Altova MapForce 2025 Professional Edition

ALTOVA®
MapForce®

User & Reference Manual

Altova MapForce 2025 Professional Edition
User & Reference Manual

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

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: 2025

Table of Contents

1 Introduction 12

1.1 New Features 13

1.1.1 Version 2025 13

1.1.2 Version 2024 13

1.1.3 Version 2023 14

1.1.4 Version 2022 15

1.1.5 Version 2021 16

1.2 What Is MapForce? 17

1.2.1 Mapping: Sources and Targets 18

1.2.2 Mapping Scenarios 18

1.2.3 Transformation Languages 19

1.2.4 Integration with Altova Products 20

1.3 User Interface Overview 22

1.3.1 Bars 23

1.3.2 Windows 23

1.3.3 Messages Window 26

1.3.4 Panes 28

2 Mapping Fundamentals 32

2.1 Components 34

2.1.1 Add Components 38

2.1.2 Component Basics 40

2.1.3 File Paths 43

2.2 Connections 48

2.2.1 Connection Types 51

2.2.2 Connection Settings 58

2.2.3 Connection Context Menu 59

2.2.4 Faulty Connections 60

2.2.5 Keep Connections after Deleting Components 62

2.3 General Procedures and Features 64
2.3.1 Validation 64
2.3.2 Code Generation 66
2.3.3 Text View Features 68
2.3.4 Text View Search 72
2.3.5 Mapping Settings 75
2.4 Basic Rules and Strategies 77
2.4.1 Sequences 78
2.4.2 Context and Processing Order 79
2.4.3 Parent Context 82
2.4.4 Priority Context 86
2.4.5 Multiple Target Components 91
2.5 Projects 95
2.5.1 Project Basics 95
2.5.2 Project Settings 97
2.5.3 Project Folders 98

3 Tutorials 100

3.1 One Source to One Target 101
3.1.1 Create and Save Design 102
3.1.2 Add Source Component 103
3.1.3 Add Target Component 104
3.1.4 Connect Source and Target 106
3.1.5 Preview Mapping Result 108
3.2 Multiple Sources to One Target 110
3.2.1 Prepare Mapping Design 111
3.2.2 Add Second Source 111
3.2.3 Configure Output 112
3.2.4 Connect Second Source and Target 113
3.3 Chained Mapping 115
3.3.1 Prepare Mapping Design 115
3.3.2 Configure Second Target 116
3.3.3 Connect Targets 118
3.3.4 Filter Data 118

3.3.5 Preview and Save Output 121
3.4 Multiple Sources to Multiple Targets 123
3.4.1 Configure Input 125
3.4.2 Configure Output Part 1 126
3.4.3 Configure Output Part 2 128

4 Structural Components 131
4.1 XML and XML Schema 132
4.1.1 XML Component Settings 133
4.1.2 Derived Types 137
4.1.3 NULL Values 139
4.1.4 Comments and Processing Instructions 143
4.1.5 CDATA Sections 143
4.1.6 Wildcards: xs:any/xs:anyAttribute 145
4.1.7 Custom Namespaces 147
4.1.8 Schema Manager 149
4.2 Databases 165
4.2.1 Connect to a Data Source 168
4.2.2 General Procedures and Features 230
4.2.3 DB Table Actions 258
4.2.4 DB Query Pane 277
4.2.5 Map XML Data to/from DB Fields 287
4.2.6 Stored Procedures 297
4.3 CSV and Text Files 323
4.3.1 Example: Mapping CSV Files to XML 323
4.3.2 Example: Iterating Through Items 325
4.3.3 Example: Creating Hierarchies from CSV and Fixed-Length Text Files 328
4.3.4 Setting the CSV Options 331
4.3.5 FLF to Database 335
4.3.6 Setting the FLF Options 339

5 Transformation Components 345
5.1 Simple Input 346

5.1.1 Adding Simple Input Components 347
5.1.2 Simple Input Component Settings 348
5.1.3 Creating a Default Input Value 349
5.1.4 Example: Using File Names as Mapping Parameters 350
5.2 Simple Output 356
5.2.1 Adding Simple Output Components 357
5.2.2 Example: Previewing Function Output 358
5.3 Variables 360
5.3.1 Add a Variable 362
5.3.2 Scope and Context of Variables 366
5.3.3 Example: Counting Database Table Rows 368
5.3.4 Example: Filtering and Numbering Nodes 369
5.3.5 Example: Grouping and Subgrouping Records 371
5.4 Join Components 373
5.4.1 Adding Join Conditions 375
5.4.2 Joining Three or More Structures 378
5.4.3 Example: Join XML Structures 379
5.4.4 Join Database Data 384
5.5 Sort Components 402
5.5.1 Sorting by Multiple Keys 404
5.5.2 Sorting with Variables 405
5.6 Filters and Conditions 408
5.6.1 Example: Filtering Nodes 410
5.6.2 Example: Returning a Value Conditionally 412
5.6.3 Filter and Sort Database Data 413
5.7 Value-Maps 421
5.7.1 Example: Replacing Weekdays 425
5.7.2 Example: Replacing Job Titles 428
5.8 Exceptions 432
5.8.1 Example: Exception on "Greater Than" Condition 433
5.8.2 Example: Exception When Node Does Not Exist 433
5.9 Functions 436
5.9.1 Functions Basics 436
5.9.2 Manage Function Libraries 438
5.9.3 Defaults and Node Functions 442

5.9.4 User-Defined Functions 457
5.9.5 Custom Functions 475
5.9.6 Regular Expressions 508
5.9.7 Function Library Reference 511

6 Advanced Mapping Procedures 725
6.1 Map Node Names 726
6.1.1 Get Access to Node Names 727
6.1.2 Get Access to Nodes of Specific Type 735
6.1.3 Example: Map Element Names to Attribute Values 738
6.1.4 Example: Group and Filter Nodes by Name 742
6.2 Batch-Process Files 746
6.2.1 Example: Split One XML File into Many 748
6.2.2 Example: Split Database Table into Many XML Files 750
6.3 Parse and Serialize Strings 753
6.3.1 About the Parse/Serialize Component 753
6.3.2 Example: Serialize to String (XML to Database) 755

7 Mapping Documentation 761
7.1 Predefined StyleVision Power Stylesheets 764
7.2 Custom Stylesheets 769

8 Debugger 771
8.1 Debugger Preparation 774
8.2 About the Debug Mode 775
8.3 Adding and Removing Breakpoints 778
8.4 Using the Values Window 780
8.5 Using the Context Window 782
8.6 Using the Breakpoints Window 784
8.7 Previewing Partially Generated Output 786
8.8 Viewing the Current Value of a Connector 787
8.9 Stepping back into Recent Past 788
8.10 Viewing the History of Values Processed by a Connector 789

8.11 Setting the Context to a Value 790

9 Automation with Altova Products 791

9.1 Automation with RaptorXML Server 792
9.2 Automation with MapForce Server 793
9.3 Preparing Mappings for Server Execution 794
9.4 Compiling Mappings to MapForce Server Execution Files 799
9.5 Deploying Mappings to FlowForce Server 802
9.6 StyleVision Output Panes 807
9.7 MapForce Command Line Interface 811

10 Altova Global Resources 815

10.1 Global Resource Setup Part 1 816
10.2 Global Resource Setup Part 2 818
10.3 XML Files as Global Resources 822
10.4 Folders as Global Resources 824
10.5 Databases as Global Resources 826
10.6 Transformation Results as Global Resources 828
10.7 Global Resources in Execution Environments 832
10.7.1 Global Resources in Generated Code 832
10.7.2 Global Resources in MapForce Server 833
10.7.3 Global Resources in FlowForce Server 833

11 Catalogs in MapForce 836

11.1 How Catalogs Work 837
11.2 Catalog Structure in MapForce 839
11.3 Customizing Your Catalogs 841
11.4 Environment Variables 843

12 MapForce Plug-in for Visual Studio 844

13 MapForce Plug-in for Eclipse 847

13.1 Installing the MapForce Plug-in for Eclipse ... 848
13.2 The MapForce Perspective ... 850
13.3 Accessing Common Menus and Functions ... 853
13.4 Working with Mappings and Projects ... 857
13.4.1 Creating a MapForce/Eclipse Project ... 857
13.4.2 Creating New Mappings ... 859
13.4.3 Importing Existing Mappings into an Eclipse Project ... 861
13.4.4 Configuring Automatic Build and Generation of MapForce Code ... 864
13.5 Extending MapForce Plug-in for Eclipse ... 867

14 Code Generator 868
14.1 Generate, Build, and Run Code ... 870
14.2 Integrate Generated Code ... 876
14.2.1 Modify Input/Output, Define Error Handling ... 876
14.2.2 Change Data Type of Input/Output ... 879
14.2.3 Generate Code from XML Schemas or DTDs ... 885
14.2.4 Generated Classes (C++) ... 929
14.2.5 Generated Classes (C#) ... 944
14.2.6 Generated Classes (Java) ... 959
14.2.7 SPL Reference ... 973

15 MapForce COM API 992

16 ActiveX Integration 993
16.1 Prerequisites ... 994
16.2 Adding the ActiveX Controls to the Toolbox ... 996
16.3 Integration at Application Level ... 998
16.4 Integration at Document Level ... 1001
16.5 ActiveX Integration Examples ... 1004
16.5.1 C# ... 1004
16.5.2 Java ... 1011
16.5.3 VB.NET ... 1020
16.6 Command Reference ... 1024

16.6.1 "File" Menu ... 1024
16.6.2 "Edit" Menu ... 1025
16.6.3 "Insert" Menu ... 1026
16.6.4 "Project" Menu ... 1026
16.6.5 "Component" Menu ... 1027
16.6.6 "Connection" Menu ... 1028
16.6.7 "Function" Menu ... 1029
16.6.8 "Output" Menu ... 1029
16.6.9 "Debug" Menu ... 1030
16.6.10 "View" Menu ... 1031
16.6.11 "Tools" Menu ... 1031
16.6.12 "Window" Menu ... 1032
16.6.13 "Help" Menu ... 1032
16.7 Object Reference ... 1034
16.7.1 MapForceCommand ... 1034
16.7.2 MapForceCommands ... 1036
16.7.3 MapForceControl ... 1037
16.7.4 MapForceControlDocument ... 1045
16.7.5 MapForceControlPlaceHolder ... 1051
16.7.6 Enumerations ... 1054

17 Menu Commands 1055
17.1 File ... 1056
17.2 Edit ... 1058
17.3 Insert ... 1059
17.4 Project ... 1062
17.5 Component ... 1064
17.6 Connection ... 1066
17.7 Function ... 1067
17.8 Output ... 1068
17.9 Debug ... 1070
17.10 View ... 1071
17.11 Tools ... 1073
17.11.1 Customize Menus ... 1074

17.11.2 Customize Shortcuts ... 1075
17.11.3 Options ... 1077
17.12 Window ... 1090
17.13 Help ... 1091

18 Appendices 1096

18.1 Support Notes ... 1097
18.1.1 Supported Sources and Targets ... 1097
18.1.2 Supported Features in Generated Code ... 1098
18.2 Engine Information ... 1101
18.2.1 XSLT and XQuery Engine Information ... 1101
18.2.2 XSLT and XPath/XQuery Functions ... 1106
18.3 Technical Data ... 1202
18.3.1 OS and Memory Requirements ... 1202
18.3.2 Altova Engines ... 1202
18.3.3 Unicode Support ... 1203
18.3.4 Internet Usage ... 1203
18.4 License Information ... 1204
18.4.1 Electronic Software Distribution ... 1204
18.4.2 Software Activation and License Metering ... 1205
18.4.3 Altova End-User License Agreement ... 1206
18.4.4 Packaging License Files with MapForce Installer ... 1206

Index 1207

1 Introduction

Altova MapForce 2025 Professional Edition is a powerful data transformation and ETL tool for integrating data. MapForce is a 32/64-bit Windows application that runs on Windows 10, Windows 11, and Windows Server 2016 or newer. 64-bit support is available for the Enterprise and Professional editions.

MapForce enables you to convert data from and to nearly any format. MapForce has a graphical interface 22 that includes many options for managing, visualizing, manipulating, and executing individual mappings and complex mapping projects. For data transformation, MapForce provides an extensive library of data processing and conversion functions 436 to filter and manipulate data according to the needs of your data integration project.

As soon as you have finished designing your mapping, you can preview the output in a separate pane and save the output to the desired location. Besides, you can generate code 66 for external execution.

You can extend MapForce functionality by integrating MapForce with other Altova products:

- You can execute your mappings using MapForce Server. This will help you automate business operations that require repetitive data transformations. MapForce Server includes a powerful data transformation engine and can perform any-to-any data conversions. Importantly, it is a cross-platform server that is available on Windows, macOS, and Linux.
- RaptorXML Server is a hyper-fast engine that validates your instances.
- FlowForce Server helps you automate your tasks and enables you to run your mappings as scheduled jobs.
- StyleVision Server generates output in HTML, RTF, PDF and Word.
- You can use StyleVision to design StyleVision Power Stylesheets that enable StyleVision Server to generate output in multiple formats.
- DatabaseSpy is a versatile tool that allows you to design, edit and query databases.
- XMLSpy is particularly useful when you want to edit your mapping files. Some MapForce dialogs allow you to open files directly in XMLSpy.
- You can also use MapForce as a plug-in of Microsoft Visual Studio and Eclipse. This enables you to access MapForce functionality without leaving your preferred development environment.

Last updated: 17 March 2025

1.1 New Features

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

1.1.1 Version 2025

Version 2025 Release 2

- Support for Shopify/GraphQL APIs (*Enterprise Edition*).
- Support for MySQL 9, DB2 12.1 and SQLite 3.47.2 (*Professional and Enterprise editions*). For details, see Databases 165.
- Internal updates and optimizations.

Version 2025

- Support for the OpenAPI Specification 2.0, 3.0, and 3.1 for HTTP APIs (*Enterprise Edition*).
- The PDF Extractor now enables you to search for text in the user interface as well as at runtime (*Enterprise Edition*).
- It is now possible to auto-generate EDIFACT CONTRL messages that are sent in response to received EDIFACT messages (*Enterprise Edition*).
- Support for UN/EDIFACT 2023A (*Enterprise Edition*).
- Support for SWIFT 2024 (*Enterprise Edition*).
- Support for Azure CosmosDB (*Enterprise Edition*). For details, see Databases 165.
- Internal updates and optimizations.

1.1.2 Version 2024

Version 2024 Release 2

- It is now possible to access various video tutorials in the **`MapForceExamples`** project (*Professional and Enterprise editions*). Besides, you can also add your own links to external resources. For details, see Project Basics 97.
- When you deploy your mapping to FlowForce Server, you can choose to attach the mapping files for later retrieval. This will prevent you from losing your mapping files and enable you to download them at any time (*Professional and Enterprise editions*). For details, see Deploying Mappings to FlowForce Server 802.
- Database components can now share the same database connection at runtime (*Professional and Enterprise editions*). For details, see Database Component Settings 242.
- It is now possible to create a Value-Map from enumeration types in XML (*all editions*) and XBRL components (*Enterprise Edition*). This feature makes it easier and faster to map enumeration values: Both sides of the Value-Map become pre-filled with all enumeration values, and you will only need to review and edit the relevant values of the Value-Map. For more information, see Value-Maps 424.
- Support for .NET 8.0 for C# code generation (*Professional and Enterprise editions*). For details, see Code Generation 66.

- Support for FORTRAS EDI messages (*Enterprise Edition*).
- Support for PostgreSQL 16, MySQL 8.2, MySQL 8.3, MariaDB 11.2, SQLite 3.45 (*Professional and Enterprise editions*). For details, see Databases 165.
- Internal updates and optimizations.

Version 2024

- A new MapForce utility called PDF Extractor is now available (*Enterprise Edition*). The PDF Extractor enables you to create PDF extraction templates that you can import into MapForce and use as source components in your mappings.
- It is now possible to create AI-powered mappings in MapForce (*Enterprise Edition*). MapForce enables you to create REST web service calls to an API, such as OpenAI API, Azure OpenAI API, AWS AI Services, etc.
- Support for SWIFT 2023 (*Enterprise Edition*).
- A new **`sleep`** function is now available, which allows passing through data after a specified delay (*Professional and Enterprise editions*). For more information, see `sleep` 659.
- Native support has been added for MySQL and MariaDB (*Professional and Enterprise editions*). For more information about supported databases, see Databases 165.
- The functionality of matching-children connections has been improved and extended to include new matching options. For details, see Matching-Children Connections 54.
- The *Client Credentials* and *Resource Owner Password Credentials* grant types are now supported in OAuth credentials, in addition to the *Authorization Code* grant type (*Enterprise Edition*).
- Internal updates and optimizations.

1.1.3 Version 2023

Version 2023 Release 2

- It is now possible to generate the `standalone="yes"` declaration in the XML declaration of XML target files. For details, see XML Component Settings 136.
- The Help system 1091 has been reorganized to provide Online Help by default, with an option to use the locally installed PDF user manual 1077 as the alternative default.
- It is now possible to add sticky-note-style comments to a mapping. For more information, see Comments 36.
- Settings have been added to define network settings 1087.
- Support for VDA EDI messages has been added (*Enterprise Edition*).
- Internal updates and optimizations.

Version 2023

- Support for the following themes has been added: *Classic*, *Light*, and *Dark*. For more information, see Window 1090.
- Internal updates and optimizations.
- Eclipse support has been updated and now covers the following versions: 2022-09, 2022-06, 2022-03, 2021-12 (*Professional and Enterprise editions*). For more details, see MapForce Plug-in for Eclipse 847.
- Support for ODETTE EDI messages (*Enterprise Edition*).
- Support for the XII Transformation Registry 5 Specification (*Enterprise Edition*).

- It is now possible to create database-based UDF parameters[464] and variables[361] with a tree of related tables (*Professional and Enterprise editions*).
- It is now possible to send an `application/x-www-form-urlencoded` request structure to a REST service (*Enterprise Edition*).
- Support for UN/EDIFACT D.21B and D.22A Directories (*Enterprise Edition*).
- Support for SQLite 3.39.2, MariaDB 10.9.2, and PostgreSQL 14.5 (*Professional and Enterprise editions*). To find out more about all supported databases, see Databases[165].
- Support for XML Schema Manager[149] that provides a centralized way to install and manage XML schemas for use across all Altova's XBRL-enabled applications.
- Support for mappable EDI delimiters (*Enterprise Edition*). The feature is currently supported for the following EDI standards: EDIFACT, X12, and NCPDP SCRIPT.

1.1.4 Version 2022

Version 2022 Release 2

- Internal updates and optimizations
- Eclipse support has been updated and now covers the following versions: 2021-12, 2021-09; 2021-06; 2021-03 (*Professional and Enterprise editions*). For details, see MapForce Plug-in for Eclipse[847].
- Support for Visual Studio 2022 in the MapForce Plug-in for Visual Studio and code generation (*Professional and Enterprise editions*). For more information, see MapForce Plug-in for Visual Studio[844] and code generation.
- Support for .NET 6.0 in code generation (*Professional and Enterprise editions*). For details, see code generation.
- New database versions are supported: PostgreSQL 14, SQLite 3.37.2, MariaDB 10.6.5, MySQL 8.0.28, IBM DB2 11.5.7 (*Professional and Enterprise editions*). To find out more about all supported databases, see Databases[165].
- It is now possible to preview images in the **Project** window (*Professional and Enterprise editions*). For more information, see Project Basics[97].
- It is now possible to create EBA-conformant filing indicators for target XBRL components (*Enterprise Edition*).

Version 2022

- Internal updates and optimizations
- Eclipse support has been updated and now covers the following versions: 2021-09; 2021-06; 2021-03; 2020-12 (*Professional and Enterprise editions*). For details, see MapForce Plug-in for Eclipse[847].
- Copy-all connections[56] now support JSON. This feature is available only for compatible JSON types (*Enterprise Edition*).
- A new StyleVision output pane called *Text* has been introduced. If an SPS file is attached to a component, the new plain text output format can be previewed in MapForce (*Professional and Enterprise editions*). For more information, see StyleVision Output Panes[807].
- Support for JSON Schema in variables[360] and UDF parameters[463] (*Enterprise Edition*).
- Support for NoSQL databases: MongoDB and CouchDB (*Enterprise Edition*). To find out more about all supported databases, see Databases[165].
- A new `bson` function library has now become available, which allows you to create and manipulate some of the BSON types (*Enterprise Edition*).
- Support for UN/EDIFACT D.20B and D.21A Directories.
- Support for SWIFT 2021.

1.1.5 Version 2021

Version 2021 Release 3

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

Version 2021 Release 2

- XSLT 3.0 is now supported as mapping language. See Generating XSLT Code 66. MapForce now includes new built-in functions that are supported when the mapping language is XSLT 3.0. For more information, see Function Library Reference 511.
- When generating C# code, you can select .NET Core 3.1 and .NET 5.0 as target frameworks from code generation options (this adds to existing support for .NET Framework projects). For details, see Generating C# code.
- Internal updates and optimizations.

Version 2021

- A MapForce mapping can read BLOB (binary large object) data from binary files and write binary files to the disk. This makes it possible, for example, to read BLOB fields from a database and save them as image files on the disk, or to read binary files such as PDFs from the disk and save them as `xs:base64Binary` fields within an XML file. See Binary Files 627 for more information.
- New database versions are supported: MariaDB 10.4, 10.5
- New Eclipse versions are supported: 2019.09, 2019.12, 2020.03, 2020.06
- When joining multiple database tables or views using SQL join components in a mapping, you can set the join mode either as LEFT OUTER JOIN or INNER JOIN, see Changing the Join Mode 387.
- Internal updates and optimizations.

1.2 What Is MapForce?

Altova website: Data mapping tool

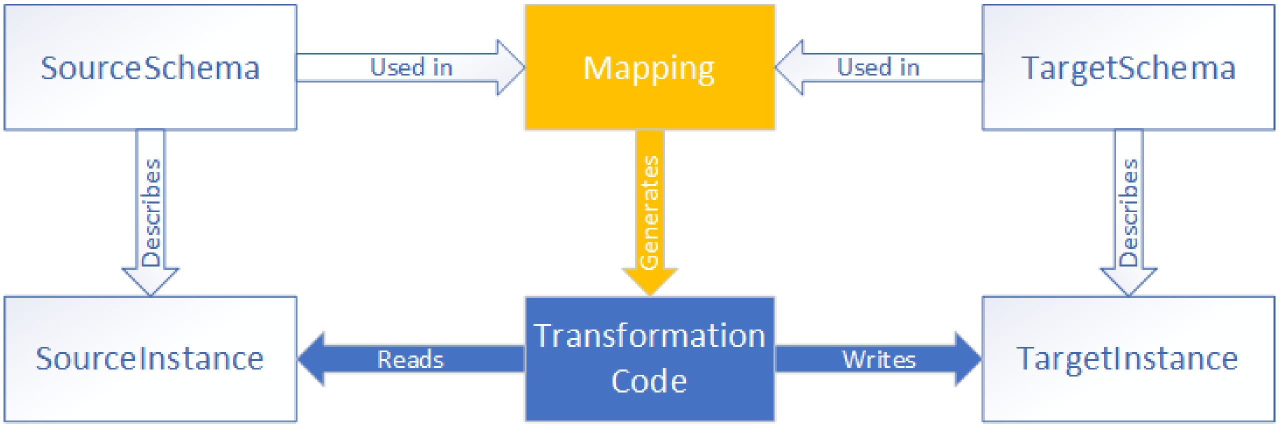
MapForce is a powerful graphical tool for any-to-any conversion and integration. See Mapping: Sources and Targets 18 for a complete list of available data formats. A typical mapping consists of one or more data sources and one or more data targets 34. The mapping can also include one or several transformation components 35 that provide you with an extensive range of data-processing and filtering options. To find out more about various mapping scenarios, see Mapping Scenarios 18 and Tutorials 100.

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

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

Abstract model

The abstract model below illustrates one of the basic scenarios of data transformation in MapForce. The source schema describes the structure of the source instance. The target schema describes the structure of the target instance. Depending on your needs, source and target schemas can be the same or different structures. When you connect the source and target, the mapping generates transformation code (in the selected transformation language 19) that reads data from the source instance and writes this data to the target instance. To see how this data transformation model is implemented in a concrete example, see Tutorial 1 101.



In real-life situations, you can mix and match any combination of data sources (e.g., XML, EDI, and text files) and map them to any combination of data targets (e.g., a database and an Excel file).

Conventions

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

- `C:\Users\<username>\Documents\Altova\MapForce2025\MapForceExamples`
- `C:\Users\<username>\Documents\Altova\MapForce2025\MapForceExamples\Tutorial`

- C:\Users\<username>\Documents\Altova\MapForce2025\MapForceExamples\Tutorial\BasicTutorials

In this section

This section is organized into the following topics:

- Mapping: Sources and Targets 18
- Mapping Scenarios 18
- Transformation Languages 19
- Integration with Altova Products 20

1.2.1 Mapping: Sources and Targets

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

MapForce Basic Edition

- XML and XML Schema

MapForce Professional Edition

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

MapForce Enterprise Edition

- XML and XML Schema
- Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format
- Data from legacy text files can be mapped and converted to other formats with MapForce FlexText
- SQL Databases: all major relational databases
- NoSQL Databases
- Binary files (raw BLOB content)
- EDI standards
- JSON files
- Microsoft Excel 2007 and later files
- XBRL instance files and taxonomies
- Protocol Buffers
- PDF files based on PDF templates created in the PDF Extractor (can only be used as data sources)

1.2.2 Mapping Scenarios

Altova website: MapForce Video Demos

Depending on your business needs and requirements, the complexity of your mappings can vary: For example, you may need to configure your mapping to read data from one source and write this data to multiple targets or to merge data from multiple sources into one target. Different data structures can be used as sources and targets: e.g., XML files, databases, EDI files, etc. To find out more about the supported data formats, see Mapping: Sources and Targets 18.

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

- Mapping one source to one target. For more information, see Tutorial 1 101.
- Merging multiple data sources into one target. For more information, see Tutorial 2 110.
- Mapping data from one source to the first target, then filtering the data in such a way that only a subset of this data is mapped to the second target. See Tutorial 3 115.
- Mapping multiple sources to multiple targets. See Tutorial 4 123.

Regardless of the technology you work with, MapForce typically determines automatically the structure of your data or suggests supplying a schema for your data. MapForce can also generate schemas from a sample instance file. For example, if you have an XML instance file but no schema definition, MapForce can generate it for you. Thus, MapForce makes the data inside the XML file available for mapping to other files or formats. To find out more about the basic terms and features of MapForce, see Mapping Fundamentals 32 and User Interface Overview 22.

Projects (Professional and Enterprise editions)
For easier access and management, you can organize your data mapping designs into mapping projects. In addition to generating code for individual mappings within the project, you can generate program code for entire projects. For details, see Projects 95.

1.2.3 Transformation Languages

In MapForce, a transformation language is used to generate transformation code that executes mappings. You can select/modify a transformation language at any time. You can generate program code via the menu command **File | Generate Code in** or **File | Generate Code in Selected Language** and use this code to carry out data transformations outside of MapForce. For more information, see Code Generation 66.

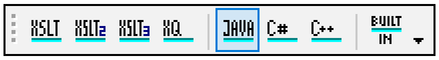
Depending on the MapForce edition, you can choose the following languages for your data transformation:

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

If you select XSLT 1-3 or XQuery as a transformation language, you will be able to view the transformation code in a separate pane of MapForce.

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

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



When you change the transformation language of the mapping, certain MapForce features may not be supported for that language. For more information, see Support Notes[1097].

As you design or preview mappings, MapForce validates the integrity of your schemas and transformations. If any validation errors occur, MapForce displays them in the Messages window[26]. This is helpful, because you can immediately review and correct these errors.

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

The Built-In engine executes mappings without the need for any external processors, which may be a good choice if memory usage is an issue. If you do not need to generate program code in a specific language, use Built-In as a default option, because it supports most MapForce features compared to other languages (see Support Notes[1097]). Furthermore, if you select Built-In as a transformation language, you will be able to automate your mappings with MapForce Server. For more information, see Automation with Altova Products[791]
.

1.2.4 Integration with Altova Products

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

XMLSpy
If XMLSpy is installed on the same machine, you can conveniently open and edit any supported file types by opening XMLSpy directly from the relevant MapForce contexts. For example, the menu command **Component | Edit Schema Definition in XMLSpy** is available when you click an XML component.
RaptorXML Server
You can choose to run the generated XSLT code directly in MapForce and preview the data transformation result immediately. When you need increased performance, you can process the mapping using RaptorXML Server, an ultra-fast XML transformation engine.
MapForce Server (Enterprise and Professional editions)
You can automate MapForce tasks with the help of Altova MapForce Server, which can be installed on Windows, Linux, and macOS systems. MapForce Server enables you to run the transformations specified in a mapping, not only from the command line of the respective OS but also through API calls (.NET, COM, Java).
FlowForce Server (Enterprise and Professional editions)
You can also automate MapForce tasks with the help of Altova FlowForce Server, which can be installed on Windows, Linux, and macOS systems. FlowForce Server enables you to carry out MapForce Server tasks according to a schedule.
StyleVision (Enterprise and Professional editions)
With the help of StyleVision, you can design or reuse existing StyleVision Power Stylesheets and preview the result of the mapping transformations as HTML, RTF, PDF or Word 2007+ documents.

MapForce as a plug-in

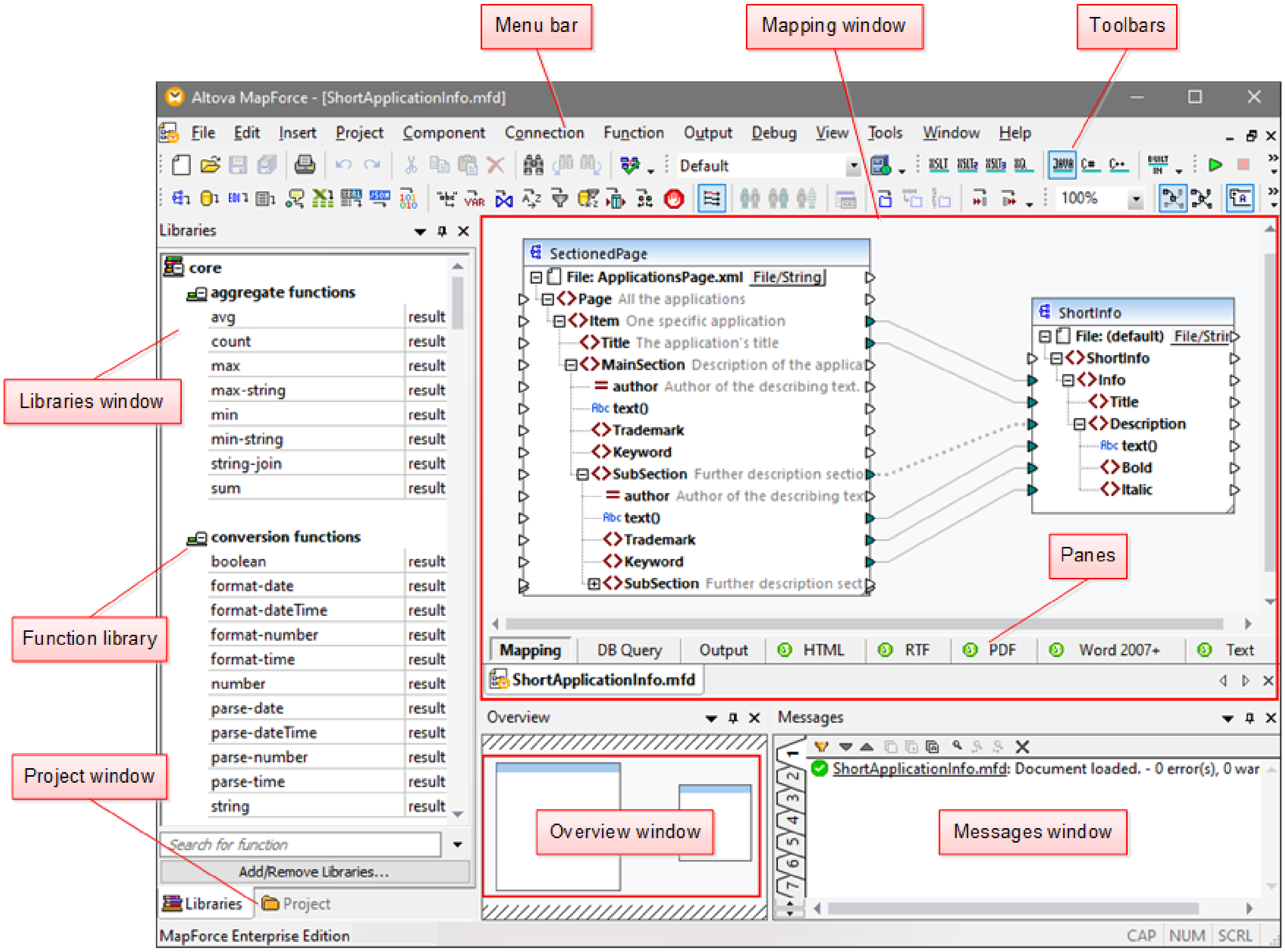
MapForce Professional and Enterprise editions can be installed as a plug-in of Visual Studio and Eclipse integrated development environments. This way, you can design mappings and get access to the MapForce functionality without leaving your preferred development environment.

For more information about automating tasks, see Automating MapForce Tasks with Altova Products 791. To find out more about using MapForce as a plug-in, see Plug-in for Visual Studio 844 and Plug-in for Eclipse 847.

1.3 User Interface Overview

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

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



ShortApplicationInfo.mfd: Document loaded. - 0 error(s), 0 war

In the screenshot above, the panes with the green logos are StyleVision panes. For details, see StyleVision Output Panes 31.

For more information about the features and functions of each part of the interface, see the topics below:

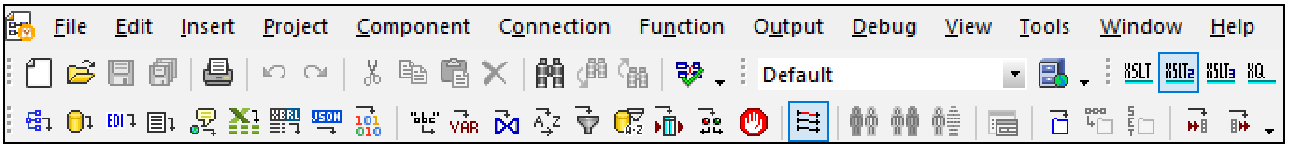
- Bars 23
- Windows 23
- Messages Window 26
- Panes 28

1.3.1 Bars

This topic gives an overview of the available bars.

Menu bar and toolbars

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



Application status bar

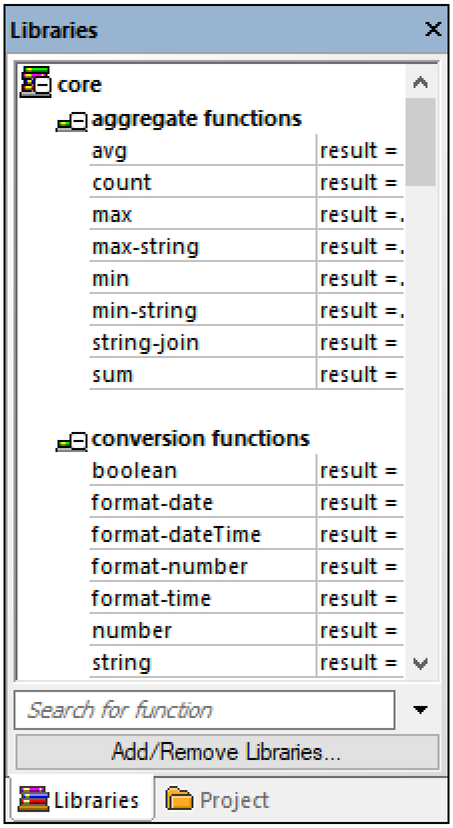
The application status bar appears at the bottom of the MapForce window and shows application-level information. Tooltips are displayed when you move the mouse over a toolbar button. If you are using the 64-bit version of MapForce, the application name appears in the status bar with the x64 suffix. There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

1.3.2 Windows

This topic gives an overview of the available windows.

Libraries window

The **Libraries** window lists the MapForce built-in functions organized by library. The list of available functions changes depending on the transformation language you select either from the **Output** menu or from the **Language Selection** toolbar. For more information, see Transformation Languages 19. If you have created user-defined functions or imported external libraries, they also appear in the **Libraries** window.



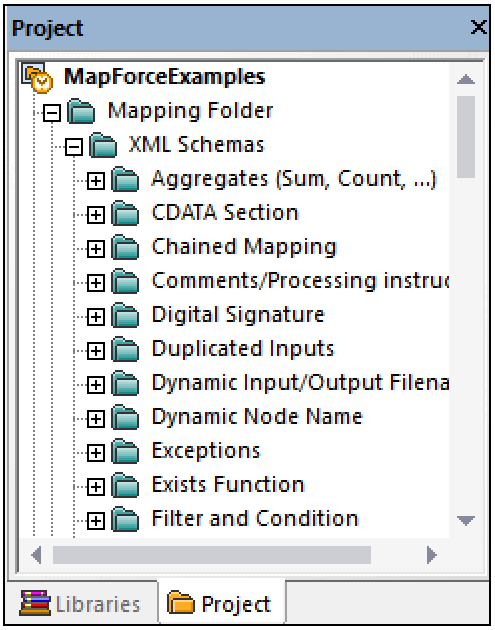
aggregate functions	
avg	result =
count	result =
max	result =.
max-string	result =.
min	result =.
min-string	result =.
string-join	result =
sum	result =

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

To search functions by name or by description, enter the search value in the text box at the bottom of the **Libraries** window. To find all occurrences of a function (within the currently active mapping), right-click the function and select **Find All Calls** from the context menu. You can also view the function data type and description directly from the **Libraries** window. For more information, see Functions 436.

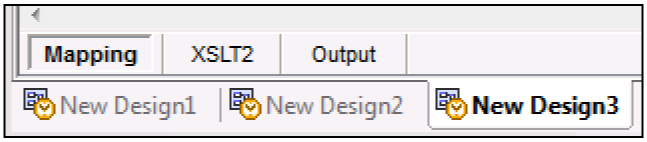
Project window (Enterprise and Professional editions)

MapForce supports the Multiple Document Interface and allows grouping your mappings into mapping projects. The **Project** window shows all files and folders that have been added to the project. Project files have a **`*.mfp`** (MapForce Project) extension. To search for mappings inside projects, click anywhere inside the **Project** window and press **CTRL + F**. For more information, see Projects 95.



Mapping window(s)

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



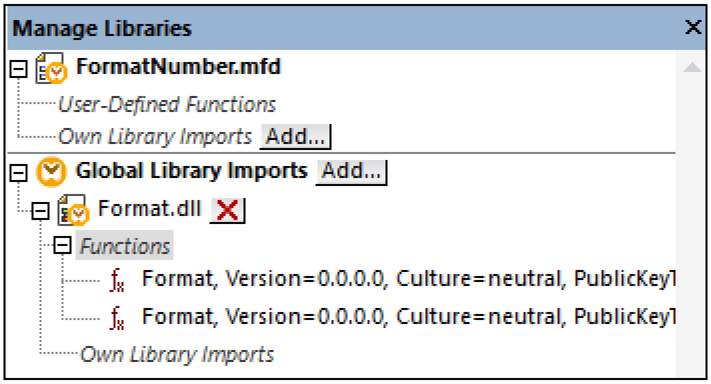
You can access Window management options using the menu command **Window | Windows**. The **Windows** dialog box allows you to perform various actions including activating, saving, closing, or minimizing open mapping windows. To select multiple windows in the **Windows** dialog box, click the required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

Manage Libraries window

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

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

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



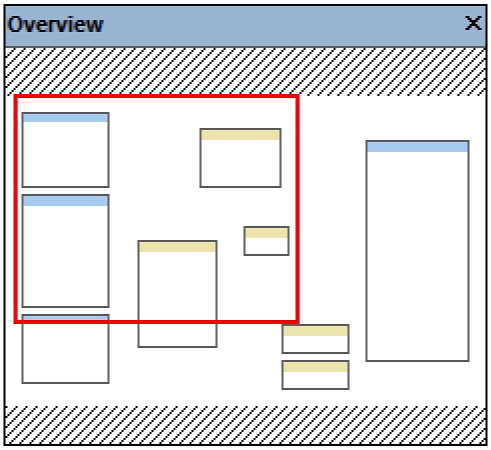
You can choose to view UDFs and libraries only for the mapping document that is currently active or for all open mapping documents. To view imported functions and libraries for all of the currently open mapping documents, right-click inside the window and select **Show Open Documents** from the context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

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

For more information, see Manage Function Libraries 438.

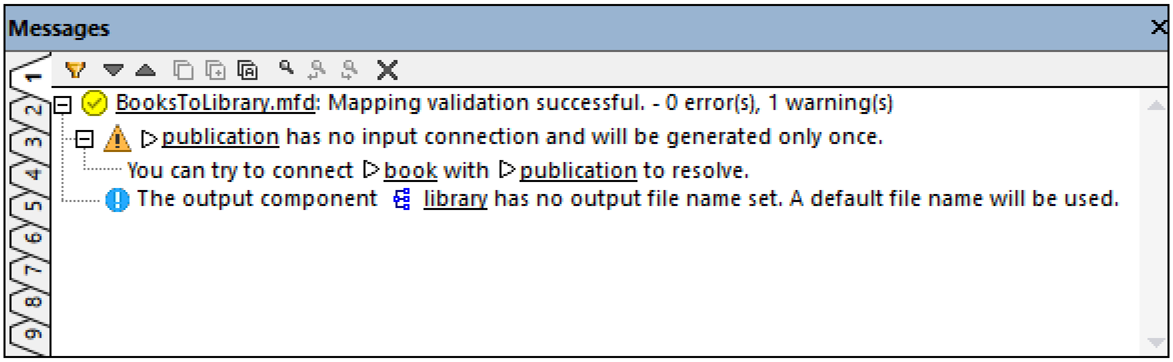
Overview window

The **Overview** window gives a bird's-eye view of the Mapping pane 28. Use it to navigate quickly to a particular location in the mapping area when the size of the mapping is very large. To navigate to a particular location in the mapping, click and drag the red rectangle.



1.3.3 Messages Window

The **Messages** window (*see screenshot below*) shows validation statuses, messages, errors, and/or warnings when you preview or validate 64 a mapping. Click the underlined text in the **Messages** window to see a component or structure which caused the information, warning, or error message.



BooksToLibrary.mfd: Mapping validation successful. - 0 error(s), 1 warning(s)
publication has no input connection and will be generated only once.
You can try to connect book with publication to resolve.
The output component library has no output file name set. A default file name will be used.

Validation status icons

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

Icon	Meaning
✅	Validation has completed successfully.
✔	Validation has completed with warnings.
❌	Validation has failed.

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

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

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

Message-related actions

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

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

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

1.3.4 Panes

This topic gives an overview of the available panes.

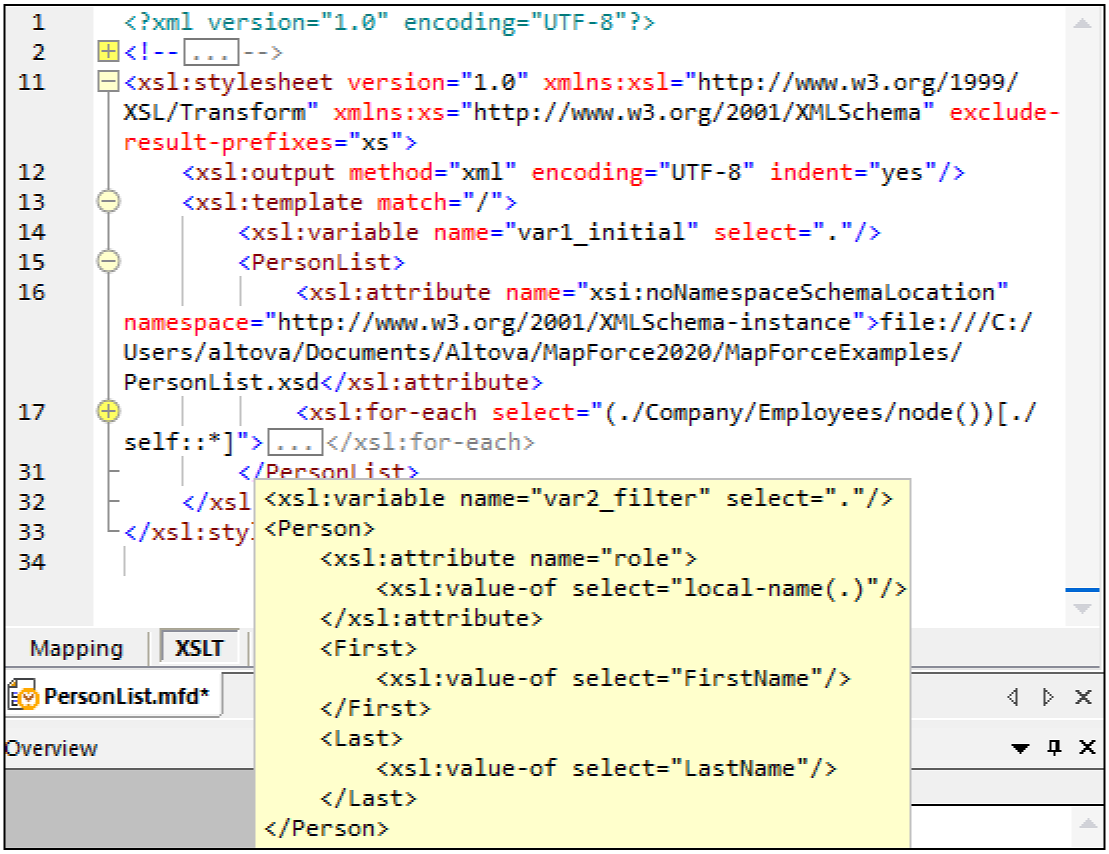
Mapping pane

The **Mapping** pane is the working area where you design mappings 64. You can add mapping components (e.g., files, schemas, constants, variables, and so on) to the mapping area from the **Insert** menu. For more information, see Add Components to Mapping 38. You can also drag functions from the **Libraries** window into the **Mapping** pane. For details, see Add a Function to the Mapping 436.

XSLT pane

The **XSLT** pane displays the XSLT transformation code generated from your mapping. To switch to this pane, select XSLT, XSLT 2 or XSLT3 as a transformation language 19 and click the tab with the same name.

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



```
<?xml version="1.0" encoding="UTF-8"?>
<!--...-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform" xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-
result-prefixes="xs">
    <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/">
        <xsl:variable name="var1_initial" select="."/>
        <PersonList>
            <xsl:attribute name="xsi:noNamespaceSchemaLocation"
namespace="http://www.w3.org/2001/XMLSchema-instance">file:///C:/
Users/altova/Documents/Altova/MapForce2020/MapForceExamples/
PersonList.xsd</xsl:attribute>
            <xsl:for-each select="(./Company/Employees/node())[./
self::*]">...</xsl:for-each>
        </PersonList>
    </xsl
</xsl:sty
```

```
<xsl:variable name="var2_filter" select="."/>
<Person>
    <xsl:attribute name="role">
        <xsl:value-of select="local-name(.)"/>
    </xsl:attribute>
    <First>
        <xsl:value-of select="FirstName"/>
    </First>
    <Last>
        <xsl:value-of select="LastName"/>
    </Last>
</Person>
```

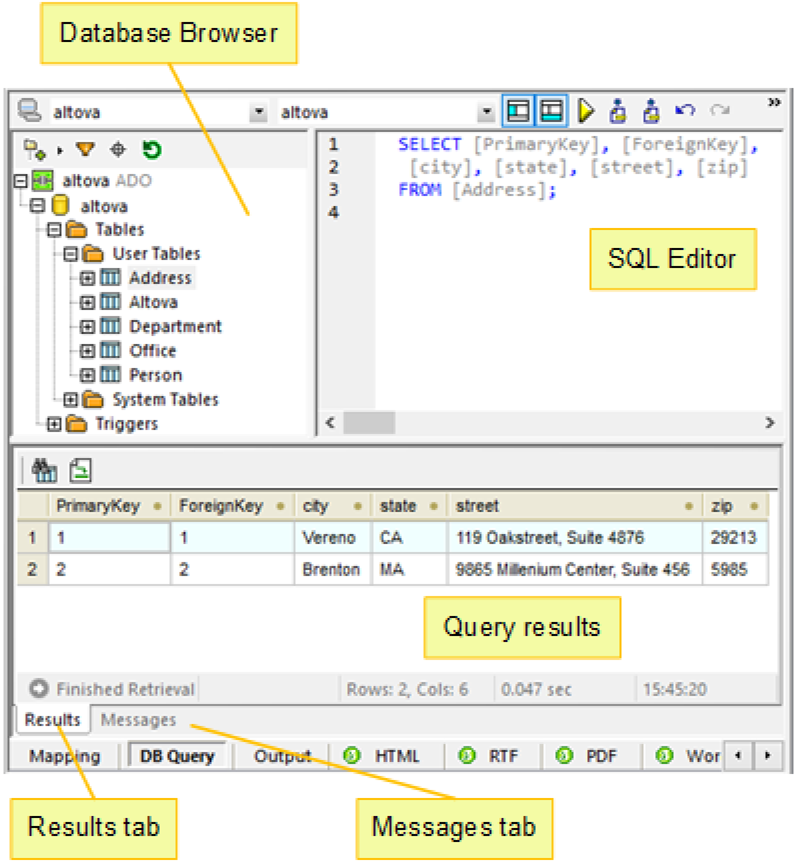
To configure the display settings, including the indentation, end of line markers, and others, right-click the pane and select **Text View Settings** from the context menu. Alternatively, click (**Text View Settings**) in the toolbar.

XQuery pane (Enterprise and Professional editions)

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

DB Query pane (Enterprise and Professional editions)

The **DB Query** pane allows you to directly query any major database. You can work with multiple active connections to different databases. For more information, see Browsing and Querying Databases 277.



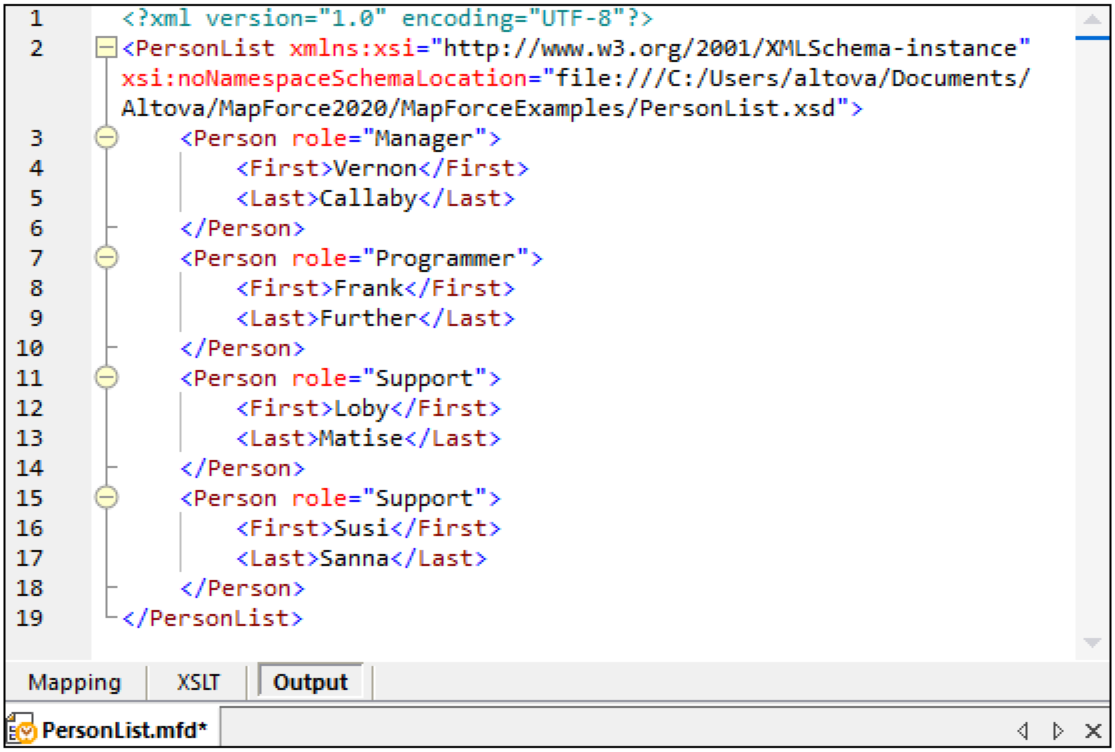
```
SELECT [PrimaryKey], [ForeignKey],
  [city], [state], [street], [zip]
FROM [Address];
```

	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    0.047 sec    15:45:20
```

Output pane

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



```
<?xml version="1.0" encoding="UTF-8"?>
<PersonList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///C:/Users/altova/Documents/
Altova/MapForce2020/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).

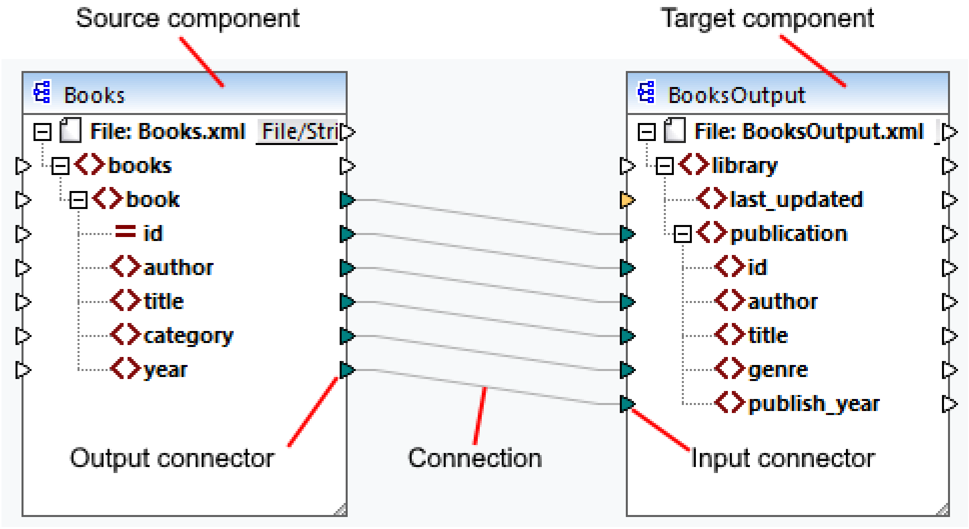
StyleVision output panes (Enterprise and Professional editions)

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

2 Mapping Fundamentals

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

The screenshot below illustrates the basic structure of a mapping:



New mapping

To create a new mapping, click 🗋 (**New**) in the toolbar. Alternatively, click **New** in the **File** menu. Then select **Mapping** and click **OK**. The next step is to add components 38 to the mapping and create connections 48.

Main parts of a mapping

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

Component

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

- Source and target components
- Structural 131 and transformation 345 components

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

- A *source* is located on the left of a target component. MapForce reads data from the source.
- A *target* is located on the right of a source. MapForce writes data to the target component.
- A *pass-through* component is a subtype of source and target components. A pass-through component acts both as a source and target. For more information, see Chained Mappings 115. Note that only structural components can be pass-through.

The second group (structural/transformation components) shows whether a component has a data structure or is used to transform data mapped from another component.

To find out more about components and component-related actions, see Components 34.

Connector

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

Connection

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

In this section

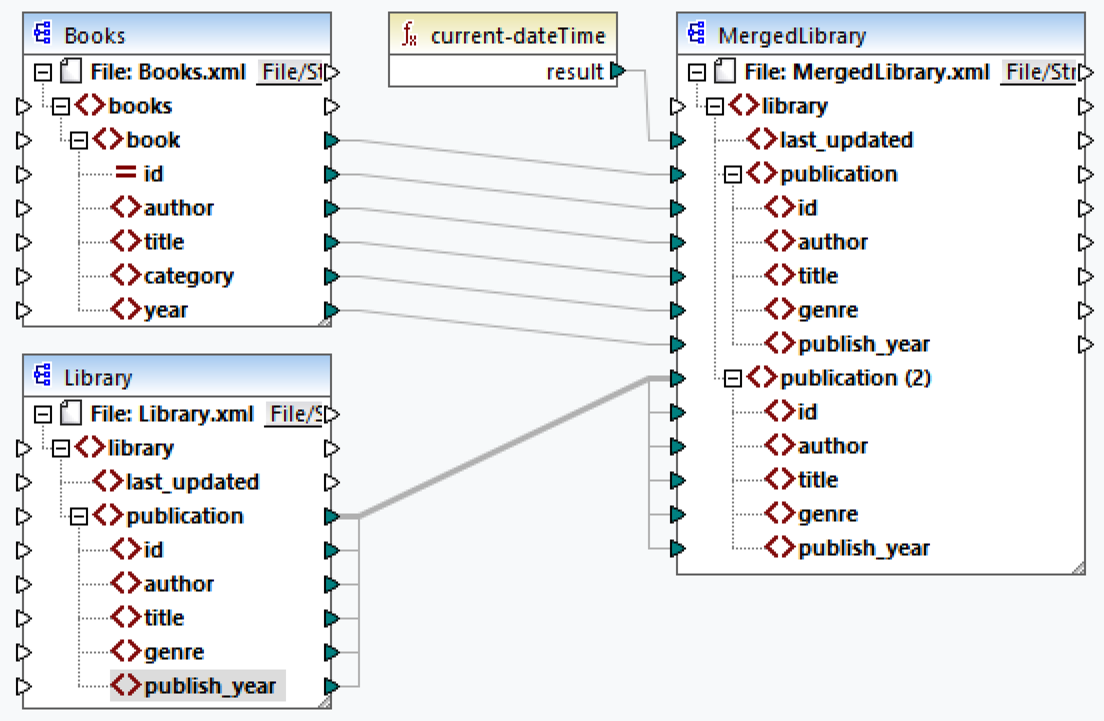
This section describes the most common MapForce tasks and concepts. The section is organized into the following subsections:

- Components 34
- Connections 48
- General Procedures and Features 64
- Basic Rules and Strategies 77
- Projects 95

2.1 Components

Components are the central elements of any mapping design in MapForce. Visually, components are represented as rectangular boxes in the mapping area. This topic gives an overview of structural and transformation components (*see example below*). The distinction is based on whether a component has a data structure or is used to transform data. See the description of these types in the subsections below. See also Mapping Fundamentals 32. Besides structural and transformation components, you can also add comments to your mapping (*see Comments below*).

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



Structural components

A structural component represents an abstract structure of your data (e.g., an XML file). The list of structural components that can be used as data sources and targets is given in Structural Components 131. Structural components can read data from some source(s), write data to some target(s), and store data at some intermediary stage in the mapping process (e.g., to preview the data). The table below gives an overview of structural components and their respective toolbar buttons.

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

Icon	Description
	Database component (*Professional and Enterprise editions* for SQL Databases; *Enterprise Edition* for NoSQL databases)
JSON	JSON component (*Enterprise Edition*)
	Microsoft Excel component (*Enterprise Edition*)
EDI	EDI component (*Enterprise Edition*)
XBRL	XBRL component (*Enterprise Edition*)
101 010	Protocol Buffers (*Enterprise Edition*)
PDF	PDF (*Enterprise Edition*)

Transformation components

Transformation components help you transform data 436, store an intermediate mapping result 360 for further processing, replace a value by another value 421, sort 402, group 561, join 373, and filter 408 your data. You can also add an exception 432, which stops the mapping process and displays an error when a condition defined by a filter occurs. The table below gives an overview of transformation components and their respective toolbar buttons.

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

Icon	Description
	Web service call to an HTTP API or to a WSDL-based service (*Enterprise Edition*)
	Web service call to the Shopify API or to a generic GraphQL API (*Enterprise Edition*)

Comments

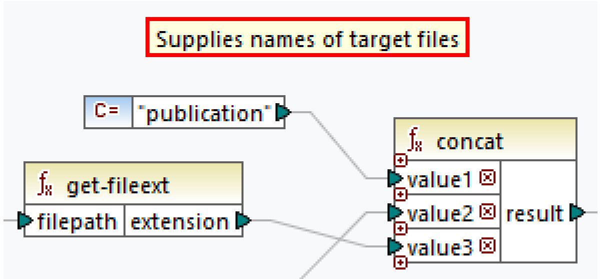
MapForce allows adding comments as standalone components and as notes under existing components. Comment text is available not only in a mapping, but it is also added to the generated mapping documentation[761]. In the generated mapping documentation, component comments are added directly below the corresponding component, and comment components are part of the *Remaining components* section.

Comment components

Comment components are free-standing boxes that can display multi-line text and cannot be connected to any other components. To add a comment component, you can select one of the following options:

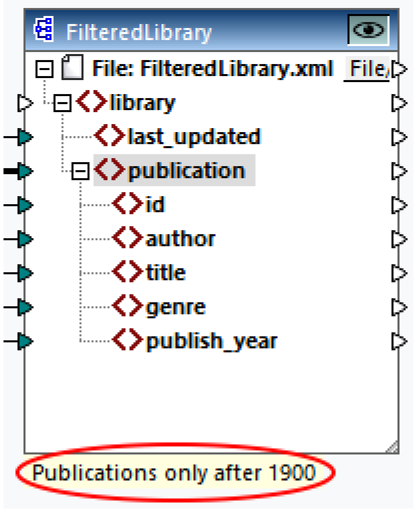
- Select the toolbar command, which opens a dialog where you can type your comment.
- Select the **Insert | Comment** menu command, which opens a dialog where you can type your comment.
- Double-click the empty area of your mapping, type the # sign, type a comment, and press **Enter**. The # sign will not appear in the comment box.

To move a comment, drag it to the desired location. To delete a comment, click on the comment box and press the **Delete** key. An example of a comment component is illustrated below (*red rectangular box*).



Component comments

Besides free-standing comment boxes, you can also add comments to any existing components. Such comments are displayed below the component (*circled red below*).



To add a comment under an existing component, you can choose one of the following options:

- Right-click inside the component and select **Edit Comment** from the context menu. This opens a dialog where you can enter your comment.
- Select a component to which you would like to add a comment. Then select **Edit Comment** in the **Component** menu. This opens a dialog where you can enter your comment.

The display of components' comments can be limited to a specified number of lines, which can be defined in the menu **Tools | Options | General | Mapping View**. For more information, see Options 1077.

To remove a component comment, take one of the following steps:

- Double-click the comment, delete all text, and click **Enter**.
- Right-click the comment or inside the component, select **Edit Comment** from the context menu, delete text, and click **OK**.

Edit comments
You can edit both types of comments in one of the following ways:

- Double-click the text of the comment and start editing directly in the box. Then press **Enter**.
- Right-click the comment box, edit the text in the **Edit Comment** dialog, and click **OK**. For component comments, you can also access the **Edit Comment** dialog by right-clicking inside the component and selecting the **Edit Comment** option in the context menu.

In this section

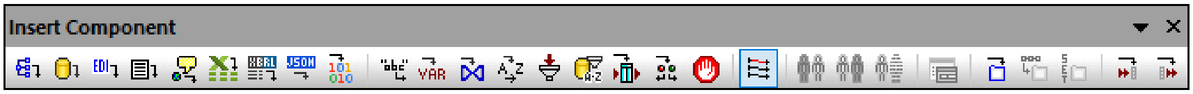
This section gives an overview of components and is organized into the following topics:

- Add Components 38
- Component Basics 40
- File Paths 43

2.1.1 Add Components

This topic explains how to add components to a mapping. To add a component, you need to create a new mapping design 32 first and then do one of the following:

- In the **Insert** menu, choose a component type (e.g., **XML Schema/File**).
- Drag a file from Windows File Explorer into the mapping area.
- Click the relevant button in the **Insert Component** toolbar (*see screenshot below*).



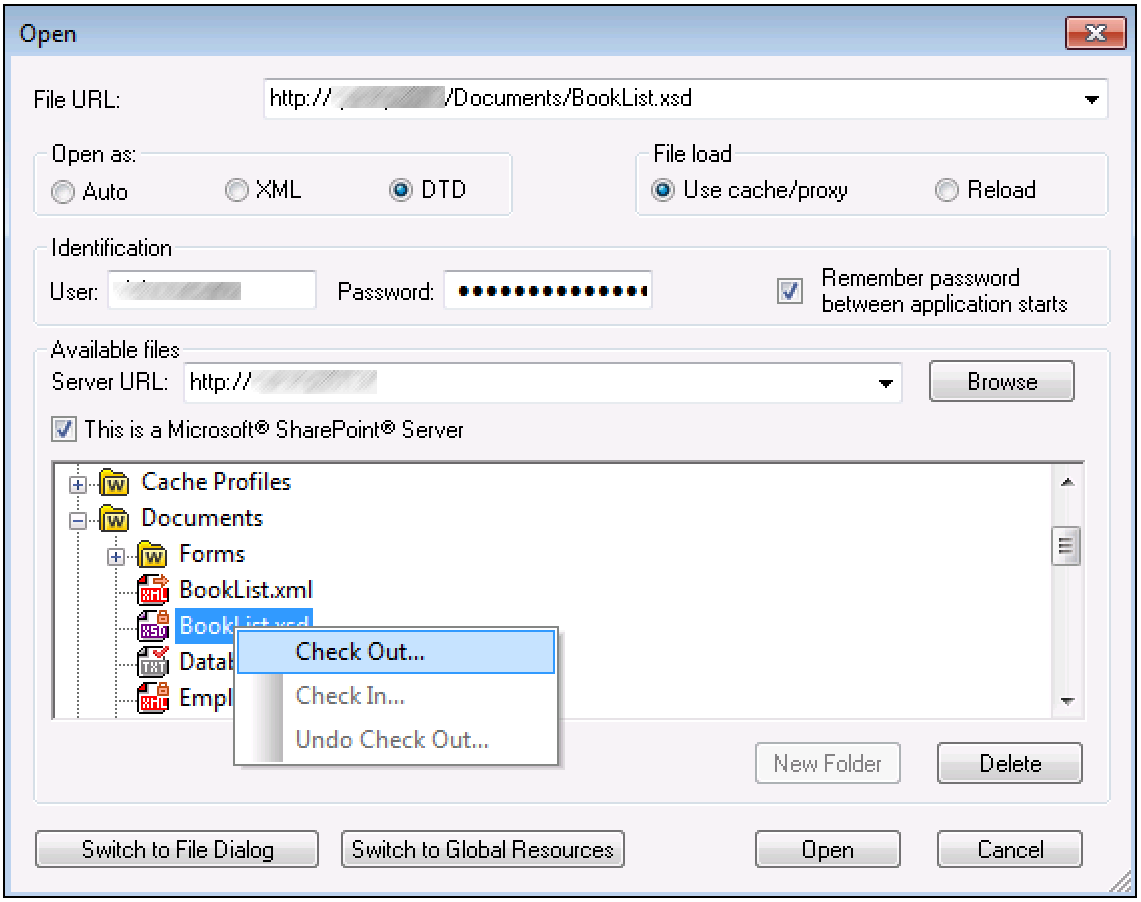
Each component type has a specific purpose and function. To get an overview of components, see Components 34. To find out more about data structures that can be used as sources and targets, see Structural Components 131. For information about MapForce components used to store data temporarily or to transform it, see Transformation Components 345.

When you want to add a structural component to your mapping, you can choose to add a local file, a component from a URL or from the list of global resources (*see subsections below*).

Add components from URL

Adding components from a URL is supported only for source components 32. The supported protocols are HTTP, HTTPS, and FTP. Depending on the type of data structure, the instructions on how to add a component from a URL may vary. For most data structures, the following instructions apply:

1. Select the component type you wish to add (e.g., **XML Schema/File**).
2. Click **Switch to URL** in the **Open** dialog box.
3. Enter the URL of the file in the *File URL* text box and click **Open** (*see below*).



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

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

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

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

- *Identification:* If the server requires password authentication, you will need to enter your user name and password. If you want your user name and password to be remembered next time, select the check box *Remember password between application starts*.

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

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

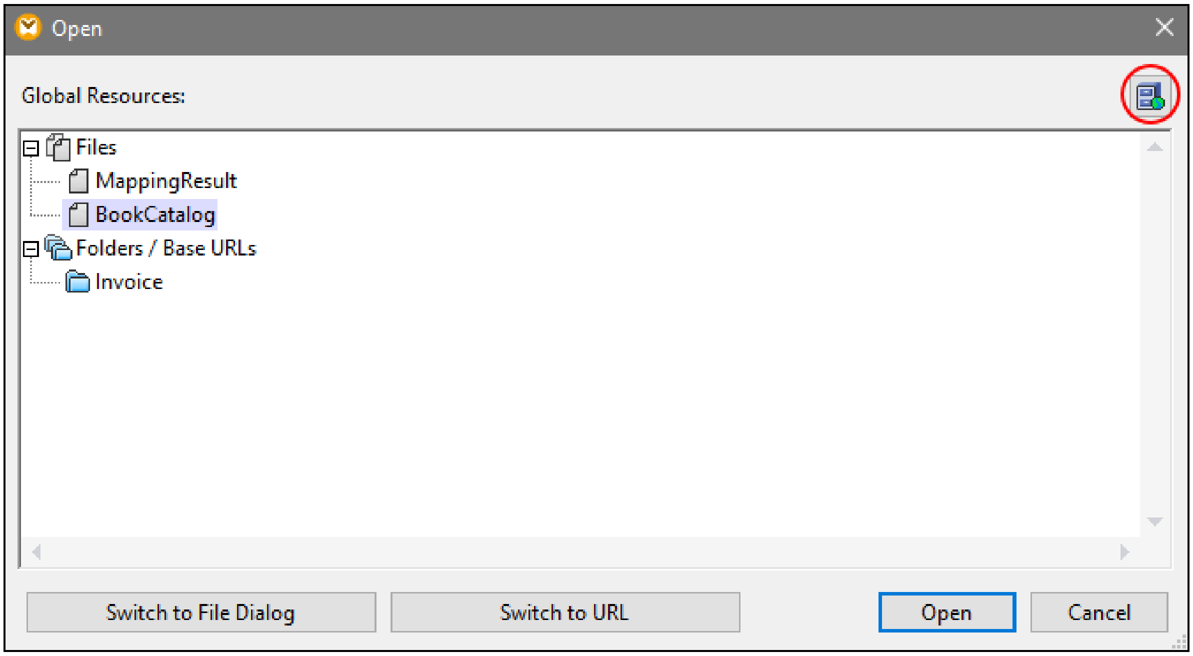
- *Switch to File Dialog:* Clicking this button will redirect you to the dialog in which you can select a local file.

- *Switch to Global Resources:* Clicking this button will redirect you to the dialog that allows selecting a global resource.

Add global resources

If you have defined a database (*Professional and Enterprise editions*), a file, or a folder as a global resource, you can add it to your mapping. For more information about global resources, see Altova Global Resources[815]. Depending on the type of data structure you are working with, the instructions on how to add a global resource may very. For most data structures, the following instructions apply:

1. Select the component type you wish to add (e.g., **XML Schema/File**).
2. Click **Global Resources** in the **Open** dialog box.
3. Select one of the resources from the list and click **Open** (*see below*).



If you want to add, edit, or delete global resources, click the **Manage Global Resources** icon (*circled red above*). The **Open** dialog for global resources allows you to switch back to the dialog with local files (**Switch to File Dialog**) or open a file from a URL (**Switch to URL**).

If you have created a database as a global resource and you want to add it to your mapping, follow the steps described in Global Resources[195].

2.1.2 Component Basics

This topic explains how to set, search and manipulate structural components[34]. For more information, see the subsections below.

Change component settings

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

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

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

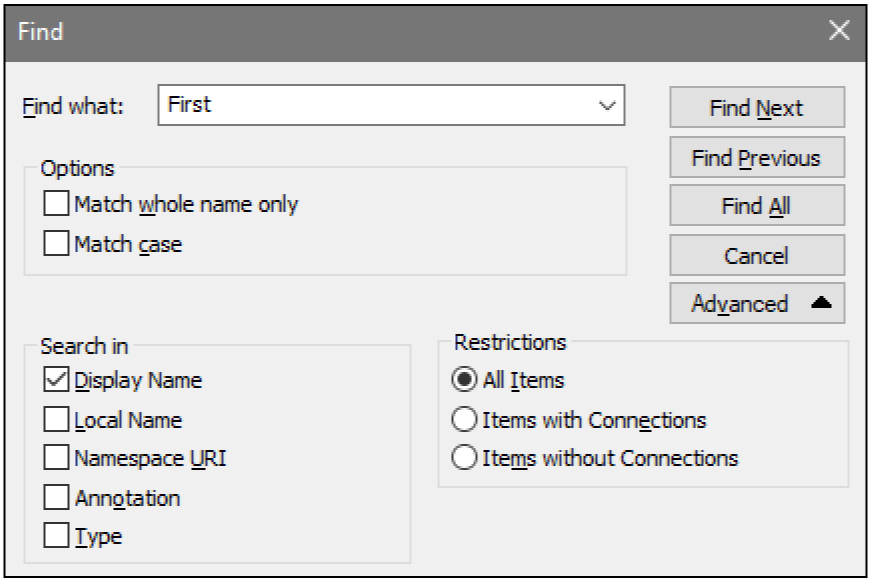
- XML Component Settings 133
- Database Component Settings 235
- CSV Component Settings 331
- Fixed-Length Field Component Settings 339

For any file-based component (e.g., an XML file) the File (*Basic Edition*) or File/String (*Professional and Enterprise editions*) button appears next to the root node. This button specifies advanced options for processing or generating multiple files in a single mapping. For more information, see Processing Multiple Input or Output Files 746. This button also allows setting options for parsing strings and serializing data to strings 753.

Search a component

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

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



Find what: First

Options

- ☐ Match whole name only
- ☐ Match case

Search in

- ☑ Display Name
- ☐ Local Name
- ☐ Namespace URI
- ☐ Annotation
- ☐ Type

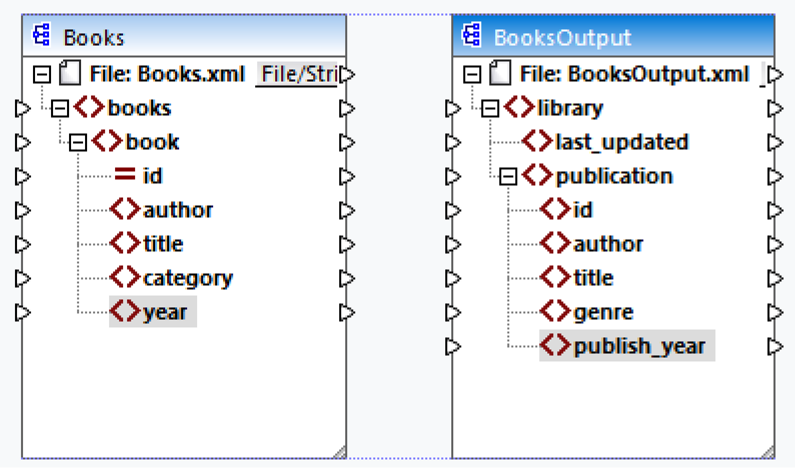
Restrictions

- ● All Items
- ○ Items with Connections
- ○ Items without Connections

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

Align components

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



To enable this option, take the steps below:

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

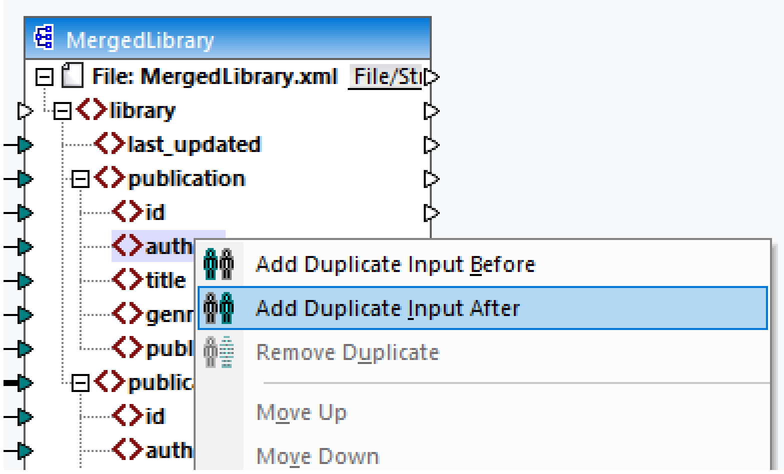
Duplicate input

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

There are two ways of duplicating input: (i) selecting **Add Duplicate Input Before/After** from the context menu and (ii) connecting a source node with a target node that is already connected to a different node. The first option is described below. Information about the second option can be found in the second tutorial 110.

Add Duplicate Input Before/After

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



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

2.1.3 File Paths

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

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

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

- Relative and Absolute Paths 43
- Paths in Execution Environments 46

2.1.3.1 Relative and Absolute Paths

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

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

- Data files (e.g., XML, JSON, CSV, etc.)
- Schema files (relevant to components with schemas)

- Files used by complex input or output parameters of user-defined functions and variables of complex type
- StyleVision Power Stylesheet (**`*.sps`**) files, which are used to format data for outputs such as PDF, HTML and Word (*Professional and Enterprise editions*)
- Database files (*Professional and Enterprise editions*)
- Schema files referenced by database components which support XML fields (*Professional and Enterprise editions*)

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

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

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

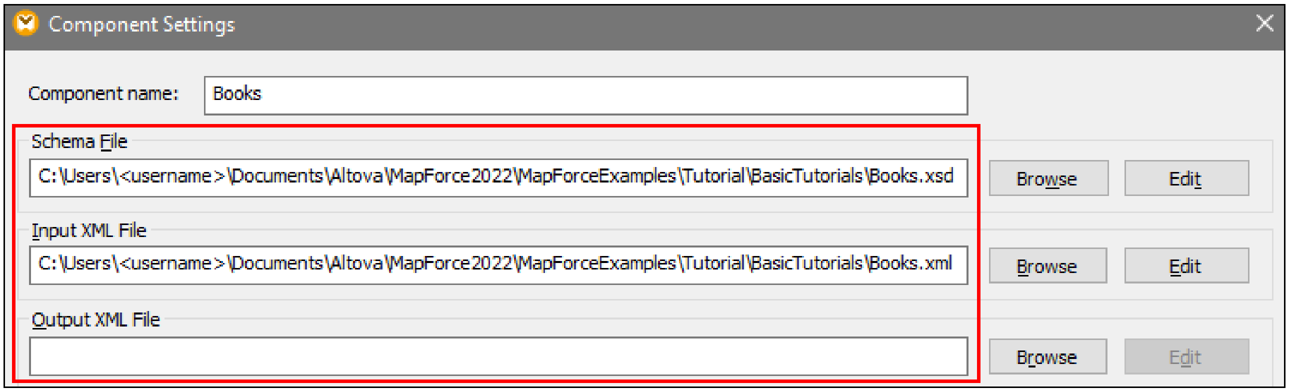
Broken path references cause MapForce to highlight the affected component in red. The solution is to double-click the red component header and update any broken path references in the **Component Settings** dialog box. See also Change Component Settings 41.

Example 1: XML component

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

1. Create a new mapping 32 and add a structural component 38: e.g., an XML file with an XML schema assigned to it.
2. Double-click the header of the component to open the **Component Settings** dialog box.
3. Check the box *Save all file paths relative to MFD file* at the bottom of the **Component Settings** dialog box.
4. Save your mapping.
5. You can now open **Component Settings** again to check the relative paths in the relevant text fields.

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



Component Settings

Component name: Books

Schema File

C:\Users\<username>\Documents\Altova\MapForce2022\MapForceExamples\Tutorial\BasicTutorials\Books.xsd

Input XML File

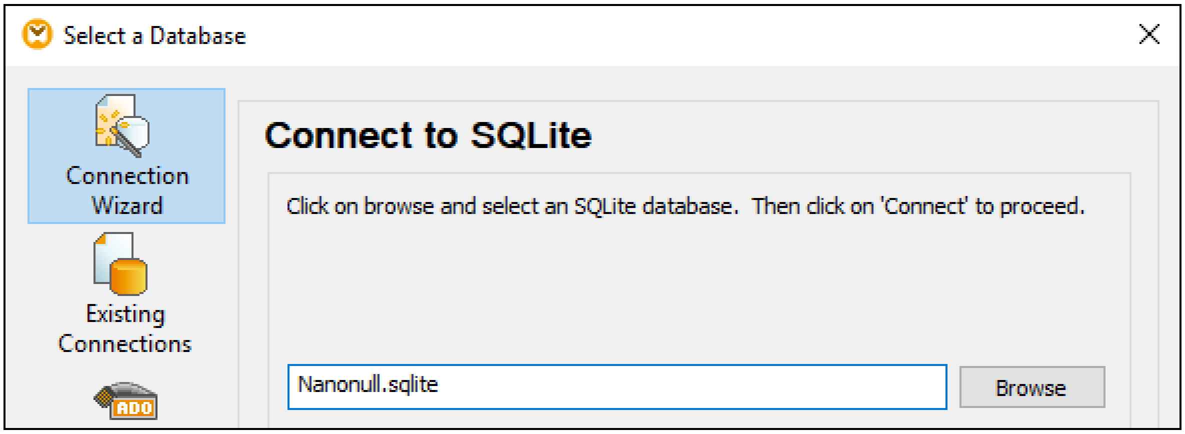
C:\Users\<username>\Documents\Altova\MapForce2022\MapForceExamples\Tutorial\BasicTutorials\Books.xml

Output XML File

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

Example 2: Database component (Professional and Enterprise editions)

When you add a database file such as Microsoft Access or SQLite to the mapping, you can enter a relative path instead of an absolute one in the **Select a Database** dialog box (*see screenshot below*). Before entering relative file paths, make sure to save the `.mfd` file first. If you want to change the path of a database component which is already in the mapping, click **Change** in the **Component Settings** dialog box. For more information about connecting to a database source, see Start Database Connection Wizard 170.



Select a Database

Connect to SQLite

Click on browse and select an SQLite database. Then click on 'Connect' to proceed.

Nanonull.sqlite

Note: When you generate program code, compile MapForce Server execution files (`.mfx`), or deploy the mapping to FlowForce Server, a relative path will be converted to an absolute path if you select the check box *Make paths absolute in generated code* in the mapping settings 75. To find out more, see About Paths in Generated Code 46.

2.1.3.2 Paths in Execution Environments

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

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

** Languages available in MapForce Professional and Enterprise editions*

Relative path to absolute path
When you generate program code, compile MapForce Server execution files (**.mfx**), or deploy the mapping to FlowForce Server, a relative path will be converted to an absolute path if you select the check box **Make paths absolute in generated code** in the mapping settings 75.

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

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

Library paths in generated code

Mapping files may optionally contain path references to different libraries. For example, you can import user-defined functions from another mapping file, from custom XSLT, XQuery*, C#* or Java* libraries, or from **.mff*** (MapForce Function) files. For more information, see Managing Function Libraries 438.

** Features available in MapForce Professional and Enterprise editions*

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

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

1. Open the mapping settings 75.
2. Select the check box **Reference libraries with paths relative to generated XSLT/XQuery file**. Make sure that the XSLT or XQuery library file exists at that path.

2.2 Connections

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

Create, copy, delete a connection

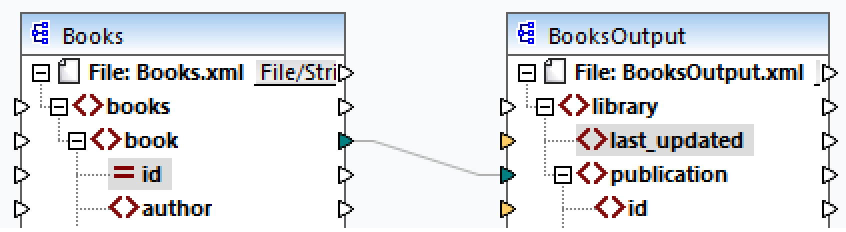
To create a connection between two items, press and hold the output connector 33 of a source node and drag it to a destination node. An input connector accepts *only one* connection. If you try to add a second connection to the same input, MapForce will ask if you want to replace the connection with a new one or duplicate the input item 42. An output connector can have several connections, each to a different input.

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

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

Mandatory inputs

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



Missing parent connections

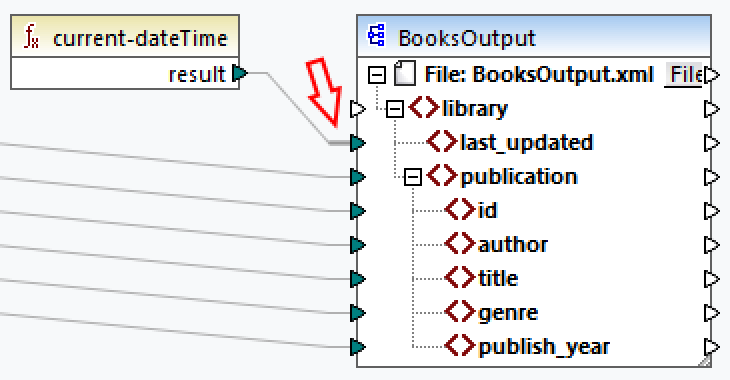
When you create connections between source and target node manually, MapForce analyzes possible mapping outcomes. If you connect two child nodes without connecting their parent nodes, you will see a notification message that suggests connecting the parent of the source node with the parent of the target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

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

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

Move a connection

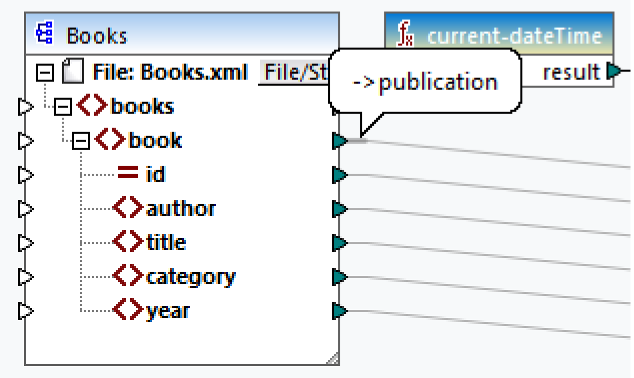
To move a connection to a different node, press and hold the thick section at the end of the connection (*see screenshot below*) and drag it to the selected destination.



See connection tooltips

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

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



Change connection settings

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

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

For more information, see Connection Settings 58.

Highlight connections selectively

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

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

Only direct connections

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

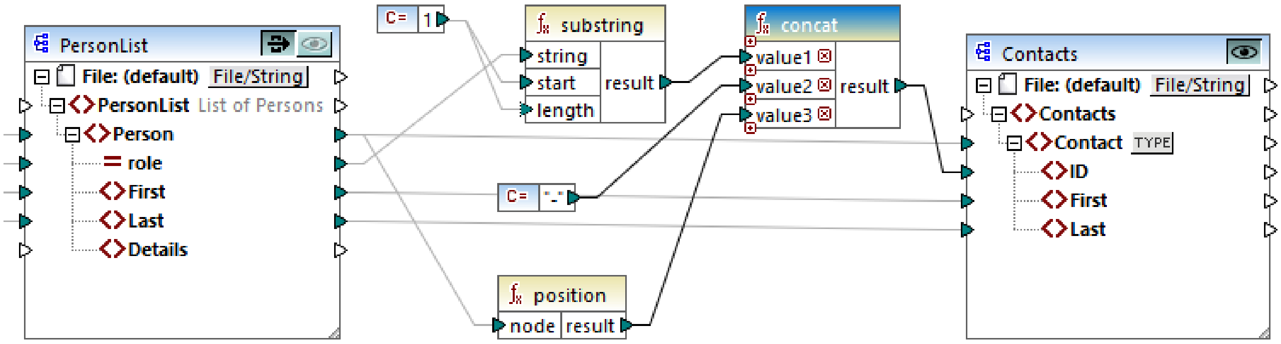
Direct and indirect connections

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

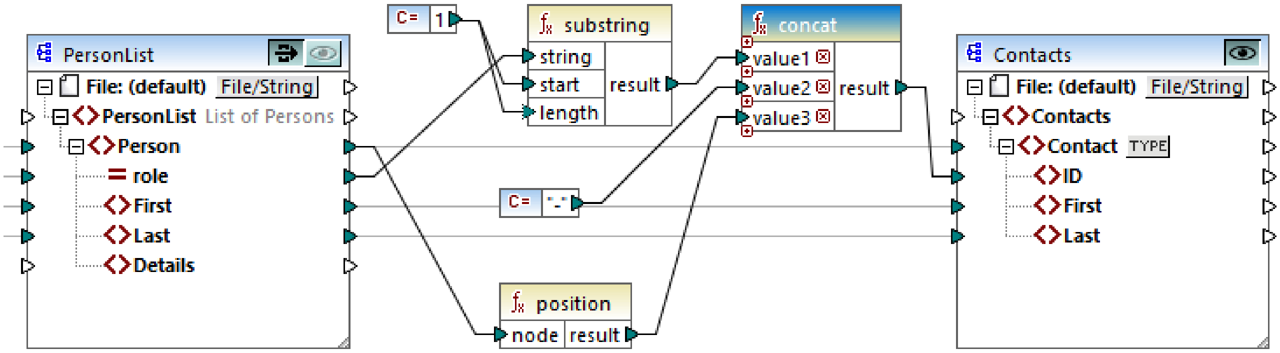
To understand how these two options work, see the example below.

Example

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



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



With the **Source-to-target Connections** button pressed, some other connections have become black: (i) the connections of the **substring** function to the **1** constant and the PersonList component, and (ii) the connection of the **position** function to the PersonList component. However, the connections between the PersonList component and its preceding component remain light gray. Thus, when you press the **Source-to-target Connections** button and click on a component, you can trace the component's direct connections. If the selected component is connected to some transformation components 35 (e.g, functions, constants, filters, sort components, SQL-NoSQL-WHERE/ORDER components, if-else conditions, value-maps), you will also be able to see their connections up to the structural components 34 (such as PersonList above), variables, join components, or web service functions, to which these transformation components are connected.

