number of hours in this Duration instance
   cout << "Hours:  " << duration.Hours() << endl;

   // Get the number of hours in this Duration instance
   cout << "Minutes: " << duration.Minutes() << endl;

   // Get the number of seconds in this Duration instance
```

```
   cout << "Seconds: " << duration.Seconds() << endl;
}
```

16.7.3 altova::DayTimeDuration

This class enables you to process XML schema duration types that consist of a day and time part.

Constructors

Name	Description
`DayTimeDuration()`	Initializes a new instance of the `DayTimeDuration` class to an empty value.
`DayTimeDuration(int days, int hours, int minutes, double seconds)`	Initializes a new instance of the `DayTimeDuration` class to the number of days, hours, minutes, and seconds supplied as arguments.
`explicit DayTimeDuration(__int64 value)`	Initializes a new instance of the `DayTimeDuration` class to a duration that consists of as many ticks (100-nanosecond intervals) as supplied in the **value** argument.

Methods

Name	Description
`int Days() const`	Returns the number of days in the current `DayTimeDuration` instance.
`int Hours() const`	Returns the number of hours in the current `DayTimeDuration` instance.
`bool IsNegative() const`	Returns Boolean **true** if the current `DayTimeDuration` instance is negative.
`bool IsPositive() const`	Returns Boolean **true** if the current `DayTimeDuration` instance is positive.
`int Minutes() const`	Returns the number of minutes in the current `DayTimeDuration` instance.
`double Seconds() const`	Returns the number of seconds in the current `DayTimeDuration` instance.
`__int64 Value() const`	Returns the value (in ticks) of the current `DayTimeDuration` instance.

16.7.4 altova::YearMonthDuration

This class enables you to process XML schema duration types that consist of a year and month part.

Constructors

Name	Description
`YearMonthDuration()`	Initializes a new instance of the `YearMonthDuration` class to an empty value.
`YearMonthDuration(int years, int months)`	Initializes a new instance of the `YearMonthDuration` class to the number of years and months supplied in the **years** and **months** arguments.
`explicit YearMonthDuration(int value)`	Initializes a new instance of the `YearMonthDuration` class to a duration that consists of as many ticks (100-nanosecond intervals) as supplied in the **value** argument.

Methods

Name	Description
`bool IsNegative() const`	Returns Boolean **true** if the current `YearMonthDuration` instance is negative.
`bool IsPositive() const`	Returns Boolean **true** if the current `YearMonthDuration` instance is positive.
`int Months() const`	Returns the number of months in the current `YearMonthDuration` instance.
`int Value() const`	Returns the value (in ticks) of the current `YearMonthDuration` instance.
`int Years()`	Returns the number of years in the current `YearMonthDuration` instance.

16.7.5 altova::meta::Attribute

This class enables you to access schema information about classes generated from attributes. Note that this class is not meant to provide dynamic information about particular instances of an attribute in an XML document. Instead, it enables you to obtain programmatically information about a particular attribute defined in the XML schema.

Methods

Name	Description
`SimpleType GetDataType()`	Returns the type of the attribute content.
`string_type GetLocalName()`	Returns the local name of the attribute.
`string_type GetNamespaceURI()`	Returns the namespace URI of the attribute.
`bool IsRequired()`	Returns true if the attribute is required.

Operators

Name	Description
`bool operator()`	Returns true if this is not the NULL Attribute.
`bool operator!()`	Returns true if this is the NULL Attribute.

16.7.6 altova::meta::ComplexType

This class enables you to access schema information about classes generated from complex types. Note that this class is not meant to provide dynamic information about particular instances of a complex type in an XML document. Instead, it enables you to obtain programmatically information about a particular complex type defined in the XML schema.

Methods

Name	Description
`Attribute FindAttribute(const char_type* localName, const char_type* namespaceURI)`	Finds the attribute with the specified local name and namespace URI.
`Element FindElement(const char_type* localName, const char_type* namespaceURI)`	Finds the element with the specified local name and namespace URI.
`std::vector<Attribute> GetAttributes()`	Returns a list of all attributes.
`ComplexType GetBaseType()`	Returns the base type of this type.
`SimpleType GetContentType()`	Returns the simple type of the content.
`std::vector<Element> GetElements()`	Returns a list of all elements.
`string_type GetLocalName()`	Returns the local name of the type.

Name	Description
`string_type GetNamespaceURI()`	Returns the namespace URI of the type.

Operators

Name	Description
`bool operator()`	Returns true if this is not the NULL ComplexType.
`bool operator!()`	Returns true if this is the NULL ComplexType.

16.7.7 altova::meta::Element

This class enables you to access information about classes generated from schema elements. Note that this class is not meant to provide dynamic information about particular instances of an element in an XML document. Instead, it enables you to obtain programmatically information about a particular element defined in the XML schema.

Methods

Name	Description
`ComplexType GetDataType()`	Returns the type of the element. Note that this is always a complex type even if declared as simple in the original schema. Use GetContentType() of the returned object to get the simple content type.
`string_type GetLocalName()`	Returns the local name of the element.
`unsigned int GetMaxOccurs()`	Returns the maxOccurs value defined in the schema.
`unsigned int GetMinOccurs()`	Returns the minOccurs value defined in the schema.
`string_type GetNamespaceURI()`	Returns the namespace URI of the element.

Operators

Name	Description
`bool operator()`	Returns true if this is not the NULL Element.
`bool operator!()`	Returns true if this is the NULL Element.

16.7.8 altova::meta::SimpleType

This class enables you to access schema information about classes generated from simple types. Note that this class is not meant to provide dynamic information about particular instances of simple types in an XML document. Instead, it enables you to obtain programmatically information about a particular simple type defined in the XML schema.

Methods

Name	Description
`SimpleType GetBaseType()`	Returns the base type of this type.
`std::vector<string_type> GetEnumerations()`	Returns a list of all enumeration facets.
`unsigned int GetFractionDigits()`	Returns the value of this facet.
`unsigned int GetLength()`	Returns the value of this facet.
`string_type GetLocalName()`	Returns the local name of the type.
`string_type GetMaxExclusive()`	Returns the value of this facet.
`string_type GetMaxInclusive()`	Returns the value of this facet.
`unsigned int GetMaxLength()`	Returns the value of this facet.
`string_type GetMinExclusive()`	Returns the value of this facet.
`string_type GetMinInclusive()`	Returns the value of this facet.
`unsigned int GetMinLength()`	Returns the value of this facet.
`string_type GetNamespaceURI()`	Returns the namespace URI of the type.
`std::vector<string_type> GetPatterns()`	Returns a list of all pattern facets.
`unsigned int GetTotalDigits()`	Returns the value of this facet.
`WhitespaceType GetWhitespace()`	Returns the value of the whitespace facet, which is one of: • Whitespace_Unknown • Whitespace_Preserve • Whitespace_Replace • Whitespace_Collapse

Operators

Name	Description
`bool operator()`	Returns true if this is not the NULL SimpleType.
`bool operator!()`	Returns true if this is the NULL SimpleType.

16.7.9 [YourSchema]::[CDoc]

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the following methods. Note that, in the method names below, "CDoc" stands for the name of the generated document class itself.

Methods

Name	Description
`static CDoc CreateDocument()`	Creates a new, empty XML document. Must be released using DestroyDocument().
`static void DeclareAllNamespacesFromSchema(ElementType& node)`	Declares all namespaces from the XML Schema on the element supplied as argument (typically, the XML root element). Calling this method is useful if your schema has multiple namespace declarations, each mapped to a prefix, and you would like to declare all of them on the element supplied as argument.
`void DestroyDocument()`	Destroys a document. All references to the document and its nodes are invalidated. This must be called when you finish working with a document.
`static CDoc LoadFromBinary(const std:vector<unsigned char>& xml)`	Loads an XML document from a byte array.
`static CDoc LoadFromFile(const string_type& fileName)`	Loads an XML document from a file.
`static CDoc LoadFromString(const string_type& xml)`	Loads an XML document from a string.
`std::vector<unsigned char> SaveToBinary(bool prettyPrint)`	Saves an XML document to a byte array. When set to true, the `prettyPrint` argument re-formats the XML document for better readability.
`std::vector<unsigned char> SaveToBinary(bool prettyPrint, const string_type & encoding)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding.
`std::vector<unsigned char> SaveToBinary(bool prettyPrint, const string_type & encoding, bool bBigEndian, bool bBOM)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
`void SaveToFile(const string_type & fileName, bool prettyPrint)`	Saves an XML document to a file, with optional "pretty-print" formatting.
`void SaveToFile(const string_type & fileName, bool omitXmlDecl)`	Saves an XML document to a file. If the `omitXmlDecl` argument is set to true, the XML declaration will not be written.

Name	Description
`void SaveToFile(const string_type & fileName, bool omitXmlDecl, const string_type & encoding)`	Saves an XML document to a file with the specified encoding. If the `omitXmlDecl` argument is set to true, the XML declaration will not be written.
`void SaveToFile(const string_type & fileName, bool prettyPrint, bool omitXmlDecl, const string_type & encoding, bool bBigEndian, bool bBOM)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
`void SaveToFile(const string_type & fileName, bool prettyPrint, bool omitXmlDecl, const string_type & encoding, bool bBigEndian, bool bBOM, const string_type & lineend)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding and the specified line end. Byte order and Unicode byte-order mark can be specified for Unicode encodings. This method is only available if you generated the code for the Xerces3 XML library (see Code Generator Options1625).
`void SaveToFile(const string_type& fileName, bool prettyPrint, bool omitXmlDecl, const string_type & encoding, const string_type & lineend)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding and the specified line end. This method is only available if you generated the code for the Xerces3 XML library (see Code Generator Options1625).
`void SaveToFile(const string_type & fileName, bool prettyPrint, const string_type & encoding)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding.
`void SaveToFile(const string_type& fileName, bool prettyPrint, const string_type & encoding, bool bBigEndian, bool bBOM)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
`void SaveToFile(const string_type& fileName, bool prettyPrint, const string_type & encoding, bool bBigEndian, bool bBOM, const string_type & lineend)`	Saves an XML document to a file with the specified encoding and the specified line end. Byte order and Unicode byte-order mark can be specified for Unicode encodings. This method is only available if you generated the code for the Xerces3 XML library (see Code Generator Options1625).
`void SaveToFile(const string_type& fileName, bool prettyPrint, const string_type & encoding, const string_type & lineend)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding and the specified line end. This method is only available if you generated the code for the Xerces3 XML library (see Code Generator Options1625).
`string_type SaveToString(bool prettyPrint)`	Saves an XML document to a string, with optional "pretty-print" formatting.

Name	Description
`string_type SaveToString(bool prettyPrint, bool omitXmlDecl)`	Saves an XML document to a string, with optional "pretty-print" formatting. If the `omitXmlDecl` argument is set to true, the XML declaration will not be written.
`void SetDTDLocation(const string_type & dtdLocation)`	Adds a DOCTYPE declaration with the specified system ID. A root element must already exist. This method is not supported for MSXML, since it is not possible to add a DOCTYPE declaration to a document in memory.
`void SetSchemaLocation(const string_type & schemaLocation)`	Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.

16.7.10 [YourSchema]::[ElementType]

This class provides methods for manipulating XML elements from your schema. Methods of this class can be called on elements, not on the XML document itself. Note that, in order to call methods of this class, you don't need to instantiate the class directly. Any element created using the **`append()`** or **`appendWithPrefix()`** methods is of `[ElementType]` type.

Methods

Name	Description

`void DeclareNamespace(const string_type prefix, const string_type nsURI)`	This method takes two arguments that are both of string type: the prefix and the namespace URI that you want to use. The prefix supplied as argument will be mapped to the namespace URI value supplied as argument. If the prefix supplied as argument is empty, the method creates or overrides the default namespace declaration in the element.
	
	For example, let's assume that the XML document has an XML element called "purchase". If you call
	
	`purchase.DeclareNamespace(_T("ord"), _T("http://OrderTypes"));`
	
	then the XML document becomes
	
	`<purchase xmlns:ord="http://OrderTypes" />`
	
	Another example, if you call:
	
	`purchase.DeclareNamespace(_T(""), _T("http://OrderTypes"));`

Name	Description
	then the XML document becomes `<purchase xmlns="http://OrderTypes" />` **Note:** The declared namespace is used when appending subsequent child elements or attributes, according to the following rules: 1. If the child namespace is the default, then use empty prefix. 2. If the child namespace is equal to the parent one, then use the parent prefix. 3. Otherwise, search for nearest prefix from parent to top, using the lookup algorithm described in section "B.2: Namespace Prefix Lookup" at https://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20021022/namespaces-algorithms.html. 4. If there is no prefix for element namespace found, then use empty prefix.

16.7.11 [YourSchema]::MemberAttribute

When code is generated from an XML schema, a class such as this one is created for each member attribute of a type.

Methods

Name	Description
`bool exists()`	Returns true if the attribute exists.
`int GetEnumerationValue()`	Generated for enumeration types only. Returns one of the constants generated for the possible values, or "Invalid" if the value does not match any of the enumerated values in the schema.
`altova::meta::Attribute info()`	Returns an object for querying schema information (see `altova::meta::Attribute` 1585).
`void remove()`	Removes the attribute from its parent element.
`void SetEnumerationValue(int)`	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.

16.7.12 [YourSchema]::MemberElement

When code is generated from an XML schema, a class such as this one is created for each member element of a type. In the descriptions below, "MemberType" stands for the name of the member element itself.

Methods

Name	Description
`Iterator<MemberType> all()`	Returns an object for iterating instances of the member element.
`MemberType append()`	Creates a new element and appends it to its parent.
`MemberType appendWithPrefix(string_type prefix)`	Creates a new element having the prefix supplied as argument, and appends it to its parent. For an example, see Example: Purchase Order 1571.
`unsigned int count()`	Returns the count of elements.
`int GetEnumerationValue()`	Generated for enumeration types only. Returns one of the constants generated for the possible values, or **Invalid** if the value does not match any of the enumerated values in the schema.
`bool exists()`	Returns true if at least one element exists.
`MemberType first()`	Returns the first instance of the member element.
`MemberType operator[](unsigned int index)`	Returns the member element specified by the index.
`altova::meta::Element info()`	Returns an object for querying schema information (see `altova::meta::Element` 1587).
`MemberType last()`	Returns the last instance of the member element.
`void remove()`	Deletes all occurrences of the element from its parent.
`void removeAt(unsigned int index)`	Deletes the occurrence of the element specified by the index.
`void SetEnumerationValue(int)`	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.

16.8 Reference to Generated Classes (C#)

This chapter includes a description of C# classes generated with MapForce from a DTD or XML schema (see Generating Code from XML Schemas or DTDs 1535). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code does include other supporting classes, which are not listed here and are subject to modification.

16.8.1 Altova.Types.DateTime

This class enables you to process XML attributes or elements that have date and time types, such as `xs:dateTime`.

Constructors

	Name	Description
	`DateTime(DateTime obj)`	Initializes a new instance of the `DateTime` class to the `DateTime` object supplied as argument.
	`DateTime(System.DateTime newvalue)`	Initializes a new instance of the `DateTime` class to the `System.DateTime` object supplied as argument.
	`DateTime(int year, int month, int day, int hour, int minute, double second, int offsetTZ)`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, second, and timezone offset supplied as arguments.
	`DateTime(int year, int month, int day, int hour, int minute, double second)`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, and second supplied as arguments.
	`DateTime(int year, int month, int day)`	Initializes a new instance of the `DateTime` class to the year, month and day supplied as arguments.

Properties

	Name	Description
	`bool HasTimezone`	Gets a Boolean value which indicates if the `DateTime` has a timezone.
	`static DateTime Now`	Gets a `DateTime` object that is set to the current date and time on this computer.
	`short TimezoneOffset`	Gets or sets the timezone offset, in minutes, of the `DateTime` object.

	Name	Description
	`System.DateTime Value`	Gets or sets the value of the `DateTime` object as a `System.DateTime` value.

Methods

	Name	Description
	`int CompareTo(object obj)`	The `DateTime` class implements the `IComparable` interface. This method compares the current instance of `DateTime` to another object and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object. See also https://msdn.microsoft.com/en-us/library/system.icomparable.compareto(v=vs.110).aspx
	`override bool Equals(object obj)`	Returns **true** if the specified object is equal to the current object; **false** otherwise.
	`System.DateTime GetDateTime(bool correctTZ)`	Returns a `System.DateTime` object from the current `Altova.Types.DateTime` instance. The `correctTZ` Boolean argument specifies whether the time of the returned object must be adjusted according to the timezone of the current `Altova.Types.DateTime` instance.
	`override int GetHashCode()`	Returns the hash code of the current instance.
	`int GetWeekOfMonth()`	Returns the number of the week in month as an integer.

	`static DateTime Parse( string s )`	Creates a `DateTime` object from the string supplied as argument. For example, the following sample string values would be converted successfully to a `DateTime` object:
		
		`2015-01-01T23:23:23`
		`2015-01-01`
		`2015-11`
		`23:23:23`
		
		An exception is raised if the string cannot be converted to a `DateTime` object.
		
		Note that this method is static and can only be called on the `Altova.Types.DateTime` class itself, not on an instance of the class.

	`static DateTime Parse(string s, DateTimeFormat format)`	Creates a `DateTime` object from a string, using the format supplied as argument. For the list of possible formats, see Altova.Types.DateTimeFormat 1597. An exception is raised if the string cannot be converted to a `DateTime` object.

	Name	Description
		Note that this method is static and can only be called on the `Altova.Types.DateTime` class itself, not on an instance of the class.
	`override string ToString()`	Converts the `DateTime` object to a string.
	`string ToString(DateTimeFormat format)`	Converts the `DateTime` object to a string, using the format supplied as argument. For the list of possible formats, see Altova.Types.DateTimeFormat 1597.

Operators

Name	Description
!=	Determines if `DateTime` a is not equal to `DateTime` b.
<	Determines if `DateTime` a is less than `DateTime` b.
<=	Determines if `DateTime` a is less than or equal to `DateTime` b.
==	Determines if `DateTime` a is equal to `DateTime` b.
>	Determines if `DateTime` a is greater than `DateTime` b.
>=	Determines if `DateTime` a is greater than or equal to `DateTime` b.

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
using Altova.Types;
```

The following code listing illustrates various ways to create `DateTime` objects:

```
protected static void DateTimeExample1()
{
   // Create a DateTime object from the current system time
   Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);
   Console.WriteLine("The current time is: " + dt.ToString());

   // Create an Altova DateTime object from parts (no timezone)
   Altova.Types.DateTime dt1 = new Altova.Types.DateTime(2015, 10, 12, 10, 50, 33);
   Console.WriteLine("My custom time is : " + dt1.ToString());

   // Create an Altova DateTime object from parts (with UTC+60 minutes timezone)
   Altova.Types.DateTime dt2 = new Altova.Types.DateTime(2015, 10, 12, 10, 50, 33, 60);
   Console.WriteLine("My custom time with timezone is : " + dt2.ToString());
```

```
   // Create an Altova DateTime object by parsing a string
   Altova.Types.DateTime dt3 = Altova.Types.DateTime.Parse("2015-01-01T23:23:23");
   Console.WriteLine("Time created from string: " + dt3.ToString());

   // Create an Altova DateTime object by parsing a string formatted as schema date
   Altova.Types.DateTime dt4 = Altova.Types.DateTime.Parse("2015-01-01",
DateTimeFormat.W3_date);
   Console.WriteLine("Time created from string formatted as schema date: " +
dt4.ToString());
}
```

The following code listing illustrates various ways to format `DateTime` objects:

```
protected static void DateTimeExample2()
{
   // Create a DateTime object from the current system time
   Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);

   // Output the unformatted DateTime
   Console.WriteLine("Unformatted time: " + dt.ToString());

   // Output this DateTime formatted using various formats
   Console.WriteLine("S_DateTime:       " + dt.ToString(DateTimeFormat.S_DateTime));
   Console.WriteLine("S_Days:           " + dt.ToString(DateTimeFormat.S_Days));
   Console.WriteLine("S_Seconds:        " + dt.ToString(DateTimeFormat.S_Seconds));
   Console.WriteLine("W3_date:          " + dt.ToString(DateTimeFormat.W3_date));
   Console.WriteLine("W3_dateTime:      " + dt.ToString(DateTimeFormat.W3_dateTime));
   Console.WriteLine("W3_gDay:          " + dt.ToString(DateTimeFormat.W3_gDay));
   Console.WriteLine("W3_gMonth:        " + dt.ToString(DateTimeFormat.W3_gMonth));
   Console.WriteLine("W3_gMonthDay:     " + dt.ToString(DateTimeFormat.W3_gMonthDay));
   Console.WriteLine("W3_gYear:         " + dt.ToString(DateTimeFormat.W3_gYear));
   Console.WriteLine("W3_gYearMonth:    " + dt.ToString(DateTimeFormat.W3_gYearMonth));
   Console.WriteLine("W3_time:          " + dt.ToString(DateTimeFormat.W3_time));
}
```

16.8.2 Altova.Types.DateTimeFormat

The `DateTimeFormat` enum type has the following constant values:

Value	Description	Example
S_DateTime	Formats the value as standard dateTime, with a precision of a ten-millionth of a second, including timezone.	`2015-11-12 12:19:03.9019132+01:00`
S_Days	Formats the value as number of days elapsed since the UNIX epoch.	`735913.6318973451087962962963`

Value	Description	Example
S_Seconds	Formats the value as number of seconds elapsed since the UNIX epoch, with a precision of a ten-millionth of a second.	`63582937678.0769062`
W3_date	Formats the value as schema date.	`2015-11-12`
W3_dateTime	Formats the value as schema dateTime.	`2015-11-12T15:12:14.5194251`
W3_gDay	Formats the value as schema gDay.	`---12` (assuming that the date is 12th of the month)
W3_gMonth	Formats the value as schema gMonth.	`--11` (assuming that the month is November)
W3_gMonthDay	Formats the value as schema gMonthDay.	`--11-12` (assuming that the date is 12th of November)
W3_gYear	Formats the value as schema gYear.	`2015` (assuming that the year is 2015)
W3_gYearMonth	Formats the value as schema gYearMonth.	`2015-11` (assuming that the year is 2015 and the month is November)
W3_time	Formats the value as schema time, with a precision of a ten-millionth of a second.	`15:19:07.5582719`

16.8.3 Altova.Types.Duration

This class enables you to process XML attributes or elements of type `xs:duration`.

Constructors

	Name	Description
	`Duration(Duration obj)`	Initializes a new instance of the `Duration` class to the `Duration` object supplied as argument.
	`Duration(System.TimeSpan newvalue)`	Initializes a new instance of the `Duration` class to the `System.TimeSpan` object supplied as argument.
	`Duration(long ticks)`	Initializes a new instance of the `Duration` class to the number of ticks supplied as argument.

	Name	Description
	`Duration(int newyears, int newmonths, int days, int hours, int minutes, int seconds, double partseconds, bool bnegative)`	Initializes a new instance of the `Duration` class to a duration built from parts supplied as arguments.

Properties

	Name	Description
	`int Months`	Gets or sets the number of months of the current instance of `Duration`.
	`System.TimeSpan Value`	Gets or sets the value (as `System.TimeSpan`) of the current instance of `Duration`.
	`int Years`	Gets or sets the number of years of the current instance of `Duration`.

Methods

	Name	Description
	`override bool Equals(object other)`	Returns **true** if the specified object is equal to the current object; **false** otherwise.
	`override int GetHashCode()`	Returns the hash code of the current instance.
	`bool IsNegative()`	Returns **true** if the current instance of `Duration` represents a negative duration.
	`static Duration Parse( string s, ParseType pt )`	Returns an `Altova.Types.Duration` object parsed from the string supplied as argument, using the parse type supplied as argument. Valid parse type values: **DURATION** Parse duration assuming that year, month, day, as well as time duration parts exist. **YEARMONTH** Parse duration assuming that only year and month parts exist. **DAYTIME** Parse duration assuming that only the day and time parts exist. Note that this method is static and can only be called on the class itself, not on an instance of the class.
	`override string ToString()`	Converts the current `Duration` instance to string. For example, a time span of 3 hours, 4 minutes, and 5 seconds would be converted to "PT3H4M5S".

	Name	Description
	string ToYearMonthString()	Converts the current Duration instance to string, using the "Year and Month" parse type.

Operators

Name	Description
!=	Determines if Duration a is not equal to Duration b.
==	Determines if Duration a is equal to Duration b.

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
using Altova.Types;
```

The following code listing illustrates various ways to create Duration objects:

```
protected static void DurationExample1()
{
   // Create a new time span of 3 hours, 4 minutes, and 5 seconds
   System.TimeSpan ts = new TimeSpan(3, 4, 5);
   // Create a Duration from the time span
   Duration dr = new Duration(ts);
   // The output is: PT3H4M5S
   Console.WriteLine("Duration created from TimeSpan: " + dr.ToString());

   // Create a negative Altova.Types.Duration from 6 years, 5 months, 4 days, 3 hours,
       // 2 minutes, 1 second, and .33 of a second
   Duration dr1 = new Duration(6, 5, 4, 3, 2, 1, .33, true);
   // The output is: -P6Y5M4DT3H2M1.33S
   Console.WriteLine("Duration created from parts: " + dr1.ToString());

   // Create a Duration from a string using the DAYTIME parse type
   Duration dr2 = Altova.Types.Duration.Parse("-P4DT3H2M1S", Duration.ParseType.DAYTIME);
   // The output is -P4DT3H2M1S
   Console.WriteLine("Duration created from string: " + dr2.ToString());

   // Create a duration from ticks
   Duration dr3 = new Duration(System.DateTime.UtcNow.Ticks);
   // Output the result
   Console.WriteLine("Duration created from ticks: " + dr3.ToString());
}
```

The following code listing illustrates getting values from Duration objects:


```
protected static void DurationExample2()
{
   // Create a negative Altova.Types.Duration from 6 years, 5 months, 4 days, 3 hours,
       // 2 minutes, 1 second, and .33 of a second
   Duration dr = new Duration(6, 5, 4, 3, 2, 1, .33, true);
   // The output is: -P6Y5M4DT3H2M1.33S
   Console.WriteLine("The complete duration is: " + dr.ToString());

   // Get only the year and month part as string
   string dr1 = dr.ToYearMonthString();
   Console.WriteLine("The YEARMONTH part is: " + dr1);

   // Get the number of years in duration
   Console.WriteLine("Years: " + dr.Years);

   // Get the number of months in duration
   Console.WriteLine("Months: " + dr.Months);
}
```

16.8.4 Altova.Xml.Meta.Attribute

This class enables you to access schema information about classes generated from attributes. Note that this class is not meant to provide dynamic information about particular instances of an attribute in an XML document. Instead, it enables you to obtain programmatically information about a particular attribute defined in the XML schema.

Properties

	Name	Description
	`SimpleType DataType`	Returns the type of the attribute content.
	`string LocalName`	Returns the local name of the attribute.
	`string NamespaceURI`	Returns the namespace URI of the attribute.
	`XmlQualifiedName QualifiedName`	Returns the qualified name of the attribute.
	`bool Required()`	Returns true if the attribute is required.

16.8.5 Altova.Xml.Meta.ComplexType

This class enables you to access schema information about classes generated from complex types. Note that this class is not meant to provide dynamic information about particular instances of a complex type in an XML document. Instead, it enables you to obtain programmatically information about a particular complex type defined in the XML schema.

Properties

	Name	Description
	`Attribute[] Attributes`	Returns a list of all attributes.
	`ComplexType BaseType`	Returns the base type of this type or null if no base type exists.
	`SimpleType ContentType`	Returns the simple type of the content.
	`Element[] Elements`	Returns a list of all elements.
	`string LocalName`	Returns the local name of the type.
	`string NamespaceURI`	Returns the namespace URI of the type.
	`XmlQualifiedName QualifiedName`	Returns the qualified name of this type.

Methods

	Name	Description
	`ComplexType BaseType`	Returns the base type of this type.
	`bool Equals(obj)`	Checks if two info objects refer to the same type, based on qualified name comparison. Returns true if the type has the same qualified name.
	`Attribute FindAttribute(string localName, string namespaceURI)`	Finds the attribute with the specified local name and namespace URI.
	`Element FindElement(string localName, string namespaceURI)`	Finds the element with the specified local name and namespace URI.

16.8.6 Altova.Xml.Meta.Element

This class enables you to access information about classes generated from schema elements. Note that this class is not meant to provide dynamic information about particular instances of an element in an XML document. Instead, it enables you to obtain programmatically information about a particular element defined in the XML schema.

Properties

	Name	Description
	`ComplexType DataType`	Returns the type of the element. Note that this is always a

	Name	Description
		complex type even if declared as simple in the original schema. Use the `ContentType` property of the returned object to get the simple content type.
	string LocalName	Returns the local name of the element.
	int MaxOccurs	Returns the `maxOccurs` value defined in the schema.
	int MinOccurs	Returns the `minOccurs` value defined in the schema.
	string NamespaceURI	Returns the namespace URI of the element.
	XmlQualifiedName QualifiedName	Returns the qualified name of the element.

16.8.7 Altova.Xml.Meta.SimpleType

This class enables you to access schema information about classes generated from simple types. Note that this class is not meant to provide dynamic information about particular instances of simple types in an XML document. Instead, it enables you to obtain programmatically information about a particular simple type defined in the XML schema.

Properties

	Name	Description
	SimpleType BaseType	Returns the base type of this type.
	string[] Enumerations	Returns a list of all enumeration facets.
	int FractionDigits	Returns the value of this facet.
	int Length	Returns the value of this facet.
	string LocalName	Returns the local name of the type.
	string MaxExclusive	Returns the value of this facet.
	string MaxInclusive	Returns the value of this facet.
	int MaxLength	Returns the value of this facet.
	string MinExclusive	Returns the value of this facet.
	string MinInclusive	Returns the value of this facet.
	int MinLength	Returns the value of this facet.
	string NamespaceURI	Returns the namespace URI of the type.
	string[] Patterns	Returns the pattern facets, or null if no patterns are specified.

	Name	Description
	`XmlQualifiedName QualifiedName`	Returns the qualified name of this type.
	`int TotalDigits`	Returns the value of this facet.
	`WhitespaceType Whitespace`	Returns the whitespace normalization facet.

16.8.8 [YourSchema].[Doc]

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the members listed below. Note that, in the method names below, "Doc" stands for the name of the generated document class itself.

Methods

	Name	Description
	`static Doc CreateDocument()`	Creates a new, empty XML document.
	`static Doc CreateDocument(string encoding)`	Creates a new, empty XML document, with encoding of type "encoding".
	`static void DeclareAllNamespacesFromSchema(Altova.Xml.ElementType node)`	Declares all namespaces from the XML Schema on the element supplied as argument (typically, the XML root element). Calling this method is useful if your schema has multiple namespace declarations, each mapped to a prefix, and you would like to declare all of them on the element supplied as argument.
	`static Doc LoadFromBinary(byte[] binary)`	Loads an XML document from a byte array.
	`static Doc LoadFromFile(string filename)`	Loads an XML document from a file.
	`static Doc LoadFromString(string xmlstring)`	Loads an XML document from a string.
	`byte[] SaveToBinary(bool prettyPrint)`	Saves an XML document to a byte array, with optional "pretty-print" formatting.
	`byte[] SaveToBinary(bool prettyPrint, string encoding)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding.
	`byte[] SaveToBinary(bool prettyPrint, string encoding, bool bBigEndian, bool bBOM)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding, byte order, and BOM (Byte Order Mark).

	Name	Description
	void SaveToFile(**string** fileName, **bool** prettyPrint)	Saves an XML document to a file, with optional "pretty-print" formatting.
	void SaveToFile(**string** fileName, **bool** prettyPrint, **bool** omitXmlDecl)	Saves an XML document to a file, with optional "pretty-print" formatting. When `omitXmlDecl` is true, the XML declaration will not be written.
	void SaveToFile(**string** fileName, **bool** prettyPrint, **bool** omitXmlDecl, **string** encoding)	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. When `omitXmlDecl` is true, the XML declaration will not be written.
	void SaveToFile(**string** fileName, **bool** prettyPrint, **string** encoding, **string** lineend)	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding, and line ending character(s).
	void SaveToFile(**string** fileName, **bool** prettyPrint, **bool** omitXmlDecl, **string** encoding, **string** lineend)	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding, and line ending character(s). When `omitXmlDecl` is true, the XML declaration will not be written.
	void SaveToFile(**string** fileName, **bool** prettyPrint, **bool** omitXmlDecl, **string** encoding, **bool** bBigEndian, **bool** bBOM, **string** lineend)	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding, byte order, BOM (Byte Order Mark), and line ending character(s). When `omitXmlDecl` is true, the XML declaration will not be written.
	void SaveToFileWithLineEnd(**string** fileName, **bool** prettyPrint, **bool** omitXmlDecl, **string** lineend)	Saves an XML document to a file, with optional "pretty-print" formatting, and line ending character(s). When `omitXmlDecl` is true, the XML declaration will not be written.
	string SaveToString(**bool** prettyPrint)	Saves an XML document to a file, with optional "pretty-print" formatting.
	string SaveToString(**bool** prettyPrint, **bool** omitXmlDecl)	Saves an XML document to a file, with optional "pretty-print" formatting. When `omitXmlDecl` is true, the XML declaration will not be written.
	void SetDTDLocation(**string** dtdLocation)	Adds a DOCTYPE declaration with the specified system ID. A root element must already exist.
	void SetSchemaLocation(**string** schemaLocation)	Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.

16.8.9 [YourSchema].[ElementType]

This class provides methods for manipulating XML elements from your schema. Methods of this class can be called on elements, not on the XML document itself. Note that, in order to call methods of this class, you don't need to instantiate the class directly. Any element created using the **`Append()`** or **`AppendWithPrefix()`** methods is of `[ElementType]` type.

Methods

	Name	Description

	`void DeclareNamespace(string prefix, string nsURI)`	This method takes two arguments that are both of string type: the prefix and the namespace URI that you want to use. The prefix supplied as argument will be mapped to the namespace URI value supplied as argument. If the prefix supplied as argument is empty, the method creates or overrides the default namespace declaration in the element.
		
		For example, let's assume that the XML document has an XML element called "purchase". If you call
		
		`purchase.DeclareNamespace("ord", "http://OrderTypes");`
		
		then the XML document becomes
		
		`<purchase xmlns:ord="http://OrderTypes" />`
		
		Another example, if you call:
		
		`purchase.DeclareNamespace("", "http://OrderTypes");`
		
		then the XML document becomes
		
		`<purchase xmlns="http://OrderTypes" />`
		
		Note: The declared namespace is used when appending subsequent child elements or attributes, according to the following rules:
		
		1. If the child namespace is the default, then use empty prefix.
		2. If the child namespace is equal to the parent one, then use the parent prefix.
		3. Otherwise, search for nearest prefix from parent to top, using the lookup algorithm described in section "B.2: Namespace Prefix Lookup" at

	Name	Description
		https://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20021022/namespaces-algorithms.html. 4. If there is no prefix for element namespace found, then use empty prefix.

16.8.10 [YourSchemaType].MemberAttribute

When code is generated from an XML schema, a class is created for each member attribute of a type. In the descriptions below, "AttributeType" stands for the type of the member attribute itself.

Methods

	Name	Description
	`bool Exists()`	Returns true if the attribute exists.
	`void Remove()`	Removes the attribute from its parent element.

Properties

	Name	Description
	`int` `EnumerationValue`	Generated for enumeration types only. Sets or gets the attribute value using one of the constants generated for the possible values. Returns **Invalid** if the value does not match any of the enumerated values in the schema.
	`Altova.Xml.Meta.Attribute Info`	Returns an object for querying schema information (see `Altova.Xml.Meta.Attribute` 1601).
	`AttributeType Value`	Sets or gets the attribute value.

16.8.11 [YourSchemaType].MemberElement

When code is generated from an XML schema, a class with the following members is created for each member element of a type. The class implements the standard `System.Collections.IEnumerable` interface, so it can be used with the `foreach` statement.

In the descriptions below, "MemberType" stands for the type of the member element itself.

Methods

	Name	Description
	`MemberType Append()`	Creates a new element and appends it to its parent.

	Name	Description
	`MemberType AppendWithPrefix(string prefix)`	Creates a new element having the prefix supplied as argument, and appends it to its parent. For an example, see Example: Purchase Order[1571].
	`MemberType At(int index)`	Returns the member element specified by the index.
	`System.Collections.IEnumerator GetEnumerator()`	Returns an object for iterating instances of the member element.
	`void Remove()`	Deletes all occurrences of the element from its parent.
	`void RemoveAt(int index)`	Deletes the occurrence of the element specified by the index.

Properties

	Name	Description
	`int Count`	Returns the count of elements.
	`int EnumerationValue`	Generated for enumeration types only. Sets or gets the element value using one of the constants generated for the possible values. Returns **Invalid** if the value does not match any of the enumerated values in the schema.
	`bool Exists`	Returns true if at least one element exists.
	`MemberType First`	Returns the first instance of the member element.
	`Altova.Xml.Meta.Element Info`	Returns an object for querying schema information (see `Altova.Xml.Meta.Element`[1602]).
	`MemberType Last`	Returns the last instance of the member element.
	`MemberType this[int index]`	Returns the member element specified by the index.
	`MemberType Value`	Sets or gets the element content (only generated if element can have mixed or simple content).

16.9 Reference to Generated Classes (Java)

This chapter includes a description of Java classes generated with MapForce from a DTD or XML schema (see Generating Code from XML Schemas or DTDs1535). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code does include other supporting classes, which are not listed here and are subject to modification.

16.9.1 com.altova.types.DateTime

This class enables you to process XML attributes or elements that have date and time types, such as `xs:dateTime`.

Constructors

	Name	Description
C	**public** DateTime()	Initializes a new instance of the `DateTime` class to an empty value.
C	**public** DateTime(DateTime newvalue)	Initializes a new instance of the `DateTime` class to the `DateTime` value supplied as argument.
C	**public** DateTime(**int** newyear, **int** newmonth, **int** newday, **int** newhour, **int** newminute, **int** newsecond, **double** newpartsecond, **int** newoffsetTZ)	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, second, the fractional part of the second, and timezone supplied as arguments. The fractional part of the second newpartsecond must be between 0 and 1. The timezone offset newoffsetTZ can be either positive or negative and is expressed in minutes.
C	**public** DateTime(**int** newyear, **int** newmonth, **int** newday, **int** newhour, **int** newminute, **int** newsecond, **double** newpartsecond)	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, second, and the fractional part of a second supplied as arguments.
C	**public** DateTime(**int** newyear, **int** newmonth, **int** newday)	Initializes a new instance of the `DateTime` class to the year, month, and day supplied as arguments.
C	**public** DateTime(Calendar newvalue)	Initializes a new instance of the `DateTime` class to the `java.util.Calendar` value supplied as argument.

Methods

	Name	Description
S	**static** DateTime now()	Returns the current time as a `DateTime` object.

	Name	Description

	static DateTime parse(String s)	Returns a `DateTime` object parsed from the string value supplied as argument. For example, the following sample string values would be converted successfully to a `DateTime` object:
		
		`2015-11-24T12:54:47.969+01:00`
		`2015-11-24T12:54:47`
		`2015-11-24`

	int getDay()	Returns the day of the current `DateTime` instance.
	int getHour()	Returns the hour of the current `DateTime` instance.
	int getMillisecond()	Returns the millisecond of the current `DateTime` instance, as an integer value.
	int getMinute()	Returns the minute of the current `DateTime` instance.
	int getMonth()	Returns the month of the current `DateTime` instance.
	double getPartSecond()	Returns the fractional part of the second of the current `DateTime` instance, as a **double** value. The return value is greater than zero and smaller than one, for example: `0.313`
	int getSecond()	Returns the second of the current `DateTime` instance.
	int getTimezoneOffset()	Returns the timezone offset, in minutes, of the current `DateTime` instance. For example, the timezone "UTC-01:00" would be returned as: `-60`
	Calendar getValue()	Returns the current `DateTime` instance as a `java.util.Calendar` value.
	int getWeekday()	Returns the day in week of the current `DateTime` instance. Values range from 0 through 6, where 0 is Monday (ISO-8601).
	int getYear()	Returns the year of the current `DateTime` instance.
	int hasTimezone()	Returns information about the timezone of the current `DateTime` instance. Possible return values are: `CalendarBase.TZ_MISSING` A timezone offset is not defined. `CalendarBase.TZ_UTC` The timezone is UTC. `CalendarBase.TZ_OFFSET` A timezone offset has been defined.
	void setDay(**int** nDay)	Sets the day of the current `DateTime` instance to the value supplied as argument.

	Name	Description
	void setHasTimezone(**int** nHasTZ)	Sets the timezone information of the current `DateTime` instance to the value supplied as argument. This method can be used to strip the timezone information or set the timezone to UTC (Coordinated Universal Time). Valid values for the nHasTZ argument: `CalendarBase.TZ_MISSING` Set the timezone offset to undefined. `CalendarBase.TZ_UTC` Set the timezone to UTC. `CalendarBase.TZ_OFFSET` If the current object has a timezone offset, leave it unchanged.
	void setHour(**int** nHour)	Sets the hour of the current `DateTime` instance to the value supplied as argument.
	void setMinute(**int** nMinute)	Sets the minute of the current `DateTime` instance to the value supplied as argument.
	void setMonth(**int** nMonth)	Sets the month of the current `DateTime` instance to the value supplied as argument.
	void setPartSecond(**double** nPartSecond)	Sets the fractional part of the second of the current `DateTime` instance to the value supplied as argument.
	void setSecond(**int** nSecond)	Sets the second of the current `DateTime` instance to the value supplied as argument.
	void setTimezoneOffset(**int** nOffsetTZ)	Sets the timezone offset of the current `DateTime` instance to the value supplied as argument. The value nOffsetTZ must be an integer (positive or negative) and must be expressed in minutes.
	void setYear(**int** nYear)	Sets the year of the current `DateTime` instance to the value supplied as argument.
	String toString()	Returns the string representation of the current `DateTime` instance, for example: `2015-11-24T15:50:56.968+01:00`

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
import com.altova.types.*;
```

The following code listing illustrates various ways to create `DateTime` objects:

```
protected static void DateTimeExample1()
{
   // Initialize a new instance of the DateTime class to the current time
   DateTime dt = new DateTime(DateTime.now());
   System.out.println("DateTime created from current date and time: " + dt.toString());

   // Initialize a new instance of the DateTime class by supplying the parts
   DateTime dt1 = new DateTime(2015, 11, 23, 14, 30, 24, .459);
   System.out.println("DateTime from parts (no timezone): " + dt1.toString());

   // Initialize a new instance of the DateTime class by supplying the parts
   DateTime dt2 = new DateTime(2015, 11, 24, 14, 30, 24, .459, -60);
   System.out.println("DateTime from parts (with negative timezone): " + dt2.toString());

   // Initialize a new instance of the DateTime class by parsing a string value
   DateTime dt3 = DateTime.parse("2015-11-24T12:54:47.969+01:00");
   System.out.println("DateTime parsed from string: " + dt3.toString());
}
```

The following code listing illustrates getting values from `DateTime` objects:

```
protected static void DateTimeExample2()
   {
      // Initialize a new instance of the DateTime class to the current time
      DateTime dt = new DateTime(DateTime.now());

      // Output the formatted year, month, and day of this DateTime instance
      String str1 = String.format("Year: %d; Month: %d; Day: %d;", dt.getYear(),
dt.getMonth(), dt.getDay());
      System.out.println(str1);

      // Output the formatted hour, minute, and second of this DateTime instance
      String str2 = String.format("Hour: %d; Minute: %d; Second: %d;", dt.getHour(),
dt.getMinute(), dt.getSecond());
      System.out.println(str2);

      // Return the timezone (in minutes) of this DateTime instance
      System.out.println("Timezone: " + dt.getTimezoneOffset());

      // Get the DateTime as a java.util.Calendar value
      java.util.Calendar dt_java = dt.getValue();
      System.out.println("" + dt_java.toString());

      // Return the day of week of this DateTime instance
      System.out.println("Weekday: " + dt.getWeekday());

      // Check whether the DateTime instance has a timezone defined
      switch(dt.hasTimezone())
      {
         case CalendarBase.TZ_MISSING:
            System.out.println("No timezone.");
            break;
         case CalendarBase.TZ_UTC:
```



```
            System.out.println("The timezone is UTC.");
            break;
         case CalendarBase.TZ_OFFSET:
            System.out.println("This object has a timezone.");
            break;
         default:
            System.out.println("Unable to determine whether a timezone is defined.");
            break;
      }
   }
```

The following code listing illustrates changing the timezone offset of a `DateTime` object:

```
protected static void DateTimeExample3()
{
   // Create a new DateTime object with timezone -0100 UTC
   DateTime dt = new DateTime(2015, 11, 24, 14, 30, 24, .459, -60);
   // Output the value before the change
   System.out.println("Before: " + dt.toString());
   // Change the offset to +0100 UTC
   dt.setTimezoneOffset(60);
   // Output the value after the change
   System.out.println("After:  " + dt.toString());
}
```

16.9.2 com.altova.types.Duration

This class enables you to process XML attributes or elements of type `xs:duration`.

Constructors

	Name	Description
C	`Duration(Duration newvalue)`	Initializes a new instance of the `Duration` class to the `Duration` object supplied as argument.
C	`Duration(int newyear, int newmonth, int newday, int newhour, int newminute, int newsecond, double newpartsecond, boolean newisnegative)`	Initializes a new instance of the `Duration` class to a duration built from parts supplied as arguments.

Methods

	Name	Description
S	`static Duration getFromDayTime( int newday,`	Returns a `Duration` object created from the number of days, hours, minutes, seconds, and fractional second parts supplied as

	Name	Description
	int newhour, **int** newminute, **int** newsecond, **double** newpartsecond)	argument.
	static Duration getFromYearMonth(**int** newyear, **int** newmonth)	Returns a `Duration` object created from the number of years and months supplied as argument.
	static Duration parse(String s)	Returns a `Duration` object created from the string supplied as argument. For example, the string -P1Y1M1DT1H1M1.333S can be used to create a negative duration of one year, one month, one day, one hour, one minute, one second, and 0.333 fractional parts of a second. To create a negative duration, append the minus sign (**-**) to the string.
	static Duration parse(String s, ParseType pt)	Returns a `Duration` object created from the string supplied as argument, using a specific parse format. The parse format can be any of the following: **ParseType.DAYTIME** — May be used when the string **s** consists of any of the following: days, hours, minutes, seconds, fractional second parts, for example -P4DT4H4M4.774S. **ParseType.DURATION** — May be used when the string **s** consists of any of the following: years, months, days, hours, minutes, seconds, fractional second parts, for example P1Y1M1DT1H1M1.333S. **ParseType.YEARMONTH** — May be used when the string **s** consists of any of the following: years, months. For example: P3Y2M.
	int getDay()	Returns the number of days in the current `Duration` instance.
	long getDayTimeValue()	Returns the day and time value (in milliseconds) of the current `Duration` instance. Years and months are ignored.
	int getHour()	Returns the number of hours in the current `Duration` instance.
	int getMillisecond()	Returns the number of milliseconds in the current `Duration` instance.
	int getMinute()	Returns the number of minutes in the current `Duration` instance.
	int getMonth()	Returns the number of months in the current `Duration` instance.
	double getPartSecond()	Returns the number of fractional second parts in the current `Duration` instance.

	Name	Description
●	**int** getSecond()	Returns the number of seconds in the current `Duration` instance.
●	**int** getYear()	Returns the number of years in the current `Duration` instance.
●	**int** getYearMonthValue()	Returns the year and month value (in months) of the current `Duration` instance. Days, hours, seconds, and milliseconds are ignored.
●	**boolean** isNegative()	Returns Boolean **true** if the current `Duration` instance is negative.
●	**void** setDayTimeValue(**long** l)	Sets the duration to the number of milliseconds supplied as argument, affecting only the day and time part of the duration.
●	**void** setNegative(**boolean** isnegative)	Converts the current `Duration` instance to a negative duration.
●	**void** setYearMonthValue(**int** l)	Sets the duration to the number of months supplied as argument. Only the years and months part of the duration is affected.
●	String toString()	Returns the string representation of the current `Duration` instance, for example: `-P4DT4H4M4.774S`
●	String toYearMonthString()	Returns the string representation of the YearMonth part of the current `Duration` instance, for example: `P1Y2M`

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
import com.altova.types.*;
import com.altova.types.Duration.ParseType;
```

The following code listing illustrates various ways to create `Duration` objects:

```
protected static void ExampleDuration()
{
   // Create a negative duration of 1 year, 1 month, 1 day, 1 hour, 1 minute, 1 second,
   // and 0.333 fractional second parts
   Duration dr = new Duration(1, 1, 1, 1, 1, 1, .333, true);

   // Create a duration from an existing Duration object
   Duration dr1 = new Duration(dr);

   // Create a duration of 4 days, 4 hours, 4 minutes, 4 seconds, .774 fractional second
```

```
parts
   Duration dr2 = Duration.getFromDayTime(4, 4, 4, 4, .774);

   // Create a duration of 3 years and 2 months
   Duration dr3 = Duration.getFromYearMonth(3, 2);

   // Create a duration from a string
   Duration dr4 = Duration.parse("-P4DT4H4M4.774S");

   // Create a duration from a string, using specific parse formats
   Duration dr5 = Duration.parse("-P1Y1M1DT1H1M1.333S", ParseType.DURATION);
   Duration dr6 = Duration.parse("P3Y2M", ParseType.YEARMONTH);
   Duration dr7 = Duration.parse("-P4DT4H4M4.774S", ParseType.DAYTIME);
}
```

The following code listing illustrates getting and setting the value of `Duration` objects:

```
protected static void DurationExample2()
{
   // Create a duration of 1 year, 2 month, 3 days, 4 hours, 5 minutes, 6 seconds,
       // and 333 milliseconds
   Duration dr = new Duration(1, 2, 3, 4, 5, 6, .333, false);
   // Output the number of days in this duration
   System.out.println(dr.getDay());

   // Create a positive duration of one year and 333 milliseconds
   Duration dr1 = new Duration(1, 0, 0, 0, 0, 0, .333, false);
   // Output the day and time value in milliseconds
   System.out.println(dr1.getDayTimeValue());

   // Create a positive duration of 1 year, 1 month, 1 day, 1 hour, 1 minute, 1 second,
       // and 333 milliseconds
   Duration dr2 = new Duration(1, 1, 1, 1, 1, 1, .333, false);
   // Output the year and month value in months
   System.out.println(dr2.getYearMonthValue());

   // Create a positive duration of 1 year and 1 month
   Duration dr3 = new Duration(1, 1, 0, 0, 0, 0, 0, false);
   // Output the value
   System.out.println("The duration is now: " + dr3.toString());
   // Set the DayTime part of duration to 1000 milliseconds
   dr3.setDayTimeValue(1000);
   // Output the value
   System.out.println("The duration is now: " + dr3.toString());
   // Set the YearMonth part of duration to 1 month
   dr3.setYearMonthValue(1);
   // Output the value
   System.out.println("The duration is now: " + dr3.toString());
   // Output the year and month part of the duration
   System.out.println("The YearMonth part of the duration is: " +
dr3.toYearMonthString());
}
```


16.9.3 com.altova.xml.meta.Attribute

This class enables you to access schema information about classes generated from attributes. Note that this class is not meant to provide dynamic information about particular instances of an attribute in an XML document. Instead, it enables you to obtain programmatically information about a particular attribute defined in the XML schema.

Methods

	Name	Description
	`SimpleType getDataType()`	Returns the type of the attribute content.
	`String getLocalName()`	Returns the local name of the attribute.
	`String getNamespaceURI()`	Returns the namespace URI of the attribute.
	`boolean isRequired()`	Returns true if the attribute is required.

16.9.4 com.altova.xml.meta.ComplexType

This class enables you to access schema information about classes generated from complex types. Note that this class is not meant to provide dynamic information about particular instances of a complex type in an XML document. Instead, it enables you to obtain programmatically information about a particular complex type defined in the XML schema.

Methods

	Name	Description
	`Attribute findAttribute(String localName, String namespaceURI)`	Finds the attribute with the specified local name and namespace URI.
	`Element findElement(String localName, String namespaceURI)`	Finds the element with the specified local name and namespace URI.
	`Attribute[] GetAttributes()`	Returns a list of all attributes.
	`ComplexType getBaseType()`	Returns the base type of this type.
	`SimpleType getContentType()`	Returns the simple type of the content.
	`Element[] GetElements()`	Returns a list of all elements.
	`String getLocalName()`	Returns the local name of the type.
	`String getNamespaceURI()`	Returns the namespace URI of the type.

16.9.5 com.altova.xml.meta.Element

This class enables you to access information about classes generated from schema elements. Note that this class is not meant to provide dynamic information about particular instances of an element in an XML document. Instead, it enables you to obtain programmatically information about a particular element defined in the XML schema.

Methods

	Name	Description
●	`ComplexType getDataType()`	Returns the type of the element. Note that this is always a complex type even if declared as simple in the original schema. Use `getContentType()` of the returned object to get the simple content type.
●	`String getLocalName()`	Returns the local name of the element.
●	`int getMaxOccurs()`	Returns the maxOccurs value defined in the schema.
●	`int getMinOccurs()`	Returns the minOccurs value defined in the schema.
●	`String getNamespaceURI()`	Returns the namespace URI of the element.

16.9.6 com.altova.xml.meta.SimpleType

This class enables you to access schema information about classes generated from simple types. Note that this class is not meant to provide dynamic information about particular instances of simple types in an XML document. Instead, it enables you to obtain programmatically information about a particular simple type defined in the XML schema.

Methods

	Name	Description
●	`SimpleType getBaseType()`	Returns the base type of this type.
●	`String[] getEnumerations()`	Returns an array of all enumeration facets.
●	`int getFractionDigits()`	Returns the value of this facet.
●	`int getLength()`	Returns the value of this facet.
●	`String getLocalName()`	Returns the local name of the type.
●	`String getMaxExclusive()`	Returns the value of this facet.
●	`String getMaxInclusive()`	Returns the value of this facet.
●	`int getMaxLength()`	Returns the value of this facet.

	Name	Description
●	`String getMinExclusive()`	Returns the value of this facet.
●	`String getMinInclusive()`	Returns the value of this facet.
●	`int getMinLength()`	Returns the value of this facet.
●	`String getNamespaceURI()`	Returns the namespace URI of the type.
●	`String[] getPatterns()`	Returns an array of all pattern facets.
●	`int getTotalDigits()`	Returns the value of this facet.

●	`int getWhitespace()`	Returns the value of the whitespace facet, which is one of:
		`com.altova.typeinfo.WhitespaceType.Whitespace_Unknown`
		`com.altova.typeinfo.WhitespaceType.Whitespace_Preserve`
		`com.altova.typeinfo.WhitespaceType.Whitespace_Replace`
		`com.altova.typeinfo.WhitespaceType.Whitespace_Collapse`

16.9.7 com.[YourSchema].[Doc]

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the members listed below. Note that, in the method names below, "Doc" stands for the name of the generated document class itself.

Methods

	Name	Description
● S	`static Doc createDocument()`	Creates a new, empty XML document.
● S	`static void declareAllNamespacesFromSchema(com.altova.xml.ElementType node)`	Declares all namespaces from the XML Schema on the element supplied as argument (typically, the XML root element). Calling this method is useful if your schema has multiple namespace declarations, each mapped to a prefix, and you would like to declare all of them on the element supplied as argument.
● S	`static Doc loadFromBinary(byte[] xml)`	Loads an XML document from a byte array.
● S	`static Doc loadFromFile(String fileName)`	Loads an XML document from a file.
● S	`static Doc loadFromString(String xml)`	Loads an XML document from a string.
●	`byte[] saveToBinary(boolean prettyPrint)`	Saves an XML document to a byte array, with optional "pretty-print" formatting.

	Name	Description
●	`byte[] saveToBinary(boolean prettyPrint, String encoding)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding.
●	`byte[] saveToBinary(boolean prettyPrint, String encoding, boolean bigEndian, boolean writeBOM)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
●	`void saveToFile(String fileName, boolean prettyPrint)`	Saves an XML document to a file, with optional "pretty-print" formatting.
●	`void saveToFile(String fileName, boolean prettyPrint, boolean omitXmlDecl)`	Saves an XML document to a file, with optional "pretty-print" formatting, with UTF-8 encoding. When `omitXmlDecl` is true, the XML declaration will not be written.
●	`void saveToFile(String fileName, boolean prettyPrint, boolean omitXmlDecl, String encoding)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. When `omitXmlDecl` is true, the XML declaration will not be written.
●	`void saveToFile(String fileName, boolean prettyPrint, boolean omitXmlDecl, String encoding, boolean bBigEndian, boolean bBOM)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. When `omitXmlDecl` is true, the XML declaration will not be written. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
●	`void saveToFile(String fileName, boolean prettyPrint, String encoding)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding.
●	`void saveToFile(String fileName, boolean prettyPrint, String encoding, boolean bBigEndian, boolean bBOM)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
●	`String saveToString(boolean prettyPrint)`	Saves an XML document to a string, with optional "pretty-print" formatting.
●	`String saveToString(boolean prettyPrint, boolean omitXmlDecl)`	Saves an XML document to a string, with optional "pretty-print" formatting. When `omitXmlDecl` is true, the XML declaration will not be written.
●	`void setSchemaLocation(String schemaLocation)`	Adds an `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` attribute to the root element. A root element must already exist.

16.9.8 com.[YourSchema].[ElementType]

This class provides methods for manipulating XML elements from your schema. Methods of this class can be called on elements, not on the XML document itself. Note that, in order to call methods of this class, you don't need to instantiate the class directly. Any element created using the **`append()`** or **`appendWithPrefix()`** methods is of `[ElementType]` type.

Methods

	Name	Description

●	`void declareNamespace(String prefix, String nsURI)`	This method takes two arguments that are both of string type: the prefix and the namespace URI that you want to use. The prefix supplied as argument will be mapped to the namespace URI value supplied as argument. If the prefix supplied as argument is empty, the method creates or overrides the default namespace declaration in the element.
		
		For example, let's assume that the XML document has an XML element called "purchase". If you call
		
		`purchase.declareNamespace("ord", "http://OrderTypes");`
		
		then the XML document becomes
		
		`<purchase xmlns:ord="http://OrderTypes" />`
		
		Another example, if you call:
		
		`purchase.declareNamespace("", "http://OrderTypes");`
		
		then the XML document becomes
		
		`<purchase xmlns="http://OrderTypes" />`
		
		Note: The declared namespace is used when appending subsequent child elements or attributes, according to the following rules:
		
		1. If the child namespace is the default, then use empty prefix.
		2. If the child namespace is equal to the parent one, then use the parent prefix.
		3. Otherwise, search for nearest prefix from parent to

	Name	Description
		top, using the lookup algorithm described in section "B.2: Namespace Prefix Lookup" at https://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20021022/namespaces-algorithms.html. 4. If there is no prefix for element namespace found, then use empty prefix.

16.9.9 com.[YourSchema].[YourSchemaType].MemberAttribute

When code is generated from an XML schema, a class is created for each member attribute of a type. In the descriptions below, "AttributeType" stands for the type of the member attribute itself.

Methods

	Name	Description
●	**boolean** exists()	Returns true if the attribute exists.
●	**int** getEnumerationValue()	Generated for enumeration types only. Returns one of the constants generated for the possible values, or **Invalid** if the value does not match any of the enumerated values in the schema.
●	com.altova.xml.meta.Attribute getInfo()	Returns an object for querying schema information (see com.altova.xml.meta.Attribute 1617).
●	AttributeType getValue()	Gets the attribute value.
●	**void** remove()	Removes the attribute from its parent element.
●	**void** setEnumerationValue(**int**)	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.
●	**void** setValue(AttributeType value)	Sets the attribute value.

16.9.10 com.[YourSchema].[YourSchemaType].MemberElement

When code is generated from an XML schema, a class with the following members is created for each member element of a type. In the descriptions below, "MemberType" stands for the type of the member element itself.

Methods

	Name	Description
●	MemberType append()	Creates a new element and appends it to its parent.

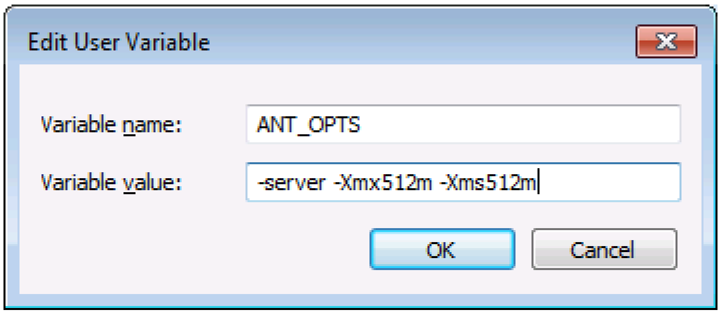
	Name	Description
●	`MemberType appendWithPrefix(String prefix)`	Creates a new element having the prefix supplied as argument, and appends it to its parent. For an example, see Example: Purchase Order[1571].
●	`MemberType at(int index)`	Returns the instance of the member element at the specified index.
●	`int count()`	Returns the count of elements.
●	`boolean exists()`	Returns true if at least one element exists.
●	`MemberType first()`	Returns the first instance of the member element.
●	`int getEnumerationValue()`	Generated for enumeration types only. Returns one of the constants generated for the possible values, or **Invalid** if the value does not match any of the enumerated values in the schema.
●	`com.altova.xml.meta.Element getInfo()`	Returns an object for querying schema information (see `com.altova.xml.meta.Element`[1618]).
●	`MemberType getValue()`	Gets the element content (only generated if element can have simple or mixed content).
●	`java.util.Iterator iterator()`	Returns an object for iterating instances of the member element.
●	`MemberType last()`	Returns the last instance of the member element.
●	`void remove()`	Deletes all occurrences of the element from its parent.
●	`void removeAt(int index)`	Deletes the occurrence of the element specified by the index.
●	`void setEnumerationValue(int index)`	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.
●	`void setValue(MemberType value)`	Sets the element content (only generated if element can have simple or mixed content).

16.10 Code Generation Tips

Resolving "Out of memory" exceptions during Java compilation

Complex mappings with large schemas can produce a large amount of code, which might cause a `java.lang.OutofMemory` exception during compilation using Ant. To rectify this:

- Add the environment variable `ANT_OPTS`, which sets specific Ant options such as the memory to be allocated to the compiler, and set its value as shown below.



- To make sure that the compiler and the generated code run in the same process as Ant, change the `fork` attribute, in **build.xml**, to `false`.

You may need to customize the values depending on the amount of memory in your machine and the size of the project you are working with. For more details, see your Java VM documentation.

> When running the `ant jar` command, you may get an error message similar to "`[...] archive contains more than 65535 entities`". To prevent this, it is recommended that you use Ant 1.9 or later, and, in the **build.xml** file, add `zip64mode="as-needed"` to the `<jar>` element.

Reserving method names

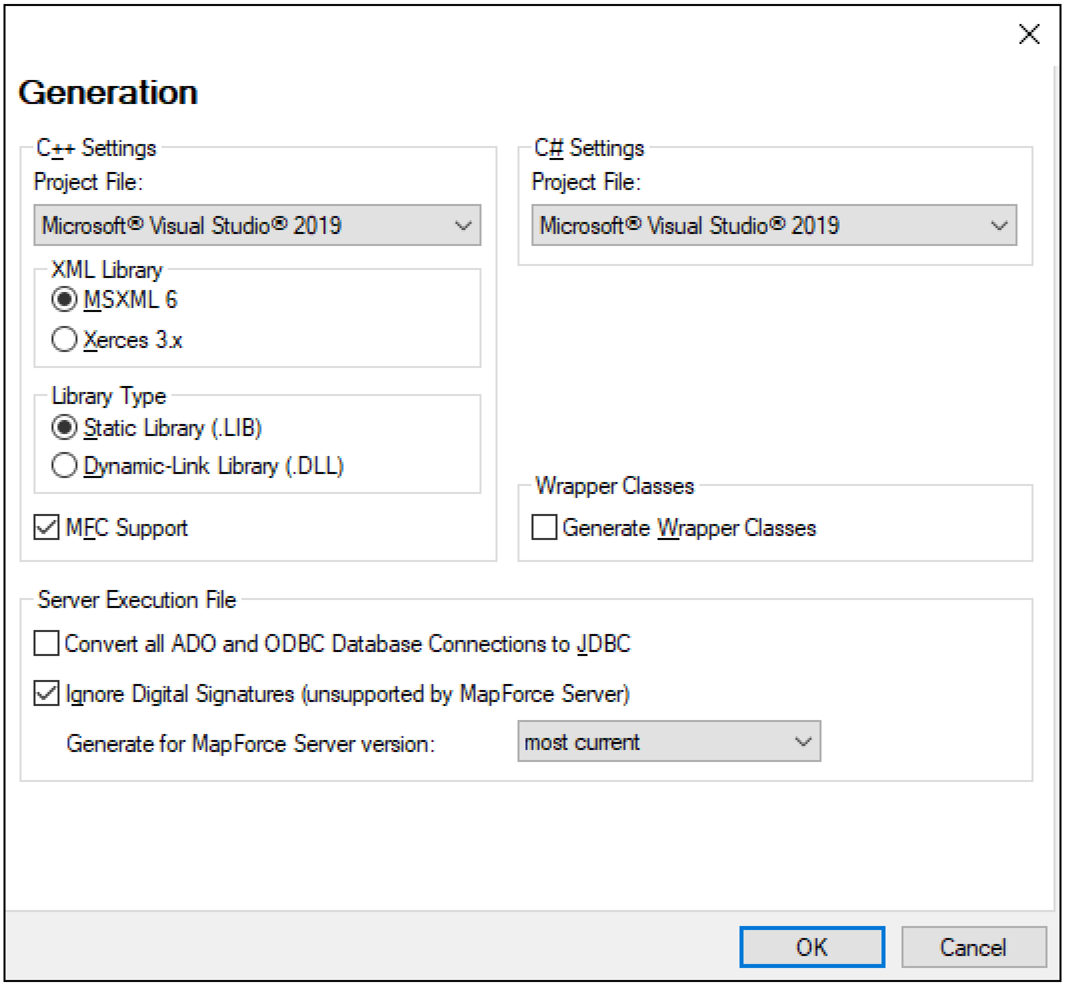
When customizing code generation using the supplied SPL files, it might be necessary to reserve names to avoid collisions with other symbols. To do this:

1. Navigate to the program installation directory, for example, `C:\Program Files\Altova\MapForce2023`.
2. In the **spl** subdirectory, locate the directory corresponding to the programming language, for example, **..\spl\java**.
3. Open the **settings.spl** file and insert a new line into the "reserve" section, for example, **`reserve "myReservedWord"`**.
4. Regenerate the program code.

16.11 Code Generator Options

To view or change the MapForce settings applicable to code generation:

- On the **Tools** menu, click **Options**, and then click **Generation**.



The available settings are as follows.

C++ Settings	Defines the specific compiler settings for the C++ environment, namely: • The Visual Studio version (2013, 2015, 2017, 2019, 2022) • The XML library (MSXML, Xerces 3.x) • Whether static or dynamic libraries must be generated • Whether code must be generated with or without MFC support
C# Settings	Select the option **Microsoft .NET Core 3.1**, **Microsoft .NET 5.0**, or **Microsoft .NET 6.0** to generate a Visual Studio solution targeting the respective platforms.

	If you need to target the **.NET Framework** platform for a specific Visual Studio version, select any of the **Microsoft Visual Studio 2010-2019** options —in this case, the generated solution will target the .NET Framework version corresponding to the respective Visual Studio version.
Wrapper Classes	Allows you to generate wrapper classes for XML schemas, see Generating Code from XML Schemas or DTDs 1535. These wrapper classes can be used by custom code that includes the code generated by MapForce.
Server Execution File	These options are applicable when you compile mappings to MapForce Server execution files. They do not affect generation of C#, C++, or Java code. For more information, see Compiling Mappings to MapForce Server Execution Files 1378.

16.12 SPL Reference

This section gives an overview of SPL (Spy Programming Language), the code generator's template language.

It is assumed that you have prior programming experience, and are familiar with operators, functions, variables and classes, as well as the basics of object-oriented programming - which is used heavily in SPL.

The templates used by MapForce are supplied in the **...\MapForce\spl** folder. You can use these files as an aid to help you in developing your own templates.

How code generator works

Inputs to the code generator are the template files (.spl) and the object model provided by MapForce. The template files contain SPL instructions for creating files, reading information from the object model and performing calculations, interspersed with literal code fragments in the target programming language.

The template file is interpreted by the code generator and outputs **.cpp, .java, .cs** source code files, project files, or any other type of file depending on the template. The source code can then be compiled into an executable file that accesses XML data described by the schema file.

SPL files have access to a wide variety of information that is collated from the source schemas. Note that an SPL file is not tied to a specific schema, but allows access to all schemas. Make sure you write your SPL files generically and avoid structures which apply to specific schemas.

Example: Creating a new file in SPL

This is a very basic SPL file. It creates a file named **test.cpp,** and places the include statement within it. The close command completes the template.

