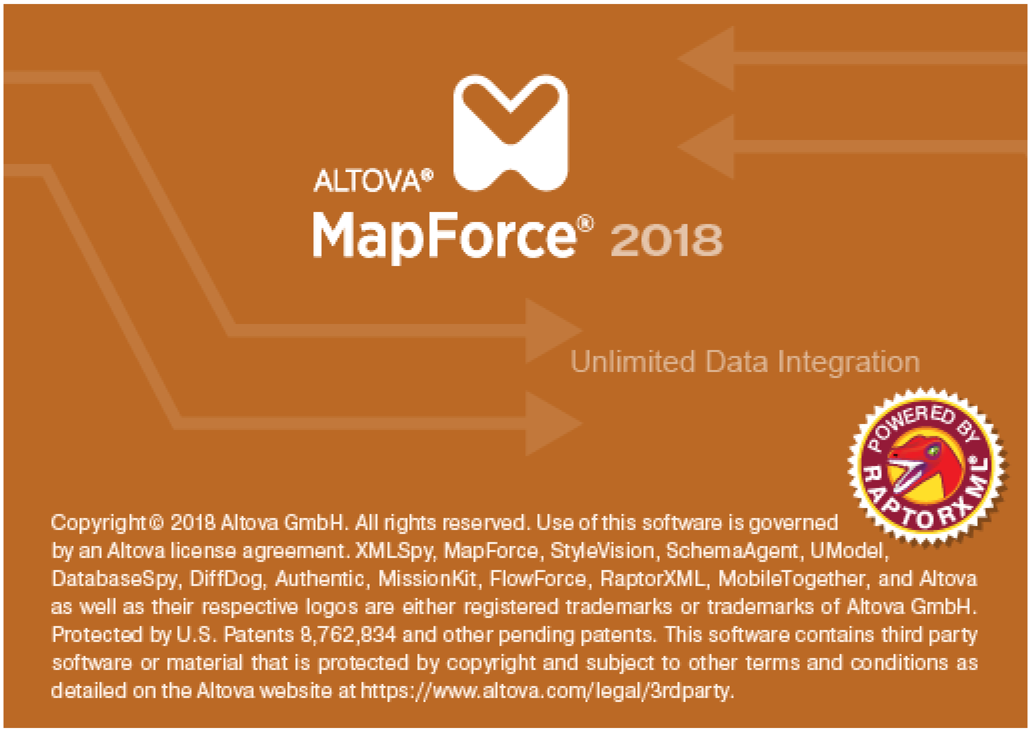
User and Reference Manual



ALTOVA® MapForce® 2018

Unlimited Data Integration

Altova MapForce 2018 Professional Edition
User & Reference Manual

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Published: 2018

Table of Contents

11 MapForce Plug-in for Visual Studio 846

12 MapForce Plug-in for Eclipse 854

13 Menu Reference 882

14 Code Generator 906

15 The MapForce API 1048

17 Appendices 1232

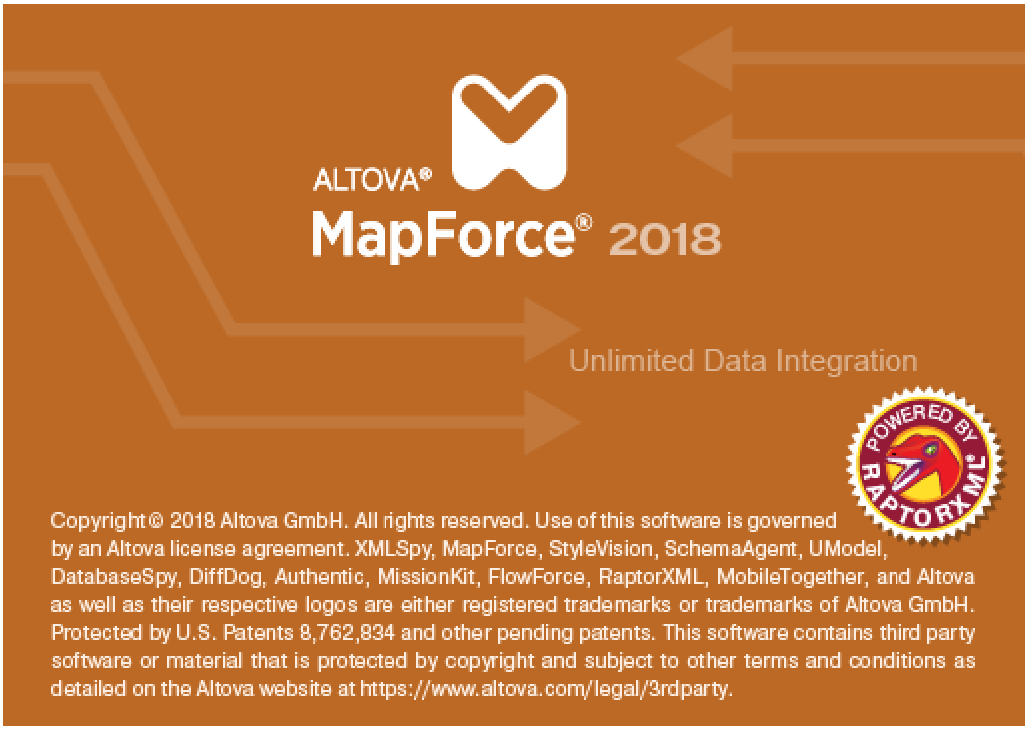
18 Glossary 1312

Chapter 1

Altova MapForce 2018 Professional Edition

1 Altova MapForce 2018 Professional Edition

MapForce® 2018 Professional 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, 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, see Support Notes.



Last updated: 21 June 2018

1.1 What's new...

New in MapForce 2018 Release 2:

- Support for the following databases: MariaDB 10, Teradata 16
- Built-in functions, user-defined functions, and constants can be conveniently added to the mapping by double-clicking an empty area on the mapping (see Add a Built-in Function to the Mapping and Add a Constant to the Mapping)
- Internal updates and optimizations

New in MapForce 2018:

- Support for generating program code for Visual Studio 2013, 2015, and 2017, see Code Generator
- Support for the following database versions: Sybase ASE 16, PostgreSQL 9.6, MySQL 5.7
- Internal updates and optimizations

New in MapForce 2017 Release 3:

- A new component type (Join) has been introduced which can be used to join data from two or more different structures based on custom-defined conditions (see Joining Data). When the mapping reads data from a database, it is possible to join database tables or views in SQL JOIN mode, see Joining Database Data.
- The text search options in the **Output** pane, the **XQuery** pane, as well as the **XSLT** pane have been enhanced (see Searching in Text View). Also, text highlighting is available in the above-mentioned panes (see Text Highlighting).
- Mappings which update databases can be optionally configured to compare data in a NULL-aware manner. NULL-aware comparisons provide a better way (tailored to each specific database) to handle data that contains null values (see Handling Nulls in Database Table Actions).
- In the MapForce ActiveX control, the structure of the `MapForceCommand` object has been enhanced to include a new `Name` property, which can be used to get the unique name of the command. This simplifies retrieving information about MapForce commands programmatically (see Retrieving Command Information).
- Internal updates and optimizations

New in MapForce 2017:

- It is possible to read node names from a source XML (or field names from a CSV/Fixed-length field file) and map this information to a target. It is also possible to dynamically create new XML attributes or elements in a target based on values supplied from a source. See Mapping Node Names.
- XML instance files can be created with custom namespaces, at element level (see Declaring Custom Namespaces)
- MapForce Server execution files (.mfx) can be compiled for specific MapForce Server versions (see Compiling mappings for a specific MapForce Server version)

- Mappings can connect to PostgreSQL databases through native connections (see Setting up a PostgreSQL Connection)
- Mappings can connect to SQL Server and other database types through ADO.NET providers (see Setting up an ADO.NET Connection)
- A new database type is supported: Progress OpenEdge. See Connecting to Progress OpenEdge (ODBC) and Connecting to Progress OpenEdge (JDBC).
- When connecting to a database through JDBC, the search path to .jar libraries can be specified directly in the database connection dialog box (see Setting up a JDBC Connection)
- When a database is updated by the mapping through "Update if... Insert Rest" actions, MERGE statements are created for selected databases (see MERGE statements)
- Internal updates and optimizations

New in MapForce 2016 R2:

- More intuitive code folding in the XSLT pane: collapsed text is displayed with an ellipsis symbol and can be previewed as a tooltip. The same rules apply for text in the XQuery pane and the SQL Editor
- You can search for all occurrences of a function within the active mapping (in the Libraries window, right-click the function, and select **Find All Calls**).
- Internal updates and optimizations

New features in MapForce 2016:

- Improved generation of XSLT 1.0 code (generated stylesheets are easier to read and often faster to execute)
- Two new aggregate functions are available in the MapForce core library: **`min-string`** and **`max-string`**. These functions enable you to get the minimum or maximum value from a sequence of strings.
- Mappings written for the Built-in execution engine can be debugged (see Debugging Mappings)
- The MapForce Plug-in for Visual Studio supports Visual Studio 2015 (adds to support for previous versions)
- New database versions are supported: SQL Server 2014, Oracle 12c, IBM DB2 10.5, PostgreSQL 9.4, MySQL 5.6 (adds to support for previous versions)
- Firebird databases are supported (see Connecting to Firebird (ODBC) and Connecting to Firebird (JDBC))

New features in MapForce Version 2015 R4:

- In the MapForce plug-in for Eclipse, the commands specific to MapForce files are now available under a new **MapForce** menu (see Accessing Common Menus and Functions)
- Internal updates and optimizations

New features in MapForce Version 2015 R3 include:

- Option to suppress the `<?xml ... ?>` declaration in XML output
- Text-based components (including EDI, CSV, fixed-length field, JSON, and XML) can parse and serialize strings in addition to plain files
- SQLite database support (see Setting up a SQLite connection)

- New string padding functions: pad-string-left and pad-string-right
- New component type: Simple Output
- Internal updates and optimizations

New features in MapForce Version 2015 include:

- New `language` argument available in the format-date and format-dateTime functions
- New sequence function: replicate-item

New features in MapForce Version 2014 R2 include:

- New sequence functions: generate sequence, item-at, etc.
- Ability to define CDATA sections in output components
- Ability to define timeout values for database execution
- Keeping connections after deleting components
- Bulk transfer of database data (bulk Insert all)
- Automatic highlighting of mandatory items in target components

New features in MapForce Version 2014 include:

- Integration of RaptorXML validator and basic support for XML Schema 1.1
- Integration of new RaptorXML XSLT and XQuery engines
- XML Schema Wildcard support, xs:any and xs:anyAttribute
- Support for Comments and Processing Instructions in XML target components
- Age function
- Ability to always insert quote character for CSV files

New features in MapForce Version 2013 R2 SP1 include:

- New super-fast transformation engine RaptorXML Server

New features in MapForce Version 2013 R2 include:

- MapForce Server support.
- Ability to generate a MapForce Server execution file from the command line and File menu, to be executed by MapForce Server.
- Ability to deploy MapForce mappings to FlowForce Server.
- Support for Informix 11.7 databases, and extended support for other databases.
- User defined end-of-line settings for output files.
- Internal updates and optimizations.

New features in MapForce Version 2013 include:

- Ability to call stored procedures in mappings
- Support for database functions (functionally similar to stored procedures)
- Support for SELECT statements with parameters
- Internal updates and optimizations

New features in MapForce Version 2012 R2 include:

- New Sort component for XSLT 2.0, XQuery, and the Built-in execution engine
- User defined component names
- Extended SQL-Where functionality: ORDER BY
- MapForce supports logical files of the IBM iSeries database and shows logical files as views
- Support for IBM DB2 logical files. A logical file in IBM iSeries editions of the DB2 database represents one or more physical files. A logical file allows users to access data in a sequence or format that can be different from the physical file. Users who connect to IBM iSeries computers may encounter existing databases constructed with logical files. These were previously not accessible, but are now supported in Version 2012 Release 2.

New features in MapForce Version 2012 include:

- Data streaming for XML, CSV and fixed-length field files (when using the built-in execution engine)
- New database engine supports direct ODBC and JDBC connections
- Auto-alignment of components in the mapping window
- New functions: parse-date and parse-time
- Find items in Project tab/window
- Prompt to connect to target parent node
- Specific rules governing the sequence that components are processed in a mapping
- New Programming Languages examples section in MapForce API

New features in MapForce Version 2011R3 include:

- Intermediate variablesSupport for .NET Framework 4.0 assembly files
- Ability to output StyleVision formatted documents from the command line

New features in MapForce Version 2011R2 include:

- Built-in Execution Engine now supports streaming output
- Find function capability in Library window
- Reverse mapping
- Extendable IF-ELSE function
- Node Name and parsing functions in Core LibraryImproved database table actions dialog with integrated key generation settings
- New option of using StyleVision Power Stylesheets when documenting a mapping

New features in MapForce Version 2011 include:

- Ability to preview intermediate components in a mapping chain of two or more components connected to a target component (pass-through preview).
- Formatting functions for dateTime and numbers for all supported languages
- Enhancement to auto-number function
- New timezone functions: remove-timezone and convert-to-utc

- Ability to preview target components using StyleVision Power Stylesheets containing StyleVision Charts

New features in MapForce Version 2010 Release 3 include:

- Support for generation of Visual Studio 2010 project files for C# and C++ added
- Support for MSXML 6.0 in generated C++ code
- Support for Nillable values, and xsi:nil attribute in XML instance files
- Ability to disable automatic casting to target types in XML documents

New features in MapForce Version 2010 Release 2 include:

- 64-bit MapForce Enterprise / Professional editions on 64-bit operating systemsAutomatic connection of identical child connections when moving a parent connection
- Support for fields in the SQL Where component
- Ability to add compiled Java .class and .NET assembly files
- Ability to tokenize input strings for further processing

New features in MapForce Version 2010 include:

- Multiple input/output files per component
- Upgraded relative path support
- xsi:type support allowing use of derived types
- New internal data type system
- Improved user-defined function navigation
- Enhanced handling of mixed content in XML elements

New features in MapForce Version 2009 SP1 include:

- Parameter order in user-defined functions can be user-defined
- Ability to process XML files that are not valid against XML Schema
- Regular (Standard) user-defined functions now support complex hierarchical parameters
- Apache Xerces 3.x support when generating C++ code

New features in MapForce Version 2009 include:

- EDI HL7 versions 3.x XML as source and target components
- Documentation of mapping projects
- Native support for XML fields in SQL Server
- Grouping of nodes or node content
- Ability to filter data based on a nodes position in a sequence
- QName support
- Item/node search in components

New features in MapForce Version 2008 Release 2 include:

- Ability to automatically generate XML Schemas for XML files
- Support for stream objects as input/output in generated Java and C# code

- Generation of Visual Studio 2008 project files for C++ and C#
- Ability to strip database schema names from generated code
- SQL SELECT Statements as virtual tables in database components
- Local Relations - on-the-fly creation of primary/foreign key relationships
- Support for Altova Global Resources
- Performance optimizations

New features in MapForce Version 2008 include:

- Aggregate functions
- Value-Map lookup component
- Enhanced XML output options: pretty print XML output, omit XML schema reference and Encoding settings for individual components
- Various internal updates

New features in MapForce Version 2007 Release 3 include:

- XML data mapping to/from database fields (see Mapping XML Data to / from Database Fields)
- Direct querying of databases
- SQL-WHERE filter and SQL statement wizard
- Code generator optimization and improved documentation

Chapter 2

Introduction

2 Introduction

This introduction includes an overview of the MapForce features and user interface, the basic concepts in MapForce, as well as the conventions used in this documentation.

2.1 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 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 2008/2010/2012/2013/2015/2017, see MapForce Plug-in for Visual Studio
- Eclipse 4.5 / 4.6 / 4.7, see MapForce Plug-in for Eclipse.

MapForce integrates with Microsoft Office products as follows:

- It can map data to or from Access databases. For supported versions, see Databases and MapForce
- It can generate mapping documentation in Word 2000 or later format, see Generating and Customizing Mapping Documentation.

For support information applicable to program code generation, see Introduction to Code Generator.

For other technical information, see Technical Data.

2.2 What Is MapForce?

Altova website: Data mapping tool

MapForce is a Windows-based, multi-purpose IDE (integrated development environment) that enables you to transform data from one format to another, or from one schema to another, by means of a visual, "drag-and-drop" -style graphical user interface that does not require writing any program code. In fact, MapForce generates for you the program code which performs the actual data transformation (or data mapping). When you prefer not to generate program code, you can just run the transformation using the MapForce built-in transformation language (available in the MapForce Professional or Enterprise Editions).

Mappings designed with MapForce enable you to conveniently convert and transform data from and to a variety of file-based and other formats. Regardless of the technology you work with, MapForce determines automatically the structure of your data, or gives you the option to supply a schema for your data, or generate it automatically 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 making the data inside the XML file available for mapping to other files or formats.

The technologies supported as mapping sources or targets are as follows.

MapForce Basic Edition	MapForce Professional Edition	MapForce Enterprise Edition
• XML and XML schema • HL7 version 3.x (schema-based)	• 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)	• 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 • Databases (all major relational databases, including Microsoft Access and SQLite databases) • 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

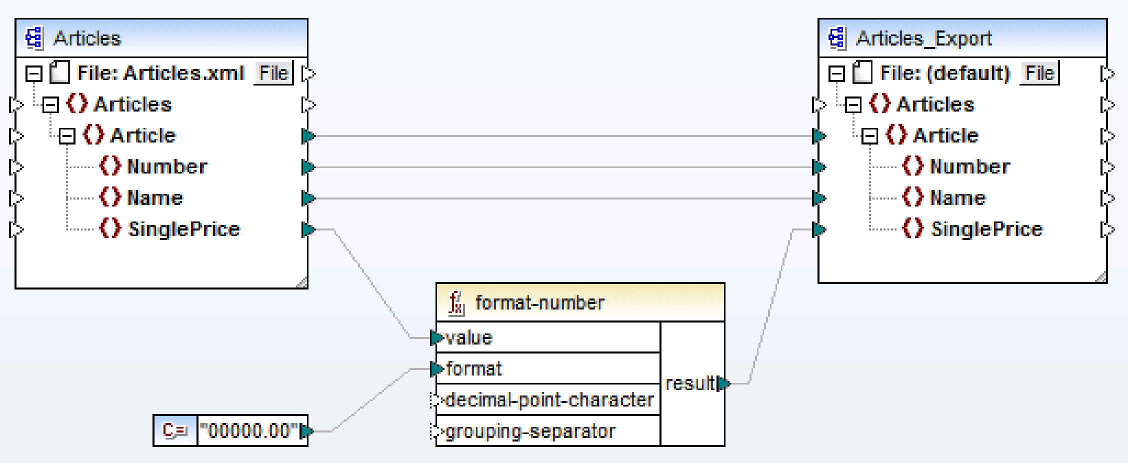
Based on the MapForce edition, you can choose the preferred language for your data transformation as follows.

MapForce Basic Edition	MapForce Professional Edition	MapForce Enterprise Edition
• XSLT 1.0 • XSLT 2.0	• MapForce built-In transformation language • XSLT 1.0 • XSLT 2.0 • XQuery • Java • C# • C++	• MapForce built-In transformation language • XSLT 1.0 • XSLT 2.0 • XQuery • Java • C# • C++

You can preview the result of all transformations, as well as the generated XSLT or XQuery code without leaving the graphical user interface. Note that, as you design or preview mappings, MapForce validates the integrity of your schemas or transformations and displays any validation errors in a dedicated window, so that you can immediately review and address them.

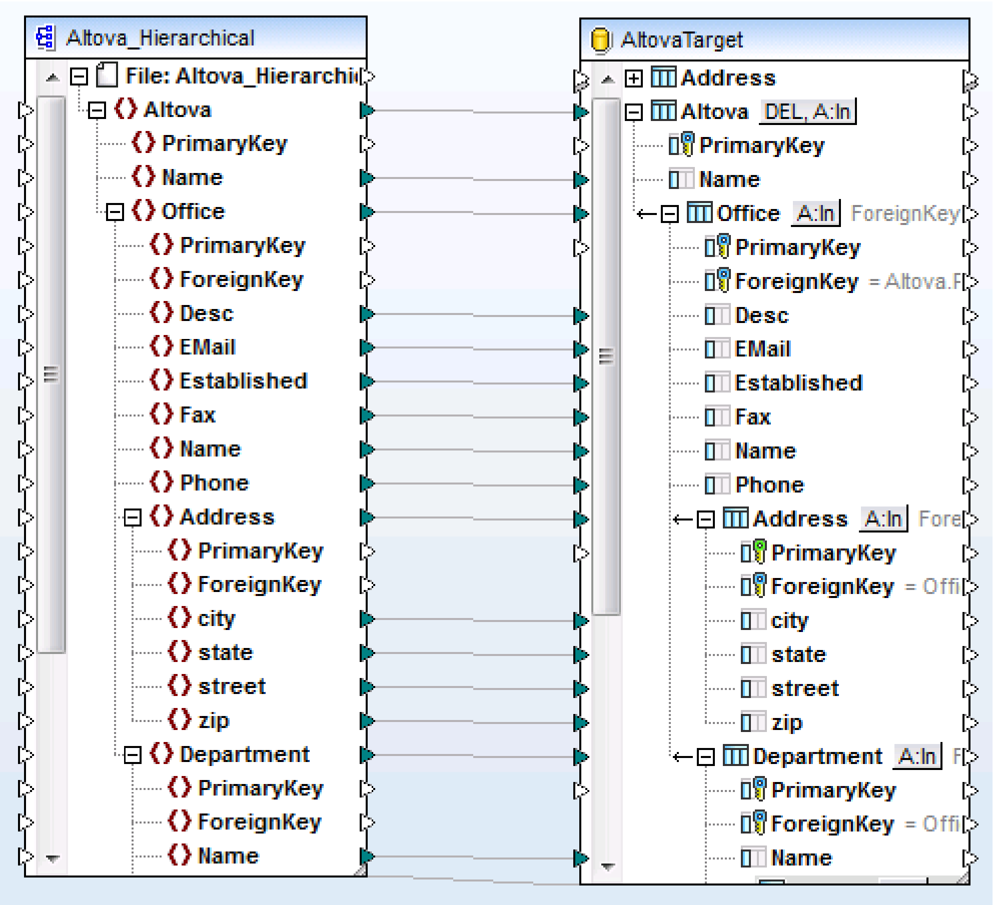
When you choose Java, C#, or C++ as transformation language, MapForce generates the required projects and solutions so that you can open them directly in Visual Studio or Eclipse, and run the generated data mapping program. For advanced data integration scenarios, you can also extend the generated program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

In MapForce, you design all mapping transformations visually. For example, in case of XML, you can connect any element, attribute, or comment in an XML file to an element or attribute of another XML file, thus instructing MapForce to read data from the source element (or attribute), and write it to the target element (or attribute).



Sample data transformation between two XML files

Likewise, when working 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 setting up a database connection from MapForce, you can flexibly choose the database driver and the connection type (ADO, ODBC, or JDBC) according to your existing infrastructure and data mapping needs. Additionally, you can visually build SQL queries, use stored procedures, or query a database directly (support varies by database type, edition and driver).



Sample data transformation between an XML file and a database

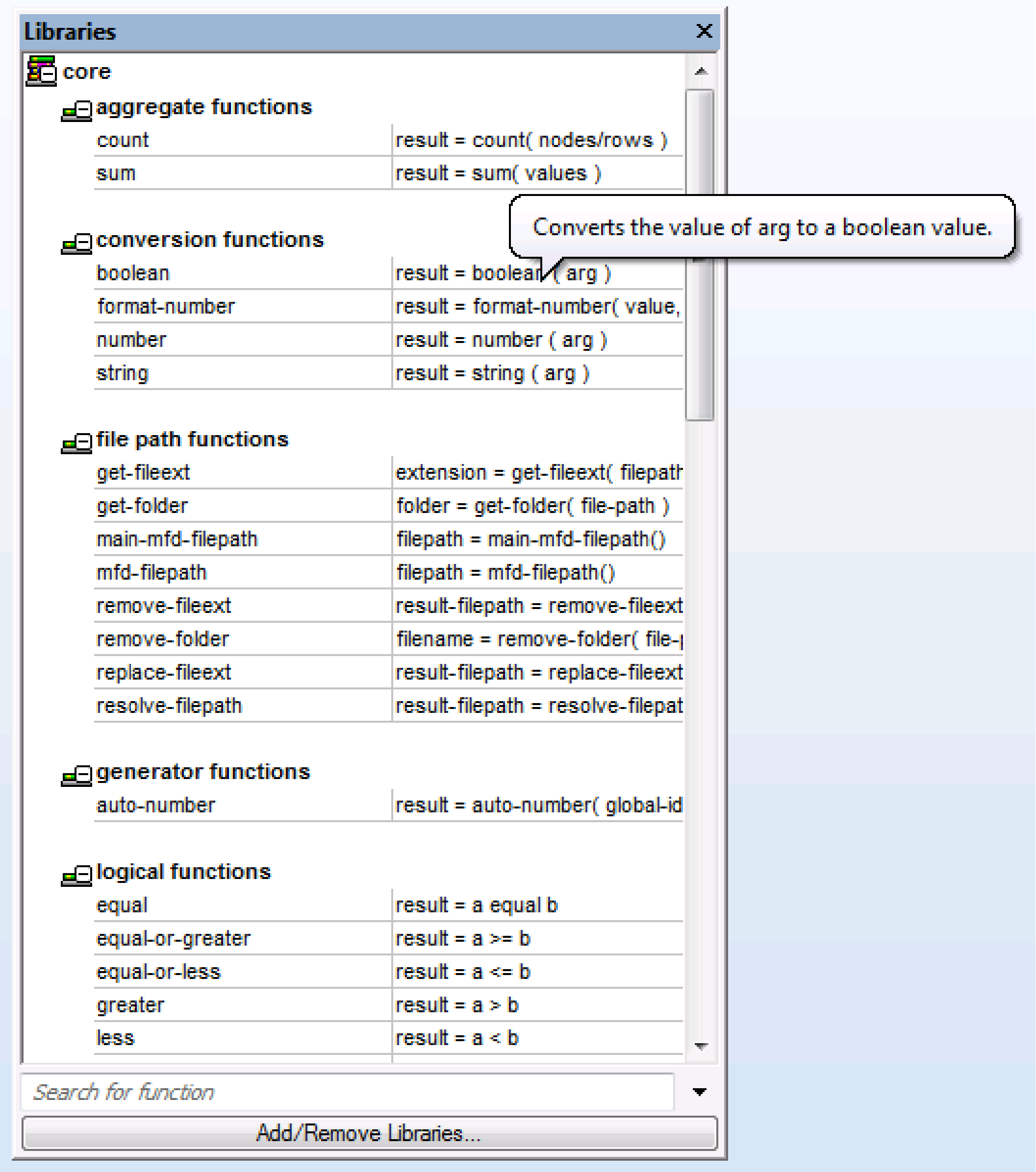
In a very simple scenario, a mapping design created with MapForce could be resumed as "read data from the source X and write it to target Y". However, you can easily design MapForce scenarios such as "read data from the source X and write it to target Y, and then read data from the source Y and write it to the target Z". These are known as "pass-through", or "chained" mappings, and enable you to access your data at an intermediary stage in the transformation process (in order to save it to a file, for example).

Note that the data mappings you can create in MapForce are not limited to single, predefined files. In the same transformation, you can process dynamically multiple input files from a directory and generate multiple output files. Therefore, you can have scenarios such as "read data from multiple X files and write it to a single Y file", or "read file X and generate multiple files Y", and so on.

Importantly, in the same transformation, you can mix multiple sources and multiple targets, which can be of any type supported by your MapForce edition. For example, in case of MapForce Professional or Enterprise, this makes it possible to merge data from two different databases into a single XML file. Or, you can merge data from multiple XML files, and write some of the data to one database, and some of the data to another database. You can preview the SQL statements before committing them to the database.

Direct conversion of data from a source to a target is not typically the only thing you want to achieve. In many cases, you might want to process your data in a particular way (for example, sort, group or filter it) before it reaches the destination. For this reason, MapForce includes, on one hand, miscellaneous functional components that are simplified programming language constructs (such as constants, variables, SQL-WHERE conditions, Filter and Sort components). On the other hand, MapForce includes rich and extensible function libraries which can assist you with virtually any kind of data manipulation.

If necessary, you can extend the built-in library either with functions you design in MapForce directly (the so-called User-Defined Functions, or UDF), or with functions or libraries created externally in XSLT, XQuery, Java, or C# languages.



aggregate functions

count	result = count(nodes/rows)
sum	result = sum(values)

Converts the value of arg to a boolean value.

conversion functions

boolean	result = boolean(arg)
format-number	result = format-number(value,
number	result = number (arg)
string	result = string (arg)

file path functions

get-fileext	extension = get-fileext(filepath
get-folder	folder = get-folder(file-path)
main-mfd-filepath	filepath = main-mfd-filepath()
mfd-filepath	filepath = mfd-filepath()
remove-fileext	result-filepath = remove-fileext
remove-folder	filename = remove-folder(file-p
replace-fileext	result-filepath = replace-fileext
resolve-filepath	result-filepath = resolve-filepat

generator functions

auto-number	result = auto-number(global-id

logical functions

equal	result = a equal b
equal-or-greater	result = a >= b
equal-or-less	result = a <= b
greater	result = a > b
less	result = a < b

Libraries pane (MapForce Basic Edition)

When your data mapping design files become too many, you can organize them into mapping projects (available in MapForce Professional and Enterprise edition). This allows for easier access and management. Importantly, you can generate program code from entire projects, in addition to generating code for individual mappings within the project.

For advanced data processing needs (such as when running mapping transformations with the MapForce Server API), you can design a mapping so that you can pass values to it at run-time, or get a simple string value from it at run-time. This feature also enables you to quickly test the output of functions or entire mappings that produce a simple string value. The Professional and Enterprise editions of MapForce also include components that enable you to perform run-time string parsing and serialization, similar to how this works in many other programming languages.

With MapForce Enterprise Edition, you can visually design SOAP 1.0 and SOAP 2.0 Web services based on Web Service Language Definition (WSDL) files. You can also call and get data from a WSDL 1.0 or a WSDL 2.0 Web service from within a mapping. This includes Web services available both through the HTTP and HTTPS protocols, as well as Web services which require that the caller uses the WS-Security mechanism, or HTTP authentication.

With MapForce Professional and Enterprise Editions, you can generate detailed documentation of your mapping design files, in HTML, Word 2007+, or RTF formats. Documentation design can be customized (for example, you can choose to include or exclude specific components from the documentation).

If you are using MapForce alongside other Altova MissionKit products, MapForce integrates with them as well as with the Altova server-based products, as shown in the following table.

<table>
<tr><th>MapForce Basic Edition</th><th>MapForce Professional Edition</th><th>MapForce Enterprise Edition</th></tr>
<tr><td colspan="3">You can choose to run the generated XSLT 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.</td></tr>
<tr><td colspan="3">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 <b>Component | Edit Schema Definition in XMLSpy</b> menu command is available when you click an XML component).</td></tr>
<tr><td></td><td colspan="2">You can run data transformations either directly in MapForce, or deploy them to a different machine and even operating system for command-line or automated execution. More specifically, you can design mappings on Windows, and run them on a Windows, Linux, or Mac server machine which runs MapForce Server (either standalone or under FlowForce Server management).</td></tr>
<tr><td></td><td colspan="2">If StyleVision is installed on the same machine, 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.</td></tr>
</table>

MapForce Professional and Enterprise edition 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 MapForce functionality without leaving your preferred development environment.

In MapForce, you can completely customize not only the look and feel of the development environment (graphical user interface), but also various other settings pertaining to each technology and to each mapping component type, for example:

- When mapping to or from XML, you can choose whether to include a schema reference, or 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, whether MapForce should use database transactions, or whether it should strip the database schema name from table names when generating code.
- In case of XBRL, you can select the structure views MapForce should display (such as the "Presentation and definition linkbases" view, the "Table Linkbase" View, or the "All concepts" view).

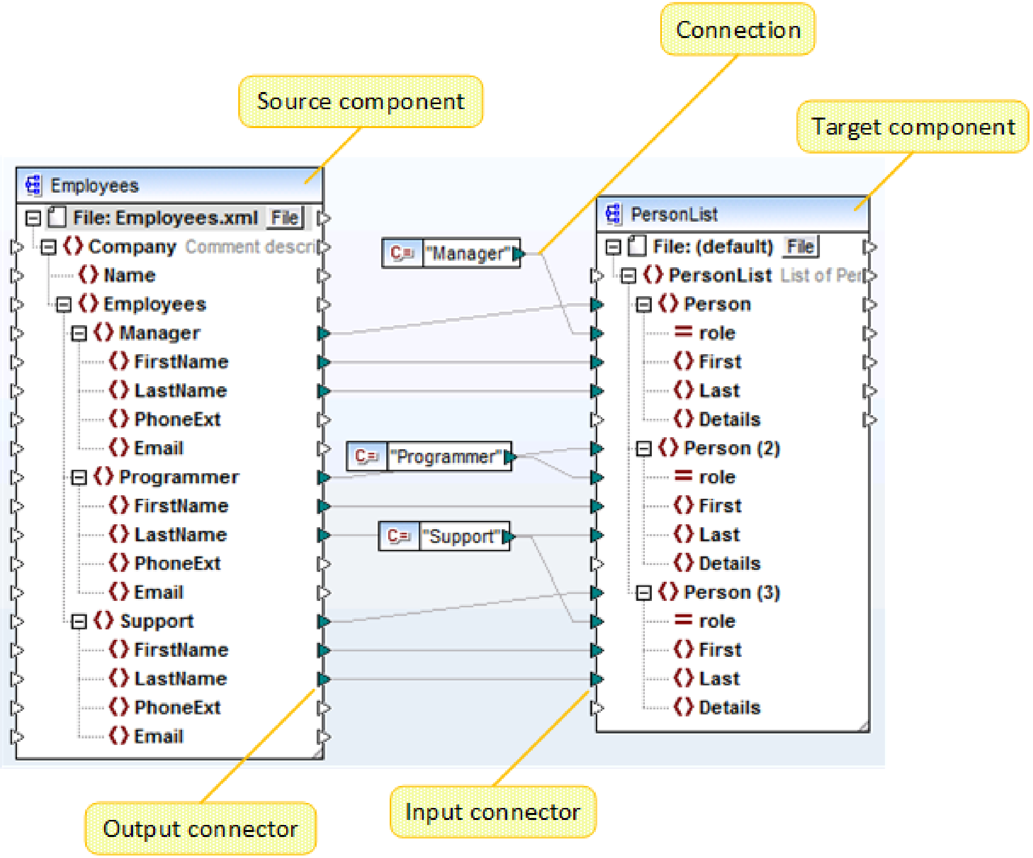
All editions of MapForce are available as a 32-bit application. The MapForce Professional and Enterprise editions are additionally available as a 64-bit application.

2.3 Basic Concepts

This section outlines the basic concepts that will help you get started with data mapping.

Mapping

A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). 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 mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.



Basic structure of a MapForce mapping

Component

In MapForce, the term "component" is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Databases
- Filters
- Conditions
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- XML Schemas and DTDs

Connector

A connector is a small triangle displayed on the left or right side of a component. The connectors displayed on the left of a component provide data entry points *to that component*. The connectors displayed on the right of a component provide data exit points *from that component*.

Connection

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

Source component

A source component is a component from which MapForce reads data. When you run the mapping, MapForce reads the data supplied by the connector of the source component, converts it to the required type, and sends it to the connector of the target component.

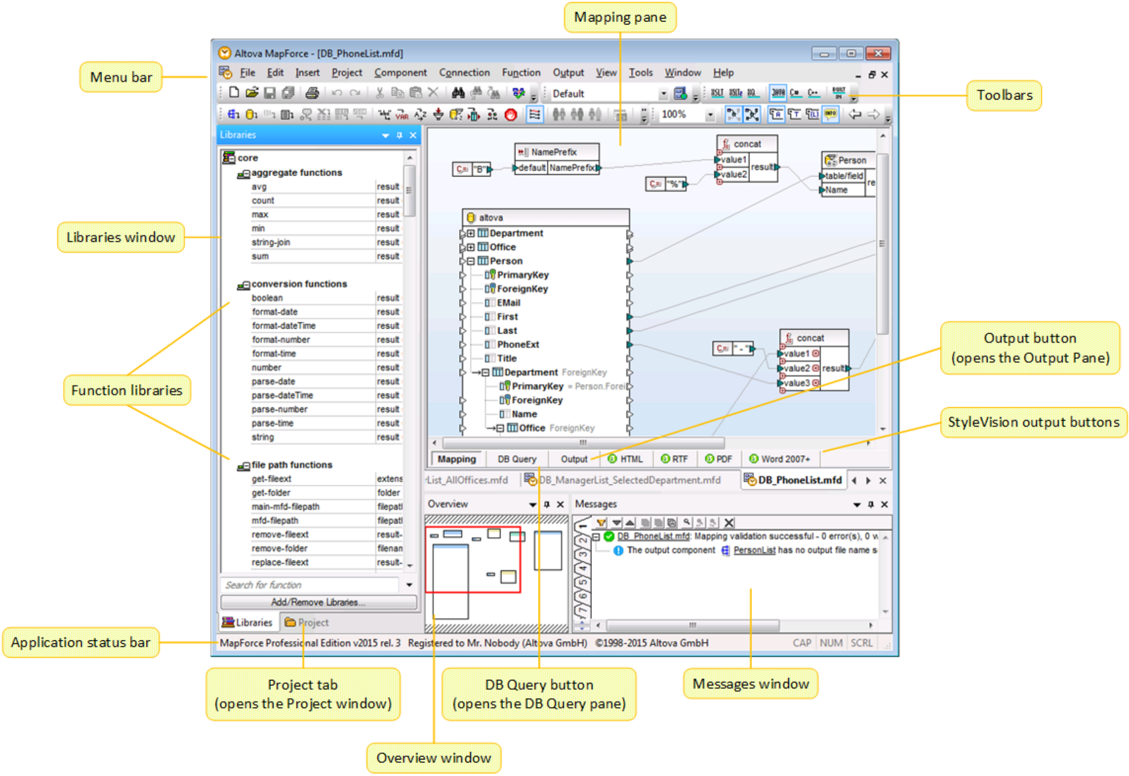
Target component

A target component is a component to which MapForce writes data. When you run the mapping, a target component instructs MapForce to either generate a file (or multiple files) or output the result as a string value for further processing in an external program. A target component is the opposite of a source component.

2.4 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**.



DB_PhoneList.mfd: Mapping validation successful - 0 error(s), 0 w

The output component PersonList has no output file name s

MapForce Professional Edition v2015 rel. 3 Registered to Mr. Nobody (Altova GmbH) ©1998-2015 Altova GmbH

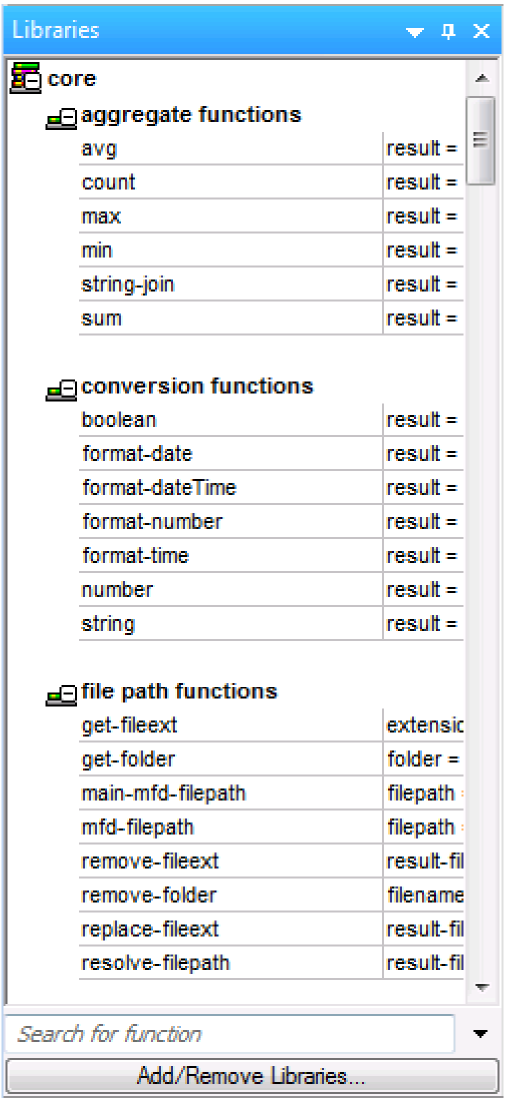
MapForce graphical user interface (MapForce Professional Edition)

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 the desired locations.

Libraries window

The Libraries window lists the MapForce built-in functions, organized by library. The list of available functions changes based on the transformation language you select. If you have created user-defined functions, or if you imported external libraries, they also appear in the Libraries window.



aggregate functions

avg	result =
count	result =
max	result =
min	result =
string-join	result =
sum	result =

conversion functions

boolean	result =
format-date	result =
format-dateTime	result =
format-number	result =
format-time	result =
number	result =
string	result =

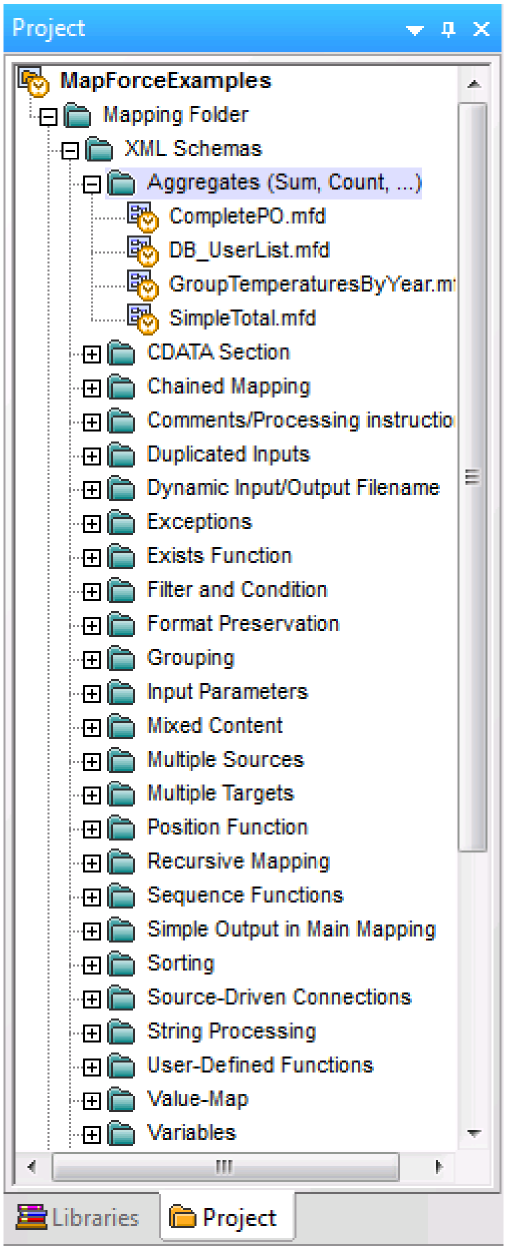
file path functions

get-fileext	extensic
get-folder	folder =
main-mfd-filepath	filepath
mfd-filepath	filepath
remove-fileext	result-fil
remove-folder	filename
replace-fileext	result-fil
resolve-filepath	result-fil

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 Working with Functions.

Project window

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



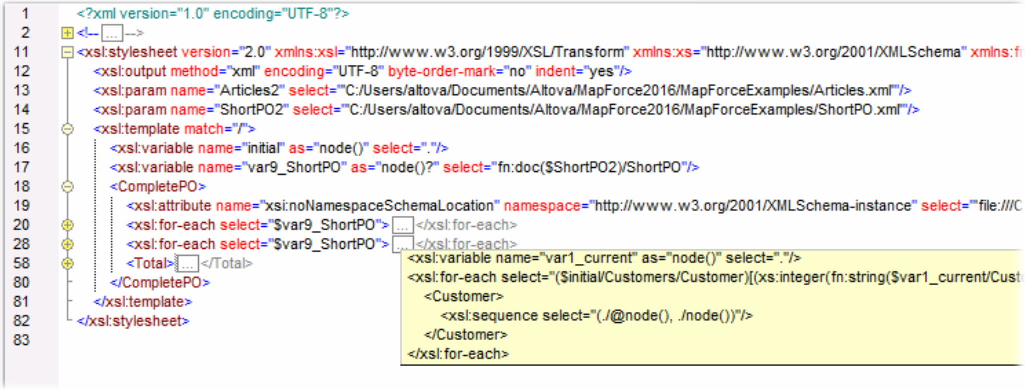
Mapping pane

The Mapping pane is the working area where you design mappings. You can add mapping components (such as files, schemas, constants, variables, and so on) to the mapping area from the **Insert** menu (see Adding Components to the Mapping). You can also drag into the Mapping pane functions displayed in the Libraries window (see Working with Functions).

XSLT (XSLT2) pane

The XSLT (or XSLT2) pane displays the XSLT 1.0 (or 2.0) transformation code generated from your mapping. To switch to this pane, select XSLT (or XSLT 2) as transformation language, and then click the **XSLT** tab (or **XSLT2** tab, respectively).

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 in the tooltip, an additional ellipsis appears at the end of the tooltip.



```
<?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:f
    <xsl:output method="xml" encoding="UTF-8" byte-order-mark="no" indent="yes"/>
    <xsl:param name="Articles2" select="'C:/Users/altova/Documents/Altova/MapForce2016/MapForceExamples/Articles.xml'"/>
    <xsl:param name="ShortPO2" select="'C:/Users/altova/Documents/Altova/MapForce2016/MapForceExamples/ShortPO.xml'"/>
    <xsl:template match="/">
        <xsl:variable name="initial" as="node()" select="."/>
        <xsl:variable name="var9_ShortPO" as="node()?" select="fn:doc($ShortPO2)/ShortPO"/>
        <CompletePO>
            <xsl:attribute name="xsi:noNamespaceSchemaLocation" namespace="http://www.w3.org/2001/XMLSchema-instance" select="'file:///C
            <xsl:for-each select="$var9_ShortPO">...</xsl:for-each>
            <xsl:for-each select="$var9_ShortPO">...</xsl:for-each>
            <Total>...</Total>
        </CompletePO>
    </xsl:template>
</xsl:stylesheet>
```

```
<xsl:variable name="var1_current" as="node()" select="."/>
<xsl:for-each select="($initial/Customers/Customer)[(xs:integer(fn:string($var1_current/Cust
    <Customer>
        <xsl:sequence select="(./@node(), ./node())"/>
    </Customer>
</xsl:for-each>
```

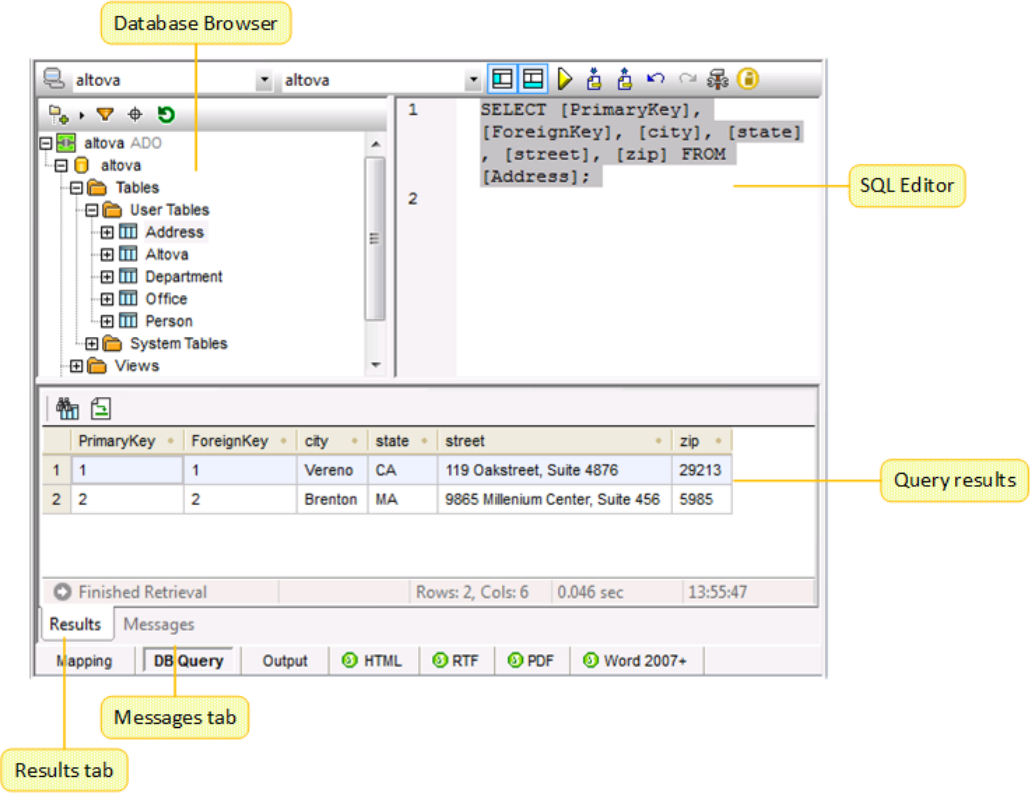
To configure the display settings (including indentation, end of line markers, and others), right-click the pane, and select **Text View Settings** from the context menu. Alternatively, click the **Text View Settings** ( ) toolbar button.

XQuery pane

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 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

The DB Query pane allows you to directly query any major database. You can work with multiple active connections to different databases.



```
SELECT [PrimaryKey],
[ForeignKey], [city], [state]
, [street], [zip] FROM
[Address];
```

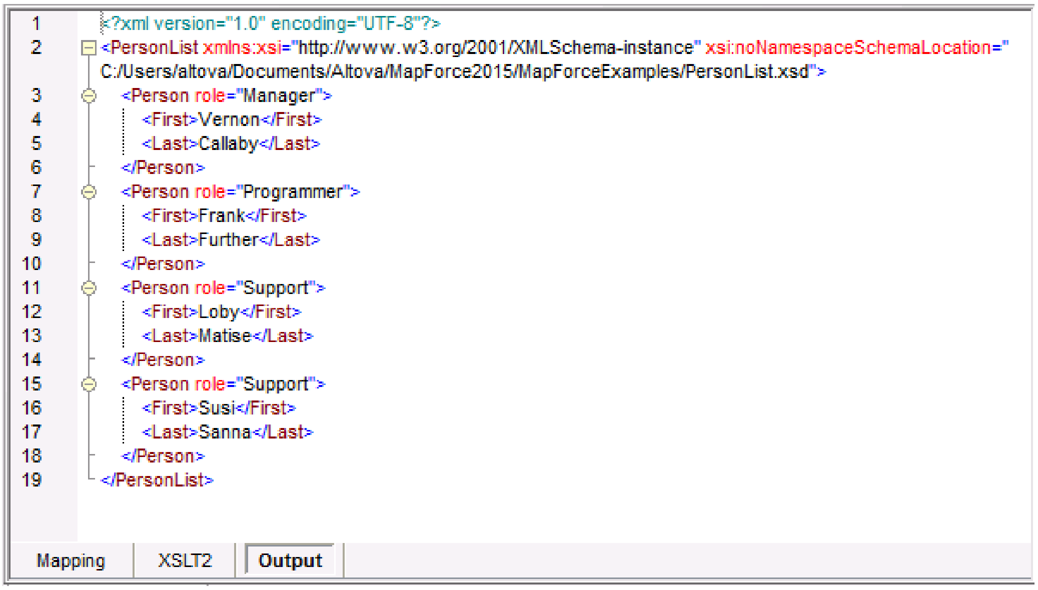
	PrimaryKey	ForeignKey	city	state	street	zip
1	1	1	Vereno	CA	119 Oakstreet, Suite 4876	29213
2	2	2	Brenton	MA	9865 Millenium Center, Suite 456	5985

```
Finished Retrieval    Rows: 2, Cols: 6    0.046 sec    13:55:47
```

For more information, see Browsing and Querying Databases.

Output pane

The Output pane displays the result of the mapping transformation (for example, an XML file), when you click the **Output** button. If the mapping generates multiple files, you can navigate sequentially through each generated file.



```
<?xml version="1.0" encoding="UTF-8"?>
<PersonList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:/Users/altova/Documents/Altova/MapForce2015/MapForceExamples/PersonList.xsd">
  <Person role="Manager">
    <First>Vernon</First>
    <Last>Callaby</Last>
  </Person>
  <Person role="Programmer">
    <First>Frank</First>
    <Last>Further</Last>
  </Person>
  <Person role="Support">
    <First>Loby</First>
    <Last>Matise</Last>
  </Person>
  <Person role="Support">
    <First>Susi</First>
    <Last>Sanna</Last>
  </Person>
</PersonList>
```

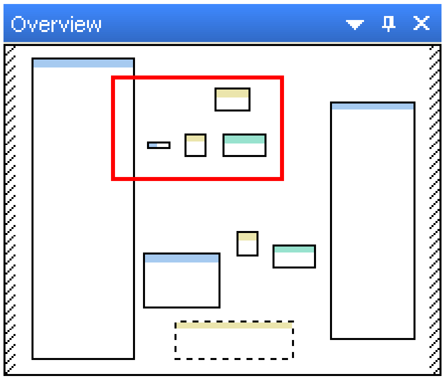
This pane also provides line numbering and code folding functionality, which works in a similar way as in the XSLT pane (see above).

StyleVision Output buttons

If you have installed Altova StyleVision (https://www.altova.com/stylevision.html), the StyleVision output buttons enable you to preview and save the mapping output in HTML, RTF, PDF, and Word 2007+ formats. This is possible by means of StyleVision Power Stylesheet (SPS) files designed in StyleVision and assigned to a mapping component in MapForce.

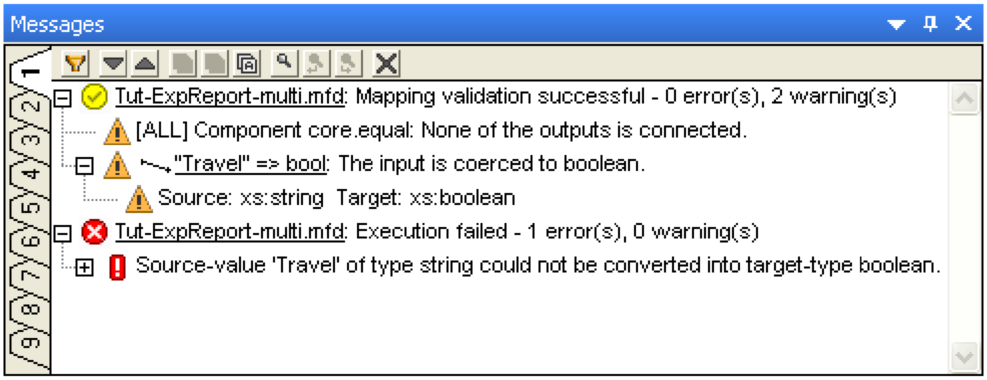
Overview window

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



Messages window

The Messages window shows messages, errors, and warnings when you execute a mapping (see Previewing the Output) or perform a mapping validation (see Validating Mappings).



Tut-ExpReport-multi.mfd: Mapping validation successful - 0 error(s), 2 warning(s)
[ALL] Component core.equal: None of the outputs is connected.
"Travel" => bool: The input is coerced to boolean.
Source: xs:string Target: xs:boolean
Tut-ExpReport-multi.mfd: Execution failed - 1 error(s), 0 warning(s)
Source-value 'Travel' of type string could not be converted into target-type boolean.

To highlight on the mapping area the component or structure which triggered the information, warning, or error message, click the underlined text in the Messages window.

The results of a mapping execution or validation operation is displayed in the Messages window with one of the following status icons:

Icon	Description
✅	Operation completed successfully.
✔	Operation completed with warnings.
❌	Operation has failed.

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

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

Other buttons in the Messages window enable you to take the following actions:

Icon	Description
⏷	Filter messages by severity (information messages, errors, warnings). Select **Check All** to include all severity levels (this is the default behaviour). 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 next line.
▲	Jump to previous line.
⧉	Copy the selected line to clipboard.
⧉	Copy the selected line to clipboard, including any lines nested under it.
⧉	Copy the full contents of the Messages window to 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 executing or validating the mapping.

Application status bar

The application status bar appears at the bottom of the application window, and shows application-level information. The most useful of this information are the tooltips that are displayed here 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 suffix (x64). There is no suffix for the 32-bit version.

2.5 Conventions

Example files

Most of the data mapping design files (files with .mfd extension, as well as other accompanying instance files) illustrated or referenced in this documentation are available in the following folders:

- `C:\Users\<username>\Documents\Altova\MapForce2018\MapForce Examples`
- `C:\Users\<username>\Documents\Altova\MapForce2018\MapForce Examples\Tutorials`

The example mappings and instance files accompanying MapForce illustrate most aspects of how it works, and you are highly encouraged to experiment with them as you learn about MapForce. When in doubt about the possible effects of making changes to the MapForce original examples, create back-ups before changing them.

Graphical user interface

Some of the images (screen shots) accompanying this documentation depict graphical user interface elements that may not be applicable to your MapForce edition. In relevant contexts, images typically include the name of the source mapping design (*.mfd) file, as well as the edition of MapForce in which the graphic was produced.

Chapter 3

Tutorials

3 Tutorials

The MapForce tutorials are intended to help you understand and use the basic data transformation capabilities of MapForce in a short amount of time. You can regard these tutorials as a "crash course" of MapForce. While the goal is not to illustrate completely all MapForce features, you will be guided through the MapForce basics step-by-step, so it is recommended that you follow the tutorials sequentially. It is important that you understand each concept before moving on to the next one, as the tutorials gradually grow in complexity. Basic knowledge of XML and XML schema will be advantageous.

Convert XML to New Schema

This tutorial shows you how to convert data from an XML structure to another using the XSLT 2.0 language, without writing any code. You will also learn about MapForce sequences and items, creating mapping connections, using a function, validating and previewing a mapping, as well as saving the resulting output to the disk.

Map Multiple Sources to One Target

This tutorial shows you how to read data from two XML files with different schema and merge it into a single target XML file. You will also learn how to change the name and instance files of each mapping component, and the concept of "duplicate inputs".

Work with Multiple Target Schemas

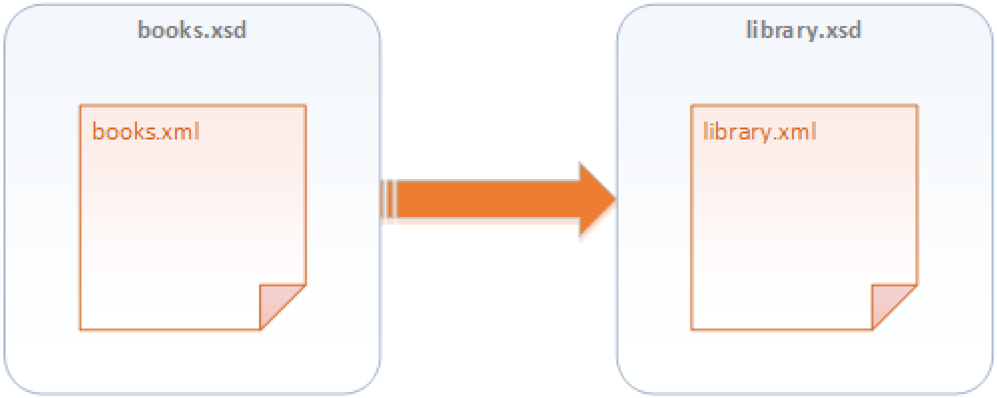
This tutorial shows you how to work with more complex mappings that produce two or more target outputs. More specifically, you will learn how to generate, in the same mapping, an XML file that stores a list of book records, and another XML file that contains only a subset of the books in the first file, filtered by a specific publication year. To support filtering data, you will use a **Filter** component, a function and a numeric constant.

Process and Generate Files Dynamically

This tutorial shows you how to read data from multiple XML instance files located in the same folder and write it to multiple XML files generated on the fly. You will also learn about stripping the XML and schema declarations and using functions to concatenate strings and extract file extensions.

3.1 Convert XML to New Schema

This tutorial shows you how to convert data between two XML files, while helping you learn the basics of the MapForce development environment. Both XML files store a list of books, but their elements are named and organized in a slightly different way (that is, the two files have different schemas).



Abstract model of the data transformation

The code listing below shows sample data from the file that will be used as data source (for the sake of simplicity, the XML and the namespace declarations are omitted).

```
<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>
```

books.xml

This is how data should look in the target (destination) file:

```
<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>
```

library.xml

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

To achieve the required data transformation, let's take the following steps.

Step 1: Select XSLT2 as transformation language

You can do this in one of the following ways:

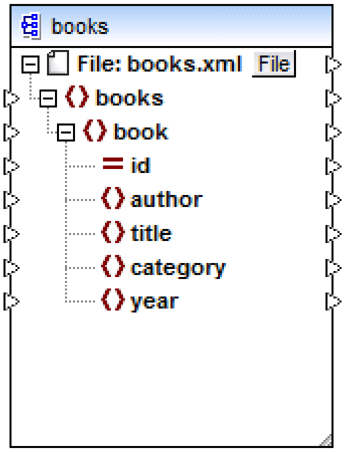
- Click the **XSLT2** ( ) toolbar button.
- On the **Output** menu, click **XSLT 2.0**.

Step 2: Add the source XML file to the mapping

The source XML file for this mapping is located at the following path: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\books.xml**. You can add it to the mapping in one of the following ways:

- Click the **Insert XML Schema/File** ( ) toolbar button.
- On the **Insert** menu, click **XML Schema/File**.
- Drag the XML file from Windows Explorer into the mapping area.

Now that the file has been added to the mapping area, you can see its structure at a glance. In MapForce, this structure is known as a mapping component, or simply component. You can expand elements in the component either by clicking the collapse (⊟) and expand icons (⊞), or by pressing the **+** and **-** keys on the numeric keypad.



Mapping component

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 corner of the component ◿ . You can also double-click the corner so that MapForce adjusts the size automatically.

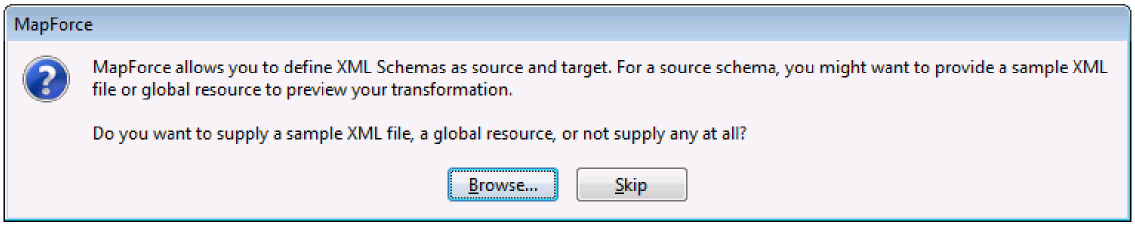
The top level node □ represents the file name; in this particular case, its title displays the name of the XML instance file. The XML elements in the structure are represented by the {} icon, while XML attributes are represented by the = icon.

The small triangles displayed on both sides of the component represent data inputs (if they are on the left side) or outputs (when they are on the right side). In MapForce, they are called input connectors and output connectors, respectively.

Step 3: Add the target XML schema to the mapping

To generate the target XML, we will use an existing XML schema file. In a real-life scenario, this file may have been provided to you by a third party, or you can create it yourself with a tool such as XMLSpy. If you don't have a schema file for your XML data, MapForce prompts you to generate it whenever you add to the mapping an XML file without an accompanying schema or schema reference.

For this particular example, we are using an existing schema file available at: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\library.xsd**. To add it to the mapping, follow the same steps as with the source XML file (that is, click the **Insert XML Schema/File** ( ) toolbar button). Click **Skip** when prompted by MapForce to supply an instance file.

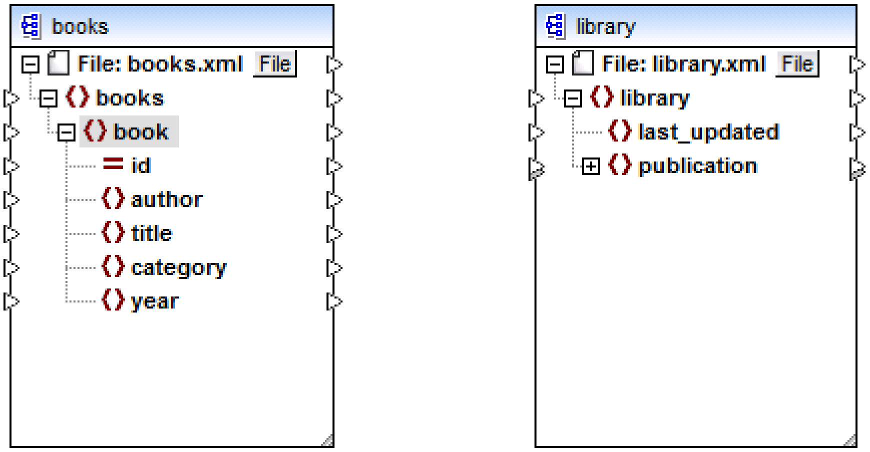


MapForce

MapForce allows you to define XML Schemas as source and target. For a source schema, you might want to provide a sample XML file or global resource to preview your transformation.

Do you want to supply a sample XML file, a global resource, or not supply any at all?

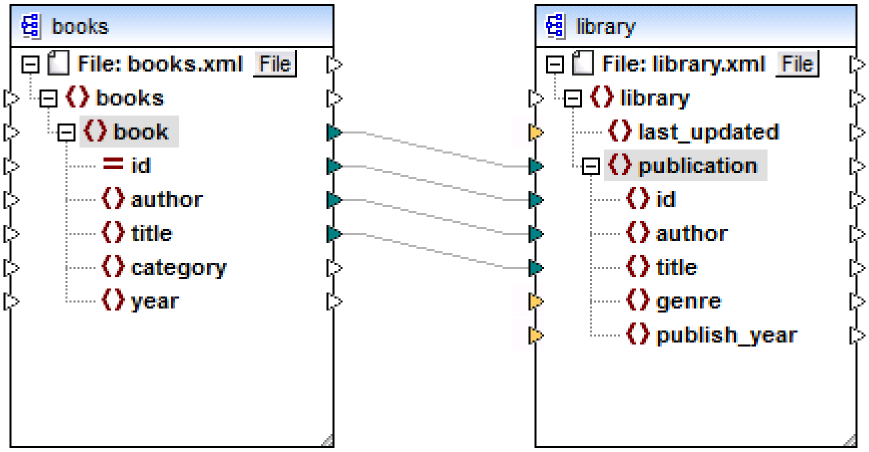
At this stage, the mapping design looks as follows:



Step 4: Make the connections

For each <book> in the source XML file, we want to create a new <publication> in the target XML file. We will therefore create a mapping connection between the <book> element in the source component and the <publication> element in the target component. To create the mapping connection, 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 elements which are children of <book> in the source file to elements having the same name in the target file; therefore, four connections are being created simultaneously. This behavior is called "Auto Connect Matching Children" and it can be disabled and customized if necessary.



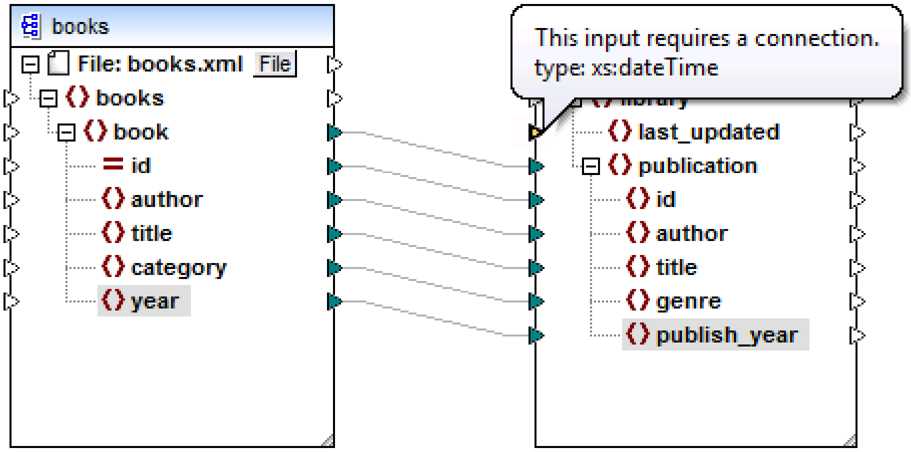
You can enable or disable the "Auto Connect Matching Children" behavior in one of the following ways:

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

Notice that some of the input connectors on the target component have been highlighted by MapForce in orange, which indicates that these items are mandatory. 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
- Connect the `<year>` element in the source with the `<publish_year>` element in the target

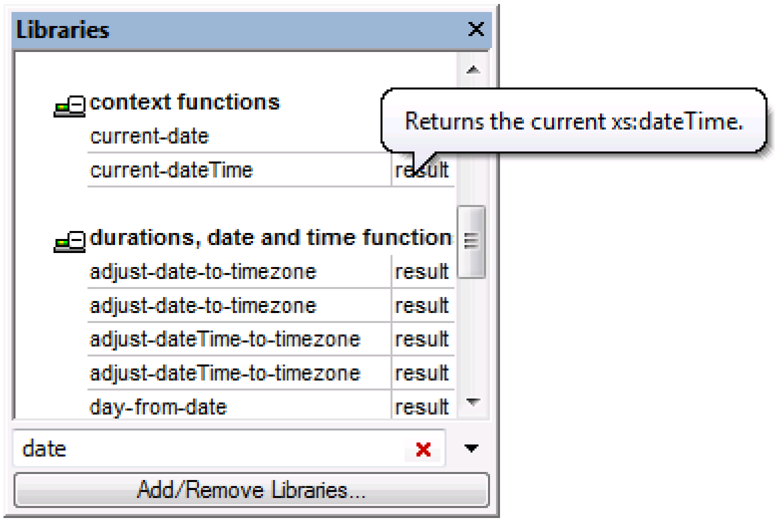
Finally, you need to supply a value to the `<last_updated>` element. If you move the mouse over its input connector, you can see that the element is of type `xs:dateTime`. Note that, for tips to be displayed, the **Show tips** ( ) toolbar button must be enabled.


This input requires a connection.
type: xs:dateTime

You can also make the data type of each item visible at all times, by clicking the **Show Data**

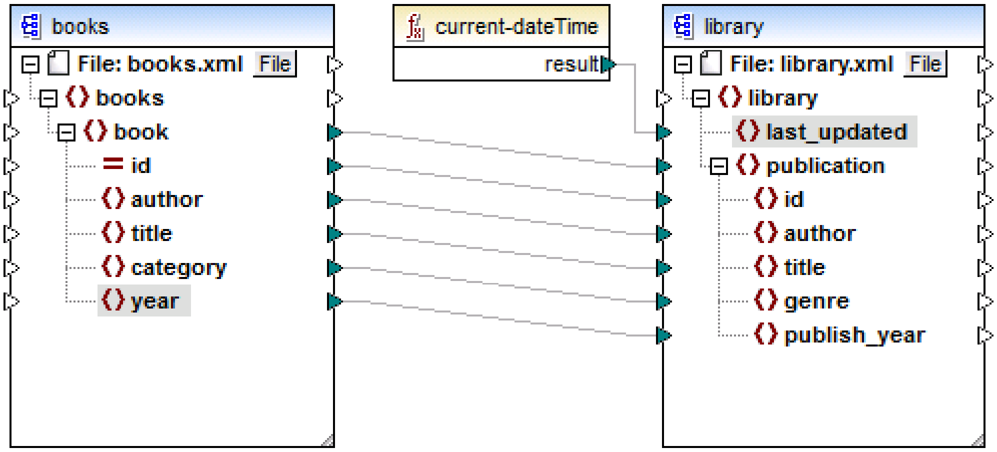
Types ( ) toolbar button.

You can get the current date and time (that is, the `xs:dateTime` value) by means of a date and time XSLT function. To find the XSLT function to the mapping, start typing "date" in the text box located in the lower part of the Libraries window. Alternatively, double-click an empty area on the mapping and start typing "current-date".

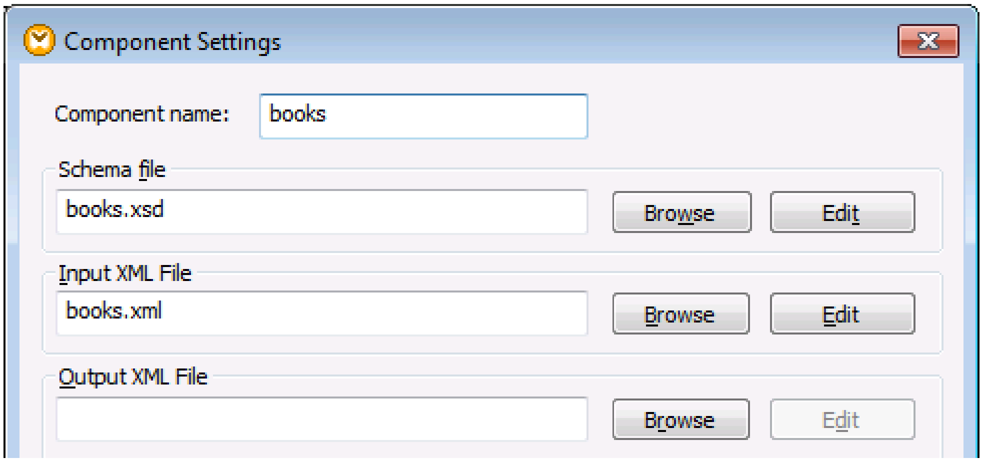

Returns the current xs:dateTime.

As shown above, if you move the mouse over the "result" part of the function, you can see its description. For tips to be displayed, make sure that the **Show tips** ( ) toolbar button is enabled.

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.



You have now created a MapForce mapping design (or simply a "mapping") which converts data from the **books.xml** instance file (having the **books.xsd** schema) to the new **library.xml** file (having the **library.xsd** schema). If you double-click the header of each component, you can view these and other settings in the Component Settings dialog box, as shown below.



Component Settings

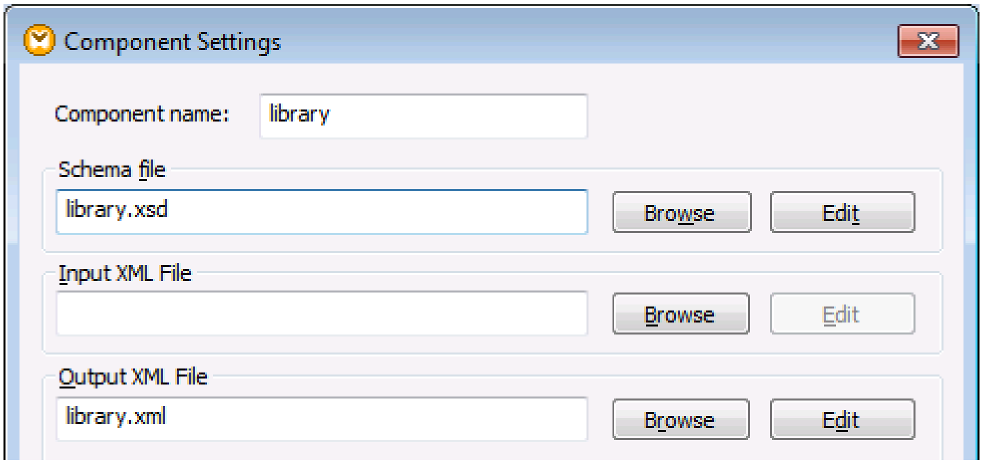
Component name: books

Schema file: books.xsd

Input XML File: books.xml

Output XML File:

Component settings for the source



Component Settings

Component name: library

Schema file: library.xsd

Input XML File:

Output XML File: library.xml

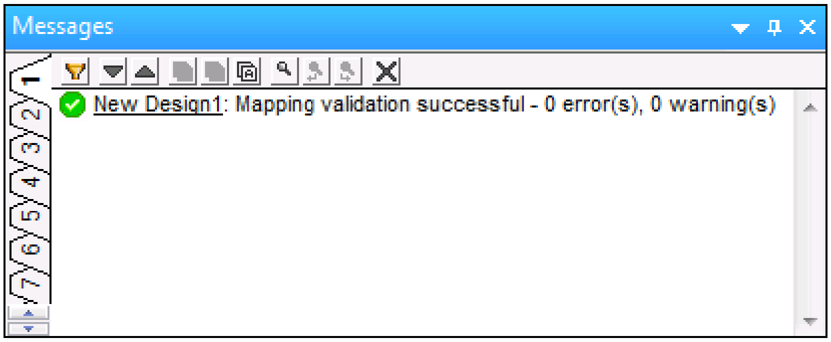
Component settings for the target

Step 5: Validate and save the mapping

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

- On the **File** menu, click **Validate Mapping**.
- Click the **Validate** ( ) toolbar button.

The Messages window displays the validation results:



New Design1: Mapping validation successful - 0 error(s), 0 warning(s)

Messages window

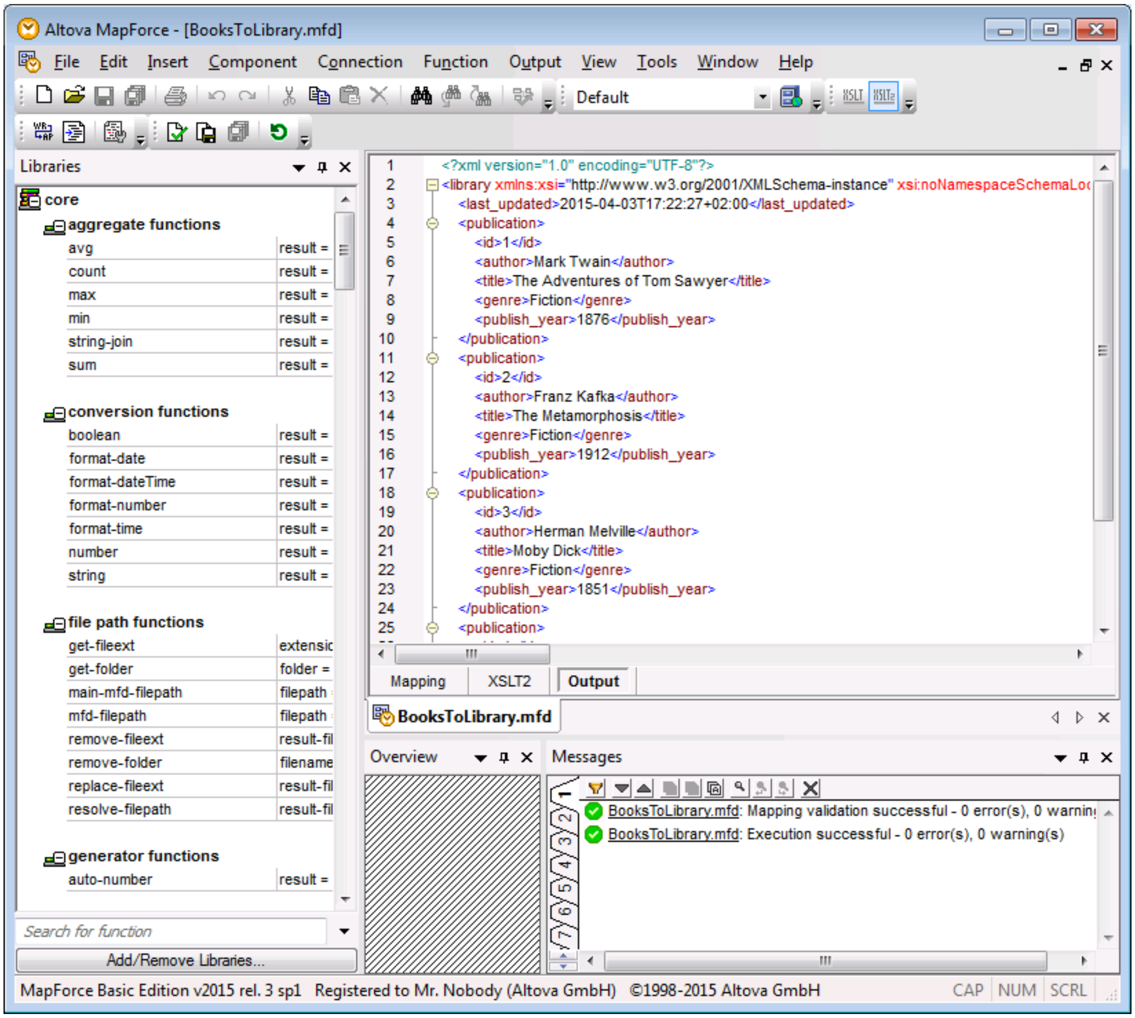
At this point, you might also want to save the mapping to a file. To save the mapping, do one of the following:

- On the **File** menu, click **Save**.
- Click the **Save** (💾) toolbar button.

For your convenience, the mapping created in this tutorial is available at the following path: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\\BooksToLibrary.mfd**. Therefore, from this point onwards, you can either continue with the mapping file you created, or with the **BooksToLibrary.mfd** file.

Step 6: Preview the mapping result

You can preview the result of the mapping directly in MapForce. To do this, click the **Output** button located in the lower part of the mapping pane. MapForce runs the transformation and displays the result of the mapping in the **Output** pane.



```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLoc
  <last_updated>2015-04-03T17:22:27+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>
  <publication>
    <id>3</id>
    <author>Herman Melville</author>
    <title>Moby Dick</title>
    <genre>Fiction</genre>
    <publish_year>1851</publish_year>
  </publication>
  <publication>
```

```
BooksToLibrary.mfd: Mapping validation successful - 0 error(s), 0 warnin
BooksToLibrary.mfd: Execution successful - 0 error(s), 0 warning(s)
```

MapForce Basic Edition v2015 rel. 3 sp1 Registered to Mr. Nobody (Altova GmbH) ©1998-2015 Altova GmbH CAP NUM SCRL

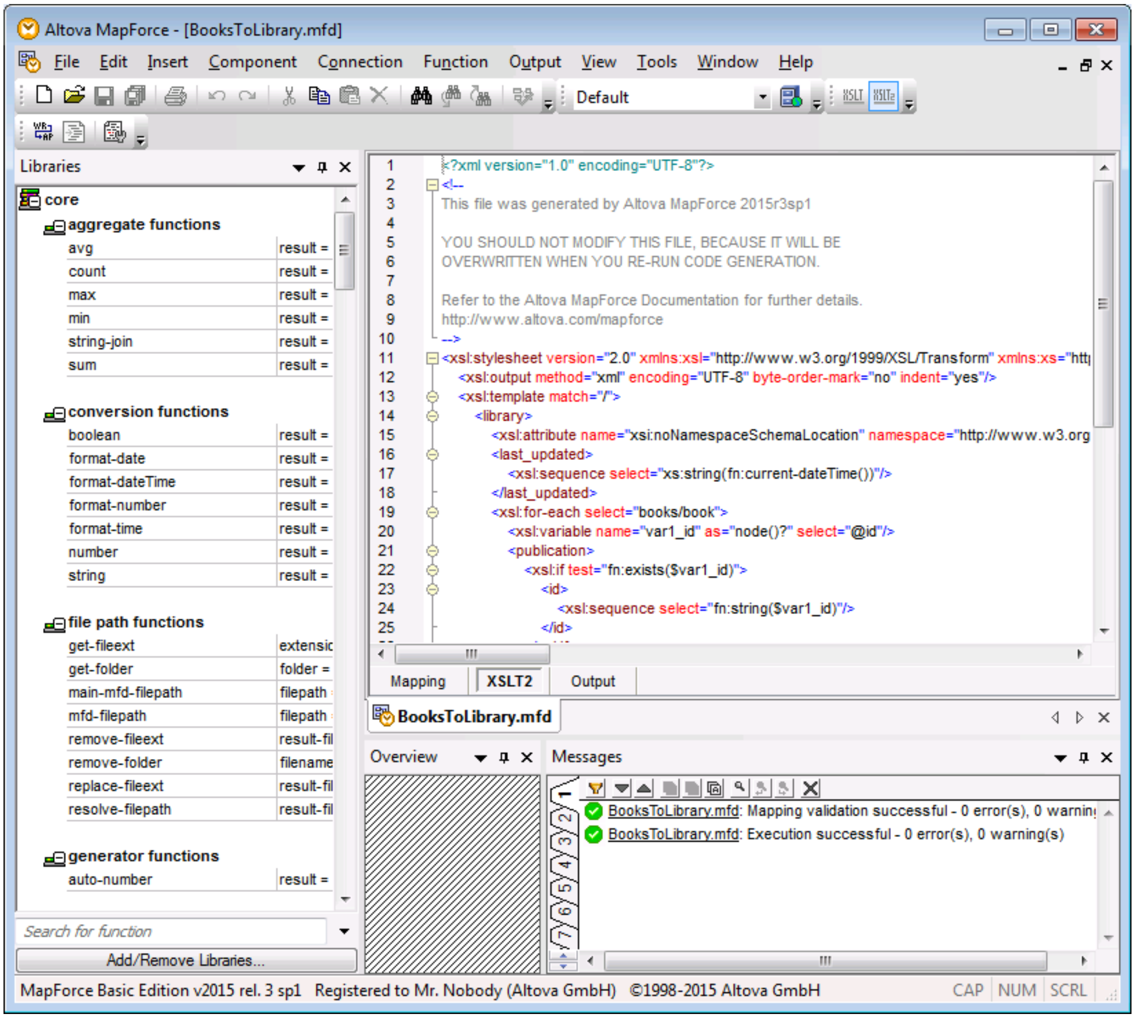
Output pane

You can now see the result of the transformation in MapForce.

By default, the files displayed for preview in the **Output** pane are not written to the disk. Instead, MapForce creates temporary files. 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.

To configure MapForce to write the output directly to final files instead of temporary, go to **Tools | Options | General**, and then select the **Write directly to final output files** check box. 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 in the lower area of the mapping pane.



```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This file was generated by Altova MapForce 2015r3sp1

YOU SHOULD NOT MODIFY THIS FILE, BECAUSE IT WILL BE
OVERWRITTEN WHEN YOU RE-RUN CODE GENERATION.

Refer to the Altova MapForce Documentation for further details.
http://www.altova.com/mapforce
-->
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xs="htt
  <xsl:output method="xml" encoding="UTF-8" byte-order-mark="no" indent="yes"/>
  <xsl:template match="/">
    <library>
      <xsl:attribute name="xsi:noNamespaceSchemaLocation" namespace="http://www.w3.org
      <last_updated>
        <xsl:sequence select="xs:string(fn:current-dateTime())"/>
      </last_updated>
      <xsl:for-each select="books/book">
        <xsl:variable name="var1_id" as="node()?" select="@id"/>
        <publication>
          <xsl:if test="fn:exists($var1_id)">
            <id>
              <xsl:sequence select="fn:string($var1_id)"/>
            </id>
```

```
BooksToLibrary.mfd: Mapping validation successful - 0 error(s), 0 warnin
BooksToLibrary.mfd: Execution successful - 0 error(s), 0 warning(s)
```

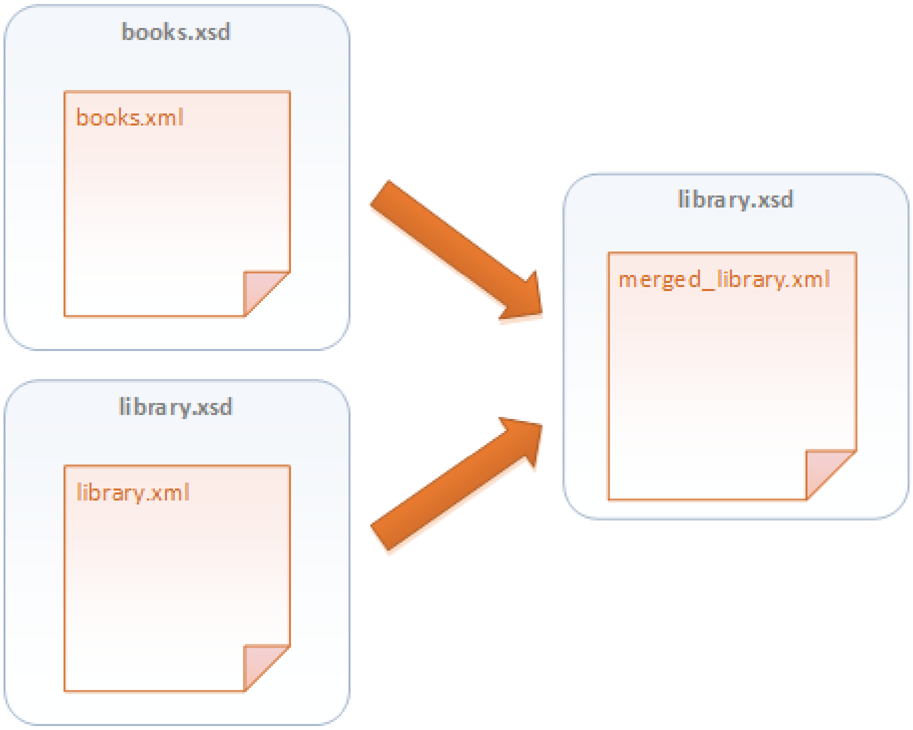
XSLT2 pane

To generate and save the XSLT2 code 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 code generation completes, the destination folder includes the following two files:

1. An XSLT transformation file, named after the target schema (in this example, **MappingMaptolibrary.xslt**).
2. A **DoTransform.bat** file. The **DoTransform.bat** file enables you to run the XSLT transformation in RaptorXML Server (for more information, see https://www.altova.com/raptorxml.html).

3.2 Map Multiple Sources to One Target

In the previous tutorial, you have converted data from a source file (**books.xml**) to a target file (**library.xml**). The target file (**library.xml**) did not exist before running the mapping; it was generated by the mapping transformation. Let's now imagine a scenario where you already have some data in the **library.xml** file, and you want to merge this data with data converted from the **books.xml**. The goal in this tutorial is to design a mapping that generates a file called **merged_library.xml**. The generated file will include data from two sources: the **books.xml** file and the **library.xml** file. Note that the files used as source (**books.xml** and **library.xml**) have different schemas. If the source files had the same schema, you could also merge their data using a different approach (see Process and Generate Files Dynamically).



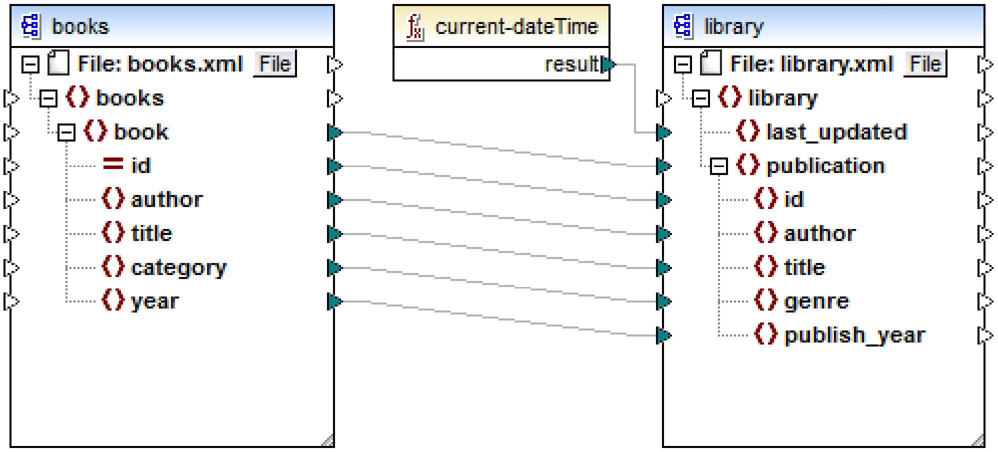
Abstract model of the data transformation

To achieve the required goal, let's take the following steps.

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **BooksToLibrary.mfd** mapping from the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder. You have already designed this mapping in the Convert XML to New Schema tutorial. To begin, open the **BooksToLibrary.mfd** file in MapForce, and save it with a new name.

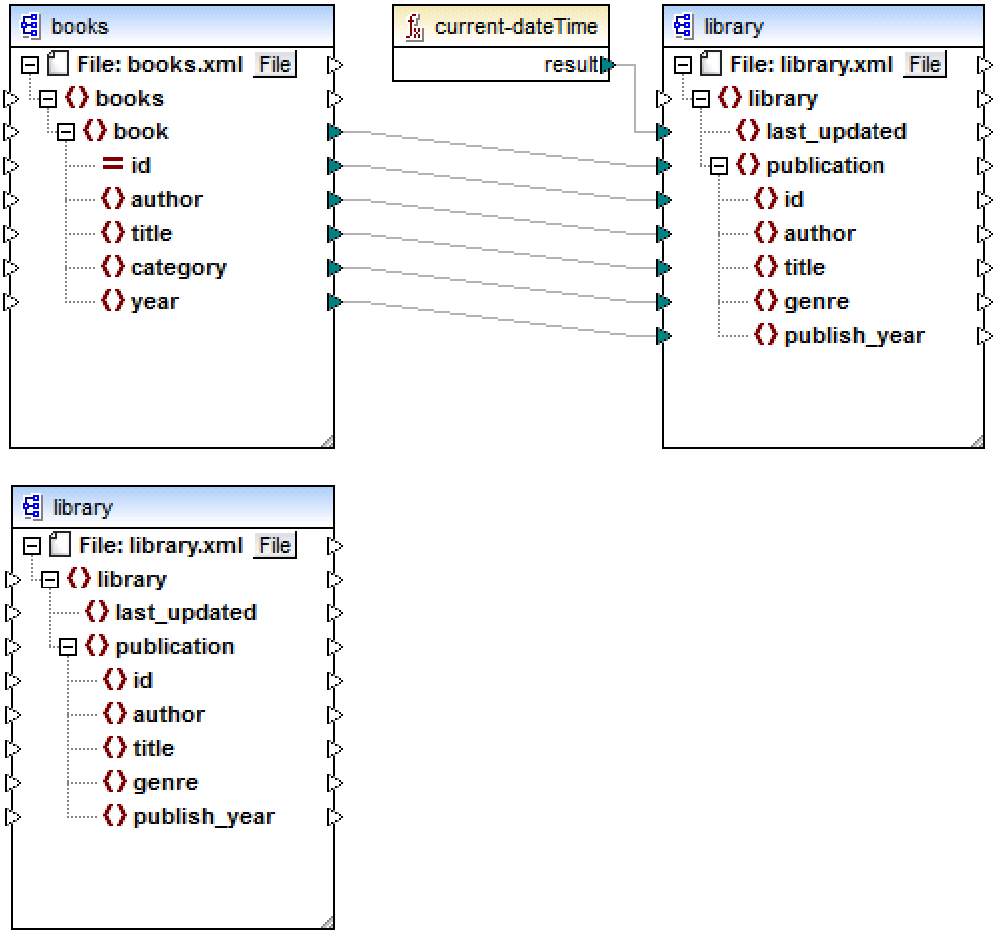
> Make sure to save the new mapping in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder, because it references several files from it.



BooksToLibrary.mfd (MapForce Basic Edition)

Step 2: Create a second source component

First, select the target component and copy it (press **Ctrl + C**), and then paste it (press **Ctrl + V**) into the same mapping. Click the header of the new component and drag it under the **books** component.

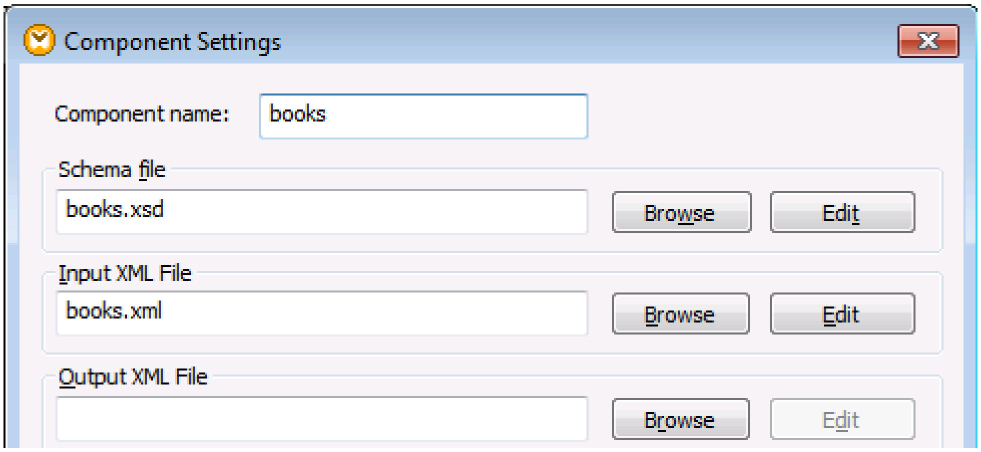


The mapping now has two source components: **books** and **library**, and one target component: **library**.

> You can always move the mapping components in any direction (left, right, top, bottom). Nevertheless, placing a source component to the left of a target component will make your mapping easier to read and understand by others. This is also the convention for all mappings illustrated in this documentation, as well as in the sample mapping files accompanying your MapForce installation.

Step 3: Verify and set the input/output files

In the previous step, the new source component was copy-pasted from the target component, so it inherits the same settings. To ensure that the name input/output instance files are correctly set, double-click the header of each component, and, in the Component Settings dialog box, verify and change the name and the input/output files of each component as shown below.



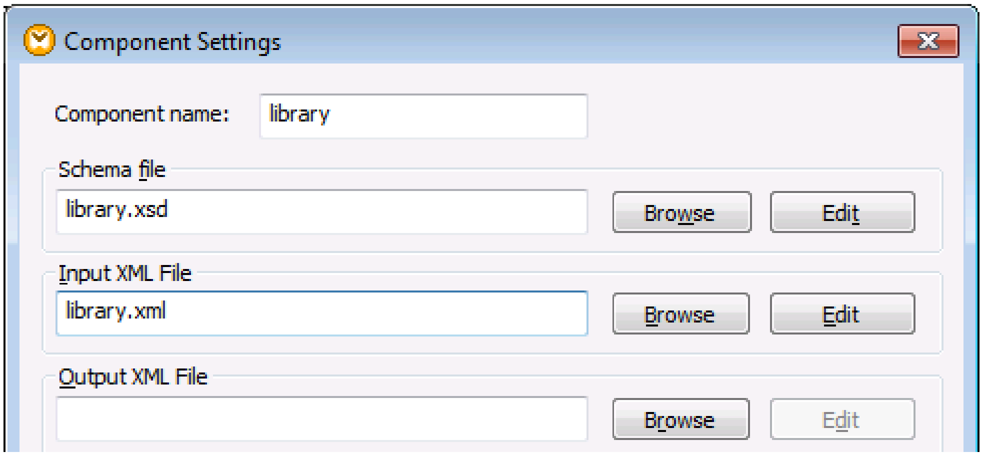
Component name: books

Schema file: books.xsd

Input XML File: books.xml

Output XML File:

*Components settings for the first source (**books**)*



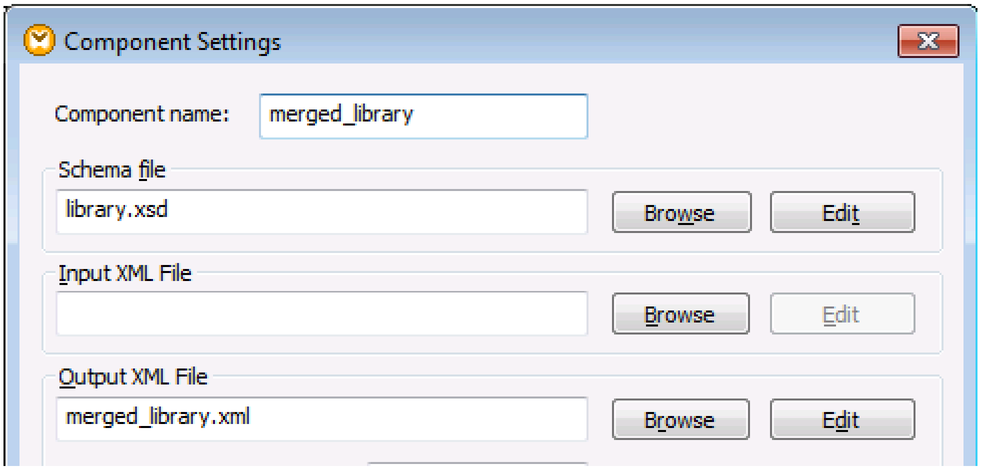
Component name: library

Schema file: library.xsd

Input XML File: library.xml

Output XML File:

*Component settings for the second source (**library**)*


Component Settings

Component name: merged_library

Schema file: library.xsd

Input XML File:

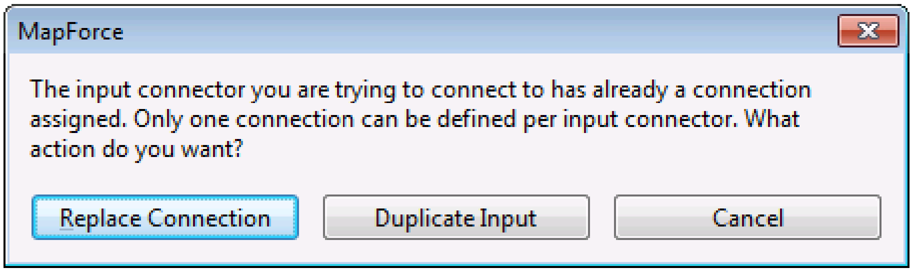
Output XML File: merged_library.xml

*Component settings for the target (**merged_library**)*

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 **merged_library.xml**.

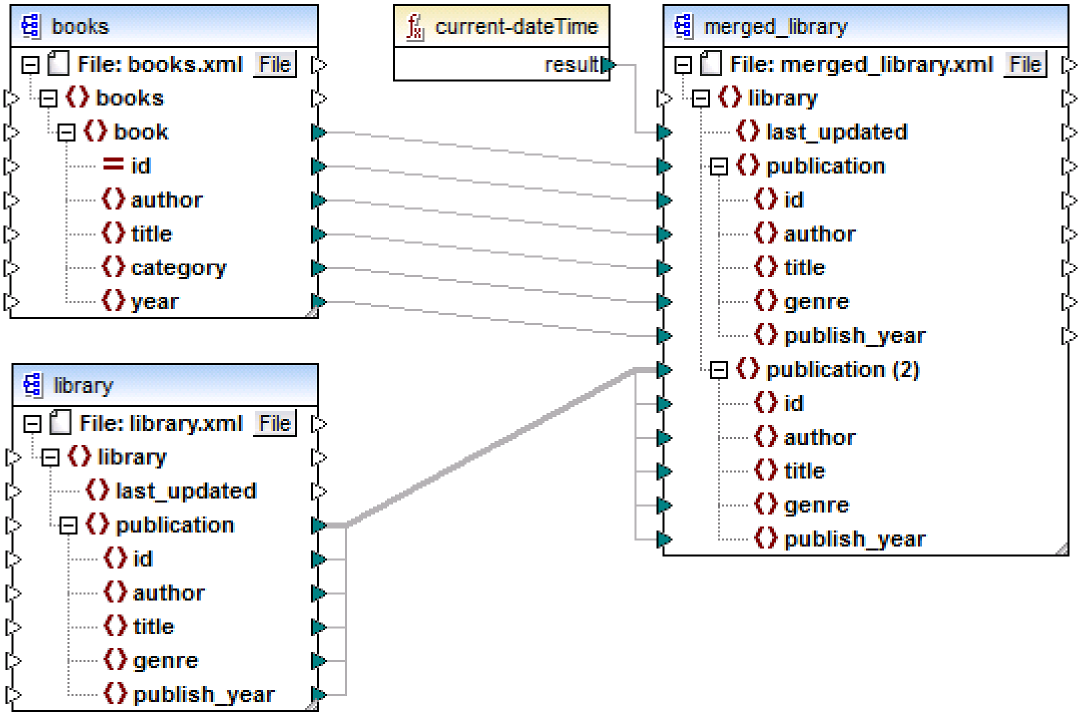
Step 4: Make the connections

To instruct MapForce to write data from the second source to the target, click the output connector (small triangle) of the `publications` item in the source **library** component and drag it to the input connector of the publications item in the target **library** component. Because the target input connector already has a connection to it, the following notification message appears.


MapForce

The input connector you are trying to connect to has already a connection assigned. Only one connection can be defined per input connector. What action do you want?

In this particular tutorial, replacing the connection is not what we want to achieve; our goal is to map data from two sources. Therefore, click **Duplicate Input**. By doing so, you configure the target component to accept data from the new source as well. The mapping now looks as follows:



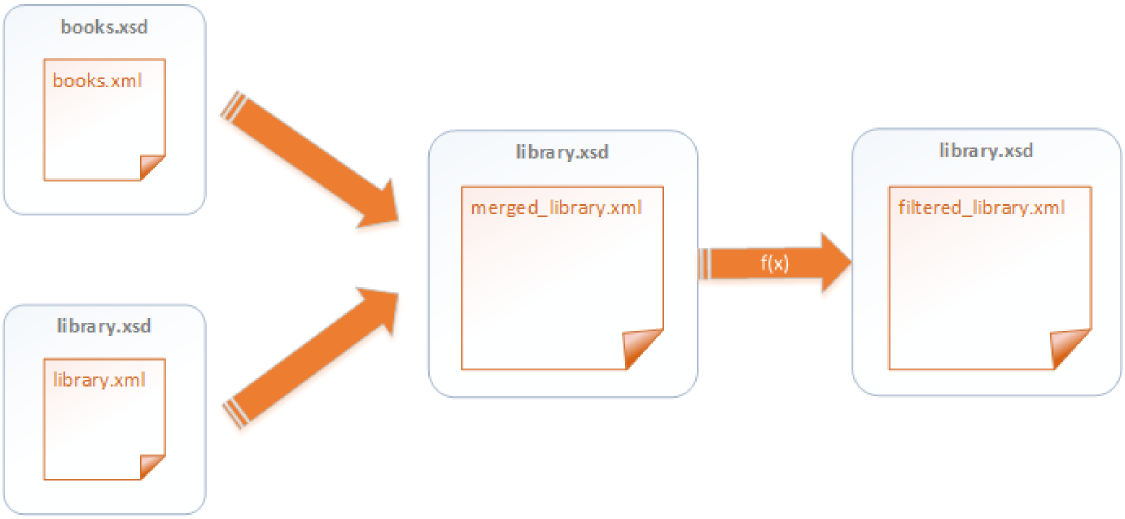
Notice that the `publication` item in the target component has now been duplicated. The new `publication(2)` node will accept data from the source **library** component. Importantly, even though the name of this node appears as `publication(2)` in the mapping, its name in the resulting XML file will be `publication`, which is the intended goal.

You can now click the **Output** button at the bottom of the mapping pane, and view the mapping result. You will notice that data from both **library.xml** and **books.xml** files has now been merged into the new **merged_library.xml** file.

3.3 Work with Multiple Target Schemas

In the previous tutorial, Map Multiple Sources to One Target, you have seen how to map data from multiple source schemas to a single target schema. You have also created a file called **merged_library.xml**, which stores book records from two sources. Now let's assume that someone from another department has asked you to provide a subset of this XML file. Specifically, you must deliver an XML file that includes only the books published after 1900.

For convenience, you can modify the existing **MultipleSourcesToOneTarget.mfd** mapping so that, whenever required, you can generate both the complete XML library, and the filtered library.



Abstract model of the data transformation

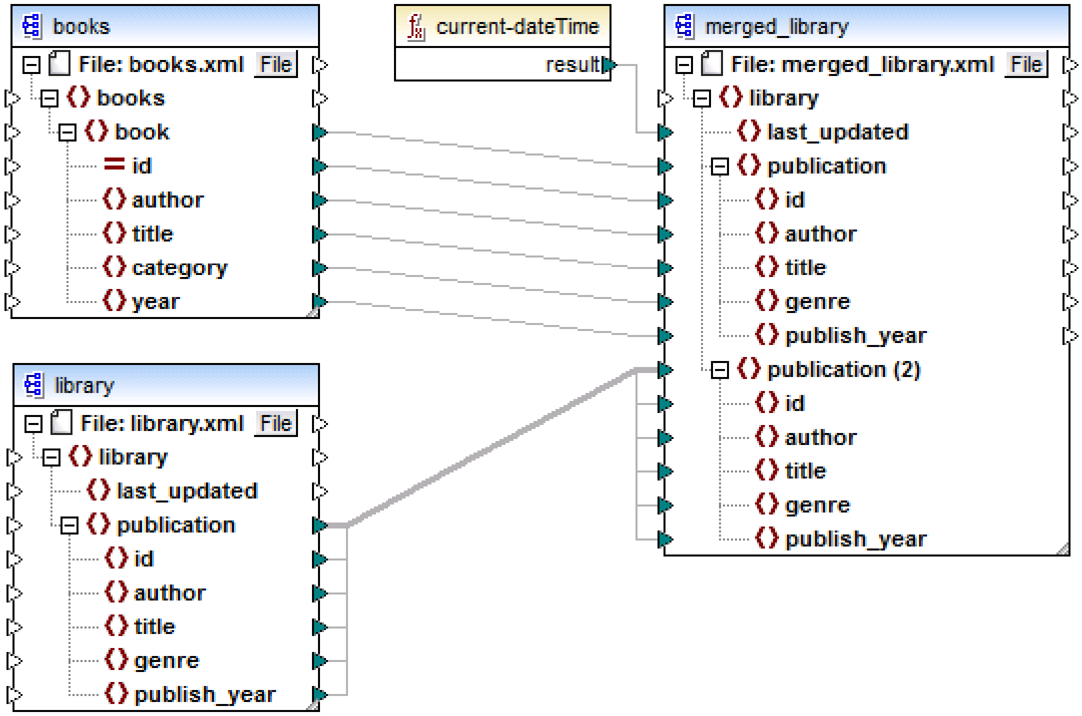
In the diagram above, the data is first merged from two different schemas (**books.xsd** and **library.xsd**) into a single XML file called **merged_library.xml**. Secondly, the data is transformed using a filtering function and passed further to the next component, which creates an XML file called **filtered_library.xml**. The "intermediate" component acts both as data target and source. In MapForce, this technique is known as "chaining mappings", which is also the subject of this tutorial.

Our goal is to make it possible to generate at any time both the **merged_library.xml** and the **filtered_library.xml**. To achieve the goal, let's take the following steps.

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **MultipleSourcesToOneTarget.mfd** mapping from the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder. You have already designed this mapping in the Map Multiple Sources to One Target tutorial. To begin, open the **MultipleSourcesToOneTarget.mfd** file in MapForce, and save it with a new name.

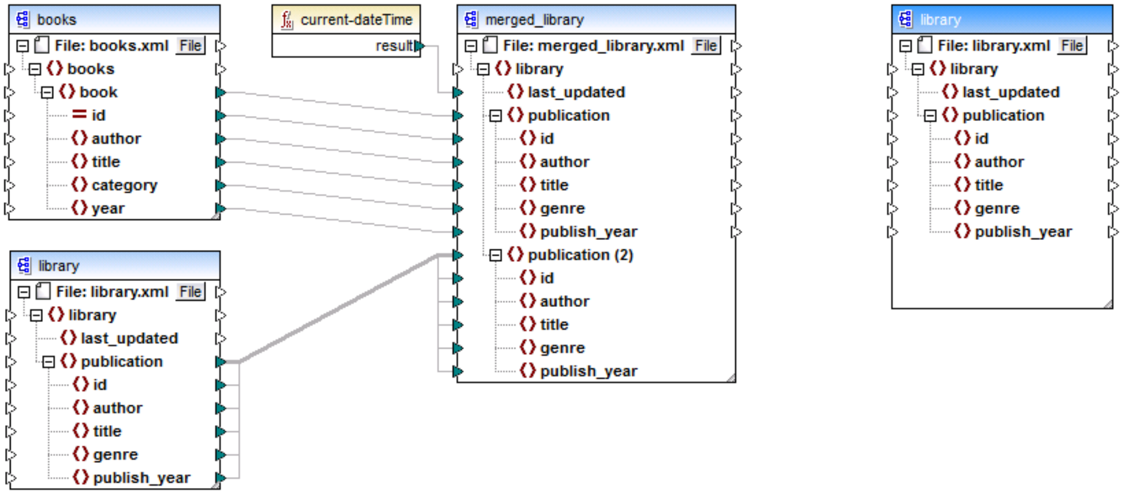
Make sure to save the new mapping in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder, because it references several files from it.



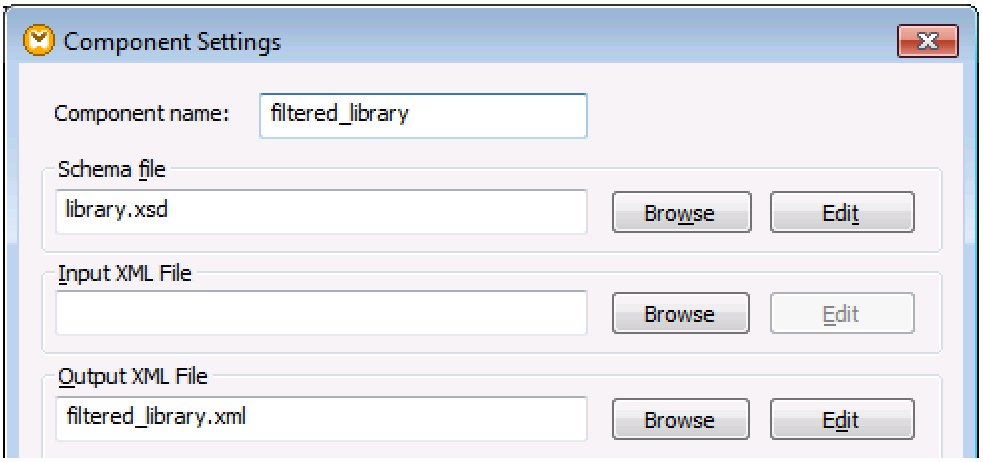
MultipleSourcesToOneTarget.mfd (MapForce Basic Edition)

Step 2: Add and configure the second target component

To add the second target component, click the **Insert XML Schema/File** ( ) toolbar button, and open the **library.xsd** file located in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder. Click **Skip** when prompted to supply a sample instance file. The mapping now looks as follows:



As shown above, the mapping now has two source components: **books** and **library**, and two target components. To distinguish between the target components, we will rename the second one to **filtered_library**, and also set the name of the XML file that should be generated by it. To do this, double-click the header of the right-most component and edit the component settings as follows:



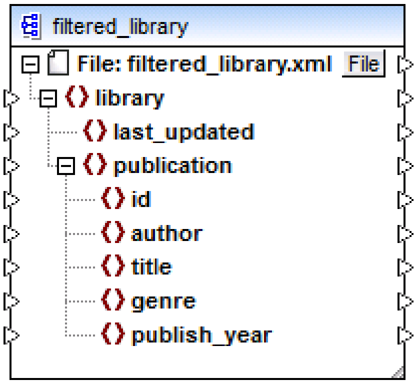
Component name: filtered_library

Schema file: library.xsd

Input XML File:

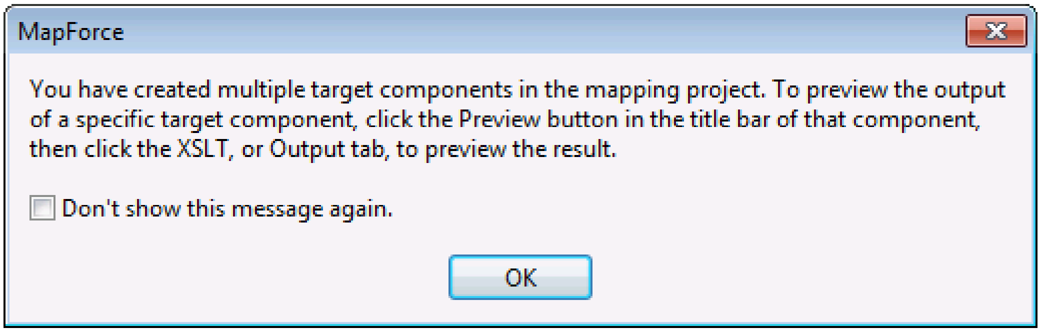
Output XML File: filtered_library.xml

Notice that the new name of the component is **filtered_library**, and the output XML file is named **filtered_library.xml**.



Step 3: Make the connections

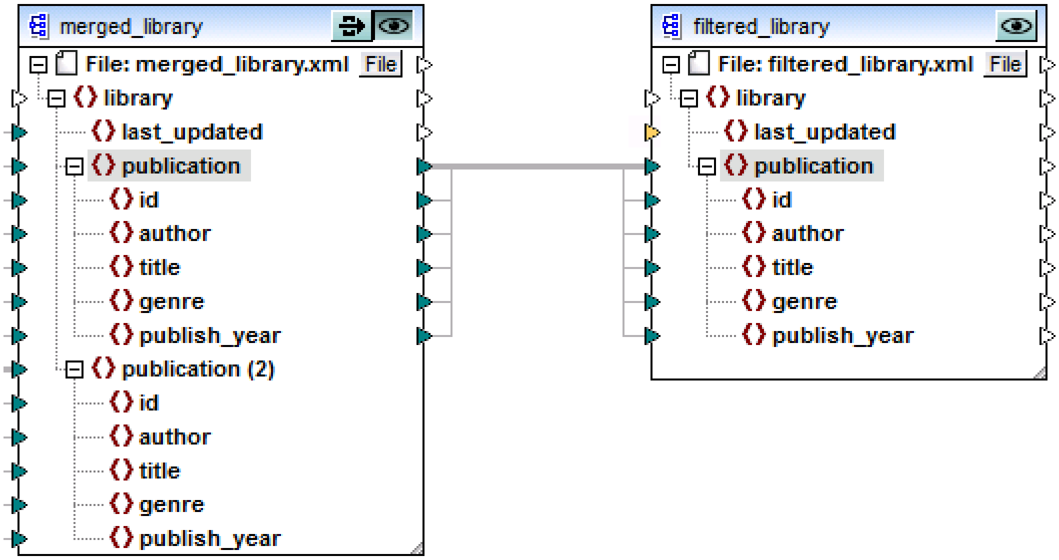
Create a connection from the item **publication** in the **merged_library** to the item **publication** in the **filtered_library**. When you do this, a notification message is displayed.



You have created multiple target components in the mapping project. To preview the output of a specific target component, click the Preview button in the title bar of that component, then click the XSLT, or Output tab, to preview the result.

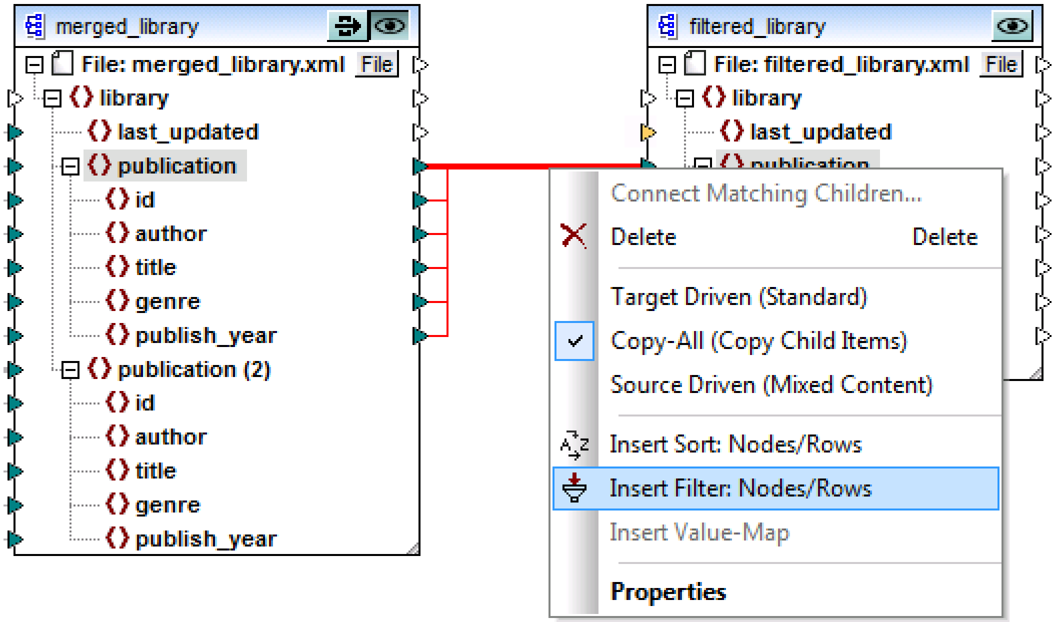
□ Don't show this message again.

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 following steps.

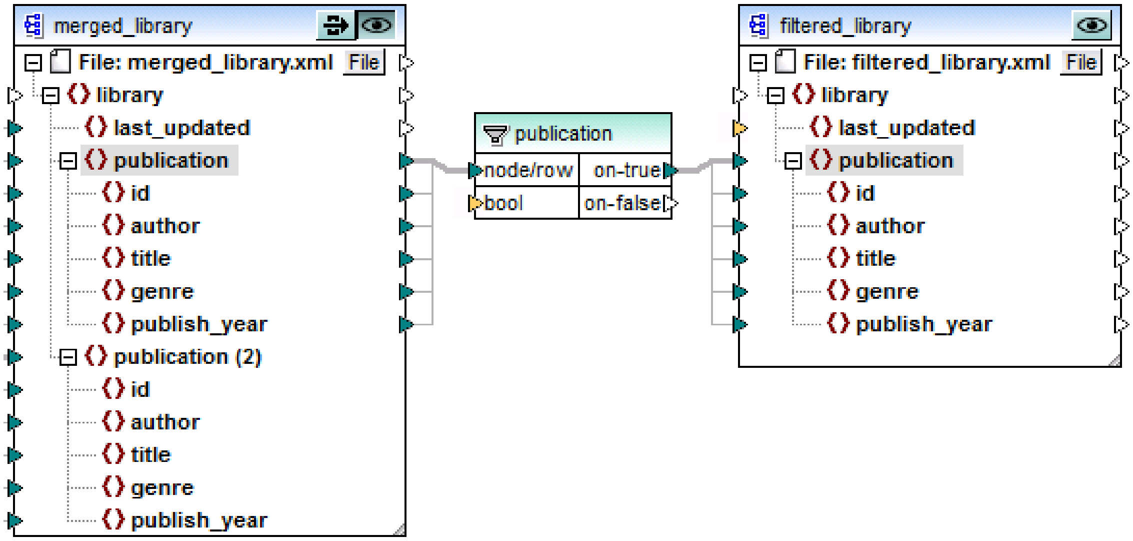


Step 4: Filter data

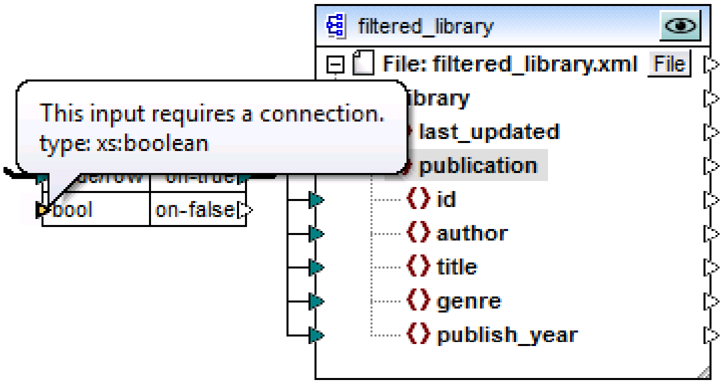
To filter data before supplying it to the **filtered_library**, we will use a **Filter** component. To add a filter component, right-click the connection between **merged_library** and **filtered_library**, and select **Insert Filter: Nodes/Rows** from the context menu.



The filter component has now been added to the mapping.



As shown above, the bool input connector is highlighted in orange, which suggests that an input is required. If you move the mouse over the connector, you can see that an input of type `xs:boolean` is required. Note that, for tips to be displayed, the **Show tips** (INFO) toolbar button must be enabled.

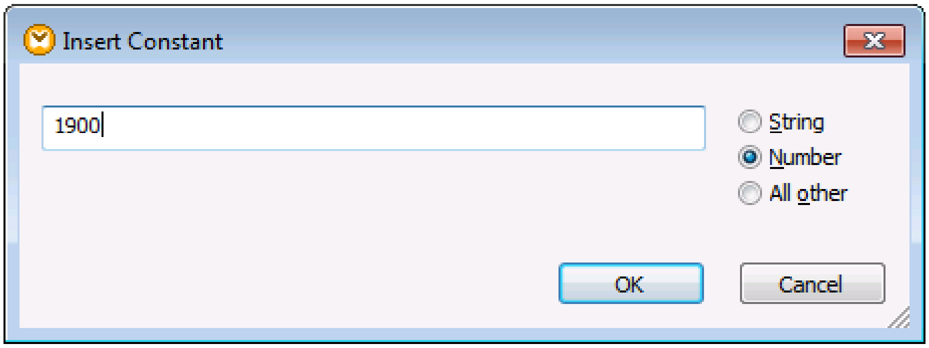


This input requires a connection.
type: xs:boolean

The filter component requires a condition that returns either `true` or `false`. When the Boolean condition returns `true`, data of the current **publication** sequence will be copied over to the target. When the condition returns `false`, data will not be copied.

In this tutorial, the required condition is to filter all books which were published after 1900. To create the condition, do the following:

1. Add a constant of numeric type having the value "1900" (On the **Insert** menu, click **Constant**). Choose **Number** as type.

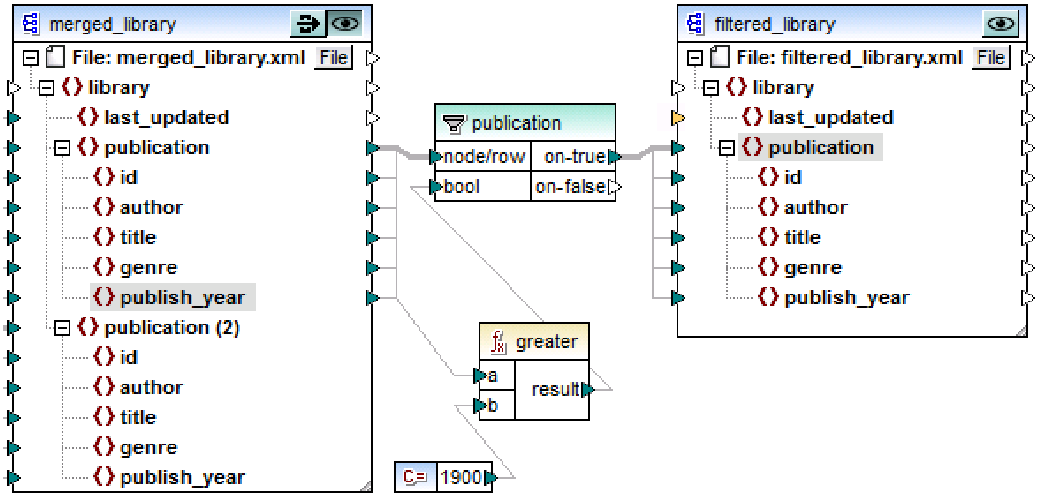


Insert Constant

1900

○ String
● Number
○ All other

2. In the Libraries window, locate the function **`greater`** and drag it to the mapping pane.
3. Make the mapping connections to and from the function **`greater`** as shown below. By doing this, you are instructing MapForce: "When `publish_year` is greater than 1900, copy the current `publication` source item to the `publication` target item".



Step 5: Preview and save the output of each target component

You are now ready to preview and save the output of both target components. When multiple target components exist in the same mapping, you can choose which one to preview by clicking the **Preview** (👁) button. When the **Preview** button is in a pressed state (👁), it indicates that that specific component is currently enabled for preview (and this particular component will generate the output in the Preview pane). Only one component at a time can have the preview enabled.

Therefore, when you want to view and save the output of the **merged_library** (that is, the "intermediate") component, do the following:

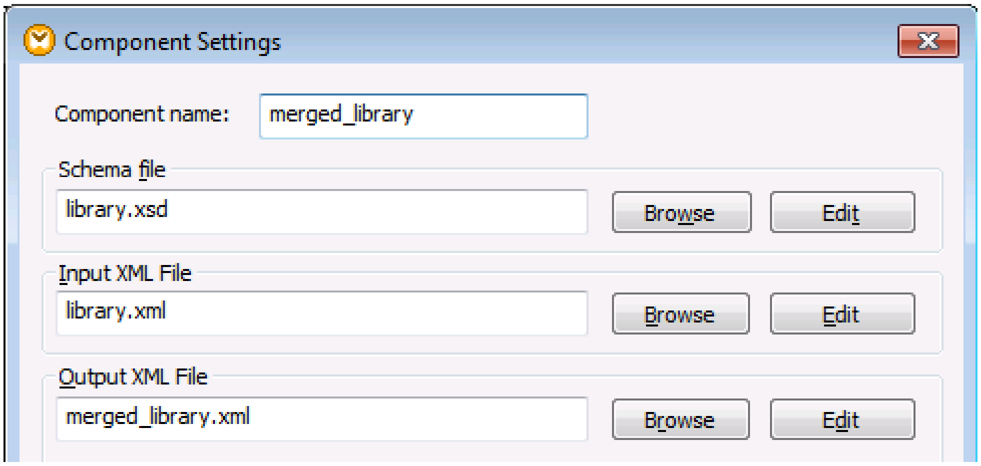
1. Click the **Preview** button (👁) on the **merged_library** component.
2. Click the **Output** button at the bottom of the mapping pane.
3. On the **Output** menu, click **Save Output File** if you want to save the output to a file.

When you want to view and save the output of the **filtered_library** component :

1. Click the **Pass-through** button (⇛) on the **merged_library** component.
2. Click the **Preview** button (👁) on the **filtered_library** component.
3. Click the **Output** button at the bottom of the mapping pane.
4. On the **Output** menu, click **Save Output File** if you want to save the output to a file.

Notice the **Pass-through** (⇛) button—clicking or not clicking it makes a big difference in any mapping which has multiple target components, including this one. When this button is in a pressed state (⇛), MapForce lets data pass through the intermediate component, so that you can preview the result of the entire mapping.

Release the button (⇛) if you want to preview only the portion of the mapping between the **merged_library** and the **filtered_library**. In the latter case, an error will be generated. This behavior is expected, because the intermediate component does not have a valid input XML file from which it should read data. To solve the problem, double-click the header of the component and edit so as to supply a valid input XML file, as shown below:



Component name: merged_library

Schema file: library.xsd

Input XML File: library.xml

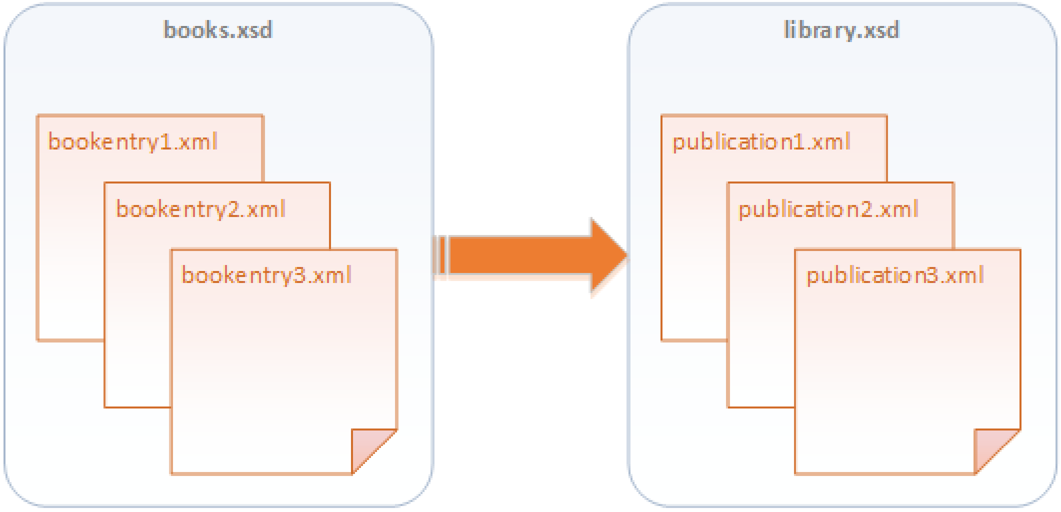
Output XML File: merged_library.xml

You have now finished designing a mapping which has multiple target components, and you can view and save the output of each target, which was the intended goal of this tutorial. For further information about working with pass-through components, see Chained mappings / pass-through components.

3.4 Process and Generate Files Dynamically

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

1. Read data from multiple XML files in the same directory.
2. Convert each file to a new XML schema.
3. For each source XML file, generate a new XML target file under the new schema.
4. Strip the XML and namespace declaration from the generated files.



Abstract model of the data transformation

We will use three source XML files as example. The files are located in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder, and they are named **bookentry1.xml**, **bookentry2.xml**, and **bookentry3.xml**. Each of the three files stores a single book.

```
<?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>
```

bookentry1.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>
```

bookentry2.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>
```

bookentry3.xml

The source XML files use the **books.xsd** schema available in the following folder: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\**. To convert the source files to a new XML schema, we will use the **library.xsd** schema (available in the same folder). After the transformation, the mapping will generate three files according to this new schema (see the code listings below). We will also configure the mapping so that the name 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.

```
<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>
```

publication1.xml

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

publication2.xml

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

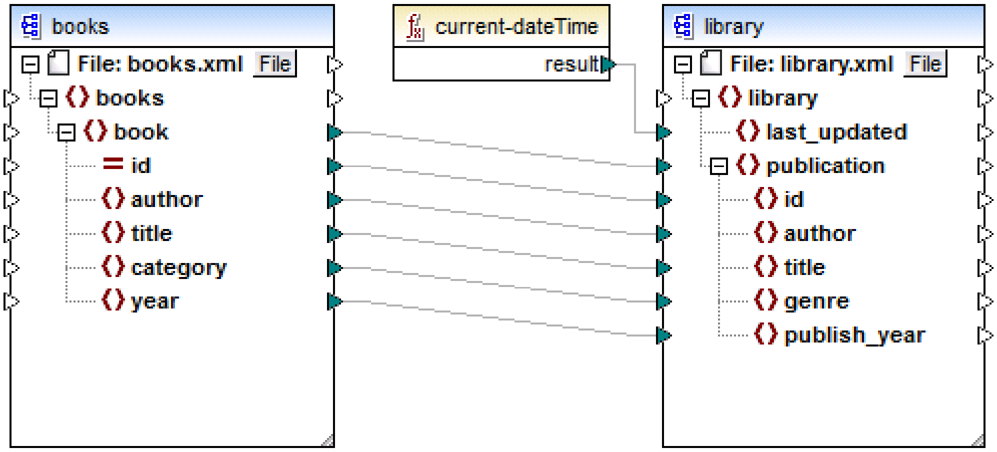
publication3.xml

To achieve the goals, let's take the following steps.

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **BooksToLibrary.mfd** mapping from the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder. You have already designed this mapping in the Convert XML to New Schema tutorial. To begin, open the **BooksToLibrary.mfd** file in MapForce, and save it with a new name, in the same folder.

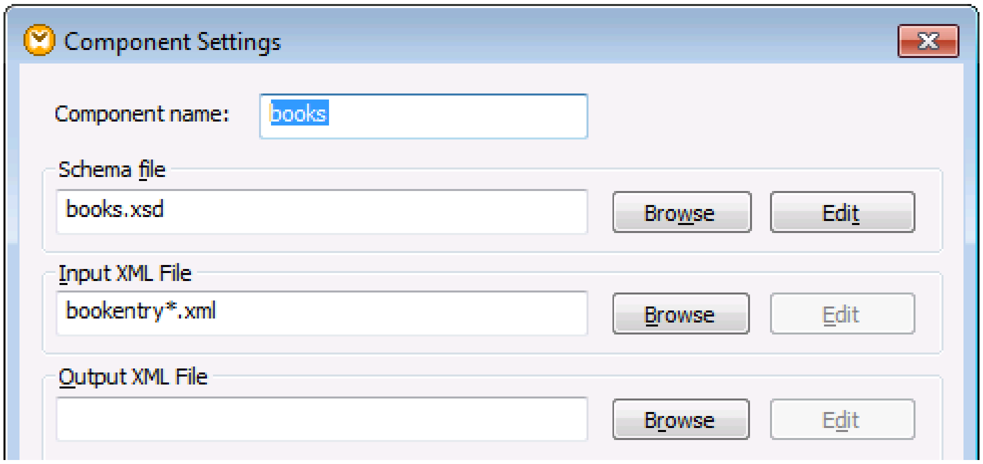
> Make sure to save the new mapping in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder, because it references several files from it.



BooksToLibrary.mfd (MapForce Basic Edition)

Step 2: Configure the input

To instruct MapForce to process multiple XML instance files, double-click the header of the source component. In the Component Settings dialog box, enter **bookentry*.xml** as input file.


Component Settings

Component name: books

Schema file: books.xsd

Input XML File: bookentry*.xml

Output XML File:

Component Settings dialog box

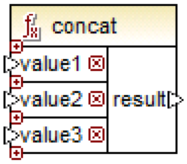
The asterisk (*) wildcard character in the file name instructs MapForce to use as mapping input all the files that have the **bookentry-** prefix. Because the path is a relative one, MapForce will look for all **bookentry-** files in the same directory as the mapping file. Note that you could also enter an absolute path if necessary, while still using the * wildcard character.

Step 3: Configure the output

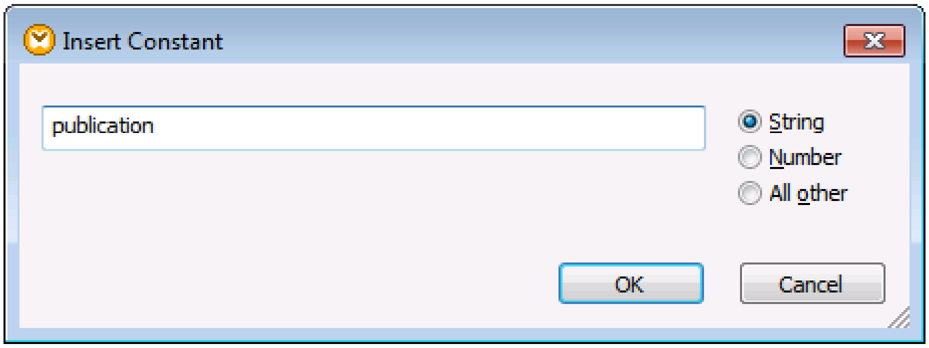
To create the file name of each output file, we will use the **`concat`** function. This function concatenates (joins) all the values supplied to it as argument.

To build the file name using the **concat** function:

1. Search for the **concat** function in the Libraries window and drag it to the mapping area. By default, this function is added to the mapping with two parameters; however, you can add new parameters if necessary. Click the **Add parameter** (⊕) symbol inside the function component and add a third parameter to it. Note that clicking the **Delete parameter** (⊠) symbol deletes a parameter.



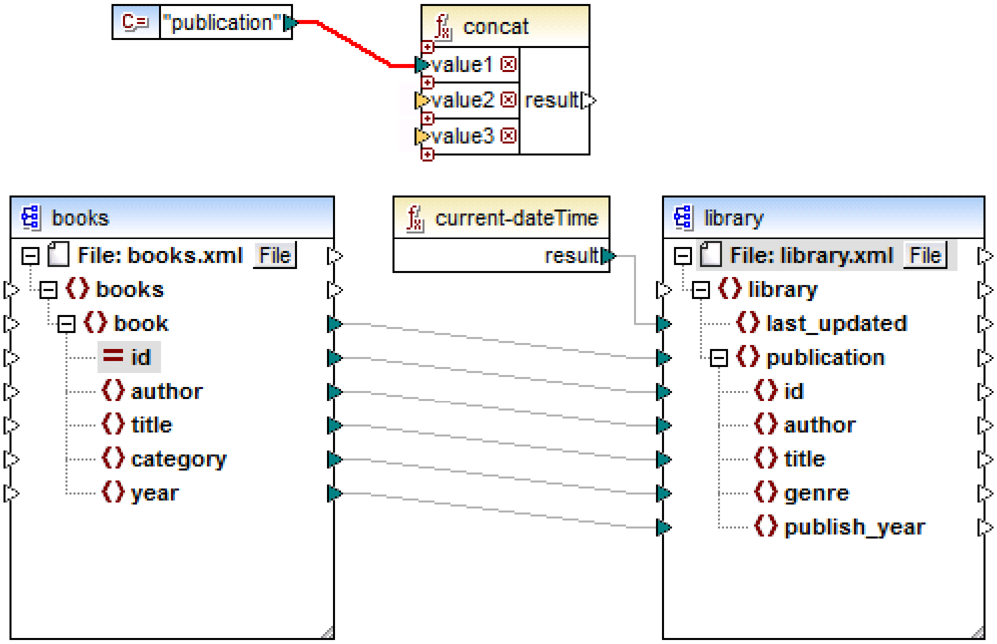
2. Insert a constant (on the **Insert** menu, click **Constant**). When prompted to supply a value, enter "publication" and leave the **String** option unchanged.



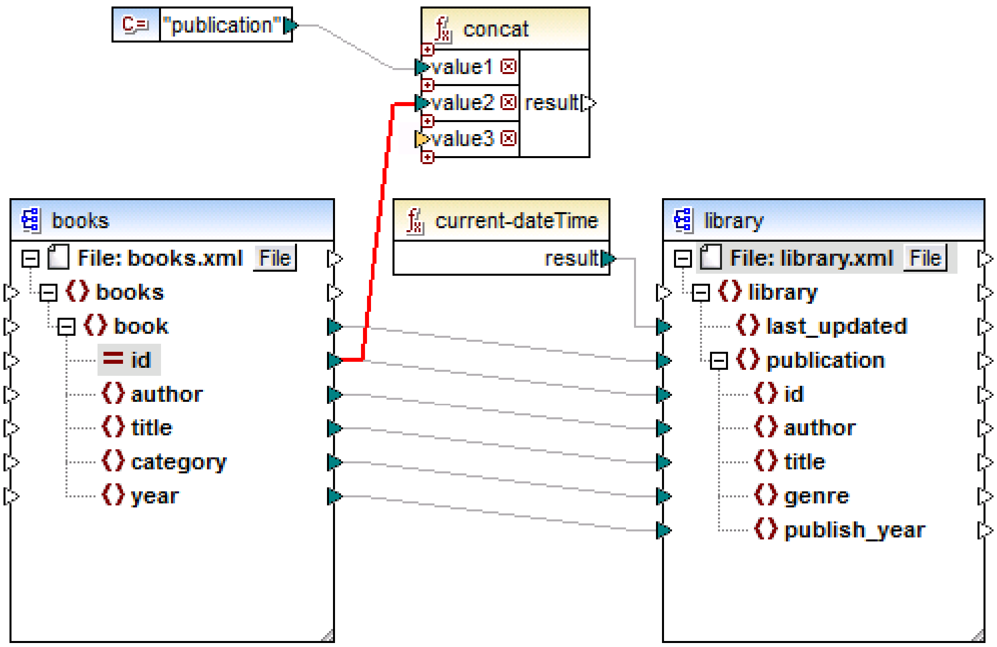
publication

● String
○ Number
○ All other

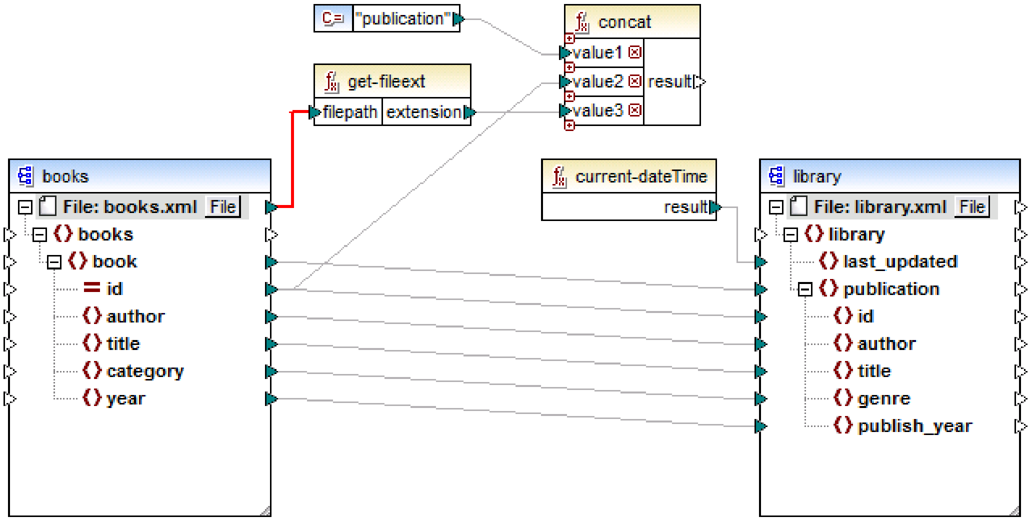
3. Connect the constant with **value1** of the **concat** function.



4. Connect the id attribute of the source component with **value2** of the concat function.



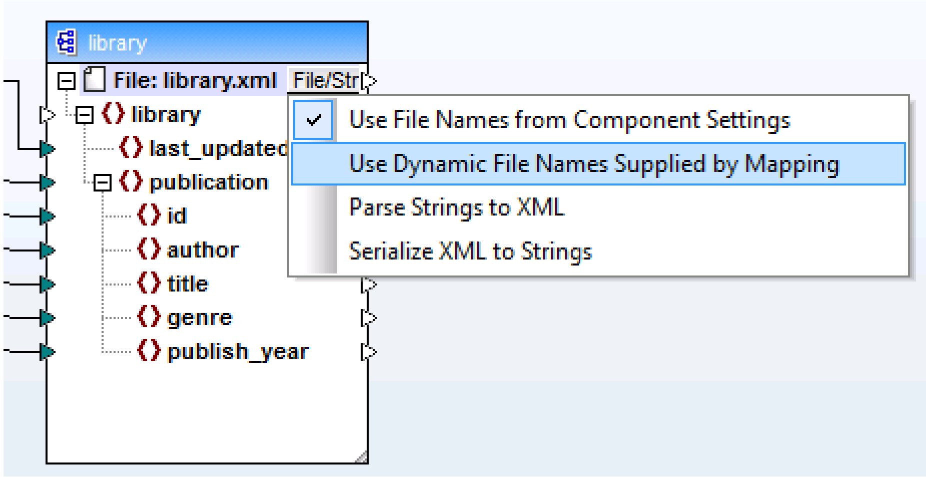
5. Search for the **`get-fileext`** function in the Libraries window and drag it to the mapping area. Create a connection from the top node of the source component (**File: books.xml**) to the **filepath** parameter of this function. Then create a connection from the result of the **`get-fileext`** function 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.



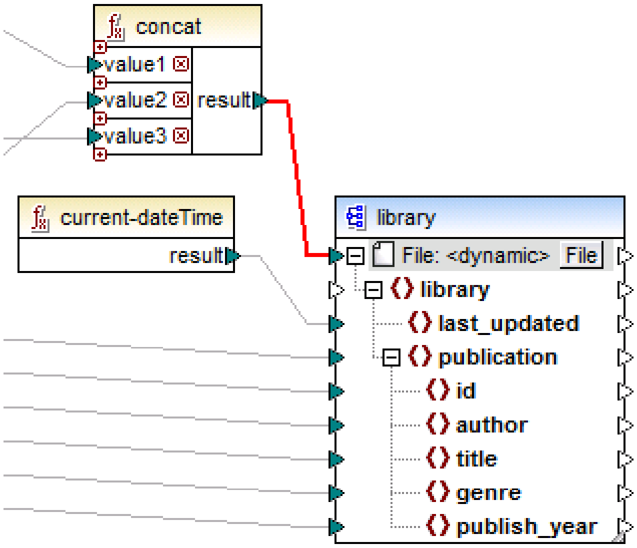
So far, you have provided as parameters to the **`concat`** function the three values which, when joined together, will create the generated file name (for example, **publication1.xml**):

Part	Example
The constant "publication" supplies the constant string value "publication".	publication
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.	1
The **`get-fileext`** function returns the extension of the file name to be generated.	.xml

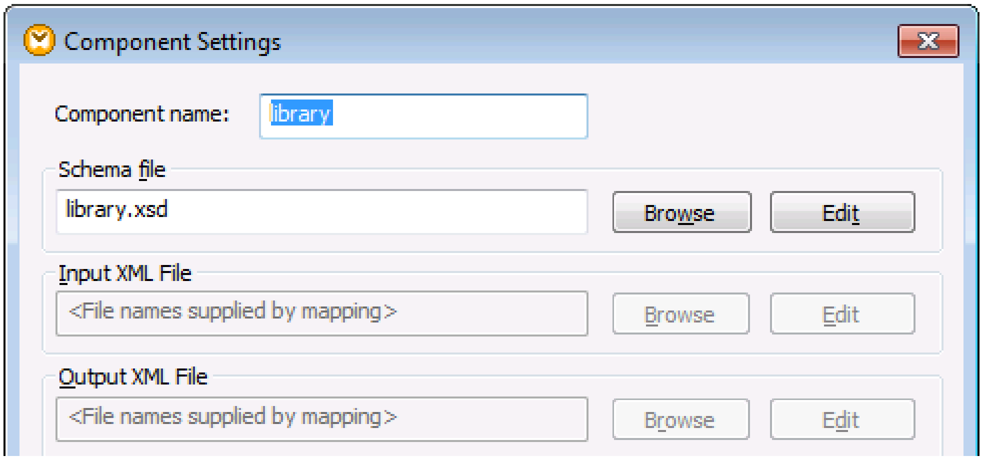
You can now instruct MapForce to actually build the file name when the mapping runs. To do this, click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.



You have now instructed MapForce to generate the instance files dynamically, with whatever name will be provided by the mapping. In this particular example, the name is created by the **`concat`** function; therefore, we will connect the result of the **`concat`** function with the **File: <dynamic>** node of the target component.



If you double-click the target component header at 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>**.


Component Settings

Component name: library

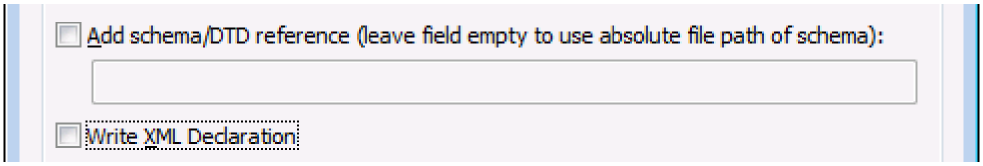
Schema file
library.xsd

Input XML File
<File names supplied by mapping>

Output XML File
<File names supplied by mapping>

This serves as an indication that you have supplied the instance file names dynamically from a mapping, so it is no longer relevant to define them in the component settings.

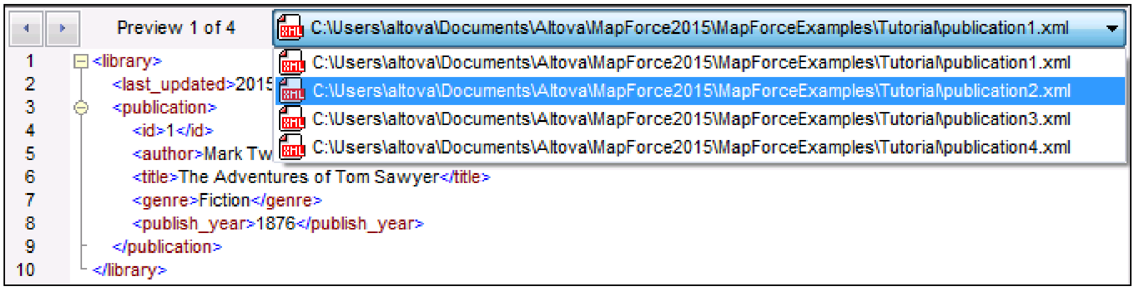
Finally, you need to strip the XML namespace and schema declaration from the target. To achieve this, clear the selection from the **Add schema/DTD reference...** and **Write XML Declaration** check boxes on the Component Settings dialog box.



□ Add schema/DTD reference (leave field empty to use absolute file path of schema):

□ Write XML Declaration

You can now run the mapping and see the result, as well as the name of 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.



Preview 1 of 4

C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Tutorial\publication1.xml

- C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Tutorial\publication1.xml
- C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Tutorial\publication2.xml
- C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Tutorial\publication3.xml
- C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Tutorial\publication4.xml

```
<library>
  <last_updated>2015
  <publication>
    <id>1</id>
    <author>Mark Tw
    <title>The Adventures of Tom Sawyer</title>
    <genre>Fiction</genre>
    <publish_year>1876</publish_year>
  </publication>
</library>
```

Chapter 4

Common Tasks

4 Common Tasks

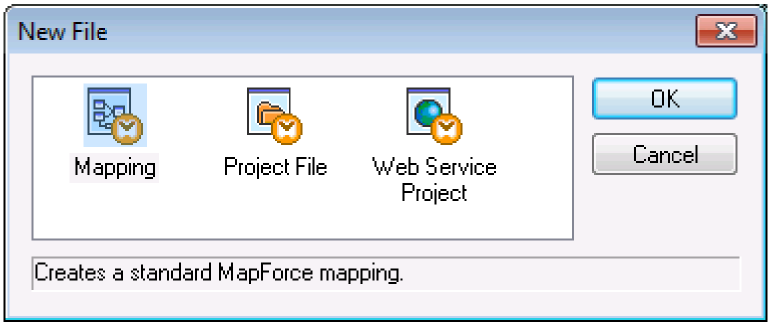
This section describes common MapForce tasks and concepts, such as working with mappings, components, connections, and mapping projects.

4.1 Working with Mappings

A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). 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 mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.

To create a new mapping:

1. Do one of the following:
   - On the **File** menu, click **New**.
   - Click the **New** ( ) toolbar button.
2. Click **Mapping**, and then click **OK**.



Creates a standard MapForce mapping.

Your mapping is now created; however, it does not yet do anything because it is empty. A mapping requires at least two connected components to become valid, so the next step is to add components to the mapping (see Adding Components to the Mapping) and draw connections between components (see Working with Connections).

4.1.1 Adding Components to the Mapping

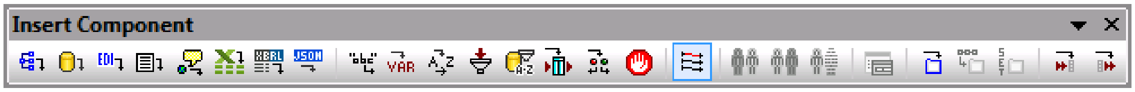
In MapForce, the term "component" is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Databases
- Filters
- Conditions
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files

- Simple input components
- Simple output components
- XML Schemas and DTDs

To add a component to the mapping, do one of the following:

- On the **Insert** menu, click the option relevant for the component type you wish to add (for example, **XML Schema/File**).
- Drag a file from Windows File Explorer onto the mapping area. Note that this operation is possible only for compatible file-based components.
- Click the relevant button on the Insert Component toolbar.



Insert Component toolbar (MapForce Enterprise Edition)

Each component type has specific purpose and behavior. For component types where that is necessary, MapForce walks you through the process by displaying contextual wizard steps or dialog boxes. For example, if you are adding an XML schema, a notification dialog box prompts you to optionally select an instance file as well.

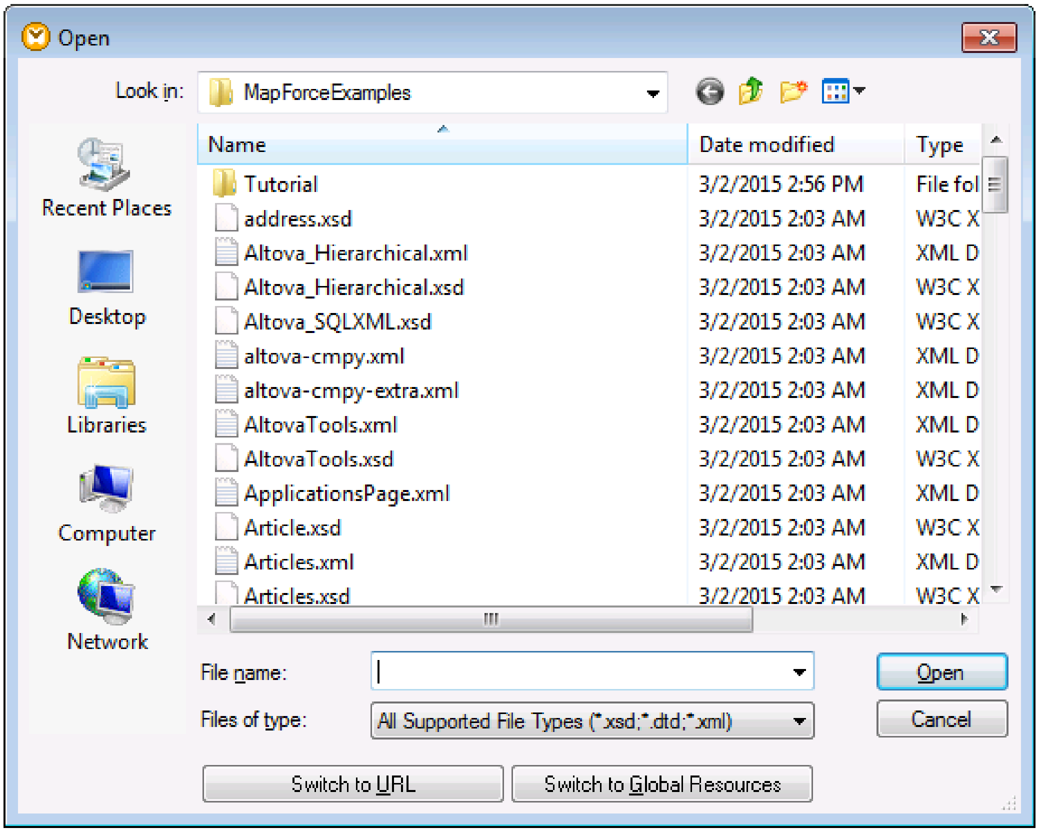
For an introduction to components, see Working with Components. For specific information about each technology supported as mapping source or target, see Data Sources and Targets. For information about MapForce built-in components used to store data temporarily or transform it (such as filtering or sorting), see Designing Mappings.

4.1.2 Adding Components from a URL

In addition to adding local files as mapping components, you can also add files from a URL. Note that this operation is supported when you add a component as source component (that is, your mapping reads data from the remote file). The supported protocols are HTTP, HTTPS, and FTP.

To add a component from a URL:

1. On the **Insert** menu, select the type of the component type you wish to add (for example, **XML Schema/File**).
2. On the **Open** dialog box, click **Switch to URL**.



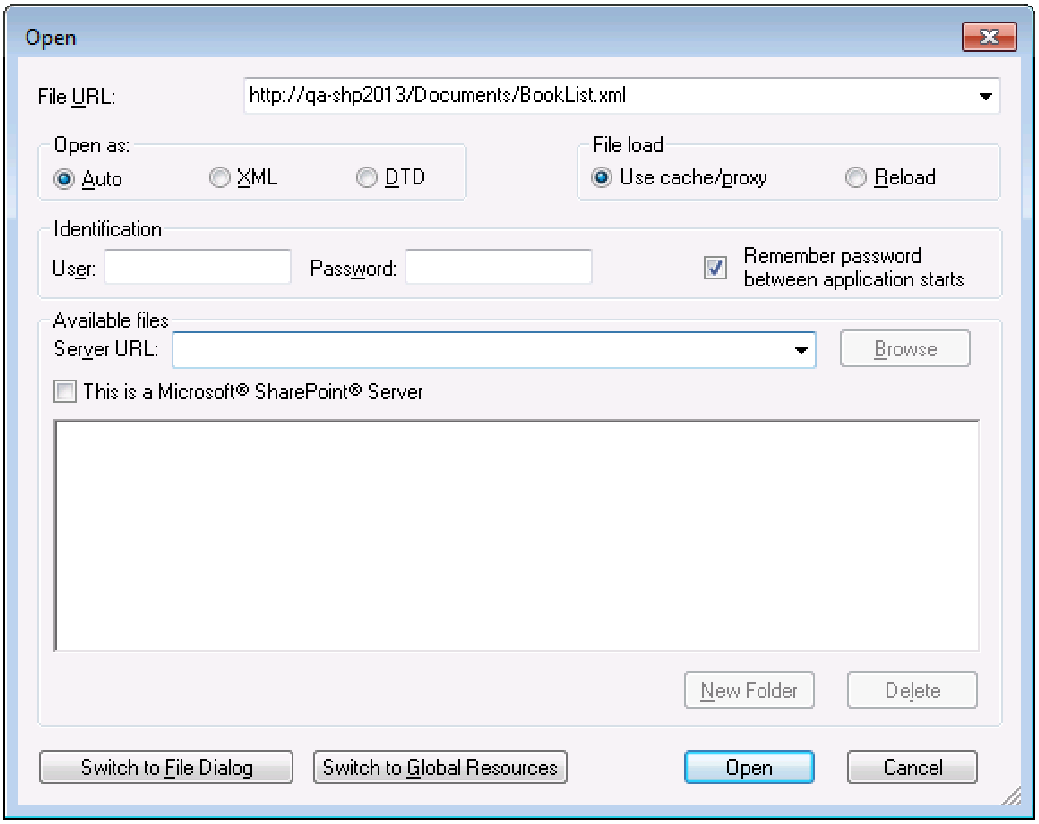
Look in: MapForceExamples

Name	Date modified	Type
Tutorial	3/2/2015 2:56 PM	File fol
address.xsd	3/2/2015 2:03 AM	W3C X
Altova_Hierarchical.xml	3/2/2015 2:03 AM	XML D
Altova_Hierarchical.xsd	3/2/2015 2:03 AM	W3C X
Altova_SQLXML.xsd	3/2/2015 2:03 AM	W3C X
altova-cmpy.xml	3/2/2015 2:03 AM	XML D
altova-cmpy-extra.xml	3/2/2015 2:03 AM	XML D
AltovaTools.xml	3/2/2015 2:03 AM	XML D
AltovaTools.xsd	3/2/2015 2:03 AM	W3C X
ApplicationsPage.xml	3/2/2015 2:03 AM	XML D
Article.xsd	3/2/2015 2:03 AM	W3C X
Articles.xml	3/2/2015 2:03 AM	XML D
Articles.xsd	3/2/2015 2:03 AM	W3C X

File name:

Files of type: All Supported File Types (*.xsd;*.dtd;*.xml)

3. Enter the URL of the file in the **File URL** text box, and click **Open**.



File URL: http://qa-shp2013/Documents/BookList.xml

Open as: ● Auto ○ XML ○ DTD

File load: ● Use cache/proxy ○ Reload

Identification

User: Password:

☑ Remember password between application starts

Available files

Server URL:

□ This is a Microsoft® SharePoint® Server

Make sure that the file type in the **File URL** text box is the same as the file type you specified in step 1.

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

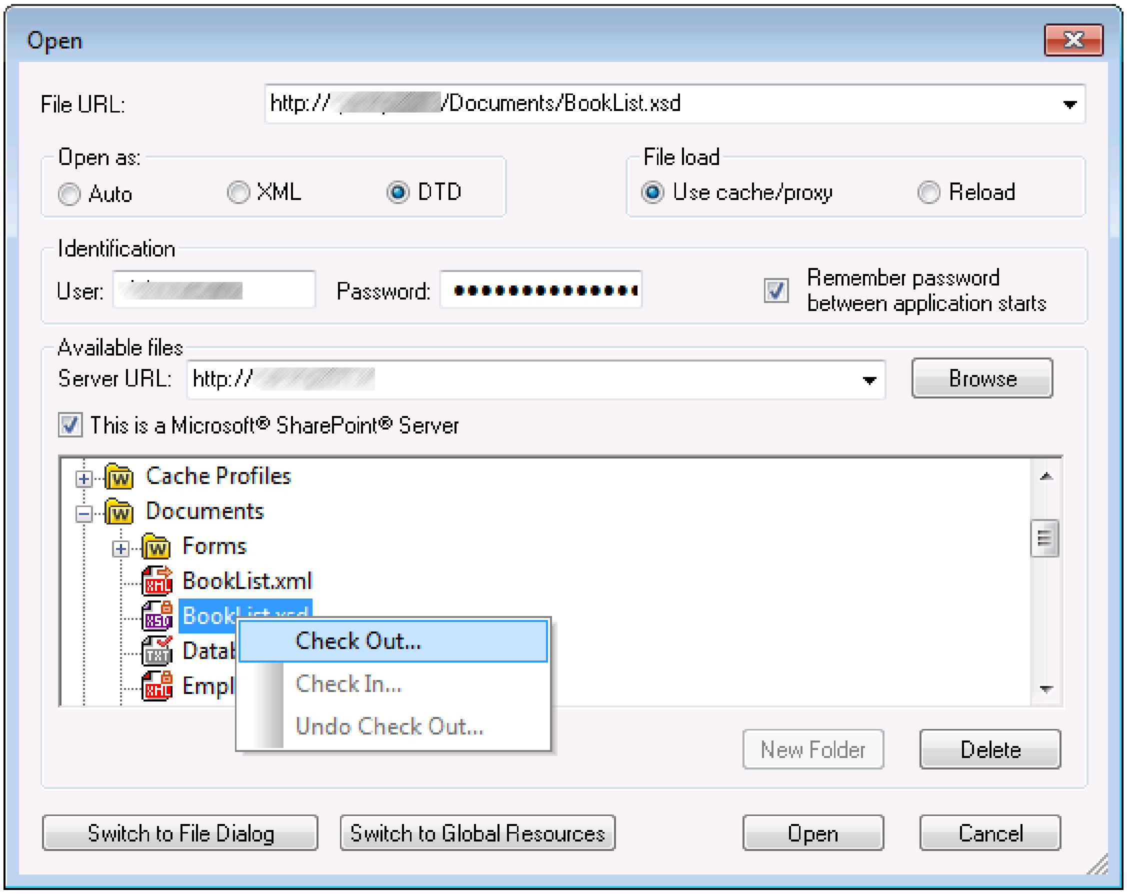
The **Open As** setting defines the grammar for the parser when opening the file. The default and recommended option is **Auto**.

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

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 that you choose to open the same file type as specified in step 1 above; otherwise, errors will occur.

If the server is a Microsoft SharePoint Server, select the **This is a Microsoft SharePoint Server** check box. Doing so 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 in

MapForce to read data from it, right-click the file and select **Check Out**. To check in any file that was previously checked out by you, right-click the file and select **Check In**.



Open

File URL: http://[redacted]/Documents/BookList.xsd

Open as: ○ Auto ○ XML ● DTD

File load: ● Use cache/proxy ○ Reload

Identification

User: [redacted] Password: •••••••••••••••

☑ Remember password between application starts

Available files

Server URL: http://[redacted]

☑ This is a Microsoft® SharePoint® Server

Open dialog box (in Switch to URL mode)

4.1.3 About Data Streaming

Data streaming is a MapForce built-in mechanism that allows you to use arbitrarily large data sources as input or output to your mappings. Data streaming should not be confused with stream objects in MapForce generated code. (The latter represent a possible way of handling data if you integrate MapForce generated code with a custom C# and Java application.)

Data streaming applies to the following data sources:

- XML files
- CSV files
- Fixed-length field files
- Databases

When you use any of the above data sources as input or output in your mappings, MapForce treats the data source as an open stream of data, and processes its contents sequentially, instead of loading all data into the memory.

Note: Data streaming is possible only if you have selected BUILT-IN as transformation language (see Selecting a transformation language).

Memory usage considerations

When you work with mapping inputs and outputs that are data streaming candidates, “Out of memory” errors can occur if your mapping requires random access to the input source.

For example, let’s assume that your mapping contains a component that applies a **`group-by`** function on the source data. If you apply the **`group-by`** function on the entire tree structure of the input file, this would require the entire source file to be loaded into memory, and, consequently, file streaming would no longer be possible. The same is true for any operation which would require the whole contents of the mapping source to be loaded into memory, such as sorting.

When situations such as the one described above occur, the transformation will nevertheless complete successfully if there is enough virtual memory and disk space available on your system.

4.1.4 Selecting a Transformation Language

To meet your data mapping needs, MapForce provides the ability to choose between various transformation languages.

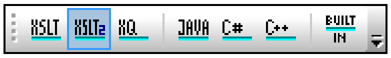
By default, MapForce provides a robust, built-in engine capable of performing the same transformations supported in other languages. When you deploy MapForce mappings to MapForce Server, the built-in engine executes them without the need for any external processors. Furthermore, if you require minimal or no manual intervention in your data transformation process, you can use FlowForce Server to automate mapping processes by means of scheduled jobs.

Consider choosing the transformation language after testing several approaches and determining what works best for your data. The available transformation languages are as follows:

- BUILT-IN (This is the default native transformation engine used by MapForce.)
- C++
- C#
- Java
- XQuery
- XSLT 1.0
- XSLT 2.0

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

- On 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.



Note: Some mapping inputs and outputs are not supported by certain languages. For example, if you use a database as mapping input or output, you cannot generate XSLT code.

Therefore, if you attempt to generate the code or preview the output of a mapping that has sources or targets not supported by the selected language, MapForce displays a relevant notification message.

Using the BUILT-IN option

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

It is recommended to set the transformation language to BUILT-IN in the following cases:

- As default option, when you do not necessarily need to use a specific language to transform data.
- If you are processing large files and memory usage is a concern.

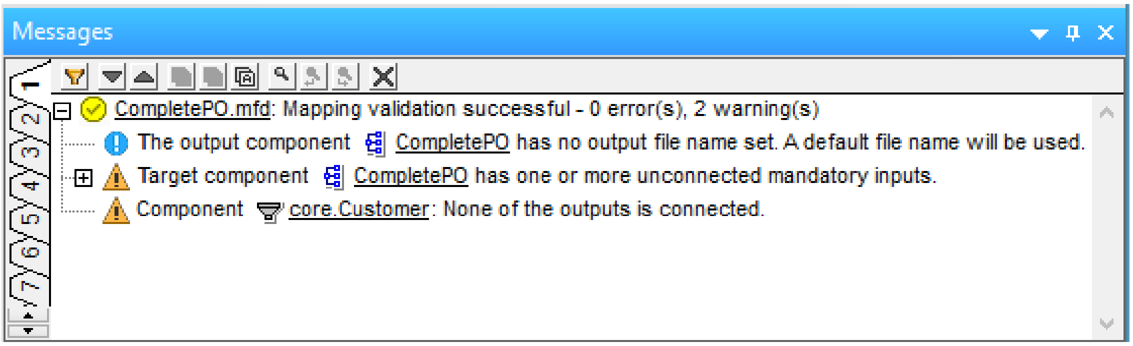
4.1.5 Validating Mappings

MapForce validates mappings automatically, when you click the **Output** tab to preview the transformation result. You can also validate a mapping explicitly, before attempting to preview its result. This helps you identify and correct potential mapping errors and warnings before the mapping is run. Note that running a mapping may generate additional runtime errors or warnings depending on the processed data, for example, when values mapped to attributes are overwritten.

To validate a mapping explicitly, do one of the following:

- On the **File** menu, click **Validate Mapping**.
- Click the **Validate** (✓) toolbar button.

The Messages window displays the validation results, for example:



CompletePO.mfd: Mapping validation successful - 0 error(s), 2 warning(s)
The output component CompletePO has no output file name set. A default file name will be used.
Target component CompletePO has one or more unconnected mandatory inputs.
Component core.Customer: None of the outputs is connected.

Messages window

When you validate a mapping, MapForce checks for the validity of the mapping (such as incorrect or missing connections, unsupported component kinds), and the validation result is then 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 Message window may additionally display any of the following message types: information messages, warnings, and errors.

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

To highlight on the mapping area the component or structure which triggered the information, warning, or error message, click the underlined text in the Messages window.

For components that transform data (such as functions or variables), MapForce validation works as follows:

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

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

Other buttons in the Messages window enable you to take the following actions:

- Filter the message by types (for example, to show only errors or warnings)
- Move up or down through the entries
- Copy the message text to the clipboard
- Find a specific text in the window
- Clear the Messages window.

For general information about the Messages window, see User Interface Overview.

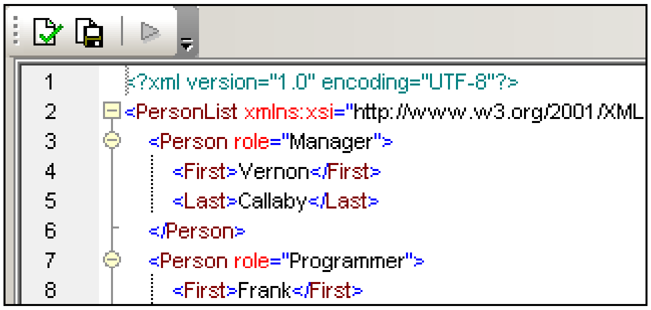
4.1.6 Validating the Mapping Output

After you click the **Output** tab to preview the mapping, the resulting output becomes available in the Output pane. You can validate this output against the schema associated with it. For example, if the mapping transformation generates an XML file, then the resulting XML document can be validated against the XML schema.

For XML files, you can specify the schema associated with the instance file in the **Add Schema/ DTD reference** field of the Component Settings dialog box (see XML Component Settings). The path specifies where the schema file referenced by the produced XML output is to be located. This ensures that the output instance can be validated when the mapping is executed. You can enter an `http://` address in this field, as well as an absolute or relative path. If you do not select the **Add Schema/DTD reference** field, then the validation of the output file against the schema is not possible. If you select this check box but leave it empty, then the schema filename of the Component Settings dialog box is generated into the output and the validation is done against it.

To validate the mapping output, do one of the following:

- Click the **Validate Output** toolbar button.

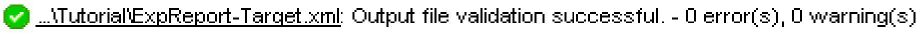


```
<?xml version="1.0" encoding="UTF-8"?>
<PersonList xmlns:xsi="http://www.w3.org/2001/XML
    <Person role="Manager">
        <First>Vernon</First>
        <Last>Callaby</Last>
    </Person>
    <Person role="Programmer">
        <First>Frank</First>
```

- On the **Output** menu, click **Validate Output File**.

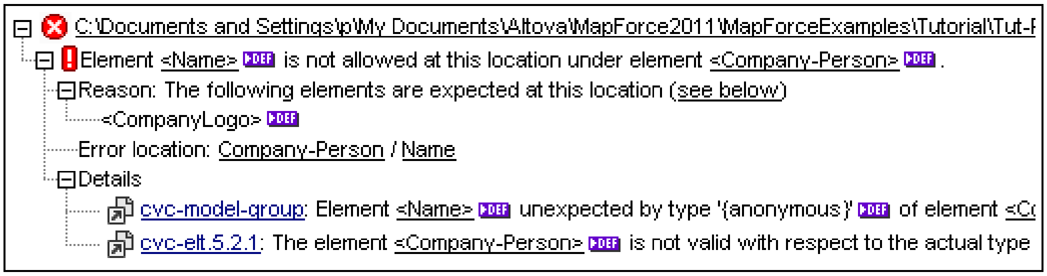
Note: The **Validate Output** button and its corresponding menu command (**Output | Validate Output File**) are enabled only if the output file supports validation against a schema.

The result of the validation is displayed in the Messages window, for example:



...\Tutorial\ExpReport-Target.xml: Output file validation successful. - 0 error(s), 0 warning(s)

If the validation was not successful, the message contains detailed information on the errors that occurred.



```
C:\Documents and Settings\p\My Documents\Altova\MapForce2011\MapForceExamples\Tutorial\Tut-I
  Element <Name> DEF is not allowed at this location under element <Company-Person> DEF .
    Reason: The following elements are expected at this location (see below)
      <CompanyLogo> DEF
    Error location: Company-Person / Name
    Details
      cvc-model-group: Element <Name> DEF unexpected by type '{anonymous}' DEF of element <C
      cvc-elt.5.2.1: The element <Company-Person> DEF is not valid with respect to the actual type
```

The validation message contains a number of hyperlinks you can click for more detailed information:

- Clicking the file path opens the output of the transformation in the **Output** tab of MapForce.
- Clicking <ElementName> link highlights the element in the **Output** tab.
- Clicking the DEF icon opens the definition of the element in XMLSpy (if installed).
- Clicking the hyperlinks in the Details subsection (e.g., cvc-model-group) opens a description of the corresponding validation rule on the https://www.w3.org/ website.

4.1.7 Previewing the Output

When working with MapForce mappings, you can preview the resulting 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 within MapForce before attempting to process the generated code externally.

When you choose to preview the mapping results, MapForce executes the mapping and populates the Output pane with the resulting output.

Once data is available in the Output pane, you can validate and save it if necessary (see Validating the Mapping Output). You can also use the **Find** command (**Ctrl + F** key combination) to quickly locate a particular text pattern within the output file (see also Searching in Text View).

Any errors, warning, or information messages related to the mapping execution are displayed in the Messages window (see User Interface Overview).

To preview the transformation output:

- Click the **Output** tab under the Mapping window. MapForce executes the mapping using the transformation language selected in the Language toolbar and populates the Output pane with the resulting output.

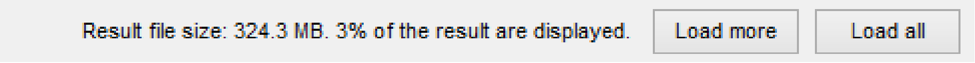
Note: If you select C++, C#, or Java as transformation language, MapForce executes the mapping using its built-in transformation engine. The result that appears in the Output pane is the same as if the Java, C++, or C# code had been generated, compiled and executed.

To save the transformation output, do one of the following:

- On the **Output** menu, click **Save Output File**.
- Click the **Save Generated Output** toolbar button.

Partial output preview

When you are previewing large output files, MapForce limits the amount of data displayed in the Output pane. More specifically, MapForce displays only a part of the file in the Output pane, and a **Load more...** button appears in the lower area of the pane. Clicking the **Load more...** button appends the next file part to the currently visible data, and so on.



Result file size: 324.3 MB. 3% of the result are displayed.

Note: The **Pretty-print** button becomes active when the complete file has been loaded into the Output pane.

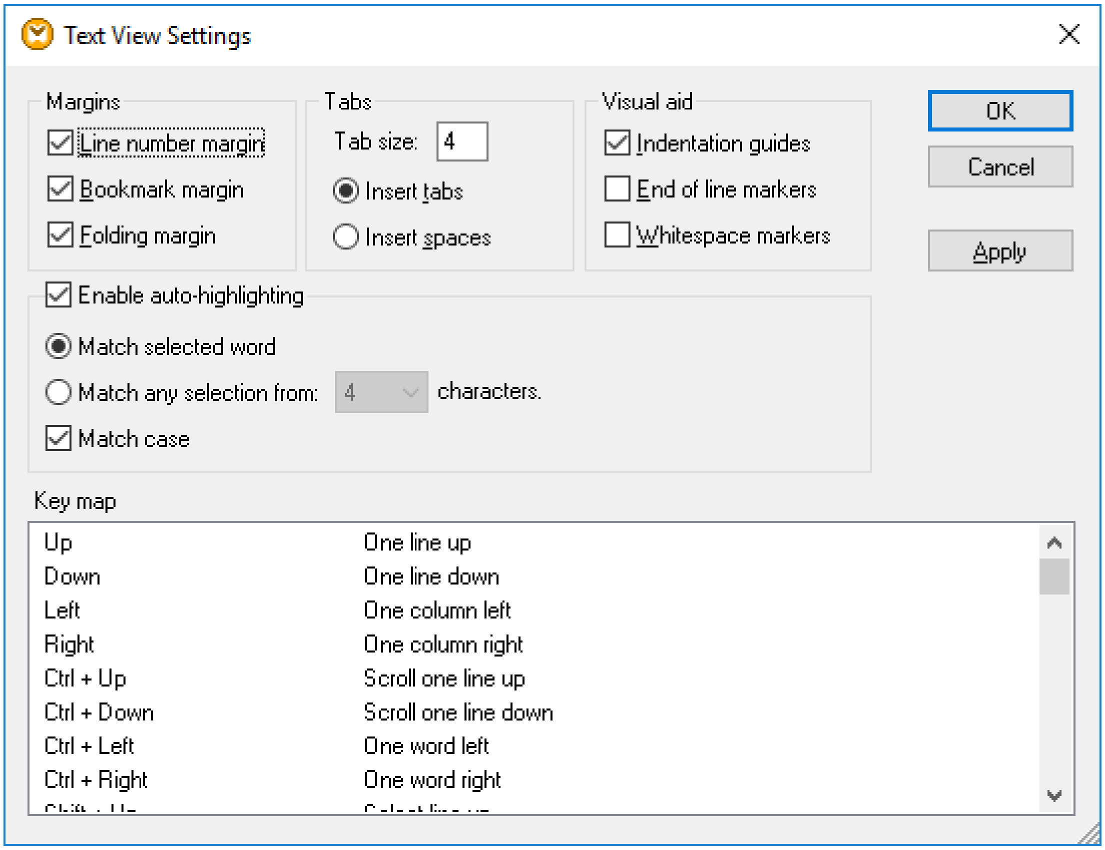
You can configure the preview settings from the **General** tab of the **Options** dialog box (see Changing the MapForce Options).

4.1.8 Text View Features

The **Output** pane, the **XSLT** pane, as well as the **XQuery** pane have multiple visual aids to make the display of text easier. These include:

- Line Numbers
- Syntax Coloring
- Bookmarks
- Source Folding
- Indentation Guides
- End-of-Line and Whitespace Markers
- Zooming
- Pretty-printing
- Word wrapping
- Text highlighting

Where applicable, you can toggle or customize the features above from the **Text View Settings** dialog box. Settings in the **Text View Settings** dialog box apply to the entire application—not only to the active document.



Text View Settings

Margins
☑ Line number margin
☑ Bookmark margin
☑ Folding margin

Tabs
Tab size: 4
● Insert tabs
○ Insert spaces

Visual aid
☑ Indentation guides
□ End of line markers
□ Whitespace markers

☑ Enable auto-highlighting
● Match selected word
○ Match any selection from: 4 characters.
☑ Match case

Key map

Up	One line up
Down	One line down
Left	One column left
Right	One column right
Ctrl + Up	Scroll one line up
Ctrl + Down	Scroll one line down
Ctrl + Left	One word left
Ctrl + Right	One word right
Shift + Up	Select line up

Text View Settings dialog box

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

- On the **Output** menu, select **Text View Settings**.
- Click the **Text View Settings** toolbar button.
- Right-click 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.

Text View toolbar

For reference to all applicable shortcuts, see the "Key Map" section of the **Text View Settings** dialog box illustrated above.

Line numbers

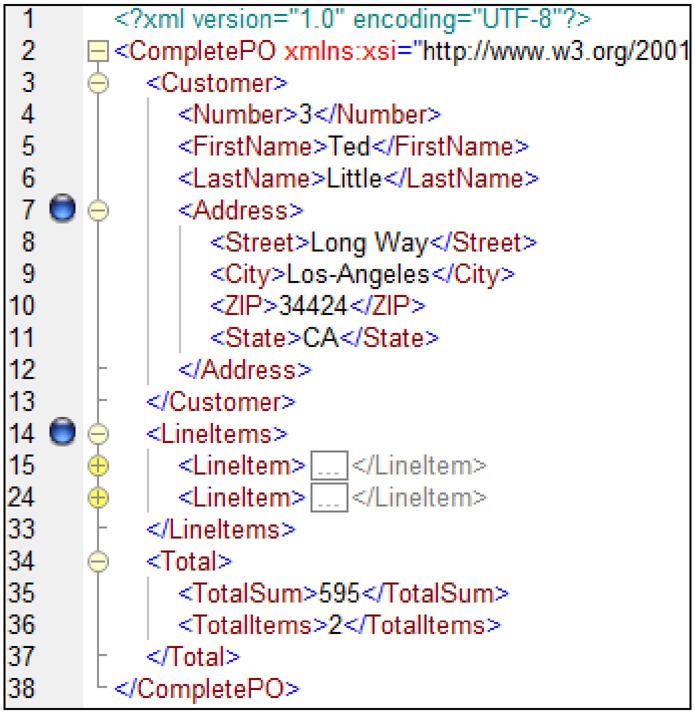
Line numbers are displayed in the line numbers margin, which can be toggled on and off in the **Text View Settings** dialog box. When a section of text is collapsed, the line numbers of the collapsed text are also hidden.

Syntax coloring

Syntax coloring is applied according to 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.

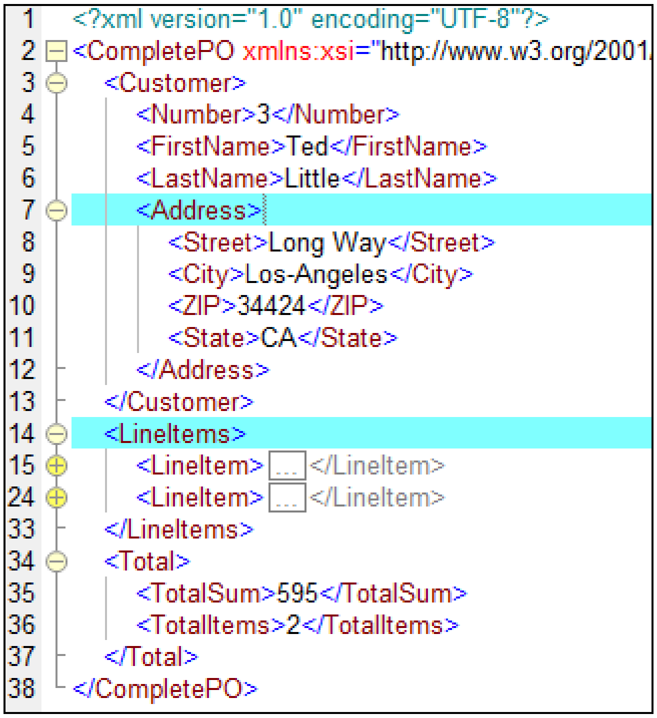
Bookmarks

Lines in the document can be bookmarked for quick reference and access. If the bookmarks margin is toggled on, bookmarks are displayed in the bookmarks margin.



```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001
  <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>...</LineItem>
    <LineItem>...</LineItem>
  </LineItems>
  <Total>
    <TotalSum>595</TotalSum>
    <TotalItems>2</TotalItems>
  </Total>
</CompletePO>
```

Otherwise, bookmarked lines are highlighted in cyan.



```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/
  <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> ... </LineItem>
    <LineItem> ... </LineItem>
  </LineItems>
  <Total>
    <TotalSum>595</TotalSum>
    <TotalItems>2</TotalItems>
  </Total>
</CompletePO>
```

The bookmarks margin can be toggled on or off in the **Text View Settings** dialog box.

You can edit and navigate bookmarks using the following commands:

Insert/Remove Bookmark (Ctrl + F2)

Go to Next Bookmark (F2)

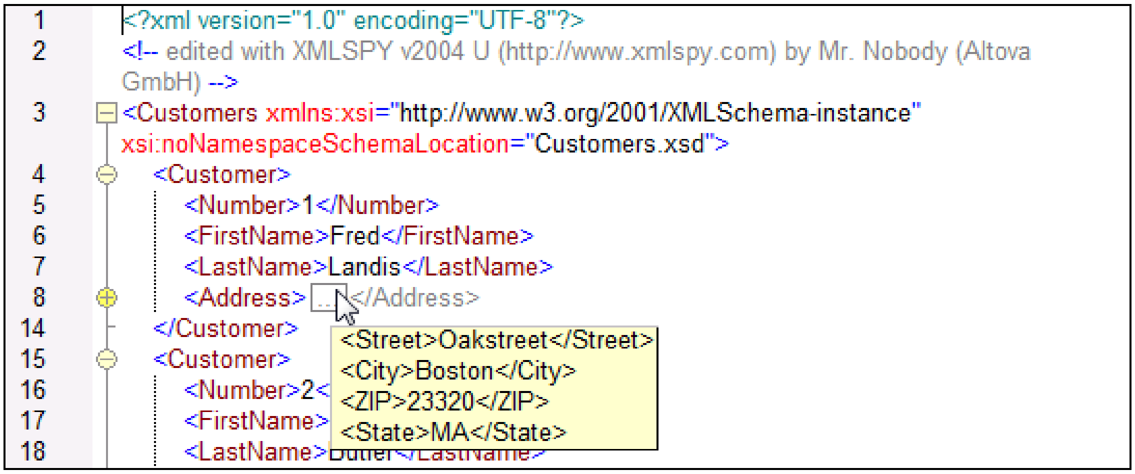
Go to Previous Bookmark (Shift + F2)

Delete All Bookmarks (Ctrl + Shift + F2)

The commands above are available in the **Output** menu. Bookmark commands are also available through the context menu, when you right-click the **Output** (or **XSLT**, or **XQuery**) pane.

Source folding

Source folding refers to the ability to expand and collapse nodes and is displayed in the source folding margin. The margin can be toggled on and off 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. 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 in the tooltip, an additional ellipsis appears at the end of the tooltip.



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 U (http://www.xmlspy.com) by Mr. Nobody (Altova
GmbH) -->
<Customers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Customers.xsd">
    <Customer>
        <Number>1</Number>
        <FirstName>Fred</FirstName>
        <LastName>Landis</LastName>
        <Address>...</Address>
    </Customer>
    <Customer>
        <Number>2<
        <FirstName>
        <LastName>Butler</LastName>
```

```
<Street>Oakstreet</Street>
<City>Boston</City>
<ZIP>23320</ZIP>
<State>MA</State>
```

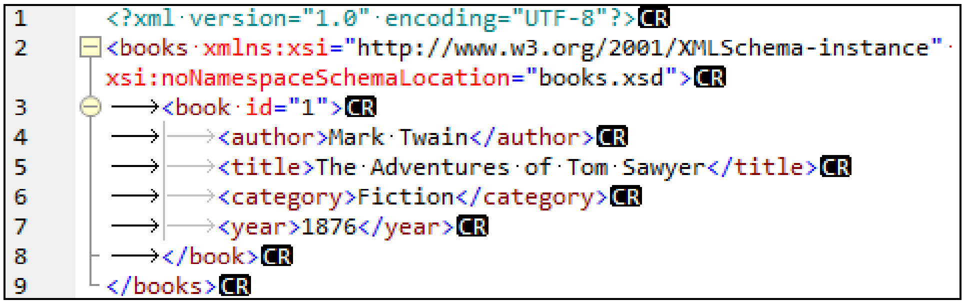
Indentation guides

Indentation guides are vertical dotted lines that indicate the extent of a line's indentation. They can be toggled on and off in the **Text View Settings** dialog box.

Note: The **Insert tabs** and **Insert spaces** options take effect when you use the **Output | Pretty-Print XML text** option.

End-of-line markers, whitespace markers

End-of-line (EOL) markers and whitespace markers can be toggled on in the **Text View Settings** dialog box. The image below shows a document where both end-of-line and whitespace markers are visible. An arrow represents a tab character, a "CR" is a carriage return, and a dot represents a space character.



```
<?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
```

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 offset from its parent by four space characters. This can be customized from the **Text View Settings** dialog box.

To pretty-print an XML document, select the **Output | Pretty-Print XML Text** menu command, or click the **Pretty Print** toolbar button.

Word wrapping

To toggle word wrapping in the currently active document, select the **Output | Word Wrap** menu command, or click the **Word Wrap** toolbar button.

Text highlighting

When you select text, all matches in the document of the text selection that you make are highlighted automatically. The selection is highlighted in pale blue, and matches are highlighted in pale orange. The selection and its matches are indicated in the scroll bar by gray marker-squares. The current cursor position is given by the blue cursor-marker in the scroll bar.

To switch text highlighting on, select **Enable auto-highlighting** in the Text View Settings dialog box. A selection can be defined to be an entire word or a fixed number of characters. You can also specify whether casing should be taken into account or not.

For a 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. In this case, one-character selections will not be matched, but a selection consisting of two or more characters will be matched. So, in this case, if you select `t`, then no matches will be shown; selecting `ty` will show all `ty` matches; selecting `typ` will show all `typ` matches; and so on.

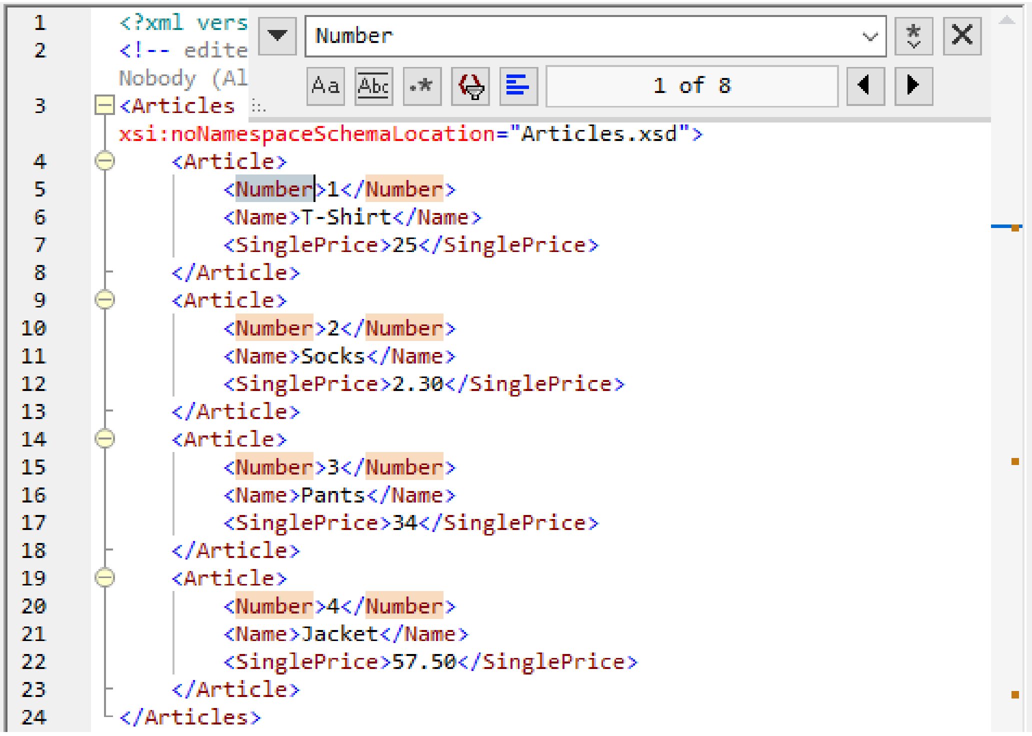
For word searches, the following are considered to be separate words: element names (without angular brackets), the angular brackets of element tags, attribute names, and attribute values without quotes.

4.1.9 Searching in Text View

The text in the **Output** pane, the **XQuery** pane, as well as the **XSLT** pane can be searched using an extensive set of options and visual aids.

To start a search , press **Ctrl+F** (or select the menu command **Edit | Find**). You can then search in the entire document or within a text selection for a search term that you enter in the dialog.

- Enter a string to find, or use the combo box to select a string from one of the last 10 strings.
- When you enter or select a string to find, all matches are highlighted and the positions of the matches are indicated by beige markers in the scroll bar.
- The currently selected match has a different highlight color than the other matches, and its position is indicated in the scroll bar by the dark blue cursor-marker.
- The total number of matches is listed below the search term field, together with the index position of the currently selected match. For example, `2 of 4` indicates that the second of four matches is currently selected.
- You can move from one match to the next, in both directions, by selecting the **Previous** ◀ **(Shift+F3)** and **Next** ▶ **(F3)** buttons at bottom right.



```
<?xml vers
<!-- edite
Nobody (Al
<Articles
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>
```

To close the Find dialog, click the **Close** ✕ button at top right, or press **Esc**.

Note the following points:

- The Find dialog is *modeless*. This means that it can remain open while you continue to use Text View.
- If text is selected prior to opening the dialog box, then the selected text is automatically inserted into the search term field.
- To search within a selection, do the following: (i) Mark the selection; (ii) Toggle on the *Find in Selection* ☰ option to lock the selection; (iii) Enter the search term. To search within another selection, unlock the current selection by toggling off the *Find in Selection* ☰ option, then make the new selection and toggle on the *Find in Selection* ☰ option.
- After the Find dialog is closed, you can repeat the current search by pressing **F3** for a forward search, or **Shift+F3** for a backward search. The Find dialog will appear again in this case.

Find options

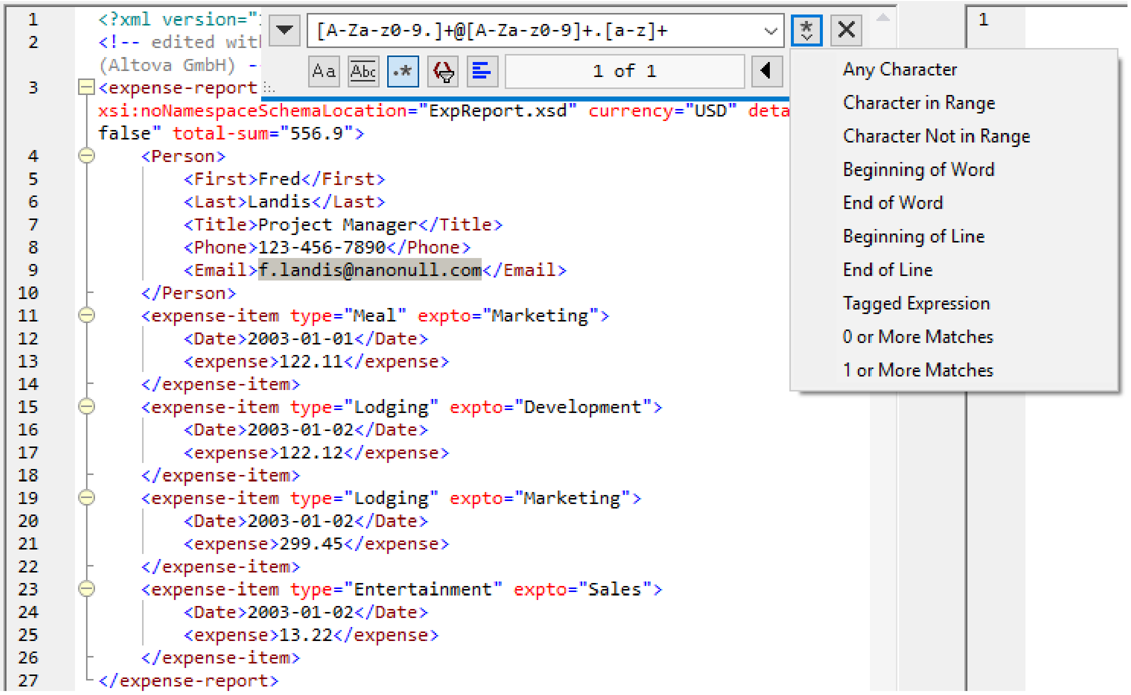
Find criteria can be specified via buttons located below the search term field. When an option is toggled on, its button color changes to blue. You can select from the following options:

Option	Icon	Description
Match case	Aa	Performs a case-sensitive search when toggled on ("Address" is not the same as "address").

Option	Icon	Description
Match whole word	Abc	Only the exact words in the text will be matched. For example, for the input string *fit*, with **Match whole word** toggled on, only the word *fit* will match the search string; the *fit* in *fitness*, for example, will not.
Regular expression	.*	If toggled on, the search term will be read as a regular expression. See "Using regular expressions" below.
Find anchor	⌶	When a search term is entered, the matches in the document are highlighted and one of these matches will be marked as the current selection. The **Find anchor** toggle determines whether that first current selection is made relative to the cursor position or not. If **Find anchor** is toggled on, then the first currently selected match will be the next match from the current cursor location. If **Find anchor** is toggled off, then the first currently selected match will be the first match in the document, starting from the top.
Find in selection	☰	When toggled on, locks the current text selection and restricts the search to the selection. Otherwise, the entire document is searched. Before selecting a new range of text, unlock the current selection by toggling off the Find in Selection option.

Using regular expressions

You can use regular expressions (regex) to find a text string. To do this, first, switch the *Regular expression* .* option on. This specifies that the text in the search term field is to be evaluated as a regular expression. Next, enter the regular expression in the search term field. For help with building a regular expression, click the **Regular Expression Builder** *⌄ button, which is located to the right of the search term field. Click an item in the Builder to enter the corresponding regex metacharacter/s in the search term field. The screenshot below shows a simple regular expression to find email addresses.



```
[A-Za-z0-9.]+@[A-Za-z0-9]+.[a-z]+
```

1 of 1

```
<?xml version="
<!-- edited wit
(Altova GmbH) -
<expense-report
xsi:noNamespaceSchemaLocation="ExpReport.xsd" currency="USD" deta
false" total-sum="556.9">
    <Person>
        <First>Fred</First>
        <Last>Landis</Last>
        <Title>Project Manager</Title>
        <Phone>123-456-7890</Phone>
        <Email>f.landis@nanonull.com</Email>
    </Person>
    <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>
</expense-report>
```

The following custom set of regular expression metacharacters are supported when finding and replacing text.

`.`	Matches any character. This is a placeholder for a single character.
`(abc)`	The `(` and `)` metacharacters mark the start and end of a tagged expression. Tagged expressions may be useful when you need to tag ("remember") a matched region for the purpose of referring to it later (back-reference). Tagged expressions are similar to matched subexpressions (indexed groups) in the .NET flavour of regular expressions. 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 literally explained as follows: match the string "the" (and remember it as a tagged region), followed by a space character, followed by a back-reference to the tagged region matched previously.
`\n`	Where `n` is `1` through `9` , n refers to the first through ninth tagged region (see above).
`\<`	Matches the start of a word.
`\>`	Matches the end of a word.
`\`	Escapes the character following the backslash. In other words, the expression \x allows you to use the character `x` literally. For example, `\[` would be interpreted as `[` and not as the start of a character set.
`[...]`	Matches any characters in this set. For example, **`[abc]`** matches any of the characters `a`, `b` or `c`. You can also use ranges: for example **`[a-z]`** for any lower case character.

`[^...]`	Matches any characters not in this set. For example, **`[^A-Za-z]`** matches any character except an alphabetic character.
`^`	Matches the start of a line (unless used inside a set, *see above*).
`$`	Matches the end of a line. For example, **`A+$`** matches one or more `A`'s at end of line.
`*`	Matches zero or more occurrences of the preceding expression. For example, **`Sa*m`** matches `Sm`, `Sam`, `Saam`, `Saaam` and so on.
`+`	Matches one or more occurrences of the preceding expression. For example, **`Sa+m`** matches `Sam`, `Saam`, `Saaam` and so on.

Finding special characters

You can search for any the following special characters within text, provided that the **Regular expression** option .* is enabled:

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

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

4.1.10 Previewing the XSLT Code

You can preview the XSLT code generated by MapForce if you selected XSLT 1.0 or XSLT 2.0 as data transformation language (see Selecting a transformation language).

To preview the generated XSLT 1.0 (or XSLT 2.0) code, do one of the following:

- To preview the XSLT 1.0 code, click the **XSLT** tab under the Mapping window.
- To preview the XSLT 2.0 code, click the **XSLT2** tab under the Mapping window.

Note: The XSLT (or XSLT2) tab becomes available if you have selected XSLT (or XSLT2, respectively) as transformation language.

4.1.11 Generating XSLT Code

To generate XSLT code:

1. Select the menu item **File | Generate code in | XSLT 1.0 (XSLT 2.0)**.
2. Select the folder 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.

The name of the generated .xslt file has the form **<A>MapTo<B>.xslt**, where:

- "<A>" is the value of the **Application Name** field in mapping settings (see Changing the Mapping Settings).

- "<B>" 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 (see Changing the Component Settings).

The folder where the .xslt file is saved also contains a batch file called **DoTransform.bat** which can be run with RaptorXML Server to transform the data (see Automation with RaptorXML Server).

To run the transformation with RaptorXML Server:

1. Download and install RaptorXML from the download page (https://www.altova.com/download#server).
2. Start the **DoTransform.bat** batch file located in the previously designated output folder.

Note that you might need to add the RaptorXML installation location to the **path** variable of the Environment Variables. You can find the RaptorXML documentation on the website documentation page (https://www.altova.com/documentation).

4.1.12 Previewing the XQuery Code

You can preview the XQuery code generated by MapForce if you selected **XQuery** as data transformation language (see Selecting a transformation language).

To preview the generated XQuery code:

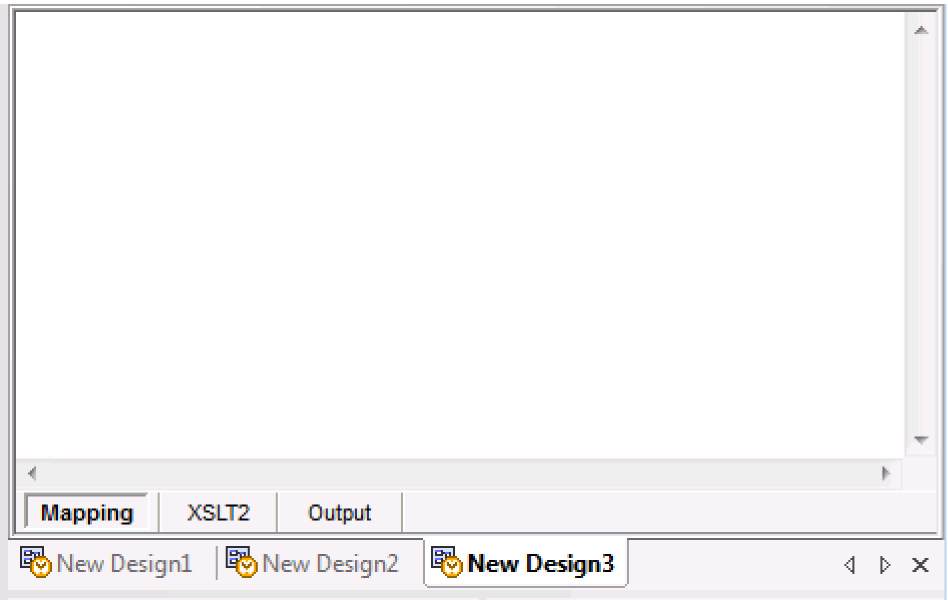
- Click the **XQuery** tab under the Mapping window.

Note: The XQuery tab becomes available if you have selected **XQuery** as transformation language.

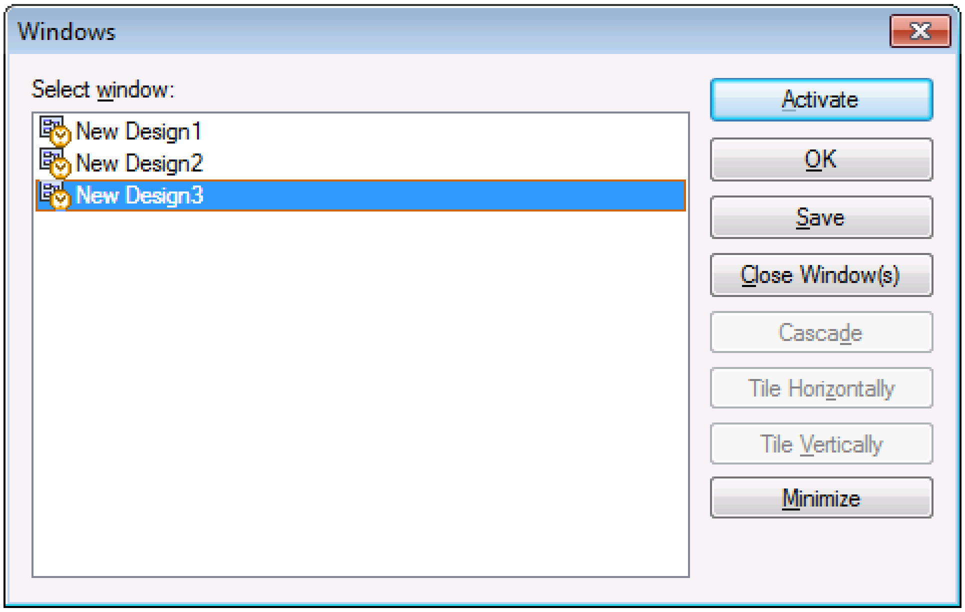
4.1.13 Working with Multiple Mapping Windows

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 in the lower part of the Mapping pane.



Window management options are available both on the **Window** menu and on the **Windows** dialog box. From the **Windows** dialog box, you can take actions against any or all currently open mapping windows (including saving, closing, or minimizing them).



Windows dialog box

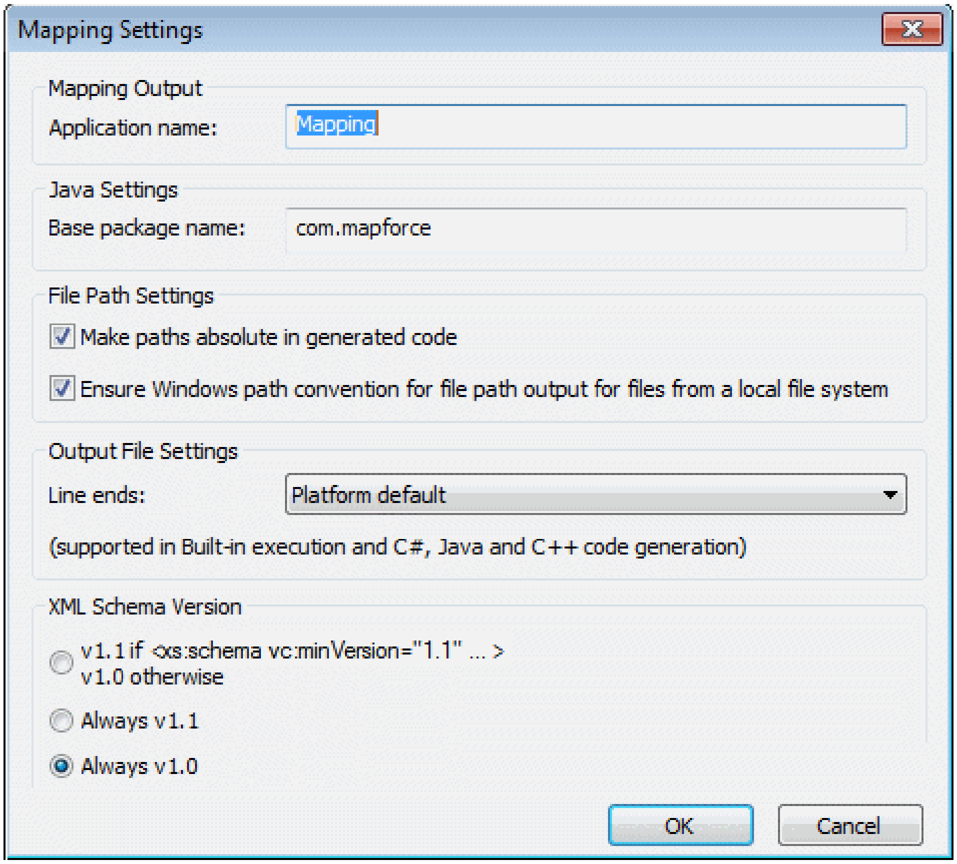
You can open the Windows dialog box using the menu command **Window | Windows...** To select multiple windows in the Windows dialog box, click the required entries while holding the **Ctrl** key pressed.

4.1.14 Changing the Mapping Settings

You can change the document-specific settings of the currently active mapping design file from the Mapping Settings dialog box. This information is stored in the *.mfd file.

To open the Mapping Settings dialog box:

- On the **File** menu, click **Mapping Settings**.



Application name: Mapping

Base package name: com.mapforce

☑ Make paths absolute in generated code

☑ Ensure Windows path convention for file path output for files from a local file system

Line ends: Platform default

(supported in Built-in execution and C#, Java and C++ code generation)

○ v1.1 if <xs:schema vc:minVersion="1.1" ... >
v1.0 otherwise

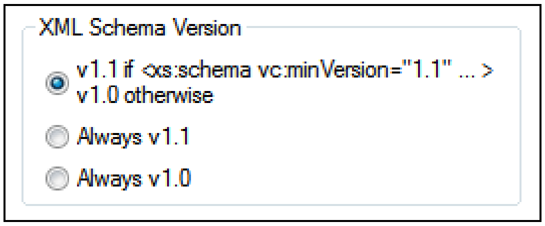
○ Always v1.1

● Always v1.0

Mapping Settings dialog box

The available settings are as follows.

Application Name	Defines the XSLT1.0/2.0 file name prefix or the Java, C# or C ++ application name for the generated transformation files.
Base Package Name	Defines the base package name for the Java output.
Make paths absolute in generated code	Defines whether the file paths should be relative or absolute in the generated program code, as well as in MapForce Server Execution files (mfx) and in mapping functions deployed to FlowForce Server. For more information, see About Paths in Generated Code.

Ensure Windows path convention for file path	The "Ensure Windows path convention...." check box makes sure that Windows path conventions are followed. When outputting XSLT2 (and XQuery), the currently processed file name is internally retrieved using the document-uri function, which returns a path in the form file:// URI for local files. When this check box is active, a file:// URI path specification is automatically converted to a complete Windows file path (e.g. "C:\...") to simplify further processing.
Line ends	This combo box allows you to specify the line endings of the output files. "Platform default" is the specific default for the target operating system, e.g. Windows (CR+LF), Mac OS X (LF), or Linux (LF). You can also select a specific line ending manually. The settings you select here are crucial when you deploy a mapping to FlowForce Server running on a different operating system.
XML Schema Version	Lets you define the XML Schema Version used in the mapping file. You can define if you always want to **load** the Schemas conforming to version 1.0 or 1.1. Note that not all version 1.1 specific features are currently supported. If the *xs:schema vc:minVersion="1.1"* declaration is present, then version 1.1 will be used; if not, version 1.0 will be used.  XML Schema Version ● v1.1 if <xs:schema vc:minVersion="1.1" ... > v1.0 otherwise ○ Always v1.1 ○ Always v1.0 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`, then XSD 1.0 will be the default mode. **Note:** Do not confuse the `vc:minVersion` attribute with the `xsd:version` attribute. The former holds the XSD version number, while the latter holds the document version number. Changing this setting in an existing mapping causes a reloading of all schemas of the selected XML schema version, and might also change its validity.

4.2 Working with Components

Components are the central elements of any mapping design in MapForce. Generally, the term "component" is a convenient way to call any object which acts as a data source, or as a data target, or represents your data in the mapping at an intermediary processing stage.

There are two main categories of components: structure components and transformation components.

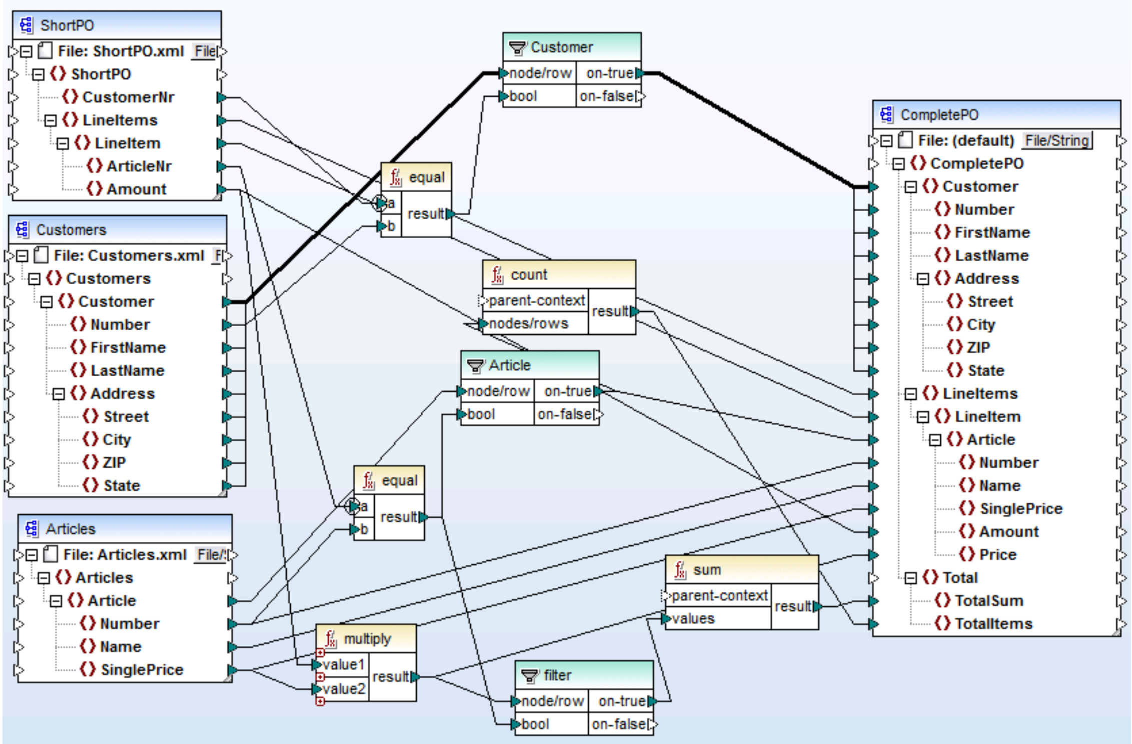
The structure components represent the abstract structure or schema of your data. For example, when you add an XML file to the mapping area (using the menu command **Insert | XML Schema/ File**), it becomes a mapping component. For further information about structure components and their specifics, see Data Sources and Targets. With a few exceptions, structure components consist of items and sequences. An item is the lowest level mapping unit (for example, a single attribute in the XML file, or an element of simple type). A sequence is a collection of items.

The transformation components either transform data (for example, functions), or assist you in transformations (for example, constants or variables). For information on how you can use these components to achieve various data transformation tasks, see Designing Mappings.

With the help of structure components, you can either read data from files or other sources, write data to files or other sources, or store data at some intermediary stage in the mapping process (for example, in order to preview it). Consequently, structure components can be of the following types:

- Source. You declare a component as source by placing it on the left of the mapping area, and, thus, instructing MapForce to read data from it.
- Target. You declare a component as target by placing on the right of the mapping area, and, thus, instructing MapForce to write data to it.
- Pass-through. This is a special component type which acts both as a source and target (for further information, see Chained mappings / pass-through components).

On the mapping area, components appear as rectangles. The following sample mapping illustrates three source components, one target XML component, and various transformation components (functions and filters) through which data goes before being written to the source.



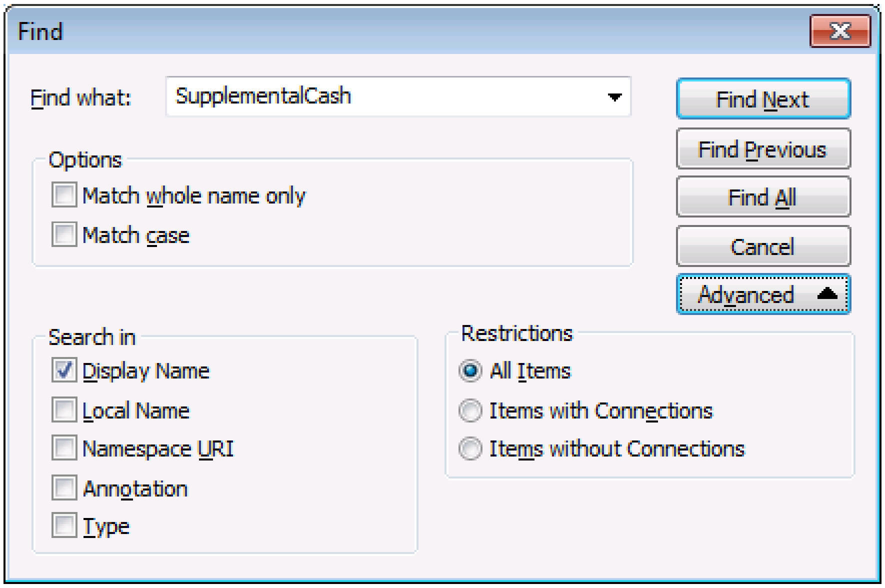
CompletePO.mfd

This mapping sample is available at the following path: **<Documents>\Altova\MapForce2018\MapForceExamples\CompletePO.mfd**.

4.2.1 Searching within Components

To search for a specific node/item in a component:

1. Click the component you want to search in, and press the CTRL+F keys.
2. Enter the search term and click **Find Next**.



Find what: SupplementalCash

Options
□ Match whole name only
□ Match case

Search in
☑ Display Name
□ Local Name
□ Namespace URI
□ Annotation
□ Type

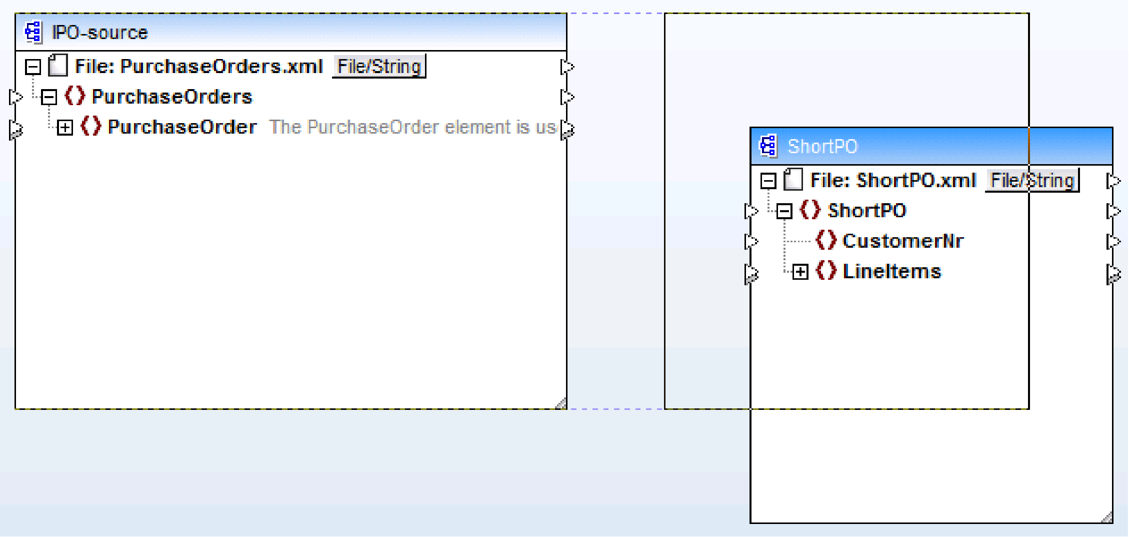
Restrictions
● All Items
○ Items with Connections
○ Items without Connections

Use the Advanced options to define which items (nodes) are to be searched, as well as restrict the search options based on the specific connections.

4.2.2 Aligning Components

When you move components in the mapping pane, MapForce displays auto-alignment guide lines. These guide lines help you align a component to any other component in the mapping window.

In the sample mapping below, the lower component is being moved. The guide lines show that it can be aligned to the component on the left side of the mapping.



Component auto-alignment guide lines

To enable or disable this option:

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

4.2.3 Changing the Component Settings

After you add a 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**.

Note that the available options depend on the type of the component. For reference to the settings applicable to each component type, see:

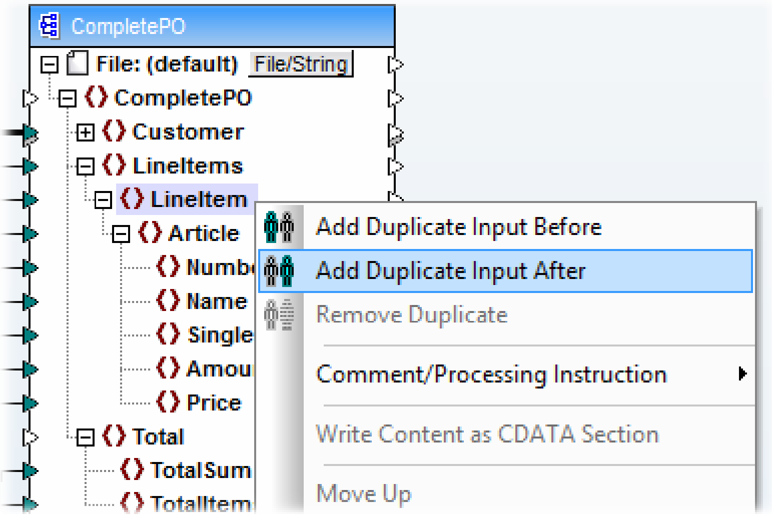
- XML Component Settings
- Database Component Settings
- CSV Component Settings
- Fixed-Length Field Component Settings

For any file-based component, such as XML, a **File/String** (File/String) button appears next to the root node. This button specifies advanced options applicable if you want to process or generate multiple files in a single mapping (see Processing Multiple Input or Output Files Dynamically). Additionally, it enables advanced options for parsing strings or serializing data to strings (see Parsing and Serializing Strings).

4.2.4 Duplicating Input

Sometimes, you may need to configure a component to accept data from more than one source. For example, you may need to convert data from two different XML schemas into a single schema. To make the destination schema accept data from both source schemas, you can duplicate any of the input items in the component. Duplicating input is meaningful only for a component which is a target component. On any given target component, you can duplicate as many items as required.

To duplicate a particular input item, right-click it and select **Add Duplicate Input After/Before** from the context menu.


CompletePO
File: (default) File/String
CompletePO
Customer
LineItems
LineItem
Article
Numb
Name
Single
Amou
Price
Total
TotalSum
TotalItem
Add Duplicate Input Before
Add Duplicate Input After
Remove Duplicate
Comment/Processing Instruction
Write Content as CDATA Section
Move Up

In the image above, the item `LineItem` is being duplicated in order to provide the ability to map data from a second source.

Once you duplicate an input, you can make connections both to the original input and to the duplicate input. For example, this would enable you to copy data from source A to original input, and data from source B to the duplicate input.

Note: Duplication of XML attributes is not allowed, as it would make the resulting XML instance invalid. In case of XML elements, duplicating input is allowed regardless of the value of the element's `maxOccurs` attribute in the schema. This behaviour is intentional, since the schema could change later, or the source data could be optional. For example, a mapping could generate a single XML element, even if the input is duplicated on the mapping.

For a step-by-step example, see Map Multiple Sources to One Target.

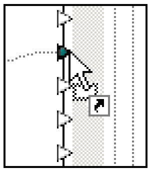
4.3 Working with Connections

A mapping is ultimately about transforming data from one format or structure into another. In a very basic mapping scenario, you add to the mapping area the components which represent your source and your target data (for example, a source XML schema and a destination one), and then draw visually the mapping connections between the two structure. A connection is, therefore, the visual representation of how data is mapped from a source to a destination.

Components have inputs and outputs which appear on the mapping as small triangles, called connectors. Input connectors are positioned to the left of any item to which you can draw a connection. Output connectors are positioned to the right of any item from which you can draw a connection.

To draw a connection between two items:

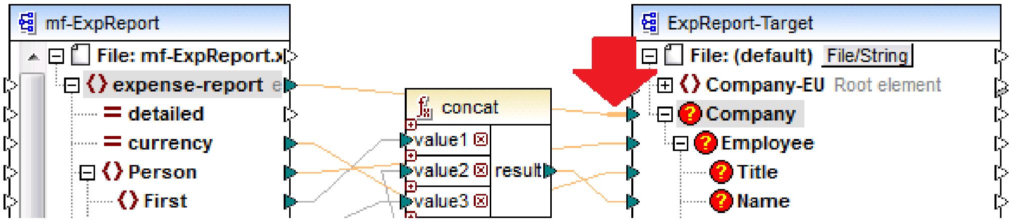
- Click the output connector of a source item and drag it to a destination item. When the drop action is allowed, a link tooltip appears next to the text cursor.



An input connector accepts only one incoming connection. If you try to add a second connection to the same input, a message box appears asking if you want to replace the connection with a new one or duplicate the input item. An output connector can have several connections, each to a different input.

To move a connection to a different item:

- Click the stub of the connection (the straight section closer to the target) and drag it to the destination.

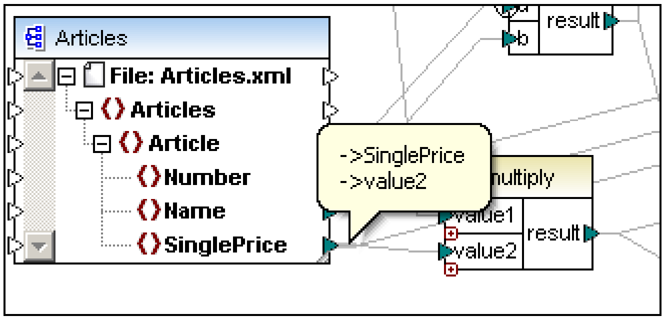


To copy a connection to a different item:

- Click the stub of the connection (the straight section closer to the target), and drag it to the destination while holding down the **Ctrl** key.

To view the item(s) at the other end of a connection:

- Point to the straight section of a connection (close to the input/output connector). A tooltip appears which displays the name(s) of the item(s) at the other end of the connection. If multiple connections have been defined from the same output, then a maximum of ten item names are displayed. In the sample below, the two target items are **SinglePrice** and **value2** of the multiply function.



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

- On the **Connection** menu, click **Properties** (this menu item becomes enabled when you select a connection).
- Double-click the connection.
- Right-click the connection, and then click **Properties**.

See also Connection Settings.

To delete a connection, do one of the following:

- Click the connection, and then press the **Delete** key.
- Right-click the connection, and then click **Delete**.

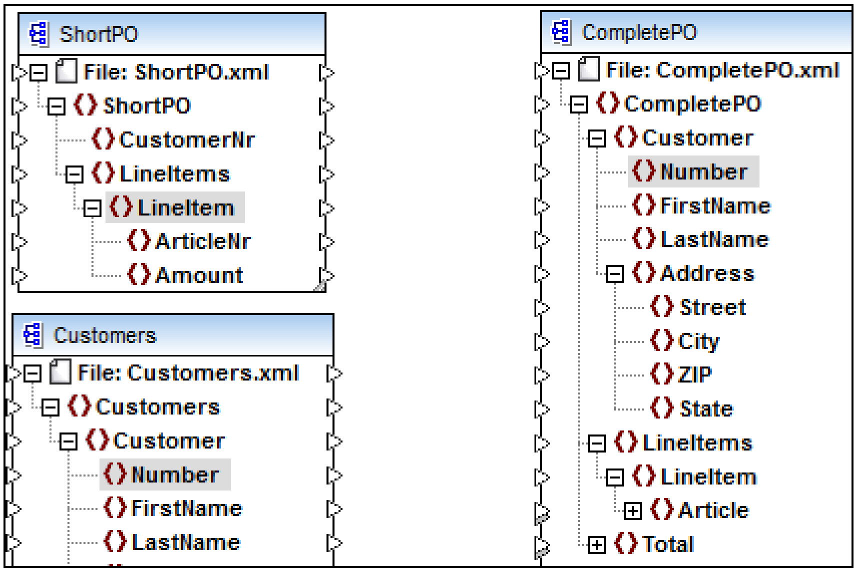
4.3.1 About Mandatory Inputs

To aid you in the mapping process, MapForce highlights in orange the mandatory inputs in target components:

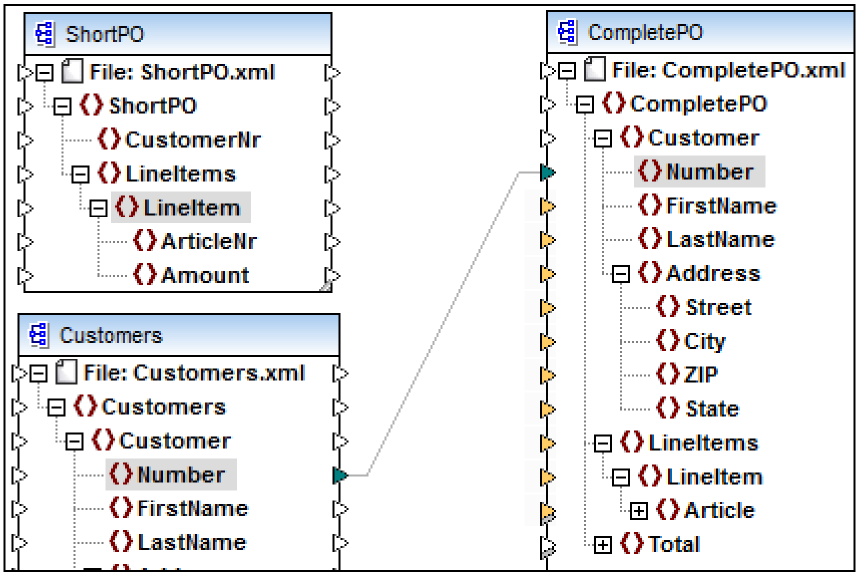
- In XML and EDI components these are items where the minOccurs parameter is equal/ greater than 1.
- In databases these are fields that have been defined as "not null"
- WSDL calls and WSDL response (all nodes)
- XBRL nodes that have been defined as mandatory
- In functions these are the specific mandatory parameters such that once one parameter has been mapped, then the other mandatory ones will be highlighted to show that a connection is needed. E.g. once one of the filter input parameters is mapped, then the other one is automatically highlighted.
- Worksheet names in MS Excel sheets

Example:
When creating a mapping like CompletePO.mfd, available in the ...\MapForceExamples folder, the

inserted XML Schema files exist as shown below.



The Number element of the Customers component is then connected to the Number element of the CompletePO component. As soon as the connection has been made, the mandatory items/ nodes of the CompletePO component are highlighted. Note that the collapsed "Article" node/icon is also highlighted.



4.3.2 Changing the Connection Display Preferences

You can selectively view the connections in the mapping window.

Show selected component connectors switches between showing:

- all mapping connectors in black, or
- those connectors relating to the currently selected component in black. Other connectors appear dimmed.

Show connectors from source to target switches between showing:

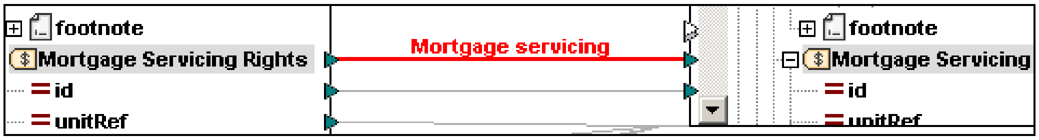
- connectors that are **directly** connected to the currently selected component, or
- connectors linked to the currently selected component, originating from source and terminating at the target components.

4.3.3 Annotating Connections

Individual connections can be labeled allowing you to comment your mapping in great detail. This option is available for **all connection types**.

To annotate to a connection:

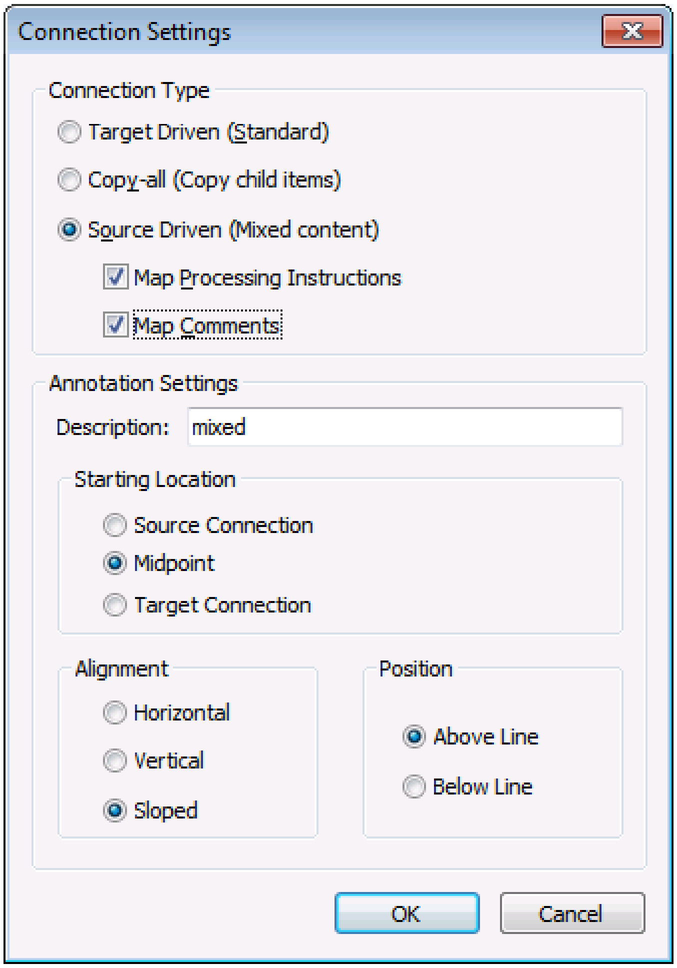
1. Right-click the connection, and select **Properties** from the context menu.
2. Enter the name of the currently selected connection in the **Description** field. This enables all the options in the Annotation Settings group.
2. Use the remaining groups to define the **starting location**, **alignment** and **position** of the label.
3. Activate the **Show annotations** icon in the View Options toolbar to see the annotation text.



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

4.3.4 Connection Settings

Right-clicking a connection and selecting **Properties** from the context menu, or double-clicking a connection, opens the Connection Settings dialog box in which you can define the settings of the current connection. Note that unavailable options are disabled.



Connection Type

○ Target Driven (Standard)

○ Copy-all (Copy child items)

● Source Driven (Mixed content)

☑ Map Processing Instructions

☑ Map Comments

Annotation Settings

Description: mixed

Starting Location

○ Source Connection

● Midpoint

○ Target Connection

Alignment

○ Horizontal

○ Vertical

● Sloped

Position

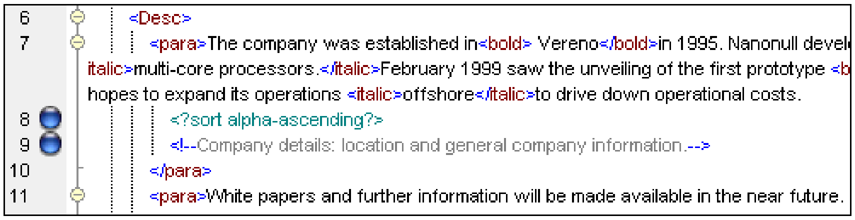
● Above Line

○ Below Line

Connection Settings dialog box

For items of **complexType**, you can choose one of the following connection types for mapping (note that these settings also apply to **complexType** items which do not have any text nodes):

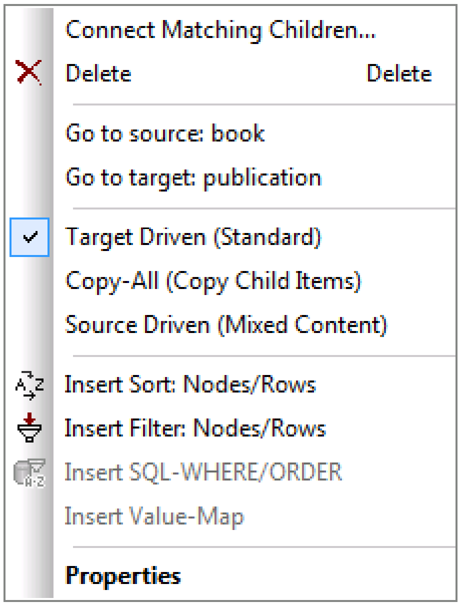
Target Driven (Standard)	Changes the connection type to "Target-driven" (see Target-driven / Standard mapping).
Copy-all (Copy child items)	Changes the connection type to "Copy-all" and automatically connects all identical items in the source and target components (see Copy-all connections).
Source Driven (mixed content)	Changes the connection type to "Source-driven", and enables the selection of additional elements to be mapped. The additional elements must be child items of the mapped item in the XML source file, to qualify for mapping. Activating the **Map Processing Instructions** or **Map Comments** check boxes enables you to include these data groups in the output file.

	 <Desc> <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull devel italic>multi-core processors.</italic>February 1999 saw the unveiling of the first prototype <b 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. Note: CDATA sections are treated as text.

The Annotation Settings group enables you to annotate the connection (see Annotating Connections).

4.3.5 Connection Context Menu

When you right-click a connection, the following context commands are available.


Connect Matching Children...
Delete
Delete
Go to source: book
Go to target: publication
Target Driven (Standard)
Copy-All (Copy Child Items)
Source Driven (Mixed Content)
Insert Sort: Nodes/Rows
Insert Filter: Nodes/Rows
Insert SQL-WHERE/ORDER
Insert Value-Map
Properties

Connect matching children	Opens the "Connect Matching Children" dialog box (see Connecting Matching Children). This command is enabled when the connection is eligible to have matching children.
Delete	Deletes the selected connection.
Go to source: <item name>	Selects the source connector of the current connection.
Go to target: <item name>	Selects the target connector of the current connection.
Target Driven (Standard)	Changes the connection type to "Target-driven" (see Target-driven connections).

Copy-All (Copy Child Items)	Changes the connection type to "Copy-all" and automatically connects all identical items in the source and target components (see Copy-all connections). This command is enabled (and meaningful) when both the source item and the target item have children items.
Source Driven (Mixed Content)	Changes the connection type to "Source-driven" (see Source-driven connections). This command is enabled (and meaningful) when both the source item and the target item have children items.
Insert Sort: Nodes/Rows	Adds a Sort component between the source and the target item (see Sorting Data).
Insert Filter: Nodes/Rows	Adds a Filter component between the source and the target item (see Filters and Conditions).
Insert SQL-Where Condition	Adds a SQL-Where component between the source and the target item (see SQL WHERE / ORDER Component).
Insert Value-Map	Adds a Value-Map component between the source and the target item (see Using Value-Maps).
Properties	Opens the Connections Settings dialog box (see Connection Settings).

4.3.6 Connecting Matching Children

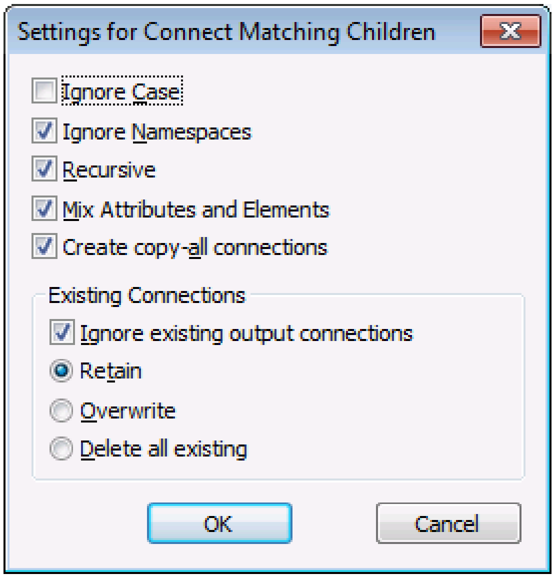
You can create multiple connections between items of the **same name** in both the source and target components. Note that a "Copy-all" connection (see Copy-all connections) is created by default.

To toggle the "Auto Connect Matching Children" option on or off, do one of the following:

- Click the **Auto Connect Matching Children** (☰) toolbar button.
- On the **Connection** menu, click **Auto Connect Matching Children**.

To change the settings for "Connect Matching Children":

1. Connect two (parent) items that share identically named **child items** in both components.
2. Right click the connection and select the **Connect matching child elements** option.



Settings for Connect Matching Children

☐ Ignore Case

☑ Ignore Namespaces

☑ Recursive

☑ Mix Attributes and Elements

☑ Create copy-all connections

Existing Connections

☑ Ignore existing output connections

● Retain

○ Overwrite

○ Delete all existing

3. Select the required options (see the table below), and click OK. Connections are created for all the child items that have identical names and adhere to the settings defined in the dialog box.

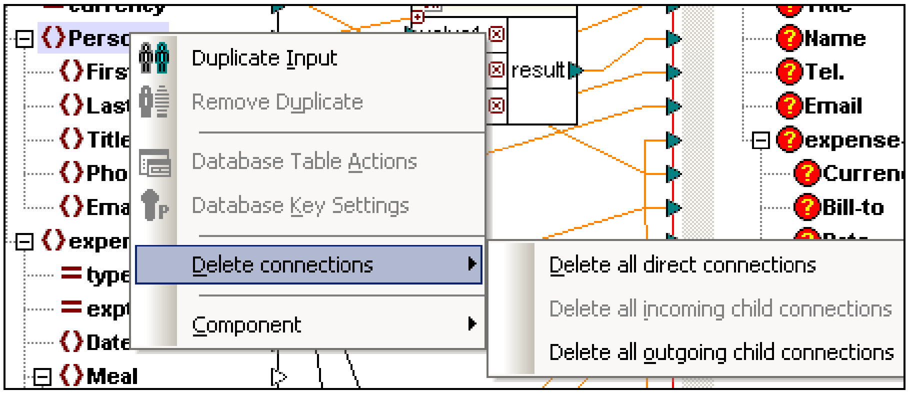
Note: The settings you define here are applied when connecting two items if the **Toggle auto connect of children** (☰) toolbar button is active.

Ignore Case	Ignores the case of the child item names.
Ignore Namespaces	Ignores the namespaces of the child items.
Recursive	Creates new connections between any matching items recursively. That is, a connection is created no matter how deep the items are nested in the hierarchy, as long as they have the same name.
Mix Attributes and Elements	When enabled, allows connections to be created between attributes and elements which have the same name. For example, a connection is created if two "Name" items 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 connection of type "Copy-all" between source and target items.
Ignore existing output connections	Creates additional connections for any matching items, even if they already have outgoing connections.
Retain	Retains existing connections.
Overwrite	Recreates connections according to the settings defined. Existing connections are discarded.

Delete all existing	Deletes all existing connections, before creating new ones.

Deleting connections

Connections that have been created using the Connect Matching Children dialog, or during the mapping process, can be removed as a group.



To delete connections:

1. Right-click the item name in the component, not the connection itself ("Person" in this example).
2. Select **Delete Connections | Delete all ... connections**.

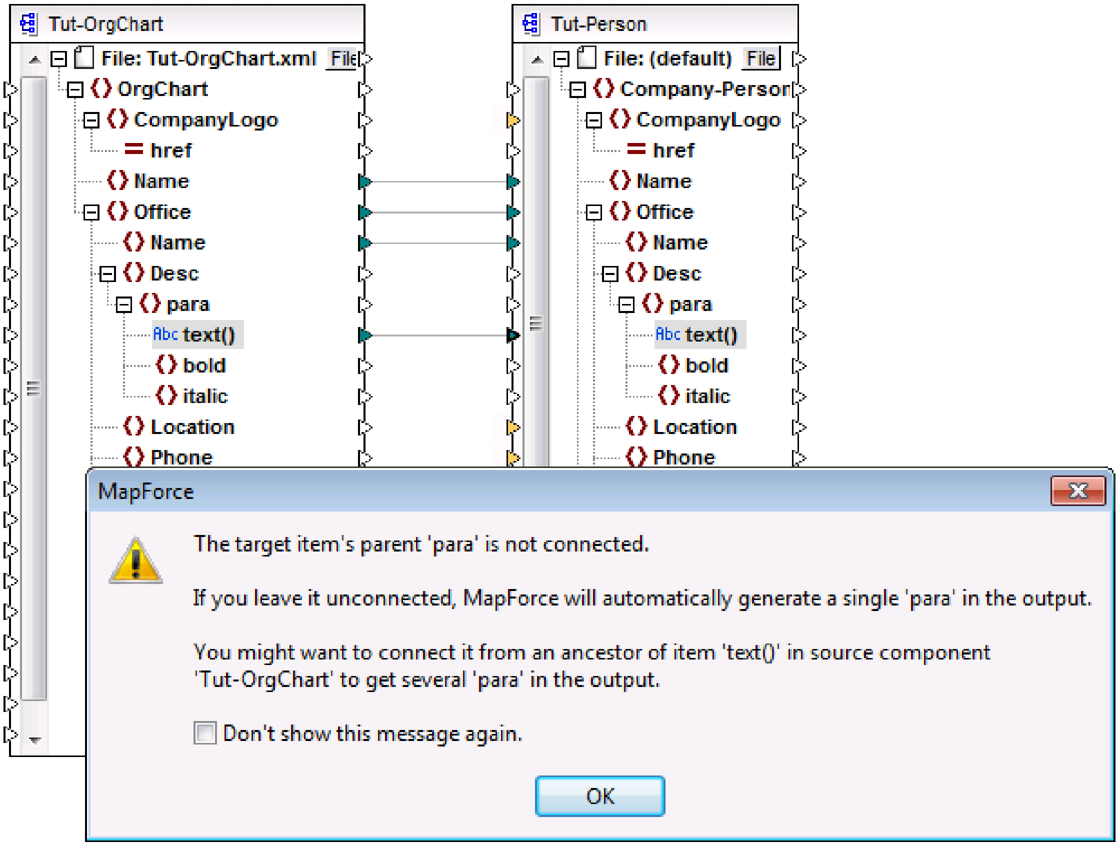
Delete all direct connections	Deletes all connections directly mapped to, or from, the current component to any other source or target components.
Delete all incoming child connections	Only active if you have right clicked an item in a target component. Deletes all incoming child connections.
Delete all outgoing child connections	Only active if you have right clicked an item in a source component. Deletes all outgoing child connections.

4.3.7 Notifications on Missing Parent Connections

When you create connections between source and target items manually, MapForce automatically analyzes the possible mapping outcomes. If you are mapping two child items, a notification message can appear suggesting that you also connect the parent of the source item with the parent in the target item.

This notification message helps you prevent situations where a single child item appears in the Output window when you preview the mapping. This will generally be the case if the source node supplies a sequence instead of a single value.

To understand how this works, open the sample mapping **Tut-OrgChart.mfd** available in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder. If you connect the source `text()` item to the target `text()` item, a notification message appears, stating that the parent item "para" is not connected and will only be generated once in the output.



The target item's parent 'para' is not connected.

If you leave it unconnected, MapForce will automatically generate a single 'para' in the output.

You might want to connect it from an ancestor of item 'text()' in source component 'Tut-OrgChart' to get several 'para' in the output.

□ Don't show this message again.

Tut-OrgChart.mfd (MapForce Basic Edition)

To generate multiple `para` items in the target, connect the source and target `para` items to each other.

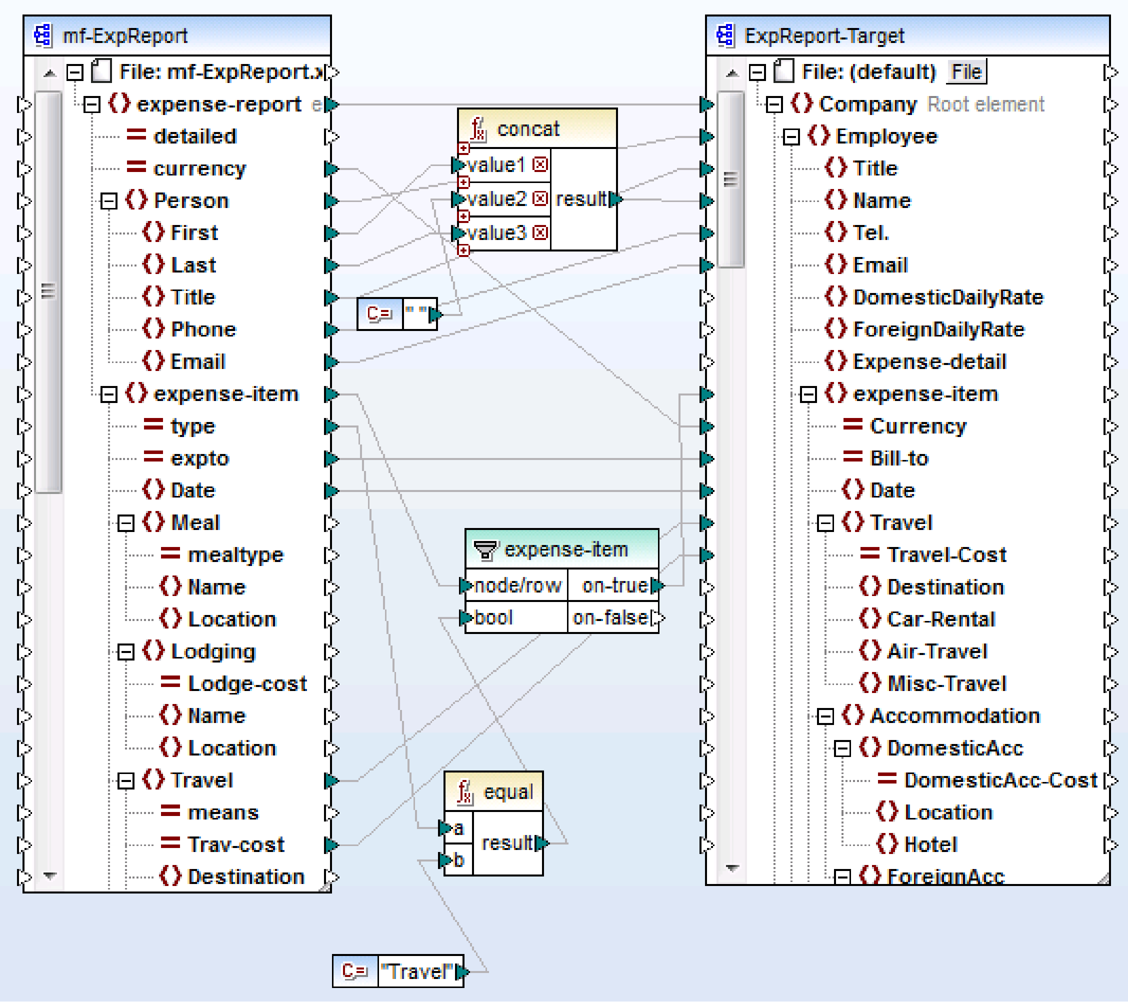
To disable such notifications, do the following:

1. On the **Tools** menu, click **Options**.
2. Click the **Messages** group.
3. Click to clear the **When creating a connection, suggest connecting ancestor items** check box.

4.3.8 Moving Connections and Child Connections

When you move a connection to a different component, MapForce automatically matches identical child connections and will prompt you whether it should move them to the new location as well. A common use of this feature is if you have an existing mapping and then change the root element of the target schema. Normally, when this happens, you would need to remap all descending connections manually. This feature helps you prevent such situations.

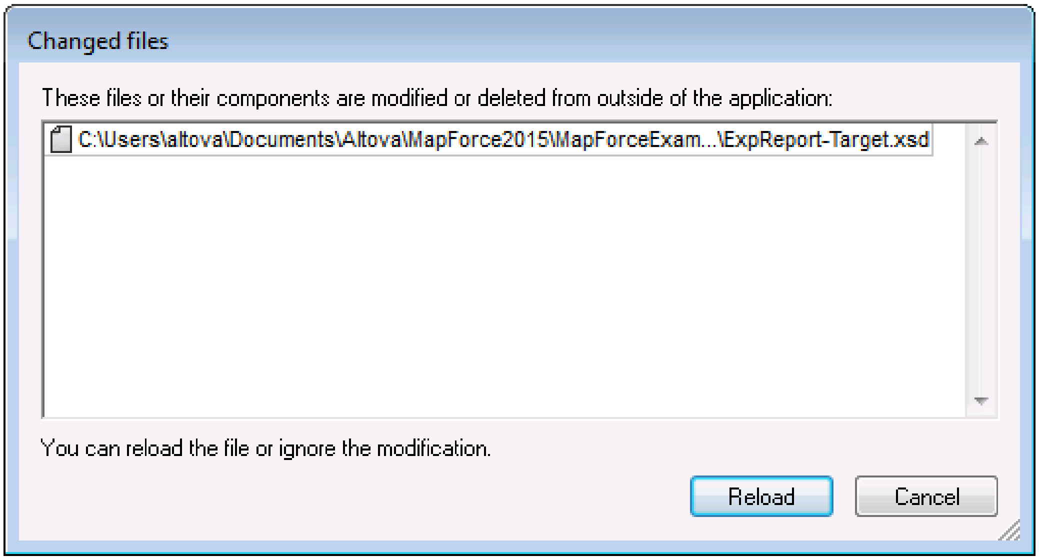
This example uses the **Tut-ExpReport.mfd** file available in the **<Documents>\Altova \MapForce2018\MapForceExamples\Tutorial\** folder.



Tut-ExpReport.mfd (MapForce Basic Edition)

To understand how it works, do the following:

1. Open the **Tut-ExpReport.mfd** sample mapping.
2. Edit the **ExpReport-Target.xsd** schema outside MapForce so as to change the `Company` root element of the target schema to `Company-EU`. You do not need to close MapForce.
3. After you have changed the `Company` root element of the target schema to `Company-EU`, a "Changed files" prompt appears in MapForce.

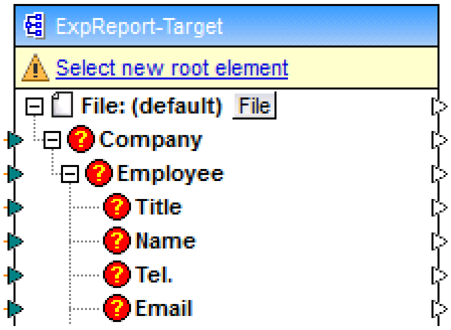

Changed files

These files or their components are modified or deleted from outside of the application:

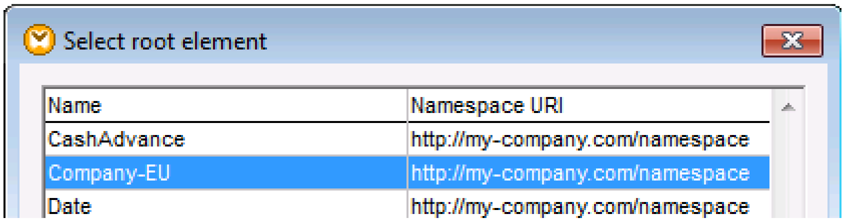
C:\Users\altova\Documents\Altova\MapForce2015\MapForceExam...\ExpReport-Target.xsd

You can reload the file or ignore the modification.

4. Click the **Reload** button to reload the updated Schema. Since the root element was deleted, the component displays multiple missing nodes.

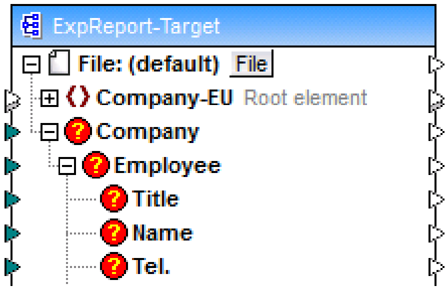


5. Click **Select new root element** at the top of the component. (You can also change the root element by right clicking the component header and selecting **Change Root Element** from the context menu.)

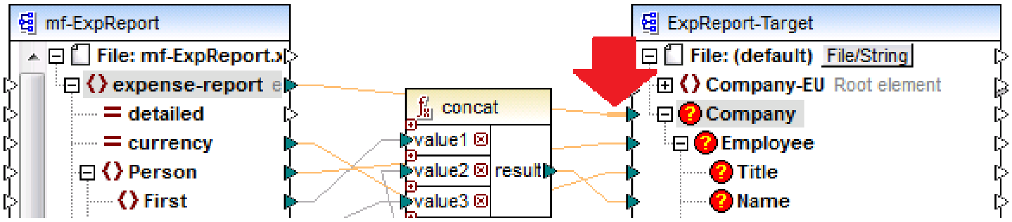

Select root element

Name	Namespace URI
CashAdvance	http://my-company.com/namespace
Company-EU	http://my-company.com/namespace
Date	http://my-company.com/namespace

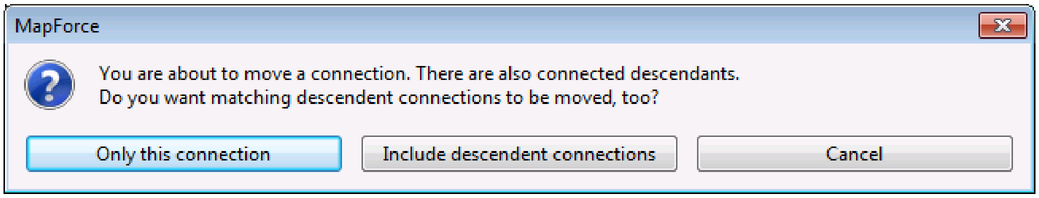
6. Select `Company-EU` as new root element and click OK to confirm. The `Company-EU` root element is now visible at the top of the component.



7. Click the target stub of the connection that exists between the `expense-report` item of the source component and the `Company` item of the target component, and then drag-and-drop it on the `Company-EU` root element of the target component.



A notification dialog box appears.



You are about to move a connection. There are also connected descendants.
Do you want matching descendent connections to be moved, too?

8. Click **Include descendent connections**. This instructs MapForce to re-map the correct child items under the new root element, and the mapping becomes valid again.

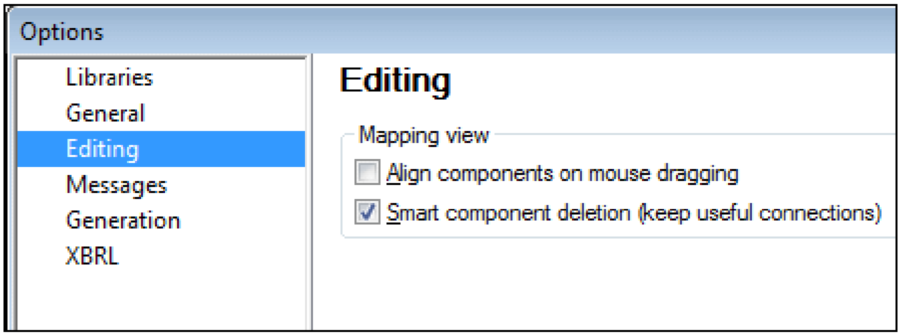
Note: If the node to which you are mapping has the same name as the source node but is in a different namespace, then the notification dialog box will contain an additional button: "Include descendants and map namespace". Clicking this button moves the child connections of the same namespace as the source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

4.3.9 Keeping Connections After Deleting Components

You can decide what happens when you delete a component that has multiple (child) connections to another component, e.g. a filter or sort component. This is very useful if you want to keep all the child connections and not have to restore each one individually.

You can opt to keep/restore the child connections after the component is deleted, or to delete all child connections immediately.

Select **Tools | Options | Editing** (tab) to see the current setting. The default setting for the check box is **inactive**, i.e. "Smart component deletion (keep useful connections)" is disabled.



Options

- Libraries
- General
- Editing
- Messages
- Generation
- XBRL

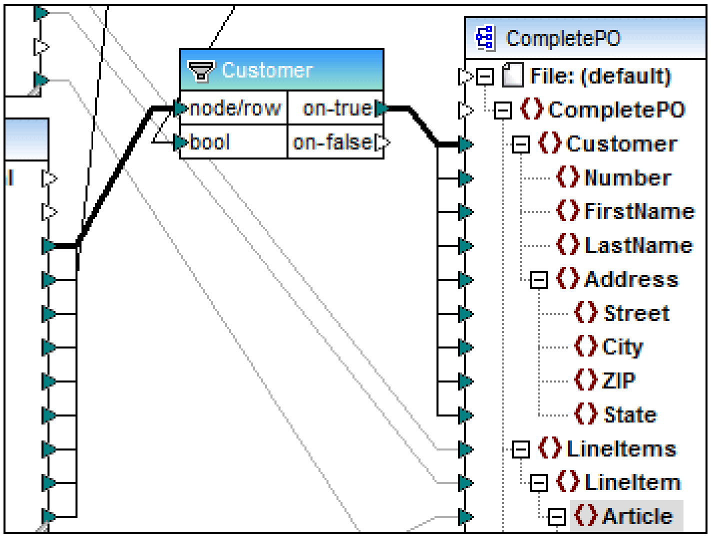
Editing

Mapping view

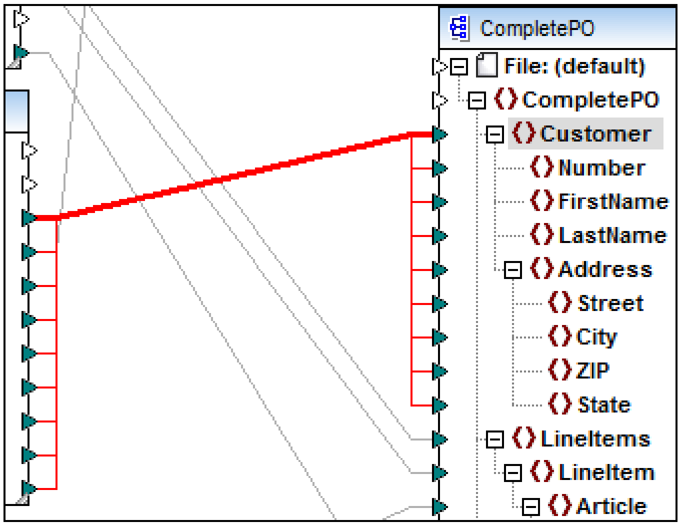
☐ Align components on mouse dragging

☑ Smart component deletion (keep useful connections)

E.g. using the CompletePO.mfd mapping in the ...\MapForceExamples folder, and the check box is active, the Customer filter is a **copy-all** connection with many connected child items, as shown below.



Deleting the Customer filter opens a prompt asking if you really want to delete it. If you select Yes, then the filter is deleted but all the child connectors remain.



Note that the remaining connectors are still selected (i.e. shown in red). If you want to delete them as well, hit the Del. key.

Clicking anywhere in the mapping area deselects the connectors.

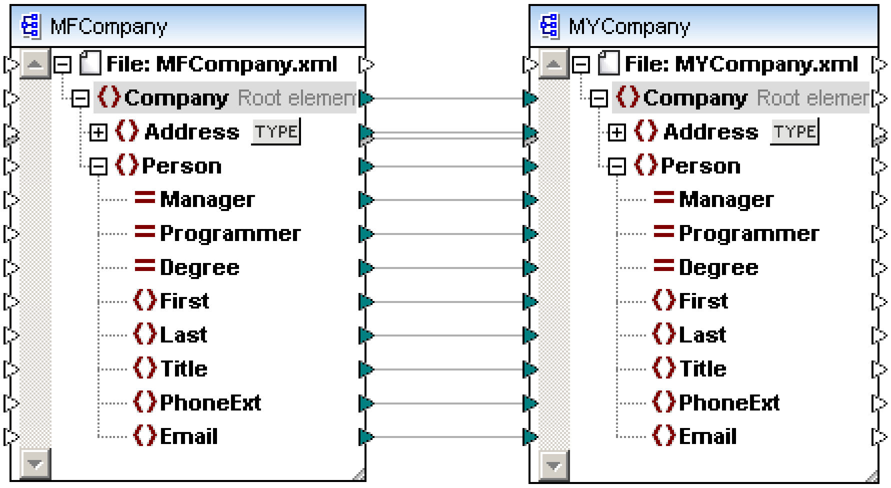
If the "Smart component deletion..." check box is **inactive**, then deleting the filter will delete all child connectors immediately.

Note: If a filter component has both "on-true" and "on-false" outputs connected, then the connectors for both outputs will be retained.

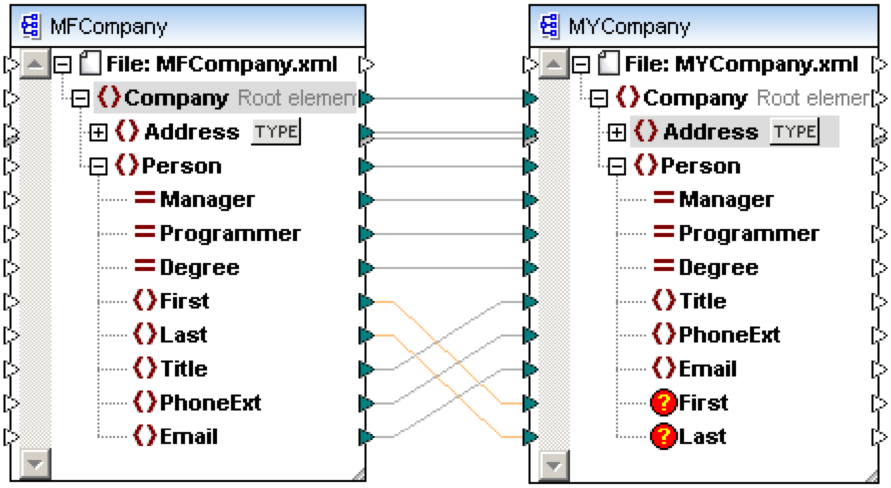
4.3.10 Dealing with Missing Items

Over time, it is likely that the structure of one of the components in a mapping may change e.g. elements or attributes are added/deleted to an XML schema. MapForce uses placeholder items to retain all the connectors, and any relevant connection data between components, when items have been deleted.

Example:
Using the **MFCompany.xsd** schema file as an example. The schema is renamed to MyCompany.xsd and a connector is created between the Company item in both schemas. This creates connectors for all child items between the components, if the Autoconnect Matching Children is active.



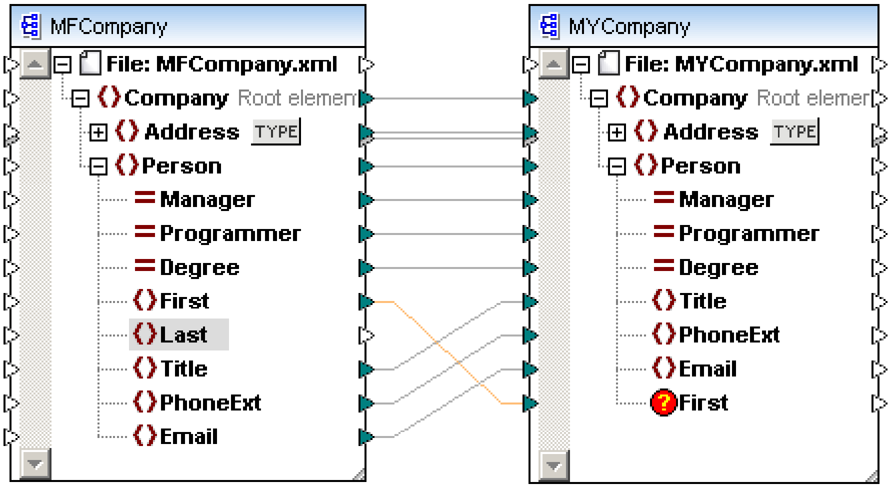
While editing MyCompany.xsd, in XMLSpy, the First and Last items in the schema are deleted. Returning to MapForce opens a Changed Files notification dialog box, prompting you to reload the schema. Clicking **Reload** updates the components in MapForce.



The deleted **items** and their **connectors** are now marked in the MyCompany component. You could now reconnect the connectors to other items if necessary, or delete the connectors.

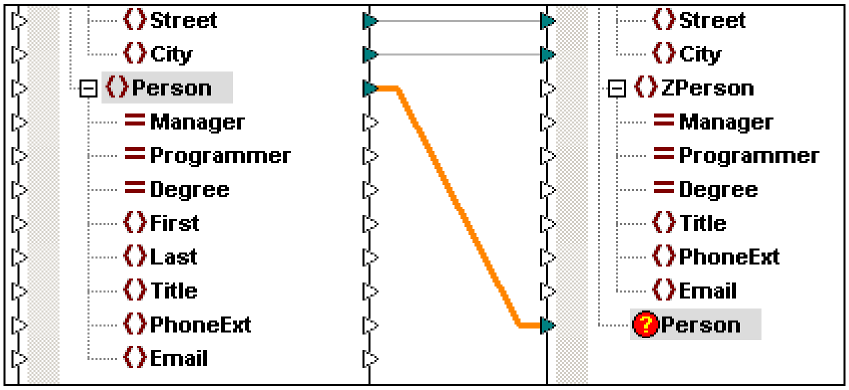
Note that you can still preview the mapping (or generate code), but warnings will appear in the Messages window if you do so at this point. All connections to, and from, missing items are ignored during preview or code-generation.

Clicking one of the highlighted connectors and deleting it, removes the "missing" item from the component, e.g. Last, in MyCompany.



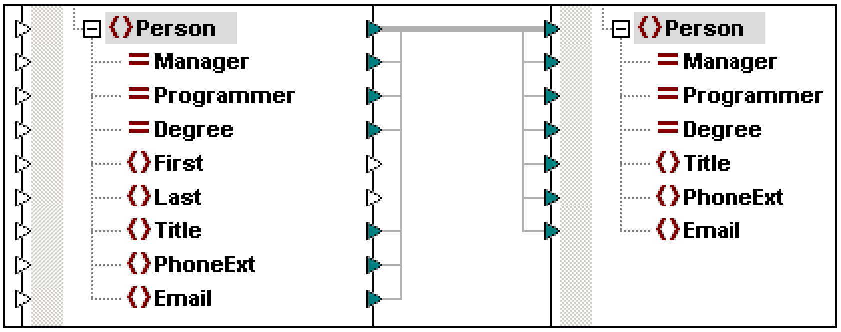
Renamed items

If a parent item is renamed e.g. Person to ZPerson, then the original parent item connector is retained and the child items and their connectors are deleted.



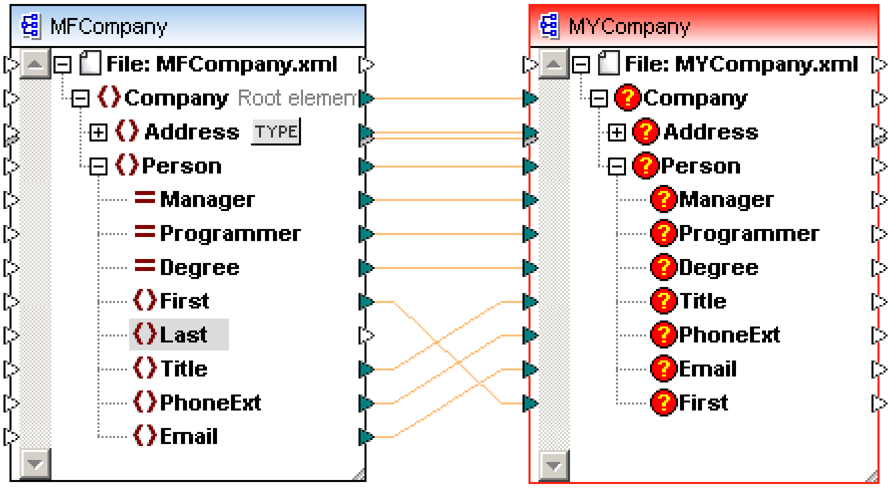
"Copy all" connectors and missing items

Copy all connections are treated in the same way as normal connections, with the only difference being that the connectors to the missing child items are not retained or displayed.

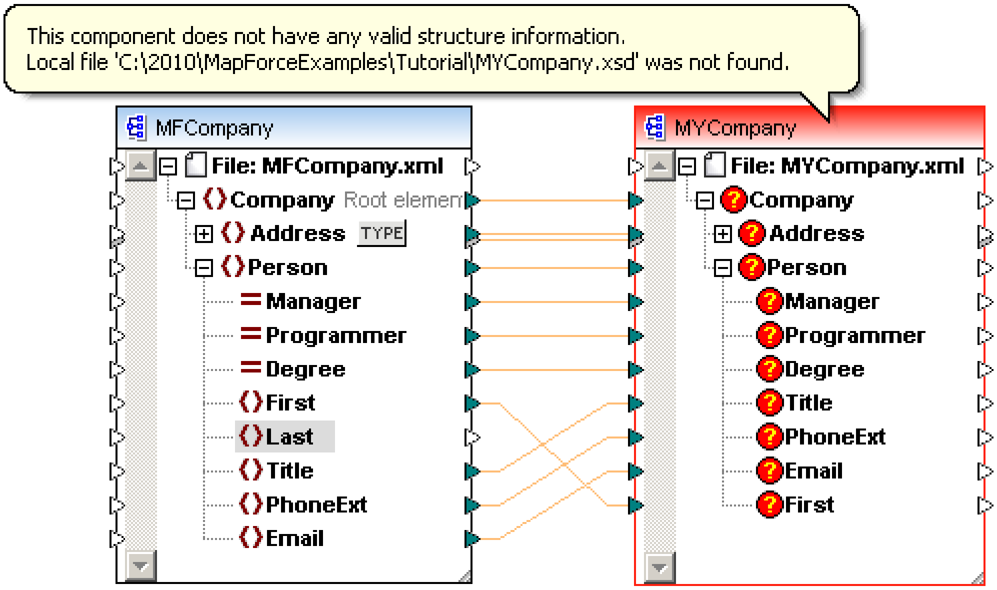


Renamed or deleted component sources

If the **data source** of a component i.e. schema, database etc. has been renamed or deleted, then all items it contained are highlighted. The red frame around the component denotes that there is no valid connection to a schema or database file and prevents preview and code generation.

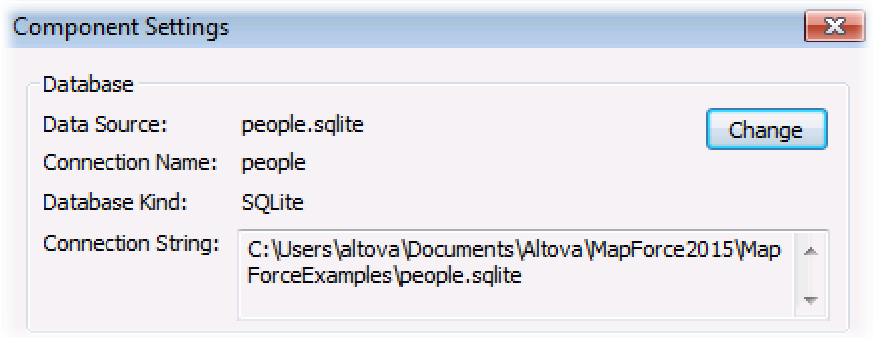


Placing the mouse cursor over the highlighted component, opens a popup containing pertinent information.


This component does not have any valid structure information.
Local file 'C:\2010\MapForceExamples\Tutorial\MYCompany.xsd' was not found.

Double-clicking the title bar of the highlighted component opens the Component Settings dialog box. Clicking the **Browse** button in the **Schema file** group allows you to select a different, or backed-up version of the schema. Please see "Component" in the Reference section for more information.

Clicking the **Change** button in the dialog box that opens if the component is a database, allows you to select a different database, or change the tables that appear in the database component. Connectors to tables of the same name will be retained.



Data Source: people.sqlite

Connection Name: people

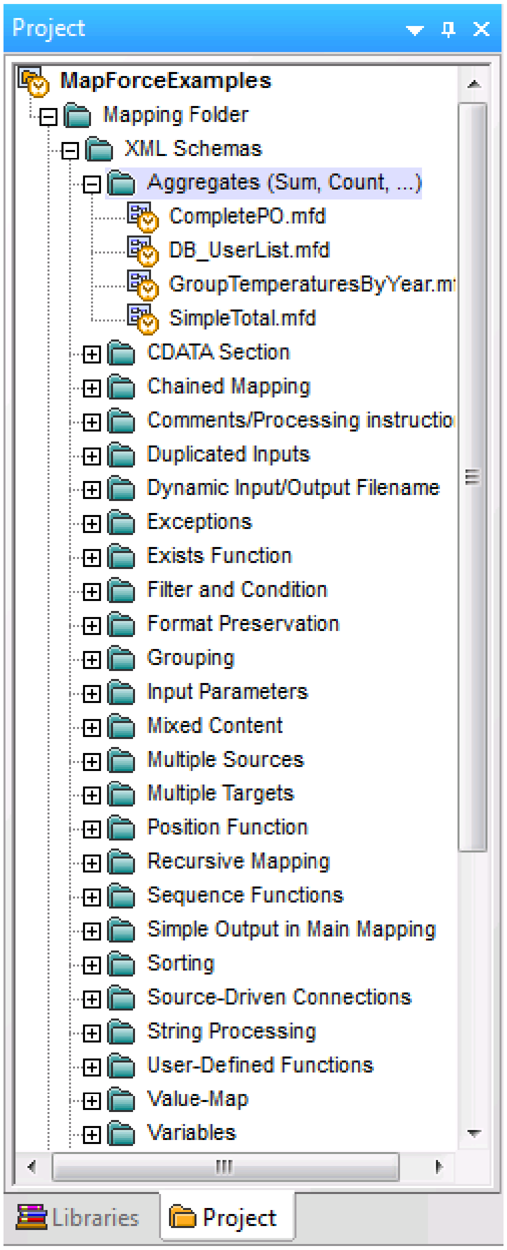
Database Kind: SQLite

Connection String: C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\people.sqlite

All valid/correct connections (and relevant database data, if the component is a database) will be retained if you select a schema or database of the same structure.

4.4 Working with Mapping Projects

In addition to creating standalone mappings, you can also create mapping projects that include multiple mappings. Mappings added to a project are easily accessible from the Project window.



- MapForceExamples
  - Mapping Folder
    - XML Schemas
      - Aggregates (Sum, Count, ...)
        - CompletePO.mfd
        - DB_UserList.mfd
        - GroupTemperaturesByYear.m
        - SimpleTotal.mfd
      - CDATA Section
      - Chained Mapping
      - Comments/Processing instructio
      - Duplicated Inputs
      - Dynamic Input/Output Filename
      - Exceptions
      - Exists Function
      - Filter and Condition
      - Format Preservation
      - Grouping
      - Input Parameters
      - Mixed Content
      - Multiple Sources
      - Multiple Targets
      - Position Function
      - Recursive Mapping
      - Sequence Functions
      - Simple Output in Main Mapping
      - Sorting
      - Source-Driven Connections
      - String Processing
      - User-Defined Functions
      - Value-Map
      - Variables

Project window (MapForce Enterprise Edition)

The main advantage of projects is that you can define common code generation settings (such as the target language and the output directory) for all the mapping files 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 the MapForce-generated program code (in C++, C#, and Java), see Code Generator.

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 Web Services Description Language (WSDL) files.

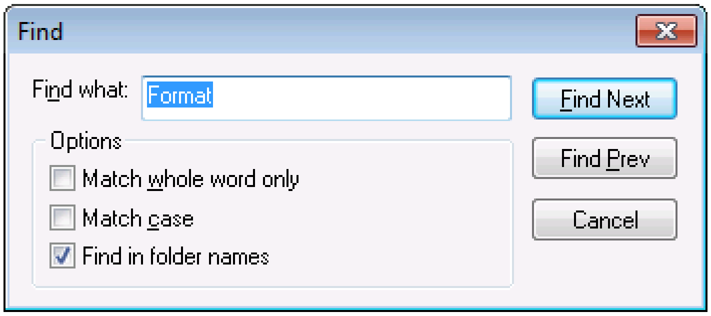
4.4.1 Opening, Searching, and Closing Projects

MapForce project files have the *.mfp extension. You can open existing MapForce projects in the same way as you open mappings (on the **File** menu, click **Open**).

When a mapping project is opened in MapForce, the Project window shows all files and folders that have been added to the project. By default, when you run MapForce for the first time, it loads the **MapForceExamples.mfp** project in the Project window.

To search for files within a project:

1. In the Project window, click the project or the folder to be searched.
2. Press **Ctrl + F**.
3. Optionally, select your search options. For example, if you want to include folder names in the search, select the **Find in folder names** option.



Find

Find what: Format

Options

☐ Match whole word only

☐ Match case

☑ Find in folder names

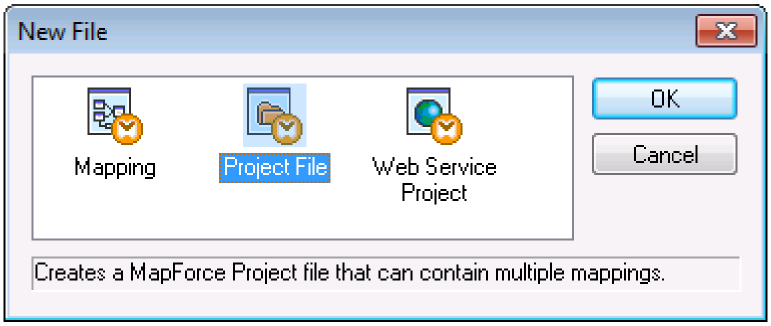
To close a project:

- On the **Project** menu, click **Close Project**.

4.4.2 Creating a New Project

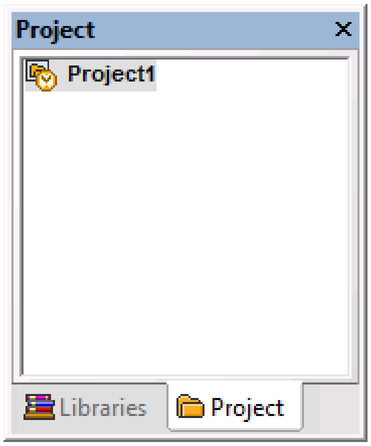
To create a new project:

1. On the **File** menu, click **New**.
2. Select **Project File**, and then click **OK**.



Creates a MapForce Project file that can contain multiple mappings.

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.



You can now add mappings to the project.

To add the currently active mapping to the project, do one of the following:

- On the **Project** menu, click **Add Active File to Project** .
- Right-click the project, and select **Add Active File to Project** .

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

- On the **Project** menu, click **Add Files to Project** .
- Right-click the project, and select **Add Files to Project** .

Tip: To open multiple files, hold the **Ctrl** key while selecting the files in the Open dialog box.

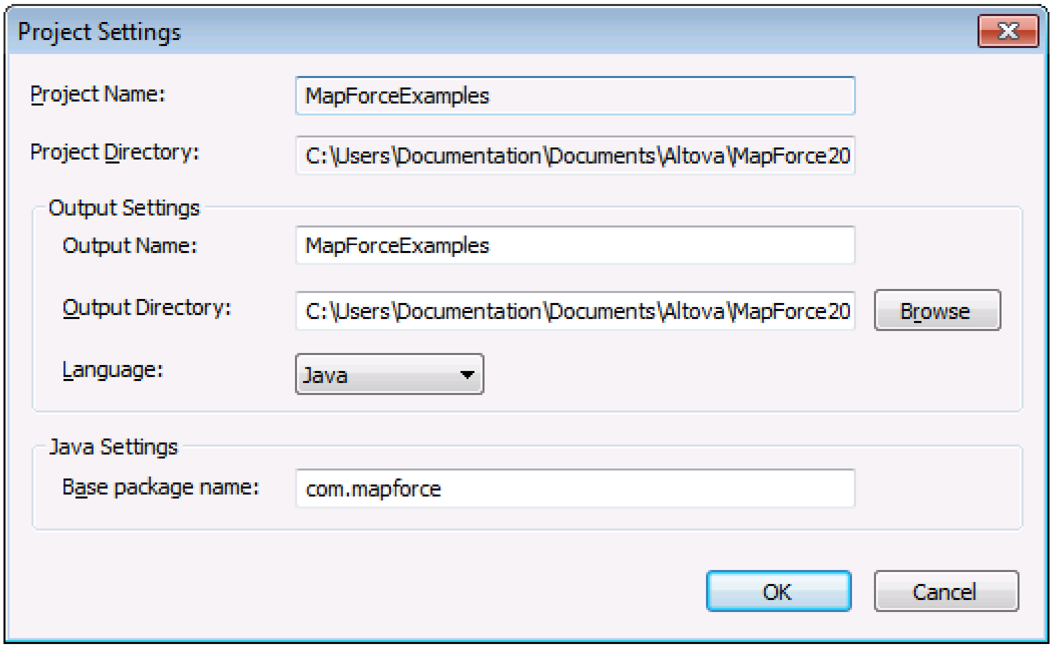
To remove a file or folder from a project, do one of the following:

- Right-click the file in the Project window, and select **Delete** from the context menu.
- Select the file in the Project window, and press **Delete**.

4.4.3 Setting the Code Generation Settings

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

- Right-click the project name in the **Project** window and choose **Properties** from the context menu
- On the **Project** menu, click **Properties**.



Project Name: MapForceExamples

Project Directory: C:\Users\Documentation\Documents\Altova\MapForce20

Output Name: MapForceExamples

Output Directory: C:\Users\Documentation\Documents\Altova\MapForce20

Language: Java

Base package name: com.mapforce

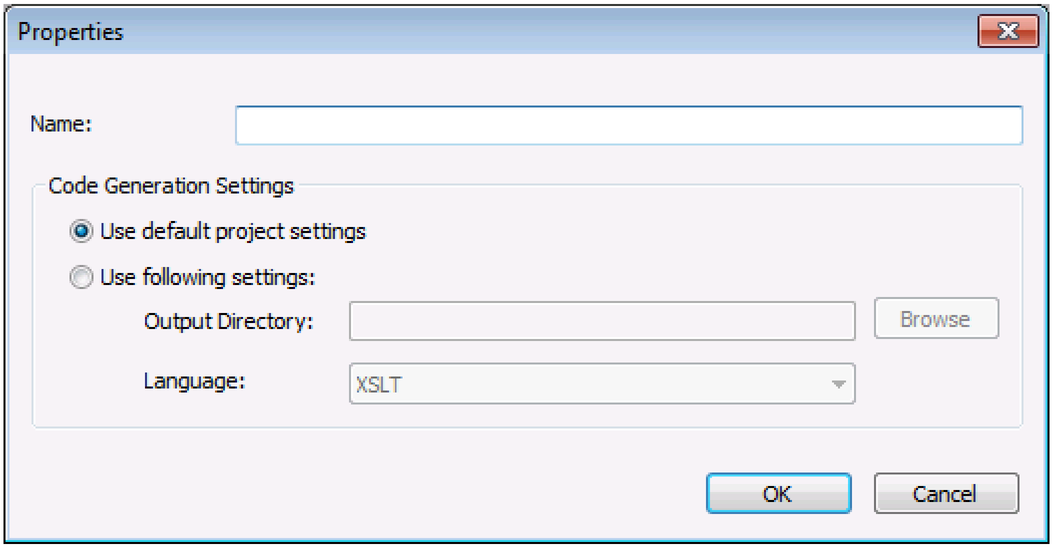
Project Settings dialog box

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

Output name	The value entered here determines the name of the generated project or solution, as well as other objects names 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, output is saved to the **output/** directory located in the project directory.
Language	Defines the code generation language for all mapping files in this project.
Base package name	This setting is applicable if you selected Java as transformation language. It defines the name of the base package in the generated Java project.

4.4.4 Managing Project Folders

If you want to organize the mappings inside a project into folders, you can create as many folders as required, and add mappings to (or drag mappings into) them. Such folders are "virtual" and meaningful only inside a MapForce project; they do not correspond to actual folders on your operating system. One of the advantages of creating folders is that you can define common code generation settings (such as the target language and the output directory) for all the mapping files under that particular folder.



Name:

Code Generation Settings

● Use default project settings

○ Use following settings:

Output Directory:

Language: XSLT

Folder Properties dialog box

To create a folder inside a MapForce project:

1. Do one of the following:
   - o On the **Project** menu, click **Create Folder** 📁.
   - o Right-click the project, and select **Create Folder** 📁.
2. In the Properties dialog box, enter the required code generation settings, and click **OK**.

The settings you can define in the Folder Properties dialog box are as follows.

Name	The name of the folder.
Use default project settings	This is the default option and it means that the code generation settings in the current folder are the same as for the entire project. Therefore, when you generate code from you project, MapForce will use the code generation settings defined at the project level, not at the folder level. If your folder requires custom code generation settings (other than those set at the project level), select **Use the**

	following settings and specify the code output directory and language as required.
Output directory	Defines the Windows folder where the generated code (from all mappings in this folder) will be saved.
Language	Defines the code generation language for all mapping files in this folder.

Chapter 5

Designing Mappings

5 Designing Mappings

Altova website: Data integration tool

This section describes how to design data mappings, and ways in which you can transform data on the mapping area. It also includes various considerations applicable to mapping design. Use the following roadmap for quick access to specific tasks or concepts:

I want to...	Read this topic...
Create or edit path references to miscellaneous schema, instance, and other files used by a mapping.	Using Relative and Absolute Paths
Fine-tune the data mapping for specific needs (for example, influence the sequence of items in a target component).	Connection Types
Use the output of a component as input of another component.	Chained mappings / pass-through components
Process multiple files (for example, all files within a directory) in the same mapping, either as a source or a target.	Processing Multiple Input or Output Files Dynamically
Pass an external value (such as a string parameter) to the mapping.	Supplying Parameters to the Mapping
Get a string value out of the mapping, instead of a file.	Returning String Values from a Mapping
Store some mapping data temporarily for later processing (similar to variables in a programming language).	Using Variables
Sort data in ascending or descending order.	Sorting Data
Filter nodes/rows based on specific criteria, or process values conditionally.	Filters and Conditions
Merge or join data from multiple sources with different schema.	Joining Data Merging Data from Multiple Schemas
Process key-value pairs, for example, to convert months from numerical representation (01, 02, and so on) to text representation (January, February, and so on).	Using Value-Maps
Configure a mapping to return an error when a specific condition occurs.	Adding Exceptions
Learn how to avoid undesired results when designing complex mappings.	Mapping rules and strategies

Importantly, MapForce additionally includes an extensive built-in function library (see Function Library Reference) to help you with a wide array of processing tasks. When the built-in library is not sufficient, you can always build your own custom functions in MapForce, or re-use external XSLT files, as well as .dll or Java .class libraries. For further information, see Using Functions.

5.1 Using Relative and Absolute Paths

A mapping design file (*.mfd) may have references to several schema and instance files. The schema files are used by MapForce to determine the structure of the data to be mapped, and to validate it. The instance files, on the other hand, are required to read, preview, and validate the source data against the schema.

Mappings may also include references to StyleVision Power Stylesheets (*.sps) files, used to format data for outputs such as PDF, HTML and Word. Also, mappings may have references to file-based databases such as Microsoft Access or SQLite.

All references to files used by a mapping design 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 for setting or changing the path to miscellaneous file types referenced by a mapping, and the implications of using relative versus absolute paths.

5.1.1 Using Relative Paths on a Component

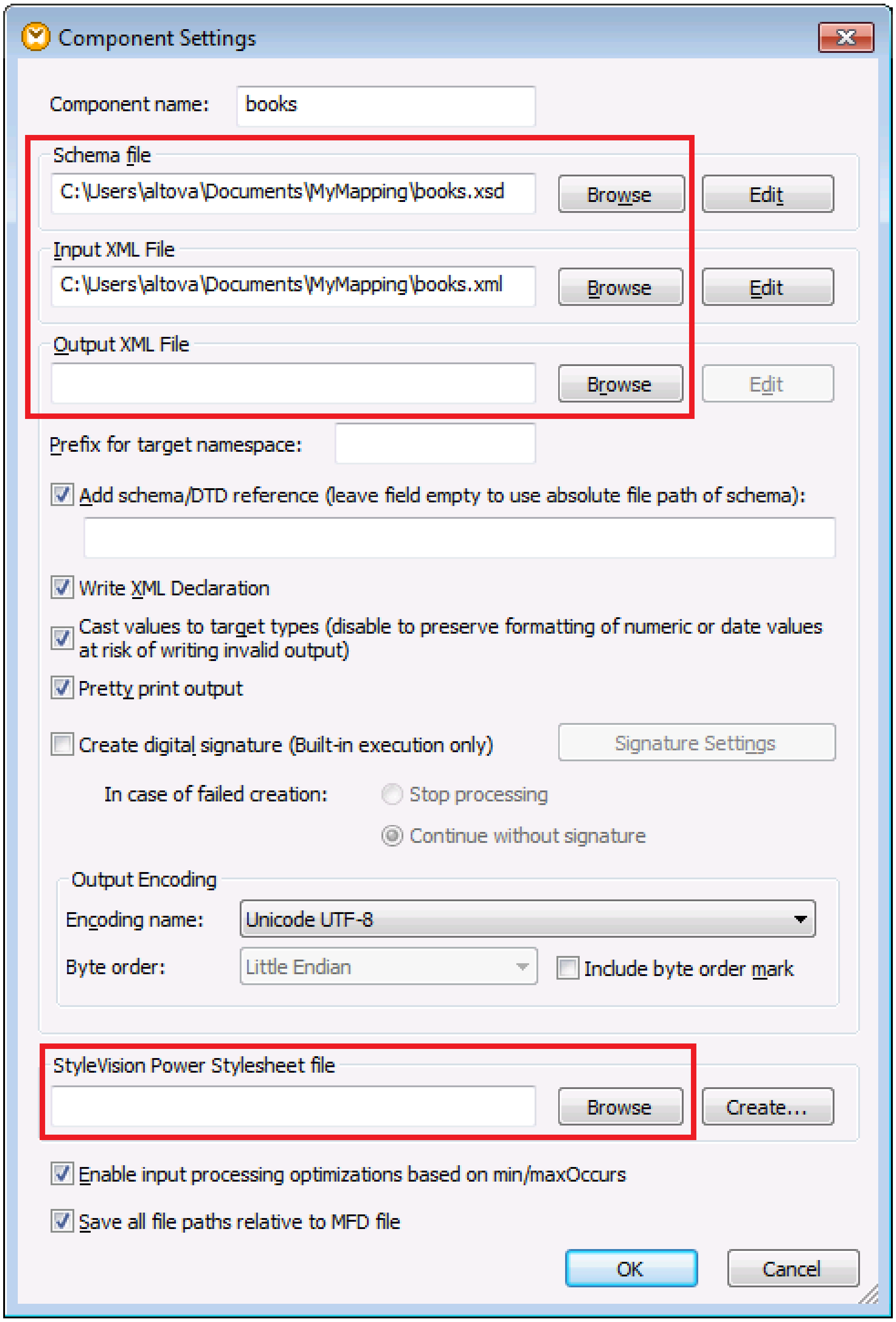
The Component Settings dialog box (illustrated below for an XML component) provides the option to specify either absolute or relative paths for various files which may be referenced by the component:

- Input files (that is, files from which MapForce reads data)
- Output files (that is, files to which MapForce writes data)
- Schema files (applicable to components which have a schema)
- Structure files (applicable to components which may have a complex structure, such as input or output parameters of user-defined functions, or variables)
- StyleVision Power Stylesheet (*.sps) files, used to format data for outputs such as PDF, HTML and Word.

You can enter relative paths directly in the relevant text boxes (shown enclosed in a red frame in the image below).

> Before entering relative file paths, make sure to save the mapping file (.mfd) first. Otherwise, all relative paths are resolved against the personal application folder of Windows (Documents \Altova\MapForce2018), which may not be the intended behavior.

You can also instruct MapForce to save all above-mentioned file paths relative to the mapping .mfd file. In the sample image below, notice the option **Save all file paths relative to MFD file**. If the check box is enabled (which is the default and recommended option), the paths of any files referenced by the component will be saved relative to the path of the mapping design file (.mfd). This affects all files referenced by the component (shown enclosed in a red frame in the image).



Component Settings

Component name: books

Schema file: C:\Users\altova\Documents\MyMapping\books.xsd

Input XML File: C:\Users\altova\Documents\MyMapping\books.xml

Output XML File:

Prefix for target namespace:

☑ Add schema/DTD reference (leave field empty to use absolute file path of schema):

☑ Write XML Declaration

☑ Cast values to target types (disable to preserve formatting of numeric or date values at risk of writing invalid output)

☑ Pretty print output

□ Create digital signature (Built-in execution only)

In case of failed creation: ○ Stop processing ● Continue without signature

Output Encoding

Encoding name: Unicode UTF-8

Byte order: Little Endian □ Include byte order mark

StyleVision Power Stylesheet file:

☑ Enable input processing optimizations based on min/maxOccurs

☑ Save all file paths relative to MFD file

Component Settings dialog box

Although the component illustrated above is an XML component, the setting **Save all file paths relative to MFD file** works in the same way for the following files:

- Structure files used by complex input or output parameters of user-defined functions and

variables of complex type
- Input or output flat files *
- Schema files referenced by database components which support XML fields *
- Input or output XBRL, FlexText, EDI, Excel 2007+, JSON files **

* MapForce Professional and Enterprise Edition
** MapForce Enterprise Edition only

Taking the component above as an example, if the .mfd file is in the same folder as the **books.xsd** and **books.xml** files, the paths will be changed as follows:

C:\Users\altova\Documents\MyMapping\books.xsd will change to **books.xsd**
C:\Users\altova\Documents\MyMapping\books.xml will change to **books.xml**

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

When the check box is enabled, MapForce will also keep track of the files referenced by the component if you save the mapping to a new folder using the **Save as** menu command. Also, if all files are in the same directory as the mapping, path references will not be broken when you move the entire directory to a new location on the disk.

Using relative paths (and, therefore, enabling the **Save all file paths relative to MFD file** check box) may be important in many cases, for example:

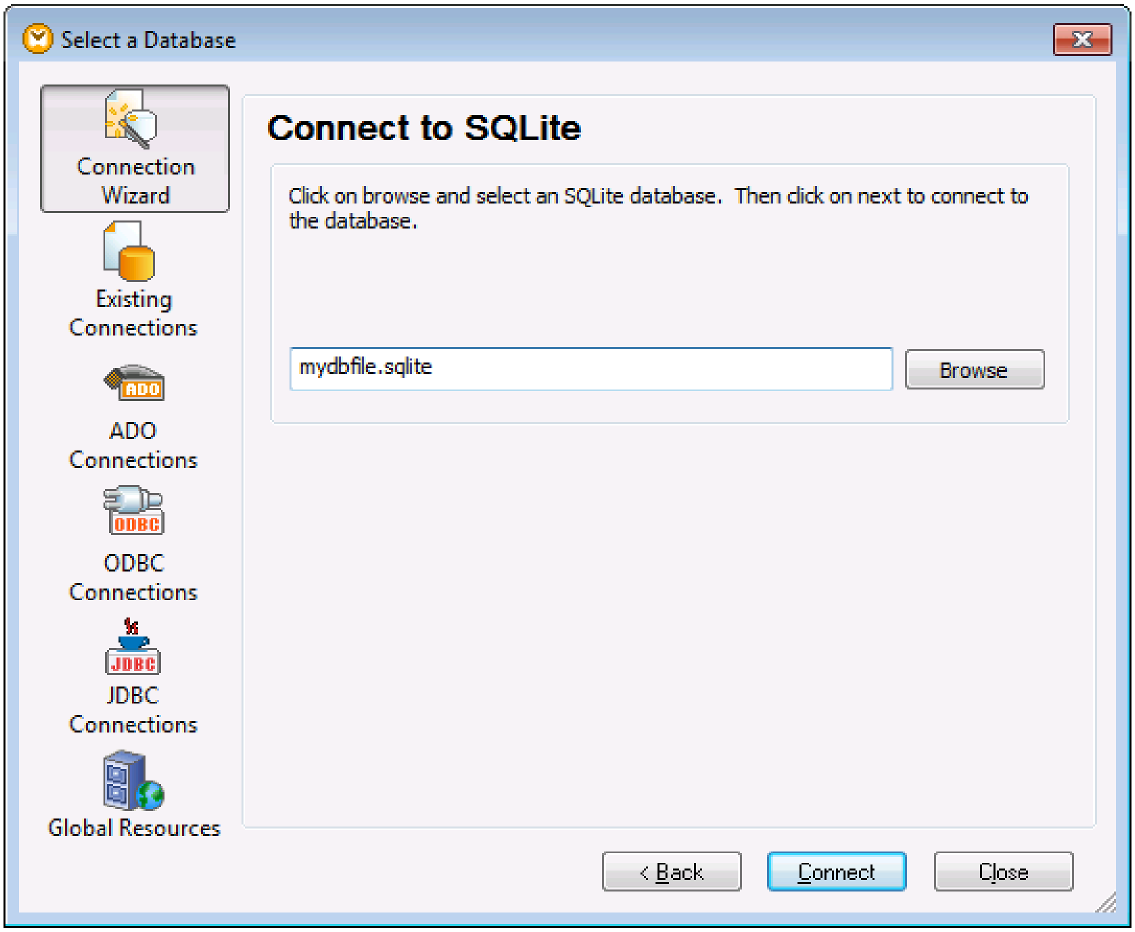
- The location of the mapping on your operating system is likely to change in future.
- The mapping is stored in a directory which is under source control (using a version control system such as TortoiseSVN, for example).
- You intend to deploy the mapping for execution to a different machine or even to a different operating system.

If the **Save all file paths relative to MFD file** check box is disabled, saving the mapping does not modify the file paths (that is, they remain as they appear in the Component Settings dialog box).

5.1.2 Setting the Path to File-Based Databases

When you add a database file such as Microsoft Access or SQLite to the mapping (see Starting the Database Connection Wizard), you can use a relative path instead of an absolute one. To use a relative path, enter the required relative path instead of clicking **Browse** in the Database Connection Wizard.

> Before entering relative file paths, make sure to save the mapping file (.mfd) first. Otherwise, all relative paths are resolved against the personal application folder of Windows (Documents \Altova\MapForce2018), which may not be the intended behavior.



Connect to SQLite

Click on browse and select an SQLite database. Then click on next to connect to the database.

mydbfile.sqlite

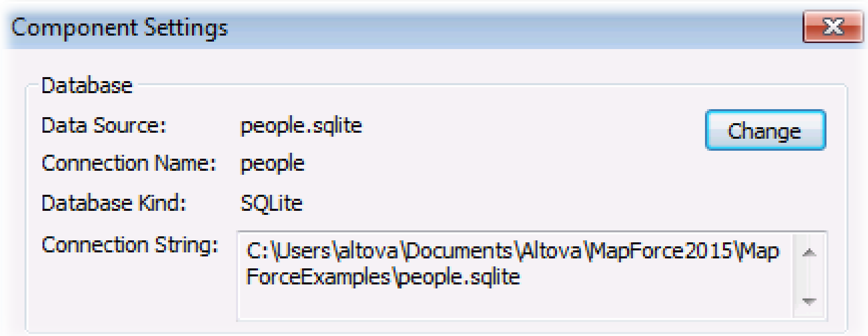
Database Connection Wizard

If the database is a SQLite database, the **Connect** button becomes enabled if the following is true:

- The path points to a file that can be resolved relatively to the mapping (.mfd) file
- The referenced file is a SQLite database.

To change the path of a database component which is already in the mapping, do the following:

1. Right-click the header of the database component, and select **Properties** (see also Changing the Component Settings). Alternatively, double-click the component title bar.
2. On the Component Settings dialog box, click **Change**.



Data Source: people.sqlite
Connection Name: people
Database Kind: SQLite
Connection String: C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\people.sqlite

This re-opens the Database Connection Wizard, from where you can change the database connection properties (including the path) as already shown above.

Note that “Connection String” always contains an absolute path. It is the database which is used for the structure information in the component. The relative path in “Data Source” indicates that the component was created with a relative file path.

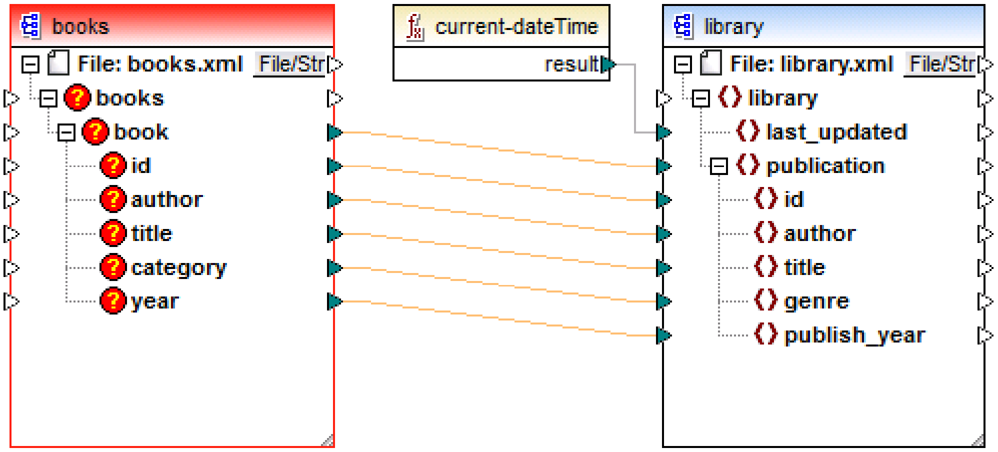
Note: When you generate program code, or when you compile MapForce Server execution files (.mfx), or when you deploy the mapping to FlowForce Server, a relative path will be converted to an absolute path if the check box **Make paths absolute in generated code** is selected from the mapping settings (see About Paths in Generated Code).

5.1.3 Fixing Broken Path References

When you add or change a file reference in a mapping, and the path cannot be resolved, MapForce displays a warning message. This way, MapForce diminishes the chance for broken path references to happen. Nevertheless, broken path references may still occur in cases such as:

- 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 this happens, MapForce highlights the component in red, for example:



Broken path reference

The solution in this case is to double-click the component header and update any broken path references in the **Component Settings** dialog box (see also Changing the Component Settings).

5.1.4 Paths in Various 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 no longer run by MapForce. Instead, the mappings are run by the target environment you have chosen (for example, RaptorXML Server, MapForce Server, or a C# application). The implication is that, for the mapping to run successfully, any relative paths must be meaningful in the environment where the mapping runs.

Consequently, when the mapping uses relative paths to instance or schema files, consider the base path to be as follows for each target language:

Target language	Base path
XSLT/XSLT2	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 (.mfd) file.
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*

If required, you can instruct MapForce to convert all paths from relative to absolute when generating code for a mapping. This option might be useful if you run the mapping code (or the MapForce Server execution file) on the same operating system, or perhaps on another operating system where any absolute path references used by the mapping can still be resolved.

To convert all paths to absolute in the generated code, select the **Make paths absolute in generated code** check box, on the Mapping Settings dialog box (see Changing the Mapping Settings).

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

- Input and output instance files for all file-based component kinds
- Access and SQLite database files used as mapping components

When the check box is not selected, the file paths will be preserved as they are defined in the component settings.

5.1.5 Copy-Paste and Relative Paths

When you copy a component from a mapping and paste it into another, a check is performed to ensure that relative paths of schema files can be resolved against the folder of the destination mapping. If the path cannot be resolved, you will be prompted to make the relative paths absolute by means of the folder of the source mapping. It is recommended to save the destination mapping first, otherwise relative paths are resolved against the personal application folder.

5.2 Connection Types

When you create a mapping connection (and both the source and the target item have child items), you can optionally choose the type of the connection to be one of the following.

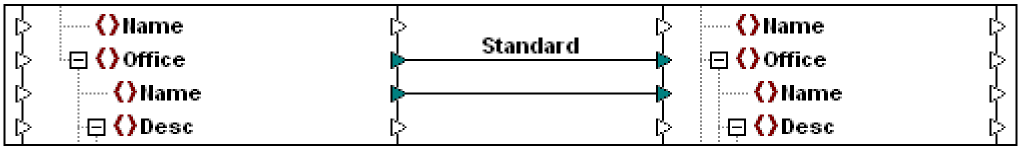
- Target Driven (Standard)
- Source Driven (Mixed Content)
- Copy-All (Copy Child Items).

The connection type determines the sequence of children items in the output generated by the mapping. This section provides information about each connection type and the scenarios when they are useful.

5.2.1 Target-driven connections

When a connection is "target-driven" (or "standard"), the sequence of child nodes in the mapping output is determined by the sequence of nodes in the target schema. This connection type is suitable for most mapping scenarios and is the default connection type used in MapForce.

On a mapping, target-driven connections are shown with a solid line.



Target-driven connections might not be suitable when you want to map XML nodes that contain mixed context (character data as well as child elements), for example:

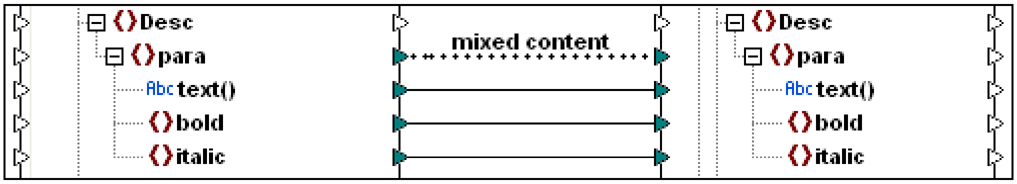
```
<p>This is our <i>best-selling</i> product.</p>
```

With mixed content, it is likely that you want to preserve the sequence of items as they appear in the source file, in which case a source-driven connection is recommended (see Source-driven connections).

5.2.2 Source-driven connections

Source-driven (Mixed Content) mapping enables you to automatically map text and child nodes in the same sequence that they appear in the XML **source** file.

- Mixed content text node content is supported/mapped.
- The sequence of child nodes is dependent on the source XML instance file.



Mixed content mappings are shown with a dotted line.

Source-driven / mixed content mapping can also be applied to XML schema **complexType** items. Child nodes will then be mapped according to their sequence in the XML source file.

Source-driven / mixed content mapping supports:

Mappings from

- As **source** components:
  - XML schema complexTypes (including mixed content, i.e. mixed=true)
  - XML schema complexTypes (including mixed content) in embedded schemas of a database field
- As **target** components:
  - XML schema complexTypes (including mixed content),
  - XML schema complexTypes (including mixed content) in embedded schemas of a database field

Note: CDATA sections are treated as text.

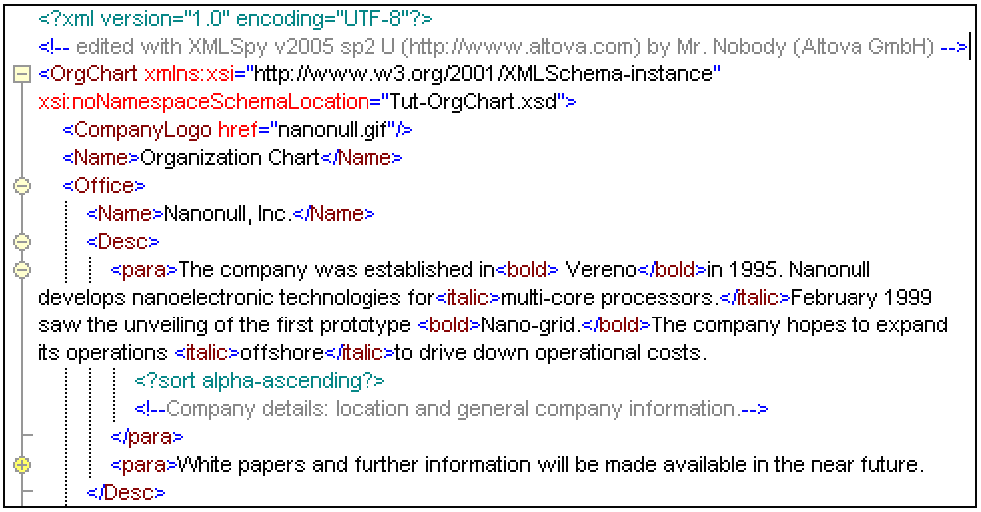
5.2.2.1 Mapping mixed content

The files used in the following example (**Tut-OrgChart.mfd, Tut-OrgChart.mfd.xml, Tut-OrgChart.mfd.xsd, Tut-Person.xsd**) are available in the **...\MapForceExamples\Tutorial\** folder.

Source XML instance

A portion of the **Tut-OrgChart.xml** file used in this section is shown below. Our area of concern is the mixed content element "para", along with its child nodes "bold" and "italic".

The `para` element also contains a Processing Instruction (`<?sort alpha-ascending?>`) as well as Comment text (`<!--Company details... -->`) which can also be mapped, as shown below.



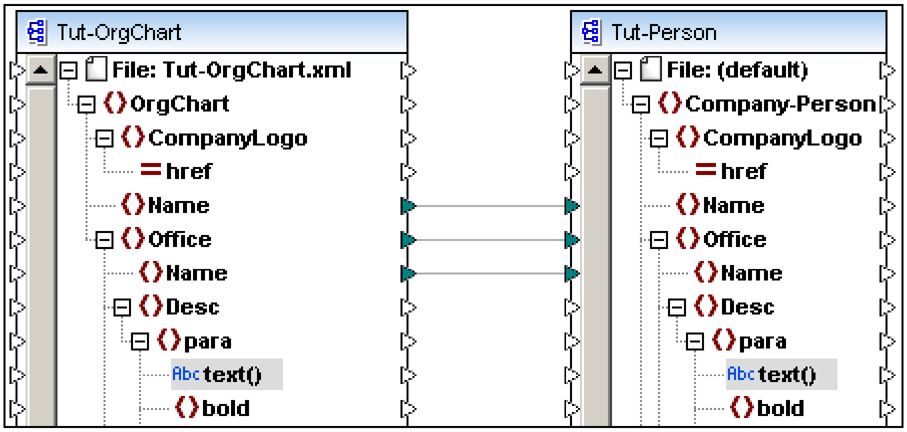
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp2 U (http://www.altova.com) by Mr. Nobody (Altova GmbH) -->
<OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Tut-OrgChart.xsd">
  <CompanyLogo href="nanonull.gif"/>
  <Name>Organization Chart</Name>
  <Office>
    <Name>Nanonull, Inc.</Name>
    <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.
    </Desc>
```

Note the sequence of the text and bold/italic nodes in the XML instance file:

```
<para> The company...
    <bold>Vereno</bold>in 1995 ...
    <italic>multi-core...</italic>February 1999
    <bold>Nano-grid.</bold>The company ...
    <italic>offshore...</italic>to drive...
</para>
```

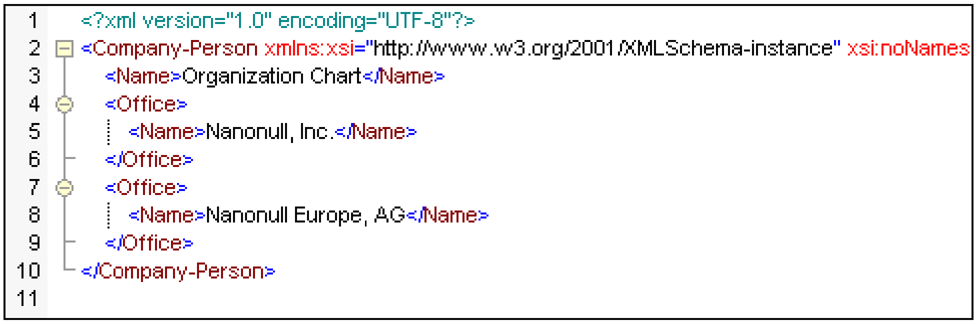
Initial mapping

The initial state of the mapping when you open **Tut-Orgchart.mfd** is shown below.