In this section

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

- Connection Types 51
- Connection Settings 58
- Connection Context Menu 59
- Faulty Connections 60
- Keep Connections after Deleting Components 62

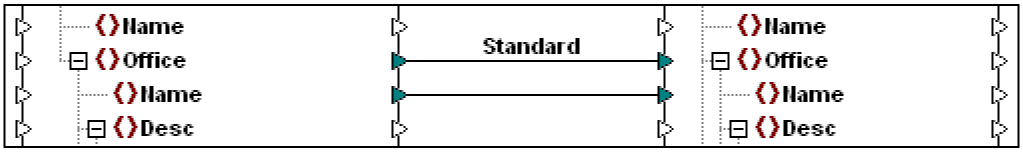
2.2.1 Connection Types

The following connection types are available in MapForce:

- Target-driven connections 51 (Standard)
- Source-driven connections 52 (Mixed Content)
- Matching-children connections 54
- Copy-all connections 56 (Copy Child Items)

Target-driven vs. source-driven connections

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



Target-driven connections might not be suitable when you want to map XML nodes with mixed context (child nodes and text). In this case, a source-driven connection 52 is recommended: The order of nodes in the output is determined by the *source* schema.

Matching-children and copy-all connections

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

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

2.2.1.1 Source-Driven Connections

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

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

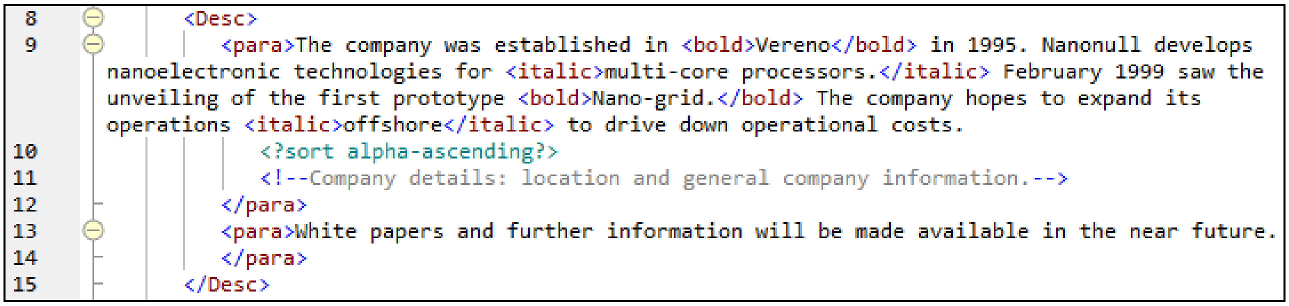
Note: In order to accept mixed content, target components must have mixed-content nodes.

Mapping mixed content

This topic explains how to map mixed content using a source-driven connection. You will need the following files: **`Tut-OrgChart.xml`**, **`Tut-Orgchart.mfd`**, **`Tut-Person.xsd`**, and **`Tut-OrgChart.xsd`**, which are available in the Tutorial folder 17.

Source XML instance

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

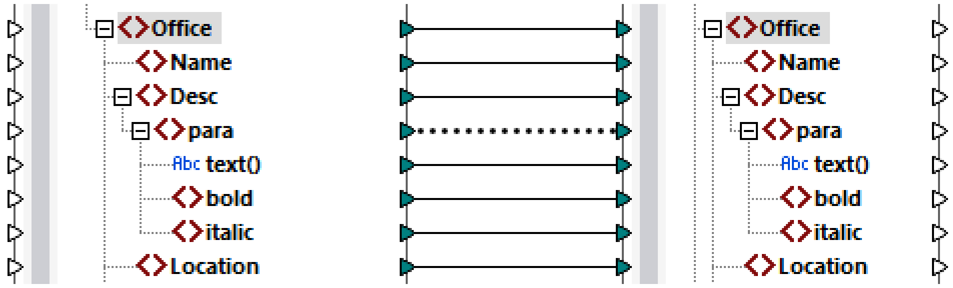


```
        <Desc>
            <para>The company was established in <bold>Vereno</bold> in 1995. Nanonull develops
nanoelectronic technologies for <italic>multi-core processors.</italic> February 1999 saw the
unveiling of the first prototype <bold>Nano-grid.</bold> The company hopes to expand its
operations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

Mapping the <para> element

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

1. Select the menu command **Connection | Auto Connect Matching Children**, which will connect matching child nodes [54] automatically. Alternatively, you can manually map the `<para>` node with its child nodes.
2. Connect the `<para>` item in the source component with the `<para>` item in the target component. A message box will ask if you would like to define the connection as source-driven.
3. Click **Yes** to create a mixed-content connection.
4. Click the **Output** pane to see the result of the mapping. Click the button (**Wrap**) in the **Output** pane toolbar to view the full (i.e., not going beyond the scroll bar) code listing in the **Output** pane. The mixed content of the `<para>` node has been mapped in the same order it appears in the XML source file.



Processing instructions and comments

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

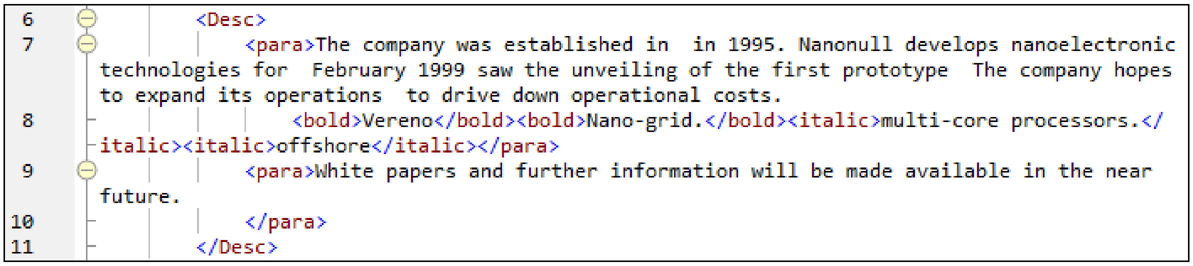
1. Right-click the mixed-content connection (dotted line) and select **Properties**.
2. Under **Source-Drive (Mixed content)**, select the check boxes **Map Processing Instructions** and/or **Map Comments**.

Target-driven connections with mixed content

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

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

2. Press the toolbar button (Auto Connect Matching Children[54]). Clear the check box **Create copy-all connections** in the settings for matching-children connections[54]. This will prevent MapForce from creating copy-all connections[56] automatically.
3. Create a connection between the `para` node in the source and the `para` node in the target. A message will ask if you would like to define the connections as source-driven. Click **No**. This creates a target-driven connection.
4. Click the **Output** pane to see the result of the mapping (*screenshot below*).



```
        <Desc>
            <para>The company was established in  in 1995. Nanonull develops nanoelectronic
technologies for  February 1999 saw the unveiling of the first prototype  The company hopes
to expand its operations  to drive down operational costs.
                <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>
```

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

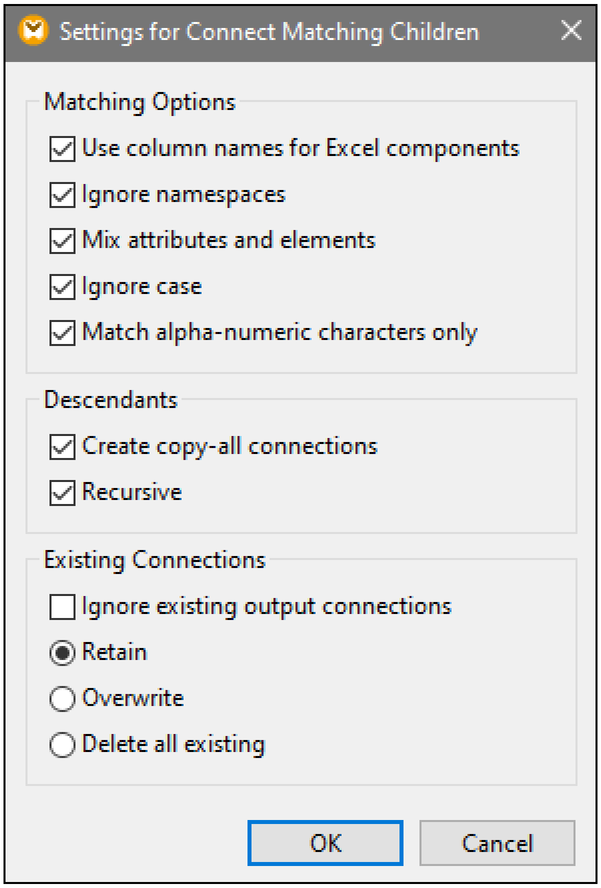
2.2.1.2 Matching-Children Connections

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

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

Settings for matching-children connections

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



Settings for Connect Matching Children

Matching Options

☑ Use column names for Excel components

☑ Ignore namespaces

☑ Mix attributes and elements

☑ Ignore case

☑ Match alpha-numeric characters only

Descendants

☑ Create copy-all connections

☑ Recursive

Existing Connections

☐ Ignore existing output connections

● Retain

○ Overwrite

○ Delete all existing

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

Matching Options
The *Matching Options* section enables you to relax matching criteria and define how to compare node names. The following options are available:

- *Use column names for Excel components:* This option applies only to Excel components (*Enterprise Edition*). This option means that user-defined column names (e.g., `Company`) will be used for comparison instead of column reference names (e.g., A, B, C). User-defined column names are set in the **Select Range of Cells** dialog and appear as annotations in an Excel component.
- *Ignore namespaces:* Matching children will be connected regardless of the namespaces of child nodes.
- *Mix Attributes and Elements:* This option allows creating connections between attributes and elements that have the same names. For example, a connection is created if two `ID` nodes exist, even though one is an element, and the other is an attribute.
- *Ignore Case:* Matching children will be connected regardless of the case of child node names.
- *Match alpha-numeric character only:* When this option is enabled, only digits and letters will be compared. Other characters such as spaces, commas, dots, etc. will be discarded before comparison.

Descendants
The *Descendants* section defines how to proceed with child nodes. The following options are available:

- *Create copy-all connections:* This setting is active by default. It creates (if possible) a copy-all connection 56, which maps data between nodes with child nodes that are similar or the same. A copy-all connection is represented by one thick line, which prevents clutter and makes the mapping easier to understand.
- *Recursive:* This option creates new connections between any matching nodes if they have the same names. It does not matter how deep the nodes are nested in the tree.

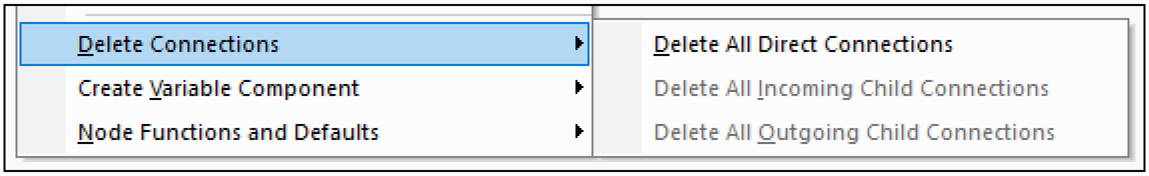
Existing Connections
The *Existing Connections* section specifies what to do with existing connections. The following options are available:

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

Delete connections as a group

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

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


Delete Connections
Create Variable Component
Node Functions and Defaults
Delete All Direct Connections
Delete All Incoming Child Connections
Delete All Outgoing Child Connections

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

2.2.1.3 Copy-All Connections

Copy-all connections map data between nodes with child nodes that are similar or the same. Copy-all connections are possible only for identical formats (e.g., JSON to JSON or XML to XML). This principle also applies to all text components: flat files, FlexText and EDI files. Since these formats are all text files, you can combine any of them and create a copy-all connection between EDI and FlexText files, for example.

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

Create copy-all connections automatically

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

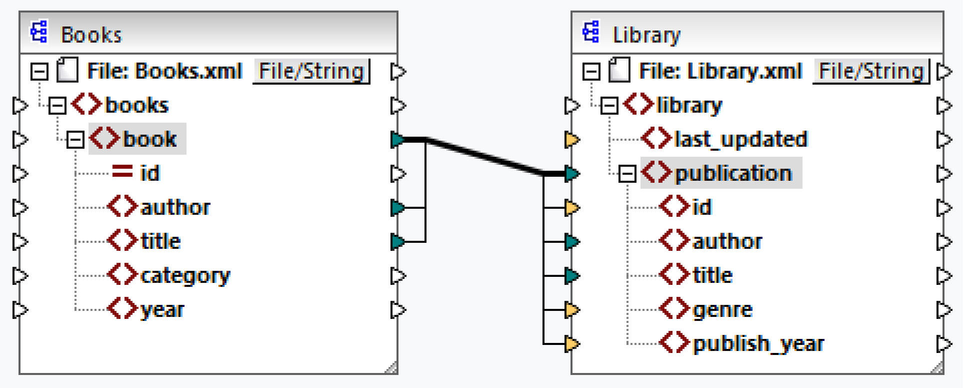
1. Go to the **Connection** menu.
2. Click **Settings for Connect Matching Children**.
3. Check the box **Create copy-all connections** and click **OK**.
4. Press the toolbar button **Toggle auto connect of children**. Alternatively, go to the **Connection** menu and click **Auto Connect Matching Children**.

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

Create copy-all connections manually

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

1. Add a source file: Click **XML Schema/File** in the **Insert** menu and browse for **Books.xml** located in the BasicTutorials folder 17.
2. Add a target file: Click **XML Schema/File** in the **Insert** menu and browse for **Library.xsd** located in the same folder as **Books.xml**. Click **Skip** when MapForce suggests adding an XML sample file.
3. Map the <book> node of the **Books** component to the <publication> node of the **Library** component. As the structures of the <book> and <publication> elements do not fully coincide, the copy-all connection is not created. Instead, the **Auto Connect Matching Children** function automatically connects all the child nodes with the same name, which is discussed in Tutorial 1 106.
4. To change the automatic connection to a copy-all connection, right-click the connection between <book> and <publication> and select **Copy-All (Copy Child Items)** from the context menu.
5. A pop-up window will suggest replacing the existing connections with a copy-all connection. Click **OK**. Now the source and target have a copy-all connection (*see screenshot below*).



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

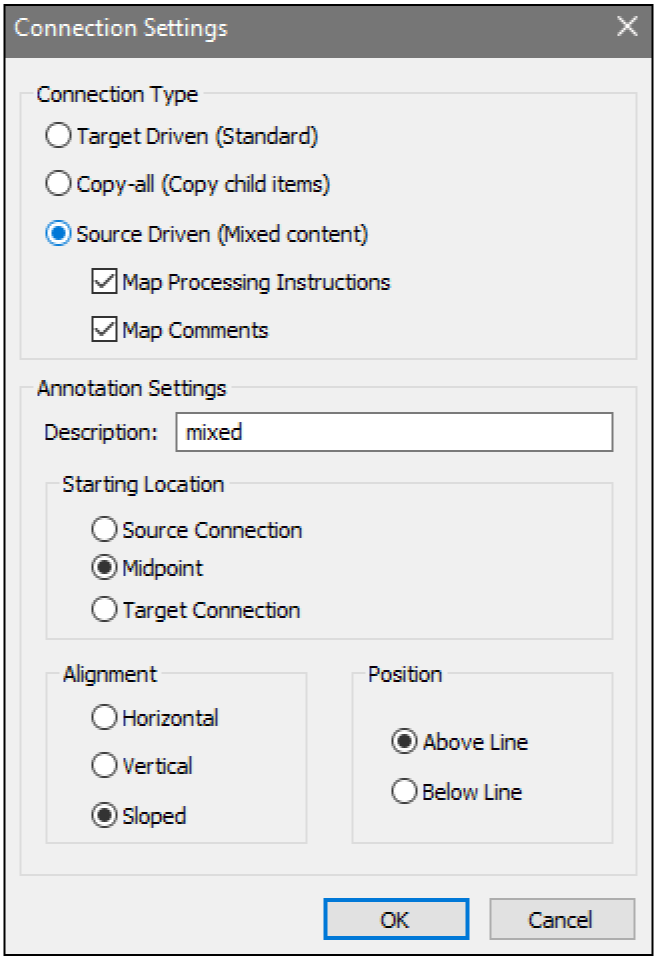
Duplicating input 42 only makes sense if you want the target to accept data from more than one input, which is not required here. If you choose to replace the copy-all connection, a message box prompts you again to resolve or delete the copy-all connection. Click **Resolve copy-all connection** if you want to replace the copy-all connection with individual target-driven connections[51]. If you prefer to remove the copy-all connection completely, click **Delete child connections**.

Important

When you create a copy-all connection between a schema and a parameter of a user-defined function[457], the two components must be based on the same schema. It is not necessary that they both have the same root elements, however.

2.2.2 Connection Settings

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



Connection Settings

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

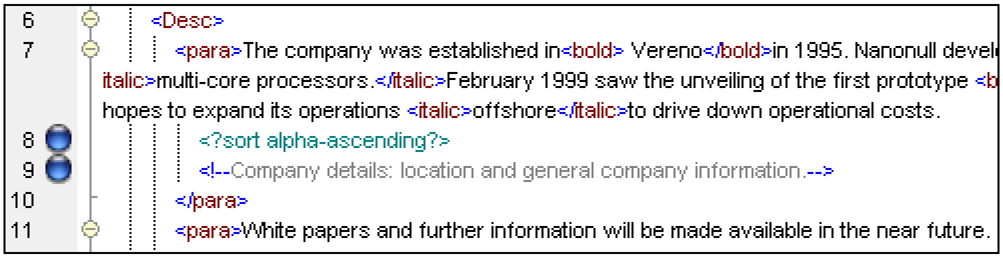
Alignment
○ Horizontal
○ Vertical
● Sloped

Position
● Above Line
○ Below Line

⊟ Connection types

You can choose one of the connection types described below:

- Target-driven (Standard)[51] connections are suitable for most mapping scenarios.
- Copy-all (Copy child items)[56] connections: If a source and target components have identical or similar nodes with matching child nodes, a copy-all connection will automatically be created between these matching nodes.
- Source-driven (mixed content)[52] connections map mixed content (text and child nodes) in the same order as in the XML source file. If you select **Map Processing Instructions** and/or **Map Comments**, you will be able to include these data groups in the output file (*see screenshot below*).

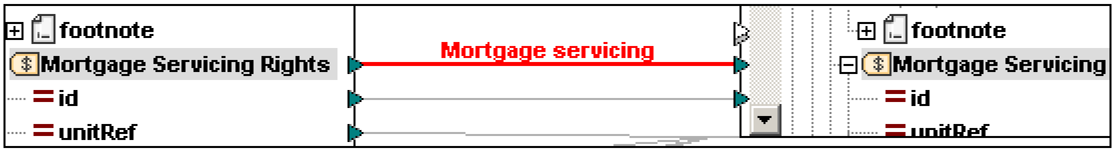


```
<Desc>
  <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull devel
italic>multi-core processors.</italic>February 1999 saw the unveiling of the first prototype <b
hopes to expand its operations <italic>offshore</italic>to drive down operational costs.
    <?sort alpha-ascending?>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

Annotation settings

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

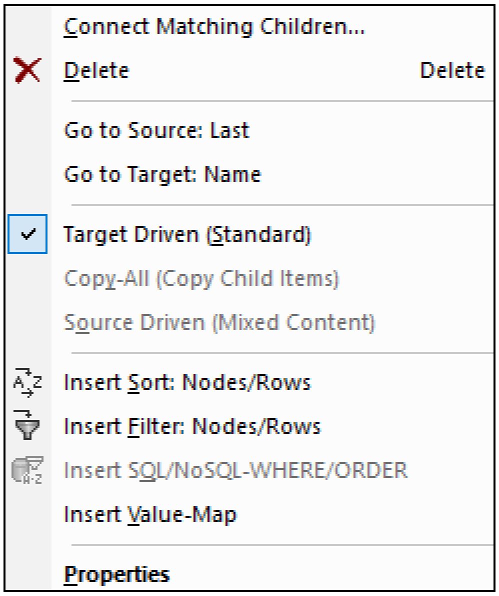
1. Right-click the connection and select **Properties** from the context menu. Alternatively, double-click the connection.
2. Enter the name of the selected connection in the **Description** field. This enables all the options in the **Annotation Settings** section.
2. Use the remaining groups to define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



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

2.2.3 Connection Context Menu

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


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

For more information, see the subsections below.

General settings

- *Connect Matching Children:* Opens the Connect Matching Children[54] dialog box. This command is enabled when the connection is allowed to have matching children.
- *Delete:* Deletes the selected connection.
- *Go to Source: <item name>:* Highlights the output connector[33] of the selected connection.
- *Go to Target: <item name>:* Highlights the input connector[33] of the selected connection.

Connection types

See details about connection types in Connection Types[51] and Connection Settings[58].

Insert commands

- *Insert Sort: Nodes/Rows:* Adds a sort[402] component between a source node and a target node.
- *Insert Filter: Nodes/Rows:* Adds a filter[408] component between a source node and a target node.
- *Insert SQL/NoSQL-WHERE/ORDER:* Adds an SQL/NoSQL-WHERE/ORDER component between a source node and a target node (*Professional and Enterprise editions*). For details, see Filter and Sort Database Data[413].
- *Insert Value-Map:* Adds a value-map[421] between a source node and a target node.

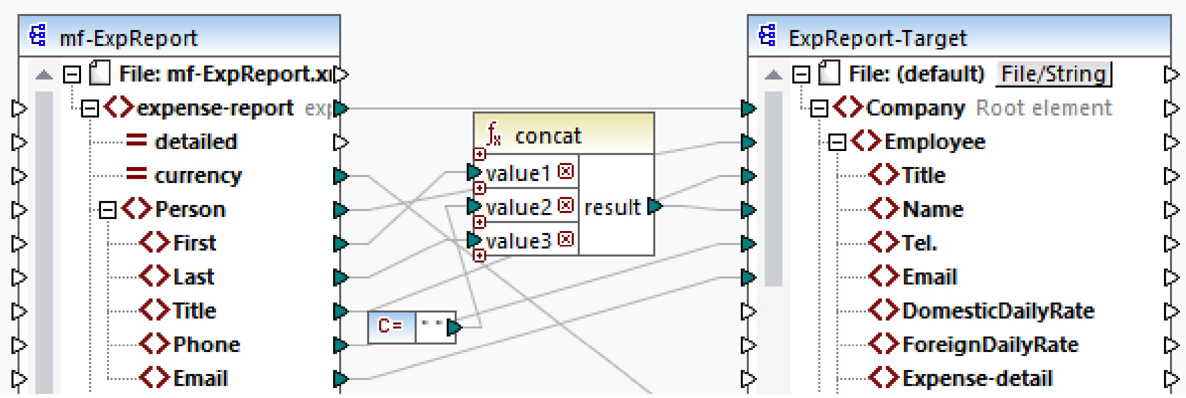
Properties

Opens the Connection Settings[58] dialog box.

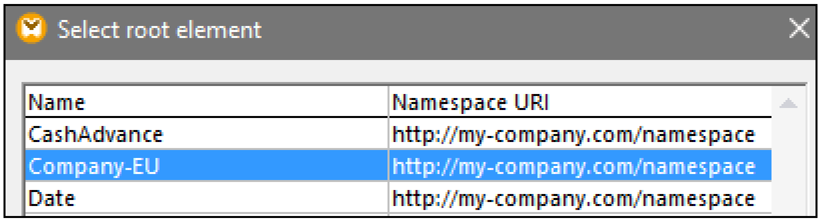
2.2.4 Faulty Connections

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

1. Open **`Tut-ExpReport.mfd`** available in the Tutorial folder[17]. The portion of this mapping is shown below.

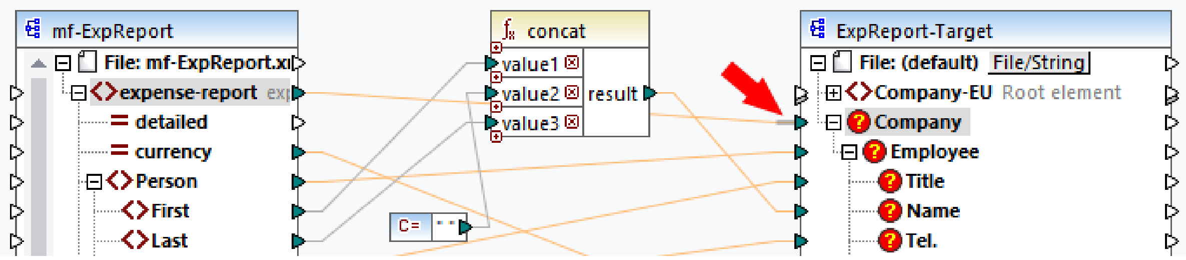


2. Open **`ExpReport-Target.xsd`** in an editor (e.g., Altova XMLSpy) and change the `Company` root element in the target schema to `Company-EU`. You do not need to close MapForce.
3. After you have edited the root element of the target schema**,** the **Changed files** prompt appears in MapForce. Click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


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

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



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

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

Alternative solution

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

Faulty connections in databases (Professional and Enterprise editions)

If your database component has faulty connections, you will need to change the component settings 41. Clicking the **Change** button in the **Component Settings** dialog box allows you to select a different database or change tables in your database component. All valid/correct connections and relevant database data will be kept if you select a database with the same structure.

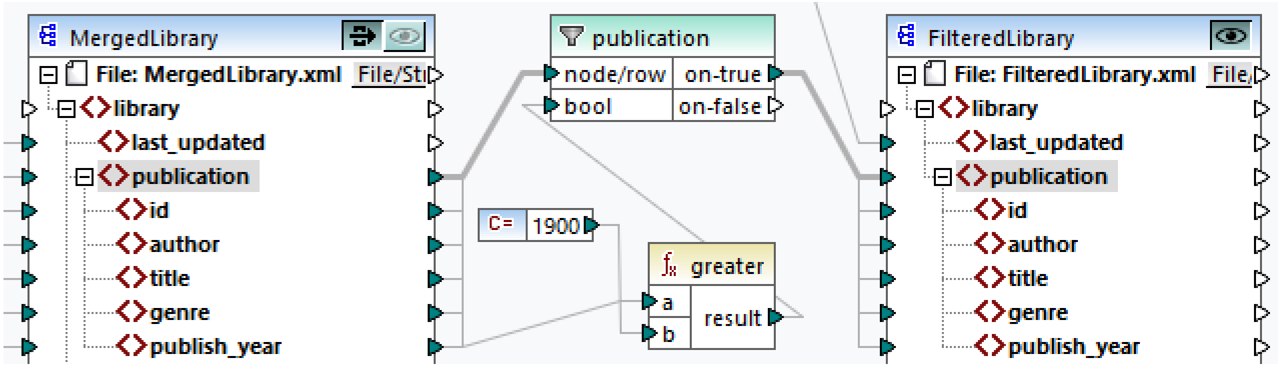
2.2.5 Keep Connections after Deleting Components

MapForce allows you to keep connections even after deleting some transformation components 34: e.g., variables, sort and filter components, value-maps, simple inputs, SQL/NoSQL-WHERE/ORDER components. Connections can be single or multiple. Keeping connections might be particularly useful with multiple child connections, because you will not have to restore every single child connection manually after deleting a transformation component. To enable this option, go to **Tools | Options | Editing** and select **Smart component deletion (keep useful connections)**. By default, this option is disabled, which means that deleting a transformation component will also delete its direct connections.

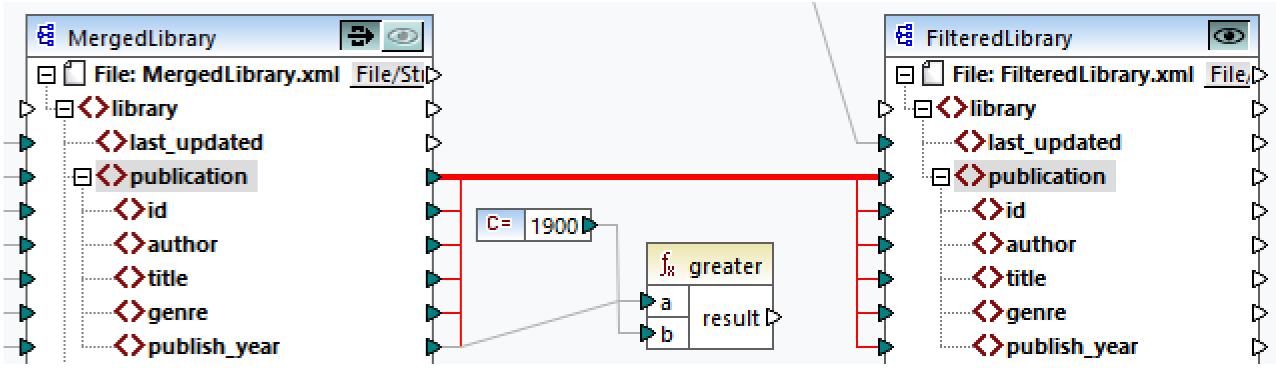
Example

The sample file called **Tut3-ChainedMapping** is used to illustrate smart component deletion. The sample file is available in the BasicTutorials 17 folder.

Before deletion
The screenshot below shows that copy-all connections 56 exist between the MergedLibrary component and the publication filter, and between the publication filter and the FilteredLibrary component. Now we want to delete the publication filter but keep the copy-all connections. In order to do that, select the check box **Smart component deletion** in the **Options** dialog box (*see above*).



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



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

2.3 General Procedures and Features

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

- Validation 64
- Code Generation 66
- Text View Features 68
- Text View Search 72
- Mapping Settings 75

2.3.1 Validation

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

Validate mappings

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

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

When you validate a mapping, MapForce checks, for example, for unsupported component types, incorrect or missing connections. To find out more about validation statuses, see Messages Window 27. The **Messages** window also allows you to take message-related actions 27. To display the result of each validation in an individual tab, click the numbered tabs available on the left side of the **Messages** window. This may be useful, for example, if you work with multiple mapping files simultaneously.

Validation of transformation components
Validation of transformation components 35 works as follows:

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

Preview and validate output

MapForce allows you to preview the output without having to run and compile the generated code with an external processor or compiler. In general, it is a good idea to preview the transformation output in MapForce before processing the generated code externally. When you preview the mapping results, MapForce executes the mapping and shows the output in the Output pane 30.

Once the data is available in the **Output** pane, you can validate and save it if necessary. You can also use the **Find** command (**Ctrl + F**) to quickly locate a particular text pattern in the output file. For more information, see Text View Search 72. Any error, warning or information messages related to the mapping execution are displayed in the Messages window 26.

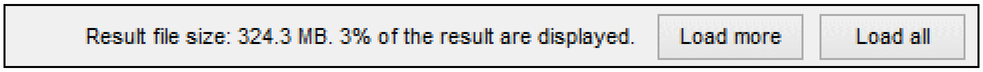
If you select C++, C#, or Java (*Professional and Enterprise editions*) as a transformation language 19, MapForce executes the mapping using its built-in transformation engine and displays the result in the **Output** pane.

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

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

Load options

When you preview large output files, MapForce limits the amount of data displayed in the **Output** pane. In this case, the **Load more** button appears in the lower area of the pane (*see screenshot below*). Clicking the **Load more** button adds the next piece of data. You can configure the preview settings from the **General** tab of the **Options** dialog box. For more information, see MapForce Options 1077.



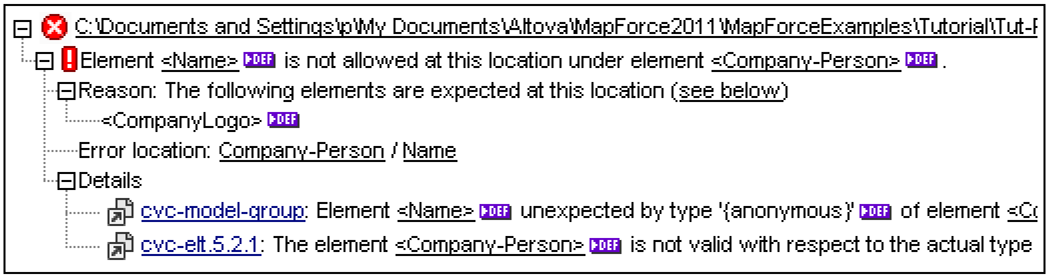
Result file size: 324.3 MB. 3% of the result are displayed.

Validate output

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

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

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



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

2.3.2 Code Generation

Code Generator is a MapForce built-in feature which enables you to generate code from mapping files. You can use the generated code to execute your mappings outside of MapForce, which will enable you to automate your mapping operations. You can generate code in the following data transformation languages 19:

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

You can generate code from a single mapping design (**`.mfd`**) or from a mapping project (**`.mfp`**). Code generation from a project is supported only in Professional and Enterprise editions. For details, see the subsections below.

Important points
Note the following code-generation aspects:

- Certain MapForce features are not supported in generated program code. For details, see Supported features in generated code 1098.
- For information about handling paths in generated code, see Paths in Execution Environments 46.
- *Professional and Enterprise editions:* You can change general code-generation options in the *Generation* section of the **Options** dialog. For details, see Generation 1080.
- *Professional and Enterprise editions:* Support for database connections varies depending on the platform, and there are connection types that are not supported on all platforms. If your mapping connects to a database, choose a database connection that is compatible with the target environment for which you generate code. For details, see Database mappings in various execution environments 166.

Support information
The table below summarizes support information for C++, C#, and Java.

Target Language	**C++**	**C#**	**Java**
Development environments	Microsoft Visual Studio 2013, 2015, 2017, 2019, 2022	Microsoft Visual Studio 2013, 2015, 2017, 2019, 2022 Target frameworks: • .NET Framework • .NET Core 3.1 • .NET 5.0 • .NET 6.0 • .NET 8.0	Java SE JDK 8, 11, 17, 21 (including OpenJDK) Eclipse 4.4 or later Apache Ant
XML DOM implementations	MSXML 6.0 Apache Xerces 3	System.Xml	JAXP
Database API	ADO	ADO.NET	JDBC

Generate code from a mapping

To generate code from a mapping design (**.mfd**), follow the instructions below:

1. Select the relevant code-generation options in the *Generation* section of the **Options** dialog (applicable to C# and C++) and in the Mapping Settings 75. For details about the code-generation settings in the **Options** dialog, see Generation 1080.
2. Click **File | Generate code in** and select the relevant transformation language. Alternatively, you can select **File | Generate Code in Selected Language**. In this case, code will be generated in the language selected in the toolbar.
3. Select a destination directory for the generated files and then click **OK** to confirm. MapForce generates the code and displays the result of the operation in the Messages window 26.

Generate code from a project (Professional and Enterprise editions)

You can generate code from a mapping project (**.mfp**) that consists of multiple mapping design files (**.mfd**). Note that all mapping design files in the project must qualify for generation, which means that all their components must be supported in the selected transformation language (see Supported features in generated code 1098).

To generate code from a mapping project, follow the instructions below.

1. Open the relevant mapping project, for which you wish to generate code.
2. Right-click the project name in the Project window and then select **Properties** from the context menu. Alternatively, click the project name and select the **Project | Properties** menu item.
3. Review and change the project settings if required. In particular, ensure that the target language and the output directory are set correctly. Then click **OK**.
4. Click **Generate Code for Entire Project** in the **Project** menu.

Irrespective of the language selected in the **Project Properties** dialog, you can always choose to generate project code in a different language, by selecting the menu command **Project | Generate Code in | <language>**.

The progress and result of the code generation process are 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.

You can change the output directory and/or the name of the project in the **Project Properties** dialog. If your MapForce project contains folders, you can configure the code generation settings for each individual folder: Right-click a folder of interest and select **Properties** from the context menu. Otherwise, all project folders inherit the settings defined at top level. For more information about projects and project-related settings and procedures, see Projects 95.

Next steps

Depending on the transformation language you have selected for code generation, the subsequent steps vary. If you have generated code in XSLT 1-3 or XQuery, the next step will be to run the transformation from the command line (*see details below*).

If you have generated Java, C#, or C++ code, the next steps will be to build and run the generated code. For more information about these procedures, see Code Generator 868. You can also modify the generated Java, C#, and C++ code and integrate it into your custom code. For details, see Integrate Generated Code 876.

XSLT and XQuery code
After you have generated XSLT 1-3 code, the destination folder will include the following files:

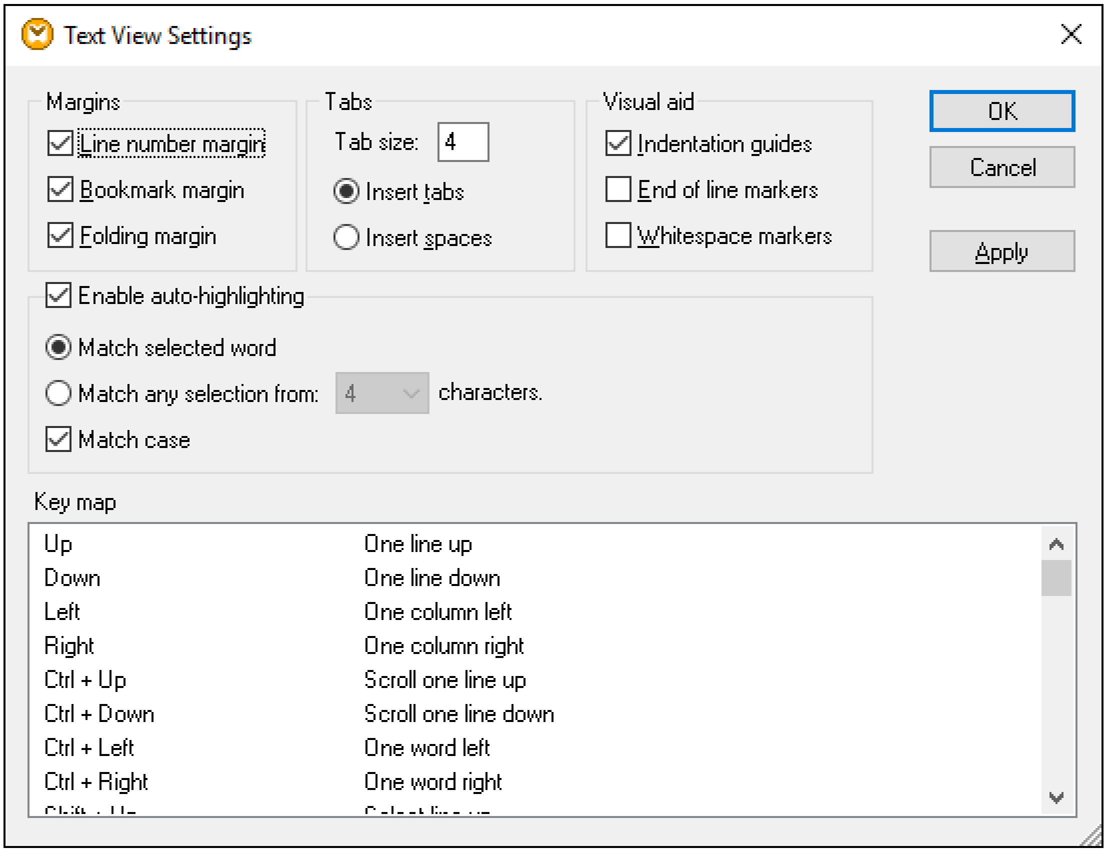
1. An XSLT transformation file that has the following format: **`<Mapping>MapTo<TargetFileName>.xslt`**. `<Mapping>` is the value of the *Application Name* field in the mapping settings 75. `<TargetFileName>` is the name of the target component. To change this value, open the settings of the target component and edit the value of the *Component Name* field. For more information, see Change Component Settings 41 and Library paths in generated code 46.
2. A **`DoTransform.bat`** file, which enables you to run the XSLT transformation with Altova RaptorXML Server from the command line. In order to run the command, you will need to install RaptorXML.

If your mapping is chained 115, a separate transformation file will be generated for each target component.

XQuery code generation is similar to XSLT code generation, except that the transformation file(s) have a **`.xq`** extension and the following format: **`<Mapping>MapTo<TargetFileName>.xq`**.

2.3.3 Text View Features

The Output pane 30, the XQuery pane 29, and the XSLT pane 28 have multiple visual aids to make the display of text easier: e.g., margins, text highlighting, indentation guides, end-of-line and whitespace markers. You can customize these features in the **Text View Settings** dialog box (*see screenshot below*). The settings in this dialog box apply to the entire application.



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

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

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

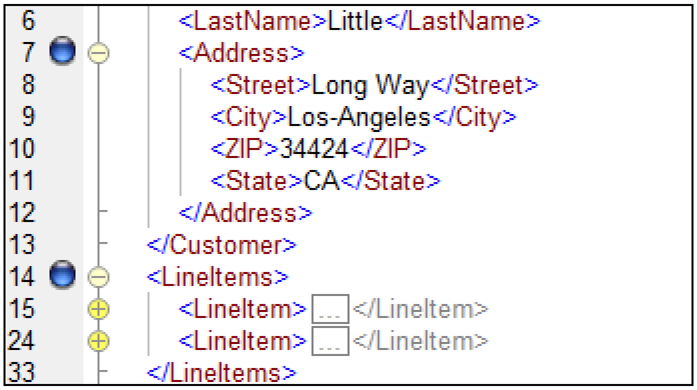
Some of the navigation aids can also be toggled from the **Text View** toolbar, the application menu, or keyboard shortcuts. For more information about shortcuts, see the **Key map** section of the **Text View Settings** dialog box shown above.

See the list of available settings below.

⊟ Margins

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

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



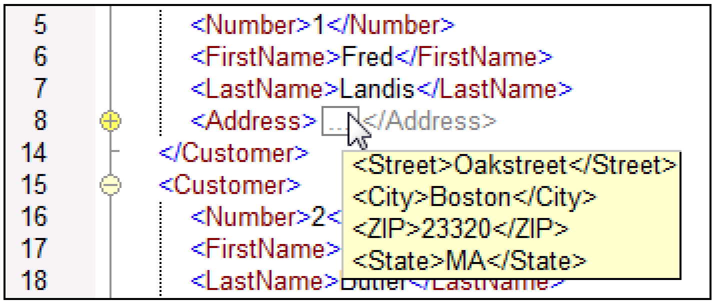
```
        <LastName>Little</LastName>
        <Address>
            <Street>Long Way</Street>
            <City>Los-Angeles</City>
            <ZIP>34424</ZIP>
            <State>CA</State>
        </Address>
    </Customer>
    <LineItems>
        <LineItem> ... </LineItem>
        <LineItem> ... </LineItem>
    </LineItems>
```

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

⚑	**Insert/Remove Bookmark (Ctrl + F2)**
→⚑	**Go to Next Bookmark (F2)**
←⚑	**Go to Previous Bookmark (Shift + F2)**
⚑✕	**Delete All Bookmarks (Ctrl + Shift + F2)**

Folding margin

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



```
        <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>
```

⊟ Enable auto-highlighting

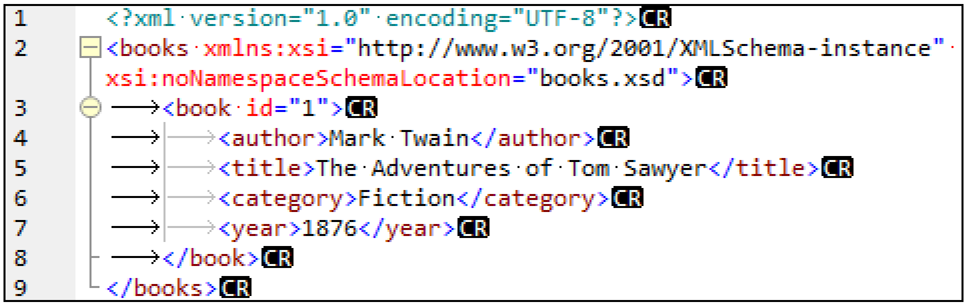
The **Enable auto-highlighting** setting allows you to see all the matches of the selected piece of text. The selection is highlighted in pale blue, and the matches are highlighted in light brown. The selection and its matches are indicated as gray marker-squares in the scroll bar. The current cursor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

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

⊟ Visual aid

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

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



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

⊟ Other text view settings

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

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

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

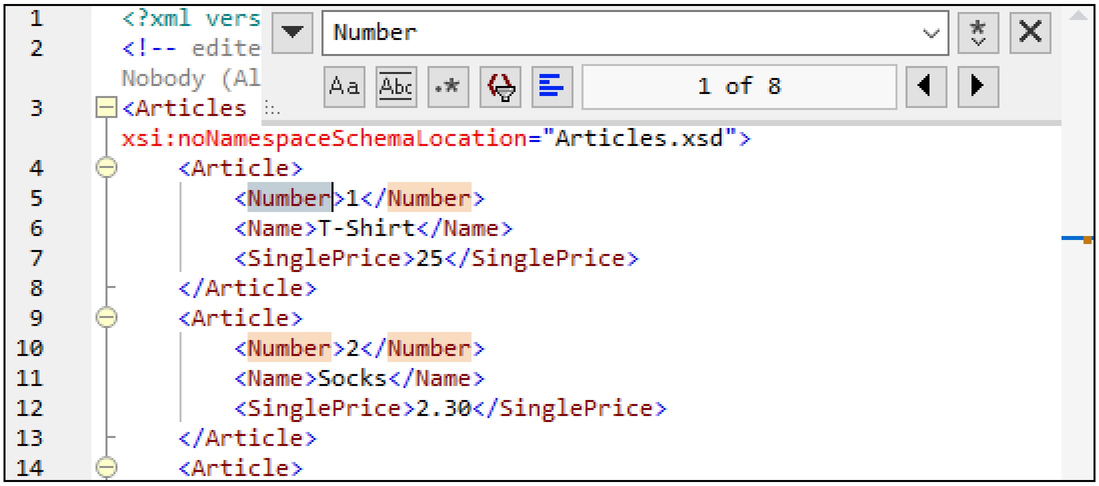
Word wrapping
Word wrapping helps display a code listing within the borders of the working area. If the word wrap setting is not enabled, some portions of text may not be fully visible in the working area. To toggle word wrapping in the currently active document, select the menu command **Output | Word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

2.3.4 Text View Search

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

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

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



```
<?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>
```

Find options

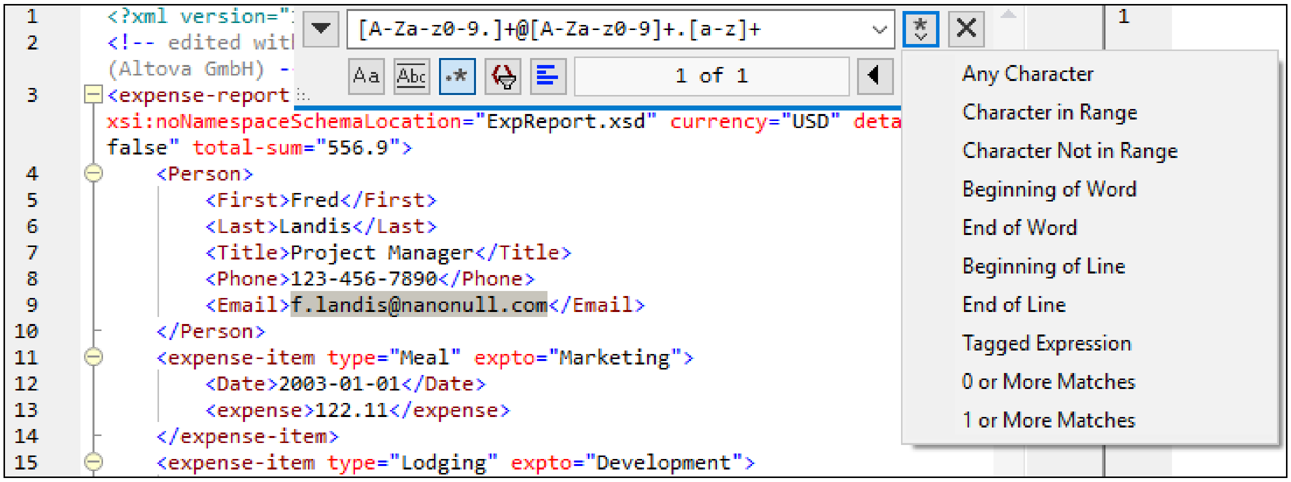
You can specify find criteria with the help of the buttons located under the search field. The list of the available options is given in the table below.

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

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

Regular expressions

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



```
[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">
```

Regular expression metacharacters

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

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

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

Find special characters

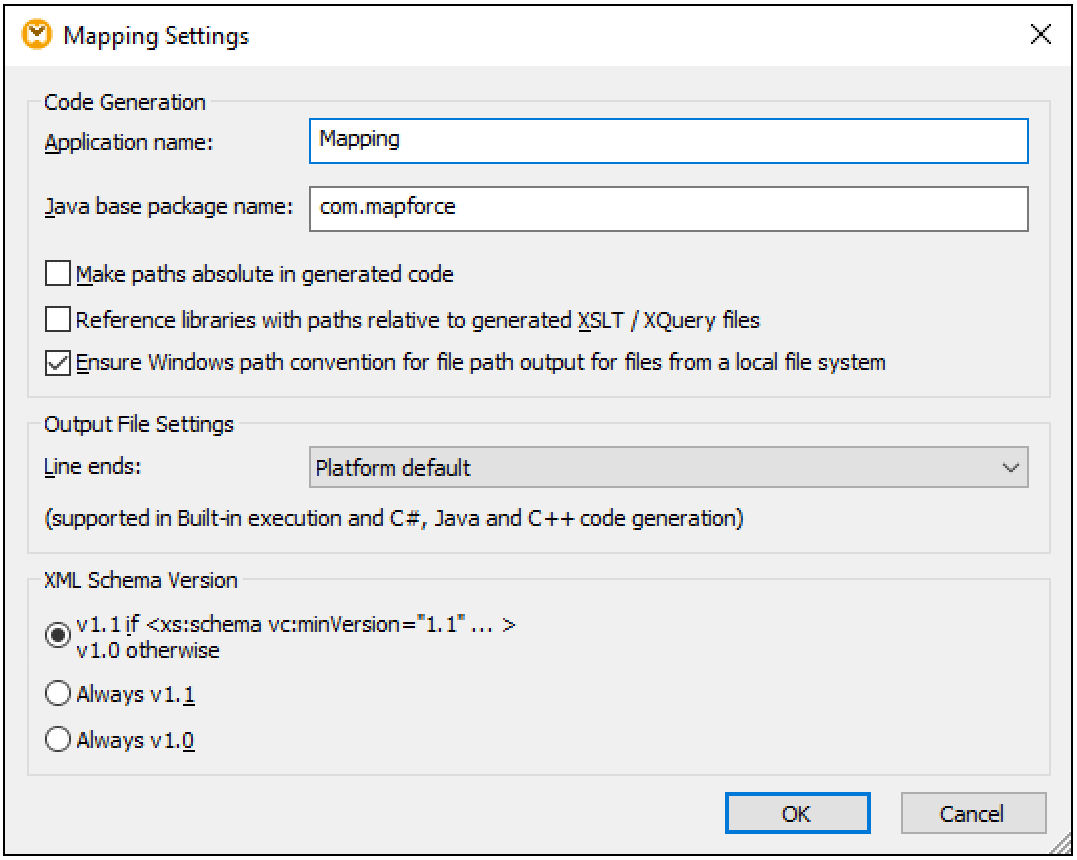
If the **Use regular expressions** option is enabled, you can search for any of the following special characters within the text:

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

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

2.3.5 Mapping Settings

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



Mapping Settings

Code Generation

Application name: Mapping

Java base package name: com.mapforce

□ Make paths absolute in generated code

□ Reference libraries with paths relative to generated XSLT / XQuery files

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

Output File Settings

Line ends: Platform default

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

XML Schema Version

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

○ Always v1.1

○ Always v1.0

The available settings are described in the subtopics below.

⊟ Code Generation

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

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

- Make paths absolute in generated code: This check box affects all paths in mapping components, except paths to external library files (e.g., XSLT libraries). The check box defines whether the file paths should be relative or absolute in the generated program code, in MapForce Server Execution files (`.mfx`) and in mapping functions deployed to FlowForce Server. For more information, see Paths in Execution Environments 46.

- *Reference libraries with paths relative to the generated XSLT/XQuery files:* This check box is applicable when the mapping language is XQuery (*Professional and Enterprise editions*) or XSLT. This option is useful if your mapping references an XSLT or XQuery library, and you plan to generate XSLT or XQuery files from the mapping. Select this check box if you want the library paths to be relative to the directory of the generated XSLT or XQuery code. If the check box is not selected, the library paths will be absolute in the generated code. See also Library paths in generated code 46.

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

⊟ Output File Settings (Professional and Enterprise editions)

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

⊟ XML Schema Version

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

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

⊟ Web Service Operation Settings (Enterprise edition)

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

2.4 Basic Rules and Strategies

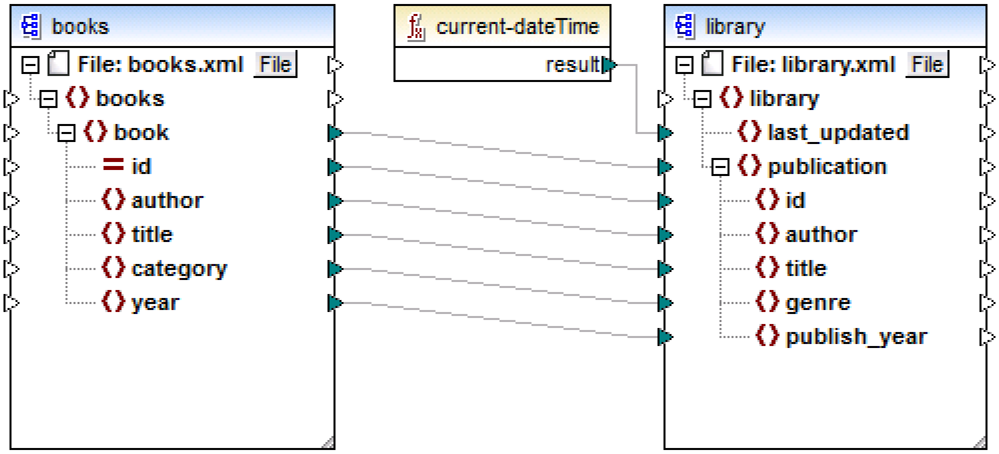
In general, MapForce maps data in an intuitive way, but you may come across situations where the output contains too many or too few items. This chapter describes the main mapping rules and helps you avoid situations in which the mapping produces undesired output due to incorrect connections or context.

In order to be valid, a mapping must include at least one source connected to at least one target component. A source component reads data, typically from a file or database. A target component writes data, typically to a file or database.

All mapping connections that you draw make together a mapping algorithm. At mapping runtime, MapForce evaluates the algorithm and processes data based on it. The integrity and efficiency of the mapping algorithm depends primarily on the connections. You can also tweak some settings at mapping 75 level, at component 41 level, or even at connection 51 level, but it is the *connections* that determine how your data is processed.

Mapping rules

This subsection describes the main rules of data processing. To illustrate the rules, we will use the mapping shown below.



Rule 1
For each source item, one item is created in the target. **This is the main mapping rule in MapForce.** In the mapping above, if the source XML contains three `book` elements, then three `publication` elements will be created on the target side. Note that there are also a few special cases (see Sequences 78).

Rule 2
MapForce always starts processing from the *target root item (node)*. This is where the execution of a mapping actually begins. After processing the target node, MapForce progresses down the hierarchy.

Rule 3
Each connection creates the *current mapping context*. The context determines what data is available *for the current target node and its descendants*. The context, therefore, determines which source items are actually copied from the source to the target component. By drawing or omitting a connection, you may inadvertently change the current context and thus affect the output of the mapping. For example, your mapping might

unnecessarily call a database or a web service multiple times in the same mapping execution. This concept is further described in the chapter Context and Processing Order 79.

Rule 4
When you draw a connection from a source item to a target item, the mapping reads data from that source item, passes the data to the target item, and generates instance data in the target format.

The source item may have zero, one, or multiple occurrences in the source instance file. Irrespective of the number of occurrences, MapForce always displays only schema hierarchies in structural components (e.g., XML). That is the reason why, for example, the source component in the mapping above has only one book element.

2.4.1 Sequences

Understanding sequences is essential for mastering mapping techniques in MapForce. The main rule for sequences runs as follows:

> *If a sequence of more than one node is connected to a target node, a loop is created that generates as many target nodes as there are source nodes.*

If there is a filter on the way, MapForce checks if there is at least one Boolean condition in the sequence that evaluates to true and maps the data that satisfies this condition to the target.

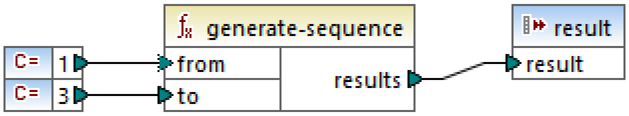
If a sequence is empty, nothing is generated on the target side. For example, if the target is an XML document and the source sequence is empty, no XML elements would be created in the target at all.

Exceptions

Note some exceptions to the rule about sequences (*see below*).

Exception 1
If the target item is an XML root element, it is created only once. If you connect a sequence to it, the result might not be schema-valid. If attributes of the root element are also connected, the XML serialization will fail at runtime. Therefore, avoid connecting a sequence to the root XML element.

Exception 2
If the target item accepts only one value, it is created only once. Examples of items that accept only one value are as follows: XML attributes, database fields, simple output components. For example, the mapping below generates a sequence of three integers (1, 2, 3) with the help of the **generate-sequence** function. Nevertheless, the output will contain only one integer, because the target is a simple-output component that accepts a single value. The other two values are ignored.



Exception 3
If the source schema specifies that a specific item occurs only once, but the instance file contains many items, MapForce may extract the first item from the source (which must exist according to the schema) and

create only one item in the target. To disable this behavior, clear the check box *Enable input processing optimizations based on min/maxOccurs* from the component settings 133.

Functions

If a non-sequence function gets a sequence as input, the function is called as many times and produces as many results as there are items in the sequence. If a function gets an empty sequence as input, it returns an empty result and, consequently, produces no output at all.

Functions that return a value even if they get an empty sequence
However, there are some categories of functions that, by virtue of their design, return a value even if they get an empty sequence as input:

- **`exists`**, **`not-exists`**, **`substitute-missing`**
- **`is-null`**, **`is-not-null`**, **`substitute-null`** (these three functions are aliases of the previous three)
- aggregate functions (**`sum`**, **`count`**, and so on)
- user-defined functions that accept sequences and are regular (not inlined) functions

If you need to replace a missing value, add the **`substitute-missing`** function to the mapping and replace the missing value with a substitute value of choice. Alternatively, you can achieve the same result by using defaults and node functions 442.

Functions with multiple inputs
Functions may have multiple inputs. If a sequence is connected to each input, this produces a Cartesian product of all inputs (a set of all ordered pairs that can be created by combining elements from two or more sets), which is typically not the desired outcome. To avoid this, connect only one sequence to a function with multiple parameters; all other parameters must be connected to "singular" items from parents or other components.

2.4.2 Context and Processing Order

Context is one of the most crucial aspects in mapping execution. Understanding context is key to achieving the desired result in the output. Context determines what data is available *for the current target node and its descendants* and which source items are actually selected and copied from the source to the target component.

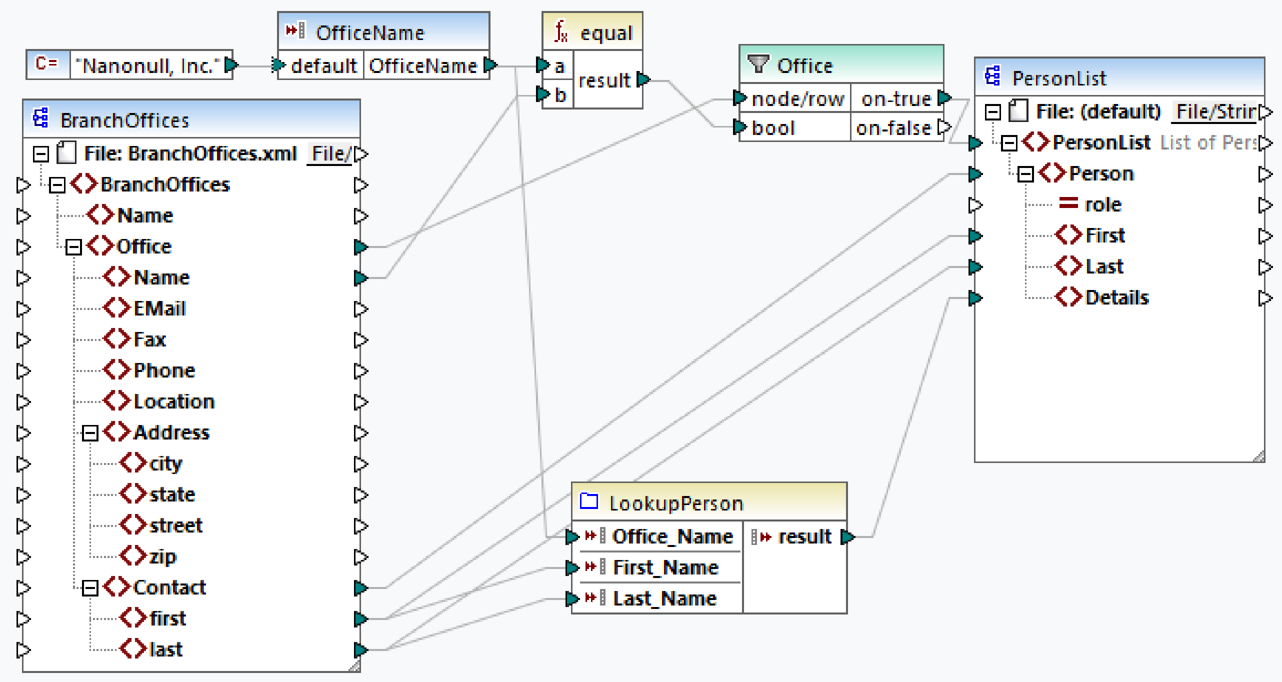
MapForce always establishes the current context starting from the *target root item (node)*. This is where mapping execution actually begins. After processing the target node, MapForce progresses down the hierarchy. For each target node, the connections are traced back to the sources, through any functions in between.

For each target node, a new context is established that initially contains all the current source nodes of the parent and all function results. While MapForce evaluates the connections leading directly or indirectly to the target node, additional sources and function results are added to the context. Target sibling nodes are thus independent of each other but have access to all the sources of their parents.

For a recap of the fundamental rules of mapping execution, see Basic Rules and Strategies 77.

Example

To put the principles described above into practice, we will discuss the following mapping: **`MapForceExamples\PersonListByBranchOffice.mfd`** (*illustrated below*). The main objective of the mapping is to get a list of employees with their details from a particular office.



Source and target structures

In our example, the mapping contains a source XML component and a target XML component. Besides, there is a simple-input component that also acts as a source and supplies the name of the office in whose employees we are interested. The mapping also features various transformation components such as a filter and some functions.

It is important to note that MapForce always displays only schema hierarchies in structural components (e.g., XML). For example, the `BranchOffices` component shows only one `Contact` element despite the fact that the source instance file **`BranchOffices.xml`** contains multiple `Contact` nodes with different content and in different `Office` parent nodes. An extract from **`BranchOffices.xml`** is shown below.

```
<BranchOffices xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="BranchOffices.xsd">
      <Name>Nanonull</Name>
      <Office>
            <Name>Nanonull, Inc.</Name>
            <EMail>office@nanonull.com</EMail>
            <Fax>+1 (321) 555 5155 - 9</Fax>
            <Phone>+1 (321) 555 5155</Phone>
            <Address>
                  <city>Vereno</city>
                  <state>CA</state>
                  <street>119 Oakstreet, Suite 4876</street>
                  <zip>29213</zip>
            </Address>
            <Contact>
                  <first>Vernon</first>
                  <last>Callaby</last>
            </Contact>
            <Contact>...</Contact>
            <Contact>...</Contact>
            <Contact>...</Contact>
            <...>
      </Office>
      <Office>
            <Name>Nanonull Partners, Inc.</Name>
            <EMail>nextoffice@nanonull.com</EMail>
            <Fax>+1 (927) 555 1845</Fax>
            <Phone>+1 (927) 555 0094</Phone>
            <Address>
                  <city>Brenton</city>
                  <state>MA</state>
                  <street>9865 Millenium Center, Suite 456</street>
                  <zip>5985</zip>
            </Address>
            <Contact>
                  <first>Steve</first>
                  <last>Meier</last>
            </Contact>
            <Contact>...</Contact>
            <Contact>...</Contact>
            <...>
      </Office>
</BranchOffices>
```

Context and processing order
In our mapping, processing begins from the target root node called `PersonList`. We can trace back the connection via the filter and the `equal` function to its source item - the `Office` element.

Single office as context for the target
Since our mapping has a filter, only the data that satisfies the Boolean condition will be mapped to the target. This condition is satisfied only once, because there is only one office called Nanonull, Inc. in the source XML file. Consequently, the connection from the `PersonList` list through the filter to the `Office` node defines a single office as the context for the entire target document. This means that all further connections to the descendants of the root element will be affected by the filter and will have access only to the data of the Nanonull, Inc. office, and no other office will exist in the current context.

One Person for each Contact
The next connection is between the `Contact` and `Person` nodes. According to the general mapping rule[77], this connection will create one target `Person` for each source `Contact`. On each iteration, this connection establishes one current `Contact` for the connections to the children of the `Person` element. Therefore, the child connections (`first` to `First`, `last` to `Last`) will select the first and last names of the *current* `Contact` element and write them to the target.

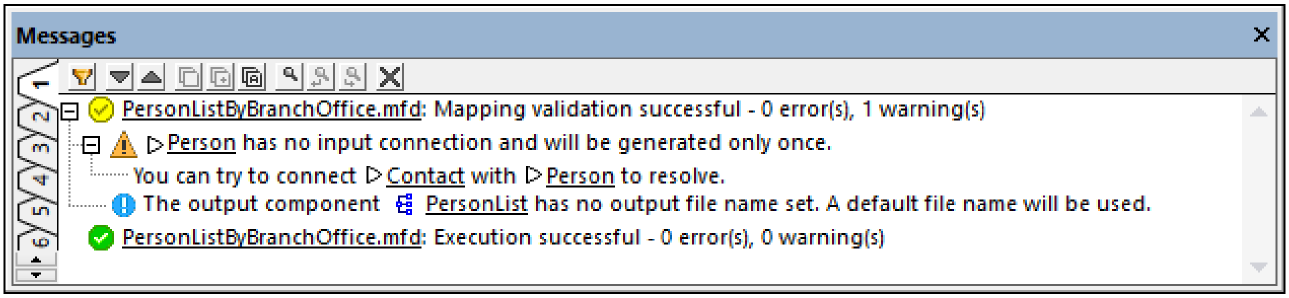
User-defined function
The mapping also includes a user-defined function[457] called `LookupPerson`. The user-defined function is also executed in the context of each `Person` because of the parent connection between `Contact` and `Person`. Each

time a new Person item is created on the target side, the function is called to populate the Details element of the person.

The function takes three input parameters. The OfficeName parameter reads data from the simple-input component. The source data for this parameter could as well be provided by the Name source item, without changing in any way the mapping output. In either case, the source value is the same and taken from the parent context. Internally, the look-up function concatenates the values received as arguments and produces a single value.

For more information about how the **LookupPerson** function is configured, see Look-up Implementation 472.

If parents are not connected
If there were no connection between Contact and Person, the mapping would create only one Person with multiple First, Last, and Details nodes. In such cases, MapForce displays a warning in the Messages window (*screenshot below*) and suggests connecting the parent nodes to resolve the issue.



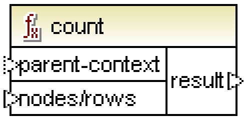
PersonListByBranchOffice.mfd: Mapping validation successful - 0 error(s), 1 warning(s)
▷Person has no input connection and will be generated only once.
You can try to connect ▷Contact with ▷Person to resolve.
The output component PersonList has no output file name set. A default file name will be used.
PersonListByBranchOffice.mfd: Execution successful - 0 error(s), 0 warning(s)

2.4.3 Parent Context

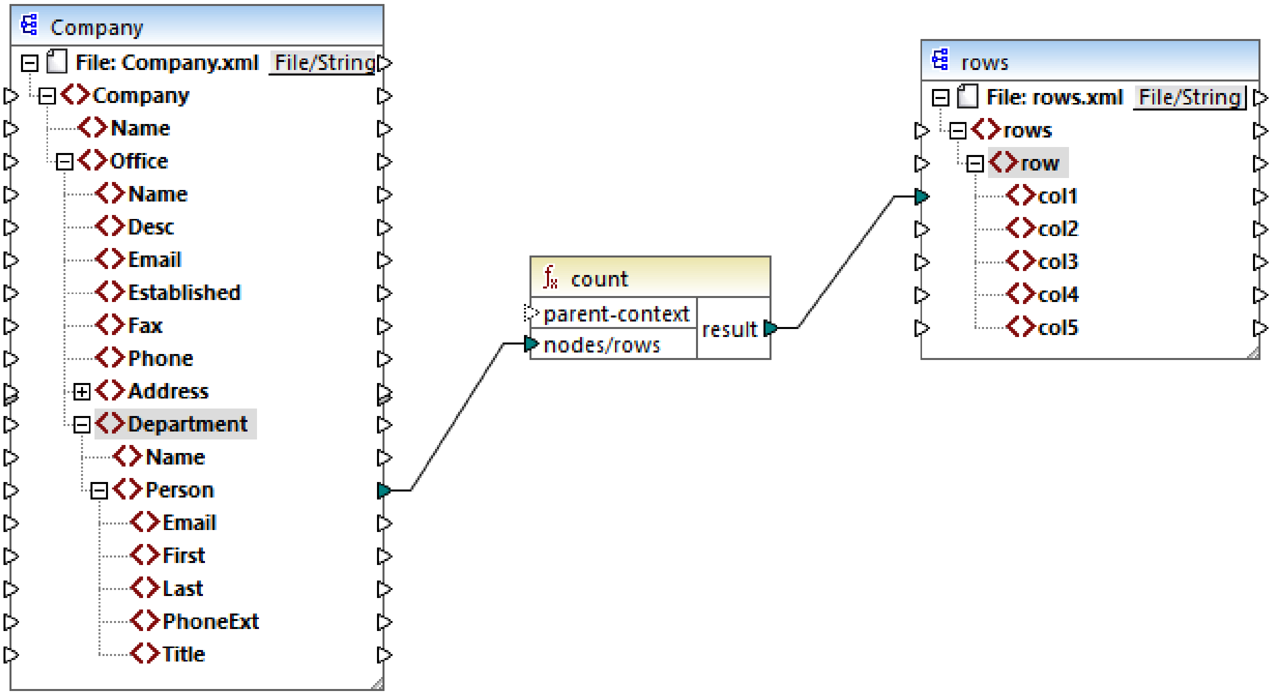
Some mapping components have an optional **parent-context** item.

The parent-context argument is an optional argument in some MapForce core aggregation functions (e.g., **min**, **max**, **avg**, **count**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.

With the help of this item you can influence the mapping context in which that component should operate and consequently change the mapping output. The components that have an optional **parent-context** are: aggregate functions, variables, and Join components.



For a demo of how changing the parent context is useful, open the following mapping: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\ParentContext.mfd**.



In the source XML of the mapping above, there is a single **Company** node which contains two **Office** nodes. Each **Office** node contains multiple **Department** nodes, and each **Department** contains multiple **Person** nodes. If you open the XML file in an XML editor, you can see that the distribution of people by office and department is as follows:

Office	Department	Number of people
Nanonull, Inc.	Administration	3
	Marketing	2
	Engineering	6
	IT & Technical Support	4
Nanonull Partners, Inc.	Administration	2
	Marketing	1
	IT & Technical Support	3

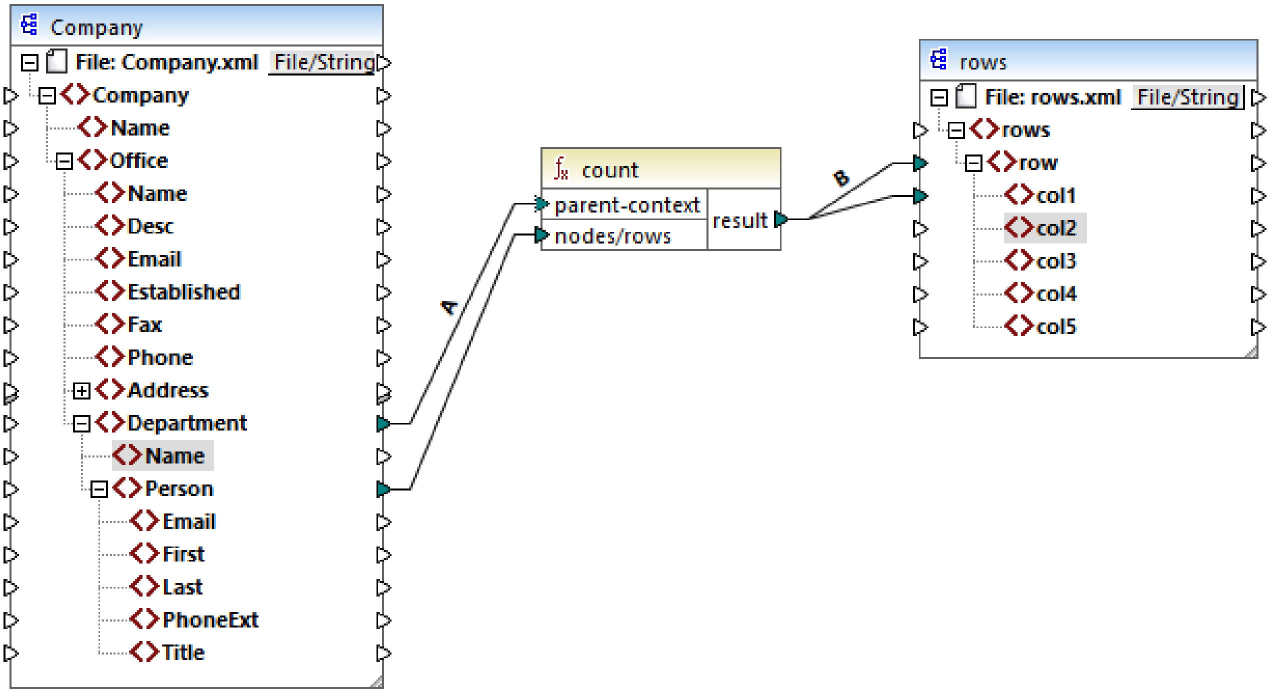
The mapping counts all people in all departments. For this purpose, it uses the **count** function from the `core` library. If you click the **Output** pane to preview the mapping, you will notice that it produces a single value, **21**, which corresponds to the total number of people in the source XML file.

The mapping works as follows:

- As usual, the mapping execution starts from the top node of the target component (**rows**, in this example). There is no incoming connection to **rows**. As a result, an implicit mapping context is established between **Company** (top item of the source component) and **rows** (top item of the target component).

- The function's result is a single value, because there is only one company in the source file.
- To populate the **col1** target item, MapForce executes the **count** function in the *implicit parent context* mentioned above, so it will count all **Person** nodes from all offices and from all departments.

The **parent-context** argument of the function lets you change the mapping context. This enables you, for example, to count the number of people in each department. To do this, draw two more connections as shown below:

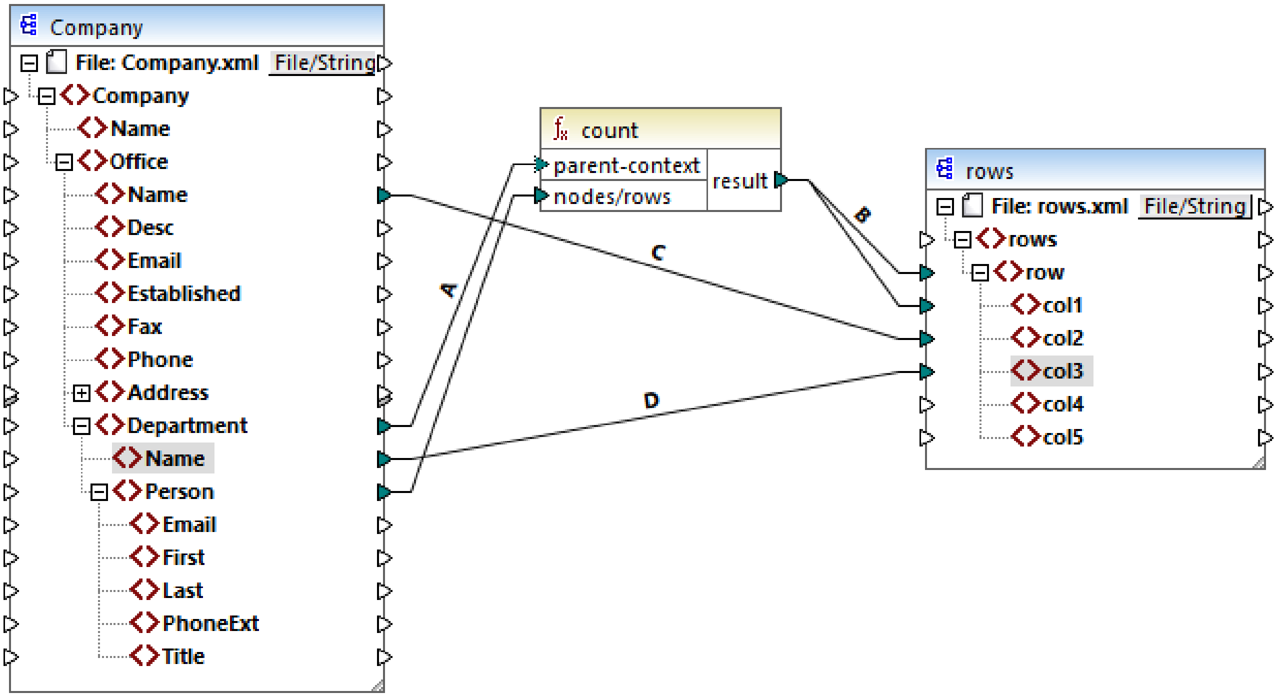


In the mapping above, connection A changes the parent context of the **count** function to **Department**. As a result, the function will count the number of people in each department. Very importantly, the function will now return a *sequence* of results instead of a single result, because multiple departments exist in the source. This is the reason why connection B exists: for each item in the resulting sequence it creates a new row in the target file. The mapping output has now changed accordingly (notice the numbers correspond exactly to the count of people in each department):

```
<rows>
    <row>
        <col1>3</col1>
    </row>
    <row>
        <col1>2</col1>
    </row>
    <row>
        <col1>6</col1>
    </row>
    <row>
```

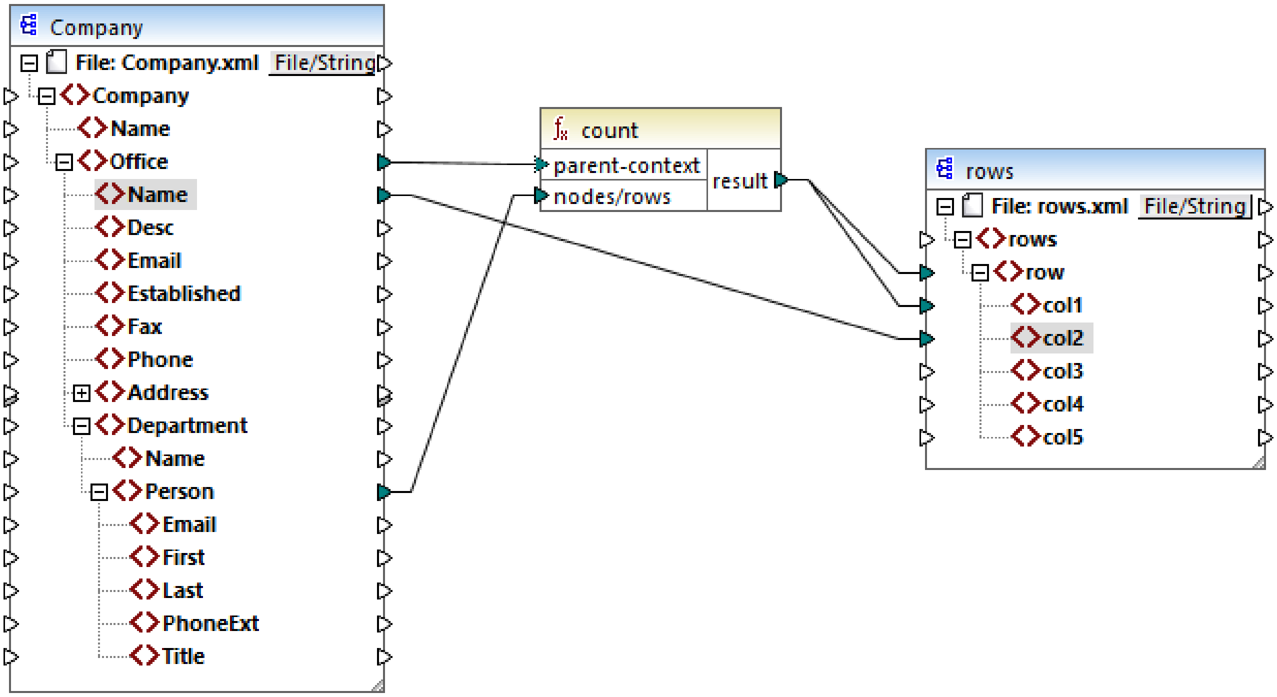
```
        <col1>4</col1>
    </row>
    <row>
        <col1>2</col1>
    </row>
    <row>
        <col1>1</col1>
    </row>
    <row>
        <col1>3</col1>
    </row>
</rows>
```

Given that the current mapping creates a row for each department, you can optionally copy the office name and the department name as well into the target file, by drawing connections C and D:



This way, the output will display not only the count of people but also the corresponding office and department name.

If you would like to count the number of people in each office, connect the parent context of **count** function to the **Office** item in the source.



With the connections shown above, the **count** function returns one result for each office. There are two offices in the source file, so the function will now return two sequences. Consequently, there will be two rows in the output, where each row is the number of people in that office:

```
<rows>
    <row>
        <col1>15</col1>
        <col2>Nanonull, Inc.</col2>
    </row>
    <row>
        <col1>6</col1>
        <col2>Nanonull Partners, Inc.</col2>
    </row>
</rows>
```

2.4.4 Priority Context

Priority context is a way to influence the order in which input parameters of a function are evaluated. Setting a priority context may be necessary if your mapping joins data from two unrelated sources.

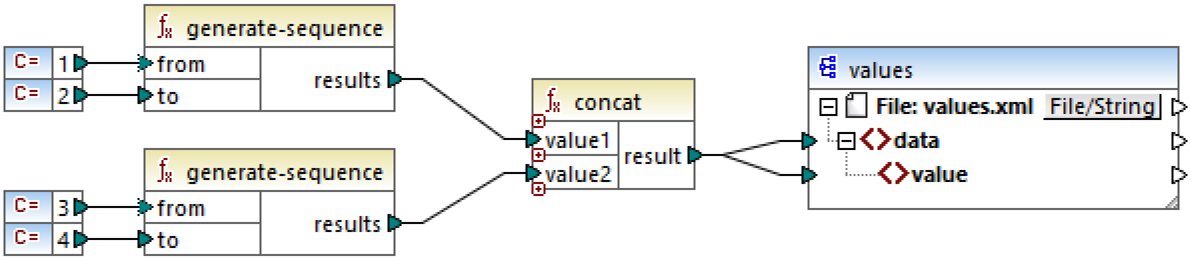
To understand how priority context works, recall that, when a mapping runs, the connection to an input item may carry a *sequence* of multiple values. For functions with two input parameters, this means that MapForce must create two loops, one of which must be processed first. The loop that is processed first is the "outer" loop. For example, the **equal** function receives two parameters: *a* and *b*. If both *a* and *b* get a sequence of values, then MapForce processes as follows:

- For each occurrence of *a*
    - For each occurrence of *b*
        - Is *a* equal to *b*?

As you can see from above, each *b* is evaluated in the context of each *a*. Priority context lets you alter the processing logic so that each *a* is evaluated in the context of each *b*. In other words, it lets you swap the inner loop with the outer loop, for example:

- For each occurrence of *b*
    - For each occurrence of *a*
        - Is *a* equal to *b*?

Let's now examine a mapping where priority context affects the mapping output. In the mapping below, the **concat** function has two input parameters. Each input parameter is a sequence that was generated with the help of the **generate-sequence** function. The first sequence is "1,2" and the second sequence is "3,4".



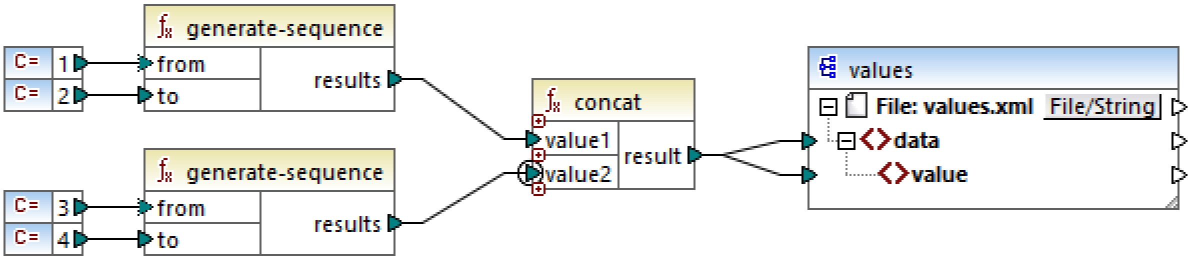
First, let's run the mapping without setting a priority context. The **concat** function starts evaluating the top sequence first, so it combines values in the following order:

- 1 with 3
- 1 with 4
- 2 with 3
- 2 with 4

This is reflected in the mapping output as well:

```
<data>
    <value>13</value>
    <value>14</value>
    <value>23</value>
    <value>24</value>
</data>
```

If you right-click the second input parameter and select **Priority Context** from the context menu, it will become the priority context. As illustrated below, the priority context input is encircled.



This time, the second input parameter will be evaluated first. The `concat` function will still concatenate the same values, but this time it will process the sequence `3,4` first. Consequently, the output becomes:

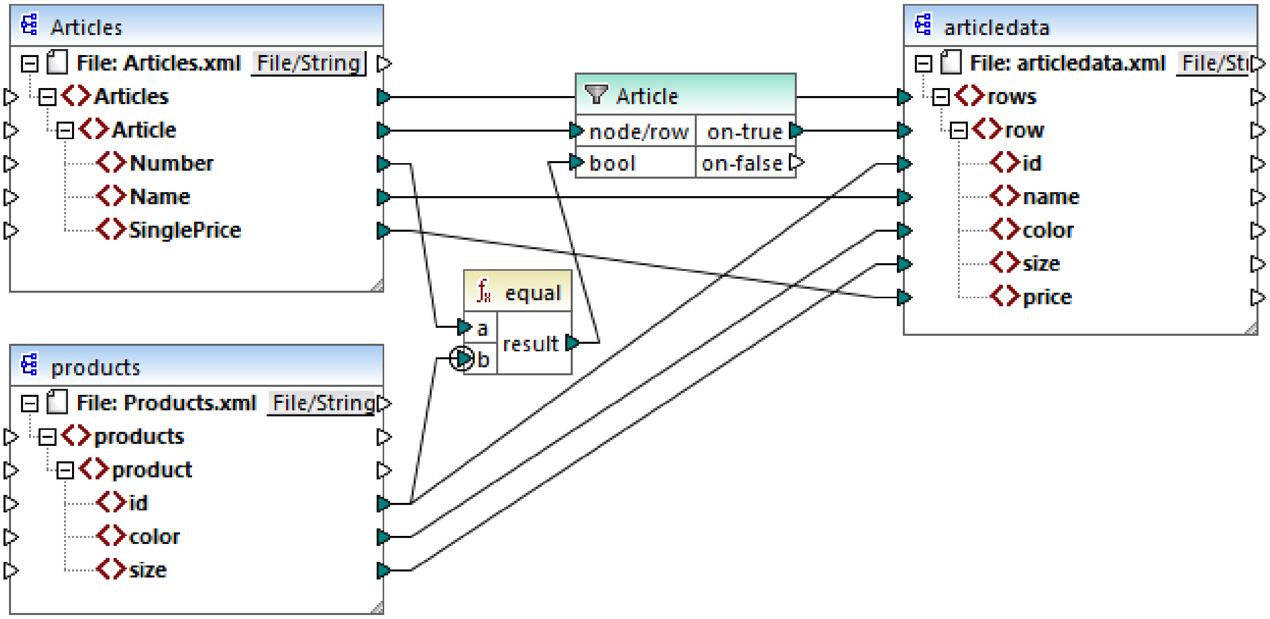
```
<data>
    <value>13</value>
    <value>23</value>
    <value>14</value>
    <value>24</value>
</data>
```

So far, you have seen only the theoretical part behind priority context. For a more practical scenario, see Example: Filter with priority context 88.

2.4.4.1 Example: Filter with priority context

When a function is connected to a filter, priority context affects not only the function itself, but also the evaluation of the filter. The mapping below illustrates a typical case when it's required to set a priority context in order to get the correct output. You can find this mapping at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\FilterWithPriority.mfd**.

Note: This mapping uses XML components, but the same logic as described below applies for all other component types in MapForce, including EDI, JSON, and so on. For databases, it's advisable to perform filtering using SQL WHERE 413 components rather than standard filters.