```
[create "test.cpp"]
#include "stdafx.h"
[close]
```

16.12.1 Basic SPL structure

An SPL file contains literal text to output, interspersed with code generator instructions.

Code generator instructions are enclosed in square brackets '**[**' and '**]**'. Multiple statements can be included in a bracket pair. Additional statements have to be separated by a new line or a colon '**:**'.

Valid examples are:

```
[$x = 42
$x = $x + 1]
```

or

```
[$x = 42: $x = $x + 1]
```

Adding text to files

Text not enclosed by **[** and **]**, is written directly to the current output file. If there is no current output file, the text is ignored (see Using files 1632 how to create an output file).

To output literal square brackets, escape them with a backslash: **\[** and **\]**; to output a backslash use ****.

Comments

Comments inside an instruction block always begin with a **'** character, and terminate on the next line, or at a block close character **]**.

16.12.2 Declarations

The following statements are evaluated while parsing the SPL template file. They are **not** affected by flow control statements like conditions, loops or subroutines, and are always evaluated exactly once.

These keywords, like all keywords in SPL, are not case sensitive.

Remember that all of these declarations must be inside a block delimited by square brackets.

map ... to ...

```
map mapname key to value [, key to value ]...
```

This statement adds information to a map. See below for specific uses.

```
map schemanativetype schematype to typespec
```

The specified built-in XML Schema type will be mapped to the specified native type or class, using the specified formatter. This setting applies only to code generation for version 2007r3 and higher. Typespec is a native type or class name, followed by a comma, followed by the formatter class instance.

Example:

```
map schemanativetype "double" to "double,Altova::DoubleFormatter"
```

map type ... to ...

```
map type schematype to classname
```

The specified built-in XML Schema type will be mapped to the specified class. This setting applies only to code generation for version 2007 or lower.

Example:

```
map type "float" to "CSchemaFloat"
```

default ... is ...

```
default setting is value
```

This statement allows you to affect how class and member names are derived from the XML Schema. Note that the setting names are case sensitive.

Example:

```
default "InvalidCharReplacement" is "_"
```

Setting name	Explanation
ValidFirstCharSet	Allowed characters for starting an identifier
ValidCharSet	Allowed characters for other characters in an identifier
InvalidCharReplacement	The character that will replace all characters in names that are not in the ValidCharSet
AnonTypePrefix	Prefix for names of anonymous types*
AnonTypeSuffix	Suffix for names of anonymous types*
ClassNamePrefix	Prefix for generated class names
ClassNameSuffix	Suffix for generated class names
EnumerationPrefix	Prefix for symbolic constants declared for enumeration values
EnumerationUpperCase	"on" to convert the enumeration constant names to upper case
FallbackName	If a name consists only of characters that are not in ValidCharSet, use this one

* Names of anonymous types are built from AnonTypePrefix + element name + AnonTypeSuffix

reserve

```
reserve word
```

Adds the specified word to the list of reserved words. This ensures that it will never be generated as a class or member name.

Example:

```
reserve "while"
```

include

includes the specified file as SPL source. This allows you to split your template into multiple files for easier editing and handling.

```
include filename
```

Example:

```
include "Module.cpp"
```

16.12.3 Variables

Any non-trivial SPL file will require variables. Some variables are predefined [1631] by the code generator, and new variables may be created simply by assigning values to them.

The **$** character is used when **declaring** or **using** a variable, a variable name is always prefixed by **$**. Variable names are **case sensitive**.

Variables types:

- integer - also used as boolean, where 0 is false and everything else is true
- string
- object - provided by MapForce
- iterator - see foreach [1636] statement

Variable types are declared by first assignment:

```
[$x = 0]
```

x is now an integer.

```
[$x = "teststring"]
```

x is now treated as a string.

Strings

String constants are always enclosed in double quotes, like in the example above. **\n** and **\t** inside double quotes are interpreted as newline and tab, **\"** is a literal double quote, and **** is a backslash. String constants can also span multiple lines.

String concatenation uses the **&** character:

```
[$BasePath = $outputpath & "/" & $JavaPackageDir]
```

Objects

Objects represent the information contained in the XML schemas, database structures, text files and mappings. Objects have **properties**, which can be accessed using the **.** operator. It is not possible to create new objects in SPL (they are predefined by the code generator, derived from the input mapping), but it is possible to assign objects to variables.

Example:

```
class [=$class.Name]
```

This example outputs the word "class", followed by a space and the value of the **Name** property of the **$class** object.

16.12.4 Predefined variables

After a Schema file is analyzed by the code generator, the objects in the table below exist in the Template Engine.

Name	Type	Description
$schematype	integer	1 for DTD, 2 for XML Schema
$TheLibrary	Library 1640	The library derived from the XML Schema or DTD
$module	string	Name of the source Schema without extension
$outputpath	string	The output path specified by the user, or the default output path

For C++ generation only:

Name	Type	Description
$domtype	integer	1 for MSXML, 2 for Xerces
$libtype	integer	1 for static LIB, 2 for DLL
$mfc	boolean	True if MFC support is enabled

Name	Type	Description
$VSVersion	integer	Specifies the Visual Studio version. Valid values: **0** No Visual Studio project **2010** Visual Studio 2010 **2013** Visual Studio 2013 **2015** Visual Studio 2015 **2017** Visual Studio 2017 **2019** Visual Studio 2019

For C# generation only:

Name	Type	Description
$VSVersion	integer	Specifies the Visual Studio version. Valid values: **0** No Visual Studio project **2010** Visual Studio 2010 **2013** Visual Studio 2013 **2015** Visual Studio 2015 **2017** Visual Studio 2017 **2019** Visual Studio 2019

16.12.5 Creating output files

These statements are used to create output files from the code generation. Remember that all of these statements must be inside a block delimited by square brackets.

create

```
create filename
```

creates a new file. The file has to be closed with the **close** statement. All following output is written to the specified file.

Example:


```
[create $outputpath & "/" & $JavaPackageDir & "/" & $application.Name & ".java"]
package [=$JavaPackageName];

public class [=$application.Name]Application {
...
}
[close]
```

close

closes the current output file.

```
=$variable
```

writes the value of the specified variable to the current output file.

Example:

```
[$x = 20+3]
The result of your calculation is [=$x] - so have a nice day!
```

The file output will be:

```
The result of your calculation is 23 - so have a nice day!
```

write

```
write string
```

writes the string to the current output file.

Example:

```
[write "C" & $name]
```

This can also be written as:

```
C[=$name]
```

filecopy ... to ...

```
filecopy source to target
```

copies the source file to the target file, without any interpretation.

Example:

```
filecopy "java/mapforce/mapforce.png" to $outputpath & "/" & $JavaPackageDir &
"/mapforce.png"
```

16.12.6 Operators

Operators in SPL work like in most other programming languages.

List of SPL operators in descending precedence order:

.	Access object property
( )	Expression grouping
true	boolean constant "true"
false	boolean constant "false"
&	String concatenation
-	Sign for negative number
not	Logical negation
*	Multiply
/	Divide
%	Modulo
+	Add
-	Subtract
<=	Less than or equal
<	Less than
>=	Greater than or equal
>	Greater than
=	Equal
<>	Not equal
and	Logical conjunction (with short circuit evaluation)
or	Logical disjunction (with short circuit evaluation)
=	Assignment

16.12.7 Conditions

SPL allows you to use standard "if" statements. The syntax is as follows:

```
if condition
	statements
else
	statements
endif
```

or, without else:

```
if condition
	statements
endif
```

Note: There are no round brackets enclosing the condition.

As in any other programming language, conditions are constructed with logical and comparison operators 1634.

Example:

```
[if $namespace.ContainsPublicClasses and $namespace.Prefix <> ""]
	whatever you want ['inserts whatever you want, in the resulting file]
[endif]
```

Switch

SPL also contains a multiple choice statement.

Syntax:

```
switch $variable
	case X:
		statements
	case Y:
	case Z:
		statements
	default:
		statements
endswitch
```

The case labels must be constants or variables.

The switch statement in SPL does not fall through the cases (as in C), so there is no need for a "break" statement.

16.12.8 Collections and foreach

Collections and iterators

A collection contains multiple objects - like a ordinary array. Iterators solve the problem of storing and incrementing array indexes when accessing objects.

Syntax:

```
foreach iterator in collection
       statements
next
```

Example:

```
[foreach $class in $classes
       if not $class.IsInternal
              ]      class [=$class.Name];
[      endif
next]
```

Example 2:

```
[foreach $i in 1 To 3
      Write "// Step " & $i & "\n"
      ‘ Do some work
next]
```

In the first line:

$classes is the **global object** [1631] of all generated types. It is a collection of single class objects.

Foreach steps through all the items in $classes, and executes the code following the instruction, up to the **next** statement, for each of them.

In each iteration, **$class** is assigned to the next class object. You simply work with the class object instead of using, classes[i]->Name(), as you would in C++.

All collection iterators have the following additional properties:

Index	The current index, starting with 0
IsFirst	true if the current object is the first of the collection (index is 0)
IsLast	true if the current object is the last of the collection
Current	The current object (this is implicit if not specified and can be left out)

Example:

```
[foreach $enum in $facet.Enumeration
	if not $enum.IsFirst
		], [
	endif
	]"[=$enum.Value]"[
next]
```

16.12.9 Subroutines

Code generator supports subroutines in the form of procedures or functions.

Features:

- By-value and by-reference passing of values
- Local/global parameters (local within subroutines)
- Local variables
- Recursive invocation (subroutines may call themselves)

16.12.9.1 Subroutine declaration

Subroutines

Syntax example:

```
Sub SimpleSub()

	... lines of code
 EndSub
```

- **Sub** is the keyword that denotes the procedure.
- **SimpleSub** is the name assigned to the subroutine.
- Round **parenthesis** can contain a parameter list.
- The code block of a subroutine starts immediately after the closing parameter parenthesis.
- **EndSub** denotes the end of the code block.

Note: Recursive or cascaded subroutine **declaration** is not permitted, i.e. a subroutine may not contain another subroutine.

Parameters

Parameters can also be passed by procedures using the following syntax:

- All parameters must be variables
- Variables must be prefixed by the **$** character
- Local variables are defined in a subroutine
- Global variables are declared explicitly, outside of subroutines
- Multiple parameters are separated by the comma character "**,**" within round parentheses
- Parameters can pass values

Parameters - passing values

Parameters can be passed in two ways, by value and by reference, using the keywords **ByVal** and **ByRef** respectively.

Syntax:

```
' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
] ...
```

- **ByVal** specifies that the parameter is passed by value. Note that most objects can only be passed by reference.
- **ByRef** specifies that the parameter is passed by reference. This is the default if neither ByVal nor ByRef is specified.

Function return values

To return a value from a subroutine, use the **return** statement. Such a function can be called from within an expression.

Example:

```
' define a function
[Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )
if $namespacePrefix = ""
  return $localName
else
  return $namespacePrefix & ":" & $localName
endif
EndSub
]
```

16.12.9.2 Subroutine invocation

Use **call** to invoke a subroutine, followed by the procedure name and parameters, if any.

```
Call SimpleSub()
```

or

```
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
```

Function invocation

To invoke a function (any subroutine that contains a **return** statement), simply use its name inside an expression. Do not use the **call** statement to call functions. Example:

```
$QName = MakeQualifiedName($namespace, "entry")
```

16.12.9.3 Subroutine example

The following example shows subroutine declaration and invocation.

```
[create $outputpath & $module & "output.txt"

' define sub SimpleSub()
Sub SimpleSub()
]SimpleSub() called
[endsub

' execute sub SimpleSub()
Call SimpleSub()

$ParamByValue       = "Original Value"
]ParamByValue       = [=$ParamByValue]
[$ParamByRef  = "Original Value"
]ParamByRef   = [=$ParamByRef]

' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
]CompleteSub called.
       param = [=$param]
       paramByValue = [=$paramByValue]
       paramByRef = [=$paramByRef]
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]      Set values inside Sub
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]CompleteSub finished.
[endsub

' run sub CompleteSub()
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
]
ParamByValue=[=$ParamByValue]
ParamByRef=[=$ParamByRef]
[
Close
]
```

16.12.10 Built in Types

The section describes the properties of the built-in types used in the predefined variables[1631] which describe the parsed schema.

16.12.10.1 Library

This object represents the whole library generated from the XML Schema or DTD.

Property	Type	Description
SchemaNamespaces	Namespace[1640] collection	Namespaces in this library
SchemaFilename	string	Name of the XSD or DTD file this library is derived from
SchemaType	integer	1 for DTD, 2 for XML Schema
Guid	string	A globally unique ID
CodeName	string	Generated library name (derived from schema file name)

16.12.10.2 Namespace

One namespace object per XML Schema namespace is generated. Schema components that are not in any namespace are contained in a special namespace object with an empty NamespaceURI. Note that for DTD, namespaces are also derived from attributes whose names begin with "xmlns".

Property	Type	Description
CodeName	string	Name for generated code (derived from prefix)
LocalName	string	Namespace prefix
NamespaceURI	string	Namespace URI
Types	Type[1640] collection	All types contained in this namespace
Library	Library[1640]	Library containing this namespace

16.12.10.3 Type

This object represents a complex or simple type. It is used to generate a class in the target language. There is one additional type per library that represents the document, which has all possible root elements as members.

Anonymous types have an empty LocalName.

Property	Type	Description
CodeName	string	Name for generated code (derived from local name or parent declaration)
LocalName	string	Original name in the schema
Namespace	Namespace [1640]	Namespace containing this type
Attributes	Member [1642] collection	Attributes contained in this type*
Elements	Member [1642] collection	Child elements contained in this type
IsSimpleType	boolean	True for simple types, false for complex types
IsDerived	boolean	True if this type is derived from another type, which is also represented by a Type object
IsDerivedByExtension	boolean	True if this type is derived by extension
IsDerivedByRestriction	boolean	True if this type is derived by restriction
IsDerivedByUnion	boolean	True if this type is derived by union
IsDerivedByList	boolean	True if this type is derived by list
BaseType	Type	The base type of this type (if IsDerived is true)
IsDocumentRootType	boolean	True if this type represents the document itself
Library	Library [1640]	Library containing this type
IsFinal	boolean	True if declared as final in the schema
IsMixed	boolean	True if this type can have mixed content
IsAbstract	boolean	True if this type is declared as abstract
IsGlobal	boolean	True if this type is declared globally in the schema
IsAnonymous	boolean	True if this type is declared locally in an element

For simple types only:

Property	Type	Description
IsNativeBound	boolean	True if native type binding exists
NativeBinding	NativeBinding [1643]	Native binding for this type
Facets	Facets [1643]	Facets of this type

Property	Type	Description
Whitespace	string	Shortcut to the Whitespace facet

* Complex types with text content (these are types with mixed content and complexType with simpleContent) have an additional unnamed attribute member that represents the text content.

16.12.10.4 Member

This object represents an attribute or element in the XML Schema. It is used to create class members of types.

Property	Type	Description
CodeName	string	Name for generated code (derived from local name or parent declaration)
LocalName	string	Original name in the schema. Empty for the special member representing text content of complex types.
NamespaceURI	string	The namespace URI of this Element/Attribute within XML instance documents/streams.
DeclaringType	Type 1640	Type originally declaring the member (equal to ContainingType for non-inherited members)
ContainingType	Type 1640	Type where this is a member of
DataType	Type 1640	Data type of this member's content
Library	Library 1640	Library containing this member's DataType
IsAttribute	boolean	True for attributes, false for elements
IsOptional	boolean	True if minOccurs = 0 or optional attribute
IsRequired	boolean	True if minOccurs > 0 or required attribute
IsFixed	boolean	True for fixed attributes, value is in Default property
IsDefault	boolean	True for attributes with default value, value is in Default property
IsNillable	boolean	True for nillable elements
IsUseQualified	boolean	True if NamespaceURI is not empty
MinOccurs	integer	minOccurs, as in schema. 1 for required attributes
MaxOccurs	integer	maxOccurs, as in schema. 0 for prohibited attributes, -1 for unbounded

Property	Type	Description
Default	string	Default value

16.12.10.5 NativeBinding

This object represents the binding of a simple type to a native type in the target programming language, as specified by the "schemanativetype" map.

Property	Type	Description
ValueType	string	Native type
ValueHandler	string	Formatter class instance

16.12.10.6 Facets

This object represents all facets of a simple type. Inherited facets are merged with the explicitly declared facets. If a Length facet is in effect, MinLength and MaxLength are set to the same value.

Property	Type	Description
DeclaringType	Type	Type facets are declared on
Whitespace	string	"preserve", "collapse" or "replace"
MinLength	integer	Facet value
MaxLength	integer	Facet value
MinInclusive	integer	Facet value
MinExclusive	integer	Facet value
MaxInclusive	integer	Facet value
MaxExclusive	integer	Facet value
TotalDigits	integer	Facet value
FractionDigits	integer	Facet value
List	Facet collection	All facets as list

Facet

This object represents a single facet with its computed value effective for a specific type.

Property	Type	Description
LocalName	string	Facet name
NamespaceURI	string	Facet namespace
FacetType	string	one of "normalization", "lexicalspace", "valuespace-length", "valuespace-enum" or "valuespace-range"
DeclaringType	Type 1640	Type this facet is declared on
FacetCheckerName	string	Name of facet checker (from schemafacet map)
FacetValue	string or integer	Actual value of this facet

17 The MapForce API

The COM-based API of MapForce enables clients to access the functionality of MapForce from a custom code or application, and automate a wide range of tasks.

The MapForce COM API follows the common specifications for automation servers as set out by Microsoft. MapForce is automatically registered as a COM server object during installation. Once the COM server object is registered, you can invoke it from within applications and scripting languages that have programming support for COM calls. This makes it possible to access the MapForce API not only from development environments using .NET, C++ and Visual Basic, but also from scripting languages like JScript and VBScript.

Note the following:

- If you use the MapForce API to create an application that you intend to distribute to other clients, MapForce must be installed on each client computer. Also, your custom integration code must be deployed to (or your application installed on) each client computer.
- Certain API methods, such as `Document.GenerateOutput`, require that the MapForce main window is visible, or MapForce (running as a COM server) is embedded within a graphical user interface. If you need to run mappings in a fully unattended manner, across various platforms, consider using MapForce Server (https://www.altova.com/mapforce-server).

17.1 Accessing the API

To access the MapForce COM API, a new instance of the `Application` object must be created in your application (or script). Once this object is created, you can interact with MapForce by invoking its methods and properties as required (for example, create a new document, open an existing document, generate mapping code, etc).

Prerequisites

To make the MapForce COM object available in your Visual Studio project, add a reference to the MapForce type library (.tlb) file. The following instructions are applicable to Visual Studio 2013, but are similar in other Visual Studio versions:

1. On the **Project** menu, click **Add Reference**.
2. Click **Browse** and select the **MapForce.tlb** file located in the MapForce installation folder.

A sample MapForce API client in C# is available at: **C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\API\C#**.

In Java, the MapForce API is available through Java-COM bridge libraries. These libraries are available in the MapForce installation folder: **C:\Program Files (x86)\Altova\MapForce2023\JavaAPI** (note this path is valid when 32-bit MapForce runs on 64-bit Windows, otherwise adjust the path accordingly).

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers
- `AltovaAutomation.jar`: Java classes to access Altova automation servers
- `MapForceAPI.jar`: Java classes that wrap the MapForce automation interface
- `MapForceAPI_JavaDoc.zip`: a `Javadoc` file containing help documentation for the Java API

To allow access to the MapForce automation server directly from Java code, the libraries above must be in the Java `classpath`.

A sample MapForce API client in Java is available at: **C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\API\Java**.

In scripting languages such as JScript or VBScript, the MapForce COM object is accessible through the Microsoft Windows Script Host (see https://msdn.microsoft.com/en-us/library/9bbdkx3k.aspx). Such scripts can be written with a text editor, and do not need compilation, since they are executed by the Windows Script Host packaged with Windows. (To check that the Windows Script Host is running, type `wscript.exe /?` at the command prompt). Several JScript example files that call the MapForce API are available at: **C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\API\JScript**.

Note: For 32-bit MapForce, the registered name, or programmatic identifier (ProgId) of the COM object is `MapForce.Application`. For 64-bit MapForce, the name is `MapForce_x64.Application`. Be aware, though, that the calling program will access the CLASSES registry entries in its own registry hive, or group (32-bit or 64-bit). Therefore, if you run scripts using the standard command prompt and Windows Explorer on 64-bit Windows, the 64-bit registry entries will be accessed, which point to the 64-bit MapForce. For this reason, if both MapForce 32-bit and 64-bit are installed, special handling is required in order to call the 32-bit MapForce. For example, assuming that Windows Scripting Host is the calling program, do the following:

1. Change the current directory to **C:\Windows\SysWOW64**.

2. At the command line, type **wscript.exe** followed by the path to the script that you would like to run, for example:

```
wscript.exe "C:\Users\...
\Documents\Altova\MapForce2023\MapForceExamples\API\JScript\start.js"
```

Guidelines

The following guidelines should be considered in your client code:

- Do not hold references to objects in memory longer than you need them. If a user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Be aware that if your client code crashes, instances of MapForce may still remain in the system. For details on how to avoid error messages, see Error handling 1650.
- Free references explicitly, if using languages such as C++.

Creating the Application object

The syntax to create the starting `Application` object depends on the programming language, as shown in the examples below:

C#

```
// Create a new instance of MapForce via its automation interface.
MapForceLib.Application objMapForce = new MapForceLib.Application();
```

Java

```
// Start MapForce as COM server.
com.altova.automation.MapForce.Application objMapForce = new Application();
// COM servers start up invisible so we make it visible
objMapForce.setVisible(true);
```

JScript

```
// Access a running instance, or create a new instance of MapForce.
try
{
	objMapForce = WScript.GetObject ("", "MapForce.Application");
	// unhide application if it is a new instance
	objMapForce.Visible = true;
}
catch(err) { WScript.Echo ("Can't access or create MapForce.Application"); }
```

VBA

```
' Create a new instance of MapForce.
Dim objMapForce As Application
Set objMapForce = CreateObject("MapForce.Application")
```

VBScript

```
' Access a running instance, or create a new instance of MapForce.
Set objMapForce = GetObject("MapForce.Application");
```

Visual Basic

```
Dim objMapForce As MapForceLib.Application = New MapForceLib.Application
```


17.2 The Object Model

The starting point for every application which uses the MapForce API is the `Application` [1674] object. All other interfaces are accessed through the `Application` object as the starting point.

The object model of the MapForce API can be represented as follows (each indentation level indicates a child–parent relationship with the level directly above):

- `Application` [1674]
  - `Options` [1781]
  - `Project` [1790]
    - `ProjectItem` [1805]
  - `Documents` [1743]
    - `Document` [1725]
      - `MapForceView` [1762]
      - `Mapping` [1769]
        - `Component` [1706]
          - `Datapoint` [1723]
        - `Components` [1719]
        - `Connection` [1721]
      - `Mappings` [1779]
  - `ErrorMarkers` [1752]
    - `ErrorMarker` [1748]
- `AppOutputLines` [1701]
  - `AppOutputLine` [1690]
    - `AppOutputLines`
      - `...`
    - `AppOutputLineSymbol` [1703]

For information about creating an instance the `Application` object, see Accessing the API [1646]. For reference to the objects exposed by the API, see Object Reference [1674].

17.3 Error Handling

The MapForce API returns errors in two different ways. Every API method returns an HRESULT. This return value informs the caller about any malfunctions during the execution of the method. If the call was successful, the return value is equal to S_OK. C/C++ programmers generally use HRESULT to detect errors.

Visual Basic, scripting languages, and other high-level development environments do not give the programmer access to the returning HRESULT of a COM call. They use the second error-raising mechanism supported by the MapForce API, the IErrorInfo interface. If an error occurs, the API creates a new object that implements the IErrorInfo interface. The development environment takes this interface and fills its own error-handling mechanism with the provided information.

The following text describes how to deal with errors raised from the MapForce API in different development environments.

Visual Basic

A common way to handle errors in Visual Basic is to define an error handler. This error handler can be set with the On Error statement. Usually the handler displays an error message and does some cleanup to avoid spare references and any kind of resource leaks. Visual Basic fills its own Err object with the information from the IErrorInfo interface.

```
Sub Validate()
  'place variable declarations here

  'set error handler
  On Error GoTo ErrorHandler

  'if generation fails, program execution continues at ErrorHandler:
  objMapForce.ActiveDocument.GenerateXSLT()

  'additional code comes here

  'exit
  Exit Sub

  ErrorHandler:
  MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &
     "Description: " & Err.Description)
End Sub
```

JavaScript

The Microsoft implementation of JavaScript (JScript) provides a try-catch mechanism to deal with errors raised from COM calls. It is very similar to the VisualBasic approach, in that you also declare an error object containing the necessary information.

```
function Generate() {
    // please insert variable declarations here

    try {
        objMapForce.ActiveDocument.GenerateXSLT();
    }
    catch (Error) {
        sError = Error.description;
        nErrorCode = Error.number & 0xffff;
        return false;
    }

    return true;
}
```

C/C++

C/C++ gives you easy access to the `HRESULT` of the COM call and to the `IErrorInterface`.

```
   HRESULT hr;

   // Call GenerateXSLT() from the MapForce API
   if(FAILED(hr = ipDocument->GenerateXSLT()))
   {
      IErrorInfo *ipErrorInfo = Null;

      if(SUCCEEDED(::GetErrorInfo(0, &ipErrorInfo)))
      {
         BSTR   bstrDescr;
         ipErrorInfo->GetDescription(&bstrDescr);

         // handle Error information
         wprintf(L"Error message:\t%s\n",bstrDescr);
         ::SysFreeString(bstrDescr);

         // release Error info
         ipErrorInfo->Release();
      }
   }
```

17.4 Example C# Project

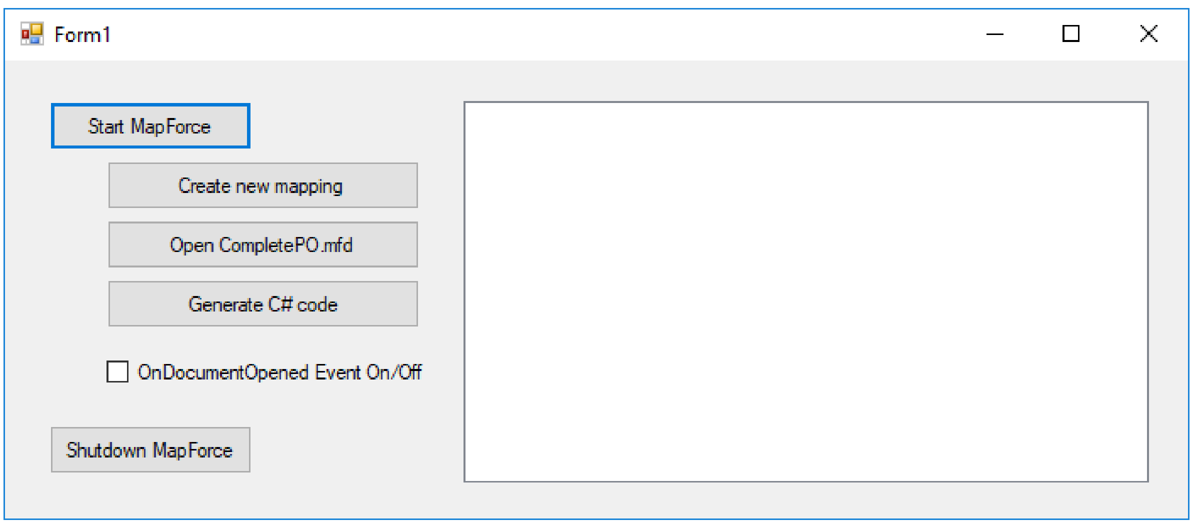
After you install MapForce, an example MapForce API client project for C# is available in the directory **C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\API**.

To compile and run the example, open the solution .sln file in Visual Studio and run **Debug | Start Debugging** or **Debug | Start Without Debugging**.

Note: If you have a 64-bit operating system and are using a 32-bit installation of MapForce, add the **x86** platform in the solution's Configuration Manager and build the sample using this configuration. A new x86 platform (for the active solution in Visual Studio) can be created in the New Solution Platform dialog (**Build | Configuration Manager | Active solution platform | <New…>**).

When you run the example, a Windows form is displayed, containing buttons that invoke basic MapForce operations:

- Start MapForce
- Create a new mapping design
- Open the CompletePO.mfd file from the **...\MapForceExamples** folder (note that you may need to adjust the path to point to the **\MapForceExamples** folder on your machine)
- Generate C# code in a temp directory
- Shut down MapForce



Code listing

The listing is commented for ease of understanding. The code essentially consists of a series of handlers for the buttons in the user interface shown above.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```



```
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // An instance of MapForce accessed via its automation interface.
        MapForceLib.Application MapForce;

        // Location of examples installed with MapForce
        String strExamplesFolder;

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        // handler for the "Start MapForce" button
        private void StartMapForce_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
            {
                Cursor.Current = Cursors.WaitCursor;

                // if we have no MapForce instance, we create one a nd make it visible.
                MapForce = new MapForceLib.Application();
                MapForce.Visible = true;

                // locate examples installed with MapForce.
                int majorVersionYear = MapForce.MajorVersion + 1998;
                strExamplesFolder = Environment.GetEnvironmentVariable("USERPROFILE") +
"\\My Documents\\Altova\\MapForce" + Convert.ToString(majorVersionYear) + "\
\MapForceExamples\\";

                Cursor.Current = Cursors.Default;
            }
            else
            {
                // if we have already an MapForce instance running we toggle its
visibility flag.
                MapForce.Visible = !MapForce.Visible;
            }
        }

        // handler for the "Open CompletePO.mfd" button
        private void openCompletePO_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
```

```
                StartMapForce_Click(null, null);

            // Open one of the sample files installed with the product.
            MapForce.OpenDocument(strExamplesFolder + "CompletePO.mfd");
        }

        // handler for the "Create new mapping" button
        private void newMapping_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
                StartMapForce_Click(null, null);

            // Create a new mapping
            MapForce.NewMapping();
        }

        // handler for the "Shutdown MapForce" button
        // shut-down application instance by explicitly releasing the COM object.
        private void shutdownMapForce_Click(object sender, EventArgs e)
        {
            if (MapForce != null)
            {
                // allow shut-down of MapForce by releasing UI
                MapForce.Visible = false;

                // explicitly release COM object
                try
                {
                    while
(System.Runtime.InteropServices.Marshal.ReleaseComObject(MapForce) > 0) ;
                }
                finally
                {
                    // avoid later access to this object.
                    MapForce = null;
                }
            }
        }

        // handler for button "Generate C# Code"
        private void generateCppCode_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
                listBoxMessages.Items.Add("start MapForce first.");
            // COM errors get returned to C# as exceptions. We use a try/catch block to
handle them.
            try
            {
                MapForceLib.Document doc = MapForce.ActiveDocument;

                listBoxMessages.Items.Add("Active document " + doc.Name);
                doc.GenerateCHashCode();

            }
```

```
            catch (Exception ex)
            {
                // The COM call was not successful.
                // Probably no application instance has been started or no document is
open.
                MessageBox.Show("COM error: " + ex.Message);
            }
        }

        delegate void addListBoxItem_delegate(string sText);
        // called from the UI thread
        private void addListBoxItem(string sText)
        {
            listBoxMessages.Items.Add(sText);
        }
        // wrapper method to allow to call UI controls methods from a worker thread
        void syncWithUIthread(Control ctrl, addListBoxItem_delegate methodToInvoke,
String sText)
        {
            // Control.Invoke: Executes on the UI thread, but calling thread waits for
completion before continuing.
            // Control.BeginInvoke: Executes on the UI thread, and calling thread doesn't
wait for completion.
            if (ctrl.InvokeRequired)
                ctrl.BeginInvoke(methodToInvoke, new Object[] { sText });
        }

        // event handler for OnDocumentOpened event
        private void handleOnDocumentOpened(MapForceLib.Document i_ipDocument)
        {
            String sText = "";

            if (i_ipDocument.Name.Length > 0)
                sText = "Document " + i_ipDocument.Name + " was opened!";
            else
                sText = "A new mapping was created.";

            // we need to synchronize the calling thread with the UI thread because
            // the COM events are triggered from a working thread
            addListBoxItem_delegate methodToInvoke = new
addListBoxItem_delegate(addListBoxItem);
            // call syncWithUIthread with the following arguments:
            // 1 - listBoxMessages - list box control to display messages from COM events
            // 2 - methodToInvoke  - a C# delegate which points to the method which will
be called from the UI thread
            // 3 - sText           - the text to be displayed in the list box
            syncWithUIthread(listBoxMessages, methodToInvoke, sText);
        }

        private void checkBoxEventOnOff_CheckedChanged(object sender, EventArgs e)
        {
            if (MapForce != null)
            {
```

```
                if (checkBoxEventOnOff.Checked)
                    MapForce.OnDocumentOpened += new
MapForceLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpened);
                else
                    MapForce.OnDocumentOpened -= new
MapForceLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpened);
            }
        }
    }
}
```

17.5 Example Java Project

After you install MapForce, an example MapForce API client project for Java is available in the directory **C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\API**.

You can test the Java example directly from the command line, using the batch file `BuildAndRun.bat`, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list

The Java examples folder contains all the files required to run the example project. These files are listed below:

`AltovaAutomation.dll`	Java-COM bridge: DLL part
`AltovaAutomation.jar`	Java-COM bridge: Java library part
`MapForceAPI.jar`	Java classes of the MapForce API
`RunMapForce.java`	Java example source code
`BuildAndRun.bat`	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
`.classpath`	Eclipse project helper file
`.project`	Eclipse project file
`MapForceAPI_JavaDoc.zip`	`Javadoc` file containing help documentation for the Java API

What the example does

The example starts up MapForce and performs a few operations, including opening and closing documents. When done, MapForce stays open. You must close it manually.

Running the example from the command line

To run the example from the command line, open a command prompt window, go to the Java folder of the API Examples folder (*see above for location*), and then type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a Java Development Kit (JDK) 7 or later installation on your computer.

Press the **Return** key. The Java source in `RunMapForce.java` will be compiled and then executed.

Loading the example in Eclipse

Open Eclipse and use the **File | Import... | General | Existing Projects into Workspace** command to add the Eclipse project file (`.project`) located in the Java folder of the API Examples folder (*see above for location*). The project `RunMapForce` will then appear in your Package Explorer or Navigator.

Select the project and then the command **Run as | Java Application** to execute the example.

Note: You can select a class name or method of the Java API and press F1 to get help for that class or method.

Java source code listing

The Java source code in the example file `RunMapForce.java` is listed below with comments.

```
// access general JAVA-COM bridge classes
import java.util.Iterator;

import com.altova.automation.libs.*;

// access XMLSpy Java-COM bridge
import com.altova.automation.MapForce.*;
import com.altova.automation.MapForce.Enums.ENUMProgrammingLanguage;

/**
 * A simple example that starts the COM server and performs a few operations on it.
 * Feel free to extend.
 */
public class RunMapForce
{
   public static void main(String[] args)
   {
      // an instance of the application.
      Application mapforce = null;

      // instead of COM error handling use Java exception mechanism.
      try
      {
         // Start MapForce as COM server.
         mapforce = new Application();
         // COM servers start up invisible so we make it visible
         mapforce.setVisible(true);

         // The following lines attach to the application events using a default
implementation
         // for the events and override one of its methods.
         // If you want to override all document events it is better to derive your
listener class
         // from DocumentEvents and implement all methods of this interface.
         mapforce.addListener(new ApplicationEventsDefaultHandler()
         {
            @Override
            public void onDocumentOpened(Document i_ipDoc) throws AutomationException
            {
               String name = i_ipDoc.getName();

               if (name.length() > 0)
                  System.out.println("Document " + name + " was opened.");
               else
                  System.out.println("A new mapping was created.");
            }
         });
```

```
         // Locate samples installed with the product.
         int majorVersionYear = mapforce.getMajorVersion() + 1998;
         String strExamplesFolder = System.getenv("USERPROFILE") + "\\Documents\\Altova\
\MapForce" + Integer.toString(majorVersionYear) + "\\MapForceExamples\\";
         // create a new MapForce mapping and generate c++ code
         Document newDoc = mapforce.newMapping();
         ErrorMarkers err1 = newDoc.generateCodeEx(ENUMProgrammingLanguage.eCpp);
         display(err1);
         // open CompletePO.mfd and generate c++ code
         Document doc = mapforce.openDocument(strExamplesFolder + "CompletePO.mfd");
         ErrorMarkers err2 = doc.generateCodeEx(ENUMProgrammingLanguage.eCpp);
         display(err2);

         doc.close();
         doc = null;

         System.out.println("Watch MapForce!");
      }
      catch (AutomationException e)
      {
         // e.printStackTrace();
      }
      finally
      {
         // Make sure that MapForce can shut down properly.
         if (mapforce != null)
            mapforce.dispose();

         // Since the COM server was made visible and still is visible, it will keep
running
         // and needs to be closed manually.
         System.out.println("Now close MapForce!");
      }
   }

   public static void display(ErrorMarkers err) throws AutomationException
   {
      Iterator<ErrorMarker> itr = err.iterator();

      if (err.getCount() == 0)
          System.out.print("Code generation completed successfully.\n");

      while (itr.hasNext())
      {
         String sError = "";
          Object element = itr.next();
          if (element instanceof ErrorMarker)
             sError = ((ErrorMarker)element).getText();
          System.out.print("Error text: " + sError + "\n");
      }
   }
}
```

17.6 JScript Examples

After you install MapForce, several JScript example files are available in the directory **C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples\API**.

The example files can be run in one of two ways:

- *From the command line:*
  Open a command prompt window and type the name of one of the example scripts (for example, `Start.js`). The Windows Scripting Host that is packaged with Windows will execute the script.

- *From Windows Explorer:*
  In Windows Explorer, browse for the JScript file and double-click it. The Windows Scripting Host that is packaged with Windows will execute the script. After the script is executed, the command console gets closed automatically.

The following example files are included:

`Start.js`	Start Mapforce registered as an automation server or connect to a running instance. You can also view this code listing in Start Application 1660.
`DocumentAccess.js`	Shows how to open, iterate and close documents. You can also view this code listing in Simple Document Access 1661.
`GenerateCode.js`	Shows how to invoke code generation using JScript. You can also view this code listing in Generate Code 1662.
`Readme.txt`	Provides basic help to run the scripts.

This documentation additionally includes a few extra JScript code listings:

- Example: Code Generation 1664
- Example: Mapping Execution 1666
- Example: Project Support 1669

17.6.1 Start Application

The JScript code listing below starts the application and shuts it down. If an instance of the application is already running, the running instance will be returned. To run the script, start it from a command prompt window or from Windows Explorer, see also Accessing the API 1646.

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.

try {   objMapForce = WScript.GetObject("", "MapForce.Application");   }
```

```
catch(err) {}

if( typeof( objMapForce ) == "undefined" )
{
   try   {   objMapForce = WScript.GetObject("", "MapForce_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create MapForce.Application" );
      WScript.Quit();
   }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objMapForce.Visible = true;

WScript.Echo(objMapForce.Edition + " has successfully started. ");

objMapForce.Visible = false; // will shutdown application if it has no more COM
connections
//objMapForce.Visible = true;   // will keep application running with UI visible
```

17.6.2 Simple Document Access

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and close documents.

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.
try {   objMapForce = WScript.GetObject("", "MapForce.Application");   }
catch(err) {}

if( typeof( objMapForce ) == "undefined" )
{
   try   {   objMapForce = WScript.GetObject("", "MapForce_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create MapForce.Application" );
      WScript.Quit();
   }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objMapForce.Visible = true;

// **************************** code snippet for "Simple Document Access"
**********************
```

```
// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major
release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
majorVersionYear = objMapForce.MajorVersion + 1998
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\Documents\
\Altova\\MapForce" + majorVersionYear + "\\MapForceExamples\\";

objMapForce.Documents.OpenDocument(strExampleFolder + "CompletePO.mfd");
objMapForce.Documents.OpenDocument(strExampleFolder + "Altova_Hierarchical_DB.mfd");

// **************************** code snippet for "Simple Document Access"
***********************

// **************************** code snippet for "Iteration"
************************************

// go through all open documents using a JScript Enumerator
for (var iterDocs = new Enumerator(objMapForce.Documents); !iterDocs.atEnd();
iterDocs.moveNext())
{
   objName = iterDocs.item().Name;
   WScript.Echo("Document name: " + objName);
}

// go through all open documents using index-based access to the document collection
for (i = objMapForce.Documents.Count; i > 0; i--)
   objMapForce.Documents.Item(i).Close();


// **************************** code snippet for "Iteration"
************************************

//objMapForce.Visible = false;      // will shutdown application if it has no more COM
connections
objMapForce.Visible = true;   // will keep application running with UI visible
```

The code listed above is available as a sample file (see JScript Examples[1660]). To run the script, start it from a command prompt window or from Windows Explorer.

17.6.3 Generate Code

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and generate C++ code.

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.
try {   objMapForce = WScript.GetObject("", "MapForce.Application");   }
catch(err) {}
```

```
if( typeof( objMapForce ) == "undefined" )
{
   try   {   objMapForce = WScript.GetObject("", "MapForce_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create MapForce.Application" );
      WScript.Quit();
   }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objMapForce.Visible = true;

// **************************** code snippet for "Simple Document Access"
************************

// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major
release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
majorVersionYear = objMapForce.MajorVersion + 1998
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\Documents\
\Altova\\MapForce" + majorVersionYear + "\\MapForceExamples\\";

objMapForce.Documents.OpenDocument(strExampleFolder + "CompletePO.mfd");
//objMapForce.Documents.OpenDocument(strExampleFolder + "Altova_Hierarchical_DB.mfd");
objMapForce.Documents.NewDocument();

// **************************** code snippet for "Simple Document Access"
************************

// **************************** code snippet for "Iteration"
************************************

objText = "";
// go through all open documents using a JScript Enumerator and generate c++ code
for (var iterDocs = new Enumerator(objMapForce.Documents); !iterDocs.atEnd();
iterDocs.moveNext())
{
   objText += "Generated c++ code result for document " + iterDocs.item().Name + " :\n";
   objErrorMarkers = iterDocs.item().generateCodeEx(1); // ENUMProgrammingLanguage.eCpp =
1

   bSuccess = true;
   for (var iterErrorMarkers = new
Enumerator(objErrorMarkers); !iterErrorMarkers.atEnd(); iterErrorMarkers.moveNext())
   {
      bSuccess = false;
      objText += "\t" + iterErrorMarkers.item().Text + "\n";
   }

   if (bSuccess)
       objText += "\tCode generation completed successfully.\n";
```

```
   objText += "\n";
}

WScript.Echo(objText);

// go through all open documents using index-based access to the document collection
for (i = objMapForce.Documents.Count; i > 0; i--)
   objMapForce.Documents.Item(i).Close();


// *************************** code snippet for "Iteration"
************************************

//objMapForce.Visible = false;      // will shutdown application if it has no more COM
connections
objMapForce.Visible = true;   // will keep application running with UI visible
```

The code listed above is available as a sample file (see JScript Examples 1660). To run the script, start it from a command prompt window or from Windows Explorer.

17.6.4 Generate Code (Alternative)

The following JScript example shows how to load an existing document and generate different kinds of mapping code for it.

```
// ------------------- begin JScript example ---------------------
// Generate Code for existing mapping.
// works with Windows scripting host.

// ----------------- helper function ------------------
function Exit(strErrorText)
{
   WScript.Echo(strErrorText);
   WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
   if (objErr != null)
      Exit ("ERROR: (" + (objErr.number & 0xffff) + ")" + objErr.description + " - " +
strText);
   else
      Exit ("ERROR: " + strText);
}
// ---------------------------------------------------

// ----------------- MAIN ------------------

// ----- create the Shell and FileSystemObject of the windows scripting
try
{
   objWshShell = WScript.CreateObject("WScript.Shell");
```

```
   objFSO = WScript.CreateObject("Scripting.FileSystemObject");
}
catch(err)
   { Exit("Can't create WScript.Shell object"); }

// ----- open MapForce or access running instance and make it visible
try
{
   objMapForce = WScript.GetObject ("", "MapForce.Application");
   objMapForce.Visible = true;      // remove this line to perform background processing
}
catch(err) { WScript.Echo ("Can't access or create MapForce.Application"); }

// ----- open an existing mapping. adapt this to your needs!
objMapForce.OpenDocument(objFSO.GetAbsolutePathName ("Test.mfd"));

// ----- access the mapping to have access to the code generation methods
var objDoc = objMapForce.ActiveDocument;

// ----- set the code generation output properties and call the code generation methods.
// ----- adapt the output directories to your needs
try
{
   // ----- code generation uses some of these options
   var objOptions = objMapForce.Options;

   // ----- generate XSLT -----
   objOptions.XSLTDefaultOutputDirectory = "C:\\test\\TestCOMServer\\XSLT";
   objDoc.GenerateXSLT();

   // ----- generate Java Code -----
   objOptions.CodeDefaultOutputDirectory = "C:\\test\\TestCOMServer\\Java";
   objDoc.GenerateJavaCode();

   // ----- generate CPP Code, use same cpp code options as the last time -----
   objOptions.CodeDefaultOutputDirectory = "C:\\test\\TestCOMServer\\CPP";
   objDoc.GenerateCppCode();

   // ----- generate C# Code, use options C# code options as the last time -----
   objOptions.CodeDefaultOutputDirectory = "C:\\test\\TestCOMServer\\CHash";
   objDoc.GenerateCHashCode();
}
catch (err)
   { ERROR ("while generating XSL or program code", err); }


// hide MapForce to allow it to shut down
objMapForce.Visible = false;

// -------------------- end example ---------------------
```

17.6.5 Run a Mapping

The following JScript example shows how to load an existing document with a simple mapping, access its components, set input- and output-instance file names and execute the mapping.

```
/*
   This sample file performs the following operations:

   Load existing MapForce mapping document.
   Find source and target component.
   Set input and output instance filenames.
   Execute the transformation.

   Works with Windows scripting host.
*/

// ---- general helpers ------------------------------

function Exit( message )
{
   WScript.Echo( message );
   WScript.Quit(-1);
}

function ERROR( message, err )
{
   if( err != null )
      Exit( "ERROR: (" + (err.number & 0xffff) + ") " + err.description + " - " + message
);
   else
      Exit( "ERROR: " + message );
}



// ---- MapForce constants -----------------------

var eComponentUsageKind_Unknown     = 0;
var eComponentUsageKind_Instance    = 1;
var eComponentUsageKind_Input       = 2;
var eComponentUsageKind_Output      = 3;



// ---- MapForce helpers -----------------------

// Searches in the specified mapping for a component by name and returns it.
// If not found, throws an error.
function FindComponent( mapping, component_name )
{
   var components = mapping.Components;
   for( var i = 0 ; i < components.Count ; ++i )
   {
```

```
      var component = components.Item( i + 1 );
      if( component.Name == component_name )
         return component;
   }
   throw new Error( "Cannot find component with name " + component_name );
}



// Browses components in a mapping and returns the first one found acting as
// source component (i.e. having connections on its right side).
function GetFirstSourceComponent( mapping )
{
   var components = mapping.Components;
   for( var i = 0 ; i < components.Count ; ++i )
   {
      var component = components.Item( i + 1 );
      if( component.UsageKind == eComponentUsageKind_Instance &&
         component.HasOutgoingConnections )
      {
         return component;
      }
   }
   throw new Error( "Cannot find a source component" );
}

// Browses components in a mapping and returns the first one found acting as
// target component (i.e. having connections on its left side).
function GetFirstTargetComponent( mapping )
{
   var components = mapping.Components;
   for( var i = 0 ; i < components.Count ; ++i )
   {
      var component = components.Item( i + 1 );
      if( component.UsageKind == eComponentUsageKind_Instance &&
         component.HasIncomingConnections )
      {
         return component;
      }
   }
   throw new Error( "Cannot find a target component" );
}



function IndentTextLines( s )
{
   return "\t" + s.replace( /\n/g, "\n\t" );
}

function GetAppoutputLineFullText( oAppoutputLine )
{
   var s = oAppoutputLine.GetLineText();
   var oAppoutputChildLines = oAppoutputLine.ChildLines;
```

```
   var i;

   for( i = 0 ; i < oAppoutputChildLines.Count ; ++i )
   {
      oAppoutputChildLine = oAppoutputChildLines.Item( i + 1 );
      sChilds = GetAppoutputLineFullText( oAppoutputChildLine );
      s += "\n" + IndentTextLines( sChilds );
   }

   return s;
}

// Create a nicely formatted string from AppOutputLines
function GetResultMessagesString( oAppoutputLines )
{
   var s1 = "Transformation result messages:\n";
   var oAppoutputLine;
   var i;

   for( i = 0 ; i < oAppoutputLines.Count ; ++i )
   {
      oAppoutputLine = oAppoutputLines.Item( i + 1 );
      s1 += GetAppoutputLineFullText( oAppoutputLine );
      s1 += "\n";
   }

   return s1;
}



// ---- MAIN -------------------------------------

var wshShell;
var fso;
var mapforce;

// create the Shell and FileSystemObject of the windows scripting system
try
{
   wshShell = WScript.CreateObject( "WScript.Shell" );
   fso = WScript.CreateObject( "Scripting.FileSystemObject" );
}
catch( err )
   { ERROR( "Can't create windows scripting objects", err ); }

// open MapForce or access currently running instance
try
{
   mapforce = WScript.GetObject( "", "MapForce.Application" );
}
catch( err )
   { ERROR( "Can't access or create MapForce.Application", err ); }
```

```
try
{
   // Make MapForce UI visible. This is an API requirement for output generation.
  mapforce.Visible = true;

   // open an existing mapping.
   // **** adjust the examples path to your needs ! **************
   var sMapForceExamplesPath = fso.BuildPath(
            wshShell.SpecialFolders( "MyDocuments" ),
            "Altova\\MapForce2023\\MapForceExamples" );
   var sDocFilename = fso.BuildPath( sMapForceExamplesPath, "PersonList.mfd" );
   var doc = mapforce.OpenDocument( sDocFilename );

   // Find existing components by name in the main mapping.
   // Note, the names of components may not be unique as a schema component's name
   // is derived from its schema file name.
   var source_component = FindComponent( doc.MainMapping, "Employees" );
   var target_component = FindComponent( doc.MainMapping, "PersonList" );
   // If you do not know the names of the components for some reason, you could
   // use the following functions instead of FindComponent.
   //var source_component = GetFirstSourceComponent( doc.MainMapping );
   //var target_component = GetFirstTargetComponent( doc.MainMapping );

   // specify the desired input and output files.
   source_component.InputInstanceFile = fso.BuildPath( sMapForceExamplesPath,
"Employees.xml" );
   target_component.OutputInstanceFile = fso.BuildPath( sMapForceExamplesPath,
"test_transformation_results.xml" );

   // Perform the transformation.
   // You can use doc.GenerateOutput() if you do not need result messages.
   // If you have a mapping with more than one target component and you want
   // to execute the transformation only for one specific target component,
   // call target_component.GenerateOutput() instead.
   var result_messages = doc.GenerateOutputEx();

   var summary_info =
         "Transformation performed from " + source_component.InputInstanceFile + "\n" +
         "to " + target_component.OutputInstanceFile + "\n\n" +
         GetResultMessagesString( result_messages );
   WScript.Echo( summary_info );
}
catch( err )
{
   ERROR( "Failure", err );
}
```

17.6.6 Project Tasks

The following JScript example shows how to use the MapForce API to automate tasks pertaining to MapForce projects. Before running the example, make sure to edit the variable strSamplePath so that it points to the following folder of your MapForce installation: **`C:\Users\<username>\Documents\Altova\MapForce2023\MapForceExamples`**.

To successfully run all operations in this example below, you will need the Enterprise edition of MapForce. If you have the Professional edition, comment out the lines that insert the WebService project.

```
// //////////// global variables /////////////////
var objMapForce = null;
var objWshShell = null;
var objFSO = null;

// !!! adapt the following path to your needs. !!!
var strSamplePath = "C:\\Users\\<username>\\Documents\\Altova\\MapForce2023\
\MapForceExamples\\";

// ///////////////////// Helpers //////////////////////////////

function Exit(strErrorText)
{
   WScript.Echo(strErrorText);
   WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
   if (objErr != null)
      Exit ("ERROR: (" + (objErr.number & 0xffff) + ")" + objErr.description + " - " +
strText);
   else
      Exit ("ERROR: " + strText);
}

function CreateGlobalObjects ()
{
   // the Shell and FileSystemObject of the windows scripting host often useful
   try
   {
      objWshShell = WScript.CreateObject("WScript.Shell");
      objFSO = WScript.CreateObject("Scripting.FileSystemObject");
   }
   catch(err)
      { Exit("Can't create WScript.Shell object"); }

   // create the MapForce connection
   // if there is a running instance of MapForce (that never had a connection) - use it
   // otherwise, we automatically create a new instance
   try
   {
      objMapForce = WScript.GetObject("", "MapForce.Application");
   }
   catch(err)
   {
      { Exit("Can't access or create MapForce.Application"); }
   }
}
```

```
// ---------------------------------------------------------
// print project tree items and their properties recursively.
// ---------------------------------------------------------
function PrintProjectTree( objProjectItemIter, strTab )
{
   while ( ! objProjectItemIter.atEnd() )
   {
      // get current project item
      objItem = objProjectItemIter.item();

      try
      {
         // ----- print common properties
         strGlobalText += strTab + "[" + objItem.Kind + "]" + objItem.Name + "\n";

         // ----- print code generation properties, if available
         try
         {
            if ( objItem.CodeGenSettings_UseDefault )
               strGlobalText += strTab + "  Use default code generation settings\n";
            else
               strGlobalText += strTab + "  code generation language is " +
                                                    objItem.CodeGenSettings_Language +
                                   " output folder is " +
objItem.CodeGenSettings_OutputFolder + "\n";
         }
         catch( err ) {}

         // ----- print WSDL settings, if available
         try
         {
            strGlobalText += strTab + "  WSDL File is " + objItem.WSDLFile +
                              " Qualified Name is " + objItem.QualifiedName + "\n";
         }
         catch( err ) {}
      }
      catch( ex )
         { strGlobalText += strTab + "[" + objItem.Kind + "]\n" }

      // ---- recurse
      PrintProjectTree( new Enumerator( objItem ), strTab + '   ' );

      objProjectItemIter.moveNext();
   }
}

// ---------------------------------------------------------
// Load example project installed with MapForce.
// ---------------------------------------------------------
function LoadSampleProject()
{
   // close open project
   objProject = objMapForce.ActiveProject;
   if ( objProject != null )
```

```
      objProject.Close();

   // open sample project and iterate through it.
   objProject = objMapForce.OpenProject(strSamplePath + "MapForceExamples.mfp");
   // dump properties of all project items
   strGlobalText = '';
   PrintProjectTree( new Enumerator (objProject), ' ' )
   WScript.Echo( strGlobalText );

   objProject.Close();
}

// ---------------------------------------------------------
// Create a new project with some folders, mappings and a
// Web service project.
// ---------------------------------------------------------
function CreateNewProject()
{
   try
   {
      // create new project and specify file to store it.
      objProject = objMapForce.NewProject(strSamplePath + "Sample.mfp");

      // create a simple folder structure
      objProject.CreateFolder( "New Folder 1");
      objFolder1 = objProject.Item(1);
      objFolder1.CreateFolder( "New Folder 2");
      objFolder2 = ( new Enumerator( objFolder1 ) ).item();   // an alternative to
Item(0)

      // add two different mappings to folder structure
      objFolder1.AddFile( strSamplePath + "DB_Altova_SQLXML.mfd");
      objMapForce.Documents.OpenDocument(strSamplePath + "InspectionReport.mfd");
      objFolder2.AddActiveFile();

      // override code generation settings for this folder
      objFolder2.CodeGenSettings_UseDefault = false;
      objFolder2.CodeGenSettings_OutputFolder = strSamplePath + "SampleOutput"
      objFolder2.CodeGenSettings_Language = 1;      //C++

      // insert Web service project based on a wsdl file from the installed examples
      objProject.InsertWebService( strSamplePath + "TimeService/TimeService.wsdl",
                                "{http://www.Nanonull.com/TimeService/}TimeService",
                         "TimeServiceSoap",
                         true );
      objProject.Save();
      if ( ! objProject.Saved )
         WScript.Echo("problem occurred when saving project");

      // dump project tree
      strGlobalText = '';
      PrintProjectTree( new Enumerator (objProject), ' ' )
      WScript.Echo( strGlobalText );
   }
   catch (err)
```

```
   { ERROR("while creating new project", err ); }
}

// ----------------------------------------------------------
// Generate code for a project's sub-tree. Mix default code
// generation parameters and overloaded parameters.
// ----------------------------------------------------------
function GenerateCodeForNewProject()
{
   // since the Web service project contains only initial mappings,
   // we generate code only for our custom folder.
   // code generation parameters from project are used for Folder1,
   // whereas Folder2 provides overwritten values.
   objFolder = objProject.Item(1);
   objFolder1.GenerateCode();
}

// ////////////////////// MAIN //////////////////////////////

CreateGlobalObjects();
objMapForce.Visible = true;

LoadSampleProject();
CreateNewProject();
GenerateCodeForNewProject();

// uncomment to shut down application when script ends
// objMapForce.Visible = false;
```

17.7 Object Reference

This section provides reference to the objects of the MapForce COM API. The objects are described in a generic manner, since the API may be used with virtually any language that supports calling a COM object. For language-specific examples, see:

- Example C# Project 1652
- Example Java Project 1657
- JScript Examples 1660

The API reference contains two main sections, each describing the interfaces and the enumeration types used in the API, respectively. The enumeration values contain both the string name and a numeric value. If your scripting environment does not support enumerations, use the number-values instead.