Output of above mapping

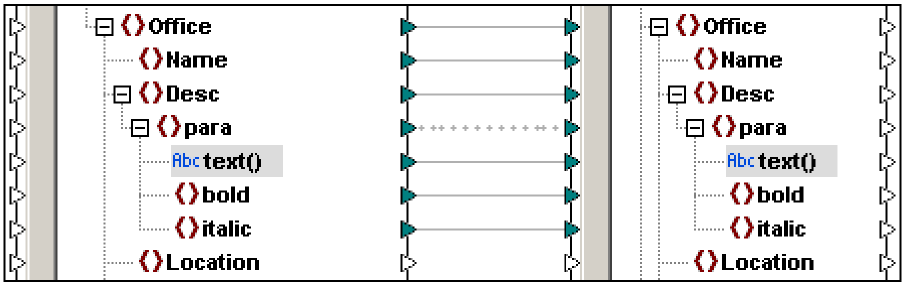
The result of the initial mapping is shown below: Organization Chart as well as the individual office names have been output.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
    <Name>Organization Chart</Name>
    <Office>
        <Name>Nanonull, Inc.</Name>
    </Office>
    <Office>
        <Name>Nanonull Europe, AG</Name>
    </Office>
</Company-Person>
```

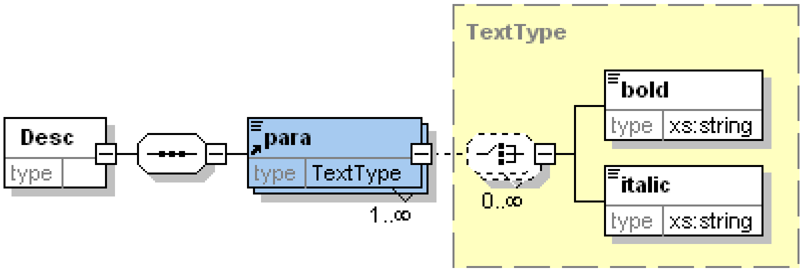
Mapping the para element

The image below shows an example of mixed content mapping. The para element is of mixed content, and the connector is shown as a **dotted** line to highlight this. The **text()** node contains the textual data and needs to be mapped for the text to appear in the target component.



To annotate (add a label to) any connection, right-click it and select **Properties** (see Annotating Connections).

The image below shows the content model of the Description element (Desc) of the **Tut-OrgChart.xsd** schema file. This definition is identical in both the source and target schemas used in this example.

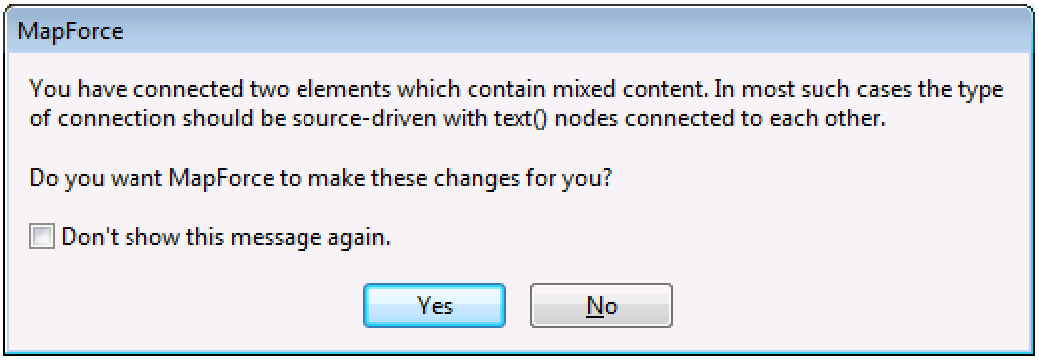


Note the following properties of the **para** element in the Content model:

- **para** is a complexType with mixed="true", of type "TextType"
- **bold** and **italic** elements are both of type "xs:string", they have not been defined as recursive in this example, i.e. neither **bold**, nor **italic** are of type "TextType"
- **bold** and **italic** elements can appear any number of times in any sequence within **para**
- any number of text nodes can appear within the **para** element, interspersed by any number of **bold** and **italic** elements.

To create mixed content connections between items:

1. Select the menu option **Connection | Auto Connect Matching Children** to activate this option, if it is not currently activated.
2. Connect the **para** item in the source schema, with the **para** item in the target schema. A message appears, asking if you would like MapForce to define the connectors as source driven.



MapForce

You have connected two elements which contain mixed content. In most such cases the type of connection should be source-driven with text() nodes connected to each other.

Do you want MapForce to make these changes for you?

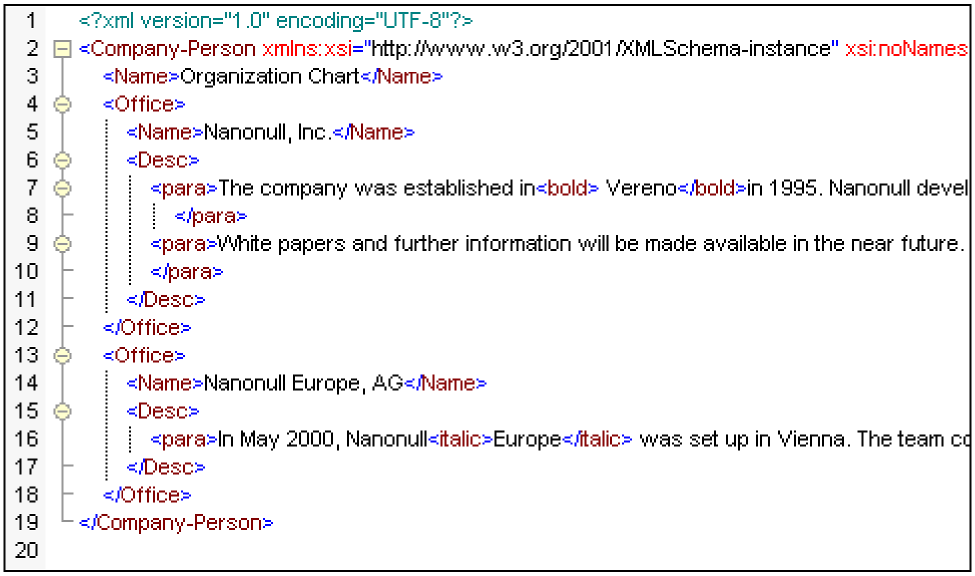
□ Don't show this message again.

3. Click Yes to create a mixed content connection.

Note: Para is of mixed content, and makes the message appear at this point. The mixed-content message also appears if you only map the para items directly, without having the autoconnect option activated.

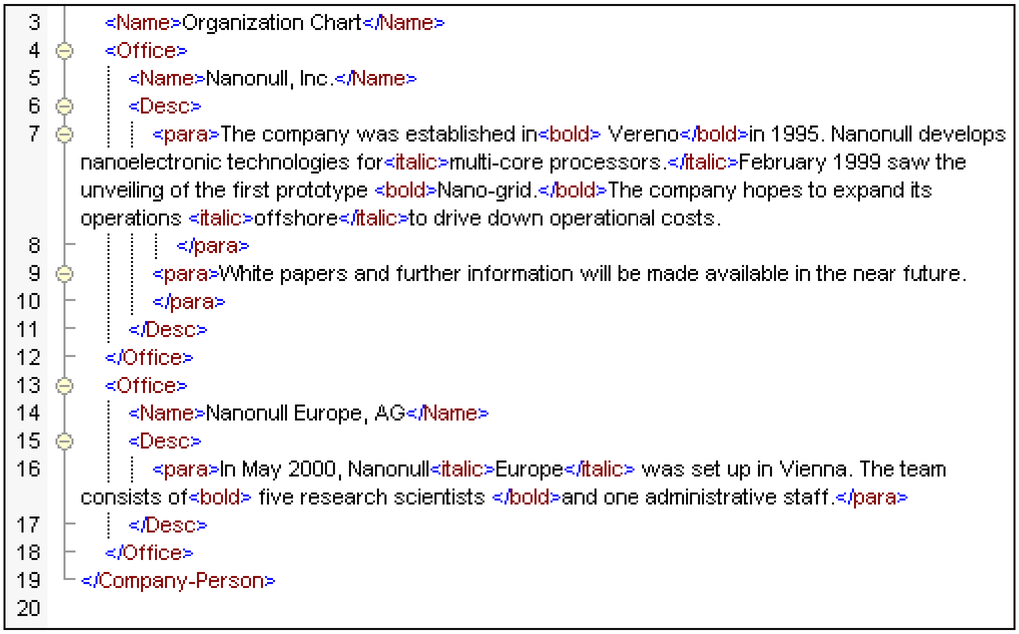
All child items of para have been connected. The connector joining the para items is displayed as a dotted line, to show that it is of type mixed content.

4. Click the Output tab to see the result of the mapping.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
  <Name>Organization Chart</Name>
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Desc>
      <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull devel
        </para>
      <para>White papers and further information will be made available in the near future.
      </para>
    </Desc>
  </Office>
  <Office>
    <Name>Nanonull Europe, AG</Name>
    <Desc>
      <para>In May 2000, Nanonull<italic>Europe</italic> was set up in Vienna. The team co
    </Desc>
  </Office>
</Company-Person>

```

5. Click the word **Wrap** icon in the Output tab icon bar, to view the complete text in the Output window.



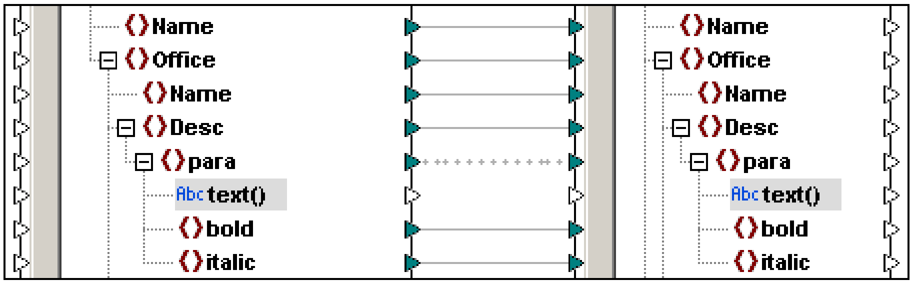
```
    <Name>Organization Chart</Name>
    <Office>
        <Name>Nanonull, Inc.</Name>
        <Desc>
            <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull develops
nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999 saw the
unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand its
operations <italic>offshore</italic>to drive down operational costs.
                </para>
            <para>White papers and further information will be made available in the near future.
            </para>
        </Desc>
    </Office>
    <Office>
        <Name>Nanonull Europe, AG</Name>
        <Desc>
            <para>In May 2000, Nanonull<italic>Europe</italic> was set up in Vienna. The team
consists of<bold> five research scientists </bold>and one administrative staff.</para>
        </Desc>
    </Office>
</Company-Person>
```

The mixed content text of each office description has been mapped correctly; the text, as well as the bold and italic tag content, have been mapped as they appear in the XML **source** file.

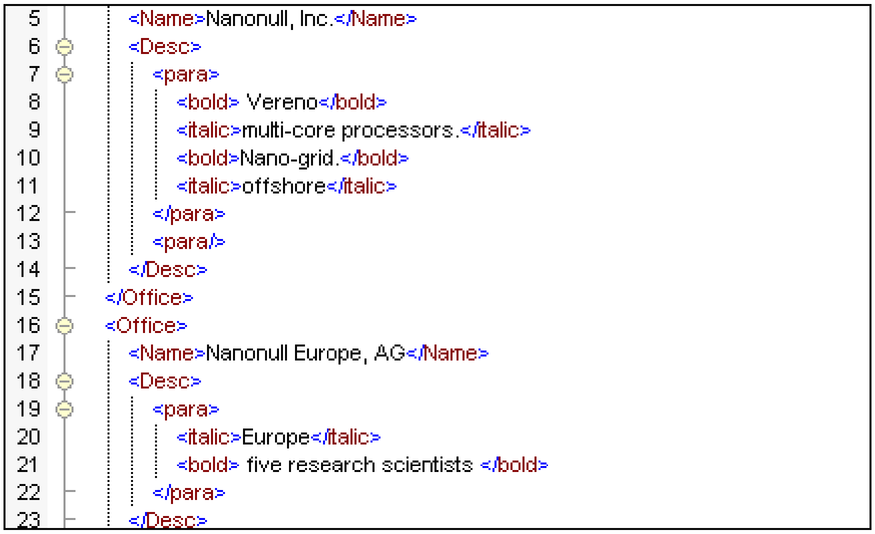
6. Switch back to the Mapping view.

To remove text nodes from mixed content items:

1. Click the **text()** node connector and press Del. to delete it.



2. Click the Output tab to see the result of the mapping.



```
    <Name>Nanonull, Inc.</Name>
    <Desc>
        <para>
            <bold> Vereno</bold>
            <italic>multi-core processors.</italic>
            <bold>Nano-grid.</bold>
            <italic>offshore</italic>
        </para>
        <para/>
    </Desc>
</Office>
<Office>
    <Name>Nanonull Europe, AG</Name>
    <Desc>
        <para>
            <italic>Europe</italic>
            <bold> five research scientists </bold>
        </para>
    </Desc>
```

Result:

- all **text** nodes of the para element have been removed.
- mapped bold and italic text content remain
- the bold and italic item **sequence** still follows that of the source XML file.

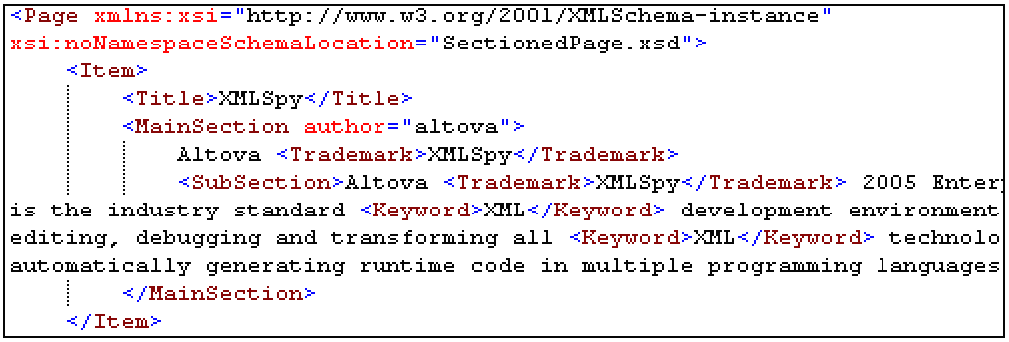
To map the Processing Instructions and Comments:

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

5.2.2.2 *Mixed content example*

The following example is available as "**ShortApplicationInfo.mfd**" in the **...\MapForceExamples** folder.

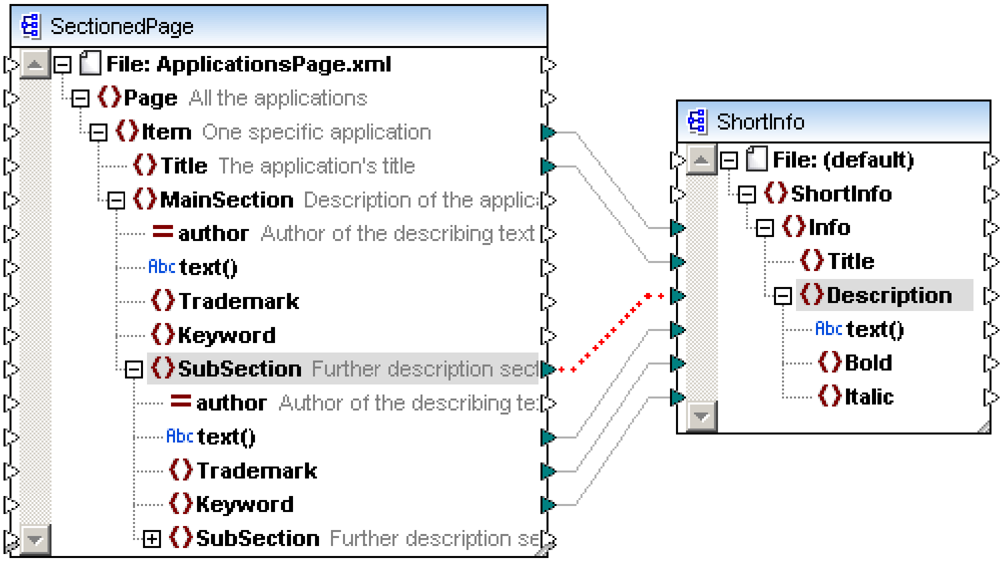
A snippet of the XML source file for this example is shown below.



```
<Page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SectionedPage.xsd">
    <Item>
        <Title>XMLSpy</Title>
        <MainSection author="altova">
            Altova <Trademark>XMLSpy</Trademark>
            <SubSection>Altova <Trademark>XMLSpy</Trademark> 2005 Enter
is the industry standard <Keyword>XML</Keyword> development environment
editing, debugging and transforming all <Keyword>XML</Keyword> technolo
automatically generating runtime code in multiple programming languages
        </MainSection>
    </Item>
```

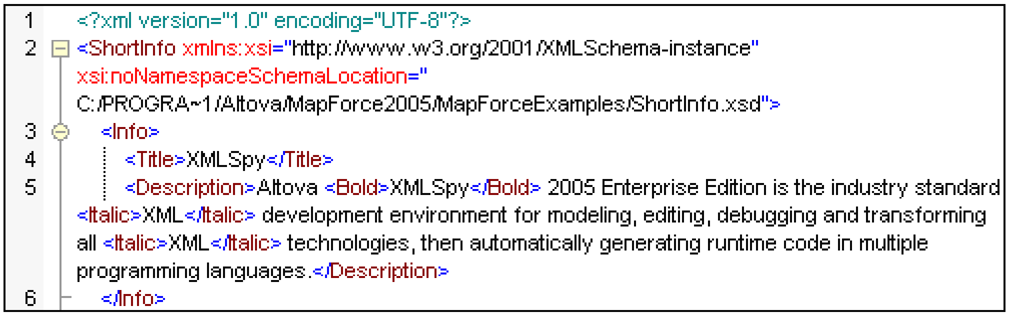
The mapping is shown below. Please note the following:

- The "SubSection" item connector is of mixed content, and is mapped to the Description item in the target XML/schema.
- The text() nodes are mapped to each other
- Trademark text is mapped to the Bold item in the target
- Keyword text is mapped to the Italic item in the target


SectionedPage
File: ApplicationsPage.xml
Page All the applications
Item One specific application
Title The application's title
MainSection Description of the applic
author Author of the describing text
text()
Trademark
Keyword
SubSection Further description sect
author Author of the describing te
text()
Trademark
Keyword
SubSection Further description se
ShortInfo
File: (default)
ShortInfo
Info
Title
Description
text()
Bold
Italic

Mapping result

The mixed content text of each description has been mapped correctly; the text, as well as the bold and italic tag content, have been mapped as they appear in the XML source file.



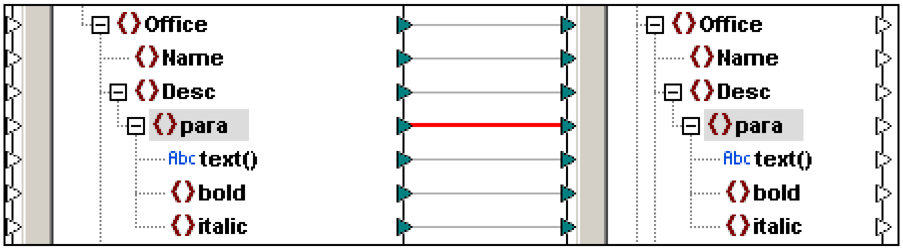
```
<?xml version="1.0" encoding="UTF-8"?>
<ShortInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="
C:/PROGRA~1/Altova/MapForce2005/MapForceExamples/ShortInfo.xsd">
  <Info>
    <Title>XMLSpy</Title>
    <Description>Altova <Bold>XMLSpy</Bold> 2005 Enterprise Edition is the industry standard
<Italic>XML</Italic> development environment for modeling, editing, debugging and transforming
all <Italic>XML</Italic> technologies, then automatically generating runtime code in multiple
programming languages.</Description>
  </Info>
```

5.2.2.3 *Using standard connections on mixed content items*

As mentioned before, source-driven (not standard) connections are normally used when mapping data from mixed content nodes. Otherwise, the resulting output may be undesirable. To see the consequences of using a standard (target-driven) connection when mapping data from a mixed content node, follow the steps below:

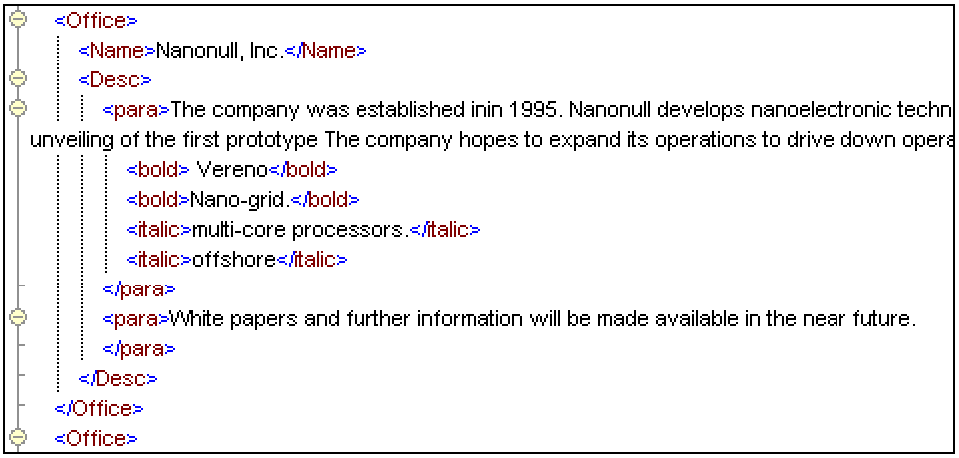
1. Open the mapping **Tut-OrgChart.mfd** from the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder.
2. Create a connection between the `para` node in the source and the `para` node in the target. A message appears, asking if you would like MapForce to define the connections

as source-driven. Click **No** (this disregards the MapForce suggestion and creates a standard connection).



Note: Make sure that the connection is standard (target-driven), as shown above. If a Copy-All connection is created automatically, right-click the connection, and select **Target Driven (Standard)** from the context menu.

3. Click the **Output** tab to see the result of the mapping.



```
<Office>
  <Name>Nanonull, Inc.</Name>
  <Desc>
    <para>The company was established inin 1995. Nanonull develops nanoelectronic techn
unveiling of the first prototype The company hopes to expand its operations to drive down opera
      <bold> Vereno</bold>
      <bold>Nano-grid.</bold>
      <italic>multi-core processors.</italic>
      <italic>offshore</italic>
    </para>
    <para>White papers and further information will be made available in the near future.
    </para>
  </Desc>
</Office>
<Office>
```

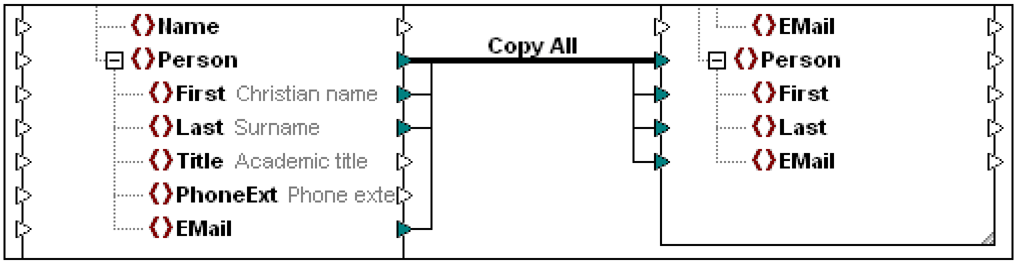
As illustrated above, mapping mixed content nodes using standard connections produces the following result:

- The content of the `text()` source item is copied to the target; however, the sequence of child nodes (`bold` and `italic`, in this case) in the output corresponds to the sequence in the target XML schema. In other words, the child nodes (`bold` and `italic`, in this case) appear after the mixed content node text.
- For each `para` element, MapForce has mapped the `text()` node first, then all `bold` items, and, finally, all `italic` items. As a result, multiple `bold` and `italic` items appear stacked on each other. Note that the content of each item is mapped if a connection exists to it from the source.

5.2.3 Copy-All Connections

Copy-All connections map data between complex structures (nodes with children items) that are very similar or identical. The main benefit of "Copy-All" connections is that they simplify the mapping workspace (one "thick" connection is created instead of multiple).

On the mapping, a "Copy-All" connection appears as a single bold line (with input and output "forks" for each child item) that connects two identical or similar structures.



Copy-All connection

When you draw a mapping connection between two structures on the mapping, MapForce creates a "Copy-All" connection automatically if it detects that the source and target structure are assignment compatible (that is, when both structures are either of the same type, or the target is a subtype of the source type). At mapping runtime, all instance data will be copied from the source to the target recursively, including children.

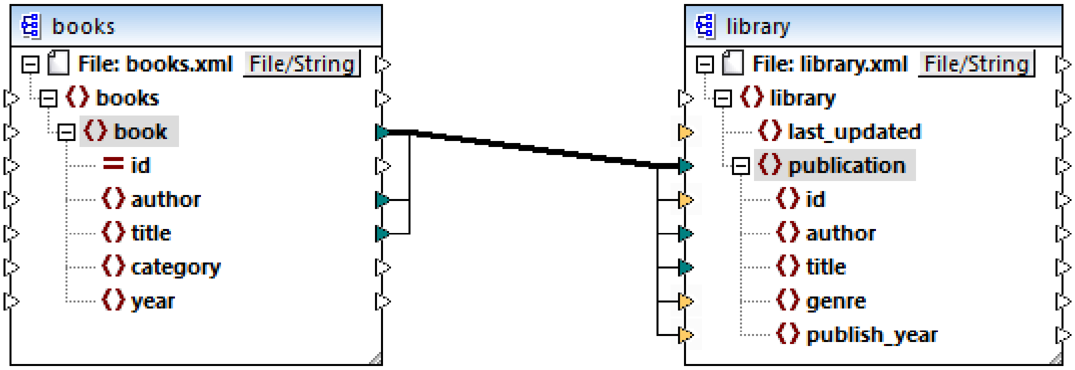
To create a "Copy-All" connection manually, right-click an existing connection between two similar nodes with child items, and select **Copy-All (Copy Child Items)** from the context menu.

Note the following:

- In contexts where a "Copy-All" connection is not meaningful or not supported, it is not possible to create this kind of connection manually.
- A "Copy-All" connection cannot be created to the `root` element of an XML/Schema component.
- When creating "Copy-All" connections between a schema and a parameter of a user-defined function, the two components must be based on the same schema. It is not necessary that they both have the same root elements, however.

For an example of a "Copy-All" connection created manually, take the following steps:

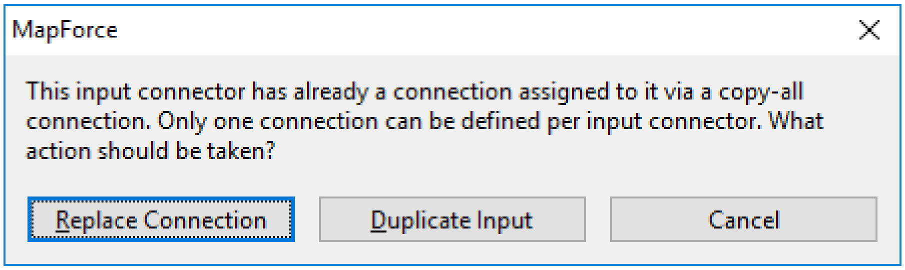
1. Create a new mapping.
2. On the **Insert** menu, click **XML Schema/File** and browse for the **books.xml** file located in the folder **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\**.
3. On the **Insert** menu, click **XML Schema/File** and browse for the **library.xsd** file located in the folder **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\**.
4. Draw a mapping connection between the `book` node of the "books" component to the `publication` node of the "library" component.
5. Right-click the new connection, and select **Copy-All (Copy Child Items)** from the context menu.



If there are slight differences between the source and the target structures, the "Copy-All" connection will enumerate, at mapping runtime, the source items (such as elements and attributes) and will copy only those that exist in the target type. This is repeated recursively.

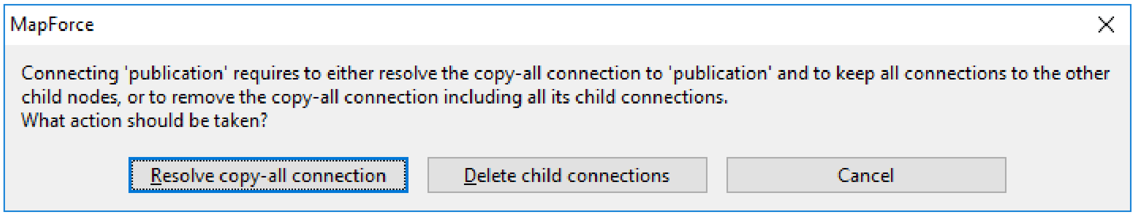
For example, in the mapping above, only two child items are identical between the two structures (`author` and `title`) and thus they are mapped to the target. The item `id` is not included automatically because it is an attribute in the source and an element in the target. If you need to map, for example, `category` to `genre`, the "Copy-All" connection is no longer possible, because these are different items.

When an input connector (the small triangle to the side of the component) receives a "Copy-All" connection, it cannot accept any other connections. In the example above, if you attempt to create a connection between `category` and `genre`, MapForce prompts you to either replace it, or duplicate the input.


MapForce

This input connector has already a connection assigned to it via a copy-all connection. Only one connection can be defined per input connector. What action should be taken?

Duplicating input is meaningful only if you want the target to accept data from more than one input, which is not required here (see also Duplicating Input). If you choose to replace the "Copy-All" connection, a message box prompts you again to either resolve or delete the "Copy-All" connection.


MapForce

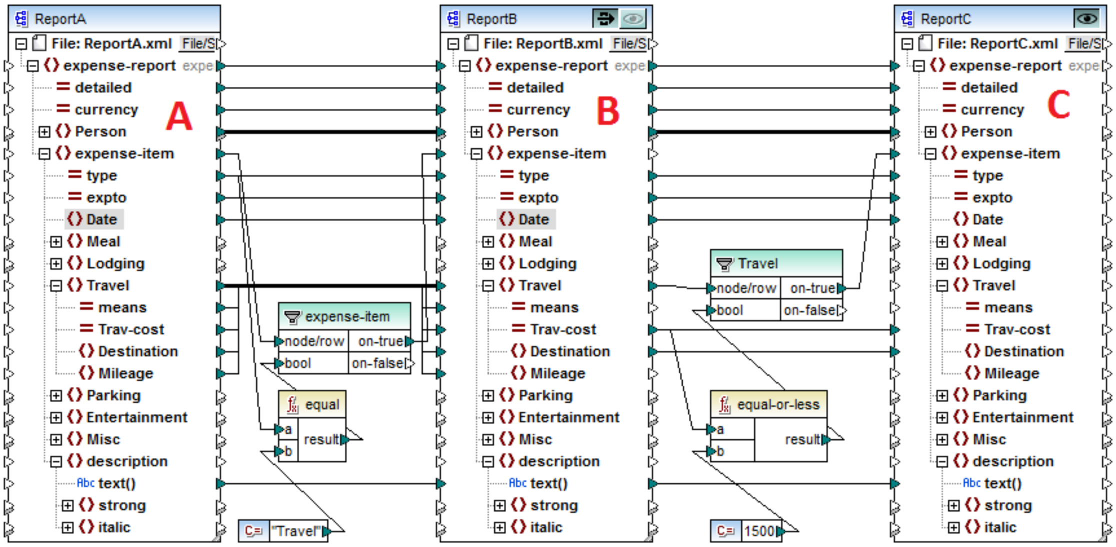
Connecting 'publication' requires to either resolve the copy-all connection to 'publication' and to keep all connections to the other child nodes, or to remove the copy-all connection including all its child connections.
What action should be taken?

Click **Resolve copy-all connection** if you want to replace the "Copy-All" connection by standard individual target-driven connections to corresponding child items. If you prefer to remove the "Copy-All" connection completely, click **Delete child connections**.

5.3 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\MapForce2018\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.

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 and Example: Pass-Through Inactive, illustrate in more detail how the source data is transferred differently when the pass-through button is active or inactive.

5.3.1 Example: Pass-Through Active

The mapping used in this example (**ChainedReports.mfd**) is available in the **<Documents>\Altova\MapForce2018\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:

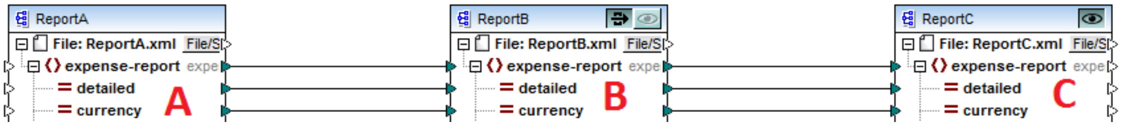
```xml
<?xml version="1.0" encoding="UTF-8"?>
<expense-report currency="USD" detailed="true">
   <Person>
      <First>Fred</First>
      <Last>Landis</Last>
      <Title>Project Manager</Title>
      <Phone>123-456-78</Phone>
      <Email>f.landis@nanonull.com</Email>
   </Person>
   <expense-item type="Travel" 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

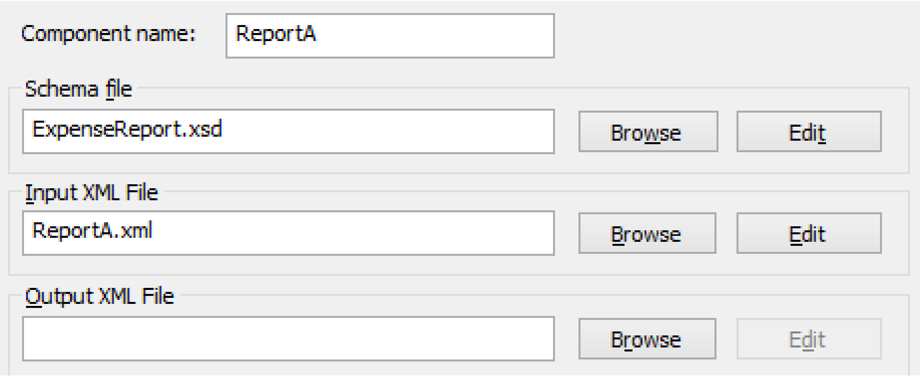
The goal of the mapping it to produce, based on the file above, two further reports:

- **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.



Component name: ReportA

Schema file

ExpenseReport.xsd

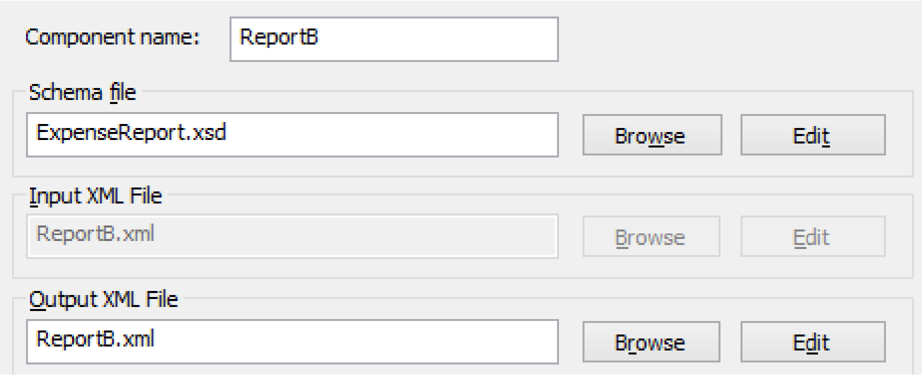
Input XML File

ReportA.xml

Output XML File

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.



Component name: ReportB

Schema file

ExpenseReport.xsd

Input XML File

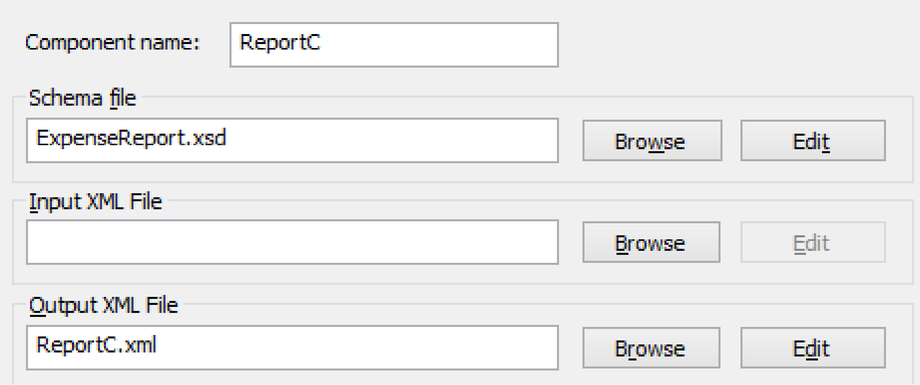
ReportB.xml

Output XML File

ReportB.xml

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.



Component name: ReportC

Schema file

ExpenseReport.xsd

Input XML File

Output XML File

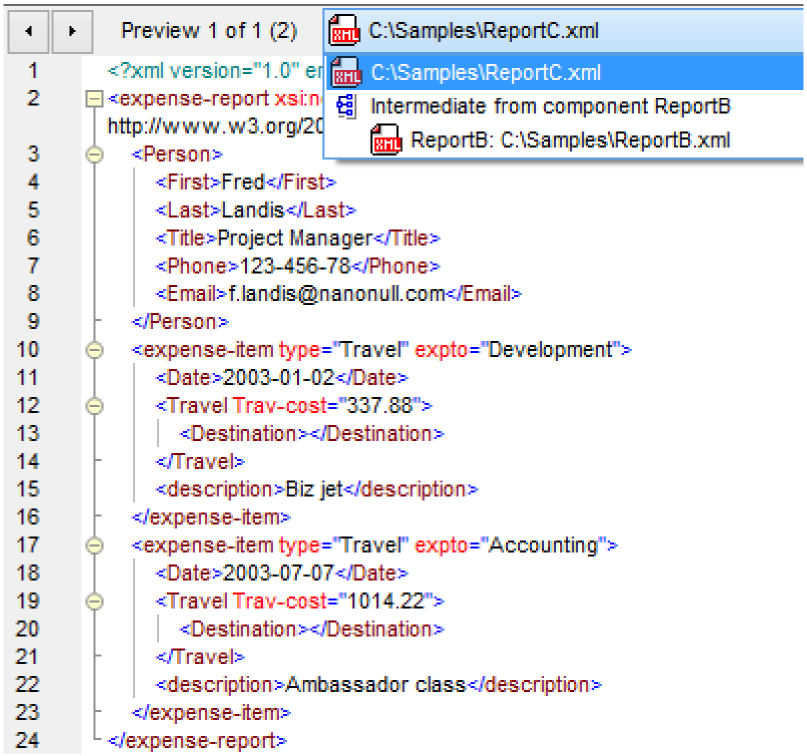
ReportC.xml

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.

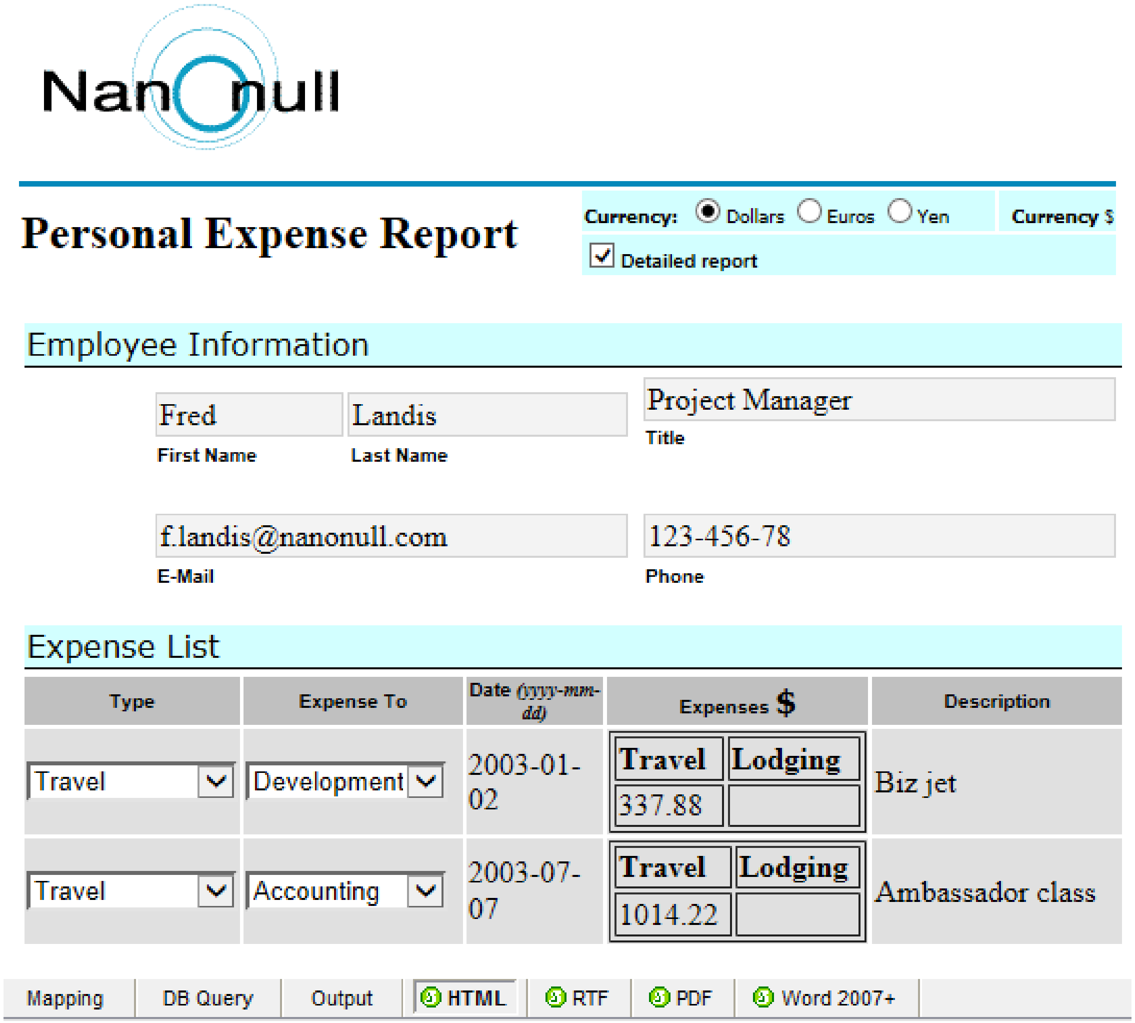

Preview 1 of 1 (2)
C:\Samples\ReportC.xml
C:\Samples\ReportC.xml
Intermediate from component ReportB
ReportB: C:\Samples\ReportB.xml

```
<?xml version="1.0" er
<expense-report xsi:n
http://www.w3.org/20
  <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></Destination>
    </Travel>
    <description>Biz jet</description>
  </expense-item>
  <expense-item type="Travel" expto="Accounting">
    <Date>2003-07-07</Date>
    <Travel Trav-cost="1014.22">
      <Destination></Destination>
    </Travel>
    <description>Ambassador class</description>
  </expense-item>
</expense-report>
```

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 file. To generate and view this output in MapForce, click the tab with the corresponding name.



Personal Expense Report

Currency: ● Dollars ○ Euros ○ Yen Currency $

☑ Detailed report

Employee Information

Fred	Landis	Project Manager
First Name	Last Name	Title
f.landis@nanonull.com		123-456-78
E-Mail		Phone

Expense List

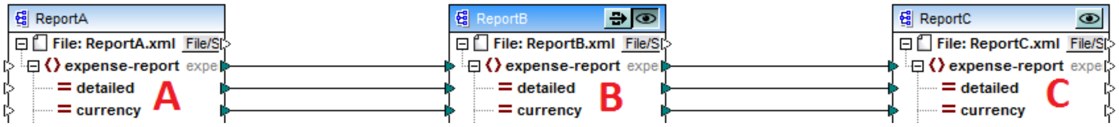
Type	Expense To	Date (yyyy-mm-dd)	Expenses $	Description
Travel	Development	2003-01-02	Travel | Lodging 337.88 |	Biz jet
Travel	Accounting	2003-07-07	Travel | Lodging 1014.22 |	Ambassador class

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).

5.3.2 Example: Pass-Through Inactive

The mapping used in this example (**ChainedReports.mfd**) is available in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder. This example illustrates how output is generated differently when the pass-through button is deactivated on the intermediate component.

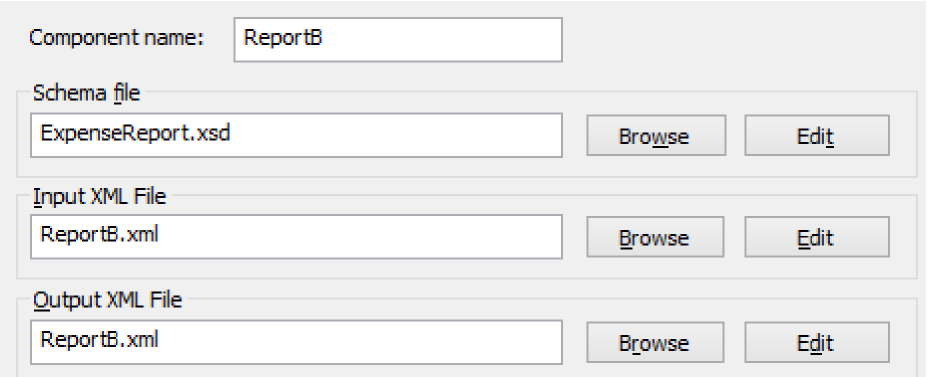


As explained in Example: Pass-Through Active, the goal of the mapping is to produce two separate reports. In the previous example, the pass-through button was active ⇛, and both reports were generated as expected and could be viewed in the **Output** tab. However, if you want to preview only one of the 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).


Component name: ReportB

Schema file
ExpenseReport.xsd
Browse Edit

Input XML File
ReportB.xml
Browse Edit

Output XML File
ReportB.xml
Browse Edit

Settings of the intermediate component

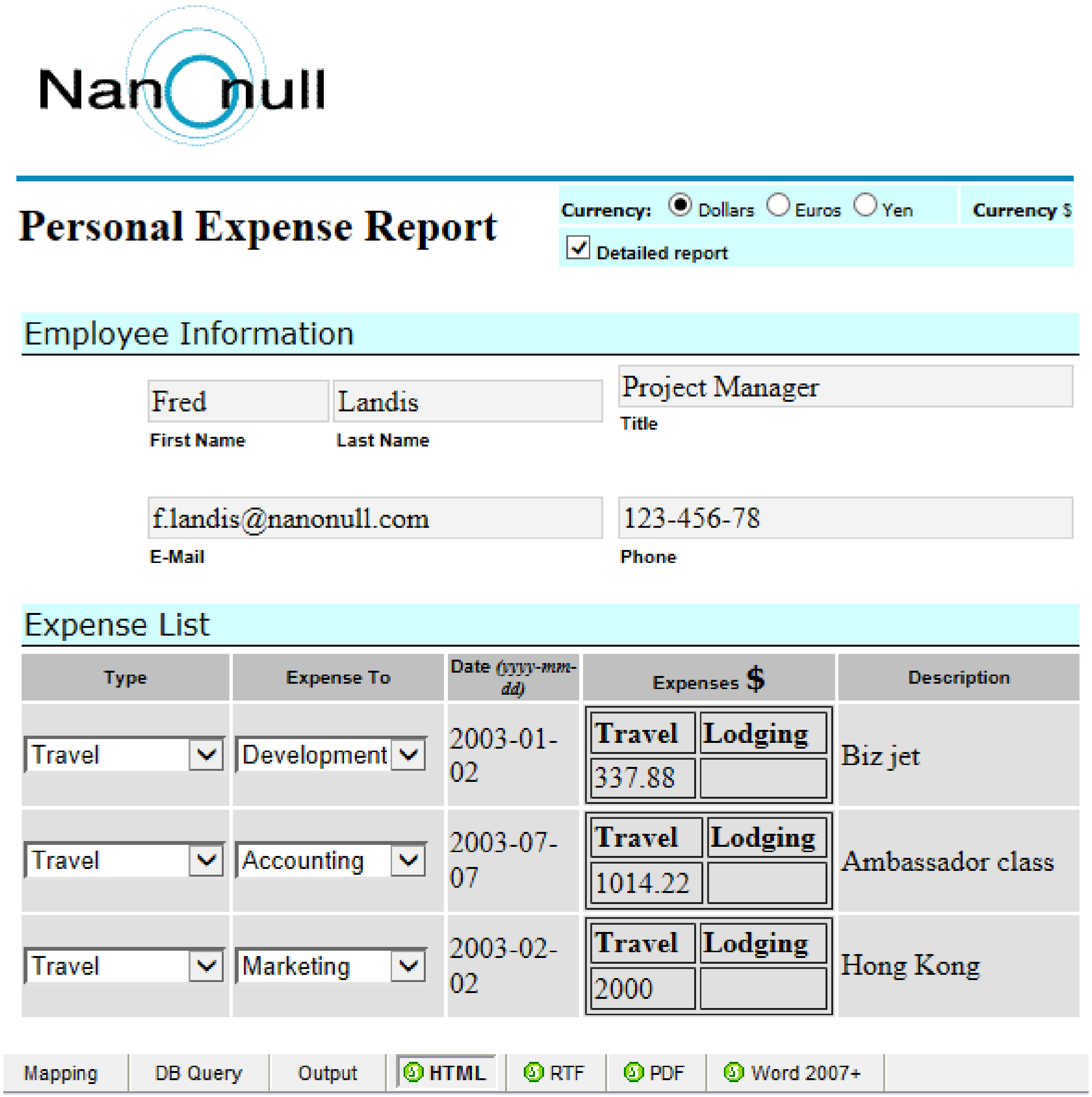
Having the same input and output file on the intermediate component is particularly important if you intend to generate code from the mapping, or run the mapping with MapForce Server. As previously mentioned, in these environments, all outputs created by each component in the mapping chain are generated. 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\MapForce2018\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):



Personal Expense Report

Currency: ● Dollars ○ Euros ○ Yen Currency $

☑ Detailed report

Employee Information

Fred	Landis	Project Manager
First Name	Last Name	Title
f.landis@nanonull.com		123-456-78
E-Mail		Phone

Expense List

Type	Expense To	Date *(yyyy-mm-dd)*	Expenses $	Description
Travel	Development	2003-01-02	Travel Lodging 337.88	Biz jet
Travel	Accounting	2003-07-07	Travel Lodging 1014.22	Ambassador class
Travel	Marketing	2003-02-02	Travel Lodging 2000	Hong Kong

HTML output of the intermediate component

5.4 Processing Multiple Input or Output Files Dynamically

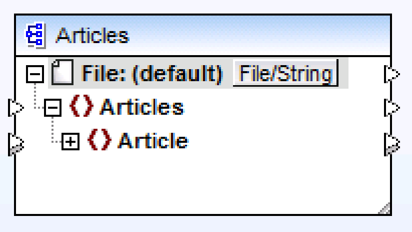
You can configure MapForce to process multiple files (for example, all files in a directory) when the mapping runs. Using this feature, you can solve tasks such as:

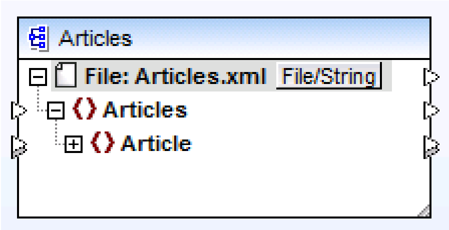
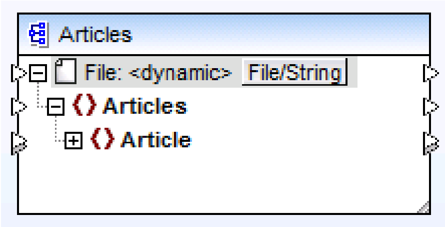
- Supply to the mapping a list of input files to be processed
- Generate as mapping output a list of files instead of a single output file
- Generate a mapping application where both the input and output file names are defined at runtime
- Convert a set of files to another format
- Split a large file (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). 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:

Use File Names from Component Settings	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.  Articles File: (default) File/String Articles Article 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. If the name of the input is the same with that of the output file, it is displayed as name of the root node.  Articles File: Articles.xml File/String Articles Article Note that you can select either this option or the *Use Dynamic File Names Supplied by Mapping* option.
Use Dynamic File Names Supplied by Mapping	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>**.  Articles File: <dynamic> File/String Articles Article This option is mutually exclusive with the *Use File Names from Component Settings* option.
Parse Strings to XML, Parse Strings to JSON, Parse Strings to CSV, Parse Strings to FLF, Parse Strings to EDI	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.
Serialize XML to Strings, Serialize JSON to Strings, Serialize CSV to Strings,	When switched on, this option enables the component to accept a structure as input, and

Serialize FLF to Strings, Serialize EDI to Strings	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.

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**

** 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)	*
XQuery	*	*(1)	Not supported by XQuery
C++	*	*	*
C#	*	*	*
Java	*	*	*
BUILT-IN	*	*	*

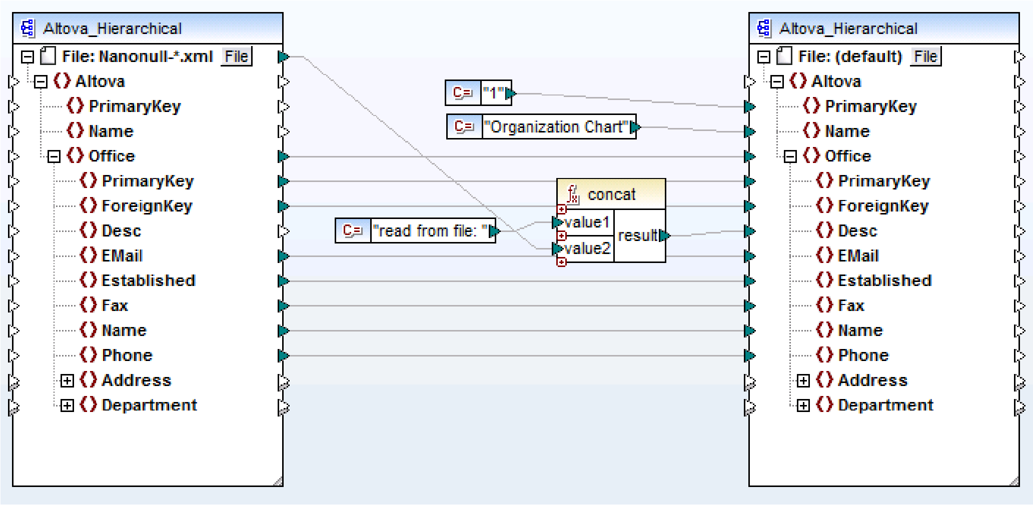
Legend:

*	Supported
(1)	Uses the `fn:collection` function. The implementation in the Altova XSLT 2.0 and XQuery engines resolves wildcards. Other engines may behave differently. For details on how to transform XSLT 1.0/2.0 and XQuery code using the RaptorXML Server engine, see Generating XSLT 1.0, or 2.0 code and Generating XQuery 1.0 code.

5.4.1 Mapping Multiple Input Files to a Single Output File

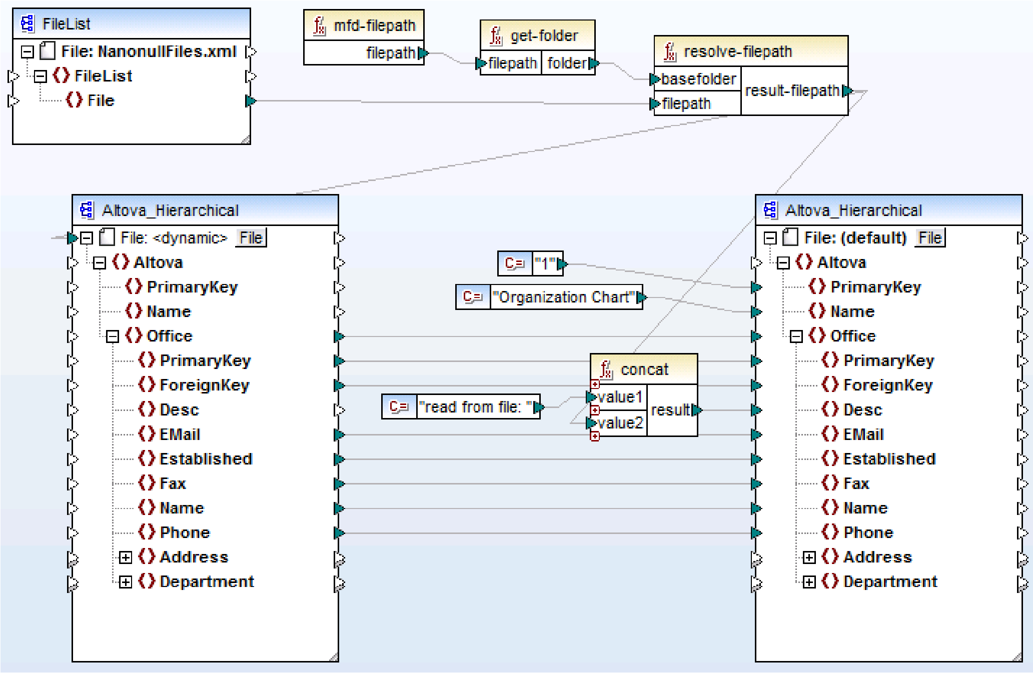
To process multiple input files, do one of the following:

- 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.

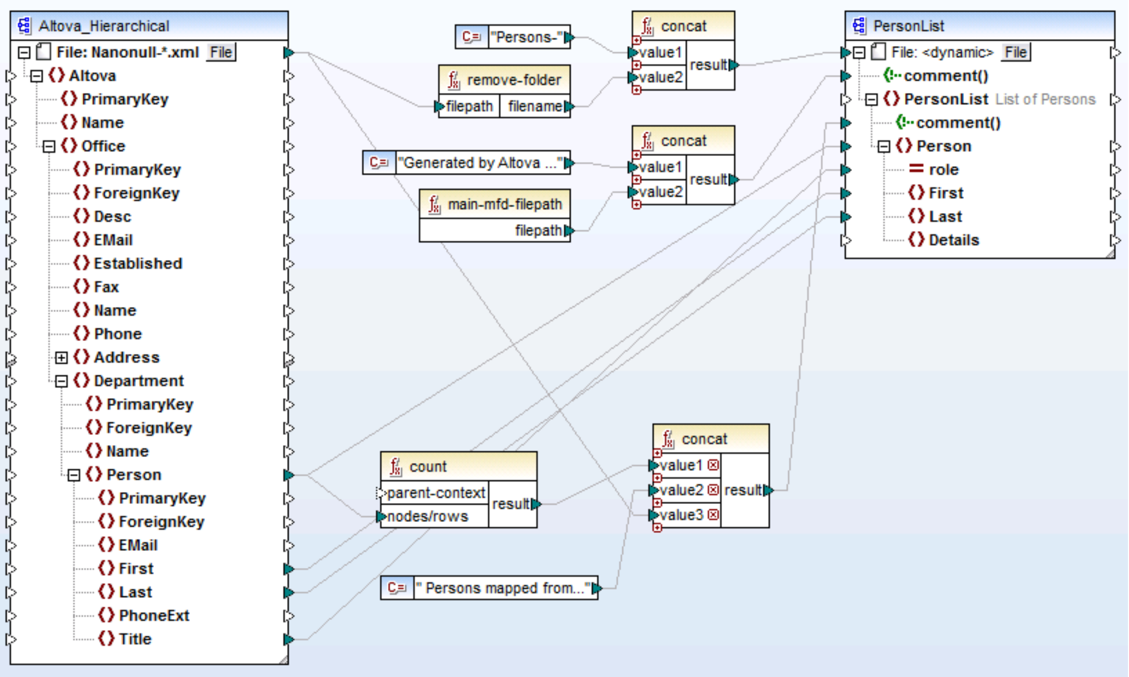


MergeMultipleFiles_List.mfd (MapForce Basic Edition)

5.4.2 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).

5.4.3 Supplying File Names as Mapping Parameters

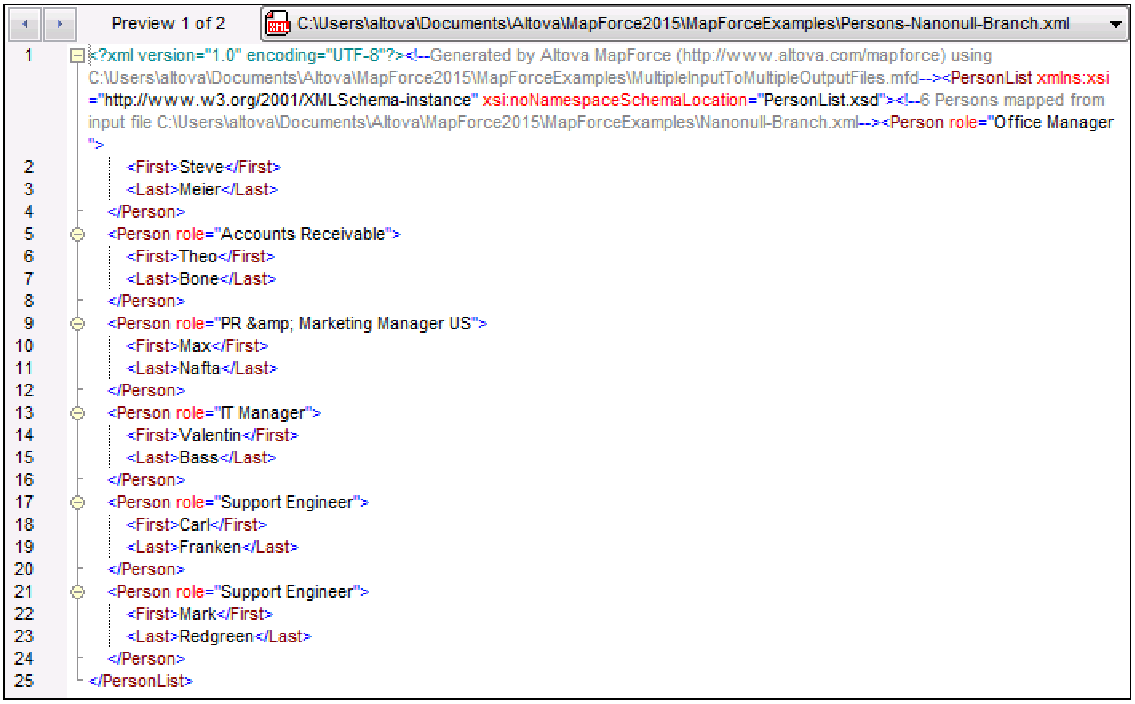
To supply custom file names as input parameters to the mapping, do the following:

1. Add an Input component to the mapping (On the **Function** menu, click **Insert Input**). For more information about such components, see Simple Input.
1. Click the **File** (File) or **File/String** (File/String) button of the source component and select **Use Dynamic File Names Supplied by Mapping**.
2. Connect the 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.

5.4.4 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.



Preview 1 of 2
C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Persons-Nanonull-Branch.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

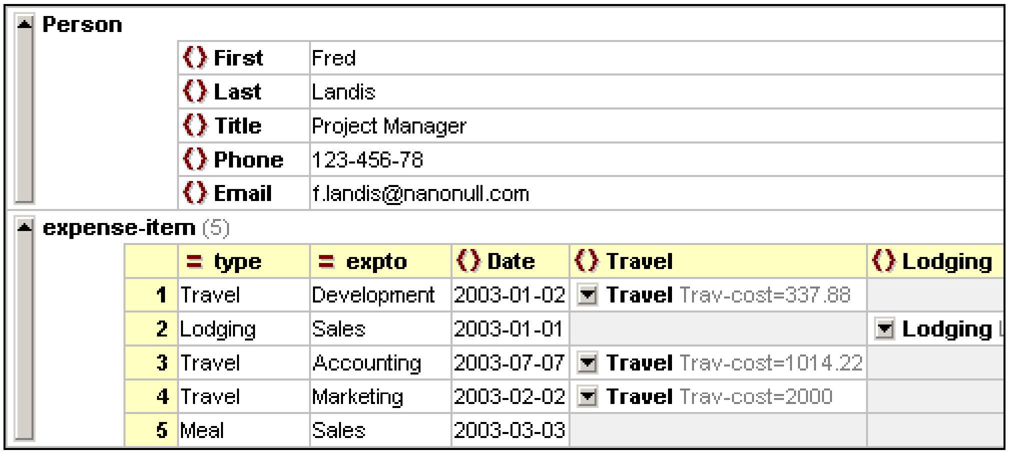
To save the generated output files, do one of the following:

- On the **Output** menu, click **Save All Output Files** (🗐).
- Click the **Save all generated outputs** (🗐) toolbar button.

5.4.5 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\MapForce2018\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
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		

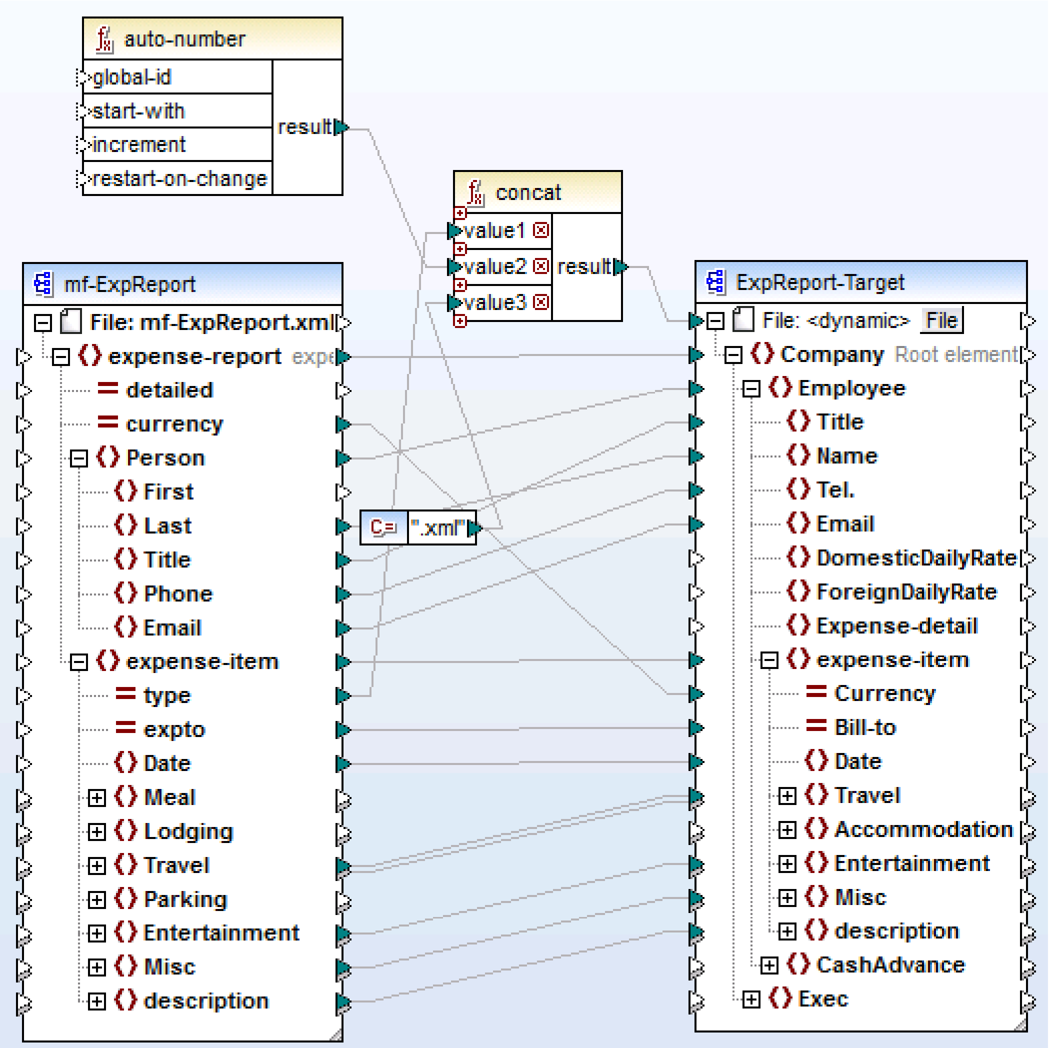
mf-ExpReport.xml (as shown in XMLSpy Grid view)

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).

1. 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.



Tut-ExpReport-dyn.mfd (MapForce Basic Edition)

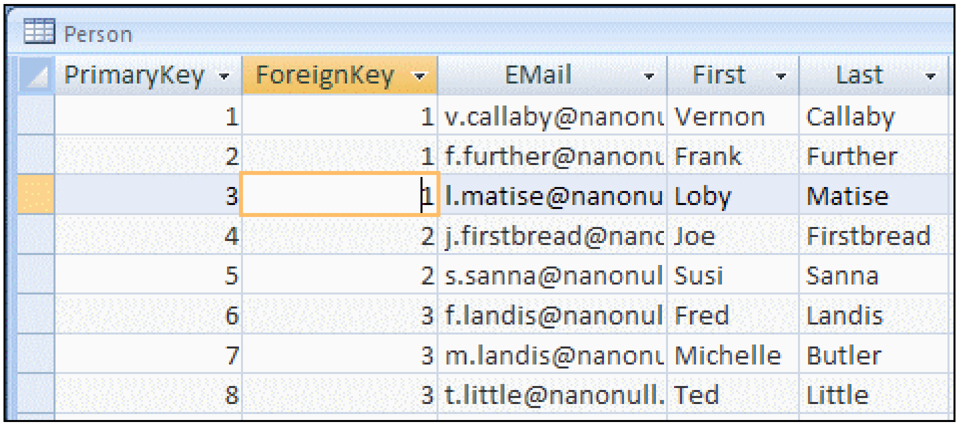
Note that the resulting output files are named dynamically as follows:

- 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.

5.4.6 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\MapForce2018\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.

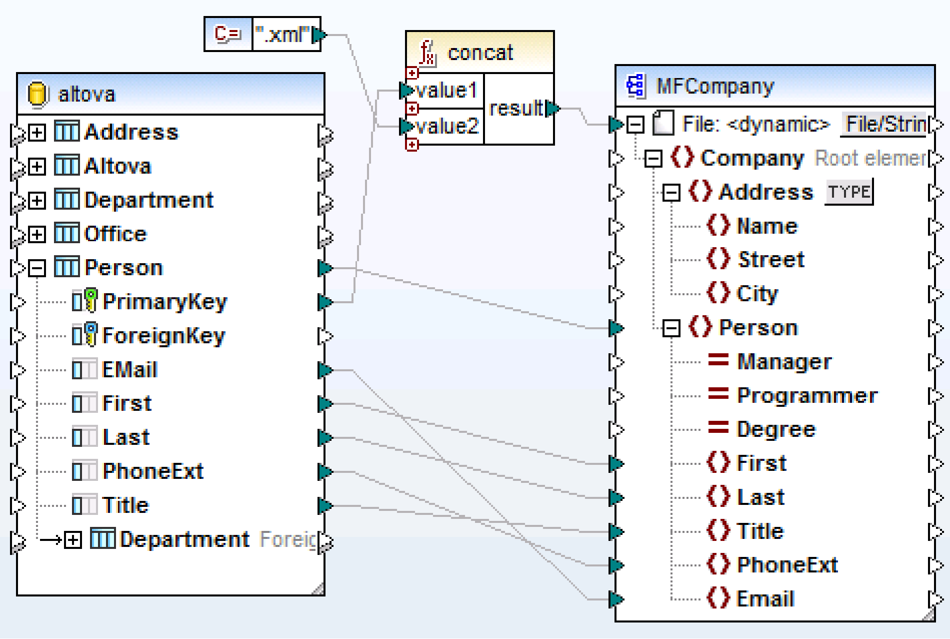


Person

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.

5.5 Supplying Parameters to the Mapping

You can pass simple values to a mapping by means of simple input components. On the mapping area, simple input components play the role of a source component which has a simple data type (for example, string, integer, and so on) instead of a structure of items and sequences. Consequently, you can create a simple input component instead of (or in addition to) a file-based source component.

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
- 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 Input Component Settings). If necessary, you can also create default values for the mapping input parameters (see Creating a Default Input Value). This enables you to safely run the mapping even if you do not explicitly supply a parameter value at mapping execution time.

Input parameters added on the main mapping area should not be confused with input parameters in user-defined functions (see User-defined functions). 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.

For an example, see Example: Using File Names as Mapping Parameters.

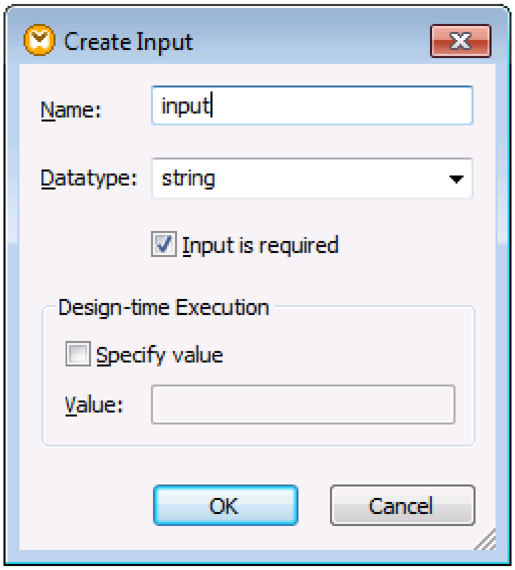
5.5.1 Adding Simple Input Components

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. On the **Function** menu, click **Input**.
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.

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**.


Create Input

Name: input

Datatype: string

☑ Input is required

Design-time Execution

☐ Specify value

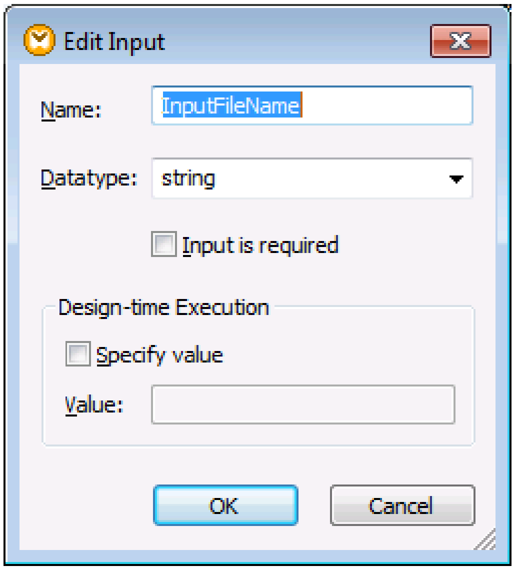
Value:

Create Input dialog box

You can change later any of the settings defined here (see Simple Input Component Settings).

5.5.2 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.



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**.

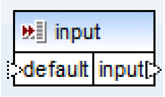
The available settings are as follows.

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). Disable this check box if you want to specify a default value for the input parameter (see Creating a Default Input Value).
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.

5.5.3 Creating a Default Input Value

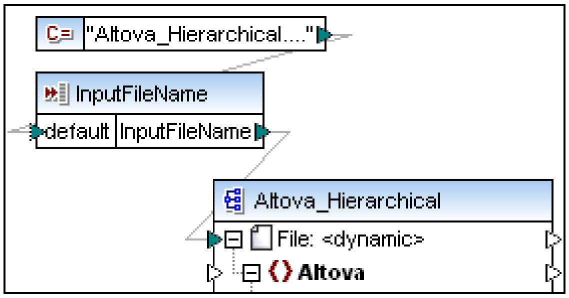
After you add an Input component to the mapping area, notice the **default** item to the left of the component.



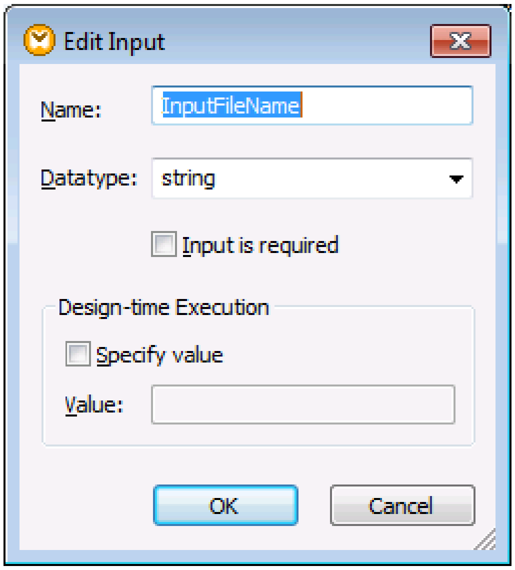
Simple input component

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 make sure that the **Input is required** check box is disabled. When you create a default input value, this setting is not meaningful and causes mapping validation warnings.



Name: InputFileName

Datatype: string

☐ Input is required

Design-time Execution

☐ Specify value

Value:

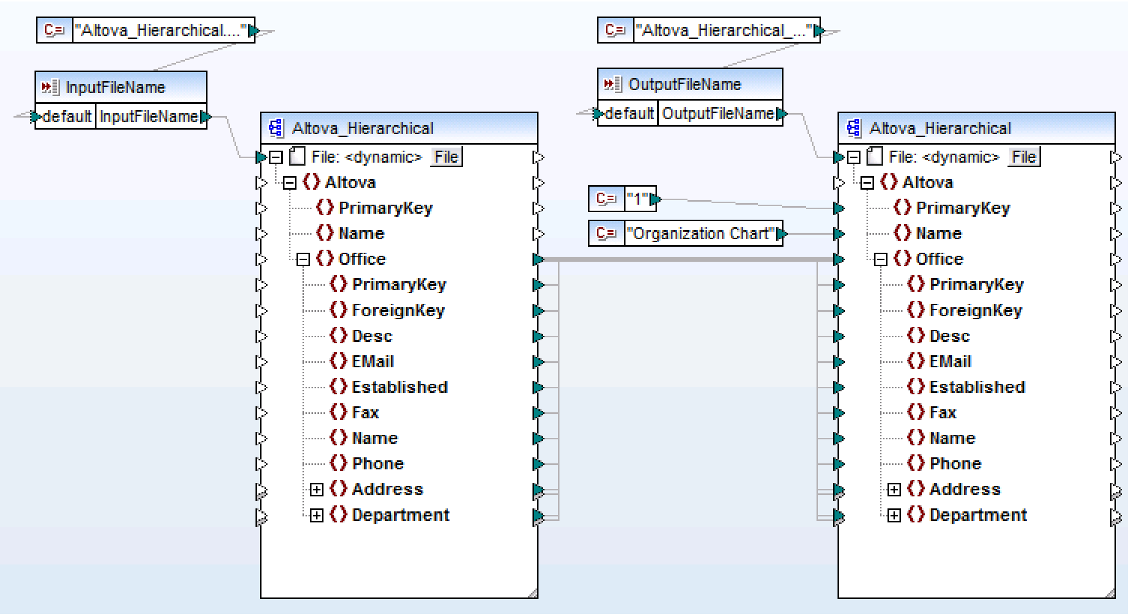
3. Click **OK**.

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 same value has no effect in the generated code.

5.5.4 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\MapForce2018\MapForceExamples \FileNamesAsParameters.mfd**.

The mapping uses 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.



FileNamesAsParameters.mfd (MapForce Basic Edition)

Both the **InputFileName** and **OutputFileName** components are simple input components in the mapping, so you can supply them as input parameters when executing the mapping. The following sections illustrate how to do this in the following transformation languages:

- XSLT 2.0, using RaptorXML Server
- Built-in (MapForce Server Execution File), using MapForce Server
- Java

XSLT 2.0

If you generate code in XSLT 1.0 or XSLT 2.0, the input parameters are written to the

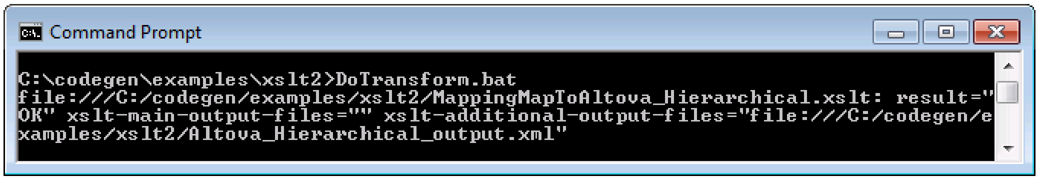
DoTransform.bat batch file, for execution by RaptorXML Server (see Automation with RaptorXML Server). To use a different input (or output) file, you can either pass the required parameters at command line, when calling the **DoTransform.bat** file, or edit the latter to include the required parameters.

To supply a custom input parameter in the **DoTransform.bat** file:

1. Generate the XSLT 2.0 code (**File | Generate Code In | XSLT 2.0**) from the **FileNamesAsParameters.mfd** sample.
2. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2018\MapForceExamples\** directory to the directory where you generated the XSLT 2.0 code (in this example, **c:\codegen\examples\xslt2\**). This file will act as custom parameter.
3. Edit **DoTransform.bat** to include the custom input parameter either before or after `%*` (as highlighted below). Note that the parameter value is enclosed with single quotes. The available input parameters are listed in the **rem** (Remark) section.

```
@echo off

RaptorXML xslt --xslt-version=2 --
input="MappingMapToAltova_Hierarchical.xslt" --
param=InputFileName:'Altova_Hierarchical.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 parameter.



```
C:\codegen\examples\xslt2>DoTransform.bat
file:///C:/codegen/examples/xslt2/MappingMapToAltova_Hierarchical.xslt: result="
OK" xslt-main-output-files="" xslt-additional-output-files="file:///C:/codegen/e
xamples/xslt2/Altova_Hierarchical_output.xml"
```

MapForce Server Execution File

To supply custom input parameters to a MapForce Server execution file:

1. Compile the **FileNamesAsParameters.mfd** to a MapForce Server execution file (see Compiling Mappings to MapForce Server Execution Files). When prompted, save the .mfx execution file to a directory on your computer (in this example, **c:\codegen\examples\mfx\**).
2. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2018\MapForceExamples\** directory to the directory where you saved the .mfx file. This file will act as the custom parameter supplied to the mapping execution file.
3. 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 path every time when you run a mapping, add to your operating system's PATH environment variable the path of the directory where MapForce Server executable is installed (for example, **C:\Program Files (x86)\Altova\MapForceServer2018\bin**) .

Java

To supply a custom input parameter to a Java .jar application:

1. Generate the Java code (**File | Generate Code In | Java**) from the **FileNamesAsParameters.mfd** sample.
2. Compile the Java code into an executable JAR file (for instructions on how to do this in Eclipse, see Example: Build a Java application with Eclipse and Ant).
3. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2018\MapForceExamples\** directory to the directory where the .jar file is. This file will act as the custom parameter supplied to the Java mapping application.
4. At the command line, enter: `java -jar Mapping.jar /InputFileName "InputFile.xml"`

> If you use wildcards when passing parameters to .jar files, place the wildcard parameters in quotes, for example:
>
> ```
> java -jar Mapping.jar /InputFileName "altova-*.xml"
> ```

5.6 Returning String Values from a Mapping

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). 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). 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), 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 **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	If the generated XSLT files, a simple output components defined in the mapping becomes the output of the XSLT transformation.
	If you are using RaptorXML Server, you can instruct RaptorXML Server to write the mapping output to the file passed as value to the `--output` parameter.
	To write the output to a file, add or edit to the `--output` 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).
	`RaptorXML xslt --xslt-version=2 --`
	`input="MappingMapToResult1.xslt" --`
	`output="Output.txt" %* "MappingMapToResult1.xslt"`
	If an `--output` parameter is not defined, the mapping output will be written to the standard output stream (stdout) when the mapping is executed.

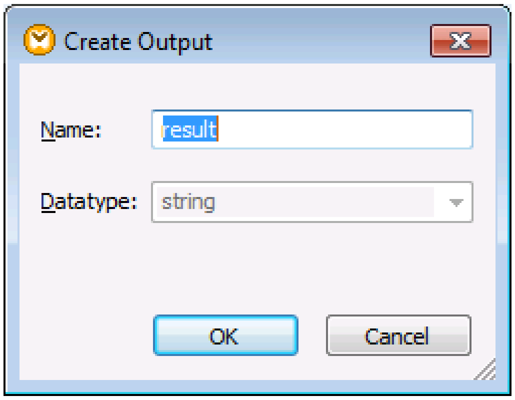
C++, C#, Java	In the generated C++, C#, and Java code, the mapping output is written to the standard output of the generated application. If the mapping contains multiple target components, the generated application concatenates the standard output of each target component and returns it as one unified standard output.

When you create a reversed mapping (using the menu command **Tools \| Create Reversed Mapping**), the simple output component becomes a simple input component.

5.6.1 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. On the **Function** menu, click **Output**.
3. Enter a name for the component.
4. Click **OK**.



Name: result

Datatype: string

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**.

5.6.2 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 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` 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` 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".

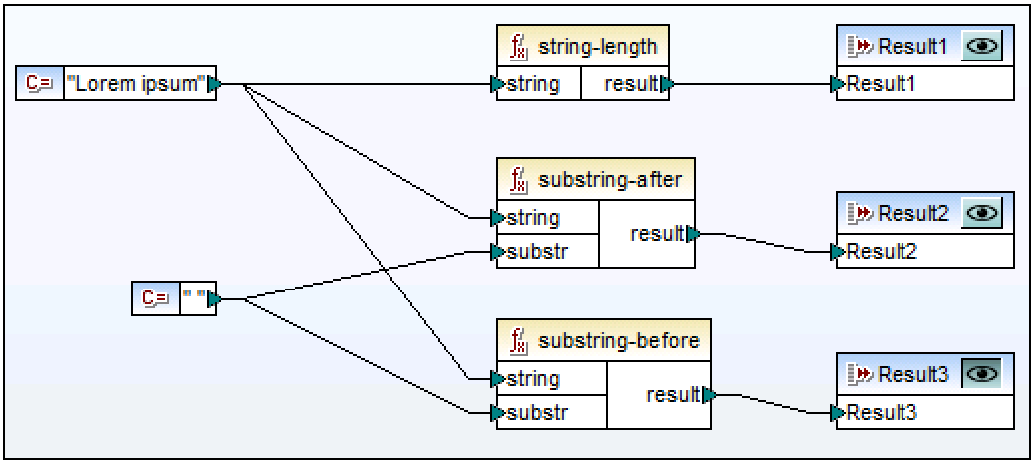
`substring-before` Returns the part of the string that occurs before the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "Lorem".

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.

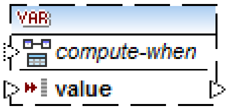
To preview the output of any function

- Click the **Preview** (👁) button in the component title bar, and then click the **Output** tab on the Mapping window.

5.7 Using Variables

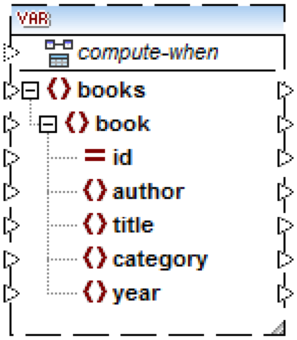
Variables are a special type of component used to store an intermediate mapping result for further processing. They might be necessary in situations where you want to temporarily "remember" some data on the mapping and process it in some way (for example, filter it, or apply some functions) before it is copied to the target component.

Variables can be of simple type (for example, string, integer, boolean, etc) or complex type (a tree structure).



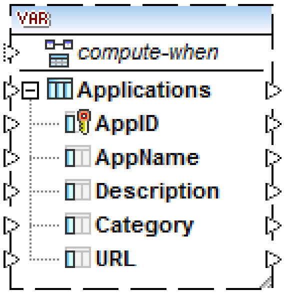
Simple variable

You can create a variable of complex type by supplying an XML schema which expresses the structure of the variable. 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 any associated instance XML file; the data of the variable is computed at mapping runtime.



Complex variable created from an XML schema

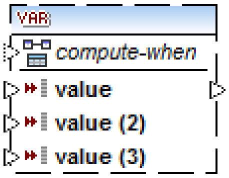
It is also possible to create variables of complex type from databases. In case of databases, you can choose a specific database table as root item for the variable structure.



Complex variable created from a database table

In the images above, you may notice that 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 on the mapping (see Changing the Context and Scope of Variables).

When necessary, items of a variable structure can be duplicated to accept data from more than one source connection, similar to how this is done for standard components (see Duplicating Input). This does not apply, however, to variables created from database tables.



Simple variable with duplicated inputs

One of the most important things about variables is that they are sequences, and can be used to create sequences. The term "sequence" here means a list of zero or more items (see also Mapping Rules and Strategies). This makes it possible for a variable to process multiple items for the duration of the mapping lifetime. If, however, you want to assign a value once to a variable and keep it the same for the rest of the mapping, it is also possible (see Changing the Context and Scope of Variables).

To some extent, variables can be compared to intermediate components of a chained mapping (see Chained Mappings). However, they are more flexible and convenient if you don't need to produce intermediary files at each stage in the mapping. The following table outlines differences between variables and chained mappings.

Chained mappings	Variables
Chained mappings involve two totally independent steps. For example, let's assume a mapping that has three components A, B, and C. Running the mapping involves two stages: executing the mapping from A to B, and then executing the mapping from B to C.	While the mapping is executed, variables are evaluated according to their context and scope. Their context and scope can be influenced (see Changing the Context and Scope of Variables).

Chained mappings	Variables
When the mapping is executed, intermediate results are stored externally in files.	When the mapping is executed, intermediate results are stored internally. No external files containing a variable's results are produced.
The intermediate result can be previewed using the preview button.	A variable's result cannot be previewed, since it is computed at mapping runtime.

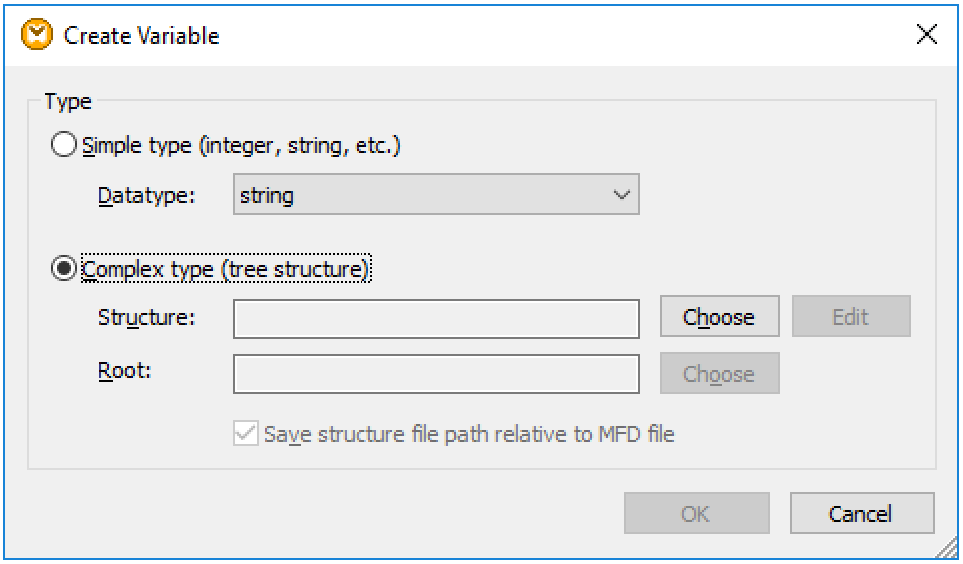
Note: Variables are not supported if the mapping transformation language is set to XSLT 1.0.

5.7.1 Adding Variables

There are several ways to add variables to a mapping, as shown below.

Using a menu or toolbar command

1. On the **Insert** menu, click **Variable**. (Alternatively, click the **Variable** toolbar button).



Create Variable

Type

○ Simple type (integer, string, etc.)

Datatype: string

● Complex type (tree structure)

Structure:

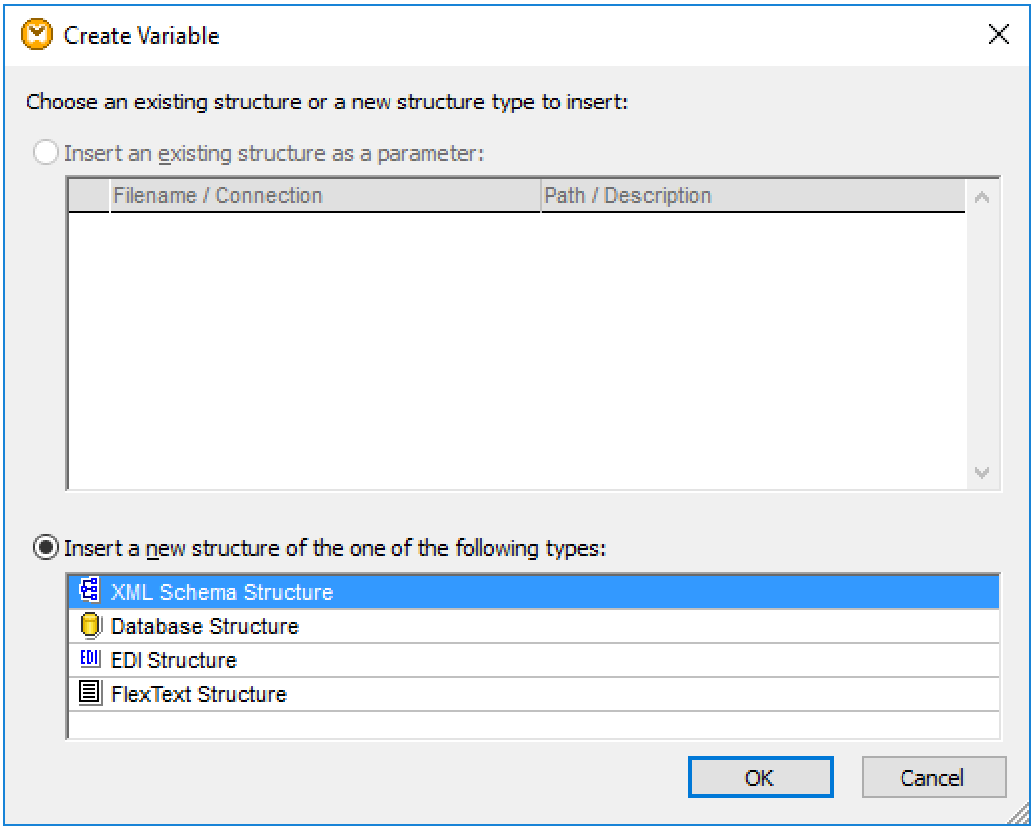
Root:

☑ Save structure file path relative to MFD file

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 (for example, an XML Schema or database).



Create Variable

Choose an existing structure or a new structure type to insert:

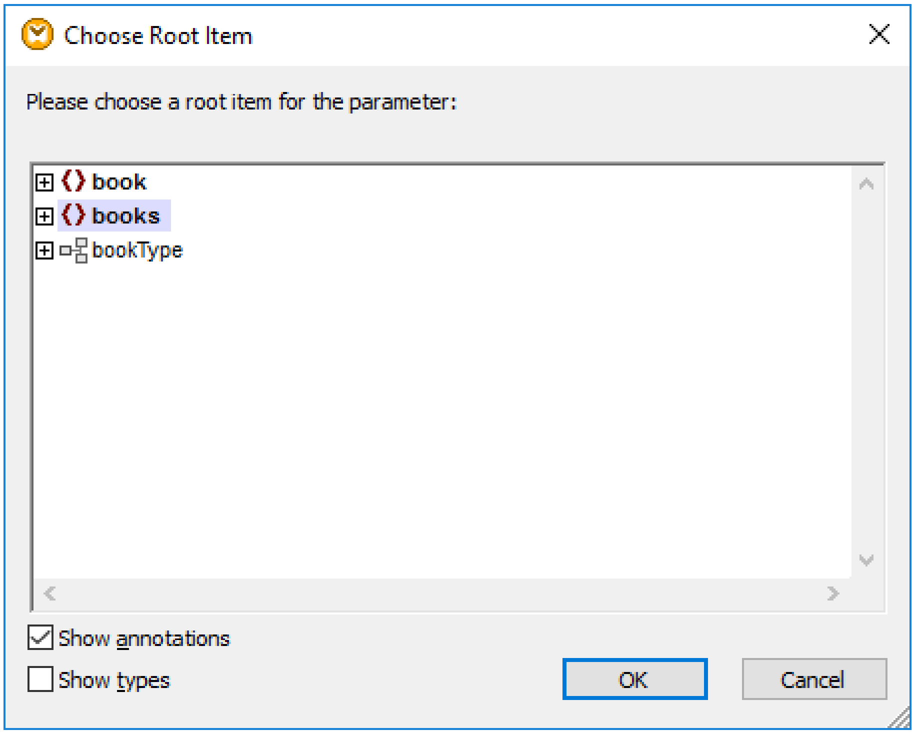
○ Insert an existing structure as a parameter:

Filename / Connection	Path / Description

● Insert a new structure of the one of the following types:

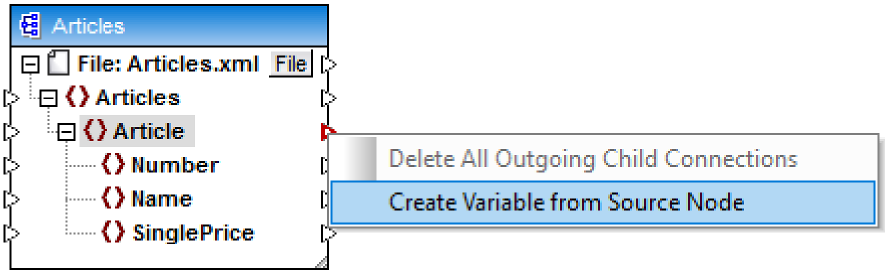
- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure

4. When prompted, specify a root item of the structure. In case of XML Schemas, the root item can be any element defined globally. In case of databases, the root item can be any table.

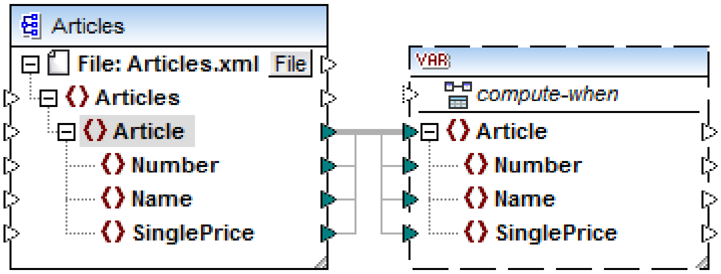


Using a context menu

- Right-click the output connector of a component (in this example, "Article") and select **Create Variable from Source node**.



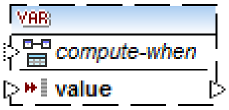
This creates a complex variable using the same source schema and automatically connects all items with a Copy-All connection.



- Right-click the input connector of a target component and select **Create Variable for Target Node**. This creates a complex variable using the same schema as the target, and automatically connects all items with a Copy-All connection.
- 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 using the source schema, and automatically uses the item linked to the filter input as the root element of the intermediate component.

5.7.2 Changing the Context and Scope of Variables

Every variable has a `compute-when` input item which allows you to control the scope of the variable; in other words, 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 "compute-when" item

In the following text, a *subtree* means the 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* means 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* (either 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 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.

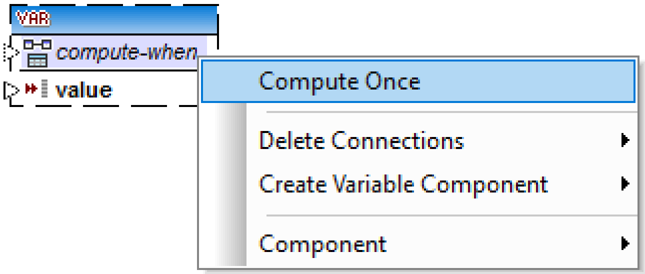
"Compute-when" is connected

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 relates to the general rule governing connections in MapForce: "for each source item, create one target item". With `compute-when`, it means "for each source item, compute the variable value" (see Mapping Rules and Strategies).

"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:



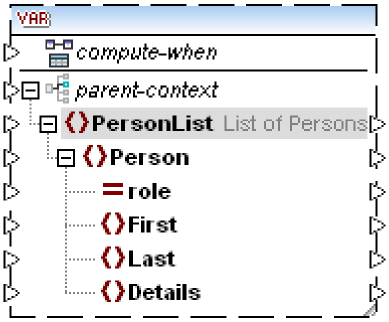
When you change the scope of a variable to `compute-when=once`, the input connector is removed from the `compute-when` item, since such a variable is only evaluated once.

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

Adding a parent-context may be necessary, for example, if your mapping uses multiple filters and you need an additional parent node to iterate over, see also Overriding the Mapping Context.

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.

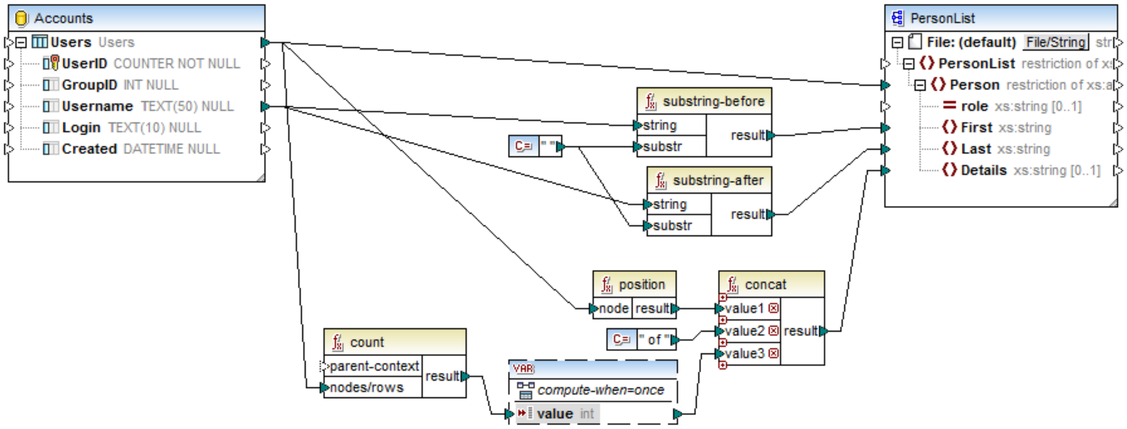


The parent context adds a virtual "parent" node to the hierarchy within the component. This allows you to iterate over an additional node in the same, or in a different source component.

5.7.3 Example: Counting Database Table Rows

The mapping illustrated in this example is available as **DB_UserList.mfd** in the **<Documents>\Altova\MapForce2018\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.

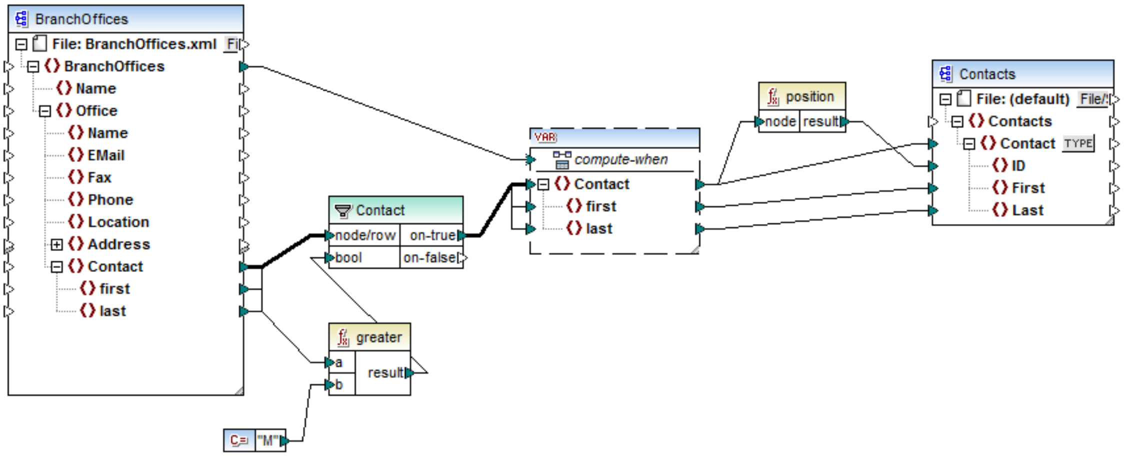
The value of the field "Username" is supplied to the **`substring-before`** and **`substring-after`** functions. These two functions extract the text before and after the space character (" "), respectively, which takes care of the second mapping goal.

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)

5.7.4 Example: Filtering and Numbering Nodes

The mapping illustrated in this example is available as **PositionInFilteredSequence.mfd** in the **<Documents>\Altova\MapForce2018\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)
- A complex variable (see Adding Variables)
- The functions `greater` and `position` (see Working with Functions)
- A constant (To add a constant, select the menu command **Insert | Constant**).

The variable uses the same schema as the source component. If your 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 first name is equal or greater than M". To achieve this, the function `greater` 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.

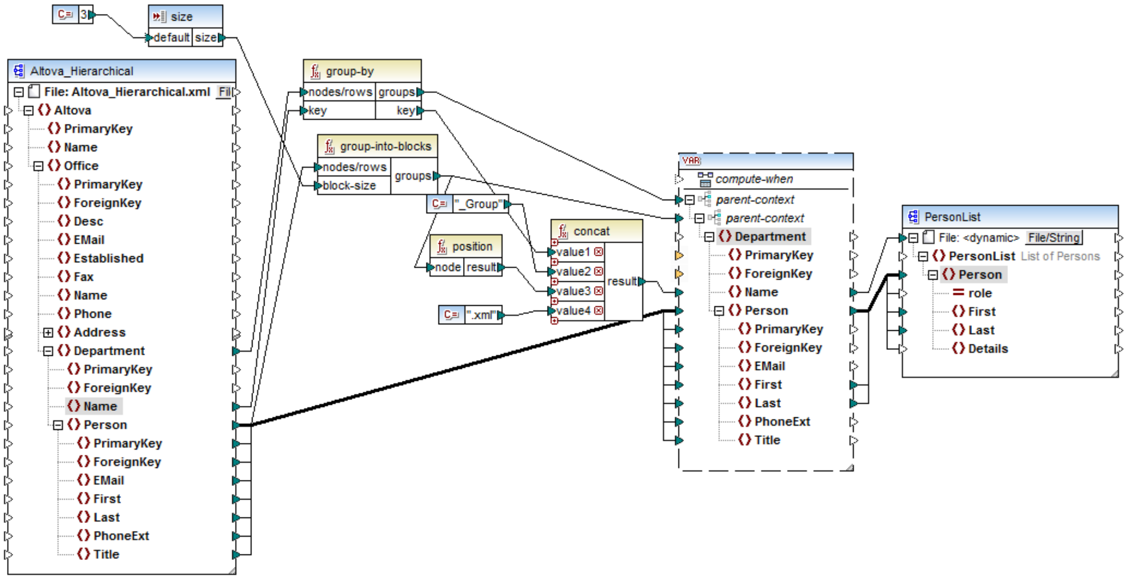
The `position` functions returns, for each iteration of the variable, the number of the current sequence. Only eight contacts meet the filter condition; therefore, if you preview the mapping and look at the output, notice how IDs 1 through 8 were written to the `ID` element of the target component.

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.

5.7.5 Example: Grouping and Subgrouping Records

The mapping illustrated in this example is available as **DividePersonsByDepartmentIntoGroups.mfd** in the **<Documents>\Altova\MapForce2018\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).

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`** 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`** 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.

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`** 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"

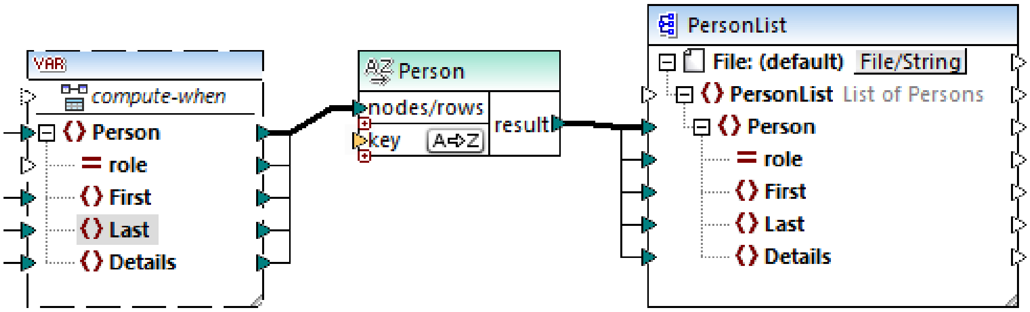
The result of this concatenation is stored in the `Name` item of the variable, and then supplied as a dynamic file name to the target component. This causes a new file name to be created for each received value. In this example, the variable computes eight groups in total, so eight output files are created when the mapping runs, as required. For more information about this technique, see Processing Multiple Input or Output Files Dynamically.

5.8 Sorting Data

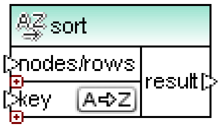
To sort input data based on a specific sort key, use a Sort component. The Sort component supports the following target languages: XSLT2, XQuery, and Built-in. When the transformation language is "Built-in", the Sort component can be used to sort database table data. Better performance is, however, achieved using an SQL-WHERE/ORDER component. For more details, see Filtering and Sorting Database Data (SQL WHERE/ORDER).

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).



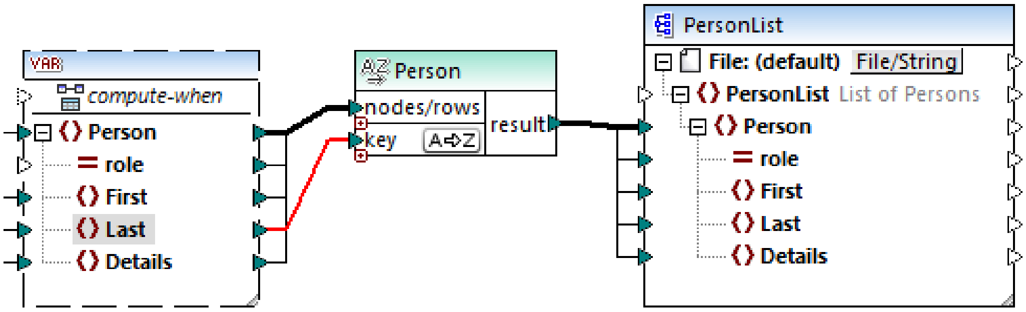
- On the **Insert** menu, click **Sort** (alternatively, click the **Sort** toolbar button). This inserts the Sort component in its "unconnected" form.



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`.

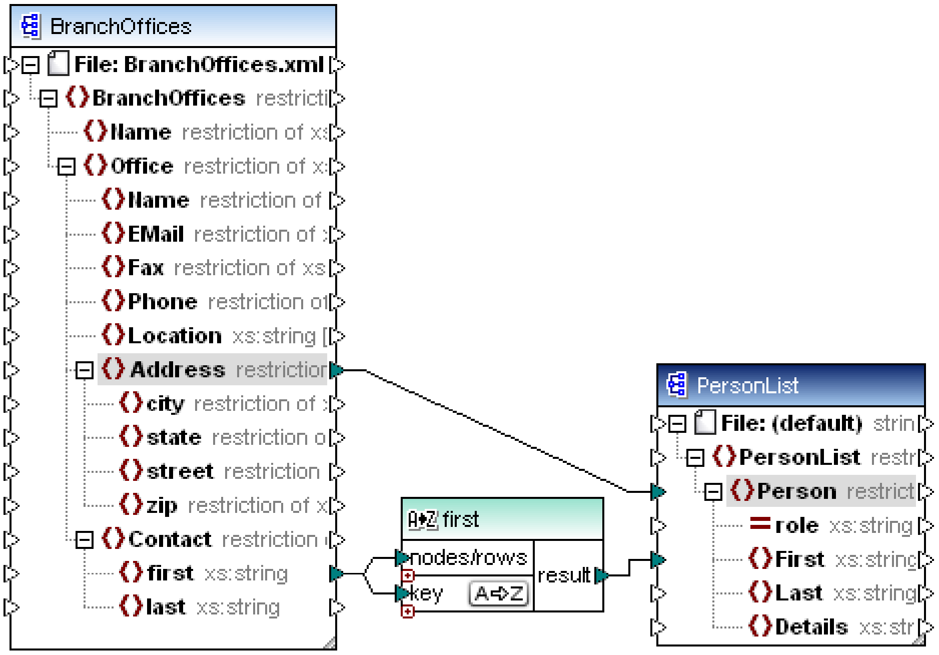


To change the sort order:

- Click the A⇨Z icon in the Sort component. It changes to Z➔A to show that the sort order has been reversed.

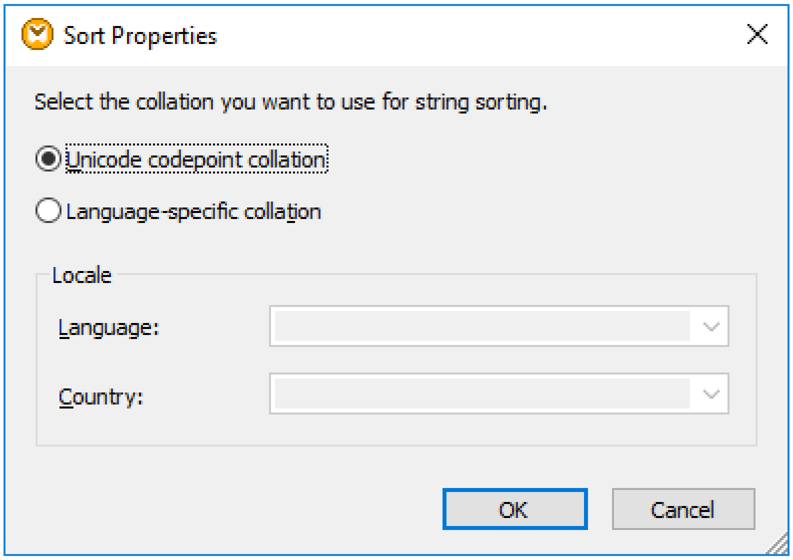
To sort input data consisting of simple type items:

- 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.



Sort Properties

Select the collation you want to use for string sorting.

● Unicode codepoint collation

○ Language-specific collation

Locale

Language:

Country:

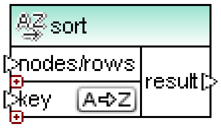
OK Cancel

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.

5.8.1 Sorting by Multiple Keys

After you add a Sort component to the mapping, one sorting key called key is created by default.

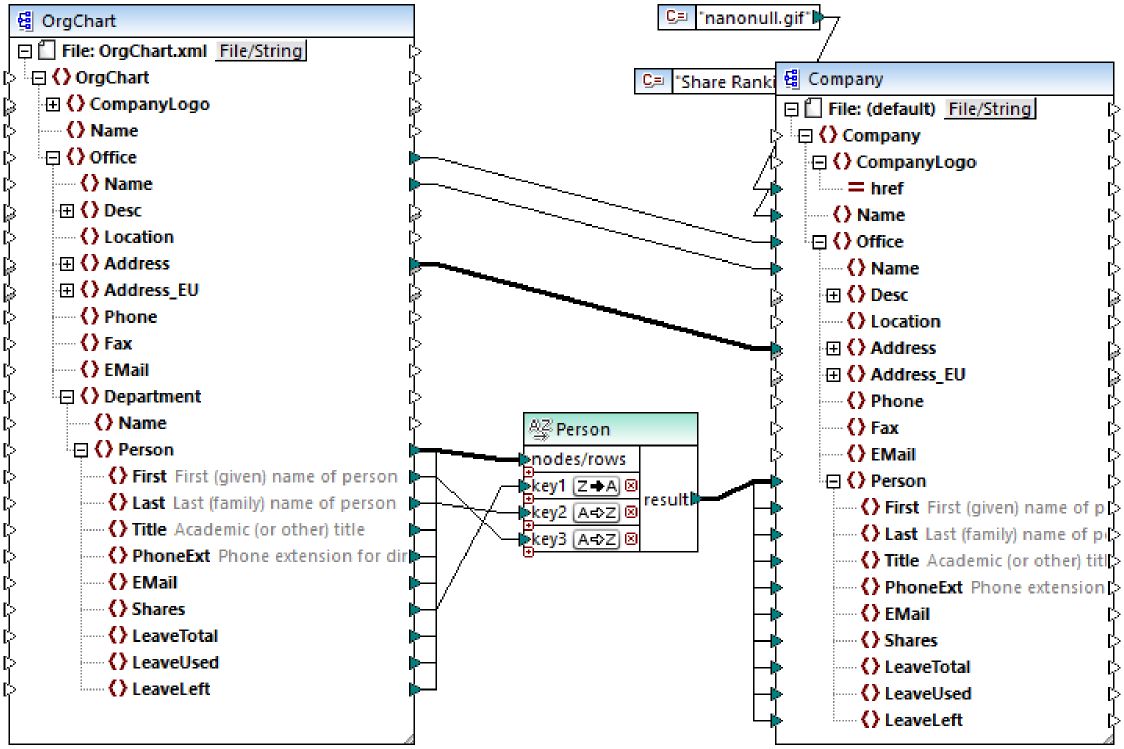


Default Sort component

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\MapForce2018\MapForceExamples\SortByMultipleKeys.mfd**.



SortByMultipleKeys.mfd

In the mapping above, `Person` records are sorted by three sorting keys:

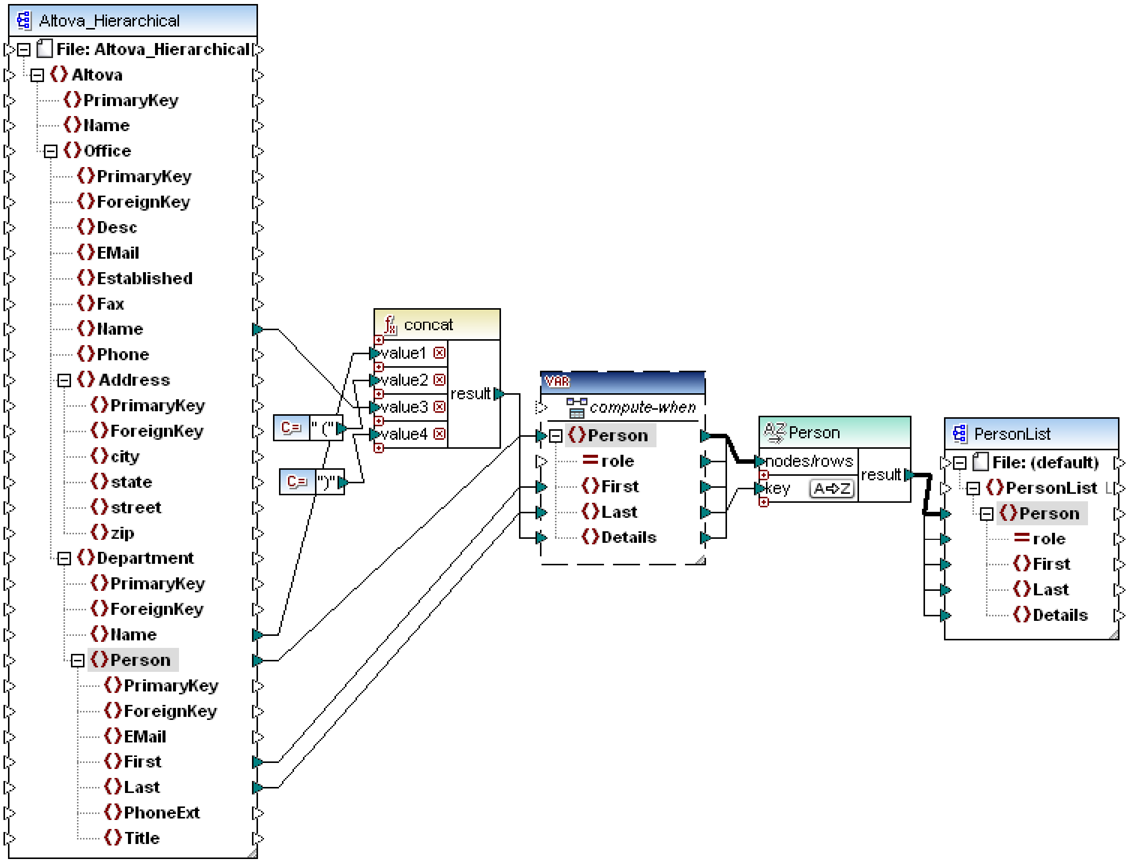
1. `Shares` (number of shares a person holds)
2. `Last` (last name)
3. `First` (first name)

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).

5.8.2 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\MapForce2018\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 Working with Functions.
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). 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 and Copy-All Connections.

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.

5.9 Filters and Conditions

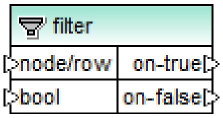
When you need to filter data, or get a value conditionally, you can use one of the following component types:

- 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:



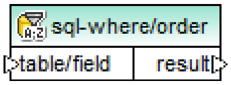
In the structure above, the condition connected to **bool** determines whether the connected **node/row** goes to the **on-true** or **on-false** output. Namely, if the condition is true, the **node/row** will be redirected the **on-true** output. Conversely, if the condition is false, the **node/row** 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).

For a step-by-step mapping example, see Example: Filtering Nodes.

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.

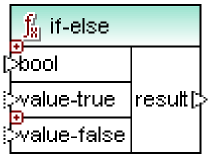
Returning a value conditionally

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>
```

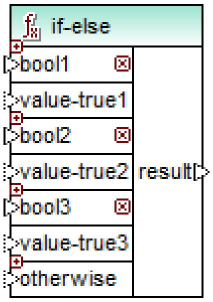
An **If-Else Condition** would enable you to return, for each item in the list, the value "high" if temperature exceeds 20 degrees Celsius, and value "low" if temperature is lower than 5 degrees Celsius.

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.



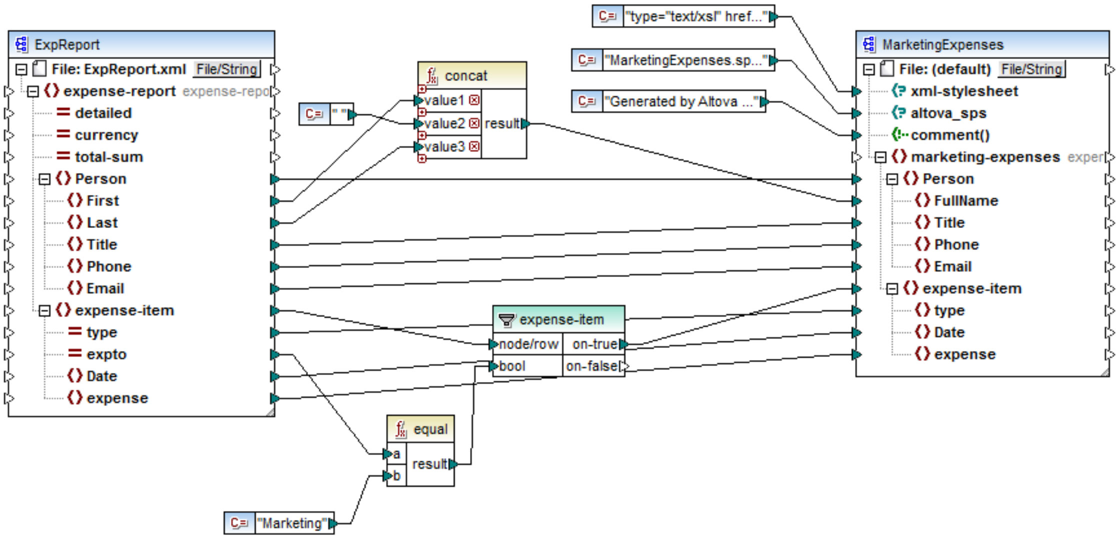
Expanded **If-Else Conditions** are evaluated from top to bottom (first conditions is checked first, then the second one, and so on). If you want to return a value when none of the conditions are true, connect it to **otherwise**.

For a step-by-step mapping example, see Example: Returning a Value Conditionally.

5.9.1 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).

The mapping described in this example is available at the following path: **<Documents>\Altova \MapForce2018\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 (for more information, see Working with Functions). The **`equal`** function compares the value of the "type" 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>
```

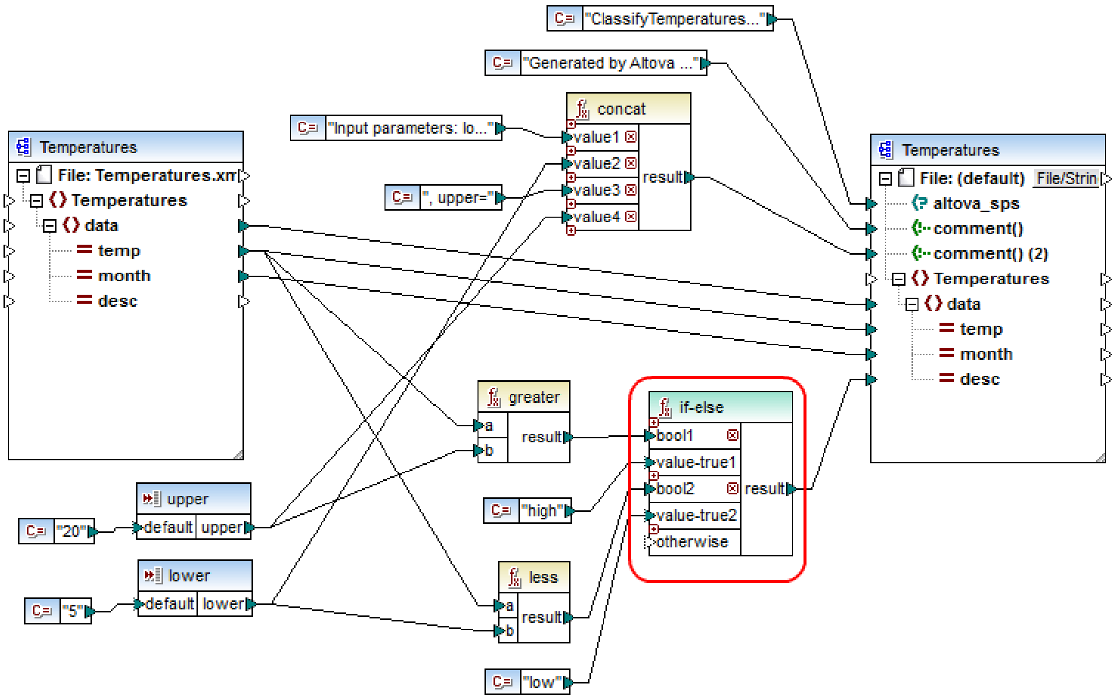
```
        <Date>2003-01-01</Date>
        <expense>122.11</expense>
    </expense-item>
    <expense-item>
        <type>Lodging</type>
        <Date>2003-01-02</Date>
        <expense>299.45</expense>
    </expense-item>
...
```

XML output after the mapping is executed

5.9.2 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).

The mapping described in this example is available at the following path: **<Documents>\Altova\MapForce2018\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 (for more information, see Working with Functions). 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 Mapping.)

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"/>
...
```

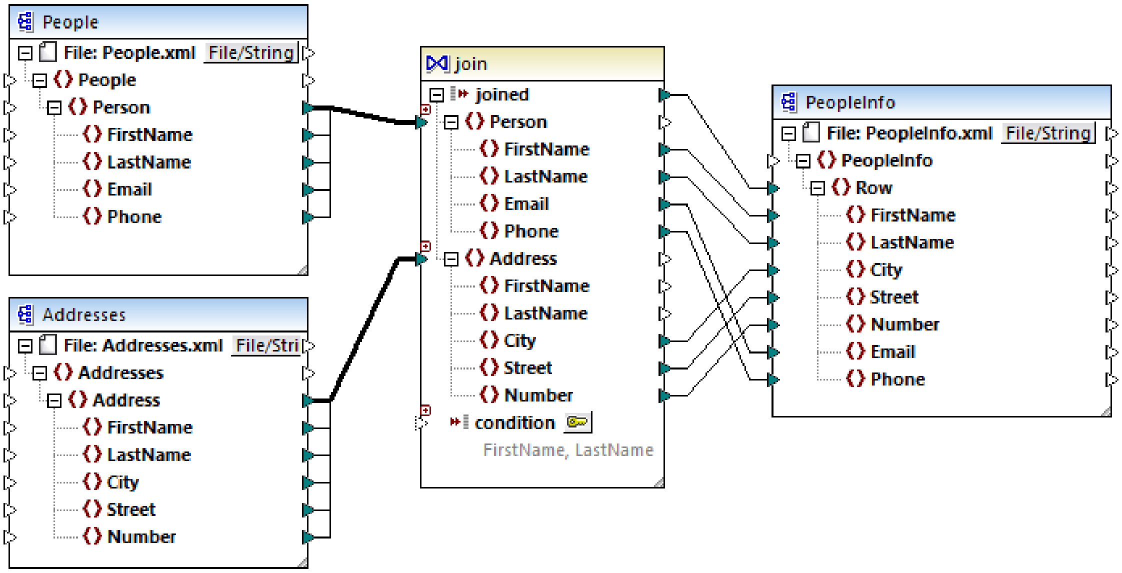
XML output after the mapping is executed

5.10 Joining Data

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.



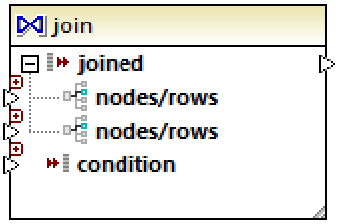
JoinPeopleInfo.mfd

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).

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.

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.



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.

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).
- 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 About Joins in SQL Mode).
- 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 and Filters and Conditions, 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).

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 Overriding the Mapping Context.

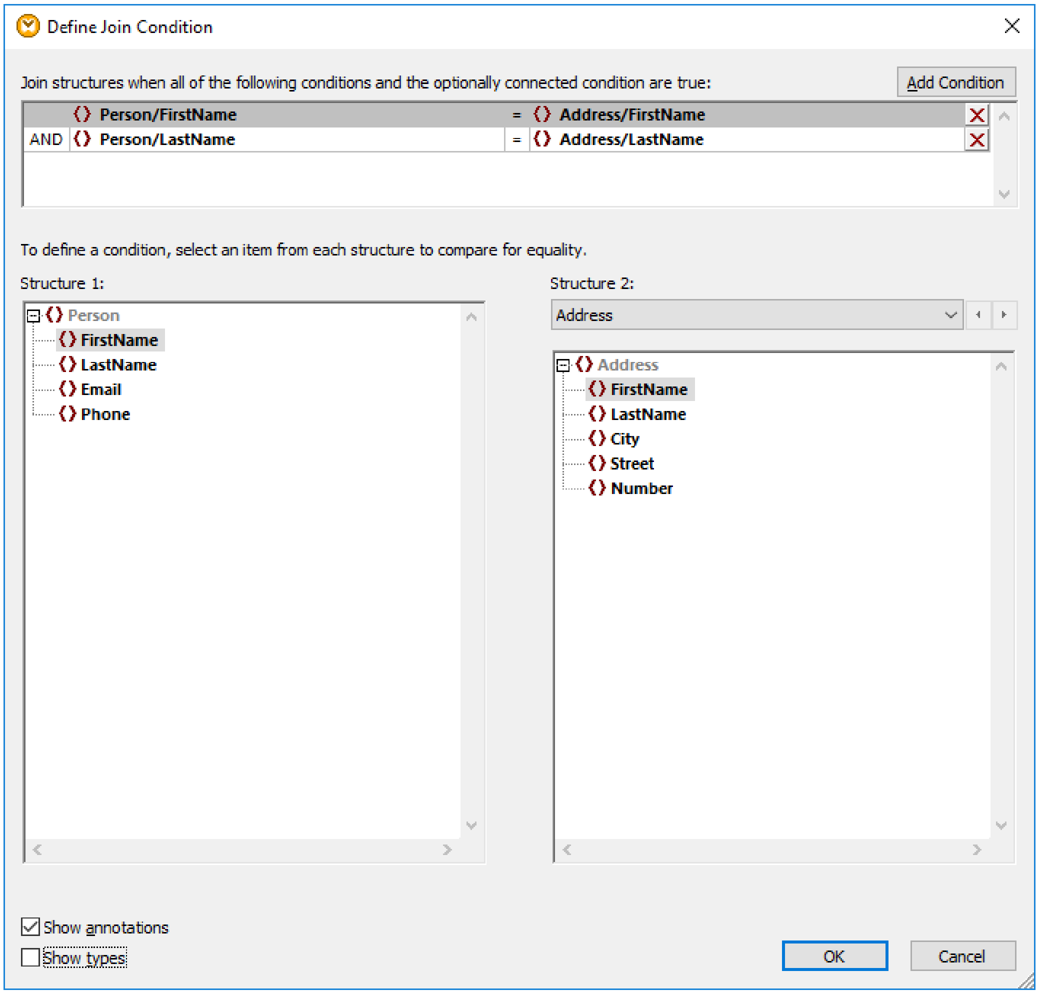
5.10.1 Adding Join Conditions

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.

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\MapForce2018\MapForceExamples\Tutorial\**. This mapping is discussed in more detail in Example: Join XML Structures.
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).



Join structures when all of the following conditions and the optionally connected condition are true:

	{} Person/FirstName	=	{} Address/FirstName	☒
AND	{} Person/LastName	=	{} Address/LastName	☒

To define a condition, select an item from each structure to compare for equality.

Structure 1:

- {} Person
  - {} FirstName
  - {} LastName
  - {} Email
  - {} Phone

Structure 2: Address

- {} Address
  - {} FirstName
  - {} LastName
  - {} City
  - {} Street
  - {} Number

☑ Show annotations

□ Show types

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:

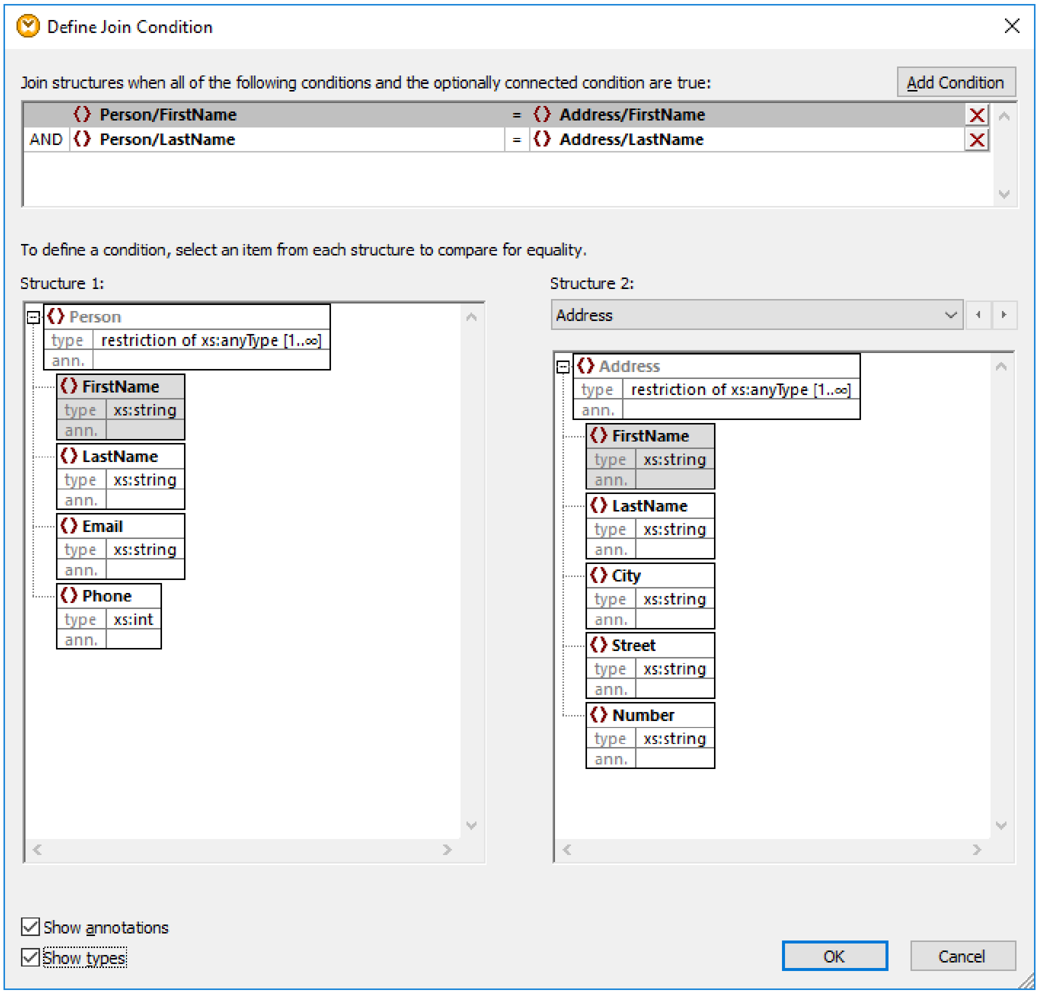
1. `FirstName` in the Structure 1 must be equal to `FirstName` in Structure 2, and
2. `LastName` in Structure 1 must be equal to `LastName` in Structure 2.

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.

- 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:



Join structures when all of the following conditions and the optionally connected condition are true:

	Person/FirstName	=	Address/FirstName
AND	Person/LastName	=	Address/LastName

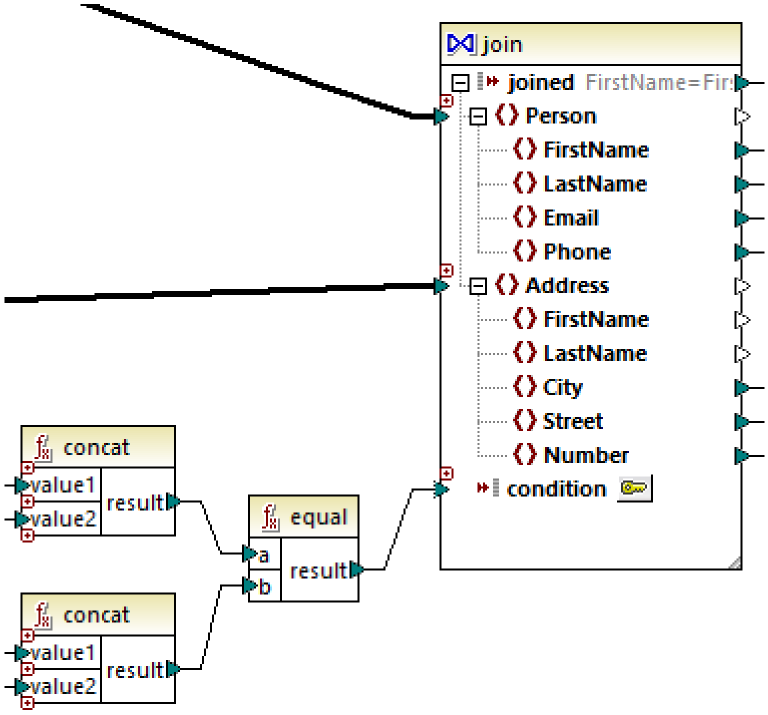
To define a condition, select an item from each structure to compare for equality.

☑ Show annotations

☑ Show types

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.



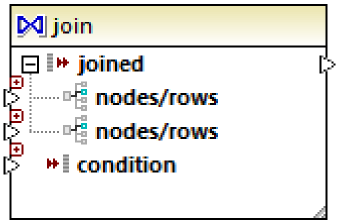
Note: If no condition is defined from the join component properties (Approach 1), the `condition` item of the join component must be connected (Approach 2).

Approach 3: Mixed approach

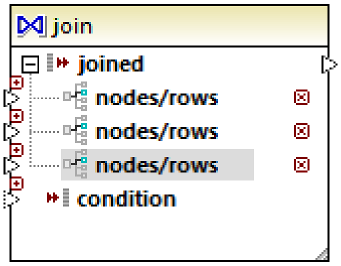
In the same mapping, it is possible to define some join conditions in component properties (Approach 1) and combine them with the one from the mapping (Approach 2). However, if you intend to join database tables in SQL mode, the conditions must be defined strictly using Approach 1 (see also About Joins in SQL Mode).

5.10.2 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. For a step-by-step example of how to join multiple database tables, see Example: Create CSV Report from Multiple Tables.

5.10.3 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\MapForce2018\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

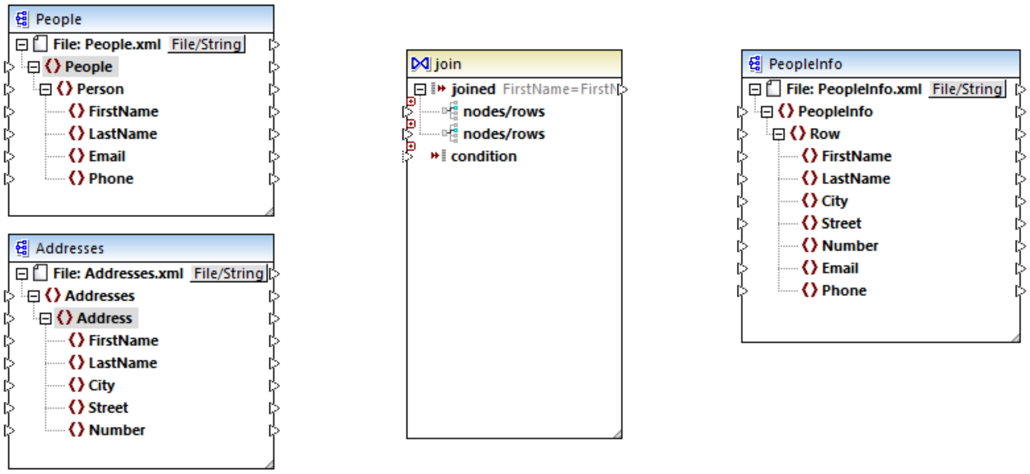
1. On the **Insert** menu, click **XML Schema/File**, and browse for the following source file: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\People.xml**.
2. Repeat the step above for **Addresses.xml** (the second source file).

Step 2: Add the target schema file to the mapping

- On the **Insert** menu, click **XML Schema/File**, and browse for **<Documents>\Altova \MapForce2018\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.

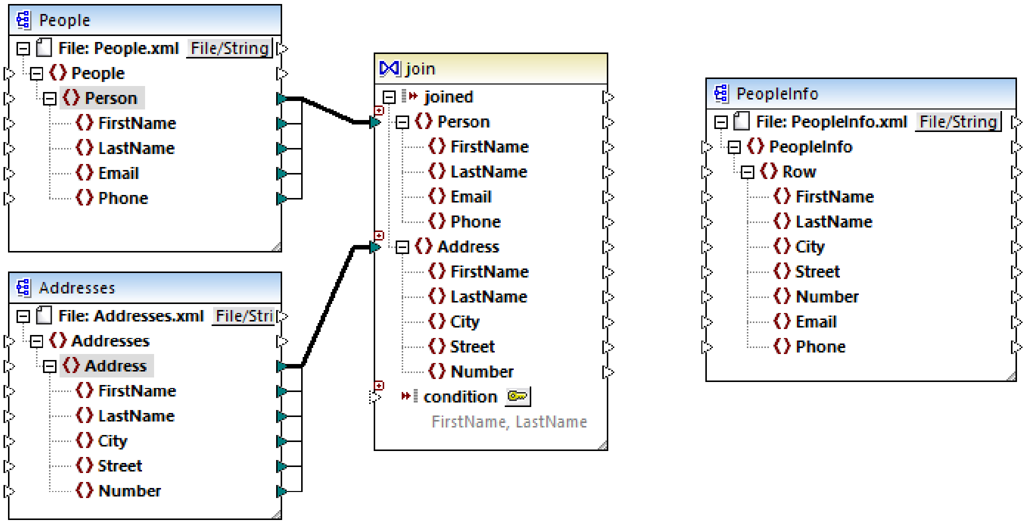
Step 3: Add the Join component

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):

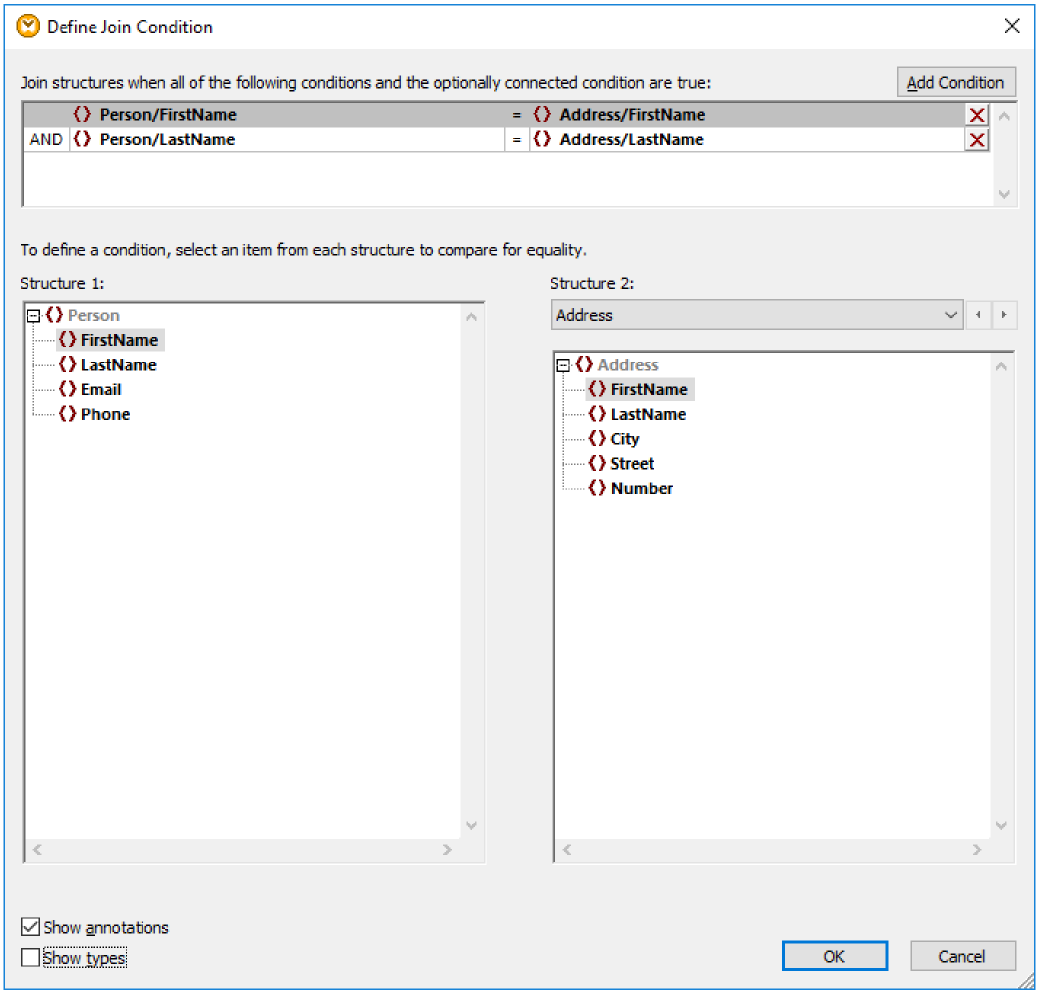


Observe the structure of the Join component. It has two `nodes/rows` items, which makes it possible to connect to it the two structures that need to be compared (in this case, the `Person` and the `Address` structures).

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`.



Define Join Condition

Join structures when all of the following conditions and the optionally connected condition are true:

	Person/FirstName	=	Address/FirstName
AND	Person/LastName	=	Address/LastName

To define a condition, select an item from each structure to compare for equality.

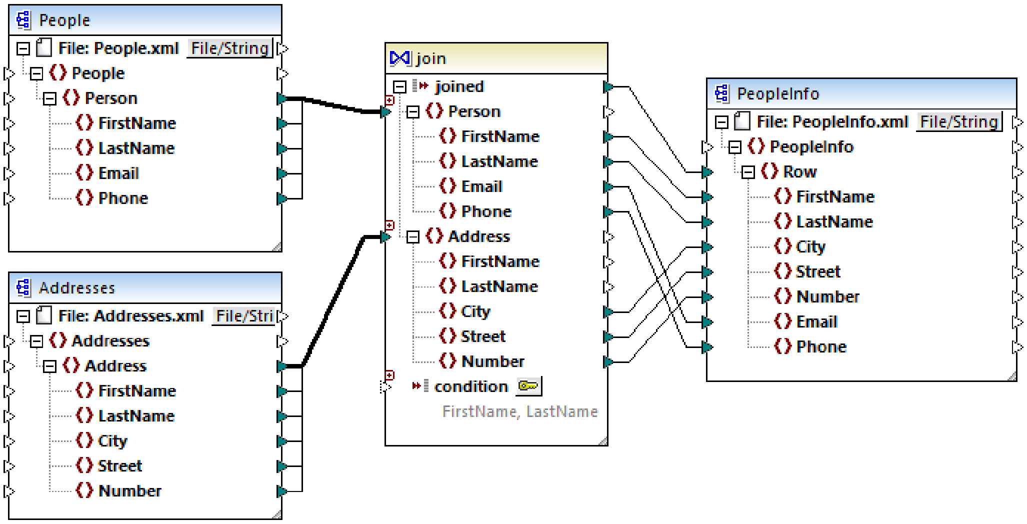
In some mappings, a condition consisting of one comparison may be sufficient to perform the join. However, in this example, it is important that two comparisons are created:

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.

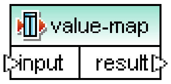
5.11 Using Value-Maps

The Value-Map component allows you to transform an input value to a different output value using a lookup table. This is useful for converting different enumeration types. The component only has one input and output item.

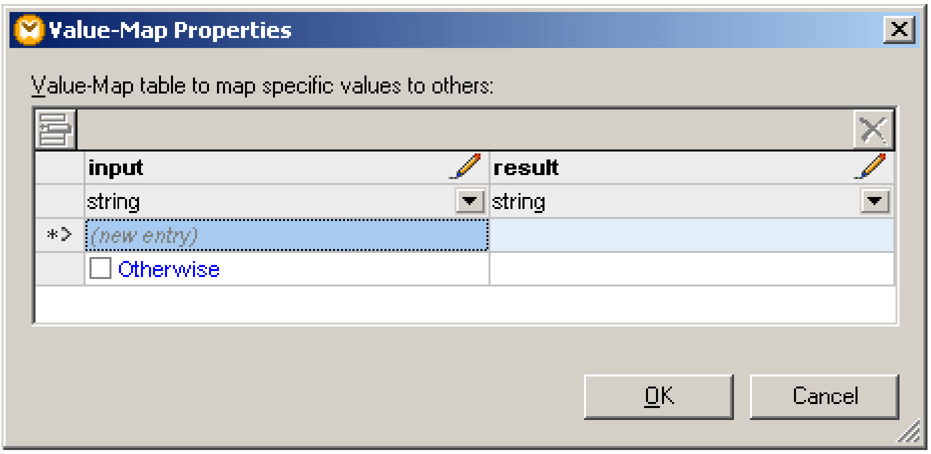
Note: if you want to retrieve/filter data based on specific criteria, please use the **Filter** component, see Filters and Conditions.

To use a Value-Map component:

1. Select the menu option **Insert | Value-Map**, or click the Value-Map icon in the icon bar.



2. Double click the Value-Map component to open the value map table.

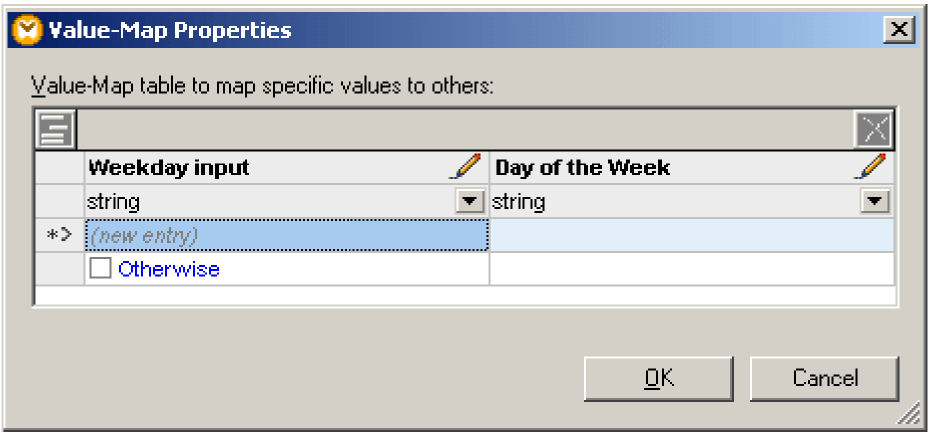


Value-Map Properties

Value-Map table to map specific values to others:

	input	result
	string	string
*>	(new entry)	
	☐ Otherwise	

3. Click into the column headers and enter **Weekday input** in the first, and **Day of the Week** in the second.



Value-Map Properties

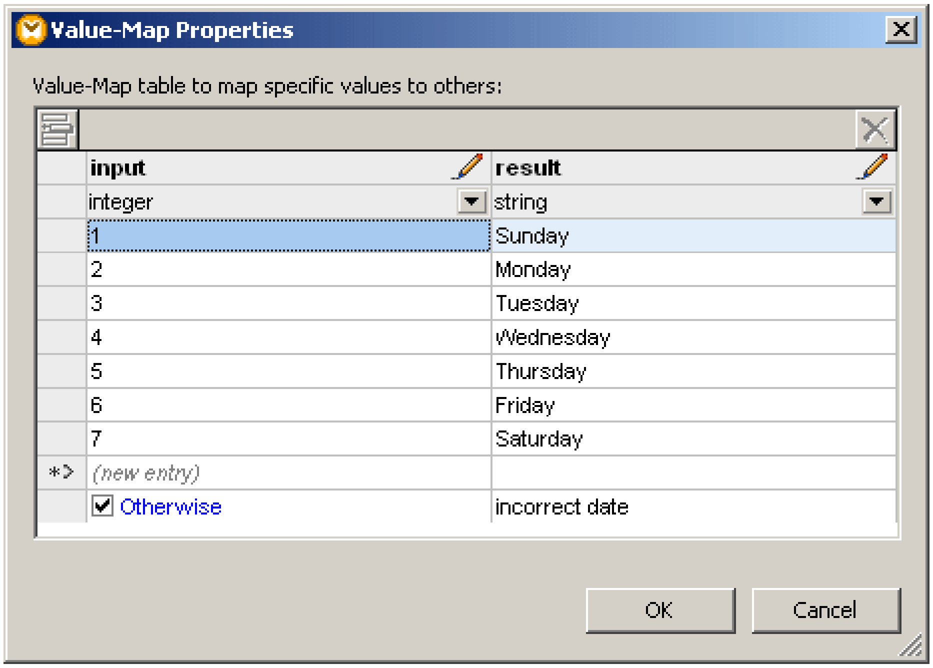
Value-Map table to map specific values to others:

	Weekday input	Day of the Week
	string	string
*>	(new entry)	
	☐ Otherwise	

4. Enter the input value that you want to transform, in the **Weekday input** column.
5. Enter the output value you want to transform that value to, in the **Day of the week**

column.

6. Simply type in the ***(new entry)*** input field to enter a new value pair.
7. Click the **datatype** combo box, below the column header to select the input and output datatypes, e.g. integer and string.



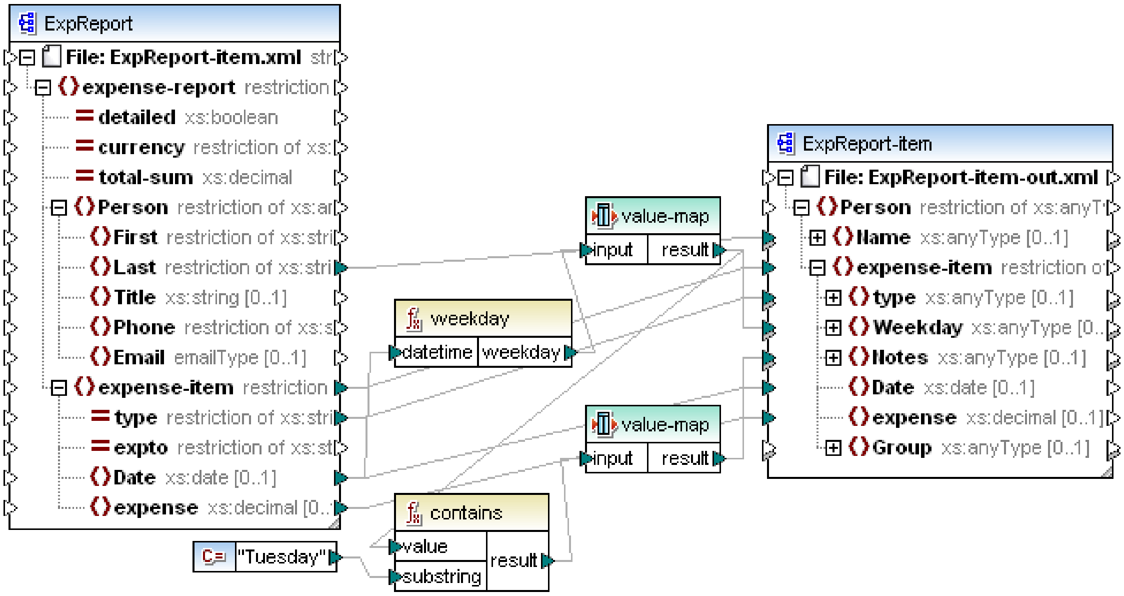
Value-Map table to map specific values to others:

	input	result
	integer	string
	1	Sunday
	2	Monday
	3	Tuesday
	4	Wednesday
	5	Thursday
	6	Friday
	7	Saturday
*>	*(new entry)*	
	☑ Otherwise	incorrect date

Note: activate the **Otherwise** check box, and enter the value, to define an alternative output value if the supplied values are not available on input. To pass through source data without changing it please see Passing data through a Value-Map unchanged.

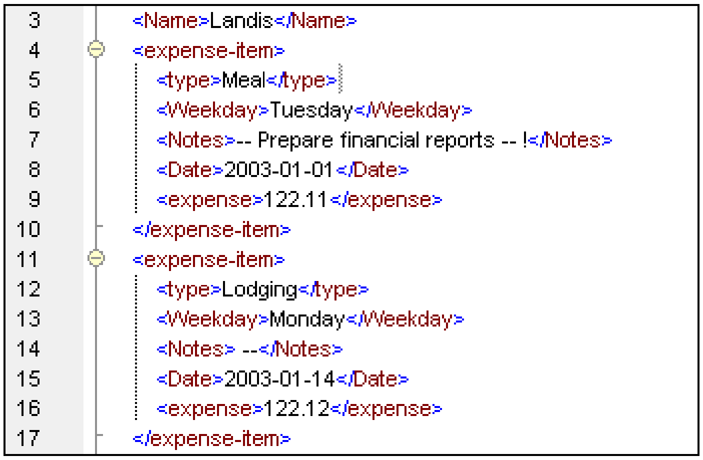
8. You can click the edit icons in the header rows to change the column names, which are also displayed in the mapping. This will make it easier to identify the purpose of the component in the mapping.

The **Expense-valmap.mfd** file in the ...\MapForceExamples\Tutorial\ folder is a sample mapping that shows how the Value-Map can be used.



What this mapping does:
Extracts the day of the week from the Date item in the data source, converts the numerical value into text, and places it in the Weekday item of the target component i.e. Sunday, Monday etc.

- The **weekday** function extracts the weekday number from the **Date** item in the ExpReport source file. The result of this function are integers ranging from 1 to 7.
- The Value-Map component transforms the integers into weekdays, i.e. Sunday, Monday, etc. as shown in the graphic at the top of this section.
- If the output contains "Tuesday", then the corresponding output "Prepare Financial Reports" is mapped to the Notes item in the target component.
- Clicking the Output tab displays the target XML file with the transformed data.



```
    <Name>Landis</Name>
    <expense-item>
        <type>Meal</type>
        <Weekday>Tuesday</Weekday>
        <Notes>-- Prepare financial reports -- !</Notes>
        <Date>2003-01-01</Date>
        <expense>122.11</expense>
    </expense-item>
    <expense-item>
        <type>Lodging</type>
        <Weekday>Monday</Weekday>
        <Notes> --</Notes>
        <Date>2003-01-14</Date>
        <expense>122.12</expense>
    </expense-item>
```

Note:

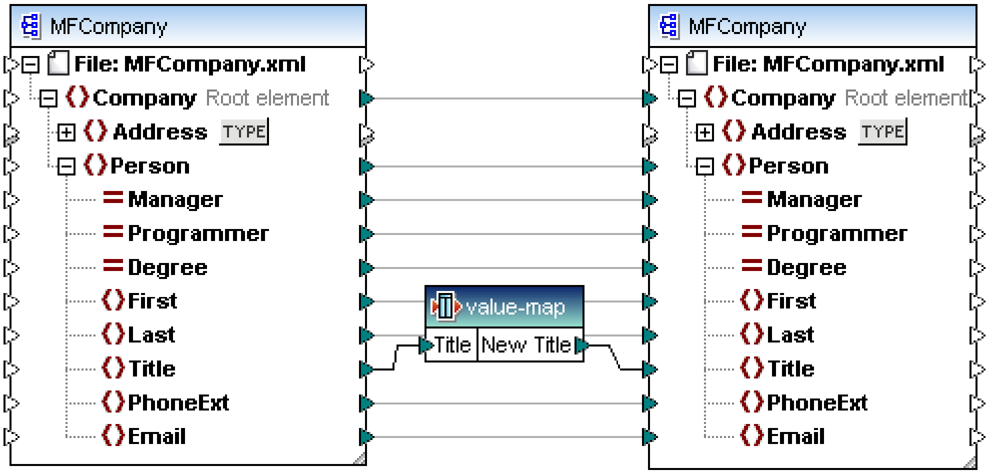
Placing the mouse cursor over the value map component opens a popup containing the currently defined values.

The output from various types of logical, or string functions, can only be a boolean "**true**" or "**false**" value. The value you want to test for, must thus be entered into the **input** field of the value map table e.g. "true".

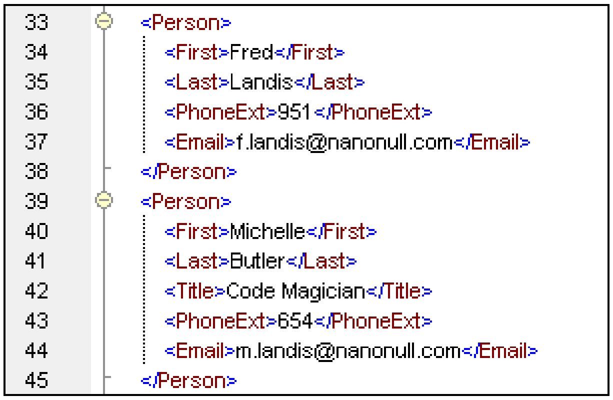
5.11.1 Passing data through a Value-Map unchanged

This section describes a mapping situation where some specific node data have to be transformed, while the rest of the node data have to be passed on to the target node unchanged.

An example of this would be a company that changes some of the titles in a subsidiary. In this case it might change two title designations and want to keep the rest as they currently are.

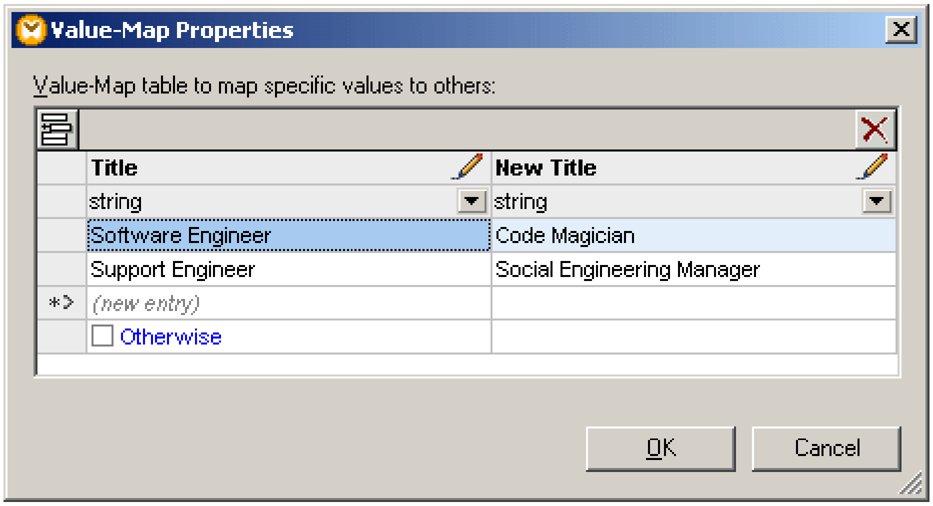


The obvious mapping would be the one shown above, which uses the value-map component to transform the specific titles.
Clicking the Output tab shows us the result of the mapping:



```
<Person>
    <First>Fred</First>
    <Last>Landis</Last>
    <PhoneExt>951</PhoneExt>
    <Email>f.landis@nanonull.com</Email>
</Person>
<Person>
    <First>Michelle</First>
    <Last>Butler</Last>
    <Title>Code Magician</Title>
    <PhoneExt>654</PhoneExt>
    <Email>m.landis@nanonull.com</Email>
</Person>
```

For those persons who are neither of the two types shown in the value-map component, the Title element is deleted in the output file.



Value-Map Properties

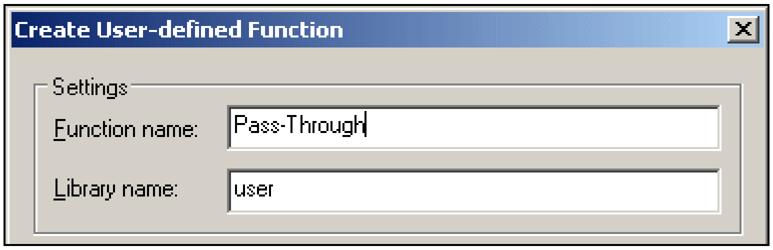
Value-Map table to map specific values to others:

	Title	New Title
	string	string
	Software Engineer	Code Magician
	Support Engineer	Social Engineering Manager
*>	*(new entry)*	
	□ Otherwise	

Possible alternative:
Clicking the **Otherwise** check box and entering a substitute term, does make the Title node reappear in the output file, but it now contains the same **New Title** for all other persons of the company.

Solution:
Create a user-defined function containing the value-map component, and use the **substitute-missing** function to supply the original data for the empty nodes.

1. Click the value-map component and select **Function | Create user-defined function from Selection**.



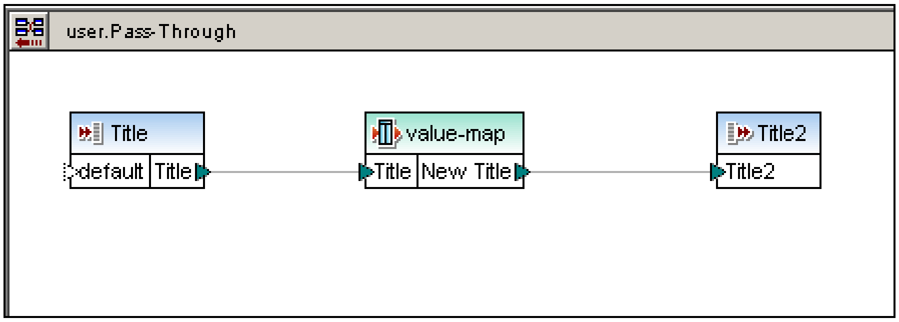
Create User-defined Function

Settings

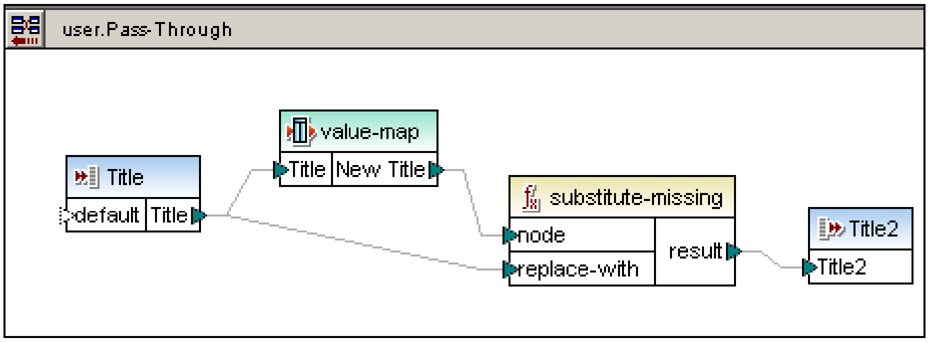
Function name: Pass-Through

Library name: user

2. Enter a name for the function e.g. Pass-Through and click OK.



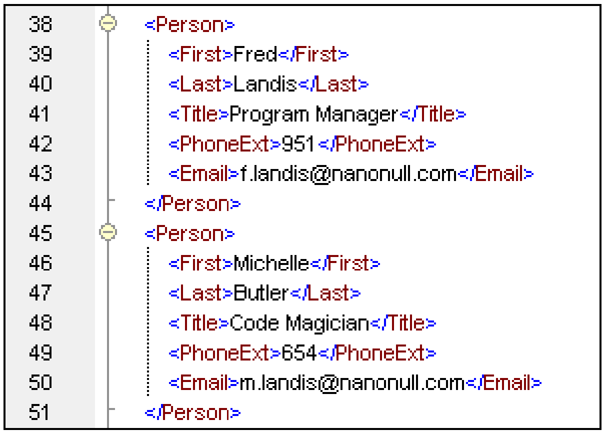
3. Insert a **substitute-missing** function from the **core | node function** section of the Libraries pane, and create the connections as shown in the screen shot below.



4. Click the Output tab to see the result:

Result of the mapping:

- The two Title designations in the value-map component are transformed to New Title.
- All other Title nodes of the source file, retain their original Title data in the target file.



```
<Person>
  <First>Fred</First>
  <Last>Landis</Last>
  <Title>Program Manager</Title>
  <PhoneExt>951</PhoneExt>
  <Email>f.landis@nanonull.com</Email>
</Person>
<Person>
  <First>Michelle</First>
  <Last>Butler</Last>
  <Title>Code Magician</Title>
  <PhoneExt>654</PhoneExt>
  <Email>m.landis@nanonull.com</Email>
</Person>
```

Why is this happening:
The value-map component evaluates the input data.

- If the incoming data **matches one** of the entries in the first column, the data is transformed and passed on to the node parameter of substitute-missing, and then on to Title2.

- If the incoming data does not match **any entry** in the left column, then nothing is passed on from value-map to the node parameter i.e. this is an **empty node**.

  When this occurs the substitute-missing function retrieves the original node and data from the Title node, and passes it on through the **replace-with** parameter, and then on to Title2.

5.11.2 Value-Map component properties

Actions:

Click the insert icon to **insert** a new row before the currently active one.

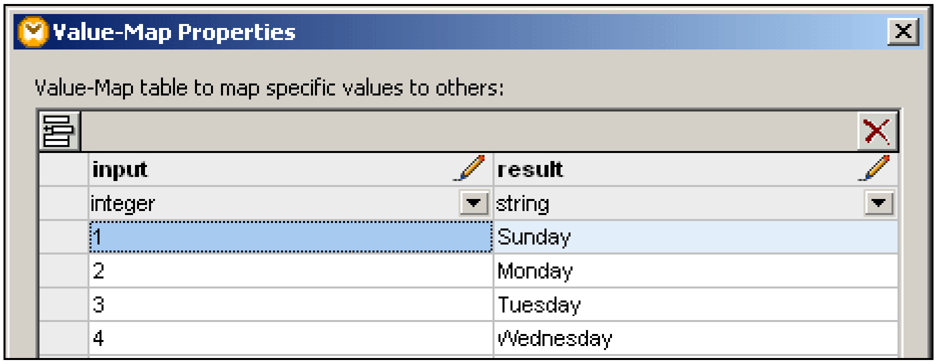
✕ Click the delete icon to **delete** the currently active row.

✎ Click the edit icon to **edit** the column header.

You can also reorder lines by dragging them.

Changing the column header:

Double clicking the column header, or clicking the pencil icon, allows you to edit the column name and change it to something more meaningful. This will make it easier to identify the purpose of the component, as the column names are also displayed in the mapping.



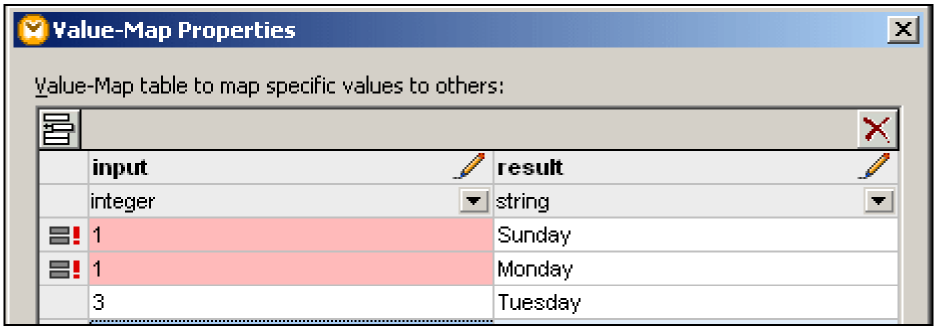
Value-Map Properties

Value-Map table to map specific values to others:

input	result
integer	string
1	Sunday
2	Monday
3	Tuesday
4	Wednesday

Using unique Input values:

The values entered into the input column must be unique. If you enter two identical values, both are automatically highlighted for you to enable you to correct one of them.



Value-Map Properties

Value-Map table to map specific values to others:

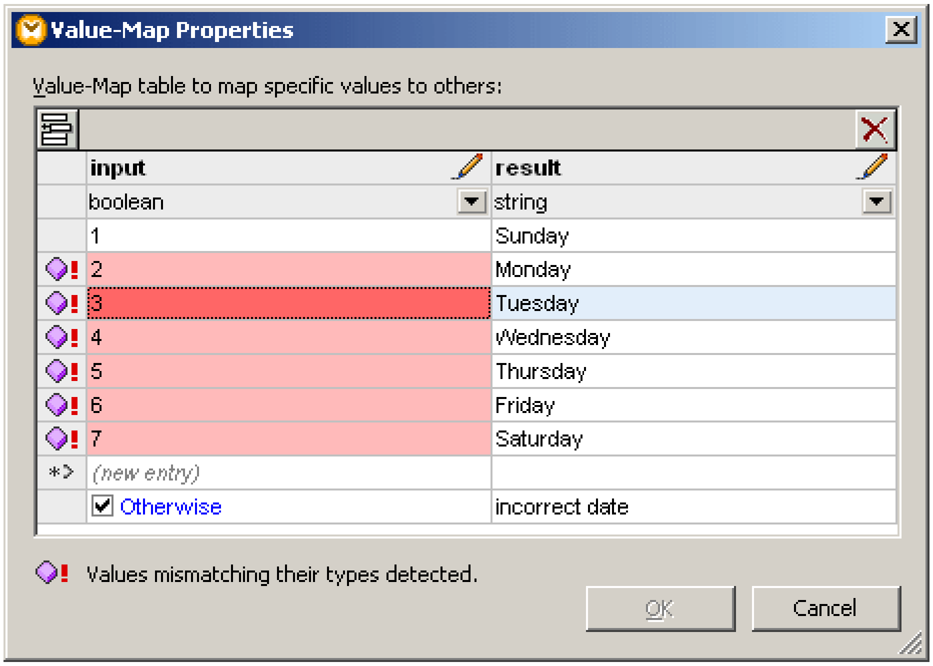
	input	result
	integer	string
!	1	Sunday
!	1	Monday
	3	Tuesday

Having corrected one of the values, the OK button is again enabled.

Input and output datatypes

The input and result datatypes are automatically checked when a selection is made using the combo box. If a mismatch occurs, then the respective fields are highlighted and the OK button is disabled. Change the datatype to one that is supported.

In the screenshot below a boolean and string have been selected.



Value-Map table to map specific values to others:

	input	result
	boolean	string
	1	Sunday
◇!	2	Monday
◇!	3	Tuesday
◇!	4	Wednesday
◇!	5	Thursday
◇!	6	Friday
◇!	7	Saturday
*>	(new entry)	
	☑ Otherwise	incorrect date

◇! Values mismatching their types detected.

5.12 Adding 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). For example, you may want to throw an exception if the value of some mapping item is greater than some custom threshold.

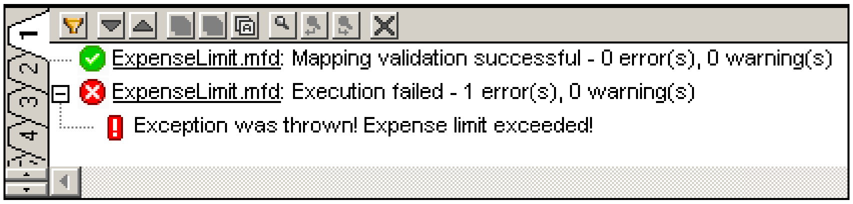
To add an exception to the mapping:

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:

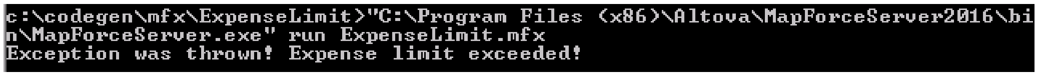
- In MapForce, the Messages window displays an error, and the exception text (in this case, "Expense limit exceeded").



ExpenseLimit.mfd: Mapping validation successful - 0 error(s), 0 warning(s)
ExpenseLimit.mfd: Execution failed - 1 error(s), 0 warning(s)
Exception was thrown! Expense limit exceeded!

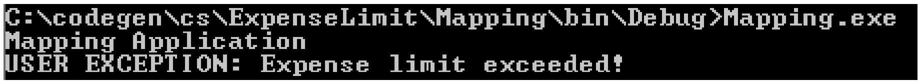
If the mapping language is XSLT 2.0 or XQuery, an "Execution failed" error appears in the Messages window, and the respective XSLT2 or XQuery tab is opened. The error line is highlighted in the Messages window.

- 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!
```

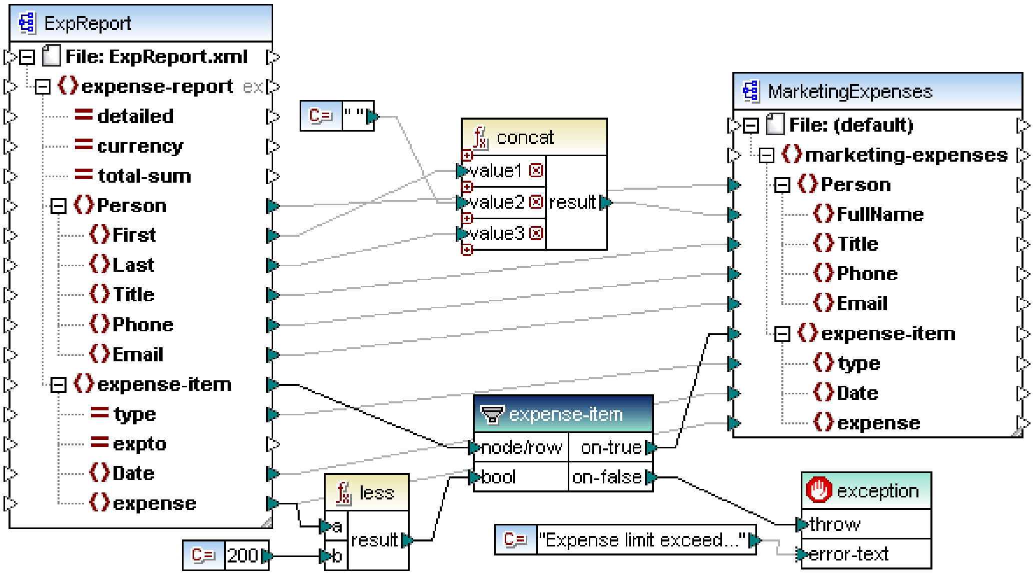
- If you run the mapping from the generated C#, C++, or Java code, the error "USER EXCEPTION" is returned, followed by the custom exception text you have defined in MapForce.



```
C:\codegen\cs\ExpenseLimit\Mapping\bin\Debug>Mapping.exe
Mapping Application
USER EXCEPTION: Expense limit exceeded!
```

5.12.1 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\MapForce2018\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.

5.12.2 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.

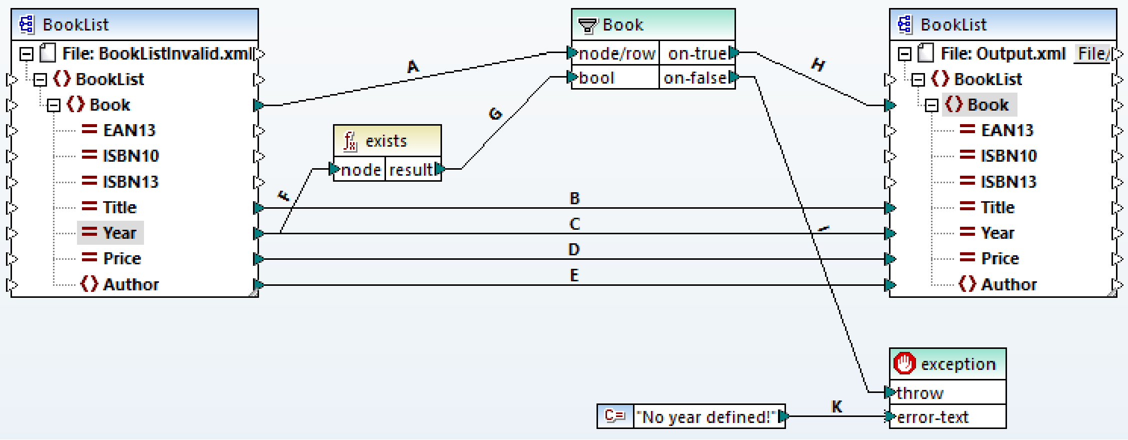
To add the source schema to the mapping:

1. On the **Insert** menu, click **XML Schema/File**, and browse for **<Documents>\Altova\MapForce2018\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)
- 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\MapForce2018\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.

5.13 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 into database fields.
- 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

String parsing and serialization is available for the following MapForce component types:

- Text (CSV, fixed-length field text)
- XML schema files

String parsing and serialization is supported in MapForce target languages as follows.

Language	Reading	Writing
BUILT-IN (preview the mapping transformation)	Yes	Yes
BUILT-IN (run the MapForce Server execution file)	Yes	Yes

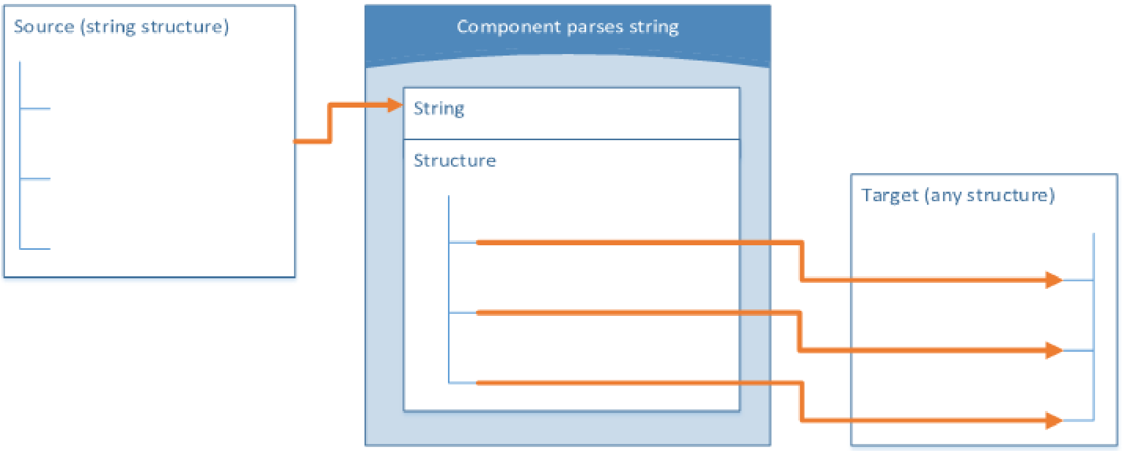
This section includes the following topics:

- About the Parse/Serialize Component
- Example: Serialize to String (XML to Database)

5.13.1 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.

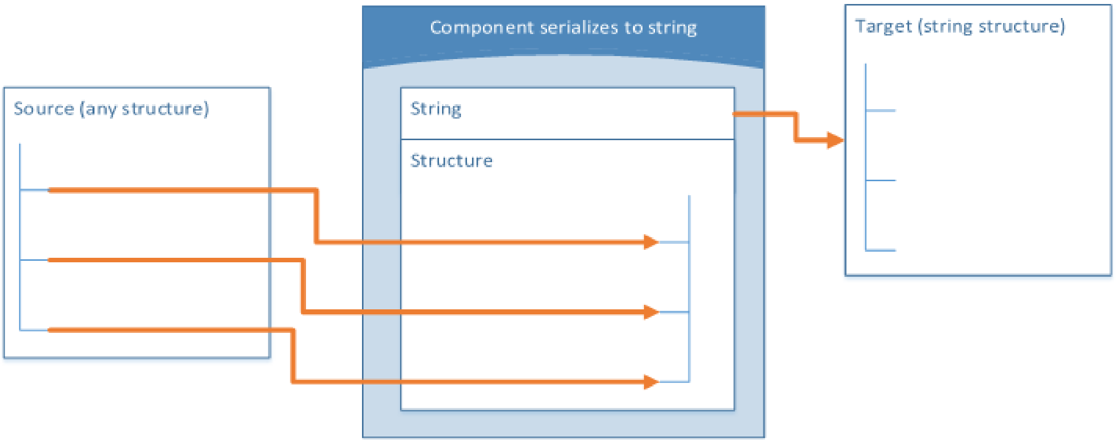
You can use a "Parse/Serialize String" component for string parsing when, for some reason, you need to convert a string that has structure (for example, some XML stored as string in a database) into another format. Parsing data from the source string to the "Parse/Serialize" component means that the 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.

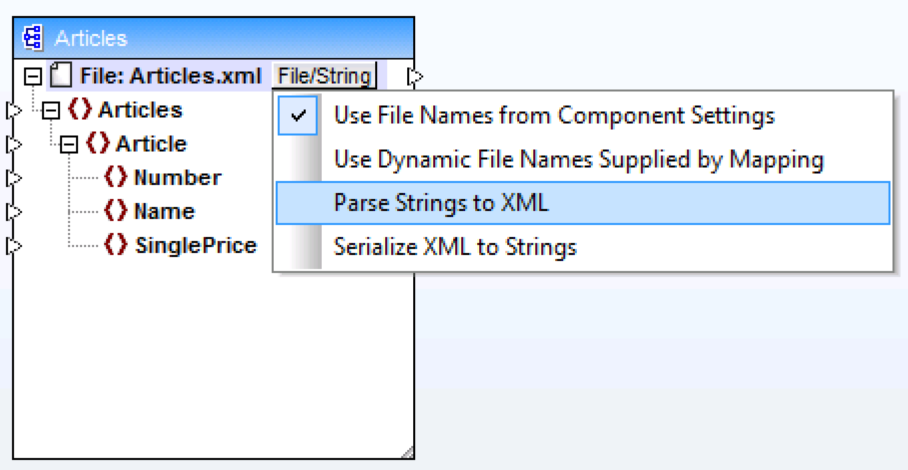
When you serialize data from a component to string, the reverse happens. Specifically, the entire structure of the MapForce component becomes a string structure which you can further manipulate as necessary. For example, this enables you to write an XML file (or XML fragment) to a database field or to 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).

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). Note that not all component settings are available when a component is in either "Parse" or "Serialize" mode.

5.13.2 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\MapForce2018\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>
```

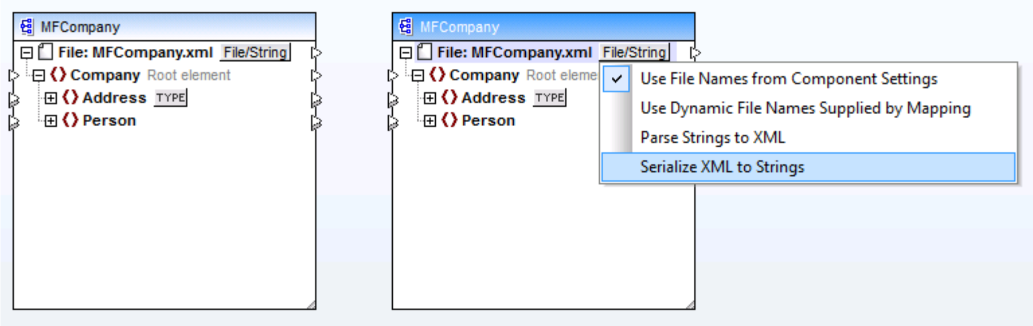
Your goal is to extract each `<Person>` element from the XML file and insert it literally (including XML tags) as a new database record in the `PEOPLE` table of a SQLite database. The `PEOPLE` table contains only two columns: `ID` and `PERSON`. Its full definition is as follows:

```
CREATE TABLE PEOPLE (ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, PERSON
TEXT);
```

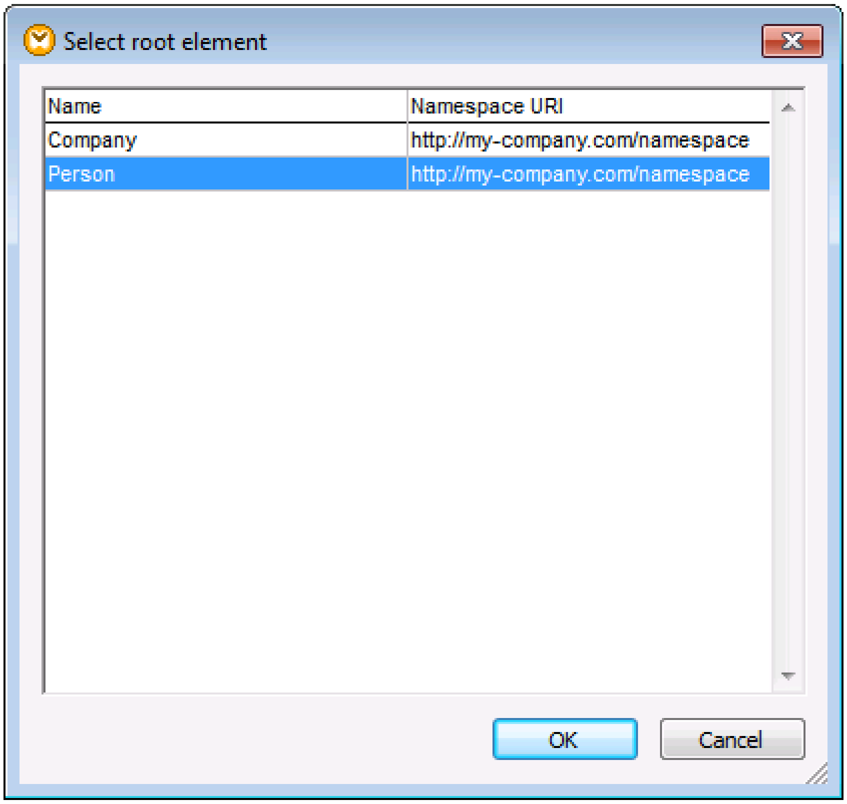
After the mapping is executed, the expected result is that the `PEOPLE` table will have the same number of rows as the number of <Person> elements in the XML file.

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\MapForce2018\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**.



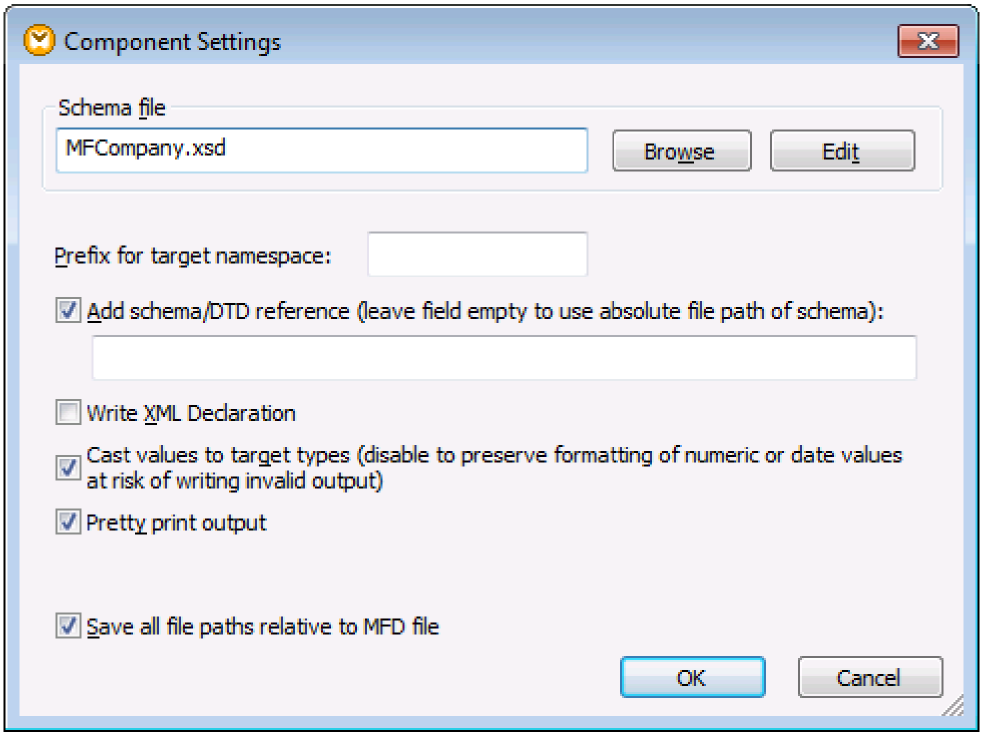
4. Right-click the duplicated component and select **Change Root Element** from the context menu. Then change the root element to `<Person>`.



Name	Namespace URI
Company	http://my-company.com/namespace
Person	http://my-company.com/namespace

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 disable the **Write XML Declaration** option. This prevents the XML declaration from being written for each `<Person>` element.



Component Settings

Schema file

MFCompany.xsd

Prefix for target namespace:

☑ Add schema/DTD reference (leave field empty to use absolute file path of schema):

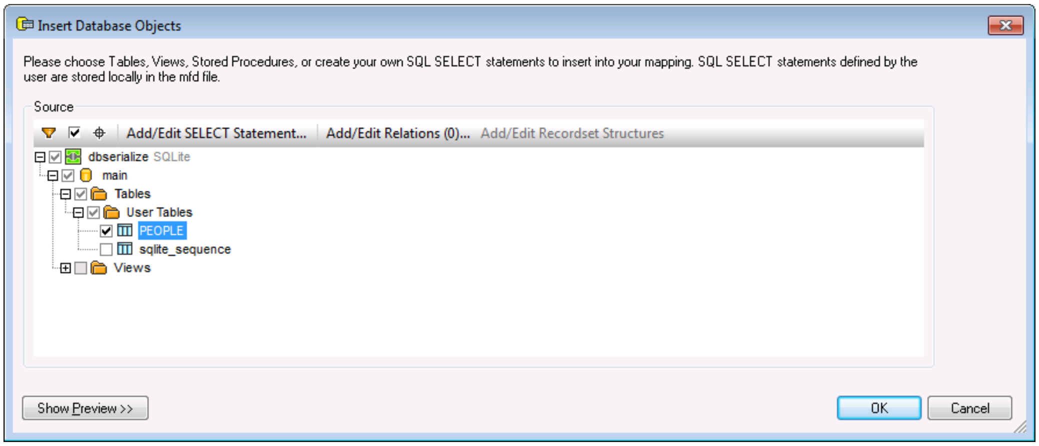
☐ Write XML Declaration

☑ Cast values to target types (disable to preserve formatting of numeric or date values at risk of writing invalid output)

☑ Pretty print output

☑ Save all file paths relative to MFD file

6. Add to the mapping area the target SQLite database component, from the following path: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\\dbserialize.db**. (To add the database component, use the **Insert | Database** menu command, see also Connecting to a Database). When prompted to insert a database object, select the PEOPLE table.



Insert Database Objects

Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

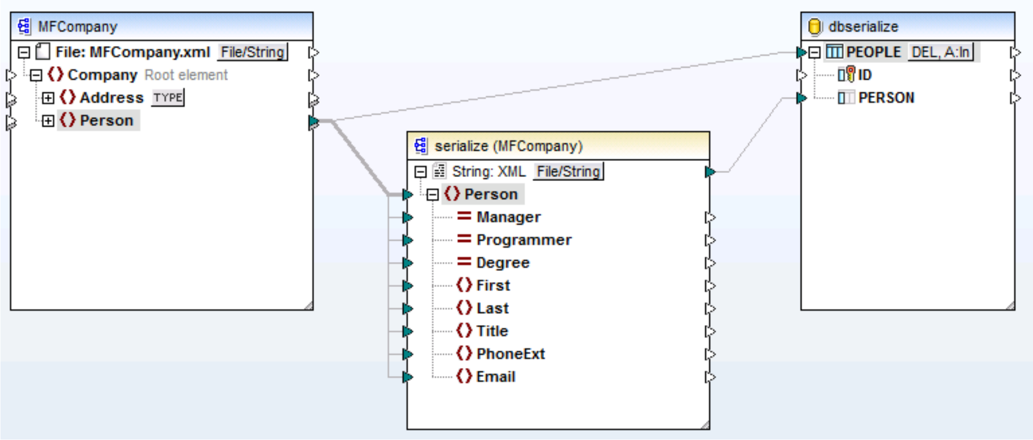
Source

Add/Edit SELECT Statement...
Add/Edit Relations (0)...
Add/Edit Recordset Structures

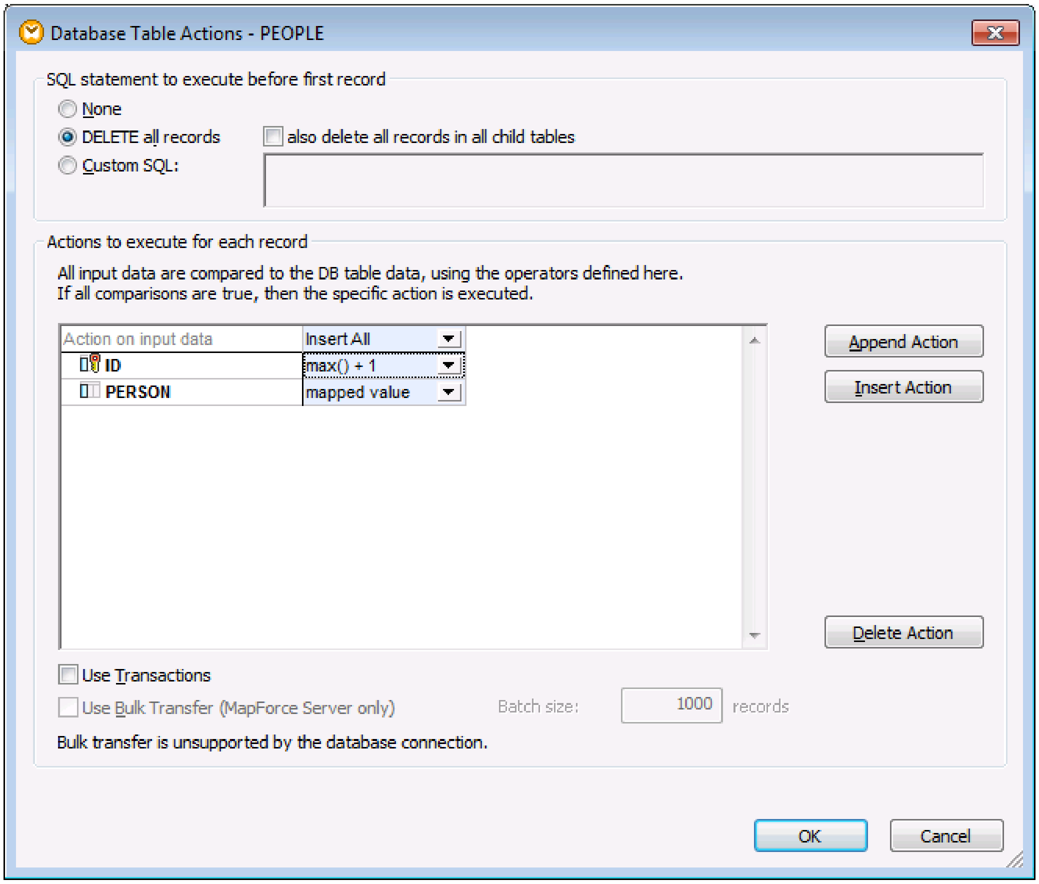
- ☑ dbserialize SQLite
  - ☑ main
    - ☑ Tables
      - ☑ User Tables
        - ☑ PEOPLE
        - ☐ sqlite_sequence
    - ☐ Views

Show Preview >>
OK
Cancel

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 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 instruct MapForce to perform the following actions every time when the mapping transformation runs:
   a. Delete all records from the table;
   b. Increment the value of ID by 1.


Database Table Actions - PEOPLE

SQL statement to execute before first record

○ None
● DELETE all records □ also delete all records in all child tables
○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on input data	Insert All
ID	max() + 1
PERSON	mapped value

□ Use Transactions
□ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records

Bulk transfer is unsupported by the database connection.

Observe the **max()+1** action selected for the ID column. This instructs MapForce to analyze what is the maximum ID value already existing in the database, 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.

5.14 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, which the subject of this topic, you can perform data conversions such as the one above.

Note: You can also perform the transformation above by using the `node-name` and `static-node-name` 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*, XSLT2, XQuery*, C#*, C++*, Java*.

* Requires MapForce Professional or Enterprise Edition.

For information about how dynamic node names work, Getting Access to Node Names. For a step-by-step mapping example, see Example: Map Element Names to Attribute Values.

5.14.1 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\MapForce2018\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\MapForce2018\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 children elements

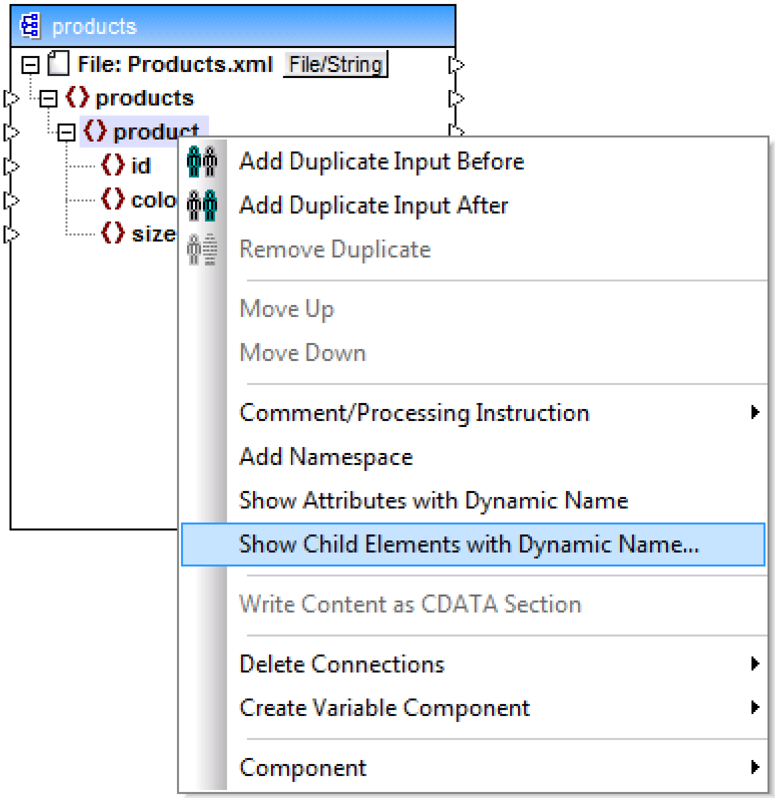
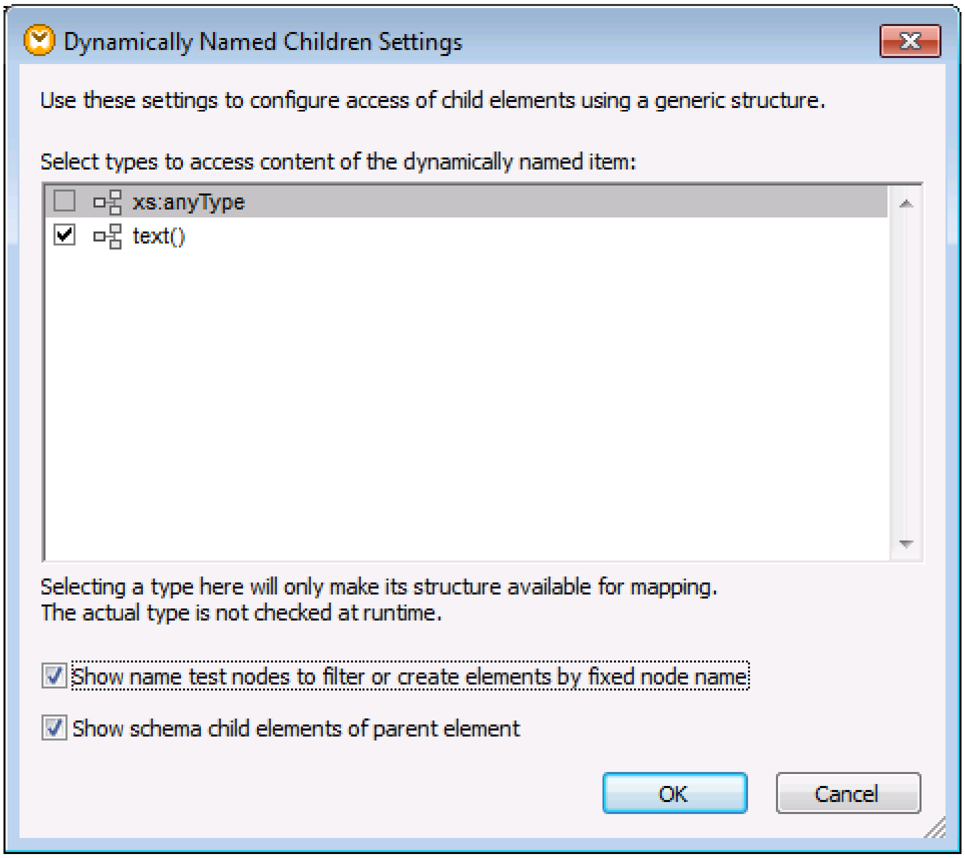

products
File: Products.xml
File/String
products
product
id
colo
size
Add Duplicate Input Before
Add Duplicate Input After
Remove Duplicate
Move Up
Move Down
Comment/Processing Instruction
Add Namespace
Show Attributes with Dynamic Name
Show Child Elements with Dynamic Name...
Write Content as CDATA Section
Delete Connections
Create Variable Component
Component

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.



Dynamically Named Children Settings

Use these settings to configure access of child elements using a generic structure.

Select types to access content of the dynamically named item:

- □ xs:anyType
- ☑ text()

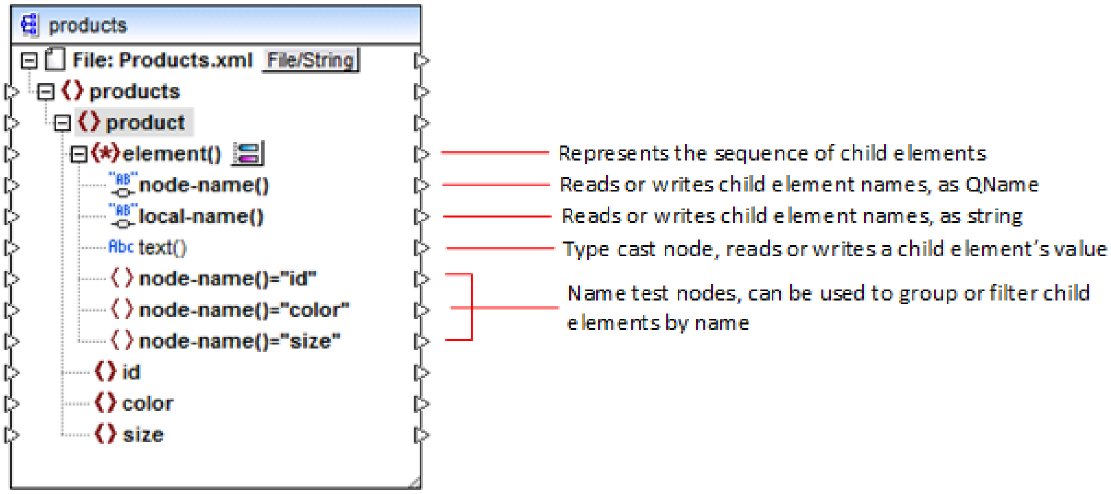
Selecting a type here will only make its structure available for mapping.
The actual type is not checked at runtime.

☑ Show name test nodes to filter or create elements by fixed node name

☑ Show schema child elements of parent element

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.