The aim of the mapping above is to copy data from **Articles.xml** into a new XML file with a different schema, **articledata.xml**. At the same time, the mapping should look up the details of each article in the **Products.xml** file and join them to the respective article record. Note that each record in **Articles.xml** has a **Number** and each record in **Products.xml** has an **id**. If these two values are equal, then all the other values (**Name**, **SinglePrice**, **color**, **size**) should be copied to the same **row** in the target.

This goal has been accomplished by adding a filter. Each filter requires a Boolean condition as input; only those nodes/rows that satisfy the condition will be copied over to the target. For this purpose, there is an **equal** function on the mapping. The **equal** function checks if the article number and product ID are equal in both sources. The result is then supplied as input to the filter. If **true**, then the **Article** item is copied to the target.

Notice that a priority context has been defined on the second input parameter of the second **equal** function. In this mapping, the priority context makes a big difference, and not setting it will result in incorrect mapping output.

Initial mapping: No priority context

Here is the mapping logic without priority context:

- According to the general mapping rule, for each **Article** that satisfies the filter condition, a new **row** is created in the target. The connection between **Article** and **row** (via the function and filter) takes care of this part.
- The filter checks the condition for each article. To do this, it iterates through all products, and brings multiple products in the current context.
- To populate the **id** on the target side, MapForce follows the general rule (for each item in the source, create an item in the target). However, as explained above, all products from **Products.xml** are in the current context. There is no connection between **product** to anywhere else in the target so as to read the **id** of a specific product only. As a consequence, multiple **id** elements will be created for each **Article** in the target. The same happens with **color** and **size**.

To summarize: items from **Products.xml** have the filter's context (which must iterate through each product); therefore, the **id**, **color**, and **size** values will be copied to each target **row** as many times as there are products in the source file, and generate incorrect output like the one below:

```
<rows>
    <row>
        <id>1</id>
        <id>2</id>
        <id>3</id>
        <name>T-Shirt</name>
        <color>red</color>
        <color>blue</color>
        <color>green</color>
        <size>10</size>
        <size>20</size>
        <size>30</size>
        <price>25</price>
    </row>
</rows>
```

Solution A: Use priority context

The problem above was solved by adding a priority context to the function that computes the filter's Boolean condition.

Specifically, if the second input parameter of the `equal` function is designated as priority context, the sequence incoming from **Products.xml** is prioritized. This translates to the following mapping logic:

- For each product, populate input **b** of the `equal` function (in other words, prioritize **b**). At this stage, the details of the current product are in context.
- For each article, populate input **a** of the `equal` function and check if the filter condition is true. If yes, then put the article details as well into the current context.
- Next, copy the article and product details from the current context to the respective items in the target.

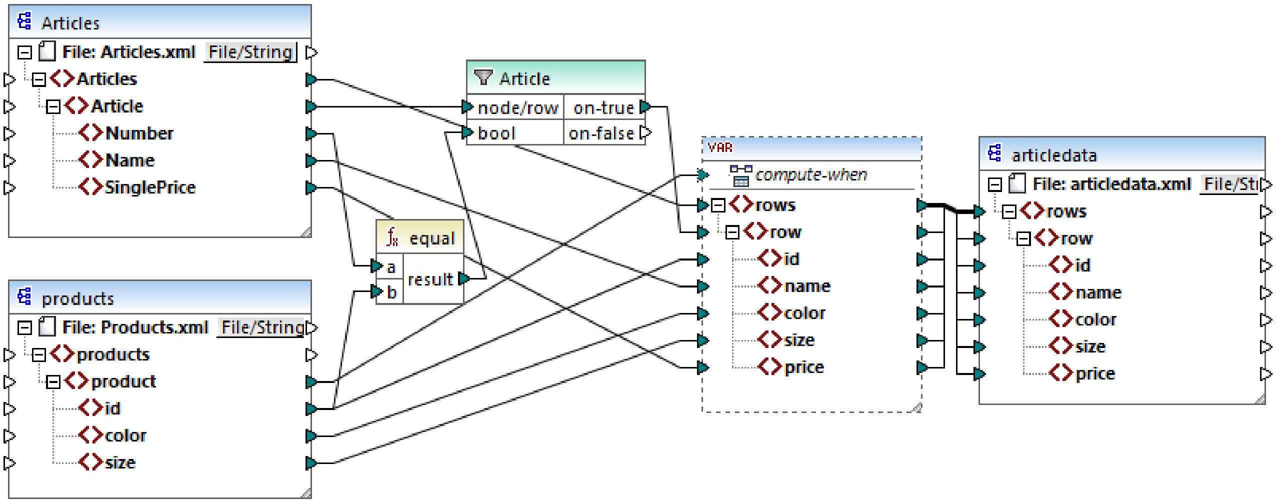
The mapping logic above produces correct output, for example:

```
<rows>
    <row>
        <id>1</id>
        <name>T-Shirt</name>
        <color>red</color>
        <size>10</size>
        <price>25</price>
    </row>
</rows>
```

Solution B: Use a variable

As an alternative solution, you could bring each article and product that matches the filter's condition into the same context with the help of an intermediate variable. Variables are suitable for scenarios like this one because they let you store data temporarily on the mapping, and thus help you change the context as necessary.

For scenarios like this one, you can add to the mapping a variable that has the same schema as the target component. On the **Insert** menu, click **Variable**, and supply the **articledata.xsd** schema as structure when prompted.



In the mapping above, the following happens:

- Priority context is not used any longer. There is a variable instead, which has the same structure as the target component.
- As usual, the mapping execution starts from the target root node. Before populating the target, the mapping collects data into the variable.
- The variable is computed in the context of each product. This happens because there is a connection from **product** to the **compute-when** input of the variable.
- The filter condition is thus checked in the context of each product. Only if the condition is true will the variable's structure be populated and passed on to the target.

2.4.5 Multiple Target Components

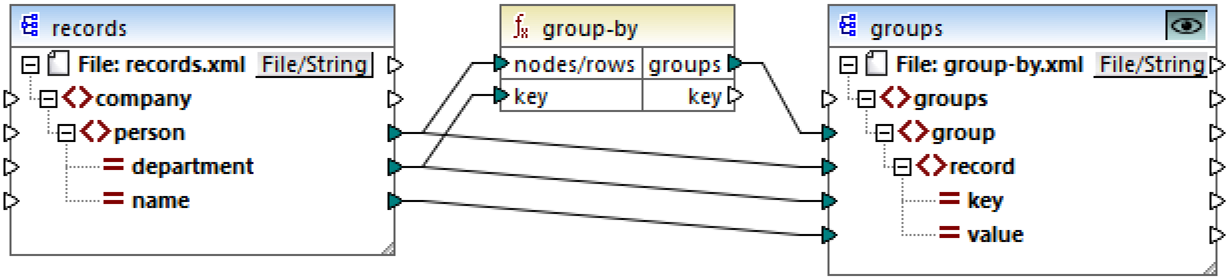
A mapping may have multiple source and target components. When there are multiple target components, you can preview only one component output at a time in MapForce, the one that you indicate by clicking the **Preview** button. In other execution environments (MapForce Server or generated code), all of the target components will be executed sequentially, and the respective output of each component will be produced.

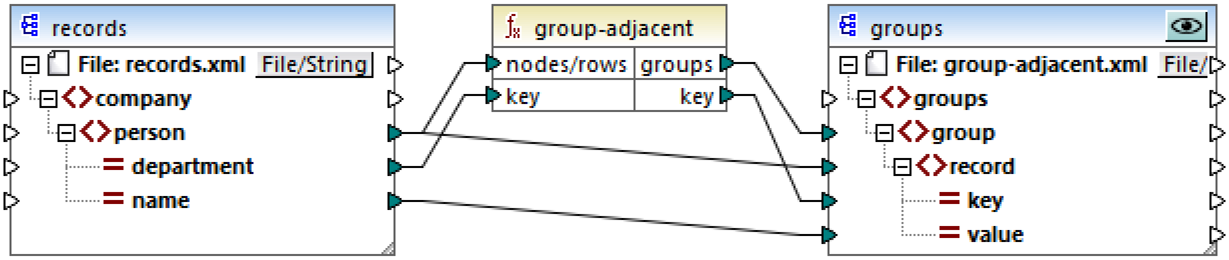
By default, target components are processed from top to bottom and from left to right. If necessary, you can influence this order by changing the position of target components in the mapping window. The point of reference is each component's top left corner. Note the following:

- If two components have the same vertical position, then the topmost component takes precedence.
- If two components have the same horizontal position, then the leftmost component takes precedence.

- In the unlikely event that components have the exact same position, then an unique internal component ID is automatically used, which guarantees a well-defined order but which cannot be changed.

For an example of how this works, open the following demo mapping: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\GroupingFunctions.mfd**. This mapping consists of multiple source and multiple target components; only a fragment is shown below.





According to the rules, the default processing order of this mapping in MapForce Server and in generated code is from top to bottom. You can check that this is the case by generating XSLT 2.0 code, for example.

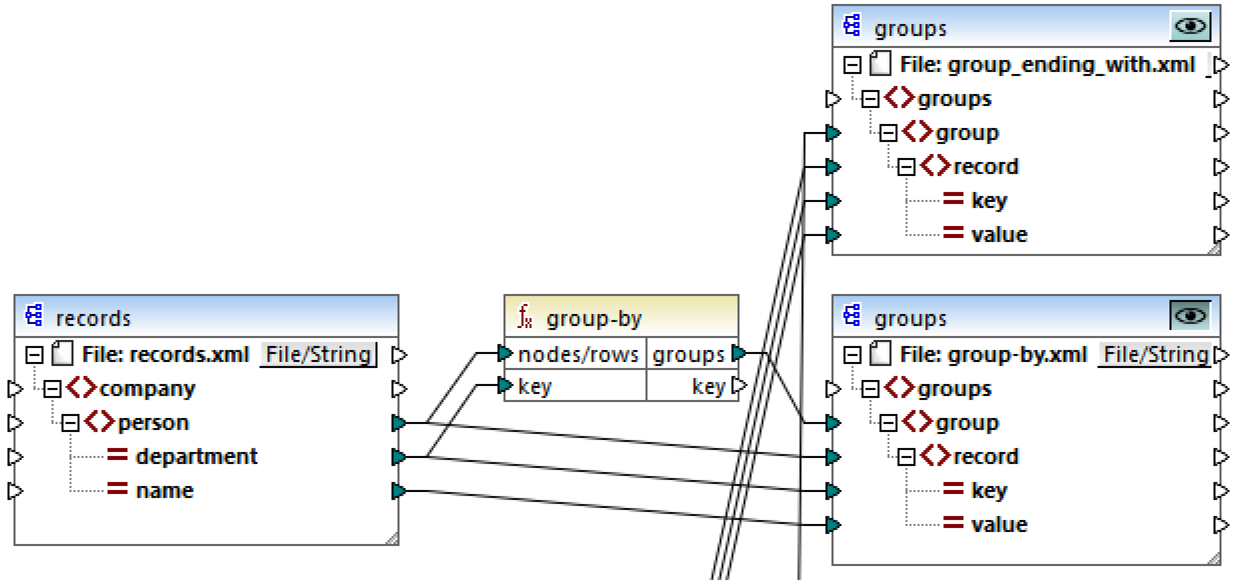
1. On the **File** menu, click **Generate code in | XSLT 2.0**.
2. When prompted, select a target directory for the generated code.

After generation, the target directory includes several XSLT files and a **DoTransform.bat** file. The latter can be executed by RaptorXML Server (requires a separate license). The **DoTransform.bat** file processes components in the same order as they were defined on the mapping, from top to bottom. This can be verified by looking at the `--output` parameter of each transformation.

```
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-by.xml" --xml-
validation-error-as-warning=true %* "MappingMapTogroups.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-adjacent.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-into-blocks.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups3.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v2.xml" --output="group-starting-
with.xml" --xml-validation-error-as-warning=true %* "MappingMapTogroups4.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v3.xml" --output="group_ending_with.xml"
```

```
--xml-validation-error-as-warning=true %* "MappingMapTogroups5.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

The last transformation produces an output file called **group-ending-with.xml**. Let's now move this target component on the mapping to the very top:



If you now generate the XSLT 2.0 code again, the processing order changes accordingly:

```
RaptorXML xslt --xslt-version=2 --input="records-v3.xml" --output="group_ending_with.xml"
--xml-validation-error-as-warning=true %* "MappingMapTogroups.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-by.xml" --xml-
validation-error-as-warning=true %* "MappingMapTogroups2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-adjacent.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups3.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-into-blocks.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups4.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v2.xml" --output="group-starting-
with.xml" --xml-validation-error-as-warning=true %* "MappingMapTogroups5.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

In the code listing above, the first call now produces **group-ending-with.xml**.

You can change the processing order in a similar way in other code languages and in compiled MapForceServer execution files (.mfx).

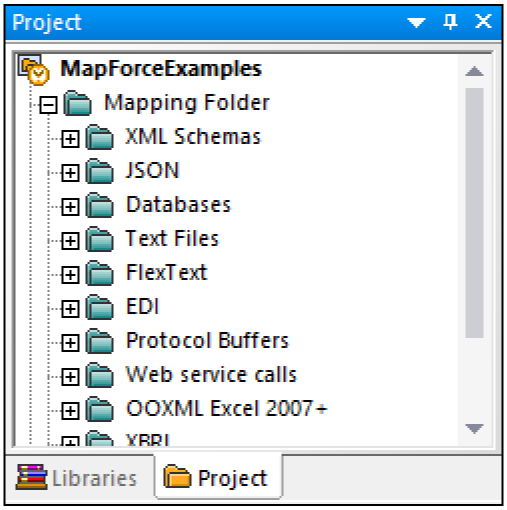
Chained mappings

The same processing sequence as described above is followed for chained mappings. The chained mapping group is taken as one unit, however. Repositioning the intermediate or final target component of a single chained mapping has no effect on the processing sequence. Only if multiple "chains" or multiple target components exist in a mapping does the position of the final target components of each group determine which is processed first.

- If two final target components have the same vertical position, then the leftmost takes precedence.
- If two final target components have the same horizontal position, then the highest takes precedence.
- In the unlikely event that components have the exact same position, then an unique internal component ID is automatically used, which guarantees a well-defined order but which cannot be changed.

2.5 Projects

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



The main advantage of projects is that you can define common code generation settings[96] (e.g., the target language and the output directory) for all the mappings included in that particular project. You can also create folders inside projects and specify custom code generation settings for each individual folder in a project. MapForce Project files are saved with the `.mfp` extension.

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

2.5.1 Project Basics

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

New project

To create a new project, take the steps below:

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

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

Project organization

Add a mapping to a project

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

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

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

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

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

Delete a file from a project

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

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

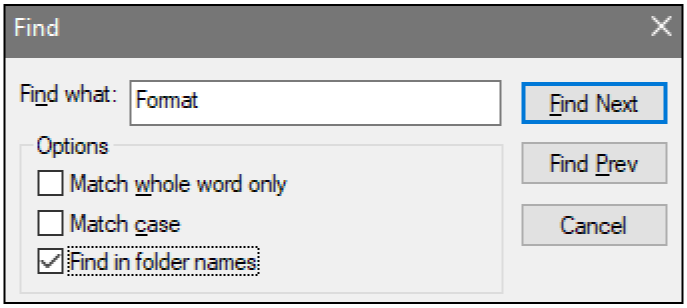
MapForce project files have a `.mfp` extension. You can open existing MapForce projects in the same way as mappings: Go to the **File** menu and click **Open**. By default, when you run MapForce for the first time, you will see the `MapForceExamples.mfp` project in the **Project** window.

Project-related actions

Search a project

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

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



Find

Find what: Format

Options

☐ Match whole word only

☐ Match case

☑ Find in folder names

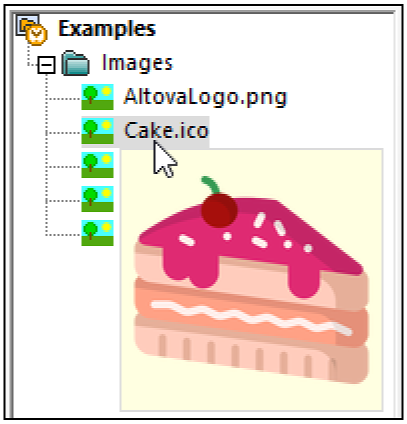
Generate code for your project

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

To generate code for a mapping or folder in your project, right-click the relevant mapping or folder and select **Generate Code** or **Generate code in**. If you select **Generate Code**, the code will be generated in the language specified in the project settings 97. You can also choose to generate code in one of the languages available in your MapForce edition. For more information, see Code Generation 66.

To generate code for the entire project, go to the **Project** menu and select **Generate Code for Entire Project**. Alternatively, right-click the name of the project in the **Project** window and select **Generate Code**. The code will be generated in the language specified in the project settings 97. For the entire project, you can also select a language to generate code in in the **Project** menu or in the project's context menu. The choice of languages depends on your MapForce edition. For more information, see Code Generation 66.

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



Watch video tutorials and add Web links
Besides multiple sample files, the **`MapForceExamples`** project also contains links to various video tutorials on the Altova website. Double-clicking any link will open the corresponding page in your default Web browser.

You can also insert your own external links in any of the following ways:

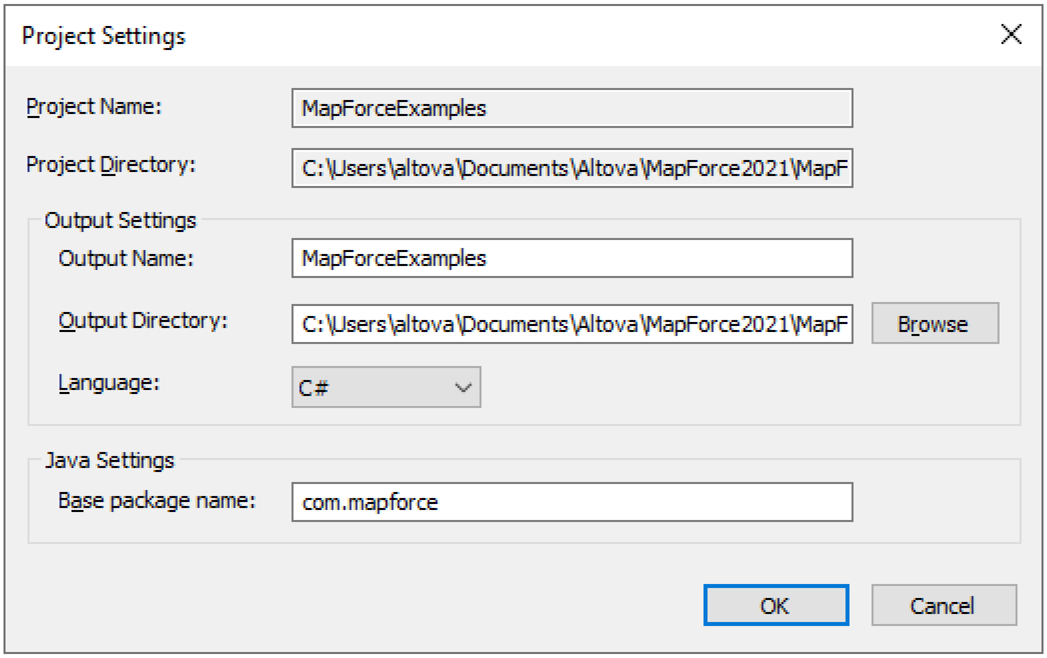
- By right-clicking the project name or a folder of interest and selecting **Create Web Link** from the context menu.
- By selecting the project name or a folder of interest and then clicking **Create Web Link** in the **Project** menu.

Either method causes the **Web Link Properties** dialog to pop up. Type the link's name that will be visible in the interface and the URL of this resource. You also need to select an icon for your resource: as a general Web link or as a link to a video. You can always change the location of any link, by dragging the link to the desired location. You can also use the combination of the **Ctrl + C** and **Ctrl + V** keys to copy a link and paste it into the desired location.

2.5.2 Project Settings

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

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



Project Settings

Project Name: MapForceExamples

Project Directory: C:\Users\altova\Documents\Altova\MapForce2021\MapF

Output Settings

Output Name: MapForceExamples

Output Directory: C:\Users\altova\Documents\Altova\MapForce2021\MapF

Language: C#

Java Settings

Base package name: com.mapforce

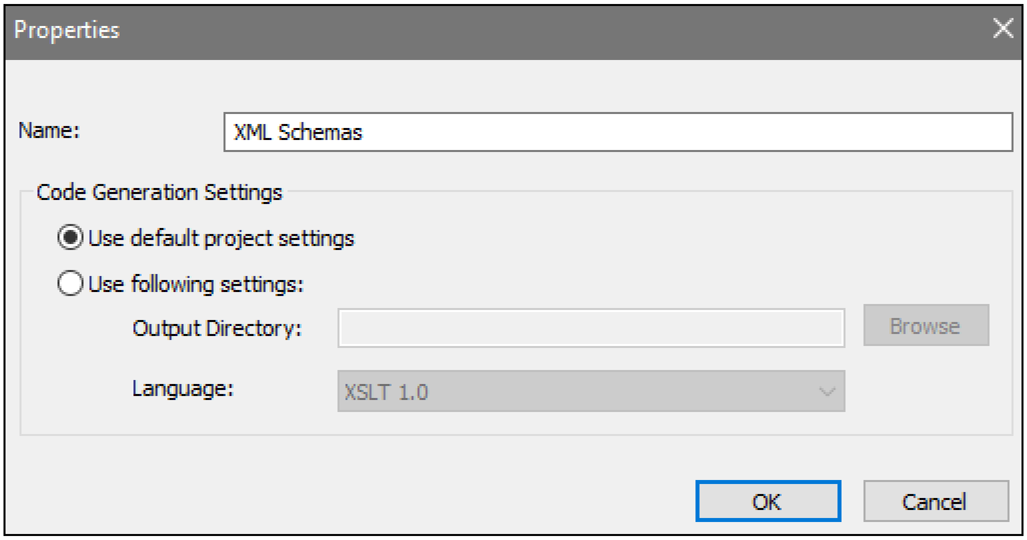
The available settings are listed below. Note that the project name and the project directory cannot be changed after the project has been created.

- *Output name:* The value entered in this text field determines the names of the generated project/solution and other objects in the generated code.
- *Output directory:* Defines the Windows folder where the generated code from all mappings in this project will be saved. By default, the output is saved to the **`MapForceExamples\output\`** directory.
- *Language:* Defines a code generation language for all mapping files in this project. For details about generating code, see Code Generation 66.
- *Base package name:* This setting applies if Java has been selected as a transformation language. The setting defines the name of the base package in the generated Java project.

2.5.3 Project Folders

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

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



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

- *Name:* This is the name of the folder in your project.
- *Use default project settings:* This option means that the code generation settings in the current folder are the same as in the entire project. Therefore, when you generate code from your project, MapForce will use the code generation settings defined in the project settings 97. If your folder requires custom code generation settings, select **Use following settings** and specify the code output directory and language as required.
- *Output directory:* This is the folder where the generated code from all the mappings in this folder will be saved.
- *Language:* This option defines the code generation language for all the mapping files in this folder.

3 Tutorials

Altova website: MapForce Video Demos

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

Example files
The mapping files illustrated or referenced in these tutorials are available in the BasicTutorials folder[17]. When you are in doubt about the possible effects of changing the original MapForce examples, create back-ups before changing them.

List of tutorials

One source to one target
This tutorial[101] shows how to use key MapForce mechanisms to map the nodes of a source file to the nodes of a target file. The tutorial goes on to explain how to convert an XML file defined by one XML schema to an XML file defined by a different XML schema.

Multiple sources to one target
This tutorial[110] shows how to merge data from multiple source XML files to one target file.

Chained mappings
In this tutorial[115], we create a simple mapping as in the second tutorial, then filter the data produced by this mapping and pass the filtered data to the second target file.

Multiple sources to multiple targets
This tutorial[123] shows how to read data from multiple XML instance files located in the same folder and write this data to multiple XML files generated on the fly.

3.1 One Source to One Target

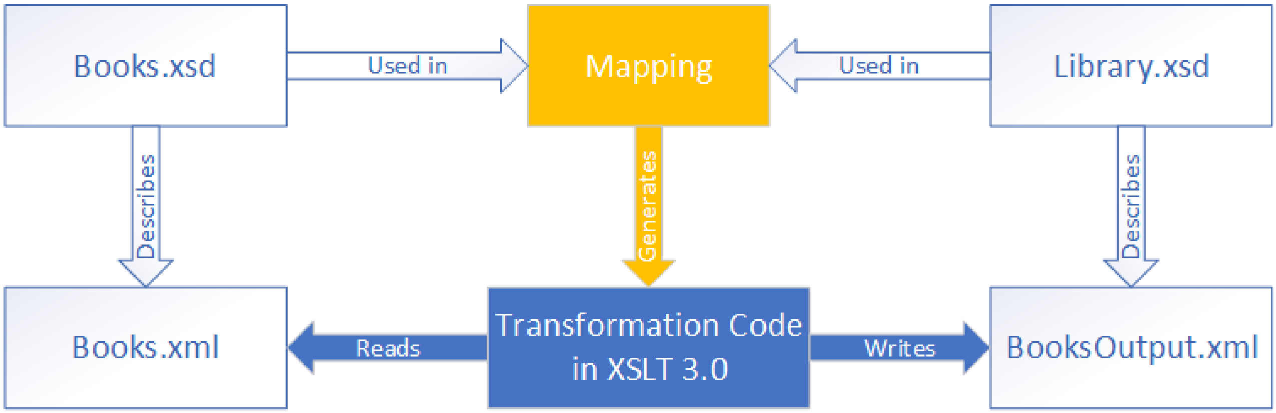
This tutorial describes how to create a mapping for one of the most basic scenarios. Our goal is to transfer data from XML file A (with XML schema A assigned to it) to XML file B (with XML schema B assigned to it). The broad outline of our method will be as follows:

1. Since we are using two data structures, we will create two components (*Source* and *Target*) in our mapping design.
2. To transform one document into another, we will select a suitable transformation language 19.
3. Then we will need to connect source nodes to the desired target nodes. It is these connections that constitute the mapping and determine what source node maps to what target node.
4. As a result of the mapping, we will get the target XML document that is valid in accordance with the target schema.
5. Finally, we can save the output XML file.

For more information about how data transformation is carried out, see the abstract model below.

Abstract model

The abstract model below illustrates the data transformation in this tutorial:



Our mapping has one source and one target. The source schema (**Books.xsd**) describes the structure of the source instance file (**Books.xml**). The target schema (**Library.xsd**) describes the structure of the target instance file (**BooksOutput.xml**). When you connect the nodes of the source to the corresponding nodes of the target, the mapping generates transformation code in XSLT 3.0. The transformation code reads data from **Books.xml** and writes this data to **BooksOutput.xml**.

Source and target files

The code listing below shows sample data from **Books.xml** that will be used as a data source.

```
<books>
   <book id="1">
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <category>Fiction</category>
      <year>1876</year>
   </book>
   <book id="2">
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <category>Fiction</category>
      <year>1912</year>
   </book>
</books>
```

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

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

Our goal is to populate the elements `<author>`, `<title>`, `<genre>` and `<publish_year>` of the target file with the content of the equivalent elements in the source file (`<author>`, `<title>`, `<category>`, `<year>`). The attribute `id` in the source file will be mapped to the `<id>` element in the target file. Finally, we will populate the `<last_updated>` element of the target file with the date and time indicating when the file was last updated.

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

3.1.1 Create and Save Design

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

Create a new design

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

- Go to the **File** menu and click **New**. Then select **Mapping** and click **OK**.
- Click 🗋 in the toolbar. Then select **Mapping** and click **OK**.

Select a transformation language

Depending on your MapForce edition, different transformation languages[19] are available. For this tutorial, we have selected XSLT3. You can select this transformation language in one of the following ways:

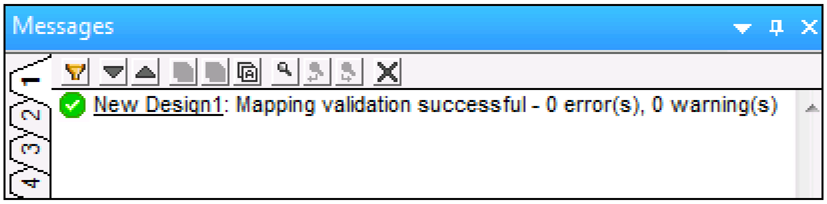
- Click **XSLT3** in the toolbar.
- Open the **Output** menu and click **XSLT 3.0**.

Validate and save the design

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

- Click **Validate Mapping** in the **File** menu.
- Click ✓ in the toolbar.

The Messages window displays the validation results as follows:



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

To save the mapping, do one of the following:

- Click **Save** in the **File** menu.
- Click 💾 in the toolbar.

For your convenience, the mapping created in this tutorial is saved as **`Tut1_OneToOne.mfd`**.

3.1.2 Add Source Component

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

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

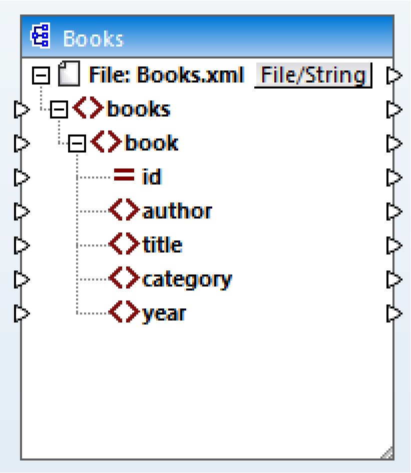
If you select one of the first two options, the **Insert XML Schema File** dialog will suggest choosing between a prepackaged schema or a local or remote file. In our tutorials, all the files are local. For more information about how to add XML files, see XML and XML Schema[132].

If a component is created from an XSD file, you will be prompted to supply an XML file that will be used as the component's data file. If a component is created from an XML file, the XSD file that is referenced from the XML file will be used to define the structure of the component's data. If no reference to an XSD file exists, MapForce will suggest generating an XSD file for this component.

Since we add the schema file first, MapForce suggests adding a sample XML file. Click **Browse** and search for **`Books.xml`** that is located in the same folder. Thus, our source file contains both a schema and content.

View the structure

Now that the source file has been added to the mapping area, you can see its structure. In MapForce, this structure is known as a mapping component or simply a component 32. You can expand elements in the component by clicking the ⊞ icon. Alternatively, you can press the **+** key on the numeric keypad. The screnshot below illustrates the source component:



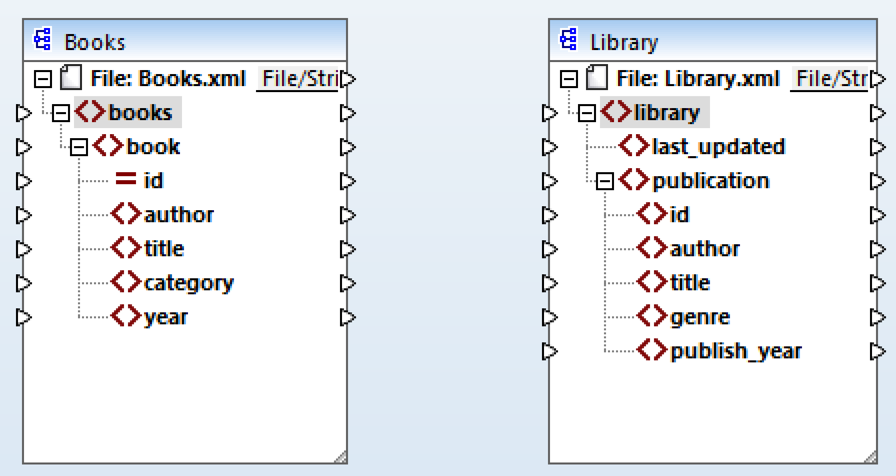
The name of the header (`Books`) only refers to the name of the component and not the name of the schema this file is based on. To see the name of the schema and other properties of the component, double-click the component's header. This will open the Component Settings 133 dialog box.

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

For more information about components, connections, general procedures and features, see Mapping Fundamentals 32.

3.1.3 Add Target Component

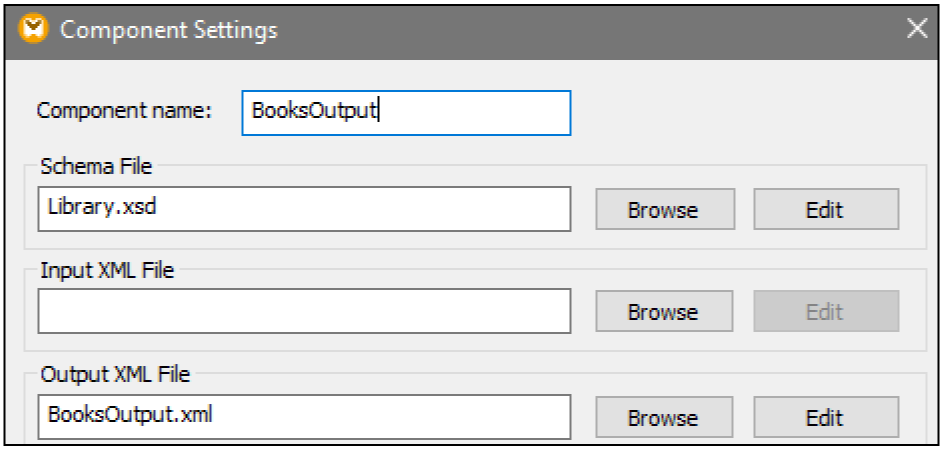
The next step is to add a target component and define its settings. Add the target file called **`Library.xsd`** to the mapping. Click **Skip** when MapForce suggests supplying an instance file. At this stage, the mapping design looks as follows:



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

Component settings

Now we need to rename the target component **`BooksOutput`**. The output file will be called **`BooksOutput.xml`**. This will allow us to avoid confusion in the next tutorials, as we are going to use a separate file called **`Library.xml`**, which has its own content and is based on the same **`Library.xsd`** schema. To rename the target file, double-click the header of the target component. This opens the Component Settings dialog box 133 (*see screenshot below*), in which we need to change the names of the target component and target file as follows:



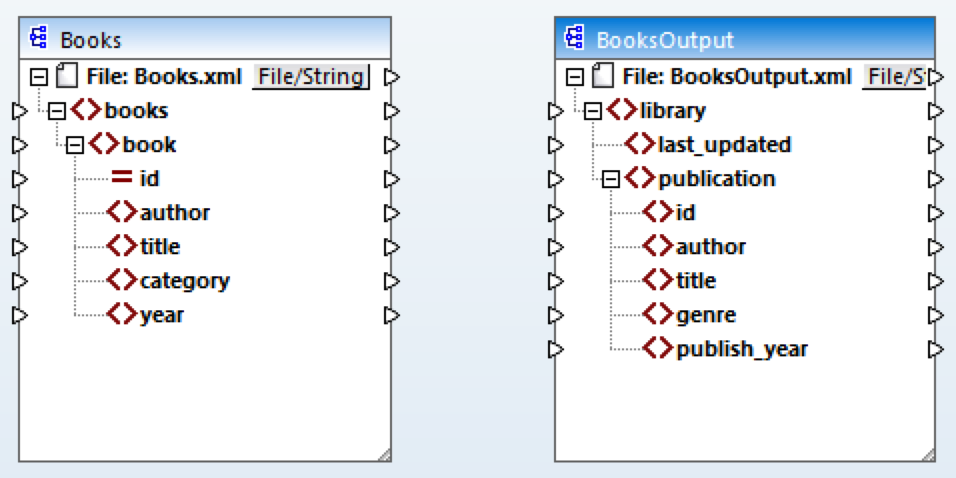
Component name: BooksOutput

Schema File: Library.xsd

Input XML File:

Output XML File: BooksOutput.xml

The mapping design now looks as follows:

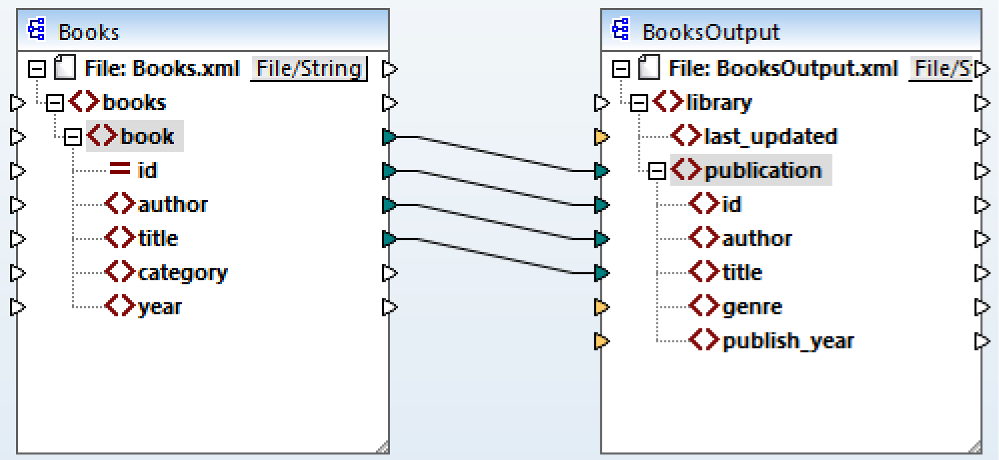


3.1.4 Connect Source and Target

In this step, we will map the data in the source file to the target file. We will also supply information about the current date and time using the XPath function `current-dateTime` 667.

Automatic connections

We will now create a mapping connection between the `<book>` element in the source component and the `<publication>` element in the target component. To do this, press and hold the output connector (the small triangle) to the right of the `<book>` element and drag the line to the input connector of the `<publication>` element in the target. When you do this, MapForce may automatically connect all the child nodes of `<book>` in the source file to the nodes with the same names in the target file. In our example, four connections have been created simultaneously (*see screenshot below*). This feature is called *Auto Connect Matching Children* 54 and can be disabled and customized if necessary.



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

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

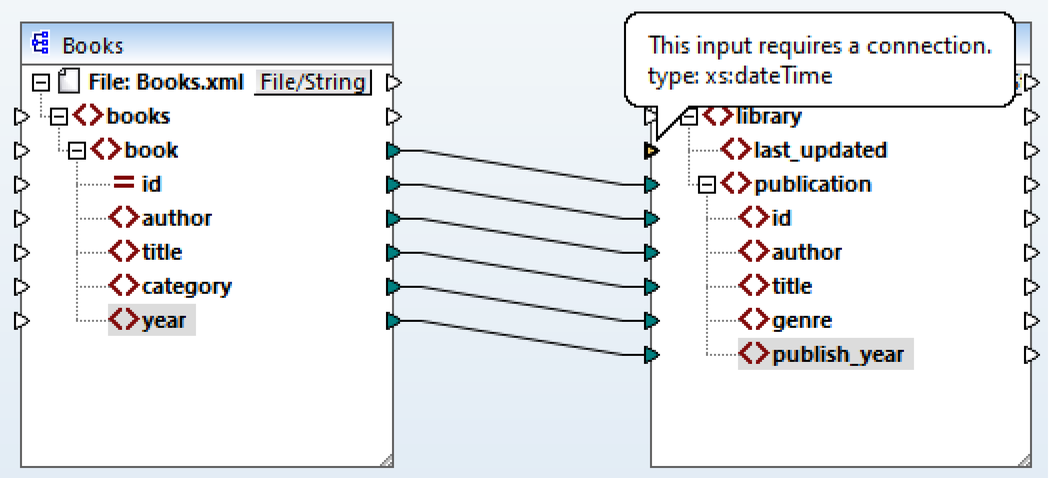
Connect mandatory items

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

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

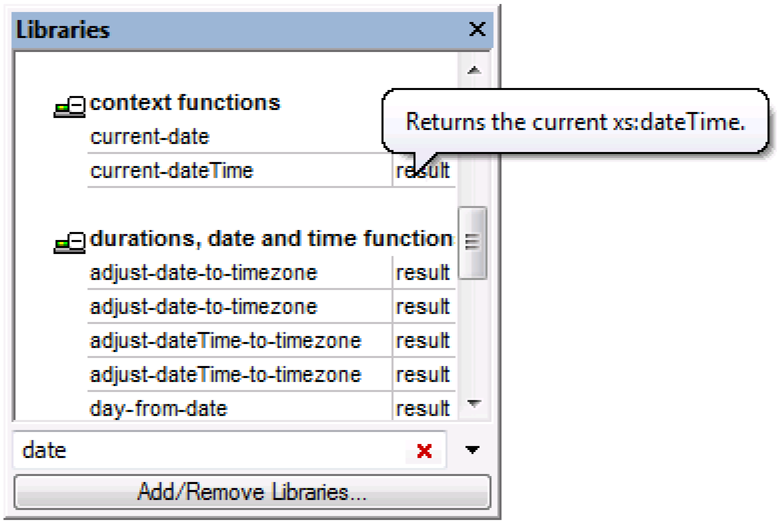
Add the current date and time

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



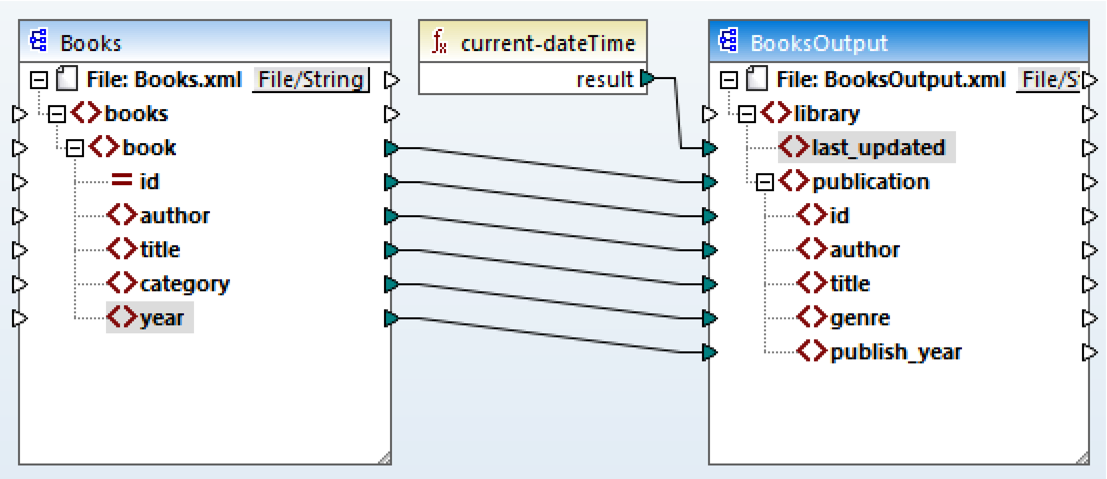
This input requires a connection.
type: xs:dateTime

You can get the current date and time by means of the **`current-dateTime`** function. To find this function, type it in the text box located at the bottom of the [Libraries window][23] (*see screenshot below*). Alternatively, double-click an empty area inside the Mapping pane and start typing `current-date`.



Returns the current xs:dateTime.

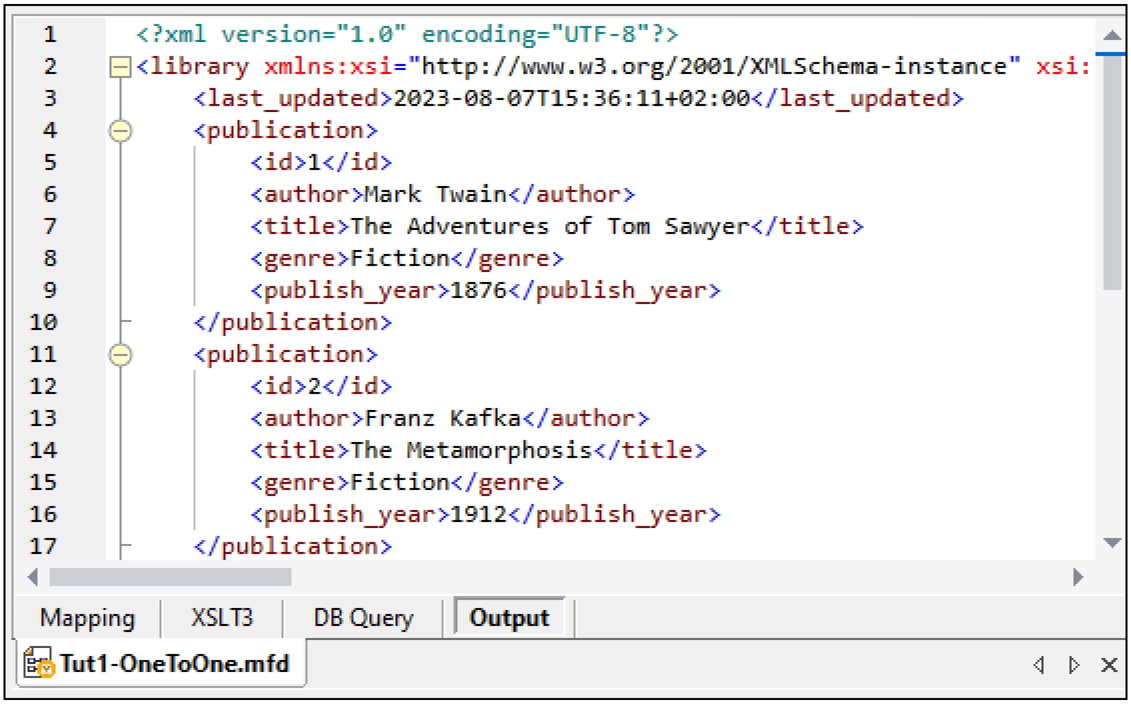
To add the function to the mapping, drag the function into the Mapping pane. Then connect its output to the input of the `<last_updated>` element (*see screenshot below*).



You can now validate and save your mapping, as shown in Create and Save Design[102].

3.1.5 Preview Mapping Result

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



```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    <last_updated>2023-08-07T15:36:11+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>
```

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

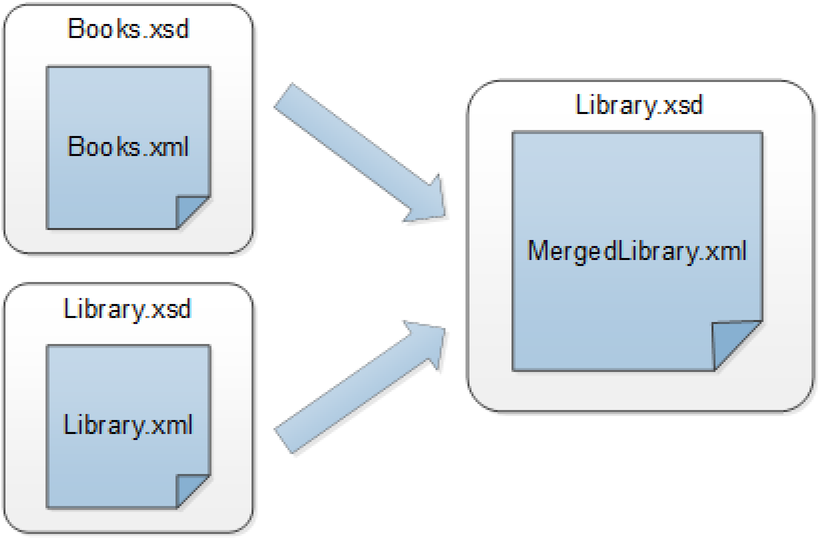
To configure MapForce to write the output directly to final files instead of temporary ones, go to **Tools | Options | General** and select the check box *Write directly to final output files*. Note that enabling this option is not recommended while you follow this tutorial, because you may unintentionally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

Preview generated code
You can also preview the generated XSLT code that performs the transformation. To preview the code, open the XSLT3 pane located at the bottom of the Mapping window. To generate the XSLT3 code and save it to a file, select the menu item **File | Generate Code in | XSLT 3.0**. When prompted, select a folder where the generated code must be saved. After the code generation has been completed, the destination folder will include the following two files:

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

3.2 Multiple Sources to One Target

In this tutorial, you will learn to merge the data from a new file called **`Library.xml`** with the data from **`Books.xml`**. The result will be a target file called **`MergedLibrary.xml`**, which will contain the data from both source files. The target file will be based on the **`Library.xsd`** schema. Note that the source files have different schemas. If the source files had the same schema, you could also merge their data using a different approach, described in [Multiple Sources to Multiple Targets](123). The image below represents an abstract model of the data transformation described in this tutorial.



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

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

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

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

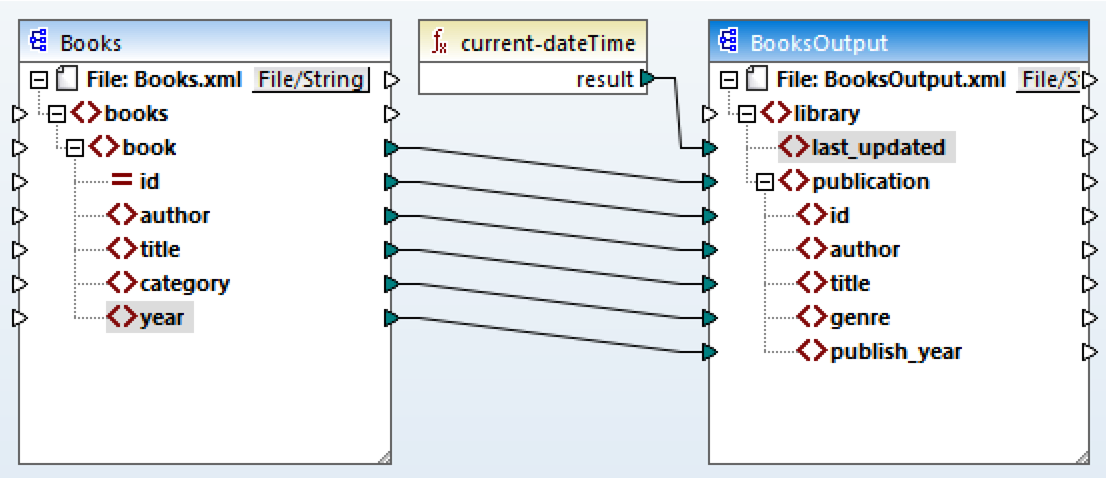
This is how we want our merged data to look in the target file called **`MergedLibrary.xml`**:

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

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

3.2.1 Prepare Mapping Design

The starting point of this tutorial is the **Tut1_OneToOne.mfd** mapping (*screenshot below*) that was designed in Tutorial 1 101. Before making any changes to the mapping, make sure to save this design with a new name in the **BasicTutorials** folder.

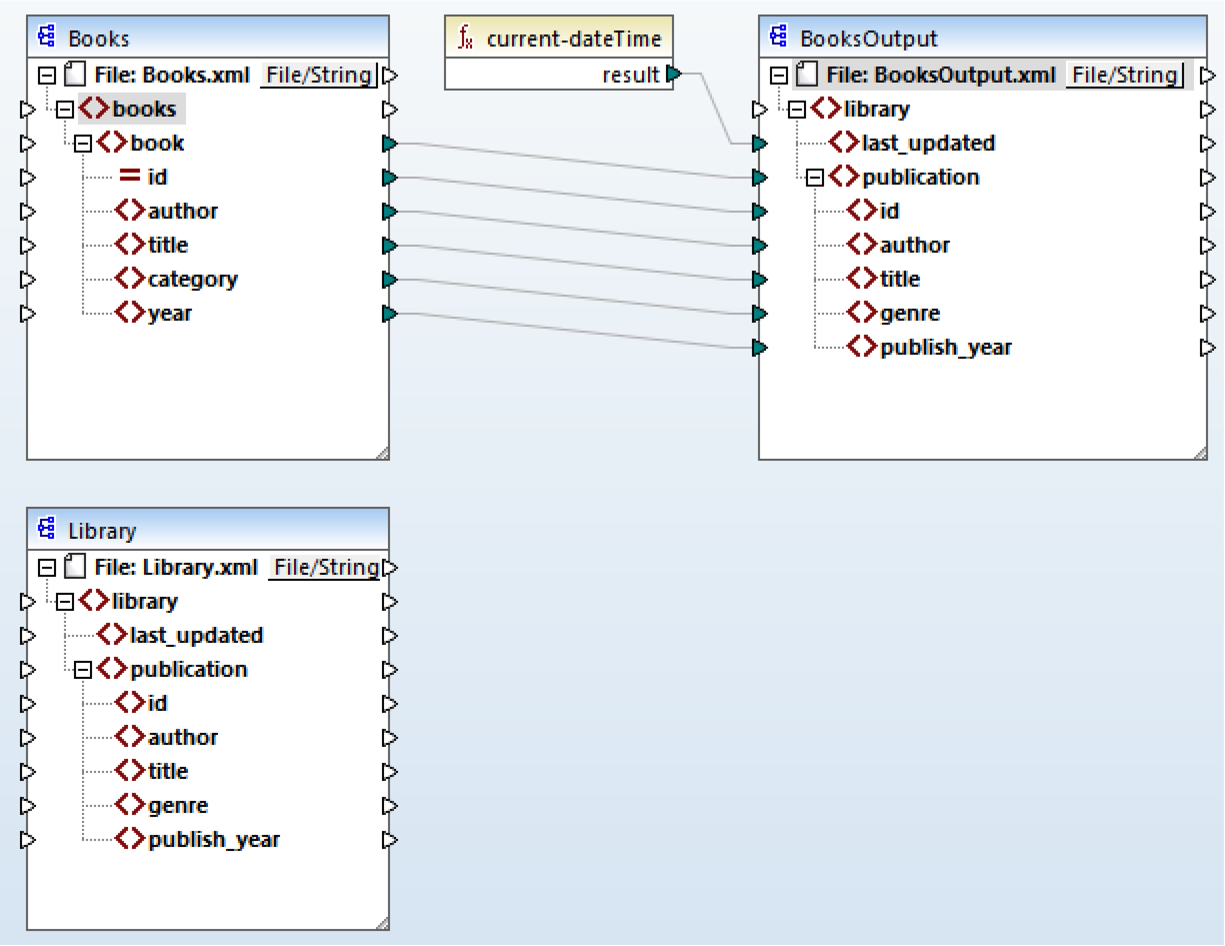


3.2.2 Add Second Source

The next step is to add the second source file: Insert **Library.xml** into the mapping. Since the schema (**Library.xsd**) is referenced in this XML file, you do not need to add the schema file in a separate step. To check whether the schema reference is correct, open the Component Settings 133.

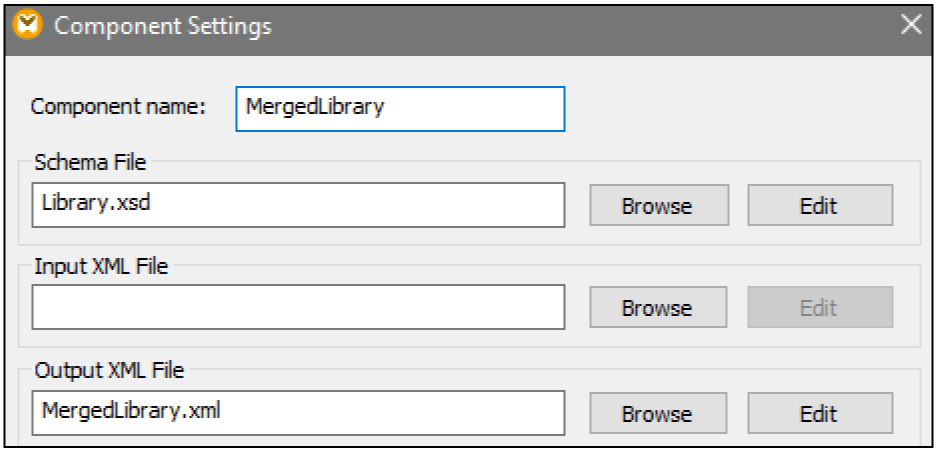
Then press and hold the header of the new component and place it under the `Books` component. You can always move components in any direction. Nevertheless, placing a source component to the left of a target component will make your mapping easier to read and understand. This is also the convention for all the mappings illustrated in this documentation and the sample mapping files accompanying your MapForce installation.

At this stage, the mapping design looks as follows:



3.2.3 Configure Output

At his stage, the mapping has two source components (`Books` and `Library`) and one target component (`BooksOutput`). For consistency and to avoid confusion, we will need to change the settings of the `BookOutput` component. Double-click the header of the target component. This will open the Component Settings dialog box 133. Configure the settings as shown below.



Component name: MergedLibrary

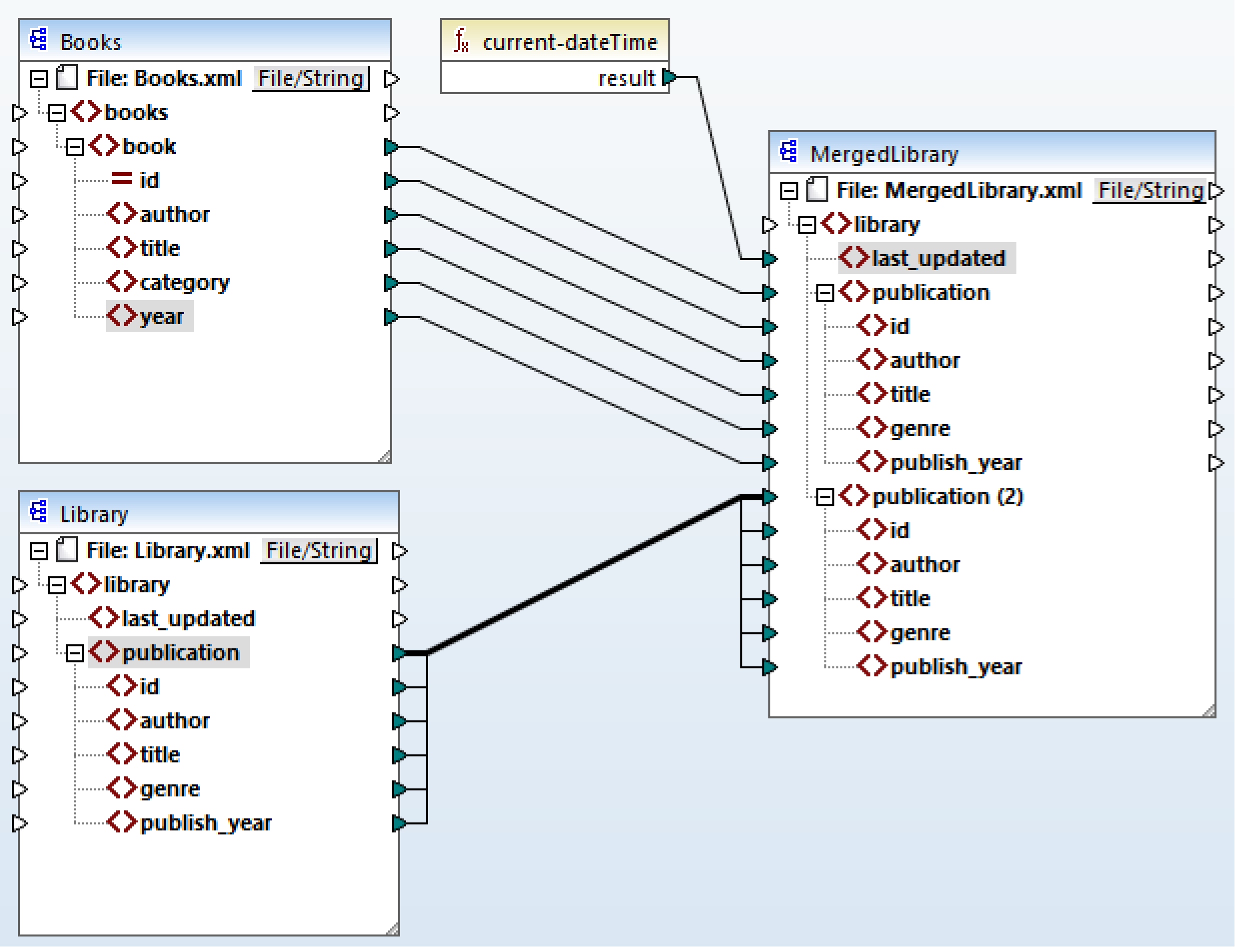
Schema File: Library.xsd

Input XML File:

Output XML File: MergedLibrary.xml

3.2.4 Connect Second Source and Target

The last step of the tutorial is to connect the second source component (`Library`) with the target component (`MergedLibrary`). In order to do this, connect the `<publication>` element in **`Library.xml`** with the `<publication>` element in **`MergedLibrary.xml`**. Since the target input connector already has a connection, MapForce will prompt you to replace the connection or to duplicate the input. In this tutorial, our goal is to map data from two sources to one target. Therefore, click **Duplicate Input**. By doing so, you configure the target component in such a way that it will accept the data from the second source, too. The mapping now looks as follows:



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

Copy-all connection

Since the child elements of the `publication` element in the `Library` component and the `publication` element in the `MergedLibrary` component have the same names and data types, these elements are connected with one thick line. Such a connection is called a *copy-all connection* 56, which makes the mapping easier to understand.

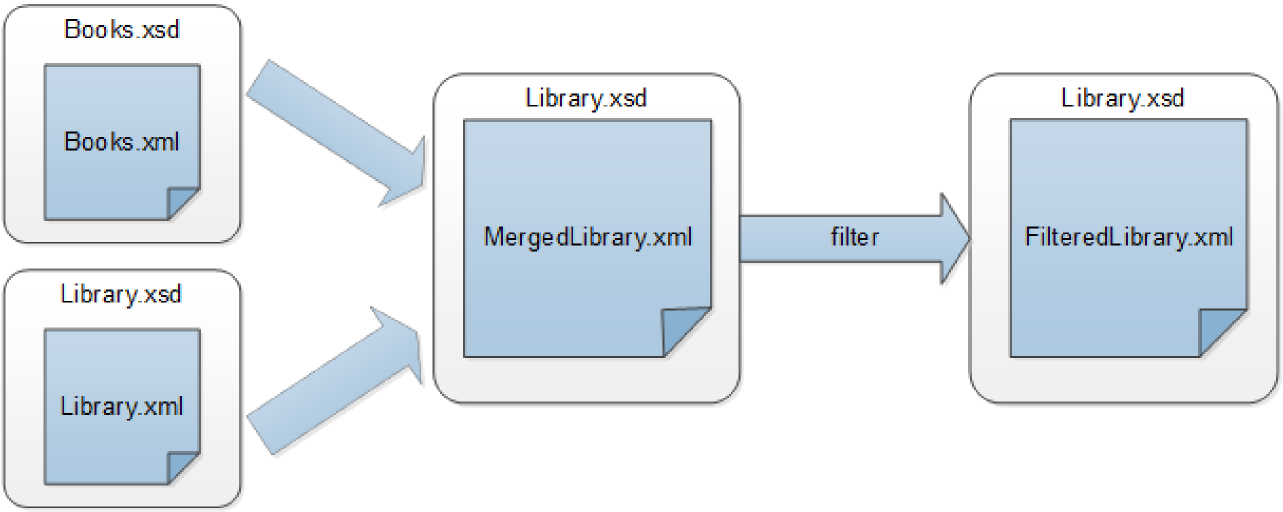
Preview the output

Open the Output pane to view the result. You will notice that the data from both **`Books.xml`** and **`Library.xml`** has now been merged into the new **`MergedLibrary.xml`** file. For your convenience, the mapping design in this tutorial is saved as **`Tut2_MultipleToOne.mfd`**. This mapping will be used as a starting point in the next tutorial 115.

3.3 Chained Mapping

Altova website: Chained Mapping Video Tutorial

This tutorial shows how to work with multiple target components. The goal of the tutorial is to merge data from two source files into one target file, then filter the data of this target file, and pass the filtered data to the second target file. The image below illustrates an abstract model of the data transformation described in this tutorial.



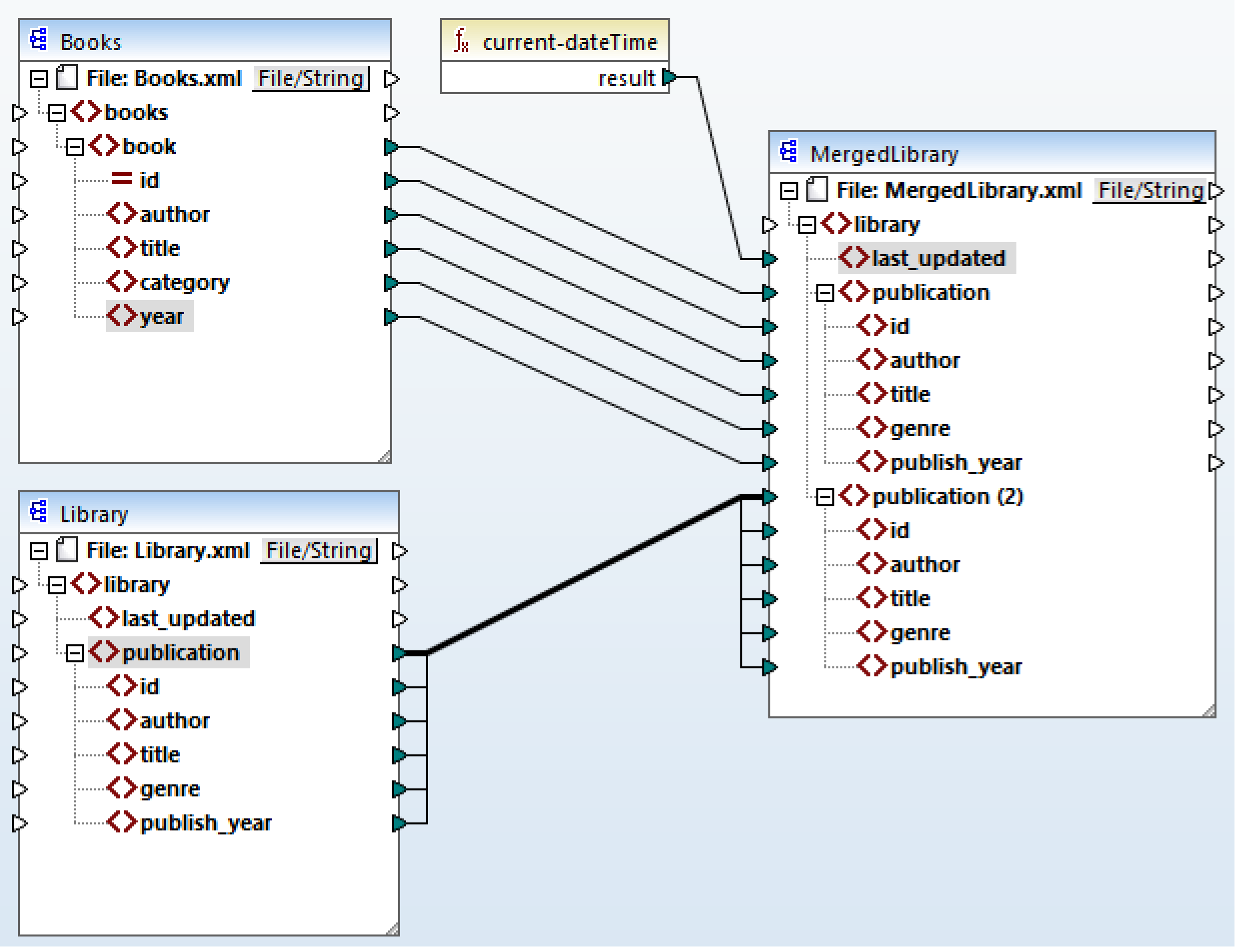
In the diagram above, the data is first merged from two source files (**Books.xml** and **Library.xml**) into one target file called **MergedLibrary.xml**. Then the data is transformed with a filtering function and passed further to the next component called **FilteredLibrary.xml**. Note that **FilteredLibrary.xml** is based on the **Library.xsd** schema. The intermediate component acts both as a data target and source. In MapForce, this technique is known as a *chained mapping*. Chained mappings allow previewing and saving the intermediate result(s) of the mapping (in our case, **MergedLibrary.xml**) and the result of the last target component (in our case, **FilteredLibrary.xml**).

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

For another example of a chained mapping, see the video tutorial linked at the top of the page.

3.3.1 Prepare Mapping Design

The starting point of this tutorial is **Tut2_MultipleToOne.mfd** (*see screenshot below*). This mapping was designed in the previous tutorial 110. Before making any changes to the mapping, make sure to save this design with a new name in the **BasicTutorials** folder.

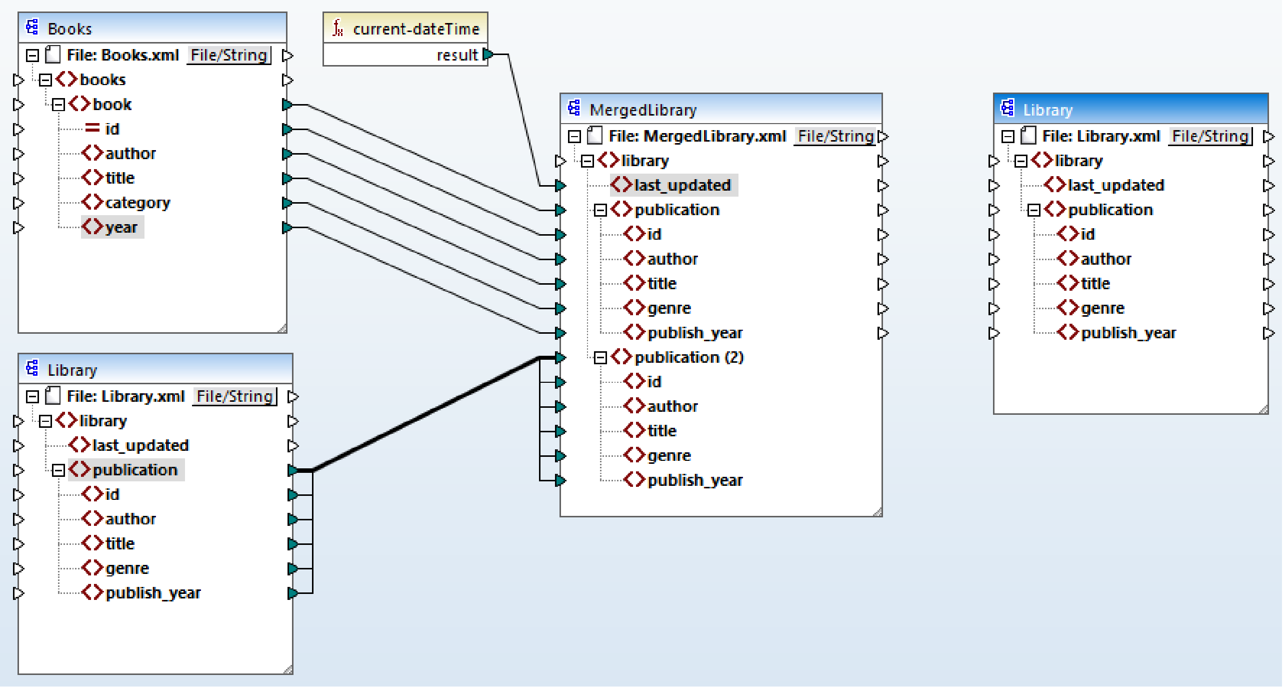


3.3.2 Configure Second Target

The next step is to add and configure the second target file, which will contain only a subset of the `publication` data from **`MergedLibrary.xml`**.

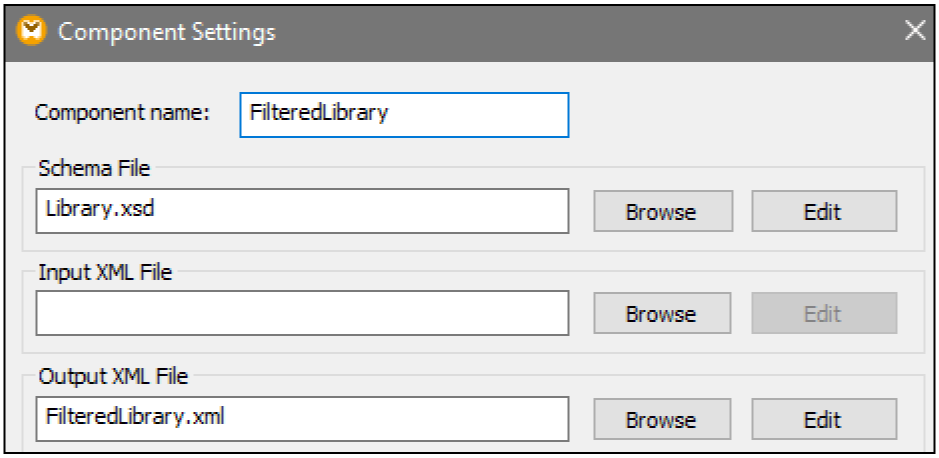
Add the second target component

Add **`Library.xsd`** to the mapping and click **Skip** when you are prompted to supply a sample instance file. The second target component has only structure but no content. At a later stage, we will map the filtered data to this target file. The mapping design now looks as follows:



Configure the second target component

As shown above, the mapping now has two source components (`Books` and `Library`) and two target components (`MergedLibrary` and `Library`). To avoid confusion, we will rename the newly added component `FilteredLibrary`. To do this, double-click the header of the right-most component and edit the component settings 133 as follows:



Component name: FilteredLibrary

Schema File

Library.xsd

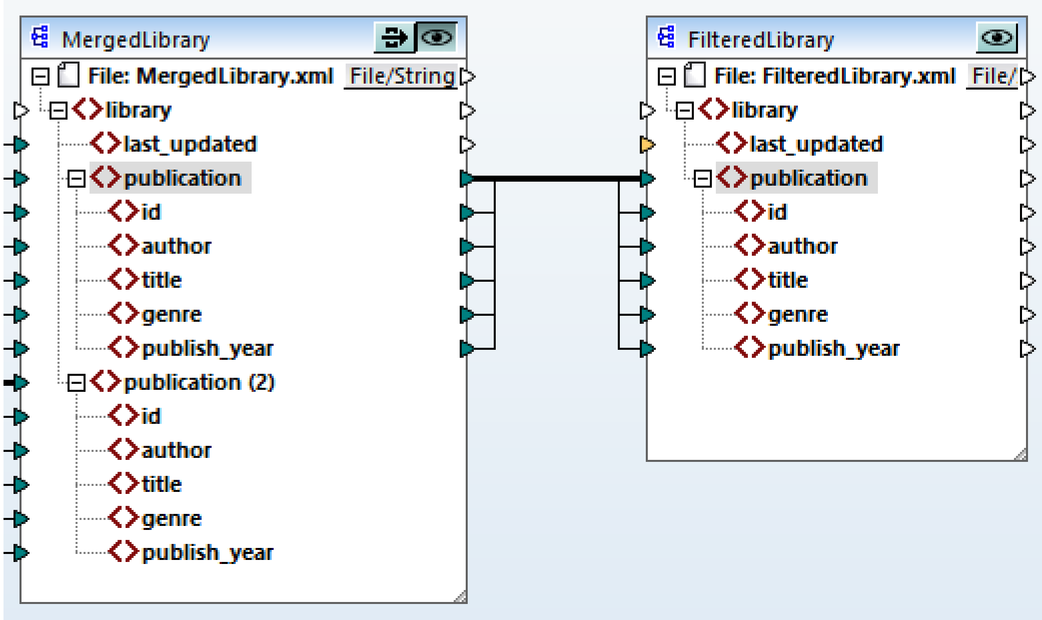
Input XML File

Output XML File

FilteredLibrary.xml

3.3.3 Connect Targets

The next step is to map the `publication` element in `MergedLibrary` to the `publication` element in `FilteredLibrary`. When you connect the output connector of `MergedLibrary` with the input connector of `FilteredLibrary`, MapForce will inform you that you have created multiple target components in the mapping. Notice that new buttons are now available in the upper-right corner of both target components: 👁 (**Preview**) and ⇛ (**Pass-through**). These buttons will be used and explained in the next steps.

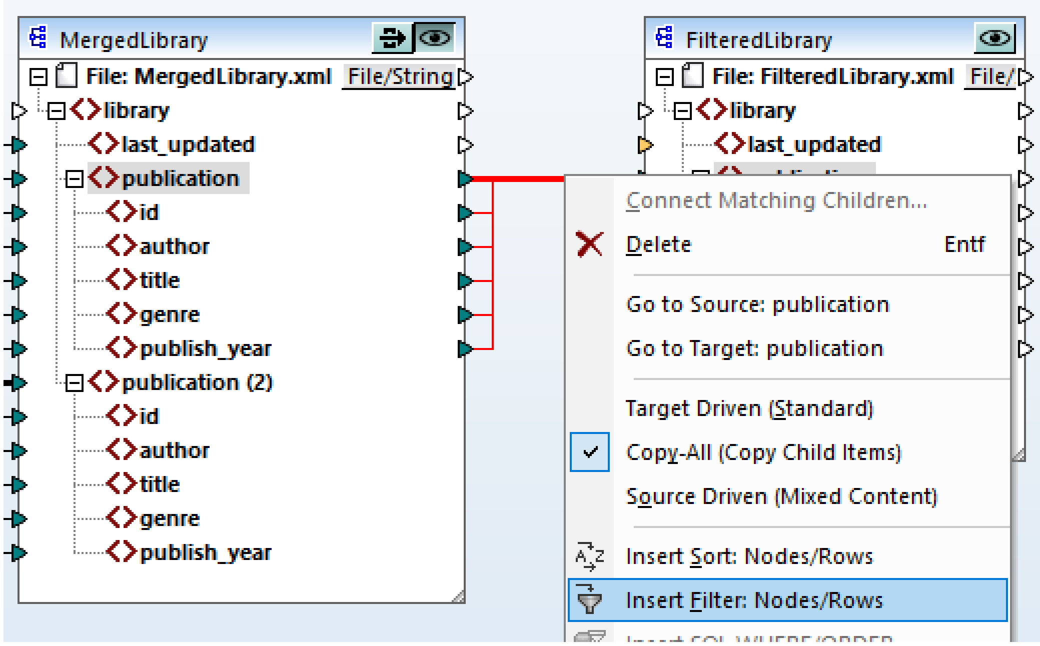


3.3.4 Filter Data

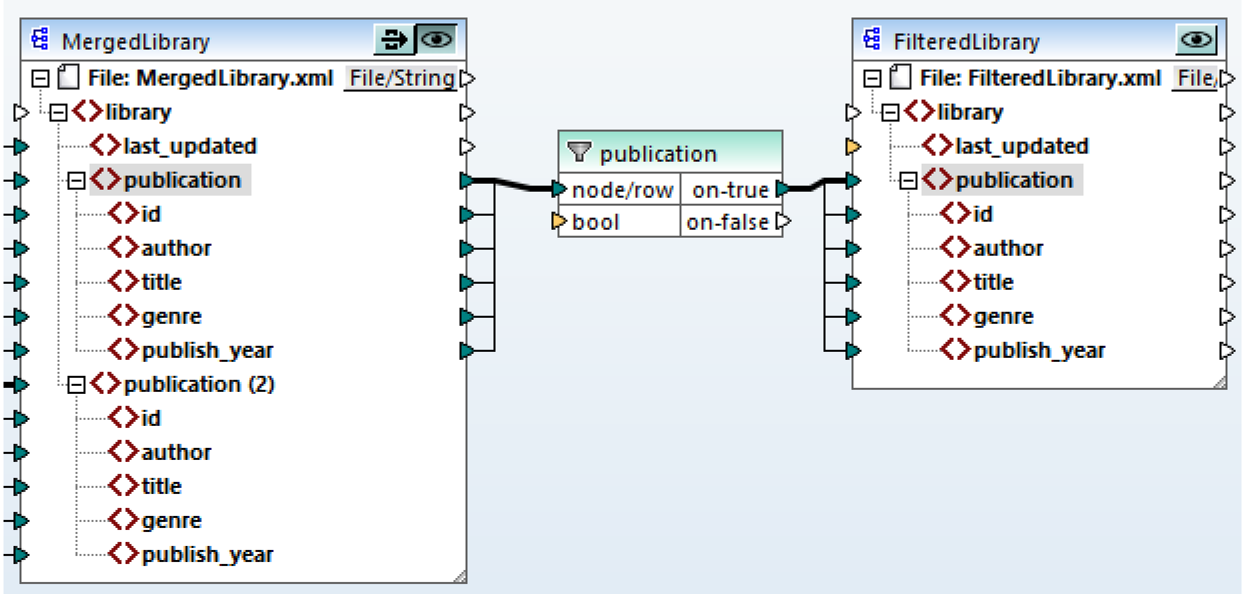
In this step, we will filter the data from `MergedLibrary` in such a way that only the books published after 1900 will be passed to the `FilteredLibrary` component. We will use a Filter component for this purpose.

Add a filter

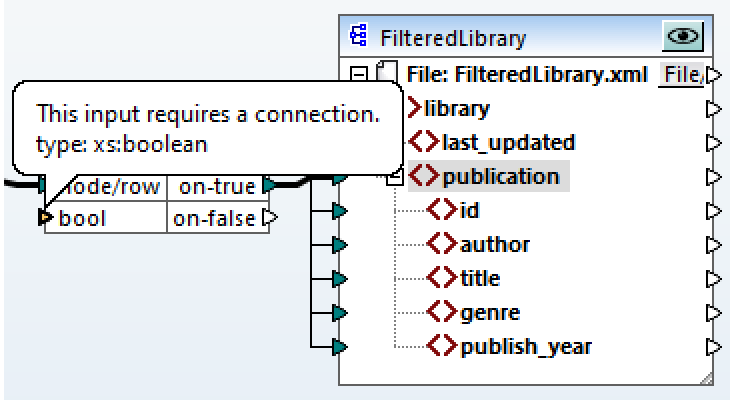
To add a filter, right-click the connection between `MergedLibrary` and `FilteredLibrary` and select **Insert Filter: Nodes/Rows** from the context menu (*screenshot below*).


MergedLibrary
File: MergedLibrary.xml File/String
library
last_updated
publication
id
author
title
genre
publish_year
publication (2)
id
author
title
genre
publish_year
FilteredLibrary
File: FilteredLibrary.xml File/
library
last_updated
Connect Matching Children...
Delete Entf
Go to Source: publication
Go to Target: publication
Target Driven (Standard)
Copy-All (Copy Child Items)
Source Driven (Mixed Content)
Insert Sort: Nodes/Rows
Insert Filter: Nodes/Rows

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



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

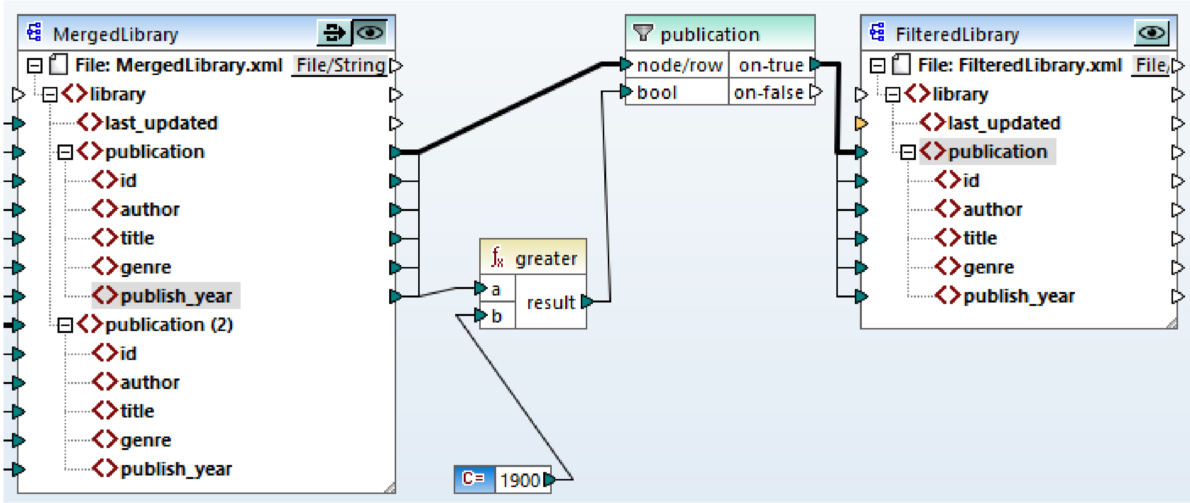


This input requires a connection.
type: xs:boolean

Only books after 1900

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

1. Click **Constant** in the toolbar and type *1900* in the text bar. Select *Number* as a type.
2. Add the **`greater`** function to the mapping.
3. Make the mapping connections to and from the **`greater`** function, as shown below. The **`greater`** function will compare the value of the `publish_year` element of each publication with the value of the constant. Only those publication records whose publication year is greater than 1900 will be mapped to the target.



3.3.5 Preview and Save Output

We are now ready to preview and save the output of each target component. When there are multiple target components in the same mapping, each target component has a 👁 (**Preview**) button. Only one component at a time can have the preview enabled. Using the **Preview** buttons, you can see the intermediate mapping result (in our example, the data mapped to the `MergedLibrary` component) as well as the final result of the chained mapping (the data mapped to the `FilteredLibrary` component). The `MergedLibrary` component also has a ⇒ (**Pass-through**) button. The **Pass-through** button controls how output will be generated (*see details below*).

Preview and save intermediate and final outputs
If you want to view and save the outputs of both the `MergedLibrary` and `FilteredLibrary` components, take the steps below:

1. Press the **Pass-through** button in the `MergedLibrary` component.
2. Make sure the **Preview** button of the `FilteredLibrary` component is also pressed.
3. Open the Output pane. The **Back** and **Forward** buttons enable you to switch between the outputs.
4. Switch to the output you wish to save to a file and click **Save Output File** in the toolbar. If you wish to save both outputs, click the **Save All Generated Outputs** button in the toolbar.

When the pass-through feature is active, the *Input XML File* field of the intermediate component is automatically deactivated. This is because the output generated when you preview the first portion of the mapping between the sources (`Books` and `Library`) and the first target (`MergedLibrary`) is used by default as input when you preview the second portion of the mapping between the first target (`MergedLibrary`) and the second target (`FilteredLibrary`).

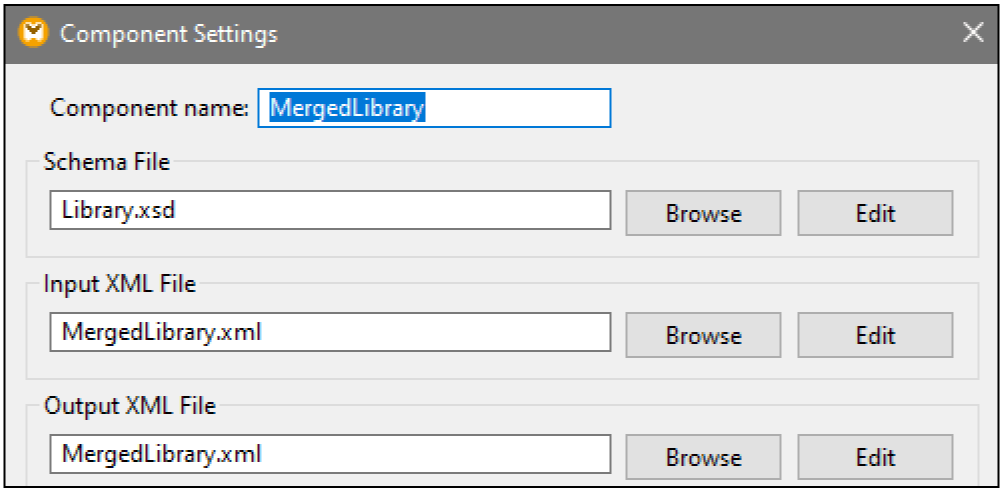
Preview and save intermediate output
Intermediate components with the pass-through button active cannot be previewed. The preview button of the intermediate component is automatically disabled, because it is not meaningful to preview and let data pass through at the same time. If you want to view and save the output of the intermediate component (`MergedLibrary`) only, take the steps below:

1. Deactivate the **Pass-though** button of the `MergedLibrary` component if the button was previously enabled.
2. Click the **Preview** button on the `MergedLibrary` component.
3. Open the Output pane.
4. Click the **Save Output File** button in the toolbar to save the output to a file.

Preview and save final output
If you want to view and save only the data mapped from the intermediate component to the second target component, follow the instructions below.

1. Deactivate the **Pass-though** button of the `MergedLibrary` component if the button was previously enabled.
2. Double-click the header of the `MergedLibrary` component.
3. Make sure to supply the same file name in the *Input XML File* field as in the *Output XML File* field (*screenshot below*). When you disable the **Pass-through** button, you can choose what input file should be read by the intermediate component. In most cases, this should be the same file as defined in the *Output XML File* field. It usually makes sense for the intermediate component to receive one file for processing and forward the same file to the subsequent mapping rather than look for a different file name.

Having the same input and output file for the intermediate component is important when the **Pass-through** button of the intermediate component is deactivated. This ensures that the output generated when you preview the first portion of the mapping between the sources (`Books` and `Library`) and the first target (`MergedLibrary`) is used as input when you preview the second portion of the mapping between the first target (`MergedLibrary`) and the second target (`FilteredLibrary`). If you execute your mapping with MapForce Server (*Professional and Enterprise editions*) or via generated code, the same names of the input and output files of the intermediate component ensure that the mapping chain is not broken. Note that *not* having the same input and output file names in the intermediate component (when the **Pass-through** button is inactive) might cause errors in MapForce, generated code, and MapForce Server execution.



Component Settings

Component name: MergedLibrary

Schema File

Library.xsd

Input XML File

MergedLibrary.xml

Output XML File

MergedLibrary.xml

4. Press the **Preview** button of the `FilteredLibrary` component.
5. Open the Output pane.
6. Click the **Save Output File** button in the toolbar to save the output to a file.

Important

- The pass-through feature is available only for file-based components, such as XML, CSV, and text. Database components (*Professional and Enterprise editions*) can be intermediate, but the pass-through button is not shown.
- When the mapping is executed by MapForce, the setting *Write directly to final output file* (configured from **Tools | Options | General**[1077]) determines whether outputs are saved as temporary files or as physical files. Note that this is only valid when the mapping is previewed directly in MapForce. If this mapping is executed by MapForce Server or by generated code, actual files will be produced at each stage of the mapping chain.