In .NET, for every interface of the MapForce COM automation interface, a .NET class exists with the same name. Also, COM types will be converted to the appropriate .NET type. For example, a type such as `Long` in the COM API would appear as `System.Int32` in .NET.

In Java, note the following syntax variations:

- **Classes and class names**. For every interface of the MapForce automation interface, a Java class exists with the name of the interface.
- **Method names**. Method names on the Java interface are the same as used on the COM interfaces, but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with get and set can be used. If a property does not support write-access, no setter method is available. For example, for the `Name` property of the `Document` interface, the Java methods `getName` and `setName` are available.
- **Enumerations**. For every enumeration defined in the automation interface, a Java enumeration is defined with the same name and values.
- **Events and event handlers**. For every interface in the automation interface that supports events, a Java interface with the same name plus 'Event' is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name of this Java class is the name of the event interface plus 'DefaultHandler'. For example:

```
Application // Java class to access the application
ApplicationEvents // Events interface for the application
ApplicationEventsDefaultHandler // Default handler for "ApplicationEvents"
```

17.7.1 Interfaces

17.7.1.1 Application

The `Application` interface is the interface to a MapForce application object. It represents the main access point for the MapForce application itself. This interface is the starting point to do any further operations with MapForce or to retrieve or create other MapForce related automation objects. For information about creating an instance the Application object, see Accessing the API 1646.

Properties to navigate the object model:

- `Application`
- `Parent`
- `Options`
- `Project`
- `Documents`

Application status:

- `Visible`
- `Name`
- `Quit`
- `Status`
- `WindowHandle`

MapForce designs:

- `NewDocument`
- `OpenDocument`
- `OpenURL`
- `ActiveDocument`

MapForce projects:

- `NewProject`
- `OpenProject`
- `ActiveProject`

MapForce code generation:

- `HighlightSerializedMarker`

Global resources:

- `GlobalResourceConfig`
- `GlobalResourceFile`

Version information:

- `Edition`
- `IsAPISupported`
- `MajorVersion`
- `MinorVersion`

Properties

Name	Description
`ActiveDocument` 1678	Read-only. Returns the automation object of the currently active document. This property returns the same as `Documents.ActiveDocument`.
`ActiveProject` 1679	Read-only. Returns the automation object of the currently active project.

Name	Description
Application 1679	Read-only. Retrieves the application's top-level object.
Documents 1679	Read-only. Returns a collection of all currently open documents.
Edition 1680	Read-only. Returns the edition of the application, e.g. "Altova MapForce Enterprise Edition" for the enterprise edition.
GlobalResourceConfig 1680	Gets or sets the name of the active global resource configuration file. By default, the file is called **GlobalResources.xml**. The configuration file can be renamed and saved to any location. You can therefore have multiple Global Resources XML files. However, only one of these Global Resources XML File can be active, per application, at one time, and only the definitions contained in this file will be available to the application.
GlobalResourceFile 1681	Gets or sets the global resource definition file. By default, the file is called GlobalResources.xml.
IsAPISupported 1681	Read-only. Returns true if the API is supported in this version of MapForce.
LibraryImports 1681	Read-only. Gets a collection of imported libraries. In the MapForce graphical user interface, these correspond to entries from the **Manage Libraries** window, added at application level.
MajorVersion 1682	Read-only. Gets the major version number of MapForce. The version is calculated starting from 1998, and is incremented by one every year. For example, the major version is "18" for the release 2016.
MinorVersion 1682	Read-only. The minor version number of the product, e.g. 2 for 2006 R2 SP1.
Name 1683	Read-only. The name of the application.
Options 1683	Read-only. This property gives access to options that configure the generation of code.
Parent 1683	Read-only. The parent object according to the object model.
ServicePackVersion 1684	Read-only.

Name	Description
	The service pack version number of the product, e.g. 1 for 2016 R2 SP1.
Status [1684]	Read-only. The status of the application. It is one of the values of the `ENUMApplicationStatus` enumeration.
Visible [1684]	**True** if MapForce is displayed on the screen (though it might be covered by other applications or be iconized). **False** if MapForce is hidden. The default value for MapForce when automatically started due to a request from the automation server `Application` is false. In all other cases, the property is initialized to **true**. An application instance that is visible is said to be controlled by the user (and possibly by clients connected via the automation interface). It will only shut down due to an explicit user request. To shut down an application instance, set its visibility to false and clear all references to this instance within your program. The application instance will shut down automatically when no further COM clients are holding references to it.
WindowHandle [1685]	Read-only. Retrieves the application's Window Handle.

Methods

Name	Description
HighlightSerializedMarker [1685]	Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document has not already been loaded, it will be loaded first. See `Document.GenerateCodeEx` for a method to retrieve a serialized marker.
NewDocument [1686]	Creates a new empty document. The newly opened document becomes the `ActiveDocument`. This method is a shortened form of `Documents.NewDocument`.
NewProject [1686]	Creates a new empty project. The current project is closed. The new project is accessible under `ActiveProject`.
NewWebServiceProject [1687]	Creates a new empty Web Service project. The new project is accessible under `ActiveProject`. This method is available in MapForce Enterprise Edition only.
OpenDocument [1687]	Loads a previously saved document file and continues working on it. The newly opened document becomes the `ActiveDocument`. This method is a shorter form of `Documents.OpenDocument`.

Name	Description
OpenProject 1688	Opens an existing Mapforce project (*.mfp). The current project is closed. The newly opened project is accessible under `ActiveProject`.
OpenURL 1689	Loads a previously saved document file from an URL location. Allows user name and password to be supplied.
Quit 1689	Disconnects from MapForce to allow the application to shutdown. Calling this method is optional since MapForce keeps track of all external COM connections and automatically recognizes a disconnection. For more information on automatic shutdown see the `Visible` property.

Events

Name	Description
OnDocumentOpened 1690	This event is triggered when an existing or new document is opened. The corresponding close event is `Document.OnDocumentClosed`.
OnProjectOpened 1690	This event is triggered when an existing or new project is loaded into the application. The corresponding close event is `Project.OnProjectClosed`.
OnShutdown 1690	This event is triggered when the application is shutting down.

17.7.1.1.1 Properties

17.7.1.1.1.1 ActiveDocument

Returns the automation object of the currently active document. This property returns the same as `Documents.ActiveDocument`.

Signature

```
ActiveDocument : Document
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.2 ActiveProject

Returns the automation object of the currently active project.

Signature

```
ActiveProject : Project
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.3 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.4 Documents

Returns a collection of all currently open documents.

Signature

```
Documents : Documents
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.5 Edition

Returns the edition of the application, e.g. "Altova MapForce Enterprise Edition" for the enterprise edition.

Signature

```
Edition : String
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.6 GlobalResourceConfig

Gets or sets the name of the active global resource configuration file. By default, the file is called **GlobalResources.xml**.

The configuration file can be renamed and saved to any location. You can therefore have multiple Global Resources XML files. However, only one of these Global Resources XML File can be active, per application, at one time, and only the definitions contained in this file will be available to the application.

Signature

```
GlobalResourceConfig : String
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.7 GlobalResourceFile

Gets or sets the global resource definition file. By default, the file is called GlobalResources.xml.

Signature

```
GlobalResourceFile : String
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.8 IsAPISupported

Returns true if the API is supported in this version of MapForce.

Signature

```
IsAPISupported : Boolean
```

Errors

Error code	Description
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.9 LibraryImports

Gets a collection of imported libraries. In the MapForce graphical user interface, these correspond to entries from the **Manage Libraries** window, added at application level.

Signature

```
LibraryImports : LibraryImports
```

Errors

Error code	Description
1200	The object is no longer valid.

Error code	Description
1201	Invalid address for the return parameter was specified.

17.7.1.1.1.10 MajorVersion

Gets the major version number of MapForce. The version is calculated starting from 1998, and is incremented by one every year. For example, the major version is "18" for the release 2016.

Signature

```
MajorVersion : Long
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.11 MinorVersion

The minor version number of the product, e.g. 2 for 2006 R2 SP1.

Signature

```
MinorVersion : Long
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.12 Name

The name of the application.

Signature

```
Name : String
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.13 Options

This property gives access to options that configure the generation of code.

Signature

```
Options : Options
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.14 Parent

The parent object according to the object model.

Signature

```
Parent : Application
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.15 ServicePackVersion

The service pack version number of the product, e.g. 1 for 2016 R2 SP1.

Signature

```
ServicePackVersion : Long
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.16 Status

The status of the application. It is one of the values of the `ENUMApplicationStatus` enumeration.

Signature

```
Status : ENUMApplicationStatus
```
1820

Errors

Error code	Description
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.17 Visible

True if MapForce is displayed on the screen (though it might be covered by other applications or be iconized).

False if MapForce is hidden. The default value for MapForce when automatically started due to a request from the automation server `Application` is false. In all other cases, the property is initialized to **true**.

An application instance that is visible is said to be controlled by the user (and possibly by clients connected via the automation interface). It will only shut down due to an explicit user request. To shut down an application instance, set its visibility to false and clear all references to this instance within your program. The application instance will shut down automatically when no further COM clients are holding references to it.

Signature

```
Visible : Boolean
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.1.18 WindowHandle

Retrieves the application's Window Handle.

Signature

```
WindowHandle : Long
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.2 Methods

17.7.1.1.2.1 HighlightSerializedMarker

Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document has not already been loaded, it will be loaded first. See `Document.GenerateCodeEx` for a method to retrieve a serialized marker.

Signature

```
HighlightSerializedMarker(in i_strSerializedMarker:String) -> Void
```

Parameters

Name	Type	Description
i_strSerializedMarker	**`String`**	The `ErrorMarker` object to highlight. Use `ErrorMaker.Serialized` to obtain this value.

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.
1007	The string passed in **i_strSerializedMarker** is not recognized as a serialized MapForce marker.
1008	The marker points to a location that is no longer valid.

17.7.1.1.2.2 NewDocument

Creates a new empty document. The newly opened document becomes the `ActiveDocument`. This method is a shortened form of `Documents.NewDocument`.

Signature

```
NewDocument() -> Document
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.2.3 NewProject

Creates a new empty project. The current project is closed. The new project is accessible under `ActiveProject`.

Signature

```
NewProject() -> Project
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.2.4 NewWebServiceProject

Creates a new empty Web Service project. The new project is accessible under `ActiveProject`. This method is available in MapForce Enterprise Edition only.

Signature

```
NewWebServiceProject() -> Project
```

Errors

Error code	Description
1004	Error creating new project.
1005	Wrong edition of MapForce.

17.7.1.1.2.5 OpenDocument

Loads a previously saved document file and continues working on it. The newly opened document becomes the `ActiveDocument`. This method is a shorter form of `Documents.OpenDocument`.

Signature

```
OpenDocument(in i_strFileName:String) -> Document
```

Parameters

Name	Type	Description
i_strFileName	`String`	The path of the document to open.

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.2.6 OpenProject

Opens an existing Mapforce project (*.mfp). The current project is closed. The newly opened project is accessible under `ActiveProject`.

Signature

```
OpenProject(in i_strFileName:String) -> Project
```

Parameters

Name	Type	Description
i_strFileName	`String`	The path of the project to open.

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.
1002	The supplied filename is not valid.

17.7.1.1.2.7 OpenURL

Loads a previously saved document file from an URL location. Allows user name and password to be supplied.

Signature

```
OpenURL(in strURL:String, in strUser:String, in strPassword:String) -> Void
```

Parameters

Name	Type	Description
strURL	`String`	The URL from which the document should be loaded.
strUser	`String`	The username required to access the URL.
strPassword	`String`	The password required to access the URL.

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.
1002	The supplied URL is not valid.
1006	Error while opening the URL file.

17.7.1.1.2.8 Quit

Disconnects from MapForce to allow the application to shutdown. Calling this method is optional since MapForce keeps track of all external COM connections and automatically recognizes a disconnection. For more information on automatic shutdown see the `Visible` property.

Signature

```
Quit() -> Void
```

Errors

Error code	Description
1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.7.1.1.3 Events

17.7.1.1.3.1 OnDocumentOpened

This event is triggered when an existing or new document is opened. The corresponding close event is `Document.OnDocumentClosed`.

Signature

```
OnDocumentOpened(in i_ipDocument:Document) : Void
```

17.7.1.1.3.2 OnProjectOpened

This event is triggered when an existing or new project is loaded into the application. The corresponding close event is `Project.OnProjectClosed`.

Signature

```
OnProjectOpened(in i_ipProject:Project) : Void
```

17.7.1.1.3.3 OnShutdown

This event is triggered when the application is shutting down.

Signature

```
OnShutdown : Void
```

17.7.1.2 AppOutputLine

Represents a message line. In contrast to `ErrorMarker`, its structure is more detailed and can contain a collection of child lines, therefore forming a tree of message lines.

Properties to navigate the object model:

- `Application`
- `Parent`

Line access:

- `GetLineSeverity`
- `GetLineSymbol`
- `GetLineText`
- `GetLineTextEx`
- `GetLineTextWithChildren`
- `GetLineTextWithChildrenEx`

A single `AppOutputLine` consists of one or more sub-lines. Sub-line access:

- `GetLineCount`

A sub-line consists of one or more cells. Cell access:

- `GetCellCountInLine`
- `GetCellIcon`
- `GetCellSymbol`
- `GetCellText`
- `GetCellTextDecoration`
- `GetIsCellText`

Below an `AppOutputLine` there can be zero, one, or more child lines which themselves are of type `AppOutputLine`, which thus form a tree structure.

Child lines access:

- `ChildLines`

Properties

Name	Description
`Application` 1692	Read-only. Retrieves the application's top-level object.
`ChildLines` 1693	Read-only. Returns a collection of the current line's direct child lines.
`Parent` 1693	Read-only. The parent object according to the object model.

Methods

Name	Description
`GetCellCountInLine` 1694	Gets the number of cells in the sub-line indicated by `nLine` in the current `AppOutputLine`.

Name	Description
GetCellIcon 1694	Gets the icon of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine
GetCellSymbol 1695	Gets the symbol of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine
GetCellText 1696	Gets the text of the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine
GetCellTextDecoration 1696	Gets the decoration of the text cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine. It can be one of the ENUMAppOutputLine_TextDecoration values.
GetIsCellText 1697	Returns true if the cell indicated by nCell in the current AppOutputLine's sub-line indicated by nLine is a text cell.
GetLineCount 1697	Gets the number of sub-lines the current line consists of.
GetLineSeverity 1698	Gets the severity of the line. It can be one of the ENUMAppOutputLine_Severity values.
GetLineSymbol 1698	Gets the symbol assigned to the whole line.
GetLineText 1699	Gets the contents of the line as text.
GetLineTextEx 1699	Gets the contents of the line as text using the specified part and line separators.
GetLineTextWithChildren 1700	Gets the contents of the line including all child and descendant lines as text.
GetLineTextWithChildrenEx 1700	Gets the contents of the line including all child and descendant lines as text using the specified part, line, tab and item separators.

17.7.1.2.1 Properties

17.7.1.2.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.1.2 ChildLines

Returns a collection of the current line's direct child lines.

Signature

```
ChildLines : AppOutputLines
```

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.1.3 Parent

The parent object according to the object model.

Signature

```
Parent : AppOutputLines
```

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2 Methods

17.7.1.2.2.1 GetCellCountInLine

Gets the number of cells in the sub-line indicated by `nLine` in the current `AppOutputLine`.

Signature

```
GetCellCountInLine(in nLine:Long) -> Long
```

Parameters

Name	Type	Description
nLine	**Long**	Specifies the zero-based index of the line.

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.2 GetCellIcon (obsolete)

Gets the icon of the cell indicated by `nCell` in the current `AppOutputLine`'s sub-line indicated by `nLine`

Signature

```
GetCellIcon(in nLine:Long, in nCell:Long) -> Long
```

Parameters

Name	Type	Description
nLine	**Long**	
nCell	**Long**	

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.3 GetCellSymbol

Gets the symbol of the cell indicated by `nCell` in the current `AppOutputLine`'s sub-line indicated by `nLine`

Signature

```
GetCellSymbol(in nLine:Long, in nCell:Long) -> AppOutputLineSymbol
```

Parameters

Name	Type	Description
nLine	**Long**	Specifies the zero-based index of the line.
nCell	**Long**	Specifies the zero-based index of the cell.

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.4 GetCellText

Gets the text of the cell indicated by `nCell` in the current `AppOutputLine`'s sub-line indicated by `nLine`

Signature

```
GetCellText(in nLine:Long, in nCell:Long) -> String
```

Parameters

Name	Type	Description
nLine	**Long**	Specifies the zero-based index of the line.
nCell	**Long**	Specifies the zero-based index of the cell.

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.5 GetCellTextDecoration

Gets the decoration of the text cell indicated by `nCell` in the current `AppOutputLine`'s sub-line indicated by `nLine`. It can be one of the `ENUMAppOutputLine_TextDecoration` values.

Signature

```
GetCellTextDecoration(in nLine:Long, in nCell:Long) -> Long
```

Parameters

Name	Type	Description
nLine	**Long**	Specifies the zero-based index of the line.
nCell	**Long**	Specifies the zero-based index of the cell.

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.6 GetIsCellText

Returns true if the cell indicated by `nCell` in the current `AppOutputLine`'s sub-line indicated by `nLine` is a text cell.

Signature

```
GetIsCellText(in nLine:Long, in nCell:Long) -> Boolean
```

Parameters

Name	Type	Description
nLine	**Long**	Specifies the zero-based index of the line.
nCell	**Long**	Specifies the zero-based index of the cell.

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.7 GetLineCount

Gets the number of sub-lines the current line consists of.

Signature

```
GetLineCount() -> Long
```

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.8 GetLineSeverity

Gets the severity of the line. It can be one of the `ENUMAppOutputLine_Severity` values.

Signature

```
GetLineSeverity() -> Long
```

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.9 GetLineSymbol

Gets the symbol assigned to the whole line.

Signature

```
GetLineSymbol() -> AppOutputLineSymbol
```

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.10 GetLineText

Gets the contents of the line as text.

Signature

```
GetLineText() -> String
```

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.11 GetLineTextEx

Gets the contents of the line as text using the specified part and line separators.

Signature

```
GetLineTextEx(in psTextPartSeperator:String, in psLineSeperator:String) -> String
```

Parameters

Name	Type	Description
psTextPartSeperator	`String`	
psLineSeperator	`String`	

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.12 GetLineTextWithChildren

Gets the contents of the line including all child and descendant lines as text.

Signature

```
GetLineTextWithChildren() -> String
```

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.2.2.13 GetLineTextWithChildrenEx

Gets the contents of the line including all child and descendant lines as text using the specified part, line, tab and item separators.

Signature

```
GetLineTextWithChildrenEx(in psPartSep:String, in psLineSep:String, in psTabSep:String,
in psItemSep:String) -> String
```

Parameters

Name	Type	Description
psPartSep	String	
psLineSep	String	
psTabSep	String	
psItemSep	String	

Errors

Error code	Description
4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

17.7.1.3 AppOutputLines

Represents a collection of `AppOutputLine` message lines.

Properties to navigate the object model:

- `Application`
- `Parent`

Iterating through the collection:

- `Count`
- `Item`

Properties

Name	Description
`Application` 1701	Read-only. Retrieves the application's top-level object.
`Count` 1702	Read-only. Retrieves the number of lines in the collection.
`Item` 1702	Read-only. Retrieves the line at index `n` from the collection. Indices start with 1.
`Parent` 1703	Read-only. The parent object according to the object model.

17.7.1.3.1 Properties

17.7.1.3.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
4000	The object is no longer valid.

Error code	Description
4001	Invalid address for the return parameter was specified.

17.7.1.3.1.2 Count

Retrieves the number of lines in the collection.

Signature

```
Count : Integer
```

Errors

Error code	Description
4000	The object is no longer valid.
4001	Invalid address for the return parameter was specified.

17.7.1.3.1.3 Item

Retrieves the line at index n from the collection. Indices start with 1.

Signature

```
Item(in n:Integer) : AppOutputLine
```

Errors

Error code	Description
4000	The object is no longer valid.
4001	Invalid address for the return parameter was specified.

17.7.1.3.1.4 Parent

The parent object according to the object model.

Signature

```
Parent : AppOutputLine
```

Errors

Error code	Description
4000	The object is no longer valid.
4001	Invalid address for the return parameter was specified.

17.7.1.4 AppOutputLineSymbol

An `AppOutputLineSymbol` represents a link in an `AppOutputLine` message line which can be clicked in the MapForce Messages window. It is applied to a cell of an `AppOutputLine` or to the whole line itself.

Properties to navigate the object model:

- `Application`
- `Parent`

Access to AppOutputLineSymbol methods:

- `GetSymbolHREF`
- `GetSymbolID`
- `IsSymbolHREF`

Properties

Name	Description
`Application` 1704	Read-only. Retrieves the application's top-level object.
`Parent` 1704	Read-only. The parent object according to the object model.

Methods

Name	Description
`GetSymbolHREF` 1705	If the symbol is of type URL, returns the URL as a string.

Name	Description
GetSymbolID [1705]	Gets the ID of the symbol.
IsSymbolHREF [1706]	Returns true if the symbol is of kind URL.

17.7.1.4.1 Properties

17.7.1.4.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

17.7.1.4.1.2 Parent

The parent object according to the object model.

Signature

```
Parent : Application
```

Errors

Error code	Description
4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

17.7.1.4.2 Methods

17.7.1.4.2.1 GetSymbolHREF

If the symbol is of type URL, returns the URL as a string.

Signature

```
GetSymbolHREF() -> String
```

Errors

Error code	Description
4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

17.7.1.4.2.2 GetSymbolID

Gets the ID of the symbol.

Signature

```
GetSymbolID() -> Long
```

Errors

Error code	Description
4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

17.7.1.4.2.3 IsSymbolHREF

Returns true if the symbol is of kind URL.

Signature

```
IsSymbolHREF() -> Boolean
```

Errors

Error code	Description
4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

17.7.1.5 Component

A `Component` represents a MapForce component.

To navigate the control, use the `Application` and `Parent` properties.

Component properties:

- `HasIncomingConnections`
- `HasOutgoingConnections`
- `CanChangeInputInstanceFile`
- `CanChangeOutputInstanceFile`
- `ComponentName`.
- `ID`
- `IsParameterInputRequired`
- `IsParameterSequence`
- `Name`
- `Preview`
- `Schema`
- `SubType`
- `Type`

Instance related properties:

- `InputInstanceFile`
- `OutputInstanceFile`

Datapoints:

- `GetRootDatapoint`

Execution:

- `GenerateOutput`

Properties

Name	Description
`Application` 1709	Read-only. Retrieves the application's top-level object.
`CanChangeInputInstanceFile` 1709	Read-only. Indicates if the input instance file name can be changed. Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true. If the component does not have a filename node, false is returned.
`CanChangeOutputInstanceFile` 1709	Read-only. Indicates if the output instance file name can be changed. Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true. If the component does not have a filename node, false is returned.
`ComponentName` 1710	Gets or sets the component's name.
`HasIncomingConnections` 1710	Read-only. Indicates if the component has any incoming connections (on its left side) not including the filename node. An incoming connection on the filename node does not have any effect on the returned value.
`HasOutgoingConnections` 1711	Read-only. Indicates if the component has any outgoing connections (on its right side).
`ID` 1711	Read-only. Retrieves the component ID.
`InputInstanceFile` 1712	Gets or sets the component's input instance file.
`IsParameterInputRequired` 1712	Gets or sets, if the input parameter component requires an ingoing connection on the function call component of the user-defined function this input parameter component is in. This property works only for components, which are input parameter components.
`IsParameterSequence` 1712	Gets or sets, if the input or output parameter component supports sequences. This property works only for components, which are input or output parameter components.
`Name` 1713	Read-only.

Name	Description
	Gets the component's name.
OutputInstanceFile [1713]	Gets or sets the component's output instance file. Trying to access the `OutputInstanceFile` of a component via the API does not return any data if the "File" connector of the component has been connected to another item in the mapping.
Parent [1714]	Read-only. The parent object according to the object model.
Preview [1714]	Gets or sets if the component is the current preview component. This property works only for components, which are target components in the document's main mapping. Only one target component in the main mapping can be the preview component at any time. When setting this property, it is only possible to set it to true. This then will also implicitly set the `Preview` property of all other components to false. If there is just a single target component in the main mapping, it is also the preview component.
Schema [1715]	Read-only. Retrieves the component's schema file name.
SubType [1715]	Read-only. Retrieves the component's subtype.
Type [1716]	Read-only. Retrieves the component's type.
UsageKind [1716]	Read-only. Retrieves the component's usage kind.

Methods

Name	Description
GenerateOutput [1716]	Generates the output file(s) defined in the mapping for the current component only, using a MapForce internal mapping language. The name(s) of the output file(s) are defined as property of the current component which is the output item in the mapping for this generation process.
GetRootDatapoint [1717]	Gets a root datapoint on the left (input) or right (output) side of a component. To access children and descendants, the `Datapoint` object provides further methods.

17.7.1.5.1 Properties

17.7.1.5.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.2 CanChangeInputInstanceFile

Indicates if the input instance file name can be changed.

Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true. If the component does not have a filename node, false is returned.

Signature

```
CanChangeInputInstanceFile : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.3 CanChangeOutputInstanceFile

Indicates if the output instance file name can be changed.

Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true.
If the component does not have a filename node, false is returned.

Signature

```
CanChangeOutputInstanceFile : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.4 ComponentName

Gets or sets the component's name.

Signature

```
ComponentName : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1246	The component does not support setting its name.
1247	Invalid component name.

17.7.1.5.1.5 HasIncomingConnections

Indicates if the component has any incoming connections (on its left side) not including the filename node. An incoming connection on the filename node does not have any effect on the returned value.

Signature

```
HasIncomingConnections : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.6 HasOutgoingConnections

Indicates if the component has any outgoing connections (on its right side).

Signature

```
HasOutgoingConnections : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.7 ID

Retrieves the component ID.

Signature

```
ID : Long
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.8 InputInstanceFile

Gets or sets the component's input instance file.

Signature

```
InputInstanceFile : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.9 IsParameterInputRequired

Gets or sets, if the input parameter component requires an ingoing connection on the function call component of the user-defined function this input parameter component is in. This property works only for components, which are input parameter components.

Signature

```
IsParameterInputRequired : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1232	This operation works only for an input parameter component.
1240	Changing the document not allowed. It is read-only.

17.7.1.5.1.10 IsParameterSequence

Gets or sets, if the input or output parameter component supports sequences. This property works only for components, which are input or output parameter components.

Signature

```
IsParameterSequence : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1233	This operation works only for an input or output parameter component.
1240	Changing the document not allowed. It is read-only.

17.7.1.5.1.11 Name

Gets the component's name.

Signature

```
Name : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.12 OutputInstanceFile

Gets or sets the component's output instance file.

Trying to access the `OutputInstanceFile` of a component via the API does not return any data if the "File" connector of the component has been connected to another item in the mapping.

Signature

```
OutputInstanceFile : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.13 Parent

The parent object according to the object model.

Signature

```
Parent : Mapping
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.14 Preview

Gets or sets if the component is the current preview component.

This property works only for components, which are target components in the document's main mapping. Only one target component in the main mapping can be the preview component at any time.

When setting this property, it is only possible to set it to true. This then will also implicitly set the `Preview` property of all other components to false.

If there is just a single target component in the main mapping, it is also the preview component.

Signature

```
Preview : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.

Error code	Description
1201	Invalid address for the return parameter was specified.
1234	Only a target component in the main mapping can be set as preview component.
1235	A component cannot be set as non-preview component. Set another component as preview component instead.

17.7.1.5.1.15 Schema

Retrieves the component's schema file name.

Signature

```
Schema : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.16 SubType

Retrieves the component's subtype.

Signature

```
SubType : ENUMComponentSubType
```
1822

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.1.17 Type

Retrieves the component's type.

Signature

```
Type : ENUMComponentType
```
1823

17.7.1.5.1.18 UsageKind

Retrieves the component's usage kind.

Signature

```
UsageKind : ENUMComponentUsageKind
```
1823

Errors

Error code	**Description**
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.5.2 Methods

17.7.1.5.2.1 GenerateOutput

Generates the output file(s) defined in the mapping for the current component only, using a MapForce internal mapping language. The name(s) of the output file(s) are defined as property of the current component which is the output item in the mapping for this generation process.

Signature

```
GenerateOutput(out pbError:Boolean) -> AppOutputLines
```

Parameters

Name	**Type**	**Description**
pbError	`Boolean`	This is an output-only parameter. You will receive a value only if the calling language supports output

Name	Type	Description
		parameters. If not, the value you pass here will remain unchanged when the function has finished.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1248	Generating output is only supported when the graphical user interface is visible.

17.7.1.5.2.2 GetRootDatapoint

Gets a root datapoint on the left (input) or right (output) side of a component. To access children and descendants, the `Datapoint` object provides further methods.

Signature

```
GetRootDatapoint(in side:ENUMComponentDatapointSide, in strNamespace:String, in
strLocalName:String, in strParameterName:String) -> Datapoint
```
1822

Parameters

Name	Type	Description
side	`ENUMComponentDatapointSide` 1822	The side parameter indicates if an input, or output, datapoint of a component is to be retrieved.
strNamespace	`String`	The specified namespace and local name, indicate the specific name of the node whose datapoint is to be retrieved. For components with structural information such as schema components, you will have to provide the namespace together with the local name, or you can just pass an empty string for the namespace. File-based components like the schema component contain a special node on their root, the

Name	Type	Description
		filename node. There, `GetRootDatapoint` can only find the filename node. You will have to pass namespace `"http://www.altova.com/mapforce"` and local name `"FileInstance"` to retrieve a datapoint of this node.
strLocalName	`String`	See above.
strParameterName	`String`	The specified parameter name should be an empty string unless the component in question is a function call component. Since a user-defined function might contain input or output parameters of the same structure, the function call component calling this user-defined function can have more than one root node with an identical namespace and local name. They will then differ only by their parameter names, which are in fact the names of the according parameter components in the user-defined function mapping itself. It is not mandatory to specify the parameter name, though. In that case, the method will return the first root datapoint matching the specified namespace and local name.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1248	Datapoint not found.

17.7.1.6 Components

Represents a collection of `Component` objects.

To navigate the object model, use the following properties:

- `Application`
- `Parent`

To iterate through the collection:

- `Count`
- `Item`

Properties

Name	Description
`Application` 1719	Read-only. Retrieves the application's top-level object.
`Count` 1720	Read-only. Retrieves the number of components in the collection.
`Item` 1720	Read-only. Retrieves the component at index `n` from the collection. Indices start with 1.
`Parent` 1721	Read-only. The parent object according to the object model.

17.7.1.6.1 Properties

17.7.1.6.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1200	The object is no longer valid.

Error code	Description
1201	Invalid address for the return parameter was specified.

17.7.1.6.1.2 Count

Retrieves the number of components in the collection.

Signature

```
Count : Integer
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.6.1.3 Item

Retrieves the component at index n from the collection. Indices start with 1.

Signature

```
Item(in n:Integer) : Component
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.6.1.4 Parent

The parent object according to the object model.

Signature

```
Parent : Mapping
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.7 Connection

A `Connection` object represents a connector between two components.

To navigate the object model, use the following properties:

- `Application`
- `Parent`

To get or set the connection's type, use `ConnectionType`.

Properties

Name	Description
`Application` 1722	Read-only. Retrieves the application's top-level object.
`ConnectionType` 1722	Gets or sets the connection's type.
`Parent` 1723	Read-only. The parent object according to the object model.

17.7.1.7.1 Properties

17.7.1.7.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
2100	The object is no longer valid.
2101	Invalid address for the return parameter was specified.

17.7.1.7.1.2 ConnectionType

Gets or sets the connection's type.

Signature

```
ConnectionType : ENUMConnectionType
```
1823

Errors

Error code	Description
2100	The object is no longer valid.
2101	Invalid address for the return parameter was specified.
2102	Changing the document is not allowed. It is read-only.
2103	Failed changing connection type.

17.7.1.7.1.3 Parent

The parent object according to the object model.

Signature

```
Parent : Mapping
```

Errors

Error code	Description
2100	The object is no longer valid.
2101	Invalid address for the return parameter was specified.

17.7.1.8 Datapoint

A `Datapoint` object represents an input or output icon of a component.

Properties

Name	Description
`Application` [1724]	Read-only. Retrieves the application's top-level object.
`Parent` [1724]	Read-only. The parent object according to the object model.

Methods

Name	Description
`GetChild` [1724]	Scans for a direct child datapoint of the current datapoint, by namespace and local name. If a schema component has elements that contain mixed content, each displays an additional child node, the so-called **text()** node. To retrieve a datapoint of a **text()** node, you will have to pass an empty string in `strNamespace` as well as `"#text"` in `strLocalName` and `eSearchDatapointElement` in `searchFlags`.

17.7.1.8.1 Properties

17.7.1.8.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
2000	The object is no longer valid.
2001	Invalid address for the return parameter was specified.

17.7.1.8.1.2 Parent

The parent object according to the object model.

Signature

```
Parent : Component
```

Errors

Error code	Description
2000	The object is no longer valid.
2001	Invalid address for the return parameter was specified.

17.7.1.8.2 Methods

17.7.1.8.2.1 GetChild

Scans for a direct child datapoint of the current datapoint, by namespace and local name.

If a schema component has elements that contain mixed content, each displays an additional child node, the so-called **text()** node. To retrieve a datapoint of a **text()** node, you will have to pass an empty string in `strNamespace` as well as `"#text"` in `strLocalName` and `eSearchDatapointElement` in `searchFlags`.

Signature

```
GetChild(in strNamespace:String, in strLocalName:String, in
searchFlags:ENUMSearchDatapointFlags 1826) -> Datapoint
```

Parameters

Name	Type	Description
strNamespace	`String`	The namespace of the direct child datapoint.
strLocalName	`String`	The name of the direct child datapoint.
searchFlags	`ENUMSearchDatapointFlags` 1826	Search flags can be passed as combination of values (combined using binary OR) of the `ENUMSearchDatapointFlags` enumeration.

Errors

Error code	Description
2000	The object is no longer valid.
2001	Invalid address for the return parameter was specified.
2002	Datapoint not found.

17.7.1.9 Document

A `Document` object represents a MapForce document (a loaded MFD file). A document contains a main mapping and zero or more local user-defined-function mappings.

To navigate the object model, use the following properties:

- `Application`
- `Parent`

For file handling, use:

- `Activate`
- `Close`
- `FullName`
- `Name`
- `Path`
- `Saved`

- `Save`
- `SaveAs`

For mapping handling, use:

- `MainMapping`
- `Mappings`
- `CreateUserDefinedFunction`

For component handling, use:

- `FindComponentByID`

For code generation, use:

- `OutputSettings_ApplicationName`
- `JavaSettings_BasePackageName`
- `GenerateCHashCode`
- `GenerateCodeEx`
- `GenerateCppCode`
- `GenerateJavaCode`
- `GenerateXQuery`
- `GenerateXSLT`
- `GenerateXSLT2`
- `GenerateXSLT3`
- `HighlightSerializedMarker`

For mapping execution, use:

- `GenerateOutput`
- `GenerateOutputEx`

View access:

- `MapForceView`

Obsolete:

- `OutputSettings_Encoding`

Properties

Name	Description
`Application` 1729	Read-only. Retrieves the application's top-level object.
`FullName` 1730	Path and name of the document file.
`JavaSettings_BasePackageName` 1730	Sets or retrieves the base package name used when generating Java code. In the MapForce graphical user interface, this setting is available in the **Mapping Settings** dialog box (right-click the mapping and select **Mapping Settings** from the context menu).

Name	Description
LibraryImports 1730	Read-only. Gets a collection of imported libraries. In the MapForce graphical user interface, these correspond to entries from the **Manage Libraries** window, added at document level.
MainMapping 1731	Read-only. Retrieves the main mapping of the document.
MapForceView 1731	Read-only. This property gives access to functionality specific to the MapForce view.
Mappings 1732	Read-only. Returns a collection of mappings contained in the document.
Name 1732	Read-only. Name of the document file without file path.
OutputSettings_ApplicationName 1732	Sets or retrieves the application name available in the **Mapping Settings** dialog box (To display this dialog box in MapForce, right-click the mapping and select **Mapping Settings** from the context menu).
OutputSettings_Encoding 1733	This property is no longer supported. Mapping output encoding settings do no longer exist. Components have individual output encoding settings.
Parent 1733	Read-only. The parent object according to the object model.
Path 1733	Read-only. Path of the document file without name.
Saved 1734	Read-only. **True** if the document was not modified since the last save operation, **false** otherwise.

Methods

Name	Description
Activate 1734	Makes this document the active document.
Close 1735	Closes the document without saving.
CreateUserDefinedFunction 1735	Creates a user defined function in the current document.
FindComponentByID 1736	Searches in the whole document, also all its mappings, for the component with the specified id.
GenerateCHashCode 1736	Generates C# code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code

Name	Description
	generation.
GenerateCodeEx 1737	Generates code that will perform the mapping. The parameter **i_nLanguage** specifies the target language. The method returns an object that can be used to enumerate all messages created by the code generator. These are the same messages that get displayed in the Messages window of MapForce.
GenerateCppCode 1737	Generates C++ code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.
GenerateJavaCode 1738	Generates Java code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.
GenerateOutput 1738	Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping. **Note:** This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.
GenerateOutputEx 1739	Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping. This method is identical to `GenerateOutput` except for its return value containing the resulting messages, warnings and errors arranged as trees of `AppOutputLines`. **Note:** This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.
GenerateXQuery 1739	Generates mapping code as XQuery. Uses the properties defined in `Application.Options` to configure code generation.
GenerateXSLT 1740	Generates mapping code as XSLT. Uses the properties defined in `Application.Options` to configure code generation.
GenerateXSLT2 1740	Generates mapping code as XSLT2. Uses the properties defined in `Application.Options` to configure code generation.
GenerateXSLT3 1741	Generates XSLT 3.0 mapping code. Uses the properties defined in `Application.Options` to configure code generation.
HighlightSerializedMarker 1741	Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document is not

Name	Description
	already loaded, it will be loaded first. See `GenerateCodeEx` for a method to retrieve a serialized marker.
`Save` 1742	Saves the document to the file defined by `Document.FullName`.
`SaveAs` 1742	Saves the document to the specified file name, and sets `Document.FullName` to this value if the save operation was successful.

Events

Name	Description
`OnDocumentClosed` 1743	This event is triggered when a document is closed. The document object passed into the event handler should not be accessed. The corresponding open event is `Application.OnDocumentOpened`.
`OnModifiedFlagChanged` 1743	This event is triggered when a document's modification status changes.

17.7.1.9.1 Properties

17.7.1.9.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.2 FullName

Path and name of the document file.

Signature

```
FullName : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.3 JavaSettings_BasePackageName

Sets or retrieves the base package name used when generating Java code. In the MapForce graphical user interface, this setting is available in the **Mapping Settings** dialog box (right-click the mapping and select **Mapping Settings** from the context menu).

Signature

```
JavaSettings_BasePackageName : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.4 LibraryImports

Gets a collection of imported libraries. In the MapForce graphical user interface, these correspond to entries from the **Manage Libraries** window, added at document level.

Signature

```
LibraryImports : LibraryImports
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.5 MainMapping

Retrieves the main mapping of the document.

Signature

```
MainMapping : Mapping
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.6 MapForceView

This property gives access to functionality specific to the MapForce view.

Signature

```
MapForceView : MapForceView
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.7 Mappings

Returns a collection of mappings contained in the document.

Signature

```
Mappings : Mappings
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.8 Name

Name of the document file without file path.

Signature

```
Name : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.9 OutputSettings_ApplicationName

Sets or retrieves the application name available in the **Mapping Settings** dialog box (To display this dialog box in MapForce, right-click the mapping and select **Mapping Settings** from the context menu).

Signature

```
OutputSettings_ApplicationName : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.10 OutputSettings_Encoding (obsolete)

This property is no longer supported. Mapping output encoding settings do no longer exist. Components have individual output encoding settings.

Signature

```
OutputSettings_Encoding : String
```

17.7.1.9.1.11 Parent

The parent object according to the object model.

Signature

```
Parent : Documents
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.12 Path

Path of the document file without name.

Signature

```
Path : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.1.13 Saved

True if the document was not modified since the last save operation, **false** otherwise.

Signature

```
Saved : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.2 Methods

17.7.1.9.2.1 Activate

Makes this document the active document.

Signature

```
Activate() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.

17.7.1.9.2.2 Close

Closes the document without saving.

Signature

```
Close() -> Void
```

Errors

Error code	**Description**
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.2.3 CreateUserDefinedFunction

Creates a user defined function in the current document.

Signature

```
CreateUserDefinedFunction(in strFunctionName:String, in strLibraryName:String, in
strSyntax:String, in strDetails:String, in bInlinedUse:Boolean) -> Mapping
```

Parameters

Name	**Type**	**Description**
strFunctionName	`String`	The name of the function.
strLibraryName	`String`	The name of the library to which this function belongs.
strSyntax	`String`	A string that describes the syntax of this function (this is for information purpose only).
strDetails	`String`	A description of this function.
bInlinedUse	`Boolean`	Boolean flag that specifies if the function has inlined use.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1208	Failed creating user-defined function.
1209	Changing the document not allowed. It is read-only.

17.7.1.9.2.4 FindComponentByID

Searches in the whole document, also all its mappings, for the component with the specified id.

Signature

```
FindComponentByID(in nID:Unsigned Long) -> Component
```

Parameters

Name	Type	Description
nID	Unsigned Long	The ID of the component to search for.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.2.5 GenerateCHashCode

Generates C# code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Signature

```
GenerateCHashCode() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1205	Error during code generation.

17.7.1.9.2.6 GenerateCodeEx

Generates code that will perform the mapping. The parameter **i_nLanguage** specifies the target language. The method returns an object that can be used to enumerate all messages created by the code generator. These are the same messages that get displayed in the Messages window of MapForce.

Signature

```
GenerateCodeEx(in i_nLanguage:ENUMProgrammingLanguage 1824) -> ErrorMarkers
```

Parameters

Name	Type	Description
i_nLanguage	ENUMProgrammingLanguage 1824	Specifies the target code generation language.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1205	Error during code generation.

17.7.1.9.2.7 GenerateCppCode

Generates C++ code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Signature

```
GenerateCppCode() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1205	Error during code generation.

17.7.1.9.2.8 GenerateJavaCode

Generates Java code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Signature

```
GenerateJavaCode() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1205	Error during code generation.

17.7.1.9.2.9 GenerateOutput

Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping.

Note: This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.

Signature

```
GenerateOutput() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1206	Error during execution of mapping algorithm.
1210	Generating output is only supported when the graphical user interface is visible.

17.7.1.9.2.10 GenerateOutputEx

Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping. This method is identical to `GenerateOutput` except for its return value containing the resulting messages, warnings and errors arranged as trees of `AppOutputLines`.

Note: This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.

Signature

```
GenerateOutputEx() -> AppOutputLines
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1206	Error during execution of mapping algorithm.
1210	Generating output is only supported when the graphical user interface is visible.

17.7.1.9.2.11 GenerateXQuery

Generates mapping code as XQuery. Uses the properties defined in `Application.Options` to configure code generation.

Signature

```
GenerateXQuery() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1204	Error during XSLT/XSLT2/XSLT3/XQuery code generation.

17.7.1.9.2.12 GenerateXSLT

Generates mapping code as XSLT. Uses the properties defined in `Application.Options` to configure code generation.

Signature

```
GenerateXSLT() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1204	Error during XSLT/XSLT2/XSLT3/XQuery code generation.

17.7.1.9.2.13 GenerateXSLT2

Generates mapping code as XSLT2. Uses the properties defined in `Application.Options` to configure code generation.

Signature

```
GenerateXSLT2() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

Error code	Description
1204	Error during XSLT/XSLT2/XSLT3/XQuery code generation.

17.7.1.9.2.14 GenerateXSLT3

Generates XSLT 3.0 mapping code. Uses the properties defined in `Application.Options` to configure code generation.

Signature

```
GenerateXSLT3() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1204	Error during XSLT/XSLT2/XSLT3/XQuery code generation.

17.7.1.9.2.15 HighlightSerializedMarker

Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document is not already loaded, it will be loaded first. See `GenerateCodeEx` for a method to retrieve a serialized marker.

Signature

```
HighlightSerializedMarker(in i_strSerializedMarker:String) -> Void
```

Parameters

Name	Type	Description
i_strSerializedMarker	`String`	The `ErrorMarker` object to highlight. Use `ErrorMaker.Serialized` to obtain this value.

Errors

Error code	Description
1000	The object is no longer valid.
1001	Invalid address for the return parameter was specified.
1007	The string passed in `i_strSerializedMarker` is not recognized a serialized MapForce marker.
1008	The marker points to a location that is no longer valid.

17.7.1.9.2.16 Save

Saves the document to the file defined by `Document.FullName`.

Signature

```
Save() -> Void
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.2.17 SaveAs

Saves the document to the specified file name, and sets `Document.FullName` to this value if the save operation was successful.

Signature

```
SaveAs(in i_strFileName:String) -> Void
```

Parameters

Name	Type	Description
i_strFileName	`String`	Specifies the path where to save the document.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.9.3 Events

17.7.1.9.3.1 OnDocumentClosed

This event is triggered when a document is closed. The document object passed into the event handler should not be accessed. The corresponding open event is `Application.OnDocumentOpened`.

Signature

```
OnDocumentClosed(in i_ipDocument:Document) : Void
```

17.7.1.9.3.2 OnModifiedFlagChanged

This event is triggered when a document's modification status changes.

Signature

```
OnModifiedFlagChanged(in i_bIsModified:Boolean) : Void
```

17.7.1.10 Documents

Represents a collection of `Document` objects.

Properties to navigate the object model:

- `Application`
- `Parent`

Open and create mappings:

- `OpenDocument`
- `NewDocument`

Iterating through the collection:

- `Count`

- `Item`
- `ActiveDocument`

Properties

Name	Description
`ActiveDocument` 1744	Read-only. Retrieves the active document. If no document is open, `null` is returned.
`Application` 1745	Read-only. Retrieves the application's top-level object.
`Count` 1745	Read-only. Retrieves the number of documents in the collection.
`Item` 1746	Read-only. Retrieves the document at index `n` from the collection. Indices start with 1.
`Parent` 1746	Read-only. The parent object according to the object model.

Methods

Name	Description
`NewDocument` 1747	Creates a new document, adds it to the end of the collection, and makes it the active document.
`OpenDocument` 1747	Opens an existing mapping document (*.mfd). Adds the newly opened document to the end of the collection and makes it the active document.

17.7.1.10.1 Properties

17.7.1.10.1.1 ActiveDocument

Retrieves the active document. If no document is open, `null` is returned.

Signature

```
ActiveDocument : Document
```

Errors

Error code	Description
1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

17.7.1.10.1.2 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

17.7.1.10.1.3 Count

Retrieves the number of documents in the collection.

Signature

```
Count : Integer
```

Errors

Error code	Description
1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

17.7.1.10.1.4 Item

Retrieves the document at index n from the collection. Indices start with 1.

Signature

```
Item(in n:Integer) : Document
```

Errors

Error code	Description
1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

17.7.1.10.1.5 Parent

The parent object according to the object model.

Signature

```
Parent : Application
```

Errors

Error code	Description
1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

17.7.1.10.2 Methods

17.7.1.10.2.1 NewDocument

Creates a new document, adds it to the end of the collection, and makes it the active document.

Signature

```
NewDocument() -> Document
```

Errors

Error code	Description
1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

17.7.1.10.2.2 OpenDocument

Opens an existing mapping document (*.mfd). Adds the newly opened document to the end of the collection and makes it the active document.

Signature

```
OpenDocument(in strPath:String) -> Document
```

Parameters

Name	Type	Description
strPath	`String`	The path of the mapping file.

Errors

Error code	Description
1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

17.7.1.11 ErrorMarker

Represents a simple message line. Unlike `AppOutputLine` objects, error markers do not have a hierarchical structure.

Properties to navigate the object model:

- `Application`
- `Parent`

Access to message information:

- `DocumentFileName`
- `ErrorLevel`
- `Highlight`
- `Serialization`
- `Text`

Properties

Name	Description
`Application` 1749	Read-only. Retrieves the application's top-level object.
`DocumentFileName` 1749	Read-only. Retrieves the name of the mapping file that the error marker is associated with.
`ErrorLevel` 1750	Read-only. Retrieves the severity of the error.
`Parent` 1750	Read-only. The parent object according to the object model.
`Serialization` 1750	Read-only. Serialize error marker into a string. Use this string in calls to `Application.HighlightSerializedMarker` or `Document.HighlightSerializedMarker` to highlight the marked item in the mapping. The string can be persisted and used in other instantiations of MapForce or its Control.
`Text` 1751	Read-only. Retrieves the message text.

Methods

Name	Description
`Highlight` 1751	Highlights the item that the error marker is associated with. If the corresponding document is not open, it will be opened.

17.7.1.11.1 Properties

17.7.1.11.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

17.7.1.11.1.2 DocumentFileName

Retrieves the name of the mapping file that the error marker is associated with.

Signature

```
DocumentFileName : String
```

Errors

Error code	Description
1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

17.7.1.11.1.3 ErrorLevel

Retrieves the severity of the error.

Signature

```
ErrorLevel : ENUMCodeGenErrorLevel
```

1822

Errors

Error code	Description
1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

17.7.1.11.1.4 Parent

The parent object according to the object model.

Signature

```
Parent : ErrorMarkers
```

Errors

Error code	Description
1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

17.7.1.11.1.5 Serialization

Serialize error marker into a string. Use this string in calls to `Application.HighlightSerializedMarker` or `Document.HighlightSerializedMarker` to highlight the marked item in the mapping. The string can be persisted and used in other instantiations of MapForce or its Control.

Signature

```
Serialization : String
```

Errors

Error code	Description
1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

17.7.1.11.1.6 Text

Retrieves the message text.

Signature

```
Text : String
```

Errors

Error code	Description
1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

17.7.1.11.2 Methods

17.7.1.11.2.1 Highlight

Highlights the item that the error marker is associated with. If the corresponding document is not open, it will be opened.

Signature

```
Highlight() -> Void
```

Errors

Error code	Description
1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.
1008	The marker points to a location that is no longer valid.

17.7.1.12 ErrorMarkers

Represents a collection of `ErrorMarker` objects.

Properties to navigate the object model:

- `Application`
- `Parent`

Iterating through the collection:

- `Count`
- `Item`

Properties

Name	Description
`Application` 1752	Read-only. Retrieves the application's top-level object.
`Count` 1753	Read-only. Retrieves the number of error markers in the collection.
`Item` 1753	Read-only. Retrieves the error marker at index `n` from the collection. Indices start with 1.
`Parent` 1754	Read-only. The parent object according to the object model.

17.7.1.12.1 Properties

17.7.1.12.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1800	The object is no longer valid.

Error code	Description
1801	Invalid address for the return parameter was specified.

17.7.1.12.1.2 Count

Retrieves the number of error markers in the collection.

Signature

```
Count : Integer
```

Errors

Error code	Description
1800	The object is no longer valid.
1801	Invalid address for the return parameter was specified.

17.7.1.12.1.3 Item

Retrieves the error marker at index n from the collection. Indices start with 1.

Signature

```
Item(in n:Integer) : ErrorMarker
```

Errors

Error code	Description
1800	The object is no longer valid.
1801	Invalid address for the return parameter was specified.

17.7.1.12.1.4 Parent

The parent object according to the object model.

Signature

```
Parent : Application
```

Errors

Error code	Description
1800	The object is no longer valid.
1801	Invalid address for the return parameter was specified.

17.7.1.13 LibraryImport

A `LibraryImport` represents an imported library file (an entry from the **Manage Libraries** window).

Properties

Name	Description
Application [1755]	Read-only. Retrieves the application's top level object.
Parent [1755]	Read-only. Retrieves the parent object, according to the object model.
Path [1756]	Read-only. Gets the path of the imported library.
SaveRelativePath [1756]	When you save the document, this property specifies whether the library path should be saved as absolute or relative. When **true**, the path of the library will be relative to the document. When **false**, the library path will be absolute. Do not rely on this property to determine whether the path is absolute or relative, since the path may have been changed (either from the user interface or via API) since the document was loaded from the .mfd file. If you set this property (either via API or via user interface), the "Manage Libraries" window does immediately show the correct path's state in the user interface. Internally, however, the `Path` of the `ImportedLibrary` object will not be changed until the document is saved.

Name	Description
	Libraries imported globally cannot be saved with a relative path. Only libraries imported at document level can.

17.7.1.13.1 Properties

17.7.1.13.1.1 Application

Retrieves the application's top level object.

Signature

```
Application : Application
```

Errors

Error code	Description
2500	The object is no longer valid.
2501	Invalid address for the return parameter was specified.

17.7.1.13.1.2 Parent

Retrieves the parent object, according to the object model.

Signature

```
Parent : LibraryImports
```

Errors

Error code	Description
2500	The object is no longer valid.
2501	Invalid address for the return parameter was specified.

17.7.1.13.1.3 Path

Gets the path of the imported library.

Signature

```
Path : String
```

Errors

Error code	Description
2500	The object is no longer valid.
2501	Invalid address for the return parameter was specified.

17.7.1.13.1.4 SaveRelativePath

When you save the document, this property specifies whether the library path should be saved as absolute or relative. When **true**, the path of the library will be relative to the document. When **false**, the library path will be absolute.

Do not rely on this property to determine whether the path is absolute or relative, since the path may have been changed (either from the user interface or via API) since the document was loaded from the .mfd file.

If you set this property (either via API or via user interface), the "Manage Libraries" window does immediately show the correct path's state in the user interface. Internally, however, the `Path` of the `ImportedLibrary` object will not be changed until the document is saved.

> Libraries imported globally cannot be saved with a relative path. Only libraries imported at document level can.

Signature

```
SaveRelativePath : Boolean
```

Errors

Error code	Description
2500	The object is no longer valid.
2501	Invalid address for the return parameter was specified.
2502	Global imports cannot be saved with a relative path.

17.7.1.14 LibraryImports

Represents a collection of imported libraries (`LibraryImport` objects). Use the `Application` and `Parent` properties to navigate the object model. Use the `Count` and `Item` properties to iterate through the collection. You can get this collection as follows:

- Locally (at document level), through the `Document.LibraryImports` property
- Globally (at application level), through the `Application.LibraryImports` property.

> If you get the `LibraryImports` collection from the application object, the `Parent` property of the collection will be null.

Properties

Name	Description
`Application` 1758	Read-only. Gets the application's top level object.
`Count` 1758	Read-only. Gets the count of `LibraryImport` objects in this collection.
`Item` 1759	Read-only. Retrieves a library entry at index *n* from this collection. The index is 1-based.
`Parent` 1759	Read-only. Gets the parent document for local library imports. If you get the `LibraryImports` collection from the application object, the `Parent` property of the collection will be null.

Methods

Name	Description
`Add` 1759	Adds a new library to this `LibraryImports` object. The new library will have the path supplied by the **i_strFileName** parameter.
`Find` 1760	Returns a library reference given the path to the library file.
`Remove` 1761	Removes a library reference from the **Manage Libraries** window.

17.7.1.14.1 Properties

17.7.1.14.1.1 Application

Gets the application's top level object.

Signature

```
Application : Application
```

Errors

Error code	Description
2400	The object is no longer valid.
2401	Invalid address for the return parameter was specified.

17.7.1.14.1.2 Count

Gets the count of `LibraryImport` objects in this collection.

Signature

```
Count : Integer
```

Errors

Error code	Description
2400	The object is no longer valid.
2401	Invalid address for the return parameter was specified.

17.7.1.14.1.3 Item

Retrieves a library entry at index *n* from this collection. The index is 1-based.

Signature

```
Item(in n:Integer) : LibraryImport
```

Errors

Error code	Description
2400	The object is no longer valid.
2401	Invalid address for the return parameter was specified.

17.7.1.14.1.4 Parent

Gets the parent document for local library imports. If you get the `LibraryImports` collection from the application object, the `Parent` property of the collection will be null.

Signature

```
Parent : Document
```

Errors

Error code	Description
2400	The object is no longer valid.
2401	Invalid address for the return parameter was specified.

17.7.1.14.2 Methods

17.7.1.14.2.1 Add

Adds a new library to this `LibraryImports` object. The new library will have the path supplied by the **i_strFileName** parameter.

Signature

```
Add(in i_strFileName:String) -> LibraryImport
```

Parameters

Name	Type	Description
i_strFileName	`String`	Specifies the path of the library file. This path can be either absolute or relative to the mapping, depending on the state in which it was passed to the object. When the document is saved, the path will be made relative if the `LibraryImport.SaveRelativePath` flag is **true**; otherwise, it will be made absolute.

Errors

Error code	Description
2400	The object is no longer valid.
2401	Invalid address for the return parameter was specified.
2402	Adding library file failed.

17.7.1.14.2.2 Find

Returns a library reference given the path to the library file.

Signature

```
Find(in i_strFileName:String) -> LibraryImport
```

Parameters

Name	Type	Description
i_strFileName	`String`	The path of the library file to search for. For locally imported libraries, you can specify either the absolute or the relative path to the library file (unlike the `Remove` method, which requires the exact path). For globally imported libraries, the path must always be absolute

Name	Type	Description
		(since globally imported libraries cannot have a relative path).

Errors

Error code	Description
2400	The object is no longer valid.
2401	Invalid address for the return parameter was specified.

17.7.1.14.2.3 Remove

Removes a library reference from the **Manage Libraries** window.

Signature

```
Remove(in i_strFileName:String) -> Void
```

Parameters

Name	Type	Description
i_strFileName	`String`	The path of the library file to remove. Note that the path must reflect exactly the current (most recent) state of the `LibraryImport` object. Remember that the path may be either relative or absolute, and it may have changed if you saved the document, depending on the `LibraryImport.SaveRelativePath` flag. Therefore, if the `LibraryImport` object currently contains a relative path, then you should supply a relative path as value of this parameter. Otherwise, the library is not found and the `Remove` method fails. The above applies only for locally imported libraries. For globally imported libraries, the path must always be absolute (since

Name	Type	Description
		globally imported libraries cannot have a relative path)

Errors

Error code	Description
2400	The object is no longer valid.

17.7.1.15 MapForceView

Represents the current view in the MapForce Mapping tab for a document. A document has exactly one `MapForceView` which displays the currently active mapping.

Properties to navigate the object model:

- `Application`
- `Parent`

View activation and view properties:

- `Active`
- `ShowItemTypes`
- `ShowLibraryInFunctionHeader`
- `HighlightMyConnections`
- `HighlightMyConnectionsRecursivly`

Mapping related properties:

- `ActiveMapping`
- `ActiveMappingName`

Adding items:

- `InsertWSDLCall`
- `InsertXMLFile`
- `InsertXMLSchema`
- `InsertXMLSchemaWithSample`

Properties

Name	Description
`Active` 1764	Use this property to query if the mapping view is the active view, or set this view to be the active one.

Name	Description
`ActiveMapping`[1764]	Gets or sets the currently active mapping in the document this `MapForceView` belongs to.
`ActiveMappingName`[1765]	Gets or sets the currently active mapping by name in the document this MapForceView belongs to.
`Application`[1765]	Read-only. Retrieves the application's top-level object.
`HighlightMyConnections`[1765]	This property defines whether connections from the selected item only should be highlighed.
`HighlightMyConnectionsRecursively`[1766]	This property defines if only the connections coming directly or indirectly from the selected item should be highlighted.
`Parent`[1766]	Read-only. The parent object according to the object model.
`ShowItemTypes`[1767]	This property defines if types of items should be shown in the mapping diagram.
`ShowLibraryInFunctionHeader`[1767]	This property defines whether the name of the function library should be part of function names.

Methods

Name	Description
`InsertWSDLCall`[1768]	Adds a new WSDL call component to the mapping.
`InsertXMLFile`[1768]	`MapForceView.InsertXMLFile` is obsolete. Use `Mapping.InsertXMLFile` instead.
`InsertXMLSchema`[1769]	`MapForceView.InsertXMLSchema` is obsolete. Use `Mapping.InsertXMLSchema` instead.
`InsertXMLSchemaWithSample`[1769]	`MapForceView.InsertXMLSchemaWithSample` is obsolete. Use `Mapping.InsertXMLFile` instead. Notice, `Mapping.InsertXMLFile` does not require a parameter for passing the root element. The root element is automatically set as the XML file's root element name.

17.7.1.15.1 Properties

17.7.1.15.1.1 Active

Use this property to query if the mapping view is the active view, or set this view to be the active one.

Signature

```
Active : Boolean
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.1.2 ActiveMapping

Gets or sets the currently active mapping in the document this `MapForceView` belongs to.