```
products
File: Products.xml  File/String
  products
    product
      element()
        node-name()
        local-name()
        text()
        node-name()="id"
        node-name()="color"
        node-name()="size"
      id
      color
      size
```

Represents the sequence of child elements

Reads or writes child element names, as QName

Reads or writes child element names, as string

Type cast node, reads or writes a child element's value

Name test nodes, can be used to group or filter child elements by name

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.

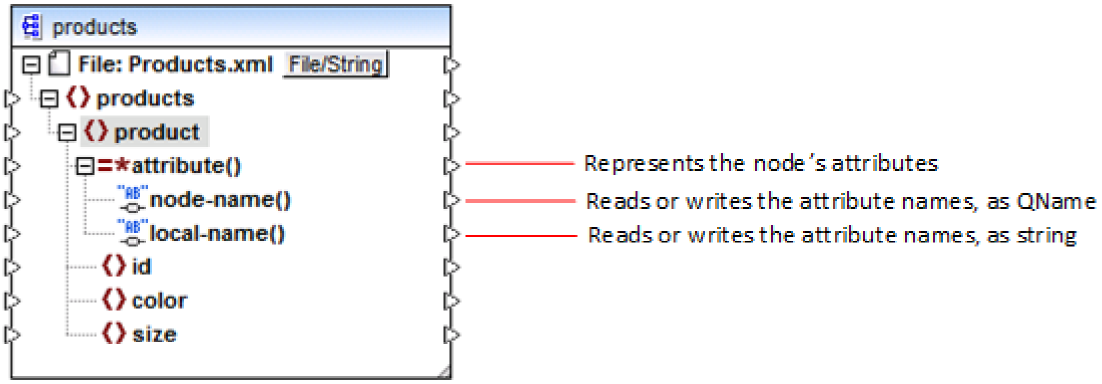

products
File: Products.xml
File/String
products
product
attribute()
node-name()
local-name()
id
color
size
Represents the node's attributes
Reads or writes the attribute names, as QName
Reads or writes the attribute names, as string

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.

The node types that you can work with in "dynamic node name" mode are described below.

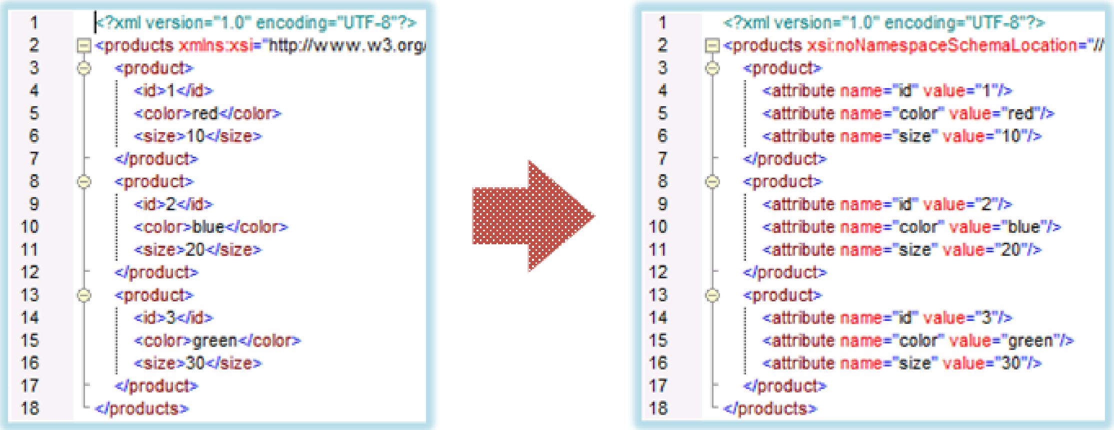
element()

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:



```
<?xml version="1.0" encoding="UTF-8"?>
<products xmlns:xsi="http://www.w3.org/
    <product>
        <id>1</id>
        <color>red</color>
        <size>10</size>
    </product>
    <product>
        <id>2</id>
        <color>blue</color>
        <size>20</size>
    </product>
    <product>
        <id>3</id>
        <color>green</color>
        <size>30</size>
    </product>
</products>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<products xsi:noNamespaceSchemaLocation="//
    <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>
```

Fig. 6 Mapping XML element names to attribute values (requirement)

The mapping which would achieve this goal looks as follows:

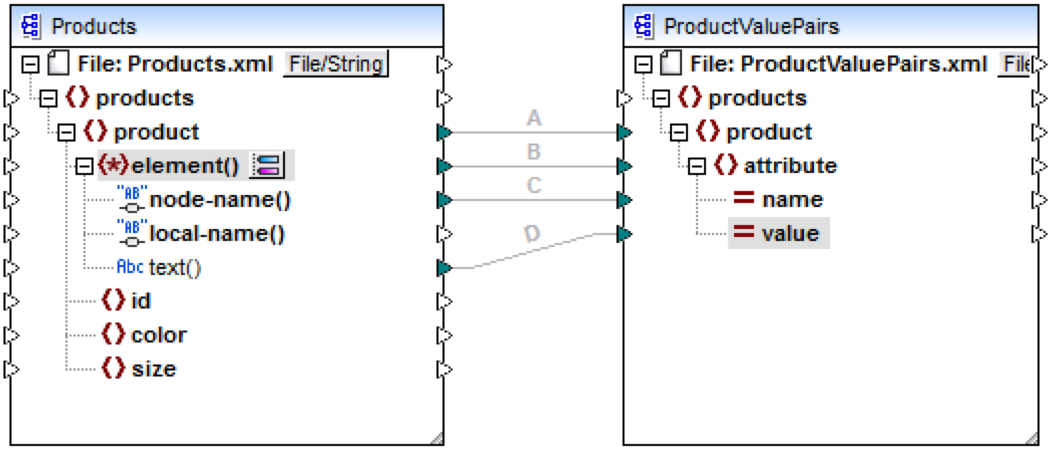


Fig. 7 Mapping XML element names to attribute values (in MapForce)

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.

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).

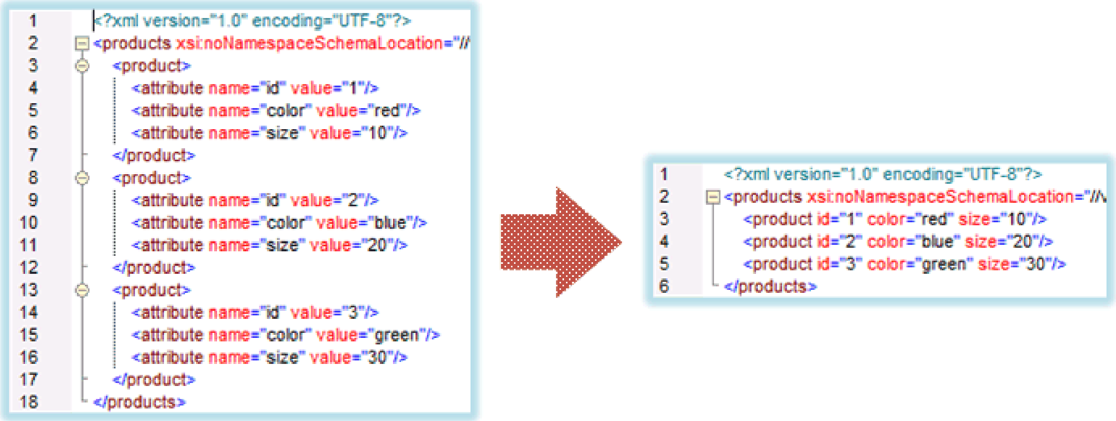
attribute()

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:



```
<?xml version="1.0" encoding="UTF-8"?>
<products xsi:noNamespaceSchemaLocation="//
    <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>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<products xsi:noNamespaceSchemaLocation="//\
    <product id="1" color="red" size="10"/>
    <product id="2" color="blue" size="20"/>
    <product id="3" color="green" size="30"/>
</products>
```

Fig. 8 Mapping attribute values to attribute names (requirement)

The mapping which would achieve this goal looks as follows:

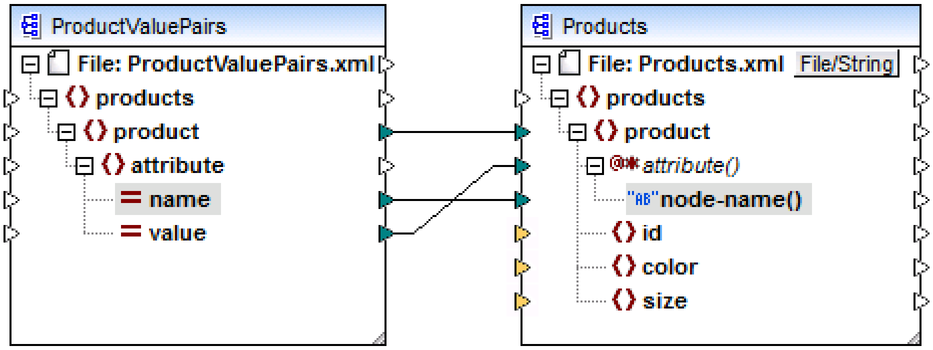


Fig. 9 Mapping attribute values to attribute names (in MapForce)

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\MapForce2018 \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.

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.

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). For an example, see Example: Group and Filter Nodes by Name.

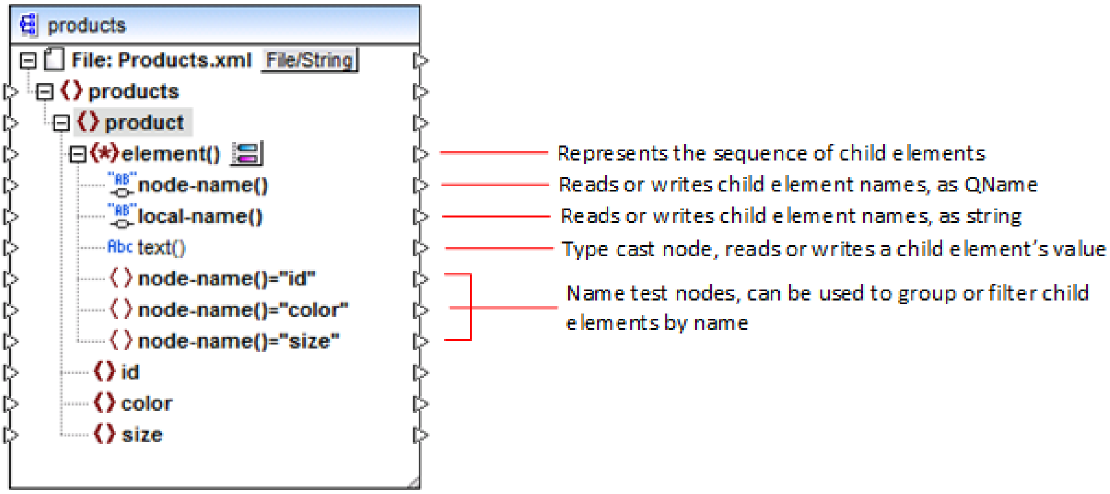
In general, the name test nodes work almost like normal element nodes for reading and writing values and subtree structures. However, because the mapping semantics is different when dynamic access is enabled, there are some limitations. For example, you cannot concatenate the value of two name test nodes.

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()`.

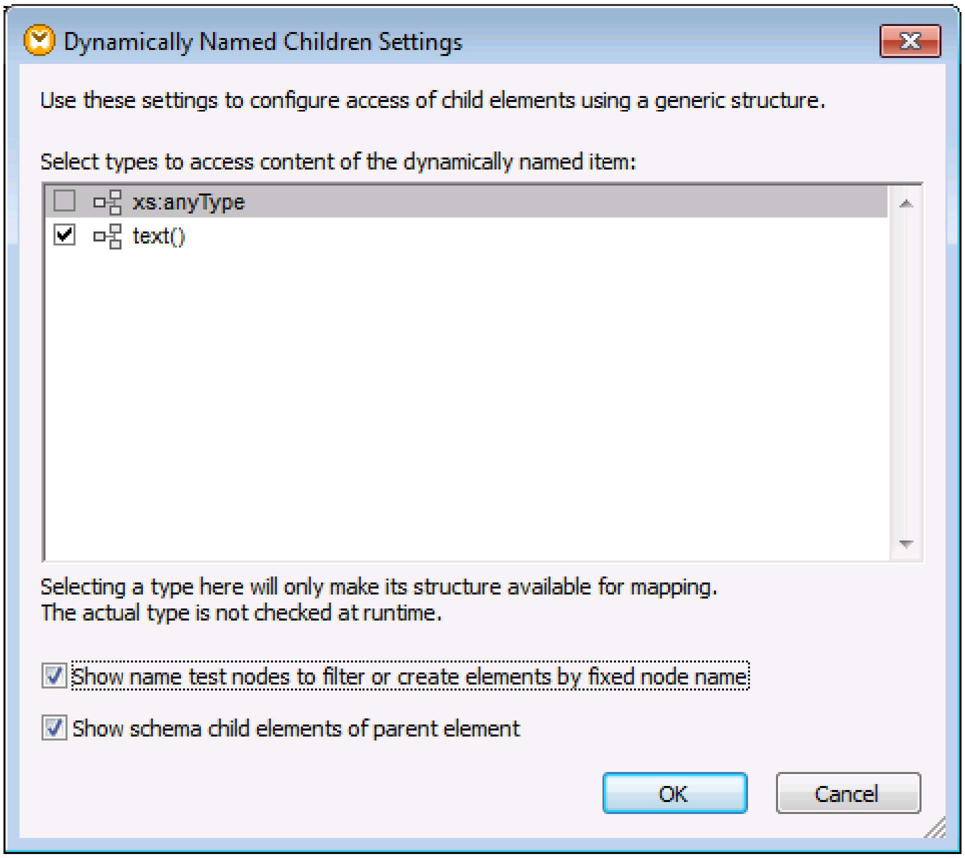
If necessary, you can hide the name test nodes from the component. To do this, click the **Change Selection** ( ) button next to the `element()` node. Then, in the "*Dynamically Named Children Settings*" dialog box (Fig. 2), clear the **Show name test nodes...** check box.

5.14.2 Accessing Nodes of Specific Type

As mentioned in the previous section, Getting Access to Node Names, 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.


products
File: Products.xml
File/String
products
product
element()
node-name()
local-name()
text()
node-name()="id"
node-name()="color"
node-name()="size"
id
color
size
Represents the sequence of child elements
Reads or writes child element names, as QName
Reads or writes child element names, as string
Type cast node, reads or writes a child element's value
Name test nodes, can be used to group or filter child elements by name

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

Use these settings to configure access of child elements using a generic structure.

Select types to access content of the dynamically named item:

□ xs:anyType

☑ text()

Selecting a type here will only make its structure available for mapping.
The actual type is not checked at runtime.

☑ Show name test nodes to filter or create elements by fixed node name

☑ Show schema child elements of parent element

"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. 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:

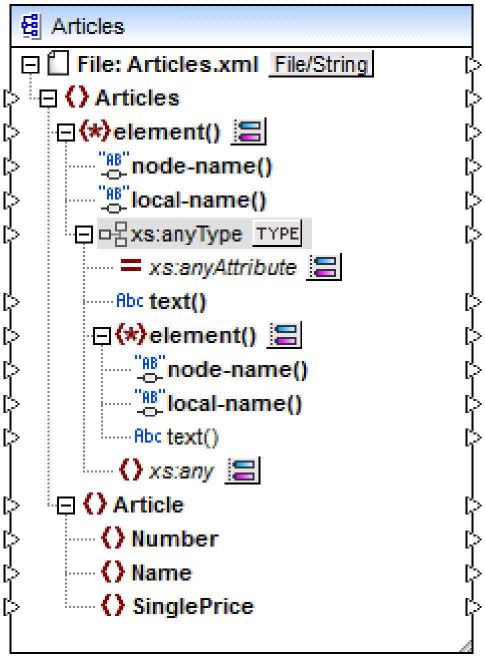
a) Set the type cast node to be of the complex type that you need to match (see item 2 in

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.

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, 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", "grand-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\MapForce2018 \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).
- 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). If using this option, make sure that the selected attribute or element can actually exist at that particular level according to the schema.

5.14.3 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\MapForce2018\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.

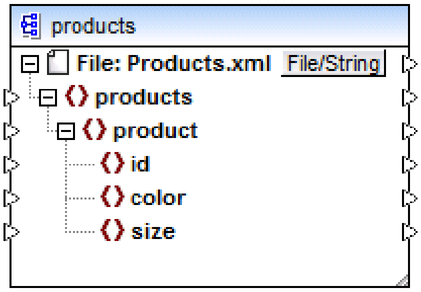
Step 1: Add the source XML component to the mapping

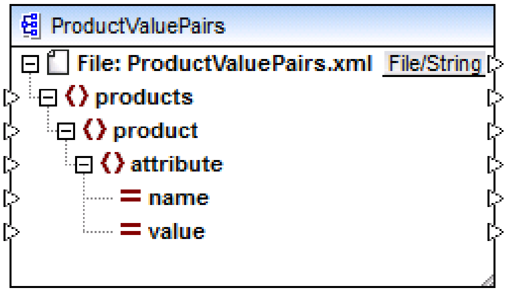
- On the **Insert** menu, click **XML Schema/File**, and browse for the following file: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\Products.xml**. This XML file points to the **Products.xsd** schema located in the same folder.

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\MapForce2018\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.

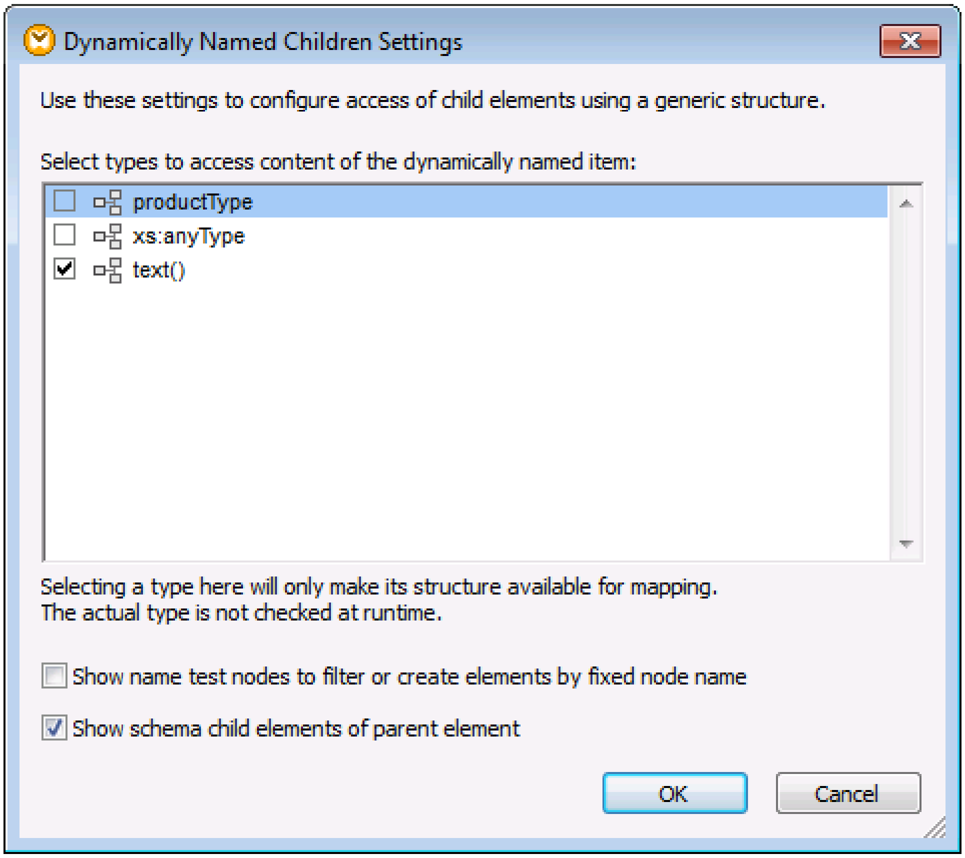
At this stage, the mapping should look as follows:





Step 3: Enable dynamic access to child nodes

1. Right-click the `products` 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.



Use these settings to configure access of child elements using a generic structure.

Select types to access content of the dynamically named item:

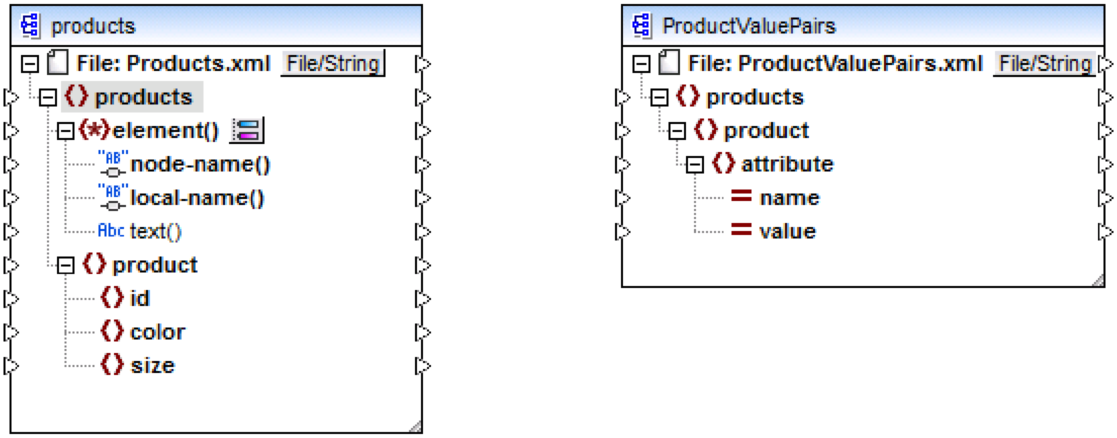
- ☐ productType
- ☐ xs:anyType
- ☑ text()

Selecting a type here will only make its structure available for mapping.
The actual type is not checked at runtime.

☐ Show name test nodes to filter or create elements by fixed node name

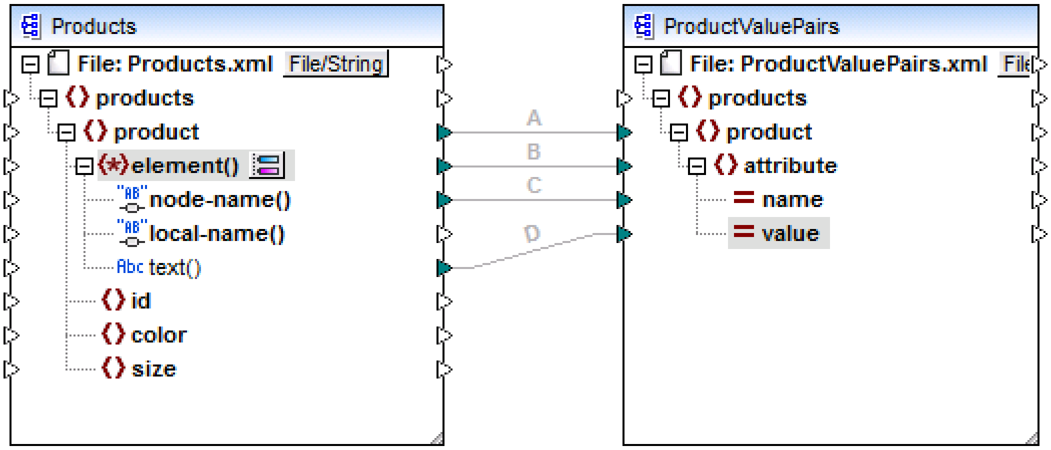
☑ Show schema child elements of parent element

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`.

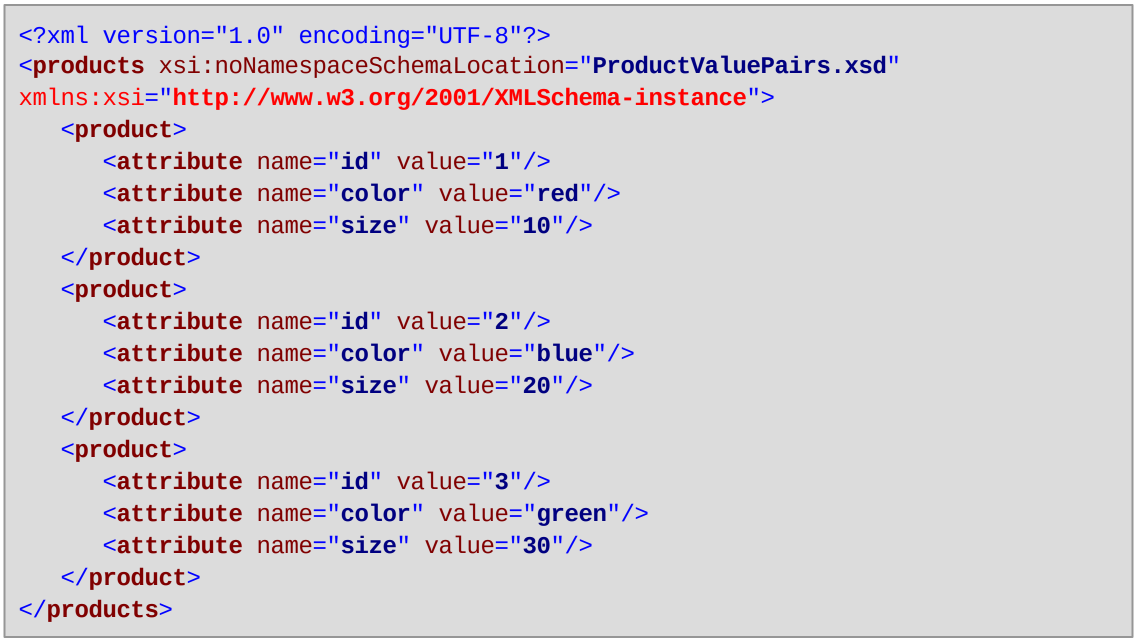
Connection B is a crucial connection in the mapping. To reiterate the goal of this connection, it carries a *sequence* of child elements of `product` from the source to the target. It does not carry the actual *names* or *values*. Therefore, it must be understood as follows: if the source **element()** 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
<?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

5.14.4 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 OS X 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\MapForce2018\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 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".

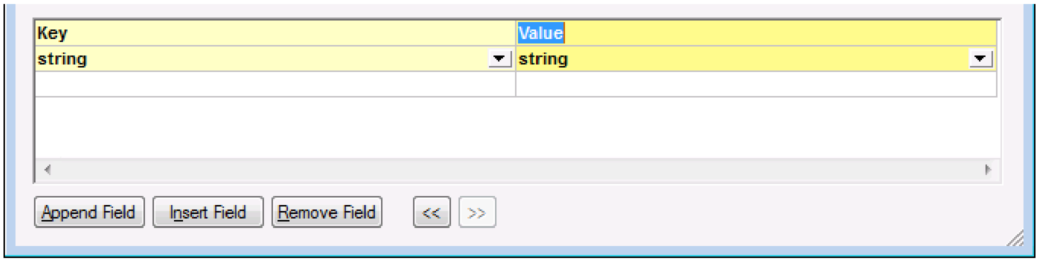
The following steps show how the required mapping can be created.

Step 1: Add the source XML component to the mapping

1. Set the mapping transformation language to BUILT-IN (see Selecting a Transformation Language).
2. On the **Insert** menu, click **XML Schema/File**, and browse for the following file: **<Documents>\Altova\MapForce2018\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

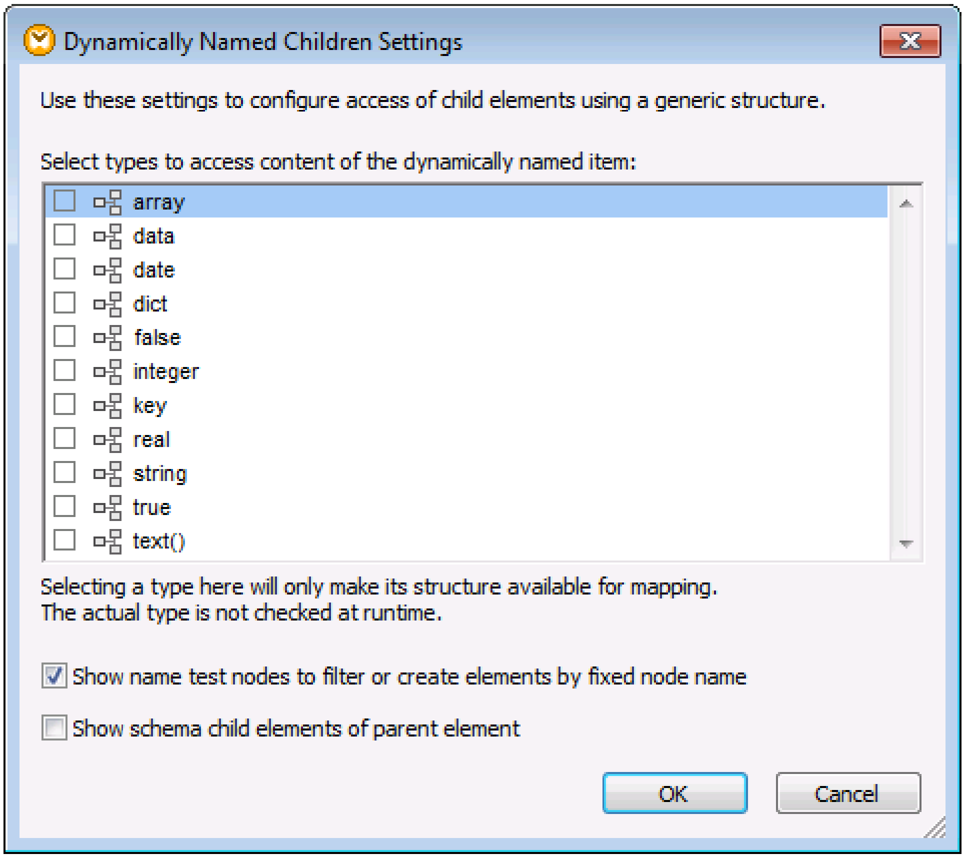
1. On the **Insert** menu, click **Text File**. When prompted, select the **Use simple processing for standard CSV...** option.
2. Add a CSV field to the component, by clicking **Append field**.
3. Double-click the name of each field, and enter "Key" as name of the first field, and "Value" as name of 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.



Key	Value
string	string

Step 3: Add the filter and functions

1. Drag the **`equal`**, **`exists`** and **`group-starting-with`** functions from the Libraries window into the mapping. For general information about functions, see Working with Functions.
2. To add the filter, click the **Insert** menu, and then click **Filter: Nodes/Rows**. For general information about filters, see Filters and Conditions.
3. On the **Insert** menu, click **Constant**, and then enter the text "string".
4. In the source component, right-click the `dict` node 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.



Dynamically Named Children Settings

Use these settings to configure access of child elements using a generic structure.

Select types to access content of the dynamically named item:

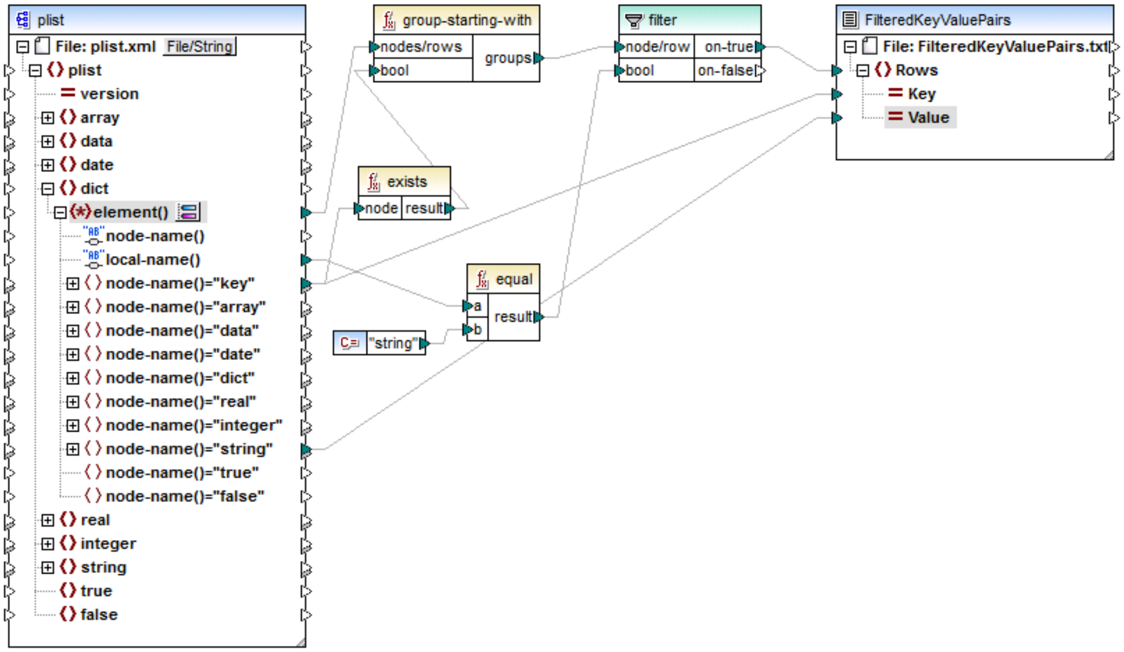
- □ array
- □ data
- □ date
- □ dict
- □ false
- □ integer
- □ key
- □ real
- □ string
- □ true
- □ text()

Selecting a type here will only make its structure available for mapping.
The actual type is not checked at runtime.

☑ Show name test nodes to filter or create elements by fixed node name

□ Show schema child elements of parent element

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.

5.15 Mapping Rules and Strategies

MapForce generally maps data in an intuitive way, but you may come across situations where the resulting output seems to have too many, or too few items. This topic is intended to help you avoid such mapping problems.

General rule

Generally, every connection between a source and target item means: for each source item, create one target item. If the source node contains simple content (for example, string or integer) and the target node accepts simple content, then MapForce copies the content to the target node and, if necessary, converts the data type.

This generally holds true for all connections, with the following exceptions:

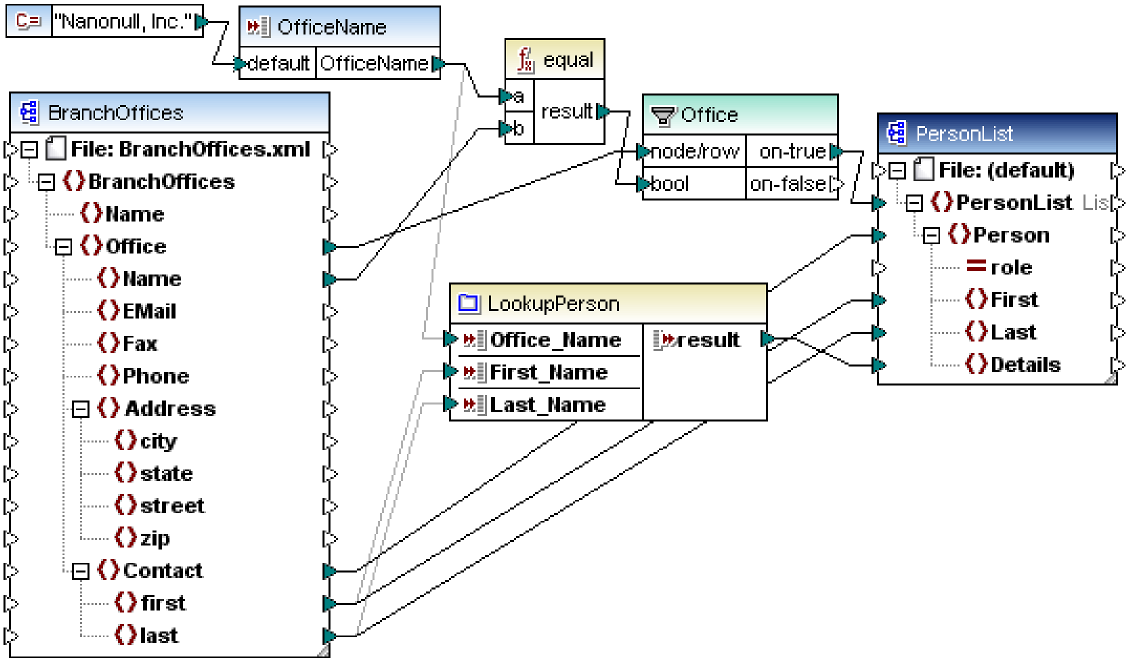
- A target XML root element is always created once and only once. If you connect a sequence to it, only the contents of the element will be repeated, but not the root element itself, and the result might not be schema-valid. If attributes of the root element are also connected, the XML serialization will fail at runtime, so you should avoid connecting a sequence to the root element. If what you want to achieve is creating multiple output files, connect the sequence to the "File" node instead, via some function that generates file names.
- Some nodes accept a single value, not a sequence (for example, XML attributes, database fields , and output components in user-defined functions).

The "context" and "current" items

MapForce displays the structure of a schema, database file as a hierarchy of mappable items in the component. Each of these nodes may have many instances (or none) in the instance file or database.

Example: If you look at the source component in **PersonListByBranchOffice.mfd**, there is only a single node **first** (under **Contact**). In the **BranchOffices.xml** instance file, there are multiple **first** nodes and **Contact** nodes having different content, under different **Office** parent nodes.

It depends on the current **context** (of the **target** node) which source nodes are actually selected and have their data copied, via the connector, to the target component/item.



PersonListByBranchOffice.mfd

This context is defined by the **current target node** and the connections to its ancestors:

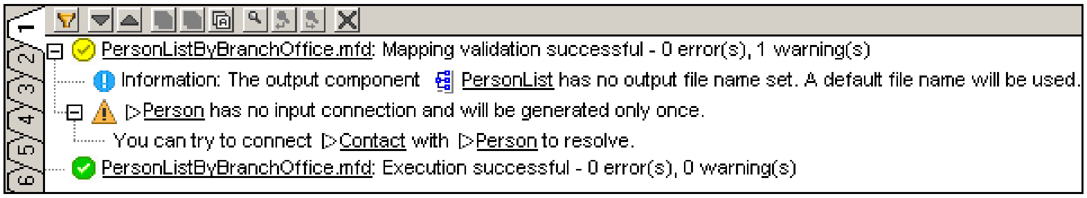
- Initially the context contains only the source components, but no specific nodes. When evaluating the mapping, MapForce processes the **target root** node first (PersonList), then works down the hierarchy.
- The connector to the **target** node is traced back to all source items directly or indirectly connected to it, even via functions that might exist between the two components. The source items and functions results are added to the context for this node.
- For each new target node a new context is established, that initially contains all items of the parent node's context. Target sibling nodes are thus independent of each other, but have access to all source data of their parent nodes.

Applied to the example mapping above (**PersonListByBranchOffice.mfd**):

- The connection from **Office** through the filter (Office) to **PersonList** defines a **single** office as the context for the whole target document (because PersonList is the root element of the target component). The office name is supplied by the input component, which has a default containing "Nanonull, Inc."
- All connections/data to the **descendants** of the root element PersonList, are automatically affected by the filter condition, because the selected single office is in the context.
- The connection from **Contact** to **Person** creates one target Person **per** Contact item of the source XML (general rule). For each Person one specific Contact is added to the context, from which the children of Person will be created.
- The connector from **first** to **First** selects the first name of the current Contact and writes it to the target item First.

Leaving out the connector from **Contact** to **Person** would create only **one** Person with multiple

First, Last, and Detail nodes, which is not what we want here. In such situations, MapForce issues a warning and a suggestion to fix the problem: "You can try to connect Contact with Person to resolve":



PersonListByBranchOffice.mfd: Mapping validation successful - 0 error(s), 1 warning(s)
Information: The output component PersonList has no output file name set. A default file name will be used.
▷Person has no input connection and will be generated only once.
You can try to connect ▷Contact with ▷Person to resolve.
PersonListByBranchOffice.mfd: Execution successful - 0 error(s), 0 warning(s)

Sequences

MapForce displays the structure of a schema, database file as a hierarchy of mappable items in the component.

Depending on the (target) context, each mappable item of a source component can represent:

- a **single instance** node of the assigned input file (or database)
- a **sequence** of zero to **multiple instance** nodes of the input file (or database)

If a sequence is connected to a **target** node, a loop is created to create as many target nodes as there are source nodes.

If a **filter** is placed between the sequence and target node, the bool condition is checked for each input node i.e. each item in the sequence. More exactly, a check is made to see if there is at least one bool in each sequence that evaluates to true. The priority context setting can influence the order of evaluation, see below.

As noted above, filter conditions automatically apply to all descendant nodes.

Note: If the source schema specifies that a specific node occurs exactly once, MapForce may remove the loop and take the first item only, which it knows must exist. This optimization can be disabled in the source Component Settings dialog box (check box "Enable input processing optimizations based on min/maxOccurs").

Function inputs (of normal, non-sequence functions) work similar to target nodes: If a sequence is connected to such an input, a loop is created around the function call, so it will produce as **many results** as there are items in the sequence.

If a sequence is connected to **more than one** such function input, MapForce creates nested loops which will process the **Cartesian product** of all inputs. Usually this is not desired, so only one single sequence with multiple items should be connected to a function (and all other parameters bound to singular current items from parents or other components).

Note: If an empty sequence is connected to such a function (e.g. concat), you will get an **empty sequence** as result, which will produce no output nodes at all. If there is no result in your target output because there is no input data, you can use the “substitute-missing” function to insert a substitute value.

Functions with **sequence inputs** are the only functions that can produce a result if the input sequence is **empty**:

- **`exists`**, **`not-exists`** and **`substitute-missing`** (also, **`is-not-null`**, **`is-null`** and **`substitute-null`**, which are aliases for the first three)

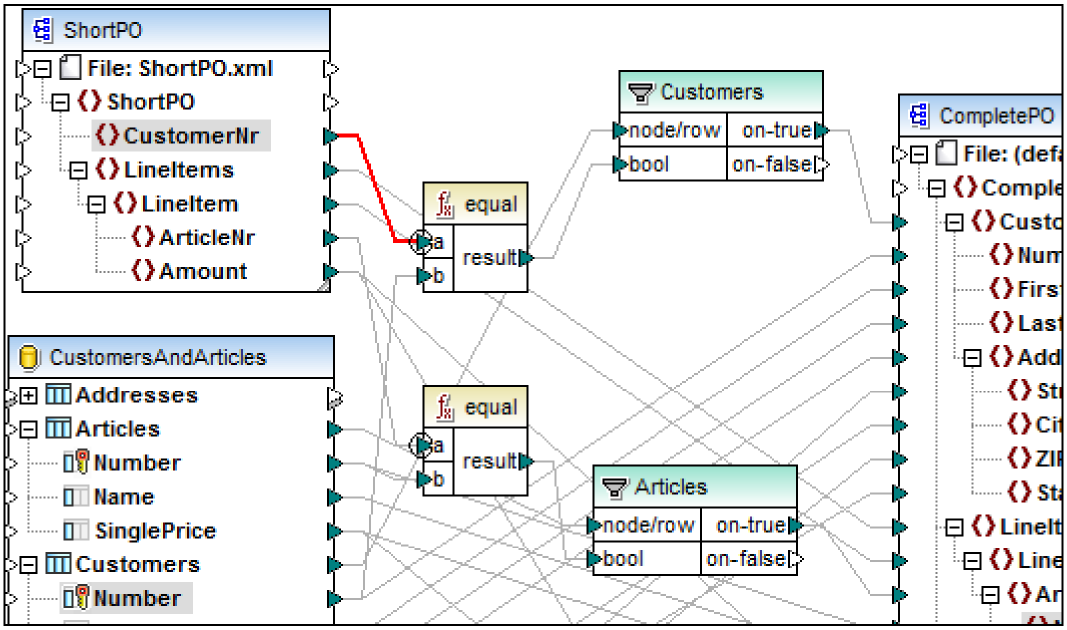
- aggregate functions (**`sum`**, **`count`**, etc.)
- regular user-defined functions that accept sequences (i.e. non-inlined functions)

The sequence input to such functions is always evaluated independently of the current target node in the context of its ancestors. This also means that any filter or SQL-Where components connected to such functions, do not affect any other connections.

Priority context

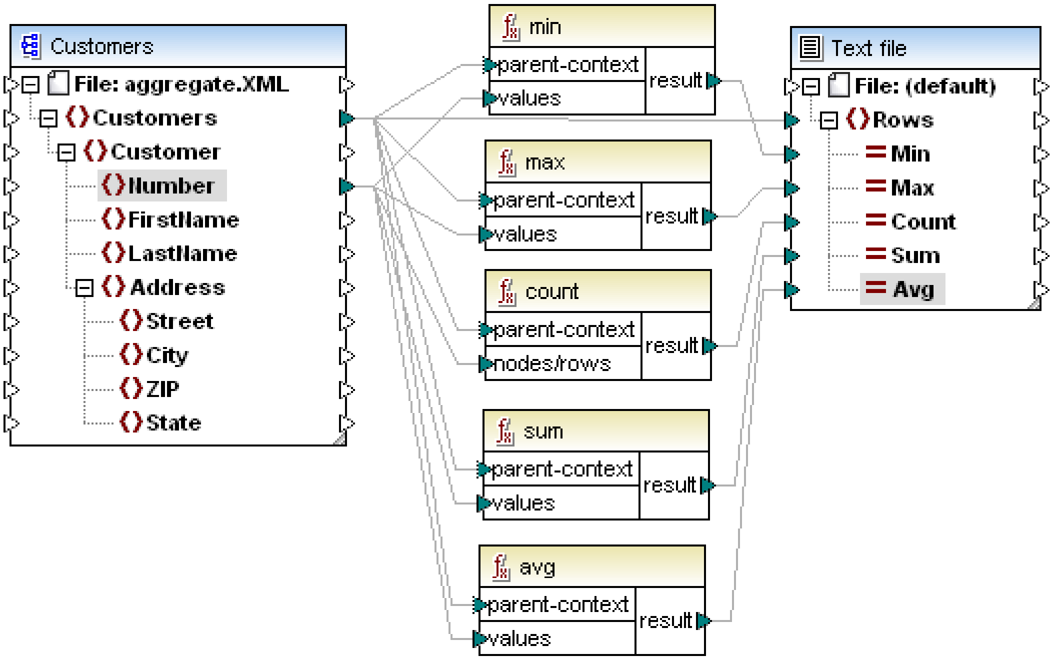
Usually, function parameters are evaluated from top to bottom, but its is possible to define one parameter to be evaluated before all others, using the **priority context** setting.

In functions connected to the bool input of **filter** conditions, the priority context affects not only the comparison function itself but also the evaluation of the filter, so it is possible to join together two source sequences (see CompletePO.mfd, CustomerNo and Number). See Priority Context node/item



Overriding the context

Some aggregate functions have an optional “parent-context” input. If this input is not connected, it has no effect and the function is evaluated in the normal context for sequence inputs (that is, in the context of the target node's parent).



If the `parent-context` input is connected to a source node, the function is evaluated for each `parent-context` node and will produce a separate result for each occurrence. See also Overriding the Mapping Context.

Bringing multiple nodes of the same source component into the context

This is required in some special cases and can be done with Intermediate variables.

5.15.1 Changing the Processing Order of Mapping Components

MapForce supports mappings that have several target components. Each of the target components has a preview button allowing you to preview the mapping result for that specific component.

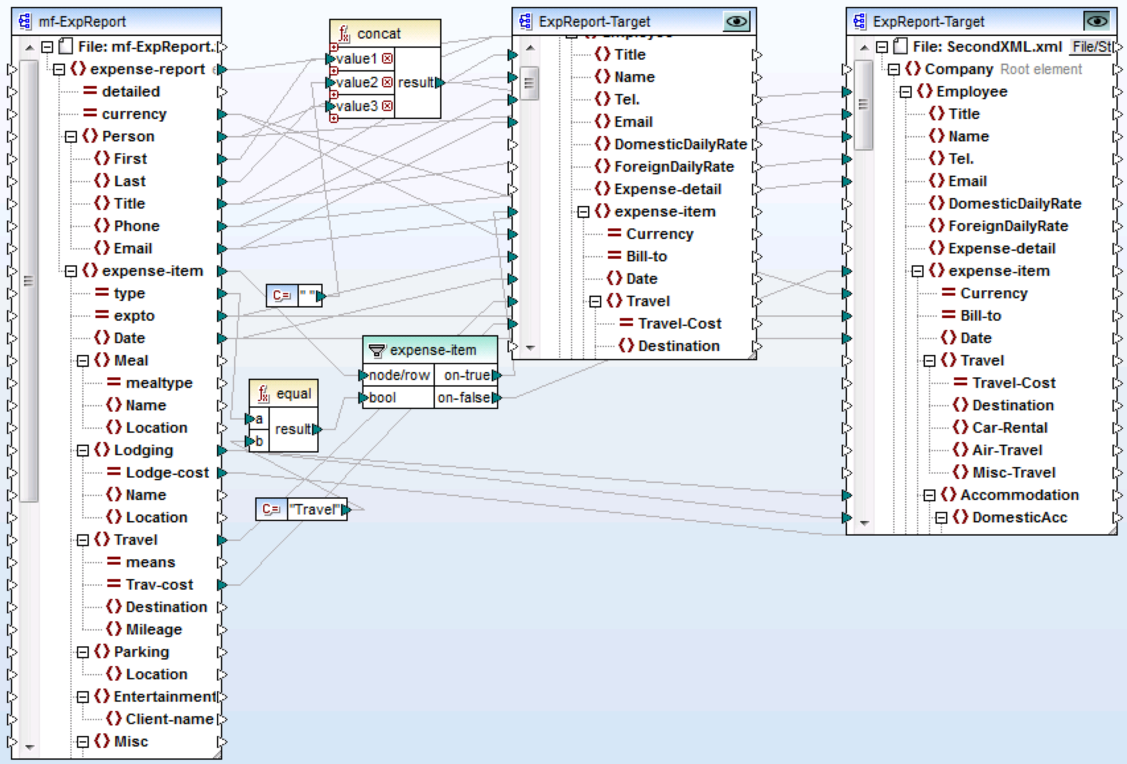
If the mapping is executed from the command line or from generated code, then, regardless of the currently active preview, the full mapping is executed and the output for each target component is generated.

The order in which the target components are processed can be directly influenced by changing the position of target components in the mapping window. The **position** of a component is defined as its top left corner.

Target components are processed according to their Y-X position on screen, from top to bottom and left to right.

- If two components have the same vertical position, then the leftmost takes precedence.
- If two component 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.

The screenshot below shows the tutorial sample **Tut-ExpReport-multi.mfd** available in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder. Both target components (ExpReport-Target) have the same **vertical** position, and the preview button is active on the right hand target component.

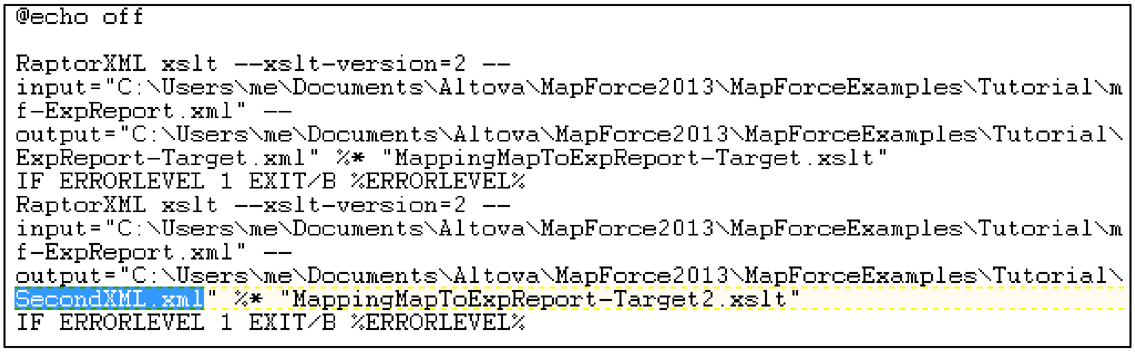


Tut-ExpReport-multi.mfd (MapForce Enterprise Edition)

Having selected XSLT2 and generated the code:

- The leftmost target component is processed first and generates the **ExpReport.xml** file.
- The component to the right of it is processed next and generates the **SecondXML.xml** file.

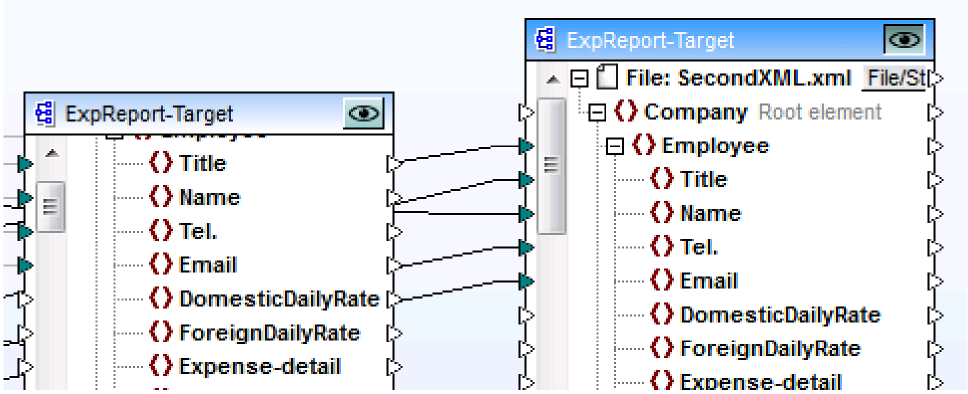
You can check that this is the case by opening the **DoTransform.bat** file (in the output folder you specified) and see the sequence the output files are generated. **ExpReport-Target.xml** is the first output to be generated by the batch file, and **SecondXML.xml** the second.



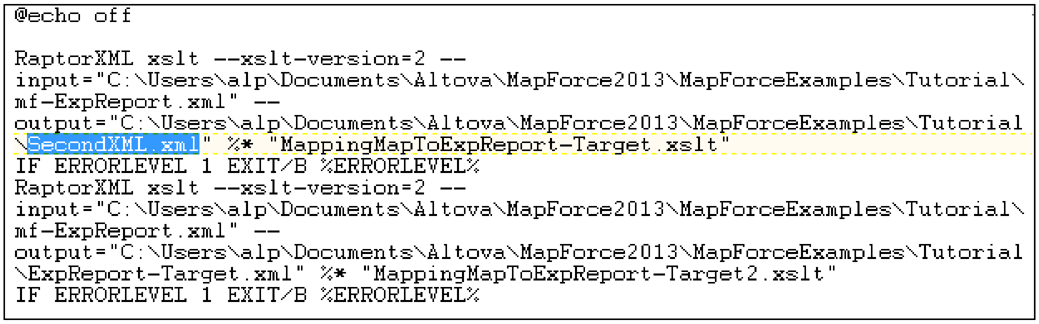
```
@echo off

RaptorXML xslt --xslt-version=2 --
input="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\m
f-ExpReport.xml" --
output="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
ExpReport-Target.xml" %* "MappingMapToExpReport-Target.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --
input="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\m
f-ExpReport.xml" --
output="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
SecondXML.xml" %* "MappingMapToExpReport-Target2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

Changing the mapping processing sequence:

1. Click the left target component and move it below the one at right.



2. Regenerate your code and take a look at the **DoTransform.bat** file.



```
@echo off

RaptorXML xslt --xslt-version=2 --
input="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
mf-ExpReport.xml" --
output="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial
\SecondXML.xml" %* "MappingMapToExpReport-Target.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --
input="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
mf-ExpReport.xml" --
output="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial
\ExpReport-Target.xml" %* "MappingMapToExpReport-Target2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

SecondXML.xml is now the first output to be generated by the batch file, and **ExpReport-Target.xml** the second.

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 component 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.

5.15.2 Priority Context node/item

When applying a function to different items in a schema or database, MapForce needs to know what the context node will be. All other items are then processed relative to this one. This is achieved by designating the item (or node) as the priority context.

Priority-context is used to prioritize execution when mapping unrelated items.

Mappings are always executed top-down; if you loop/search through two tables then each loop is processed consecutively. When mapping unrelated elements, without setting the priority context, MapForce does not know which loop needs to be executed first, it therefore automatically selects the first table, or data source.

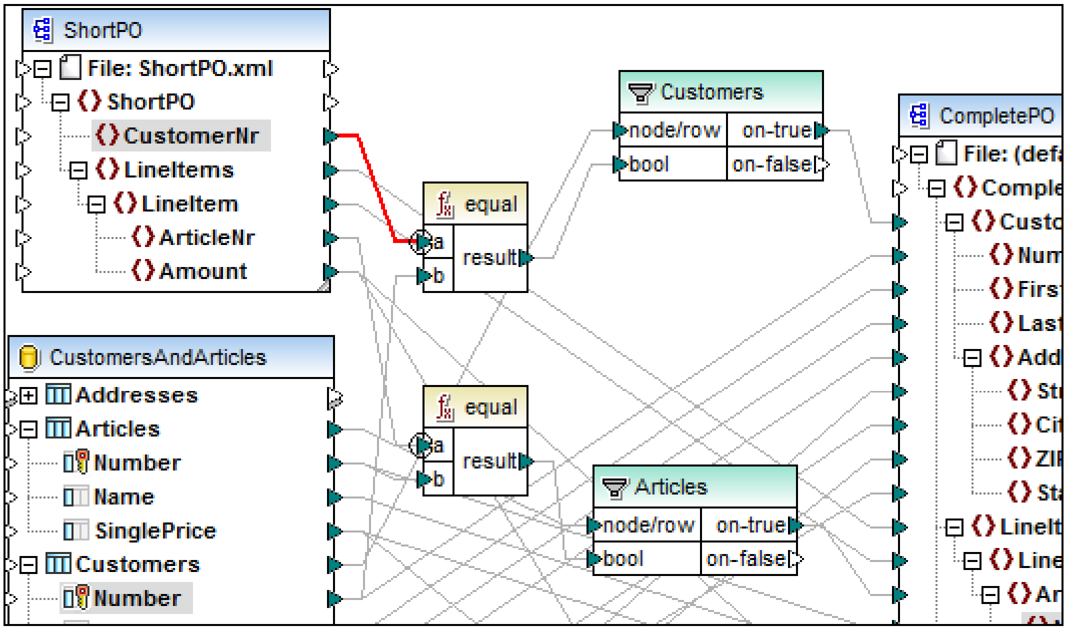
Solution:
Decide which table, or source data is to be looped/searched first, and then set the priority context on the connector to that table.

A simplified version of the complete **DB_CompletePO.mfd** file available in the **...\MapForceExamples** folder, is shown below.

Note that there are multiple source components in this example. **ShortPO** is a Schema with an associated XML instance file, while **CustomersAndArticles** is a database. The data from both are then mapped to the CompletePO schema / XML file. The priority context icon is enclosed in a circle as a visual indication.

Designating the **a** parameter of the equal function as the **priority context** would cause:

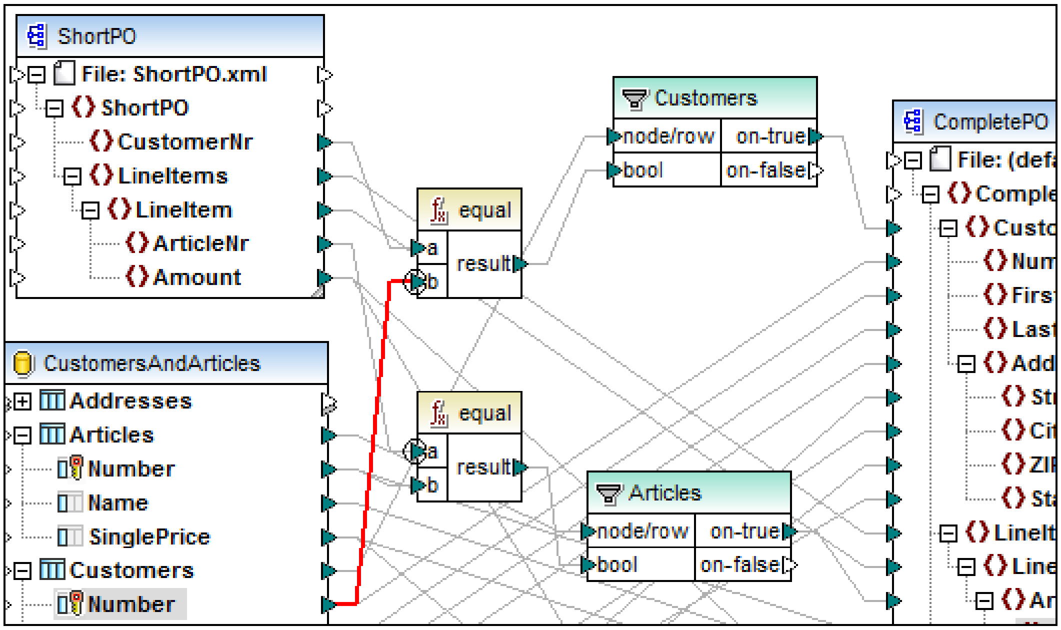
- The **CustomerNr** in ShortPO is compared with the item **Number** in the database.
- **CustomerNr** has been designated as the **priority context**, and is placed in the **a** parameter of the equal function.
- The **CustomersAndArticles** database is then searched (**once**) for the **same** number. The **b** parameter contains the Number item from the database.
- If the number is found, then the result is passed to the **bool** parameter of the **filter** component (Customers).
- The **node/row** parameter passes on the **Customer** data to "on-true" when the bool parameter is true, i.e. when the same number has been found.
- The rest of the customer data is then passed on as: Number, FirstName, LastName items, are all connected to the corresponding items in the target schema.



This means that the database is only searched once per CustomerNr supplied by ShortPO.

Designating the **b** parameter of the equal function as the **priority context** would cause:

- MapForce to search and load the first Number into the **b** parameter from the database
- Check against the **CustomerNr** in the **a** parameter of ShortPO
- If not equal, search through all CustomerNr of ShortPO
- Search the database and load the next Number into **b**, check against a, and
- Iterate through every Number in the database while trying to find that number in ShortPO.

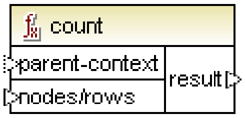


This means that a database query is generated for each Number and the result is then compared to every CustomerNr of ShortPO.

Priority context and user-defined functions:
If a user-defined function has been defined of type "inline", the default setting, then a priority context cannot be defined on one of the parameters of the user-defined function. The user-defined function can, of course, contain other regular (non-inlined) user-defined functions which have priority contexts set on their parameters.

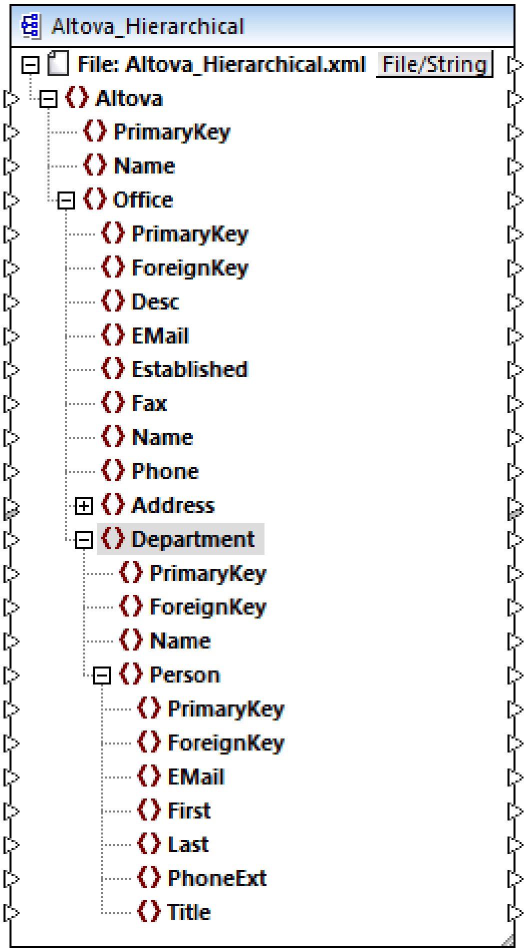
5.15.3 Overriding the Mapping Context

In some mappings, in order to achieve the desired mapping output, it may be necessary to override the mapping context. For this reason, some components provide an optional `parent-context` item in their structure which enables you to influence the mapping context if so required. Examples of such components are: aggregate functions, variables, and Join components.



An aggregate function with optional parent-context

To understand why the mapping context is important, let's add to the mapping an XML file that contains nested nodes with multiple levels. On the **Insert** menu, click **XML Schema/File**, and browse for the file: **<Documents>\Altova\MapForce2018\MapForceExamples\Altova_Hierarchical.xml**.

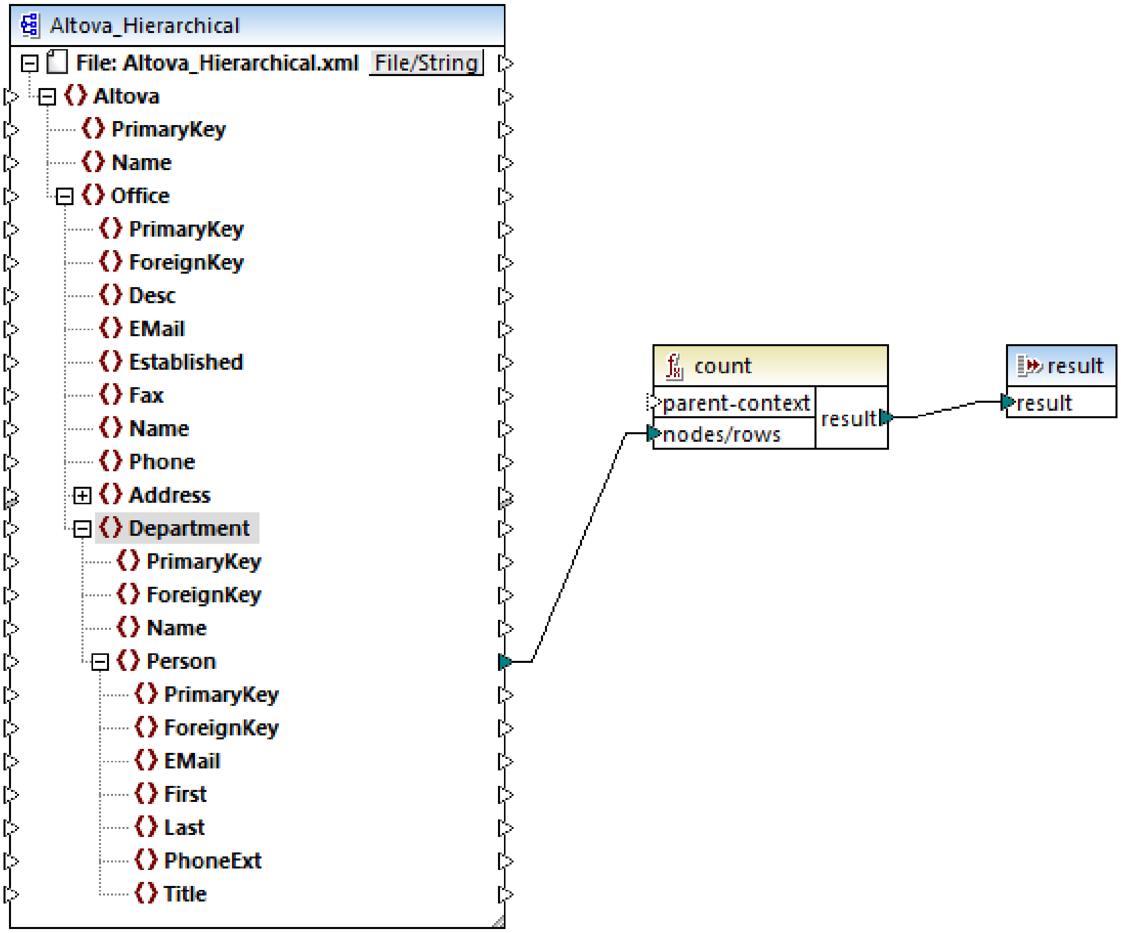


Altova_Hierarchical.xml

Importantly, in the XML file above, the `Office` parent node contains multiple `Department` nodes, and each `Department` contains multiple `Person` nodes. If you open the actual XML file in an XML editor, you can see that the distribution of people by office and department is as follows:

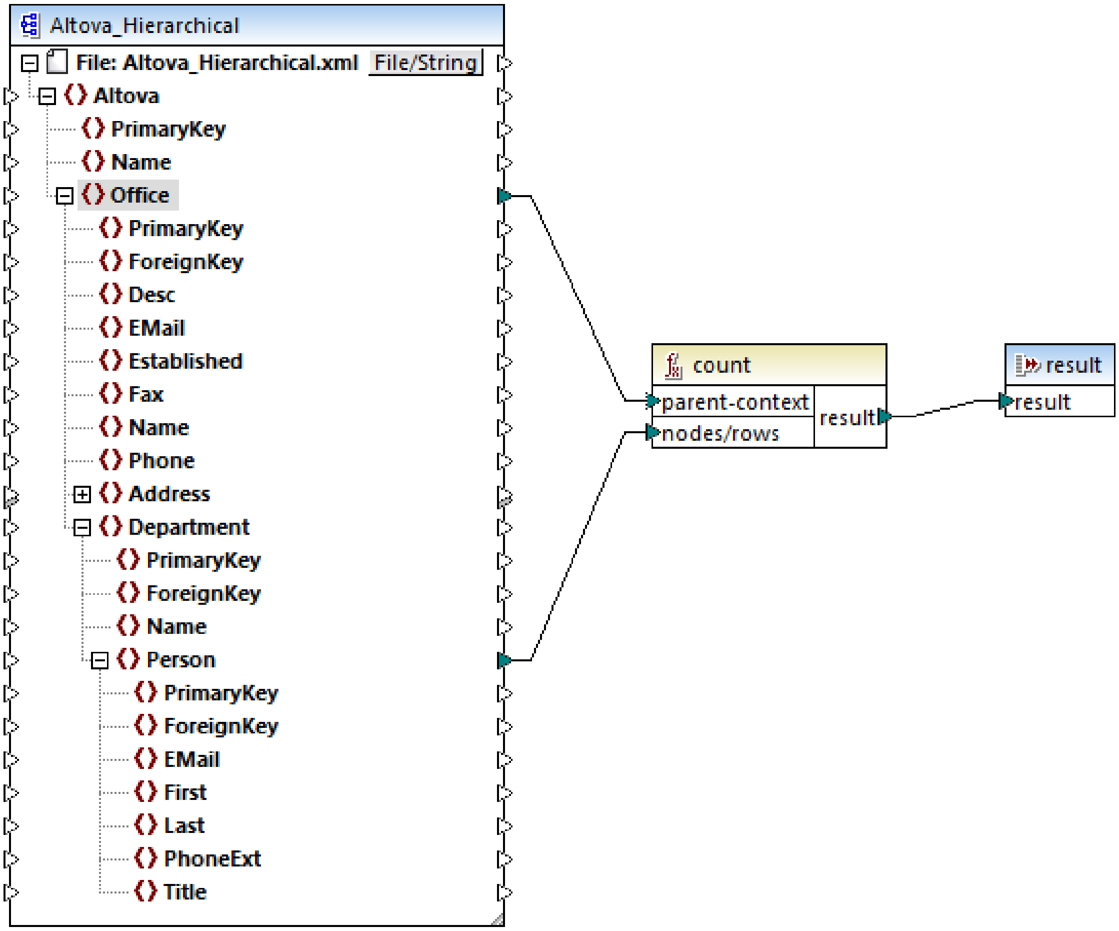
Office	Department	Number of people
Nanonull, Inc.	Administration	3
	Marketing	2
	Engineering	6
	IT & Technical Support	4
Nanonull Partners, Inc.	Administration	2
	Marketing	1
	IT & Technical Support	3

Now let's assume that your mapping should count all people in all departments. To achieve this requirement, you can add the **count** function from core | aggregate functions and map data as follows:



If you preview the mapping at this stage, the output is `21`, which corresponds to the total number of people in all departments. Notice that the **count** function includes an optional `parent-context` item, which so far has not been connected. As a result, the parent context of the **count** function is the default root node of the source component (which, in this case, is the `Altova` item). This means that all the persons, from all departments, are considered for the scope of the **count** function. This is the way the mapping context works by default, as outlined in Mapping Rules and Strategies, and this is sufficient in most mapping scenarios.

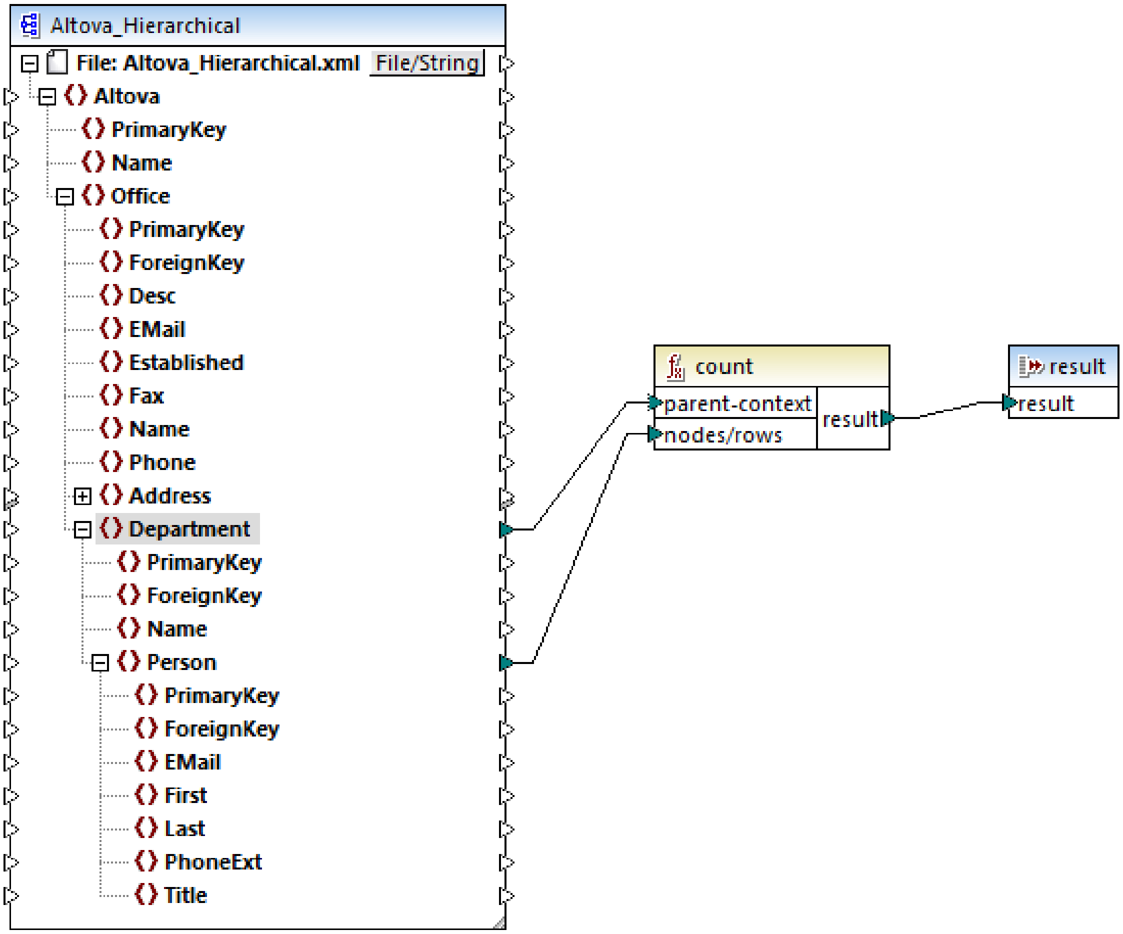
However, it is possible to override the default mapping context if necessary. To do this, add a connection from the `Department` node to the `parent-context` item as shown below.



By changing the mapping as shown above, you are instructing the mapping to iterate over people records *in the context of each office*. Therefore, if you preview the mapping now, the output will be `15*`. This is exactly the number of people in the first office, "Nanonull, Inc.". The explanation is that this time the people nodes were counted twice (once for each office). The count of people in each office was 15 and 6, respectively. However, only the first result was returned (because the function cannot return a sequence of values, only a simple value).

** Assuming that the target language of the mapping is other than XSLT 1.0.*

You can further modify the mapping so as to change the mapping context to `Department`, as shown below. This time the people records would be counted in the context of each department (that is, 7 times, which corresponds to the total number of departments). Again, only the first of the results is returned, so the mapping output is `3`, which corresponds to the number of people in the first department of the first office.



While this mapping is not doing much yet, its point is to illustrate how the `parent-context` item influences the output of the mapping. Having this in mind, you can override the `parent-context` in other mappings, such as those that contain variables or Join components. See also Example: Grouping and Subgrouping Records.

Chapter 6

Debugging Mappings

6 Debugging Mappings

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.

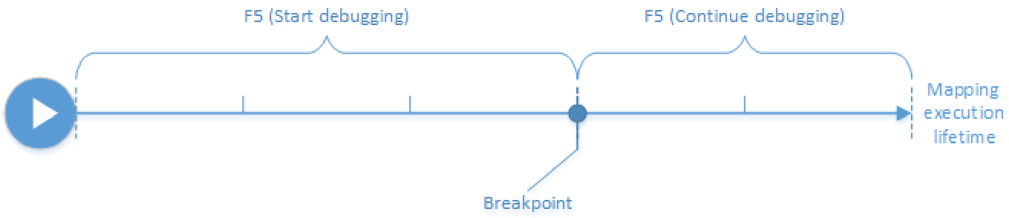
The mapping debugger is available when the transformation language of the mapping is BUILT-IN. If you start debugging a mapping designed for a different language, you will be prompted to change the mapping language to BUILT-IN. You can also convert a mapping to BUILT-IN by selecting the menu command **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.

Debug with breakpoints

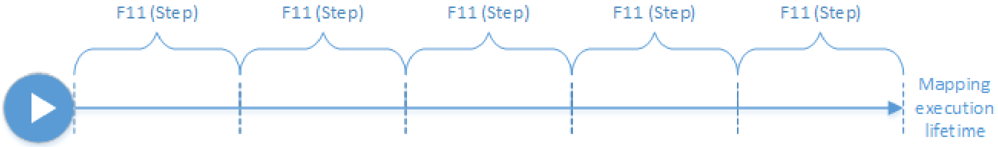
When you need to stop the debugging execution at a particular place in the mapping, you can set breakpoints, similar to how you would do that in a traditional development environment. The difference is that breakpoints are added not to a line of code, but to an input or output connector of a mapping 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.

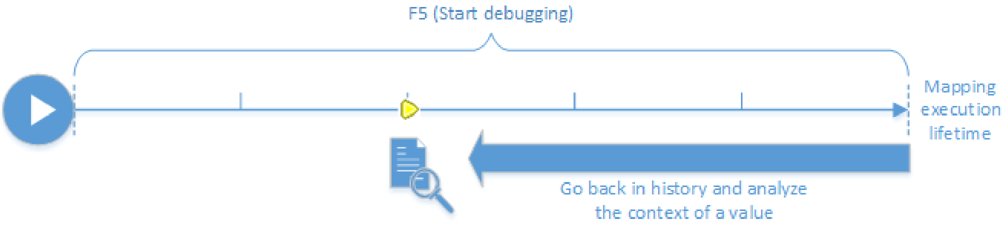
Debug step-by-step

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.



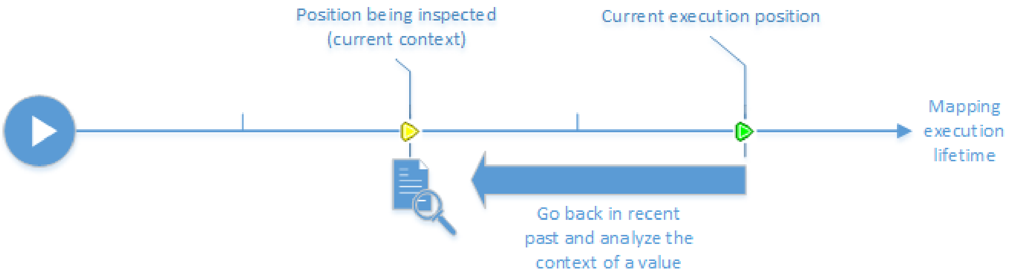
Analyze the log of values

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.


F5 (Start debugging)
Mapping execution lifetime
Go back in history and analyze the context of a value

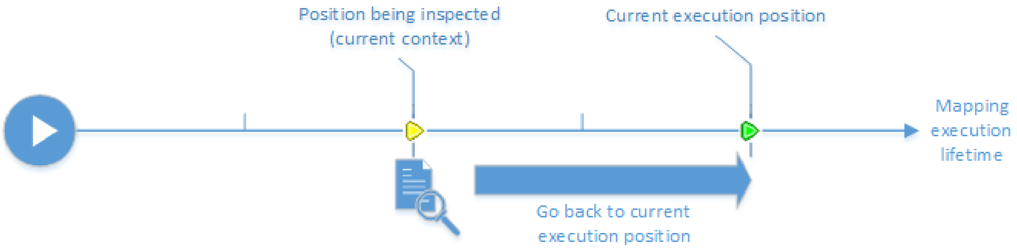
Set the context to a value related to the current execution position

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.

6.1 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 Settings)

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 Mappings), 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 Breakpoints).

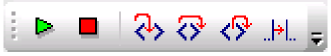
6.2 Debugger Commands

You can access the debugger commands as follows:

- In the **Debug** menu
- As keyboard shortcuts
- In the Debug toolbar.

Menu Command	Keyboard Shortcut	Toolbar button	Description
Debug | Start debugging	**F5**	▶	Starts or continues debugging until a breakpoint is hit or the mapping finishes.
Debug | Stop debugging	**Shift + F5**	🟥	Stops debugging. This command exits the debug mode and switches MapForce back to standard mode.
Debug | Step Into	**F11**		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 translates into "go to the left/go to target child/go to source parent".
Debug | Step Over	**F10**		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.
Debug | Step Out	**Shift + F11**		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 translates into "go to the right/go to target parent/go to source child".
Debug | Minimal Step	**Ctrl + F11**		Continues execution until a value is produced or consumed. This command subdivides a step and will typically stop twice for each connection: once when its source produces a value and once when its target consumes it. MapForce does not

Menu Command	Keyboard Shortcut	Toolbar button	Description
			necessarily compute values in the order the mapping would suggest, so production and consumption events do not always follow each other.



Debug toolbar

6.3 About the Debug Mode

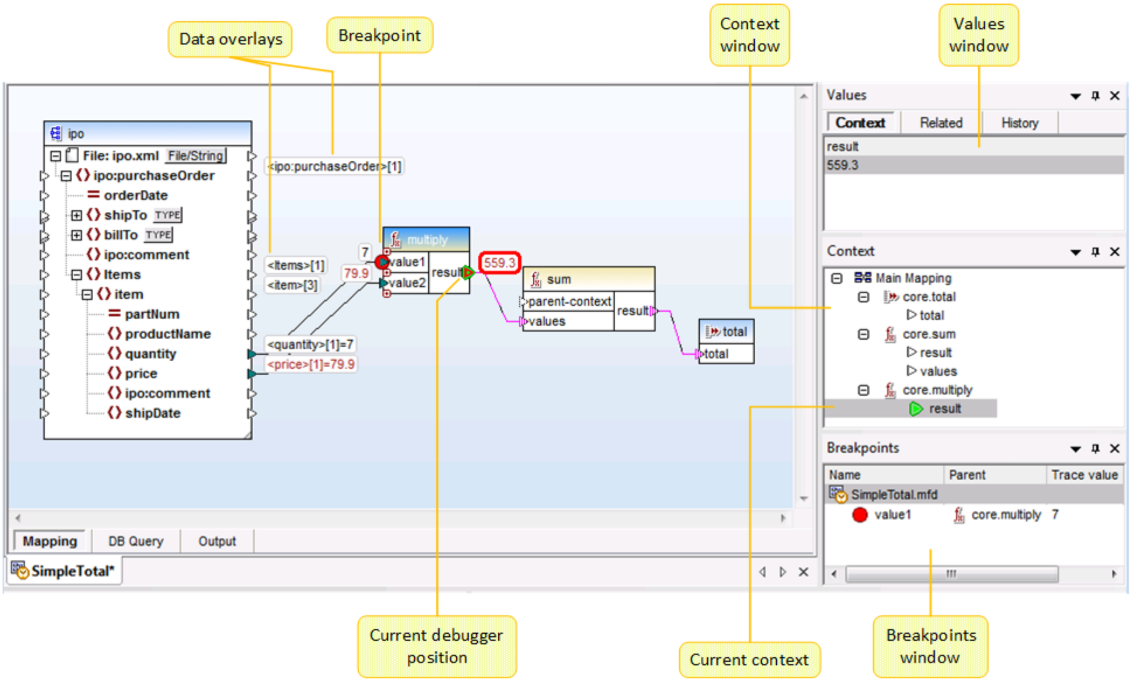
When you start debugging (by pressing **F5**, or **F11**, or **Ctrl + F11**), MapForce executes the mapping in debug mode.

> 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\MapForce2018\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.

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).

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).
▷	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).

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.

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.

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.

6.4 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.

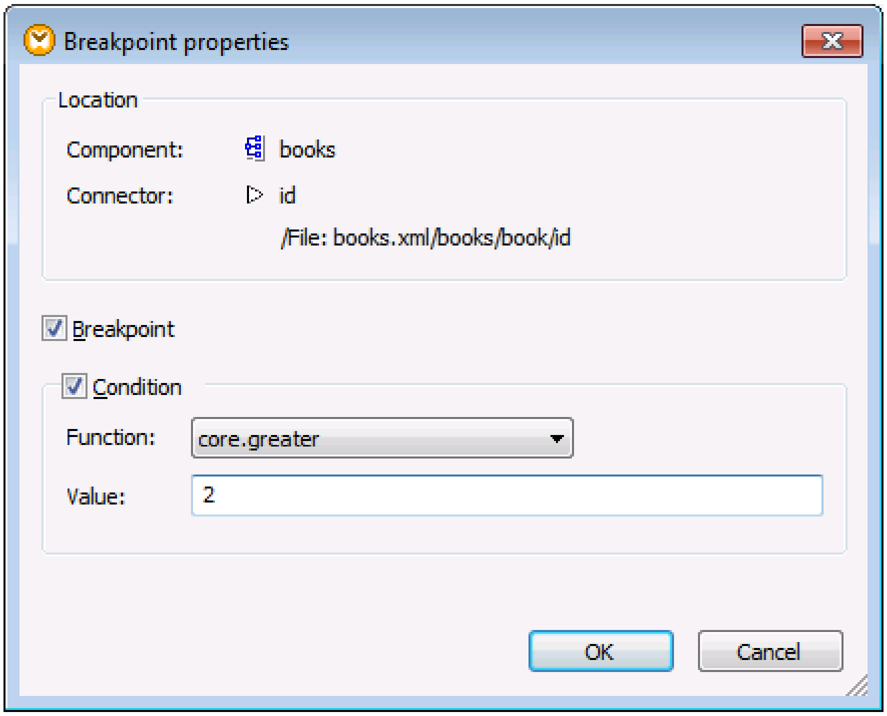
Breakpoints can be simple or conditional. Simple breakpoints stop the mapping execution unconditionally. Conditional breakpoints stop the mapping execution only when the condition assigned to them is satisfied. Conditions take the form of MapForce built-in library functions to which you supply custom 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**.



Breakpoint properties

Location

Component: books

Connector: ▷ id

/File: books.xml/books/book/id

☑ Breakpoint

☑ Condition

Function: core.greater

Value: 2

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.

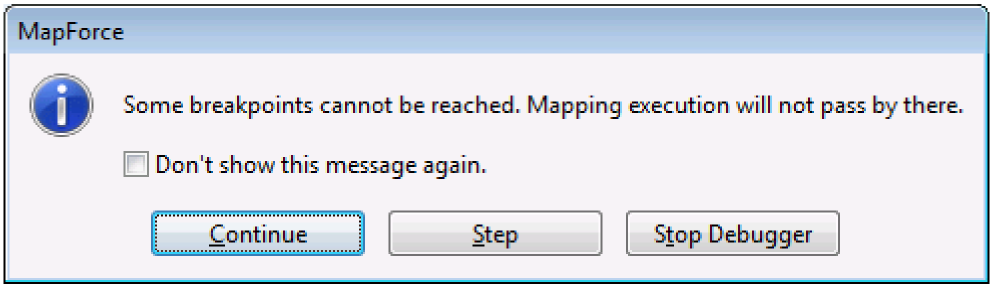
Removing breakpoints

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).

Unreachable breakpoints

There may be cases when MapForce displays a "Breakpoints cannot be reached" message:


MapForce

Some breakpoints cannot be reached. Mapping execution will not pass by there.

□ Don't show this message again.

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).

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.

6.5 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:

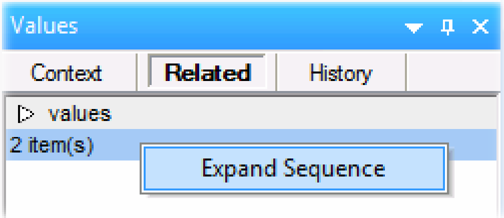
▷	Green is "the present"; it indicates the current execution position (see Viewing the Current Value of a Connector).
▷	Yellow is "the past"; it indicates that you are looking at some connector in the past, relative to the current execution position. This may happen after you set a context manually (see Setting the Context to a Value).

The "Related" tab

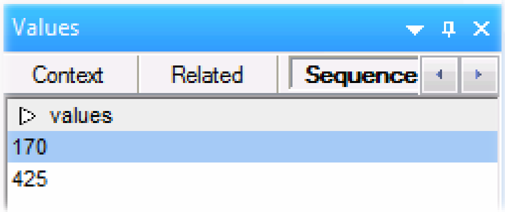
The **Related** tab displays values that are related to (or represent the "near past" of) the currently processed value. Normally, you do need to explicitly click this tab; MapForce switches to it automatically when you click the data overlay of a connector that is related to the current execution position of the debugger. See Stepping back into Recent Past.

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.



170
425

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.

6.6 Using the Context Window

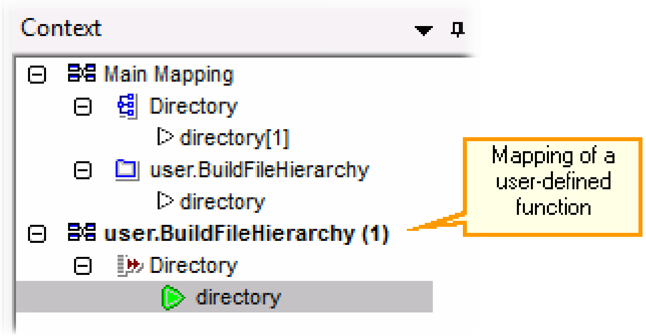
While MapForce is in debug mode, the Context window displays a structure of connectors that are relevant to the current position of the debugger. In other words, it provides the mapping context responsible for producing the current mapping value.

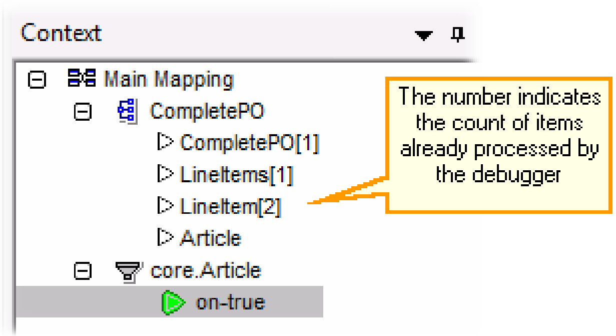
MapForce builds the current context as follows:

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:

Icon	Description
	Represents the mapping to which the context belongs. This can be either the main mapping or the mapping of a user-defined function.  Context Main Mapping Directory ▷ directory[1] user.BuildFileHierarchy ▷ directory **user.BuildFileHierarchy (1)** Directory directory Mapping of a user-defined function
▷	Represents a connector. The target nodes processed so far have their position displayed in square brackets.

Icon	Description
	 Context Main Mapping CompletePO CompletePO[1] LineItems[1] LineItem[2] Article core.Article on-true The number indicates the count of items already processed by the debugger
▶	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).

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:

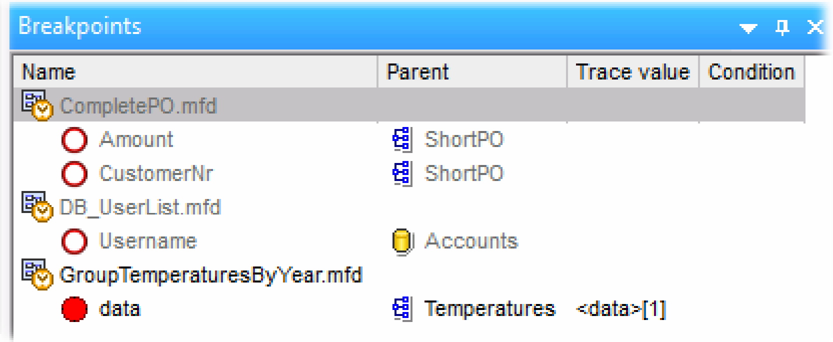
1. From the target to the output connector of the user-defined function to the input connector of the user-defined function
2. From there further to the left.

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.

6.7 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. To make the Breakpoints window visible at all times, select the menu command **View | Debug Windows | Breakpoints**.

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.



Name	Parent	Trace value	Condition
CompletePO.mfd			
○ Amount	ShortPO		
○ CustomerNr	ShortPO		
DB_UserList.mfd			
○ Username	Accounts		
GroupTemperaturesByYear.mfd			
● data	Temperatures	<data>[1]	

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.

Icon	Description
○	Inactive breakpoint. Denotes a breakpoint from a mapping that is open, but is not currently being debugged.
⚠	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**.

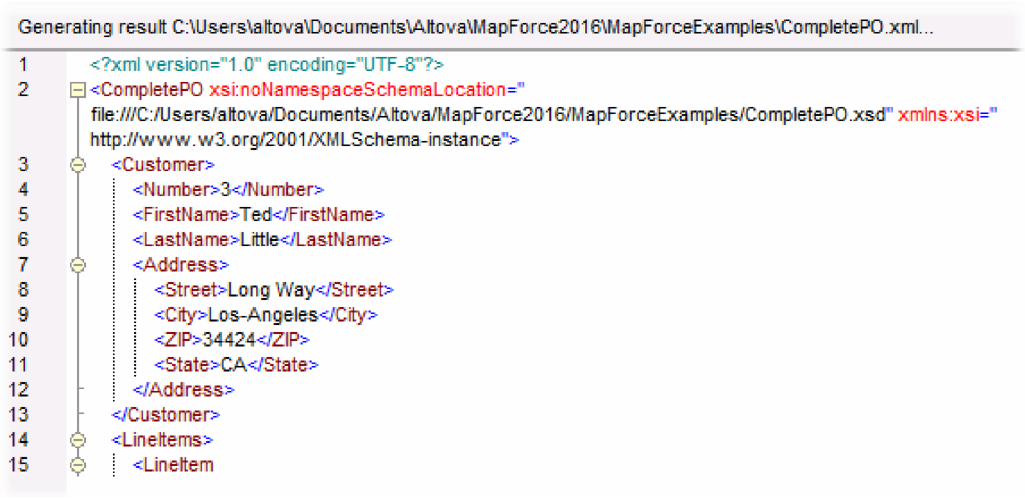
The context command **Delete All Breakpoints** removes all breakpoints displayed in the Breakpoints window, regardless of the mapping where they belong.

See also: Adding and Removing Breakpoints

6.8 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
```

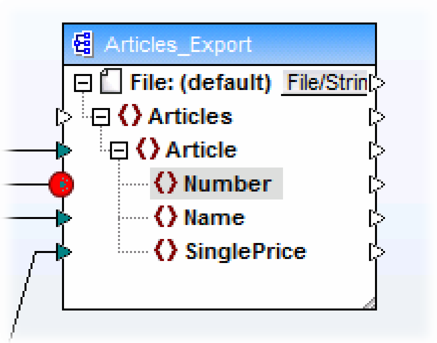
Limitations

- 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.

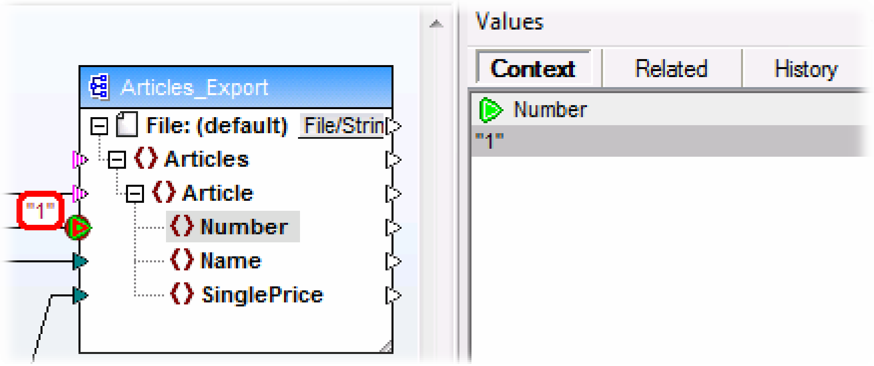
6.9 Viewing the Current Value of a Connector

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).

To understand this case, open the **PreserveFormatting.mfd** sample from the **<Documents>\Altova\MapForce2018\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.



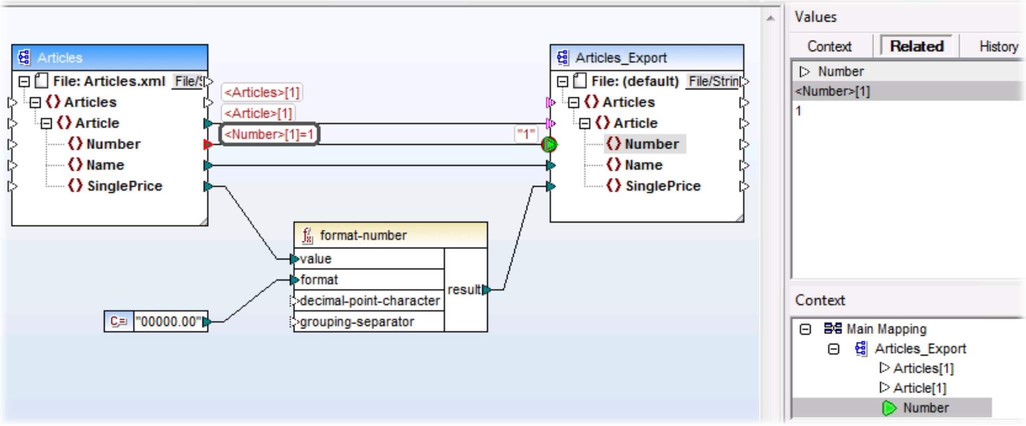
As shown in the image, the current debugger position ▷ (and the breakpoint ●) is on the `Number` node of the target component. The Values window indicates that this node processes the value "1" (this value is also highlighted with a thick red border on the mapping).

6.10 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\MapForce2018\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).

6.11 Viewing the History of Values Processed by a Connector

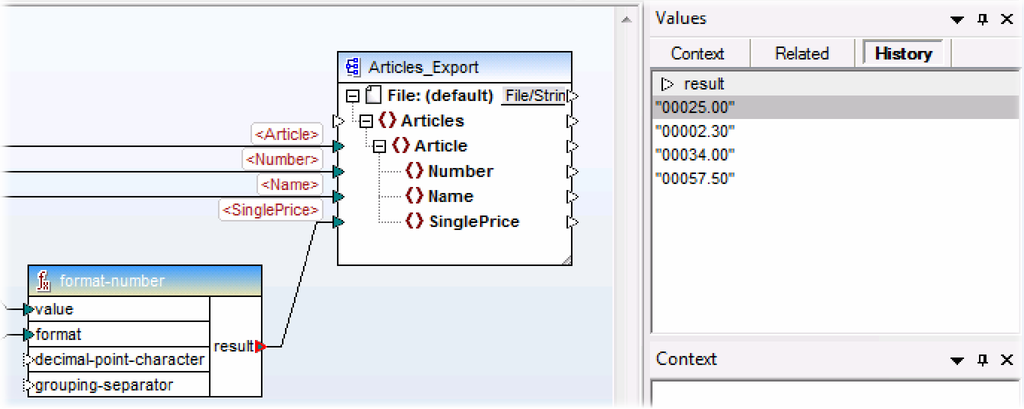
If the option **Keep full trace history** is enabled (see Debugger Settings), you can view the history of all values that were processed by that connector (up to the current execution position).

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\MapForce2018\MapForceExamples\** directory. If it is already open, make sure to do the following:

- Clear any breakpoints, if such exist (see Adding and Removing Breakpoints)
- 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.

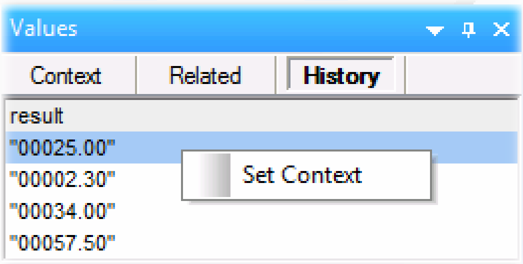


result
"00025.00"
"00002.30"
"00034.00"
"00057.50"

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).

6.12 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), 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.



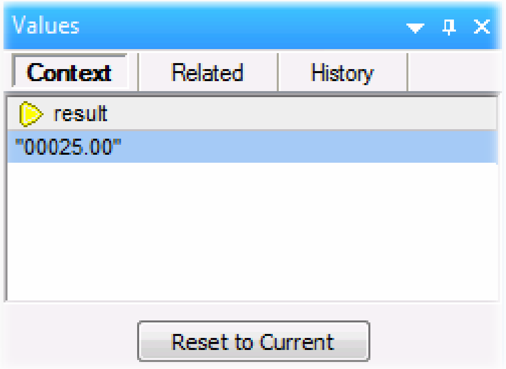
```
result
"00025.00"
"00002.30"
"00034.00"
"00057.50"
```

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. For information about the context itself, see Using the Context Window.

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.



```
result
"00025.00"
```

6.13 Debugger Settings

To access the settings applicable to the MapForce debugger, select the menu command **Tools | Options**, and then click **Debugger**. The available settings are as follows:

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 by MapForce 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.

Chapter 7

Data Sources and Targets

7 Data Sources and Targets

This section provides information specific to various source and target component types that MapForce can map from or to:

- XML and XML Schema
- Databases
- CSV and Text Files
- HL7 Version 3

7.1 XML and XML schema

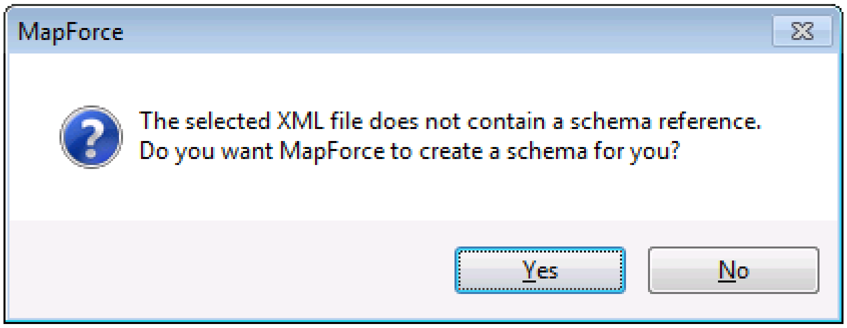
Altova website: XML mapping

In the introductory part of this documentation, you have seen examples of simple mappings that use XML and XML schema files as source or target components. This section provides further information about using XML components in your mappings. It includes the following topics:

- XML Component Settings
- Using DTDs as "schema" components
- Derived XML Schema types - mapping to
- QName support
- Nil Values / Nillable
- Comments and Processing Instructions
- CData sections
- Wildcards - xs:any

7.1.1 Generating an XML Schema

MapForce can automatically generate an XML schema based on an existing XML file if the XML Schema is not available. Whenever you add to the mapping area an XML file without a schema (using the menu command **Insert | XML Schema/File**), the following dialog box appears.



The selected XML file does not contain a schema reference.
Do you want MapForce to create a schema for you?

Click **Yes** to generate the schema, you will then be prompted to select the directory where the generated schema should be saved.

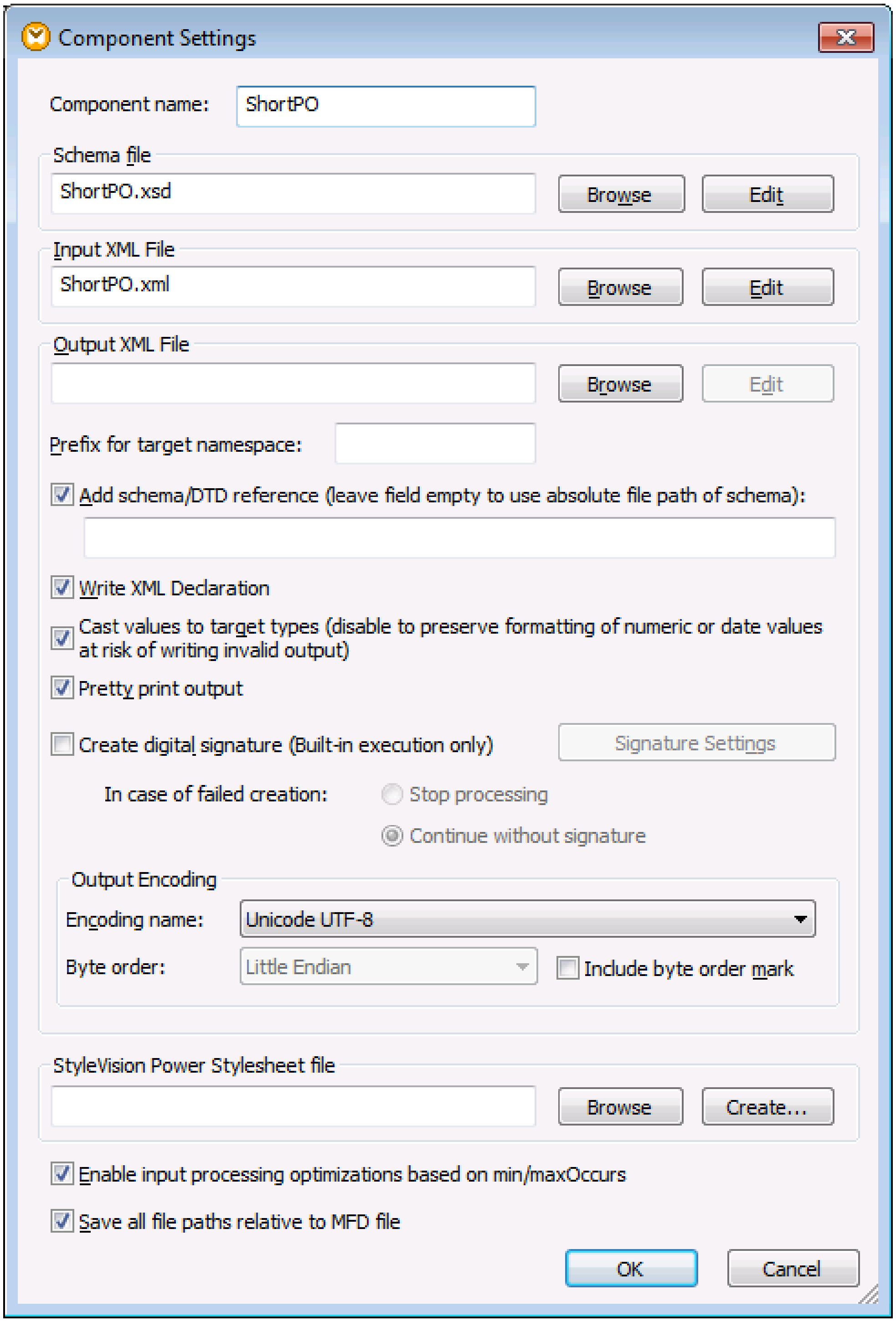
When MapForce generates a schema from an XML file, data types for elements/attributes must be inferred from the XML instance document and may not be exactly what you expect. It is recommended that you check whether the generated schema is an accurate representation of the instance data.

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.

7.1.2 XML Component Settings

After you add an XML 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

Component name: ShortPO

Schema file
ShortPO.xsd

Input XML File
ShortPO.xml

Output XML File

Prefix for target namespace:

☑ Add schema/DTD reference (leave field empty to use absolute file path of schema):

☑ Write XML Declaration

☑ Cast values to target types (disable to preserve formatting of numeric or date values at risk of writing invalid output)

☑ Pretty print output

☐ Create digital signature (Built-in execution only)

In case of failed creation: ○ Stop processing

● Continue without signature

Output Encoding

Encoding name: Unicode UTF-8

Byte order: Little Endian ☐ Include byte order mark

StyleVision Power Stylesheet file

☑ Enable input processing optimizations based on min/maxOccurs

☑ Save all file paths relative to MFD file

XML 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 (for example, "Source XML File") and full stop characters (for example, "Orders.EDI"). 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.
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 the new file. To edit the file in 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 to it an XML instance file. 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 location of the file, click **Browse** and select the new file. To edit the file in 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 location of the file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.
Prefix for target namespace	Allows you to enter a prefix for the target namespace. Ensure that the target namespace is defined in the target schema, before assigning the prefix.
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 files in the `xsi:schemaLocation` attribute, or into the `DOCTYPE`

<table>
<tr><td></td><td>declaration if a DTD is used.

Note that, 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 <b>http://</b> address as well as an absolute or relative path in this field.

Deactivating this option allows you to decouple the XML instance from the referenced XML Schema or DTD (for example, if you want to send the resulting XML output to someone who does not have access to the underlying XML Schema).</td></tr>
<tr><td><i>Write XML declaration</i></td><td>This option enables you to suppress the XML declaration from the generated output. By default, the option is enabled, meaning that the XML declaration is written to the output.

This feature is supported as follows in MapForce target languages and execution engines.

<table>
<tr><th>Target language / Execution engine</th><th>When output is a file</th><th>When output is a string</th></tr>
<tr><td>Built-in</td><td>Yes</td><td>Yes</td></tr>
<tr><td>MapForce Server</td><td>Yes</td><td>Yes</td></tr>
<tr><td>XSLT, XQuery</td><td>Yes</td><td>No</td></tr>
<tr><td>Code generator (C ++, C#, Java)</td><td>Yes</td><td>Yes</td></tr>
</table>
</td></tr>
<tr><td><i>Cast values to target types</i></td><td>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 <b>string</b> 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.</td></tr>
</table>

	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.
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. 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.
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 Using Relative Paths on a Component.

7.1.3 Using DTDs as "Schema" Components

Starting with MapForce 2006 SP2, namespace-aware DTDs are supported for source and target components. The namespace-URIs are extracted from the DTD "xmlns"-attribute declarations, to make mappings possible.

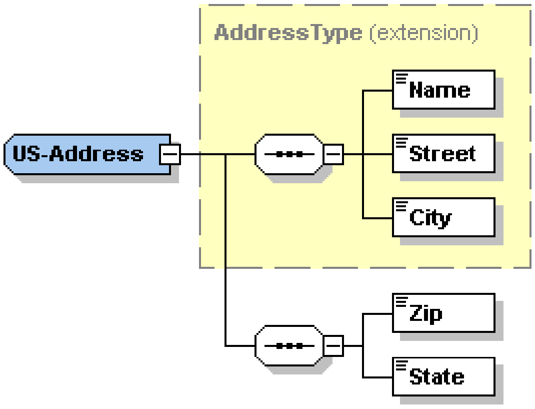
However, some DTDs contain xmlns*-attribute declarations without namespace-URIs (for example, DTDs used by StyleVision). Such DTDs have to be extended to make them useable in MapForce. Specifically, you can make such DTDs useable by defining the xmlns-attribute with the namespace-URI, as shown below:

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

7.1.4 Derived XML Schema Types

MapForce supports the mapping to/from derived types of a complex type. Derived types are complex types of an XML Schema that use the **xsi:type** attribute to identify the specific derived types.

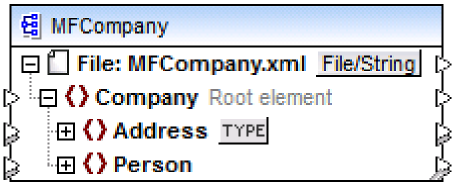
The screenshot below shows the definition of a derived type called `US-Address`, in XMLSpy. The base type (or originating complex type) is `AddressType`. Two extra elements were added to create the derived type `US-Address`: `Zip` and `State`.



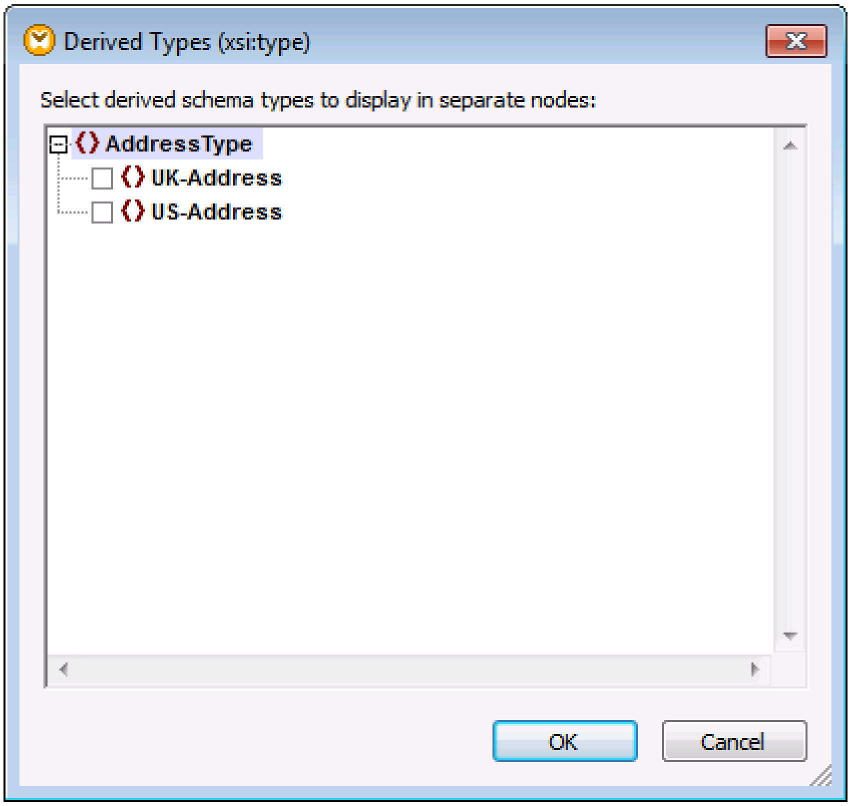
Sample derived type (XMLSpy schema view)

The following example shows you how to map data to or from derived XML schema types.

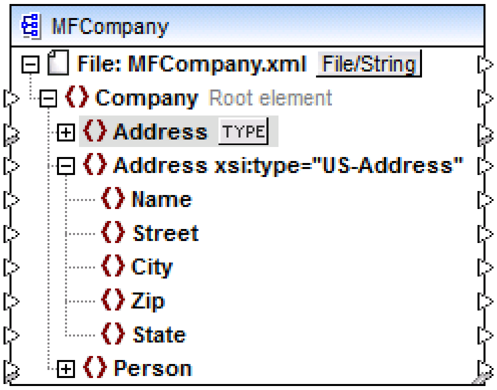
1. On the **Insert** menu, click **XML Schema/File**, and open the following XML Schema: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial \MFCompany.xsd**.
2. When prompted to supply an instance file, click **Skip**, and then select `Company` as the root element.



3. Click the TYPE button next to the `Address` element. This button indicates that derived types exist for this element in the schema.



4. Select the check box next to the derived type you want to use (`US-Address`, in this case), and confirm with OK. A new element `Address xsi:type="US-Address"` has been added to the component.



You can now map data to or from the `US-Address` derived type.

Note that you can also include multiple derived types by selecting them in the Derived Types dialog box. In this case, each would have its own `xsi:type` element in the component.

7.1.5 QNames

MapForce resolves QName (qualified name) prefixes (https://www.w3.org/TR/xml-names/#ns-qualnames) when reading data from XML files at mapping execution run-time.

QNames are used to reference and abbreviate namespace URIs in XML instance documents. There are two types of QNames: Prefixed and Unprefixed QNames.

PrefixedName	Prefix ':' LocalPart	
UnPrefixedName	LocalPart where LocalPart is an Element or Attribute name.	

For example, in the listing below, `<x:p/>` is a QName, where:

- the prefix "x" is an abbreviation of the namespace "**http://myCompany.com**".
- `p` is the local part.

```
<?xml version='1.0'?>
<doc xmlns:x="http://myCompany.com">
   <x:p/>
</doc>
```

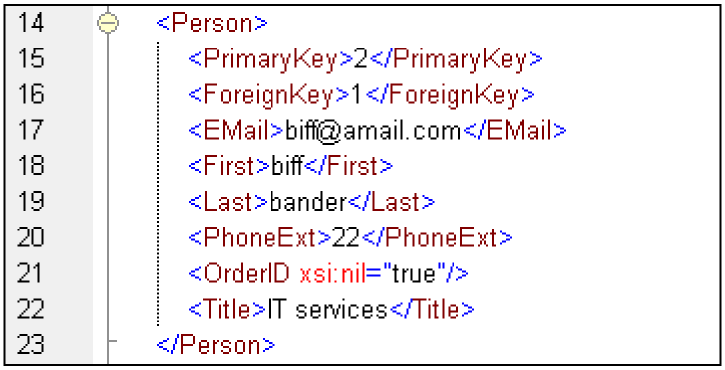
MapForce also includes several QName-related functions in the **`core | QName functions`** library.

7.1.6 Nil Values / Nillable

The XML Schema specification allows for an element to be valid without content if the `nillable="true"` attribute has been defined for that specific element in the schema. In the instance XML document, you can then indicate that the value of an element is nil by adding the `xsi:nil="true"` attribute to it. This section describes how MapForce handles nil elements in source and target components.

'xsi:nil' versus 'nillable'

The `xsi:nil="true"` attribute is defined in the XML **instance** document.

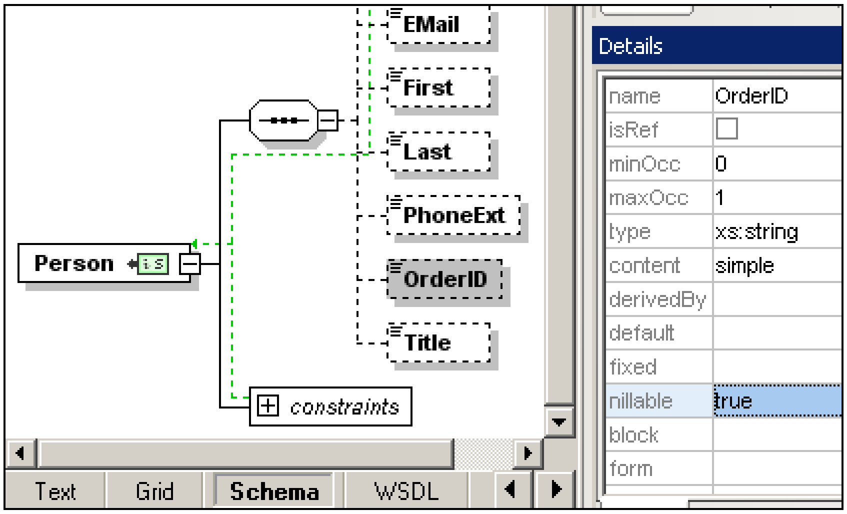


```
<Person>
    <PrimaryKey>2</PrimaryKey>
    <ForeignKey>1</ForeignKey>
    <EMail>biff@amail.com</EMail>
    <First>biff</First>
    <Last>bander</Last>
    <PhoneExt>22</PhoneExt>
    <OrderID xsi:nil="true"/>
    <Title>IT services</Title>
</Person>
```

The `xsi:nil="true"` attribute indicates that, although the element exists, it has no content. Note that the `xsi:nil="true"` attribute applies to element values, and not to attribute values. An element with `xsi:nil="true"` may still have other attributes, even if it does not have content.

The `xsi:nil` attribute is not displayed explicitly in the MapForce graphical mapping, because it is handled automatically in most cases. Specifically, a "nilled" node (one that has the `xsi:nil="true"` attribute) exists, but its content does not exist.

The `nillable="true"` attribute is defined in the XML **schema**. In MapForce, it can be present in both the source and target components.



name	OrderID
isRef	□
minOcc	0
maxOcc	1
type	xs:string
content	simple
derivedBy	
default	
fixed	
nillable	true
block	
form	

Nillable elements as mapping source

MapForce checks the `xsi:nil` attribute automatically, whenever a mapping reads data from nilled XML elements. If the value of `xsi:nil` is `true`, the content will be treated as non-existent.

When you create a **Target-driven** mapping from a nillable source element to a nillable target element with **simple content** (a single value with optional attributes, but without child elements), where `xsi:nil` is set on a source element, MapForce adds the `xsi:nil` attribute to the target element (for example, `<OrderID xsi:nil="true"/>`).

When you create a **Copy-All** mapping from a nillable source element to a nillable target element, where `xsi:nil` is set on a source element, MapForce adds the `xsi:nil` attribute to the target element (for example, `<OrderID xsi:nil="true"/>`).

To check explicitly whether a source element has the `xsi:nil` attribute set to `true`, use the **`is-xsi-nil`** function. It returns TRUE for nilled elements and FALSE for other nodes.

To substitute a nilled (non-existing) source element value with something specific, use the **`substitute-missing`** function.

Notes:

- Connecting the **`exists`** function to a nilled source element returns TRUE, since the element node actually exists, even if it has no content.
- Using functions that expect simple values (such as **`multiply`** and **`concat`**) on elements where `xsi:nil` has been set does not yield a result, as no element content is present and no value can be extracted. These functions behave as if the source node did not exist.

Nillable elements as mapping target

When you create a **Target-driven** mapping from a nillable source element to a nillable target element with **simple content** (a single value with optional additional attributes, but without child elements), where `xsi:nil` is set on a source element, MapForce inserts the `xsi:nil` attribute into the target element (for example, `<OrderID xsi:nil="true"/>`). If the `xsi:nil="true"` attribute has not been set in the XML source element, then the element content is mapped to the target element in the usual fashion.

When mapping to a nillable target element with **complex type** (with child elements), the `xsi:nil` attribute will **not** be written automatically, because MapForce cannot know at the time of writing the element's attributes if any child elements will follow. For such cases, define a **Copy-All** connection to copy the `xsi:nil` attribute from the source element.

When mapping an **empty sequence** to a target element, the element will not be created at all, independent of its nillable designation.

To force the creation of an empty target element with `xsi:nil` set to `true`, connect the **`set-xsi-nil`** function directly to the target element. This works for target elements with simple and complex types.

If the node has simple type, use the **`substitute-missing-with-xsi-nil`** function to insert `xsi:nil` in the target if no value from your mapping source is available. This can happen if the source node does not exist at all, or if a calculation (for example, multiply) involved a nilled source node and therefore yielded no result.

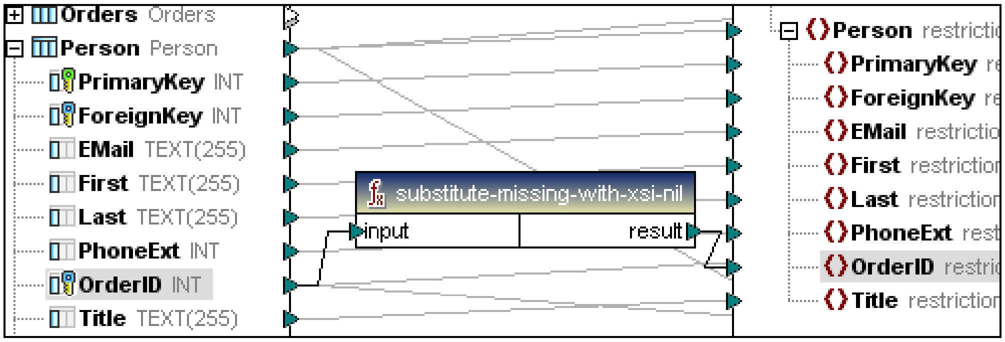
Note:

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

Mapping NULL database fields to xsi:nil

If you map a NULL database field to an nillable element of an XML schema, MapForce generates only those target elements which actually contain database data. Elements of NULL database fields are not created in the target component. Connecting the **`exists`** node function to such a source element results in `false` for the NULL fields.

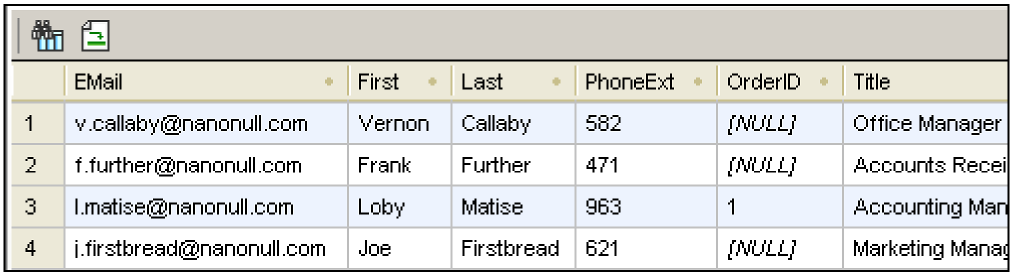
To force the creation of all elements in the target component, use the **`substitute-missing-with-xsi-nil`** function from the node functions of the core library.



The screenshot above illustrates how the **`substitute-missing-with-xsi-nil`** function is used to create target elements for all database fields:

- All missing/NULL database fields contain <OrderID xsi:nil="true"/> in the target element.
- Existing data from database fields is mapped directly to the target element e.g. <OrderID>1</OrderID>.

To see the NULL fields of a database component, click the **Database Query** button and run a query on the database table(s). Null fields are shown as *[NULL]* in the Results window.



	EMail	First	Last	PhoneExt	OrderID	Title
1	v.callaby@nanonull.com	Vernon	Callaby	582	*[NULL]*	Office Manager
2	f.further@nanonull.com	Frank	Further	471	*[NULL]*	Accounts Recei
3	l.matise@nanonull.com	Loby	Matise	963	1	Accounting Man
4	j.firstbread@nanonull.com	Joe	Firstbread	621	*[NULL]*	Marketing Manag

Mapping xsi:nil to NULL database fields

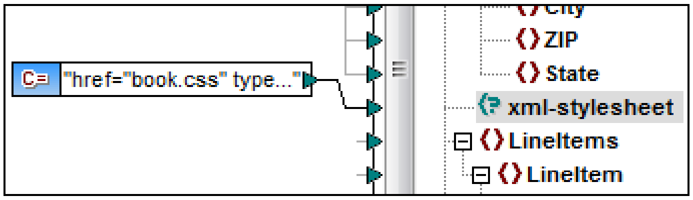
If you map a nilled XML element to a database column, MapForce writes a NULL value to the database. You can also use the **`set-null`** function if you want to set a database field to NULL unconditionally.

7.1.7 Comments and Processing Instructions

Comments and Processing Instructions can be inserted into target XML components. Processing instructions are used to pass information to applications that further process XML documents. Note that Comments and Processing instructions cannot be defined for nodes that are part of a copy-all mapped group.

To insert a Processing Instruction:

1. Right-click an element in the target component and select Comment/Processing Instruction, then one of the Processing Instruction options from the menu (Before, After)
2. Enter the Processing Instruction (target) name in the dialog and press OK to confirm, e.g. xml-stylesheet.
   This adds a node of this name to the component tree.

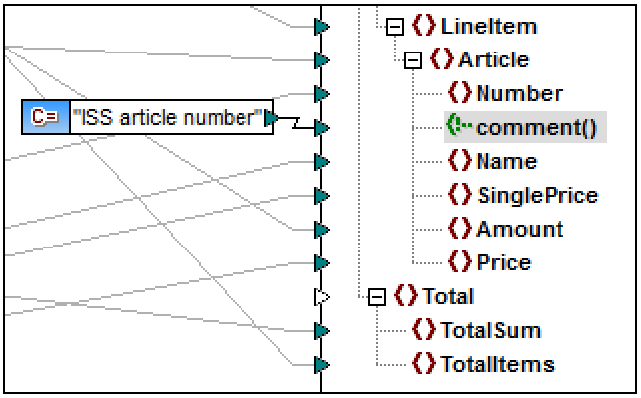


3. You can use, for example, a constant component to supply the value of the Processing Instruction attribute, e.g. `href="book.css" type="text/css"`.

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

To insert a comment:

1. Right-click an element in the target component and select Comment/Processing Instruction, then one of the Comment options from the menu (Before, After).



   This adds the comment node (`<!--comment()`) to the component tree.
2. Use a constant component to supply the comment text, or connect a source node to the comment node.

Note:

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

To delete a Comment/Processing Instruction:

- Right-click the respective node, select Comment/Processing Instruction, then select Delete Comment/Processing Instruction from the flyout menu.

7.1.8 CDATA Sections

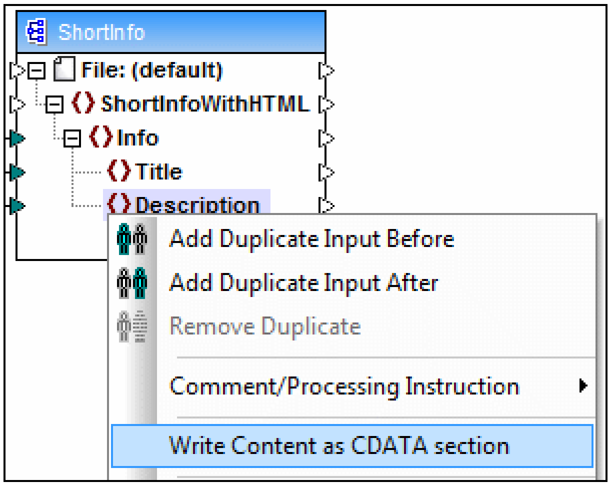
CDATA sections are used to escape blocks of text containing characters which would normally be interpreted as markup. CDATA sections start with "<![CDATA[" and end with the "]]>".

Target nodes can now write the input data that they receive as CDATA sections. The target node components can be:

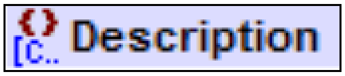
- XML data
- XML data embedded in database fields
- XML child elements of typed dimensions in an XBRL target

To create a CDATA section:

1. Right-click the target node that you want to define as the CDATA section and select "Write Content as CDATA section".


ShortInfo
File: (default)
ShortInfoWithHTML
Info
Title
Description
Add Duplicate Input Before
Add Duplicate Input After
Remove Duplicate
Comment/Processing Instruction
Write Content as CDATA section

A prompt appears warning you that the input data should not contain the CDATA section close delimiter ']]>', click OK to close the prompt.
The [C.. icon shown below the element tag shows that this node is now defined as a CDATA section.



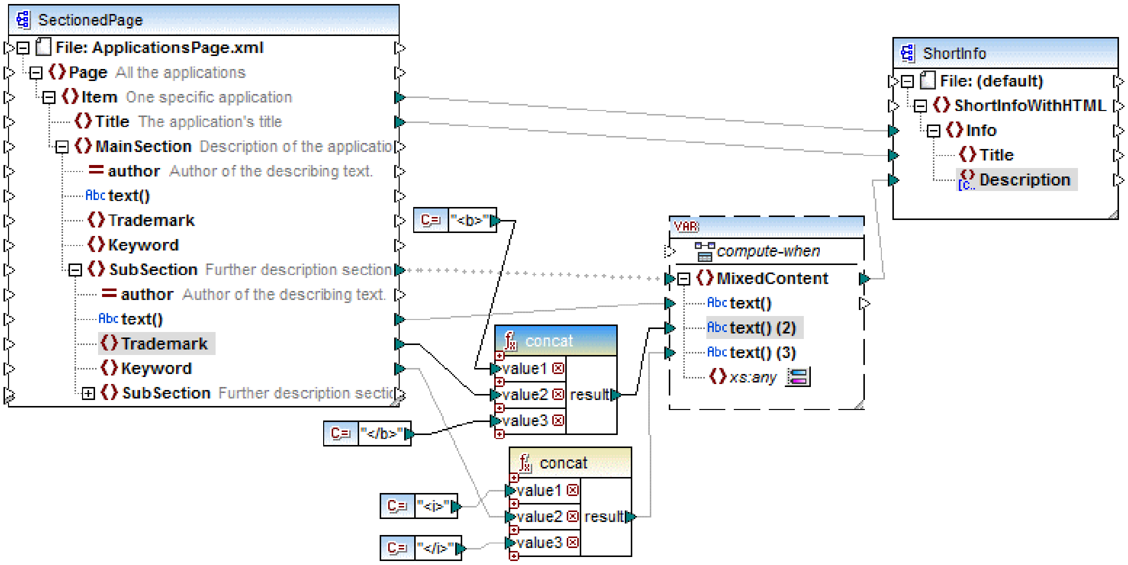
Note:

CDATA sections can also be defined on duplicate nodes, and xsi:type nodes.

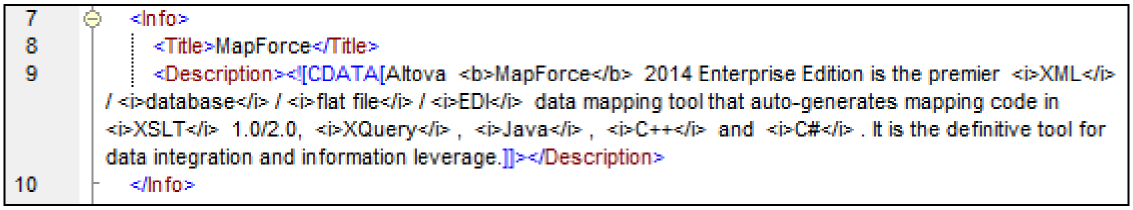
Example:
The **HTMLinCDATA.mfd** mapping file available in the ...\MapForceExamples folder shows an example of where CDATA sections can be very useful.

In this example:

- Bold start (<b>) and end (</b>) tags are added to the content of the **Trademark** source element.
- Italic start (<i>) and end (</i>) tags are added to the content of the **Keyword** source element.
- The resulting data is passed on to duplicate **text()** nodes in the order that they appear in the source document, due to the fact the Subsection element connector, has been defined as a Source Driven (Mixed content) node.
- The output of the MixedContent node is then passed on to the **Description** node in the ShortInfo target component, which has been defined as a CDATA section.


SectionedPage
File: ApplicationsPage.xml
Page All the applications
Item One specific application
Title The application's title
MainSection Description of the applicatio
author Author of the describing text.
text()
Trademark
Keyword
SubSection Further description section
author Author of the describing text.
text()
Trademark
Keyword
SubSection Further description sectio
"<b>"
"</b>"
"<i>"
"</i>"
concat
value1
value2
value3
result
concat
value1
value2
value3
result
VAR
compute-when
MixedContent
text()
text() (2)
text() (3)
xs:any
ShortInfo
File: (default)
ShortInfoWithHTML
Info
Title
Description

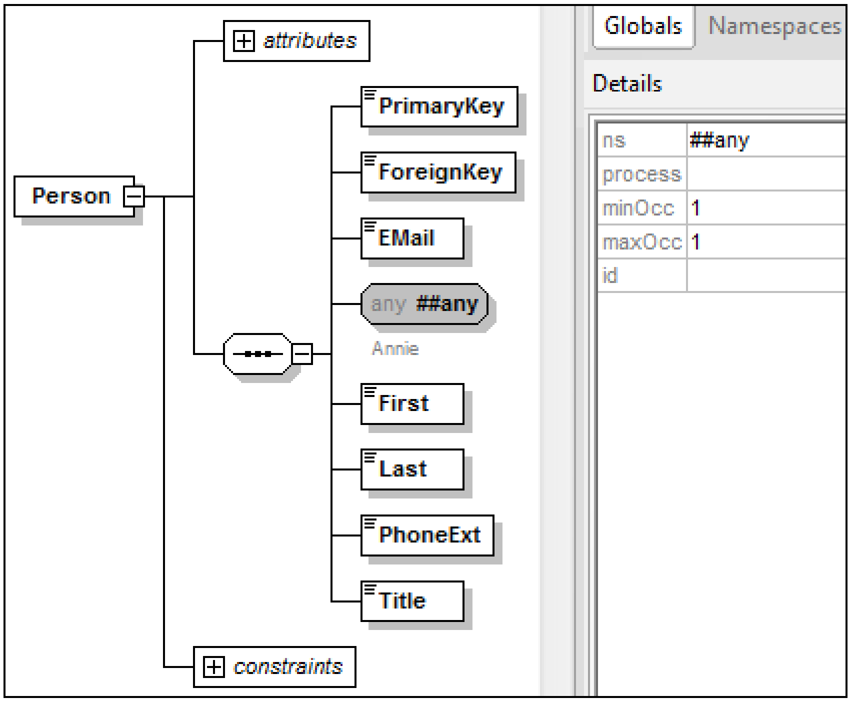
Clicking the Output button shows the CDATA section containing the marked-up text.



```
<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>
```

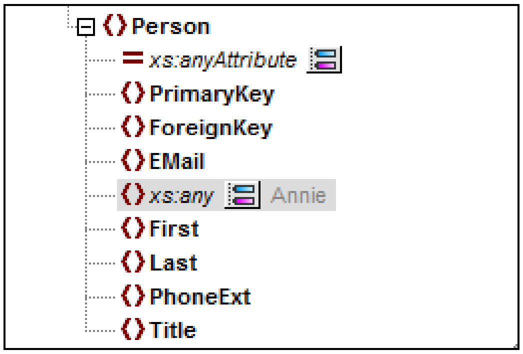
7.1.9 Wildcards - xs:any / xs:anyAttribute

The wildcards `xs:any` (and `xs:anyAttribute`) allow you to use any elements/attributes from schemas. The screenshot shows the "any" element in the Schema view of XMLSpy.

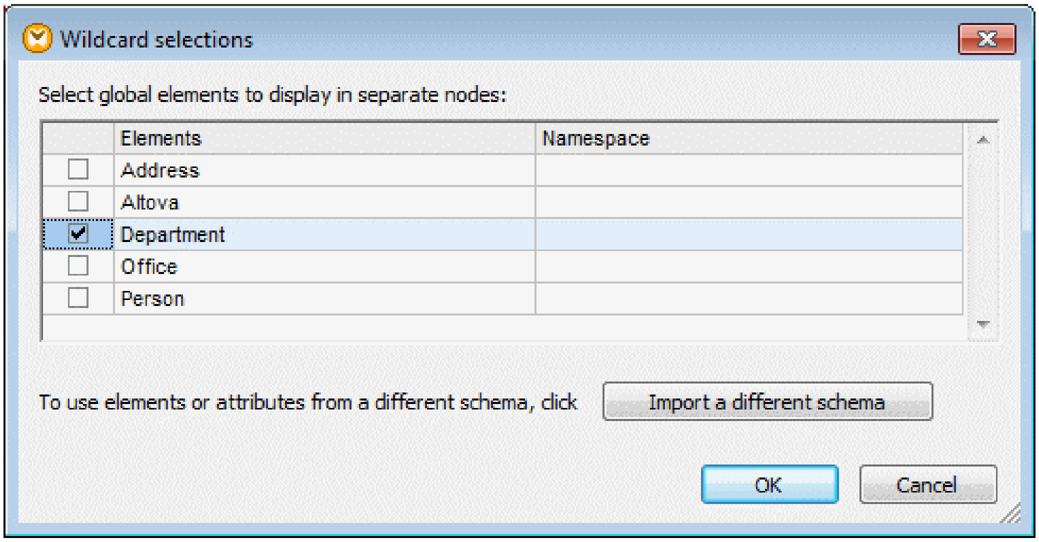


ns	##any
process	
minOcc	1
maxOcc	1
id	

In MapForce, a **Change Selection** (☰) button appears to the right of the `xs:any` element (or `xs:anyAttribute`).



When clicked, the **Change Selection** button ☰ opens the "Wildcard selections" dialog box. The entries in this list show the global elements and attributes declared in the current schema.



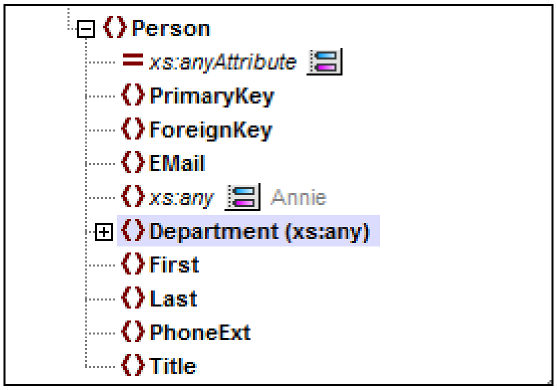
Wildcard selections

Select global elements to display in separate nodes:

	Elements	Namespace
☐	Address	
☐	Altova	
☑	Department	
☐	Office	
☐	Person	

To use elements or attributes from a different schema, click

Clicking one or more of the check boxes and confirming with OK, inserts that element/attribute (and any other child nodes) into the component. The wildcard elements or attributes are inserted immediately after the node whose **Change Selection** (☰) button was clicked.



```
⊟ {} Person
     = xs:anyAttribute
     {} PrimaryKey
     {} ForeignKey
     {} EMail
     {} xs:any  Annie
   ⊞ {} Department (xs:any)
     {} First
     {} Last
     {} PhoneExt
     {} Title
```

You can now map to/from these nodes as with any other element.

> On a component, the wildcard elements or attributes can be recognized by the **(xs:any)** text appended to their name.

To remove a wildcard element, click the Change Selection (☰) button, and then deselect it from the "Wildcard selections" dialog box.

Wildcards and dynamic node names

Mapping data to or from wildcards is generally suitable where all possible elements or attributes that appear in the XML instance are declared by the component's XML schema (or can be imported from external schemas). However, there may be situations where elements or attributes appearing in an instance are too many to be declared in the schema. Consider the following instance where the number of child elements of `<message>` is arbitrary:

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

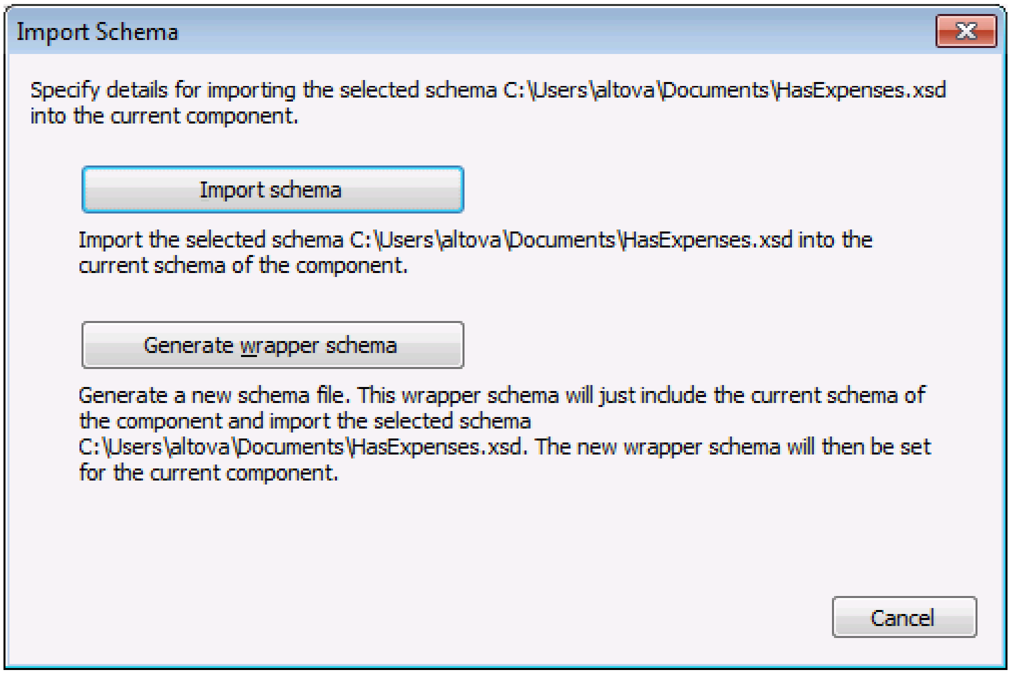
For such situations, use dynamic access to node names (see Mapping Node Names) instead of wildcards.

Adding elements from a different schema as wildcards

Elements from a schema other than the one assigned to the component can also be used as wildcards. To make such elements visible on the component, click the **Import a different schema** button on the "Wildcard selections" dialog box. This opens a new dialog box where you have two options:

1. Import schema
2. Generate wrapper schema

For example, the image below illustrates what happens if you attempt to import an external schema called **HasExpenses.xsd** into a current schema assigned to a component.



Import Schema

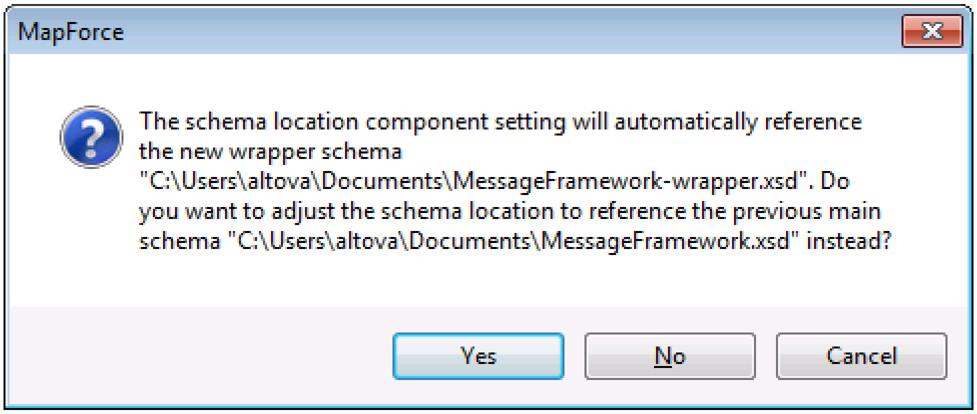
Specify details for importing the selected schema C:\Users\altova\Documents\HasExpenses.xsd into the current component.

Import the selected schema C:\Users\altova\Documents\HasExpenses.xsd into the current schema of the component.

Generate a new schema file. This wrapper schema will just include the current schema of the component and import the selected schema C:\Users\altova\Documents\HasExpenses.xsd. The new wrapper schema will then be set for the current component.

The **Import schema** option imports the external schema into the current schema assigned to the component. Be aware that this option overrides the existing schema of the component on the disk. If the current schema is a remote schema that was opened from a URL (see Adding Components from a URL) and not from the disk, it cannot be modified. In this case, use the **Generate wrapper schema** option.

The **Generate wrapper schema** option creates a new schema file called a "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 (that is, the wrapper schema) which will include both the existing schema and the schema to be imported. When you select this option, you are prompted to choose where the wrapper schema should be saved. By default, the wrapper schema has a name in the form **somefile-wrapper.xsd**. After you save the wrapper schema, it is by default automatically assigned to the component, and a dialog box prompts you:



The schema location component setting will automatically reference the new wrapper schema "C:\Users\altova\Documents\MessageFramework-wrapper.xsd". Do you want to adjust the schema location to reference the previous main schema "C:\Users\altova\Documents\MessageFramework.xsd" instead?

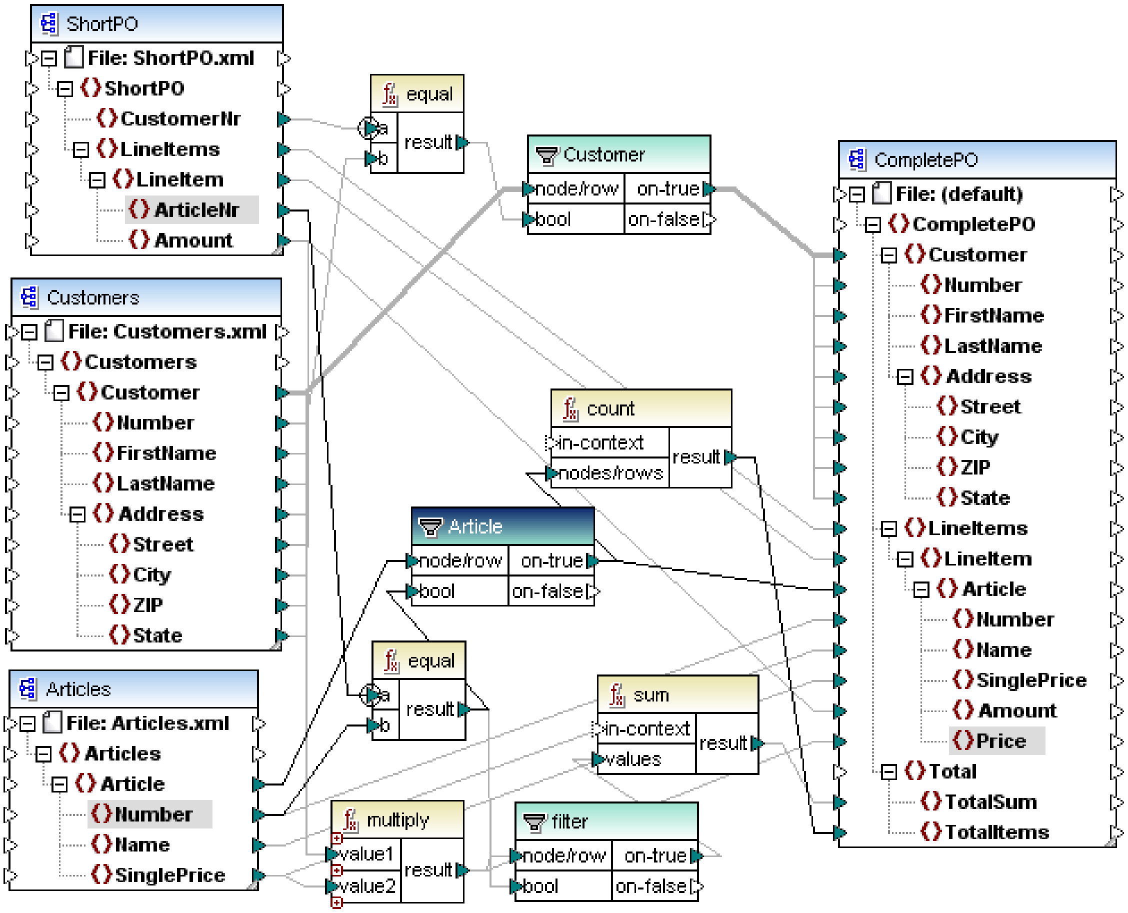
Click **Yes** to revert to the previous schema; otherwise click **No** to keep the newly created wrapper schema assigned to the component.

7.1.10 Merging Data from Multiple Schemas

MapForce allows you to merge multiple files into a single target file.

This example merges multiple source components with different schemas to a target schema. To merge an arbitrary number of files using the same schema, see Processing Multiple Input or Output Files Dynamically.

The **CompletePO.mfd** file available in the **...\MapForceExamples** folder shows how three XML files are merged into one purchasing order XML file.



Note that multiple source component data are combined into one target XML file - CompletePO

- **ShortPO** is a schema with an associated XML instance file and contains only customer number and article data, i.e. Line item, number and amount. (There is only one customer in this file with the Customer number of 3)
- **Customers** is a schema with an associated XML instance file and contains customer number and customer information details, i.e. Name and Address info.
- **Articles** is a schema with an associated XML instance and contains article data, i.e. article name number and price.
- **CompletePO** is a schema file without an instance file as all the data is supplied by the three XML instance files. The hierarchical structure of this file makes it possible to merge and output all XML data.

This schema file has to be created in an XML editor such as XMLSpy, it is not generated by MapForce (although it would be possible to create if you had a **CompletePO.xml** instance file).

The structure of CompletePO is a combination of the source XML file structures.

The **filter** component (Customer) is used to find/filter the data where the customer numbers are identical in both the ShortPO and Customers XML files, and pass on the associated data to the target CompletePO component.

- The **CustomerNr** in ShortPO is compared with the **Number** in Customers using the "equal" function.
- As ShortPO only contains one customer (number 3), only customer and article data for

customer number 3, can be passed on to the filter component.

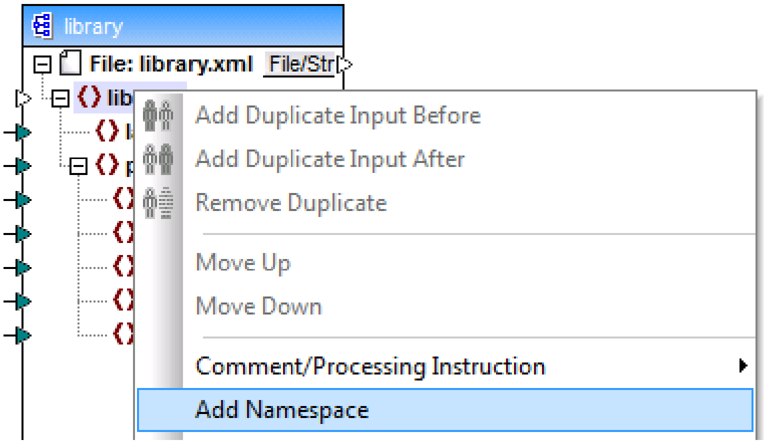
- The **node/row** parameter, of the filter component, passes on the **Customer** data to "on-true" when the bool parameter is true, i.e. when the same number has been found, in this case customer number 3.
- The rest of the customer and article data are passed on to the target schema through the two other filter components.

7.1.11 Declaring Custom Namespaces

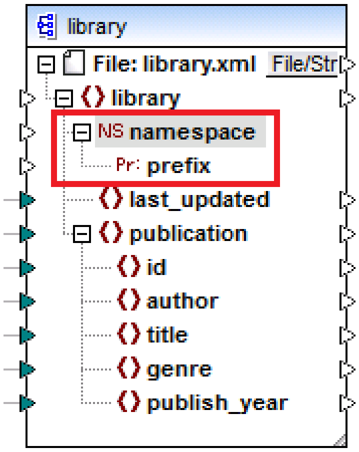
By default, when a mapping produces XML output, the namespace (or set of namespaces) of each element and attribute is automatically derived by MapForce from the schema associated with the target component. This is the default behavior in MapForce and is suitable for most mapping scenarios that involve generation of XML output.

However, there might be cases when you want to have more control over namespaces of elements in the resulting XML output. For example, you may want to manually declare the namespace of an element directly from the mapping.

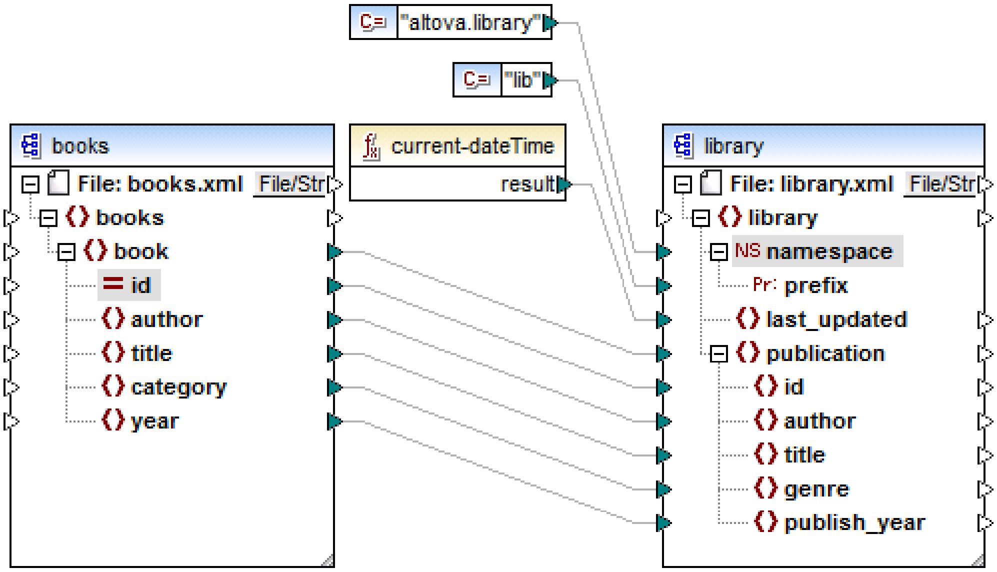
To understand how this works, open the **BooksToLibrary.mfd** mapping available in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\**. Right-click the `library` node, and select **Add Namespace** from the context menu.


library
File: library.xml
File/Str
lib
Add Duplicate Input Before
Add Duplicate Input After
Remove Duplicate
Move Up
Move Down
Comment/Processing Instruction
Add Namespace

Notice that two new nodes are now available under the `library` node: a `namespace` and a `prefix`.



You can now map to them string values from the mapping. In the image below, two constants were defined (from **Insert | Constant** menu command) that provide the namespace "altova.library" and the prefix "lib":



The result is that, in the generated output, an `xmlns:<prefix>="<namespace>"` attribute is added to the element, where `<prefix>` and `<namespace>` are values that come from the mapping (in this case, from constants). The generated output will now look as follows (notice the highlighted part):

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

Note: Declaring custom namespaces (and the **Add Namespace** command) is meaningful only for target XML components, and applies to elements only. The **Add Namespace**

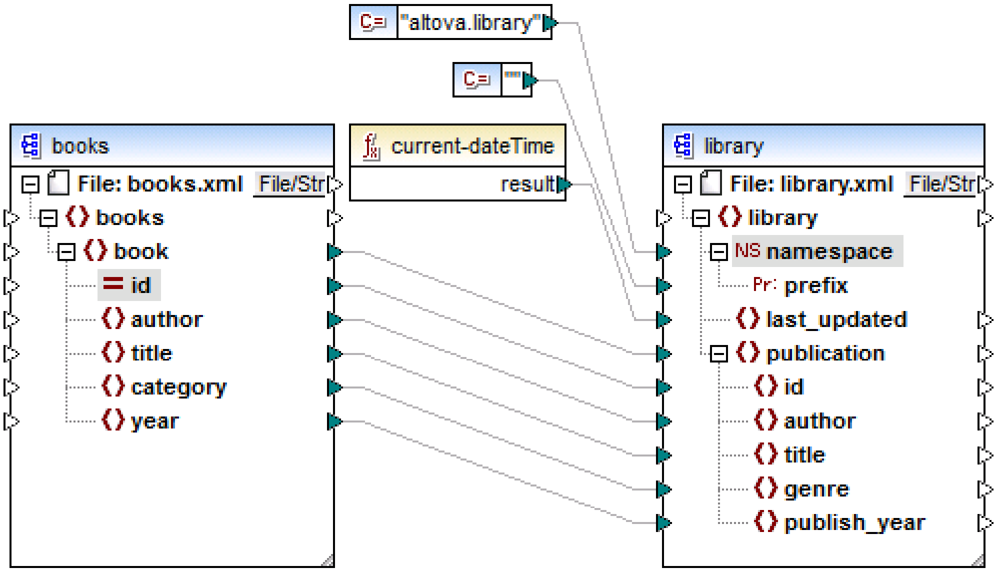
command is not available for attributes and wildcard nodes. It is also not available for nodes which receive data by means of a **Copy-All** connection.

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 the new prefix and namespace values.

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

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

If you want to declare a default namespace (that is, one in the format `xmlns="mydefaultnamespace"`), map an empty string value to `prefix`. To see this case in action, edit the example mapping above so as to make the second constant an empty string.



The resulting output would then looks as follows:

```
<?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 either enabling dynamic access to node's attributes (see Mapping Node Names), or by editing the schema so that it has a `prod:id` attribute for `<number>`.

7.2 Databases and MapForce

Altova website: 🔗 Database mapping

MapForce 2018 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 following databases are supported. The available root object for each database is also listed. 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	Root Object	Notes
Firebird 2.5.4	database	
IBM DB2 8.x, 9.1, 9.5, 9.7, 10.1, 10.5	schema	
IBM DB2 for i 6.1, 7.1	schema	Logical files are supported and shown as views.
IBM Informix 11.70	database	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	database	
Microsoft Access 2003, 2007, 2010, 2013	database	
Microsoft Azure SQL Database	database	SQL Server 2016 codebase
Microsoft SQL Server 2005, 2008, 2012, 2014, 2016	database	
MySQL 5.0, 5.1, 5.5, 5.6, 5.7	database	
Oracle 9i, 10g, 11g, 12c	schema	
PostgreSQL 8.0, 8.1, 8.2, 8.3, 9.0.10, 9.1.6, 9.2.1, 9.4, 9.5, 9.6	database	PostgreSQL connections are supported both as native connections and driver-based connections through interfaces (drivers) such as ODBC or JDBC. Native connections do not require any drivers.

Database	Root Object	Notes
Progress OpenEdge 11.6	database	
SQLite 3.x	database	SQLite connections are supported as native, direct connections to the SQLite database file. No separate drivers are required.
Sybase ASE 15, 16	database	
Teradata 16	database	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

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 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:

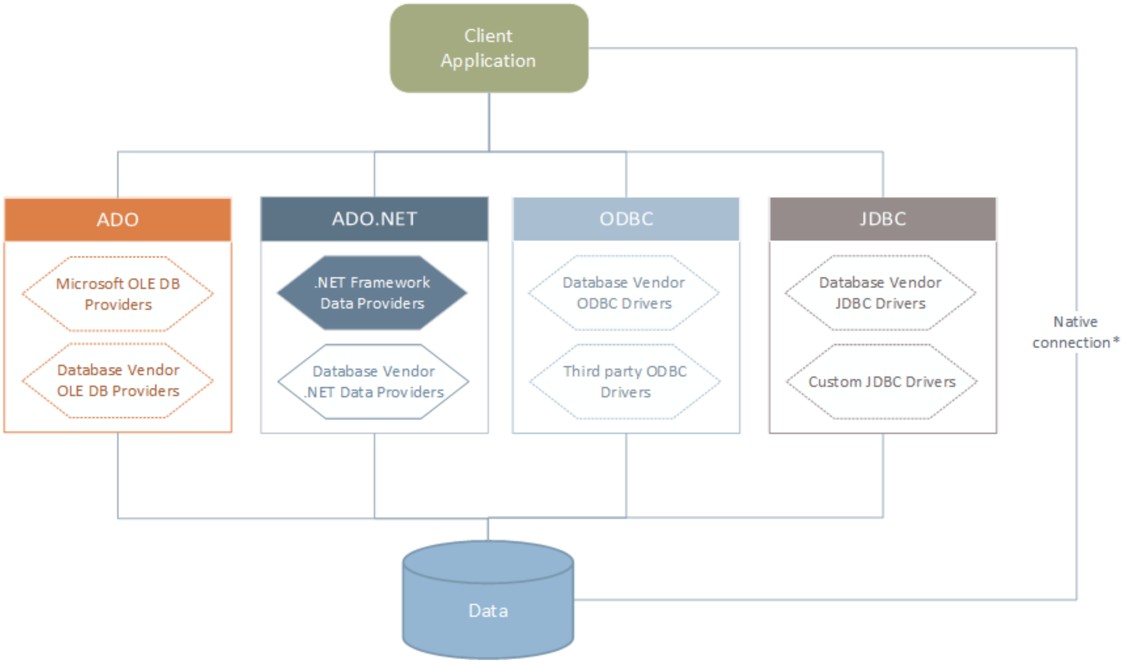
- Database Connections on Linux and Mac
- Compiling Mappings to MapForce Server Execution Files
- Deploying Mappings to FlowForce Server
- Code Generator

7.2.1 Connecting to a Database

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).


Client Application
ADO
Microsoft OLE DB Providers
Database Vendor OLE DB Providers
ADO.NET
.NET Framework Data Providers
Database Vendor .NET Data Providers
ODBC
Database Vendor ODBC Drivers
Third party ODBC Drivers
JDBC
Database Vendor JDBC Drivers
Custom JDBC Drivers
Native connection*
Data

** Direct native connections are supported for SQLite and PostgreSQL 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)

> Some ADO.NET providers are not supported or have limited support. See ADO.NET Support Notes.

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.

7.2.1.1 Starting the Database Connection Wizard

Whenever you take an action that requires a database connection, a wizard appears that guides you through the steps required to set up the connection.

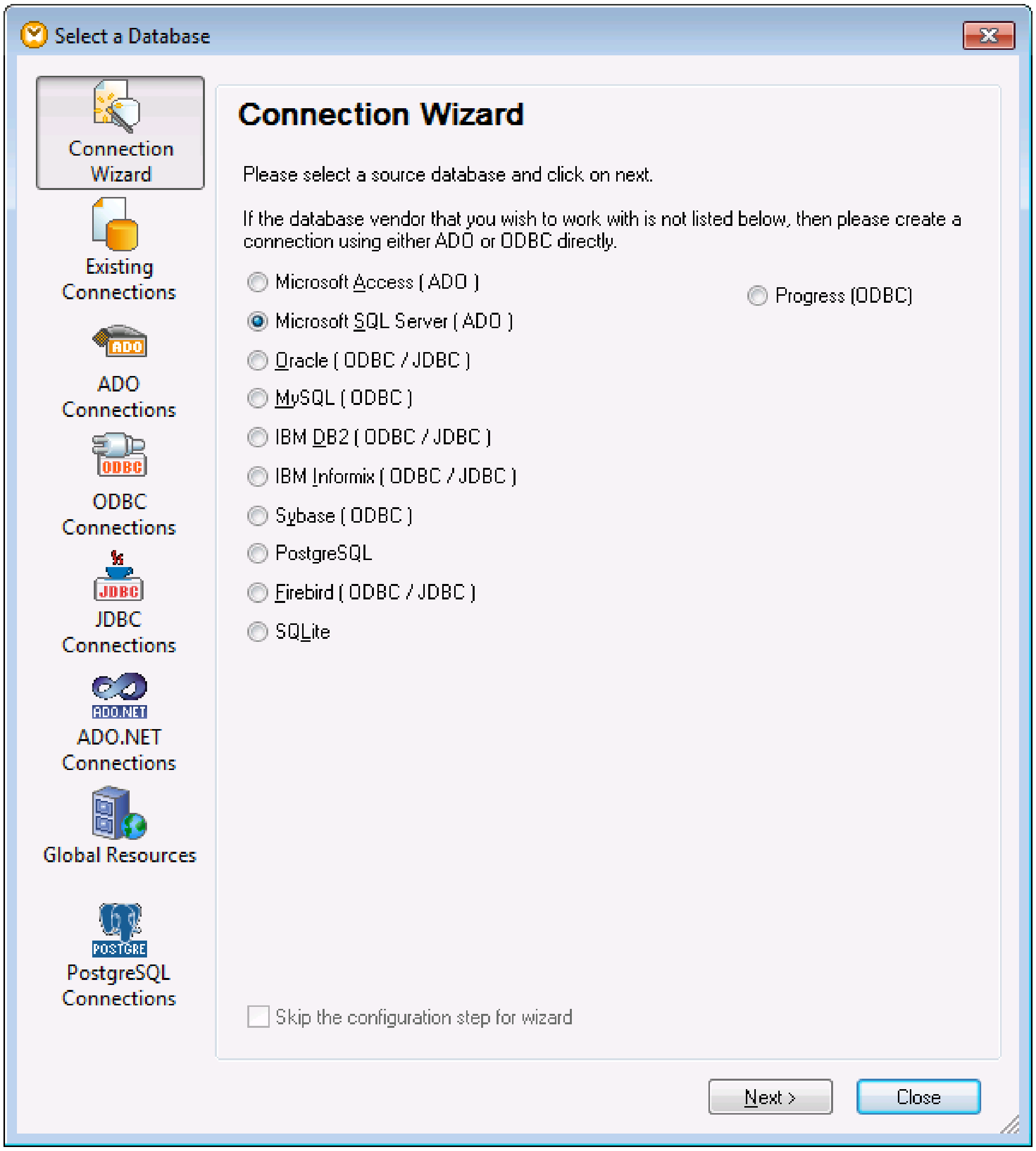
Before you go through the wizard steps, be aware that for some database types it is necessary to install and configure separately several database prerequisites, such as a database driver or database client software. These are normally provided by the respective database vendors, and include documentation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

To add the database as a source or target component on a mapping:

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

To add the database as a reusable global resource:

1. On the **Tools** menu, click **Global Resources**.
2. Click **Add**, and then click Database.
3. Click **Choose Database**.



Connection Wizard

Please select a source database and click on next.

If the database vendor that you wish to work with is not listed below, then please create a connection using either ADO or ODBC directly.

○ Microsoft Access (ADO)
● Microsoft SQL Server (ADO)
○ Oracle (ODBC / JDBC)
○ MySQL (ODBC)
○ IBM DB2 (ODBC / JDBC)
○ IBM Informix (ODBC / JDBC)
○ Sybase (ODBC)
○ PostgreSQL
○ Firebird (ODBC / JDBC)
○ SQLite
○ Progress (ODBC)

□ Skip the configuration step for wizard

After you select a database type and click **Next**, the on-screen instructions will depend on the database kind, technology (ADO, ADO.NET, ODBC, JDBC) and driver used.

For examples applicable to each database type, see Database Connection Examples. For instructions applicable to each database access technology, refer to the following topics:

- Setting up an ADO Connection
- Setting up an ADO.NET Connection
- Setting up an ODBC Connection
- Setting up a JDBC Connection

7.2.1.2 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.
- 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.
- 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.
- 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.
- 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/)

Database	Interface	Drivers
	JDBC	Firebird JDBC driver (https://www.firebirdsql.org/en/jdbc-driver/)
	ODBC	Firebird ODBC driver (https://www.firebirdsql.org/en/odbc-driver/)
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 Provider for SQL Server • SQL Server Native Client
	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	• SQL Server Native Client
MySQL	ADO.NET	Connector/NET (https://dev.mysql.com/downloads/connector/net/)

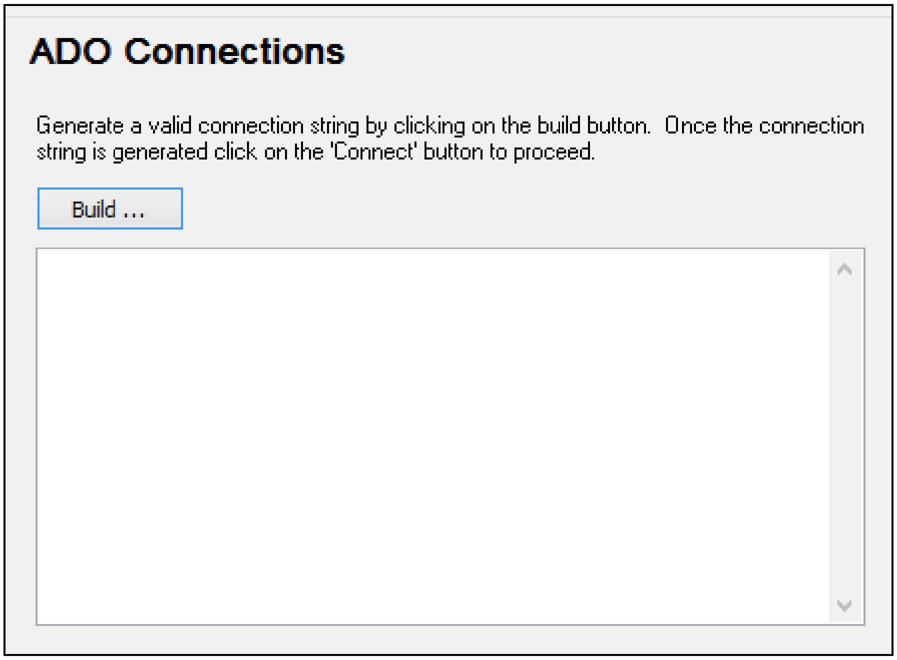
Database	Interface	Drivers
	JDBC	Connector/J (https://dev.mysql.com/downloads/connector/j/)
	ODBC	Connector/ODBC (https://dev.mysql.com/downloads/connector/odbc/)
Oracle	ADO	• Oracle Provider for OLE DB • Microsoft OLE DB Provider for Oracle
	ADO.NET	Oracle Data Provider for .NET (http://www.oracle.com/technetwork/topics/dotnet/index-085163.html)
	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)

7.2.1.3 Setting up an 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 the typical choice for connecting to Microsoft native databases such as Microsoft Access or SQL Server, although you can also use it for other data sources.

To set up an ADO connection:

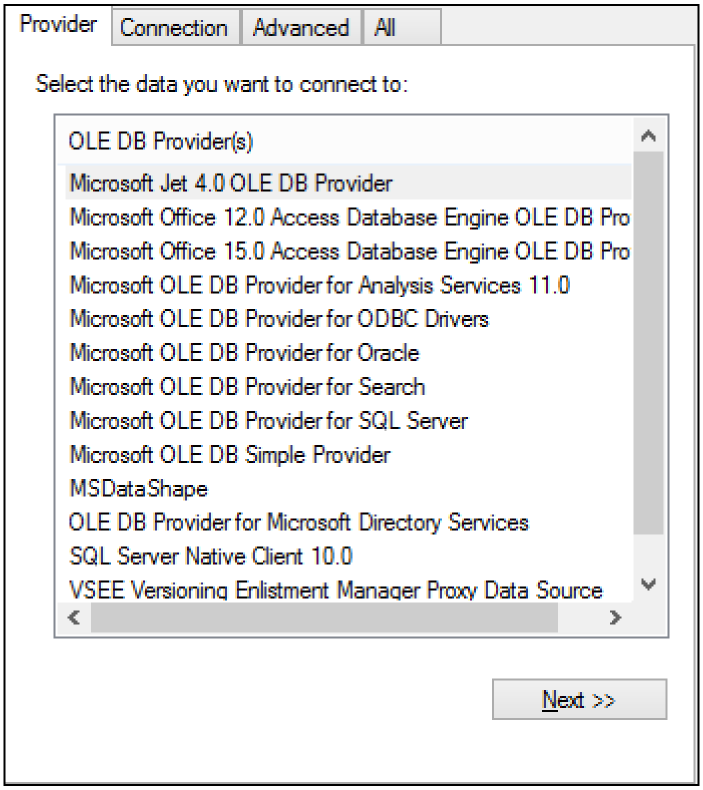
1. Start the database connection wizard.
2. Click **ADO Connections**.



ADO Connections

Generate a valid connection string by clicking on the build button. Once the connection string is generated click on the 'Connect' button to proceed.

3. Click **Build**.



Select the data you want to connect to:

OLE DB Provider(s)

- Microsoft Jet 4.0 OLE DB Provider
- Microsoft Office 12.0 Access Database Engine OLE DB Pro
- Microsoft Office 15.0 Access Database Engine OLE DB Pro
- Microsoft OLE DB Provider for Analysis Services 11.0
- Microsoft OLE DB Provider for ODBC Drivers
- Microsoft OLE DB Provider for Oracle
- Microsoft OLE DB Provider for Search
- Microsoft OLE DB Provider for SQL Server
- Microsoft OLE DB Simple Provider
- MSDataShape
- OLE DB Provider for Microsoft Directory Services
- SQL Server Native Client 10.0
- VSEE Versioning Enlistment Manager Proxy Data Source

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** When connecting to Microsoft Access 2003, you can also use the **Microsoft Jet OLE DB Provider**.
SQL Server	• **SQL Server Native Client** • **Microsoft OLE DB Provider for SQL Server**
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). Alternatively, set up an ODBC or JDBC connection. If the operating system has an ODBC driver to the required database, you can also use the **Microsoft OLE DB Provider for ODBC Drivers**.

5. 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, as well as the database username and password. For Microsoft Access, you will be asked to browse for or provide the path to the database file.

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. 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
- Setting up the Microsoft Access Data Link Properties

Connecting to an Existing Microsoft Access Database

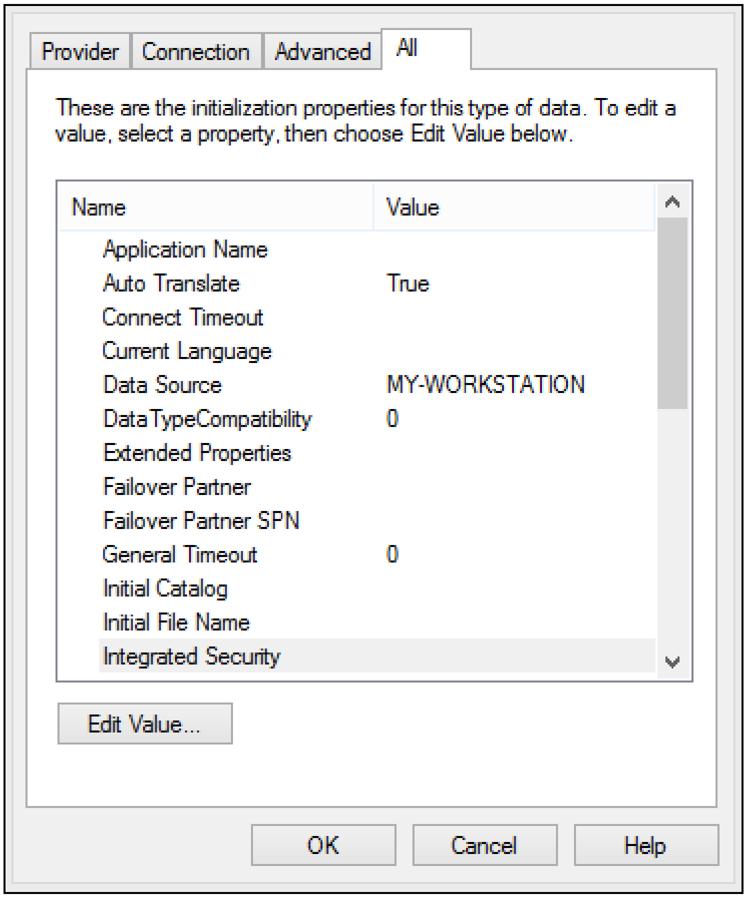
This approach is suitable when you want to connect to a Microsoft Access database which is not password-protected. If the database is password-protected, set up the database password as shown in Connecting to Microsoft Access (ADO).

To connect to an existing Microsoft Access database:

1. Run the database connection wizard (see Starting the Database Connection Wizard).
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**.

Setting up the SQL Server Data Link Properties

When you connect to a Microsoft SQL Server database through ADO (see Setting up an ADO Connection), ensure that the following data link properties are configured correctly in the **All** tab of the Data Link Properties dialog box.



These are the initialization properties for this type of data. To edit a value, select a property, then choose Edit Value below.

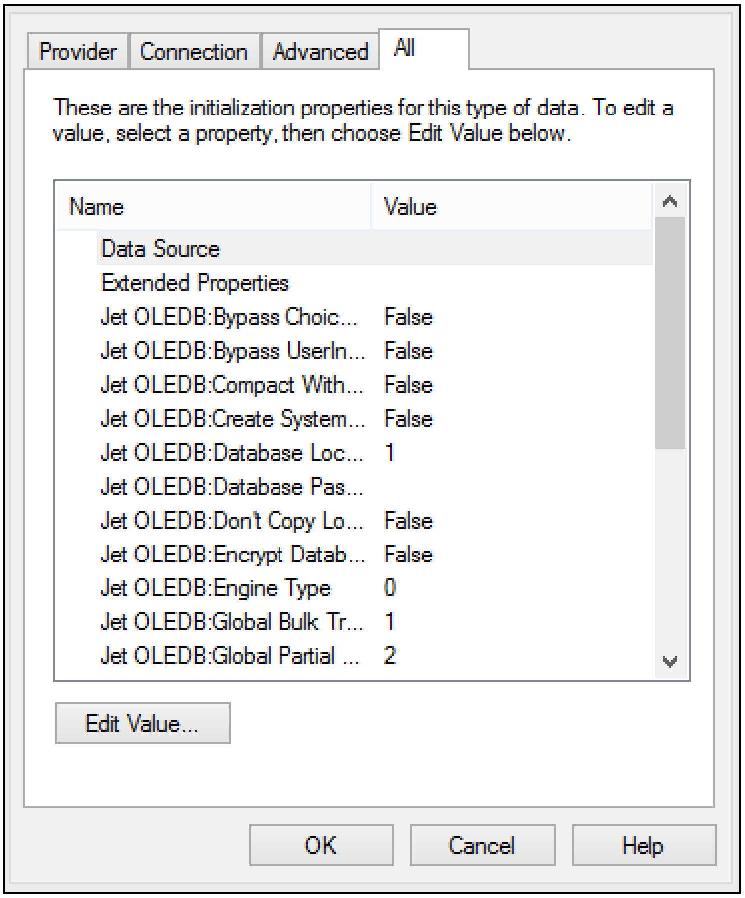
Name	Value
Application Name	
Auto Translate	True
Connect Timeout	
Current Language	
Data Source	MY-WORKSTATION
DataTypeCompatibility	0
Extended Properties	
Failover Partner	
Failover Partner SPN	
General Timeout	0
Initial Catalog	
Initial File Name	
Integrated Security	

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**.

Setting up the Microsoft Access Data Link Properties

When you connect to a Microsoft Access database through ADO (see Setting up an ADO Connection), ensure that the following properties are configured correctly in the **All** tab of the Data Link Properties dialog box.

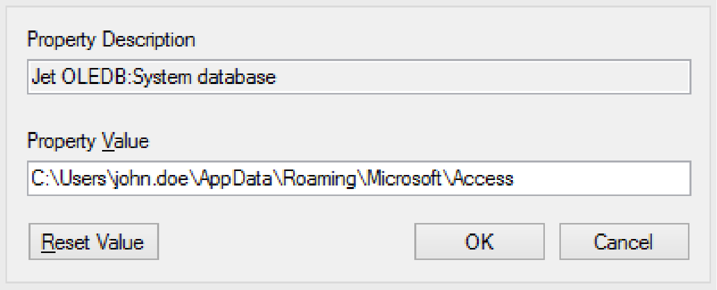
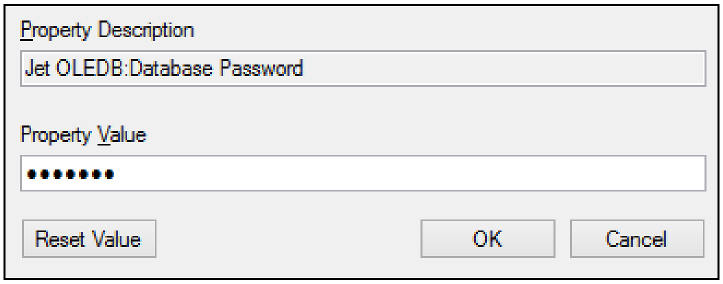


These are the initialization properties for this type of data. To edit a value, select a property, then choose Edit Value below.

Name	Value
Data Source	
Extended Properties	
Jet OLEDB:Bypass Choic...	False
Jet OLEDB:Bypass UserIn...	False
Jet OLEDB:Compact With...	False
Jet OLEDB:Create System...	False
Jet OLEDB:Database Loc...	1
Jet OLEDB:Database Pas...	
Jet OLEDB:Don't Copy Lo...	False
Jet OLEDB:Encrypt Datab...	False
Jet OLEDB:Engine Type	0
Jet OLEDB:Global Bulk Tr...	1
Jet OLEDB:Global Partial ...	2

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`
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.

Property	Notes
	 Property Description Jet OLEDB:System database Property Value C:\Users\john.doe\AppData\Roaming\Microsoft\Access
Jet OLEDB:Database Password	If the database is password-protected, set the value of this property to the database password.  Property Description Jet OLEDB:Database Password Property Value ●●●●●●●

7.2.1.4 Setting up an 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.

To set up an ADO.NET connection:

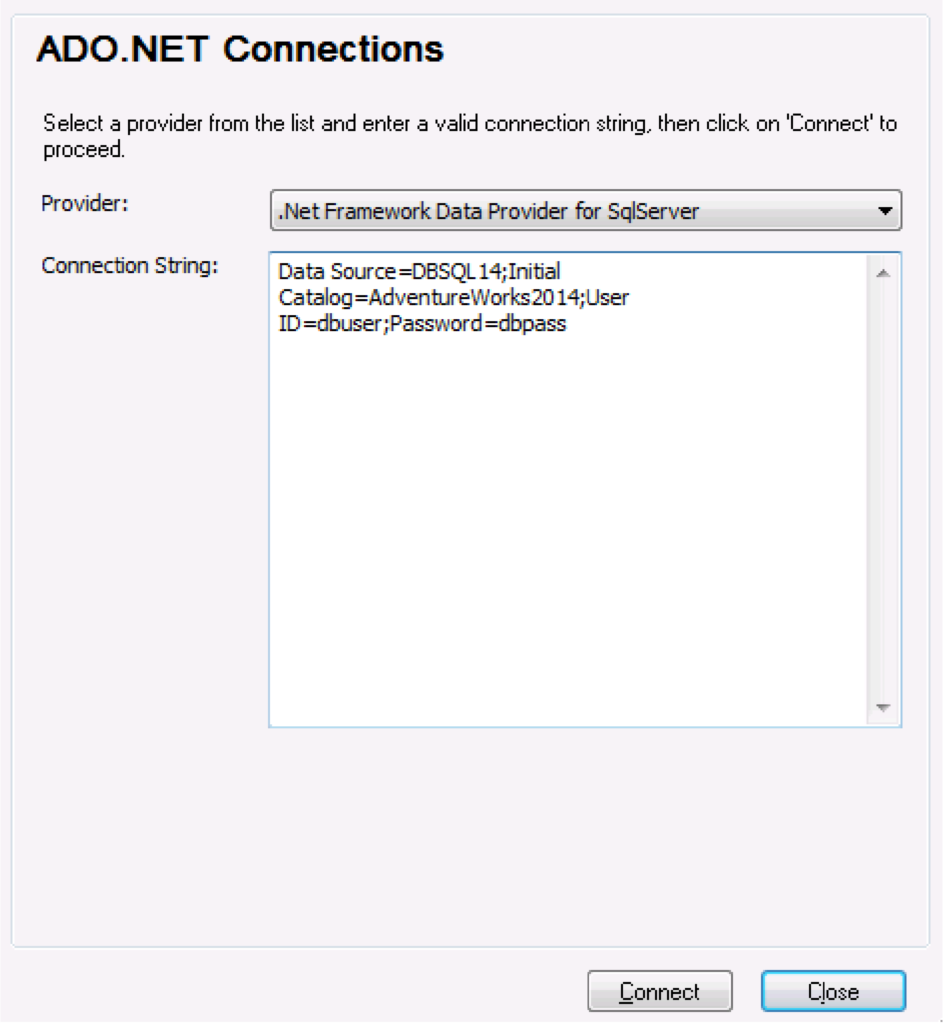
1. Start the database connection wizard.
2. Click **ADO.NET Connections**.

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).

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


ADO.NET Connections
Select a provider from the list and enter a valid connection string, then click on 'Connect' to proceed.
Provider:
Connection String:
Connect
Close

Provider: .Net Framework Data Provider for SqlServer

Connection String:

```
Data Source=DBSQL14;Initial
Catalog=AdventureWorks2014;User
ID=dbuser;Password=dbpass
```

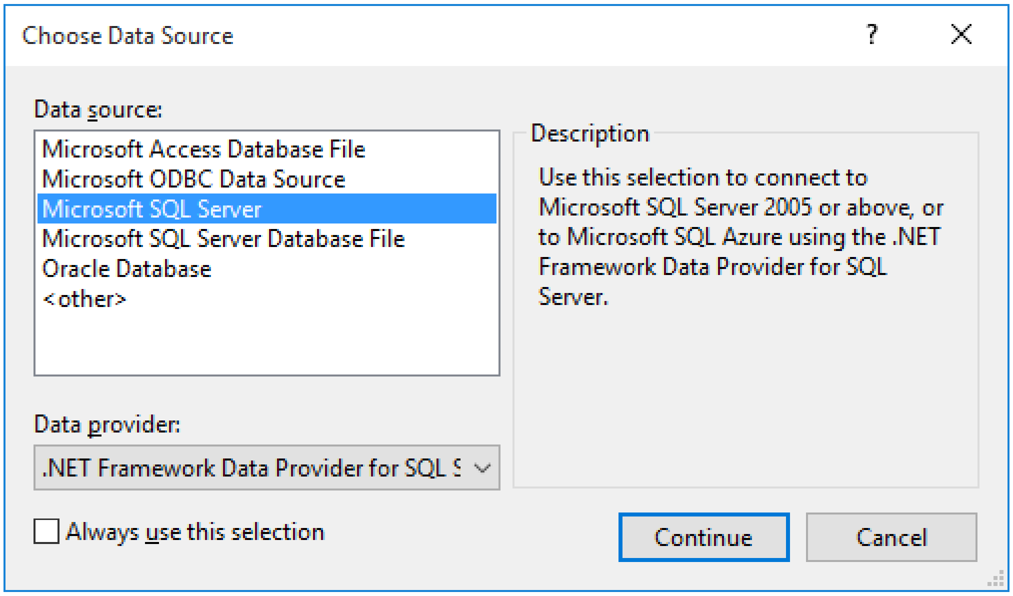
5. Click **Connect**.

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.

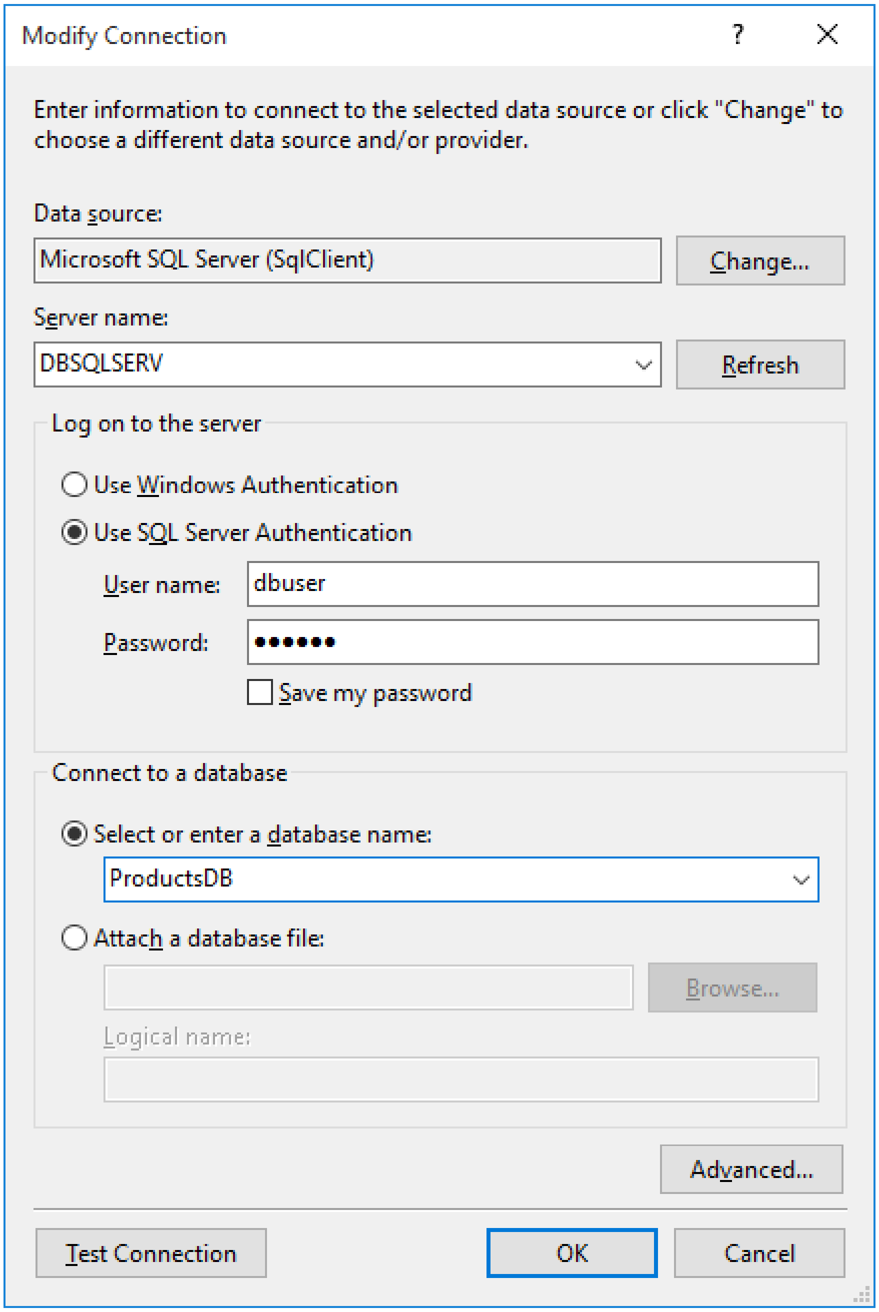


Choose Data Source
?
Data source:
Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Database File
Oracle Database
<other>
Description
Data provider:
.NET Framework Data Provider for SQL S
Continue
Cancel

Use this selection to connect to Microsoft SQL Server 2005 or above, or to Microsoft SQL Azure using the .NET Framework Data Provider for SQL Server.

□ Always use this selection

3. Click **Continue**.



Modify Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: DBSQLSERV

Log on to the server

○ Use Windows Authentication

● Use SQL Server Authentication

User name: dbuser

Password: ●●●●●●

□ Save my password

Connect to a database

● Select or enter a database name: ProductsDB

○ Attach a database file:

Logical name:

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.

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). 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 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).

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

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. See Setting up an ODBC Connection.
	.Net Framework Data Provider for OleDb	Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ADO direct connections instead. See Setting up an

Database	Driver	Support notes
		ADO Connection.
Firebird	**Firebird ADO.NET Data Provider**	Limited support. It is recommended to use ODBC or JDBC instead. See Connecting to Firebird (ODBC) and Connecting to Firebird (JDBC).
Informix	**IBM Informix Data Provider for .NET Framework 4.0**	Not supported. Use **DB2 Data Server 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.
Sybase	**-**	No ADO.NET drivers for this vendor are supported.

7.2.1.5 Setting up an ODBC Connection

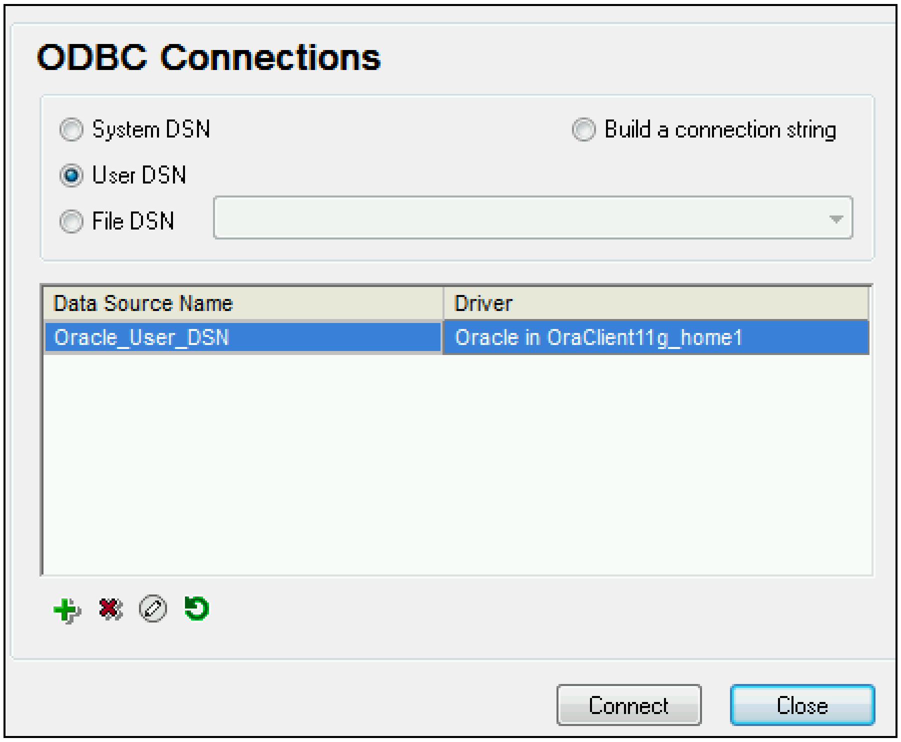
ODBC (Open Database Connectivity) is a widely used data access technology that enables you to connect to a database from MapForce. It can be used either as primary means to connect to a database, or as an alternative to OLE DB- or JDBC-driven connections.

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

○ System DSN
● User DSN
○ File DSN
○ Build a connection string

Data Source Name	Driver
Oracle_User_DSN	Oracle in OraClient11g_home1

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).

To connect by using a new DSN:

1. Start the database connection wizard.
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).
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.
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.
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:

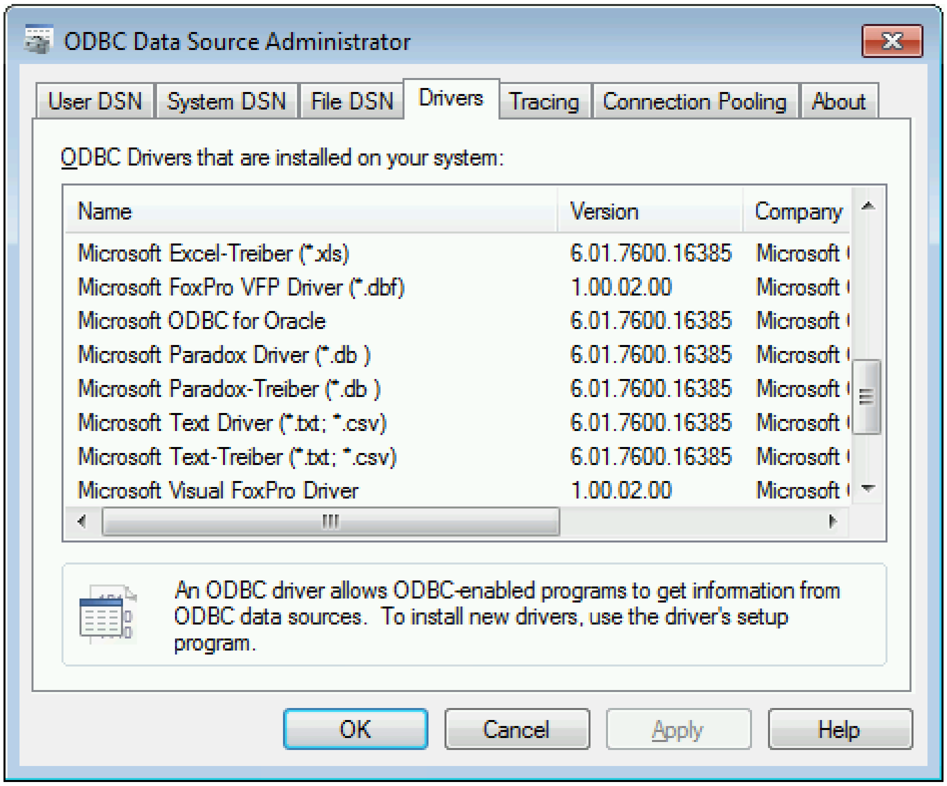
1. Start the database connection wizard.
2. Click **ODBC Connections**.
3. Select **Build a connection string**.
4. Paste the connection string into the provided box, and then click **Connect**.

Viewing the 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 Drivers that are installed on your system:

Name	Version	Company
Microsoft Excel-Treiber (*.xls)	6.01.7600.16385	Microsoft (
Microsoft FoxPro VFP Driver (*.dbf)	1.00.02.00	Microsoft (
Microsoft ODBC for Oracle	6.01.7600.16385	Microsoft (
Microsoft Paradox Driver (*.db)	6.01.7600.16385	Microsoft (
Microsoft Paradox-Treiber (*.db)	6.01.7600.16385	Microsoft (
Microsoft Text Driver (*.txt; *.csv)	6.01.7600.16385	Microsoft (
Microsoft Text-Treiber (*.txt; *.csv)	6.01.7600.16385	Microsoft (
Microsoft Visual FoxPro Driver	1.00.02.00	Microsoft (

An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

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). Once the ODBC driver is available on your system, you are ready to create ODBC connections with it (see Setting up an ODBC Connection).

7.2.1.6 Setting up a 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

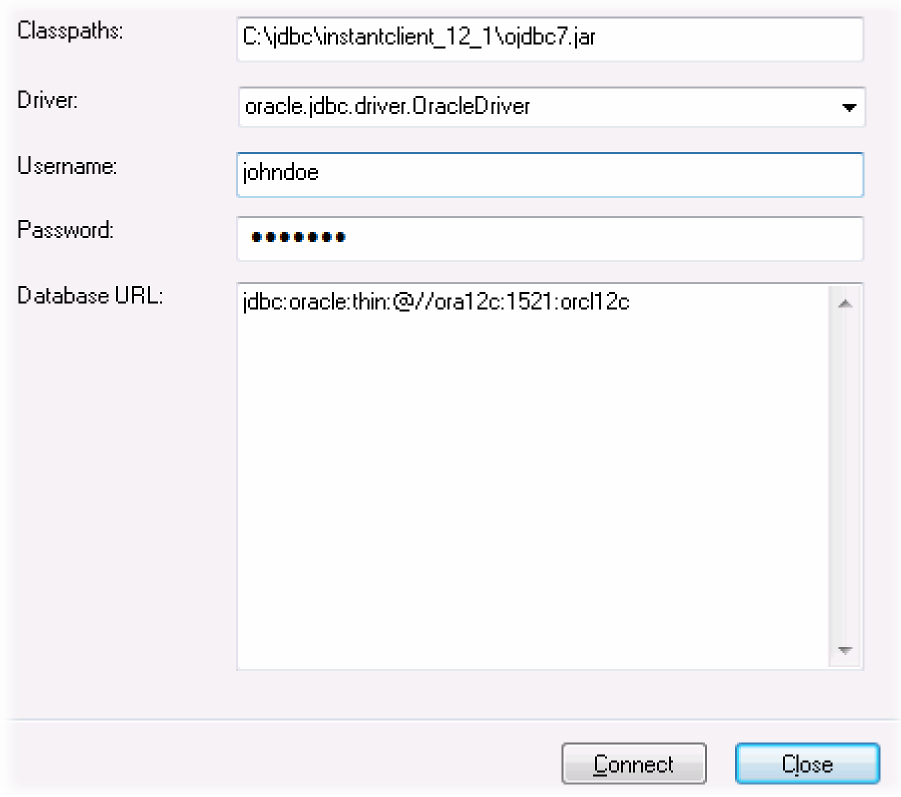
- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. If you have not installed it already, check the official Java website for the download package and installation instructions.
- The JDBC drivers from the database vendor must be installed. If you are connecting to an Oracle database, note that some Oracle drivers are specific to certain JRE versions and may require additional components and configuration. The documentation of your Oracle product (for example, the "Oracle Database JDBC Developer's Guide and Reference") includes detailed instructions about the configuration procedure for each JDBC driver.
- 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 `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. The documentation of the JDBC driver will typically include step-by-step instructions on setting the CLASSPATH variable (see also Configuring the CLASSPATH).

Setting up a JDBC connection

1. Start the database connection wizard.
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).



Classpaths: C:\jdbc\instantclient_12_1\ojdbc7.jar

Driver: oracle.jdbc.driver.OracleDriver

Username: johndoe

Password: •••••••

Database URL: jdbc:oracle:thin:@//ora12c:1521:orcl12c

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), 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 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`**`:`**`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**.

Configuring the CLASSPATH

The `CLASSPATH` environment variable is used by the Java Runtime Environment (JRE) 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 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;`
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 (;).

7.2.1.7 *Setting up a 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 OS X server, no drivers are required to be installed on the target server as well (see also Database Connections on Linux and Mac).

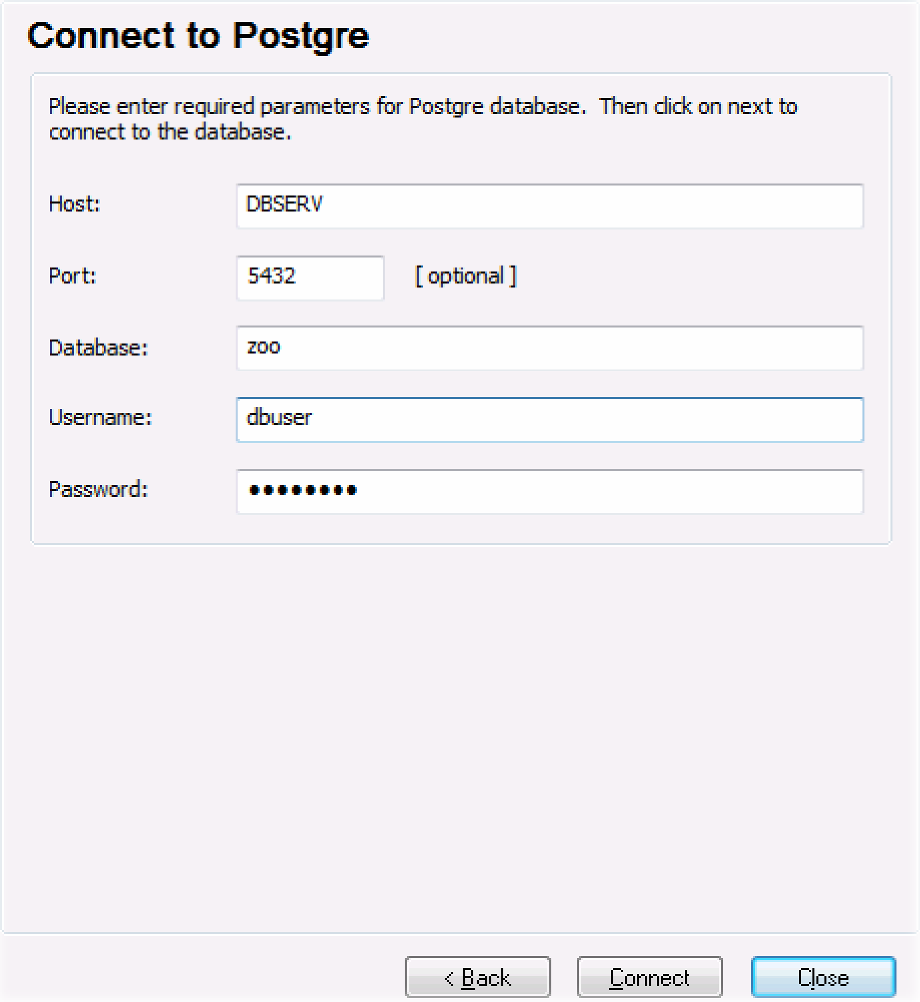
If you prefer to establish a connection by means of a non-native driver, see the following topics:

- Setting up a JDBC Connection
- Connecting to PostgreSQL (ODBC)

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.
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.



Connect to Postgre

Please enter required parameters for Postgre database. Then click on next to connect to the database.

Host: DBSERV

Port: 5432 [optional]

Database: zoo

Username: dbuser

Password: •••••••••

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**.

7.2.1.8 *Setting up a SQLite Connection*

SQLite (https://www.sqlite.org/index.html) 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

- 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).
- 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. In MapForce, all processed values must be compatible with the declared column type; therefore, 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 either create it with a tool such as DatabaseSpy (https://www.altova.com/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). For complete reference to SQLite command syntax, refer to the official SQLite documentation.

Connecting to an Existing SQLite Database

To connect to an existing SQLite database:

1. Run the database connection wizard (see Starting the Database Connection Wizard).
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**.

7.2.1.9 *Using a Connection from Global Resources*

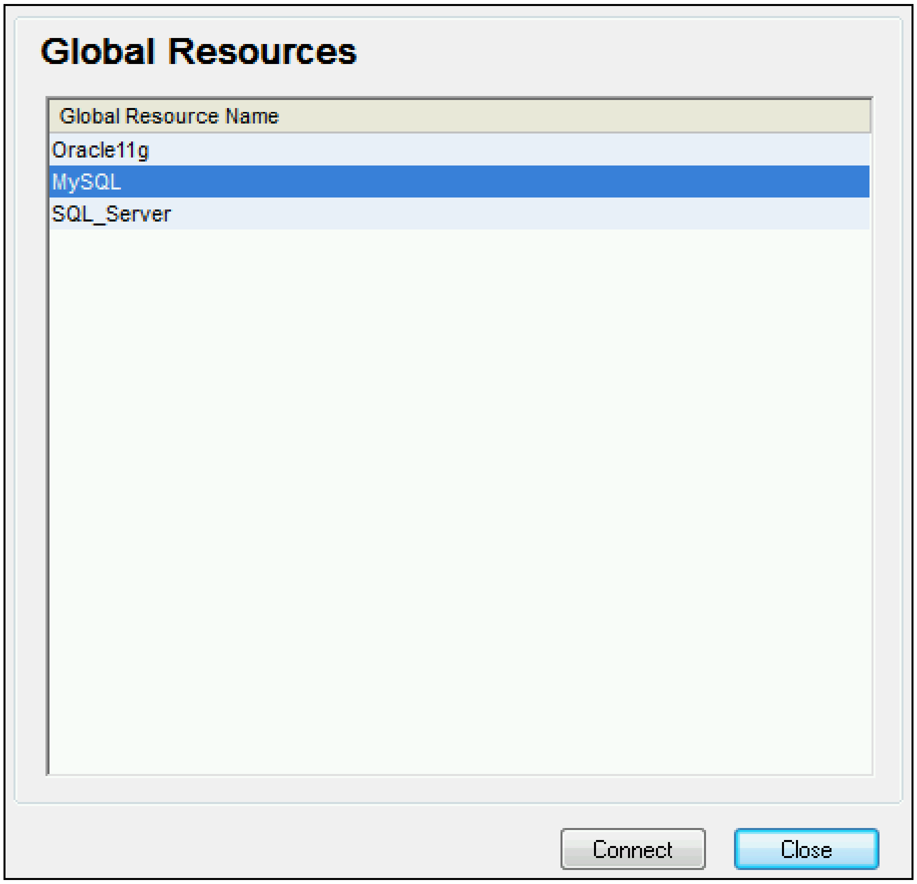
Altova Global Resources represent a way to refer to files, folders, or databases so as to make these resources reusable, configurable and available across multiple Altova applications.

If you have already configured a database connection to be available as a global resource, you can reuse the connection at any time (even across different Altova applications).

To use a database connection from Global Resources:

1. Start the database connection wizard.

2. Click **Global Resources**. Any database connections previously configured as global resources are listed.



Global Resources

Global Resource Name
Oracle11g
MySQL
SQL_Server

3. Select the database connection record, and click **Connect**.

Tip: To get additional information about each global resource, move the mouse cursor over the global resource.

7.2.1.10 Database Connection Examples

This section includes sample procedures for connecting to a database from MapForce. Note that your Windows machine, the network environment, and the database client or server software is likely to have a configuration that is not exactly the same as the one presented in the following examples.

Note: 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.

Connecting to Firebird (ODBC)

This topic provides sample instructions for connecting to a Firebird 2.5.4 database running on a Linux server.

Prerequisites:

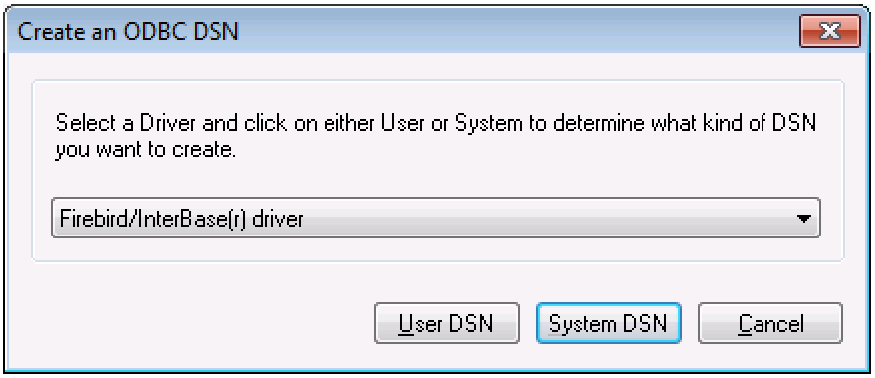
- The Firebird database server is configured to accept TCP/IP connections from clients.
- The Firebird ODBC driver must be installed on your operating system. This example uses the Firebird ODBC driver version 2.0.3.154 downloaded from the Firebird website (https://www.firebirdsql.org/).
- The Firebird 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.
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** ✚ .



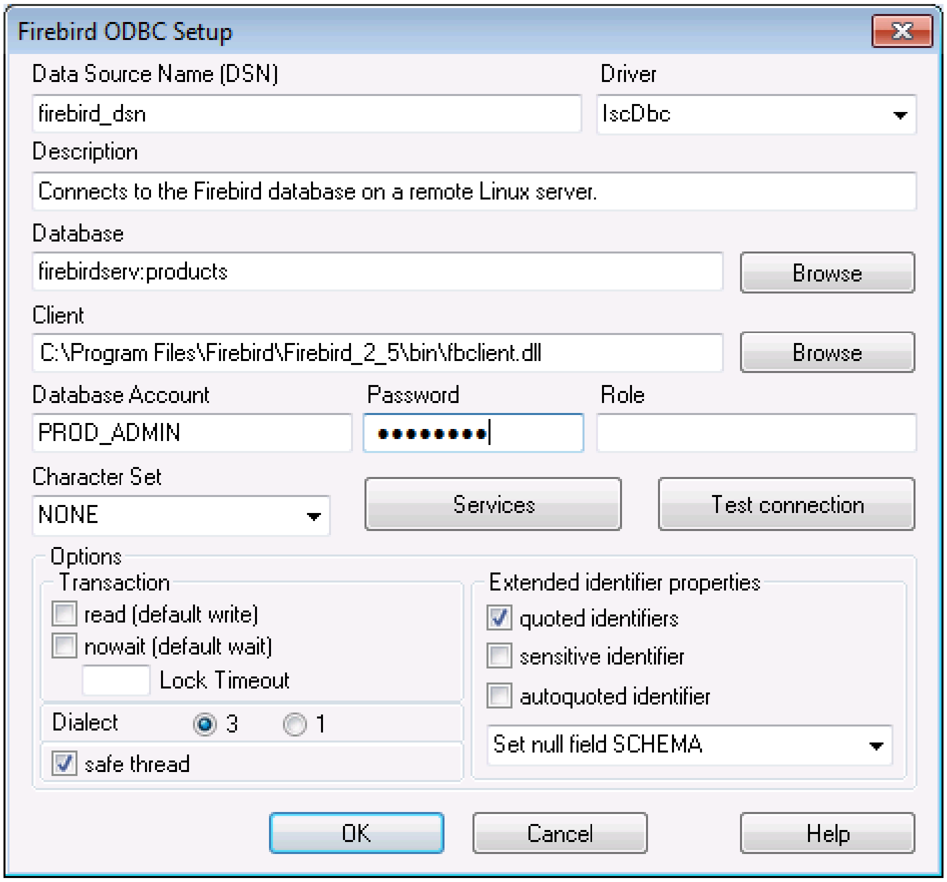
Create an ODBC DSN

Select a Driver and click on either User or System to determine what kind of DSN you want to create.

Firebird/InterBase(r) driver

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).



Firebird ODBC Setup

Data Source Name (DSN): firebird_dsn

Driver: IscDbc

Description: Connects to the Firebird database on a remote Linux server.

Database: firebirdserv:products

Client: C:\Program Files\Firebird\Firebird_2_5\bin\fbclient.dll

Database Account: PROD_ADMIN

Password: ●●●●●●●●

Role:

Character Set: NONE

Options

Transaction

□ read (default write)

□ nowait (default wait)

Lock Timeout

Dialect ● 3 ○ 1

☑ safe thread

Extended identifier properties

☑ quoted identifiers

□ sensitive identifier

□ autoquoted identifier

Set null field SCHEMA

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**.

Connecting to Firebird (JDBC)

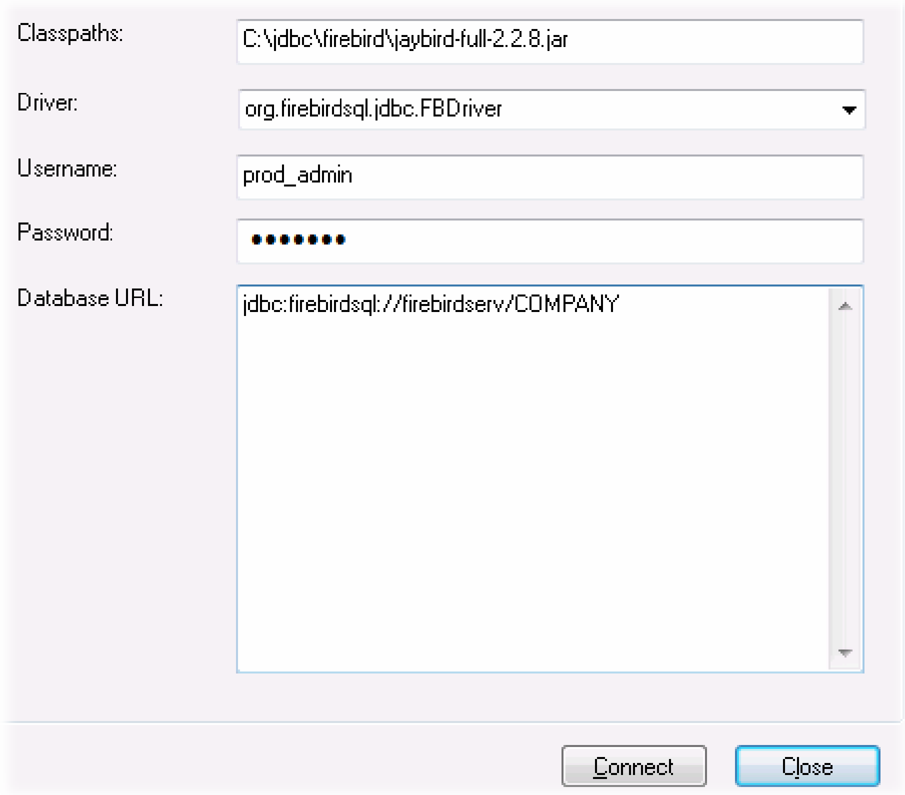
This topic provides sample instructions for connecting to a Firebird database server through JDBC.

Prerequisites:

- Java Runtime Environment (JRE) or Java Development Kit (JDK) must be installed on your operating system.
- 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 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.
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).
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).



Classpaths: C:\jdbc\firebird\jaybird-full-2.2.8.jar

Driver: org.firebirdsql.jdbc.FBDriver

Username: prod_admin

Password: •••••••

Database URL: jdbc:firebirdsql://firebirdserv/COMPANY

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**.

Connecting to IBM DB2 (ODBC)

This topic provides sample instructions for connecting to an IBM DB2 database through ODBC.

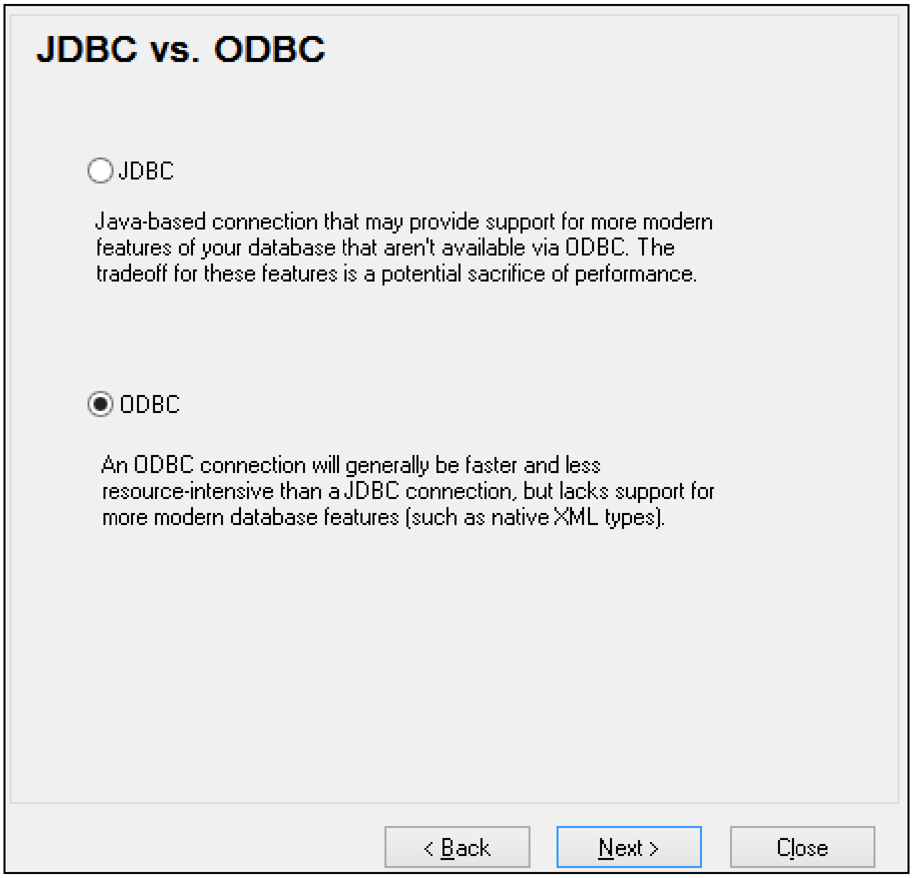
Prerequisites:

- 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).
- 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 and select **IBM DB2 (ODBC/JDBC)**.
2. Click **Next**.



JDBC vs. ODBC

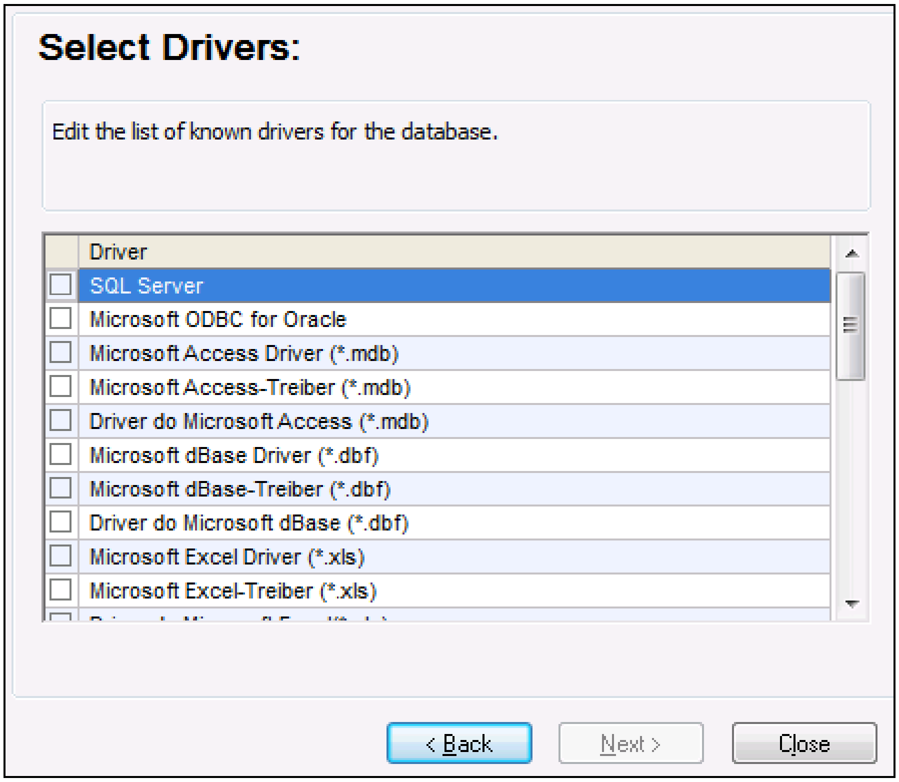
○ JDBC

Java-based connection that may provide support for more modern features of your database that aren't available via ODBC. The tradeoff for these features is a potential sacrifice of performance.

● ODBC

An ODBC connection will generally be faster and less resource-intensive than a JDBC connection, but lacks support for more modern database features (such as native XML types).

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), and click **Next**.

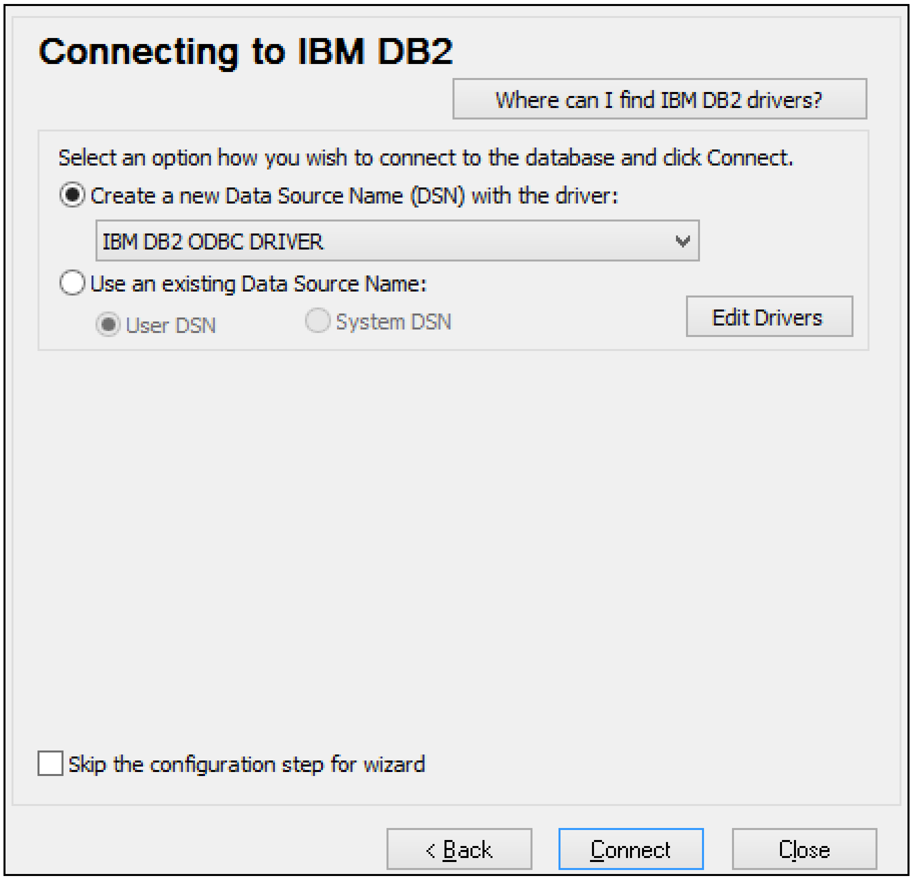


Select Drivers:

Edit the list of known drivers for the database.

	Driver
□	SQL Server
□	Microsoft ODBC for Oracle
□	Microsoft Access Driver (*.mdb)
□	Microsoft Access-Treiber (*.mdb)
□	Driver do Microsoft Access (*.mdb)
□	Microsoft dBase Driver (*.dbf)
□	Microsoft dBase-Treiber (*.dbf)
□	Driver do Microsoft dBase (*.dbf)
□	Microsoft Excel Driver (*.xls)
□	Microsoft Excel-Treiber (*.xls)

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.)



Connecting to IBM DB2

Select an option how you wish to connect to the database and click Connect.

● Create a new Data Source Name (DSN) with the driver:

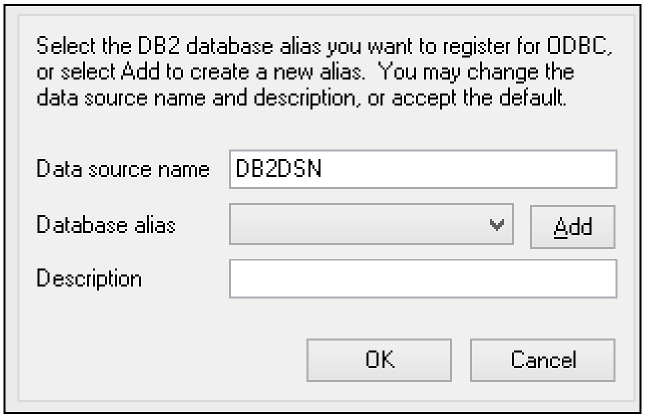
IBM DB2 ODBC DRIVER

○ Use an existing Data Source Name:

● User DSN ○ System DSN

□ Skip the configuration step for wizard

5. Enter a data source name (in this example, **DB2DSN**), and then click **Add**.



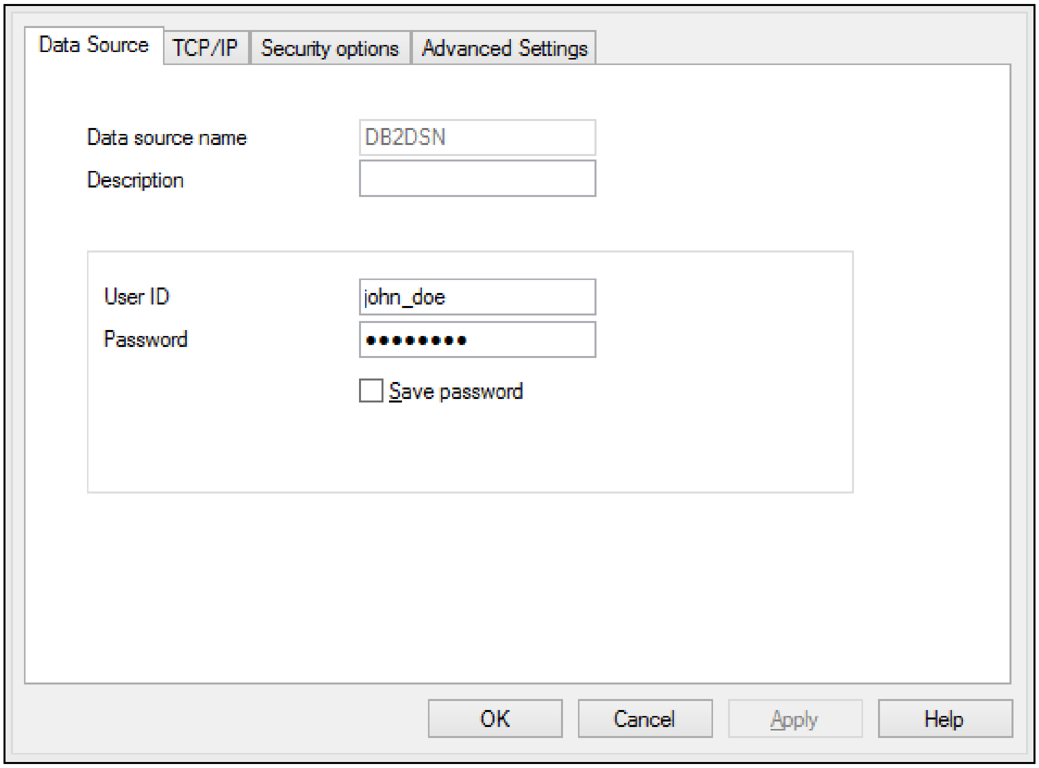
Select the DB2 database alias you want to register for ODBC, or select Add to create a new alias. You may change the data source name and description, or accept the default.

Data source name: DB2DSN

Database alias:

Description:

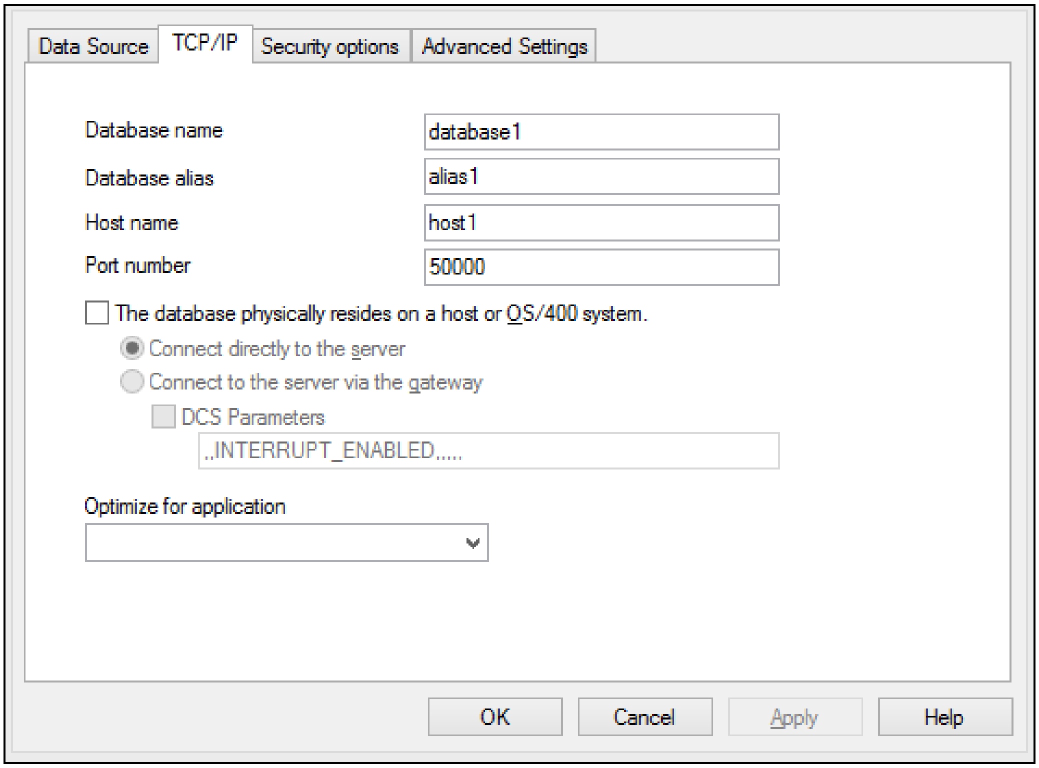
6. On the **Data Source** tab, enter the user name and password to the database.



Data source name: DB2DSN
Description:

User ID: john_doe
Password: ●●●●●●●●
□ Save password

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.

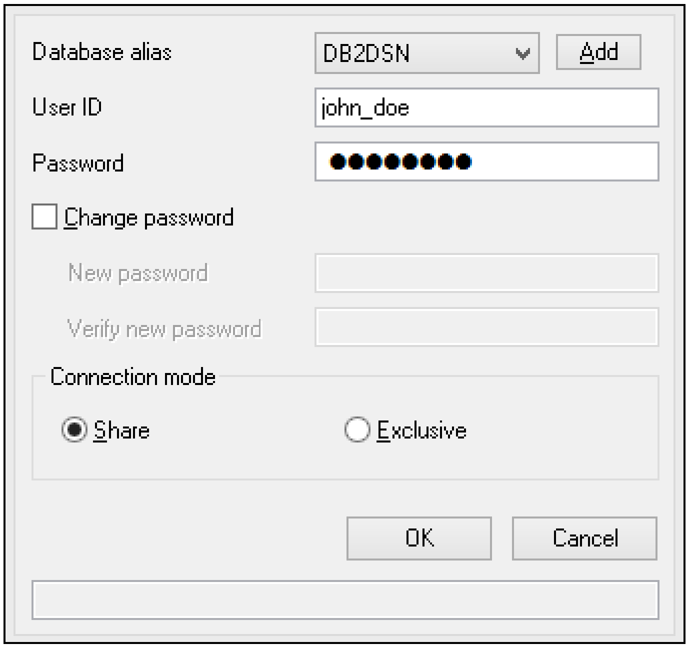


Database name: database1
Database alias: alias1
Host name: host1
Port number: 50000

□ The database physically resides on a host or OS/400 system.
● Connect directly to the server
○ Connect to the server via the gateway
□ DCS Parameters
,,INTERRUPT_ENABLED,,,,,

Optimize for application:

8. Enter again the username and password, and then click **OK**.



Database alias: DB2DSN

User ID: john_doe

Password: ●●●●●●●●●

□ Change password

New password

Verify new password

Connection mode

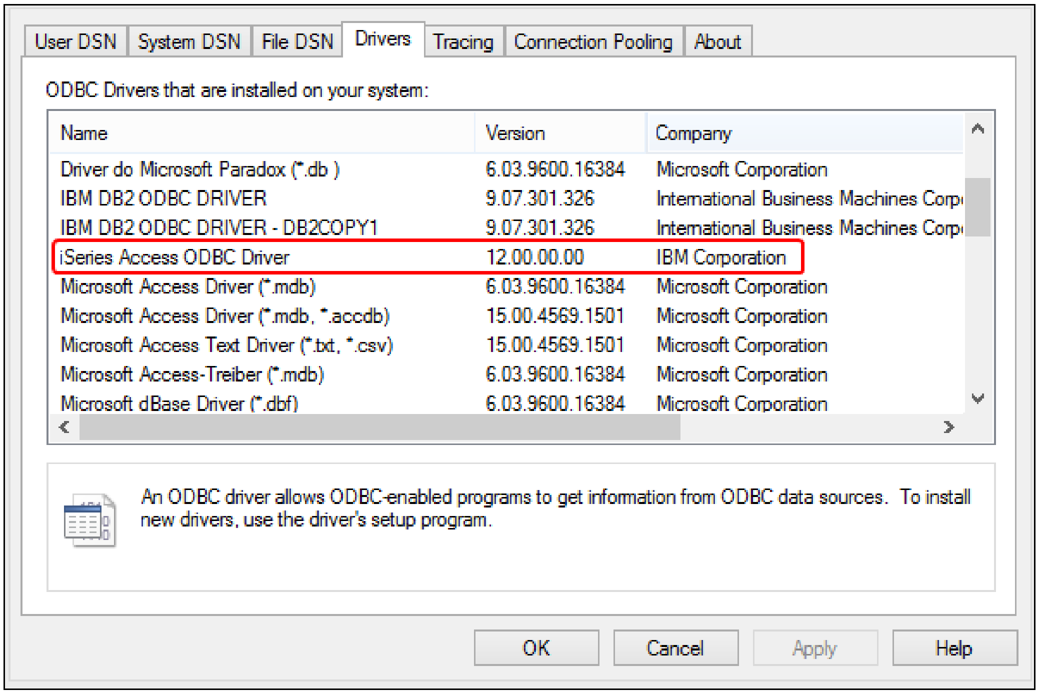
● Share ○ Exclusive

Connecting to IBM DB2 for i (ODBC)

This topic provides sample instructions for connecting 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).



ODBC Drivers that are installed on your system:

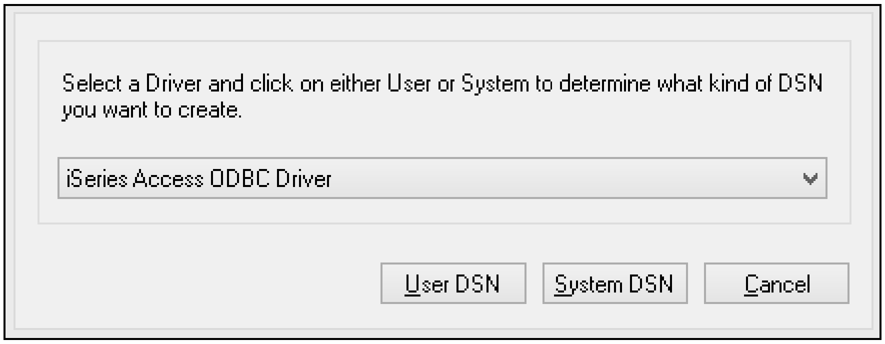
Name	Version	Company
Driver do Microsoft Paradox (*.db)	6.03.9600.16384	Microsoft Corporation
IBM DB2 ODBC DRIVER	9.07.301.326	International Business Machines Corp
IBM DB2 ODBC DRIVER - DB2COPY1	9.07.301.326	International Business Machines Corp
iSeries Access ODBC Driver	12.00.00.00	IBM Corporation
Microsoft Access Driver (*.mdb)	6.03.9600.16384	Microsoft Corporation
Microsoft Access Driver (*.mdb, *.accdb)	15.00.4569.1501	Microsoft Corporation
Microsoft Access Text Driver (*.txt, *.csv)	15.00.4569.1501	Microsoft Corporation
Microsoft Access-Treiber (*.mdb)	6.03.9600.16384	Microsoft Corporation
Microsoft dBase Driver (*.dbf)	6.03.9600.16384	Microsoft Corporation

An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

- 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:

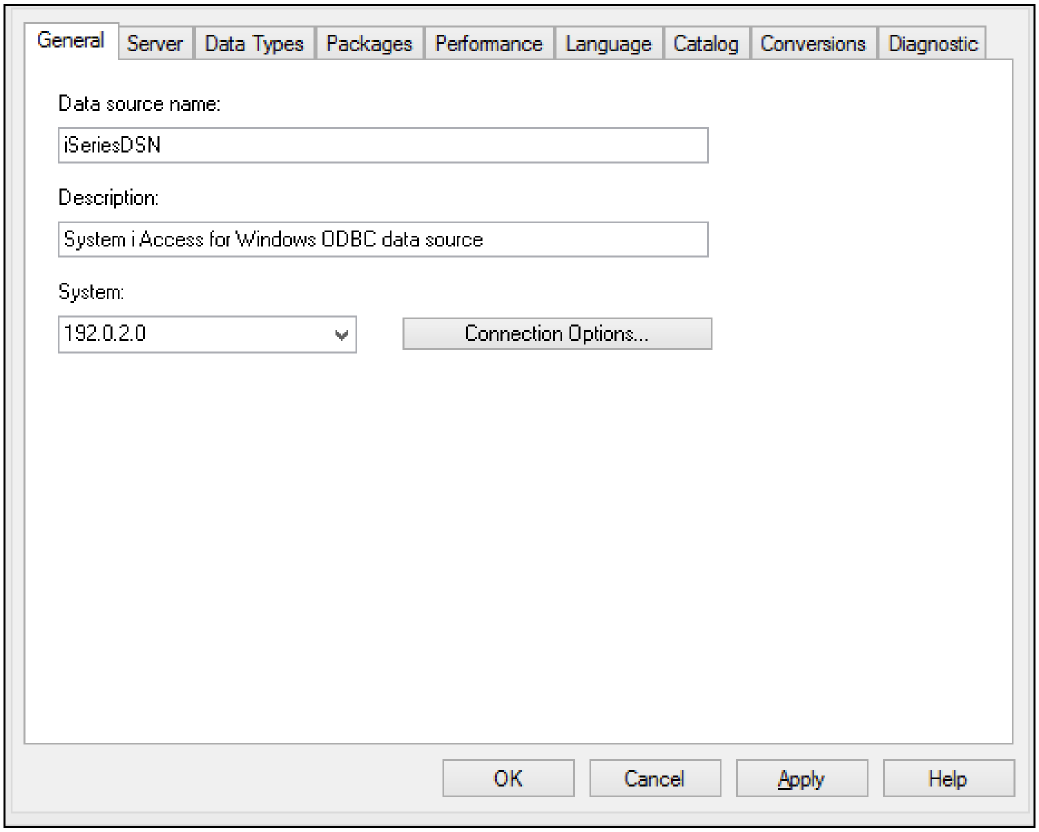
1. Start the database connection wizard.
2. Click **ODBC connections**.
3. Click **User DSN** (alternatively, click **System DSN**, or **File DSN**, in which case the subsequent instructions will be similar).
4. Click **Add** ✚ .
5. Select the **iSeries Access ODBC Driver** from the list, and click **User DSN** (or **System DSN**, if applicable).


Select a Driver and click on either User or System to determine what kind of DSN you want to create.

iSeries Access ODBC Driver

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**.



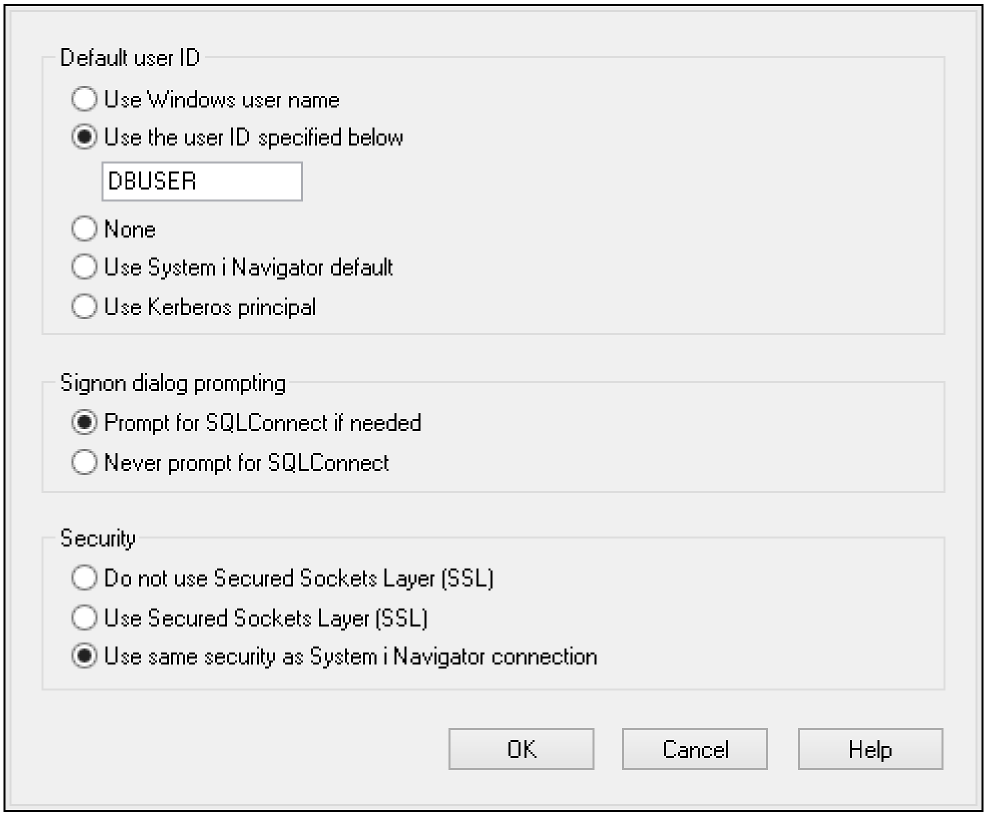
Data source name:
iSeriesDSN

Description:
System i Access for Windows ODBC data source

System:
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**).



Default user ID

○ Use Windows user name
● Use the user ID specified below

DBUSER

○ None
○ Use System i Navigator default
○ Use Kerberos principal

Signon dialog prompting

● Prompt for SQLConnect if needed
○ Never prompt for SQLConnect

Security

○ Do not use Secured Sockets Layer (SSL)
○ Use Secured Sockets Layer (SSL)
● Use same security as System i Navigator connection

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**.

Connecting to IBM Informix (JDBC)

This topic provides sample instructions for connecting to an IBM Informix database server through JDBC.

Prerequisites:

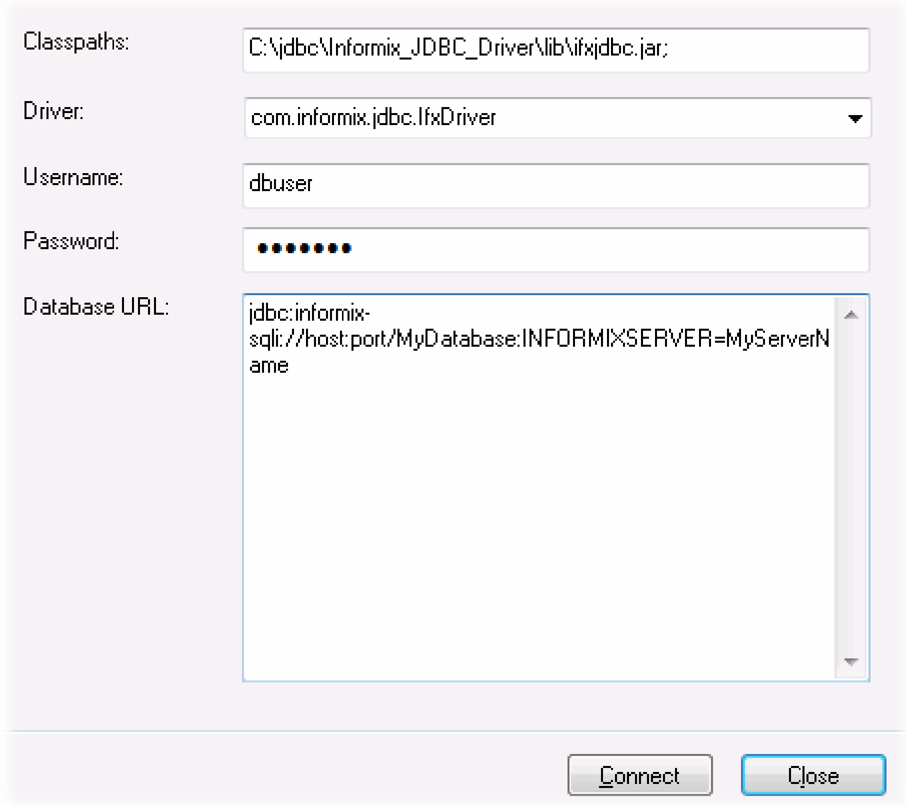
- Java Runtime Environment (JRE) must be installed on your operating system.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. In this example, IBM Informix JDBC driver version 3.70 is used. For the driver's installation 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.
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).

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 "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



Classpaths: C:\jdbc\Informix_JDBC_Driver\lib\ifxjdbc.jar;

Driver: com.informix.jdbc.IfxDriver

Username: dbuser

Password: ●●●●●●●

Database URL: jdbc:informix-sqli://host:port/MyDatabase:INFORMIXSERVER=MyServerName

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**.

Connecting to MariaDB (ODBC)

This example illustrates how to connect to a MariaDB database server through ODBC.

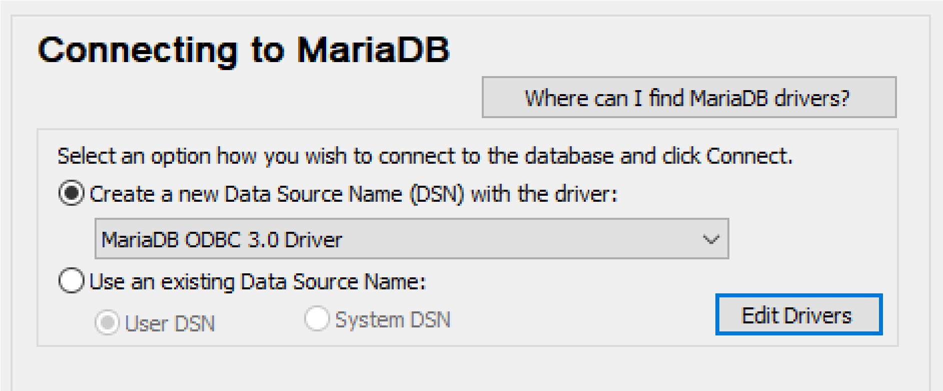
Prerequisites:

- 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.
2. Select **MariaDB (ODBC)**, and then click **Next**.



Connecting to MariaDB

Select an option how you wish to connect to the database and click Connect.

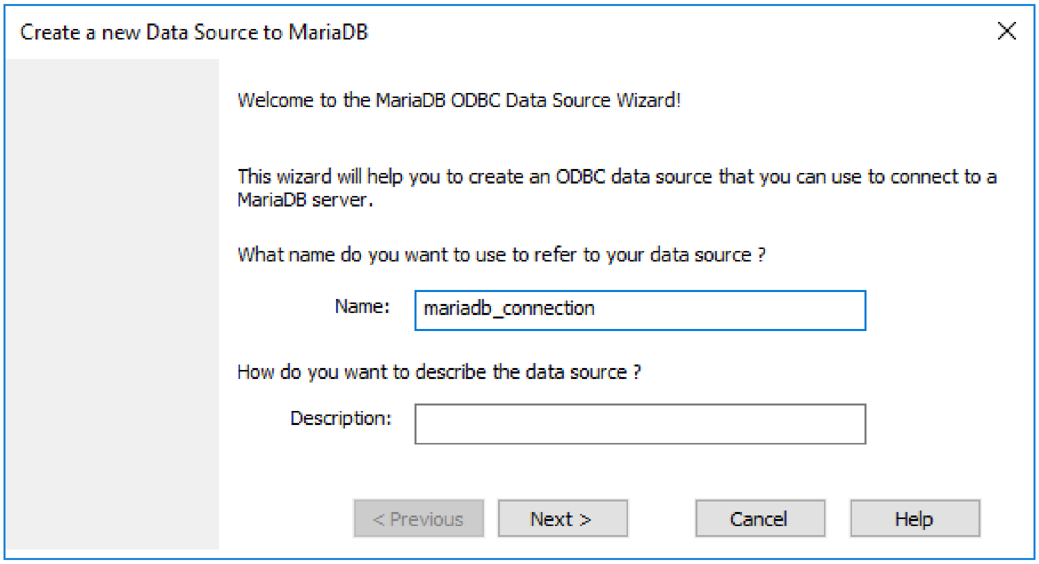
● Create a new Data Source Name (DSN) with the driver:

MariaDB ODBC 3.0 Driver

○ Use an existing Data Source Name:

● User DSN ○ System DSN

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**.



Create a new Data Source to MariaDB

Welcome to the MariaDB ODBC Data Source Wizard!

This wizard will help you to create an ODBC data source that you can use to connect to a MariaDB server.

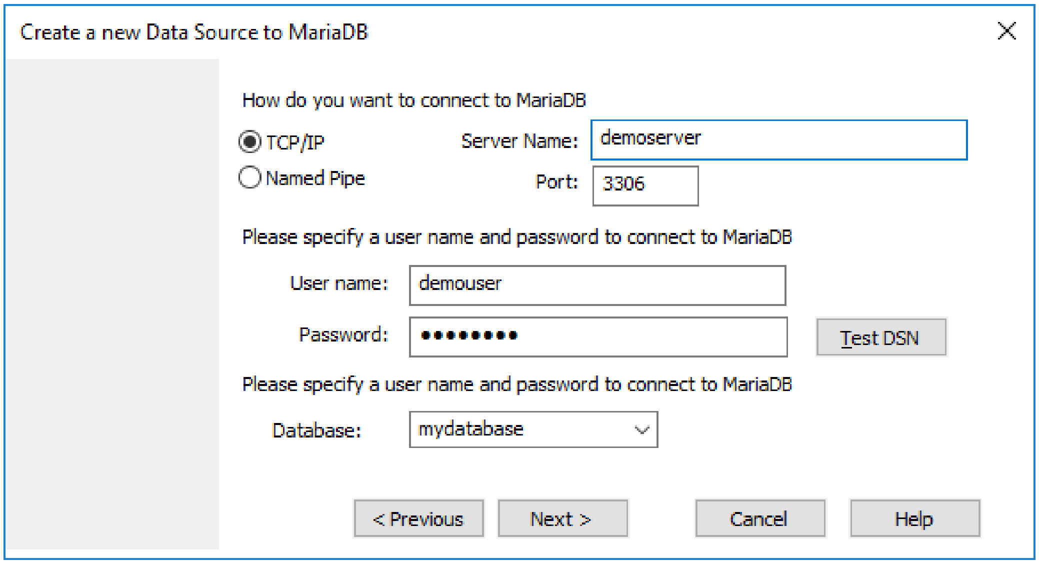
What name do you want to use to refer to your data source ?

Name: mariadb_connection

How do you want to describe the data source ?

Description:

5. Enter name and, optionally, a description that will help you identify this ODBC data source in future.



Create a new Data Source to MariaDB

How do you want to connect to MariaDB

● TCP/IP Server Name: demoserver

○ Named Pipe Port: 3306

Please specify a user name and password to connect to MariaDB

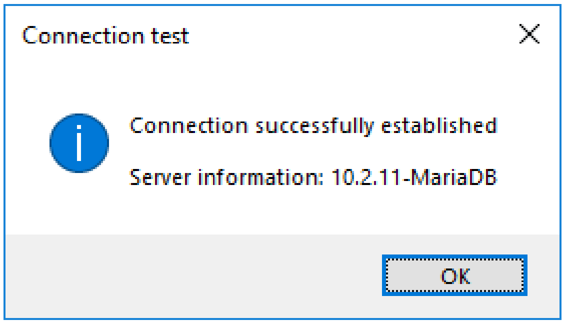
User name: demouser

Password: ●●●●●●●●●

Please specify a user name and password to connect to MariaDB

Database: mydatabase

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:



Connection test

Connection successfully established

Server information: 10.2.11-MariaDB

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.

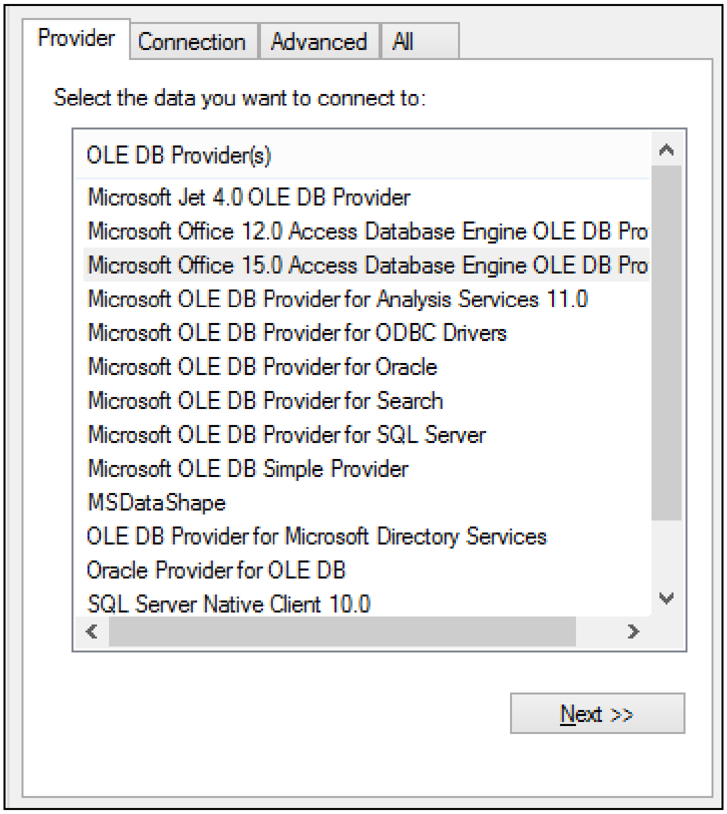
Connecting to Microsoft Access (ADO)

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. 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.

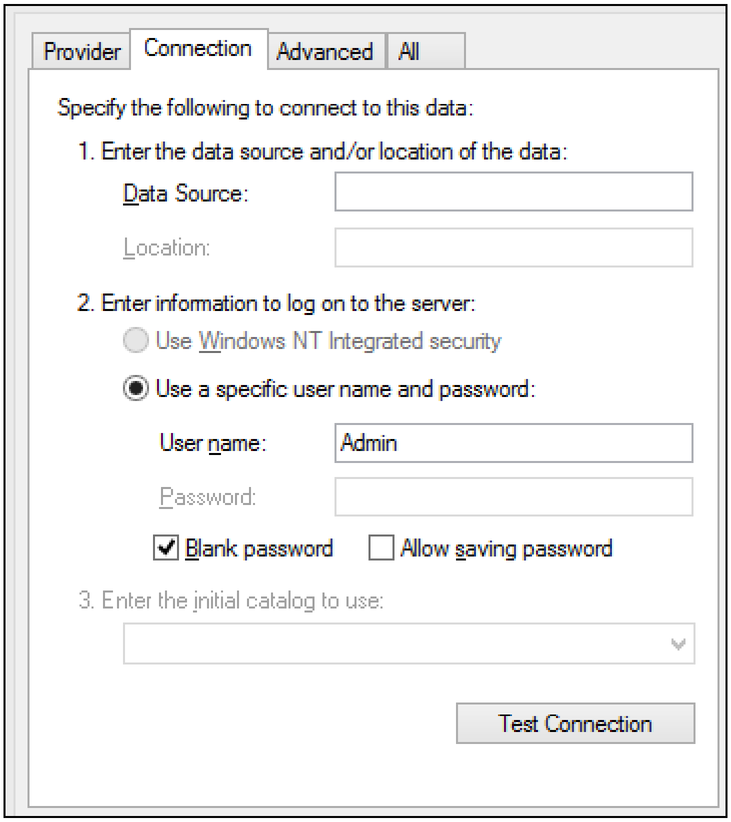
It is also possible to connect to Microsoft Access through an ODBC connection, but there are some limitations in this scenario, so it is best to avoid it.

To connect to a password-protected Microsoft Access database:

1. Start the database connection wizard.
2. Click **ADO Connections**.
3. Click **Build**.


Provider
Connection
Advanced
All
Select the data you want to connect to:
OLE DB Provider(s)
Microsoft Jet 4.0 OLE DB Provider
Microsoft Office 12.0 Access Database Engine OLE DB Pro
Microsoft Office 15.0 Access Database Engine OLE DB Pro
Microsoft OLE DB Provider for Analysis Services 11.0
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for Search
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
MSDataShape
OLE DB Provider for Microsoft Directory Services
Oracle Provider for OLE DB
SQL Server Native Client 10.0
Next >>

4. Select the **Microsoft Office 15.0 Access Database Engine OLE DB Provider**, and then click **Next**.



Specify the following to connect to this data:

1. Enter the data source and/or location of the data:

Data Source:

Location:

2. Enter information to log on to the server:

○ Use Windows NT Integrated security

● Use a specific user name and password:

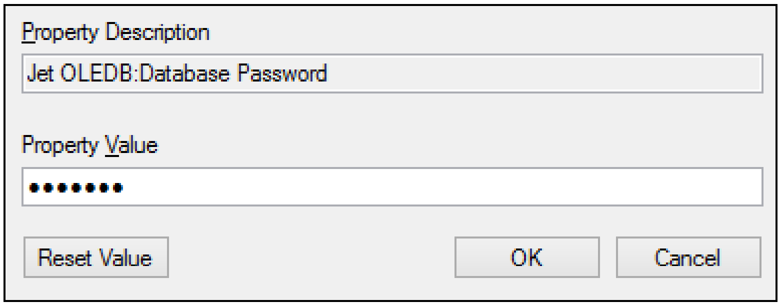
User name: Admin

Password:

☑ Blank password □ Allow saving password

3. Enter the initial catalog to use:

5. In the Data Source box, enter the path to the Microsoft Access file. Because the file is on the local network share **U:\Departments\Finance\Reports\Revenue.accdb**, we will convert it to UNC format, and namely **\\server1\\dfs\Departments\Finance\Reports\Revenue.accdb**, where **server1** is the name of the server and **dfs** 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.



Property Description

Jet OLEDB:Database Password

Property Value

•••••••

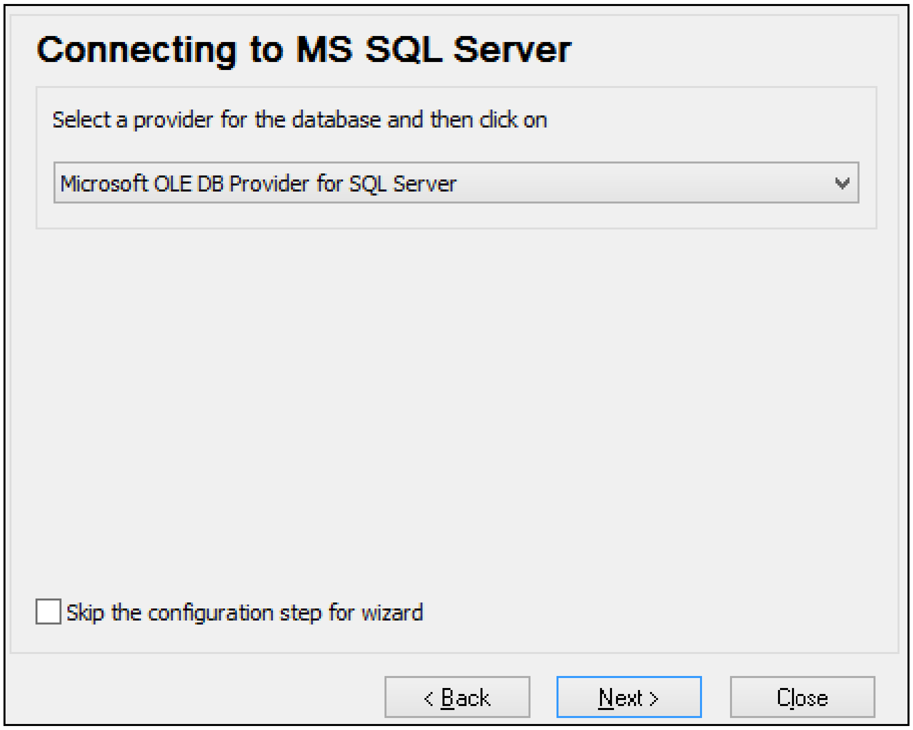
Note: If you are still unable to connect, locate the workgroup information file (**System.MDW**) applicable to your user profile (see http://support.microsoft.com/kb/305542 for instructions), and set the value of the **Jet OLEDB: System database** property to the path of the **System.MDW** file.

Connecting to Microsoft SQL Server (ADO)

This example illustrates how to connect to a SQL Server database through ADO.

To connect to SQL Server using the Microsoft OLE DB Provider:

1. [Start the database connection wizard](#).
2. Select **Microsoft SQL Server (ADO)**, and then click **Next**. The list of available ADO drivers is displayed.



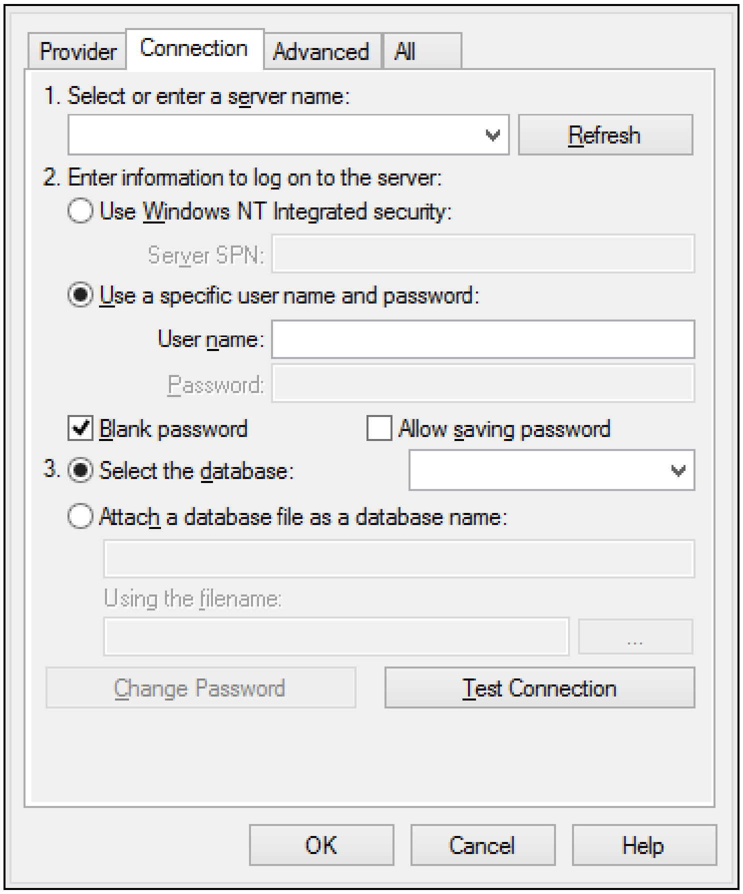
Connecting to MS SQL Server

Select a provider for the database and then click on

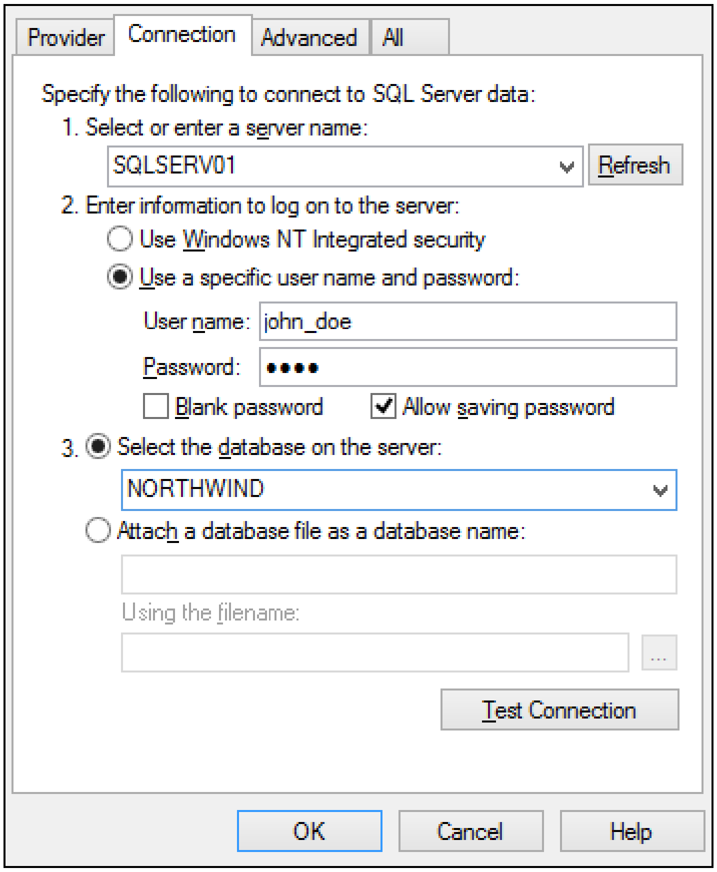
Microsoft OLE DB Provider for SQL Server

□ Skip the configuration step for wizard

3. Select **Microsoft OLE DB Provider for SQL Server**, and then click **Next**.


Provider
Connection
Advanced
All
1. Select or enter a server name:
Refresh
2. Enter information to log on to the server:
○ Use Windows NT Integrated security:
Server SPN:
● Use a specific user name and password:
User name:
Password:
☑ Blank password
□ Allow saving password
3. ● Select the database:
○ Attach a database file as a database name:
Using the filename:
...
Change Password
Test Connection
OK
Cancel
Help

4. Select or enter the name of the database server (in this example, **SQLSERV01**). To view the list of all servers on the network, expand the drop-down list.
5. If the database server was configured to allow connections from users authenticated on the Windows domain, select **Use Windows NT integrated security**. Otherwise, select **Use a specific user name and password**, and type them in the relevant boxes.
6. Select the database to which you are connecting (in this example, **NORTHWIND**).
7. To test the connection at this time, click **Test Connection**. This is an optional, recommended step.
8. Do one of the following:
   a. Select the **Allow saving password** check box.
   b. On the **All** tab, change the value of the **Persist Security Info** property to **True**.



Specify the following to connect to SQL Server data:

1. Select or enter a server name: SQLSERV01

2. Enter information to log on to the server:

○ Use Windows NT Integrated security

● Use a specific user name and password:

User name: john_doe

Password: ●●●●

□ Blank password ☑ Allow saving password

3. ● Select the database on the server: NORTHWIND

○ Attach a database file as a database name:

Using the filename:

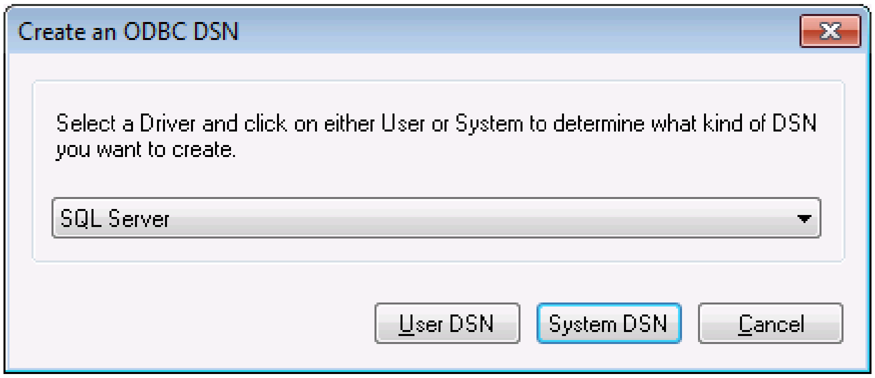
9. Click **OK**.

Connecting to Microsoft SQL Server (ODBC)

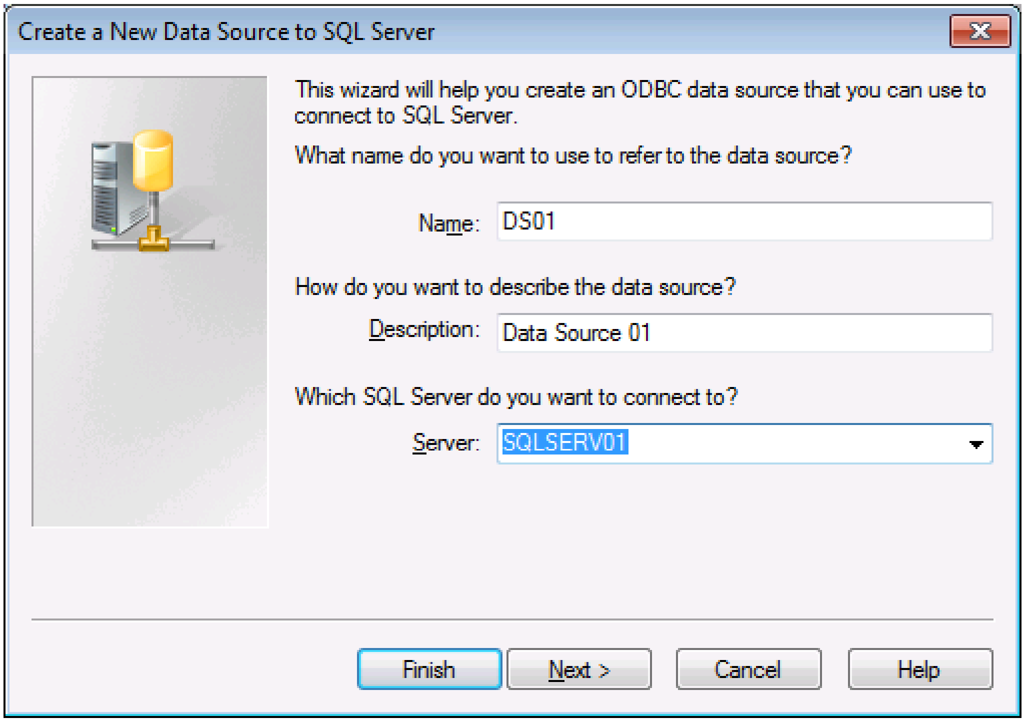
This example illustrates how to connect to a SQL Server database through ODBC.

To connect to SQL Server using ODBC:

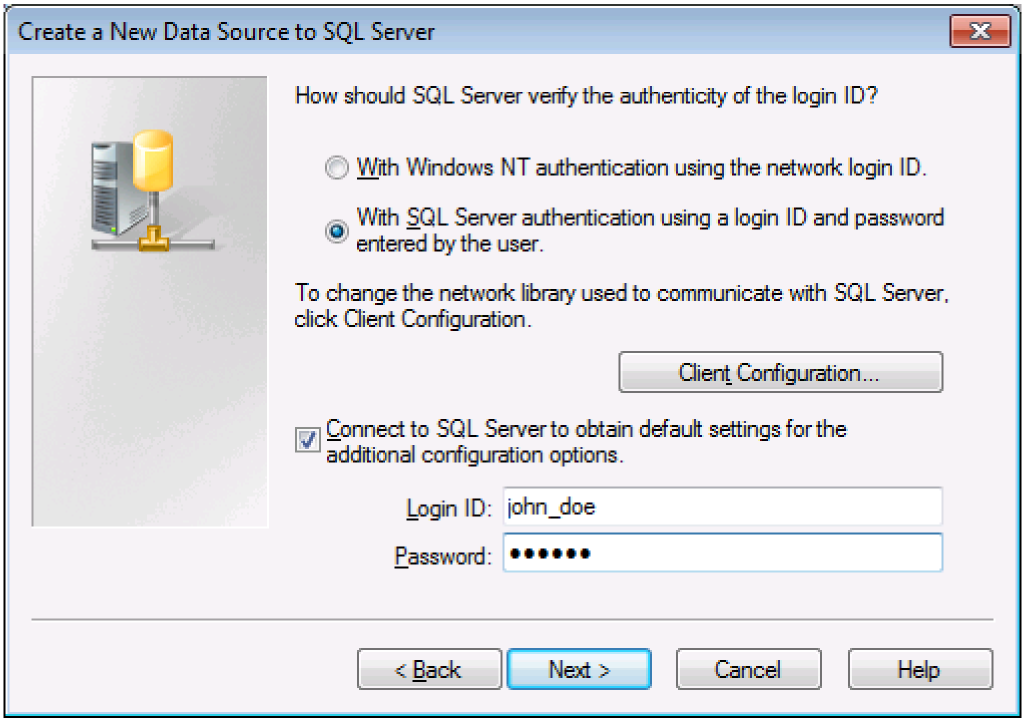
1. Start the database connection wizard.
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .


Create an ODBC DSN
Select a Driver and click on either User or System to determine what kind of DSN you want to create.
SQL Server
User DSN
System DSN
Cancel

4. Select **SQL Server** (or **SQL Server Native Client**, if available), and then click **User DSN** (or **System DSN** if you are creating a System DSN).


Create a New Data Source to SQL Server
This wizard will help you create an ODBC data source that you can use to connect to SQL Server.
What name do you want to use to refer to the data source?
Name: DS01
How do you want to describe the data source?
Description: Data Source 01
Which SQL Server do you want to connect to?
Server: SQLSERV01
Finish
Next >
Cancel
Help

5. Enter a name and description to identify this connection, and then select from the list the SQL Server to which you are connecting (**SQLSERV01** in this example).



Create a New Data Source to SQL Server

How should SQL Server verify the authenticity of the login ID?

○ With Windows NT authentication using the network login ID.

● With SQL Server authentication using a login ID and password entered by the user.

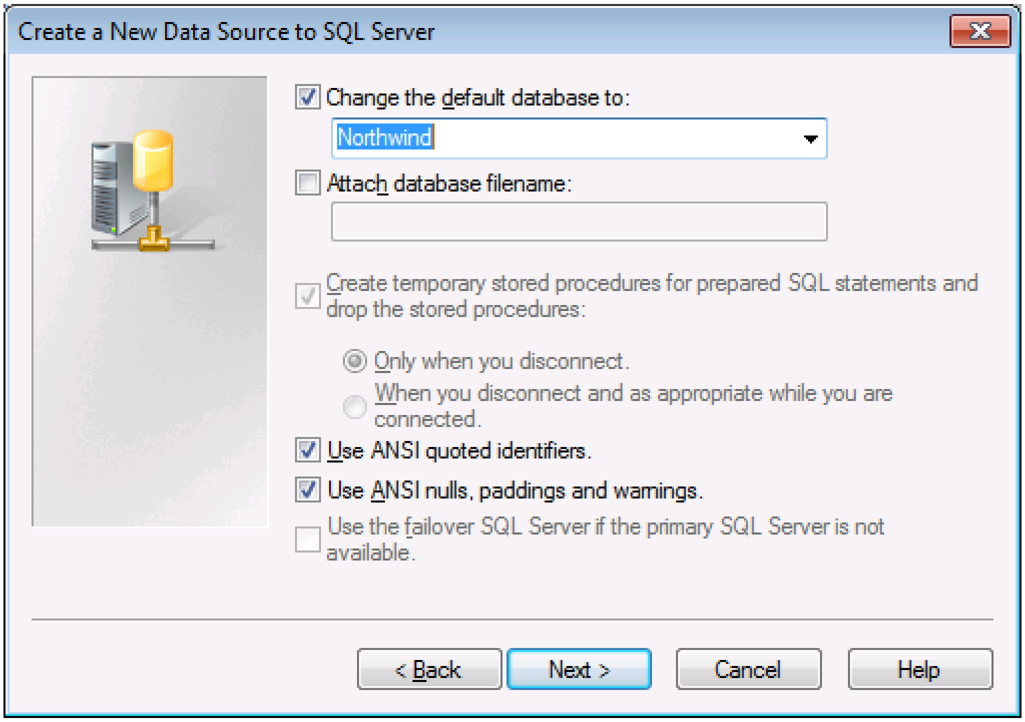
To change the network library used to communicate with SQL Server, click Client Configuration.

☑ Connect to SQL Server to obtain default settings for the additional configuration options.

Login ID: john_doe

Password: ••••••

6. If the database server was configured to allow connections from users authenticated on the Windows domain, select **With Windows NT authentication**. Otherwise, select **With SQL Server authentication...** and type the user name and password in the relevant boxes.



Create a New Data Source to SQL Server

☑ Change the default database to:

Northwind

☐ Attach database filename:

☑ Create temporary stored procedures for prepared SQL statements and drop the stored procedures:

● Only when you disconnect.

○ When you disconnect and as appropriate while you are connected.

☑ Use ANSI quoted identifiers.

☑ Use ANSI nulls, paddings and warnings.

☐ Use the failover SQL Server if the primary SQL Server is not available.

7. Select the name of the database to which you are connecting (in this example,

Northwind).

8. Click **Finish**.

Connecting to MySQL (ODBC)

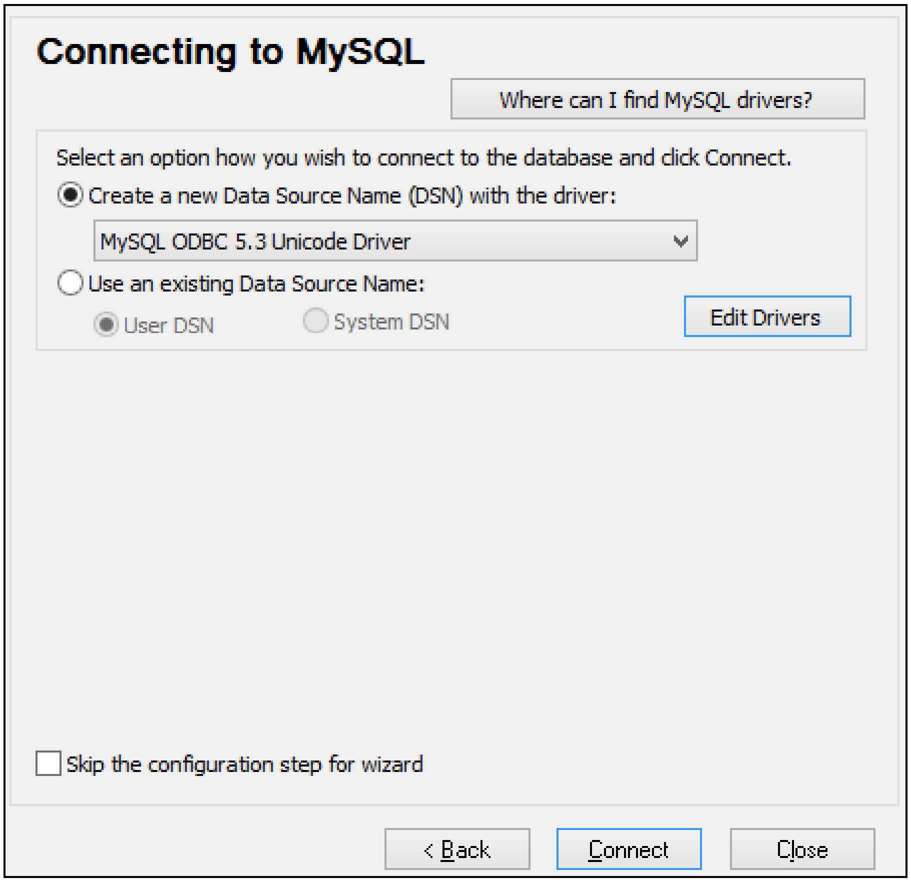
This topic provides sample instructions for connecting to a MySQL database server from a Windows machine through the ODBC driver. The MySQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses MySQL ODBC driver version 5.3.4 downloaded from the official website (see also Database Drivers Overview).

Prerequisites:

- MySQL ODBC driver must be installed on your operating system (for installation instructions, check the documentation supplied with the driver).
- You have the following database connection details: host, database, port, username, and password.

To connect to MySQL via ODBC:

1. Start the database connection wizard.
2. Select **MySQL (ODBC)**, and then click **Next**.



Connecting to MySQL

Select an option how you wish to connect to the database and click Connect.

● Create a new Data Source Name (DSN) with the driver:

MySQL ODBC 5.3 Unicode Driver

○ Use an existing Data Source Name:

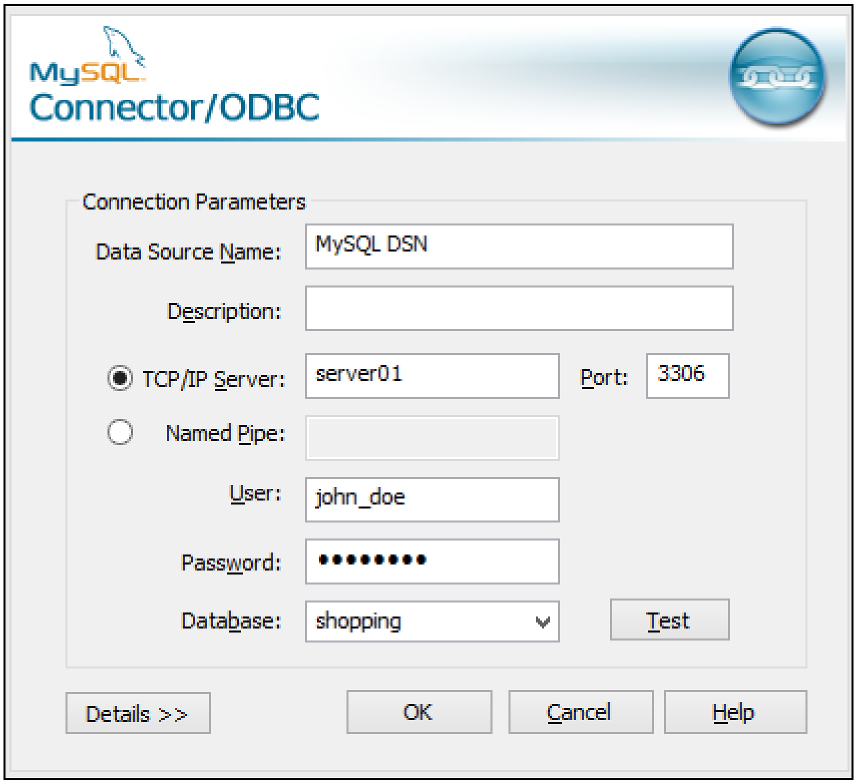
● User DSN ○ System DSN

□ Skip the configuration step for wizard

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).

4. Click **Connect**.



Connection Parameters

Data Source Name: MySQL DSN

Description:

● TCP/IP Server: server01 Port: 3306

○ Named Pipe:

User: john_doe

Password: •••••••••

Database: shopping

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.

Connecting to 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 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.

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)
    )
  )
```