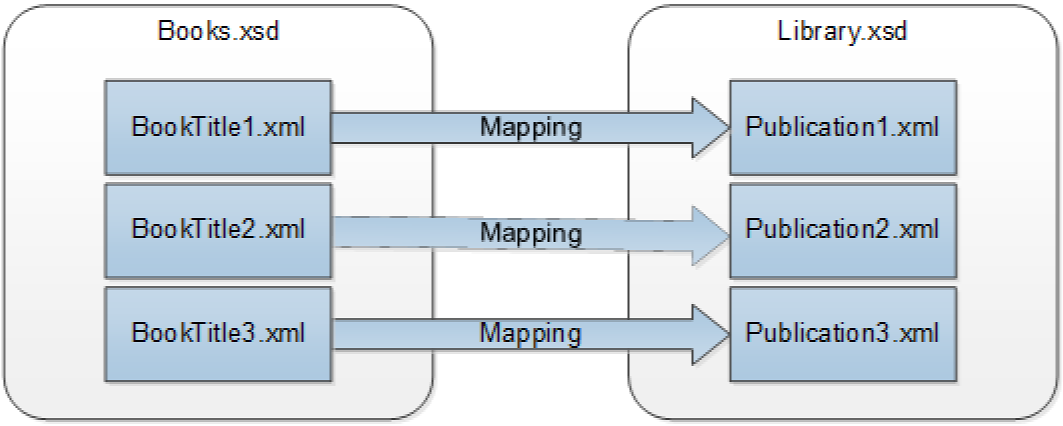
You have now finished designing the chained mapping that produces two output files. For your convenience, the mapping design in this tutorial is saved as **`Tut3_ChainedMapping.mfd`**.

3.4 Multiple Sources to Multiple Targets

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

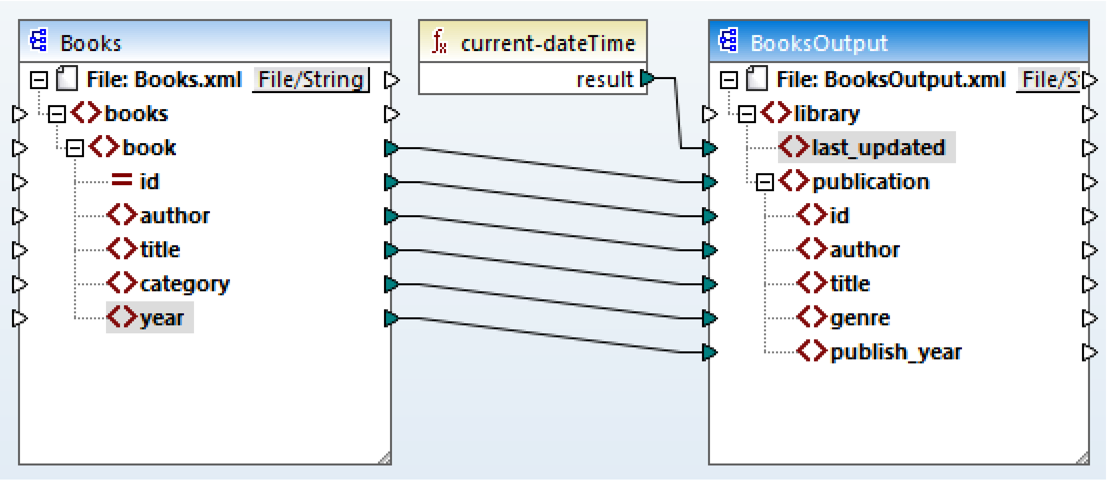
1. To read data from multiple XML files located in the same directory. The files are based on the same source schema.
2. For each source XML file, to generate a new XML target file. The target files will be based on a new target schema.

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



Broad outline

The starting point of this tutorial is the **`Tut1_OneToOne.mfd`** mapping from the first tutorial 101 (*screenshot below*).



Modify source component

We will modify the component settings of the source component so that it reads data from multiple source files: **`BookTitle1.xml`**, **`BookTitle2.xml`**, and **`BookTitle3.xml`**. Each of these files is based on **`Books.xsd`** and stores one book record (*see below*).

`BookTitle1.xml`

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

`BookTitle2.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Books.xsd">
   <book id="2">
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <category>Fiction</category>
      <year>1912</year>
   </book>
</books>
```

`BookTitle3.xml`

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

Modify target component

We will also configure the target component in such a way that the data will be written to multiple target files. The target files will be based on the same schema called **`Library.xsd`**. The generated target files will be called **`Publication1.xml`**, **`Publication2.xml`**, and **`Publication3.xml`** (*code listings below*).

`Publication1.xml`

```
<library>
   <publication>
      <id>1</id>
      <author>Mark Twain</author>
```

```
            <title>The Adventures of Tom Sawyer</title>
            <genre>Fiction</genre>
            <publish_year>1876</publish_year>
         </publication>
      </library>
```

Publication2.xml

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

Publication3.xml

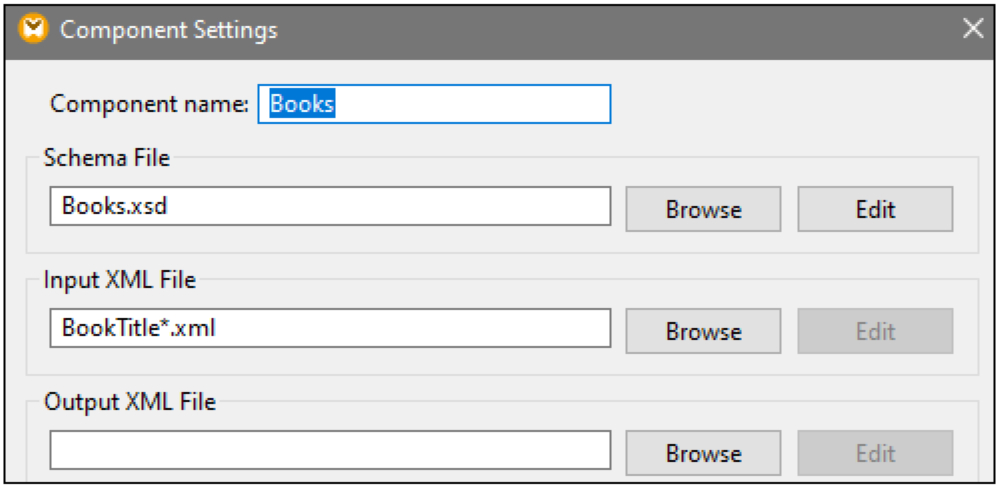
```
      <library>
         <publication>
            <id>3</id>
            <author>Herman Melville</author>
            <title>Moby Dick</title>
            <genre>Fiction</genre>
            <publish_year>1851</publish_year>
         </publication>
      </library>
```

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

3.4.1 Configure Input

The first step is to modify the component settings of the source component. Before changing the component settings, make sure to save your mapping with a new name in the **BasicTutorials** folder.

To instruct MapForce to process multiple XML instance files, double-click the component's header and type **BookTitle*.xml** in the *Input XML File* field (*screenshot below*). The asterisk (*) in the file name instructs MapForce to use all the files with the BookTitle prefix as mapping inputs. Because the path is relative, MapForce will look for all BookTitle files in the same directory as the mapping file. You can also enter an absolute path if necessary.



Component Settings

Component name: Books

Schema File

Books.xsd

Input XML File

BookTitle*.xml

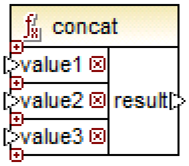
Output XML File

3.4.2 Configure Output Part 1

The next step is to create the file name of each output file. For this purpose, we will use the **concat** 592 function that joins all the values supplied to it. When these values are joined together, they will create an output file name (e.g., **Publication1.xml**). To generate the file names using the **concat** function, take the steps described below.

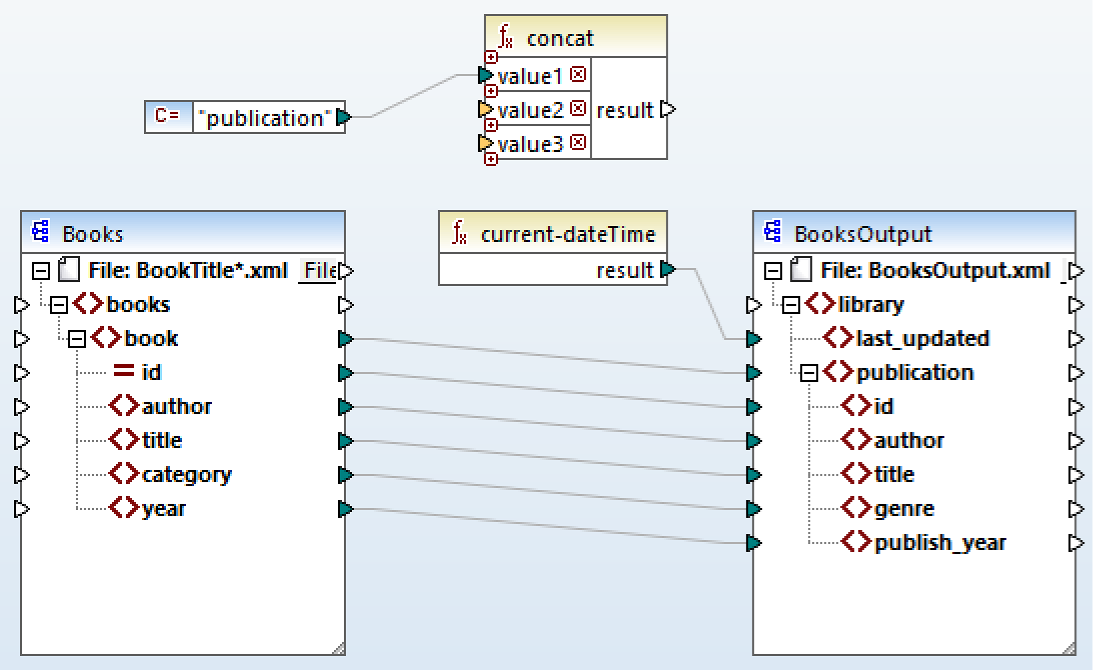
Step 1: Add the concat function

Add 106 the **concat** function (*screenshot below*) to the mapping area. By default, this function has two parameters when it is added to the mapping. In our example, we need three parameters. Click ⊞ (**Add parameter**) inside the function component and add a third parameter to it. Note that clicking ☒ (**Delete parameter**) deletes a parameter.



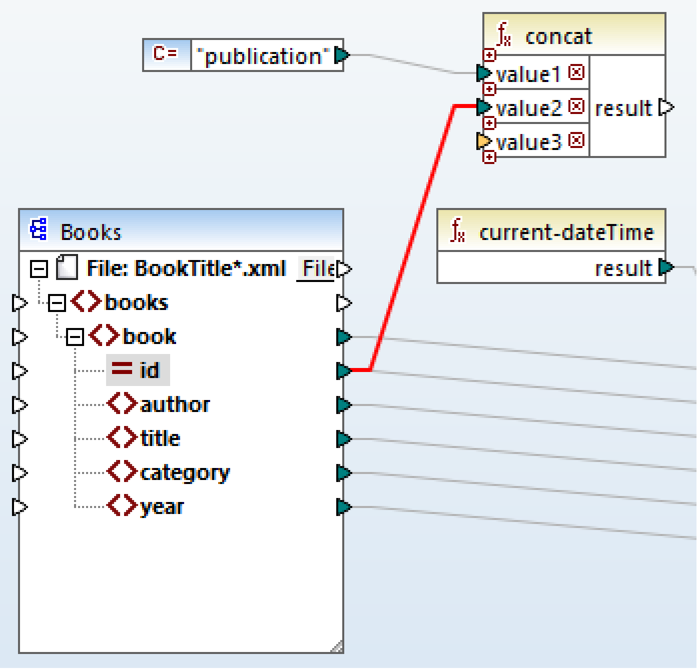
Step 2: Insert a constant

The next step is to add a constant. When you are prompted to supply a value, enter `publication` and leave the *String* option unchanged. Connect the constant with `value1` of the **concat** function, as shown in the screenshot below:



Step 3: Supply IDs

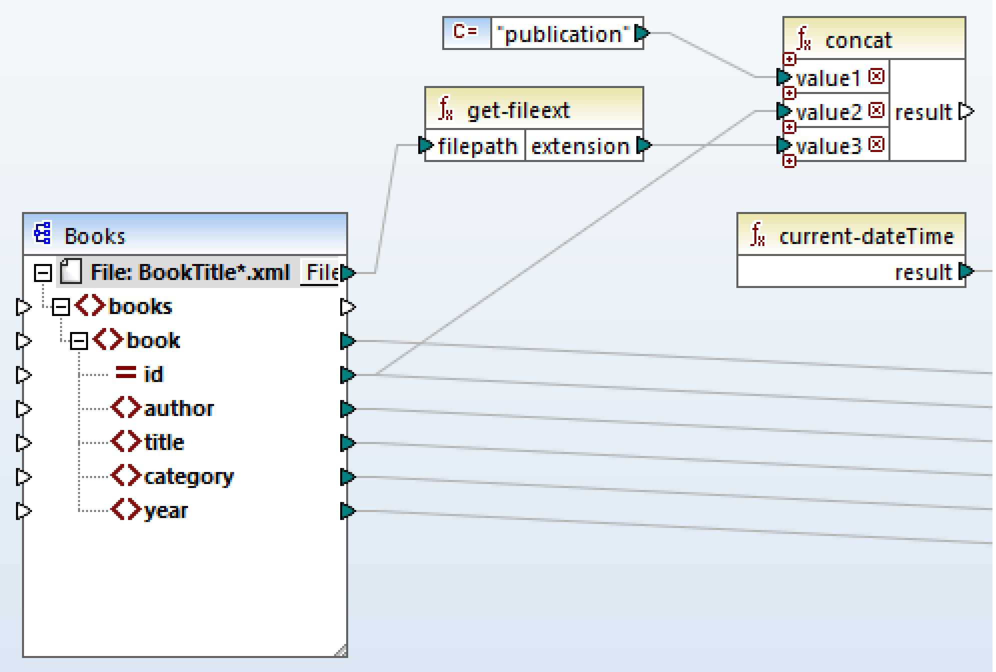
Connect the `id` attribute of the source component with `value2` of the **`concat`** function (*screenshot below*). The `id` attribute of the source XML file supplies a unique identifier value for each file. This helps prevent the files from being generated with the same name. The connection becomes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



Step 4: Extract the file extension

Add the **get-fileext** 537 function to the mapping area. Then create a connection from the top node of the source component (**File: BookTitle*.xml**) to the filepath parameter of this function (*screenshot below*).

The next step is to connect the extension parameter of the **get-fileext** function to value3 of the **concat** function. By doing this, you are extracting only the extension part (in this case, **.xml**) from the source file name and passing it to the output file name.

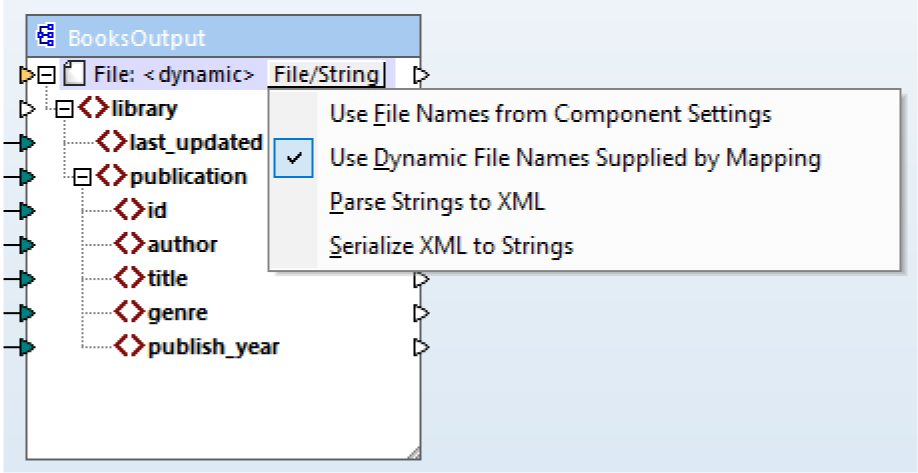


3.4.3 Configure Output Part 2

In the second part of the output configuration, we will instruct MapForce to generate output files dynamically, which means that the every output file will receive its name based on the arguments supplied to the **concat** function. In order to do this, we will use dynamic file names (*see subsections below*). For more information about dynamic file names, see Processing Multiple Input and Output Files 746.

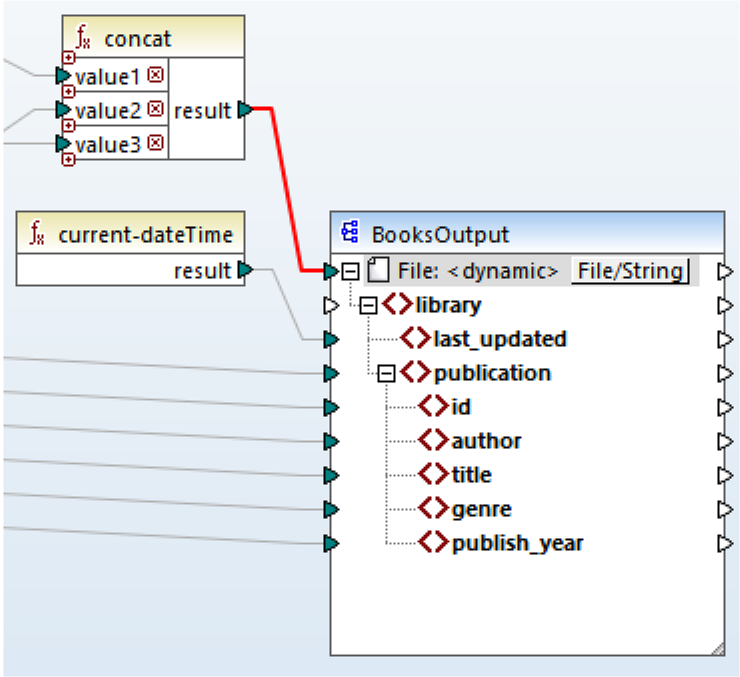
Step 1: Set dynamic file names

To instruct MapForce to generate the instance files dynamically, click File or File/String next to the File node of the target component and select **Use Dynamic File Names Supplied by Mapping** from the context menu (*screenshot below*).



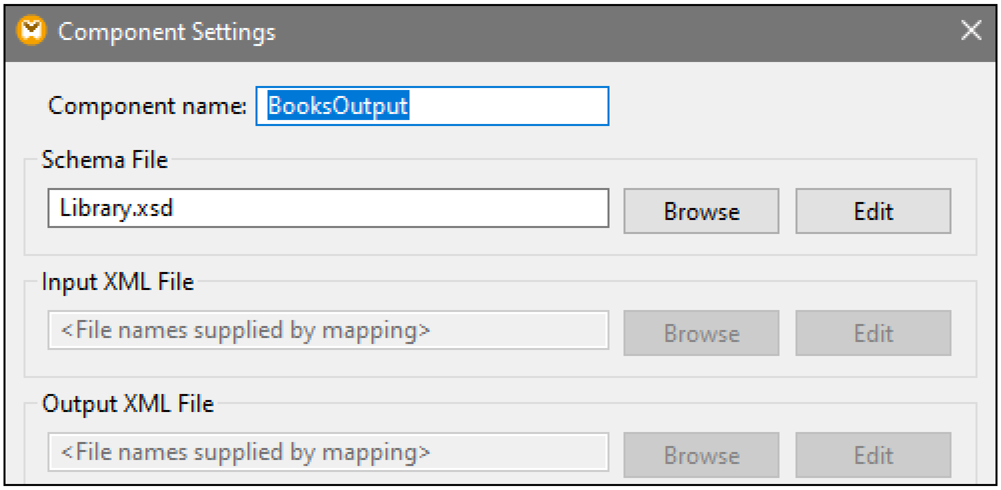
Step 2: Connect concat function and dynamic node

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



Step 3: Check component settings

In the Component Settings, you will notice that the *Input XML File* and *Output XML File* text boxes are disabled and show <*File names supplied by the mapping*> (*screenshot below*). This indicates that you have supplied the instance file names dynamically from the mapping. Therefore, it is no longer possible to define them in the component settings.



Component Settings

Component name: BooksOutput

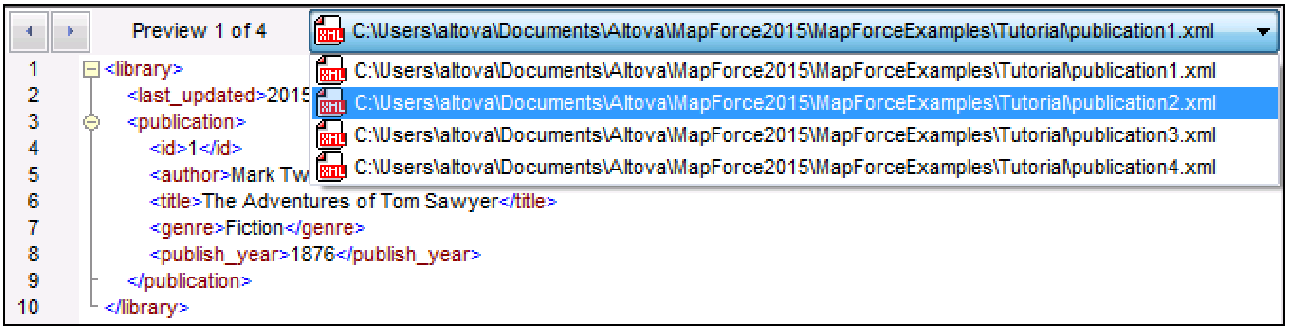
Schema File: Library.xsd

Input XML File: <File names supplied by mapping>

Output XML File: <File names supplied by mapping>

Step 4: Generate output files

You can now run the mapping and see the result as well as the names of the generated 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 clicking a file from the adjacent drop-down list (*screenshot below*).



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>
```

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

4 Structural Components

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

- XML and XML Schema 132
- Databases 165
- CSV and Text Files 323

MapForce also enables you to map data to and from binary files (BLOB data). For more information, see the `lang | file` 627 functions.

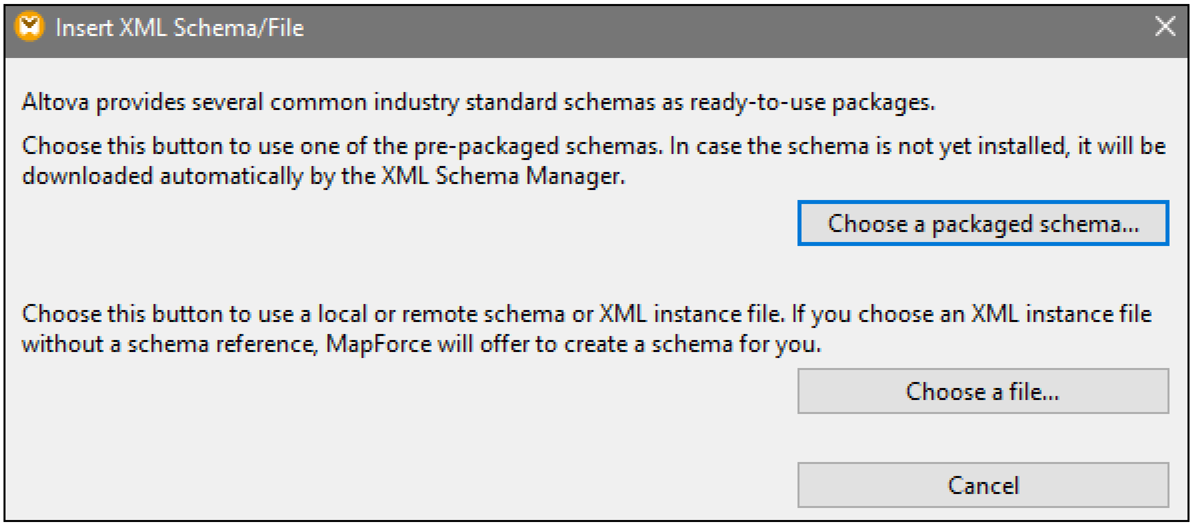
4.1 XML and XML Schema

Altova website: XML Mapping

XML is a markup language for text documents. XML Schema defines the structure and constraints of XML documents. In MapForce, XML files are structural components [34] that can be used as data sources and targets. For information about basic data transformation scenarios, see Tutorials [100].

Insert XML schema/file

To insert an XML schema/file, select the menu command **Insert | XML Schema/File** or the toolbar button. The dialog box (*see screenshot below*) will prompt you to choose between a packaged industry-standard schema and a local or remote schema/instance file. If you choose a packaged schema, you will be prompted to select an entry point. If the schema you wish to use is not yet installed, you will be redirected to the XML Schema Manager [149] to download it.



Insert XML Schema/File

Altova provides several common industry standard schemas as ready-to-use packages.

Choose this button to use one of the pre-packaged schemas. In case the schema is not yet installed, it will be downloaded automatically by the XML Schema Manager.

Choose this button to use a local or remote schema or XML instance file. If you choose an XML instance file without a schema reference, MapForce will offer to create a schema for you.

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

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

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

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

StyleVision). To make such DTDs usable in MapForce, define the `xmlns` attribute with the namespace URI as follows:

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

Note about enumeration values

For nodes whose data types have enumeration facets, you can create a Value-Map that will have all enumeration values pre-filled. This makes it easier to process and map enumeration values. For more information, see Value-Maps 424.

In this section

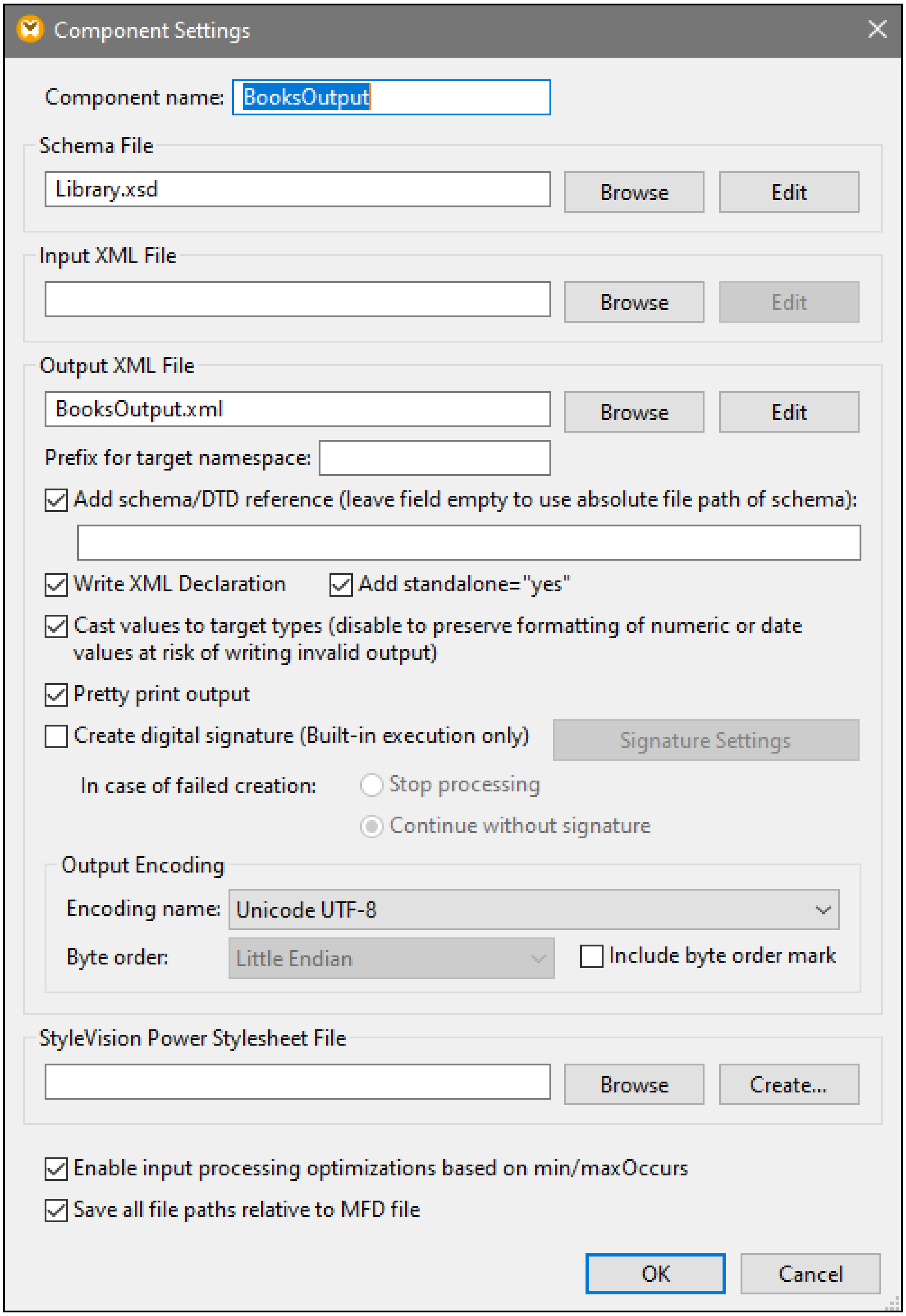
The section is organized into the following topics:

- XML Component Settings 133
- Derived Types 137
- NULL Values 139
- Comments and Processing Instructions 143
- CDATA Sections 143
- Wildcards: xs:any/xs:anyAttribute 145
- Custom Namespaces 147
- XML Schema Manager 149

4.1.1 XML Component Settings

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

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



Component Settings

Component name: BooksOutput

Schema File

Library.xsd

Input XML File

Output XML File

BooksOutput.xml

Prefix for target namespace:

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

☑ Write XML Declaration ☑ Add standalone="yes"

☑ 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

The available settings are described in the subsections below.

General settings

⊟ Component name

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

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

⊟ Schema File

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

⊟ Input XML File

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

⊟ Output XML File

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

⊟ Prefix for target namespace

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

⊟ Add schema/DTD reference

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

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

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

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

⊟ Write XML declaration

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

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

⊟ Add standalone="yes"

Selecting this option inserts the `standalone="yes"` declaration into the XML declaration of your target XML file. For more information, see *Standalone Document Declaration*.

Note the following points:

- When the *standalone="yes"* option is selected, output generation is compatible with XSLT 1-3, Built-In, and generated code (C#, Java, C++ MSXML, C+ Xerces). The Built-In[19] transformation language and generated code are available in Professional and Enterprise editions. For more information about code generation, see Code Generator[868].
- There is no support for XML embedded in database fields and web service requests (*Professional and Enterprise editions*).

⊟ Cast values to target types

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

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

⊟ Pretty print output

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

⊟ Create digital signature (*Enterprise Edition*)

Allows you to add a digital signature to the XML output instance file. Adding a digital signature is possible when you select Built-In as a transformation language.

Output Encoding

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

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

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

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

StyleVision Power Stylesheet file

This option allows you to select or create an Altova StyleVision stylesheet file. Such files enable you to output data from the XML instance file to a variety of formats suitable for reporting, such as HTML, RTF, and others. See also Using Relative Paths on a Component 43.

Other settings

⊟ Enable input processing optimizations based on min/maxOccurs

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

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

⊟ Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed on the **Component Settings** dialog box relative to the location of the MapForce Design (**`.mfd`**) file. See also Relative and Absolute Paths 43.

4.1.2 Derived Types

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

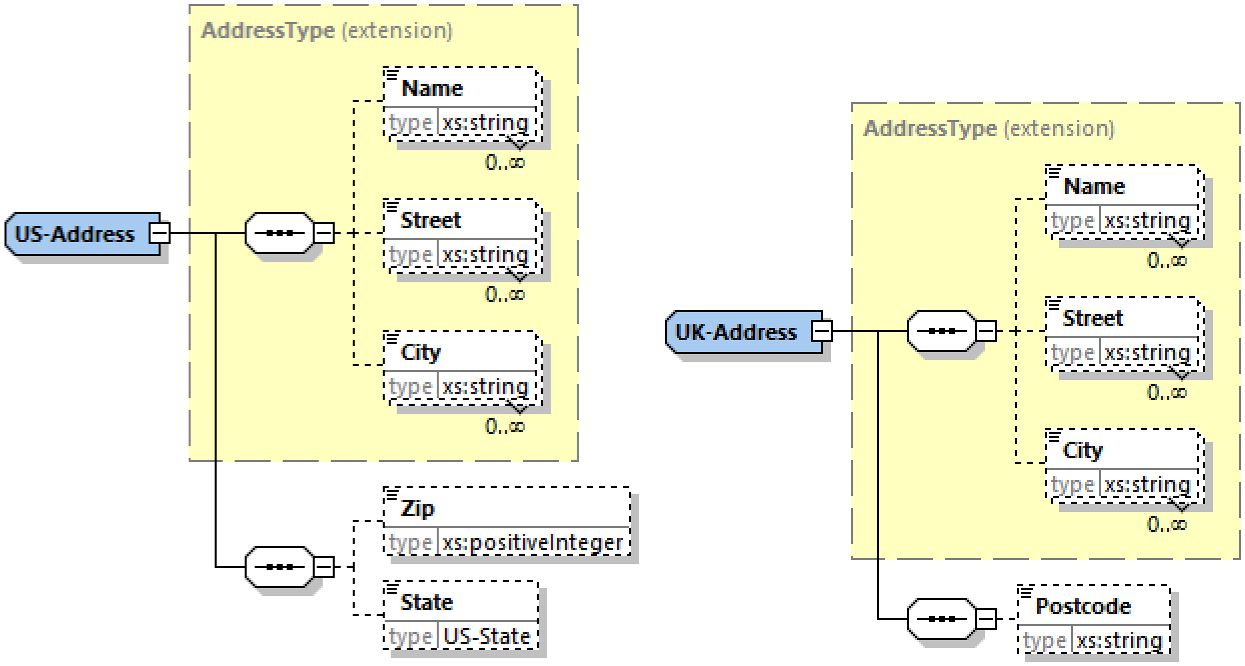
Possible scenario

This subsection describes a possible scenario of using a derived type. For example, we have a company with two branches: one in the UK and the other in the US. Now we would like to have two lists (`UKCustomers` and

USCustomers), each of which will include information about the respective branch address and all the customers associated with this branch.

Definition of derived types

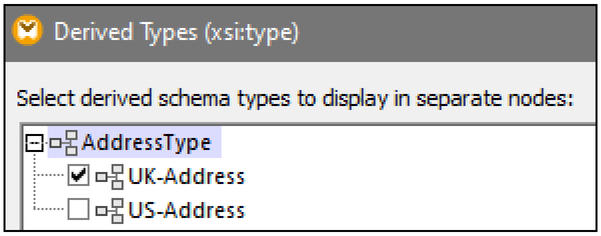
The screenshots below illustrate the definition of derived types called US-Address and UK-Address (*XMLSpy Schema view*). The UK-Address and UK-Address elements have the same base type called AddressType that includes the Name, Street, and City elements. In the US-Address element, the base type has been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



Derived type in a mapping

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

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

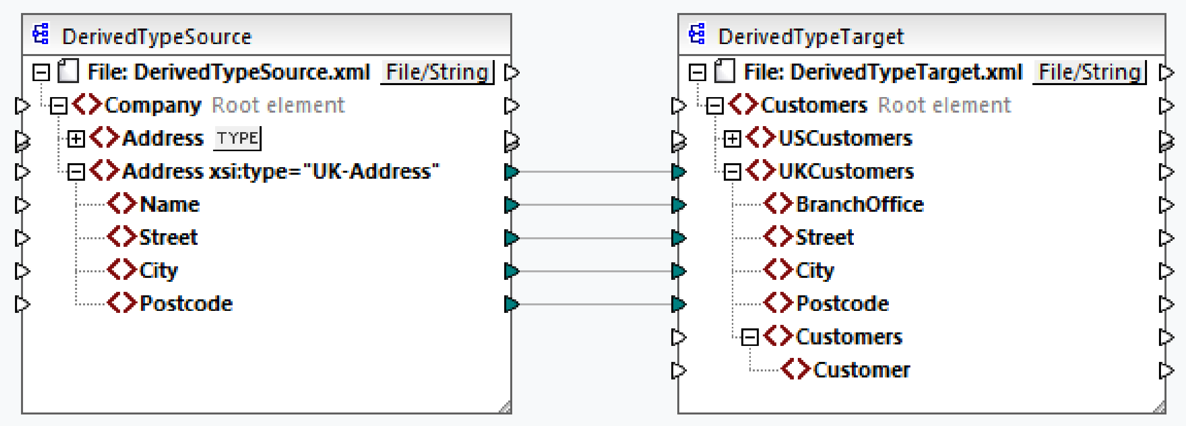


Derived Types (xsi:type)

Select derived schema types to display in separate nodes:

- AddressType
  - ☑ UK-Address
  - □ US-Address

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



Output

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

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

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

4.1.3 NULL Values

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

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

NULL values in XML components

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

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

This scenario has the following conditions:

- The connection is target-driven 51.
- The source element has an `xsi:nil="true"` attribute. The corresponding target element does not have this attribute.
- Alternatively, both source and target elements can have `xsi:nil="true"` attributes.
- The `nillable="true"` attributes must be set in the source and target schemas.
- The source and target element are of simple type.

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

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

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

Only the target element has xsi:nil="true"

This scenario has the following conditions:

- The connection is target-driven 51.
- The source element does not have an `xsi:nil="true"` attribute.
- The corresponding target element has an `xsi:nil="true"` attribute.
- The source and target elements can be of simple or complex type.

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

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

Complex-type source element/both complex-type elements have xsi:nil="true"

This scenario has the following conditions:

- The connection is target-driven[51].
- The source element is of complex type. In our example, the source element has an `id="213"` attribute and an `xsi:nil="true"` attribute. The corresponding target element is also of complex type and has an `id="124"` attribute, but does not have an `xsi:nil="true"` attribute.
- Alternatively, the source and target elements, both of which are of complex type, can have `xsi:nil="true"` attributes.

In this case, the source element will overwrite the target element (*highlighted in yellow below*). However, the `xsi:nil="true"` attribute will not be written to the output file automatically. To see the `xsi:nil="true"` attribute in the target element in the output file, use a copy-all[56] connection.

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

Useful functions

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

- **`is-xsi-nil`**[555]: Helps to check explicitly whether a source element has a `xsi:nil` attribute set to `true`.
- **`substitute-missing`**[589]: Substitutes a NULL value in the source element with something specific.
- **`set-xsi-nil`**[557]: Assigns `xsi:nil="true"` attribute to a target element. This works for target elements of simple and complex types.
- **`substitute-missing-with-xsi-nil`**[559]: If there is content, it will be written to the target element; if there are any missing values, using this function will result in a target element with a `xsi:nil="true"` attribute in the output.
- Connecting the **`exists`**[563] function to a source element with a NULL value returns `true` even though the element has no content.

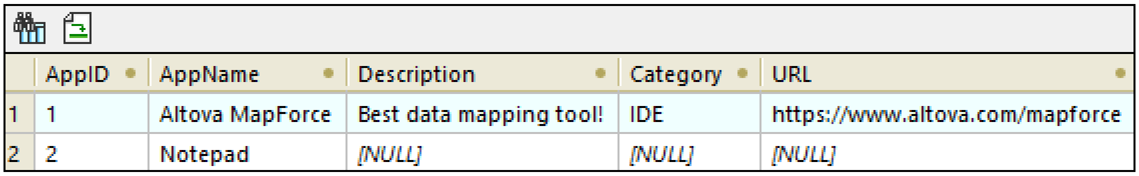
Note that functions which generate `xsi:nil` cannot be passed through functions or components which only operate on values (such as the **`if-else`** function).

NULL values in database components

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

Mapping NULL database fields to NULL elements
Target elements that receive NULL values from database fields are not created in the output automatically. To see such elements in the output, you need to (i) add `nillable="true"` attributes to the relevant target elements in the schema file and (ii) use the **`substitute-missing-with-xsi-nil`**[559] function in the mapping. The example below shows how to handle NULL values in mappings with a source database component.

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



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

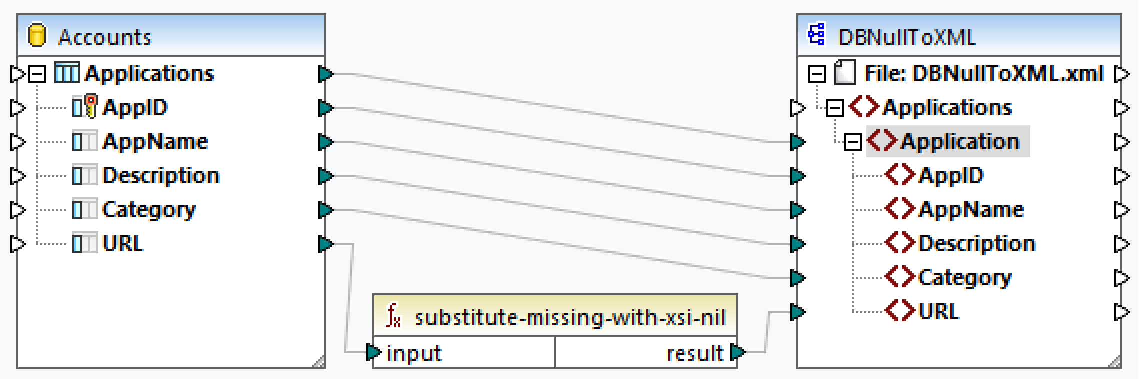
To see the `Application` table, take the steps below:

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

To find out more about querying databases, see DB Query Pane 277.

Mapping

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



Output

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

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

```
    <URL xsi:nil="true"/>
</Application>
```

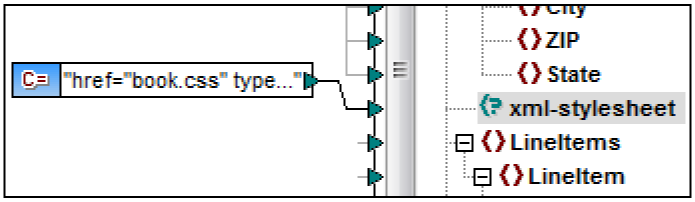
Mapping NULL elements to NULL database fields
When you map a NULL XML element to a database column, MapForce writes the NULL value to the corresponding database column. You can also use the **`set-null`** 607 function if you want to set a database field to NULL. To find out more about database-related functions, see the DB library 606.

4.1.4 Comments and Processing Instructions

This topic explains how to insert comments and processing instructions into target XML components. Note that comment and processing instruction nodes have only input connections. Comments and processing instructions cannot be defined for nodes that are part of a copy-all connection 56. Comments and processing instructions are defined in the W3C Specification.

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

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



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

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

Note: Only one comment can be added before and after a single target node. To create multiple comments, use the duplicate input function 42.

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

4.1.5 CDATA Sections

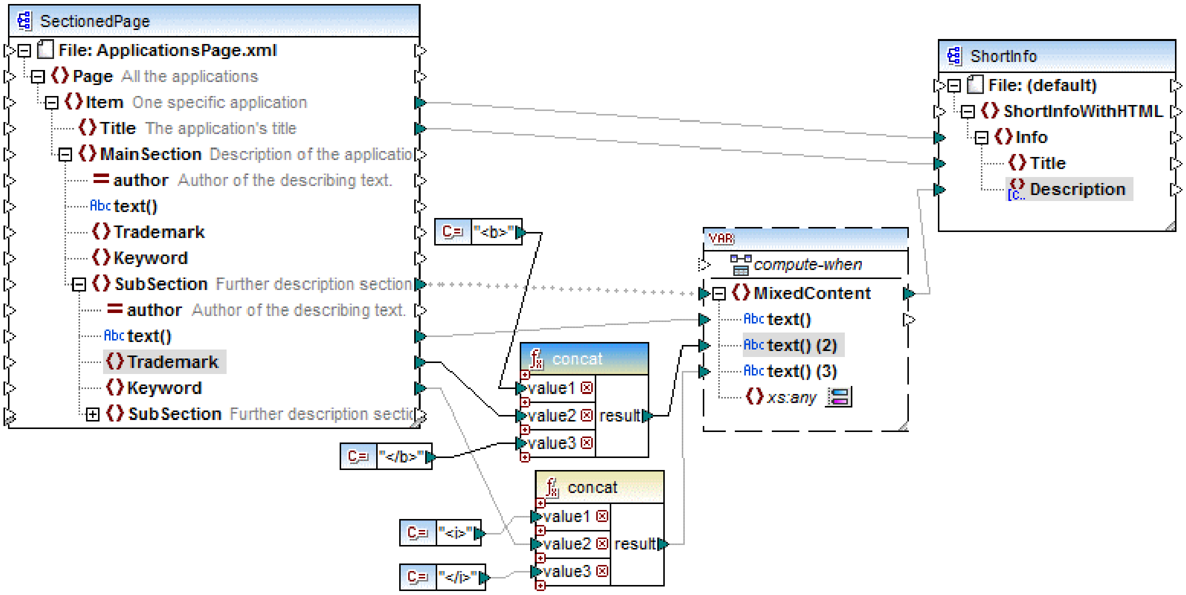
CDATA sections are used to represent parts of a document as character data which would normally be interpreted as markup. For more information about CDATA sections, see the W3C Specification. Target nodes receiving data as CDATA sections can be any of the following: XML data, XML data embedded in database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

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

Example

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

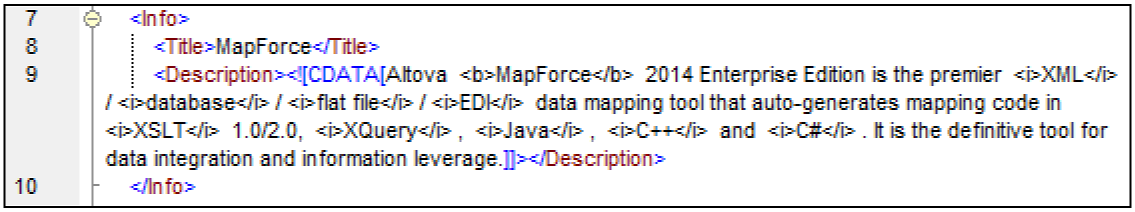
- The `SubSection` element has mixed content. For more information about mixed-content nodes, see Source-Driven Connections 52.
- With the help of the **concat** function, the content of the `Trademark` element will have the `<b></b>` tags.
- The content of the `Keyword` element will have the `<i></i>` tags.
- The data with the new tags is passed on to the duplicate `text()` nodes in the same order as in the source document.
- The output of the `MixedContent` node is then passed on to the `Description` node in the `ShortInfo` target component. The `Description` node has been defined as a CDATA section.



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>"
concat
value1
value2
value3
result
concat
value1
value2
value3
result
"<i>"
"</i>"
VAR
compute-when
MixedContent
text()
text() (2)
text() (3)
xs:any
ShortInfo
File: (default)
ShortInfoWithHTML
Info
Title
Description

Output

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



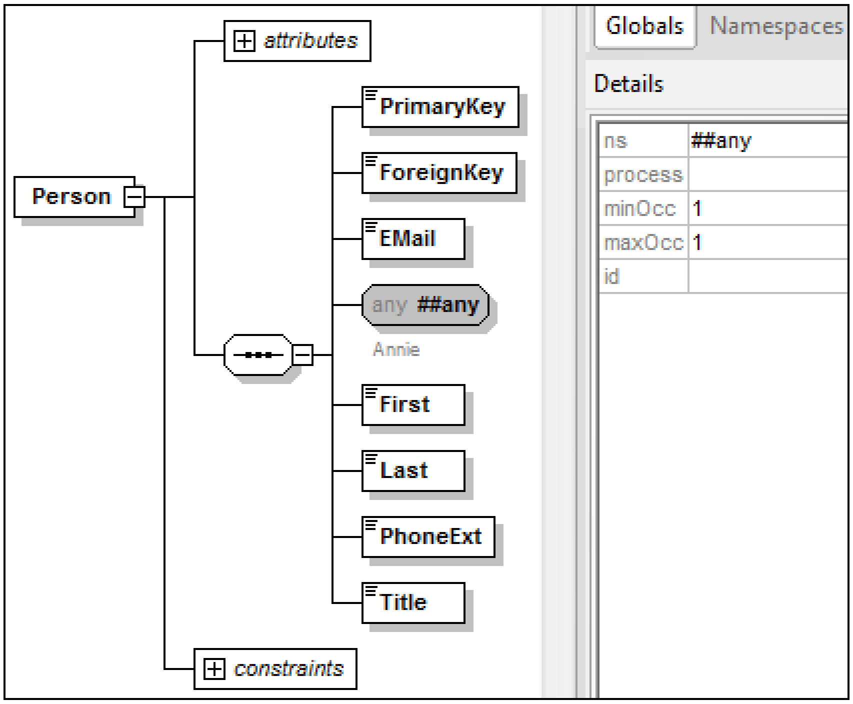
```
<Info>
    <Title>MapForce</Title>
    <Description><![CDATA[Altova <b>MapForce</b> 2014 Enterprise Edition is the premier <i>XML</i> / <i>database</i> / <i>flat file</i> / <i>EDI</i> data mapping tool that auto-generates mapping code in <i>XSLT</i> 1.0/2.0, <i>XQuery</i> , <i>Java</i> , <i>C++</i>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

4.1.6 Wildcards: xs:any/xs:anyAttribute

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

Wildcards in schema definition

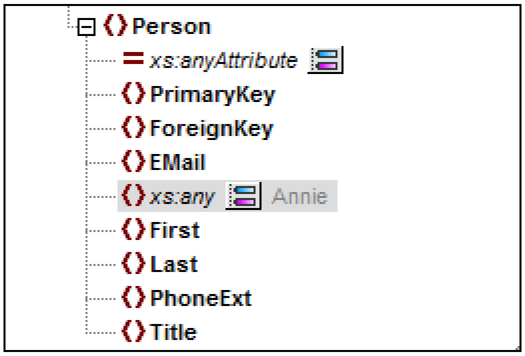
The screenshot below shows that an `xs:any` element has been defined as a child element of the `Person` element (*Schema view in Altova XMLSpy*).



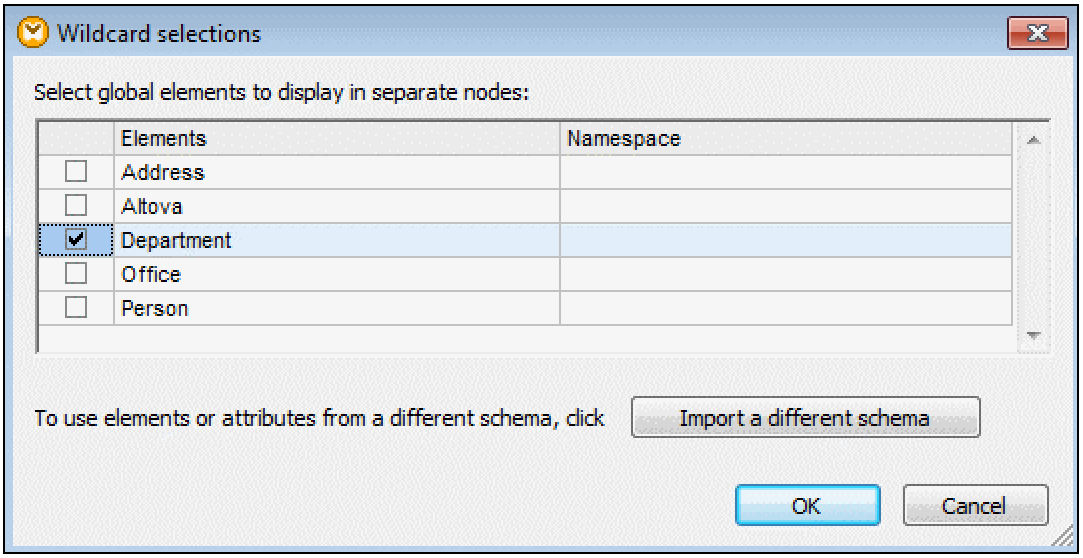
ns	##any
process	
minOcc	1
maxOcc	1
id	

Wildcards in MapForce

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



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



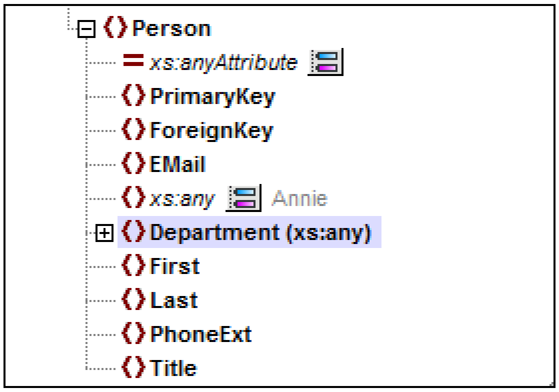
Wildcard selections

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

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



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

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

Elements/attributes from a different schema

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

Import schema

The **Import schema** option imports the external schema into the current schema assigned to the component. Note that this option overrides the existing schema on the disk. If the current schema is a remote schema that has been opened from a URL (see Adding Components from a URL 38) and not from the disk, the schema cannot be modified. In this case, use the **Generate wrapper schema** option.

Generate wrapper schema

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

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

Wildcards vs. dynamic node names

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

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

For such situations, it is recommended to use dynamic node names instead of wildcards. For more information, see Mapping Node Names 726.

4.1.7 Custom Namespaces

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

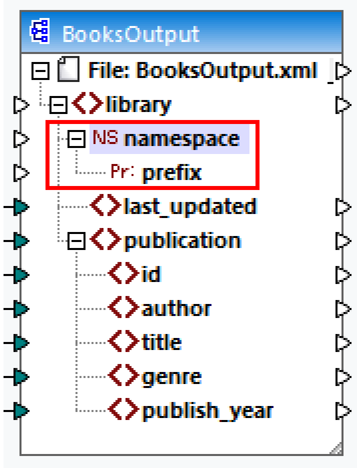
Declaring custom namespaces is meaningful only for target XML components and applies to elements only. The **Add Namespace** command is not available for attributes, wildcard nodes, and for nodes which receive data from a copy-all connection 56.

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

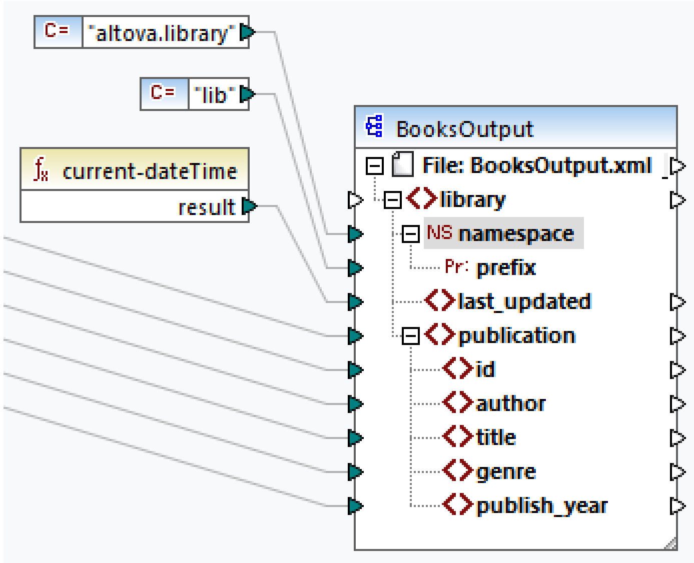
Declare namespace manually

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

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



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



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

Output

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

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

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

Declare a default namespace

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

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

If you need to create prefixes for attribute names, for example `<number prod:id="prod557">557</number>`, you can achieve this by using dynamic access to a node's attributes (see Mapping Node Names 726) or by editing the schema so that it has a `prod:id` attribute for `<number>`.

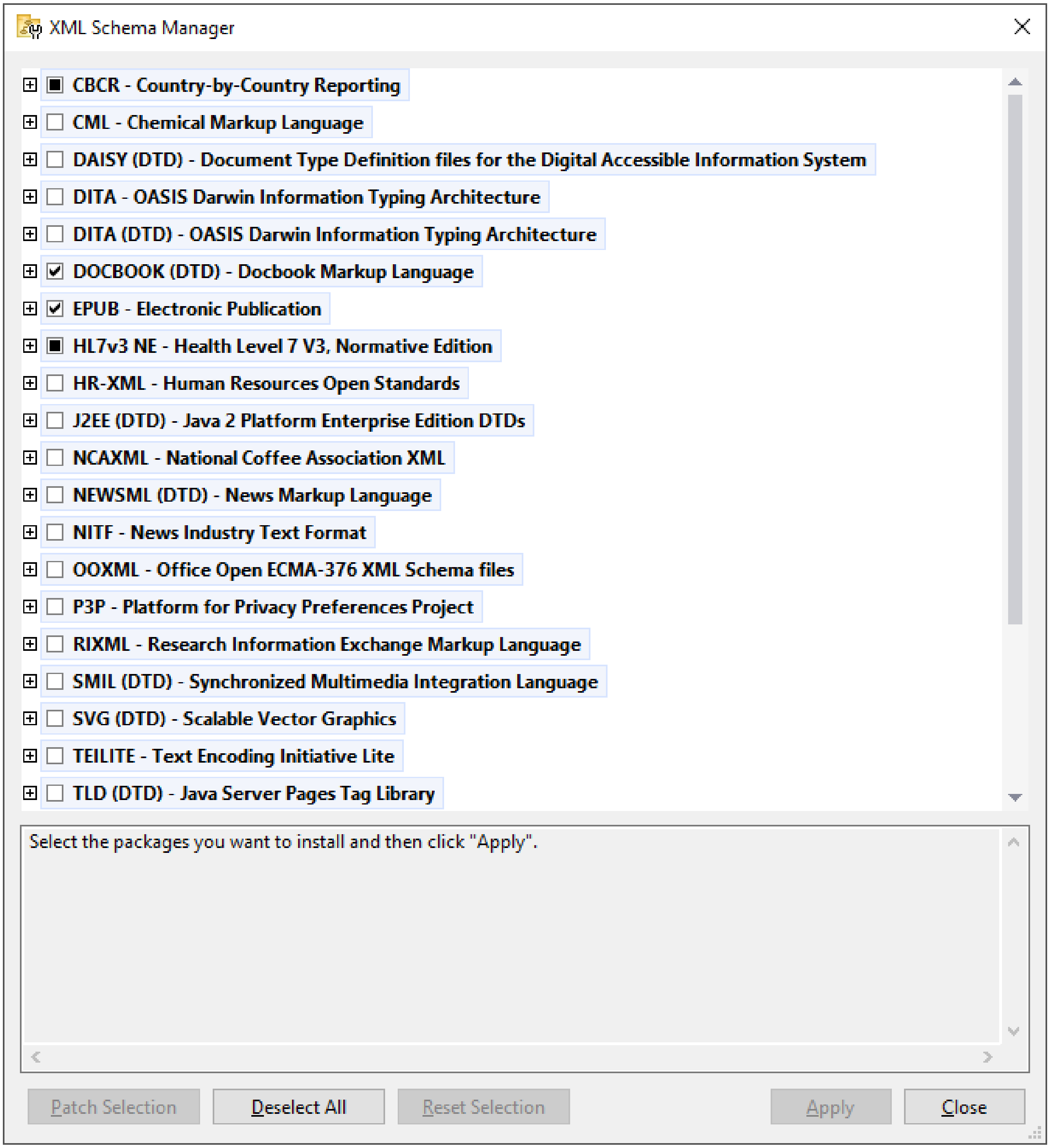
Remove a namespace

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

4.1.8 Schema Manager

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

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



XML Schema Manager
⊞ ■ CBCR - Country-by-Country Reporting
⊞ □ CML - Chemical Markup Language
⊞ □ DAISY (DTD) - Document Type Definition files for the Digital Accessible Information System
⊞ □ DITA - OASIS Darwin Information Typing Architecture
⊞ □ DITA (DTD) - OASIS Darwin Information Typing Architecture
⊞ ☑ DOCBOOK (DTD) - Docbook Markup Language
⊞ ☑ EPUB - Electronic Publication
⊞ ■ HL7v3 NE - Health Level 7 V3, Normative Edition
⊞ □ HR-XML - Human Resources Open Standards
⊞ □ J2EE (DTD) - Java 2 Platform Enterprise Edition DTDs
⊞ □ NCAXML - National Coffee Association XML
⊞ □ NEWSML (DTD) - News Markup Language
⊞ □ NITF - News Industry Text Format
⊞ □ OOXML - Office Open ECMA-376 XML Schema files
⊞ □ P3P - Platform for Privacy Preferences Project
⊞ □ RIXML - Research Information Exchange Markup Language
⊞ □ SMIL (DTD) - Synchronized Multimedia Integration Language
⊞ □ SVG (DTD) - Scalable Vector Graphics
⊞ □ TEILITE - Text Encoding Initiative Lite
⊞ □ TLD (DTD) - Java Server Pages Tag Library
Patch Selection
Deselect All
Reset Selection
Apply
Close

Select the packages you want to install and then click "Apply".

Altova applications that operate with Schema Manager

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

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

Installation and de-installation of Schema Manager

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

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

Schema Manager features

Schema Manager provides the following features:

- Shows XML schemas installed on your computer and checks whether new versions are available for download.
- Downloads newer versions of XML schemas independently of the Altova product release cycle. (Altova stores schemas online, and you can download them via Schema Manager.)
- Install or uninstall any of the multiple versions of a given schema (or all versions if necessary).
- An XML schema may have dependencies on other schemas. When you install or uninstall a particular schema, Schema Manager informs you about dependent schemas and will automatically install or remove them as well.
- Schema Manager uses the XML catalog mechanism to map schema references to local files. In the case of large XML schemas, processing will therefore be faster than if the schemas were at a remote location.
- All major schemas are available via Schema Manager and are regularly updated for the latest versions. This provides you with a convenient single resource for managing all your schemas and making them readily available to all of Altova's XML-schema-aware applications.
- Changes made in Schema Manager take effect for all Altova products installed on that machine.
- In an Altova product, if you attempt to validate on a schema that is not installed but which is available via Schema Manager, then installation is triggered automatically. However, if the schema package contains namespace mappings, then there will be no automatic installation; in this case, you must start Schema Manager, select the package/s you want to install, and run the installation. If, after installation, your open Altova application does not restart automatically, then you must restart it manually.

How it works

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

Schema Manager also installs schemas in one other way. At the Altova website (https://www.altova.com/schema-manager) you can select a schema and its dependent schemas that you want to install. The website will prepare a file of type **`.altova_xmlschemas`** for download that contains information about your schema selection. When you double-click this file or pass it to Schema Manager via the CLI as an argument of the `install` 160 command, Schema Manager will install the schemas you selected.

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

Windows	`C:\ProgramData\Altova\pkgs\.cache`
Linux	`/var/opt/Altova/pkgs\.cache`
macOS	`/var/Altova/pkgs`

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

- Every time you start Schema Manager.
- When you start MapForce for the first time on a given calendar day.
- If MapForce is open for more than 24 hours, the cache is updated every 24 hours.
- You can also update the cache by running the `update` 163 command at the command line interface.

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

> **Do not modify the cache manually!**
> The local cache directory is maintained automatically based on the schemas you install and uninstall. It should not be altered or deleted manually. If you ever need to reset Schema Manager to its original "pristine" state, then, on the command line interface (CLI): (i) run the `reset` 161 command, and (ii) run the `initialize` 159 command. (Alternatively, run the **`reset`** command with the **`--i`** option.)

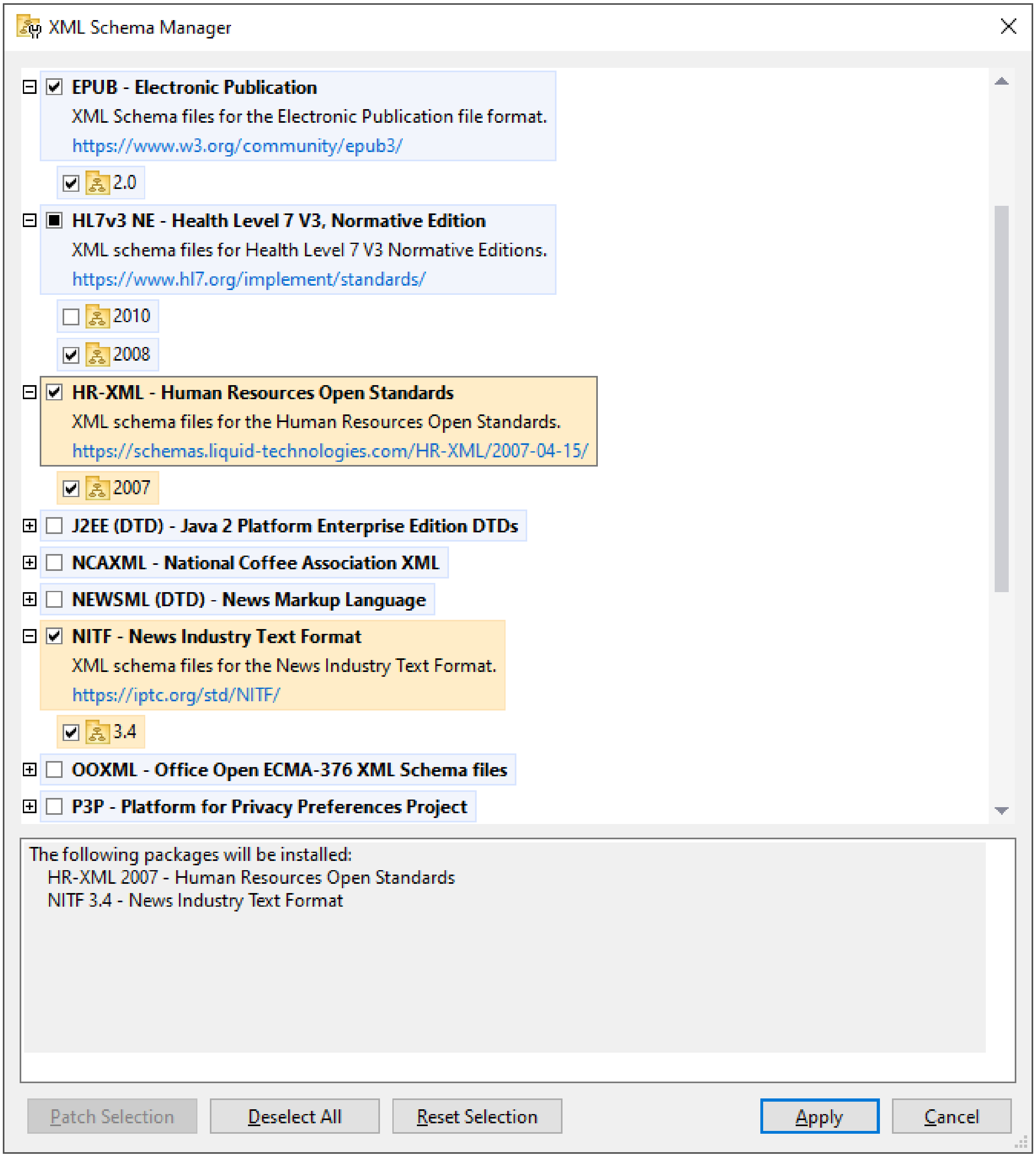
4.1.8.1 Run Schema Manager

Graphical User Interface

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

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

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



⊟ ☑ **EPUB - Electronic Publication**
XML Schema files for the Electronic Publication file format.
https://www.w3.org/community/epub3/
☑ 2.0

⊟ ■ **HL7v3 NE - Health Level 7 V3, Normative Edition**
XML schema files for Health Level 7 V3 Normative Editions.
https://www.hl7.org/implement/standards/
☐ 2010
☑ 2008

⊟ ☑ **HR-XML - Human Resources Open Standards**
XML schema files for the Human Resources Open Standards.
https://schemas.liquid-technologies.com/HR-XML/2007-04-15/
☑ 2007

⊞ ☐ **J2EE (DTD) - Java 2 Platform Enterprise Edition DTDs**

⊞ ☐ **NCAXML - National Coffee Association XML**

⊞ ☐ **NEWSML (DTD) - News Markup Language**

⊟ ☑ **NITF - News Industry Text Format**
XML schema files for the News Industry Text Format.
https://iptc.org/std/NITF/
☑ 3.4

⊞ ☐ **OOXML - Office Open ECMA-376 XML Schema files**

⊞ ☐ **P3P - Platform for Privacy Preferences Project**

The following packages will be installed:
HR-XML 2007 - Human Resources Open Standards
NITF 3.4 - News Industry Text Format

When you click **Apply**, the progress of the installation is displayed. If there is an error (for example, a connection error), then an error message is displayed. In this case, click the **Advanced** button that appears in the dialog, check the schema selection and retry with **Apply**.

Command line interface

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

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

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

You can then use any of the commands listed in the CLI command reference section 158.

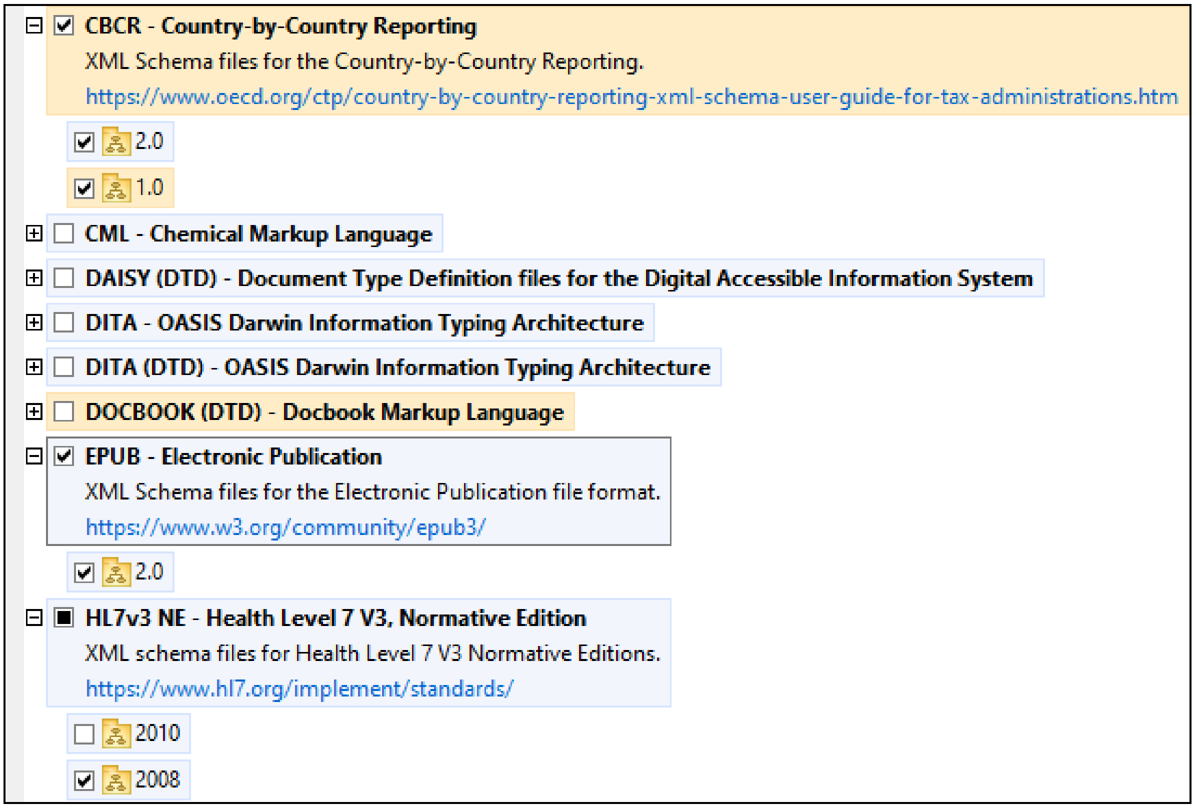
To display help for the commands, run the following:

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

4.1.8.2 Status Categories

Schema Manager categorizes the schemas under its management as follows:

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


⊟ ☑ CBCR - Country-by-Country Reporting
XML Schema files for the Country-by-Country Reporting.
https://www.oecd.org/ctp/country-by-country-reporting-xml-schema-user-guide-for-tax-administrations.htm
☑ 2.0
☑ 1.0
⊞ □ CML - Chemical Markup Language
⊞ □ DAISY (DTD) - Document Type Definition files for the Digital Accessible Information System
⊞ □ DITA - OASIS Darwin Information Typing Architecture
⊞ □ DITA (DTD) - OASIS Darwin Information Typing Architecture
⊞ □ DOCBOOK (DTD) - Docbook Markup Language
⊟ ☑ EPUB - Electronic Publication
XML Schema files for the Electronic Publication file format.
https://www.w3.org/community/epub3/
☑ 2.0
⊟ ■ HL7v3 NE - Health Level 7 V3, Normative Edition
XML schema files for Health Level 7 V3 Normative Editions.
https://www.hl7.org/implement/standards/
□ 2010
☑ 2008

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

Points to note

- In the screenshot above, both CBCR schemas are checked. The one with the blue background is already installed. The one with the yellow background is uninstalled and has been selected for installation. Note that the HL7v3 NE 2010 schema is not installed and has not been selected for installation.
- A yellow background means that the schema will be modified in some way when the **Apply** button is clicked. If a schema is unchecked and has a yellow background, it means that it will be uninstalled when the **Apply** button is clicked. In the screenshot above the DocBook DTD has such a status.
- When running Schema Manager from the command line, the `list` 160 command is used with different options to list different categories of schemas:

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

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

4.1.8.3 Patch or Install a Schema

Patch an installed schema

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

In the GUI

Patches are indicated by the icon. (*Also see the previous topic about status categories*[154].) If patches are available, the **Patch Selection** button will be enabled. Click it to select and prepare all patches for installation. In the GUI, the icon of each schema that will be patched changes from to , and the Messages pane at the bottom of the dialog lists the patches that will be applied. When you are ready to install the selected patches, click **Apply**. All patches will be applied together. Note that if you deselect a schema marked for patching, you will actually be uninstalling that schema.

On the CLI

To apply a patch at the command line interface:

1. Run the `list -u`[160] command. This lists any schemas for which upgrades are available.
2. Run the `upgrade`[163] command to install all the patches.

Install an available schema

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

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

In the GUI

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

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

On the CLI

To install schemas via the command line, run the `install`[160] command:

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

where **`Schema`** is the schema (or schemas) you want to install or a **`.altova_xmlschemas`** file. A schema is referenced by an identifier of format **`<name>-<version>`**. (The identifiers of schemas are displayed when you run the `list`[160] command.) You can enter as many schemas as you like. For details, see the description of the `install`[160] command.

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

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

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

4.1.8.4 Uninstall a Schema, Reset

Uninstall a schema

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

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

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

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

On the CLI
To uninstall schemas via the command line, run the `uninstall`[162] command:

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

where each **`Schema`** argument is a schema you want to uninstall or a **`.altova_xmlschemas`** file. A schema is specified by an identifier that has a format of **`<name>-<version>`**. (The identifiers of schemas are displayed when you run the `list`[160] command.) You can enter as many schemas as you like. For details, see the description of the `uninstall`[162] command.

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

Reset Schema Manager

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

- In the GUI, click **Reset Selection**.
- On the CLI, run the `reset`[161] command.

After running this command, make sure to run the `initialize` 159 command in order to recreate the cache directory. Alternatively, run the `reset` 161 command with the `-i` option.

Note that `reset -i` 161 restores the original installation of the product, so it is recommended to run the `update` 163 command after performing a reset. Alternatively, run the `reset` 161 command with the `-i` and `-u` options.

4.1.8.5 Command Line Interface (CLI)

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

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

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

Command line syntax

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

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

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

4.1.8.5.1 help

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

Syntax

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

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

Note the following:

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

Example

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

```
xmlschemamanager help list
```

4.1.8.5.2 info

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

Syntax

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

- The **`Schema`** argument is the name of a schema or a part of a schema's name. (To display a schema's package ID and detailed information about its installation status, you should use the `list` 160 command.)
- Use **`<exec> info -h`** to display help for the command.

Example

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

```
xmlschemamanager info doc nitf
```

4.1.8.5.3 initialize

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

Syntax

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

Options

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

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

Example

The following command initializes Schema Manager:

```
xmlschemamanager initialize
```

4.1.8.5.4 install

This command installs one or more schemas.

Syntax

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

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

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

- A schema identifier (having a format of **<name>-<version>**, for example: **cbcr-2.0**). To find out the schema identifiers of the schemas you want, run the list 160 command. You can also use an abbreviated identifier if it is unique, for example **docbook**. If you use an abbreviated identifier, then the latest version of that schema will be installed.
- The path to a **.altova_xmlschemas** file downloaded from the Altova website. For information about these files, see *Introduction to SchemaManager: How It Works* 149.

Options

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

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

Example

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

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

4.1.8.5.5 list

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

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

Syntax

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

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

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

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

Examples

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

4.1.8.5.6 reset

This command removes all installed schemas and the cache directory. You will be completely resetting your schema environment. After running this command, be sure to run the `initialize` 159 command to recreate the cache directory. Alternatively, run the **`reset`** command with the **`-i`** option. Since **`reset -i`** restores the original installation of the product, we recommend that you run the `update` 163 command after performing a reset and initialization. Alternatively, run the **`reset`** command with both the **`-i`** and **`-u`** options.

Syntax

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

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

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

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

Examples

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

4.1.8.5.7 uninstall

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

Syntax

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

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

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

- A schema identifier (having a format of **`<name>-<version>`**, for example: **`cbcr-2.0`**). To find out the schema identifiers of the schemas that are installed, run the `list -i` 160 command. You can also use an abbreviated schema name if it is unique, for example **`docbook`**. If you use an abbreviated name, then all schemas that contain the abbreviation in its name will be uninstalled.
- The path to a **`.altova_xmlschemas`** file downloaded from the Altova website. For information about these files, see *Introduction to SchemaManager: How It Works* 149.

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

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

Example

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

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

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

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

4.1.8.5.8 update

This command queries the list of schemas available from the online storage and updates the local cache directory. You should not need to run this command unless you have performed a `reset` 161 and `initialize` 159.

Syntax

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

Options

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

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

Example

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

```
xmlschemamanager update
```

4.1.8.5.9 upgrade

This command upgrades all installed schemas that can be upgraded to the latest available *patched* version. You can identify upgradeable schemas by running the `list -u` 160 command.

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

Syntax

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

Options

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

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

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

4.2 Databases

Altova website: Database mapping

MapForce enables you to use databases as data sources and targets.

The table below lists all the supported databases. If your Altova application is a 64-bit version, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

Database	Notes
Firebird 2.x, 3.x, 4.x	
IBM DB2 8.x, 9.x, 10.x, 11.x, 12.x	
IBM Db2 for i 6.x, 7.4, 7.5	Logical files are supported and shown as views.
IBM Informix 11.70 and later	Informix supports connections via ADO, JDBC and ODBC. The implementation does not support large object data types in any of the code generation languages. MapForce will generate an error message (during code generation) if any of these data types are used.
MariaDB 10 and later	MariaDB supports native connections. No separate drivers are required.
Microsoft Access 2003 and later	You can connect to an Access 2019 database from Altova products (i) only if the corresponding version of Microsoft Access Runtime is installed and (ii) only if the database does not use the "Large Number" data type.
Microsoft Azure SQL Database	SQL Server 2016 codebase
Microsoft SQL Server 2005 and later Microsoft SQL Server on Linux	
MySQL 5 and later	MySQL 5.7 and later supports native connections. No separate drivers are required.
Oracle 9i and later	
PostgreSQL 8 and later	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	Notes
Progress OpenEdge 11.6	
SQLite 3.x	SQLite connections are supported as native, direct connections to the SQLite database file. No separate drivers are required.
Sybase ASE 15, 16	
Teradata 16	Connections are supported through ADO.NET, JDBC, and ODBC. When a mapping inserts data into a database table, database-generated identity fields are not supported.

Database mappings in various execution environments

When you generate program code [66] from a mapping, compile a mapping to MapForce Server execution files [799], or deploy a mapping to FlowForce Server [802], the database connection details saved with the generated files are adapted to drivers supported for the chosen target environment (*see 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 (e.g., you may need to check if the database driver is installed, the database path is correct, the database server is accessible, etc.).

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

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

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

Table legend:

- *As is* means that the database connection type (e.g., JDBC) remains as defined in MapForce.
- *Converted to JDBC* means that the database connection will be converted to a JDBC-like database connection URL.
- *ADO bridge* and *ODBC bridge* mean 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 (e.g., `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 235 dialog box.

Preventing possible issues with JDBC connections in Java environment

If the mapping connects to a database through JDBC, ensure that the JDBC driver used by the mapping is installed on your system. To view the current JDBC settings of any database component in MapForce, double-click the database component's header. This will open the **Component Settings** dialog. For more information, see Database Component Settings 235 and Creating a JDBC connection 187.

If the mapping uses a non-JDBC database connection, the connection may be converted to JDBC during Java code generation, to provide compatibility in a Java environment. For details, see the table above.

If you run the generated Java application, the JDBC driver may need to be added as a classpath entry to the current configuration. Otherwise, running the application could result in an error similar to the following: `java.lang.ClassNotFoundException: com.mysql.jdbc.Driver`.

This subsection discusses some of the possible approaches to fixing JDBC-related issues.

Approach 1: Add JDBC driver as a dependency in Eclipse

If you work with Eclipse, you will need to add a JDBC driver as a dependency as follows:

1. Generate Java code in MapForce and import the project into Eclipse, as described in Generate, Build, and Run Code 872.
2. Click the desired configuration in Eclipse (e.g., `MappingApplication`).
3. In the *Dependencies* tab, click **Classpath entries** and then click **Add External JARs**.
4. Browse for the **`.jar`** file of your JDBC driver (e.g., **`C:\jdbc\mysql\mysql-connector-java-5.1.16-bin.jar`**).
5. Click **Run** to run the program with the database JDBC driver added as a dependency.

Approach 2: Add JDBC driver to classpath of test task in build.xml

If you get an error related to the JDBC driver when you run the Ant **`build.xml`** file, add the JDBC driver to the classpath of the `test` task in **`build.xml`**. The code listing below is an example of an Ant `test` task that includes a reference to the **`.jar`** file of the JDBC driver (*highlighted in yellow below*).