Signature

```
ActiveMapping : Mapping
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.1.3 ActiveMappingName

Gets or sets the currently active mapping by name in the document this MapForceView belongs to.

Signature

```
ActiveMappingName : String
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.1.4 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.1.5 HighlightMyConnections

This property defines whether connections from the selected item only should be highlighed.

Signature

```
HighlightMyConnections : Boolean
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.1.6 HighlightMyConnectionsRecursively

This property defines if only the connections coming directly or indirectly from the selected item should be highlighted.

Signature

```
HighlightMyConnectionsRecursively : Boolean
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.1.7 Parent

The parent object according to the object model.

Signature

```
Parent : Document
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.1.8 ShowItemTypes

This property defines if types of items should be shown in the mapping diagram.

Signature

```
ShowItemTypes : Boolean
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.1.9 ShowLibraryInFunctionHeader

This property defines whether the name of the function library should be part of function names.

Signature

```
ShowLibraryInFunctionHeader : Boolean
```

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.2 Methods

17.7.1.15.2.1 InsertWSDLCall

Adds a new WSDL call component to the mapping.

Signature

```
InsertWSDLCall(in i_strWSDLFileName:String) -> Void
```

Parameters

Name	Type	Description
i_strWSDLFileName	`String`	Specifies the path of the WSDL file to add to the mapping.

Errors

Error code	Description
1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.7.1.15.2.2 InsertXMLFile (obsolete)

`MapForceView.InsertXMLFile` is obsolete. Use `Mapping.InsertXMLFile` instead.

Signature

```
InsertXMLFile(in i_strFileName:String, in i_strXMLRootName:String) -> Void
```

Parameters

Name	Type	Description
i_strFileName	`String`	
i_strXMLRootName	`String`	

17.7.1.15.2.3 InsertXMLSchema (obsolete)

`MapForceView.InsertXMLSchema` is obsolete. Use `Mapping.InsertXMLSchema` instead.

Signature

```
InsertXMLSchema(in i_strSchemaFileName:String, in i_strXMLRootName:String) -> Void
```

Parameters

Name	Type	Description
i_strSchemaFileName	`String`	
i_strXMLRootName	`String`	

17.7.1.15.2.4 InsertXMLSchemaWithSample (obsolete)

`MapForceView.InsertXMLSchemaWithSample` is obsolete. Use `Mapping.InsertXMLFile` instead. Notice, `Mapping.InsertXMLFile` does not require a parameter for passing the root element. The root element is automatically set as the XML file's root element name.

Signature

```
InsertXMLSchemaWithSample(in i_strSchemaFileName:String, in i_strXMLExampleFile:String,
in i_strXMLRootName:String) -> Void
```

Parameters

Name	Type	Description
i_strSchemaFileName	`String`	
i_strXMLExampleFile	`String`	
i_strXMLRootName	`String`	

17.7.1.16 Mapping

A `Mapping` object represents a mapping in a document, either the main mapping, or a local user-defined-function mapping.

Properties to navigate the object model:

- `Application`
- `Parent`

Mapping properties:

- `IsMainMapping`
- `Name`

Components in the mapping:

- `Components`

Adding items:

- `CreateConnection`
- `InsertFunctionCall`
- `InsertXMLFile`
- `InsertXMLSchema`
- `InsertXMLSchemaInputParameter`
- `InsertXMLSchemaOutputParameter`

Properties

Name	Description
`Application` 1772	Read-only. Retrieves the application's top-level object.
`Components` 1772	Read-only. Returns a collection of all components in the current mapping.
`IsMainMapping` 1772	Read-only. Indicates if the current mapping is the main mapping of the document the mapping is in. **True** means it is the main mapping. **False** means it is a user-defined function (UDF).
`Name` 1773	Read-only. The name of the mapping or user defined-function (UDF).
`Parent` 1773	Read-only. The parent object according to the object model.

Methods

Name	Description
`CreateConnection` 1774	Creates a connection between the two supplied datapoints (`DatapointFrom` & `DatapointTo`). It will fail to do so if the `DatapointFrom` is not an output-side datapoint, the `DatapointTo` is not an input-side datapoint, or a connection between these two datapoints already exists.

Name	Description
InsertFunctionCall 1774	Inserts a function call component into the current mapping. The specified library and function names indicate the function or user-defined function to be called.
InsertXMLFile 1775	Adds a new XML schema component to the mapping. The component's internal structure is determined by the schema referenced in the specified XML file (`i_strFileName`) or, if the XML file does not reference a schema file, by the separately specified schema file (`i_strSchemaFileName`). If the XML file has a schema file reference, then the parameter `i_strSchemaFileName` is ignored. The root element of the XML file will be used in the component. The specified XML file is used as the input sample to evaluate the mapping.
InsertXMLSchema 1776	Adds a new XML schema component to the mapping. The component's internal structure is determined by the schema file specified in the first parameter. The second parameter defines the root element of this schema if there is more than one candidate. If the passed root element is an empty string and more candidates are available, a **Select Root Element** dialog box will pop up if MapForce is visible. If MapForce is invisible, no dialog box will pop up and only an error is returned. No XML input sample is assigned to this component.
InsertXMLSchemaInputParameter 1777	Inserts an XML schema input parameter component into the current mapping. The current mapping has to be a user-defined function. Trying to insert it (the schema input parameter) into the main mapping will fail.
InsertXMLSchemaOutputParameter 1778	Inserts an XML schema output parameter component into the current mapping. The current mapping has to be a user-defined function. Trying to insert it (the schema output paramter) into the main mapping will fail.

17.7.1.16.1 Properties

17.7.1.16.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.16.1.2 Components

Returns a collection of all components in the current mapping.

Signature

```
Components : Components
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.16.1.3 IsMainMapping

Indicates if the current mapping is the main mapping of the document the mapping is in.

True means it is the main mapping.
False means it is a user-defined function (UDF).

Signature

```
IsMainMapping : Boolean
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.16.1.4 Name

The name of the mapping or user defined-function (UDF).

Signature

```
Name : String
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.16.1.5 Parent

The parent object according to the object model.

Signature

```
Parent : Document
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.16.2 Methods

17.7.1.16.2.1 CreateConnection

Creates a connection between the two supplied datapoints (`DatapointFrom` & `DatapointTo`).

It will fail to do so if the `DatapointFrom` is not an output-side datapoint, the `DatapointTo` is not an input-side datapoint, or a connection between these two datapoints already exists.

Signature

```
CreateConnection(in DatapointFrom:Datapoint, in DatapointTo:Datapoint) -> Connection
```

Parameters

Name	Type	Description
DatapointFrom	**Datapoint**	The datapoint from which the connection is to be created.
DatapointTo	**Datapoint**	The destination datapoint.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1241	Failed creating the connection.

17.7.1.16.2.2 InsertFunctionCall

Inserts a function call component into the current mapping.

The specified library and function names indicate the function or user-defined function to be called.

Signature

```
InsertFunctionCall(in strFunctionName:String, in strLibraryName:String) -> Component
```

Parameters

Name	Type	Description
strFunctionName	`String`	The name of the function to be inserted.
strLibraryName	`String`	The library name of the function to be inserted

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1242	Failed creating function call component.

17.7.1.16.2.3 InsertXMLFile

Adds a new XML schema component to the mapping.

The component's internal structure is determined by the schema referenced in the specified XML file (`i_strFileName`) or, if the XML file does not reference a schema file, by the separately specified schema file (`i_strSchemaFileName`).

If the XML file has a schema file reference, then the parameter `i_strSchemaFileName` is ignored.

The root element of the XML file will be used in the component.

The specified XML file is used as the input sample to evaluate the mapping.

Signature

```
InsertXMLFile(in i_strFileName:String, in i_strSchemaFileName:String) -> Component
```

Parameters

Name	Type	Description
i_strFileName	`String`	The path of the instance XML file to add.

Name	Type	Description
i_strSchemaFileName	`String`	The path of the XML Schema Definition file to add.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1244	Failed creating component.

17.7.1.16.2.4 InsertXMLSchema

Adds a new XML schema component to the mapping.

The component's internal structure is determined by the schema file specified in the first parameter.

The second parameter defines the root element of this schema if there is more than one candidate.

If the passed root element is an empty string and more candidates are available, a **Select Root Element** dialog box will pop up if MapForce is visible. If MapForce is invisible, no dialog box will pop up and only an error is returned.

No XML input sample is assigned to this component.

Signature

```
InsertXMLSchema(in i_strSchemaFileName:String, in i_strXMLRootName:String) -> Component
```

Parameters

Name	Type	Description
i_strSchemaFileName	`String`	The path of the XML Schema Definition file to add.
i_strXMLRootName	`String`	The root element of the schema (applicable when the schema has more than one root element).

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1244	Failed creating component.

17.7.1.16.2.5 InsertXMLSchemaInputParameter

Inserts an XML schema input parameter component into the current mapping.

The current mapping has to be a user-defined function. Trying to insert it (the schema input parameter) into the main mapping will fail.

Signature

```
InsertXMLSchemaInputParameter(in strParamName:String, in strSchemaFileName:String, in strXMLRootElementName:String) -> Component
```

Parameters

Name	Type	Description
strParamName	`String`	The name of the input parameter component to create.
strSchemaFileName	`String`	The path of the XML Schema Definition file to add.
strXMLRootElementName	`String`	The root element of the schema (applicable when the schema has more than one root element). If the passed root element is an empty string and more candidates are available, a **Select Root Element** dialog will pop up if MapForce is visible. If MapForce is invisible, no dialog box will pop up and only an error is returned.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1243	Failed creating parameter component.
1245	This operation is not supported for the main mapping.

17.7.1.16.2.6 InsertXMLSchemaOutputParameter

Inserts an XML schema output parameter component into the current mapping.

The current mapping has to be a user-defined function. Trying to insert it (the schema output paramter) into the main mapping will fail.

Signature

```
InsertXMLSchemaOutputParameter(in strParamName:String, in strSchemaFileName:String, in
strXMLRootElementName:String) -> Component
```

Parameters

Name	Type	Description
strParamName	`String`	The name of the output parameter component to create.
strSchemaFileName	`String`	The path of the XML Schema Definition file to add.
strXMLRootElementName	`String`	The root element of the schema (applicable when the schema has more than one root element). If the passed root element is an empty string and more candidates are available, a **Select Root Element** dialog will pop up if MapForce is visible. If MapForce is invisible, no dialog box will pop up and only an error is returned.

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1243	Failed creating parameter component.
1245	This operation is not supported for the main mapping.

17.7.1.17 Mappings

Represents a collection of `Mapping` objects.

Properties to navigate the object model:

- `Application`
- `Parent`

Iterating through the collection:

- `Count`
- `Item`

Properties

Name	Description
`Application` 1780	Read-only. Retrieves the application's top-level object.
`Count` 1780	Read-only. Retrieves the number of mappings in the collection.
`Item` 1781	Read-only. Retrieves the mapping at index `n` from the collection. Indices start with 1.
`Parent` 1781	Read-only. The parent object according to the object model.

17.7.1.17.1 Properties

17.7.1.17.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.17.1.2 Count

Retrieves the number of mappings in the collection.

Signature

```
Count : Integer
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.17.1.3 Item

Retrieves the mapping at index `n` from the collection. Indices start with 1.

Signature

```
Item(in n:Integer) : Mapping
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.17.1.4 Parent

The parent object according to the object model.

Signature

```
Parent : Document
```

Errors

Error code	Description
1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.7.1.18 Options

This object gives access to all MapForce options available in the **Tools | Options** dialog.

Properties to navigate the object model:

- `Application`
- `Parent`

General options:

- `ShowLogoOnPrint`
- `ShowLogoOnStartup`
- `UseGradientBackground`

Options for code generation:

- `DefaultOutputEncoding`
- `DefaultOutputByteOrder`
- `DefaultOutputByteOrderMark`
- `XSLTDefaultOutputDirectory`
- `CodeDefaultOutputDirectory`
- `CPPSettings_DOMType`
- `CPPSettings_GenerateVC6ProjectFile`
- `CppSettings_GenerateVSProjectFile`
- `CPPSettings_LibraryType`
- `CPPSettings_UseMFC`
- `CSharpSettings_ProjectType`

Properties

Name	Description
`Application` 1783	Read-only. Retrieves the application's top-level object.
`CodeDefaultOutputDirectory` 1783	Specifies the target directory where files generated by `Document.GenerateCppCode`, `Document.GenerateJavaCode` and `Document.GenerateCHashCode` are placed.
`CPPSettings_DOMType` 1784	Specifies the DOM type used by `Document.GenerateCppCode`.
`CPPSettings_GenerateVC6ProjectFile` 1784	Specifies if VisualC++ 6.0 project files should be generated by `Document.GenerateCppCode`.
`CppSettings_GenerateVSProjectFile` 1785	Specifies the version of Visual Studio in which project files should be generated by `Document.GenerateCppCode`.
`CPPSettings_LibraryType` 1785	Specifies the library type used by `Document.GenerateCppCode`.
`CPPSettings_UseMFC` 1786	Specifies if MFC support should be used by C++ code generated by `Document.GenerateCppCode`.
`CSharpSettings_ProjectType` 1786	Specifies the type of C# project used by `Document.GenerateCHashCode`.
`DefaultOutputByteOrder` 1787	Byte order for the file encoding used for output files.
`DefaultOutputByteOrderMark` 1787	Indicates if a byte order mark (BOM), is to be included in the file encoding of output files.
`DefaultOutputEncoding` 1787	File encoding used for output files.
`GenerateWrapperClasses` 1788	Indicates if wrapper classes are also to be generated when generating code.
`JavaSettings_ApacheAxisVersion` 1788	This property is obsolete.
`Parent` 1788	Read-only.

Name	Description
	The parent object according to the object model.
ShowLogoOnPrint 1789	Show or hide the MapForce logo on printed outputs.
ShowLogoOnStartup 1789	Show or hide the MapForce logo on application startup.
UseGradientBackground 1790	Set or retrieve the background color mode for a mapping window.
XSLTDefaultOutputDirectory 1790	Specifies the target directory where files generated by `Document.GenerateXSLT` are placed.

17.7.1.18.1 Properties

17.7.1.18.1.1 Application

Retrieves the application's top-level object.

Signature

```
Application : Application
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.2 CodeDefaultOutputDirectory

Specifies the target directory where files generated by `Document.GenerateCppCode`, `Document.GenerateJavaCode` and `Document.GenerateCHashCode` are placed.

Signature

```
CodeDefaultOutputDirectory : String
```

Errors

Error code	Description
1400	The object is no longer valid.

Error code	Description
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.3 CPPSettings_DOMType

Specifies the DOM type used by `Document.GenerateCppCode`.

Signature

```
CPPSettings_DOMType : ENUMDOMType
```

1824

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.
1402	The parameter value is out of range.
1403	The parameter value is not available anymore.

17.7.1.18.1.4 CPPSettings_GenerateVC6ProjectFile (obsolete)

Specifies if VisualC++ 6.0 project files should be generated by `Document.GenerateCppCode`.

Signature

```
CPPSettings_GenerateVC6ProjectFile : Boolean
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.
1402	The parameter value is out of range.
1403	The parameter value is not available anymore.

17.7.1.18.1.5 CppSettings_GenerateVSProjectFile

Specifies the version of Visual Studio in which project files should be generated by `Document.GenerateCppCode`.

Signature

```
CppSettings_GenerateVSProjectFile : ENUMProjectType
```

1825

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.
1402	The parameter value is out of range.
1403	The parameter value is not available anymore.

17.7.1.18.1.6 CPPSettings_LibraryType

Specifies the library type used by `Document.GenerateCppCode`.

Signature

```
CPPSettings_LibraryType : ENUMLibType
```

1824

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.7 CPPSettings_UseMFC

Specifies if MFC support should be used by C++ code generated by `Document.GenerateCppCode`.

Signature

```
CPPSettings_UseMFC : Boolean
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.8 CSharpSettings_ProjectType

Specifies the type of C# project used by `Document.GenerateCHashCode`.

Signature

```
CSharpSettings_ProjectType : ENUMProjectType
```
1825

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.
1402	The parameter value is out of range.
1403	The parameter value is not available anymore.

17.7.1.18.1.9 DefaultOutputByteOrder

Byte order for the file encoding used for output files.

Signature

```
DefaultOutputByteOrder : String
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.10 DefaultOutputByteOrderMark

Indicates if a byte order mark (BOM), is to be included in the file encoding of output files.

Signature

```
DefaultOutputByteOrderMark : Boolean
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.11 DefaultOutputEncoding

File encoding used for output files.

Signature

```
DefaultOutputEncoding : String
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.12 GenerateWrapperClasses

Indicates if wrapper classes are also to be generated when generating code.

Signature

```
GenerateWrapperClasses : Boolean
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.13 JavaSettings_ApacheAxisVersion (obsolete)

This property is obsolete.

Signature

```
JavaSettings_ApacheAxisVersion : ENUMApacheAxisVersion
```
1819

17.7.1.18.1.14 Parent

The parent object according to the object model.

Signature

```
Parent : Application
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.15 ShowLogoOnPrint

Show or hide the MapForce logo on printed outputs.

Signature

```
ShowLogoOnPrint : Boolean
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.16 ShowLogoOnStartup

Show or hide the MapForce logo on application startup.

Signature

```
ShowLogoOnStartup : Boolean
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.17 UseGradientBackground

Set or retrieve the background color mode for a mapping window.

Signature

```
UseGradientBackground : Boolean
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.18.1.18 XSLTDefaultOutputDirectory

Specifies the target directory where files generated by `Document.GenerateXSLT` are placed.

Signature

```
XSLTDefaultOutputDirectory : String
```

Errors

Error code	Description
1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

17.7.1.19 Project

A `Project` object represents a project and its tree of project items in MapForce.

Properties to navigate the object model:

- `Application`
- `Parent`

File handling:

- `FullName`
- `Name`
- `Path`

- `Saved`
- `Save`
- `Close`

Project tree navigation:

- `Count`
- `Item`
- `_NewEnum`

Project tree manipulation:

- `AddActiveFile`
- `AddFile`
- `InsertWebService` (Enterprise edition only)
- `CreateFolder`

Code generation:

- `Output_Folder`
- `Output_Language`
- `Output_TextEncoding`
- `Java_BasePackageName`
- `GenerateCode`
- `GenerateCodeEx`
- `GenerateCodeIn`
- `GenerateCodeInEx`

For examples of how to use the properties and methods listed above, see Example: Project Tasks 1669. Note that, in order to perform operations that involve Web services, MapForce Enterprise edition is required.

Properties

Name	Description
`_NewEnum` 1794	Read-only. This property supports language-specific standard enumeration.
`Application` 1795	Read-only. Retrieves the top-level application object.
`Count` 1795	Read-only. Retrieves number of children of the project's root item. For examples, see `Item` or `_NewEnum`
`FullName` 1796	Path and name of the project file.
`Item` 1796	Read-only. Returns the child at *n* position of the project's root. The index is 1-based (the first index is 1). The largest valid index is `Count`. For an alternative, see `_NewEnum`.
`Java_BasePackageName` 1796	Sets or gets the base package name of the Java packages that will be generated. This property is used only when generating

Name	Description
	Java code.
Name[1797]	Read-only. Name of the project file without file path.
Output_Folder[1797]	Sets or gets the default output folder used with `GenerateCode` and `GenerateCodeIn`. Project items can overwrite this value in their `CodeGenSettings_OutputFolder` property, when `CodeGenSettings_UseDefault` is set to false.
Output_Language[1798]	Sets or gets the default language for code generation when using `GenerateCode`. Project items can overwrite this value in their `CodeGenSettings_OutputLanguage` property, when `CodeGenSettings_UseDefault` is set to false.
Output_TextEncoding[1798]	Sets or gets the text encoding used when generating XML-based code.
Parent[1798]	Read-only. The parent object according to the object model.
Path[1799]	Read-only. Path of the project file without name.
Saved[1799]	Read-only. **True** if the project was not modified since the last `Save` operation, **false** otherwise.

Methods

Name	Description
AddActiveFile[1800]	Adds the currently open document to the mapping folder of the project's root.
AddFile[1800]	Adds the specified document to the mapping folder of the project's root.
Close[1801]	Closes the project without saving.
CreateFolder[1801]	Creates a new folder as a child of the project's root item.
GenerateCode[1802]	Generates code for all project items of the project. The code language and output location is determined by properties of the project and project items.
GenerateCodeEx[1802]	Generates code for all project items of the project. The code language and output location are determined by properties of the project and project items. An object that can be used to iterate through all messages issued by the code generation process is

Name	Description
	returned. These messages are the same as those shown in the Messages window of MapForce.
`GenerateCodeIn` 1802	Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items.
`GenerateCodeInEx` 1803	Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items. Returns an object that can be used to iterate through all messages issued by the code generation process. These messages are the same as those shown in the **Messages** window of MapForce.
`InsertWebService` 1804	Inserts a new Web service project into the project's Web service folder. If **i_bGenerateMappings** is true, initial mapping documents for all ports get generated automatically.
`Save` 1805	Saves the project to the file defined by `FullName`.

Events

Name	Description
`OnProjectClosed` 1805	This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is `Application.OnProjectOpened`.

17.7.1.19.1 Properties

17.7.1.19.1.1 _NewEnum

This property supports language-specific standard enumeration.

Signature

```
_NewEnum : IUnknown
```

Errors

Error code	Description
1500	The object is no longer valid.

Examples

```
// ---------------------------------------------------------------
// JScript sample - enumeration of a project's project items.
function AllChildrenOfProjectRoot()
{
    objProject = objMapForce.ActiveProject;
    if ( objProject != null )
    {

for ( objProjectIter = new Enumerator(objProject); ! objProjectIter.atEnd(); objProjectIt
er.moveNext() )
        {
            objProjectItem = objProjectIter.item();

            // do something with project item here
        }
    }
}
```

```
// ---------------------------------------------------------------
// JScript sample - iterate all project items, depth first.
function IterateProjectItemsRec(objProjectItemIter)
{
    while ( ! objProjectItemIter.atEnd() )
    {
        objProjectItem = objProjectItemIter.item();
        // do something with project item here

        IterateProjectItemsRec( new Enumerator(objProjectItem) );

        objProjectItemIter.moveNext();
    }
```

```
}
function IterateAllProjectItems()
{
    objProject = objMapForce.ActiveProject;
    if ( objProject != null )
    {
        IterateProjectItemsRec( new Enumerator(objProject) );
    }
}
```

17.7.1.19.1.2 Application

Retrieves the top-level application object.

Signature

```
Application : Application
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

17.7.1.19.1.3 Count

Retrieves number of children of the project's root item. For examples, see `Item` or `_NewEnum`

Signature

```
Count : Integer
```

Errors

Error code	Description
1500	The object is no longer valid.

17.7.1.19.1.4 FullName

Path and name of the project file.

Signature

```
FullName : String
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

17.7.1.19.1.5 Item

Returns the child at *n* position of the project's root. The index is 1-based (the first index is 1). The largest valid index is `Count`. For an alternative, see `_NewEnum`.

Signature

```
Item(in n:Integer) : ProjectItem
```

Errors

Error code	Description
1500	The object is no longer valid.

Examples

```
// -----------------------------------------------------------
// JScript code snippet - enumerate children using Count and Item.
for( nItemIndex = 1; nItemIndex <= objProject.Count; nItemIndex++ )
{
    objProjectItem = objProject.Item(nItemIndex);
    // do something with project item here
}
```

17.7.1.19.1.6 Java_BasePackageName

Sets or gets the base package name of the Java packages that will be generated. This property is used only when generating Java code.

Signature

```
Java_BasePackageName : String
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid package name specified. Invalid address for the return parameter was specified.

17.7.1.19.1.7 Name

Name of the project file without file path.

Signature

```
Name : String
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

17.7.1.19.1.8 Output_Folder

Sets or gets the default output folder used with `GenerateCode` and `GenerateCodeIn`. Project items can overwrite this value in their `CodeGenSettings_OutputFolder` property, when `CodeGenSettings_UseDefault` is set to false.

Signature

```
Output_Folder : String
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid folder name specified. Invalid address for the return parameter was specified.

17.7.1.19.1.9 Output_Language

Sets or gets the default language for code generation when using `GenerateCode`. Project items can overwrite this value in their `CodeGenSettings_OutputLanguage` property, when `CodeGenSettings_UseDefault` is set to false.

Signature

```
Output_Language : ENUMProgrammingLanguage
```
1824

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid language specified. Invalid address for the return parameter was specified.

17.7.1.19.1.10 Output_TextEncoding

Sets or gets the text encoding used when generating XML-based code.

Signature

```
Output_TextEncoding : String
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid text encoding specified. Invalid address for the return parameter was specified.

17.7.1.19.1.11 Parent

The parent object according to the object model.

Signature

```
Parent : Application
```


Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

17.7.1.19.1.12 Path

Path of the project file without name.

Signature

```
Path : String
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

17.7.1.19.1.13 Saved

True if the project was not modified since the last `Save` operation, **false** otherwise.

Signature

```
Saved : Boolean
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

17.7.1.19.2 Methods

17.7.1.19.2.1 AddActiveFile

Adds the currently open document to the mapping folder of the project's root.

Signature

```
AddActiveFile() -> ProjectItem
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.
1503	No active document is available.
1504	Active documents needs to be given a path name before it can be added to the project.
1705	Mapping could not be assigned to project. Maybe it is already contained in the target folder.

17.7.1.19.2.2 AddFile

Adds the specified document to the mapping folder of the project's root.

Signature

```
AddFile(in i_strFileName:String) -> ProjectItem
```

Parameters

Name	Type	Description
i_strFileName	`String`	Specifies the path of the document to add.

Errors

Error code	Description
1500	The object is no longer valid.

Error code	Description
1501	Invalid address for the return parameter was specified.
1705	Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.

17.7.1.19.2.3 Close

Closes the project without saving.

Signature

```
Close() -> Void
```

Errors

Error code	Description
1500	The object is no longer valid.

17.7.1.19.2.4 CreateFolder

Creates a new folder as a child of the project's root item.

Signature

```
CreateFolder(in i_strFolderName:String) -> ProjectItem
```

Parameters

Name	Type	Description
i_strFolderName	`String`	The name of the folder to create.

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid folder name or invalid address for the return parameter was specified.

17.7.1.19.2.5 GenerateCode

Generates code for all project items of the project. The code language and output location is determined by properties of the project and project items.

Signature

```
GenerateCode() -> Void
```

Errors

Error code	Description
1500	The object is no longer valid.
1706	Error during code generation.

17.7.1.19.2.6 GenerateCodeEx

Generates code for all project items of the project. The code language and output location are determined by properties of the project and project items. An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the Messages window of MapForce.

Signature

```
GenerateCodeEx() -> ErrorMarkers
```

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.
1706	Error during code generation.

17.7.1.19.2.7 GenerateCodeIn

Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items.

Signature

```
GenerateCodeIn(in i_nLanguage:ENUMProgrammingLanguage) -> Void
```
1824

Parameters

Name	Type	Description
i_nLanguage	ENUMProgrammingLanguage [1824]	Specifies the programming language in which code should be generated.

Errors

Error code	Description
1500	The object is no longer valid.
1706	Error during code generation.

17.7.1.19.2.8 GenerateCodeInEx

Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items. Returns an object that can be used to iterate through all messages issued by the code generation process. These messages are the same as those shown in the **Messages** window of MapForce.

Signature

```
GenerateCodeInEx(in i_nLanguage:ENUMProgrammingLanguage) -> ErrorMarkers
```
1824

Parameters

Name	Type	Description
i_nLanguage	ENUMProgrammingLanguage [1824]	Specifies the programming language in which code should be generated.

Errors

Error code	Description
1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

Error code	Description
1706	Error during code generation.

17.7.1.19.2.9 InsertWebService

Inserts a new Web service project into the project's Web service folder. If **i_bGenerateMappings** is true, initial mapping documents for all ports get generated automatically.

Signature

```
InsertWebService(in i_strWSDLFile:String, in i_strService:String, in i_strPort:String, in
i_bGenerateMappings:Boolean) -> ProjectItem
```

Parameters

Name	Type	Description
i_strWSDLFile	`String`	Specifies the path of the WSDL file to add.
i_strService	`String`	Specifies the name of the Web service to add.
i_strPort	`String`	Specifies the port of the Web service to add.
i_bGenerateMappings	`Boolean`	If this parameter is true, initial mapping documents for all ports get generated automatically.

Errors

Error code	Description
1500	The object is no longer valid.
1501	WSDL file can not be found or is invalid. Service or port names are invalid. Invalid address for the return parameter was specified.
1503	Operation not supported by current edition.

17.7.1.19.2.10 Save

Saves the project to the file defined by `FullName`.

Signature

```
Save() -> Void
```

Errors

Error code	Description
1500	The object is no longer valid.
1502	Can't save to file.

17.7.1.19.3 Events

17.7.1.19.3.1 OnProjectClosed

This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is `Application.OnProjectOpened`.

Signature

```
OnProjectClosed(in i_ipProject:Project) : Void
```

17.7.1.20 ProjectItem

A `ProjectItem` object represents one item in a project tree.

Properties to navigate the object model:

- `Application`
- `Parent`

Project tree navigation:

- `Count`
- `Item`
- `_NewEnum`

Project item properties:

- `Kind`

- `Name`
- `WSDLFile` (only available to Web service project items)
- `QualifiedName` (only available to Web service project items)

Project tree manipulation:

- `AddActiveFile` (only available to folder items)
- `AddFile` (only available to folder items)
- `CreateFolder` (only available to folder items)
- `CreateMappingForProject` (only available to Web service operations)
- `Remove`

Document access:

- `Open` (only available to mapping items and Web service operations)

Code-generation:

- `CodeGenSettings_UseDefault`
- `CodeGenSettings_OutputFolder`
- `CodeGenSettings_Language`
- `GenerateCode`
- `GenerateCodeEx`
- `GenerateCodeIn`
- `GenerateCodeInEx`

For examples of how to use the properties and methods listed above, see Example: Project Tasks 1669. For operations with Web services, the MapForce Enterprise edition is required.

Properties

Name	Description
`_NewEnum` 1809	Read-only. This property supports language specific standard enumeration. For examples, see `Project.Item` or `Project._NewEnum`.
`Application` 1809	Read-only. Retrieves the top-level application object.
`CodeGenSettings_Language` 1810	Gets or sets the language to be used with `GenerateCode` or `Project.GenerateCode`. This property is consulted only if `CodeGenSettings_UseDefault` is set to false.
`CodeGenSettings_OutputFolder` 1810	Gets or sets the output directory to be used with `GenerateCode`, `GenerateCodeIn`, `Project.GenerateCode` or `Project.GenerateCodeIn`. This property is consulted only if `CodeGenSettings_UseDefault` is set to false.
`CodeGenSettings_UseDefault` 1810	Gets or sets whether output directory and code language are used as defined by either (a) the parent folders, or (b) the project root. This property is used with calls to `GenerateCode`, `GenerateCodeIn`, `Project.GenerateCode` and

Name	Description
	Project.GenerateCodeIn. If this property is set to false, the values of CodeGenSettings_OutputFolder and CodeGenSettings_Language are used to generate code for this project item.
Count 1811	Read-only. Retrieves the number of children of this project item. See also Item. For examples, see Project.Item or Project._NewEnum.
Item 1811	Read-only. Returns the child at n position of this project item. The index is 1-based (the first index is 1). The largest valid index is ProjectItem.Count. For an alternative, see ProjectItem._NewEnum. For examples, see Project.Item or Project._NewEnum.
Kind 1811	Read-only. Retrieves the kind of the project item. Availability of some properties and the applicability of certain methods is restricted to specific kinds of project items. The description of all methods and properties contains information about these restrictions.
Name 1812	Retrieves or sets the name of a project item. The name of most items is read-only. Exceptions are user-created folders, the names of which can be altered after creation.
Parent 1812	Read-only. Retrieves the project that this item is a child of. Has the same effect as Application.ActiveProject.
QualifiedName 1813	Read-only. Retrieves the qualified name of a Web service item.
WSDLFile 1813	Read-only. Retrieves the file name of the WSDL file defining the Web service that hosts the current project item.

Methods

Name	Description
AddActiveFile 1814	Adds the currently active document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.
AddFile 1814	Adds the specified document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.
CreateFolder 1815	Creates a new folder as a child of this project item.

Name	Description
CreateMappingForProject [1815]	Creates an initial mapping document for a Web service operation and saves it to **i_strFileName**. When using `Project.InsertWebService` you can set the **i_bGenerateMappings** flag to let MapForce automatically generate initial mappings for all ports.
GenerateCode [1816]	Generates code for this project item and its children. The code language and output location is determined by `CodeGenSettings_UseDefault`, `CodeGenSettings_Language` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.
GenerateCodeEx [1816]	Generates code for this project item and its children. The code language and output location are determined by `CodeGenSettings_UseDefault`, `CodeGenSettings_Language` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.
GenerateCodeIn [1817]	Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.
GenerateCodeInEx [1817]	Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation. An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the Messages window of MapForce.
Open [1818]	Opens the project item as a document or makes the corresponding document the active one, if it is already open. The project item must be a MapForce mapping or, for Enterprise edition only, Web service operation.
Remove [1819]	Remove this project item and all its children from the project tree.

Events

Name	Description
OnModifiedFlagChanged [1819]	Occurs when the `ProjectItem`'s modification status changes.

Name	Description
`OnProjectClosed` [1819]	This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is `Application.OnProjectOpened`.

17.7.1.20.1 Properties

17.7.1.20.1.1 _NewEnum

This property supports language specific standard enumeration. For examples, see `Project.Item` or `Project._NewEnum`.

Signature

```
_NewEnum : IUnknown
```

Errors

Error code	Description
1700	The object is no longer valid.

17.7.1.20.1.2 Application

Retrieves the top-level application object.

Signature

```
Application : Application
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

17.7.1.20.1.3 CodeGenSettings_Language

Gets or sets the language to be used with `GenerateCode` or `Project.GenerateCode`. This property is consulted only if `CodeGenSettings_UseDefault` is set to false.

Signature

```
CodeGenSettings_Language : ENUMProgrammingLanguage
```
1824

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid language or invalid address for the return parameter was specified.

17.7.1.20.1.4 CodeGenSettings_OutputFolder

Gets or sets the output directory to be used with `GenerateCode`, `GenerateCodeIn`, `Project.GenerateCode` or `Project.GenerateCodeIn`. This property is consulted only if `CodeGenSettings_UseDefault` is set to false.

Signature

```
CodeGenSettings_OutputFolder : String
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	An invalid output folder or an invalid address for the return parameter was specified.

17.7.1.20.1.5 CodeGenSettings_UseDefault

Gets or sets whether output directory and code language are used as defined by either (a) the parent folders, or (b) the project root. This property is used with calls to `GenerateCode`, `GenerateCodeIn`, `Project.GenerateCode` and `Project.GenerateCodeIn`. If this property is set to false, the values of `CodeGenSettings_OutputFolder` and `CodeGenSettings_Language` are used to generate code for this project item.

Signature

```
CodeGenSettings_UseDefault : Boolean
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

17.7.1.20.1.6 Count

Retrieves the number of children of this project item. See also `Item`. For examples, see `Project.Item` or `Project._NewEnum`.

Signature

```
Count : Integer
```

Errors

Error code	Description
1700	The object is no longer valid.

17.7.1.20.1.7 Item

Returns the child at `n` position of this project item. The index is 1-based (the first index is 1). The largest valid index is `ProjectItem.Count`. For an alternative, see `ProjectItem._NewEnum`. For examples, see `Project.Item` or `Project._NewEnum`.

Signature

```
Item(in n:Integer) : ProjectItem
```

Errors

Error code	Description
1700	The object is no longer valid.

17.7.1.20.1.8 Kind

Retrieves the kind of the project item. Availability of some properties and the applicability of certain methods is restricted to specific kinds of project items. The description of all methods and properties contains information about these restrictions.

Signature

```
Kind : ENUMProjectItemType
```
1825

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

17.7.1.20.1.9 Name

Retrieves or sets the name of a project item. The name of most items is read-only. Exceptions are user-created folders, the names of which can be altered after creation.

Signature

```
Name : String
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	Project item does not allow to alter its name.

17.7.1.20.1.10 Parent

Retrieves the project that this item is a child of. Has the same effect as `Application.ActiveProject`.

Signature

```
Parent : Project
```

Errors

Error code	Description
1700	The object is no longer valid.

Error code	Description
1701	Invalid address for the return parameter was specified.

17.7.1.20.1.11 QualifiedName

Retrieves the qualified name of a Web service item.

Signature

```
QualifiedName : String
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	The project item is not a part of a Web service.

17.7.1.20.1.12 WSDLFile

Retrieves the file name of the WSDL file defining the Web service that hosts the current project item.

Signature

```
WSDLFile : String
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	The project item is not a part of a Web service.

17.7.1.20.2 Methods

17.7.1.20.2.1 AddActiveFile

Adds the currently active document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.

Signature

```
AddActiveFile() -> ProjectItem
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	The file name is empty. Invalid address for the return parameter was specified.
1703	No active document is available.
1704	Active documents needs to be given a path name before it can be added to the project.
1705	Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.

17.7.1.20.2.2 AddFile

Adds the specified document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.

Signature

```
AddFile(in i_strFilePath:String) -> ProjectItem
```

Parameters

Name	Type	Description
i_strFilePath	`String`	The path of the document to add.

Errors

Error code	Description
1700	The object is no longer valid.

Error code	Description
1701	The file name is empty. Invalid address for the return parameter was specified.
1705	Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.

17.7.1.20.2.3 CreateFolder

Creates a new folder as a child of this project item.

Signature

```
CreateFolder(in i_strFolderName:String) -> ProjectItem
```

Parameters

Name	Type	Description
i_strFolderName	`String`	The name of the folder to create.

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid folder name or invalid address for the return parameter was specified.
1702	The project item does not support children.

17.7.1.20.2.4 CreateMappingForProject

Creates an initial mapping document for a Web service operation and saves it to **i_strFileName**. When using `Project.InsertWebService` you can set the **i_bGenerateMappings** flag to let MapForce automatically generate initial mappings for all ports.

Signature

```
CreateMappingForProject(in i_strFileName:String) -> ProjectItem
```

Parameters

Name	Type	Description
i_strFileName	`String`	Specifies the path where the mapping should be saved.

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1707	Cannot create new mapping. The project item does not support auto-creation of initial mappings or a mapping already exists.
1708	Operation not supported in current edition.

17.7.1.20.2.5 GenerateCode

Generates code for this project item and its children. The code language and output location is determined by `CodeGenSettings_UseDefault`, `CodeGenSettings_Language` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

Signature

```
GenerateCode() -> Void
```

Errors

Error code	Description
1700	The object is no longer valid.
1706	Error during code generation.

17.7.1.20.2.6 GenerateCodeEx

Generates code for this project item and its children. The code language and output location are determined by `CodeGenSettings_UseDefault`, `CodeGenSettings_Language` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

Signature

```
GenerateCodeEx() -> ErrorMarkers
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1706	Error during code generation.

17.7.1.20.2.7 GenerateCodeIn

Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

Signature

```
GenerateCodeIn(in i_nLanguage:ENUMProgrammingLanguage) -> Void
```
1824

Parameters

Name	Type	Description
i_nLanguage	`ENUMProgrammingLanguage` 1824	Specifies the programming language for code generation.

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid language specified.
1706	Error during code generation.

17.7.1.20.2.8 GenerateCodeInEx

Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the Messages window of MapForce.

Signature

```
GenerateCodeInEx(in i_nLanguage:ENUMProgrammingLanguage) -> ErrorMarkers
```
1824

Parameters

Name	Type	Description
i_nLanguage	ENUMProgrammingLanguage 1824	Specifies the programming language for code generation.

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid language specified or invalid address for the return parameter was specified.
1706	Error during code generation.

17.7.1.20.2.9 Open

Opens the project item as a document or makes the corresponding document the active one, if it is already open. The project item must be a MapForce mapping or, for Enterprise edition only, Web service operation.

Signature

```
Open() -> Document
```

Errors

Error code	Description
1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	The project item does not refer to a MapForce mapping file.
1708	Operation not supported in current edition.

17.7.1.20.2.10 Remove

Remove this project item and all its children from the project tree.

Signature

```
Remove() -> Void
```

Errors

Error code	Description
1700	The object is no longer valid.

17.7.1.20.3 Events

17.7.1.20.3.1 OnModifiedFlagChanged

Occurs when the `ProjectItem`'s modification status changes.

Signature

```
OnModifiedFlagChanged(in i_bIsModified:Boolean) : Void
```

17.7.1.20.3.2 OnProjectClosed

This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is `Application.OnProjectOpened`.

Signature

```
OnProjectClosed(in i_ipProject:Project) : Void
```

17.7.2 Enumerations

17.7.2.1 ENUMApacheAxisVersion (obsolete)

This enumeration type is obsolete.

Members

eApacheAxisVersion_Axis = 1

eApacheAxisVersion_Axis2 = 2

17.7.2.2 ENUMApplicationStatus

Enumeration values to indicate the status of the application.

Members

eApplicationRunning = 0

eApplicationAfterLicenseCheck = 1

eApplicationBeforeLicenseCheck = 2

eApplicationConcurrentLicenseCheckFailed = 3

eApplicationProcessingCommandLine = 4

17.7.2.3 ENUMAppOutputLine_Severity

Enumeration values to identify the severity of an `AppOutputLine`.

Members

eSeverity_Undefined = -1

eSeverity_Info = 0

eSeverity_Warning = 1

eSeverity_Error = 2

eSeverity_CriticalError = 3

eSeverity_Success = 4

eSeverity_Summary = 5

eSeverity_Progress = 6

eSeverity_DataEdit = 7

eSeverity_ParserInfo = 8

eSeverity_PossibleInconsistencyWarning = 9

eSeverity_Message = 10

eSeverity_Document = 11

eSeverity_Rest = 12

eSeverity_NoSelect = 13

eSeverity_Select = 14

eSeverity_Autoinsertion = 15

eSeverity_GlobalResources_DefaultWarning = 16

eSeverity_XPath_Styles_Changed = 17

eSeverity_XPath_Styles_Unchanged = 18

eSeverity_XPath_Styles_Skipped = 19

eSeverity_XPath_ComboBox_Values_Changed = 20

eSeverity_XPath_ComboBox_Values_Unchanged = 21

eSeverity_XPath_ComboBox_Values_Skipped = 22

eSeverity_XPath_Assertions_Changed = 23

eSeverity_XPath_Assertions_Unchanged = 24

eSeverity_XPath_Assertions_Skipped = 25

17.7.2.4 ENUMAppOutputLine_TextDecoration

Enumeration values for the different kinds of text decoration of an `AppOutputLine`.

Members

eTextDecorationDefault = 0

eTextDecorationBold = 1

eTextDecorationDebugValues = 2

eTextDecorationDB_ObjectName = 3

eTextDecorationDB_ObjectLink = 4

eTextDecorationDB_ObjectKind = 5

eTextDecorationDB_TimeoutValue = 6

eTextDecorationFind_MatchingString = 7

eTextDecorationValidation_Speclink = 8

eTextDecorationValidation_ErrorPosition = 9 eTextDecorationValidation_UnkownParam = 10

17.7.2.5 ENUMCodeGenErrorLevel

Enumeration values to identify severity of code generation messages.

Members

eCodeGenErrorLevel_Information = 0

eCodeGenErrorLevel_Warning = 1

eCodeGenErrorLevel_Error = 2

eCodeGenErrorLevel_Undefined = 3

17.7.2.6 ENUMComponentDatapointSide

Enumeration values to indicate the side of a datapoint on its component. See also `Component.GetRootDatapoint`.

Members

eDatapointSideInput = 0

eDatapointSideOutput = 1

17.7.2.7 ENUMComponentSubType

Enumeration values to indicate component sub types.

Members

eComponentSubType_None = 0

eComponentSubType_Text_EDI = 1

eComponentSubType_Text_Flex = 2

eComponentSubType_Text_CSVFLF = 3

17.7.2.8 ENUMComponentType

Enumeration values to indicate component types.

Members

eComponentType_Unknown = 0

eComponentType_XML = 1

eComponentType_DB = 2

eComponentType_Text = 3

eComponentType_Excel = 4

eComponentType_WSDL = 5

eComponentType_XBRL = 6

eComponentType_Input = 7

eComponentType_JSON = 8

17.7.2.9 ENUMComponentUsageKind

Enumeration values to indicate component usage kind.

Members

eComponentUsageKind_Unknown = 0

eComponentUsageKind_Instance = 1

eComponentUsageKind_Input = 2

eComponentUsageKind_Output = 3

eComponentUsageKind_Variable = 4

eComponentUsageKind_String = 5

17.7.2.10 ENUMConnectionType

Enumeration values to indicate the type of a connection. See also `Connection.ConnectionType`.

Members

eConnectionTypeTargetDriven = 0

eConnectionTypeSourceDriven = 1

eConnectionTypeCopyAll = 2

17.7.2.11 ENUMDOMType

Enumeration values to specify the DOM type used by generated C++ mapping code.

NOTE: The value `eDOMType_xerces` is obsolete. `eDOMType_xerces3` indicates Xerces 3.x usage. Obsolete in this context means that this value is not supported and should not be used.

Members

eDOMType_xerces = 1 (obsolete)

eDOMType_xerces3 = 2

eDOMType_msxml6 = 3

17.7.2.12 ENUMLibType

Enumeration values to specify the library type used by the generated C++ mapping code.

Members

eLibType_static = 0

eLibType_dll = 1

17.7.2.13 ENUMProgrammingLanguage

Enumeration values to select a programming language.

Members

eUndefinedLanguage = -1

eJava = 0

eCpp = 1

eCSharp = 2

eXSLT = 3

eXSLT2 = 4

eXQuery = 5

eXSLT3 = 6

17.7.2.14 ENUMProjectItemType

Enumeration to identify the different kinds of project items that can be children of `Project` or folder-like `ProjectItems`. See also `ProjectItem.Kind`.

Members

eProjectItemType_MappingFolder = 0

eProjectItemType_Mapping = 1

eProjectItemType_WebServiceFolder = 2

eProjectItemType_WebServiceRoot = 3

eProjectItemType_WebServiceService = 4

eProjectItemType_WebServicePort = 5

eProjectItemType_WebServiceOperation = 6

eProjectItemType_ExternalFolder = 7

eProjectItemType_LibraryFolder = 8

eProjectItemType_ResourceFolder = 9

eProjectItemType_VirtualFolder = 10

eProjectItemType_Count = 11

eProjectItemType_Invalid = -1

17.7.2.15 ENUMProjectType

Enumeration values to select a project type for generated C# and C++ mapping code.

Members

eVisualStudio2010Project = 6

eVisualStudio2013Project = 7

eVisualStudio2015Project = 8

eVisualStudio2017Project = 9

eVisualStudio2019Project = 10

eDotNetCore3_1 = 11

eDotNet5_0 = 12

17.7.2.16 ENUMSearchDatapointFlags

Enumeration values used as bit-flags; to be used as combination of flags when searching for a datapoint. See also
`GetChild`.

Members

eSearchDatapointElement = 1

eSearchDatapointAttribute = 2

17.7.2.17 ENUMViewMode

Enumeration values to select a MapForce view.

Members

eMapForceView = 0

eXSLView = 1

eOutputView = 2

18 ActiveX Integration

The MapForce user interface and the functionality described in this section can be integrated into custom applications that can consume ActiveX controls. ActiveX technology enables a wide variety of languages to be used for integration, such as C++, C#, VB.NET, HTML. (Note that ActiveX components integrated in HTML must be run with Microsoft Internet Explorer versions and platforms that support ActiveX). All components are full OLE Controls. Integration into Java is provided through wrapper classes.

> To integrate the ActiveX controls into your custom code, the MapForce Integration Package must be installed (see https://www.altova.com/components/download). Ensure that you install MapForce first, and then the MapForce Integration Package. Other prerequisites apply, depending on language and platform (see Prerequisites 1828).

You can flexibly choose between two different levels of integration: application level and document level.

Integration at application level means embedding the complete interface of MapForce (including its menus, toolbars, panes, etc) as an ActiveX control into your custom application. For example, in the most simple scenario, your custom application could consist of only one form that embeds the MapForce graphical user interface. This approach is easier to implement than integration at document level but may not be suitable if you need flexibility to configure the MapForce graphical user interface according to your custom requirements.

Integration at document level means embedding MapForce into your own application piece-by-piece. This includes implementing not only the main MapForce control but also the main document editor window, and, optionally, any additional windows. This approach provides greater flexibility to configure the GUI, but requires advanced interaction with ActiveX controls in your language of choice.

The sections Integration at the Application Level 1832 and Integration at Document Level 1835 describe the key steps at these respective levels. The ActiveX Integration Examples 1839 section provides examples in C#, HTML, and Java. Looking through these examples will help you to make the right decisions quickly. The Object Reference 1867 section describes all COM objects that can be used for integration, together with their properties and methods.

For information about using MapForce as a Visual Studio plug-in, see MapForce in Visual Studio 1426.

18.1 Prerequisites

To integrate the MapForce ActiveX control into a custom application, the following must be installed on your computer:

- MapForce
- The MapForce Integration Package, available for download at https://www.altova.com/components/download

To integrate the 64-bit ActiveX control, install the 64-bit versions of MapForce and MapForce Integration Package. For applications developed under Microsoft .NET platform with Visual Studio, both the 32-bit and 64-bit versions of MapForce and MapForce Integration Package must be installed, as explained below.

Microsoft .NET (C#, VB.NET) with Visual Studio

To integrate the MapForce ActiveX control into a 32-bit application developed under Microsoft .NET, the following must be installed on your computer:

- Microsoft .NET Framework 4.0 or later
- Visual Studio 2012/2013/2015/2017/2019/2022
- MapForce 32-bit and MapForce Integration Package 32-bit
- The ActiveX controls must be added to the Visual Studio toolbox (see Adding the ActiveX Controls to the Toolbox 1830).

If you want to integrate the 64-bit ActiveX control, the following prerequisites apply in addition to the ones above:

- MapForce 32-bit and MapForce Integration Package 32-bit must still be installed (this is required to provide the 32-bit ActiveX control to the Visual Studio designer, since Visual Studio runs on 32-bit)
- MapForce 64-bit and MapForce Integration Package 64-bit must be installed (provides the actual 64-bit ActiveX control to your custom application at runtime)
- In Visual Studio, create a 64-bit build configuration and build your application using this configuration. For an example, see Running the Sample C# Solution 1839.

Java

To integrate the MapForce ActiveX control into Java application using the Eclipse development environment, the following must be installed on your computer:

- Java Runtime Environment (JRE) or Java Development Kit (JDK) 7 or later
- Eclipse
- MapForce and MapForce Integration Package

Note: To run the 64-bit version of the MapForce ActiveX control, use a 64-bit version of Eclipse, as well as the 64-bit version of MapForce and the MapForce Integration Package.

MapForce integration and deployment on client computers

If you create a .NET application and intend to distribute it to other clients, you will need to install the following on the client computer(s):

- MapForce
- The MapForce Integration Package
- The custom integration code or application.

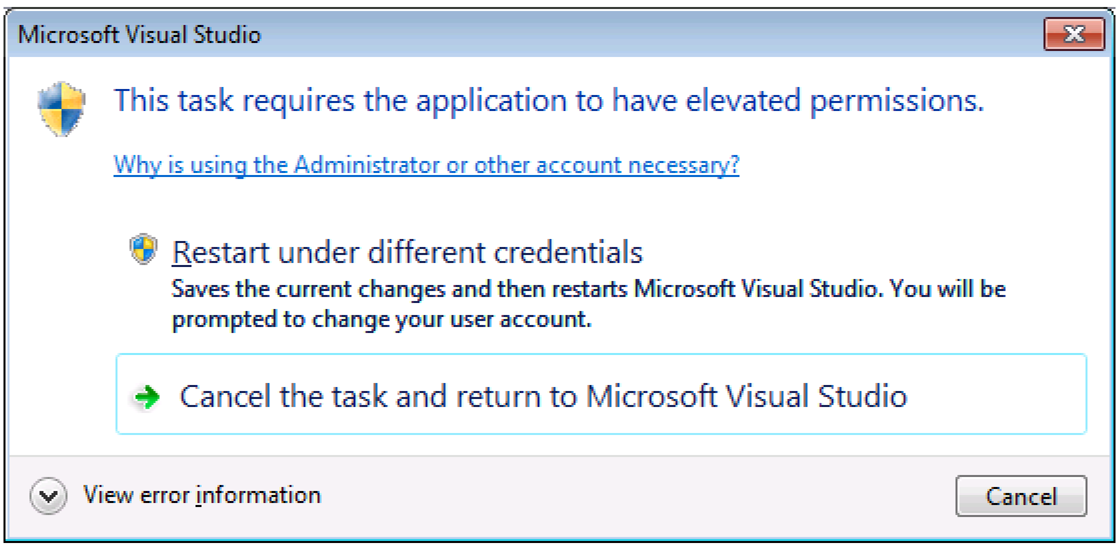
18.2 Adding the ActiveX Controls to the Toolbox

To use the MapForce ActiveX controls in an application developed with Visual Studio, the controls must first be added to the Visual Studio Toolbox, as follows:

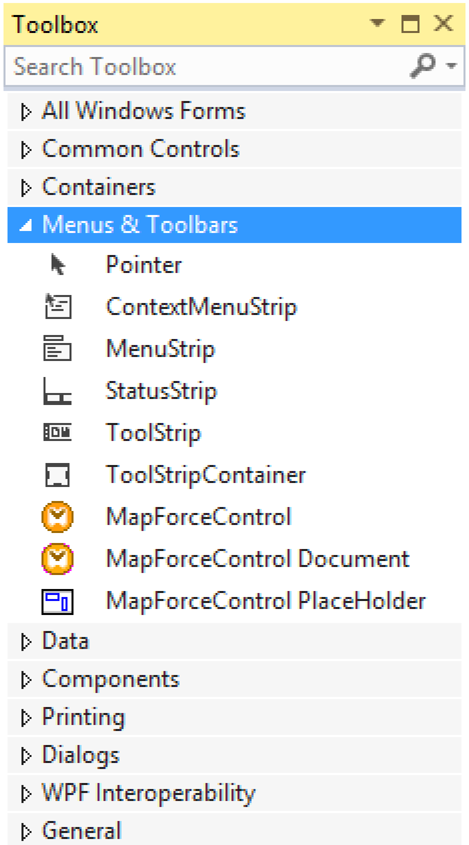
1. On the **Tools** menu of Visual Studio, click **Choose Toolbox Items**.
2. On the **COM Components** tab, select the check boxes next to the MapForceControl, MapForceControl Document, and MapForceControl Placeholder.

In case the controls above are not available, follow the steps below:

1. On the **COM Components** tab, click **Browse**, and select the **MapForceControl.ocx** file from the MapForce installation folder. Remember that the MapForce Integration Package must be installed; otherwise, this file is not available, see Prerequisites 1828.
2. If prompted to restart Visual Studio with elevated permissions, click **Restart under different credentials**.



If the steps above were successful, the MapForce ActiveX controls become available in the Visual Studio Toolbox.



Note: For an application-level integration, only the **MapForceControl** ActiveX control is used (see Integration at Application Level[1832]). The **MapForceControl Document** and **MapForceControl Placeholder** controls are used for document-level integration (see Integration at Document Level[1835]).

18.3 Integration at Application Level

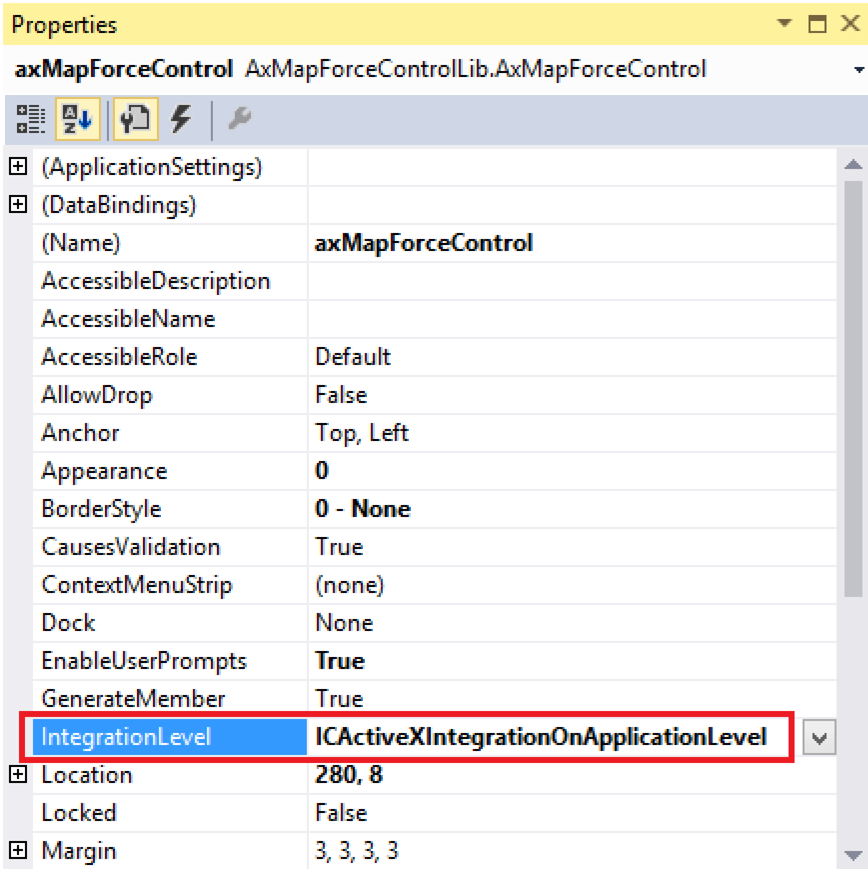
Integration at application level allows you to embed the complete interface of MapForce into a window of your application. With this type of integration, you get the whole user interface of MapForce, including all menus, toolbars, the status bar, document windows, and helper windows. Customization of the application's user interface is restricted to what MapForce provides. This includes rearrangement and resizing of helper windows and customization of menus and toolbars.

The only ActiveX control you need to integrate is `MapForceControl` 1870. Do not instantiate or access `MapForceControlDocument` 1878 or `MapForceControlPlaceHolder` 1884 ActiveX controls when integrating at application-level.

If you have any initialization to do or if you want to automate some behaviour of MapForce, use the properties, methods, and events described for `MapForceControl` 1870. Consider using `MapForceControl.Application` 1871 for more complex access to MapForce functionality.

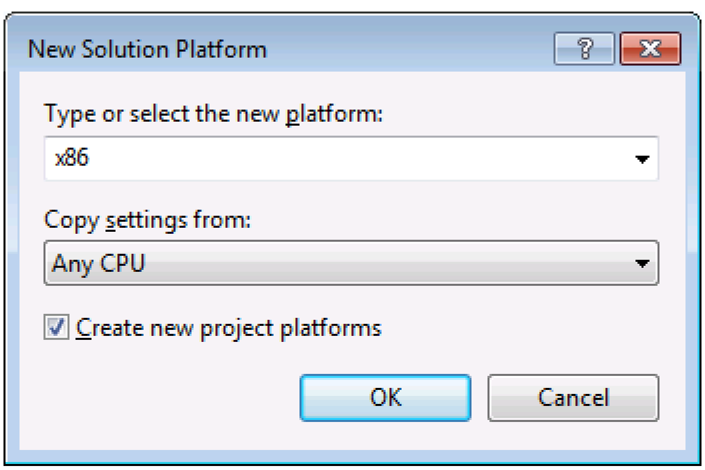
In C# or VB.NET with Visual Studio, the steps to create a basic, one-form application which integrates the MapForce ActiveX controls at application level are as follows:

1. Check that all prerequisites are met (see Prerequisites 1828).
2. Create a new Visual Studio Windows Forms project with a new empty form.
3. If you have not done that already, add the ActiveX controls to the toolbox (see Adding the ActiveX Controls to the Toolbox 1830).
4. Drag the **MapForceControl** from the toolbox onto your new form.
5. Select the **MapForceControl** on the form, and, in the Properties window, set the **IntegrationLevel** property to **ICActiveXIntegrationOnApplicationLevel**.



6. Create a build platform configuration that matches the platform under which you want to build (x86, x64). Here is how you can create the build configuration:

   a. Right-click the solution in Visual Studio, and select **Configuration Manager**.
   b. Under **Active solution platform**, select **New...** and then select the x86 or x64 configuration (in this example, **x86**).



You are now ready to build and run the solution in Visual Studio. Remember to build using the configuration that matches your target platform (x86, x64).

18.4 Integration at Document Level

Compared to integration at application level, integration at document level is a more complex, yet more flexible way to embed MapForce functionality into your application by means of ActiveX controls. With this approach, your code can access selectively the following parts of the MapForce user interface:

- Document editing window
- Project window
- Libraries window
- Overview window
- Messages window

As mentioned in Integration at Application Level[1832], for an ActiveX integration at application level, only one control is required, namely the **MapForceControl**. However, for an ActiveX integration at document level, MapForce functionality is provided by the following ActiveX controls:

- MapForceControl[1870]
- MapForceControl Document[1878]
- MapForceControl Placeholder[1884]

These controls are supplied by the **MapForceControl.ocx** file available in the application installation folder of MapForce. When you develop the ActiveX integration with Visual Studio, you will need to add these controls to the Visual Studio toolbox (see Adding the ActiveX Controls to the Toolbox[1830]).