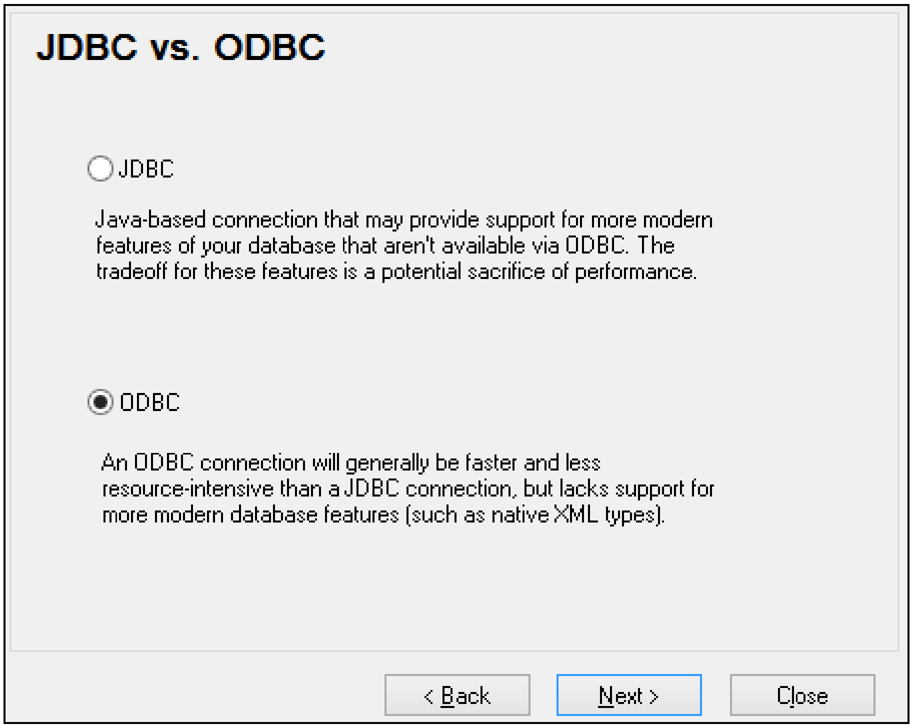
The path to the **tnsnames.ora** file depends on the location where Oracle home directory was installed. For Oracle database client 11.2.0, the default Oracle home directory path could be as follows:

```
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).

To connect to Oracle using ODBC:

1. Start the database connection wizard.
2. Select **Oracle (ODBC / JDBC)**, and then click **Next**.



JDBC vs. ODBC

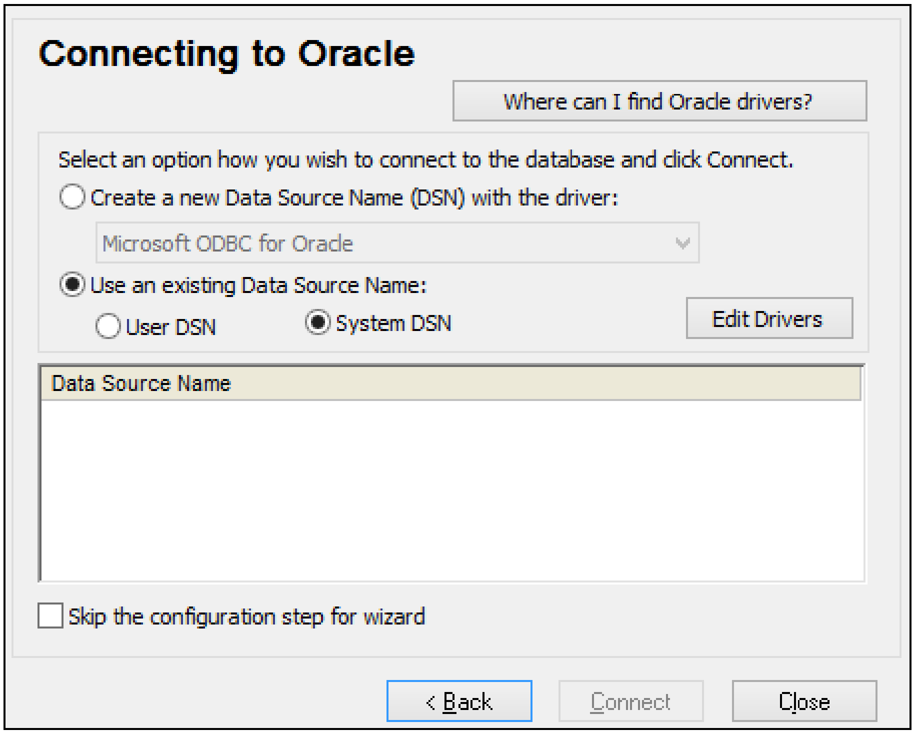
○ JDBC

Java-based connection that may provide support for more modern features of your database that aren't available via ODBC. The tradeoff for these features is a potential sacrifice of performance.

● ODBC

An ODBC connection will generally be faster and less resource-intensive than a JDBC connection, but lacks support for more modern database features (such as native XML types).

3. Select **ODBC**.



Connecting to Oracle

Select an option how you wish to connect to the database and click Connect.

○ Create a new Data Source Name (DSN) with the driver:

Microsoft ODBC for Oracle

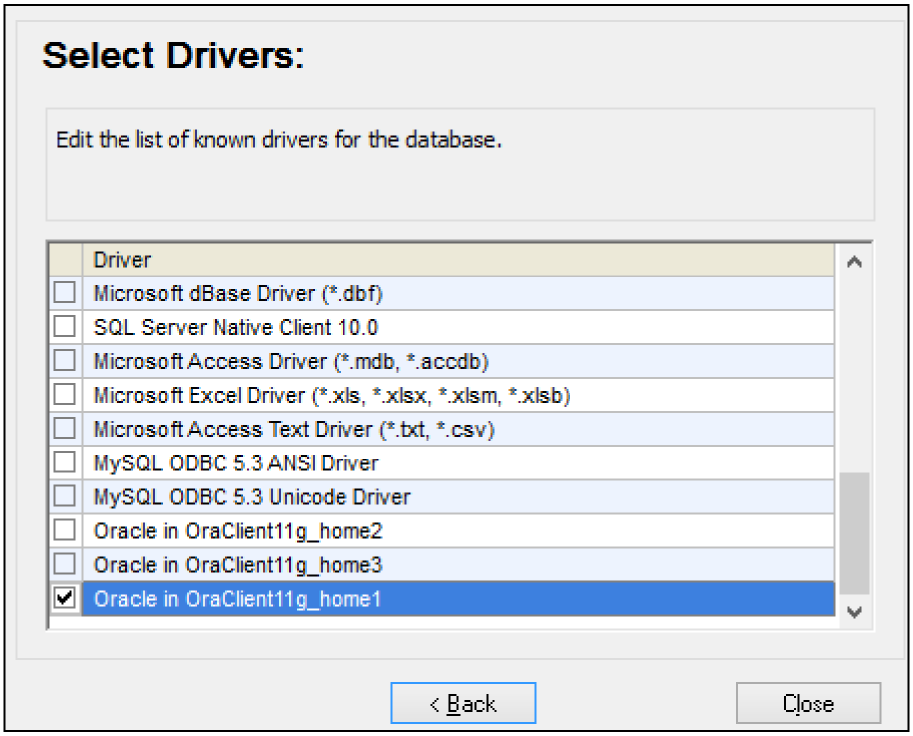
● Use an existing Data Source Name:

○ User DSN ● System DSN

Data Source Name

□ Skip the configuration step for wizard

4. Click **Edit Drivers**.

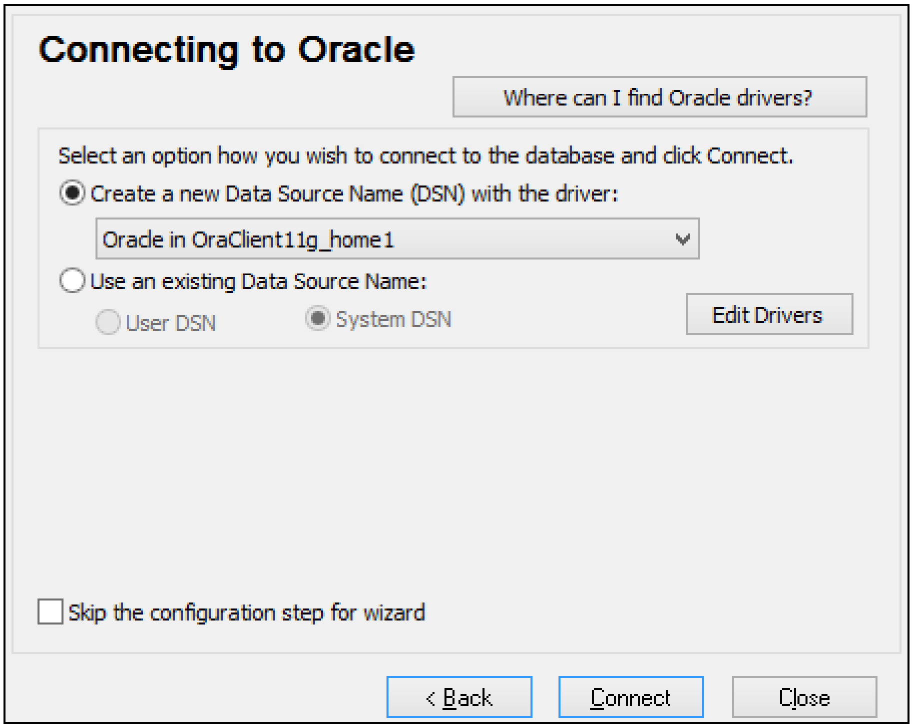


Select Drivers:

Edit the list of known drivers for the database.

	Driver
□	Microsoft dBase Driver (*.dbf)
□	SQL Server Native Client 10.0
□	Microsoft Access Driver (*.mdb, *.accdb)
□	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)
□	Microsoft Access Text Driver (*.txt, *.csv)
□	MySQL ODBC 5.3 ANSI Driver
□	MySQL ODBC 5.3 Unicode Driver
□	Oracle in OraClient11g_home2
□	Oracle in OraClient11g_home3
☑	Oracle in OraClient11g_home1

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.



Connecting to Oracle

Select an option how you wish to connect to the database and click Connect.

● Create a new Data Source Name (DSN) with the driver:

Oracle in OraClient11g_home1

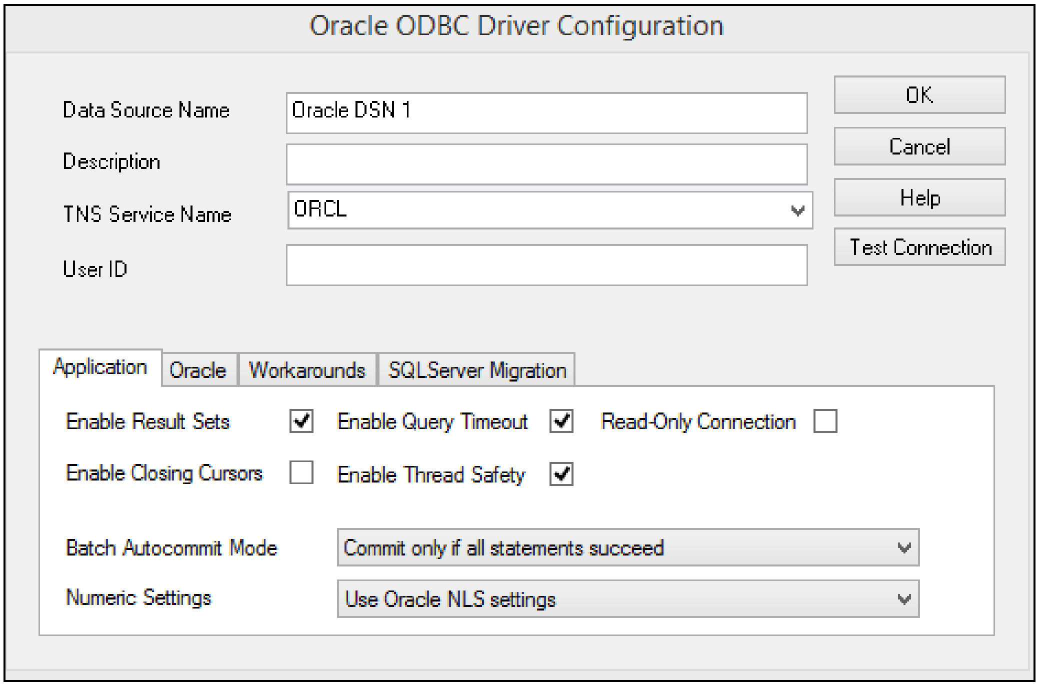
○ Use an existing Data Source Name:

○ User DSN ● System DSN

□ Skip the configuration step for wizard

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**.



Oracle ODBC Driver Configuration

Data Source Name: Oracle DSN 1

Description:

TNS Service Name: ORCL

User ID:

Application | Oracle | Workarounds | SQLServer Migration

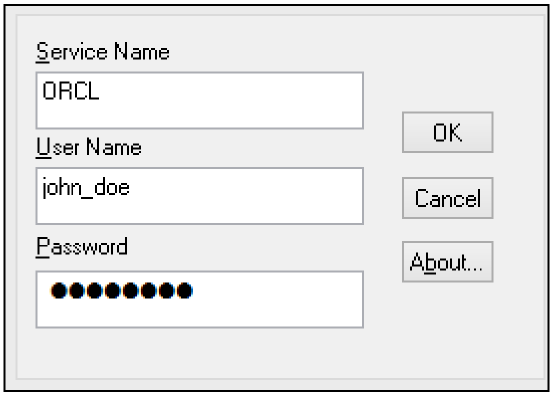
Enable Result Sets ☑ Enable Query Timeout ☑ Read-Only Connection □

Enable Closing Cursors □ Enable Thread Safety ☑

Batch Autocommit Mode: Commit only if all statements succeed

Numeric Settings: Use Oracle NLS settings

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). In this example, the connection name is **ORCL**.
11. Click **OK**.



Service Name: ORCL

User Name: john_doe

Password: ●●●●●●●●

12. Enter the username and password to the database, and then click OK.

Connecting to Oracle (JDBC)

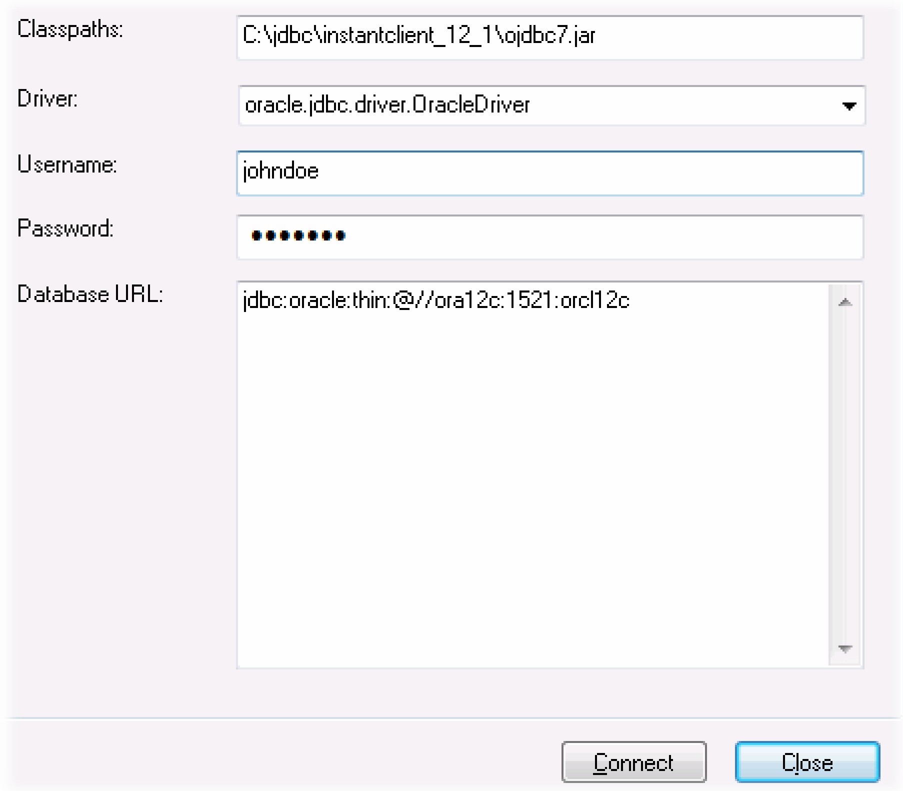
This example shows you how to connect to an Oracle database server from a client machine, using the JDBC interface. The connection is created as a pure Java connection, using the **Oracle Instant Client Package (Basic)** available from the Oracle website. The advantage of this connection type is that 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:

- Java Runtime Environment (JRE) or Java Development Kit (JDK) must be installed on your operating system.
- 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 **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.
- 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.
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\odbc7.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).
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 "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.



Classpaths: C:\jdbc\instantclient_12_1\ojdbc7.jar

Driver: oracle.jdbc.driver.OracleDriver

Username: johndoe

Password: •••••••

Database URL: jdbc:oracle:thin:@//ora12c:1521:orcl12c

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**.

Connecting to PostgreSQL (ODBC)

This topic provides sample instructions for connecting 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 09_03_300-1) downloaded from the official website (see also Database Drivers Overview).

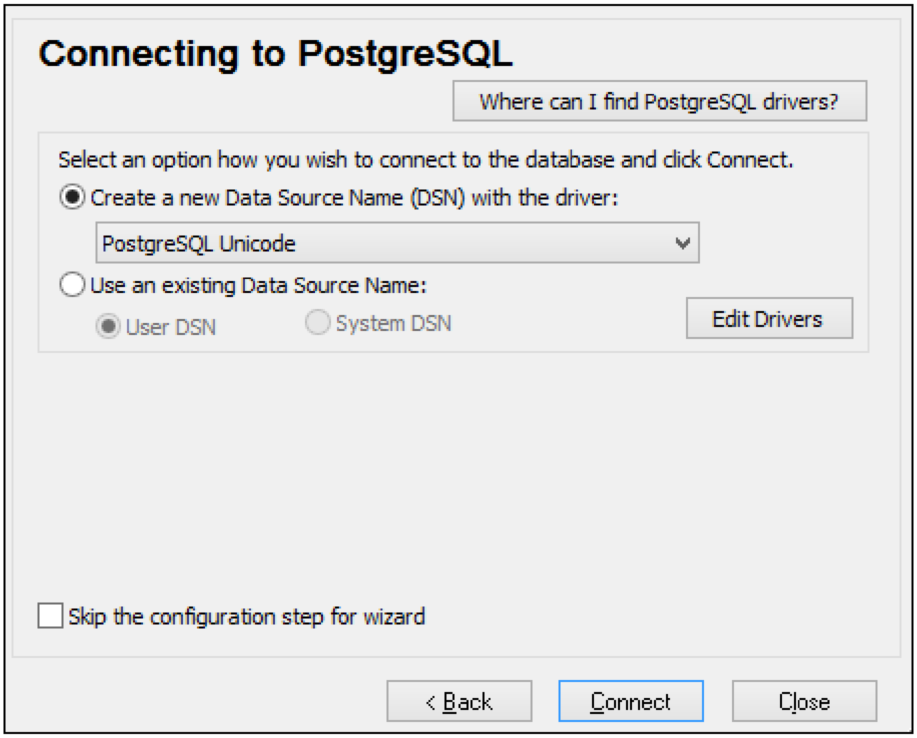
Note: You can also connect to a PostgreSQL database server directly (without the ODBC driver), see Setting up a PostgreSQL Connection.

Prerequisites:

- *psqlODBC* driver must be installed on your operating system (for installation instructions, check the documentation supplied with the driver).
- You have the following database connection details: server, port, database, user name, and password.

To connect to PostgreSQL using ODBC:

1. Start the database connection wizard.
2. Select **PostgreSQL (ODBC)**, and then click **Next**.



Connecting to PostgreSQL

Select an option how you wish to connect to the database and click Connect.

● Create a new Data Source Name (DSN) with the driver:

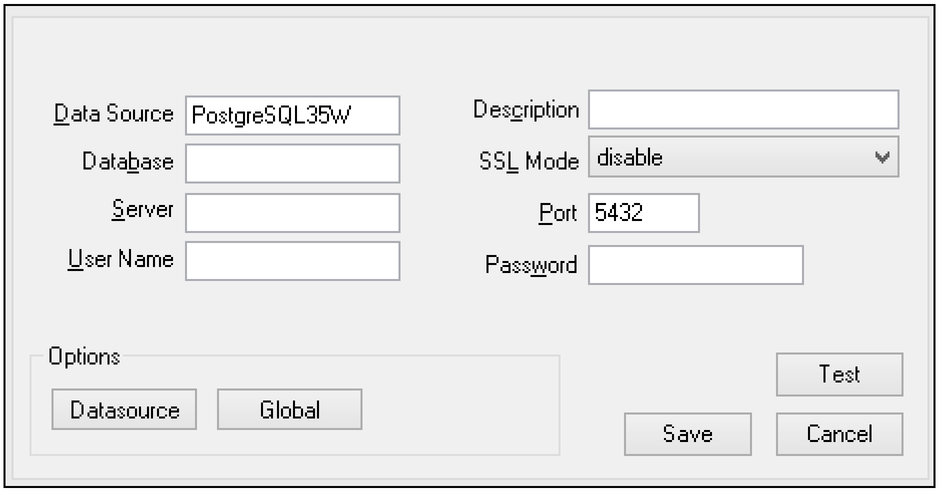
PostgreSQL Unicode

○ Use an existing Data Source Name:

● User DSN ○ System DSN

□ Skip the configuration step for wizard

3. Select **Create a new Data Source Name (DSN) with the driver**, and select the PostgreSQL driver. If no PostgreSQL driver is available in the list, click **Edit Drivers**, and select any available PostgreSQL drivers (the list contains all ODBC drivers installed on your operating system).
4. Click **Connect**.



Data Source: PostgreSQL35W

Database:

Server:

User Name:

Description:

SSL Mode: disable

Port: 5432

Password:

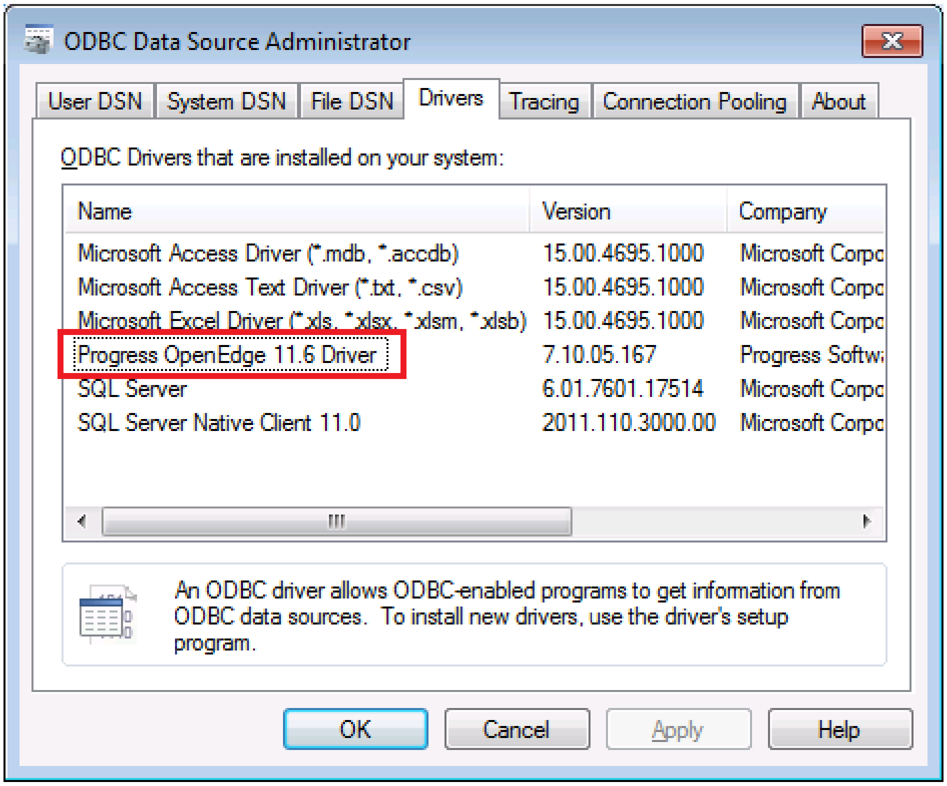
5. Fill in the database connection credentials (Database, Server, Port, User Name, Password), and then click **OK**.

Connecting to Progress OpenEdge (ODBC)

This topic provides sample instructions for connecting 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). 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).



ODBC Drivers that are installed on your system:

Name	Version	Company
Microsoft Access Driver (*.mdb, *.accdb)	15.00.4695.1000	Microsoft Corpc
Microsoft Access Text Driver (*.txt, *.csv)	15.00.4695.1000	Microsoft Corpc
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)	15.00.4695.1000	Microsoft Corpc
Progress OpenEdge 11.6 Driver	7.10.05.167	Progress Softw
SQL Server	6.01.7601.17514	Microsoft Corpc
SQL Server Native Client 11.0	2011.110.3000.00	Microsoft Corpc

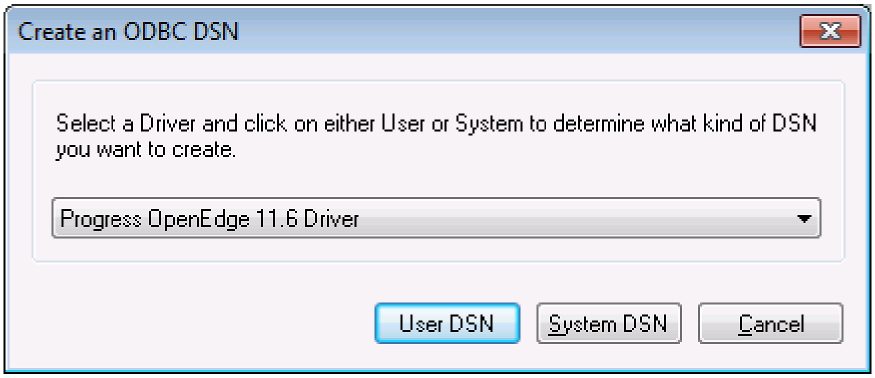
An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

- 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.
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** ✚ .

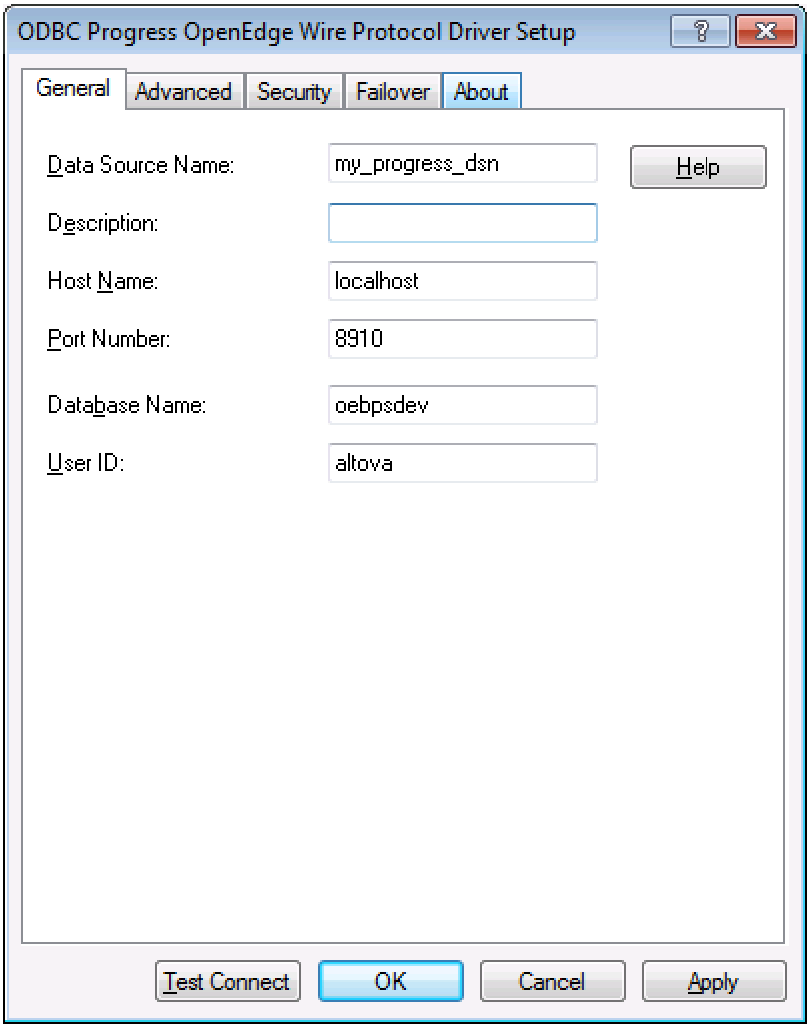
5. Select the **Progress OpenEdge Driver** from the list, and click **User DSN** (or **System DSN**, if applicable).



Select a Driver and click on either User or System to determine what kind of DSN you want to create.

Progress OpenEdge 11.6 Driver

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**.



Data Source Name: my_progress_dsn

Description:

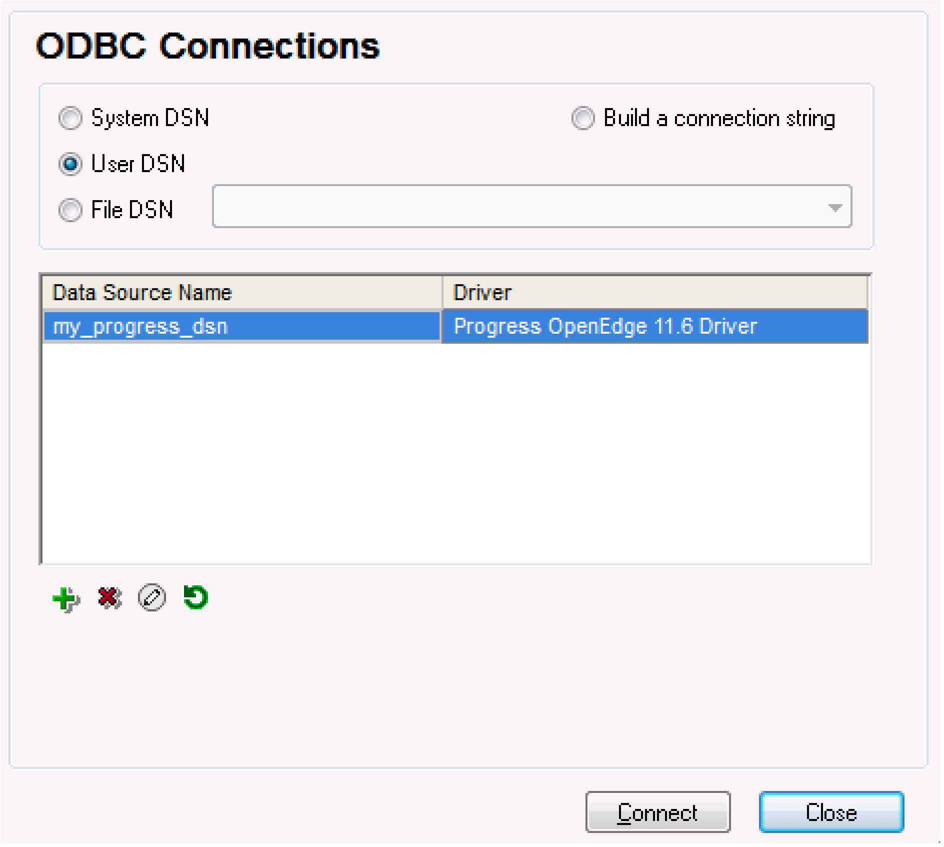
Host Name: localhost

Port Number: 8910

Database Name: oebpsdev

User ID: altova

7. Click OK. The new data source now appears in the list of ODBC data sources.



Data Source Name	Driver
my_progress_dsn	Progress OpenEdge 11.6 Driver

8. Click **Connect**.

Connecting to Progress OpenEdge (JDBC)

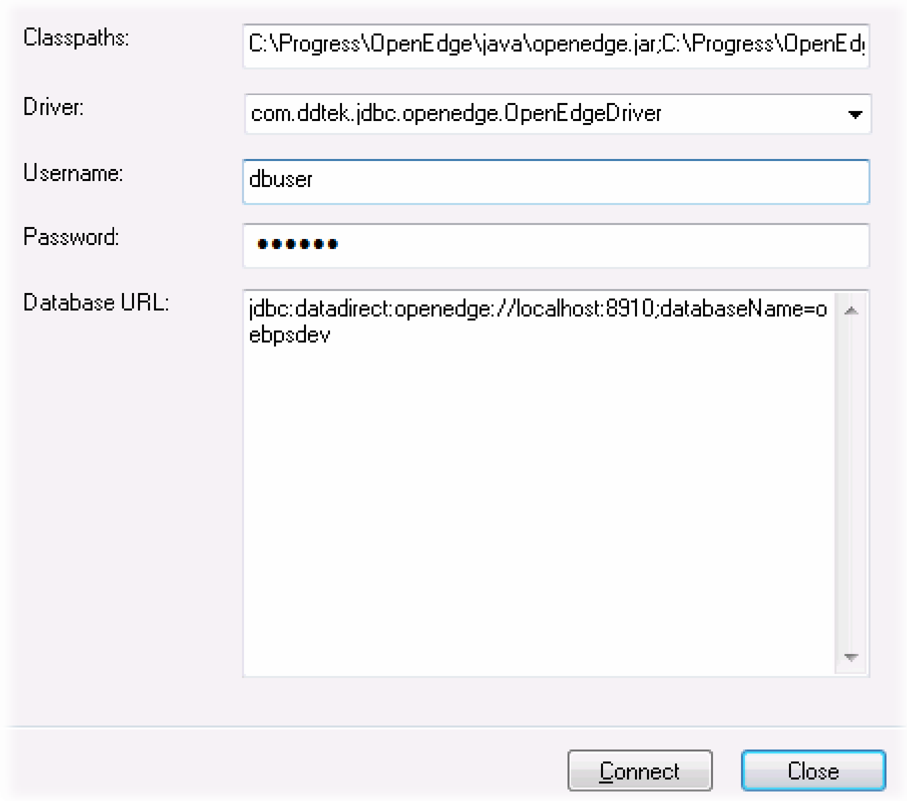
This topic provides sample instructions for connecting to a Progress OpenEdge 11.6 database server through JDBC.

Prerequisites

- Java Runtime Environment (JRE) or Java Development Kit (JDK) must be installed on your operating system. 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.
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).
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 "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).


Classpaths: C:\Progress\OpenEdge\java\openedge.jar;C:\Progress\OpenEd

Driver: com.ddtek.jdbc.openedge.OpenEdgeDriver

Username: dbuser

Password: ••••••

Database URL: jdbc:datadirect:openedge://localhost:8910;databaseName=oebpsdev

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**.

Connecting to Sybase (JDBC)

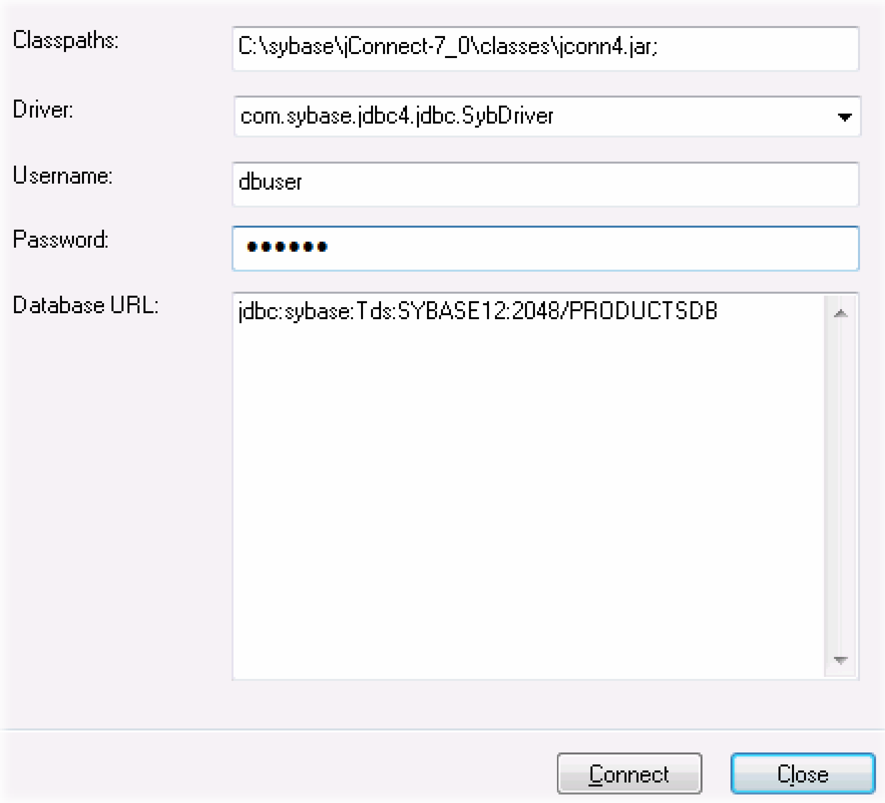
This topic provides sample instructions for connecting to a Sybase database server through JDBC.

Prerequisites:

- Java Runtime Environment (JRE) must be installed on your operating system.
- 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.
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).
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 "Classpath" text box, or in the operating system's CLASSPATH environment variable (see the previous step).



Classpaths: C:\sybase\jConnect-7_0\classes\jconn4.jar;

Driver: com.sybase.jdbc4.jdbc.SybDriver

Username: dbuser

Password: ••••••

Database URL: jdbc:sybase:Tds:SYBASE12:2048/PRODUCTSDB

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**.

Connecting to Teradata (ODBC)

This example illustrates how to connect to a Teradata database server through ODBC.

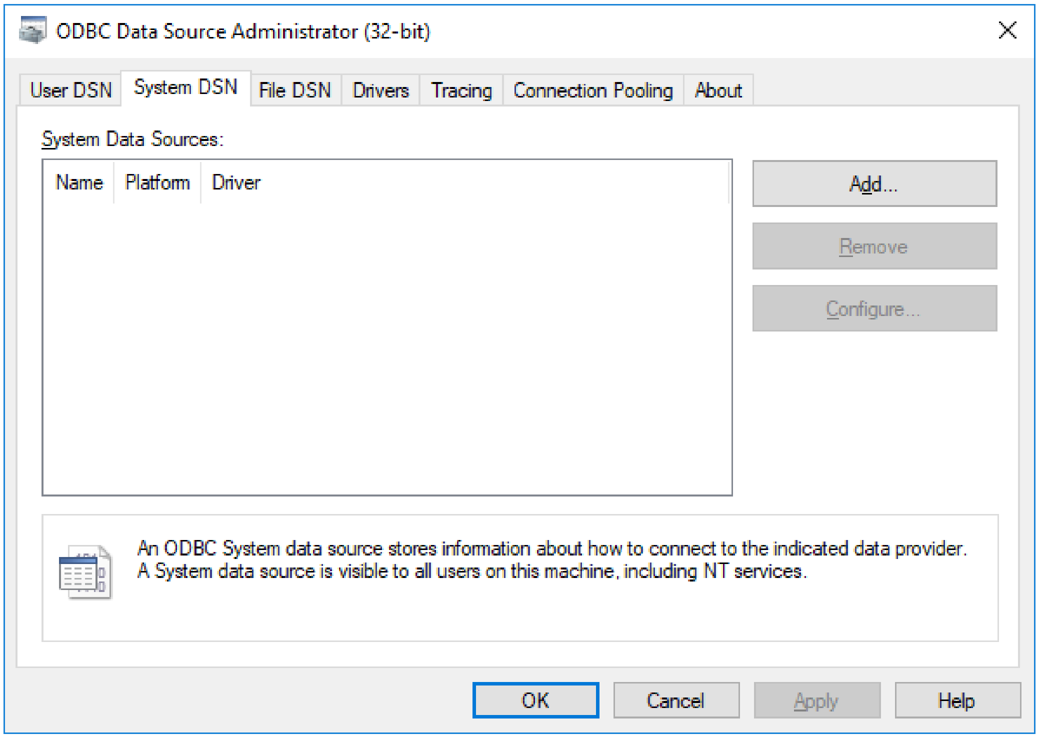
Prerequisites:

- 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:

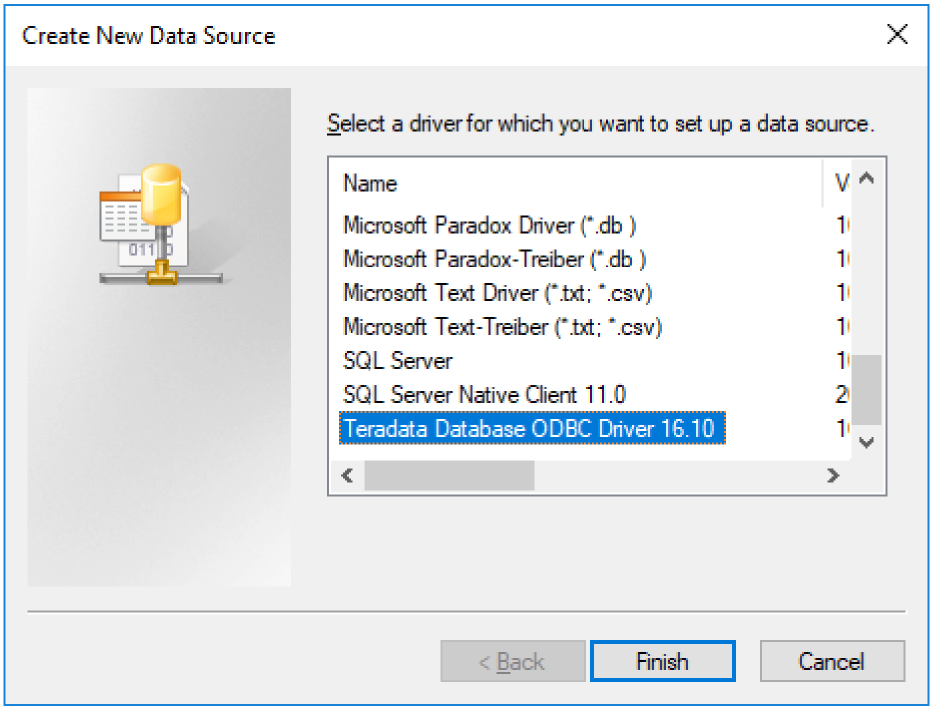
1. Press the **Windows** key, start typing "ODBC", and select **Set up ODBC data sources (32-bit)** from the list of suggestions. If you have a 64-bit ODBC driver, select **Set up**

ODBC data sources (64-bit) and use 64-bit MapForce in the subsequent steps.



An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

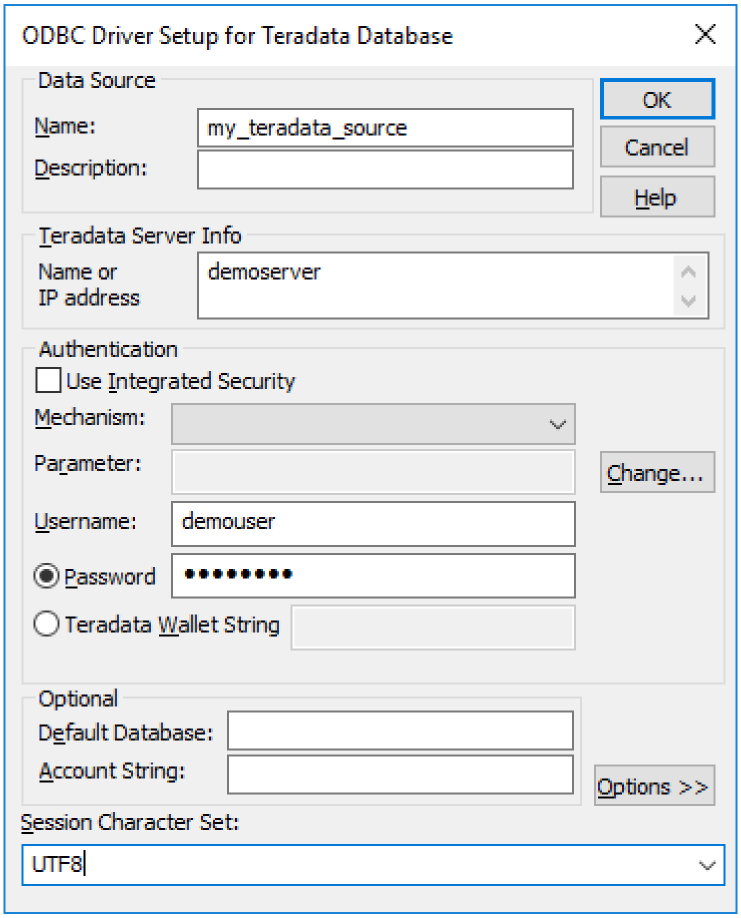
2. Click the **System DSN** tab, and then click **Add**.



Select a driver for which you want to set up a data source.

Name	V
Microsoft Paradox Driver (*.db)	1
Microsoft Paradox-Treiber (*.db)	1
Microsoft Text Driver (*.txt; *.csv)	1
Microsoft Text-Treiber (*.txt; *.csv)	1
SQL Server	1
SQL Server Native Client 11.0	2
Teradata Database ODBC Driver 16.10	1

3. Select **Teradata Database ODBC Driver** and click **Finish**.



ODBC Driver Setup for Teradata Database

Data Source

Name: my_teradata_source

Description:

Teradata Server Info

Name or IP address: demoserver

Authentication

□ Use Integrated Security

Mechanism:

Parameter:

Username: demouser

● Password •••••••••

○ Teradata Wallet String

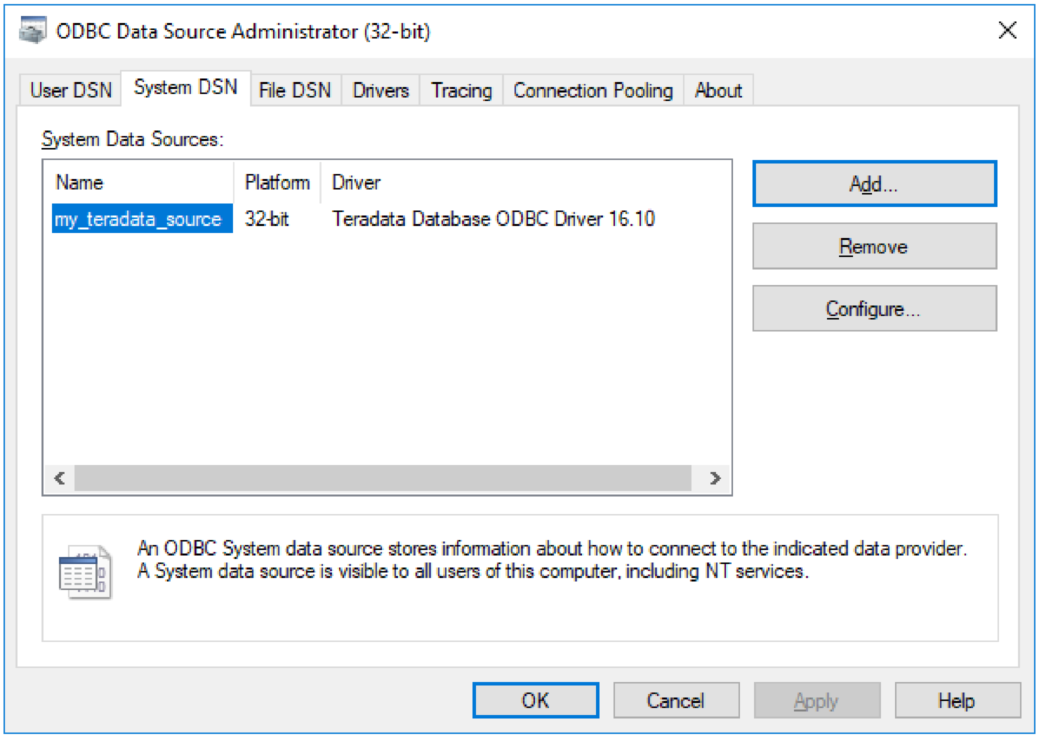
Optional

Default Database:

Account String:

Session Character Set: UTF8

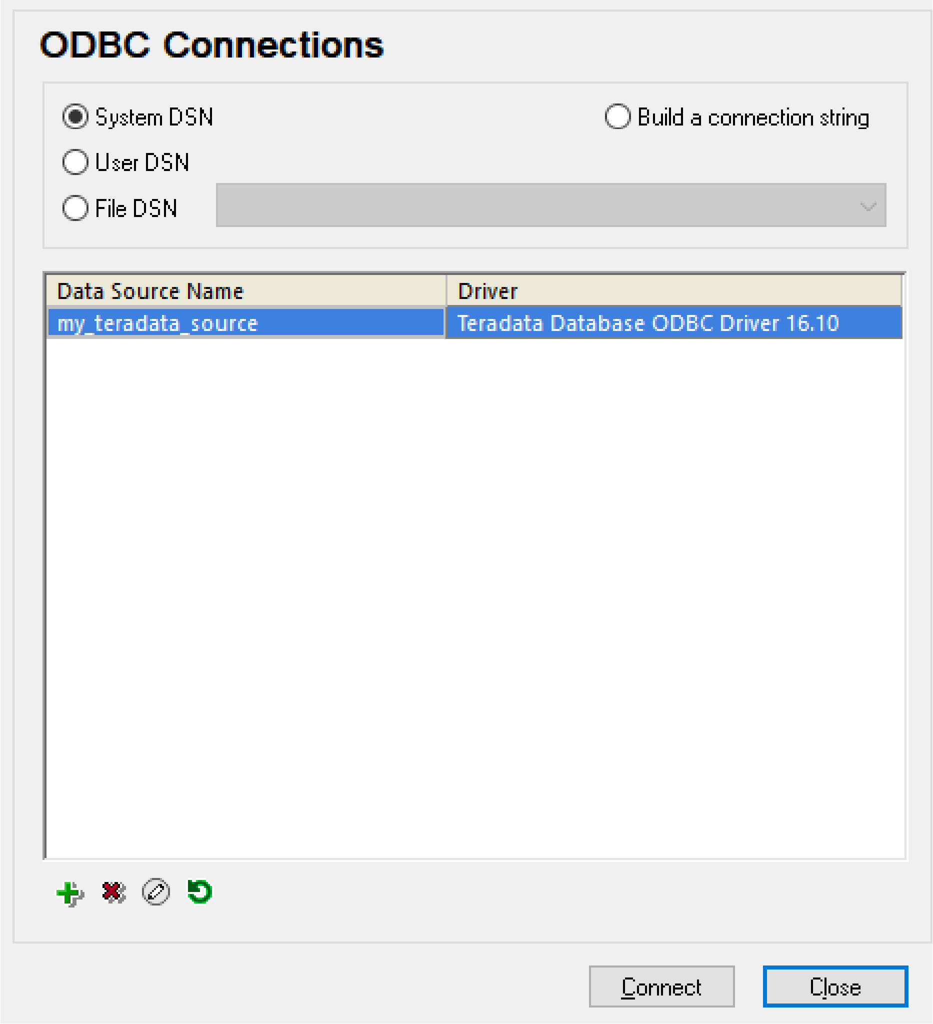
4. Enter name and, optionally, a description that will help you identify this ODBC data source in future. Also, enter the database connection credentials (Database server, User, Password), and, optionally, select a database.
5. Click **OK**. The data source now appears in the list.



Name	Platform	Driver
my_teradata_source	32-bit	Teradata Database ODBC Driver 16.10

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users of this computer, including NT services.

6. Run MapForce and start the database connection wizard.
7. Click **ODBC Connections**.



Data Source Name	Driver
my_teradata_source	Teradata Database ODBC Driver 16.10

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.

Connecting to Teradata (JDBC)

This example illustrates how to connect to a Teradata database server through JDBC.

Prerequisites:

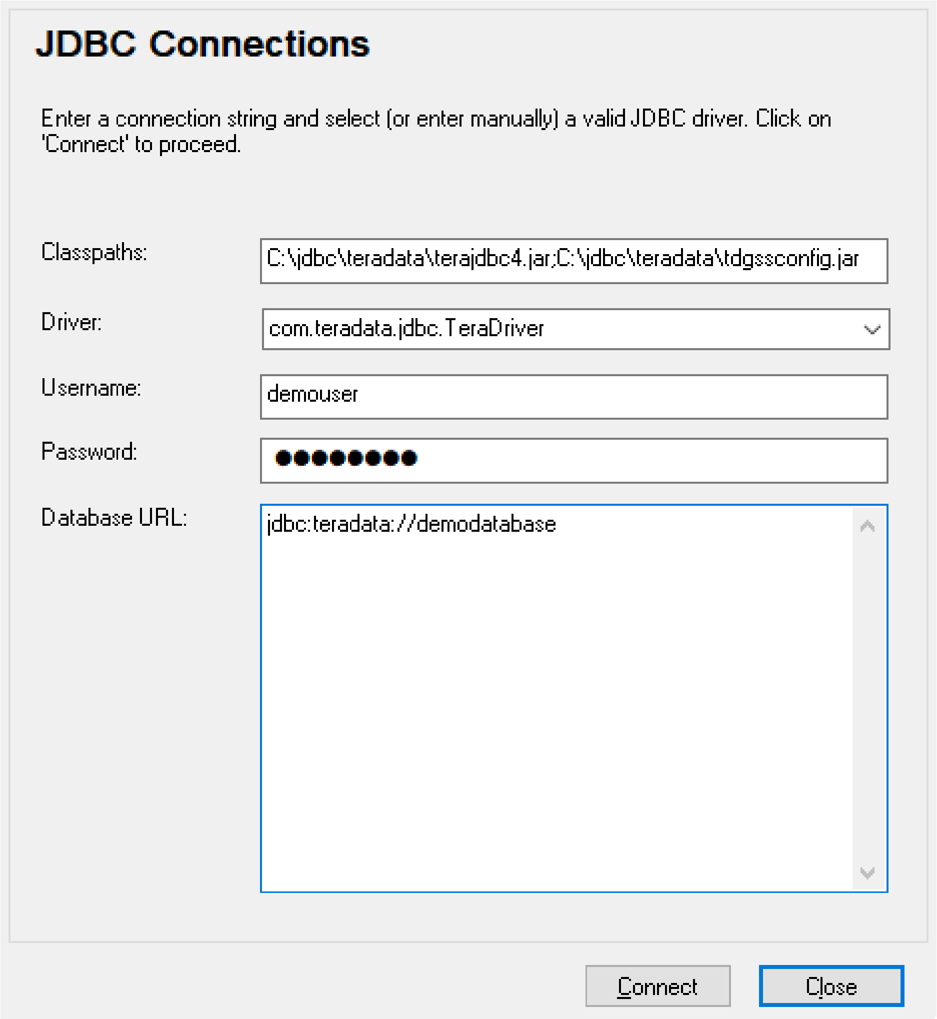
- Java Runtime Environment (JRE) or Java Development Kit (JDK) must be installed on your operating system.
- The JDBC driver (one or more .jar files that provide connectivity to the database) must be available on your operating system. In this example, Teradata JDBC Driver 16.20.00.02 is

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.
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).
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).



JDBC Connections

Enter a connection string and select (or enter manually) a valid JDBC driver. Click on 'Connect' to proceed.

Classpaths: C:\jdbc\teradata\terajdbc4.jar;C:\jdbc\teradata\tdgssconfig.jar

Driver: com.teradata.jdbc.TeraDriver

Username: demouser

Password: ●●●●●●●●

Database URL: jdbc:teradata://demodatabase

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**.

7.2.1.11 *Database Connections on Linux and macOS*

If you have licensed any of the following Altova server products—MobileTogether Server, MapForce Server, or StyleVision Server, a common scenario is to design MobileTogether designs, MapForce mappings, or StyleVision transformations on a Windows desktop machine, and then deploy them to a server machine (either Windows, Linux, or OS X / macOS) to automate their execution.

In this documentation, the term "server execution files" is used to denote the following file types:

- MapForce Server execution files (.mfx)
- MobileTogether design files (.mtd)
- StyleVision transformations (.sps) packaged as Portable XML Forms (.pxf).

The following scenarios are possible when deploying server execution files:

1. **"Design and execute on Windows"**. In this scenario, you design the MobileTogether designs, MapForce mappings, or StyleVision transformations on Windows, and then run their corresponding server execution files on a Windows system as well (which can either be the same Windows machine, or a remote Windows server).
2. **"Design on Windows, execute on Linux or macOS"**. In this scenario, you design all of the above files on Windows, and then deploy their corresponding server execution files to Linux or OS X / macOS for execution.

In the **"Design and execute on Windows"** scenario, the selection of available database technologies and drivers comprises any of ADO, ODBC, JDBC, as well as SQLite connections (see Database Drivers Overview).

In the **"Design on Windows, execute on Linux or macOS"** scenario, ADO and ODBC connections are not supported. In this scenario, you can use direct SQLite connections (see SQLite connections) and JDBC connections (see JDBC connections).

When you deploy server execution files to a server, databases are not included in the deployed package (this also applies to file-based databases such as SQLite and Microsoft Access), so a connection to them must be set up on the deployment server as well. In other words, the same database configuration must be in place both on the operating system where you design and on the server to which you deploy the files. An exception to this rule are native (not driver-based) PostgreSQL connections. Native PostgreSQL connections do not require configuration outside MapForce. For more information, see Setting up a PostgreSQL Connection.

In general, the scenario in which you deploy server execution files to a different operating system is slightly more complex, since it requires that the same database configuration exist on both machines. To bypass complexity while designing locally and deploying remotely, consider using the Global Resources feature available in MapForce, MobileTogether Designer, and StyleVision.

For example, you can define two different Global Resource configurations to connect to the same database: one which would specify the connection settings using the Windows-style path conventions, and another one—using Linux-style path conventions. You could then use the first connection to test your files during the design phase, and the second connection to run the execution file on the Linux server.

SQLite connections on Linux and macOS

There is no need to separately install SQLite on Linux and macOS since support for it is integrated into Altova server products as well. Therefore, if your server execution files include calls to a SQLite database, you will be able to run them without having to install SQLite first. You need to ensure, however, that the server execution files use the correct path to the database file on the Linux or OS X / macOS machine. That is, before running the server execution files on the Linux or OS X / macOS server, make sure that the SQLite database file is referenced through a path which is POSIX (Portable Operating System Interface) compliant. This assumes that no Windows-style drive letters are used in the path, and directories are delimited by the forward slash character (/). For example, the path **/usr/local/mydatabase.db** is POSIX compliant, while the path **c:\sqlite \mydatabase.db** isn't.

JDBC connections on Linux and macOS

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 Connections on OS X Yosemite

On OS X Yosemite, you can connect to an Oracle database through the **Oracle Database Instant Client**. Note that, if you have a Mac with a Java version prior to Java 8, you can also connect through the **JDBC Thin for All Platforms** library, in which case you may disregard the instructions in this topic.

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.

7.2.2 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). 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.

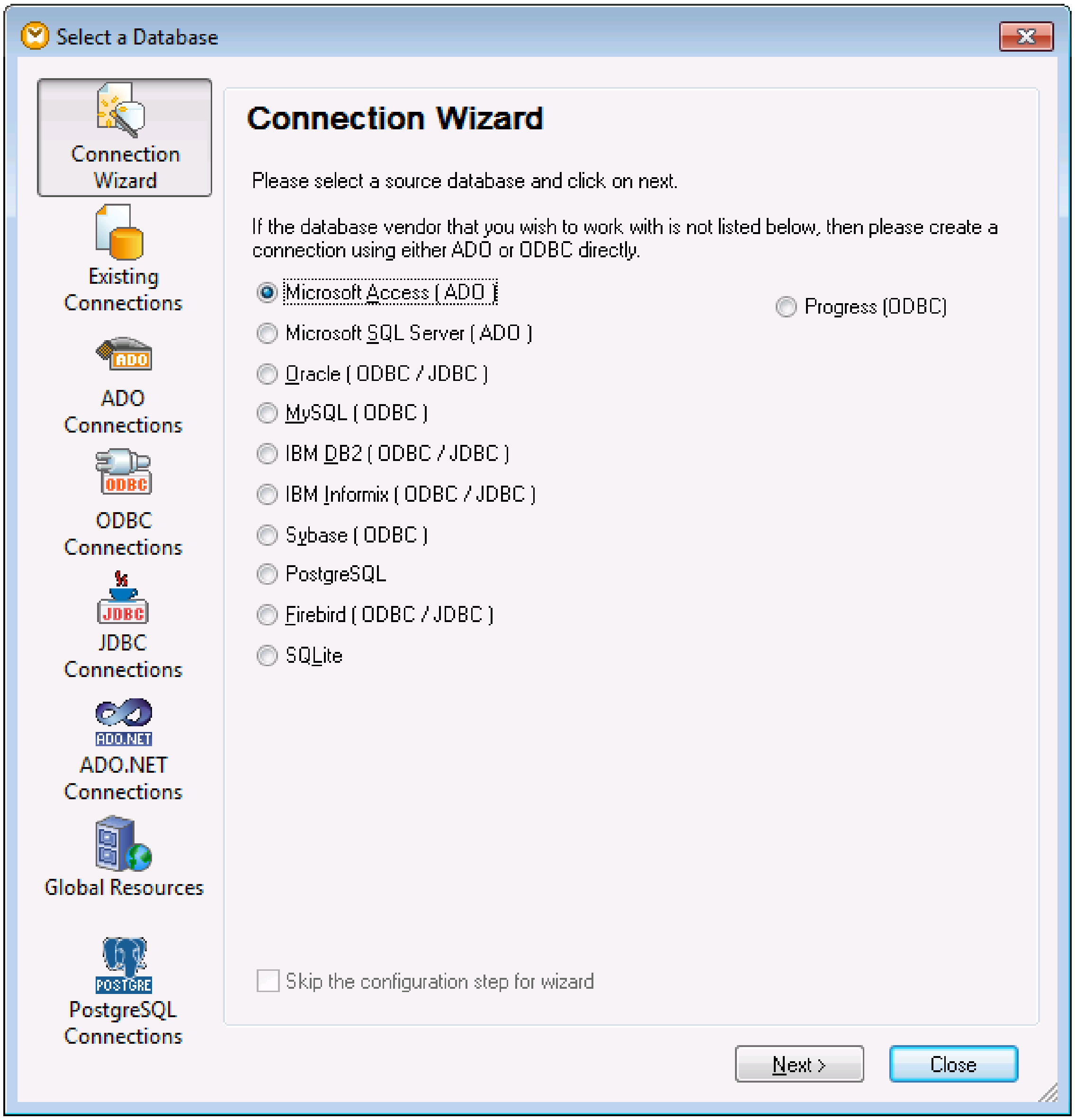
7.2.2.1 Adding Databases to the Mapping

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 Selecting a Transformation Language). 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.



Select a Database

Connection Wizard

Please select a source database and click on next.

If the database vendor that you wish to work with is not listed below, then please create a connection using either ADO or ODBC directly.

● Microsoft Access (ADO)
○ Microsoft SQL Server (ADO)
○ Oracle (ODBC / JDBC)
○ MySQL (ODBC)
○ IBM DB2 (ODBC / JDBC)
○ IBM Informix (ODBC / JDBC)
○ Sybase (ODBC)
○ PostgreSQL
○ Firebird (ODBC / JDBC)
○ SQLite
○ Progress (ODBC)

□ Skip the configuration step for wizard

Database Connection Wizard

Note: In some advanced scenarios, databases can also be added to the mapping as variables (see Using Variables). When you choose to add a database structure as a variable, the same database connection wizard mentioned above appears.

For instructions on how to proceed with this wizard so as to set up a connection to any of the databases supported by MapForce, see Connecting to a Database and, in particular, the step-by-step examples, including:

- Connecting to Firebird (ODBC)
- Connecting to IBM DB2 (ODBC)
- Connecting to an Existing Microsoft Access Database
- Connecting to Microsoft SQL Server (ADO)

- Connecting to MySQL (ODBC)
- Connecting to Oracle (ODBC)
- Connecting to PostgreSQL (ODBC)
- Connecting to Progress OpenEdge (JDBC)
- Connecting to an Existing SQLite Database
- Connecting to Sybase (JDBC)

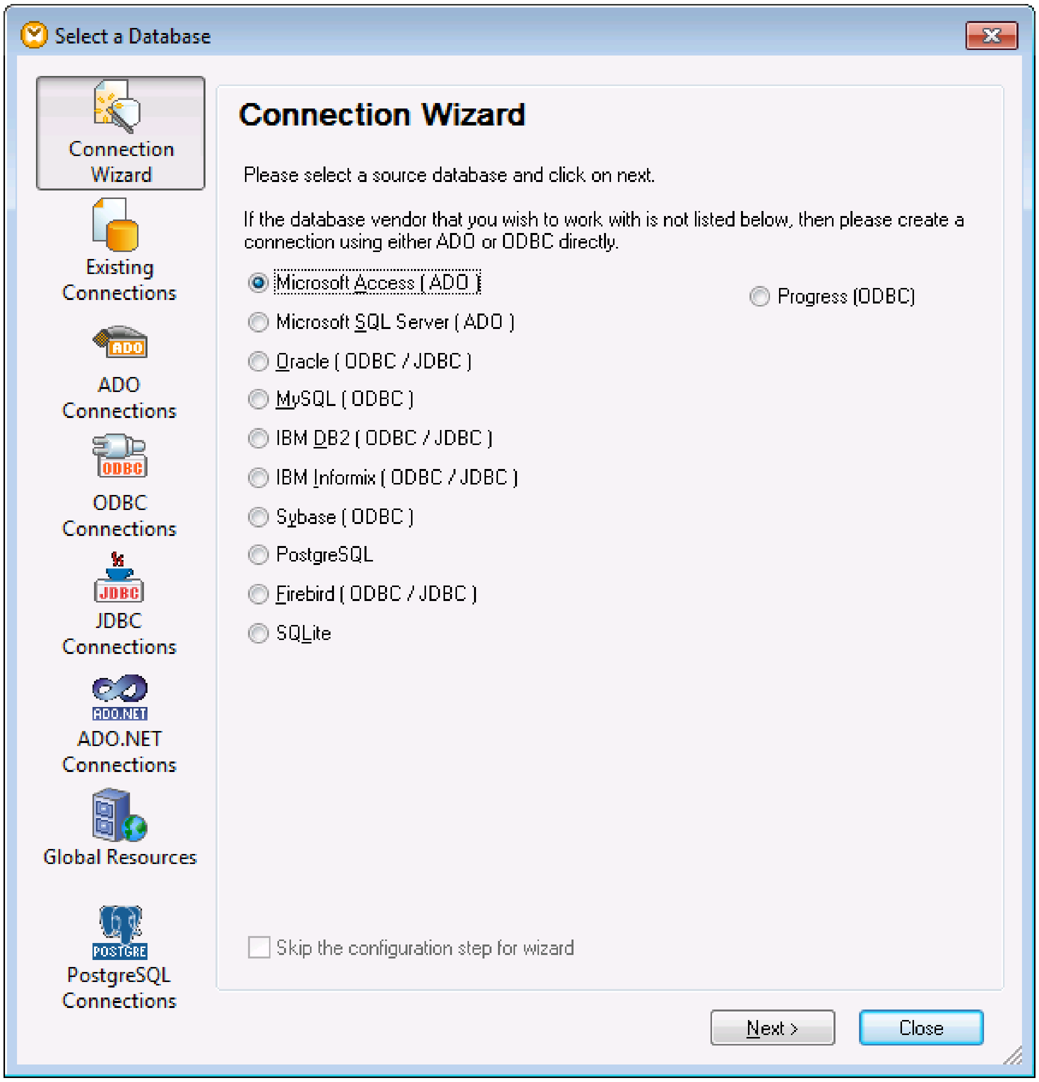
Once the database connection is successfully established, you are prompted to select the database objects that should appear on the mapping. See Adding, Editing, and Removing Database Objects for further information.

7.2.2.2 *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 \MapForce2018\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.



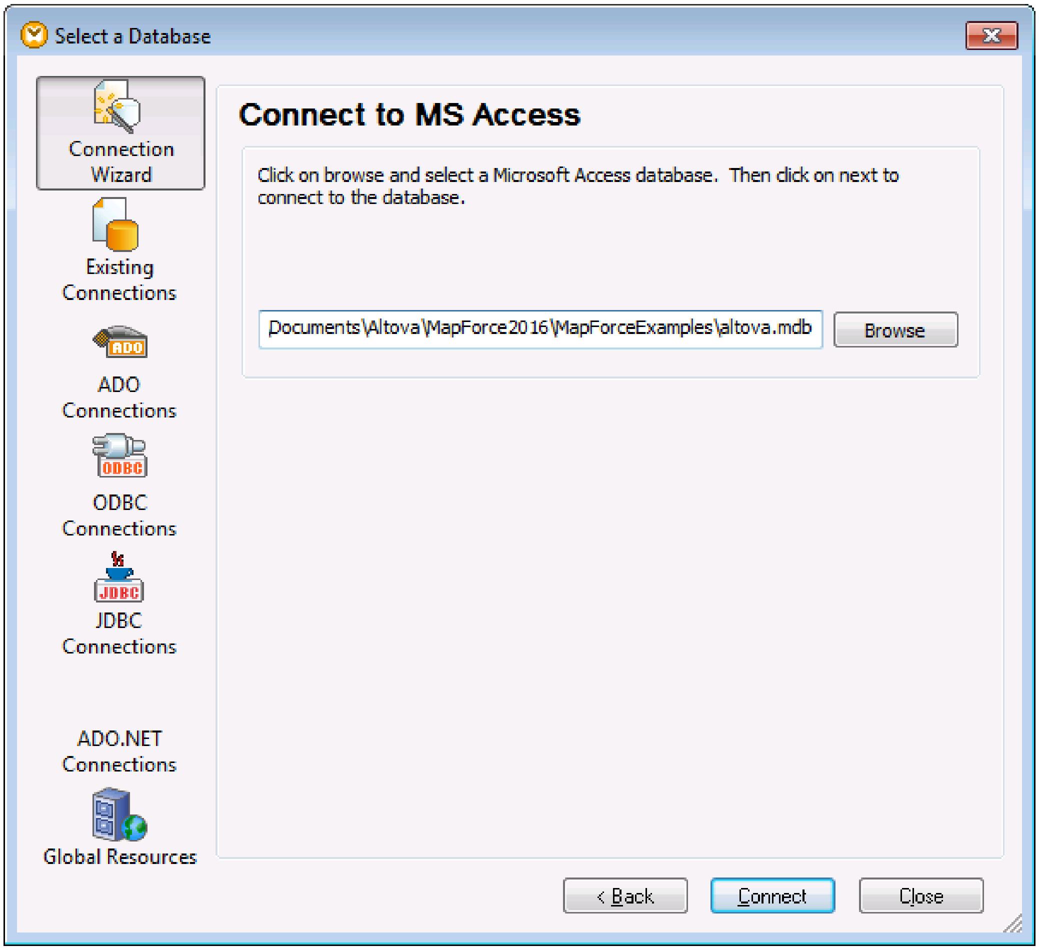
Please select a source database and click on next.

If the database vendor that you wish to work with is not listed below, then please create a connection using either ADO or ODBC directly.

● Microsoft Access (ADO)
○ Microsoft SQL Server (ADO)
○ Oracle (ODBC / JDBC)
○ MySQL (ODBC)
○ IBM DB2 (ODBC / JDBC)
○ IBM Informix (ODBC / JDBC)
○ Sybase (ODBC)
○ PostgreSQL
○ Firebird (ODBC / JDBC)
○ SQLite
○ Progress (ODBC)

□ Skip the configuration step for wizard

2. Click **Microsoft Access (ADO)**, and then click **Next**.

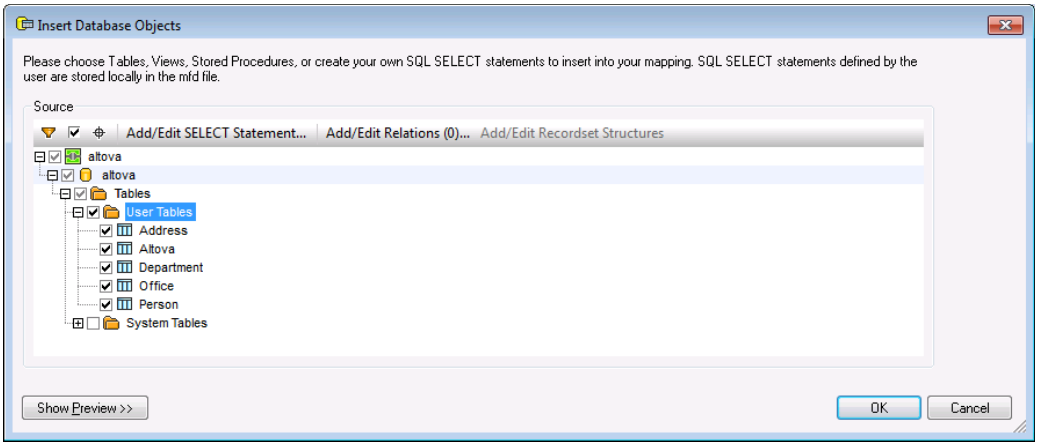


Connect to MS Access

Click on browse and select a Microsoft Access database. Then click on next to connect to the database.

Documents\Altova\MapForce2016\MapForceExamples\altova.mdb

3. Browse for the **altova.mdb** file available in the **<Documents>\Altova\MapForce2018\MapForceExamples\** folder, and then click **Connect**.
4. When prompted to select the database objects, select **User Tables**.



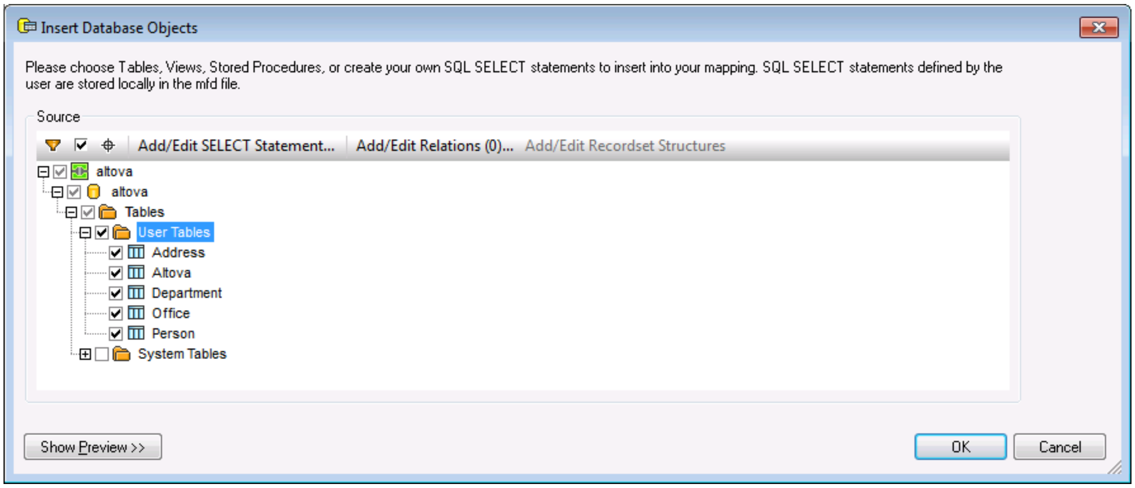
Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

7.2.2.3 *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 \MapForce2018\MapForceExamples\** folder.

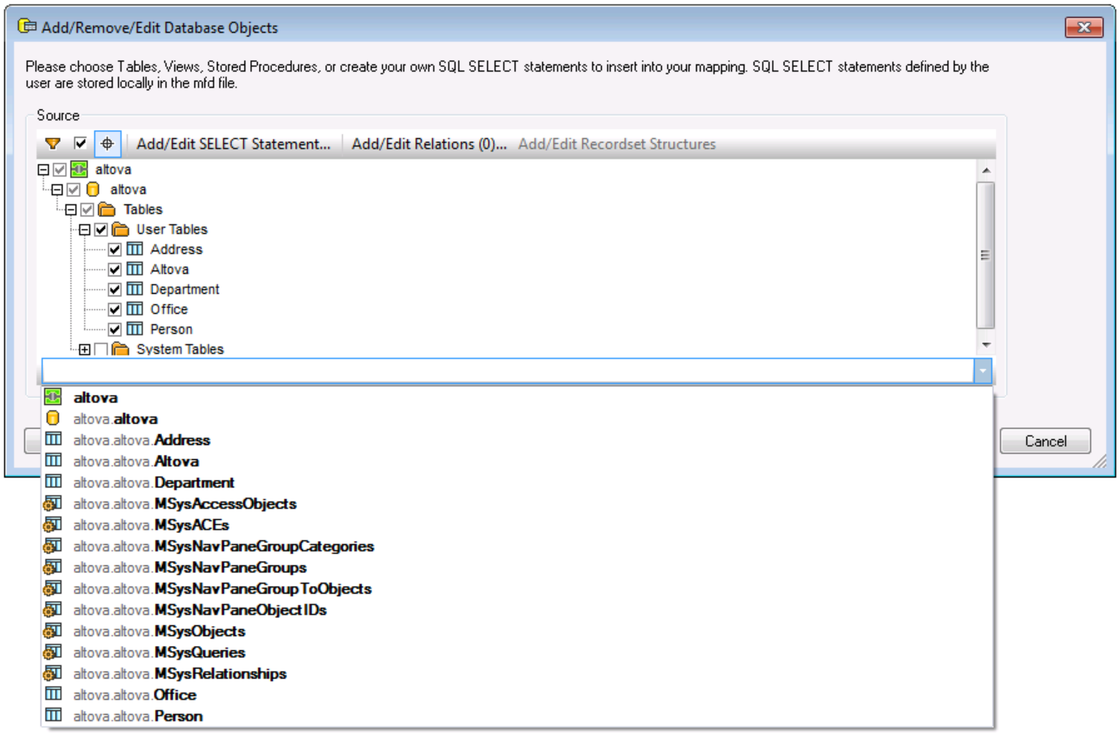
A dialog box appears, enabling you to select the database objects that you want to be included into the mapping.



Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

To include a database object (for example, a table) it in the mapping, click the check box next to it. For the purpose of this example, click 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 dialog box.

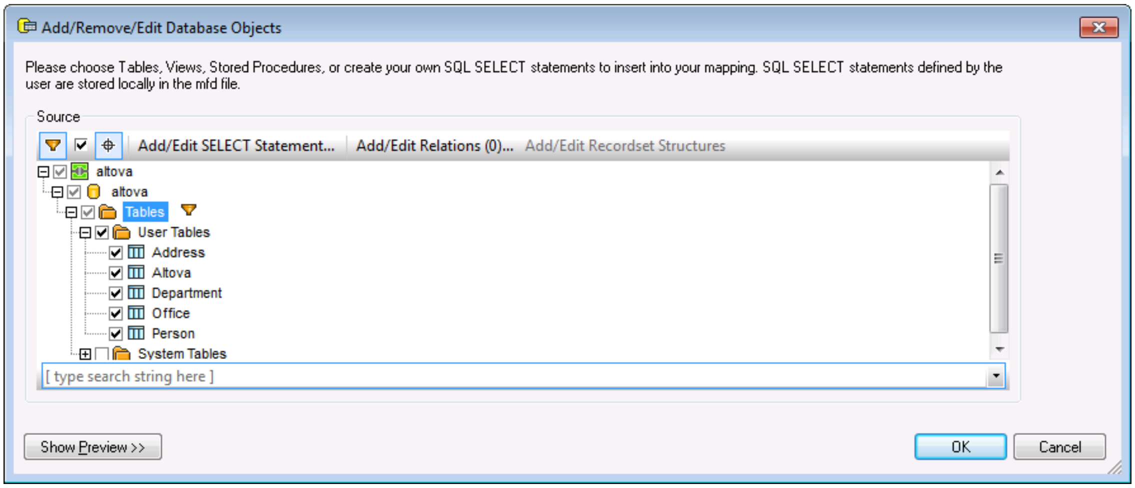


Add/Remove/Edit Database Objects

Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

Source
Add/Edit SELECT Statement...
Add/Edit Relations (0)...
Add/Edit Recordset Structures
altova
altova
Tables
User Tables
Address
Altova
Department
Office
Person
System Tables
altova
altova.altova
altova.altova.Address
altova.altova.Altova
altova.altova.Department
altova.altova.MSysAccessObjects
altova.altova.MSysACEs
altova.altova.MSysNavPaneGroupCategories
altova.altova.MSysNavPaneGroups
altova.altova.MSysNavPaneGroupToObjects
altova.altova.MSysNavPaneObjectIDs
altova.altova.MSysObjects
altova.altova.MSysQueries
altova.altova.MSysRelationships
altova.altova.Office
altova.altova.Person
Cancel

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 supports filtering (in this example, "Tables").

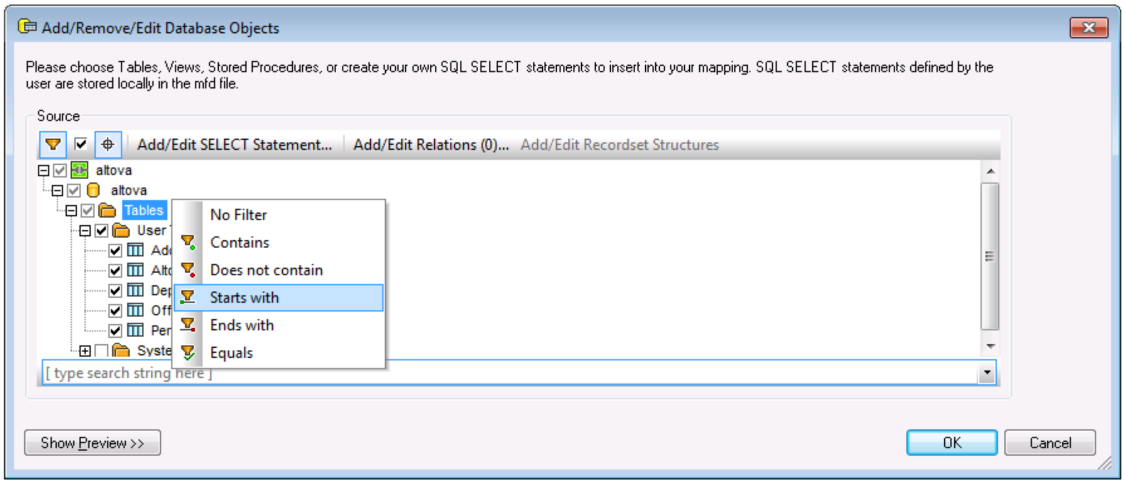


Add/Remove/Edit Database Objects

Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

Source
Add/Edit SELECT Statement...
Add/Edit Relations (0)...
Add/Edit Recordset Structures
altova
altova
Tables
User Tables
Address
Altova
Department
Office
Person
System Tables
[type search string here]
Show Preview >>
OK
Cancel

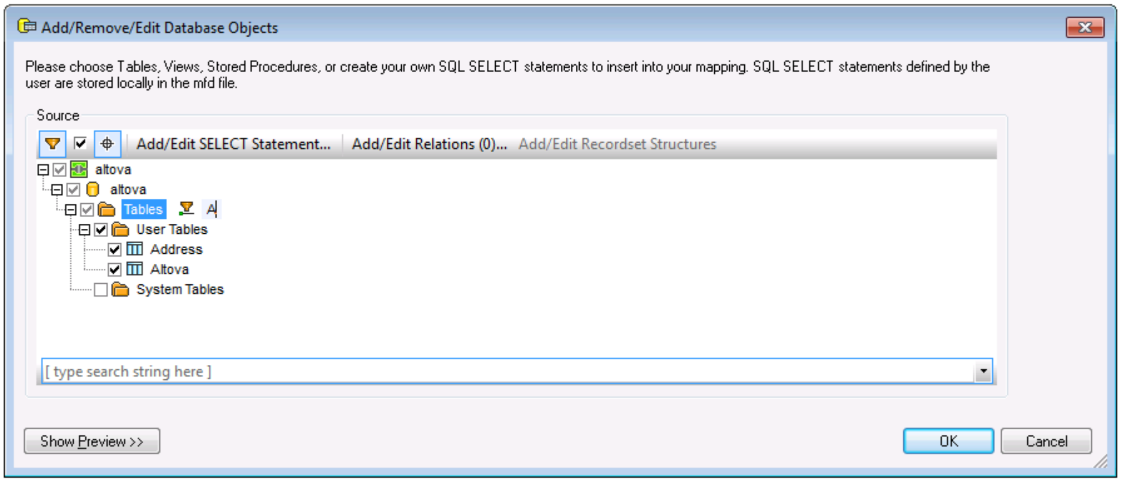
Click the filter icon to choose whether the object name should begin with, end with, be equal with, or contain the search text.



Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

Add/Remove/Edit Database Objects
Source
Add/Edit SELECT Statement...
Add/Edit Relations (0)...
Add/Edit Recordset Structures
altova
altova
Tables
User
Syste
No Filter
Contains
Does not contain
Starts with
Ends with
Equals
[type search string here]
Show Preview >>
OK
Cancel

Now you can enter the search text next to the filter (in this example, "A"):



Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

Add/Remove/Edit Database Objects
Source
Add/Edit SELECT Statement...
Add/Edit Relations (0)...
Add/Edit Recordset Structures
altova
altova
Tables
A
User Tables
Address
Altova
System Tables
[type search string here]
Show Preview >>
OK
Cancel

The **Show checked objects only** button (☑) displays those items where a check box is active.

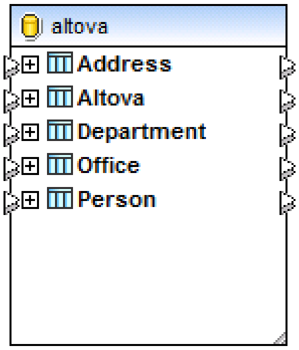
The **Add/Edit SELECT Statement** button enables you to add or edit custom SELECT statements for the current database. The data returned by such statements then becomes available as mapping source. For more information, see SQL SELECT Statements as Virtual Tables.

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.

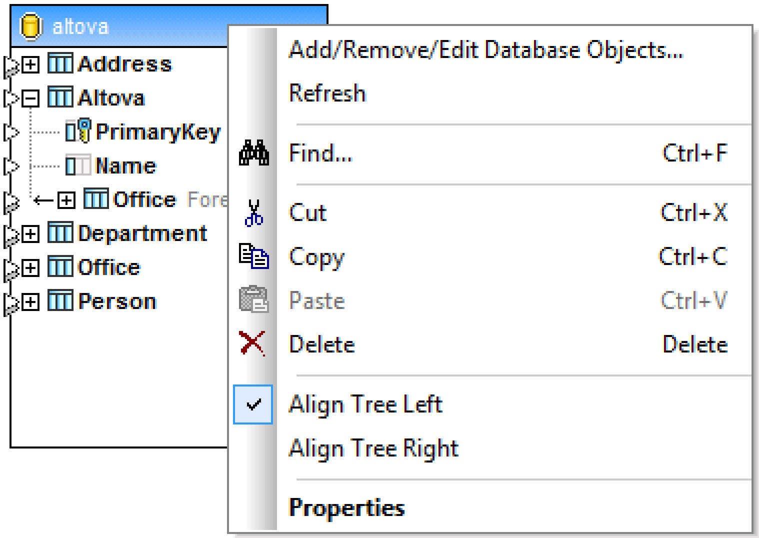
The **Add/Edit Recordset Structures** button applies for databases that support stored procedures. It is only enabled if a stored procedure is currently selected from the database tree. For more information, see Stored Procedures.

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.

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**.



7.2.2.4 *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 \MapForce2018\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 "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.

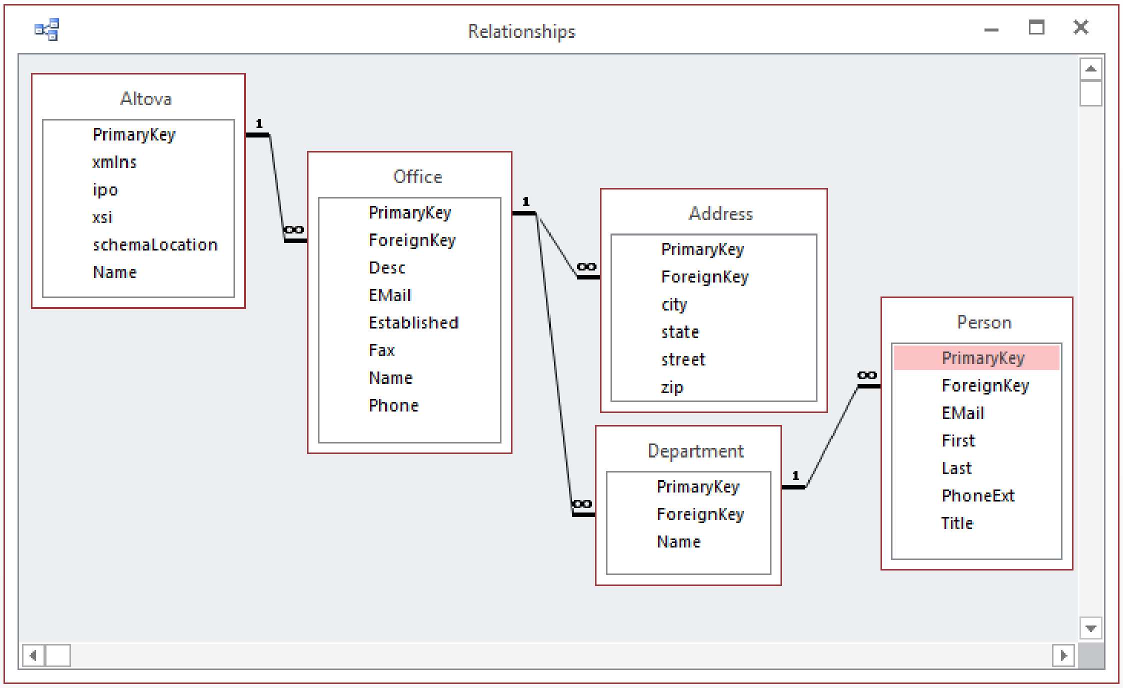
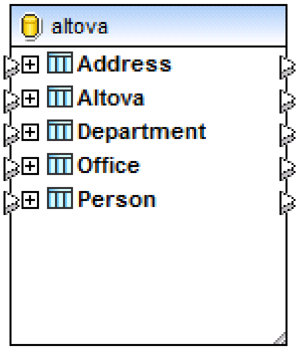


Table relationships in altova.mdb database (Microsoft Access "Relationships" view)

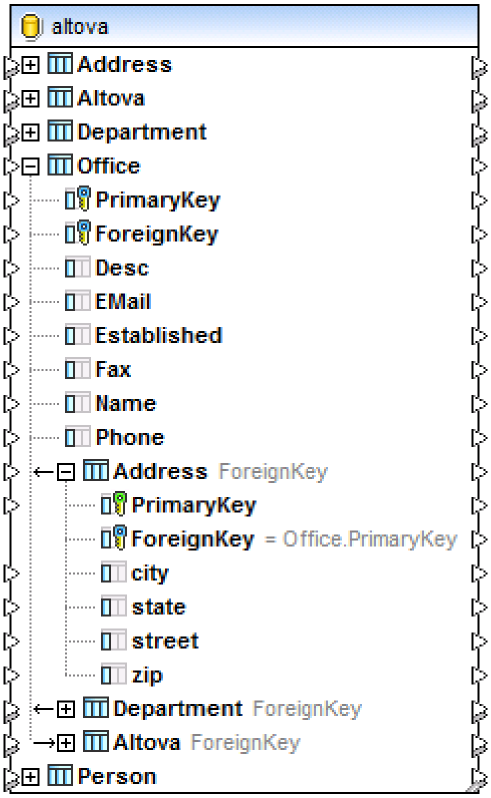
Relationships between database tables are important for mapping purposes. MapForce keeps track of such database relationships when you add a database to the mapping. This enables you to preserve the database relationships when mapping data to or from a database. To understand this concept better, add 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:

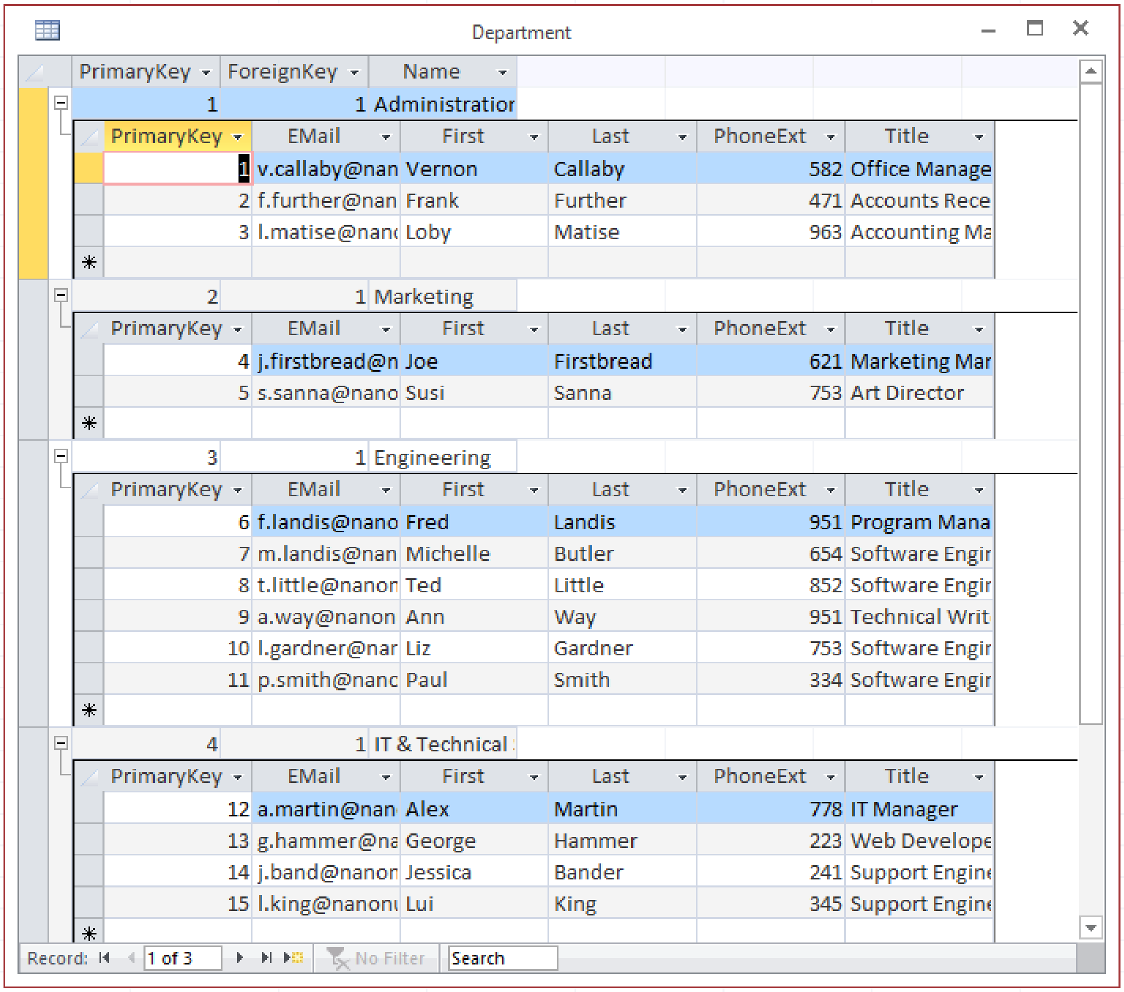
- A left arrow (←) in front of a table denotes that this is a child table. For example, **Address** is a child of **Office**. **Department** is also a child of **Office**, as well as a "sibling" table of **Address**, so both have the same indentation level. As you can see, the relationship on the mapping 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**.



altova
- Address
- Altova
- Department
- Office
  - PrimaryKey
  - ForeignKey
  - Desc
  - EMail
  - Established
  - Fax
  - Name
  - Phone
  - ← Address ForeignKey
    - PrimaryKey
    - ForeignKey = Office.PrimaryKey
    - city
    - state
    - street
    - zip
  - ← Department ForeignKey
  - → Altova ForeignKey
- Person

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:



PrimaryKey	ForeignKey	Name
1	1	Administratior

PrimaryKey	EMail	First	Last	PhoneExt	Title
1	v.callaby@nan	Vernon	Callaby	582	Office Manage
2	f.further@nan	Frank	Further	471	Accounts Rece
3	l.matise@nan	Loby	Matise	963	Accounting Ma
*					

PrimaryKey	ForeignKey	Name
2	1	Marketing

PrimaryKey	EMail	First	Last	PhoneExt	Title
4	j.firstbread@n	Joe	Firstbread	621	Marketing Mar
5	s.sanna@nano	Susi	Sanna	753	Art Director
*					

PrimaryKey	ForeignKey	Name
3	1	Engineering

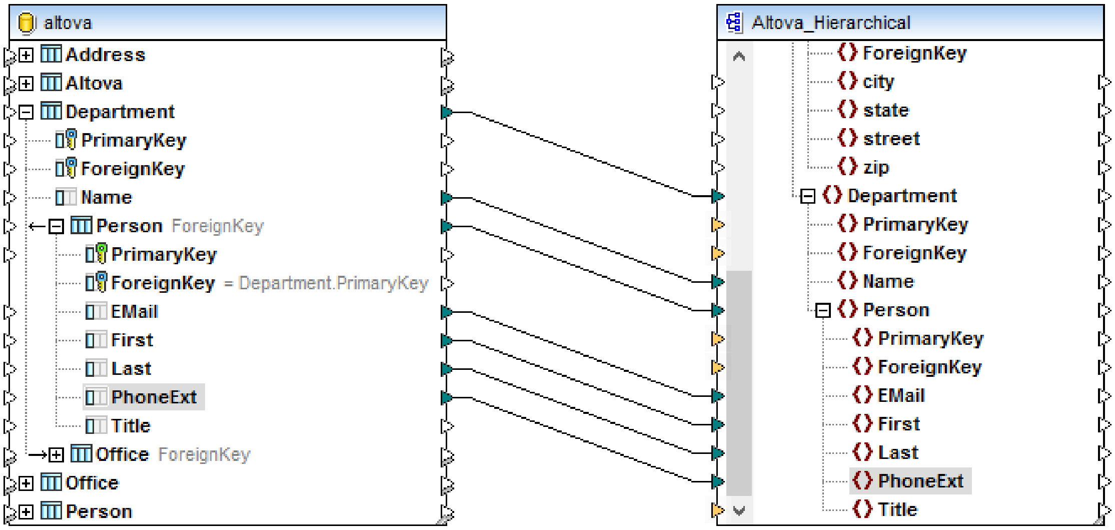
PrimaryKey	EMail	First	Last	PhoneExt	Title
6	f.landis@nano	Fred	Landis	951	Program Mana
7	m.landis@nan	Michelle	Butler	654	Software Engir
8	t.little@nanor	Ted	Little	852	Software Engir
9	a.way@nanon	Ann	Way	951	Technical Writ
10	l.gardner@nar	Liz	Gardner	753	Software Engir
11	p.smith@nanc	Paul	Smith	334	Software Engir
*					

PrimaryKey	ForeignKey	Name
4	1	IT & Technical

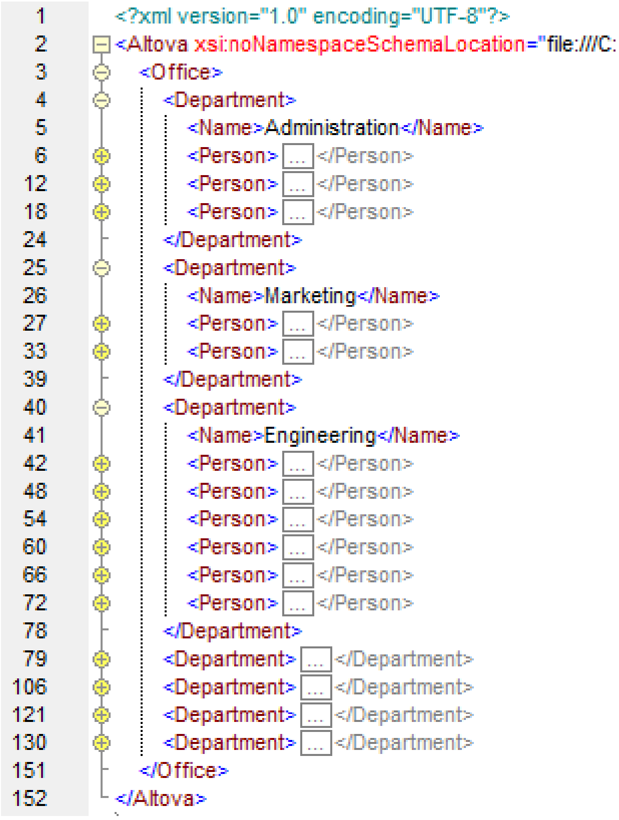
PrimaryKey	EMail	First	Last	PhoneExt	Title
12	a.martin@nan	Alex	Martin	778	IT Manager
13	g.hammer@na	George	Hammer	223	Web Develope
14	j.band@nanor	Jessica	Bander	241	Support Engine
15	l.king@nanon	Lui	King	345	Support Engine
*					

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**:

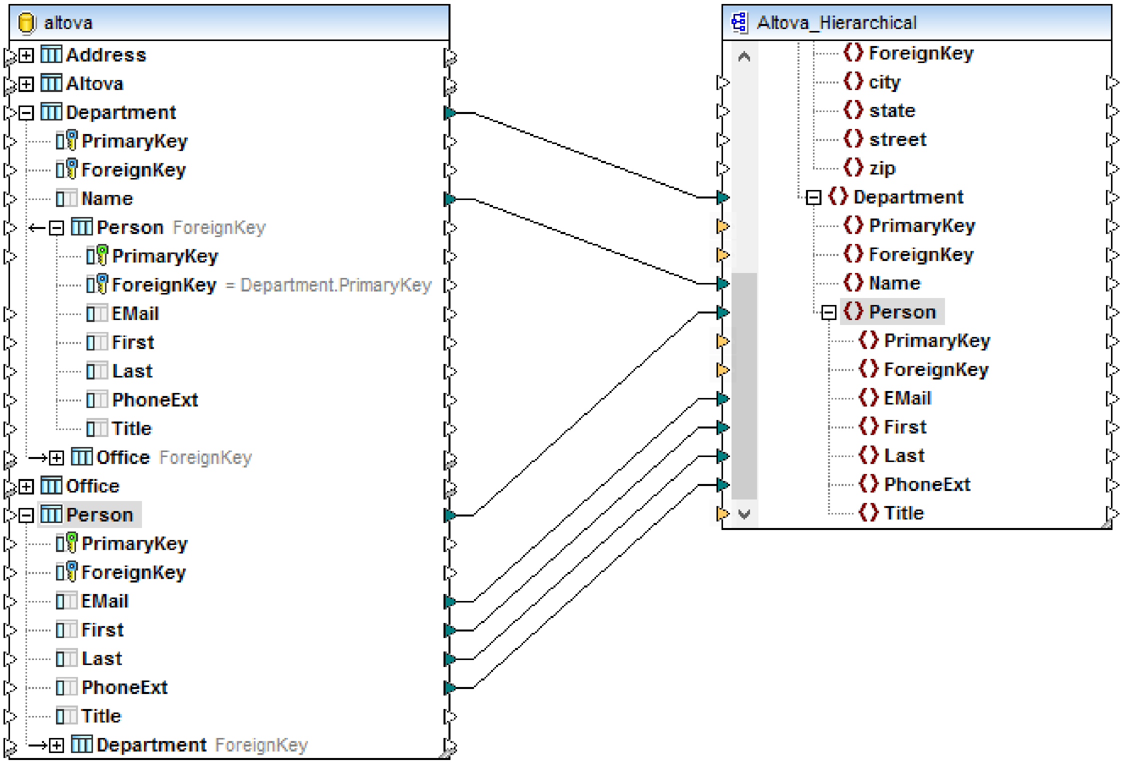


The mapping above is a modified **DB_Altova_Hierarchical.mfd** from the **<Documents>\Altova\MapForce2018\MapForceExamples\** folder. When you preview the mapping, the result is that each person is grouped by department, which was the intended behaviour. That is, "Administration" has three people, "Marketing" has two people, "Engineering" has six people, etc.

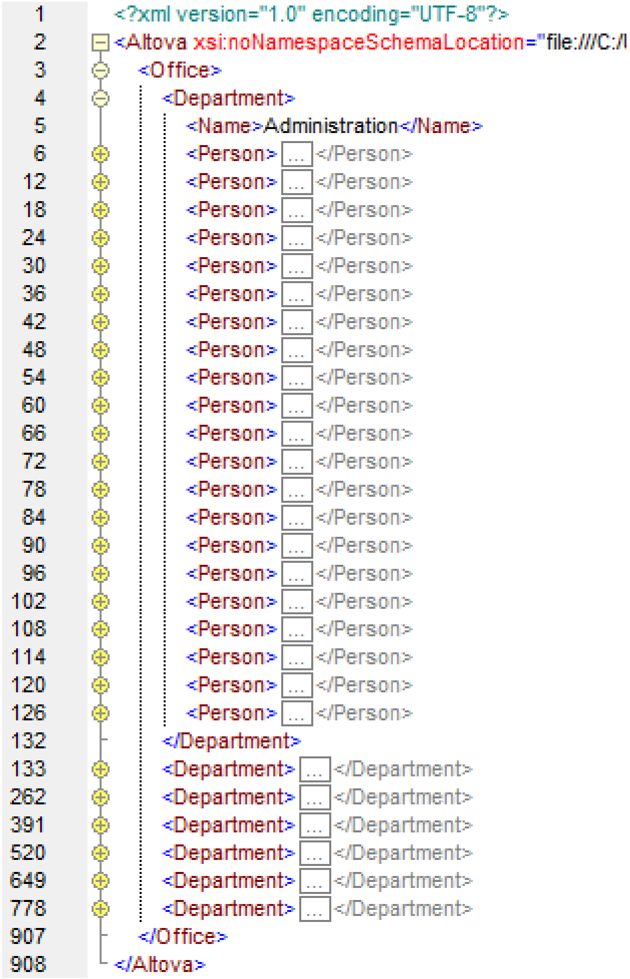


```
<?xml version="1.0" encoding="UTF-8"?>
<Altova xsi:noNamespaceSchemaLocation="file:///C:
  <Office>
    <Department>
      <Name>Administration</Name>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
    </Department>
    <Department>
      <Name>Marketing</Name>
      <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.

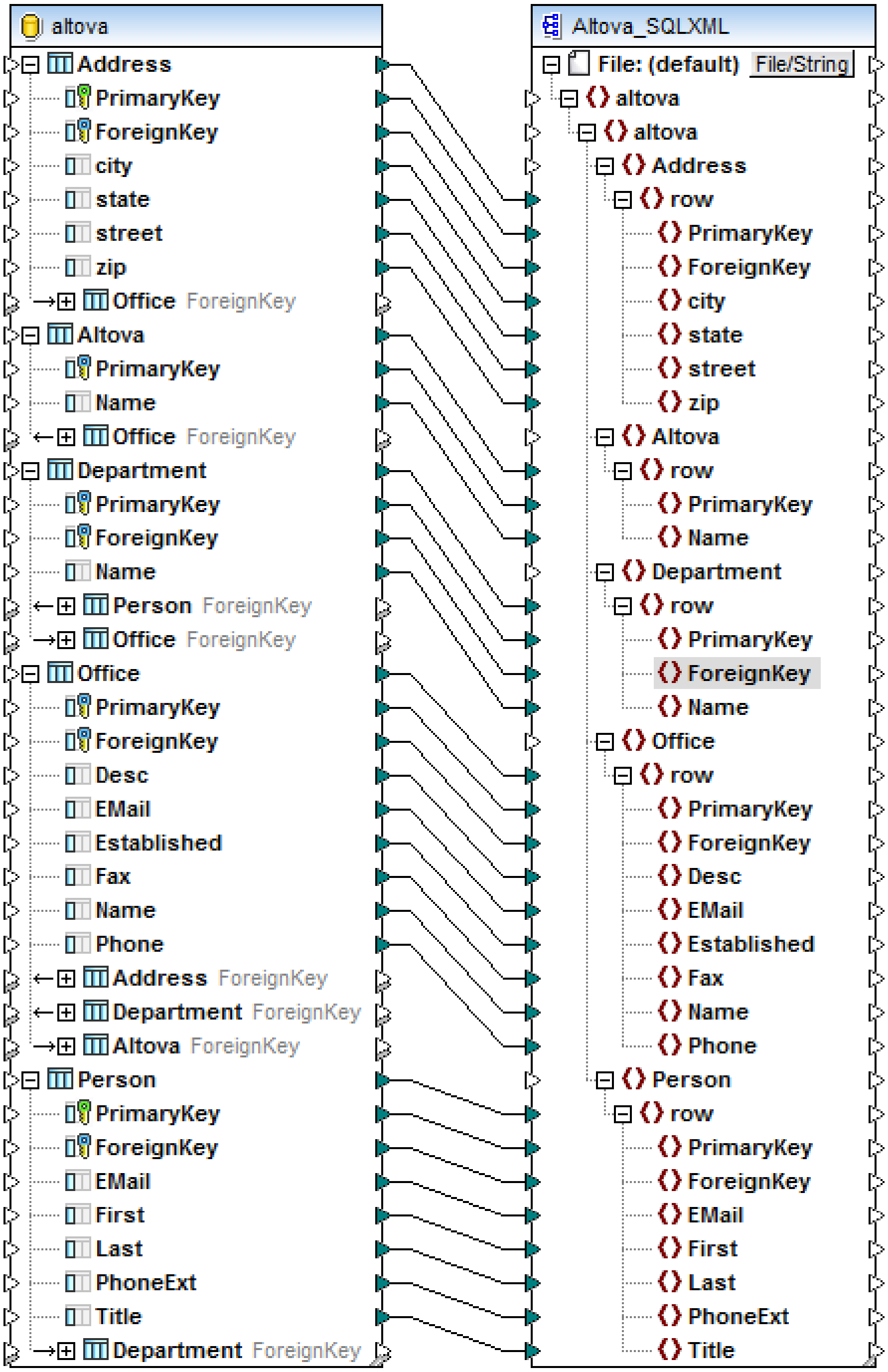


```
<?xml version="1.0" encoding="UTF-8"?>
<Altova xsi:noNamespaceSchemaLocation="file:///C:/I
  <Office>
    <Department>
      <Name>Administration</Name>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
      <Person> ... </Person>
    </Department>
    <Department> ... </Department>
    <Department> ... </Department>
    <Department> ... </Department>
    <Department> ... </Department>
    <Department> ... </Department>
    <Department> ... </Department>
  </Office>
</Altova>
```

In the second example, the database relationships are disregarded, due to the way the connections were made.

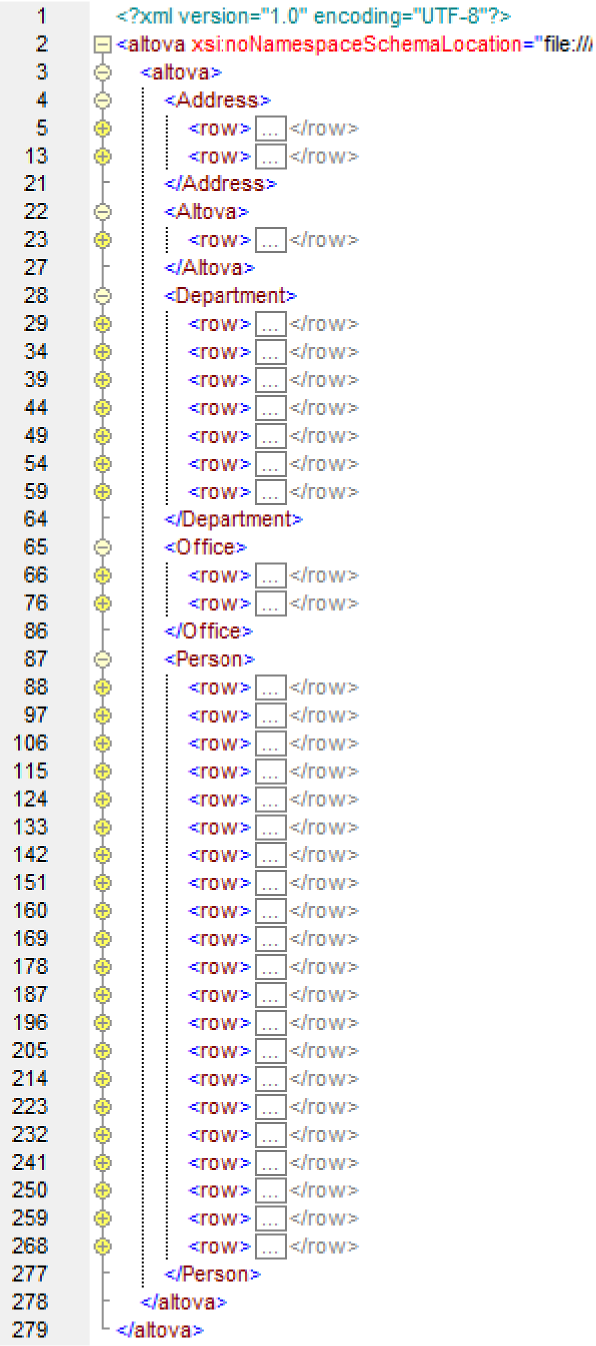
Therefore, when you want to preserve database relationships, make sure that connections are drawn to or from the same "root" table, which contains the child tables whose relationships you want to preserve. This works in the same way for both source and target databases. For examples of database mappings which preserve relationships, see the **DB_Altova_Hierarchical.mfd** and **Altova_Hierarchical_DB.mfd** files available in the **<Documents>\Altova\MapForce2018\MapForceExamples\** folder (see also Inserting Data into Multiple Linked Tables).

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\MapForce2018\MapForceExamples\** folder. The goal of the mapping is to get database data as 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.

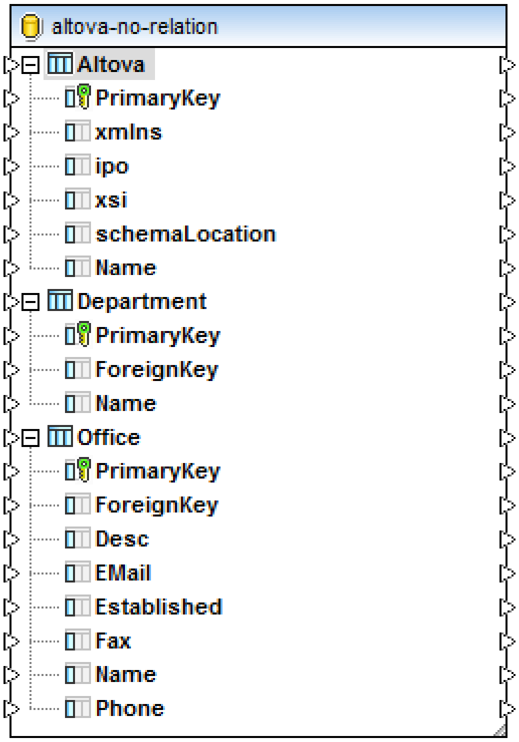
7.2.2.5 *Defining Local Relationships*

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. Local relations can be defined for the following database objects:

- Database tables
- Database views
- Stored procedures and functions
- User-defined SELECT statements

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.

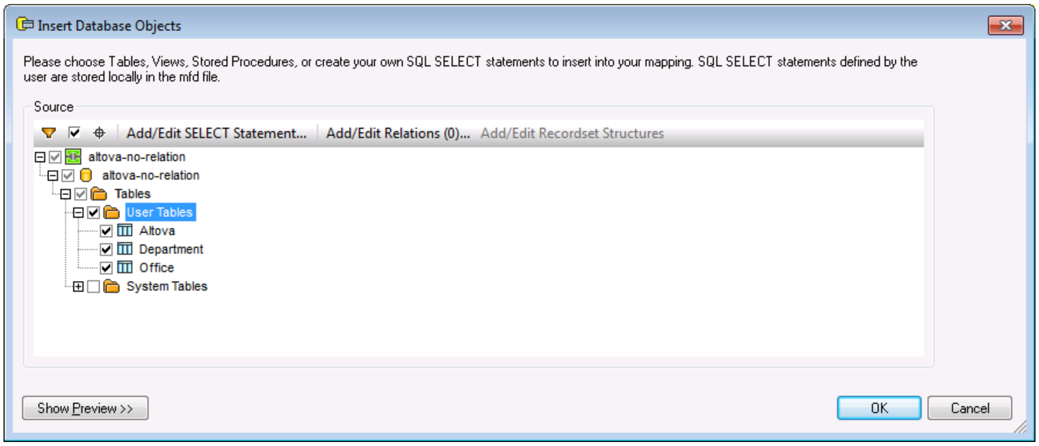


Database structure with no explicit relationships

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.

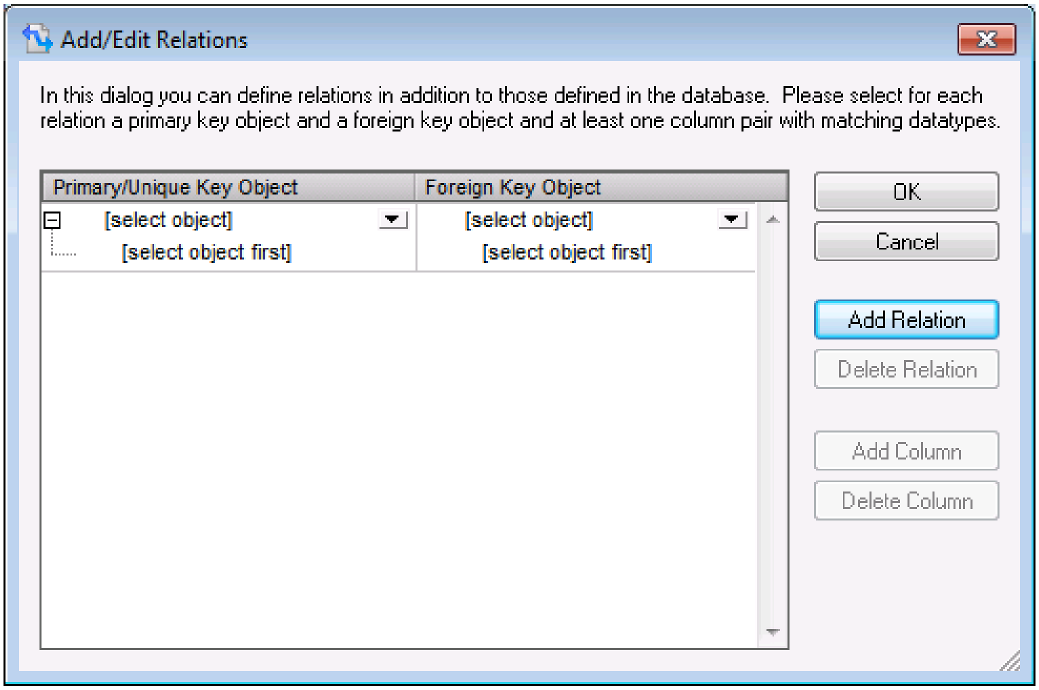
Local relations can be defined while adding a database to the mapping, or by right-clicking an existing database component and selecting **Add/Remove/Edit Database Objects** from the context menu, as illustrated in the steps below.

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 \MapForce2018\MapForceExamples\Tutorial\** folder, and click **Connect**.
4. Select the **User Tables** check box.



Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

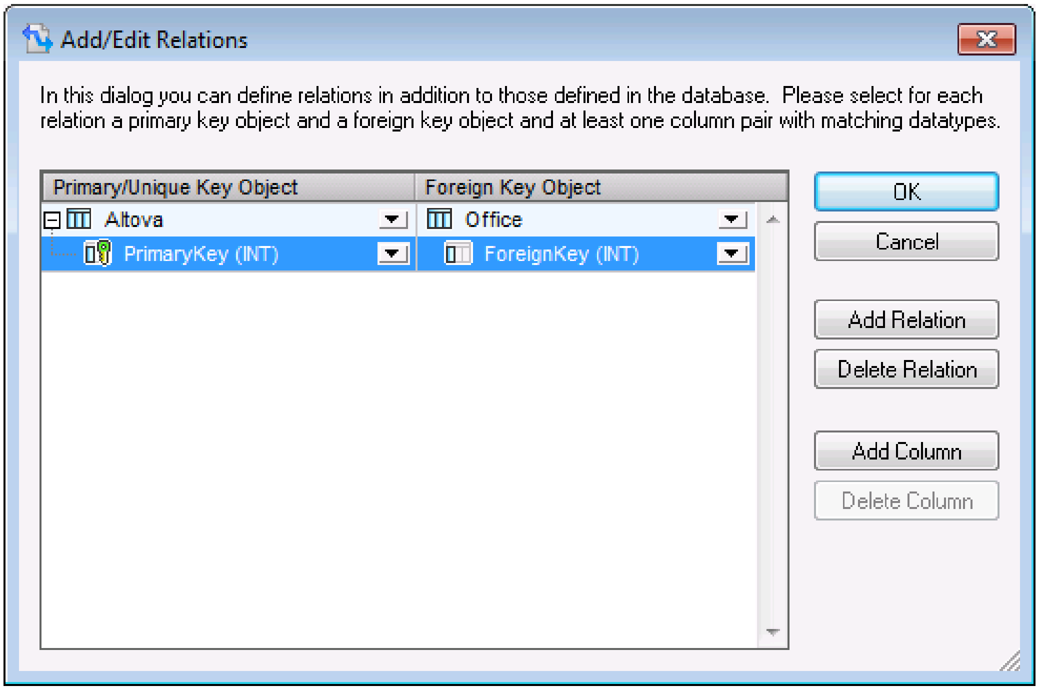
5. Click the **Add/Edit Relations** button in the icon bar.
6. The "Add/Edit Table Relations" dialog box opens. Click **Add Relation**.



In this dialog you can define relations in addition to those defined in the database. Please select for each relation a primary key object and a foreign key object and at least one column pair with matching datatypes.

Primary/Unique Key Object	Foreign Key Object
[select object]	[select object]
[select object first]	[select object first]

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).

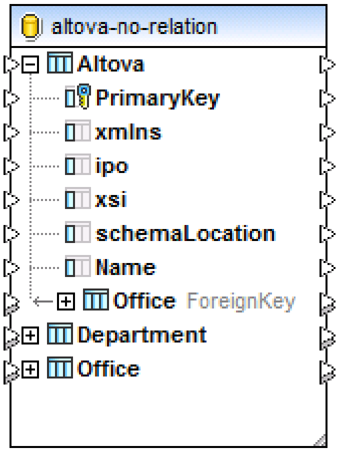

Add/Edit Relations

In this dialog you can define relations in addition to those defined in the database. Please select for each relation a primary key object and a foreign key object and at least one column pair with matching datatypes.

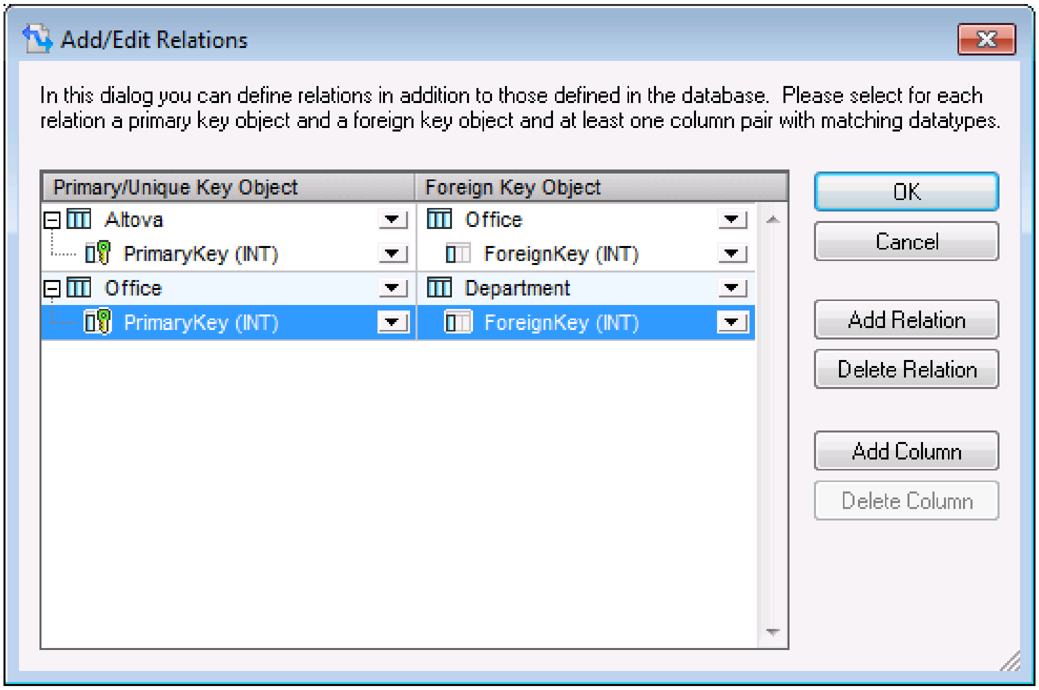
Primary/Unique Key Object	Foreign Key Object
Altova	Office
PrimaryKey (INT)	ForeignKey (INT)

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.



Add/Edit Relations

In this dialog you can define relations in addition to those defined in the database. Please select for each relation a primary key object and a foreign key object and at least one column pair with matching datatypes.

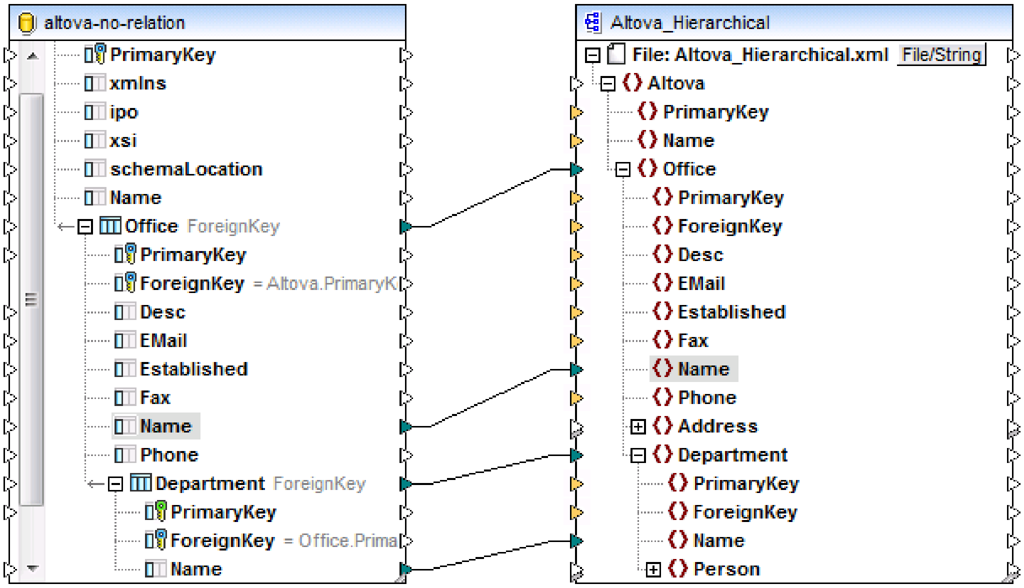
Primary/Unique Key Object	Foreign Key Object
Altova	Office
PrimaryKey (INT)	ForeignKey (INT)
Office	Department
PrimaryKey (INT)	ForeignKey (INT)

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.

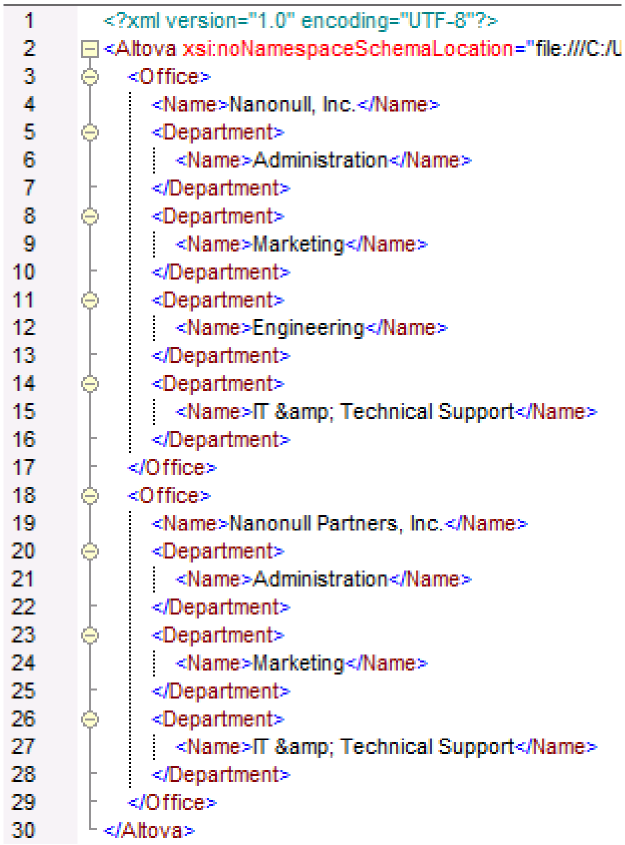
Finally, add the target schema to the mapping as follows:

1. On the **Insert** menu, click **Insert XML Schema/File**.
2. Browse for the **Altova_Hierarchical.xsd** file available in the **<Documents>\Altova\MapForce2018\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.



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.

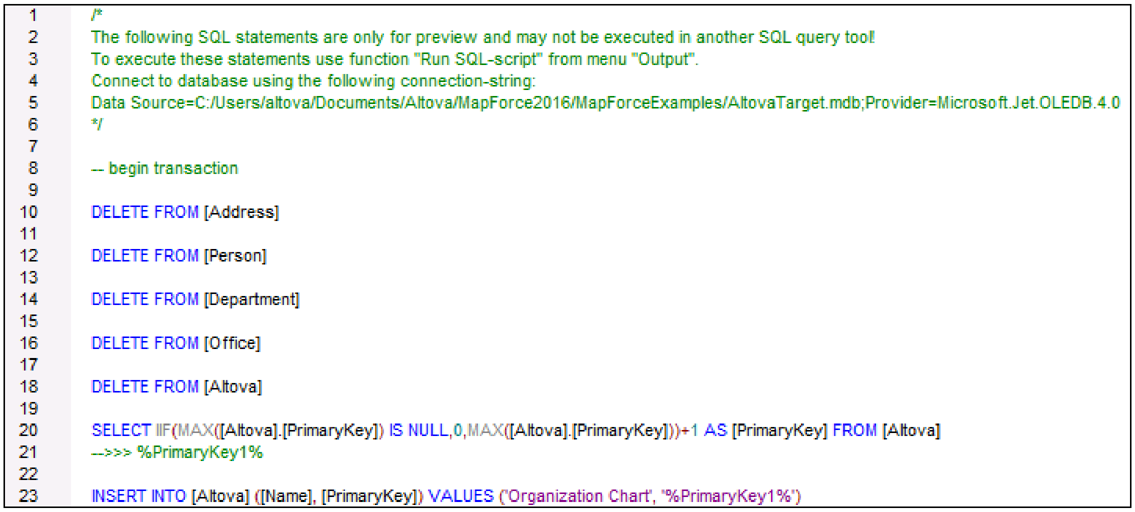


```
<?xml version="1.0" encoding="UTF-8"?>
<Altova xsi:noNamespaceSchemaLocation="file:///C:/U
    <Office>
        <Name>Nanonull, Inc.</Name>
        <Department>
            <Name>Administration</Name>
        </Department>
        <Department>
            <Name>Marketing</Name>
        </Department>
        <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>
```

7.2.2.6 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. This script shows the SQL statements that are to be applied against the database. 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)

Note that this 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. Also, if multiple actions are defined against a table (for example, "Update if... Insert Rest") only the first action is shown in the preview, since the second action is executed conditionally.

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\MapForce2018\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\MapForce2018\MapForceExamples\** folder before updating any files in it.

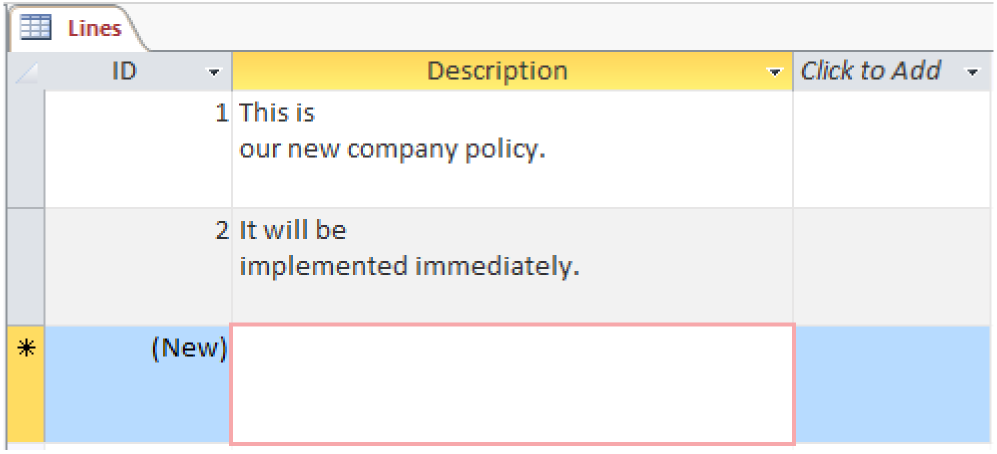
For information about running mappings in execution environments other than MapForce, see:

- Deploying Mappings to FlowForce Server
- Compiling Mappings to MapForce Server Execution Files
- Code Generator

7.2.2.7 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. This can be done with the help of the MapForce library function **`char-from-code`**.

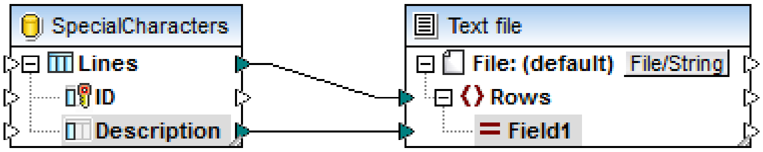
Consider a Microsoft Access database consisting of a table "Lines" which has two columns: "ID" and "Description".



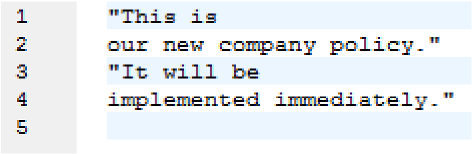
Lines

ID	Description	*Click to Add*
1	This is our new company policy.	
2	It will be implemented immediately.	
(New)		

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:



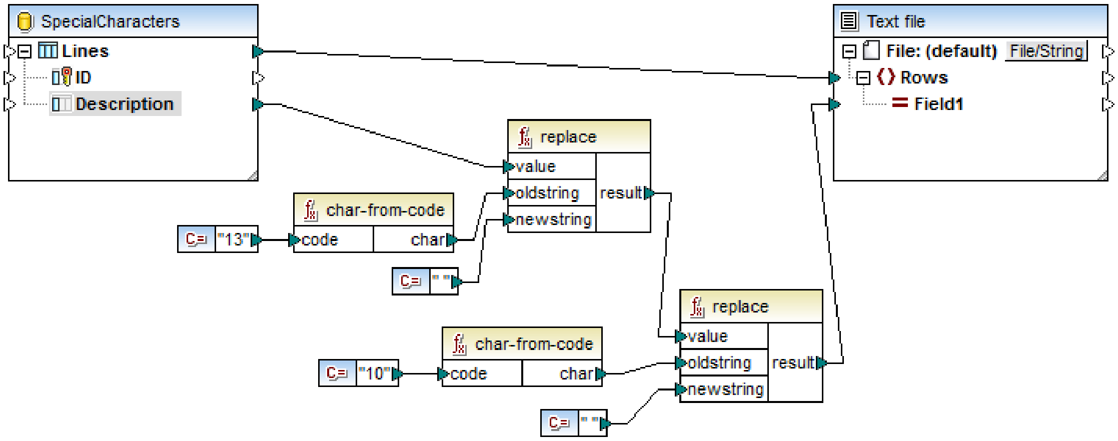
However, because each "Description" row in Access contains multiple lines separated by CR/LF characters, the mapping output includes line breaks also, which is not the intended result:



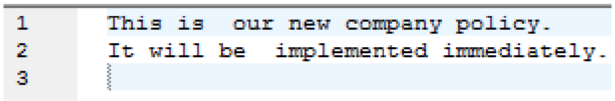
```
"This is
our new company policy."
"It will be
implemented immediately."

```

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 Working with Functions). 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.

```

7.2.2.8 Handling Null Values

To check at mapping runtime whether a database field is null, use the **`is-null`** and **`is-not-null`** MapForce library functions. To see from MapForce if a table has null fields, query it using the Database Browser (see Browsing and Querying Databases).

To set a database field to null, use the **`set-null`** function.

To replace null database values with a string, use the **`substitute-null`** function. A sample mapping that illustrates this is **DB_ApplicationList.mfd** available in the **<Documents>\Altova \MapForce2018\MapForceExamples\** folder.

For information about handling NULL value comparisons in mappings which update databases, see Handling Nulls in Database Table Actions.

For information about handling nulls when mapping database to or from XML documents, see Nil Values / Nillable.

7.2.2.9 *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:

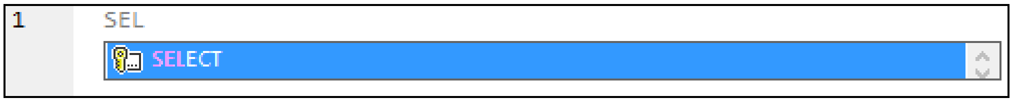
- **`auto-number`** (available in the "core | generator functions" library). This function is generally used to generate primary key values for a numeric field.
- **`create-guid`** (available in the "lang | generator functions" library). This function creates a globally-unique identifier (as a hex-encoded 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) and is particularly useful when generating primary keys.

7.2.2.10 *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)
- "Custom SQL" text box in the "Database Table Actions" dialog box (see Database Table Actions Settings)
- "Enter a SQL SELECT statement" dialog box (see Creating SELECT Statements)



```
1    SEL
     SELECT
```

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.

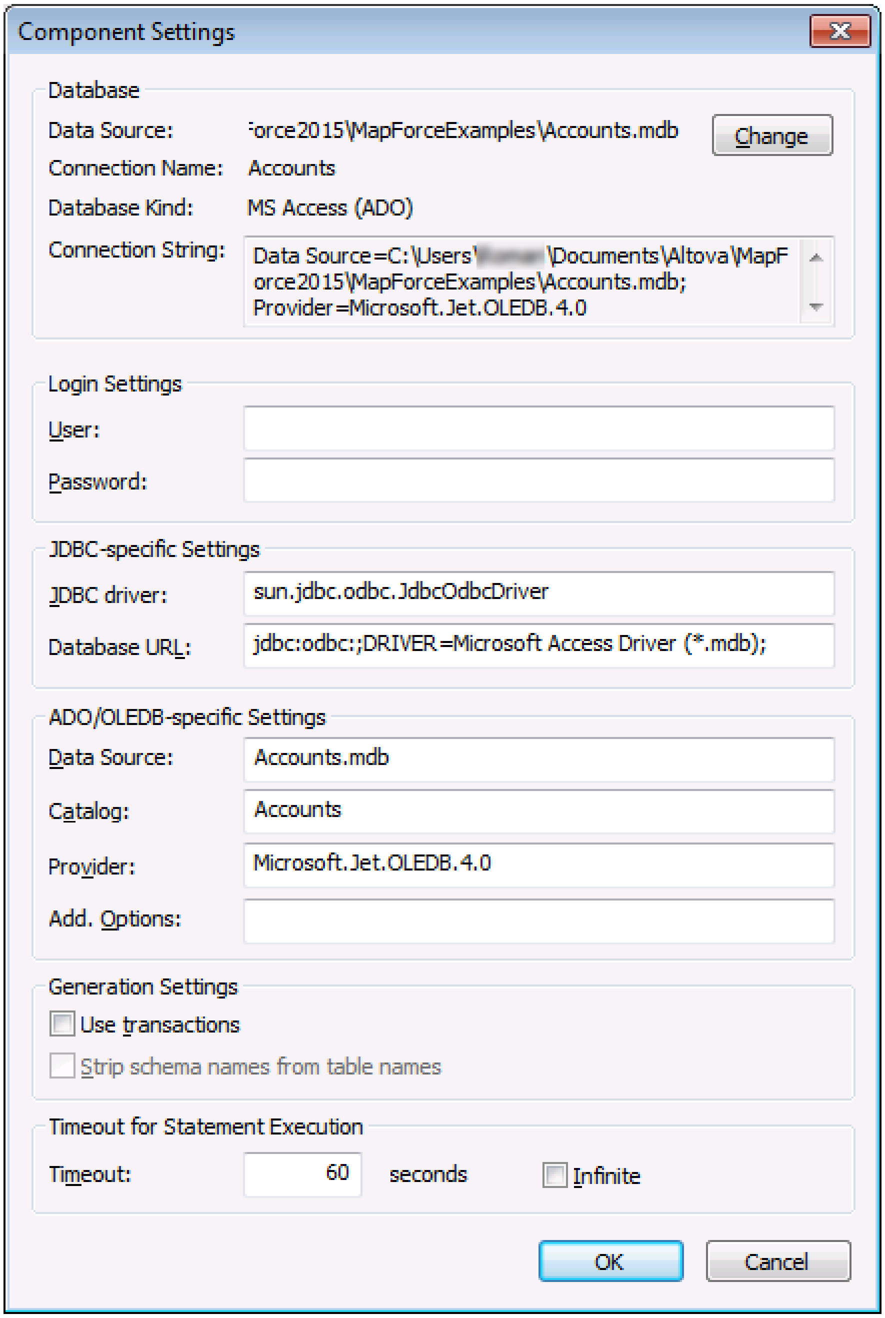
To invoke auto-completion suggestions manually as and when required:

- Press **Ctrl+Space**.

7.2.2.11 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, and then, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component, and then click **Properties**.



Database
Data Source: ·orce2015\MapForceExamples\Accounts.mdb
Connection Name: Accounts
Database Kind: MS Access (ADO)
Connection String: Data Source=C:\Users\[redacted]\Documents\Altova\MapForce2015\MapForceExamples\Accounts.mdb;Provider=Microsoft.Jet.OLEDB.4.0

Login Settings
User:
Password:

JDBC-specific Settings
JDBC driver: sun.jdbc.odbc.JdbcOdbcDriver
Database URL: jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);

ADO/OLEDB-specific Settings
Data Source: Accounts.mdb
Catalog: Accounts
Provider: Microsoft.Jet.OLEDB.4.0
Add. Options:

Generation Settings
☐ Use transactions
☐ Strip schema names from table names

Timeout for Statement Execution
Timeout: 60 seconds ☐ Infinite

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 is the same as the data source name if the database uses a data source)
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 OS X / macOS machine, see Database mappings in various execution environments.

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. 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.

Use transactions	Enables transaction processing when using a database as a target. A dialog box opens when an error is encountered allowing you to choose how to proceed. Transaction processing is enabled for all tables of the database component when you select this option. For more information, see Using Transaction Rollback.
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 only works for SQL Select statements generated by MapForce. User-defined SQL-Statements, when creating virtual tables, will not be modified.

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.

7.2.3 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
Control how primary key values are to be created...	• Inserting Data into a Table • Inserting Data into Multiple Linked Tables
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 • Database Table Actions Settings
Preserve the hierarchical relationship of records in tables linked by foreign keys...	• Inserting Data into Multiple Linked Tables
Update a table conditionally...	• Updating a Table
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 • MERGE statements
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
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 • "Ignore if..." Action
Preserve data integrity in case of failed mapping execution...	• Using Transaction Rollback
Insert multiple records into a database table in bulk (combine multiple INSERT statements in one query)...	• Bulk Inserts (MapForce Server)
Avoid undesired results when mapping data to target database tables that contain null	• Handling Nulls in Database Table Actions

I want to...	Read this topic...
values...	
View an example of how to create a mapping which updates a database.	• Example: Mapping Data from XML to SQLite

7.2.3.1 *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\MapForce2018\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.

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

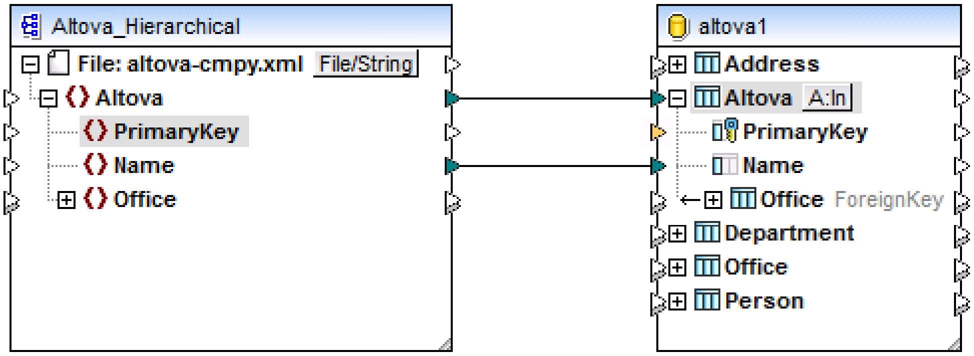
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **Altova-cmpy.xml**.

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).

Step 3: Draw the connections

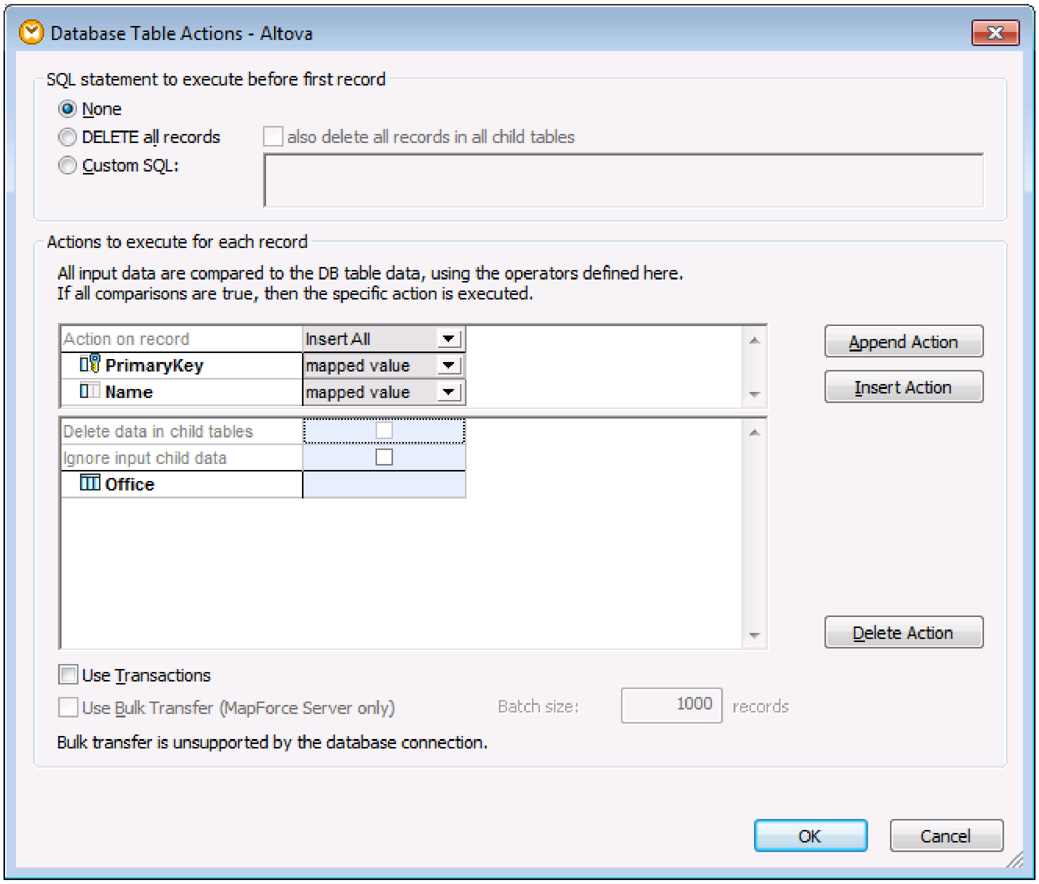
- Draw the mapping connections as shown below.



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.


Database Table Actions - Altova

SQL statement to execute before first record

● None
○ DELETE all records □ also delete all records in all child tables
○ Custom SQL:

Actions to execute for each record

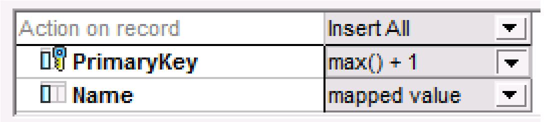
All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Insert All
PrimaryKey	mapped value
Name	mapped value

Delete data in child tables	□
Ignore input child data	□
Office	

□ Use Transactions
□ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records
Bulk transfer is unsupported by the database connection.

2. In the Database Table Actions dialog box, under **Insert All**, next to **PrimaryKey**, select the **max() + 1** option.



Action on record	Insert All
PrimaryKey	max() + 1
Name	mapped value

The options available in this list have the following meaning:

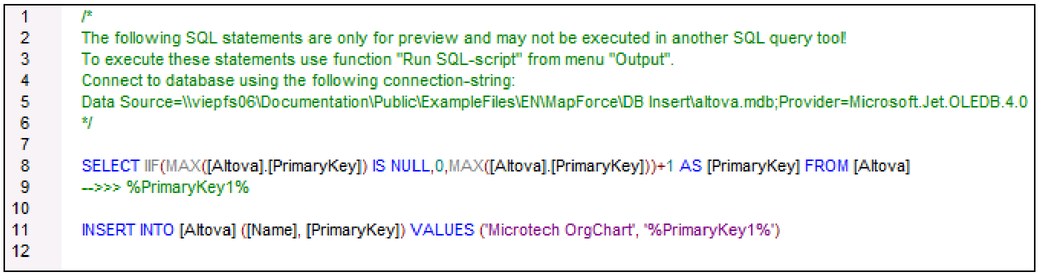
Option	Description
mapped value	Allows source data to be mapped to the database field directly, and is the standard setting for all database fields. It is also possible to use a stored procedure to supply a key value by defining a relation, see Using stored procedures to generate primary keys.
max() + 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.

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.



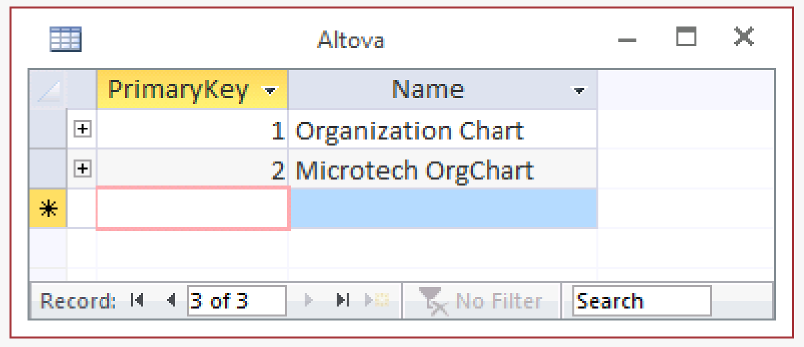
```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
Data Source=\\viepfs06\Documentation\Public\ExampleFiles\EN\MapForce\DB 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.

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.



PrimaryKey	Name
1	Organization Chart
2	Microtech OrgChart

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.

7.2.3.2 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. 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\MapForce2018\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.

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).

To achieve the mapping goal, we will take the steps below.

Step 1: Insert the source XML component

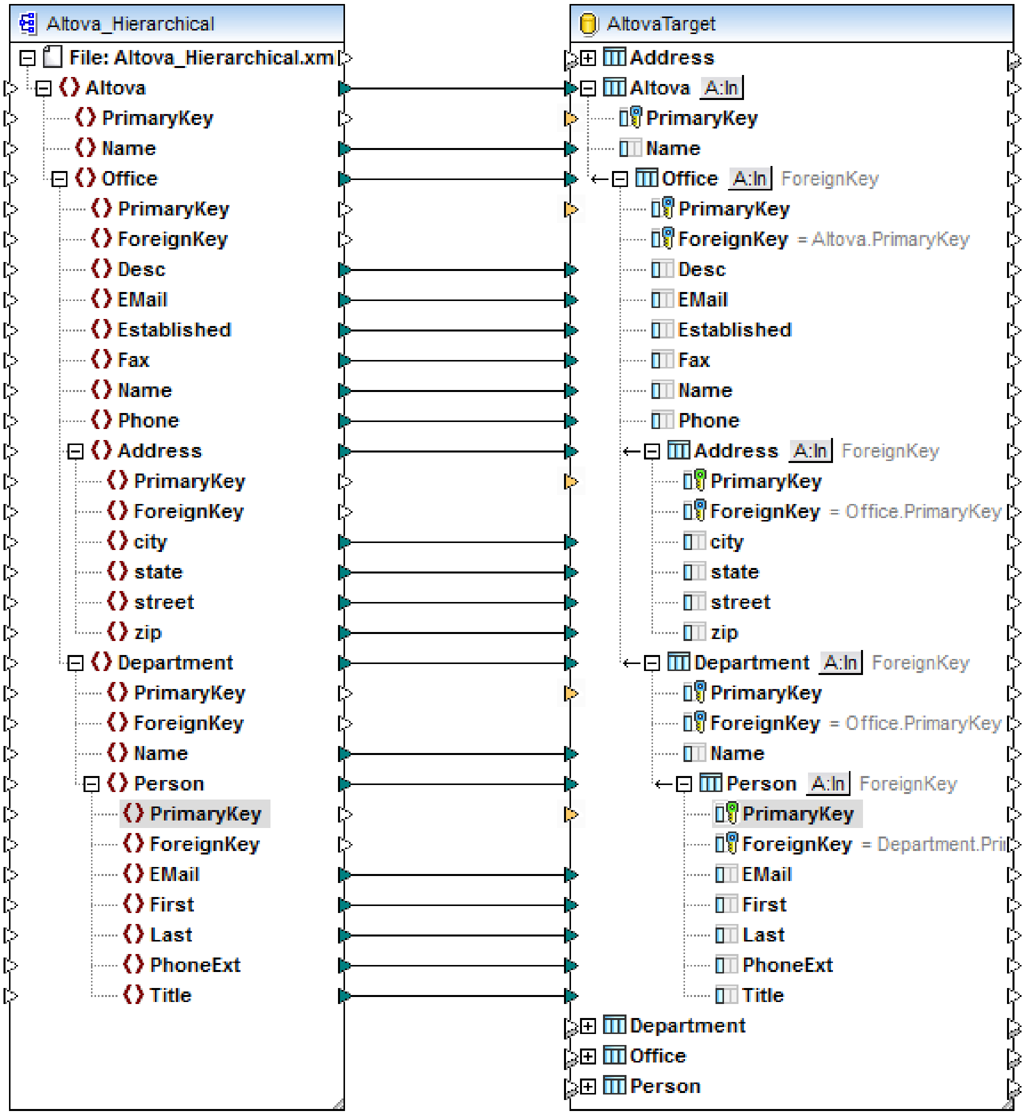
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **Altova_Hierarchical.xml**.

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).

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.

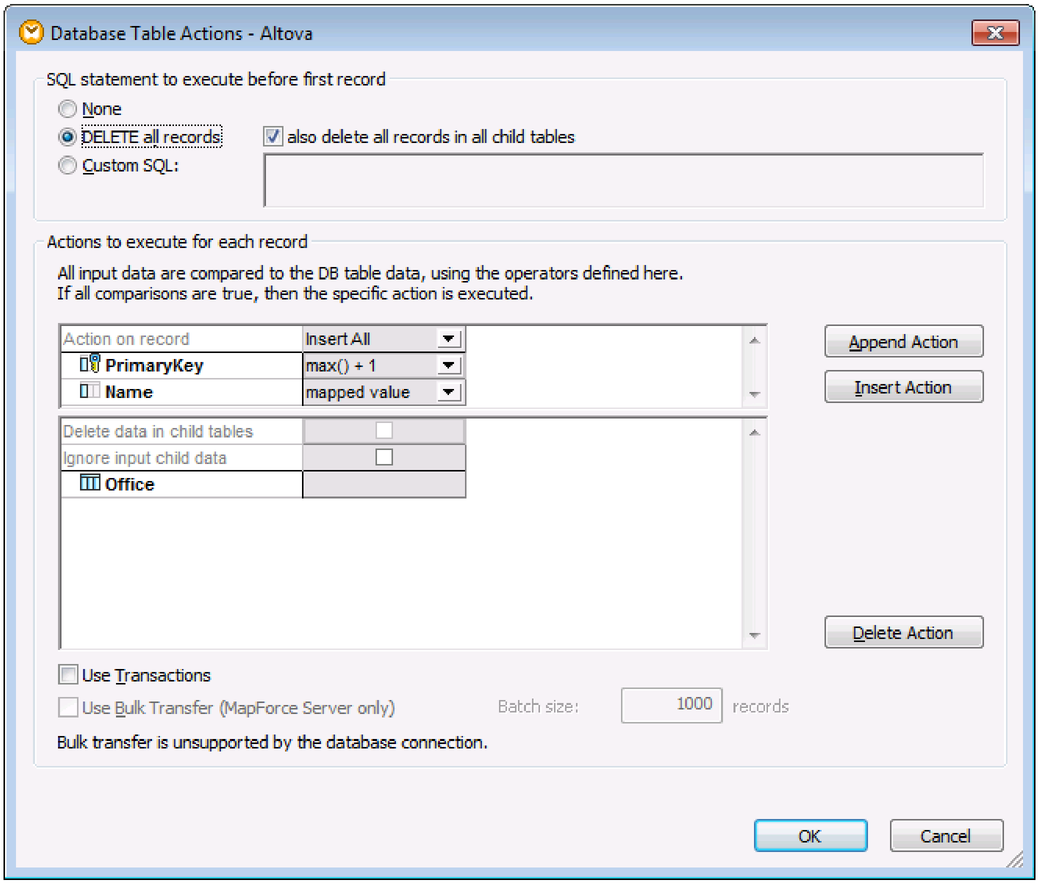


Note: If unwanted connections are automatically drawn for descending items, the option "Auto-connect children" is active. In this case, to undo the last action, select the menu option **Edit | Undo**. To disable the auto-connect option, select the menu option **Connection | Auto-connect matching children**.

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.



Database Table Actions - Altova

SQL statement to execute before first record

○ None

● DELETE all records ☑ also delete all records in all child tables

○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Insert All
PrimaryKey	max() + 1
Name	mapped value

Delete data in child tables	□
Ignore input child data	□
Office	

□ Use Transactions

□ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records

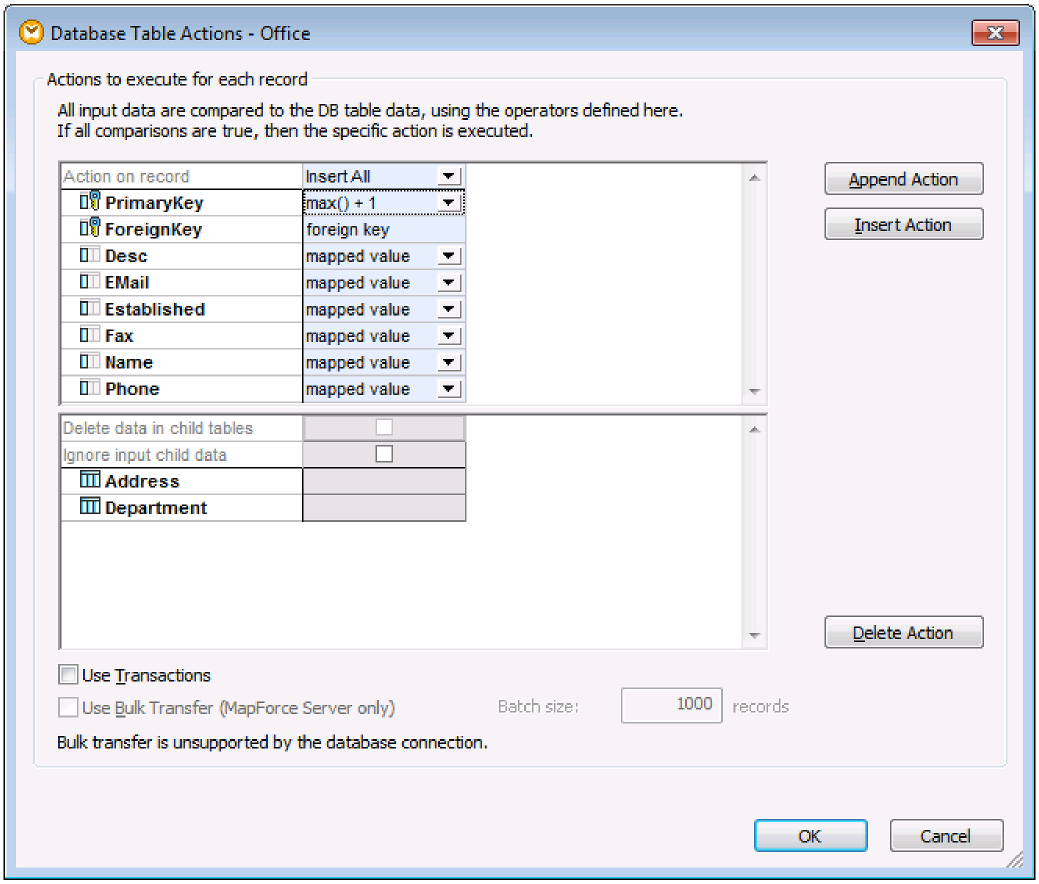
Bulk transfer is unsupported by the database connection.

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 "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.

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.



Database Table Actions - Office

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Insert All
PrimaryKey	max() + 1
ForeignKey	foreign key
Desc	mapped value
EMail	mapped value
Established	mapped value
Fax	mapped value
Name	mapped value
Phone	mapped value

Delete data in child tables	☐
Ignore input child data	☐
Address	
Department	

☐ Use Transactions

☐ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records

Bulk transfer is unsupported by the database connection.

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.

Step 5: Preview the mapping and update the database

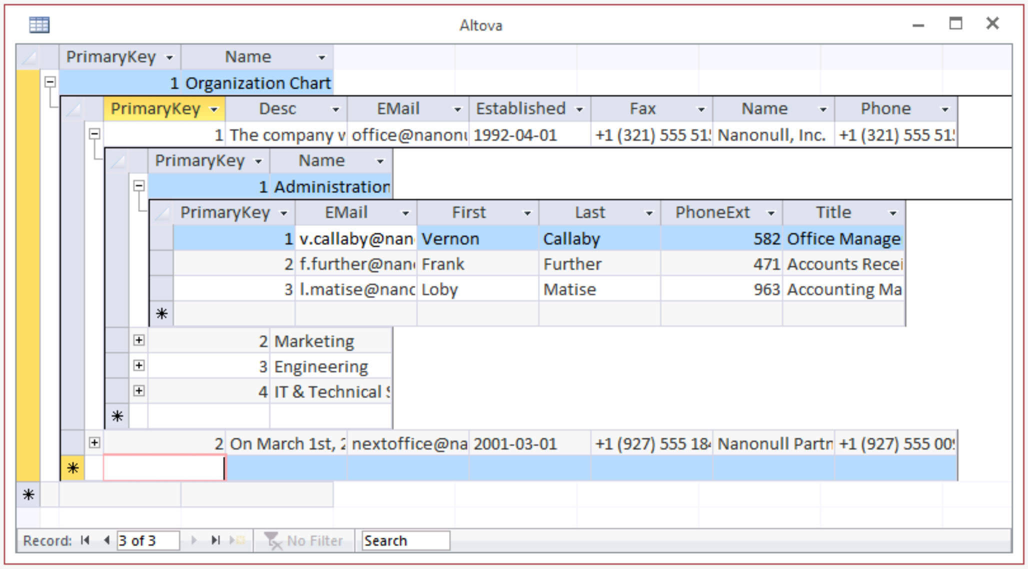
Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview.

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.

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.



PrimaryKey	Name
1	Organization Chart

PrimaryKey	Desc	EMail	Established	Fax	Name	Phone
1	The company v	office@nanon	1992-04-01	+1 (321) 555 51	Nanonull, Inc.	+1 (321) 555 51

PrimaryKey	Name
1	Administration

PrimaryKey	EMail	First	Last	PhoneExt	Title
1	v.callaby@nan	Vernon	Callaby	582	Office Manage
2	f.further@nan	Frank	Further	471	Accounts Recei
3	l.matise@nanc	Loby	Matise	963	Accounting Ma

PrimaryKey	Name
2	Marketing
3	Engineering
4	IT & Technical S

PrimaryKey	Desc	EMail	Established	Fax	Name	Phone
2	On March 1st, 2	nextoffice@na	2001-03-01	+1 (927) 555 18	Nanonull Partn	+1 (927) 555 00

You have now finished creating a mapping which inserts data into multiple database tables, while preserving the table integrity relationships.

7.2.3.3 *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\MapForce2018\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.

The goal of the mapping is to update all records in "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**.

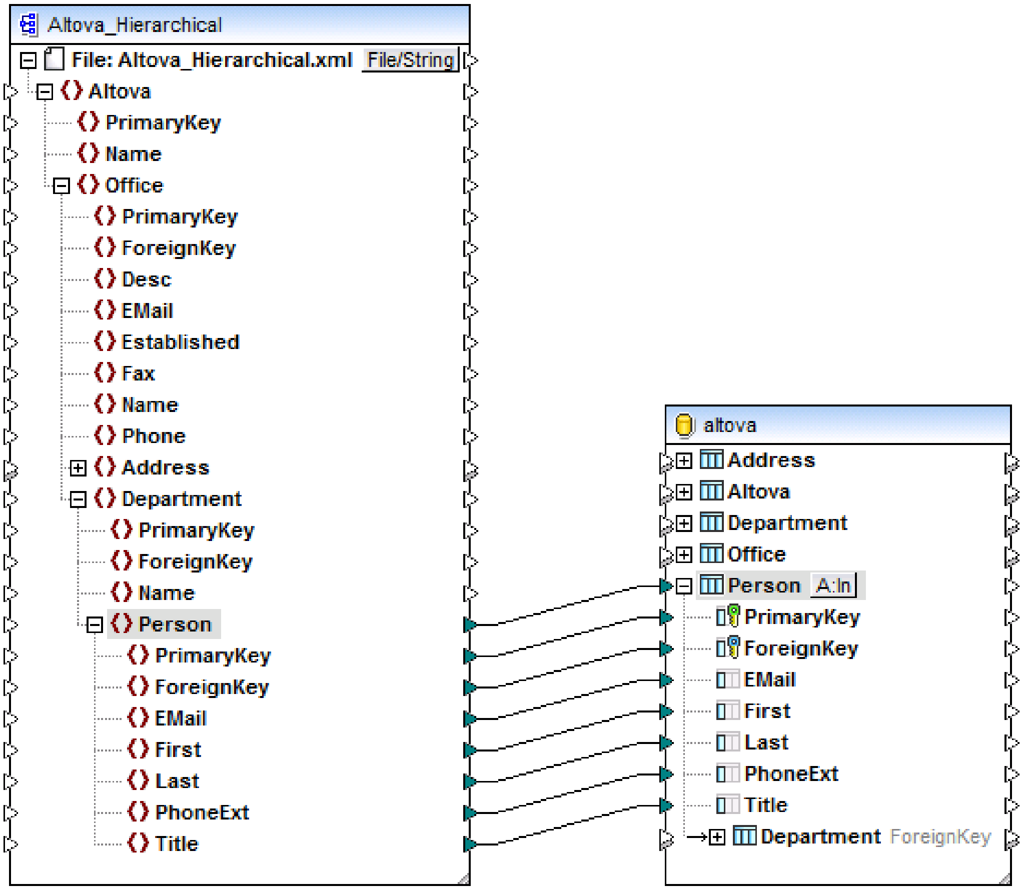
When prompted to supply an instance file, browse for **altova-cmpy.xml**.

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).

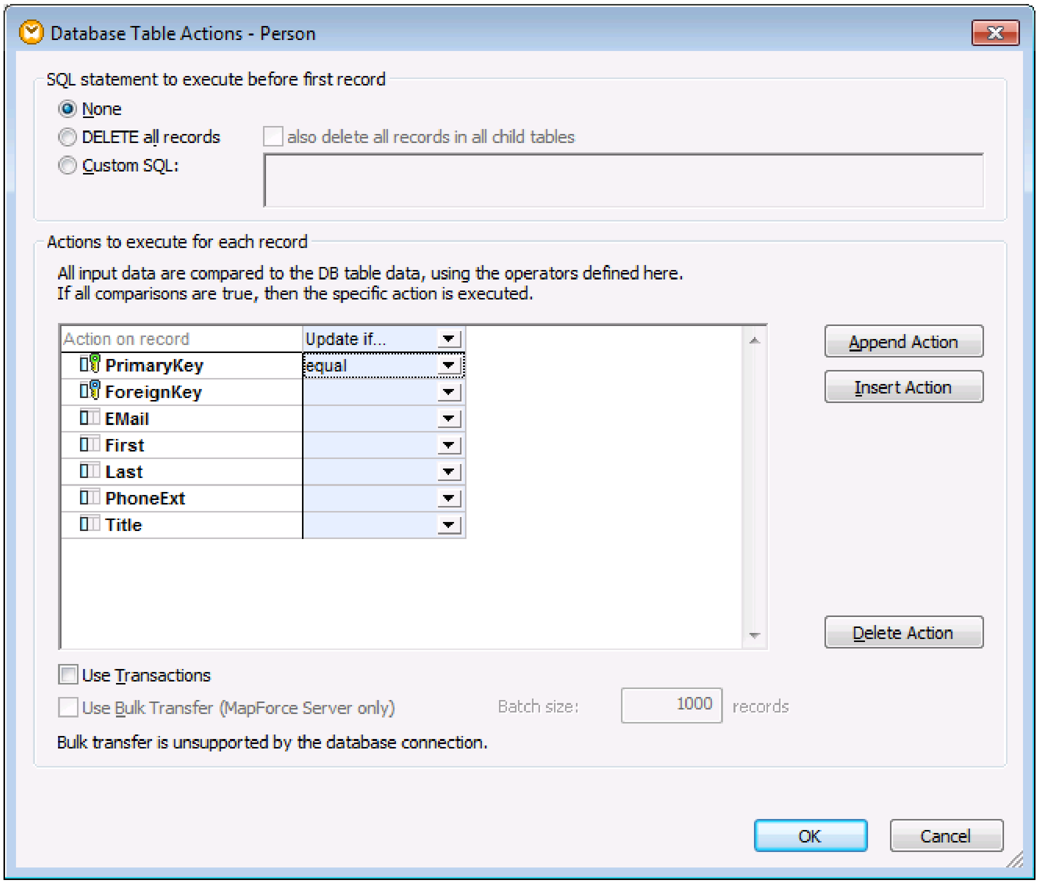
Step 3: Draw the connections

- 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.



SQL statement to execute before first record

● None

○ DELETE all records □ also delete all records in all child tables

○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Update if...
PrimaryKey	equal
ForeignKey	
EMail	
First	
Last	
PhoneExt	
Title	

□ Use Transactions

□ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records

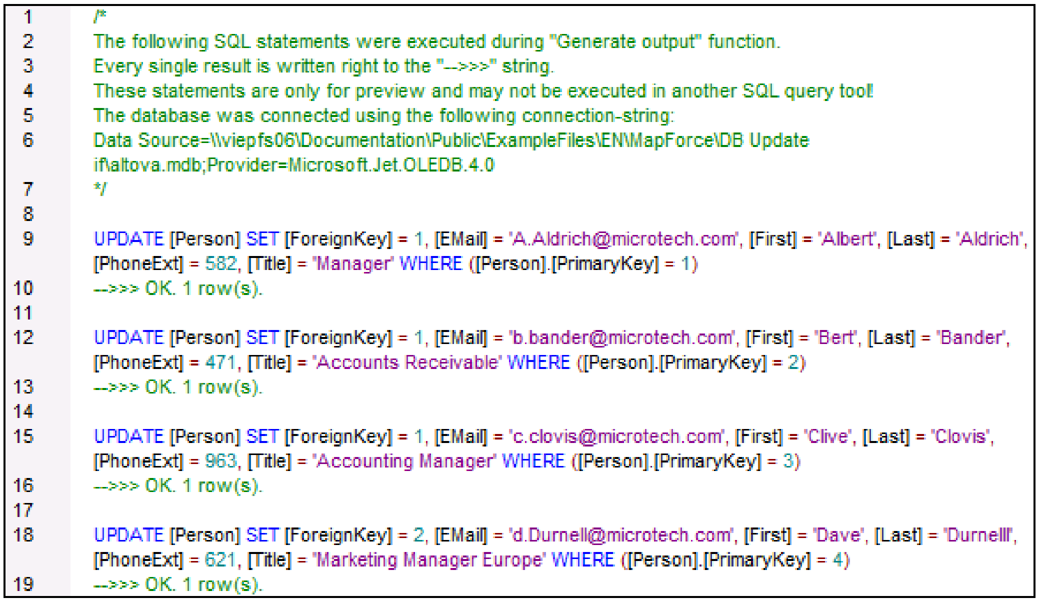
Bulk transfer is unsupported by the database connection.

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.

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.

7.2.3.4 "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:

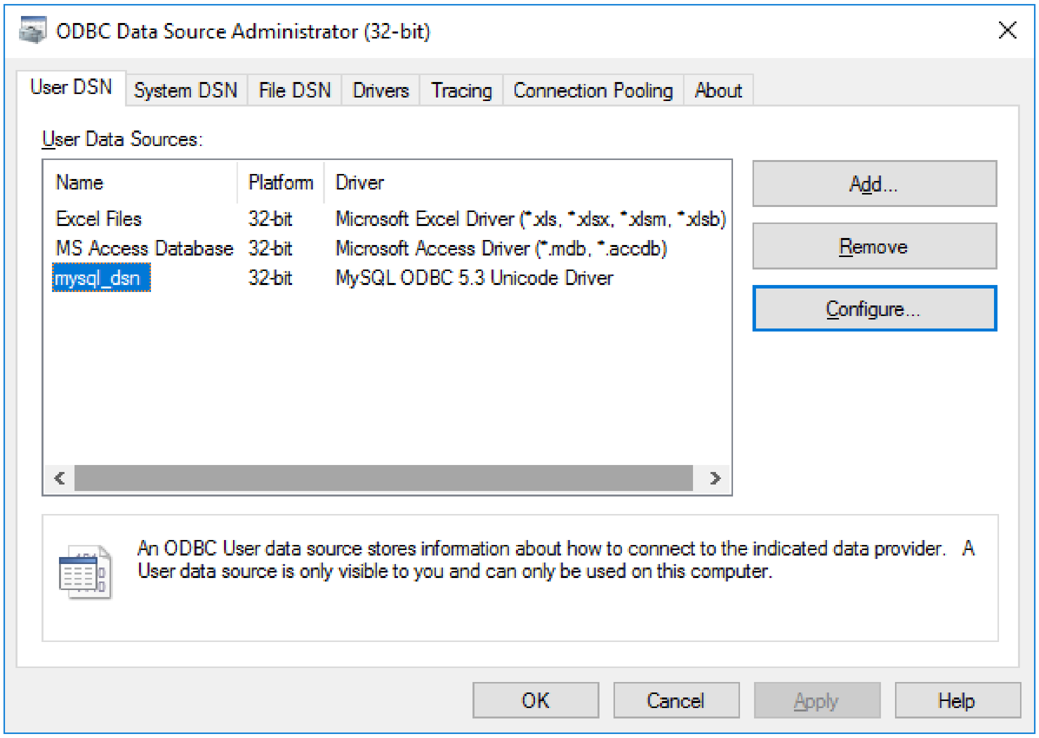
- If the **Update if** condition is true, then the existing database record is updated with data from the mapping.
- If the **Update if** condition is false, and an **Insert Rest** condition exists, then a new record is inserted.
- If records exist in the database with no counterpart in the source file, then these records remain unchanged.

MySQL ODBC note

> If the target database is MySQL 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 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**.



User Data Sources:

Name	Platform	Driver
Excel Files	32-bit	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)
MS Access Database	32-bit	Microsoft Access Driver (*.mdb, *.accdb)
mysql_dsn	32-bit	MySQL ODBC 5.3 Unicode Driver

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you and can only be used on this computer.

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\MapForce2018\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.

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

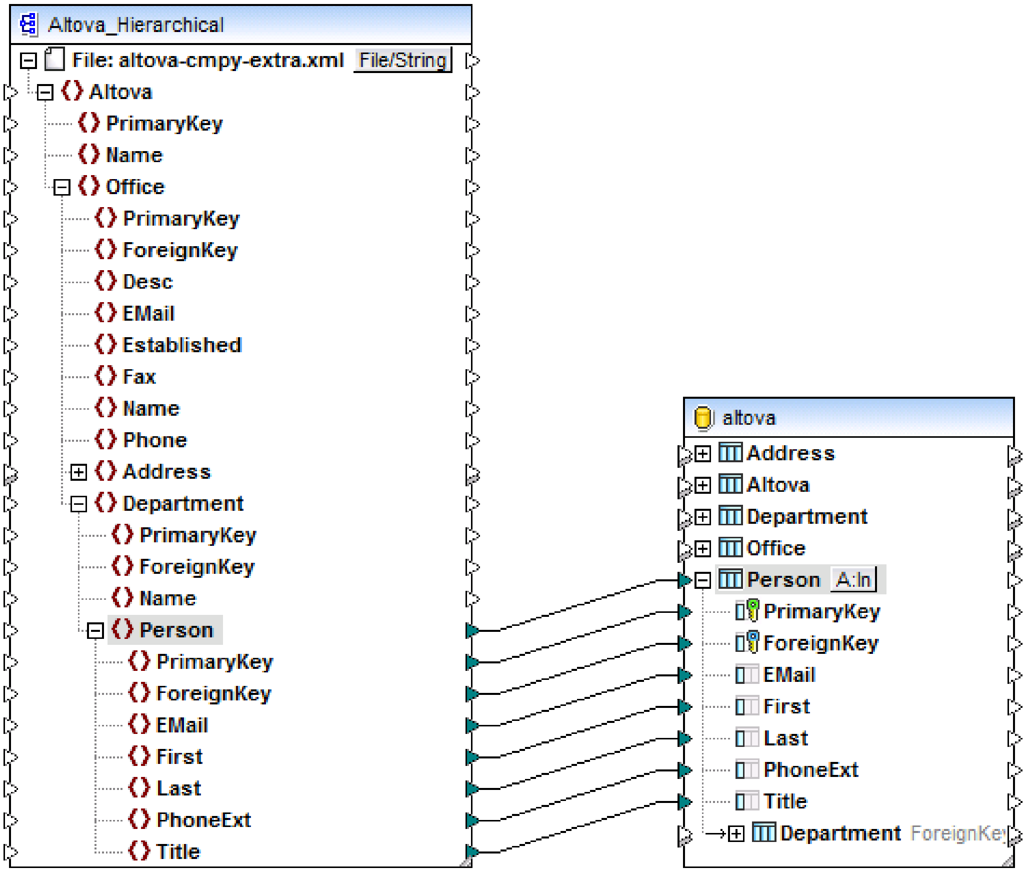
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **altova-cmpy-extra.xml**.

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).

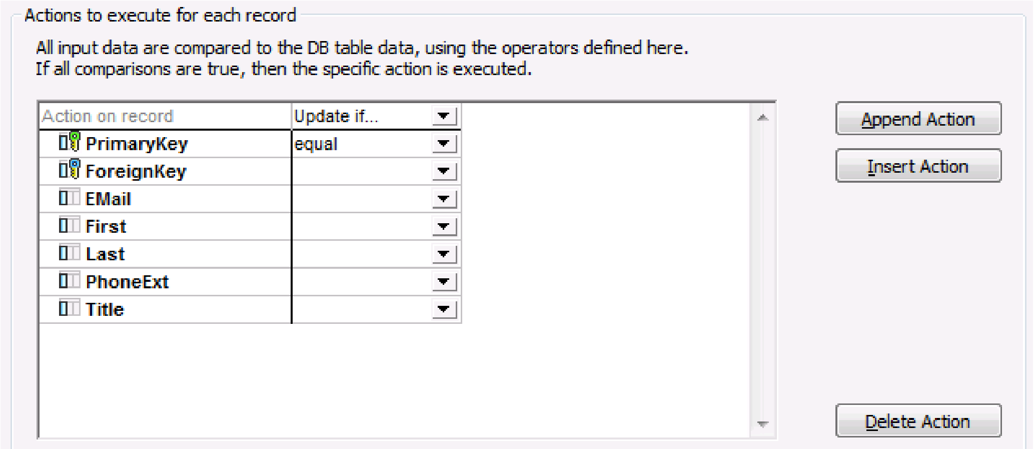
Step 3: Draw the connections

- 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.

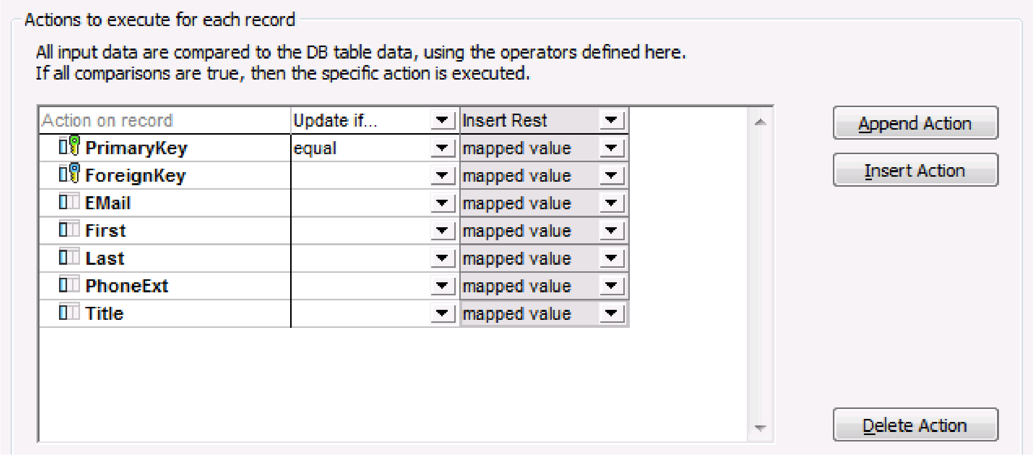


Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Update if...
PrimaryKey	equal
ForeignKey	
EMail	
First	
Last	
PhoneExt	
Title	

4. Click **Append Action**. This adds a new action to the right of the existing **Update If** action. Configure the new action as **Insert Rest**:



Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Update if...	Insert Rest
PrimaryKey	equal	mapped value
ForeignKey		mapped value
EMail		mapped value
First		mapped value
Last		mapped value
PhoneExt		mapped value
Title		mapped value

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.

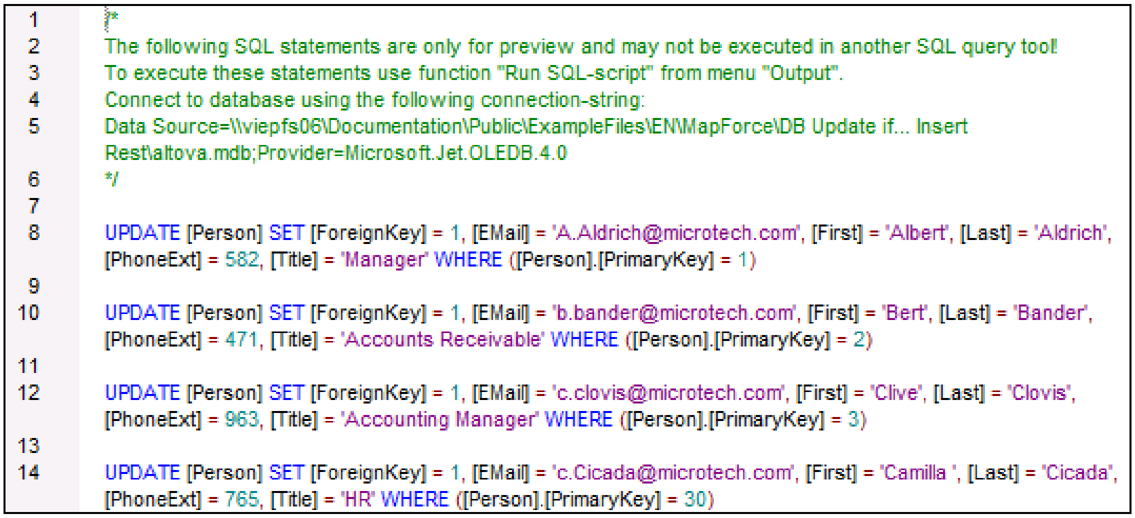
It is also possible to define more than two actions against the same database table (this is not necessary in this example, however). At mapping runtime, actions are executed from left to right. The last **Insert** action is considered final; any other actions added after it will be ignored.

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**.

5. 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

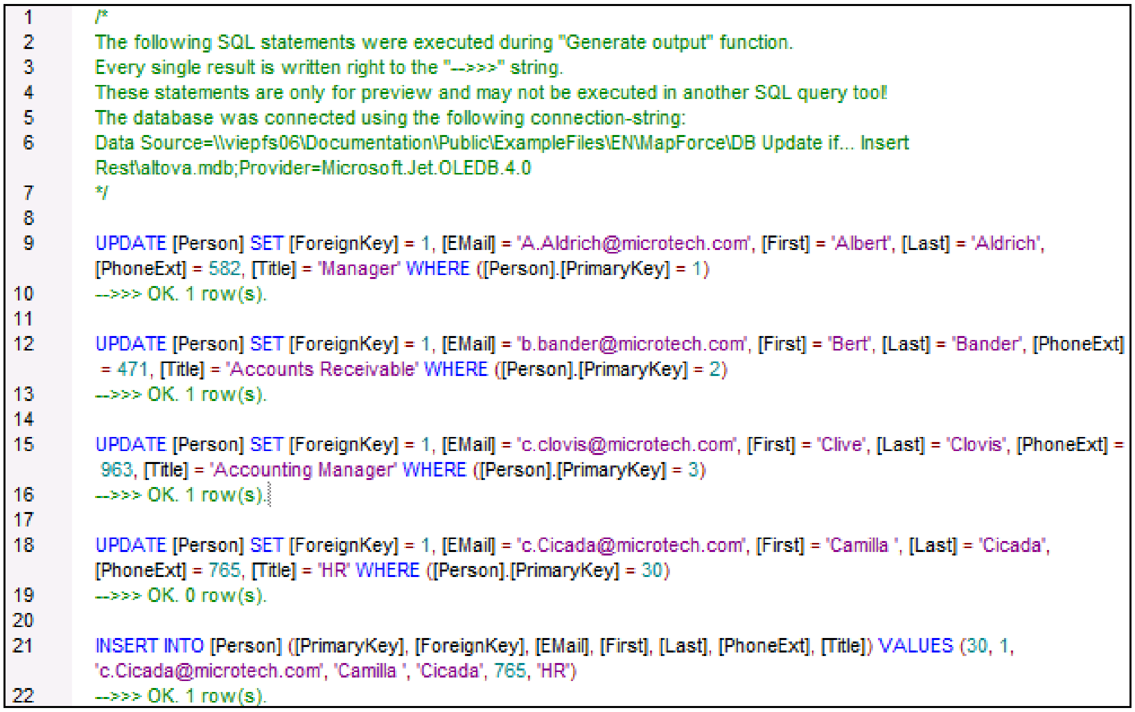
You may notice that no INSERT statements are visible in the preview script. This is normal behavior, because records are inserted conditionally, and the INSERT statements depend on the result of the **Update If** action (which is not known before the mapping runs).

Note: For certain database types, MapForce creates MERGE statements instead of UPDATE statements. For further information, see MERGE statements.

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.



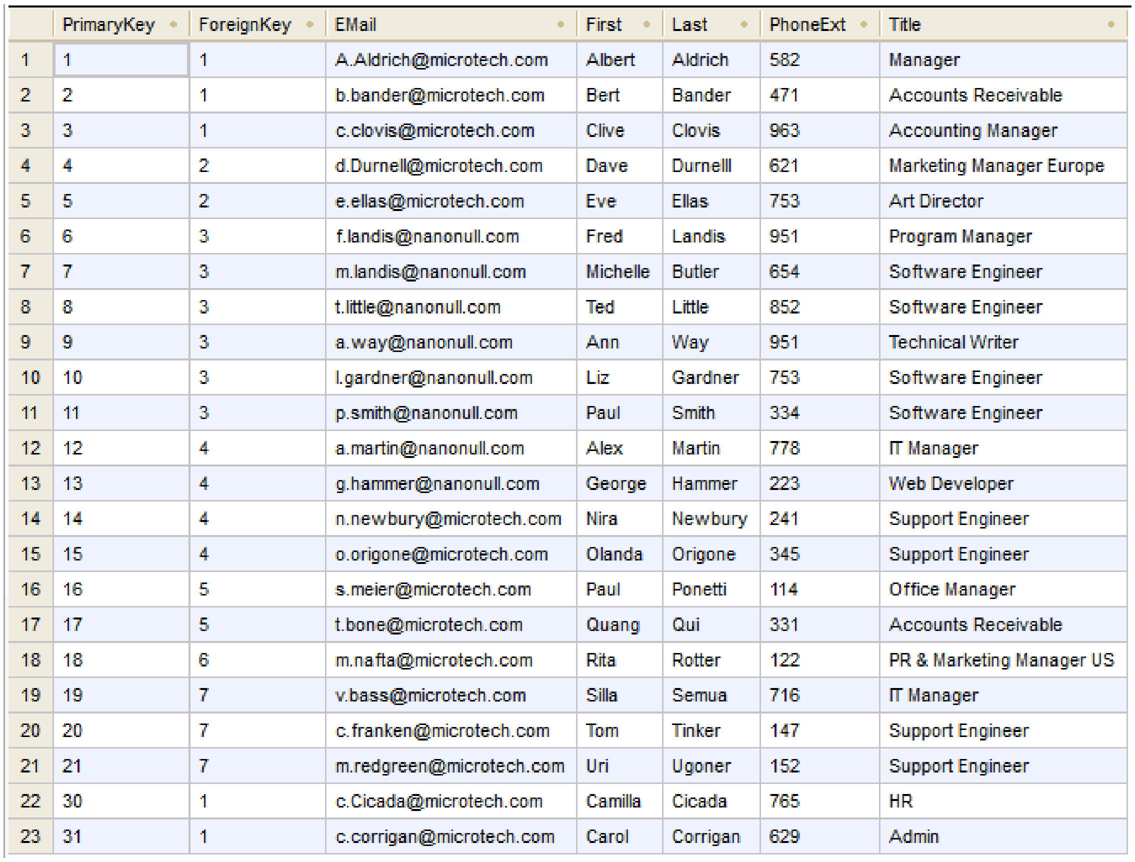
```
/*
The following SQL statements were executed during "Generate output" function.
Every single result is written right to the "-->>>" string.
These statements are only for preview and may not be executed in another SQL query tool!
The database was connected using the following connection-string:
Data 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.

If you open the "Person" table in the DB query tab of MapForce (see Browsing and Querying Databases), 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	Durnell	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

7.2.3.5 MERGE Statements

For certain mappings which both update and insert data into a database table (see also "Update if... Insert Rest" Action), 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.

MERGE statements are supported for the following database types:

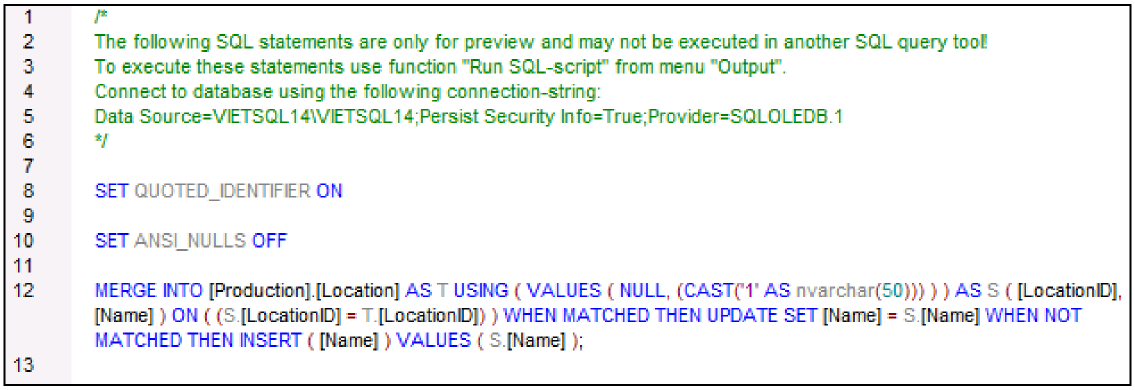
- SQL Server 2008 and later
- Oracle
- DB2
- Firebird

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. Note that 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.

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.
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=VIETSQL14\VIETSQL14;Persist Security Info=True;Provider=SQLOLEDB.1
*/

SET QUOTED_IDENTIFIER ON

SET ANSI_NULLS OFF

MERGE INTO [Production].[Location] AS T USING ( VALUES ( NULL, (CAST('1' AS nvarchar(50))) ) ) AS S ( [LocationID], [Name] ) ON ( (S.[LocationID] = T.[LocationID]) ) WHEN MATCHED THEN UPDATE SET [Name] = S.[Name] WHEN NOT MATCHED THEN INSERT ( [Name] ) VALUES ( S.[Name] );

```

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).

Limitations:

- With MERGE statements, the "Bulk Transfer" option (see Bulk Inserts (MapForce Server) is supported only for ODBC and JDBC database connections.

7.2.3.6 *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:

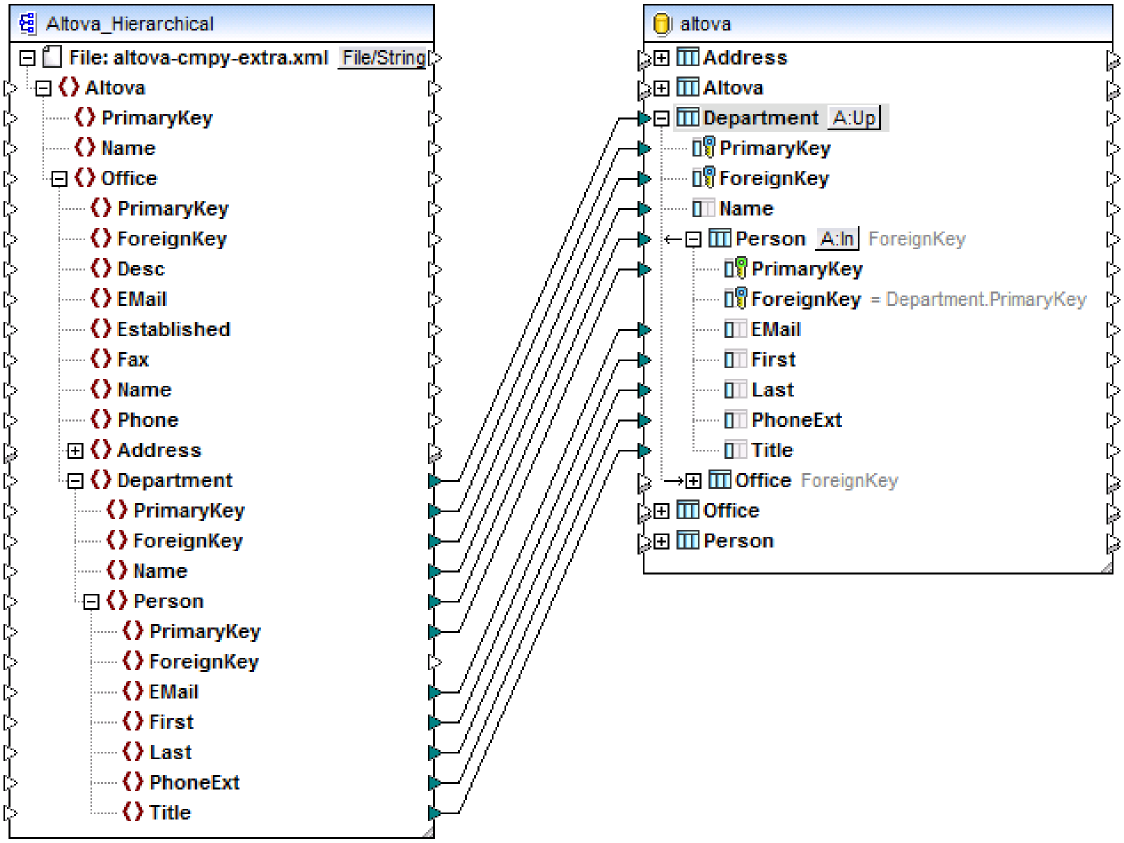
- **altova-cmpy-extra.xml** — contains the source data to be inserted into the database.
- **Altova_Hierarchical.xsd** — the schema used to validate the instance file above.
- **altova.mdb** — the target database to be updated.

All files are available in the **<Documents>\Altova\MapForce2018\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.

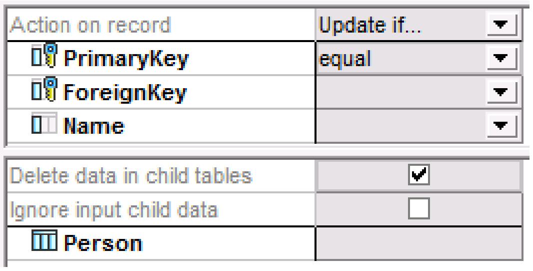
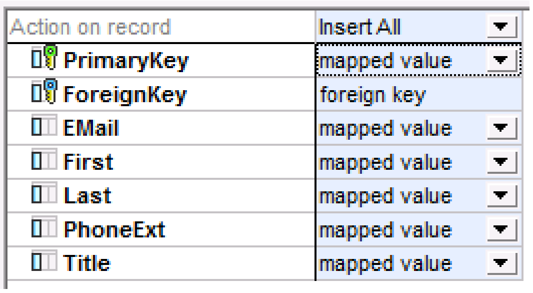
First, add the source XML schema and instance as well as the target database to the mapping (see Example: Adding the "altova.mdb" Database to the Mapping). Follow the same steps as in "Update if... Insert Rest" Action. Secondly, draw the mapping connections as shown below:



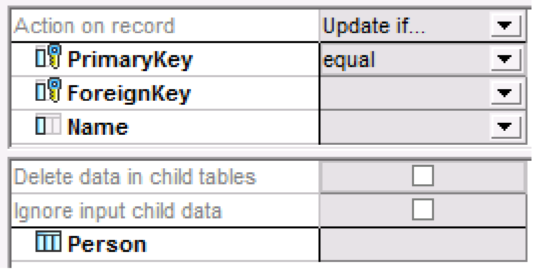
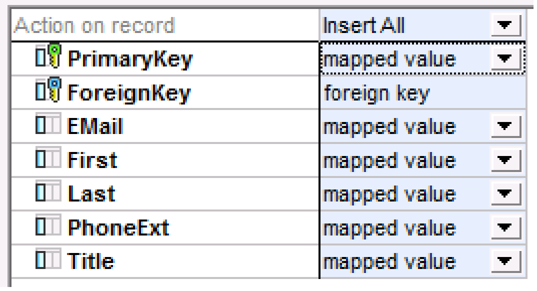
As illustrated above, the mapping updates the "Department" table in the target database. The "Department" table is chosen as "root" table. For more information about what a root table is and why it is necessary, see Handling Database Relationships. The action to be taken against the child "Person" table 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.

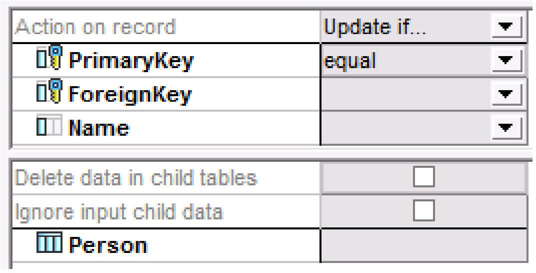
Configuration A

Settings	Mapping result
 Action on record: Update if... PrimaryKey: equal ForeignKey: Name: Delete data in child tables: ☑ Ignore input child data: ☐ Person *"Department" table*  Action on record: Insert All PrimaryKey: mapped value ForeignKey: foreign key EMail: mapped value First: mapped value Last: mapped value PhoneExt: mapped value Title: mapped value *"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). • 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.

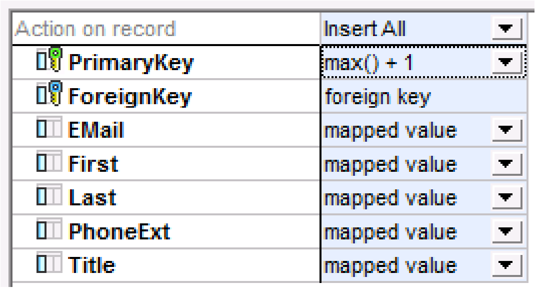
Configuration B

Settings	Mapping result
 Action on record: Update if... PrimaryKey: equal ForeignKey: Name: Delete data in child tables: □ Ignore input child data: □ Person *"Department" table*  Action on record: Insert All PrimaryKey: mapped value ForeignKey: foreign key EMail: mapped value First: mapped value Last: mapped value PhoneExt: mapped value Title: mapped value *"Person" table*	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.

Configuration C

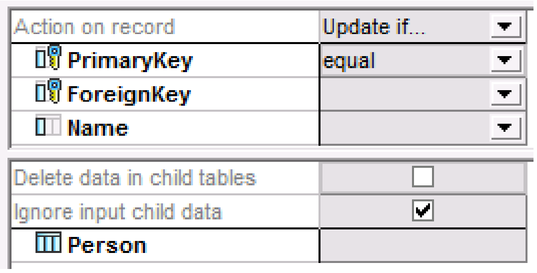
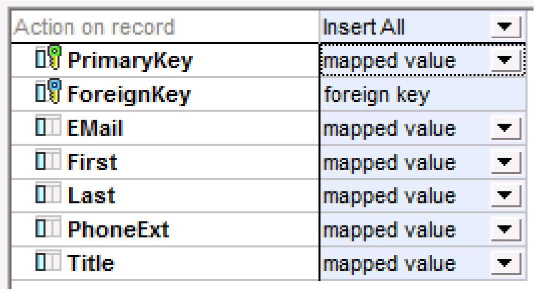
<table>
<tr><th>Settings</th><th>Mapping result</th></tr>
<tr><td>
Action on record: Update if...
PrimaryKey: equal
ForeignKey:
Name:
Delete data in child tables: □
Ignore input child data: □
Person

<i>"Department" table</i>


Action on record: Insert All
PrimaryKey: max() + 1
ForeignKey: foreign key
EMail: mapped value
First: mapped value
Last: mapped value
PhoneExt: mapped value
Title: mapped value

<i>"Person" table</i></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>

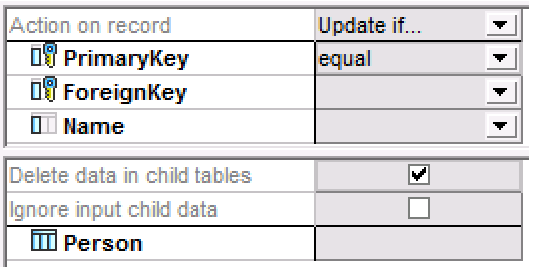
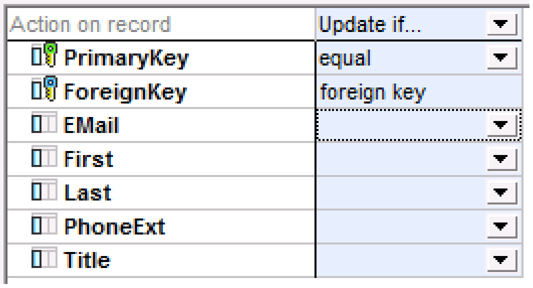
Configuration D

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). - No records are inserted in the Person table because the option **Ignore input child data** is enabled for the parent Departments table.

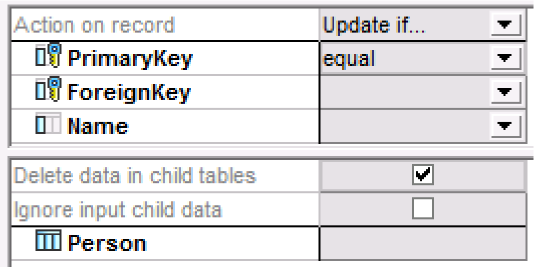
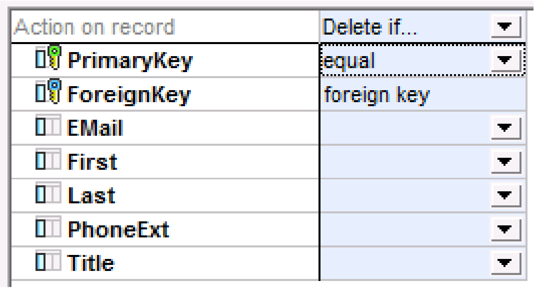
Action on record	Update if...
PrimaryKey	equal
ForeignKey	
Name	
Delete data in child tables	☐
Ignore input child data	☑
Person	

Action on record	Insert All
PrimaryKey	mapped value
ForeignKey	foreign key
EMail	mapped value
First	mapped value
Last	mapped value
PhoneExt	mapped value
Title	mapped value

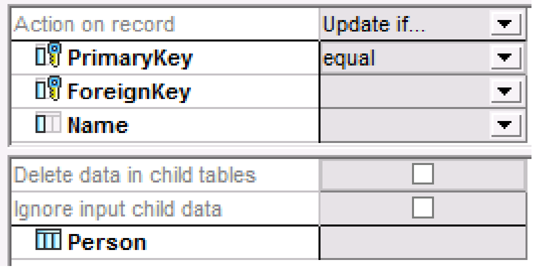
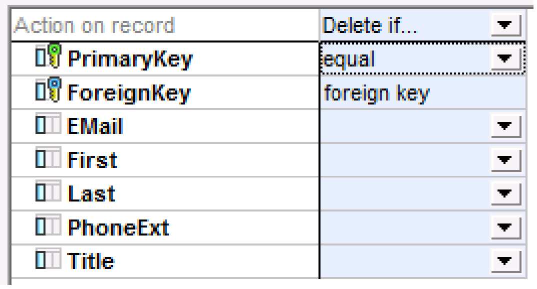
Configuration E

Settings	Mapping result
 Action on record: Update if... PrimaryKey: equal ForeignKey: Name: Delete data in child tables: ☑ Ignore input child data: ☐ Person *"Department" table*  Action on record: Update if... PrimaryKey: equal ForeignKey: foreign key EMail: First: Last: PhoneExt: Title: *"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

Settings	Mapping result
 Action on record Update if... PrimaryKey equal ForeignKey Name Delete data in child tables ☑ Ignore input child data ☐ Person *"Department" table*  Action on record Delete if... PrimaryKey equal ForeignKey foreign key EMail First Last PhoneExt Title *"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.

Configuration G

Settings	Mapping result
 Action on record: Update if... PrimaryKey: equal ForeignKey: Name: Delete data in child tables: □ Ignore input child data: □ Person *"Department" table*  Action on record: Delete if... PrimaryKey: equal ForeignKey: foreign key EMail: First: Last: PhoneExt: Title: *"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 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.

7.2.3.7 *"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). 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\MapForce2018\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.

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

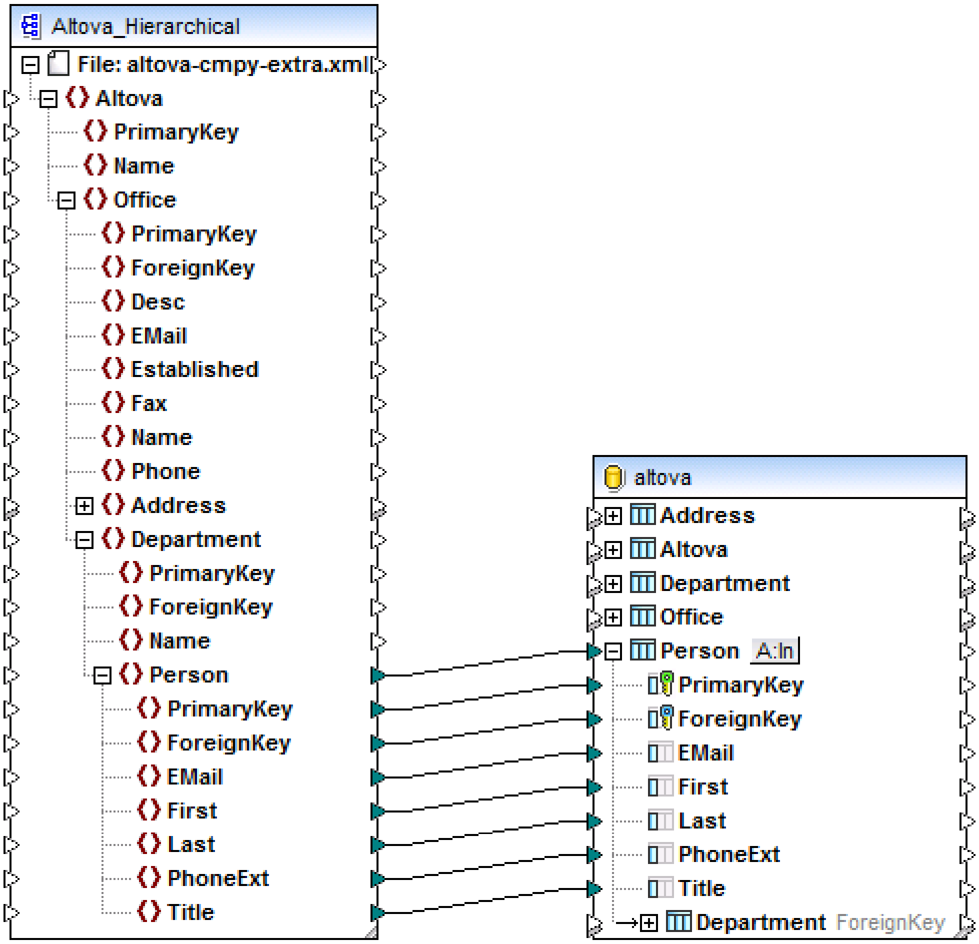
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **altova-cmpy-extra.xml**.

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).

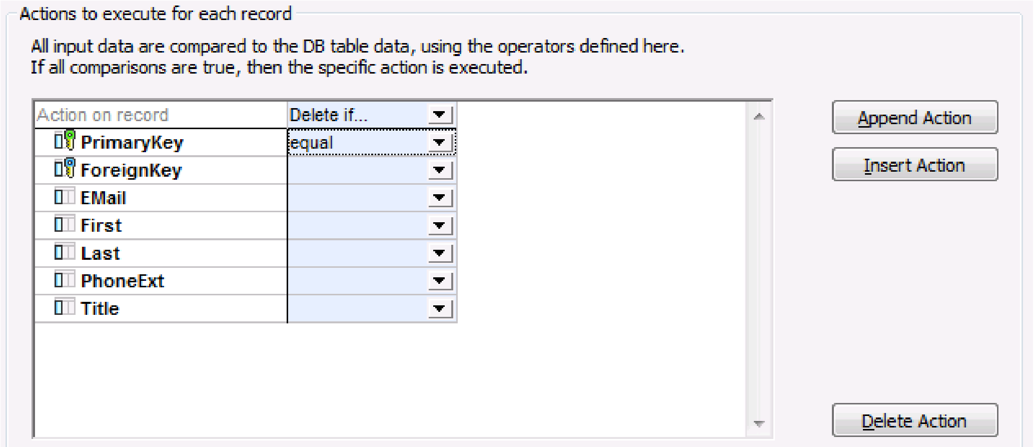
Step 3: Draw the connections

- 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.

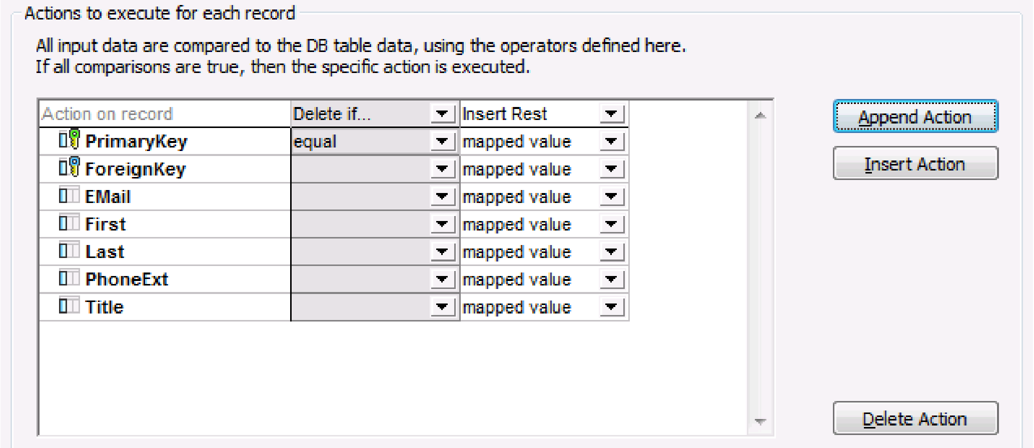


Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Delete if...
PrimaryKey	equal
ForeignKey	
EMail	
First	
Last	
PhoneExt	
Title	

4. Click **Append Action**. This adds a new action to the right of the existing **Delete If** action. Configure the new action as **Insert Rest**:



Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Delete if...	Insert Rest
PrimaryKey	equal	mapped value
ForeignKey		mapped value
EMail		mapped value
First		mapped value
Last		mapped value
PhoneExt		mapped value
Title		mapped value

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.

5. 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: Delete; Insert** (A:De,In) 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

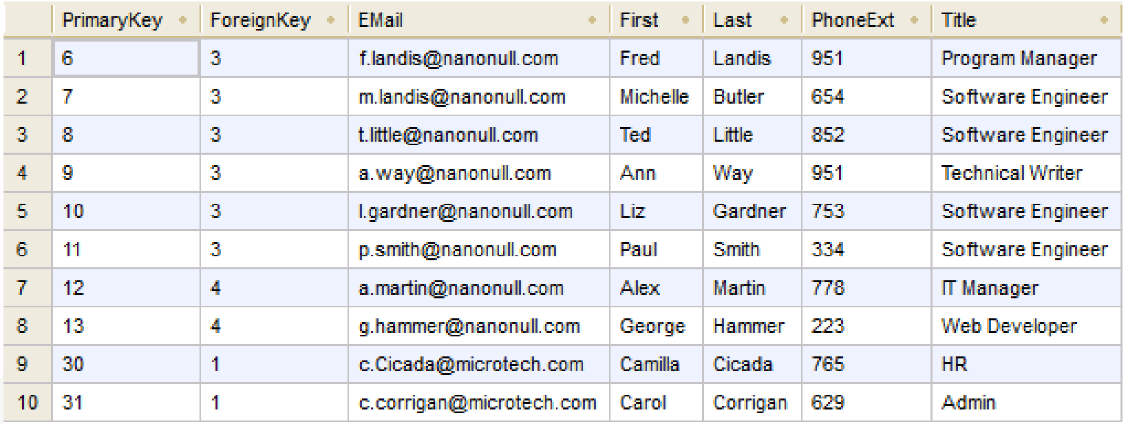
Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview. To run the script against the database:

- 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.

If you open the "Person" table in the DB query tab of MapForce (see Browsing and Querying Databases), 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

7.2.3.8 "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\MapForce2018\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.

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

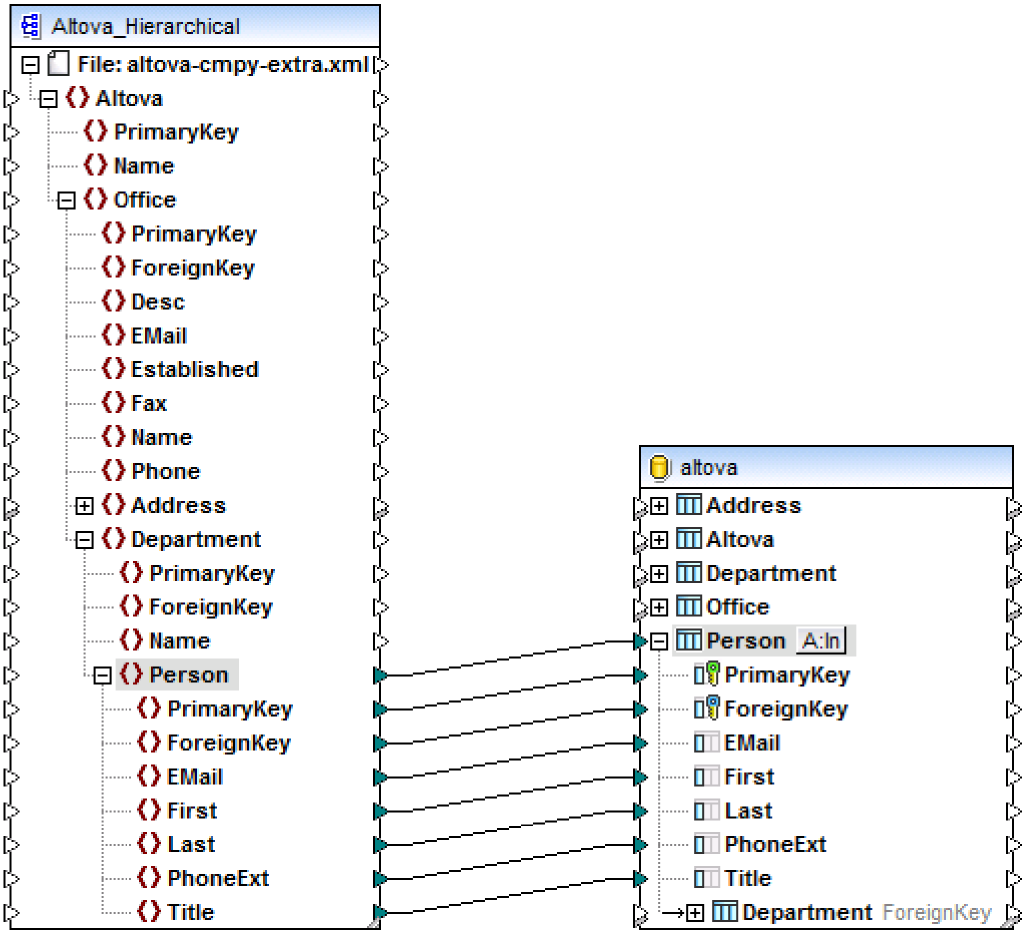
- On the **Insert** menu, click **XML Schema/File**, and browse for **Altova_Hierarchical.xsd**. When prompted to supply an instance file, browse for **altova-cmpy-extra.xml**.

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).

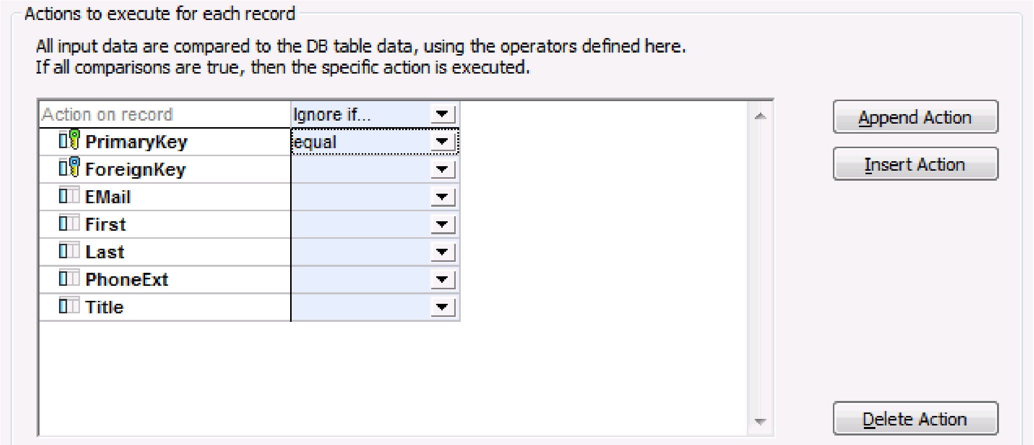
Step 3: Draw the connections

- 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.

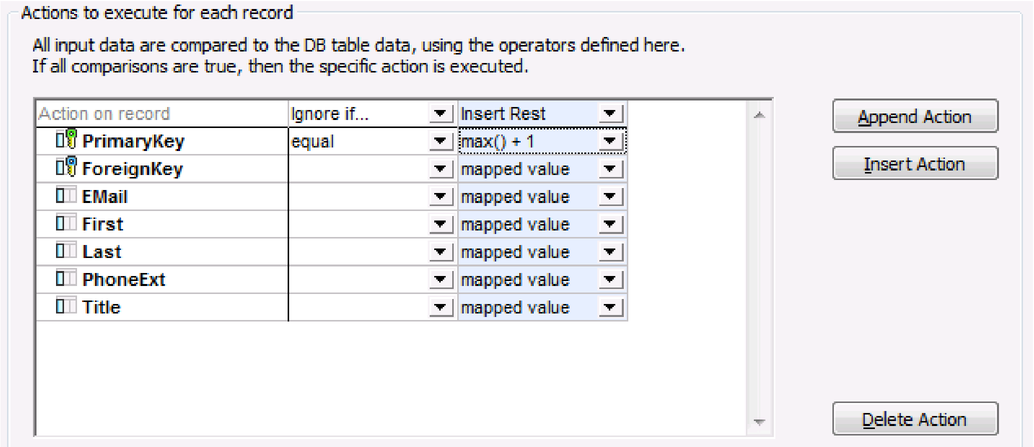


Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Ignore if...
PrimaryKey	equal
ForeignKey	
EMail	
First	
Last	
PhoneExt	
Title	

4. 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:



Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Ignore if...	Insert Rest
PrimaryKey	equal	max() + 1
ForeignKey		mapped value
EMail		mapped value
First		mapped value
Last		mapped value
PhoneExt		mapped value
Title		mapped value

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.

5. 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

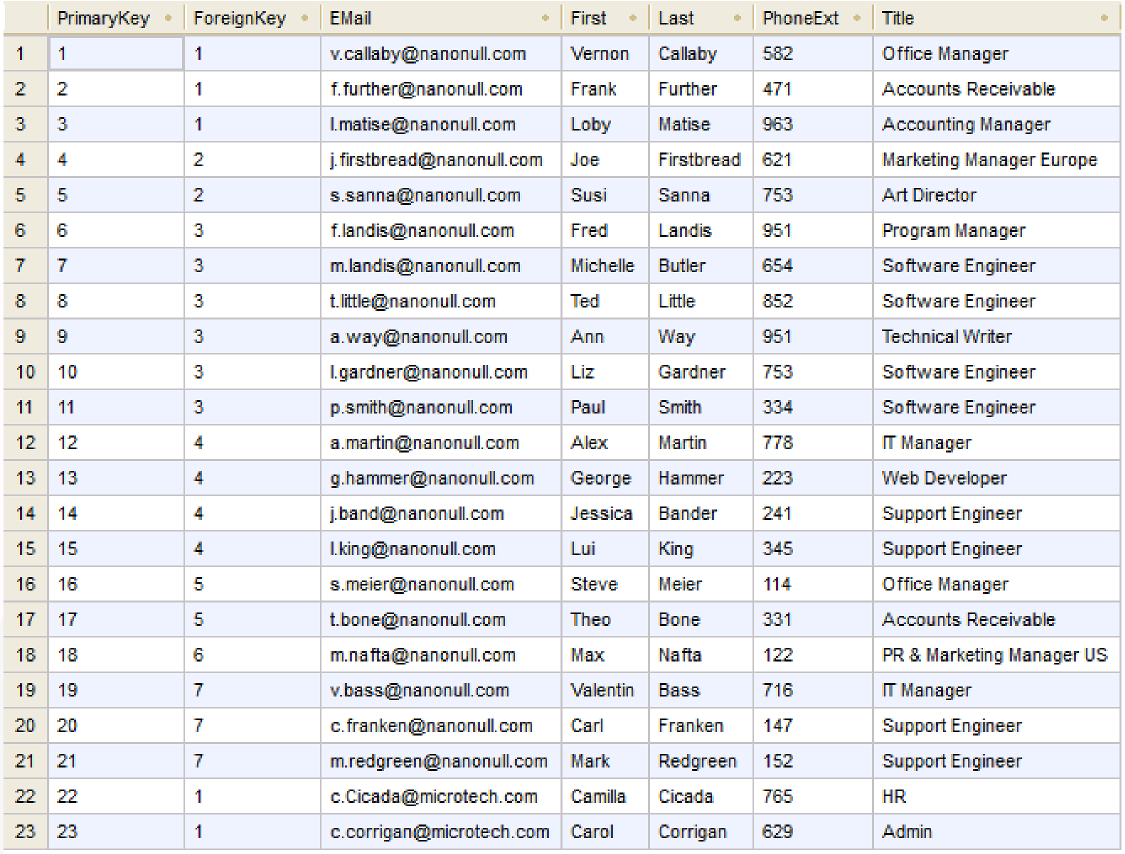
Click the **Output** tab to preview the mapping. A SQL script is generated, containing actions to be executed against the database. The script has not modified the database yet; it is only for preview. To run the script against the database:

- 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.

If you open the "Person" table in the DB query tab of MapForce (see Browsing and Querying Databases), 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.

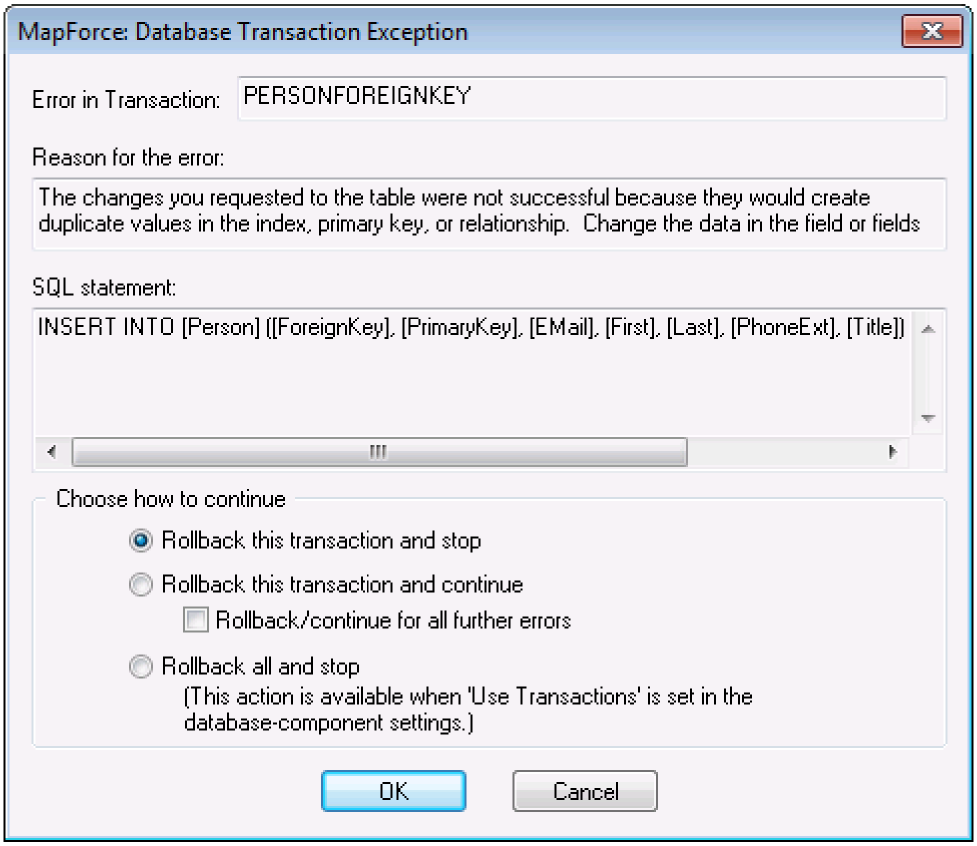


	PrimaryKey	ForeignKey	EMail	First	Last	PhoneExt	Title
1	1	1	v.callaby@nanonull.com	Vernon	Callaby	582	Office Manager
2	2	1	f.further@nanonull.com	Frank	Further	471	Accounts Receivable
3	3	1	l.matise@nanonull.com	Loby	Matise	963	Accounting Manager
4	4	2	j.firstbread@nanonull.com	Joe	Firstbread	621	Marketing 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

7.2.3.9 Using Transaction Rollback

Transaction rollback is a feature that enables you to decide what should happen when a database exception occurs for whatever reason. Namely, when you attempt to run a SQL script from MapForce, and an error occurs, a dialog box such as the one below opens, prompting you to choose how to continue. Because there might be SQL statements (transactions) that were already executed successfully before the exception occurred, you can choose to roll back (undo) all previously executed transactions, or only the current one.



MapForce: Database Transaction Exception

Error in Transaction: PERSONFOREIGNKEY

Reason for the error:

The changes you requested to the table were not successful because they would create duplicate values in the index, primary key, or relationship. Change the data in the field or fields

SQL statement:

```
INSERT INTO [Person] ([ForeignKey], [PrimaryKey], [EMail], [First], [Last], [PhoneExt], [Title])
```

Choose how to continue

● Rollback this transaction and stop

○ Rollback this transaction and continue

□ Rollback/continue for all further errors

○ Rollback all and stop

(This action is available when 'Use Transactions' is set in the database-component settings.)

Database Transaction Exception dialog box

To be able to roll back unsuccessful database actions, you must enable transactions first. You can enable database transactions as follows:

- At database level. To enable transactions at database level, select the **Use transactions** check box from the Database Component Settings dialog box (see also Database Component Settings).
- For each individual table. To enable transactions for each individual table, select the **Use transactions** check box from the Database Table Actions dialog box (see also Database Table Actions Settings).

You can enable transactions at database level without enabling them at table level, and vice versa. Be aware that, when a database exception occurs, the available rollback options depend on how transactions were enabled:

- If transactions were not enabled at all, and an error occurs, execution stops at the point the error occurs. All previously successful SQL statements are executed and the results are stored in the database. It is not possible to roll back any transactions.
- If transactions were enabled at table level but not at database level, execution stops at the point the error occurs. In this case, the option **Rollback all and stop** is disabled. You can roll back only the current transaction for that specific table, and either continue

with the mapping execution or stop running the mapping completely.

- If transactions were enabled at the database level only, execution stops at the point the error occurs. All previously successful SQL statements are rolled back. No changes are made to the database.
- If transactions were enabled at both the database level and table level, execution stops at the point the error occurs. In this case, select **Rollback all and stop** to roll back all previously successful SQL statements for the entire database. To roll back only the last transaction, and continue executing the mapping, select **Rollback this transaction and continue**.

Note that the Database Transaction Exception dialog box is displayed only when you run the mapping directly from MapForce, using the **SQL | Run SQL-Script** menu command. When the mapping is executed in another environment (either by the generated code, or by MapForce Server), and an error is encountered, an automatic rollback of the erroneous transaction occurs. In this case, any successful SQL statements that were previously executed are rolled back only if you enabled transactions at the database level, as explained above.

If you click **Cancel** on the dialog box, the execution rolls back the current SQL statement and stops.

7.2.3.10 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 Selecting a Transformation Language.
- 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 and Deploying Mappings to FlowForce Server.
- 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.
- 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	Yes	n/a
MySQL	n/a	Yes*	Yes	Yes	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
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.*

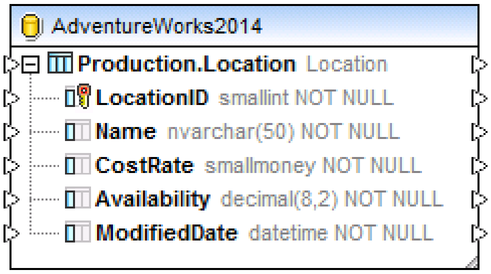
This example shows you how create a mapping which bulk loads data from a sample **source.txt** file into a target database. The example uses SQL Server 2014 and the AdventureWorks 2014 database. The latter can be downloaded from the CodePlex website (https://msftdbprodsamples.codeplex.com/).

```
Location A,15.3,39
Location B,46,34
Location C,56.33,0
Location D,0,399
Location E,0,97.43
```

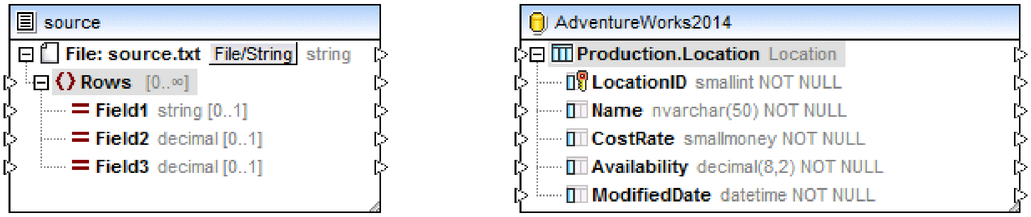
source.txt

To define a bulk insert:

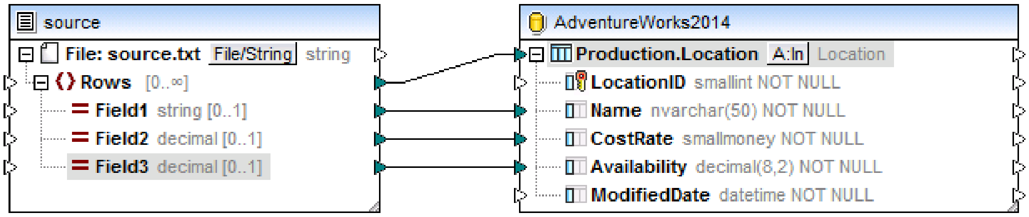
1. Set the transformation language to BUILT-IN (BUILT IN).
2. Connect to the AdventureWorks 2014 database and add the "Production.Location" table to the mapping. For more information, see Adding Databases to the Mapping.


AdventureWorks2014
Production.Location Location
LocationID smallint NOT NULL
Name nvarchar(50) NOT NULL
CostRate smallmoney NOT NULL
Availability decimal(8,2) NOT NULL
ModifiedDate datetime NOT NULL

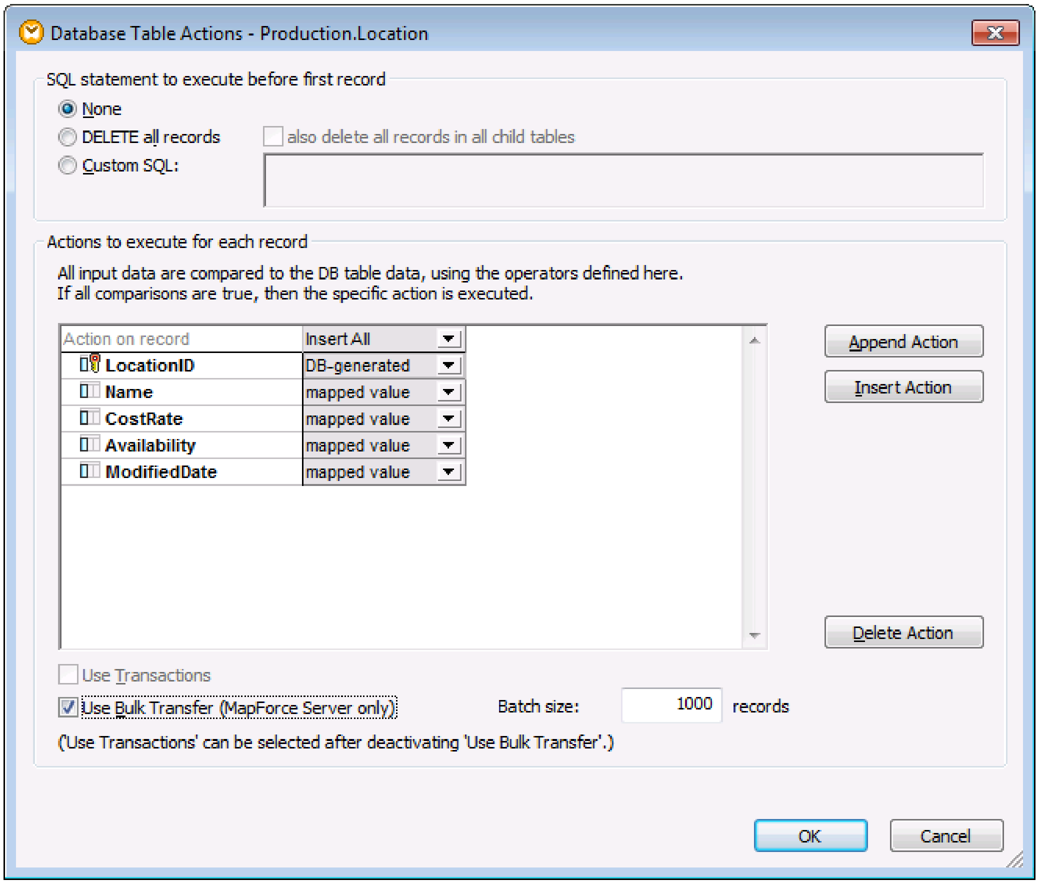
3. On the **Insert** menu, click **Text**, and add a source text file (such as the **source.txt** sample above) to the mapping. For more information, see CSV and Text Files. Make sure that the data types of both the source and the target components are compatible. Data types are visible on component when the **Show Data Types** ( ) toolbar button must be enabled.


source
File: source.txt File/String string
Rows [0..∞]
Field1 string [0..1]
Field2 decimal [0..1]
Field3 decimal [0..1]
AdventureWorks2014
Production.Location Location
LocationID smallint NOT NULL
Name nvarchar(50) NOT NULL
CostRate smallmoney NOT NULL
Availability decimal(8,2) NOT NULL
ModifiedDate datetime NOT NULL

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.


source
File: source.txt File/String string
Rows [0..∞]
Field1 string [0..1]
Field2 decimal [0..1]
Field3 decimal [0..1]
AdventureWorks2014
Production.Location A:In Location
LocationID smallint NOT NULL
Name nvarchar(50) NOT NULL
CostRate smallmoney NOT NULL
Availability decimal(8,2) NOT NULL
ModifiedDate datetime NOT NULL

5. Click the **Database Actions** button (A:In), select the **Use Bulk Transfer** check box, and click OK to confirm.



Database Table Actions - Production.Location

SQL statement to execute before first record

● None

○ DELETE all records □ also delete all records in all child tables

○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on record	Insert All
LocationID	DB-generated
Name	mapped value
CostRate	mapped value
Availability	mapped value
ModifiedDate	mapped value

□ Use Transactions

☑ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records

('Use Transactions' can be selected after deactivating 'Use Bulk Transfer'.)

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 and Deploying Mappings to FlowForce Server.

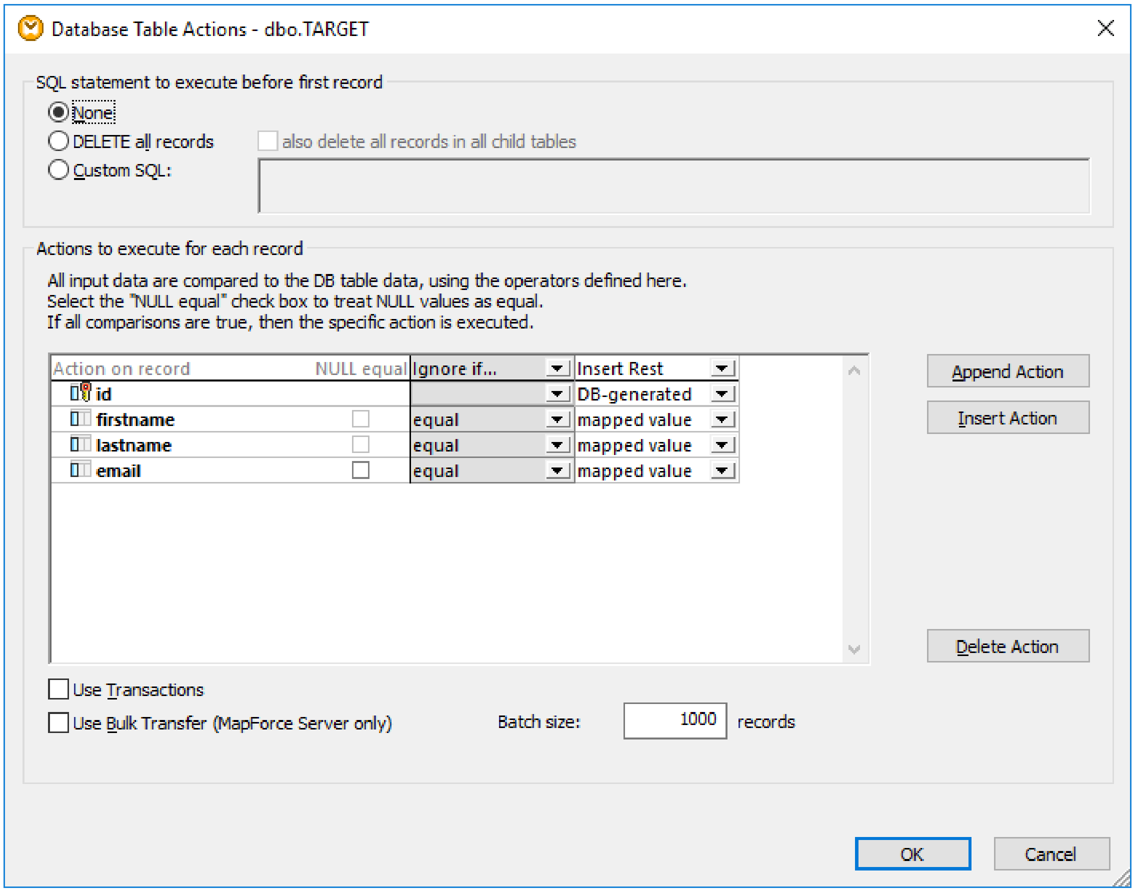
7.2.3.11 Handling Nulls in Database Table Actions

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). The generated queries reflect the comparison conditions that were defined from the "Database Table Actions" dialog box (see also Database Table Actions Settings).

Null comparisons are a complex subject in the context of SQL and databases, in the sense that there is no commonly accepted way to compare null values across various database types. From a MapForce perspective, it is possible to configure a database mapping so that data comparison is done in a NULL-aware manner, according to rules applicable to the database kind involved in the mapping. "NULL-awareness" means that any `NULL` values will be treated as such for the scope of data comparison (otherwise, you may get undesired results from the mapping). NULL-awareness should be enabled if:

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. NULL values in the source table must be treated as equal with NULL values in the target table.

By default, NULL-awareness is disabled. If the conditions above are true and NULL-awareness is disabled, there may be instances where the target database table is not updated as expected (for example, more rows are inserted or updated than necessary). This happens because `NULL` values affect the data comparison and could produce undesired results. To prevent this from happening, select the check box next to each nullable field (`email`, in the image below) from the "Database Table Actions" dialog box. 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.



SQL statement to execute before first record

● None
○ DELETE all records □ also delete all records in all child tables
○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
Select the "NULL equal" check box to treat NULL values as equal.
If all comparisons are true, then the specific action is executed.

Action on record	NULL equal	Ignore if...	Insert Rest
id			DB-generated
firstname	□	equal	mapped value
lastname	□	equal	mapped value
email	□	equal	mapped value

□ Use Transactions
□ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records

Database Table Actions dialog box

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\MapForce2018\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

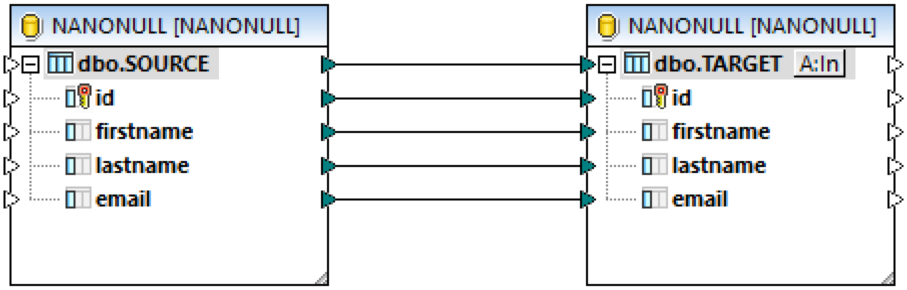
```
+----+-----------+-----------+-------------------------+
| id | firstname | lastname  | email                   |
+----+-----------+-----------+-------------------------+
| 1  | Mia       | Dahill    | NULL                    |
| 2  | Fred      | Weinstein | f.weinstein@nanonull.com |
+----+-----------+-----------+-------------------------+
```

The TARGET table

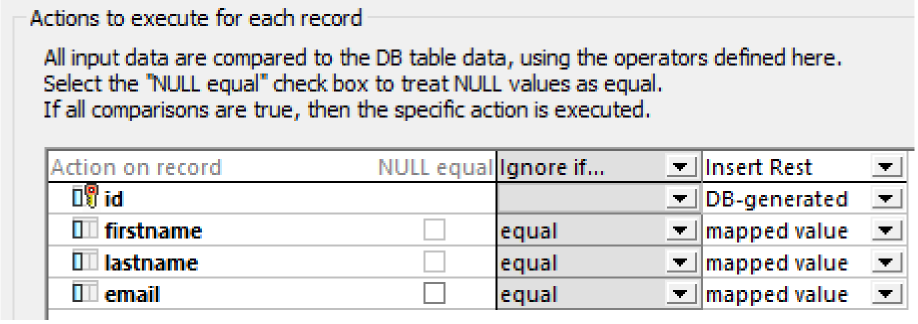
Let's suppose your task is to merge data from the `SOURCE` table into the `TARGET` table. Only the new records must be inserted into the `TARGET` table (in this example, "Tobie Hughey"). The records which exist in both tables ("Mia Dahill" and "Fred Weinstein") must be ignored.

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). 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:



Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
Select the "NULL equal" check box to treat NULL values as equal.
If all comparisons are true, then the specific action is executed.

Action on record	NULL equal	Ignore if...	Insert Rest
id			DB-generated
firstname	□	equal	mapped value
lastname	□	equal	mapped value
email	□	equal	mapped value

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 fields 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;
```

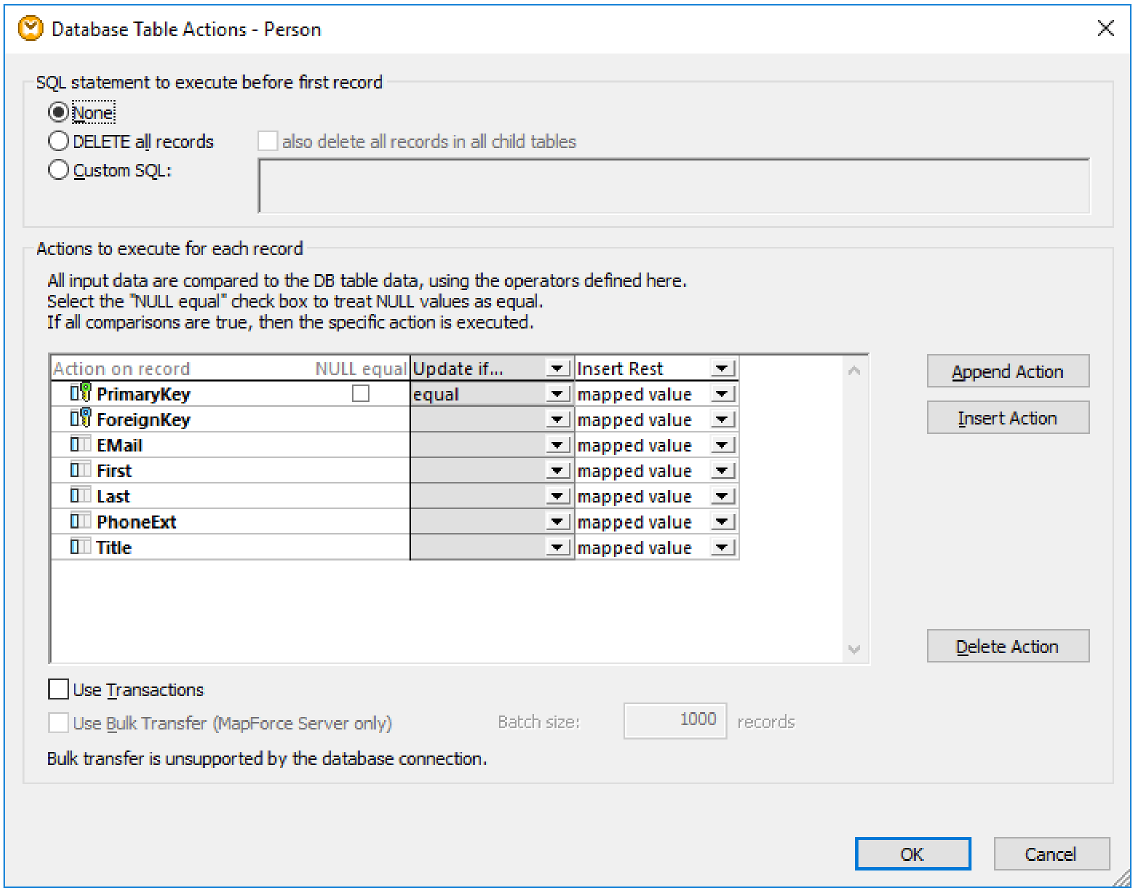
In order to be NULL aware, the query above would have to be rewritten as follows:

```
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. When NULL awareness is enabled, MapForce adapts the syntax of generated queries according to the database type (since various database vendors have different approaches to handling null comparisons).

7.2.3.12 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.



SQL statement to execute before first record

● None
○ DELETE all records □ also delete all records in all child tables
○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
Select the "NULL equal" check box to treat NULL values as equal.
If all comparisons are true, then the specific action is executed.

Action on record	NULL equal	Update if...	Insert Rest
PrimaryKey	□	equal	mapped value
ForeignKey			mapped value
EMail			mapped value
First			mapped value
Last			mapped value
PhoneExt			mapped value
Title			mapped value

□ Use Transactions
□ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records
Bulk transfer is unsupported by the database connection.

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.
- **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.

In the "NULL Equal" column, the check box next to each record, where applicable, enables you to explicitly instruct MapForce that the column may contain NULL values and should be treated as such (see also Handling Nulls in Database Table Actions).

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.
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.

For examples which illustrate various combinations of actions, see:

- Inserting Data into a Table
- Inserting Data into Multiple Linked Tables
- Updating a Table
- Options for Child Tables When Updating a Parent Table
- "Update if... Insert Rest" Action
- "Delete if..." Action
- "Ignore if..." Action

Use Transactions

Enables database transactions for this particular table action. For more information, see Using Transaction Rollback.

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).

7.2.3.13 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\MapForce2018\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:

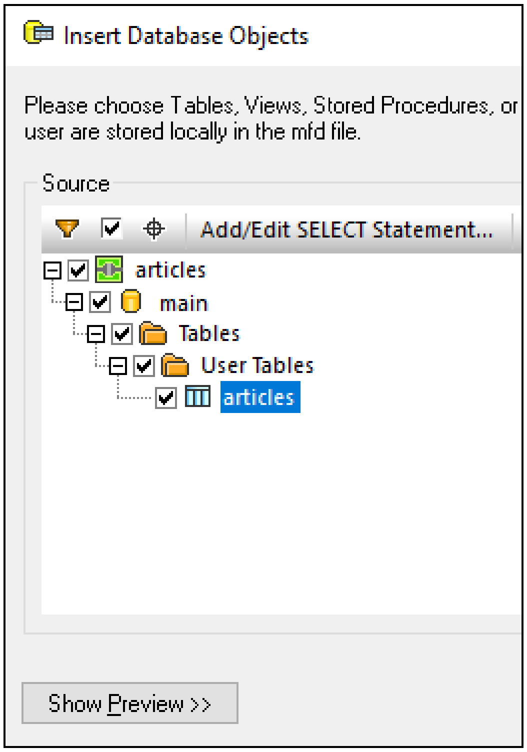
```
.save articles.sqlite
```

   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.

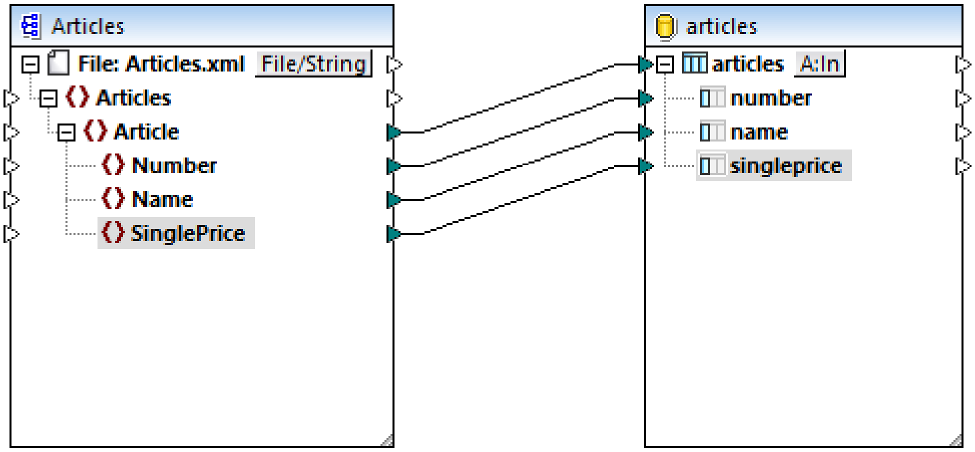
You have now finished creating the sample SQLite database required for this example.

To create the XML to SQLite mapping design:

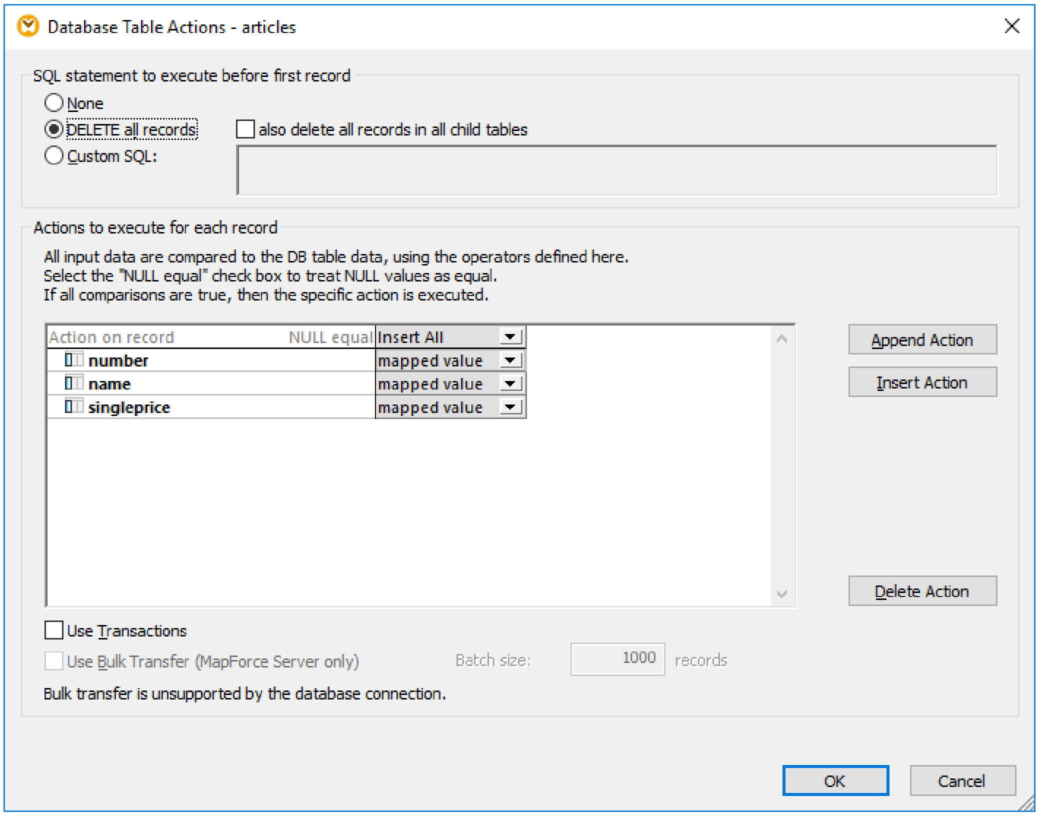
1. Run MapForce and make sure that the transformation language is set to BUILT-IN (use the menu command **Output | Built-in Execution Engine**).
2. Add to the mapping area the file **Articles.xml** located in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial** folder (use the menu command **File | Insert XML Schema/File**).
3. Add to the mapping area the database **articles.sqlite** created in previous steps (use the menu command **File | Insert Database**), and then select SQLite.
4. Click **Connect**. When prompted to choose the database objects, select the `articles` table.


Insert Database Objects
Please choose Tables, Views, Stored Procedures, or
user are stored locally in the mfd file.
Source
Add/Edit SELECT Statement...
articles
main
Tables
User Tables
articles
Show Preview >>

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.



Database Table Actions - articles

SQL statement to execute before first record

○ None

● DELETE all records □ also delete all records in all child tables

○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
Select the "NULL equal" check box to treat NULL values as equal.
If all comparisons are true, then the specific action is executed.

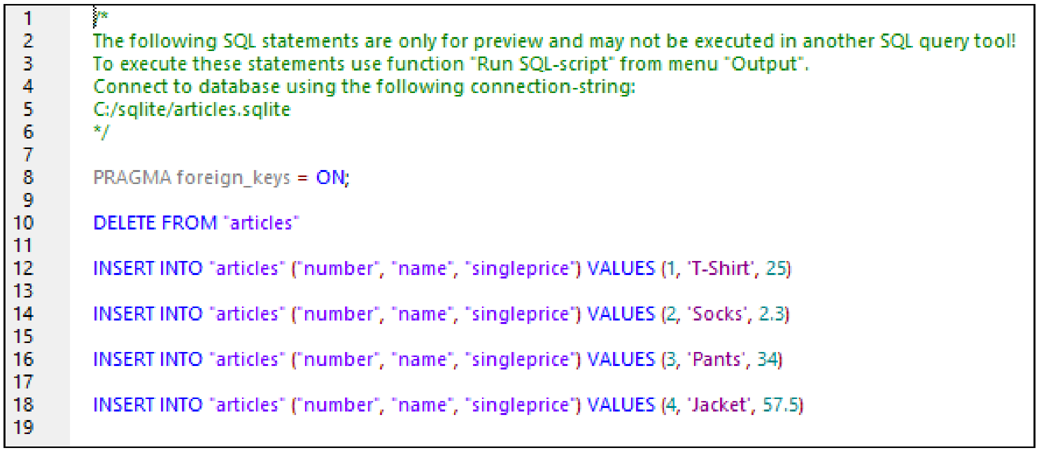
Action on record	NULL equal	Insert All
number		mapped value
name		mapped value
singleprice		mapped value

□ Use Transactions

□ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records

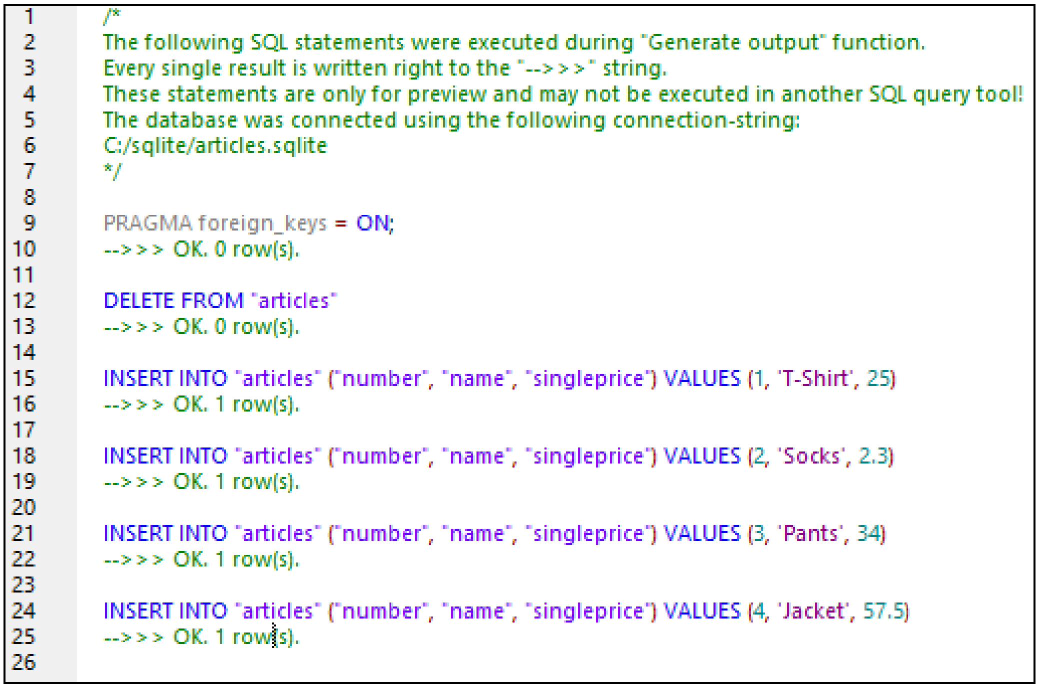
Bulk transfer is unsupported by the database connection.

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.

7.2.4 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 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.

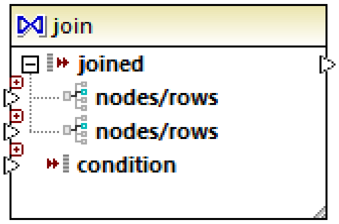
Joins in non-SQL mode are more flexible because they support more component types as input (for example, the join can be between tables from different databases, or between XML structures and database tables). For an example of a non-SQL join, see Example: Join XML Structures. On the other hand, a non-SQL 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 takes place very fast and is negligible 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 About Joins in SQL Mode.

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. 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.



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.

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).
- 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 About Joins in SQL Mode).
- 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.

7.2.4.1 About 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 automatically 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.

Note: From a database and SQL perspective, MapForce-generated joins are always INNER joins. That is, only records which satisfy the condition in both input sets are returned by the Join component.

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 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).

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.

4. All database tables must not yet be in the current target context. For example, if the mapping is designed in such a way that tables are queried by the mapping before the join operation, this could make the join impossible. For more information about how a mapping is executed, see Mapping Rules and Strategies.

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.

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").

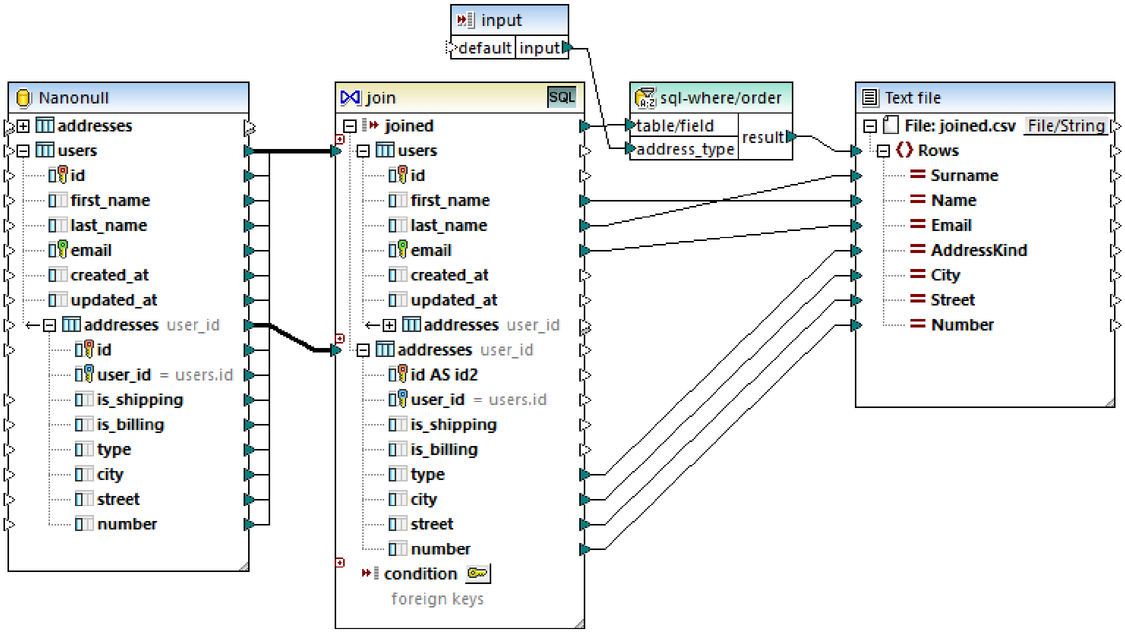
For an example mapping which uses SQL WHERE/ORDER clauses, see Example: Join Tables in SQL Mode.

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).

7.2.4.2 Example: Join Tables in SQL Mode

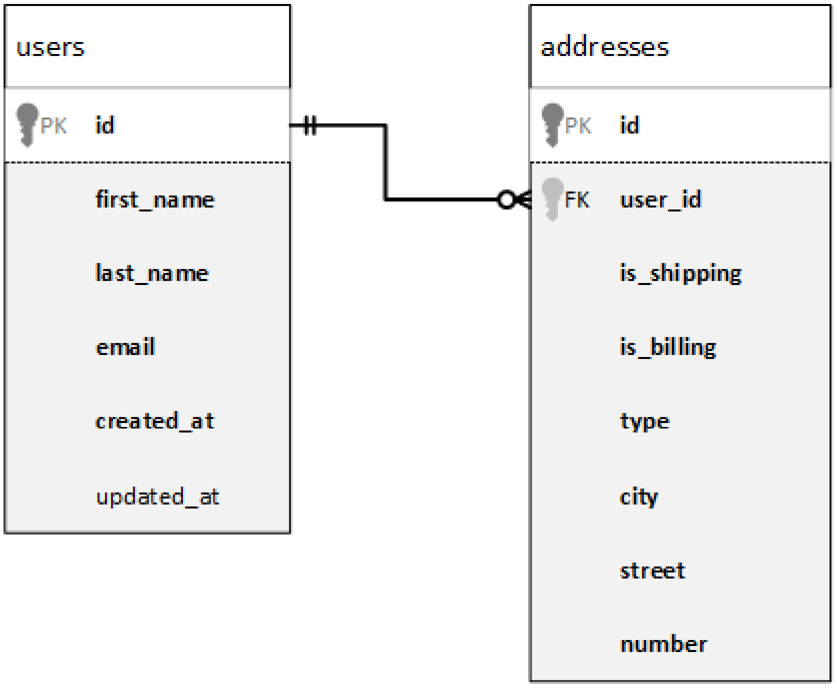
This example illustrates how to join data from two database tables, using a MapForce join component. The join operation is performed in SQL mode, as described in About Joins in SQL Mode. Note that joining three or more tables works in a very similar way, see also Example: Create CSV Report from Multiple Tables.

The example is accompanied by a mapping sample which is available at the following path: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial \JoinDatabaseTables.mfd**.

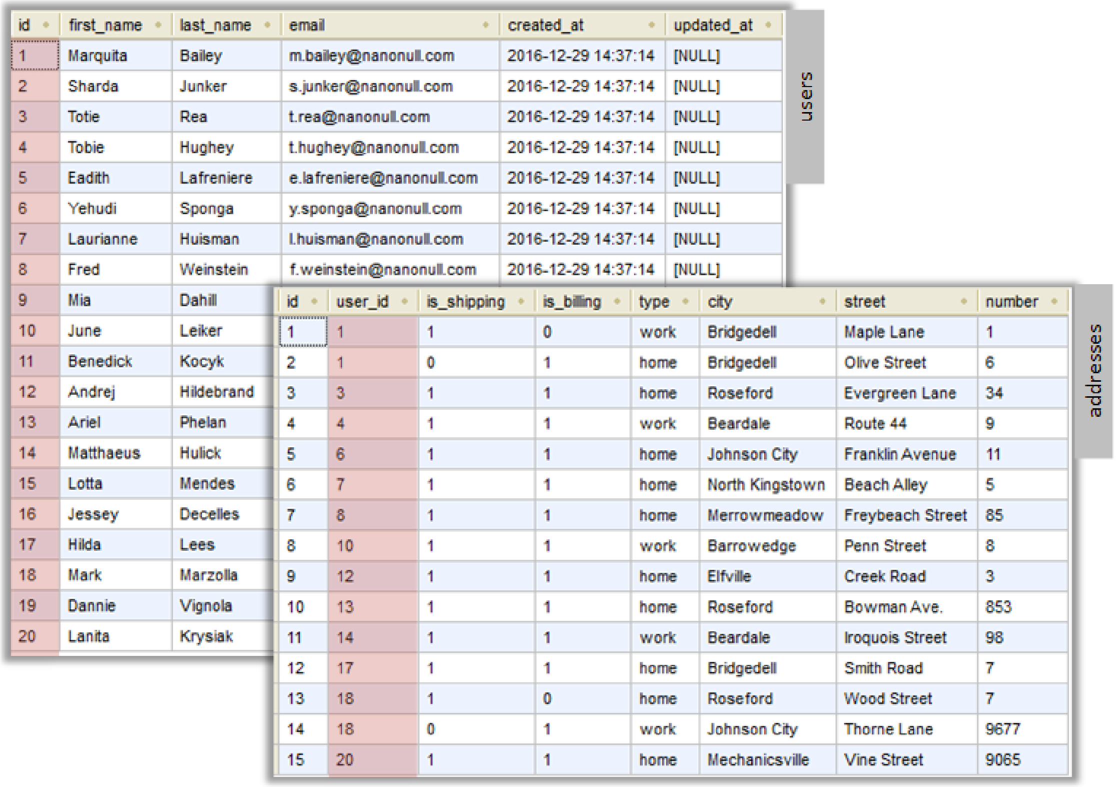


JoinDatabaseTables.mfd

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.

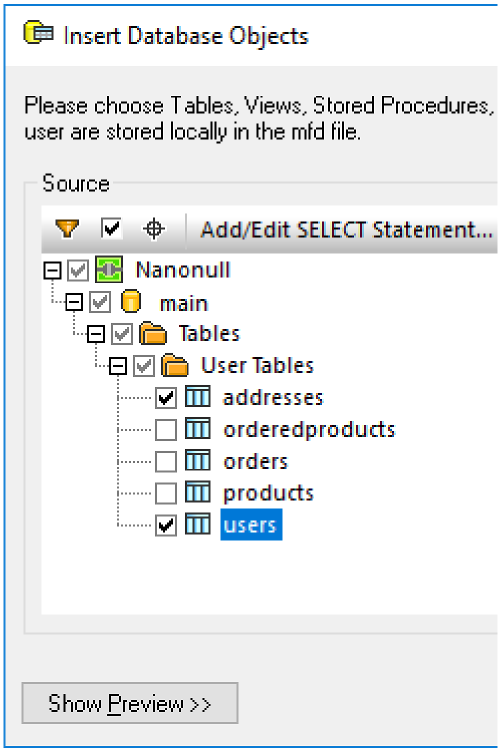
The goal of the mapping is to retrieve full data (name, surname, email, city, street, number) of all users that have at least one address in the addresses table. It should also be possible to easily retrieve only addresses of a specific kind (for example, only home addresses, or only work addresses). 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. 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 \MapForce2018\MapForceExamples\Tutorial\**, and click **Connect**.
4. When prompted, select the `addresses` and `users` tables.



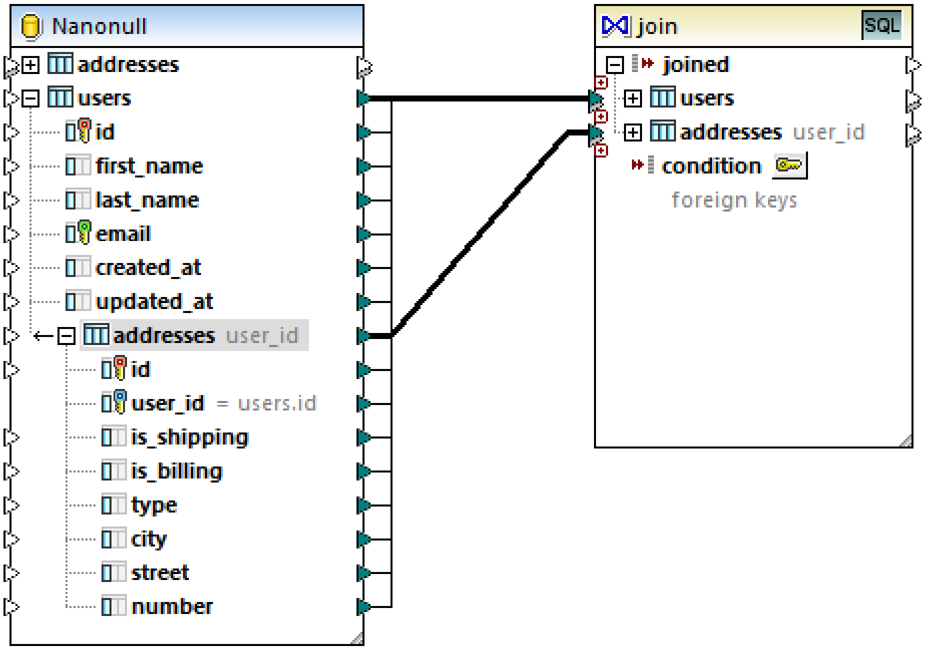
Insert Database Objects

Please choose Tables, Views, Stored Procedures, user are stored locally in the mfd file.

Source

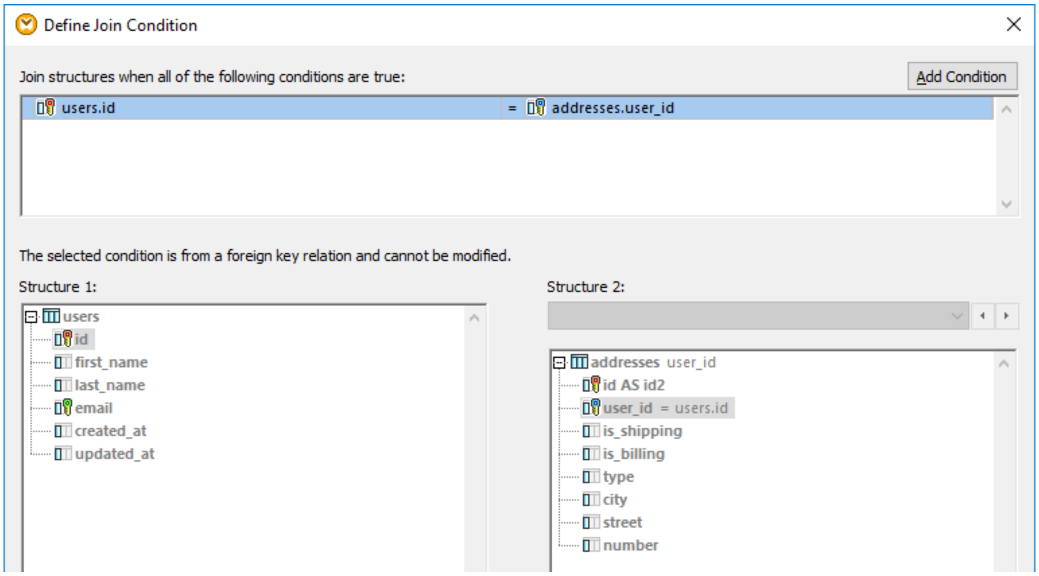
Step 2: Add the join component

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. 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`).


Define Join Condition

Join structures when all of the following conditions are true:

Add Condition

users.id	= addresses.user_id

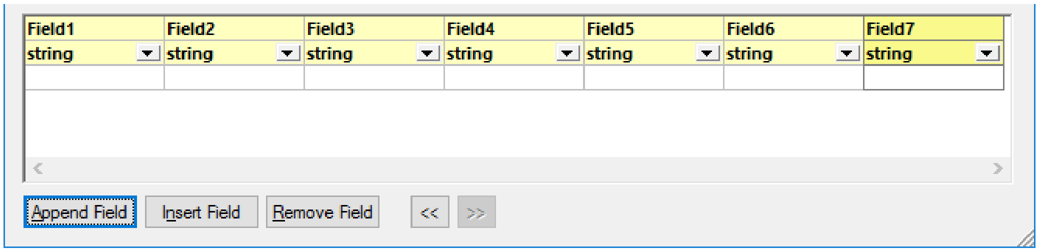
The selected condition is from a foreign key relation and cannot be modified.

Structure 1:

Structure 2:

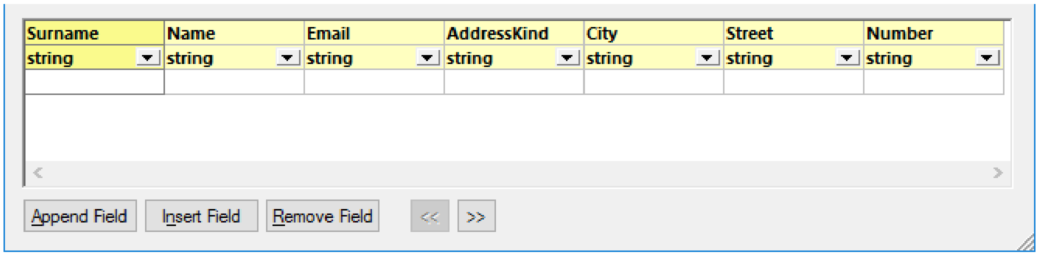
Step 3: Add the target CSV component

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.



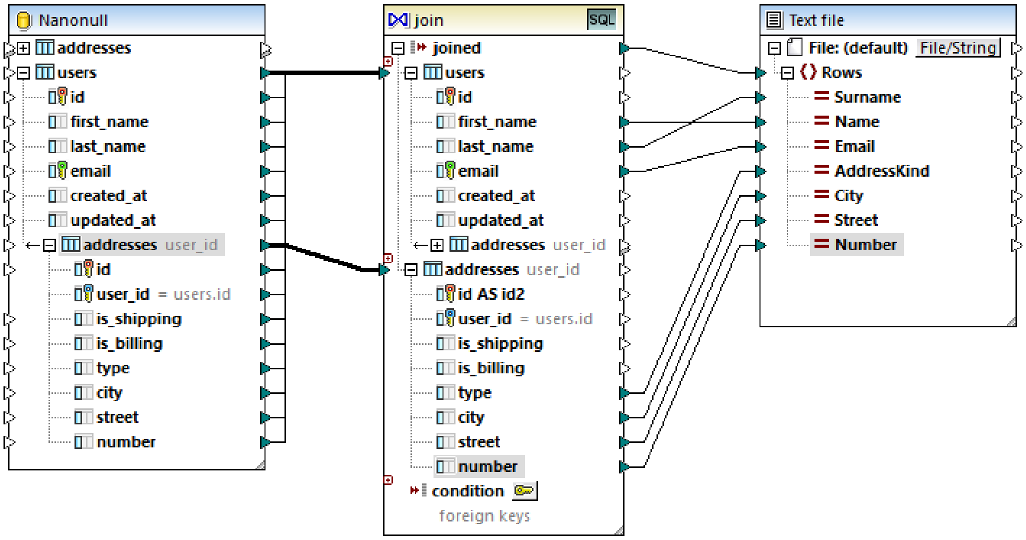
Field1	Field2	Field3	Field4	Field5	Field6	Field7
string	string	string	string	string	string	string

4. Double-click the title cell of each field to give it a descriptive name (this will make your mapping easier to read).



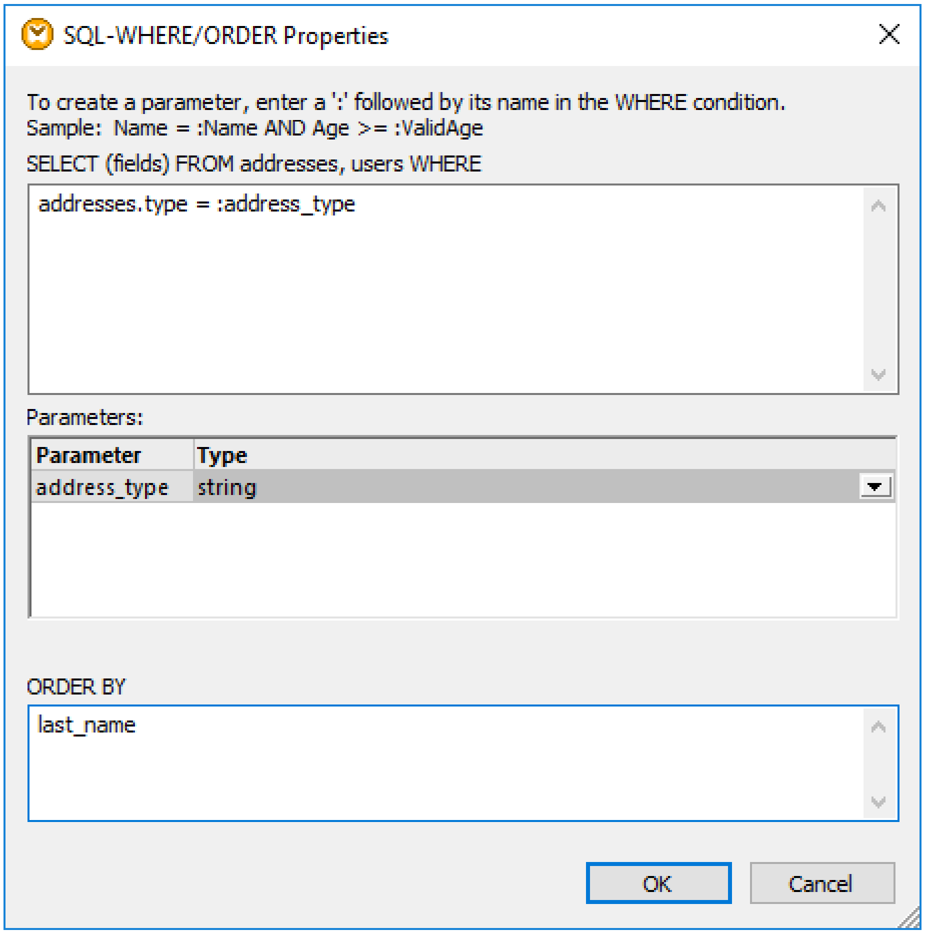
Surname	Name	Email	AddressKind	City	Street	Number
string	string	string	string	string	string	string

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.



SQL-WHERE/ORDER Properties

To create a parameter, enter a ':' followed by its name in the WHERE condition.
Sample: Name = :Name AND Age >= :ValidAge

SELECT (fields) FROM addresses, users WHERE

```
addresses.type = :address_type
```

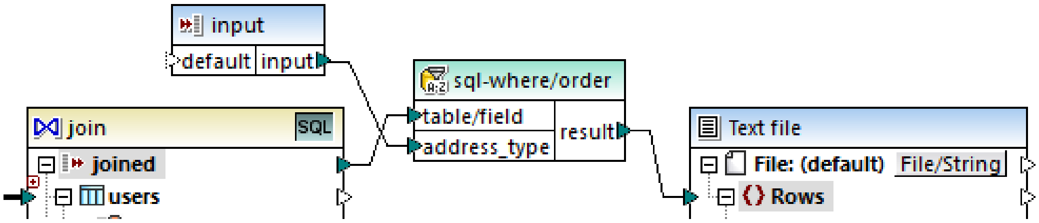
Parameters:

Parameter	Type
address_type	string

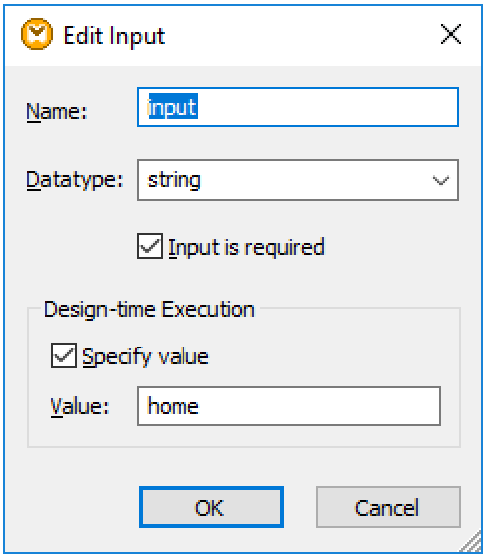
ORDER BY

```
last_name
```

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".



Name: input

Datatype: string

☑ Input is required

Design-time Execution

☑ Specify value

Value: home

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). Although not applicable to this example, it is also possible to define join conditions manually, see Example: Create CSV Report from Multiple Tables.

The two source tables are from the same database and from the same component, so this join benefits from the SQL (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 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).

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.

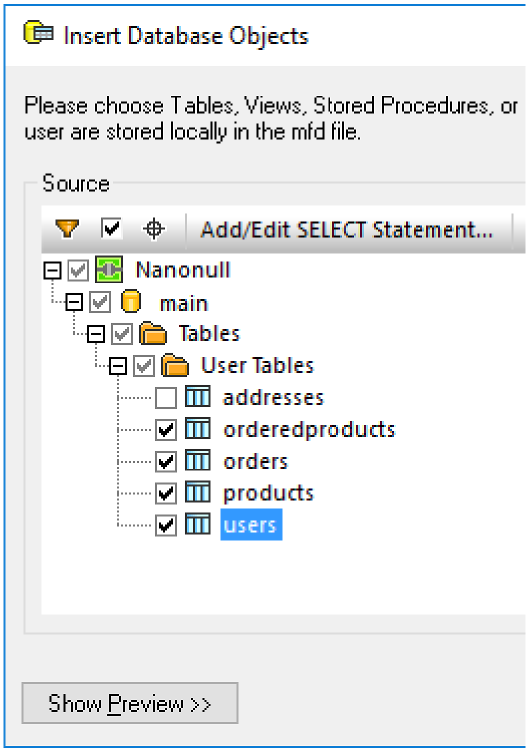
7.2.4.3 *Example: Create CSV Report from Multiple Tables*

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\MapForce2018**

\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:

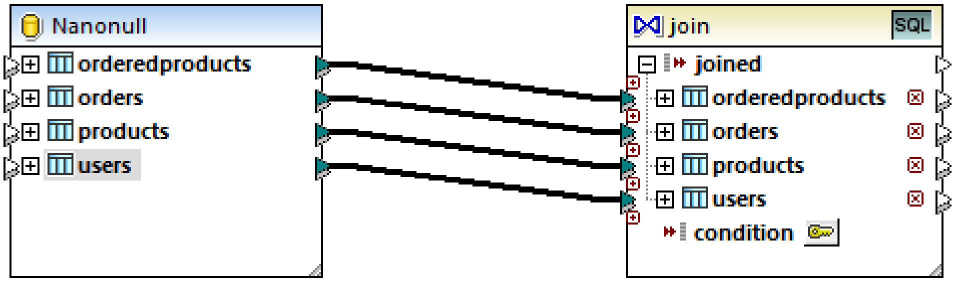
1. On the **Insert** menu, click **Database**.
2. When prompted to select a database kind, click **SQLite**, and then click **Next**.
3. Browse for the **Nanonull.sqlite** database mentioned above, and click **Connect**.
4. When prompted, select the tables `orderedproducts`, `orders`, `products`, and `users`, and click **OK**.



Please choose Tables, Views, Stored Procedures, or user are stored locally in the mfd file.

- ☑ Nanonull
  - ☑ main
    - ☑ Tables
      - ☑ User Tables
        - □ addresses
        - ☑ orderedproducts
        - ☑ orders
        - ☑ products
        - ☑ users

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.

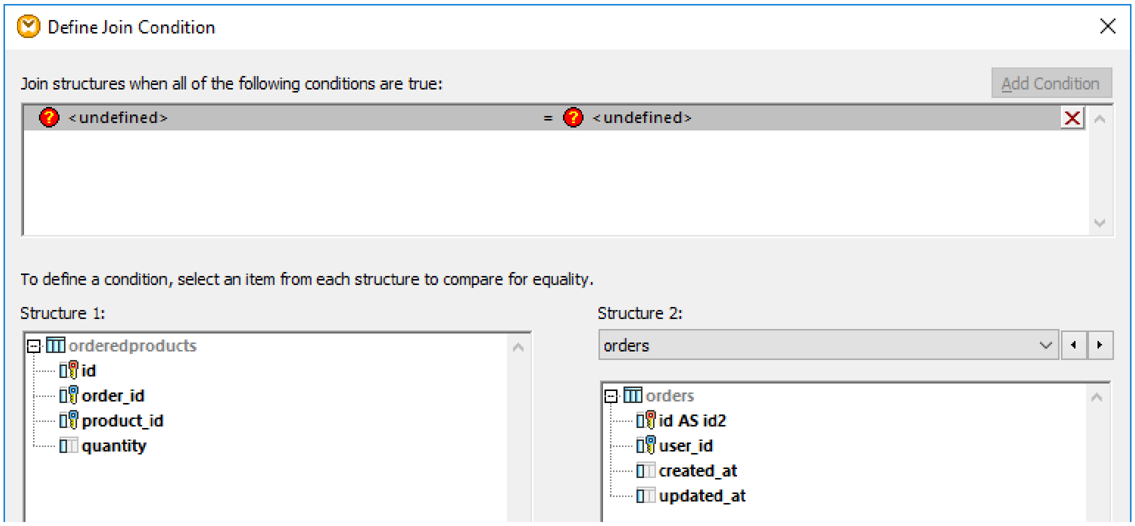


Note: In an alternative scenario, you could connect to the Join component the table `orderedproducts`, then the table `orders` (the one which is nested under it, not the one at the same level), and so on, so that all joined tables are nested under the same "root" table, see also Handling Database Relationships. The mapping result would be the same if you joined tables this way. The difference is that in this example the join conditions must be created manually, as shown below, 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. Another mapping where all joined tables are under the same "root" table is available at the following path: **<Documents>\Altova\MapForce2018\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**.



Join structures when all of the following conditions are true:

<undefined> = <undefined>

To define a condition, select an item from each structure to compare for equality.

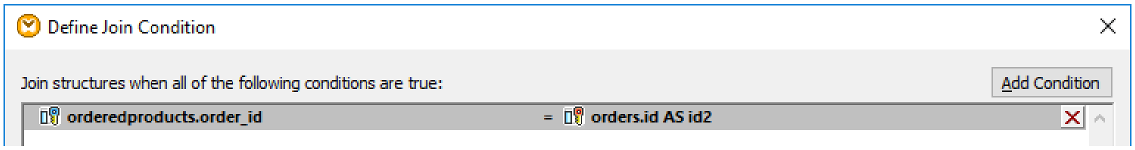
Structure 1:

- orderedproducts
  - id
  - order_id
  - product_id
  - quantity

Structure 2: orders

- orders
  - id AS id2
  - user_id
  - created_at
  - updated_at

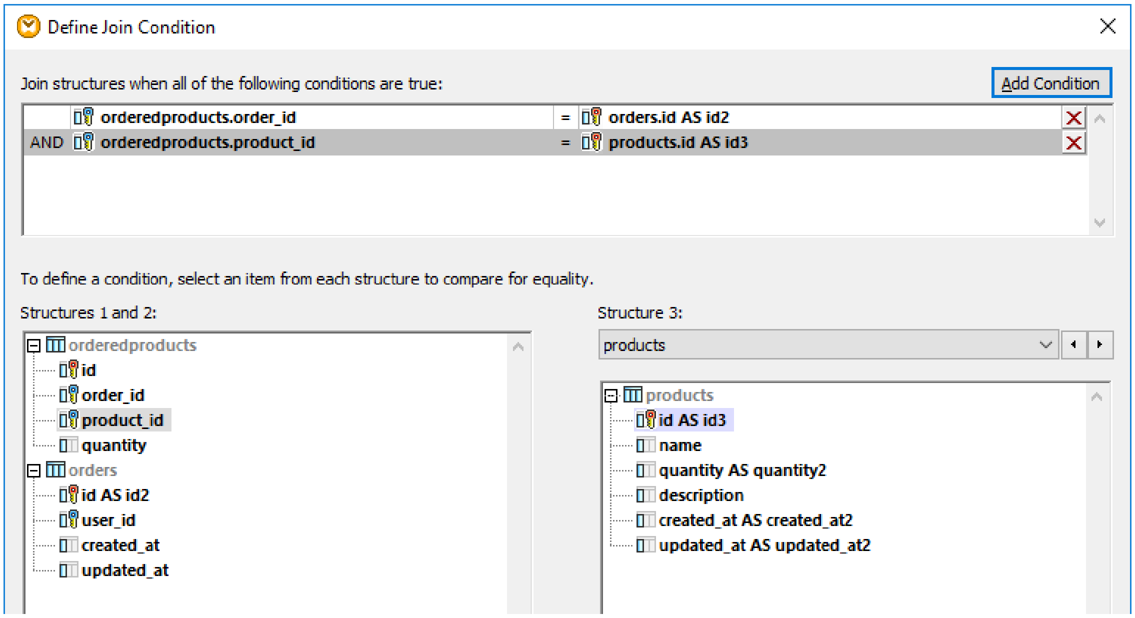
To define the first join condition, click the `order_id` item in the left pane and the `id` item in the right pane. Now the fields `orderedproducts.order_id` and `orders.id`. are paired:



Join structures when all of the following conditions are true:

orderedproducts.order_id = orders.id AS id2

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`).



Join structures when all of the following conditions are true:

	orderedproducts.order_id	=	orders.id AS id2
AND	orderedproducts.product_id	=	products.id AS id3

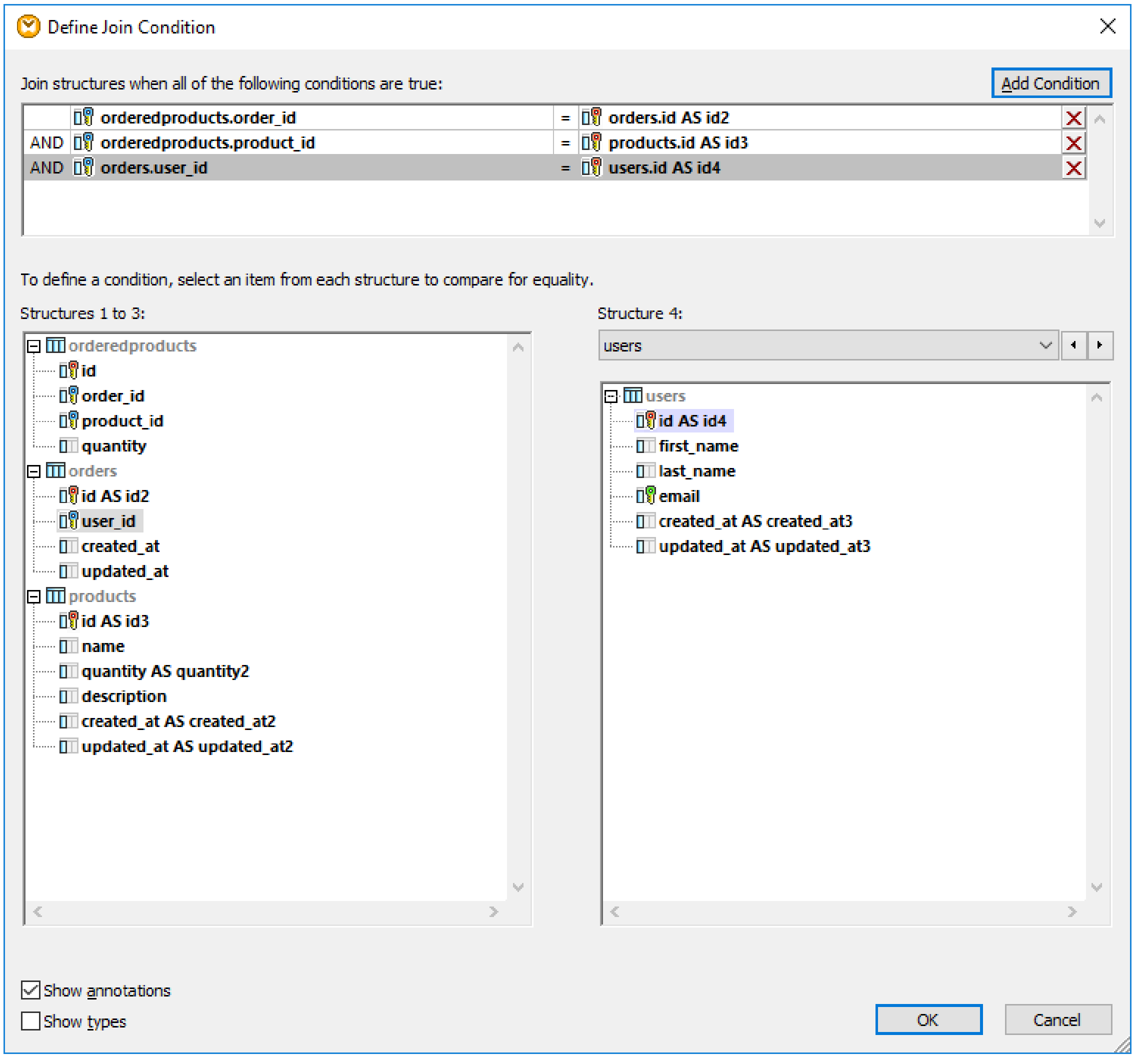
To define a condition, select an item from each structure to compare for equality.

Structures 1 and 2:

Structure 3:

products

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`.



Define Join Condition

Join structures when all of the following conditions are true:

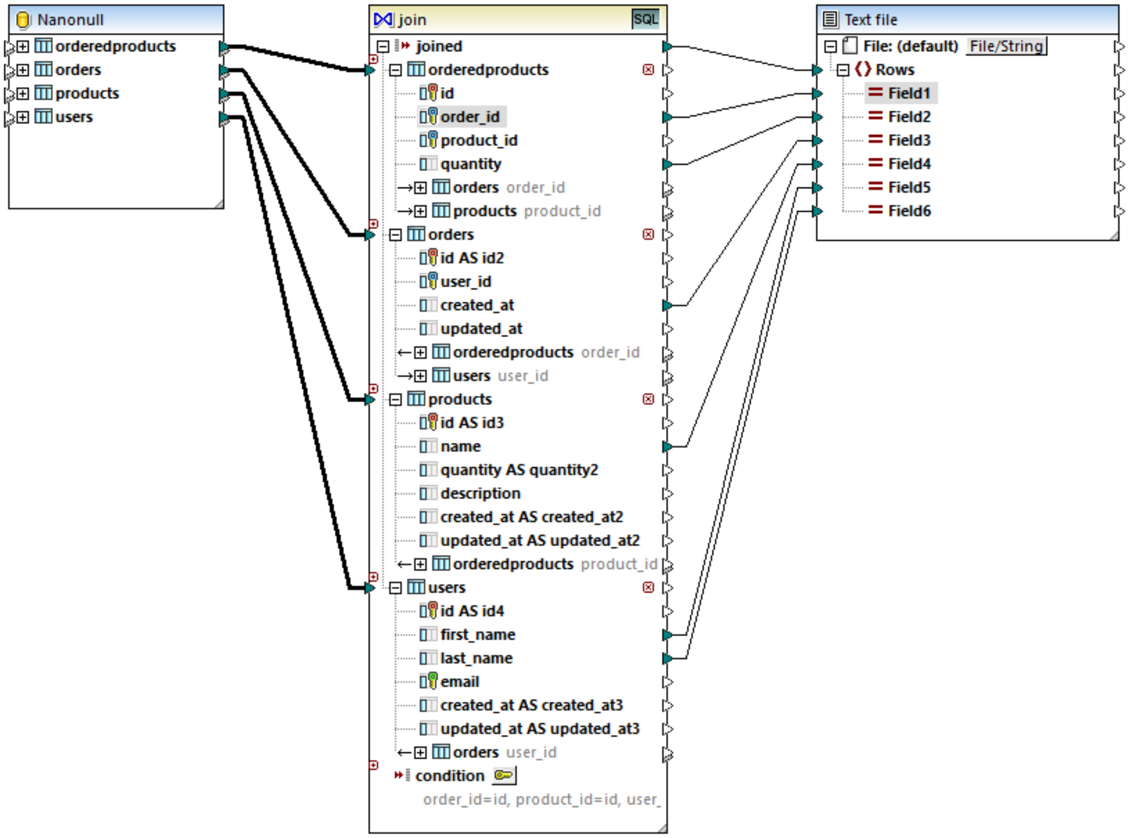
	orderedproducts.order_id	=	orders.id AS id2
AND	orderedproducts.product_id	=	products.id AS id3
AND	orders.user_id	=	users.id AS id4

To define a condition, select an item from each structure to compare for equality.

☑ Show annotations

☐ Show types

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), 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).

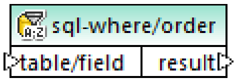
7.2.5 Filtering and Sorting Database Data (SQL WHERE/ORDER)

When you need to filter and sort database data, use an SQL WHERE/ORDER component. This enables you to manually enter, from the MapForce graphical user interface, a SQL WHERE clause that filters data. Optionally, you can also specify an ORDER BY clause if you want to sort the recordset by a particular database field, in ascending or descending order.

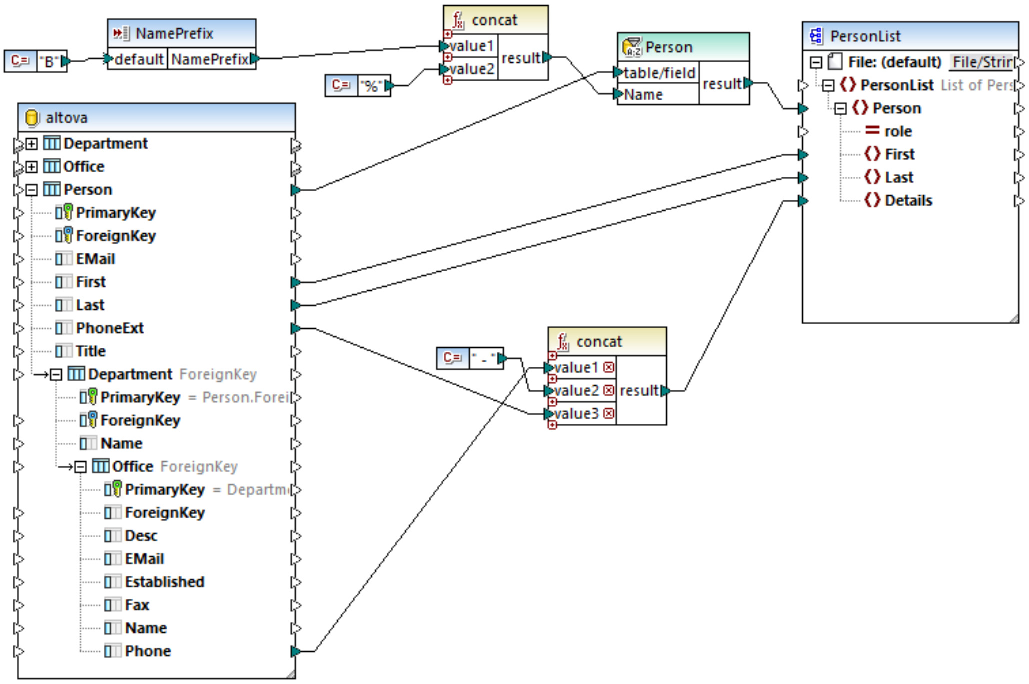
The SQL WHERE/ORDER component must be connected to a table or field of a database mapping component. It is also possible to connect the SQL WHERE/ORDER to a Join component, if you need to filter the joined set or records (see Joining Database Data).

Adding a SQL WHERE/ORDER component to the mapping

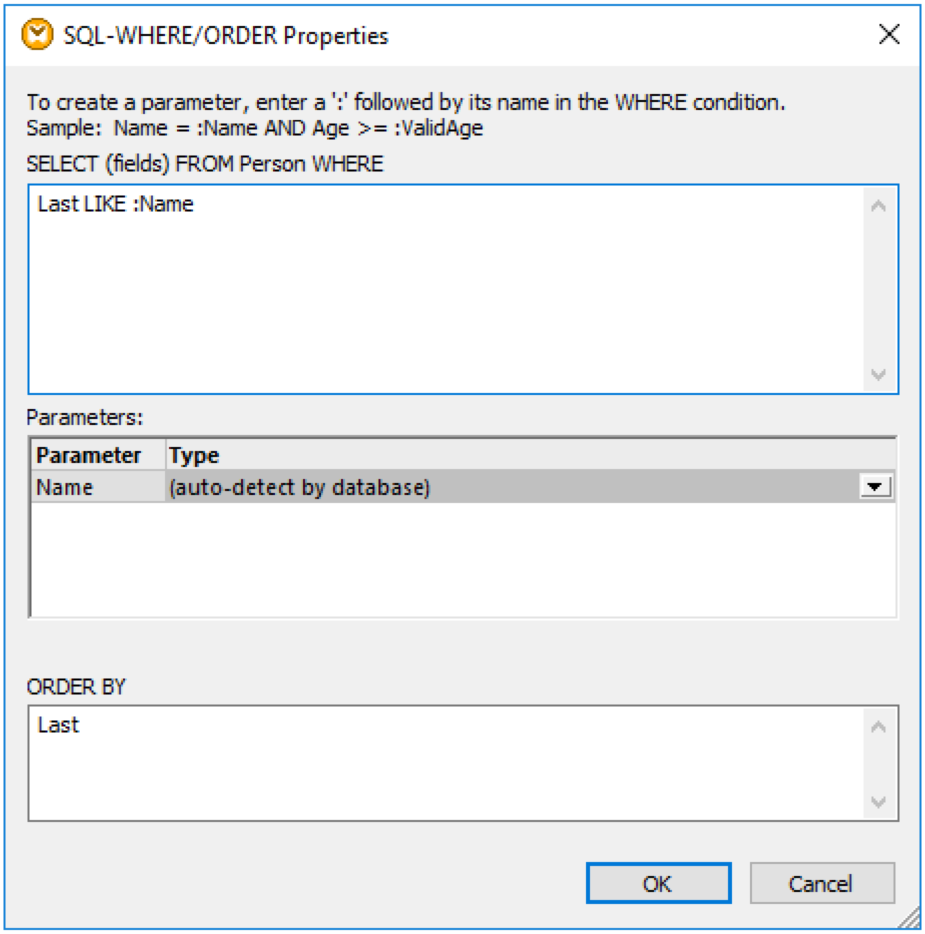
1. On the **Insert** menu, click **SQL WHERE/ORDER**. By default, the SQL WHERE/ORDER component has the following structure:



2. Connect a source database table or field to the `table/field` item of the SQL WHERE/ORDER. For an example, open the mapping **DB_PhoneList.mfd** from the folder **<Documents>\Altova\MapForce2018\MapForceExamples\**. In this mapping, the SQL WHERE/ORDER is used to filter from the source table "Person" all records where the last name begins with letter "B".



3. Double-click the header of the SQL WHERE/ORDER component (or right-click it and select **Properties** from the context menu). This opens the "SQL WHERE/ORDER Properties" dialog box.



To create a parameter, enter a ':' followed by its name in the WHERE condition.
Sample: Name = :Name AND Age >= :ValidAge
SELECT (fields) FROM Person WHERE

```
Last LIKE :Name
```

Parameters:

Parameter	Type
Name	(auto-detect by database)

ORDER BY

```
Last
```

4. Type the SQL WHERE clause in the text box at the top. Optionally, type the ORDER BY clause. The image above illustrates the WHERE and ORDER BY clauses defined in the **DB_PhoneList.mfd** mapping (these settings are further explained below). For more examples, see Creating WHERE and ORDER BY Clauses.

Supplying parameters to a SQL WHERE/ORDER

The SQL WHERE/ORDER component used in the mapping **DB_PhoneList.mfd** defines a WHERE clause as follows:

```
Last LIKE :Name
```

"Last" refers to the name of a database field in the connected table. "LIKE" is an SQL operator. ":Name" creates a parameter called "Name" on the mapping.

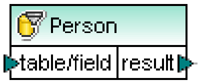
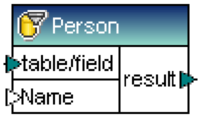
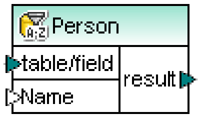
Parameters in SQL WHERE/ORDER components are optional; they are useful if you want to pass a value to the WHERE clause from the mapping. Without parameters, the WHERE clause above could have been written as follows:

```
Last LIKE "B%"
```

This would retrieve all persons whose last name begins with letter "B". In order to make this query even more flexible, we added a parameter instead of "B%". This makes it possible to supply any other letter from the mapping (for example, "C", and thus retrieve people whose last name begins with "C" simply by changing a constant, or a mapping input parameter).

Appearance of SQL WHERE/ORDER components

An important thing about SQL WHERE/ORDER components is that they change appearance depending on the settings defined in them. This way you can quickly view directly from the mapping what the SQL WHERE/ORDER component does, for example:

	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.

Placing the mouse cursor over the SQL WHERE/ORDER header opens a tooltip displaying the various clauses that have been defined.

7.2.5.1 *Creating WHERE and ORDER BY Clauses*

After an SQL 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 "SQL WHERE/ORDER Properties" dialog box of MapForce, as shown in the 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

Operator	Description
<>	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.

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 "SQL WHERE/ORDER Properties" dialog box. 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
```

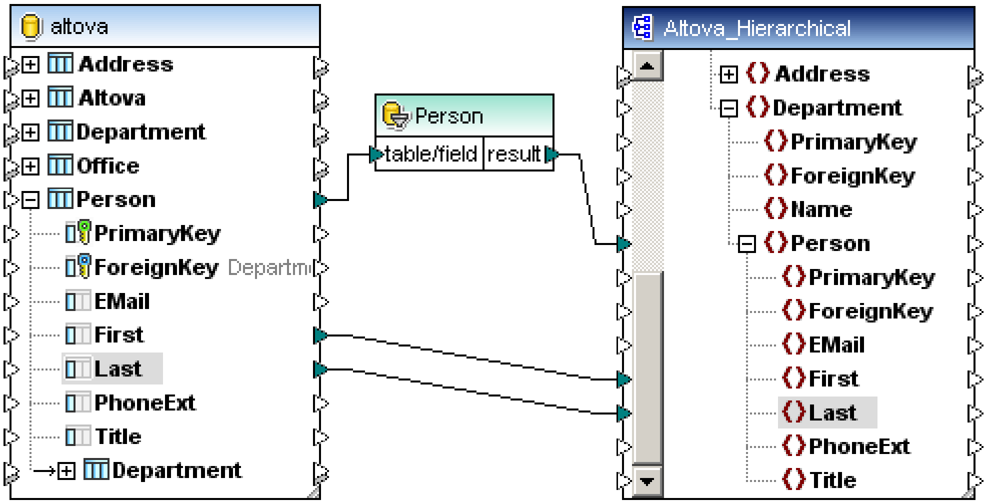
Example 1

The following WHERE condition is attached to the `Person` table of the **altova.mdb** database component. It retrieves those records where `First` and `Last` are greater than the letter "C". In other words, it retrieves all names from "Callaby" onwards.

```
First > "C" AND Last > "C"
```

Note how the connections are placed:

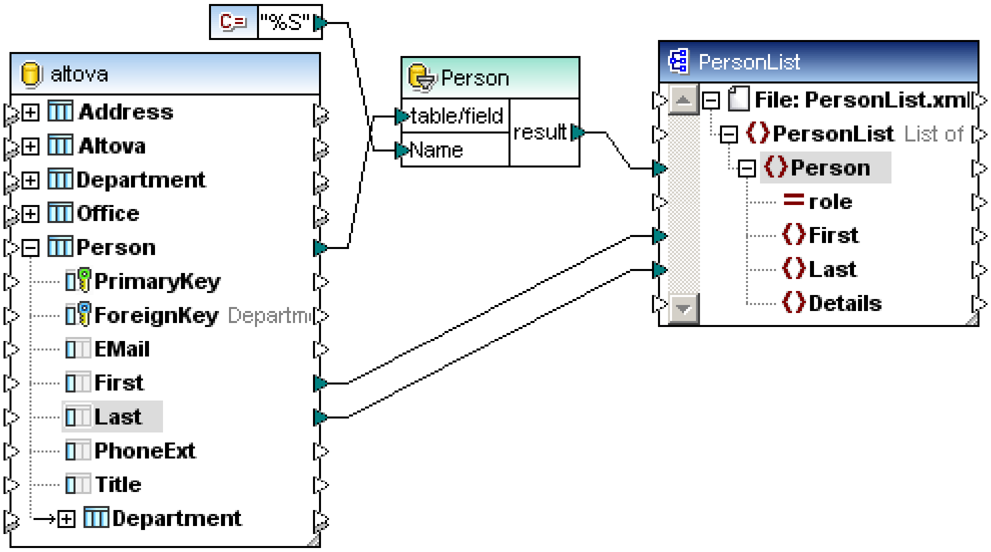
- The connection to **table/field** originates in the table that you want to query, "Person" in this case.
- The **result** output is connected to a "parent" item of the fields that are queried/filtered, in this case the `Person` item.



Example 2

The following WHERE condition creates a parameter **Name** which then appears in the SQL WHERE/ORDER component on the mapping.

```
Last LIKE :Name
```

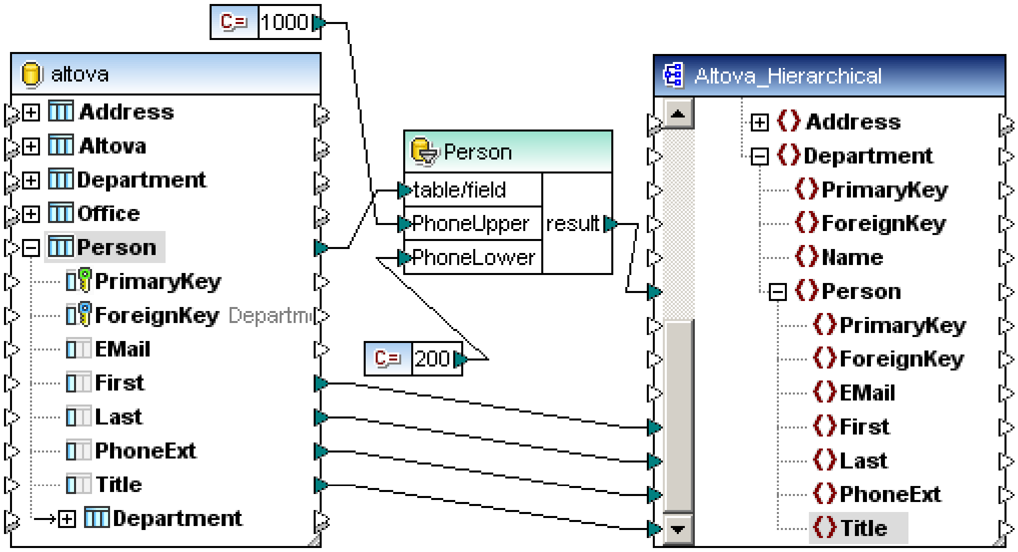


The constant component `%S` supplies the value of the **Name** parameter. The wildcard % denotes any number of characters. This causes the mapping to search for a pattern in the column "Last" (all last names ending in "S").

Example 3

The following WHERE condition creates two parameters, `PhoneUpper` and `PhoneLower`, to which the current values of `PhoneExt` are compared. The upper and lower values are supplied by two constant components shown in the diagram below.