```
<target name="test" depends="compile">
   <java classpath="C:\codegen\java\mysql_mapping"
classname="com.mapforce.MappingConsole" fork="true" failonerror="true">
      <classpath>
         <pathelement path="${classpath}"/>
         <pathelement location="C:\jdbc\mysql\mysql-connector-java-5.1.16-bin.jar"/>
      </classpath>
      <arg line="${cmdline}"/>
   </java>
</target>
```

Approach 3: Include JDBC driver in application's manifest

If you build JAR files from the generated Java application, you will need to add a reference to the database driver in the `manifest` section of the **`build.xml`** file. This ensures that the reference to the database driver is available in the manifest (**`MANIFEST.MF`**) file after you build the project.

To add the database reference to the manifest file, take the steps below:

1. Locate the `manifest` element in the **`build.xml`** file.
2. Add a new element called `attribute` where the `name` attribute is `Class-Path` and the `value` attribute is the name of the **`.jar`** file. For example, for MySQL 5.1.16, the updated `manifest` file could look as follows (*note the line highlighted in yellow*):

```
<manifest file="C:\codegen\java\mysql_mapping/META-INF/MANIFEST.MF" mode="replace">
   <attribute name="Created-By" value="MapForce 2025"/>
   <attribute name="Main-Class" value="com.mapforce.MappingConsole"/>
   <attribute name="Class-Path" value="mysql-connector-java-5.1.16-bin.jar"/>
</manifest>
```

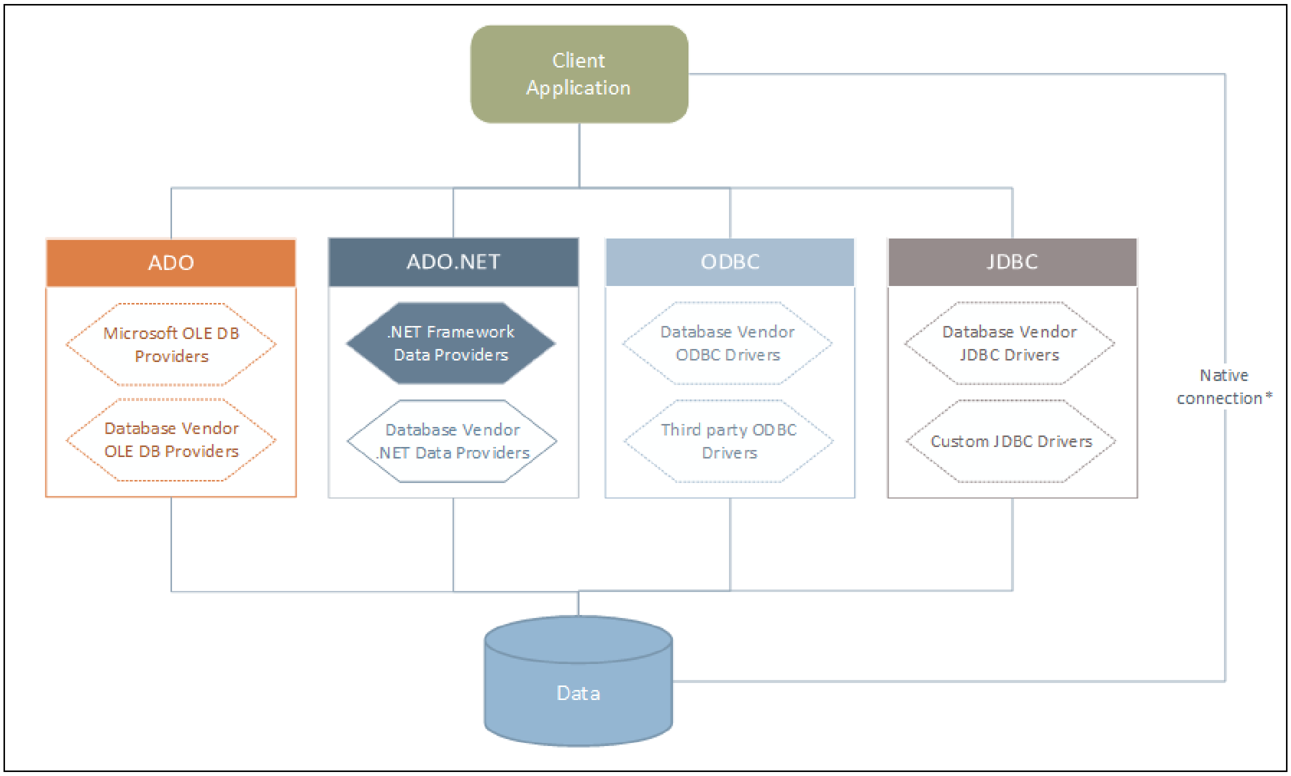
3. Copy the JAR file of the JDBC driver to the folder that contains the JAR file of the generated application.

4.2.1 Connect to a Data Source

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

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

The diagram below illustrates 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[194] are supported for SQLite, MySQL, MariaDB, PostgreSQL databases. To connect to such databases, you do not need to install any additional drivers on your system.*

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

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

Note: Some ADO.NET providers are not supported or have limited support. See ADO.NET Support Notes[186].

About data access technologies

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

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

4.2.1.1 Start Database Connection Wizard

MapForce provides a Database Copnnection Wizard that guides you through the steps required to set up a connection to a data source. Before you go through the wizard steps, be aware that for some database types it is necessary to install and separately configure several database prerequisites, such as a database driver or database client software. These are normally provided by the respective database vendors, and include documentation tailored to your specific Windows version. For a list of database drivers grouped by database type, see Database Drivers Overview 172.

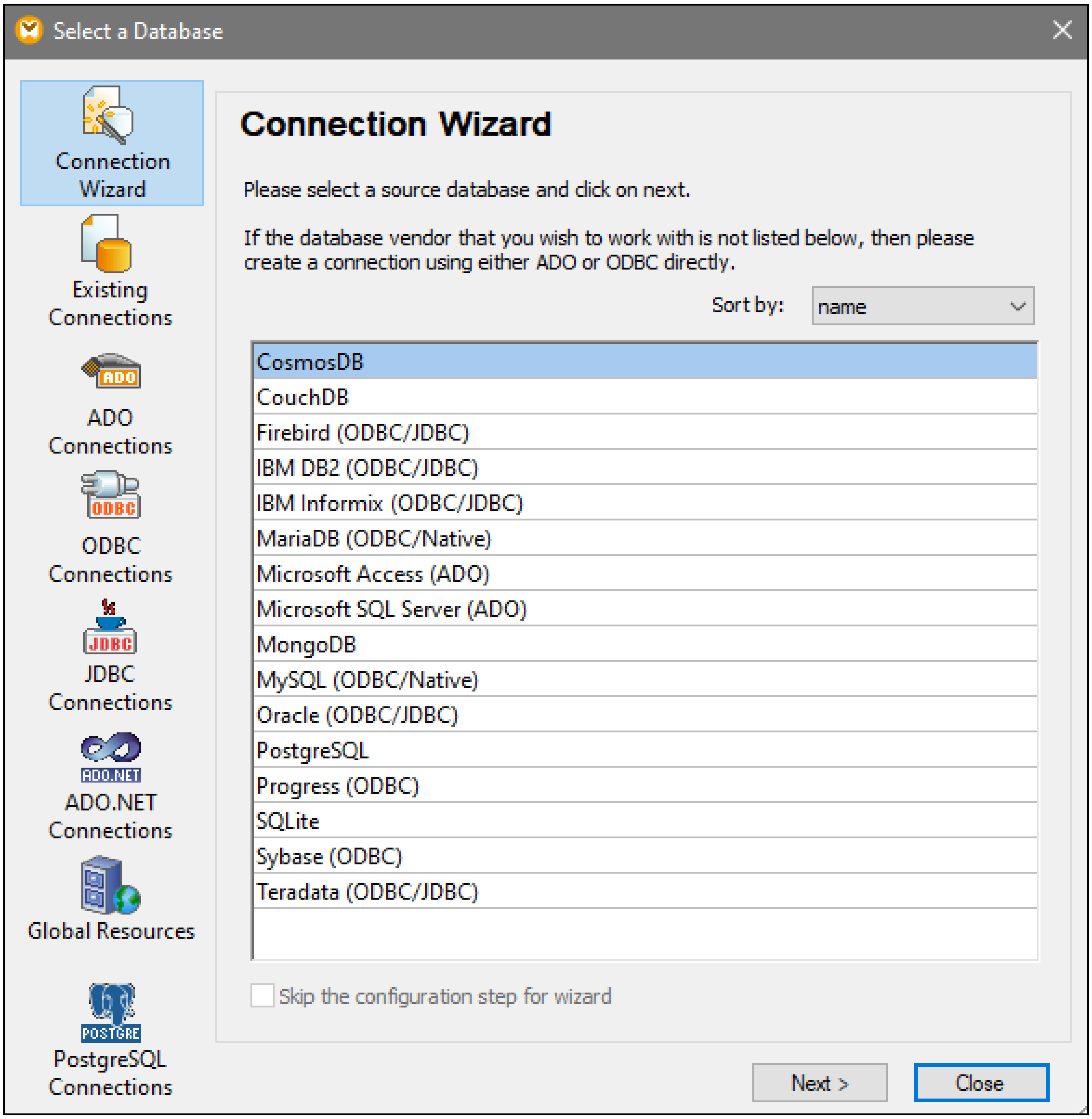
To start the Database Connection Wizard (*see screenshot below*), do the following:

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

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

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

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



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.

Sort by: name

- CosmosDB
- CouchDB
- Firebird (ODBC/JDBC)
- IBM DB2 (ODBC/JDBC)
- IBM Informix (ODBC/JDBC)
- MariaDB (ODBC/Native)
- Microsoft Access (ADO)
- Microsoft SQL Server (ADO)
- MongoDB
- MySQL (ODBC/Native)
- Oracle (ODBC/JDBC)
- PostgreSQL
- Progress (ODBC)
- SQLite
- Sybase (ODBC)
- Teradata (ODBC/JDBC)

☐ Skip the configuration step for wizard

The wizard will take you through the next steps according to the database type, connection technology (ADO, ADO.NET, ODBC, JDBC), and driver that will be used. For examples applicable to each database type, see Database Connection Examples 196.

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

- Setting up an ADO Connection 176
- Setting up an ADO.NET Connection 181
- Setting up an ODBC Connection 190
- Setting up a JDBC Connection 187

4.2.1.2 Database Drivers Overview

This topic gives an overview of database drivers. 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.

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, we recommend that you carefully read 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 186.
- 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 192.
- When setting up an ODBC data source, it is recommended that you create the data source name (DSN) as a System DSN instead of as a User DSN. For more information, see Setting up an ODBC Connection 190.
- If the target database is MySQL or MariaDB through ODBC, the option *Return matched rows instead of affected rows* must be enabled in the Cursor/Results tab of MySQL ODBC Connector. Alternatively, if you enter the connection string manually in the Database Connection wizard, add `Option=2` to the connection string (e.g., `Dsn=mydsn;Option=2;`).
- When setting up a JDBC data source, ensure that JRE (Java Runtime Environment) or Java Development Kit (JDK) is installed and that the **`CLASSPATH`** environment variable of the operating system is configured. For more information, see Setting up a JDBC Connection 187.
- 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.

Available data-access technologies

The table below lists common database drivers you can use to connect to a particular database through a particular data access technology. Note that this list is neither exhaustive nor prescriptive; you can use other native or third party alternatives in addition to the drivers shown below.

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

Database	Interface	Drivers
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/)
	Native connection	Available. No drivers are required.
Microsoft SQL Server	ADO	• Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL) • Microsoft OLE DB Provider for SQL Server (SQLOLEDB) • SQL Server Native Client (SQLNCLI)
	ADO.NET	• .NET Framework Data Provider for SQL Server • .NET Framework Data Provider for OLE DB
	JDBC	• Microsoft JDBC Driver for SQL Server (https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server)
	ODBC	• ODBC Driver for Microsoft SQL Server (https://docs.microsoft.com/en-us/SQL/connect/odbc/download-odbc-driver-for-sql-server)
MySQL	ADO.NET	• Connector/NET (https://dev.mysql.com/downloads/connector/net/)
	JDBC	Connector/J (https://dev.mysql.com/downloads/connector/j/)
	ODBC	Connector/ODBC (https://dev.mysql.com/downloads/connector/odbc/)
	Native connection	Available for MySQL 5.7 and later. No drivers are required.
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)

Database	Interface	Drivers
	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. No drivers are required.
Progress OpenEdge	JDBC	JDBC Connector (https://www.progress.com/jdbc/openedge)
	ODBC	ODBC Connector (https://www.progress.com/odbc/openedge)
SQLite	Native connection	Available. No drivers are required.
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)

Database	Interface	Drivers
	ODBC	Teradata ODBC Driver for Windows (https://downloads.teradata.com/download/connectivity/odbc-driver/windows)

4.2.1.3 ADO Connection

Microsoft ActiveX Data Objects (ADO) is a data access technology that enables you to connect to a variety of data sources through OLE DB. OLE DB is an alternative interface to ODBC or JDBC; it provides uniform access to data in a COM (Component Object Model) environment. ADO is a precursor of the newer ADO.NET 181 and is still one of the possible ways to connect to Microsoft native databases such as Microsoft Access or SQL Server, although you can also use it for other data sources.

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

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

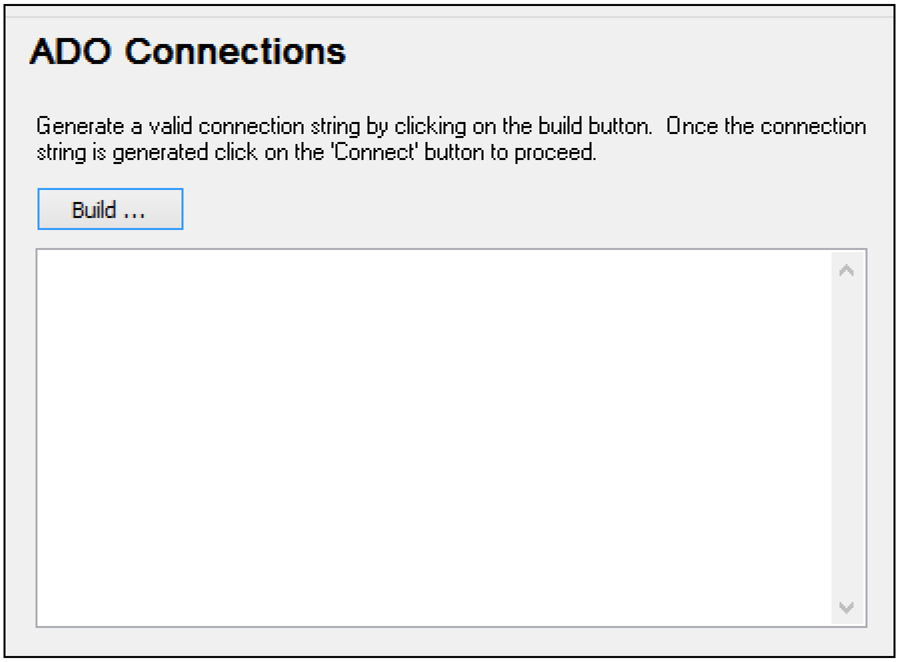
From the providers listed above, the recommended one is MSOLEDBSQL; you can download it from https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15. Note that it must match the platform of MapForce (32-bit or 64-bit). The SQLOLEDB and SQLNCLI providers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

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

Set up an ADO connection

To set up an ADO connection, do the following:

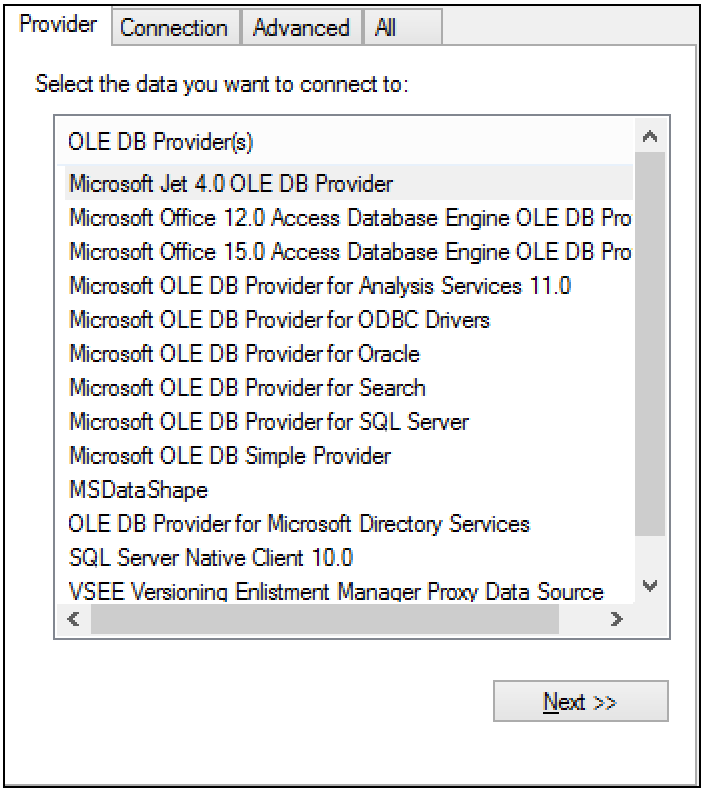
1. Start the database connection wizard 170.
2. Click **ADO Connections**.



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** (recommended) - **Microsoft Jet OLE DB Provider** If the **Microsoft Office Access Database Engine OLE DB Provider** is not available in the list, make sure that you have installed either Microsoft Access or the Microsoft Access Database Engine Redistributable (https://www.microsoft.com/en-us/download/details.aspx?id=54920) on your computer.
SQL Server	- **Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL)** - this is the recommended OLE DB provider. In order for this provider to appear in the list, it must be downloaded from https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15 and installed. - **Microsoft OLE DB Provider for SQL Server (OLEDBSQL)** - **SQL Server Native Client (SQLNCLI)**
Other database	Select the provider applicable to your database. If an OLE DB provider to your database is not available, install the required driver from the database vendor (see Database Drivers Overview 172). Alternatively, set up an ADO.NET, ODBC, or JDBC connection. If the operating system has an ODBC driver to the required database, you could also use the **Microsoft OLE DB Provider for ODBC Drivers**, or preferably opt for an ODBC connection 190.

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

The subsequent wizard steps are specific to the provider you chose. For SQL Server, you will need to provide or select the host name of the database server, the authentication method, the database name, as well as the database username and password. For an example, see Connecting to Microsoft SQL Server (ADO) 211. For Microsoft Access, you will be asked to browse for or provide the path to the database file. For an example, see Connecting to Microsoft Access (ADO) 209.

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

- Setting up the SQL Server Data Link Properties 179
- Setting up the Microsoft Access Data Link Properties 180

4.2.1.3.1 Connect to an Existing MS Access Database

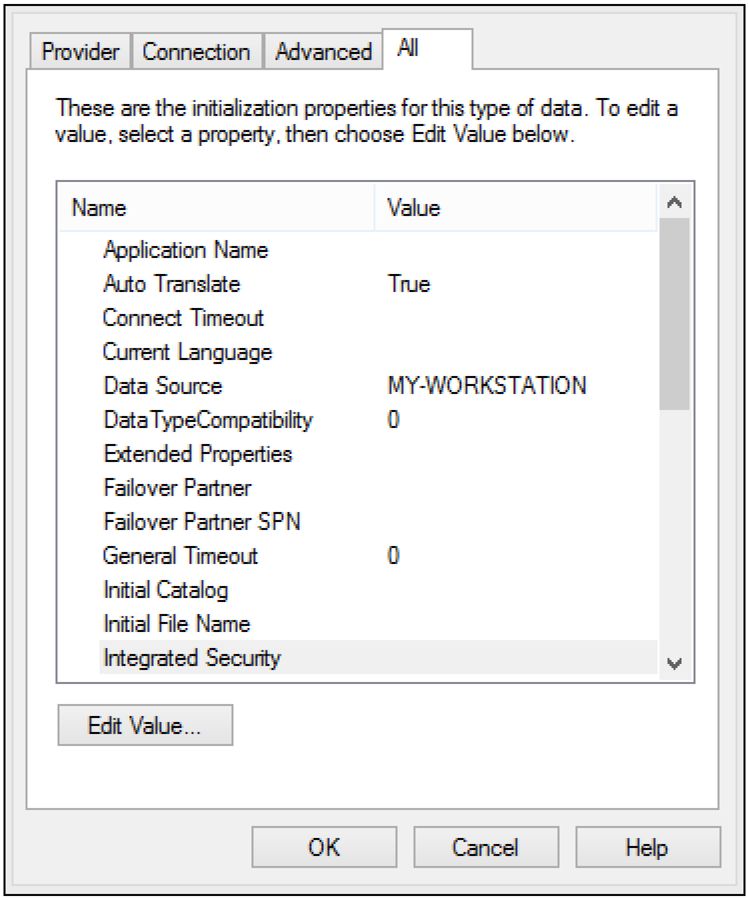
The procedure given below is suitable if you want to connect to a Microsoft Access database that is not password-protected. If the database is password-protected, use the method given in the Connection Example for Microsoft Access (ADO)[209].

1. Run the database connection wizard (see Starting the Database Connection Wizard[170]).
2. Select **Microsoft Access (ADO)**, and then click **Next**.
3. Browse for the database file, or enter the path to it (either relative or absolute).
4. Click **Connect**.

4.2.1.3.2 Set Up SQL Server Data Link Properties

If you connect to a Microsoft SQL Server database through ADO[176], check the following properties in the *All* tab of the Data Link Properties dialog box (*screenshot below*).

- *Integrated Security:* If SQL Server Native Client is the data provider on the Provider tab, set this property to a space character.
- *Persist Security Info:* Set this property to *True*.

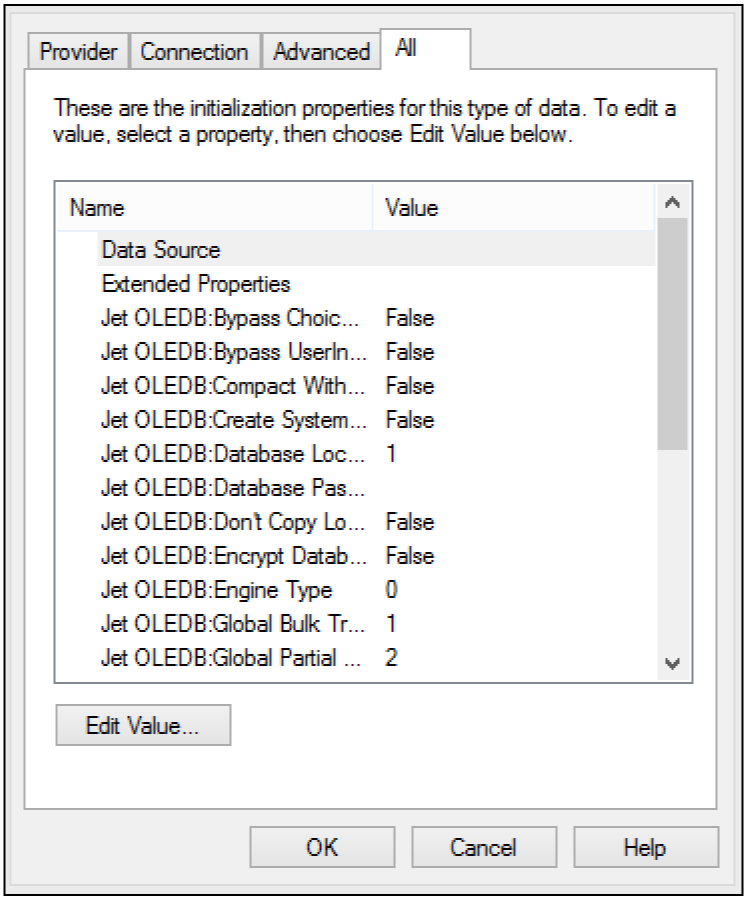


These are the initialization properties for this type of data. To edit a value, select a property, then choose Edit Value below.

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	

4.2.1.3.3 Set Up MS Access Data Link Properties

If you connect to a Microsoft Access database through ADO[176], then in the *All* tab of the Data Link Properties dialog box (*screenshot below*), check the properties listed below the screenshot.

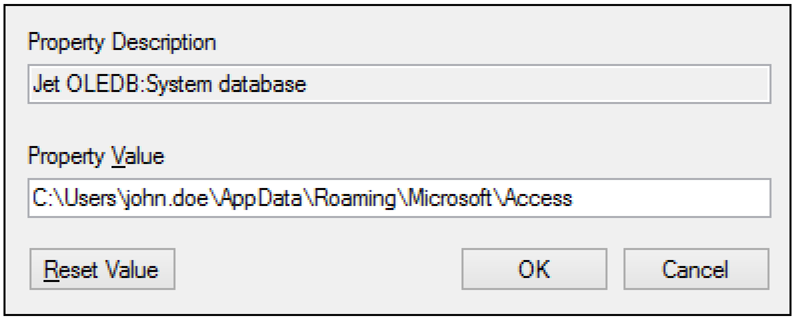


These are the initialization properties for this type of data. To edit a value, select a property, then choose Edit Value below.

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 Source
This property stores the path to the Microsoft Access database file. We recommend using 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 Description

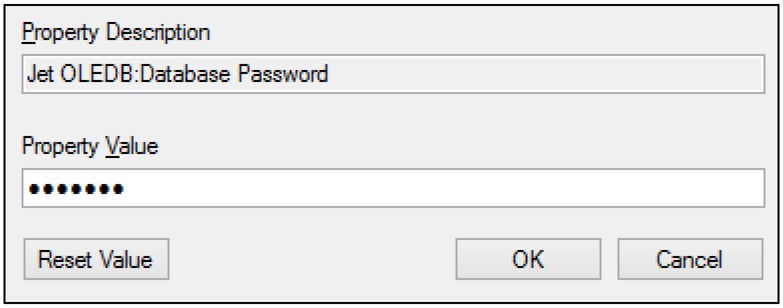
Jet OLEDB:System database

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

•••••••

4.2.1.4 ADO.NET Connection

ADO.NET is a set of Microsoft .NET Framework libraries designed to interact with data, including data from databases. To connect to a database from MapForce through ADO.NET, Microsoft .NET Framework 4 or later is required. 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. So, when you use ADO.NET, MapForce interacts with a database through a data provider—one that is optimized to work with the specific type of data source that it is designed for.

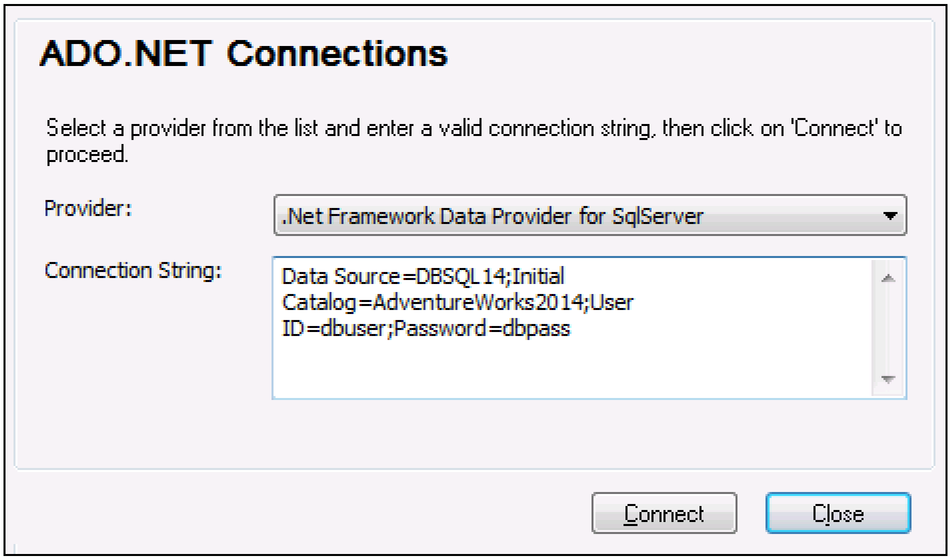
There are two types of .NET providers:

- Supplied by default with Microsoft .NET Framework.
- 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 186.

Set up the connection as follows.

1. Start the database connection wizard[170].
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[182]). The syntax of the connection string depends on the provider selected from the "Provider" list. For examples, see Sample ADO.NET Connection Strings[185].



ADO.NET Connections

Select a provider from the list and enter a valid connection string, then click on 'Connect' to proceed.

Provider: .Net Framework Data Provider for SqlServer

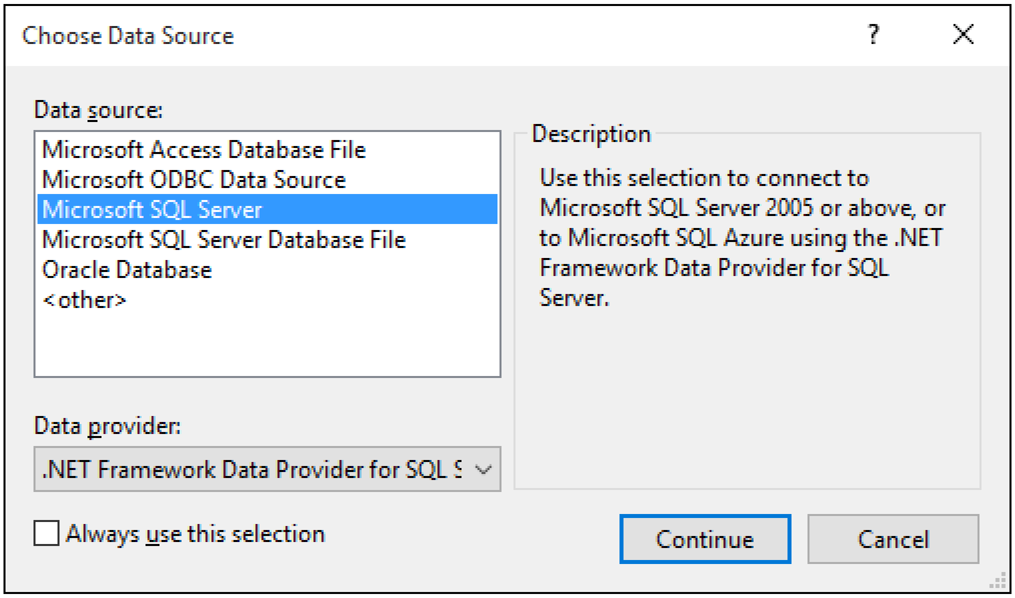
Connection String: Data Source=DBSQL14;Initial Catalog=AdventureWorks2014;User ID=dbuser;Password=dbpass

5. Click Connect.

4.2.1.4.1 Creating a Connection String in Visual Studio

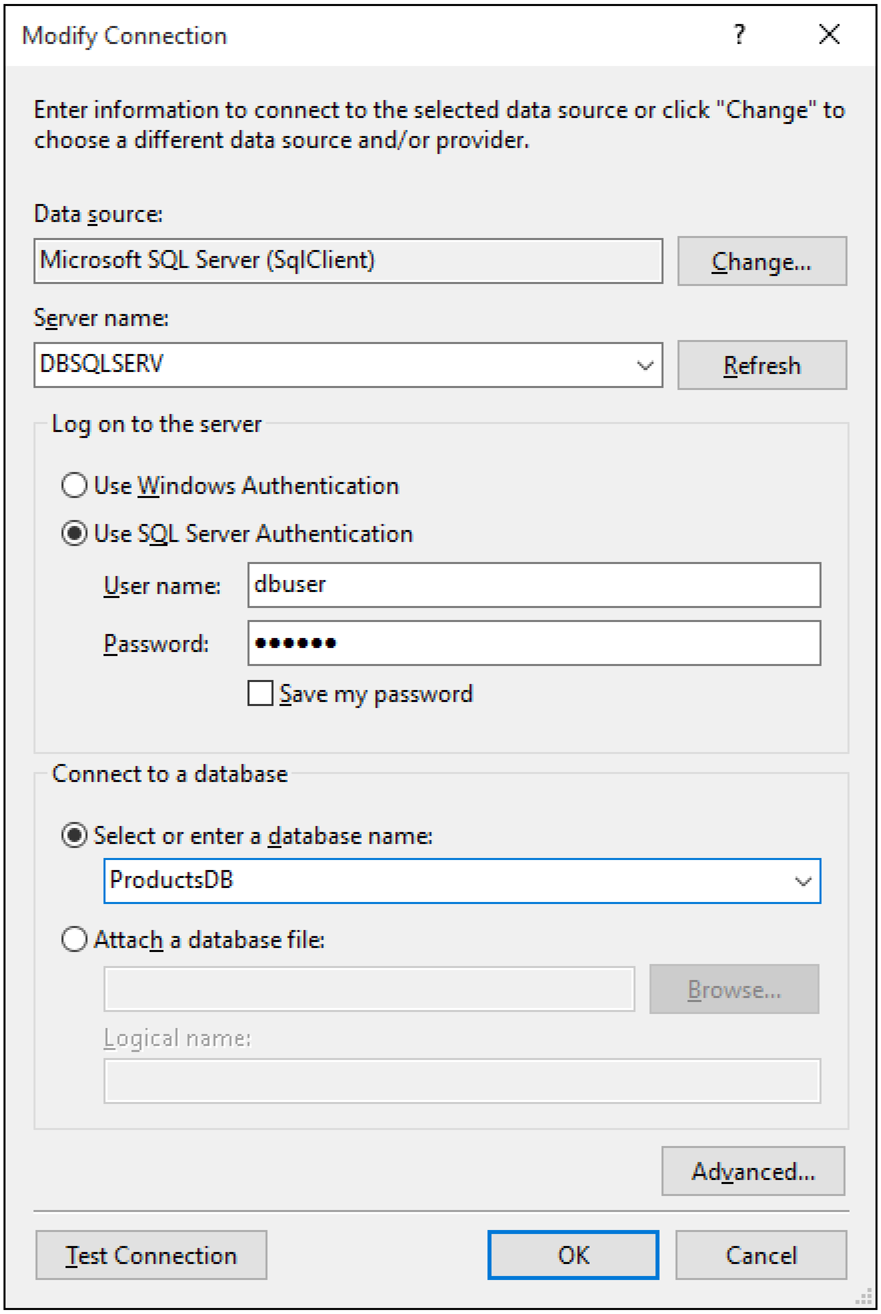
In order to connect to a data source using ADO.NET, a valid database connection string is required. The following instructions show you how to create a connection string from Visual Studio.

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>
Data provider:
.NET Framework Data Provider for SQL S
Description
□ Always use this selection
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.

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.

4.2.1.4.2 Sample ADO.NET Connection Strings

To set up an ADO.NET connection, you need to select an ADO.NET provider from the database connection dialog box and enter a connection string (see also Setting up an ADO.NET Connection 181). 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
A sample connection string looks as follows:

```
    Database=PRODUCTS;UID=user;Password=password;Server=localhost:50000;
```

Note: This provider is typically installed with the IBM DB2 Data Server Client package. If the provider is missing from the list of ADO.NET providers after installing IBM DB2 Data Server Client package, refer to the following technical note: https://www-01.ibm.com/support/docview.wss?uid=swg21429586. See also: https://www.ibm.com/support/knowledgecenter/en/SSEPGG_10.1.0/com.ibm.swg.im.dbclient.adonet.ref.doc/doc/DB2ConnectionClassConnectionStringProperty.html

Oracle Data Provider for .NET (ODP.NET)
The installation package which includes the ODP.NET provider can be downloaded from the Oracle website (see http://www.oracle.com/technetwork/topics/dotnet/downloads/index.html). A sample connection string looks as follows:

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

where `DSORCL` is the name of the data source which points to an Oracle service name defined in the **`tnsnames.ora`** file, as described in Connecting to Oracle (ODBC) 220.

To connect without configuring a service name in the **`tnsnames.ora`** file, use a string such as:

```
Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=host)(PORT=port)))
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=MyOracleSID)));User
Id=user;Password=password;
```

See also: https://docs.oracle.com/cd/B28359_01/win.111/b28375/featConnecting.htm.

4.2.1.4.3 ADO.NET Support Notes

The following table lists known ADO.NET database drivers that are currently not supported or have limited support in MapForce.

Database	Driver	Support notes
All databases	**.Net Framework Data Provider for ODBC**	Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ODBC direct connections instead.
	.Net Framework Data Provider for OleDb	Limited support. Known issues exist with Microsoft Access connections. It is recommended to use ADO direct connections instead.
Firebird	**Firebird ADO.NET Data Provider**	Limited support. It is recommended to use ODBC or JDBC instead.
Informix	**IBM Informix Data Provider for .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. Use a native connection instead.

Database	Driver	Support notes
Sybase	—	No ADO.NET drivers for this vendor are supported.

4.2.1.5 JDBC Connection

JDBC (Java Database Connectivity) is a database access interface which is part of the Java software platform from Oracle. JDBC connections are generally more resource-intensive than ODBC connections but may provide features not available through ODBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings 1082; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC drivers from the database vendor must be installed. These may be JDBC drivers installed as part of a database client installation, or supported JDBC libraries (`.jar` files) that are downloaded separately. See also Database Connection Examples 196.
- The `CLASSPATH` environment variable must include the path to the JDBC driver (one or several `.jar` files) on your Windows operating system. When you install some database clients, the installer may configure this variable automatically. See also Configuring the CLASSPATH 189.

Connect to SQL Server via JDBC with Windows credentials

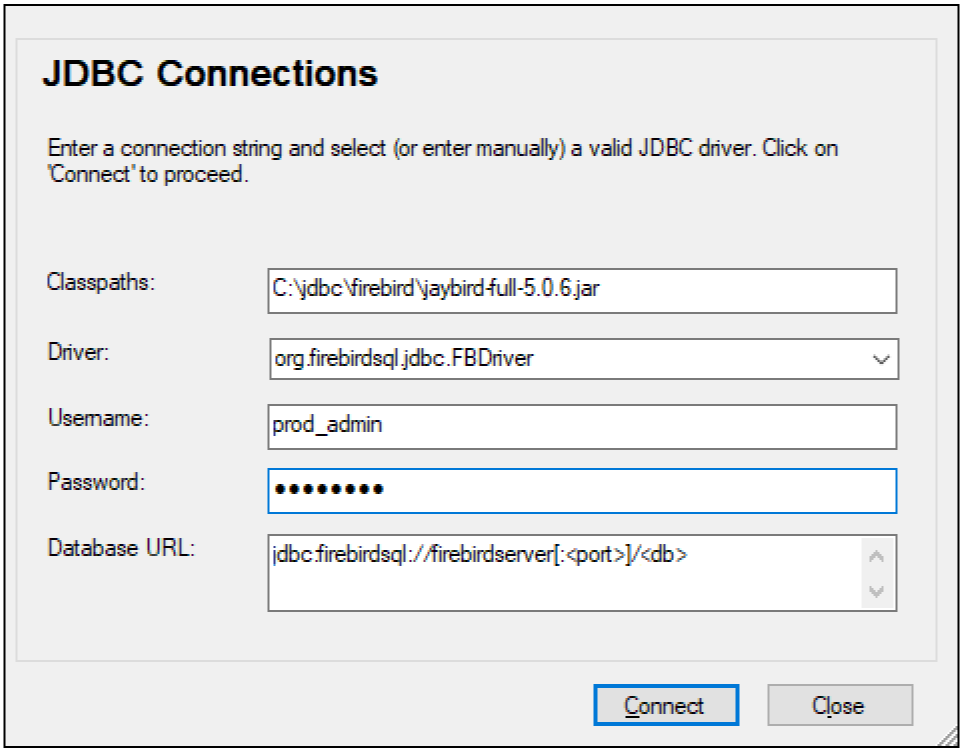
If you connect to SQL Server through JDBC with Windows credentials (integrated security), note the following:

- The **`sqljdbc_auth.dll`** file included in the JDBC driver package must be copied to a directory that is on the system `PATH` environment variable. There are two such files, one for the x86 and one for x64 platform. Make sure that you add to the `PATH` the one that corresponds to your JDK platform. Also, make sure that you restart MapForce (or the program that runs the mapping) after changing the environment variable.
- The JDBC connection string must include the property **`integratedSecurity=true`**. You can add this property from various places:
  - from the database connection wizard, see below
  - from the database component settings 235
  - if applicable, by editing the database connection string in generated Java code.

For further information, refer to *Microsoft JDBC driver for SQL Server* documentation, https://docs.microsoft.com/en-us/sql/connect/jdbc/building-the-connection-url.

Set up a JDBC connection

1. Start the database connection wizard 170 and click JDBC Connections.
2. If required, enter a semicolon-separated list of `.jar` file paths in the *Classpaths* field. The `.jar` libraries entered here will be loaded into the environment in addition to those already defined in the `CLASSPATH` 189 environment variable. JDBC drivers found in the source .jar libraries referenced via the *Classpaths* field and the system's `CLASSPATH` 189 are listed in the Driver dropdown list (see next step).



JDBC Connections

Enter a connection string and select (or enter manually) a valid JDBC driver. Click on 'Connect' to proceed.

Classpaths: C:\jdbc\firebird\jaybird-full-5.0.6.jar

Driver: org.firebirdsql.jdbc.FBDriver

Username: prod_admin

Password: ●●●●●●●●●

Database URL: jdbc:firebirdsql://firebirdserver[:<port>]/<db>

3. In the *Driver* field, select a JDBC driver from the list or enter a Java class name. The list will contain the JDBC drivers configured through in the *Classpaths* field (see above) and the `CLASSPATH` 189 environment variable.

> 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 that you 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.

4. Enter the username and password of the database in the corresponding fields.
5. In the *Database URL* field, enter the JDBC connection URL (JDBC connection string) in the format specific to your database type. The following table describes the syntax of JDBC connection strings for common database types.

Database	JDBC Connection URL
Firebird	`jdbc:firebirdsql://<host>[:<port>]/<database path or alias>`
IBM DB2	`jdbc:db2://hostName:port/databaseName`
IBM DB2 for i	`jdbc:as400://[host]`
IBM Informix	`jdbc:informix-sqli://hostName:port/databaseName:INFORMIXSERVER=myserver`

Database	JDBC Connection URL
MariaDB	`jdbc:mariadb://hostName:port/databaseName`
Microsoft SQL Server	`jdbc:sqlserver://hostName:port;databaseName=name`
MySQL	`jdbc:mysql://hostName:port/databaseName`
Oracle	`jdbc:oracle:thin:@hostName:port:SID` `jdbc:oracle:thin:@//hostName:port/service`
Oracle XML DB	`jdbc:oracle:oci:@//hostName:port:service`
PostgreSQL	`jdbc:postgresql://hostName:port/databaseName`
Progress OpenEdge	`jdbc:datadirect:openedge://host:port;databaseName=db_name`
Sybase	`jdbc:sybase:Tds:hostName:port/databaseName`
Teradata	`jdbc:teradata://databaseServerName`

Note that syntax variations of the formats listed above are also possible. For example, the database URL may exclude the port or may include the username and password of the database. Check the documentation of the database vendor for further details.

6. Click **Connect**.

4.2.1.5.1 Configuring the CLASSPATH

The `CLASSPATH` environment variable is used by the Java Runtime Environment (JRE) or the Java Development Kit (JDK) to locate Java classes and other resource files on your operating system. When you connect to a database through JDBC, this variable must be configured to include the path to the JDBC driver on your operating system, and, in some cases, the path to additional library files specific to the database type you are using.

The following table lists sample file paths that must be typically included in the `CLASSPATH` variable. Importantly, you may need to adjust this information based on the location of the JDBC driver on your system, the JDBC driver name, as well as the JRE/JDK version present on your operating system. To avoid connectivity problems, check the installation instructions and any pre-installation or post-installation configuration steps applicable to the JDBC driver installed on your operating system.

Database	Sample CLASSPATH entries
Firebird	`C:\Program Files\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar`
IBM DB2	`C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;`
IBM DB2 for i	`C:\jt400\jt400.jar;`
IBM Informix	`C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;`

Database	Sample CLASSPATH entries
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`

Note the following points:

- The `CLASSPATH` setting is available in the Environment Variables setting of your Windows system. Include in the `CLASSPATH` the path where the JDBC driver is located, separating it from other paths by a semi-colon.
- 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.

4.2.1.6 ODBC Connection

ODBC (Open Database Connectivity) is a widely used data access technology that enables you to connect to a database from MapForce. It can be used either as primary means to connect to a database, or as an alternative to native, OLE DB, or JDBC-driven connections.

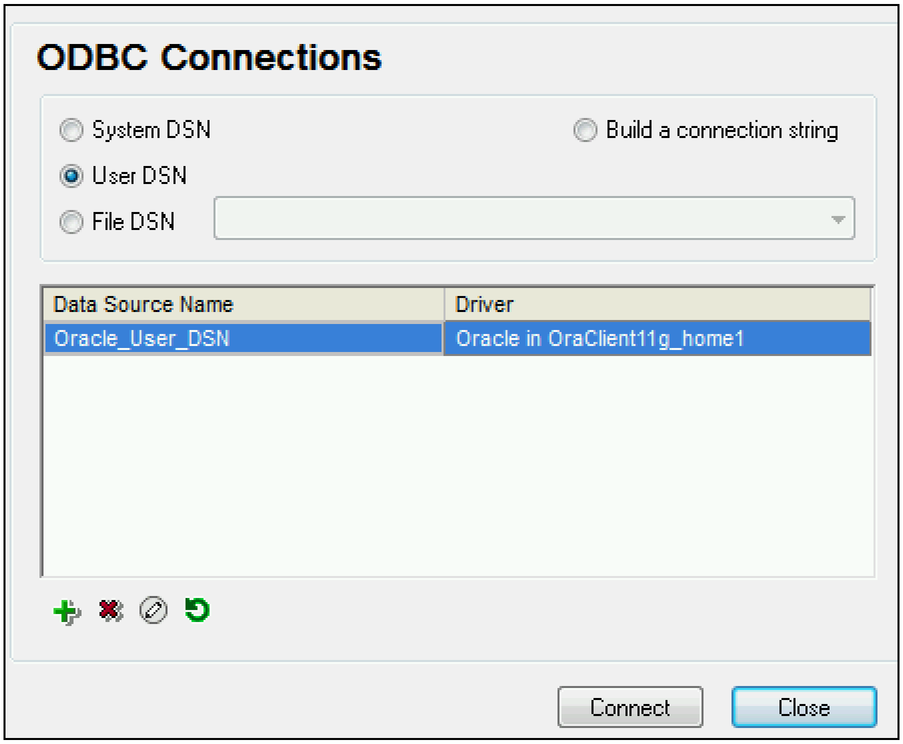
To connect to a database through ODBC, first you need to create an ODBC data source name (DSN) on the operating system. This step is not required if the DSN has already been created, perhaps by another user of the operating system. The DSN represents a uniform way to describe the database connection to any ODBC-aware client application on the operating system, including MapForce. DSNs can be of the following types:

- System DSN

- User DSN
- File DSN

A *System* data source is accessible by all users with privileges on the operating system. A *User* data source is available to the user who created it. Finally, if you create a *File DSN*, the data source will be created as a file with the .dsn extension which you can share with other users, provided that they have installed the drivers used by the data source.

Any DSNs already available on your machine are listed by the database connection dialog box when you click **ODBC connections** on the ODBC connections dialog box (*screenshot below*).



Data Source Name	Driver
Oracle_User_DSN	Oracle in OraClient11g_home1

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 192).

Connect with a new DSN

To connect with a new DSN, carry out the following steps. Note that, to create a System DSN, you need administrative rights on the operating system and MapForce must be run as administrator..

1. Start the database connection wizard 170.
2. On the database connection dialog box, click **ODBC Connections**.
3. Select a data source type (User DSN, System DSN, File DSN).
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 172).

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.

Connect with an existing DSN

To connect with an existing DSN, do the following.

1. Start the database connection wizard 170.
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**.

Build a connection string based on an existing .dsn file

Do this as follows.

1. Start the database connection wizard 170.
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**.

Connect by using a prepared connection string

Do this as follows.

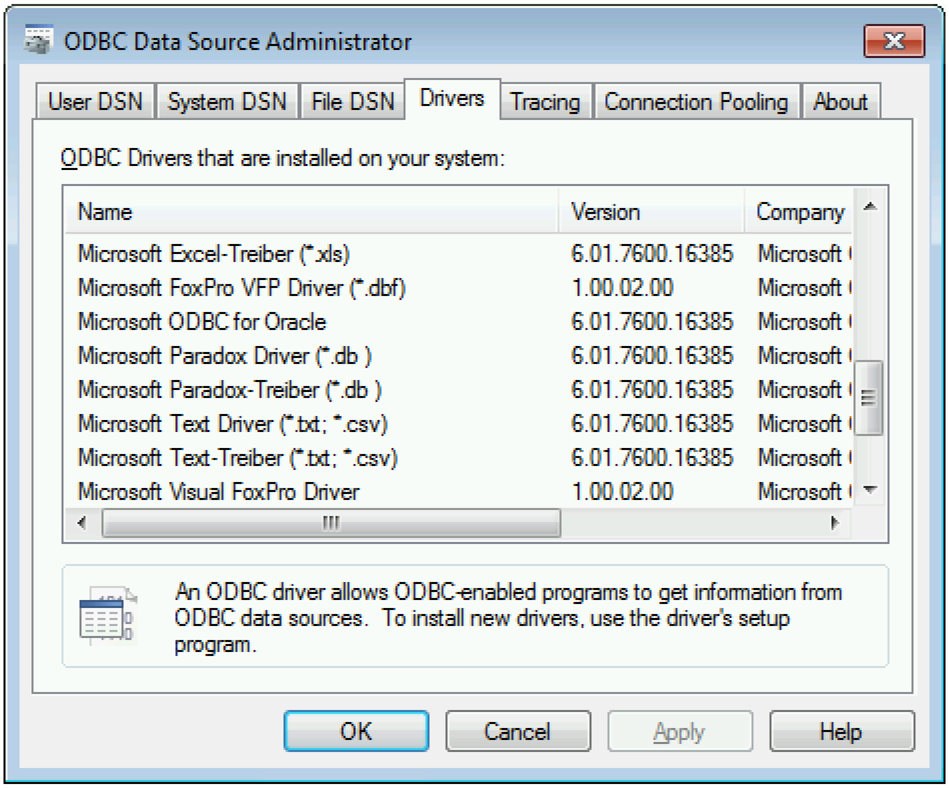
1. Start the database connection wizard 170.
2. Click **ODBC Connections**.
3. Select **Build a connection string**.
4. Paste the connection string into the provided box, and then click **Connect**.

4.2.1.6.1 Available ODBC Drivers

You can view the ODBC drivers available on your operating system in the ODBC Data Source Administrator. You can access the ODBC Data Source Administrator (**Odbcad32.exe**) from the Windows Control Panel, under Administrative Tools. On 64-bit operating systems, there are two versions of this executable:

- The 32-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\SysWoW64** directory (assuming that **C:** is your system drive).
- The 64-bit version of the **Odbcad32.exe** file is located in the C**:\Windows\System32** directory.

Any installed 32-bit database drivers are visible in the 32-bit version of ODBC Data Source Administrator (*screenshot below*), 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.

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 172). Once the ODBC driver is available on your system, you are ready to create ODBC connections with it (see Setting up an ODBC Connection 190).

4.2.1.7 SQLite Connection

SQLite is a file-based, self-contained database type, which makes it ideal in scenarios where portability and ease of configuration is important. Since SQLite databases are natively supported by MapForce, you do not need to install any drivers to connect to them.

SQLite database support notes

- On Linux, statement execution timeout for SQLite databases is not supported.
- Full text search tables are not supported.
- SQLite allows values of different data types in each row of a given table. All processed values must be compatible with the declared column type; therefore, unexpected values can be retrieved and run-time errors may occur if your SQLite database has row values which are not the same as the declared column type.
- If your mapping should write data to a SQLite database, and if you don't have the target database file already, you will need to create it separately. In this case, you can create it with a tool such as

DatabaseSpy or download the SQLite command-line shell from the official website, and create the database file from the command line (see also Example: Mapping data from XML to SQLite). For complete reference to SQLite command syntax, refer to the official SQLite documentation.

- SQLite databases are supported in the MapForce BUILT-IN transformation language (either when you preview the mapping or when you run a MapForce Server execution file).
- SQLite databases are not supported in user-defined functions (UDF).

Important: It is recommended to create tables with the `STRICT` keyword to ensure more predictable behavior of your data. Otherwise, the data may not be read or written correctly when values of different types are mixed in one column. To find out more about STRICT tables, see the SQLite documentation.

4.2.1.7.1 Connect to an Existing SQLite Database

To connect to an existing SQLite database, do the following.

1. Run the database connection wizard (see Starting the Database Connection Wizard 170).
2. Select **SQLite**, and then click **Next**.
3. Browse for the SQLite database file, or enter the path (either relative or absolute) to the database. The **Connect** button becomes enabled once you enter the path to a SQLite database file.
4. Click **Connect**.

4.2.1.8 Native Connections

Native connections are direct connections to a database's own network protocols and drivers. Therefore, no additional drivers need to be installed. Native connections provide the most efficient method of interacting with the database and full feature support.

If you intend to deploy files for execution on a Linux or macOS server, you do not need to install drivers on the target server.

You can set up native connections to the following DBs:

- MariaDB
- MySQL
- SQLite
- PostgreSQL

Connection setup

To set up a native connection, follow the steps below:

1. Start the database connection wizard 170.
2. Select the DB you want to connect to.
3. In the dialog that appears, enter the relevant connection details, for example, the host (e.g., *localhost*), optionally the port (typically 5432), SSL Mode in the case of MySQL, the database name, username, and password in the corresponding boxes.
4. Click **Connect**.

SQLite conections

For detailed information about SQLite connections, see the topic SQLite Connection 193.

Notes for PostrgreSQL

If the PostgreSQL database server is on a different machine, note the following:

- The PostgreSQL database server must be configured to accept connections from clients. Specifically, the **pg_hba.conf** file must be configured to allow non-local connections. Secondly, the **postgresql.conf** file must be configured to listen on specified IP address(es) and port. For more information, check the PostgreSQL documentation (https://www.postgresql.org/docs/9.5/static/client-authentication-problems.html).
- The server machine must be configured to accept connections on the designated port (typically, 5432) through the firewall. For example, on a database server running on Windows, a rule may need to be created to allow connections on port 5432 through the firewall, from **Control Panel > Windows Firewall > Advanced Settings > Inbound Rules**.

4.2.1.9 Global Resources

After you have created a database as a global resource, its connection details are stored and can be used across all Altova products installed on your machine.

Create a database as a global resource

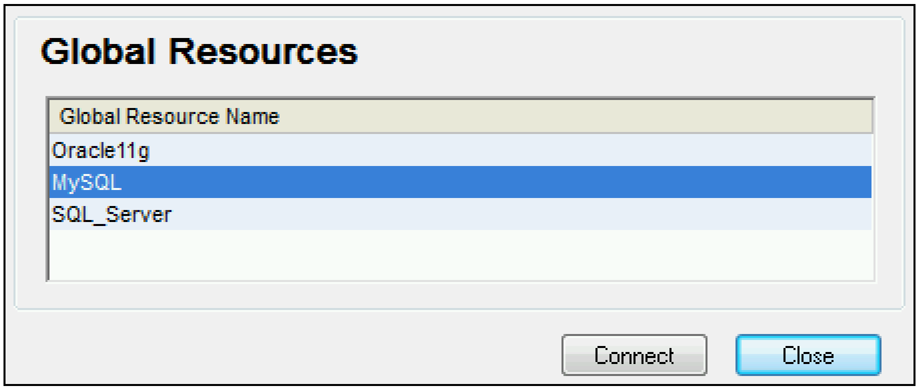
To create a database as a global resource, do the following

1. On the **Tools** menu of MapForce, click **Global Resources**.
2. Click **Add**, and then click Database.
3. Type in a name for the global resource in the *Resource Alias* field.
4. Click **Choose Database**. The Connection Wizard 170 appears.
5. Use the Connection Wizard to add a database connection as described above.

Use a global-resource database

To use a database that has been created as a global resource (*see above*), do the following:

1. Start the Connection Wizard as described above.
2. Select Global Resources. All the databases that have been created as global resources will be listed by their names in the Global Resources pane (*see screenshot below*).



3. Select the global resource that you want. Tip: Move the mouse cursor over a global resource in the list to see information about the database.

4.2.1.10 Database Connection Examples

This section includes examples for connecting to a database from MapForce through ADO, ODBC, or JDBC. The ADO.NET connection examples are listed separately, see Sample ADO.NET Connection Strings[185]. For instructions about establishing a native connection to PostgreSQL and SQLite, see Setting up a PostgreSQL Connection[194] and Setting up a SQLite Connection[193], respectively.

Note the following points:

- The instructions may differ if your Windows configuration, network environment and the database client or server software are not the same as the ones described in each example.
- For most database types, it is possible to connect using more than one data access technology (ADO, ADO.NET, ODBC, JDBC) or driver. The performance of the database connection, as well as its features and limitations will depend on the selected driver, database client software (if applicable), and any additional connectivity parameters that you may have configured outside MapForce.

4.2.1.10.1 Firebird (JDBC)

This example illustrates how to connect to a Firebird database via JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings[1082]; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The Firebird JDBC driver must be available on your operating system (it takes the form of a .jar file which provides connectivity to the database). The driver can be downloaded from the Firebird website (https://www.firebirdsql.org/). This example uses *Jaybird 5.0.6*.

- You have the following database connection details: host, port, database path, name, or alias, username, and password.

Connection

1. Start the database connection wizard[170] and click **JDBC Connections** (*see JDBC Connection*[187] *for a screenshot of the dialog*).
2. In the *Classpaths* field, enter the path to the `.jar` file that provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of `.jar` file paths. If you have added the filepath to the CLASSPATH[189] of the system, you can leave this field empty.
3. In the *Driver* field, select the appropriate driver. Relevant drivers will be available only if a valid `.jar` file path is in the *Classpaths* field or in the `CLASSPATH` environment variable.
4. Enter the username and password for the database in the corresponding fields.
5. Enter the JDBC connection string in the *Database URL* field according to the pattern in the table below, replacing the highlighted values with the ones applicable to your database server.
6. Click **Connect**.

Connection details of the Firebird example

Field	**Value**
Classpaths	`C:\jdbc\firebird\jaybird-full-5.0.6.jar`
Driver	*org.firebirdsql.jdbc.FBDriver*
Database URL	`jdbc:firebirdsql://`**`<host>`**`[:`**`<port>`**`]/`**`<database path or alias>`**

4.2.1.10.2 Firebird (ODBC)

This example illustrates how to connect to a Firebird 2.5.4 database running on a Linux server.

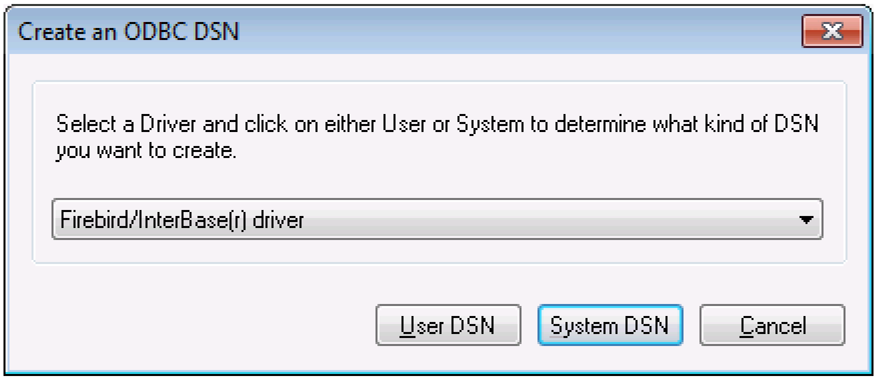
Prerequisites

- The Firebird database server is configured to accept TCP/IP connections from clients.
- The Firebird ODBC driver must be installed on your operating system. This example uses the Firebird ODBC driver version 2.0.3.154 downloaded from the Firebird website.
- 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. 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.
- You have the following database connection details: server host name or IP address, database path (or alias) on the server, user name, and password.

Important: The platform of both the Firebird ODBC driver and client (32-bit or 64-bit) must correspond to that of MapForce. Additionally, the version of the Firebird client must correspond to the version of Firebird server to which you are connecting.

Connection

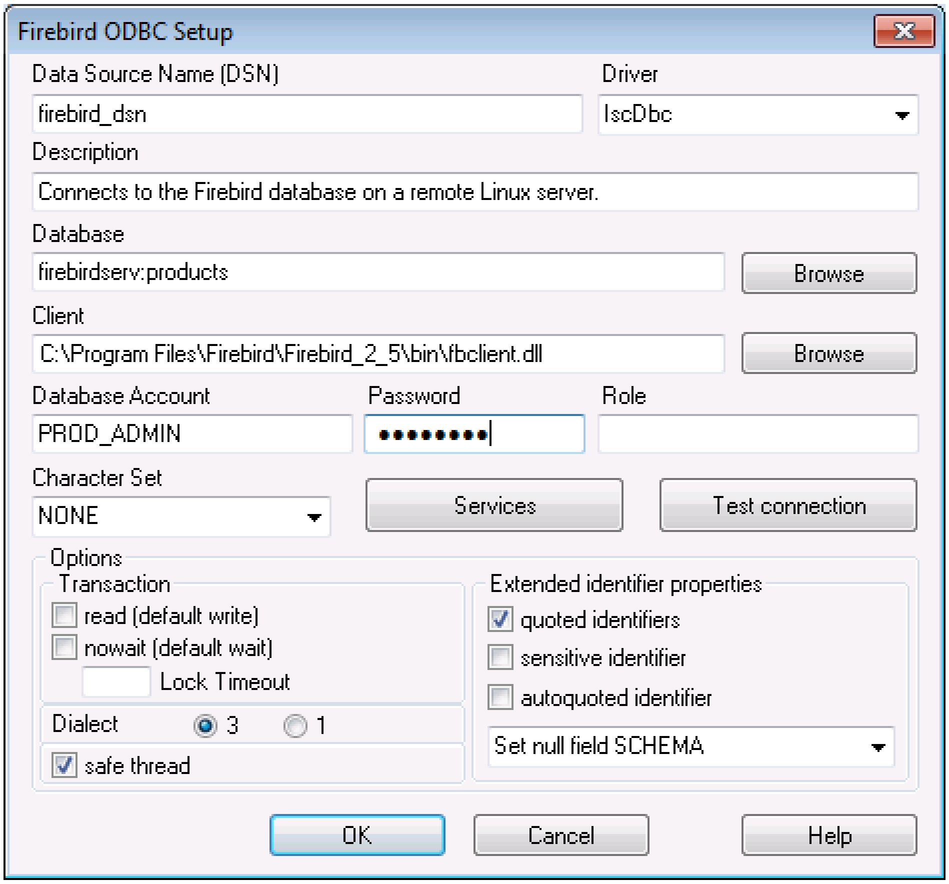
1. Start the database connection wizard[170] and click **ODBC Connections**.
2. Select *User DSN* (or *System DSN* if you have administrative privileges), and then click **Create a New DSN** .


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

3. 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[192]).


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

4. 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.

5. Click **OK** to finish.

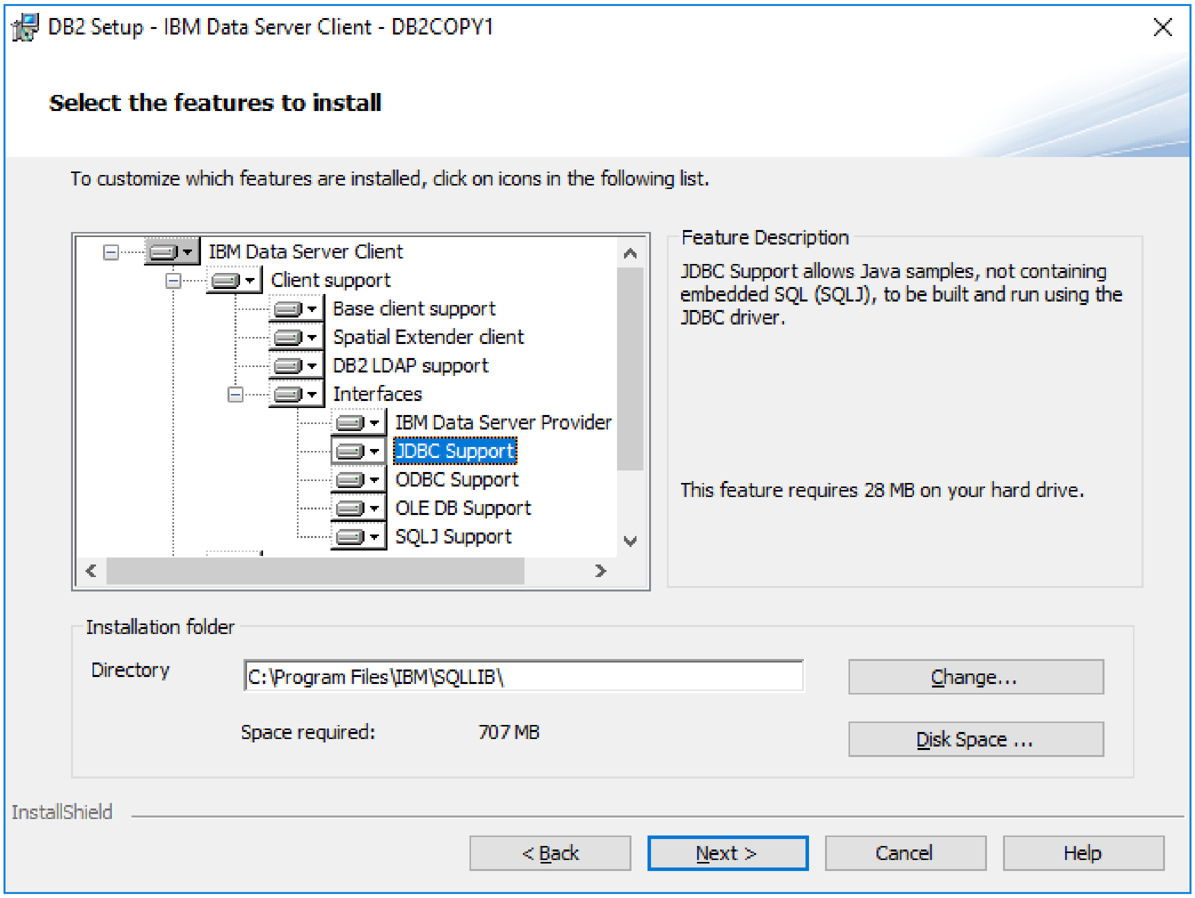
4.2.1.10.3 IBM DB2 (JDBC)

This example illustrates how to connect to an IBM DB2 database server via JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings 1082; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. This example uses the JDBC driver available after installing the IBM Data

Server Client version 10.1 (64-bit). For the JDBC drivers to be installed, choose a *Typical* installation, or select this option explicitly on the installation wizard.



DB2 Setup - IBM Data Server Client - DB2COPY1

Select the features to install

To customize which features are installed, click on icons in the following list.

Feature Description

JDBC Support allows Java samples, not containing embedded SQL (SQLJ), to be built and run using the JDBC driver.

This feature requires 28 MB on your hard drive.

Installation folder

Directory: C:\Program Files\IBM\SQLLIB\

Space required: 707 MB

If you did not change the default installation path, the required `.jar` files will be in the `C:\Program Files\IBM\SQLLIB\java` directory after installation.

- You have the following database connection details: host, port, database path, name, or alias, username, and password.

Connection

1. Start the database connection wizard [170] and click **JDBC Connections** (*see JDBC Connection* [187] *for a screenshot of the dialog*).
2. In the *Classpaths* field, enter the path to the `.jar` file that provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of `.jar` file paths. If you have added the filepath to the CLASSPATH [189] of the system, you can leave this field empty.
3. In the *Driver* field, select the appropriate driver. Relevant drivers will be available only if a valid `.jar` file path is in the *Classpaths* field or in the `CLASSPATH` environment variable.
4. Enter the username and password for the database in the corresponding fields.
5. Enter the JDBC connection string in the *Database URL* field according to the pattern in the table below, replacing the highlighted values with the ones applicable to your database server.
6. Click **Connect**.

Connection details of the IBM DB2 example

Field	Value
Classpaths	`C:\Program Files\IBM\SQLLIB\java\db2jcc.jar`
Driver	*com.ibm.db2.jcc.DB2Driver*
Database URL	`jdbc:db2://`**`hostName`**`:`**`port`**`/`**`databaseName`**

4.2.1.10.4 IBM DB2 (ODBC)

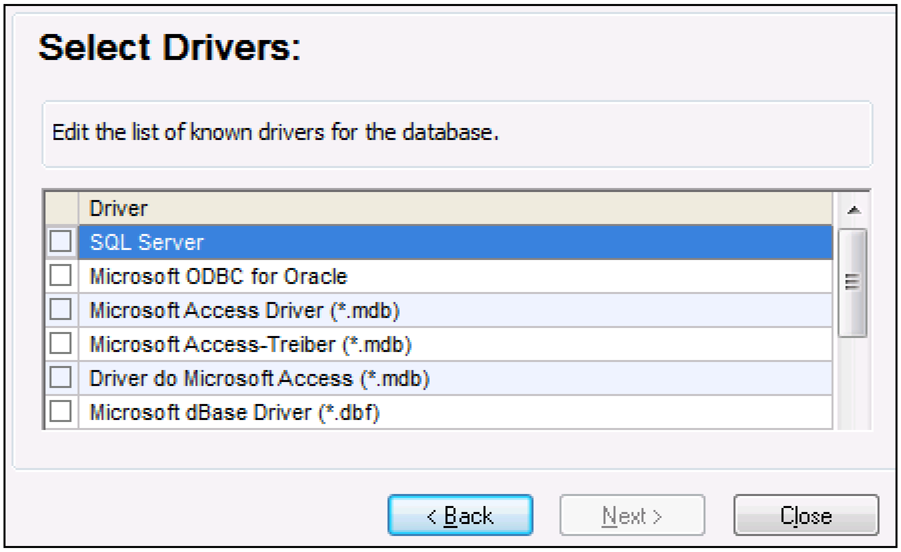
This example illustrates how to connect to an IBM DB2 database via ODBC.

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 192).
- Create a database alias. There are several ways to do this:
  - From IBM DB2 Configuration Assistant
  - From IBM DB2 Command Line Processor
  - 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.

Connection

1. Start the database connection wizard 170, select **Connection Wizard** and then *IBM DB2 (ODBC/JDBC)*. Click **Next**.
2. Select *ODBC* and click **Next**.
3. If prompted to edit the list of known drivers for the database, select the database drivers applicable to IBM DB2 (see *Prerequisites* above) and click **Next**.

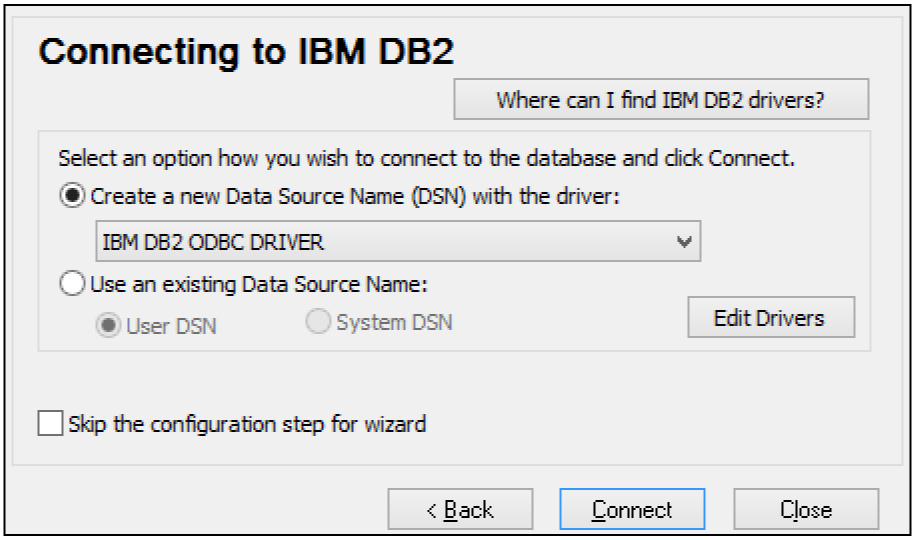


Select Drivers:

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)

4. Select the IBM DB2 driver you want 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:

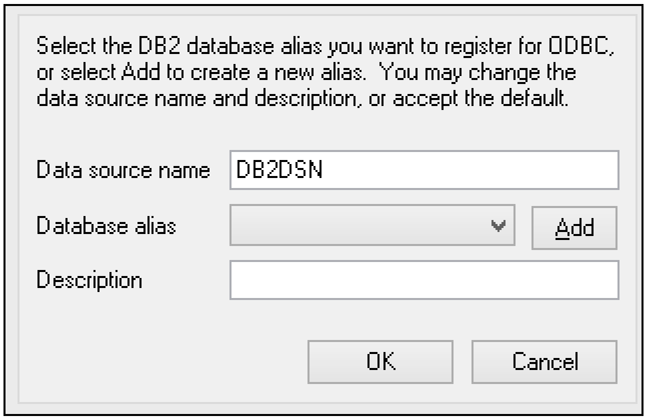
IBM DB2 ODBC DRIVER

○ Use an existing Data Source Name:

● User DSN ○ System DSN

□ Skip the configuration step for wizard

5. Enter a data source name (in the screenshot below, DB2DSN), and then click **Add**.

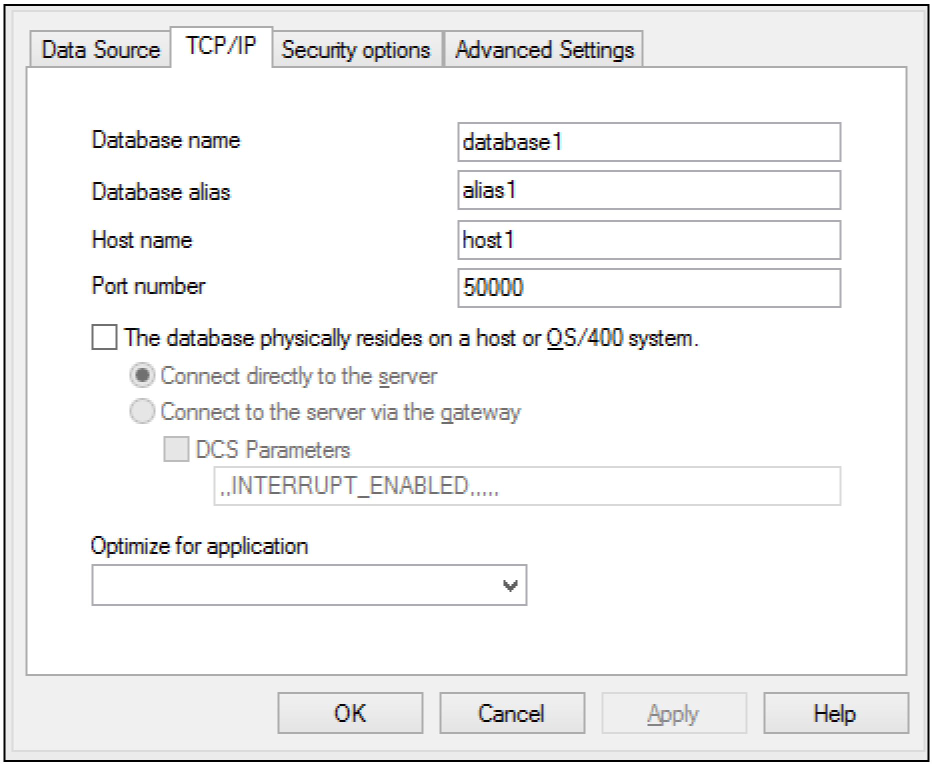

Select the DB2 database alias you want to register for ODBC, or select Add to create a new alias. You may change the data source name and description, or accept the default.

Data source name: DB2DSN

Database alias:

Description:

6. On the *Data Source* tab, enter the user name and password for the database.
7. On the *TCP/IP* tab, enter the database name, a name for the alias, the host name, and the port number, and then click **OK**.


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 the username and password again, and then click **OK**.

4.2.1.10.5 IBM DB2 for i (JDBC)

This example illustrates how to connect to an IBM DB2 for i database server via JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings[1082]; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. This example uses the open source **Toolbox for Java/JTOpen** version 9.8 (http://jt400.sourceforge.net/). After you download the package and unpack to a local directory, the required .jar files will be available in the **lib** subdirectory.
- You have the following database connection details: host, port, database path, name, or alias, username, and password.

Connection

1. Start the database connection wizard[170] and click **JDBC Connections** (*see JDBC Connection[187] for a screenshot of the dialog*).
2. In the *Classpaths* field, enter the path to the `.jar` file that provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of `.jar` file paths. If you have added the filepath to the CLASSPATH[189] of the system, you can leave this field empty.
3. In the *Driver* field, select the appropriate driver. Relevant drivers will be available only if a valid `.jar` file path is in the *Classpaths* field or in the `CLASSPATH` environment variable.
4. Enter the username and password for the database in the corresponding fields.
5. Enter the JDBC connection string in the *Database URL* field according to the pattern in the table below, replacing the highlighted values with the ones applicable to your database server.
6. Click **Connect**.

Connection details of the IBM DB2 for i example

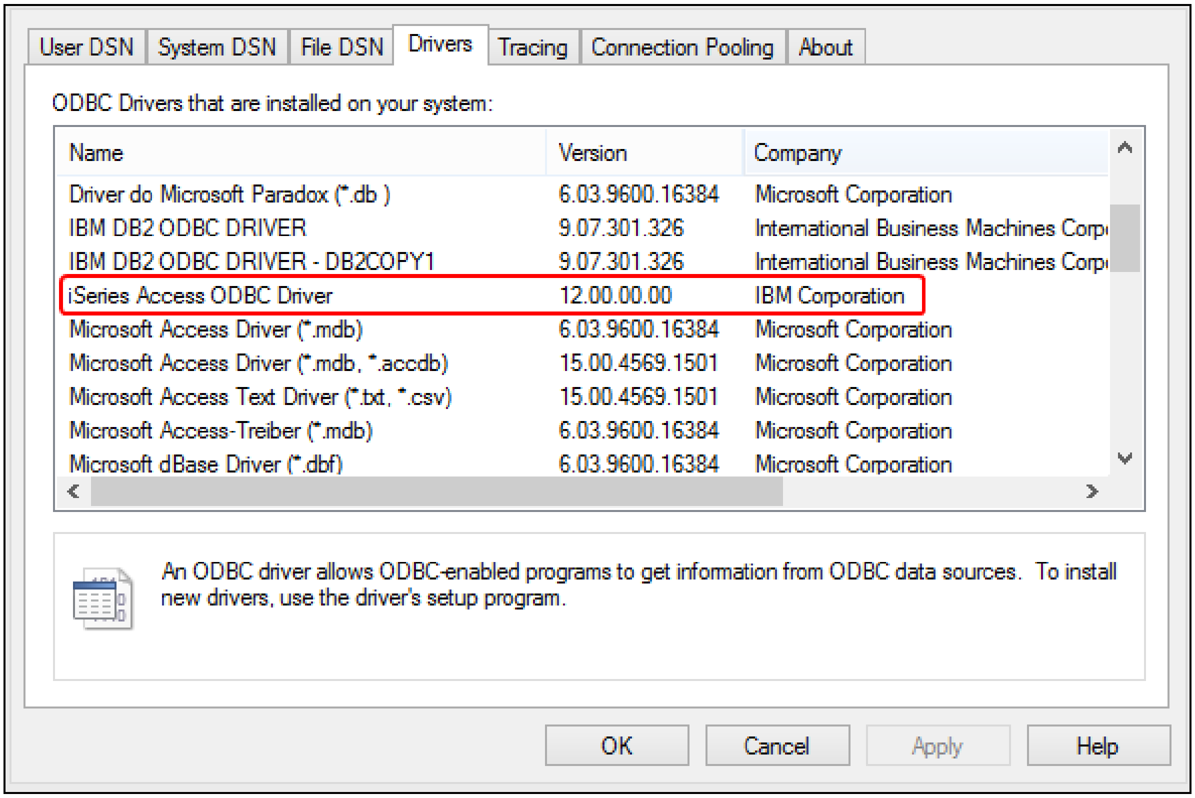
Field	Value
Classpaths	`C:\jdbc\jtopen_9_8\jt400.jar`
Driver	*com.ibm.as400.access.AS400JDBCDriver*
Database URL	`jdbc:as400://`**`host`**`[:`**`<port>`**`]/`**`<database path or alias>`**

4.2.1.10.6 IBM DB2 for i (ODBC)

This example illustrates how to connect to an *IBM DB2 for i* database via ODBC.

Prerequisites

- *IBM System i Access for Windows* must be installed on your operating system. 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[192].



ODBC Drivers that are installed on your system:

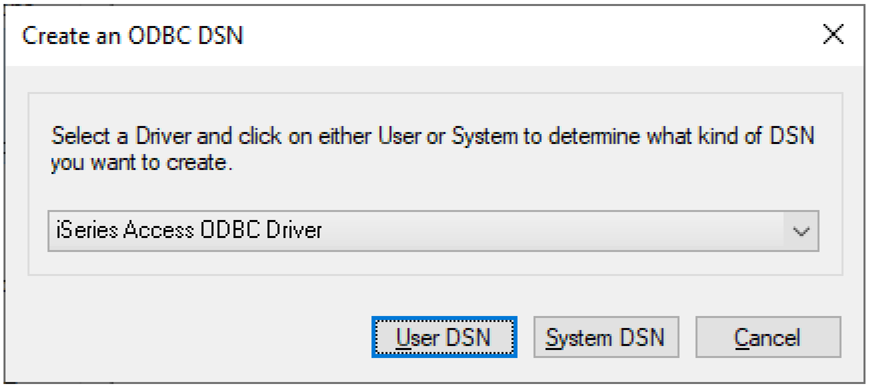
Name	Version	Company
Driver do Microsoft Paradox (*.db)	6.03.9600.16384	Microsoft Corporation
IBM DB2 ODBC DRIVER	9.07.301.326	International Business Machines Corp
IBM DB2 ODBC DRIVER - DB2COPY1	9.07.301.326	International Business Machines Corp
iSeries Access ODBC Driver	12.00.00.00	IBM Corporation
Microsoft 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 IP address of the database server, and the 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 IP address of the database server. After creating the connection, verify the connection by click it and selecting **File | Diagnostics | Verify Connection**.

Connection

1. Start the database connection wizard[170] and click **ODBC Connections**.
2. Click *User DSN*. and click **Create a New DSN** ✚ .
3. Select *iSeries Access ODBC Driver* from the list, and click **User DSN** (or **System DSN** if applicable).

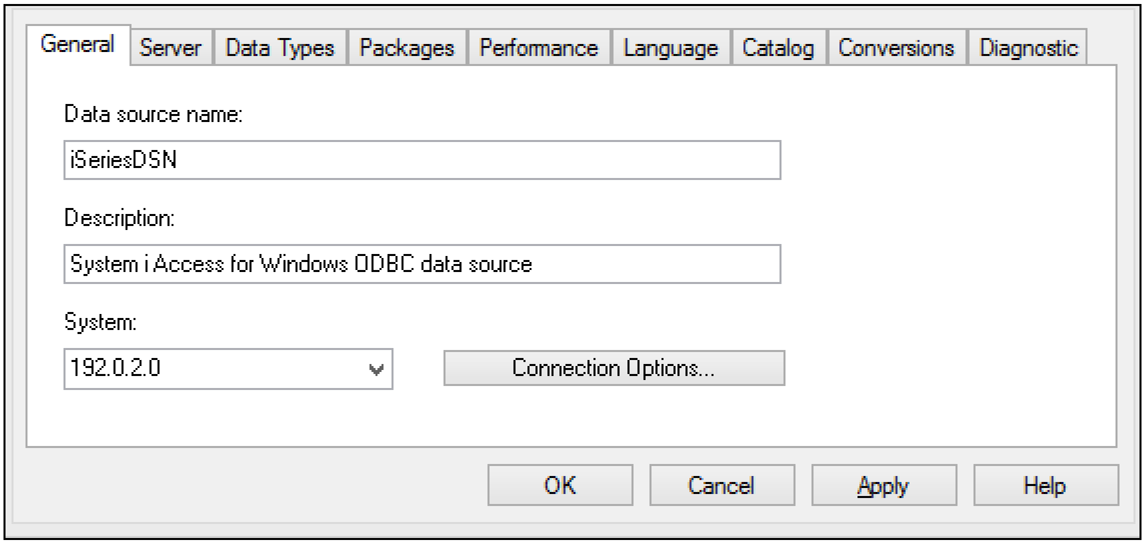


Create an ODBC DSN

Select a Driver and click on either User or System to determine what kind of DSN you want to create.

iSeries Access ODBC Driver

4. Enter a data source name and select the connection from the System combo box. In this example, the data source name is iSeriesDSN and the System is 192.0.2.0.Note that when an ODBC data source for an IBM DB2 for i database is added, 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.

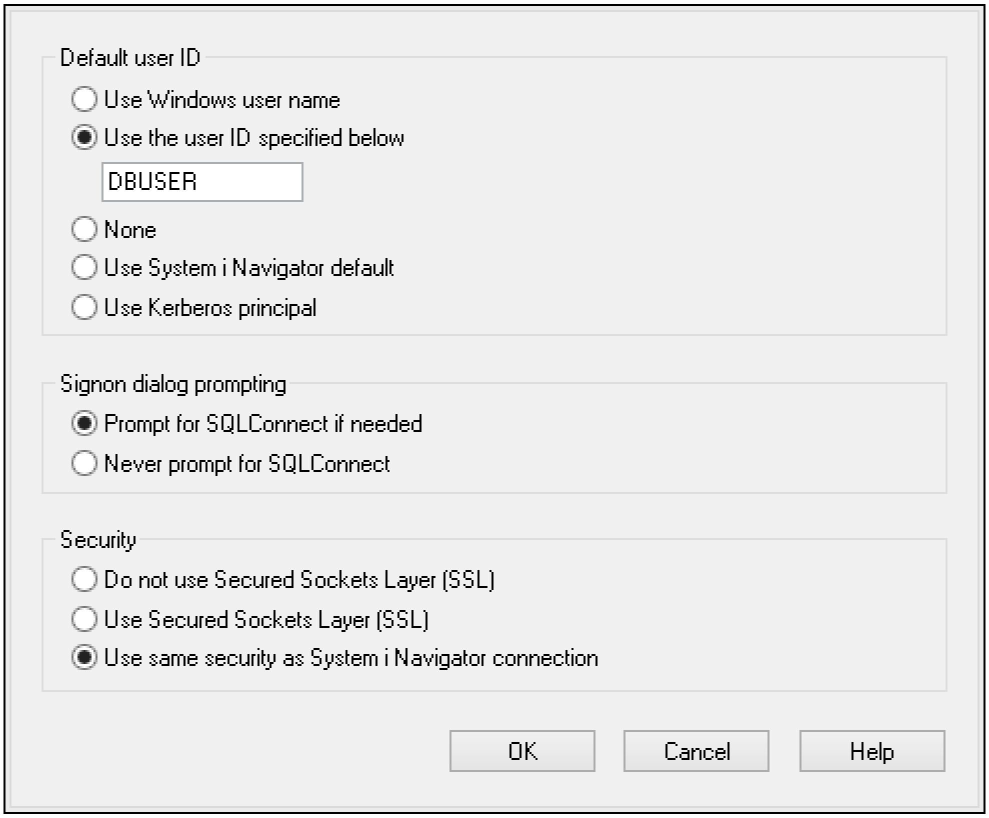


Data source name: iSeriesDSN

Description: System i Access for Windows ODBC data source

System: 192.0.2.0

5. 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

6. Click **OK**. The new data source becomes available in the list of DSNs.
7. Click **Connect**.
8. Enter the user name and password to the database when prompted, and then click **OK**.

4.2.1.10.7 IBM Informix (JDBC)

This example illustrates how to connect to an IBM Informix database server via JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings 1082; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or several .jar files that provide connectivity to the database) must be available on your operating system. In this example, IBM Informix JDBC driver version 3.70 is used. For the driver's installation instructions, see the documentation accompanying the driver or the "IBM Informix JDBC Driver Programmer's Guide").
- You have the following database connection details: host, name of the Informix server, database, port, username, and password.

Connection

1. Start the database connection wizard[170] and click **JDBC Connections** (*see JDBC Connection*[187] *for a screenshot of the dialog*).
2. In the *Classpaths* field, enter the path to the `.jar` file that provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of `.jar` file paths. If you have added the filepath to the CLASSPATH[189] of the system, you can leave this field empty.
3. In the *Driver* field, select the appropriate driver. Relevant drivers will be available only if a valid `.jar` file path is in the *Classpaths* field or in the `CLASSPATH` environment variable.
4. Enter the username and password for the database in the corresponding fields.
5. Enter the JDBC connection string in the *Database URL* field according to the pattern in the table below, replacing the highlighted values with the ones applicable to your database server.
6. Click **Connect**.

Connection details of the IBM Informix example

Field	**Value**
Classpaths	`C:\Informix_JDBC_Driver\lib\ifxjdbc.jar`
Driver	*com.informix.jdbc.IfxDriver*
Database URL	`jdbc:informix-sqli://`**`hostName`**`:`**`port`**`/`**`databaseName`**`:INFORMIXSERVER=`**`myserver`**`;`

4.2.1.10.8 MariaDB (ODBC)

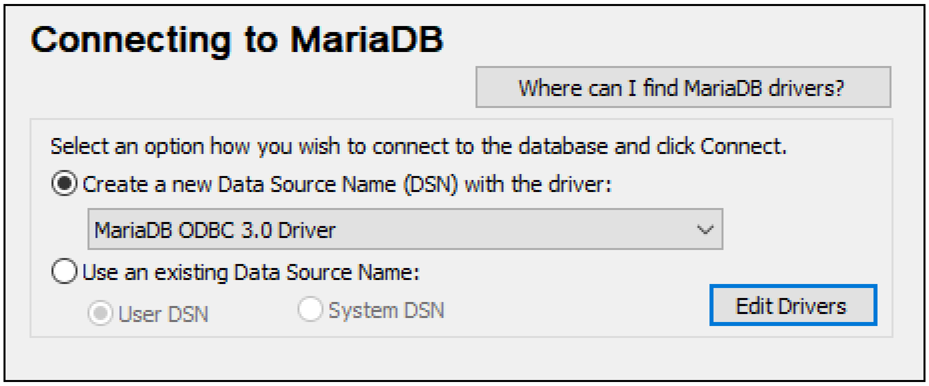
This example illustrates how to connect to a MariaDB database server via ODBC.

Prerequisites

- The MariaDB ODBC Connector must be installed.
- You have the following database connection details: host, database, port, username, and password.

Connection

1. Start the database connection wizard[170], select **Connection Wizard** and then *MariaDB (ODBC/Native)*. Click **Next**.
2. 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 driver from the the list of ODBC drivers installed on your system.) Click **Connect**. to start the New Data Source Wizard.



Connecting to MariaDB

Select an option how you wish to connect to the database and click Connect.

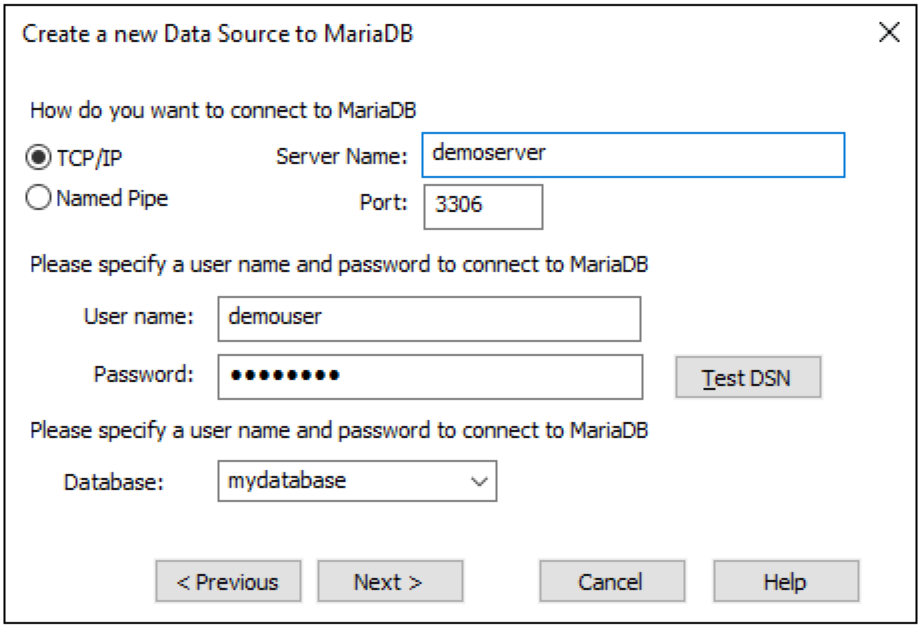
● Create a new Data Source Name (DSN) with the driver:

MariaDB ODBC 3.0 Driver

○ Use an existing Data Source Name:

● User DSN ○ System DSN

3. In the first screen of the wizard, enter a name for the data source, optionally, a description to help you identify this ODBC data source in the future, and click **OK**.
4. In the next screen (screenshot below), fill in the database connection credentials (TCP/IP Server, User Name, and Password), select a database, and click **Test DSN**.



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

5. Upon successful connection, a message to that effect is displayed. Click **Next** and complete the wizard. Other parameters may be required, for example, SSL certificates if you are connecting to MariaDB via a secure connection.

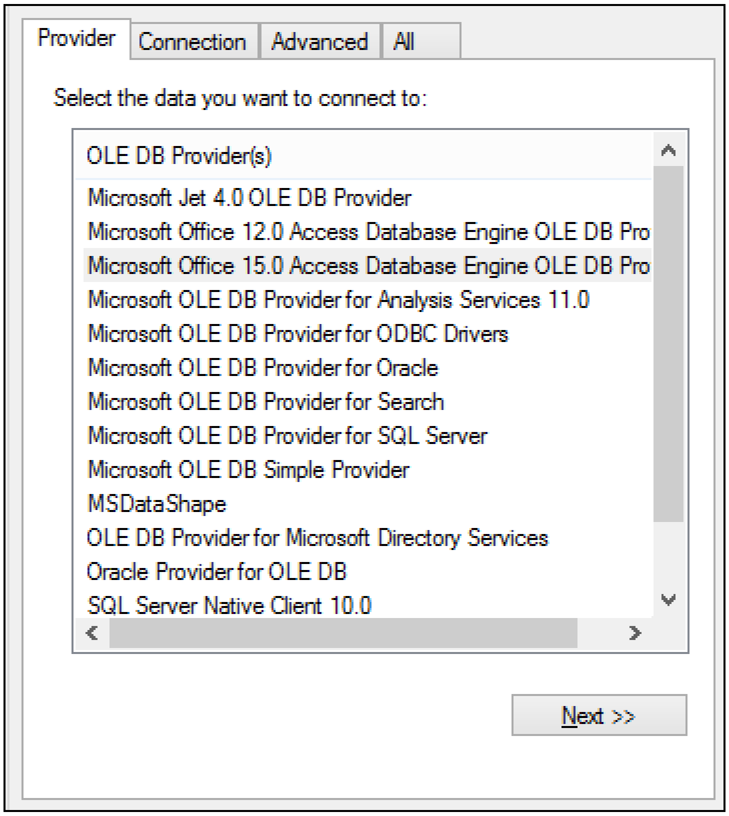
Note: If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address.

4.2.1.10.9 Microsoft Access (ADO)

A simple way to connect to a Microsoft Access database is to follow the wizard and browse for the database file, as shown in Connecting to an Existing Microsoft Access Database[179]. An alternative approach is to set up an ADO connection explicitly, as shown in this topic. This approach is useful if your database is password-protected. (It is also possible to connect to Microsoft Access through an ODBC connection, but it has limitations, so it is best to avoid it.)

Connection

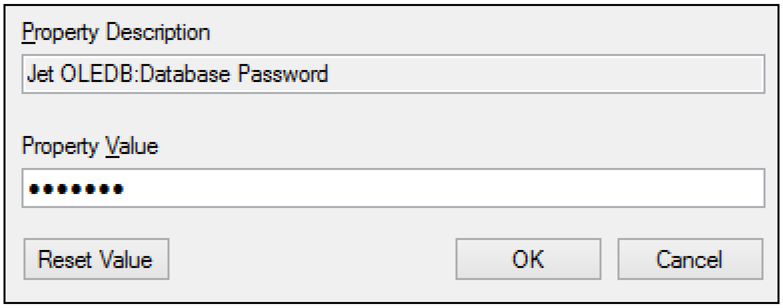
1. Start the database connection wizard 170.
2. Click **ADO Connections**, then click **Build**.
3. In the dialog that appears (*screenshot below*), go to the *Provider* tab, select *Microsoft Office 15.0 Access Database Engine OLE DB Provider*, and then click **Next**.



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

4. Go to the dialog's *Connection* tab, and in the *Data Source* field, enter the path to the Microsoft Access file in UNC format, for example, `\\myserver\\mynetworkshare\Reports\Revenue.accdb`, where `myserver` is the name of the server and `mynetworkshare` is the name of the network share.
5. On the *All* tab, double click the *Jet OLEDB:Database Password* property and enter the database password as property value (*see screenshot below*)



Property Description: Jet OLEDB:Database Password

Property Value: •••••••

If you are still unable to connect, locate the workgroup information file (`System.MDW`) applicable to your user profile, and set the value of the *Jet OLEDB: System database* property to the path of the `System.MDW` file.

4.2.1.10.10 Microsoft Azure SQL (ODBC)

In order to connect properly to an Azure SQL database, you must install the latest SQL Server Native Client.

For information about connecting to an Azure SQL database in the cloud, see this Altova blog entry.

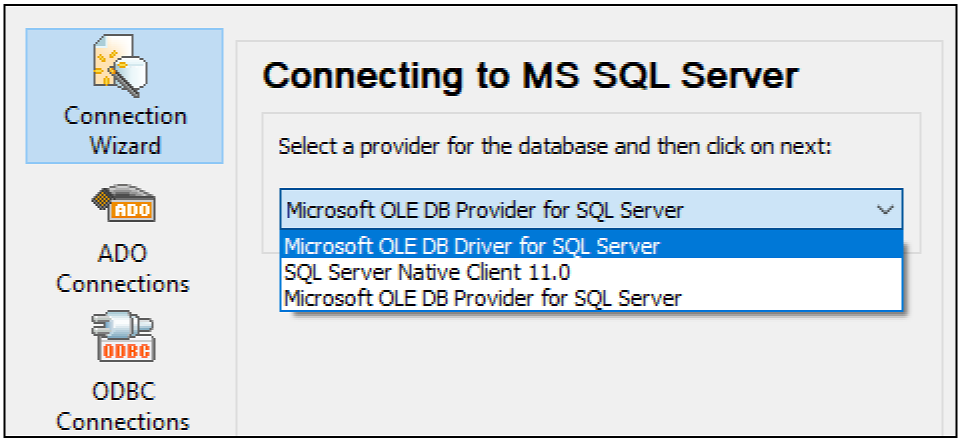
4.2.1.10.11 Microsoft SQL Server (ADO)

This example illustrates how to connect to a SQL Server database through ADO. These instructions work with the recommended Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL) that matches your MapForce platform (32-bit or 64-bit). For other ADO providers, the instructions would be similar but you may need to set additional connection properties as described in Setting up the SQL Server Data Link Properties 179.

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

Connection

1. Start the database connection wizard 170, select **Connection Wizard** and then *Microsoft SQL Server (ADO).* Click **Next**.
2. A list of available ADO providers is displayed (*screenshot below*). In this example, we use *Microsoft OLE DB Driver for SQL Server*. If it's not in the list, make sure that it is installed on your computer.



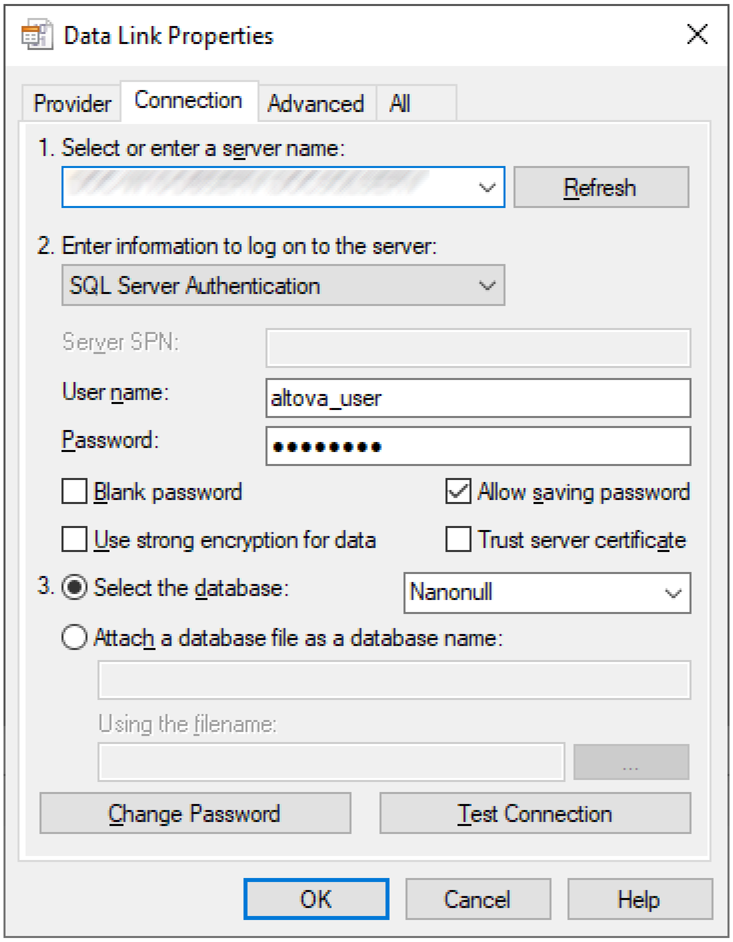
Connecting to MS SQL Server

Select a provider for the database and then click on next:

Microsoft OLE DB Provider for SQL Server

Microsoft OLE DB Driver for SQL Server
SQL Server Native Client 11.0
Microsoft OLE DB Provider for SQL Server

3. Click **Next**. The Data Link Properties dialog appears (*screenshot below*).



Data Link Properties

1. Select or enter a server name: [redacted] Refresh

2. Enter information to log on to the server:

SQL Server Authentication

Server SPN:

User name: altova_user

Password: ●●●●●●●●●

□ Blank password ☑ Allow saving password

□ Use strong encryption for data □ Trust server certificate

3. ● Select the database: Nanonull

○ Attach a database file as a database name:

Using the filename:

4. In the *Connection* tab, select or enter the name of the database server, for example, *SQLSERV01*. If you are connecting to a named SQL Server instance, the server name will be something like *SQLSERV01\SOMEINSTANCE*.
5. If the database server was configured to allow connections from users authenticated on the Windows domain, select Windows Authentication. Otherwise, select SQL Server Authentication, clear the Blank password check box, and enter the database credentials in the relevant boxes.
6. Select the *Allow saving password* check box and the database to which you are connecting (in this example, *Nanonull*).
7. You can test the connection by clicking **Test Connection**.
8. Click **OK** when done.

4.2.1.10.12 Microsoft SQL Server (ODBC)

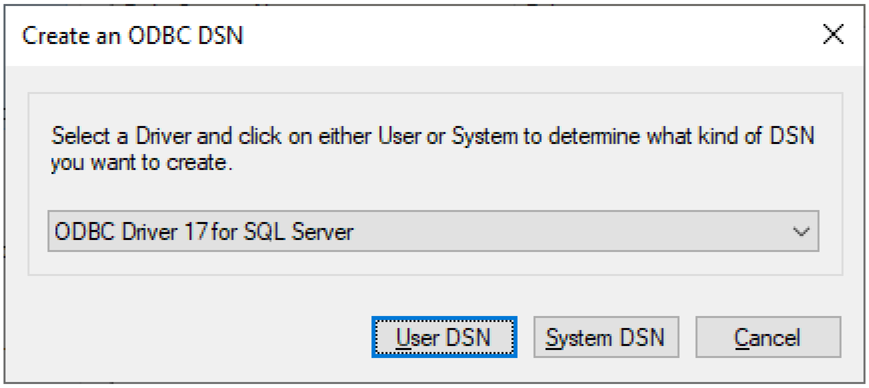
This example illustrates how to connect to a SQL Server database via ODBC.

Prerequisites

Download and install the Microsoft ODBC Driver for SQL Server from the Microsoft website. This example uses Microsoft ODBC Driver 17 for SQL Server to connect to a SQL Server 2016 database. You might need a different ODBC driver version, depending on your SQL Server version. For information about compatibility, see the driver's system requirements.

Connection

1. Start the database connection wizard[170] and click **ODBC Connections**.
2. Select *User DSN* (or *System DSN* if you have administrative privileges), and then click **Create a New DSN** .
3. Select the driver from the list. Note that the driver appears in the list only after it has been installed.

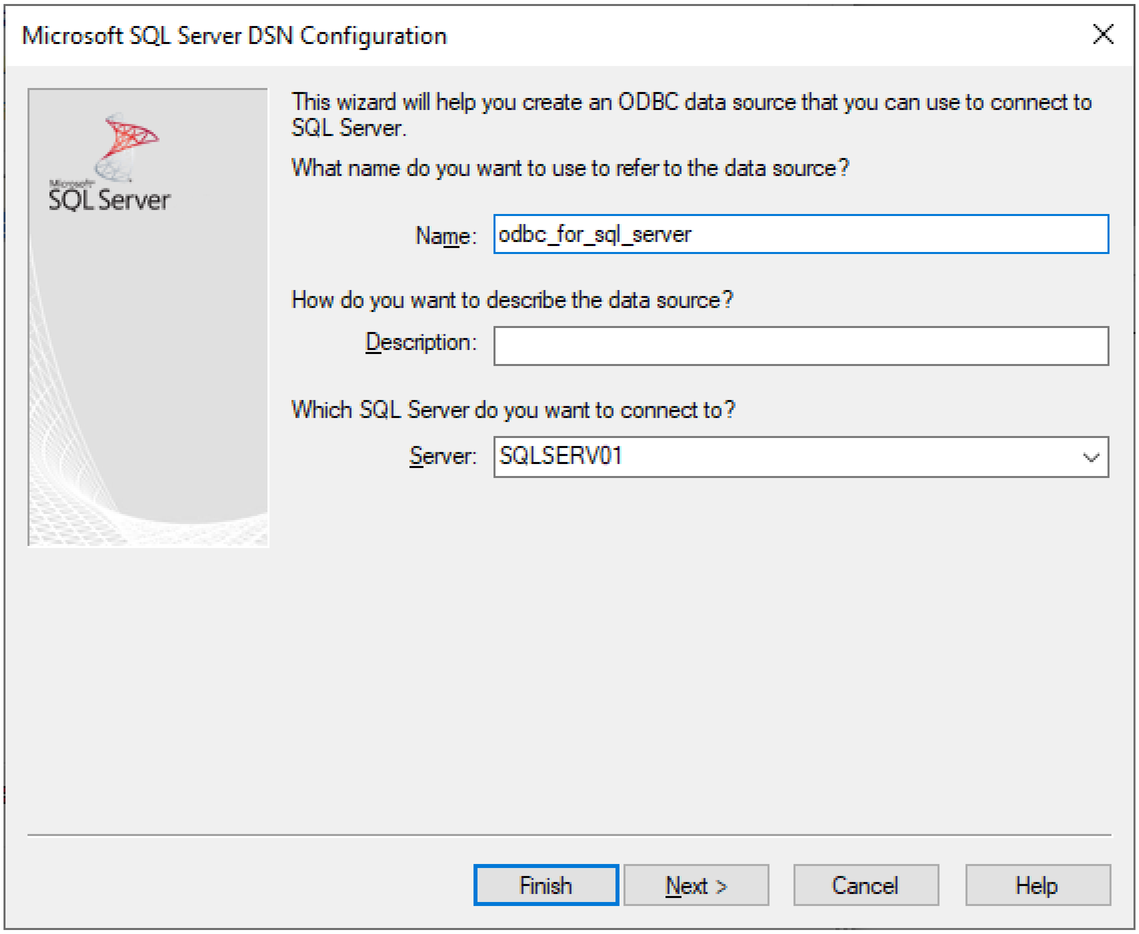


Create an ODBC DSN

Select a Driver and click on either User or System to determine what kind of DSN you want to create.

ODBC Driver 17 for SQL Server

4. Click *User DSN* (or *System DSN* if you are creating a System DSN). Creating a System DSN requires that MapForce be run as an administrator. Therefore, in order to create a System DSN, cancel the wizard, make sure that you run MapForce as an administrator, and perform the steps above again.
5. Enter a name for the data source, enter, optionally, a description to identify this connection, and select from the list the SQL Server to which you are connecting. Click **Next** to go to the next screen.



Microsoft SQL Server DSN Configuration

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: odbc_for_sql_server

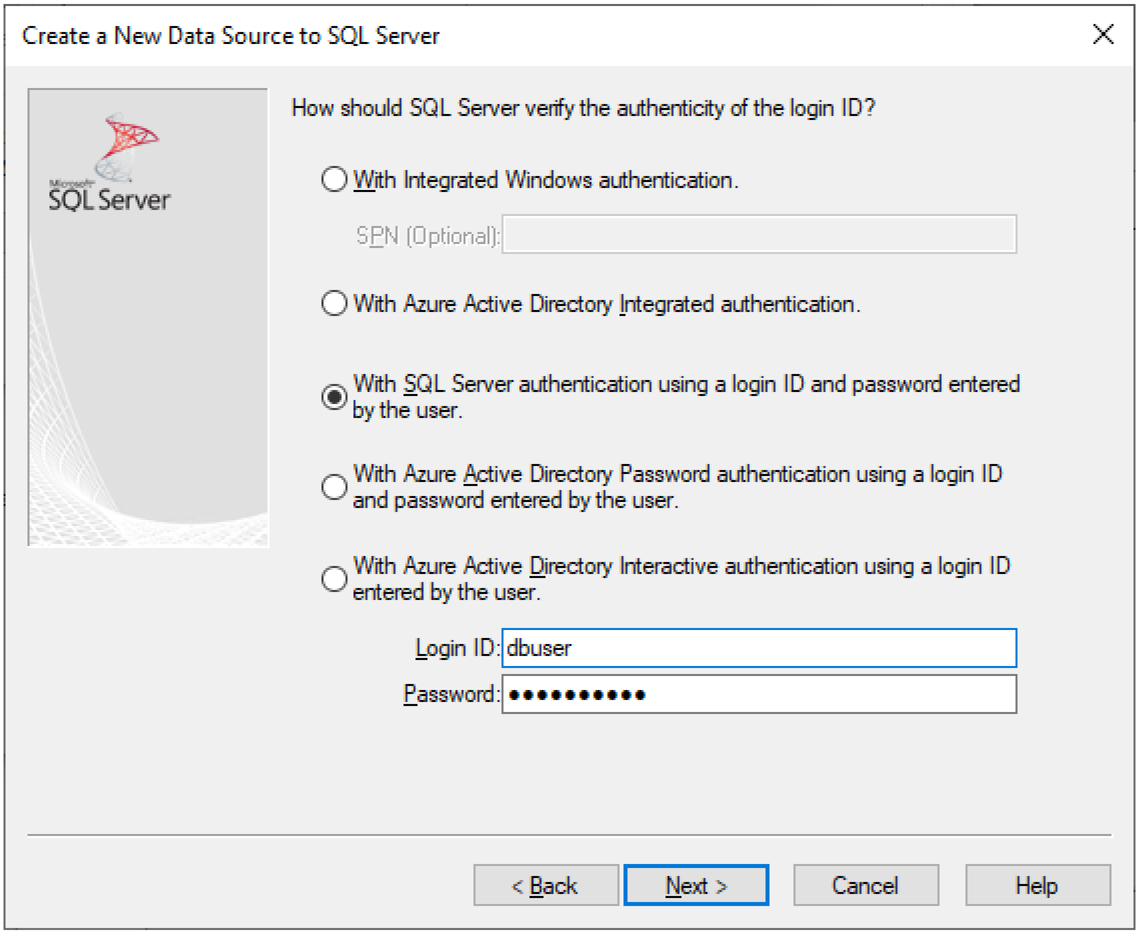
How do you want to describe the data source?

Description:

Which SQL Server do you want to connect to?

Server: SQLSERV01

6. In the next screen, select the authentication method to use. If the database server was configured to allow connections from users authenticated on the Windows domain, select *With Integrated Windows authentication*. Otherwise, select one of the other options, as applicable. This example uses *With SQL Server Authentication*, which requires that the user name and password be entered in the relevant boxes. Click **Next** when done.



Create a New Data Source to SQL Server

How should SQL Server verify the authenticity of the login ID?

○ With Integrated Windows authentication.

SPN (Optional):

○ With Azure Active Directory Integrated authentication.

● With SQL Server authentication using a login ID and password entered by the user.

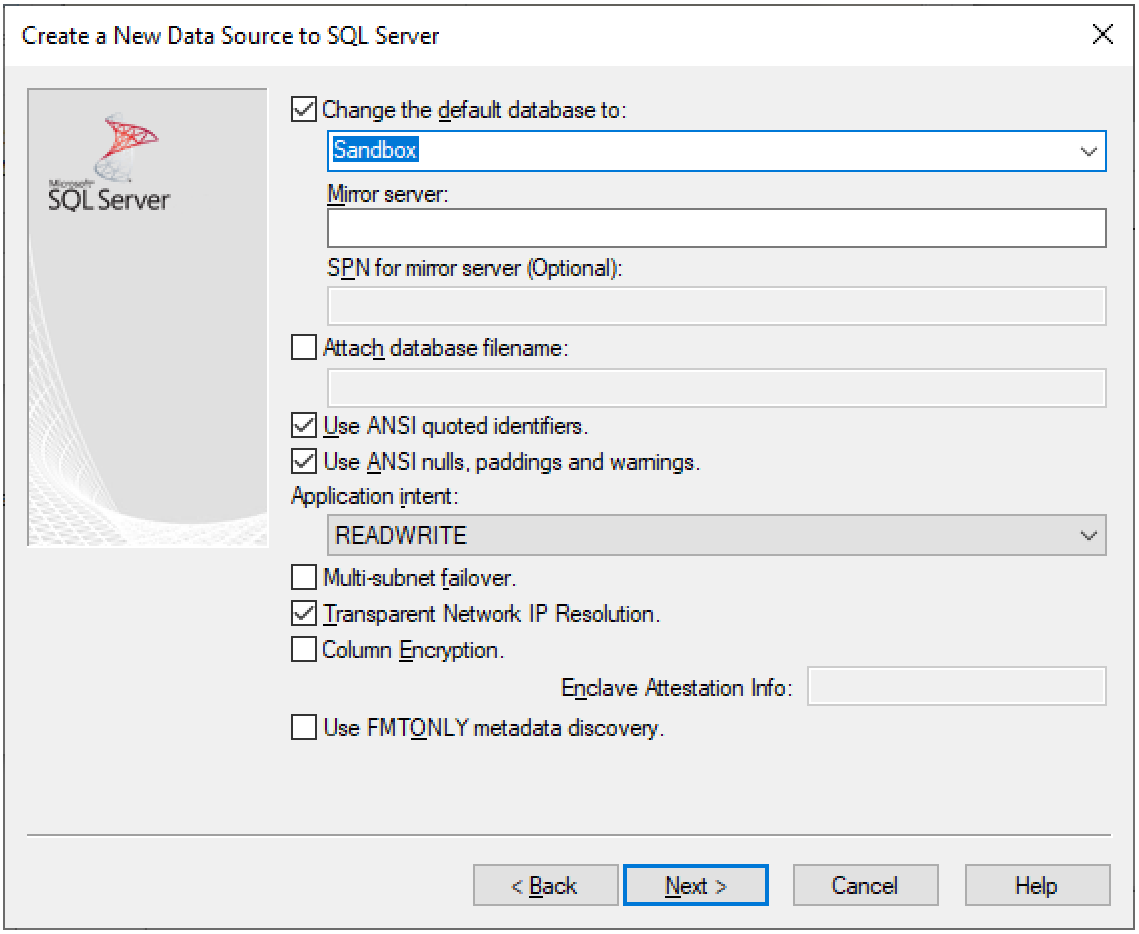
○ With Azure Active Directory Password authentication using a login ID and password entered by the user.

○ With Azure Active Directory Interactive authentication using a login ID entered by the user.

Login ID: dbuser

Password: ●●●●●●●●●●●

7. In the next screen of our example (*screenshot below*), we have selected *Change the default database* and entered *Sandbox* (the name of the database to which to connect). Click **Next** after you have entered the settings you want.



Create a New Data Source to SQL Server

☑ Change the default database to:

Sandbox

Mirror server:

SPN for mirror server (Optional):

□ Attach database filename:

☑ Use ANSI quoted identifiers.

☑ Use ANSI nulls, paddings and warnings.

Application intent:

READWRITE

□ Multi-subnet failover.

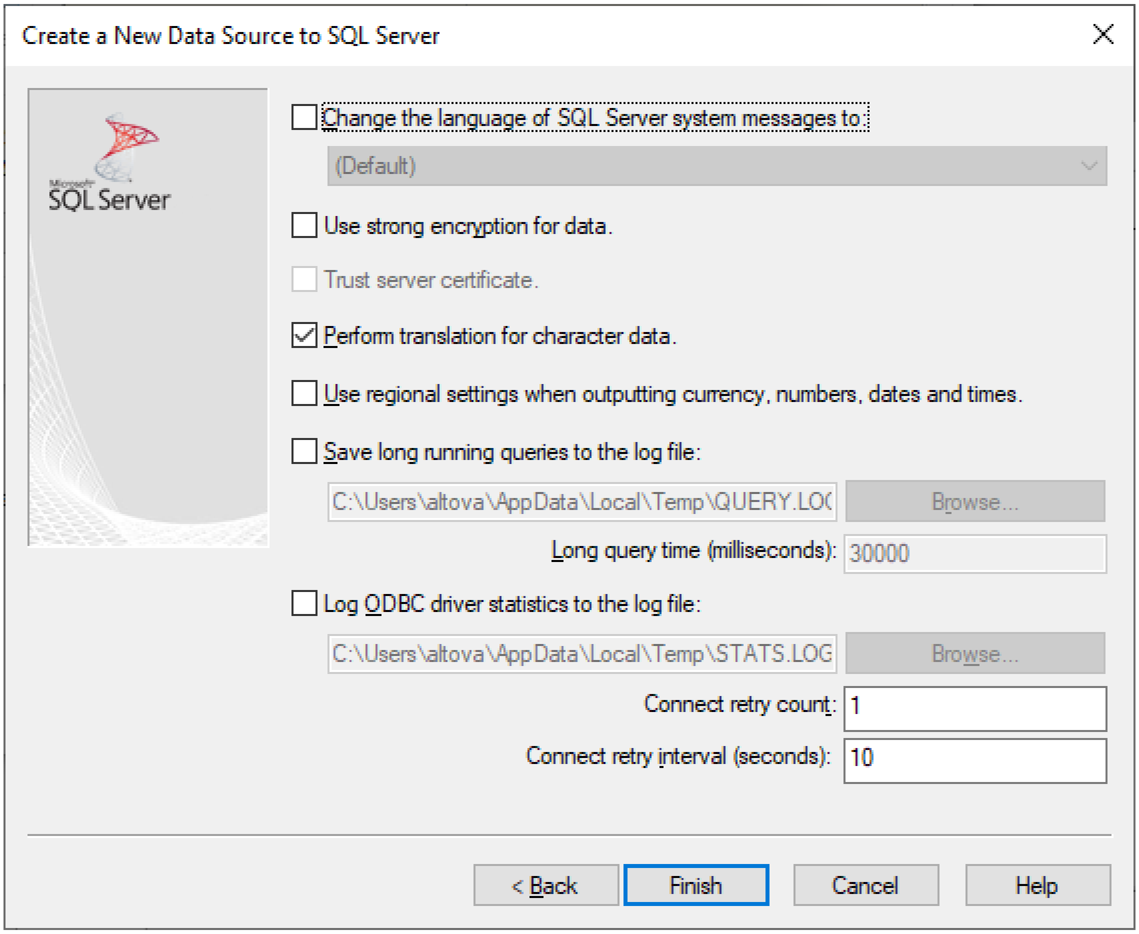
☑ Transparent Network IP Resolution.

□ Column Encryption.

Enclave Attestation Info:

□ Use FMTONLY metadata discovery.

8. Click **Next** and, optionally, configure additional parameters for this connection. Click **Finish** when done.



Create a New Data Source to SQL Server

☐ Change the language of SQL Server system messages to:

(Default)

☐ Use strong encryption for data.

☐ Trust server certificate.

☑ Perform translation for character data.

☐ Use regional settings when outputting currency, numbers, dates and times.

☐ Save long running queries to the log file:

C:\Users\altova\AppData\Local\Temp\QUERY.LOG

Long query time (milliseconds): 30000

☐ Log ODBC driver statistics to the log file:

C:\Users\altova\AppData\Local\Temp\STATS.LOG

Connect retry count: 1

Connect retry interval (seconds): 10

9. A confirmation dialog box listing the connection details opens. Click **OK**.
10. The data source now appears in the list of *User* or *System* data sources.

4.2.1.10.13 MySQL (ODBC)

This example illustrates how to connect to a MySQL database server from a Windows machine via the MySQL ODBC, which driver must be downloaded and installed separately. This example uses MySQL Connector/ODBC 8.0.

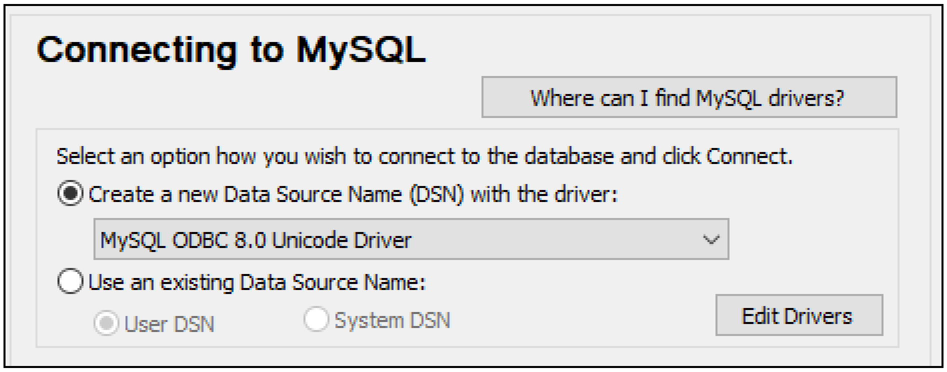
Prerequisites

- MySQL ODBC driver must be installed on your operating system. Check the MySQL documentation for the driver version recommended for your database server version.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the MySQL ODBC driver.
- You have the following database connection details: host, database, port, username, and password.

Connection

1. Start the database connection wizard 170, select **Connection Wizard** and then *MySQL (ODBC/Native)*. Click **Next**.

2. Select *Create a new Data Source Name (DSN) with the driver*, and select a MySQL driver. (If no such driver is available in the list, click **Edit Drivers**, and select any available MySQL driver from the the list of ODBC drivers installed on your system. See also Viewing the Available ODBC Drivers 192.)


Connecting to MySQL

Select an option how you wish to connect to the database and click Connect.

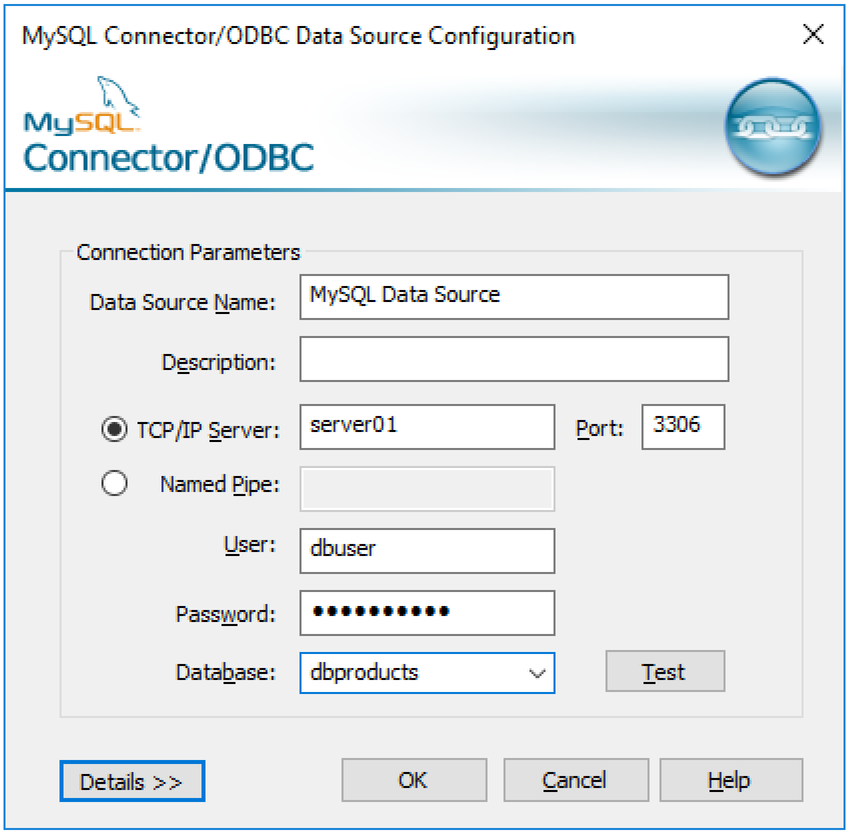
● Create a new Data Source Name (DSN) with the driver:

MySQL ODBC 8.0 Unicode Driver

○ Use an existing Data Source Name:

○ User DSN ○ System DSN

3. Click **Connect**. to open the Data Source Configuration dialog (*screenshot below*).


MySQL Connector/ODBC Data Source Configuration

Connection Parameters

Data Source Name: MySQL Data Source

Description:

● TCP/IP Server: server01 Port: 3306

○ Named Pipe:

User: dbuser

Password: ●●●●●●●●●●●

Database: dbproducts

4. In the Data Source Name box, enter a descriptive name that will help you identify this ODBC data source in future. Optionally enter a description.
5. 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 the default values.

4.2.1.10.14 Oracle (JDBC)

This example shows how to connect to an Oracle database server 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.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings[1082]; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The Oracle Instant Client Package (Basic) must be available on your operating system. The package can be downloaded from the official Oracle website. This example uses Oracle Instant Client Package version 12.1.0.2.0, for Windows 32-bit and, consequently, Oracle JDK 32-bit.
- You have the following database connection details: host, port, service name, username, and password.

Connection

1. Start the database connection wizard[170] and click **JDBC Connections** (*see JDBC Connection*[187] *for a screenshot of the dialog*).
2. In the *Classpaths* field, enter the path to the `.jar` file that provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of `.jar` file paths. If you have added the filepath to the CLASSPATH[189] of the system, you can leave this field empty.
3. In the *Driver* field, select the appropriate driver. Relevant drivers will be available only if a valid `.jar` file path is in the *Classpaths* field or in the `CLASSPATH` environment variable.
4. Enter the username and password for the database in the corresponding fields.
5. Enter the JDBC connection string in the *Database URL* field according to the pattern in the table below, replacing the highlighted values with the ones applicable to your database server.
6. Click **Connect**.

Connection details of the Oracle example

Field	**Value**
Classpaths	`C:\jdbc\instantclient_12_1\ojdbc7.jar`
Driver	*oracle.jdbc.OracleDriver* or *oracle.jdbcdriver.OracleDriver*
Database URL	`jdbc:oracle:thin:@//`**`host`**`:`**`port`**`:`**`service`**

4.2.1.10.15 Oracle (ODBC)

This example shows how to connect from MapForce to an Oracle database server on a network machine, via an Oracle database client installed on the local operating system.

The example includes instructions for setting up an ODBC data source (DSN) using the database connection wizard in MapForce. If you have already created a DSN, or if you prefer to create it directly from the *ODBC Data Source Administrator* in Windows, you can do so, and then select it when prompted by the wizard. For more information about ODBC data sources, see Setting up an ODBC Connection[190].

Prerequisites

- The Oracle database client (which includes the ODBC Oracle driver) must be installed and configured on your system. For instructions, see the Oracle documentation.
- 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)
      )
    )
  ```