The basic steps to integrate the ActiveX controls at document level into your application are as follows:

1. First, instantiate **MapForceControl** in your application. Instantiating this control is mandatory; it enables support for the **MapForceControl Document** and **MapForceControl Placeholder** controls mentioned above. It is important to set the `IntegrationLevel`[1872] property to **ICActiveXIntegrationOnDocumentLevel** (or "1"). To hide the control from the user, set its **Visible** property to **False**.

Note: When integrating at document level, do not use the **Open** method of the **MapForceControl**; this might lead to unexpected results. Use the corresponding open methods of **MapForceControl Document** and **MapForceControl PlaceHolder** instead.

2. Create at least one instance of **MapForceControl Document** in your application. This control supplies the document editing window of MapForce to your application and can be instantiated multiple times if necessary.

   Use the method **Open** to load any existing file. To access document-related functionality, use the **Path** and **Save** or methods and properties accessible via the property **Document**.

Note: The control does not support a read-only mode. The value of the property **ReadOnly** is ignored.

3. Optionally, add to your application the **MapForceControl Placeholder** control for each additional window (other than the document window) that must be available to your application.

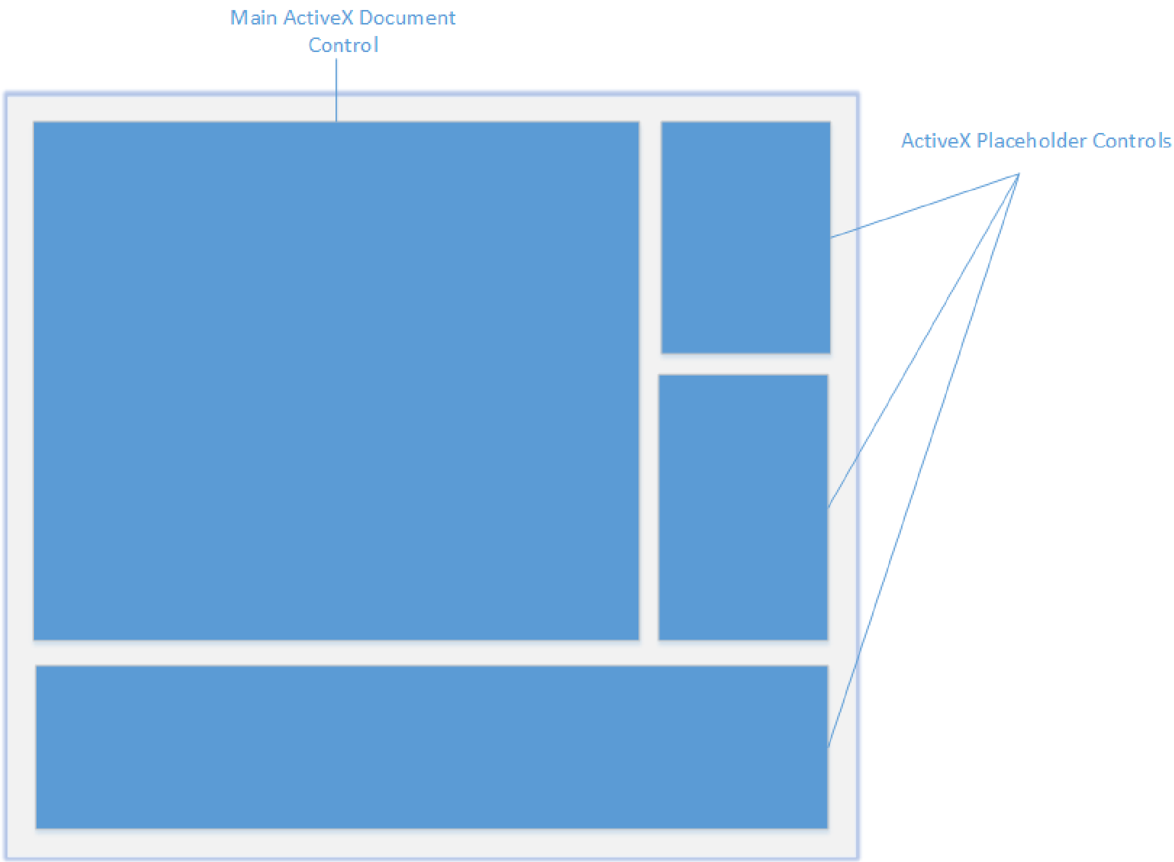
   Instances of **MapForceControl PlaceHolder** allow you to selectively embed additional windows of MapForce into your application. The window kind (for example, Project window) is defined by the property **PlaceholderWindowID**. Therefore, to set the window kind, set the property **PlaceholderWindowID**. For valid window identifiers, see MapForceControlPlaceholderWindow 1887.

Note: Use only one **MapForceControl PlaceHolder** for each window identifier.

For placeholder controls that select the MapForce project window, additional methods are available. Use **OpenProject** to load a MapForce project. Use the property Project and the methods and properties from the MapForce automation interface to perform any other project related operations.

For example, in C# or VB.NET with Visual Studio, the steps to create a basic, one-form application which integrates the MapForce ActiveX controls at document level could be similar to those listed below. Note that your application may be more complex if necessary; however, the instructions below are important to understand the minimum requirements for an ActiveX integration at document level.

1. Create a new Visual Studio Windows Forms project with a new empty form.
2. If you have not done that already, add the ActiveX controls to the toolbox (see Adding the ActiveX Controls to the Toolbox 1830).
3. Drag the **MapForceControl** 1870 from the toolbox onto your new form.
4. Set the **IntegrationLevel** property of the **MapForceControl** to **ICActiveXIntegrationOnDocumentLevel**, and the **Visible** property to **False**. You can do this either from code or from the **Properties** window.
5. Drag the **MapForceControl Document** 1878 from the toolbox onto the form. This control provides the main document window of MapForce to your application, so you may need to resize it to a reasonable size for a document.
6. Optionally, add one or more **MapForceControl Placeholder** 1884 controls to the form (one for each additional window type that your application needs, for example, the **Project** window). You will typically want to place such additional placeholder controls either below or to the right or left of the main document control, for example:



7. Set the **PlaceholderWindowID** property of each **MapForceControl Placeholder** control to a valid window identifier. For the list of valid values, see MapForceControlPlaceholderWindow[1887].
8. Add commands to your application (at minimum, you will need to open, save and close documents), as shown below.

Querying MapForce Commands

When you integrate at document level, no MapForce menu or toolbar is available to your application. Instead, you can retrieve the required commands, view their status, and execute them programmatically, as follows:

- To retrieve all available commands, use the `CommandsList`[1871] property of the **MapForceControl**.
- To retrieve commands organized according to their menu structure, use the `MainMenu`[1872] property.
- To retrieve commands organized by the toolbar in which they appear, use the `Toolbars`[1873] property.
- To send commands to MapForce, use the `Exec`[1874] method.
- To query if a command is currently enabled or disabled, use the `QueryStatus`[1875] method.

This enables you to flexibly integrate MapForce commands into your application's menus and toolbars.

Your installation of MapForce also provides you with command label images used within MapForce. See the folder **<ApplicationFolder>\Examples\ActiveX\Images** of your MapForce installation for icons in GIF format. The file names correspond to the command names as they are listed in the Command Reference[1858] section.

General considerations

To automate the behaviour of MapForce, use the properties, methods, and events described for the **MapForceControl** (1870), **MapForceControl Document** (1878), and **MapForceControl Placeholder** (1884).

For more complex access to MapForce functionality, consider using the following properties:

- `MapForceControl.Application` (1871)
- `MapForceControlDocument.Document` (1879)
- `MapForceControlPlaceHolder.Project` (1885)

These properties give you access to the MapForce automation interface (MapForceAPI)

Note: To open a document, always use `MapForceControlDocument.Open` (1881) or `MapForceControlDocument.New` (1880) on the appropriate document control. To open a project, always use `MapForceControlPlaceHolder.OpenProject` (1885) on a placeholder control embedding a MapForce project window.

For examples that show how to instantiate and access the necessary controls in different programming environments, see ActiveX Integration Examples (1839).

18.5 ActiveX Integration Examples

This section contains examples of MapForce document-level integration using different container environments and programming languages. Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX` of your MapForce installation.

18.5.1 C#

A basic ActiveX integration example solution for C# and Visual Studio is available in the folder `<ApplicationFolder>\Examples\ActiveX\C#`. Before you compile the source code and run the sample, make sure that all prerequisites are met (see Running the Sample C# Solution 1839).

18.5.1.1 Running the Sample C# Solution

The sample Visual Studio solution available in the folder **<ApplicationFolder>\Examples\ActiveX\C#** illustrates how to consume the MapForce ActiveX controls. Before attempting to build and run this solution, note the following steps:

Step 1: Check the prerequisites

Visual Studio 2010 or later is required to open the sample solution. For the complete list of prerequisites, see Prerequisites 1828.

Step 2: Copy the sample to a directory where you have write permissions

To avoid running Visual Studio as an Administrator, copy the source code to a directory where you have write permissions, instead of running it from the default location.

Step 3: Check and set all required control properties

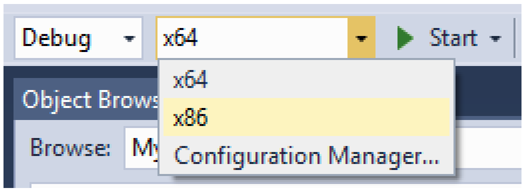
The sample application contains one instance of MapForceControlDocument 1878 and several instances of MapForceControlPlaceHolder 1884 controls. Double-check that the following properties of these controls are set as shown in the table below:

Control name	Property	Property value
axMapForceControl	IntegrationLevel	ICActiveXIntegrationOnDocumentLevel
axMapForceControlLibrary	PlaceholderWindowID	0
axMapForceControlOutput	PlaceholderWindowID	2
axMapForceControlPreview	PlaceholderWindowID	1

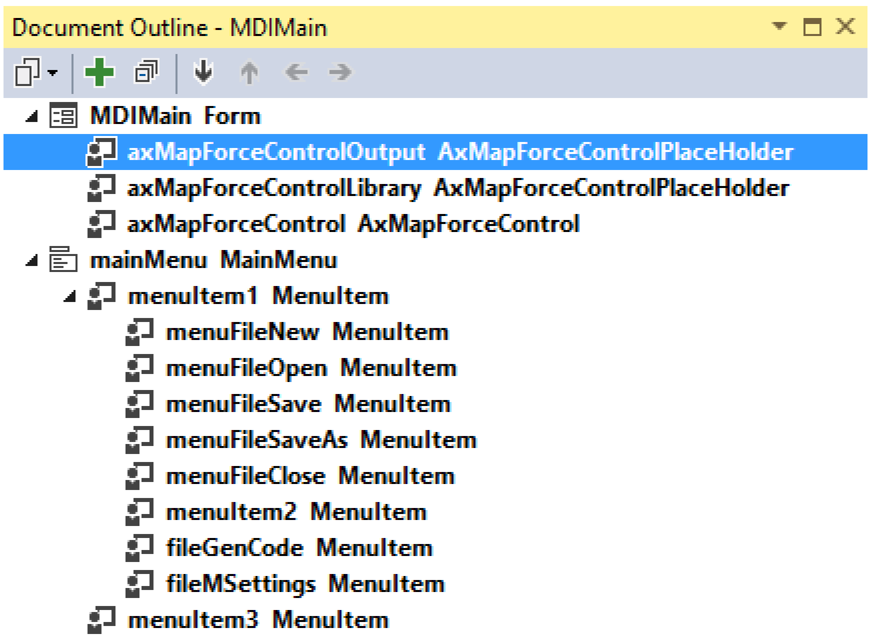
Here is how you can view or set the properties of an ActiveX control:

1. Open the **MDIMain.cs** form in the designer window.

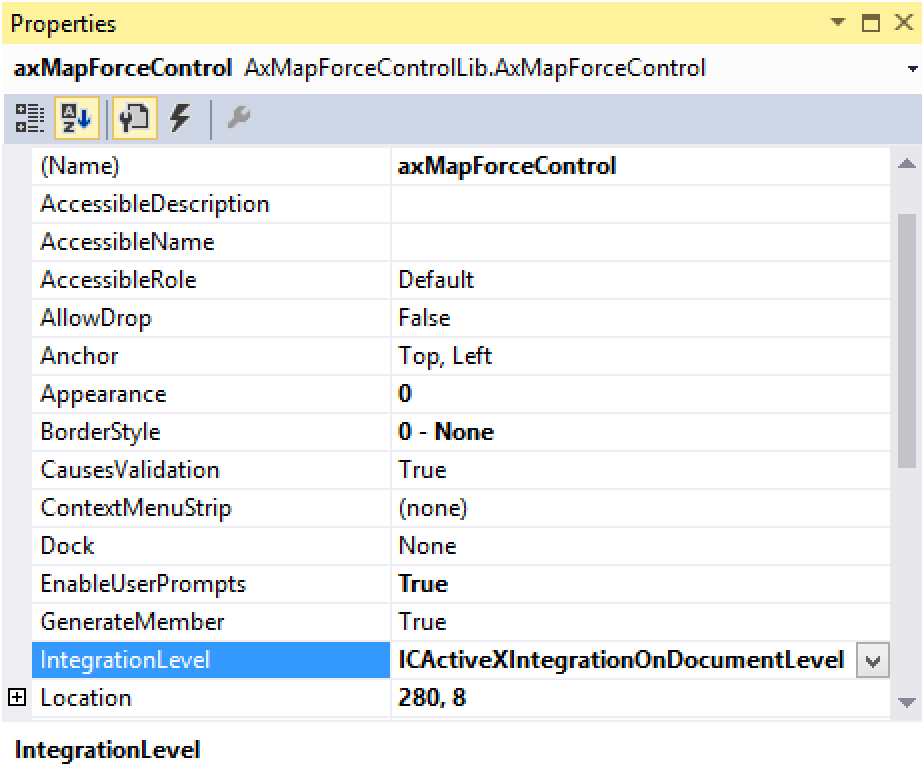
Note: On 64-bit Windows, it may be necessary to change the build configuration of the Visual Studio solution to "x86" **before** opening the designer window. If you need to build the sample as a 64-bit application, see Prerequisites 1828.



2. Open the **Document Outline** window of Visual Studio (On the **View** menu, click **Other Windows | Document Outline**).



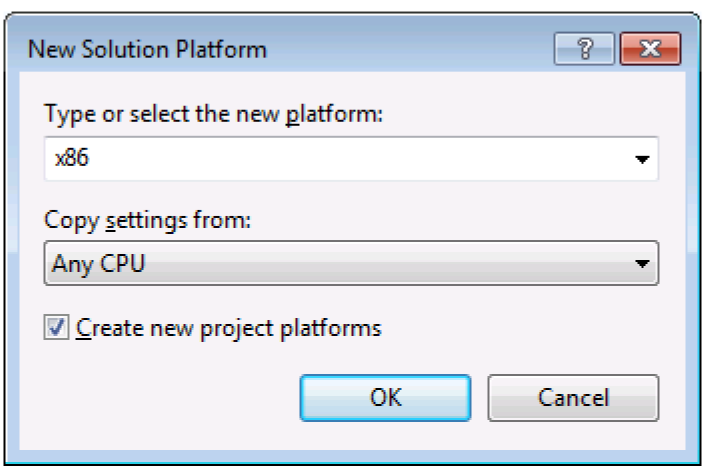
3. Click an ActiveX control in the **Document Outline** window, and edit its required property in the **Properties** window, for example:



Step 4: Set the build platform

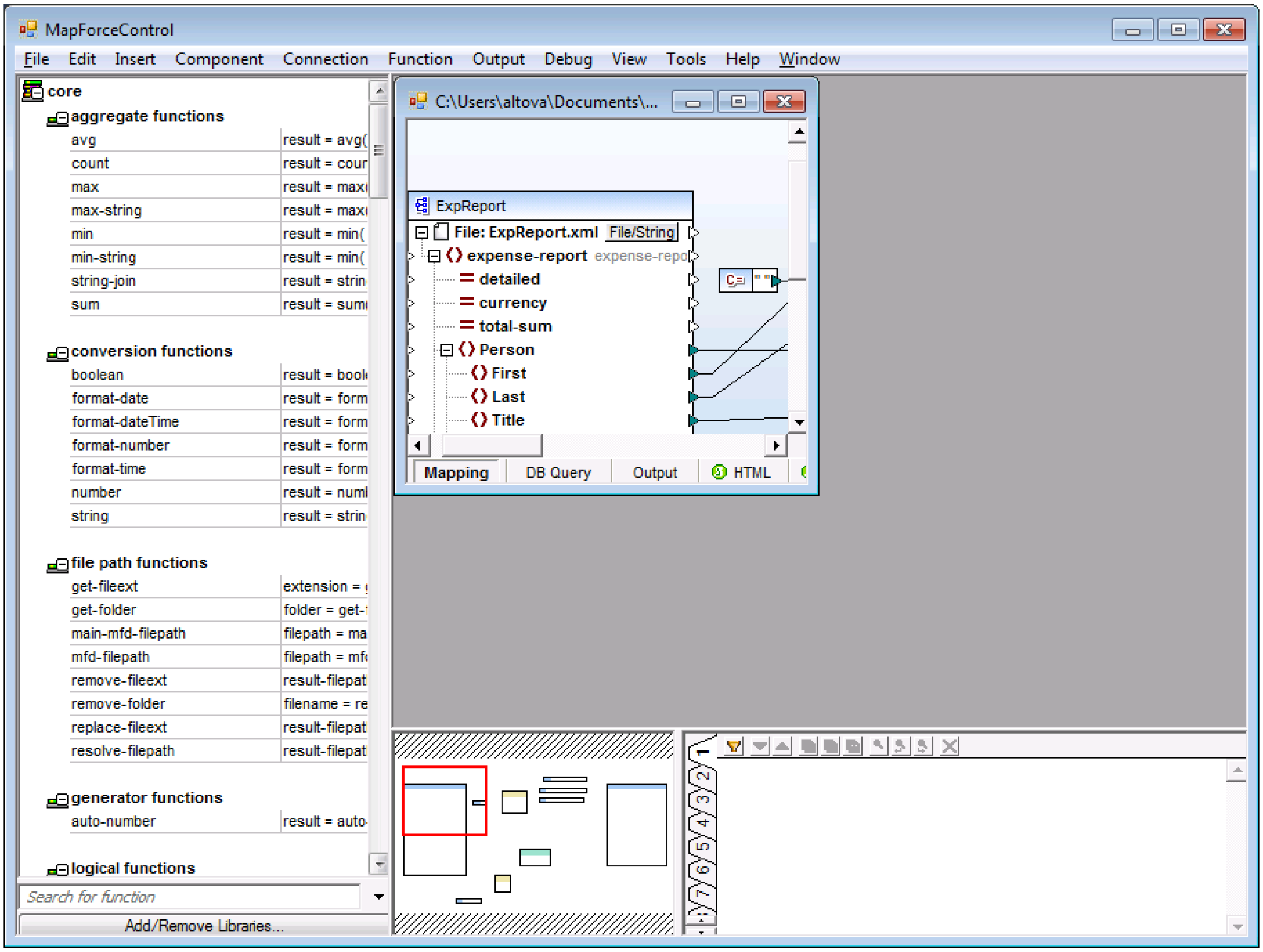
- Create a build platform configuration that matches the platform under which you want to build (x86, x64). Here is how you can create the build configuration:

  a. Right-click the solution in Visual Studio, and select **Configuration Manager**.
  b. Under **Active solution platform**, select **New...** and then select the x86 or x64 configuration (in this example, **x86**).



You are now ready to build and run the solution in Visual Studio. Remember to build using the configuration that matches your target platform (x86, x64); otherwise, runtime errors might occur.

On running the sample, the main MDI Frame window is displayed. Use **File | Open** to open a mapping file (for example, `MarketingExpenses.mfd`, which is in the MapForce examples folder). The file is loaded and displayed in a new document child window:



After you load the document, you can execute commands against the active document using the menu. Context menus are also available. You can also load additional documents. Save any modifications using the **File | Save** command.

18.5.1.2 Retrieving Command Information

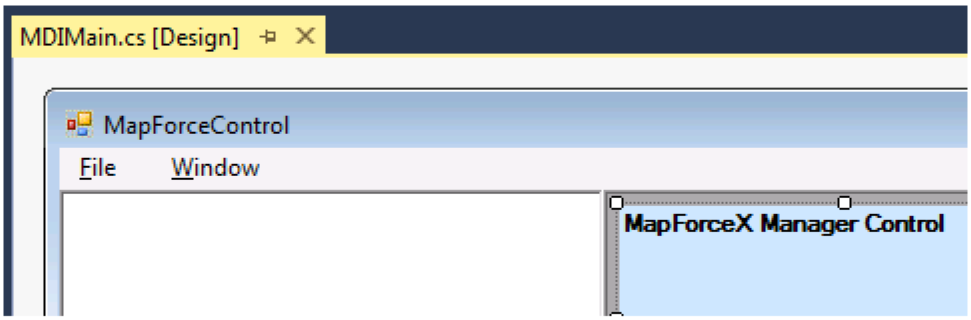
The MapForceControl gives access to all commands of MapForce through its `CommandsList` , `MainMenu`, and `Toolbars` properties. The example project available in the folder **<ApplicationFolder>\Examples\ActiveX\C#** uses the `MainMenu` property to create the MapForce menu structure dynamically.

The code that gets the menu commands can be found in the `MDIMain` method in `MDIMain.cs` file:

```
public MDIMain()
{
```

```
    // ...

    // Get the MainMenu property of the control and create the menu structure from it.
    MFLib.MapForceCommand objCommand = this.axMapForceControl.MainMenu;
    InsertMenuStructure(mainMenu, objCommand);
}
```

In the code listing above, `mainMenu` is the existing static menu of the main MDI Frame window. If you open the **MDIMain.cs** form in the Visual Studio Designer, you will notice that this menu contains two menu items: **File** and **Window**.



MDIMain.cs

The method `InsertMenuStructure` takes as parameters the `mainMenu` and the `objCommand` objects (the former is the existing static menu, while the latter contains the full menu structure retrieved from the MapForce ActiveX control). The retrieved MapForce menu structure is then merged into the existing static menu. Note that the menus **File**, **Project**, and **Window** are not added dynamically. This is intentional, because these menus deal with actively open documents, and they would require code which is beyond the scope of this example. The basic file management commands (create, open, save, bring into focus) are handled by the existing static menus **File** and **Window**. All other menus are inserted dynamically based on the information taken from the `MainMenu` property of the ActiveX control. The new menus are inserted after "File" but before "Window", i.e. starting at menu index 1.

The method `InsertMenuStructure` iterates through all top-level menus found in `MapForceCommand` object and adds a new menu item for each. Since each top-level menu has its own child menu items, a call to the method `InsertMenuCommand` takes place for each encountered child menu item. Furthermore, since each child menu item can have its own children menu items, and so on, the `InsertMenuCommand` method recurses into itself until no more child menu items exist.

The commands added dynamically are instances of the class `CustomMenuItem`, which is defined in `CustomMenuItem.cs`. This class is derived from `System.Windows.Forms.MenuItem` class and has an additional member to store the MapForce command ID.

```
public class CustomMenuItem : System.Windows.Forms.MenuItem
{
       public int m_MapForceCmdID;
}
```

All dynamically added commands (except those that are containers for other commands) get the same event handler `AltovaMenuItem_Click` which does the processing of the command:

```
private void AltovaMenuItem_Click(object sender, EventArgs e)
{
  if(sender.GetType() == System.Type.GetType("MapForceApplication.CustomMenuItem"))
  {
    CustomMenuItem   customItem = (CustomMenuItem)sender;
    ProcessCommand(customItem.m_MapForceCmdID);
  }
}
```

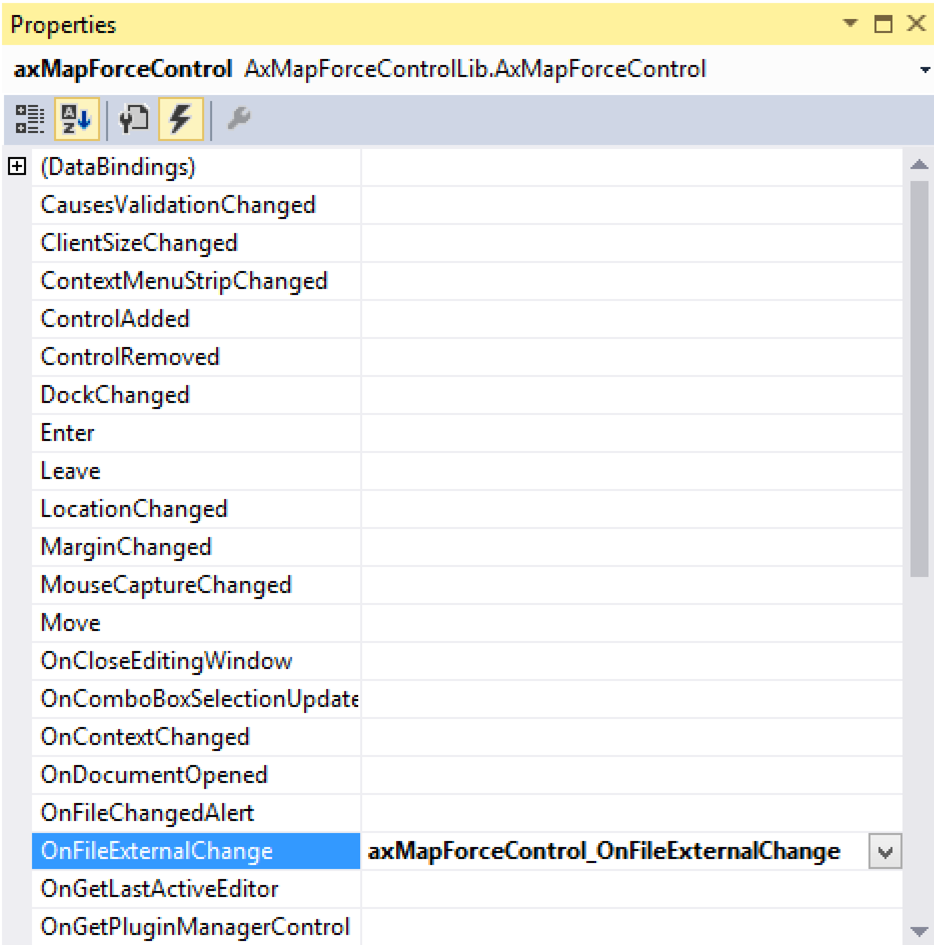
If the command is a container for other commands (that is, if it has child commands), it gets the event handler `AltovaSubMenu_Popup`. This handler queries the status of each child command and enables or disables it as required. This ensures that each command is enabled only when that is meaningful (for example, the **File | Save** menu item should be disabled if there is no active document open).

The method `ProcessCommand` delegates the execution either to the MapForceControl itself or to any active MapForce document loaded in a `MapForceControlDocument` control. This is necessary because the MapForceControl has no way to know which document is currently active in the hosting application.

```
private void ProcessCommand(int nID)
{
   MapForceDoc docMapForce = GetCurrentMapForceDoc();

   if(docMapForce != null)
      docMapForce.axMapForceControlDoc.Exec(nID);
   else
      axMapForceControl.Exec(nID);
}
```

18.5.1.3 Handling Events

Because all events in the MapForce library are based on connection points, you can use the C# delegate mechanism to provide the custom event handlers. You will always find a complete list of events on the property page of each control of the MapForce library. The image below shows the events of the main MapForceControl:



As you can see, the example project only overrides the `OnFileExternalChange` event. The creation of the C# delegate is done for you by the C# Framework. All you need to do is fill in the empty event handler.

For example, the handler implementation shown below turns off any file reloading and displays a message box to inform the user that a file loaded by the MapForceControl has been changed from outside:

```
private void axMapForceControl_OnFileExternalChange(object sender,
AxMapForceControlLib._DMapForceControlEvents_OnFileExternalChangeEvent e)
{
    MessageBox.Show("Attention: The file " + e.strPath + " has been changed from
outside\nbut reloading is turned off in the sample application!");

    // This turns off any file reloading:
    e.varRet = false;
}
```

18.5.2 Java

MapForce ActiveX components can be accessed from Java code. Java integration is provided by the libraries listed below. These libraries are available in the folder `<ApplicationFolder>\Examples\JavaAPI` of your MapForce installation, after you have installed both MapForce and the MapForce Integration Package (see also Prerequisites 1828).

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers (in case of the 32-bit installation of MapForce)
- `AltovaAutomation_x64.dll`: a JNI wrapper for Altova automation servers (in case of the 64-bit installation of MapForce)
- `AltovaAutomation.jar`: Java classes to access Altova automation servers
- `MapForceActiveX.jar`: Java classes that wrap the MapForce ActiveX interface
- `MapForceActiveX_JavaDoc.zip`: a Javadoc file containing help documentation for the Java interface

Note: In order to use the Java ActiveX integration, the .dll and .jar files must be included in the Java class search path.

Example Java project

An example Java project is supplied with your product installation. You can test the Java project and modify and use it as you like. For more details, see Example Java Project 1847.

Rules for mapping the ActiveX Control names to Java

For the documentation of ActiveX controls, see Object Reference 1867. Note that the object naming conventions are slightly different in Java compared to other languages. Namely, the rules for mapping between the ActiveX controls and the Java wrapper are as follows:

- ***Classes and class names***
  For every component of the MapForce ActiveX interface a Java class exists with the name of the component.

- ***Method names***
  Method names on the Java interface are the same as used on the COM interfaces but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with get and set can be used. If a property does not support write-access, no setter method is available. Example: For the `IntegrationLevel` property of the MapForceControl, the Java methods `getIntegrationLevel` and `setIntegrationLevel` are available.

- ***Enumerations***
  For every enumeration defined in the ActiveX interface, a Java enumeration is defined with the same name and values.

- ***Events and event handlers***
  For every interface in the automation interface that supports events, a Java interface with the same name plus 'Event' is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name of this Java class is the name of the event interface plus '`DefaultHandler`'. For example:
  `MapForceControl`: Java class to access the application
  `MapForceControlEvents`: Events interface for the MapForceControl

`MapForceControlEventsDefaultHandler`: Default handler for `MapForceControlEvents`

Exceptions to mapping rules

There are some exceptions to the rules listed above. These are listed below:

Interface	Changes in Java class
`MapForceControlDocument`, method `New`	Renamed to `newDocument`
`MapForceControlDocument`, method `OpenDocument`	Removed. Use the `Open` method
`MapForceControlDocument`, method `NewDocument`	Removed. Use the `newDocument` method
`MapForceControlDocument`, method `SaveDocument`	Removed. Use the `Save` method

This section

This section shows how some basic MapForce ActiveX functionality can be accessed from Java code. It is organized into the following sub-sections:

18.5.2.1 Example Java Project

The MapForce installation package contains an example Java project, located in the ActiveX Examples folder of the application folder: `<ApplicationFolder>\Examples\ActiveX\Java\`.

The Java example shows how to integrate the MapForceControl in a common desktop application created with Java. You can test it directly from the command line using the batch file `BuildAndRun.bat`, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list

The Java examples folder contains all the files required to run the example project. These files are listed below:

`.classpath`	Eclipse project helper file
`.project`	Eclipse project file
`AltovaAutomation.dll`	Java-COM bridge: DLL part (for the 32-bit installation)
`AltovaAutomation_x64.dll`	Java-COM bridge: DLL part (for the 64-bit installation)

`AltovaAutomation.jar`	Java-COM bridge: Java library part
`BuildAndRun.bat`	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
`MapForceActiveX.jar`	Java classes of the MapForce ActiveX control
`MapForceActiveX_JavaDoc.zip`	Javadoc file containing help documentation for the Java API
`MapForceContainer.java`	Java example source code
`MapForceContainerEventHandler.java`	Java example source code
`MapForceTable.java`	Java example source code

What the example does

The example places one MapForce document editor window, the MapForce project window, the MapForce library window and the MapForce validation window in an AWT frame window. It reads out the main menu defined for MapForce and creates an AWT menu with the same structure. You can use this menu or the project window to open and work with files in the document editor.

You can modify the example in any way you like.

The following specific features are described in code listings:

- Creating the ActiveX Controls 1849: Starts MapForce, which is registered as an automation server, or activates MapForce if it is already running.
- Loading Data in the Controls 1850: Locates one of the example documents installed with MapForce and opens it.
- Basic Event Handling 1850: Changes the view of all open documents to Text View. The code also shows how to iterate through open documents.
- Menus 1851: Validates the active document and shows the result in a message box. The code shows how to use output parameters.
- UI Update Event Handling 1853: Shows how to handle MapForce events.
- Creating a MapForce Mapping Table 1853: Shows how to create a MapForce mapping table and prepare it for modal activation.

Updating the path to the Examples folder

Before running the provided sample, you may need to edit the **MapForceContainer.java** file. Namely, check that the following path refers to the actual folder where the MapForce example files are stored on your operating system:

```
// Locate samples installed with the product.
final String strExamplesFolder = System.getenv( "USERPROFILE" ) + "\\Documents\\Altova\
\MapForce2023\\MapForceExamples\\";
```

Running the example from the command line

To run the example from the command line:

1. Check that all prerequisites are met (see Prerequisites[1828]).
2. Open a command prompt window, change the current directory to the sample Java project folder, and type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

3. Press **Enter**.

The Java source in MapForceContainer.java will be compiled and then executed.

Compiling and running the example in Eclipse

To import the sample Java project into Eclipse:

1. Check that all prerequisites are met (see Prerequisites[1828]).
2. On the **File** menu, click **Import**.
3. Select **Existing Projects into Workspace**, and browse for the Eclipse project file located at <ApplicationFolder>\Examples\ActiveX\Java\. Since you may not have write-access in this folder, it is recommended to select the **Copy projects into workspace** check box on the Import dialog box.

To run the example application, right-click the project in Package Explorer and select the command **Run as | Java Application**.

Help for Java API classes is available through comments in code as well as the Javadoc view of Eclipse. To enable the Javadoc view in Eclipse, select the menu command **Window | Show View | JavaDoc**.

18.5.2.2 Creating the ActiveX Controls

The code listing below show how ActiveX controls can be created. The constructors will create the Java wrapper objects. Adding these Canvas-derived objects to a panel or to a frame will trigger the creation of the wrapped ActiveX object.

```
/**
 * MapForce manager control - always needed
 */
public static MapForceControl         mapForceControl = null;

/**
 * MapForceDocument editing control
 */
public static MapForceControlDocument     mapForceDocument = null;

/**
 * Tool windows - MapForce place-holder controls
 */
private static MapForceControlPlaceHolder   mapForceProjectToolWindow = null;
private static MapForceControlPlaceHolder   mapForceValidationToolWindow = null;
private static MapForceControlPlaceHolder   mapForceLibraryToolWindow = null;

```

```
// Create the MapForce ActiveX control; the parameter determines that we want
// to place document controls and place-holder controls individually.
// It gives us full control over the menu, as well.
   mapForceControl = new MapForceControl(
   ICActiveXIntegrationLevel.ICActiveXIntegrationOnDocumentLevel.getValue(), false );

   mapForceDocument = new MapForceControlDocument();
   frame.add( mapForceDocument, BorderLayout.CENTER );


// Create a project window and open the sample project in it
   mapForceProjectToolWindow = new MapForceControlPlaceHolder(
   MapForceControlPlaceholderWindow.MapForceXProjectWindow.getValue(),
   strExamplesFolder + "MapForceExamples.mfp" ) ;
   mapForceProjectToolWindow.setPreferredSize( new Dimension( 200, 200 ) );
```

18.5.2.3 Loading Data in the Controls

The code listing below show how data can be loaded in the ActiveX controls.

```
// Locate samples installed with the product.
   final String strExamplesFolder = System.getenv( "USERPROFILE" ) +
   "\\Documents\\Altova\\MapForce2022\\MapForceExamples\\";
   mapForceProjectToolWindow = new
MapForceControlPlaceHolder( MapForceControlPlaceholderWindow.MapForceXProjectWindow.getValu
e(), strExamplesFolder + "MapForceExamples.mfp" ) ;
```

18.5.2.4 Basic Event Handling

The code listing below shows how basic events can be handled. When calling the MapForceControl's `open` method, or when trying to open a file via the menu or Project tree, the `onOpenedOrFocused` event is sent to the attached event handler. The basic handling for this event is opening the file by calling the MapForceDocumentControl's `open` method.

```
    // Open the Marketing file when button is pressed
    btnMarkExp.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          // Instruct the Document control to open the file - avoid calling the open
method of MapForceControl (see help)
          mapForceDocument.open( strExamplesFolder + "MarketingExpenses.mfd" );
          mapForceDocument.requestFocusInWindow();
        } catch (AutomationException e1) {
          e1.printStackTrace();
        }
      }
    } );
    public void onOpenedOrFocused( String i_strFileName, boolean
i_bOpenWithThisControl, boolean i_bFileAlreadyOpened ) throws AutomationException
  {
    // Handle the New/Open events coming from the Project tree or from the menus
    if ( !i_bFileAlreadyOpened )
```

```
    {
      // This is basically an SDI interface, so open the file in the already existing document control
      try {
        MapForceContainer.mapForceDocument.open( i_strFileName );
        MapForceContainer.mapForceDocument.requestFocusInWindow();
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
```

18.5.2.5 Menus

The code listing below shows how menu items can be created. Each `MapForceCommand` object gets a corresponding `MenuItem` object, with the `ActionCommand` set to the ID of the command. The actions generated by all menu items are handled by the same function, which can perform specific handlings (like reinterpreting the closing mechanism) or can delegate the execution to the `MapForceControl` object by calling its `exec` method. The `menuMap` object that is filled during menu creation is used later (see section UI Update Event Handling1853).

```

      // Load the file menu when the button is pressed
      btnMenu.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          try {
            // Create the menubar that will be attached to the frame
            MenuBar mb = new MenuBar();
            // Load the main menu's first item - the File menu
            MapForceCommand xmlSpyMenu = mapForceControl.getMainMenu().getSubCommands().getItem( 0 );
            // Create Java menu items from the Commands objects
            Menu fileMenu = new Menu();
            handlerObject.fillMenu( fileMenu, xmlSpyMenu.getSubCommands() );
            fileMenu.setLabel( xmlSpyMenu.getLabel().replace( "&", "" ) );
            mb.add( fileMenu );
            frame.setMenuBar( mb );
            frame.validate();
          } catch (AutomationException e1) {
            e1.printStackTrace();
          }
          // Disable the button when the action has been performed
          ((AbstractButton) e.getSource()).setEnabled( false );
        }
      } ) ;
  /**
   * Populates a menu with the commands and submenus contained in an MapForceCommands object
   */
  public void fillMenu(Menu newMenu, MapForceCommands mapForceMenu) throws AutomationException
  {
    // For each command/submenu in the mapForceMenu
    for ( int i = 0 ; i < mapForceMenu.getCount() ; ++i  )
    {
```

```
      MapForceCommand mapForceCommand = mapForceMenu.getItem( i );
      if ( mapForceCommand.getIsSeparator() )
        newMenu.addSeparator();
      else
      {
        MapForceCommands subCommands = mapForceCommand.getSubCommands();
        // Is it a command (leaf), or a submenu?
        if ( subCommands.isNull() || subCommands.getCount() == 0 )
        {
          // Command -> add it to the menu, set its ActionCommand to its ID and store it in the menuMap
          MenuItem mi = new MenuItem( mapForceCommand.getLabel().replace( "&", "" ) );
          mi.setActionCommand( "" + mapForceCommand.getID() );
          mi.addActionListener( this );
          newMenu.add( mi );
          menuMap.put( mapForceCommand.getID(), mi );
        }
        else
        {
          // Submenu -> create submenu and repeat recursively
          Menu newSubMenu = new Menu();
          fillMenu( newSubMenu, subCommands );
          newSubMenu.setLabel( mapForceCommand.getLabel().replace( "&", "" ) );
          newMenu.add( newSubMenu );
        }
      }
    }
  }
  /**
   * Action handler for the menu items
   * Called when the user selects a menu item; the item's action command corresponds to the command table for MapForce
   */
  public void actionPerformed( ActionEvent e )
  {
    try
    {
      int iCmd = Integer.parseInt( e.getActionCommand() );
      // Handle explicitly the Close commands
      switch ( iCmd )
      {
        case 57602:       // Close
        case 34050:       // Close All
          MapForceContainer.initMapForceDocument();
          break;
        default:
          MapForceContainer.mapForceControl.exec( iCmd );
          break;
      }
    }
    catch ( Exception ex )
    {
      ex.printStackTrace();
    }

  }
```

18.5.2.6 UI Update Event Handling

The code listing below shows how a UI-Update event handler can be created.

```
/**
   * Call-back from the MapForceControl.
   * Called to enable/disable commands
   */
  @Override
  public void onUpdateCmdUI() throws AutomationException
  {
    // A command should be enabled if the result of queryStatus contains the Supported (1) and Enabled (2) flags
    for ( java.util.Map.Entry<Integer, MenuItem> pair : menuMap.entrySet() )
      pair.getValue().setEnabled( MapForceContainer.mapForceControl.queryStatus( pair.getKey() ) > 2 );
  }
 /**
   * Call-back from the MapForceControl.
   * Usually called while enabling/disabling commands due to UI updates
   */
  @Override
  public boolean onIsActiveEditor( String i_strFilePath ) throws AutomationException
  {
    try {
      return MapForceContainer.mapForceDocument.getDocument().getFullName().equalsIgnoreCase( i_strFilePath );
    } catch ( Exception e ) {
      return false;
    }
  }
```

18.5.2.7 Listing the Properties of a MapForce Mapping

The listing below shows how a Mapping object in MapForce can be loaded as a table and prepared for modal activation.

```
//access MapForce Java-COM bridge
import com.altova.automation.MapForce.*;
import com.altova.automation.MapForce.Component;
import com.altova.automation.MapForce.Enums.ENUMComponentUsageKind;

//access AWT and Swing components
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;


/**
 * A simple example of a table control loading the structure from a Mapping object.
```

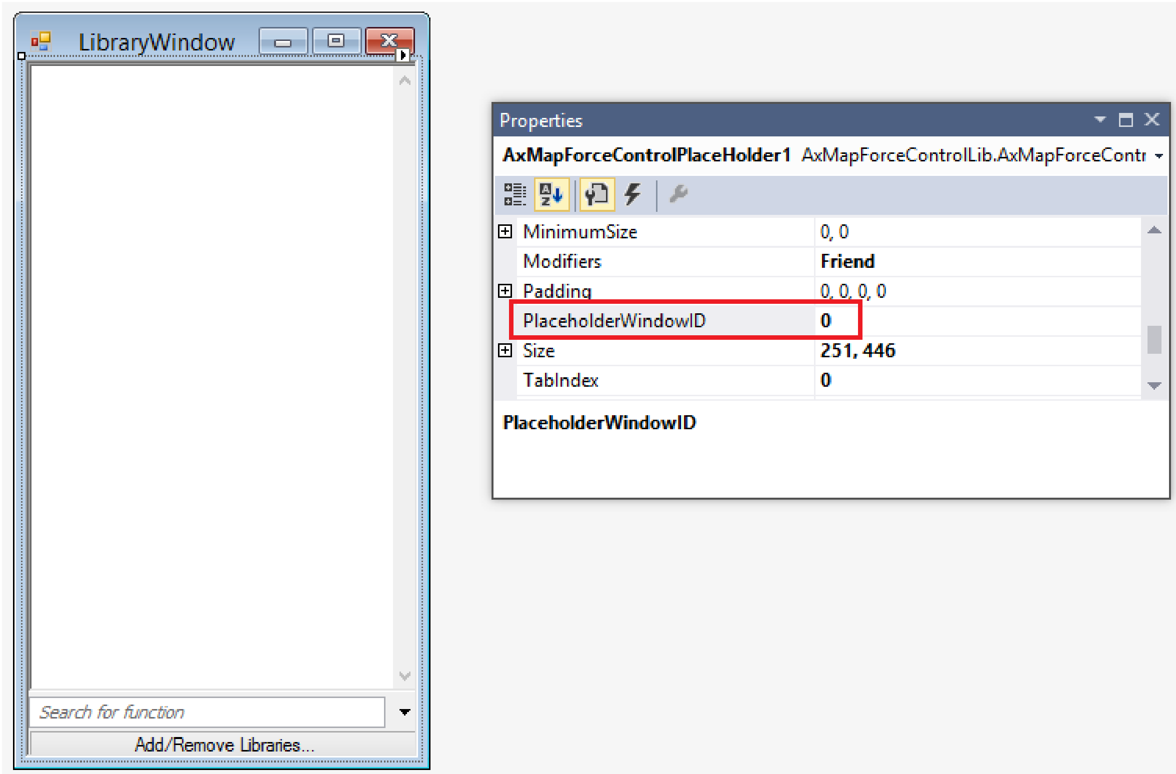
```
 * The class receives an Mapping object, loads its components in a JTable, and prepares
 * for modal activation.
 *
 * Feel free to modify and extend this sample.
 *
 * @author Altova GmbH
 */
class MapForceTable extends JDialog
{
  /**
   * The table control
   */
  private JTable myTable;

  /**
   * Constructor that prepares the modal dialog containing the filled table control
   * @param mapping The data to be displayed in the table
   * @param parent  Parent frame
   */
  public MapForceTable( Mapping mapping, Frame parent )
  {
    // Construct the modal dialog
    super( parent, "MapForce component table", true );
    // Build up the tree
    fillTable( mapping );
    // Arrange controls in the dialog
    setContentPane( new JScrollPane( myTable ) );
  }

  /**
   * Loads the components of a Mapping object in the table
   * @param mapping Source data
   */
  private void fillTable( Mapping mapping)
  {
    try
    {
      // count how many Instance components do we have
      int size = 0;
      for (Component comp : mapping.getComponents())
        if ( comp.getUsageKind() == ENUMComponentUsageKind.eComponentUsageKind_Instance )
          ++size;

      // Prepare data
      final String[] columnNames = { "Component", "Has inputs", "Has outputs", "Input file", "Output file", "Schema" };
      final Object[][] data = new Object[size ][ 7 ] ;
      int index = 0 ;
      for (Component comp : mapping.getComponents())
        if ( comp.getUsageKind() == ENUMComponentUsageKind.eComponentUsageKind_Instance )
        {
          int i = 0;
          data[ index ][ i++ ] = comp.getName() ;
          data[ index ][ i++ ] = new Boolean( comp.getHasIncomingConnections() );
          data[ index ][ i++ ] = new Boolean( comp.getHasOutgoingConnections() );
          data[ index ][ i++ ] = comp.getInputInstanceFile();
```

```
          data[ index ][ i++ ] = comp.getOutputInstanceFile();
          data[ index++ ][ i ] = comp.getSchema() ;
        }

      // Set up table
      myTable = new JTable( new AbstractTableModel() {
          public String getColumnName(int col) { return columnNames[col]; }
          public int getRowCount() { return data.length; }
          public int getColumnCount() { return columnNames.length; }
          public Object getValueAt(int row, int col) { return data[row][col]; }
          public boolean isCellEditable(int row, int col) { return false; }
          public Class getColumnClass(int c) { return getValueAt(0, c).getClass(); }

      } );

      // Set width
      for( index = 0 ; index < columnNames.length ; ++index )
        myTable.getColumnModel().getColumn( index ).setMinWidth( 80 );
      myTable.getColumnModel().getColumn( 5 ).setMinWidth( 400 );
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

}
```

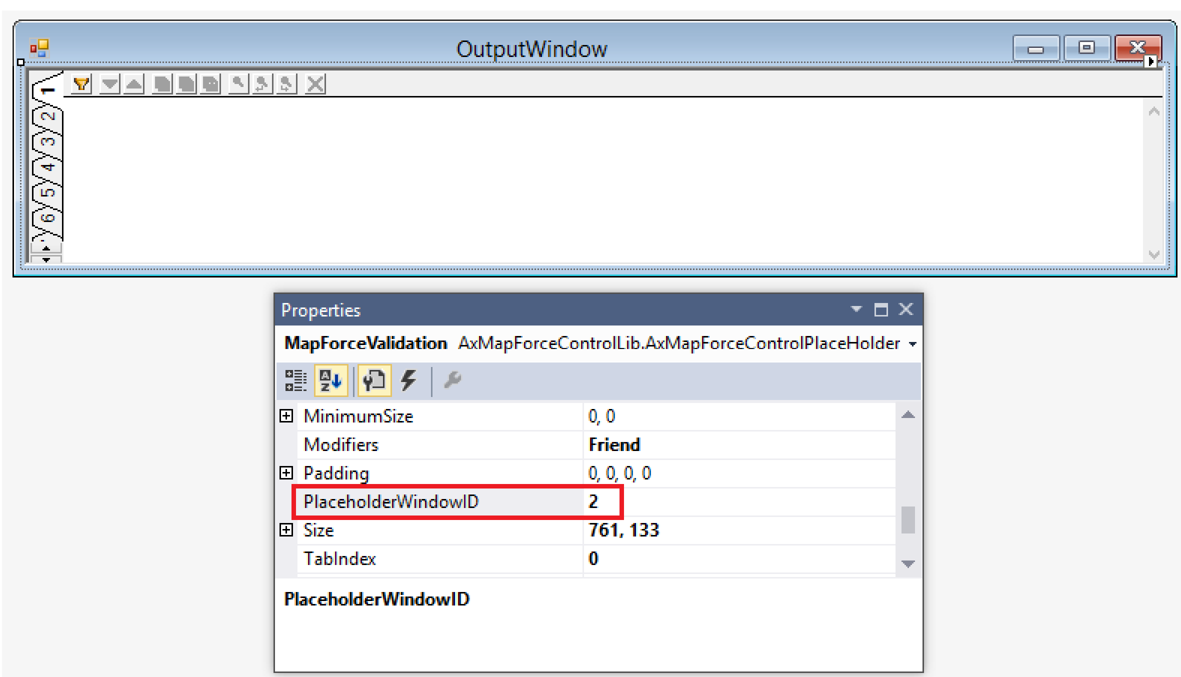
18.5.3 VB.NET

Source code which illustrates integration of MapForceControl into a VB.NET application can be found in the folder `<ApplicationFolder>\Examples\ActiveX\VB.NET` of your MapForce installation. The solution consists of three windows, as follows:

1. **MainWindow.vb** - the main document window, which also includes a basic application menu.
2. **LibraryWindow.vb** - the Library window. The contents of this window is populated by a Placeholder control which has the **PlaceholderWindowID** property set to 0 (this value instructs the control to display specifically the Library window).



3. **OutputWindow.vb** - the Messages (Output) window. The contents of this window is populated by a Placeholder control which has the **PlaceholderWindowID** property set to 2 (this value instructs the control to display specifically the Output window).



Before attempting to build and run this solution, note the following steps:

Step 1: Check the prerequisites

For the list of prerequisites, see Prerequisites 1828.

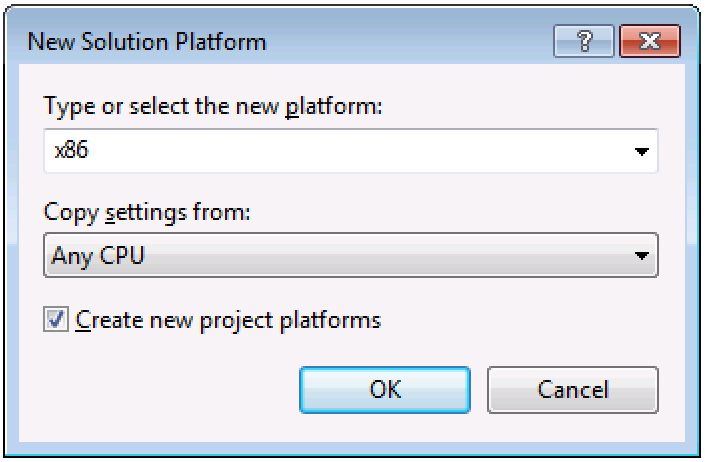
Step 2: Copy the sample to a directory where you have write permissions

To avoid running Visual Studio as an Administrator, copy the source code to a directory where you have write permissions, instead of running it from the default location.

Step 3: Set the build platform

- Create a build platform configuration that matches the platform under which you want to build (x86, x64). Here is how you can create the build configuration:

  a. Right-click the solution in Visual Studio, and select **Configuration Manager**.
  b. Under **Active solution platform**, select **New...** and then select the x86 or x64 configuration (in this example, **x86**).



You are now ready to build and run the solution in Visual Studio. Remember to build using the configuration that matches your target platform (x86, x64); otherwise, runtime errors might occur.

18.6 Command Reference

This section lists the names and identifiers of all menu commands that are available within MapForce. Every sub-section lists the commands from the corresponding top-level menu of MapForce. The command tables are organized as follows:

- The "Menu Item" column shows the command's menu text as it appears in MapForce, to make it easier for you to identify the functionality behind the command.
- The "Command Name" column specifies the string that can be used to get an icon with the same name from **ActiveX\Images** folder of the MapForce installation directory.
- The "ID" column shows the numeric identifier of the column that must be supplied as argument to methods which execute or query this command.

To execute a command, use the `MapForceControl.Exec` 1874 or the `MapForceControlDocument.Exec` 1880 methods. To query the status of a command, use the `MapForceControl.QueryStatus` 1875 or `MapForceControlDocument.QueryStatus` 1881 methods.

Depending on the edition of MapForce you have installed, some of these commands might not be supported.

18.6.1 "File" Menu

The "File" menu has the following commands:

Menu item	Command name	ID
New...	ID_FILE_NEW	57600
Open...	ID_FILE_OPEN	57601
Save	ID_FILE_SAVE	57603
Save As...	ID_FILE_SAVE_AS	57604
Save All	ID_FILE_SAVEALL	32377
Reload	IDC_FILE_RELOAD	32467
Close	ID_WINDOW_CLOSE	32453
Close All	ID_WINDOW_CLOSEALL	32454
Print...	ID_FILE_PRINT	57607
Print Preview	ID_FILE_PRINT_PREVIEW	57609
Print Setup...	ID_FILE_PRINT_SETUP	57606
Validate Mapping	ID_MAPPING_VALIDATE	32347
Mapping Settings	ID_MAPPING_SETTINGS	32396
Generate Code in Selected Language	ID_FILE_GENERATE_SELECTED_CODE	32362

Menu item	Command name	ID
XSLT 1.0	ID_FILE_GENERATEXSLT	32360
XSLT 2.0	ID_FILE_GENERATEXSLT2	32361
XQuery	ID_FILE_GENERATEXQUERY	32359
Java	ID_FILE_GENERATEJAVACODE	32358
C# (Sharp)	ID_FILE_GENERATECSCODE	32357
C++	ID_FILE_GENERATECPPCODE	32356
Compile to MapForce Server Execution File...	ID_FILE_CREATE_SERVER_EXECUTION_FILE	32517
Deploy to FlowForce Server...	ID_FILE_DEPLOY_MAPPING	32506
Generate Documentation...	ID_FILE_GENERATE_DOCUMENTATION	32468
Recent File	ID_FILE_MRU_FILE1	57616
Exit	ID_APP_EXIT	57665

18.6.2 "Edit" Menu

The "Edit" menu has the following commands:

Menu item	Command name	ID
Undo	ID_EDIT_UNDO	57643
Redo	ID_EDIT_REDO	57644
Find...	ID_EDIT_FIND	57636
Find Next	ID_EDIT_FINDNEXT	32349
Find Previous	ID_EDIT_FINDPREV	32350
Cut	ID_EDIT_CUT	57635
Copy	ID_EDIT_COPY	57634
Paste	ID_EDIT_PASTE	57637
Delete	ID_EDIT_CLEAR	57632
Select All	ID_EDIT_SELECT_ALL	57642

18.6.3 "Insert" Menu

The "Insert" menu has the following commands:

Menu item	Command name	ID
XML Schema/File...	ID_INSERT_XSD	32393
Database...	ID_INSERT_DATABASE	32389
EDI...	ID_INSERT_EDI	32390
Text File...	ID_INSERT_TXT	32392
Web Service Function...	ID_INSERT_WEBSERVICE_FUNCTION	32319
Excel 2007+ File...	ID_INSERT_EXCEL	32376
XBRL Document...	ID_INSERT_XBRL	32469
JSON Schema/File...	ID_INSERT_JSON	32531
Insert Input...	ID_FUNCTION_INSERT_INPUT	32383
Insert Output...	ID_FUNCTION_INSERT_OUTPUT	32402
Constant...	ID_INSERT_CONSTANT	32388
Variable...	ID_INSERT_VARIABLE	32500
Join	ID_INSERT_JOIN	32581
Sort: Nodes/Rows	ID_INSERT_SORT	32444
Filter: Nodes/Rows	ID_INSERT_FILTER	32391
SQL-WHERE/ORDER	ID_INSERT_SQLWHERE_CONDITION	32351
Value-Map	ID_INSERT_VALUEMAP	32354
IF-Else Condition	ID_INSRT_CONDITION	32394
Exception	ID_INSERT_EXCEPTION	32311

18.6.4 "Project" Menu

The "Project" menu has the following commands:

Menu item	Command name	ID
Reload Project	ID_PROJECT_RELOAD	32476

Menu item	Command name	ID
Close Project	ID_FILE_CLOSEPROJECT	32355
Save Project	ID_FILE_SAVEPROJECT	32378
Add Files to Project...	ID_PROJECT_ADDFILESTOPROJECT	32420
Add Active File to Project	ID_PROJECT_ADDACTIVEFILETOPROJECT	32419
Create Folder...	ID_PROJECT_CREATE_FOLDER	32310
Open Mapping	ID_PROJECT_OPEN_MAPPING	32307
Create Mapping for Operation...	ID_PROJECT_CREATE_MAPPING_FOR_OPERATION	32399
Add Mapping File for Operation...	ID_PROJECT_ADD_MAPPING	32309
Insert Web Service...	ID_PROJECT_INSERT_WEBSERVICE	32306
Open File in XMLSpy	ID_PROJECT_OPEN_IN_XMLSPY	32305
Generate Code for Entire Project	ID_PROJECT_GENERATE_ALL	32303
XSLT 1.0	ID_PROJECT_GENERATEXSLTCODE_ENTIRE	32408
XSLT 2.0	ID_PROJECT_GENERATEXSLT2CODE_ENTIRE	32409
XQuery	ID_PROJECT_GENERATEXQUERYCODE_ENTIRE	32410
Java	ID_PROJECT_GENERATEJAVACODE_ENTIRE	32411
C# (Sharp)	ID_PROJECT_GENERATECSCODE_ENTIRE	32412
C++	ID_PROJECT_GENERATECPPCODE_ENTIRE	32413
Properties	ID_PROJECT_PROPERTIES	32404
Recent Project	ID_FILE_MRU_PROJECT1	32364

18.6.5 "Component" Menu

The "Component" menu has the following commands:

Menu item	Command name	ID
Change Root Element...	ID_COMPONENT_CHANGEROOTELEMENT	32334
Edit Schema Definition in XMLSpy	ID_COMPONENT_EDIT_SCHEMA	32337

Menu item	Command name	ID
Edit FlexText Configuration	ID_COMPONENT_EDIT_MFT	32301
Add/Remove/Edit Database Objects...	ID_COMPONENT_SELECTTABLES	32346
Create Mapping to EDI X12 997	ID_COMPONENT_CREATE_MAPPING_TO_997	32483
Create Mapping to EDI X12 999	ID_COMPONENT_CREATE_MAPPING_TO_999	32484
Refresh	IDC_COMMAND_REFRESH_COMPONENT	32373
Add Duplicate Input Before	ID_COMPONENT_CREATE_DUPLICATE_ICON_BEFORE	32503
Add Duplicate Input After	ID_COMPONENT_CREATE_DUPLICATE_ICON	32335
Remove Duplicate	ID_COMPONENT_REMOVE_DUPLICATE_ICON	32339
Add Comment Before	ID_COMPONENT_ADD_COMMENT_BEFORE	32518
Add Comment After	ID_COMPONENT_ADD_COMMENT_AFTER	32519
Add Processing Instruction Before...	ID_COMPONENT_ADD_PI_BEFORE	32520
Add Processing Instruction After...	ID_COMPONENT_ADD_PI_AFTER	32521
Edit Processing Instruction Name...	ID_COMPONENT_EDIT_PI	32524
Delete Comment/Processing Instruction	ID_COMPONENT_REMOVE_COMMENT_PI	32522
Write Content as CDATA Section	ID_COMPONENT_TOGGLE_CDATA	32525
Database Table Actions	ID_POPUP_DATABASETABLEACTIONS	32400
Query Database...	ID_QUERY_DATABASE	32341
Align Tree Left	ID_COMPONENT_LEFTALIGNTREE	32338
Align Tree Right	ID_COMPONENT_RIGHTALIGNTREE	32340
Properties	ID_COMPONENT_PROPERTIES	32336

18.6.6 "Connection" Menu

The "Connection" menu has the following commands:

Menu item	Command name	ID
Auto Connect Matching Children	ID_CONNECTION_AUTOCONNECTCHILDREN	32342

Menu item	Command name	ID
Settings for Connect Matching Children	ID_CONNECTION_SETTINGS	32344
Connect Matching Children...	ID_CONNECTION_MAPCHILDELEMENTS	32343
Target Driven (Standard)	ID_POPUP_NORMALCONNECTION	32401
Copy-All (Copy Child Items)	ID_POPUP_NORMALWITHCHILDREN_CONNECTION	32460
Source Driven (Mixed Content)	ID_POPUP_ORDERBYSOURCECONNECTION	32403
Properties	ID_POPUP_CONNECTION_SETTINGS	32398

18.6.7 "Function" Menu

The "Function" menu has the following commands:

Menu item	Command name	ID
Create User-Defined Function...	ID_FUNCTION_CREATE_EMPTY	32380
Create User-Defined Function from Selection...	ID_FUNCTION_CREATE_FROM_SELECTION	32381
Function Settings	ID_FUNCTION_SETTINGS	32387
Remove Function	ID_FUNCTION_REMOVE	32385
Insert Input...	ID_FUNCTION_INSERT_INPUT	32383
Insert Output...	ID_FUNCTION_INSERT_OUTPUT	32402

18.6.8 "Output" Menu

The "Output" menu has the following commands:

Menu item	Command name	ID
XSLT 1.0	ID_SELECT_LANGUAGE_XSLT	32433
XSLT 2.0	ID_SELECT_LANGUAGE_XSLT2	32434
XQuery	ID_SELECT_LANGUAGE_XQUERY	32432
Java	ID_SELECT_LANGUAGE_JAVA	32431
C# (Sharp)	ID_SELECT_LANGUAGE_CSHARP	32430
C++	ID_SELECT_LANGUAGE_CPP	32429

Menu item	Command name	ID
Built-In Execution Engine	ID_SELECT_LANGUAGE_BUILTIN	32490
Validate Output File	ID_XML_VALIDATE	32458
Save Output File...	IDC_FILE_SAVEGENERATEDOUTPUT	32321
Save All Output Files...	IDC_FILE_SAVEALLGENERATEDOUTPUT	32374
Regenerate Output	ID_REGENERATE_PREVIEW_OUTPUT	32480
Run SQL-Script	ID_TRANSFORM_RUN_SQL	32442
Insert/Remove Bookmark	ID_TOGGLE_BOOKMARK	32317
Next Bookmark	ID_GOTONEXTBOOKMARK	32315
Previous Bookmark	ID_GOTOPREVBOOKMARK	32314
Remove All Bookmarks	ID_REMOVEALLBOOKMARKS	32313
Pretty-Print XML Text	ID_PRETTY_PRINT_OUTPUT	32363
Text View Settings	ID_TEXTVIEWSETTINGSDIALOG	32472

18.6.9 "Debug" Menu

The "Debug" menu has the following commands:

Menu item	Command name	ID
Start Debugging	ID_DEBUG_START	32540
Stop Debugging	ID_DEBUG_STOP	32541
Step Into	ID_DEBUG_STEP_INTO	32545
Step Over	ID_DEBUG_STEP_OVER	32551
Step Out	ID_DEBUG_STEP_OUT	32552
Minimal Step	ID_DEBUG_STEP_NEXT_TRACE	32554

18.6.10 "View" Menu

The "View" menu has the following commands:

Menu item	Command name	ID
Show Annotations	ID_SHOW_ANNOTATION	32435

Menu item	Command name	ID
Show Types	ID_SHOW_TYPES	32437
Show Library in Function Header	ID_VIEW_SHOWLIBRARYINFUNCTIONHEADER	32448
Show Tips	ID_SHOW_TIPS	32436
XBRL Display Options	ID_VIEW_XBRL_DISPLAY_OPTIONS	32473
Show Selected Component Connectors	ID_VIEW_AUTOHIGHLIGHTCOMPONENTCONNECTIONS	32443
Show Connectors from Source to Target	ID_VIEW_RECURSIVEAUTOHIGHLIGHT	32447
Zoom...	ID_VIEW_ZOOM	32451
Back	ID_CMD_BACK	32479
Forward	ID_CMD_FORWARD	32478
Status Bar	ID_VIEW_STATUS_BAR	59393
Library Window	ID_VIEW_LIBRARY_WINDOW	32445
Messages	ID_VIEW_VALIDATION_OUTPUT	32450
Overview	ID_VIEW_OVERVIEW_WINDOW	32446
Project Window	ID_VIEW_PROJECT_WINDOW	32302
Values	ID_DEBUG_VIEW_VALUES_WINDOW	32544
Context	ID_DEBUG_VIEW_CONTEXT_WINDOW	32546
Breakpoints	ID_DEBUG_VIEW_DEBUGPOINTS_WINDOW	32547

18.6.11 "Tools" Menu

The "Tools" menu has the following commands:

Menu item	Command name	ID
Global Resources	IDC_GLOBALRESOURCES	37401
<plugin not loaded>	IDC_GLOBALRESOURCES_SUBMENUENTRY1	37408
Create Reversed Mapping	ID_CREATE_REVERSED_MAPPING	32489
Customize...	IDC_APP_TOOLS_CUSTOMIZE	32959
Options...	ID_TOOLS_OPTIONS	32441

18.6.12 "Window" Menu

The "Window" menu has the following commands:

Menu item	Command name	ID
Cascade	ID_WINDOW_CASCADE	57650
Tile Horizontal	ID_WINDOW_TILE_HORZ	57651
Tile Vertical	ID_WINDOW_TILE_VERT	57652

18.6.13 "Help" Menu

The "Help" menu has the following commands:

Menu item	Command name	ID
Table of Contents...	IDC_HELP_CONTENTS	32966
Index...	IDC_HELP_INDEX	32967
Search...	IDC_HELP_SEARCH	32969
Software Activation...	IDC_ACTIVATION	32970
Order Form...	IDC_OPEN_ORDER_PAGE	32971
Registration...	IDC_REGISTRATION	32972
Check for Updates...	IDC_CHECK_FOR_UPDATES	32973
MapForce Product Comparison...	IDC_PRODUCT_COMPARISON	32955
Support Center...	IDC_OPEN_SUPPORT_PAGE	32961
FAQ on the Web...	IDC_OPEN_FAQ_PAGE	32962
Download Components and Free Tools...	IDC_OPEN_COMPONENTS_PAGE	32963
MapForce on the Internet..	IDC_OPEN_HOME_PAGE	32964
MapForce Training...	IDC_OPEN_TRAINING_PAGE	32965
About MapForce...	ID_APP_ABOUT	57664

18.7 Object Reference

Objects:
MapForceCommand[1867]
MapForceCommands[1869]
MapForceControl[1870]
MapForceControlDocument[1878]
MapForceControlPlaceHolder[1884]

To give access to standard MapForce functionality, objects of the **MapForce automation interface** can be accessed as well. See MapForceControl.Application[1871], MapForceControlDocument.Document[1879] and MapForceControlPlaceHolder.Project[1885] for more information.