```
PhoneExt < :PhoneUpper and PhoneExt > :PhoneLower
```



The WHERE condition in this example could also be written using the BETWEEN operator:

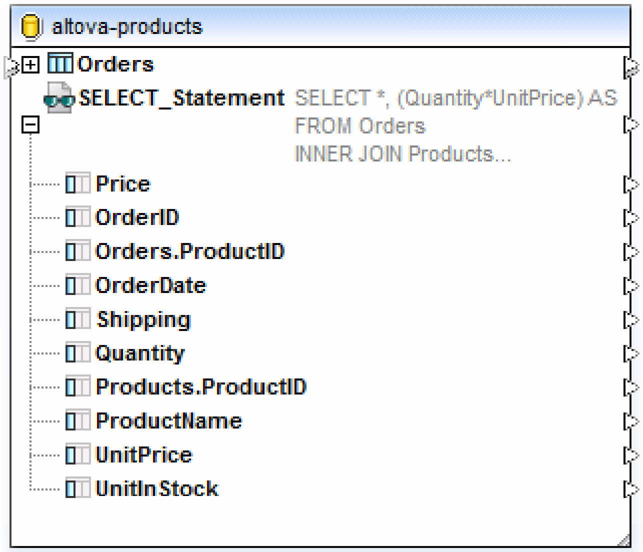
```
PhoneExt BETWEEN :PhoneUpper and :PhoneLower
```

7.2.6 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 component.

Once the SELECT statement has been added to a database component, the fields returned by it are available for mapping, for example:



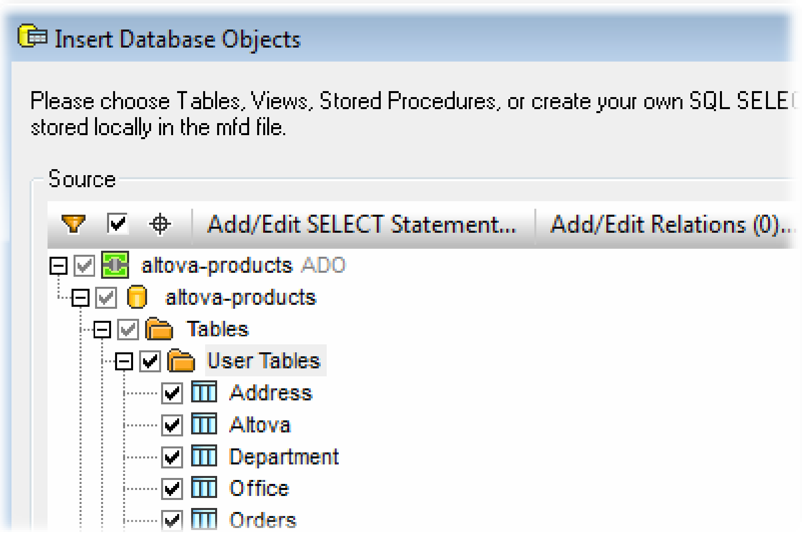
The number of visible lines of the SELECT statement is configurable. To define the number of lines you want to see on the component, select the menu command **Tools | Options**, click the **General** tab and enter the number of lines in the Mapping View group.

7.2.6.1 *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). 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\MapForce2018\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**).

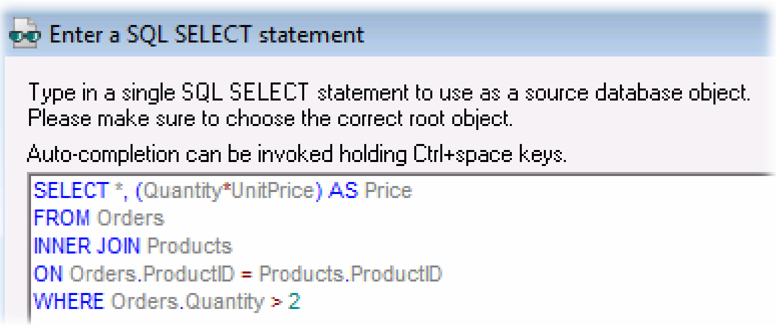


Insert Database Objects

Please choose Tables, Views, Stored Procedures, or create your own SQL SELE
stored locally in the mfd file.

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 \MapForce2018\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
```



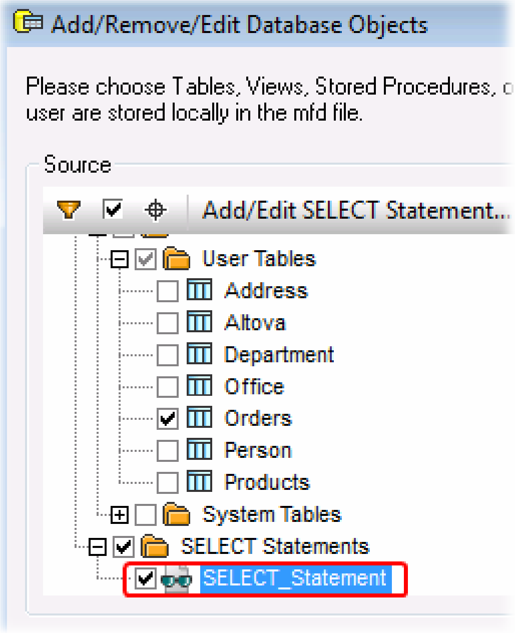
Enter a SQL SELECT statement

Type in a single SQL SELECT statement to use as a source database object.
Please make sure to choose the correct root object.

Auto-completion can be invoked holding Ctrl+space keys.

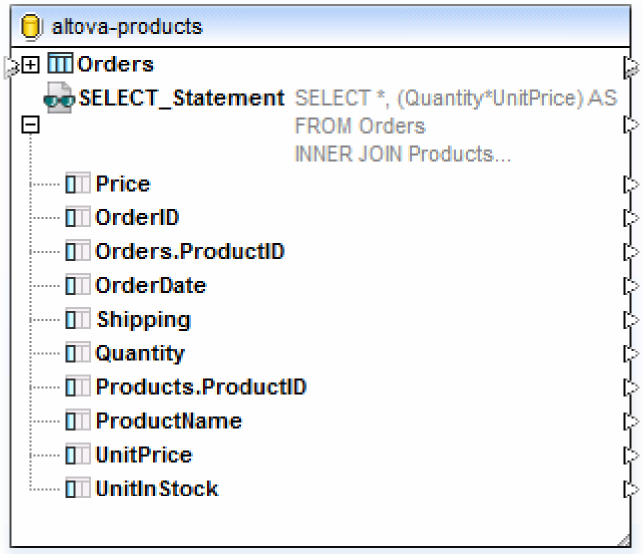
```
SELECT *, (Quantity*UnitPrice) AS Price
FROM Orders
INNER JOIN Products
ON Orders.ProductID = Products.ProductID
WHERE Orders.Quantity > 2
```

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.



Please choose Tables, Views, Stored Procedures, c
user are stored locally in the mfd file.

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.



```
SELECT *, (Quantity*UnitPrice) AS
FROM Orders
INNER JOIN Products...
```

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**.

7.2.6.2 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\MapForce2018\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.

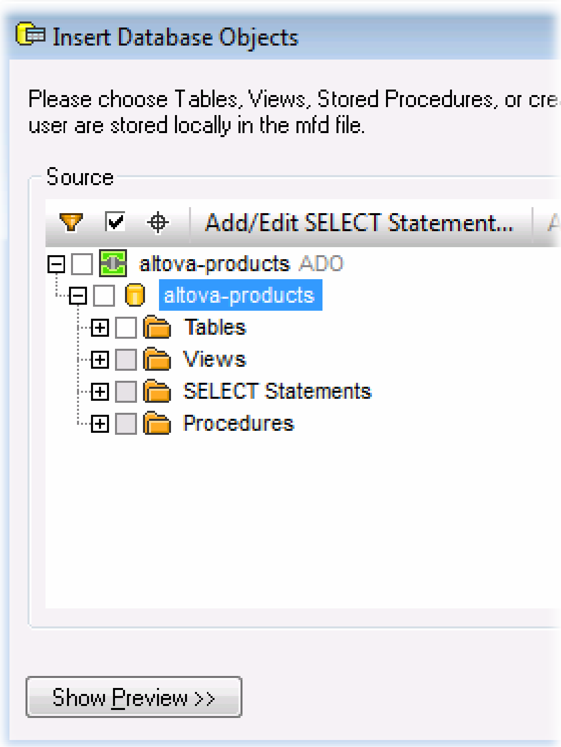
Although this example uses a Microsoft Access database, the process works in the same way for other database types. For information about connecting to other databases, see Connecting to a Database.

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 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\MapForce2018\MapForceExamples\Tutorial\** folder.



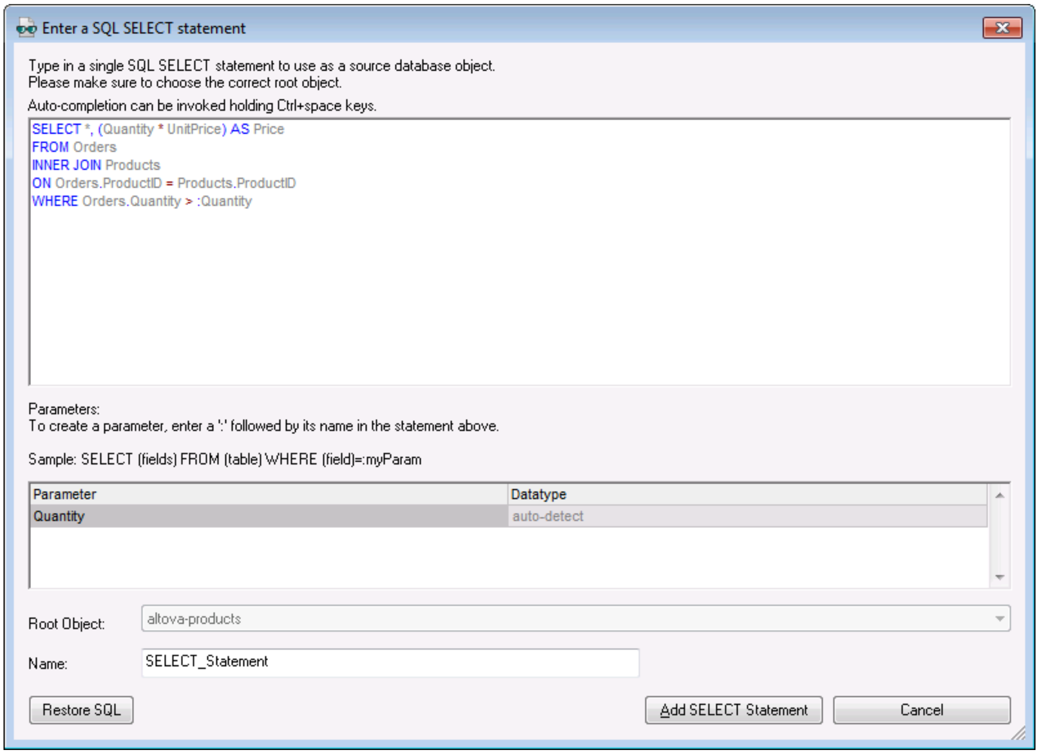
Insert Database Objects

Please choose Tables, Views, Stored Procedures, or cre
user are stored locally in the mfd file.

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.



Enter a SQL SELECT statement

Type in a single SQL SELECT statement to use as a source database object.
Please make sure to choose the correct root object.
Auto-completion can be invoked holding Ctrl+space keys.

```
SELECT *, (Quantity * UnitPrice) AS Price
FROM Orders
INNER JOIN Products
ON Orders.ProductID = Products.ProductID
WHERE Orders.Quantity > :Quantity
```

Parameters:
To create a parameter, enter a ':' followed by its name in the statement above.

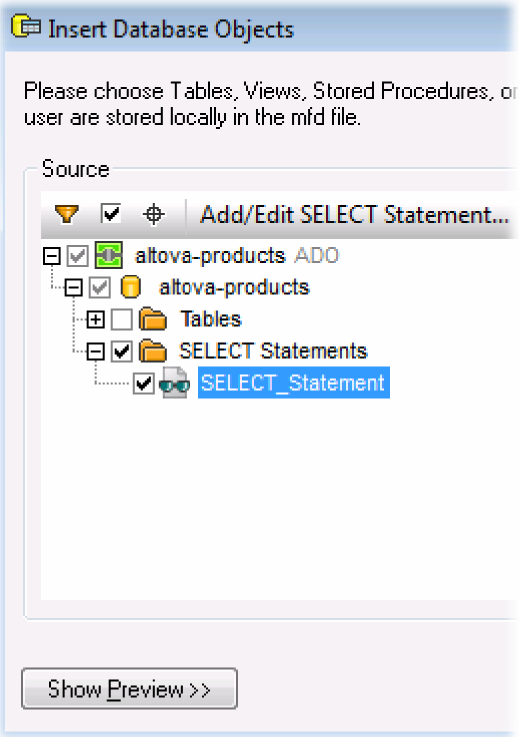
Sample: SELECT (fields) FROM (table) WHERE (field)=:myParam

Parameter	Datatype
Quantity	auto-detect

Root Object: altova-products

Name: SELECT_Statement

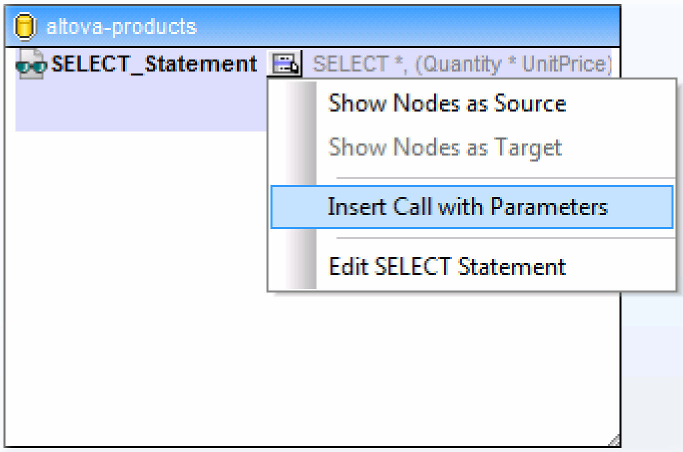
4. Click **Add SELECT statement**.



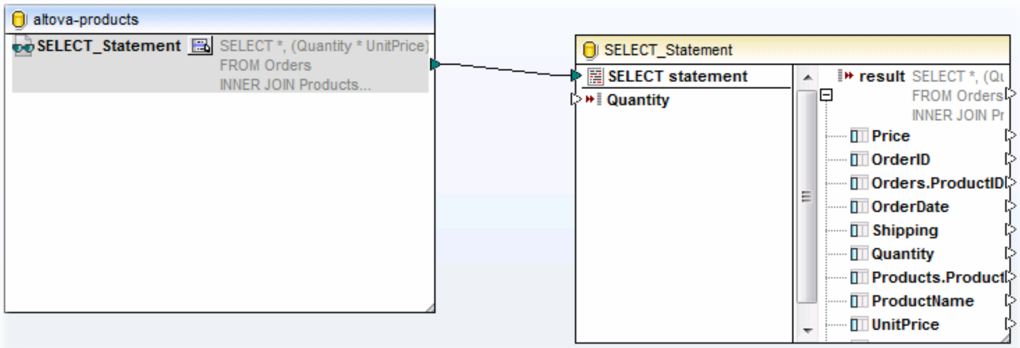
Insert Database Objects

Please choose Tables, Views, Stored Procedures, o
user are stored locally in the mfd file.

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**.

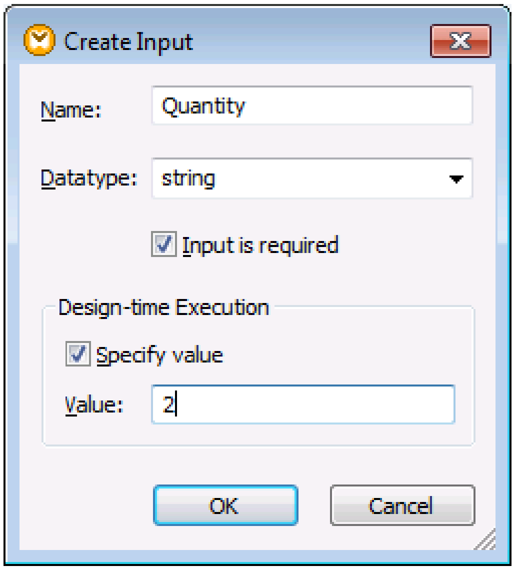


A new structure (SELECT_Statement) is now available on the mapping. It is split into two parts: the left part supplies input connectors and the right part supplies output connectors. Notice that the left part also includes the **Quantity** parameter defined previously.



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.



Create Input

Name: Quantity

Datatype: string

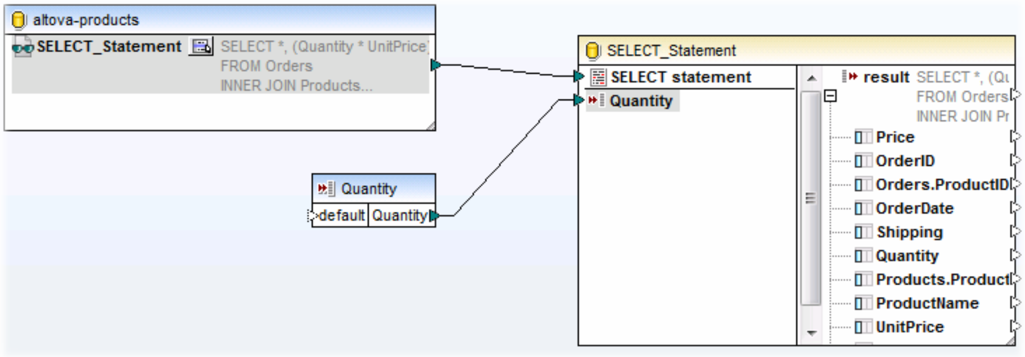
☑ Input is required

Design-time Execution

☑ Specify value

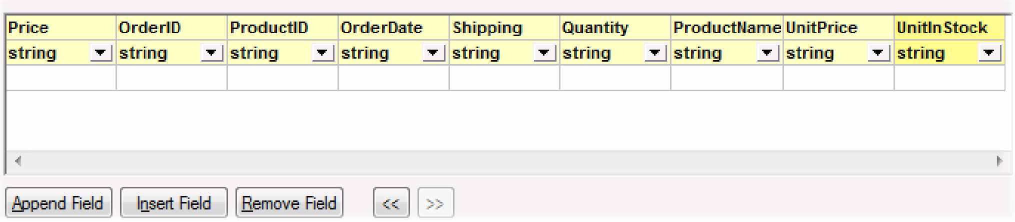
Value: 2

You can now connect the input parameter to the database call structure, as shown below.



Step 3: Add the target CSV component

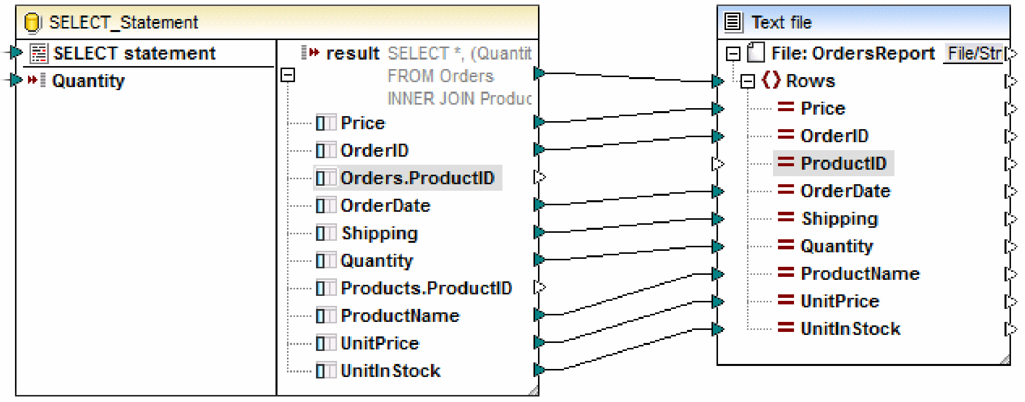
1. On the **Insert** menu, click **Text File**.
2. Select **Use simple processing for standard CSV...**, and then click **Continue**.
3. On the Component Settings dialog box, click **Append Field** and add nine new fields. It is recommended that you give to the CSV fields the same name as the name of 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.



Price	OrderID	ProductID	OrderDate	Shipping	Quantity	ProductName	UnitPrice	UnitInStock
string	string	string	string	string	string	string	string	string

4. Create a connection between the **result** node of the SELECT structure and the **Rows**

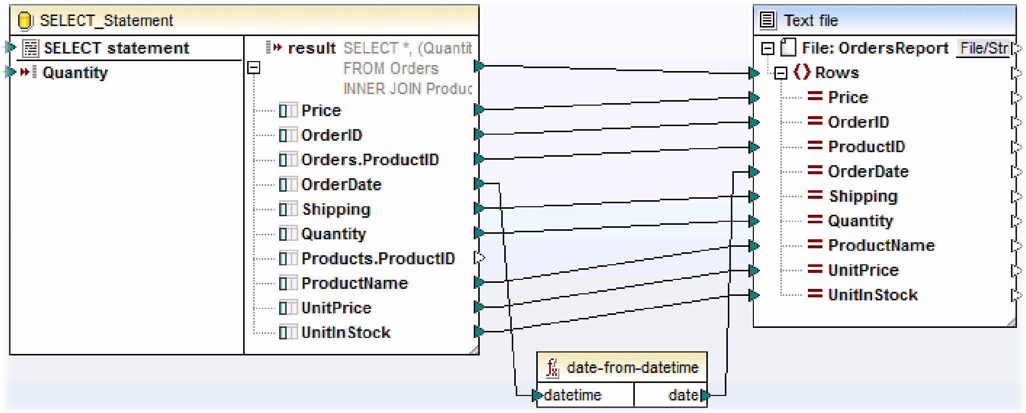
node of the CSV component.



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.

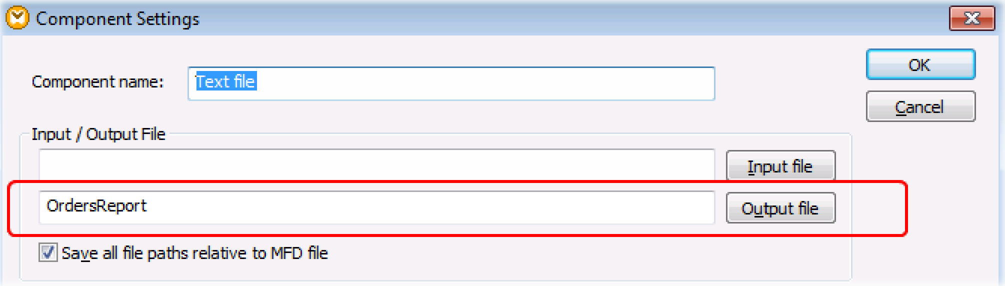
Step 4: Convert the date

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 CSV component, and enter the value in the Output File box.



Component Settings

Component name: Text file

Input / Output File

OrdersReport

☑ Save all file paths relative to MFD file

7.2.7 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.

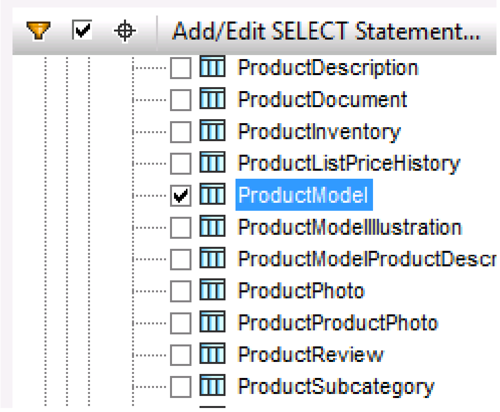
Some databases support XML encoding for XML fields (which may not necessarily be the same as that of the database character set). If supported by the database, the XML document encoding declaration is assumed to be the one declared in the XML field. For information about the XML encoding support provided by various databases, refer to their documentation.

7.2.7.1 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 Adventure Works 2014 database. The latter can be downloaded from the CodePlex website (https://msftdbprodsamples.codeplex.com/). 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:

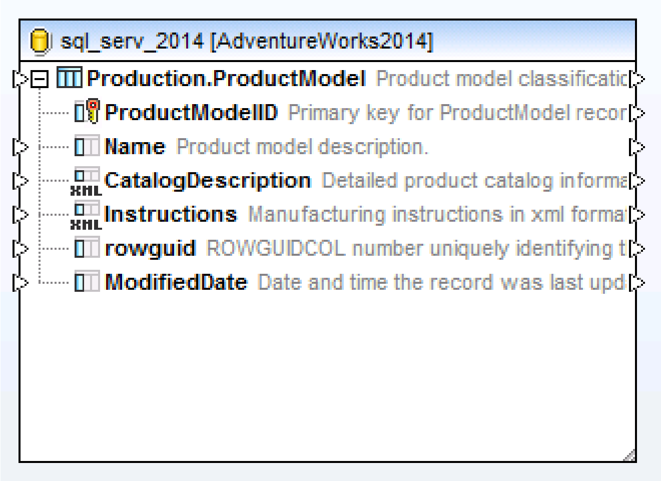
1. On the **Insert** menu, click **Database**, and follow the wizard to connect to the database using your preferred method (ADO or ODBC). For more information, see Connecting to Microsoft SQL Server (ADO) and Connecting to Microsoft SQL Server (ODBC). NOTE: If you use the **SQL Server Native Client** driver, you might need to set the **Integrated Security** property to a space character (see Setting up the SQL Server Data Link Properties).
2. On the **Insert Database Object** dialog box, expand the **Production** schema, and then select the **ProductModel** table.


Add/Edit SELECT Statement...

□ ProductDescription
□ ProductDocument
□ ProductInventory
□ ProductListPriceHistory
☑ ProductModel
□ ProductModelIllustration
□ ProductModelProductDescr
□ ProductPhoto
□ ProductProductPhoto
□ ProductReview
□ ProductSubcategory

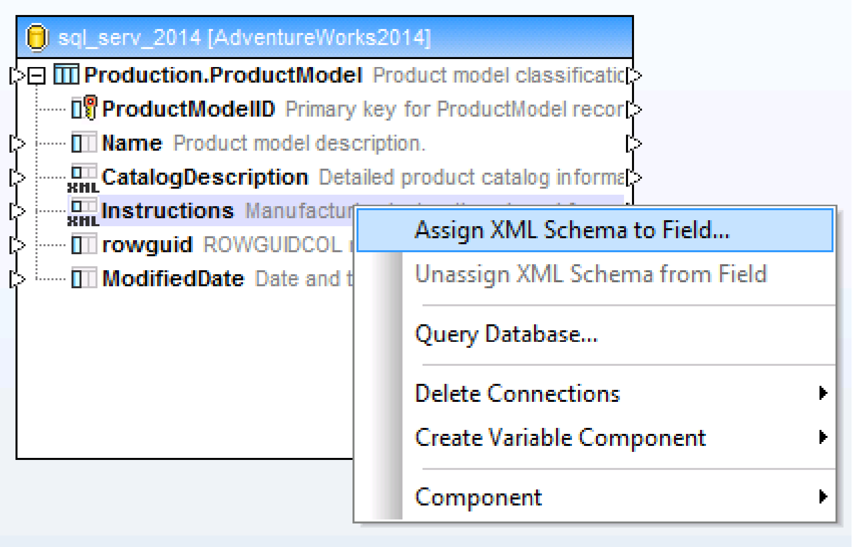
3. Click OK.

The database table has now been added to the mapping area. Notice that this table has two fields of XML type: **CatalogDescription** and **Instructions**:

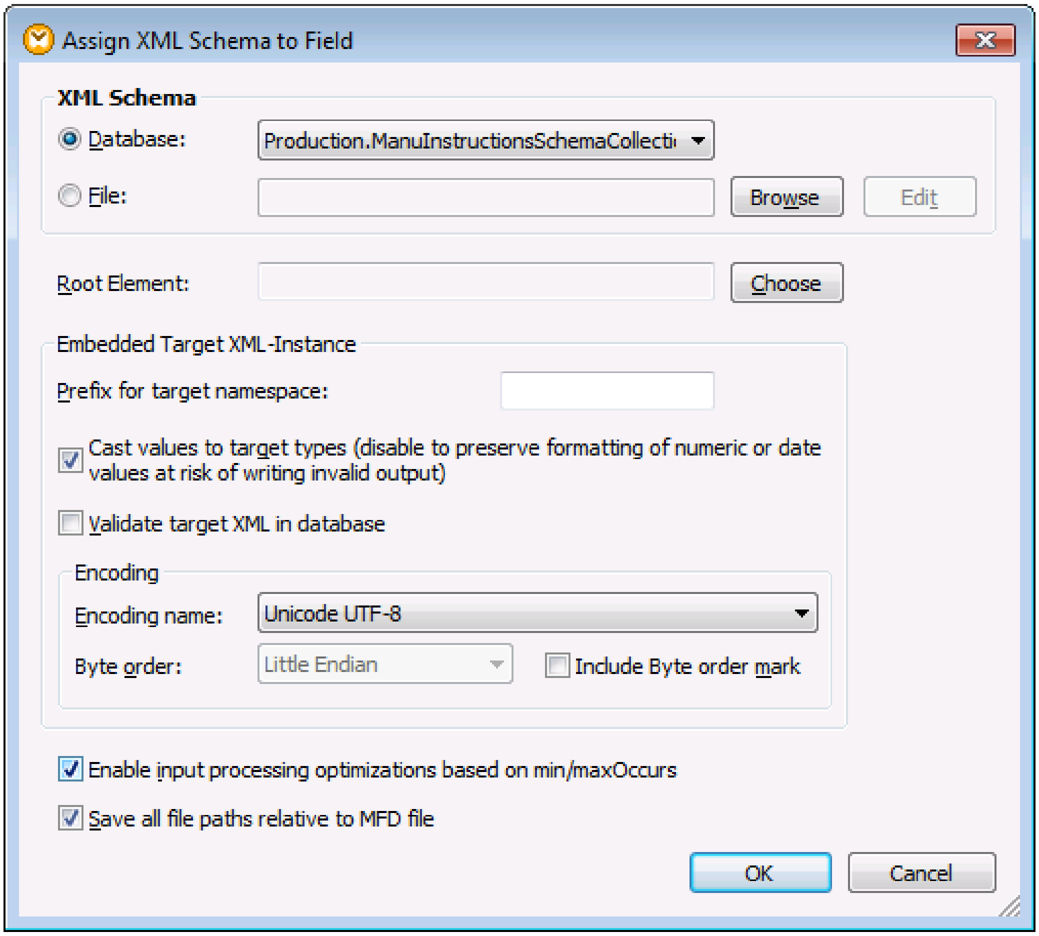

sql_serv_2014 [AdventureWorks2014]

Production.ProductModel Product model classificatio
ProductModelID Primary key for ProductModel recor
Name Product model description.
CatalogDescription Detailed product catalog informa
Instructions Manufacturing instructions in xml forma
rowguid ROWGUIDCOL number uniquely identifying t
ModifiedDate Date and time the record was last upd

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.


sql_serv_2014 [AdventureWorks2014]
Production.ProductModel Product model classificatio
ProductModelID Primary key for ProductModel recor
Name Product model description.
CatalogDescription Detailed product catalog informa
Instructions Manufactur
rowguid ROWGUIDCOL
ModifiedDate Date and t
Assign XML Schema to Field...
Unassign XML Schema from Field
Query Database...
Delete Connections
Create Variable Component
Component

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.


Assign XML Schema to Field

XML Schema

● Database: Production.ManuInstructionsSchemaCollecti

○ File:

Root Element:

Embedded Target XML-Instance

Prefix for target namespace:

☑ Cast values to target types (disable to preserve formatting of numeric or date values at risk of writing invalid output)

☐ Validate target XML in database

Encoding

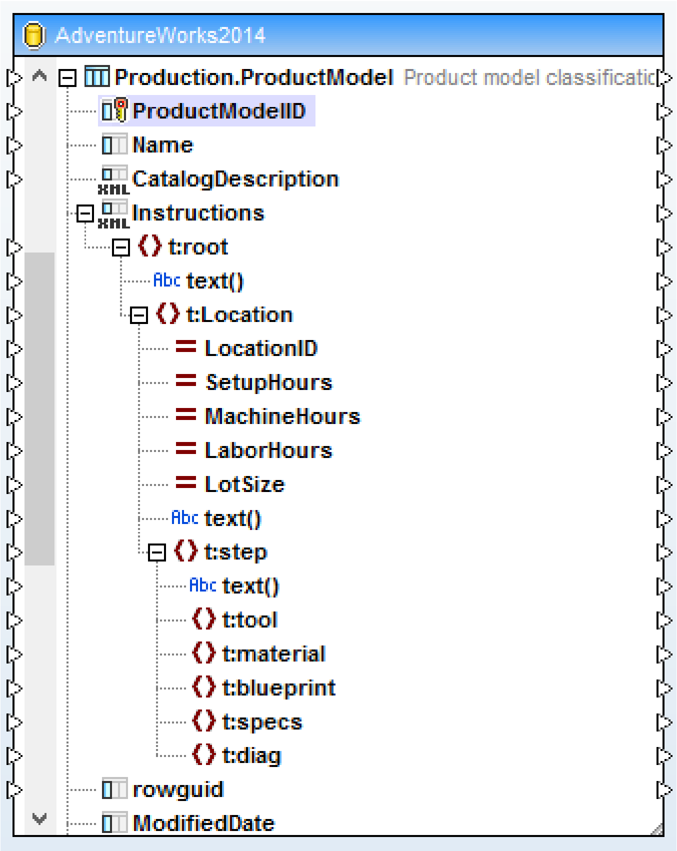
Encoding name: Unicode UTF-8

Byte order: Little Endian ☐ Include Byte order mark

☑ Enable input processing optimizations based on min/maxOccurs

☑ Save all file paths relative to MFD file

The structure of the XML field now appears on the component. You can now draw connections (and map data) to or from this field.



7.2.7.2 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 \MapForce2018\MapForceExamples\Tutorial\**. The file names are as follows:

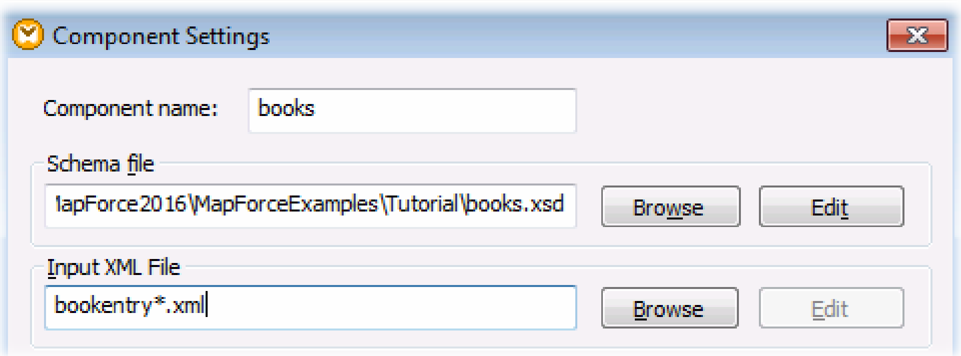
The mapping design file	• **XmlToSqliteField.mfd**
The source XML files	• **bookentry1.xml** • **bookentry2.xml** • **bookentry3.xml**
The XML schema used for validation	• **books.xsd**
The target SQLite database	• **Library.sqlite**

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\MapForce2018\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 Dynamically.



Component Settings

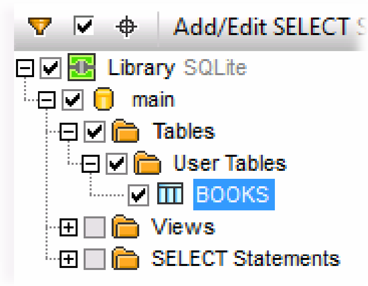
Component name: books

Schema file: lapForce2016\MapForceExamples\Tutorial\books.xsd

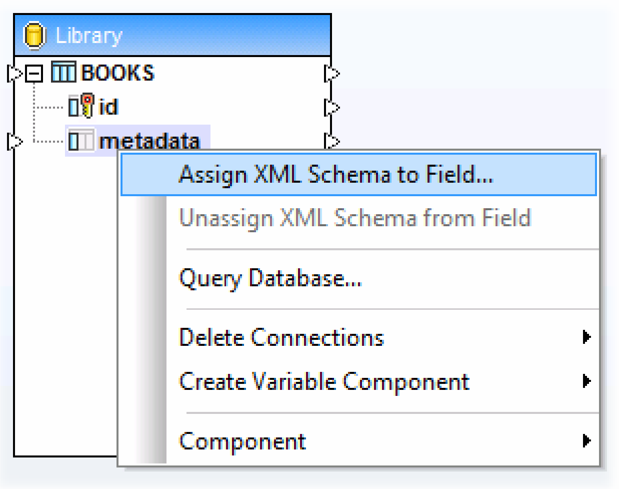
Input XML File: bookentry*.xml

Step 2: Add the SQLite component

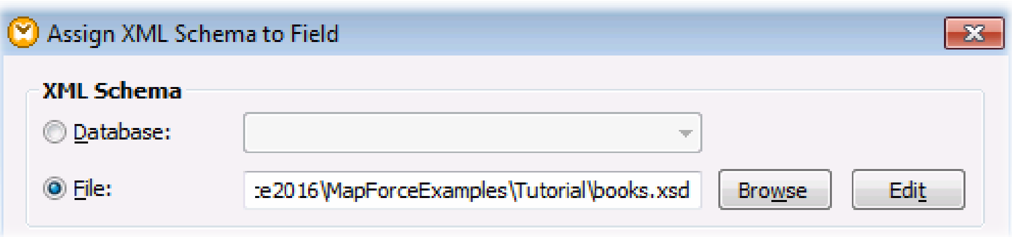
On the **Insert** menu, click **Database**, and follow the wizard to connect to the **Library.sqlite** database file from the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** directory (see also Connecting to an Existing SQLite Database). When prompted to select the database objects, select 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.

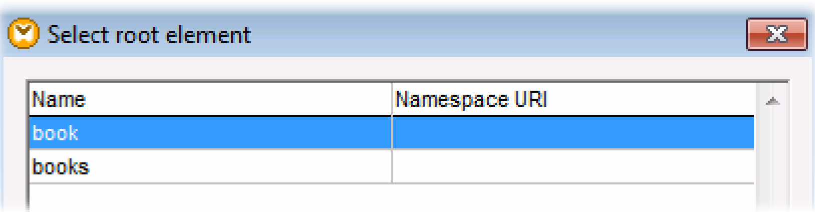

Library
BOOKS
id
metadata
Assign XML Schema to Field...
Unassign XML Schema from Field
Query Database...
Delete Connections
Create Variable Component
Component

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\MapForce2018\MapForceExamples\Tutorial\** directory:



○ Database:
● File: :e2016\MapForceExamples\Tutorial\books.xsd

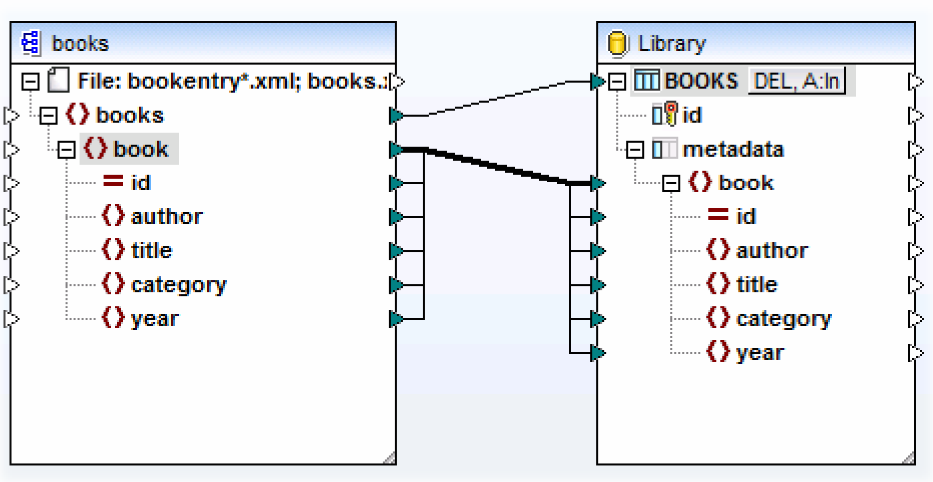
The **books.xsd** schema has two elements with global declaration: `book` and `books`. In this example, we will set `book` as the root element of the XML written to the database field. Click **Choose**, and select `book` as root element:



Name	Namespace URI
book	
books	

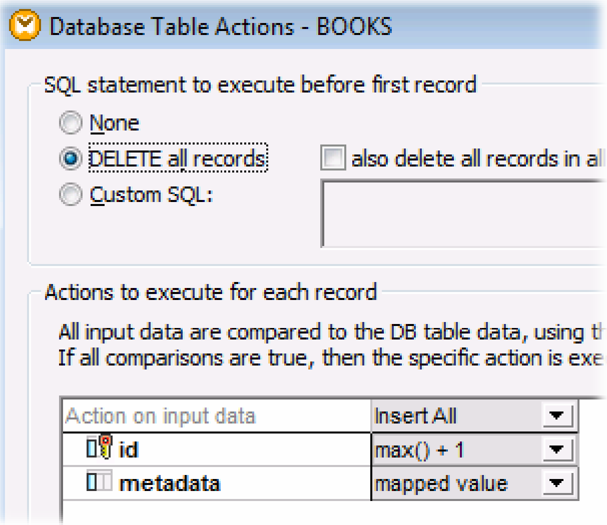
Step 3: Create the mapping connections and configure the database INSERT action

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](#).

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:


Database Table Actions - BOOKS

SQL statement to execute before first record

○ None
● DELETE all records ☐ also delete all records in al
○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using t
If all comparisons are true, then the specific action is exe

Action on input data	Insert All
id	max() + 1
metadata	mapped value

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).

7.2.7.3 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).
3. Paste the following text into the SQL Editor. This SQL query creates a database table called `ARTICLES` and populates it with data.

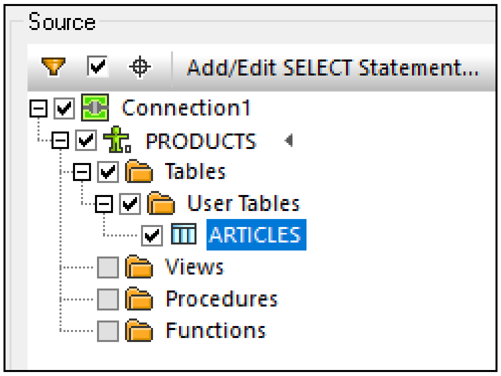
```
-- Create the table
CREATE TABLE
    ARTICLES (
        id INTEGER NOT NULL,
        article XML ) ;
-- Populate the table
INSERT INTO ARTICLES VALUES
   (1, '<Article>
      <Number>1</Number>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>'),
(2, '<Article>
      <Number>2</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.

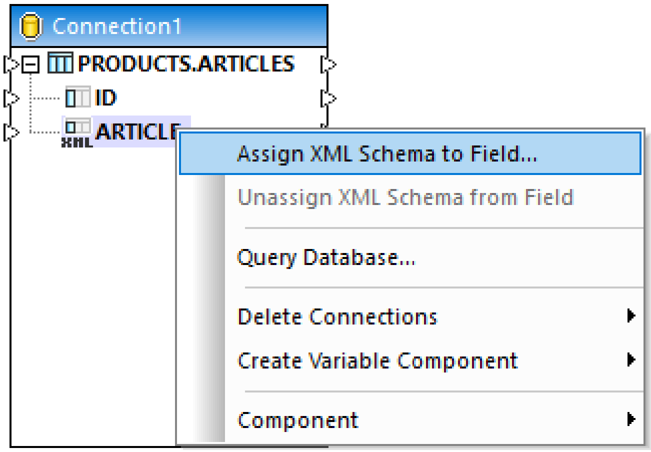
Step 1: Add the database

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

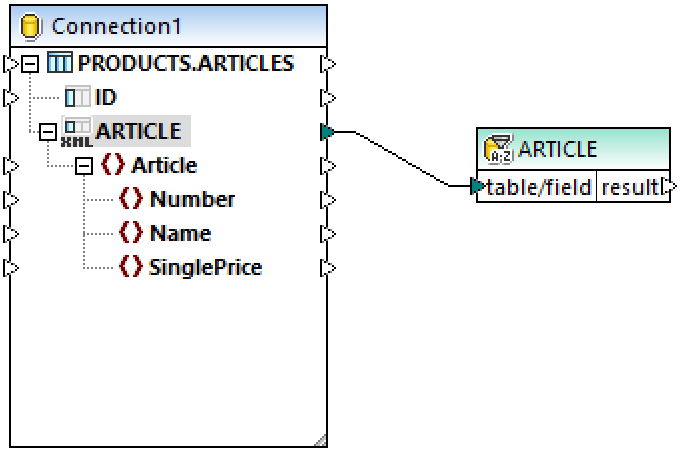
1. Right-click the `ARTICLE` item of the component, and select **Assign XML Schema to field** from the context menu.


Connection1
PRODUCTS.ARTICLES
ID
ARTICLE
XML
Assign XML Schema to Field...
Unassign XML Schema from Field
Query Database...
Delete Connections
Create Variable Component
Component

2. Select **File**, and browse for the following schema: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\DB2xsd.xsd**.

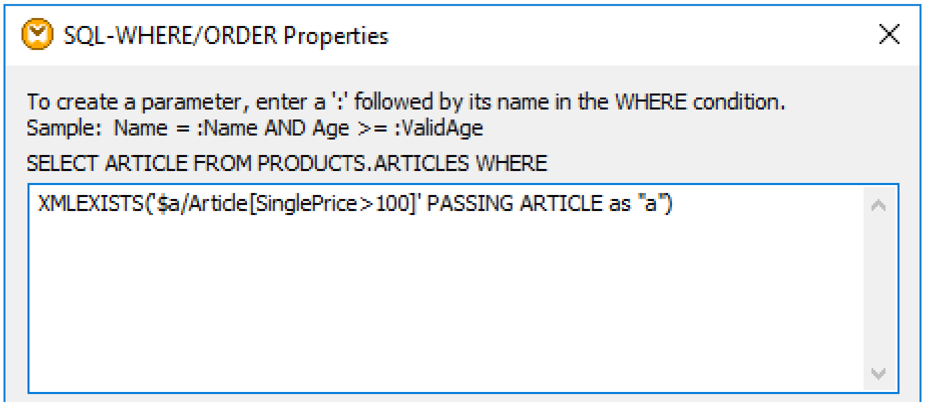
Step 3: Add the SQL WHERE/ORDER component

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")
```



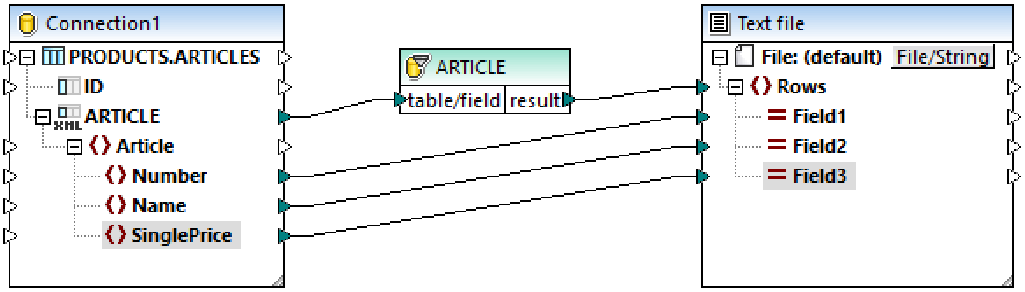
To create a parameter, enter a ':' followed by its name in the WHERE condition.
Sample: Name = :Name AND Age >= :ValidAge
SELECT ARTICLE FROM PRODUCTS.ARTICLES WHERE

```
XMLEXISTS('$a/Article[SinglePrice>100]' PASSING ARTICLE as "a")
```

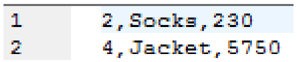
The text above represents the "WHERE" part of the SQL query. At mapping runtime, it will be combined with the "SELECT" part displayed on the dialog box. This statement uses the `XMLEXISTS` function and syntax specific to IBM DB2 databases.

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.

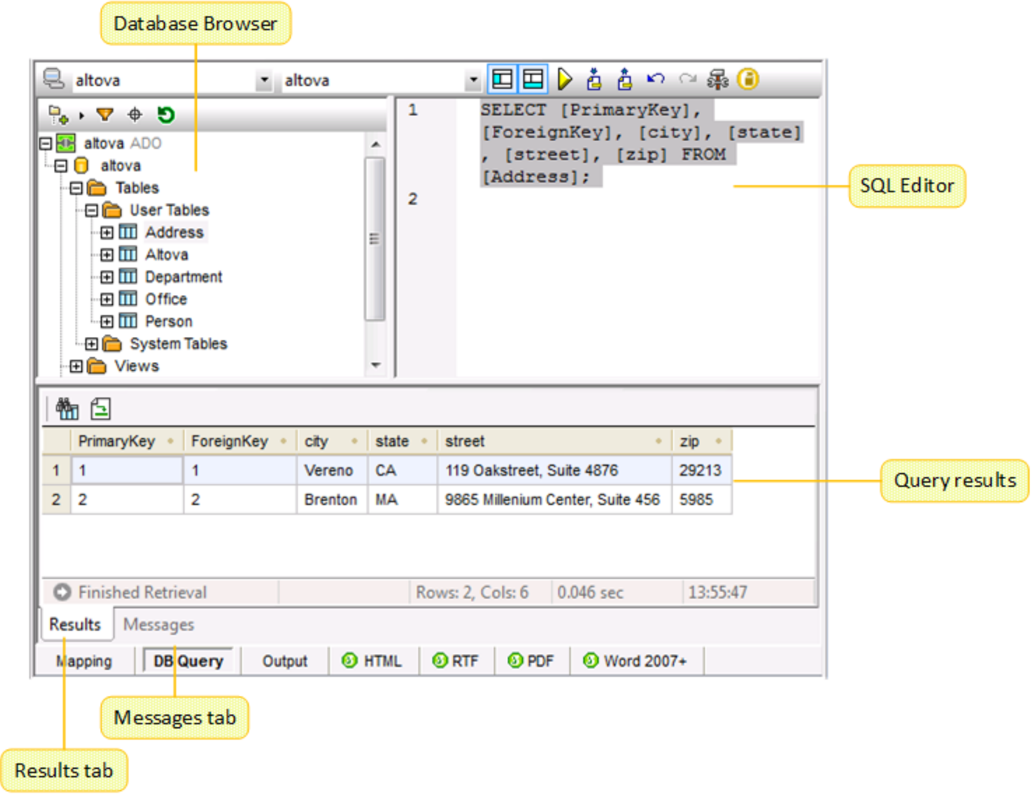


```
1    2,Socks,230
2    4,Jacket,5750
```

7.2.8 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 (see Global Resources).



```
SELECT [PrimaryKey],
[ForeignKey], [city], [state]
, [street], [zip] FROM
[Address];
```

	PrimaryKey	ForeignKey	city	state	street	zip
1	1	1	Vereno	CA	119 Oakstreet, Suite 4876	29213
2	2	2	Brenton	MA	9865 Millenium Center, Suite 456	5985

Finished Retrieval Rows: 2, Cols: 6 0.046 sec 13:55:47

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.

7.2.8.1 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). You can also select an existing database connection from Global Resources, if one has been defined as such (see Global Resources).

Once you are connected to the database, you can create database queries using one of the following methods:

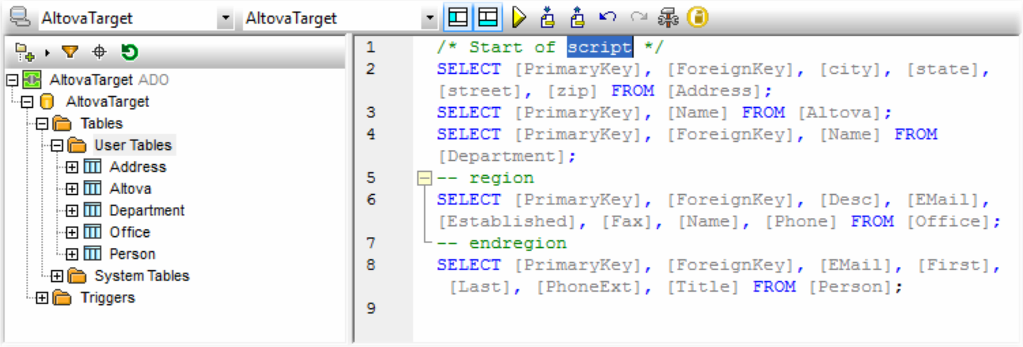
- Import the SQL query into the SQL Editor pane from an existing SQL file.
- Write the query in the SQL Editor pane.
- Right-click an object in the Database Browser pane and generate a query (typically, SELECT).

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 Database Query - Results & Messages tab).

7.2.8.2 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), regions, and line or block comments.



```
/* Start of script */
SELECT [PrimaryKey], [ForeignKey], [city], [state],
[street], [zip] FROM [Address];
SELECT [PrimaryKey], [Name] FROM [Altova];
SELECT [PrimaryKey], [ForeignKey], [Name] FROM
[Department];
-- region
SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail],
[Established], [Fax], [Name], [Phone] FROM [Office];
-- endregion
SELECT [PrimaryKey], [ForeignKey], [EMail], [First],
 [Last], [PhoneExt], [Title] FROM [Person];
```

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

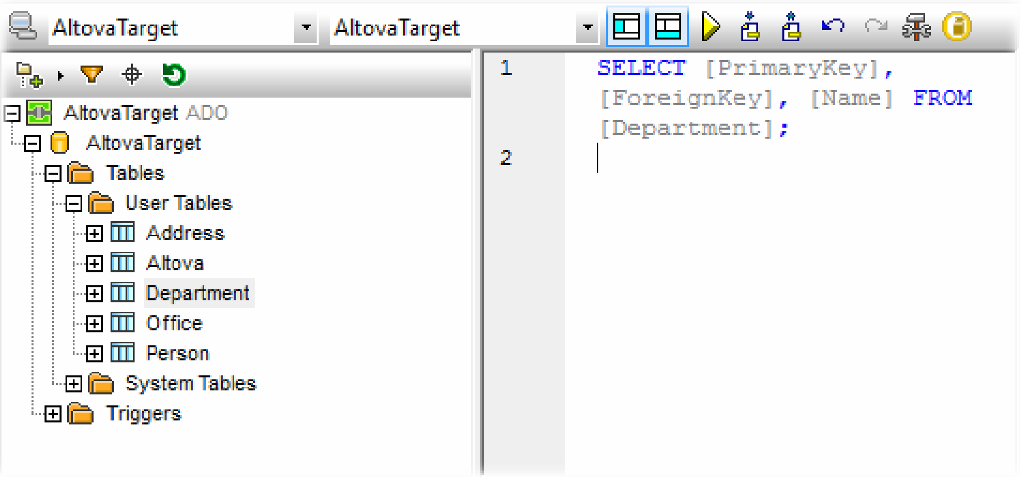
	SQL Editor.
	Options: Opens the Options dialog box allowing you to define general database query settings as well as SQL Editor settings.

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.

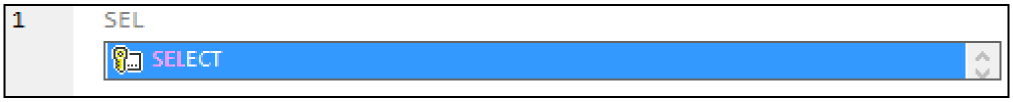


```
SELECT [PrimaryKey],
[ForeignKey], [Name] FROM
[Department];
```

- 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.



```
SEL
```

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).

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).
2. Click the **Execute** (▷) button.

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:

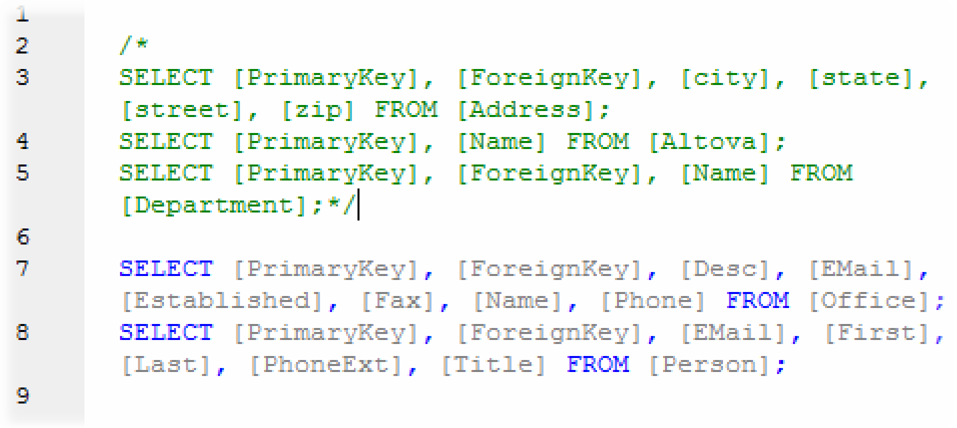
- Click **Import SQL file** ( ), and select the SQL file you want to open.

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:

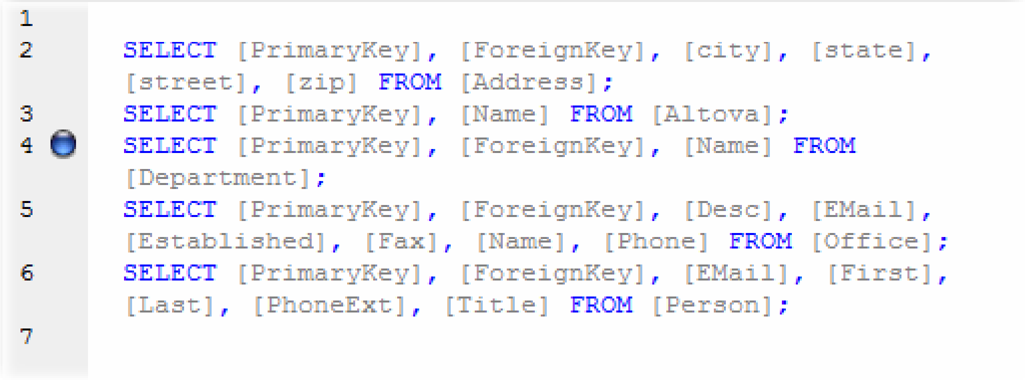
1. Select the part of the statement that is commented out. If you want to remove a line comment, it is sufficient to select only the comment marks -- before the comment.
2. Right-click and select **Insert / Remove Block (or Line) Comment**.

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**.

Inserting Regions

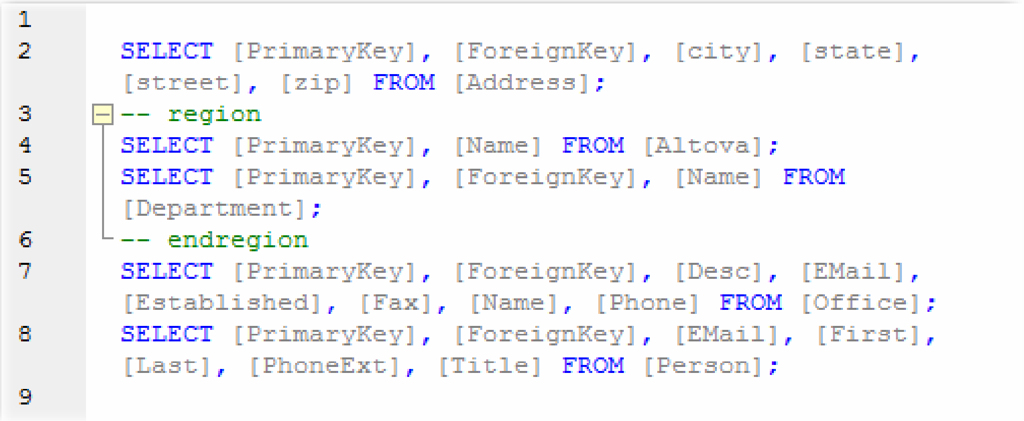
Regions are sections of text that you mark and declare as a unit to structure your SQL scripts. Regions can be collapsed and expanded to display or hide parts of SQL scripts. It is also possible to nest regions within other regions.

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 **Add 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.

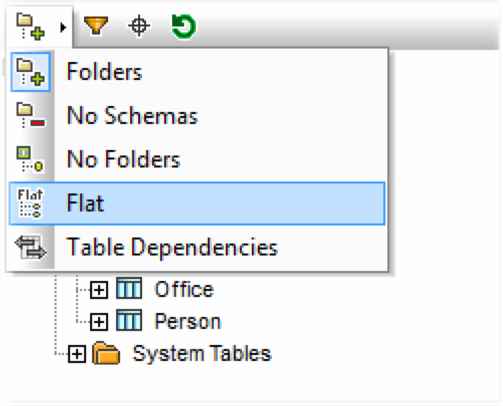
To remove a region:

- Delete the `-- region` and `-- endregion` comments.

7.2.8.3 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).
- Filter and search the displayed database objects (see Filtering and Searching Database Objects).

- 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:

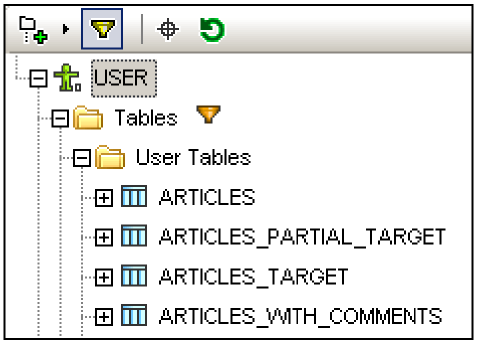
- At the top of the **Database Browser**, click **Refresh** (⟲).

Filtering and Searching Database Objects

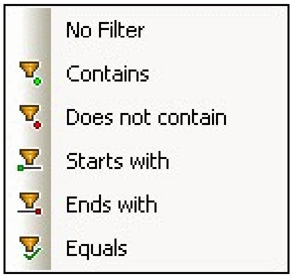
You can filter any database objects (schemas, tables, views, etc) displayed in the **Database Browser** by name or part of a name. Objects are filtered as you type in the characters. Filtering is case-insensitive by default. Filtering is not supported if you have selected the "No Folders" layout.

Filtering database objects

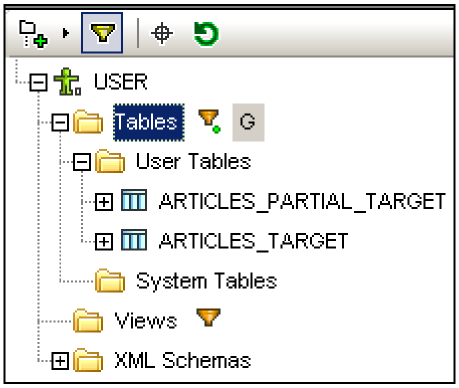
1. At the top of the Database Browser, click **Filter Folder contents** (▽). Filter icons appear next to all folders in the currently selected layout.



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**).



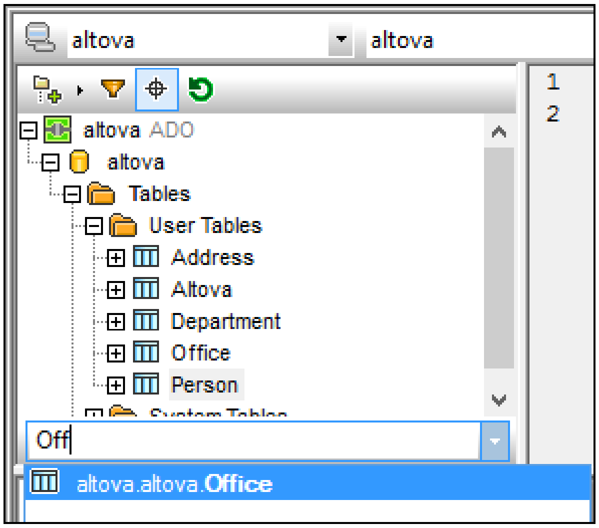
3. In the empty field which appears next to the filter icon, enter the search text (for example, "G"). The results are adjusted as you type.



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**.

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`.

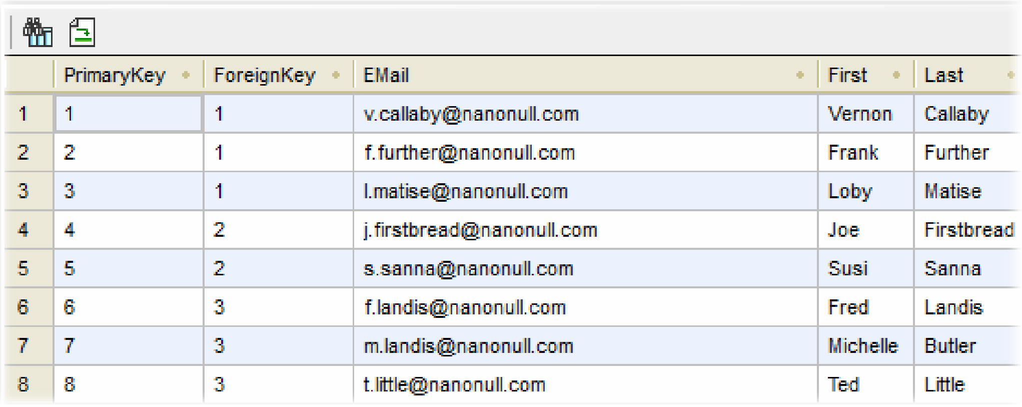
If you selected multiple indexes, the names or paths are printed on separate lines, separated by commas.

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.

7.2.8.4 *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.

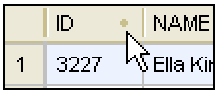
  Note: The context menu can also be used to select data, **Selection | Row | Column | All.**

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



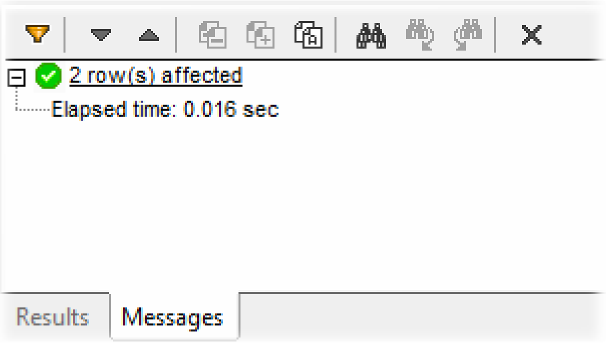
The data is sorted according to the contents of the sorted column.

To restore the default sort order:

- Right-click anywhere in the table and choose **Sorting | Restore default** from the context menu.

7.2.8.5 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.


2 row(s) affected

Elapsed time: 0.016 sec

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.

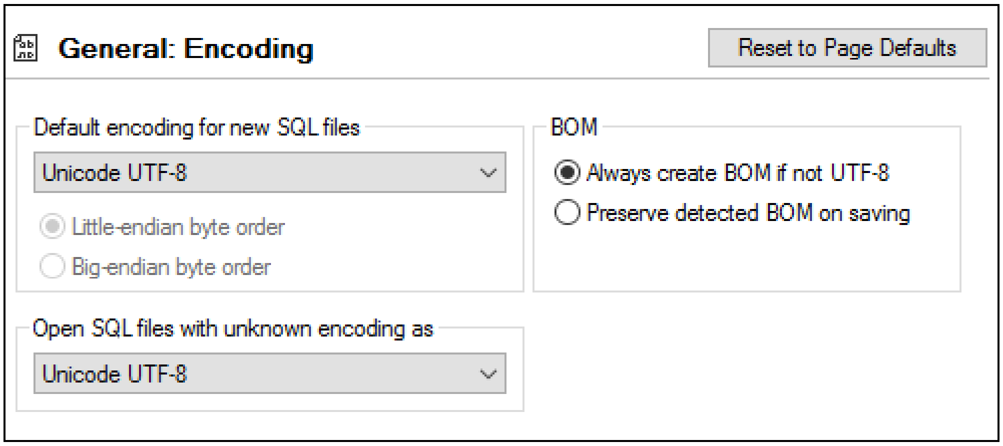
7.2.8.6 *Database Query Settings*

This section includes information about configuring miscellaneous settings applicable to SQL statements entered or loaded in SQL Editor, as well as the query results displayed after a query is executed.

SQL File Encoding Settings

You can specify the encoding options for SQL files created or opened with SQL Editor as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options** , and then click **General | Encoding**.



General: Encoding Reset to Page Defaults

Default encoding for new SQL files: Unicode UTF-8
○ Little-endian byte order
○ Big-endian byte order

BOM
● Always create BOM if not UTF-8
○ Preserve detected BOM on saving

Open SQL files with unknown encoding as: Unicode UTF-8

Default encoding for new SQL files

Define the default encoding for new files so that each new document includes the encoding specification that you specify here. If a two- or four-byte encoding is selected as the default encoding (for example, UTF-16, UCS-2, or UCS-4), you can also choose between little-endian and big-endian byte ordering for the SQL files.

The encoding of existing files is not affected by this setting.

Open SQL files with unknown encoding as

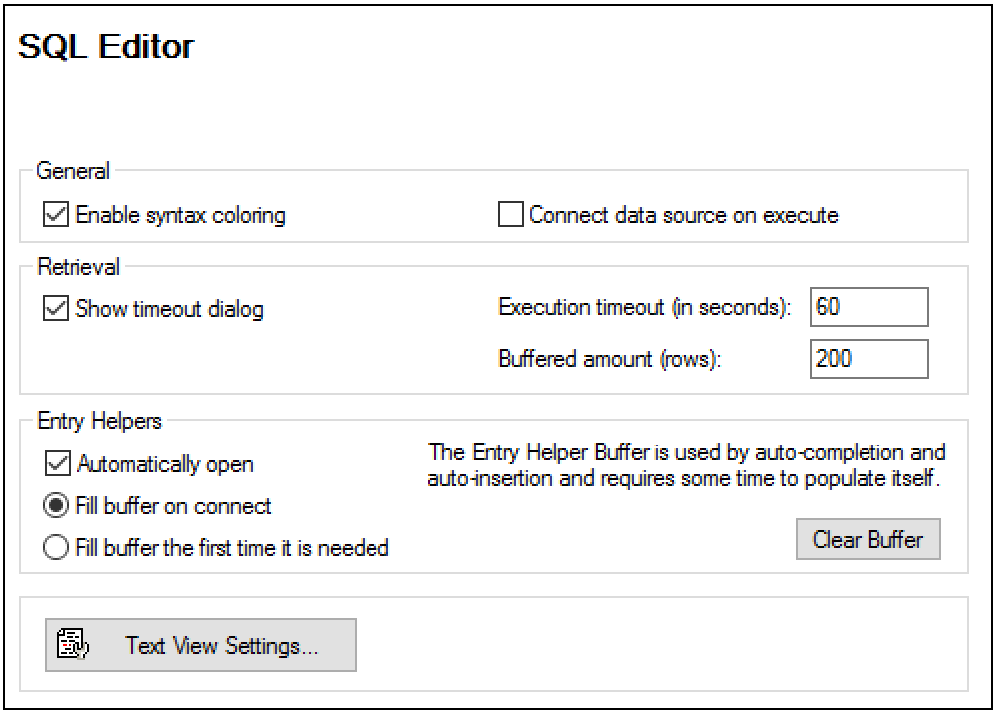
You can select the encoding with which to open an SQL file with no encoding specification or where the encoding cannot be detected.

Note: SQL files which have no encoding specification are saved with a UTF-8 encoding**.**

SQL Editor General Settings

You can change the general settings applicable to the SQL Editor as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options** , and then click **SQL Editor**.



SQL Editor

General

☑ Enable syntax coloring

□ Connect data source on execute

Retrieval

☑ Show timeout dialog

Execution timeout (in seconds): 60

Buffered amount (rows): 200

Entry Helpers

☑ Automatically open

● Fill buffer on connect

○ Fill buffer the first time it is needed

The Entry Helper Buffer is used by auto-completion and auto-insertion and requires some time to populate itself.

General

Syntax coloring emphasizes different elements of SQL syntax using different colors.

Activate the **Connect datasource on execute** check box to connect to the corresponding data source automatically whenever a SQL statement is executed and its data source is not connected.

Retrieval

Specify the maximum amount of time permissible for SQL execution (Execution timeout) in seconds.

Activating the **Show timeout dialog** check box allows you to change the time-out settings when the permissible execution period is exceeded.

Entry Helper Buffer

To enable auto-completion suggestions as you start typing SQL statements, select the **Automatically open** check box (see also Auto-Completion).

The entry helper buffer for auto-completion can be filled either when you connect to a data source or when it is used for the first time. Note that filling the buffer may take some time. Use the **Clear Buffer** button to reset the buffer.

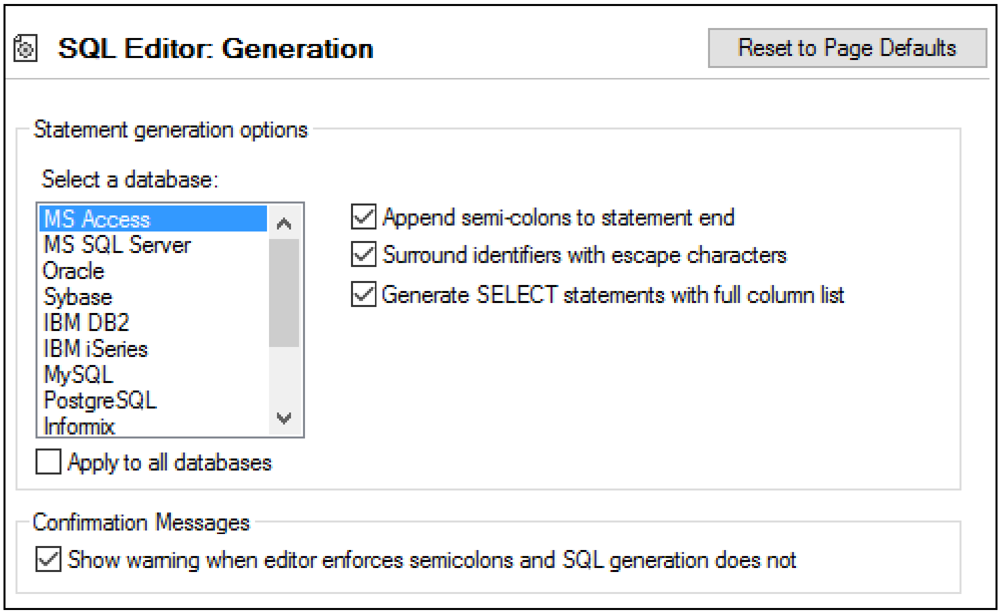
Text View Settings

Allows you to define the specific Text view settings: Margins, Tabs, Visual aids, as well as showing you the Text view navigation hotkeys.

SQL Statement Generation Settings

You can specify the SQL statement generation syntax for various database kinds as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options** , and then click **SQL Editor**.



SQL Editor: Generation

Statement generation options

Select a database:

- MS Access
- MS SQL Server
- Oracle
- Sybase
- IBM DB2
- IBM iSeries
- MySQL
- PostgreSQL
- Informix

☑ Append semi-colons to statement end
☑ Surround identifiers with escape characters
☑ Generate SELECT statements with full column list

☐ Apply to all databases

Confirmation Messages

☑ Show warning when editor enforces semicolons and SQL generation does not

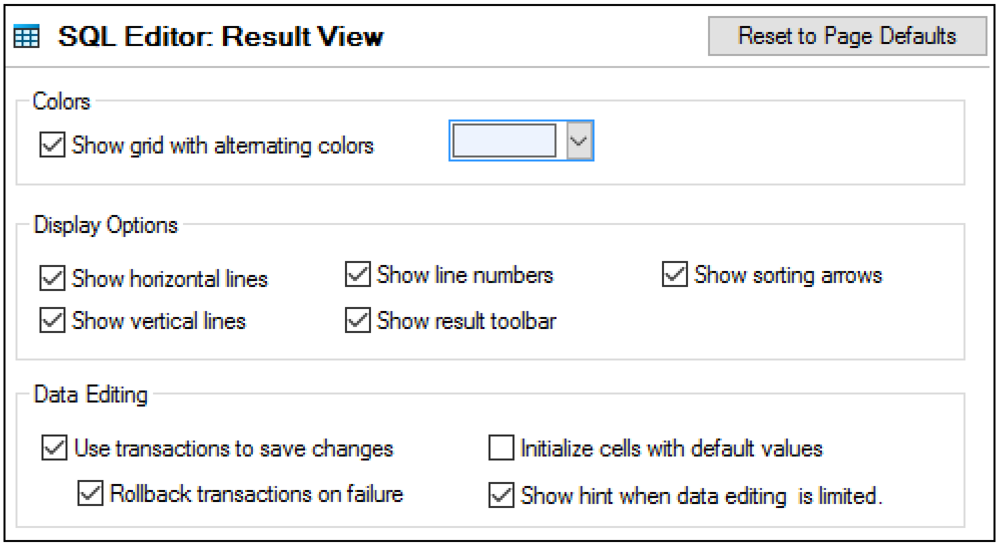
To define the syntax preferences for a specific database, select it from the list, and then enable or disable the three check boxes to the right.

To define a unique syntax for all databases, select **Apply to all databases**, and then enable or disable the three check boxes to the right. Note that using common settings for all databases may cause inability to edit data in Oracle and IBM DB2 and iSeries databases via a JDBC connection.

Query Result View Settings

You can configure the appearance of the **Results** tab of the **DB Query** pane as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options** , and then click **SQL Editor | Result View**.



SQL Editor: Result View

Colors

☑ Show grid with alternating colors

Display Options

☑ Show horizontal lines ☑ Show line numbers ☑ Show sorting arrows

☑ Show vertical lines ☑ Show result toolbar

Data Editing

☑ Use transactions to save changes ☐ Initialize cells with default values

☑ Rollback transactions on failure ☑ Show hint when data editing is limited.

Select the **Show grid with alternating colors** check box to display rows in Result tabs as simple grid or with alternating white and colored rows. The alternating color is configurable.

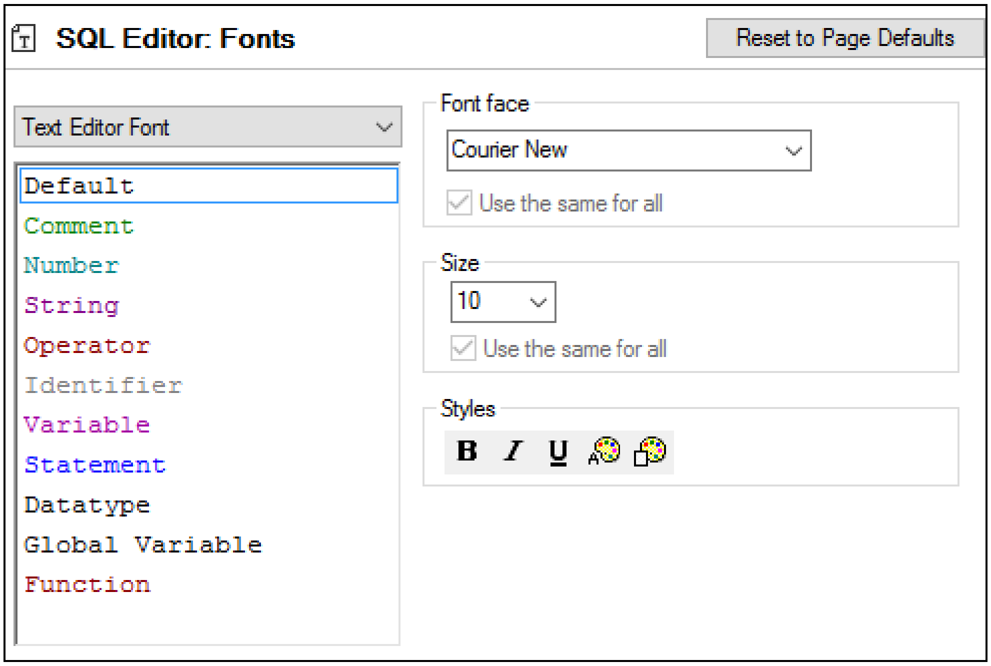
The **Display Options** group lets you define how horizontal and vertical grid lines, as well as line numbers and the **Result** toolbar, are displayed. You can switch any of these options off by deactivating the respective check box.

The **Data Editing** group lets you define the 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.

SQL Editor Font Settings

You can configure color and font settings of SQL statements that appear in SQL Editor as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options** , and then click **SQL Editor | Fonts**.



SQL Editor: Fonts

Reset to Page Defaults

Text Editor Font

Default
Comment
Number
String
Operator
Identifier
Variable
Statement
Datatype
Global Variable
Function

Font face: Courier New

☑ Use the same for all

Size: 10

☑ Use the same for all

Styles: B I U

The font settings listed in the Font Settings list box are elements of SQL statements. You can choose the common font face, style, and size of all text that appears in SQL Editor. Note that the same font and size is used for all text types.

Only the style can be changed for individual text types. This enables the syntax coloring feature. Click the **Reset to default** button to restore the original settings.

7.2.9 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, e.g. send e-mails.

Stored procedures in MapForce:

- Can be present (and called) in both source and target database components.
- Can have data be mapped to them by input parameters, as well as mapped from them, by output parameters.
- Can be inserted as a function-like call. This allows you to provide input data, execute the stored procedure, and read/map the output data to other components.
- Are visible with their unique name and a clickable button, inside the database component once the database has been inserted into the mapping area.
- Cannot be edited from within MapForce
- Can only be used in the BUILT-IN execution engine. Code generation in C++, C#, or Java is not supported.

Note: To illustrate how MapForce implements stored procedures, this chapter uses Microsoft SQL Server 2008 and the "AdventureWorks" database. The latter can be downloaded from the CodePlex website (http://sqlserversamples.codeplex.com).

Support notes

- 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.

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. • 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. • 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

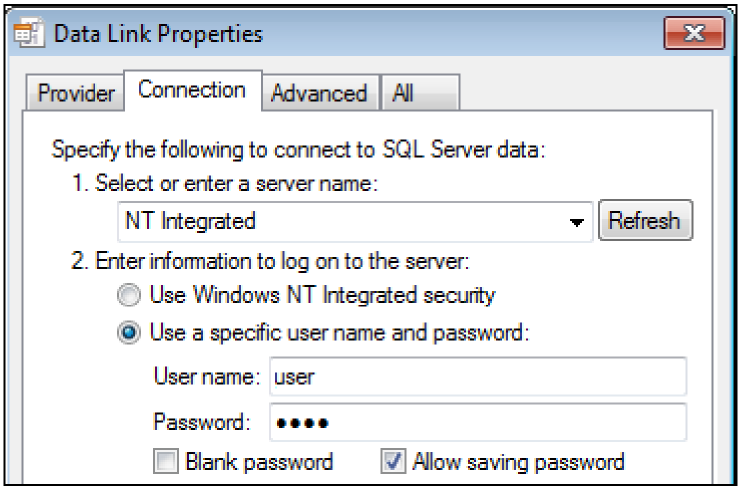
Database	Support notes
	and number of recordsets is therefore always fixed for Oracle stored procedures.
PostgreSQL	- Supported in MapForce: scalar functions, row-valued functions, table-valued functions. - In PostgreSQL, any output parameters defined in a function describe the columns of the result 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	- 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.
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.

7.2.9.1 Inserting stored procedures in database components

Stored procedures can be incorporated into a database component when inserting it into the mapping area. This follows the usual sequence of inserting a database component into MapForce.

To insert a database component containing stored procedures:

1. Click the Insert Database icon, or select the menu option **File | Insert Database**.
2. Use the Connection Wizard to connect to the database.
3. Having filled in the Connection tab of the Data Link Properties dialog box, click the OK button.



Data Link Properties

Specify the following to connect to SQL Server data:

1. Select or enter a server name:

NT Integrated

2. Enter information to log on to the server:

○ Use Windows NT Integrated security

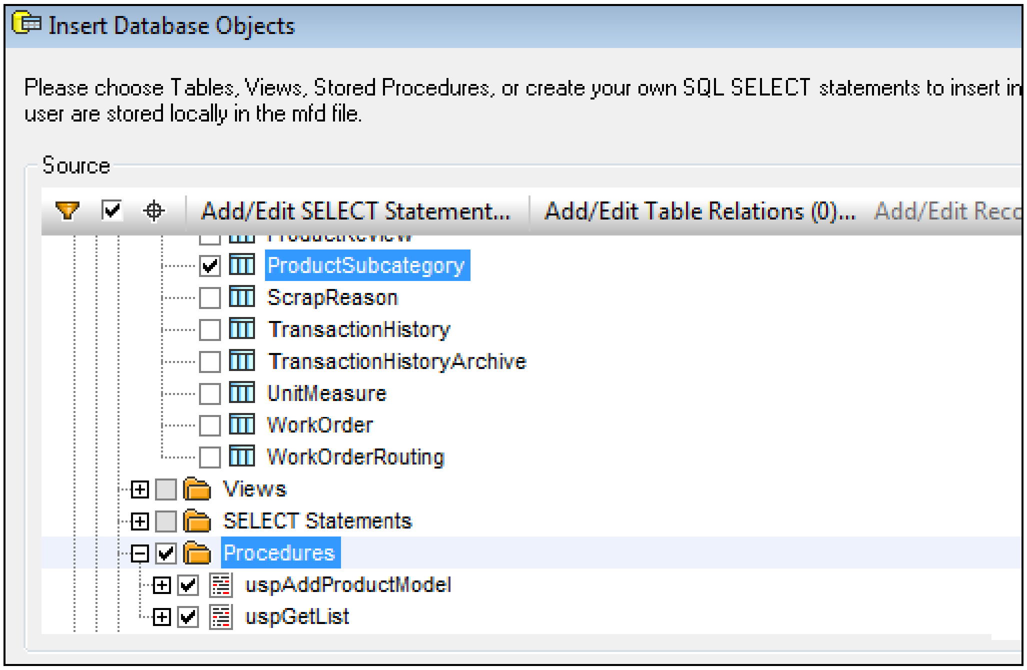
● Use a specific user name and password:

User name: user

Password: ••••

□ Blank password ☑ Allow saving password

4. Click the expand button to select the database tables you want to insert, and select the specific tables.



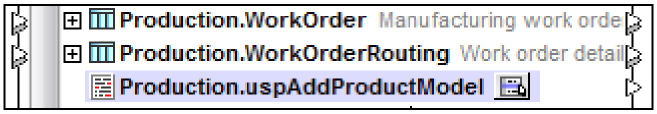
Insert Database Objects

Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert in user are stored locally in the mfd file.

Source

☑ ProductSubcategory
□ ScrapReason
□ TransactionHistory
□ TransactionHistoryArchive
□ UnitMeasure
□ WorkOrder
□ WorkOrderRouting
□ Views
□ SELECT Statements
☑ Procedures
☑ uspAddProductModel
☑ uspGetList

5. Click the expand button of the Procedures folder to select the stored procedures that you want to insert along with the tables, then click OK.



Production.WorkOrder Manufacturing work orde

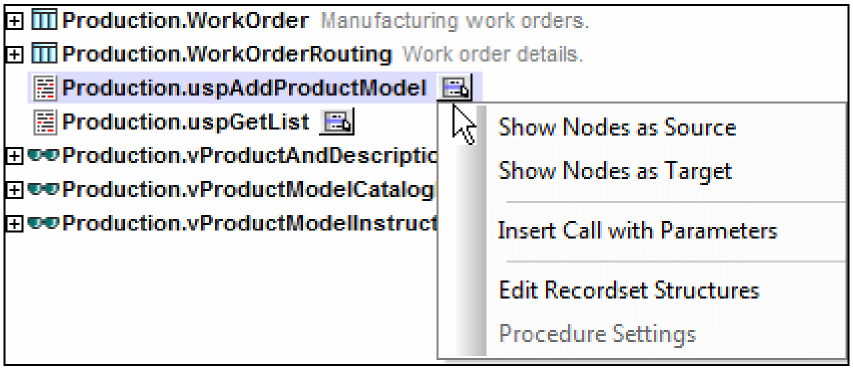
Production.WorkOrderRouting Work order detail

Production.uspAddProductModel

The database component is inserted and shows the selected tables followed by the stored procedures that you selected.

- Tables, views and procedures are sorted alphabetically in the database component.
- Each stored procedure is shown as an item in the database component containing the procedure name and a clickable button. The button allows you to select if the procedure is to be used as a source or target, as well as other procedure settings.

- At this point, MapForce has no specific information if the parameters of the stored procedure are to be used as source or target parameters. This is achieved by clicking the stored procedure button and selecting the specific option.


Production.WorkOrder Manufacturing work orders.
Production.WorkOrderRouting Work order details.
Production.uspAddProductModel
Production.uspGetList
Production.vProductAndDescriptic
Production.vProductModelCatalog
Production.vProductModelInstruct
Show Nodes as Source
Show Nodes as Target
Insert Call with Parameters
Edit Recordset Structures
Procedure Settings

7.2.9.2 Use cases

The following uses cases should cover most common types of stored procedures and how to define them in MapForce.

I want to:	Read this section
I want to call a stored procedure to **retrieve** data from a database and map it to another component. E.g. I want to use a stored procedure as a data source to write the resulting data into another file (XML, TXT, EDI, etc.).	Stored procedures in Source components
I want to call a stored procedure to **modify** the database or perform another specific action.	Stored procedures in Target components
I want to use stored procedures to **generate** one or more values/keys for an Insert statement in the same database.	Using stored procedures to generate primary keys

7.2.9.3 Stored procedures and local relations

By using local relations (see Defining Local Relationships), you can define a hierarchical order in which to call stored procedures or perform actions (insert, update, ...) on database tables. They can be used in source and target components.

A local relation always has a parent object (containing a primary/unique key) and a child object (containing a foreign key).

Possible parent objects and their fields used in a relation are:
- Database table or view (column)
- Stored procedure (output parameter or return value)

- Recordset of a stored procedure (column) - only for source and procedure call components
- User-defined SELECT statement (column)

Possible child objects and their fields are:

- Database table or view (column, produces a WHERE condition)
- Stored procedure (input parameter)
- User-defined SELECT statement (input parameter)

In source components, this makes it easy to read data from related objects, e.g. 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 allow defining a clear order in which multiple related procedures are to be called, e.g. one that creates an ID value, and another that inserts related information into another table. It is also possible to mix stored procedures and tables in local relations, e.g. perform the insert directly on the related table instead of calling another procedure.

7.2.9.4 Stored procedures as a data source

The output of a stored procedure can be zero or more output or return parameters, and zero or more recordsets from SELECT statements embedded inside the stored procedure. A recordset or result set is the output of such a SELECT statement, similar to a table or view. Output parameters and recordsets can be mapped to target components.

The column structures of these recordsets cannot be directly read from the database catalog, they must therefore be detected by executing the stored procedure at design time or by being defined manually - see Defining recordsets for details.

Depending on whether the stored procedure has input parameters or not, the handling in MapForce is different:

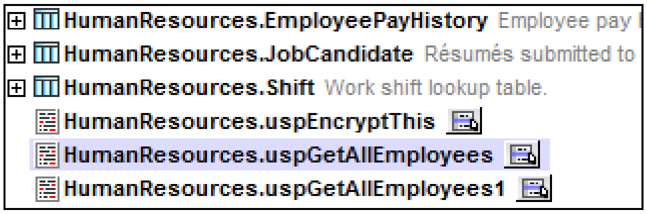
The stored procedure has no input parameters	Stored procedures without input parameters
I want to supply the values for the procedure's input parameters by mapping from an XML, Text, or other type of file, or from mapping input parameters or constants	Call with parameters - input and output
I want to supply the values for the procedure's input parameters from a table or view in the same database, or from the output of another stored procedure	Source components and Local Relations

Stored procedures without input parameters

Use this option (for example) if you want to use the stored procedure in a source component **without** having any **input** parameters.

E.g. this could be a stored procedure that is a pure SELECT-type query without any input parameters, where you want to map the result of the SELECT statement to a target component.

E.g. HumanResources.uspGetAllEmployees of the AdventureWorks database.


HumanResources.EmployeePayHistory Employee pay
HumanResources.JobCandidate Résumés submitted to
HumanResources.Shift Work shift lookup table.
HumanResources.uspEncryptThis
HumanResources.uspGetAllEmployees
HumanResources.uspGetAllEmployees1

Stored procedure:

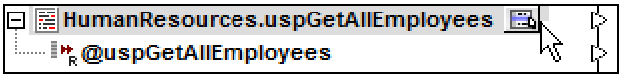
```
PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment;
```

The columns of the recordset cannot be directly read from the database catalog by MapForce, they must therefore be detected by executing the stored procedure once or by being defined manually.

Defining the output recordset of a source component:

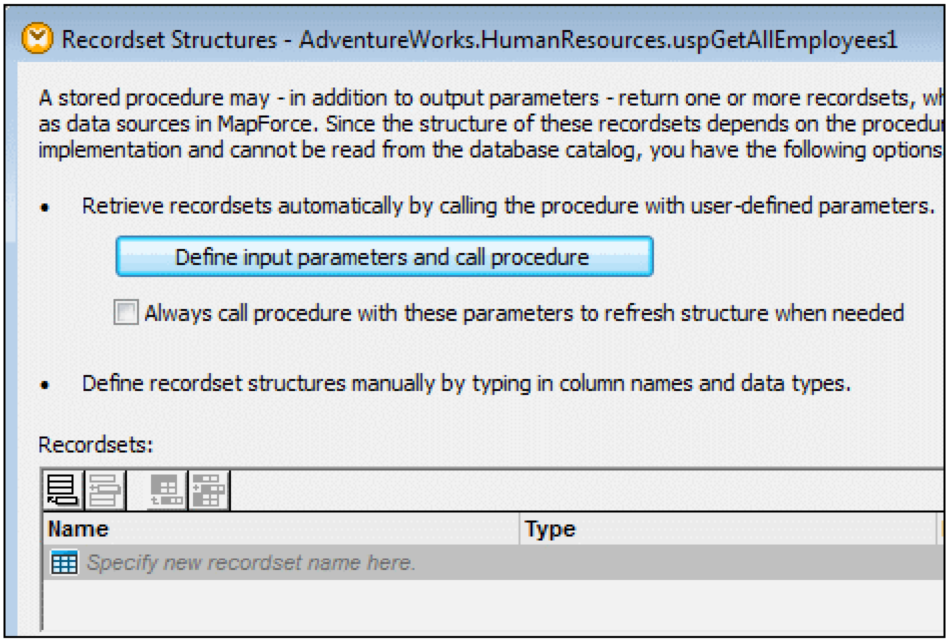
Having inserted the AdventureWorks database component and selected the HumanResources tables and included stored procedures:

1. Click the "stored procedure" button and select the option "Show nodes as Source".



The Return value node is inserted below the stored procedure name. Since we want to read the recordset and not the return value, click the stored procedure button again and select **Edit RecordSet structures**.

2. Click the "Define input parameters and call procedure" button.



Recordset Structures - AdventureWorks.HumanResources.uspGetAllEmployees1

A stored procedure may - in addition to output parameters - return one or more recordsets, wh as data sources in MapForce. Since the structure of these recordsets depends on the procedu implementation and cannot be read from the database catalog, you have the following options

- Retrieve recordsets automatically by calling the procedure with user-defined parameters.

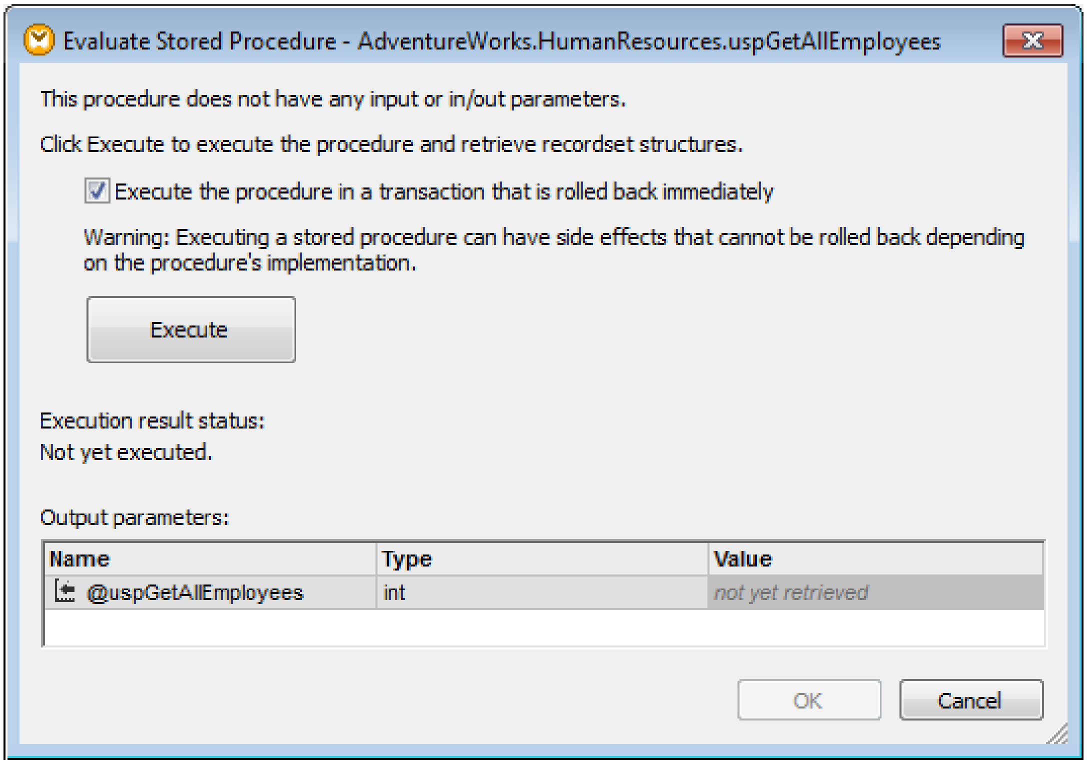
□ Always call procedure with these parameters to refresh structure when needed

- Define recordset structures manually by typing in column names and data types.

Recordsets:

Name	Type
Specify new recordset name here.	

This opens the "Evaluate Stored Procedure" dialog box.



Evaluate Stored Procedure - AdventureWorks.HumanResources.uspGetAllEmployees

This procedure does not have any input or in/out parameters.

Click Execute to execute the procedure and retrieve recordset structures.

☑ Execute the procedure in a transaction that is rolled back immediately

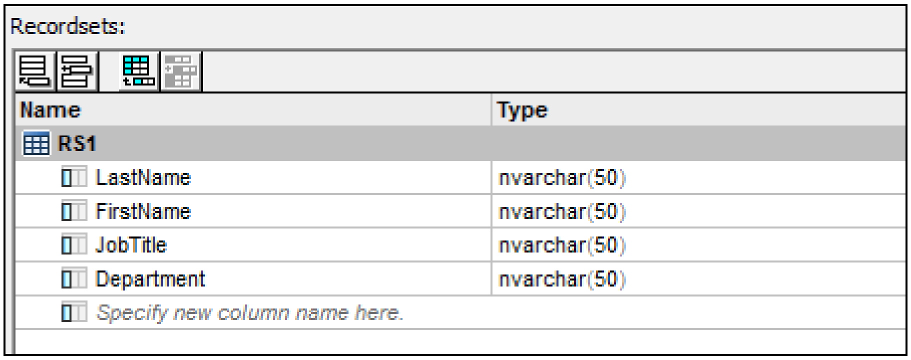
Warning: Executing a stored procedure can have side effects that cannot be rolled back depending on the procedure's implementation.

Execution result status:
Not yet executed.

Output parameters:

Name	Type	Value
@uspGetAllEmployees	int	*not yet retrieved*

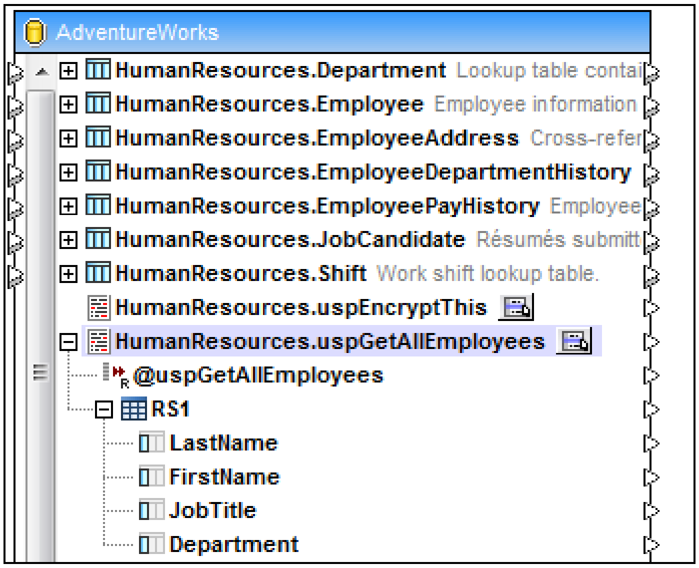
3. Click the "Execute" button, then click OK.
The recordset fields are now visible in the Recordsets section of the dialog box.



Recordsets:

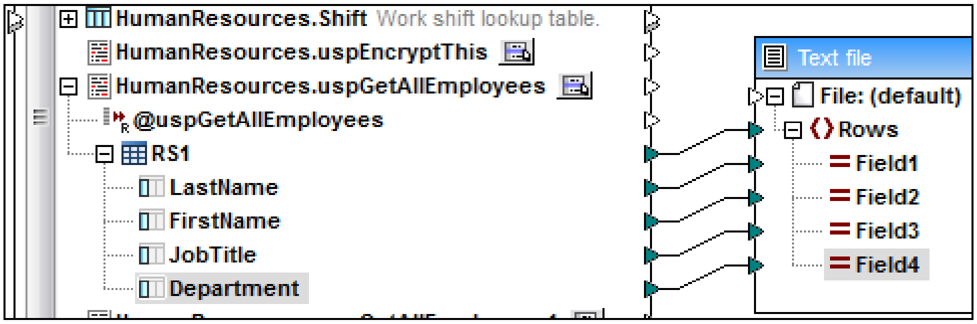
Name	Type
RS1	
LastName	nvarchar(50)
FirstName	nvarchar(50)
JobTitle	nvarchar(50)
Department	nvarchar(50)
Specify new column name here.	

4. Click the OK button again to complete the recordset definition.
   The columns, LastName etc., are shown as nodes below the recordset node **RS1**. (Click the "+" button to expand the recordset if not visible).

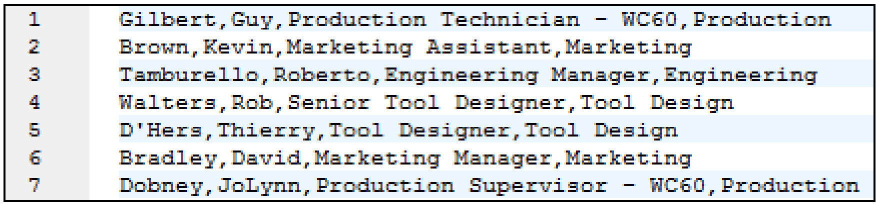

AdventureWorks
HumanResources.Department Lookup table contai
HumanResources.Employee Employee information
HumanResources.EmployeeAddress Cross-refer
HumanResources.EmployeeDepartmentHistory
HumanResources.EmployeePayHistory Employee
HumanResources.JobCandidate Résumés submitt
HumanResources.Shift Work shift lookup table.
HumanResources.uspEncryptThis
HumanResources.uspGetAllEmployees
@uspGetAllEmployees
RS1
LastName
FirstName
JobTitle
Department

Completing the mapping:

1. Insert a Text file component and map the output icons to the text file.



2. Click the Output button to see the result.



```
Gilbert,Guy,Production Technician - WC60,Production
Brown,Kevin,Marketing Assistant,Marketing
Tamburello,Roberto,Engineering Manager,Engineering
Walters,Rob,Senior Tool Designer,Tool Design
D'Hers,Thierry,Tool Designer,Tool Design
Bradley,David,Marketing Manager,Marketing
Dobney,JoLynn,Production Supervisor - WC60,Production
```

Note:
If executing the stored procedure has side effects (depending on the procedure implementation) that you want to avoid at design time, recordsets can be also be defined manually in the Recordset Structures dialog box, by adding recordsets and their associated columns. Click the Add recordset, or Add column buttons in the Recordset Structures dialog box.

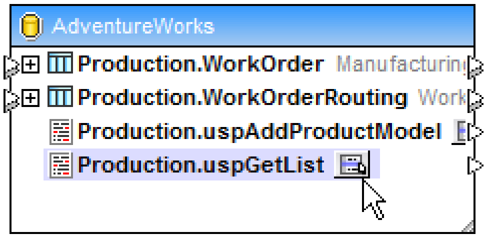
Call with parameters - input and output

Stored procedures can also be used as a function-like call. This allows you to:

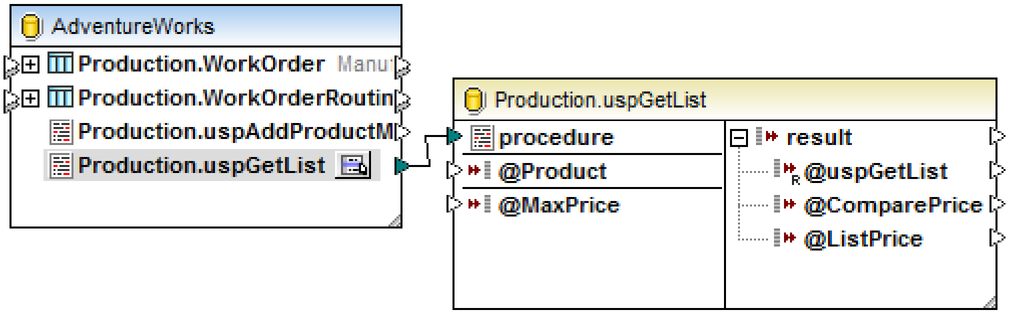
- provide input data to the procedure
- execute the procedure
- map the procedure output data to other components

To use a stored procedure as a function-like call:
Having inserted the AdventureWorks database component and selected the Production tables and included stored procedures:

1. Click the "stored procedure" button of Production.uspGetList, to open the menu.


AdventureWorks
Production.WorkOrder Manufacturing
Production.WorkOrderRouting Work
Production.uspAddProductModel
Production.uspGetList

2. Select the option "Insert Call with Parameters".

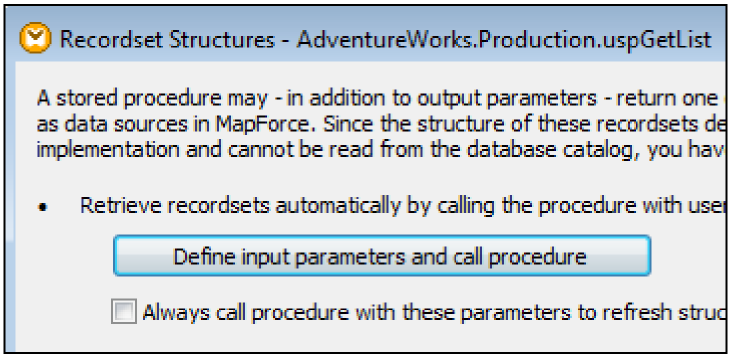


This inserts the procedure component into the mapping. The component looks and works similar to a web service, or user-defined, component. The procedure name is automatically connected to the "procedure" item of the component.

The procedure input parameters are shown on the left, while the output parameters are shown at right. This particular stored procedure returns output parameters and also a recordset, however we must define its structure before we can see and use it in MapForce:

To define the recordset structure:

1. Click the "stored procedure" button, of uspGetList, and select "Edit recordset structures".
2. Click the "Define input parameters and call procedure" button, then click Execute in the dialog box that opens.



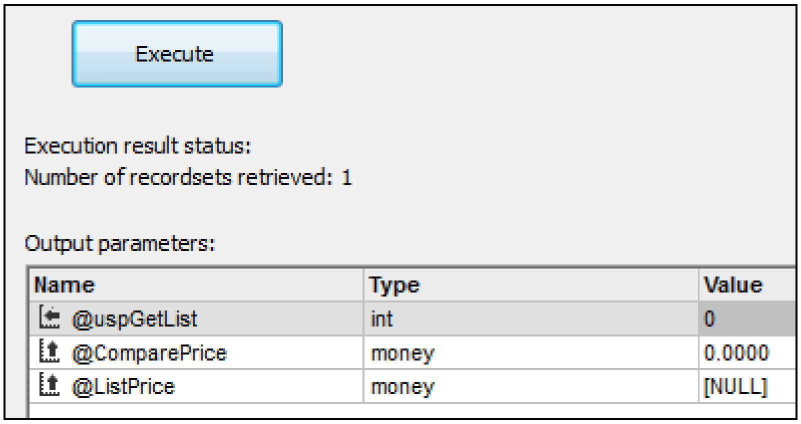
Recordset Structures - AdventureWorks.Production.uspGetList

A stored procedure may - in addition to output parameters - return one
as data sources in MapForce. Since the structure of these recordsets de
implementation and cannot be read from the database catalog, you hav

- Retrieve recordsets automatically by calling the procedure with user

□ Always call procedure with these parameters to refresh struc

This writes the returned output parameter values into the table below and displays that one recordset was retrieved.

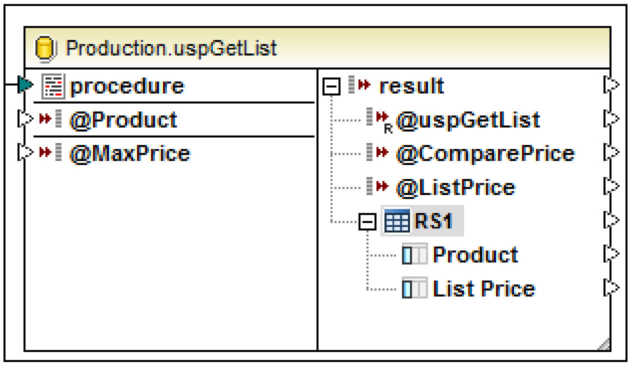


Execution result status:
Number of recordsets retrieved: 1

Output parameters:

Name	Type	Value
@uspGetList	int	0
@ComparePrice	money	0.0000
@ListPrice	money	[NULL]

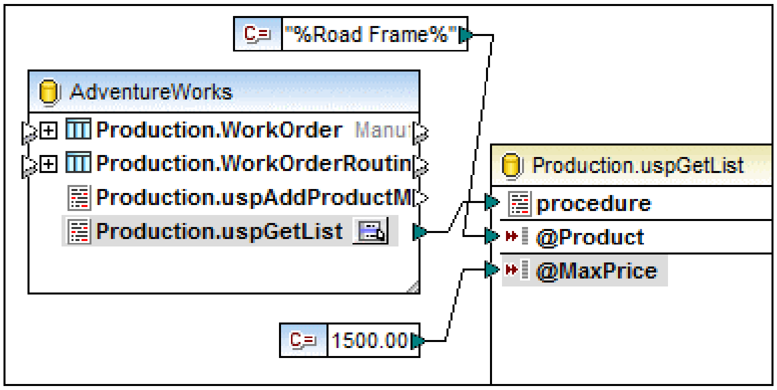
3. Click the OK button to confirm, then click OK to close the Recordset dialog box.



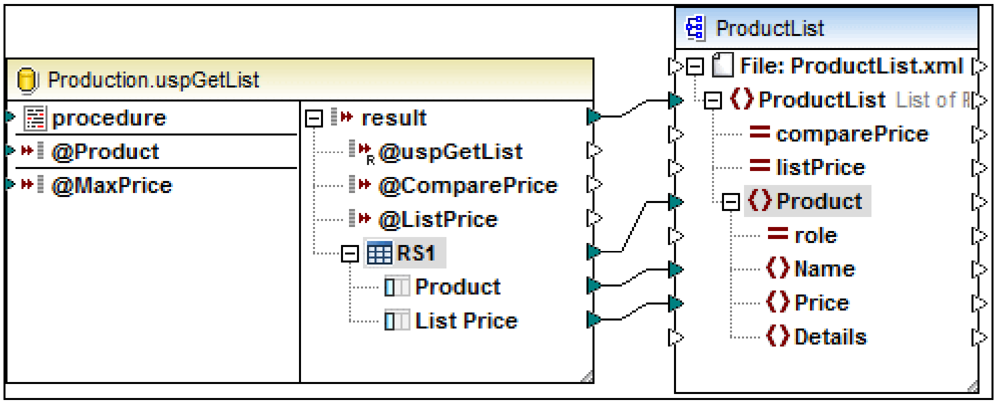
The recordset has been added to the output section of the stored procedure component.

Using the call parameter component:

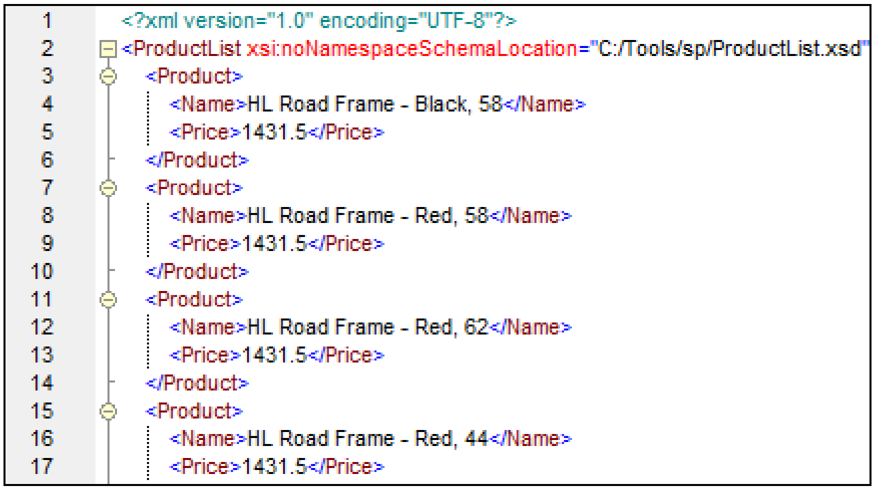
1. Define the components you want to use to supply the input parameters, e.g. two constant components as shown in the screen shot, and connect them to the input parameters.



2. Define and insert the target component which will be used to contain the stored procedure output, e.g. an XML document as shown below.



3. Click the Output button to see the result of the mapping.



```
<?xml version="1.0" encoding="UTF-8"?>
<ProductList xsi:noNamespaceSchemaLocation="C:/Tools/sp/ProductList.xsd"
  <Product>
    <Name>HL Road Frame - Black, 58</Name>
    <Price>1431.5</Price>
  </Product>
  <Product>
    <Name>HL Road Frame - Red, 58</Name>
    <Price>1431.5</Price>
  </Product>
  <Product>
    <Name>HL Road Frame - Red, 62</Name>
    <Price>1431.5</Price>
  </Product>
  <Product>
    <Name>HL Road Frame - Red, 44</Name>
    <Price>1431.5</Price>
```

The various road frame products are listed.

Source components and Local Relations

Use this option if you want to combine data supplied by a stored procedure recordset with data from another table, to which there is no direct relationship in the database.

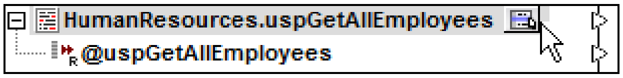
```
PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment;
```

The columns of the recordset cannot be directly read from the database catalog by MapForce, they must therefore be detected by executing the stored procedure once or by being defined manually.

Defining the output recordset of a source component:

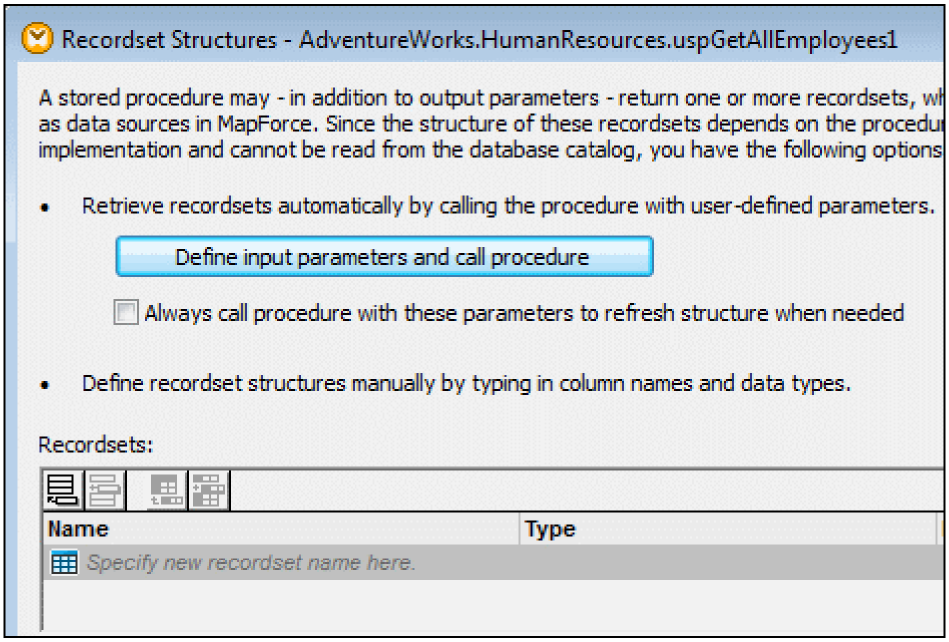
Having inserted the AdventureWorks database component and selected the HumanResources tables and included stored procedures:

1. Click the "stored procedure" button and select the option "Show nodes as Source".



The Return value node is inserted below the stored procedure name. Since we want to read the recordset and not the return value, click the stored procedure button again and select **Edit RecordSet structures**.

2. Click the "Define input parameters and call procedure" button.


Recordset Structures - AdventureWorks.HumanResources.uspGetAllEmployees1

A stored procedure may - in addition to output parameters - return one or more recordsets, wh
as data sources in MapForce. Since the structure of these recordsets depends on the procedu
implementation and cannot be read from the database catalog, you have the following options

- Retrieve recordsets automatically by calling the procedure with user-defined parameters.

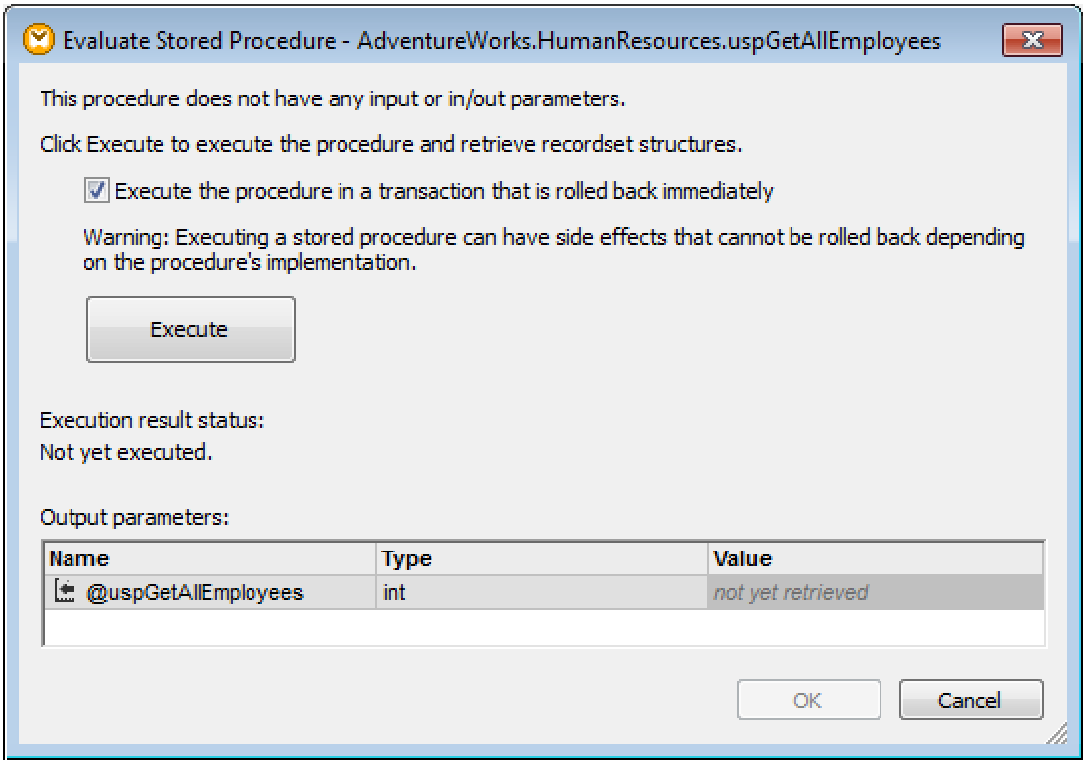
□ Always call procedure with these parameters to refresh structure when needed

- Define recordset structures manually by typing in column names and data types.

Recordsets:

Name	Type
Specify new recordset name here.	

This opens the "Evaluate Stored Procedure" dialog box.



Evaluate Stored Procedure - AdventureWorks.HumanResources.uspGetAllEmployees

This procedure does not have any input or in/out parameters.

Click Execute to execute the procedure and retrieve recordset structures.

☑ Execute the procedure in a transaction that is rolled back immediately

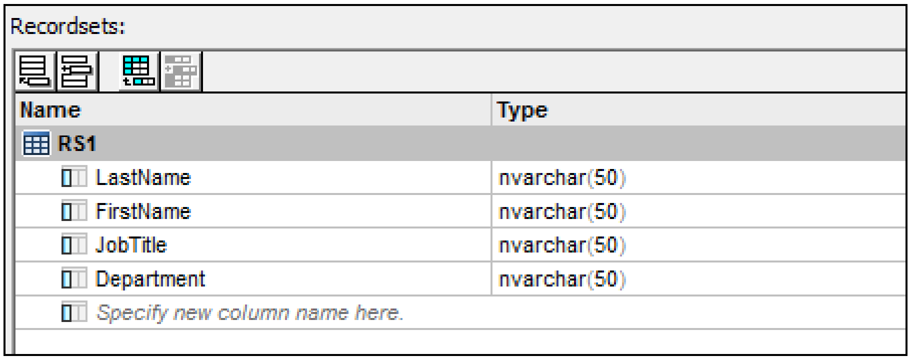
Warning: Executing a stored procedure can have side effects that cannot be rolled back depending on the procedure's implementation.

Execution result status:
Not yet executed.

Output parameters:

Name	Type	Value
@uspGetAllEmployees	int	*not yet retrieved*

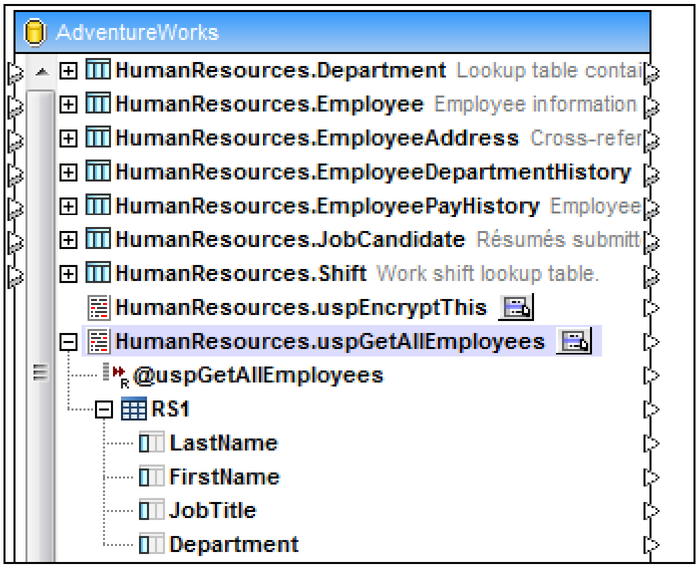
3. Click the "Execute" button, then click OK.
   The recordset fields are now visible in the Recordsets section of the dialog box.



Recordsets:

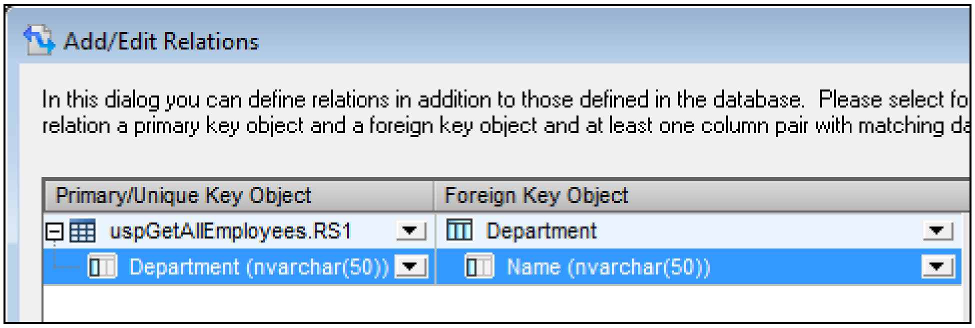
Name	Type
RS1	
LastName	nvarchar(50)
FirstName	nvarchar(50)
JobTitle	nvarchar(50)
Department	nvarchar(50)
Specify new column name here.	

4. Click the OK button again to complete the recordset definition.
   The columns, LastName etc., are shown as nodes below the recordset node **RS1**. (Click the "+" button to expand the recordset if not visible).


AdventureWorks
HumanResources.Department Lookup table contai
HumanResources.Employee Employee information
HumanResources.EmployeeAddress Cross-refer
HumanResources.EmployeeDepartmentHistory
HumanResources.EmployeePayHistory Employee
HumanResources.JobCandidate Résumés submitt
HumanResources.Shift Work shift lookup table.
HumanResources.uspEncryptThis
HumanResources.uspGetAllEmployees
@uspGetAllEmployees
RS1
LastName
FirstName
JobTitle
Department

Defining a Local relation to a different table:

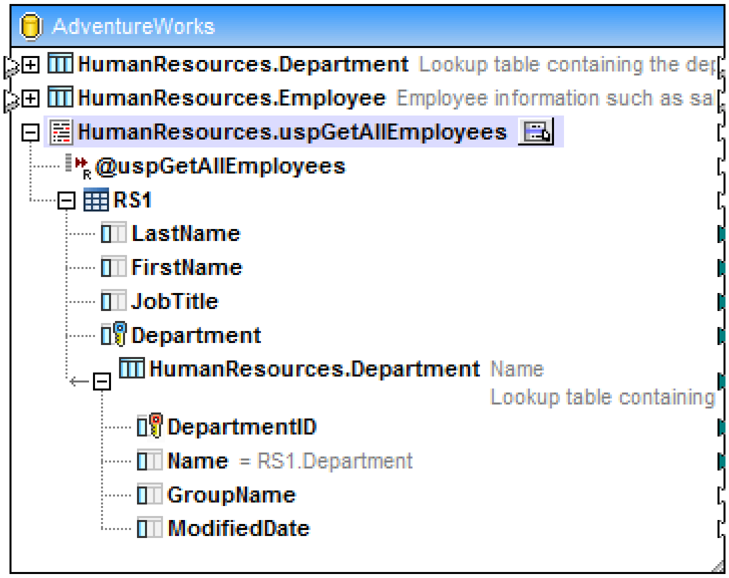
1. Right click the Component header, and select Add/Remove/Edit Database Objects.
2. Click the Add/Edit Relations button to open the Add/Edit Relations dialog box, then click the "Add Relation" button.
3. Define the Primary/Unique Key Object as the stored procedure **uspGetAllEmployees.RS1** and the column as the **@Department** parameter.
4. Define the Foreign Key Object as **Department** and the column as **Name**.


Add/Edit Relations

In this dialog you can define relations in addition to those defined in the database. Please select fo relation a primary key object and a foreign key object and at least one column pair with matching da

Primary/Unique Key Object	Foreign Key Object
uspGetAllEmployees.RS1	Department
Department (nvarchar(50))	Name (nvarchar(50))

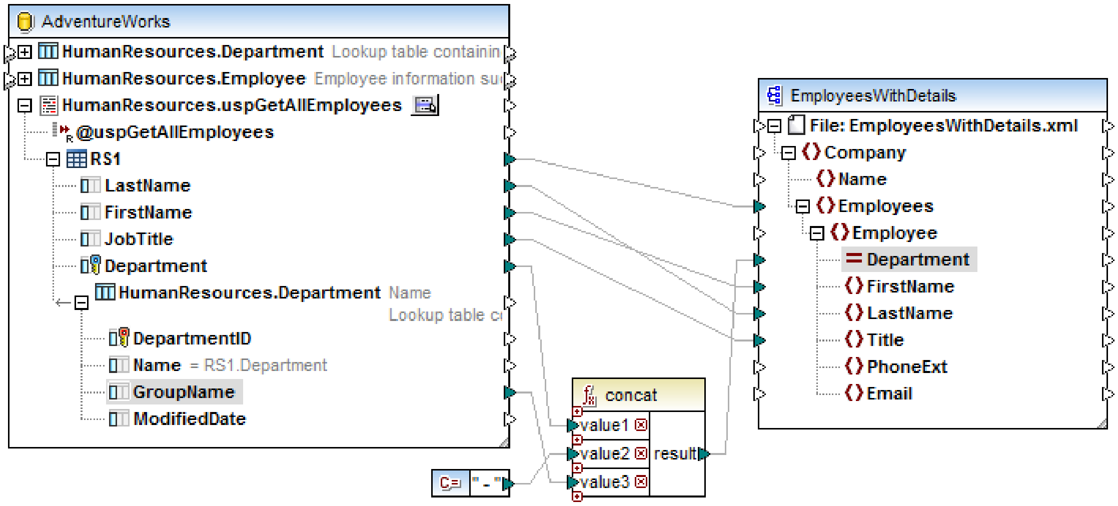
5. Click the OK button in the various dialog boxes.


AdventureWorks
HumanResources.Department Lookup table containing the dep
HumanResources.Employee Employee information such as sa
HumanResources.uspGetAllEmployees
@uspGetAllEmployees
RS1
LastName
FirstName
JobTitle
Department
HumanResources.Department Name
Lookup table containing
DepartmentID
Name = RS1.Department
GroupName
ModifiedDate

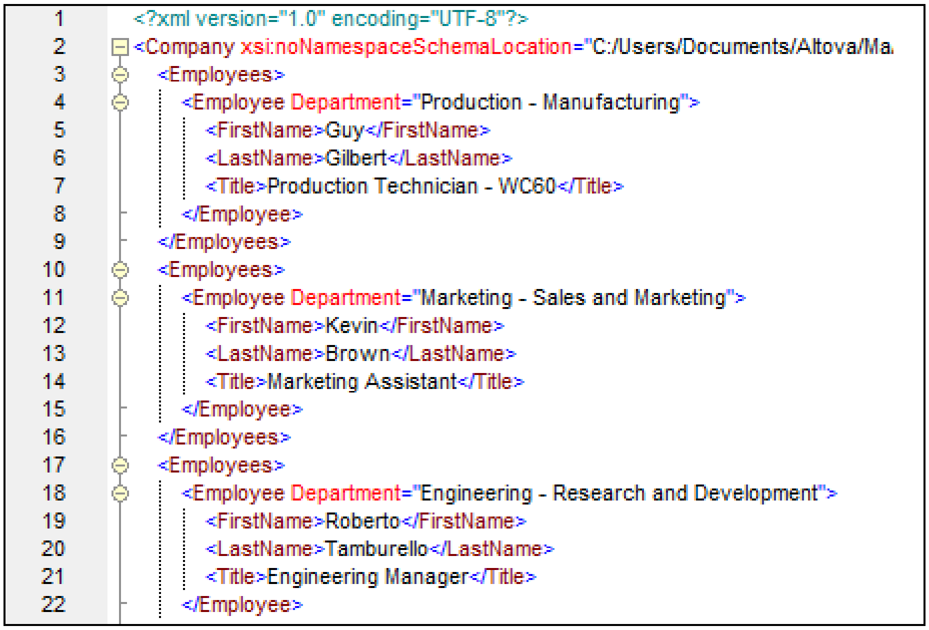
The Department table is now displayed as a child of the stored procedure.

Completing the mapping

1. Insert the target schema to which you want to map the source database data, and add the connections as shown below.



2. Click the Output button to see the result.



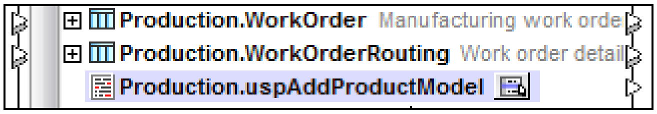
```
<?xml version="1.0" encoding="UTF-8"?>
<Company xsi:noNamespaceSchemaLocation="C:/Users/Documents/Altova/Ma
  <Employees>
    <Employee Department="Production - Manufacturing">
      <FirstName>Guy</FirstName>
      <LastName>Gilbert</LastName>
      <Title>Production Technician - WC60</Title>
    </Employee>
  </Employees>
  <Employees>
    <Employee Department="Marketing - Sales and Marketing">
      <FirstName>Kevin</FirstName>
      <LastName>Brown</LastName>
      <Title>Marketing Assistant</Title>
    </Employee>
  </Employees>
  <Employees>
    <Employee Department="Engineering - Research and Development">
      <FirstName>Roberto</FirstName>
      <LastName>Tamburello</LastName>
      <Title>Engineering Manager</Title>
    </Employee>
```

7.2.9.5 Stored procedures in Target components

Choose this option when the stored procedure makes changes to the database, e.g. add/update/ delete etc., and you are not interested in any stored procedure output.

To use stored procedures in a target component:
This option adds the child nodes of the **input** parameters (as well as in/out parameters) under the stored procedure item in the target database component.

E.g.: You want to add a new product model to the database, using the uspAddProductModel stored procedure of the AdventureWorks database.


⊞ Production.WorkOrder Manufacturing work orde
⊞ Production.WorkOrderRouting Work order detail
Production.uspAddProductModel

Stored procedure:

```
PROCEDURE Production.uspAddProductModel
@ModelName nvarchar(50),
@Inst xml

as
INSERT INTO [AdventureWorks].[Production].[ProductModel]
           ([Name]
           --,[CatalogDescription]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           --,<CatalogDescription, ProductDescriptionSchemaCollection,>
           ,@Inst
```

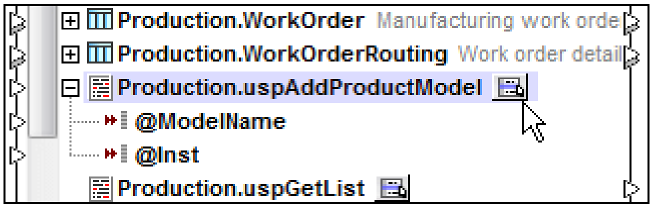
```
        ,NEWID()
        ,GETDATE());
```

At runtime, MapForce executes the stored procedure using all the mapped **input** parameters while ignoring the stored procedure data output.

To create the input parameter items in a target component:

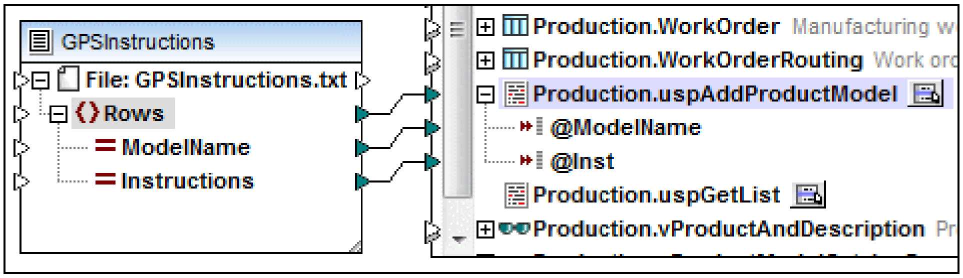
Having inserted the AdventureWorks database component and selected the Production tables and included stored procedures:

1. Click the "stored procedure" button and select the option "Show nodes as Target".


Production.WorkOrder Manufacturing work orde
Production.WorkOrderRouting Work order detail
Production.uspAddProductModel
@ModelName
@Inst
Production.uspGetList

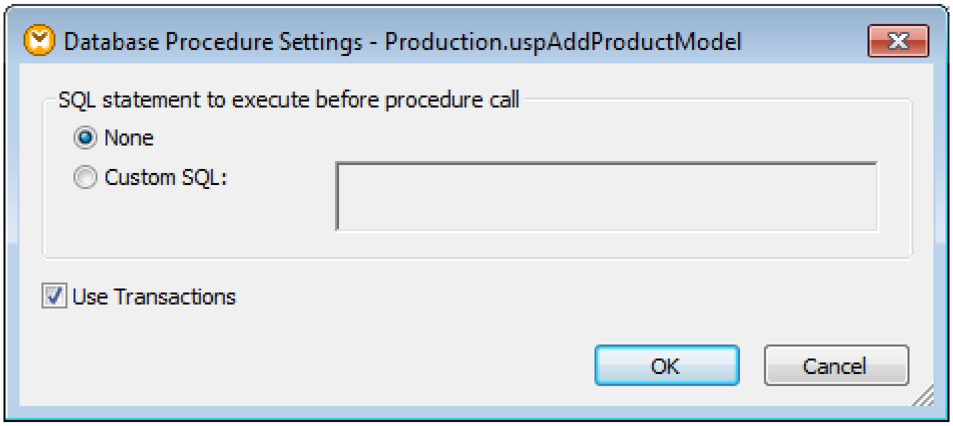
This inserts the @ModelName and @Inst input parameters below the stored procedure name. Only the **input** icons of the input parameters are available in the target component.

2. Insert a source component, e.g. text file, XML file, etc., and map the items that are to supply the input parameter data, to the input icons of the stored procedure.


GPSInstructions
File: GPSInstructions.txt
Rows
ModelName
Instructions
Production.WorkOrder Manufacturing w
Production.WorkOrderRouting Work ord
Production.uspAddProductModel
@ModelName
@Inst
Production.uspGetList
Production.vProductAndDescription Pr

To define transactions for a stored procedure:

1. Click the "stored procedure" button and select the option "Procedure settings". This opens the Database Procedure Settings dialog box.



Database Procedure Settings - Production.uspAddProductModel

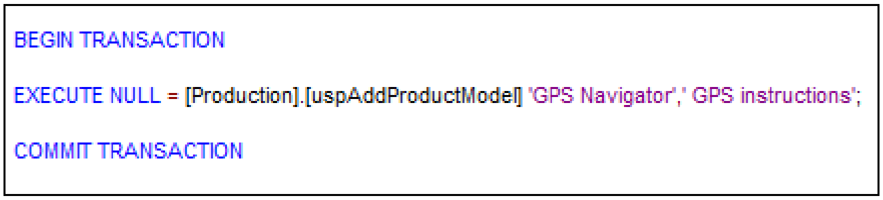
SQL statement to execute before procedure call

● None

○ Custom SQL:

☑ Use Transactions

2. Click the "Use Transactions" check box and click OK to confirm.
   The transaction setting makes sure that the procedure commands can be rolled back if an error occurs during execution.
3. Click the Output button to see the commands that will be sent to the database.

   ```
   BEGIN TRANSACTION

   EXECUTE NULL = [Production].[uspAddProductModel] 'GPS Navigator',' GPS instructions';

   COMMIT TRANSACTION
   ```

   This dialog box also allows you to define SQL statements to be executed before the stored procedure is called.

Notes:
The "Add Duplicate input..." context menu options are disabled for the stored procedure **parameters**, as each parameter is an atomic value (and could also be "nullable").

The "Add duplicate input..." context menu options are however available for a stored **procedure** item. This would call the stored procedure for each duplicated item/node.

Using stored procedures to generate primary keys

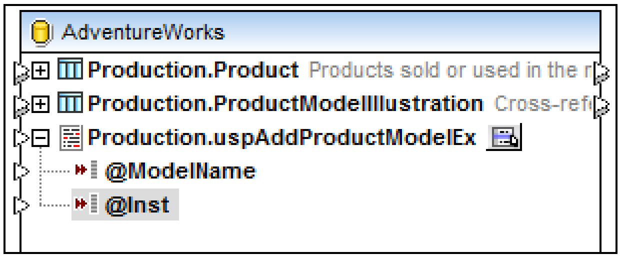
Choose this option when the stored procedure makes changes to a database table, and you also want to use the procedure output parameter to generate a primary key in a different table.

The uspAddProductModelEx procedure is a variation of the uspAddProductModel stored procedure in the AdventureWorks database.

```
procedure Production.uspAddProductModelEx
@ModelName nvarchar(50),
@Inst xml,
@ProductModelID int OUTPUT
as begin
INSERT INTO [AdventureWorks].[Production].[ProductModel]
           ([Name]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           ,@Inst
           ,NEWID()
           ,GETDATE());
   SELECT @ProductModelID = SCOPE_IDENTITY()
end;
```

Having inserted the AdventureWorks database component, selected the Production tables and included stored procedures:

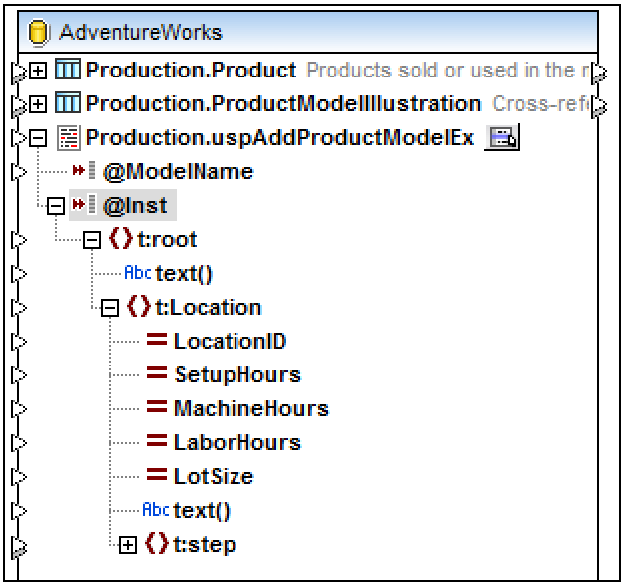
1. Click the "stored procedure" button and select the option "Show nodes as Target".


AdventureWorks
Production.Product Products sold or used in the r
Production.ProductModelIllustration Cross-ref
Production.uspAddProductModelEx
@ModelName
@Inst

This inserts the ModelName and Inst parameters below the procedure name. Only the input parameters of the stored procedure are visible in the component.

As the Inst parameter is of type XML, we need to assign it a relevant XML Schema to supply the XML data.

2. Right click the Inst parameter and select "Assign XML Schema to field...".
3. Select the provided "Production.ManuInstructionsSchemaCollection in the "Database" combo box, and click OK.

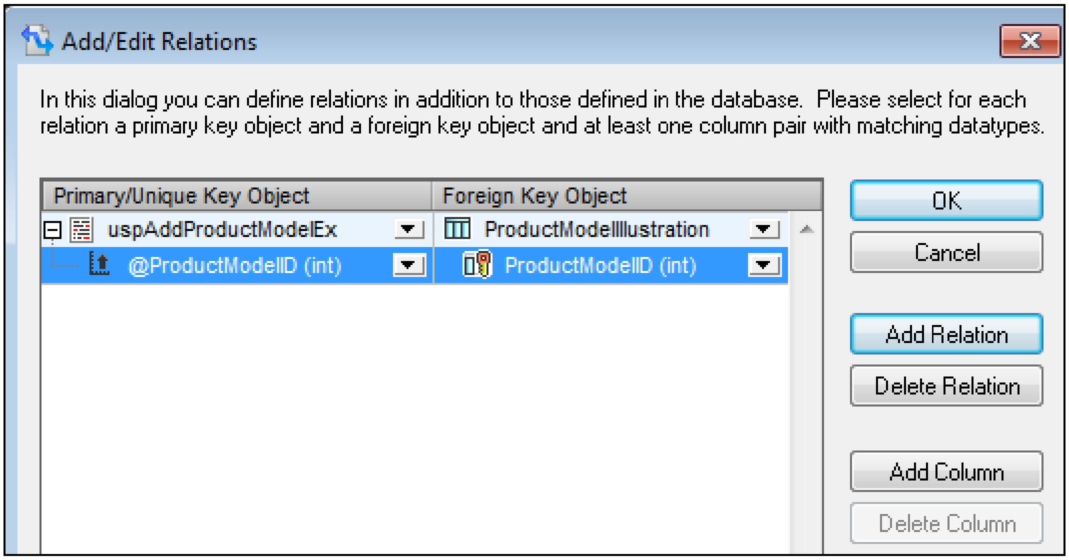

AdventureWorks
Production.Product Products sold or used in the r
Production.ProductModelIllustration Cross-ref
Production.uspAddProductModelEx
@ModelName
@Inst
t:root
text()
t:Location
LocationID
SetupHours
MachineHours
LaborHours
LotSize
text()
t:step

This adds the XML Schema elements and attributes to the component. The ModelName parameter and all the Inst parameters are now available in the component.

We now want to define a Local relation to a table that has no direct connection to the table referenced by the stored procedure parameters (production.product).

Defining a Local relation to a table in which you want to generate a primary key:

1. Right click the Component header, and select Add/Remove/Edit Database Objects.
2. Click the Add/Edit Relations button to open the Add/Edit Relations dialog box, then click the "Add Relation" button.
3. Define the Primary/Unique Key Object as the stored procedure **uspAddProductModelEx** and the column as the **@ProductModelID** parameter.
4. Define the Foreign Key Object as **ProductModelIllustration** and the column as **ProductModelID**.

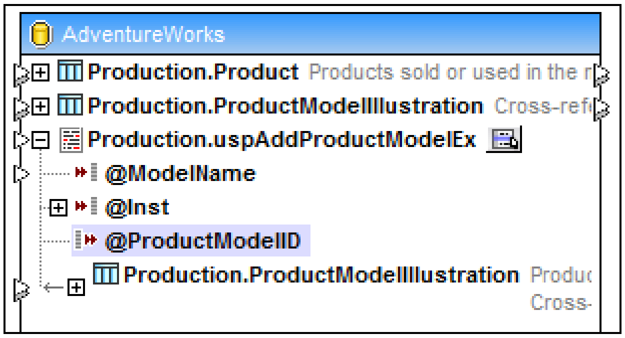


Add/Edit Relations

In this dialog you can define relations in addition to those defined in the database. Please select for each relation a primary key object and a foreign key object and at least one column pair with matching datatypes.

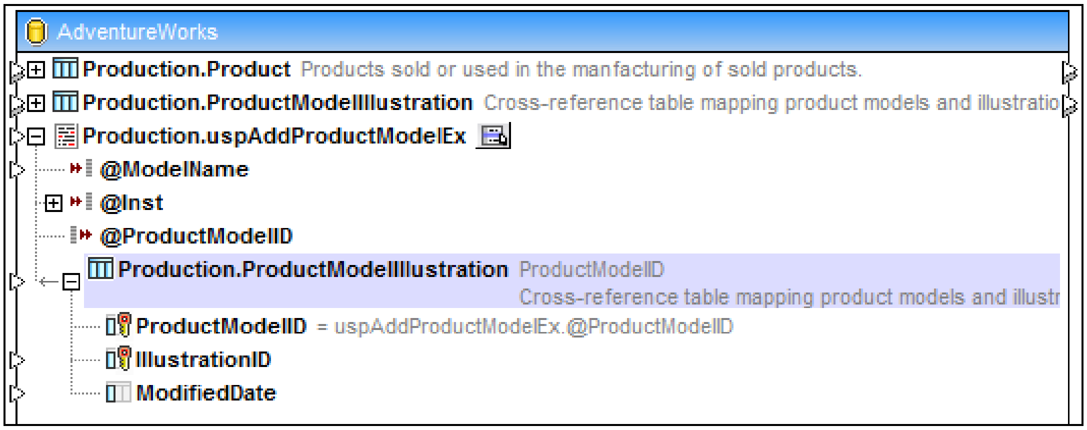
Primary/Unique Key Object	Foreign Key Object
uspAddProductModelEx	ProductModelIllustration
@ProductModelID (int)	ProductModelID (int)

5. Click the OK button in the various dialog boxes.



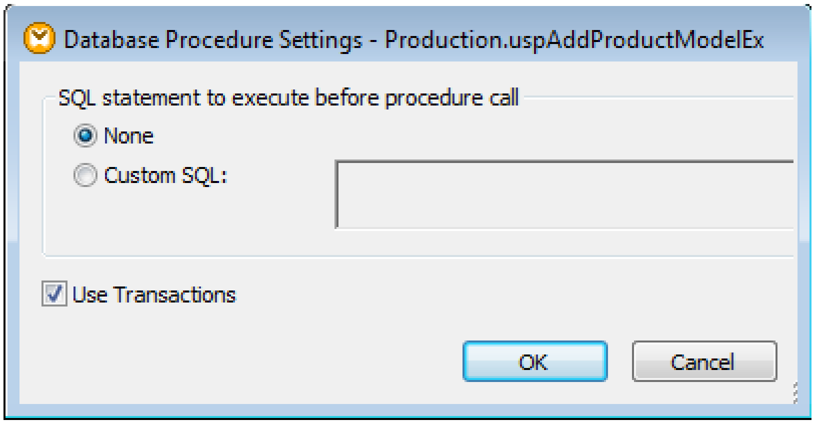
AdventureWorks
Production.Product Products sold or used in the r
Production.ProductModelIllustration Cross-refe
Production.uspAddProductModelEx
@ModelName
@Inst
@ProductModelID
Production.ProductModelIllustration Produc
Cross-

- The stored procedure output parameter (ProductModelID) has been added to the stored procedure as an indicator that it will be used in the local relation, but does not have any input or output icons.
- The table ProductModelIllustration has also been added as a child item to the stored procedure.
- Expanding the table shows the keys and columns of the table. Note that ProductModelID key shows the stored procedure and parameter name it is related to.
- Local relations that use the (output) recordset of the stored procedure, cannot be used here.
- Clicking the stored procedure button and selecting "Procedure Settings" allows you to define an SQL Statement to be run before the procedure is called, as well as activate transaction settings.


AdventureWorks
Production.Product Products sold or used in the manfacturing of sold products.
Production.ProductModelIllustration Cross-reference table mapping product models and illustratio
Production.uspAddProductModelEx
@ModelName
@Inst
@ProductModelID
Production.ProductModelIllustration ProductModelID
Cross-reference table mapping product models and illustr
ProductModelID = uspAddProductModelEx.@ProductModelID
IllustrationID
ModifiedDate

Defining a transaction for a stored procedure:

1. Click the stored procedure icon and select "Procedure Settings".
2. Click the Use Transactions check box, then click OK to confirm.



Database Procedure Settings - Production.uspAddProductModelEx

SQL statement to execute before procedure call

● None

○ Custom SQL:

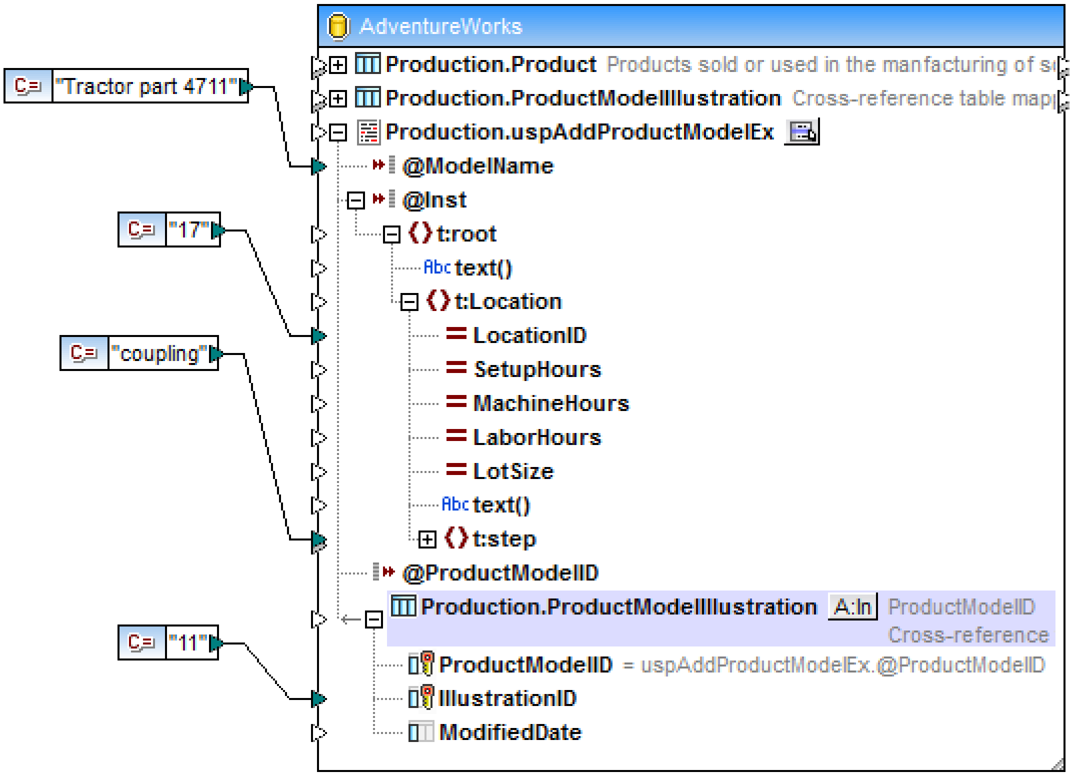
☑ Use Transactions

Defining the transaction for the stored procedure ensures that both retrieving the key and inserting the record both occur during the same transaction.

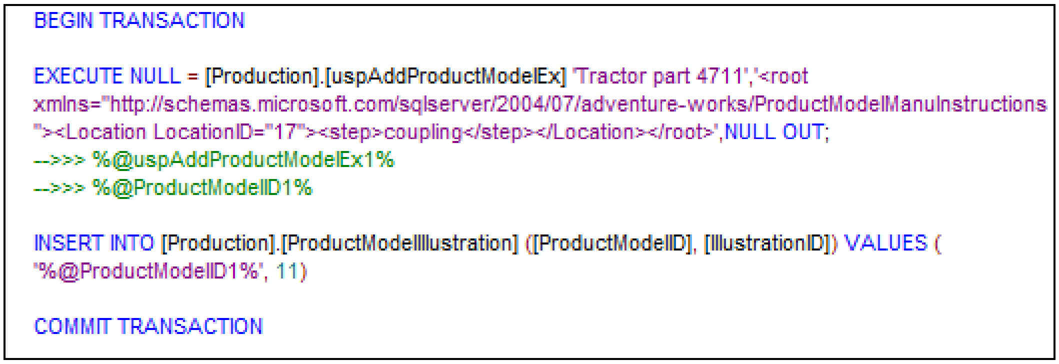
Completing the mapping:

The screenshot below shows only a subset of the data you would normally map.

1. Map the data source items to the target database; in this case constants.


"Tractor part 4711"
"17"
"coupling"
"11"
AdventureWorks
Production.Product Products sold or used in the manfacturing of s
Production.ProductModelIllustration Cross-reference table map
Production.uspAddProductModelEx
@ModelName
@Inst
t:root
text()
t:Location
LocationID
SetupHours
MachineHours
LaborHours
LotSize
text()
t:step
@ProductModelID
Production.ProductModelIllustration A:In ProductModelID Cross-reference
ProductModelID = uspAddProductModelEx.@ProductModelID
IllustrationID
ModifiedDate

2. Click the Output button to see the pseudo SQL that will be sent to the database.



```
BEGIN TRANSACTION

EXECUTE NULL = [Production].[uspAddProductModelEx] 'Tractor part 4711','<root
xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/ProductModelManuInstructions
"><Location LocationID="17"><step>coupling</step></Location></root>',NULL OUT;
-->>> %@uspAddProductModelEx1%
-->>> %@ProductModelID1%

INSERT INTO [Production].[ProductModelIllustration] ([ProductModelID], [IllustrationID]) VALUES (
'%@ProductModelID1%', 11)

COMMIT TRANSACTION
```

MapForce automatically calls the stored procedure for each record before the Insert action.

7.3 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

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.

Mapping data to or from text files is supported in any one of the following languages: Java, C#, C++, or BUILT-IN.

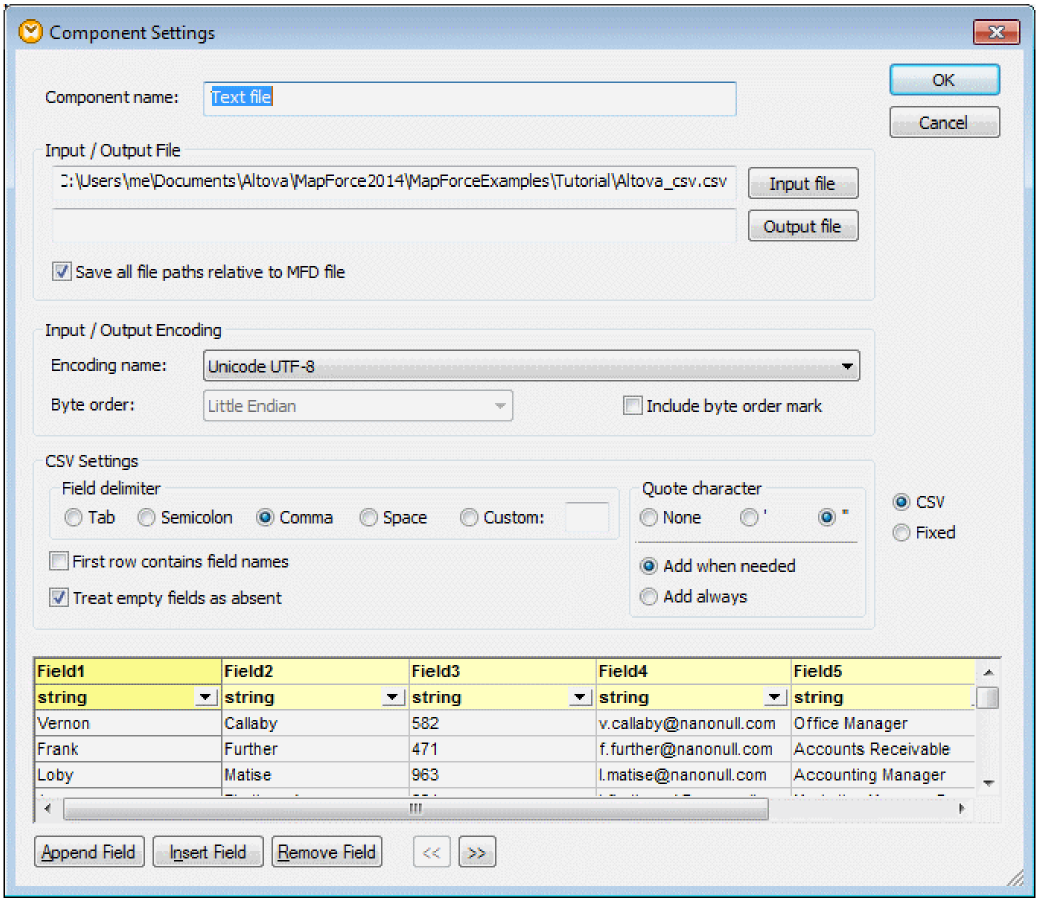
There are two ways that mapped flat file data can be generated:

- 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.

7.3.1 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\MapForce2018\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. 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.



Component Settings

Component name: Text file

Input / Output File: C:\Users\me\Documents\Altova\MapForce2014\MapForceExamples\Tutorial\Altova_csv.csv

☑ Save all file paths relative to MFD file

Input / Output Encoding

Encoding name: Unicode UTF-8

Byte order: Little Endian

□ Include byte order mark

CSV Settings

Field delimiter: ○ Tab ○ Semicolon ● Comma ○ Space ○ Custom:

Quote character: ○ None ○ ' ● "

● Add when needed ○ Add always

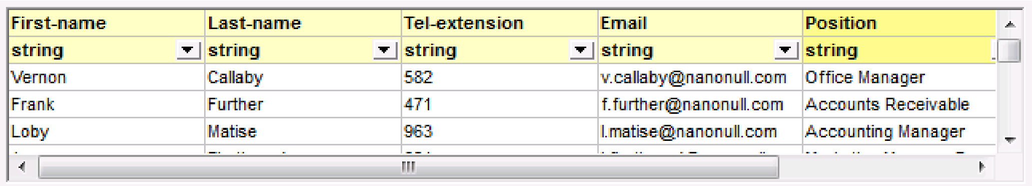
● CSV ○ Fixed

□ First row contains field names

☑ Treat empty fields as absent

Field1	Field2	Field3	Field4	Field5
string	string	string	string	string
Vernon	Callaby	582	v.callaby@nanonull.com	Office Manager
Frank	Further	471	f.further@nanonull.com	Accounts Receivable
Loby	Matise	963	l.matise@nanonull.com	Accounting Manager

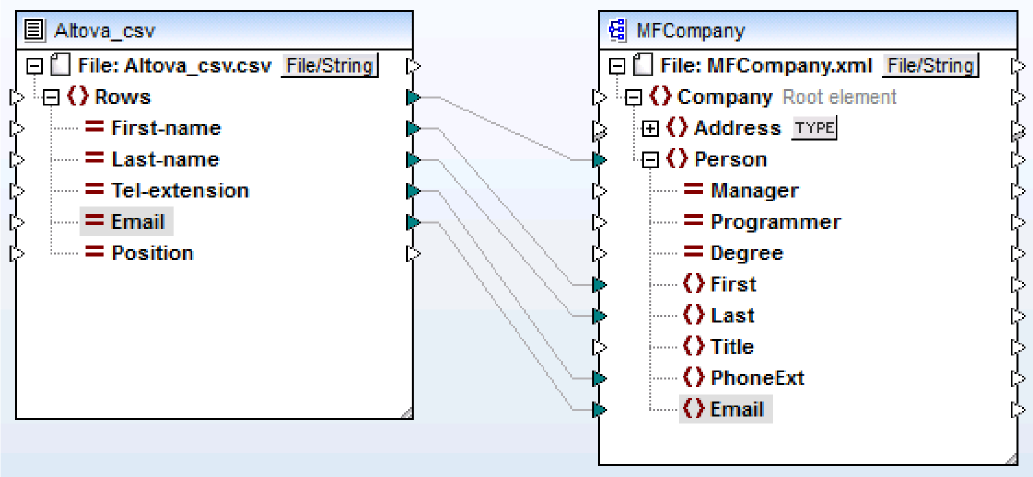
4. Click inside the **Field1** header and change the text to First-name. Do the same for all the other fields, as follows: Field 2 => Last-name, Field 3 =>Tel-extension, Field 4 => Email, Field 5 => Position. TIP: Press the **Tab** key to cycle through all the fields: header1, header2 etc.



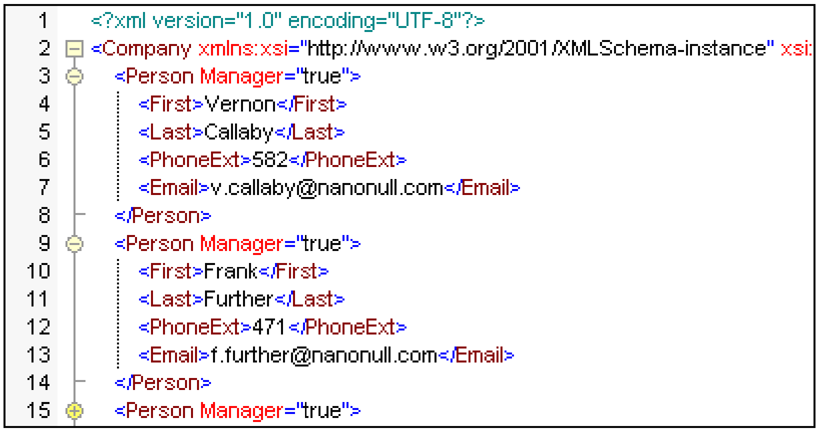
First-name	Last-name	Tel-extension	Email	Position
string	string	string	string	string
Vernon	Callaby	582	v.callaby@nanonull.com	Office Manager
Frank	Further	471	f.further@nanonull.com	Accounts Receivable
Loby	Matise	963	l.matise@nanonull.com	Accounting Manager

5. Click OK.
6. When prompted to change the component name, click "Change component name". The CSV component is now visible in the mapping.
7. Add **MFCompany.xsd** as the target XML component of the mapping (on the **Insert** menu, click **XML/Schema file**).
8. Click **Skip** when prompted to supply a sample XML file, and select `Company` as the root element.
9. 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.



10. Click the Output tab to see the result.

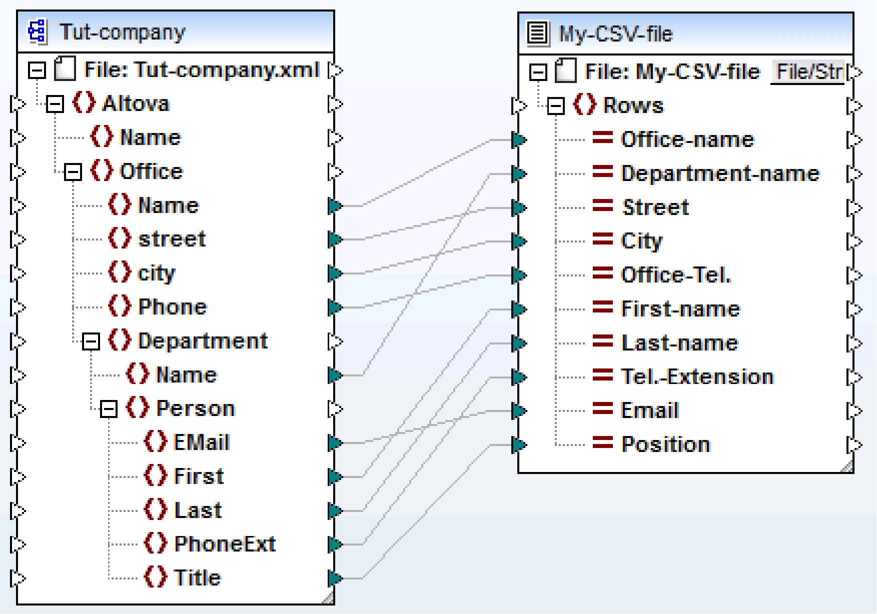


```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
  <Person Manager="true">
    <First>Vernon</First>
    <Last>Callaby</Last>
    <PhoneExt>582</PhoneExt>
    <Email>v.callaby@nanonull.com</Email>
  </Person>
  <Person Manager="true">
    <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.

7.3.2 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\MapForce2018\MapForceExamples\Tutorial\Tut-xml2csv.mfd**.

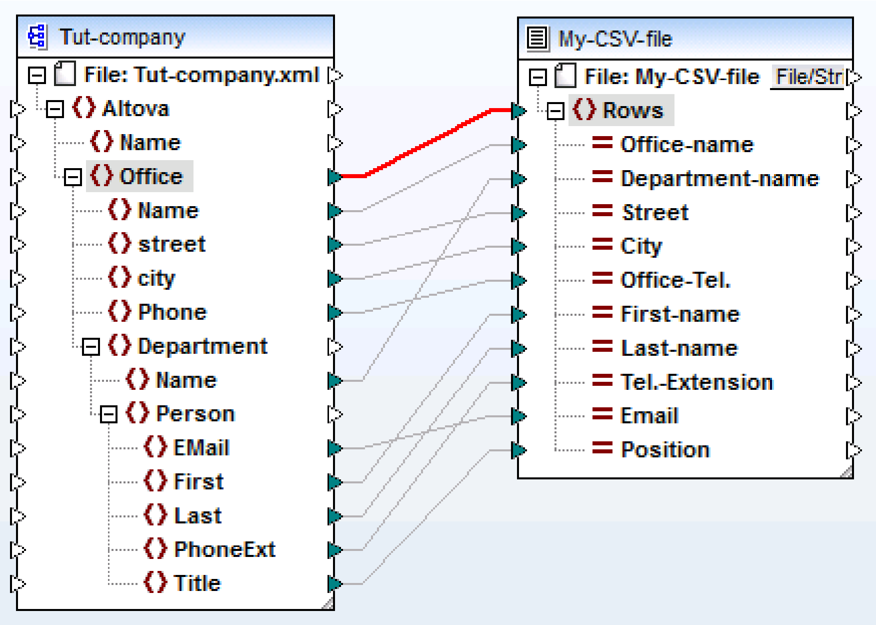


Tut-xml2csv.mfd

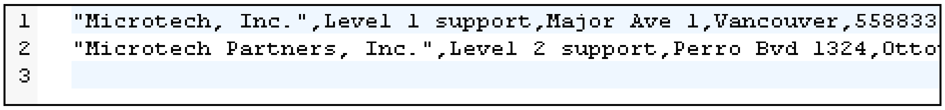
This mapping has been intentionally created as incomplete. If you attempt to validate the example file using the menu command **File | Validate Mapping**, you will notice that validation warnings occur. Also, if you preview the mapping output, a single row is produced, which may or may not be your intended 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.

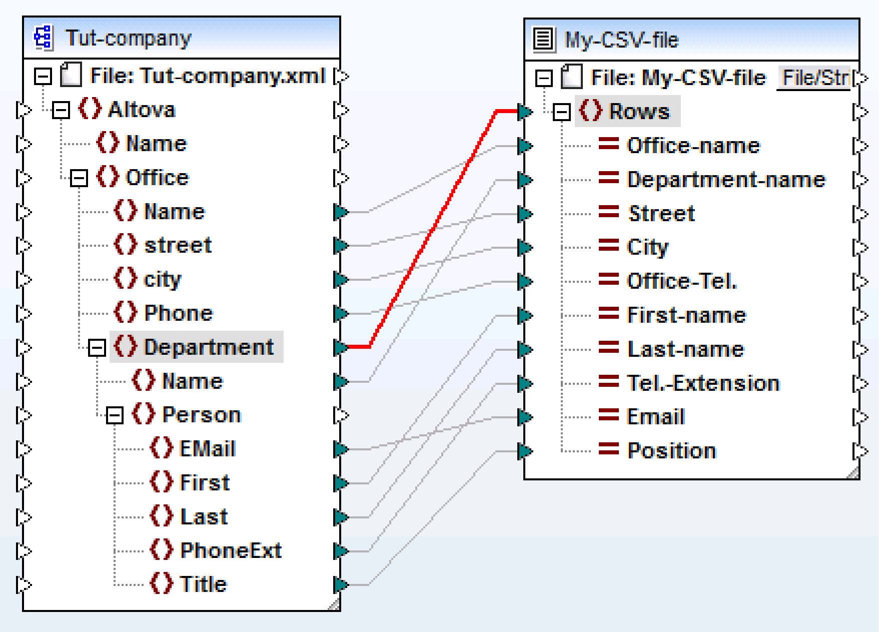


The Rows item in the CSV component acts as an iterator for the sequence of items connected to it. Therefore, if you connect the Office item, the output creates a row for each office found in the source XML.

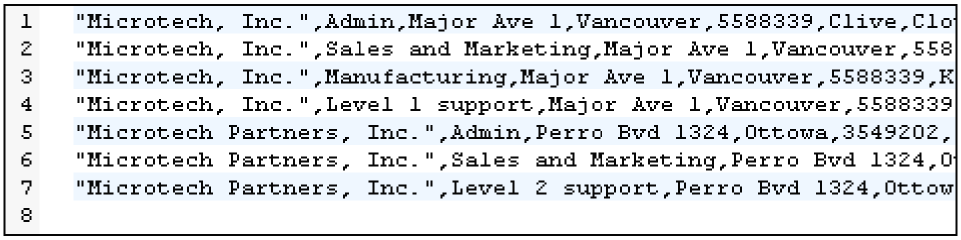


```
"Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,558833
"Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Otto

```

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
```

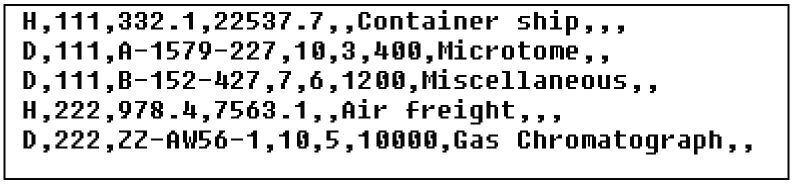
Finally, mapping `Person` to the `Rows` item results in all the Persons being output. In this case, MapForce will iterate through the records as follows: each Person within each Department, within each Office.

7.3.3 Example: Creating Hierarchies from CSV and Fixed-Length Text Files

This example is available at the following path: **<Documents>\Altova\MapForce2018\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.

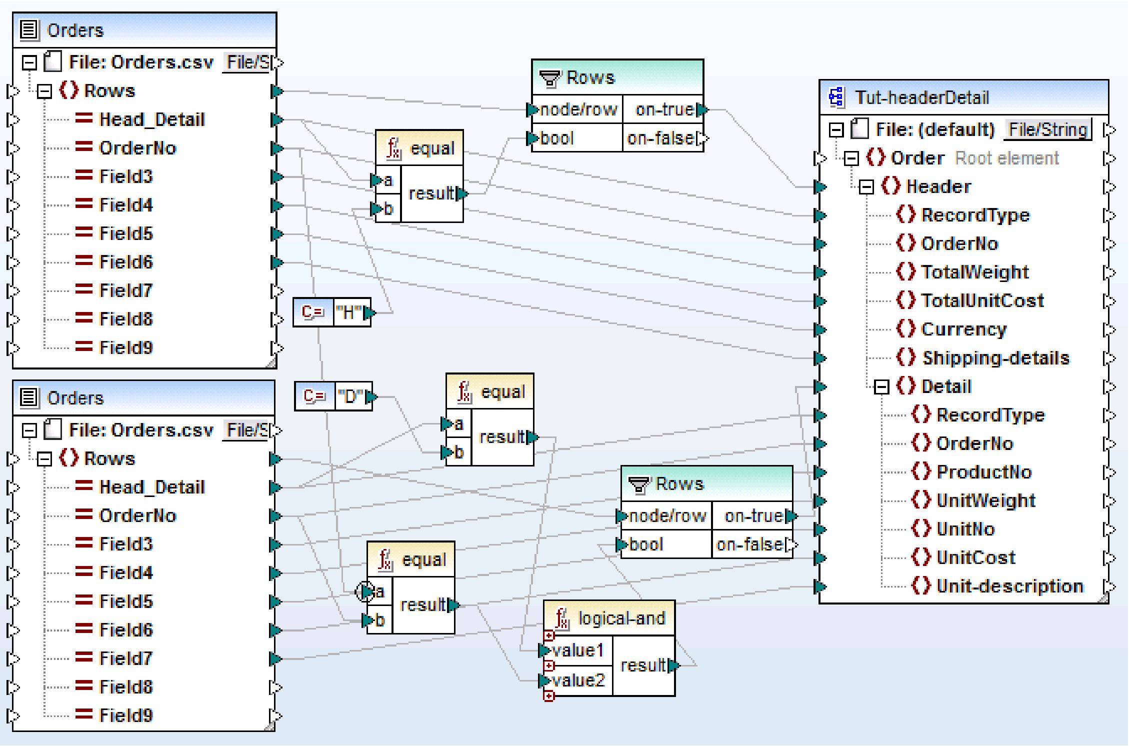
The contents of the Orders.csv file are shown below.



```
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.

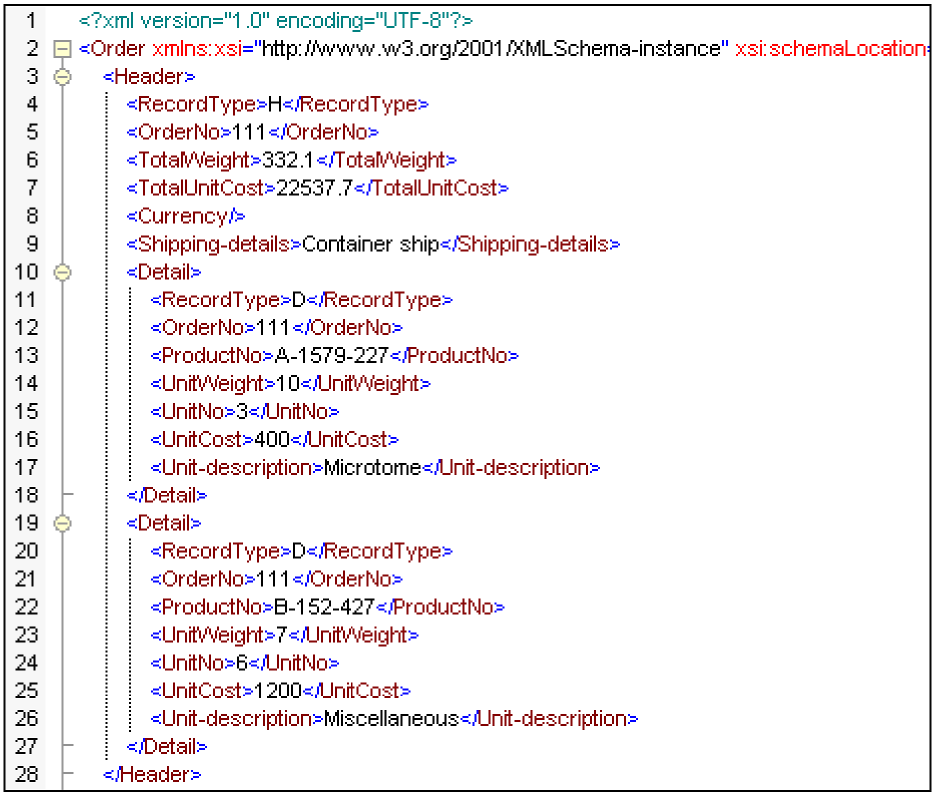
The first Orders.csv file supplies the **Header** records (and all mapped fields) to the Header item in the schema target file. The filter component is used to filter out all records other than those starting with H. The **Rows** item supplies these filtered records to the Header item in the schema file.

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 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.

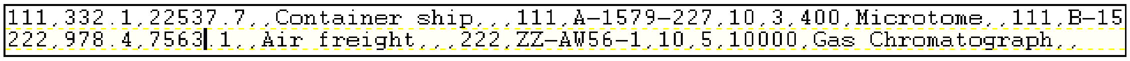
Clicking the Output tab produces the XML file displayed below. Each Header record contains its data, and all associated Detail records that have the same Order No.



```
<?xml 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\MapForce2018\MapForceExamples\Tutorial\** folder as **Head-detail-inline.mfd**. The difference is that:

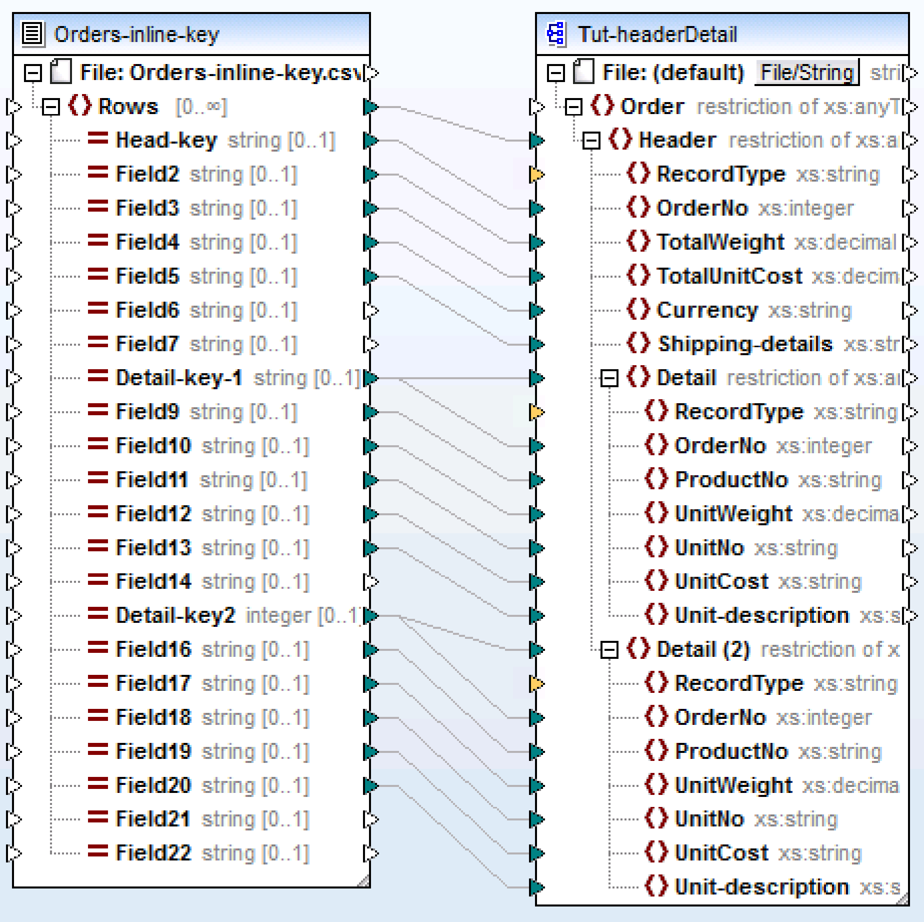
- No record designator (H, or D) is available
- A common key field, the first field of the CSV file, still exists for both header and detail records (Head-key, Detail-key...). The field is mapped to OrderNo in the schema target
- Header and all respective Detail fields are all on the same line.



```
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.

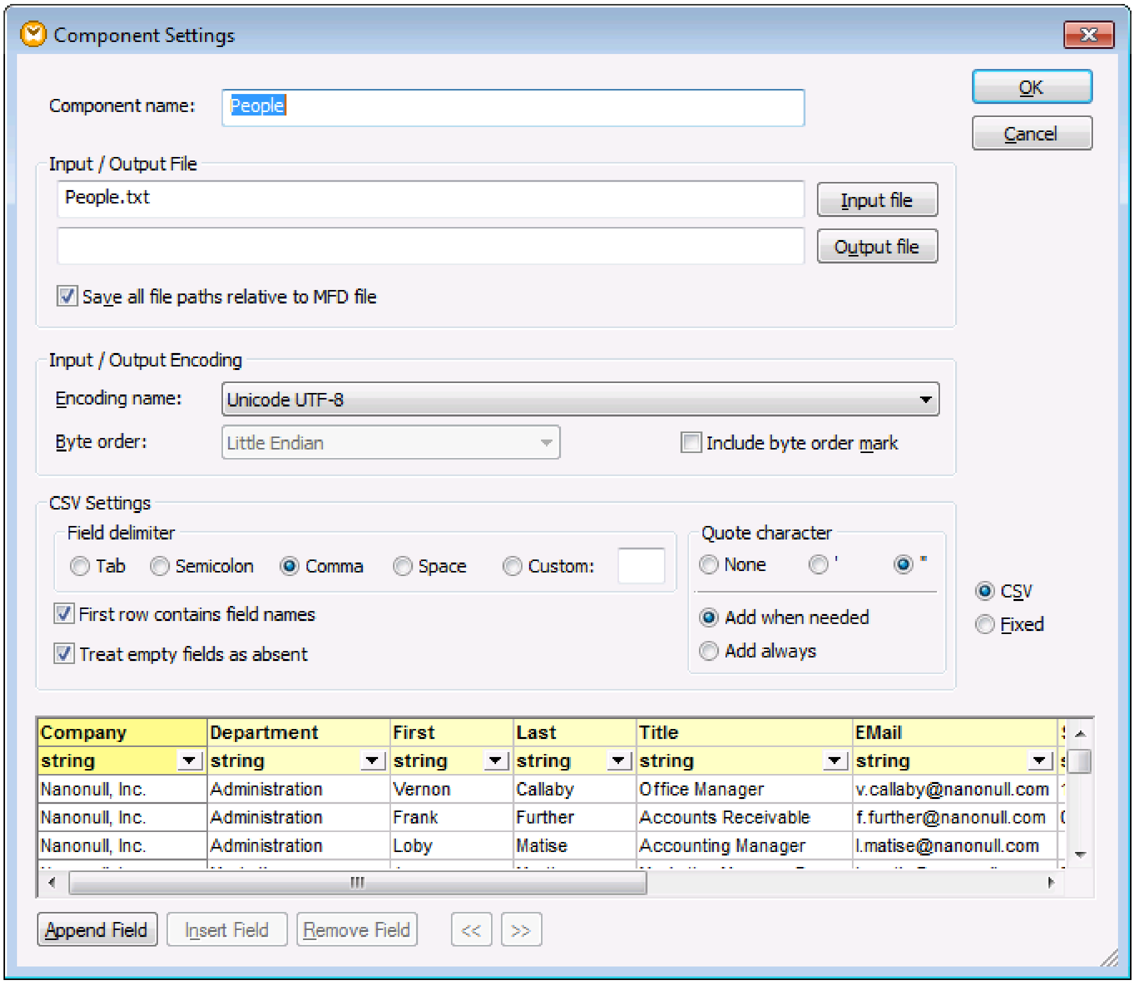


Head-detail-inline.mfd

7.3.4 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**.



Component name: People

Input / Output File

People.txt

☑ Save all file paths relative to MFD file

Input / Output Encoding

Encoding name: Unicode UTF-8

Byte order: Little Endian

☐ Include byte order mark

CSV Settings

Field delimiter: ○ Tab ○ Semicolon ● Comma ○ Space ○ Custom:

Quote character: ○ None ○ ' ● "

● Add when needed
○ Add always

☑ First row contains field names

☑ Treat empty fields as absent

● CSV
○ Fixed

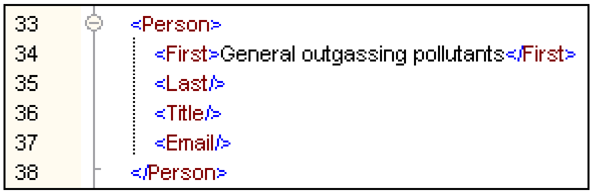
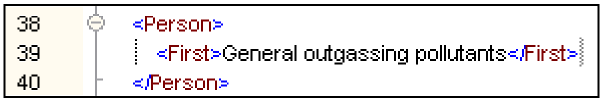
Company	Department	First	Last	Title	EMail
string	string	string	string	string	string
Nanonull, Inc.	Administration	Vernon	Callaby	Office Manager	v.callaby@nanonull.com
Nanonull, Inc.	Administration	Frank	Further	Accounts Receivable	f.further@nanonull.com
Nanonull, Inc.	Administration	Loby	Matise	Accounting Manager	l.matise@nanonull.com

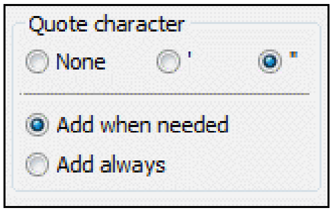
Text Component Settings dialog box (in CSV mode)

The available settings are as follows.

Component name	The component name is automatically generated when you create 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.
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.
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

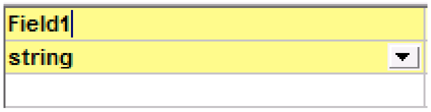
	will not produce a corresponding empty item (element or attribute) in the target file.
	For example, the CSV record "`General outgassing pollutants,,,,`" consists of four fields, the last three of which are empty.
	Assuming that the output is an XML file, when this option is disabled, the empty fields 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>`

Quote character	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.  Quote character ○ None ○ ' ● " ● Add when needed ○ Add always For output files, you can specify additional settings: **Add when needed** Adds the selected quote character to fields where the text contains the field line breaks. **Add always** Adds the selected quote character to the generated CSV file.
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.

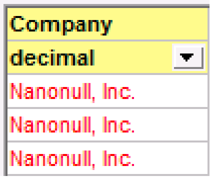
Append field	Creates a new field after the last CSV
Insert field	Creates a new field immediately befor currently selected CSV record.
Remove field	Deletes the currently selected field.
<<	Moves the currently selected field one the left.
>>	Moves the currently selected field one the right.

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.



Field1
string

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.



Company
decimal
Nanonull, Inc.
Nanonull, Inc.
Nanonull, Inc.

The field types are based on the default XML schema data types. For example, the Date type is in the form **YYYY-MM-DD**.

7.3.5 Example: Mapping Fixed-Length Text Files to Databases

This example illustrates a data mapping operation between a fixed-length text file (FLF) text file and a Microsoft Access database. The files used in the example are available in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder. Both the source text file and the target database store a list of employees. In the source file, the records are implicitly delimited by their size, as follows:

Field position and name	Size (in characters)
Field 1 (First name)	8

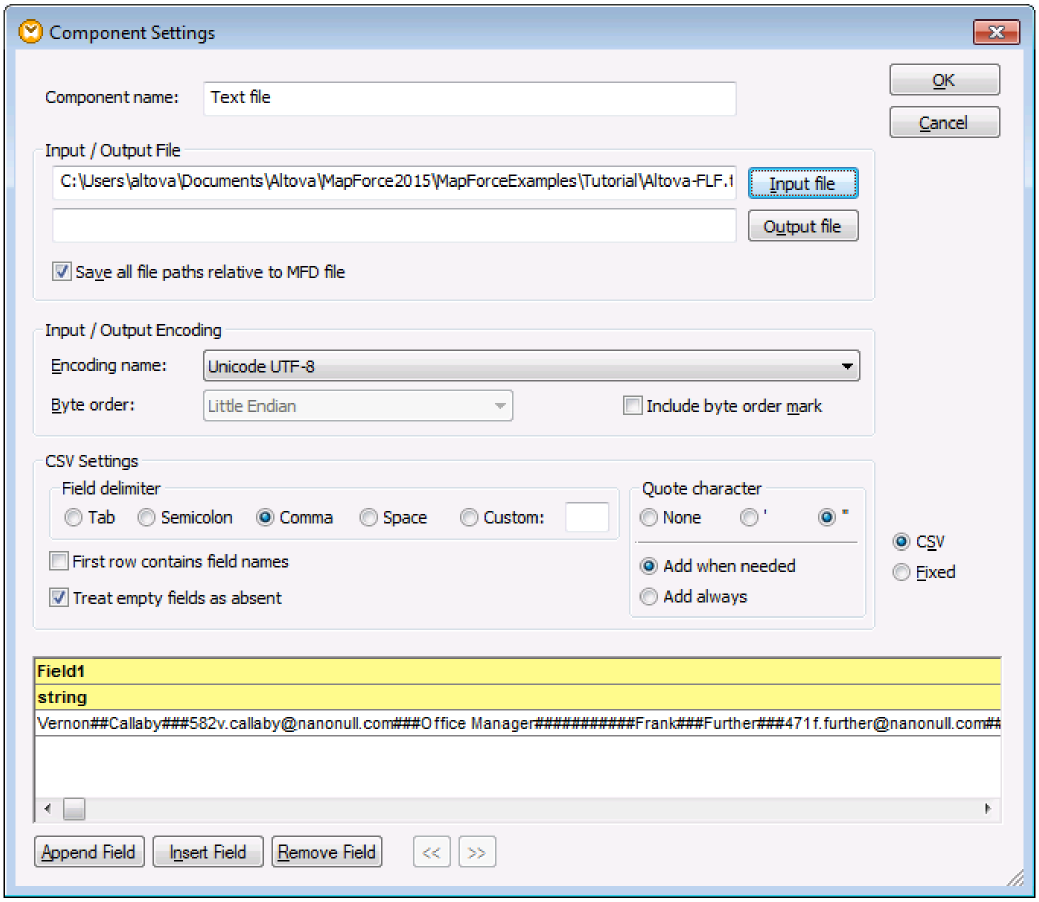
Field position and name	Size (in characters)
Field 2 (Last name)	10
Field 3 (Phone extension)	3
Field 4 (Email)	25
Field 5 (Position)	25

The goals of the mapping is to update the phone extension of each employee in the database to the one existing in the source file, while adding the prefix "100" to each extension. To achieve the goal, take the following steps:

Step 1: Insert and configure the text component
Step 2: Insert the database component
Step 3: Design the mapping
Step 4: Run the mapping

Step 1: Insert and configure the text component

1. Select the menu option **Insert | Text file**, or click the insert Text file icon .
2. Click the **Input file** button and select the file **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\Altova-FLF.txt** file. You will notice that the file is made up of a single string, and contains fill characters of type #.



Component Settings

Component name: Text file

Input / Output File

C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Tutorial\Altova-FLF.t

☑ Save all file paths relative to MFD file

Input / Output Encoding

Encoding name: Unicode UTF-8

Byte order: Little Endian

□ Include byte order mark

CSV Settings

Field delimiter: ○ Tab ○ Semicolon ● Comma ○ Space ○ Custom:

Quote character: ○ None ○ ' ● "

● Add when needed

○ Add always

□ First row contains field names

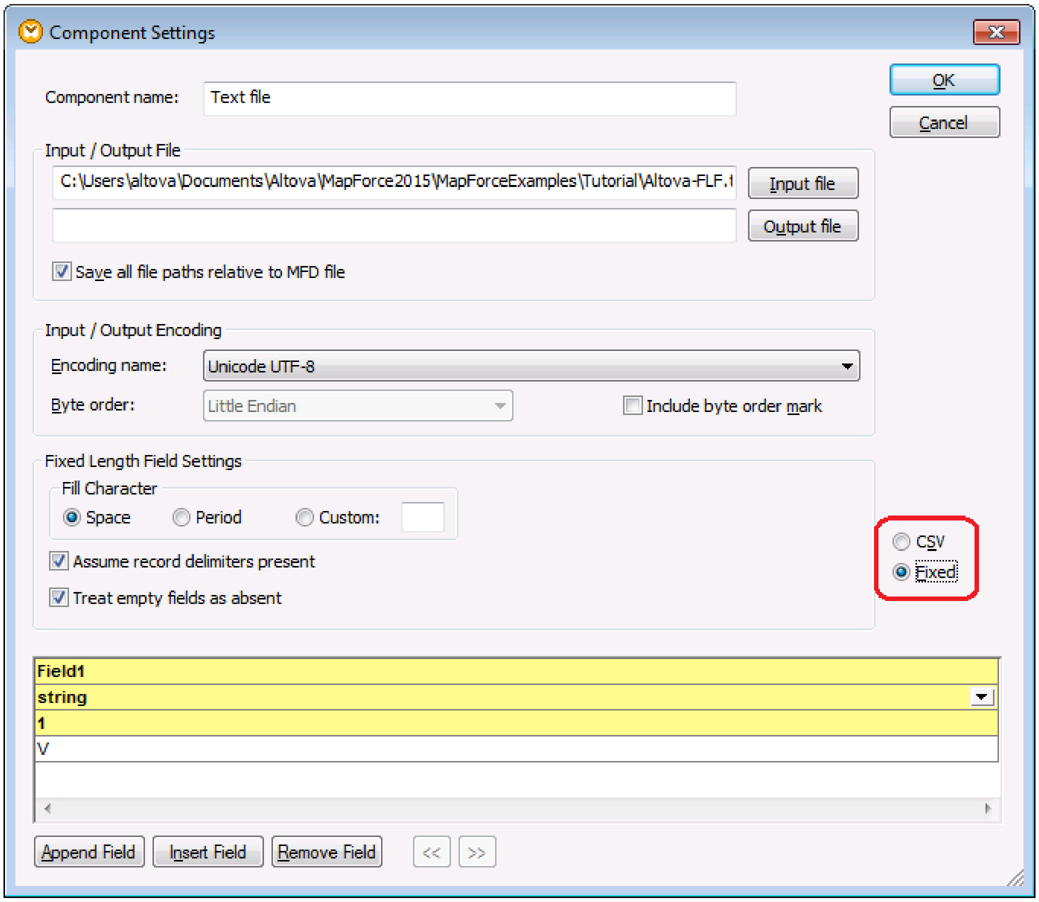
☑ Treat empty fields as absent

● CSV

○ Fixed

Field1
string
Vernon##Callaby###582v.callaby@nanonull.com###Office Manager###########Frank###Further###471f.further@nanonull.com##

3. Select **Fixed**.



Component Settings

Component name: Text file

Input / Output File

C:\Users\altova\Documents\Altova\MapForce2015\MapForceExamples\Tutorial\Altova-FLF.t

☑ Save all file paths relative to MFD file

Input / Output Encoding

Encoding name: Unicode UTF-8

Byte order: Little Endian

□ Include byte order mark

Fixed Length Field Settings

Fill Character

● Space ○ Period ○ Custom:

☑ Assume record delimiters present

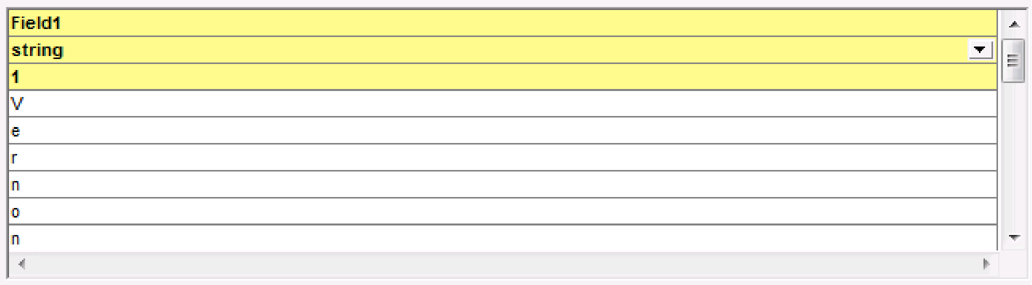
☑ Treat empty fields as absent

○ CSV

● Fixed

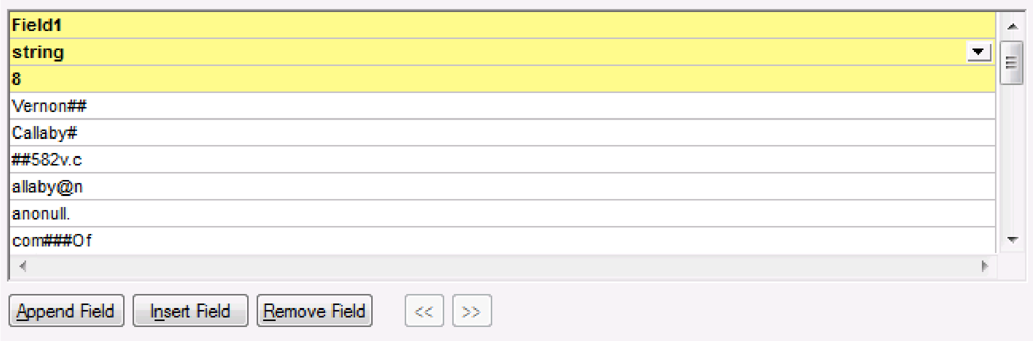
Field1
string
1
V

4. Uncheck the **Assume record delimiters present** check box.



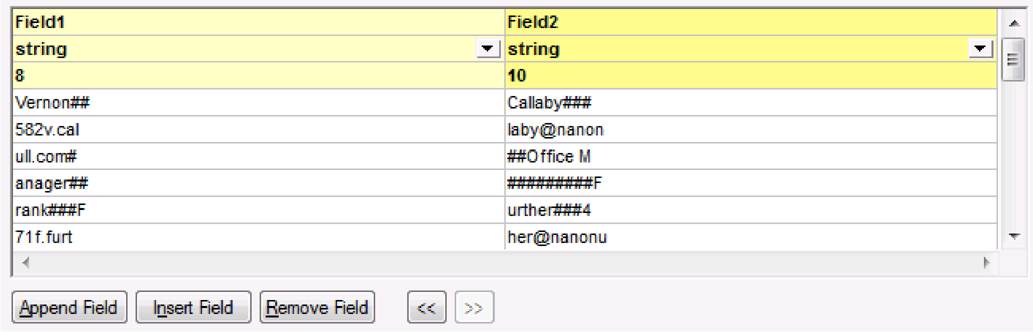
Field1
string
1
V
e
r
n
o
n

5. The three rows highlighted in yellow are editable, and enable you to specify i) the field name ii) the data type and iii) the field size. Type **8** as the new field size, and press **Enter**. More data is now visible in the first column, which is now defined as 8 characters wide.



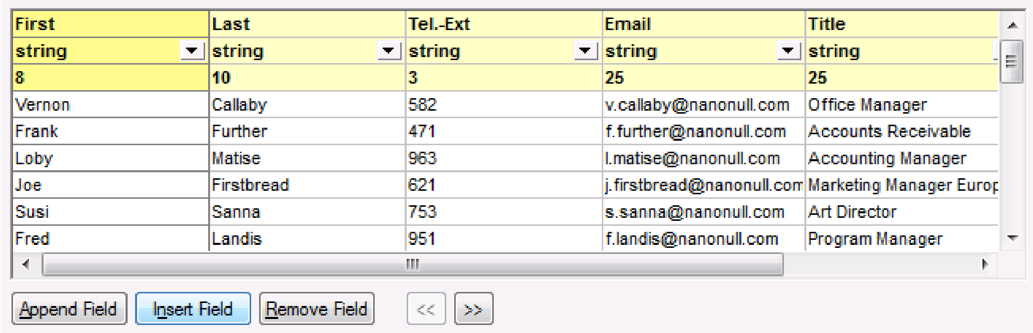
Field1
string
8
Vernon##
Callaby#
##582v.c
allaby@n
anonull.
com###Of

6. Click **Append Field** to add a new field, and set the length of the second field to 10 characters.



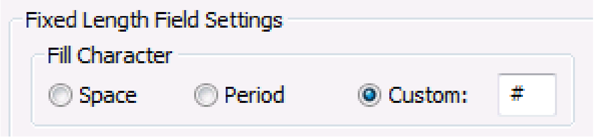
Field1	Field2
string	string
8	10
Vernon##	Callaby###
582v.cal	laby@nanon
ull.com#	##Office M
anager##	#########F
rank###F	urther###4
71f.furt	her@nanonu

7. Use the same method to create three more fields of the following lengths: 3, 25, and 25 characters, and change the field headers to make them easier to map: First, Last, Tel.-Ext, Email, Title. The preview will then look like this:



First	Last	Tel.-Ext	Email	Title
string	string	string	string	string
8	10	3	25	25
Vernon	Callaby	582	v.callaby@nanonull.com	Office Manager
Frank	Further	471	f.further@nanonull.com	Accounts Receivable
Loby	Matise	963	l.matise@nanonull.com	Accounting Manager
Joe	Firstbread	621	j.firstbread@nanonull.com	Marketing Manager Europ
Susi	Sanna	753	s.sanna@nanonull.com	Art Director
Fred	Landis	951	f.landis@nanonull.com	Program Manager

8. In the Fixed Length Field Settings group, select Custom, and type the hash (#) character. This instructs MapForce to treat the # character as fill character.

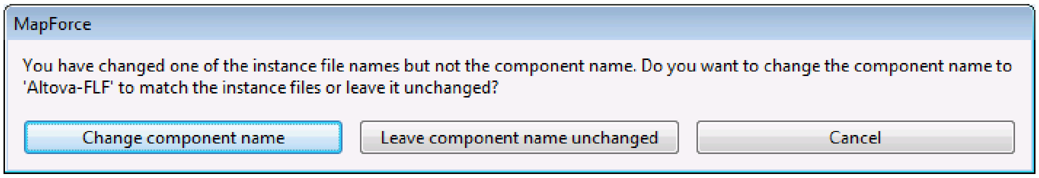


Fixed Length Field Settings

Fill Character

○ Space ○ Period ● Custom: #

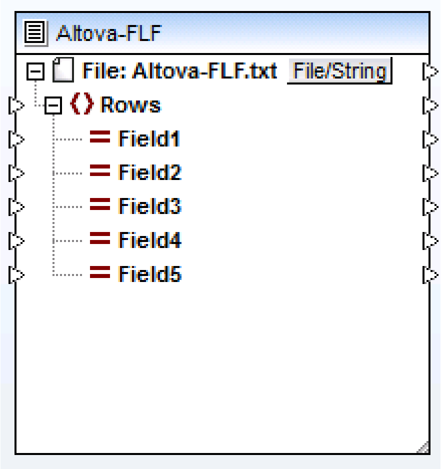
9. Click **OK**.



MapForce

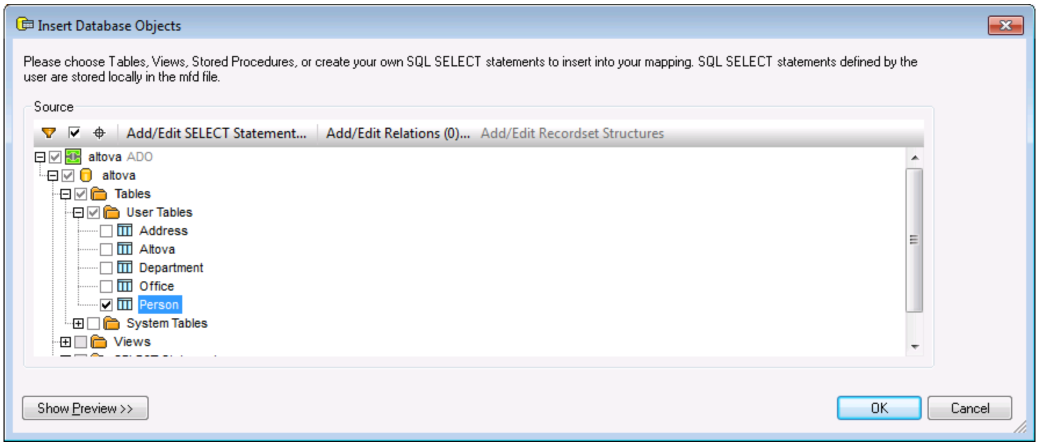
You have changed one of the instance file names but not the component name. Do you want to change the component name to 'Altova-FLF' to match the instance files or leave it unchanged?

10. Click **Change component name**. The Text file component appears in the Mapping window. Data can now be mapped to and from this component.



Step 2: Insert the database component

1. Select the menu command **Insert | Database**, select **Microsoft Access**, and then click **Next**.
2. Select the **altova.mdb** database available in the **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\** folder, and click **Connect**.
3. Select the **Person** table and click **OK**.

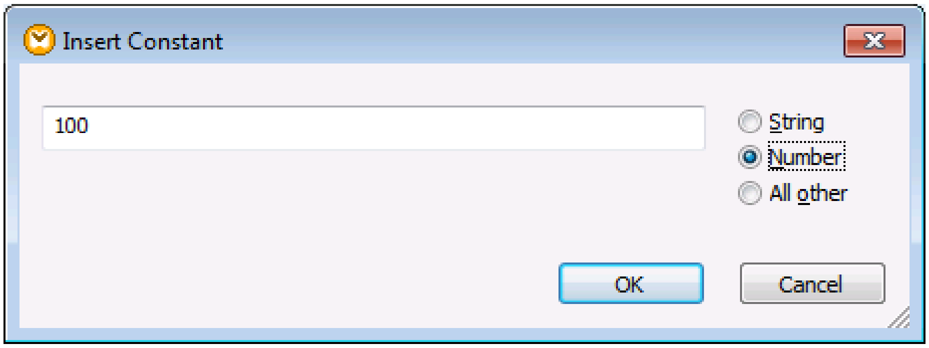


Insert Database Objects

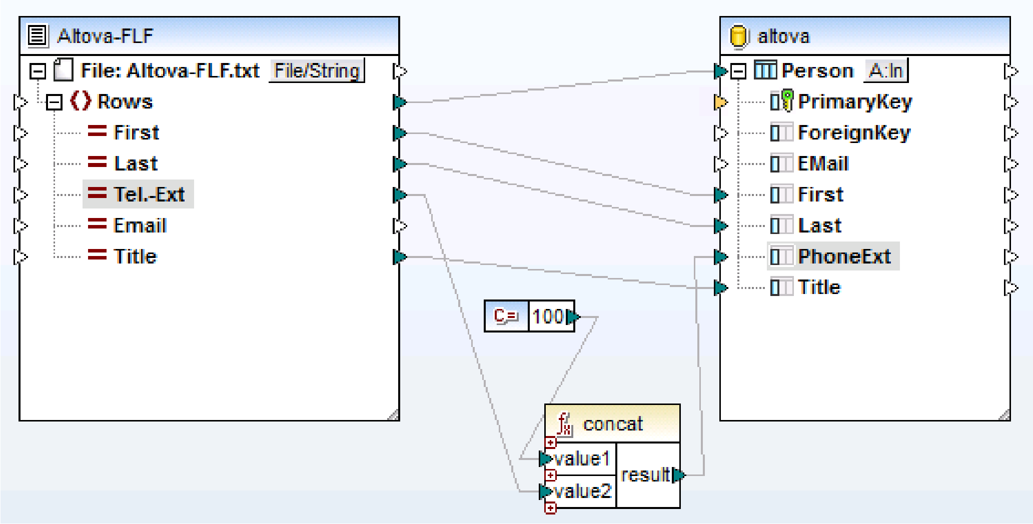
Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

Step 3: Design the mapping

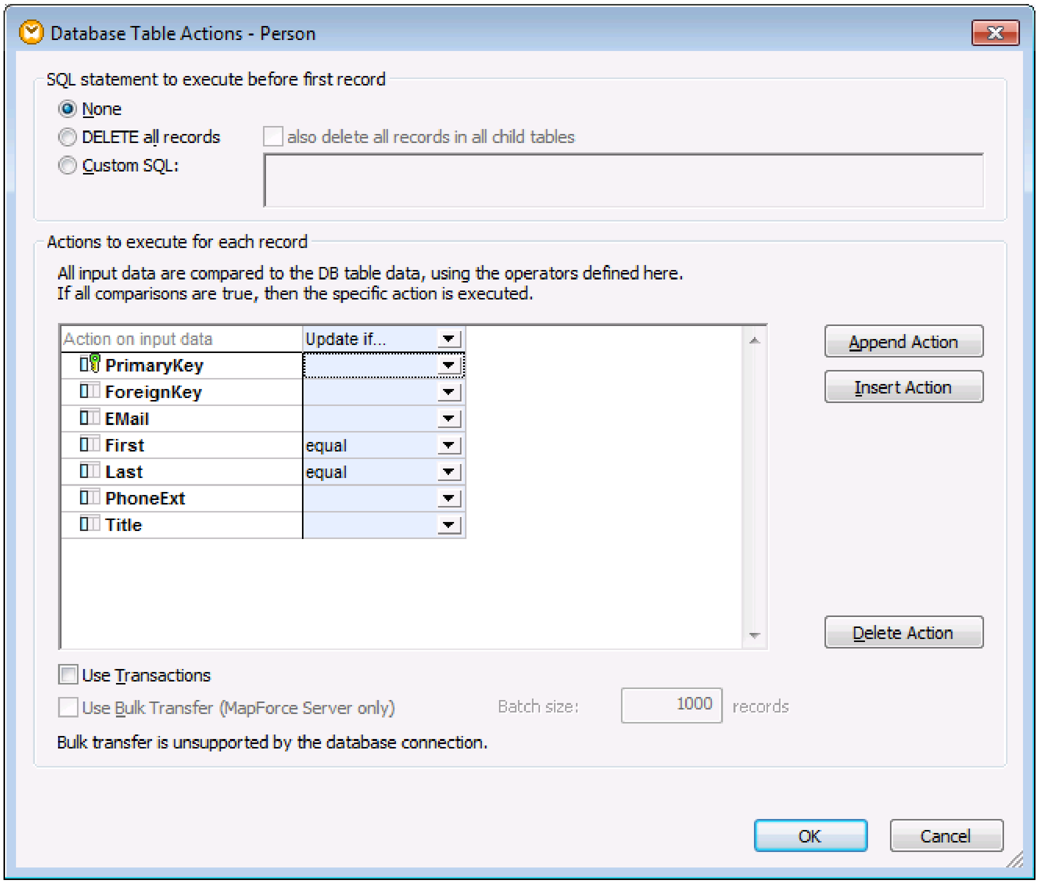
1. Drag the **core | concat** function from the Libraries window into the mapping.
2. Select the menu command **Insert | Constant**, select Number as type, and enter 100 as value. This constant stores the new telephone extension prefix.



3. Create the mapping as shown below.



4. On the database component, click the **Table Action** button A:In next to **Person**.
5. Next to **Action on input data**, select **Update If**, and ensure that the action for **First** and **Last** fields is set to **equal**. This instructs MapForce to update the Person table only if the first and last name in the source file is equal to the corresponding database field. When this condition is true, the action taken is defined by the mapping. In this case, the telephone extension is prefixed by 100, and copied to the **PhoneExt** field of the Person table.



Database Table Actions - Person

SQL statement to execute before first record

● None

○ DELETE all records □ also delete all records in all child tables

○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on input data	Update if...
PrimaryKey	
ForeignKey	
EMail	
First	equal
Last	equal
PhoneExt	
Title	

□ Use Transactions

□ Use Bulk Transfer (MapForce Server only) Batch size: 1000 records

Bulk transfer is unsupported by the database connection.

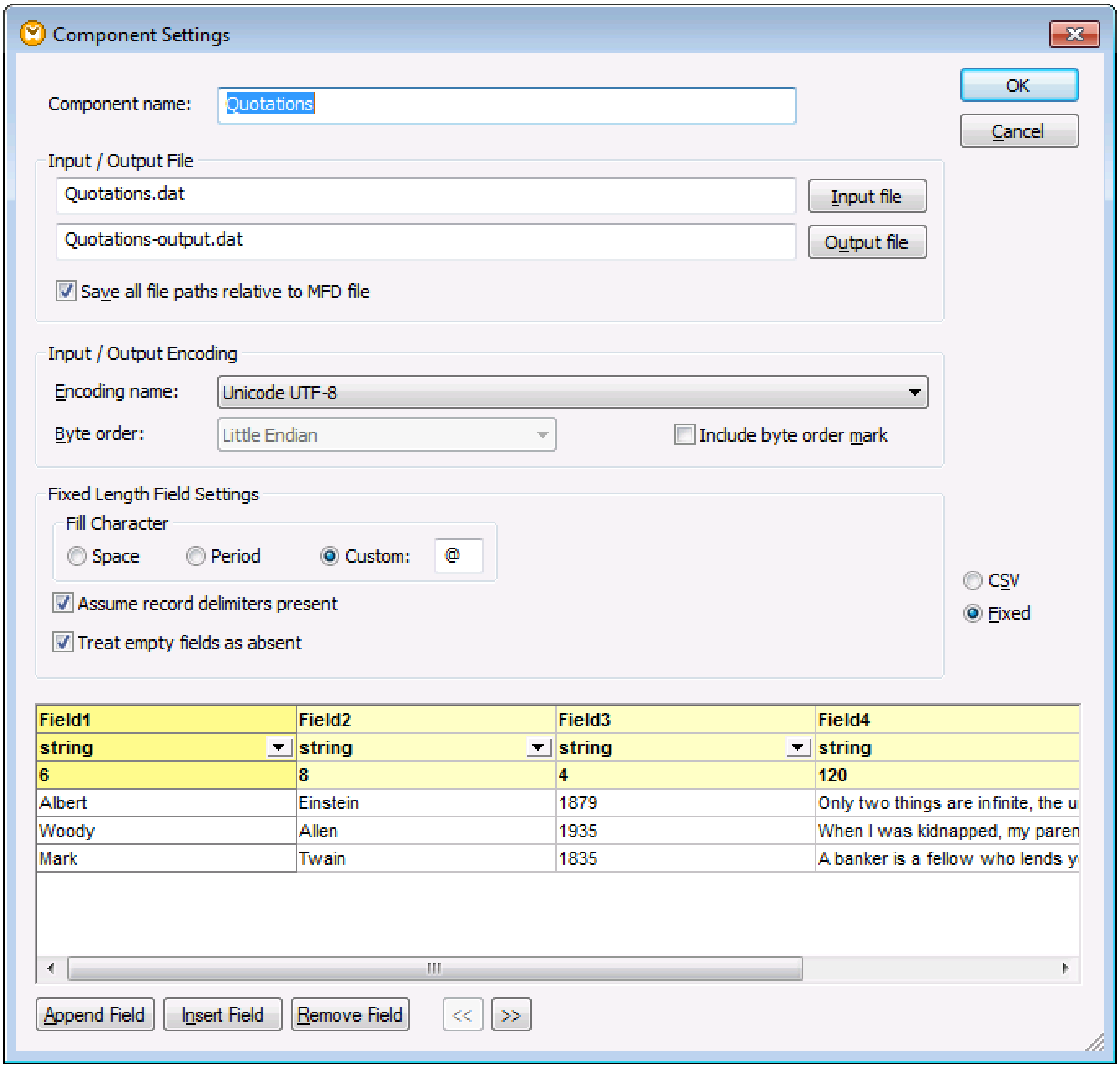
Step 4: Run the mapping

To generate the SQL statement (for preview in MapForce), click the Output tab. To run the SQL statements against the database, click the Run SQL-script button .

7.3.6 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**.



Component Settings

Component name: Quotations

Input / Output File

Quotations.dat

Quotations-output.dat

☑ Save all file paths relative to MFD file

Input / Output Encoding

Encoding name: Unicode UTF-8

Byte order: Little Endian

□ Include byte order mark

Fixed Length Field Settings

Fill Character

○ Space ○ Period ● Custom: @

☑ Assume record delimiters present

☑ Treat empty fields as absent

○ CSV

● Fixed

Field1	Field2	Field3	Field4
string	string	string	string
6	8	4	120
Albert	Einstein	1879	Only two things are infinite, the u
Woody	Allen	1935	When I was kidnapped, my paren
Mark	Twain	1835	A banker is a fellow who lends y

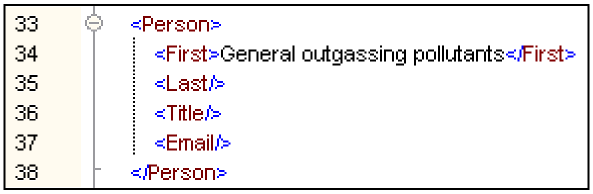
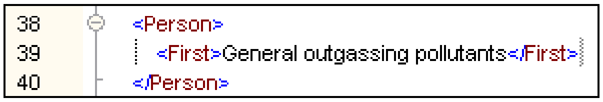
Text Component Settings dialog box (in fixed-length field mode)

The available settings are as follows.

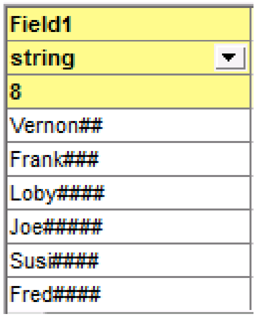
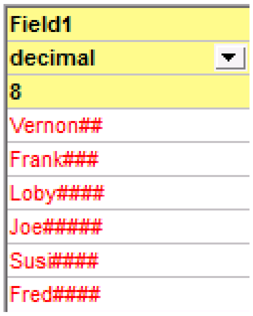
Component name	The component name is automatically generated when you create the component. You can however change the name at any time. If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well. 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.
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.
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 in the Custom field. 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 section below.

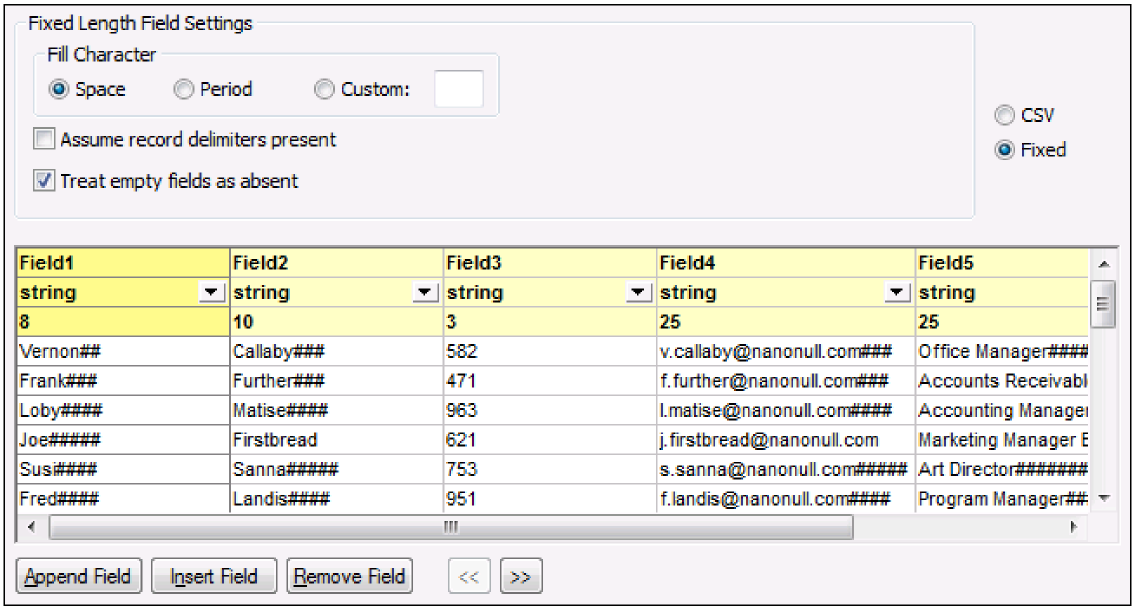
Treat empty fields as absent	When this option is enabled, empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file.
	Assuming that the output is an XML file, when this option is disabled, the empty fields will be created in the output with 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 reco **Insert field** Creates a new field immediately befor currently selected record. **Remove field** Deletes the currently selected field. **<<** Moves the currently selected field one the left. **>>** Moves the currently selected field one the right.

	To change the name of a field, click the header (in this example, **Field1**), and type the new value.  Field1 string 8 Vernon## Frank### Loby#### Joe##### Susi#### Fred#### 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.  Field1 decimal 8 Vernon## Frank### Loby#### Joe##### Susi#### Fred#### To set the size of the field in characters, enter the field size in the third row from the top.

Understanding the "Assume record delimiters present" option

To better understand this option, open the **Altova-FLF.txt** file available in the **<Documents>\Altova\MapForce2018\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.



Fixed Length Field Settings

Fill Character

● Space ○ Period ○ Custom:

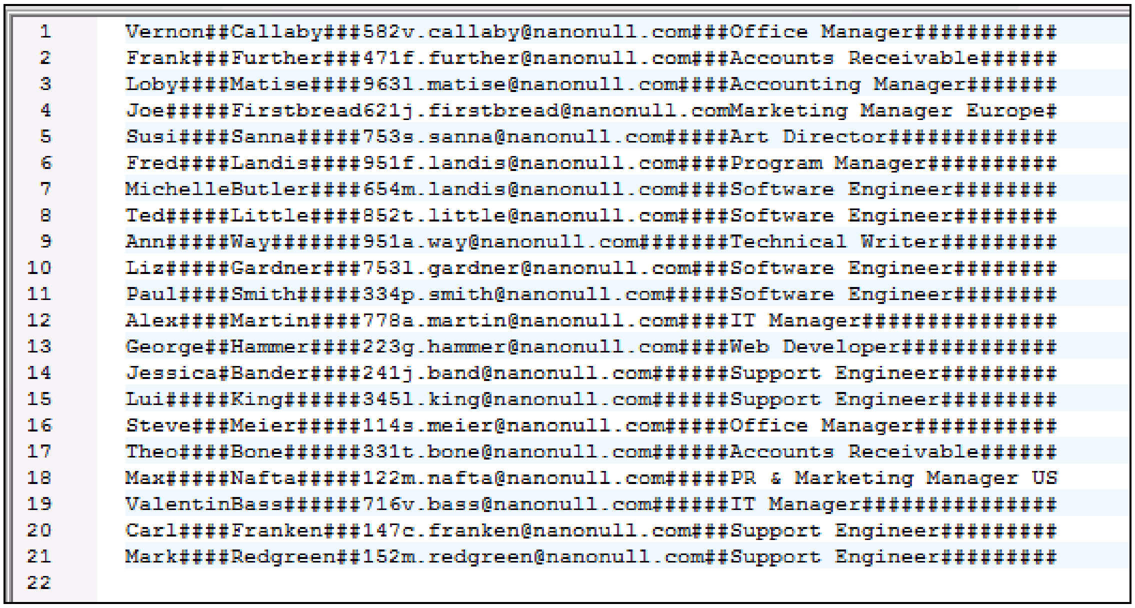
□ Assume record delimiters present

☑ Treat empty fields as absent

○ CSV
● Fixed

Field1	Field2	Field3	Field4	Field5
string	string	string	string	string
8	10	3	25	25
Vernon##	Callaby###	582	v.callaby@nanonull.com###	Office Manager####
Frank###	Further###	471	f.further@nanonull.com###	Accounts Receivabl
Loby####	Matise####	963	l.matise@nanonull.com####	Accounting Manage
Joe#####	Firstbread	621	j.firstbread@nanonull.com	Marketing Manager E
Susi####	Sanna#####	753	s.sanna@nanonull.com#####	Art Director#######
Fred####	Landis####	951	f.landis@nanonull.com####	Program Manager##

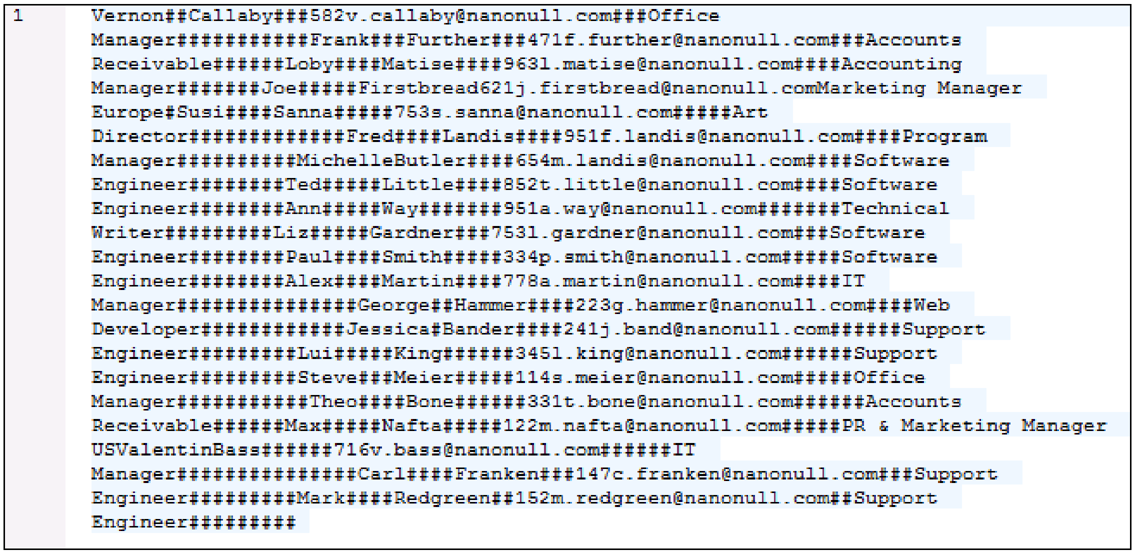
If you would need to write data from this file to a destination file which uses the same structure, then enabling **Assume record delimiters present** creates a new record after every 71 characters.



```
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

7.4 HL7 Version 3

Support for HL7 version **3.x** is automatically included in MapForce 2018 as it is XML based.

A separate installer for the HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the Libraries page of the Altova website (https://www.altova.com/mapforce/download/libraries) Select the Custom Setup in the installer, to only install the HL7 V3 components and XML Schemas.

Location of HL7 XML Schemas after installation:

32-bit MapForce on 32-bit operating system, or 64-bit MapForce on 64-bit operating system	C:\Program Files\Altova\Common2018\Schemas\hl7v3
32-bit MapForce on 64-bit operating system	C:\Program Files(x86)\Altova\Common2018\Schemas \hl7v3

HL7 documents can be used as source and target components in MapForce. This data can also be mapped to any number of XML schema, database or other components.

Chapter 8

Functions

8 Functions

Functions represent a powerful way to transform data according to your specific needs. This section provides instructions on working with functions (regardless if they are built-in to MapForce, defined by you, or reused from external sources). Use the following roadmap for quick access to specific tasks related to functions:

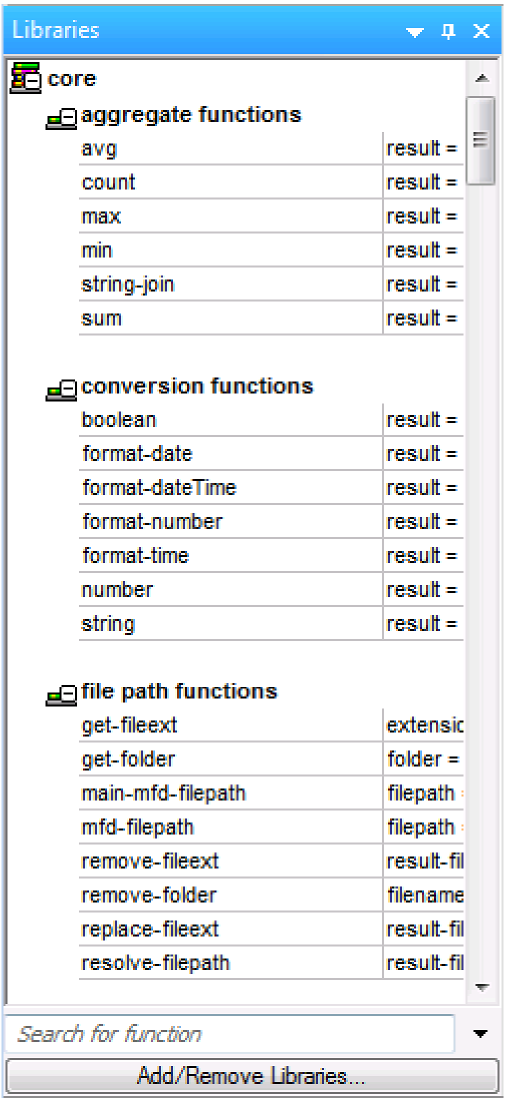
I want to...	Read this topic...
Add MapForce built-in functions or constants to the mapping	• Add a Built-in Function to the Mapping • Add a Constant to the Mapping • Search for a Function • View a Function's Type and Description • Add or Delete Function Arguments
Create my own functions in MapForce	User-Defined Functions
Add custom XSLT functions to MapForce	Importing Custom XSLT 1.0 or 2.0 Functions
Add custom .NET DLL and Java .class libraries to MapForce	• Importing Custom Java and .NET Libraries • Referencing Java, C# and C++ Libraries Manually
Write my own Java library for use with MapForce	Create a Java library
Write my own C# library for use with MapForce	Create a C# library
Write my own C++ library for use with MapForce	Create a C++ library
View all built-in MapForce functions, or look up the description of a specific function.	Function Library Reference

8.1 How To...

8.1.1 Add a Built-in Function to the Mapping

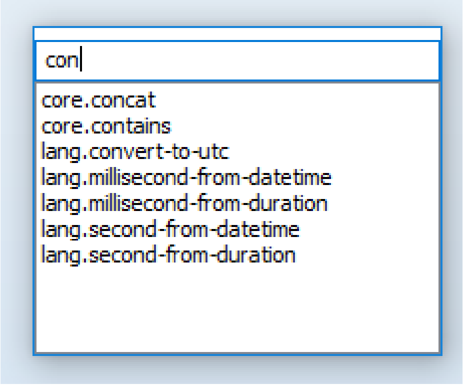
To use a function in a mapping:

1. Select the transformation language (see Selecting a transformation language). Note that the list of available functions depends on the selected transformation language.
2. Click 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 located in the lower part of the window:



Alternatively, you can also quickly add a function to the mapping as follows:

1. Double-click anywhere on the empty area of the mapping and start typing the function name. A combo box appears with the same functions as in the Libraries window, filtered by the text you entered. To see a tooltip with more details about each function, select any function in the list.


con
core.concat
core.contains
lang.convert-to-utc
lang.millisecond-from-datetime
lang.millisecond-from-duration
lang.second-from-datetime
lang.second-from-duration

2. Select the required function, and press **Enter** to add it to the mapping. To close the combo box without selecting a function, press **Escape**, or click anywhere outside the box.

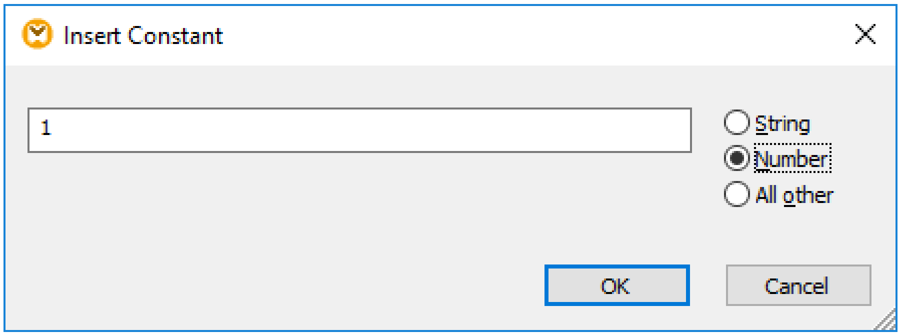
Note: Using the "double-click" alternative way described above, you can also add user-defined functions to the mapping.

8.1.2 Add a Constant to the Mapping

Constants enable you to supply custom text or numbers to the mapping. A constant's value, as the name implies, will remain the same for the duration of the mapping lifetime.

To add a constant to the mapping:

1. Do one of the following:
   a. On the **Insert** menu, click **Constant**.
   b. Right-click the mapping, and select **Insert Constant** from the context menu.


Insert Constant

1

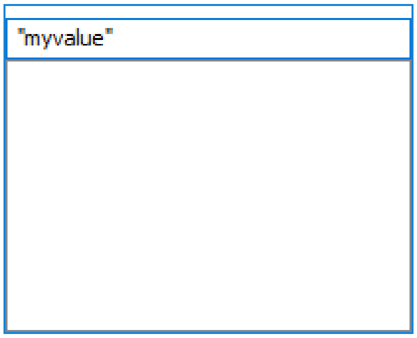
○ String
● Number
○ All other

2. Enter the value of the constant, select the data type ("String", "Number", "All other"), and click **OK**.

Alternatively, you can also quickly add a constant as follows:

1. Double-click anywhere on an empty mapping area.

2. Do one of the following:
   a. To add a string constant, start typing a double quote followed by the constant value. The closing double quote is optional.



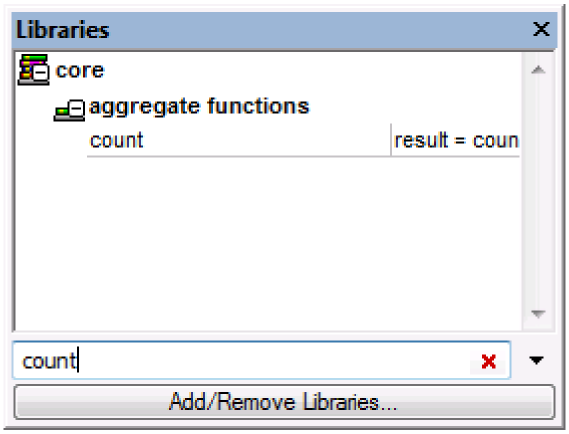
"myvalue"

   b. To add a numeric constant, just type the number.
3. Press **Enter**.

8.1.3 Search for a Function

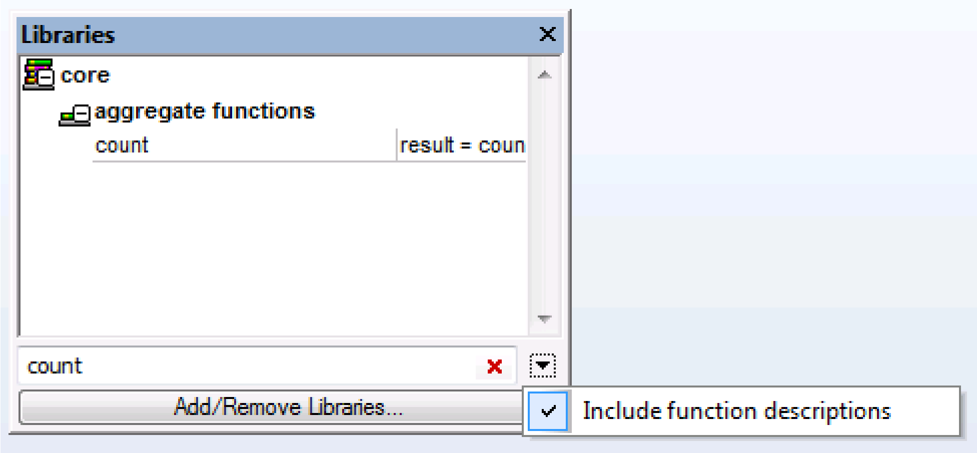
To search for a function in the Libraries window:

1. Start typing the function name in the text box located in the lower part of the Libraries window.



count

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 **Include function descriptions** option.



To cancel the search, press the **Esc** key or click ✕.

> The functions available in the Libraries window depend on the transformation language currently selected, see Selecting a Transformation Language.

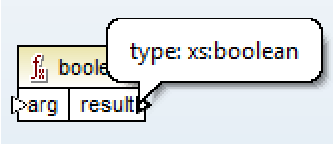
To find all occurrences of a function within the currently active mapping:

- 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.

8.1.4 View a Function's Type and Description

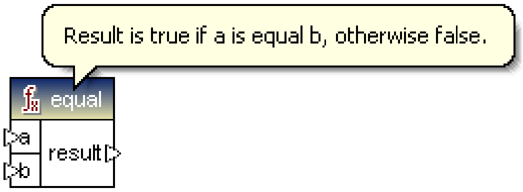
To view the data type of a function input or output argument:

1. Make sure that the **Show tips** INFO toolbar button is enabled.
2. Move your mouse over the argument part of a function.



To view the description of a function:

1. Make sure that the **Show tips** INFO toolbar button is enabled.
2. Move your mouse of the function (this works both in the Libraries pane and on the mapping area)

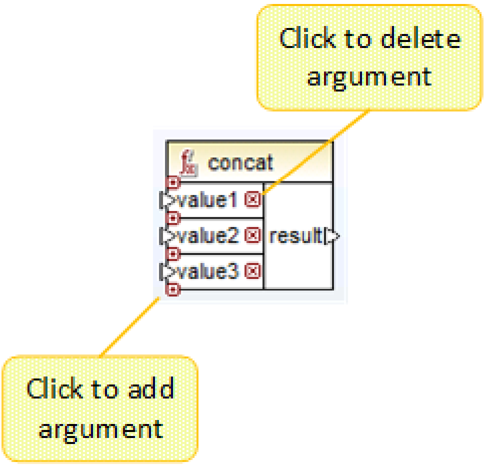


Result is true if a is equal b, otherwise false.

8.1.5 Add or Delete Function Arguments

To add or delete function arguments (for functions where that is applicable):

- Click **Add parameter** (⊡) or **Delete parameter** (⊠) next to the parameter you want to add or delete, respectively.



Click to delete argument

Click to add argument

Dropping a connection on the ⊡ symbol automatically adds the parameter and connects it.

8.2 Defaults and Node Functions

When MapForce reads or writes data, it is often the case that either the source or destination file or database has empty or null fields. To handle such cases, MapForce provides various built-in functions, if-else conditions, and other mechanisms that let you replace missing or null data with something else, or perhaps throw an exception when missing fields are encountered.

Furthermore, you may want to set a default value for multiple items simultaneously (for example, all children of an XML element). Alternatively, you may want to create a simple function that substitutes an empty value with some text (for example, "n/a"), and then apply this function to multiple items. Under normal circumstances, in order to do this, you would need to copy-paste the same function multiple times on the mapping. However, this would also add clutter to the mapping and make it more difficult to understand. As a simpler alternative, you could use defaults and node functions, which are the subject of this chapter.

Note: Defaults and node functions are supported when the target language of the mapping is BUILT-IN. 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.

The term "node function" means that the function applies at node level, be it an XML node or CSV, JSON, EDI, or database field. The node function may apply either to a single item or to multiple items at once. Likewise, the term "default" refers to a default value that you want to apply at node level, for either a single item or multiple items. Note that, at mapping runtime, a node function or default is called once for each item in a sequence.

Defaults and node functions are particularly useful when you want to apply the same processing logic to multiple descendant items in a structure, for example:

- Every time when an empty or null value is encountered, replace it with some other value, and do this recursively for all descendant items
- Every time when a specific value is encountered (for example, "N/A"), 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 string or custom text (or with 0, in case of numeric fields) when reading from a database table
- Trim all trailing spaces for all values that are coming from some source database
- Append a custom prefix or suffix to all values that are written to a target file or database
- Produce a null value each time when a specific value is encountered

Note: It is important to distinguish between "null" and "empty" values, since they are not the same. A null value means "nothing" (the absence of a value), whereas an empty value is typically an empty string (''). MapForce provides various ways to handle both, including (but not limited to) node functions and defaults.

8.2.1 How to Create Defaults and Node Functions

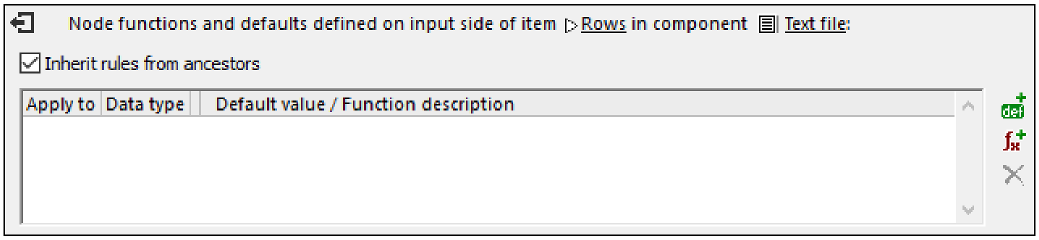
In order to create a default or node function, first determine the item (node, or field) where you want to define the default or node function. This can be either a "leaf" item (with no descendants) or an item that has descendants. In the latter case, it will be possible to apply the function or default to all descendant items as well.

Prerequisites:

- You can create defaults or node functions either on an 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 Choosing the Input or Output Side.
- Defaults and node functions require that the connection type between source and target is either "Source Driven" or Target Driven". "Copy-All" connections are not supported. Specifically, node functions and defaults are not applied to descendants of "Copy-All" connections. The node that has the "Copy-all" connection itself will apply node functions and defaults, but only if it has a simple value, for example, an XML element with simple-type content and attributes. Therefore, if you want to define a function or set a default on a node with descendants, the connection type between source and target must not be "Copy-All". To view or change the connection type, right-click the connection and select **Target Driven (Standard)** from the context menu. For more information, see Connection Types.
- Note that creating defaults or node functions is not supported for the "File" node. This node lets you create or read file names dynamically, see Processing Multiple Input or Output Files Dynamically.

To create a default or a node function:

1. Right-click the item (node) of interest, and select **Node Functions and Defaults | Input Node Functions and Defaults** from the context menu (or **Output Node Functions and Defaults**, depending on the case). Alternatively, right-click a connector—in this case, MapForce will show the node function command for that side only. The **Mapping** pane displays a grid at the top, for example:


Node functions and defaults defined on input side of item Rows in component Text file:
☑ Inherit rules from ancestors
Apply to
Data type
Default value / Function description

If the item where you define the rule has a parent, the parent may also have rules (node functions or defaults) defined against it. To inherit such rules, select the **Inherit rules from ancestors** check box. For more information about inheritance, see How Defaults and Node Functions Work.

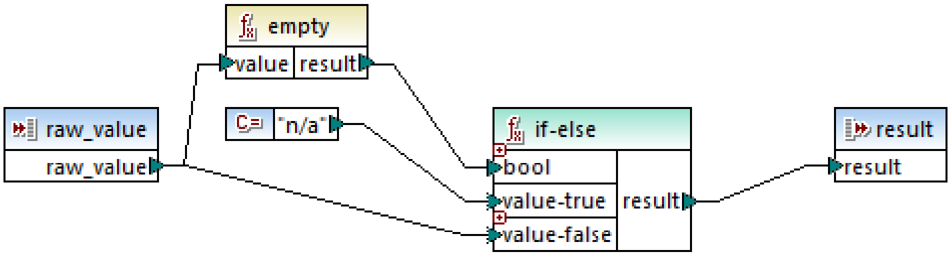
2. Do one of the following:
   a. To add a default, click **Add Default** (def).
   b. To add a function, click **Add Function** (fx).

This creates a new rule (a row in the grid at the top of the **Mapping** pane where you can choose the criteria for this rule). Configure the rule as follows:

Apply To	Select whether the rule should apply to the current item, or to all descendant items regardless of their depth, or to direct child items only.

	If the item you selected in step 1 has no descendants, then "Current item" is the only choice.
Data Type	Click the **Ellipsis** button … and select a data type from the dialog box. The rule (default or node function) will apply only to items that have this data type (or a derived data type). If the item you selected in step 1 is one without descendants, then the item's data type is the only choice.
Default Value / Function Description	If you are defining a default (def), type here the default value that you wish to set for the selected item (and all descendants, if applicable). To set an empty string as default, leave this field empty. If you are defining a function (fx), this field is for information purpose only. It displays a summary of the function. You can define the function's body in the next step.

3. If you are defining a function, the mapping area changes to display the function's input (illustrated below as "raw_value") and output ("result"). This mapping area is a mini-mapping, and the same general rules apply here as when you define a standard mapping. For example, the body of a function could look as follows:



The node function illustrated above replaces any empty value with the value "n/a". For more information about this example, see Example: Replace Empty CSV Fields.

Note the following:

- Inside a node function, only certain MapForce components meaningful in this context are supported, such as built-in functions, variables, if-else conditions, and others. Complex structures such as XML, JSON, EDI, or databases are not supported. Adding inline user-defined functions or join components to a node function is also not supported.
- Never delete the function's input component ("raw_value"), even if you don't need an input for your function; otherwise, validation errors will appear when you run the mapping. The same applies for the function's output. Should you need to restore an accidentally deleted input component, run the menu command **Function | Insert Input**.
- In some cases, you might find it more convenient to create a node function as follows: drag a function from the **Libraries** window to an input or output connector ▷ . This makes sense only for simple functions like `right-trim` or `uppercase`.

To exit a node function:

- Click the **Go back** ⏎ button in the upper-left corner of the **Mapping** pane, or press **Escape**.

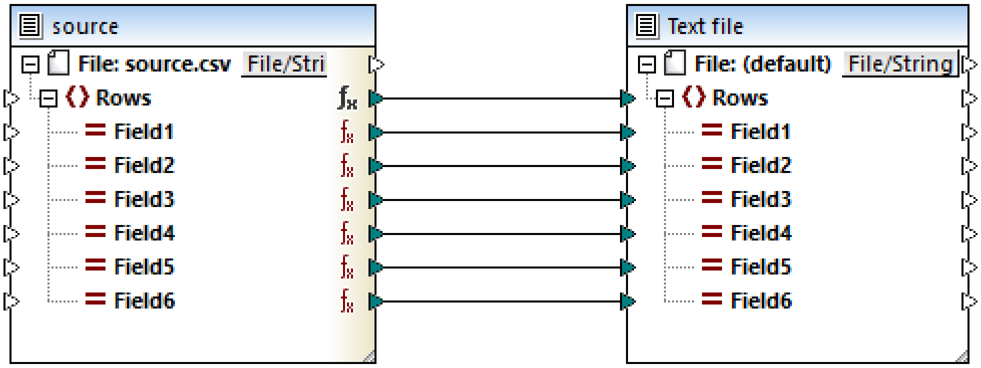
To view or modify a previously defined node function:

1. In the main mapping, click the fx icon (black or red color) next to the node of interest. This icon is present on any node where you previously defined a node function. For more information about the meaning of node function icons, see How Defaults and Node Functions Work.
2. Select the function from the grid at the top of the **Mapping** pane, and click Edit . If the Edit button is not present, then the function is most likely defined on some ancestor, not on the current item (see previous step).

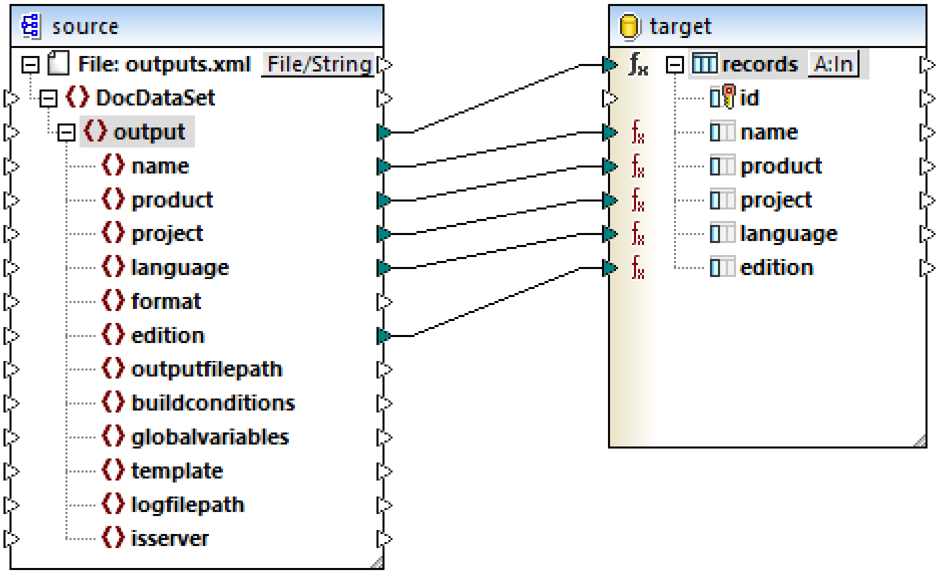
8.2.2 Choosing the Input or Output Side

Because any MapForce component always has an input and an output side, you can define a node function or a default value on either side, depending on your needs. To understand this better, let's recall how a mapping works: it first reads data from a source component (for example, a database or a file), then optionally processes it in some way (for example, using functions or filters), and finally writes data to some target component (for example, a database or a file). Considering this, you can apply node functions and set defaults at various stages:

- Immediately after data is read from the source file or database (but before it is further processed by your mapping). For example, in the mapping below the function or default is defined on the output side of the source component (notice the fx icon, which denotes that node functions or defaults are present):



- Immediately before data is written to the target file or database (and after it finished all intermediary processing). For example, in the mapping below, the function or default is defined on the input side of the target component:



- At an intermediary stage in the mapping process. For example, if the mapping contains an intermediary variable of complex type (say, an XML structure), you could trim all values before they are supplied to the XML structure, or immediately after they are returned by the XML structure).

To summarize the above, you can define node functions either on the "input" or the "output" side of a component. Functions (or defaults) defined on the input side will process data before it enters the corresponding item on the component. Conversely, when defined on the output side, they will process data immediately after it is returned by the corresponding item. If the item where you defined the function has child items, then you can optionally propagate the default function to apply to all children as well.

8.2.3 How Defaults and Node Functions Work

As explained in How to Create Defaults and Node Functions, you can create node functions or defaults for nearly any item (node) on the mapping. Let's call this process defining a *rule*. Rules have the following important characteristics that make them extremely flexible:

- *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 where 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.
- *Data type 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 a function) for all items of string type, and a different default (or a function) for all items of numeric type.

The behavior described above has implications. Namely, it is important to make a difference between defining a rule and actually applying one. When you define a rule on some item, it does not necessarily mean that the rule will affect that item. The rule will apply to the item or its descendants only if the rule criteria (data type and inheritance) allow it.

To help you understand which rules are defined and which ones apply, MapForce provides the following visual clues on the mapping:

Icon	Description
fx	This icon (black color) indicates that a rule is defined for this item, and may affect all its descendants. Click the icon to modify or delete the rule.
fx	This icon (red brick color) indicates that the item qualifies (is eligible) for a rule defined at some ancestor level. In other words, there exists a rule that *applies to* (and may affect) this item.
fx	This icon (bold, red brick color) indicates that a rule is defined for this item, and at the same time a rule applies to this item. This icon usually appears when a default is defined for a single node.
fx	This icon indicates that, even though a rule applies to this item, it is deliberately blocked. You can do this for certain items where you do not want the rule to apply. **Note:** This icon is displayed only if inheritance is blocked and no other rules are defined at this node. If a rule from an ancestor does apply, the fx icon has priority.
fx	This icon (grayed out) indicates that, even though a rule applies to this item, it is inactive. For example, this icon may appear for items that are not connected yet on the mapping.

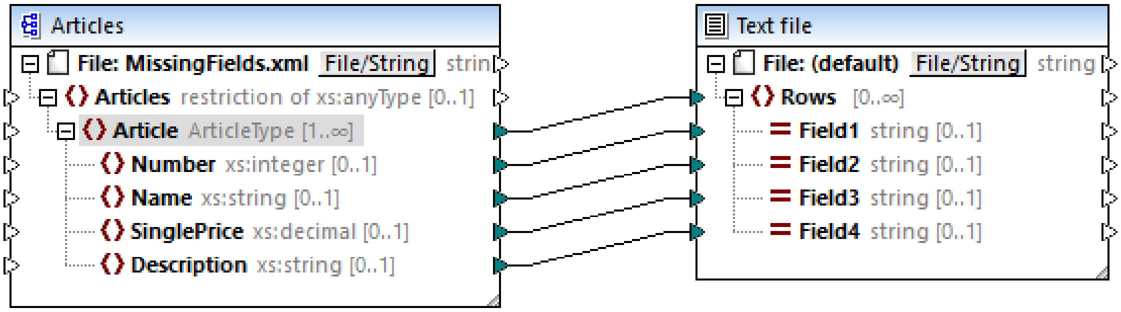
In general, when multiple node functions or defaults exist for one and the same item, keep in mind the following rule of thumb:

> For any single item on the mapping, MapForce always applies only one node function and only one default, regardless of how many node functions or defaults qualify to apply for that item.

In practice, this translates as follows:

- When multiple rules exist for one and the same item, MapForce will apply to an item the rule that is closer to that item. For example, let's assume that you have defined a node function three times: on a root XML node called **Company**, on its child node called **Department**, and on the grandchild **Employee**. In this case, MapForce will apply to the **Employee** item the function defined on the **Employee** item, since it is closer. Had there been no function there, it would look up to find the function of the immediate ancestor, **Department**. If there is no function for **Department**, then it looks further up to the root node, which in this case is **Company**. Inheritance is optional; to disable it, clear the **Inherit rules from ancestors** check box. When this check box is cleared, the item gets the "blocked rule" fx icon.
- When one and the same item has multiple rules, then MapForce applies the first matching rule from the grid at the top of the Mapping pane. To change the order of rules in this grid, click a rule and then drag and drop to a new position within the grid. Note that you can drag a rule in the grid only if it is defined for the current item. You cannot change the position of inherited rules; you can only enabler or disable inheritance.

To better illustrate how this works, we will use a mapping available at the following path: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\MissingFields.mfd**.


Articles
File: MissingFields.xml File/String strin
Articles restriction of xs:anyType [0..1]
Article ArticleType [1..∞]
Number xs:integer [0..1]
Name xs:string [0..1]
SinglePrice xs:decimal [0..1]
Description xs:string [0..1]
Text file
File: (default) File/String string
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

MissingFields.mfd

As shown above, this mapping reads data from a source XML file into a target text file (fixed-length fields). In the source XML file, the element **Article** has child elements of different type: "integer", "string", and "decimal". Note that each child element is optional (`minOccurs="0"`). Therefore, if any of these elements does not exist in the source XML, you will want to provide a default value; otherwise, you will see empty fields in the target CSV file, for example:

```
   T-Shirt   25    Available in all sizes
2            2.3
3  Pants           Limited stock
4  Jacket    57.5
```

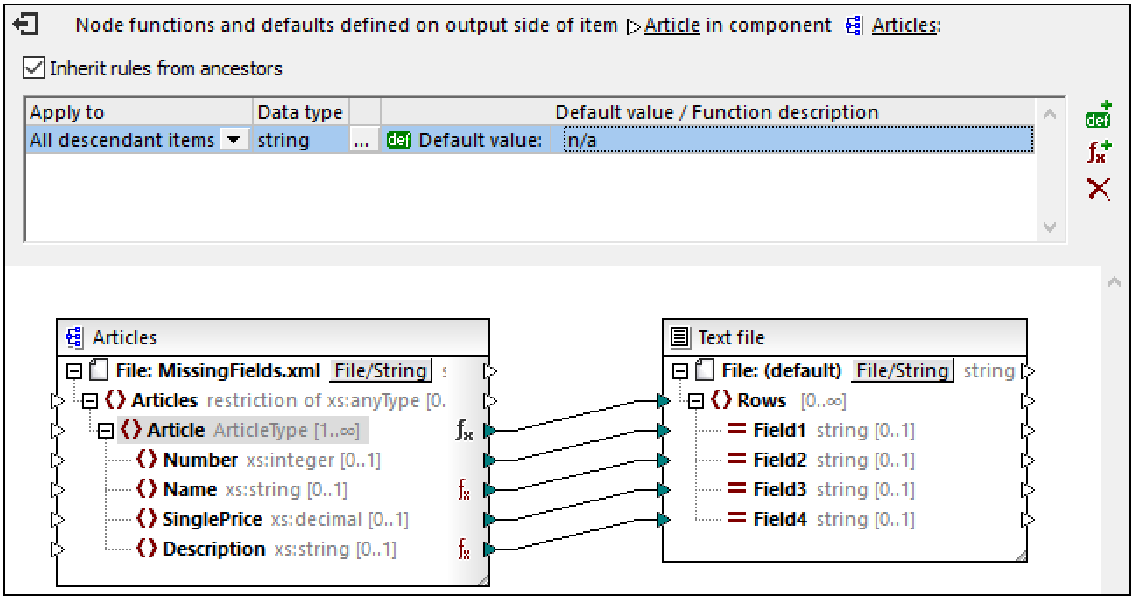
Below we illustrate various ways to handle missing data by means of rules, along with explanations of how rules affect the mapping result. They will also help you understand or control which rule should prevail when multiple rules exist for a given item.

Example 1: Provide defaults for all string items

Given the mapping **MissingFields.mfd**, let's assume that you have the following requirement: *If any child of **Article** is of type "string" and is missing, use "n/a" as default value.*

To satisfy this requirement, take the steps below:

1. Right-click the **Article** item, and select **Node Functions and Defaults | Output Node Functions and Defaults** from the context menu.
2. Click **Add default** (def).
3. Under **Default value**, type "n/a" and press **Enter**.



Node functions and defaults defined on output side of item ▷Article in component Articles:

☑ Inherit rules from ancestors

Apply to	Data type		Default value / Function description
All descendant items	string	...	def Default value: n/a

Articles
File: MissingFields.xml File/String
Articles restriction of xs:anyType [0.
Article ArticleType [1..∞]
Number xs:integer [0..1]
Name xs:string [0..1]
SinglePrice xs:decimal [0..1]
Description xs:string [0..1]
Text file
File: (default) File/String string
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

In the mapping above, the rule criteria are set as follows: *For all descendant items of **Article**, if the data type is "string", and if the source XML element is missing, use the default value "n/a".* In this example, there are two items of type "string", **Name** and **Description**, so the rule will apply to both.

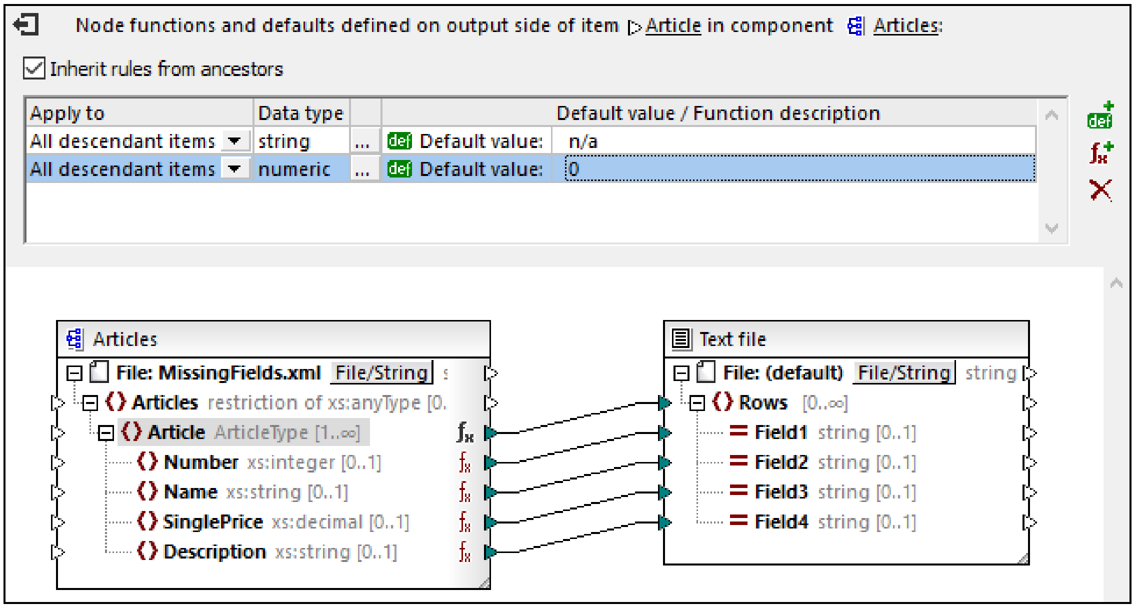
As stated before, the item where a rule is defined has the ƒx icon next to it. Items where the rule will apply have the ƒx icon. If you preview the mapping at this stage, you can see that all missing strings have now been replaced with "n/a" in the output:

```
   T-Shirt   25    Available in all sizes
2  n/a       2.3   n/a
3  Pants           Limited stock
4  Jacket    57.5  n/a
```

Example 2: Provide defaults conditionally based on data type

Let's now assume that, in addition to defaults for string items, you must also supply a default value **0** for any item of numeric type. To satisfy this requirement, take the steps below:

1. Click the **Article** item.
2. Click **Add default** (def) and add a second rule with the following criteria:



Node functions and defaults defined on output side of item Article in component Articles:

☑ Inherit rules from ancestors

Apply to	Data type		Default value / Function description	
All descendant items	string	...	Default value:	n/a
All descendant items	numeric	...	Default value:	0

Articles
File: MissingFields.xml File/String
Articles restriction of xs:anyType [0.
Article ArticleType [1..∞]
Number xs:integer [0..1]
Name xs:string [0..1]
SinglePrice xs:decimal [0..1]
Description xs:string [0..1]
Text file
File: (default) File/String string
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

In the mapping above, the rule criteria are set as follows:

- *For all descendant items of **Article**, if the data type is "string", and if the source XML element is missing, use the default value "n/a"*
- *For all descendant items of **Article**, if the data type is numeric, and if the source XML element is missing, use the default value "0".*

Consequently, the output looks as follows:

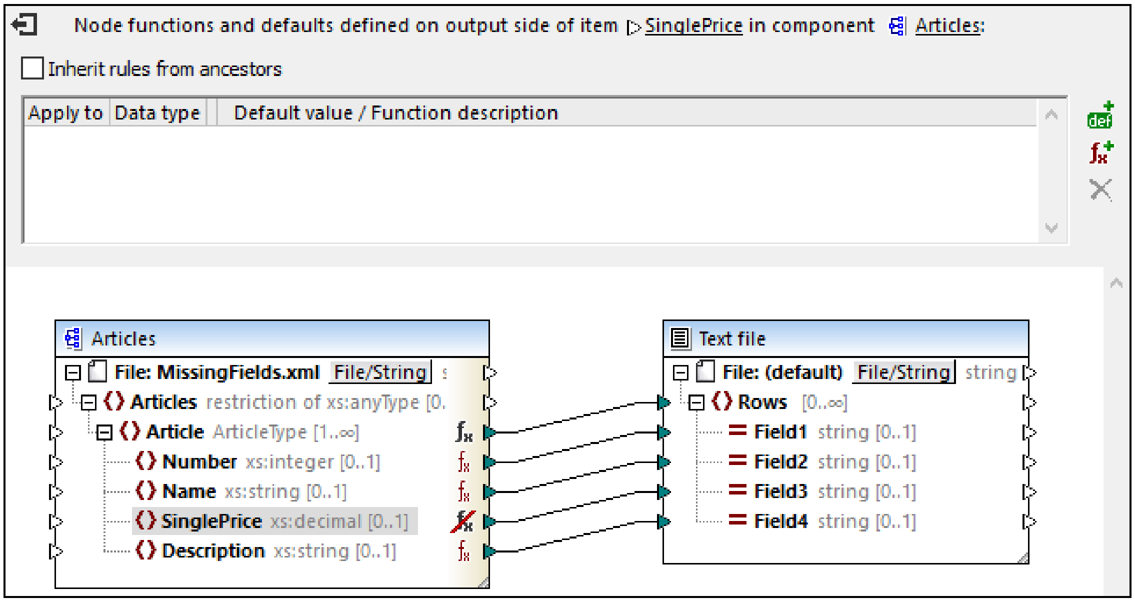
```
0  T-Shirt   25    Available in all sizes
2  n/a       2.3   n/a
3  Pants     0     Limited stock
4  Jacket    57.5  n/a
```

Note: The data type "numeric" is actually a type category, because it includes both the "integer" and "decimal" data types. It also includes the types "float" and "double", although such types are not present here. In this example, the rule will apply to both **Number** and **SinglePrice** elements. If you select "decimal" as data type, the rule will still apply to both **Number** and **SinglePrice**, because type "integer" derives from type "decimal", in the XML schema type hierarchy (see §3 in "XML Schema Part 2: Datatypes Second Edition", https://www.w3.org/TR/xmlschema-2). If you select "integer" as data type, however, the rule will apply only to **Number**.

Example 3: Block rule for a specific item

Let's now assume that you still want to apply defaults for all string and numeric items, like in the previous example. However, you do not want to set any default to the **SinglePrice** item.

To satisfy this requirement, click the item **SinglePrice**, and then clear the check box **Inherit rules from ancestors**.



Node functions and defaults defined on output side of item ▷ SinglePrice in component Articles:

□ Inherit rules from ancestors

Apply to	Data type	Default value / Function description

def
fx
Articles
File: MissingFields.xml File/String
Articles restriction of xs:anyType [0..
Article ArticleType [1..∞]
Number xs:integer [0..1]
Name xs:string [0..1]
SinglePrice xs:decimal [0..1]
Description xs:string [0..1]
Text file
File: (default) File/String string
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

In the mapping above, the item **SinglePrice** no longer inherits rules from its parent, **Article**. Therefore, a "blocked rule" icon fx appears next to it.

Consequently, the corresponding field still appears empty in the output:

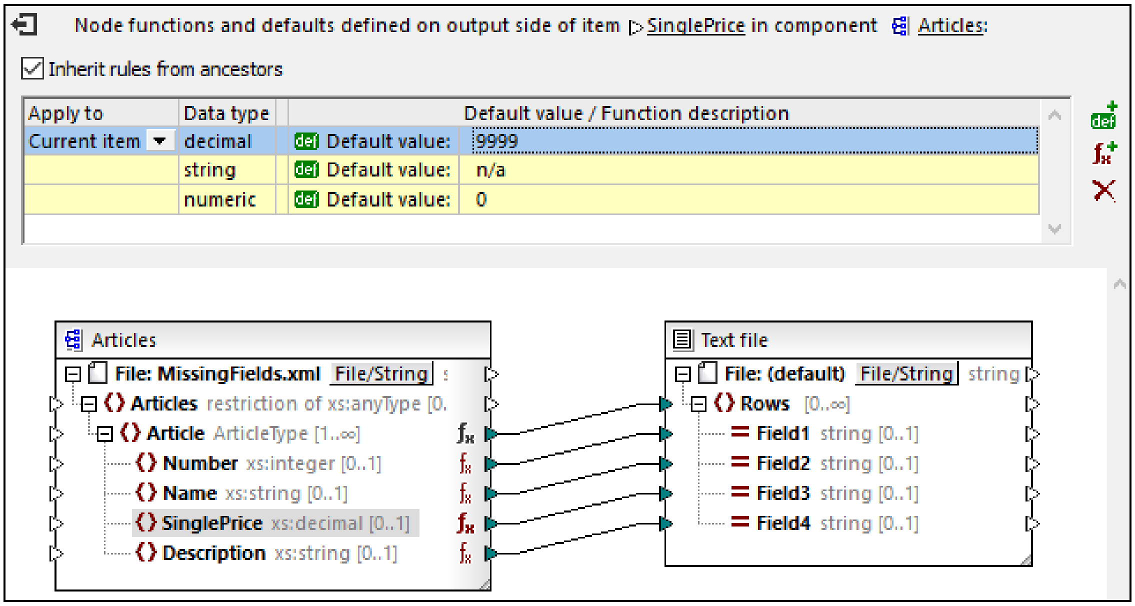
```
0  T-Shirt   25    Available in all sizes
2  n/a       2.3   n/a
3  Pants           Limited stock
4  Jacket    57.5  n/a
```

Example 4: Override inherited rule for a specific item

Let's assume that you still want to supply defaults for all string and numeric values; however, for item **SinglePrice** exclusively, you want to set a default value of **9999**.

To satisfy this requirement, take the steps below:

1. Click the item **SinglePrice**.
2. Click **Add default** (def) and type a default value of **9999**.
3. Optionally, select the **Inherit rules from ancestors** check box. This step is merely to illustrate that, in this case, the inherited rules will be overridden anyway.



Node functions and defaults defined on output side of item ▷SinglePrice in component Articles:

☑ Inherit rules from ancestors

Apply to	Data type	Default value / Function description	
Current item	decimal	Default value:	9999
	string	Default value:	n/a
	numeric	Default value:	0

Articles
File: MissingFields.xml File/String
Articles restriction of xs:anyType [0.
Article ArticleType [1..∞]
Number xs:integer [0..1]
Name xs:string [0..1]
SinglePrice xs:decimal [0..1]
Description xs:string [0..1]
Text file
File: (default) File/String string
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

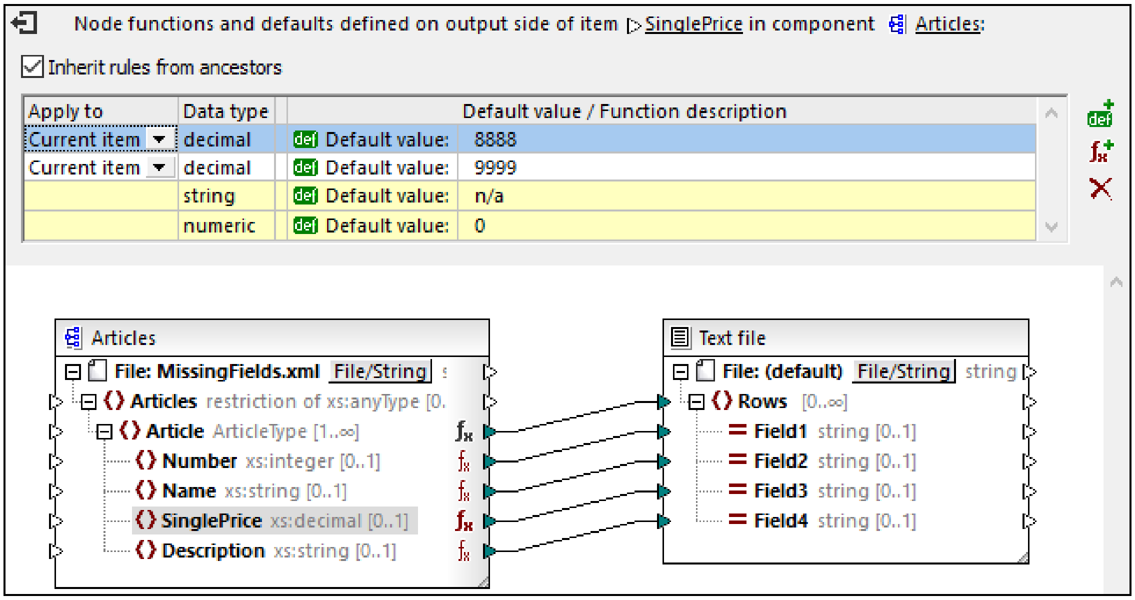
Note: Inherited rules have yellow background.

In the mapping above, there are three rules that may apply for item **SinglePrice**: two inherited ones, and a direct one. In this case, the rule defined directly on the item wins. The inherited rules will be disregarded. Therefore, the output looks as follows:

```
0  T-Shirt   25    Available in all sizes
2  n/a       2.3   n/a
3  Pants     9999  Limited stock
4  Jacket    57.5  n/a
```

Example 5: Set the priority of rules

Let's expand the previous example further and assume that you define one more rule for item **SinglePrice**, a default of **8888**. As stated before, the rule defined directly on the current item wins. However, since two rules now exist on the current item (in addition to the inherited ones), the legitimate question is, which of the two defaults will apply, **8888** or **9999**?



Node functions and defaults defined on output side of item ▷SinglePrice in component Articles:

☑ Inherit rules from ancestors

Apply to	Data type	Default value / Function description
Current item	decimal	Default value: 8888
Current item	decimal	Default value: 9999
	string	Default value: n/a
	numeric	Default value: 0

When multiple rules exist for the same item like in the mapping above, you can choose the winning rule manually, by dragging it up to the top of the grid. The topmost rule always wins. Therefore, the default value for **SinglePrice** will be **8888** if this rule is at the top of the grid:

```
0  T-Shirt   25   Available in all sizes
2  n/a       2.3  n/a
3  Pants     8888 Limited stock
4  Jacket    57.5 n/a
```

8.2.4 Example: Replace Empty CSV Fields

This example shows you how to create a MapForce mapping that reads data from a CSV file and writes data to another CSV file. The goal is to replace all empty fields from the source CSV file with a custom value ("n/a"). In other words, assuming that the source CSV file looks as follows:

```
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,,
```

then the desired mapping output should look as follows:

```
H,111,332.1,22537.7,n/a,Container ship,n/a,n/a,n/a
D,111,A-1579-227,10,3,400,Microtome,n/a,n/a
D,111,B-152-427,7,6,1200,Miscellaneous,n/a,n/a
H,222,978.4,7563.1,n/a,Air freight,n/a,n/a,n/a
D,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,n/a,n/a
```

You can find the mapping created in this example at the following path: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\ReplaceEmptyFields.mfd**. The source CSV file for this mapping is called **Orders.csv** and is in the same folder. The target CSV file will be generated by MapForce.

To achieve the mapping goal, we will create a single node function that replaces each encountered empty value with "n/a". As shown below, this function is defined only once but it applies to multiple descendant CSV fields.

Step 1: Add the source CSV file to the mapping

You can add the source CSV file to the mapping as follows:

1. On the **Insert** menu, click **Text File**.
2. (MapForce Enterprise Edition only) Select the option **Use simple processing for standard CSV (delimited) and/or FLF (fixed-length) fields**, and click **Continue**.
3. Click **Input File** and browse for the following file: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\Orders.csv**.

> If the check box **Treat empty fields as absent** is selected, clear it. When selected, this check box suppresses the empty values and thus will prevent the node function from working. For more information, see Setting the CSV Options.

4. Click **OK**.
5. If prompted to change the component name to "Orders", click the option you prefer (for example, **Leave component name unchanged**).

For more information about CSV components in MapForce, see CSV and Text Files.

Step 2: Add the target CSV file to the mapping

You can add the target CSV file to the mapping as follows:

1. On the **Insert** menu, click **Text File**.
2. (MapForce Enterprise Edition only) Select the option **Use simple processing for standard CSV (delimited) and/or FLF (fixed-length) fields**, and click **Continue**.
3. The target file must have the same number of fields as the source one. Therefore, click the **Append Field** button multiple times to add nine fields.
4. Click **OK**.

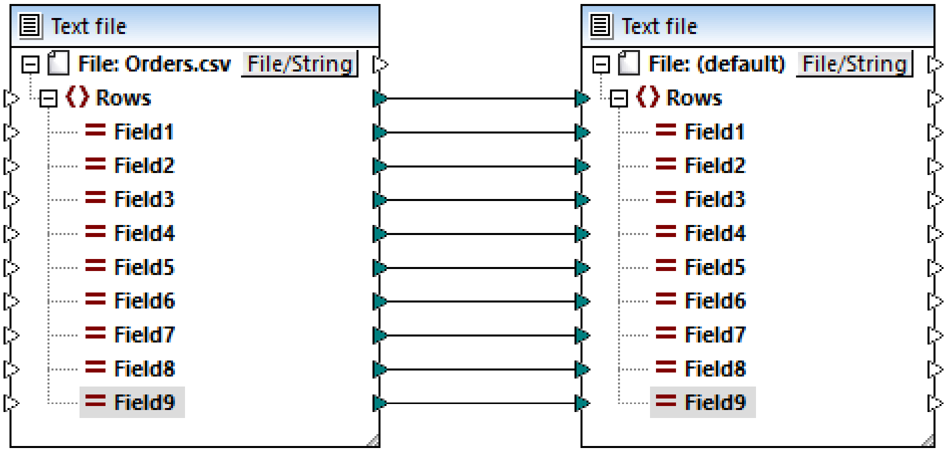
Step 3: Draw the mapping connections

At this stage, the mapping contains two components: the source CSV file and the target one. Click the output connector ▷ next to the **Rows** item on the source component and drag the cursor to the input connector ▷ of the **Rows** item in the target component. When you do this, MapForce may automatically connect all descendant items and create a so-called "Copy-All" connection, depending on your settings. This happens only if the **Auto-connect matching children** toolbar option is active. As mentioned previously, node functions are not applied to descendants of "Copy-All" connections. Therefore, the "Copy-All" connection must first be changed to a standard one. To do this, connect **Field1** from source to **Field1** from target. When prompted, click **Replace Connection**, and then click **Resolve copy-all connection**.

If the **Auto-connect matching children** option is not active, you can create connections between the source and target as follows:

1. Connect the **Rows** item in the source to the **Rows** item in the target.
2. Right-click the connection, and select **Connect Matching Children** from the context menu.
3. Clear the **Create copy-all connections** check box.
4. Click **OK**.

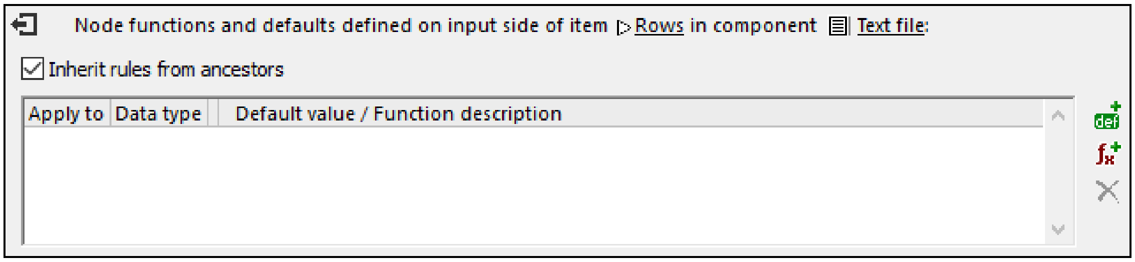
Your mapping should now look as follows:



Step 4: Create the node function

You can create a node function either immediately after data leaves the source, or immediately before it is written to the target. For the purpose of this example, let's create the node function on the input side of the target component; this essentially means "immediately before data is written to the target". For more information, see Choosing When the Function or Default Should Apply.

Right-click the `Rows` item on the target component, and select **Node Functions and Defaults | Input Node Functions and Defaults** from the context menu. An empty grid appears at the top of the **Mapping** pane.

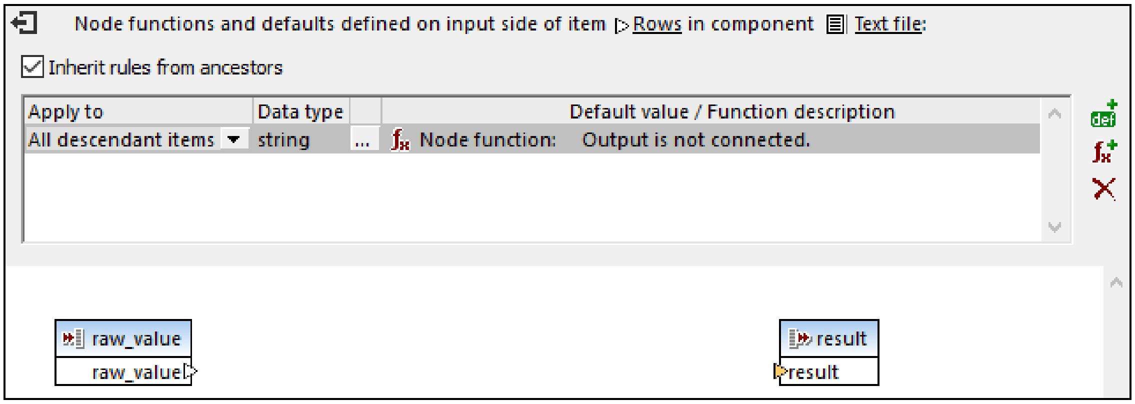


Node functions and defaults defined on input side of item ▷ Rows in component Text file:

☑ Inherit rules from ancestors

Apply to	Data type	Default value / Function description

Next, click the **Add function** button to the right of the grid. The mapping now displays the function's input ("raw_value") and output ("result").



Node functions and defaults defined on input side of item ▷ Rows in component Text file:

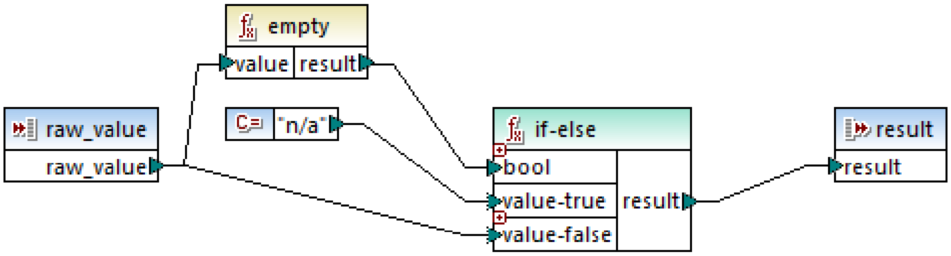
☑ Inherit rules from ancestors

Apply to	Data type		Default value / Function description
All descendant items	string	...	Node function: Output is not connected.