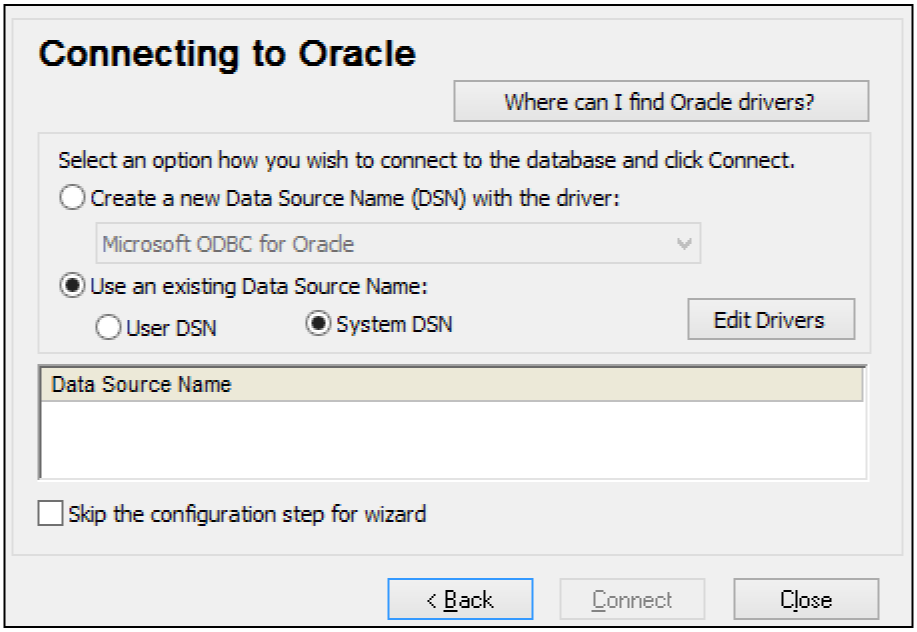
  For Oracle database client 11.2.0, the default Oracle home directory path could be as follows:

  ```
  C:\app\username\product\11.2.0\client_1\network\admin\tnsnames.ora
  ```

  You can add new entries to **tnsnames.ora**, either by pasting the connection details and saving the file, or by running the Oracle Net Configuration Assistant, if available. If you want these values to appear in dropdown lists during the configuration process, then you may need to add the path to the admin folder as a **TNS_ADMIN** environment variable.

Connection

1. Start the database connection wizard[170], select **Connection Wizard** and then *Oracle (ODBC/JDBC)*. Click **Next**.
2. Select *ODBC* and click **Next**.
3. In the dialog that appears (*screenshot below*), click **Edit Drivers**.



Connecting to Oracle

Where can I find Oracle drivers?

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

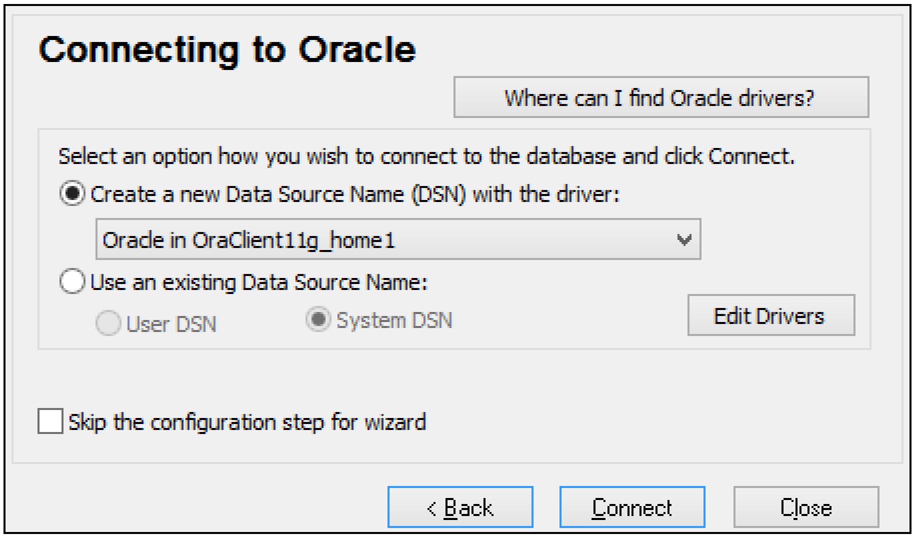
Edit Drivers

Data Source Name

□ Skip the configuration step for wizard

< Back Connect Close

4. From the list of Oracle drivers available on your system, select the Oracle driver you wish to use and click **Back**.
5. Select *Create a new data source name (DSN) with the driver*, and then select the Oracle driver chosen in step 4. (Avoid using the Microsoft-supplied driver called Microsoft ODBC for Oracle driver. Microsoft recommends using the ODBC driver provided by Oracle.)



Connecting to Oracle

Where can I find Oracle drivers?

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:

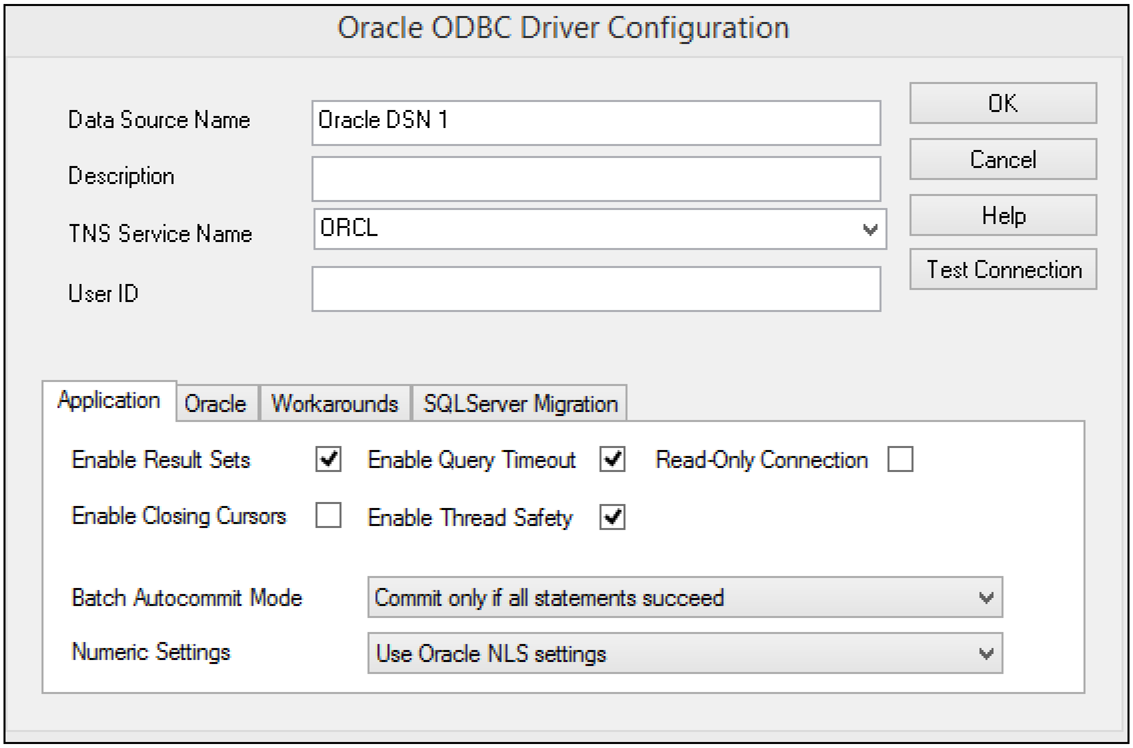
○ User DSN ● System DSN

Edit Drivers

□ Skip the configuration step for wizard

< Back Connect Close

6. Click **Connect**.
7. In the dialog that appears fill in the required fields (*shown filled in the screenshot below*). The connection name must be entered in the *TNS Service Name* field as it is defined in the `tnsnames.ora` file (see *Prerequisites* above). *Note:* If you wish to have the dropdown list of the combo box automatically filled with the values of the **`tnsnames.ora`** file, then you may need to add the path to the admin folder as a **`TNS_ADMIN`** environment variable.



Oracle ODBC Driver Configuration

Data Source Name: Oracle DSN 1

Description:

TNS Service Name: ORCL

User ID:

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

8. Click **OK** when done.
9. In the dialog that appears, enter the username and password to the database, and then click **OK**.

4.2.1.10.16 PostgreSQL (ODBC)

This example shows how to connect to a PostgreSQL database server from a Windows machine via a PostgreSQL ODBC driver installed on the local machine. This example uses the *psqlODBC* driver (version 11.0) downloaded from the official website (see also Database Drivers Overview[172]).

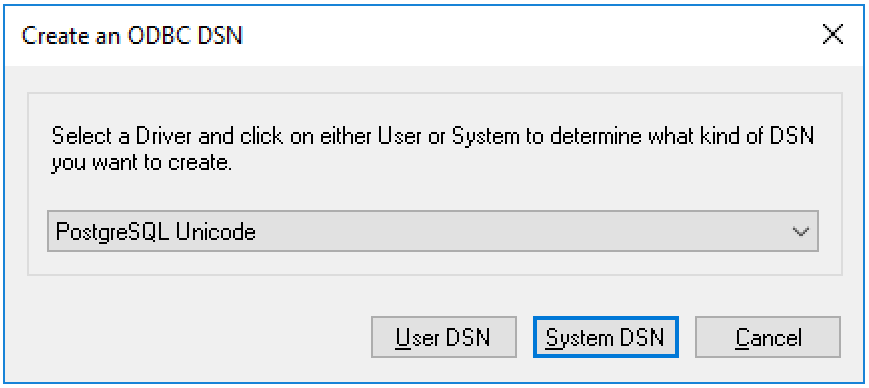
Note: You can also connect to a PostgreSQL database server directly (without the ODBC driver), see Native Connections[194].

Prerequisites

- *psqlODBC* driver must be installed on your system.
- You have the following database connection details: server, port, database, user name, and password.

Connection

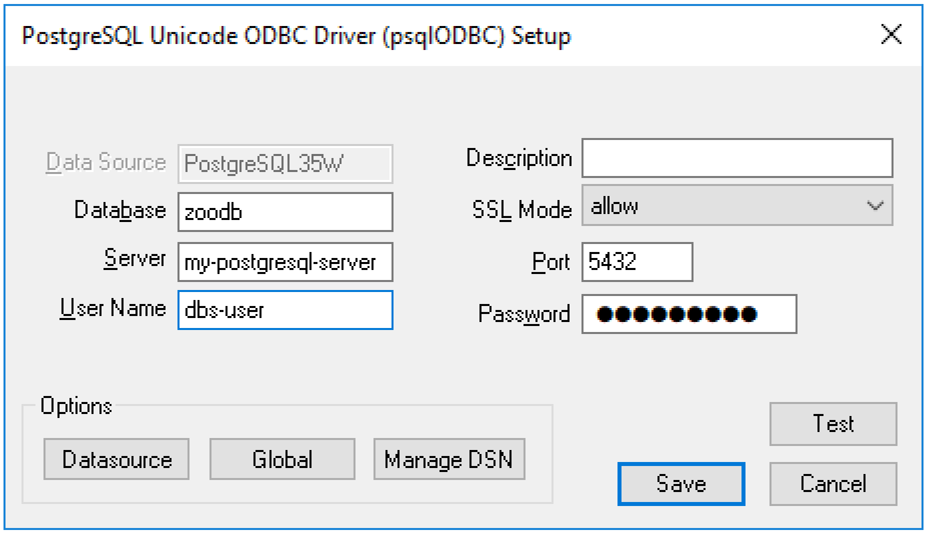
1. Start the database connection wizard[170] and click **ODBC Connections**.
2. Select *User DSN* (or *System DSN* if you have administrative privileges), and then click **Create a New DSN** ✚ .
3. Select the driver from the drop-down list (*screenshot below*) and click **User DSN**. (If no PostgreSQL driver is available in the list, make sure that the PostgreSQL ODBC driver is installed on your operating system.)



Select a Driver and click on either User or System to determine what kind of DSN you want to create.

PostgreSQL Unicode

4. In the dialog that appears (*screenshot below*), fill in the database connection (supplied by the database administrator).



Data Source: PostgreSQL35W
Database: zoodb
Server: my-postgresql-server
User Name: dbs-user
Description:
SSL Mode: allow
Port: 5432
Password: ●●●●●●●●●

5. Click **Save**.

The connection will become now available in the list of ODBC connections. To connect to the database, you can either double-click the connection or select it and then click **Connect**.

4.2.1.10.17 Progress OpenEdge (JDBC)

This example illustrates how to connect to a Progress OpenEdge 11.6 database server via JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings 1082; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.

- The operating system's PATH environment variable must include the path to the bin directory of the JRE or JDK installation directory, for example C:\Program Files (x86)\Java\jre1.8.0_51\bin.
- The Progress OpenEdge JDBC driver must be available on your operating system. In this example, JDBC connectivity is provided by the openedge.jar and pool.jar driver component files available in C:\Progress\OpenEdge\java as part of the OpenEdge SDK installation.
- You have the following database connection details: host, port, database path, name, or alias, username, and password.

Connection

1. Start the database connection wizard[170] and click **JDBC Connections** (*see JDBC Connection*[187] *for a screenshot of the dialog*).
2. In the *Classpaths* field, enter the path to the .jar file that provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of .jar file paths. If you have added the filepath to the CLASSPATH[189] of the system, you can leave this field empty.
3. In the *Driver* field, select the appropriate driver. Relevant drivers will be available only if a valid .jar file path is in the *Classpaths* field or in the CLASSPATH environment variable.
4. Enter the username and password for the database in the corresponding fields.
5. Enter the JDBC connection string in the *Database URL* field according to the pattern in the table below, replacing the highlighted values with the ones applicable to your database server.
6. Click **Connect**.

Connection details of the Progress OpenEdge example

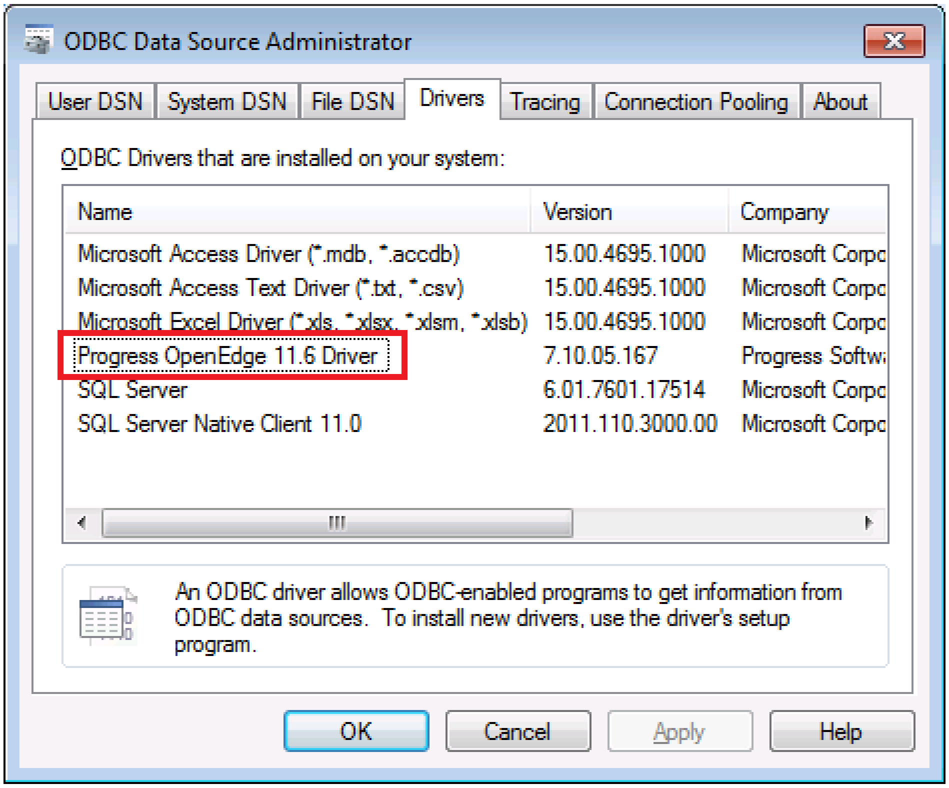
Field	**Value**
Classpaths	C:\Progress\OpenEdge\java\openedge.jar;C:\Progress\OpenEdge\java\pool.jar;
Driver	*com.ddtek.jdbc.openedge.OpenEdgeDriver*
Database URL	jdbc:datadirect:openedge://**host**:**port**;databaseName=**db_name**

4.2.1.10.18 Progress OpenEdge (ODBC)

This example shows how to connect to a Progress OpenEdge database server via the Progress OpenEdge 11.6 ODBC driver.

Prerequisites

- The *ODBC Connector for Progress OpenEdge* driver must be installed on your system (see also Database Drivers Overview[172]). Download the version that corresponds to your version of MapForce (32-bit or 64-bit). After installation, check if the ODBC driver is available on your machine[192].



ODBC Drivers that are installed on your system:

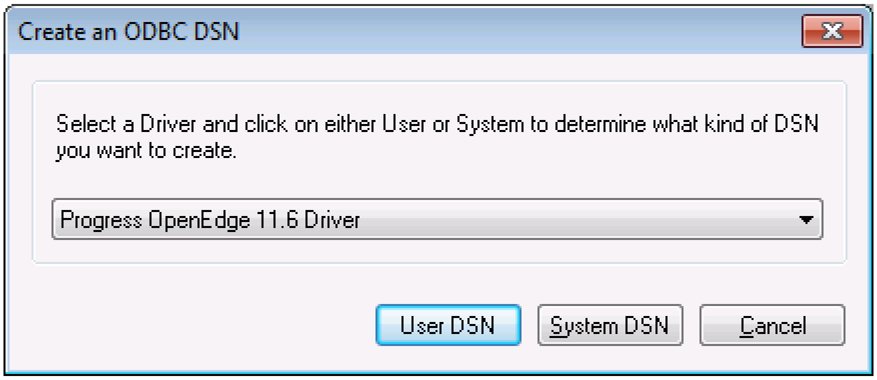
Name	Version	Company
Microsoft Access Driver (*.mdb, *.accdb)	15.00.4695.1000	Microsoft Corpc
Microsoft Access Text Driver (*.txt, *.csv)	15.00.4695.1000	Microsoft Corpc
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)	15.00.4695.1000	Microsoft Corpc
Progress OpenEdge 11.6 Driver	7.10.05.167	Progress 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.

Connection

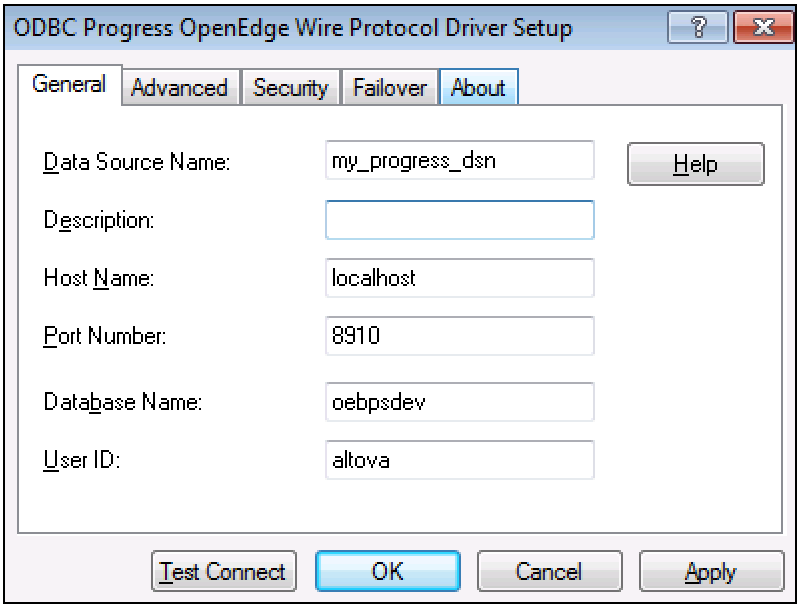
1. Start the database connection wizard 170 and click **ODBC Connections**.
2. Select *User DSN* (or *System DSN* if you have administrative privileges), and then click **Create a New DSN** .
3. Select *Progress OpenEdge Driver* from the drop-down list (*screenshot below*) and click **User DSN** (or **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

4. Fill in the database connection details (Database, Server, Port, User Name, Password), and click **OK**.



Data Source Name: my_progress_dsn
Description:
Host Name: localhost
Port Number: 8910
Database Name: oebpsdev
User ID: altova

5. To verify connectivity before saving the entered data, click **Test Connect**. To save, click **OK**.
6. The new data source now appears in the list of ODBC data sources. Click **Connect** to start the connection.

4.2.1.10.19 Sybase (JDBC)

This example illustrates how to connect to a Sybase database server via JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings 1082; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- Sybase *jConnect* component must be installed on your operating system (in this example, *jConnect 7.0* is used, installed as part of the *Sybase Adaptive Server Enterprise PC Client* installation).
- You have the following database connection details: host, port, database path, name, or alias, username, and password.

Connection

1. Start the database connection wizard 170 and click **JDBC Connections** (*see JDBC Connection 187 for a screenshot of the dialog*).
2. In the *Classpaths* field, enter the path to the `.jar` file that provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of `.jar` file paths. If you have added the filepath to the CLASSPATH 189 of the system, you can leave this field empty.
3. In the *Driver* field, select the appropriate driver. Relevant drivers will be available only if a valid `.jar` file path is in the *Classpaths* field or in the `CLASSPATH` environment variable.

4. Enter the username and password for the database in the corresponding fields.
5. Enter the JDBC connection string in the *Database URL* field according to the pattern in the table below, replacing the highlighted values with the ones applicable to your database server.
6. Click **Connect**.

Connection details of the Sybase example

Field	Value
Classpaths	`C:\sybase\jConnect-7_0\classes\jconn4.jar`
Driver	*com.sybase.jdbc4.jdbc.SybDriver*
Database URL	`jdbc:sybase:Tds:`**`hostName`**`:`**`port`**`/`**`databaseName`**

4.2.1.10.20 Teradata (JDBC)

This example illustrates how to connect to a Teradata database server via JDBC.

Prerequisites

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. This may be either Oracle JDK or an open source build such as Oracle OpenJDK. MapForce will determine the path to the Java Virtual Machine (JVM) from the following locations, in this order: (i) the custom JVM path you may have set in application **Options**; see Java Settings[1082]; (ii) the JVM path found in the Windows registry; (iii) the `JAVA_HOME` environment variable.
- Make sure that the platform of MapForce (32-bit, 64-bit) matches that of the JRE/JDK.
- The JDBC driver (one or more `.jar` files that provide connectivity to the database) must be available on your operating system. In this example, Teradata JDBC Driver 16.20.00.02 is used. For more information, see https://downloads.teradata.com/download/connectivity/jdbc-driver.
- You have the following database connection details: host, port, database path, name, or alias, username, and password.

Connection

1. Start the database connection wizard[170] and click **JDBC Connections** (*see JDBC Connection[187] for a screenshot of the dialog*).
2. In the *Classpaths* field, enter the path to the `.jar` file that provides connectivity to the database. If necessary, you can also enter a semicolon-separated list of `.jar` file paths. If you have added the filepath to the CLASSPATH[189] of the system, you can leave this field empty.
3. In the *Driver* field, select the appropriate driver. Relevant drivers will be available only if a valid `.jar` file path is in the *Classpaths* field or in the `CLASSPATH` environment variable.
4. Enter the username and password for the database in the corresponding fields.
5. Enter the JDBC connection string in the *Database URL* field according to the pattern in the table below, replacing the highlighted values with the ones applicable to your database server.
6. Click **Connect**.

Connection details of the Teradata example

Field	Value
Classpaths	`C:\jdbc\teradata\`
Driver	*com.teradata.jdbc.TeraDriver*
Database URL	`jdbc:teradata://`**`databaseServerName`**

4.2.1.10.21 Teradata (ODBC)

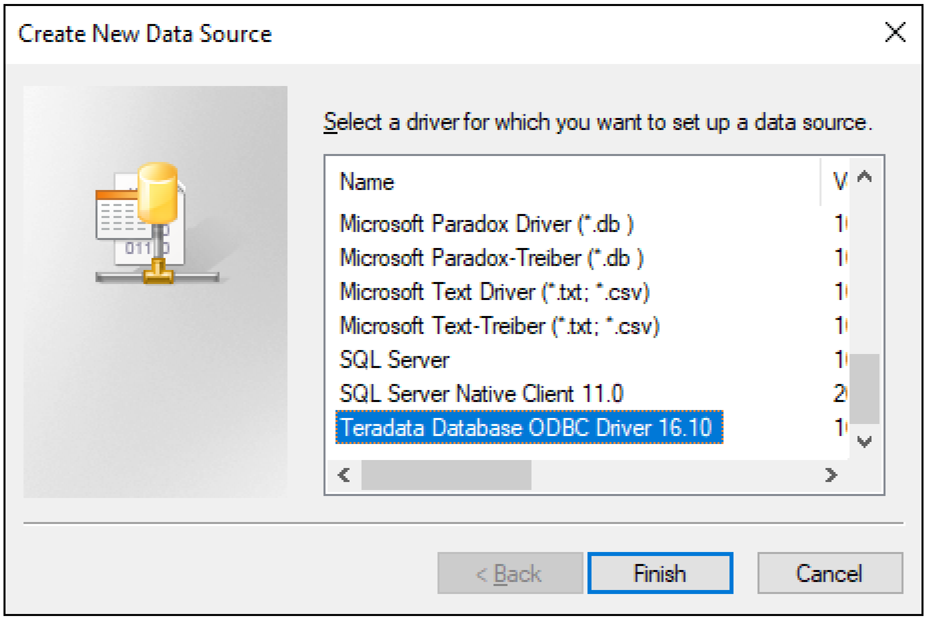
This example illustrates how to connect to a Teradata database server via ODBC.

Prerequisites

- The Teradata ODBC driver must be installed (downloadable from the Teradata website).
- You have the following database connection details: host, username, and password.

Connection

1. Press the **Windows** key, start typing `ODBC`, and select *Set up ODBC data sources (32-bit)* from the list of suggestions. (If appropriate, select the 64-bit option instead.)
2. In the ODBC Data Source Administrator dialog that appears, click the *System DSN* tab, and then click **Add**.
3. In the Create New Data Source dialog (screenshot below), select *Teradata Database ODBC Driver* and click **Finish**.

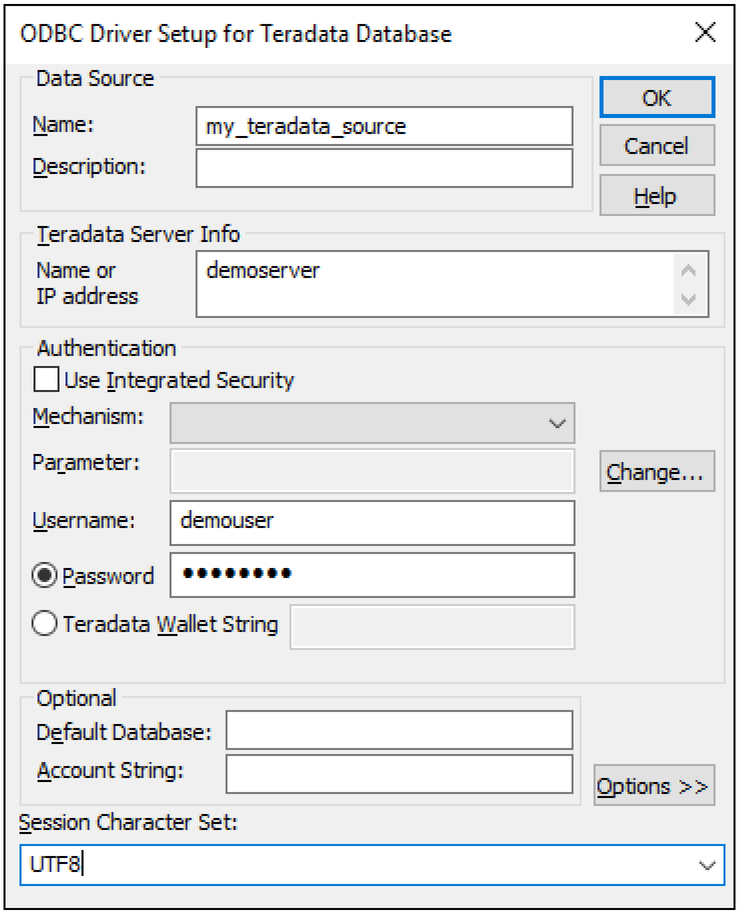


Create New Data Source

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

4. Enter the database and authentication details, and click **OK** when done.



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

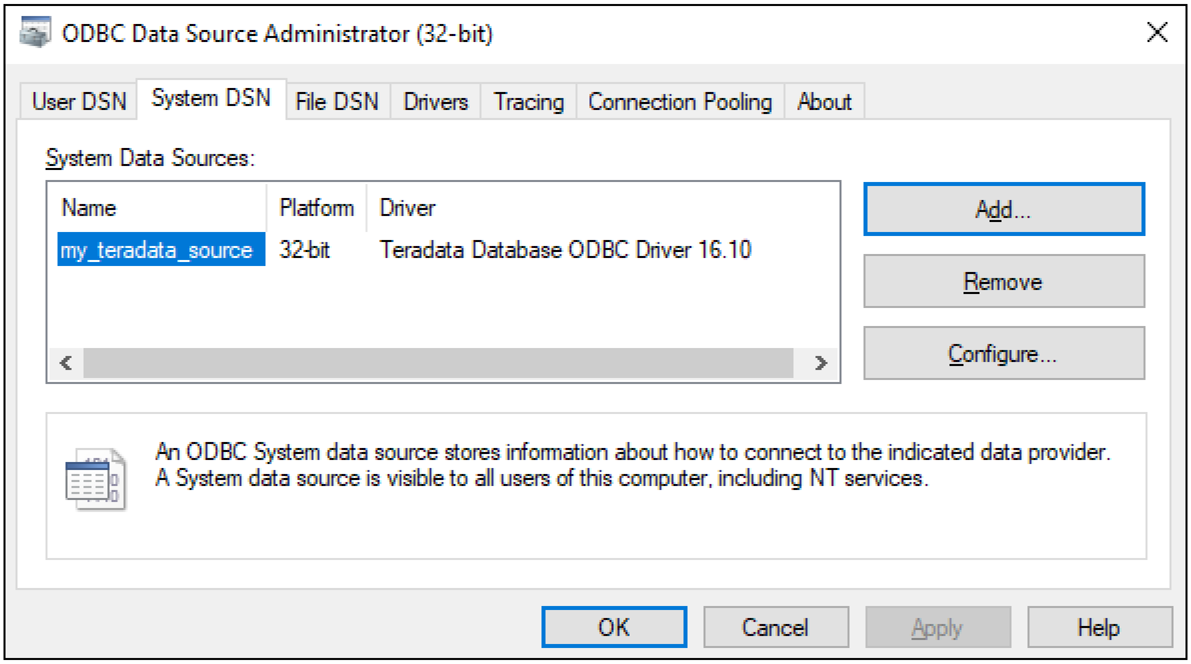
Optional

Default Database:

Account String:

Session Character Set: UTF8

5. Click **OK**. The data source now appears in the Data Sources list.



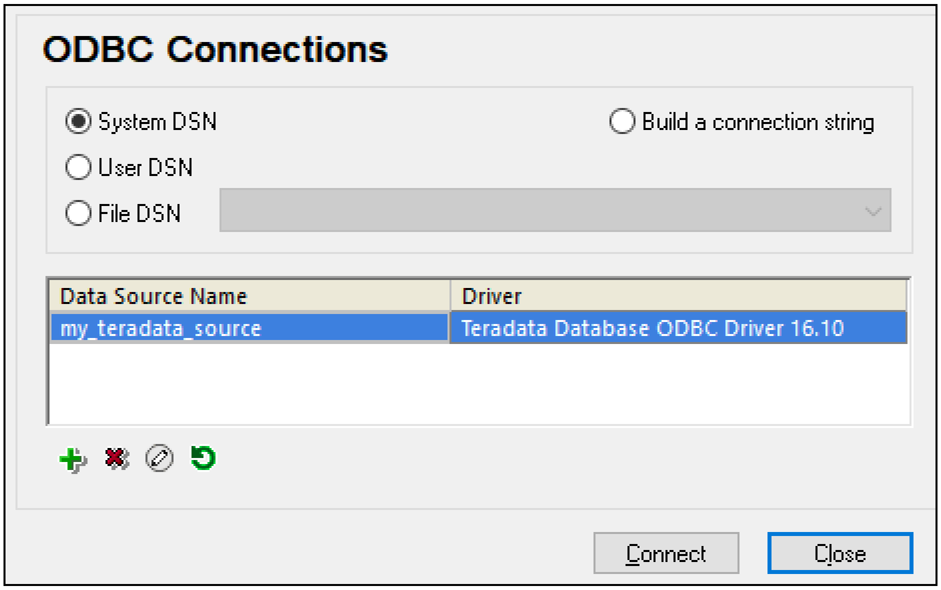
ODBC Data Source Administrator (32-bit)

System Data Sources:

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[170].
7. Click **ODBC Connections** and select *System DSN* (*see screenshot below*).



ODBC Connections

● System DSN ○ Build a connection string

○ User DSN

○ File DSN

Data Source Name	Driver
my_teradata_source	Teradata Database ODBC Driver 16.10

8. The data source you just created will be listed. Click **Connect** to start teh connection.

Note: If you get the following error: *"The driver returned invalid (or failed to return) SQL_DRIVER_ODBC_VER: 03.80"*, then make sure that the path to the ODBC client (for example, `C:\Program Files\Teradata\Client\16.10\bin`) exists in your system's **`PATH`** environment variable. If this path is missing, then add it manually.

4.2.2 General Procedures and Features

This section explains how to add a database to your mapping, select, remove, and edit database objects, handle database relationships, and configure various database settings.

Database column icons

Database tables are represented by the ▥ icon. Database columns are represented by the ▯ icon. If there is a constraint set for the column, the column's icon will have an additional symbol. If a column has more than one constraint assigned to it, only the constraint with the highest priority is shown in the column icon. The priority of constraints is described in the table below, starting with the highest priority.

🔑	This column is used as the table's primary key.
🔑	This column has a unique constraint.
🔑	This column has a foreign key that references the primary key of a different table.
XML	This column contains XML data[287].

▯D	There is a default value set for this column. If no value is supplied to this column, the default value will be inserted instead.

Add a database to your mapping

To be able to add a database to your mapping, you must select one of the following transformation languages[19]: Built-In, C++, C#, or Java. SQLite databases are supported only in Built-In. If you intend to deploy the mapping to FlowForce Server, execute it with MapForce Server, or use features such as Bulk Transfer[265] and stored procedures[297], you must select Built-In.

Once the desired transformation language is selected, you can add a database to the mapping in one of the following ways:

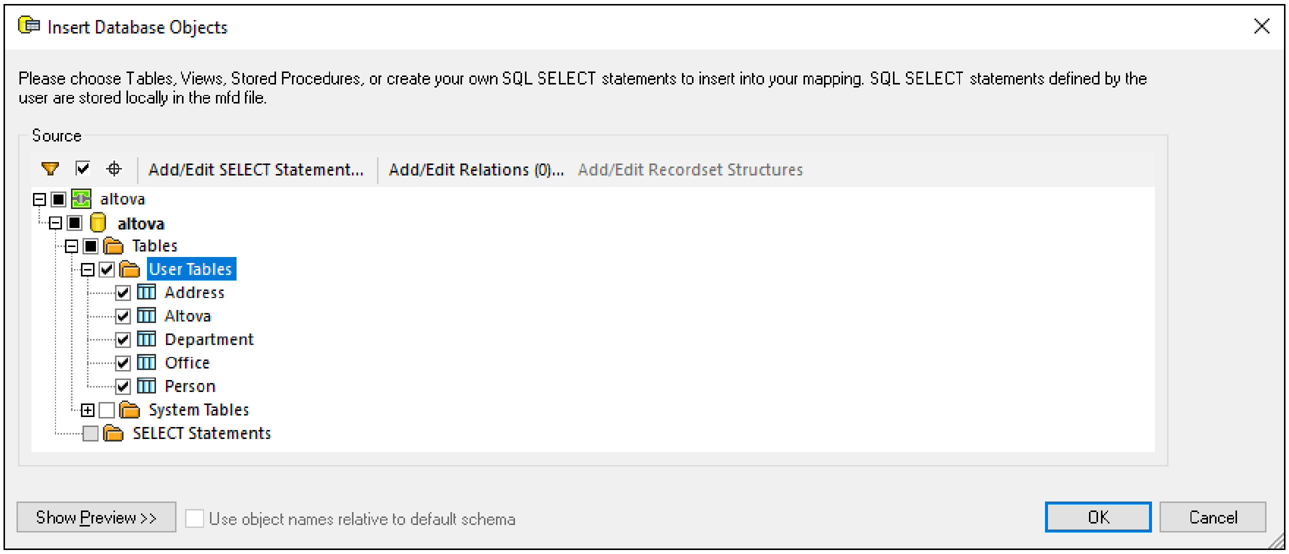
- Select **Database** in the **Insert** menu.
- Click the toolbar button.

When you take any of these actions, the database connection wizard[170] appears, guiding you through the steps required to connect to the database. For more information about how to connect to a database, see Connecting to a Data Source[168]. Once the database connection is successfully established, you are prompted to select database objects that you would like to add to your mapping (*see subsections below*).

Databases can also be added to the mapping as variables[360]. When you add a database structure as a variable, the same database connection wizard appears.

Add database objects

As soon as you have connected to the data source, you are prompted to select data objects you would like to include in your mapping. The **Insert Database Objects** dialog below shows the structure of the **`Altova.sqlite`** database. To include a database object in the mapping, select the check box next to it and click **OK**. In our example, we have included all the user tables.



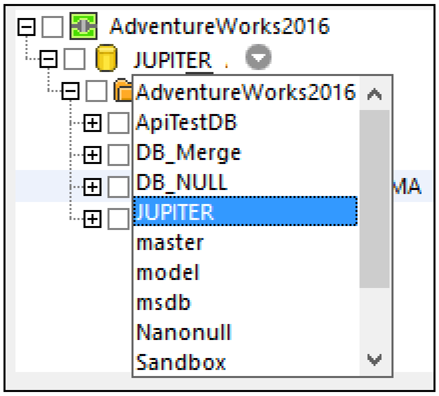
Insert Database Objects

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.

Structure of Insert Database Objects dialog

The top node in the structure indicates the database connection. The subsequent structure varies depending on the database kind. For example, Oracle and IBM DB2 databases have a schema node under the connection node, while other database types have a catalog (database) node. The bold font indicates the default catalog (database) or schema, as applicable.

If your database user account has access to multiple databases or schemas on the server, you can switch to any of them by clicking the icon (*see below*).

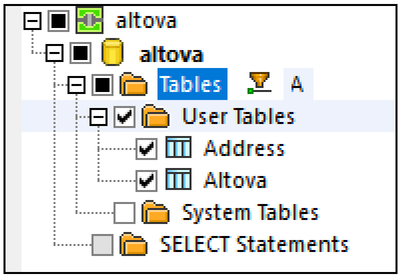


Options available in Insert Database Objects dialog

The options available in the **Insert Database Objects** dialog are described below.

⊟ Filter

The (**Filter**) button allows you to filter objects by name. Once you click the **Filter** button, the filter icon is available next to objects which support filtering (in this example, `Tables`). Click the filter icon to choose one of the following options: *No Filter*, *Contains*, *Does Not Contain*, *Start With*, *Ends With*, *Equals*. In our example, we have decided to include only tables whose names start with `A` (*see below*).



⊟ Show checked objects only

The (**Show checked objects only**) button displays only items with selected check boxes.

⊟ Object locator

The (**Object Locator**) button allows you to find specific database items. Select a particular object or type its name in the combo box which appears in the lower area of the dialog box.

⊟ Add/Edit SELECT statement

The **Add/Edit SELECT Statement** button enables you to add and edit custom SELECT statements for the current database. The data returned by such statements becomes available as a mapping source. For more information, see Custom SELECT Statements 243.

⊟ Add/Edit relations

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 exist in the database. For more information, see Local Relationships 253.

⊟ Add/Edit recordset structures

The **Add/Edit Recordset Structures** button applies to databases that support stored procedures 297. The button is enabled only if a stored procedure is currently selected in the database tree.

⊟ Show preview

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 Query Databases 277.

⊟ Use object names relative to default schema

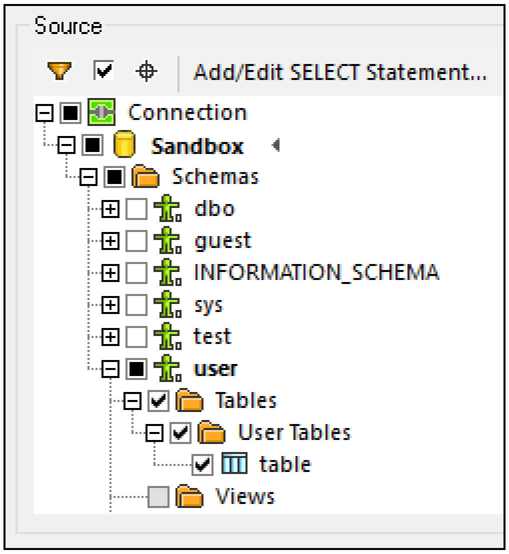
In MapForce, making database objects names relative to a schema is important if you plan to switch to a different database later. This is also useful if the database schema has been renamed on the server, and you need to update the mapping accordingly. If the new schema has the same structure as the one used at mapping design time, you can switch to it without having to change the mapping connections manually.

Note the following:

- Using object names relative to a default schema is possible for only databases that support schemas: IBM DB2, IBM Informix, IBM Db2 for i (iSeries), Oracle, PostgreSQL, Progress OpenEdge, SQL Server and Sybase.
- It is not possible to use relative names if the database component includes local relationships 253 or SELECT Statements as Virtual Tables 243.
- The *Use object names relative to default schema* check box affects the generated C#, C++, and Java program code. When this check box is selected, all the database references also become relative in the generated code.

To make database objects names relative to the default schema, take the following steps:

1. Open the **Insert Database Objects** dialog or right-click the title bar of the existing database component and select **Add/Remove/Edit Database Objects** from the context menu.
2. Select one or more objects that *belong to the default schema* or to the *default catalog (database) and schema*. The default database and schema are shown in bold. In the example below, the default catalog is `Sandbox`, and the default schema is `user`. This structure is specific for SQL Server databases and may vary in other database types.



3. Select the *Use object names relative to default schema* check box. Note that this check box is grayed out if the database does not support relative object names.

If the objects that you need in the mapping are in a different schema (not the default one), you have the following alternatives:

- You can connect as another database user that has access to the required default schema.
- If you have the required privileges, you can reconfigure the database server so as to change the default schema of the existing database user.

The example below shows how to change the default schema of a database user. The example is based on SQL Server and assumes that the `Sandbox` catalog and both the user and the schema already exist.

```
USE [Sandbox]
GO
ALTER USER [test_user] WITH DEFAULT_SCHEMA=[test_schema]
GO
```

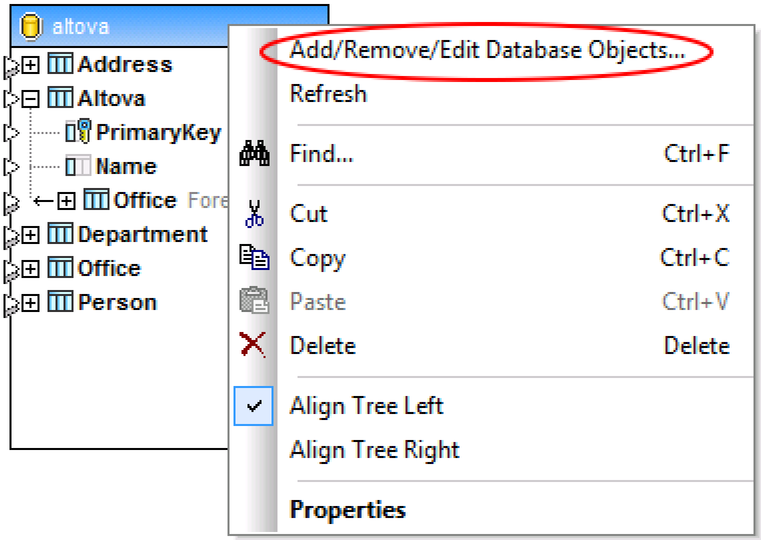
Switch to a DB/schema without losing mapping connections
When database objects names are relative to a schema, you can switch to a new database or schema *without losing mapping connections*. The following options are available:

- Open the database component settings[235] and click **Change**. Follow the wizard steps to connect to the new database *as a new user*. If the new database has the same structure, all the connections in the mapping will be updated automatically. This means that these connections will now match *the default catalog and schema of the new database user*.
- If you need to switch to a new database on a regular basis, it is recommended to define the database connection as a Global Resource[815]. For example, the Global Resource could have two configurations: a default configuration for the development database and a production configuration.

If database objects appear in red after switching, this indicates that they do not exist in the new database schema.

Edit database objects

To change database objects, right-click the database component and select **Add/Remove/Edit Database Objects** from the context menu (*see below*). This opens the **Add/Remove/Edit Database Objects** dialog, which allows you to define the same settings and properties as in the **Insert Database Objects** dialog.



SQL auto-completion suggestions

When you type SQL statements in certain contexts, MapForce may suggest text entries automatically. Auto-completion is available in the SQL Editor (see DB Query Pane 277), the *Custom SQL* text box in the Database Table Actions 259 dialog box, and the Add SELECT Statement 243 dialog box.

To disable auto-completion suggestions, take the following steps:

1. Select the **Tools | Options** menu item or press **Ctrl+Alt+O**.
2. Open the *Database | SQL Editor* section.
3. Clear the *Automatically open* check box in the *Entry Helpers* section.

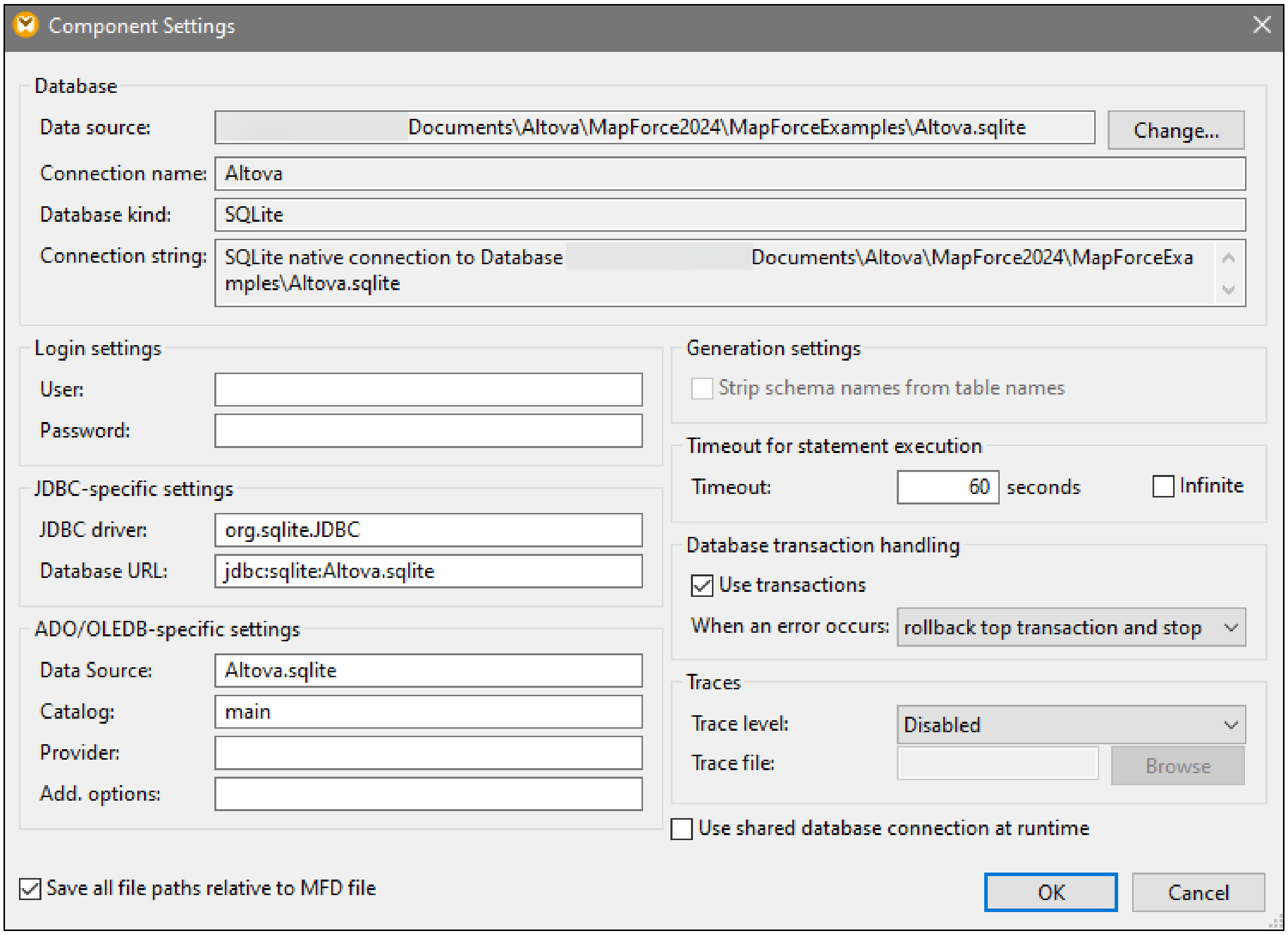
To invoke auto-completion suggestions manually, press **Ctrl+Space**.

To find out more about other database-related settings, see Database 1083.

4.2.2.1 DB Component Settings

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

- Double-click the component title bar.
- Right-click the component and click **Properties**.
- Select the component in the mapping. Then click the **Component** menu and select **Properties** from the context menu.



Component Settings

Database

Data source: [redacted] Documents\Altova\MapForce2024\MapForceExamples\Altova.sqlite

Connection name: Altova

Database kind: SQLite

Connection string: SQLite native connection to Database [redacted] Documents\Altova\MapForce2024\MapForceExamples\Altova.sqlite

Login settings

User:

Password:

JDBC-specific settings

JDBC driver: org.sqlite.JDBC

Database URL: jdbc:sqlite:Altova.sqlite

ADO/OLEDB-specific settings

Data Source: Altova.sqlite

Catalog: main

Provider:

Add. options:

Generation settings

□ Strip schema names from table names

Timeout for statement execution

Timeout: 60 seconds □ Infinite

Database transaction handling

☑ Use transactions

When an error occurs: rollback top transaction and stop

Traces

Trace level: Disabled

Trace file:

□ Use shared database connection at runtime

☑ Save all file paths relative to MFD file

The available settings are listed below.

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. Connections to tables with the same names will be retained. You can also change the tables[235] in the component, by right clicking a database component and selecting **Add/Remove/Edit Database Objects**.

⊟ Data Source

Specifies the absolute or relative path of the current data source.

⊟ Connection Name

Specifies the name of the connection. This name is generated automatically by MapForce. Typically, it is the same as the data source name, but it may also be an alias if you are connecting with Altova Global Resources[815]. If there are multiple database components with the same connection in the mapping, the connection name will look as follows: `<connection1>`, `<connection2>`, etc.

⊟ Database Kind

Specifies the database type (e.g., SQLite).

⊟ Connection String

Displays the current database connection string. This read-only field is generated based on the information you supply when you create or change a 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 a database. Mandatory if the database requires a user name to connect.

⊟ Password

Enables you to change the password for connecting to a database. Mandatory if the database requires a password to connect.

JDBC-specific settings

The JDBC-specific settings are relevant when the mapping contains a JDBC connection and is executed by generated Java code or by MapForce Server.

Note: ADO, ADO.NET, and ODBC connections are converted to JDBC (and the JDBC settings below apply) when the mapping is run on a Linux or macOS machine. For details, see *Database Mappings in Various Execution Environments*[166].

⊟ 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

The ADO/OLEDB-specific settings are relevant when the mapping contains an ADO connection and is executed by the generated C# code, C++ code, or by MapForce Server running on Windows. For details, see *Database Mappings in Various Execution Environments*[166]. 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 of 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. The *Strip schema names from table names* option allows you to strip database schema names from the generated code and retain only the table names. Note that this option works only for SQL SELECT statements generated by MapForce. User-defined SQL statements[243] will not be modified.

The *Generation settings* option is supported for backward compatibility and should be avoided. To make database object names relative to the default schema, use the approach described in *Use Object Names Relative to Default Schema*[233].

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.

⊟ Timeout

Defines a period of time, in seconds, during which the execution engine must wait for a database response before aborting the execution of a database statement. The default timeout is 60 seconds.

⊟ Infinite

When enabled, this option instructs the execution engine to never time out.

Note: Timeout for statement execution is not applicable to SQLite databases.

Database transaction handling

During the execution of a mapping that has a database component, there may be various database-related errors (e.g., duplicate index keys, NULL values inserted into non-NULL columns, etc.). To be able to roll back your database data in case of an error, you need to enable database transaction handling. You can enable transaction rollback at database-component level (*current setting*), at table-action level[263], and at stored-procedure level[309]. For more information about some of the possible transaction-handling scenarios, see Transaction Rollback: Scenarios[271].

Some mappings may contain multiple database components that may have the same or different database connections. The outcome of such mappings in case of a database-related error depends on the execution engine:

- If the mapping is run with MapForce, only one target component can be executed when the mapping runs. This is the component where the 👁 button is enabled. If a database error occurs in that component, and if the *Use transactions* check box is enabled, all the changes done by the component will be rolled back.
- If the mapping is run with MapForce Server or a MapForce-generated program, all the target components are executed, sequentially. In this case, when a database error occurs, the rollback will take place for the database component where the error occurred. The mapping will stop or continue executing the next target component depending on the value you selected from the *When an error occurs* drop-down list (*see below*).

⊟ Use transactions

Enables transaction processing for a target database component. Transaction processing is enabled for all tables of the database component when you select this option. Enabling transaction handling at database-component level encloses all database changes in a single transaction that will be rolled back in case of a database-related error.

⊟ When an error occurs

If you have selected the *Use transaction* check box, you can choose what to do when a database-related error occurs:

- *Rollback top transaction and stop:* The transaction which encloses all the database changes is rolled back, and the execution of the mapping stops.
- *Rollback top transaction and continue:* Same as above, but the mapping continues to run after the rollback (e.g., to process a second target component).

At database-component level, you control whether processing should continue for other target components. For example, in an XML-DB-JSON mapping, an error has occurred in the database component. In this case, the JSON file can still be processed and retrieved if you have enabled the *Rollback top transaction and continue* option.

Traces

When a mapping writes data to a database, you can enable database tracing and error logging. Tracing is useful if you want to track all the changes made to the database during the execution of the mapping. The changes made to the database are logged in a trace report. If there are errors during the execution, these errors will also be logged. Tracing is compatible only with the Built-In transformation language.

You can enable tracing at different levels:

- *Database-component level:* Tracing at this level could be useful for mappings that have multiple target database components, and you need to enable tracing only for some of them. Enabling tracing at database-component level automatically enables it for all tables and stored procedures in that component. To be able to be traced, the relevant tables and stored procedures must be connected to source nodes.
- *Table* [264] *or stored-procedure* [309] *level:* You can decide whether to enable tracing for a specific table or stored procedure. At table level, tracing includes events related to table actions (e.g., *Insert All*). In the case of stored procedures, events related to the stored procedure call are traced.
- *Database-field level* [264]*:* By default, all fields are traced, but you can exclude certain fields from the trace report or choose to include certain fields only in case of an error.

Importantly, the three levels above are hierarchical. This means that in order to set tracing at a lower level, you must enable tracing at parent level first. For example, if you need to set tracing at table level, you must first enable tracing at database-component level. The same principle applies when you narrow down the tracing level. For example, if you limit tracing only to errors at database-component level, it is not possible to use full tracing at table and stored-procedure level.

MapForce allows you to set the following tracing options:

⊟ Trace level

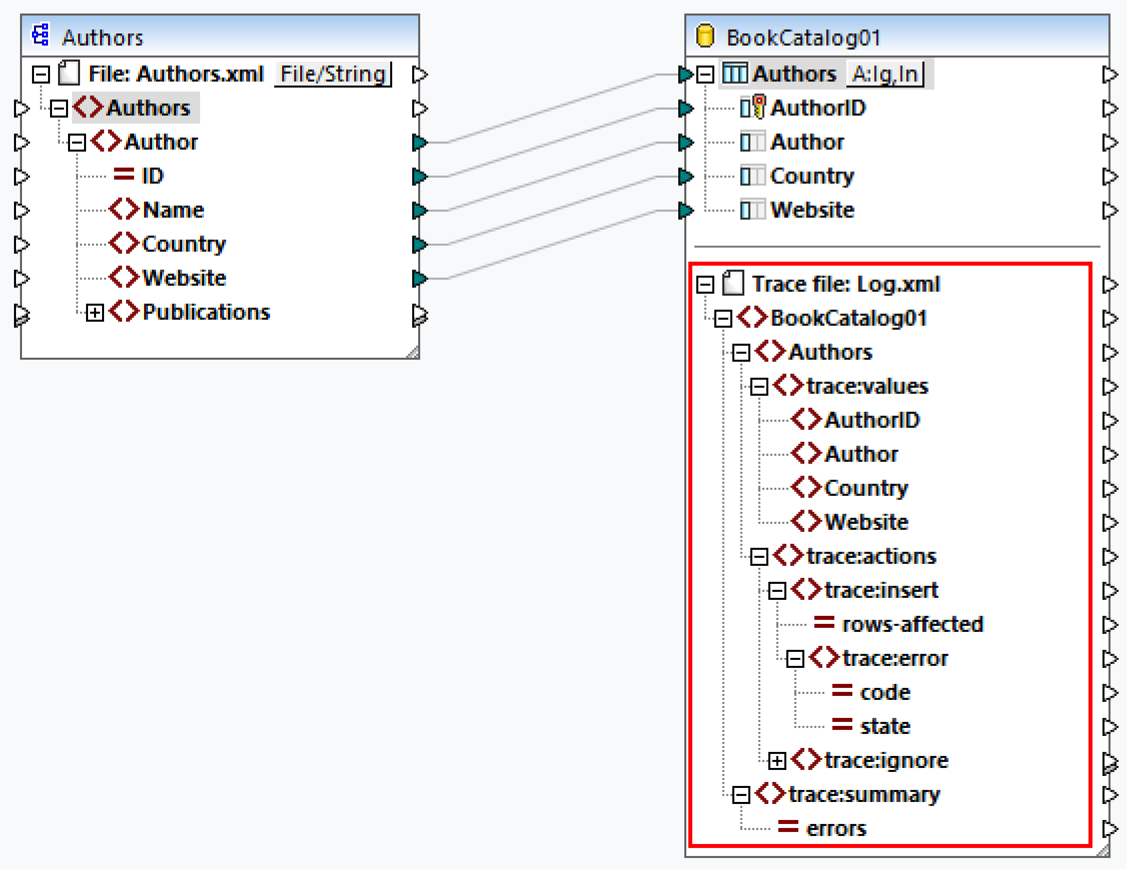
When tracing is enabled, the actions performed by the mapping against the database are logged in a trace file. You can choose to log all actions, only errors, or disable tracing completely.