18.7.1 MapForceCommand

Properties:
`ID`[1868]
`Label`[1868]
`Name`[1868]
`IsSeparator`[1868]
`ToolTip`[1869]
`StatusText`[1869]
`Accelerator`[1867]
`SubCommands`[1869]

Description:
A command object can be one of the following: an executable command, a command container (for example, a menu, submenu, or toolbar), or a menu separator. To determine what kind of information is stored in the current `Command` object, query its `ID`, `IsSeparator`, and `SubCommands` properties, as follows.

The Command object is...	When...
An executable command	• `ID` is greater than zero • `IsSeparator` is false • `SubCommands` is empty
A command container	• `ID` is zero • `IsSeparator` is false • `SubCommands` contains a collection of `Command` objects.
Separator	• `ID` is zero • `IsSeparator` is true

18.7.1.1 Accelerator

Property: `Accelerator` as string

Description:
Returns the accelerator key defined for the command. If the command has no accelerator key assigned, this property returns the empty string. The string representation of the accelerator key has the following format:

```
[ALT+][CTRL+][SHIFT+]key
```

Where `key` is converted using the Windows Platform SDK function `GetKeyNameText`.

18.7.1.2 ID

Property: `ID` as long

Description:
This property gets the unique identifier of the command. A command's ID is required to execute the command (using `Exec`1874) or query its status (using `QueryStatus`1875). If the command is a container for other commands (for example, a top-level menu), or a separator, the ID is 0.

18.7.1.3 IsSeparator

Property: `IsSeparator` as boolean

Description:
The property returns `true` if the command object is a menu separator; `false` otherwise. See also Command1867.

18.7.1.4 Label

Property: `Label` as string

Description:
This property gets the text of the command as it is displayed in the graphical user interface of MapForce. If the command is a separator, "Label" is an empty string. This property may also return an empty string for some toolbar commands that do not have any GUI text associated with them.

18.7.1.5 Name

Property: `Name` as string

Description:
This property gets the unique name of the command. This value can be used to get the icon file of the command, where it is available. The available icon files can be found in the folder **<ApplicationFolder>\Examples\ActiveX\Images** of your MapForce installation.

18.7.1.6 StatusText

Property: `Label` as string

Description:
The status text is the text shown in the status bar of MapForce when the command is selected. It applies only to command objects that are not separators or containers of other commands; otherwise, the property is an empty string.

18.7.1.7 SubCommands

Property: `SubCommands` as `Commands` 1869

Description:
The `SubCommands` property gets the collection of `Command` 1867 objects that are sub-commands of the current command. The property is applicable only to commands that are containers for other commands (menus, submenus, or toolbars). Such container commands have the `ID` set to 0, and the `IsSeparator` property set to `false`.

18.7.1.8 ToolTip

Property: `ToolTip` as string

Description:
This property gets the text that is shown as a tool-tip for each command. If the command does not have a tooltip text, the property returns an empty string.

18.7.2 MapForceCommands

Properties:
`Count` 1869
`Item` 1870

Description:
Collection of `Command` 1867 objects to get access to command labels and IDs of the MapForceControl. Those commands can be executed with the `Exec` 1874 method and their status can be queried with `QueryStatus` 1875.

18.7.2.1 Count

Property: `Count` as long

Description:
Number of `Command` 1867 objects on this level of the collection.

18.7.2.2 Item

Property: `Item` (n as long) as `Command`[1867]

Description:
Gets the command with the index n in this collection. Index is 1-based.

18.7.3 MapForceControl

Properties:
`IntegrationLevel`[1872]
`Appearance`[1871]
`Application`[1871]
`BorderStyle`[1871]
`CommandsList`[1871]
`EnableUserPrompts`[1872]
`MainMenu`[1872]
`Toolbars`[1873]

Methods:
`Open`[1874]
`Exec`[1874]
`QueryStatus`[1875]

Events:
`OnUpdateCmdUI`[1877]
`OnOpenedOrFocused`[1877]
`OnCloseEditingWindow`[1875]
`OnFileChangedAlert`[1876]
`OnContextChanged`[1876]
`OnDocumentOpened`[1876]
`OnValidationWindowUpdated`[1877]

This object is a complete ActiveX control and should only be visible if the MapForce library is used in the Application Level mode.

```
CLSID: A38637E9-5759-4456-A167-F01160CC22C1
ProgID: Altova.MapForceControl
```

18.7.3.1 Properties

The following properties are defined:

`IntegrationLevel`[1872]
`EnableUserPrompts`[1872]
`Appearance`[1871]
`BorderStyle`[1871]

Command related properties:

[CommandsList](1871)
[MainMenu](1872)
[Toolbars](1873)

Access to MapForceAPI:
[Application](1871)

18.7.3.1.1 Appearance

Property: `Appearance` as short

Dispatch Id: *-520*

Description:
A value not equal to `0` displays a client edge around the control. Default value is `0`.

18.7.3.1.2 Application

Property: `Application` as Application

Dispatch Id: *1*

Description:
The `Application` property gives access to the `Application` object of the complete MapForce automation server API. The property is read-only.

18.7.3.1.3 BorderStyle

Property: `BorderStyle` as short

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

18.7.3.1.4 CommandsList

Property: `CommandList` as `Commands` [1869] (read-only)

Dispatch Id: *1004*

Description:
This property returns a flat list of all commands defined available with MapForceControl. To get commands organized according to their menu structure, use `MainMenu` [1872]. To get toolbar commands, use `Toolbars` [1873].

```
public void GetAllMapForceCommands()
{
    // Get all commands from the MapForce ActiveX control assigned to the current form
    MapForceControlLib.MapForceCommands commands = this.axMapForceControl1.CommandList;
    // Iterate through all commands
    for (int i = 0; i < commands.Count; i++)
    {
      // Get each command by index and output it to the console
      MapForceControlLib.MapForceCommand cmd = axMapForceControl1.CommandList[i];
      Console.WriteLine("{0} {1} {2}", cmd.ID, cmd.Name, cmd.Label.Replace("&", ""));
    }
}
```

C# example

18.7.3.1.5 EnableUserPrompts

Property: `EnableUserPrompts` as boolean

Dispatch Id: *1006*

Description:
Setting this property to *false*, disables user prompts in the control. The default value is *true*.

18.7.3.1.6 IntegrationLevel

Property: `IntegrationLevel` as `ICActiveXIntegrationLevel` 1886

Dispatch Id: *1000*

Description:
The `IntegrationLevel` property determines the operation mode of the control. See also Integration at Application Level 1832 and Integration at Document Level 1835 for more information.

Note: It is important to set this property immediately after the creation of the `MapForceControl` object.

18.7.3.1.7 MainMenu

Property: `MainMenu as` Command 1867 (read-only)

Dispatch Id: *1003*

Description:
This property provides information about the structure and commands available in the MapForceControl main menu, as a `Command` object. The `Command` object contains all available submenus of MapForce (for example "File", "Edit", "View" etc.). To access the submenu objects, use the `SubCommands` property of the `MainMenu` property. Each submenu is also a `Command` object. For each submenu, you can then further iterate through their

SubCommands property in order to get their corresponding child commands and separators (this technique may be used, for example, to create the application menu programmatically). Note that some menu commands act as containers ("parents") for other menu commands, in which case they also have a SubCommands property. To get the structure of all menu commands programmatically, you will need a recursive function, as illustrated for C# in Retrieving Command Information 1842.

```
public void GetMapForceMenus()
{
    // Get the main menu from the MapForce ActiveX control assigned to the current form
    MapForceControlLib.MapForceCommand mainMenu = this.axMapForceControl1.MainMenu;

    // Loop through entries of the main menu (e.g. File, Edit, etc.)
    for (int i = 0; i < mainMenu.SubCommands.Count; i++)
    {
      MapForceControlLib.MapForceCommand menu = mainMenu.SubCommands[i];
      Console.WriteLine("{0} menu has {1} children items (including separators)",
menu.Label.Replace("&", ""), menu.SubCommands.Count);
    }
}
```

C# example

18.7.3.1.8 Toolbars

Property: Toolbars as Commands 1869 (read-only)

Dispatch Id: *1005*

Description:
This property provides information about the structure of MapForceControl toolbars, as a Command object. The Command object contains all available toolbars of MapForce. To access the toolbars, use the SubCommands property of the Toolbars property. Each toolbar is also a Command object. For each toolbar, you can then further iterate through their SubCommands property in order to get their commands (this technique may be used, for example, to create the application's toolbars programmatically).

```
public void GetMapForceToolbars()
{
    // Get the application toolbars from the MapForce ActiveX control assigned to the
current form
    MapForceControlLib.MapForceCommands toolbars = this.axMapForceControl1.Toolbars;

    // Iterate through all toolbars
    for (int i = 0; i < toolbars.Count; i++)
    {
      MapForceControlLib.MapForceCommand toolbar = toolbars[i];
      Console.WriteLine();
      Console.WriteLine("The toolbar \"{0}\" has the following commands:",
toolbar.Label);

      // Iterate through all commands of this toolbar
```

```
      for (int j = 0; j < toolbar.SubCommands.Count; j++)
      {
         MapForceControlLib.MapForceCommand cmd = toolbar.SubCommands[j];
         // Output only command objects that are not separators
         if (!cmd.IsSeparator)
         {
            Console.WriteLine("{0}, {1}, {2}", cmd.ID, cmd.Name, cmd.Label.Replace("&",
""));
         }
      }
    }
}
```

C# example

18.7.3.2 Methods

The following methods are defined:

[Open](1874)
[Exec](1874)
[QueryStatus](1875)

18.7.3.2.1 Exec

Method: `Exec` (`nCmdID` as long) as boolean

Dispatch Id: *6*

Description:
This method calls the MapForce command with the ID `nCmdID`. If the command can be executed, the method returns `true`. To get a list of all available commands, use `CommandsList` [1871]. To retrieve the status of any command, use `QueryStatus` [1875].

18.7.3.2.2 Open

Method: `Open` (`strFilePath` as string) as boolean

Dispatch Id: *5*

Description:
The result of the method depends on the extension passed in the argument `strFilePath`. If the file extension is `.sps`, a new document is opened. If the file extension is `.svp`, the corresponding project is opened. If a different file extension is passed into the method, the control tries to load the file as a new component into the active document.

Do not use this method to load documents or projects when using the control in document-level integration mode. Instead, use `MapForceControlDocument.Open` [1881] and `MapForceControlPlaceHolder.OpenProject` [1885].

18.7.3.2.3 QueryStatus

Method: `QueryStatus` (`nCmdID` as long) as long

Dispatch Id: *7*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if `QueryStatus` returns `0` the command ID is not recognized as a valid MapForce command. If `QueryStatus` returns a value of `1` or `5`, the command is disabled.

18.7.3.3 Events

The MapForceControl ActiveX control provides the following connection point events:

`OnUpdateCmdUI` 1877
`OnOpenedOrFocused` 1877
`OnCloseEditingWindow` 1875
`OnFileChangedAlert` 1876
`OnContextChanged` 1876

`OnDocumentOpened` 1876
`OnValidationWindowUpdated` 1877

18.7.3.3.1 OnCloseEditingWindow

Event: `OnCloseEditingWindow` (i_strFilePath as String) as boolean

Dispatch Id: *1002*

Description:
This event is triggered when MapForce needs to close an already open document. As an answer to this event, clients should close the editor window associated with *i_strFilePath*. Returning *true* from this event indicates that the client has closed the document. Clients can return *false* if no specific handling is required and MapForceControl should try to close the editor and destroy the associated document control.

18.7.3.3.2 OnContextChanged

Event: `OnContextChanged` (`i_strContextName` as String, `i_bActive` as bool) as bool

Dispatch Id: *1004*

Description:
This event is not used in MapForce

18.7.3.3.3 OnDocumentOpened

Event: `OnDocumentOpened` (`objDocument` as Document)

Dispatch Id: *1*

Description:
This event is triggered whenever a document is opened. The argument `objDocument` is a `Document` object from the MapForce automation interface and can be used to query for more details about the document, or perform additional operations. When integrating on document-level, it is often better to use the event `MapForceControlDocument.OnDocumentOpened` 1883 instead.

18.7.3.3.4 OnFileChangedAlert

Event: `OnFileChangedAlert` (`i_strFilePath` as String) as bool

Dispatch Id: *1001*

Description:
This event is triggered when a file loaded with MapForceControl is changed on the hard disk by another application. Clients should return true, if they handled the event, or false, if MapForce should handle it in its customary way, i.e. prompting the user for reload.

18.7.3.3.5 OnLicenseProblem

Event: `OnLicenseProblem` (`i_strLicenseProblemText` as String)

Dispatch Id: *1005*

Description:
This event is triggered when MapForceControl detects that no valid license is available for this control. In case of restricted user licenses this can happen some time after the control has been initialized. Integrators should use this event to disable access to this control's functionality. After returning from this event, the control will block access to its functionality (e.g. show empty windows in its controls and return errors on requests).

18.7.3.3.6 OnOpenedOrFocused

Event: `OnOpenedOrFocused` (`i_strFilePath` as String, `i_bOpenWithThisControl` as bool)

Dispatch Id: *1000*

Description:
When integrating at application level, this event informs clients that a document has been opened, or made active by MapForce.

When integrating at document level, this event instructs the client to open the file `i_strFilePath` in a document window. If the file is already open, the corresponding document window should be made the active window.

if `i_bOpenWithThisControl` is `true`, the document must be opened with MapForceControl, since internal access is required. Otherwise, the file can be opened with different editors.

18.7.3.3.7 OnToolWindowUpdated

Event: `OnToolWindowUpdated(pToolWnd` as long `)`

Dispatch Id: *1006*

Description:
This event is triggered when the tool window is updated.

18.7.3.3.8 OnUpdateCmdUI

Event: `OnUpdateCmdUI ()`

Dispatch Id: *1003*

Description:
Called frequently to give integrators a good opportunity to check status of MapForce commands using `MapForceControl.QueryStatus` 1875. Do not perform long operations in this callback.

18.7.3.3.9 OnValidationWindowUpdated

Event: `OnValidationWindowUpdated` ()

Dispatch Id: *3*

Description:
This event is triggered whenever the validation output window is updated with new information.

18.7.4 MapForceControlDocument

Properties:
Appearance 1879
BorderStyle 1879
Document 1879
IsModified 1879
Path 1879
ReadOnly 1880

Methods:
Exec 1880
New 1880
Open 1881
QueryStatus 1881
Reload 1881
Save 1881
SaveAs 1882

Events:
OnDocumentOpened 1883
OnDocumentClosed 1882
OnModifiedFlagChanged 1883
OnContextChanged 1882
OnFileChangedAlert 1883
OnActivate 1882

If the MapForceControl is integrated in the Document Level mode each document is displayed in an own object of type `MapForceControlDocument`. The `MapForceControlDocument` contains only one document at the time but can be reused to display different files one after another.

This object is a complete ActiveX control.

```
CLSID: DFBB0871-DAFE-4502-BB66-08CEB7DF5255
ProgID: Altova.MapForceControlDocument
```

18.7.4.1 Properties

The following properties are defined:

ReadOnly 1880
IsModified 1879
Path 1879
Appearance 1879
BorderStyle 1879

Access to MapForceAPI:
Document 1879

18.7.4.1.1 Appearance

Property: `Appearance` as short

Dispatch Id: *-520*

Description:
A value not equal to `0` displays a client edge around the document control. Default value is `0`.

18.7.4.1.2 BorderStyle

Property: `BorderStyle` as short

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

18.7.4.1.3 Document

Property: `Document` as Document

Dispatch Id: *1*

Description:
The `Document` property gives access to the `Document` object of the MapForce automation server API. This interface provides additional functionality which can be used with the document loaded in the control. The property is read-only.

18.7.4.1.4 IsModified

Property: `IsModified` as boolean (read-only)

Dispatch Id: *1006*

Description:
`IsModified` is *true* if the document content has changed since the last open, reload or save operation. It is *false*, otherwise.

18.7.4.1.5 Path

Property: `Path` as string

Dispatch Id: *1005*

Description:
Sets or gets the full path name of the document loaded into the control.

18.7.4.1.6 ReadOnly

Property: `ReadOnly` as boolean

Dispatch Id: *1007*

Description:
Using this property you can turn on and off the read-only mode of the document. If `ReadOnly` is `true` it is not possible to do any modifications.

18.7.4.2 Methods

The following methods are defined:

Document handling:
`New` [1880]
`Open` [1881]
`Reload` [1881]
`Save` [1881]
`SaveAs` [1882]

Command Handling:
`Exec` [1880]
`QueryStatus` [1881]

18.7.4.2.1 Exec

Method: `Exec` (`nCmdID` as long) as boolean

Dispatch Id: *8*

Description:
`Exec` calls the MapForce command with the ID `nCmdID`. If the command can be executed, the method returns `true`. This method should be called only if there is currently an active document available in the application.

To get commands organized according to their menu structure, use the `MainMenu` [1872] property of MapForceControl. To get toolbar commands, use the `Toolbars` [1873] property of the MapForceControl.

18.7.4.2.2 New

Method: `New` () as boolean

Dispatch Id: *1000*

Description:
This method initializes a new document inside the control.

18.7.4.2.3 Open

Method: `Open` (`strFileName` as string) as boolean

Dispatch Id: *1001*

Description:
`Open` loads the file `strFileName` as the new document into the control.

18.7.4.2.4 QueryStatus

Method: `QueryStatus` (`nCmdID` as long) as long

Dispatch Id: *9*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if `QueryStatus` returns `0` the command ID is not recognized as a valid MapForce command. If `QueryStatus` returns a value of `1` or `5` the command is disabled. The client should call the `QueryStatus` method of the document control if there is currently an active document available in the application.

18.7.4.2.5 Reload

Method: `Reload` () as boolean

Dispatch Id: *1002*

Description:
`Reload` updates the document content from the file system.

18.7.4.2.6 Save

Method: `Save` () as boolean

Dispatch Id: *1003*

Description:
`Save` saves the current document at the location `Path` [1879].

18.7.4.2.7 SaveAs

Method: `SaveAs` (`strFileName` as string) as boolean

Dispatch Id: *1004*

Description:
`SaveAs` sets `Path` [1879] to *strFileName* and then saves the document to this location.

18.7.4.3 Events

The MapForceControlDocument ActiveX control provides following connection point events:

`OnDocumentOpened` [1883]
`OnDocumentClosed` [1882]
`OnModifiedFlagChanged` [1883]
`OnContextChanged` [1882]
`OnFileChangedAlert` [1883]
`OnActivate` [1882]
`OnSetEditorTitle` [1884]

18.7.4.3.1 OnActivate

Event: `OnActivate` ()

Dispatch Id: *1005*

Description:
This event is triggered when the document control is activated, has the focus, and is ready for user input.

18.7.4.3.2 OnContextChanged

Event: `OnContextChanged` (`i_strContextName` as String, `i_bActive` as bool) as bool

Dispatch Id: *1004*

Description: None

18.7.4.3.3 OnDocumentClosed

Event: `OnDocumentClosed` (`objDocument` as Document)

Dispatch Id: *1001*

Description:
This event is triggered whenever the document loaded into this control is closed. The argument `objDocument` is a `Document` object from the MapForce automation interface and should be used with care.

18.7.4.3.4 OnDocumentOpened

Event: `OnDocumentOpened` (`objDocument` as Document)

Dispatch Id: *1000*

Description:
This event is triggered whenever a document is opened in this control. The argument `objDocument` is a `Document` object from the MapForce automation interface, and can be used to query for more details about the document, or perform additional operations.

18.7.4.3.5 OnDocumentSaveAs

Event: `OnContextDocumentSaveAs` (`i_strFileName` as String)

Dispatch Id: *1007*

Description:
This event is triggered when this document gets internally saved under a new name.

18.7.4.3.6 OnFileChangedAlert

Event: `OnFileChangedAlert` () as bool

Dispatch Id: *1003*

Description:
This event is triggered when the file loaded into this document control is changed on the hard disk by another application. Clients should return true, if they handled the event, or false, if MapForce should handle it in its customary way, i.e. prompting the user for reload.

18.7.4.3.7 OnModifiedFlagChanged

Event: `OnModifiedFlagChanged` (`i_bIsModified` as boolean)

Dispatch Id: *1002*

Description:
This event gets triggered whenever the document changes between modified and unmodified state. The parameter *i_bIsModifed* is *true* if the document contents differs from the original content, and *false*, otherwise.

18.7.4.3.8 OnSetEditorTitle

Event: OnSetEditorTitle ()

Dispatch Id: *1006*

Description:
This event is being raised when the contained document is being internally renamed.

18.7.5 MapForceControlPlaceHolder

Properties available for all kinds of placeholder windows:
`PlaceholderWindowID` [1885]

Properties for project placeholder window:
`Project` [1885]

Methods for project placeholder window:
`OpenProject` [1885]
`CloseProject` [1885]

The `MapForceControlPlaceHolder` control is used to show the additional MapForce windows like Overview, Library or Project window. It is used like any other ActiveX control and can be placed anywhere in the client application.

```
CLSID: FDEC3B04-05F2-427d-988C-F03A85DE53C2
ProgID: Altova.MapForceControlPlaceHolder
```

18.7.5.1 Properties

The following properties are defined:

`PlaceholderWindowID` [1885]

Access to MapForceAPI:
`Project` [1885]

18.7.5.1.1 Label

Property: `Label` as String (read-only)

Dispatch Id: *1001*

Description:
This property gives access to the title of the placeholder. The property is read-only.

18.7.5.1.2 PlaceholderWindowID

Property: `PlaceholderWindowID` as `MapForceControlPlaceholderWindow` [1887]

Dispatch Id: *1*

Description:
This property specifies which MapForce window should be displayed in the client area of the control. The `PlaceholderWindowID` can be set at any time to any valid value of the `MapForceControlPlaceholderWindow` [1887] enumeration. The control changes its state immediately and shows the new MapForce window.

18.7.5.1.3 Project

Property: `Project` as Project (read-only)

Dispatch Id: *2*

Description:
The `Project` property gives access to the `Project` object of the MapForce automation server API. This interface provides additional functionality which can be used with the project loaded into the control. The property will return a valid project interface only if the placeholder window has `PlaceholderWindowID` [1885] with a value of `MapForceXProjectWindow (=3)`. The property is read-only.

18.7.5.2 Methods

The following method is defined:

`OpenProject` [1885]
`CloseProject` [1885]

18.7.5.2.1 OpenProject

Method: `OpenProject` (`strFileName` as `string`) as `boolean`
Dispatch Id: *3*
Description:
`OpenProject` loads the file `strFileName` as the new project into the control. The method will fail if the placeholder window has a **`PlaceholderWindowID`** [1885] different to `MapForceXProjectWindow (=3)`.

18.7.5.2.2 CloseProject

Method: `CloseProject ()`

Dispatch Id: *4*

Description:
`CloseProject` closes the project loaded by the control. The method will fail if the placeholder window has a `PlaceholderWindowID` 1885 different to `MapForceXProjectWindow (=3)`.

18.7.5.3 Events

The MapForceControlPlaceholder ActiveX control provides following connection point events:

`OnModifiedFlagChanged` 1886

18.7.5.3.1 OnModifiedFlagChanged

Event: `OnModifiedFlagChanged` (`i_bIsModified` as boolean)

Dispatch Id: *1*

Description:
This event gets triggered only for placeholder controls with a `PlaceholderWindowID` 1885 of `MapForceXProjectWindow (=3)`. The event is fired whenever the project content changes between modified and unmodified state. The parameter *i_bIsModifed* is *true* if the project contents differs from the original content, and *false*, otherwise.

18.7.5.3.2 OnSetLabel

Event: `OnSetLabel(i_strNewLabel as` string)

Dispatch Id: *1000*

Description:
Raised when the title of the placeholder window is changed.

18.7.6 Enumerations

The following enumerations are defined:

`ICActiveXIntegrationLevel` 1886
`MapForceControlPlaceholderWindow` 1887

18.7.6.1 ICActiveXIntegrationLevel

Possible values for the IntegrationLevel 1872 property of the MapForceControl.

```
ICActiveXIntegrationOnApplicationLevel  = 0
ICActiveXIntegrationOnDocumentLevel     = 1
```

18.7.6.2 MapForceControlPlaceholderWindow

This enumeration contains the list of the supported additional MapForce windows.

```
MapForceXNoWindow                 = -1
MapForceXLibraryWindow            = 0
MapForceXOverviewWindow           = 1
MapForceXValidationWindow         = 2
MapForceXProjectWindow            = 3
MapForceXDebuggerValuesWindow     = 4
MapForceXDebuggerContextWindow    = 5
MapForceXDebuggerPointsWindow     = 6
```

19 Appendices

These appendices contain technical information about MapForce, its technical aspects and licensing. It also provides the list of key terms specific to MapForce and MapForce-related products. The section is organized into the following subsections:

19.1 Support Notes

MapForce® is a 32/64-bit Windows application that runs on the following operating systems:

- Windows 7 SP1 with Platform Update, Windows 8, Windows 10, Windows 11
- Windows Server 2008 R2 SP1 with Platform Update or newer

64-bit support is available for the Enterprise and Professional editions.

MapForce is optionally available as a plug-in to the following integrated development environments:

- Visual Studio 2012/2013/2015/2017/2019/2022. See MapForce Plug-in for Visual Studio [1426].
- Eclipse 2022-09, 2022-06, 2022-03, 2021-12. See MapForce Plug-in for Eclipse [1429].

MapForce can integrate with Microsoft Office products:

- It can map data to or from Access databases. For supported versions, see Databases and MapForce [161].
- It can generate mapping documentation in Word 2000 or later versions. See Generating and Customizing Mapping Documentation [1340].
- It can map data to or from Excel 2007 or later spreadsheets. See Microsoft OOXML Excel 2007+ [715].

19.1.1 Supported Sources and Targets

When you change the transformation language of a MapForce mapping, certain features may not be supported for that specific language. The following table summarizes the compatibility of mapping formats and transformation languages in **MapForce Enterprise Edition**.

Remarks:

- *Built-in* means that you can execute the mapping by clicking the **Output** tab in MapForce or run it with MapForce Server.

Mapping format		XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
XML[1]		●	●	●	●	●	●	●	●
CSV and text						●	●	●	●
Text, via MapForce FlexText [658]						●	●	●	●
JSON							●	●	●
XBRL[2]						●	●	●	●
Excel 2007	As data source		●	●			●	●	●

Mapping format		XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
and later[3]	As data target (generate new file)						●	●	●
	As data target (update existing file)								●
EDI	ASC X12					●	●	●	●
	HIPAA X12					●	●	●	●
	HL7 version 2					●	●	●	●
	HL7 version 3	●	●	●	●	●	●	●	●
	IATA PADIS					●	●	●	●
	NCPDP SCRIPT EDIFACT format					●	●	●	●
	NCPDP SCRIPT XML format	●	●	●	●	●	●	●	●
	SAP IDoc					●	●	●	●
	SWIFT MT								●
	SWIFT MX / ISO 20022	●	●	●	●	●	●	●	●
	TRADACOMS					●	●	●	●
	UN/EDIFACT					●	●	●	●

Mapping format		XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
Databases[4]	ADO					●	●		●
	ADO.NET						●		●
	JDBC							●	●
	Native SQLite								●
	Native PostgreSQL								●
	ODBC					●	●	●	●
Web Services[5]	REST-Style								●
	SOAP						●	●	●
Protocol Buffers									●
Raw binary files									●

Footnotes:

1. XML with digital signatures processing is supported only in BUILT-IN.
2. XBRL with digital signatures processing is supported only in BUILT-IN.
3. This refers to Excel 2007 or later files in *Office Open XML* format (.xlsx). Note that the *Strict Open XML Spreadsheet* format also uses the .xlsx extension but is supported only as a data source (not as a data target). For more information, see Microsoft OOXML Excel 2007+ [715].
4. Limitations apply depending on the database type and the target environment. For more information, see Database mappings in various execution environments [162].
5. This refers to calling Web Services, not generating them from mappings. For SOAP Web Services, note that client certificates, preemptive authentication and WS-Security are not supported in the generated C# and Java code. For more information, see Calling Web Services [586].

19.1.2 Supported Features in Generated Code

The following table lists the features relevant to code generation and the extent of support in each language in **MapForce Enterprise Edition**.

Feature	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
Supply parameters to the mapping [833]	●	●	●	●	●	●	●	●

Feature	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
Supply the input file names dynamically from the mapping 1313	●	●	●	●	●	●	●	●
Supply wildcard file names as mapping input 1313 [1]		●	●	●	●	●	●	●
Generate the output file names dynamically from the mapping 1313		●	●		●	●	●	●
Return string values from the mapping 843	●	●	●		●	●	●	●
Variables 847		●	●	●	●	●	●	●
Sort components 872		●	●	●				●
Grouping functions 894		●	●		●	●	●	●
Filters 878	●	●	●	●	●	●	●	●
Join components 860								●
Value-Map components 884	●	●	●	●	●	●	●	●
Defaults and node functions 912								●
Mapping exceptions 900		●	●	●	●	●	●	●
String parsing and serialization 1324 [2]								●
Dynamic node names 1255		●	●	●	●	●	●	●
XML and XBRL digital signatures 139								●

Feature	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
Database bulk inserts 319								●
Database SQL SELECT without input parameters 357					●	●	●	●
Database SQL SELECT with input parameters 357								●
Database stored procedures 390								●
Database exception handling 415 [3]					●	●	●	●
Database tracing and error logging 427								●
Quick (non-standard) conversion of EDI to XML 458								●
Generate SOAP Web Services 1204						●	●	
Generate MapForce Server execution files 1378								●
Deploy mappings to FlowForce Server 1381								●
Read data from binary files 827								●
Write data to binary files 827								●

Footnotes:

1. XSLT 2.0, XSLT 3.0, and XQuery use the `fn:collection` function. The implementation in the Altova XSLT 2.0, XSLT 3.0, and XQuery engines resolves wildcards. Other engines may behave differently.
2. For JSON, parsing and serialization are additionally supported in Java and C#.
3. Database exception handling is possible when the mapping language is supported by the currently connected database driver, as indicated in the previous table.

19.2 Engine Information

This section contains information about implementation-specific features of the Altova XML Validator, Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery Engine.

19.2.1 XSLT and XQuery Engine Information

The XSLT and XQuery engines of MapForce follow the W3C specifications closely and are therefore stricter than previous Altova engines—such as those in previous versions of XMLSpy. As a result, minor errors that were ignored by previous engines are now flagged as errors by MapForce.

For example:

- It is a type error (`err:XPTY0018`) if the result of a path operator contains both nodes and non-nodes.
- It is a type error (`err:XPTY0019`) if `E1` in a path expression `E1/E2` does not evaluate to a sequence of nodes.

If you encounter this kind of error, modify either the XSLT/XQuery document or the instance document as appropriate.

This section describes implementation-specific features of the engines, organized by specification:

19.2.1.1 XSLT 1.0

The XSLT 1.0 Engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999. Note the following information about the implementation.

Notes about the implementation

When the `method` attribute of `xsl:output` is set to HTML, or if HTML output is selected by default, then special characters in the XML or XSLT file are inserted in the HTML document as HTML character references in the output. For instance, the character U+00A0 (the hexadecimal character reference for a non-breaking space) is inserted in the HTML code either as a character reference (` ` or ` `) or as an entity reference, ` ` .

19.2.1.2 XSLT 2.0

This section:

Conformance

The XSLT 2.0 engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XSLT 2.0 Recommendation of 23 January 2007 and XPath 2.0 Recommendation of 14 December 2010.

Backwards Compatibility

The XSLT 2.0 engine is backwards compatible. The only time the backwards compatibility of the XSLT 2.0 engine comes into effect is when using the XSLT 2.0 engine to process an XSLT 1.0 stylesheet. Note that there could be differences in the outputs produced by the XSLT 1.0 Engine and the backwards-compatible XSLT 2.0 engine.

Namespaces

Your XSLT 2.0 stylesheet should declare the following namespaces in order for you to be able to use the type constructors and functions available in XSLT 2.0. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
XPath 2.0 functions	`fn:`	`http://www.w3.org/2005/xpath-functions`

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   ...
</xsl:stylesheet>
```

The following points should be noted:

- The XSLT 2.0 engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.
- When using type constructors and types from the XML Schema namespace, the prefix used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).
- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string('Hello')`, the expression evaluates as `fn:string('Hello')`—not as `xs:string('Hello')`.

Schema-awareness

The XSLT 2.0 engine is schema-aware. So you can use user-defined schema types and the `xsl:validate` instruction.

Implementation-specific behavior

Given below is a description of how the XSLT 2.0 engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

`xsl:result-document`
Additionally supported encodings are (the Altova-specific): `x-base16tobinary` and `x-base64tobinary`.

`function-available`
The function tests for the availability of in-scope functions (XSLT, XPath, and extension functions).

`unparsed-text`
The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol. Additionally supported encodings are (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

`unparsed-text-available`
The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol. Additionally supported encodings are (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

Note: The following encoding values, which were implemented in earlier versions of RaptorXML's predecessor product, AltovaXML, are now deprecated: `base16tobinary`, `base64tobinary`, `binarytobase16` and `binarytobase64`.

19.2.1.3 XQuery 1.0

This section:

Conformance

The XQuery 1.0 Engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XQuery 1.0 Recommendation of 14 December 2010. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the XQuery 1.0 Engine implements these features.

Schema awareness

The XQuery 1.0 Engine is **schema-aware**.

Encoding

The UTF-8 and UTF-16 character encodings are supported.

Namespaces

The following namespace URIs and their associated bindings are pre-defined.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
Schema instance	`xsi:`	`http://www.w3.org/2001/XMLSchema-instance`
Built-in functions	`fn:`	`http://www.w3.org/2005/xpath-functions`
Local functions	`local:`	`http://www.w3.org/2005/xquery-local-functions`

The following points should be noted:

- The XQuery 1.0 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.
- Since the built-in functions namespace listed above is the default functions namespace in XQuery, the `fn:` prefix does not need to be used when built-in functions are invoked (for example, `string("Hello")` will call the `fn:string` function). However, the prefix `fn:` can be used to call a built-in function without having to declare the namespace in the query prolog (for example: `fn:string("Hello")`).
- You can change the default functions namespace by declaring the `default function namespace` expression in the query prolog.
- When using types from the XML Schema namespace, the prefix `xs:` may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: `xs:date` and `xs:yearMonthDuration`.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Note that the `untypedAtomic`, `dayTimeDuration`, and `yearMonthDuration` datatypes have been moved, with the CRs of 23 January 2007, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is reported. Note, however, that some functions have the same name as schema datatypes, e.g. `fn:string` and `fn:boolean`. (Both `xs:string` and `xs:boolean` are defined.) The namespace prefix determines whether the function or type constructor is used.

XML source document and validation

XML documents used in executing an XQuery document with the XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

Static and dynamic type checking

The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression `xs:string("1") + 1` returns an error because the addition operation cannot be carried out on an operand of type `xs:string`.

Library Modules

Library modules store functions and variables so they can be reused. The XQuery 1.0 Engine supports modules that are stored in **a single external XQuery file**. Such a module file must contain a `module` declaration in its prolog, which associates a target namespace. Here is an example module:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the `import module` statement in the query prolog. The `import module` statement only imports functions and variables declared directly in the library module file. As follows:

```
import module namespace modlib = "urn:module-library" at "modulefilename.xq";
if         ($modlib:company = "Altova")
then        modlib:webaddress()
else        error("No match found.")
```

External functions

External functions are not supported, i.e. in those expressions using the `external` keyword, as in:

```
declare function hoo($param as xs:integer) as xs:string external;
```

Collations

The default collation is the Unicode-codepoint collation, which compares strings on the basis of their Unicode codepoint. Other supported collations are the ICU collations listed here 1899. To use a specific collation, supply its URI as given in the list of supported collations 1899. Any string comparisons, including for the `fn:max` and `fn:min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Precision of numeric types

- The `xs:integer` datatype is arbitrary-precision, i.e. it can represent any number of digits.
- The `xs:decimal` datatype has a limit of 20 digits after the decimal point.
- The `xs:float` and `xs:double` datatypes have limited-precision of 15 digits.

XQuery Instructions Support

The `Pragma` instruction is not supported. If encountered, it is ignored and the fallback expression is evaluated.

19.2.2 XSLT and XPath/XQuery Functions

This section lists Altova extension functions and other extension functions that can be used in XPath and/or XQuery expressions. Altova extension functions can be used with Altova's XSLT and XQuery engines, and provide functionality additional to that available in the function libraries defined in the W3C standards.

General points

The following general points should be noted:

- Functions from the core function libraries defined in the W3C specifications can be called without a prefix. That's because the XSLT and XQuery engines read non-prefixed functions as belonging to a default functions namespace which is that specified in the XPath/XQuery functions specifications `http://www.w3.org/2005/xpath-functions`. If this namespace is explicitly declared in an XSLT or XQuery document, the prefix used in the namespace declaration can also optionally be used on function names.
- In general, if a function expects a sequence of one item as an argument, and a sequence of more than one item is submitted, then an error is returned.
- All string comparisons are done using the Unicode codepoint collation.
- Results that are QNames are serialized in the form `[prefix:]localname`.

Precision of xs:decimal

The precision refers to the number of digits in the number, and a minimum of 18 digits is required by the specification. For division operations that produce a result of type `xs:decimal`, the precision is 19 digits after the decimal point with no rounding.

Implicit timezone

When two `date`, `time`, or `dateTime` values need to be compared, the timezones of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `implicit-timezone()` function.

Collations

The default collation is the Unicode codepoint collation, which compares strings on the basis of their Unicode codepoint. The engine uses the Unicode Collation Algorithm. Other supported collations are the ICU collations listed below; to use one of these, supply its URI as given in the table below. Any string comparisons, including for the `max` and `min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Language	**URIs**
`da`: Danish	`da_DK`
`de`: German	`de_AT, de_BE, de_CH, de_DE, de_LI, de_LU`
`en`: English	`en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA, en_GB, en_GU, en_HK, en_IE, en_IN, en_JM, en_MH, en_MP, en_MT, en_MU, en_NA, en_NZ, en_PH, en_PK, en_SG, en_TT, en_UM, en_US, en_VI, en_ZA, en_ZW`
`es`: Spanish	`es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE`
`fr`: French	`fr_BE, fr_BF, fr_BI, fr_BJ, fr_BL, fr_CA, fr_CD, fr_CF, fr_CG, fr_CH, fr_CI, fr_CM, fr_DJ, fr_FR, fr_GA, fr_GN, fr_GP, fr_GQ, fr_KM, fr_LU, fr_MC, fr_MF, fr_MG, fr_ML, fr_MQ, fr_NE, fr_RE, fr_RW, fr_SN, fr_TD, fr_TG`
`it`: Italian	`it_CH, it_IT`
`ja`: Japanese	`ja_JP`
`nb`: Norwegian Bokmal	`nb_NO`
`nl`: Dutch	`nl_AW, nl_BE, nl_NL`
`nn`: Nynorsk	`nn_NO`
`pt`: Portuguese	`pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST`
`ru`: Russian	`ru_MD, ru_RU, ru_UA`
`sv`: Swedish	`sv_FI, sv_SE`

Namespace axis
The namespace axis is deprecated in XPath 2.0. Use of the namespace axis is, however, supported. To access namespace information with XPath 2.0 mechanisms, use the `in-scope-prefixes()`, `namespace-uri()` and `namespace-uri-for-prefix()` functions.

19.2.2.1 Altova Extension Functions

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for

Altova extension functions in that release.

Functions defined in the W3C's XPath/XQuery Functions specifications can be used in: (i) XPath expressions in an XSLT context, and (ii) in XQuery expressions in an XQuery document. In this documentation we indicate the functions that can be used in the former context (XPath in XSLT) with an **XP** symbol and call them XPath functions; those functions that can be used in the latter (XQuery) context are indicated with an **XQ** symbol; they work as XQuery functions. The W3C's XSLT specifications—not XPath/XQuery Functions specifications—also define functions that can be used in XPath expressions in XSLT documents. These functions are marked with an **XSLT** symbol and are called XSLT functions. The XPath/XQuery and XSLT versions in which a function can be used are indicated in the description of the function (*see symbols below*). Functions from the XPath/XQuery and XSLT function libraries are listed without a prefix. Extension functions from other libraries, such as Altova extension functions, are listed with a prefix.

XPath functions (used in XPath expressions in XSLT):	**XP1** **XP2** **XP3.1**
XSLT functions (used in XPath expressions in XSLT):	**XSLT1** **XSLT2** **XSLT3**
XQuery functions (used in XQuery expressions in XQuery):	**XQ1** **XQ3.1**

XSLT functions 1901

XSLT functions can only be used in XPath expressions in an XSLT context (similarly to XSLT 2.0's `current-group()` or `key()` functions). These functions are not intended for, and will not work in, a non-XSLT context (for instance, in an XQuery context). Note that XSLT functions for XBRL can be used only with editions of Altova products that have XBRL support.

XPath/XQuery functions

XPath/XQuery functions can be used both in XPath expressions in XSLT contexts as well as in XQuery expressions:

- Date/Time 1904
- Geolocation 1921
- Image-related 1933
- Numeric 1937
- Sequence 1959
- String 1967
- Miscellaneous 1973

Chart functions (Enterprise and Server Editions only)

Altova extension functions for charts 1976 are supported only in the Enterprise and Server Editions of Altova products and enable charts to be generated from XML data.

19.2.2.1.1 XSLT Functions

XSLT extension functions can be used in XPath expressions in an XSLT context. They will not work in a non-XSLT context (for instance, in an XQuery context).

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace,** **`http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

General functions

distinct-nodes [altova:]

`altova:distinct-nodes(`*`node()*`***`) as node()*`** XSLT1 XSLT2 XSLT3

Takes a set of one or more nodes as its input and returns the same set minus nodes with duplicate values. The comparison is done using the XPath/XQuery function `fn:deep-equal`.

Examples

- **`altova:distinct-nodes`**`(country)` returns all child `country` nodes less those having duplicate values.

evaluate [altova:]

`altova:evaluate(XPathExpression` *`as xs:string`***`[, ValueOf$p1, ... ValueOf$pN])`** XSLT1 XSLT2 XSLT3

Takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression. For example: **`altova:evaluate('//Name[1]')`** returns the contents of the first `Name` element in the document. Note that the expression `//Name[1]` is passed as a string by enclosing it in single quotes.

The `altova:evaluate` function can optionally take additional arguments. These arguments are the values of in-scope variables that have the names `p1`, `p2`, `p3`... `pN`. Note the following points about usage: (i) The variables must be defined with names of the form `pX`, where `X` is an integer; (ii) the `altova:evaluate` function's arguments (*see signature above*), from the second argument onwards, provide the values of the variables, with the sequence of the arguments corresponding to the numerically ordered sequence of variables: `p1` to `pN`: The second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on; (iii) The variable values must be of type `item*`.

Example

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
```
outputs `"hi 20 10"`

In the listing above, notice the following:

- The second argument of the `altova:evaluate` expression is the value assigned to the variable `$p1`, the third argument that assigned to the variable `$p2`, and so on.
- Notice that the fourth argument of the function is a string value, indicated by its being enclosed in quotes.
- The `select` attribute of the `xs:variable` element supplies the XPath expression. Since this expression must be of type `xs:string`, it is enclosed in single quotes.

Examples to further illustrate the use of variables

- ```
  <xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, //Name[1])" />
  ```
  Outputs value of the first `Name` *element.*

- ```
  <xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, '//Name[1]')" />
  ```
  Outputs "`//Name[1]`*"*

The `altova:evaluate()` extension function is useful in situations where an XPath expression in the XSLT stylesheet contains one or more parts that must be evaluated dynamically. For example, consider a situation in which a user enters his request for the sorting criterion and this criterion is stored in the attribute `UserReq/@sortkey`. In the stylesheet, you could then have the expression: `<xsl:sort select="`**`altova:evaluate(../UserReq/@sortkey)`**`" order="ascending"/>`. The `altova:evaluate()` function reads the `sortkey` attribute of the `UserReq` child element of the parent of the context node. Say the value of the `sortkey` attribute is `Price`, then `Price` is returned by the `altova:evaluate()` function and becomes the value of the `select` attribute: `<xsl:sort select="`**`Price`**`" order="ascending"/>`. If this `sort` instruction occurs within the context of an element called `Order`, then the `Order` elements will be sorted according to the values of their `Price` children. Alternatively, if the value of `@sortkey` were, say, `Date`, then the `Order` elements would be sorted according to the values of their `Date` children. So the sort criterion for `Order` is selected from the `sortkey` attribute at runtime. This could not have been achieved with an expression like: `<xsl:sort select="`**`../UserReq/@sortkey`**`" order="ascending"/>`. In the case shown above, the sort criterion would be the `sortkey` attribute itself, not `Price` or `Date` (or any other current content of `sortkey`).

Note: The static context includes namespaces, types, and functions—but not variables—from the calling environment. The base URI and default namespace are inherited.

More examples

- Static variables: `<xsl:value-of select="$i3, $i2, $i1" />`
  Outputs the values of three variables.

- Dynamic XPath expression with dynamic variables:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />
  ```
  Outputs "`30 20 10`*"*

- Dynamic XPath expression with no dynamic variable:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath)" />
  ```
  Outputs error: No variable defined for $p3.

▼ encode-for-rtf [altova:]

altova:encode-for-rtf(input *as xs:string*, **preserveallwhitespace** *as xs:boolean*, **preservenewlines** *as xs:boolean*) **as xs:string** XSLT2 XSLT3
Converts the input string into code for RTF. Whitespace and new lines will be preserved according to the boolean value specified for their respective arguments.

[Top 1901]

XBRL functions

Altova XBRL functions can be used only with editions of Altova products that have XBRL support.

▼ xbrl-footnotes [altova:]

altova:xbrl-footnotes(*node()***) as node()*** XSLT2 XSLT3
Takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

▼ xbrl-labels [altova:]

altova:xbrl-labels(*xs:QName*, *xs:string***) as node()*** XSLT2 XSLT3
Takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL label nodes associated with the input node.

[Top 1901]

19.2.2.1.2 XPath/XQuery Functions: Date and Time

Altova's date/time extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data held as XML Schema's various date and time datatypes. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, **http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1

XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

Grouped by functionality

Listed alphabetically

[altova:time-no-TZ](1910)
[altova:weekday-from-date](1913)
[altova:weekday-from-dateTime](1913)
[altova:weeknumber-from-date](1914)
[altova:weeknumber-from-dateTime](1914)

[Top 1904]

Add a duration to xs:dateTime XP3.1 XQ3.1

These functions add a duration to **`xs:dateTime`** and return **`xs:dateTime`**. The `xs:dateTime` type has a format of `CCYY-MM-DDThh:mm:ss.sss`. This is a concatenation of the `xs:date` and `xs:time` formats separated by the letter `T`. A timezone suffix (**`+01:00`**, for example) is optional.

▼ add-years-to-dateTime [altova:]

`altova:add-years-to-dateTime(DateTime` *`as xs:dateTime`*, **`Years`** *`as xs:integer`*) **`as xs:dateTime`** XP3.1 XQ3.1

Adds a duration in years to an `xs:dateTime` (*see examples below*). The second argument is the number of years to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

⊟ *Examples*

- **`altova:add-years-to-dateTime`**`(xs:dateTime("2014-01-15T14:00:00"), 10)` returns `2024-01-15T14:00:00`
- **`altova:add-years-to-dateTime`**`(xs:dateTime("2014-01-15T14:00:00"), -4)` returns `2010-01-15T14:00:00`

▼ add-months-to-dateTime [altova:]

`altova:add-months-to-dateTime(DateTime` *`as xs:dateTime`*, **`Months`** *`as xs:integer`*) **`as xs:dateTime`** XP3.1 XQ3.1

Adds a duration in months to an `xs:dateTime` (*see examples below*). The second argument is the number of months to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

⊟ *Examples*

- **`altova:add-months-to-dateTime`**`(xs:dateTime("2014-01-15T14:00:00"), 10)` returns `2014-11-15T14:00:00`
- **`altova:add-months-to-dateTime`**`(xs:dateTime("2014-01-15T14:00:00"), -2)` returns `2013-11-15T14:00:00`

▼ add-days-to-dateTime [altova:]

`altova:add-days-to-dateTime(DateTime` *`as xs:dateTime`*, **`Days`** *`as xs:integer`*) **`as xs:dateTime`** XP3.1 XQ3.1

Adds a duration in days to an `xs:dateTime` (*see examples below*). The second argument is the number of days to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

⊟ *Examples*

- **`altova:add-days-to-dateTime`**`(xs:dateTime("2014-01-15T14:00:00"), 10)` returns `2014-`

01-25T14:00:00

- **altova:add-days-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -8) returns 2014-01-07T14:00:00

add-hours-to-dateTime [altova:]

altova:add-hours-to-dateTime(DateTime *as xs:dateTime*, **Hours** *as xs:integer*) **as xs:dateTime** XP3.1 XQ3.1

Adds a duration in hours to an xs:dateTime (*see examples below*). The second argument is the number of hours to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- **altova:add-hours-to-dateTime**(xs:dateTime("2014-01-15T13:00:00"), 10) returns 2014-01-15T23:00:00
- **altova:add-hours-to-dateTime**(xs:dateTime("2014-01-15T13:00:00"), -8) returns 2014-01-15T05:00:00

add-minutes-to-dateTime [altova:]

altova:add-minutes-to-dateTime(DateTime *as xs:dateTime*, **Minutes** *as xs:integer*) **as xs:dateTime** XP3.1 XQ3.1

Adds a duration in minutes to an xs:dateTime (*see examples below*). The second argument is the number of minutes to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- **altova:add-minutes-to-dateTime**(xs:dateTime("2014-01-15T14:10:00"), 45) returns 2014-01-15T14:55:00
- **altova:add-minutes-to-dateTime**(xs:dateTime("2014-01-15T14:10:00"), -5) returns 2014-01-15T14:05:00

add-seconds-to-dateTime [altova:]

altova:add-seconds-to-dateTime(DateTime *as xs:dateTime*, **Seconds** *as xs:integer*) **as xs:dateTime** XP3.1 XQ3.1

Adds a duration in seconds to an xs:dateTime (*see examples below*). The second argument is the number of seconds to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), 20) returns 2014-01-15T14:00:30
- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), -5) returns 2014-01-15T14:00:05

[Top 1904]

Add a duration to xs:date XP3.1 XQ3.1

These functions add a duration to **xs:date** and return **xs:date**. The `xs:date` type has a format of `CCYY-MM-DD`.

▼ add-years-to-date [altova:]

altova:add-years-to-date(Date *as xs:date*, **Years** *as xs:integer*) **as xs:date** XP3.1 XQ3.1

Adds a duration in years to a date. The second argument is the number of years to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

⊟ *Examples*

- **altova:add-years-to-date**(xs:date("2014-01-15"), 10) returns 2024-01-15
- **altova:add-years-to-date**(xs:date("2014-01-15"), -4) returns 2010-01-15

▼ add-months-to-date [altova:]

altova:add-months-to-date(Date *as xs:date*, **Months** *as xs:integer*) **as xs:date** XP3.1 XQ3.1

Adds a duration in months to a date. The second argument is the number of months to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

⊟ *Examples*

- **altova:add-months-to-date**(xs:date("2014-01-15"), 10) returns 2014-11-15
- **altova:add-months-to-date**(xs:date("2014-01-15"), -2) returns 2013-11-15

▼ add-days-to-date [altova:]

altova:add-days-to-date(Date *as xs:date*, **Days** *as xs:integer*) **as xs:date** XP3.1 XQ3.1

Adds a duration in days to a date. The second argument is the number of days to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

⊟ *Examples*

- **altova:add-days-to-date**(xs:date("2014-01-15"), 10) returns 2014-01-25
- **altova:add-days-to-date**(xs:date("2014-01-15"), -8) returns 2014-01-07

[Top 1904]

Format and retrieve durations XP3.1 XQ3.1

These functions parse an input **xs:duration** or **xs:string** and return, respectively, an **xs:string** or **xs:duration**.

▼ format-duration [altova:]

altova:format-duration(Duration *as xs:duration*, **Picture** *as xs:string*) **as xs:string** XP3.1 XQ3.1

Formats a duration, which is submitted as the first argument, according to a picture string submitted as the second argument. The output is a text string formatted according to the picture string.

⊟ *Examples*

- **altova:format-duration**(xs:duration("P2DT2H53M11.7S"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "Days:02 Hours:02 Minutes:53

`Seconds:11 Fractions:7"`

- **altova:format-duration**(xs:duration("P3M2DT2H53M11.7S"), "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "Months:03 Days:02 Hours:02 Minutes:53"

▼ parse-duration [altova:]

altova:parse-duration(InputString *as xs:string*, **Picture** *as xs:string***) as xs:duration** XP3.1 XQ3.1

Takes a patterned string as the first argument, and a picture string as the second argument. The input string is parsed on the basis of the picture string, and an `xs:duration` is returned.

⊟ *Examples*

- **altova:parse-duration**("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "P2DT2H53M11.7S"
- **altova:parse-duration**("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "P3M2DT2H53M"

[Top 1904]

Add a duration to xs:time XP3.1 XQ3.1

These functions add a duration to **xs:time** and return **xs:time**. The `xs:time` type has a lexical form of `hh:mm:ss.sss`. An optional time zone may be suffixed. The letter `Z` indicates Coordinated Universal Time (UTC). All other time zones are represented by their difference from UTC in the format `+hh:mm`, or `-hh:mm`. If no time zone value is present, it is considered unknown; it is not assumed to be UTC.

▼ add-hours-to-time [altova:]

altova:add-hours-to-time(Time *as xs:time*, **Hours** *as xs:integer***) as xs:time** XP3.1 XQ3.1

Adds a duration in hours to a time. The second argument is the number of hours to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`.

⊟ *Examples*

- **altova:add-hours-to-time**(xs:time("11:00:00"), 10) returns 21:00:00
- **altova:add-hours-to-time**(xs:time("11:00:00"), -7) returns 04:00:00

▼ add-minutes-to-time [altova:]

altova:add-minutes-to-time(Time *as xs:time*, **Minutes** *as xs:integer***) as xs:time** XP3.1 XQ3.1

Adds a duration in minutes to a time. The second argument is the number of minutes to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`.

⊟ *Examples*

- **altova:add-minutes-to-time**(xs:time("14:10:00"), 45) returns 14:55:00
- **altova:add-minutes-to-time**(xs:time("14:10:00"), -5) returns 14:05:00

▼ add-seconds-to-time [altova:]

altova:add-seconds-to-time(Time *as xs:time*, **Minutes** *as xs:integer*) **as xs:time** XP3.1 XQ3.1
Adds a duration in seconds to a time. The second argument is the number of seconds to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`. The Seconds component can be in the range of `0` to `59.999`.

Examples

- `altova:add-seconds-to-time(xs:time("14:00:00"), 20)` returns `14:00:20`
- `altova:add-seconds-to-time(xs:time("14:00:00"), 20.895)` returns `14:00:20.895`

[Top1904]

Remove the timezone part from date/time datatypes XP3.1 XQ3.1

These functions remove the timezone from the current **xs:dateTime**, **xs:date**, or **xs:time** values, respectively. Note that the difference between `xs:dateTime` and `xs:dateTimeStamp` is that in the case of the latter the timezone part is required (while it is optional in the case of the former). So the format of an `xs:dateTimeStamp` value is: `CCYY-MM-DDThh:mm:ss.sss±hh:mm`. or `CCYY-MM-DDThh:mm:ss.sssZ`. If the date and time is read from the system clock as `xs:dateTimeStamp`, the `current-dateTime-no-TZ()` function can be used to remove the timezone if so required.

current-date-no-TZ [altova:]

altova:current-date-no-TZ() as xs:date XP3.1 XQ3.1
This function takes no argument. It removes the timezone part of `current-date()` (which is the current date according to the system clock) and returns an `xs:date` value.

Examples

If the current date is **2014-01-15+01:00**:

- `altova:current-date-no-TZ()` returns `2014-01-15`

current-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() as xs:dateTime XP3.1 XQ3.1
This function takes no argument. It removes the timezone part of `current-dateTime()` (which is the current date-and-time according to the system clock) and returns an `xs:dateTime` value.