As mention before, the function's goal is to convert any empty value into the string "n/a". To achieve this, let's add the following additional components to the mapping:

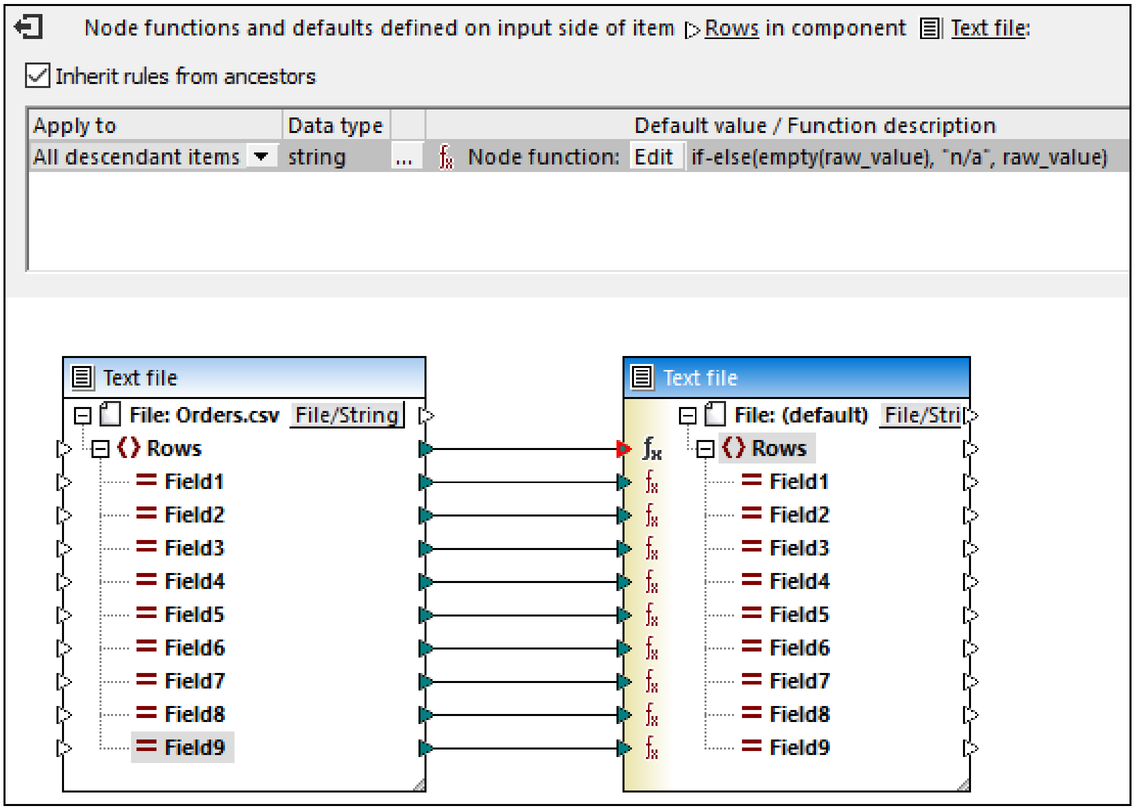
- The MapForce built-in function **`empty`**. This function returns true if the value supplied as argument is empty; false otherwise. You can drag the function into the mapping from the Libraries window, or just double-click the mapping and type "empty", see also Add a Built-in Function to the Mapping.
- A text value "n/a". To add this value, double-click an empty area on the mapping and enter "n/a" surrounded by double quotes, see also Add a Constant to the Mapping.
- An **If-Else Condition**. To add it to the mapping, click the **If-Else Condition** ( ) toolbar button. For more information about such components, see Example: Returning a Value Conditionally.

With the help of these components, design the function as follows:



The design illustrated above works as follows: first, any input value from the outer mapping enters the function through the **raw_value** input. The raw value is then supplied as input to the **`empty`** function. Then, the **If-Else Component** evaluates the Boolean result (**true** or **false**) returned by the **`empty`** function. When the result is **true**, the constant "n/a" becomes the function's result. When the result is **false**, the function's raw input value becomes the function's result. The function's **result** (which is either "n/a" or **raw_value**) is then returned to the outer mapping.

Click **Exit** (or press **Escape**) to exit the function's editing area.



Node functions and defaults defined on input side of item Rows in component Text file:

☑ Inherit rules from ancestors

Apply to	Data type		Default value / Function description
All descendant items	string	...	Node function: Edit if-else(empty(raw_value), "n/a", raw_value)

In the mapping illustrated above, note the following:

- The text at the top of the window clearly indicates where the function is defined. This is particularly useful in situations where multiple node functions are defined for the same component.
- The **Apply to** option in the grid is set to **All descendant items**. In this example, this is the intended behavior. That is, all descendant items of **Rows** must be affected if they qualify. As you can see on the mapping, the left (input) side of the target component displays multiple icons, even though the function was defined only once, for the parent item.
- The **Data Type** option is set to "string". In this example, since we are dealing with text data, this is the intended behavior. It is also the default behavior.
- The **Edit** button lets you go back to the function's definition and change it if necessary. If you don't see this button, click the icon first.

Step 5: Run the mapping

To preview the mapping result directly in MapForce, click the **Output** tab. If any validation errors are encountered, these are displayed in the **Messages** window, see Validating Mappings. Upon success, the resulting CSV is displayed in the **Output** pane.

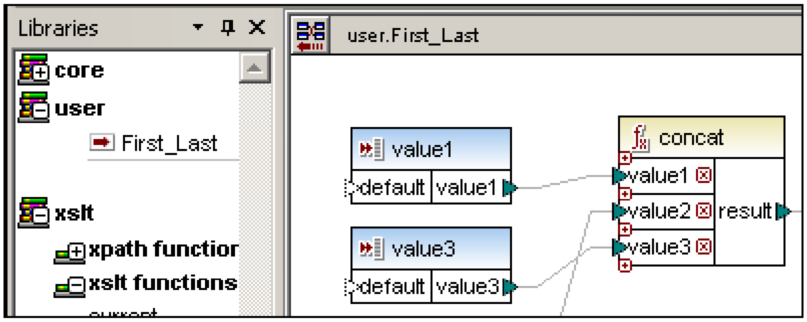
You can also execute such mappings on a server machine, with MapForce Server Advanced Edition, in one of the following ways:

- If you have MapForce Server Advanced Edition standalone license, compile the mapping to an execution file and then copy it to the target machine, see Compiling Mappings to MapForce Server Execution Files.
- If you have licensed both FlowForce Server and MapForce Server Advanced Edition, you can deploy the mapping directly to FlowForce Server and configure it to run as a scheduled or on-demand job, see Deploying Mappings to FlowForce Server.

8.3 User-Defined Functions

MapForce allows you to create user-defined functions visually, in the same way as in the main mapping window.

These functions are then available as function entries in the Libraries window (for example, "First_Last" in the image below), and are used in the same way as the currently existing functions. This allows you to organize your mapping into separate building blocks which are reusable across different mappings.



User-defined functions are stored in the ***.mfd** file, along with the main mapping.

A user-defined function uses **input** and **output components** to pass information from the main mapping (or another user-defined function) to the user-defined function and back.

User-defined functions can contain "local" source components (i.e that are within the user-defined function itself) such as XML schemas or databases, which are useful when implementing lookup functions.

User-defined functions can contain any number of input and outputs where any of these can be in the form of: simple values, XML nodes, or databases.

User-defined functions are useful when:

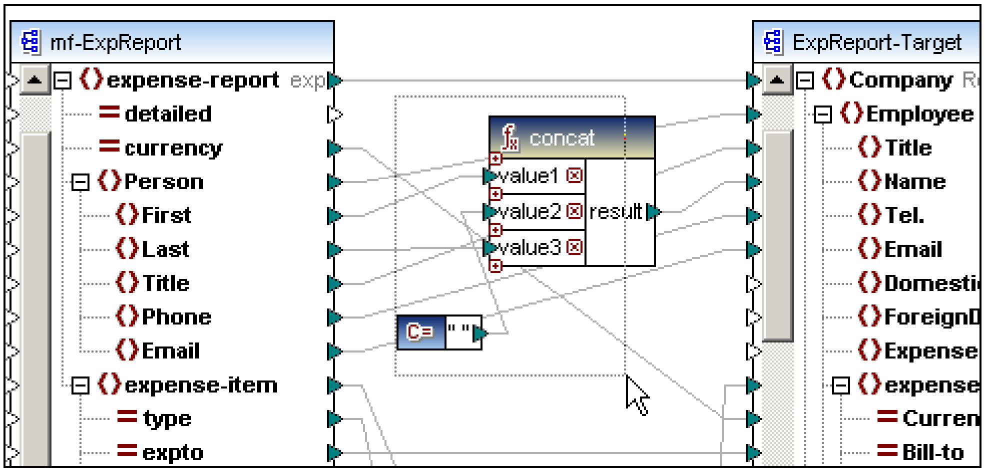
- combining multiple processing functions into a single component, e.g. for formatting a specific field or looking up a value
- reusing these components any number of times
- importing user-defined functions into other mappings (by loading the mapping file as a library)
- using inline functions to break down a complex mapping into smaller parts that can be edited individually
- mapping **recursive schemas** by creating recursive user-defined functions

User-defined functions can be either built from scratch, or from functions already available in the mapping tab.

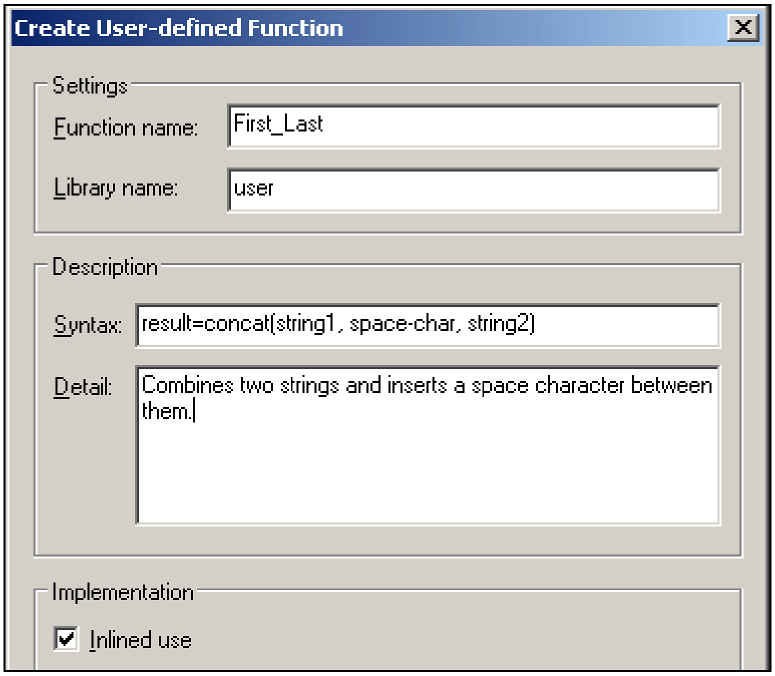
This example uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples\Tutorial\ folder.

Creating user-defined function from existing components

1. Drag to select both the "concat" and the constant components (you can also hold down the CTRL key and click the functions individually).



2. Select the menu option **Function | Create User-Defined Function from Selection**.
3. Enter the name of the new user-defined function (First_Last).
   Note: valid characters are: alphanumeric, a-z, A-Z, 0-9 as well as underscore "_", hyphen/ dash "-" and colon ":".
4. Use the Syntax and Detail fields to add extra information on the new function, and click OK to confirm. The text you enter will appear as a tooltip when the cursor is placed over the function.
   The library name "user" is supplied as a default, you can of course define your own library name in this field.



Create User-defined Function

Settings

Function name: First_Last

Library name: user

Description

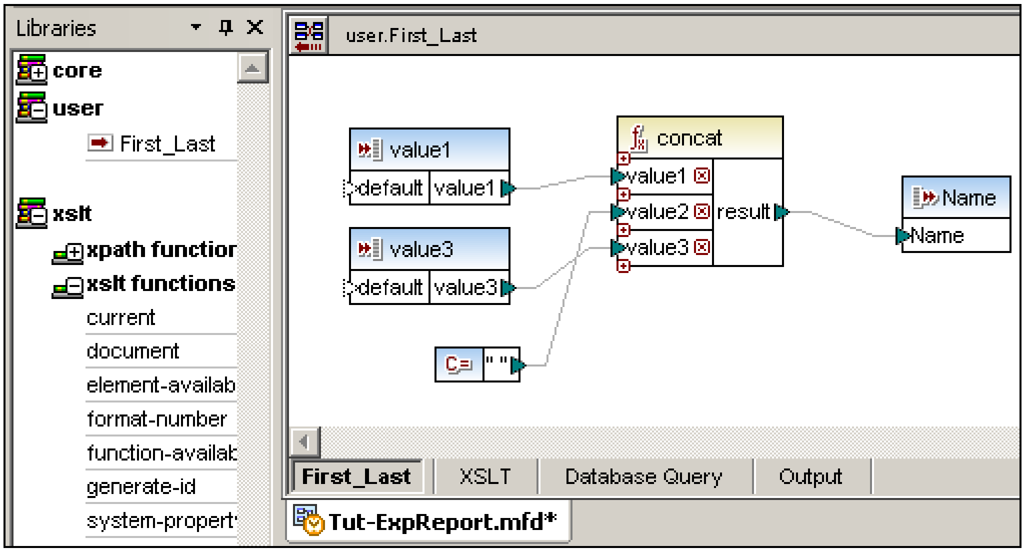
Syntax: result=concat(string1, space-char, string2)

Detail: Combines two strings and inserts a space character between them.

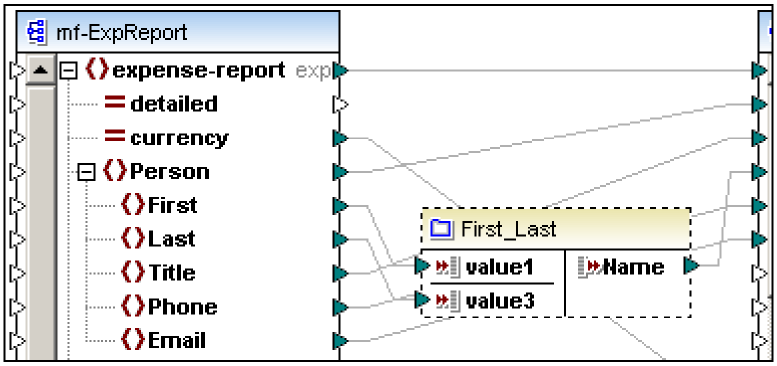
Implementation

☑ Inlined use

The individual elements that make up the function group appear in a tab with the function name. The new library "user" appears in the Libraries pane with the function name "First_Last" below it.



Click the Home button to return to the main mapping window. The components have now been combined into a single function component called First_Last. The input and output parameters have been automatically connected.



Note that inline user-defined functions are displayed with a dashed outline. See Inline user-defined functions for more information.

Dragging the function name from the Libraries pane and dropping it in the mapping window, allows you to use it anywhere in the current mapping. To use it in a different mapping, please see Reusing user-defined functions

Opening user-defined functions

To open a user-defined function, do one of the following:

- Double-click the title bar of a user-defined function component
- Double-click the specific user-defined function in the Libraries window.

This displays the individual components inside the function in a tab of that name. Click the Home button to return to the main mapping. Double-clicking a user-defined function of a different *.mfd file (in the main mapping window) opens that .mfd file in a new tab.

Navigating user-defined functions

When navigating the various tabs (or user-defined function tabs) in MapForce, a history is automatically generated which allows you to travel forward or backward through the various tabs, by clicking the back/forward icons. The history is session-wide, allowing you to traverse multiple MFD files.

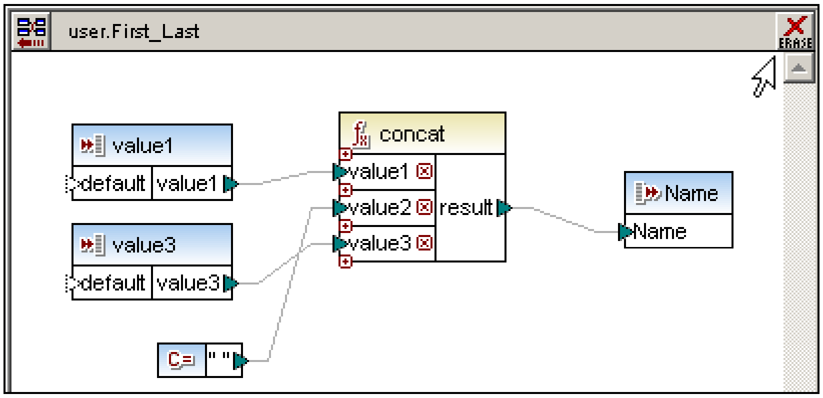
The Home button returns you to the main mapping tab from within the user-defined function.

The Back button takes you back through your history

The Forward button moves you forward through your history

Deleting user-defined functions from a library

1. Double-click the specific user-defined function in the Libraries window.
2. Click the **Erase** button in the top right of the title bar.



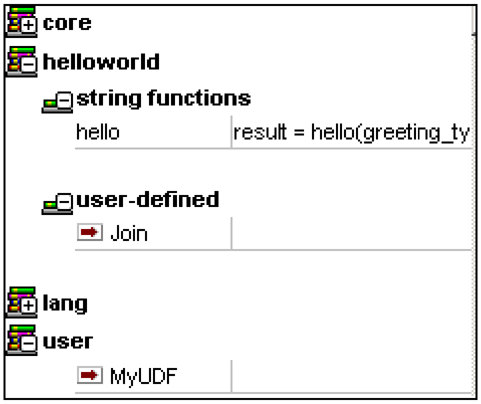
Reusing (importing) user-defined functions

User-defined functions defined in one mapping can be imported into any other mapping as follows:

1. Click the **Add/Remove Libraries** button at the base of the Libraries window.
2. Click **Add** and select the *.mfd file that contains the user-defined function(s) you want to import. The user-defined function now appears in the Libraries window. The library name is "user" if you created the user-defined function with the default library name. Otherwise, look for the library name that you specified when creating the user-defined function.
2. Drag the imported function from the Libraries window into the mapping.

If the same library name is specified across multiple *.mfd files or custom libraries (see Importing Custom Java and .NET Libraries), functions from all available sources appear under the same library name in the Libraries window. However, only the functions in the currently active document can be edited by double-clicking.

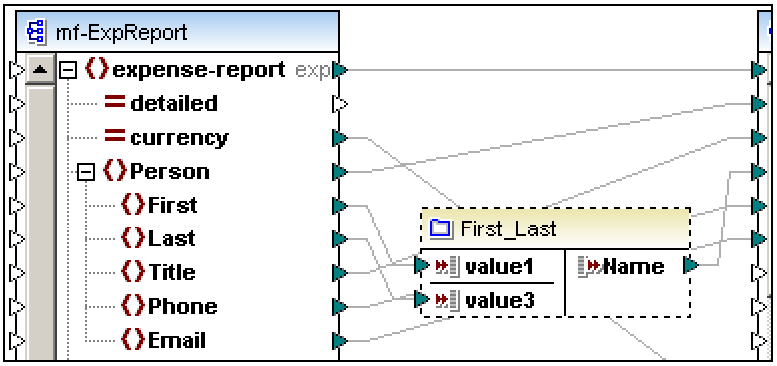
Consider the following example:



- The function "hello" in the "helloworld" library was imported from a custom library
- The function "Join" in the "helloworld" library is a user-defined function defined in the current *.mfd file
- The function "MyUDF" in the "user" library is also a user-defined function defined in the current *.mfd file

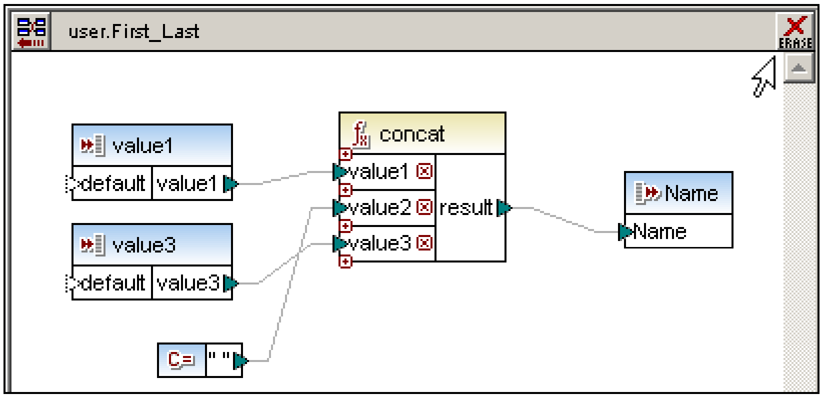
Note that possible changes in imported functions are applied to importing mappings when saving the library ***.mfd** file.

Parameter order in user-defined functions



The parameter order within user-defined functions can be directly influenced:

- Input and output parameters are sorted by their position from top to bottom (from the top left corner of the parameter component).
- 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.



Notes:

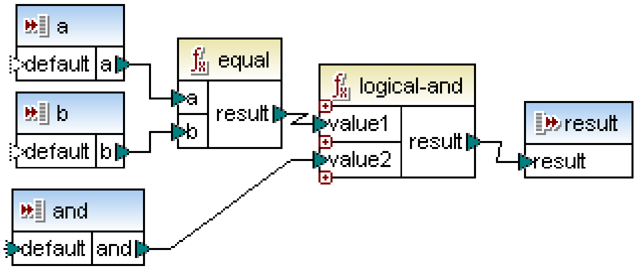
- The Component positioning and resizing actions are undoable.
- Newly added input or output components are created below the last input or output component.
- Complex and simple parameters can be mixed. The parameter order is derived from the component positions.

8.3.1 Function parameters

Function **parameters** are represented inside a user-defined function by **input** and **output components.**

Input components/parameters: **a**, **b**, **and**

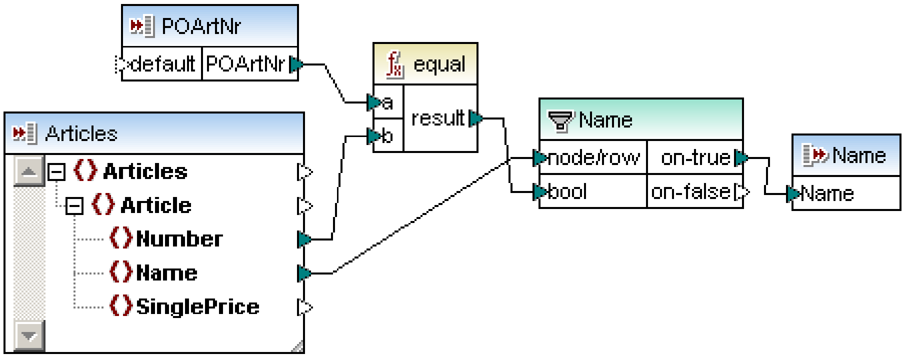
Output component/parameter: **result**



Input parameters are used to pass data from the main mapping into the user-defined function, while output parameters are used to return data back to the main mapping. Note that user-defined functions can also be called from other user-defined functions.

Simple and complex parameters
The input and output parameters of user-defined functions can be of various types:

- Simple values, e.g. string or integer
- Complex node trees, e.g. an XML element with attributes and child nodes



Input parameter **POArtNr** is a simple value of datatype "string"

Input parameter **Articles** is a **complex** XML document node tree

Output parameter **Name** is a simple value of type string

Note:

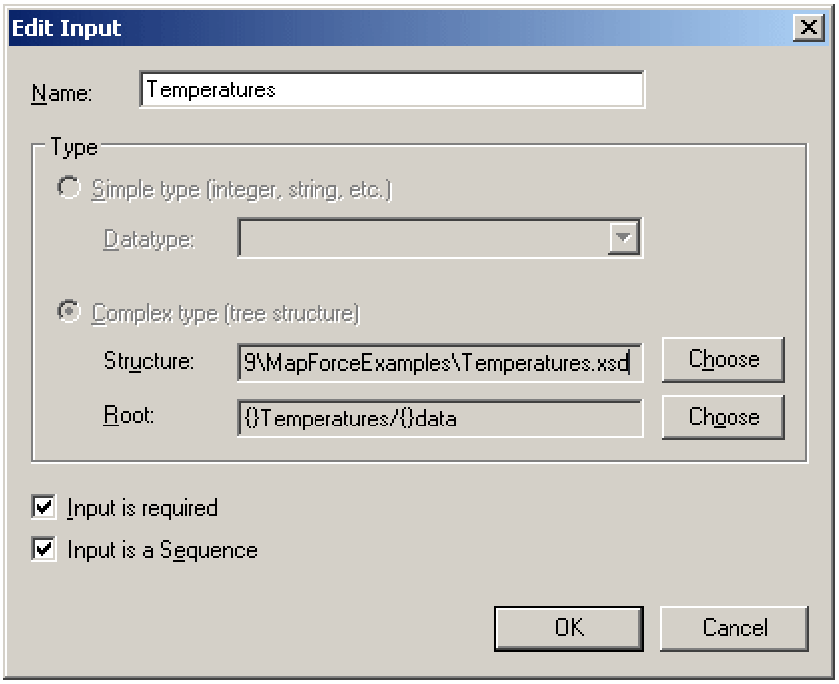
The user-defined functions shown above are all available in the **PersonListByBranchOffice.mfd** file available in the ...\MapForceExamples folder.

Sequences

Sequences are data consisting of a range, or sequence, of values. Simple and complex user-defined **parameters** (input/output) can be defined as sequences in the component properties dialog box.

Aggregate functions, e.g. min, max, avg, etc., can use this type of input to supply a single specific value from the input sequence.

When the "**Input is a Sequence**" check box is active, the component handles the input as a sequence. When inactive, input is handled as a single value.


Edit Input

Name: Temperatures

Type

○ Simple type (integer, string, etc.)

Datatype:

● Complex type (tree structure)

Structure: 9\MapForceExamples\Temperatures.xsd

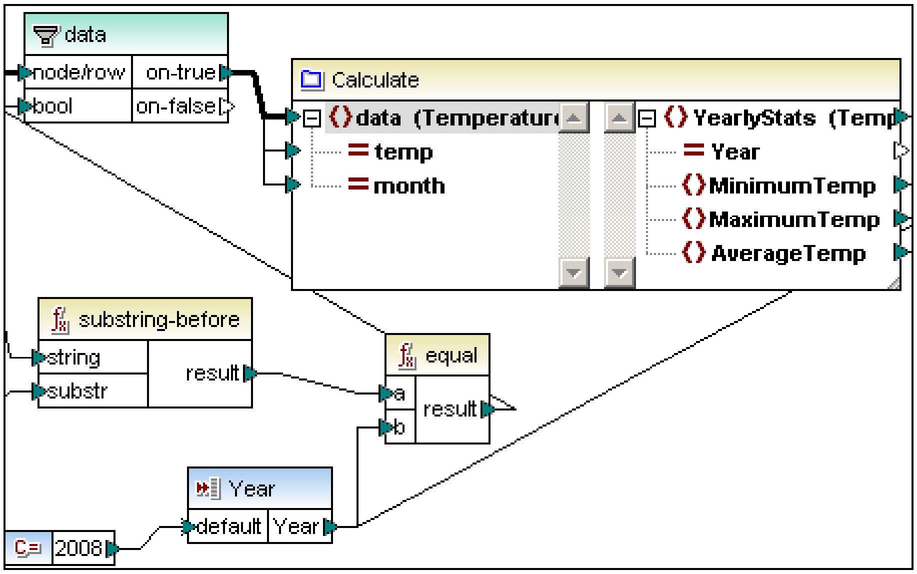
Root: {}Temperatures/{}data

☑ Input is required

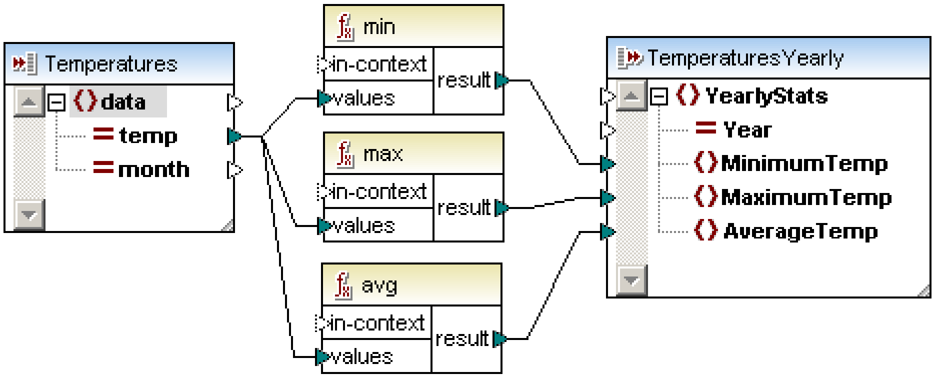
☑ Input is a Sequence

This type of input data, sequence or non-sequence, determines **how often** the function is called.

- When connected to a **sequence** parameter the user-defined function is called only **once** and the complete sequence is passed into the user-defined function.



The screenshot shows the user-defined function "Calculate" of the "**InputIsSequence.mfd**" mapping in the ...\MapForceExamples folder. The **Temperatures** input component (shown below) is defined as a sequence.



- When connected to a **non-sequence** parameter, the user-defined function is called **once** for **each** single item in the sequence.

Please note:

The sequence setting of input/output parameters is ignored when the user-defined function is of type inline.

> Connecting an empty sequence to a non-sequence parameter has the result that 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, either use the substitute-missing function before the function input to ensure that the sequence is never empty, or set the parameter to sequence, and add handling for the empty sequence inside the function.

When a function passes a sequence of multiple values to its output component, and the output component is not set to sequence, only the first result is used when the function is called.

8.3.2 Inline and regular user-defined functions

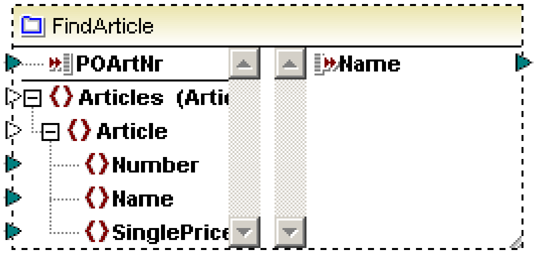
Inline functions differ fundamentally from regular functions, in the way that they are implemented when code is generated.

- The code for **inline** type functions is inserted at **all locations** where the user-defined functions are called/used
- The code of a **regular** function is implemented as a function call.

  Inline functions thus behave as if they had been replaced by their implementation. That makes them ideal for breaking down a complex mapping into smaller encapsulated parts.

Please note:
using **inline** functions can significantly increase the amount of generated program code! The user-defined function code is actually inserted at all locations where the function is called/used, and thus increases the code size substantially - as opposed to using a regular function.

INLINE user-defined functions are shown with a **dashed** outline:



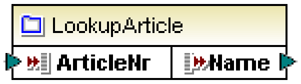
Inline user-defined functions **support**:

- **Multiple output components** within a function

do not support:

- The setting of a priority context on a parameter
- Recursive calls to an inline user-defined function

REGULAR user-defined functions i.e. non-inline functions are shown with a **solid** outline:



Regular (non-inline) user-defined functions **support**:

- Only a **single output component** within a function

- **Recursive** calls (where the exit condition must be supplied, e.g. use an If-Else condition where one branch, or value, exits the recursion)
- Setting a priority context on a parameter

Please note:

Although regular functions **do not** support multiple output components, they **can be created** in this type of function. However, an error message appears when you try to generate code, or preview the result of the mapping.

If you are not using recursion in your function, you can change the type of the function to "inline".

do not support:

- Direct connection of filters to simple non-sequence **input** components
- Sequence or aggregate functions on simple input components (like exists, substitute-missing, sum, group-by, ...)

Code generation

The implementation of a regular user-defined function is generated only once as a callable XSLT template or function. Each user-defined function component generates code for a **function call**, where inputs are passed as parameters, and the output is the function (component) return value.

At runtime, all the input parameter values are evaluated first, then the function is called for each occurrence of the input data. See Function parameters for details about this process.

To change the user-defined function "type":

1. Double click the user-defined function to see its constituent components.
2. Select the menu option **Function | Function settings** and click the "Inlined use" check box.

User-defined functions and Copy-all connections

When creating Copy-all connections between a schema and a complex user-defined function parameter, the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components - defining" for an example.

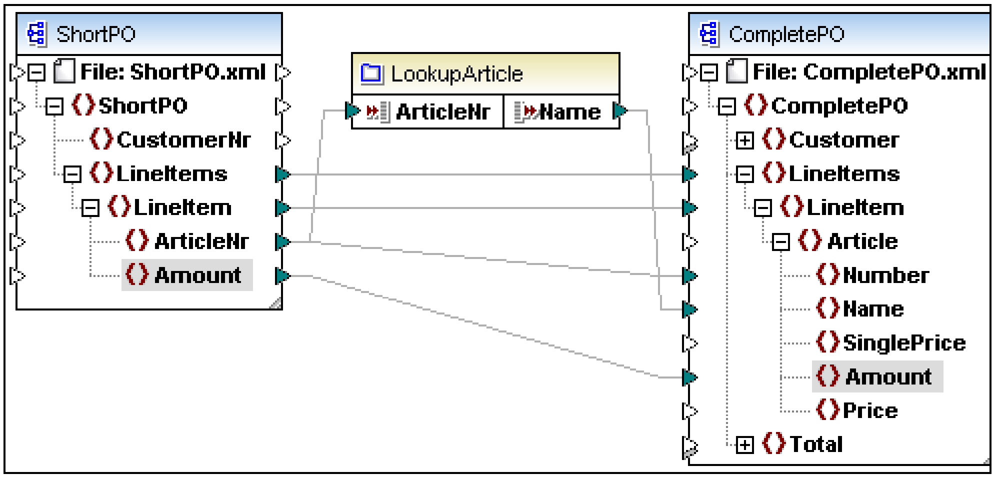
8.3.3 Creating a simple look-up function

This example is provided as the **lookup-standard.mfd** file available in the **...\MapForceExamples** folder.

Aim:

To create a generic look-up function that:

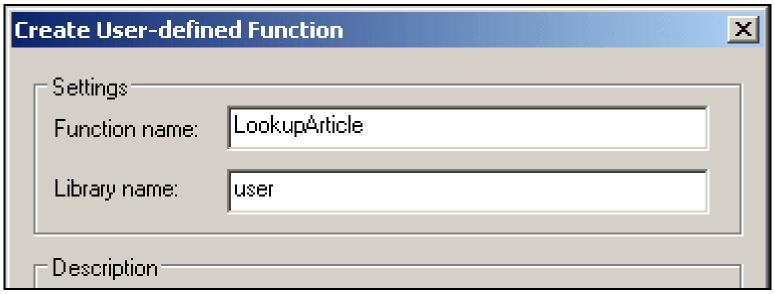
- supplies Articles/Number data from the Articles XML file, to be compared to Article numbers of a different XML file, ShortPO in this case.



- Insert the ShortPO.xsd and assign ShortPO.xml as the source XML file.
- Insert the CompletePO.xsd schema file, and select CompletePO as the root element.
- Insert a new user-defined function using the method described below.

To create a user-defined function:

1. Select the menu option **Function | Create User-defined function**.
2. Enter the name of the function e.g. LookupArticle.


Create User-defined Function

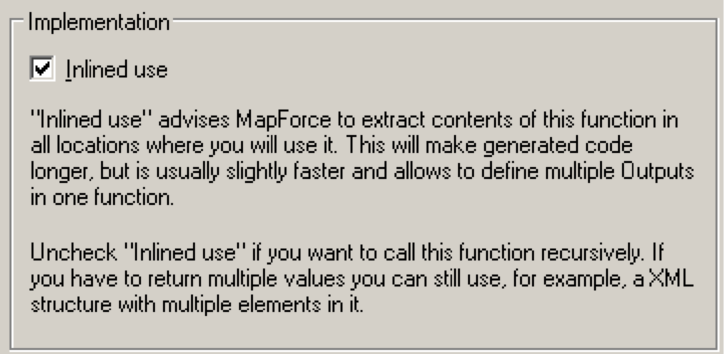
Settings

Function name: LookupArticle

Library name: user

Description

3. Uncheck the "Inlined use" check box and click OK to confirm

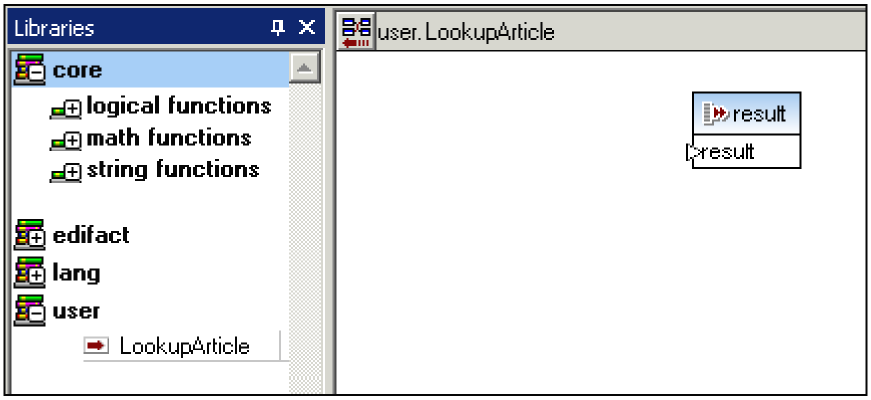

Implementation

☑ Inlined use

"Inlined use" advises MapForce to extract contents of this function in all locations where you will use it. This will make generated code longer, but is usually slightly faster and allows to define multiple Outputs in one function.

Uncheck "Inlined use" if you want to call this function recursively. If you have to return multiple values you can still use, for example, a XML structure with multiple elements in it.

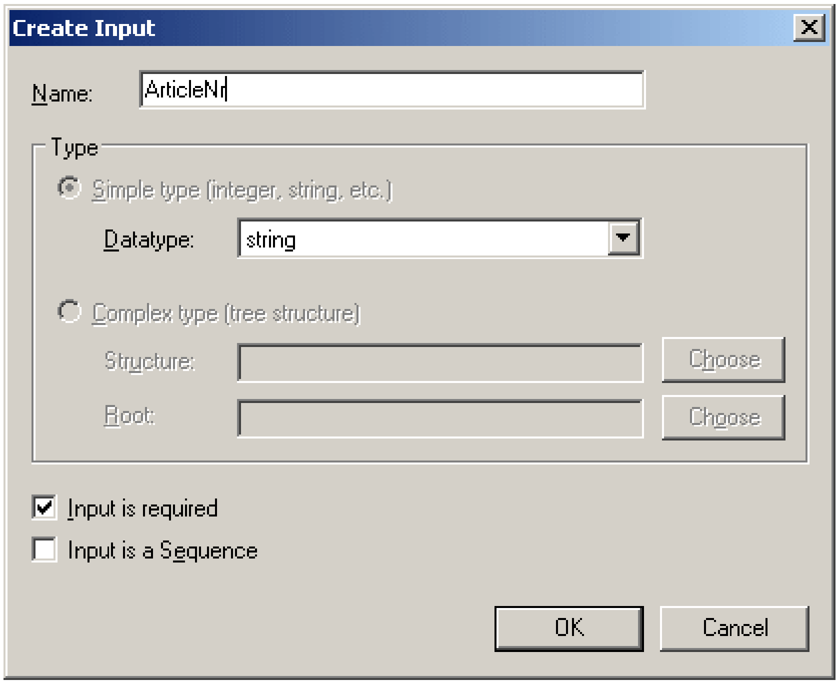
A tab only containing only one item, an output function currently called "result", is displayed.



This is the working area used to define the user-defined function.

A new library has been created in the Libraries pane with the name "user" and the function name "LookupArticle".

3. Click the **Insert Schema/XML file** icon to insert the **Articles** schema and select the XML file of the same name to act as the data source.
4. Click the **Insert input component** icon to insert an input component.
5. Enter the name of the input parameter, ArticleNr in this case, and click OK.



Create Input

Name: ArticleNr

Type

● Simple type (integer, string, etc.)

Datatype: string

○ Complex type (tree structure)

Structure:

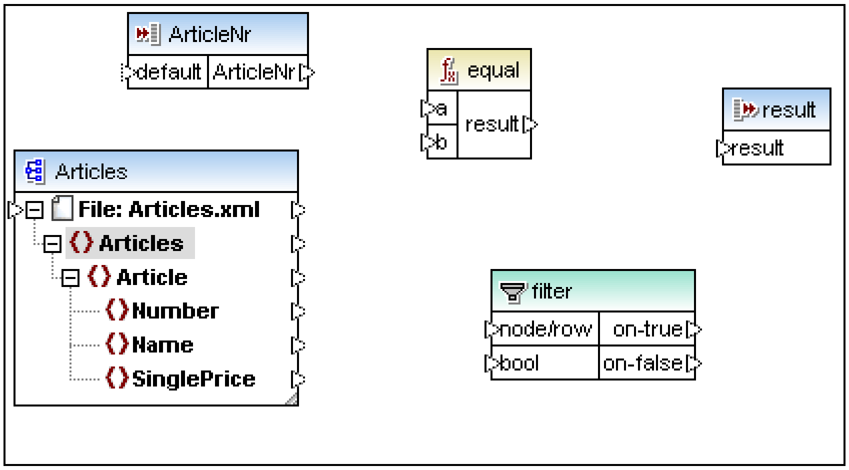
Root:

☑ Input is required

☐ Input is a Sequence

This component acts as a data input to the user-defined function and supplies the input icon of the user-defined function.

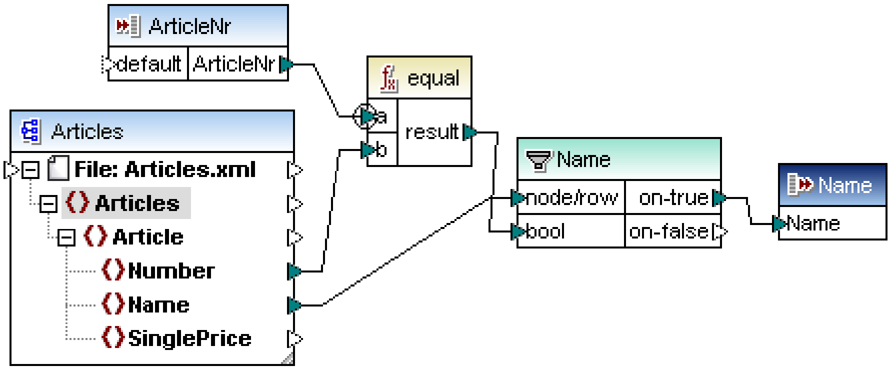
6. Insert an "**equal**" component by dragging it from the core library/logical functions group.
7. Insert a **filter** component by clicking the Insert Filter icon in the toolbar.



Use the diagram below as an aid to creating the mappings in the user-defined function, please take note of the following:

8 Right click the **a** parameter and select **Priority context** from the pop up menu.

9. **Double click** the output function and enter the name of the output parameter, in this case "**Name**".



This ends the definition of the user-defined function.

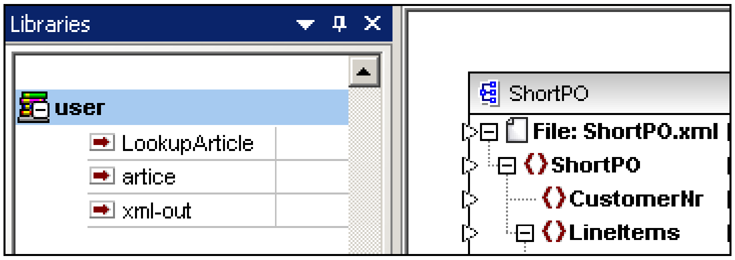
Please note:

Double clicking the **input** and **output** functions opens a dialog box in which you can change the name and datatype of the input parameter, as well as define if the function is to have an input icon (Input is required) and additionally if it should be defined as a sequence.

This user-defined function:

- has one **input** function, ArticleNr, which will receive data from the ShortPO XML file.
- **compares** the ShortPO ArticleNr, with the Article/Number from the **Articles** XML instance file, inserted into the user-defined function for this purpose.
- uses a **filter** component to forward the Article/Name records to the output component, if the comparison returns true.
- has one output function, Name, which will forward the Article Name records to the CompletePO XML file.

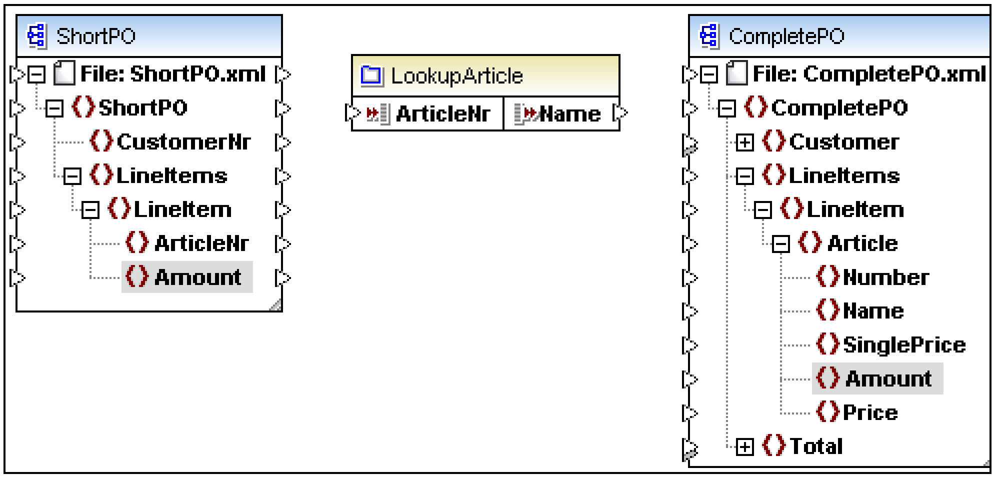
10. Click the Home icon ⌂ to return to the main mapping.
The LookupArticle user-defined function, is now available under the **user** library.



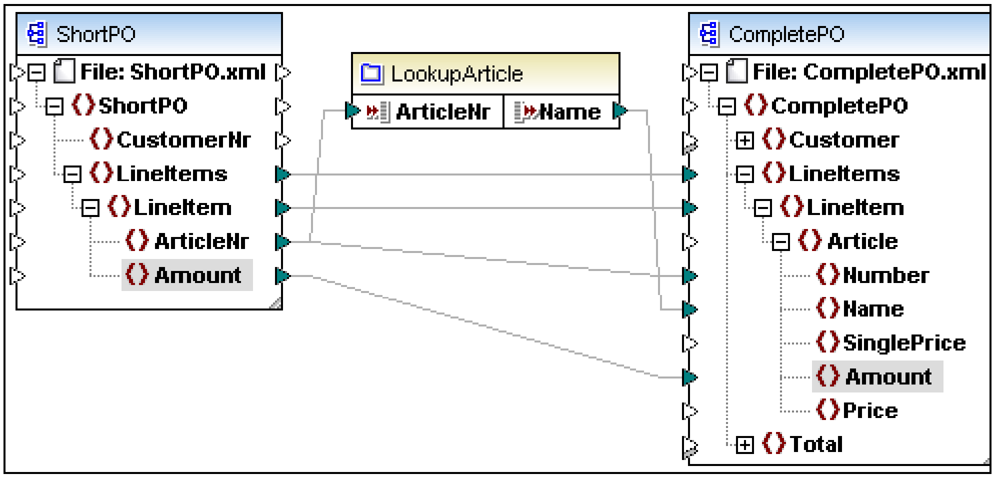
11. Drag the LookupArticle function into the Mapping window.

The user-defined function is displayed:

- with its name "LookupArticle" in the title/function bar,
- with named input and output icons.



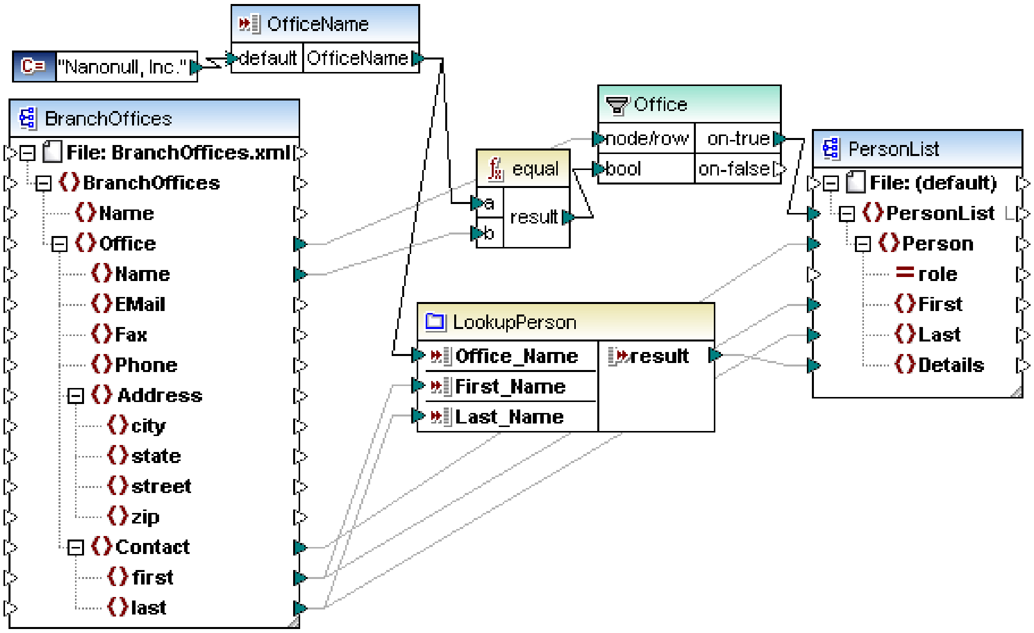
10. Create the connections displayed in the graphic below and click the Output tab to see the result of the mapping.



8.3.4 User-defined function - example

The **PersonListByBranchOffice.mfd** file available in the **<Documents>\Altova\MapForce2018\MapForceExamples\** folder illustrates the following features:

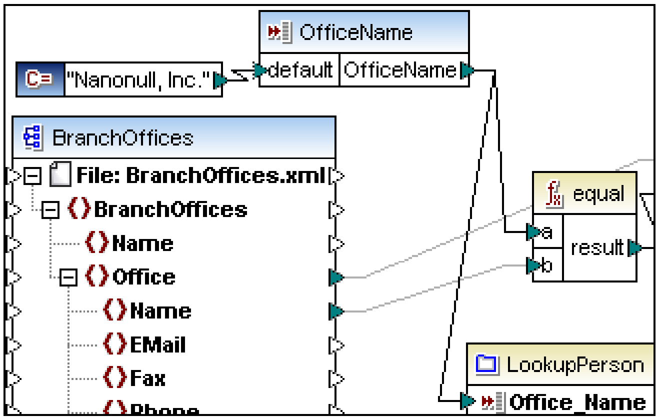
- Nested User-defined functions e.g. **LookupPerson**
- Look-up functions that generate a string output e.g. **LookupPerson**
- **Optional** input-parameters which can also supply a **default** value e.g. the **EqualAnd** component (contained in the LookupPerson component)
- **Configurable** input parameters, which can also double as a command line parameter(s) when executing the generated mapping code!



Configurable input parameters

The input component (OfficeName) receives data supplied when a mapping is executed. This is possible in two ways:

- as a **command line** parameter when executing the generated code, e.g. Mapping.exe /OfficeName "Nanonull Partners, Inc."
- as a **preview** value when using the Built-in execution engine to preview the data in the Output window.

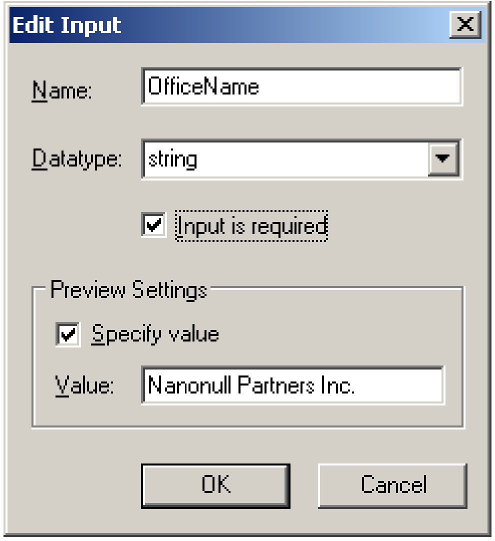


To define the Input value:

1. Double click the input component and enter a different value in the "Value" text box of the Preview Mode group e.g. "Nanonull Partners, Inc.", and click OK to confirm.
2. Click the Output tab to see the effect.
   A different set of persons are now displayed.

   Please note that the data entered in this dialog box is only used in "**preview**" mode i.e. when clicking the Output tab. If a value is not entered, or the check box is deactivated, then the data mapped to the input icon "default" is used.

   Please see Input Components for more information.



Edit Input

Name: OfficeName

Datatype: string

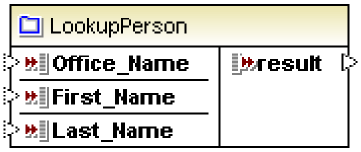
☑ Input is required

Preview Settings

☑ Specify value

Value: Nanonull Partners Inc.

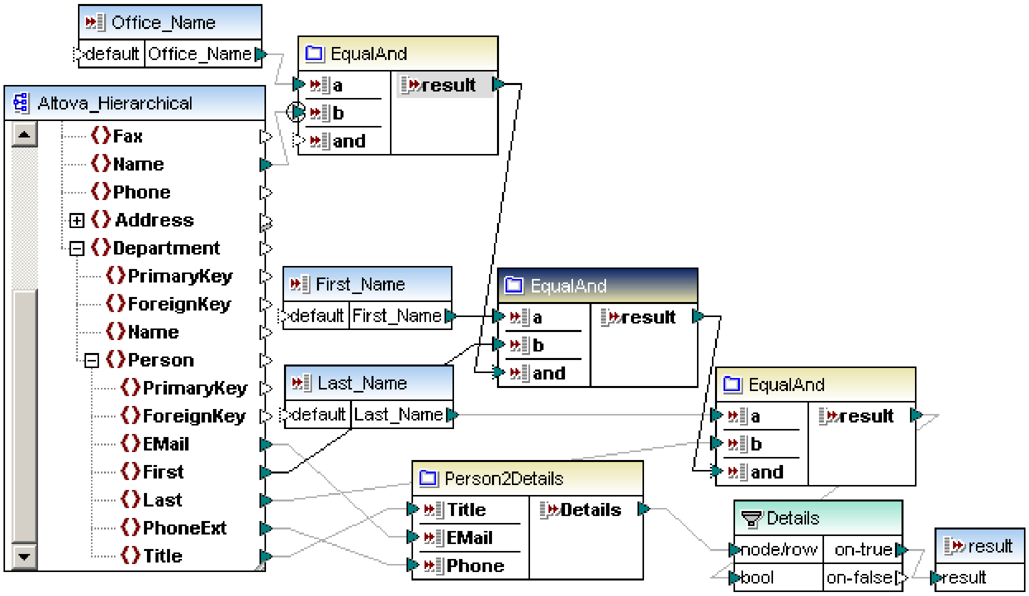
LookupPerson component



Double clicking this user-defined component displays its constituent components shown below. What this component does is:

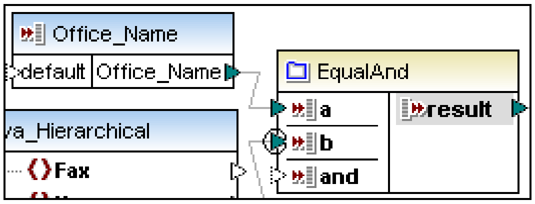
- **Compares** the Office, First, and Last names of BranchOffices.xml, with the same fields of the Altova_Hierarchical.xml file, using the **input** components and the **EqualAnd** user-defined components.
- **Combines** the Email, PhoneExt and Title items using the **Person2Details** user-defined function
- **Passes on** the combined person data to the **output** component if the previous EqualAnd comparisons are all true (i.e. supplied "true" to the filter component).

A user-defined function always outputs a value, which may even be an empty string! This would be the case if the filter component bool value is false. Only an empty string would be output instead of data supplied by the Person2Details component.



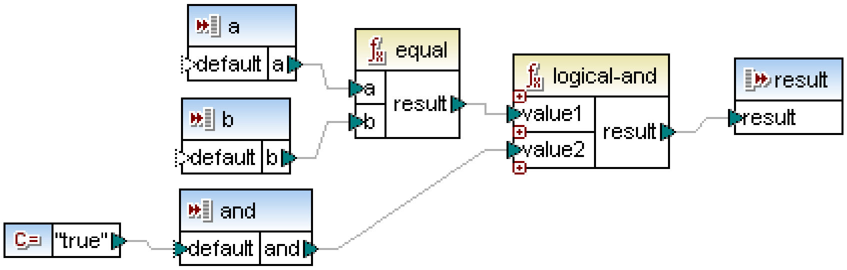
- The three **input** components, Office_Name, First_Name, Last_Name, receive their data from the BranchOffices.xml file.
- The **EqualAnd** component compares two values and provides an **optional** comparison value, as well as a default value.
- Person2Details combines three person fields and passes on the result to the filter component.

EqualAnd component



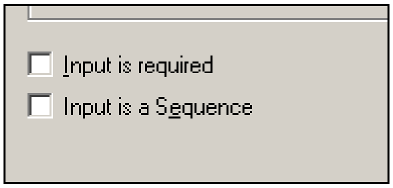
Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- Compare two input parameters **a** and **b**, and pass the result on to the logical-and component. Note that the **b** parameter has been defined as the **priority context** (right click the icon to do so). This ensures that the person data of the specific office, supplied by the input parameter **a**, is processed first.
- **Logical-and** the result of the first comparison, with an **optional** input parameter, "and".
- Pass on the boolean value of this comparison to the output parameter.



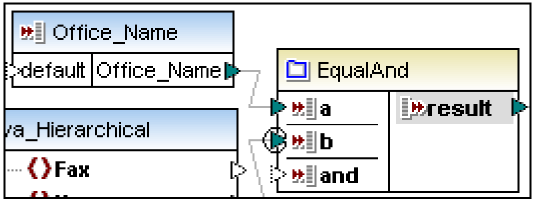
Optional parameters

Double clicking the "and" parameter, of the EqualAnd user-defined function shown above, allows you to make parameters optional, by unchecking the "Input is required" check box.



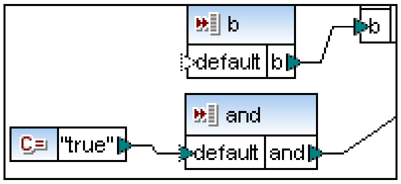
☐ Input is required

☐ Input is a Sequence



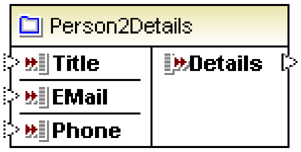
If "Input is required" is **unchecked**, then:

- A mapping connector is not required for the input icon of this user-defined function, e.g. the **and** parameter of the first EqualAnd function, does not have an input connector. The input icon has a dashed outline to show this visually.
- A **default** value can be supplied by connecting a component, within the user-defined function e.g. using a constant component containing the value "true".



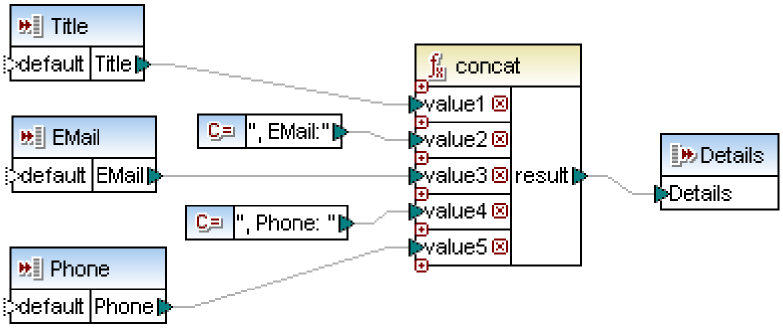
- A mapping from another item, mapped to the optional Input, takes precedence over the default value. E.g. the "and" parameter of second EqualAnd function, receives input data from the "result" parameter of the first EqualAnd user-defined function.

Person2Details component



Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- Concatenate three inputs and pass on the result string to the output parameter.
- Double clicking an output parameter allows you to change the parameter name (Details), and select the datatype (String).



8.3.5 Complex user-defined function - XML node as input

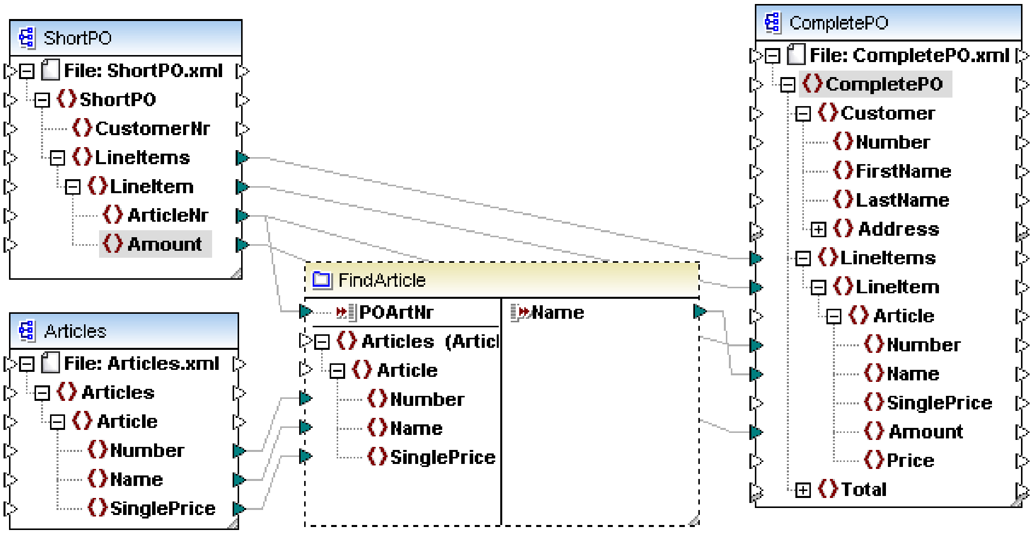
This example is provided as the **lookup-udf-in.mfd** file available in the ...\MapForceExamples folder. This section illustrates how to define an inline user-defined function that contains a complex input component.

Note that the user-defined function "FindArticle" consists of two halves.

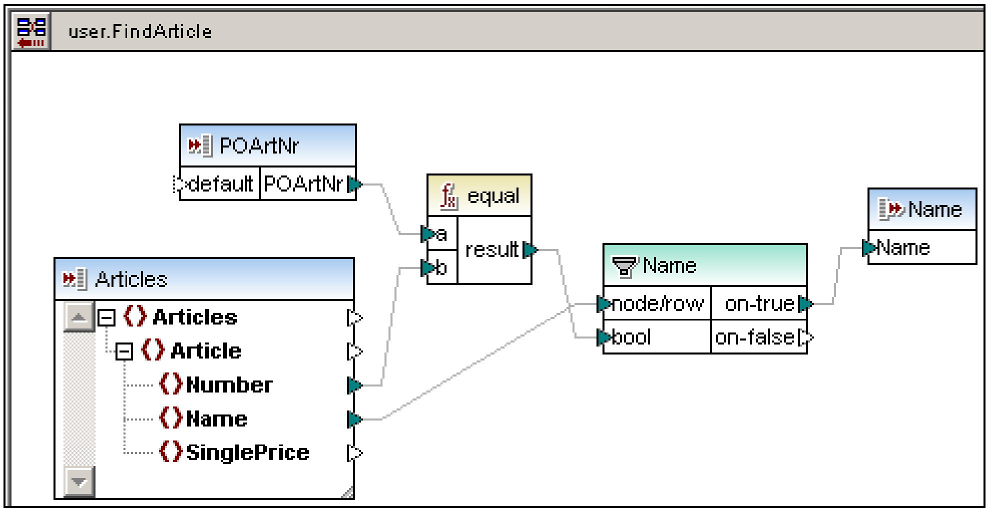
The left half contains the input parameters:

- a simple input parameter **POArtNr**
- a complex input component **Articles,** with mappings directly to its XML child nodes

The right half contains a simple output parameter called "**Name"**.



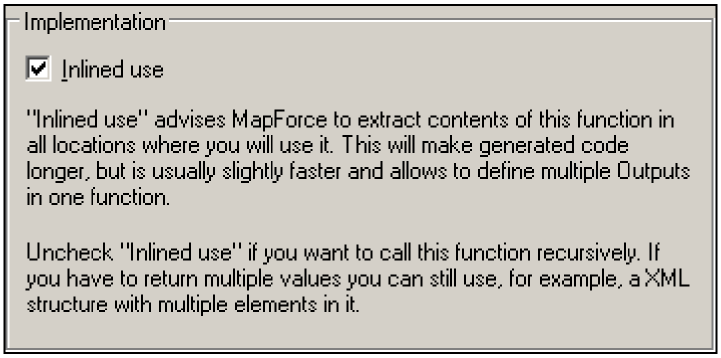
The screenshot below shows the constituent components of the user-defined function, the two input components to the left and the output component to the right.



8.3.5.1 Defining Complex Input Components

Follow these steps to create a function that takes an XML structure as input parameter:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** and click OK to confirm. Note that the **Inlined use** check box is automatically selected.

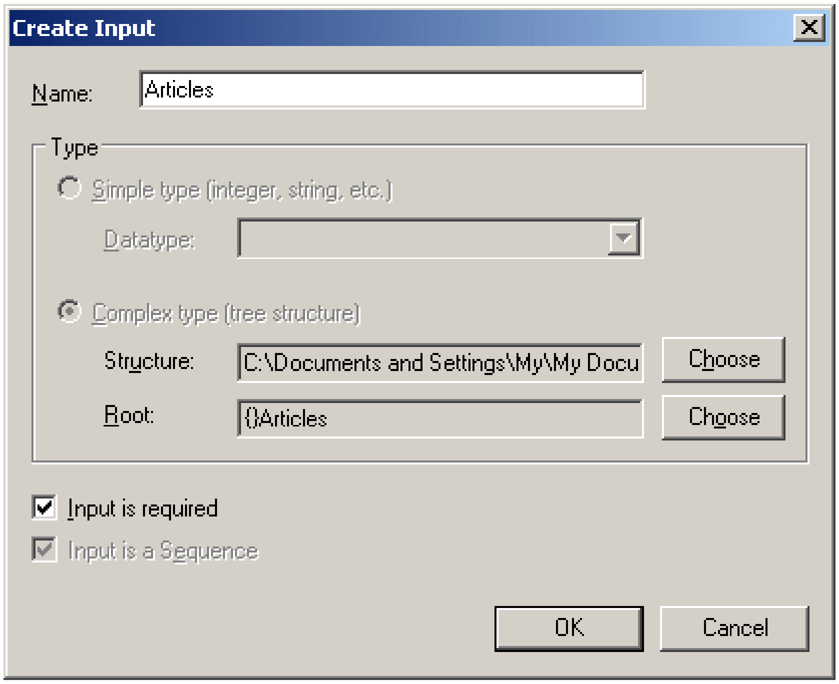

Implementation

☑ Inlined use

"Inlined use" advises MapForce to extract contents of this function in all locations where you will use it. This will make generated code longer, but is usually slightly faster and allows to define multiple Outputs in one function.

Uncheck "Inlined use" if you want to call this function recursively. If you have to return multiple values you can still use, for example, a XML structure with multiple elements in it.

2. Click the **Insert input component** icon in the icon bar.
3. Enter the name of the input component into the Name field.



Name: Articles

○ Simple type (integer, string, etc.)

Datatype:

● Complex type (tree structure)

Structure: C:\Documents and Settings\My\My Docu

Root: {}Articles

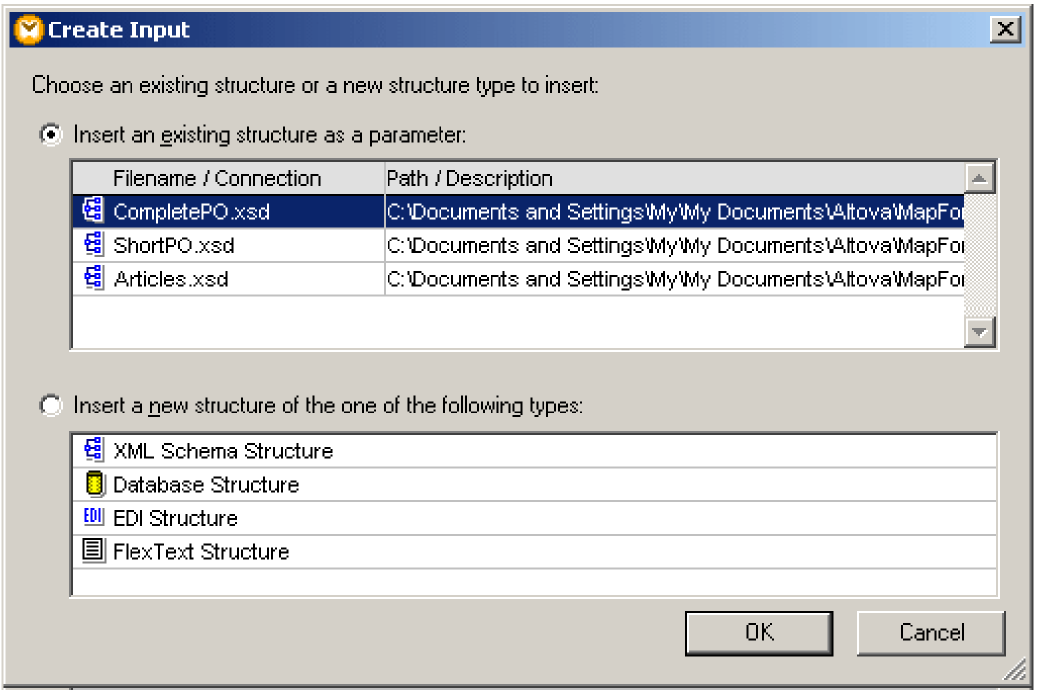
☑ Input is required

☑ Input is a Sequence

4. Click the **Complex type (tree structure)** radio button, then click the "**Choose**" button next to the Structure field. This opens another dialog box.

   The top list box displays the **existing** components in the mapping (three schemas if you opened the example mapping). Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML schema, database file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, file.



Create Input

Choose an existing structure or a new structure type to insert:

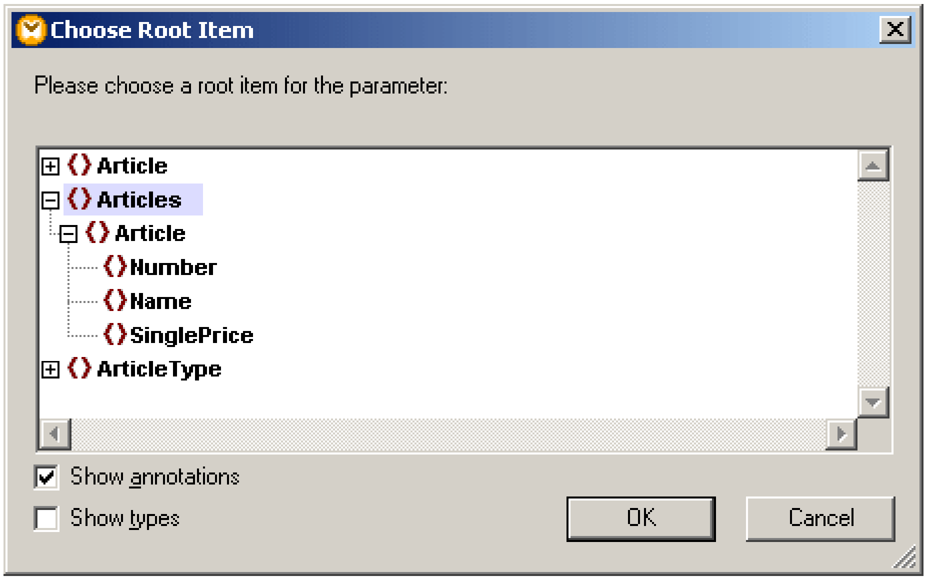
● Insert an existing structure as a parameter:

Filename / Connection	Path / Description
CompletePO.xsd	C:\Documents and Settings\My\My Documents\Altova\MapFo
ShortPO.xsd	C:\Documents and Settings\My\My Documents\Altova\MapFo
Articles.xsd	C:\Documents and Settings\My\My Documents\Altova\MapFo

○ Insert a new structure of the one of the following types:

- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure

5. Click "Insert a new structure... " radio button, select the XML Schema Structure entry, and click OK to continue.
6. Select **Articles.xsd** from the "Open" dialog box.
7. Click the element that you would like to become the root element in the component, e.g. Articles, and click OK, then OK again to close both dialog boxes.



Choose Root Item

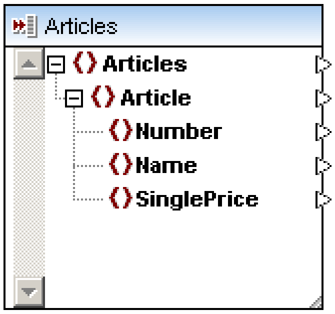
Please choose a root item for the parameter:

```
⊞ {} Article
⊟ {} Articles
   ⊟ {} Article
        {} Number
        {} Name
        {} SinglePrice
⊞ {} ArticleType
```

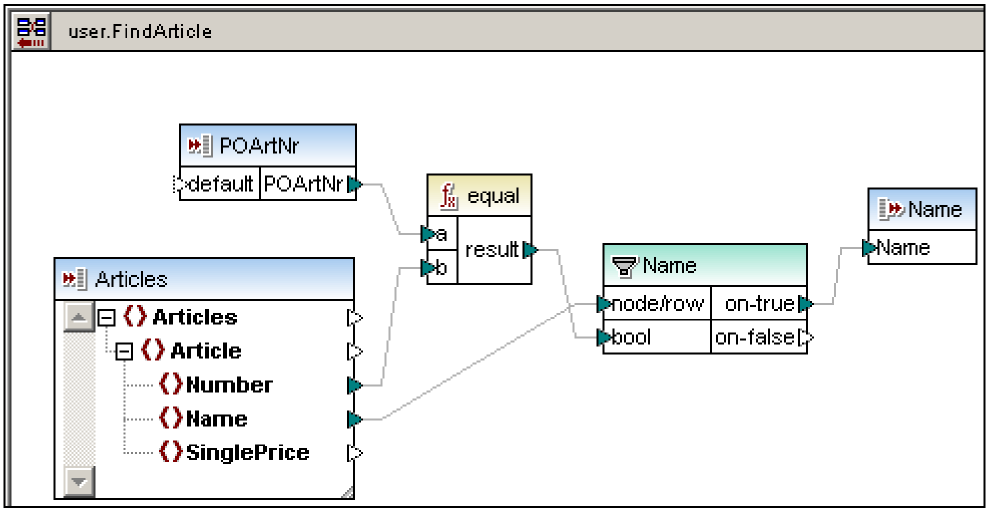
☑ Show annotations

☐ Show types

The Articles component is inserted into the user-defined function. Please note the input icon to the left of the component name. This shows that the component is used as a complex input component.

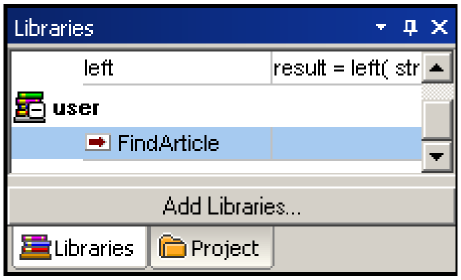


8. Insert the rest of the components as shown in the screenshot below, namely: a second "simple" input component (called POArtNr), filter, equal and output component (called Name), and connect them as shown.

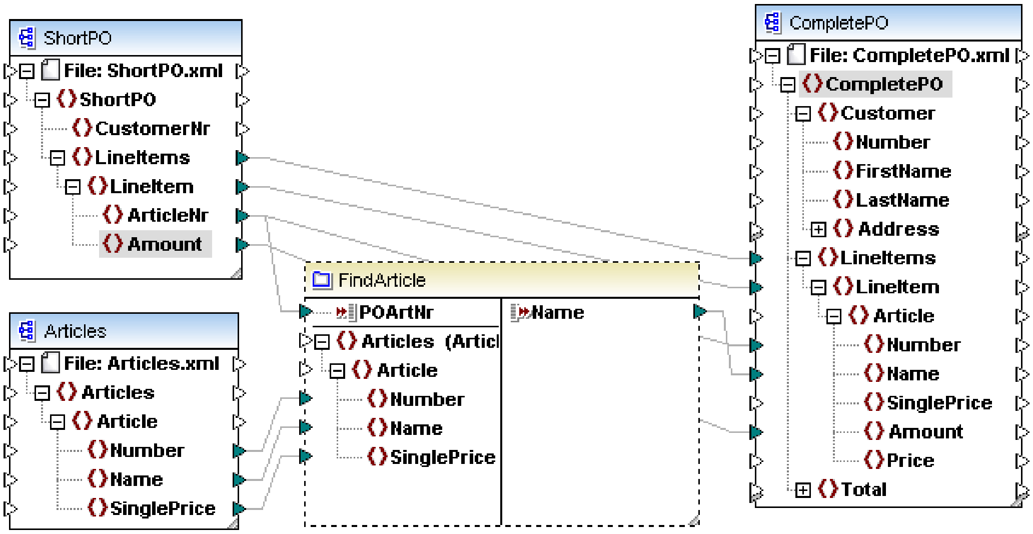


- The **Articles** input component receives its data from **outside** of the user-defined function. Input icons that allow mapping to this component, are available there.
- An XML **instance** file to provide data from within the user-defined function, cannot be assigned to a complex input component.
- The other input component POArtNr, supplies the ShortPO article number data to which the **Article | Number** is compared.
- The filter component filters the records where both numbers are identical, and passes them on to the output component.

10. Click the Home icon to return to the mapping.
11. Drag the newly created user-defined component from the Libraries pane into the mapping.



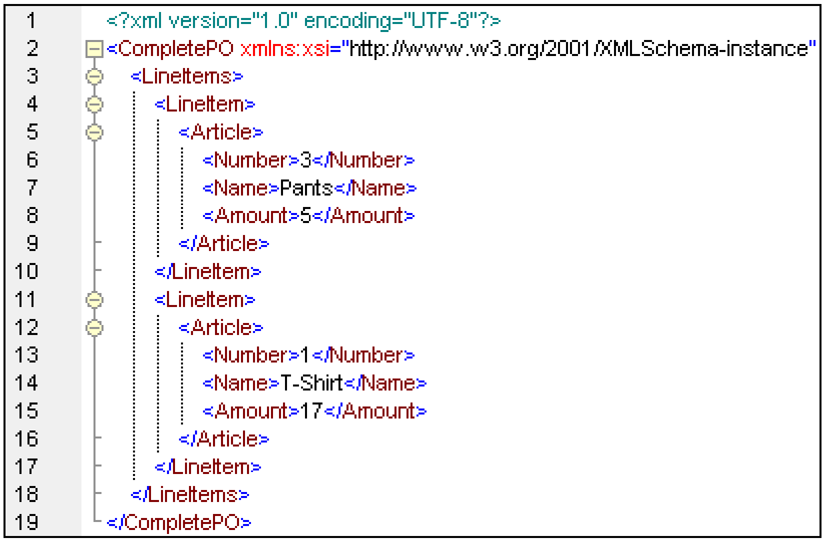
12. Create the connections as shown in the screenshot below.



The left half contains the input parameters to which items from two schema/xml files are mapped:

- **ShortPO** supplies the data for the input component **POArtNr.**
- **Articles** supplies the data for the complex input component. The Articles.xml instance file was assigned to the Articles schema file when the component was inserted.
- The complex input component **Articles** with its XML child nodes, to which data has been mapped from the Articles component.

The right half contains a simple output parameter called "**Name"**, which passes the filtered line items which have the same Article number to the "Name" item of **CompletePO**.



```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <LineItems>
    <LineItem>
      <Article>
        <Number>3</Number>
        <Name>Pants</Name>
        <Amount>5</Amount>
      </Article>
    </LineItem>
    <LineItem>
      <Article>
        <Number>1</Number>
        <Name>T-Shirt</Name>
        <Amount>17</Amount>
      </Article>
    </LineItem>
  </LineItems>
</CompletePO>
```

Note: When creating **Copy-all** connections between a schema and a user-defined function parameter, the two components must be based on the same schema. It is not necessary that they both have the same root elements however.

8.3.6 Complex user-defined function - XML node as output

This example is provided as the **lookup-udf-out.mfd** file available in the **...\MapForceExamples** folder. What this section will show is how to define an inline user-defined function that allows a complex output component.

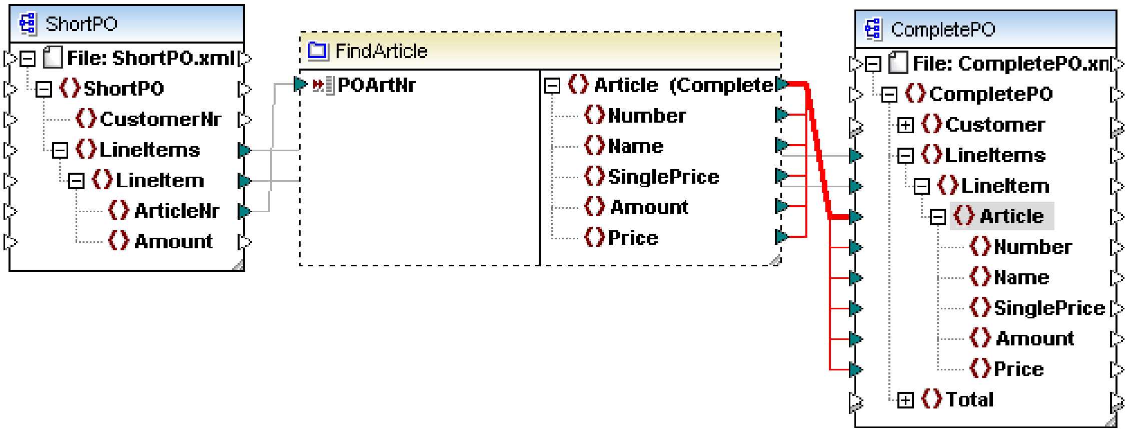
Note that the user-defined function FindArticle consists of two halves.

A left half which contains the input parameter:

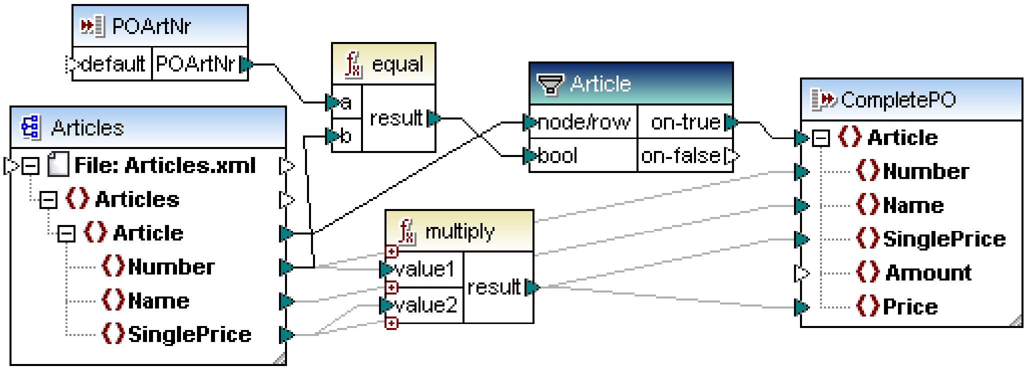
- a simple input parameter **POArtNr**

A right half which contains:

- a complex output component **Article (CompletePO)** with its XML child nodes mapped to CompletePO.



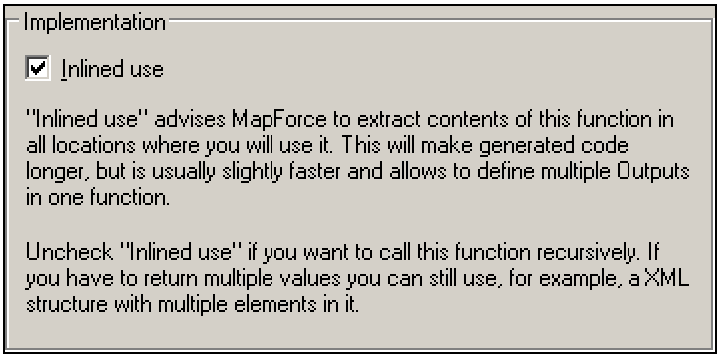
The screenshot below shows the constituent components of the user-defined function, the input components at left and the complex output component at right.



8.3.6.1 *Defining Complex Output Components*

Follow these steps to create a function that returns an XML structure as output parameter:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** name it **FindArticle**, and click OK to confirm. Note that the **Inline...** option is automatically selected.



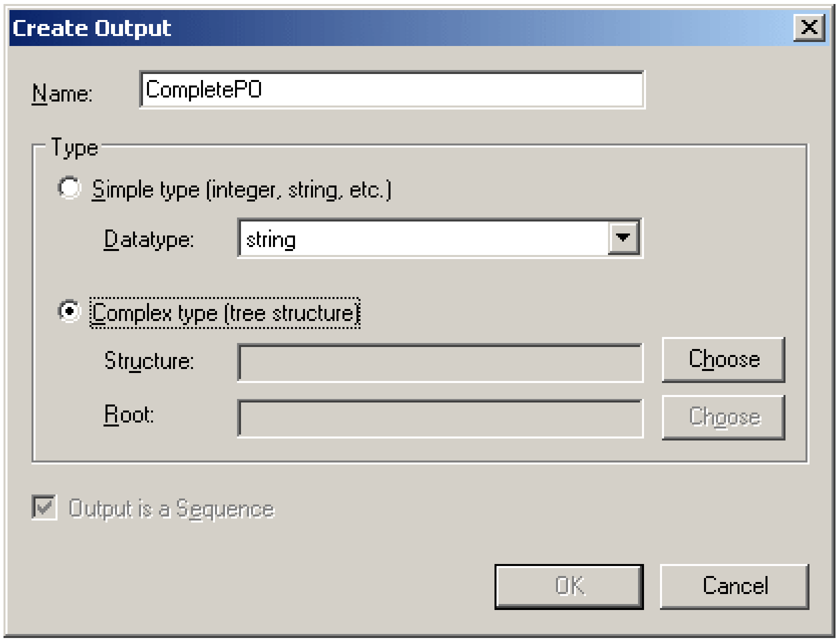
Implementation

☑ Inlined use

"Inlined use" advises MapForce to extract contents of this function in all locations where you will use it. This will make generated code longer, but is usually slightly faster and allows to define multiple Outputs in one function.

Uncheck "Inlined use" if you want to call this function recursively. If you have to return multiple values you can still use, for example, a XML structure with multiple elements in it.

2. Click the Insert **Output** icon ⊞ in the icon bar, and enter a name e.g. CompletePO.



Create Output

Name: CompletePO

Type

○ Simple type (integer, string, etc.)

Datatype: string

● Complex type (tree structure)

Structure:

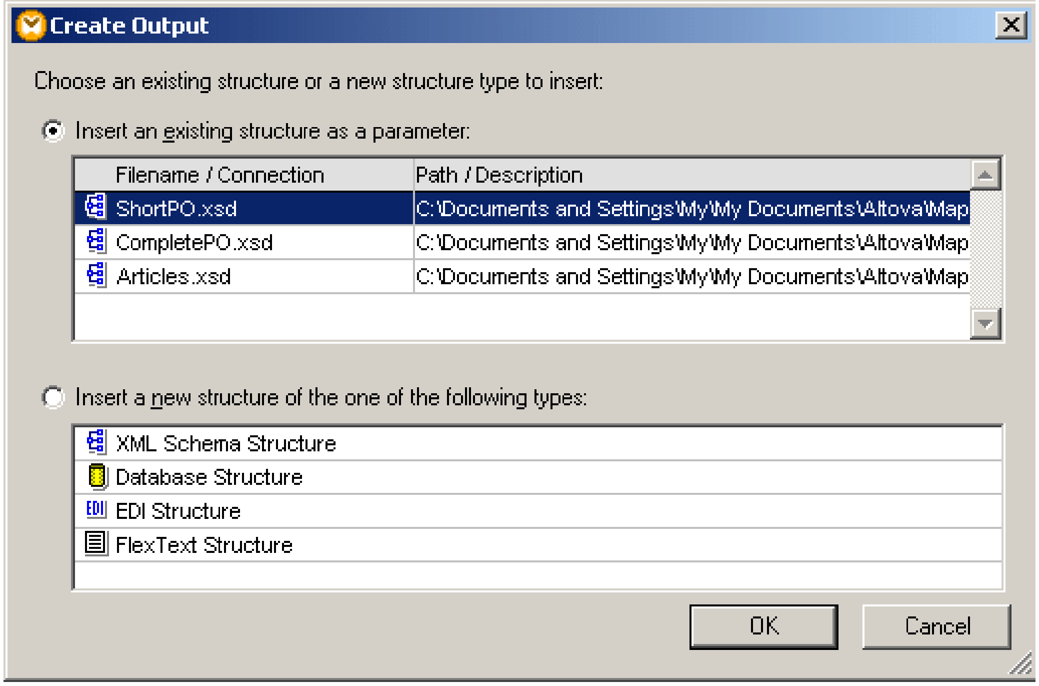
Root:

☑ Output is a Sequence

3. Click the **Complex type...** radio button, then click the "**Choose**" button. This opens another dialog box.

   The top list box displays the **existing** components in the mapping, (three schemas if you opened the example file). Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML Schema , database file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, file.



Create Output

Choose an existing structure or a new structure type to insert:

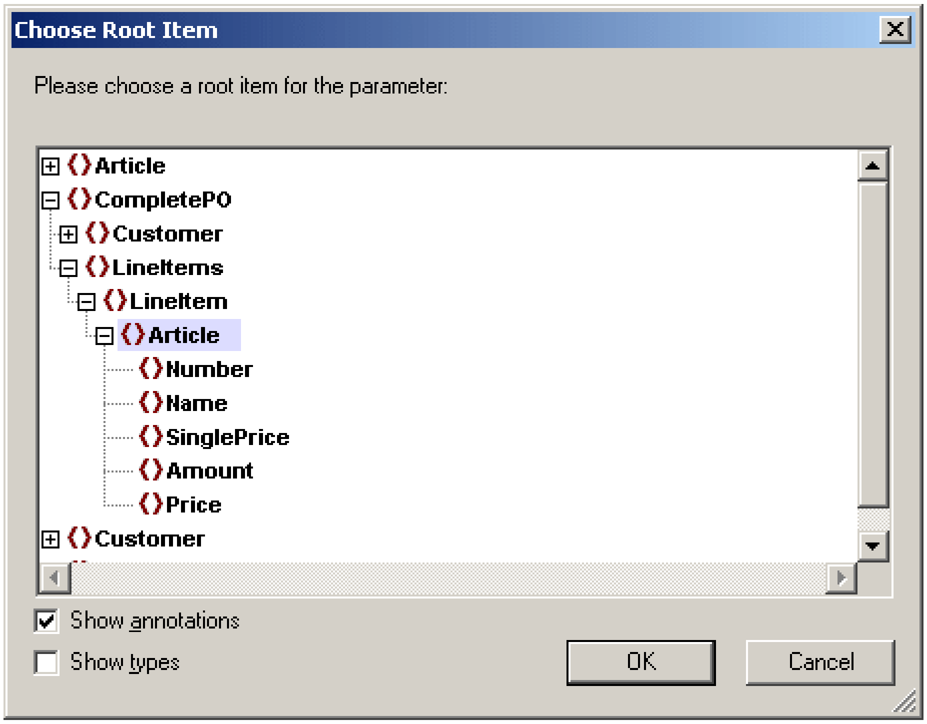
● Insert an existing structure as a parameter:

Filename / Connection	Path / Description
ShortPO.xsd	C:\Documents and Settings\My\My Documents\Altova\Map
CompletePO.xsd	C:\Documents and Settings\My\My Documents\Altova\Map
Articles.xsd	C:\Documents and Settings\My\My Documents\Altova\Map

○ Insert a new structure of the one of the following types:

- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure

4. Click "**Insert new structure...** " radio button, select the XML Schema Structure entry, and click OK to continue.
5. Select the **CompletePO.xsd** from the "Open" dialog box.
6. Click the element that you would like to become the root element in the component, e.g. **Article**, and click OK, then OK again to close the dialog boxes.



Choose Root Item

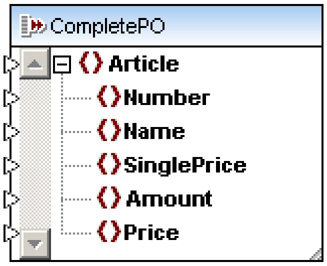
Please choose a root item for the parameter:

```
⊞ {} Article
⊟ {} CompletePO
   ⊞ {} Customer
   ⊟ {} LineItems
      ⊟ {} LineItem
         ⊟ {} Article
               {} Number
               {} Name
               {} SinglePrice
               {} Amount
               {} Price
⊞ {} Customer
```

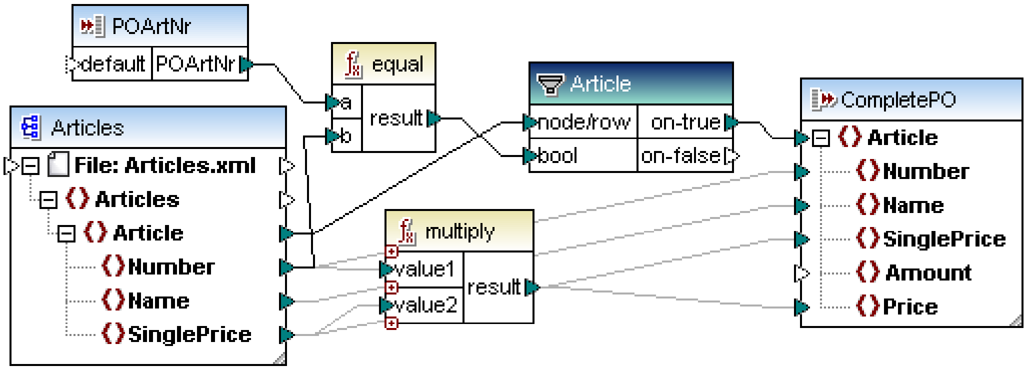
☑ Show annotations

☐ Show types

The CompletePO component is inserted into the user-defined function. Please note the output icon to the left of the component name. This shows that the component is used as a complex output component.

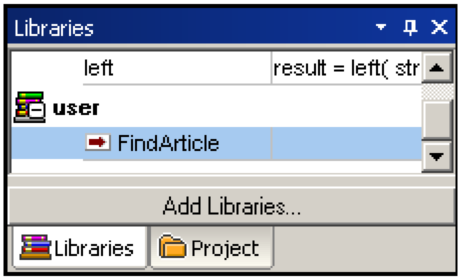


7. Insert the Articles schema/XML file into the user-defined function and assign the **Articles.xml** as the XML instance.
8. Insert the rest of the components as shown in the screenshot below, namely: the "simple" input components (POArtNr), filter, equal and multiply components, and connect them as shown.



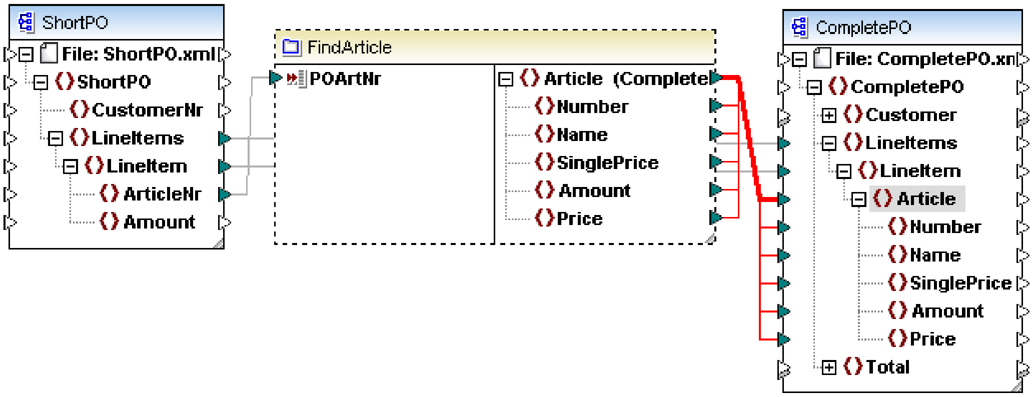
- The **Articles** component receives its data from the Articles.xml instance file, within the user-defined function.
- The input components supply the POArtNr (article number) and Amount data to which the Articles | Number & Price are compared.
- The filter component filters the records where both numbers are identical, and passes them on to the CompletePO output component.

9. Click the Home icon to return to the mapping.
10. Drag the newly created user-defined component from the Libraries pane into the mapping.



11. Create the connections as shown in the screenshot below.
Having created the Article (CompletePO) connector to the target, right click it and select

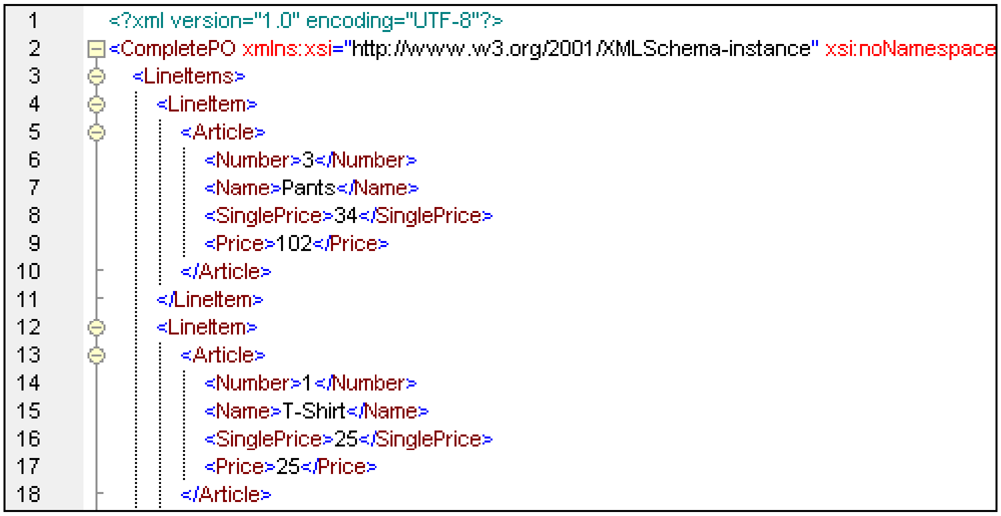
"Copy-all" from the context menu. The rest of the connectors are automatically generated, and are highlighted in the screenshot below.



Please note:
When creating **Copy-all** connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema. It is not necessary that they both have the same root elements however.

The left half contains the input parameter to which a single item is mapped; ShortPO supplies the article number to the **POArtNr** input component.

The right half contains a complex output component called "**Article (CompletePO)**" with its XML child nodes, which maps the filtered items, of the same Article number, to CompletePO.



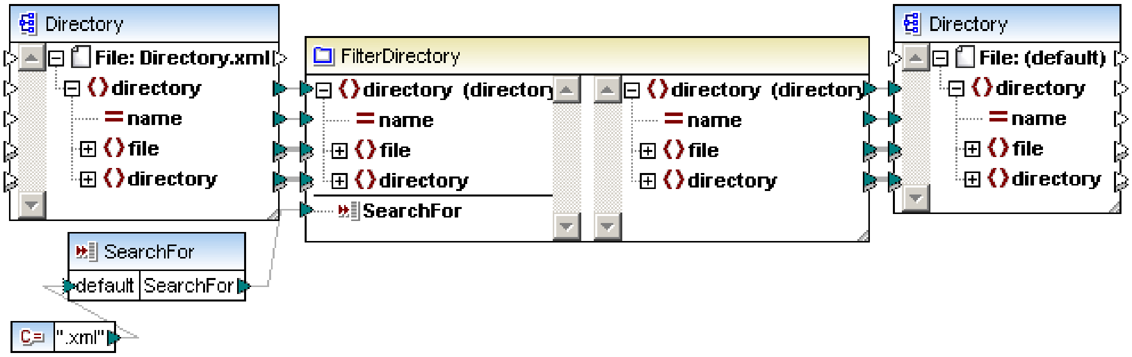
```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
  <LineItems>
    <LineItem>
      <Article>
        <Number>3</Number>
        <Name>Pants</Name>
        <SinglePrice>34</SinglePrice>
        <Price>102</Price>
      </Article>
    </LineItem>
    <LineItem>
      <Article>
        <Number>1</Number>
        <Name>T-Shirt</Name>
        <SinglePrice>25</SinglePrice>
        <Price>25</Price>
      </Article>
```

8.3.7 Recursive user-defined mapping

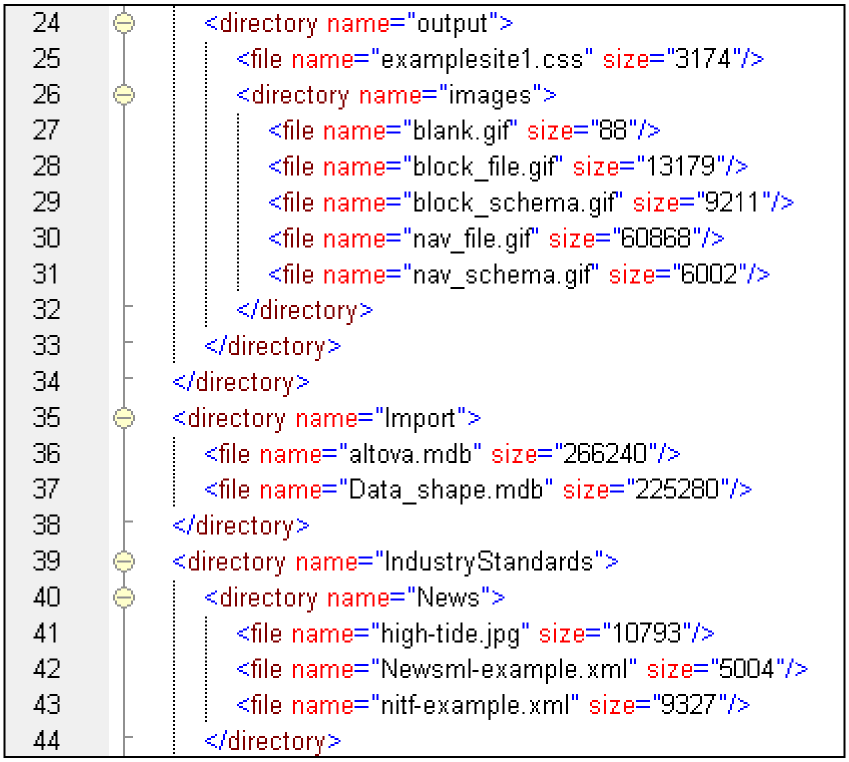
This section will describe how the mapping **RecursiveDirectoryFilter.mfd**, available in the ...**\MapForceExamples** folder, was created and how recursive mappings are designed. The MapForceExamples project folder contains further examples of recursive mappings.

The screenshot below shows the finished mapping containing the recursive user-defined function

FilterDirectory, the aim being to filter a list of the .xml files in the source file.

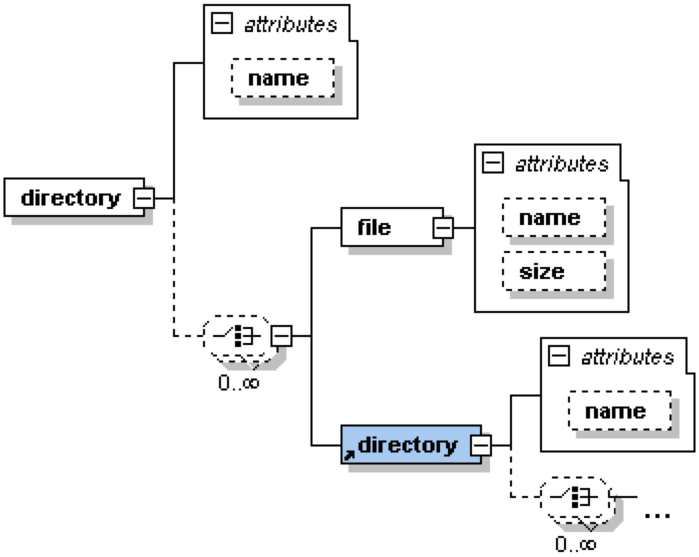


The **source file** that contains the file and directory data for this mapping is Directory.xml. This XML file supplies the directory and file data in the hierarchical form you see below.



```
  <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>
<directory name="Import">
  <file name="altova.mdb" size="266240"/>
  <file name="Data_shape.mdb" size="225280"/>
</directory>
<directory name="IndustryStandards">
  <directory name="News">
    <file name="high-tide.jpg" size="10793"/>
    <file name="Newsml-example.xml" size="5004"/>
    <file name="nitf-example.xml" size="9327"/>
  </directory>
```

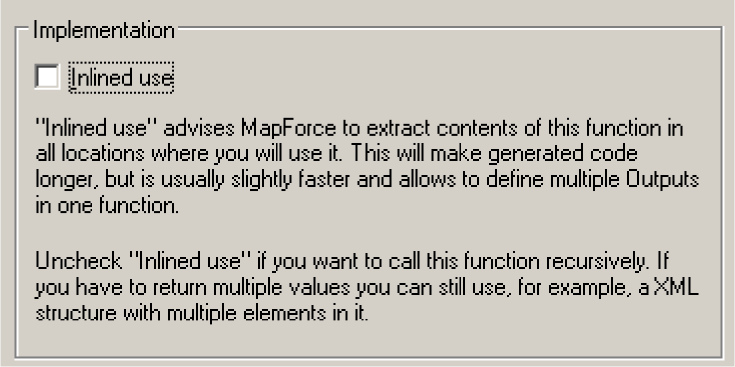
The XML schema file referenced by Directory.xml has a **recursive** element called "directory" which allows for any number of subdirectories and files below the directory element.



8.3.7.1 Defining a recursive user-defined function

Follow these steps to create a recursive user-defined function:

1. Select **Function | Create User defined Function** to start designing the function and enter a name e.g. FilterDirectory.
2. Make sure that you **deselect** the **Inlined Use** check box in the Implementation group, to make the function recursive, then click OK.



Implementation

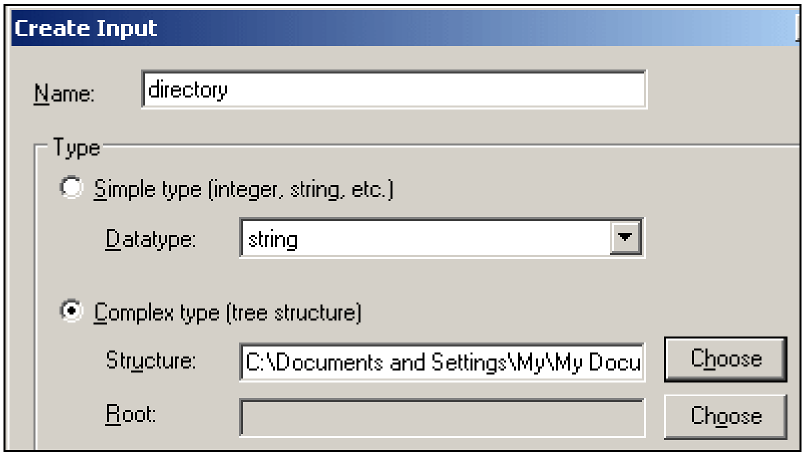
□ Inlined use

"Inlined use" advises MapForce to extract contents of this function in all locations where you will use it. This will make generated code longer, but is usually slightly faster and allows to define multiple Outputs in one function.

Uncheck "Inlined use" if you want to call this function recursively. If you have to return multiple values you can still use, for example, a XML structure with multiple elements in it.

You are now in the **FilterDirectory** window where you create the user-defined function.

3. Select **Function | Insert Input** to insert an input component.
4. Give the component a name e.g. "directory" and click on the **Complex Type** (tree structure) radio button.



Create Input

Name: directory

Type

○ Simple type (integer, string, etc.)

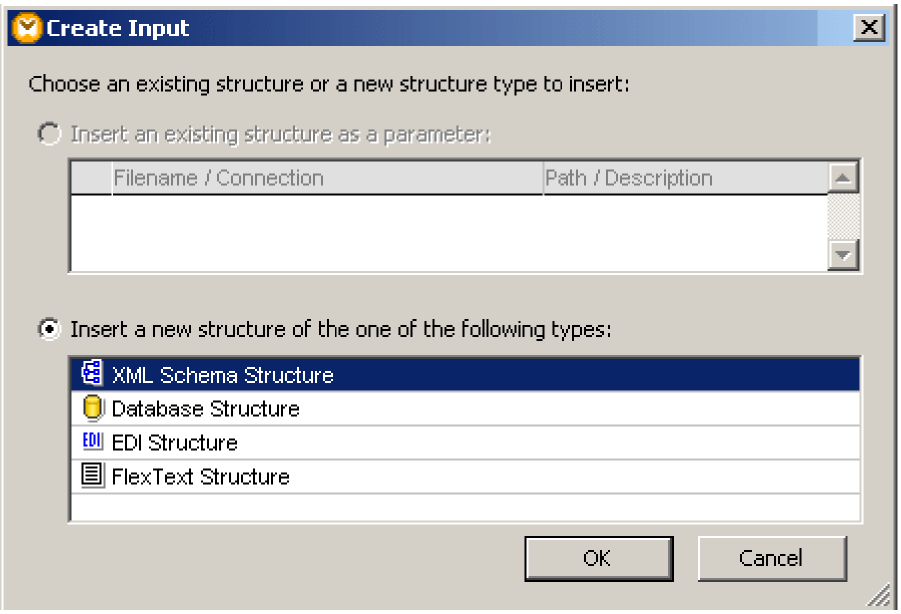
Datatype: string

● Complex type (tree structure)

Structure: C:\Documents and Settings\My\My Docu

Root:

5. Click the **Choose** button, click the "XML Schema Structure" entry in the lower pane, then click OK.



Create Input

Choose an existing structure or a new structure type to insert:

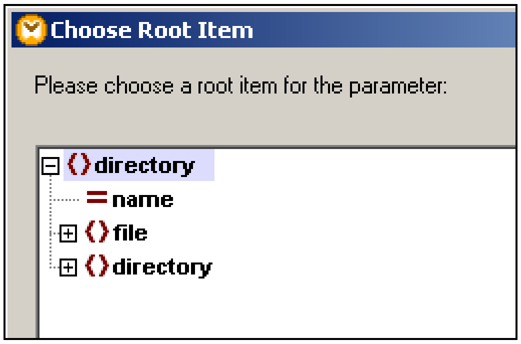
○ Insert an existing structure as a parameter:

Filename / Connection	Path / Description

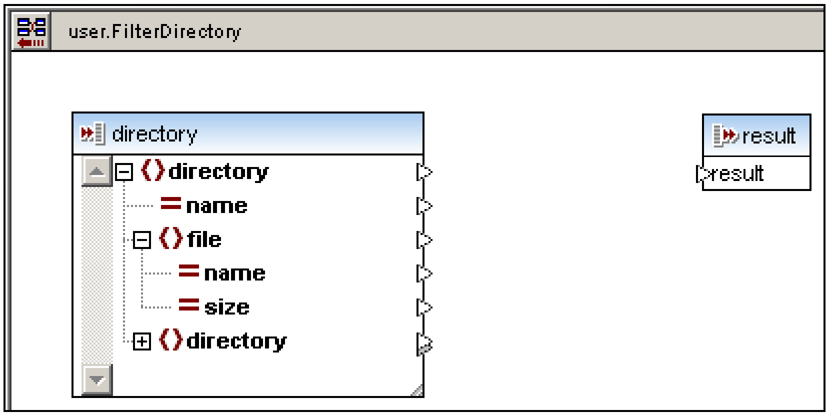
● Insert a new structure of the one of the following types:

- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure

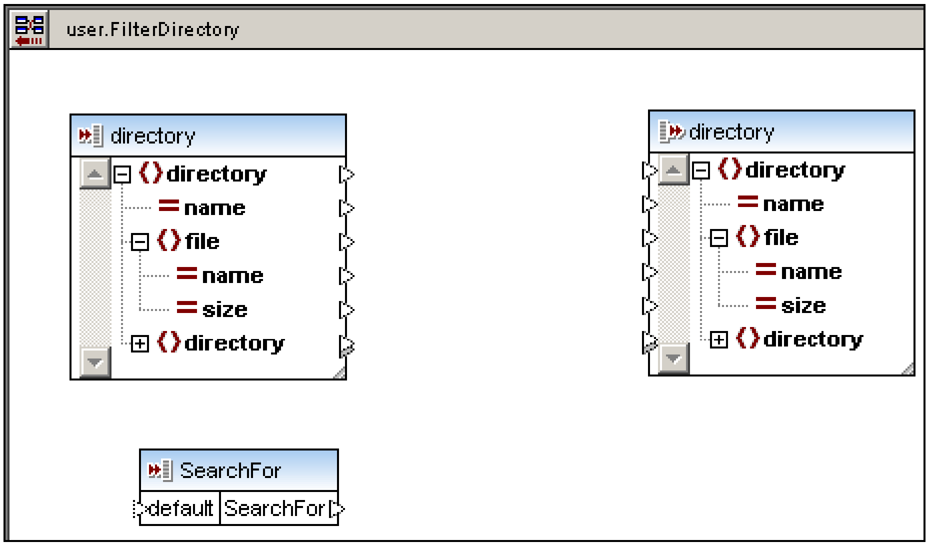
6. Select the **Directory.xsd** file in the ...\MapForceExamples folder and click the Open button.
7. Click OK again when asked to select the root item, which should be "directory" as shown below.



8. Click OK again to insert the complex input parameter.
   The user-defined function is shown below.



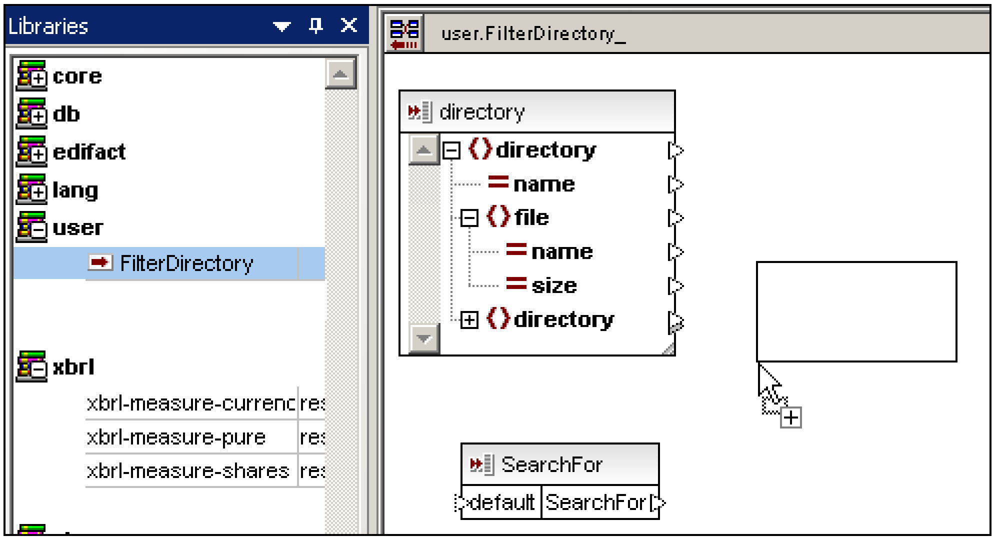
9. Delete the simple result output component, as we need to insert a complex output component here.
10. Select **Function | Insert Output...** to insert an **output** component and use the same method as outlined above, to make the output component, "directory", a complex type. You now have two complex components, one input and the other output.
11. Select **Function | Insert Input...** and insert a component of type Simple type, and enter a name e.g. **SearchFor**. Deselect the "Input is required" check box.



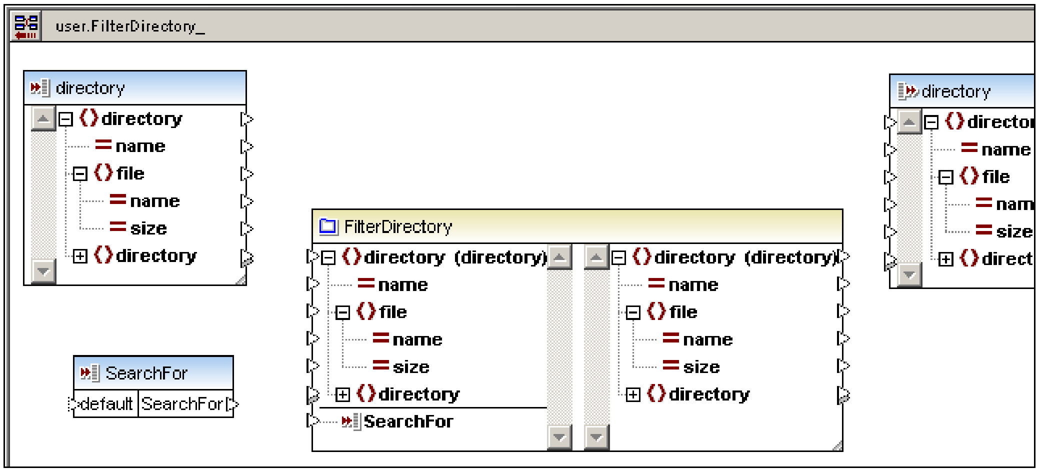
Inserting the recursive user-defined function

At this point, all the necessary input and output components have been defined for the user-defined function. What we need to do now is insert the "unfinished" function into the current user-defined function window. (You could do this at almost any point however.)

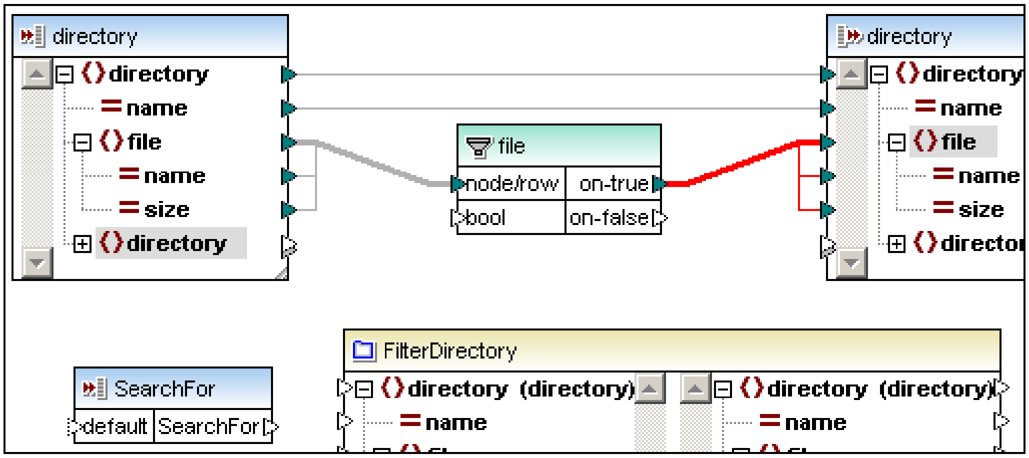
1. Find the FilterDirectory function in the **user** section of the **Libraries** window.
2. Click FilterDirectory then drag and drop it into the FilterDirectory window you have just been working in.



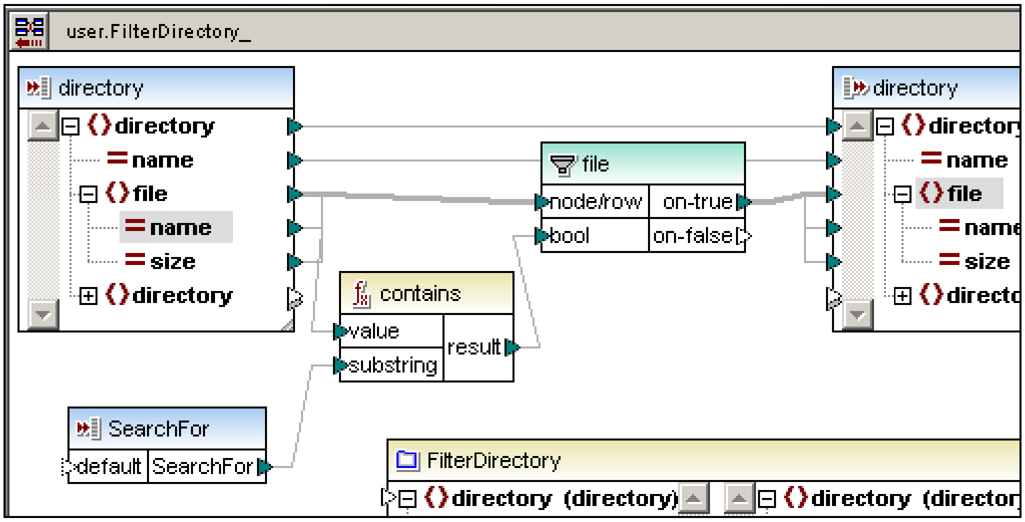
The user-defined function now appears in the user-defined function window as a recursive component.



3. Connect the **directory, name** and **file** items of the input component to the same items in the output component.
4. Right click the connector between the **file** items and select "Insert Filter" to insert a filter component.
5. Right click the on-true connector and select **Copy-All** from the context menu.
   The connectors change appearance to Copy-All connectors.



6. Insert a Contains function from the **Core | String functions** library.
7. Connect **name** to **value** and the **SearchFor** parameter to **substring**, then result to the bool item of the filter.



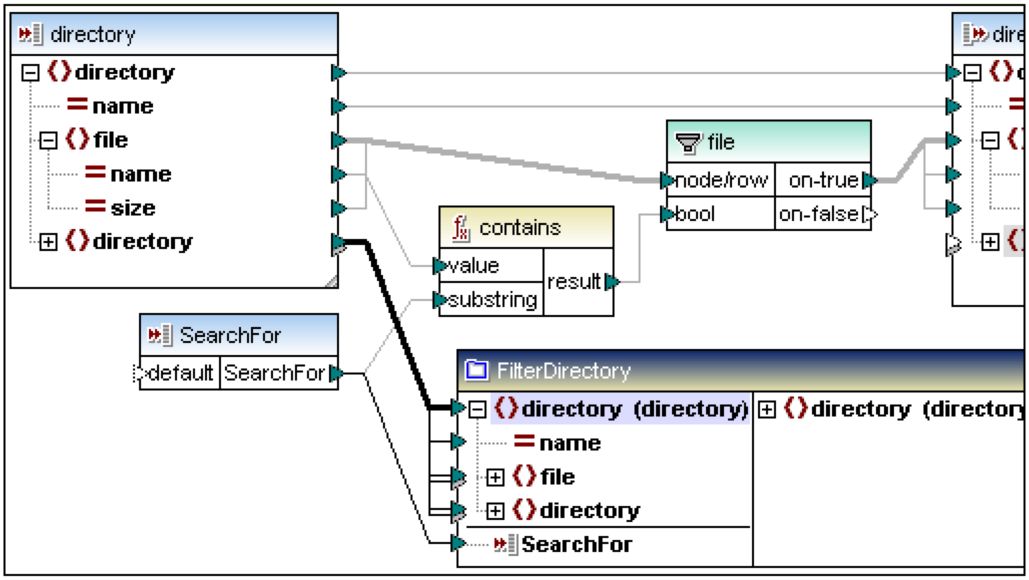
8. Connect the SearchFor item of the input component to the SearchFor item of the user-defined function.

Defining the recursion

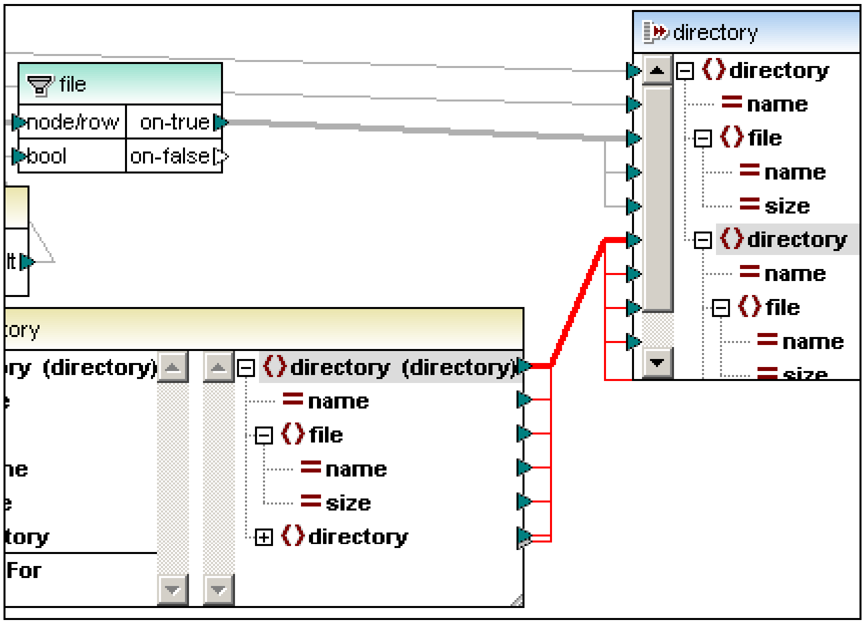
At this point, the mapping of a single directory recursion level is complete. Now we just need to define how to process a subdirectory.

Making sure that the Toggle Autoconnect icon is active in the icon bar:

1. Connect the lower **directory** item of the input component to the top **directory** item of the recursive user-defined function.



2. Connect the top output directory item of the user-defined function to the lower directory item of the output component.
3. Right click the top connector, select Copy-All from the context menu and click OK when prompted if you want to create Copy-All connection.

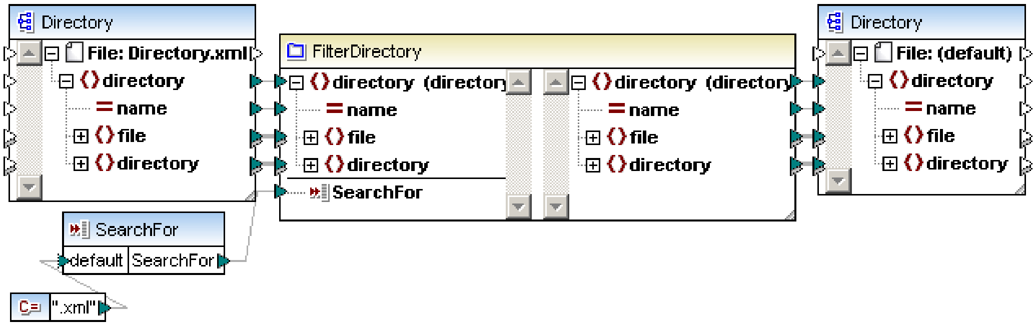


This completes the definition of the user-defined function in this window.

Click the Return to main mapping window icon, to continue defining the mapping there.

Main Mapping window

1. Drag the FilterDirectory function from the **user** section of the Libraries window, into the **main** mapping area.
2. Use **Insert | XML Schema file** to insert Directory.xsd and select Directory.xml as the instance file.
3. Use the same method to insert Directory.xsd and select Skip, to create the output component.
4. Insert a constant component, then a Input component e.g. SearchFor.
5. Create the connections as shown in the screenshot below.
6. When connecting the top-level connectors, directory to directory, on both sides of the user-defined component, right click the connector and select **Copy-All** from the context menu.



7. Click the Output tab to see the result of the mapping.

Notes:
Double clicking the lowest "directory" item in the Directory component, opens a new level of recursion, i.e. you will see a new **directory | file | directory** sublevel. Using the Copy-all connector automatically uses all existing levels of recursion in the XML instance, you do not need expand the recursion levels manually.

8.4 Importing Custom XSLT 1.0 or 2.0 Functions

You can extend the XSLT 1.0 and 2.0 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. On the **Tools** menu, click **Options**. (Alternatively, click **Add/Remove Libraries** in the lower area of the Libraries window.)
2. Next to **Libraries**, click **Add** and browse for the .xsl or .xslt file.

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 selected XSLT as transformation language (see Selecting a Transformation Language).

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.

Datatypes 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.

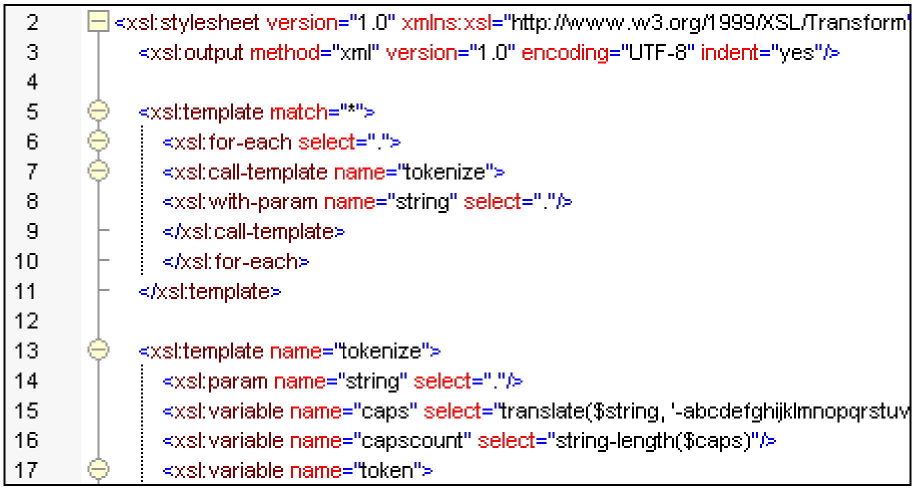
See also:

Example: Adding Custom XSLT 1.0 Functions
Example: Summing Node Values
XSLT 1.0 engine implementation
XSLT 2.0 engine implementation

8.4.1 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 **<Documents>\Altova\MapForce2018\MapForceExamples\** directory.

- **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.

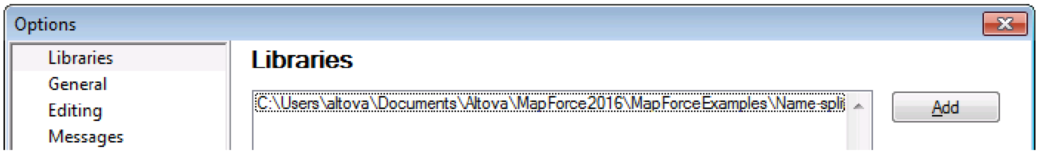


```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform'
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="*">
        <xsl:for-each select=".">
        <xsl:call-template name="tokenize">
        <xsl:with-param name="string" select="."/>
        </xsl:call-template>
        </xsl:for-each>
    </xsl:template>

    <xsl:template name="tokenize">
        <xsl:param name="string" select="."/>
        <xsl:variable name="caps" select="translate($string, '-abcdefghijklmnopqrstuv
        <xsl:variable name="capscount" select="string-length($caps)"/>
        <xsl:variable name="token">
```

- **Name-splitter.xml** (the source XML instance file to be processed)
- **Customers.xsd** (the source XML schema)
- **CompletePO.xsd** (the target XML schema)

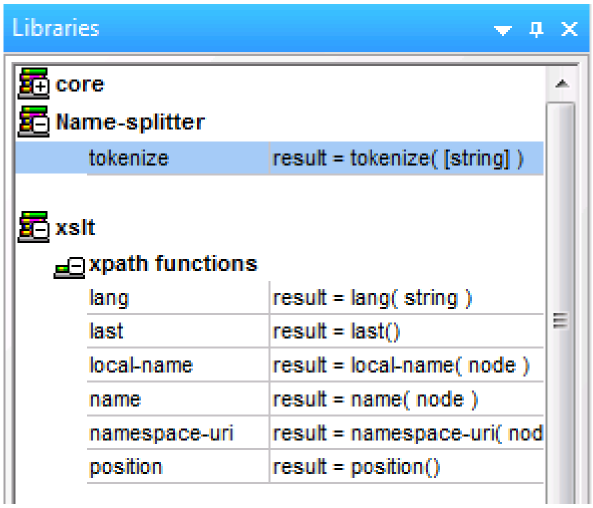
To add a custom XSLT function:

1. Select XSLT as transformation language (see Selecting a Transformation Language).
2. Click the **Add/Remove Libraries** button, in the lower area of the Libraries window. Alternatively, on the **Tools** menu, click **Options**, and then select **Libraries**.
3. Click **Add**, and 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**.



C:\Users\altova\Documents\Altova\MapForce2016\MapForceExamples\Name-spli

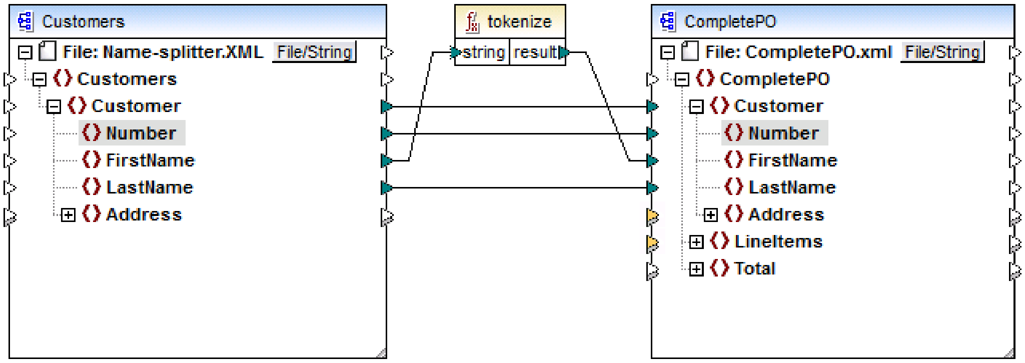
4. Click **OK**. The XSLT file name appears in the Libraries window, along with the functions defined as named templates (in this example, **Name-splitter** with the **`tokenize`** function).



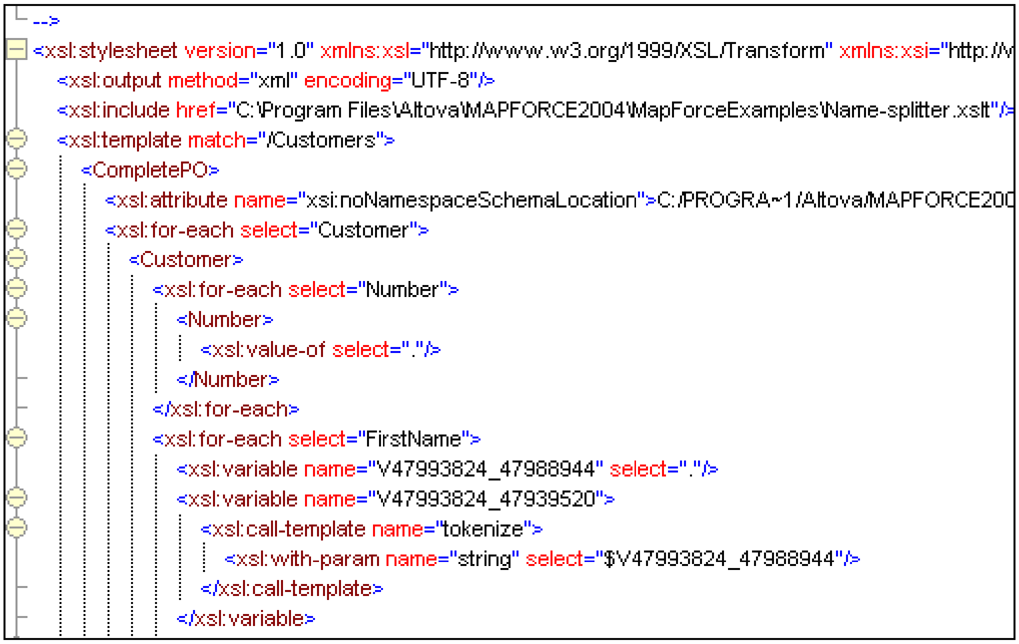
tokenize	result = tokenize([string])
lang	result = lang(string)
last	result = last()
local-name	result = local-name(node)
name	result = name(node)
namespace-uri	result = namespace-uri(nod
position	result = position()

To use the XSLT function in your mapping:

1. Drag the **`tokenize`** function into the Mapping window and map the items as show below.



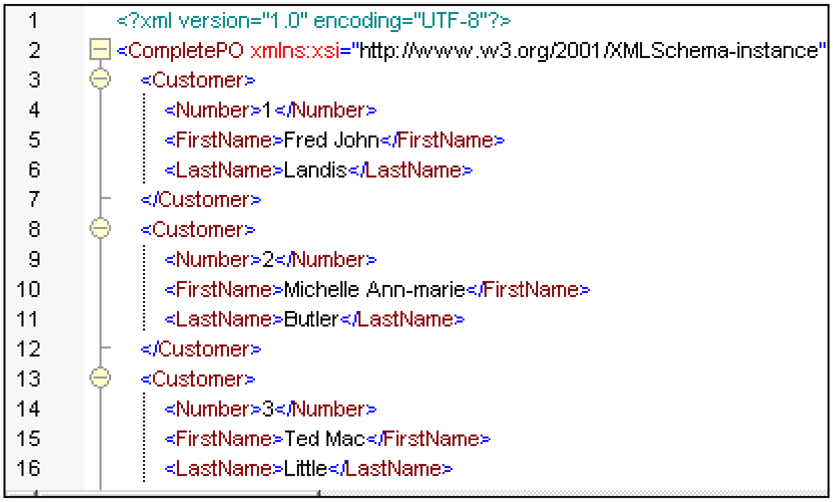
2. Click the XSLT tab to see the generated XSLT code.



```
-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xsi="http://v
    <xsl:output method="xml" encoding="UTF-8"/>
    <xsl:include href="C:\Program Files\Altova\MAPFORCE2004\MapForceExamples\Name-splitter.xslt"/>
    <xsl:template match="/Customers">
        <CompletePO>
            <xsl:attribute name="xsi:noNamespaceSchemaLocation">C:/PROGRA~1/Altova/MAPFORCE200
            <xsl:for-each select="Customer">
                <Customer>
                    <xsl:for-each select="Number">
                        <Number>
                            <xsl:value-of select="."/>
                        </Number>
                    </xsl:for-each>
                    <xsl:for-each select="FirstName">
                        <xsl:variable name="V47993824_47988944" select="."/>
                        <xsl:variable name="V47993824_47939520">
                            <xsl:call-template name="tokenize">
                                <xsl:with-param name="string" select="$V47993824_47988944"/>
                            </xsl:call-template>
                        </xsl:variable>
```

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.



```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Customer>
        <Number>1</Number>
        <FirstName>Fred John</FirstName>
        <LastName>Landis</LastName>
    </Customer>
    <Customer>
        <Number>2</Number>
        <FirstName>Michelle Ann-marie</FirstName>
        <LastName>Butler</LastName>
    </Customer>
    <Customer>
        <Number>3</Number>
        <FirstName>Ted Mac</FirstName>
        <LastName>Little</LastName>
```

To remove custom XSLT libraries from MapForce:

1. Click the **Add/Remove Libraries** button, in the lower area of the Libraries window.
2. Click the XSLT library to be deleted, and then click **Delete**.

8.4.2 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\MapForce2018\MapForceExamples\Tutorial\** folder:

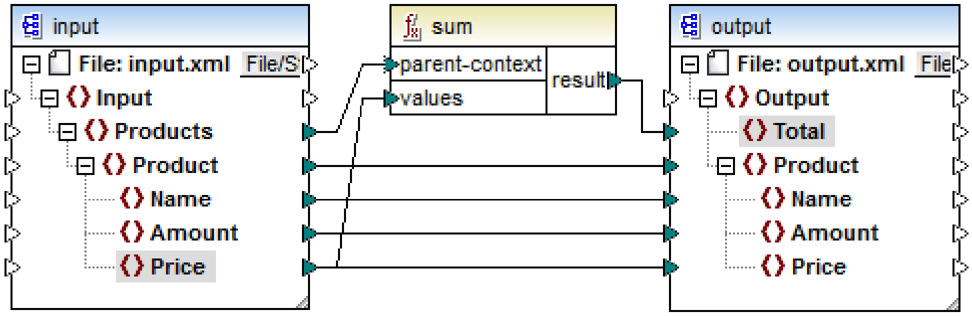
- **Summing-nodes.mfd** — the mapping file
- **input.xml** — the source XML file
- **input.xsd** — the source XML schema
- **output.xsd** — the target XML schema
- **Summing-nodes.xslt** — A custom XSLT stylesheet containing a named template to sum the individual nodes.

There are two different ways to achieve the goal of the mapping:

- By using the **sum** aggregate function of the **core** library. This function is available in the **Libraries** window (see also Working with Functions).
- By importing a custom XSLT stylesheet into MapForce.

Solution 1: Using the "sum" aggregate function

To use the **sum** aggregate 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.

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>
```

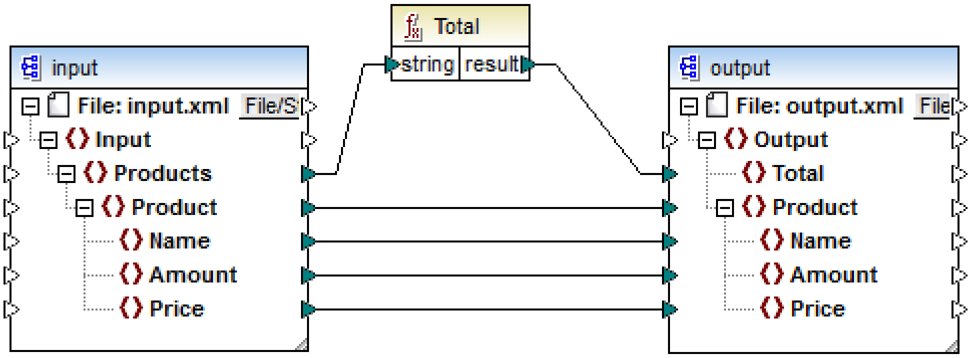
The image below shows a custom XSLT stylesheet which uses the named template "Total" and a single parameter `string`. The template works through the XML input file and sums all the values obtained by the XPath expression `/Product/Price`.

```
<?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"`.

To import the XSLT stylesheet into MapForce:

1. Select XSLT as transformation language. For more information, see Selecting a Transformation Language.
2. In the **Libraries** window, click **Add/Remove Libraries**.
3. On the **Options** dialog box, click the **Libraries** tab.
4. Click **Add** and browse for **<Documents>\Altova\MapForce2018\MapForceExamples \Tutorial\Summing-nodes.xslt**.
5. Drag the Total function from the newly created "Summing-nodes" library 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 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>
```

8.5 Importing Custom XQuery 1.0 Functions

When XQuery is selected as mapping transformation language, MapForce displays the 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. On the **Tools** menu, click **Options**. (Alternatively, click **Add/Remove Libraries** in the lower area of the Libraries window.)
2. Next to **Libraries**, click **Add** and browse for the .xq or .xquery library file.

If the imported library module is not supported, a message box prompts you. Otherwise, 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.

If you do not see the imported XQuery library module, make sure that XQuery is selected as transformation language (see Selecting a Transformation Language).

See also:

XQuery engine implementation

8.6 Importing Custom Java and .NET Libraries

Compiled Java class files as well as .NET DLL assemblies (including .NET 4.0 assemblies) can be imported 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 generated code.

Notes:

- To import custom Java or .NET functions, you need compiled Java classes (.class) or the .NET.dll assembly files. Importing Java .jar files or .dll files that are not a .NET assembly is not supported.
- 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 support).
- .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# (for example, C++.NET or VB.NET), provided they use only the basic data types from the System Assembly as parameters and return types (see also .NET Function Support).
- Setting the mapping language to C++ is not supported if the mapping uses imported Java .class or .NET DLL assemblies.
- Importing functions from native C++ DLLs is limited and requires a special approach. For more information, see Referencing Java, C# and C++ Libraries Manually.
- Setting the mapping language to XSLT is not supported 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).
- All functions called from a MapForce mapping should be “idempotent” (this means that they 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 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, as well as 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, be 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`
`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`

Schema type	.NET type	C# type
`xs:unsignedShort`	`System.UInt16`	`ushort`
`xs:unsignedInt`	`System.UInt32`	`uint`
`xs:unsignedLong`	`System.UInt64`	`ulong`
`xs:boolean`	`System.Boolean`	`bool`
`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.

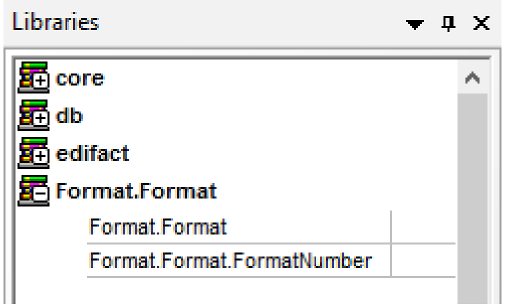
8.6.1 Example: Import Custom Java Class

This example illustrates how to import a custom Java .class file into MapForce.

Note: Java SE 7 Runtime Environment or later is required to complete this example.

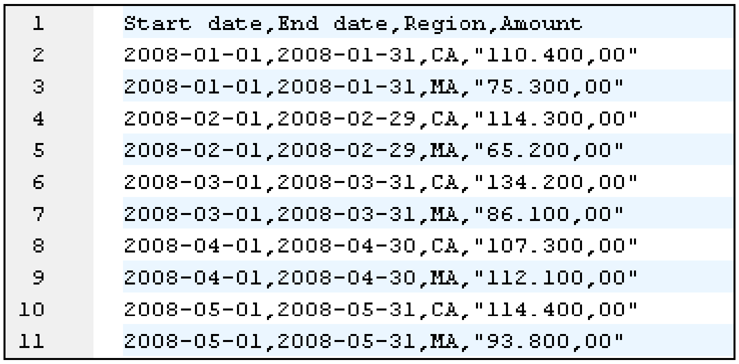
To add the Java .class file as MapForce library:

1. Set the transformation language to Java (see Selecting a Transformation Language).
2. Click the **Add/Remove Libraries** button in the lower area of the Libraries window.
3. Next to Libraries, click **Add**, and select the **Format.class** file from the **...\MapForceExamples\Java\Format\** directory. A message appears telling you that a new library has been added. The imported library is now visible in the Libraries window.



To preview the mapping output in MapForce:

1. Open the **FormatNumber.mfd** file available in the **...\MapForceExamples\Java** folder.
2. Click the Output button to see the result of the mapping.



```
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"
```

To run the mapping in Java:

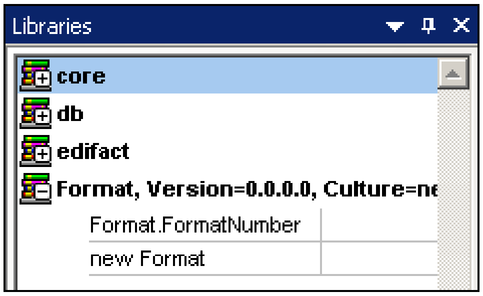
1. On the **File** menu, click **Generate Code In | Java**.
2. Select a target directory where the code should be generated, and click OK.
3. Import the generated libraries into your Java project and build the Java application (for an example, see Example: Build a Java application with Eclipse and Ant).

8.6.2 Example: Import Custom .NET DLL Assembly

This example illustrates how to import into MapForce a custom .NET DLL assembly created in C#.

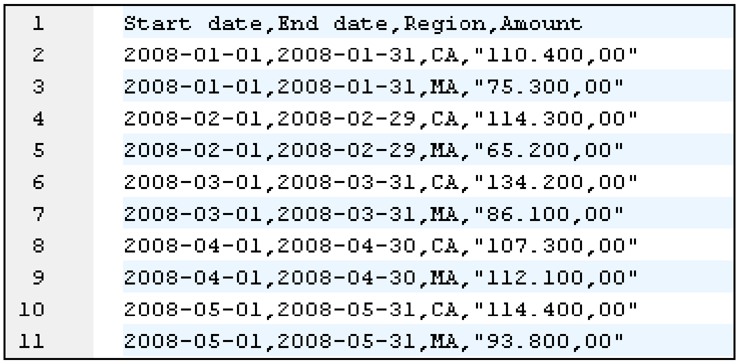
To add the .NET assembly file:

1. Set the transformation language to C# (see Selecting a Transformation Language).
2. Click the **Add/Remove Libraries** button in the lower area of the Libraries window.
3. Next to Libraries, click **Add**, and select the **Format.dll** file from the **...\MapForceExamples\C#\Format\bin\Debug\** directory. A message appears telling you that a new library has been added. The imported library is now visible in the Libraries window.



To preview the mapping output:

1. Open the **FormatNumber.mfd** file available in the **...\MapForceExamples\C#** folder.
2. Click the Output button to see the result of the mapping.



```
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"
```

To run the mapping from a custom C# application:

1. On the **File** menu, click **Generate Code In | C#**.
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 (see also Generating C# code).

8.7 Referencing Java, C# and C++ Libraries Manually

As an alternative approach to importing custom libraries into MapForce directly, you can create references to them using a custom .mff file (MapForce Function File) recognized by MapForce. The .mff library file is essentially an XML file where you manually define the linking between class definitions in your custom code and MapForce. Once you create the custom .mff file, you can import it into MapForce, similar to how you would import a .NET DLL or Java class file.

Notes:

- For an imported function to appear in the Libraries window, its return type and parameters must be of a simple type. For a list of data types available for each language, see Data Type Mapping.
- When you import function libraries from custom .mff files, the preview of the mapping directly in MapForce (by clicking the **Output** button) is limited. For libraries written in C++, preview of the mapping in MapForce is not supported. In case of Java and C#, preview is available when your library uses native language types, but it is not available if your library imports the Altova generated classes. Note, however, that 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 cache calculated results for reuse, or evaluate expressions in any order. It is therefore strongly 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, and they cannot and need not be saved to an *.mff file, because they are saved together with the mapping .mfd file where they have been created. For more information, see Reusing (importing) User-Defined Functions.
- If you are upgrading from a MapForce version earlier than 2010, you may need to update the data types used in your custom functions (see Data Type Mapping).

For instructions on how to create and configure a custom .mff file, see Configuring the .mff File. For examples, see:

- Example: Create a Custom C# Library
- Example: Create a Custom C++ Library
- Example: Create a Custom Java Library

8.7.1 Configuring the .mff File

The MapForce Function File (.mff) is a configuration file in XML format that makes it possible to adapt functions from custom Java, C#, or C++ libraries into MapForce, so that they appear in the Libraries window. An .mff file essentially intermediates between your custom libraries and MapForce, and it must be configured to specify a) the interfaces to the custom functions and b) where the implementation can be found in generated code. This topic provides instructions on how to do this.

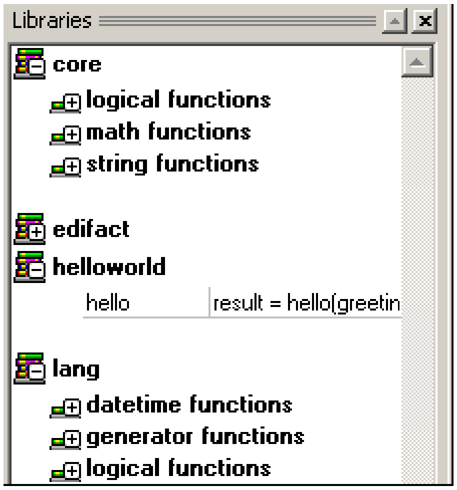
Note: The ***.mff** library files must be valid against the **mff.xsd** schema file found in the **Altova\MapForce2018\MapForceExamples** folder, relative to your **(My) Documents** folder.

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.

The following code listing illustrates a sample .mff file for C#:

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:noNamespaceSchemaLocation="mff.xsd" version="8" library="helloworld">
   <implementations>
      <implementation language="cs">
         <setting name="namespace" value="HelloWorldLibrary"/>
         <setting name="class" value="Greetings"/>
         <setting name="reference" value="C:\HelloWorldLibrary
\HelloWorldLibrary.dll"/>
      </implementation>
   </implementations>
   <group name="string functions">
      <component name="hello">
         <sources>
            <datapoint name="greeting_type" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="cs">
               <function name="HelloFunction"/>
            </implementation>
         </implementations>
         <description>
            <short>result = hello(greeting_type)</short>
            <long>Returns a greeting sentence according to the given
greeting_type.</long>
         </description>
      </component>
   </group>
</mapping>
```

The image below shows a custom .mff file may look after import into MapForce. Notice that the custom library "helloworld" appears as a library entry (sorted alphabetically), containing the "hello" string function.



The steps needed to adapt the mff file to suit your needs are described below.

Configuring the library name

The library name is found in the .mff file line shown below. By convention, the library name is written in lowercase letters.

```
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:noNamespaceSchemaLocation="mff.xsd" version="8" library="helloworld">
```

In the sample above, the entry that will appear in the Libraries window will be called "helloworld".

Configuring the language implementations

The `<implementations>` element is mandatory element which specifies which languages your library should support, and it must be added as child of `<mapping>`, for example:

```
...
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:noNamespaceSchemaLocation="mff.xsd" version="8" library="helloworld">
   <implementations>
      <implementation language="cs">
         <setting name="namespace" value="HelloWorldLibrary"/>
         <setting name="class" value="Greetings"/>
         <setting name="reference" value="C:\HelloWorldLibrary
\HelloWorldLibrary.dll"/>
      </implementation>
   </implementations>
...
```

The settings within each `<implementation>` allow the generated code to call the specific functions defined in Java, C++ or C#.

An .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

```
...
<implementation language="java">
   <setting name="package" value="com.hello.functions"/>
   <setting name="class" value="Hello"/>
</implementation>
...
```

It is important for the generated code to be able to find your **Hello.class** file. Therefore, make sure that your class is in the Java classpath. The default Java classpath is found in the system environment variables.

Note that it is only possible to have one class per *.mff file when working with custom Java libraries.

C#

```
...
   <implementation language="cs">
      <setting name="namespace" value="HelloWorldLibrary"/>
      <setting name="class" value="Hello"/>
      <setting name="reference" value=" C:\HelloWorldLibrary
\HelloWorldLibrary.dll"/>
   </implementation>
...
```

For C#, it is important that the namespace in the code corresponds to the namespace defined in the .mff file (in the code listing above, the namespace is `HelloWorldLIbrary`). The same is true for the class name (in the code listing above, the class name is `Hello`). The third setting, `reference`, provides the path of the dll that is to be linked to the generated code.

C++

```
...
   <implementation language="cpp">
         <setting name="namespace" value="helloworld"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\HelloWorldLibrary"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
   </implementation>
...
```

For the C++ sample listing above, note the following:

- namespace is the namespace in which your Greetings class will be defined. It must be equal to the library name.
- 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** (defined when selecting the menu option **File | Generate xxx code**, and selecting the directory) and included in the project file.

All the include files you supply will be included in the generated algorithm.

Adding a component

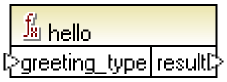
In the Libraries window of the MapForce graphics user interface, each function appears nested under a function group, for example "string functions". In the .mff file, a function corresponds to a <component> element. Conversely, each <component> must be nested under a <group> element, for example:

```
...
<group name="string functions">
   <component name="hello">
      …
   </component>
</group>
...
```

The code shown below defines a sample function (component) called hello.

```
...
<component name="hello">
   <sources>
      <datapoint name="greeting_type" type="xs:boolean"/>
   </sources>
   <targets>
      <datapoint name="result" type="xs:string"/>
   </targets>
   <implementations>
   …
   </implementations>
   <description>
      <short>result = hello(greeting_type)</short>
      <long>Returns a greeting sentence according to the given
greeting_type.</long>
   </description>
</component>
...
```

Here is how the component above would look in MapForce:



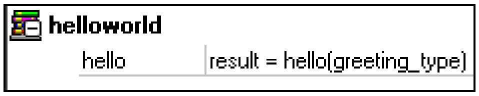
In the code listing above, a `<datapoint>` can be loosely defined as the input or output parameter of a function (also known as input or output connector). The `type` argument of the <datapoint> specifies the data type of the parameter (or the data type of the return value).

Only one target datapoint is allowed for each function. There is no limitation as to how many source datapoints you can define.

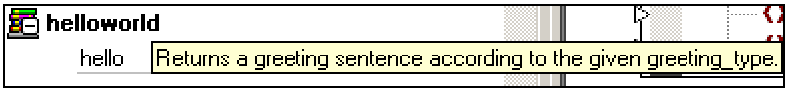
The data type of each datapoint must be one of the XML Schema types (for example, `xs:string`, `xs:integer`, etc.) These data types have to correspond to the data types of the function's parameters you defined in your Java, C++ or C# library. For the mapping of XML Schema datatypes to language types, see Data Type Mapping.

Functions are accompanied by short and long descriptions in the library 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.

Short description:



Long description:



Returns a greeting sentence according to the given greeting_type.

Defining language implementations

We are now at the point where we need to make a connection between the function in the Libraries window, and the function in the custom Java, C# or C++ classes. This is achieved through the `<implementation>` element.

As previously stated, one function may have multiple implementation elements – one for each supported programming language. A function may be called "helloFunction" in Java, or "HelloFunctionResponse" 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 like the following:

```
...
<component name="hello">
...
   <implementations>
      <implementation language="cs">
         <function name="HelloFunction"/>
      </implementation>
      <implementation language="java">
         <function name="helloFunction"/>
      </implementation>
      <implementation language="cpp">
         <function name="HelloFunctionResponse"/>
      </implementation>
   </implementations>
...
</component>
...
```

The value you supply as function name must exactly match the name of the method in the Java, C# or C++ class.

8.7.2 Importing the .mff File Into MapForce

After you have created a custom .mff file (see Configuring the .mff File), you can import it into MapForce as follows:

1. On the **Tools** menu, click **Options**. (Alternatively, click **Add/Remove Libraries** in the lower area of the Libraries window.)
2. Next to Libraries, click **Add**, and select the custom .mff file.

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 the **...\Altova\MapForce2018\MapForceLibraries** folder relative to the **Program Files** (or **Program Files (x86)** folder), then the library is automatically loaded into 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).

8.7.3 Data Type Mapping

The following table lists the data types supported as function return types and parameter types when you create custom .mff files that adapt your Java, C#, and C++ libraries to MapForce. The table lists both native and non-native data types. Note that, 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 Altova libraries. In case of C++, Altova libraries must always be imported. For information about how to generate the Altova libraries, see Code Generator.

XML Schema Type	Java Type	C# Type	C++ Type
anyAtomicType	String	string	string_type
anySimpleType	String	string	string_type
anyURI	String	string	string_type

XML Schema Type	Java Type	C# Type	C++ Type
base64Binary	byte[]	byte[]	altova::mapforce::blob
boolean	boolean	bool	bool
byte	int	int	int
date	com.altova.types.DateTime	Altova.Types.DateTime	altova::DateTime
dateTime	com.altova.types.DateTime	Altova.Types.DateTime	altova::DateTime
dayTimeDuration	com.altova.types.Duration	Altova.Types.Duration	altova::Duration
decimal	java.math.BigDecimal	decimal	double
double	double	double	double
duration	com.altova.types.Duration	Altova.Types.Duration	altova::Duration
ENTITIES	String	string	string_type
ENTITY	String	string	string_type
float	double	double	double
gDay	com.altova.types.DateTime	Altova.Types.DateTime	altova::DateTime
gMonth	com.altova.types.DateTime	Altova.Types.DateTime	altova::DateTime
gMonthDay	com.altova.types.DateTime	Altova.Types.DateTime	altova::DateTime
gYear	com.altova.types.DateTime	Altova.Types.DateTime	altova::DateTime
gYearMonth	com.altova.types.DateTime	Altova.Types.DateTime	altova::DateTime
hexBinary	byte[]	byte[]	altova::mapforce::blob
ID	String	string	string_type
IDREF	String	string	string_type
IDREFS	String	string	string_type
int	int	int	int
integer	java.math.BigInteger	decimal	__int64

XML Schema Type	Java Type	C# Type	C++ Type
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	Altova.Types.DateTime	altova::DateTime
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	Altova.Types.Duration	altova::Duration

8.7.4 Example: Create a Custom C# Library

This topic describes how to create a sample C# library and configure the .mff file so that it appears in the Libraries window of MapForce.

1. Create a new class library project in Visual Studio. Notice that the function has been defined as `public static`.

```
using System;

namespace HelloWorldLibrary
{
    public class Greetings
    {
        public static string HelloFunction(bool GreetingType)
        {
            if (GreetingType)
                return "Hello World!";
            return "Hello User!";
        }
    }
}
```

2. If you need special XML Schema types (such as date and duration), you will need 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, on the **Project** menu, click **Add Reference** and browse for the **Altova.dll** file. Also, add to your code the following line: **`using Altova.Types;`** . For information about how XML Schema types map to C# types, see Data Type Mapping.
3. Build your Visual Studio project. The **HelloWorldLibrary.dll** is generated in your project output directory.
4. Using an XML editor, create a new .mff file and validate it against the **..\Program Files\MapForceLibraries\mff.xsd** folder. Make sure that the text highlighted below points to the **HelloWorldLibrary.dll** file. For more information, see Configuring the .mff File.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="helloworld" 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="HelloWorldLibrary"/>
         <setting name="class" value="Greetings"/>
         <setting name="reference" value="C:\Projects
\HelloWorldLibrary.dll"/>
      </implementation>
   </implementations>
   <group name="Greetings">
```

```
      <component name="HelloFunction">
         <sources>
            <datapoint name="greeting_type" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string" />
         </targets>
         <implementations>
            <implementation language="cs">
               <function name="HelloFunction"/>
            </implementation>
         </implementations>
         <description>
            <short>result = hello(greeting_type)</short>
            <long>Returns a greeting according to the given greeting
type.</long>
         </description>
      </component>
   </group>
</mapping>
```

You have now finished creating a custom library and the .mff file which adapts it to MapForce. The custom .mff file can now be used in MapForce (see Importing the .mff File Into MapForce).

8.7.5 Example: Create a Custom C++ Library

This topic describes how to create a sample C++ library and configure a .mff file for it so that the library appears in the Libraries window of MapForce.

1. Create a header (.h) file for your class library. The following code listing illustrates a sample header file called **Greetings.h**.

```
#ifndef HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED
#define HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED

#if _MSC_VER > 1000
   #pragma once
#endif // _MSC_VER > 1000

using namespace altova;

namespace helloworld {

class ALTOVA_DECLSPECIFIER Greetings
{
public:
   static string_type   HelloFunctionResponse(bool greetingType);
};

} // namespace HelloWorldLibrary
```

```
#endif // HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED
```

Notice that the function has been 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 compile correctly—whether you use dynamic or static linkage in subsequently generated code.

2. 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 helloworld {

   string_type   Greetings::HelloFunctionResponse(bool greetingType)
   {
      if( greetingType )
         return _T("Hello World!");
      return _T("Hello User!");
   }

}
```

Notice the lines that import the **StdAfx.h** and several Altova libraries. These lines must be left unchanged. The paths to the Altova libraries is 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 are compiled with the rest of the generated mapping code.

3. Using an XML editor, create a new .mff file and validate it against the **..\Program Files \MapForceLibraries\mff.xsd** folder. Make sure that the text highlighted below points to the directory of the header and cpp files created previously. Remember that the namespace and function names and data types defined here must correspond to those in the C++ code, as described in Configuring the .mff File. For information about data type support, see Data Type Mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="helloworld" 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="helloworld"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Projects\HelloWorld"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
      </implementation>
   </implementations>
   <group name="Greetings">
      <component name="HelloFunctionResponse">
         <sources>
            <datapoint name="greeting_type" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="cpp">
               <function name="HelloFunctionResponse"/>
            </implementation>
         </implementations>
         <description>
            <short>result = hello(greeting_type)</short>
            <long>Returns a greeting according to the given greeting
type.</long>
         </description>
      </component>
   </group>
</mapping>
```

You have now finished creating a custom library and the .mff file which adapts it to MapForce. The custom .mff file can now be used in MapForce (see Importing the .mff File Into MapForce). Remember that, 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.

Resolving C++ compile errors

If you get a compiler error at the line shown below, modify the project properties to include a reference to the **msado15.dll** file.

```
#import "msado15.dll" rename("EOF", "EndOfFile")
```

In Visual Studio 2008:

1. On the **Tools** menu, click **Options**.
2. Expand **Projects and Solutions > VC++ Directories**.
3. Under "Show directories for", select **Include files**, and add a new entry that points to the directory where **msado15.dll** file is located (usually, **C:\Program Files\Common Files\System\ADO**).
4. Build the project.

8.7.6 Example: Create a Custom Java Library

This topic describes how to create a sample Java library and configure a .mff file for it so that the library appears in the Libraries window of MapForce.

1. Create a new Java project in your preferred development environment (for example, Eclipse).
2. Add to the project a new package called `com.hello.functions` which consists of a class called `Hello`. In the code listing below, notice that the `HelloFunction` function has been defined as `public static`.

```
package com.hello.functions;

public class Hello {
   public static String HelloFunction ( boolean greetingType ) {
      if( greetingType )
         return "Hello World!";
      return "Hello User!";
   }
}
```

3. 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.*;
```

4. Compile your custom library to a class file, and add it to the Java classpath.
5. Using an XML editor, create a new .mff file and validate it against the **..\Program Files \MapForceLibraries\mff.xsd** folder. Make sure that the text highlighted below points to the namespace and class defined previously in the Java code. For more information, see Configuring the .mff File.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="helloworld" 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="namespace" value="com.hello.functions"/>
         <setting name="class" value="Hello"/>
      </implementation>
   </implementations>
   <group name="Greetings">
      <component name="HelloFunction">
         <sources>
            <datapoint name="greeting_type" type="xs:boolean"/>
         </sources>
         <targets>
```

```
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="java">
               <function name="HelloFunction"/>
            </implementation>
         </implementations>
         <description>
            <short>result = hello(greeting_type)</short>
            <long>Returns a greeting according to the given greeting
type.</long>
         </description>
      </component>
   </group>
</mapping>
```

You have now finished creating a custom library and the .mff file which adapts it to MapForce. The custom .mff file can now be used in MapForce (see Importing the .mff File Into MapForce).

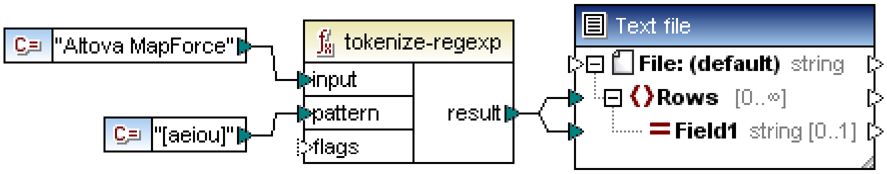
8.8 Regular Expressions

MapForce can use regular expressions in the **pattern** parameter of the **match-pattern** and **tokenize-regexp** functions, to find specific strings.

The regular expression syntax and semantics for XSLT and XQuery are identical to those defined in https://www.w3.org/TR/xmlschema-2/. Please note that there are slight differences in regular expression syntax between the various programming languages.

Terminology

input	the string that the regex works on
pattern	the regular expression
flags	optional parameter to define how the regular expression is to be interpreted
result	the result of the function



Tokenize-regexp returns a sequence of strings. The connection to the Rows item creates one row per item in the sequence.

regex syntax

Literals e.g. a single character:
e.g. The letter "a" is the most basic regex. It matches the first occurrence of the character "a" in the string.

Character classes []

This is a set of characters enclosed in square brackets.

One, and only one, of the characters in the square brackets are matched.

pattern **[aeiou]**
Matches a lowercase vowel.

pattern **[mj]ust**
Matches must or just

Please note that "pattern" is case sensitive, a lower case **a** does not match the uppercase **A.**

Character ranges **[a-z]**
Creates a range between the two characters. Only one of the characters will be matched at one time.

pattern **[a-z]**
Matches any lowercase characters between a and z.

negated classes **[^]**
using the caret as the first character after the opening bracket, negates the character class.

pattern **[^a-z]**
Matches any character not in the character class, including newlines.

Meta characters "."

Dot meta character
matches **any single** character (except for newline)

pattern **.**
Matches any single character.

Quantifiers ? + * {}

Quantifiers define how often a regex component must repeat within the input string, for a match to occur.

?
zero or one preceding string/chunk is optional

+
one or more preceding string/chunks may match one or more times

*****
zero or more preceding string/chunks may match zero or more times

{}
min / max repetitions no. of repetitions a string/chunks has to match

e.g. mo{1,3} matches mo, moo, mooo.

()
subpatterns
parentheses are used to group parts of a regex together.

|
Alternation/or allows the testing of subexpressions form left to right.
(horse|make) sense - will match "horse sense" or "make sense"

Flags

These are optional parameters that define how the regular expression is to be interpreted. Individual letters are used to set the options, i.e. the character is present. Letters may be in any order and can be repeated.

s

If present, the matching process will operate in the "dot-all" mode.

The meta character "**.**" matches any character whatsoever. If the input string contains "hello" and "world" on two *different* lines, the regular expression "hello*world" will only match if the **s** flag/ character is set.

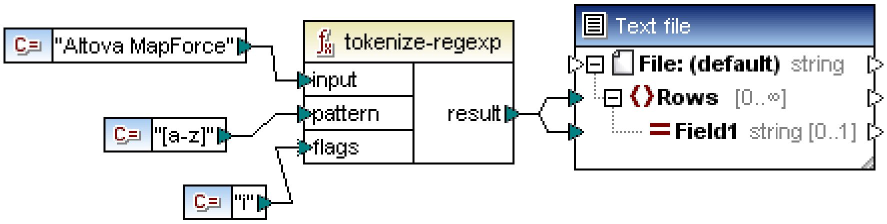
m
If present, the matching process operates in multi-line mode.

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 present, the matching process operates in case-insensitive mode.
The regular expression [a-z] plus the **i** flag would then match all letters a-z and A-Z.



x
If present, whitespace characters are removed from the regular expression prior to the matching process. Whitespace chars. are #x09, #x0A, #x0D and #x20.

Exception:
Whitespace characters within character class expressions are not removed e.g. [#x20].

Please note:
When generating code, the advanced features of the regex syntax might differ slightly between the various languages, please see the specific regex documentation for your language.

8.9 Function Library Reference

This reference chapter describes the MapForce built-in functions available in the Libraries pane, organized by library.

The availability of function libraries in the Libraries pane depends on the transformation language you have selected (see Selecting a transformation language). The **core** library is a collection of functions available in C++, C#, Java languages and in at least one of the following: XQuery, XPath, or XSLT. The **lang** library is dedicated to functions available in C++, C#, and Java languages. Other libraries contain functions associated with each separate type of output.

XPath 2.0 restrictions: Several XPath 2.0 functions dealing with sequences are currently not available.

8.9.1 core | aggregate functions

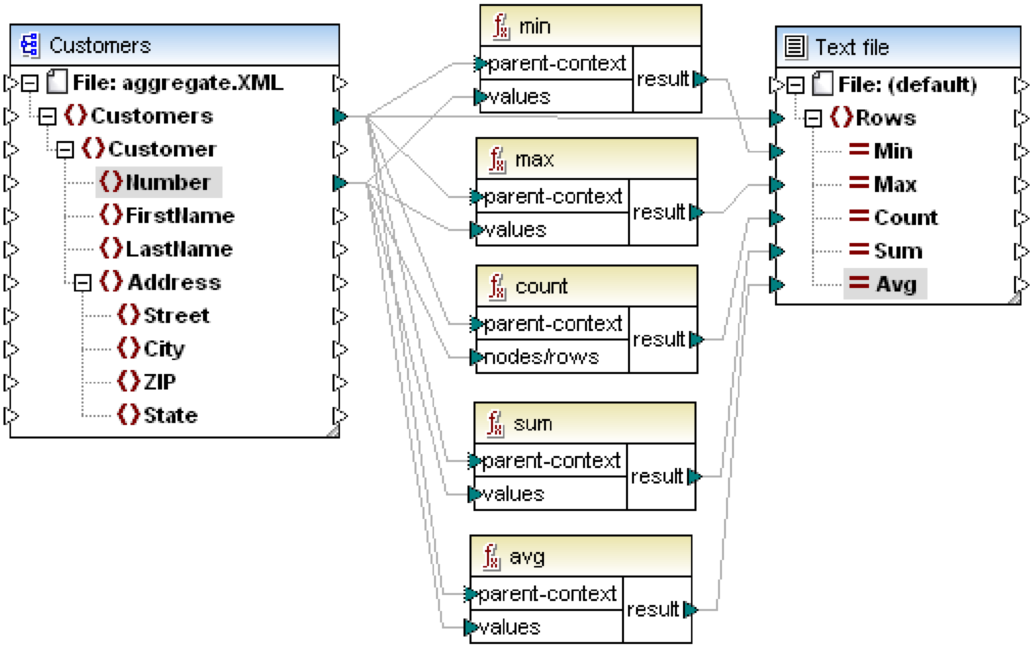
Aggregate functions perform operations on a set, or sequence, of input values. The input data for min, max, sum and avg is converted to the **decimal** data type for processing.

- The input values must be connected to the **values** parameter of the function.
- A context node (item) can be connected to the **parent-context** parameter to override the default context from which the input sequence is taken. The parent-context parameter is optional.
- The **result** of the function is connected to the specific target item.

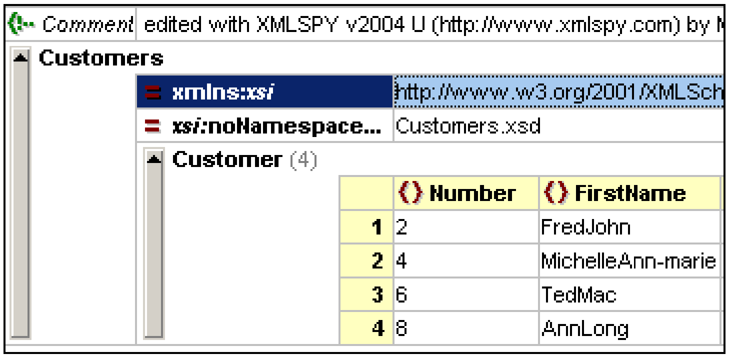
The mapping shown below is available as **Aggregates.mfd** in the ...\Tutorial folder and shows how these functions are used.

Aggregate functions have two input items.

- **values** (nodes/rows) is connected to the source item that provides the data, in this case Number.
- **parent-context** is connected to the item you want to iterate over, i.e. the context, in this case over all Customers. The parameter is, however, optional.



The input instance in this case is an XML file containing the following data:


Comment edited with XMLSPY v2004 U (http://www.xmlspy.com) by M

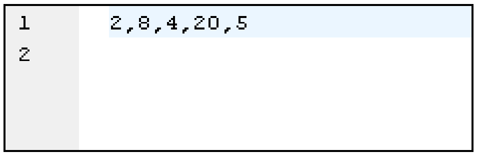
Customers

xmlns:xsi	http://www.w3.org/2001/XMLSch
xsi:noNamespace...	Customers.xsd

Customer (4)

	Number	FirstName
1	2	FredJohn
2	4	MichelleAnn-marie
3	6	TedMac
4	8	AnnLong

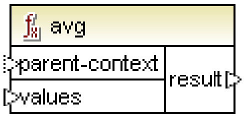
- The source data supplied to the values item is the number sequence 2,4,6,8.
- The output component in this case is a simple text file.
  Clicking the Output tab for the above mapping delivers the following result:


```
2,8,4,20,5
```

min=2, max=8, count=4, sum=20 and avg=5.

8.9.1.1 avg

Returns the average value of all values within the input sequence. The average of an empty set is an empty set. Not available in XSLT1.

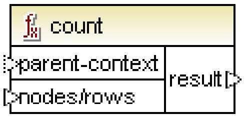


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the Mapping Context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2018\MapForceExamples\** directory.

8.9.1.2 count

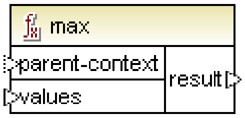
Returns the number of individual items making up the input sequence. The count of an empty set is zero. Limited functionality in XSLT1.



Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the Mapping Context.
nodes/rows	This argument must be connected to the source item to be counted.

8.9.1.3 max

Returns the maximum value of all numeric values in the input sequence. Note that this function returns an empty set if the **strings** argument is an empty set. Not available in XSLT1.



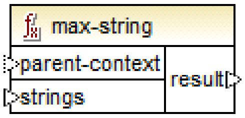
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the Mapping Context.

Argument	Description
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`** function.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2018\MapForceExamples\** directory.

8.9.1.4 *max-string*

Returns the maximum value of all string values in the input sequence. For example, `max-string("a", "b", "c")` returns "c". This function is not available in XSLT1.

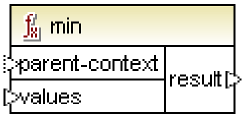


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the Mapping Context.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

Note that the function returns an empty set if the **strings** argument is an empty set.

8.9.1.5 *min*

Returns the minimum value of all numeric values in the input sequence. The minimum of an empty set is an empty set. Not available in XSLT1.

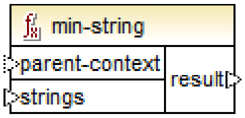


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the Mapping Context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the minimum from a sequence of strings, use the **`min-string`** function.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2018\MapForceExamples\** directory.

8.9.1.6 *min-string*

Returns the minimum value of all string values in the input sequence. For example, `min-string("a", "b", "c")` returns "a". This function is not available in XSLT1.

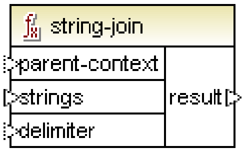


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the Mapping Context.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

Note that the function returns an empty set if the **strings** argument is an empty set.

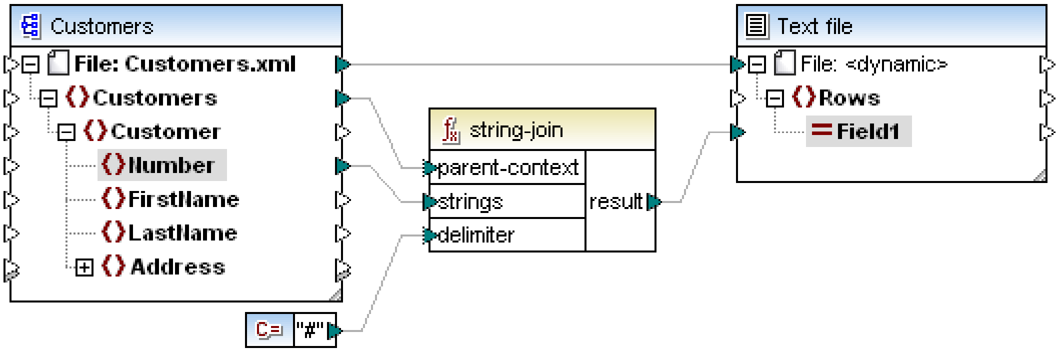
8.9.1.7 *string-join*

Concatenates all the values of the input sequence into one string delimited by whatever string you choose to use as the delimiter. The string-join of an empty set is the empty string. Not available in XSLT1.



The example below contains four separate customer numbers 2 4 6 and 8. The constant character supplies a hash character "#" as the delimiter.

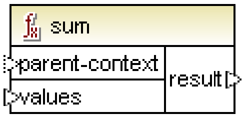
Result = 2#4#6#8



If you do not supply a delimiter, then the default is an empty string, i.e. no delimiter of any sort. Result = 2468.

8.9.1.8 sum

Returns the arithmetic sum of all values in the input sequence. The sum of an empty set is zero.



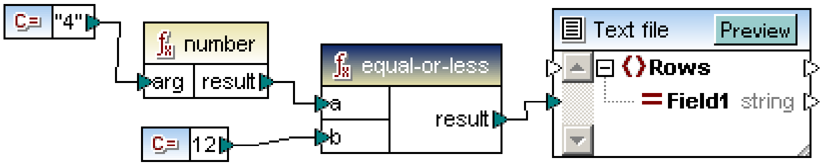
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the Mapping Context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

See also Example: Summing Node Values.

8.9.2 core | conversion functions

To support explicit data type conversion, several type conversion functions are available in the **conversion** library. Note that, in most cases, MapForce creates necessary conversions automatically and these functions need to be used only in special cases.

If the input nodes are of differing types, e. g. integer and string, you can use the conversion functions to force a string or numeric comparison.



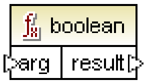
In the example above the first constant is of type string and contains the string "4".
The second constant contains the numeric constant 12. To be able to compare the two values explicitly the types must agree.

Adding a **number** function to the first constant converts the string constant to the numeric value of 4. The result of the comparisons is then "true".

Note that if the number function were not be used, i.e 4 would be connected directly to the **a** parameter, a string compare would occur, with the result being false.

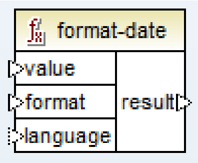
8.9.2.1 *boolean*

Converts an input numeric value into a boolean (as well as a string to numeric - true to 1). E.g. 0 to "false", or 1 to "true", for further use with logical functions (equal, greater etc.) filters, or if-else functions.



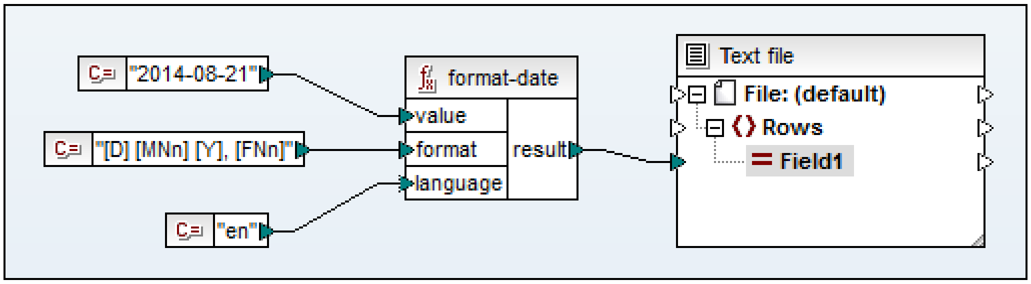
8.9.2.2 *format-date*

Converts an `xs:date` input value into a string and formats it according to specified options.



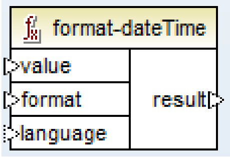
Argument	Description
value	The date to be formatted.
format	A format string identifying the way in which the date is to be formatted. This argument is used in the same way as the `format` argument in the **`format-dateTime`** function.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values: **en (default)** English **es** Spanish **de** German **ja** Japanese

In the following example, the output result is: "21 August 2014, Thursday". To translate this value to Spanish, set the value of the language argument to `es`.



8.9.2.3 format-dateTime

Converts a date and time value (`xs:dateTime`) into a string. The string representation of date and time is formatted according to the value of the `format` argument.



Argument	Description
value	The `xs:dateTime` value to be formatted.
format	A format string identifying the way in which **value** is to be formatted.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values: **en (default)** English **es** Spanish **de** German **ja** Japanese

Note: If the function's output (result) is connected to a node of type other than string, the formatting may be lost as the value is cast to the target type. This automatic cast can be disabled by unchecking the **Cast target values to target types** check box in the Component Settings of the target component (see Changing the Component Settings).

The **`format`** argument consists of a string containing so-called variable markers enclosed in square brackets. Characters outside the square brackets are literal characters to be copied into the result. 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, ...	1, 2, 3
01	decimal format, two digits: 01, 02, 03, ...	01, 02, 03
N	name of component, upper case	MONDAY, TUESDAY [1)]
n	name of component, lower case	monday, tuesday [1)]
Nn	name of component, title case	Monday, Tuesday [1)]

Note: N, n, and Nn modifiers only support the following components: M, d, D.

The width modifier, if present, is introduced by a comma. It takes the form:

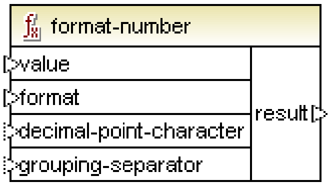
```
, min-width ("-" max-width)?
```

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

8.9.2.4 *format-number*

Converts a number into a string. The function is available for XSLT 1.0, XSLT 2.0, Java, C#, C++ and Built-in execution engine.



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. This argument is used in the same way as the `format` argument in the **`format-`**

Argument	Description
	`dateTime` function.
`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 (i.e. result) is connected to a node of type other than string, the formatting may be lost as the value is cast to the target type. This automatic cast can be disabled by unchecking the **Cast target values to target types** check box in the component settings of the target component.

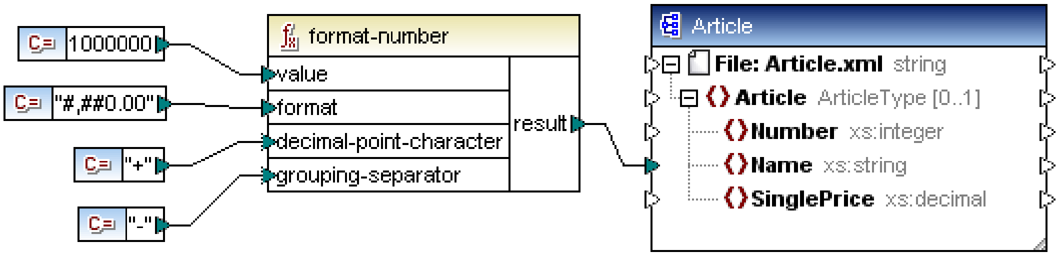
Format:

```
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*.

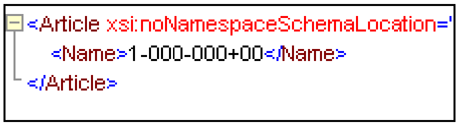
Special Character	default	Description
zero-digit	0	A digit will always appear at this point in the result
digit	#	A digit will appear at this point in the result string unless it is a redundant leading or trailing zero
decimal-point	.	Separates the integer and the fraction part of the number.
grouping-separator	,	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 characters used for decimal-point-character and grouping-separator are always "." and "," respectively. They can, however, be changed in the formatted output, by mapping constants to these nodes.



The result of the format number function shown above.

- The decimal-point character was changed to a "+".
- The grouping separator was changed to a "-"



```
<Article xsi:noNamespaceSchemaLocation='
    <Name>1-000-000+00</Name>
</Article>
```

Rounding

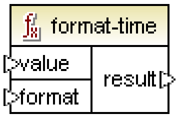
The rounding method used for this function is "half up", e.g. the value gets rounded up if the fraction is greater than or equal to 0.5. The value gets rounded down if the fraction is less than 0.5. This method of rounding only applies to generated code and the built-in execution engine.

In XSLT 1.0, the rounding mode is undefined. In XSLT 2.0, the rounding mode is "round-half-to-even".

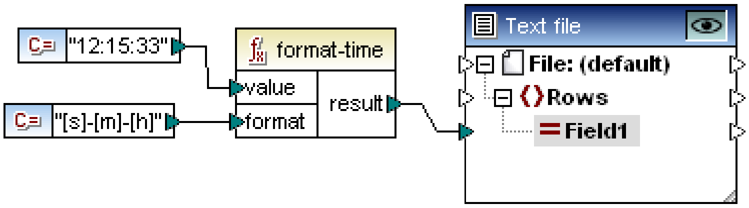
Number	Format String	Result
1234.5	#,##0.00	1,234.50
123.456	#,##0.00	123.46
1000000	#,##0.00	1,000,000.00
-59	#,##0.00	-59.00
1234	###0.0###	1234.0
1234.5	###0.0###	1234.5
.00025	###0.0###	0.0003
.00035	###0.0###	0.0004
0.25	#00%	25%
0.736	#00%	74%
1	#00%	100%
-42	#00%	-4200%
-3.12	#.00;(#.00)	(3.12)
-3.12	#.00;#.00CR	3.12CR

8.9.2.5 format-time

Converts an xs:time input value into a string. The format argument is used in the same way as the format argument in the **format-dateTime** function.



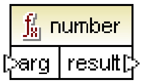
E.g



Result: 33-15-12

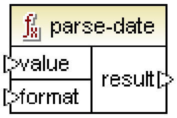
8.9.2.6 number

Converts an input string into a number. Also converts a boolean input to a number.



8.9.2.7 parse-date

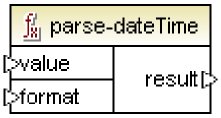
Available for Java, C#, C++, and the Built-in execution engine.



Converts a string into a date, while ignoring the time component. This function uses the parse-dateTime function as a basis, while ignoring the time component. The result is of type xs:date.

8.9.2.8 parse-dateTime

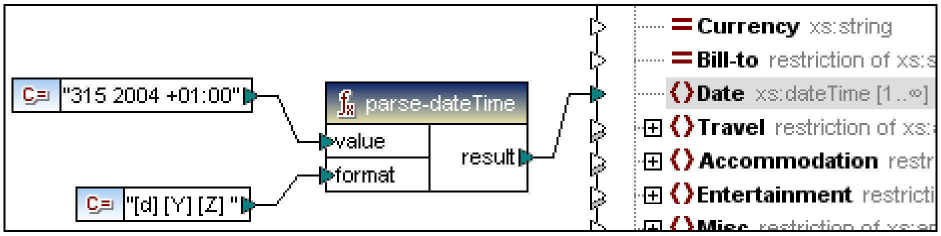
Available for Java, C#, C++, and the Built-in execution engine.



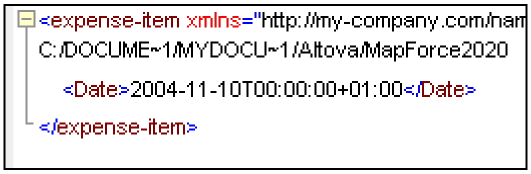
Converts a date/time value expressed as a string into a value of type `dateTime`. This function takes the following arguments:

Argument	Description
value	The string value to be converted.
format	Specifies the format mask to apply to *value*.

For example, in the mapping below, the string value **`315 2004 +01:00`** specifies the 315th day of year 2004, in the time zone GMT+01:00. This value is converted into its `dateTime` equivalent, by applying the format mask **`[d] [Y] [Z]`**.



The result is as follows:



```
<expense-item xmlns="http://my-company.com/nam
C:/DOCUME~1/MYDOCU~1/Altova/MapForce2020
    <Date>2004-11-10T00:00:00+01:00</Date>
</expense-item>
```

A format mask can consist of the following components:

Component	Description	Default Presentation
Y	year (absolute value)	four digits (2010)
M	month of the year	1-12
D	day of month	1-31

Componen t	Description	Default Presentation
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

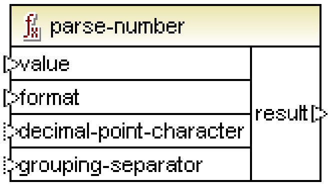
Note: *N*, *n*, and *Nn* modifiers support only the component *M* (month).

The table below lists a few more 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`

8.9.2.9 *parse-number*

Available for Java, C#, C++, and the Built-in execution engine.

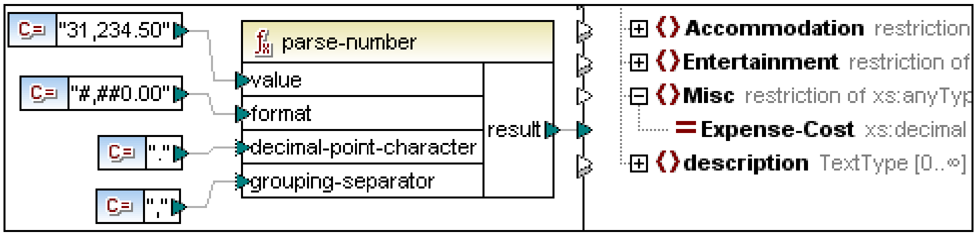


Converts an input string into a decimal number.

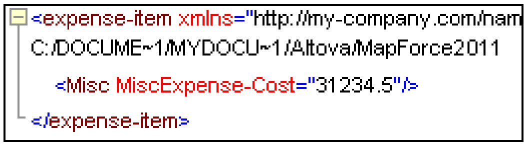
Argument	Description
value	The string to be parsed/converted to a number
format	A format string that identifies the way in which the number is currently formatted (optional). Default is "#,##0.#"
decimal-point-character	The character to be used as the decimal point character. Default is the '**.**' character (optional)
grouping-separator	The separator/delimiter used to separate groups of numbers. Default is the "**,**" character (optional)

The ***format*** string used in parse-number is the same as that used in format-number.

Example in MapForce:



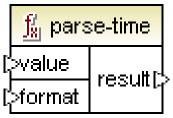
Result:



```
<expense-item xmlns="http://my-company.com/nam
C:/DOCUME~1/MYDOCU~1/Altova/MapForce2011
    <Misc MiscExpense-Cost="31234.5"/>
</expense-item>
```

8.9.2.10 *parse-time*

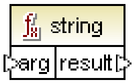
Available for Java, C#, C++, and the Built-in execution engine.



Converts a string into a time, while ignoring the date component. This function uses the parse-dateTime function as a basis, while ignoring the date component. The result is of type xs:time.

8.9.2.11 string

Converts an input value into a string. The function can also be used to retrieve the text content of a node.



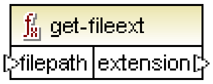
If the input node is a XML complex type, then all descendents are also output as a single string.

8.9.3 core | file path functions

The **file path** functions allow you to directly access and manipulate file path data, i.e. folders, file names, and extensions for further processing in your mappings. They can be used in all languages supported by MapForce.

8.9.3.1 get-fileext

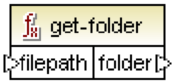
Returns the extension of the file path including the dot "." character.



E.g. 'c:\data\Sample.mfd' returns '.mfd'

8.9.3.2 get-folder

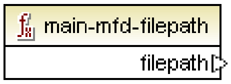
Returns the folder name of the file path including the trailing slash, or backslash character.



E.g. 'c:/data/Sample.mfd' returns 'c:/data/'

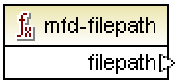
8.9.3.3 *main-mfd-filepath*

Returns the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.



8.9.3.4 *mfd-filepath*

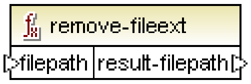
If the function is called in the main mapping, it returns the same as main-mfd-filepath function, i.e. the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.



If called within an **user-defined function** which is **imported** by a mfd-file, it returns the full path of the imported mfd file which contains the **definition** of the user-defined function.

8.9.3.5 *remove-fileext*

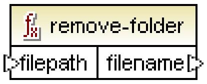
Removes the extension of the file path including the dot-character.



E.g. 'c:/data/Sample.mfd' returns 'c:/data/Sample'.

8.9.3.6 *remove-folder*

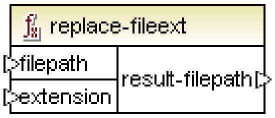
Removes the directory of the file path including the trailing slash, or backslash character.



E.g. 'c:/data/Sample.mfd' returns 'Sample.mfd'.

8.9.3.7 *replace-fileext*

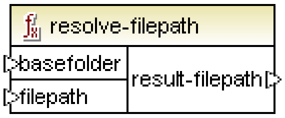
Replaces the extension of the file path supplied by the filepath parameter, with the one supplied by the connection to the extension parameter.



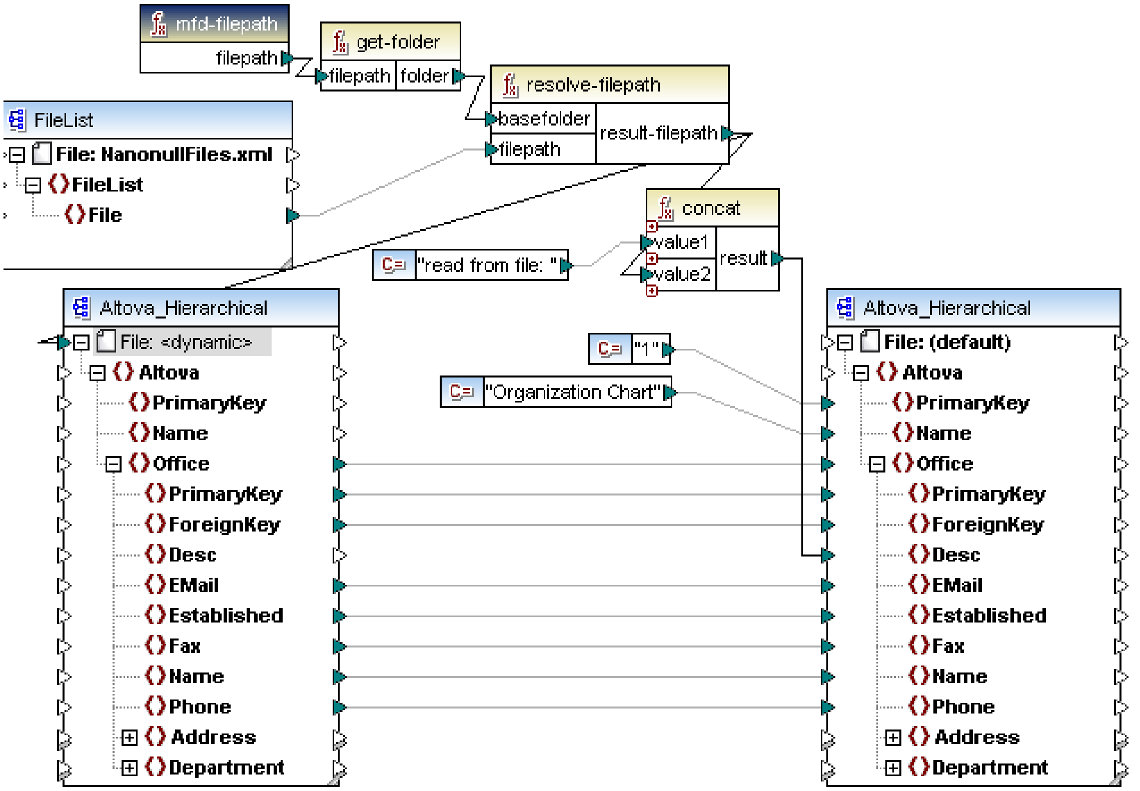
E.g. c:/data/Sample.**mfd'** as the input filepath, and '**.mfp'** as the extension, returns 'c:/data/Sample.mfp'

8.9.3.8 *resolve-filepath*

Resolves a relative file path to a relative, or absolute, base folder. The function supports '.' (current directory) and '..' (parent directory).



For an example, see the mapping **MergeMultipleFiles_List.mfd** available in the ... \MapForceExamples folder.

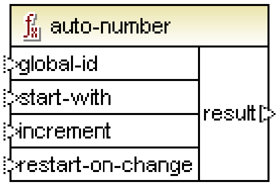


8.9.4 core | generator functions

The **core / generator** functions library includes functions which generate values.

8.9.4.1 *auto-number*

The **auto-number** function generates integers in target nodes of a component, depending on the various parameters you define. The function result is a value starting at **start_with** and increased by **increment**. Default values are: start-with=1 and increase=1. Both parameters can be negative.

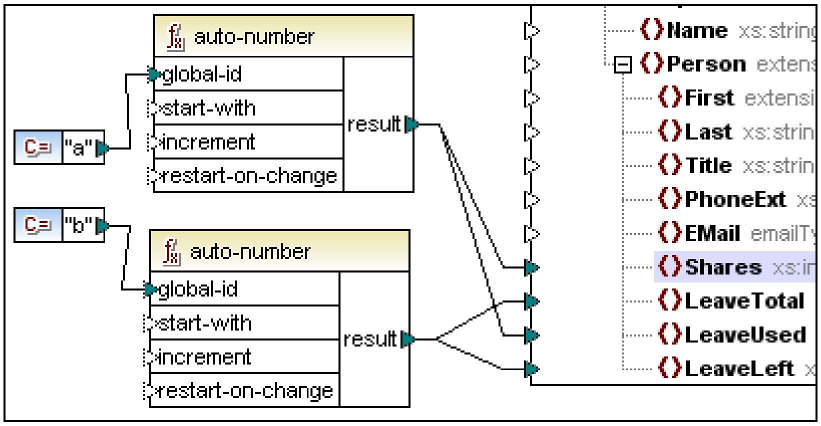


Make sure that the result connector (of the auto-number function) is **directly** connected to a target node. The exact order in which functions are called by the generated mapping code is undefined. MapForce may choose to cache calculated results for reuse, or evaluate expressions in any order. It is therefore strongly recommended to take care when using the auto-number function.

global-id
This parameter allows you to synchronize the number sequence output of two separate auto-number functions connected to a single target component.

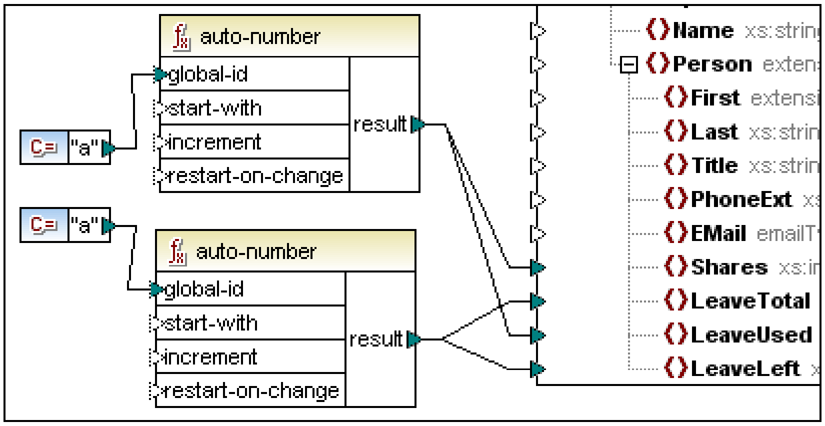
If the two auto-number functions do **not** have the same global-id, then each increments the target items separately. In the example below, each function has a different global-id i.e. a and b.

The output of the mapping is 1,1,2,2. The top function supplies the first 1 and the lower one the second 1.



If both functions have **identical** global-ids, **a** in this case, then each function "knows" about the current auto-number state (or actual value) of the other, and both numbers are then synchronised/ in sequence.

The output of the mapping is therefore 1, 2, 3, 4.The top function supplies the first 1 and the lower one now supplies a 2.

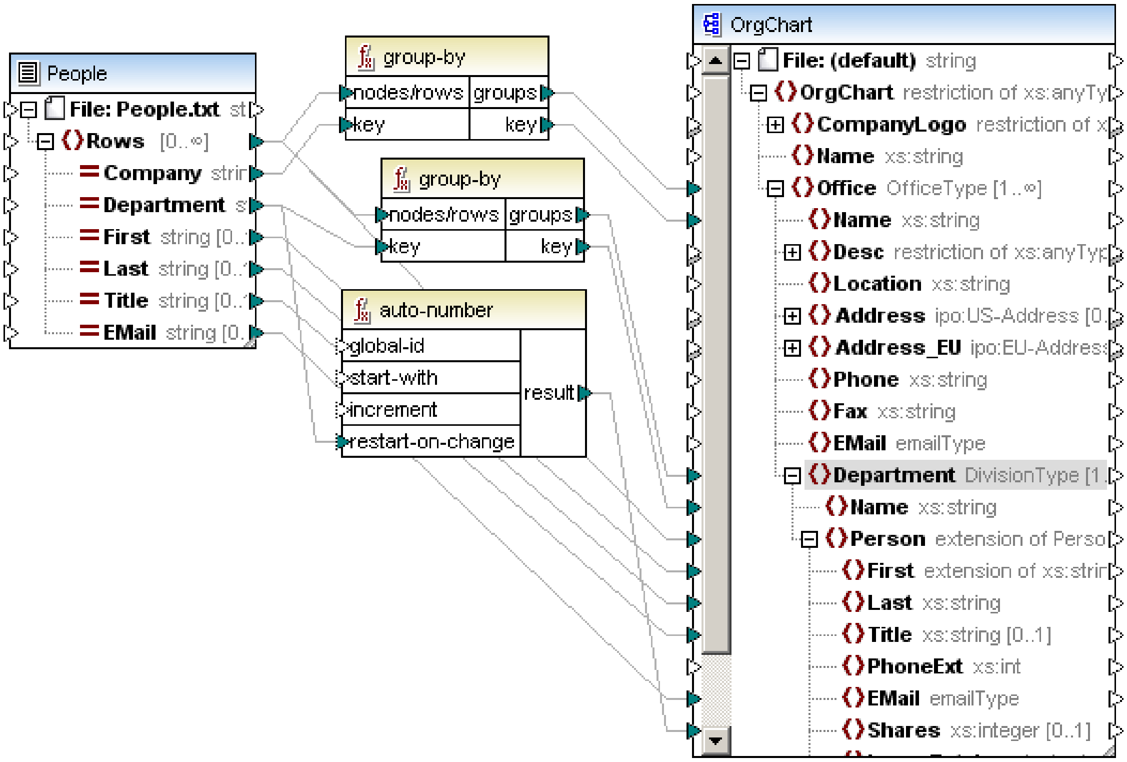


start-with
The initial value used to start the auto numbering sequence. Default is 1.

increment
The increment you want auto-number sequence to increase by. Default is 1.

restart on change
Resets the auto-number counter to "start-with", when the **content** of the connected item changes.

In the example below, start-with and increment are both using the default 1. As soon as the **content** of Department changes, i.e. the department name changes, the counter is reset and starts at 1 for each new department.



8.9.5 core | logical functions

Logical functions are (generally) used to compare input data with the result being a boolean "true" or " false". They are generally used to test data before passing on a subset to the target component using a filter.

input parameters = **a** | **b**, or **value1** | **value2**
output parameter = result

The evaluation result of two input nodes 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 strings contain '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 data types are of the same type, e.g. all input nodes are numerical types, or strings, then the comparison is done for the common type.

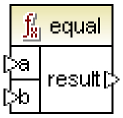
If the input nodes are of differing types (for example, integer and string, or string and date), then the data type used for the comparison **is the most general (least restrictive) input data type** of the two input types.

Before comparing two values, all input values are converted to a common datatype. Using the previous example; the datatype "string" is less restrictive than "integer". Comparing integer value 4 with the string '12', converts 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 argument to a logical function, it returns a null value. For more information about handling null values, see Nil Values / Nillable.

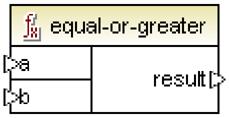
8.9.5.1 equal

Result is true if a=b, else false.



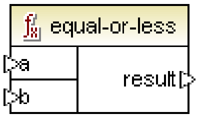
8.9.5.2 equal-or-greater

Result is true if a is equal/greater than b, else false.



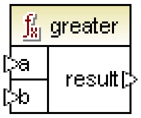
8.9.5.3 equal-or-less

Result is true if a is equal/less than b, else false.



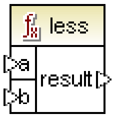
8.9.5.4 greater

Result is true if a is greater than b, else false.



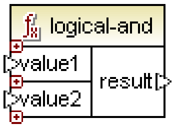
8.9.5.5 less

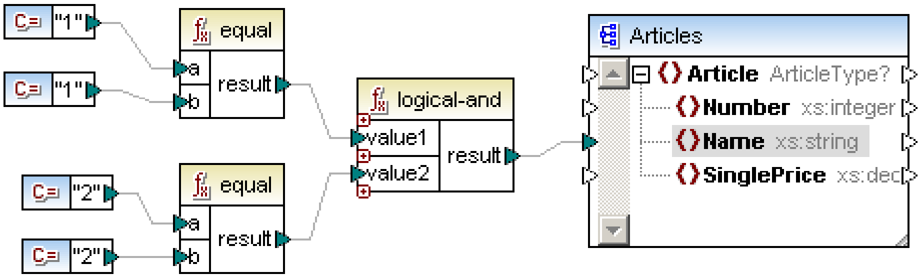
Result is true if a is less than b, else false.



8.9.5.6 logical-and

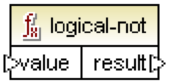
If **both** value1 and value2 of the logical-and function are **true**, then result is true; if different then false.



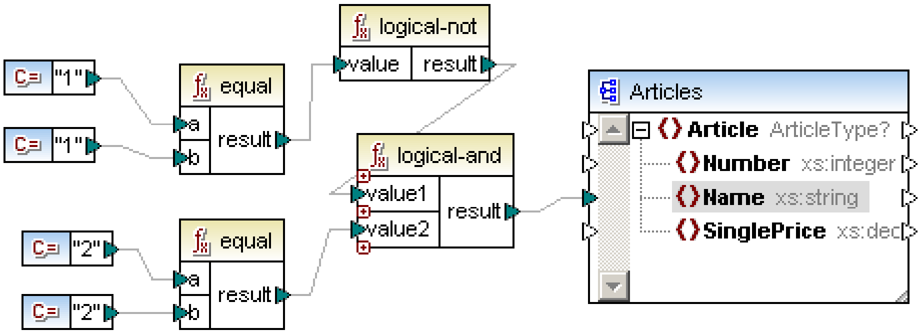


8.9.5.7 logical-not

Inverts or flips the logical state/result; if input is true, result of logical-not function is false. If input is false then result is true.

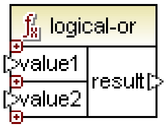


The logical-not function shown below, inverts the result of the equal function. The logical-and function now only returns true if boolean values of value1 and value2 are different, i.e. true-false, or false-true.



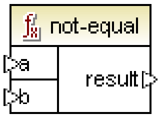
8.9.5.8 logical-or

Requires both input values to be boolean. If **either** value1 or value2 of the logical-or function are **true**, then the result is true. If both values are false, then result is false.



8.9.5.9 not-equal

Result is true if a is not equal to b.

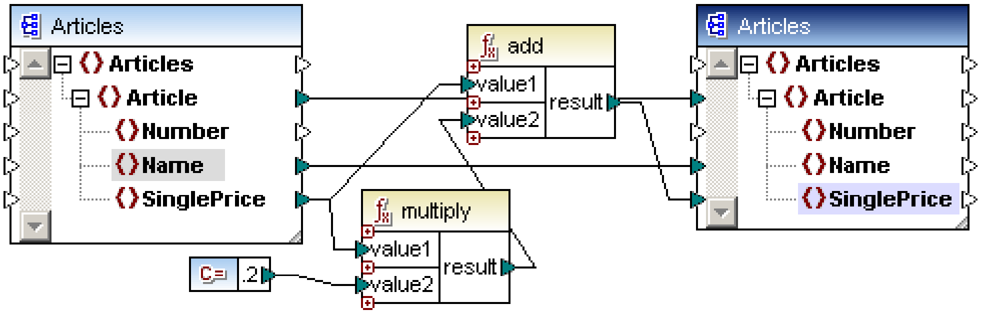


8.9.6 core | math functions

Math functions are used to perform basic mathematical functions on data. Note that they cannot be used to perform computations on durations, or datetimes.

input parameters = **value1** | **value2**
output parameter = **result**

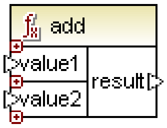
input values are automatically converted to decimal for further processing.



The example shown above, adds 20% sales tax to each of the articles mapped to the target component.

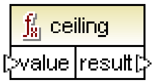
8.9.6.1 *add*

Result is the decimal value of adding **value1** to **value2**.



8.9.6.2 *ceiling*

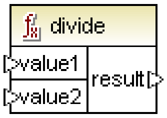
Result is the smallest integer that is greater than or equal to **value**, i.e. the next highest integer value of the decimal input **value**.



E.g. if the result of a division function is 11.2, then applying the ceiling function to it makes the result 12, i.e. the next highest whole number.

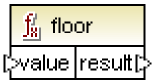
8.9.6.3 divide

Result is the decimal value of dividing **value1** by **value2**. The result precision depends on the target language. Use the round-precision function to define the precision of result.



8.9.6.4 floor

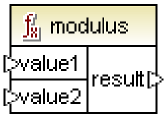
Result is the largest integer that is less than or equal to **value**, i.e. the next lowest integer value of the decimal input **value**.



E.g. if the result of a division function is 11.2, then applying the floor function to it makes the result 11, i.e. the next lowest whole number.

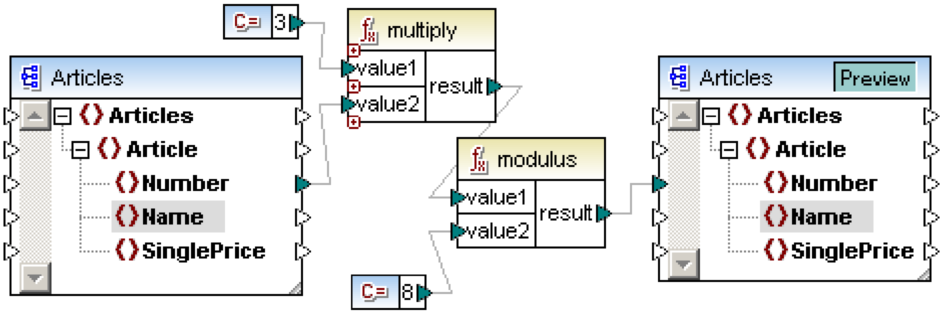
8.9.6.5 modulus

Result is the remainder of dividing **value1** by **value2**.



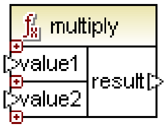
In the mapping below, the numbers have been multiplied by 3 and passed on to value1 of the modulus function. Input values are now 3, 6, 9, and 12.

When applying/using modulus 8 as value2, the remainders are 3, 6, 1, and 4.



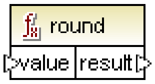
8.9.6.6 multiply

Result is the decimal value of multiplying **value1** by **value2**.



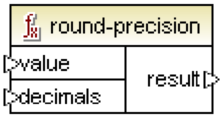
8.9.6.7 round

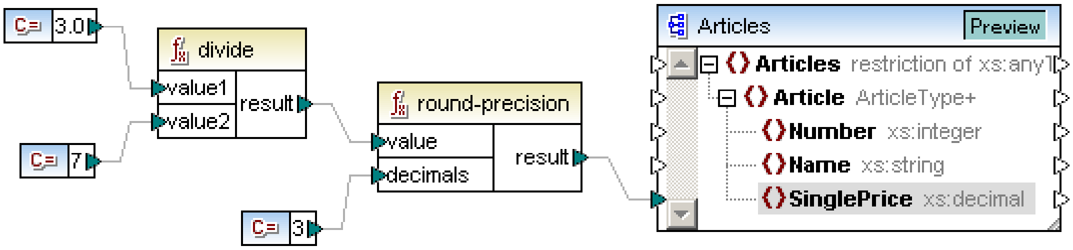
Returns the value rounded to the nearest integer. When the value is exactly in between two integers, the "Round Half Towards Positive Infinity" algorithm is used. For example, the value "10.5" gets rounded to "11", and the value "-10.5" gets rounded to "-10".



8.9.6.8 round-precision

Result is the decimal value of the number rounded to the decimal places defined by "decimals".

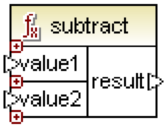




In the mapping above, the result is 0.429. For the result to appear correctly in an XML file, make sure to map it to an element of xs:decimal type.

8.9.6.9 subtract

Result is the decimal value of subtracting **value2** from **value1**.

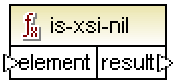


8.9.7 core | node functions

The node functions allow you to access nodes, or process nodes in a particular way.

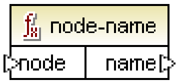
8.9.7.1 is-xsi-nil

Returns true (<OrderID>true</OrderID>) if the element node, of the source component, has the xsi:nil attribute set to "true".



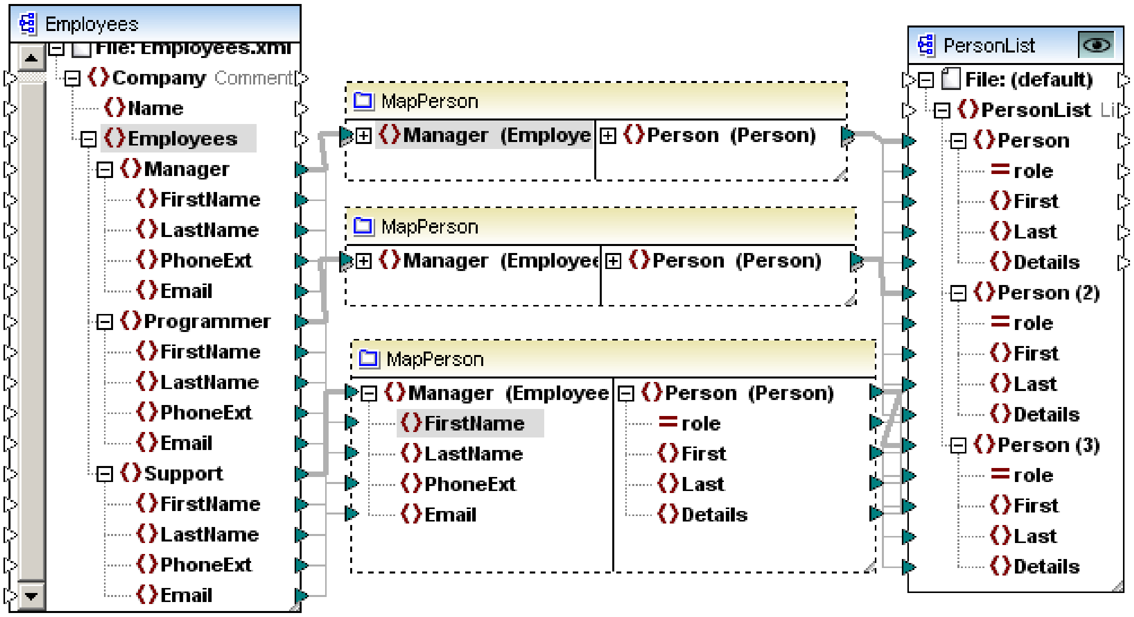
8.9.7.2 node-name

Returns the qualified name (QName) of the connected node. If the node is an XML **text()** node, an empty QName is returned. This function only works on those nodes that have a name. If XSLT is the target language (which calls fn:node-name), the function returns an empty sequence for nodes which have no names.

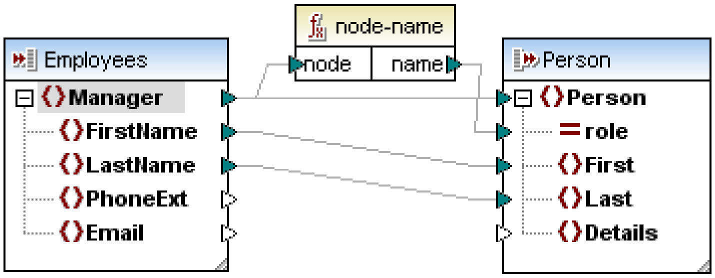


- Getting a name from database tables/fields is not supported.
- XBRL and Excel are not supported.
- Getting a name of File input node is not supported.

- WebService nodes behave like XML nodes except that:
  - node-name from "part" is not supported.
  - node-name from root node ("Output" or "Input") is not supported.



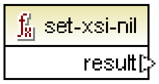
The MapPerson user-defined function uses **node-name** to return the name of the input **node**, and place it in the role attribute. The root node of the Employees.xsd, in the user-defined function, has been defined as "Manager".



Manager gets its data from **outside** the user-defined function, where it can be either: Manager, Programmer, or Support. This is the data that is then passed on to the role attribute in PersonList.

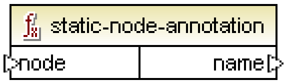
8.9.7.3 set-xsi-nil

Sets the target node to xsi:nil.



8.9.7.4 static-node-annotation

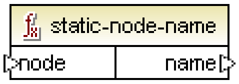
Returns the string with annotation of the connected node. The input must be: (i) a source component node, or (ii) an inline function that is directly connected to a parameter, which in turn is directly connected to a node in the calling mapping.



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.

8.9.7.5 static-node-name

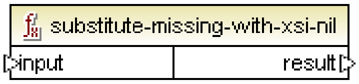
Returns the string with the name of the connected node. The input must be: (i) a source component node, or (ii) an inline function that is directly connected to a parameter, which in turn is directly connected to a node in the calling mapping.



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.

8.9.7.6 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.

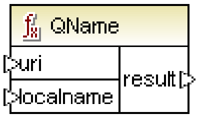


8.9.8 core | QName functions

QName functions provide ways to manipulate the Qualified Names (QName) in XML documents.

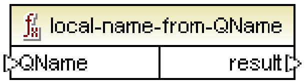
8.9.8.1 QName

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 localname parameters can be supplied by a constant function.



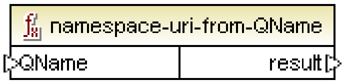
8.9.8.2 local-name-from-QName

Returns the local name part of the QName.



8.9.8.3 namespace-uri-from-QName

Returns the namespace URI part of the QName.



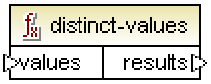
8.9.9 core | sequence functions

Sequence functions allow processing of input sequences and grouping of their content. The value/ content of the **key** input parameter, mapped to nodes/rows, is used to group the sequence.

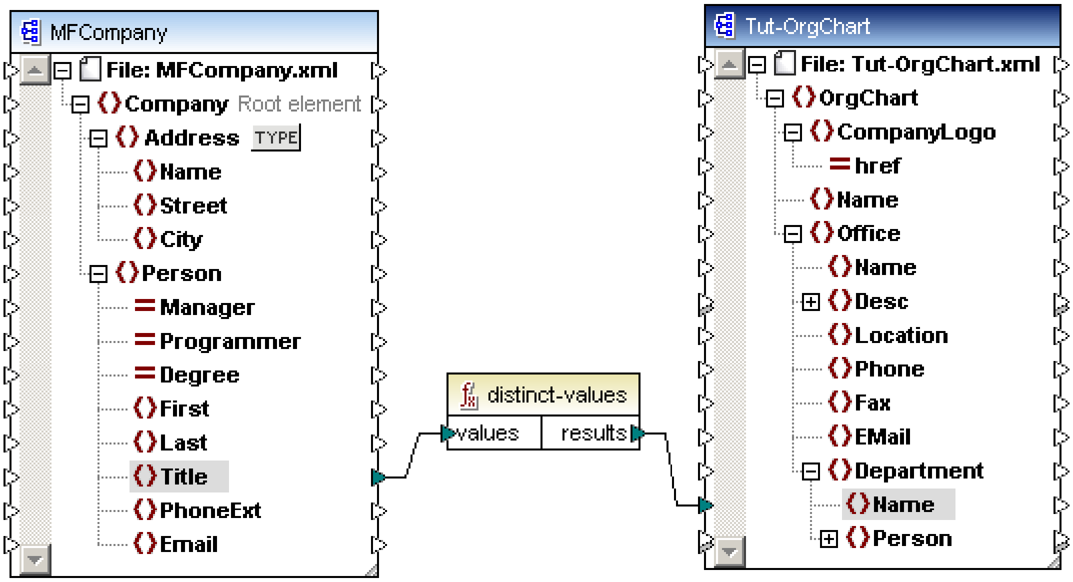
- Input parameter **key** is of an arbitrary data type that can be converted to string for **group-adjacent** and **group-by**
- Input parameter **bool** is of type Boolean for **group-starting-with** and **group-ending-with**
- The output **key** is the key of the current group.

8.9.9.1 distinct-values

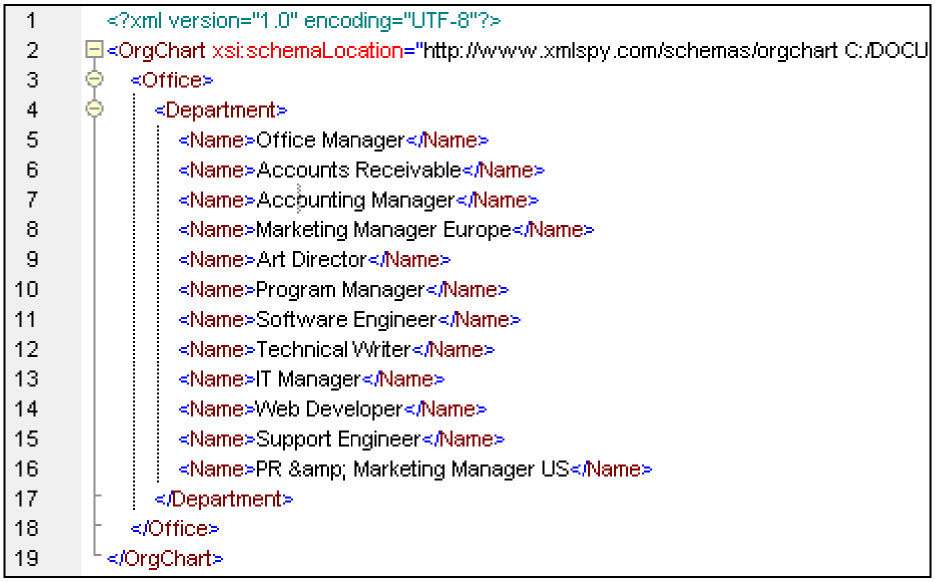
Allows you to remove duplicate values from a sequence and map the unique items to the target component.



In the example below, the content of the source component "Title" items, are scanned and each unique title is mapped to the Department / Name item in the target component.



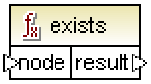
Note that the sequence of the individual Title items in the source component are retained when mapped to the target component.



```
<?xml version="1.0" encoding="UTF-8"?>
<OrgChart xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart C:/DOCU
  <Office>
    <Department>
      <Name>Office Manager</Name>
      <Name>Accounts Receivable</Name>
      <Name>Accounting Manager</Name>
      <Name>Marketing Manager Europe</Name>
      <Name>Art Director</Name>
      <Name>Program Manager</Name>
      <Name>Software Engineer</Name>
      <Name>Technical Writer</Name>
      <Name>IT Manager</Name>
      <Name>Web Developer</Name>
      <Name>Support Engineer</Name>
      <Name>PR &amp; Marketing Manager US</Name>
    </Department>
  </Office>
</OrgChart>
```

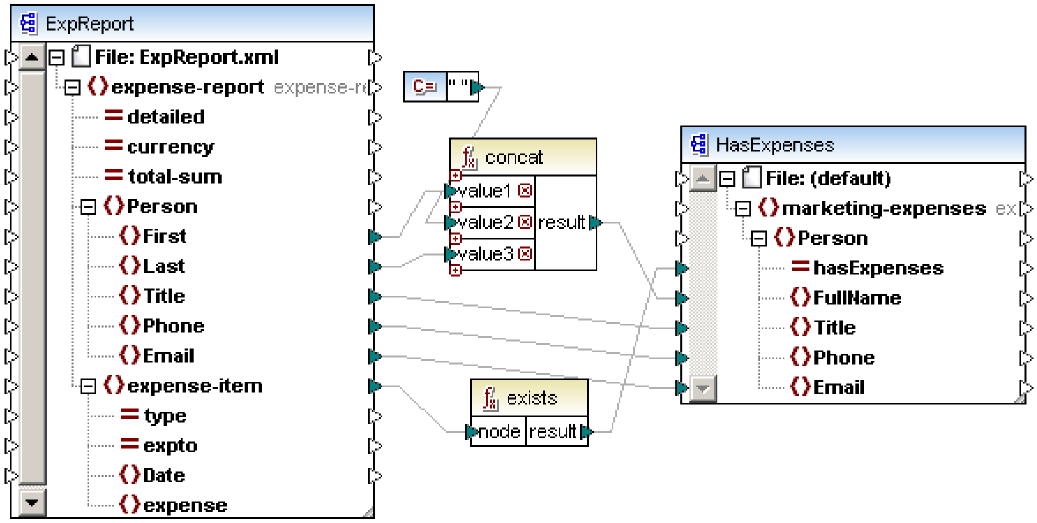
8.9.9.2 exists

Returns true if the node exists, else returns false.



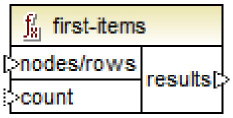
The "**HasMarketingExpenses.mfd**" file in the ...\MapForceExamples folder contains the small example shown below.

If an expense-item exists in the source XML, then the "hasExpenses" attribute is set to "true" in the target XML/Schema file.



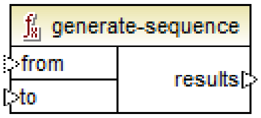
8.9.9.3 first-items

Returns the first "X" items of the input sequence, where X is the number supplied by the "count" parameter. E.g. if the value 3 is mapped to the count parameter and a parent node to the nodes/row parameter, then the first three items will be listed in the output.



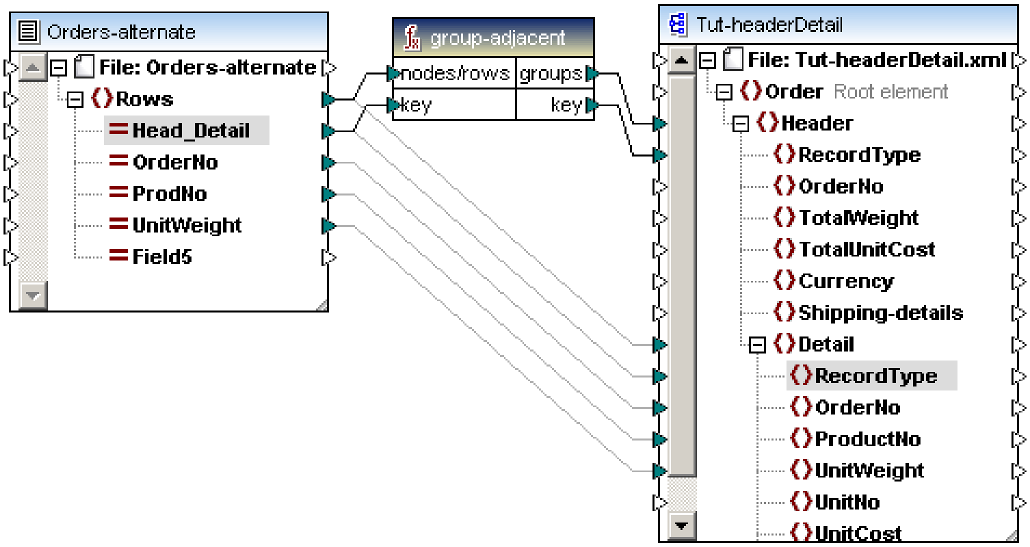
8.9.9.4 generate-sequence

Creates a sequence of integers using the "from" and "to" parameters as the boundaries.

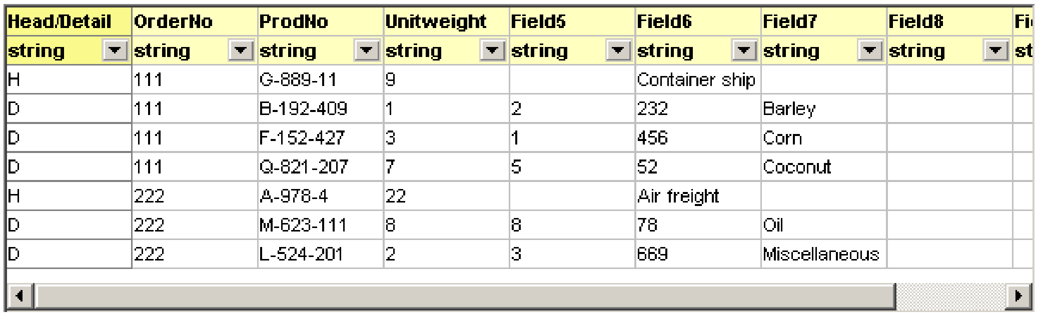


8.9.9.5 group-adjacent

Groups the input sequence **nodes/rows** into groups of adjacent items sharing the same **key**. Note that group-adjacent uses the **content** of the node/item as the grouping key.

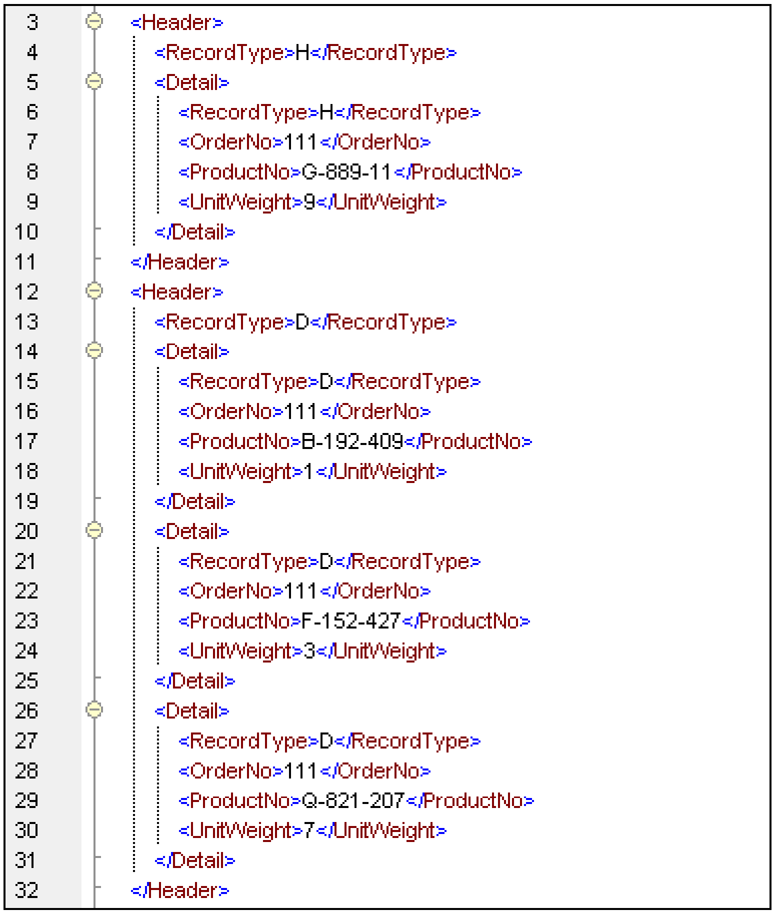


Given the CSV file shown below, what we want to happen is to have all the Header and Detail records in their own groups.



Head/Detail	OrderNo	ProdNo	Unitweight	Field5	Field6	Field7	Field8	Fi
string	string	string	string	string	string	string	string	st
H	111	G-889-11	9		Container ship			
D	111	B-192-409	1	2	232	Barley		
D	111	F-152-427	3	1	456	Corn		
D	111	Q-821-207	7	5	52	Coconut		
H	222	A-978-4	22		Air freight			
D	222	M-623-111	8	8	78	Oil		
D	222	L-524-201	2	3	669	Miscellaneous		

- A new group is started with the first element, in this case H.
- As the next element (or key) in the sequence is different, i.e. D, this starts a second group called D.
- The next two D elements are now added to the same group D, as they are of the same type.
- A new H group is started with a single H element.
- Followed by a new D group containing two D elements.

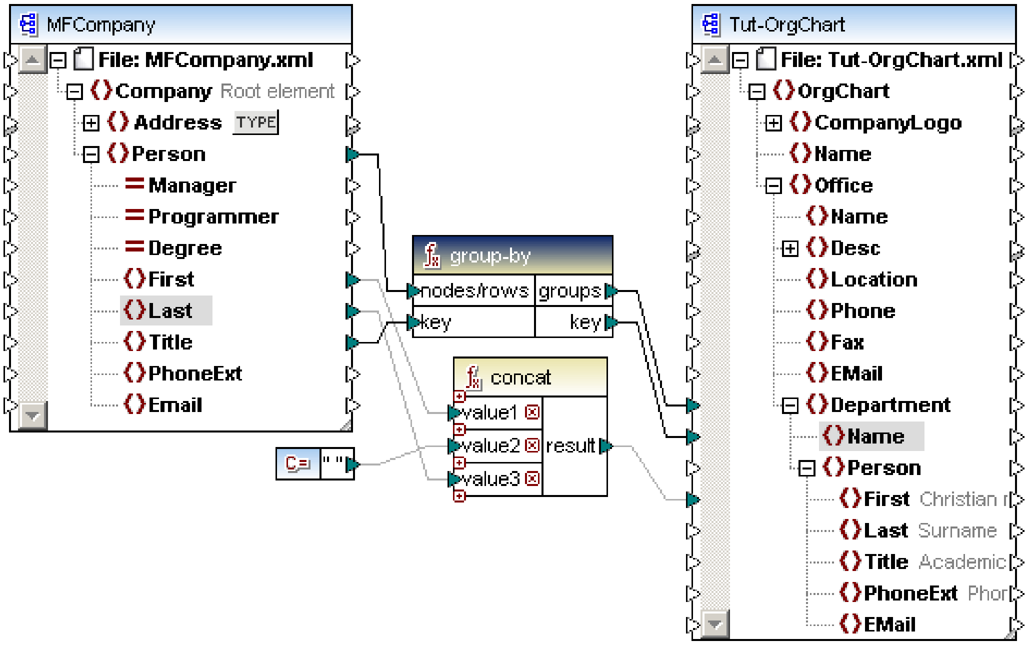


```
<Header>
  <RecordType>H</RecordType>
  <Detail>
    <RecordType>H</RecordType>
    <OrderNo>111</OrderNo>
    <ProductNo>G-889-11</ProductNo>
    <UnitWeight>9</UnitWeight>
  </Detail>
</Header>
<Header>
  <RecordType>D</RecordType>
  <Detail>
    <RecordType>D</RecordType>
    <OrderNo>111</OrderNo>
    <ProductNo>B-192-409</ProductNo>
    <UnitWeight>1</UnitWeight>
  </Detail>
  <Detail>
    <RecordType>D</RecordType>
    <OrderNo>111</OrderNo>
    <ProductNo>F-152-427</ProductNo>
    <UnitWeight>3</UnitWeight>
  </Detail>
  <Detail>
    <RecordType>D</RecordType>
    <OrderNo>111</OrderNo>
    <ProductNo>Q-821-207</ProductNo>
    <UnitWeight>7</UnitWeight>
  </Detail>
</Header>
```

8.9.9.6 *group-by*

Groups the input sequence **nodes/rows** into groups of not necessarily adjacent items sharing the same **key**. Groups are output in the order the key occurs in the input sequence. The example below shows how this works:

- The key that defines the specific groups of the source component is the Title item. This is used to group the persons of the company.
- The group name is placed in the Department/Name item of the target component, while the concatenated person's first and last names are placed in the Person/First child item.

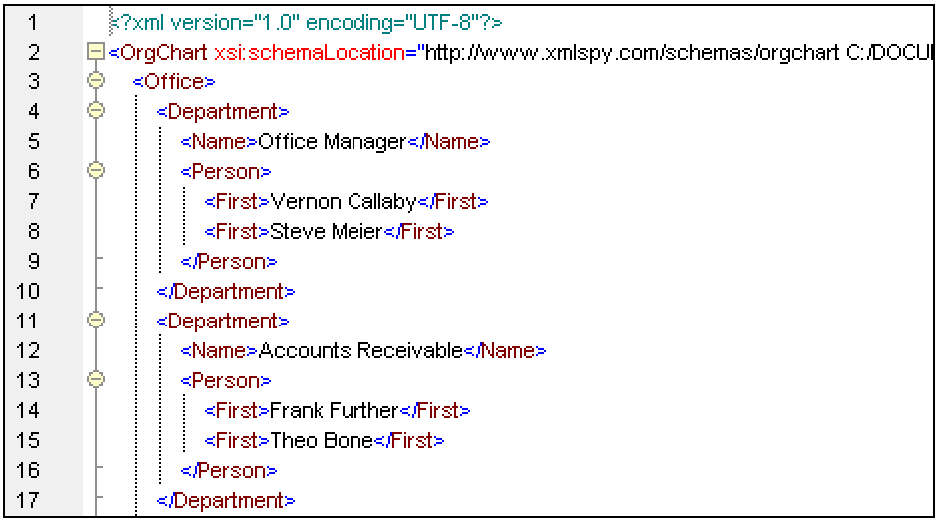


Note that group-by uses the **content** of the node/item as the grouping key. The content of the Title field is used to group the persons and is mapped to the Department/Name item in the target.

Note also: there is an **implied filter** of the rows from the source document to the target document, which can be seen in the included example. In the target document, each Department item only has those Person items that **match** the grouping **key**, as the group-by component creates the necessary hierarchy on the fly.

If you have a flat hierarchy (CSV, FLF, etc) with a dynamic output file name, built in part from the key value, the implied filter still exists. This means that you may not need to connect the 'groups' output to any item in the target component.

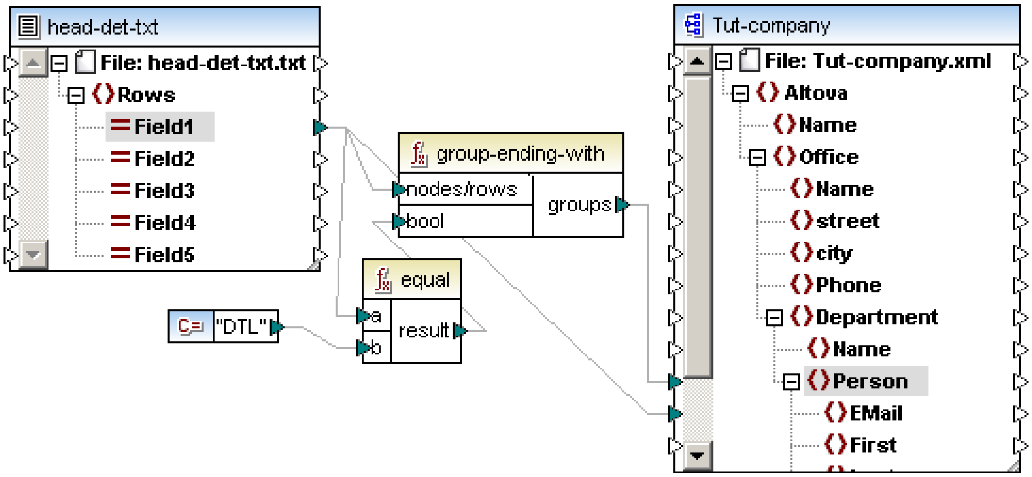
Clicking the Output button shows the result of the grouping process.



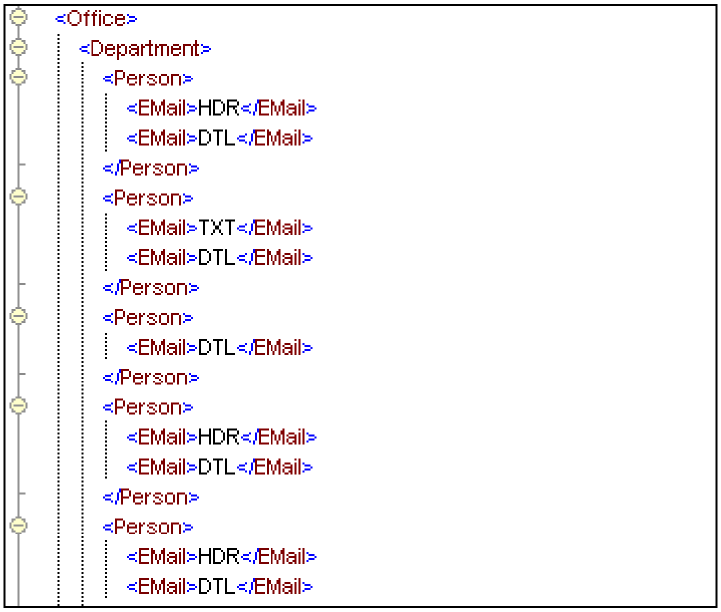
```
<?xml version="1.0" encoding="UTF-8"?>
<OrgChart xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart C:/DOCU
  <Office>
    <Department>
      <Name>Office Manager</Name>
      <Person>
        <First>Vernon Callaby</First>
        <First>Steve Meier</First>
      </Person>
    </Department>
    <Department>
      <Name>Accounts Receivable</Name>
      <Person>
        <First>Frank Further</First>
        <First>Theo Bone</First>
      </Person>
    </Department>
```

8.9.9.7 *group-ending-with*

This function groups the input sequence **nodes/rows** into groups, ending a new group whenever **bool** is true. This example shows the result when using "DTL" as the group-ending-with item.



In this case the **value** of the item/nodes do not need to be identical or even exist. The node "**pattern**" i.e. the node/item names need to be identical for the grouping to occur.

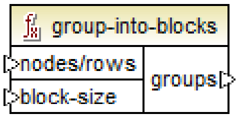


```
<Office>
  <Department>
    <Person>
      <EMail>HDR</EMail>
      <EMail>DTL</EMail>
    </Person>
    <Person>
      <EMail>TXT</EMail>
      <EMail>DTL</EMail>
    </Person>
    <Person>
      <EMail>DTL</EMail>
    </Person>
    <Person>
      <EMail>HDR</EMail>
      <EMail>DTL</EMail>
    </Person>
    <Person>
      <EMail>HDR</EMail>
      <EMail>DTL</EMail>
```

The result above shows that a new group was started wherever "DTL" can be the last element.

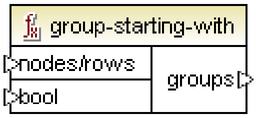
8.9.9.8 *group-into-blocks*

Groups the input sequence **nodes/rows** into blocks of the same size defined by the number supplied by the **block-size** parameter.

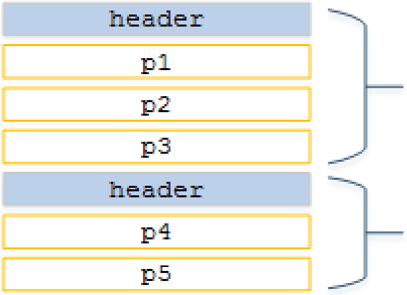


8.9.9.9 *group-starting-with*

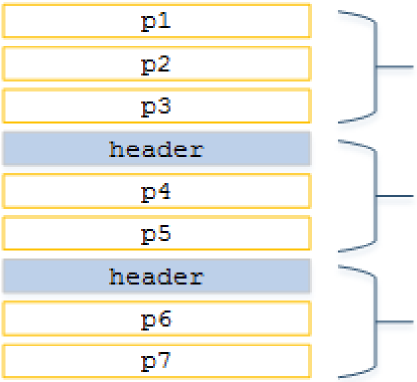
This function groups the input sequence **nodes/rows** into groups, starting a new group when **bool** is `true`.



The following example illustrates a sequence of nodes where **bool** returns `true` whenever the node "header" is encountered. Applying the **`group-starting-with`** function on this sequence of nodes results in two groups, as shown below.

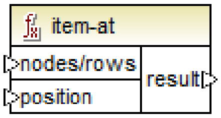


Note that the first node in the sequence starts a new group regardless of the value of **bool**. In other words, a sequence such as the one below would create three groups.



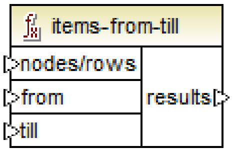
8.9.9.10 *item-at*

Returns the **nodes/rows** at the position supplied by the **position** parameter. The first item is at position "1".



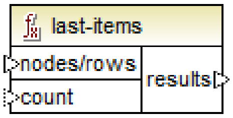
8.9.9.11 items-from-till

Returns a sequence of **nodes/rows** using the "from" and "till" parameters as the boundaries. The first item is at position "1".



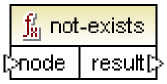
8.9.9.12 last-items

Returns the last "X" **nodes/rows** of the sequence where X is the number supplied by the "count" parameter. The first item is at position "1".



8.9.9.13 not-exists

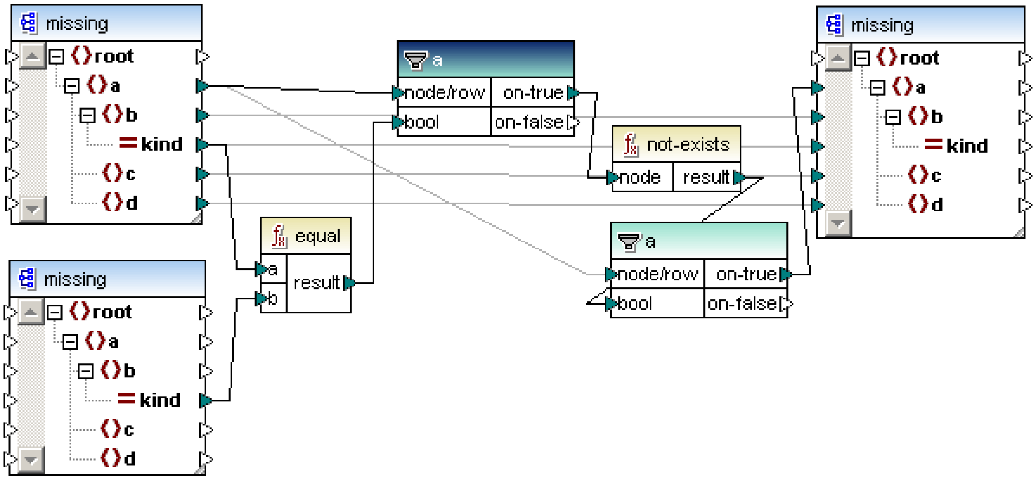
Returns false if the node exists, else returns true.



The example below shows how you can use the not-exists function to map nodes that do not exist in one of a pair of source files.

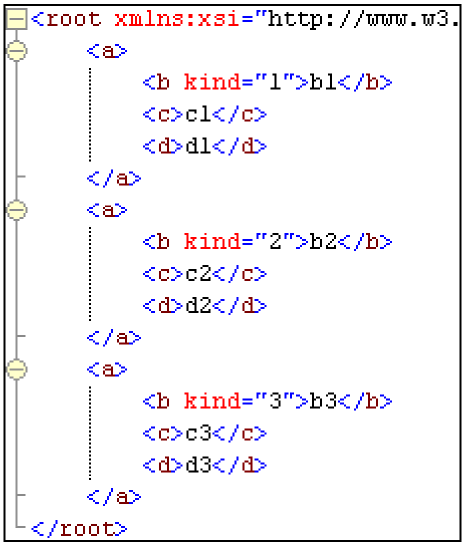
What this mapping does:

- Compare the nodes of two source XML files
- Filter out the nodes of the first source XML file, that do not exist in the second source XML file
- Map only the missing nodes, and their content, to the target file.

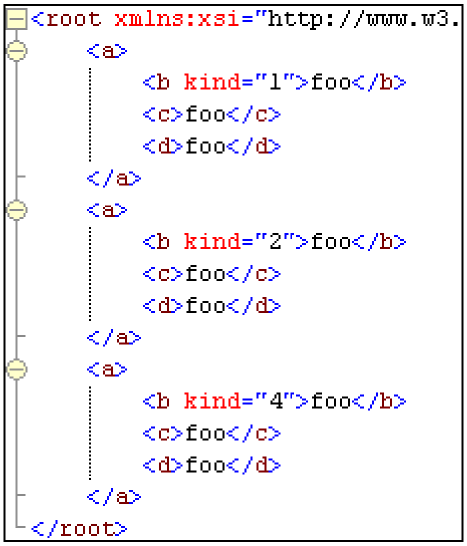


The two XML instance files are shown below, the differences between them are:

- **a.xml** (left) contains the node <b **kind="3"**>, which is missing from b.xml.
- **b.xml** (right) contains the node <b **kind="4">** which is missing from a.xml.



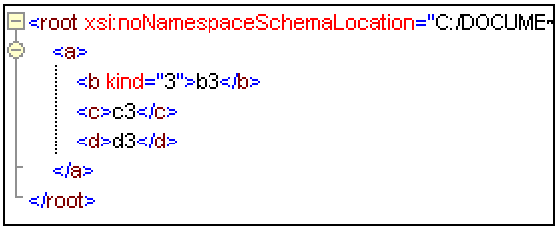
```
<root xmlns:xsi="http://www.w3.
    <a>
        <b kind="1">b1</b>
        <c>c1</c>
        <d>d1</d>
    </a>
    <a>
        <b kind="2">b2</b>
        <c>c2</c>
        <d>d2</d>
    </a>
    <a>
        <b kind="3">b3</b>
        <c>c3</c>
        <d>d3</d>
    </a>
</root>
```



```
<root xmlns:xsi="http://www.w3.
    <a>
        <b kind="1">foo</b>
        <c>foo</c>
        <d>foo</d>
    </a>
    <a>
        <b kind="2">foo</b>
        <c>foo</c>
        <d>foo</d>
    </a>
    <a>
        <b kind="4">foo</b>
        <c>foo</c>
        <d>foo</d>
    </a>
</root>
```

- The equal function compares the kind attribute of both XML files and passes the result to the filter.
- A not-exists function is placed after the initial filter, to select the missing nodes of each of the source files.
- The second filter is used to pass on the missing node and other data only from the a.xml file to the target.

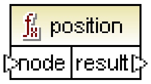
The mapping result is that the node missing from b.xml, <b kind="3">, is passed on to the target component.



```
<root xsi:noNamespaceSchemaLocation="C:/DOCUME
    <a>
        <b kind="3">b3</b>
        <c>c3</c>
        <d>d3</d>
    </a>
</root>
```

8.9.9.14 position

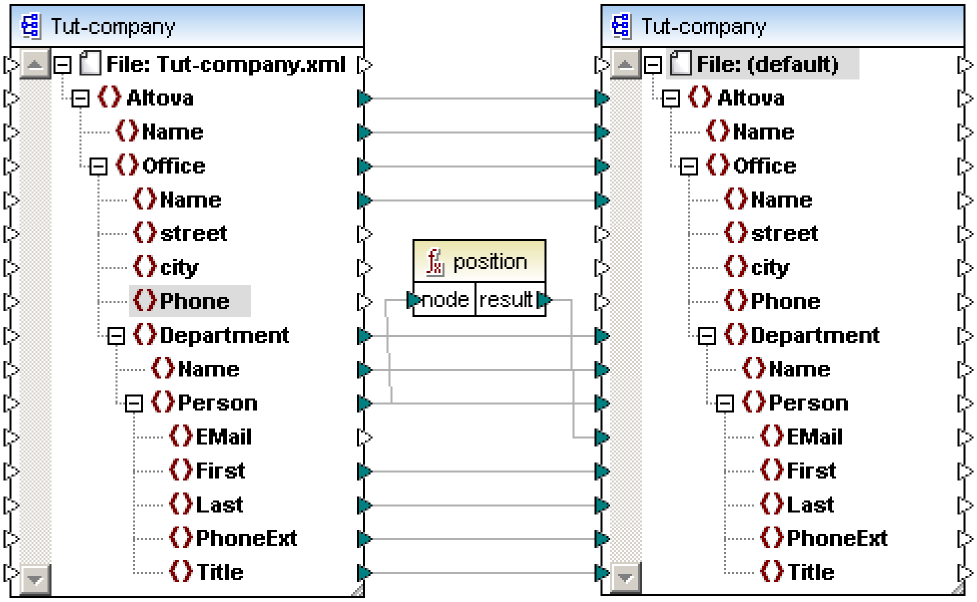
Returns the position of a node inside its containing sequence.



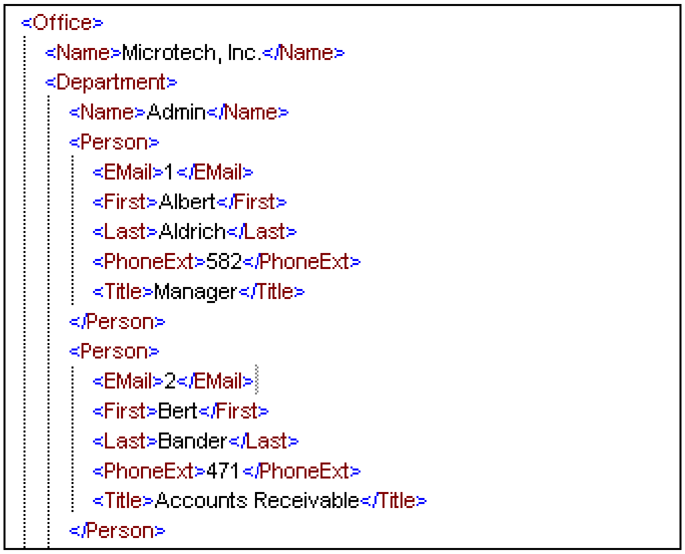
The position function allows you to determine the position of a specific node in a sequence, or use a specific position to filter out items based on that position.

The context item is defined by the item connected to the "node" parameter of the position function, Person, in the example below.

The simple mapping below adds a position number to each Person of each Department.



The position number is reset for each Department in the Office.

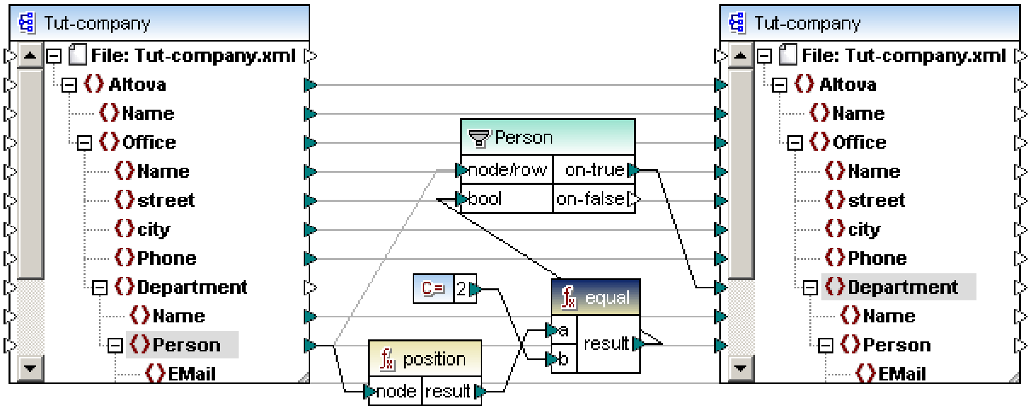


```
<Office>
  <Name>Microtech, Inc.</Name>
  <Department>
    <Name>Admin</Name>
    <Person>
      <EMail>1</EMail>
      <First>Albert</First>
      <Last>Aldrich</Last>
      <PhoneExt>582</PhoneExt>
      <Title>Manager</Title>
    </Person>
    <Person>
      <EMail>2</EMail>
      <First>Bert</First>
      <Last>Bander</Last>
      <PhoneExt>471</PhoneExt>
      <Title>Accounts Receivable</Title>
    </Person>
```

Using the position function to filter out specific nodes

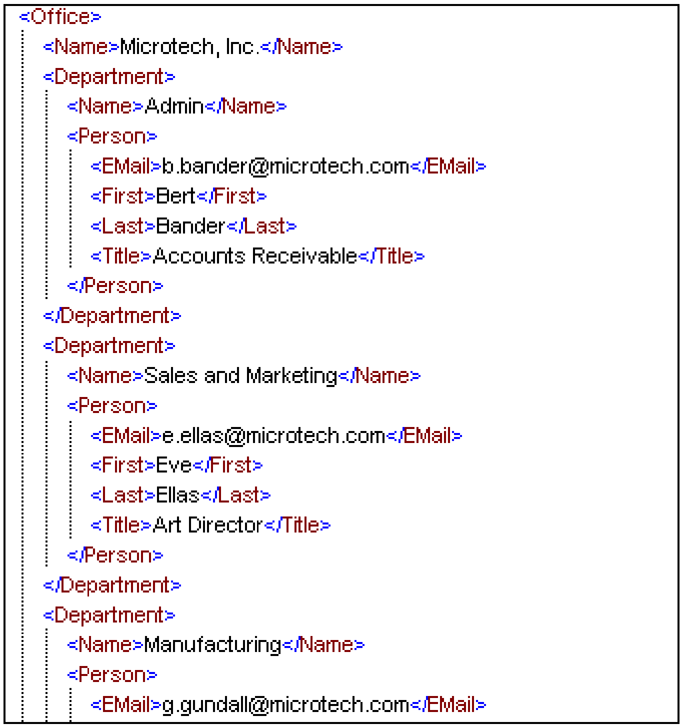
Using the position function in conjunction with a filter allows you to map only those specific nodes that have a certain position in the source component.

The filter "node/row" parameter and the position "node" must be connected to the same item of the source component, to filter out a specific position of that sequence.



What this mapping does is to output:

- The second Person in each Department
- of each Office in Altova.



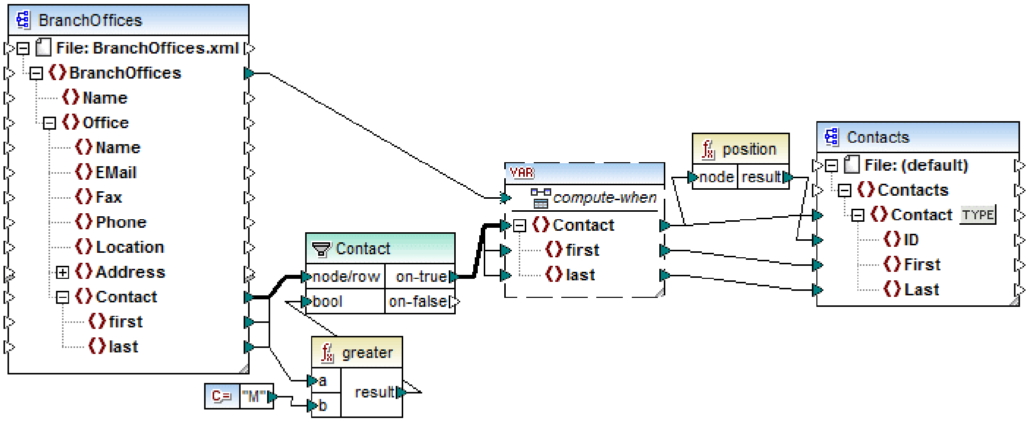
```
<Office>
  <Name>Microtech, Inc.</Name>
  <Department>
    <Name>Admin</Name>
    <Person>
      <EMail>b.bander@microtech.com</EMail>
      <First>Bert</First>
      <Last>Bander</Last>
      <Title>Accounts Receivable</Title>
    </Person>
  </Department>
  <Department>
    <Name>Sales and Marketing</Name>
    <Person>
      <EMail>e.ellas@microtech.com</EMail>
      <First>Eve</First>
      <Last>Ellas</Last>
      <Title>Art Director</Title>
    </Person>
  </Department>
  <Department>
    <Name>Manufacturing</Name>
    <Person>
      <EMail>g.gundall@microtech.com</EMail>
```

Finding the position of items in a filtered sequence:

As the filter component is not a sequence function, it cannot be used directly in conjunction with the position function to find the position of filtered items. To do this you have to use the "Variable" component.

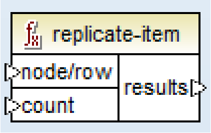
The results of a Variable component are always sequences, i.e. a delimited list of values, which can also be used to create sequences.

- The variable component is used to collect the filtered contacts where the last name starts with a letter higher than "M".
- The contacts are then passed on (from the variable) to the target component
- The position function then numbers these contacts sequentially



8.9.9.15 replicate-item

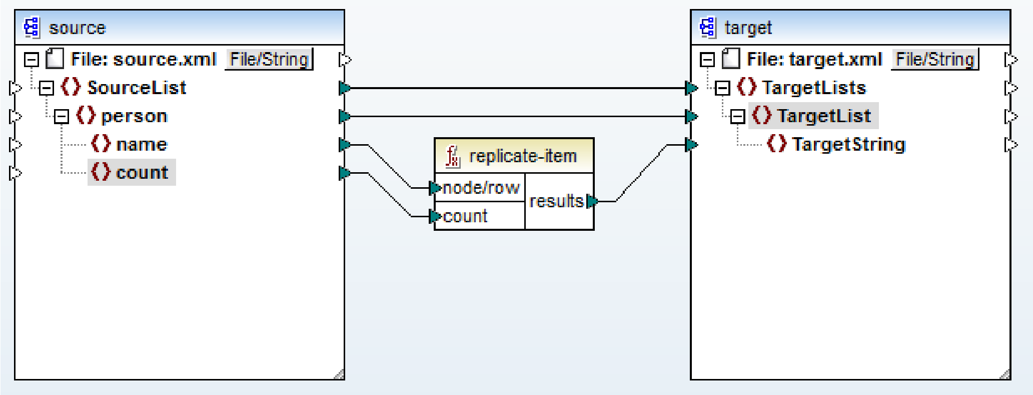
Repeats every item in the input sequence the number of times specified in the `count` argument. If you connect a single item to the `node/row` argument, the function returns N items, where N is the value of the `count` argument. If you connect a sequence of items to the `node/row` argument, the function 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)`.



Note that you can supply a different `count` value for each item. For example, let's assume that you have a source XML file with the following structure:

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

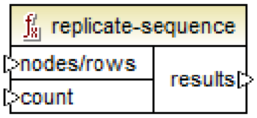


The output is as follows:

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

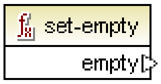
8.9.9.16 replicate-sequence

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)`.



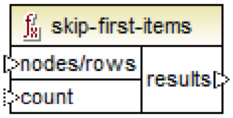
8.9.9.17 set-empty

Returns an empty sequence.



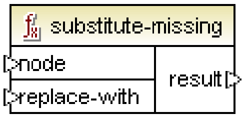
8.9.9.18 skip-first-items

Skips the first "X" items/nodes of the input sequence, where X is the number supplied by the "count" parameter, and returns the rest of the sequence.



8.9.9.19 substitute-missing

This function is a convenient combination of exists and a suitable if-else condition. Used to map the current field content if the node exists in the XML source file, otherwise use the item mapped to the "replace-with" parameter.

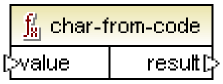


8.9.10 core | string functions

The string functions allow you to use the most common string functions to manipulate many types of source data to: extract portions, test for substrings, or retrieve information on strings.

8.9.10.1 char-from-code

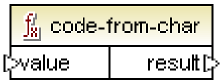
Result is the character representation of the decimal Unicode value of **value**.



For an example, see Replacing Special Characters.

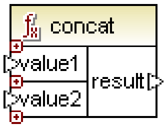
8.9.10.2 code-from-char

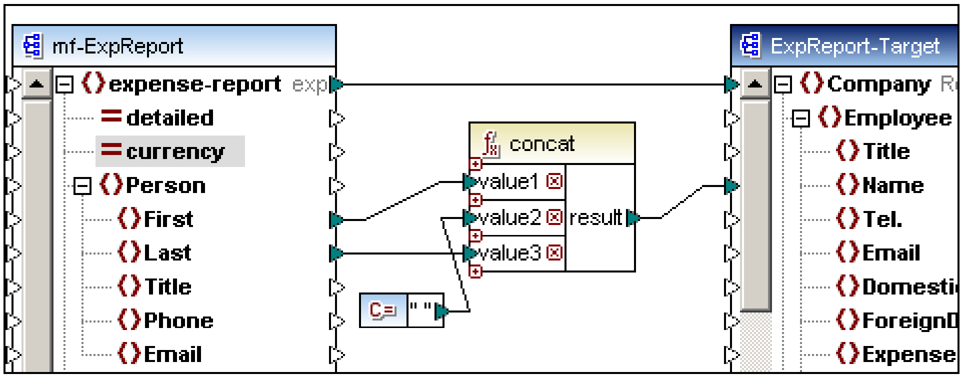
Result is the decimal Unicode value of the first character of **value**.



8.9.10.3 concat

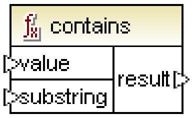
Concatenates (appends) two or more values into a single result string. All input values are automatically converted to type string.





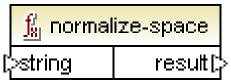
8.9.10.4 contains

Result is true if data supplied to the value parameter contains the string supplied by the substring parameter.



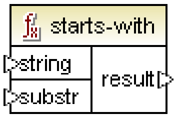
8.9.10.5 normalize-space

Result is the normalized input string, i.e. leading and trailing spaces are removed, then each sequence of multiple consecutive whitespace characters are replaced by a single whitespace character. The Unicode character for "space" is (U+0020).



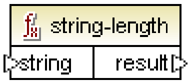
8.9.10.6 starts-with

Result is true if the input string "string" starts with **substr**, else false.



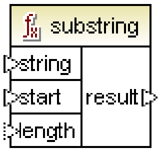
8.9.10.7 string-length

Result is the number of characters supplied by the **string** parameter.



8.9.10.8 substring

Result is the substring (string fragment) of the "string" parameter where "start" defines the position of the start character, and "length" the length of the substring.

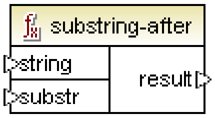


If the length parameter is not specified, the result is a fragment starting at the start position and ending at the end position of the string. Indices start counting at 1.

E.g. substring("56789",2,3) results in 678.

8.9.10.9 substring-after

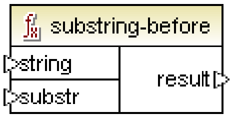
Result is the remainder of the "**string**" parameter, where the first occurrence of the **substr** parameter defines the start characters; the remainder of the string is the result of the function. An empty string is the result, if **substr** does not occur in **string**.



E.g. substring-after("2009/01/04","/") results in the substring 01/04. substr in this case is the first "/"character.

8.9.10.10 substring-before

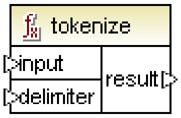
Result is the string fragment of the "**string**" parameter, up to the first occurrence of the **substr** characters. An empty string is the result, if **substr** does not occur in **string**.



E.g. substring-before ("2009/01/04","/") results in the substring 2009. substr in this case is the first "**/"** character.

8.9.10.11 tokenize

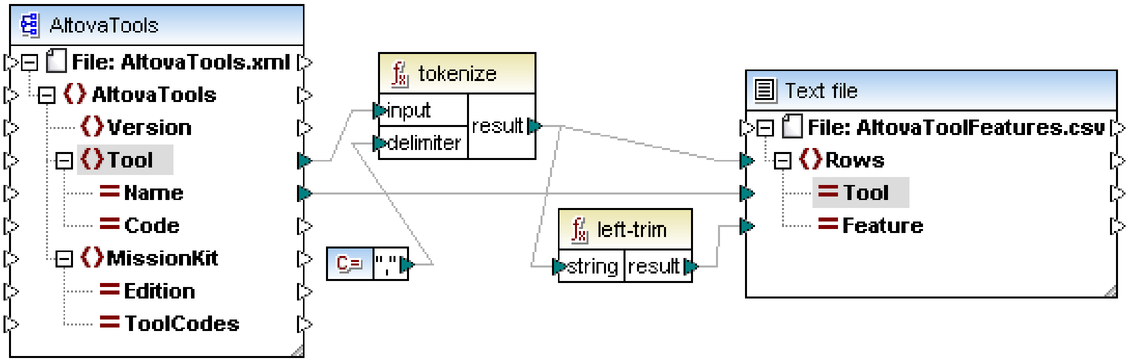
Result is the input string split into a sequence of chunks/sections defined by the delimiter parameter. The result can then be passed on for further processing.



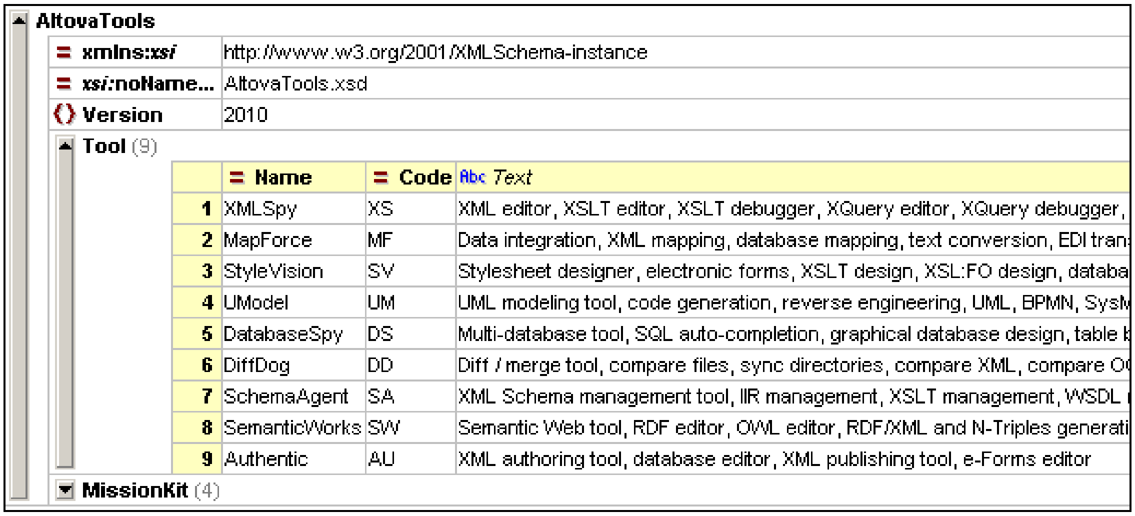
E.g. Input string is A,B,C and delimiter is "," - then result is A B C.

Example

The **tokenizeString1.mfd** file available in the ...\MapForceExamples folder shows how the tokenize function is used.



The XML source file is shown below. The **Tool** element has two attributes: Name and Code, with the Tool element data consisting of comma delimited text.



AltovaTools

xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:noName...	AltovaTools.xsd
Version	2010

Tool (9)

	Name	Code	Text
1	XMLSpy	XS	XML editor, XSLT editor, XSLT debugger, XQuery editor, XQuery debugger,
2	MapForce	MF	Data integration, XML mapping, database mapping, text conversion, EDI tran
3	StyleVision	SV	Stylesheet designer, electronic forms, XSLT design, XSL:FO design, databa
4	UModel	UM	UML modeling tool, code generation, reverse engineering, UML, BPMN, SysM
5	DatabaseSpy	DS	Multi-database tool, SQL auto-completion, graphical database design, table b
6	DiffDog	DD	Diff / merge tool, compare files, sync directories, compare XML, compare O
7	SchemaAgent	SA	XML Schema management tool, IIR management, XSLT management, WSDL
8	SemanticWorks	SW	Semantic Web tool, RDF editor, OWL editor, RDF/XML and N-Triples generati
9	Authentic	AU	XML authoring tool, database editor, XML publishing tool, e-Forms editor

MissionKit (4)

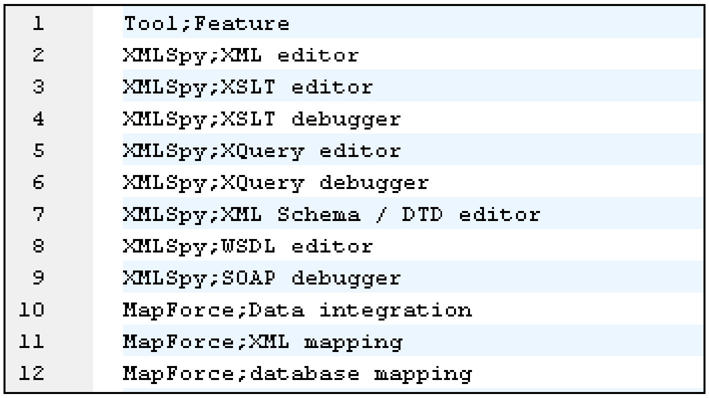
What the mapping does:

- The tokenize function receives data from the **Tool** element/item and uses the comma "**,**" delimiter to split that data into separate chunks. I.e. the first chunk "XML editor".
- As the result parameter is mapped to the **Rows** item in the target component, one **row** is generated for each chunk.
- The result parameter is also mapped to the **left-trim** function which removes the leading white space of each chunk.
- The result of the left-trim parameter (each chunk) is mapped 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).

Result of the mapping:

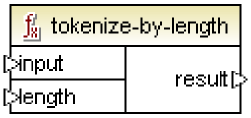
- For each Tool element of the source file
- The (Tool) Name is mapped to the Tool item in the target component
- Each chunk of the tokenized Tool content is appended to the (Tool Name) Feature item
- E.g. The first tool, XMLSpy, gets the first Feature chunk "XML editor"
- This is repeated for all chunks of the current Tool and then for all Tools.
- Clicking the Output tab delivers the result shown below.



```
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
```

8.9.10.12 *tokenize-by-length*

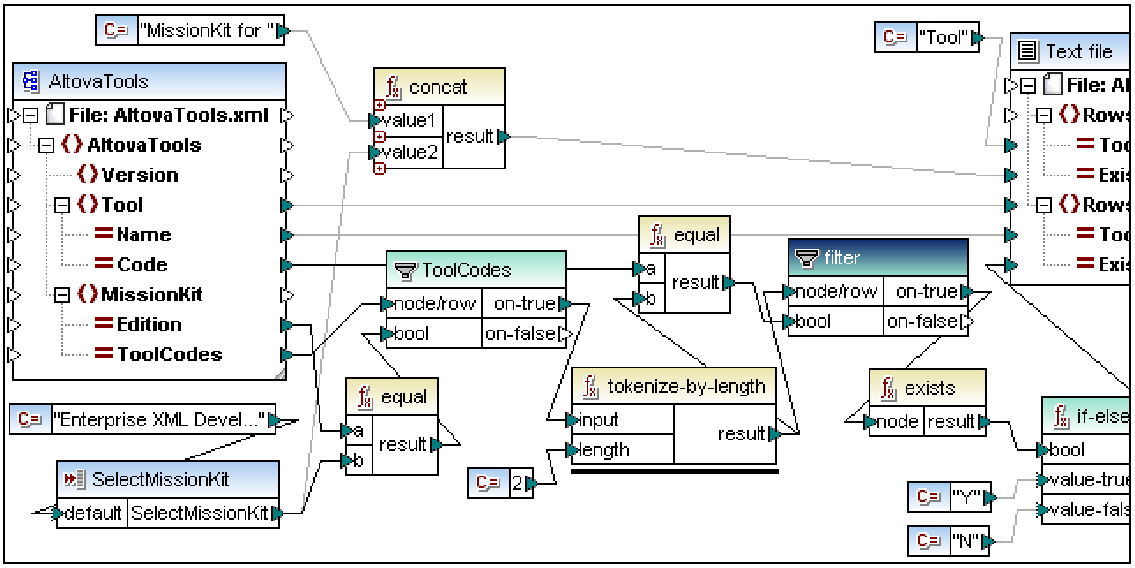
Result is the input string split into a sequence of chunks/sections defined by the length parameter. The result can then be passed on for further processing.



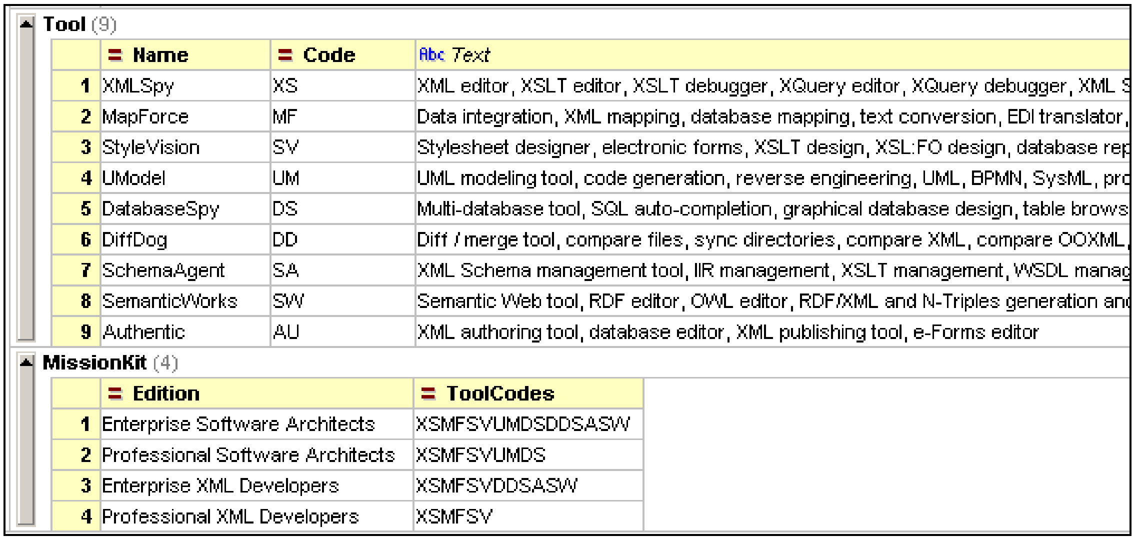
E.g. Input string is ABCDEF and length is "2" - then result is AB CD EF.

Example

The **tokenizeString2.mfd** file available in the ...\MapForceExamples folder shows how the **tokenize-by-length** function is used.



The XML source file is shown below, and is the same as the one used in the previous example. The **MissionKit** element also has two attributes: **Edition** and **ToolCodes**, but no MissionKit element content.



Tool (9)

	Name	Code	Text
1	XMLSpy	XS	XML editor, XSLT editor, XSLT debugger, XQuery editor, XQuery debugger, XML S
2	MapForce	MF	Data integration, XML mapping, database mapping, text conversion, EDI translator,
3	StyleVision	SV	Stylesheet designer, electronic forms, XSLT design, XSL:FO design, database rep
4	UModel	UM	UML modeling tool, code generation, reverse engineering, UML, BPMN, SysML, pro
5	DatabaseSpy	DS	Multi-database tool, SQL auto-completion, graphical database design, table brows
6	DiffDog	DD	Diff / merge tool, compare files, sync directories, compare XML, compare OOXML,
7	SchemaAgent	SA	XML Schema management tool, IIR management, XSLT management, WSDL manag
8	SemanticWorks	SW	Semantic Web tool, RDF editor, OWL editor, RDF/XML and N-Triples generation and
9	Authentic	AU	XML authoring tool, database editor, XML publishing tool, e-Forms editor

MissionKit (4)

	Edition	ToolCodes
1	Enterprise Software Architects	XSMFSVUMDSDDSASW
2	Professional Software Architects	XSMFSVUMDS
3	Enterprise XML Developers	XSMFSVDDSASW
4	Professional XML Developers	XSMFSV

Aim of the mapping:

To generate a list showing which Altova tools are part of the respective MissionKit editions.

How the mapping works:

- The SelectMissionKit **Input** component receives its default input from a constant component, in this case "Enterprise XML Developers".
- The **equal** function compares the input value with the "**Edition**" value 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.

What the tokenize-by-length function does:

- The ToolCodes **input** value XSMFSVDDSASW, is split into multiple chunks of two characters each, defined by **length** parameter, which is 2, thus giving 6 chunks.
- Each chunk (placed in the b parameter) of the equal function, is compared to the 2 character **Code** value of the source file (of which there are 9 entries/items in total).
- The result of the comparison (true/false) is passed on to the **bool** parameter of the filter.
- Note that **all** chunks, of the tokenize-by-length function, are passed on to the **node/row** parameter of the filter.

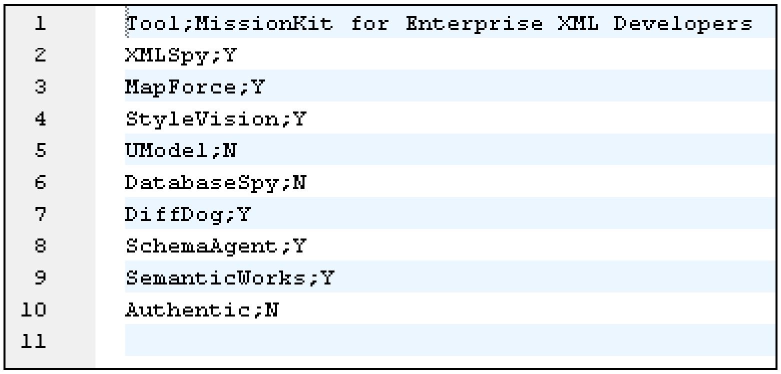
- The **exists** functions now checks for existing/non-existing nodes passed on to it by the **on-true** parameter of the filter component.

  Existing nodes are those where there **is** a match between the **ToolCodes** chunk and the Code value.