⊟ Trace file

Specifies the file to which database trace information will be written when the mapping runs. This path can be absolute or relative and is influenced by the *Save all file paths relative to MFD* file check box. The trace file is in XML format. If you prefer the log file to be in a format other than XML, you can map data from it to some other component (e.g., a text file, another database, etc.).

Structure of a trace file

When you enable tracing for a database component, a tracing structure becomes available in the lower half of the component (*screenshot below*).



The screenshot above shows that the top node in the tracing structure is the name of the trace file (`Log.xml`). The rest of the tracing structure is modeled based on the structure of the database tables or stored procedures that take part in the mapping. In this example, the top element has the same name as the database. The `BookCatalog01` element has two child elements: `Authors` and `trace:summary`. The `Authors` element reflects the structure of the table that is added to the database component. The `trace:summary` element includes an `errors` attribute which reports the number of encountered errors.

The `Authors` element contains two child elements: `trace:values` and `trace:actions`. The `trace:values` structure displays all the columns of the database table. By default, all columns are traced, but you can change this by configuring tracing at database-filed level (*see above*). For stored procedures, this structure displays the parameters of the stored procedure.

The `trace:actions` element includes information about all the actions that are defined for this particular database table. In our example, two actions have been configured for the `Authors` table: *Ignore If* and *Insert Rest*. Each traced action has a `rows-affected` attribute that specifies how many rows have been affected by the respective database action. The `trace:error` element is populated only if an error occurs. This element has two attributes: `code` and `state`. The text of the error and the attribute values are supplied by the database driver and will therefore be different for different databases.

Trace file in the Output pane
To preview the trace file, click the Output pane. Note that the trace report displayed in the Output pane is for information purposes only and does not reflect the actual execution results. To produce an actual trace report, run the SQL script 258 from the Output pane. An example of a trace file is given below:

```
<BookCatalog01>
     <Authors>
          <trace:values>
               <Author>Neil Gaiman</Author>
               <Website>www.neilgaiman.com</Website>
          </trace:values>
          <trace:actions>
               <trace:ignore/>
          </trace:actions>
     </Authors>
     <Authors>...</Authors>
     <Authors>...</Authors>
     <trace:summary errors="0"/>
</BookCatalog01>
```

For more information about tracing, see *Scenario 1* in Transaction Rollback: Scenarios 271.

Save all file paths relative to MFD file

When this option is enabled, MapForce saves the file paths displayed in the **Component Settings** dialog box relative to the location of the MapForce design file (**`.mfd`**). Use relative paths if you intend to run the mapping with MapForce Server on a different operating system. See also Relative and Absolute Paths 43.

Use shared database connection at runtime

This option enables you to choose whether several database components that use the same data source and feature in the same mapping should share the same database connection. By default, this option is disabled; otherwise, it might change the behavior of the mapping, especially when the same connection is shared between one or more source components and a target component.

Sharing the same database connection enables you to solve various issues, for example, with table/row locks, transaction isolation, and the number of server connections (*see details below*).

- When you read a row from a table and try to update the same row, it might happen (depending on the vendor) that you run into a table/row lock error. With a shared database connection, this issue can be avoided.
- With connection-sharing enabled, you will be able to read already changed rows that are wrapped in a transaction. With separate connections, only committed changes are visible.
- Connection sharing also helps reduce the number of database logins, which will enable you to reduce the overall processing time for mappings with a lot of database components that use the same data source. The database logon procedure can be time consuming, especially with cloud server instances over a slow network or when the database server is busy.

Note that DB connection sharing is *not* supported in user-defined functions 457.

4.2.2.2 Custom SELECT Statements

MapForce allows you to create custom SQL SELECT statements with or without parameters. These statements are represented as table-like structures, from which you can map data to other components. For example, you can create a custom statement to join tables, filter your database data, and define parameters that can accept values from another component in the mapping.

SQL SELECT statements without parameters are supported in C++, C#, Java, and Built-In languages. SQL SELECT statements with input parameters are compatible only with the Built-In transformation language.

Create/Edit/Remove a SELECT statement
To add a SELECT statement to a database component, follow the instructions below:

1. Right-click the title of the database component and select **Add/Remove/Edit Database Objects** from the context menu. Alternatively, select the database component and select the menu command **Component | Add/Remove/Edit Database Objects**.
2. In the **Add/Remove/Edit Database Objects** dialog, do one of the following:

   - To enter a custom SELECT statement, click the **Add/Edit SELECT Statement** button.
   - To generate the SELECT statement for a particular table, right-click the relevant table and select **Generate and add an SQL statement** from the context menu. You will be able to edit the generated statement afterwards.

To edit an existing SELECT statement, do one of the following:

- Right-click the SELECT statement in the component and select **Edit SELECT Statement**.
- Right-click the database component and select **Add/Remove/Edit Database Objects** from the context menu. Then double-click the relevant SELECT statement in the **Add/Remove/Edit Database Objects** dialog.
- In the **Add/Remove/Edit Database Objects** dialog, select the relevant SELECT statement and click **Add/Edit SELECT Statement**.
- In the **Add/Remove/Edit Database Objects** dialog, right-click the relevant SELECT statement and select **Edit a SELECT Statement**.

To remove a SELECT statement, take the steps below:

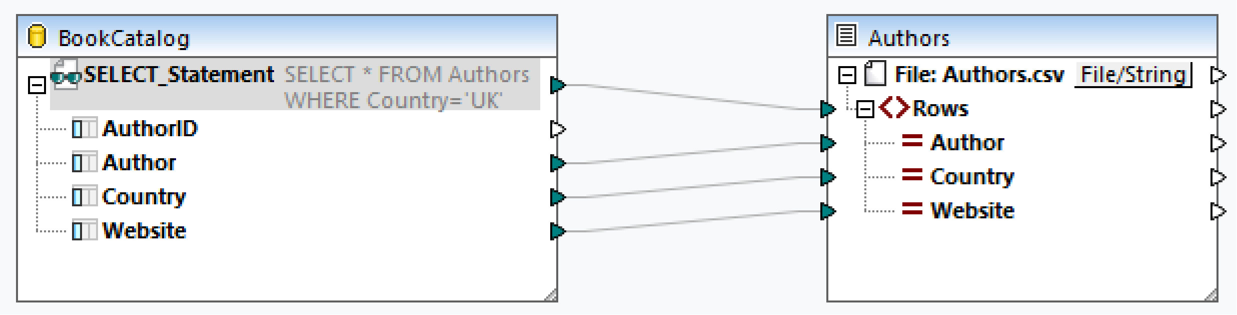
1. Right-click the database component and select **Add/Remove/Edit Database Objects**.
2. Right-click the SELECT statement you want to delete and select **Remove SELECT Statement** from the context menu.

Important notes
Note the following points:

- All calculated expressions in the SELECT statement must have a unique correlation name (e.g., `SELECT *, (Quantity*UnitPrice) AS Price`) to become available as mappable items.
- If you connect to an Oracle or IBM DB2 database using JDBC, the `SELECT` statement must not have the final semicolon.

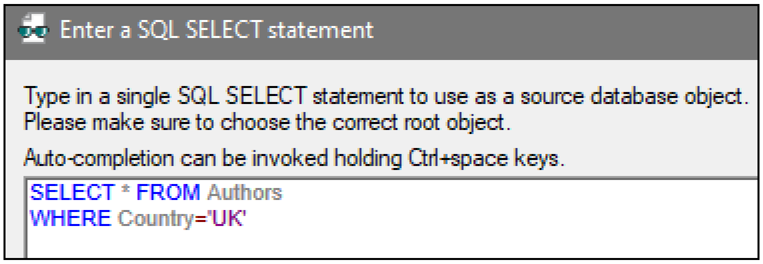
SQL SELECT statements without parameters

The example below shows how to work with custom SELECT statements without parameters. In the mapping shown below, we map database data to a text file. The **BookCatalog.sqlite** database has a parent table called Authors and a child table called Books. However, only the SELECT statement with a tree structure is displayed in the component. The structure of the tree depends on the SQL query you define in the **Enter a SQL SELECT Statement** dialog. Since nothing will be mapped from the Authors and Books tables, these tables are absent from the component.



SELECT statement

For the database component, we have added the following SQL statement (*see instructions in Create/Edit/Remove a SELECT Statement*):

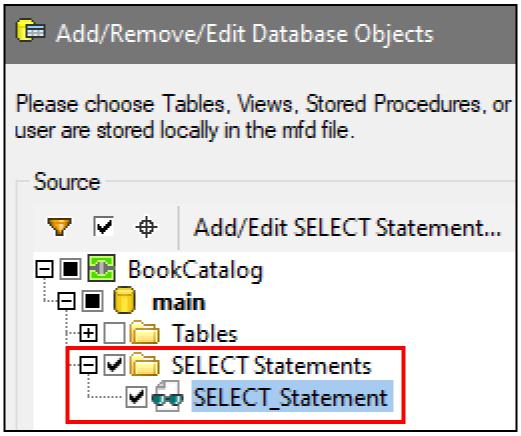


Type in a single SQL SELECT statement to use as a source database object. Please make sure to choose the correct root object.

Auto-completion can be invoked holding Ctrl+space keys.

```
SELECT * FROM Authors
WHERE Country='UK'
```

The SQL statement selects all the tables from the Authors table and filters database data to include only authors from the UK. As soon as we add this statement to the **Enter a SQL SELECT Statement** dialog, the statement becomes available in the **Add/Remove/Edit Database Objects** dialog (*screenshot below*). The statement is also visible in the database component (*see mapping above*). The number of visible lines of the SELECT statement can be configured in the Options[1077] dialog box (the *Limit annotation display* option).



Add/Remove/Edit Database Objects

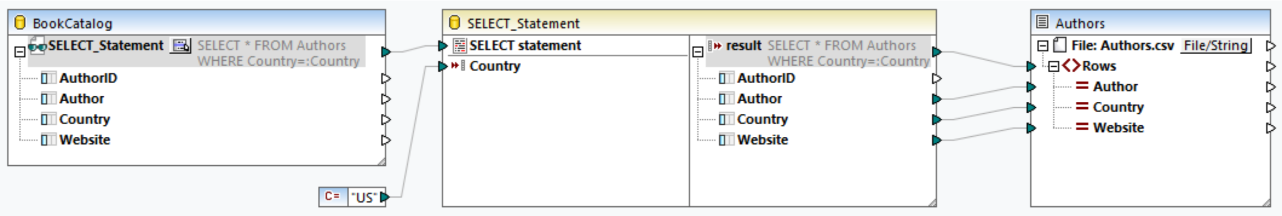
Please choose Tables, Views, Stored Procedures, or user are stored locally in the mfd file.

Output
The output displays a list of comma-separated values that include authors only from the UK (*code listing below*).

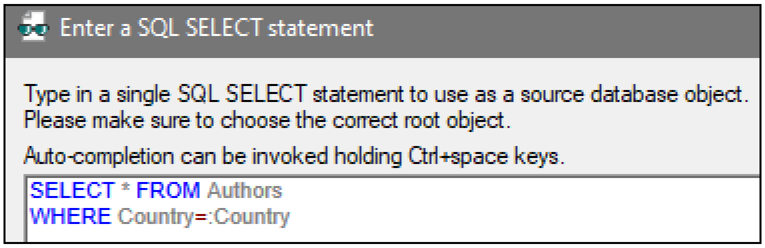
```
Author,Country,Website
Bram Stoker,UK,www.bramstoker.org
Charles Dickens,UK,www.charlesdickensinfo.com
Emily Brontë,UK,n/a
James Herbert,UK,www.james-herbert.co.uk
Neil Gaiman,UK,www.neilgaiman.com
Terry Pratchett,UK,www.terrypratchettbooks.com
Agatha Christie,UK,www.agathachristie.com
Roald Dahl,UK,www.roalddahlfans.com
David Walliams,UK,www.worldofdavidwalliams.com
Kenneth Grahame,UK,n/a
Philip Pullman,UK,www.philip-pullman.com
J.K. Rowling,UK,www.jkrowling.com
Ann Cleeves,UK,www.anncleeves.com
```

SQL SELECT statements with parameters

Our second example illustrates a mapping in which the database component has a custom SELECT statement with a parameter (*screenshot below*).



SELECT statement
For the `BookCatalog` component, we have entered the following SQL 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 * FROM Authors
WHERE Country=:Country
```

The statement uses the `Country` parameter. This parameter will accept values from the constant (*under the BookCatalog component*). To be able to map data from the SELECT statement with the parameter, click the button next to the `SELECT_Statement` node in the database component (*mapping above*) and select **Insert Call with Parameters** from the context menu. This inserts a Call component with parameters (*central component in mapping above*). The Call component has two parts: The left part accepts an input parameter (in our case, `Country`), and the right part replicates the SELECT statement with the tree structure from the database component. The filtered data is then mapped to the `Authors` text file.

Output
The output now displays authors only from the USA (*code listing below*).

```
Author,Country,Website
Stephen King,US,www.stephenking.com
Frank Herbert,US,n/a
Isaac Asimov,US,www.asimovonline.com
Blake Crouch,US,www.blakecrouch.com
Ray Bradbury,US,www.raybradbury.com
Joe Hill,US,www.joehillfiction.com
Josh Malerman,US,www.joshmalerman.com
George R. R. Martin,US,www.georgerrmartin.com
A. J. Finn,US,n/a
Dan Brown,US,www.danbrown.com
Dean Koontz,US,www.deankoontz.com
```

Example files

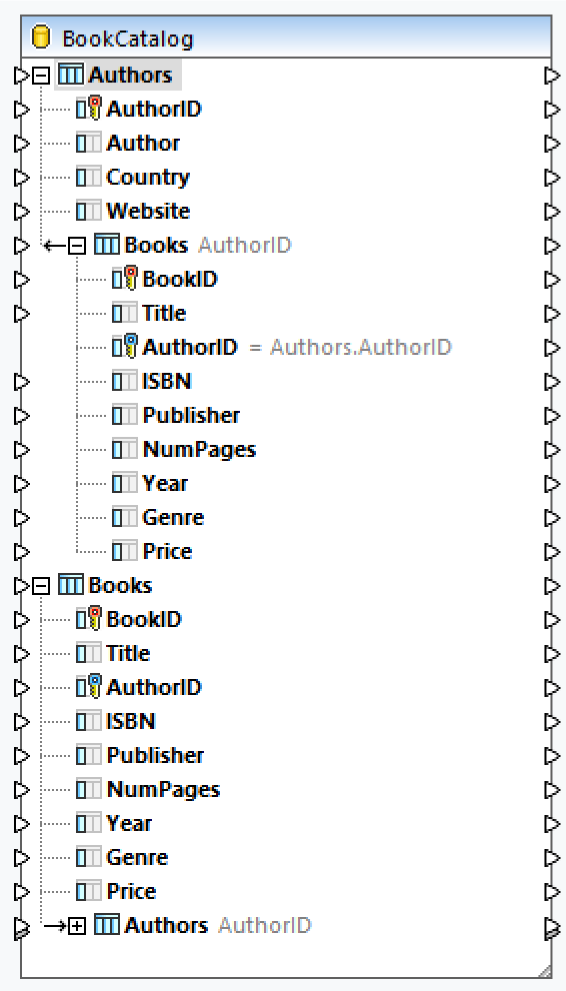
For more information about mappings that use custom SQL SELECT statements as input, see the following examples in the **`MapForceExamples`** folder:

- **`DB_EmployeeListByTitle.mfd`**
- **`DB_MostExpensiveArticle.mfd`**
- **`DB_ManagerList_AllOffices.mfd`**
- **`DB_ManagerList_SelectedDepartment.mfd`**
- **`DB_ManagerList_SelectedOffice.mfd`**

4.2.2.3 DB Relationships

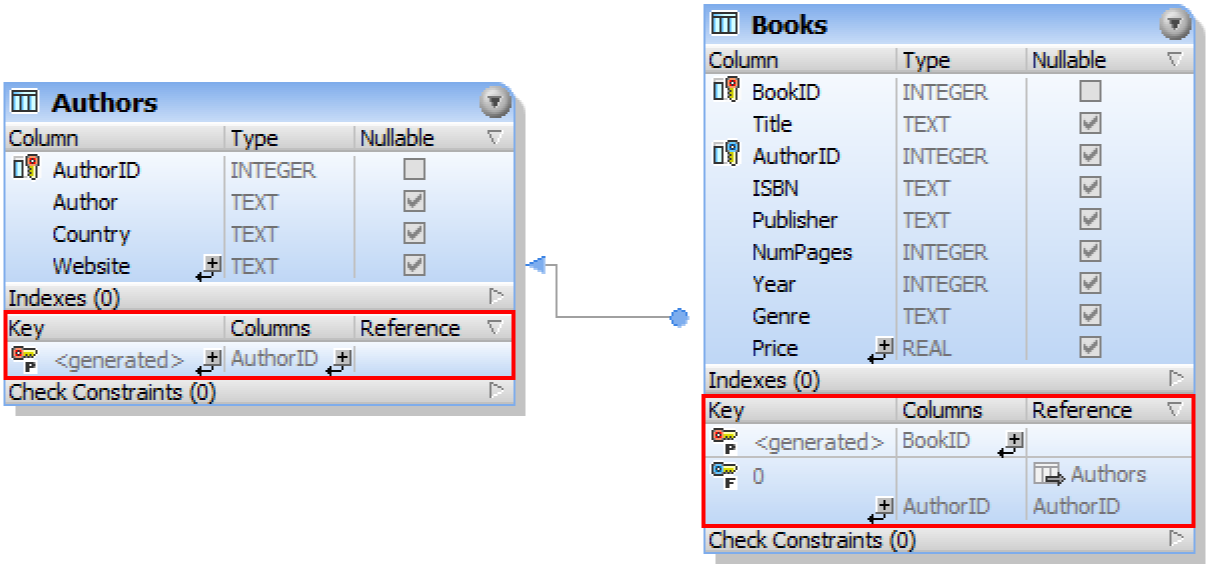
When you add a database 231 as a source component to your mapping, each table appears as the root table (*screenshot below*). When you click on the plus icon of a root table, you can see all related tables below the root table. The database component below displays two types of arrows that mean the following:

- The arrow pointing to the left (←) indicates that the Books table is a child table of the Authors table.
- The arrow pointing to the right (→) shows that the Authors table is the parent of the Books table.



Structure of BookCatalog.sqlite

Depending on your business needs, you can use various mapping scenarios. The subsections below describe some of the possible scenarios. All the scenarios described below feature a hierarchical database called **BookCatalog.sqlite**. The database has two tables (Authors and Books) that have a foreign-key relationship. The screenshot below shows that the Books table has a foreign key called AuthorID that references the primary key in the Authors table.



Authors

Column	Type	Nullable
AuthorID	INTEGER	☐
Author	TEXT	☑
Country	TEXT	☑
Website	TEXT	☑

Indexes (0)

Key	Columns	Reference
<generated>	AuthorID	

Check Constraints (0)

Books

Column	Type	Nullable
BookID	INTEGER	☐
Title	TEXT	☑
AuthorID	INTEGER	☑
ISBN	TEXT	☑
Publisher	TEXT	☑
NumPages	INTEGER	☑
Year	INTEGER	☑
Genre	TEXT	☑
Price	REAL	☑

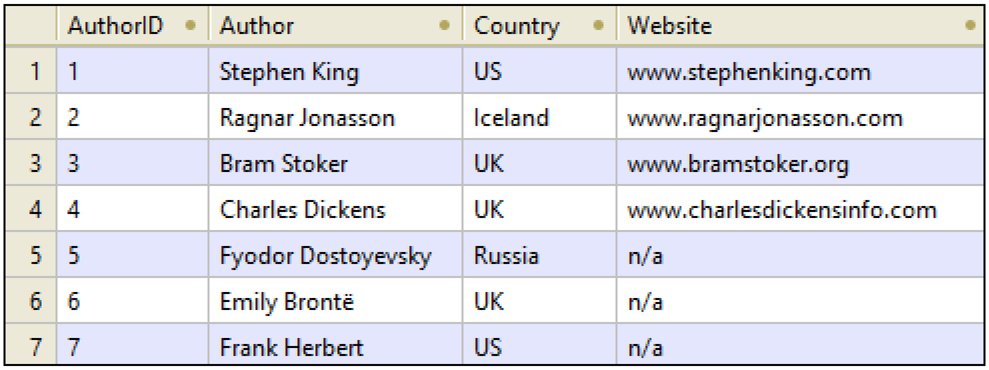
Indexes (0)

Key	Columns	Reference
<generated>	BookID	
0		Authors
	AuthorID	AuthorID

Check Constraints (0)

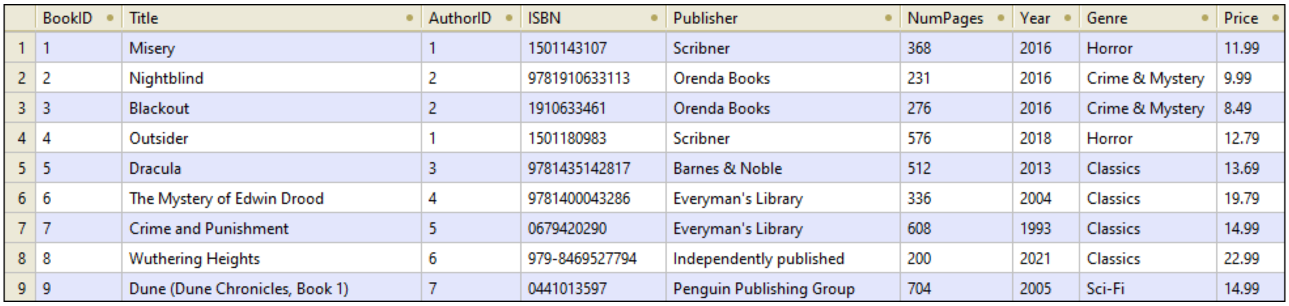
Sample data from BookCatalog.sqlite

The extracts from the `Authors` and `Books` tables are given below:



	AuthorID	Author	Country	Website
1	1	Stephen King	US	www.stephenking.com
2	2	Ragnar Jonasson	Iceland	www.ragnarjonasson.com
3	3	Bram Stoker	UK	www.bramstoker.org
4	4	Charles Dickens	UK	www.charlesdickensinfo.com
5	5	Fyodor Dostoyevsky	Russia	n/a
6	6	Emily Brontë	UK	n/a
7	7	Frank Herbert	US	n/a

Authors table

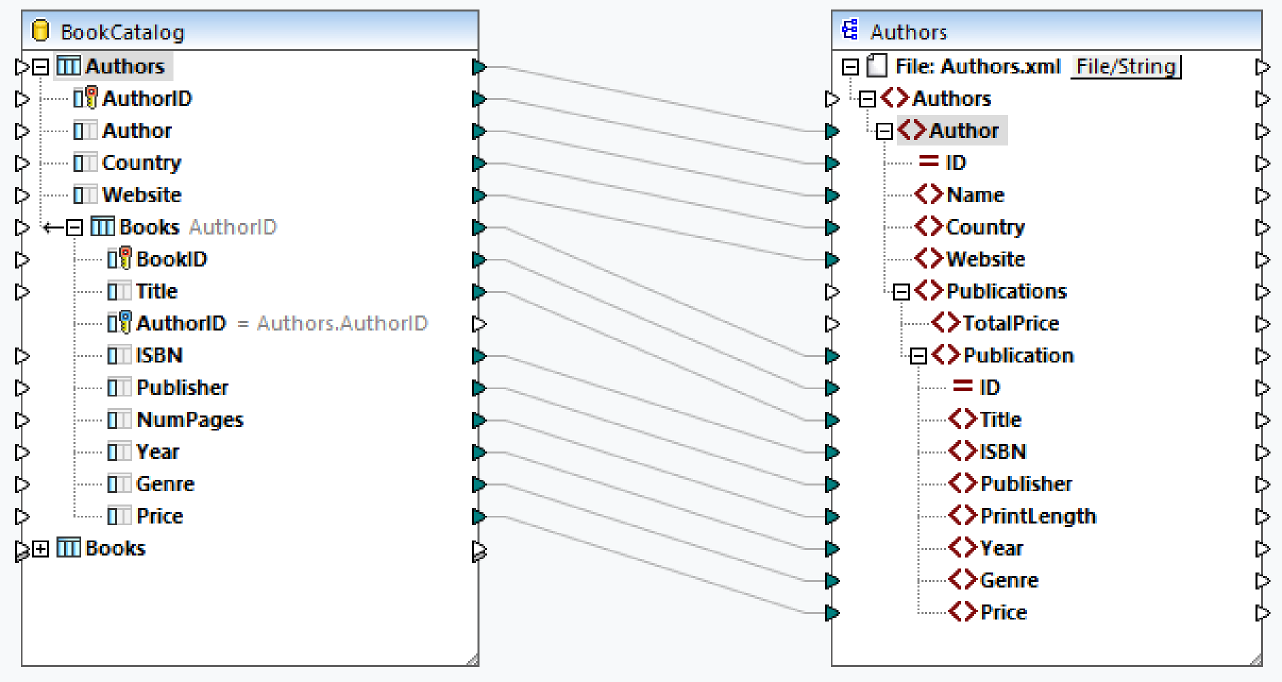


	BookID	Title	AuthorID	ISBN	Publisher	NumPages	Year	Genre	Price
1	1	Misery	1	1501143107	Scribner	368	2016	Horror	11.99
2	2	Nightblind	2	9781910633113	Orenda Books	231	2016	Crime & Mystery	9.99
3	3	Blackout	2	1910633461	Orenda Books	276	2016	Crime & Mystery	8.49
4	4	Outsider	1	1501180983	Scribner	576	2018	Horror	12.79
5	5	Dracula	3	9781435142817	Barnes & Noble	512	2013	Classics	13.69
6	6	The Mystery of Edwin Drood	4	9781400043286	Everyman's Library	336	2004	Classics	19.79
7	7	Crime and Punishment	5	0679420290	Everyman's Library	608	1993	Classics	14.99
8	8	Wuthering Heights	6	979-8469527794	Independently published	200	2021	Classics	22.99
9	9	Dune (Dune Chronicles, Book 1)	7	0441013597	Penguin Publishing Group	704	2005	Sci-Fi	14.99

Books table

Scenario 1: Preserve the hierarchy

In our first scenario, we map data from **`BookCatalog.sqlite`** to **`Authors.xsd`** (*see screenshot below*). In this mapping, `Authors` is the root table. Our goal is to preserve the hierarchical relationship and get all the authors with their corresponding books in the output.

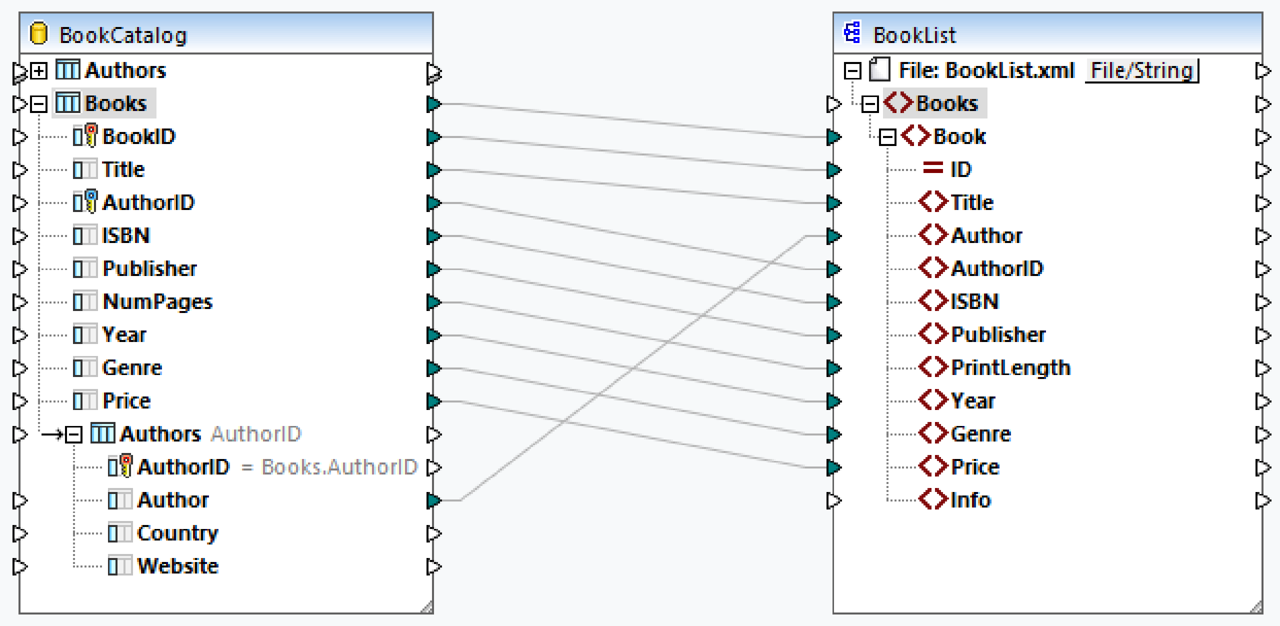


The code listing below shows an extract of the output:

```
<Authors>
      <Author ID="23">
            <Name>Fredrik Backman</Name>
            <Country>Sweden</Country>
            <Website>www.fredrikbackmanbooks.com</Website>
            <Publications>
                  <Publication ID="26">
                        <Title>Anxious People</Title>
                        <ISBN>978-1-4059-3025-3</ISBN>
                        <Publisher>Penguin Books Ltd</Publisher>
                        <PrintLength>416</PrintLength>
                        <Year>2021</Year>
                        <Genre>Humor</Genre>
                        <Price>9.99</Price>
                  </Publication>
                  <Publication ID="27">
                        <Title>A Man Called Ove</Title>
                        <ISBN>9781444775815</ISBN>
                        <Publisher>Sceptre</Publisher>
                        <PrintLength>320</PrintLength>
                        <Year>2015</Year>
                        <Genre>Humor</Genre>
                        <Price>11.46</Price>
                  </Publication>
            </Publications>
      </Author>
</Authors>
```

Scenario 2: Swap the tables

In the second scenario, our goal is to get a list of books and their details in the output file. To achieve the goal, we will use Books as the root table. The table relationships will stay intact. The mapping design looks as follows:



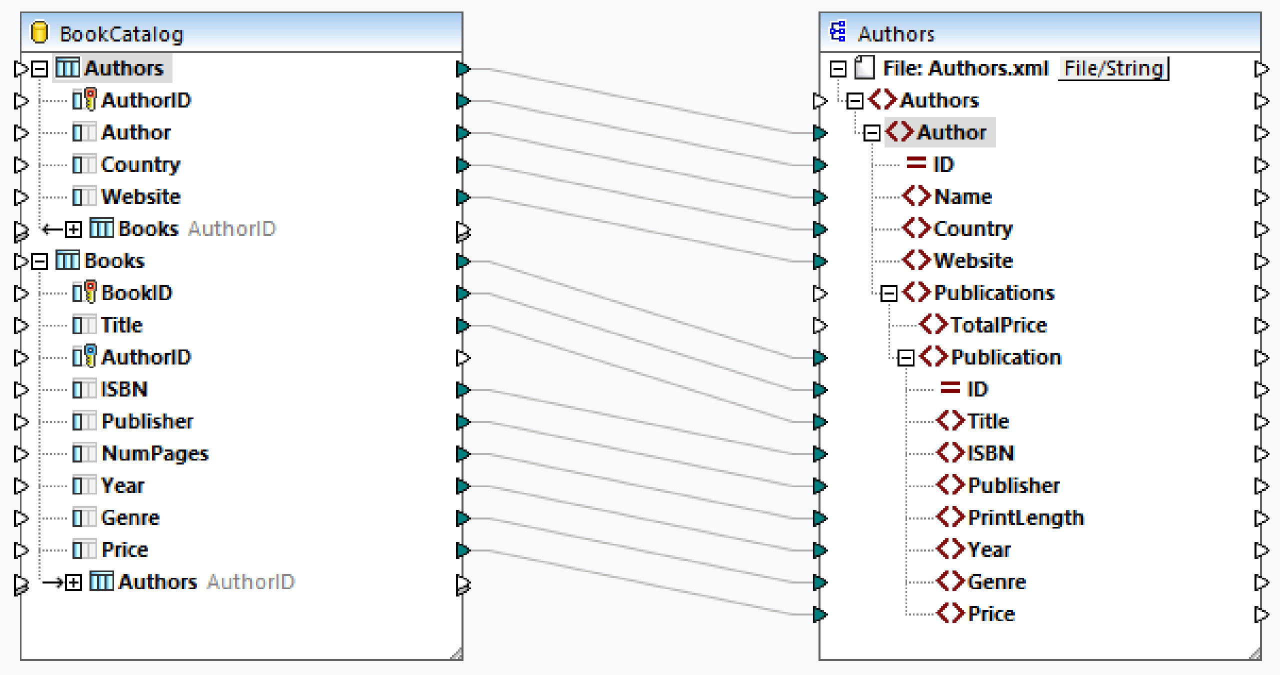
The code listing below shows an extract of the output:

```
<Books>
      <Book ID="3">
            <Title>Blackout</Title>
            <Author>Ragnar Jonasson</Author>
            <AuthorID>2</AuthorID>
            <ISBN>1910633461</ISBN>
            <Publisher>Orenda Books</Publisher>
            <PrintLength>276</PrintLength>
            <Year>2016</Year>
            <Genre>Crime &amp; Mystery</Genre>
            <Price>8.49</Price>
      </Book>
      <Book ID="4">
            <Title>Outsider</Title>
            <Author>Stephen King</Author>
            <AuthorID>1</AuthorID>
            <ISBN>1501180983</ISBN>
            <Publisher>Scribner</Publisher>
            <PrintLength>576</PrintLength>
```

```
                <Year>2018</Year>
                <Genre>Horror</Genre>
                <Price>12.79</Price>
            </Book>
        </Books>
```

Scenario 3: Map DB data from different root tables

In the third scenario, we will map data from each root table of the database component to **`Authors.xsd`** (*see screenshot below*). The related tables will be ignored.



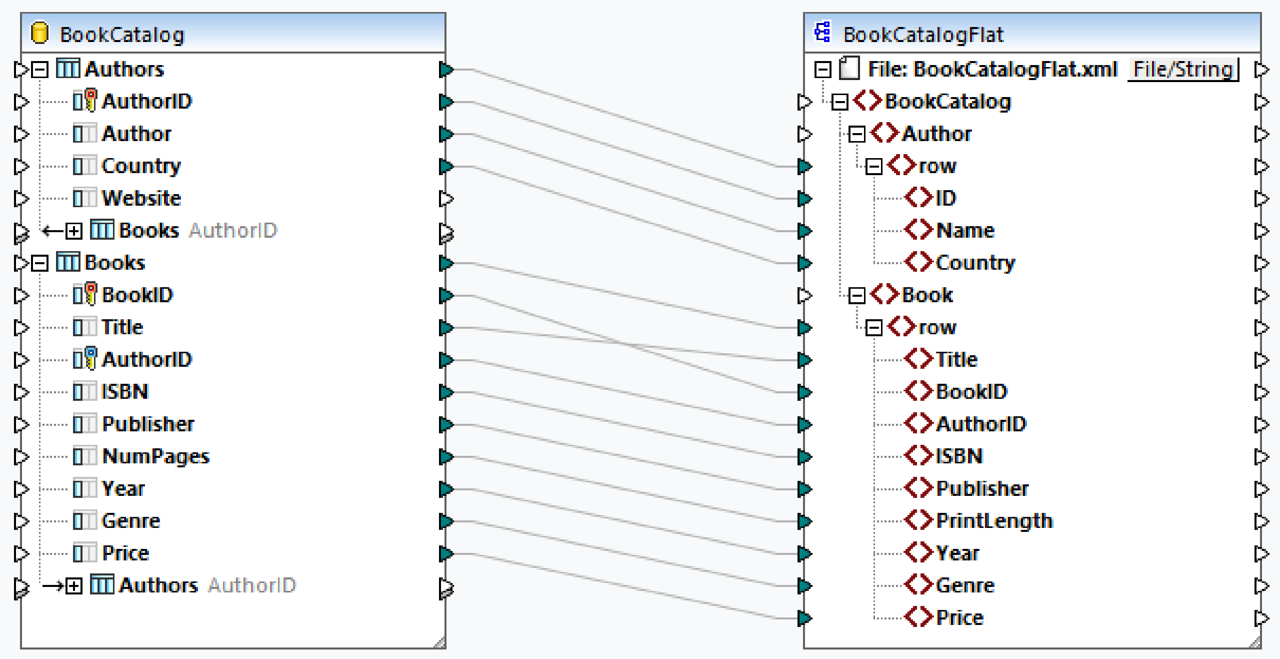
As a result, every single book, regardless of its author, will be listed under each author (*code listing below*).

```
        <Author ID="19">
            <Name>Sebastian Fitzek</Name>
            <Country>Germany</Country>
            <Website>www.sebastianfitzek.com</Website>
            <Publications>
                <Publication ID="1">
                    <Title>Misery</Title>
                    <ISBN>1501143107</ISBN>
                    <Publisher>Scribner</Publisher>
                    <PrintLength>368</PrintLength>
                    <Year>2016</Year>
                    <Genre>Horror</Genre>
                    <Price>11.99</Price>
                </Publication>
                <Publication ID="2">
                    <Title>Nightblind</Title>
                    <ISBN>9781910633113</ISBN>
                    <Publisher>Orenda Books</Publisher>
                    <PrintLength>231</PrintLength>
                    <Year>2016</Year>
                    <Genre>Crime &amp; Mystery</Genre>
                    <Price>9.99</Price>
               </Publication>
               <Publication ID="3">...</Publication>
               <Publication ID="4">...</Publication>
               <Publication ID="5">...</Publication>
               <Publication ID="6">...</Publication>
               <Publication ID="7">...</Publication>
               <Publication ID="8">...</Publication>
          </Publications>
     </Author>
```

Scenario 4: Map DB data to SQL/XML structure

In the fourth scenario, our goal is to map database data to a flat schema structure (SQL/XML Standard). The flat schema model is based on the ISO-ANSI SQL/XML specification INCITS/ISO/IEC 9075-14-2008. The SQL/XML specification defines how to map databases to XML. Relationships are defined in schemas using identity constraints; there are no references to elements. Therefore, the schema is a flat structure which resembles a tree-like view of the database. The specification can be purchased at the ANSI store. For more information, see www.iso.org.

The mapping below shows that database data is mapped from different root tables to a flat SQL/XML structure. The related tables are ignored. It is also possible to map database data from the related tables. However, if there are `Book` records that do not belong to an `Author`, these `Book` records will not be mapped to the target.



As a result, we will get a list of `Author` rows and a separate list of `Book` rows (*screenshot below*).

```
     <Author>
          <row>
               <ID>1</ID>
               <Name>Stephen King</Name>
```

```
                  <Country>US</Country>
            </row>
            <row>
                  <ID>2</ID>
                  <Name>Ragnar Jonasson</Name>
                  <Country>Iceland</Country>
            </row>
            <row>...</row>
            <row>...</row>
      </Author>
      <Book>
            <row>
                  <Title>Misery</Title>
                  <BookID>1</BookID>
                  <AuthorID>1</AuthorID>
                  <ISBN>1501143107</ISBN>
                  <Publisher>Scribner</Publisher>
                  <PrintLength>368</PrintLength>
                  <Year>2016</Year>
                  <Genre>Horror</Genre>
                  <Price>11.99</Price>
            </row>
            <row>
                  <Title>Nightblind</Title>
                  <BookID>2</BookID>
                  <AuthorID>2</AuthorID>
                  <ISBN>9781910633113</ISBN>
                  <Publisher>Orenda Books</Publisher>
                  <PrintLength>231</PrintLength>
                  <Year>2016</Year>
                  <Genre>Crime &amp; Mystery</Genre>
                  <Price>9.99</Price>
            </row>
            <row>...</row>
            <row>...</row>
      </Book>
```

For more information about this scenario, see also the following mapping: **MapForceExamples\DB_Altova_SQLXML.mfd**.

4.2.2.4 Local Relationships

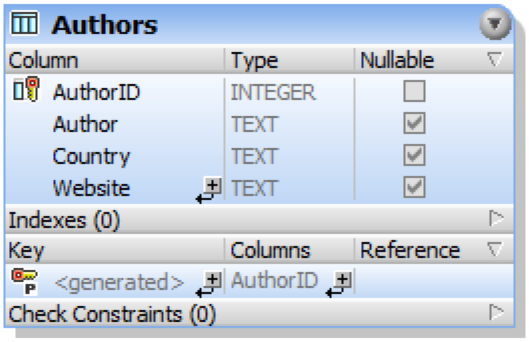
When database tables do not have relationships between them, you can create primary and foreign key relationships between columns of different tables directly in MapForce (i.e., local relationships). Any database columns can be used as primary or foreign keys. You can also create new relations in addition to those existing in the database. Locally defined relationships are saved together with the mapping.

The following table lists all the possible fields between which you can define local relations. Mixed relationships are possible (e.g., mapping the output of a stored procedure to a database column). The fields taking part in the relationship must have the same or compatible data types.

Primary/unique key	Foreign key
• Column of a database table or view • Output parameter or return value of a stored procedure (see also Stored Procedures 297) • Column of a recordset returned by a stored procedure. Applicable if the stored procedure is called as a data source (without parameters) or as a function (with input and output parameters). In order for the recordset to become available for selection, you must execute the stored procedure once, to retrieve the recordset. • Column of a user-defined SELECT statement (see also SQL SELECT Statements as Virtual Tables 243)	• Column of a database table or view • Input parameter of a stored procedure • Input parameter of a user-defined SELECT statement

Example

The **`BookCatalogNoRelation.sqlite`** database has two tables: `Authors` and `Books` (*screenshot below*). At this stage, no foreign-key relationship exists between the tables.



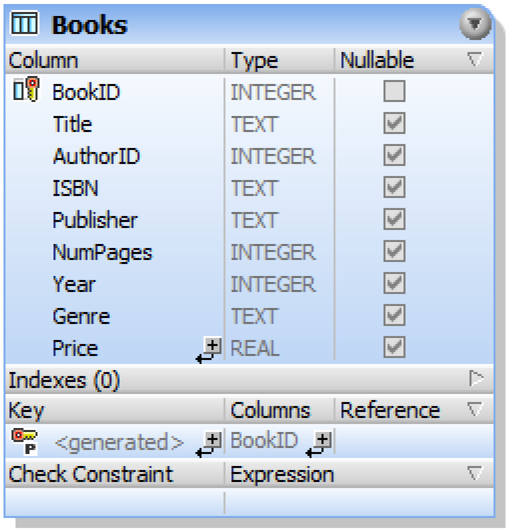
Authors

Column	Type	Nullable
AuthorID	INTEGER	□
Author	TEXT	☑
Country	TEXT	☑
Website	TEXT	☑

Indexes (0)

Key	Columns	Reference
<generated>	AuthorID	

Check Constraints (0)



Books

Column	Type	Nullable
BookID	INTEGER	□
Title	TEXT	☑
AuthorID	INTEGER	☑
ISBN	TEXT	☑
Publisher	TEXT	☑
NumPages	INTEGER	☑
Year	INTEGER	☑
Genre	TEXT	☑
Price	REAL	☑

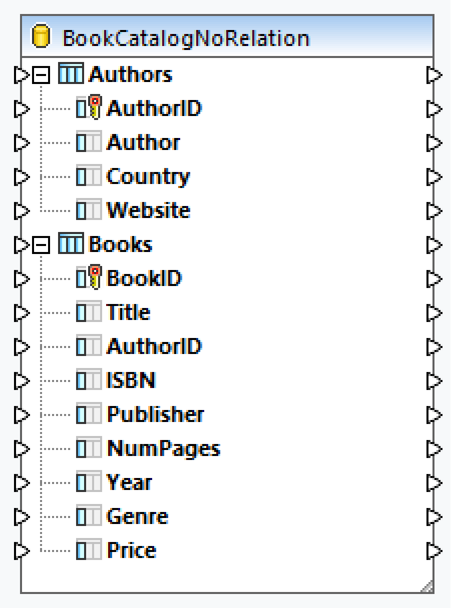
Indexes (0)

Key	Columns	Reference
<generated>	BookID	

Check Constraint	Expression

DB component without relationships

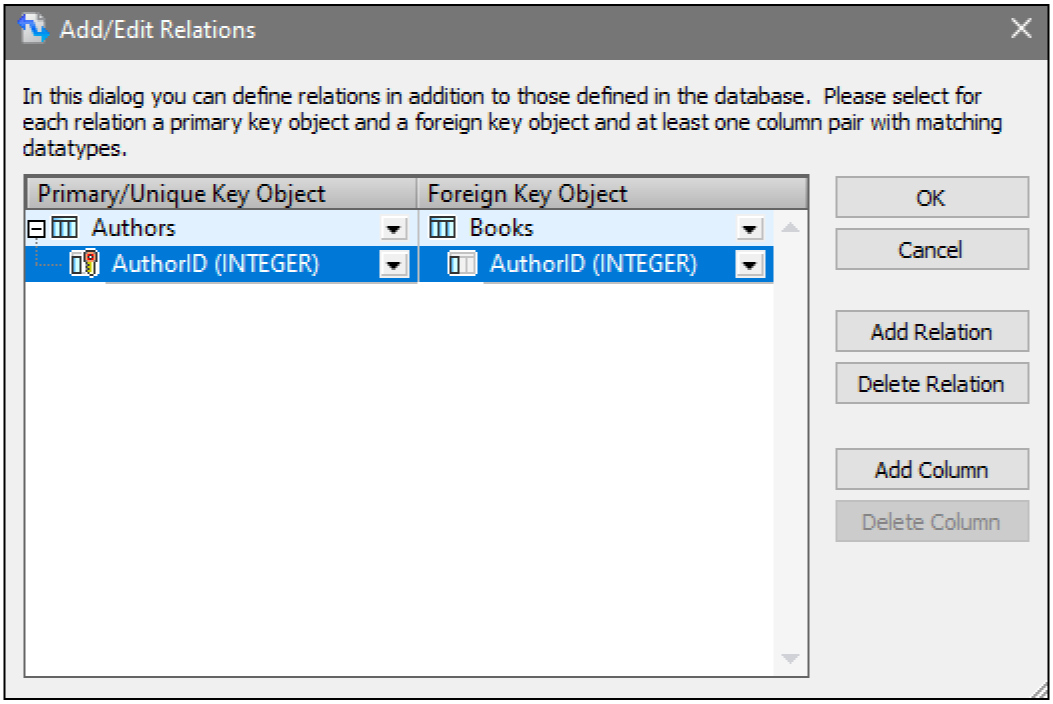
When we insert the database into the mapping, the database component looks as follows:



Local relationship definition

In this example, our goal is to reference the `Authors` table in the `Books` table. Follow the instructions below:

1. Right-click inside the component and select **Add/Remove/Edit Database Objects** from the context menu.
2. Click the **Add/Edit Relations** button in the **Add/Remove/Edit Database Objects** dialog.
3. Click **Add Relation** in the **Add/Edit Relations** dialog (*screenshot below*).
4. Click *[select object]* in the *Primary/Unique Key Object* column and select `Authors`. Then select `AuthorID` in the *[select column]* drop-down list.
5. Click *[select object]* in the *Foreign Key Object* column and select `Books`. Then select `AuthorID` in the *[select column]* drop-down list.
6. Click **OK** to complete the local relation definition.

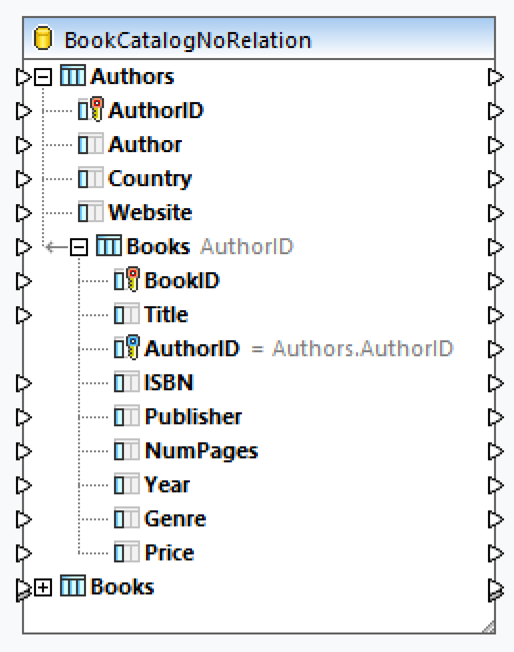


Add/Edit Relations

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
Authors	Books
AuthorID (INTEGER)	AuthorID (INTEGER)

DB component with relationships
As soon as you have finished defining the local relations, the database component becomes available in the mapping area (*screenshot below*). The component displays two possible database structures. In each of these structures, the root table is different. For example, in the expanded structure below, `Authors` is the root table. Depending on your needs, you can map data to and from any of the structures available in the component. You can also mix and match tables from different structures in the component. For more information about these scenarios, see Database Relationships 246.



4.2.2.5 DB-related Functions

When you work with databases, you may need to use various functions to handle null values, generate sequential and unique values, and replace special characters. For more information, see the subsections below.

Handle null values

MapForce provides the following functions to handle null values:

- To check at mapping runtime whether a database field is null, use the `is-null`[606] and `is-not-null`[606] functions. To see if a table has null fields, query it using the Database Browser in MapForce (see DB Query Pane[277]).
- To set a database field to null, use the `set-null`[607] function.
- To replace null database values with a string, use the `substitute-null`[607] function.

For information about handling NULL values in a database, see *Null Equal*[262]. See also *Null Values in Database Components*[139].

Generate sequential and unique values

When you update database records, you might need to create on-the-fly sequential or unique values for those database fields which do not receive any input data from the source. In such cases, you can use the following functions:

- The `auto-number`[541] function can be used to generate primary key values.
- The `create-guid`[632] function creates a globally-unique identifier (as a hex-encoded string) for a specific field.

Note that values for database fields can also be written using database-generated values. This option is available in the Database Table Actions[259] dialog box and is particularly useful when you need to generate primary keys.

Replace special characters

When you update database data, you might need to remove special characters (e.g., carriage return/line feed (CR/LF) characters). To achieve this, you can use the following approaches:

1. You can define a node function for a specific database field (or multiple fields) that you need to process. The node function will receive the value of the database field as input, process this value, and then return the outcome to the mapping. For more information about this approach, see Defaults and Node Functions[442].
2. You can also process database values with the help of MapForce built-in functions. For example, to identify specific characters, including control characters, you can use the `char-from-code`[589] function. To replace values, use the `replace`[656] function.

4.2.2.6 Efficient Usage of DB Resources

To improve efficiency and decrease usage of hardware or network resources, you will typically want to avoid calling the same database multiple times in the same mapping unnecessarily. There may still be situations where you simply cannot avoid calling a database multiple times because of the nature of the mapping, but here are some general considerations:

- If you need only one database call, avoid placing the database component in a parent context that would demand calling the database multiple times. This could happen, for example, if you add a database component inside a user-defined function that receives a sequence of values as input and thus gets called for each item in the sequence. See also User-Defined Functions 469. Variables are typically helpful to gather data into the same context before you pass it on to the target component.
- If you need to aggregate values from a database (for example, to count the number of records using the **count** function), it is recommended to connect the output of the aggregate function to a variable where `compute-when=once`. This prevents repetitive calls to the database, as described in Example: Counting Database Table Rows 368.
- Try to extract all database data in one call (e.g., a SQL-SELECT statement or a stored procedure), as opposed to adding the same database component multiple times in the mapping.
- If you need to extract data from multiple tables or views from the same database, it is advisable to use a Join component 384 (in SQL mode) or an SQL SELECT statement 243. The latter is more convenient if you prefer to write the SQL SELECT statement yourself. If you need to join database data to some non-database data or data from different databases, use non-SQL joins 373. To optimize execution of non-SQL joins in data-intensive mappings, run mappings with MapForce Server Advanced Edition.
- If you need to filter data from a database, it is more efficient to use an SQL-WHERE component 413 instead of a standard filter, since the former component is optimized for working with databases specifically, in the grammar of the corresponding database.

For information about context and general rules and strategies, see Basic Rules and Strategies 77.

4.2.3 DB Table Actions

When you use a database as a target component, you can configure various database table actions. For example, you can insert all records from the source file into your database. You can also decide when to update, delete, and ignore records. This section provides an overview of all the available actions and shows some of the possible scenarios of using table actions.

SQL statements in the output

When you map data to a database and preview the result of the mapping in the Output pane, you will see an SQL script. The script shows pseudo-SQL statements for information purposes only. You must not apply this SQL script manually to the database, using SQL tools other than the following execution engines: MapForce, MapForce Server 799 (both standalone or under FlowForce Server management 802), or the execution environment of the code generated for C++, C#, or Java. The script in the Output pane may contain values that are not understood by external SQL editors.

If you want to apply changes to the database directly from MapForce, open the Output pane and click the **Run SQL/NoSQL-Script** command in the toolbar or in the **Output** menu. This action will modify the database with immediate effect.

When the mapping is executed with MapForce Server (standalone or under FlowForce Server management), the changes to the database are made with immediate effect. The same happens in the generated code: The database changes are made when you compile and run the code (e.g., by clicking the **Run** command in Visual Studio).

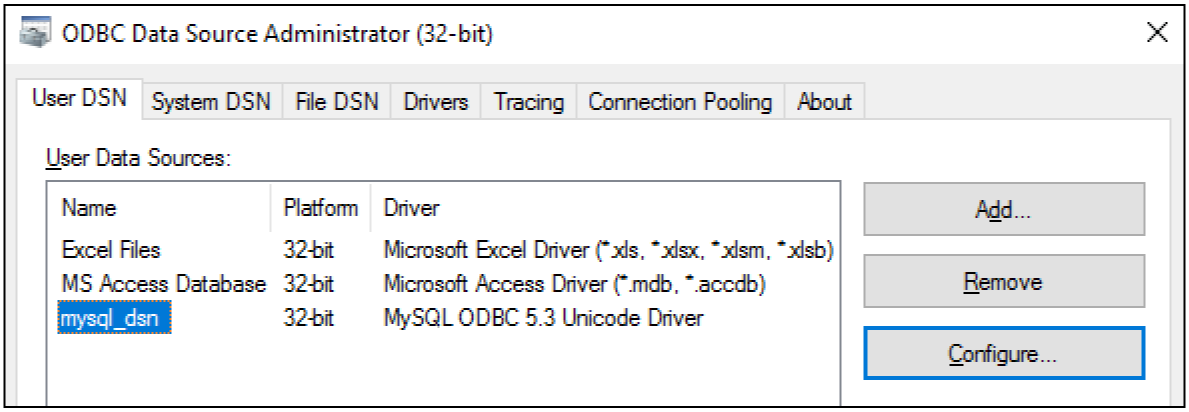
Important note
Your MapForce installation includes several sample databases that are available in the **`MapForceExamples`** folder. It is not recommended to modify any databases in this folder, as this may render several examples unusable. A simple way to avoid overriding original data is to back up the entire **`MapForceExamples`** folder before updating any files in it.

Note about MySQL/MariaDB ODBC

If the target database is MySQL or MariaDB through ODBC, the option *Return matched rows instead of affected rows* must be enabled in the Cursor/Results tab of MySQL ODBC Connector. Alternatively, if you enter the connection string manually in the Database Connection wizard, add `Option=2` to the connection string (e.g., `Dsn=mydsn;Option=2;`).

To enable this option from the MySQL ODBC Connector, take the steps below:

1. Press the **Windows** key and start typing *ODBC*.
2. Run the ODBC Data Sources Administrator (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** (*see below*).



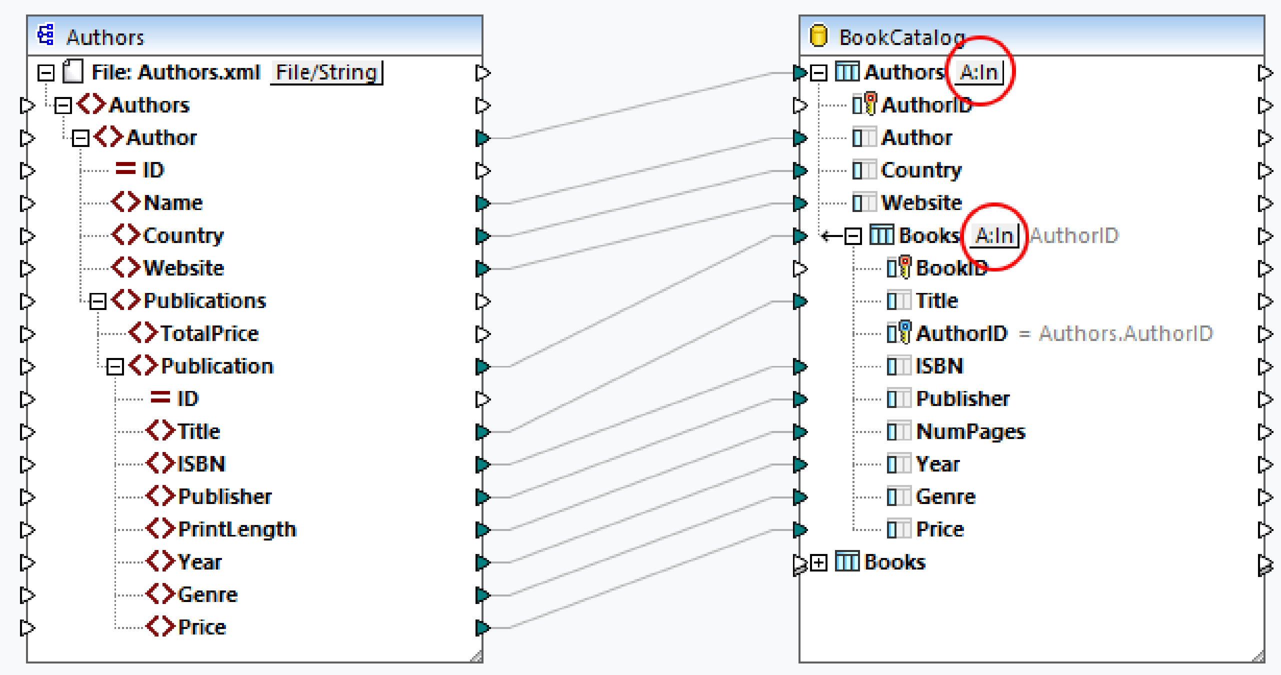
User Data Sources:

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

4. Click **Details >>** to make the advanced options available.
5. Click the *Cursors/Results* tab and select the check box *Return matched rows instead of affected rows*.

4.2.3.1 DB Table Actions: Settings

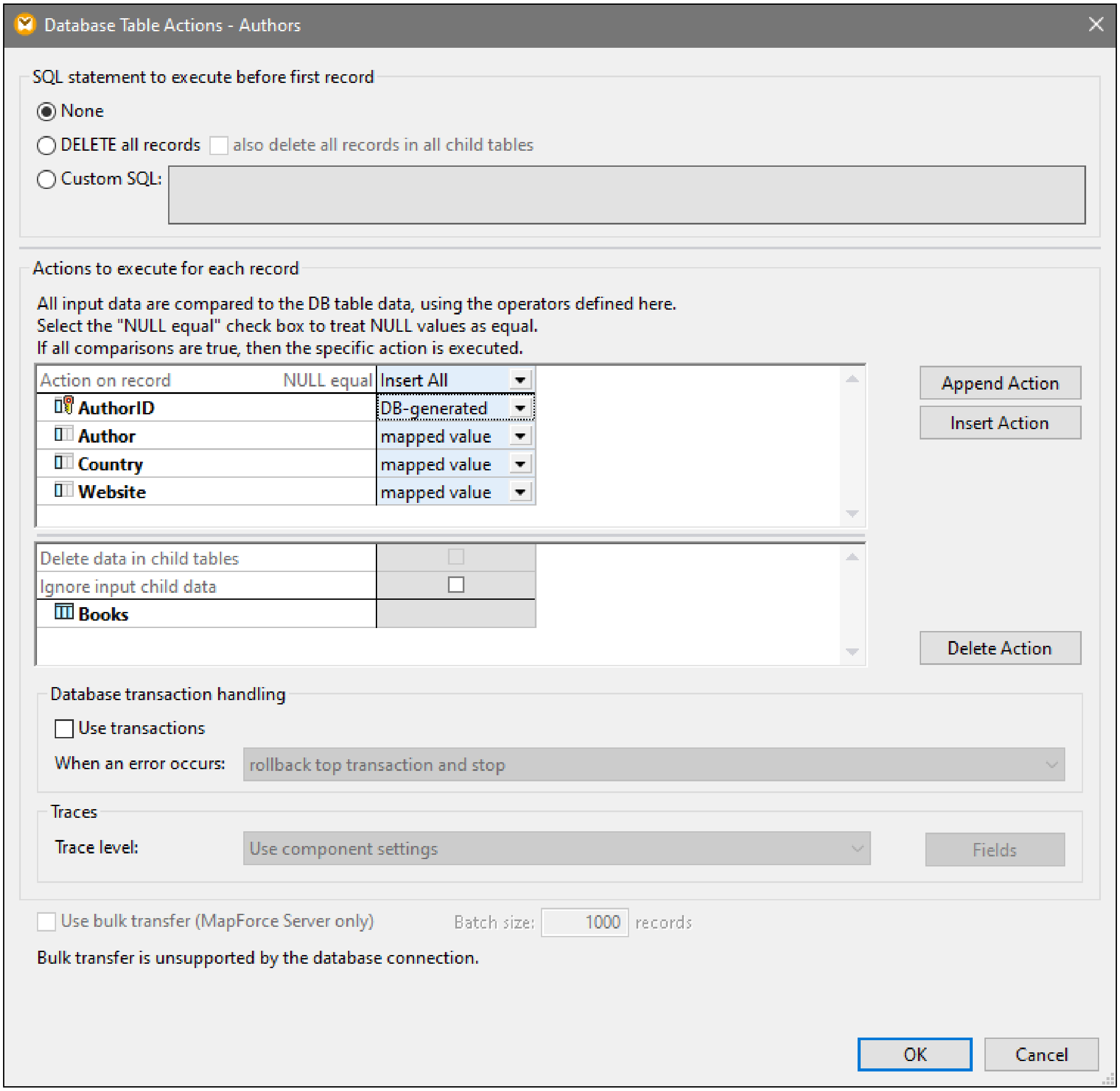
When you map data to a database table, this table will have the **Database Actions** button next to it. The screenshot below shows that we map XML data to two database tables, and each table has its own **Database Actions** button (*circled red below*).



Clicking the **Database Actions** button opens the **Database Table Actions** dialog box (*see below*), in which you can configure various settings, actions and options. The **Database Table Actions** dialog comprises five parts:

1. Actions to be performed before any actions defined for each record
2. Actions defined for each record
3. Database transaction settings
4. Tracing and error logging settings
5. Bulk transfer settings

To find out more about each part of the dialog, see the subsections below.



Database Table Actions - Authors

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
AuthorID		DB-generated
Author		mapped value
Country		mapped value
Website		mapped value

Delete data in child tables	□
Ignore input child data	□
Books	

Database transaction handling

□ Use transactions

When an error occurs: rollback top transaction and stop

Traces

Trace level: Use component settings

□ Use bulk transfer (MapForce Server only) Batch size: 1000 records

Bulk transfer is unsupported by the database connection.

SQL statement to execute before first record

In this section, you can define SQL statements that will be executed *before* any actions defined in the *Actions to execute for each record* section. The following options are available:

- The *None* option means that no action will be performed. This is the default setting.
- The *DELETE all records* option will delete all records from the selected table. Additionally, you can choose to delete all records in all child tables (the *also delete all records in all child tables* check box).
- The *Custom SQL* setting enables you to write a custom SQL statement that will affect the whole table. For example, you can add a statement that will supply tracking information about a mapping. Note that

support for multiple SQL statements in one query depends on the database, connection method, and the driver used.

For more information about these actions, see DB Table Actions: Scenarios 266.

Actions to execute for each record

The *Actions to execute for each record* section allows you to define database actions that will be performed for each record in your database. To manage table actions, click the **Append Action**, **Insert Action**, or **Delete Action** buttons. Multiple actions can be defined if necessary. Any table actions defined after the first *Insert All* or *Insert Rest* action will never be executed, because the subsequent conditions cannot be satisfied. If you have added a table action after the *Insert All* or *Insert Rest* action, a dialog box will inform you that the subsequent table action will be deleted.

For each action, all input data is compared to the database data. If all the comparisons are true, a specific action will be performed. The defined table actions are processed from left to right. For example, if you have defined an *Update If* condition and then an *Insert Rest* condition, the *Update If* action will be processed first. If the *Update If* condition is not satisfied, then the *Insert Rest* action will be performed. If neither condition is satisfied, no action will take place.

Mapped value/DB-generated/max() + 1

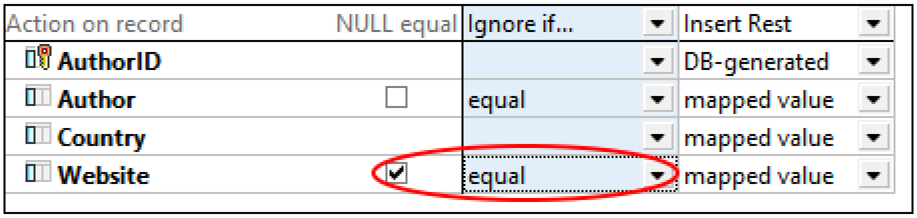
When you set up the *Insert All* or *Insert Rest* action, you can decide how values are to be generated. The following options are available: *mapped value*, *DB-generated*, and *max() + 1*. The *mapped value* option means that source data will be mapped to the database field directly. The *mapped value* option is the standard setting for most database fields.

When autoincrement is set for the primary key(s) in your database, you can choose between a mapped value and a DB-generated value. When there is no autoincrement, you can choose between a mapped value and the *max() + 1* option. The *max() + 1* option generates numeric values based on the existing values in the database. For example, if a table has three records with primary keys `1`, `2`, and `3`, then the primary key of a new record will be `4`.

NULL equal

When you select the *NULL equal* check box next to a record (*screenshot below*), null values in the source record and null values in the target record will be considered equal. Not selecting this check box may lead to incorrect results. You can use the *NULL equal* check box if *all* of the following conditions are true:

- The field for which you want to enable the *NULL equal* option is nullable. This means that this field has been configured in such a way that it can accept NULL values.
- One or more of the following actions have been configured: *Ignore If*, *Update If*, or *Delete If*.
- One or more table actions (e.g., *Ignore If*) have at least one *equal* or *equal (ignore case)* condition (see *Author field in screenshot below*).



Action on record	NULL equal	Ignore if...	Insert Rest
AuthorID			DB-generated
Author	☐	equal	mapped value
Country			mapped value
Website	☑	equal	mapped value

The image above shows that two actions have been set up for the `Authors` table: *Ignore If* and *Insert Rest*. The *Ignore If* condition will compare the `Author` and `Website` values in the source with the `Author` and `Website` values in the database. If these values are the same, these records will be ignored in the database, and the records from the source that have no counterparts in the database will simply be inserted into the database.

The *NULL equal* option has been enabled for the `Website` column. For example, one of `Author` records has a null value in the `Website` field in the source and target. With the *NULL equal* option enabled, this `Author` record will be ignored and will not be inserted into the database. However, if the *NULL equal* check box is not selected, the record will no longer satisfy the *Ignore If* condition and will be inserted into the database.

Child tables
If a foreign-key relationship exists in your database, you will be able to see the names of child tables in the **Database Table Actions** dialog (*Books in Database Table Actions dialog above*). The following options will become available:

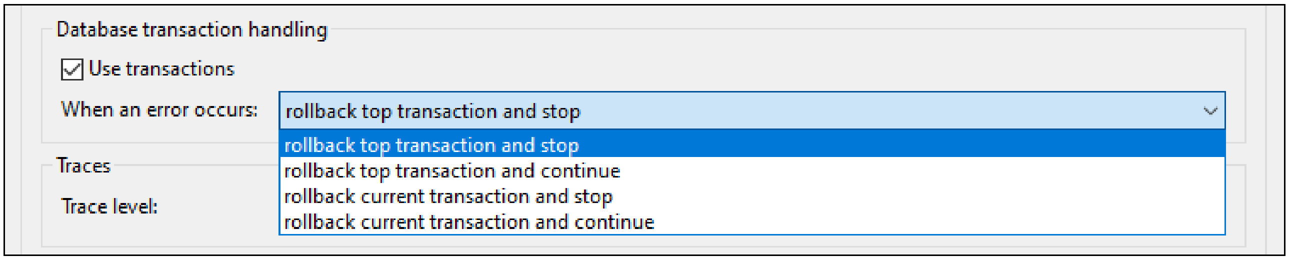
- *Delete data in child tables:* This option may be particularly meaningful when you set up, for example, the *Update If* action for a parent table and the *Insert All* action for its child table. The *Update If* condition will update only those parent records that exist both in the source and the database. The *Update If* condition will prevent you from getting duplicate records in the parent table.

  The *Delete data in child tables* option will delete only those `Book` records whose parent records (`Authors`) satisfy the *Update If* condition (i.e., exist in the source and the database). Deleting child records will prevent you from getting duplicate or orphaned data in the child table. The *Insert All* action defined for the child table will insert only those source child records whose parent records satisfy the *Update If* condition.

- *Ignore input child data:* This option is useful when you want to update only a parent table and leave all its child records unchanged.

Database transaction handling

The *Database transaction handling* section enables you to roll back an operation or sequence of operations (transaction) in case of a database-related error (e.g., NULL values inserted into non-NULL columns). To enable transactions at table-action level, select the *Use transactions* check box in the **Database Table Actions** dialog box (*screenshot below*).


Database transaction handling
☑ Use transactions
When an error occurs:
rollback top transaction and stop
rollback top transaction and stop
rollback top transaction and continue
rollback current transaction and stop
rollback current transaction and continue
Traces
Trace level:

The following transaction-handling options are available:

- *Rollback top transaction and stop:* Since transaction handling can be configured at different levels of the database hierarchy, the *top transaction* can refer to (i) the transaction at database-component level 235 if you have enabled transactions at that level or (ii) to the transaction defined in the topmost table. The changes made to the database will be rolled back for each level of the hierarchy up to the topmost level, and then execution will stop.
- *Rollback top transaction and continue:* Same as above, but the mapping continues to run after the rollback (e.g., to process another target component if such a component exists).

- *Rollback current transaction and stop:* When a database-related error occurs, only the changes enclosed in the current transaction will be rolled back, and the processing of subsequent records will stop. The changes made previously outside of the current transaction will be committed.
- *Rollback current transaction and continue:* Same as above, but the mapping continues to run after the rollback.

For more information about various transaction-handling scenarios, see Transaction Rollback: Scenarios[271].

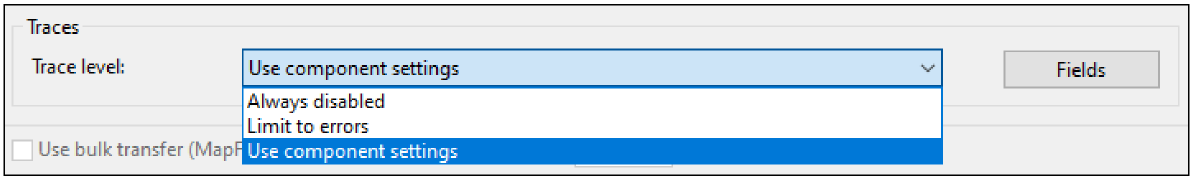
Traces

When a mapping writes data to a database, you can enable database tracing and error logging. Tracing is useful if you want to track all the changes made to the database during the execution of the mapping. The changes made to the database are logged in a trace report. If there are errors during the execution, these errors will also be logged.

For more information about the structure of a trace report, see Trace File[239]. For an example that has tracing enabled, see *Scenario 1* in Transaction Rollback: Scenarios[275].

To enable tracing at table level, take the steps below:

1. Make sure that the tracing level at database-component level[239] is set to *Always* or *Error*.
2. Click the **Table Action** button next to the table for which you want to enable tracing.
3. Select one of the following trace levels in the *Traces* section of the **Database Table Actions** (*screenshot below*): (i) the *Always disabled* option means that no tracing will occur for this table; (ii) the *Limit to errors* option restricts tracing only to error events; (iii) the *Use component settings* option inherits the settings that were defined at component level.



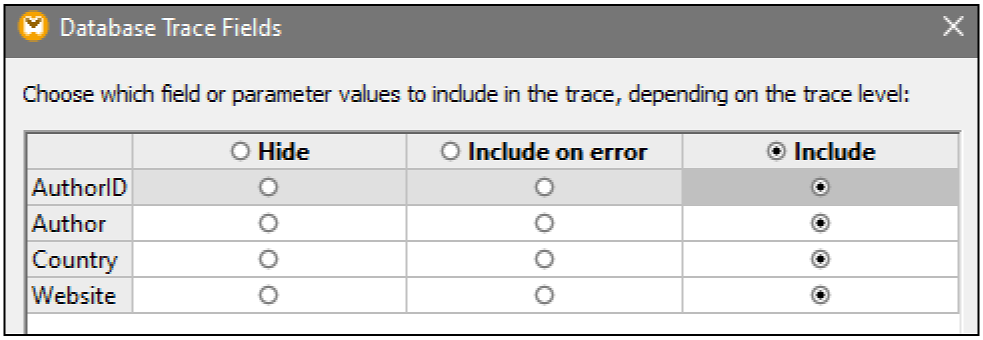
Traces

Trace level: Use component settings

Always disabled
Limit to errors
Use component settings

Tracing at database field level

When you enable tracing at database-component level and table level, all the fields (database columns) are included in the tracing report by default (*screenshot below*). If you want the trace report to include tracing information only about specific database fields, click the **Fields** button in the **Database Table Actions** dialog box. You can choose to hide fields, include them in any case or include them only in case of an error.



Database Trace Fields

Choose which field or parameter values to include in the trace, depending on the trace level:

	○ Hide	○ Include on error	● Include
AuthorID	○	○	●
Author	○	○	●
Country	○	○	●
Website	○	○	●

Use bulk transfer

The *Use bulk transfer* option means that multiple INSERT statements are executed as one query. Using this option dramatically speeds up the Insert process, because only one statement needs to be executed instead of many. The *Use bulk transfer* option is configured in MapForce, but the actual bulk transfer of data occurs when the mapping is run by MapForce Server.

Prerequisites

The bulk transfer option is supported when the following conditions are true:

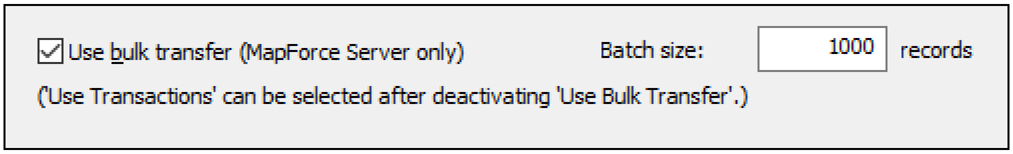
- The mapping transformation language is set to Built-In.
- The mapping is run by MapForce Server[799] (standalone or under FlowForce Server management[802]).
- The MapForce Server license is not limited to single-thread execution on a multi-core machine. This means that the *Limit to single thread execution* option in the Server Management tab of Altova LicenseServer must be disabled.
- The *Insert All* action is set for the relevant database table.
- The table into which you want to bulk-insert data must not have any related tables, views, or stored procedures referencing the table in the mapping.
- The database driver supports bulk-insert operation in WHERE conditions.

Support for the bulk-insert operation depends on the database type and the driver used (*table below*).

DB Type	ADO	ODBC	JDBC	ADO.NET	Native
Access	No	No	n/a	n/a	n/a
DB2	No	Yes	Yes	Yes	n/a
Firebird	n/a	Yes	Yes	No	n/a
Informix	No	Yes	Yes	Yes	n/a
iSeries	No	Yes	Yes	Yes	n/a
MariaDB	No	Yes	Yes	No	n/a
MySQL	n/a	Yes *(MySQL Version 5 or later is required)*	Yes	No	n/a
Oracle	No	Yes	Yes	Yes	n/a
PostgreSQL	n/a	Yes	Yes	n/a	Yes
Progress	n/a	Yes	Yes	n/a	n/a
SQL Server	Yes	Yes	Yes	Yes	n/a
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

Note: To enable bulk-insert support for MySQL and MariaDB via JDBC, use the `rewriteBatchedStatements=true` connection option.

Sample DB
To test the bulk-insert operation, you can use SQL Server and the AdventureWorks database. When you load this database into your mapping and open the **Database Table Actions** dialog, you will be able to specify the batch size (1000 records in our example). The batch size defines the number of records that will be inserted at a time. It is important to note that the *Use bulk transfer* option and the *Use transactions* option are mutually exclusive: When one of these options is enabled, the other option becomes inactive.


☑ Use bulk transfer (MapForce Server only) Batch size: 1000 records
('Use Transactions' can be selected after deactivating 'Use Bulk Transfer'.)

Next step
Now that the bulk-insert option is enabled, the next step is to execute the mapping in MapForce Server[799] (standalone or under FlowForce Server management[802]).

4.2.3.2 DB Table Actions: Scenarios

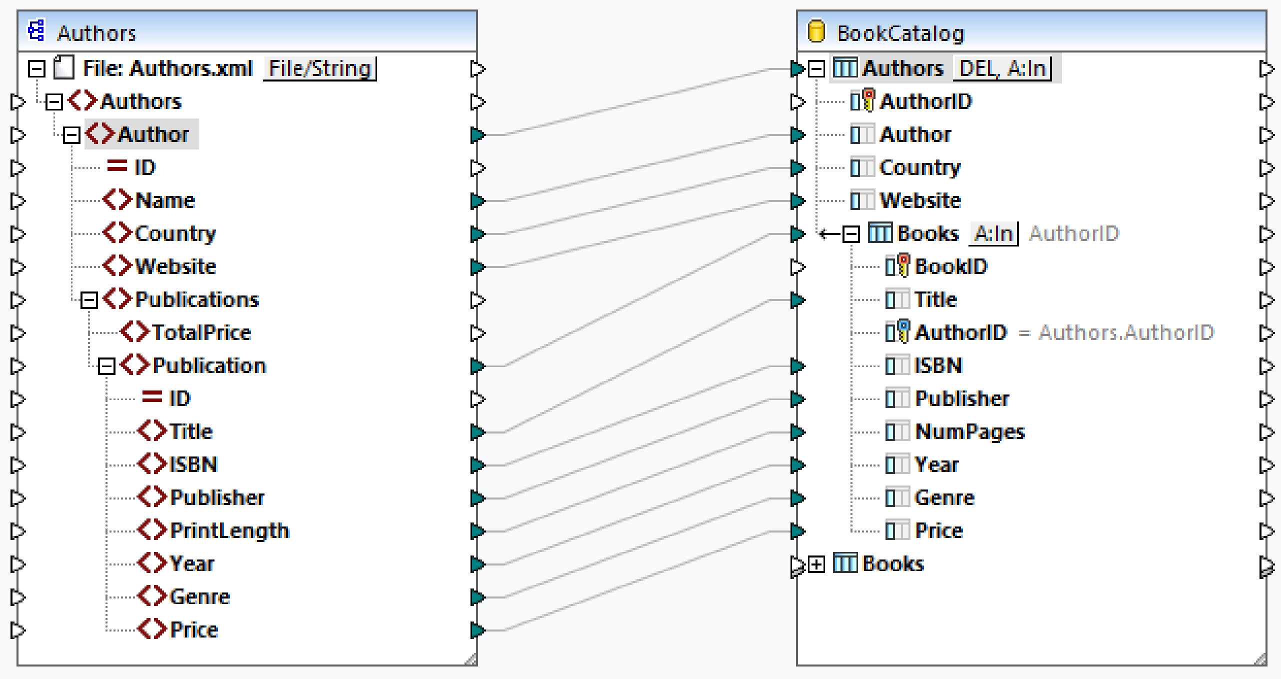
Altova website: MapForce Video Demos

This chapter discusses some of the possible scenarios of using database table actions[259]. All the scenarios use a hierarchical database called **`BookCatalog.sqlite`**. This database has three tables: `Authors` (parent), `Books` (child), `TrackingInfo` (not connected to any other tables). The `Authors` and `Books` tables have a foreign-key relationship. It is important to note that there is autoincrement set for the primary keys in the `Authors` and `Books` tables. The structure of the database is described in the script below:

```
CREATE TABLE
       "main"."Authors" (
 "AuthorID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
       "Author" TEXT,
       "Country" TEXT,
       "Website" TEXT
);

CREATE TABLE
       "main"."Books" (
       "BookID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
       "Title" TEXT,
       "AuthorID" INTEGER,
       "ISBN" TEXT,
       "Publisher" TEXT,
       "NumPages" INTEGER,
       "Year" INTEGER,
       "Genre" TEXT,
       "Price" DECIMAL,
        FOREIGN KEY ("AuthorID") REFERENCES "Authors" ("AuthorID") ON DELETE CASCADE
ON UPDATE NO ACTION
);

CREATE TABLE
        "main"."TrackingInfo" (
        "MappedOn" DATETIME
);
```

Scenario 1: Delete all DB data and insert all source data

In the first scenario, we want to delete *all* the data from the **BookCatalog.sqlite** database and populate the database with *all* the data from the source file. Our mapping looks as follows:

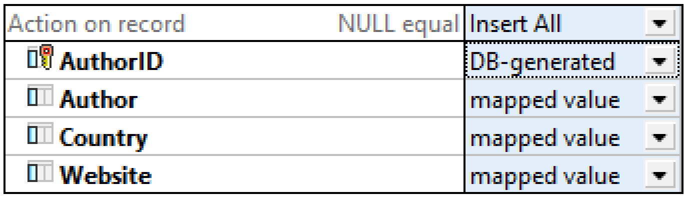


Even though the database has three tables, only `Authors` and `Books` are included in the database component. Since nothing is mapped to the `TrackingInfo` table, this table is absent from the component. Since autoincrement is set for the `Authors` and `Books` tables, we do not need to connect anything to the `AuthorID` and `BookID` columns: these IDs will be generated automatically by the database.

DB table actions for Authors

The table actions for the `Authors` table (*screenshot below*) have been configured in the following way:

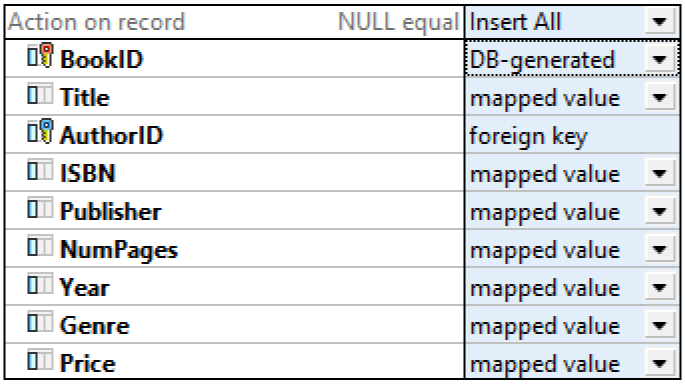
- In the *SQL statement to execute before first record* section, we have set up the DELETE action that will delete all the records from the database, including all the child records.
- In the *Actions to execute for each record* section, we have set up the *Insert All* action.
- Authors' IDs will be generated automatically by the database (the *DB-generated* option in the *Insert All* action).
- The other values will be mapped from the source file.



Action on record	NULL equal	Insert All
AuthorID		DB-generated
Author		mapped value
Country		mapped value
Website		mapped value

DB table actions for Books

For the `Books` table, we have set up the same *Insert All* action (*screenshot below*). Book IDs will be generated by the database. The `AuthorID` column references the primary key in the `Authors` table. The values for this column will be supplied automatically. All the other values will be mapped from the source file.



Action on record	NULL equal	Insert All
BookID		DB-generated
Title		mapped value
AuthorID		foreign key
ISBN		mapped value
Publisher		mapped value
NumPages		mapped value
Year		mapped value
Genre		mapped value
Price		mapped value

Output

The code listing below shows an extract of the output:

```
DELETE FROM "Books"

DELETE FROM "Authors"

INSERT INTO "Authors" ("Author", "Country", "Website") VALUES ('Stephen King', 'US',
'www.stephenking.com')

SELECT "AuthorID" FROM "Authors" WHERE "AuthorID" = last_insert_rowid()
-- >>> %AuthorID1%

INSERT INTO "Books" ("AuthorID", "Title", "ISBN", "Publisher", "NumPages", "Year",
"Genre", "Price") VALUES ('%AuthorID1%', 'Misery', '1501143107', 'Scribner', 368,
2016, 'Horror', 11.99)

INSERT INTO "Books" ("AuthorID", "Title", "ISBN", "Publisher", "NumPages", "Year",
"Genre", "Price") VALUES ('%AuthorID1%', 'Outsider', '1501180983', 'Scribner', 576,
2018, 'Horror', 12.79)
```

Note that the SQL statements in the output are for information purposes only. To execute the SQL statements, open the Output pane and run the **Run SQL/NoSQL-Script** toolbar command. For more information, see *SQL Statements in the Output* 258.

Scenario 2: Update authors and books, insert rest, insert tracking info

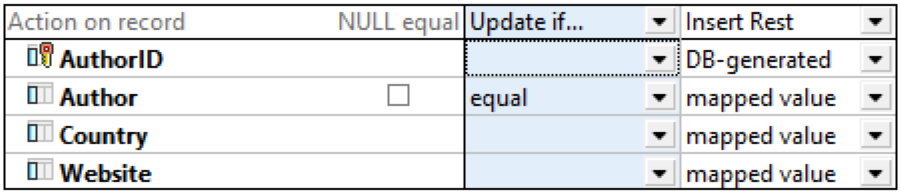
In real-life situations, the database is constantly changing, and several people may be working on the same database. Therefore, deleting all the database records may not be desirable. In the second scenario, we have the following goals:

- To update authors and their books that exist both in the source file and the database
- To insert source records that do not exist in the database
- To supply tracking information about the date and time of the mapping

Our mapping has the same components and connections as in the first scenario. However, the database table actions have been configured differently (*see below*).

DB table actions for Authors

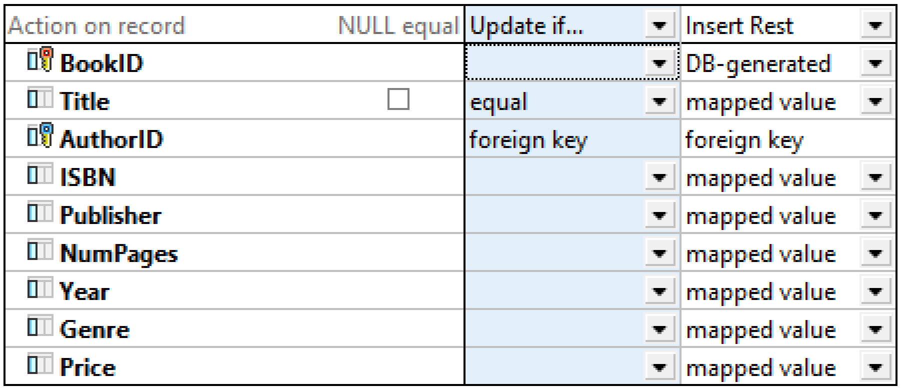
The screenshot below shows the actions defined for the `Authors` table. In the *Update If* condition, we have set up the *equal* value on the `Author` column. This means that if there are authors with the same names in the source and in the database, only these authors' records will be updated in the database. Authors that exist only in the source file will simply be inserted into the database. Authors that exist only in the database will remain unchanged.



Action on record	NULL equal	Update if...	Insert Rest
AuthorID			DB-generated
Author	□	equal	mapped value
Country			mapped value
Website			mapped value

DB table actions for Books

We have the same combination of actions for the `Books` table (*see below*). This time, the equal value is set on the `Title` column. This means that if there are the same books in the source and the database, these book records will be updated in the database. Books that exist only in the source will be inserted into the database. Books that exist only in the database will remain unchanged.



Action on record	NULL equal	Update if...	Insert Rest
BookID			DB-generated
Title	□	equal	mapped value
AuthorID		foreign key	foreign key
ISBN			mapped value
Publisher			mapped value
NumPages			mapped value
Year			mapped value
Genre			mapped value
Price			mapped value

No duplicate records

The major advantage of the *Update If* condition is that it prevents us from getting duplicate records in the database.

Output

The code listing below shows an extract of the output:

```
UPDATE "Authors" SET "Country" = 'US', "Website" = 'www.stephenking.com' WHERE
("Authors"."Author" = 'Stephen King')

SELECT "AuthorID" FROM "Authors" WHERE ("Author" = 'Stephen King')
-- >>> %AuthorID1%

UPDATE "Books" SET "ISBN" = '1501143107', "Publisher" = 'Scribner', "NumPages" =
368, "Year" = 2016, "Genre" = 'Horror', "Price" = 11.99 WHERE ("Books"."AuthorID" =
'%AuthorID1%') AND ("Books"."Title" = 'Misery')