Examples

If the current dateTime is **2014-01-15T14:00:00+01:00**:

- `altova:current-dateTime-no-TZ()` returns `2014-01-15T14:00:00`

current-time-no-TZ [altova:]

altova:current-time-no-TZ() as xs:time XP3.1 XQ3.1
This function takes no argument. It removes the timezone part of `current-time()` (which is the current time according to the system clock) and returns an `xs:time` value.

Examples

If the current time is **14:00:00+01:00**:

- **altova:current-time-no-TZ**() returns 14:00:00

▼ date-no-TZ [altova:]

altova:date-no-TZ(InputDate *as xs:date***) as xs:date** XP3.1 XQ3.1
This function takes an `xs:date` argument, removes the timezone part from it, and returns an `xs:date` value. Note that the date is not modified.
⊟ *Examples*

- **altova:date-no-TZ**(xs:date("2014-01-15+01:00")) returns 2014-01-15

▼ dateTime-no-TZ [altova:]

altova:dateTime-no-TZ(InputDateTime *as xs:dateTime***) as xs:dateTime** XP3.1 XQ3.1
This function takes an `xs:dateTime` argument, removes the timezone part from it, and returns an `xs:dateTime` value. Note that neither the date nor the time is modified.
⊟ *Examples*

- **altova:dateTime-no-TZ**(xs:date("2014-01-15T14:00:00+01:00")) returns 2014-01-15T14:00:00

▼ time-no-TZ [altova:]

altova:time-no-TZ(InputTime *as xs:time***) as xs:time** XP3.1 XQ3.1
This function takes an `xs:time` argument, removes the timezone part from it, and returns an `xs:time` value. Note that the time is not modified.
⊟ *Examples*

- **altova:time-no-TZ**(xs:time("14:00:00+01:00")) returns 14:00:00

[Top 1904]

Return the number of days, hours, minutes, seconds from durations XP3.1 XQ3.1

These functions return the number of days in a month, and the number of hours, minutes, and seconds, respectively, from durations.

▼ days-in-month [altova:]

altova:days-in-month(Year *as xs:integer***, Month** *as xs:integer***) as xs:integer** XP3.1 XQ3.1
Returns the number of days in the specified month. The month is specified by means of the `Year` and `Month` arguments.
⊟ *Examples*

- **altova:days-in-month**(2018, 10) returns 31
- **altova:days-in-month**(2018, 2) returns 28
- **altova:days-in-month**(2020, 2) returns 29

hours-from-dayTimeDuration-accumulated

altova:hours-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration***) as xs:integer** XP3.1 XQ3.1

Returns the total number of hours in the duration submitted by the `DayAndTime` argument (which is of type `xs:duration`). The hours in the `Day` and `Time` components are added together to give a result that is an integer. A new hour is counted only for a full 60 minutes. Negative durations result in a negative hour value.

Examples

- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5D"))` returns `120`, which is the total number of hours in 5 days.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H"))` returns `122`, which is the total number of hours in 5 days plus 2 hours.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H60M"))` returns `123`, which is the total number of hours in 5 days plus 2 hours and 60 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H119M"))` returns `123`, which is the total number of hours in 5 days plus 2 hours and 119 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H120M"))` returns `124`, which is the total number of hours in 5 days plus 2 hours and 120 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("-P5DT2H"))` returns `-122`

minutes-from-dayTimeDuration-accumulated

altova:minutes-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration***) as xs:integer** XP3.1 XQ3.1

Returns the total number of minutes in the duration submitted by the `DayAndTime` argument (which is of type `xs:duration`). The minutes in the `Day` and `Time` components are added together to give a result that is an integer. Negative durations result in a negative minute value.

Examples

- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT60M"))` returns `60`
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT1H"))` returns `60`, which is the total number of minutes in 1 hour.
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT1H40M"))` returns `100`
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("P1D"))` returns `1440`, which is the total number of minutes in 1 day.
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("-P1DT60M"))` returns `-1500`

seconds-from-dayTimeDuration-accumulated

altova:seconds-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration***) as xs:integer** XP3.1 XQ3.1

Returns the total number of seconds in the duration submitted by the `DayAndTime` argument (which is of type `xs:duration`). The seconds in the `Day` and `Time` components are added together to give a result that is an integer. Negative durations result in a negative seconds value.

Examples

- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1M"))` returns `60`, which is the total number of seconds in 1 minute.

- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1H")) returns 3600, which is the total number of seconds in 1 hour.
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1H2M")) returns 3720
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("P1D")) returns 86400, which is the total number of seconds in 1 day.
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("-P1DT1M")) returns -86460

Return the weekday from xs:dateTime or xs:date XP3.1 XQ3.1

These functions return the weekday (as an integer) from `xs:dateTime` or `xs:date`. The days of the week are numbered (using the American format) from `1` to `7`, with `Sunday=1`. In the European format, the week starts with Monday (`=1`). The American format, where `Sunday=1`, can be set by using the integer `0` where an integer is accepted to indicate the format.

▼ weekday-from-dateTime [altova:]

altova:weekday-from-dateTime(DateTime *as xs:dateTime***) as xs:integer** XP3.1 XQ3.1

Takes a date-with-time as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Sunday=1`. If the European format is required (where `Monday=1`), use the other signature of this function (*see next signature below*).

⊟ *Examples*

- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00")) returns 2, which would indicate a Monday.

altova:weekday-from-dateTime(DateTime *as xs:dateTime*, **Format** *as xs:integer***) as xs:integer** XP3.1 XQ3.1

Takes a date-with-time as its first argument and returns the day of the week of this date as an integer. If the second (integer) argument is `0`, then the weekdays are numbered `1` to `7` starting with `Sunday=1`. If the second argument is an integer other than `0`, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

⊟ *Examples*

- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 1) returns 1, which would indicate a Monday
- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 4) returns 1, which would indicate a Monday
- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 0) returns 2, which would indicate a Monday.

▼ weekday-from-date [altova:]

altova:weekday-from-date(Date *as xs:date***) as xs:integer** XP3.1 XQ3.1

Takes a date as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Sunday=1`. If the European format is required (where `Monday=1`), use the other signature of this function (*see next signature below*).

⊟ *Examples*

- **altova:weekday-from-date**(xs:date("2014-02-03+01:00")) returns 2, which would indicate a Monday.

altova:weekday-from-date(Date *as xs:date*, **Format** *as xs:integer*) **as xs:integer** XP3.1 XQ3.1
Takes a date as its first argument and returns the day of the week of this date as an integer. If the second (`Format`) argument is `0`, then the weekdays are numbered `1` to `7` starting with `Sunday=1`. If the second argument is an integer other than `0`, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

Examples

- **altova:weekday-from-date**(xs:date("2014-02-03"), 1) returns 1, which would indicate a Monday
- **altova:weekday-from-date**(xs:date("2014-02-03"), 4) returns 1, which would indicate a Monday
- **altova:weekday-from-date**(xs:date("2014-02-03"), 0) returns 2, which would indicate a Monday.

[Top 1904]

Return the week number from xs:dateTime or xs:date XP2 XQ1 XP3.1 XQ3.1

These functions return the week number (as an integer) from `xs:dateTime` or `xs:date`. Week-numbering is available in the US, ISO/European, and Islamic calendar formats. Week-numbering is different in these calendar formats because the week is considered to start on different days (on Sunday in the US format, Monday in the ISO/European format, and Saturday in the Islamic format).

weeknumber-from-date [altova:]

altova:weeknumber-from-date(Date *as xs:date*, **Calendar** *as xs:integer*) **as xs:integer** XP2 XQ1 XP3.1 XQ3.1
Returns the week number of the submitted **`Date`** argument as an integer. The second argument (**`Calendar`**) specifies the calendar system to follow.
Supported **`Calendar`** values are:

- **0 = US calendar** (*week starts Sunday*)
- **1 = ISO standard, European calendar** (*week starts Monday*)
- **2 = Islamic calendar** (*week starts Saturday*)

Default is **`0`**.

Examples

- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 0) returns 13
- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 1) returns 12
- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 2) returns 13
- **altova:weeknumber-from-date**(xs:date("2014-03-23")) returns 13

The day of the date in the examples above (`2014-03-23`) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

▼ weeknumber-from-dateTime [altova:]

altova:weeknumber-from-dateTime(DateTime *as xs:dateTime*, **Calendar** *as xs:integer*) **as xs:integer** XP2 XQ1 XP3.1 XQ3.1

Returns the week number of the submitted **DateTime** argument as an integer. The second argument (**Calendar**) specifies the calendar system to follow.
Supported **Calendar** values are:

- **0 = US calendar** (*week starts Sunday*)
- **1 = ISO standard, European calendar** (*week starts Monday*)
- **2 = Islamic calendar** (*week starts Saturday*)

Default is **0**.

⊟ *Examples*

- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 0) returns 13
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 1) returns 12
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 2) returns 13
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00")) returns 13

The day of the dateTime in the examples above (2014-03-23T00:00:00) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

[Top 1904]

Build date, time, and duration datatypes from their lexical components XP3.1 XQ3.1

The functions take the lexical components of the xs:date, xs:time, or xs:duration datatype as input arguments and combine them to build the respective datatype.

▼ build-date [altova:]

altova:build-date(Year *as xs:integer*, **Month** *as xs:integer*, **Date** *as xs:integer*) **as xs:date** XP3.1 XQ3.1

The first, second, and third arguments are, respectively, the year, month, and date. They are combined to build a value of xs:date type. The values of the integers must be within the correct range of that particular date part. For example, the second argument (for the month part) should not be greater than 12.

⊟ *Examples*

- **altova:build-date**(2014, 2, 03) returns 2014-02-03

▼ build-time [altova:]

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*) **as xs:time** XP3.1 XQ3.1

The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. They are combined to build a value of xs:time type. The values of the integers must be within the correct range of that particular time part. For example, the second (Minutes) argument should not be greater than `59`. To add a timezone part to the value, use the other signature of this function (*see next signature*).

Examples

- **altova:build-time**(23, 4, 57) returns `23:04:57`

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*, **TimeZone** *as xs:string*) **as xs:time** **XP3.1** **XQ3.1**

The first, second, and third arguments are, respectively, the hour (`0` to `23`), minutes (`0` to `59`), and seconds (`0` to `59`) values. The fourth argument is a string that provides the timezone part of the value. The four arguments are combined to build a value of `xs:time` type. The values of the integers must be within the correct range of that particular time part. For example, the second (`Minutes`) argument should not be greater than `59`.

Examples

- **altova:build-time**(23, 4, 57, '+1') returns `23:04:57+01:00`

build-duration [altova:]

altova:build-duration(Years *as xs:integer*, **Months** *as xs:integer*) **as xs:yearMonthDuration** **XP3.1** **XQ3.1**

Takes two arguments to build a value of type `xs:yearMonthDuration`. The first argument provides the `Years` part of the duration value, while the second argument provides the `Months` part. If the second (`Months`) argument is greater than or equal to `12`, then the integer is divided by `12`; the quotient is added to the first argument to provide the `Years` part of the duration value while the remainder (of the division) provides the `Months` part. To build a duration of type `xs:dayTimeDuration.`, see the next signature.

Examples

- **altova:build-duration**(2, 10) returns `P2Y10M`
- **altova:build-duration**(14, 27) returns `P16Y3M`
- **altova:build-duration**(2, 24) returns `P4Y`

altova:build-duration(Days *as xs:integer*, **Hours** *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*) as **xs:dayTimeDuration** **XP3.1** **XQ3.1**

Takes four arguments and combines them to build a value of type `xs:dayTimeDuration`. The first argument provides the `Days` part of the duration value, the second, third, and fourth arguments provide, respectively, the `Hours`, `Minutes`, and `Seconds` parts of the duration value. Each of the three Time arguments is converted to an equivalent value in terms of the next higher unit and the result is used for calculation of the total duration value. For example, 72 seconds is converted to `1M+12S` (1 minute and 12 seconds), and this value is used for calculation of the total duration value. To build a duration of type `xs:yearMonthDuration.`, see the previous signature.

Examples

- **altova:build-duration**(2, 10, 3, 56) returns `P2DT10H3M56S`
- **altova:build-duration**(1, 0, 100, 0) returns `P1DT1H40M`
- **altova:build-duration**(1, 0, 0, 3600) returns `P1DT1H`

[Top[1904]]

Construct date, dateTime, and time datatypes from string input XP2 XQ1 XP3.1 XQ3.1

These functions take strings as arguments and construct `xs:date`, `xs:dateTime`, or `xs:time` datatypes. The string is analyzed for components of the datatype based on a submitted pattern argument.

- parse-date [altova:]

altova:parse-date(Date *as xs:string*, **DatePattern** *as xs:string*) **as xs:date** XP2 XQ1 XP3.1 XQ3.1

Returns the input string **Date** as an **xs:date** value. The second argument **DatePattern** specifies the pattern (sequence of components) of the input string. **DatePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

D	Date
M	Month
Y	Year

The pattern in **DatePattern** must match the pattern in **Date**. Since the output is of type **xs:date**, the output will always have the lexical format **YYYY-MM-DD**.

Examples

- `altova:parse-date(xs:string("09-12-2014"), "[D]-[M]-[Y]")` returns `2014-12-09`
- `altova:parse-date(xs:string("09-12-2014"), "[M]-[D]-[Y]")` returns `2014-09-12`
- `altova:parse-date("06/03/2014", "[M]/[D]/[Y]")` returns `2014-06-03`
- `altova:parse-date("06 03 2014", "[M] [D] [Y]")` returns `2014-06-03`
- `altova:parse-date("6 3 2014", "[M] [D] [Y]")` returns `2014-06-03`

- parse-dateTime [altova:]

altova:parse-dateTime(DateTime *as xs:string*, **DateTimePattern** *as xs:string*) **as xs:dateTime** XP2 XQ1 XP3.1 XQ3.1

Returns the input string **DateTime** as an **xs:dateTime** value.The second argument **DateTimePattern** specifies the pattern (sequence of components) of the input string. **DateTimePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

D	Date
M	Month
Y	Year
H	Hour
m	minutes
s	seconds

The pattern in **DateTimePattern** must match the pattern in **DateTime**. Since the output is of type **xs:dateTime**, the output will always have the lexical format **YYYY-MM-DDTHH:mm:ss**.

Examples

- `altova:parse-dateTime(xs:string("09-12-2014 13:56:24"), "[M]-[D]-[Y] [H]:[m]:`

[s]") returns 2014-09-12T13:56:24
- **altova:parse-dateTime**("time=13:56:24; date=09-12-2014", "time=[H]:[m]:[s]; date=[D]-[M]-[Y]") returns 2014-12-09T13:56:24

parse-time [altova:]

altova:parse-time(Time *as xs:string*, **TimePattern** *as xs:string***) as xs:time** XP2 XQ1 XP3.1 XQ3.1

Returns the input string **Time** as an **xs:time** value.The second argument **TimePattern** specifies the pattern (sequence of components) of the input string. **TimePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

H	Hour
m	minutes
s	seconds

The pattern in **TimePattern** must match the pattern in **Time**. Since the output is of type **xs:time**, the output will always have the lexical format **HH:mm:ss**.

Examples

- **altova:parse-time**(xs:string("13:56:24"), "[H]:[m]:[s]") returns 13:56:24
- **altova:parse-time**("13-56-24", "[H]-[m]") returns 13:56:00
- **altova:parse-time**("time=13h56m24s", "time=[H]h[m]m[s]s") returns 13:56:24
- **altova:parse-time**("time=24s56m13h", "time=[s]s[m]m[H]h") returns 13:56:24

[Top1904]

Age-related functions XP3.1 XQ3.1

These functions return the age as calculated (i) between one input argument date and the current date, or (ii) between two input argument dates. The **altova:age** function returns the age in terms of years, the **altova:age-details** function returns the age as a sequence of three integers giving the years, months, and days of the age.

age [altova:]

altova:age(StartDate *as xs:date***) as xs:integer** XP3.1 XQ3.1

Returns an integer that is the age *in years* of some object, counting from a start-date submitted as the argument and ending with the current date (taken from the system clock). If the input argument is a date anything greater than or equal to one year in the future, the return value will be negative.

Examples

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2013-01-15")) returns 1
- **altova:age**(xs:date("2013-01-16")) returns 0
- **altova:age**(xs:date("2015-01-15")) returns -1
- **altova:age**(xs:date("2015-01-14")) returns 0

altova:age(StartDate *as xs:date*, **EndDate** *as xs:date*) **as xs:integer** XP3.1 XQ3.1
Returns an integer that is the age *in years* of some object, counting from a start-date that is submitted as the first argument up to an end-date that is the second argument. The return value will be negative if the first argument is one year or more later than the second argument.

Examples

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2000-01-15"), xs:date("2010-01-15")) returns 10
- **altova:age**(xs:date("2000-01-15"), current-date()) returns 14 if the current date is 2014-01-15
- **altova:age**(xs:date("2014-01-15"), xs:date("2010-01-15")) returns -4

age-details [altova:]

altova:age-details(InputDate *as xs:date*) as (**xs:integer**)* XP3.1 XQ3.1
Returns three integers that are, respectively, the years, months, and days between the date that is submitted as the argument and the current date (taken from the system clock). The sum of the returned years+months+days together gives the total time difference between the two dates (the input date and the current date). The input date may have a value earlier or later than the current date, but whether the input date is earlier or later is not indicated by the sign of the return values; the return values are always positive.

Examples

If the current date is **2014-01-15**:

- **altova:age-details**(xs:date("2014-01-16")) returns (0 0 1)
- **altova:age-details**(xs:date("2014-01-14")) returns (0 0 1)
- **altova:age-details**(xs:date("2013-01-16")) returns (1 0 1)
- **altova:age-details**(current-date()) returns (0 0 0)

altova:age-details(Date-1 *as xs:date*, **Date-2** *as xs:date*) as (**xs:integer**)* XP3.1 XQ3.1
Returns three integers that are, respectively, the years, months, and days between the two argument dates. The sum of the returned years+months+days together gives the total time difference between the two input dates; it does not matter whether the earlier or later of the two dates is submitted as the first argument. The return values do not indicate whether the input date occurs earlier or later than the current date. Return values are always positive.

Examples

- **altova:age-details**(xs:date("2014-01-16"), xs:date("2014-01-15")) returns (0 0 1)
- **altova:age-details**(xs:date("2014-01-15"), xs:date("2014-01-16")) returns (0 0 1)

[Top 1904]

Epoch time (Unix time) functions XP3.1 XQ3.1

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. Altova's Epoch time extension functions convert **xs:dateTime** values to Epoch time values and vice versa.

▼ dateTime-from-epoch [altova:]

altova:dateTime-from-epoch(Epoch *as xs:decimal* **as xs:dateTime** **XP3.1** **XQ3.1**
Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **dateTime-from-epoch** function returns the **xs:dateTime** equivalent of an Epoch time, adjusts it for the local timezone, and includes the timezone information in the result.

The function takes an `xs:decimal` argument and returns an `xs:dateTime` value that includes a **TZ** (timezone) part. The result is obtained by calculating the UTC **dateTime** equivalet of the Epoch time, and adding to it the local timezone (taken from the system clock). For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then after the UTC **dateTime** equivalent has been calculated, one hour will be added to the result. The timezone information, which is an optional lexical part of the **xs:dateTime** result, is also reported in the **dateTime** result. Compare this result with that of **dateTime-from-epoch-no-TZ**, and also see the function **epoch-from-dateTime**.

⊟ *Examples*

The examples below assume a local timezone of UTC +01:00. Consequently, the UTC **dateTime** equivalent of the submitted Epoch time will be incremented by one hour. The timezone is reported in the result.

- **altova:dateTime-from-epoch**(34) returns `1970-01-01T01:00:34+01:00`
- **altova:dateTime-from-epoch**(62) returns `1970-01-01T01:01:02+01:00`

▼ dateTime-from-epoch-no-TZ [altova:]

altova:dateTime-from-epoch-no-TZ(Epoch *as xs:decimal* **as xs:dateTime** **XP3.1** **XQ3.1**
Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **dateTime-from-epoch-no-TZ** function returns the **xs:dateTime** equivalent of an Epoch time, adjusts it for the local timezone, but does not include the timezone information in the result.

The function takes an `xs:decimal` argument and returns an `xs:dateTime` value that does not includes a **TZ** (timezone) part. The result is obtained by calculating the UTC **dateTime** equivalet of the Epoch time, and adding to it the local timezone (taken from the system clock). For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then after the UTC **dateTime** equivalent has been calculated, one hour will be added to the result. The timezone information, which is an optional lexical part of the **xs:dateTime** result, is not reported in the **dateTime** result. Compare this result with that of **dateTime-from-epoch**, and also see the function **epoch-from-dateTime**.

⊟ *Examples*

The examples below assume a local timezone of UTC +01:00. Consequently, the UTC **dateTime** equivalent of the submitted Epoch time will be incremented by one hour. The timezone is not reported in the result.

- **altova:dateTime-from-epoch**(34) returns `1970-01-01T01:00:34`
- **altova:dateTime-from-epoch**(62) returns `1970-01-01T01:01:02`

epoch-from-dateTime [altova:]

`altova:epoch-from-dateTime(dateTimeValue` *`as xs:dateTime`***`)`** **`as xs:decimal`** **XP3.1** **XQ3.1**
Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **`epoch-from-dateTime`** function returns the Epoch time equivalent of the **`xs:dateTime`** that is submitted as the argument of the function. Note that you might have to explicitly construct the **`xs:dateTime`** value. The submitted **`xs:dateTime`** value may or may not contain the optional **`TZ`** (timezone) part.

Whether the timezone part is submitted as part of the argument or not, the local timezone offset (taken from the system clock) is subtracted from the submitted **`dateTimeValue`** argument. This produces the equivalent UTC time, from which the equivalent Epoch time is calculated. For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then one hour is subtracted from the submitted **`dateTimeValue`** before the Epoch value is calculated. Also see the function **`dateTime-from-epoch`**.

Examples

The examples below assume a local timezone of UTC +01:00. Consequently, one hour will be subtracted from the submitted **`dateTime`** before the Epoch time is calculated.

- `altova:epoch-from-dateTime(xs:dateTime("1970-01-01T01:00:34+01:00"))` returns `34`
- `altova:epoch-from-dateTime(xs:dateTime("1970-01-01T01:00:34"))` returns `34`
- `altova:epoch-from-dateTime(xs:dateTime("2021-04-01T11:22:33"))` returns `1617272553`

[Top 1904]

19.2.2.1.3 XPath/XQuery Functions: Geolocation

The following geolocation XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, **`http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	**XP1** **XP2** **XP3.1**
XSLT functions (used in XPath expressions in XSLT):	**XSLT1** **XSLT2** **XSLT3**
XQuery functions (used in XQuery expressions in XQuery):	**XQ1** **XQ3.1**

format-geolocation [altova:]

altova:format-geolocation(Latitude *as xs:decimal*, **Longitude** *as xs:decimal*, **GeolocationOutputStringFormat** *as xs:integer*) **as xs:string** XP3.1 XQ3.1

Takes the latitude and longitude as the first two arguments, and outputs the geolocation as a string. The third argument, **GeolocationOutputStringFormat**, is the format of the geolocation output string; it uses integer values from `1` to `4` to identify the output string format (*see 'Geolocation output string formats' below*). Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data` 1933 function and the Exif metadata's attributes can be used to supply the input strings.

Examples

- `altova:format-geolocation(33.33, -22.22, 4)` returns the `xs:string` `"33.33 -22.22"`
- `altova:format-geolocation(33.33, -22.22, 2)` returns the `xs:string` `"33.33N 22.22W"`
- `altova:format-geolocation(-33.33, 22.22, 2)` returns the `xs:string` `"33.33S 22.22E"`
- `altova:format-geolocation(33.33, -22.22, 1)` returns the `xs:string` `"33°19'48.00"S 22°13'12.00"E"`

Geolocation output string formats:

The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (`1 to 4`). Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

1
Degrees, minutes, decimal seconds, with suffixed orientation (`N/S, E/W`) **`D°M'S.SS"N/S  D°M'S.SS"E/W`** *Example*: **`33°55'11.11"N  22°44'66.66"W`**

2
Decimal degrees, with suffixed orientation (`N/S`, `E/W`) **`D.DDN/S  D.DDE/W`** *Example*: **`33.33N  22.22W`**

3
Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); plus sign for (**`N/E`**) is optional **`+/-D°M'S.SS"  +/-D°M'S.SS"`** *Example*: **`33°55'11.11"  -22°44'66.66"`**

4
Decimal degrees, with prefixed sign (`+/-`); plus sign for (**`N/E`**) is optional **`+/-D.DD  +/-D.DD`** *Example*: **`33.33 -22.22`**

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

parse-geolocation [altova:]

`altova:parse-geolocation(GeolocationInputString` *`as xs:string`***`) as xs:decimal+`** **XP3.1** **XQ3.1**
Parses the supplied `GeolocationInputString` argument and returns the geolocation's latitude and longitude (in that order) as a sequence two `xs:decimal` items. The formats in which the geolocation input string can be supplied are listed below.

Note: The `image-exif-data` [1933] function and the Exif metadata's `@Geolocation` [1933] attribute can be used to supply the geolocation input string (*see example below*).

Examples

- `altova:parse-geolocation("33.33  -22.22")` returns the sequence of two `xs:decimals` `(33.33, 22.22)`
- `altova:parse-geolocation("48°51'29.6""N  24°17'40.2""")` returns the sequence of two `xs:decimals (48.8582222222222, 24.2945)`
- `altova:parse-geolocation('48°51''29.6"N  24°17''40.2"')` returns the sequence of two `xs:decimals (48.8582222222222, 24.2945)`
- `altova:parse-geolocation( image-exif-data(//MyImages/Image20141130.01)/@Geolocation )` returns a sequence of two `xs:decimals`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)

`D°M'S.SS"N/S  D°M'S.SS"W/E`
Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (**N/E**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (**N/E**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **+33°55.55' -22°44.44'**

- Decimal degrees, with suffixed orientation (N/S, E/W)
  `D.DDN/S  D.DDW/E`
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/S E/W**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

geolocation-distance-km [altova:]

altova:geolocation-distance-km(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) **as xs:decimal** XP3.1 XQ3.1

Calculates the distance between two geolocations in kilometers. The formats in which the geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The `image-exif-data` 1933 function and the Exif metadata's `@Geolocation` 1933 attribute can be used to supply geolocation input strings.

⊟ *Examples*

- `altova:geolocation-distance-km("33.33 -22.22", "48°51'29.6""N 24°17'40.2""")`
  returns the `xs:decimal` `4183.08132372392`

⊟ *Geolocation input string formats:*

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`
`33.33  22°44'55.25"W`
`33.33  22.45`

⊟ *Altova Exif Attribute: Geolocation*

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

▼ geolocation-distance-mi [altova:]

altova:geolocation-distance-mi(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) **as xs:decimal** XP3.1 XQ3.1

Calculates the distance between two geolocations in miles. The formats in which a geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data[1933] function and the Exif metadata's @Geolocation[1933] attribute can be used to supply geolocation input strings.

⊟ *Examples*

- **altova:geolocation-distance-mi**("33.33 -22.22", "48°51'29.6""N 24°17'40.2""") returns the xs:decimal 2599.40652340653

⊟ *Geolocation input string formats:*

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (**N/E**) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`
`33.33  22°44'55.25"W`
`33.33  22.45`

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

- geolocations-bounding-rectangle [altova:]

`altova:geolocations-bounding-rectangle(Geolocations` *`as xs:sequence`*, **`GeolocationOutputStringFormat`** *`as xs:integer`*) **`as xs:string`** XP3.1 XQ3.1
Takes a sequence of strings as its first argument; each string in the sequence is a geolocation. The function returns a sequence of two strings which are, respectively, the top-left and bottom-right geolocation coordinates of a bounding rectangle that is optimally sized to enclose all the geolocations submitted in the first argument. The formats in which a geolocation input string can be supplied are listed below (*see 'Geolocation input string formats'*). Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

The function's second argument specifies the format of the two geolocation strings in the output sequence. The argument takes an integer value from `1` to `4`, where each value identifies a different geolocation string format (*see 'Geolocation output string formats' below*).

Note: The `image-exif-data` 1933 function and the Exif metadata's attributes can be used to supply the input strings.

⊟ *Examples*

- `altova:geolocations-bounding-rectangle(("48.2143531 16.3707266", "51.50939 -0.11832"), 1)` returns the sequence `("51°30'33.804"N 0°7'5.952"W", "48°12'51.67116"N 16°22'14.61576"E")`
- `altova:geolocations-bounding-rectangle(("48.2143531 16.3707266", "51.50939 -0.11832", "42.5584577 -70.8893334"), 4)` returns the sequence `("51.50939 -70.8893334", "42.5584577 16.3707266")`

⊟ *Geolocation input string formats:*

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`

```
33.33   22°44'55.25"W
33.33   22.45
```

Geolocation output string formats:

The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (`1 to 4`). Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

1
Degrees, minutes, decimal seconds, with suffixed orientation (`N/S, E/W`) **`D°M'S.SS"N/S  D°M'S.SS"E/W`** *Example*: **`33°55'11.11"N  22°44'66.66"W`**

2
Decimal degrees, with suffixed orientation (`N/S`, `E/W`) **`D.DDN/S  D.DDE/W`** *Example*: **`33.33N  22.22W`**

3
Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); plus sign for (**`N/E`**) is optional **`+/-D°M'S.SS"  +/-D°M'S.SS"`** *Example*: **`33°55'11.11"  -22°44'66.66"`**

4
Decimal degrees, with prefixed sign (`+/-`); plus sign for (**`N/E`**) is optional **`+/-D.DD  +/-D.DD`** *Example*: **`33.33 -22.22`**

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

geolocation-within-polygon [altova:]

altova:geolocation-within-polygon(Geolocation *as xs:string*, **((PolygonPoint** *as xs:string*)**+)) as xs:boolean** XP3.1 XQ3.1

Determines whether **`Geolocation`** (the first argument) is within the polygonal area described by the **`PolygonPoint`** arguments. If the `PolygonPoint` arguments do not form a closed figure (formed when the first point and the last point are the same), then the first point is implicitly added as the last point in order to close the figure. All the arguments (`Geolocation` and `PolygonPoint+`) are given by geolocation input strings (*formats listed below*). If the `Geolocation` argument is within the polygonal area, then the function returns `true()`; otherwise it returns `false()`. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data` 1933 function and the Exif metadata's `@Geolocation` 1933 attribute can be used to supply geolocation input strings.

Examples

- `altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48 24", "58 -32"))` returns `true()`
- `altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48 24"))` returns `true()`
- `altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48°51'29.6""N 24°17'40.2"""))` returns `true()`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)

D.DDN/S D.DDW/E
Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/S E/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

geolocation-within-rectangle [altova:]

altova:geolocation-within-rectangle(Geolocation *as xs:string*, **RectCorner-1** *as xs:string*, **RectCorner-2** *as xs:string*) **as xs:boolean** **XP3.1** **XQ3.1**
Determines whether **Geolocation** (the first argument) is within the rectangle defined by the second and third arguments, **RectCorner-1** and **RectCorner-2**, which specify opposite corners of the rectangle. All the arguments (`Geolocation`, **RectCorner-1** and **RectCorner-2**) are given by geolocation input strings (*formats listed below*). If the `Geolocation` argument is within the rectangle, then the function returns `true()`; otherwise it returns `false()`. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data` 1933 function and the Exif metadata's `@Geolocation` 1933 attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", "-48 24") returns `true()`
- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", "48 24") returns `false()`
- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", "48°51'29.6""S 24°17'40.2"""") returns `true()`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`).

Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`
`33.33  22°44'55.25"W`
`33.33  22.45`

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

[Top 1921]

19.2.2.1.4 XPath/XQuery Functions: Image-Related

The following image-related XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace,** **`http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

suggested-image-file-extension [altova:]

`altova:suggested-image-file-extension(Base64String` *`as string`***`)`** **`as string?`** XP3.1 XQ3.1
Takes the Base64 encoding of an image file as its argument and returns the file extension of the image as recorded in the Base64-encoding of the image. The returned value is a suggestion based on the image type information available in the encoding. If this information is not available, then an empty string is returned. This function is useful if you wish to save a Base64 image as a file and wish to dynamically retrieve an appropriate file extension.

Examples

- **`altova:suggested-image-file-extension`**`(/MyImages/MobilePhone/Image20141130.01)` returns `'jpg'`
- **`altova:suggested-image-file-extension`**`($XML1/Staff/Person/@photo)` returns `''`

In the examples above, the nodes supplied as the argument of the function are assumed to contain a Base64-encoded image. The first example retrieves `jpg` as the file's type and extension. In the second example, the submitted Base64 encoding does not provide usable file extension information.

image-exif-data [altova:]

`altova:image-exif-data(Base64BinaryString` *`as string`***`)`** **`as element?`** XP3.1 XQ3.1
Takes a Base64-encoded JPEG image as its argument and returns an element called **`Exif`** that contains the Exif metadata of the image. The Exif metadata is created as attribute-value pairs of the `Exif` element.

The attribute names are the Exif data tags found in the Base64 encoding. The list of Exif-specification tags is given below. If a vendor-specific tag is present in the Exif data, this tag and its value will also be returned as an attribute-value pair. Additional to the standard Exif metadata tags (*see list below*), Altova-specific attribute-value pairs are also generated. These Altova Exif attributes are listed below.

Examples

- To access any one attribute, use the function like this:
  `image-exif-data(//MyImages/Image20141130.01)/@GPSLatitude`
  `image-exif-data(//MyImages/Image20141130.01)/@Geolocation`
- To access all the attributes, use the function like this:
  `image-exif-data(//MyImages/Image20141130.01)/@*`
- To access the names of all the attributes, use the following expression:
  `for $i in image-exif-data(//MyImages/Image20141130.01)/@* return name($i)`
  This is useful to find out the names of the attributes returned by the function.

Altova Exif Attribute: Geolocation

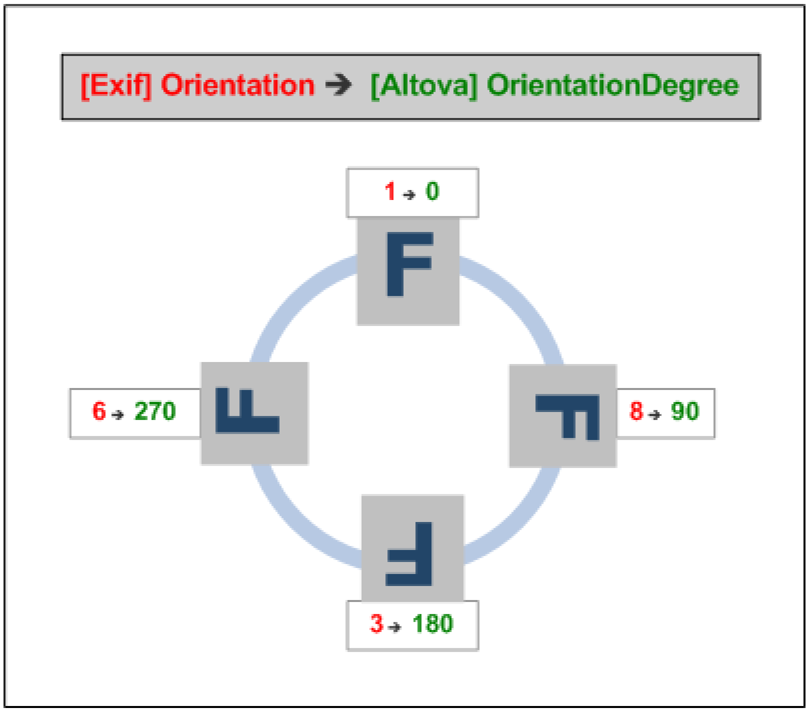
The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

Altova Exif Attribute: OrientationDegree

The Altova XPath/XQuery Engine generates the custom attribute **`OrientationDegree`** from the Exif metadata tag **`Orientation`**.

`OrientationDegree` translates the standard Exif tag `Orientation` from an integer value (`1`, `8`, `3`, or `6`) to the respective degree values of each (`0`, `90`, `180`, `270`), as shown in the figure below. Note that there are no translations of the `Orientation` values of `2`, `4`, `5`, `7`. (These orientations are obtained by flipping image `1` across its vertical center axis to get the image with a value of `2`, and then rotating this image in 90-degree jumps clockwise to get the values of `7`, `4`, and `5`, respectively).



Listing of standard Exif meta tags

- `ImageWidth`
- `ImageLength`
- `BitsPerSample`
- `Compression`
- `PhotometricInterpretation`
- `Orientation`
- `SamplesPerPixel`
- `PlanarConfiguration`
- `YCbCrSubSampling`
- `YCbCrPositioning`
- `XResolution`
- `YResolution`
- `ResolutionUnit`
- `StripOffsets`
- `RowsPerStrip`
- `StripByteCounts`
- `JPEGInterchangeFormat`
- `JPEGInterchangeFormatLength`
- `TransferFunction`
- `WhitePoint`
- `PrimaryChromaticities`
- `YCbCrCoefficients`
- `ReferenceBlackWhite`
- `DateTime`
- `ImageDescription`
- `Make`

- `Model`
- `Software`
- `Artist`
- `Copyright`

`------------------------------`

- `ExifVersion`
- `FlashpixVersion`
- `ColorSpace`
- `ComponentsConfiguration`
- `CompressedBitsPerPixel`
- `PixelXDimension`
- `PixelYDimension`
- `MakerNote`
- `UserComment`
- `RelatedSoundFile`
- `DateTimeOriginal`
- `DateTimeDigitized`
- `SubSecTime`
- `SubSecTimeOriginal`
- `SubSecTimeDigitized`
- `ExposureTime`
- `FNumber`
- `ExposureProgram`
- `SpectralSensitivity`
- `ISOSpeedRatings`
- `OECF`
- `ShutterSpeedValue`
- `ApertureValue`
- `BrightnessValue`
- `ExposureBiasValue`
- `MaxApertureValue`
- `SubjectDistance`
- `MeteringMode`
- `LightSource`
- `Flash`
- `FocalLength`
- `SubjectArea`
- `FlashEnergy`
- `SpatialFrequencyResponse`
- `FocalPlaneXResolution`
- `FocalPlaneYResolution`
- `FocalPlaneResolutionUnit`
- `SubjectLocation`
- `ExposureIndex`
- `SensingMethod`
- `FileSource`
- `SceneType`
- `CFAPattern`
- `CustomRendered`
- `ExposureMode`
- `WhiteBalance`
- `DigitalZoomRatio`
- `FocalLengthIn35mmFilm`
- `SceneCaptureType`

- `GainControl`
- `Contrast`
- `Saturation`
- `Sharpness`
- `DeviceSettingDescription`
- `SubjectDistanceRange`
- `ImageUniqueID`

`------------------------------`

- `GPSVersionID`
- `GPSLatitudeRef`
- `GPSLatitude`
- `GPSLongitudeRef`
- `GPSLongitude`
- `GPSAltitudeRef`
- `GPSAltitude`
- `GPSTimeStamp`
- `GPSSatellites`
- `GPSStatus`
- `GPSMeasureMode`
- `GPSDOP`
- `GPSSpeedRef`
- `GPSSpeed`
- `GPSTrackRef`
- `GPSTrack`
- `GPSImgDirectionRef`
- `GPSImgDirection`
- `GPSMapDatum`
- `GPSDestLatitudeRef`
- `GPSDestLatitude`
- `GPSDestLongitudeRef`
- `GPSDestLongitude`
- `GPSDestBearingRef`
- `GPSDestBearing`
- `GPSDestDistanceRef`
- `GPSDestDistance`
- `GPSProcessingMethod`
- `GPSAreaInformation`
- `GPSDateStamp`
- `GPSDifferential`

[Top 1933]

19.2.2.1.5 XPath/XQuery Functions: Numeric

Altova's numeric extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

Auto-numbering functions

generate-auto-number [altova:]

altova:generate-auto-number(ID *as xs:string*, **StartsWith** *as xs:double*, **Increment** *as xs:double*, **ResetOnChange** *as xs:string***)** as `xs:integer` XP1 XP2 XQ1 XP3.1 XQ3.1

Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the `StartsWith` argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the `Increment` argument. In effect, the `altova:generate-auto-number` function creates a counter having a name specified by the `ID` argument, with this counter being incremented each time the function is called. If the value of the `ResetOnChange` argument changes from that of the previous function call, then the value of the number to be generated is reset to the `StartsWith` value. Auto-numbering can also be reset by using the `altova:reset-auto-number` function.

Examples

- **`altova:generate-auto-number`**`("ChapterNumber", 1, 1, "SomeString")` will return one number each time the function is called, starting with `1`, and incrementing by `1` with each call to the function. As long as the fourth argument remains `"SomeString"` in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called `ChapterNumber`) will reset to 1. The value of `ChapterNumber` can also be reset by a call to the `altova:reset-auto-number` function, like this: `altova:reset-auto-number("ChapterNumber")`.

reset-auto-number [altova:]

altova:reset-auto-number(ID *as xs:string***)** XP1 XP2 XQ1 XP3.1 XQ3.1

This function resets the number of the auto-numbering counter named in the `ID` argument. The number is reset to the number specified by the `StartsWith` argument of the `altova:generate-auto-number` function that created the counter named in the `ID` argument.

Examples

- **`altova:reset-auto-number`**`("ChapterNumber")` resets the number of the auto-numbering counter named `ChapterNumber` that was created by the `altova:generate-auto-number` function. The number is reset to the value of the `StartsWith` argument of the `altova:generate-auto-number` function that created `ChapterNumber`.

[Top[1937]]

Numeric functions

▼ hex-string-to-integer [altova:]

altova:hex-string-to-integer(HexString *as xs:string***) as xs:integer** XP3.1 XQ3.1

Takes a string argument that is the Base-16 equivalent of an integer in the decimal system (Base-10), and returns the decimal integer.

⊟ *Examples*

- `altova:hex-string-to-integer('1')` returns `1`
- `altova:hex-string-to-integer('9')` returns `9`
- `altova:hex-string-to-integer('A')` returns `10`
- `altova:hex-string-to-integer('B')` returns `11`
- `altova:hex-string-to-integer('F')` returns `15`
- `altova:hex-string-to-integer('G')` returns an error
- `altova:hex-string-to-integer('10')` returns `16`
- `altova:hex-string-to-integer('01')` returns `1`
- `altova:hex-string-to-integer('20')` returns `32`
- `altova:hex-string-to-integer('21')` returns `33`
- `altova:hex-string-to-integer('5A')` returns `90`
- `altova:hex-string-to-integer('USA')` returns an error

▼ integer-to-hex-string [altova:]

altova:integer-to-hex-string(Integer *as xs:integer***) as xs:string** XP3.1 XQ3.1

Takes an integer argument and returns its Base-16 equivalent as a string.

⊟ *Examples*

- `altova:integer-to-hex-string(1)` returns `'1'`
- `altova:integer-to-hex-string(9)` returns `'9'`
- `altova:integer-to-hex-string(10)` returns `'A'`
- `altova:integer-to-hex-string(11)` returns `'B'`
- `altova:integer-to-hex-string(15)` returns `'F'`
- `altova:integer-to-hex-string(16)` returns `'10'`
- `altova:integer-to-hex-string(32)` returns `'20'`
- `altova:integer-to-hex-string(33)` returns `'21'`
- `altova:integer-to-hex-string(90)` returns `'5A'`

[Top[1937]]

Number-formatting functions

[Top[1937]]

19.2.2.1.6 XPath/XQuery Functions: Schema

The Altova extension functions listed below return schema information. Given below are descriptions of the functions, together with (i) examples and (ii) a listing of schema components and their respective properties. They can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines and are available in XPath/XQuery contexts.

Schema information from schema documents
The function **`altova:schema`** has two arguments: one with zero arguments and the other with two arguments. The zero-argument function returns the whole schema. You can then, from this starting point, navigate into the schema to locate the schema components you want. The two-argument function returns a specific component kind that is identified by its QName. In both cases, the return value is a function. To navigate into the returned component, you must select a property of that specific component. If the property is a non-atomic item (that is, if it is a component), then you can navigate further by selecting a property of this component. If the selected property is an atomic item, then the value of the item is returned and you cannot navigate any further.

Note: In XPath expressions, the schema must be imported into the processing environment (for example, into XSLT) with the **`xslt:import-schema`** instruction. In XQuery expressions, the schema must be explicitly imported using a schema import.

Schema information from XML nodes
The function **`altova:type`** submits the node of an XML document and returns the node's type information from the PSVI.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, **`http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ Schema (zero arguments)

`altova:schema()` `as (function(xs:string) as item()*)?` XP3.1 XQ3.1
Returns the **schema** component as a whole. You can navigate further into the **schema** component by selecting one of the **schema** component's properties.

- If this property is a component, you can navigate another step deeper by selecting one of this component's properties. This step can be repeated to navigate further into the schema.
- If the component is an atomic value, the atomic value is returned and you cannot navigate any deeper.

The properties of the **schema** component are:

```
"type definitions"
"attribute declarations"
"element declarations"
"attribute group definitions"
"model group definitions"
"notation declarations"
"identity-constraint definitions"
```

The properties of all other component kinds (besides **schema**) are listed below.

Note: In XQuery expressions, the schema must be explicitly imported. In XPath expressions, the schema must have been imported into the processing environment, for example, into XSLT with the **xslt:import** instruction.

⊟ *Examples*

- `import schema "" at "C:\Test\ExpReport.xsd"; for $typedef in altova:schema() ("type definitions")` `return $typedef ("name")` returns the names of all simple types or complex types in the schema

- `import schema "" at "C:\Test\ExpReport.xsd";` `altova:schema() ("type definitions")[1]("name")` returns the name of the first of all simple types or complex types in the schema

Components and their properties

⊟ Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

⊟ Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	

value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"

name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	**Property type**	**Property value**
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	

nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"

name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key"|"unique"|"keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

Schema (two arguments)

altova:schema(ComponentKind *as xs:string*, **Name** *as xs:QName*) **as (function(xs:string) as item()*)?** XP3.1 XQ3.1

Returns the component kind that is specified in the first argument which has a name that is the same as the name supplied in the second argument. You can navigate further by selecting one of the component's properties.

- If this property is a component, you can navigate another step deeper by selecting one of this component's properties. This step can be repeated to navigate further into the schema.
- If the component is an atomic value, the atomic value is returned and you cannot navigate any deeper.

Note: In XQuery expressions, the schema must be explicitly imported. In XPath expressions, the schema must have been imported into the processing environment, for example, into XSLT with the **xslt:import** instruction.

Examples

- ```
  import schema "" at "C:\Test\ExpReport.xsd";
  altova:schema("element declaration", xs:QName("OrgChart"))("type definition")
  ("content type")("particles")[3]!.("term")("kind")
  ```
  returns the `kind` property of the term of the third `particles` component. This particles component is a descendant of the element declaration having a `QName` of `OrgChart`

- ```
  import schema "" at "C:\Test\ExpReport.xsd";
  let $typedef := altova:schema("type definition", xs:QName("emailType"))
  for $facet in $typedef ("facets")
  return [$facet ("kind"), $facet("value")]
  ```
  returns, for each **facet** of each **emailType** component, an array containing that facet's kind and value

Components and their properties

Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing	

	Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

⊟ Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

⊟ Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

⊟ Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

⊟ Complex Type

Property name	Property type	Property value

kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	

value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value

kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key"|"unique"|"keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	

term	Element Declaration, Element Wildcard, or ModelGroup	

⊟ Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

⊟ Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

⊟ XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

Type

altova:type(Node *as item?*) **as (function(xs:string) as item()*)?** XP3.1 XQ3.1

The function **altova:type** submits an element or attribute node of an XML document and returns the node's type information from the PSVI.

Note: The XML document must have a schema declaration so that the schema can be referenced.

Examples

- ```
  for $element in //Email
  let $type := altova:type($element)
  return $type
  ```
  returns a function that contains the `Email` node's type information

- ```
  for $element in //Email
  let $type := altova:type($element)
  return $type ("kind")
  ```
  takes the `Email` node's type component (Simple Type or Complex Type) and returns the value of the component's **kind** property

The **"_props"** parameter returns the properties of the selected component. For example:

- ```
  for $element in //Email
  let $type := altova:type($element)
  return ($type ("kind"), $type ("_props"))
  ```
  takes the `Email` node's type component (Simple Type or Complex Type) and returns (i) the value of the component's **kind** property, and then (ii) the properties of that component.

Components and their properties

Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing	
  ```

	Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Complex Type

Property name	Property type	Property value

kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	

value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value

kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key"|"unique"|"keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

⊟ Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

⊟ Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

⊟ Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

⊟ Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	

term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

19.2.2.1.7 XPath/XQuery Functions: Sequence

Altova's sequence extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, **`http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

attributes [altova:]

`altova:attributes(AttributeName` *`as xs:string`***`) as attribute()*`** XP3.1 XQ3.1
Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis. This means that the context node must be the parent element node.

Examples

- **`altova:attributes`**`("MyAttribute")` returns `MyAttribute()*`

`altova:attributes(AttributeName` *`as xs:string`*, **`SearchOptions`** *`as xs:string`***`) as attribute()*`** XP3.1 XQ3.1
Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis. The context node must be the parent element node. The second argument is a string containing option flags. Available flags are:
`r` = switches to a regular-expression search; `AttributeName` must then be a regular-expression search string;
`f` = If this option is specified, then `AttributeName` provides a full match; otherwise `AttributeName` need only partially match an attribute name to return that attribute. For example: if **`f`** is not specified, then `MyAtt` will return `MyAttribute`;
`i` = switches to a case-insensitive search;
`p` = includes the namespace prefix in the search; `AttributeName` should then contain the namespace prefix, for example: `altova:MyAttribute`.
The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument

(*previous signature*). However, an empty sequence is not allowed as the second argument.

Examples

- **altova:attributes**("MyAttribute", "rfip") returns MyAttribute()*
- **altova:attributes**("MyAttribute", "pri") returns MyAttribute()*
- **altova:attributes**("MyAtt", "rip") returns MyAttribute()*
- **altova:attributes**("MyAttributes", "rfip") returns no match
- **altova:attributes**("MyAttribute", "") returns MyAttribute()*
- **altova:attributes**("MyAttribute", "Rip") returns an unrecognized-flag error.
- **altova:attributes**("MyAttribute",) returns a missing-second-argument error.

elements [altova:]

altova:elements(ElementName *as xs:string***) as element()*** XP3.1 XQ3.1

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The context node must be the parent node of the element/s being searched for.

Examples

- **altova:elements**("MyElement") returns MyElement()*

altova:elements(ElementName *as xs:string*, **SearchOptions** *as xs:string***) as element()*** XP3.1 XQ3.1

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The context node must be the parent node of the element/s being searched for. The second argument is a string containing option flags. Available flags are:

r = switches to a regular-expression search; `ElementName` must then be a regular-expression search string;

f = If this option is specified, then `ElementName` provides a full match; otherwise `ElementName` need only partially match an element name to return that element. For example: if **f** is not specified, then `MyElem` will return `MyElement`;

i = switches to a case-insensitive search;

p = includes the namespace prefix in the search; `ElementName` should then contain the namespace prefix, for example: `altova:MyElement`.

The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed.

Examples

- **altova:elements**("MyElement", "rip") returns MyElement()*
- **altova:elements**("MyElement", "pri") returns MyElement()*
- **altova:elements**("MyElement", "") returns MyElement()*
- **altova:elements**("MyElem", "rip") returns MyElement()*
- **altova:elements**("MyElements", "rfip") returns no match
- **altova:elements**("MyElement", "Rip") returns an unrecognized-flag error.
- **altova:elements**("MyElement",) returns a missing-second-argument error.

find-first [altova:]

altova:find-first((Sequence *as item()**)**,** **(Condition(Sequence-Item** *as xs:boolean*)**) as item()? XP3.1 XQ3.1**

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, `Condition`, is a reference to an XPath function that takes one argument (has an arity of **1**) and returns a boolean. Each item of **Sequence** is submitted, in turn, to the function referenced in `Condition`. (*Remember:* This function takes a single argument.) The first **Sequence** item that causes the function in **Condition** to evaluate to **true()** is returned as the result of **altova:find-first**, and the iteration stops.

Examples

- `altova:find-first(5 to 10, function($a) {$a mod 2 = 0})` returns `xs:integer 6`

The **Condition** argument references the XPath 3.0 inline function, **function()**, which declares an inline function named **$a** and then defines it. Each item in the **Sequence** argument of **altova:find-first** is passed, in turn, to **$a** as its input value. The input value is tested on the condition in the function definition (`$a mod 2 = 0`). The first input value to satisfy this condition is returned as the result of **altova:find-first** (in this case **6**).

- `altova:find-first((1 to 10), (function($a) {$a+3=7}))` returns `xs:integer 4`

Further examples

If the file **C:\Temp\Customers.xml** exists:

- `altova:find-first( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `xs:string C:\Temp\Customers.xml`

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** exists:

- `altova:find-first( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `xs:string http://www.altova.com/index.html`

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** also does not exist:

- `altova:find-first( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns no result

Notes about the examples given above

- The XPath 3.0 function, `doc-available`, takes a single string argument, which is used as a URI, and returns `true` if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The `doc-available` function can be used for **Condition**, the second argument of `altova:find-first`, because it takes only one argument (arity=1), because it takes an `item()` as input (a string which is used as a URI), and returns a boolean value.
- Notice that the `doc-available` function is only referenced, not called. The `#1` suffix that is attached to it indicates a function with an arity of 1. In its entirety `doc-available#1` simply means: *Use the doc-availabe() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence*. As a result, each of the two strings will be passed

to **`doc-available()`**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **`doc-available()`** evaluates to `true()` and that string is returned as the result of the **`altova:find-first`** function. *Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

find-first-combination [altova:]

`altova:find-first-combination((Seq-01` *`as item()*`***`), (Seq-02`** *`as item()*`***`), (Condition( Seq-01-Item, Seq-02-Item`** *`as xs:boolean`***`)) as item()*`** **XP3.1** **XQ3.1**

This function takes three arguments:

- The first two arguments, **`Seq-01`** and **`Seq-02`**, are sequences of one or more items of any datatype.
- The third argument, **`Condition`**, is a reference to an XPath function that takes two arguments (has an arity of **`2`**) and returns a boolean.

The items of **`Seq-01`** and **`Seq-02`** are passed in ordered pairs (one item from each sequence making up a pair) as the arguments of the function in **`Condition`**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X1 Y2), (X1 Y3) ... (X1 Yn), (X2 Y1), (X2 Y2) ... (Xn Yn)
```

The first ordered pair that causes the **`Condition`** function to evaluate to **`true()`** is returned as the result of **`altova:find-first-combination`**. Note that: (i) If the **`Condition`** function iterates through the submitted argument pairs and does not once evaluate to **`true()`**, then **`altova:find-first-combination`** returns *No results*; (ii) The result of **`altova:find-first-combination`** will always be a pair of items (of any datatype) or no item at all.

Examples

- **`altova:find-first-combination`**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns the sequence of `xs:integers (11, 21)`
- **`altova:find-first-combination`**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns the sequence of `xs:integers (11, 22)`
- **`altova:find-first-combination`**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 34})` returns the sequence of `xs:integers (11, 23)`

find-first-pair [altova:]

`altova:find-first-pair((Seq-01` *`as item()*`***`), (Seq-02`** *`as item()*`***`), (Condition( Seq-01-Item, Seq-02-Item`** *`as xs:boolean`***`)) as item()*`** **XP3.1** **XQ3.1**

This function takes three arguments:

- The first two arguments, **`Seq-01`** and **`Seq-02`**, are sequences of one or more items of any datatype.
- The third argument, **`Condition`**, is a reference to an XPath function that takes two arguments (has an arity of **`2`**) and returns a boolean.

The items of **`Seq-01`** and **`Seq-02`** are passed in ordered pairs as the arguments of the function in **`Condition`**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The first ordered pair that causes the **`Condition`** function to evaluate to **`true()`** is returned as the result of **`altova:find-first-pair`**. Note that: (i) If the **`Condition`** function iterates through the submitted argument pairs and does not once evaluate to **`true()`**, then **`altova:find-first-pair`** returns *No results*; (ii) The result of **`altova:find-first-pair`** will always be a pair of items (of any datatype) or no item at all.

Examples

- `altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns the sequence of `xs:integers (11, 21)`
- `altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns *No results*

Notice from the two examples above that the ordering of the pairs is: `(11, 21) (12, 22) (13, 23)...(20, 30)`. This is why the second example returns *No results* (because no ordered pair gives a sum of `33`).

find-first-pair-pos [altova:]

`altova:find-first-pair-pos((Seq-01` *`as item()*`***`), (Seq-02`** *`as item()*`***`), (Condition( Seq-01-Item, Seq-02-Item`** *`as xs:boolean`***`))`** **`as xs:integer`** `XP3.1` `XQ3.1`

This function takes three arguments:

- The first two arguments, **`Seq-01`** and **`Seq-02`**, are sequences of one or more items of any datatype.
- The third argument, **`Condition`**, is a reference to an XPath function that takes two arguments (has an arity of **`2`**) and returns a boolean.

The items of **`Seq-01`** and **`Seq-02`** are passed in ordered pairs as the arguments of the function in **`Condition`**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The index position of the first ordered pair that causes the **`Condition`** function to evaluate to **`true()`** is returned as the result of **`altova:find-first-pair-pos`**. Note that if the **`Condition`** function iterates through the submitted argument pairs and does not once evaluate to **`true()`**, then **`altova:find-first-pair-pos`** returns *No results*.

Examples

- `altova:find-first-pair-pos(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns `1`
- `altova:find-first-pair-pos(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})`

returns *No results*

Notice from the two examples above that the ordering of the pairs is: `(11, 21) (12, 22) (13, 23)...(20, 30)`. In the first example, the first pair causes the **`Condition`** function to evaluate to **`true()`**, and so its index position in the sequence, **`1`**, is returned. The second example returns *No results* because no pair gives a sum of `33`.

▼ find-first-pos [altova:]

`altova:find-first-pos((Sequence` *`as item()*`*`),` **`(Condition( Sequence-Item`** *`as xs:boolean`*`))`
`as xs:integer` **`XP3.1`** **`XQ3.1`**

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, `Condition`, is a reference to an XPath function that takes one argument (has an arity of **`1`**) and returns a boolean. Each item of **`Sequence`** is submitted, in turn, to the function referenced in `Condition`. (*Remember:* This function takes a single argument.) The first **`Sequence`** item that causes the function in **`Condition`** to evaluate to **`true()`** has its index position in **`Sequence`** returned as the result of **`altova:find-first-pos`**, and the iteration stops.

Examples

- **`altova:find-first-pos`**`(5 to 10, function($a) {$a mod 2 = 0})` returns `xs:integer 2`
The **`Condition`** argument references the XPath 3.0 inline function, **`function()`**, which declares an inline function named **`$a`** and then defines it. Each item in the **`Sequence`** argument of **`altova:find-first-pos`** is passed, in turn, to **`$a`** as its input value. The input value is tested on the condition in the function definition (`$a mod 2 = 0`). The index position in the sequence of the first input value to satisfy this condition is returned as the result of **`altova:find-first-pos`** (in this case **`2`**, since **`6`**, the first value (in the sequence) to satisfy the condition, is at index position **`2`** in the sequence).

- **`altova:find-first-pos`**`((2 to 10), (function($a) {$a+3=7}))` returns `xs:integer 3`

Further examples
If the file **`C:\Temp\Customers.xml`** exists:

- **`altova:find-first-pos`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `1`

If the file **`C:\Temp\Customers.xml`** does not exist, and **`http://www.altova.com/index.html`** exists:

- **`altova:find-first-pos`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `2`

If the file **`C:\Temp\Customers.xml`** does not exist, and **`http://www.altova.com/index.html`** also does not exist:

- **`altova:find-first-pos`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns no result

Notes about the examples given above

- The XPath 3.0 function, `doc-available`, takes a single string argument, which is used as a URI, and returns `true` if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The `doc-available` function can be used for **`Condition`**, the second argument of `altova:find-first-pos`, because it takes only one argument (arity=1), because it takes an `item()` as input (a string which is used as a URI), and returns a boolean value.
- Notice that the `doc-available` function is only referenced, not called. The `#1` suffix that is attached to it indicates a function with an arity of 1. In its entirety `doc-available#1` simply means: *Use the doc-availabe() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence*. As a result, each of the two strings will be passed to **`doc-available()`**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **`doc-available()`** function evaluates to `true()` and the index position of that string in the sequence is returned as the result of the **`altova:find-first-pos`** function. *Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

for-each-attribute-pair [altova:]

`altova:for-each-attribute-pair(Seq1` *`as element()?`*, **`Seq2`** *`as element()?`*, **`Function`** *`as function()`*) **`as item()*`** **XP3.1** **XQ3.1**

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then the pair is "disjoint", meaning that it consists of one member only. The function item (third argument `Function`) is applied separately to each pair in the sequence of pairs (joint and disjoint), resulting in an output that is a sequence of items.

Examples

- **`altova:for-each-attribute-pair`**`(/Example/Test-A, /Example/Test-B, function($a, $b){$a+b})` returns ...

  `(2, 4, 6)` if
  ```
  <Test-A att1="1" att2="2" att3="3" />
  <Test-B att1="1" att2="2" att3="3" />
  ```

  `(2, 4, 6)` if
  ```
  <Test-A att2="2" att1="1" att3="3" />
  <Test-B att3="3" att2="2" att1="1" />
  ```

  `(2, 6)` if
  ```
  <Test-A att4="4" att1="1" att3="3" />
  <Test-B att3="3" att2="2" att1="1" />
  ```

  Note: The result **`(2, 6)`** *is obtained by way of the following action:* **`(1+1, ()+2, 3+3, 4+())`**. *If one of the operands is the empty sequence, as in the case of items 2 and 4, then the result of the addition is an empty sequence.*

- **`altova:for-each-attribute-pair`**`(/Example/Test-A, /Example/Test-B, concat#2)` returns

...

```
(11, 22, 33) if
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />

(11, 2, 33, 4) if
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />
```

for-each-combination [altova:]

altova:for-each-combination(FirstSequence *as item()**, **SecondSequence** *as item()**, **Function($i,$j){$i || $j}) as item()*** XP3.1 XQ3.1

The items of the two sequences in the first two arguments are combined so that each item of the first sequence is combined, in order, once with each item of the second sequence. The function given as the third argument is applied to each combination in the resulting sequence, resulting in an output that is a sequence of items (*see example*).