  Non-existing nodes are where there was **no ToolCodes** chunk to match a Code value.

- The bool results of the **exists** function are passed on to the **if-else** function which passes on a Y to the target if the node exists, or a N, if the node does not exist.

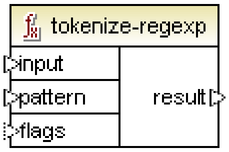
Result of the mapping:

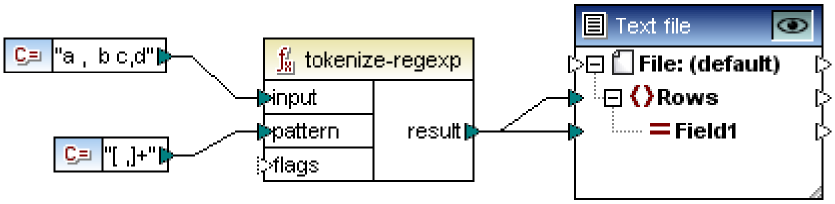


```
Tool;MissionKit for Enterprise XML Developers
XMLSpy;Y
MapForce;Y
StyleVision;Y
UModel;N
DatabaseSpy;N
DiffDog;Y
SchemaAgent;Y
SemanticWorks;Y
Authentic;N

```

8.9.10.13 tokenize-regexp

Result is the input string split into a sequence of strings, where the supplied regular expression **pattern** match defines the separator. The separator strings are not output by the result parameter. Optional flags may also be used.



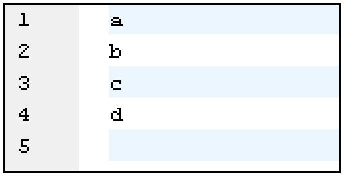


In the example shown above:
input string is a succession of characters separated by spaces and/or commas, i.e. a , b c,d

The regex **pattern** defines a character class **[**"space""comma"**]** - of which one and only one character will be matched in a character class, i.e. either space or comma.

The **+** quantifier specifies "one or more" occurrences of the character class/string.

result string is:



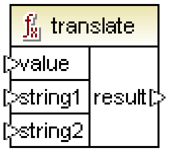
```
1    a
2    b
3    c
4    d
5
```

Please note that there are slight differences in regular expression syntax between the various languages. Tokenize-regexp in C++ is only available in Visual Studio 2008 SP1 and later.

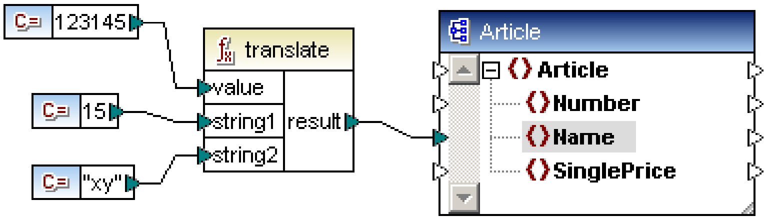
For more information on regular expressions, see Regular expressions.

8.9.10.14 *translate*

The characters of **string1** (search string) are replaced by the characters at the same position in **string2 (replace string**), in the input string "**value**".



When there are no corresponding characters in string2, the character is removed.



E.g.
input string is 123145
(search) string1 is 15
(replace) string2 is xy

So:

each 1 is replaced by **x** in the input string value
each 5 is replaced by **y** in the input sting value

Result string is **x23x4y**

If string2 is empty (fewer characters than string1) then the character is removed.

E.g.2
input string aabaacbca
string1 is "**a**"
string2 is **""** (empty string)

result string is "bcbc"

E.g.3
input string aabaacbca
string1 is "**ac**"
string2 is "**ca**"

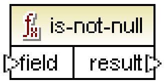
result string is "ccbccabac"

8.9.11 db

The **db** library contains functions that allow you to define the mapping results when encountering null fields in databases.

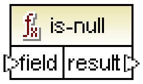
8.9.11.1 is-not-null

Returns false if the field is null, otherwise returns true.



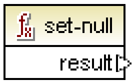
8.9.11.2 is-null

Returns true if the field is null, otherwise returns false.



8.9.11.3 set-null

Used to set a database column to null. This function will also overwrite a default value with null. If connected to something else i.e. not a database field, it will behave like an empty sequence.

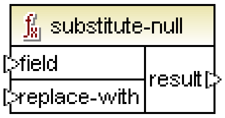


Please note:

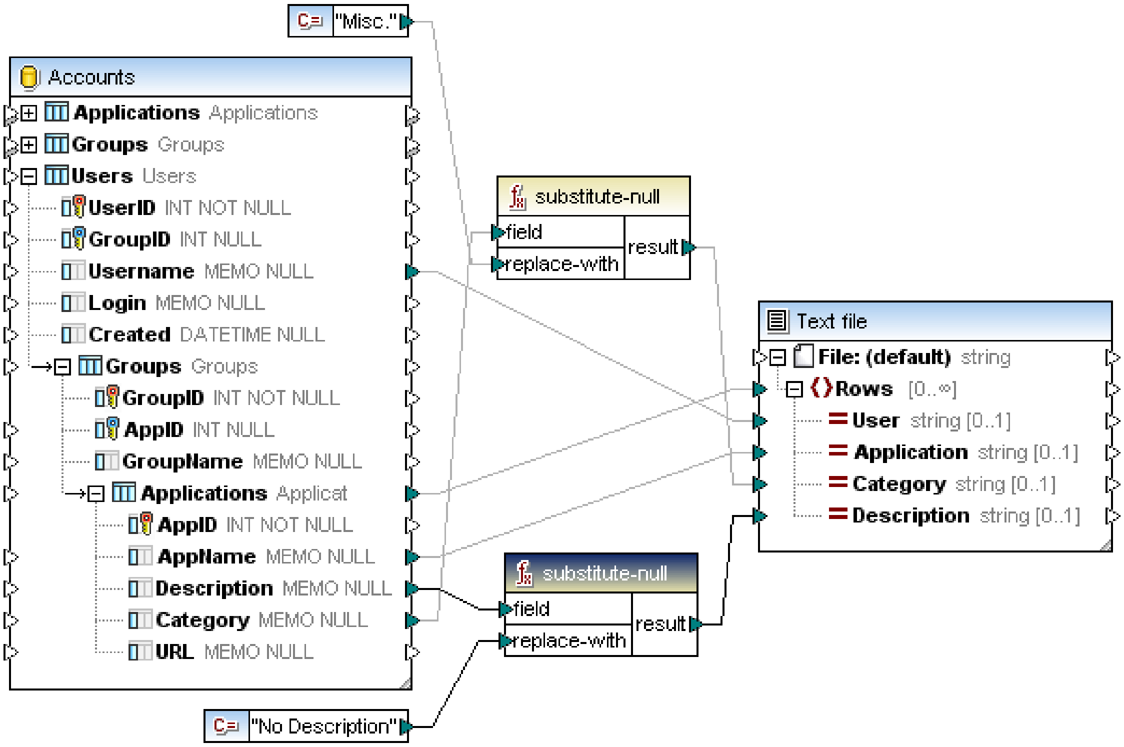
- Connecting set-null to a different function will usually result in the other function not being called at all. Connecting set-null to a sequence function, e.g. count, will call the function with an empty sequence.
- Connecting to special functions, Filters and IF-Else conditions works as expected, 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.

8.9.11.4 substitute-null

Used to map the current field content if it exists, otherwise use the item mapped to the replace-with parameter.



The image below shows an example of the substitute-null function in use, and is available as "**DB-ApplicationList**" in the **...\MapForceExamples** folder.



The first function checks if a Category entry exists in the Applications table. As one does not exist for the Notepad application, "Misc" is mapped to the Category item of the Text file.

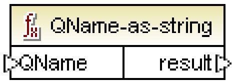
The second function checks if a Description entry exist, and maps the string "No description" if one does not exist, which is also the case with the Notepad application.

8.9.12 lang | QName functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

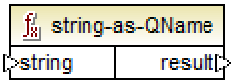
8.9.12.1 QName-as-string

Result is the unique string representation of the QName.



8.9.12.2 string-as-QName

Converts the string representation of a QName back to a QName.

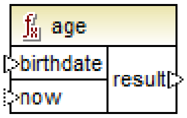


8.9.13 lang | datetime functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

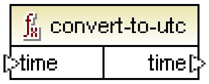
8.9.13.1 age

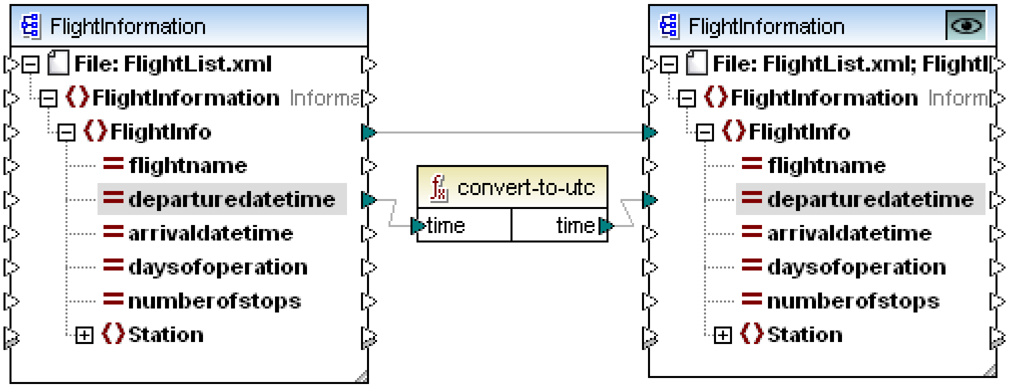
age
Result is the age of the person in full years. The *now* argument is optional and the default is the current system date. The result is then the full amount of years between the birthdate and now. If a value is mapped to the *now* argument, the result is the difference between the two dates in full years.



8.9.13.2 convert-to-utc

Converts the local "time" input parameter into Coordinated Universal Time, or GMT/Zulu time. (The function takes the timezone component, e.g. +5:00, into account).





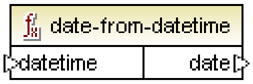
E.g. Instance document datetime:
`departuredatetime="2001-12-17T`**`09`**`:30:02+05:00"`

Result:
`departuredatetime="2001-12-17T`**`04`**`:30:02"`

Please note:
If the source dateTime is in the form `departuredatetime="2001-12-17T09:30:02`**`Z`**`"` then no conversion will take place because the trailing "**Z**" defines this time to be Zulu time, i.e. UTC. The result will be `departuredatetime="2001-12-17T`**`09`**`:30:02"`.

8.9.13.3 date-from-datetime

Result is the date part of a datetime input argument. The time part of the dateTime, starting with T in the instance document, is set to zero. Note that the timezone increment is not changed.

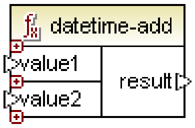


E.g. Instance document datetime:
`departuredatetime="2001-12-17T09:30:02+05:00"`

Result:
`departuredatetime="2001-12-17T`**`00:00:00`**`+05:00"`

8.9.13.4 datetime-add

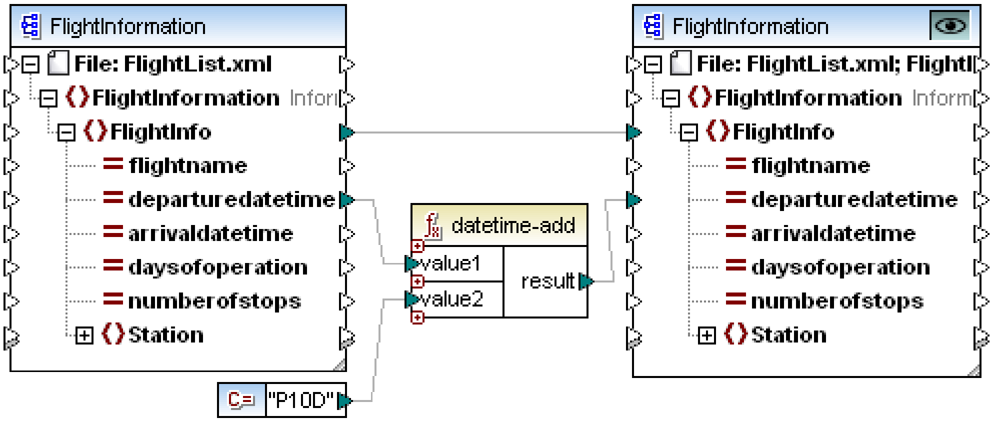
Result is the datetime obtained by adding a duration (second argument) to a datetime (first argument).



Durations must be entered in the form: P1Y2M3DT04H05M. Periods can be made negative by adding the minus character before the P designator, e.g. -P1D.

P is the period designator, and is mandatory; the rest of above period is therefore:
1 Year, 2 Months, 3 Days T(ime designator), 04 Hours, 05 Minutes.

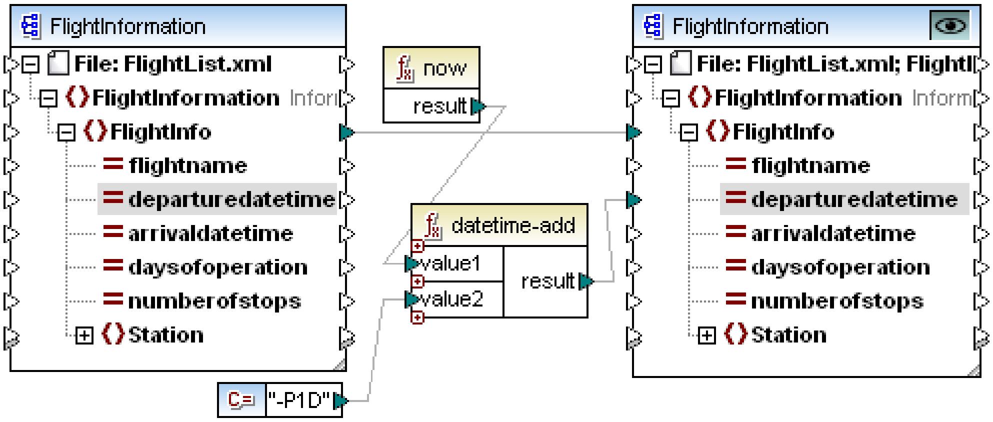
The example shown below, adds 10 days to the departuredatetime, i.e. P10D.



E.g. Instance document datetime:
`departuredatetime="2001-12-17T09:30:02+05:00"`

Result:
`departuredatetime="2001-12-27T09:30:02+05:00"`

To extract yesterdays date from dateTime input:
Use the "now" function to input the current date/time including timezone. A period can be made negative by using the minus character before the P designator, e.g. -P1D (minus 1 day).



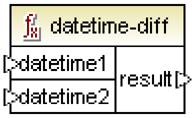
E.g. datetime now is 28th Feb 2012, 17:19:54.748(millisec)+01timezone.
now=`"2012-02-28T17:19:54.748+01:00"`

```
Result:
departuredatetime="2012-02-27T17:19:54.748+01:00"
```

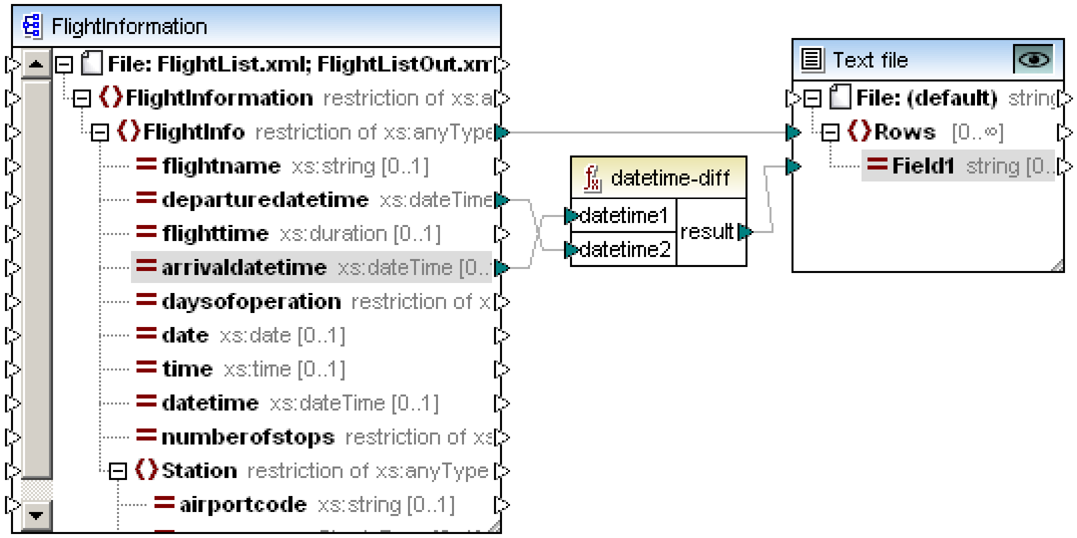
i.e. 27th Feb 2012, 17:19:54.748(millisec)+01timezone

8.9.13.5 datetime-diff

Result is the duration obtained by subtracting datetime2 (second argument) from datetime1 (first argument). The result can be mapped to a string, or duration, datatype.



Note that the arrivaldatetime has been connected to datetime1 and departuredatetime to datetime2.


FlightInformation
File: FlightList.xml; FlightListOut.xm
FlightInformation restriction of xs:a
FlightInfo restriction of xs:anyType
flightname xs:string [0..1]
departuredatetime xs:dateTime
flighttime xs:duration [0..1]
arrivaldatetime xs:dateTime [0..
daysofoperation restriction of x
date xs:date [0..1]
time xs:time [0..1]
datetime xs:dateTime [0..1]
numberofstops restriction of xs
Station restriction of xs:anyType
airportcode xs:string [0..1]
datetime-diff
datetime1
datetime2
result
Text file
File: (default) string
Rows [0..∞]
Field1 string [0..

E.g. We want to find the difference, as a duration, between the departure and arrival times.

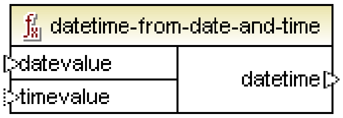
```
datetime1 arrivaldatetime="2001-12-17T19:30:02+05:00"
datetime2 departuredatetime="2001-12-17T09:30:02+05:00"
```

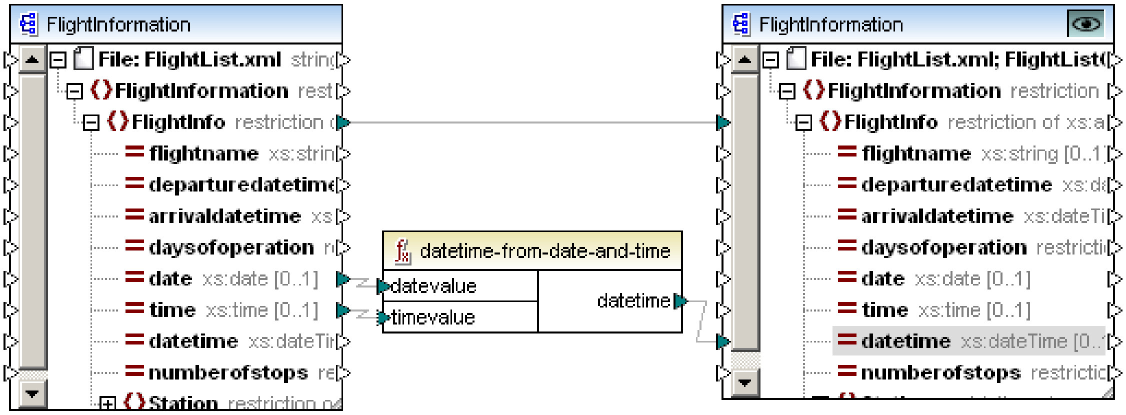
Result: the difference between the two is 10 hours:

```
result= PT10H
```

8.9.13.6 *datetime-from-date-and-time*

Result is a datetime built from a datevalue (first argument) and a timevalue (second argument). The first argument must be of type xs:date and the second xs:time. The result can be mapped to a sting or dateTime datatype.





E.g.

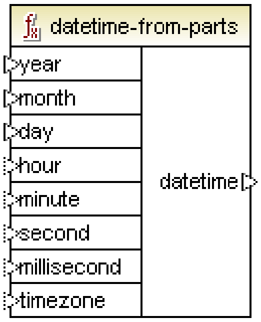
```
date="2012-06-29"
time="11:59:55"
```

Result:

```
dateTime="2012-06-29T11:59:55"
```

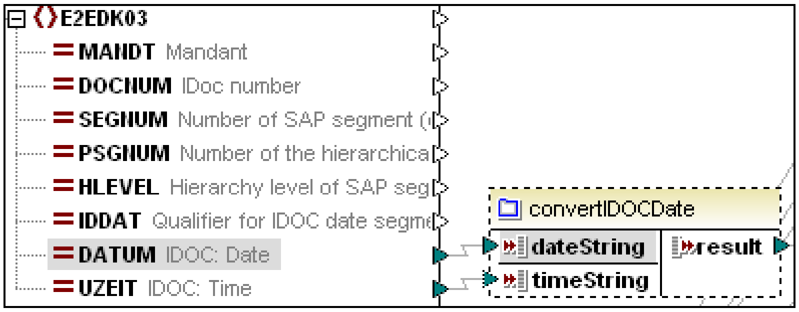
8.9.13.7 *datetime-from-parts*

Result is a 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 e.g. 32nd of January will automatically be changed to 1st February.



All of the arguments are of type xs:int except for millisecond, which is of type xs:decimal. The datetime result parameter is of type xs:dateTime.

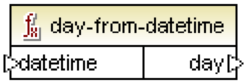
The date and time fields are supplied by the IDOC instance file:

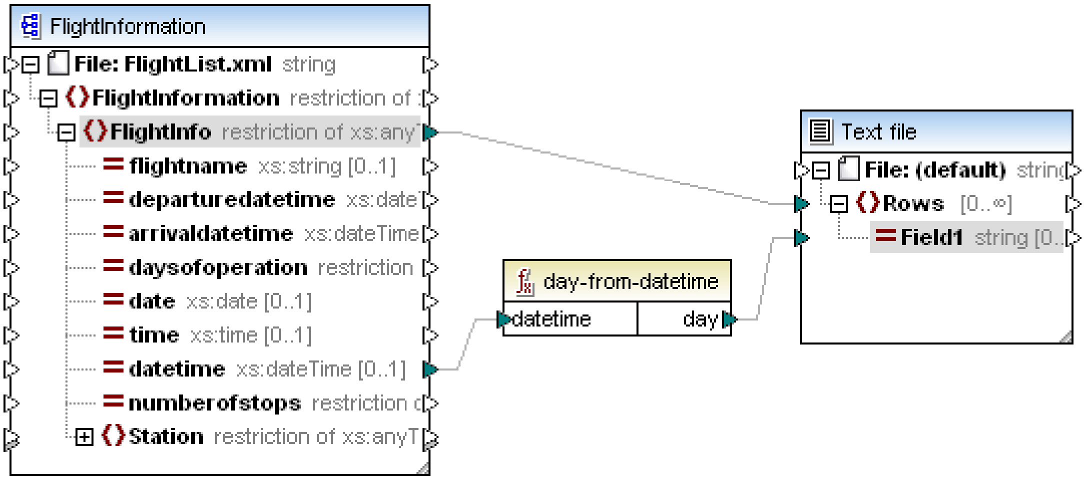

E2EDK03
MANDT Mandant
DOCNUM IDoc number
SEGNUM Number of SAP segment (
PSGNUM Number of the hierarchica
HLEVEL Hierarchy level of SAP seg
IDDAT Qualifier for IDOC date segme
DATUM IDOC: Date
UZEIT IDOC: Time
convertIDOCDate
dateString
timeString
result

```
IDOC:Date    19990621
ICOC:Time    0930
Result       1999-06-21T09:30:00
```

8.9.13.8 day-from-datetime

Result is the day from the datetime argument.



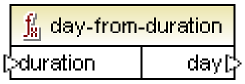

FlightInformation
File: FlightList.xml string
FlightInformation restriction of
FlightInfo restriction of xs:any
flightname xs:string [0..1]
departuredatetime xs:date
arrivaldatetime xs:dateTime
daysofoperation restriction
date xs:date [0..1]
time xs:time [0..1]
datetime xs:dateTime [0..1]
numberofstops restriction
Station restriction of xs:anyT
day-from-datetime
datetime
day
Text file
File: (default) strin
Rows [0..∞]
Field1 string [0..

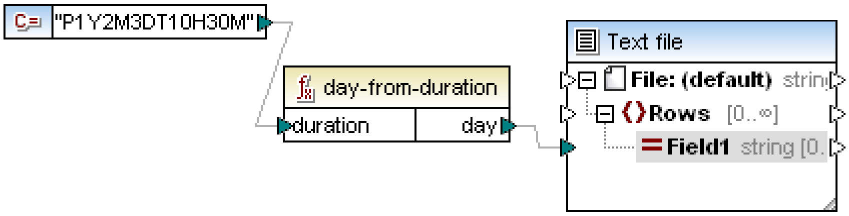
E.g.
datetime="2001-12-**17**T10:30:03+01:00"

Result: 17

8.9.13.9 day-from-duration

Result is the day from the duration argument.

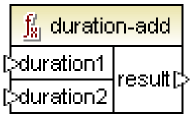




E.g.
duration="P1Y2M**3D**T10H30M"
Result: 3

8.9.13.10 duration-add

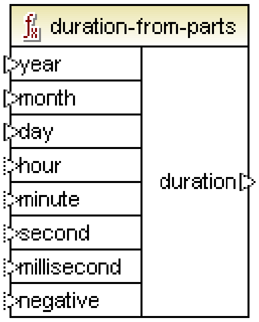
Result is the duration obtained by adding two durations.



E.g.
```
duration1="P0Y0M3DT03H0M" (3days 3 hours)
duration2="P0Y0M3DT01H0M" (3days 1 hour)
Result: P6DT4H (6days 4 hours)
```

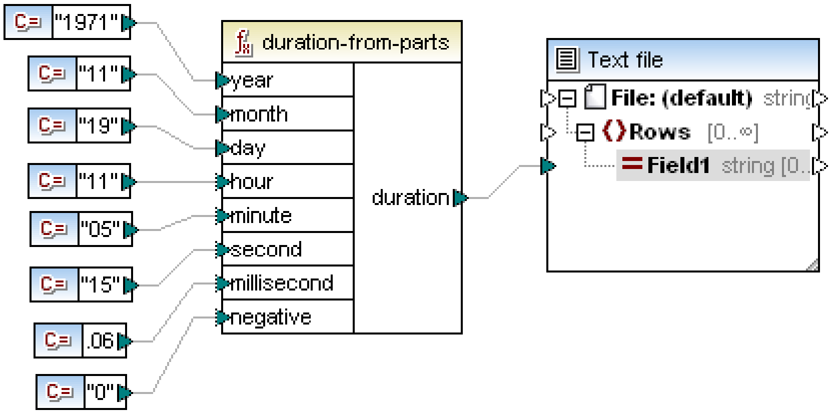
8.9.13.11 duration-from-parts

Result is a duration calculated by combining the following parts supplied as arguments: year, month, day, hour, minute, second, millisecond, negative.



Durations are in the form P1Y2M3DT04H05M06.07S i.e. P(eriod) 1 Year, 2 Months, 3 Days, T(ime designator), 04 Hours, 05 Minutes, 06.07 seconds.milliseconds.

All of the arguments are of type xs:int except for millisecond, which is of type xs:decimal, and negative, which is of type xs:boolean (i.e. 1 for true, 0 for false). The duration parameter is of type xs:duration.



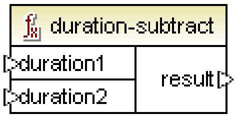
Parts: 1971 year, 11 month, 19 day, 11 hour, 05 minutes, 15.06 seconds, negative period "false".

Result:

```
duration="P1971Y11M19DT11H5M15.00006S"
```

8.9.13.12 duration-subtract

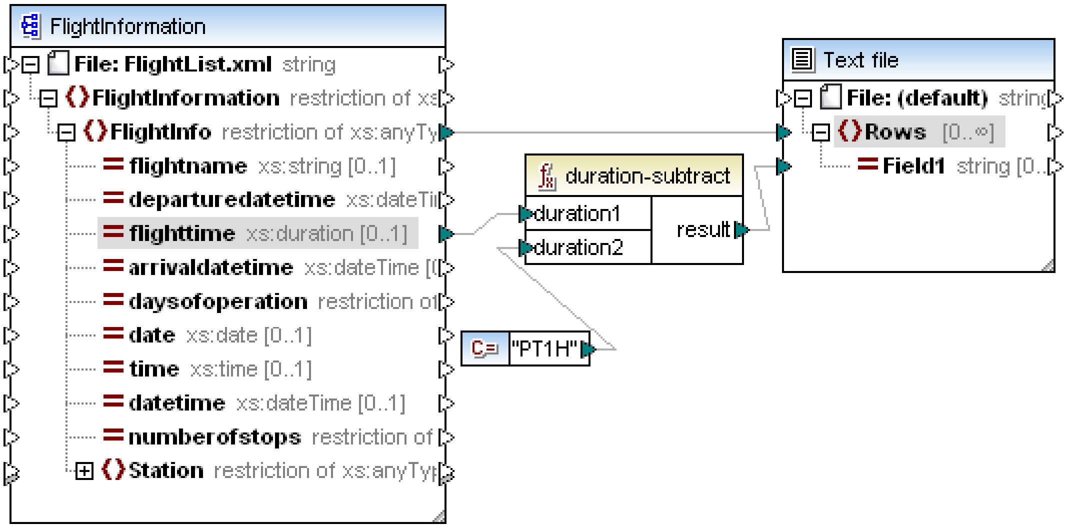
Result is the duration obtained by subtracting duration2 from duration1.



Durations must be entered in the form: P1Y2M3DT04H05M. Periods can be made negative by using the minus character before the P designator, e.g. -P1D.

P is the period designator, and is mandatory; the rest of period is therefore:
1 Year, 2 Months, 3 Days T(ime designator), 04 Hours, 05 Minutes.

The example shown below, subtracts 1 hour from flighttime, i.e. PT1H.

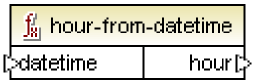

FlightInformation
File: FlightList.xml string
FlightInformation restriction of xs
FlightInfo restriction of xs:anyTy
flightname xs:string [0..1]
departuredatetime xs:dateTi
flighttime xs:duration [0..1]
arrivaldatetime xs:dateTime [0
daysofoperation restriction of
date xs:date [0..1]
time xs:time [0..1]
datetime xs:dateTime [0..1]
numberofstops restriction of
Station restriction of xs:anyTy
duration-subtract
duration1
duration2
result
"PT1H"
Text file
File: (default) string
Rows [0..∞]
Field1 string [0..

E.g.
duration1="P0Y0M0DT**05**H07M"
duration2="PT1H"

Result: PT4H7M

8.9.13.13 hour-from-datetime

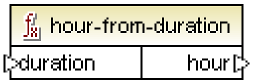
Result is the hour part of the datetime argument.



E.g.
datetime="2001-12-17T**09**:30:02+05:00"
hour= 9

8.9.13.14 hour-from-duration

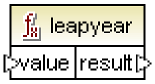
Result is the hour component of the duration argument.

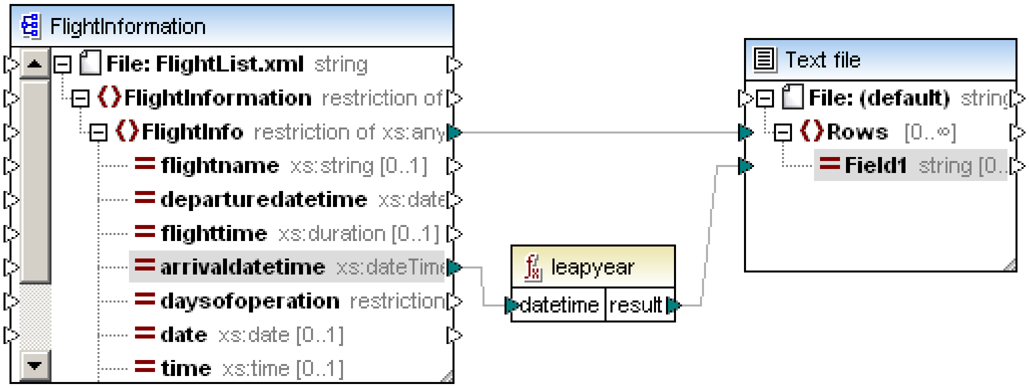


E.g.
duration="P0Y0M0DT**05**H07M"
hour= 5

8.9.13.15 leapyear

Result is true or false depending on whether the year of the supplied dateTime is in a leap year.



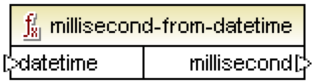

FlightInformation
File: FlightList.xml string
FlightInformation restriction of
FlightInfo restriction of xs:any
flightname xs:string [0..1]
departuredatetime xs:date
flighttime xs:duration [0..1]
arrivaldatetime xs:dateTime
daysofoperation restriction
date xs:date [0..1]
time xs:time [0..1]
leapyear
datetime
result
Text file
File: (default) string
Rows [0..∞]
Field1 string [0..

E.g.

```
arrivaldatetime="2001-12-17T19:30:02+05:00"
result="false"
```

8.9.13.16 millisecond-from-datetime

Result is the millisecond part of the datetime argument.

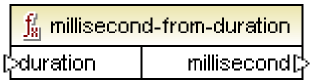


E.g.

```
datetime="2001-12-17T09:30:02.544+05:00"
millisecond= 544
```

8.9.13.17 millisecond-from-duration

Result is the millisecond component of the duration argument.

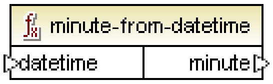


E.g.

```
duration="P0Y0M0DT05H07M02.227S"
millisecond= 227
```

8.9.13.18 minute-from-datetime

Result is the minute part of the datetime argument.

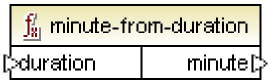


E.g.

```
datetime="2001-12-17T09:30:02.544+05:00"
minute= 30
```

8.9.13.19 minute-from-duration

Result is the minute component of the duration argument.

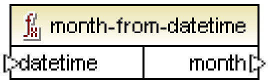


E.g.

```
duration="P0Y0M0DT05H07M02.227S"
minute= 7
```

8.9.13.20 month-from-datetime

Result is the month part of the dateTime argument.

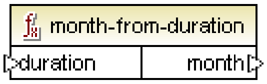


E.g.

```
datetime="2001-12-17T09:30:02.544+05:00"
month= 12
```

8.9.13.21 month-from-duration

Result is the month component of the duration argument.

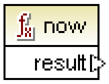


E.g.

```
duration="P0Y04M0DT05H07M02.227S"
month= 4
```

8.9.13.22 now

Result is the current dateTime (including timezone).

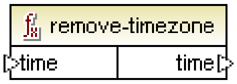


E.g.
result= 2012-03-06T14:44:57.567+01:00

For an example on how to extract yesterday's date, see the core | lang | datetime-add function.

8.9.13.23 remove-timezone

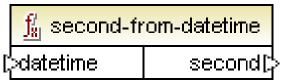
Removes the timezone component, e.g. +5:00, from the time input parameter.



E.g.
departuredatetime="2001-12-17T09:30:02**+05:00**"
time: 2001-12-17T09:30:02

8.9.13.24 second-from-datetime

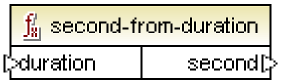
Result is the seconds part of the dateTime argument.



E.g.
datetime="2001-12-17T09:30:**02**.544+05:00"
second= 2

8.9.13.25 second-from-duration

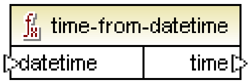
Result is the seconds component of the duration argument.



E.g.
duration="P0Y04M0DT05H07M**02**.227S"
second= 2

8.9.13.26 time-from-datetime

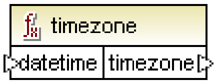
Result is the time part of the dateTime argument.



E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
time= 09:31:02+05:00
```

8.9.13.27 timezone

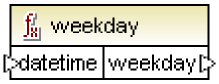
Returns the timezone (i.e. +05:00 here) relative to UTC of the dateTime value. NB timezone unit is minutes.



E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
timezone= 300
```

8.9.13.28 weekday

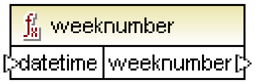
Returns the weekday of the dateTime value, starting with `Monday=1` to `Sunday=7`.



E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
weekday= 1
```

8.9.13.29 weeknumber

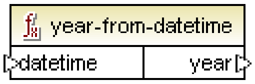
Returns the week number within the year specified by the dateTime value.



E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
weeknumber= 51
```

8.9.13.30 year-from-datetime

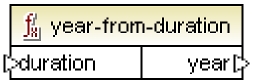
Result is the year part of the dateTime argument.



E.g.
datetime="**2001**-12-17T09:30:02.544+05:00"
year= 2001

8.9.13.31 year-from-duration

Result is the year component of the duration argument.



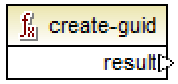
E.g.
duration="P**01**Y04M0DT05H07M02.227S"
year= 1

8.9.14 lang | generator functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages. The generator functions generate values for database fields, which do not have any input data from the Schema, database or EDI source component.

8.9.14.1 create-guid

Result is a globally-unique identifier (as a hex-encoded string) for the specific field.

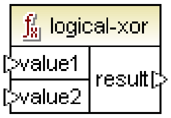


8.9.15 lang | logical functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

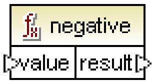
8.9.15.1 logical-xor

Result is true if value1 is different than value2, otherwise false.



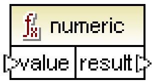
8.9.15.2 negative

Result is true if value is negative, i.e. less than zero, otherwise false.



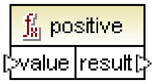
8.9.15.3 numeric

Result is true if value is a number, otherwise false. The input will usually be a string.



8.9.15.4 positive

Result is true if value is positive, i.e. equal to or greater than zero, otherwise false.

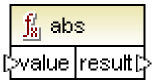


8.9.16 lang | math functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

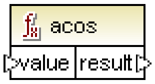
8.9.16.1 abs

Result is the absolute value of the input **value**.



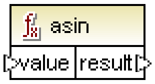
8.9.16.2 acos

Result is the arc cosine of **value**.



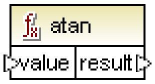
8.9.16.3 asin

Result is the arc sine of **value**.



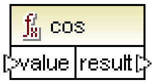
8.9.16.4 atan

Result is the arc tangent of **value**.



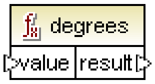
8.9.16.5 cos

Result is the cosine of **value**.



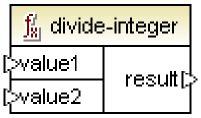
8.9.16.6 degrees

Result is the conversion of **value** in radians into degrees.



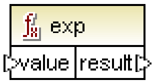
8.9.16.7 divide-integer

Result is the integer result of dividing **value1** by **value2**. E.g. 15 divide-integer 2, integer result is 7.



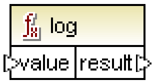
8.9.16.8 exp

Result is **e** (base natural logarithm) raised to the **value**th power.



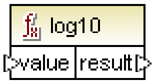
8.9.16.9 log

Result is the natural logarithm of **value**.



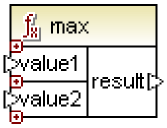
8.9.16.10 log10

Result is logarithm (base 10) of **value**.



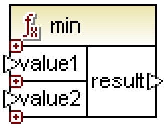
8.9.16.11 max

Result is the numerically larger value of **value1** compared to **value2**.



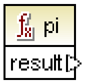
8.9.16.12 min

Result is the numerically smaller value of **value1** compared to **value2**.



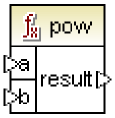
8.9.16.13 pi

Result is the value of pi.



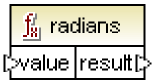
8.9.16.14 pow

Result is the value of **a** raised to the power **bth power**.



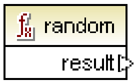
8.9.16.15 radians

Result is the conversion of **value** in degrees to radians.



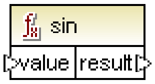
8.9.16.16 random

Result is a pseudorandom value between 0.0 and 1.0



8.9.16.17 sin

Result is the sine of **value**.



8.9.16.18 sqrt

Result is the square root of **value**.

8.9.16.19 tan

Result is the tangent of **value**.

8.9.16.20 unary-minus

Result is the negation of the signed input **value**. E.g. +3 result is -3, while -3 result is 3.

8.9.17 lang | string functions

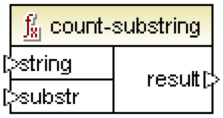
The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

8.9.17.1 capitalize

Result is the input string **value,** where the first letter of each word is capitalized (initial caps).

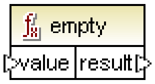
8.9.17.2 count-substring

Result is the number of times that **substr** occurs in **string**.



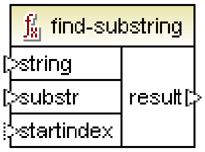
8.9.17.3 empty

Result is true if the input string **value** is empty, otherwise false.



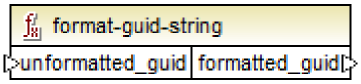
8.9.17.4 find-substring

Returns the position of the first occurrence of substr. within string, starting at position startindex. The first character has position 1. If the substring could not be found, then the result is 0.



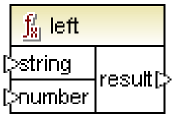
8.9.17.5 format-guid-string

Result is a correctly formatted GUID string **formatted_guid,** using **unformatted_guid** as the input string, for use in database fields. See also the create-guid function in the **lang | generator functions** library.



8.9.17.6 left

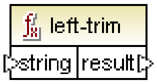
Result is a string containing the first **number** characters of **string**.



E.g. string="This is a sentence" and number=4, result is "This".

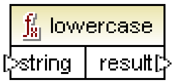
8.9.17.7 left-trim

Result is the input string with all leading whitespace characters removed.



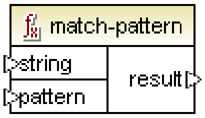
8.9.17.8 lowercase

Result is the lowercase version of the input **string**. For Unicode characters the corresponding lower-case characters (defined by the Unicode consortium) are used.



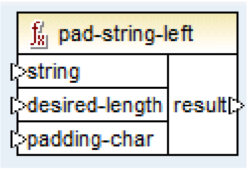
8.9.17.9 match-pattern

Result is true if the input **string** matches the regular expression defined by **pattern**, else false. The specific regular expression syntax depends on the target language (see also Regular expressions).



8.9.17.10 pad-string-left

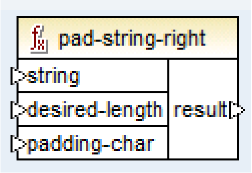
Returns a string which is padded to the left by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.



string	Specifies the input string.
desired-length	Defines the desired length of the string after padding.
padding-char	Defines the character to use as padding character.

8.9.17.11 *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.



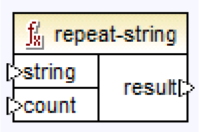
string Specifies the input string.

desired-length Defines the desired length of the string after padding.

padding-char Defines the character to use as padding character.

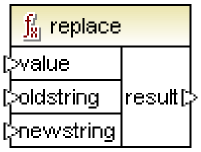
8.9.17.12 *repeat-string*

Repeats the string supplied as argument *n* times. The **count** argument specifies the number of times to repeat the string.



8.9.17.13 *replace*

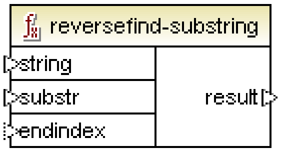
Result is a new string where each instance of **oldstring**, in the input string **value**, is replaced by **newstring**.



For an example, see Replacing Special Characters.

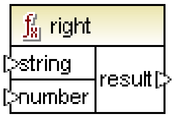
8.9.17.14 *reversefind-substring*

Returns the position of the first occurrence of **substr**. within **string**, starting at position **endindex**, i.e. from right to left. The first character has position 1. If the substring could not be found, then the result is 0.



8.9.17.15 right

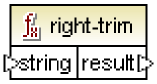
Result is a string containing the last **number** characters of **string.**



E.g. string="This is a sentence" and number=5, result is "tence".

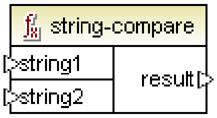
8.9.17.16 right-trim

Result is the input string with all trailing whitespace characters removed.



8.9.17.17 string-compare

Returns the result of a string comparison of **string1** with **string2** taking case into account. If string1=string2 then result is 0.

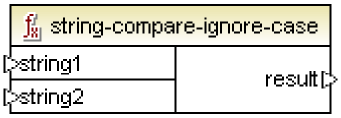


If string1 is smaller than string2 then result is < 0.

If string1 is larger than string2 then result is > 0

8.9.17.18 string-compare-ignore-case

Returns the result of a string comparison of string1 with string2 ignoring case. If string1=string2 then result is 0.

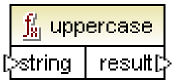


If string1 is smaller than string2 then result is < 0.

If string1 is larger than string2 then result is > 0.

8.9.17.19 uppercase

Result is the string input converted into uppercase. For Unicode characters the corresponding upper-case characters (defined by the Unicode consortium) are used.



8.9.18 xpath2 | accessors

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

8.9.18.1 base-uri

The `base-uri` function takes a node argument as input, and returns the URI of the XML resource containing the node. The output is of type `xs:string`. MapForce returns an error if no input node is supplied.

8.9.18.2 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` function (in the library of QName-related functions).

8.9.18.3 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.)

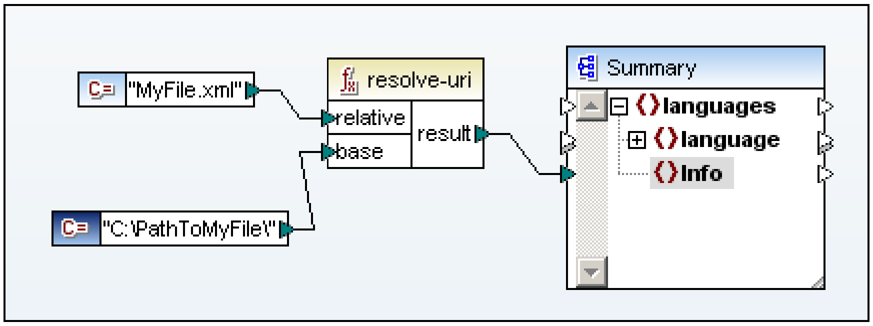
8.9.19 xpath2 | anyURI functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

8.9.19.1 resolve-uri

The `resolve-uri` function takes a URI as its first argument (datatype `xs:string`) and resolves it against the URI in the second argument (datatype `xs:string`).

The result (datatype `xs:string`) is a combined URI. In this way a relative URI (the first argument) can be converted to an absolute URI by resolving it against a base URI.



In the screenshot above, the first argument provides the relative URI, the second argument the base URI. The resolved URI will be a concatenation of base URI and relative URI, so `C:\PathtoMyFile\MyFile.xml`.

Note: Both arguments are of datatype `xs:string` and the process of combining is done by treating both inputs as strings. So there is no way of checking whether the resources identified by these URIs actually exist. MapForce returns an error if the second argument is not supplied.

8.9.20 xpath2 | boolean functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.
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.

8.9.20.1 false

Returns the Boolean value "false".

8.9.20.2 true

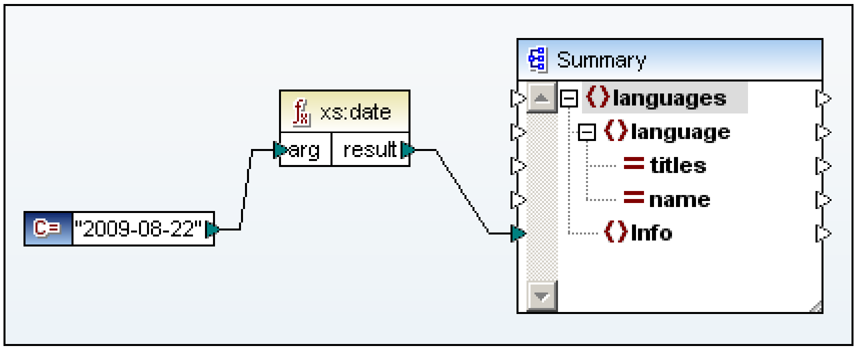
Returns the Boolean value "true".

8.9.21 xpath2 | constructors

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The functions in the Constructors part of the XPath 2.0 functions library construct specific datatypes from the input text. Typically, the lexical format of the input text must be that expected of the datatype to be constructed. Otherwise, the transformation will not be successful.

For example, if you wish to construct an `xs:date` datatype, use the `xs:date` constructor function. The input text must have the lexical format of the `xs:date` datatype, which is: `YYYY-MM-DD` (*screenshot below*).



In the screenshot above, a string constant (`2009-08-22`) has been used to provide the input argument of the function. The input could also have been obtained from a node in the source document.

The `xs:date` function returns the input text (`2009-08-22`), which is of `xs:string` datatype (specified in the *Constant* component), as output of `xs:date` datatype.

When you mouseover the input argument in a function box, the expected datatype of the argument is displayed in a popup.

8.9.22 xpath2 | context functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The Context functions library contains functions that provide the current date and time, the default collation used by the processor, and the size of the current sequence and the position of the current node.

8.9.22.1 current-date

Returns the current date (`xs:date`) from the system clock.

8.9.22.2 current-dateTime

Returns the current date and time (`xs:dateTime`) from the system clock.

8.9.22.3 current-time

Returns the current time (xs:time) from the system clock.

8.9.22.4 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.

The Altova XSLT 2.0 Engine supports the Unicode codepoint collation only. Comparisons, including for the fn:max and fn:min functions, are based on this collation.

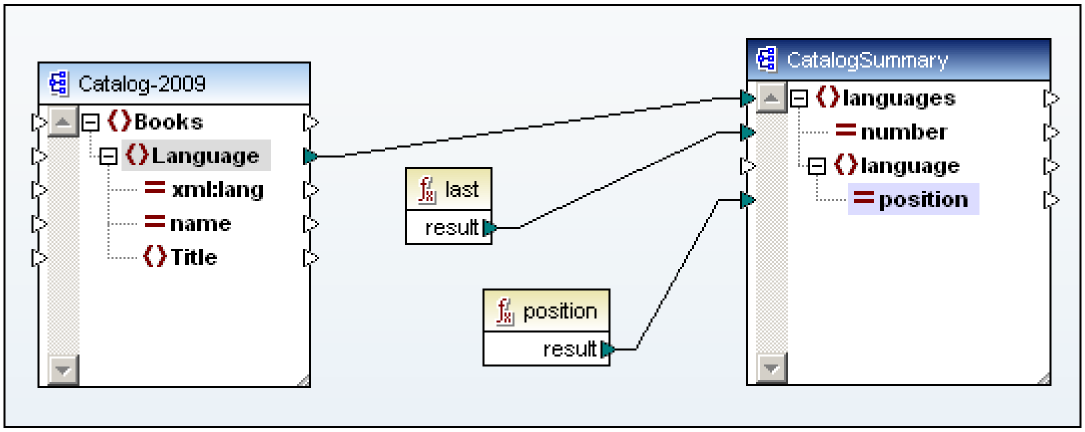
8.9.22.5 implicit-timezone

Returns the value of the "implicit timezone" property from the evaluation context.

8.9.22.6 last

The last and position functions take no argument. The last function returns the position of the last node in the context nodeset. The position function returns the position of the current node in the nodeset being processed.

The context nodeset at the nodes where the functions are directed, is the nodeset to which the functions will apply. In the screenshot below, the nodeset of Language elements is the context nodeset for the last and position functions.



In the example above, the last function returns the position of the last node of the context nodeset (the nodeset of Language elements) as the value of the number attribute. This value is also the size of the nodeset since it indicates the number of nodes in the nodeset.

The position function returns the position of the Language node being currently processed. For each Language element node, its position within the nodeset of Langauge elements is output to the language/@position attribute node.

We would advise you to use the **position** and **count** functions from the **core** library.

8.9.23 xpath2 | durations, date and time functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The XPath 2 duration and date and time functions enable you to adjust dates and times for the timezone, extract particular components from date-time data, and subtract one date-time unit from another.

The 'Adjust-to-Timezone' functions

Each of these related functions takes a date, time, or dateTime as the first argument and adjusts the input by adding, removing, or modifying the timezone component depending on the value of the second argument.

The following situations are possible when the first argument contains no timezone (for example, the date `2009-01` or the time `14:00:00`).

- Timezone argument (the second argument of the function) is present: The result will contain the timezone specified in the second argument. The timezone in the second argument is added.
- Timezone argument (the second argument of the function) is absent: The result will contain the implicit timezone, which is the system's timezone. The system's timezone is added.
- Timezone argument (the second argument of the function) is empty: The result will contain no timezone.

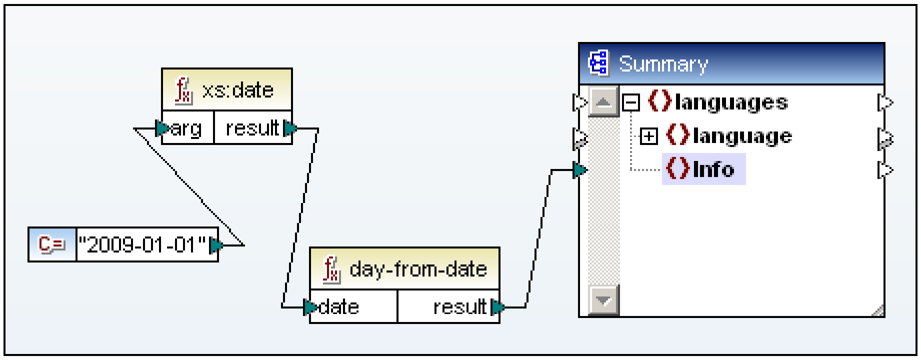
The following situations are possible when the first argument contains a timezone (for example, the date `2009-01-01+01:00` or the time `14:00:00+01:00`).

- Timezone argument (the second argument of the function) is present: The result will contain the timezone specified in the second argument. The original timezone is replaced by the timezone in the second argument.
- Timezone argument (the second argument of the function) is absent: The result will contain the implicit timezone, which is the system's timezone. The original timezone is replaced by the system's timezone.
- Timezone argument (the second argument of the function) is empty: The result will contain no timezone.

The 'From' functions

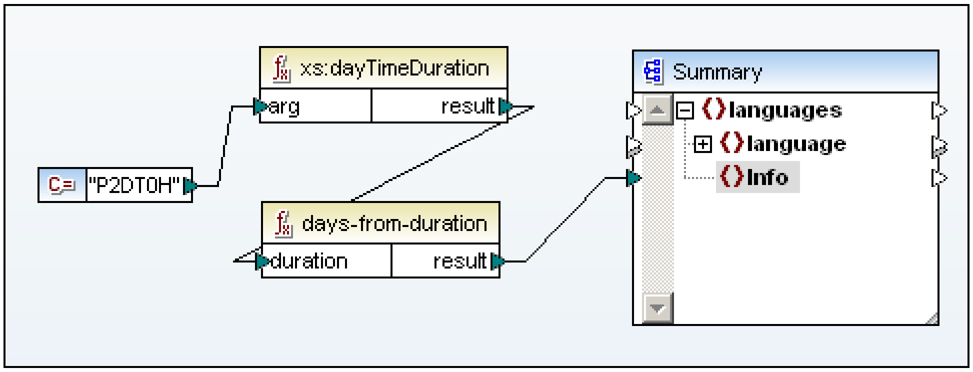
Each of the 'From' functions extracts a particular component from: (i) date or time data, and (ii) duration data. The results are of the xs:decimal datatype.

As an example of extracting a component from date or time data, consider the `day-from-date` function (*screenshot below*).



The input argument is a date (2009-01-01) of type xs:date. The day-from-date function extracts the day component of the date (1) as an xs:decimal datatype.

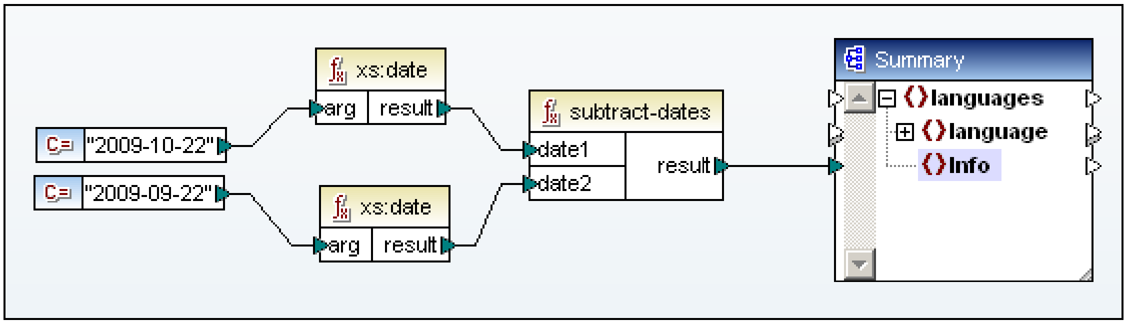
Extraction of time components from durations requires that the duration be specified either as xs:yearMonthDuration (for extracting years and months) or xs:dayTimeDuration (for extracting days, hours, minutes, and seconds). The result will be of type xs:decimal. The screenshot below shows a dayTimeDuration of P2DT0H being input to the days-from-duration function. The result is the xs:decimal 2.



The 'Subtract' functions

Each of the three subtraction functions enables you to subtract one time value from another and return a duration value. The three subtraction functions are: subtract-dates, subtract-times, subtract-dateTimes.

The screenshot below shows how the subtract-dates function is used to subtract two dates (2009-10-22 minus 2009-09-22). The result is the dayTimeDuration P30D.

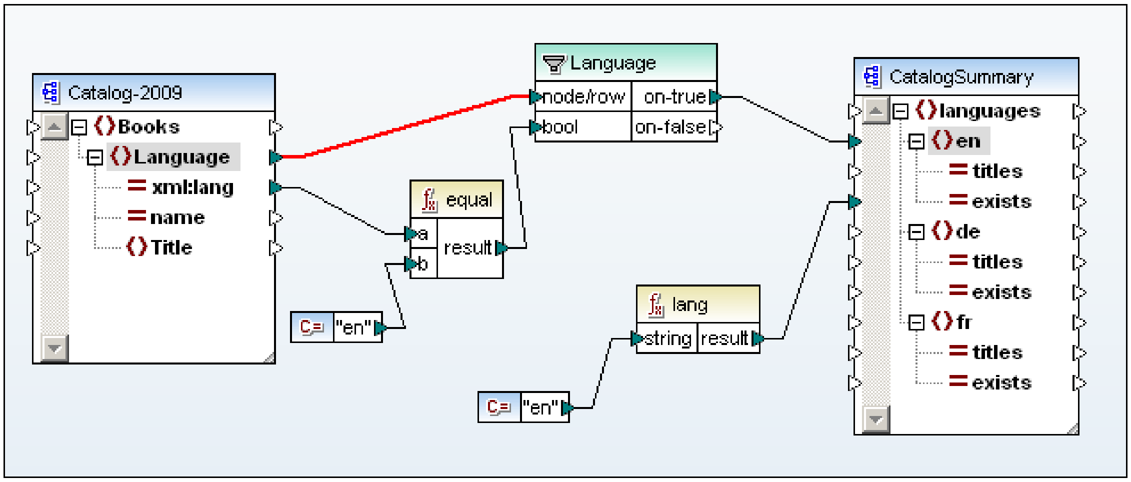


8.9.24 xpath2 | node functions

The following XPath 2 node functions are available:

lang

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.



In the screenshot above notice the following:

1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. `Language` nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the point where the `en` element is created in the output document.
4. The output of the `lang` function (`true` or `false`) is sent to the `en/@exists` attribute node of the output. The argument of the function is provided by the string constant `en`. The `lang` function then checks whether the context node at this point (the `Language` element) has an `xml:lang` attribute with a value of `en` (the argument of the function). If yes, then `true` is returned, otherwise `false`.

local-name, name, namespace-uri

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 (say, `http://www.altova.com/mapforce`).

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` function above).
- An argument that must be a node: the function is applied to the submitted node.

The output of each of these six variants is a string.

number

Converts an input string into a number. Also converts a boolean input to a number.

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. The only types that can be converted to numbers are booleans, strings, and other numeric types. Non-numeric input values (such as a non-numeric string) result in `NaN` (Not a Number).

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` function above).
- An argument that must be a node: the function is applied to the submitted node.

8.9.25 xpath2 | numeric functions

The following XPath 2 numeric functions are available:

abs

The `abs` function takes a numeric value as input and returns its absolute value as a decimal. For example, if the input argument is `-2` or `+2`, the function returns `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 `2.141`. 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`.

8.9.26 xpath2 | string functions

The following XPath 2 string functions are available:

compare

The `compare` function takes two strings as arguments and compares them for equality and alphabetically. If *String-1* is alphabetically less than *String-2* (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 *String-1* is greater than *String-2* (for example, `B` and `A`), then the function returns `+1`.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

ends-with

The `ends-with` function tests whether *String-1* ends with *String-2*. If yes, the function returns `true`, otherwise `false`.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

escape-uri

The `escape-uri` function takes a URI as input for the first string argument and applies the URI escaping conventions of RFC 2396 to the string. The second boolean argument (`escape-reserved`) should be set to `true()` if characters with a reserved meaning in URIs are to be escaped (for example "+" or "/").

For example:

`escape-uri("My A+B.doc", true())` would give `My%20A%2B.doc`
`escape-uri("My A+B.doc", false())` would give `My%20A+B.doc`

lower-case

The `lower-case` function takes a string as its argument and converts every upper-case character in the string to its corresponding lower-case character.

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.

The function takes an optional `flags` argument. Four flags are defined (`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 meaning of the four flags are as follows:

`i` Use case-insensitive mode. The default is case-sensitive.

`m` Use multiline 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.

`x` Ignore whitespace. By default whitespace characters are not ignored.

normalize-unicode

The `normalize-unicode` function normalizes the input string (the first argument) according to the rules of the normalization form specified (the second argument). The normalization forms NFC, NFD, NFKC, and NFKD are supported.

replace

The `replace` function takes the string supplied in the first argument as input, looks for matches as specified in a regular expression (the second argument), and replaces the matches with the string in the third argument.

The rules for matching are as specified for the matches attribute above. The function also takes an optional `flags` argument. The flags are as described in the `matches` function above.

starts-with

The `starts-with` function tests whether *String-1* starts with *String-2*. If yes, the function returns `true`, otherwise `false`.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

substring-after

The substring-after function returns that part of *String-1* (the first argument) that occurs after the test string, *String-2* (the second argument). An optional third argument specifies the collation to use for the string comparison. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

substring-before

The substring-before function returns that part of *String-1* (the first argument) that occurs before the test string, *String-2* (the second argument). An optional third argument specifies the collation to use for the string comparison. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

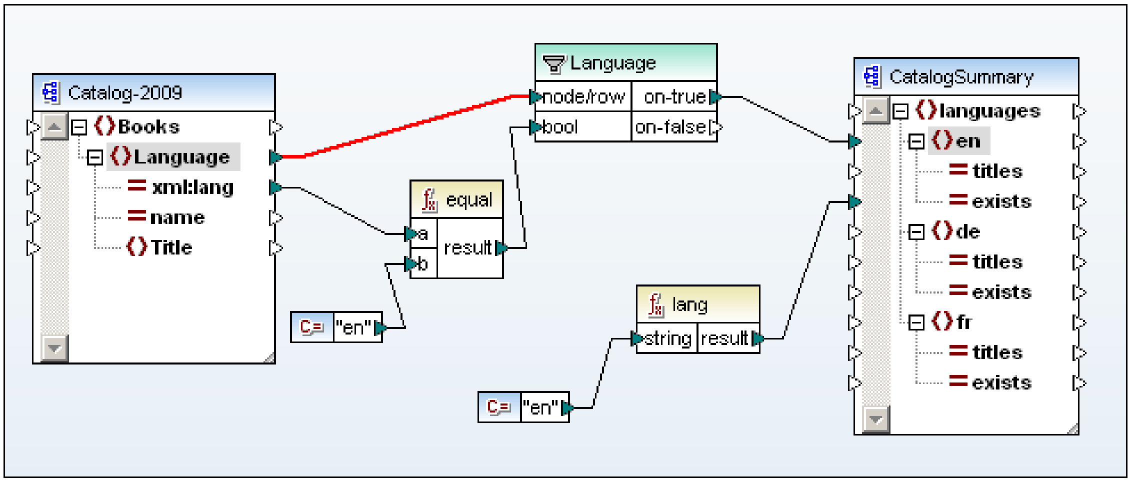
upper-case

The `upper-case` function takes a string as its argument and converts every lower-case character in the string to its corresponding upper-case character.

8.9.27 xslt | xpath functions

The functions in the XPath Functions library are XPath 1.0 nodeset functions. Each of these functions takes a node or nodeset as its context and returns information about that node or nodeset. These function typically have:

- a context node (in the screenshot below, the context node for the `lang` function is the Language element of the source schema).
- an input argument (in the screenshot below, the input argument for the `lang` function is the string constant `en`). The `last` and `position` functions take no argument.



lang

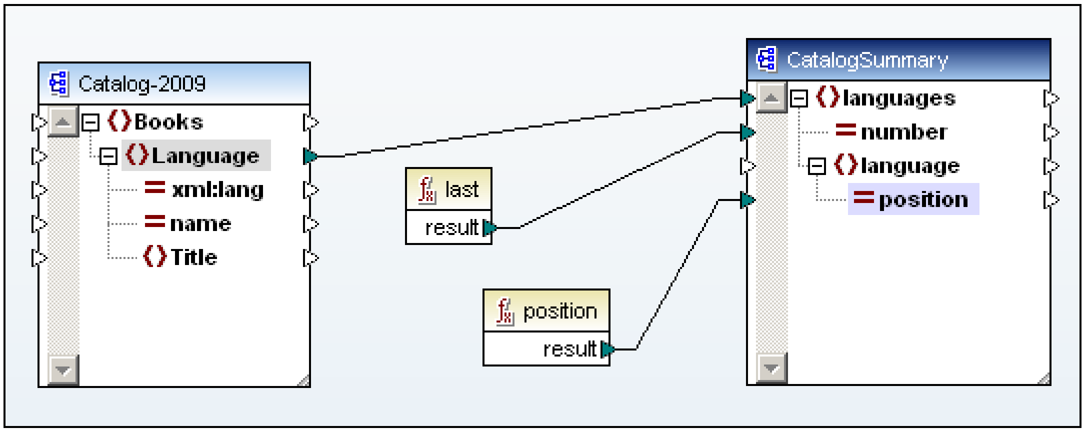
The `lang` function takes a string argument that identifies a language code (such as `en`). The function returns `true` or `false` depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function. In the screenshot above notice the following:

1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. `Language` nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the point where the `en` element is created in the output document.
4. The output of the `lang` function (`true` or `false`) is sent to the `en/@exists` attribute node of the output. The argument of the function is provided by the string constant `en`. The `lang` function then checks whether the context node at this point (the `Language` element) has an `xml:lang` attribute with a value of `en` (the argument of the function). If yes, then `true` is returned, otherwise `false`.

last, position

The `last` and `position` functions take no argument. The `last` function returns the position of the last node in the context nodeset. The `position` function returns the position of the current node in the nodeset being processed.

The context nodeset at the nodes where the functions are directed is the nodeset to which the functions will apply. In the screenshot below, the nodeset of `Language` elements is the context nodeset for the `last` and `position` functions.



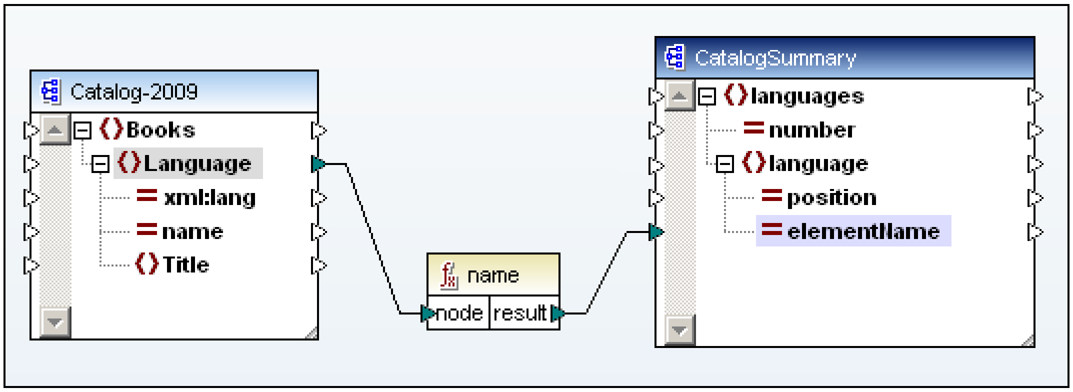
In the example above, the `last` function returns the position of the last node of the context nodeset (the nodeset of `Language` elements) as the value of the `number` attribute. This value is also the size of the nodeset since it indicates the number of nodes in the nodeset.

The `position` function returns the position of the `Language` node being currently processed. For each `Language` element node, its position within the nodeset of `Language` elements is output to the `language/@position` attribute node.

name, local-name, namespace-uri

These functions are all used the same way and return, respectively, the name, local-name, and namespace URI of the input node. The screenshot below shows how these functions are used. Notice that no context node is specified.

The name function returns the name of the `Language` node and outputs it to the `language/@elementname` attribute. If the argument of any of these functions is a nodeset instead of a single node, the name (or local-name or namespace URI) of the first node in the nodeset is returned.



The `name` function returns the QName of the node; the `local-name` function returns the local-name part of the node's QName. For example, if a node's QName is `altova:MyNode`, then `MyNode` is the local name.

The namespace URI is the URI of the namespace to which the node belongs. For example, the `altova:` prefix can be declared to map to a namespace URI in this way: `xmlns:altova="http://www.altova.com/namespaces"`.

Note: Additional XPath 1.0 functions can be found in the Core function library.

8.9.28 xslt | xslt functions

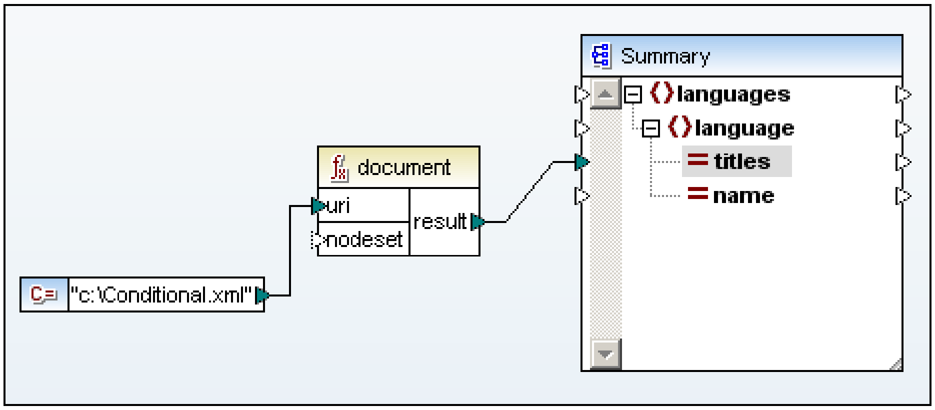
The functions in the XSLT Functions library are XSLT 1.0 functions.

8.9.28.1 currrent

The **current** function takes no argument and returns the current node.

8.9.28.2 document

The **document** function addresses an external XML document (with the `uri` argument; *see screenshot below*). The optional nodeset argument specifies a node, the base URI of which is used to resolve the URI supplied as the first argument if this URI is relative. The result is output to a node in the output document.

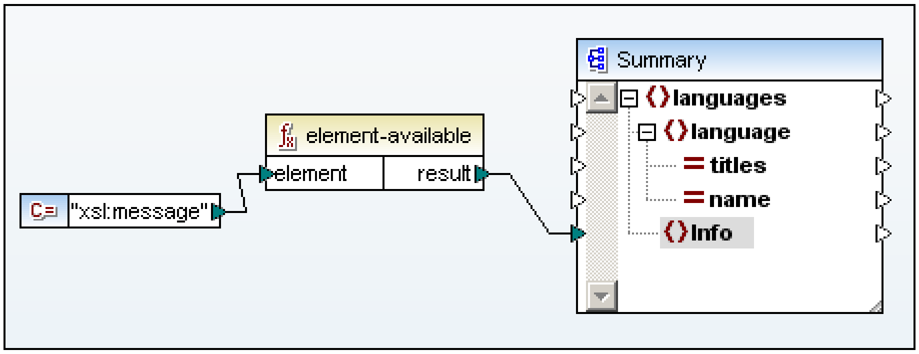


Note that the `uri` argument is a string that must be an absolute file path.

8.9.28.3 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.

8.9.28.4 *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.

8.9.28.5 *generate-id*

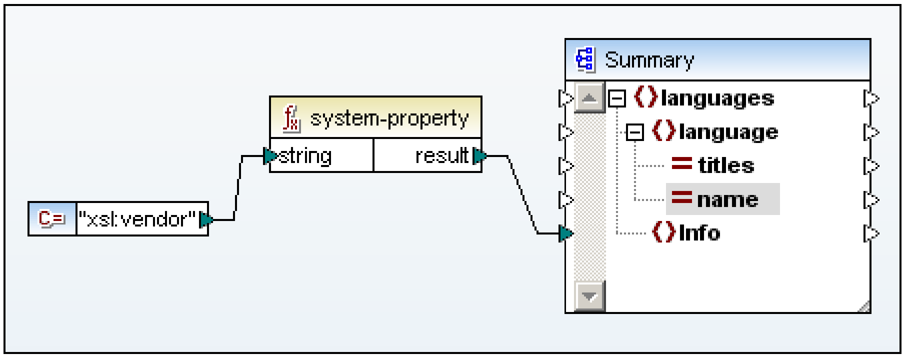
The **generate-id** function generates a unique string that identifies the first node in the nodeset identified by the optional input argument.

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.

8.9.28.6 *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.



8.9.28.7 unparsed-enity-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.

Chapter 9

Automating Mappings and MapForce

9 Automating Mappings and MapForce

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, or XQuery. See Automation with RaptorXML Server.
- *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.

** 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.

9.1 Automation with RaptorXML Server

RaptorXML Server (hereafter also called RaptorXML for short) is Altova's third-generation, super-fast XML and XBRL processor. It has been built to be 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. This edition is part of the FlowForce Server installation package.
- 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 1.0 or 2.0, 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 Server.

Note: You can also preview the XSLT and XQuery code using the built-in engine.

9.2 Automation with MapForce Server

MapForce Server is an enterprise server software solution for Windows, Linux and Mac OS X 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). 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.

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 the FlowForce Server administration interface. For further information, see Deploying Mappings to FlowForce Server.

There are two editions of MapForce Server:

- MapForce Server
- MapForce Server Advanced Edition (this edition is part of the FlowForce Server installation package)

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.

Limitations:

- XML digital signatures are not supported
- ADO, ADO.NET, and ODBC database connections are supported only on Windows (for other operating systems, see Database Connections on Linux and Mac).

9.3 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 OS X / 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).

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.
- 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.
- 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 and output files and schemas 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.
- 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, see "Linux and OS X / macOS considerations" 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, you can copy them manually to the target machine.
- Some mappings read multiple input files using a wildcard path (see Processing Multiple Input or Output Files Dynamically). 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).
- 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 .
- 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.

Linux and OS X / macOS considerations

If you intend to run the mapping on a Linux or OS X / macOS 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, see Using Relative and Absolute Paths. For example, you can copy all input or output files required by the mapping into the same directory as the mapping, and then reference them just by file name. 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. 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.

As for database connections, be aware that ADO, ADO.NET, and ODBC connections are not supported on Linux and OS X / macOS machines. Therefore, if the target machine is Linux or OS X / 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)
- In Mapforce, fill the JDBC database connection details in the "JDBC-specific Settings" section of the database component (see Database Component Settings).

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. If the mapping connects to a file-based database, such as Microsoft Access and SQLite, additional configuration is required, see "File-based databases" below.

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).
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. 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://manual.altova.com/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).

Global Resources

If a mapping includes references to Global Resources instead of direct paths or database connections, such references are preserved when you compile the mapping to a server execution file (.mfx), or when you deploy the mapping to FlowForce Server, see Global Resources in Various Execution Environments.

Note: FlowForce Server does not currently support Global Resources. Do not use Global Resources if you intend to execute the mapping with MapForce Server running under FlowForce Server management.

9.4 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 Server). 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).
- Alternatively, if MapForce Server runs under FlowForce Server management, the mapping can be deployed to a machine where both MapForce Server and FlowForce Server run (see Deploying Mappings to FlowForce Server).

Prerequisites

See Preparing Mappings for Server Execution.

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\MapForce2018\MapForceExamples\SimpleTotal.mfd** to a MapForce Server execution file that will be created in the target output directory **C:\Users\altova\Desktop**.

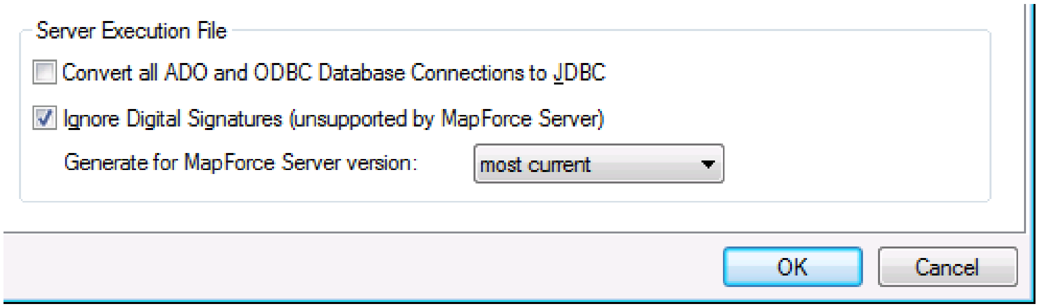
```
"C:\Program Files (x86)\Altova\MapForce2018\MapForce.exe" "C:\Users\altova
\Documents\Altova\MapForce2018\MapForceExamples\SimpleTotal.mfd" /COMPILE "C:
\Users\altova\Desktop"
```

See also the MapForce Command Line Interface.

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.



Server Execution File

□ Convert all ADO and ODBC Database Connections to JDBC

☑ Ignore Digital Signatures (unsupported by MapForce Server)

Generate for MapForce Server version: most current

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\MapForce2018\MapForce.exe" /COMPILE /
MFXVERSION:2018
```

See also the MapForce Command Line Interface.

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). The JDBC connection will be used implicitly if the target machine is a Linux or macOS server (see Database Connections on Linux and Mac).
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.

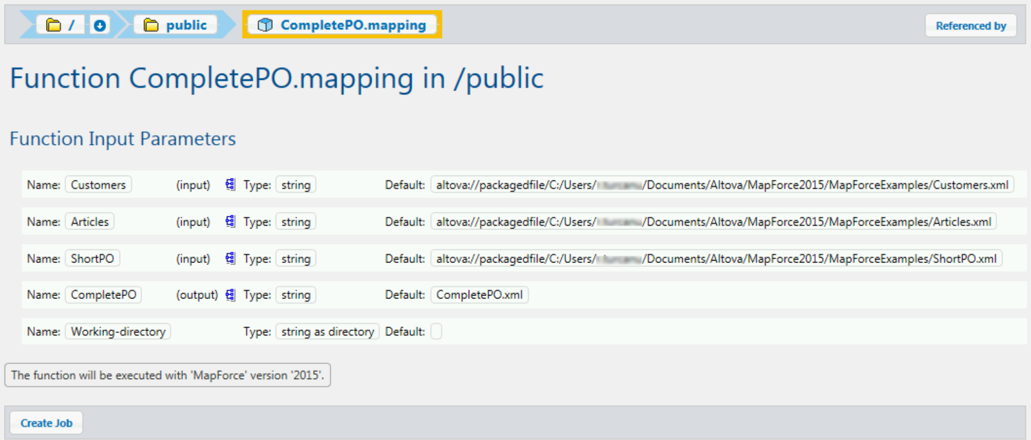
9.5 Deploying Mappings to FlowForce Server

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 OS X / 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.



Function CompletePO.mapping in /public

Function Input Parameters

Name:	Customers	(input)	Type:	string	Default:	altova://packagedfile/C:/Users/[redacted]/Documents/Altova/MapForce2015/MapForceExamples/Customers.xml
Name:	Articles	(input)	Type:	string	Default:	altova://packagedfile/C:/Users/[redacted]/Documents/Altova/MapForce2015/MapForceExamples/Articles.xml
Name:	ShortPO	(input)	Type:	string	Default:	altova://packagedfile/C:/Users/[redacted]/Documents/Altova/MapForce2015/MapForceExamples/ShortPO.xml
Name:	CompletePO	(output)	Type:	string	Default:	CompletePO.xml
Name:	Working-directory		Type:	string as directory	Default:	

The function will be executed with 'MapForce' version '2015'.

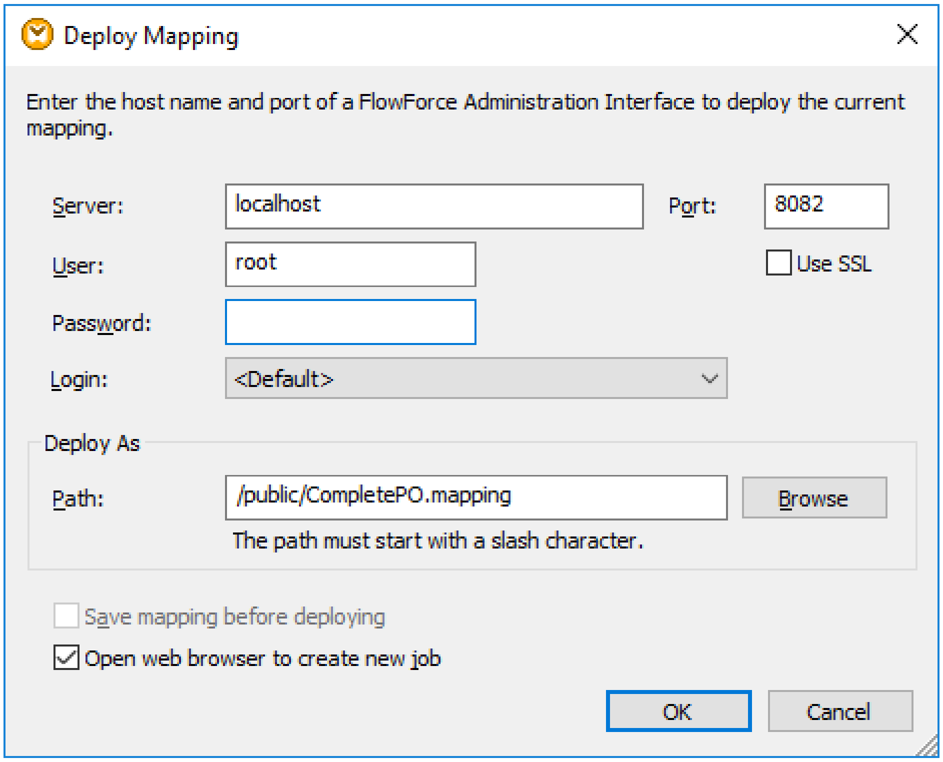
- All kinds of input instance files (XML, CSV, Text) that are used by the mapping.

Prerequisites

See Preparing Mappings for Server Execution.

Deploying the mapping to FlowForce Server

1. Ensure that the transformation language is set to BUILT-IN (see Selecting a Transformation Language).
2. On the **File** menu, click Deploy to **FlowForce Server**. The Deploy Mapping dialog box opens.



Deploy Mapping

Enter the host name and port of a FlowForce Administration Interface to deploy the current mapping.

Server: localhost

Port: 8082

User: root

□ Use SSL

Password:

Login: <Default>

Deploy As

Path: /public/CompletePO.mapping

The path must start with a slash character.

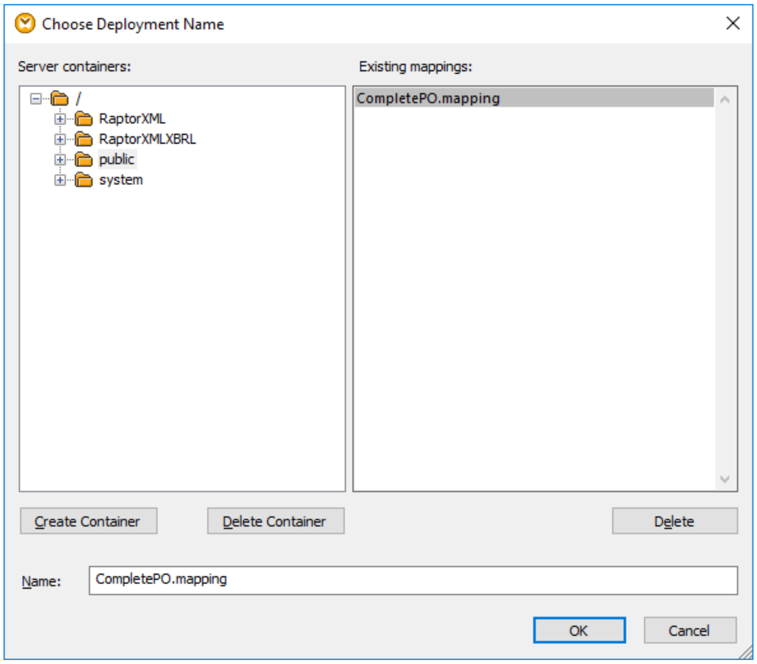
□ Save mapping before deploying

☑ Open web browser to create new job

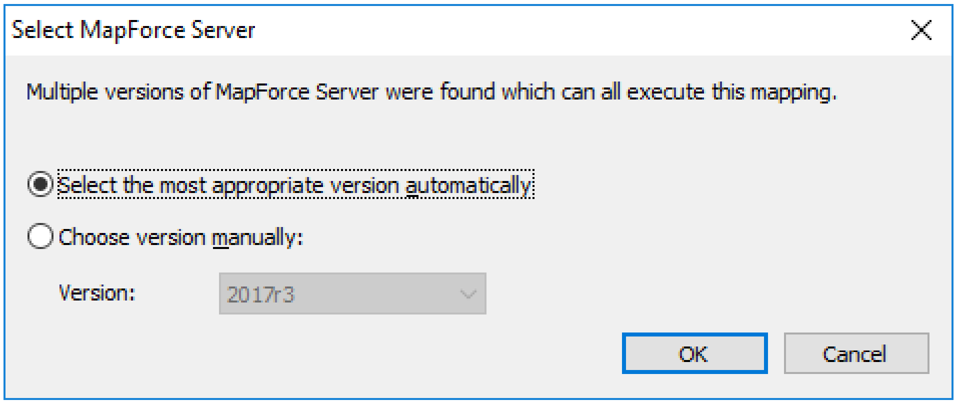
3. Enter your deployment settings (as described below), and click OK. If you selected the **Open web browser to create new job** check box, the FlowForce Server administration interface opens in the browser, and you can start creating a FlowForce Server job immediately.

The following table lists the mapping deployment settings available on the Deploy Mapping dialog box.

Setting	Description
Server and Port	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 configured to accept SSL connections. For more information, refer to FlowForce Server 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 Windows user name and password, and select the Windows domain name from the Login drop-down list.
Login	If Windows Active Directory integration is enabled in FlowForce Server, select the Windows domain name from this drop-down list, and enter your Windows credentials in the User and Password fields (see previous option).
Path	Click **Browse**, and select the path where the mapping function should be saved in FlowForce Server container hierarchy. By default, the path is set to the **/public** container of FlowForce Server. From the Choose Deployment Name 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.  Choose Deployment Name Server containers: Existing mappings: / RaptorXML RaptorXMLXBRL public system CompletePO.mapping Create Container Delete Container Delete OK Cancel Name: CompletePO.mapping *Choose Deployment Name dialog box*
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.

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.



Select MapForce Server

Multiple versions of MapForce Server were found which can all execute this mapping.

● Select the most appropriate version automatically

○ Choose version manually:

Version: 2017r3

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\FlowForceServer2018\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\MapForceServer2018\etc**.

9.6 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.

Target	Description
	`NOXMLSIGNATURES` Suppresses the generation of digital signatures in the MapForce Server Execution file (note that digital signatures are not supported by MapForce Server).
`/GENERATE`	Generates project code for all mappings in the project file using the current folder settings, see Managing Project Folders. 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.
`/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[:{VS2008|VS2010|VS2013|VS2015|VS2017}]` `VS2008` Visual Studio 2008 `VS2010` Visual Studio 2010 `VS2013` Visual Studio 2013 `VS2015` Visual Studio 2015 `VS2017` Visual Studio 2017 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.
`/CPP`	Generates C++ code. This command also optionally allows setting specific code generation options, namely: `/CPP[:{VS2008|VS2010|VS2013|VS2015|VS2017},{MSXML|XERCES3},{LIB|DLL},{MFC|NoMFC}]` The first option group set the target Visual Studio version. Valid values: `VS2008` Visual Studio 2008 `VS2010` Visual Studio 2010

Target	Description
	`VS2013` Visual Studio 2013 `VS2015` Visual Studio 2015 `VS2017` Visual Studio 2017 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.

<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.

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.
`/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. 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. These libraries are temporarily (for this one run) added to the libraries from **Tools | Options | Libraries**.
`/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 only specify the file name, then the file will be placed in the `<outputdir>` directory.

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).

12) To compile a mapping to a MapForce Server execution file, for MapForce Server version 2018, and suppress XML signatures:

```
MapForce.exe <filename>.mfd /COMPILE:NOXMLSIGNATURES <outputdir> /
MFXVERSION:2018 /LOG <logfilename>
```

Chapter 10

Customizing MapForce

10 Customizing MapForce

This section provides information about working with Altova Global Resources, customizing the mapping output, generating and customizing mapping documentation, and working with catalog files.

10.1 Changing the MapForce Options

You can change the general and other preferences in MapForce as follows:

- On the **Tools** menu, click **Options**.

The available options are grouped as shown below.

Libraries

From this page, you can add or delete custom function libraries to MapForce. For more information, see Importing Custom XSLT 1.0 or 2.0 Functions, Importing Custom XQuery 1.0 Functions, Importing Custom Java and .NET Libraries).

General

The settings available in this page are as follows:

Show logo | Show on start	Shows or hides an image (splash screen) while MapForce starts.
Show gradient background	Enables or disables the gradient background in the Mapping pane.
Limit annotation display to N lines	This option applies to components which support annotations (for example, XML schema, EDI). If the annotation text contains multiple lines, then enabling this option shows only the first N lines on the component, where N is the value you specify. This setting also applies to SELECT statements visible in a component.
Encoding name	Sets the default character encoding for new components. This setting can also be changed individually for each component, see Changing the Component Settings.
Use execution timeout	Sets an execution timeout when previewing the mapping result in the **Output** pane.
Generate output to temporary files	When this option is set, the output generated when you preview the mapping result will be written 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. **Warning**: 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.
Write directly to final output files	When this option is set, the output generated when you preview the mapping result will be written to actual files. If the output file path contains folders that do not exist yet, then a mapping error occurs. **Warning**: This option overwrites any existing output files

Show logo | Show on start	Shows or hides an image (splash screen) while MapForce starts.
	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 click a **Load more** button to load the next chunk. For more information, see Previewing the Output.

Editing

The settings available in this page are as follows:

Align components on mouse dragging	Specify whether components or functions should be aligned with other components, while you drag them with the mouse, see Aligning Components.
Smart component deletion	When enabled, this option "remembers" connections of deleted components, see Keeping Connections After Deleting Components.

Messages

From this page, you can re-enable message notifications that were previously disabled using the "Do not show this message again" option.

Generation

From this page, you can define settings applicable when you generate program code and MapForce Server Execution files. For more information, see Code Generator Options and Compiling Mappings to MapForce Server Execution Files, respectively.

Database

From this page, you can define settings applicable when querying databases in the **DB Query** tab (see Database Query Settings).

Network proxy

See Network Proxy Settings.

10.2 Altova Global Resources

Altova Global Resources represent a way to refer to files, folders, or databases so as to make these resources reusable, configurable and available across multiple Altova applications. For example, let's assume that several MapForce mappings routinely read data from the same XML file which is critical for your business workflow. If this file has been renamed on the disk for whatever reason, this would cause "file not found" errors in multiple contexts, and break the workflow. To prevent such issues, it is possible to create a so-called "file alias" (in other words, a Global Resource), and change all mappings to refer to this Global Resource instead of the actual file on disk. This way, if the file name ever changes, you would only need to change the file alias, in one place.

Global Resources can be defined and shared between the following Altova desktop applications: Authentic, MobileTogether Designer, MapForce, DatabaseSpy, and XMLSpy. On the server side, Global Resources can be consumed by the following Altova server applications: MapForce Server, MapForce Server Advanced Edition, RaptorXML Server, RaptorXML+XBRL Server.

Global Resources (be they file, folder, or database references) can be used in MapForce for various scenarios, for example:

- To supply a configurable file path as mapping input, see Example: Run Mappings with Variable Input Files.
- To redirect the mapping output to a configurable path. For more information, see Example: Generate Mapping Output to Variable Folders.
- To supply a configurable path to a StyleVision Power Stylesheet (.sps) file, if one is used by the mapping. For example, instead of referencing a plain StyleVision .sps file from the MapForce component settings, you could refer to an .sps file previously defined as a Global Resource, which has two possible configurations (let's say, "Website.sps" and "Print.sps"). See also Styling Mapping Output with StyleVision.
- To reuse a database connection. If a database has already been defined as a Global Resource (in any Altova application), you can connect to it without going through all the set-up steps again, see Using a Connection from Global Resources.
- To easily switch the database from which the mapping reads data, or the one to which the mapping writes data (provided that two or more databases have the same structure but different data, see Example: Switch Databases).

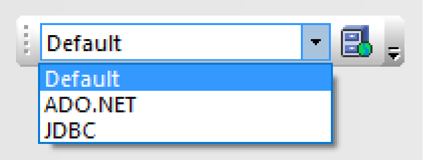
Note:

- FlowForce Server does not support Global Resources. MapForce Server can consume Global Resources either at the command line or at API level.
- MapForce Basic Edition does not support consuming database connections defined as Global Resources.

10.2.1 Creating Global Resources

A Global Resource alias is a reusable reference which represents a file or folder path, or a database connection. Aliases are defined only once and can be reused as many times as necessary in contexts which support them, including across multiple Altova applications. Taking databases as example, if you frequently work with a specific database in more than one Altova application, then it is a good idea to add the database connection as a Global Resource. This way, you wouldn't need to go through all the Database Connection Wizard steps each time when you need to connect to the same database from another Altova application.

File, folder, and database aliases are configurable in their turn, by means of so-called "configurations". Configurations make it possible to easily switch between files, folders and databases that are consumed or produced by Altova applications, which is particularly useful for testing scenarios. For example, you could create a database alias that consists of three separate connections to the same database, each with a different driver kind: (a) ODBC, the default connection kind, (b) JDBC, and (c) ADO.NET. This way, to connect to the database with a specific driver, you would just select the corresponding configuration from the Global Resources drop-down list before running the mapping.



Global Resources drop-down list

Configurations can also help you generate mapping output to variable folders, with a click of a button. For example, you could create a folder alias with two configurations: (a) "Testing", which points to directory **C:\Testing** and (b) "Production", which points to directory **C:\Production**. It is then possible to configure a mapping to generate output to either **C:\Testing** or **C:\Production** folders, just by selecting the required configuration from the Global Resources drop-down list before running the mapping. This example is discussed in more detail in Example: Generate Output to Variable Folders.

How to create a Global Resource alias

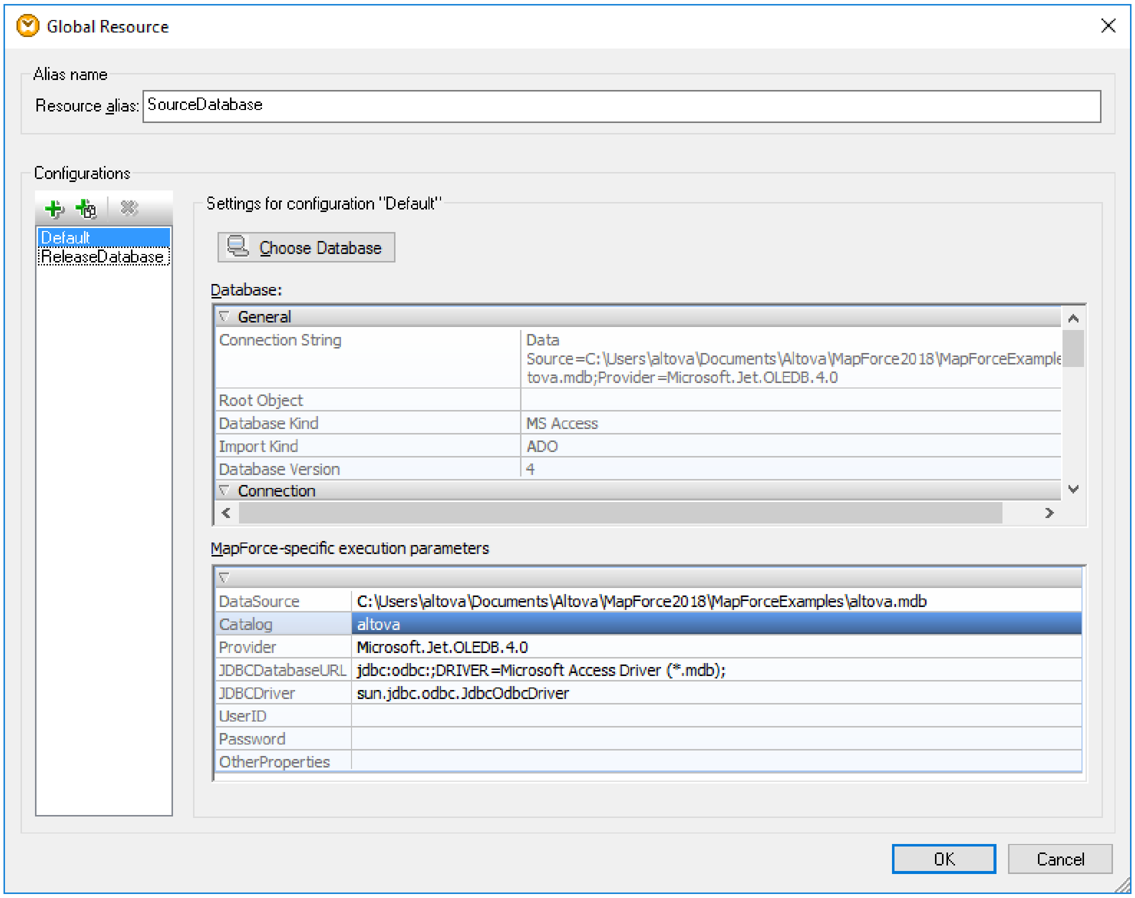
1. On the **Tools** menu, click **Global Resources**. (Alternatively, click the **Global Resource** toolbar button.)
2. Click **Add** and select the resource type you wish to create (file, folder, database).
3. Enter a descriptive name for this alias in the **Resource alias** text box (for example, "MappingInputFile", "MappingOutputFolder", "DatabaseConnection").
4. Set up the "Default" configuration:
   a) If it's a file or folder, browse for the file or folder to which this resource should point by default.
   b) If it's a database connection, click **Choose Database** and follow the Database Connection Wizard to connect to the database (see Connecting to a Database). This database connection will be used by default when the mapping runs (unless a different configuration is explicitly selected from the Global Resources drop-down list or supplied as a command line parameter in server execution).
5. Optionally, if the resource should have an additional configuration (for example, a driver kind in case of databases, or an alternative path in case of files or folders), click the **Add configuration** button, enter a descriptive name (for example "ProductionFolder" or "JDBC_Alternative"), and set it up as follows:
   a) If it's a file or folder, browse for the file or folder to which this resource should point as an alternative to the default configuration defined in previous step.
   b) If it's a database connection, follow the Database Connection Wizard to connect to the database. This database connection will be used as an alternative to the default one.

In some cases, it might be more convenient to create a configuration as a copy of the default configuration, and then edit it. In this case, click the **Add configuration as a copy of the currently selected configuration** button.

6. Repeat the previous step for each additional configuration required.

10.2.2 Databases as Global Resources

When you add a database connection as a Global Resource, the database connection parameters are automatically populated on the Global Resource dialog box.



Resource alias: SourceDatabase

Configurations: Default, ReleaseDatabase

Settings for configuration "Default"

Database:

General	
Connection String	Data Source=C:\Users\altova\Documents\Altova\MapForce2018\MapForceExample tova.mdb;Provider=Microsoft.Jet.OLEDB.4.0
Root Object	
Database Kind	MS Access
Import Kind	ADO
Database Version	4
Connection	

MapForce-specific execution parameters

DataSource	C:\Users\altova\Documents\Altova\MapForce2018\MapForceExamples\altova.mdb
Catalog	altova
Provider	Microsoft.Jet.OLEDB.4.0
JDBCDatabaseURL	jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);
JDBCDriver	sun.jdbc.odbc.JdbcOdbcDriver
UserID	
Password	
OtherProperties	

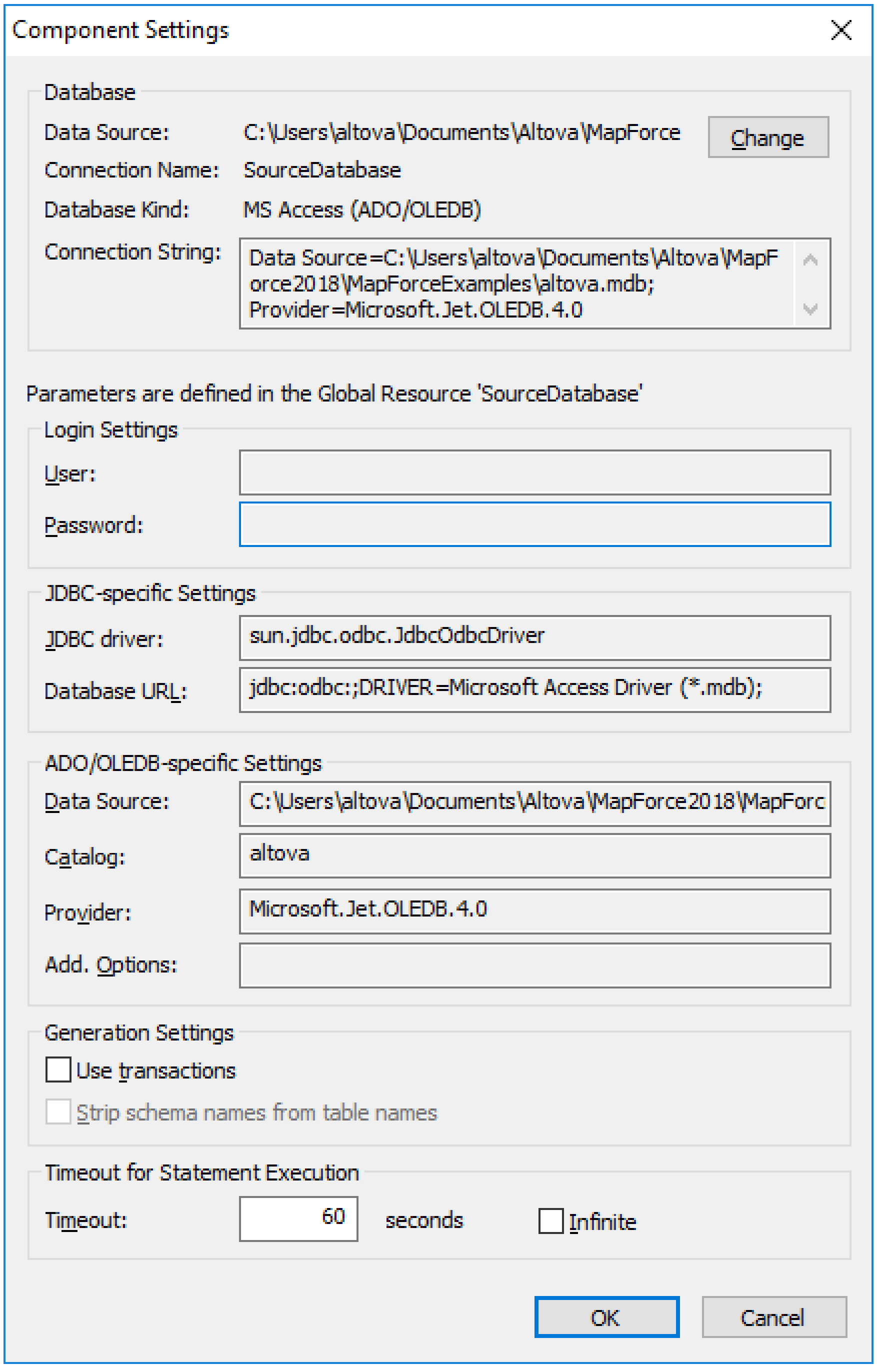
Global Resource dialog box

On the Global Resource dialog box, it is possible to edit some of the database connection parameters. As illustrated above, the parameters are grouped into two categories:

Database	These parameters are shared between Altova applications. In MapForce, they are used at design time, that is, when the mapping is loaded, or when you click the **Output** tab 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 compiled the mapping to a MapForce Server execution file, and automatic JDBC conversion took place. For more information about automatic JDBC conversion, see Database mappings in various execution environments.

If a mapping uses a Global Resource to connect to a database, then the database connection details visible in the Global Resource dialog box take precedence over those defined on the mapping. In the Component Settings dialog box, illustrated below, notice that the database connection settings become grayed out. The dialog box also informs you that the connectivity parameters are defined as a Global Resource.



Database

Data Source: C:\Users\altova\Documents\Altova\MapForce

Connection Name: SourceDatabase

Database Kind: MS Access (ADO/OLEDB)

Connection String: Data Source=C:\Users\altova\Documents\Altova\MapForce2018\MapForceExamples\altova.mdb; Provider=Microsoft.Jet.OLEDB.4.0

Parameters are defined in the Global Resource 'SourceDatabase'

Login Settings

User:

Password:

JDBC-specific Settings

JDBC driver: sun.jdbc.odbc.JdbcOdbcDriver

Database URL: jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);

ADO/OLEDB-specific Settings

Data Source: C:\Users\altova\Documents\Altova\MapForce2018\MapForc

Catalog: altova

Provider: Microsoft.Jet.OLEDB.4.0

Add. Options:

Generation Settings

☐ Use transactions

☐ Strip schema names from table names

Timeout for Statement Execution

Timeout: 60 seconds ☐ Infinite

Component Settings dialog box

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.

10.2.3 MapForce and StyleVision Transformation Result as Global Resource

It is also possible to create Global Resources which, instead of pointing to a static file, read a specific file produced by either a MapForce mapping or StyleVision transformation. In this case, the Altova application which consumes the Global Resource will first call either MapForce or StyleVision, run the corresponding mapping or transformation, and finally fetch the resulting file. This makes it possible to define data workflows between Altova applications (for example, pass the result of a MapForce mapping or StyleVision transformation as input to another mapping or transformation). For an example which illustrates how XMLSpy consumes the result (output) of a MapForce mapping using Global Resources, see Example: Create Application Workflow.

> Note:
>
> - In order to make a mapping result (output) available as a Global Resource, either 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 (for example, some XSLT functions are not supported by the BUILT-IN language.)
> - MapForce Basic Edition does not support providing mapping transformation results as Global Resources.

10.2.4 The Global Resources XML File

By default, all Global Resources, regardless of the Altova application where they were created, are stored at the following path: **C:\Users\Documents\Altova\GlobalResources.xml**. This makes them transparent, easy to backup, as well as portable to other workstations where Altova products are installed. It is also possible to rename or duplicate the **GlobalResources.xml** file and thus create multiple Global Resource files. However, only one Global Resource file can be active at a time in an Altova application.

To set up the active Global Resource file:

1. On the **Tools** menu, click **Global Resources**. (Alternatively, click the **Global Resource** toolbar button.)
2. Click **Browse** and select the required Global Resource XML file.

> If you are using multiple Global Resource files, make sure that the currently active Global Resource file contains all Global Resources required to run the mapping. For example, if a mapping was configured to read data from a path using a Global Resource, then the currently active Global Resource file must contain that specific Global Resource. Otherwise, error messages like "Errors resolving global resource" will occur in the **Messages** window.

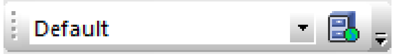
10.2.5 Global Resources in Various Execution Environments

A mapping using Global Resources will behave differently in each environment where it is run, as shown below.

Global Resources in XSLT, XSLT2, 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, the input and output files are not a permanent assignment and can be specified flexibly anyway when you run the XSLT stylesheet in your XSLT processor. The same applies for generated XQuery code.

An exception to this is the **DoTransform.bat** file generated for RaptorXML execution. Any Global Resources used by the mapping will be resolved to actual paths in **DoTransform.bat**, taking into account the value (configuration) which is currently selected from the Global Resource drop-down list.

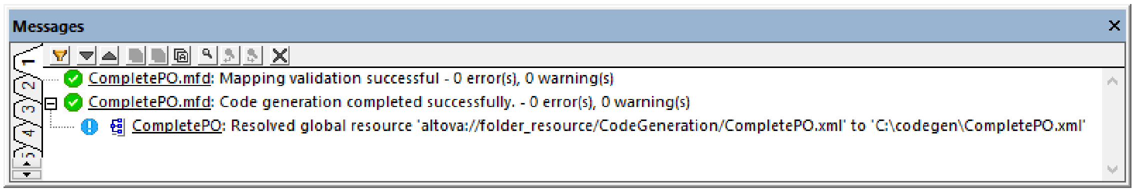


Default

For information about supplying Global Resources to RaptorXML, refer to the RaptorXML documentation (see https://www.altova.com/documentation.html).

Global Resources in C#, C++, Java

When you generate C#, C++, or Java program code, any Global Resources used by the mapping will be resolved. For example, a file or folder alias defined as Global Resource will be converted to the actual file or folder path. If a particular Global Resource configuration is selected from the Global Resources drop-down list, then the code will be generated for the selected configuration. The **Messages** window provides information as to how exactly a Global Resource was resolved, for example:



```
CompletePO.mfd: Mapping validation successful - 0 error(s), 0 warning(s)
CompletePO.mfd: Code generation completed successfully. - 0 error(s), 0 warning(s)
    CompletePO: Resolved global resource 'altova://folder_resource/CodeGeneration/CompletePO.xml' to 'C:\codegen\CompletePO.xml'
```

To generate code for a particular Global Resource configuration, select it from the Global Resource 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 (see also MapForce Command Line Interface).

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 not only the path but also the data type of input or output, see Changing the data type of the mapping input/output (C#, Java).

Global Resources in MapForce Server

When you compile a mapping to a MapForce Server execution file (.mfx), any Global Resources references used by the mapping are preserved as such. In MapForce Server, the following is required to run an .mfx file compiled from a mapping which uses Global Resources:

1. The path to the Global Resource XML file (that is, the file where Global Resources are defined, see The Global Resources XML File).
2. The Global Resource configuration name. The name of the default configuration is "Default". If you created additional configurations, as explained in Creating Global Resources, then the desired configuration must be called by its corresponding name.

The Global Resource file path and the name of the configuration can be specified as follows:

- 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\MapForceServer2018\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`** two times before calling the **`Run`** method. The first call is required to supply the Global Resource XML file path as option, and the second one is required to supply the Global Resource configuration name.

For more information, refer to the MapForce Server documentation (see https://www.altova.com/documentation.html).

10.2.6 Example: Run Mapping with Variable Input Files

Let's assume that, as part of your job duties, you frequently run a mapping that takes as input an XML file. Under normal circumstances, whenever you want to change the input XML of the mapping, you can open the properties of the source XML component and browse for the new input file, see Changing the Component Settings. This is easy to accomplish if it's a one time task. However, what if you need to change the input XML file of the mapping multiple times per day, or even per hour? For example, every morning you need to run the mapping and generate a report by using one XML file as mapping input, and every evening the same report must be generated from another XML file. This is where Global Resources can help you: 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 (a so-called "file alias"). To address the requirement laid out in this example, the file alias could be configured to have two configurations:

1. "Default" - This configuration would supply a "morning" XML file as mapping input
2. "EveningReports" - This configuration would supply an "evening" XML file as mapping input.

Having these configurations in place would make it possible to run the mapping with either input file. Once the file alias is set up as shown below, you will be able to select the desired configuration from a drop-down list, before running the mapping.

Step 1: Create the Global Resource

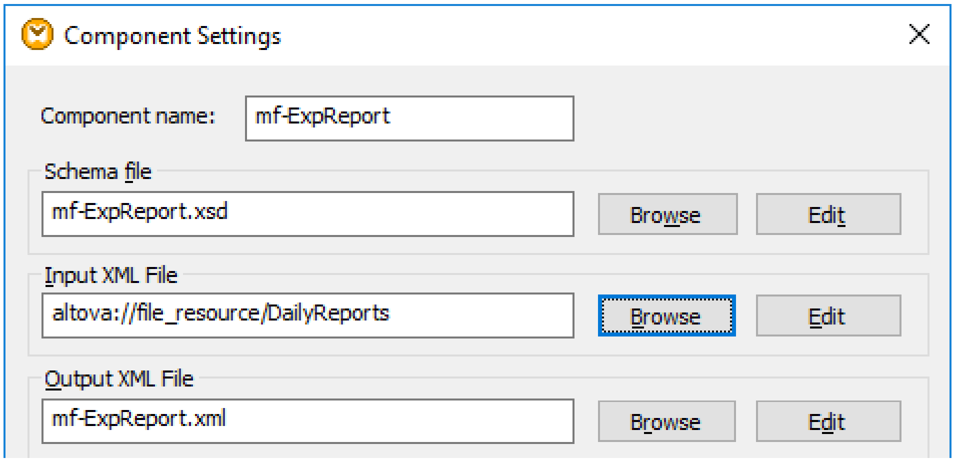
The file alias can be created as follows:

1. On the **Tools** menu, click **Global Resources**. (Alternatively, click the **Global Resource** toolbar button.)
2. Click **Add | File**.
3. Enter a name in the **Resource alias** text box (in this example, "DailyReports" would be an appropriate name).
4. Click **Browse** and select the following file: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\mf-ExpReport.xml**.
5. Click **Add Configuration** ✚ and name it "EveningReports".
6. Click **Browse** and this time select the following file: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\mf-ExpReport2.xml**.

Step 2: Use the Global Resource in the mapping

The required Global Resource has now been created; however, the mapping is not using it yet. To change the mapping so that it reads from the previously defined file alias (Global Resource), do the following:

1. Open the following mapping **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\Tut-ExpReport.mfd**.
2. Right-click the header of the source component on the mapping, and select **Properties** from the context menu.
3. Next to **Input XML file**, click **Browse**.
4. Click **Switch to Global Resources** and select the file alias "DailyReports" defined previously.
5. Click **Open**. The input XML file path has now become **altova://file_resource/DailyReports**, which indicates that the path uses a Global Resource.


Component Settings

Component name: mf-ExpReport

Schema file
mf-ExpReport.xsd

Input XML File
altova://file_resource/DailyReports

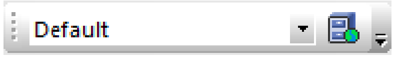
Output XML File
mf-ExpReport.xml

Step 3: Run the mapping with the desired configuration

You can now easily switch the input XML file before running the mapping, as follows:

- On the **Tools** menu, click **Active Configuration | Default**, to use the file **mf-ExpReport.xml** as input.
- On the **Tools** menu, click **Active Configuration | EveningReports**, to use the file **mf-ExpReport2.xml** as input.

Alternatively, select the required configuration from the **Global Resources** drop-down list.


Default

To preview the mapping result with either configuration, click the **Output** tab and observe differences in the generated output.

10.2.7 Example: Generate Output to Variable Folders

This example illustrates how mapping output can be redirected to different folders by means of Global Resources.

Let's suppose that sometimes you need to generate the mapping output to one directory (for example, **C:\Testing**), while in certain cases output must be generated to another directory (for example, **C:\Production**). With Global Resources, this is possible by creating a folder alias with two configurations:

1. "Default" configuration - Generates output to **C:\Testing**
2. "Production" configuration - Generates output to **C:\Production**.

The steps below illustrate how to achieve this goal.

Step 1: Create the Global Resource

The folder alias can be created as follows:

1. On the **Tools** menu, click **Global Resources**. (Alternatively, click the **Global Resource** toolbar button.)
2. Click **Add | Folder**.
3. Enter a name in the **Resource alias** text box (in this example, "OutputDirectory" could be an appropriate name).
4. Click **Browse** and select the following folder: **C:\Testing**. (Make sure that this folder already exists on your operating system.)
5. Click **Add Configuration** and enter a name for the new configuration (in this example, "ProductionDirectory").
6. Click **Browse** and this time select the following folder: **C:\Production**. (Make sure that this folder already exists on your operating system.)

Step 2: Use the Global Resource in the mapping

The required Global Resource has now been created; however, the mapping is not using it yet. To change the mapping so that it uses from the previously defined folder alias (Global Resource), do the following:

1. Open the following mapping **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\Tut-ExpReport.mfd**.
2. Right-click the target component on the mapping, and select **Properties** from the context menu.
3. Next to **Output XML file**, click **Browse**.
4. Click **Switch to Global Resources**, and then click **Save**.
5. When prompted to save the output XML file, enter **output.xml** (or another descriptive file name that you wish to give to the output file). 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 easily switch to the desired mapping output folder file before running the mapping, as follows:

- On the **Tools** menu, click **Active Configuration | Default**, and then click the **Output** tab to preview the mapping result. The mapping output (either a temporary or a permanent file, as explained below) will be generated in the **C:\Testing** directory.
- On the **Tools** menu, click **Active Configuration | ProductionDirectory**, and then click the **Output** tab. The mapping output (either a temporary or a permanent file, as explained below) will be generated in the **C:\Production** directory.

Note: The mapping output is written by default as a temporary file, unless you explicitly configured MapForce to write output to permanent files.

To configure MapForce to generate permanent files instead of temporary, do the following:

1. On the **Tools** menu, click **Options**.
2. In the **General** section, select the option **Write directly to final output files**.

10.2.8 Example: Switch Databases

When a mapping reads or writes data from a database, it is possible to switch the database connection immediately before mapping runtime (for example, from a release to a production database, and vice versa). This example illustrates how to accomplish this by means of Global Resources. Switching databases this way implies that both databases have the same structure but different data. For the purpose of this example, we will be working with the following Microsoft Access databases:

- **altova.mdb**, from the directory: **<Documents>\Altova\MapForce2018\MapForceExamples\**. This database plays the role of the default development database.
- **altova.mdb**, from the directory: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\**. This database plays the role of the production database.

The requirement is to easily supply to the mapping either of the two databases immediately before runtime, without editing the mapping. To achieve this requirement, we will create a database Global Resource (database alias) with two configurations:

1. *Default*. This configuration will point to the default development database.
2. *Release*. This configuration will point to the release database.

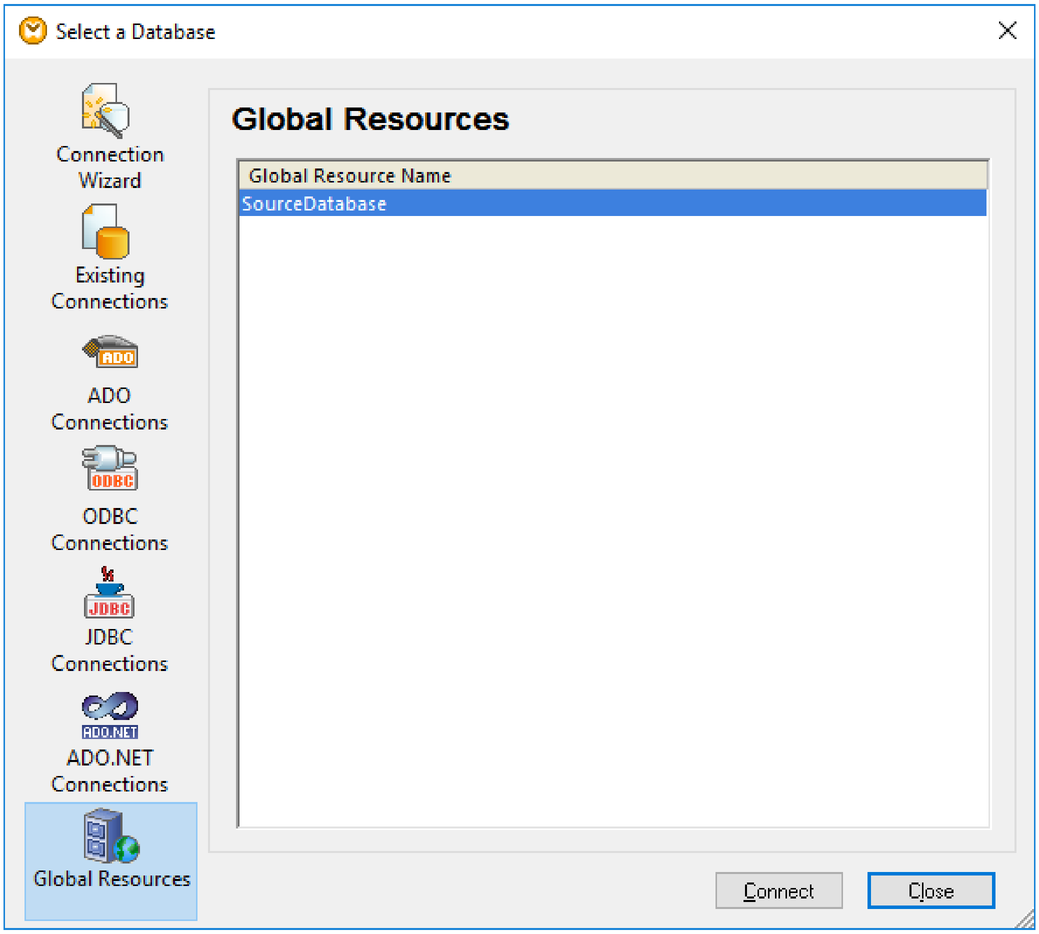
Step 1: Create the Global Resource (database alias)

1. On the **Tools** menu, click **Global Resources**. (Alternatively, click the **Global Resource** toolbar button.)
2. Click **Add | Database**.
3. Enter a descriptive name in the **Resource alias** text box (for example, "SourceDatabase").
4. Click **Choose Database**, select **Microsoft Access (ADO)**, and browse for the development database (**<Documents>\Altova\MapForce2018\MapForceExamples\altova.mdb**).
5. Click **Add Configuration** and name it "ReleaseDatabase".
6. Click **Choose Database**, select **Microsoft Access (ADO)**, and this time browse for the production database (**<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\altova.mdb**).

Step 2: Use the Global Resource in the mapping

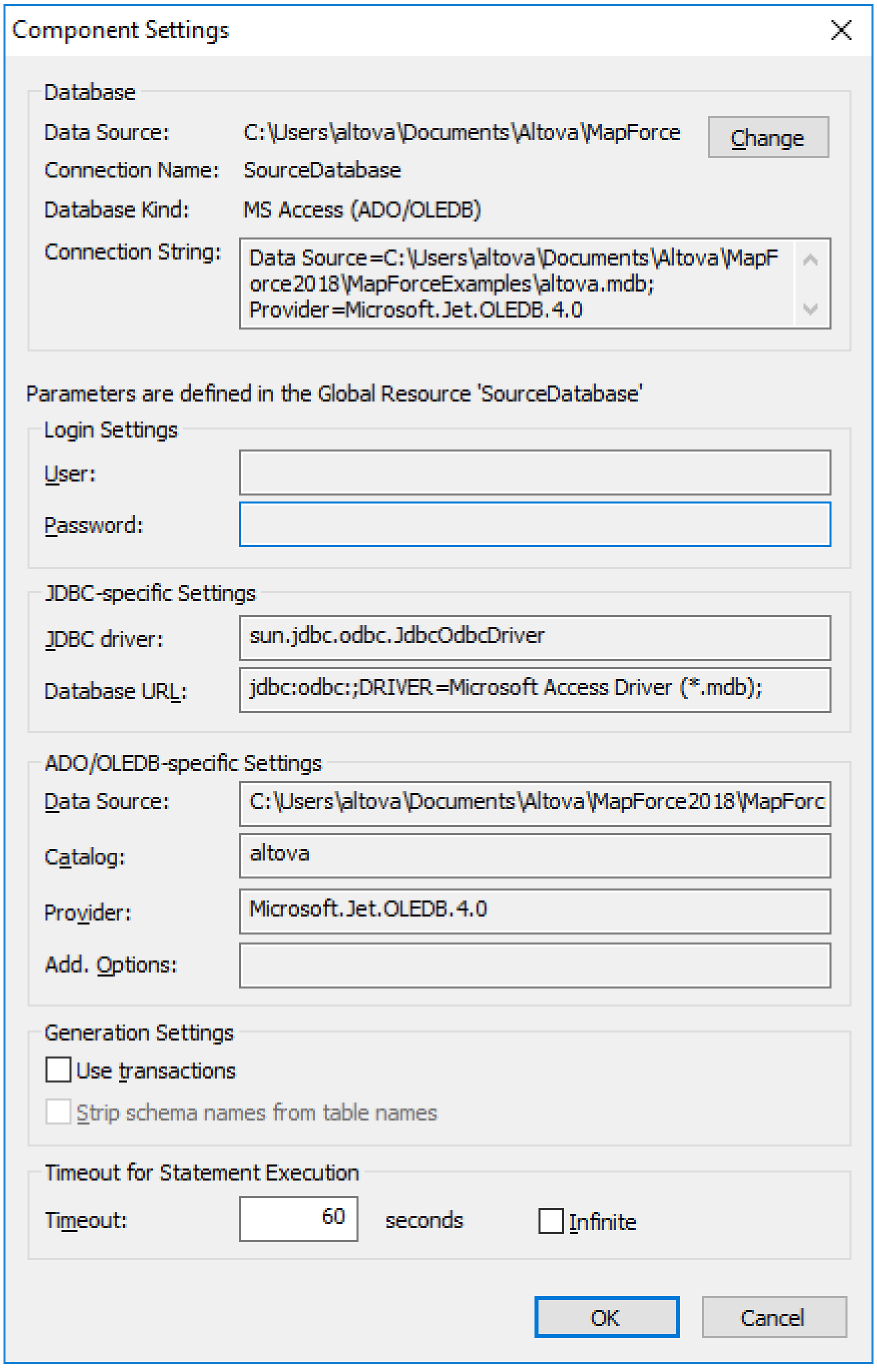
Now that the database alias has been created, the mapping must be modified to use it.

1. Open the following mapping: **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\PersonDB.mfd**.
2. Right-click the database component, and select **Properties** from the context menu.
3. Click **Change**, and select the "SourceDatabase" alias created previously.



4. Click **Connect**.
5. When prompted to select the database objects, leave the default selection as is, and click OK.

Note: When a database connection is defined as a Global Resource, the settings on the Component Settings dialog are grayed out, as illustrated below. As mentioned by the text on the dialog box, the connectivity parameters can be changed from the Global Resources dialog box (which can be opened by clicking the **Global Resource** toolbar button). See also Databases as Global Resources.



Component Settings

Database

Data Source: C:\Users\altova\Documents\Altova\MapForce

Connection Name: SourceDatabase

Database Kind: MS Access (ADO/OLEDB)

Connection String: Data Source=C:\Users\altova\Documents\Altova\MapForce2018\MapForceExamples\altova.mdb;
Provider=Microsoft.Jet.OLEDB.4.0

Parameters are defined in the Global Resource 'SourceDatabase'

Login Settings

User:

Password:

JDBC-specific Settings

JDBC driver: sun.jdbc.odbc.JdbcOdbcDriver

Database URL: jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);

ADO/OLEDB-specific Settings

Data Source: C:\Users\altova\Documents\Altova\MapForce2018\MapForc

Catalog: altova

Provider: Microsoft.Jet.OLEDB.4.0

Add. Options:

Generation Settings

□ Use transactions

□ Strip schema names from table names

Timeout for Statement Execution

Timeout: 60 seconds □ Infinite

Step 3: Run the mapping with the desired configuration

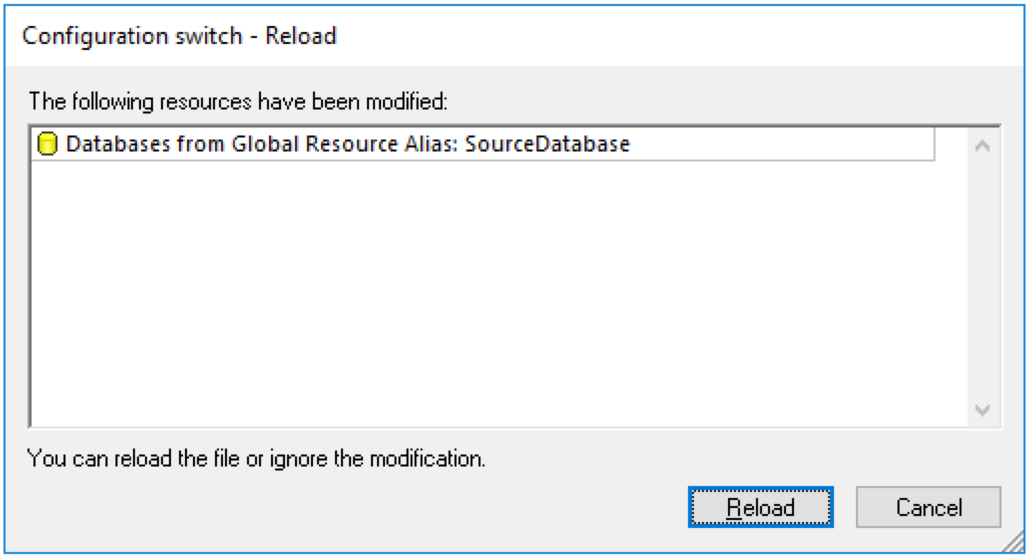
You can now easily switch to the desired database before running the mapping, as follows:

- On the **Tools** menu, click **Active Configuration | Default**, to read data from the development database.
- On the **Tools** menu, click **Active Configuration | ReleaseDatabase**, to read data from

the production database.

Alternatively, select the required configuration from the **Global Resources** drop-down list.

When you switch configurations, a dialog box prompts you that the source database is about to be reloaded:


Configuration switch - Reload

The following resources have been modified:

Databases from Global Resource Alias: SourceDatabase

You can reload the file or ignore the modification.

Note: Both databases used in this example contain similar data, so there are no differences in the generated output after you run the mapping.

10.2.9 Example: Create an Application Workflow

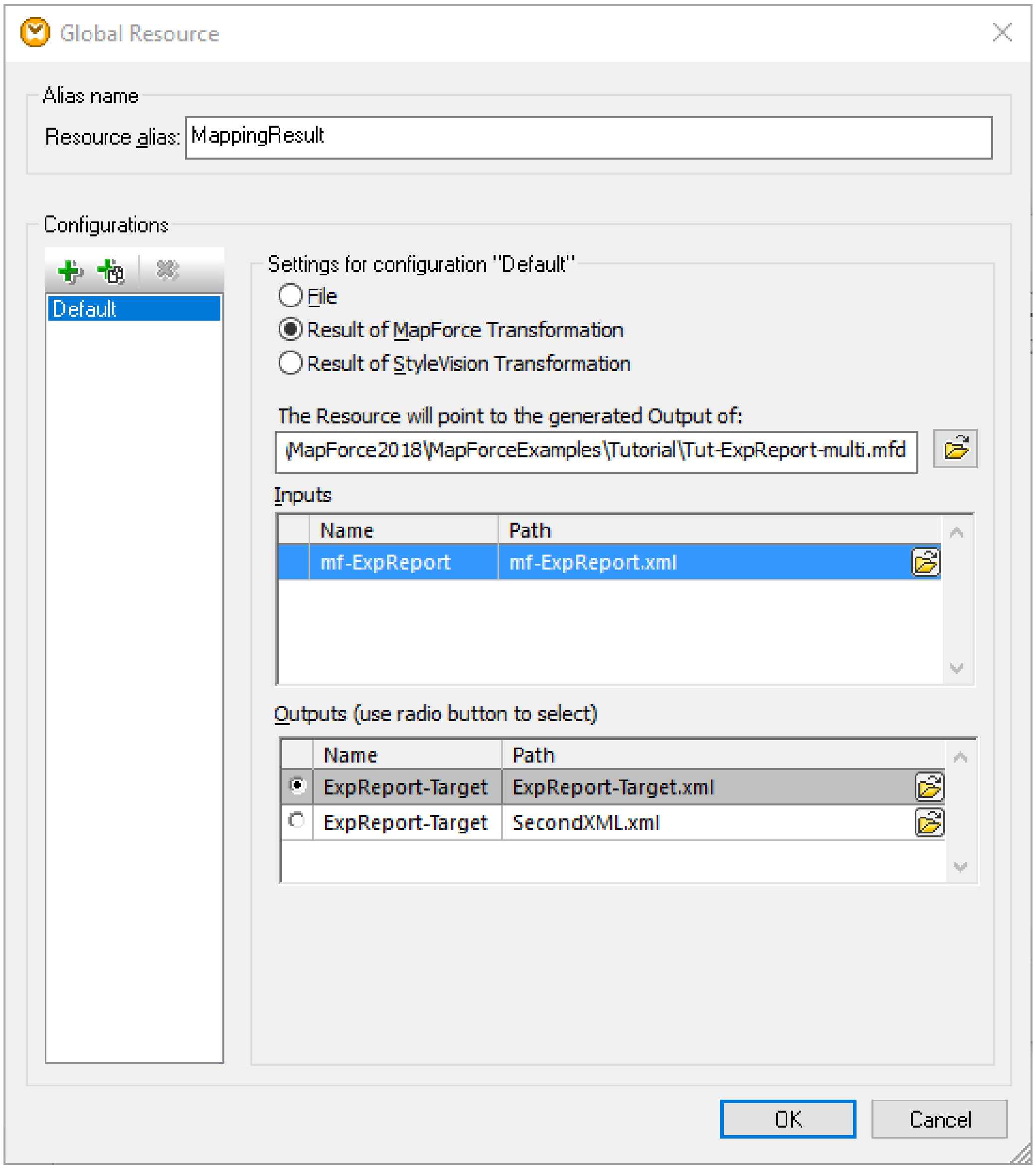
This example illustrates how to create a simple workflow between Altova MapForce and Altova XMLSpy, using Global Resources. Specifically, it shows you how to trigger the execution of a MapForce mapping directly from XMLSpy, and open in XMLSpy the mapping output generated by MapForce. To make this possible, we will create a Global Resource of type "Result of MapForce Transformation", as illustrated below.

Step 1: Create the Global Resource

This step can be performed from both MapForce and XMLSpy.

1. On the **Tools** menu, click **Global Resources**. (Alternatively, click the **Global Resource** toolbar button.)
2. Click **Add | File**.
3. Enter a descriptive name in the **Resource alias** text box (in this example, "MappingResult").
4. Select the option **Result of MapForce Transformation**.
5. Click **Browse** and select the mapping **<Documents>\Altova\MapForce2018\MapForceExamples\Tutorial\Tut-ExpReport-multi.mfd**. Be patient while the "Inputs" and "Outputs" sections on the dialog box are populated. As shown below, this mapping

has one input and two outputs.


Global Resource
Alias name
Resource alias: MappingResult
Configurations
Default
Settings for configuration "Default"
○ File
● Result of MapForce Transformation
○ Result of StyleVision Transformation
The Resource will point to the generated Output of:
\MapForce2018\MapForceExamples\Tutorial\Tut-ExpReport-multi.mfd
Inputs
Outputs (use radio button to select)
OK
Cancel

Inputs:

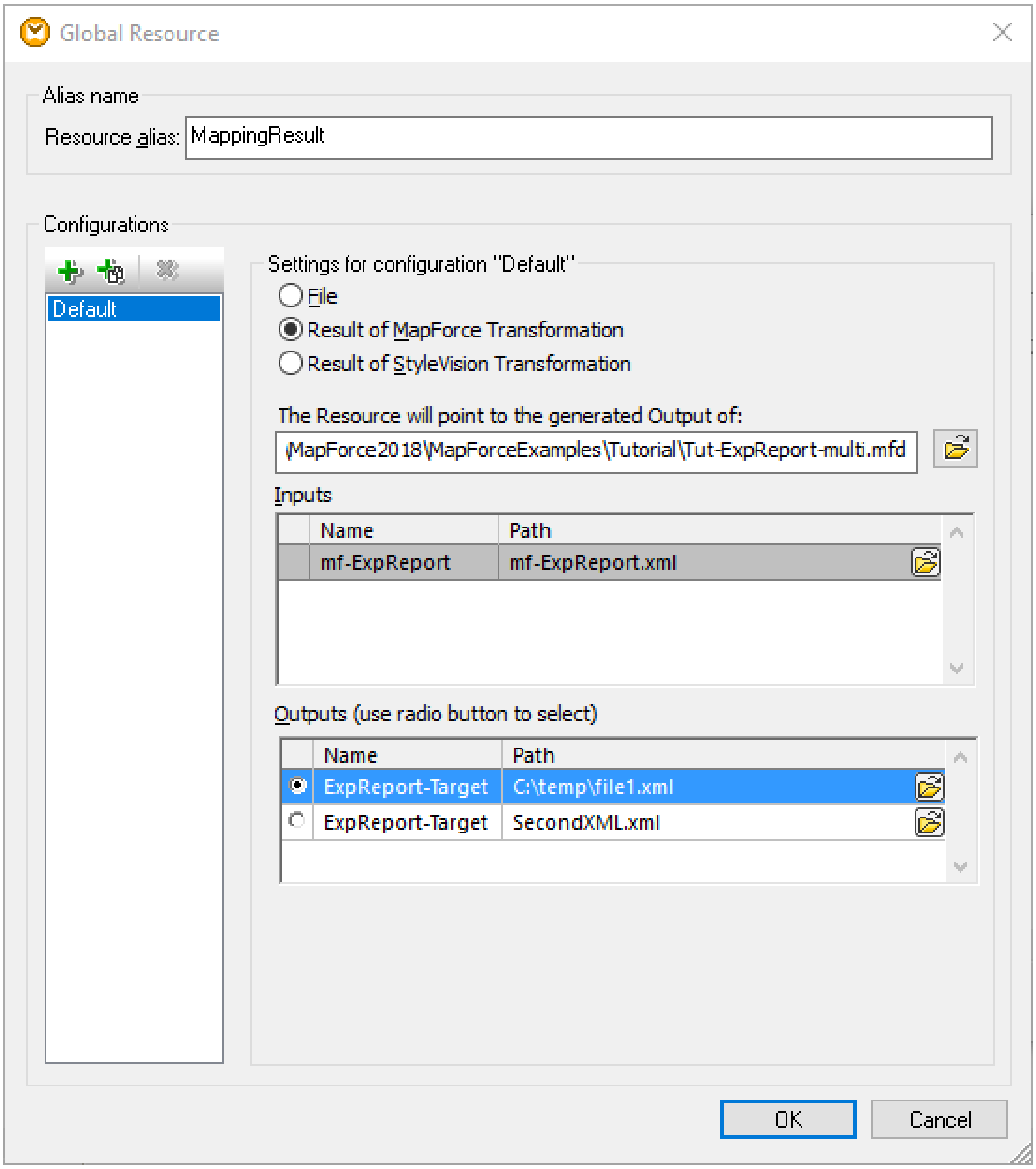
Name	Path
mf-ExpReport	mf-ExpReport.xml

Outputs (use radio button to select):

	Name	Path
●	ExpReport-Target	ExpReport-Target.xml
○	ExpReport-Target	SecondXML.xml

For the scope of this example, we would like to generate each of the two output files to the folder **C:\temp**, and change the default file name. To achieve this, we will create a configuration for each desired output, as follows:

1. Under "Outputs", click **Browse** next to the first output and enter **C:\temp\file1.xml** as destination file name. This is the default configuration which will produce the first output file when triggered.



Resource alias: MappingResult

Settings for configuration "Default"

○ File
● Result of MapForce Transformation
○ Result of StyleVision Transformation

The Resource will point to the generated Output of:
\MapForce2018\MapForceExamples\Tutorial\Tut-ExpReport-multi.mfd

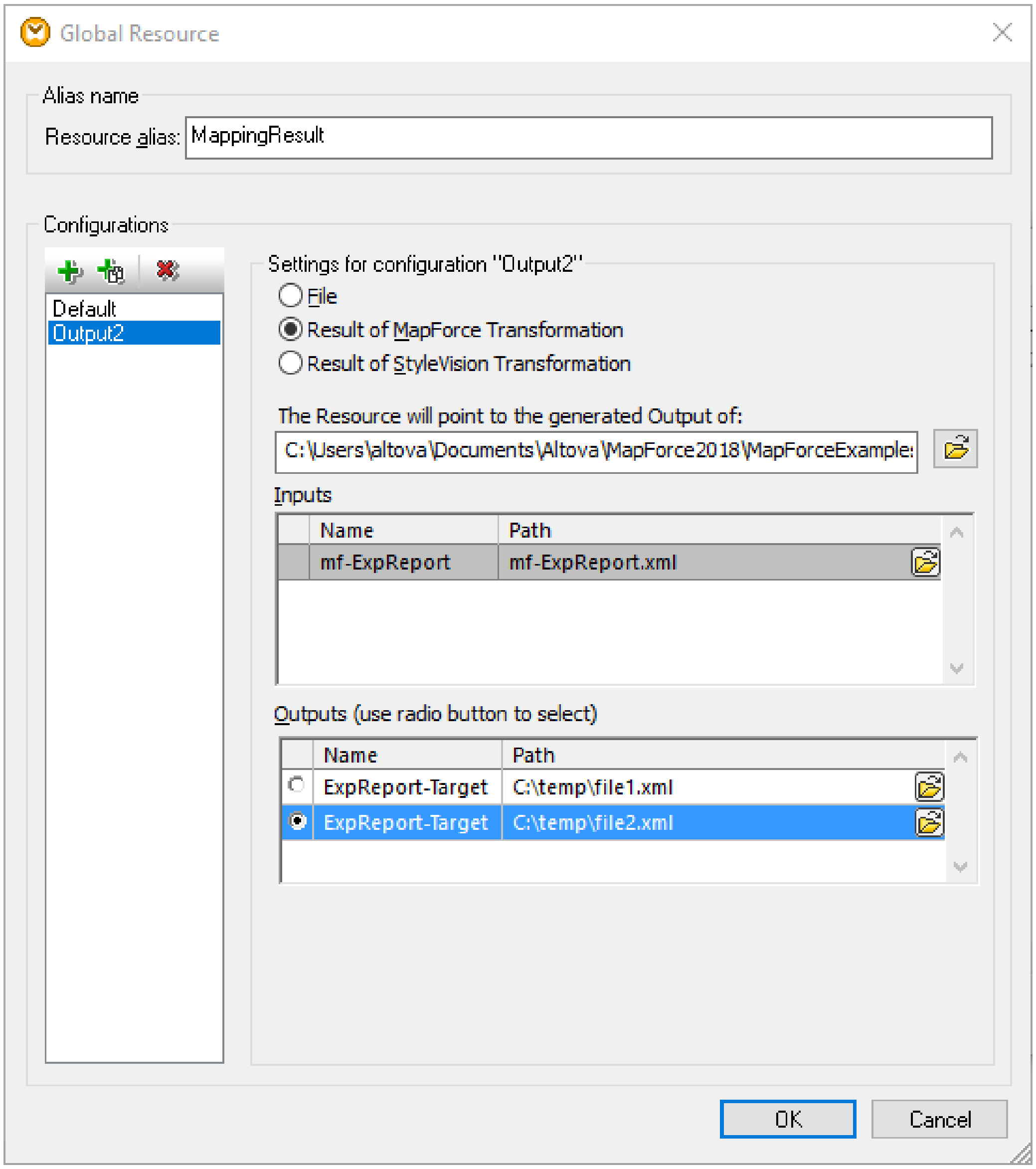
Inputs

Name	Path
mf-ExpReport	mf-ExpReport.xml

Outputs (use radio button to select)

	Name	Path
●	ExpReport-Target	C:\temp\file1.xml
○	ExpReport-Target	SecondXML.xml

2. Click **Add Configuration as a copy...** and enter a name for the new configuration (in this example, "Output2").
3. Under "Outputs", click **Browse** next to the second output and enter **C:\temp\file2.xml** as destination file name. This is the alternative configuration which produces the second output file.



Alias name

Resource alias: MappingResult

Configurations

Default
Output2

Settings for configuration "Output2"

○ File
● Result of MapForce Transformation
○ Result of StyleVision Transformation

The Resource will point to the generated Output of:
C:\Users\altova\Documents\Altova\MapForce2018\MapForceExample:

Inputs

Name	Path
mf-ExpReport	mf-ExpReport.xml

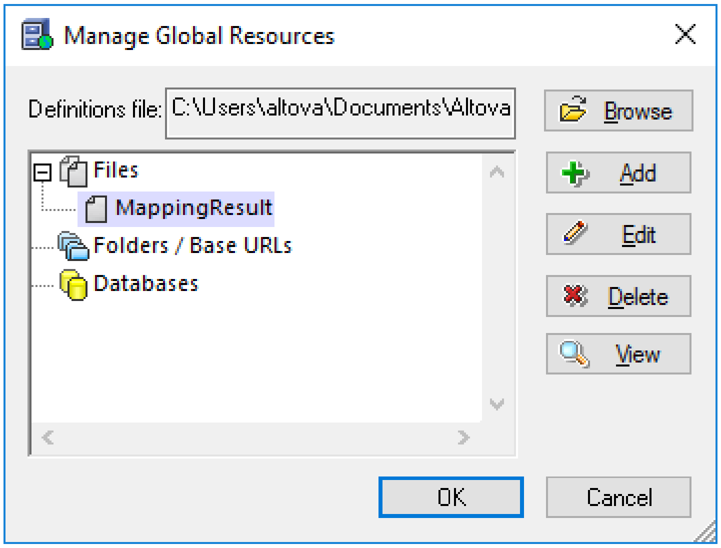
Outputs (use radio button to select)

	Name	Path
○	ExpReport-Target	C:\temp\file1.xml
●	ExpReport-Target	C:\temp\file2.xml

Step 2: Trigger the workflow

The Global Resource created in the previous step can now be consumed from XMLSpy, as follows.

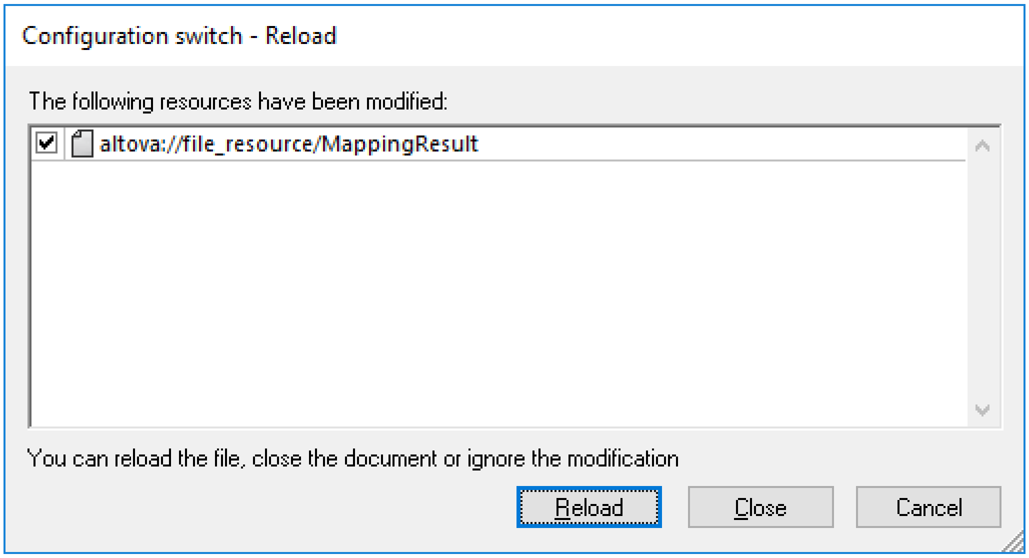
1. Run XMLSpy.
2. On the **Tools** menu, click **Global Resources**.
3. In the "Files" group, click the "MappingResult" Global Resource created previously, and then click **View**.



This executes the mapping, produces the default output (**file1.xml**) and loads it into the main pane of XMLSpy. The file is also saved as **C:\temp\file1.xml**.

To trigger the mapping execution with the alternative configuration, do the following:

1. On the **Tools** menu, click **Active Configuration | Output2**.


Configuration switch - Reload
The following resources have been modified:
☑ altova://file_resource/MappingResult
You can reload the file, close the document or ignore the modification
Reload
Close
Cancel

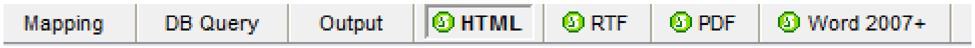
2. Click **Reload** when prompted.

As a result, the alternative output file is loaded into the main pane of XMLSpy. The file is also saved as: **C:\temp\file2.xml**.

10.3 Styling Mapping Output with StyleVision

In mappings where the target component is XML , it is possible to preview and save the mapping output as HTML, RTF, PDF and Word 2007+ documents, provided that Altova StyleVision is installed on your computer. If you are using the Enterprise edition of StyleVision, then charts will also be rendered in these previews.

When a mapping supports preview in any of these formats, additional tabs become available next to the **Output** tab, for example:



StyleVision preview tabs (MapForce Enterprise Edition)

Note the following:

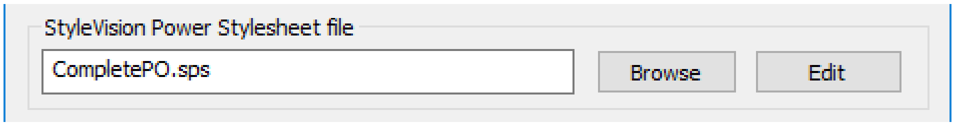
- When StyleVision Professional is installed, it is possible to preview **HTML** and **RTF** output. With StyleVision Enterprise, it is possible to preview **HTML**, **RTF**, **PDF**, and **Word 2007+** output.
- Previewing mapping output as PDF requires Java, Acrobat Reader, and FOP (Formatting Objects 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 a non-embedded application.
- If your mapping contains components that act both as source and target (pass-through components), the StyleVision preview will only be possible for those components where the **Preview** button 👁 of the component has been set as active. For more information about such mappings, see Chained Mappings.

In order to preview data from a mapping in this way, 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 to 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.

Assigning a StyleVision Power Stylesheet to a target component

1. In StyleVision, create the required stylesheet file. Make sure to use as source the same XML schema as that of the MapForce component.
2. In MapForce, right-click the target XML component, and select **Properties**.
3. On the Component Settings dialog box, next to **StyleVision Power Stylesheet file**, browse for the stylesheet file created previously.


StyleVision Power Stylesheet file

CompletePO.sps

Note: The path to the StyleVision Power Stylesheet file can be absolute or relative, see Using Relative and Absolute Paths.

Saving the StyleVision-generated output

You can save the StyleVision-generated HTML, PDF, RTF, or Word 2007+ output to a file in a similar way as saving the result of any other mapping. Namely, after previewing the mapping, do one of the following:

- Click the **Save generated output** (💾) toolbar button.
- On the **Output** menu, click **Save Output File**.

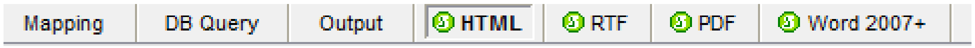
Automating generation of HTML, PDF, RTF, Word 2007+ files with Altova product suite

If your mapping should generate HTML, PDF, RTF, and Word 2007+ files automatically (either on the same or on a different computer or even platform), this is possible with MapForce Server and StyleVision Server (these are separately licensed server products that extend the functionality of MapForce and StyleVision, respectively). In this scenario, each application plays the following distinct role:

- MapForce - enables you to design the mapping (.mfd file) which defines the data transformation inputs and outputs (for example, database to XML)
- MapForceServer - runs the executable mapping (.mfx file) at the command-line or from an API (either on the same or a different operating system)
- StyleVision - enables you to design the stylesheet (.sps file) required to transform mapping output to HTML, PDF, RTF, Word 2007+
- StyleVision Server - runs the .sps stylesheet which transforms the mapping output to a target desired format. This happens at the command line or from an API (either 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, and thus be fully automated.

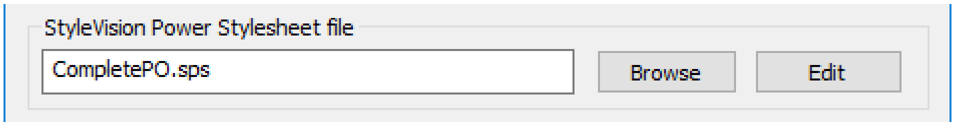
10.3.1 Examples of Mappings with StyleVision Stylesheets

Many of the mappings included in the MapForce examples folder (**<Documents>\Altova \MapForce2018\MapForceExamples\**) have StyleVision Power Stylesheets (.sps files) assigned to their target components. When that is the case, the mapping contains the additional StyleVision preview tabs.



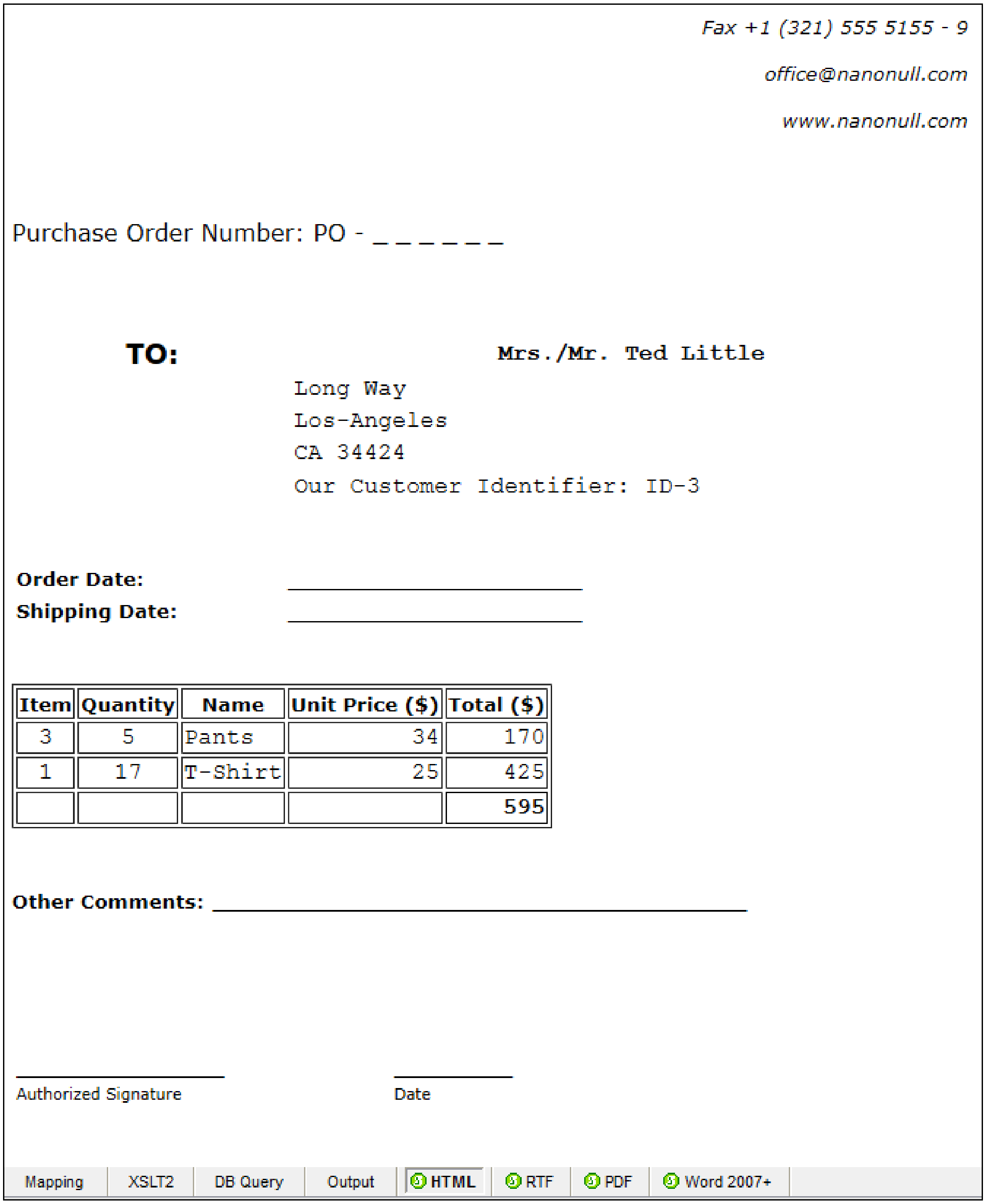
StyleVision preview tabs (MapForce Enterprise Edition)

One such example is **CompletePO.mfd** available at the following path: **<Documents>\Altova\MapForce2018\MapForceExamples\CompletePO.mfd**. This mapping produces a purchase order in XML format. Right-click the target component, select **Properties**, and notice that it has an .sps file assigned to it.


StyleVision Power Stylesheet file

CompletePO.sps

Click the **Output** tab to view the output data in HTML format.



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

HTML preview of CompletePO.mfd (MapForce Enterprise edition)

10.4 Generating and Customizing Mapping Documentation

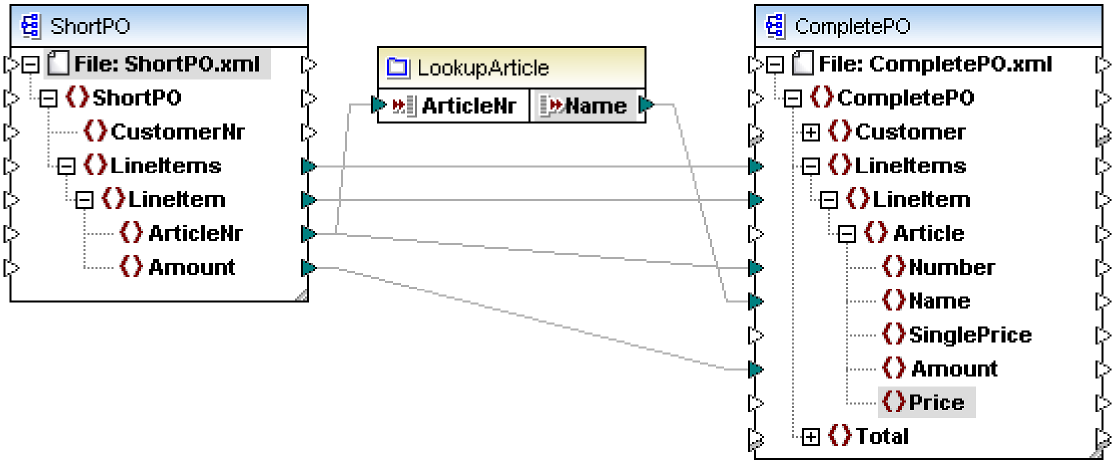
The **Generate Documentation** command generates detailed documentation about your mapping in HTML, MS Word, RTF or PDF. The documentation generated by this command can be freely altered and used; permission from Altova to do so is not required.

Documentation is generated for components you select in the Generate Documentation dialog box. You can either use the fixed design, or use a StyleVision Power Stylesheet (SPS) for the design. Using a StyleVision SPS enables you to customize the design of the generated documentation (see User-Defined Design).

Note: To use an SPS to generate mapping documentation, you must have StyleVision installed on your machine. Related elements are typically hyperlinked in the onscreen output, enabling you to navigate from component to component.

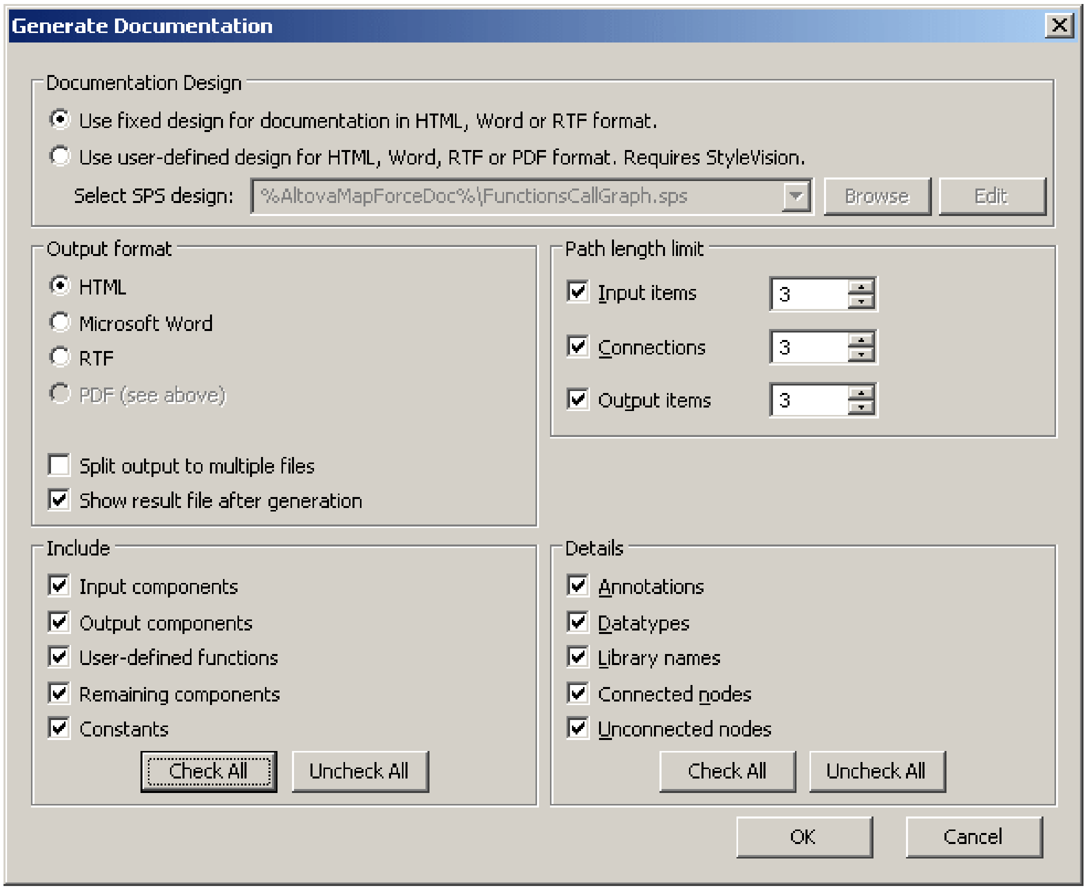
To generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

The screenshot below shows a portion of the **Lookup-standard.mfd** file available in the **...\MapForceExamples** folder.



Having opened a mapping file e.g. Lookup-standard.mfd:

- Select the menu option **File | Generate Documentation**.
  This opens the "Generate documentation" dialog box. The screenshot below shows the default dialog box settings.



Generate Documentation

Documentation Design
● Use fixed design for documentation in HTML, Word or RTF format.
○ Use user-defined design for HTML, Word, RTF or PDF format. Requires StyleVision.
Select SPS design: %AltovaMapForceDoc%\FunctionsCallGraph.sps

Output format
● HTML
○ Microsoft Word
○ RTF
○ PDF (see above)

□ Split output to multiple files
☑ Show result file after generation

Path length limit
☑ Input items 3
☑ Connections 3
☑ Output items 3

Include
☑ Input components
☑ Output components
☑ User-defined functions
☑ Remaining components
☑ Constants

Details
☑ Annotations
☑ Datatypes
☑ Library names
☑ Connected nodes
☑ Unconnected nodes

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\MapForce2018 \Documentation\MapForce\** folder.
- Click **Browse** to browse for a predefined SPS file.
- Click **Edit** to launch StyleVision and open the selected SPS in a StyleVision window.

The following predefined SPS stylesheets are available in the ...MapForce2018 \Documentation\MapForce folder:

- **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. The output using this option outputs the maximum detail and is identical to the built-in "fixed design..." output.

Output Format

- The output format is specified here: either 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 is only available if you use a StyleVision SPS to generate the documentation.

- Select "**Split output to multiple files**" if you would like separate input, output, constant components, user-defined functions from the Library component documentation. In fixed designs, links between multiple documents are created automatically.

- The "**Show Result File...**" option is enabled for all output options. When checked, the result files are displayed in default browser (HTML output), MS Word (MS Word output), and the default application for .rtf files (RTF output).

Path length limit

Allows you to define the maximum "path" length to be shown for items.

- E.g. .../ShortPO/LineItems/LineItem, which would be the maximum length for the default setting 3.

Include

- Allows you to define the specific components to appear in the documentation.

Details

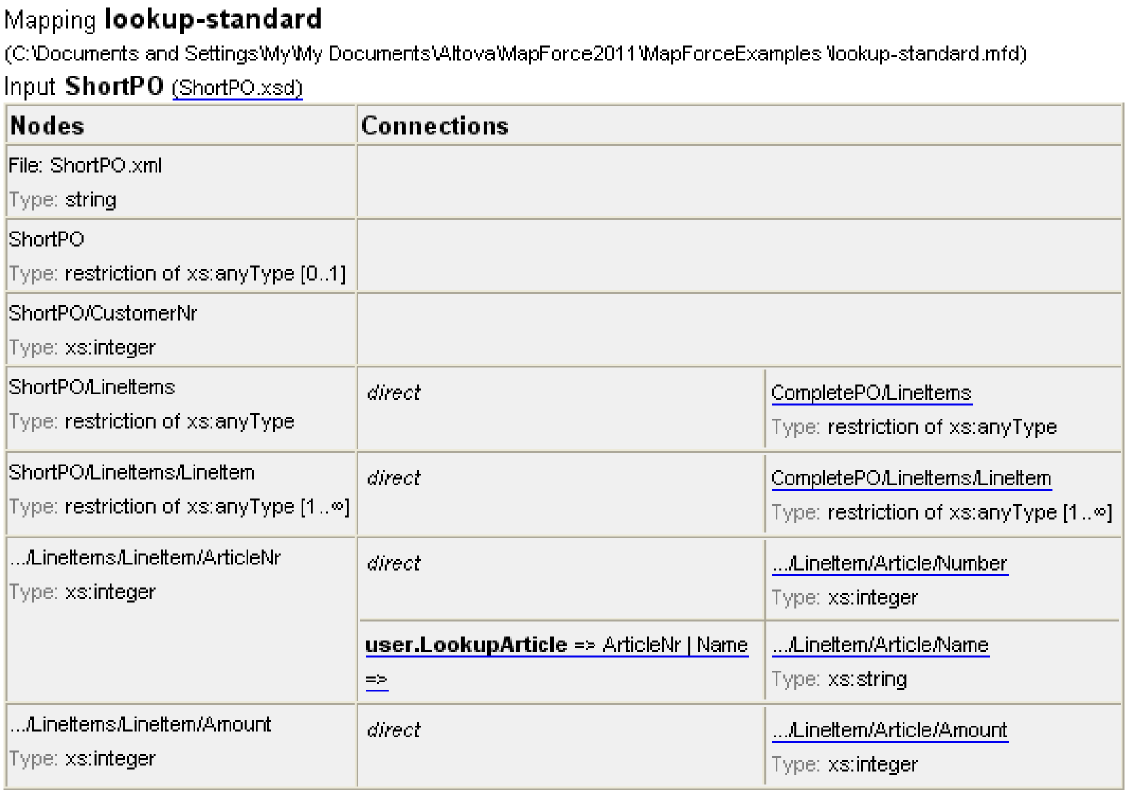
Allows you to set the specific details to appear in the documentation.

- selecting "Library Names" would insert the "core" prefix for functions.
- You can document both connected, as well as unconnected nodes.

Note:
The **Check**/**Uncheck All** buttons allow you to check/uncheck all check boxes of that group.

Having used the default settings shown above, clicking **OK,** prompts you for the name of the output file and the location to which it should be saved. A portion of the fixed design generated documentation is shown below. Note that this shows a single output file.

This table shows the connections **from** the source component to the target component(s).



Mapping **lookup-standard**

(C:\Documents and Settings\My\My Documents\Altova\MapForce2011\MapForceExamples\lookup-standard.mfd)

Input **ShortPO** (ShortPO.xsd)

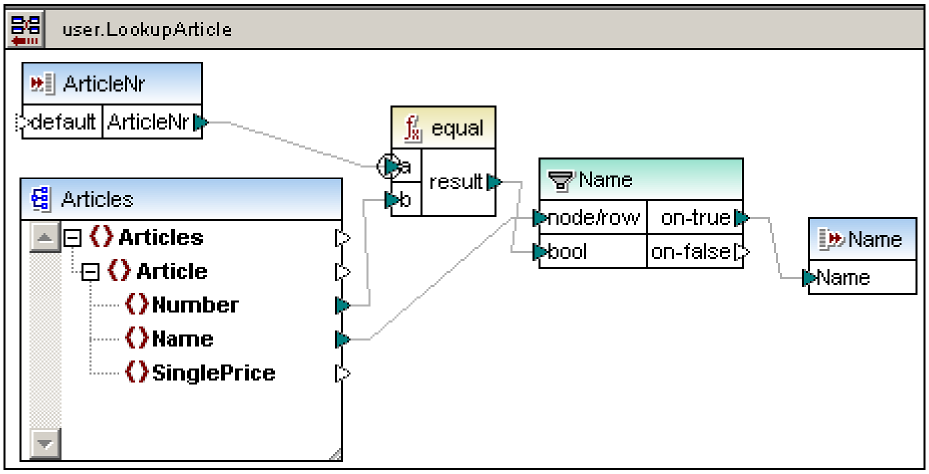
Nodes	Connections	
File: ShortPO.xml Type: string		
ShortPO Type: restriction of xs:anyType [0..1]		
ShortPO/CustomerNr Type: xs:integer		
ShortPO/LineItems Type: restriction of xs:anyType	*direct*	CompletePO/LineItems Type: restriction of xs:anyType
ShortPO/LineItems/LineItem Type: restriction of xs:anyType [1..∞]	*direct*	CompletePO/LineItems/LineItem Type: restriction of xs:anyType [1..∞]
.../LineItems/LineItem/ArticleNr Type: xs:integer	*direct*	.../LineItem/Article/Number Type: xs:integer
	user.LookupArticle => ArticleNr | Name =>	.../LineItem/Article/Name Type: xs:string
.../LineItems/LineItem/Amount Type: xs:integer	*direct*	.../LineItem/Article/Amount Type: xs:integer

The sequence in which the components are documented is: Input, Output, Constant, User-defined functions, then Library functions.

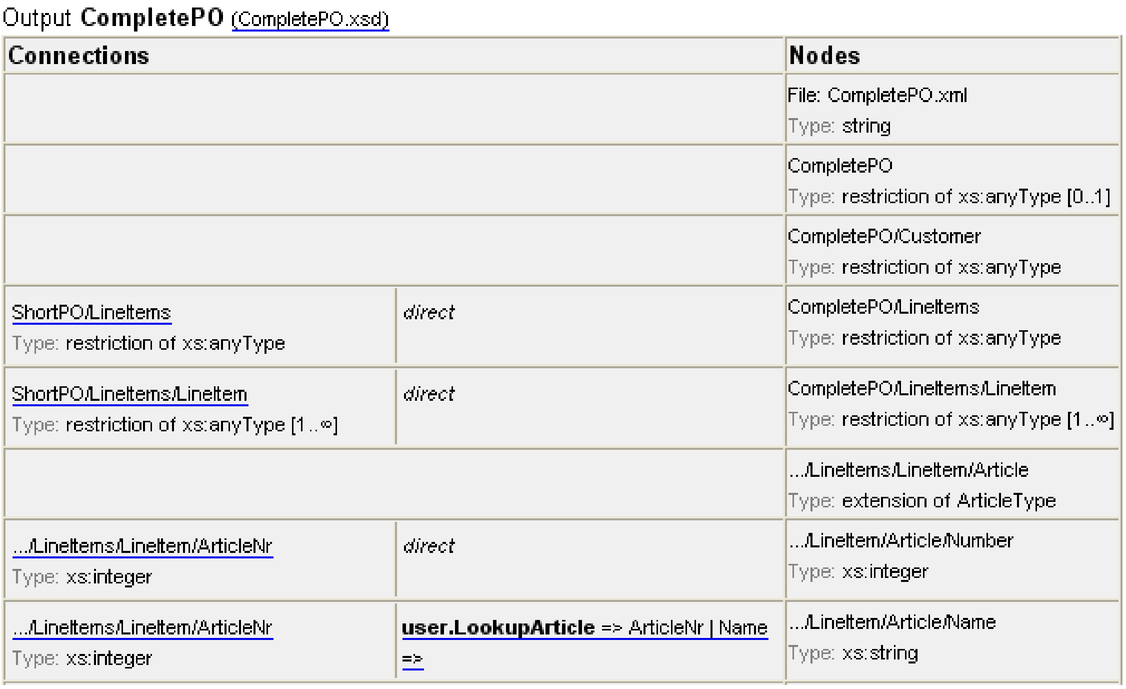
E.g. **Input component** ShortPO:

- The first two items ShortPO and ShortPO/CustomerNr are not connected to any item in the target, thus the Connections column is empty.
- **ShortPO/LineItems** is directly connected to CompletePO/**LineItems** in the target.
- /LineItems/LineItem/**ArticleNr** has two connections:
  - directly to LineItem/Article/**Number** in the target
  - to the User-defined function **LookupArticle,** with ArticleNr as the input parameter, and Name as the output parameter of the user-defined function.

  The contents of the user-defined function are shown below.



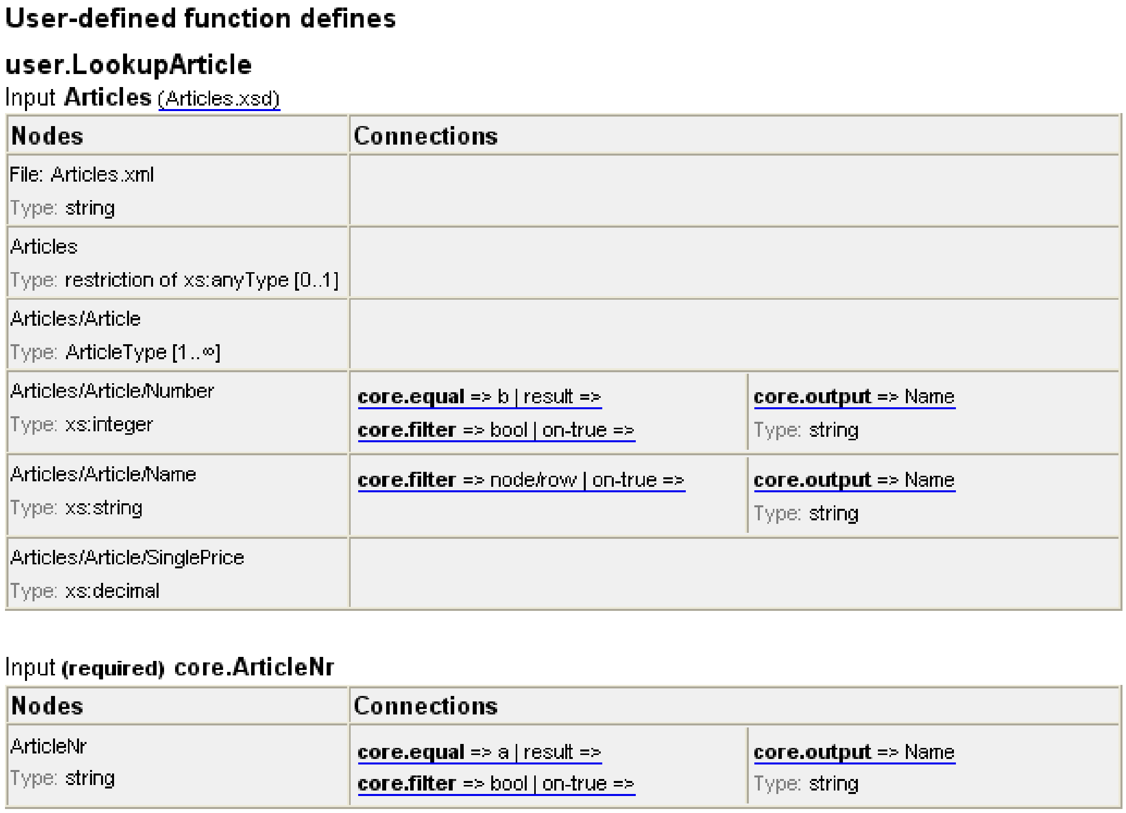
Output component CompletePO: This table shows the connections **to** the target component from the source component(s).



Output **CompletePO** (CompletePO.xsd)

Connections		Nodes
		File: CompletePO.xml Type: string
		CompletePO Type: restriction of xs:anyType [0..1]
		CompletePO/Customer Type: restriction of xs:anyType
ShortPO/LineItems Type: restriction of xs:anyType	*direct*	CompletePO/LineItems Type: restriction of xs:anyType
ShortPO/LineItems/LineItem Type: restriction of xs:anyType [1..∞]	*direct*	CompletePO/LineItems/LineItem Type: restriction of xs:anyType [1..∞]
		.../LineItems/LineItem/Article Type: extension of ArticleType
.../LineItems/LineItem/ArticleNr Type: xs:integer	*direct*	.../LineItem/Article/Number Type: xs:integer
.../LineItems/LineItem/ArticleNr Type: xs:integer	**user.LookupArticle** => ArticleNr | Name =>	.../LineItem/Article/Name Type: xs:string

- The first two items CompletePO and CompletePO/Customer are not connected to any item in the source component, thus the Connections column is empty.
- **CompletePO**/LineItems is directly connected to **ShortPO**/LineItems in the source component.
- LineItem/Article/Name is connected to the User-defined function **LookupArticle,** with LineItems/LineItem/ArticleNr as the source item.



User-defined function defines

user.LookupArticle

Input **Articles** (Articles.xsd)

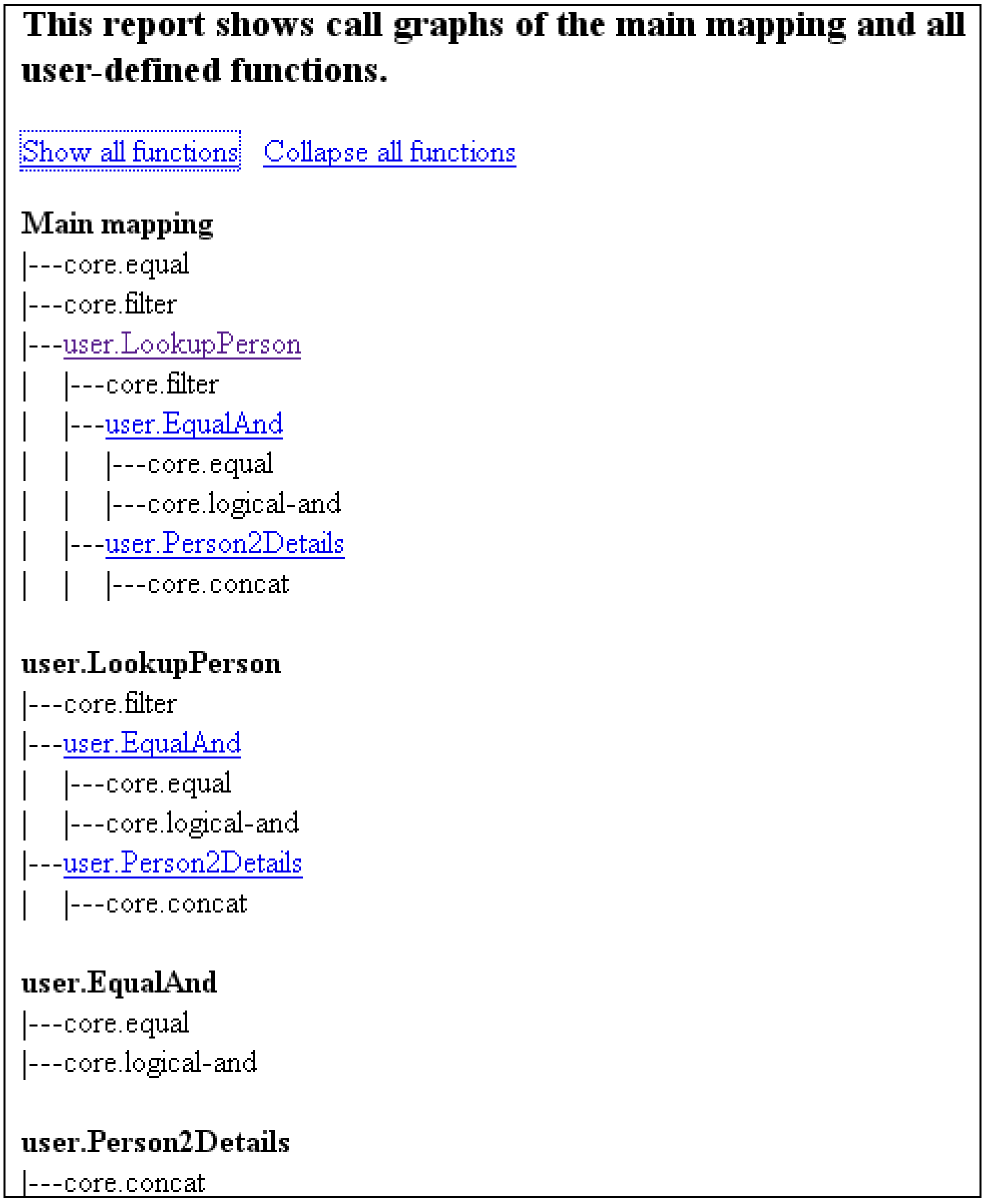
Nodes	Connections	
File: Articles.xml Type: string		
Articles Type: restriction of xs:anyType [0..1]		
Articles/Article Type: ArticleType [1..∞]		
Articles/Article/Number Type: xs:integer	**core.equal** => b | result => **core.filter** => bool | on-true =>	**core.output** => Name Type: string
Articles/Article/Name Type: xs:string	**core.filter** => node/row | on-true =>	**core.output** => Name Type: string
Articles/Article/SinglePrice Type: xs:decimal		

Input **(required) core.ArticleNr**

Nodes	Connections	
ArticleNr Type: string	**core.equal** => a | result => **core.filter** => bool | on-true =>	**core.output** => Name Type: string

10.4.1 Predefined StyleVision Power Stylesheets

Function Call Graphs - PersonListByBranchOffice.mfd



This report shows call graphs of the main mapping and all user-defined functions.

Show all functions Collapse all functions

Main mapping

```
|---core.equal
|---core.filter
|---user.LookupPerson
|    |---core.filter
|    |---user.EqualAnd
|    |    |---core.equal
|    |    |---core.logical-and
|    |---user.Person2Details
|    |    |---core.concat
```

user.LookupPerson

```
|---core.filter
|---user.EqualAnd
|    |---core.equal
|    |---core.logical-and
|---user.Person2Details
|    |---core.concat
```

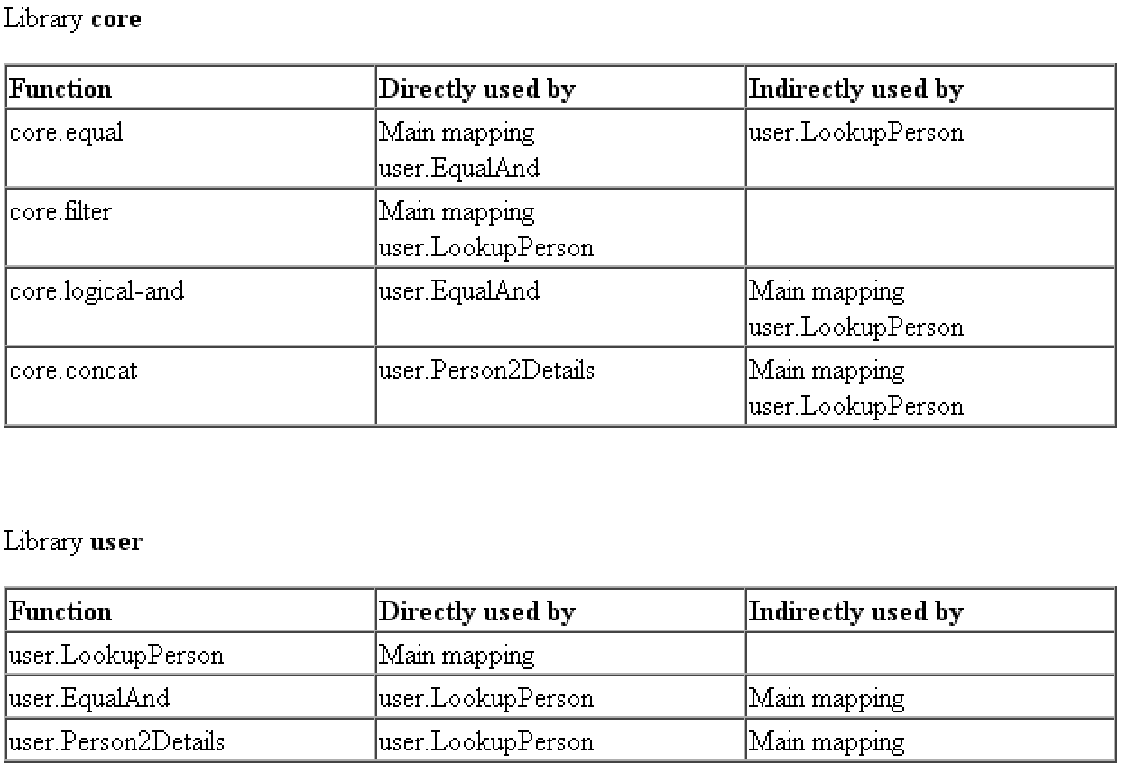
user.EqualAnd

```
|---core.equal
|---core.logical-and
```

user.Person2Details

```
|---core.concat
```

Functions Used By - PersonListByBranchOffice.mfd



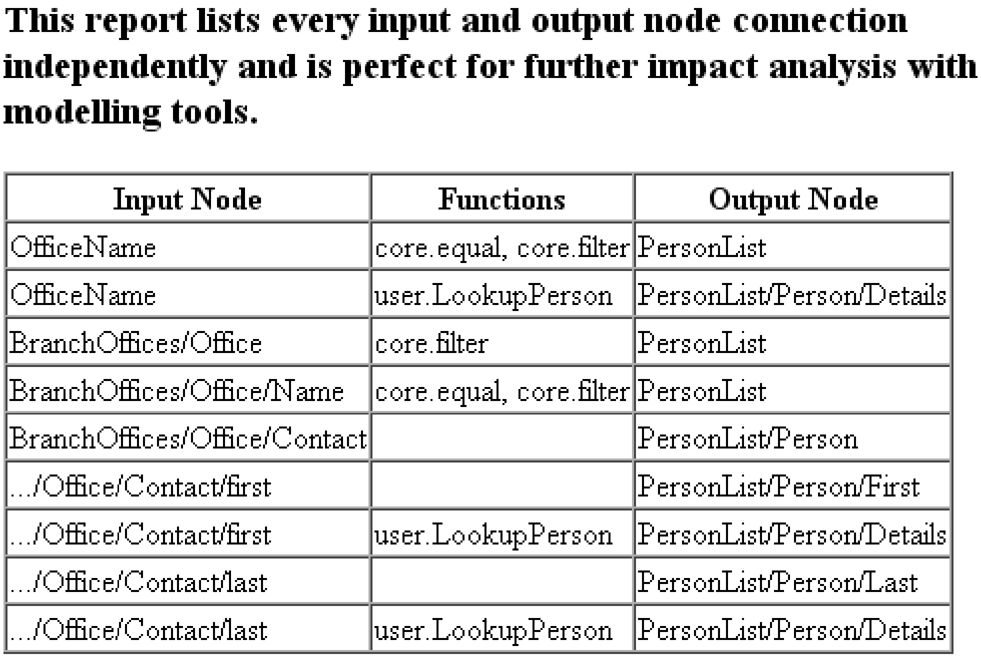
Library **core**

Function	Directly used by	Indirectly used by
core.equal	Main mapping user.EqualAnd	user.LookupPerson
core.filter	Main mapping user.LookupPerson	
core.logical-and	user.EqualAnd	Main mapping user.LookupPerson
core.concat	user.Person2Details	Main mapping user.LookupPerson

Library **user**

Function	Directly used by	Indirectly used by
user.LookupPerson	Main mapping	
user.EqualAnd	user.LookupPerson	Main mapping
user.Person2Details	user.LookupPerson	Main mapping

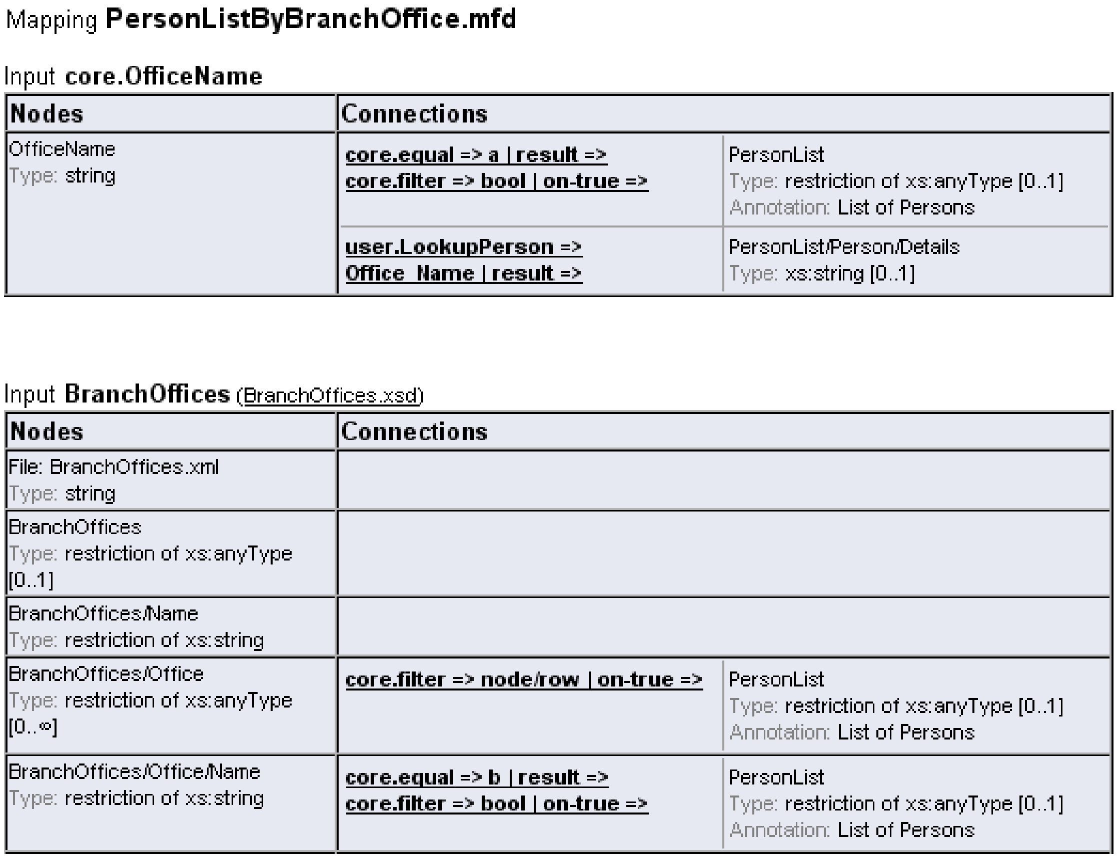
Impact Analysis - PersonListByBranchOffice.mfd



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

Overall Documentation - PersonListByBranchOffice.mfd



Mapping **PersonListByBranchOffice.mfd**

Input **core.OfficeName**

Nodes	Connections	
OfficeName Type: string	**core.equal => a | result =>** **core.filter => bool | on-true =>**	PersonList Type: restriction of xs:anyType [0..1] Annotation: List of Persons
	user.LookupPerson => **Office_Name | result =>**	PersonList/Person/Details Type: xs:string [0..1]

Input **BranchOffices** (BranchOffices.xsd)

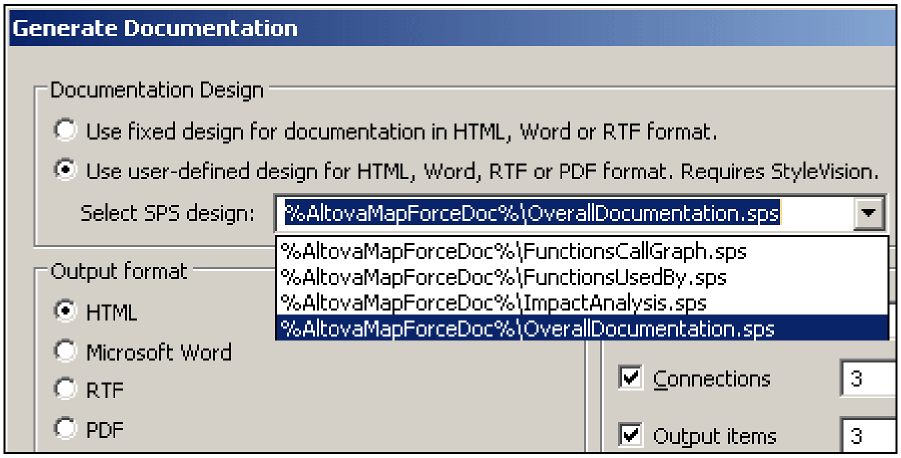
Nodes	Connections	
File: BranchOffices.xml Type: string		
BranchOffices Type: restriction of xs:anyType [0..1]		
BranchOffices/Name Type: restriction of xs:string		
BranchOffices/Office Type: restriction of xs:anyType [0..∞]	**core.filter => node/row | on-true =>**	PersonList Type: restriction of xs:anyType [0..1] Annotation: List of Persons
BranchOffices/Office/Name Type: restriction of xs:string	**core.equal => b | result =>** **core.filter => bool | on-true =>**	PersonList Type: restriction of xs:anyType [0..1] Annotation: List of Persons

10.4.2 Custom Design

Instead of the fixed design, you can create a customized design for the MapForce documentation. The customized design is created in a StyleVision SPS. Note that there are 4 predefined SPS Stylesheets supplied with MapForce, please see Documenting mapping projects.

Specifying the SPS to use for MapForce documentation

The SPS you wish to use for generating the documentation is specified in the Generate Documentation dialog (accessed via **File | Generate Documentation**). Select the "Use User-Defined Design..." radio button then click the dropdown arrow of the combo box and select the file you want. The default selection is the **OverallDocumentation.sps** entry.


Generate Documentation
Documentation Design
Output format
%AltovaMapForceDoc%\FunctionsCallGraph.sps
%AltovaMapForceDoc%\FunctionsUsedBy.sps
%AltovaMapForceDoc%\ImpactAnalysis.sps
%AltovaMapForceDoc%\OverallDocumentation.sps

○ Use fixed design for documentation in HTML, Word or RTF format.

● Use user-defined design for HTML, Word, RTF or PDF format. Requires StyleVision.

Select SPS design: %AltovaMapForceDoc%\OverallDocumentation.sps

● HTML
○ Microsoft Word
○ RTF
○ PDF

☑ Connections 3

☑ Output items 3

These predefined SPS files are located in the ...MapForce2018\Documentation\MapForce folder.

> Please note:
> To use an SPS to generate documentation, you must have StyleVision installed on your machine.

Creating the SPS

A StyleVision Power Stylesheet (or SPS) is created using Altova StyleVision (https://www.altova.com/stylevision). An SPS for generating MapForce documentation must be based on the XML Schema that specifies the structure of the XML document that contains the MapForce documentation.

This schema is called **MapForceDocumentation.xsd** and is delivered with your MapForce installation package. It is stored in the **...\Documents\Altova\MapForce2018\Documentation** folder.

When creating the SPS design in StyleVision, nodes from the MapForceDocumentation.xsd schema are placed in the design and assigned styles and properties. Note that the MapForceDocumentation.xsd **includes** the Documentation.xsd file located in the folder above it.

Additional components, such as links and images, can also be added to the SPS design. How to create an SPS design in StyleVision is described in detail in the StyleVision user manual.

The advantage of using an SPS for generating mapping documentation is that you have complete control over the design of the documentation. Note also that PDF output of the documentation is available only if an SPS is used; PDF output is not available if the fixed design is used.

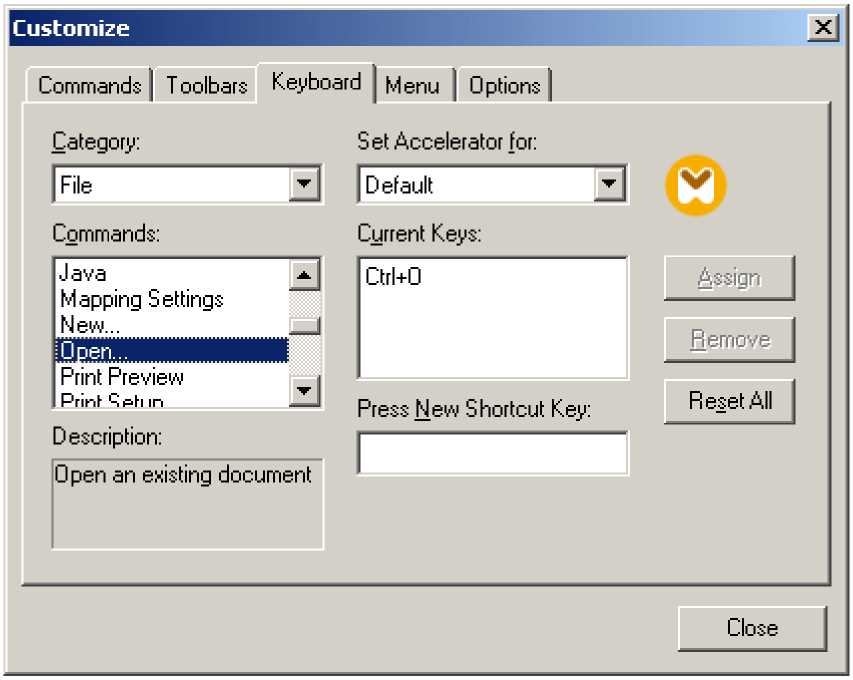
10.5 Customizing Keyboard Shortcuts

You can define or change the keyboard shortcuts in MapForce as follows:

1. On the **Tools** menu, click **Customize**.
2. Click the **Keyboard** tab.

To assign a new Shortcut to a command:

1. Select the **Tools | Customize** command and click the Keyboard tab.
2. Click the **Category** combo box to select the menu name.
3. Select the **command** you want to assign a new shortcut to, in the Commands list box
4. Click in the **Press New Shortcut Key**: text box, and press the shortcut keys that are to activate the command.



Category: File

Set Accelerator for: Default

Commands:
Java
Mapping Settings
New...
Open...
Print Preview
Print Setup

Current Keys: Ctrl+O

Press New Shortcut Key:

Description: Open an existing document

The shortcuts appear immediately in the text box. If the shortcut was assigned previously, then that function is displayed below the text box.

5. Click the **Assign** button to assign the shortcut.
   The shortcut now appears in the Current Keys list box.
   (To **clear** the entry in the Press New Shortcut Key text box, press any of the control keys, **CTRL**, **ALT** or **SHIFT**).

To de-assign or delete a shortcut:

1. Click the shortcut you want to delete in the Current Keys list box.
2. Click the **Remove** button.
3. Click the **Close** button to confirm.

Note: The **Set accelerator for** does not currently have any function.

The currently assigned keyboard shortcuts are as follows:

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

10.6 Catalog Files

MapForce supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables MapForce to retrieve commonly used schemas (as well as stylesheets and 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 below.

RootCatalog.xml

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 in a `nextCatalog` element. Each of these catalog files is looked up and the URIs in them are resolved according to the mappings specified in them.

```
<?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"/>
  <nextCatalog catalog="CoreCatalog.xml"/>
  <!-- Include all catalogs under common schemas folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/Schemas"
catalog="catalog.xml" spy:depth="1"/>
  <!-- Include all catalogs under common XBRL folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/XBRL"
catalog="catalog.xml" spy:depth="1"/>
</catalog>
```

In the listing above, notice that in the `Schemas` and `XBRL` folders of the folder identified by the variable `%AltovaCommonFolder%` are catalog files named `catalog.xml`. (The value of the `%AltovaCommonFolder%` variable is given in the table below.)

The catalog files in the Altova Common Folder map the pre-defined public and system identifiers of commonly used schemas (such as SVG and WSDL) and XBRL taxonomies to URIs that point to locally saved copies of the respective schemas. These schemas are installed in the Altova Common Folder when MapForce is installed.You should take care not to duplicate mappings in these files, as this could lead to errors.

CoreCatalog.xml, CustomCatalog.xml, and Catalog.xml

In the `RootCatalog.xml` listing above, notice that `CoreCatalog.xml` and `CustomCatalog.xml` are listed for lookup:

- `CoreCatalog.xml` contains certain Altova-specific mappings for locating schemas in the

Altova Common Folder.

- CustomCatalog.xml is a skeleton file in which you can create your own mappings. You can add mappings to CustomCatalog.xml for any schema you require but that is not addressed by the catalog files in the Altova Common Folder. Do this using the supported elements of the OASIS catalog mechanism (*see below*).
- There are a number of Catalog.xml files in the Altova Common Folder. Each is inside the folder of a specific schema or XBRL taxonomy in the Altova Common Folder, and each maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.

Location of catalog files and schemas

The files RootCatalog.xml and CoreCatalog.xml are installed in the MapForce application folder. The file 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 folders: %AltovaCommonFolder%\Schemas and %AltovaCommonFolder%\XBRL.

Shell environment variables and Altova 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:

%AltovaCommonFolder%	C:\Program Files\Altova\Common2018
%DesktopFolder%	Full path to the Desktop folder for the current user.
%ProgramMenuFolder%	Full path to the Program Menu folder for the current user.
%StartMenuFolder%	Full path to Start Menu folder for the current user.
%StartUpFolder%	Full path to Start Up folder for the current user.
%TemplateFolder%	Full path to the Template folder for the current user.
%AdminToolsFolder%	Full path to the file system directory that stores administrative tools for the current user.
%AppDataFolder%	Full path to the Application Data folder for the current user.
%CommonAppDataFolder%	Full path to the file directory containing application data for all users.
%FavoritesFolder%	Full path of the Favorites folder for the current user.
%PersonalFolder%	Full path to the Personal folder for the current user.
%SendToFolder%	Full path to the SendTo folder for the current user.

`%FontsFolder%`	Full path to the System Fonts folder.
`%ProgramFilesFolder%`	Full path to the Program Files folder for the current user.
`%CommonFilesFolder%`	Full path to the Common Files folder for the current user.
`%WindowsFolder%`	Full path to the Windows folder for the current user.
`%SystemFolder%`	Full path to the System folder for the current user.
`%CommonAppDataFolder%`	Full path to the file directory containing application data for all users.
`%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.

How catalogs work

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, the 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 point 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, 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>
```

This document is read and the catalog is searched for the `PUBLIC` identifier. 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, so the lookup for the SVG DTD is redirected to the URI `schemas/svg/svg11.dtd` (this path is relative to the catalog file), and this local file will be used as the DTD. If there is no mapping for the `Public` ID in the catalog, then the URL in the XML document will be used (in the example above: `http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`).

The catalog subset supported by MapForce

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"/>`

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. Also, a URI can be mapped to another URI using the `uri` element. The `rewriteURI` and `rewritsSystem` 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.

File extensions and intelligent editing according to a schema

Via catalog files you can also specify that documents with a particular file extension should have MapForce's intelligent editing features applied in conformance with the rules in a schema you specify. For example, if you create a custom file extension `.myhtml` for (HTML) files that are to be valid according to the HTML DTD, then you can enable intelligent editing for files with these extensions by adding the following element of text to `CustomCatalog.xml` as a child of the `<catalog>` element.

```
<spy:fileExtHelper ext="myhtml" uri="schemas/xhtml/xhtml1-transitional.dtd"/>
```

This would enable intelligent editing (auto-completion, entry helpers, etc) of `.myhtml` files in MapForce according to the XHTML 1.0 Transitional DTD. Refer to the `catalog.xml` file in the `%AltovaCommonFolder%\Schemas\xhtml` folder, which contains similar entries.

XML Schema and catalogs

XML Schema information is built into MapForce and the validity of XML Schema documents is checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema for XML Schema.

The `catalog.xml` file in the `%AltovaCommonFolder%\Schemas\schema` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against either of 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.

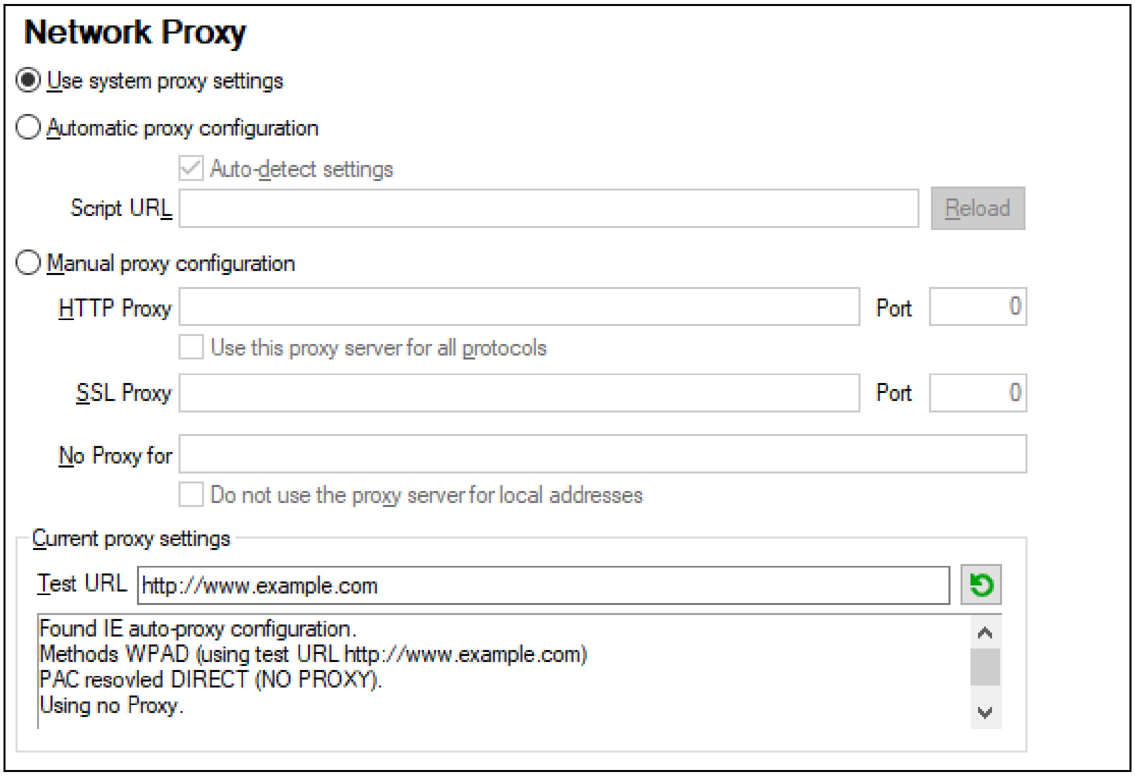
More information

For more information on catalogs, see the XML Catalogs specification.

10.7 Network Proxy Settings

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 using the options below.

Note: The network proxy settings are shared between all Altova MissionKit applications. Consequently, if you change the settings in one application, they will automatically affect all other applications.



Network Proxy

● Use system proxy settings

○ Automatic proxy configuration

☑ Auto-detect settings

Script URL:

○ Manual proxy configuration

HTTP Proxy: Port: 0

□ Use this proxy server for all protocols

SSL Proxy: Port: 0

No Proxy for:

□ Do not use the proxy server for local addresses

Current proxy settings

Test URL: http://www.example.com

```
Found IE auto-proxy configuration.
Methods WPAD (using test URL http://www.example.com)
PAC resovled DIRECT (NO PROXY).
Using no Proxy.
```

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:

- *Use this proxy for all protocols:* Uses the host name and port of the HTTP Proxy for all protocols.
- *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 (**`.`**).

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).

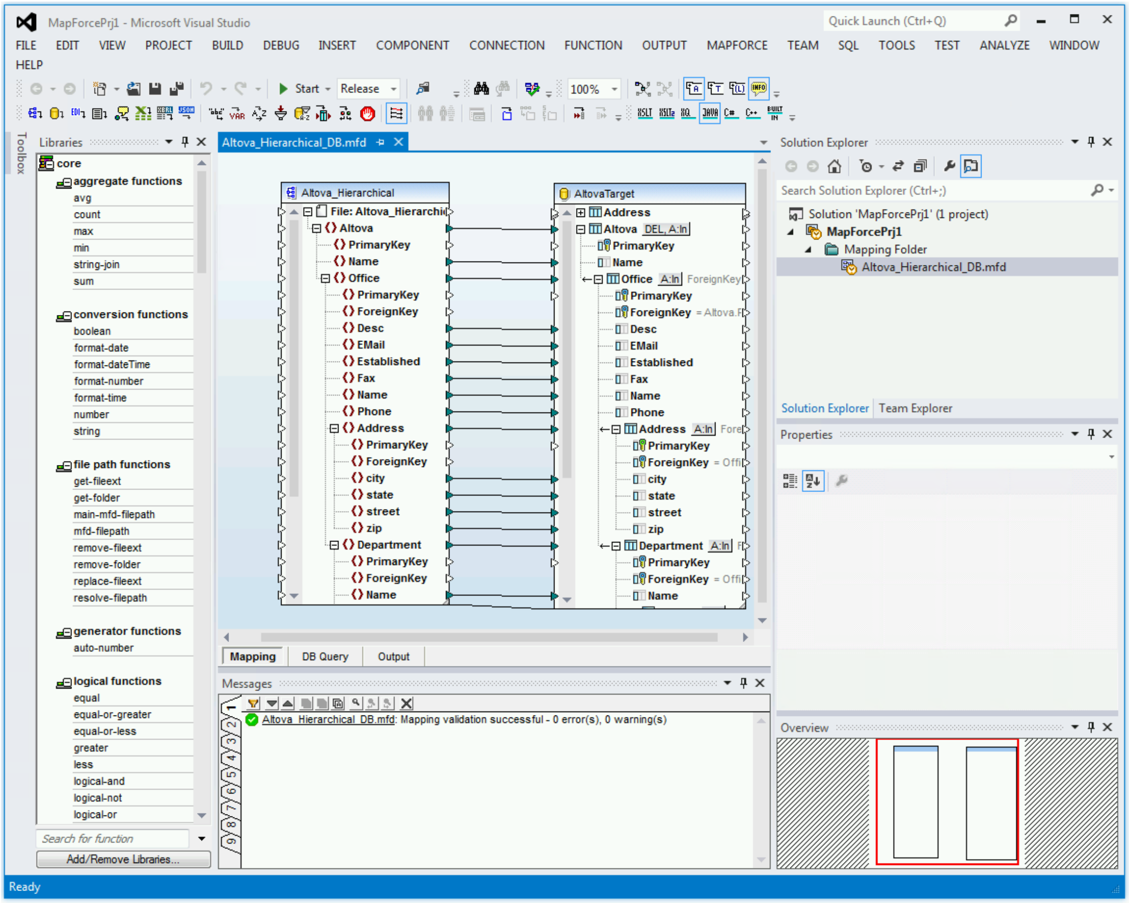
- *Test URL:* A test URL can be used to see which proxy is used for that specific URL. No I/O is done with this URL. This field must not be empty if proxy-auto-configuration is used (either through *Use system proxy settings* or *Authomatic proxy configuration*).

Chapter 11

MapForce Plug-in for Visual Studio

11 MapForce Plug-in for Visual Studio

You can integrate MapForce 2018 into the Microsoft Visual Studio versions 2008/2010/2012/2013/2015/2017. This unifies the best of both worlds, combining the mapping capabilities of MapForce with the development environment of Visual Studio. When the MapForce plug-in is enabled, you can create mapping projects and files directly from Visual Studio. You can also customize the MapForce options, including menus and toolbars, as you would do in the standalone version of MapForce.



Altova_Hierarchical_DB.mfd: Mapping validation successful - 0 error(s), 0 warning(s)

MapForce Enterprise Edition plug-in (Visual Studio 2012)

This section contains the following topics:

- Enabling the Plug-in
- Working with Mappings and Projects
- Accessing Common Menus and Functions

11.1 Enabling the Plug-in

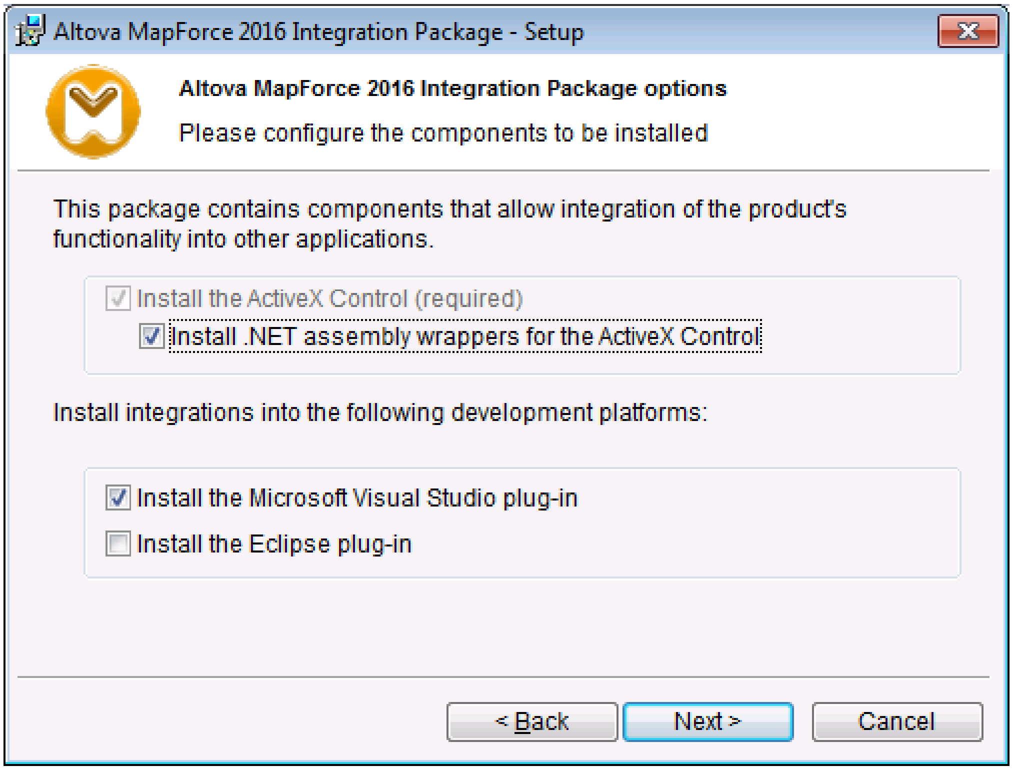
Prerequisites:

- Microsoft Visual Studio 2008/2010/2012/2013/2015/2017
- MapForce (Enterprise or Professional Edition)
- MapForce Integration Package, available for download at https://www.altova.com/mapforce/download.

Note: To use MapForce as a Visual Studio plug-in, install the 32-bit version of both MapForce and MapForce integration package, since there is currently no support for 64-bit plug-ins in Visual Studio.

To enable the MapForce plug-in for Visual Studio, download and run the MapForce Integration Package and follow the on-screen installation instructions.

During installation, ensure that the **Install the Microsoft Visual studio plug-in** option is selected:



Altova MapForce 2016 Integration Package - Setup

Altova MapForce 2016 Integration Package options

Please configure the components to be installed

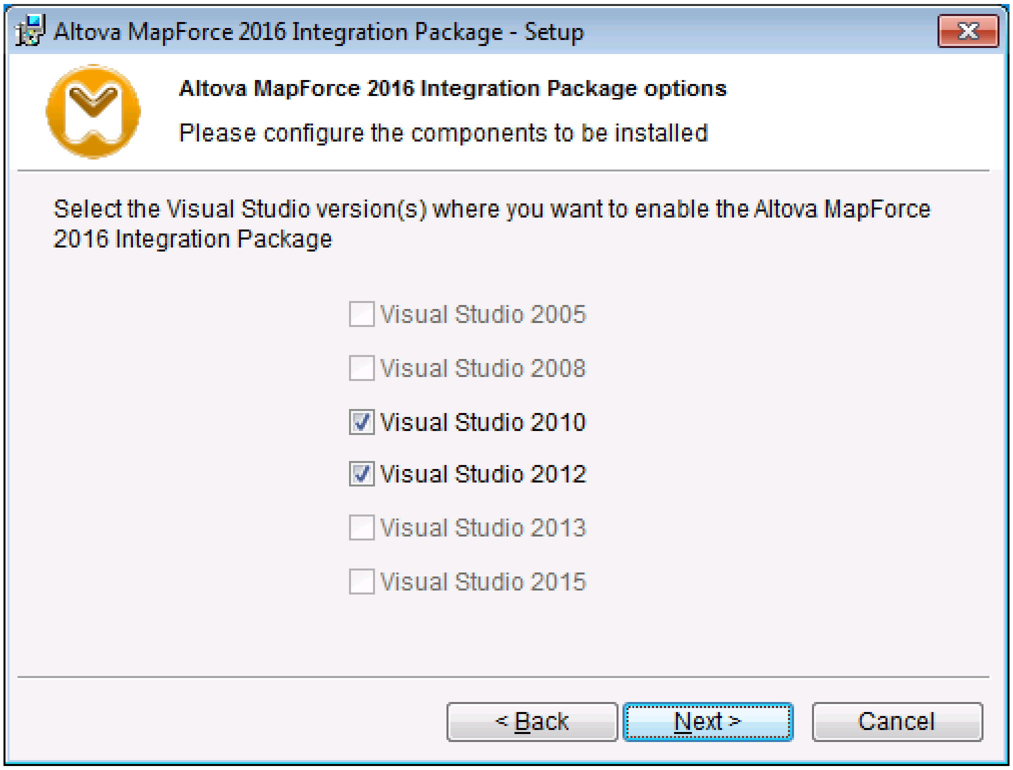
This package contains components that allow integration of the product's functionality into other applications.

☑ Install the ActiveX Control (required)
☑ Install .NET assembly wrappers for the ActiveX Control

Install integrations into the following development platforms:

☑ Install the Microsoft Visual Studio plug-in
☐ Install the Eclipse plug-in

When prompted, select the Visual Studio version(s) where the plug-in should be enabled, for example:


Altova MapForce 2016 Integration Package - Setup

Altova MapForce 2016 Integration Package options

Please configure the components to be installed

Select the Visual Studio version(s) where you want to enable the Altova MapForce 2016 Integration Package

☐ Visual Studio 2005
☐ Visual Studio 2008
☑ Visual Studio 2010
☑ Visual Studio 2012
☐ Visual Studio 2013
☐ Visual Studio 2015

Note: Only the Visual Studio versions installed on your operating system are available for selection.

Once the integration package has been installed, MapForce functionality becomes available in the Visual Studio environment.

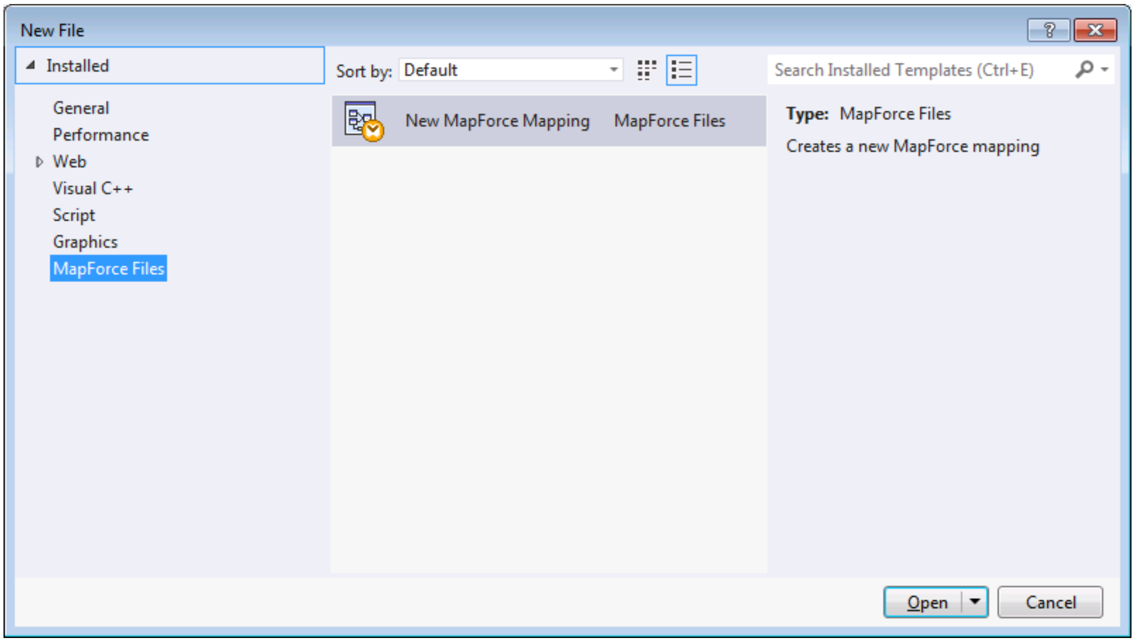
Enabling the MapForce plug-in manually

It is possible that the plug-in was not automatically enabled during the installation process. To enable it, do the following:

1. Navigate to the directory where the Visual Studio IDE executable is installed (for example, c:\Program Files\Microsoft Visual Studio 8\Common7\IDE).
2. Run the command prompt as administrator and enter `devenv.exe /setup`.
3. Wait for the process to terminate normally before starting to use the application within Visual Studio.

11.2 Working with Mappings and Projects

When MapForce plug-in for Visual Studio is enabled, you create and open mappings and mapping projects in a way that is applicable to the Visual Studio environment, as opposed to the standalone MapForce graphical user interface. For example, when you create a new file in Visual Studio (using the **File | New** menu command), or when you add a new item to a project (using the **Project | Add New Item** menu command), you can select **MapForce Files** as file type.



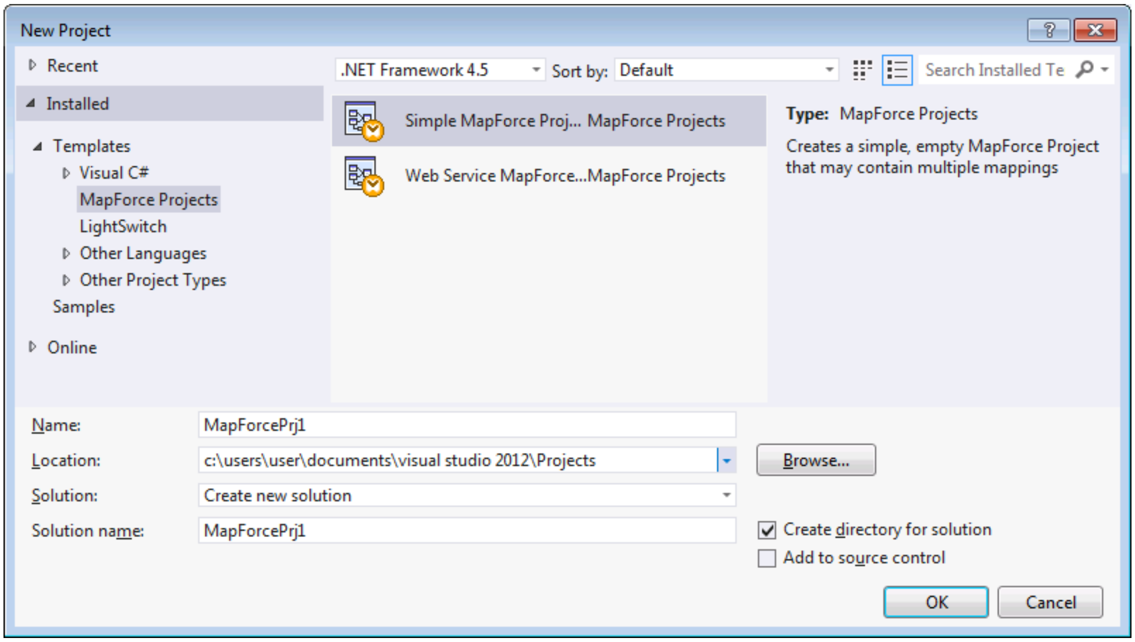
Sort by: Default

Type: MapForce Files

Creates a new MapForce mapping

New File dialog box (Visual Studio 2012 with MapForce Enterprise edition plug-in)

In a similar way, when you create a new Visual Studio project, you can select **MapForce Projects** as project template. The following screen shot illustrates a sample New Project dialog box in Visual Studio 2012 with the MapForce Enterprise Edition plug-in enabled.



Type: MapForce Projects

Creates a simple, empty MapForce Project that may contain multiple mappings

Name: MapForcePrj1

Location: c:\users\user\documents\visual studio 2012\Projects

Solution: Create new solution

Solution name: MapForcePrj1

☑ Create directory for solution

□ Add to source control

New Project dialog box (Visual Studio 2012 with MapForce Enterprise edition plug-in)

Opening existing MapForce files and projects is also done through the Visual Studio native functionality. When you need to open existing mapping files or projects, use the applicable Visual Studio menus (for example, **File | Open | Files**, or **File | Open | Project/Solutions**), and look for the MapForce-related file types.

11.3 Accessing Common Menus and Functions

When MapForce plug-in for Visual Studio is enabled, you can access common menus and functions as shown below. This is the default setup; however, you can change, if desired, the location of menus and toolbars from the **Tools | Options** menu of Visual Studio.

Global Resources

MapForce Global Resources are available in the **MapForce | Global Resources** menu of Visual Studio.

MapForce Options

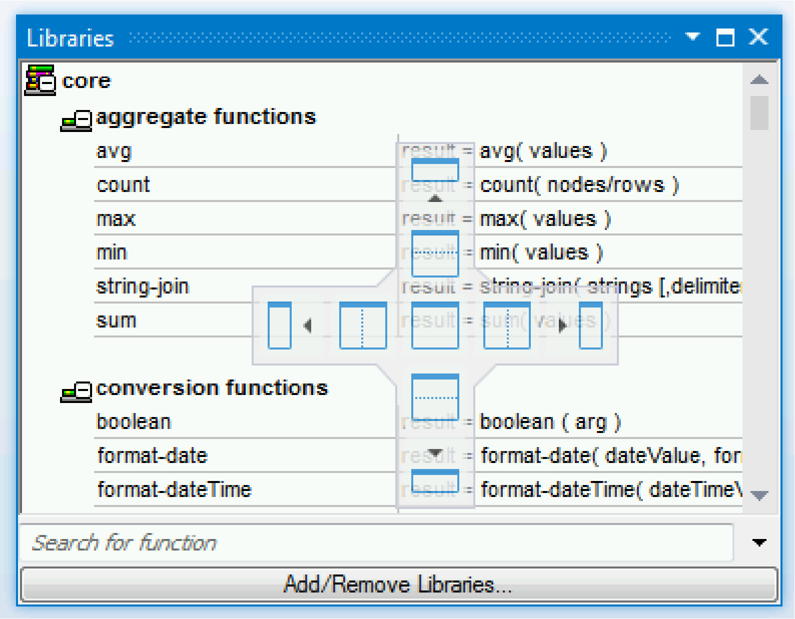
MapForce Options are available in the **Tools | MapForce Options** menu of Visual Studio.

Mapping Pane Customization

When there is a MapForce mapping opened in the main pane of Visual Studio, the **View | MapForce** menu becomes available. It includes the same options as the standalone version of MapForce.

Libraries window

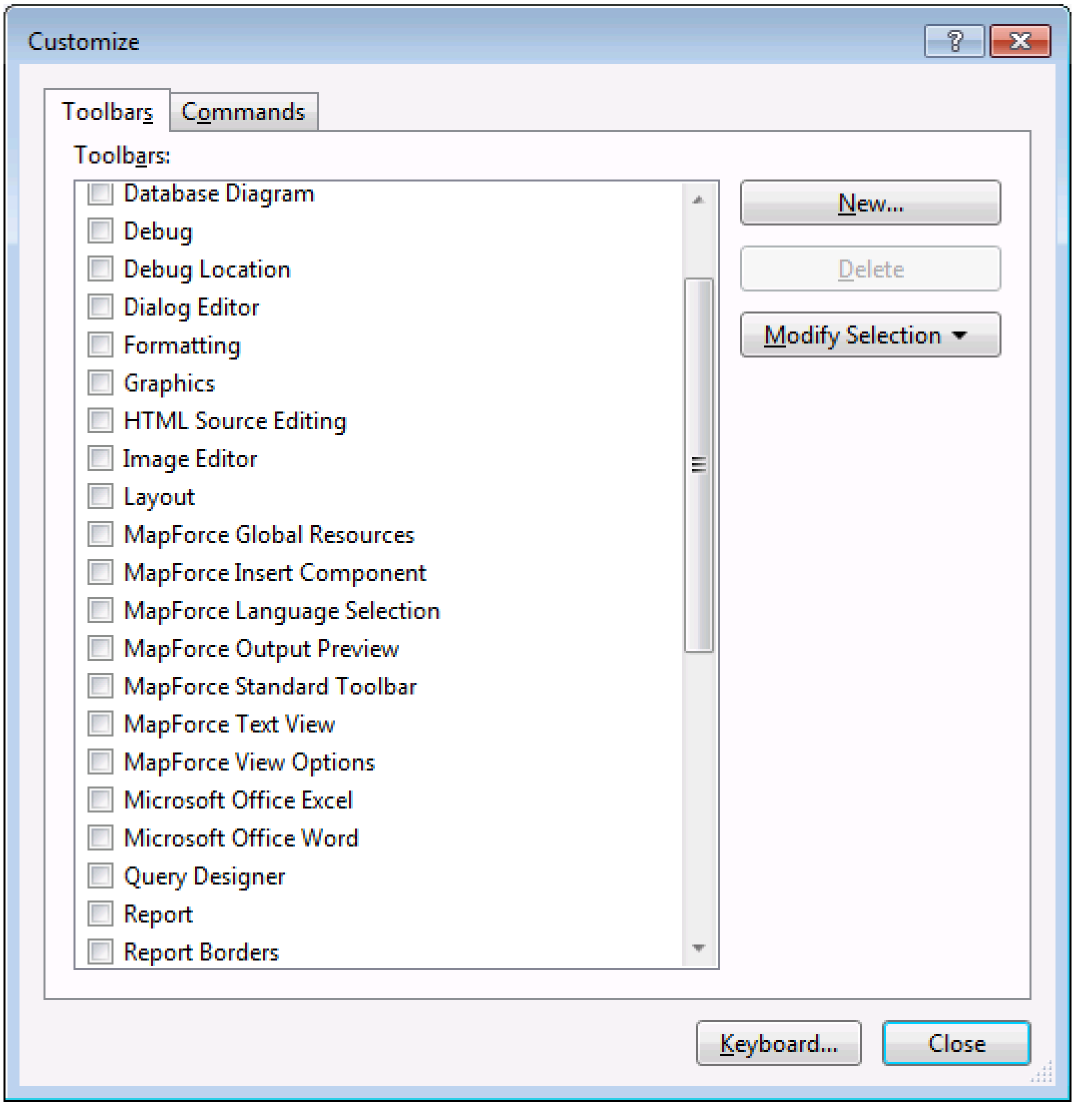
The MapForce Libraries window is not enabled by default in Visual Studio after you install the plug-in. If you work frequently with this window, you can enable it from the **View | MapForce | Library Window** menu (this menu becomes available in Visual Studio when there is a mapping file opened in the main window). Once the Libraries window is enabled, you can doc it to a particular position in the interface, like any other dockable component of Visual Studio.



The Libraries Window (Visual Studio 2012 with MapForce Enterprise edition plug-in)

Toolbar and Commands Customization

You can customize MapForce menus and toolbars from the **Tools | Options** menu of Visual Studio.



Customize

Toolbars | Commands

Toolbars:

□ Database Diagram
□ Debug
□ Debug Location
□ Dialog Editor
□ Formatting
□ Graphics
□ HTML Source Editing
□ Image Editor
□ Layout
□ MapForce Global Resources
□ MapForce Insert Component
□ MapForce Language Selection
□ MapForce Output Preview
□ MapForce Standard Toolbar
□ MapForce Text View
□ MapForce View Options
□ Microsoft Office Excel
□ Microsoft Office Word
□ Query Designer
□ Report
□ Report Borders

Customize dialog box (Visual Studio 2012 with MapForce Enterprise Edition plug-in)

Help and Support

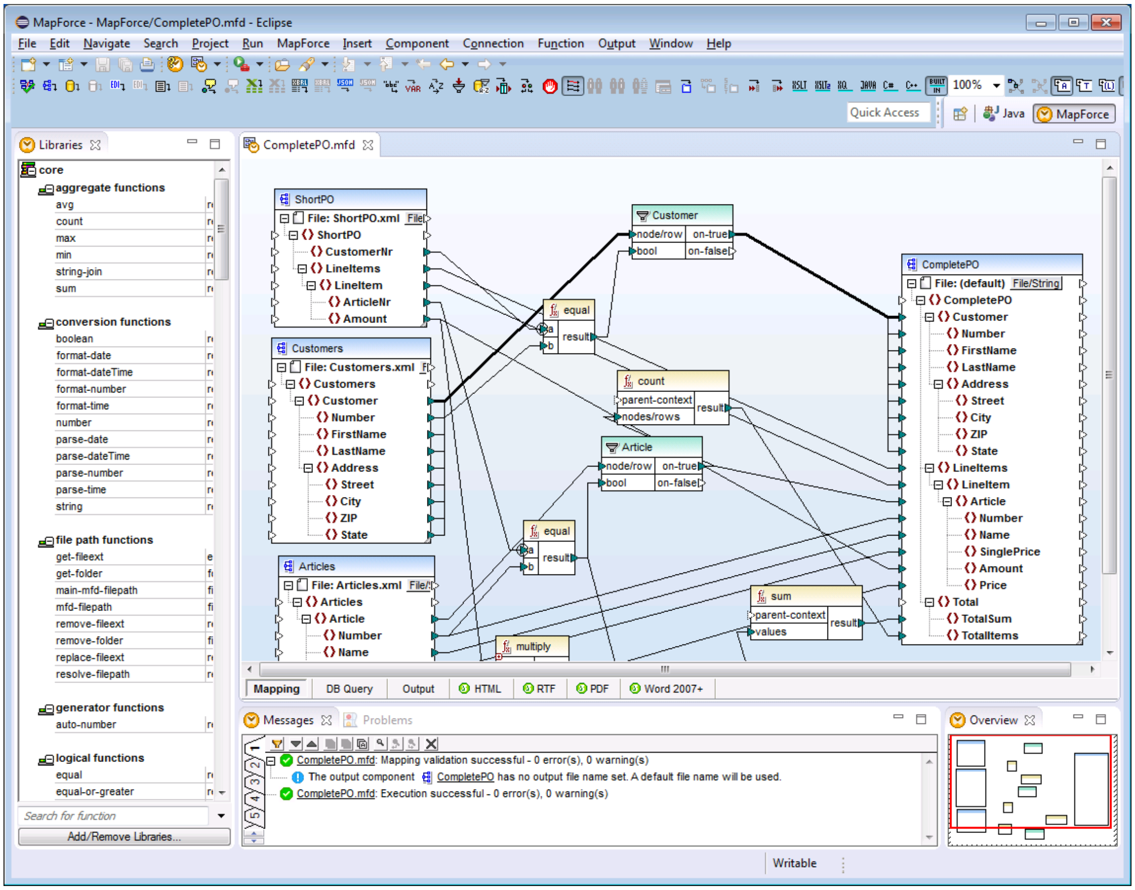
MapForce Help, Support Center, Check for Updates and About menus are available in the **Help | MapForce Help** menu of Visual Studio.

Chapter 12

MapForce Plug-in for Eclipse

12 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 4.5 / 4.6 / 4.7 and access MapForce functionality directly from Eclipse.



CompletePO.mfd: Mapping validation successful - 0 error(s), 0 warning(s)

The output component CompletePO has no output file name set. A default file name will be used.

CompletePO.mfd: Execution successful - 0 error(s), 0 warning(s)

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
- The MapForce Perspective
- Accessing Common Menus and Functions
- Working with Mapping and Projects
- Extending MapForce Plug-in for Eclipse

12.1 Installing the MapForce Plug-in for Eclipse

Prerequisites:

- Java Runtime Environment (JRE) 6.0 or later (see http://www.oracle.com/technetwork/java/javase/downloads/index.html). Install a 32-bit or 64-bit JRE to match your version of MapForce (32-bit or 64-bit).
- Eclipse Platform 4.5 / 4.6 / 4.7 (see http://www.eclipse.org). Install a 32-bit or 64-bit Eclipse to match your version of MapForce (32-bit or 64-bit).
- MapForce Enterprise or Professional Edition.

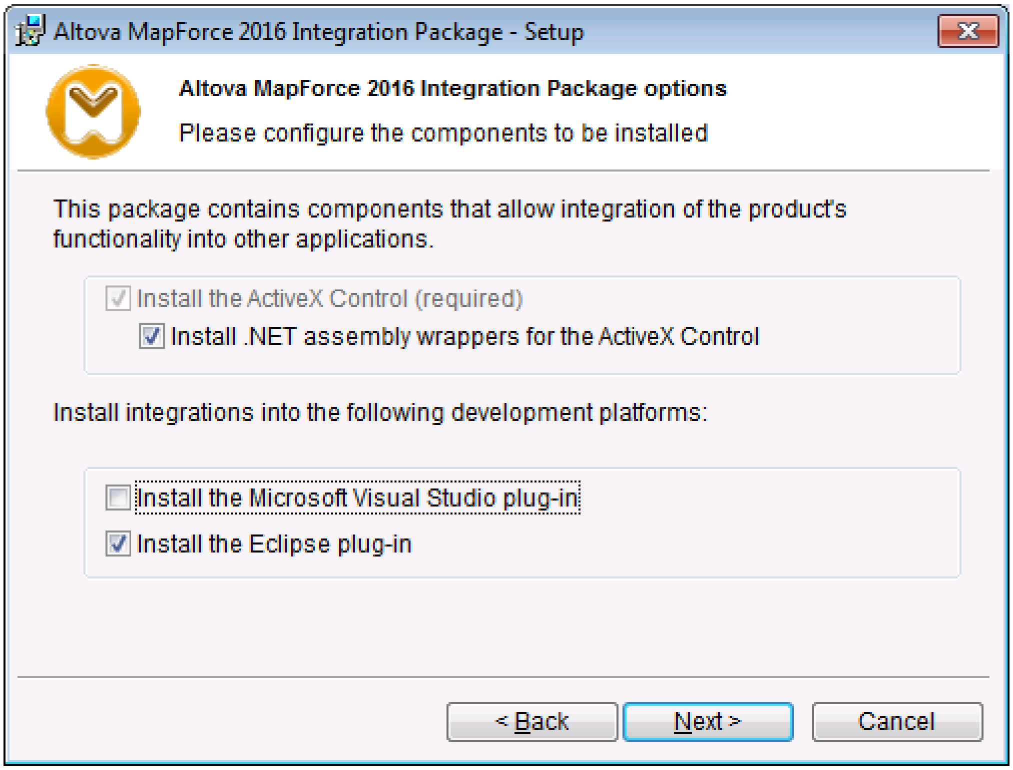
> If you installed Eclipse 4.5 using the Eclipse installer, it is not possible to run on the same machine both the 32-bit and 64-bit versions of the MapForce plug-in. This limitation originates in the Eclipse installer and does not apply if you install manually both versions of Eclipse (32-bit and 64-bit).

Installing the MapForce Plug-in for Eclipse

1. Download the MapForce Integration package from the Altova download page (https://www.altova.com/components/download).
2. Ensure that Eclipse is not running, and run the downloaded package.

> Eclipse must be closed while you install or uninstall the MapForce Integration Package.

3. When prompted, select the **Install the Eclipse plug-in** option, and then click **Next**.



Altova MapForce 2016 Integration Package - Setup

Altova MapForce 2016 Integration Package options

Please configure the components to be installed

This package contains components that allow integration of the product's functionality into other applications.

☑ Install the ActiveX Control (required)

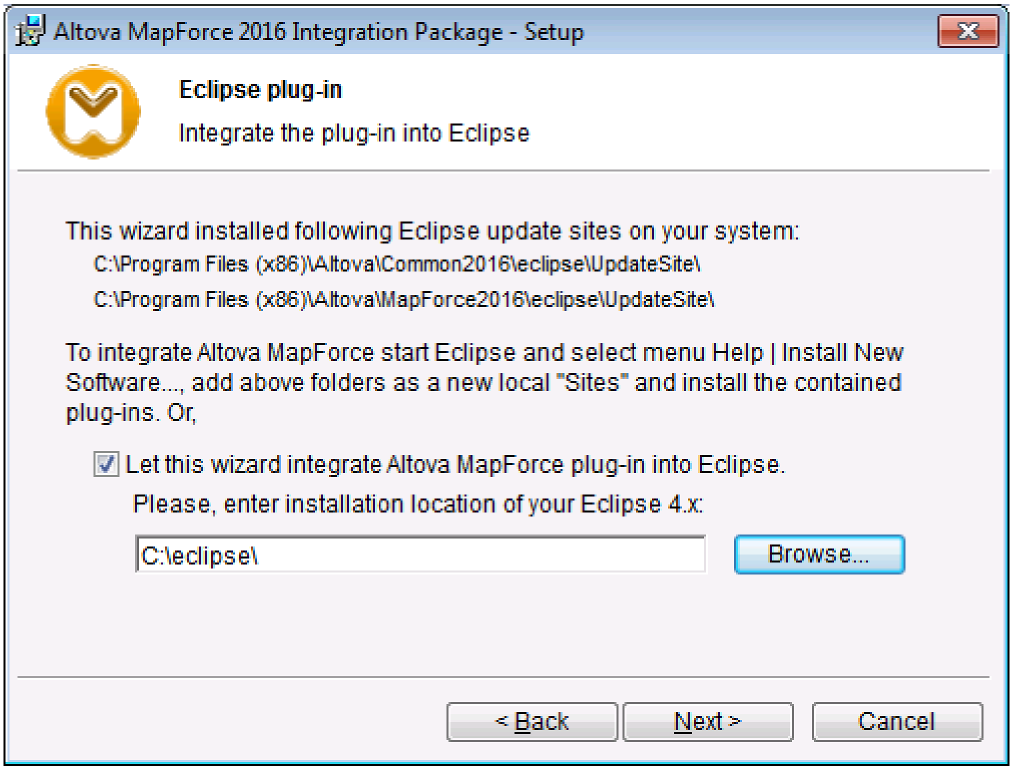
☑ Install .NET assembly wrappers for the ActiveX Control

Install integrations into the following development platforms:

☐ Install the Microsoft Visual Studio plug-in

☑ Install the Eclipse plug-in

4. When prompted to choose how the MapForce plug-in should be integrated into Eclipse, do one of the following:
   a. To complete the plug-in installation automatically (this is the recommended option), select **Let this wizard integrate Altova MapForce plug-in into Eclipse**, and browse for the directory where the Eclipse executable (`eclipse.exe`) is located.
   b. To complete the plug-in installation separately in Eclipse, click to clear the **Let this wizard...** check box (see instructions below).


Altova MapForce 2016 Integration Package - Setup
Eclipse plug-in
Integrate the plug-in into Eclipse

This wizard installed following Eclipse update sites on your system:
C:\Program Files (x86)\Altova\Common2016\eclipse\UpdateSite\
C:\Program Files (x86)\Altova\MapForce2016\eclipse\UpdateSite\

To integrate Altova MapForce start Eclipse and select menu Help | Install New Software..., add above folders as a new local "Sites" and install the contained plug-ins. Or,

☑ Let this wizard integrate Altova MapForce plug-in into Eclipse.

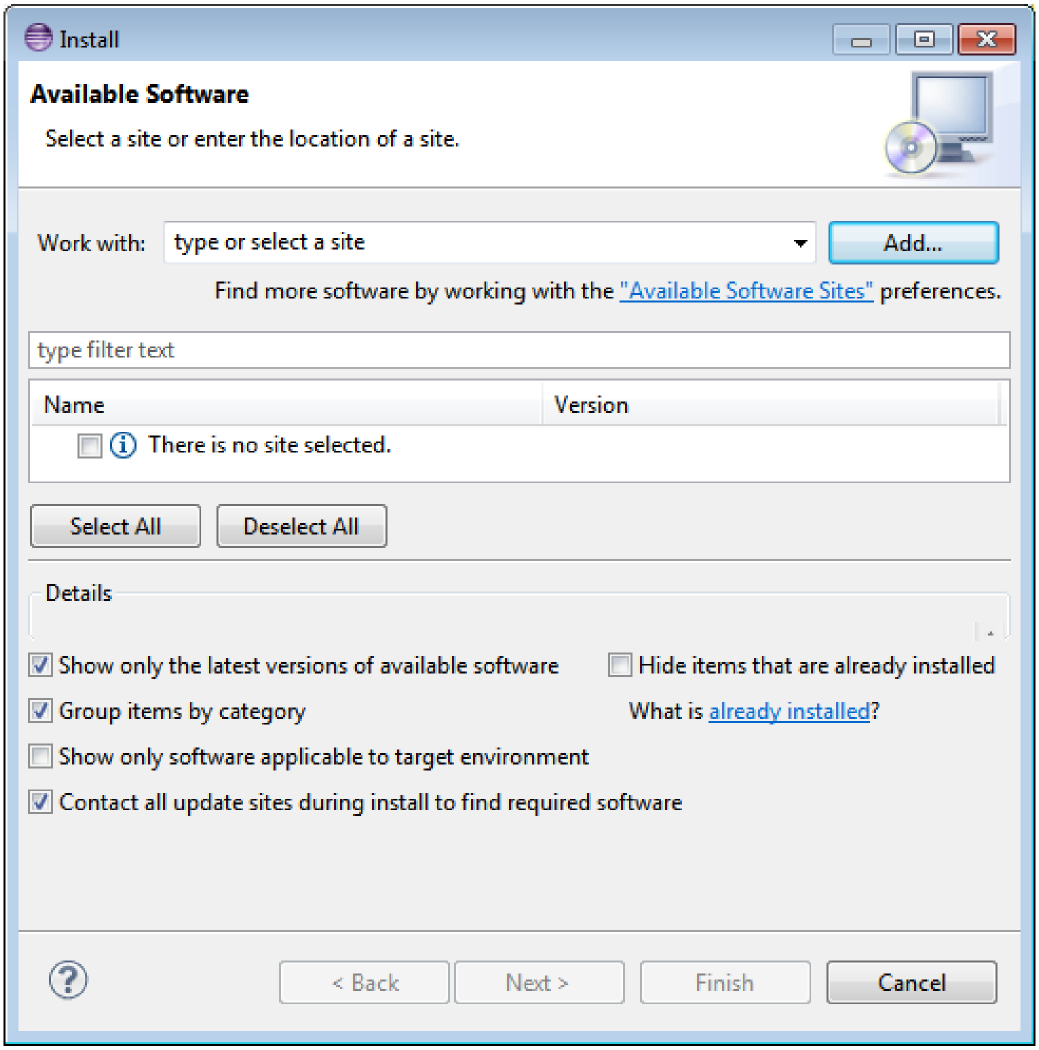
Please, enter installation location of your Eclipse 4.x:

C:\eclipse\

2. Click **Next**, and complete the installation. If you chose the automatic integration, the MapForce perspective and menus become available in Eclipse next time when you start Eclipse.

Integrating the MapForce plug-in for Eclipse manually

1. In Eclipse, click the menu command **Help | Install New Software**.
2. In the Install dialog that pops up (*screen shot below*), click the **Add** button.



Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
□ ⓘ There is no site selected.	

☑ Show only the latest versions of available software

☐ Hide items that are already installed

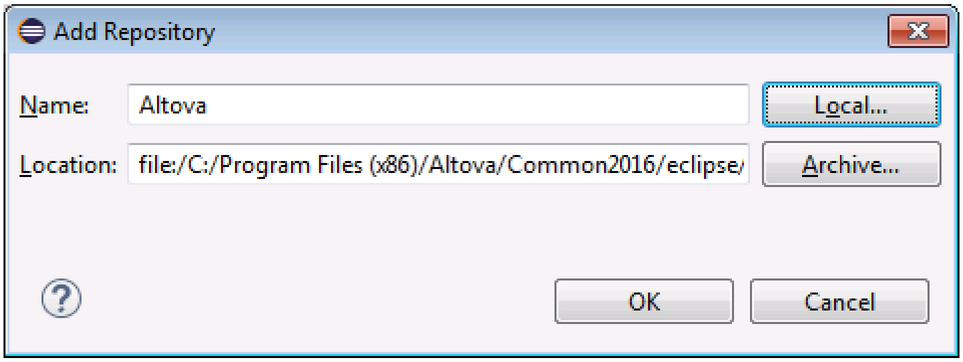
☑ Group items by category

What is already installed?

☐ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

3. In the Add Repository dialog that pops up (*screen shot below*), click the **Local** button.
4. Browse for the folder `c:\Program Files\Altova\Common2018\eclipse\UpdateSite`, and select it. Provide a name for the site (such as 'Altova'), and click **OK**.

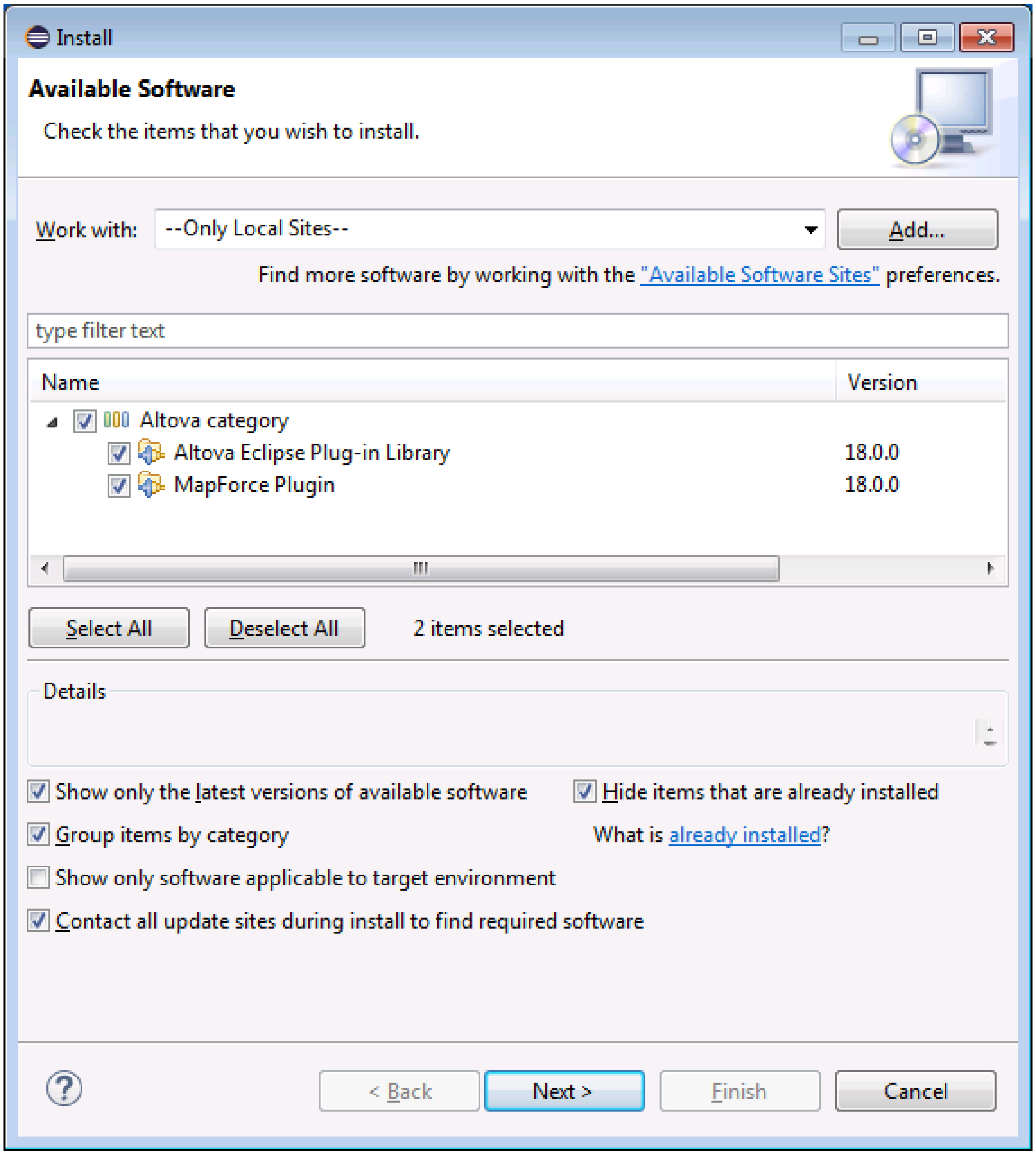


Name: Altova

Location: file:/C:/Program Files (x86)/Altova/Common2016/eclipse/

5. Repeat Steps 2 to 4, this time selecting the folder `c:\Program Files\Altova\MapForce2018\eclipse\UpdateSite`, and providing a name such as 'Altova MapForce'.
6. In the *Work With* combo box of the Install dialog, select the option -- *Only Local Sites* -- (*see screen shot below*). This causes all available plug-ins to be displayed in the pane

below. Check the top-level check box of the *Altova category* folder (*see screen shot below*). Then click the **Next** button.



Install

Available Software

Check the items that you wish to install.

Work with: --Only Local Sites--

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☑ Altova category	
☑ Altova Eclipse Plug-in Library	18.0.0
☑ MapForce Plugin	18.0.0

2 items selected

Details

☑ Show only the latest versions of available software

☑ Hide items that are already installed

☑ Group items by category

What is already installed?

□ Show only software applicable to target environment

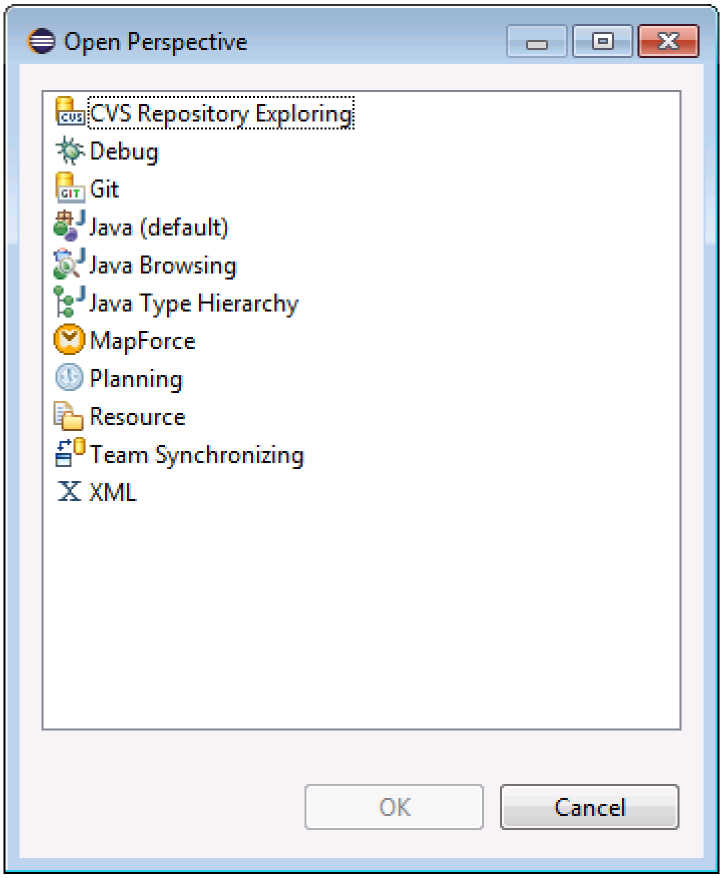
☑ Contact all update sites during install to find required software

7. An *Install Details* screen allows you to review the items to be installed. Click **Next** to proceed.
8. In the *Review Licenses* screen that appears, select *I accept the terms of the license agreement.* (No license agreement additional to your MapForce Enterprise or Professional Edition license is required for the MapForce plug-in.) Then 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.

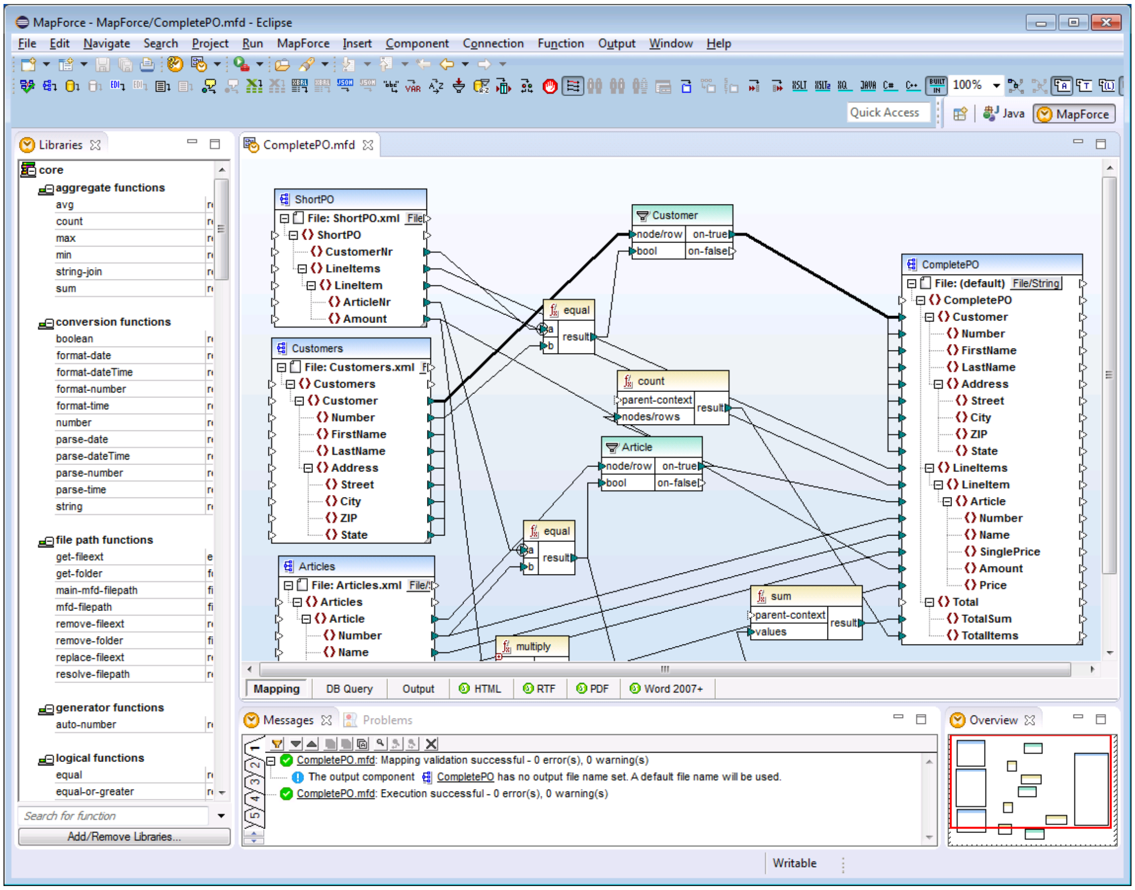
12.2 The MapForce Perspective

After you install the MapForce plug-in for Eclipse, a new perspective ("MapForce") becomes available in Eclipse. The layout of this perspective closely resembles the interface of the standalone edition of MapForce. To switch to the MapForce perspective, click **Window | Open Perspective | Other**, and choose MapForce from the list.



Selecting the MapForce perspective in Eclipse

The MapForce perspective is just like any other Eclipse perspective—you can switch to it whenever required in Eclipse (**Window | Navigation | Next Perspective**). You can also customize the views it contains, and various other options, from Eclipse preferences. (To customize the MapForce perspective in Eclipse 4.4, switch to the MapForce perspective, and then select the menu command **Window | Customize Perspective**). For more information about Eclipse perspectives, refer to Eclipse documentation. The following screen shot illustrates the Eclipse environment with the MapForce perspective switched on.



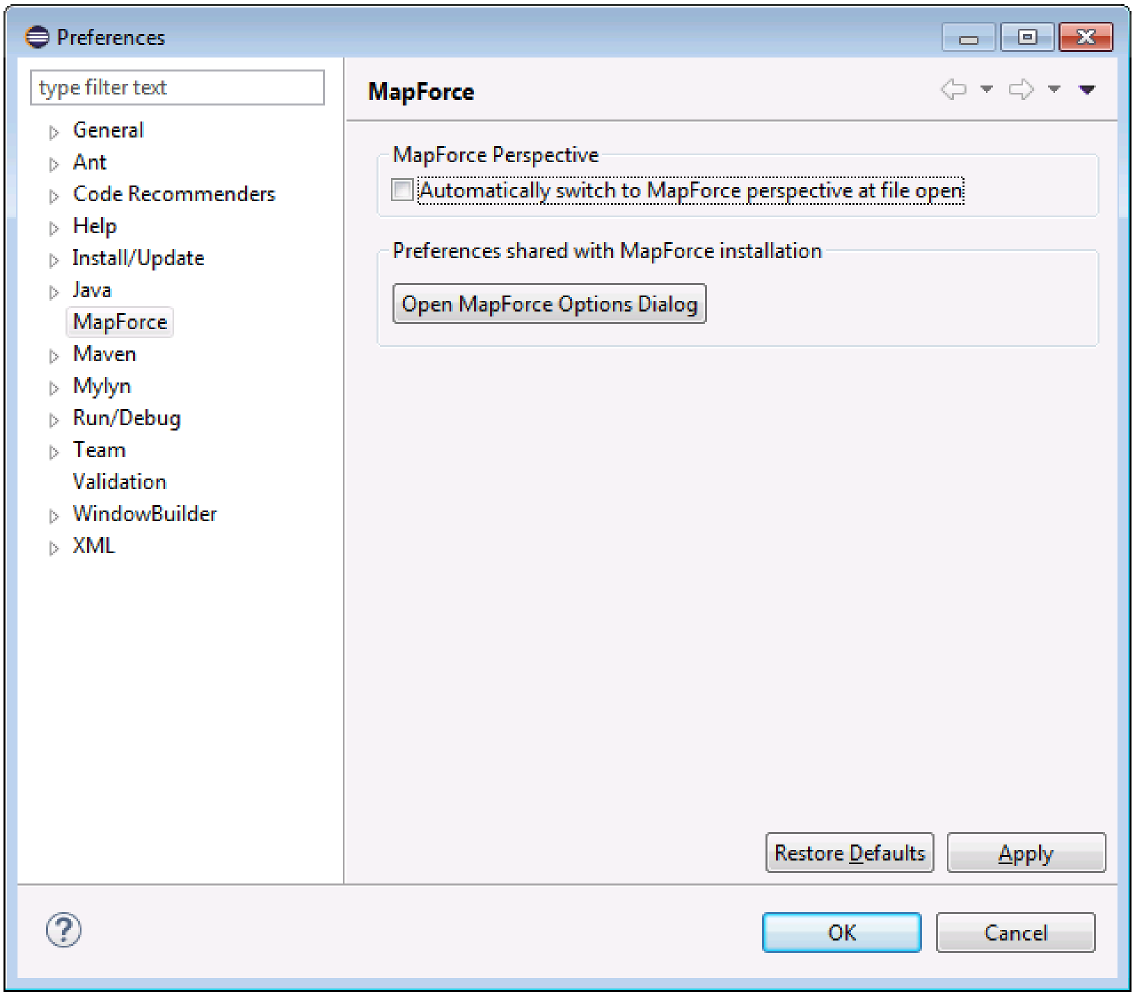
CompletePO.mfd: Mapping validation successful - 0 error(s), 0 warning(s)
The output component CompletePO has no output file name set. A default file name will be used.
CompletePO.mfd: Execution successful - 0 error(s), 0 warning(s)

MapForce perspective (MapForce Enterprise Edition plug-in for Eclipse)

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.

You can also configure Eclipse to switch to the MapForce perspective automatically when you open a MapForce mapping. To do this, select the menu command **Window | Preferences**. Select MapForce, and then select the **Automatically switch to MapForce perspective at file open** check box.



MapForce Perspective

□ Automatically switch to MapForce perspective at file open

Preferences shared with MapForce installation

Preferences dialog box

12.3 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).

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.

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.

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.

Global Resources

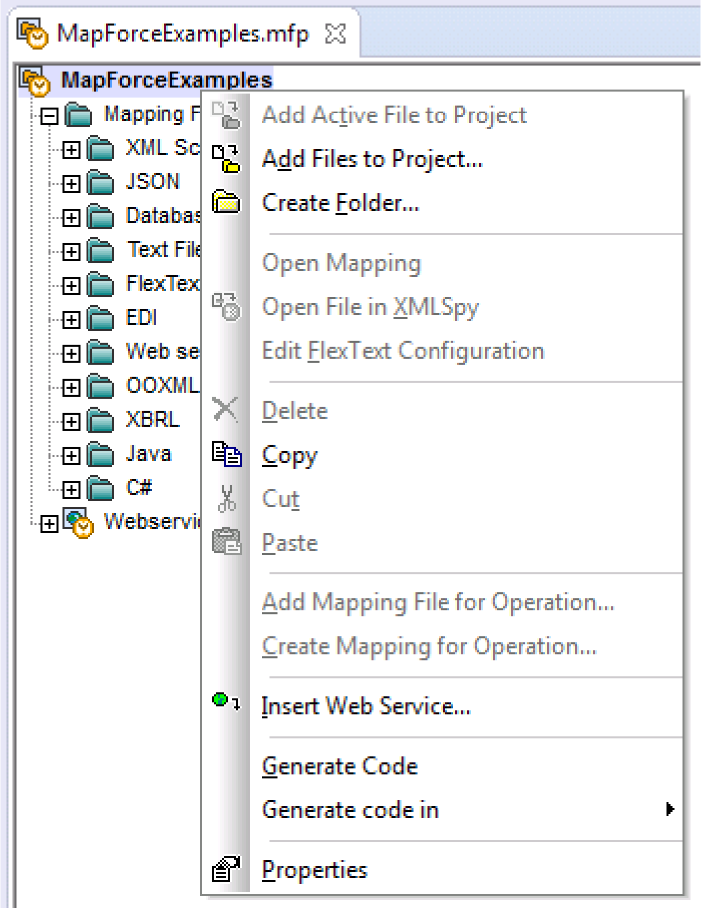
To access or manage Global Resources, do one of the following:

- 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:

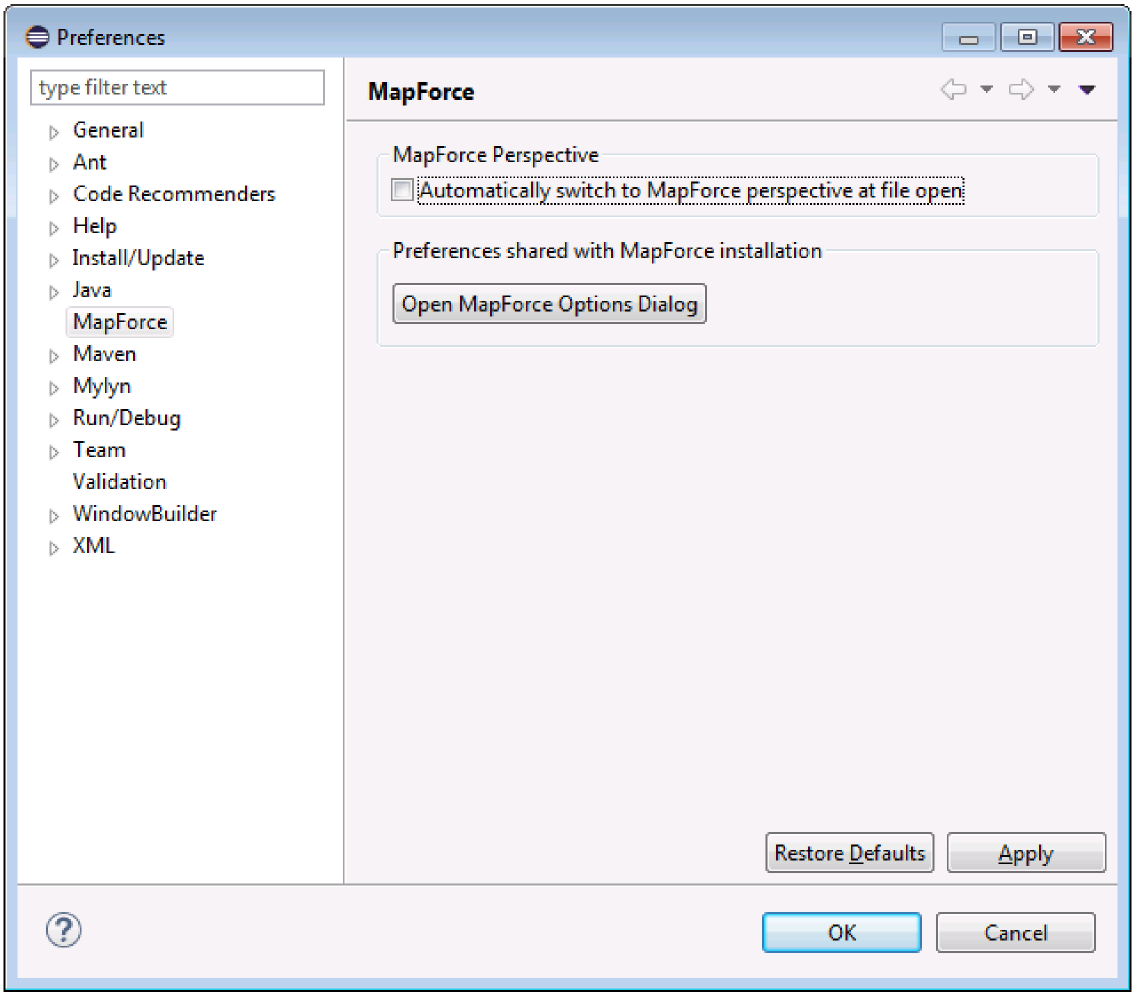
- The commands to open or save a project are available from the Eclipse **File** menu.
- Other project-specific commands are available as context commands. To display the context commands, create or open a MapForce project (.mfp) file in Eclipse, and then right-click the project.



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.

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**.



□ Automatically switch to MapForce perspective at file open

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 Eclipse interface when the MapForce perspective is switched on, see also The MapForce Perspective).

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.

12.4 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.

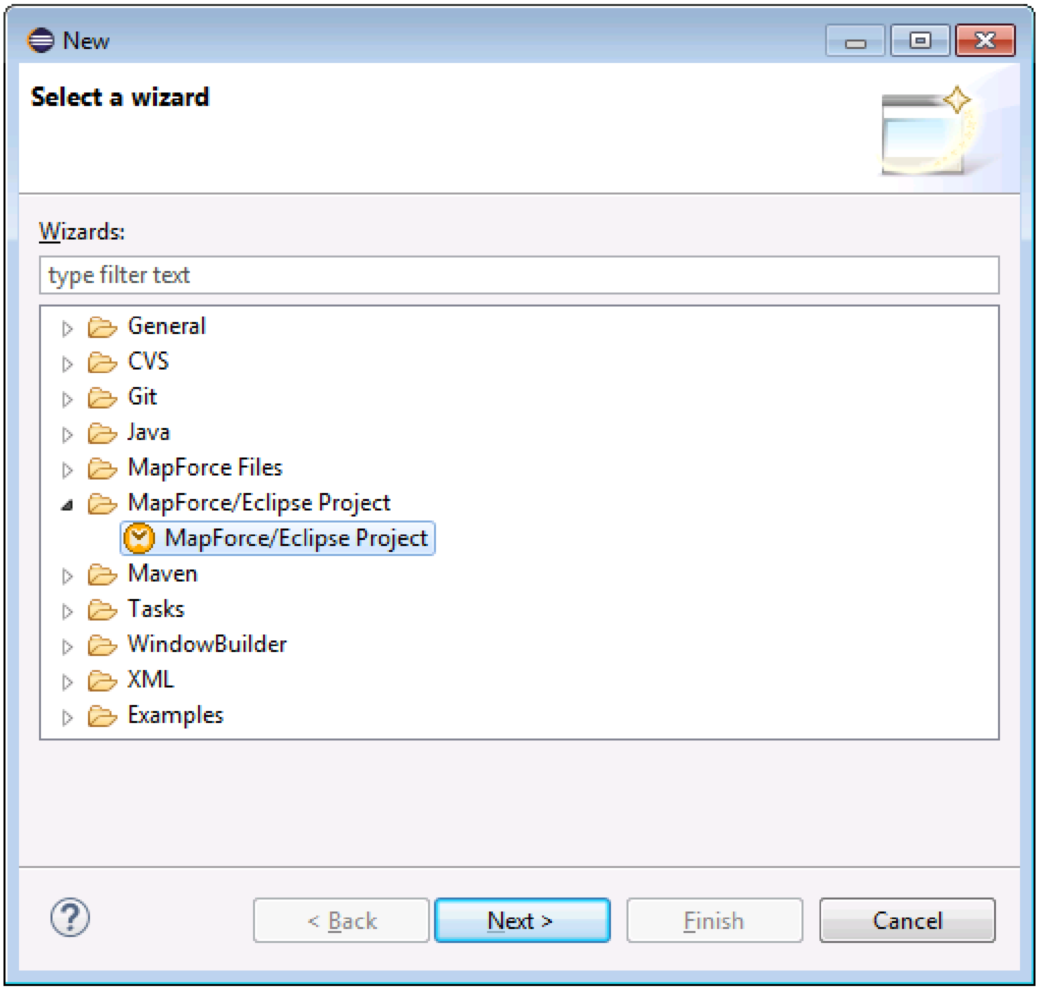
This section contains the following topics:

- Creating a MapForce/Eclipse Project
- Creating New Mappings
- Importing Existing Mappings into an Eclipse Project
- Configuring Automatic Build and Generation of MapForce Code

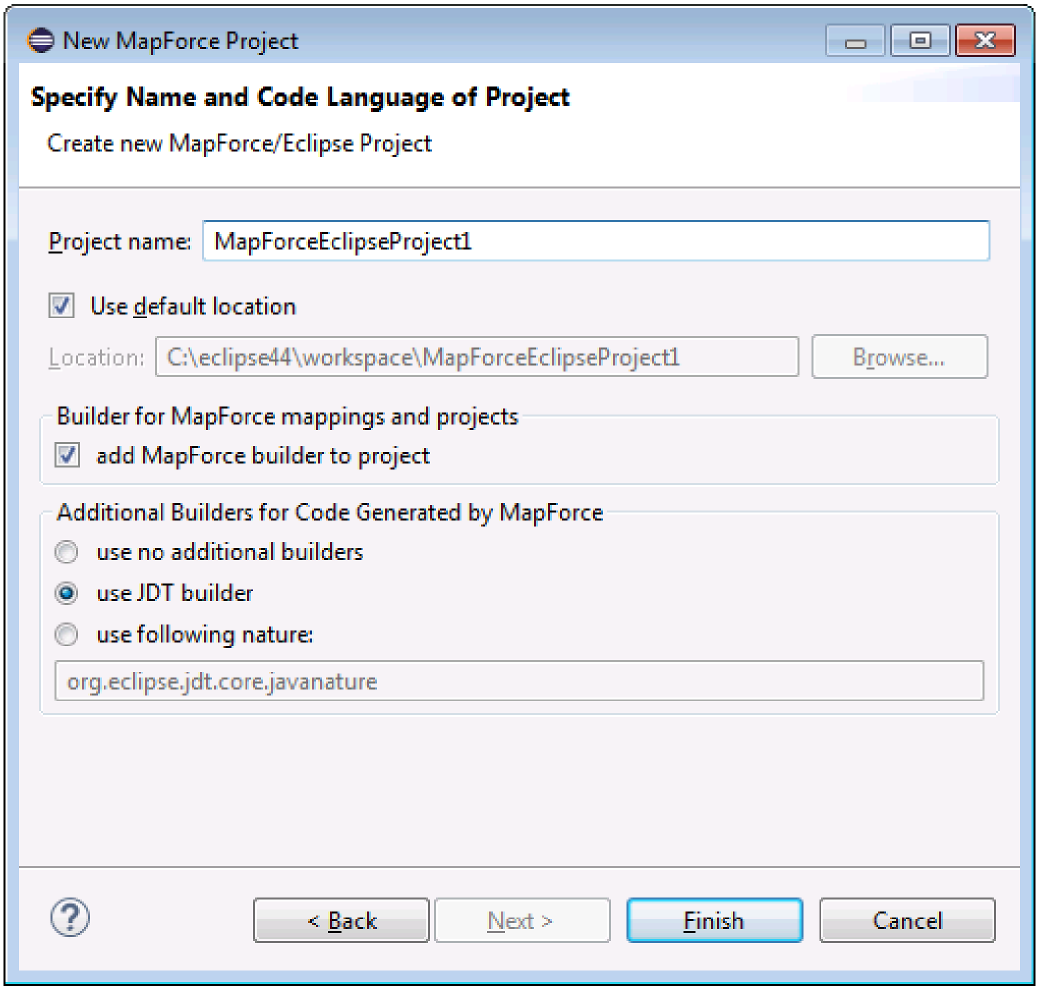
12.4.1 Creating a MapForce/Eclipse Project

To create a MapForce/Eclipse project:

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. Chick **Next**.



New MapForce Project

Specify Name and Code Language of Project

Create new MapForce/Eclipse Project

Project name: MapForceEclipseProject1

☑ Use default location

Location: C:\eclipse44\workspace\MapForceEclipseProject1

Builder for MapForce mappings and projects

☑ add MapForce builder to project

Additional Builders for Code Generated by MapForce

○ use no additional builders

● use JDT builder

○ use following nature:

org.eclipse.jdt.core.javanature

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**.

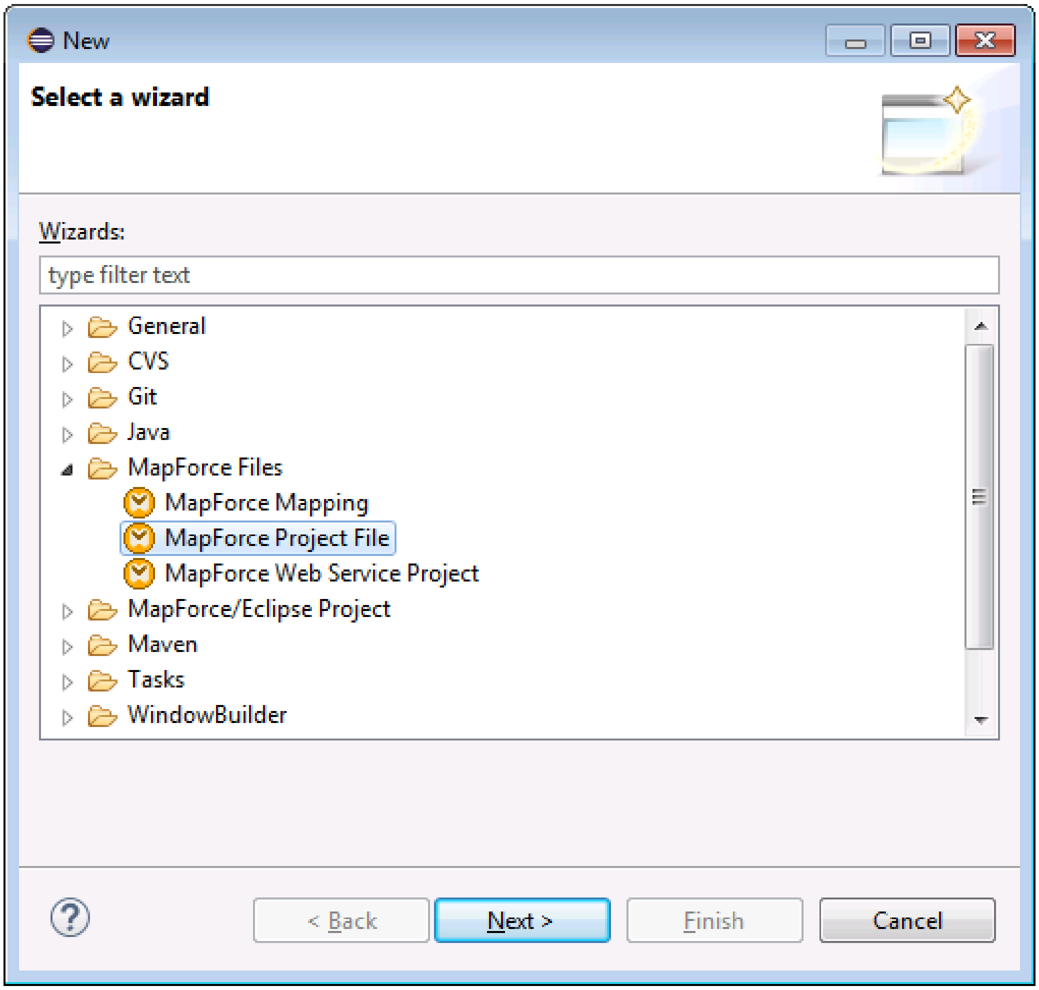
12.4.2 Creating New Mappings

You can create the following MapForce file types within an Eclipse project:

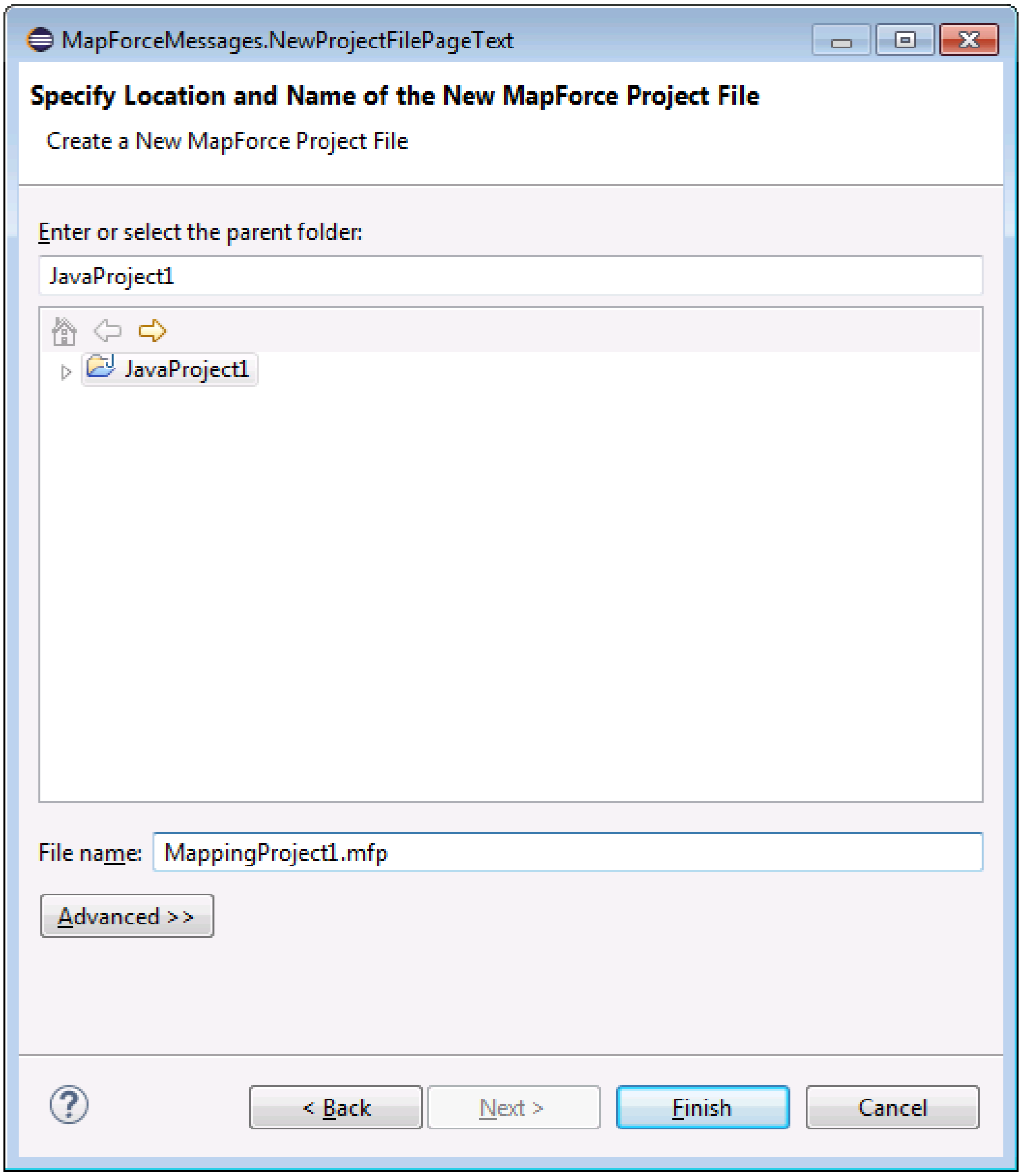
- MapForce mappings
- MapForce project files
- MapForce Web Service projects (available in MapForce Enterprise Edition)

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**.


MapForceMessages.NewProjectFilePageText

Specify Location and Name of the New MapForce Project File

Create a New MapForce Project File

Enter or select the parent folder:

JavaProject1

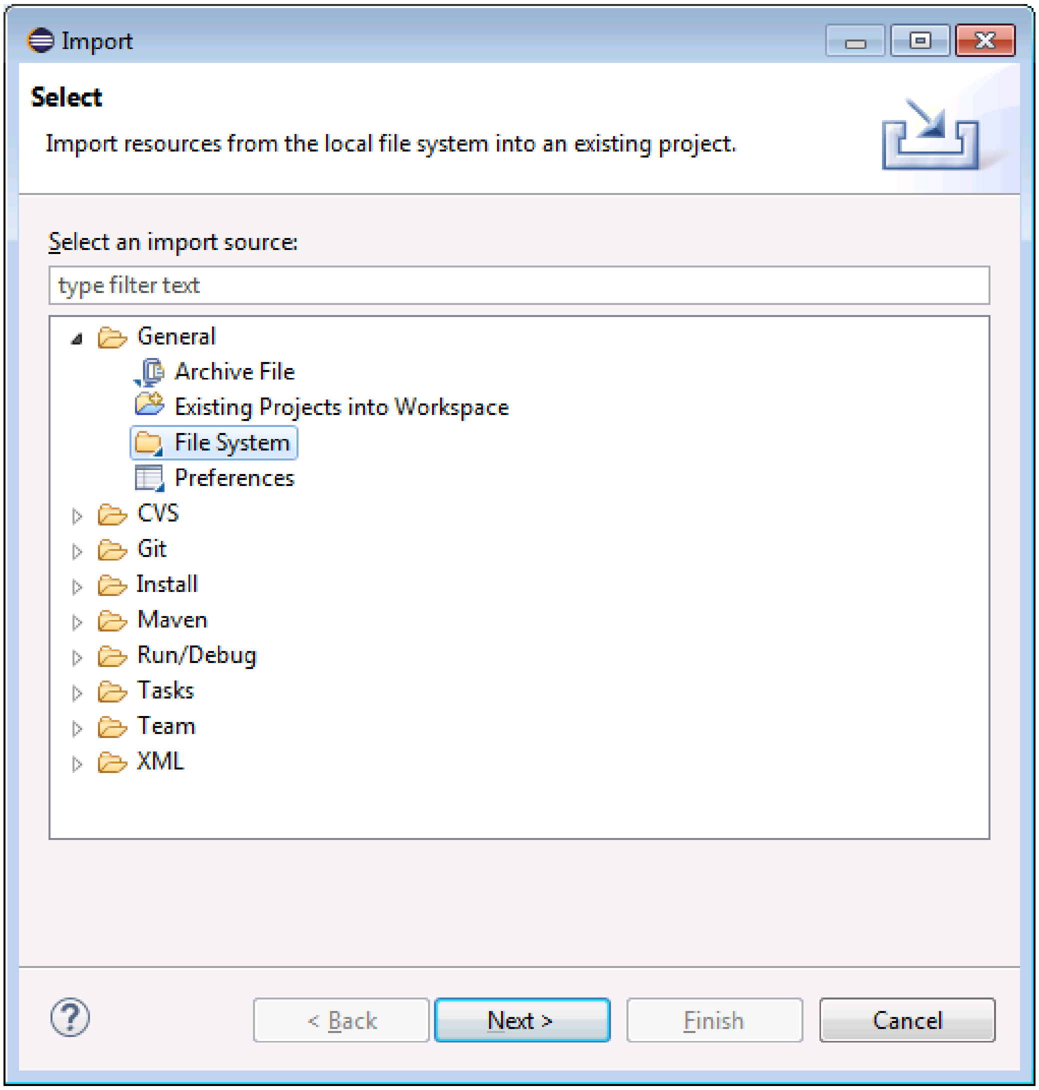
File name: MappingProject1.mfp

4. Select a parent folder in your existing project, and then click **Finish**.

12.4.3 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**.


Import

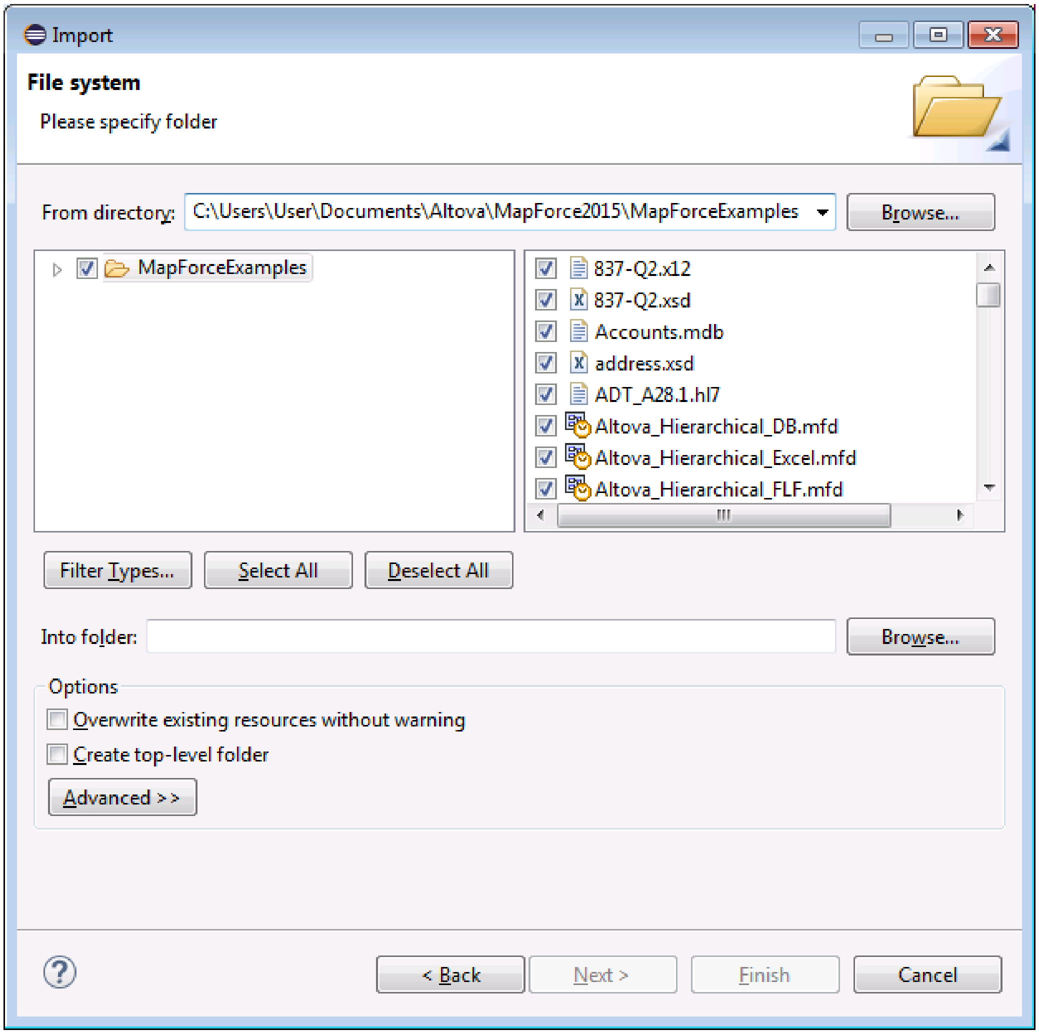
Select

Import resources from the local file system into an existing project.

Select an import source:

type filter text

3. Select **File System**, and then click **Next**.



Import

File system

Please specify folder

From directory: C:\Users\User\Documents\Altova\MapForce2015\MapForceExamples

☑ MapForceExamples

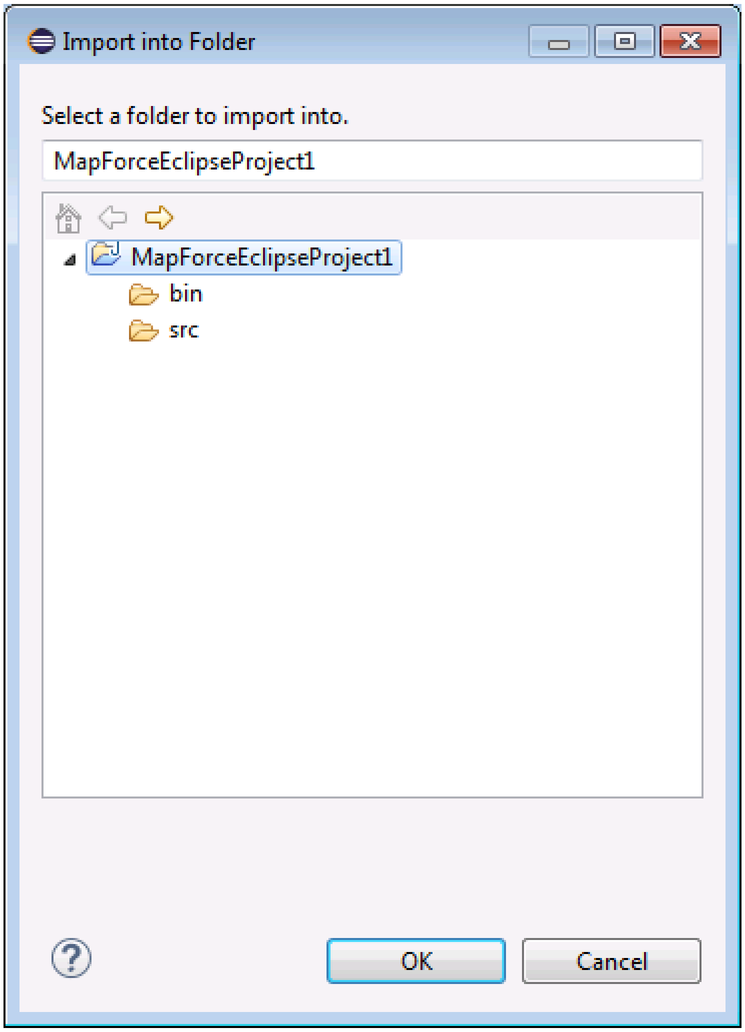
☑ 837-Q2.x12
☑ 837-Q2.xsd
☑ Accounts.mdb
☑ address.xsd
☑ ADT_A28.1.hl7
☑ Altova_Hierarchical_DB.mfd
☑ Altova_Hierarchical_Excel.mfd
☑ Altova_Hierarchical_FLF.mfd

Into folder:

Options

□ Overwrite existing resources without warning
□ Create top-level folder

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**.

12.4.4 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). 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>
```

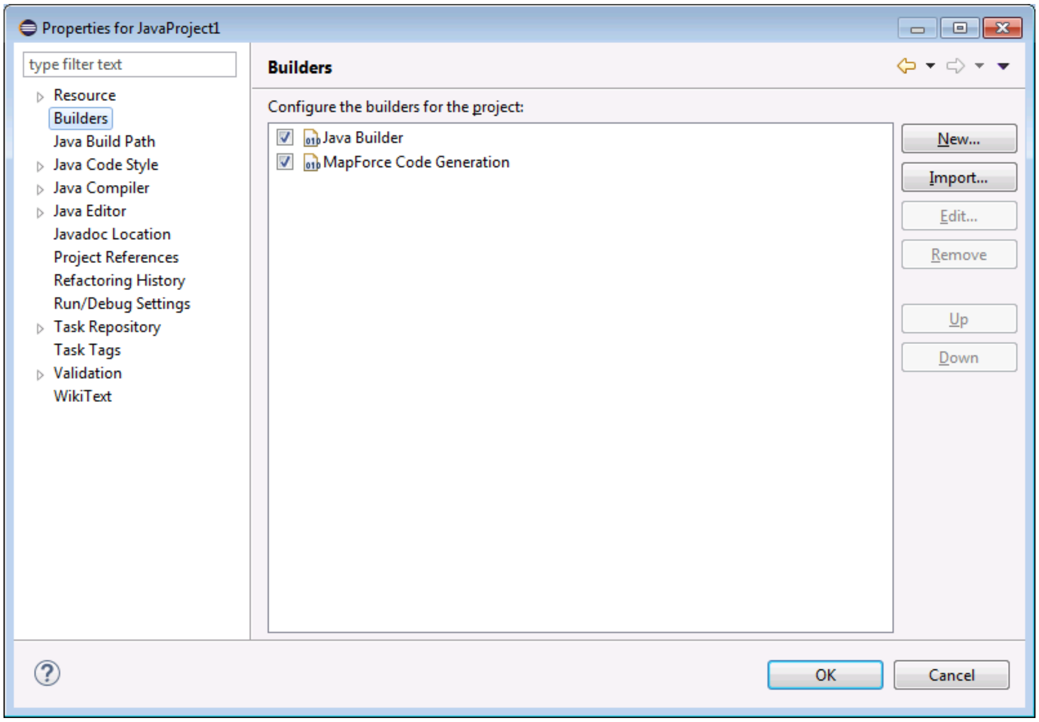
Tip: You can quickly open the **.project** file from the Navigator view of Eclipse (To enable this view, select the menu command **Window | Show View | Navigator**).

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**.


Properties for JavaProject1

Builders

Configure the builders for the project:

☑ Java Builder
☑ MapForce Code Generation

3. Click to clear the **MapForce Code Generation** check box.

12.5 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 API.

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%.

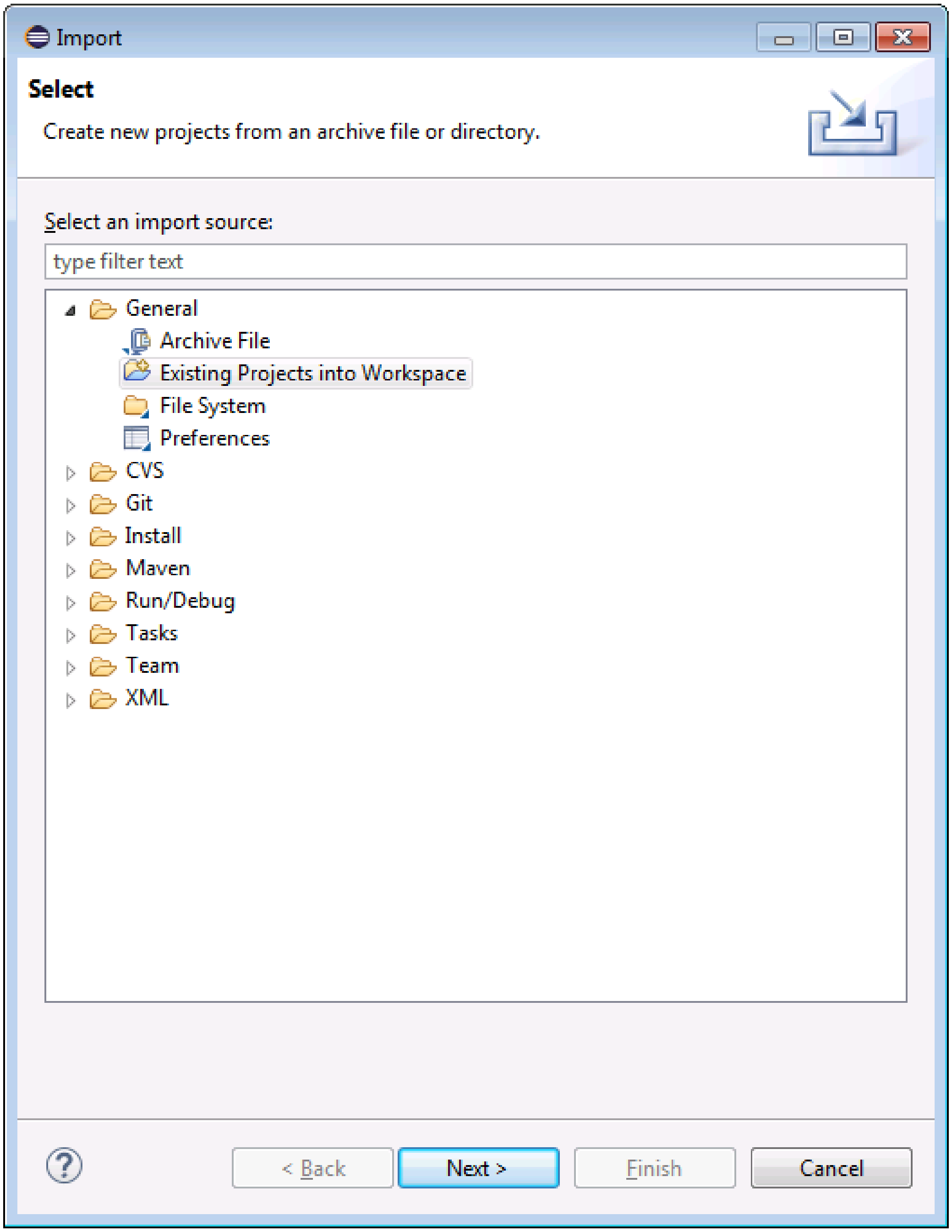
The JavaDoc documentation of the extension point is available in the MapForce plug-in installation directory (typically, **C:\Program Files\Altova\MapForce2018\eclipse\docs\**).

Before you install and run the sample MapForce plug-in, ensure that the following prerequisites are met:

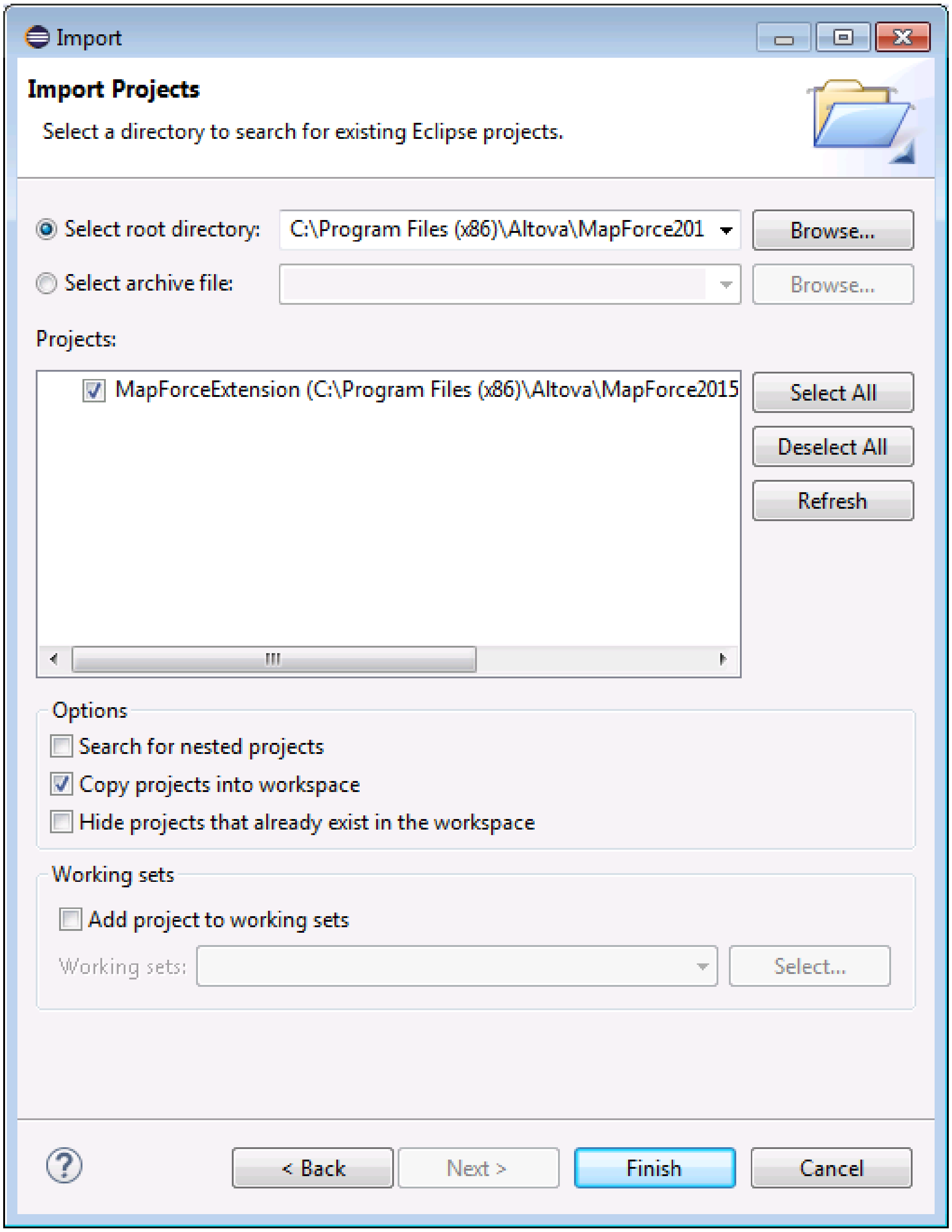
- You are using 32-bit Java, 32-bit Eclipse, 32-bit MapForce and 32-bit MapForce Integration Package.
- The JDT (Java Development Tools) plug-in is installed.
- The Eclipse PDE (plug-in development environment) is installed.

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\MapForce2018\eclipse \workspace\MapForceExtension.**



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: C:\Program Files (x86)\Altova\MapForce201

○ Select archive file:

Projects:

☑ MapForceExtension (C:\Program Files (x86)\Altova\MapForce2015

Options

☐ Search for nested projects

☑ Copy projects into workspace

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

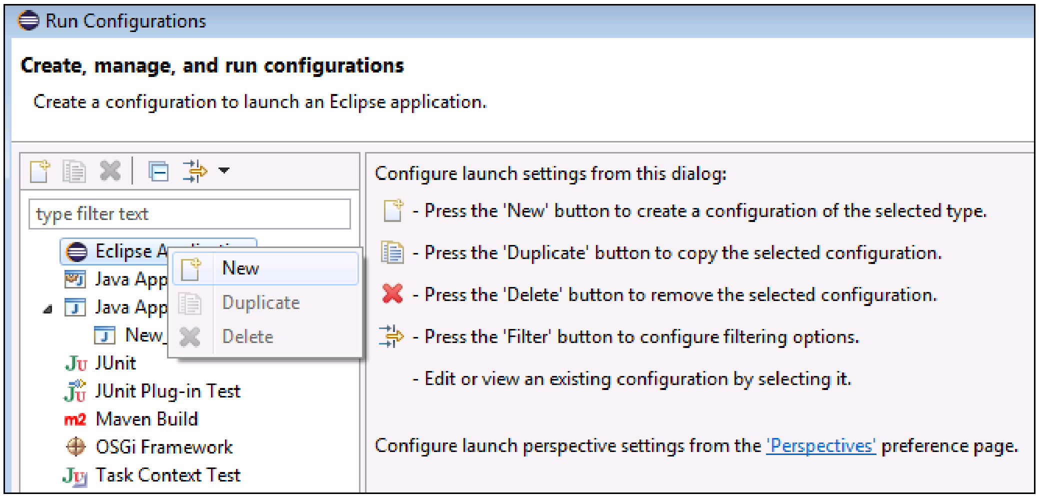
Working sets:

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.)



Run Configurations

Create, manage, and run configurations

Create a configuration to launch an Eclipse application.

type filter text

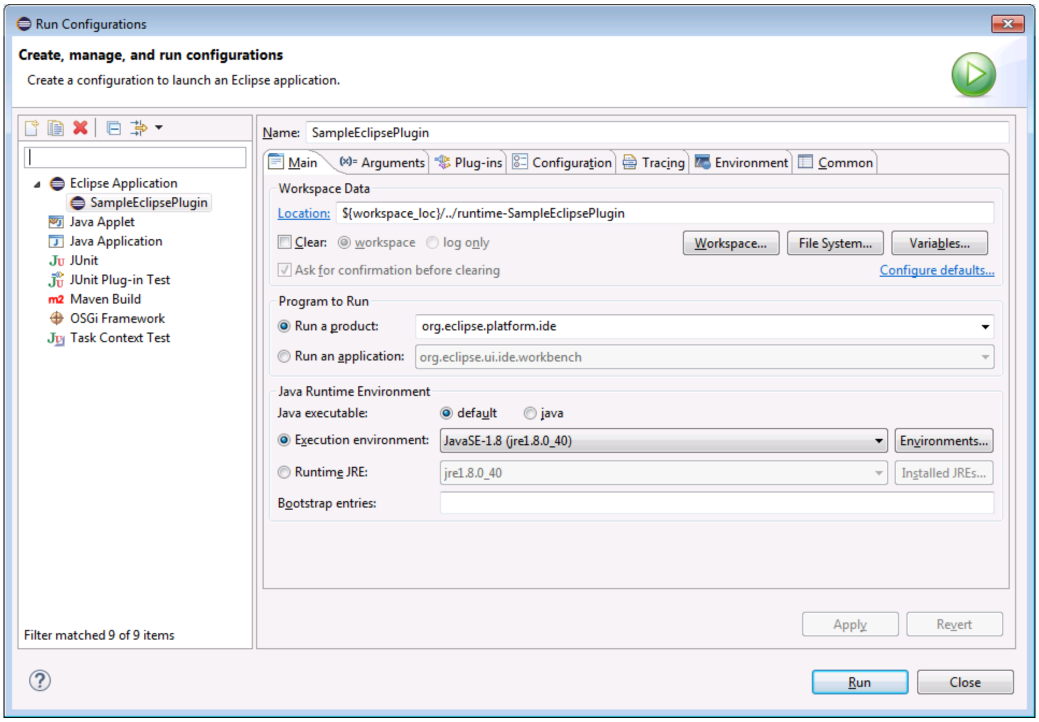
- Eclipse A
- Java App
- Java App
  - New_
- JUnit
- JUnit Plug-in Test
- Maven Build
- OSGi Framework
- Task Context Test

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

4. Enter a name for your new configuration (in this example, *SampleMapForcePlugin*), and then click **Apply**.



Run Configurations

Create, manage, and run configurations

Create a configuration to launch an Eclipse application.

- Eclipse Application
  - SampleEclipsePlugin
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Maven Build
- OSGi Framework
- Task Context Test

Filter matched 9 of 9 items

Name: SampleEclipsePlugin

Workspace Data

Location: ${workspace_loc}/../runtime-SampleEclipsePlugin

□ Clear: ○ workspace ○ log only

☑ Ask for confirmation before clearing

Configure defaults...

Program to Run

● Run a product: org.eclipse.platform.ide

○ Run an application: org.eclipse.ui.ide.workbench

Java Runtime Environment

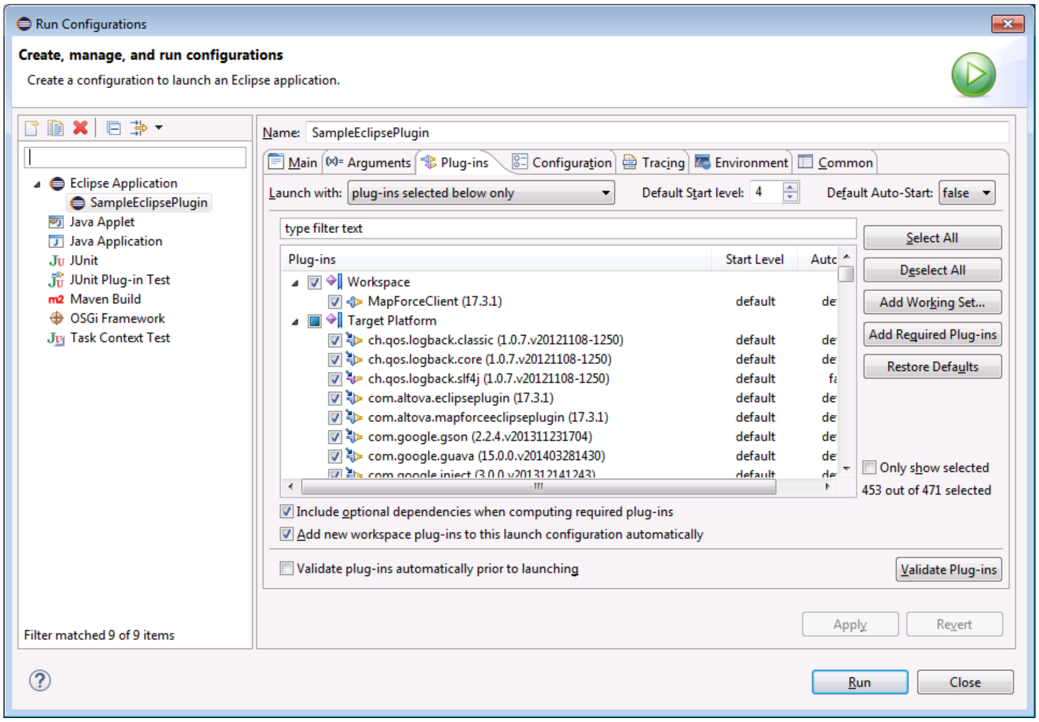
Java executable: ● default ○ java

● Execution environment: JavaSE-1.8 (jre1.8.0_40)

○ Runtime JRE: jre1.8.0_40

Bootstrap entries:

5. Check that the **MapForceClient** workspace plug-in is selected in the 'Plug-ins' tab.



Name: SampleEclipsePlugin

Launch with: plug-ins selected below only

Default Start level: 4

Default Auto-Start: false

Plug-ins	Start Level	Autc
☑ Workspace		
☑ MapForceClient (17.3.1)	default	de
■ Target Platform		
☑ ch.qos.logback.classic (1.0.7.v20121108-1250)	default	de
☑ ch.qos.logback.core (1.0.7.v20121108-1250)	default	de
☑ ch.qos.logback.slf4j (1.0.7.v20121108-1250)	default	fa
☑ com.altova.eclipseplugin (17.3.1)	default	de
☑ com.altova.mapforceeclipseplugin (17.3.1)	default	de
☑ com.google.gson (2.2.4.v201311231704)	default	de
☑ com.google.guava (15.0.0.v201403281430)	default	de
☑ com.google.inject (3.0.0.v201312141243)	default	de

□ Only show selected

453 out of 471 selected

☑ Include optional dependencies when computing required plug-ins

☑ Add new workspace plug-ins to this launch configuration automatically

□ Validate plug-ins automatically prior to launching

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%.

Chapter 13

Menu Reference

13 Menu Reference

This reference section contains a description of the MapForce menu commands.

13.1 File

New

Creates a new mapping document, or mapping project (.mfp) .

Open

Opens previously saved mapping design (.mfd) , or mapping project (.mfp) files. Note that it is not possible to open mapping files which contain features not available in your MapForce edition.

Save

Saves the currently active mapping using the currently active file name.

Save As

Saves the currently active mapping with a different name, or allows you to supply a new name if this is the first time you save it.

Save All

Saves all currently open mapping files.

Reload

Reloads the currently active mapping file. You are asked if you want to lose your last changes.

Close

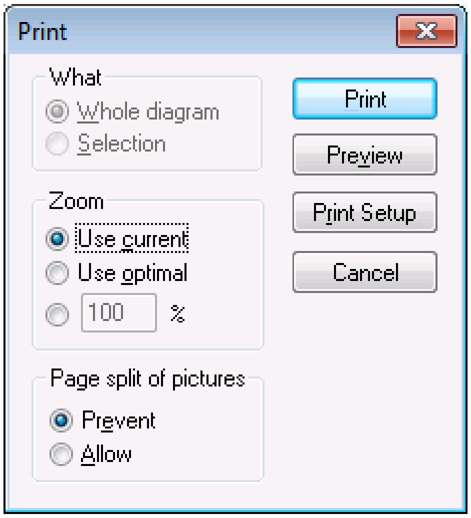
Closes the currently active mapping file. You are asked if you want to save the file before it closes.

Close All

Closes all currently open mapping files. You are asked if you want to save any of the unsaved mapping files.

Print

Opens the Print dialog box, from where you can print out your mapping as hard copy.



Print

What
○ Whole diagram
○ Selection

Zoom
● Use current
○ Use optimal
○ 100 %

Page split of pictures
● Prevent
○ Allow

Print dialog box

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 the graphics to be split over several pages or not.

Print Preview

Opens the same Print dialog box with the same settings as described above.

Print Setup

Opens the Print Setup dialog box in which you can define the printer you want to use and the paper settings.

Validate Mapping

Validates that all mappings (connectors) are valid and displays any warnings or errors (see Validating mappings).

Mapping settings

Opens the Mapping Settings dialog box where you can define the document-specific settings (see Changing the mapping settings).

Generate code in selected language

Generates code in the currently selected language of your mapping. The currently selected language is visible as a highlighted programming language icon in the toolbar: XSLT, XSLT 2, XQuery, Java, C#, or C++.

Generate code in | XSLT (XSLT2)

This command generates the XSLT file(s) needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box where you select the location of the XSLT file. The name of the generated XSLT file(s) is defined in the Mapping Settings dialog box (see Changing the mapping settings).

Generate code in | XQuery

This command generates the XQuery file(s) needed for the transformation from the source file(s).

Selecting this option opens the Browse for Folder dialog box where you select the location of the XQuery file. The name of the generated XQuery file(s) is defined in the Mapping Settings dialog box (see Changing the mapping settings).

Generate code in | Java | C# | C++

These commands generate source code for a complete application program needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box, where you select the location of the generated files. The names of the generated application files (as well as the project files: *.csproj C# project file, *.sln solution file, *.vcproj visual C++ project file) are defined in the Mapping Settings dialog box (see Changing the mapping settings).

The file name created by the executed code appears in the **Output XML File** box of the Component settings dialog box if the target is an XML/Schema document.

Compile to MapForce Server Execution File

Generates a file that can be executed by MapForce Server to run the mapping transformation (see Compiling a MapForce mapping).

Deploy to FlowForce Server

Deploys the currently active mapping to the FlowForce Server (see Deploying a MapForce mapping).

Generate documentation

Generates documentation of your mapping projects in great detail in various output formats (see Generating and Customizing Mapping Documentation).

Recent files

Displays a list of the most recently opened files.

Exit

Exits the application. You are asked if you want to save any unsaved files.

13.2 Edit

Most of the commands in this menu become active when you view the result of a mapping in the **Output** tab, or preview XSLT code in the XSLT tab.

Undo

MapForce has an unlimited number of "Undo" steps that you can use to retrace you mapping steps.

Redo

The redo command allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.

Find

Allows you to search for specific text in either the XSLT, XSLT2, XQuery or Output tab.

Find Next F3

Searches for the next occurrence of the same search string.

Find Previous Shift F3

Searches for the previous occurrence of the same search string.

Cut/Copy/Paste/Delete

The standard windows Edit commands, allow you to cut, copy etc., any components or functions visible in the mapping window.

Select all

Selects all components in the Mapping tab, or the text/code in the XSLT, XSLT2, XQuery or Output tab.

13.3 Insert

XML Schema / File

Adds to the mapping an XML schema or instance file. If you select an XML file which references a schema, no additional information is required for the mapping. If you select an XML file without a schema reference, you are prompted to generate a matching XML schema automatically (see Generating an XML Schema). If you select an XML schema file, you are prompted to include optionally an XML instance file which supplies the data for preview.

Database

Adds to the mapping a database as source or target component (see Databases and MapForce).

Text file

Adds to the mapping a flat file document, such as CSV or a fixed-length text file. Both types of file can be used as source and target components. **Insert Input**
When the mapping window displays a mapping, this command adds an input component to the mapping (see Supplying Parameters to the Mapping). When the mapping window displays a user-defined function, this command adds an input component to the user-defined function (see Defining Complex Input Components).

Insert Output

When the mapping window displays a mapping, this command adds an output component to the mapping (see Returning String Values from a Mapping). When the mapping window displays a user-defined function, this command adds an output component to the user-defined function (see Defining Complex Output Components).

Constant

Inserts a constant which supplies fixed data to an input connector. The data is entered into a dialog box when creating the component. You can select the following types of data: String, Number and All other.

Variable

Inserts an Intermediate Variable which is equivalent to a regular (non-inline) user-defined function. Variables are structural components, without instance files, and are used to simplify the mapping process (see Intermediate variables).

Sort: Nodes/Rows

Inserts a component which allows you to sort nodes (see Sort Nodes/Rows).

Filter: Nodes/Rows

Inserts a component that uses two input and output parameters: **node/row** and **bool**, and **on-true**, **on-false**. If the Boolean is true, then the value of the node/row parameter is forwarded to the on-true parameter. If the Boolean is false, then the complement value is passed on to the on-false parameter. For more information, see Filters and Conditions.

SQL-WHERE/ORDER

Inserts a component which allows you to filter database data conditionally (see SQL WHERE / ORDER Component).

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 (for example, month numbers to month names). For more information, see Using Value-Maps.

IF-Else Condition

Inserts a component of type "If-Else Condition" (see Filters and Conditions).

Exception

The exception component allows you to interrupt a mapping process when a specific condition is met. Please see Adding Exceptions for more information.

13.4 Project

MapForce supports the Multiple Document Interface and allows you to group your mappings into mapping projects (see Working with Mapping Projects).

Reload Project

Reloads the currently active project and switches to the Project tab.

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 through the Open dialog box.

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 structure, and only becomes active when this is possible. See Managing Project Folders.

Generate code for entire project

Generates project code for the entire project currently visible in the Project window. Code is generated in the currently selected default language for all of the mapping files *.mfd in each of the folders.

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-wide settings. See Setting the Code Generation Settings.

Recent projects - 1. 2. etc.

Displays a list of the most recently opened projects.

13.5 Component

Change Root Element

Allows you to change the root element of the XML instance document.

Edit Schema Definition in XMLSpy

Selecting this option, having previously clicked an XML-Schema/document, opens the XML Schema file in the Schema view of XMLSpy where you can edit it.

Add/Remove/Edit Database Objects

Allows you to add, remove, or change the database objects within the database component.

Refresh

Reloads the structure of the currently active database component from the database.

Add Duplicate Input Before

Inserts a copy/clone of the selected item before the currently selected item. Duplicate items do not have output icons, you cannot use them as data sources. For an example, see Map Multiple Sources to One Target section in the tutorial. Right clicking a duplicate item also allows you to reposition it using the menu items Move Up/Move Down, depending on where the item is.

Add Duplicate Input After

Inserts a copy/clone of the selected item after the currently selected item. Duplicate items do not have output icons, you cannot use them as data sources. For an example, see the Map Multiple Sources to One Target section in the tutorial. Right clicking a duplicate item also allows you to reposition it using the menu items Move Up/Move Down, depending on where the item is.

Remove Duplicate

Removes a previously defined duplicate item. For an example, see the Map Multiple Sources to One Target section in the tutorial.

Database Table Actions

Allows you to define the actions to be performed with the mapped data on the specific target database table. See Database Table Actions Settings for more information.

Query Database

Creates a Select statement based on the table/field you clicked in the database component. Clicking a table/field once makes this command active, and the select statement is automatically placed into the Select window.

Align Tree Left

Aligns all the items along the left hand window border.

Align Tree Right

Aligns all the items along the right hand window border. This display is useful when creating mappings to the target schema.

Properties

Opens a dialog box which displays the settings of the currently selected component. See Changing the Component Settings .

13.6 Connection

Auto Connect Matching Children

Activates or deactivates the "Auto Connect Matching Children" option, as well as the icon in the icon bar.

Settings for Connect Matching Children

Opens the Connect Matching Children dialog box in which you define the connection settings (see Connecting matching children).

Connect Matching Children

This command allows you to create multiple connectors for items of the **same name,** in both the source and target schemas. The settings you define in this dialog box are retained, and are applied when connecting two items, if the "**Auto connect child items**" icon in the title bar is active. Clicking the icon switches between an active and inactive state. For further information, see Connecting matching children.

Target Driven (Standard)

Changes the connector type to Standard mapping. For further information, see Target Driven (Standard) mapping.

Copy-all (Copy Child Items)

Creates connectors for all matching child items, where each of the child connectors are displayed as a subtree of the parent connector (see Copy-all connections).

Source Driven (Mixed Content)

Changes the connector type to *Source Driven (Mixed Content)*. For further information, see Source Driven (Mixed Content) mapping.

Properties

Opens a dialog box in which you can define the specific (mixed content) settings of the current connector. Unavailable options are greyed out. These settings also apply to **complexType** items which do not have any text nodes. For further information, see Connection settings.

13.7 Function

Create User-Defined Function

Creates a new user-defined function (see User-defined functions).

Create User-Defined Function from Selection

Creates a new user-defined function based on the currently selected elements in the mapping window.

Function Settings

Opens the settings dialog box of the currently active user-defined function allowing you to change its settings.

Remove Function

Deletes the currently active user-defined function if you are working in a context which allows this.

Insert Input

When the mapping window displays a mapping, this command adds an input component to the mapping (see Simple Input). When the mapping window displays a user-defined function, this command adds an input component to the user-defined function (see Defining Complex Input Components).

Insert Output

When the mapping window displays a mapping, this command adds an output component to the mapping (see Simple Output). When the mapping window displays a user-defined function, this command adds an output component to the user-defined function (see Defining Complex Output Components).

13.8 Output

XSLT 1.0, XSLT 2.0, XQuery, Java, C#, C++, Built-in Execution Engine

Sets the transformation language in which the mapping should be executed (see Selecting a Transformation Language).

Validate Output File

Validates the output XML file against the referenced schema (see Validating the Mapping Output).

Save Output File

Saves the data visible in the Output pane to a file.

Save All Output Files

Saves all the generated output files of dynamic mappings. See Processing Multiple Input or Output Files Dynamically for more information.

Regenerate Output

Regenerates the data visible in the Output pane.

Run SQL-Script

If an SQL script is currently visible in the Output pane, the script executes the mapping to the target database, taking the defined table actions into account.

Insert/Remove Bookmark

Inserts a bookmark at the cursor position in the Output pane.

Next Bookmark

Navigates to the next bookmark in the Output pane.

Previous Bookmark

Navigates to the 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 to give a structured display of the document. Each child node is offset from its parent by a single tab character. This is where the Tab size settings (i.e. inserting as tabs or spaces) defined in the Tabs group, take effect.

Text View Settings

Displays the Text View settings dialog box. This dialog box allows you to customize the text view settings the **Output** pane, **XSLT** pane, and **XQuery** pane, and also shows the currently defined hotkeys that apply in the window. For more information, see Text View Features.

13.9 Debug

Start Debugging (F11)

Starts or continues debugging until a breakpoint is hit or the mapping finishes.

Stop Debugging (Shift + F5)

Stops debugging. This command exits the debug mode and switches MapForce back to standard mode.

Step into (F11)

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 translates into "go to the left/go to target child/go to source parent".

Step over (F10)

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.

Step out (Shift + F11)

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 translates into "go to the right/go to target parent/go to source child".

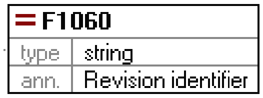
Minimal step (Ctrl + F11)

Continues execution until a value is produced or consumed. This command subdivides a step and will typically stop twice for each connection: once when its source produces a value and 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.

13.10 View

Show Annotations

Displays XML schema annotations in the component window.
If the Show Types icon is also active, then both sets of info are show in grid form.



F1060

type	string
ann.	Revision identifier

Show Types

Displays the schema datatypes for each element or attribute.
If the Show Annotations icon is also active, then both sets of info are show in grid form.

Show library in Function Header

Displays the library name in parenthesis in the function title.

Show Tips

Displays a tooltip containing explanatory text when the mouse pointer is placed over a function.

Show Selected Component Connectors

Switches between showing all mapping connectors, or those connectors relating to the currently selected components.

Show Connectors from Source to Target

Switches between showing:

- connectors that are **directly** connected to the currently selected component, or
- connectors linked to the currently selected component, originating from source and terminating at the target components.

Zoom

Opens the Zoom dialog box. You can enter the zoom factor numerically, or drag the slider to change the zoom factor interactively.

Back

Steps back through the currently open mappings of the mapping tab.

Forward

Steps forward through the currently open mappings of the mapping tab.

Status Bar

Switches on/off the Status Bar visible below the Messages window.

Library Window

Switches on/off the Library window.

Messages

Switches on/off the Validation output window. When generating code the Messages output window is automatically activated to show the validation result.

Overview

Switches on/off the Overview window. Drag the rectangle to navigate your Mapping view.

Project window

Switches on/off the Project window.

13.11 Tools

Global Resources

Opens the Manage Global Resources dialog box, where you can add, edit or delete settings applicable across multiple Altova applications (see Altova Global Resources).

Active Configuration

Allows you to select the currently active global resource configuration from a list of configurations previously defined in the Global Resources.

Create Reversed Mapping

Creates a "reversed" mapping from the currently active mapping in MapForce, which is to be the basis of a new mapping. Note that the result is not intended to be a complete mapping, only the direct connections between components are retained in the reversed mapping. It is very likely that the resulting mapping will not be valid or suitable for preview in the Output pane, without manual editing.

When you reverse a mapping, the source component becomes the target component, and target component becomes the source. If an input or output XML instance file have been assigned to a component, then they will be swapped.

The following data is retained:

- Direct connections between components
- Direct connections between components in a chained mapping
- The type of connection: Standard, Mixed content, Copy-All
- Pass-through component settings
- Database components

The following data is not retained:

- Connections via functions, filters, etc, along with the functions, filters, etc.
- User-defined functions
- Web service components

Restore Toolbars and Windows

Resets the toolbars, entry helper windows, docked windows etc. to their defaults. MapForce needs to be restarted for the changes to take effect.

Customize...

Opens a dialog box that lets you to customize the MapForce graphical user interface. This includes showing or hiding toolbars, as well as editing the context menus and keyboard shortcuts (see Customizing Keyboard Shortcuts).

Options

Opens a dialog box where you can change the default MapForce settings (see Changing the MapForce Options).

13.12 Window

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.

1
2

This list shows all currently open windows, and lets you quickly switch between them.
You can also use the Ctrl-TAB or CTRL F6 keyboard shortcuts to cycle through the open windows.

13.13 Help Menu

▼ Table of Contents

⊟ *Description*

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

⊟ *Description*

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

⊟ *Description*

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 **Return**. 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

⊟ *Description*

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 key.*** 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 key-code. Enter your name, company, and e-mail address in the dialog that appears, and click Request Now! The evaluation key is sent to the e-mail address you entered and should reach you in a few minutes. Now enter the key in the key-code field of the Software Activation dialog box and click **OK** to start working with your Altova product. The software will be unlocked for a period of 30 days.
- ***Permanent license key.*** The Software Activation dialog contains a button 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. There are two types of permanent license: single-user and multi-user. Both will be sent to you by e-mail. A *single-user license* contains your license-data and

includes your name, company, e-mail, and key-code. A *multi-user license* contains your license-data and includes your company name and key-code. Note that your license agreement does not allow you to install more than the licensed number of copies of your Altova software on the computers in your organization (per-seat license). Please make sure that you enter the data required in the registration dialog exactly as given in your **license e-mail**.

Note: When you enter your license information in the Software Activation dialog, ensure that you enter the data exactly as given in your license e-mail. For multi-user licenses, each user should enter his or her own name in the Name field.

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 details (name, company, email, key-code)
- As an attachment, a license file with a **`.altova_licenses`** file extension

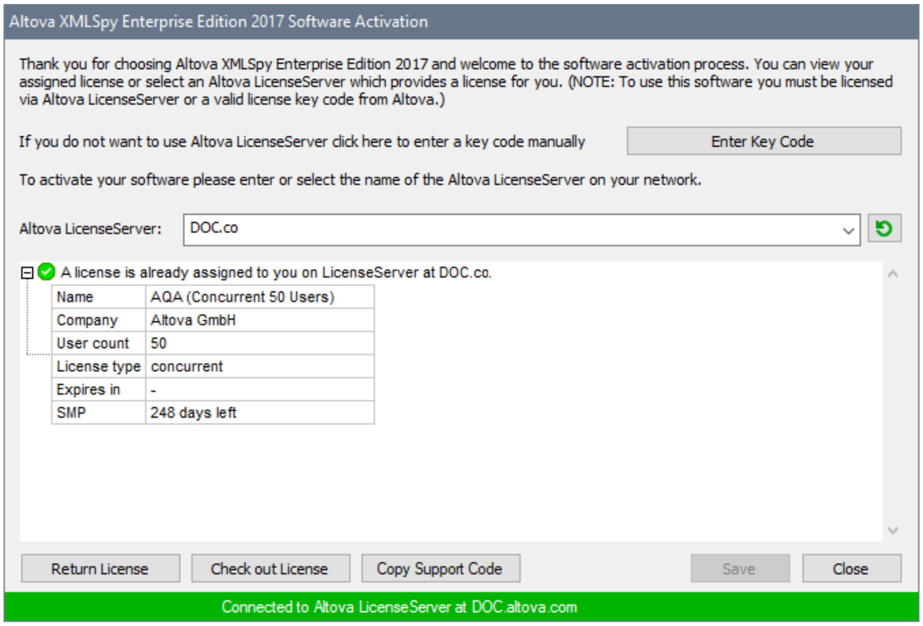
To activate your Altova product, you can do one of the following:

- Enter the email-supplied license details in the Altova product's Software Activation dialog, and click **OK**.
- 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 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, or (ii) assign the license to the product from Altova LicenseServer. *For more information about licensing via LicenseServer, read the rest of this topic.*

The Software Activation dialog (*screenshot below*) can be accessed at any time by clicking the **Help | Software Activation** command.

You can activate the software by either:

- Entering the license key information (click **Enter a New Key Code**), or
- Acquiring a license via an Altova LicenseServer on your network (click **Use Altova LicenseServer**, located at the bottom of the Software Activation dialog). 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 (*screenshot below*), and you can click **Save** to acquire the license.



Thank you for choosing Altova XMLSpy Enterprise Edition 2017 and welcome to the software activation process. You can view your assigned license or select an Altova LicenseServer which provides a license for you. (NOTE: To use this software you must be licensed via Altova LicenseServer or a valid license key code from Altova.)

If you do not want to use Altova LicenseServer click here to enter a key code manually

To activate your software please enter or select the name of the Altova LicenseServer on your network.

Altova LicenseServer: DOC.co

✓ A license is already assigned to you on LicenseServer at DOC.co.

Name	AQA (Concurrent 50 Users)
Company	Altova GmbH
User count	50
License type	concurrent
Expires in	-
SMP	248 days left

Connected to Altova LicenseServer at DOC.altova.com

After a machine-specific (aka installed) license has been acquired from a LicenseServer, it cannot be returned to the 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. The Software Activation dialog will display the check-out information, including the time when the check-out period ends. 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, make sure that the check-out period you select adequately covers the period during which you will be working offline.

Note: For license check-outs to be possible, it must be enabled on the LicenseServer. If this functionality has not been enabled, you will get an error message to this effect. 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

⊟ *Description*

When you are ready to order a licensed version of the software product, you can use either the **Order license key** button in the Software Activation dialog (*see previous section*) or the **Help | Order Form** command to proceed to the secure Altova Online Shop.

▼ Registration

⊟ *Description*

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

⊟ *Description*

Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly.

▼ Support Center

⊟ *Description*

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

⊟ *Description*

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

⊟ *Description*

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

⊟ *Description*

A link to the Altova website on the Internet. You can learn more about MapForce and related technologies and products at the Altova website.

▼ MapForce Training

⊟ *Description*

A link to the Online Training page at the Altova website. Here you can select from online courses conducted by Altova's expert trainers.

▼ About MapForce

⊟ *Description*

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.

Chapter 14

Code Generator

14 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.

If your mapping uses XML schemas or DTDs, you can also optionally generate schema wrapper libraries (see Generating Code from XML Schemas or DTDs). The schema wrapper libraries enable you to work with XML data in an abstract way, without too much concern for the underlying XML Application Program Interface (API), such as MSXML, Apache Xerces, Microsoft System.Xml, or Java Application for XML Processing (JAXP).

14.1 Introduction to code generator

The primary goal of the generated code is to execute a MapForce mapping. In addition to this, you can optionally generate schema wrapper libraries for XML schemas used by the mapping, which enables you to read or write data to/from XML instances.

The generated code is expressed in C++, Java or C# programming languages.

Target Language	**C++**	**C#**	**Java**
Development environments	Microsoft Visual Studio 2008, 2010, 2013, 2015, 2017	Microsoft Visual Studio 2008, 2010, 2013, 2015, 2017	Java 1.7 or later 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

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.

Note: 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:

1. Select all projects in the Solution Explorer.
2. On the **Project** menu, click **Properties**.
3. Click **Configuration Properties | C/C++ | Language**.
4. In the list of configurations, select *All Configurations*.
5. Change **Treat wchar_t as Built-in Type** to **No (/Zc:wchar_t-)**

C#

The generated C# code uses the .NET XML classes (System.Xml) and 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 industry-standard 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)
- A complete mapping application. When compiled and run, the application performs the mapping transformation.

Code generator templates

Output code is completely customizable via a simple yet powerful template language (SPL, from Spy Programming Language) which gives full control in mapping XML Schema built-in data-types to the primitive datatypes of a particular programming language. SPL allows you to easily replace the underlying parsing and validating engine, customize code according to your company's writing conventions, or use different base libraries such as the Microsoft Foundation Classes (MFC) and the Standard Template Library (STL).

14.2 What's new ...

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

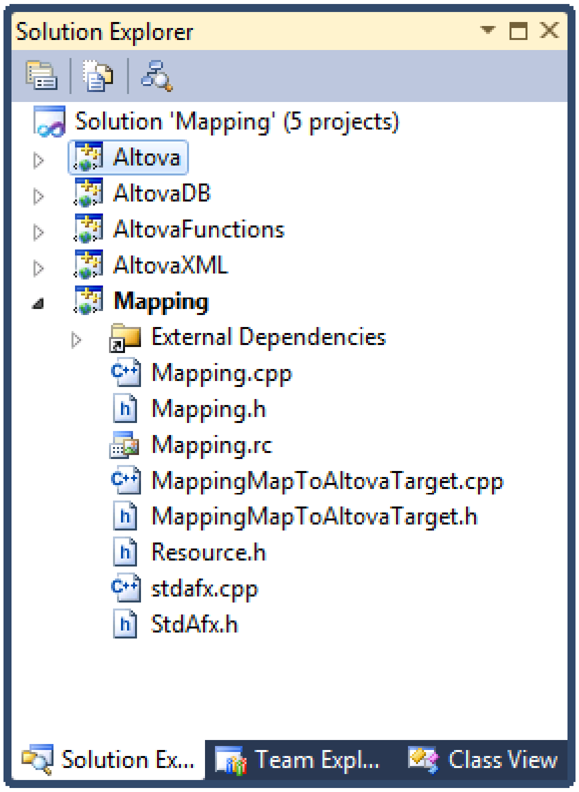
14.3 Generating C++ code

You can generate C++ code for Visual Studio 2008, 2010, 2013, 2015, 2017. The generated code includes **.sln** and **.vcproj** files for Visual Studio. Note the following when generating code:

- 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.
- 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**.
- 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**.

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
- Generating code from a mapping project
- Building the project
- Running the application

14.3.1 Generating code from a mapping

To generate C++ code from a mapping design file (.mfd):

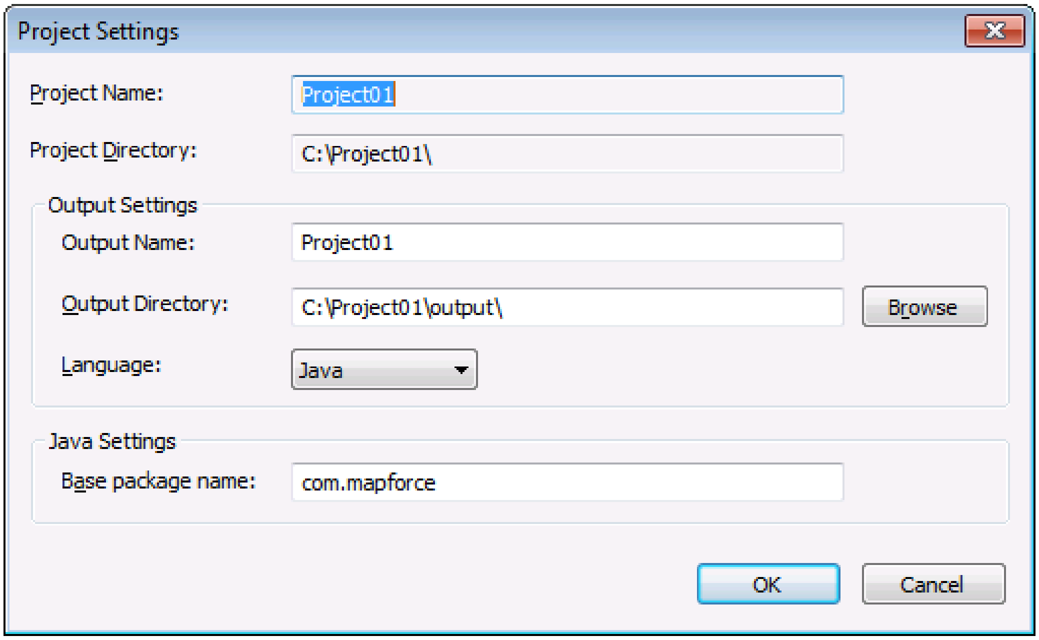
1. Review and select the desired code generation options (see Code generator options).
2. On the **File** menu, click **Generate code in | C++**.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed 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 dialog box (see Changing the mapping settings).

14.3.2 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**.



Project Settings

Project Name: Project01

Project Directory: C:\Project01\

Output Settings

Output Name: Project01

Output Directory: C:\Project01\output\

Language: Java

Java Settings

Base package name: com.mapforce

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.

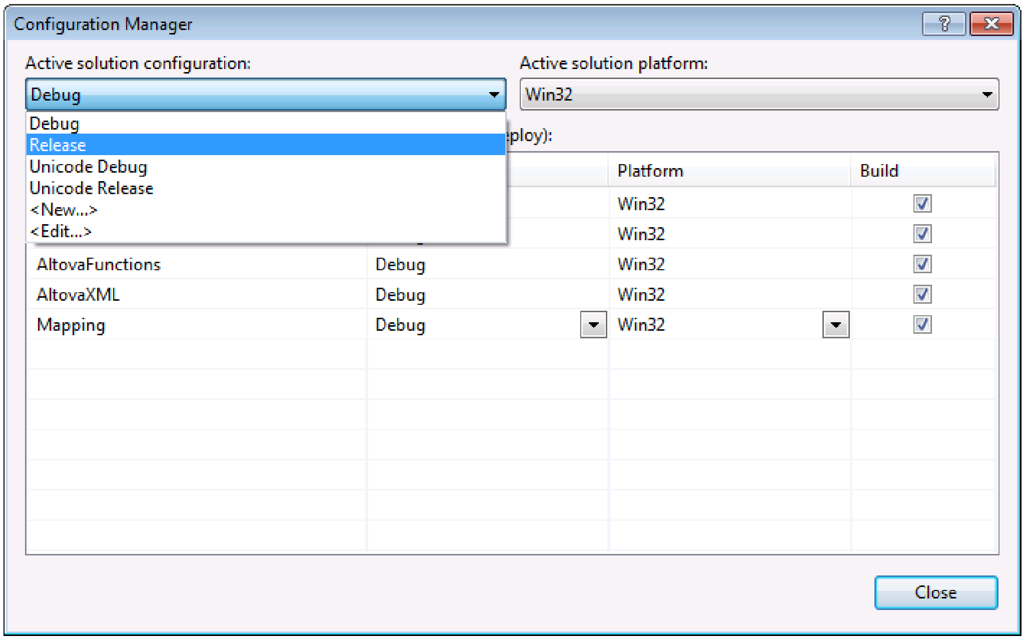
14.3.3 Building the project

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**.



Configuration Manager

Active solution configuration: Debug

Active solution platform: Win32

		Platform	Build
		Win32	☑
		Win32	☑
AltovaFunctions	Debug	Win32	☑
AltovaXML	Debug	Win32	☑
Mapping	Debug	Win32	☑

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**.

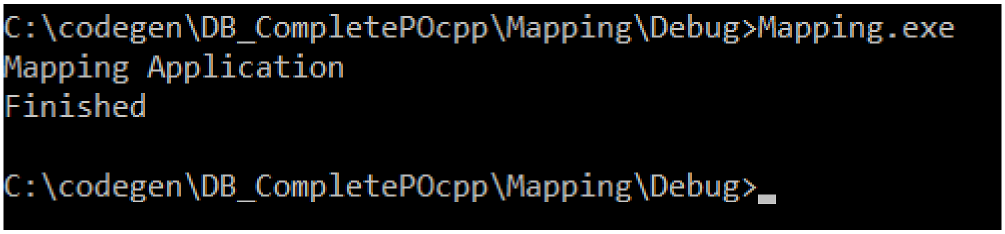
14.3.4 Running the application

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>
```

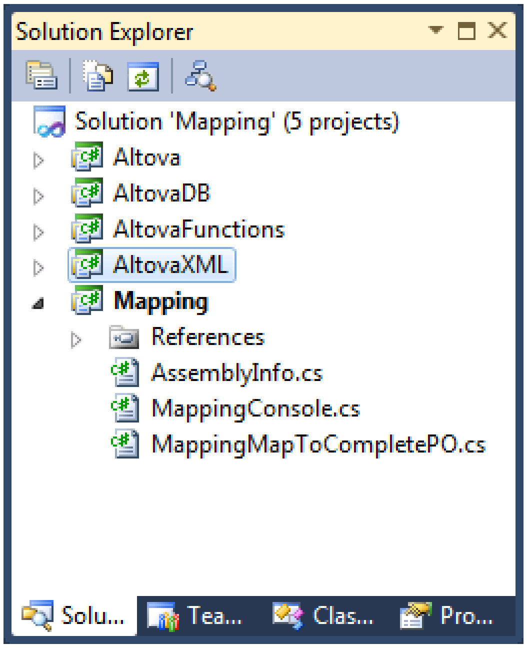
14.4 Generating C# code

You can generate C# code for Visual Studio 2008, 2010, 2013, 2015, 2017. The generated code includes **.sln** and **.csproj** files for Visual Studio. Mono users can use Visual Studio solution files with Mono's xbuild. Note the following when generating code:

- 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.
- 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**.
- 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**.

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

- Generating code from a mapping project
- Building the project
- Running the application

14.4.1 Generating code from a mapping

To generate C# code from a mapping design file (.mfd):

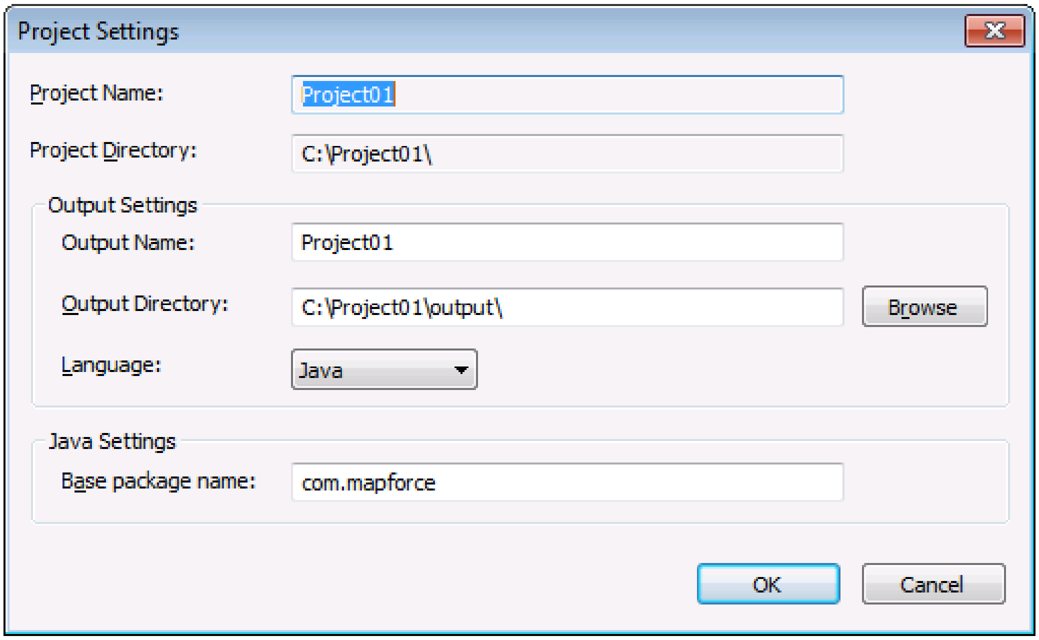
1. Review and select the desired code generation options (see Code generator options).
2. On the **File** menu, click **Generate code in | C#**.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed 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 dialog box (see Changing the mapping settings).

14.4.2 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**.


Project Settings

Project Name: Project01

Project Directory: C:\Project01\

Output Settings

Output Name: Project01

Output Directory: C:\Project01\output\

Language: Java

Java Settings

Base package name: com.mapforce

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.

14.4.3 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).
4. On the **Build** menu, click **Build Solution**.

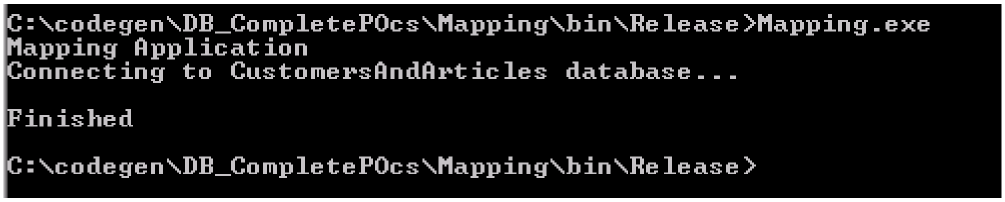
14.4.4 Running the application

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:

- 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>
```

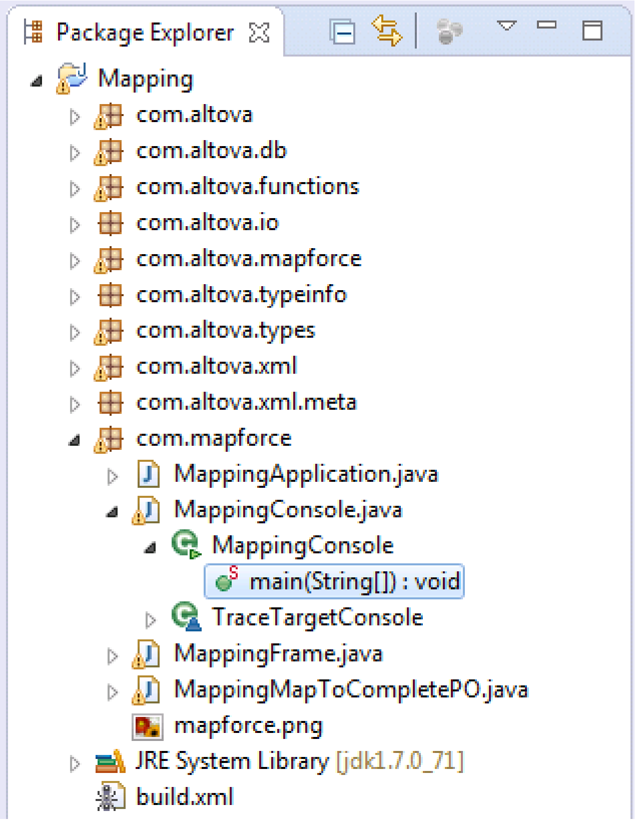
14.5 Generating Java code

You can generate program code for Java 1.7 or later. Note the following when generating code:

- 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.
- 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**.
- 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**.

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.



```
Mapping
  com.altova
  com.altova.db
  com.altova.functions
  com.altova.io
  com.altova.mapforce
  com.altova.typeinfo
  com.altova.types
  com.altova.xml
  com.altova.xml.meta
  com.mapforce
    MappingApplication.java
    MappingConsole.java
      MappingConsole
        main(String[]) : void
      TraceTargetConsole
    MappingFrame.java
    MappingMapToCompletePO.java
    mapforce.png
  JRE System Library [jdk1.7.0_71]
  build.xml
```

Sample MapForce-generated Java application (Eclipse IDE)

This section includes the following topics:

- Generating code from a mapping
- Generating code from a mapping project
- Handling JDBC references
- Building the project with Ant
- Example: Run and compile Java code with Eclipse and Ant

14.5.1 Generating code from a mapping

To generate Java code from a mapping design file (.mfd):

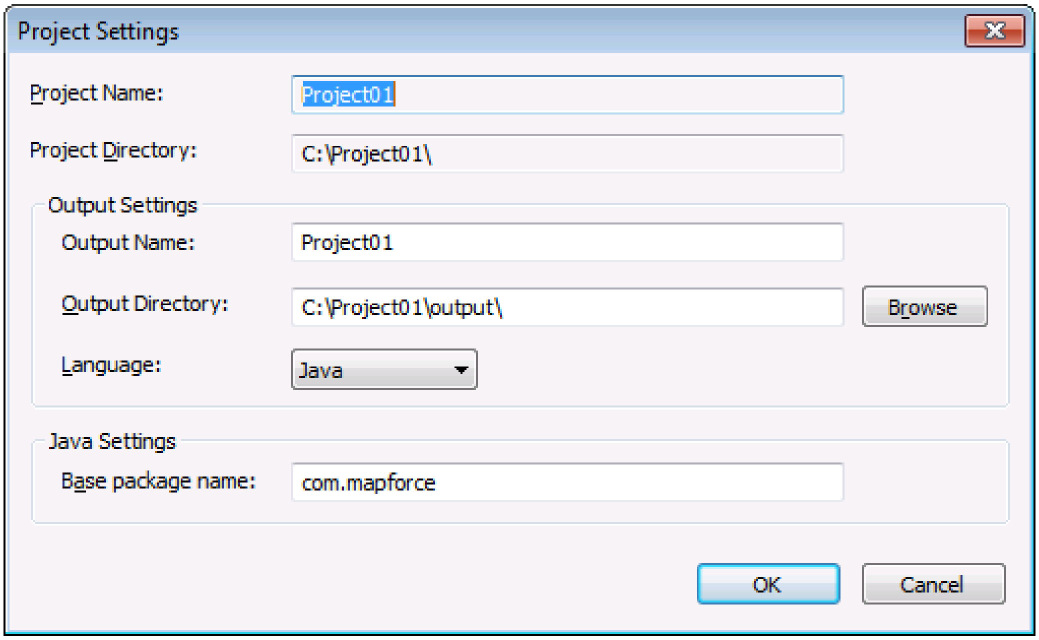
1. Review and select the desired code generation options (see Code generator options).
2. On the **File** menu, click **Generate code in | Java**.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed 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 dialog box (see Changing the mapping settings).

14.5.2 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**.



Project Name: Project01

Project Directory: C:\Project01\

Output Settings

Output Name: Project01

Output Directory: C:\Project01\output\

Language: Java

Java Settings

Base package name: com.mapforce

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.

14.5.3 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 (see Creating a JDBC connection). To view the current JDBC settings of any database component, click it, and then select **Component | Properties** from the menu.

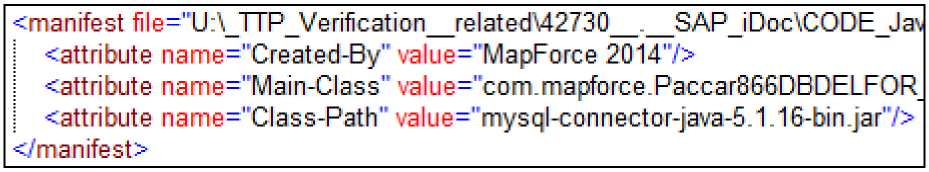
Additionally, if you build JAR files from generated Java code, add the "Class-Path" attribute for your database driver to the **build.xml** file. This ensures that the reference to the database driver is available in the manifest (**MANIFEST.MF**) file after you build the project.

To add the "Class-Path" attribute:

1. Add to the **build.xml** file a reference to the JAR file of the database driver, as a new "Class-Path" attribute. For example, for MySQL 5.1.16, the value of the "Class-Path" attribute looks 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 similar to the screen shot below.



```
<manifest file="U:\_TTP_Verification__related\42730__.__SAP_iDoc\CODE_Jav
    <attribute name="Created-By" value="MapForce 2014"/>
    <attribute name="Main-Class" value="com.mapforce.Paccar866DBDELFOR_
    <attribute name="Class-Path" value="mysql-connector-java-5.1.16-bin.jar"/>
</manifest>
```

2. Copy the JAR file of the JDBC driver to the folder that contains the JAR file of the generated application.

14.5.4 Building the project with Ant

Apache Ant is a widely used Java library (and command-line 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.

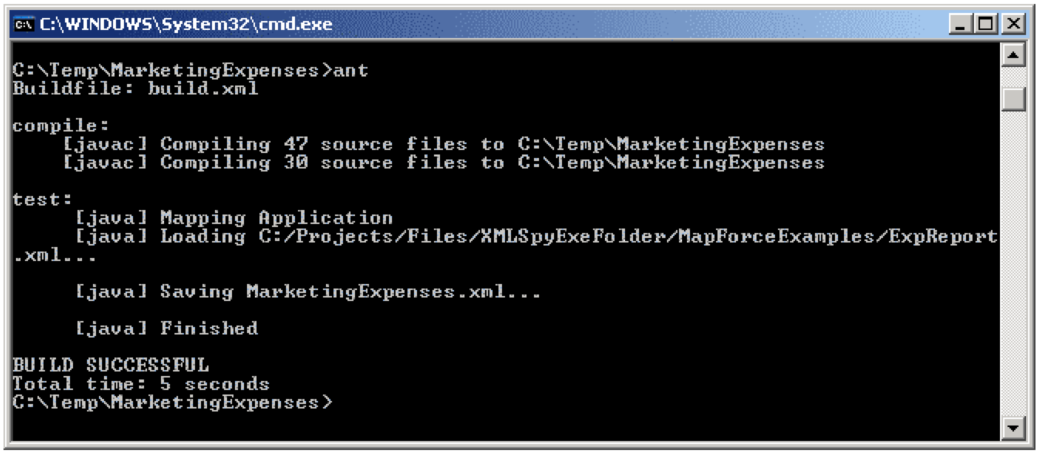
Ant may be available on your system either as a standalone installation, or bundled within Eclipse (or other Java IDEs). For instructions on how to install Ant on your system, see http://ant.apache.org/manual/. For instructions on how to use Ant in Eclipse, refer to the Eclipse tutorial, as well as the 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 options defined in the **build.xml** file.



```
C:\Temp\MarketingExpenses>ant
Buildfile: build.xml

compile:
    [javac] Compiling 47 source files to C:\Temp\MarketingExpenses
    [javac] Compiling 30 source files to C:\Temp\MarketingExpenses

test:
     [java] Mapping Application
     [java] Loading C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport
.xml...

     [java] Saving MarketingExpenses.xml...

     [java] Finished

BUILD SUCCESSFUL
Total time: 5 seconds
C:\Temp\MarketingExpenses>
```

To generate a JAR file with Ant:

- At the command prompt, enter `ant jar`.

If you need help with Ant command syntax and options, enter `ant -help` at the command line.

14.5.5 Example: Build a Java application with Eclipse and Ant

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:

- The Java Development Kit (JDK), Eclipse (https://www.eclipse.org/), and Ant (http://ant.apache.org/) must be installed on your system. Eclipse typically includes support for building projects with Ant (see also Building the project with Ant).
- The Windows PATH environment variable must include the path to the JDK's **bin** directory (for example, **C:\Java\jdk1.7.0\bin**). This is a basic requirement for developing applications for the Java platform. For instructions, see http://docs.oracle.com/javase/tutorial/essential/environment/paths.html.

This example uses the following configuration:

- JDK 1.7
- Eclipse IDE for Java Developers (Luna Service Release 4.4.1)
- Ant 1.9.2, which is already integrated into the above-mentioned edition of Eclipse; therefore, it was not installed and configured separately.

The example is organized into the following sub-tasks:

- Step 1: Generate Java Code
- Step 2: Import the Project into Eclipse

- Step 3: Run the Project as GUI Application
- Step 4: Run the Project as Console Application
- Step 5: Build the JAR file with Ant

14.5.5.1 Step 1: Generate Java code

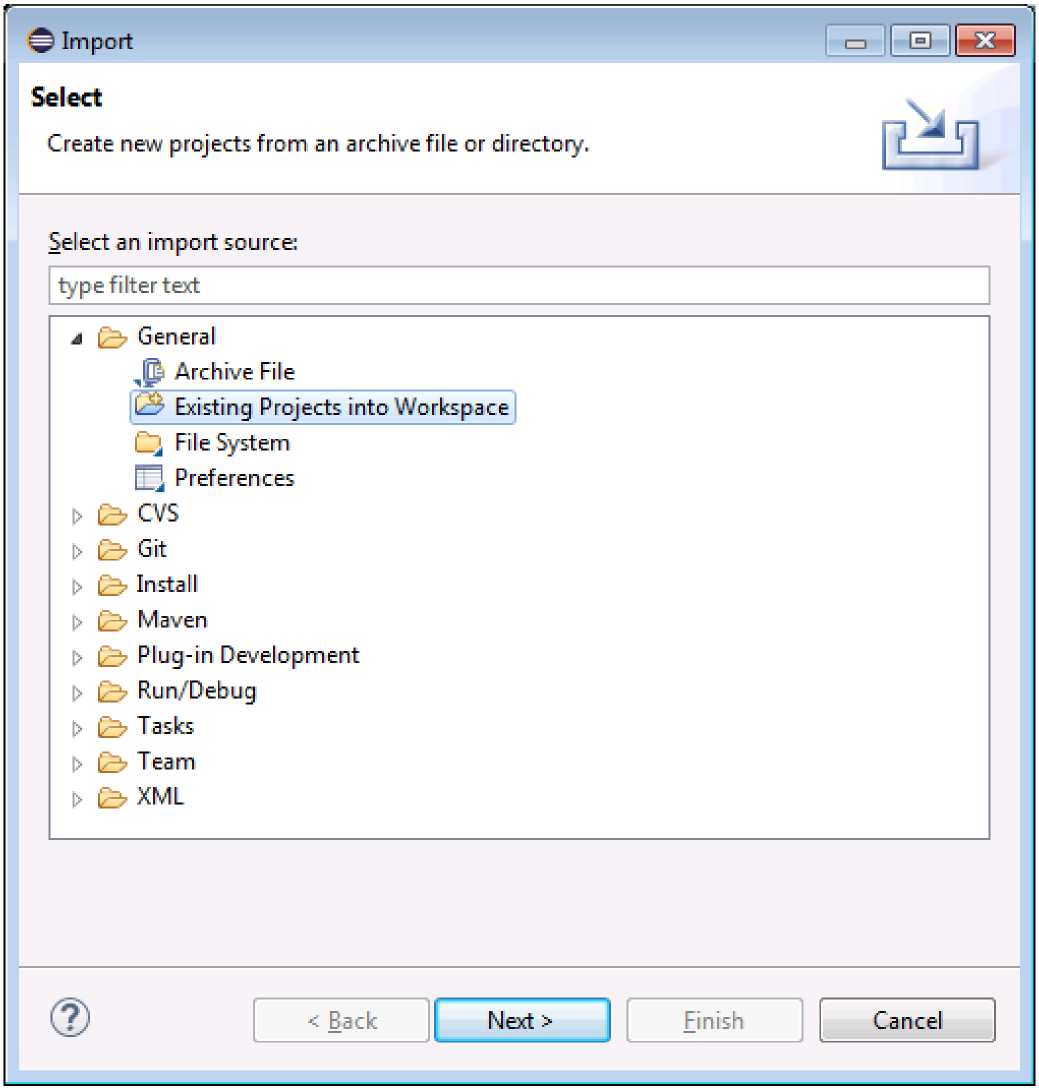
To generate the Java code in MapForce:

1. On the **File** menu, click **Open**, and browse for the **CompletePO.mfd** mapping available in the <Documents>\Altova\MapForce2018\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 to which the Java project must be saved. For convenience, you may choose to save the project to **C:\workspace\CompletePO\** (where **C:\workspace** is your default Eclipse workspace).

14.5.5.2 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 **Existing Projects into Workspace**.



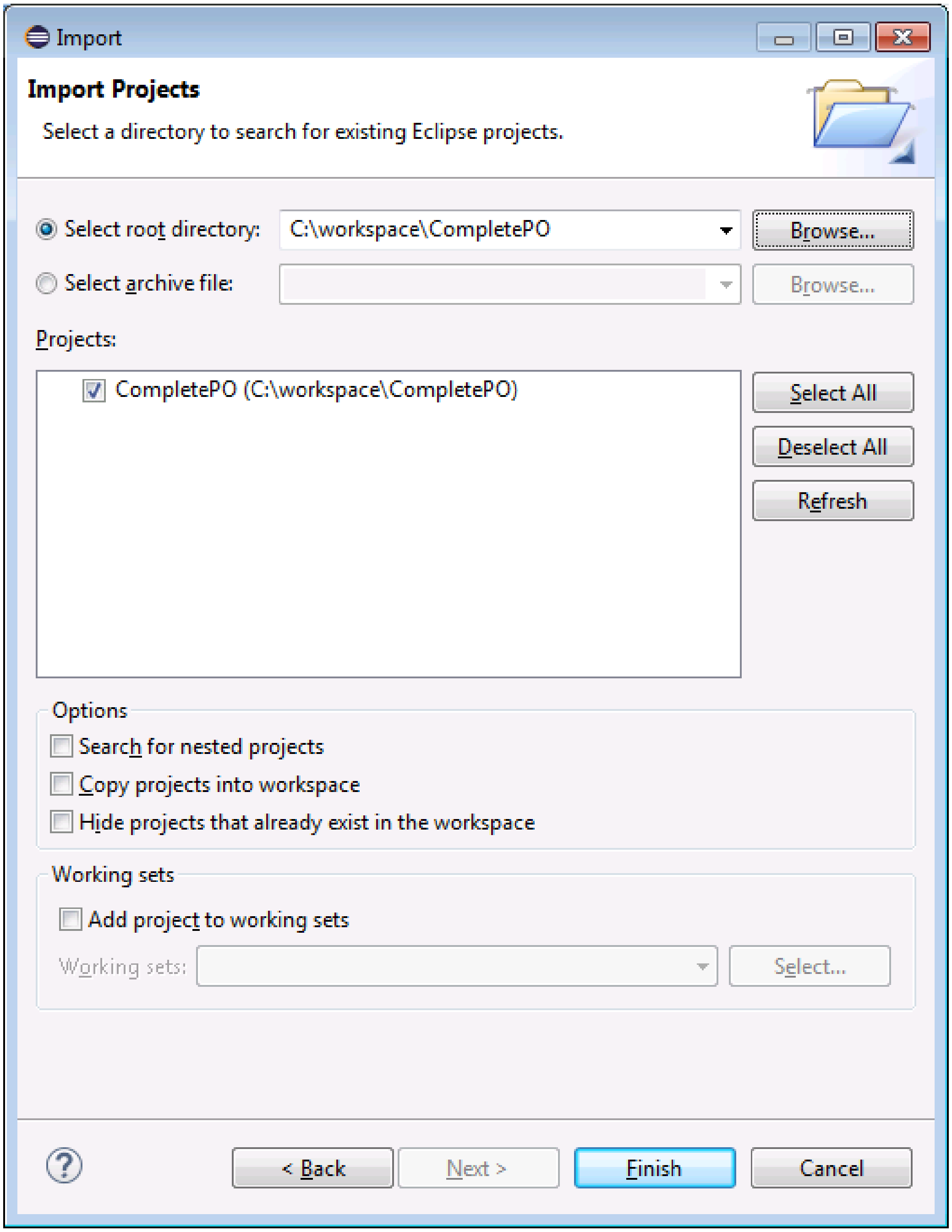
Select

Create new projects from an archive file or directory.

Select an import source:

type filter text

3. Click **Next**.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: C:\workspace\CompletePO

○ Select archive file:

Projects:

☑ CompletePO (C:\workspace\CompletePO)

Options

□ Search for nested projects

□ Copy projects into workspace

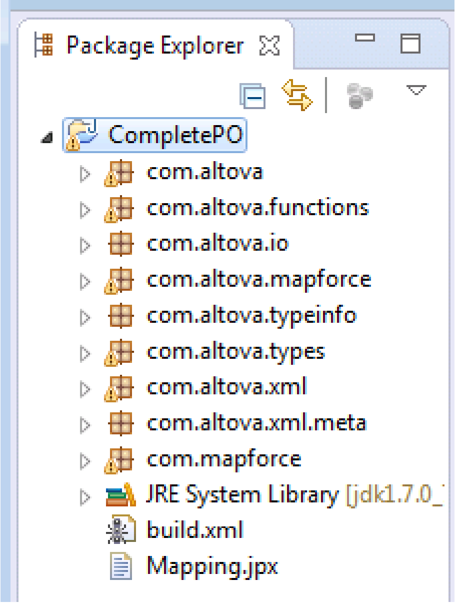
□ Hide projects that already exist in the workspace

Working sets

□ Add project to working sets

Working sets:

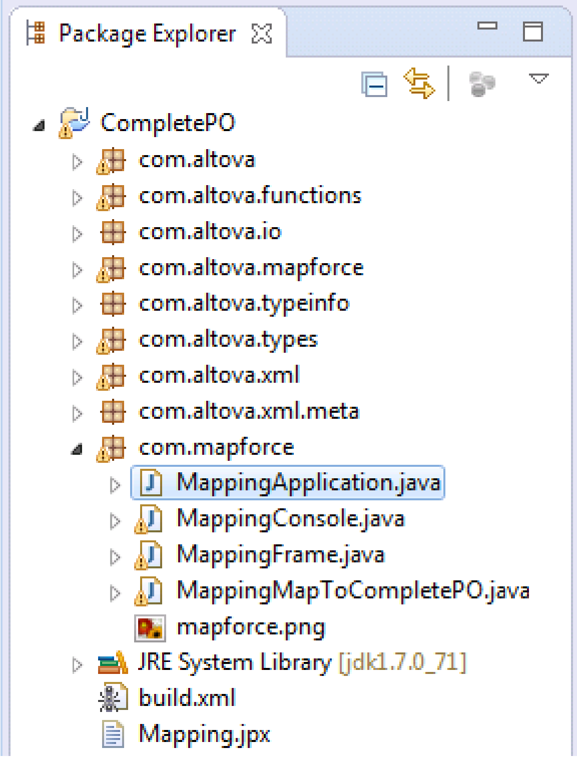
4. Browse for the folder where you have previously saved the generated code, and then click **Finish**. The Java project created by MapForce is now available in the Package Explorer view. If you cannot see the Package Explorer view, display it using the menu command **Window | Show View | Package Explorer**.



14.5.5.3 *Step 3: Run the project as dialog application*

To run the Java project as a GUI application:

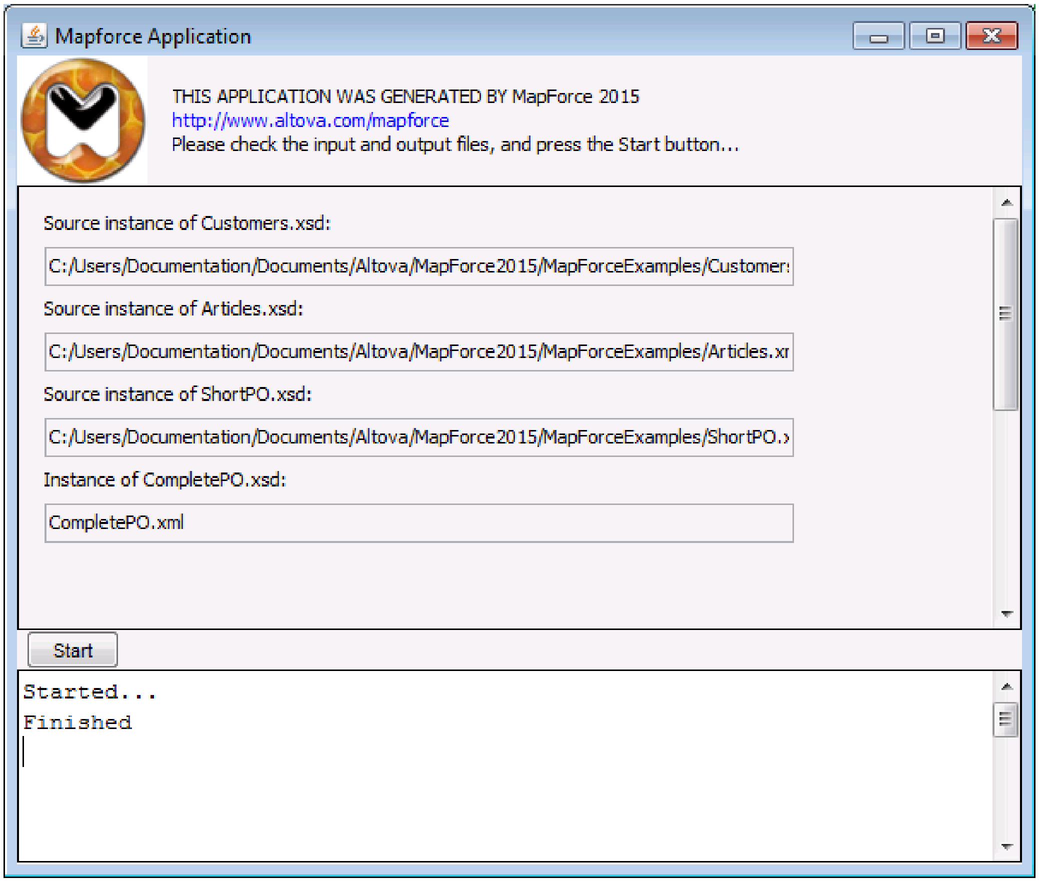
1. In the Package Explorer view of Eclipse, click the **MappingApplication.java** file available in the **com.mapforce** package.



Package Explorer

CompletePO
com.altova
com.altova.functions
com.altova.io
com.altova.mapforce
com.altova.typeinfo
com.altova.types
com.altova.xml
com.altova.xml.meta
com.mapforce
MappingApplication.java
MappingConsole.java
MappingFrame.java
MappingMapToCompletePO.java
mapforce.png
JRE System Library [jdk1.7.0_71]
build.xml
Mapping.jpx

2. On the **Run** menu, click **Run As | Java application**.
3. On the MapForce application window, click **Start** to execute the mapping.



THIS APPLICATION WAS GENERATED BY MapForce 2015
http://www.altova.com/mapforce
Please check the input and output files, and press the Start button...

Source instance of Customers.xsd:
C:/Users/Documentation/Documents/Altova/MapForce2015/MapForceExamples/Customer:

Source instance of Articles.xsd:
C:/Users/Documentation/Documents/Altova/MapForce2015/MapForceExamples/Articles.xr

Source instance of ShortPO.xsd:
C:/Users/Documentation/Documents/Altova/MapForce2015/MapForceExamples/ShortPO.›

Instance of CompletePO.xsd:
CompletePO.xml

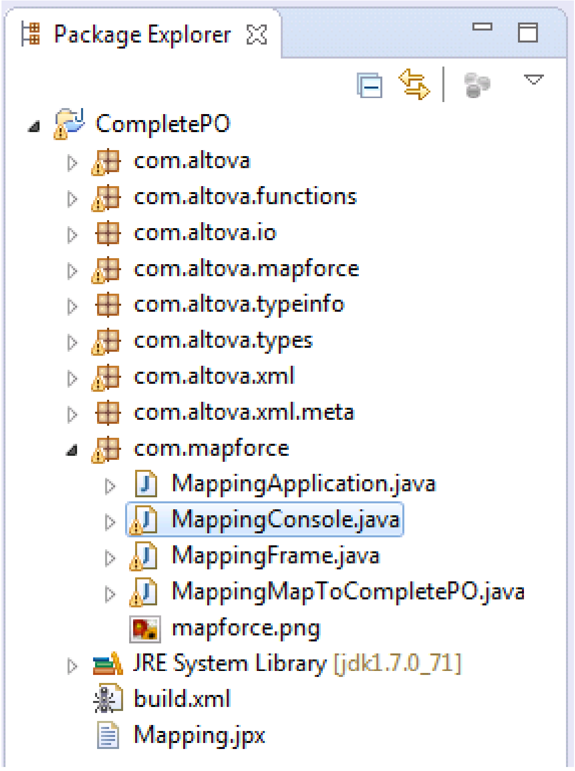
```
Started...
Finished
```

If Eclipse encounters system configuration or run-time errors, you will be prompted. Otherwise, the Java application executes the mapping transformation and generates the **CompletePO.xml** at the output path (in this example: **C:\workspace\CompletePO\CompletePO.xml**).

14.5.5.4 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.



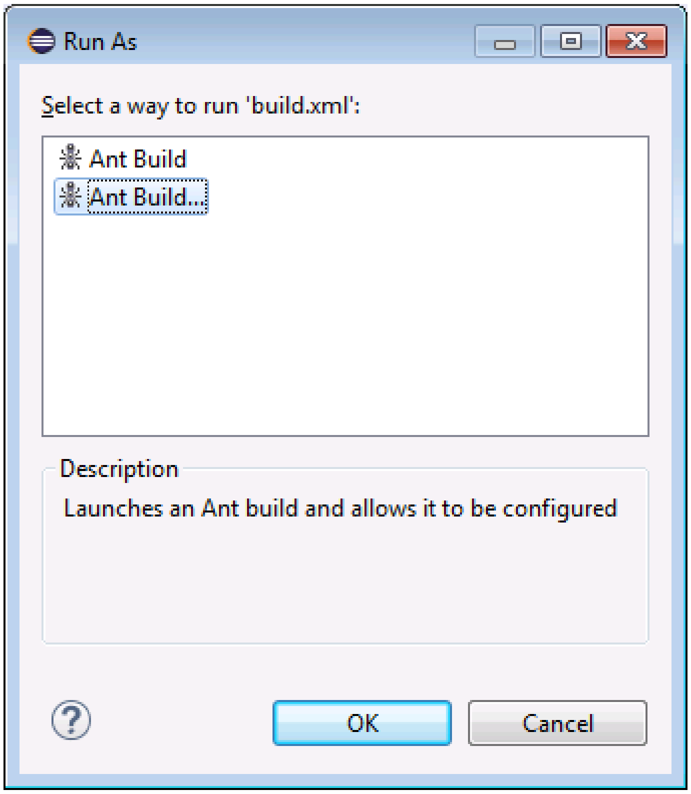
2. On the **Run** menu, click **Run As | Java application**.

If Eclipse detects system configuration or run-time errors, you will be prompted. Otherwise, the Java application executes the mapping transformation and generates the **CompletePO.xml** at the output path (in this example: **C:\workspace\CompletePO\CompletePO.xml**).

14.5.5.5 Step 5: Build the JAR file with Ant

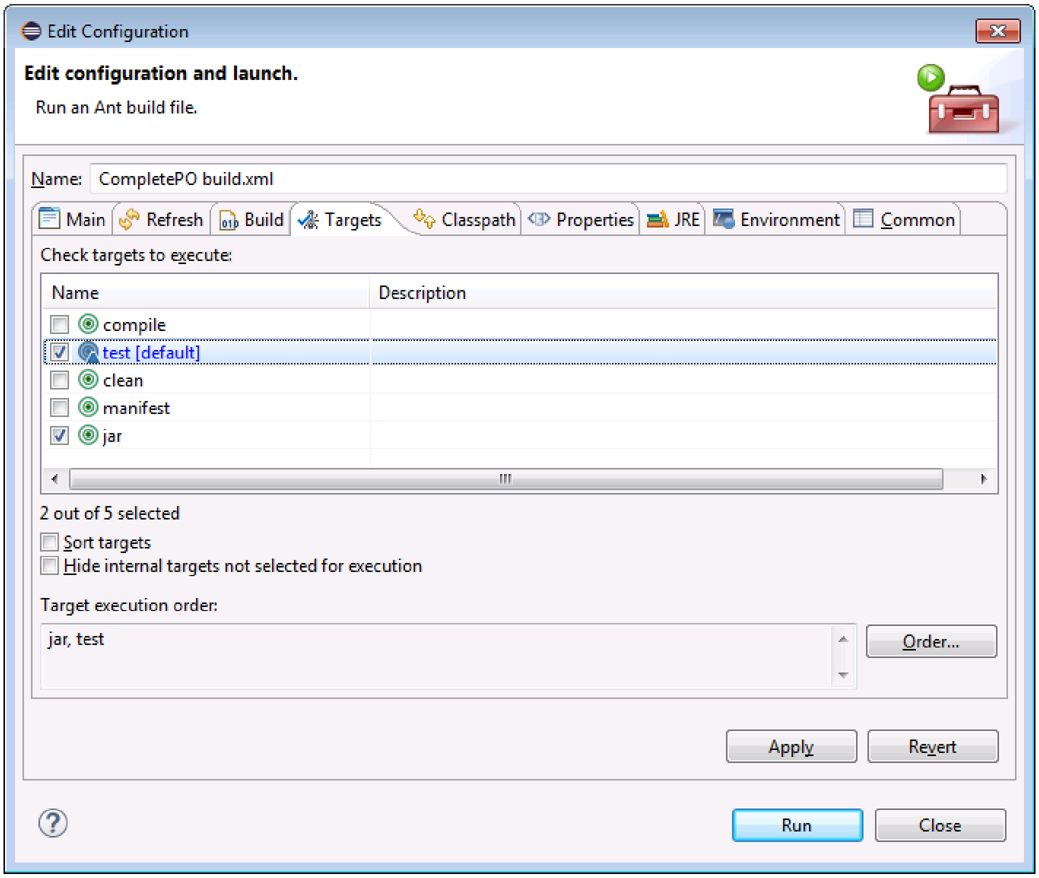
To build the .jar file with Ant:

1. In the Package Explorer view of Eclipse, click the **build.xml** file available directly in the project root.
2. On the Run menu, click Run.



Run As
Select a way to run 'build.xml':
Ant Build
Ant Build...
Description
Launches an Ant build and allows it to be configured
OK
Cancel

3. In the Run As dialog box, 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.
4. Click to enable the targets that you wish to include in the Ant build. In this example, the targets **test** and **jar** are selected.



Edit Configuration

Edit configuration and launch.

Run an Ant build file.

Name: CompletePO build.xml

Check targets to execute:

	Name	Description
□	compile	
☑	test [default]	
□	clean	
□	manifest	
☑	jar	

2 out of 5 selected

□ Sort targets

□ Hide internal targets not selected for execution

Target execution order:

jar, test

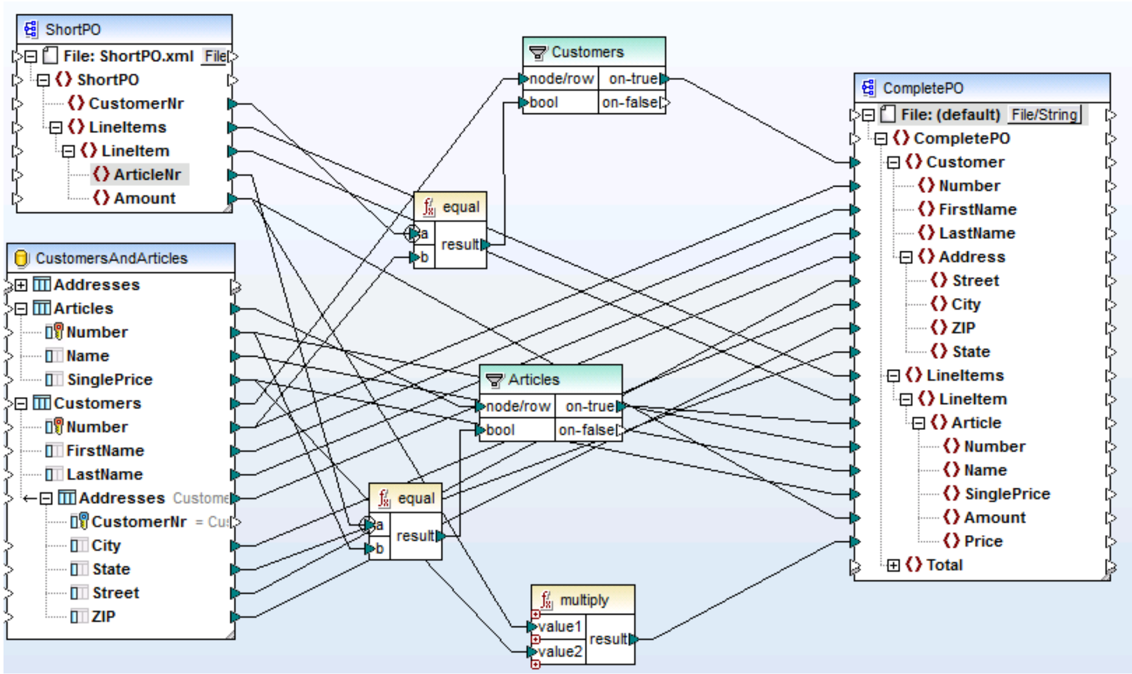
5. Click **Run**. Eclipse executes the Ant build file and displays the result in the Console view.

14.6 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\MapForce2018 \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:

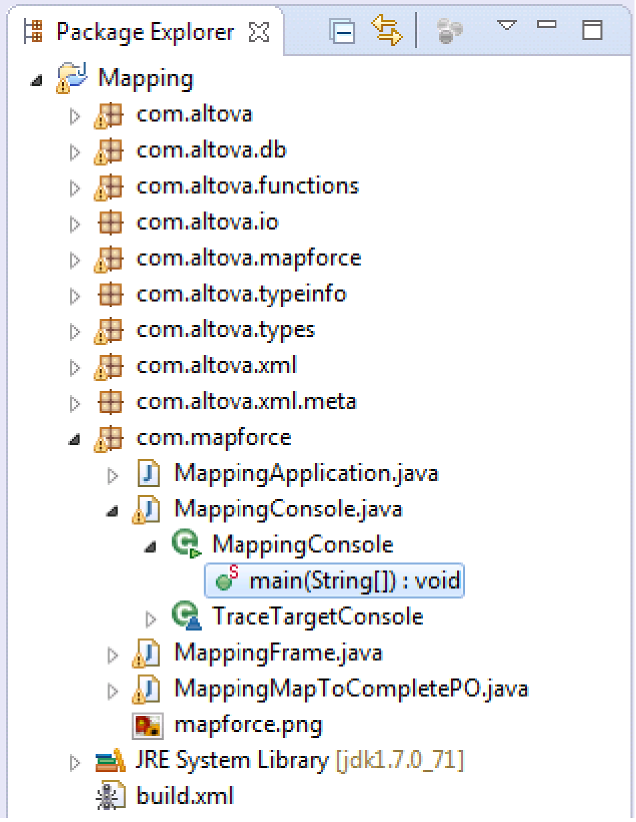
Java

```
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.

14.6.1 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\MapForce2018\MapForceExamples\ directory, and then import the project into Eclipse.


Package Explorer

```
Mapping
  com.altova
  com.altova.db
  com.altova.functions
  com.altova.io
  com.altova.mapforce
  com.altova.typeinfo
  com.altova.types
  com.altova.xml
  com.altova.xml.meta
  com.mapforce
    MappingApplication.java
    MappingConsole.java
      MappingConsole
        main(String[]) : void
      TraceTargetConsole
    MappingFrame.java
    MappingMapToCompletePO.java
    mapforce.png
  JRE System Library [jdk1.7.0_71]
  build.xml
```

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:

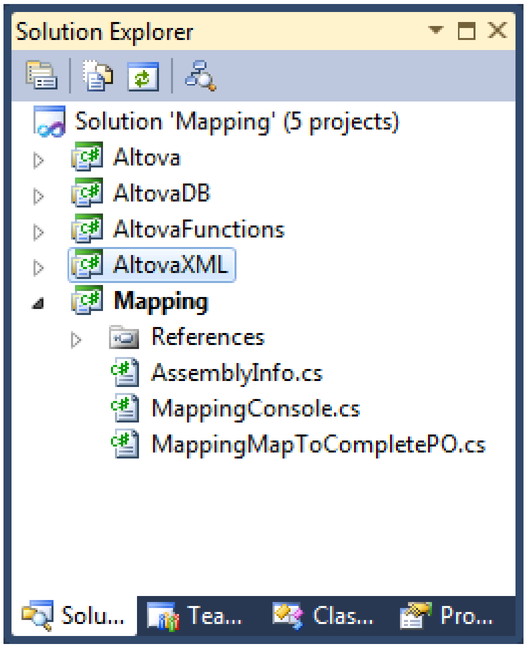
- Edit the code below the **`catch`** `(Exception e)` code (in case of a Java console application)
- Edit the code below the **`catch`** `(Exception ex) code` (in case of a Java dialog application)

For instructions on how to change the data type of parameters supplied as mapping input/output, see Changing the data type of the mapping input/output (C#, Java).

14.6.2 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\MapForce2018\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.



Solution Explorer

Solution 'Mapping' (5 projects)
- Altova
- AltovaDB
- AltovaFunctions
- AltovaXML
- Mapping
  - References
  - AssemblyInfo.cs
  - MappingConsole.cs
  - MappingMapToCompletePO.cs

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 highlighted in yellow and 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:

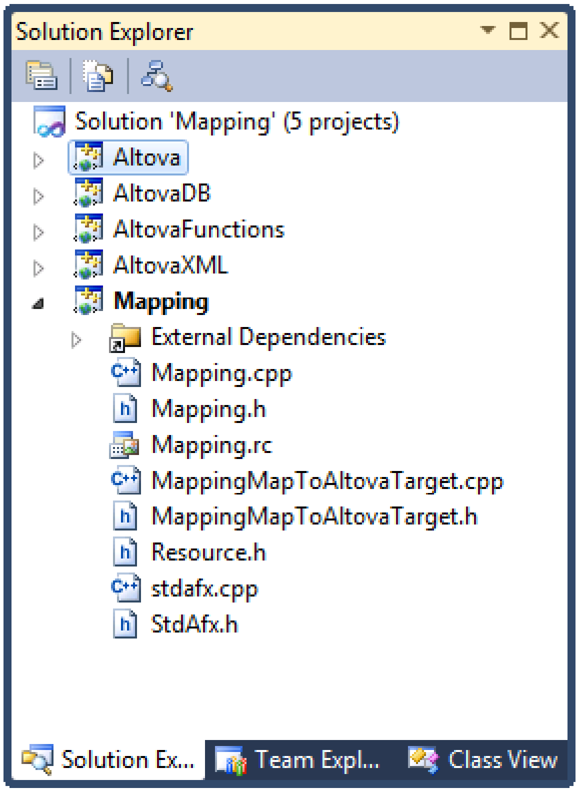
- Edit the code below the `catch (Exception e)` code

For instructions on how to change the data type of parameters supplied as mapping input/output, see Changing the data type of the mapping input/output (C#, Java).

14.6.3 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\MapForce2018\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.

14.6.4 Changing the data type of the mapping input/output (C#, Java)

This topic provides details on the object types you can use programmatically, if you intend to run MapForce mappings from a custom Java or C# application.

You can use several input and output objects (such as files, strings, DOM documents, and others) as parameters to the **run** method. The **run** method is the most important function 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. Components that process multiple files do not appear as parameters to the **run** method, because in this case the file names are processed dynamically inside the mapping.

The objects that you can provide as parameters to the **run** method are available in the **com.altova.io** package (Java) and Altova.IO namespace (C#). The base classes of the generated input and output objects are as follows:

C#

```
Altova.IO.Input
Altova.IO.Output
```

Java

```
com.altova.io.Input
com.altova.io.Output
```

The object types supported as input/output parameters to the **run** method, including their applicable input/output file formats, are listed in the following table.

Object Type	XML	Microsoft Excel*	EDI* **	FlexText*	CSV	Fixed-length files
Files	Y	Y	Y	Y	Y	Y
Binary stream objects	Y	Y	Y	Y	Y	Y
Strings	Y	–	Y	Y	Y	Y
I/O Reader/Writer (character stream objects)	Y	–	Y	Y	Y	Y
DOM documents	Y	–	–	–	–	–

***** Formats supported only in MapForce Enterprise Edition
****** Includes X12 and HL7

Files

File objects (identified in the code file names) have the following definition:

C#

```
Altova.IO.FileInput(string filename)
Altova.IO.FileOutput(string filename)
```

Java

```
com.altova.io.FileInput(String filename)
com.altova.io.FileOutput(String filename)
```

Binary stream objects

Binary stream objects in the generated code represent an alternative way to working with physical files; there are no advantages as far as memory use is concerned. Binary stream objects have the following definition:

C#

```
Altova.IO.StreamInput(System.IO.Stream stream)
Altova.IO.StreamOutput(System.IO.Stream stream)
```

Java

```
com.altova.io.StreamInput(java.io.InputStream stream)
com.altova.io.StreamOutput(java.io.OutputStream stream)
```

Notes:

- Binary stream 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 behaviour, insert the following code before calling the **run** method:

Java

```
MappingMapToSomething.setCloseObjectsAfterRun(false);  // Java
```

C#

```
MappingMapToSomething.CloseObjectsAfterRun = false;   // C#
```

Strings

String objects have the following definition:

C#

```
Altova.IO.StringInput(string content)
Altova.IO.StringOutput(StringBuilder content)
```

Java

```
com.altova.io.StringInput(String xmlcontent)
com.altova.io.StringOutput()
```

In Java, `StringOutput` does not take an argument. Content can be accessed with:

```
// mapping from String to (another) String
String MyText = "<here>is some XML text</here>";

Input input = new StringInput(MyText);
Output output = new StringOutput();

MappingMapToMyText.run(input, output);

String myTargetData = output.getString().toString();
```

The `getString()` method returns a `StringBuffer`, hence the need for `toString()`.

In C#, `StringOutput` takes an argument (`StringBuilder`) which you need to provide beforehand. The `StringBuilder` may already contain data, so the mapping output is appended to it.

Excel sources/targets cannot map to or from strings.

I/O Reader/Writer (character stream objects)

Character stream objects have the following definition:

C#

```
Altova.IO.ReaderInput(System.IO.TextReader reader)
Altova.IO.WriterOutput(System.IO.TextWriter writer)
```

Java

```
com.altova.io.ReaderInput(java.io.Reader reader)
com.altova.io.WriterOutput(java.io.Writer writer)
```

Notes:

- Character stream objects are expected to be opened and ready-to-use before calling the **run** method.
- Excel sources/targets cannot be read from, or written to, character streams.
- By default, the **run** method closes the stream when finished. To prevent this behaviour, insert the following code before calling the **run** method:

Java

```
MappingMapToSomething.setCloseObjectsAfterRun(false);  // Java
```

C#

```
MappingMapToSomething.CloseObjectsAfterRun = false;   // C#
```

DOM documents

DOM documents have the following definition:

C#

```
Altova.IO.DocumentInput(System.Xml.XmlDocument document)
Altova.IO.DocumentOutput(System.Xml.XmlDocument document)
```

Java

```
com.altova.io.DocumentInput(org.w3c.dom.Document document)
com.altova.io.DocumentOutput(org.w3c.dom.Document document)
```

Notes:

- The document passed to the **DocumentOutput** constructor as target must be empty.
- After calling **run**, the DOM Document generated by the constructor of **DocumentOutput** already contains mapped data so "save to document" is not necessary. After mapping, you can manipulate the document as necessary.
- Only XML content can be mapped to DOM documents.

Example

Let's assume you want to integrate the code generated by MapForce into your Java application. Your MapForce mapping consists of two source XML files and a target text file. When you generate the MapForce code, the **run** function looks as follows:

```
void run(Input in1, Input in2, Output out1);
```

Let's also assume that your application requires that you map data from a local file and binary stream into a character stream. Since data is supplied from other sources, your application must declare the sources and targets as:

```
String filename; // Declare the source of the first input
Java.io.InputStream stream; // Declare the source of the second input
Java.io.Writer writer; // Declare the output as character stream
```

The following wrappers must be constructed for the MapForce-generated **`run`** function:

```
// com.altova.io is considered imported here:
Input input1 = new FileInput(filename);
Input input2 = new StreamInput(stream);
Output output1 = new WriterOutput(writer);
```

Now you can call the MapForce generated run function:

```
MappingMapToSomething.run(input1, input2, output1);
```

The **C#** behavior is almost identical, except that **`run`** is called **`Run`**, and the .NET stream and reader/writer classes are named differently.

Using the same technique, you can also use other input and output types, such as strings or DOM documents.

14.7 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 in the Options dialog (see Code Generator Options). 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 identical for both DOM implementations. 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).

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 SPL (Spy Programming Language) 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 (Spy Programming Language).

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++)
- Reference to Generated Classes (C#)
- Reference to Generated Classes (Java)

For information about type conversion and other details applicable to each language, see:

- About Schema Wrapper Libraries (C++)
- About Schema Wrapper Libraries (C#)
- About Schema Wrapper Libraries (Java)

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 (Spy Programming Language), but is not enforced by generated classes.
- Default and fixed values for attributes. These are available in SPL (Spy Programming Language), 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`)

14.7.1 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`	
`xs:duration`	`altova::Duration`	
`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
altova::meta::ComplexType

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] .

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
[YourSchema]::MemberElement

Note: The actual class names depend on the name of the schema attribute or element.

See also Example: Using the Schema Wrapper Libraries.

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

n		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.
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
altova::meta::ComplexType
altova::meta::Element
altova::meta::SimpleType

14.7.2 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	

XML Schema	C#	Remarks
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	Altova.Types.DateTime	
xs:duration	Altova.Types.Duration	
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
Altova.Xml.Meta.ComplexType

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].

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
[YourSchemaType].MemberElement

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`
`Altova.Xml.Meta.ComplexType`
`Altova.Xml.Meta.Element`
`Altova.Xml.Meta.SimpleType`

The properties that return one of the metadata classes return null if the respective property does not exist.

14.7.3 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`	

XML Schema	Java	Remarks
`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`	
`xs:duration`	`com.altova.types.Duration`	
`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`
`com.altova.xml.meta.ComplexType`

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].

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
com.[YourSchema].[YourSchemaType].MemberElement

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.
`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
com.altova.xml.meta.ComplexType
com.altova.xml.meta.Element
com.altova.xml.meta.SimpleType

The properties that return one of the metadata classes return null if the respective property does not exist.

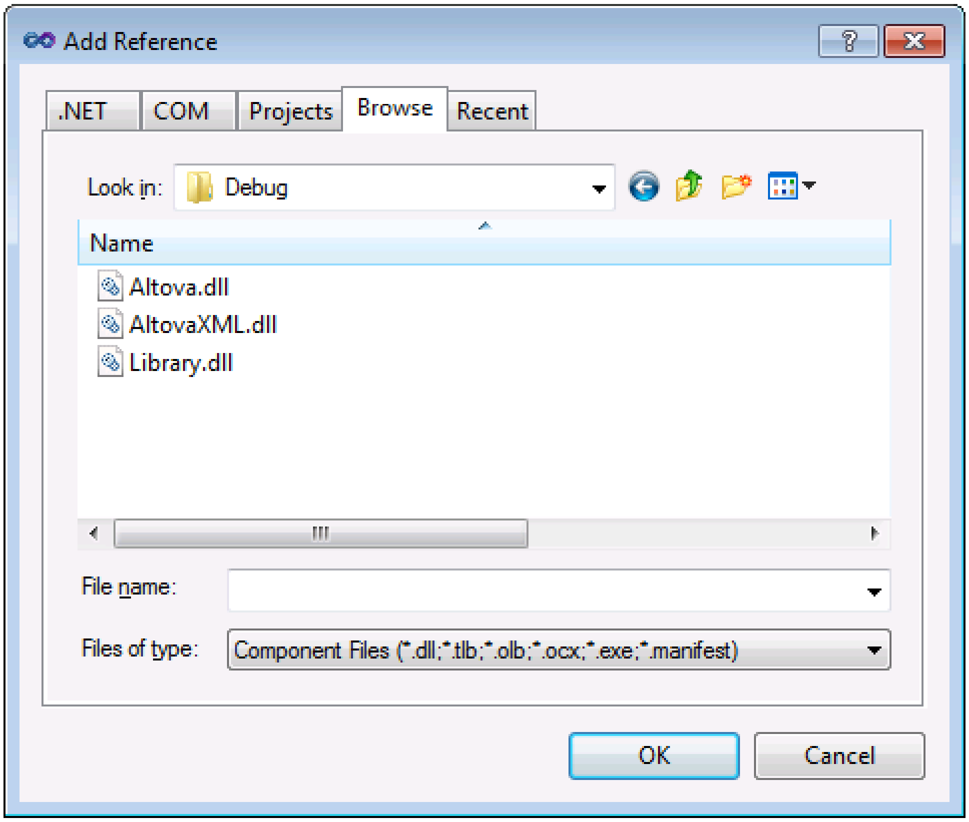
14.7.4 Integrating Schema Wrapper Libraries

To use the Altova libraries in your custom project, refer to the libraries from your project (or include them into 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**).



Look in: Debug

Name
Altova.dll
AltovaXML.dll
Library.dll

File name:

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

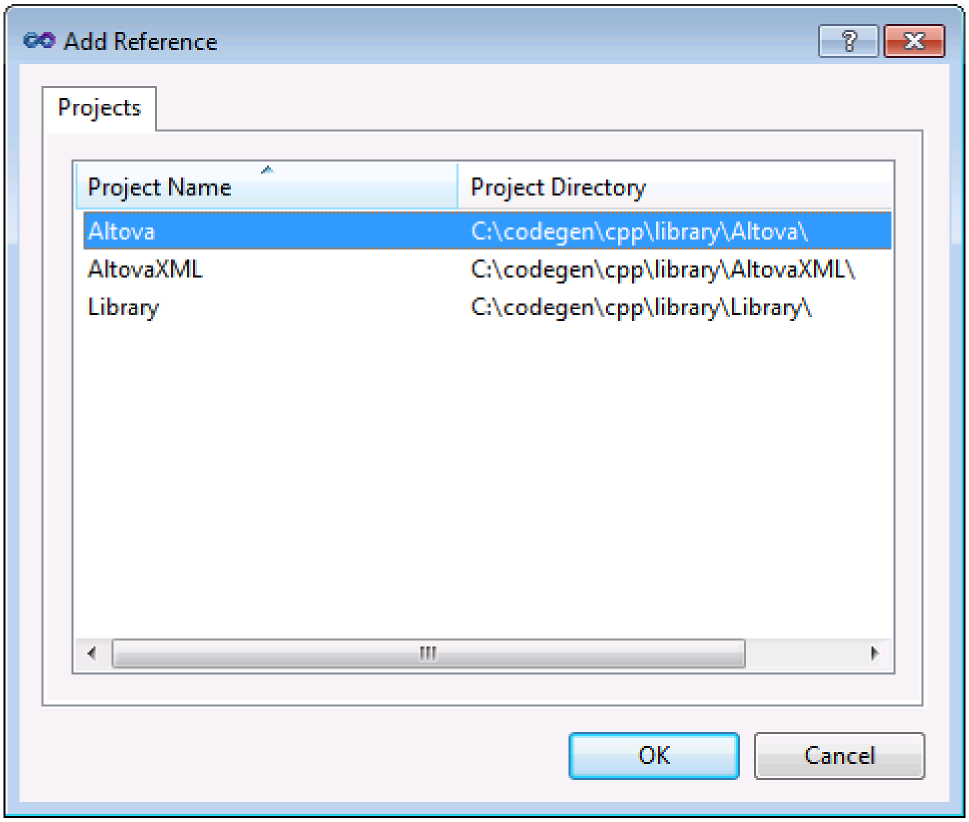
C++

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*.



Project Name	Project Directory
Altova	C:\codegen\cpp\library\Altova\
AltovaXML	C:\codegen\cpp\library\AltovaXML\
Library	C:\codegen\cpp\library\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

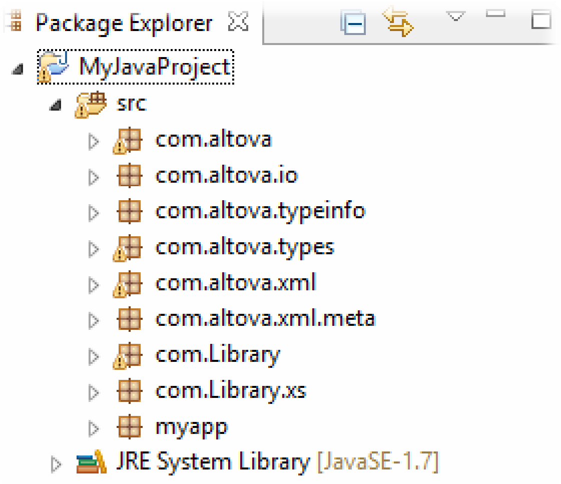
The option to generate static or dynamic libraries is available in code generation options (see Code generator options).

Java

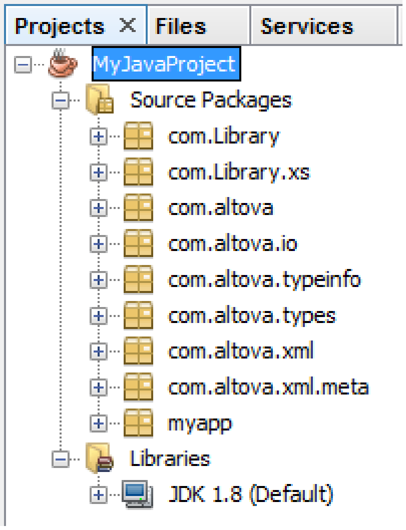
One of the ways to integrate the Altova packages into your Java project is to copy the **com** directory of the generated code to the directory which stores the source packages of your Java project (for example, **C:\Workspace\MyJavaProject\src**). For example, let's assume that you generated code in **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



MyJavaProject
Source Packages
com.Library
com.Library.xs
com.altova
com.altova.io
com.altova.typeinfo
com.altova.types
com.altova.xml
com.altova.xml.meta
myapp
Libraries
JDK 1.8 (Default)

Altova packages in NetBeans IDE 8.0.2

14.7.5 Example: Using the Schema Wrapper Libraries

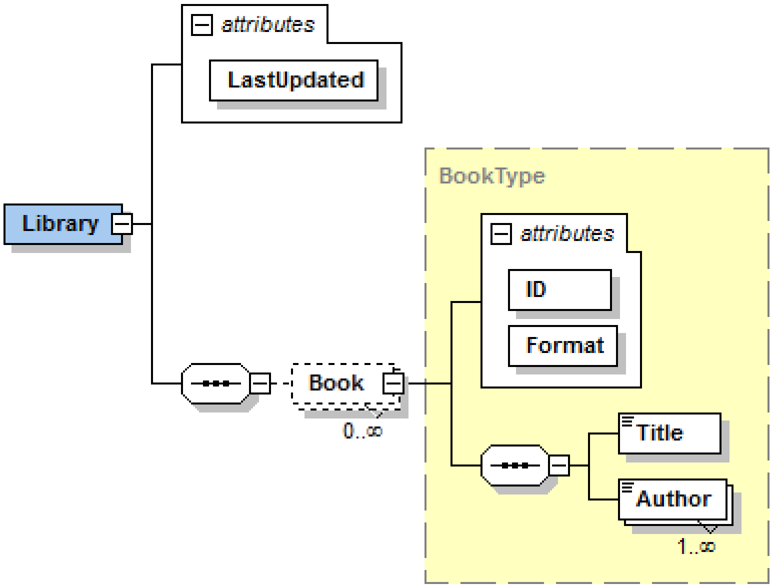
This example illustrates how to use the generated schema wrapper libraries in order to write or read programmatically XML documents conformant to the schema. Before using the sample code, take some time to understand the structure of the included example schema. You will need this schema to generate the code libraries used in this example. Understanding the example schema will help you get started with the code generated from your schema and adapt it to your needs.

14.7.5.1 Example Schema

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>
```

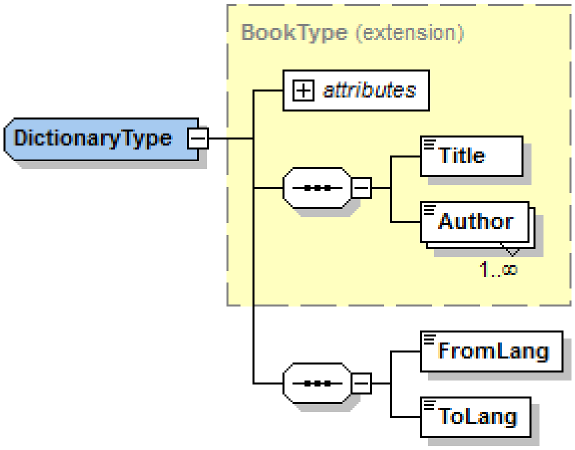
Library is a root element of a `complexType` which can be graphically represented as follows in the schema view of XMLSpy:



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>
```

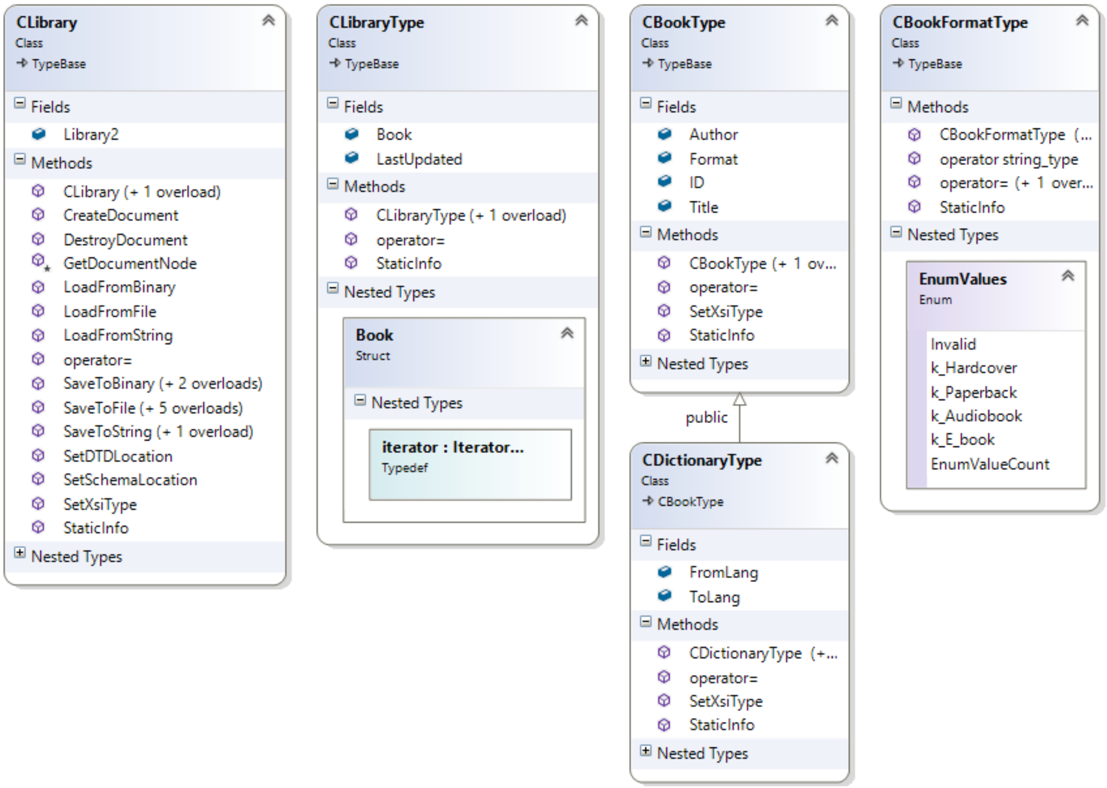
The next topics illustrate how to read from such a file programmatically, or write to such a file programmatically. To begin, generate the schema wrapper code from the schema above, using the steps described in Generating Code from XML Schemas or DTD.

14.7.5.2 Reading and Writing XML Documents (C++)

After you generate code from the Library schema (see Example Schema), a test C++ application is created, along with several supporting Altova libraries.

About the generated C++ libraries

The following diagram illustrates some of the most important classes of the generated code.



The central class of the generated code is the `CLibrary` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name (**Library.xsd**, in this example). 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 list of all members exposed by this class, see the class reference (`[YourSchema]::[CDoc]`).

The `Library2` field of the `CLibrary` class represents the actual root of the document. The number at the end is meant to avoid a naming conflict with the class name. **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 and [YourSchema]::MemberElement class reference.

The class CLibraryType is generated from the schema complex type with the same name, as mentioned in About Schema Wrapper Libraries (C++). Notice that the CLibraryType class contains a field Book, and a field 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.

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, 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).

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 Library; // required to work with Altova libraries

void Example()
{
	// Create a new, empty XML document
	CLibrary libDoc = CLibrary::CreateDocument();

	// Create the root element <Library> and add it to the document
	CLibraryType lib = libDoc.Library2.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 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.

```
using namespace Library;
void Example()
{
       // Load XML document
       CLibrary libDoc = CLibrary::LoadFromFile(_T("Library1.xml"));

       // Get the first (and only) root element <Library>
       CLibraryType lib = libDoc.Library2.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();
}
```

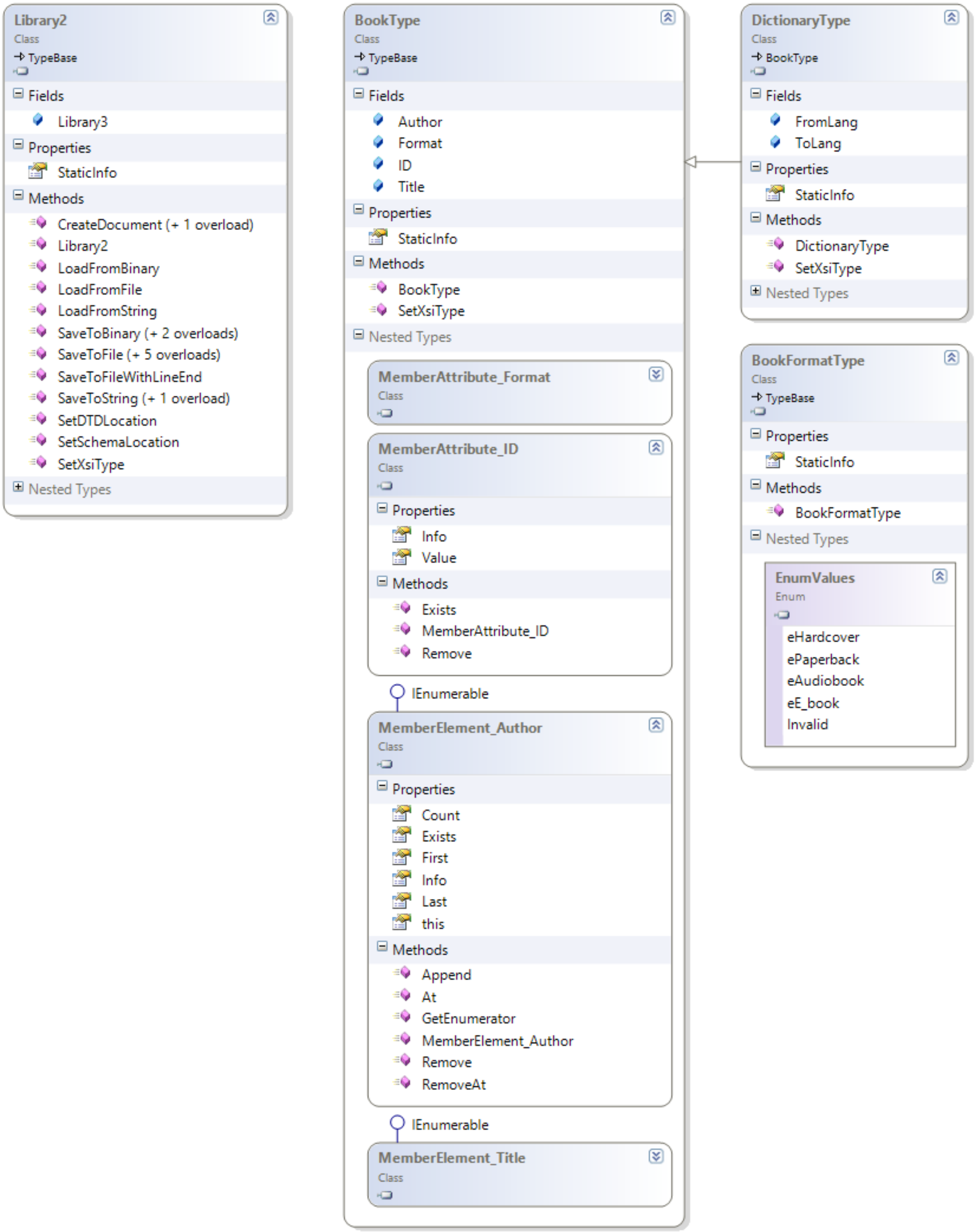
4. Press **F5** to start debugging.

14.7.5.3 Reading and Writing XML Documents (C#)

After you generate code from the Library schema (see Example Schema), a test C# application is created, along with several supporting Altova libraries.

About the generated C# libraries

The following diagram illustrates some of the most important classes of the generated code.



The central class of the generated code is the `Library2` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name (**Library.xsd**, in this example). Note that this class is called `Library2` 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]).

The Library3 member of the Library2 class represents the actual root of the document. Again, the number at the end is meant to avoid a naming conflict with the class name.

According to the code generation rules mentioned in About Schema Wrapper Libraries (C#), 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 above, 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 [YourSchemaType].MemberAttribute and [YourSchemaType].MemberElement 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, 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).

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 library
            Library2 doc = Library2.CreateDocument();
            // Append the root element
            LibraryType root = doc.Library3.Append();

            // 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;
            // 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 into a new Library instance
            Library2 doc = Library2.LoadFromFile("Library.xml");
            // Get the root element
            LibraryType root = doc.Library3.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.Library3.Append();
```

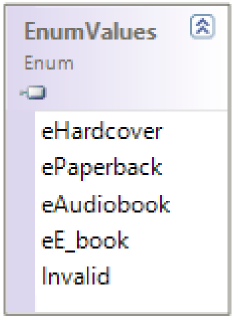
You can set the value of an attribute (like ID in this example) as follows:

```
// Set the value of the ID attribute
book.ID.Value = 1;
```

For further information, see `[YourSchemaType].MemberAttribute` and `[YourSchemaType].MemberElement` class reference.

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`:



EnumValues
Enum

eHardcover
ePaperback
eAudiobook
eE_book
Invalid

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` or `Altova.Types.Duration` object (either from `System.DateTime`, or by using parts such as hours and minutes, see `Altova.Types.DateTime` and `Altova.Types.Duration` 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` or `Altova.Types.Duration` 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` and `Altova.Types.Duration` 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:

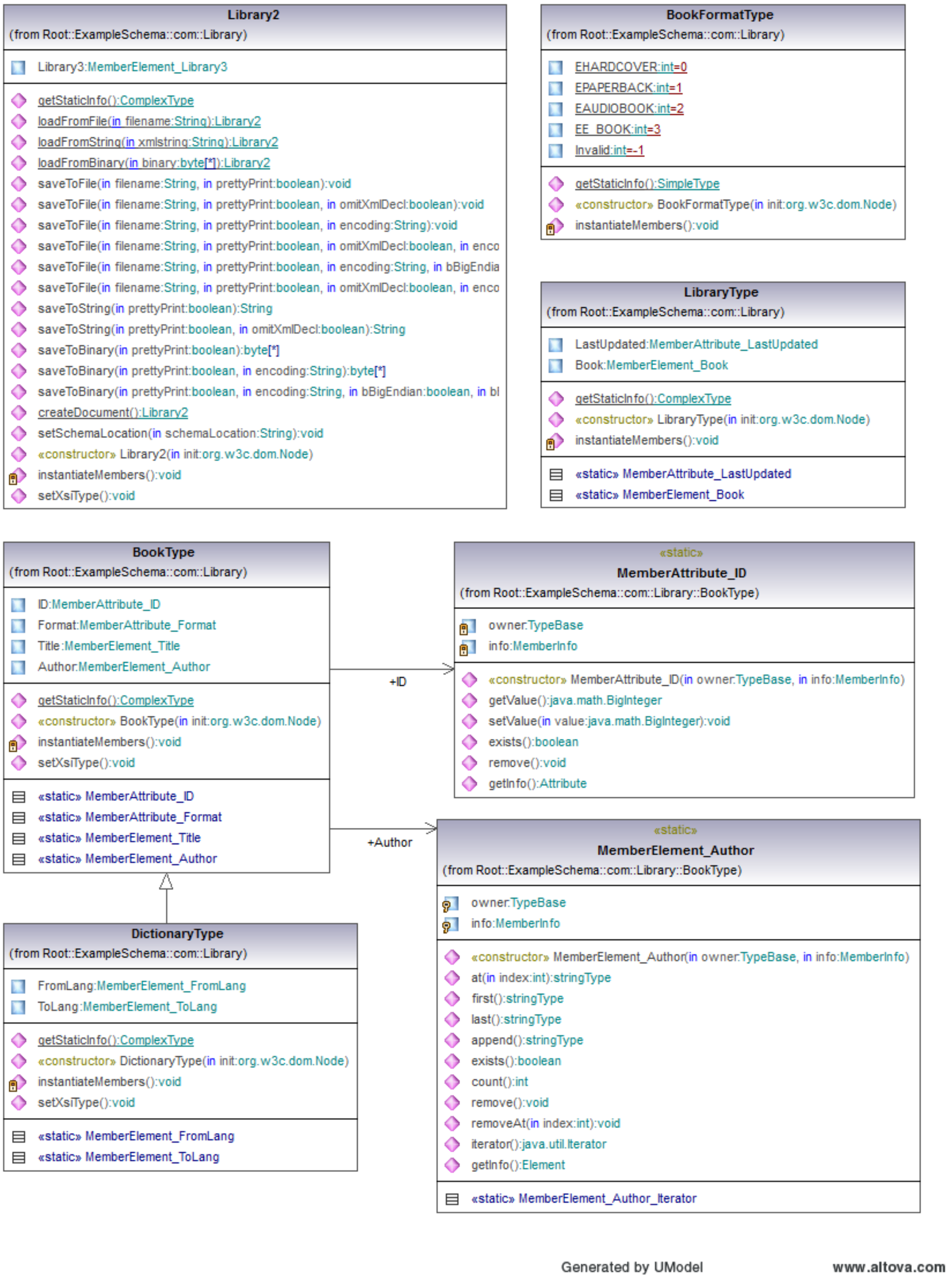
```
      // Determine if this book is of derived type
      if (book.Node.Attributes.GetNamedItem("xsi:type") != null)
      {
          // Find the value of the xsi:type attribute
          string xsiTypeValue =
book.Node.Attributes.GetNamedItem("xsi:type").Value;
          // Get the namespace URI 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");
          }
      }
```

14.7.5.4 Reading and Writing XML Documents (Java)

After you generate code from the Library schema (see Example Schema), a test Java project is created, along with several supporting Altova libraries.

About the generated Java libraries

The following diagram illustrates some of the most important classes of the generated code.


Library2
(from Root::ExampleSchema::com::Library)
Library3:MemberElement_Library3
getStaticInfo():ComplexType
loadFromFile(in filename:String):Library2
loadFromString(in xmlstring:String):Library2
loadFromBinary(in binary:byte[*]):Library2
saveToFile(in filename:String, in prettyPrint:boolean):void
saveToFile(in filename:String, in prettyPrint:boolean, in omitXmlDecl:boolean):void
saveToFile(in filename:String, in prettyPrint:boolean, in encoding:String):void
saveToFile(in filename:String, in prettyPrint:boolean, in omitXmlDecl:boolean, in enco
saveToFile(in filename:String, in prettyPrint:boolean, in encoding:String, in bBigEndia
saveToFile(in filename:String, in prettyPrint:boolean, in omitXmlDecl:boolean, in enco
saveToString(in prettyPrint:boolean):String
saveToString(in prettyPrint:boolean, in omitXmlDecl:boolean):String
saveToBinary(in prettyPrint:boolean):byte[*]
saveToBinary(in prettyPrint:boolean, in encoding:String):byte[*]
saveToBinary(in prettyPrint:boolean, in encoding:String, in bBigEndian:boolean, in bl
createDocument():Library2
setSchemaLocation(in schemaLocation:String):void
«constructor» Library2(in init:org.w3c.dom.Node)
instantiateMembers():void
setXsiType():void
BookFormatType
(from Root::ExampleSchema::com::Library)
EHARDCOVER:int=0
EPAPERBACK:int=1
EAUDIOBOOK:int=2
EE_BOOK:int=3
Invalid:int=-1
getStaticInfo():SimpleType
«constructor» BookFormatType(in init:org.w3c.dom.Node)
instantiateMembers():void
LibraryType
(from Root::ExampleSchema::com::Library)
LastUpdated:MemberAttribute_LastUpdated
Book:MemberElement_Book
getStaticInfo():ComplexType
«constructor» LibraryType(in init:org.w3c.dom.Node)
instantiateMembers():void
«static» MemberAttribute_LastUpdated
«static» MemberElement_Book
BookType
(from Root::ExampleSchema::com::Library)
ID:MemberAttribute_ID
Format:MemberAttribute_Format
Title:MemberElement_Title
Author:MemberElement_Author
getStaticInfo():ComplexType
«constructor» BookType(in init:org.w3c.dom.Node)
instantiateMembers():void
setXsiType():void
«static» MemberAttribute_ID
«static» MemberAttribute_Format
«static» MemberElement_Title
«static» MemberElement_Author
+ID
«static»
MemberAttribute_ID
(from Root::ExampleSchema::com::Library::BookType)
owner:TypeBase
info:MemberInfo
«constructor» MemberAttribute_ID(in owner:TypeBase, in info:MemberInfo)
getValue():java.math.BigInteger
setValue(in value:java.math.BigInteger):void
exists():boolean
remove():void
getInfo():Attribute
+Author
«static»
MemberElement_Author
(from Root::ExampleSchema::com::Library::BookType)
owner:TypeBase
info:MemberInfo
«constructor» MemberElement_Author(in owner:TypeBase, in info:MemberInfo)
at(in index:int):stringType
first():stringType
last():stringType
append():stringType
exists():boolean
count():int
remove():void
removeAt(in index:int):void
iterator():java.util.Iterator
getInfo():Element
«static» MemberElement_Author_Iterator
DictionaryType
(from Root::ExampleSchema::com::Library)
FromLang:MemberElement_FromLang
ToLang:MemberElement_ToLang
getStaticInfo():ComplexType
«constructor» DictionaryType(in init:org.w3c.dom.Node)
instantiateMembers():void
setXsiType():void
«static» MemberElement_FromLang
«static» MemberElement_ToLang
Generated by UModel
www.altova.com

The central class of the generated code is the `Library2` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name (**Library.xsd**, in this example). Note that this class is called `Library2` 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 (com.[YourSchema].[Doc]).

The `Library3` member of the `Library2` class represents the actual root of the document. Again, the number at the end is meant to avoid a naming conflict with the class name.

According to the code generation rules mentioned in About Generated Java Code, 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 above, 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 com.[YourSchema].[YourSchemaType].MemberAttribute and com.[YourSchema].[YourSchemaType].MemberElement 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, 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).

4. Edit the `Example()` method as shown below.

```
protected static void example() throws Exception {
      // create a new, empty XML document
      Library2 libDoc = Library2.createDocument();

      // create the root element <Library> and add it to the document
      LibraryType lib = libDoc.Library3.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
      Library2 libDoc = Library2.loadFromFile("Library1.xml");

      // get the first (and only) root element <Library>
      LibraryType lib = libDoc.Library3.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.Library.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.Library3.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));
```

For further information, see `com.[YourSchema].[YourSchemaType].MemberAttribute` and `com.[YourSchema].[YourSchemaType].MemberElement` class reference.

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` or `com.altova.types.Duration` 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` or `com.altova.types.Duration` 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` and `com.altova.types.Duration` 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.");
            }
         }
```

14.8 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). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code may include other supporting classes, which are not listed here and are subject to modification.

14.8.1 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.

Name	Description
`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.
`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; }
}
```

14.8.2 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).
`Duration(const YearMonthDuration& ym)`	Initializes a new instance of the `Duration` class to the duration defined by the `ym` argument (see altova::YearMonthDuration).
`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 and altova::DayTimeDuration).

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).
`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.
`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).
`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;
}
```

14.8.3 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.

14.8.4 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.

14.8.5 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.

14.8.6 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.
`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.

14.8.7 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.

14.8.8 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.

14.8.9 [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().
`void DestroyDocument()`	Destroys a document. All references to the document and its nodes are invalidated. This must be called when you finished 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

Name	Description
	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.
`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 Options).
`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 Options).
`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&`	Saves an XML document to a file with the

Name	Description
`fileName, bool prettyPrint, const string_type & encoding, bool bBigEndian, bool bBOM, const string_type & lineend)`	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 Options).
`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 Options).
`string_type SaveToString(bool prettyPrint)`	Saves an XML document to a string, with optional "pretty-print" formatting.
`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.

14.8.10 [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`).
`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.

14.8.11 [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.
`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.

Name	Description
`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`).
`MemberType last()`	Returns the last instance of the member element.
`void remove()`	Deletes all occurrences of the element from its parent.
`void remove(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.

14.9 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). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code may include other supporting classes, which are not listed here and are subject to modification.

14.9.1 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

	Name	Description
		`DateTime` object.
	`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.

	Name	Description
◆	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. 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.
◆	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.

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));
}
```

14.9.2 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.	`735913.63189734510879629629 63`
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	`2015-11`

Value	Description	Example
	gYearMonth.	(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`

14.9.3 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.
	`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`".
◆	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);
}
```

14.9.4 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.

14.9.5 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.

14.9.6 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 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.

14.9.7 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.
📄	XmlQualifiedName QualifiedName	Returns the qualified name of this type.
📄	int TotalDigits	Returns the value of this facet.
📄	WhitespaceType Whitespace	Returns the whitespace normalization facet.

14.9.8 [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 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).
	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	Saves an XML document to a file, with optional

	Name	Description
	`fileName, bool prettyPrint, string encoding, string lineend)`	"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.

14.9.9 [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`).
	`AttributeType Value`	Sets or gets the attribute value.

14.9.10 [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.
	`MemberType At(int index)`	Returns the member element specified by the index.

	Name	Description
	`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`).
	`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).

14.10 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 DTDs). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code may include other supporting classes, which are not listed here and are subject to modification.

14.10.1 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.

● S	**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:

	Name	Description
		`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.
●	`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

	Name	Description
		`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());
}
```

14.10.2 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, int newhour, int newminute, int newsecond, double newpartsecond )`	Returns a `Duration` object created from the number of days, hours, minutes, seconds, and fractional second parts supplied as argument.
●s	`static Duration getFromYearMonth( int newyear, int newmonth )`	Returns a `Duration` object created from the number of years and months supplied as argument.
●s	`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.
●s	`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** Must be used when the string **s** consists of any of

	Name	Description
		the following: days, hours, minutes, seconds, fractional second parts, for example `-P4DT4H4M4.774S`. **ParseType.DURATION** Must 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** Must 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.
●	`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 positive.

	Name	Description
●	`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());
}
```

14.10.3 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.

14.10.4 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`	Returns the namespace URI of the type.

	Name	Description
	`getNamespaceURI()`	

14.10.5 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.

14.10.6 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.

	Name	Description
●	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.
●	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

14.10.7 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	Creates a new, empty XML document.

	Name	Description
	`createDocument()`	
● 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.
●	`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 byte array, 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 byte array, 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 byte array, 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`	Saves an XML document to a file, with optional

	Name	Description
	`fileName, boolean prettyPrint, String encoding)`	"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 byte array, 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.

14.10.8 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`).
●	`AttributeType getValue()`	Gets the attribute value.
●	`void remove()`	Removes the attribute from its parent

	Name	Description
		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.

14.10.9 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.
●	`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`).
●	`MemberType getValue()`	Gets the element content (only generated if element can have simple or mixed content).

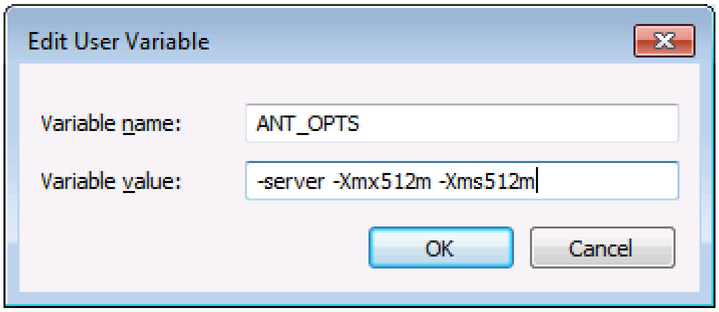
	Name	Description
●	`java.util.Iterator iterator()`	Returns an object for iterating instances of the member element.
●	`MemberType last()`	Returns the last instance of the member element.
●	`void remove()`	Deletes all occurrences of the element from its parent.
●	`void removeAt(int index)`	Deletes the 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).

14.11 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.



Variable name: ANT_OPTS

Variable value: -server -Xmx512m -Xms512m

- 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

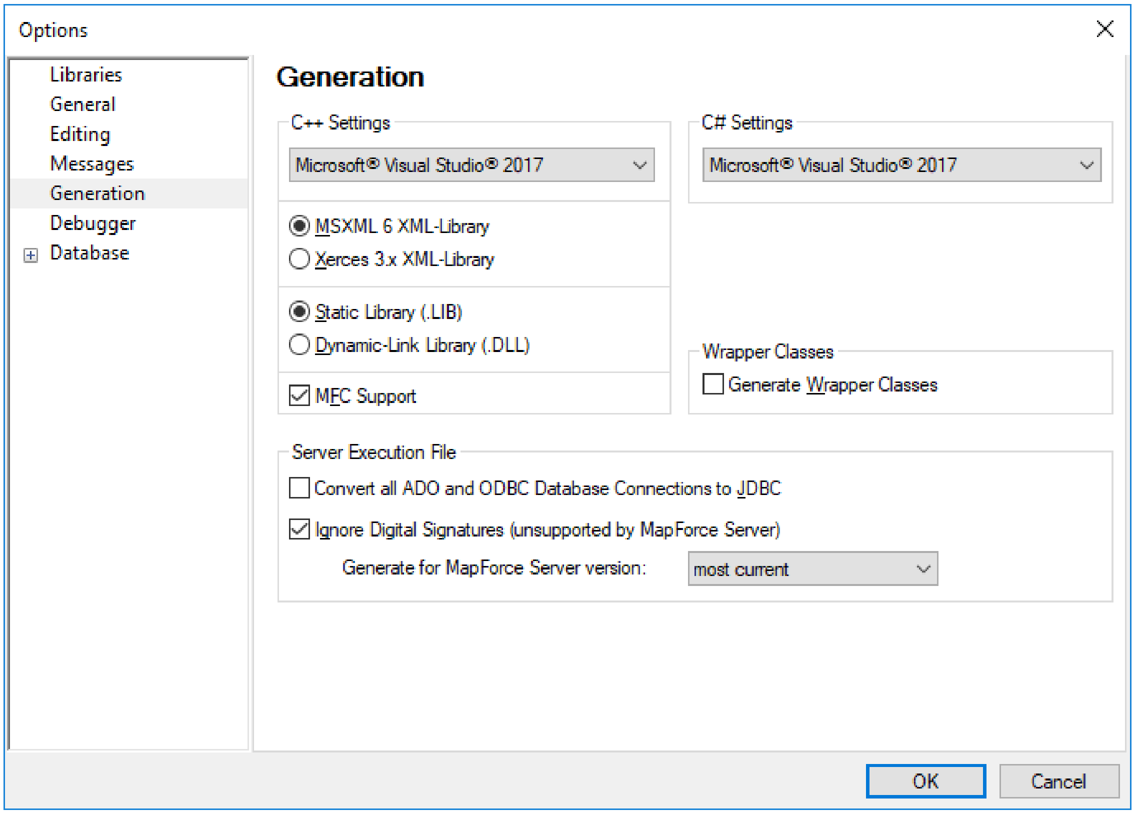
When customizing code generation using the supplied SPL files, it might be necessary to reserve names to avoid collisions with other symbols. To do this:

1. Navigate to subdirectory corresponding to the programming language of the **spl** subdirectory of the program installation directory e.g. `C:\Program Files\Altova \MapForce2018\spl\java\`.
2. Open the **settings.spl** file and insert a new line into the reserve section e.g. reserve "myReservedWord".
3. Regenerate the program code.

14.12 Code Generator Options

To view or change the MapForce settings applicable to code generation:

- On the **Tools** menu, click **Options**, and then click **Generation**.



Generation

C++ Settings: Microsoft® Visual Studio® 2017

● MSXML 6 XML-Library
○ Xerces 3.x XML-Library

● Static Library (.LIB)
○ Dynamic-Link Library (.DLL)

☑ MFC Support

C# Settings: Microsoft® Visual Studio® 2017

Wrapper Classes
□ Generate Wrapper Classes

Server Execution File
□ Convert all ADO and ODBC Database Connections to JDBC
☑ Ignore Digital Signatures (unsupported by MapForce Server)
Generate for MapForce Server version: most current

The available settings are as follows.

C++ Settings	Defines the specific compiler settings for the C++ environment, namely: • The Visual Studio version (2008, 2010, 2013, 2015, 2017) • 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	Defines the specific compiler settings for the C# environment, namely, the Visual Studio version (2008, 2010, 2013, 2015, 2017).
Wrapper Classes	Allows you to generate wrapper classes for XML schemas, see Generating Code from XML Schemas or DTDs. 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.

14.13 SPL (Spy Programming Language)

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. Please 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, avoid structures etc. which apply to specific schemas!

Example: Creating a new file in SPL:

```
[create "test.cpp"]
#include "stdafx.h"
[close]
```

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.

14.13.1 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 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 **]**.

14.13.2 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 *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 *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 *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 *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 *filename*

Example:

```
include "Module.cpp"
```

includes the specified file as SPL source. This allows you to split your template into multiple files for easier editing and handling.

14.13.3 Variables

Any non-trivial SPL file will require variables. Some variables are predefined 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 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.

14.13.4 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	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
$VSVersion	integer	Specifies the Visual Studio version. Valid values: **0** No Visual Studio project **2008** Visual Studio 2008 **2010** Visual Studio 2010 **2013** Visual Studio 2013 **2015** Visual Studio 2015 **2017** Visual Studio 2017

For C# generation only:

Name	Type	Description
$VSVersion	integer	Specifies the Visual Studio version. Valid values: **0** No Visual Studio project **2008** Visual Studio 2008 **2010** Visual Studio 2010 **2013** Visual Studio 2013 **2015** Visual Studio 2015 **2017** Visual Studio 2017

14.13.5 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 *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 *string*

writes the string to the current output file.

Example:

```
[write "C" & $name]
```

This can also be written as:

```
C[=$name]
```

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"
```

14.13.6 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

14.13.7 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
```

Please note that there are no round brackets enclosing the condition!
As in any other programming language, conditions are constructed with logical and comparison operators.

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.

14.13.8 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]
```

The first line:
$classes is the **global object** 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]
```

14.13.9 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)

14.13.9.1 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.

Please 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
]
```

14.13.9.2 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")
```

14.13.9.3 *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
]
```

14.13.10 Built in Types

The section describes the properties of the built-in types used in the predefined variables which describe the parsed schema.

14.13.10.1 Library

This object represents the whole library generated from the XML Schema or DTD.

Property	Type	Description
SchemaNamespaces	Namespace 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)

14.13.10.2 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 collection	All types contained in this namespace
Library	Library	Library containing this namespace

14.13.10.3 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

Property	Type	Description
		name or parent declaration)
LocalName	string	Original name in the schema
Namespace	Namespace	Namespace containing this type
Attributes	Member collection	Attributes contained in this type*
Elements	Member 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	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	Native binding for this type
Facets	Facets	Facets of this type
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.

14.13.10.4 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	Type originally declaring the member (equal to ContainingType for non-inherited members)
ContainingType	Type	Type where this is a member of
DataType	Type	Data type of this member's content
Library	Library	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
Default	string	Default value

14.13.10.5 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

14.13.10.6 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

Property	Type	Description
NamespaceURI	string	Facet namespace
FacetType	string	one of "normalization", "lexicalspace", "valuespace-length", "valuespace-enum" or "valuespace-range"
DeclaringType	Type	Type this facet is declared on
FacetCheckerName	string	Name of facet checker (from schemafacet map)
FacetValue	string or integer	Actual value of this facet

Chapter 15

The MapForce API

15 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: 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.

15.1 Overview

This overview of the MapForce API provides you with the object model for the API and a description of the most important API concepts. The following topics are covered:

- Accessing the API
- The Object Model
- Error Handling
- Examples

15.1.1 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: **(My) Documents\Altova\MapForce2018\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\MapForce2018\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: **(My) Documents\Altova\MapForce2018\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). A sample MapForce API client in JScript is available at: **(My) Documents\Altova\MapForce2018\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`.

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.
- 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:

Visual Basic

```
Dim objMapForce As MapForceLib.Application = New MapForceLib.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");
```

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"); }
```

15.1.2 The Object Model

The starting point for every application which uses the MapForce API is the `Application` 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
   Options
   Project
      ProjectItem
   Documents
      Document
         MapForceView
         Mapping
            Component
               Datapoint
            Components
            Connection
         Mappings
   ErrorMarkers
      ErrorMarker
AppOutputLines
   AppOutputLine
      AppOutputLines
         ...
      AppOutputLineSymbol
Enumerations
```

For information about creating an instance the Application object, see Accessing the API. For reference to the objects exposed by the API, see Object Reference.

15.1.3 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();
   }
}
```

15.1.4 Examples

Programming languages differ in the way they support COM access. The following examples for C#, Java, and JScript will help you get started. The code listings in this section are available at **C:/Users/<username>/Documents/Altova/MapForce2018/MapForceExamples/API**.

15.1.4.1 *Example C# Project*

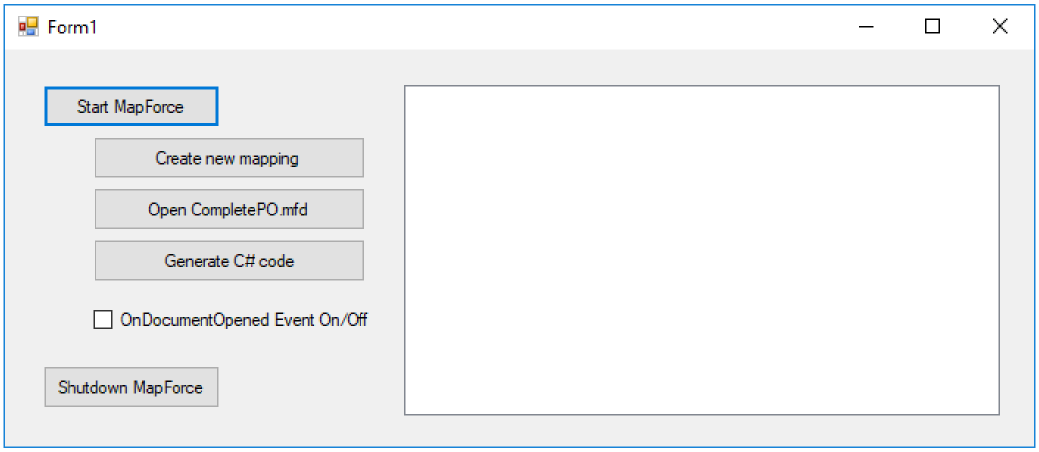
After you install MapForce, an example MapForce API client project for C# is available in the directory **C:/Users/<username>/Documents/ Altova/MapForce2018/MapForceExamples/API**.

You can compile and run the project with Visual Studio 2008 or later. 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(handleOnDocumentO
pened);
                else
                    MapForce.OnDocumentOpened -= new
MapForceLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentO
pened);
            }
        }
    }
}
```

15.1.4.2 *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/MapForce2018/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 XMLSpy 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");
      }
   }
}
```

15.1.4.3 JScript Examples

After you install MapForce, an example MapForce API client project for JScript is available in the directory **C:/Users/<username>/Documents/ Altova/MapForce2018/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 (see Start Application).
`DocumentAccess.js`	Shows how to open, iterate and close documents (see Simple Document Access).
`GenerateCode.js`	Shows how to invoke code generation using JScript (see Generate Code).
`Readme.txt`	Provides basic help to run the scripts.

This documentation additionally includes a few extra JScript code listings:

- Example: Code Generation
- Example: Mapping Execution
- Example: Project Support

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.

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`.

```
// 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
```

The code listed above is available as a sample file (see JScript Examples). To run the script, start it from a command prompt window or from Windows Explorer.

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). To run the script, start it from a command prompt window or from Windows Explorer.

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). To run the script, start it from a command prompt window or from Windows Explorer.

Example: Code Generation

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 ---------------------
```

Example: Mapping Execution

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\\MapForce2018\\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 );
}
```

Example: Project Support

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 **MapForceExamples** folder of your MapForce installation.

To successfully run all operations in this example below, you will need the Enterprise version of MapForce. If you have the Professional version running, 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\\MapForce2018\
\MapForceExamples\\Tutorial\\";

// ////////////////////// 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
always 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_Lan
guage +
                                " 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.
   // sump properties of all project items

   objProject = objMapForce.OpenProject(strSamplePath +
"MapForceExamples.mfp");
   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(0);
      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(0);
   objFolder1.GenerateCode();
}

// /////////////////////// MAIN //////////////////////////////

CreateGlobalObjects();
objMapForce.Visible = true;

LoadSampleProject();
CreateNewProject();
GenerateCodeForNewProject();

// uncomment to shut down application when script ends
// objMapForce.Visible = false;
```

15.2 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 Examples.

In Java, some syntax variations to the object names exist, as follows:

- ***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. 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`

15.2.1 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.

Events
Events

Properties and Methods
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`

15.2.1.1 Events

This object supports the following events:

`OnDocumentOpened`
`OnProjectOpened`
`OnShutdown`

OnDocumentOpened

Event: `OnDocumentOpened` (*`i_objDocument`* as `Document`)

Description
This event is triggered when an existing or new document is opened. The corresponding close event is `Document.OnDocumentClosed`.

OnProjectOpened

Event: `OnProjectOpened` (*`i_objProject`* as `Project`)

Description
This event is triggered when an existing or new project is loaded into the application. The corresponding close event is `Project.OnProjectClosed`.

OnShutdown

Event: `OnShutdown` ()

Description
This event is triggered when the application is shutting down.

15.2.1.2 ActiveDocument

Property: `ActiveDocument` as `Document` (read-only)

Description
Returns the automation object of the currently active document. This property returns the same as `Documents.ActiveDocument`.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

15.2.1.3 ActiveProject

Property: `ActiveProject` as `Project` (read-only)

Description
Returns the automation object of the currently active project.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

15.2.1.4 Application

Property: `Application` as `Application` (read-only)

Description

Retrieves the application's top-level object.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.5 Documents

Property: `Documents` as `Documents` (read-only)

Description

Returns a collection of all currently open documents.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.6 Edition

Property: `Edition` as `String` (read-only)

Description

Returns the edition of the application, e.g. "Altova MapForce Enterprise Edition" for the enterprise edition.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.7 GlobalResourceConfig

Property: `GlobalResourceConfig` as `String`

Description

Gets or sets the name of the active global resource configuration file. Per 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.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.8 GlobalResourceFile

Property: `GlobalResourceFile` as `String`

Description
Gets or sets the global resource definition file. Per default the file is called GlobalResources.xml.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.9 HighlightSerializedMarker

Method: `HighlightSerializedMarker` (*`i_strSerializedMarker`* as `String`)

Description
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.

Errors

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.

15.2.1.10 IsAPISupported

Property: `IsAPISupported` as `Boolean` (read-only)

Description
Returns true if the API is supported in this version of MapForce.

Errors

1001 Invalid address for the return parameter was specified.

15.2.1.11 MajorVersion

Property: `MajorVersion` as `Long` (read-only)

Description
Gets the major version number of 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.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.12 MinorVersion

Property: `MinorVersion` as `Long` (read-only)

Description

The minor version number of the product, e.g. `2` for `2006 R2 SP1`.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.13 Name

Property: `Name` as `String` (read-only)

Description

The name of the application.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.14 NewDocument

Method: `NewDocument` () as `Document`

Description

Creates a new empty document. The newly opened document becomes the `ActiveDocument`. This method is a shortened form of `Documents.NewDocument`.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.15 NewProject

Method: `NewProject` () as `Project`

Description

Creates a new empty project. The current project is closed. The new project is accessible under `ActiveProject`.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.16 *OpenDocument*

Method: `OpenDocument` (*`i_strFileName`* as `String`) as `Document`

Description

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`.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.17 *OpenProject*

Method: `NewProject` () as `Project`

Description

Opens an existing Mapforce project (`*.mfp`). The current project is closed. The newly opened project is accessible under `ActiveProject`.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.18 *OpenURL*

Method: `OpenURL` (*`i_strURL`* as `String`, *`i_strUser`* as `String`, *`i_strPassword`* as `String`)

Description

Loads a previously saved document file from an URL location. Allows user name and password to be supplied.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.19 Options

Property: `Options` as `Options` (read-only)

Description
This property gives access to options that configure the generation of code.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.20 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors
1000 The object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.21 Quit

Method: `Quit` ()

Description
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.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.22 ServicePackVersion

Property: `ServicePackVersion` as `Long` (read-only)

Description
The service pack version number of the product, e.g. `1` for `2010 R2 SP1`.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.23 Status

Property: `Status` as `Long` (read-only)

Description
The status of the application. It is one of the values of the `ENUMApplicationStatus` enumeration.

Errors
1001 Invalid address for the return parameter was specified.

15.2.1.24 Visible

Property: `Visible` as `Boolean`

Description
`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 `MapForce.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.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.1.25 WindowHandle

Property: `WindowHandle` () as `long` (read-only)

Description
Retrieve the application's Window Handle.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

15.2.2 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 and Methods
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`

15.2.2.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

15.2.2.2 ChildLines

Property: `ChildLines` as `AppOutputLines` (read-only)

Description
Returns a collection of the current line's direct child lines.

Errors

4100	The application object is no longer valid.
4101	Invalid address for the return parameter was specified.

15.2.2.3 GetCellCountInLine

Method: `GetCellCountInLine` (*`nLine`* as `Long`) as `Long`

Description
Gets the number of cells in the sub-line indicated by *nLine* in the current AppOutputLine.

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

15.2.2.4 GetCellIcon

Method: `GetCellIcon` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `Long`

Description
Gets the icon of the cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine*.

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

15.2.2.5 GetCellSymbol

Method: `GetCellSymbol` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `AppOutputLineSymbol`

Description
Gets the symbol of the cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine*.

Errors

4100	The object is no longer valid.

4101 Invalid address for the return parameter was specified.

15.2.2.6 GetCellText

Method: `GetCellText` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `String`

Description
Gets the text of the cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine*.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

15.2.2.7 GetCellTextDecoration

Method: `GetCellTextDecoration` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `Long`

Description
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.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

15.2.2.8 GetIsCellText

Method: `GetIsCellText` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `Boolean`

Description
Returns true, if the cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine* is a text cell.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

15.2.2.9 GetLineCount

Method: `GetLineCount` () as `Long`

Description

Gets the number of sub-lines the current line consists of.

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

15.2.2.10 GetLineSeverity

Method: `GetLineSeverity` () as `Long`

Description

Gets the severity of the line. It can be one of the `ENUMAppOutputLine_Severity` values:

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

15.2.2.11 GetLineSymbol

Method: `GetLineSymbol` () as `AppOutputLineSymbol`

Description

Gets the symbol assigned to the whole line.

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

15.2.2.12 GetLineText

Method: `GetLineText` () as `String`

Description

Gets the contents of the line as text.

Errors

4100	The object is no longer valid.
4101	Invalid address for the return parameter was specified.

15.2.2.13 GetLineTextEx

Method: `GetLineTextEx` (*`psTextPartSeperator`* as `String`, *`psLineSeperator`* as `String`) as `String`

Description
Gets the contents of the line as text using the specified part and line separators.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

15.2.2.14 GetLineTextWithChildren

Method: `GetLineTextWithChildren` () as `String`

Description
Gets the contents of the line including all child and descendant lines as text.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

15.2.2.15 GetLineTextWithChildrenEx

Method: `GetLineTextWithChildrenEx` (*`psPartSep`* as `String`, *`psLineSep`* as `String`, *`psTabSep`* as `String`, *`psItemSep`* as `String`) as `String`

Description
Gets the contents of the line including all child and descendant lines as text using the specified part, line, tab and item separators.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

15.2.2.16 Parent

Property: `Parent` as `AppOutputLines` (read-only)

Description
The parent object according to the object model.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

15.2.3 AppOutputLines

Represents a collection of AppOutputLine message lines.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Iterating through the collection:
`Count`
`Item`

15.2.3.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

4000	The object is no longer valid.
4001	Invalid address for the return parameter was specified.

15.2.3.2 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of lines in the collection.

Errors

4000	The object is no longer valid.
4001	Invalid address for the return parameter was specified.

15.2.3.3 Item

Property: `Item` (*`nIndex`* as `Integer`) as `AppOutputLine` (read-only)

Description
Retrieves the line at `nIndex` from the collection. Indices start with 1.

Errors

4000	The object is no longer valid.

4001 Invalid address for the return parameter was specified.

15.2.3.4 Parent

Property: `Parent` as `AppOutputLine` (read-only)

Description
The parent object according to the object model.

Errors

4000	The object is no longer valid.
4001	Invalid address for the return parameter was specified.

15.2.4 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 and Methods
Properties to navigate the object model:
`Application`
`Parent`

Access to AppOutputLineSymbol methods:
`GetSymbolHREF`
`GetSymbolID`
`IsSymbolHREF`

15.2.4.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

15.2.4.2 GetSymbolHREF

Method: `GetSymbolHREF` () as `String`

Description
If the symbol is of type URL, returns the URL as a string.

Errors

4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

15.2.4.3 GetSymbolID

Method: `GetSymbolHREF` () as `Long`

Description
Gets the ID of the symbol.

Errors

4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

15.2.4.4 IsSymbolHREF

Method: `IsSymbolHREF` () as `Boolean`

Description
Indicates if the symbol is of kind URL.

Errors

4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

15.2.4.5 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors

4200	The object is no longer valid.
4201	Invalid address for the return parameter was specified.

15.2.5 Component

A Component represents a MapForce component.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Component properties:
`HasIncomingConnections`
`HasOutgoingConnections`
`CanChangeInputInstanceFile`
`CanChangeOutputInstanceFile`
`ComponentName`

`ID`
`IsParameterInputRequired`
`IsParameterSequence`
`Name`
`Preview`
`Schema`
`SubType`
`Type`

Instance related properties:
`InputInstanceFile`
`OutputInstanceFile`

Datapoints:
`GetRootDatapoint`

Execution:
`GenerateOutput`

15.2.5.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.2 CanChangeInputInstanceFile

Property: CanChangeInputInstanceFile as Boolean (read-only)

Description
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.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.3 CanChangeOutputInstanceFile

Property: CanChangeOutputInstanceFile as Boolean (read-only)

Description
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.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.4 ComponentName

Property: ComponentName as `String`

Description
Gets or sets the component's name.

Errors

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.

15.2.5.5 GenerateOutput

Method: `GenerateOutput` (*[out]* *`pbError`* as `Boolean`) as `AppOutputLines`

Description
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.

Remarks
pbError is an output-only parameter. You will receive a value only if the calling language supports

output parameters. If not, the value you pass here will remain unchanged when the function has finished.

Errors

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.

15.2.5.6 GetRootDatapoint

Method: GetRootDatapoint(side as ENUMComponentDatapointSide, strNamespace as String, strLocalName as String, strParameterName as String) as Datapoint

Description

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.

The *side* parameter indicates if an input, or output, datapoint of a component is to be retrieved.

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 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.

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

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1231	Datapoint not found.

15.2.5.7 HasIncomingConnections

Property: `HasIncomingConnections` as `Boolean` (read-only)

Description
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.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.8 HasOutgoingConnections

Property: `HasOutgoingConnections` as `Boolean` (read-only)

Description
Indicates if the component has any outgoing connections (on its right side).

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.9 ID

Property: `ID` as `Unsigned Long` (read-only)

Description
Retrieves the component ID.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.10 InputInstanceFile

Property: `InputInstanceFile` as `String`

Description
Gets or sets the component's input instance file.

Errors

1200	The object is no longer valid.

1201 Invalid address for the return parameter was specified.

15.2.5.11 IsParameterInputRequired

Property: IsParameterInputRequired as `Boolean`

Description
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.

Errors
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.

15.2.5.12 IsParameterSequence

Property: IsParameterSequence as `Boolean`

Description
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.

Errors
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.

15.2.5.13 Name

Property: `Name` as `String (read only)`

Description
Gets the component's name.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

15.2.5.14 OutputInstanceFile

Property: `OutputInstanceFile` as `String`

Description
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.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.15 Parent

Property: `Parent` as `Mapping` (read-only)

Description
The parent object according to the object model.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.16 Preview

Property: Preview as `Boolean`

Description
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.

Errors

1200	The object is no longer valid.
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.

15.2.5.17 Schema

Property: `Schema` as `String` (read-only)

Description
Retrieves the component's schema file name.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.18 SubType

Property: `SubType` as `ENUMComponentSubType` (read-only)

Description
Retrieves the component's sub type.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.19 Type

Property: `Type` as `ENUMComponentType` (read-only)

Description
Retrieves the component's type.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.5.20 UsageKind

Property: `UsageKind` as `ENUMUsageKind` (read-only)

Description
Retrieves the component's usage kind.

Errors

1200	The object is no longer valid.

1201 Invalid address for the return parameter was specified.

15.2.6 Components

Represents a collection of Component objects.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Iterating through the collection:
`Count`
`Item`

15.2.6.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.6.2 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of components in the collection.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.6.3 Item

Property: `Item` (*`nIndex`* as `Integer`) as `Component` (read-only)

Description
Retrieves the component at `nIndex` from the collection. Indices start with 1.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

15.2.6.4 Parent

Property: `Parent` as `Mapping` (read-only)

Description
The parent object according to the object model.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

15.2.7 Connection

A Connection object represents a connector between two components.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Properties
`ConnectionType`

15.2.7.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
2100 The object is no longer valid.
2101 Invalid address for the return parameter was specified.

15.2.7.2 ConnectionType

Property: ConnectionType as ENUMConnectionType

Description
Gets or sets the connection's type.

Errors

2100	The application object is no longer valid.
2101	Invalid address for the return parameter was specified.
2102	Changing the document not allowed. It is read-only.
2103	Failed changing connection type.

15.2.7.3 Parent

Property: `Parent` as `Mapping` (read-only)

Description

The parent object according to the object model.

Errors

2100	The object is no longer valid.
2101	Invalid address for the return parameter was specified.

15.2.8 Datapoint

A Datapoint object represents an input or output icon of a component.

Properties and Methods

Properties to navigate the object model:

`Application`

`Parent`

Methods

`GetChild`

15.2.8.1 Application

Property: `Application` as `Application` (read-only)

Description

Retrieves the application's top-level object.

Errors

2000	The object is no longer valid.
2001	Invalid address for the return parameter was specified.

15.2.8.2 GetChild

Method: GetChild(strNamespace as String, strLocalName as String, searchFlags as ENUMSearchDatapointFlags) as Datapoint

Description
Scans for a direct child datapoint of the current datapoint, by namespace and local name.

Search flags can be passed as combination of values (combined using binary OR) of the ENUMSearchDatapointFlags enumeration.

A schema component with elements that contain mixed content, each display 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.

Errors

2000	The application object is no longer valid.
2001	Invalid address for the return parameter was specified.
2002	Datapoint not found.

15.2.8.3 Parent

Property: `Parent` as `Component` (read-only)

Description
The parent object according to the object model.

Errors

2000	The object is no longer valid.
2001	Invalid address for the return parameter was specified.

15.2.9 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.

Events
Events

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

File handling:
Activate

Close
FullName
Name
Path
Saved
Save
SaveAs

Mapping handling:
MainMapping
Mappings
CreateUserDefinedFunction

Component handling:
FindComponentByID

Code generation:
OutputSettings_ApplicationName
JavaSettings_BasePackageName

GenerateCHashCode
GenerateCodeEx
GenerateCppCode
GenerateJavaCode
GenerateXQuery
GenerateXSLT
GenerateXSLT2
HighlightSerializedMarker

Mapping execution:
GenerateOutput
GenerateOutputEx

View access:
MapForceView

Obsolete:
OutputSettings_Encoding

15.2.9.1 Events

This object supports the following events:

OnDocumentClosed
OnModifiedFlagChanged

OnDocumentClosed

Event: OnDocumentClosed (*i_objDocument* as Document)

Description
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

Event: `OnModifiedFlagChanged` (*`i_bIsModified`* as Boolean)

Description
This event is triggered when a document's modification status changes.

15.2.9.2 Activate

Method: `Activate` ()

Description
Makes this document the active document.

Errors

1200	The application object is no longer valid.

15.2.9.3 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.4 Close

Method: `Close` ()

Description
Closes the document without saving.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.5 CreateUserDefinedFunction

Method: CreateUserDefinedFunction(strFunctionName as String, strLibraryName as String, strSyntax as String, strDetails as String, bInlinedUse as Boolean) as Mapping

Description
Creates a user defined function in the current document.

Errors

1200 The application 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.

15.2.9.6 FindComponentByID

Method: `FindComponentByID` (*nID* as `Unsigned Long`) as `Component`

Description
Searches in the whole document, so all its mappings, for the component with the specified id.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

15.2.9.7 FullName

Property: `FullName` as String

Description
Path and name of the document file.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

15.2.9.8 GenerateCHashCode

Method: `GenerateCHashCode` ()

Description
Generate C# code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

15.2.9.9 GenerateCodeEx

Method: `GenerateCodeEx` (*`i_nLanguage`* as `ENUMProgrammingLanguage`) as `ErrorMarkers`

Description

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.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

15.2.9.10 GenerateCppCode

Method: `GenerateCppCode` ()

Description

Generates C++ code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

15.2.9.11 GenerateJavaCode

Method: `GenerateJavaCode` ()

Description

Generates Java code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

15.2.9.12 GenerateOutput

Method: `GenerateOutput` ()

Description

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.

Errors

1200 The application 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.

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.

15.2.9.13 GenerateOutputEx

Method: `GenerateOutputEx` () as `AppOutputLines`

Description

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`.

Errors

1200 The application 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.

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.

15.2.9.14 GenerateXQuery

Method: `GenerateXQuery` ()

Description

Generates mapping code as XQuery. Uses the properties defined in `Application.Options` to configure code generation.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1204 Error during XSLT/XSLT2/XQuery code generation.

15.2.9.15 GenerateXSLT

Method: `GenerateXSLT` ()

Description
Generates mapping code as XSLT. Uses the properties defined in `Application.Options` to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1204 Error during XSLT/XSLT2/XQuery code generation.

15.2.9.16 GenerateXSLT2

Method: `GenerateXSLT2` ()

Description
Generates mapping code as XSLT2. Uses the properties defined in `Application.Options` to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1204 Error during XSLT/XSLT2/XQuery code generation.

15.2.9.17 HighlightSerializedMarker

Method: `HighlightSerializedMarker` (*`i_strSerializedMarker`* as `String`)

Description
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.

Errors
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 a serialized MapForce marker.
1008 The marker points to a location that is no longer valid.

15.2.9.18 JavaSettings_BasePackageName

Property: `JavaSettings_BasePackageName` as `String`

Description
Sets or retrieves the base package name used when generating Java code. This property is available in UI-dialog for the Document Settings.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.19 MainMapping

Property: `MainMapping` as `Mapping` (read-only)

Description
Retrieves the main mapping of the document.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.20 MapForceView

Property: `MapForceView` as `MapForceView` (read-only)

Description
This property gives access to functionality specific to the MapForce view.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.21 Mappings

Property: `Mappings` as `Mappings` (read-only)

Description
Returns a collection of the mappings contained in the document.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.22 Name

Property: `Name` as `String`

Description
Name of the document file without file path.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.23 OutputSettings_ApplicationName

Property: `OutputSettings_ApplicationName` as `String`

Description
Sets or retrieves the application name available in the Document Settings dialog.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.24 OutputSettings_Encoding (obsolete)

Property: `OutputSettings_Encoding` as `String`

Description
obsolete
This property is not supported anymore. Mapping output encoding settings do not exist anymore. Components have individual output encoding settings.

15.2.9.25 Parent

Property: `Parent` as `Documents` (read-only)

Description
The parent object according to the object model.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.26 Path

Property: `Path` as `String`

Description
Path of the document file without name.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.27 Save

Method: `Save` ()

Description
Save the document to the file defined by `Document.FullName`.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.28 SaveAs

Method: `SaveAs` (*`i_strFileName`* as `String`)

Description
Save document to specified file name, and set `Document.FullName` to this value if save operation was successful.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.9.29 Saved

Property: `Saved` as `Boolean` (read-only)

Description
`True` if the document was not modified since the last save operation, `false` otherwise.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.10 Documents

Represents a collection of Document objects.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Open and create mappings:
`OpenDocument`
`NewDocument`

Iterating through the collection:
`Count`
`Item`
`ActiveDocument`

15.2.10.1 ActiveDocument

Property: `ActiveDocument` as Document (read-only)

Description
Retrieves the active document. If no document is open, `null` is returned.

Errors

1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

15.2.10.2 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

15.2.10.3 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of documents in the collection.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

15.2.10.4 Item

Property: `Item` (*`nIndex`* as `Integer`) as `Document` (read-only)

Description
Retrieves the document at `nIndex` from the collection. Indices start with 1.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

15.2.10.5 NewDocument

Method: `NewDocument` () as `Document`

Description
Creates a new document, adds it to the end of the collection, and makes it the active document.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

15.2.10.6 OpenDocument

Method: `OpenDocument` (*`strFilePath`* as `String`) as `Document`

Description
Opens an existing mapping document (`*.mfd`). Adds the newly opened document to the end of the collection and makes it the active document.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

15.2.10.7 Parent

Property: Parent as Application (read-only)

Description
The parent object according to the object model.

Errors

1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

15.2.11 ErrorMarker

Represents a simple message line. In difference to AppOutputLine, error markers do not have a hierarchical structure.

Properties and Methods
Properties to navigate the object model:
Application
Parent

Access to message information:
DocumentFileName
ErrorLevel
Highlight
Serialization
Text

15.2.11.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

15.2.11.2 DocumentFileName

Property: `DocumentFileName` as `String` (read-only)

Description
Retrieves the name of the mapping file that the error marker is associated with.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

15.2.11.3 ErrorLevel

Property: `ErrorLevel` as `ENUMCodeGenErrorLevel` (read-only)

Description
Retrieves the severity of the error.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

15.2.11.4 Highlight

Method: `Highlight` ()

Description
Highlights the item that the error marker is associated with. If the corresponding document is not open, it will be opened.

Errors

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.

15.2.11.5 Serialization

Property: `Serialization` as `String` (read-only)

Description
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.

Errors

1900	The object is no longer valid.

1901 Invalid address for the return parameter was specified.

15.2.11.6 Text

Property: `Text` as `String` (read-only)

Description
Retrieves the message text.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.

15.2.11.7 Parent

Property: `Parent` as `ErrorMarkers` (read-only)

Description
The parent object according to the object model.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.

15.2.12 ErrorMarkers

Represents a collection of ErrorMarker objects.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Iterating through the collection:
`Count`
`Item`

15.2.12.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
1800 The object is no longer valid.

1801 Invalid address for the return parameter was specified.

15.2.12.2 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of error markers in the collection.

Errors
1800 The object is no longer valid.
1801 Invalid address for the return parameter was specified.

15.2.12.3 Item

Property: `Item` (*`nIndex`* as `Integer`) as `ErrorMarker` (read-only)

Description
Retrieves the error marker at `nIndex` from the collection. Indices start with 1.

Errors
1800 The object is no longer valid.
1801 Invalid address for the return parameter was specified.

15.2.12.4 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors
1800 The object is no longer valid.
1801 Invalid address for the return parameter was specified.

15.2.13 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 and Methods
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`

15.2.13.1 Active

Property: `Active` as `Boolean`

Description
Use this property to query if the mapping view is the active view, or set this view to be the active one.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.13.2 ActiveMapping

Property: ActiveMapping as Mapping

Description
Gets or sets the currently active mapping in the document this MapForceView belongs to.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.13.3 ActiveMappingName

Property: ActiveMappingName as String

Description

Gets or sets the currently active mapping by name in the document this MapForceView belongs to.

Errors

1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

15.2.13.4 Application

Property: `Application` as `Application` (read-only)

Description

Retrieves the application's top-level object.

Errors

1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

15.2.13.5 HighlightMyConnections

Property: `HighlightMyConnections` as `Boolean`

Description

This property defines whether connections from the selected item only should be highlighed.

Errors

1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

15.2.13.6 HighlightMyConnectionsRecursivey

Property: `HighlightMyConnectionsRecursively` as `Boolean`

Description

This property defines if only the connections coming directly or indirectly from the selected item should be highlighed.

Errors

1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

15.2.13.7 InsertWSDLCall

Method: `InsertWSDLCall` (*`i_strWSDLFileName`* as `String`)

Description
Adds a new WSDL call component to the mapping.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.13.8 InsertXMLFile (obsolete)

Method: `InsertXMLFile` (*`i_strXMLFileName`* as `String`, *`i_strRootElement`* as `String`)

Description
obsolete
MapForceView.InsertXMLFile is obsolete. Use Mapping.InsertXMLFile instead.

Adds a new component to the mapping.

The component's internal structure is determined by the schema referenced in the specified XML file.

The second parameter defines the root element of this schema, if there is more than one candidate.
When passing an empty string as root element, the root element of the xml file will be used. Otherwise if more candidates are available, a Select Root Element dialog will pop up regardless if MapForce is visible or not.

The specified XML file is used as the input sample to evaluate the mapping.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.13.9 InsertXMLSchema (obsolete)

Method: `InsertXMLSchema` (*`i_strSchemaFileName`* as `String`, *`i_strRootElement`* as `String`)

Description
obsolete
MapForceView.InsertXMLSchema is obsolete. Use Mapping.InsertXMLSchema instead.

Adds a new component to the mapping.

The component's internal structure is determined by the specified schema file.

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 will pop up regardless if MapForce is visible or not.

No XML input sample is assigned to this component.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.13.10 InsertXMLSchemaWithSample (obsolete)

Method: `InsertXMLSchemaWithSample` (*`i_strSchemaFileName`* as `String`, *`i_strXMLSampleName`* as `String`, *`i_strRootElement`* as `String`)

Description
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.

Adds a new component to the mapping.

The component's internal structure is determined by the specified schema file.

The second parameter is stored as the XML input sample for mapping evaluation.

The third parameter defines the root element of this schema if there is more than one candidate. When passing an empty string as root element, the root element of the xml sample file will be used.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.13.11 Parent

Property: `Parent` as `Document` (read-only)

Description
The parent object according to the object model.

Errors

1300	The object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.13.12 ShowItemTypes

Property: `ShowItemTypes` as `Boolean`

Description
This property defines if types of items should be shown in the mapping diagram.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.13.13 ShowLibraryInFunctionHeader

Property: `ShowLibraryInFunctionHeader` as `Boolean`

Description
This property defines whether the name of the function library should be part of function names.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

15.2.14 Mapping

A Mapping object represents a mapping in a document, so the main mapping, or a local user-defined-function mapping.

Properties and Methods
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`

15.2.14.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.14.2 Components

Property: `Components` as `Components` (read-only)

Description
Returns a collection of all components in the current mapping.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.14.3 CreateConnection

Method: CreateConnection(DatapointFrom as Datapoint, DatapointTo as Datapoint) as Connection

Description
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.

Errors

1200	The application 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.

15.2.14.4 InsertFunctionCall

Method: InsertFunctionCall(strFunctionName as String, strLibraryName as String) as Component

Description

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.

Errors

1200	The application 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.

15.2.14.5 InsertXMLFile

Method: InsertXMLFile(i_strFileName as String, i_strSchemaFileName as String) as Component

Description
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.

Errors

1200	The application 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.

15.2.14.6 InsertXMLSchema

Method: InsertXMLSchema(i_strSchemaFileName as String, i_strXMLRootName as String) as Component

Description
Adds a new XML schema component to the mapping.

The component's internal structure is determined the specified schema file.

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 will pop up if MapForce is visible. If MapForce is invisible, no dialog will pop up and only an error is returned.

No XML input sample is assigned to this component.

Errors

1200	The application 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.

15.2.14.7 *InsertXMLSchemaInputParameter*

Method: InsertXMLSchemaInputParameter(strParamName as String, strSchemaFileName as String, strXMLRootElementName as String) as Component

Description
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.

strParamName is the name of the input parameter component to create and **strSchemaFileName** and **strXMLRootElementName** indicate the respective schema file and the root element of the schema file to be used.

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 will pop up and only an error is returned.

Errors

1200	The application 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.

15.2.14.8 *InsertXMLSchemaOutputParameter*

Method: InsertXMLSchemaOutputParameter(strParamName as String, strSchemaFileName as String, strXMLRootElementName as String) as Component

Description
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.

strParamName is the name of the output parameter component to create and **strSchemaFileName** and **strXMLRootElementName** indicate the schema file and the root element of the schema file to be used respectively.

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 will pop up and only an error is returned.

Errors

1200	The application 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.

15.2.14.9 IsMainMapping

Property: `IsMainMapping` as `Boolean` (read-only)

Description

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).

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.14.10 Name

Property: `Name` as `String` (read-only)

Description

The name of the mapping / user defined function (UDF).

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.14.11 Parent

Property: `Parent` as `Document` (read-only)

Description

The parent object according to the object model.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.15 Mappings

Represents a collection of Mapping objects.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Iterating through the collection:
`Count`
`Item`

15.2.15.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.15.2 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of mappings in the collection.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.15.3 Item

Property: `Item` (*`nIndex`* as `Integer`) as `Mapping` (read-only)

Description
Retrieves the mapping at `nIndex` from the collection. Indices start with 1.

Errors

1200	The object is no longer valid.

1201 Invalid address for the return parameter was specified.

15.2.15.4 Parent

Property: `Parent` as `Document` (read-only)

Description
The parent object according to the object model.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

15.2.16 Options

This object gives access to all MapForce options available in the **Tools | Options** dialog.

Properties and Methods
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`

15.2.16.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

15.2.16.2 CodeDefaultOutputDirectory

Property: `CodeDefaultOutputDirectory` as `String`

Description

Specifies the target directory where files generated by `Document.GenerateCppCode`, `Document.GenerateJavaCode` and `Document.GenerateCHashCode`, are placed.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

15.2.16.3 CPPSettings_DOMType

Property: `CPPSettings_DOMType` as `ENUMDOMType`

Description

Specifies the DOM type used by `Document.GenerateCppCode`.

Errors

1400 The application 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

15.2.16.4 CPPSettings_GenerateVC6ProjectFile

Property: `CPPSettings_GenerateVC6ProjectFile` as `Boolean`

Description

Specifies if VisualC++ 6.0 project files should be generated by `Document.GenerateCppCode`.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

15.2.16.5 CppSettings_GenerateVSProjectFile

Property: `CppSettings_GenerateVSProjectFile` as `ENUMProjectType`

Description

Specifies the version of Visual Studio in which project files should be generated by `Document.GenerateCppCode`.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.
1402 The parameter value is out of range
1403 The paramater value is not available anymore

15.2.16.6 CPPSettings_LibraryType

Property: `CPPSettings_LibraryType` as `ENUMLibType`

Description

Specifies the library type used by `Document.GenerateCppCode`.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

15.2.16.7 CPPSettings_UseMFC

Property: `CPPSettings_UseMFC` as `Boolean`

Description

Specifies if MFC support should be used by C++ code generated by `Document.GenerateCppCode`.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

15.2.16.8 CSharpSettings_ProjectType

Property: `CSharpSettings_ProjectType` as `ENUMProjectType`

Description

Specifies the type of C# project used by `Document.GenerateCHashCode`.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.
1402 The parameter value is out of range
1403 The paramater value is not available anymore

15.2.16.9 DefaultOutputByteOrder

Property: `DefaultOutputByteOrder` as `String`

Description
Byte order for the file encoding used for output files.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.10 DefaultOutputByteOrderMark

Property: `DefaultOutputByteOrderMark` as `Boolean`

Description
Indicates if a byte order mark (BOM), is to be included in the file encoding of output files.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.11 DefaultOutputEncoding

Property: `DefaultOutputEncoding` as `String`

Description
File encoding used for output files.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.12 GenerateWrapperClasses

Property: `GenerateWrapperClasses` as `Boolean`

Description
Indicates if wrapper classes are also to be generated when generating code.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.13 JavaSettings_ApacheAxisVersion (obsolete)

Property: `JavaSettings_ApacheAxisVersion` as `ENUMApacheAxisVersion`

Description
Specifies the Apache Axis version to use when generating Java code for web service implementations with SOAP 1.1.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.14 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors

1400	The object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.15 ShowLogoOnPrint

Property: `ShowLogoOnPrint` as `Boolean`

Description
Show or hide the MapForce logo on printed outputs.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.16 ShowLogoOnStartup

Property: `ShowLogoOnStartup` as `Boolean`

Description
Show or hide the MapForce logo on application startup.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.17 UseGradientBackground

Property: `UseGradientBackground` as `Boolean`

Description
Set or retrieve the background color mode for a mapping window.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.16.18 XSLTDefaultOutputDirectory

Property: `XSLTDefaultOutputDirectory` as `String`

Description
Specifies the target directory where files generated by `Document.GenerateXSLT` are placed.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

15.2.17 Project

A Project object represents a project and its tree of project items in MapForce.

Events
Events

Properties and Methods
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 Support. For operations with Web services, the MapForce Enterprise edition is required.

15.2.17.1 Events

This object supports the following events:

`OnProjectClosed`

OnProjectClosed

Event: `OnProjectClosed` (*`i_objProject`* as `Project`)

Description
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`.

15.2.17.2 _NewEnum

Property: `_NewEnum` () as `IUnknown` (read-only)

Description
This property supports language-specific standard enumeration.

Errors

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(); objProjectIter.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) );
    }
}
```

15.2.17.3 AddActiveFile

Method: `AddActiveFile` () as `ProjectItem`

Description

Adds the currently open document to the mapping folder of the project's root.

Errors

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.

15.2.17.4 AddFile

Method: `AddFile` (*`i_strFileName`* as `String`) as `ProjectItem`

Description
Adds the specified document to the mapping folder of the project's root.

Errors
1500 The object is no longer valid.
1501 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.

15.2.17.5 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the top-level application object.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

15.2.17.6 Close

Method: `Close` ()

Description
Closes the project without saving.

Errors
1500 The object is no longer valid.

15.2.17.7 Count

Property: `Count` as `Integer` (read-only)

Description

Retrieves number of children of the project's root item.

Errors

1500 The object is no longer valid.

Examples

See `Item` or `_NewEnum`.

15.2.17.8 CreateFolder

Method: `CreateFolder` (*`i_strFolderName`* as `String`) as `ProjectItem`

Description

Creates a new folder as a child of the project's root item.

Errors

1500 The object is no longer valid.
1501 Invalid folder name or invalid address for the return parameter was specified.

15.2.17.9 FullName

Property: `FullName` as `String` (read-only)

Description

Path and name of the project file.

Errors

1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

15.2.17.10 GenerateCode

Method: `GenerateCode` ()

Description

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.

Errors

1500 The object is no longer valid.
1706 Error during code generation

15.2.17.11 GenerateCodeEx

Method: GenerateCode () as ErrorMarkers

Description
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.

Errors

1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.
1706 Error during code generation

15.2.17.12 GenerateCodeIn

Method: GenerateCodeIn (*i_nLanguage* as ENUMProgrammingLanguage)

Description
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.

Errors

1500 The object is no longer valid.
1706 Error during code generation

15.2.17.13 GenerateCodeInEx

Method: GenerateCodeIn (*i_nLanguage* as ENUMProgrammingLanguage) as ErrorMarkers

Description
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. 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.

Errors

1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.
1706 Error during code generation

15.2.17.14 InsertWebService

Method: `InsertWebService` (*`i_strWSDLFile`* as `String`, *`i_strService`* as `String`, *`i_strPort`* as `String`, *`i_bGenerateMappings`* as `Boolean`) as `ProjectItem`

Description
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.

Errors

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.

15.2.17.15 Item

Property: `Item(`*`i_nItemIndex`* as `Integer)` as `ProjectItem` (read-only)

Description
Returns the child at `i_nItemIndex` position of the project's root. The index is zero-based. The largest valid index is `Count-1`. For an alternative to visit all children see `_NewEnum`.

Errors

1500	The object is no longer valid.

Examples

```
// ----------------------------------------------------------
// JScript code snippet - enumerate children using Count and Item.
for( nItemIndex = 0; nItemIndex < objProject.Count; nItemIndex++ )
{
    objProjectItem = objProject.Item(nItemIndex);
    // do something with project item here
}
```

15.2.17.16 Java_BasePackageName

Property: `Java_BasePackageName` as `String`

Description
Sets or gets the base package name of the Java packages that will be generated. This property is used only when generating Java code.

Errors

1500 The object is no longer valid.
1501 Invalid package name specified.
Invalid address for the return parameter was specified.

15.2.17.17 Name

Property: `Name` as `String` (read-only)

Description
Name of the project file without file path.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

15.2.17.18 Output_Folder

Property: `Output_Folder` as `String`

Description
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.

Errors
1500 The object is no longer valid.
1501 Invalid folder name specified.
Invalid address for the return parameter was specified.

15.2.17.19 Output_Language

Property: `Output_Language` as `ENUMProgrammingLanguage`

Description
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.

Errors
1500 The object is no longer valid.
1501 Invalid language specified.
Invalid address for the return parameter was specified.

15.2.17.20 Output_TextEncoding

Property: `Output_TextEncoding` as `String`

Description
Sets or gets the text encoding used when generating XML-based code.

Errors
1500 The object is no longer valid.
1501 Invalid text encoding specified.
Invalid address for the return parameter was specified.

15.2.17.21 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

15.2.17.22 Path

Property: `Path` as String (read-only)

Description
Path of the project file without name.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

15.2.17.23 Save

Method: `Save` ()

Description
Saves the project to the file defined by `FullName`.

Errors
1500 The object is no longer valid.
1502 Can't save to file.

15.2.17.24 Saved

Property: `Saved` as Boolean (read-only)

Description
True if the project was not modified since the last Save operation, false otherwise.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

15.2.18 ProjectItem

A ProjectItem object represents one item in a project tree.

Properties and Methods
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 Support. For operations with Web services, the MapForce Enterprise edition is required.

15.2.18.1 _NewEnum

Property: `_NewEnum` () as `IUnknown` (read-only)

Description
This property supports language specific standard enumeration.

Errors

1700	The object is no longer valid.

Examples
See `Project.Item` or `Project._NewEnum`.

15.2.18.2 AddActiveFile

Method: `AddActiveFile` () as `ProjectItem`

Description
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.

Errors

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.

15.2.18.3 AddFile

Method: `AddFile` (*`i_strFileName`* as `String`) as `ProjectItem`

Description
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.

Errors

1700 The object is no longer valid.
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.

15.2.18.4 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the top-level application object.

Errors
1700 The object is no longer valid.
1701 Invalid address for the return parameter was specified.

15.2.18.5 CodeGenSettings_Language

Property: `CodeGenSettings_Language` as `ENUMProgrammingLanguage`

Description
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.

Errors
1700 The object is no longer valid.
1701 Invalid language or invalid address for the return parameter was specified.

15.2.18.6 CodeGenSettings_OutputFolder

Property: `CodeGenSettings_OutputFolder` as `String`

Description
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.

Errors
1700 The object is no longer valid.
1701 An invalid output folder or an invalid address for the return parameter was specified.

15.2.18.7 CodeGenSettings_UseDefault

Property: `CodeGenSettings_UseDefault` as `Boolean`

Description
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..

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

15.2.18.8 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves number of children of this project item. Also see `Item`.

Errors

1700	The object is no longer valid.

Examples
See `Project.Item` or `Project._NewEnum`.

15.2.18.9 CreateFolder

Method: `CreateFolder` (*`i_strFolderName`* as `String`) as `ProjectItem`

Description
Creates a new folder as a child of this project item.

Errors

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.

15.2.18.10 CreateMappingForProject

Method: `CreateMappingForProject` (*`i_strFileName`* as `String`) as `ProjectItem`

Description
Creates an initial mapping document for a Web service operation and saves it to `i_strFileName`. When using `Project.InsertWebService` you can use the `i_bGenerateMappings` flag to let MapForce automatically generate initial mappings for all ports.

Errors

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.

15.2.18.11 GenerateCode

Method: `GenerateCode` ()

Description
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.

Errors

1700	The object is no longer valid.
1706	Error during code generation.

15.2.18.12 GenerateCodeEx

Method: `GenerateCode` () `as` `ErrorMarkers`

Description
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.

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.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1706	Error during code generation.

15.2.18.13 GenerateCodeIn

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`)

Description
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.

Errors

1700	The object is no longer valid.
1701	Invalid language specified.
1706	Error during code generation.

15.2.18.14 GenerateCodeInEx

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`) `as` `ErrorMarkers`

Description
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.

Errors

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.

15.2.18.15 Item

Property: `Item(`*`i_nItemIndex`* as `Integer)` as `ProjectItem` (read-only)

Description
Returns the child at `i_nItemIndex` position of this project item. The index is zero-based. The largest valid index is `Count` - 1.
For an alternative to visit all children see `_NewEnum`.

Errors

1700	The object is no longer valid.

Examples
See `Project.Item` or `Project._NewEnum`.

15.2.18.16 Kind

Property: `Kind` as `ENUMProjectItemType` (read-only)

Description
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.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

15.2.18.17 Name

Property: `Name` as `String`

Description
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.

Errors

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.

15.2.18.18 Open

Method: `Open` () as `Document`

Description
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.

Errors

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.

15.2.18.19 Parent

Property: `Parent` as `Project` (read-only)

Description
Retrieves the project that this item is a child of. Has the same effect as `Application.ActiveProject`.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

15.2.18.20 QualifiedName

Property: `QualifiedName` as `String` (read-only)

Description
Retrieves the qualified name of a Web service item.

Errors

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.

15.2.18.21 Remove

Method: `Remove` ()

Description
Remove this project item and all its children from the project tree.

Errors

1700	The object is no longer valid.

15.2.18.22 WSDLFile

Property: `WSDLFile` as `String` (read-only)

Description
Retrieves the file name of the WSDL file defining the Web service that hosts the current project item.

Errors

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.

15.3 Enumerations

This is a list of all enumerations used by the MapForce API. If your scripting environment does not support enumerations, use the number-values instead.

15.3.1 ENUMApacheAxisVersion (obsolete)

Description
Enumeration values to select the Apache Axis version.

Possible values:

eApacheAxisVersion_Axis	= 1
eApacheAxisVersion_Axis2	= 2

15.3.2 ENUMApplicationStatus

Description
Enumeration values to indicate the status of the application.

Possible values:

eApplicationRunning	= 0
eApplicationAfterLicenseCheck	= 1
eApplicationBeforeLicenseCheck	= 2
eApplicationConcurrentLicenseCheckFailed	= 3

15.3.3 ENUMAppOutputLine_Severity

Description
Enumeration values to identify the severity of an AppOutputLine.

Possible values:

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

15.3.4 ENUMAppOutputLine_TextDecoration

Description

Enumeration values for the different kinds of text decoration of an AppOutputLine.

Possible values:

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

15.3.5 ENUMCodeGenErrorLevel

Description

Enumeration values to identify severity of code generation messages.

Possible values:

eCodeGenErrorLevel_Information	= 0
eCodeGenErrorLevel_Warning	= 1
eCodeGenErrorLevel_Error	= 2
eCodeGenErrorLevel_Undefined	= 3

15.3.6 ENUMComponentDatapointSide

Description

Enumeration values to indicate the side of a datapoint on its component.

Possible values:

eDatapointSideInput	= 0
eDatapointSideOutput	= 1

See also

GetRootDatapoint

15.3.7 ENUMComponentSubType

Description
Enumeration values to indicate component sub types.

Possible values:

eComponentSubType_None	= 0
eComponentSubType_Text_EDI	= 1
eComponentSubType_Text_Flex	= 2
eComponentSubType_Text_CSVFLF	= 3

15.3.8 ENUMComponentType

Description
Enumeration values to indicate component types.

Possible values:

eComponentType_Unknown	= 0
eComponentType_XML	= 1
eComponentType_DB	= 2
eComponentType_Text	= 3
eComponentType_Excel	= 4
eComponentType_WSDL	= 5
eComponentType_XBRL	= 6

15.3.9 ENUMComponentUsageKind

Description
Enumeration values to indicate component usage kind.

Possible values:

eComponentUsageKind_Unknown	= 0
eComponentUsageKind_Instance	= 1
eComponentUsageKind_Input	= 2
eComponentUsageKind_Output	= 3
eComponentUsageKind_Variable	= 4

15.3.10 ENUMConnectionType

Description
Enumeration values to indicate the type of a connection.

Possible values:

eConnectionTypeTargetDriven	= 0
eConnectionTypeSourceDriven	= 1

eConnectionTypeCopyAll	= 2

See also
ConnectionType

15.3.11 ENUMDOMType

Description
Enumeration values to specify the DOM type used by generated C++ mapping code.

Possible values:

eDOMType_xerces3	= 2
eDOMType_msxml6	= 3

Obsolete values

eDOMType_msxml4	= 0
eDOMType_xerces	= 1

Obsolete in this context means that this value is not supported and should not be used.

`eDOMType_xerces3` indicates Xerces 3.x usage.

15.3.12 ENUMLibType

Description
Enumeration values to specify the library type used by the generated C++ mapping code.

Possible values:

eLibType_static	= 0
eLibType_dll	= 1

15.3.13 ENUMProgrammingLanguage

Description
Enumeration values to select a programming language.

Possible values:

eUndefinedLanguage	= -1
eJava	= 0
eCpp	= 1
eCSharp	= 2
eXSLT	= 3
eXSLT2	= 4
eXQuery	= 5

15.3.14 ENUMProjectItemType

WDescription
Enumeration to identify the different kinds of project items that can be children of Project or folder-like ProjectItems.

Possible values:

eProjectItemType_Invalid	= -1
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_LibrarzFolder	= 8
eProjectItemType_ResourceFolder	= 9
eProjectItemType_VirtualFolder	= 10

See also
`ProjectItem.Kind`

15.3.15 ENUMProjectType

Description
Enumeration values to select a project type for generated C# and C++ mapping code.

Possible values:	eVisualStudio2008Project	= 5
	eVisualStudio2010Project	= 6
	eVisualStudio2013Project	= 7
	eVisualStudio2015Project	= 8
	eVisualStudio2017Project	= 9
Obsolete values	eVisualStudioProject	= 0
	eVisualStudio2003Project	= 1
	eBorlandProject	= 2
	eVisualStudio2005Project	= 4

Obsolete in this context means that this value is not supported and should not be used.

15.3.16 ENUMSearchDatapointFlags

Description
Enumeration values used as bit-flags; to be used as combination of flags when searching for a datapoint.

Possible values:

eSearchDatapointElement	= 1
eSearchDatapointAttribute	= 2

See also
GetChild

15.3.17 ENUMViewMode

Description
Enumeration values to select a MapForce view.

Possible values:

eMapForceView	= 0
eXSLView	= 1
eOutputView	= 2

Chapter 16

ActiveX Integration

16 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).

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 and Integration at Document Level describe the key steps at these respective levels. The ActiveX Integration Examples section provides examples in C#, HTML, and Java. Looking through these examples will help you to make the right decisions quickly. The Object Reference 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.

16.1 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 2008/2010/2012/2013/2015/2017
- 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).

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.

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.

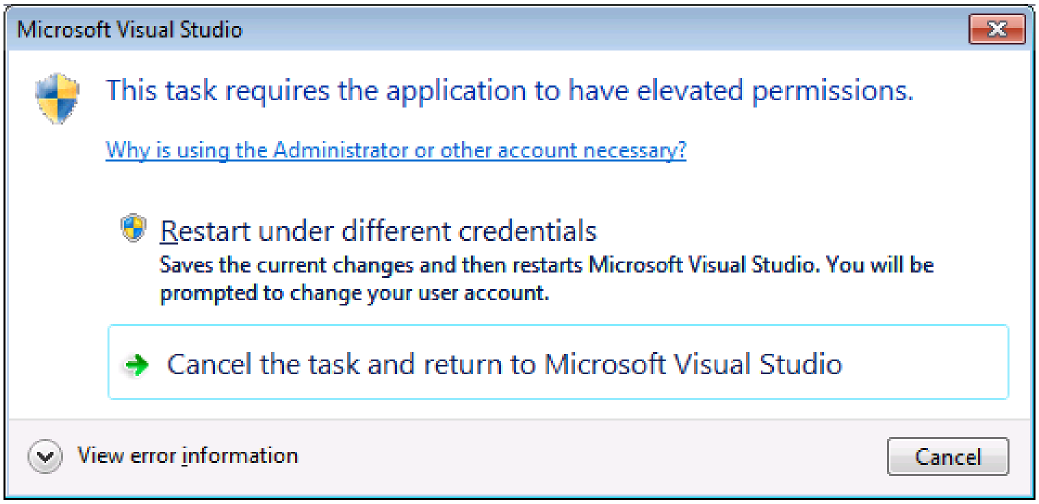
16.2 Adding the ActiveX Controls to the Toolbox

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.
2. If prompted to restart Visual Studio with elevated permissions, click **Restart under different credentials**.



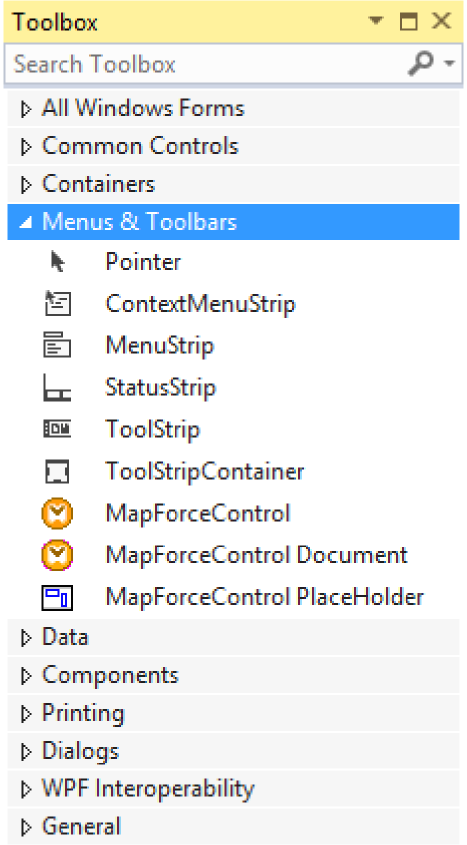
This task requires the application to have elevated permissions.

Why is using the Administrator or other account necessary?

Restart under different credentials
Saves the current changes and then restarts Microsoft Visual Studio. You will be prompted to change your user account.

Cancel the task and return to Microsoft Visual Studio

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). The **MapForceControl Document** and **MapForceControl Placeholder** controls are used for document-level integration (see Integration at Document Level).

16.3 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.

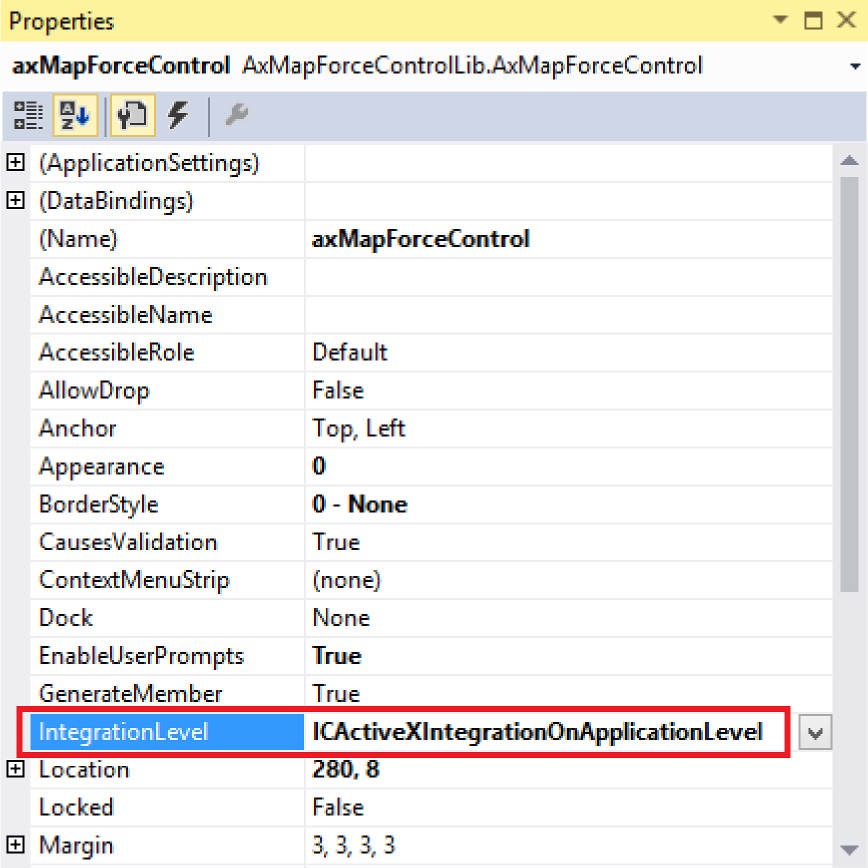
The only ActiveX control you need to integrate is `MapForceControl`. Do not instantiate or access `MapForceControlDocument` or `MapForceControlPlaceHolder` ActiveX controls when integrating at application-level.

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`. Consider using `MapForceControl.Application` for more complex access to MapForce functionality.

For an example that shows how the MapForce application can be embedded in an HTML page, see HTML Integration at Application Level.

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).
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).
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**.



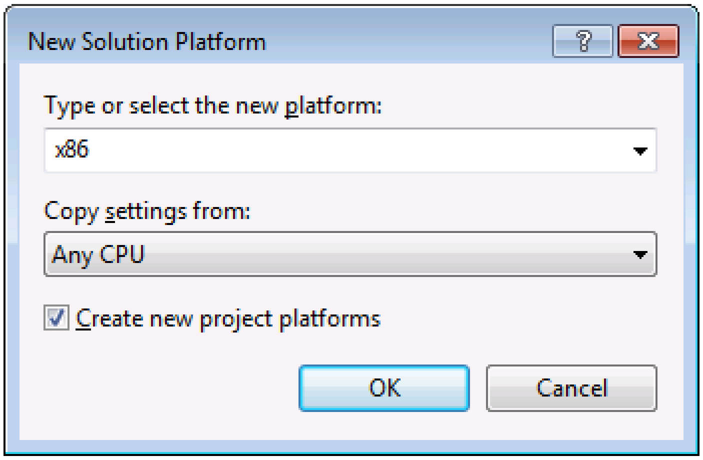
Properties

axMapForceControl AxMapForceControlLib.AxMapForceControl

⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	**axMapForceControl**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
Appearance	**0**
BorderStyle	**0 - None**
CausesValidation	True
ContextMenuStrip	(none)
Dock	None
EnableUserPrompts	**True**
GenerateMember	True
IntegrationLevel	**ICActiveXIntegrationOnApplicationLevel**
⊞ Location	**280, 8**
Locked	False
⊞ Margin	3, 3, 3, 3

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**).



New Solution Platform

Type or select the new platform: x86

Copy settings from: Any CPU

☑ Create new project platforms

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).

16.4 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, for an ActiveX integration at application level, only one control is required, namely the **MapForceControl**. However, for an ActiveX integration at document level, functionality MapForce is provided by the following ActiveX controls:

1. **MapForceControl**
2. **MapForceControl Document**
3. **MapForceControl Placeholder**

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).

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` 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.

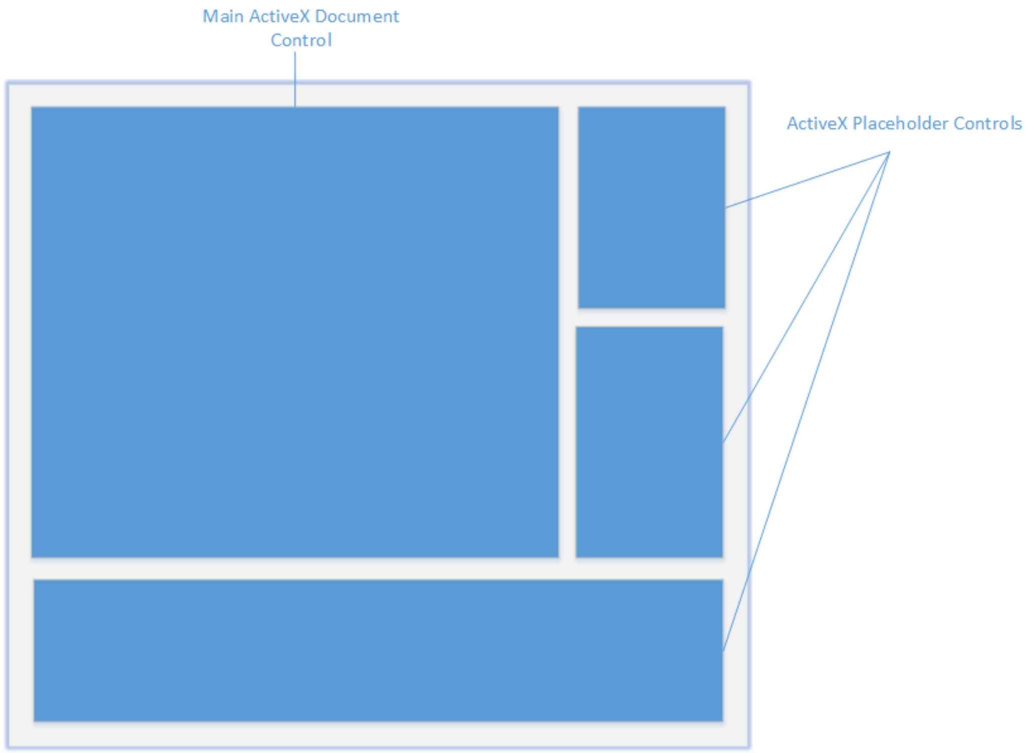
Instances of **MapForceControl PlaceHolder** allow you to selectively embed additional windows of MapForce into your application. The window kind (for example, Project window) is defined by the property **PlaceholderWindowID**. Therefore, to set the window kind, set the property **PlaceholderWindowID**. For valid window identifiers, see MapForceControlPlaceholderWindow.

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).
3. Drag the **MapForceControl** 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** 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** 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.
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` property of the **MapForceControl**.
- To retrieve commands organized according to their menu structure, use the `MainMenu` property.
- To retrieve commands organized by the toolbar in which they appear, use the `Toolbars` property.
- To send commands to MapForce, use the `Exec` method.
- To query if a command is currently enabled or disabled, use the `QueryStatus` 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 section.

General considerations

To automate the behaviour of MapForce, use the properties, methods, and events described for the **MapForceControl**, **MapForceControl Document**, and **MapForceControl Placeholder**.

For more complex access to MapForce functionality, consider using the following properties:

- `MapForceControl.Application`
- `MapForceControlDocument.Document`
- `MapForceControlPlaceHolder.Project`

These properties give you access to the MapForce automation interface (MapForceAPI)

Note: To open a document, always use `MapForceControlDocument.Open` or `MapForceControlDocument.New` on the appropriate document control. To open a project, always use `MapForceControlPlaceHolder.OpenProject` 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.

16.5 ActiveX Integration Examples

This section contains examples of MapForce document-level integration using different container environments and programming languages. (The HTML section additionally contains examples of integration at application level.) Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX` of your MapForce installation.

16.5.1 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).

16.5.1.1 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.

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 and several instances of MapForceControlPlaceHolder controls. Double-check that the following properties of these controls are set as shown in the table below:

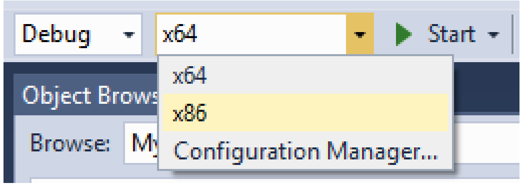
Control name	Property	Property value
axMapForceControl	IntegrationLevel	ICActiveXIntegrationOnDocumentLevel
axMapForceControlLibrary	PlaceholderWindowID	0
axMapForceControlOutput	PlaceholderWindo	2

Control name	Property	Property value
	wID	
axMapForceControlPreview	PlaceholderWindowID	1

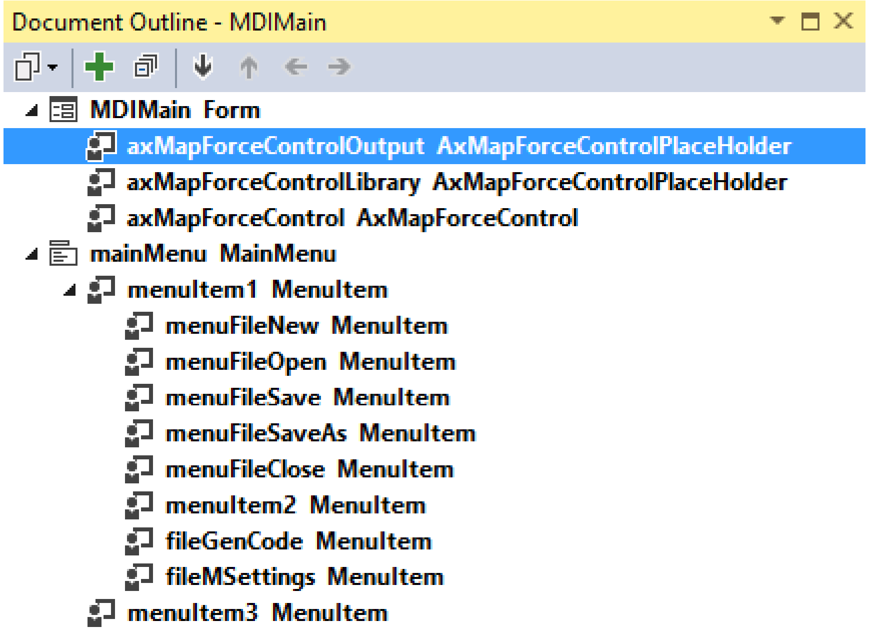
Here is how you can view or set the properties of an ActiveX control:

1. Open the **MDIMain.cs** form in the designer window.

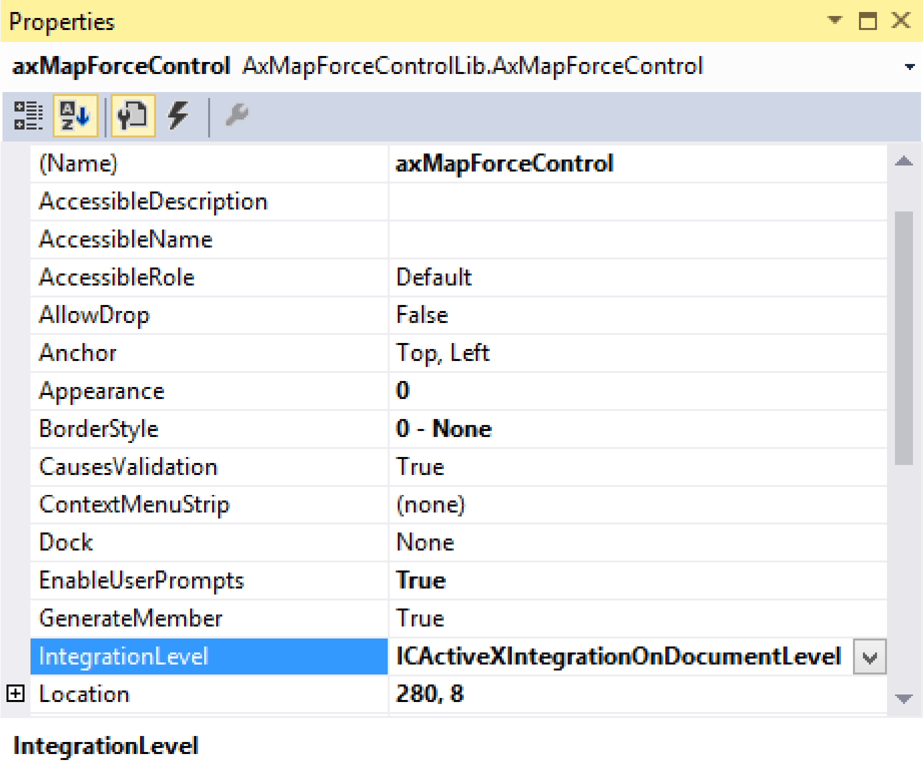
Note: On 64-bit Windows, it may be necessary to change the build configuration of the Visual Studio solution to "x86" **before** opening the designer window. If you need to build the sample as a 64-bit application, see Prerequisites.



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:



Properties

axMapForceControl AxMapForceControlLib.AxMapForceControl

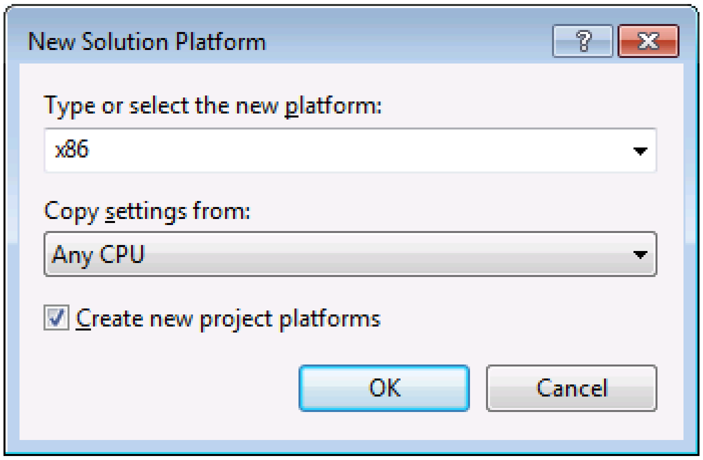
(Name)	**axMapForceControl**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
Appearance	**0**
BorderStyle	**0 - None**
CausesValidation	True
ContextMenuStrip	(none)
Dock	None
EnableUserPrompts	**True**
GenerateMember	True
IntegrationLevel	**ICActiveXIntegrationOnDocumentLevel**
⊞ Location	**280, 8**

IntegrationLevel

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**).



New Solution Platform

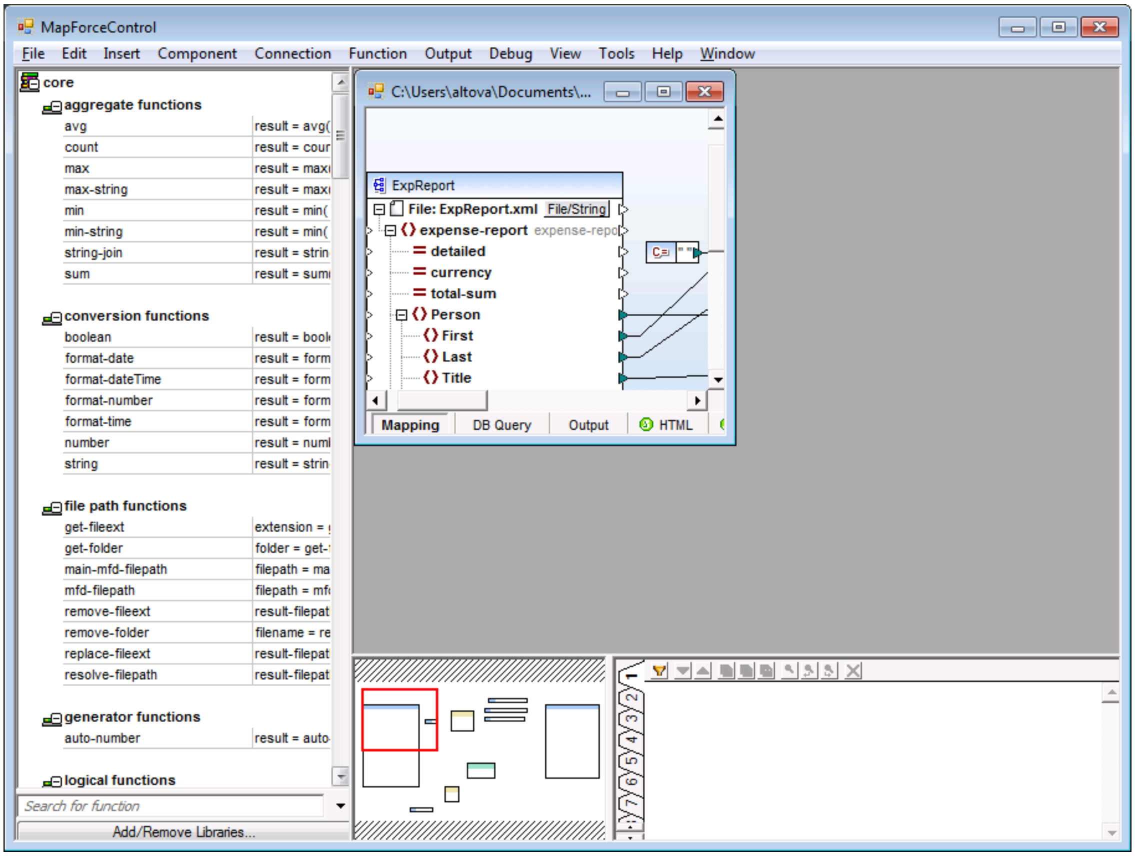
Type or select the new platform: x86

Copy settings from: Any CPU

☑ Create new project platforms

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:


MapForceControl
File Edit Insert Component Connection Function Output Debug View Tools Help Window
core
aggregate functions
avg result = avg(
count result = cour
max result = maxi
max-string result = maxi
min result = min(
min-string result = min(
string-join result = strin
sum result = sumi
conversion functions
boolean result = bool
format-date result = form
format-dateTime result = form
format-number result = form
format-time result = form
number result = numl
string result = strin
file path functions
get-fileext extension =
get-folder folder = get-
main-mfd-filepath filepath = ma
mfd-filepath filepath = mfi
remove-fileext result-filepat
remove-folder filename = re
replace-fileext result-filepat
resolve-filepath result-filepat
generator functions
auto-number result = auto
logical functions
Search for function
Add/Remove Libraries...
C:\Users\altova\Documents\...
ExpReport
File: ExpReport.xml File/String
expense-report expense-repo
detailed
currency
total-sum
Person
First
Last
Title
Mapping DB Query Output HTML

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.

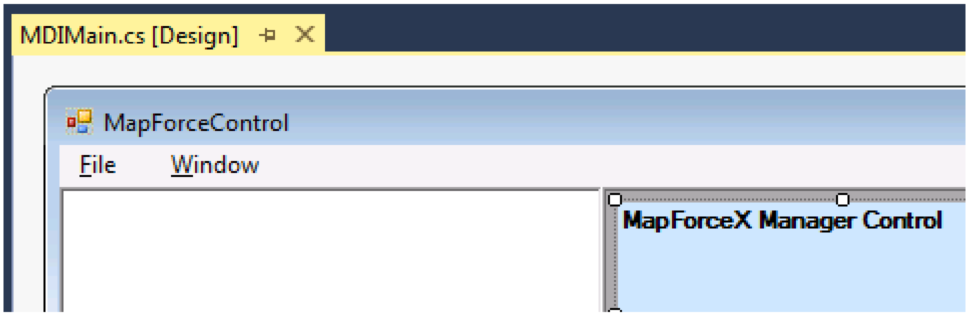
16.5.1.2 Retrieving Command Information

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);
  }
}
```

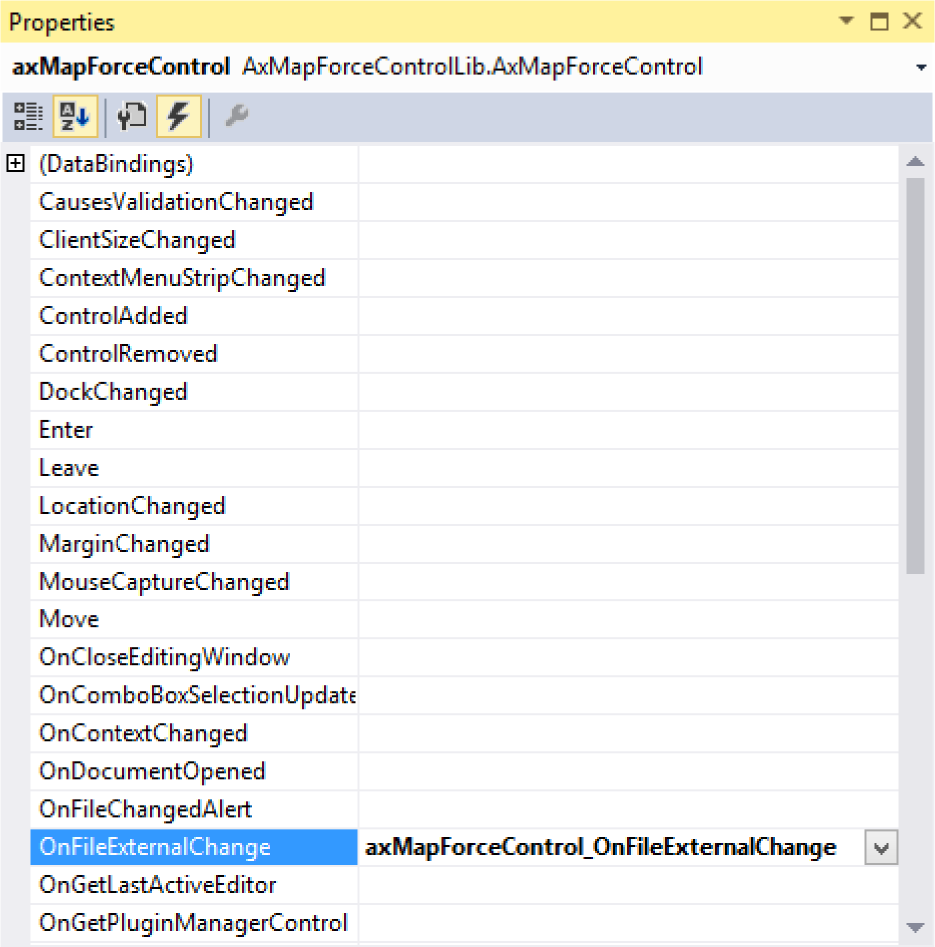
If the command is a container for other commands (that is, if it has child commands), it gets the event handler `AltovaSubMenu_Popup`. This handler queries the status of each child command and enables or disables it as required. This ensures that each command is enabled only when that is meaningful (for 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);
}
```

16.5.1.3 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:



(DataBindings)	
CausesValidationChanged	
ClientSizeChanged	
ContextMenuStripChanged	
ControlAdded	
ControlRemoved	
DockChanged	
Enter	
Leave	
LocationChanged	
MarginChanged	
MouseCaptureChanged	
Move	
OnCloseEditingWindow	
OnComboBoxSelectionUpdate	
OnContextChanged	
OnDocumentOpened	
OnFileChangedAlert	
OnFileExternalChange	axMapForceControl_OnFileExternalChange
OnGetLastActiveEditor	
OnGetPluginManagerControl	

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;
}
```

16.5.2 HTML

The code listings in this section show how to integrate the MapForceControl at application level and document level. Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX\HTML` of your MapForce installation.

Note: ActiveX controls in an HTML page are supported only by Internet Explorer when it runs as a 32-bit application. When Internet Explorer 10 or 11 runs in 64-bit mode, it does not load ActiveX controls. The default browser security settings will normally block ActiveX, so you will need to explicitly allow blocked content to run on the page when prompted by Internet Explorer.

16.5.2.1 HTML Integration at Application Level

This example shows a simple integration of the MapForce control at application-level into a HTML page. The integration is described in the following sections:

- Instantiate a MapForceControl in HTML code.
- Implement buttons to load documents and automate code-generation tasks.
- Define actions for some application events.

The code for this example is available at the following location in your MapForce installation: `<ApplicationFolder>\Examples\ActiveX\HTML\MapForceActiveX_ApplicationLevel.htm`.

Instantiate the Control

The HTML `Object` tag is used to create an instance of the MapForceControl. The `Classid` is that of MapForceControl. Width and height specify the window size. No additional parameters are necessary, since application-level is the default.

```
<OBJECT id="objMapForceControl"
        Classid="clsid:A38637E9-5759-4456-A167-F01160CC22C1"
        width="800"
        height="500"
        VIEWASTEXT>
</OBJECT>
```

Add Button to Open Default Document

As a simple example of how to automate some tasks, we add a button to the page:

```
<input type="button" value="Open Marketing Expenses"
onclick="BtnOpenMEFile()">
```

When clicked, a predefined document will be opened in the MapForceControl. The `MakeAbsolutePath` method creates an absolute path using the location of the script as a base path.

```
function BtnOpenMEFile()
{
   var strPath = MakeAbsolutePath("MarketingExpenses.mfd");
   var objDoc = objMapForceControl.Open(strPath );

   if (objDoc == null)
      alert("Unable to locate MarketingExpenses.mfd at: " +
objMapForceControl.BaseHref);
}
```

Add Buttons for Code Generation

Although code-generation for the active document is available via menus, we want to have buttons that will generate code without asking the user for the location of the output. The method is similar to that used in the previous section.

First come the buttons:

```
<input type="button" value="Generate XSLT" onclick="BtnGenerate( 0 )">
<input type="button" value="Generate Java" onclick="BtnGenerate( 1 )">
<input type="button" value="Generate C++" onclick="BtnGenerate( 2 )">
<input type="button" value="Generate C#" onclick="BtnGenerate( 3 )">
```

Then we provide the script that will generate the code into sub-folders of the currently defined default output folders.

```
// ---------------------------------------------------------------------
// generate code for active document into language-specific subfolders of
// the current default output dicrectory. No user intercation necessary.
function BtnGenerate(languageID)
{
   // get top-level object of automation interface
   var objApp = objMapForceControl.Application;

   // get the active document
   var objDocument = objApp.ActiveDocument;

   // retrive object to set the generation output path
   var objOptions = objApp.Options;

   if ( objDocument == null )
      alert( "no active document found" );
   else
   {
      objOptions.XSLTDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory = GetDefaultOutputDirectory();

      if (languageID == 0)
      {
         objOptions.XSLTDefaultOutputDirectory =
objOptions.XSLTDefaultOutputDirectory + "\\XSLTGen";
         objDocument .GenerateXSLT();
      }
      else if (languageID == 1)
      {
         objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/JavaCode";
         objDocument .GenerateJavaCode();
      }
      else if (languageID == 2)
      {
         objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/CPPCode";
         objDocument .GenerateCppCode();
      }
      else if (languageID == 3)
      {
         objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/CSharpCode";
         objDocument .GenerateCHashCode();
      }
   }
}
```

Connect to Custom Events

The example implements two event callbacks for MapForceControl custom events to show the principle:

```
<!-- ----------------------------------------------------------- -->
<!--  custom event 'OnDocumentOpened" of MapForceControl object  -->
<SCRIPT FOR="objMapForceControl" event="OnDocumentOpened( objDocument )"
LANGUAGE="javascript">
   // alert("Document '" + objDocument.Name + "' opend!");
</SCRIPT>

<!-- ----------------------------------------------------------- -->
<!--  custom event 'OnDocumentClosed" of MapForceControl object  -->
<SCRIPT FOR="objMapForceControl" event="OnDocumentClosed( objDocument )"
LANGUAGE="javascript">
   // alert("Document '" + objDocument.Name + "' closed!");
</SCRIPT>
```

16.5.2.2 HTML Integration at Document Level

This example shows an integration of the MapForce control at document-level into a HTML page. The following topics are covered:

- Instantiate a MapForceControl ActiveX control object in HTML code
- Instantiate a MapForceControlDocument ActiveX control to allow editing a MapForce file
- Instantiate one MapForceControlPlaceHolder for a MapForceControl project window
- Instantiate one MapForceControlPlaceHolder to alternatively host one of the MapForce helper windows
- Create a simple custom toolbar for some heavy-used MapForce commands
- Add some more buttons that use the COM automation interface of MapForce
- Use event handlers to update command buttons

This example is available in its entirety in the file `MapForceActiveX_ApplicationLevel.htm` within the `<ApplicationFolder>\Examples\ActiveX\HTML\` folder of your MapForce installation.

Instantiate the MapForceControl

MapForceControlThe HTML `OBJECT` tag is used to create an instance of the MapForceControl. The Classid is that of MapForceControl. Width and height are set to 0 since we use this control as manager control without use for its user interface. The integration level is specified as a parameter within the `OBJECT` tag.

```
<object id="objMapForceX" classid="clsid:A38637E9-5759-4456-A167-F01160CC22C1"
 width="0" height="0" VIEWASTEXT>
   <param name="IntegrationLevel" value="1">
</object>
```

Create Editor Window

The HTML `OBJECT` tag is used to embed an editing window. The additional custom parameter specifies that the control is to be initialized with a new empty document.

```
<object id="objDoc1" classid="clsid:DFBB0871-DAFE-4502-BB66-08CEB7DF5255"
width="600" height="500" VIEWASTEXT>
   <param name="NewDocument">
</object>
```

Create Project Window

The HTML `OBJECT` tag is used to create a MapForceControlPlaceHolder window. The parameter defines the placeholder to show the MapForce project window.

```
<!-- --------------------------------------------------- -->
<!-- create project window placeholder control.          -->
<!-- initialize it with a project.                       -->
<object id="objProjectWindow" classid="clsid:FDEC3B04-05F2-427d-988C-
F03A85DE53C2" width="200" height="200" VIEWASTEXT>
   <param name="PlaceholderWindowID" value="3">
```

```
</object>
```

Create Placeholder for Helper Windows

The **MapForceControlPlaceHolder** control is required to host an application helper window, see also [Integration at Document Level](). In the code listing below, the HTML `object` tag is used to instantiate a control that will host the **Libraries** window by default (`PlaceholderWindowID` is 0).

```
<!-- create helper window placeholder control.          -->
<!-- the editor with focus will automatically direct its -->
<!-- output to the appropriate helper window.           -->
<object id="objPlaceholderWindow" Classid="clsid:FDEC3B04-05F2-427d-988C-
F03A85DE53C2" width="200" height="200" VIEWASTEXT>
   <param name="PlaceholderWindowID" value="0">
   <param name="FileName" value="">
</object>
```

The example HTML page includes a few buttons which call the `BtnHelperWindow` method when clicked. The `BtnHelperWindow` method reassigns the `PlaceholderWindowID` of the control, and thus cause the ActiveX object to display a different helper window.

```
// specify which of the helper windows shall be shown in the placeholder
control.
function BtnHelperWindow(i_ePlaceholderWindowID)
{
   objPlaceholderWindow.PlaceholderWindowID = i_ePlaceholderWindowID;
}
```

For the list of possible values of `PlaceholderWindowID`, see [`MapForceControlPlaceholderWindow`]().

Create a Custom Toolbar

The example HTML page also includes a custom toolbar (intended as a replica of the MapForce menu). The custom toolbar consists of buttons with images of MapForce commands, for example:

```
<button id="btnInsertXML" title="Insert XML Schema/File"
onclick="BtnDoCommand(32393)">
   <img src="..\Images\ID_INSERT_XSD.gif" width="16" height="16" />
</button>
<button id="btnInsertDB" title="Insert Database"
onclick="BtnDoCommand(32389)">
   <img src="..\Images\ID_INSERT_DATABASE.gif" width="16" height="16" />
</button>
<button id="btnInsertEDI" title="Insert EDI" onclick="BtnDoCommand(32390)">
   <img src="..\Images\ID_INSERT_EDI.gif" width="16" height="16" />
</button>
<button id="btnInsertText" title="Insert Text file"
```

```
onclick="BtnDoCommand(32392)">
   <img src="..\Images\ID_INSERT_TXT.gif" width="16" height="16" />
</button>
```

The names of button images correspond to the command ID numbers, see Command Reference. On clicking the button, the corresponding command ID is sent to the main control and executed:

```
// perform any command specified by cmdID.
// command routing includes application, active document and view.
function BtnDoCommand(cmdID)
{
   objMapForceX.Exec( cmdID );
   msgtext.innerText = "Command " + cmdID + " performed.";
}
```

Create More Buttons

In the example, we add some more buttons to show some automation code.

```
<!-- add some buttons associated with above editor.               -->
<!-- generation of code is now implemented using the MapForce automation  -->
<!-- interface to select a target folder without prompting the user.      -->
<p>
   <input type="button" value="New File" onclick="BtnNewFile(objDoc1)">
   <input type="button" value="Save File" onclick="BtnSaveFile(objDoc1)">
   <input type="text" title="Path" id="strPath" width="150">
   <input type="button" value="Open MarketingExpenses"
onclick="BtnOpenMEFile(objDoc1)">
   <input type="button" value="Open MapForce Sample Project"
onclick="BtnOpenProjectFile(objDoc1)">
</p>
```

The corresponding JavaScript looks like this:

```
// -------------------------------------------------------------------
// open a new empty document in the specified document control window.
function BtnNewFile(objDocCtrl)
{
   objDocCtrl.OpenDocument("");
   objDocCtrl.setActive();
}

// -------------------------------------------------------------------
// Saves the current file in the specified document control window.
function BtnSaveFile(objDocCtrl)
{
   if(objDocCtrl.Path.length > 0)
objDocCtrl.SaveDocument();
   else
   {
if(strPath.value.length > 0)
{
   objDocCtrl.Path = strPath.value;
   objDocCtrl.SaveDocument();
}
else
{
   alert("Please set path for the document first!");
   strPath.focus();
}
   }

   objDocCtrl.setActive();
}

// ---------------------------------------------------------
// open a document in the specified document control window.
function BtnOpenMEFile(objDocCtrl)
{
   // do not use MapForceX.Application.OpenDocument(...) to open a document,
   // since then MapForceControl wouldn't know a control window to show
   // the document in. Instead:

   var strPath = MakeAbsolutePath("MarketingExpenses.mfd");
   var objDoc = objDocCtrl.OpenDocument(strPath);

   if (objDoc != null)
   {
objDocCtrl.setActive();
msgtext.innerText = "Opened mapping: " + strPath;
   }
   else
alert("Unable to open " + strPath);
}
```

Create Event Handler to Update Button Status

Availability of a command may vary with every mouse click or keystroke. The custom event `OnUpdateCmdUI` of MapForceControl gives us an opportunity to update the enabled/disabled state of buttons associated with MapForce commands. The method `MapForceControl.QueryStatus` is used to query whether a command is enabled or not.

```
<!--  custom event 'OnUpdateCmdUI" of MapForceControl object  -->
function objMapForceX::OnUpdateCmdUI()
{
   if ( document.readyState == "complete" )// 'complete'
   {
      // update status of buttons
      // set activity status of simulated toolbar
      GenerateXSLT.disabled =  ! (objDoc1.QueryStatus(32360) & 0x02);   // not
enabled
      GenerateJava.disabled =  ! (objDoc1.QueryStatus(32358) & 0x02);   // not
enabled
      GenerateCpp.disabled =   ! (objDoc1.QueryStatus(32356) & 0x02);   // not
enabled
      GenerateCSharp.disabled = ! (objDoc1.QueryStatus(32357) & 0x02);   //
not enabled

      btnInsertXML.disabled = ! (objDoc1.QueryStatus(32393) & 0x02);
      btnInsertDB.disabled = ! (objDoc1.QueryStatus(32389) & 0x02);
      btnInsertEDI.disabled = ! (objDoc1.QueryStatus(32390) & 0x02);
      btnInsertText.disabled = ! (objDoc1.QueryStatus(32392) & 0x02);

      btnInsertConstant.disabled = ! (objDoc1.QueryStatus(32388) & 0x02);
      btnInsertFilter.disabled = ! (objDoc1.QueryStatus(32391) & 0x02);
      btnInsertIFELSE.disabled = ! (objDoc1.QueryStatus(32394) & 0x02);
      btnInsertException.disabled = ! (objDoc1.QueryStatus(32311) & 0x02);

      btnFuncUserDef.disabled = ! (objDoc1.QueryStatus(32380) & 0x02);
      btnFuncUserDefSel.disabled = ! (objDoc1.QueryStatus(32381) & 0x02);
      btnFuncSettings.disabled = ! (objDoc1.QueryStatus(32387) & 0x02);
      btnInsertInput.disabled = ! (objDoc1.QueryStatus(32383) & 0x02);

      btnGenXSLT.disabled = ! (objDoc1.QueryStatus(32360) & 0x02);
      btnGenXSLT2.disabled = ! (objDoc1.QueryStatus(32361) & 0x02);
      btnGenXQuery.disabled = ! (objDoc1.QueryStatus(32359) & 0x02);
      btnGenCPP.disabled = ! (objDoc1.QueryStatus(32356) & 0x02);
      btnGenCSharp.disabled = ! (objDoc1.QueryStatus(32357) & 0x02);
      btnGenJava.disabled = ! (objDoc1.QueryStatus(32358) & 0x02);
   }
}
```

16.5.3 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).

- `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.

Rules for mapping the ActiveX Control names to Java

For the documentation of ActiveX controls, see Object Reference. 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:

- Example Java Project

- Creating the ActiveX Controls
- Loading Data in the Controls
- Basic Event Handling
- Menus
- UI Update Event Handling
- Creating a MapForce Mapping Table

16.5.3.1 *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: Starts MapForce, which is registered as an automation server, or activates MapForce if it is already running.
- Loading Data in the Controls: Locates one of the example documents installed with MapForce and opens it.
- Basic Event Handling: Changes the view of all open documents to Text View. The code also shows how to iterate through open documents.
- Menus: Validates the active document and shows the result in a message box. The code shows how to use output parameters.
- UI Update Event Handling: Shows how to handle MapForce events.
- Creating a MapForce Mapping Table: 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\\MapForce2018\\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).
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).
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**.

16.5.3.2 *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 ) );
```

16.5.3.3 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\\MapForce2018\\MapForceExamples\\";
  mapForceProjectToolWindow = new MapForceControlPlaceHolder( MapForceControlPlaceholderWindow.MapForceXProjectWindow.getValue(), strExamplesFolder + "MapForceExamples.mfp" ) ;
```

16.5.3.4 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();
    }
  }
}
```

16.5.3.5 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 Handling).

```

    // 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();
    }

  }
```

16.5.3.6 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;
    }
  }
```

16.5.3.7 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.

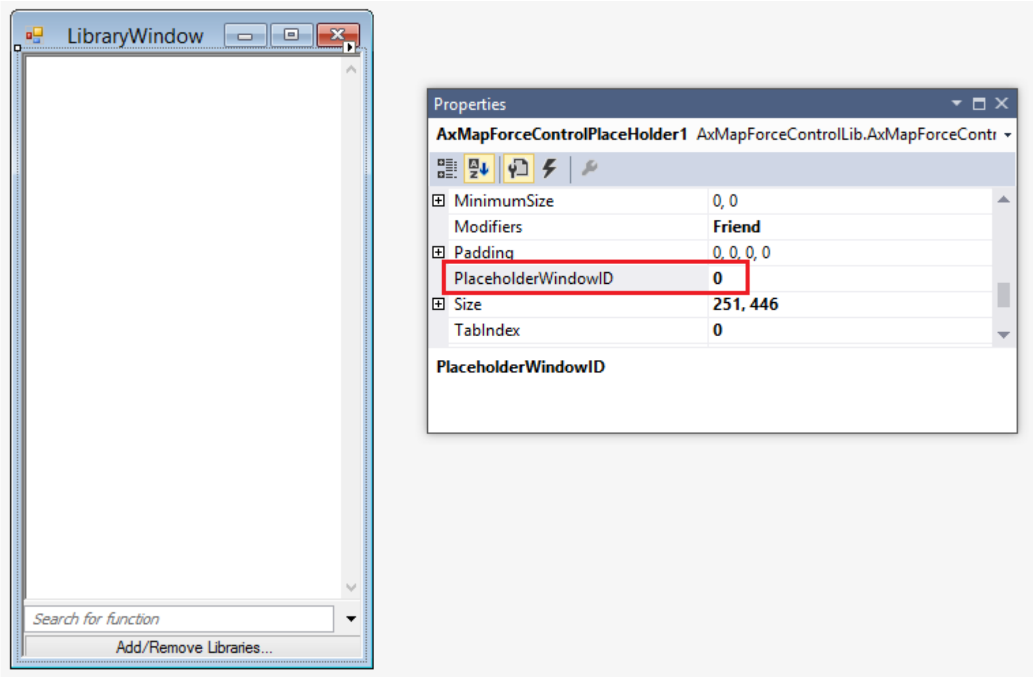
```
//access MapForce Java-COM bridge
import com.altova.automation.MapForce.*;
import com.altova.automation.MapForce.Component;
import com.altova.automation.MapForce.Enums.ENUMComponentUsageKind;

//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();
  }
 }

}
```

16.5.4 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).



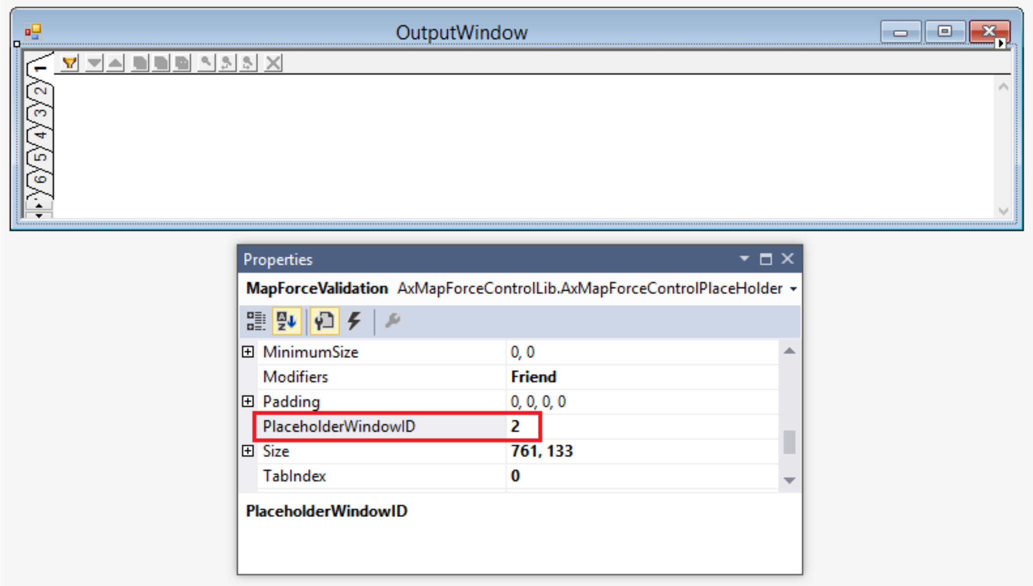
AxMapForceControlPlaceHolder1 AxMapForceControlLib.AxMapForceContr

MinimumSize	0, 0
Modifiers	Friend
Padding	0, 0, 0, 0
PlaceholderWindowID	0
Size	251, 446
TabIndex	0

PlaceholderWindowID

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).



MapForceValidation AxMapForceControlLib.AxMapForceControlPlaceHolder

⊞ MinimumSize	0, 0
Modifiers	**Friend**
⊞ Padding	0, 0, 0, 0
PlaceholderWindowID	**2**
⊞ Size	**761, 133**
TabIndex	**0**

PlaceholderWindowID

Before attempting to build and run this solution, note the following steps:

Step 1: Check the prerequisites

For the list of prerequisites, see Prerequisites.

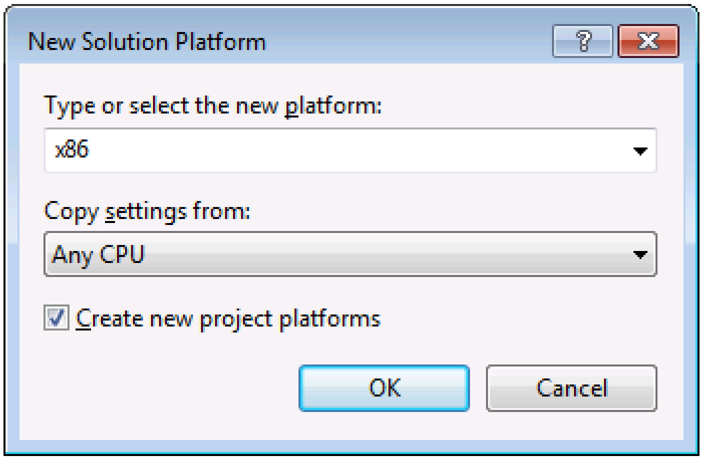
Step 2: Copy the sample to a directory where you have write permissions

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**).


New Solution Platform
?
Type or select the new platform:
x86
Copy settings from:
Any CPU
☑ Create new project platforms
OK
Cancel

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.

16.6 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` or the `MapForceControlDocument.Exec` methods. To query the status of a command, use the `MapForceControl.QueryStatus` or `MapForceControlDocument.QueryStatus` methods.

Depending on the edition of MapForce you have installed, some of these commands might not be supported.

16.6.1 "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_COD	32362

Menu item	Command name	ID
	E	
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

16.6.2 "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

16.6.3 "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

16.6.4 "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

16.6.5 "Component" Menu

The "Component" menu has the following commands:

Menu item	Command name	ID
Change Root Element...	ID_COMPONENT_CHANGEROOTELEM	32334

Menu item	Command name	ID
	ENT	
Edit Schema Definition in XMLSpy	ID_COMPONENT_EDIT_SCHEMA	32337
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

16.6.6 "Connection" Menu

The "Connection" menu has the following commands:

Menu item	Command name	ID
Auto Connect Matching Children	ID_CONNECTION_AUTOCONNECTCHILDREN	32342
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

16.6.7 "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

16.6.8 "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
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

16.6.9 "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

16.6.10 "View" Menu

The "View" menu has the following commands:

Menu item	Command name	ID
Show Annotations	ID_SHOW_ANNOTATION	32435
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

16.6.11 "Tools" Menu

The "Tools" menu has the following commands:

Menu item	Command name	ID
Global Resources	IDC_GLOBALRESOURCES	37401

Menu item	Command name	ID
<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

16.6.12 "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

16.6.13 "Help" Menu

The "Help" menu has the following commands:

Menu item	Command name	ID
Table of Contents…	IDC_HELP_CONTENTS	32322
Index…	IDC_HELP_INDEX	32323
Search...	IDC_HELP_SEARCH	32324
Software Activation...	IDC_ACTIVATION	32701
Order Form...	IDC_OPEN_ORDER_PAGE	32326
Registration...	IDC_REGISTRATION	32330
Check for Updates…	IDC_CHECK_FOR_UPDATES	32700
Support Center...	IDC_OPEN_SUPPORT_PAGE	32327
FAQ on the Web...	IDC_SHOW_FAQ	32331
Download Components and Free Tools…	IDC_OPEN_COMPONENTS_PAGE	32325
MapForce on the Internet..	IDC_OPEN_XML_SPY_HOME	32328
MapForce Training...	IDC_OPEN_MAPFORCE_TRAINING_PAGE	32300

Menu item	Command name	ID
About MapForce...	ID_APP_ABOUT	57664

16.7 Object Reference

Objects:
MapForceCommand
MapForceCommands
MapForceControl
MapForceControlDocument
MapForceControlPlaceHolder

To give access to standard MapForce functionality, objects of the **MapForce automation interface** can be accessed as well. See MapForceControl.Application, MapForceControlDocument.Document and MapForceControlPlaceHolder.Project for more information.

16.7.1 MapForceCommand

Properties:
`ID`
`Label`
`Name`
`IsSeparator`
`ToolTip`
`StatusText`
`Accelerator`
`SubCommands`

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 true • `SubCommands` contains a collection of `Command` objects.
Separator	• `ID` is zero • `IsSeparator` is true

16.7.1.1 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`.

16.7.1.2 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`) or query its status (using `QueryStatus`). If the command is a container for other commands (for example, a top-level menu), or a separator, the ID is 0.

16.7.1.3 IsSeparator

Property: `IsSeparator` as boolean

Description:
The property returns `true` if the command object is a menu separator; `false` otherwise. See also Command.

16.7.1.4 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.

16.7.1.5 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.

16.7.1.6 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.

16.7.1.7 SubCommands

Property: `SubCommands` as `Commands`

Description:
The `SubCommands` property gets the collection of `Command` 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`.

16.7.1.8 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.

16.7.2 MapForceCommands

Properties:
`Count`
`Item`

Description:
Collection of `Command` objects to get access to command labels and IDs of the MapForceControl. Those commands can be executed with the `Exec` method and their status can be queried with `QueryStatus`.

16.7.2.1 Count

Property: `Count` as long

Description:

Number of `Command` objects on this level of the collection.

16.7.2.2 Item

Property: `Item` (n as long) as `Command`

Description:
Gets the command with the index n in this collection. Index is 1-based.

16.7.3 MapForceControl

Properties:
`IntegrationLevel`
`Appearance`
`Application`
`BorderStyle`
`CommandsList`
`CommandsStructure (deprecated)`
`EnableUserPrompts`
`MainMenu`
`Toolbars`

Methods:
`Open`
`Exec`
`QueryStatus`

Events:
`OnUpdateCmdUI`
`OnOpenedOrFocused`
`OnCloseEditingWindow`
`OnFileChangedAlert`
`OnContextChanged`
`OnDocumentOpened`
`OnValidationWindowUpdated`

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
```

16.7.3.1 Properties

The following properties are defined:

`IntegrationLevel`

`EnableUserPrompts`
`Appearance`
`BorderStyle`

Command related properties:
`CommandsList`
`MainMenu`
`Toolbars`
`CommandsStructure (deprecated)`

Access to MapForceAPI:
`Application`

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`.

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.

BorderStyle

Property: `BorderStyle` as short

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

CommandsList

Property: `CommandList` as `Commands` (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`. To get toolbar commands, use `Toolbars`.

```
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

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*.

IntegrationLevel

Property: `IntegrationLevel` as `ICActiveXIntegrationLevel`

Dispatch Id: *1000*

Description:
The `IntegrationLevel` property determines the operation mode of the control. See also Integration at Application Level and Integration at Document Level for more information.

Note: It is important to set this property immediately after the creation of the `MapForceControl` object.

MainMenu

Property: `MainMenu as` [Command](#) (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 likely need to create a recursive function.

```
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

Toolbars

Property: `Toolbars` as [Commands](#) (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

16.7.3.2 Methods

The following methods are defined:

`Open`
`Exec`
`QueryStatus`

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`. To retrieve the status of any command, use `QueryStatus`.

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` and `MapForceControlPlaceHolder.OpenProject`.

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.

16.7.3.3 Events

The MapForceControl ActiveX control provides the following connection point events:

`OnUpdateCmdUI`
`OnOpenedOrFocused`
`OnCloseEditingWindow`
`OnFileChangedAlert`
`OnContextChanged`

`OnDocumentOpened`
`OnValidationWindowUpdated`

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.

OnContextChanged

Event: `OnContextChanged` (`i_strContextName` as String, `i_bActive` as bool) as bool

Dispatch Id: *1004*

Description:
This event is not used in MapForce

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` instead.

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.

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).

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.

OnToolWindowUpdated

Event: `OnToolWindowUpdated(pToolWnd` as long `)`

Dispatch Id: *1006*

Description:
This event is triggered when the tool window is updated.

OnUpdateCmdUI

Event: `OnUpdateCmdUI ()`

Dispatch Id: *1003*

Description:
Called frequently to give integrators a good opportunity to check status of MapForce commands using `MapForceControl.QueryStatus`. Do not perform long operations in this callback.

OnValidationWindowUpdated

Event: `OnValidationWindowUpdated` ()

Dispatch Id: *3*

Description:
This event is triggered whenever the validation output window is updated with new information.

16.7.4 MapForceControlDocument

Properties:
`Appearance`
`BorderStyle`
`Document`
`IsModified`
`Path`
`ReadOnly`

Methods:
`Exec`
`New`
`Open`
`QueryStatus`
`Reload`
`Save`
`SaveAs`

Events:
`OnDocumentOpened`
`OnDocumentClosed`
`OnModifiedFlagChanged`
`OnContextChanged`
`OnFileChangedAlert`
`OnActivate`

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
```

16.7.4.1 Properties

The following properties are defined:

`ReadOnly`
`IsModified`
`Path`
`Appearance`
`BorderStyle`

Access to MapForceAPI:
`Document`

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`.

BorderStyle

Property: `BorderStyle` as short

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

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.

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.

Path

Property: `Path` as string

Dispatch Id: *1005*

Description:
Sets or gets the full path name of the document loaded into the control.

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.

16.7.4.2 Methods

The following methods are defined:

Document handling:
`New`
`Open`
`Reload`
`Save`
`SaveAs`

Command Handling:
`Exec`
`QueryStatus`

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` property of MapForceControl. To get toolbar commands, use the `Toolbars` property of the MapForceControl.

New

Method: `New` () as boolean

Dispatch Id: *1000*

Description:
This method initializes a new document inside the control..

Open

Method: `Open` (`strFileName` as string) as boolean

Dispatch Id: *1001*

Description:
`Open` loads the file `strFileName` as the new document into the control.

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.

Reload

Method: `Reload` () as boolean

Dispatch Id: *1002*

Description:
`Reload` updates the document content from the file system.

Save

Method: `Save` () as boolean

Dispatch Id: *1003*

Description:
`Save` saves the current document at the location `Path`.

SaveAs

Method: `SaveAs` (`strFileName` as string) as boolean

Dispatch Id: *1004*

Description:
`SaveAs` sets `Path` to *strFileName* and then saves the document to this location.

16.7.4.3 Events

The MapForceControlDocument ActiveX control provides following connection point events:

`OnDocumentOpened`
`OnDocumentClosed`
`OnModifiedFlagChanged`
`OnContextChanged`
`OnFileChangedAlert`
`OnActivate`

`OnSetEditorTitle`

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.

OnContextChanged

Event: `OnContextChanged` (`i_strContextName` as String, `i_bActive` as bool) as bool

Dispatch Id: *1004*

Description: None

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.

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.

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.

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.

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.

OnSetEditorTitle

Event: OnSetEditorTitle ()

Dispatch Id: *1006*

Description:
This event is being raised when the contained document is being internally renamed.

16.7.5 MapForceControlPlaceHolder

Properties available for all kinds of placeholder windows:
`PlaceholderWindowID`

Properties for project placeholder window:
`Project`

Methods for project placeholder window:
`OpenProject`
`CloseProject`

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
```

16.7.5.1 Properties

The following properties are defined:

`PlaceholderWindowID`

Access to MapForceAPI:
`Project`

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.

PlaceholderWindowID

Property: `PlaceholderWindowID as MapForceControlPlaceholderWindow`

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` enumeration. The control changes its state immediately and shows the new MapForce window.

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` with a value of `MapForceXProjectWindow (=3)`. The property is read-only.

16.7.5.2 Methods

The following method is defined:

`OpenProject`
`CloseProject`

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` different to `XMLSpyXProjectWindow (=3)`.

CloseProject

Method: `CloseProject` ()

Dispatch Id: *4*

Description:
`CloseProject` closes the project loaded the control. The method will fail if the placeholder window has a `PlaceholderWindowID` different to `MapForceXProjectWindow (=3)`.

16.7.5.3 Events

The MapForceControlPlaceholder ActiveX control provides following connection point events:

`OnModifiedFlagChanged`

OnModifiedFlagChanged

Event: `OnModifiedFlagChanged` (`i_bIsModified` as boolean)

Dispatch Id: *1*

Description:
This event gets triggered only for placeholder controls with a `PlaceholderWindowID` 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.

OnSetLabel

Event: `OnSetLabel(i_strNewLabel` as string)

Dispatch Id: *1000*

Description:
Raised when the title of the placeholder window is changed.

16.7.6 Enumerations

The following enumerations are defined:

`ICActiveXIntegrationLevel`
`MapForceControlPlaceholderWindow`

16.7.6.1 ICActiveXIntegrationLevel

Possible values for the IntegrationLevel property of the MapForceControl.

```
ICActiveXIntegrationOnApplicationLevel  = 0
ICActiveXIntegrationOnDocumentLevel     = 1
```

16.7.6.2 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
```

Chapter 17

Appendices

17 Appendices

These appendices contain technical information about MapForce and important licensing information. Each appendix contains sub-sections as given below:

Technical Data

- OS and memory requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage
- License metering

License Information

- Electronic software distribution
- Copyrights
- End User License Agreement

17.1 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.

17.1.1 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:

- XSLT 1.0
- XSLT 2.0
- XQuery 1.0

17.1.1.1 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 (`&#160;` or `&#xA0;`) or as an entity reference, `&nbsp;` .

17.1.1.2 XSLT 2.0

This section:

- Engine conformance
- Backward compatibility
- Namespaces
- Schema awareness
- Implementation-specific behavior

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:
true

```
<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`.

17.1.1.3 XQuery 1.0

This section:

- Engine conformance
- Schema awareness
- Encoding

- Namespaces
- XML source and validation
- Static and dynamic type checking
- Library modules
- External functions
- Collations
- Precision of numeric data
- XQuery instructions support

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. To use a specific collation, supply its URI as given in the list of supported collations. 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.

17.1.2 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 timezone 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.

17.1.2.1 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

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
- Geolocation
- Image-related
- Numeric
- Sequence
- String
- Miscellaneous

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

Standard 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]

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]

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

- Add a duration to xs:dateTime and return xs:dateTime
- Add a duration to xs:date and return xs:date
- Add a duration to xs:time and return xs:time
- Format and retrieve durations
- Remove timezone from functions that generate current date/time
- Return weekday as integer from date
- Return week number as integer from date
- Build date, time, or duration type from lexical components of each type
- Construct date, dateTime, or time type from string input
- Age-related functions

▼ Grouped alphabetically

`altova:add-days-to-date`

[Top]

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]

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]

Format and retrieve durations 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`.

▼ 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]

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`

[Top]

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-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-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-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`

[Top]

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. The weekdays are numbered starting with `Monday=1`. 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. The weekdays are numbered starting with `Monday=1`. 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]

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]

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 arguments 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]

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`

[Top]

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]

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

▼ 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` function and the Exif metadata's `@Geolocation` 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 (48.858222222222, 24.2945)`
- **`altova:parse-geolocation`**`('48°51''29.6"N  24°17''40.2"')` returns the sequence of two `xs:decimals (48.858222222222, 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`, `W/E`)
  `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/W`**) 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`, `W/E`)
  `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/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/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` function and the Exif metadata's `@Geolocation` 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`, `W/E`)
  `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/W`**) 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`, `W/E`)
  `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/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S D.DDW/E`
  Example: **`33.33N 22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/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` function and the Exif metadata's `@Geolocation` 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`, `W/E`)
  `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/W`**) 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`, `W/E`)
  `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/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/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-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` function and the Exif metadata's `@Geolocation` 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`, `W/E`)
  `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/W`**) 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`, `W/E`)
  `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/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/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` function and the Exif metadata's `@Geolocation` 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`, `W/E`)
  `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/W`**) 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`, `W/E`)
  `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/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/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	GPSLatitude Ref	GPSLongitude	GPSLongitude Ref	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

[Top]

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*

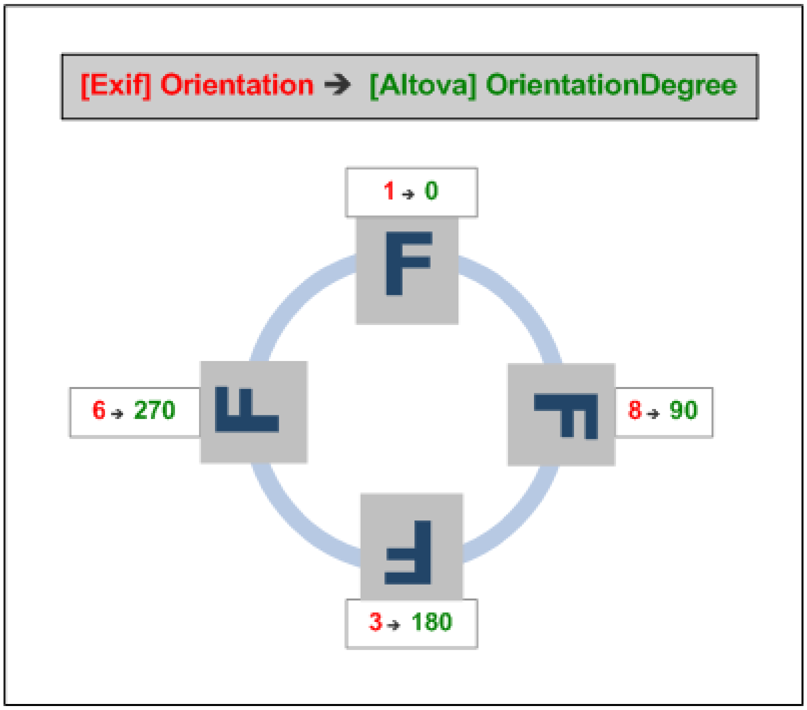
The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

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]

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]

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]

Number-formatting 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")`.

[Top]

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:attributes("MyElem", "rip")` returns `MyElement()*`
- `altova:attributes("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.*

▾ 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)`

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.

▼ 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`

▼ 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"`

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**

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`

[Top]

17.1.2.2 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 and .NET, as well as MSXSL scripts for XSLT. 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
- .NET Extension Functions
- MSXSL Scripts for XSLT

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.

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:

- Java: Constructors
- Java: Static Methods and Static Fields
- Java: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to Java
- Datatypes: Java to XPath/XQuery

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 and User-Defined Jar Files. Note that paths to class files not in the current directory and to all JAR files must be specified.

User-Defined Class Files

If access is via a class file, then there are two 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*.)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. (*See example below*.)
- The class file is in a package. The XSLT or XQuery file is at some random location. (*See example below*.)
- The class file is not packaged. The XSLT or XQuery file is at some random location. (*See example below*.)

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 not packaged, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()`method of the `Car` class of the `com.altova.extfunc` package. The `Car` class file is in the following folder location: `JavaProject/com/altova/extfunc`. 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 not 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. The location of the class file is 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.

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.

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, 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*):
  ```
  <xsl:value-of select="date:toString(date:new())"
  xmlns:date="java:java.util.Date" />
  ```

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>
```

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`.

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.

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).

.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
- .NET: Static Methods and Static Fields
- .NET: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to .NET
- Datatypes: .NET to XPath/XQuery

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.

.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, 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*):
  ```
  <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(data:get_Month(date:new(2008, 4, 29)))"
  xmlns:date="clitype:System.DateTime" />
  ```

.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>
```

.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.

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.

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).

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.

17.2 Technical Data

This section contains useful background information on the technical aspects of your software. It is organized into the following sections:

- OS and Memory Requirements
- Altova XML Validator
- Altova XSLT and XQuery Engines
- Unicode Support
- Internet Usage

17.2.1 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 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. 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.

17.2.2 Altova XML Validator

When opening any XML document, the application uses its built-in XML validator to check for well-formedness, validate the document against a schema (if specified), and 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 specification. 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.

17.2.3 Altova 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. Documentation about implementation-specific behavior for each engine is in the appendices of the documentation (Engine Information), should that engine be used in the product.

Note: Altova MapForce generates code using the XSLT 1.0, 2.0 and XQuery 1.0 engines.

17.2.4 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.

17.2.5 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.

17.3 License Information

This section contains:

- Information about the distribution of this software product
- Information about software activation and license metering
- Information about the intellectual property rights related to this software product
- The End-User License Agreement governing the use of this software product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

17.3.1 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 before making a purchasing decision.
- Once you decide to buy the software, you can place your order online at the Altova website and immediately 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 a comprehensive integrated onscreen help system. The latest version of the user manual is available at www.altova.com (i) in HTML format for online browsing, and (ii) in 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 this 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 have to purchase an Altova Software License Agreement, which is delivered in the form of a key-code that you enter into the Software Activation dialog to unlock the product. You can purchase your license at the online shop at the Altova website.

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 only distribute the Setup programs, provided that they are not modified in any way. Any person that 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.

For further details, please refer to the Altova Software License Agreement at the end of this section.

17.3.2 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 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 website (http://www.iana.org/) *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.

You will also notice that, if you are online, your Altova product contains many useful functions; these are unrelated to the license-metering technology.

17.3.3 Intellectual Property Rights

The Altova Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

Altova software contains certain Third Party Software that is also protected by intellectual property laws, including without limitation applicable copyright laws as described in detail at http://www.altova.com/legal_3rdparty.html.

All other names or trademarks are the property of their respective owners.

17.3.4 Altova End User License Agreement

- The Altova End User License Agreement is available here: http://www.altova.com/eula
- Altova's Privacy Policy is available here: http://www.altova.com/privacy

Chapter 18

Glossary

18 Glossary

The glossary section includes the list of terms pertaining to MapForce.

18.1 C

Component

In MapForce, the term "component" is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Databases
- Filters
- Conditions
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- XML Schemas and DTDs

Connection

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

Connector

A connector is a small triangle displayed on the left or right side of a component. The connectors displayed on the left of a component provide data entry points *to that component*. The connectors displayed on the right of a component provide data exit points *from that component*.

18.2 F

Fixed Length Field (FLF)

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).

FlexText

FlexText is a module in MapForce Enterprise Edition which enables you to convert data from non-standard or legacy text files of high complexity to other formats supported by MapForce, and vice versa.

18.3 G

Global Resources

Altova Global Resources represent a way to refer to files, folders, or databases so as to make these resources reusable, configurable and available across multiple Altova applications.

18.4 I

Input component

An input component is a MapForce component that enables you to pass simple values to a mapping. Input components are commonly used to pass file names or other string values to a mapping at runtime. Input components should not be confused with source components.

18.5 J

Join component

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).

18.6 M

MapForce

MapForce is a Windows-based, multi-purpose IDE (integrated development environment) that enables you to transform data from one format to another, or from one schema to another, by means of a visual, "drag-and-drop" -style graphical user interface that does not require writing any program code. In fact, MapForce generates for you the program code which performs the actual data transformation (or data mapping). When you prefer not to generate program code, you can just run the transformation using the MapForce built-in transformation language (available in the MapForce Professional or Enterprise Editions).

Mapping

A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). 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 mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.

MFF

The file name extension of MapForce function files.

MFD

The file name extension of MapForce design documents (mappings).

MFP

The file name extension of MapForce Project files.

18.7 O

Output component

An output component (or "simple output") is a MapForce component which enables you to return a string value from the mapping. Output components represent just one possible type of target components, but should not be confused with the latter.

18.8 P

parent-context

parent-context is an optional argument in some MapForce core aggregation functions such as **`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.

18.9 S

Source component

A source component is a component from which MapForce reads data. When you run the mapping, MapForce reads the data supplied by the connector of the source component, converts it to the required type, and sends it to the connector of the target component.

18.10 T

Target component

A target component is a component to which MapForce writes data. When you run the mapping, a target component instructs MapForce to either generate a file (or multiple files) or output the result as a string value for further processing in an external program. A target component is the opposite of a source component.

Index