Examples

- `altova:for-each-combination( ('a', 'b', 'c'), ('1', '2', '3'), function($i, $j) {$i || $j} )` returns `('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3')`

for-each-matching-attribute-pair [altova:]

altova:for-each-matching-attribute-pair(Seq1 *as element()?*, **Seq2** *as element()?*, **Function** *as function()*) **as item()*** XP3.1 XQ3.1

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then no pair is built. The function item (third argument `Function`) is applied separately to each pair in the sequence of pairs, resulting in an output that is a sequence of items.

Examples

- `altova:for-each-matching-attribute-pair(/Example/Test-A, /Example/Test-B, function($a, $b){$a+b})` returns ...

```
(2, 4, 6) if
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />

(2, 4, 6) if
<Test-A att2="2" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />

(2, 6) if
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att3="1" />
```

- `altova:for-each-matching-attribute-pair(/Example/Test-A, /Example/Test-B, concat#2)` returns ...

  `(11, 22, 33)` if
  ```
  <Test-A att1="1" att2="2" att3="3" />
  <Test-B att1="1" att2="2" att3="3" />
  ```

  `(11, 33)` if
  ```
  <Test-A att4="4" att1="1" att3="3" />
  <Test-B att3="3" att2="2" att1="1" />
  ```

substitute-empty [altova:]

altova:substitute-empty(FirstSequence *as item()**, **SecondSequence** *as item()*) **as item()*** XP3.1 XQ3.1

If `FirstSequence` is empty, returns `SecondSequence`. If `FirstSequence` is not empty, returns `FirstSequence`.

Examples

- `altova:substitute-empty( (1,2,3), (4,5,6) )` returns `(1,2,3)`
- `altova:substitute-empty( (), (4,5,6) )` returns `(4,5,6)`

19.2.2.1.8 XPath/XQuery Functions: String

Altova's string extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

camel-case [altova:]

altova:camel-case(InputString *as xs:string*) **as xs:string** XP3.1 XQ3.1

Returns the input string **`InputString`** in CamelCase. The string is analyzed using the regular expression **`'\s'`** (which is a shortcut for the whitespace character). The first non-whitespace character after a whitespace or sequence of consecutive whitespaces is capitalized. The first character in the output string is capitalized.

Examples

- **`altova:camel-case`**`("max")` returns `Max`
- **`altova:camel-case`**`("max max")` returns `Max Max`
- **`altova:camel-case`**`("file01.xml")` returns `File01.xml`
- **`altova:camel-case`**`("file01.xml file02.xml")` returns `File01.xml File02.xml`
- **`altova:camel-case`**`("file01.xml   file02.xml")` returns `File01.xml   File02.xml`
- **`altova:camel-case`**`("file01.xml -file02.xml")` returns `File01.xml -file02.xml`

`altova:camel-case(InputString` *`as xs:string`*, **`SplitChars`** *`as xs:string`*, **`IsRegex`** *`as xs:boolean`***`) as xs:string`** **XP3.1** **XQ3.1**

Converts the input string **`InputString`** to camel case by using **`SplitChars`** to determine the character/s that trigger the next capitalization. **`SplitChars`** is used as a regular expression when **`IsRegex = true()`**, or as plain characters when **`IsRegex = false()`**. The first character in the output string is capitalized.

Examples

- **`altova:camel-case`**`("setname getname", "set|get", true())` returns `setName getName`
- **`altova:camel-case`**`("altova\documents\testcases", "\", false())` returns `Altova\Documents\Testcases`

char [altova:]

`altova:char(Position` *`as xs:integer`***`) as xs:string`** **XP3.1** **XQ3.1**

Returns a string containing the character at the position specified by the `Position` argument, in the string obtained by converting the value of the context item to `xs:string`. The result string will be empty if no character exists at the index submitted by the `Position` argument.

Examples

If the context item is **`1234ABCD`**:

- **`altova:char`**`(2)` returns `2`
- **`altova:char`**`(5)` returns `A`
- **`altova:char`**`(9)` returns the empty string.
- **`altova:char`**`(-2)` returns the empty string.

`altova:char(InputString` *`as xs:string`*, **`Position`** *`as xs:integer`***`) as xs:string`** **XP3.1** **XQ3.1**

Returns a string containing the character at the position specified by the `Position` argument, in the string submitted as the `InputString` argument. The result string will be empty if no character exists at the index submitted by the `Position` argument.

Examples

- **`altova:char`**`("2014-01-15", 5)` returns `-`
- **`altova:char`**`("USA", 1)` returns `U`
- **`altova:char`**`("USA", 10)` returns the empty string.
- **`altova:char`**`("USA", -2)` returns the empty string.

create-hash-from-string[altova:]

altova:create-hash-from-string(InputString *as xs:string***) as xs:string** XP2 XQ1 XP3.1 XQ3.1
altova:create-hash-from-string(InputString *as xs:string*, **HashAlgo** *as xs:string***) as xs:string** XP2 XQ1 XP3.1 XQ3.1
Generates a hash string from `InputString` by using the hashing algorithm specified by the `HashAlgo` argument. The following hashing algorithms may be specified (in upper or lower case): **MD5**, **SHA-1**, **SHA-224**, **SHA-256**, **SHA-384**, **SHA-512**. If the second argument is not specified (*see the first signature above*), then the **SHA-256** hashing algorithm is used.

Examples

- `altova:create-hash-from-string('abc')` returns a hash string generated by using the **SHA-256** hashing algorithm.
- `altova:create-hash-from-string('abc', 'md5')` returns a hash string generated by using the **MD5** hashing algorithm.
- `altova:create-hash-from-string('abc', 'MD5')` returns a hash string generated by using the **MD5** hashing algorithm.

first-chars [altova:]

altova:first-chars(X-Number *as xs:integer***) as xs:string** XP3.1 XQ3.1
Returns a string containing the first `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

Examples

If the context item is **1234ABCD**:

- `altova:first-chars(2)` returns `12`
- `altova:first-chars(5)` returns `1234A`
- `altova:first-chars(9)` returns `1234ABCD`

altova:first-chars(InputString *as xs:string*, **X-Number** *as xs:integer***) as xs:string** XP3.1 XQ3.1
Returns a string containing the first `X-Number` of characters of the string submitted as the `InputString` argument.

Examples

- `altova:first-chars("2014-01-15", 5)` returns `2014-`
- `altova:first-chars("USA", 1)` returns `U`

format-string [altova:]

altova:format-string(InputString *as xs:string*, **FormatSequence** *as item()****) as xs:string** XP3.1 XQ3.1
The input string (first argument) contains positional parameters (`%1`, `%2`, etc). Each parameter is replaced by the string item that is located at the corresponding position in the format sequence (submitted as the second argument). So the first item in the format sequence replaces the positional parameter `%1`, the second item replaces `%2`, and so on. The function returns this formatted string that contains the replacements. If no string exists for a positional parameter, then the positional parameter itself is returned. This happens when the index of a positional parameter is greater than the number of items in the format sequence.

Examples

- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe'))` returns `"Hello Jane, John, Joe"`
- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe', 'Tom'))` returns `"Hello Jane, John, Joe"`
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe', 'Tom'))` returns `"Hello Jane, John, Tom"`
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe'))` returns `"Hello Jane, John, %4"`

last-chars [altova:]

`altova:last-chars(X-Number` *`as xs:integer`***`) as xs:string`** **XP3.1** **XQ3.1**
Returns a string containing the last `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

Examples

If the context item is **`1234ABCD`**:

- `altova:last-chars(2)` returns `CD`
- `altova:last-chars(5)` returns `4ABCD`
- `altova:last-chars(9)` returns `1234ABCD`

`altova:last-chars(InputString` *`as xs:string`***`, X-Number`** *`as xs:integer`***`) as xs:string`** **XP3.1** **XQ3.1**
Returns a string containing the last `X-Number` of characters of the string submitted as the `InputString` argument.

Examples

- `altova:last-chars("2014-01-15", 5)` returns `01-15`
- `altova:last-chars("USA", 10)` returns `USA`

pad-string-left [altova:]

`altova:pad-string-left(StringToPad` *`as xs:string`***`, StringLength`** *`as xs:integer`***`, PadCharacter`** *`as xs:string`***`) as xs:string`** **XP3.1** **XQ3.1**
The `PadCharacter` argument is a single character. It is padded to the left of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad`. has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

Examples

- `altova:pad-string-left('AP', 1, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 2, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 3, 'Z')` returns `'ZAP'`
- `altova:pad-string-left('AP', 4, 'Z')` returns `'ZZAP'`
- `altova:pad-string-left('AP', -3, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 3, 'YZ')` returns a pad-character-too-long error

pad-string-right [altova:]

altova:pad-string-right(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string*) **as xs:string** XP3.1 XQ3.1

The `PadCharacter` argument is a single character. It is padded to the right of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad` has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

Examples

- `altova:pad-string-right('AP', 1, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 2, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 3, 'Z')` returns `'APZ'`
- `altova:pad-string-right('AP', 4, 'Z')` returns `'APZZ'`
- `altova:pad-string-right('AP', -3, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 3, 'YZ')` returns a pad-character-too-long error

repeat-string [altova:]

altova:repeat-string(InputString *as xs:string*, **Repeats** *as xs:integer*) **as xs:string** XP2 XQ1 XP3.1 XQ3.1

Generates a string that is composed of the first `InputString` argument repeated `Repeats` number of times.

Examples

- `altova:repeat-string("Altova #", 3)` returns "`Altova #Altova #Altova #`"

substring-after-last [altova:]

altova:substring-after-last(MainString *as xs:string*, **CheckString** *as xs:string*) **as xs:string** XP3.1 XQ3.1

If `CheckString` is found in `MainString`, then the substring that occurs after `CheckString` in `MainString` is returned. If `CheckString` is not found in `MainString`, then the empty string is returned. If `CheckString` is an empty string, then `MainString` is returned in its entirety. If there is more than one occurrence of `CheckString` in `MainString`, then the substring after the last occurrence of `CheckString` is returned.

Examples

- `altova:substring-after-last('ABCDEFGH', 'B')` returns `'CDEFGH'`
- `altova:substring-after-last('ABCDEFGH', 'BC')` returns `'DEFGH'`
- `altova:substring-after-last('ABCDEFGH', 'BD')` returns `''`
- `altova:substring-after-last('ABCDEFGH', 'Z')` returns `''`
- `altova:substring-after-last('ABCDEFGH', '')` returns `'ABCDEFGH'`
- `altova:substring-after-last('ABCD-ABCD', 'B')` returns `'CD'`
- `altova:substring-after-last('ABCD-ABCD-ABCD', 'BCD')` returns `''`

substring-before-last [altova:]

altova:substring-before-last(MainString *as xs:string*, **CheckString** *as xs:string*) **as**

xs:string XP3.1 XQ3.1
If `CheckString` is found in `MainString`, then the substring that occurs before `CheckString` in `MainString` is returned. If `CheckString` is not found in `MainString`, or if `CheckString` is an empty string, then the empty string is returned. If there is more than one occurrence of `CheckString` in `MainString`, then the substring before the last occurrence of `CheckString` is returned.

Examples

- `altova:substring-before-last('ABCDEFGH', 'B')` returns `'A'`
- `altova:substring-before-last('ABCDEFGH', 'BC')` returns `'A'`
- `altova:substring-before-last('ABCDEFGH', 'BD')` returns `''`
- `altova:substring-before-last('ABCDEFGH', 'Z')` returns `''`
- `altova:substring-before-last('ABCDEFGH', '')` returns `''`
- `altova:substring-before-last('ABCD-ABCD', 'B')` returns `'ABCD-A'`
- `altova:substring-before-last('ABCD-ABCD-ABCD', 'ABCD')` returns `'ABCD-ABCD-'`

substring-pos [altova:]

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*) **as xs:integer** XP3.1 XQ3.1
Returns the character position of the first occurrence of `StringToFind` in the string `StringToCheck`. The character position is returned as an integer. The first character of `StringToCheck` has the position `1`. If `StringToFind` does not occur within `StringToCheck`, the integer `0` is returned. To check for the second or a later occurrence of `StringToCheck`, use the next signature of this function.

Examples

- `altova:substring-pos('Altova', 'to')` returns `3`
- `altova:substring-pos('Altova', 'tov')` returns `3`
- `altova:substring-pos('Altova', 'tv')` returns `0`
- `altova:substring-pos('AltovaAltova', 'to')` returns `3`

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*, **Integer** *as xs:integer*) **as xs:integer** XP3.1 XQ3.1
Returns the character position of `StringToFind` in the string, `StringToCheck`. The search for `StringToFind` starts from the character position given by the `Integer` argument; the character substring before this position is not searched. The returned integer, however, is the position of the found string within the *entire* string, `StringToCheck`. This signature is useful for finding the second or a later position of a string that occurs multiple times with the `StringToCheck`. If `StringToFind` does not occur within `StringToCheck`, the integer `0` is returned.

Examples

- `altova:substring-pos('Altova', 'to', 1)` returns `3`
- `altova:substring-pos('Altova', 'to', 3)` returns `3`
- `altova:substring-pos('Altova', 'to', 4)` returns `0`
- `altova:substring-pos('Altova-Altova', 'to', 0)` returns `3`
- `altova:substring-pos('Altova-Altova', 'to', 4)` returns `10`

trim-string [altova:]

altova:trim-string(InputString *as xs:string*) **as xs:string** XP3.1 XQ3.1
This function takes an `xs:string` argument, removes any leading and trailing whitespace, and returns a

"trimmed" `xs:string`.

⊟ *Examples*

- `altova:trim-string("   Hello World   ")` returns `"Hello World"`
- `altova:trim-string("Hello World   ")` returns `"Hello World"`
- `altova:trim-string("   Hello World")` returns `"Hello World"`
- `altova:trim-string("Hello World")` returns `"Hello World"`
- `altova:trim-string("Hello   World")` returns `"Hello   World"`

▼ trim-string-left [altova:]

`altova:trim-string-left(InputString` *`as xs:string`***`) as xs:string`** **XP3.1** **XQ3.1**

This function takes an `xs:string` argument, removes any leading whitespace, and returns a left-trimmed `xs:string`.

⊟ *Examples*

- `altova:trim-string-left("   Hello World   ")` returns `"Hello World   "`
- `altova:trim-string-left("Hello World   ")` returns `"Hello World   "`
- `altova:trim-string-left("   Hello World")` returns `"Hello World"`
- `altova:trim-string-left("Hello World")` returns `"Hello World"`
- `altova:trim-string-left("Hello   World")` returns `"Hello   World"`

▼ trim-string-right [altova:]

`altova:trim-string-right(InputString` *`as xs:string`***`) as xs:string`** **XP3.1** **XQ3.1**

This function takes an `xs:string` argument, removes any trailing whitespace, and returns a right-trimmed `xs:string`.

⊟ *Examples*

- `altova:trim-string-right("   Hello World   "))` returns `"   Hello World"`
- `altova:trim-string-right("Hello World   "))` returns `"Hello World"`
- `altova:trim-string-right("   Hello World"))` returns `"   Hello World"`
- `altova:trim-string-right("Hello World"))` returns `"Hello World"`
- `altova:trim-string-right("Hello   World"))` returns `"Hello   World"`

19.2.2.1.9 XPath/XQuery Functions: Miscellaneous

The following general purpose XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace,** **`http://www.altova.com/xslt-`**

extensions, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

decode-string [altova:]

altova:decode-string(Input *as xs:base64Binary***) as xs:string** XP3.1 XQ3.1
altova:decode-string(Input *as xs:base64Binary***, Encoding** *as xs:string***) as xs:string** XP3.1 XQ3.1

Decodes the submitted base64Binary input to a string using the specified encoding. If no encoding is specified, then the UTF-8 encoding is used. The following encodings are supported: `US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4`

Examples

- `altova:decode-string($XML1/MailData/Meta/b64B)` returns the base64Binary input as a UTF-8 encoded string
- `altova:decode-string($XML1/MailData/Meta/b64B, "UTF-8")` returns the base64Binary input as a UTF-8-encoded string
- `altova:decode-string($XML1/MailData/Meta/b64B, "ISO-8859-1")` returns the base64Binary input as an ISO-8859-1-encoded string

encode-string [altova:]

altova:encode-string(InputString *as xs:string***) as xs:base64Binaryinteger** XP3.1 XQ3.1
altova:encode-string(InputString *as xs:string***, Encoding** *as xs:string***) as xs:base64Binaryinteger** XP3.1 XQ3.1

Encodes the submitted string using, if one is given, the specified encoding. If no encoding is given, then the UTF-8 encoding is used. The encoded string is converted to base64Binary characters, and the converted base64Binary value is returned. Initially, UTF-8 encoding is supported, and support will be extended to the following encodings: `US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4`

Examples

- `altova:encode-string("Altova")` returns the base64Binary equivalent of the UTF-8 encoded string `"Altova"`
- `altova:encode-string("Altova", "UTF-8")` returns the base64Binary equivalent of the UTF-8 encoded string `"Altova"`

get-temp-folder [altova:]

altova:get-temp-folder() as xs:string XP2 XQ1 XP3.1 XQ3.1

This function takes no argument. It returns the path to the temporary folder of the current user.

⊟ *Examples*

- **altova:get-temp-folder**() would return, on a Windows machine, something like `C:\Users\<UserName>\AppData\Local\Temp\` as an `xs:string`.

▼ generate-guid [altova:]

altova:generate-guid() as xs:string XP2 XQ1 XP3.1 XQ3.1

Generates a unique string GUID string.

⊟ *Examples*

- **altova:generate-guid**() returns (for example) `85F971DA-17F3-4E4E-994E-99137873ACCD`

▼ high-res-timer [altova:]

altova:high-res-timer() as xs:double XP3.1 XQ3.1

Returns a system high-resolution timer value in seconds. A high-resolution timer, when present on a system, enables high precision time measurements when these are required (for example, in animations and for determining precise code-execution time). This function provides the resolution of the system's high-res timer.

⊟ *Examples*

- **altova:high-res-timer**() returns something like `'1.16766146154566E6'`

▼ parse-html [altova:]

altova:parse-html(HTMLText *as xs:string*) **as node()** XP3.1 XQ3.1

The `HTMLText` argument is a string that contains the text of an HTML document. The function creates an HTML tree from the string. The submitted string may or may not contain the HTML element. In either case, the root element of the tree is an element named **HTML**. It is best to make sure that the HTML code in the submitted string is valid HTML.

⊟ *Examples*

- **altova:parse-html**(`"<html><head/><body><h1>Header</h1></body></html>"`) creates an HTML tree from the submitted string

▼ sleep[altova:]

altova:sleep(Millisecs *as xs:integer*) **as empty-sequence()** XP2 XQ1 XP3.1 XQ3.1

Suspends execution of the current operation for the number of milliseconds given by the `Millisecs` argument.

⊟ *Examples*

- **altova:sleep**(1000) suspends execution of the current operation for 1000 milliseconds.

[Top 1973]

19.2.2.1.10 Chart Functions

The chart functions listed below enable you to create, generate, and save charts as images. They are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

Note: Chart functions are supported only in **Altova's Server products** and the **Enterprise Editions of Altova products**.

Note: Supported image formats for charts in server editions are `jpg`, `png`, and `bmp`. The best option is png because it is lossless and compressed. In Enterprise editions, the supported formats are `jpg`. `png`, `bmp`, and `gif`.

Functions for generating and saving charts

These functions take the chart object (obtained with the chart creation functions) and either generate an image or save an image to file

`altova:generate-chart-image` `($chart, $width, $height, $encoding) as atomic`

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `x-binarytobase64` or `x-binarytobase16`

The function returns the chart image in the specified encoding.

`altova:generate-chart-image` `($chart, $width, $height, $encoding, $imagetype) as atomic`

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `x-binarytobase64` or `x-binarytobase16`
- `$imagetype` may be one of the following image formats: `png`, `gif`, `bmp`, `jpg`, `jpeg`. Note that `gif` is not supported on server products. *Also see note at top of page*.

The function returns the chart image in the specified encoding and image format.

`altova:save-chart-image` `($chart, $filename, $width, $height) as empty()` ***(Windows only)***

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$filename` is the path to and name of the file to which the chart image is to be saved
- `$width` and `$height` must be specified with a length unit

The function saves the chart image to the file specified in `$filename`. Alternatively to this function, you could also use the `xsl:result-document` function with `encoding="x-base64tobinary"`, where the image-data content is obtained via either the `generate-chart-image()` function or `chart()` function.

`altova:save-chart-image` `($chart, $filename, $width, $height, $imagetype) as empty()` ***(Windows only)***

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$filename` is the path to and name of the file to which the chart image is to be saved
- `$width` and `$height` must be specified with a length unit
- `$imagetype` may be one of the following image formats: `png`, `gif`, `bmp`, `jpg`, `jpeg`. Note that `gif` is not supported on server products. *Also see note at top of page*.

The function saves the chart image to the file specified in `$filename` in the image format specified. Alternatively to this function, you could also use the `xsl:result-document` function with `encoding="x-base64tobinary"`, where the image-data content is obtained via either the `generate-chart-image()` function or `chart()` function.

Functions for creating charts

The following functions are used to create charts.

`altova:create-chart``($chart-config, $chart-data-series*) as chart extension item`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function

The function returns a chart extension item, which is created from the data supplied via the arguments.

`altova:chart``($chart-config, $chart-data-series*) as chart extension item`

where

- `$chart-config` is the chart-config extension item. It is an unordered series of four key: value pairs, where the four keys are `"width"`, `"height"`, `"title"`, and `"kind"`. The values of `width` and `height` are integers and specify the width and height of the chart in pixels. The value of `kind` is one of: `Pie, Pie3d, BarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, RoundGauge, BarGauge`.
- `$chart-data-series` is each an array of size 3, where each array defines a chart-data-series. Each array is composed of: (i) the name of the data series, (ii) the X-Axis values, (iii) the Y-Axis values. Multiple data series may be submitted; in the example below, for example, the two arrays respectively give data for monthly minimum and maximum temperatures.

The function returns an `xs:base64Binary` type item that contains the chart image. This image is created from the data supplied via the arguments of the function. Note that, since this function uses arrays and maps, it can be used only in XPath 3.1, XQuery 3.1, or XSLT 3.0.

Example: `altova:chart( map{'width':800, 'height':600, "kind":"LineChart", "title":"Monthly Temperatures"}, (['Min', $temps/Month, $temps/Month/@min], ['Max', $temps/Month, $temps/Month/@max]) )`

altova:create-chart-config($type-name, $title) as chart-config extension item

where

- `$type-name` specifies the type of chart to be created: `Pie, Pie3d, BarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, RoundGauge, BarGauge`
- `$title` is the name of the chart

The function returns a chart-config extension item containing the configuration information of the chart.

altova:create-chart-config-from-xml($xml-struct) as chart-config extension item

where

- `$xml-struct` is the XML structure containing the configuration information of the chart

The function returns a chart-config extension item containing the configuration information of the chart. This information is supplied in an XML data fragment 1980.

altova:create-chart-data-series($series-name?, $x-values*, $y-values*) as chart-data-series extension item

where

- `$series-name` specifies the name of the series
- `$x-values` gives the list of X-Axis values
- `$y-values` gives the list of Y-Axis values

The function returns a chart-data-series extension item containing the data for building the chart: that is, the names of the series and the Axes data.

altova:create-chart-data-row(x, y1, y2, y3, ...) as chart-data-x-*N*y-row extension item

where

- `x` is the value of the X-Axis column of the chart data row
- `yN` are the values of the Y-Axis columns

The function returns a chart-data-x-Ny-row extension item, which contains the data for the X-Axis column and Y-Axis columns of a single series.

altova:create-chart-data-series-from-rows`($series-names as xs:string*, $row*) as chart-data-series extension item`

where

- `$series-name` is the name of the series to be created
- `$row` is the chart-data-x-Ny-row extension item that is to be created as a series

The function returns a chart-data-series extension item, which contains the data for the X-Axis and Y-Axes of the series.

altova:create-chart-layer`($chart-config, $chart-data-series*) as chart-layer extension item`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function

The function returns a chart-layer extension item, which contains chart-layer data.

altova:create-multi-layer-chart`($chart-config, $chart-data-series*, $chart-layer*)`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function
- `$chart-layer` is the chart-layer extension item obtained with the `altova:create-chart-layer` function

The function returns a multi-layer-chart item.

altova:create-multi-layer-chart`($chart-config, $chart-data-series*, $chart-layer*, xs:boolean $mergecategoryvalues)`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function
- `$chart-layer` is the chart-layer extension item obtained with the `altova:create-chart-layer` function
- `$mergecategoryvalues` merges the values of multiple data series if `true`, does not merge if `false`

The function returns a multi-layer-chart item.

19.2.2.1.10.1 Chart Data XML Structure

Given below is the XML structure of chart data, how it might appear for the Altova extension functions for charts 1976. This affects the appearance of the specific chart. Not all elements are used for all chart kinds, e.g. the `<Pie>` element is ignored for bar charts.

Note: Chart functions are supported only in the **Enterprise and Server Editions** of Altova products.

```
<chart-config>
	<General
	SettingsVersion="1"   must be provided
	ChartKind="BarChart"   Pie, Pie3d, BarChart, StackedBarChart, BarChart3d, BarChart3dGrouped,
LineChart, ValueLineChart, AreaChart, StackedAreaChart, RoundGauge, BarGauge, CandleStick
	BKColor="#ffffff"   Color
	BKColorGradientEnd="#ffffff"   Color. In case of a gradient, BKColor and BKColorGradientEnd
define the gradient's colors
	BKMode="#ffffff"   Solid, HorzGradient, VertGradient
	BKFile="Path+Filename"   String. If file exists, its content is drawn over the background.
	BKFileMode="Stretch"   Stretch, ZoomToFit, Center, Tile
	ShowBorder="1"   Bool
	PlotBorderColor="#000000"   Color
	PlotBKColor="#ffffff"   Color
	Title=""   String
	ShowLegend="1"   Bool
	OutsideMargin="3.%"   PercentOrPixel
	TitleToPlotMargin="3.%"   PercentOrPixel
	LegendToPlotMargin="3.%"   PercentOrPixel
	Orientation="vert"   Enumeration: possible values are: vert, horz
	>
	<TitleFont
	   Color="#000000"   Color
	   Name="Tahoma"   String
	   Bold="1"   Bool
	   Italic="0"   Bool
	   Underline="0"   Bool
	   MinFontHeight="10.pt"   FontSize (only pt values)
	   Size="8.%"   FontSize />
	<LegendFont
	   Color="#000000"
	   Name="Tahoma"
	   Bold="0"
	   Italic="0"
	   Underline="0"
	   MinFontHeight="10.pt"
	   Size="3.5%" />
	<AxisLabelFont
	   Color="#000000"
	   Name="Tahoma"
	   Bold="1"
```

```
      Italic="0"
      Underline="0"
      MinFontHeight="10.pt"
      Size="5.%" />
   </General>

   <Line
         ConnectionShapeSize="1.%"   PercentOrPixel
         DrawFilledConnectionShapes="1"   Bool
         DrawOutlineConnectionShapes="0"   Bool
         DrawSlashConnectionShapes="0"   Bool
         DrawBackslashConnectionShapes="0"   Bool
   />

   <Bar
         ShowShadow="1"   Bool
         ShadowColor="#a0a0a0"   Color
         OutlineColor="#000000"   Color
         ShowOutline="1"   Bool
   />

   <Area
         Transparency="0"   UINT ( 0-255 ) 255 is fully transparent, 0 is opaque
         OutlineColor="#000000"   Color
         ShowOutline="1"   Bool
   />

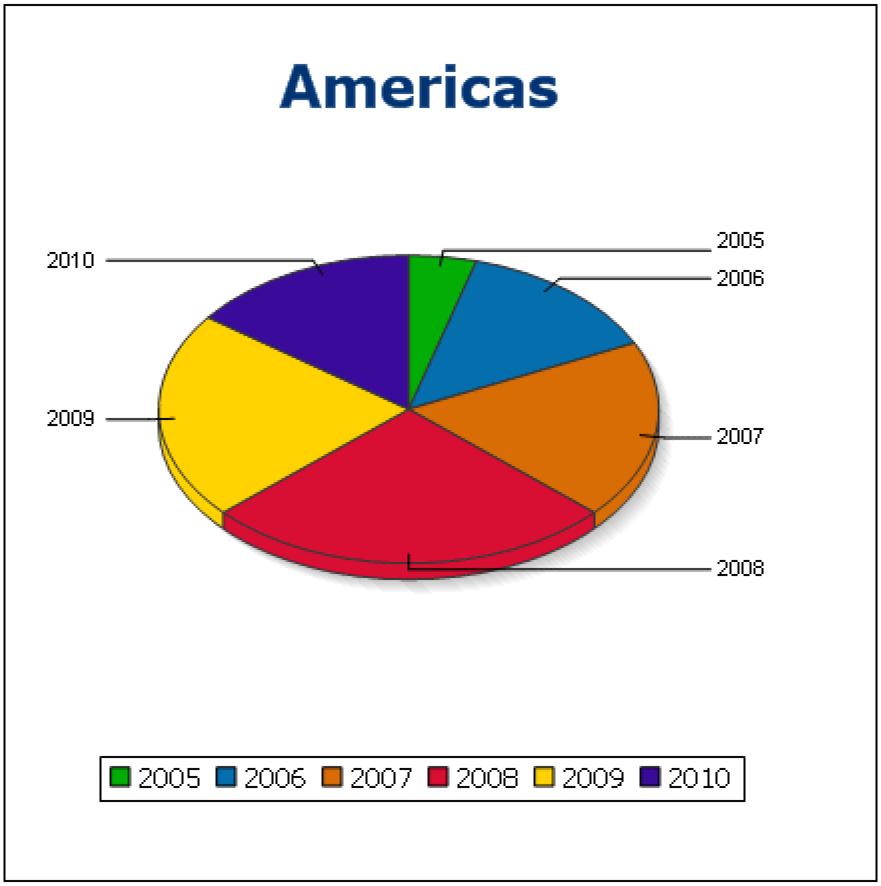
   <CandleStick
         FillHighClose="0"   Bool. If 0, the body is left empty. If 1, FillColorHighClose is used for
the candle body
         FillColorHighClose="#ffffff"   Color. For the candle body when close > open
         FillHighOpenWithSeriesColor="1"   Bool. If true, the series color is used to fill the
candlebody when open > close
         FillColorHighOpen="#000000"   Color. For the candle body when open > close and
FillHighOpenWithSeriesColor is false
   />

   <Colors   User-defined color scheme: By default this element is empty except for the style and has no
Color attributes
         UseSubsequentColors ="1"  Boolean. If 0, then color in overlay is used. If 1, then
subsequent colors from previous chart layer is used
         Style="User"   Possible values are: "Default", "Grayscale", "Colorful", "Pastel", "User"
         Colors="#52aca0"   Color: only added for user defined color set
         Colors1="#d3c15d"   Color: only added for user defined color set
         Colors2="#8971d8"   Color: only added for user defined color set
         ...
         ColorsN=""   Up to ten colors are allowed in a set: from Colors to Colors9
   </Colors>

   <Pie
         ShowLabels="1"   Bool
         OutlineColor="#404040"   Color
         ShowOutline="1"   Bool
```

```
        StartAngle="0."   Double
        Clockwise="1"   Bool
        Draw2dHighlights="1"   Bool
        Transparency="0"   Int (0 to 255: 0 is opaque, 255 is fully transparent)
        DropShadowColor="#c0c0c0"   Color
        DropShadowSize="5.%"   PercentOrPixel
        PieHeight="10.%"   PercentOrPixel. Pixel values might be different in the result because of
3d tilting
        Tilt="40.0"   Double (10 to 90: The 3d tilt in degrees of a 3d pie)
        ShowDropShadow="1"   Bool
        ChartToLabelMargin="10.%"   PercentOrPixel
        AddValueToLabel="0"   Bool
        AddPercentToLabel="0"   Bool
        AddPercentToLabels_DecimalDigits="0" UINT ( 0 – 2 )
        >
        <LabelFont
            Color="#000000"
            Name="Arial"
            Bold="0"
            Italic="0"
            Underline="0"
            MinFontHeight="10.pt"
            Size="4.%"    />
    </Pie>

    <XY>
        <XAxis   Axis
            AutoRange="1"   Bool
            AutoRangeIncludesZero="1"   Bool
            RangeFrom="0."   Double: manual range
            RangeTill="1."   Double : manual range
            LabelToAxisMargin="3.%"   PercentOrPixel
            AxisLabel=""   String
            AxisColor="#000000"   Color
            AxisGridColor="#e6e6e6"   Color
            ShowGrid="1"   Bool
            UseAutoTick="1"   Bool
            ManualTickInterval="1."   Double
            AxisToChartMargin="0.px"   PercentOrPixel
            TickSize="3.px"   PercentOrPixel
            ShowTicks="1"   Bool
            ShowValues="1"   Bool
            AxisPosition="LeftOrBottom"   Enums: "LeftOrBottom", "RightOrTop", "AtValue"
            AxisPositionAtValue = "0"   Double
            >
            <ValueFont
                Color="#000000"
                Name="Tahoma"
                Bold="0"
                Italic="0"
                Underline="0"
                MinFontHeight="10.pt"
                Size="3.%"    />
        </XAxis>
```

```
<YAxis   Axis (same as for XAxis)
        AutoRange="1"
        AutoRangeIncludesZero="1"
        RangeFrom="0."
        RangeTill="1."
        LabelToAxisMargin="3.%"
        AxisLabel=""
        AxisColor="#000000"
        AxisGridColor="#e6e6e6"
        ShowGrid="1"
        UseAutoTick="1"
        ManualTickInterval="1."
        AxisToChartMargin="0.px"
        TickSize="3.px"
        ShowTicks="1"   Bool
        ShowValues="1"   Bool
        AxisPosition="LeftOrBottom"   Enums: "LeftOrBottom", "RightOrTop", "AtValue"
        AxisPositionAtValue = "0"   Double
        >
        <ValueFont
                Color="#000000"
                Name="Tahoma"
                Bold="0"
                Italic="0"
                Underline="0"
                MinFontHeight="10.pt"
                Size="3.%"/>
</YAxis>
</XY>

<XY3d
        AxisAutoSize="1"   Bool: If false, XSize and YSize define the aspect ration of x and y axis.
If true, aspect ratio is equal to chart window
        XSize="100.%"   PercentOrPixel. Pixel values might be different in the result because of 3d
tilting and zooming to fit chart
        YSize="100.%"   PercentOrPixel. Pixel values might be different in the result because of 3d
tilting and zooming to fit chart
        SeriesMargin="30.%"   PercentOrPixel. Pixel values might be different in the result because
of 3d tilting and zooming to fit chart
        Tilt="20."   Double. -90 to +90 degrees
        Rot="20."   Double. -359 to +359 degrees
        FoV="50.">   Double. Field of view: 1-120 degree
        >
        <ZAxis
                AutoRange="1"
                AutoRangeIncludesZero="1"
                RangeFrom="0."
                RangeTill="1."
                LabelToAxisMargin="3.%"
                AxisLabel=""
                AxisColor="#000000"
                AxisGridColor="#e6e6e6"
                ShowGrid="1"
                UseAutoTick="1"
                ManualTickInterval="1."
```

```
                AxisToChartMargin="0.px"
                TickSize="3.px" >
                <ValueFont
                        Color="#000000"
                        Name="Tahoma"
                        Bold="0"
                        Italic="0"
                        Underline="0"
                        MinFontHeight="10.pt"
                        Size="3.%"/>
        </ZAxis>
</XY3d>

<Gauge
        MinVal="0."   Double
        MaxVal="100."   Double
        MinAngle="225"   UINT: -359-359
        SweepAngle="270"   UINT: 1-359
        BorderToTick="1.%"   PercentOrPixel
        MajorTickWidth="3.px"   PercentOrPixel
        MajorTickLength="4.%"   PercentOrPixel
        MinorTickWidth="1.px"   PercentOrPixel
        MinorTickLength="3.%"   PercentOrPixel
        BorderColor="#a0a0a0"   Color
        FillColor="#303535"   Color
        MajorTickColor="#a0c0b0"   Color
        MinorTickColor="#a0c0b0"   Color
        BorderWidth="2.%"   PercentOrPixel
        NeedleBaseWidth="1.5%"   PercentOrPixel
        NeedleBaseRadius="5.%"   PercentOrPixel
        NeedleColor="#f00000"   Color
        NeedleBaseColor="#141414"   Color
        TickToTickValueMargin="5.%"   PercentOrPixel
        MajorTickStep="10."   Double
        MinorTickStep="5."   Double
        RoundGaugeBorderToColorRange="0.%"   PercentOrPixel
        RoundGaugeColorRangeWidth ="6.%"   PercentOrPixel
        BarGaugeRadius="5.%"   PercentOrPixel
        BarGaugeMaxHeight="20.%"   PercentOrPixel
        RoundGaugeNeedleLength="45.%"   PercentOrPixel
        BarGaugeNeedleLength="3.%"   PercentOrPixel
        >
        <TicksFont
                Color="#a0c0b0"
                Name="Tahoma"
                Bold="0"
                Italic="0"
                Underline="0"
                MinFontHeight="10.pt"
                Size="4.%"
        />
        <ColorRanges>   User-defined color ranges. By default empty with no child element entries
                <Entry
                        From="50. "   Double
```

```
                    FillWithColor="1"   Bool
                    Color="#00ff00"   Color
                />
                <Entry
                    From="50.0"
                    FillWithColor="1"
                    Color="#ff0000"
                />
                ...
            </ColorRanges>
        </Gauge>
</chart-config>
```

19.2.2.1.10.2 Example: Chart Functions

The example XSLT document below shows how Altova extension functions for charts [1976] can be used. Given further below are an XML document and a screenshot of the output image generated when the XML document is processed with the XSLT document using the XSLT 2.0 Engine.

Note: Chart functions are supported only in the **Enterprise and Server Editions** of Altova products.

Note: For more information about how chart data tables are created, see the documentation of Altova's XMLSpy and StyleVision products.

XSLT document

This XSLT document (*listing below*) uses Altova chart extension functions to generate a pie chart. It can be used to process the XML document listed further below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:altovaext="http://www.altova.com/xslt-extensions"
   exclude-result-prefixes="#all">
   <xsl:output version="4.0" method="html" indent="yes" encoding="UTF-8"/>
   <xsl:template match="/">
      <html>
         <head>
            <title>
               <xsl:text>HTML Page with Embedded Chart</xsl:text>
            </title>
         </head>
         <body>
            <xsl:for-each select="/Data/Region[1]">
               <xsl:variable name="extChartConfig" as="item()*">
                  <xsl:variable name="ext-chart-settings" as="item()*">
                     <chart-config>
                        <General
                           SettingsVersion="1"
```

```
                              ChartKind="Pie3d"
                              BKColor="#ffffff"
                              ShowBorder="1"
                              PlotBorderColor="#000000"
                              PlotBKColor="#ffffff"
                              Title="{@id}"
                              ShowLegend="1"
                              OutsideMargin="3.2%"
                              TitleToPlotMargin="3.%"
                              LegendToPlotMargin="6.%"
                              >
                              <TitleFont
                                 Color="#023d7d"
                                 Name="Tahoma"
                                 Bold="1"
                                 Italic="0"
                                 Underline="0"
                                 MinFontHeight="10.pt"
                                 Size="8.%" />
                           </General>
                        </chart-config>
                     </xsl:variable>
                     <xsl:sequence select="altovaext:create-chart-config-from-xml( $ext-
chart-settings )"/>
                  </xsl:variable>
                  <xsl:variable name="chartDataSeries" as="item()*">
                     <xsl:variable name="chartDataRows" as="item()*">
                        <xsl:for-each select="(Year)">
                           <xsl:sequence select="altovaext:create-chart-data-row( (@id),
( .) )"/>
                        </xsl:for-each>
                     </xsl:variable>
                     <xsl:variable name="chartDataSeriesNames" as="xs:string*"
select=" ( (&quot;Series 1&quot;), &apos;&apos; )[1]"/>
                     <xsl:sequence
                        select="altovaext:create-chart-data-series-from-
rows( $chartDataSeriesNames, $chartDataRows)"/>
                  </xsl:variable>
                  <xsl:variable name="ChartObj" select="altovaext:create-
chart( $extChartConfig, ( $chartDataSeries), false() )"/>
                  <xsl:variable name="sChartFileName" select="'mychart1.png'"/>
                  <img src="{$sChartFileName, altovaext:save-chart-image( $ChartObj,
$sChartFileName, 400, 400 ) }"/>
               </xsl:for-each>
            </body>
         </html>
   </xsl:template>
</xsl:stylesheet>
```

XML document

This XML document can be processed with the XSLT document above. Data in the XML document is used to generate the pie chart shown in the screenshot below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="YearlySales.xsd">
   <ChartType>Pie Chart 2D</ChartType>
   <Region id="Americas">
      <Year id="2005">30000</Year>
      <Year id="2006">90000</Year>
      <Year id="2007">120000</Year>
      <Year id="2008">180000</Year>
      <Year id="2009">140000</Year>
      <Year id="2010">100000</Year>
   </Region>
   <Region id="Europe">
      <Year id="2005">50000</Year>
      <Year id="2006">60000</Year>
      <Year id="2007">80000</Year>
      <Year id="2008">100000</Year>
      <Year id="2009">95000</Year>
      <Year id="2010">80000</Year>
   </Region>
   <Region id="Asia">
      <Year id="2005">10000</Year>
      <Year id="2006">25000</Year>
      <Year id="2007">70000</Year>
      <Year id="2008">110000</Year>
      <Year id="2009">125000</Year>
      <Year id="2010">150000</Year>
   </Region>
</Data>
```

Output image

The pie chart show below is generated when the XML document listed above is processed with the XSLT document.



19.2.2.2 Miscellaneous Extension Functions

There are several ready-made functions in programming languages such as Java and C# that are not available as XQuery/XPath functions or as XSLT functions. A good example would be the math functions available in Java, such as `sin()` and `cos()`. If these functions were available to the designers of XSLT stylesheets and XQuery queries, it would increase the application area of stylesheets and queries and greatly simplify the tasks of stylesheet creators. The XSLT and XQuery engines used in a number of Altova products support the use of extension functions in Java 1989 and .NET 1997, as well as MSXSL scripts for XSLT 2003. This section describes how to use extension functions and MSXSL scripts in your XSLT stylesheets and XQuery documents. The available extension functions are organized into the following sections:

- Java Extension Functions 1989
- .NET Extension Functions 1997
- MSXSL Scripts for XSLT 2003

The two main issues considered in the descriptions are: (i) how functions in the respective libraries are called; and (ii) what rules are followed for converting arguments in a function call to the required input format of the function, and what rules are followed for the return conversion (function result to XSLT/XQuery data object).

Requirements

For extension functions support, a Java Runtime Environment (for access to Java functions) and .NET Framework 2.0 (minimum, for access to .NET functions) must be installed on the machine running the XSLT transformation or XQuery execution, or must be accessible for the transformations.

19.2.2.2.1 Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

Note the following

- If you are using an Altova desktop product, the Altova application attempts to detect the path to the Java virtual machine automatically, by reading (in this order): (i) the Windows registry, and (ii) the **`JAVA_HOME`** environment variable. You can also add a custom path in the Options dialog of the application; this entry will take priority over any other Java VM path detected automatically.
- If you are running an Altova server product on a Windows machine, the path to the Java virtual machine will be read first from the Windows registry; if this is not successful the **`JAVA_HOME`** environment variable will be used.
- If you are running an Altova server product on a Linux or macOS machine, then make sure that the **`JAVA_HOME`** environment variable is properly set and that the Java Virtual Machines library (on Windows, the **`jvm.dll`** file) can be located in either the **`\bin\server`** or **`\bin\client`** directory.

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with `java:` (*see below for examples*). The namespace declaration should identify a Java class, for example: `xmlns:myns="java:java.lang.Math"`. However, it could also simply be: `xmlns:myns="java"` (without a colon), with the identification of the Java class being left to the `fname()` part of the extension function.
- The `fname()` part identifies the Java method being called, and supplies the arguments for the method (*see below for examples*). However, if the namespace URI identified by the `prefix:` part does not identify a Java class (*see preceding point*), then the Java class should be identified in the `fname()` part, before the class and separated from the class by a period (*see the second XSLT example below*).

Note: The class being called must be on the classpath of the machine.

XSLT example

Here are two examples of how a static method can be called. In the first example, the class name (`java.lang.Math`) is included in the namespace URI and, therefore, must not be in the `fname()` part. In the second example, the `prefix:` part supplies the prefix `java:` while the `fname()` part identifies the class as well as the method.

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
              select="jmath:java.lang.Math.cos(3.14)" />
```

The method named in the extension function (`cos()` in the example above) must match the name of a public static method in the named Java class (`java.lang.Math` in the example above).

XQuery example

Here is an XQuery example similar to the XSLT example above:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

User-defined Java classes

If you have created your own Java classes, methods in these classes are called differently according to: (i) whether the classes are accessed via a JAR file or a class file, and (ii) whether these files (JAR or class) are located in the current directory (the same directory as the XSLT or XQuery document) or not. How to locate these files is described in the sections User-Defined Class Files 1990 and User-Defined Jar Files 1993. Note that paths to class files not in the current directory and to all JAR files must be specified.

19.2.2.2.1.1 User-Defined Class Files

If access is via a class file, then there are four possibilities:

- The class file is in a package. The XSLT or XQuery file is in the same folder as the Java package. (*See example below* 1991.)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. (*See example below* 1991.)
- The class file is in a package. The XSLT or XQuery file is at some random location. (*See example below* 1992.)
- The class file is not packaged. The XSLT or XQuery file is at some random location. (*See example below* 1992.)

Consider the case where the class file is not packaged and is in the same folder as the XSLT or XQuery document. In this case, since all classes in the folder are found, the file location does not need to be specified. The syntax to identify a class is:

```
java:classname
```

where

`java:` indicates that a user-defined Java function is being called; (Java classes in the current directory will be loaded by default)

`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call.

Class file packaged, XSLT/XQuery file in same folder as Java package

The example below calls the `getVehicleType()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is also in the folder `JavaProject`.

```
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/2005/xpath-functions"
      xmlns:car="java:com.altova.extfunc.Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file referenced, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()`method of the `Car` class. Let us say that: (i) the `Car` class file is in the following folder: `JavaProject/com/altova/extfunc`, and (ii) that this folder is the current folder in the example below. The XSLT file is also in the folder `JavaProject/com/altova/extfunc`.

```
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/2005/xpath-functions"
      xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

where

`java:` indicates that a user-defined Java function is being called
`uri-of-package` is the URI of the Java package
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:com.altova.extfunc.Car?path=file:///C:/JavaProject/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

Class file referenced, XSLT/XQuery file at any location

The example below calls the `getCarColor()`method of the `Car` class. Let us say that the `Car` class file is in the folder `C:/JavaProject/com/altova/extfunc`, and the XSLT file is at any location. The location of the class file must then be specified within the namespace URI as a query string. The syntax is:

```
java:classname[?path=<uri-of-classfile>]
```

where

`java:` indicates that a user-defined Java function is being called
`uri-of-classfile` is the URI of the folder containing the class file
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
```

```
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:fn="http://www.w3.org/2005/xpath-functions"
        xmlns:car="java:Car?path=file:///C:/JavaProject/com/altova/extfunc/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

19.2.2.2.1.2 User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

The method is then called by using the prefix of the namespace URI that identifies the class: `classNS:method()`

In the above:

`java:` indicates that a Java function is being called
`classname` is the name of the user-defined class
`?` is the separator between the classname and the path
`path=jar:` indicates that a path to a JAR file is being given
`uri-of-jarfile` is the URI of the jar file
`!/` is the end delimiter of the path
`classNS:method()` is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```
        xmlns:ns1="java:docx.layout.pages?
path=jar:file:///c:/projects/docs/docx.jar!/"
        ns1:main()

        xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
        ns2:docx.layout.pages.main()
```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```
<xsl:stylesheet version="2.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:fn="http://www.w3.org/2005/xpath-functions"
```

```
        xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/" >
    <xsl:output exclude-result-prefixes="fn car xsl xs"/>

    <xsl:template match="/">
        <xsl:variable name="myCar" select="car:Car1.new('red')" />
        <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
    </xsl:template>

    <xsl:template match="car"/>

    </xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

19.2.2.2.1.3 Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be implicitly converted to XPath/XQuery datatypes 1997, then the Java extension function will return a sequence that is an XPath/XQuery datatype. If the result of a Java constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped Java object with a type that is the name of the class returning that Java object. For example, if a constructor for the class `java.util.Date` is called (`java.util.Date.new()`), then an object having a type `java.util.Date` is returned. The lexical format of the returned object may not match the lexical format of an XPath datatype and the value would therefore need to be converted to the lexical format of the required XPath datatype and then to the required XPath datatype.

There are two things that can be done with a Java object created by a constructor:

- It can be assigned to a variable:
  ```
  <xsl:variable name="currentdate" select="date:new()"
  xmlns:date="java:java.util.Date" />
  ```
- It can be passed to an extension function (*see Instance Method and Instance Fields* 1995):
  ```
  <xsl:value-of select="date:toString(date:new())" xmlns:date="java:java.util.Date" />
  ```

19.2.2.2.1.4 Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples

Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`).) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery example

A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

19.2.2.2.1.5 Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()"/>
  <xsl:template match="/">
      <enrollment institution-id="Altova School"
                date="{date:toString($CurrentDate)}"
                type="{jlang:Object.toString(jlang:Object.getClass( date:new() ))}">
      </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the parameter `CurrentDate` takes the value returned by a constructor for the class `java.util.Date`. This value is then passed as an argument to the instance method `date:toString` in order to supply the value of `/enrollment/@date`.

19.2.2.2.1.6 Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (primitive), `java.lang.Boolean`
`xs:integer`	int, long, short, byte, float, double, and the wrapper classes of these, such as `java.lang.Integer`
`xs:float`	`float` (primitive), `java.lang.Float`, `double` (primitive)
`xs:double`	`double` (primitive), `java.lang.Double`
`xs:decimal`	`float` (primitive), `java.lang.Float`, `double`(primitive), `java.lang.Double`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

19.2.2.2.1.7 Datatypes: Java to XPath/XQuery

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.

One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

19.2.2.2.2 .NET Extension Functions

If you are working on the .NET platform on a Windows machine, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

- .NET: Constructors 2000
- .NET: Static Methods and Static Fields 2000
- .NET: Instance Methods and Instance Fields 2001

- Datatypes: XPath/XQuery to .NET 2002
- Datatypes: .NET to XPath/XQuery 2003

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

Parameters

To load an assembly, the following parameters are used:

`asm`	The name of the assembly to be loaded.
`ver`	The version number (maximum of four integers separated by periods).
`sn`	The key token of the assembly's strong name (16 hex digits).
`from`	A URI that gives the location of the assembly (DLL) to be loaded. If the URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.
`partialname`	The partial name of the assembly. It is supplied to `Assembly.LoadWith.PartialName()`, which will attempt to load the assembly. If `partialname` is present, any other parameter is ignored.
`loc`	The locale, for example, `en-US`. The default is `neutral`.

If the assembly is to be loaded from a DLL, use the `from` parameter and omit the `sn` parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the `sn` parameter and omit the `from` parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (*see example below*).

Examples of namespace declarations

An example of a namespace declaration in XSLT that identifies the system class `System.Environment`:

```
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as `Trade.Forward.Scrip`:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class `MyManagedDLL.testClass:`. Two cases are distinguished:

1. When the assembly is loaded from the GAC:
```
declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
       ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. When the assembly is loaded from the DLL (complete and partial references below):
```
       declare namespace cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
       Projects/extFunctions/MyManagedDLL.dll;

declare namespace cs="clitype:MyManagedDLL.testClass?from=MyManagedDLL.dll;
```

XSLT example

Here is a complete XSLT example that calls functions in system class `System.Math`:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes" />
   <xsl:template match="/">
      <math xmlns:math="clitype:System.Math">
         <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
         <pi><xsl:value-of select="math:PI()"/></pi>
         <e><xsl:value-of select="math:E()"/></e>
         <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
      </math>
   </xsl:template>
</xsl:stylesheet>
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`. The extension functions identify methods in the class `System.Math` and supply arguments where required.

XQuery example

Here is an XQuery example fragment similar to the XSLT example above:

```
<math xmlns:math="clitype:System.Math">
   {math:Sqrt(9)}
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this case a system class. The XQuery expression identifies the method to be called and supplies the argument.

19.2.2.2.2.1 .NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a `'No constructor found'` error is returned.

Constructors that return XPath/XQuery datatypes

If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes 1997, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

Constructors that return .NET objects

If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:

- It can be used within a variable:
  ```
  <xsl:variable name="currentdate" select="date:new(2008, 4, 29)"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be passed to an extension function (*see Instance Method and Instance Fields* 1995):
  ```
  <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be converted to a string, number, or boolean:
- ```
  <xsl:value-of select="xs:integer(date:get_Month(date:new(2008, 4, 29)))"
  xmlns:date="clitype:System.DateTime" />
  ```

19.2.2.2.2.2 .NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):
  ```

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

An XSLT example showing a call to a field (considered a method with no argument) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

An XSLT example showing a call to a property (syntax is `get_PropertyName()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length('my string')"
xmlns:string="clitype:System.String"/>
```

An XQuery example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

19.2.2.2.2.3 .NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:variable name="releasedate"
         select="date:new(2008, 4, 29)"
         xmlns:date="clitype:System.DateTime"/>
      <doc>
         <date>
            <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
               xmlns:date="clitype:System.DateTime"/>
         </date>
         <date>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008,` `4, 29)`) as its first and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

19.2.2.2.2.4 Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

`xs:string`	`StringValue, string`
`xs:boolean`	`BooleanValue, bool`
`xs:integer`	`IntegerValue, decimal, long, integer, short, byte, double, float`
`xs:float`	`FloatValue, float, double`
`xs:double`	`DoubleValue, double`
`xs:decimal`	`DecimalValue, decimal, double, float`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

19.2.2.2.2.5 Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`.

When a .NET object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example `System.DateTime.ToString()`) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

19.2.2.2.3 MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (*see example below*).

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (*see below*).

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-prefix">

   function-1 or variable-1
   ...
   function-n or variable-n
```

```
</msxsl:script>
```

The <msxsl:script> element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the <msxsl:script> element. **The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.** Consequently, the .NET scripting languages can be used within the <msxsl:script> element.

The `language` attribute accepts the same values as the `language` attribute on the HTML <script> element. If the `language` attribute is not specified, then Microsoft JScript is assumed as the default.

The implements-prefix attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the <msxsl:script> element will be in the namespace identified by the prefix specified in the `implements-prefix` attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

Example

Here is an example of a complete XSLT stylesheet that uses a function defined within a <msxsl:script> element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
    ' Input: A currency value: the wholesale price
    ' Returns: The retail price: the input value plus 20% margin,
    ' rounded to the nearest cent
    dim a as integer  = 13
    Function AddMargin(WholesalePrice) as integer
      AddMargin = WholesalePrice * 1.2 + a
    End Function
  ]]>
  </msxsl:script>

  <xsl:template match="/">
    <html>
      <body>
        <p>
          <b>Total Retail Price =
            $<xsl:value-of select="user:AddMargin(50)"/>
          </b>
          <br/>
          <b>Total Wholesale Price =
            $<xsl:value-of select="50"/>
          </b>
        </p>
      </body>
```

```
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies

An assembly can be imported into the script by using the **`msxsl:assembly`** element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl:assembly` element is to be used.

```
<msxsl:script>
       <msxsl:assembly name="myAssembly.assemblyName" />
       <msxsl:assembly href="pathToAssembly" />

       ...

</msxsl:script>
```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral PublicKeyToken=a46b3f648229c514"
```

or a short name, such as "`myAssembly.Draw`".

Namespaces

Namespaces can be declared with the **`msxsl:using`** element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl:using` element is used so as to declare namespaces.

```
<msxsl:script>
       <msxsl:using namespace="myAssemblyNS.NamespaceName" />

       ...

</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.

19.3 Technical Data

This section contains information on some technical aspects of your software. This information is organized into the following sections:

- OS and Memory Requirements 2006
- Altova Engines 2006
- Unicode Support 2007
- Internet Usage 2007

19.3.1 OS and Memory Requirements

Operating System

Altova software applications are available for the following platforms:

- Windows 7 SP1 with Platform Update, Windows 8, Windows 10, Windows 11
- Windows Server 2008 R2 SP1 with Platform Update or newer

Memory

Since the software is written in C++ it does not require the overhead of a Java Runtime Environment and typically requires less memory than comparable Java-based applications. However, each document is loaded fully into memory so as to parse it completely and to improve viewing and editing speed. As a result, the memory requirement increases with the size of the document.

Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, available memory can rapidly be depleted.

19.3.2 Altova Engines

XML Validator

When opening an XML document, the application uses its built-in XML validator to check for well-formedness, to validate the document against a schema (if specified), and to build trees and infosets. The XML validator is also used to provide intelligent editing help while you edit documents and to dynamically display any validation error that may occur.

The built-in XML validator implements the Final Recommendation of the W3C's XML Schema 1.0 and 1.1 specifications. New developments recommended by the W3C's XML Schema Working Group are continuously being incorporated in the XML validator, so that Altova products give you a state-of-the-art development environment.

XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0, 2.0, and 3.0 Engines and the Altova XQuery 1.0 and 3.1 Engines. If one of these engines is included in the product, then documentation about implementation-specific behavior for each engine is given in the appendices of the documentation.

Note: Altova MapForce generates code using the XSLT 1.0, 2.0 and XQuery 1.0 engines.

19.3.3 Unicode Support

Altova's XML products provide full Unicode support. To edit an XML document, you will also need a font that supports the Unicode characters being used by that document.

Please note that most fonts only contain a very specific subset of the entire Unicode range and are therefore typically targeted at the corresponding writing system. If some text appears garbled, the reason could be that the font you have selected does not contain the required glyphs. So it is useful to have a font that covers the entire Unicode range, especially when editing XML documents in different languages or writing systems. A typical Unicode font found on Windows PCs is Arial Unicode MS.

In the `/Examples` folder of your application folder you will find an XHTML file called `UnicodeUTF-8.html` that contains the following sentence in a number of different languages and writing systems:

- *When the world wants to talk, it speaks Unicode*
- *Wenn die Welt miteinander spricht, spricht sie Unicode*
- 世界的に話すなら、Unicode です。)

Opening this XHTML file will give you a quick impression of Unicode's possibilities and also indicate what writing systems are supported by the fonts available on your PC.

19.3.4 Internet Usage

Altova applications will initiate Internet connections on your behalf in the following situations:

- If you click the "Request evaluation key-code" in the Registration dialog (**Help | Software Activation**), the three fields in the registration dialog box are transferred to our web server by means of a regular http (port 80) connection and the free evaluation key-code is sent back to the customer via regular SMTP e-mail.
- In some Altova products, you can open a file over the Internet (**File | Open | Switch to URL**). In this case, the document is retrieved using one of the following protocol methods and connections: HTTP (normally port 80), FTP (normally port 20/21), HTTPS (normally port 443). You could also run an HTTP server on port 8080. (In the URL dialog, specify the port after the server name and a colon.)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, the referenced schema document is also retrieved through a HTTP connection (port 80) or another protocol specified in the URL (see Point 2 above). A schema document will also be retrieved when an XML file is validated. Note that validation might happen automatically upon opening a document if you have instructed the application to do this (in the File tab of the Options dialog (**Tools | Options**)).
- In Altova applications using WSDL and SOAP, web service connections are defined by the WSDL documents.
- If you are using the **Send by Mail** command (**File | Send by Mail**) in XMLSpy, the current selection or file is sent by means of any MAPI-compliant mail program installed on the user's PC.
- As part of Software Activation and LiveUpdate as further described in the Altova Software License Agreement.

19.4 License Information

This section contains information about:

- the distribution of this software product
- software activation and license metering
- the license agreement governing the use of this product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

To view the terms of any Altova license, go to the Altova Legal Information page at the Altova website.

19.4.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge for 30 days before making a purchasing decision. *(Note: Altova MobileTogether Designer is licensed free of charge.)*
- Once you decide to buy the software, you can place your order online at the Altova website and get a fully licensed product within minutes.
- When you place an online order, you always get the latest version of our software.
- The product package includes an onscreen help system that can be accessed from within the application interface. The latest version of the user manual is available at www.altova.com in (i) HTML format for online browsing, and (ii) PDF format for download (and to print if you prefer to have the documentation on paper).

30-day evaluation period

After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into the evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you must purchase a product license, which is delivered in the form of a license file containing a key code. Unlock the product by uploading the license file in the Software Activation dialog of your product.

You can purchase product licenses at https://shop.altova.com/.

Helping Others within Your Organization to Evaluate the Software

If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may distribute only the installer file, provided that this file is not modified in any way. Any person who accesses the software installer that you have provided must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.

19.4.2 Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license

When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi-user license

If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (*see the IANA Service Name Registry for details*) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

Note about certificates

Your Altova application contacts the Altova licensing server (`link.altova.com`) via HTTPS. For this communication, Altova uses a registered SSL certificate. If this certificate is replaced (for example, by your IT department or an external agency), then your Altova application will warn you about the connection being insecure. You could use the replacement certificate to start your Altova application, but you would be doing this at your own risk. If you see a *Non-secure connection* warning message, check the origin of the certificate and consult your IT team (who would be able to decide whether the interception and replacement of the Altova certificate should continue or not).

If your organization needs to use its own certificate (for example, to monitor communication to and from client machines), then we recommend that you install Altova's free license management software, Altova LicenseServer, on your network. Under this setup, client machines can continue to use your organization's certificates, while Altova LicenseServer can be allowed to use the Altova certificate for communication with Altova.

19.4.3 Altova End-User License Agreement

- The Altova End-User License Agreement is available here: https://www.altova.com/legal/eula
- Altova's Privacy Policy is available here: https://www.altova.com/privacy

Index

.

A

B

C

D

E

F

G

H

I

M

P

S

T

W

X

Y

Z