UPDATE "Books" SET "ISBN" = '1501180983', "Publisher" = 'Scribner', "NumPages" =
576, "Year" = 2018, "Genre" = 'Horror', "Price" = 12.79 WHERE ("Books"."AuthorID" =
'%AuthorID1%') AND ("Books"."Title" = 'Outsider')
```

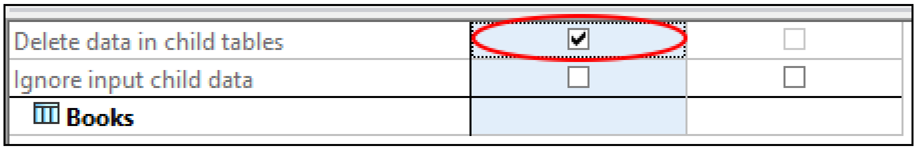
Alternative solution

To avoid duplicate records in the Books table, you can also choose to delete child records [263] (*see below*) and set the *Insert All* condition in the Books table. The actions for the Authors table remain unchanged. In this setup, all the Book records whose authors exist in the source and the database will be deleted from the database. With the *Insert All* condition set in the Books table, the following types of source Book records will be inserted:

- Book records whose authors exist in the source and the database
- Book records of new authors that exist only in the source

If there is a source Book record without a parent, this Book record will also be mapped to the database, and its parent record will be created in the Authors table. The new Author record will receive a new ID (primary key), and all the other fields will receive NULL values. This is possible *only* if all the fields in the Authors table, except for the primary key, are nullable. If the fields are not nullable, you will get an error message saying that the NOT NULL constraint has failed.

Importantly, the results of the main scenario and the alternative solution may diverge. For example, if an author in the database has five book records, and if the same author has only three records in the source, all the five database records will be deleted and will be replaced with three records from the source.



Delete data in child tables	☑	□
Ignore input child data	□	□
Books		

Insert tracking info

When several people are working on the same database, it might be a good idea to know when the mapping was last executed. There could be different possibilities: e.g., you can use various datetime [608] functions; you can also supply a custom SQL statement in the Database Table Actions [261] dialog. In our example, we will add the following SQL statement to the *Actions to execute for each record* section (Authors table):

```
INSERT INTO TrackingInfo VALUES (DATETIME())
```

When you run the SQL script [258], this SQL statement will be executed first, before any statements for database records.

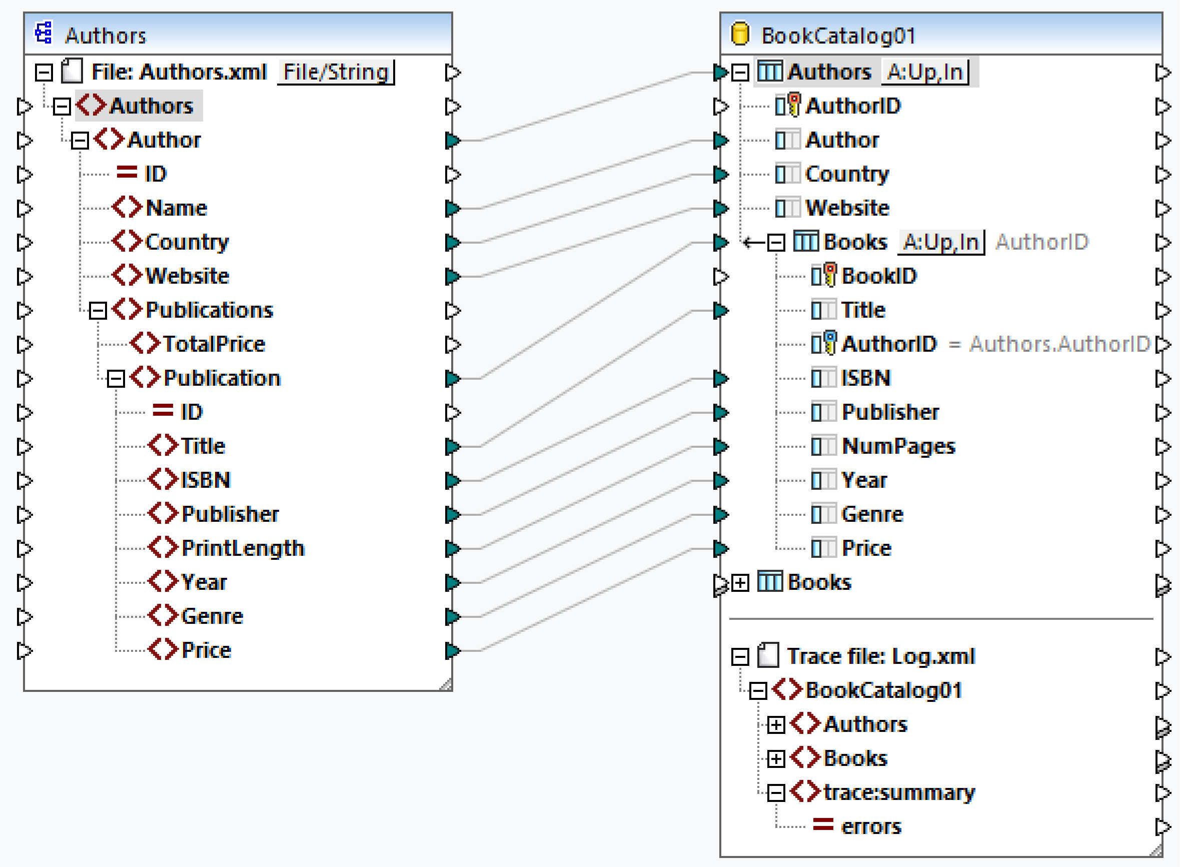
Some other possible scenarios

Instead of the *Update If* condition, you can also set the *Delete If* action. In this case, the `Author` and `Book` records that exist in the source and the database will be deleted, and new records will be inserted into the database (*Insert Rest* action). You can also choose to ignore the same records (*Ignore If* condition) and insert new records into the database (*Insert Rest* action).

4.2.3.3 Transaction Rollback: Scenarios

When you map data to a database, you may encounter various database-related errors (e.g., the database account may not have enough privileges to perform a specific database action). To prevent such errors from aborting the execution of the mapping, you can configure transaction rollback settings, which will enable you to revert changes. You can enable transaction rollback at database-component level 239, at table-action level 263, and at stored-procedure level 309.

This topic describes some of the possible scenarios of transaction rollbacks. All the scenarios in this topic feature a source XML file called **`Authors.xml`** and a target database called **`BookCatalog.sqlite`** (*mapping below*). The `Authors` and `Books` tables have a foreign-key relationship. The `Author` table is the parent of the `Books` table.



In all the scenarios, the same database table actions 259 have been configured:

- The combination of the *Update If* and *Insert Rest* actions in the `Authors` table. The *Update If* condition is set on the `Author` column.
- The combination of the *Update If* and *Insert Rest* actions in the `Books` table. The *Update If* condition is set on the `Title` column.

Besides, tracing 264 has been enabled at component level (set to *Always*), in the `Authors` table (*Use component settings*), and in the `Books` table (*Use component settings*). Transaction rollback settings will vary depending on the scenario.

Broad outline of scenarios
The scenarios described in this topic are outlined below:

- *Scenario 1:* If an `Author` record is faulty, it will be rolled back with all its child records; if only a `Book` record is faulty, only this record will be rolled back.
- *Scenario 2:* If an `Author` record is faulty, it will be rolled back with all its child records.
- *Scenario 3:* If a `Book` record is faulty, only this record will be rolled back.
- *Scenario 4:* If a `Book` record cannot be inserted, its parent record will be rolled back (similar to *Scenario 2*).
- *Scenario 5:* If there is a database-related error, all the changes made to the database will be rolled back.

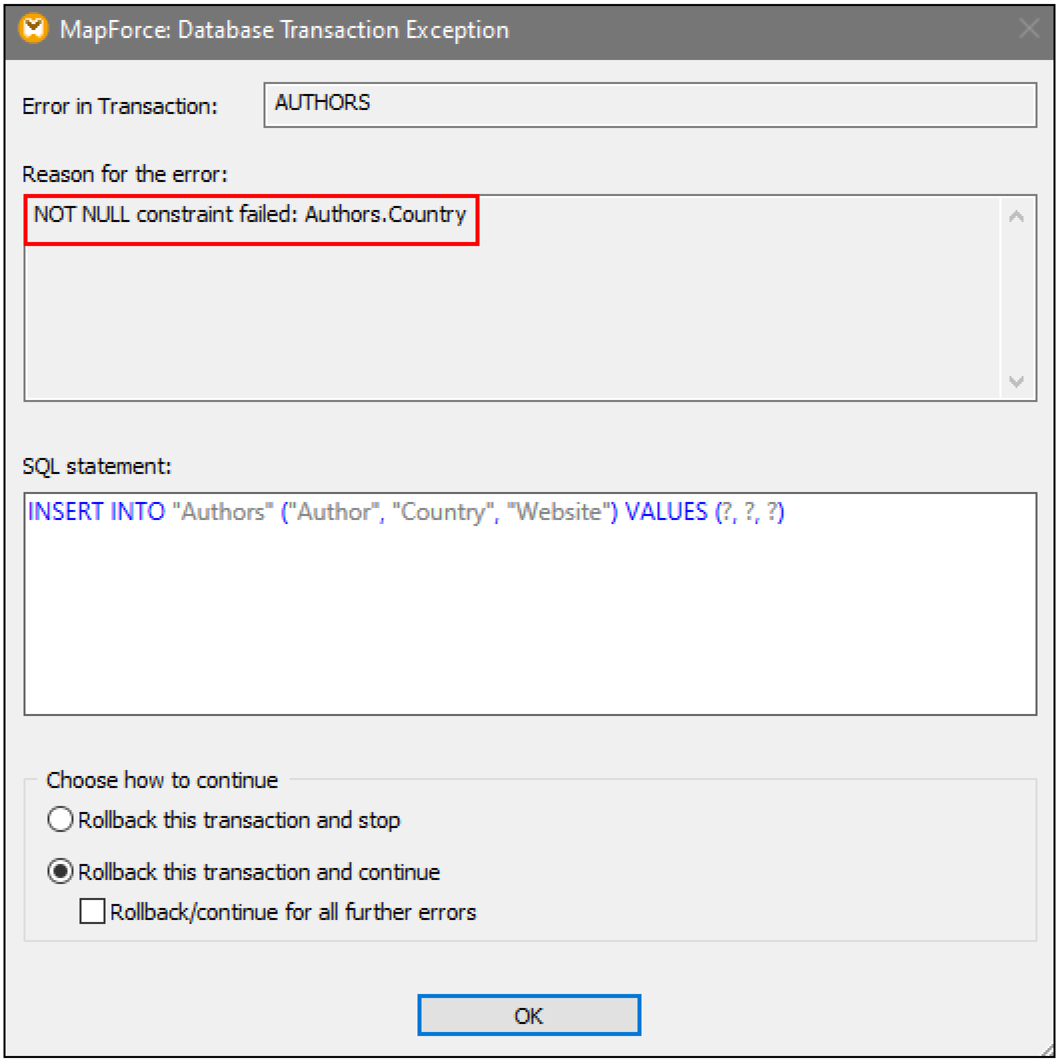
Scenario 1: Roll back current Author and current Book

In the first scenario, we enable transaction handling at table-action level: The *Rollback current transaction and continue* option is set in both tables of the database. The combination of these settings means that (i) if there is a faulty `Author` record, neither this record nor its child `Book` records will be inserted into the database, and then the processing of the next record will start; (ii) if there is a faulty `Book` record, but its parent `Author` record is valid, only the faulty `Book` record will be rolled back, and the processing of the next record will start.

Authors.xml
The **`Authors.xml`** file contains information about authors and their books. One of the `Author` records lacks information about the author's country. The source file also has one `Book` record that has no information about the publisher.

BookCatalog.sqlite
We have set a `NOT NULL` constraint on the `Country` column in the `Authors` table and the `Publisher` column in the `Books` table. Mapping null values to these columns will cause an error.

Output
After running the SQL script 258 in the Output pane, we get the following dialog:



Error in Transaction: AUTHORS

Reason for the error:

NOT NULL constraint failed: Authors.Country

SQL statement:

```
INSERT INTO "Authors" ("Author", "Country", "Website") VALUES (?, ?, ?)
```

Choose how to continue

○ Rollback this transaction and stop

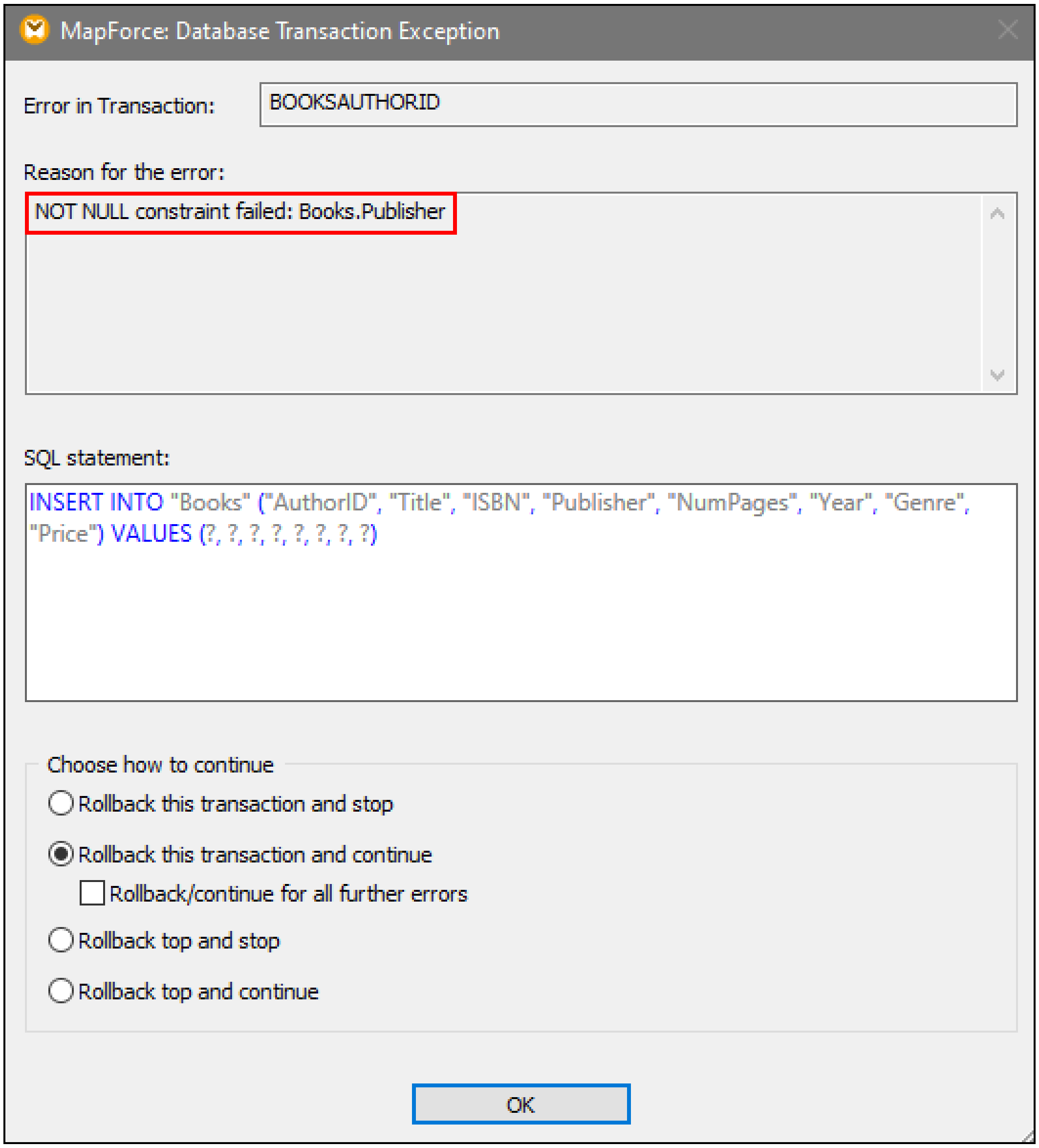
● Rollback this transaction and continue

□ Rollback/continue for all further errors

The **Database Transaction Exception** dialog informs us about a database-related error and the reason for this error. In this example, a null value could not be inserted into the `Country` column of the `Authors` table (*red rectangular above*). This dialog box also allows choosing the next action. By default, the action configured in the Database Table Actions 259 dialog is selected. It is also possible to apply the same setting to all subsequent errors or to roll back the faulty transaction and stop.

After confirming the selection, we are notified about another error that has occurred in the `Books` table (*screenshot below*). This error occurred because the mapping tried inserting absent `Publisher` information from one of our `Book` records into the `Publisher` field of the `Books` table. Since the error occurred in the child table, it is possible to roll back the current transaction (*Rollback this transaction and stop/continue*) or the changes up to the top transaction (*Rollback top and stop/continue*). The top transaction in this context means the parent record (`Author`) of the faulty child record (`Book`).

If there are multiple levels in the hierarchy, a rollback will be done for each level until the rollback reaches the topmost level at which transaction handling has been enabled. Depending on your needs, you can choose to continue or stop the execution of the mapping. The *Rollback top and continue/stop* option is available only when there are nested transactions. For more information about rolling back top transactions, see *Scenario 4* below.



MapForce: Database Transaction Exception

Error in Transaction: BOOKSAUTHORID

Reason for the error:

NOT NULL constraint failed: Books.Publisher

SQL statement:

```
INSERT INTO "Books" ("AuthorID", "Title", "ISBN", "Publisher", "NumPages", "Year", "Genre",
"Price") VALUES (?, ?, ?, ?, ?, ?, ?, ?)
```

Choose how to continue

○ Rollback this transaction and stop

● Rollback this transaction and continue

□ Rollback/continue for all further errors

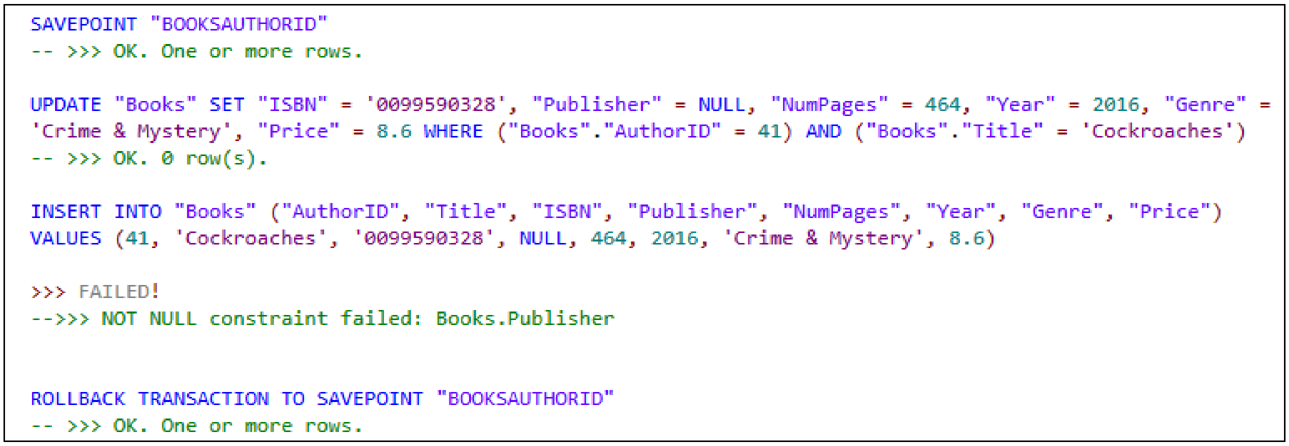
○ Rollback top and stop

○ Rollback top and continue

In this example, we leave the transaction rollback option shown in the screenshot above unchanged. After the processing of all the records has finished, we can see the generated SQL statements in the Output pane, the trace report 239 in XML format, and the data that is now available in the database.

Generated SQL statements

An extract of the generated SQL script is illustrated below. The failed insert operation has been rolled back to the `SAVEPOINT`. The `SAVEPOINT` command represents a point in a transaction, to which the transaction will be rolled back. The `SAVEPOINT` command allows undoing the changes made after the `SAVEPOINT` and restore the transaction to the state at the time of the `SAVEPOINT`.



```
SAVEPOINT "BOOKSAUTHORID"
-- >>> OK. One or more rows.

UPDATE "Books" SET "ISBN" = '0099590328', "Publisher" = NULL, "NumPages" = 464, "Year" = 2016, "Genre" =
'Crime & Mystery', "Price" = 8.6 WHERE ("Books"."AuthorID" = 41) AND ("Books"."Title" = 'Cockroaches')
-- >>> OK. 0 row(s).

INSERT INTO "Books" ("AuthorID", "Title", "ISBN", "Publisher", "NumPages", "Year", "Genre", "Price")
VALUES (41, 'Cockroaches', '0099590328', NULL, 464, 2016, 'Crime & Mystery', 8.6)

>>> FAILED!
-->>> NOT NULL constraint failed: Books.Publisher


ROLLBACK TRANSACTION TO SAVEPOINT "BOOKSAUTHORID"
-- >>> OK. One or more rows.
```

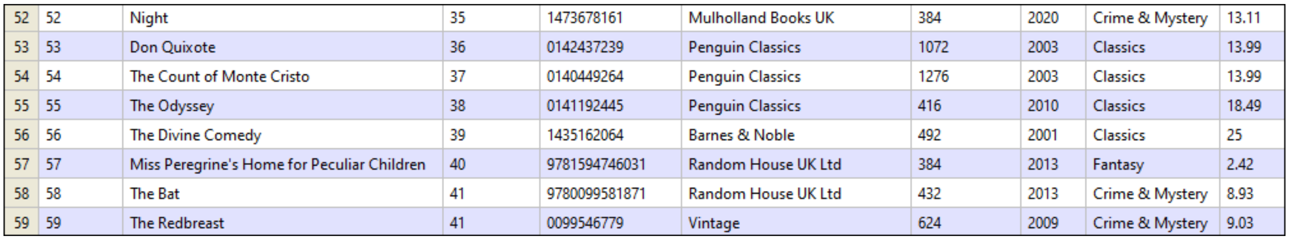
Trace report

An extract of the trace report is provided below. The trace report shows the fields and their values that were traced (e.g., `<Price>8.6</Price>`), provides information about the performed actions (e.g., `insert`) and the error (see the `trace:error` element).

```
<Books>
      <trace:values>
            <Title>Cockroaches</Title>
            <AuthorID>41</AuthorID>
            <ISBN>0099590328</ISBN>
            <Publisher xsi:nil="true"/>
            <NumPages>464</NumPages>
            <Year>2016</Year>
            <Genre>Crime &amp; Mystery</Genre>
            <Price>8.6</Price>
      </trace:values>
      <trace:actions>
            <trace:update rows-affected="0"/>
            <trace:insert>
                  <trace:error code="19" state="1299">NOT NULL constraint failed:
                  Books.Publisher</trace:error>
            </trace:insert>
      </trace:actions>
</Books>
```

Updated data in DB

To see what data is currently available in the database, you can use the DB Query pane 277. The extract of the `Books` table below shows that several new records have successfully been inserted. Since inserting one of Jo Nesbo's books has failed (*trace report above*), only two of his books out of three have been mapped (`The Bat` and `The Redbreast`).



52	52	Night	35	1473678161	Mulholland Books UK	384	2020	Crime & Mystery	13.11
53	53	Don Quixote	36	0142437239	Penguin Classics	1072	2003	Classics	13.99
54	54	The Count of Monte Cristo	37	0140449264	Penguin Classics	1276	2003	Classics	13.99
55	55	The Odyssey	38	0141192445	Penguin Classics	416	2010	Classics	18.49
56	56	The Divine Comedy	39	1435162064	Barnes & Noble	492	2001	Classics	25
57	57	Miss Peregrine's Home for Peculiar Children	40	9781594746031	Random House UK Ltd	384	2013	Fantasy	2.42
58	58	The Bat	41	9780099581871	Random House UK Ltd	432	2013	Crime & Mystery	8.93
59	59	The Redbreast	41	0099546779	Vintage	624	2009	Crime & Mystery	9.03

Other possible scenarios

Some other possible scenarios are described below.

Scenario 2: Roll back current Author and all Author's Books

In this scenario, the *Rollback current transaction and continue* option is set only in the `Authors` table. With this setup, each `Author` record and the corresponding `Book` records are seen as an atomic operation. If there is a faulty `Author` record or at least one of the author's books is faulty, this author's record and its related child records will be rolled back. After that, processing continues with the next `Author` record.

Scenario 3: Roll back only current Book

In this scenario, the *Rollback current transaction and continue* option is set only in the `Books` table. If there is a faulty `Book` record, only this `Book` record will be rolled back, and then processing will continue. If you set the *Rollback top transaction and continue* option in the `Books` table and no other transaction-handling options at parent level, still only the faulty `Book` record will be rolled back. In this case, there is no difference between rolling back the top and current transaction, because there is only one transaction level and no transaction nesting.

Scenario 4: Roll back Author if Book cannot be inserted

In this scenario, the transaction action for the `Authors` table is set to *Rollback current transaction and continue*, and the transaction action for the `Books` table is *Rollback top transaction and continue*. With this setup, if inserting a child record fails, the rollback operation will be done for each level of the hierarchy, up to the topmost level at which transaction handling has been enabled. For example, we want to insert a new `Author` record (`Jo Nesbo`) and its three child records. In one of the `Book` records (`The Cockroaches`), there is no information about the publisher. Since the `Books` table has a NOT NULL constraint on the `Publisher` column, mapping a null value to this field will cause an error. In this scenario, when an error occurs, the faulty `Book` record (`The Cockroaches`) will be rolled back together with the other two `Book` records and the parent record. Then processing will continue with the next `Author` record.

The combination of the transaction-handling options described in this scenario is particularly relevant to more complex, nested structures (e.g., a parent table with two or more child tables).

Scenario 5: Roll back all DB transactions and continue

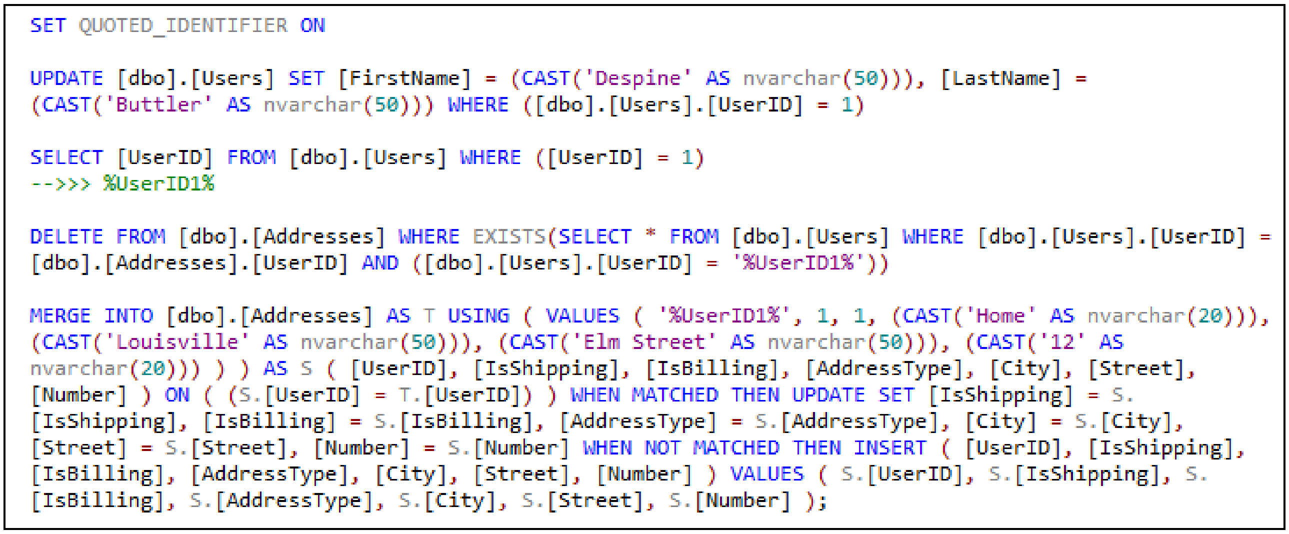
In this scenario, the *Rollback top transaction and continue* option is set at component level[235]. This setting can be particularly useful when you have a chained mapping (e.g., XML-DB-JSON). If there is a database-related error, all the changes made to the database component will be rolled back, and processing will continue with the JSON component.

4.2.3.4 MERGE Statements

For certain mappings, MapForce generates MERGE statements (*screenshot below*) that will be executed against the database at mapping runtime. The advantage of MERGE statements is that they reduce the number of database server calls, since these statements combine the INSERT and UPDATE statements into one. For MERGE statements, the consistency check is done by the database. MERGE statements are supported if:

- the database is one the following: SQL Server 2008 and later, Oracle, DB2, Firebird;
- the target database has the combination of the *Insert If* and *Insert Rest* table actions[262].

If MERGE statements are not supported by your database type, the generated SQL script includes UPDATE statements only. No INSERT statements are visible for preview, since these are executed only if the *Update If* condition is not satisfied.



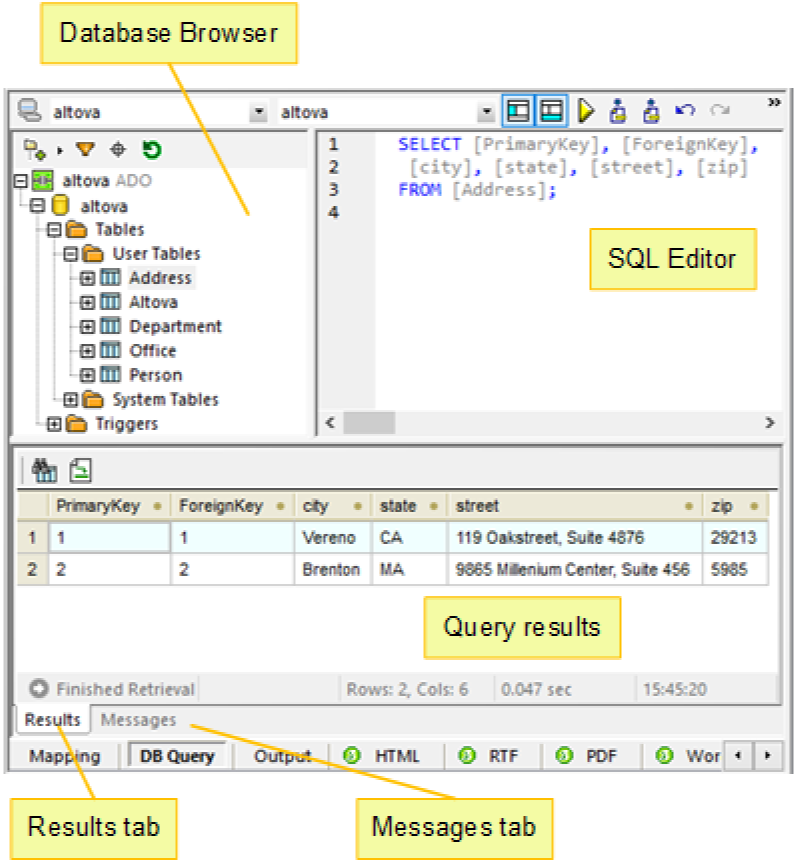
```
SET QUOTED_IDENTIFIER ON

UPDATE [dbo].[Users] SET [FirstName] = (CAST('Despine' AS nvarchar(50))), [LastName] =
(CAST('Buttler' AS nvarchar(50))) WHERE ([dbo].[Users].[UserID] = 1)

SELECT [UserID] FROM [dbo].[Users] WHERE ([UserID] = 1)
-->>> %UserID1%

DELETE FROM [dbo].[Addresses] WHERE EXISTS(SELECT * FROM [dbo].[Users] WHERE [dbo].[Users].[UserID] =
[dbo].[Addresses].[UserID] AND ([dbo].[Users].[UserID] = '%UserID1%'))

MERGE INTO [dbo].[Addresses] AS T USING ( VALUES ( '%UserID1%', 1, 1, (CAST('Home' AS nvarchar(20))),
(CAST('Louisville' AS nvarchar(50))), (CAST('Elm Street' AS nvarchar(50))), (CAST('12' AS
nvarchar(20))) ) ) AS S ( [UserID], [IsShipping], [IsBilling], [AddressType], [City], [Street],
[Number] ) ON ( (S.[UserID] = T.[UserID]) ) WHEN MATCHED THEN UPDATE SET [IsShipping] = S.
[IsShipping], [IsBilling] = S.[IsBilling], [AddressType] = S.[AddressType], [City] = S.[City],
[Street] = S.[Street], [Number] = S.[Number] WHEN NOT MATCHED THEN INSERT ( [UserID], [IsShipping],
[IsBilling], [AddressType], [City], [Street], [Number] ) VALUES ( S.[UserID], S.[IsShipping], S.
[IsBilling], S.[AddressType], S.[City], S.[Street], S.[Number] );
```

If you are updating multiple tables that have parent-child relationships, merges are created only for the deepest child table to which data is mapped. For example, if a database has a parent table called `Authors` and a child table called `Books`, the MERGE statement will be generated only for the `Books` table. For the `Authors` table, UPDATE statements will be generated instead.

With MERGE statements, the Bulk Transfer[265] option is supported only for ODBC and JDBC database connections.

4.2.4 DB Query Pane

MapForce has the DB Query pane (*illustrated below*) that allows you to query and modify your database independently of the mapping process. Such direct queries are not saved together with the mapping. A separate DB Query pane exists for each currently active mapping. In each DB Query pane, you can connect to different databases. Note that the connections created in the DB Query pane are not part of the mapping and are not preserved after you close MapForce unless you define these connections as Global Resources[815].



```
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    0.047 sec    15:45:20
```

The DB Query pane consists of the following parts:

- *Database Browser* 279, which displays connection information and database tables;
- *SQL Editor* 281, in which you write your SQL queries;
- *Results tab* 284, which displays the query results in tabular format;
- *Messages tab* 285, which displays warnings and error messages.

To configure database query settings, use the *Database* section in the **Options** dialog box. For more information, see Database Query Settings 1083.

Connect to a database
Before querying a database 281, you must connect to it. If your mapping already includes a database component, you can select the existing database connection from the upper part of the DB Query pane and start querying your database.

If your mapping does not include a database component, or if you want to connect to a new database, click button (**Quick Connect**) and follow the wizard steps (see Examples 196). You can also select an existing database connection from Global Resources 815.

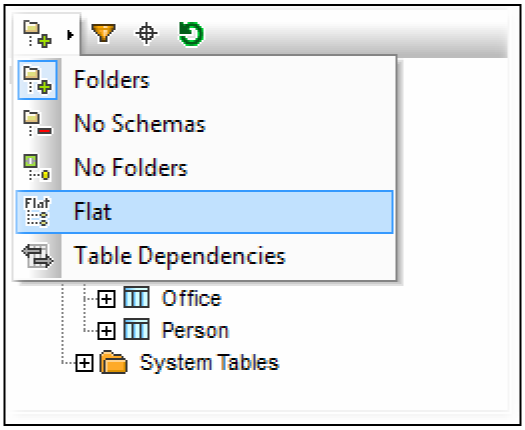
4.2.4.1 Database Browser

This topic explains how to customize the display of the database tree, filter and search database objects. It also describes context menu options that are available for different database objects.

When you are connected to one or several databases, the Database Browser gives the full overview of the objects in each database, including tables, views, procedures, and so on. For databases with XML support, the Database Browser additionally shows registered XML schemas in a separate folder.

DB tree layouts

The Database Browser enables you to customize the display of the database tree. The predefined layouts are available at the top of the Database Browser (*screenshot below*).



To select a layout, click the (**Folders Layout**) button and select the required option from the list. Note that the button changes with the selected layout. The following options are available:

- The *Folders* layout organizes database objects into folders based on the object type (*default setting*).
- The *No Schemas* layout is similar to the *Folders* layout, except that there are no database schema folders.
- The *No Folders* layout displays database objects in a hierarchy without using folders.
- The *Flat* layout displays database objects by type (e.g., all columns are displayed in a separate `Columns` folder).
- The *Table Dependencies* layout categorizes tables based on their relationships with other tables (e.g., tables with foreign keys, referenced tables).

In addition to layout navigation, you can use the Database Browser for the following tasks:

- Creating SQL statements 282
- Filtering and searching database objects (*see subsections below*)
- Sorting tables into *System* and *User* tables (*see below*)
- Refreshing the root object of the active data source (the button)

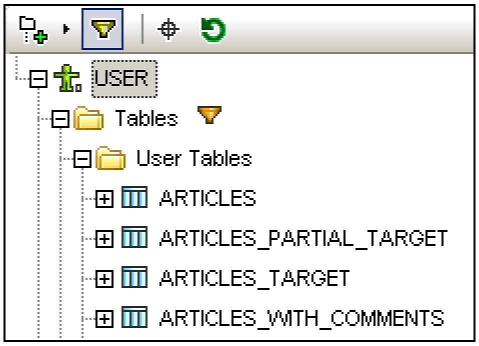
To sort tables into *User* and *System* tables, right-click the `Tables` folder in the Database Browser and select **Sort into User and System Tables** from the context menu. This function is available when one of the following layouts is selected: *Folders*, *No Schemas* or *Flat*.

Filter database objects

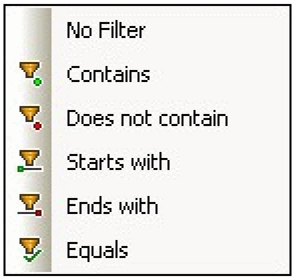
You can filter any database objects (schemas, tables, views, etc.) by name or part of a name. Objects are filtered as you type in the characters. Filtering is case-insensitive by default. Filtering is not supported if you have selected the *No Folders* layout.

To filter database objects, take the following steps:

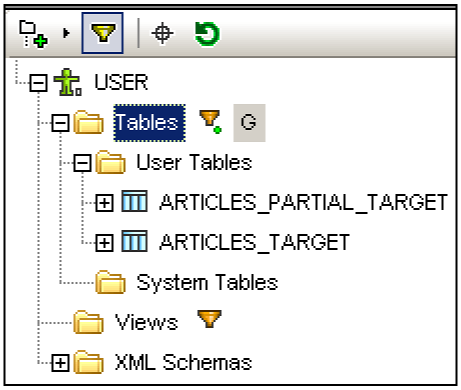
1. Click button at the top of the Database Browser. Filter icons appear next to all folders in the currently selected layout (*screenshot below*).



2. Click the filter icon next to the folder you want to filter and select the filtering option from the context menu (e.g., **Contains**).



3. Enter the search text (e.g., *G*) in the empty field which appears next to the filter icon. The results are adjusted as you type (*screenshot below*).



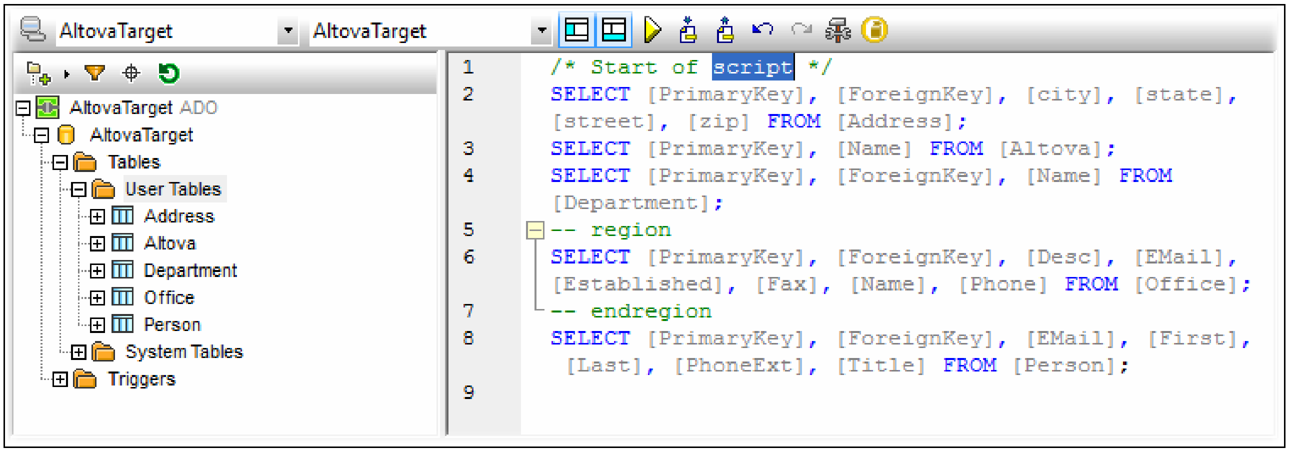
Search database objects

To find a specific database object, you can use the filtering functionality (*see above*) or the **Object Locator**. To find database objects using the Object Locator, follow the instructions below:

1. Click the ⌖ button at the top of the Database Browser.
2. Enter some search text in the text bar.
3. Press **Enter** or click an object in the list to select it.

4.2.4.2 SQL Editor

Once you are connected to your database, you can use the SQL Editor to write and execute SQL statements. The SQL Editor displays any SQL statements that you may have generated automatically, loaded from existing SQL scripts, or written manually. The SQL Editor supports auto-completion[235], regions, and line/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];
```

SQL Editor toolbar buttons

The SQL Editor toolbar has the following buttons:

◧	The **Toggle Browser** button makes the Browser pane visible/invisible.
⊟	The **Toggle Result** button makes the Result visible/invisible.
▷	The **Execute Query** button executes the SQL statements that are currently selected. If multiple statements exist and none are selected, only the result of the last statement will be shown in the Results tab. For multiple results, MapForce will suggest opening these results in Altova DatabaseSpy.
↶	The **Undo** button allows you to undo an unlimited number of edits in the SQL window.

	The **Redo** button allows you to redo previously undone commands.
	The **Import SQL file** command opens an external SQL script, which can then be executed.
	The **Export SQL file** command saves an SQL script to a desired location.
	The **Open SQL script in DatabaseSpy** command opens the current SQL script in Altova DatabaseSpy (must be installed).
	The **Options** button opens the Options[1083] dialog box that allows you to define general database query settings as well as SQL Editor settings.

Create SQL statements

You can query your database using the following methods:

- You can import an SQL script or copy some SQL statements and paste them into the SQL Editor (The **Import SQL file** toolbar command).
- You can write SQL statements in the SQL Editor manually.
- You can also right-click an object in the Database Browser[279] and generate a query (typically, SELECT).

To generate an SQL SELECT statement in the Database Browser, do one of the following:

- Press and hold a database object (e.g., a table) or a folder in the Database Browser and drag this object of folder into the SQL Editor.
- Right-click a database object in the Database Browser and select **Show in SQL Editor | SELECT**.

Execute SQL statements

The SQL statements in the SQL Editor can be executed with immediate effect. To execute an SQL statement, click the ▷ button. The result of the SQL query and the number of affected rows are displayed in the Messages tab of the DB Query pane.

When multiple SQL statements appear in the SQL Editor, only the result of the selected statement or the result of the last statement (if no statements have been selected) will be displayed in the Results tab. In case there are multiple results, MapForce will suggest opening them in Altova DatabaseSpy.

Import/Export SQL scripts

You can import and export SQL scripts. To import an external SQL file, click the toolbar button and select the SQL file you want to open. To export the contents of the SQL Editor pane to an SQL file, click the toolbar button and enter the name of the SQL script.

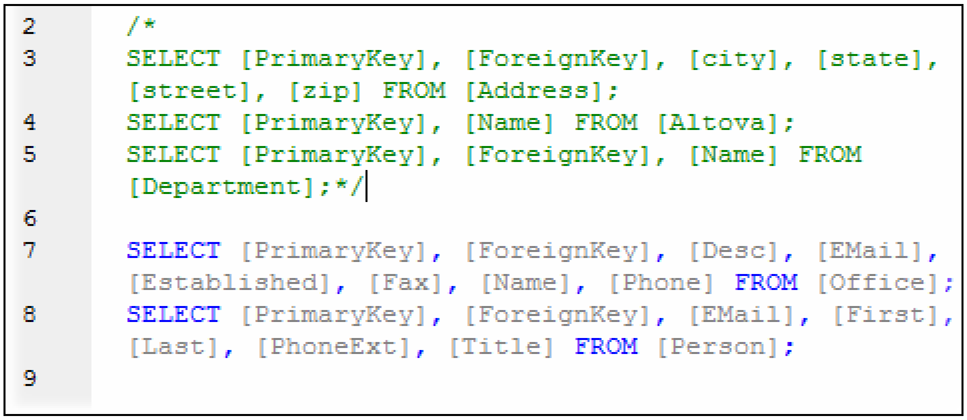
Add/Remove SQL line/block comments

The SQL Editor allows you to comment out statements, parts of statements, or groups of statements. Such statements are skipped when the SQL script is executed. You can insert a line or a block comment. The line comment indicates that the current line or the remaining part of it is commented out. The block comment can span multiple lines. The block comment can also be used to comment out individual words.

To insert a line/block comment, take the following steps:

1. Select a statement or part of a statement.
2. Right-click the selected statement and select **Insert/Remove Line/Block Comment** from the context menu.

The screenshot below illustrates a block comment (*green text*).



```
/*
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 remove a line/block comment, take the steps below:

1. Select the part of the statement that is commented out.
2. Right-click the selected part and select **Insert/Remove Block (or Line) Comment** from the context menu.

You can also remove the comment characters manually.

Add bookmarks

Bookmarks are used to mark items of interest in scripts. To add a bookmark, right-click the line you want to have bookmarked and select **Insert/Remove Bookmark** from the context menu. The bookmark icon ● is displayed in the margin at the beginning of the bookmarked line.

To remove a bookmark, right-click the line from which you want to remove the bookmark and select **Insert/Remove Bookmark** from the context menu. To remove all bookmarks, right-click anywhere inside the SQL Editor and select **Remove all Bookmarks** from the context menu.

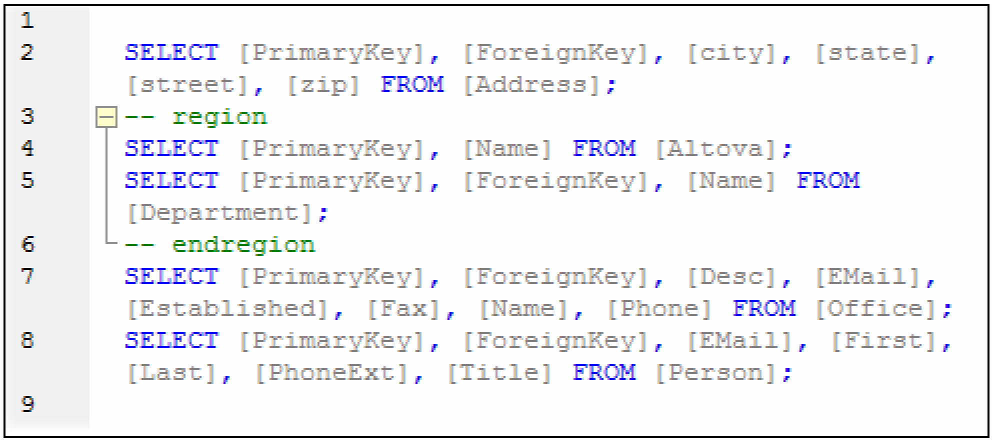
To jump to the next/previous bookmark, right-click the relevant line and select **Go to Next/Previous Bookmark**.

Insert SQL 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. You can change the name of a region by adding descriptive text to the `--region` comment. The word *region* must not be deleted.

To add a region, follow the instructions below:

1. Select the statements you want to transform into a region.
2. Right-click the selected statements and select **Insert Region** from the context menu. An example of a region is illustrated below.

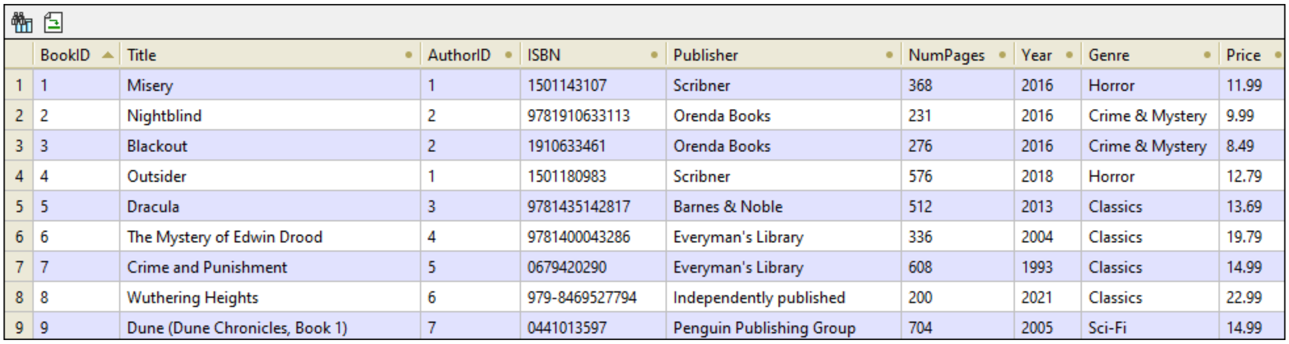


```

    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];

```

To remove a region, delete the `-- region` and `-- endregion` comments.

4.2.4.3 Results Tab

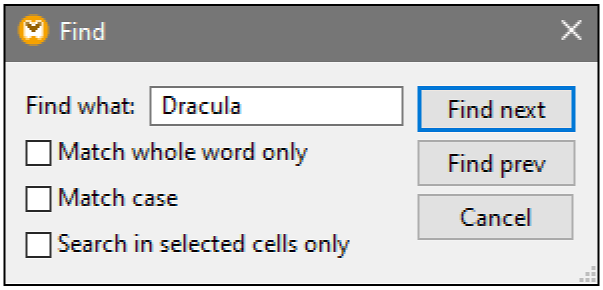
The Results tab of the DB Query pane shows the recordset retrieved as a result of a database query (*screenshot below*).



	BookID	Title	AuthorID	ISBN	Publisher	NumPages	Year	Genre	Price
1	1	Misery	1	1501143107	Scribner	368	2016	Horror	11.99
2	2	Nightblind	2	9781910633113	Orenda Books	231	2016	Crime & Mystery	9.99
3	3	Blackout	2	1910633461	Orenda Books	276	2016	Crime & Mystery	8.49
4	4	Outsider	1	1501180983	Scribner	576	2018	Horror	12.79
5	5	Dracula	3	9781435142817	Barnes & Noble	512	2013	Classics	13.69
6	6	The Mystery of Edwin Drood	4	9781400043286	Everyman's Library	336	2004	Classics	19.79
7	7	Crime and Punishment	5	0679420290	Everyman's Library	608	1993	Classics	14.99
8	8	Wuthering Heights	6	979-8469527794	Independently published	200	2021	Classics	22.99
9	9	Dune (Dune Chronicles, Book 1)	7	0441013597	Penguin Publishing Group	704	2005	Sci-Fi	14.99

Toolbar buttons for navigation

At the top of the Results tab, there are two toolbar buttons that enable you to navigate the query results. The (**Find**) toolbar button allows you to run a search in the retrieved results. When you press this button, the **Find** dialog pops up (*screenshot below*). This dialog box allows you to configure various search parameters (e.g., match the whole word). Use the **Find next** and **Find prev** buttons to switch between the occurrences of the search term.



Find what: Dracula

□ Match whole word only

□ Match case

□ Search in selected cells only

The (**Go to statement**) toolbar command 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.

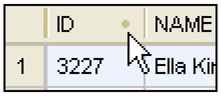
Select data
To select data in the query results, you can use the following methods:

- You can click a column's header to select the entire column.
- You can click a row's number to select the entire row.
- You can press and hold the **Shift** key to select a range of cells.
- You can right-click a cell of interest and select various options from the context menu: **Selection | [Row | Column | All]**.
- You can also select all the cells by clicking somewhere inside the table and pressing **Ctrl + A**.

Copy data
To copy the selected cells, press **Ctrl + C**. Alternatively, right-click the selected cells and select **Copy selected cells** or **Copy selected cells with header** from the context menu.

Sort data
To sort data, you can choose one of the following options:

- Right-click anywhere in the column of interest and select **Sorting | [Ascending | Descending]** from the context menu.
- Click the dot icon (*screenshot below*) in the column header until you have selected the relevant option: the arrow pointing up sorts data in ascending order, the arrow pointing down sorts data in descending order.

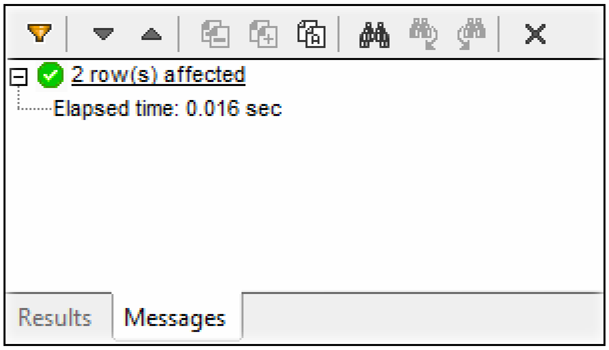


	ID	NAME
1	3227	Ella Ki

To restore the default sort order, right-click anywhere in the table and choose **Sorting | Restore default** from the context menu.

4.2.4.4 Messages Tab

The Messages tab of the DB Query pane provides information about the last executed SQL statement(s) and can display errors and warnings.


2 row(s) affected

Elapsed time: 0.016 sec

Toolbar buttons

You can use different filters to customize the view of the Messages tab. The buttons at the top of the Messages tab (*screenshot above*) are used to navigate messages, copy text to the clipboard, and hide certain parts of the message. When you right-click anywhere inside the Messages tab, you will also be able to see these options in the context menu.

▽	The **Filter** button opens a menu in which you can select types of messages you would like to see in the Messages tab. The following options are possible: *Check All*, *Uncheck All*, *Summary*, *Success*, *Warning*, *Error*, *Autoinsertion*, and *Progress*. The *Autoinsertion* option refers to messages that appear when SQL statements or SQL constructs are inserted automatically in the SQL Editor. The *Progress* option informs about the database connection result as well as the outcome of SQL parsing and data-structure loading.
▼	The **Next** button jumps to and highlights the next message.
▲	The **Previous** button 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**
🔍	The **Find** command opens the **Find** dialog box that allows you to configure search criteria.
🔍	The **Find previous** command jumps to the previous occurrence of the search string specified in the **Find** dialog box.
🔍	The **Find next** command jumps to the next occurrence of the search string specified in the **Find** dialog box.
✕	The **Clear** button removes all messages from the Message tab of the SQL Editor.

4.2.5 Map XML Data to/from DB Fields

MapForce enables you to map data to or from database fields (columns) that store XML content. This means that XML data stored by the database field (column) can be extracted and written to any other structure supported by MapForce, and the other way round. You can map data as follows:

1. To or from fields of a dedicated XML type (for example, `Xml` in SQL Server, `XMLType` in Oracle). Reading or writing XML to/from dedicated XML fields is applicable to databases that have native support for XML (such as IBM DB2, Oracle, and SQL Server).
2. To or from text fields storing XML content (for example, `Text, Varchar`). This applies to any database where the text field has sufficient length to store an XML document.

In either of the cases, a valid XML schema must exist for each database column to/from which you want to map data. When a database column stores XML, MapForce provides you with the choice to assign an XML schema directly from the database (if supported by the database), or select a schema from an external file. You can assign one XML schema per database column. If the schema has multiple root elements, you can select a single root element of that schema.

When XML is stored as a string field in a database, the character encoding of the XML document is that of the underlying string field. If the database field does not store text as Unicode, some characters cannot be represented.

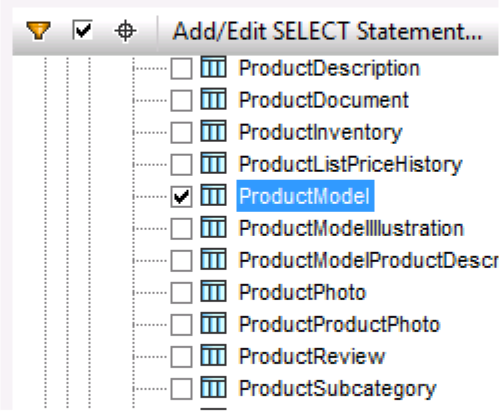
Some databases support XML encoding for XML fields (which may not necessarily be the same as that of the database character set). If supported by the database, the XML document encoding declaration is assumed to be the one declared in the XML field. For information about the XML encoding support provided by various databases, refer to their documentation.

4.2.5.1 Assigning an XML Schema to a Database Field

This topic illustrates how to assign a schema to a field that is natively defined as XML type in the database. The instructions below use SQL Server 2014 and the AdventureWorks 2014 database. The latter can be downloaded from the AdventureWorks samples page on GitHub (https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks). Note that mapping of data to or from XML fields works in the same way with other database types that support XML fields.

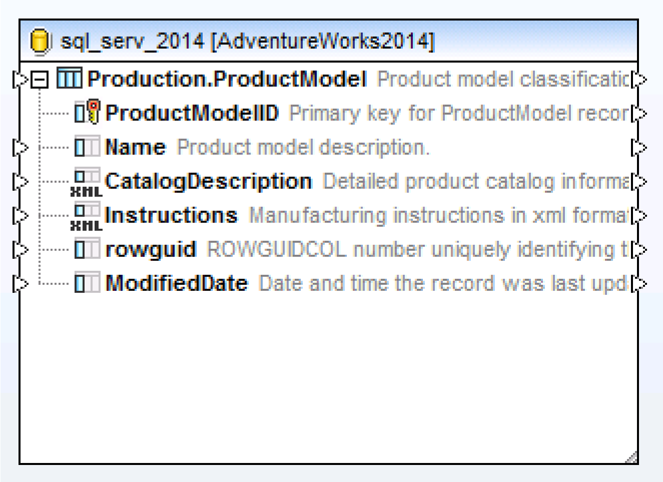
To add the Adventure Works 2014 database as a mapping component:

1. On the **Insert** menu, click **Database**, and follow the wizard to connect to the database using your preferred method (ADO or ODBC). For more information, see ADO Connection[176] and ODBC Connection[190]. 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[179]).
2. On the **Insert Database Object** dialog box, expand the **Production** schema, and then select the **ProductModel** table.

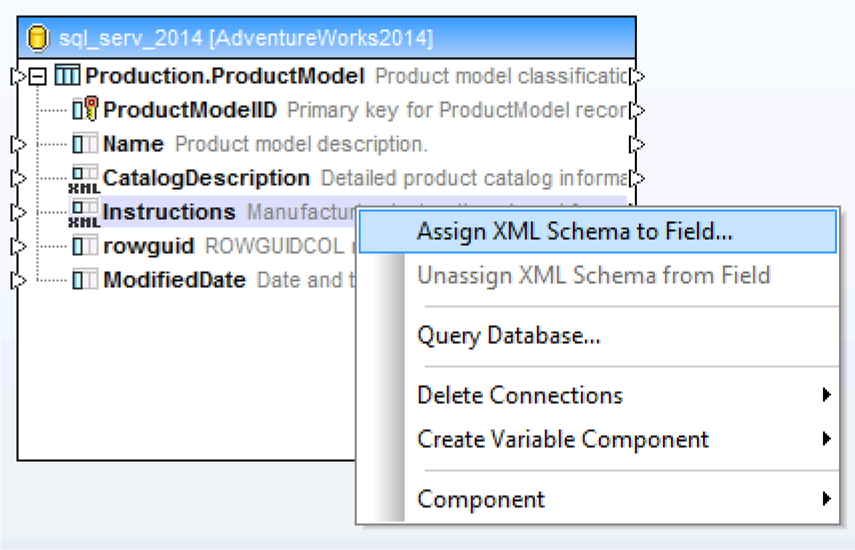


3. Click OK.

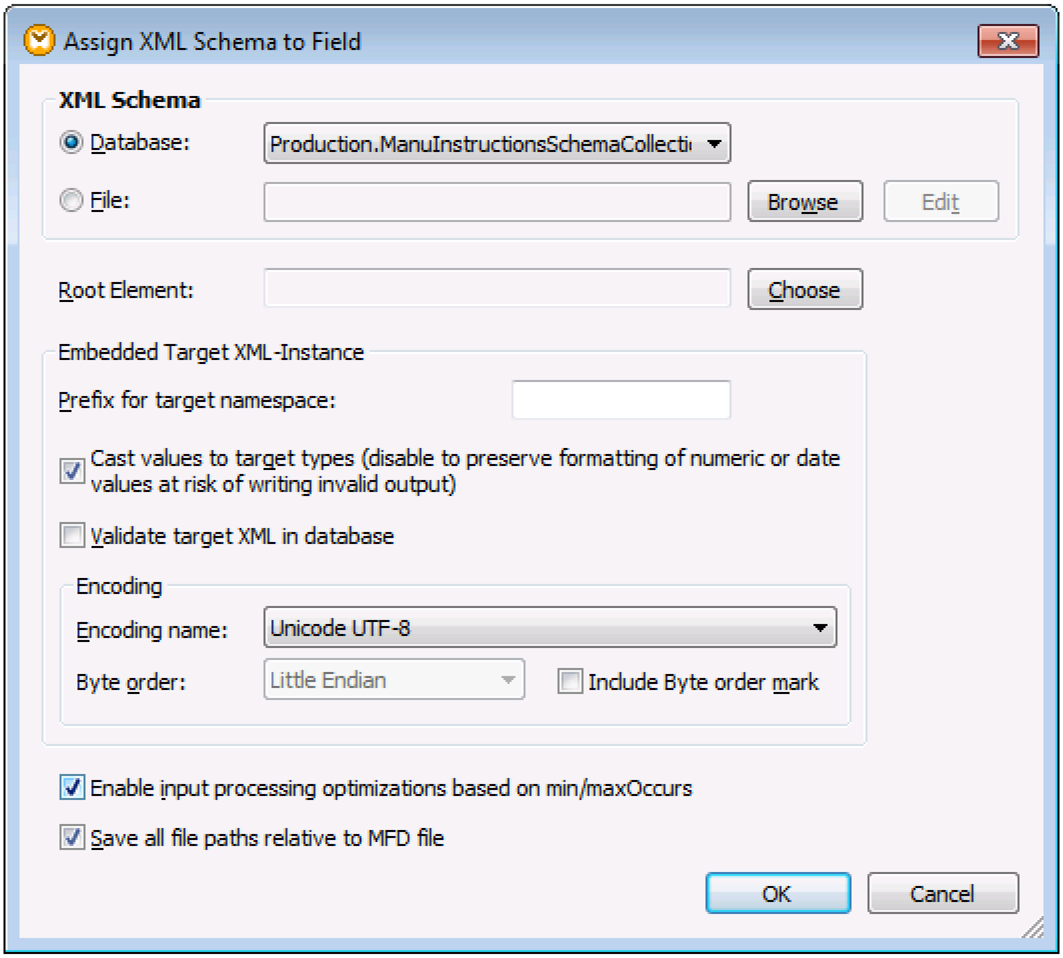
The database table has now been added to the mapping area. Notice that this table has two fields of XML type: **CatalogDescription** and **Instructions**:


sql_serv_2014 [AdventureWorks2014]
Production.ProductModel Product model classificatic
ProductModelID Primary key for ProductModel recor
Name Product model description.
CatalogDescription Detailed product catalog informa
Instructions Manufacturing instructions in xml format
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 classificatic
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

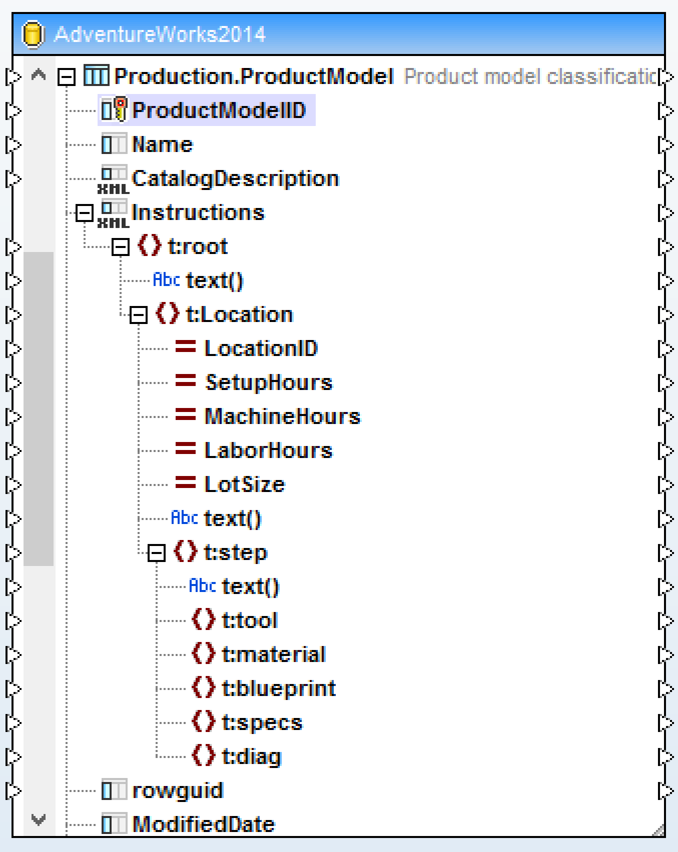
Byte order: Little Endian

□ 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.



4.2.5.2 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\MapForce2025\MapForceExamples\Tutorial\**. The file names are as follows:

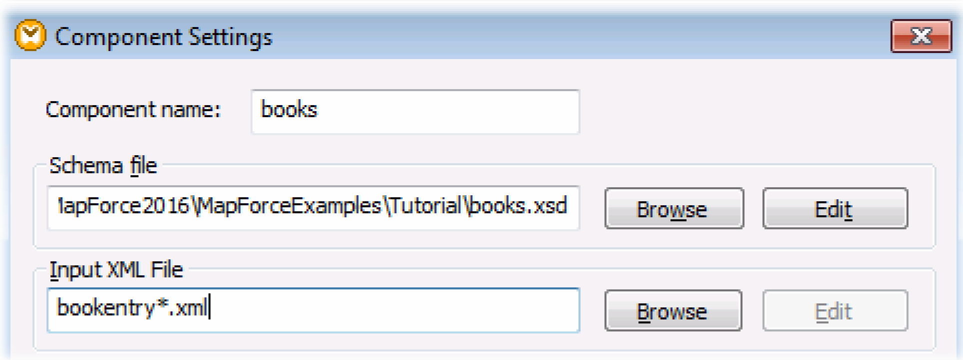
The mapping design file	• **XmlToSqliteField.mfd**
The source XML files	• **bookentry1.xml** • **bookentry2.xml** • **bookentry3.xml**
The XML schema used for validation	• **books.xsd**
The target SQLite database	• **Library.sqlite**

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\MapForce2025\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[746].



Component Settings

Component name: books

Schema file

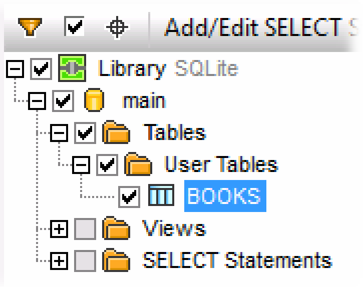
lapForce2016\MapForceExamples\Tutorial\books.xsd

Input XML File

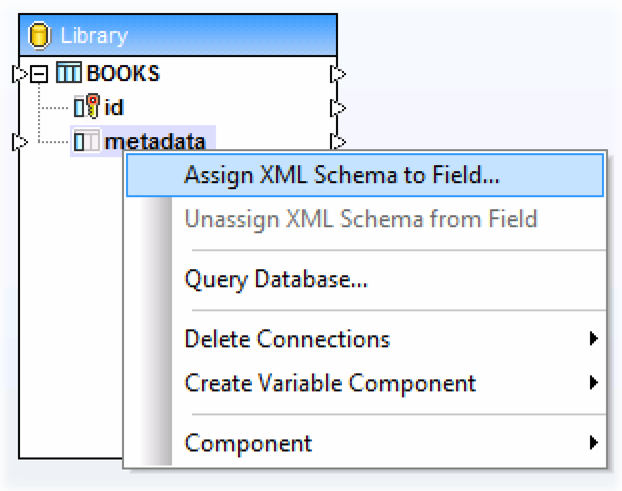
bookentry*.xml

Step 2: Add the SQLite component

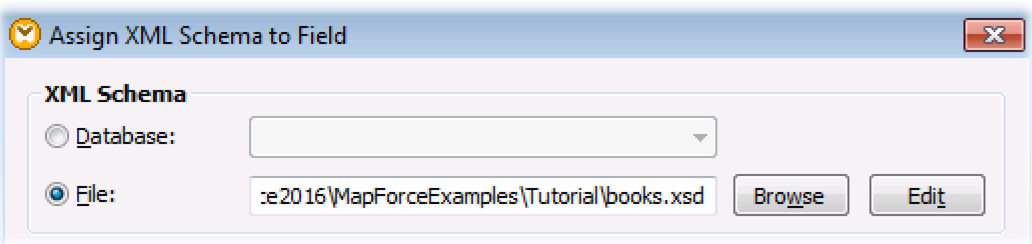
On the **Insert** menu, click **Database**, and follow the wizard to connect to the **Library.sqlite** database file from the **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\** directory (see also Connecting to an Existing SQLite Database[194]). When prompted to select the database objects, 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

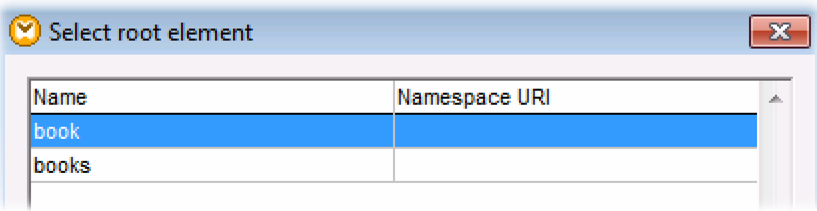
In this tutorial, the schema assigned to the `metadata` field is the same one used to validate the source XML files. Click **Browse** and select the **books.xsd** schema from the **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\** directory:



○ Database:

● File: :e2016\MapForceExamples\Tutorial\books.xsd

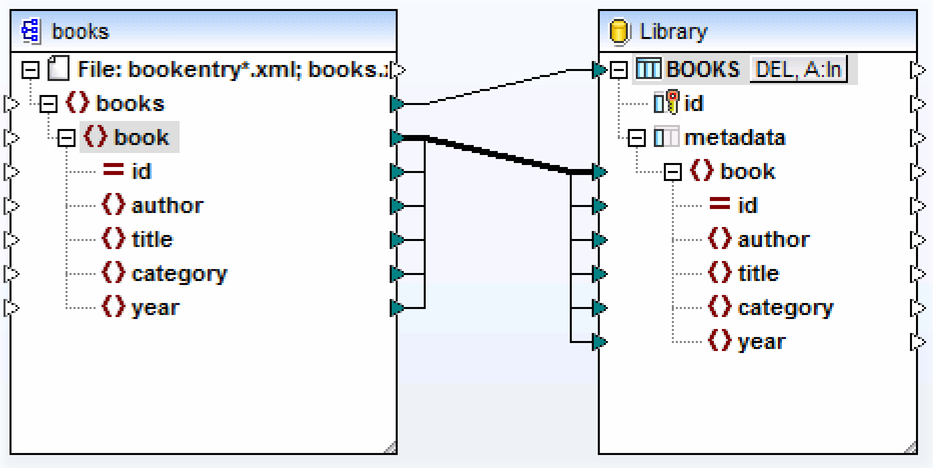
The **books.xsd** schema has two elements with global declaration: `book` and `books`. In this example, we will set `book` as the root element of the XML written to the database field. Click **Choose**, and select `book` as root element:



Name	Namespace URI
book	
books	

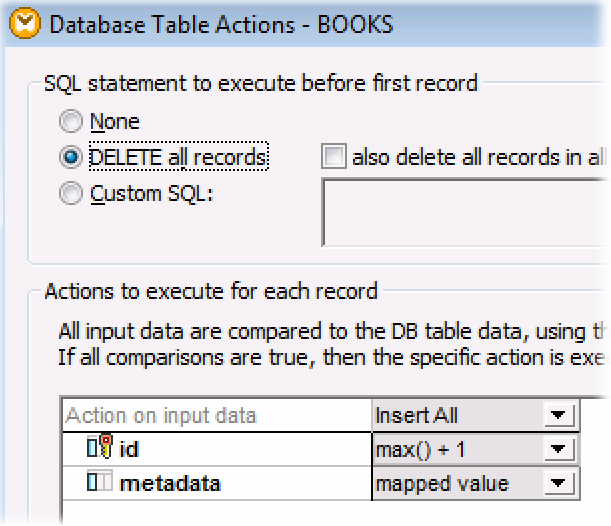
Step 3: Create the mapping connections and configure the database INSERT action

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 56.

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** pane. Note that this action does not update the database immediately. When you are ready to run the generated database script, select the menu command **Output | Run SQL Script** (or click the toolbar button).

4.2.5.3 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 196).
3. Paste the following text into the SQL Editor. This SQL query creates a database table called `ARTICLES` and populates it with data.

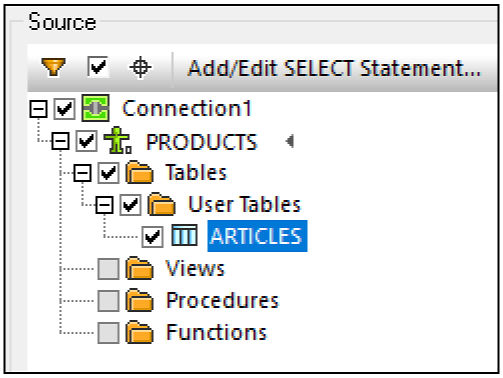
```
-- Create the table
CREATE TABLE
    ARTICLES (
        id INTEGER NOT NULL,
        article XML ) ;
-- Populate the table
INSERT INTO ARTICLES VALUES
   (1, '<Article>
      <Number>1</Number>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>'),
(2, '<Article>
      <Number>2</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.

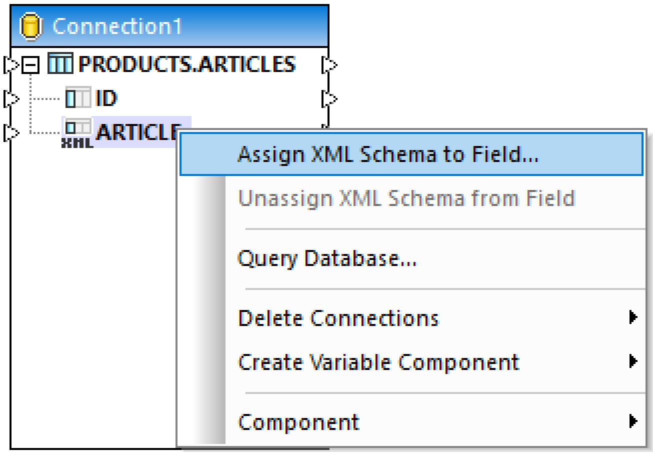
Step 1: Add the database

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

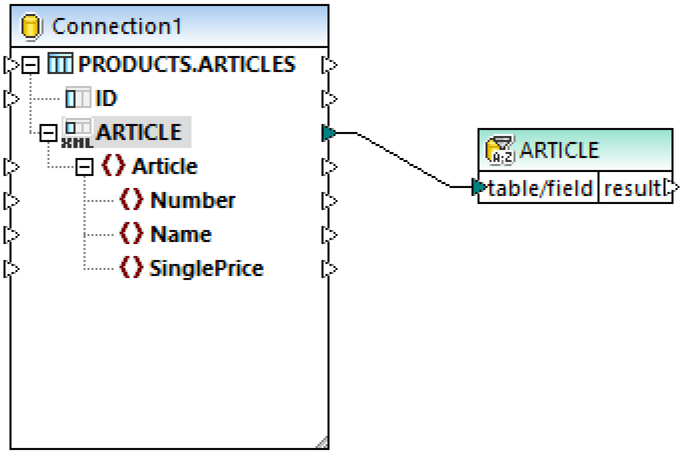
1. Right-click the `ARTICLE` item of the component, and select **Assign XML Schema to field** from the context menu.


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\MapForce2025\MapForceExamples\Tutorial\DB2xsd.xsd**.

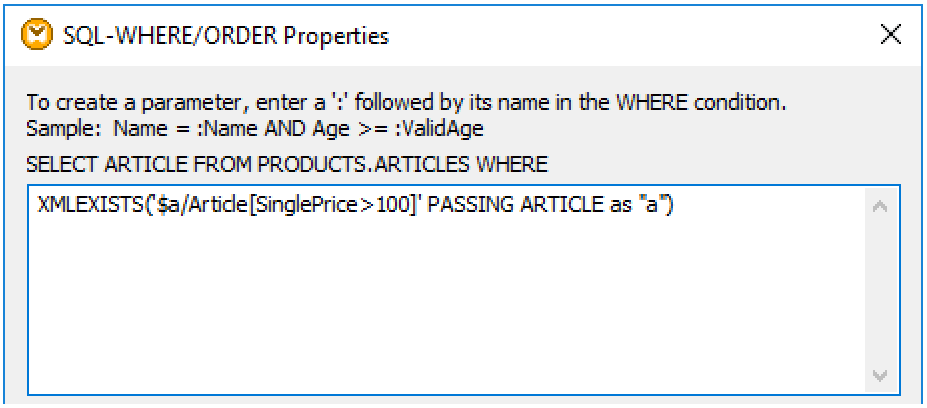
Step 3: Add the SQL WHERE/ORDER component

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

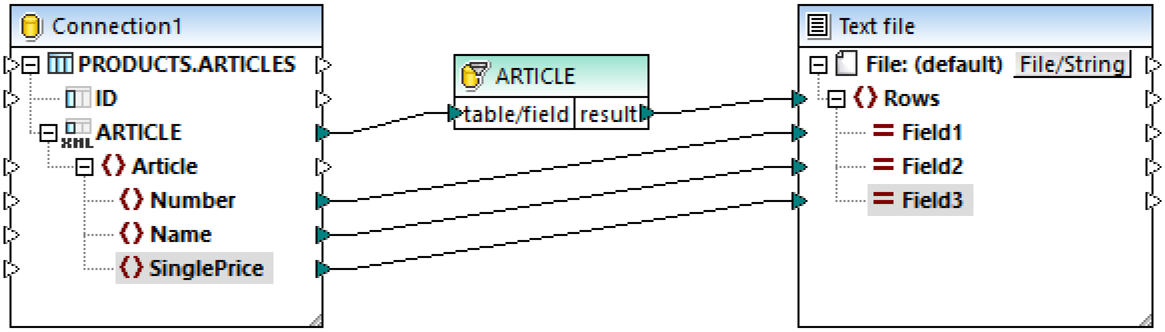
SELECT ARTICLE FROM PRODUCTS.ARTICLES WHERE

XMLEXISTS('$a/Article[SinglePrice>100]' PASSING ARTICLE as "a")

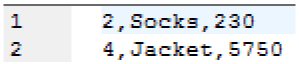
The text above represents the "WHERE" part of the SQL query. At mapping runtime, it will be combined with the "SELECT" part displayed on the dialog box. This statement uses the `XMLEXISTS` function and syntax specific to IBM DB2 databases.

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** pane. As expected, only articles with price greater than 100 are shown in the output.



```
1    2,Socks,230
2    4,Jacket,5750
```

4.2.6 Stored Procedures

Stored procedures are programs that are hosted and run on a database server. Stored procedures can be called by client applications and they are often written in some extended dialect of SQL. Some databases support also implementations in Java, .NET CLR, or other programming languages.

Typical uses of stored procedures include querying a database and returning data to the calling client, or performing modifications to the database after additional validation of input parameters. Stored procedures can also perform other actions outside the database, such as sending e-mails.

A stored procedure may have zero or more input and output parameters, and may optionally return zero or more recordsets, in addition to the default return value. Consequently, in MapForce, you can call a stored procedure in various ways:

- Call a stored procedure in order to retrieve data, as if it were a source component on the mapping. This is applicable for procedures that do not take input parameters. When the mapping runs, the procedure is called, and it returns some recordset or output parameters. You can map the recordset, or the output parameters, or both, to any other data type supported by MapForce. For an example, see Stored Procedures as Data Source 302.
- Call a stored procedure as a function-like call, with parameters. In this case, you supply all required input parameters from the mapping, and you can also map the returned recordset, or the output parameters, or both, to some other target supported by MapForce. For an example, see Stored Procedures with Input and Output 305.
- Call a stored procedure as if it were a target component on the mapping. The typical use case is calling a stored procedure with parameters in order to modify the database (for example, insert a record). This approach is suitable if you do not need any output from the stored procedure. Also, in this approach you can execute the stored procedure within a database transaction that can be rolled back in case of an error. For an example, see Stored Procedures in Target Components 309.

There are also cases where you may need to call stored procedures or perform actions on database tables in a specific order (first insert, then update, and so on). For example, you may need to pass the output parameter of a stored procedure to another stored procedure. Or you may need to combine data returned by a stored procedure with data from a table. Such actions are possible with the help of local relations defined in MapForce, even when the underlying database does not enforce primary/foreign key relationships between tables. For more information, see Stored Procedures and Local Relations 312.

Note: To illustrate how MapForce implements stored procedures, this chapter uses Microsoft SQL Server 2016 and the "AdventureWorks 2016" database. The latter can be downloaded from https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks.

Support notes

- Stored procedures can be used only in the BUILT-IN execution engine. Code generation in C++, C#, or Java is not supported.
- User-defined types, cursor types, variant types and many "exotic" database-specific data types (such as arrays, geometry, CLR types) are generally not supported as input or output parameter types.
- Procedure and function overloading (multiple definitions of routines with the same name and different parameters) is not supported.
- Some databases support default values on input parameters, this is currently not supported. You cannot omit input parameters in the mapping to use the default value.
- Stored procedures returning multiple recordsets are supported depending on the combination of driver and database API (ODBC/ADO/ADO.NET/JDBC). Only procedures that return the same number of recordsets with a fixed column structure are supported.
- Whenever possible, use the latest version of the database native driver maintained by the database vendor. Avoid using bridge drivers, such as ODBC to ADO Bridge, or ODBC to JDBC Bridge.
- You can optionally enable database transactions for stored procedures that are called as data target, see Stored Procedures in Target Components 309. Transactions are not supported for stored procedures that are called as a data source (without input parameters), or those that are called like a function (with both input and output).

The following table lists the database-specific support notes.

Database	Support notes
Access	• Stored procedures in Microsoft Access databases have very limited functionality and are not supported in MapForce.
DB2	• Supported in MapForce: stored procedures, scalar functions, table-valued functions. • Return values from DB2 stored procedures are not supported because they cannot be read via the database APIs used in MapForce. • Row-valued functions (RETURNS ROW) are not supported. • It is recommended to install at minimum "IBM_DB2 9.7 Fix Pack 3a" to avoid a confirmed JDBC driver issue when reading errors/warnings after execution. This also fixes an issue with the ADO provider that causes one missing result set row.
Firebird	• Supported in MapForce: stored procedures, table-valued functions
Informix	• Supported in MapForce: stored procedures, table-valued functions.
MariaDB	• Supported in MapForce: stored procedures, scalar functions

Database	Support notes
MySQL	• Supported in MapForce: stored procedures, scalar functions • MySQL includes complete support for stored procedures and functions starting with version 5.5. If you are using an earlier version, functionality in MapForce is limited.
Oracle	• Supported in MapForce: stored procedures, scalar functions, table-valued functions. This includes standalone stored procedures and functions as well as those defined inside an Oracle package. • It is recommended to use a native Oracle driver instead of the **Microsoft OLE DB Provider for Oracle**. • Oracle has a special way to return result sets to the client by using output parameters of type REF CURSOR. This is supported by MapForce for stored procedures, but not for functions. The names and number of recordsets is therefore always fixed for Oracle stored procedures.
PostgreSQ L	• Supported in MapForce: scalar functions, row-valued functions, table-valued functions. • In PostgreSQL, any output parameters defined in a function describe the columns of the result set. This information is automatically used by MapForce - no detection by execution or manual input of recordsets is needed. Parameters of type `refcursor` are not supported.
Progress OpenEdge	• Supported in MapForce: stored procedures.
SQL Server	• Supported in MapForce: stored procedures, scalar functions, table-valued functions. • For information about available ADO providers, see ADO Connection 176. • The ADO API has limited support for some data types introduced with SQL Server 2008 (`datetime2`, `datetimeoffset`). If you encounter data truncation issues with these temporal types when using ADO with the SQL Server Native Client, you can set the connection string argument `DataTypeCompatibility=80` or use ODBC. • SQL Server Procedures have an implicit return parameter of type `int null`, which is available for mapping. If the procedure omits a RETURN statement, the resulting value is 0.
SQLite	• SQLite does not use stored procedures.
Teradata	• Supported in MapForce: stored procedures, macros. • Scalar functions, aggregate functions and table functions are not supported • Known issue: The Teradata ODBC driver refuses to populate output parameter values after a procedure call.

4.2.6.1 Adding Stored Procedures to the Mapping

On the mapping area, stored procedures are shown as part of the database component where they belong. In order for stored procedures to be visible on the database component, you must explicitly select them when adding the database component to the mapping, as shown below. In this example, we connect to the "AdventureWorks" database running on SQL Server. Instructions are similar for other database types.

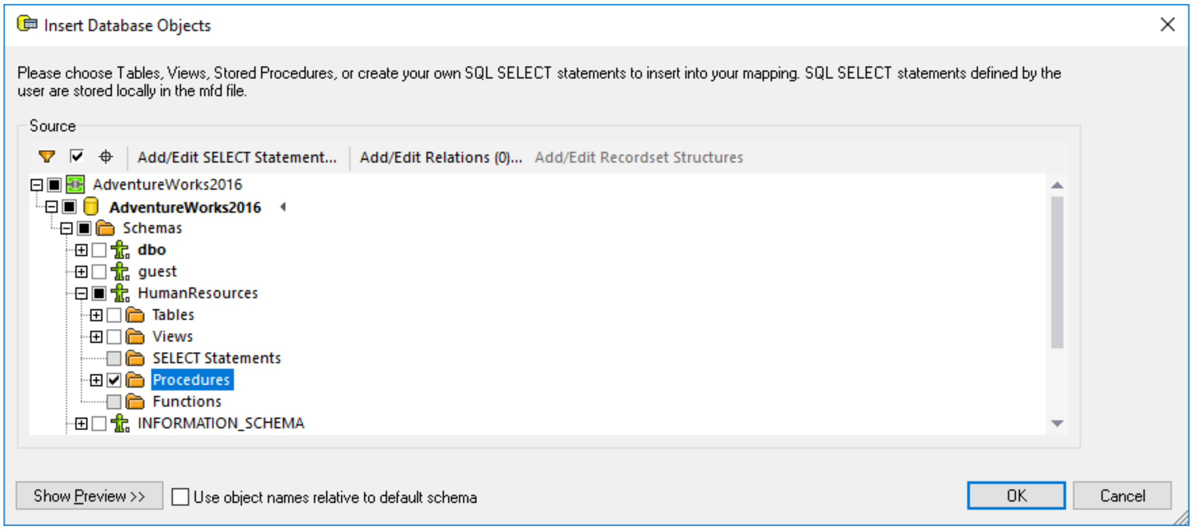
In case of Oracle databases, stored procedures or functions may be standalone or part of Oracle packages. You can add both categories to the mapping. The stored procedures or functions belonging to a package appear under the respective package name on the "Insert Database Objects" dialog box illustrated below.

To add stored procedures to the mapping:

1. Do one of the following:

   - On the **Insert** menu, click **Database**.
   - Click the **Insert Database** ( ) toolbar button.

2. Follow the database wizard steps until you get to the "Insert Database Objects" dialog box. For detailed instructions applicable to each database type, see Database Connection Examples[196].



Insert Database Objects

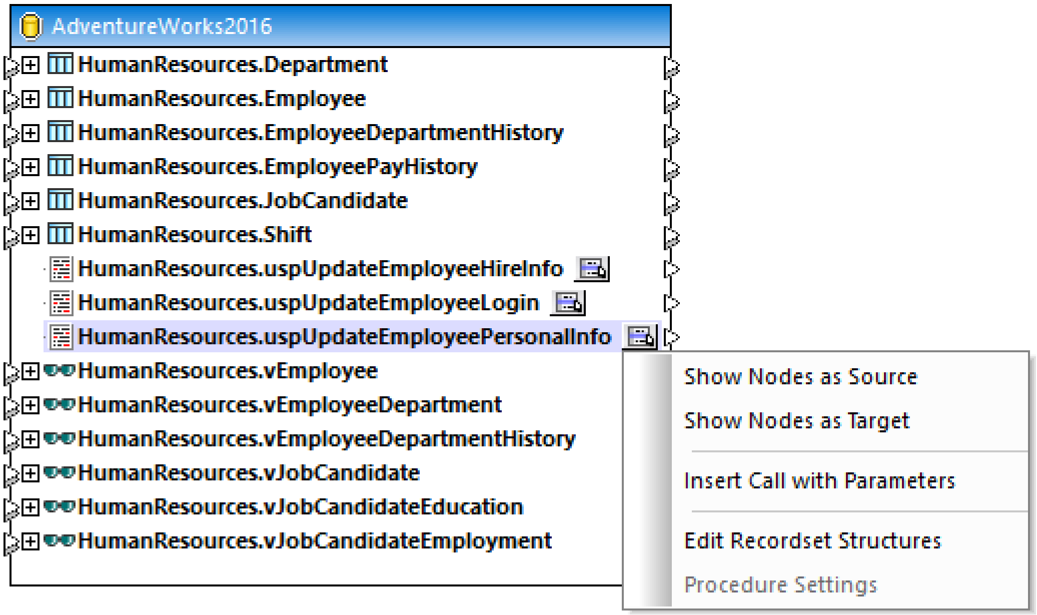
Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.

3. Select the check boxes next to the database objects that you need to be displayed on the mapping, and click **OK**. In this example, we have selected all the tables, views, and stored procedures available in the "HumanResources" schema.

Notes

- You can change the selected objects at any time later, by right-clicking the title bar of a database component, and selecting **Add/Remove/Edit Database Objects** from the context menu.
- Your database user account must have rights to view and execute stored procedures in the database.

The database component is now added to the mapping. Notice that stored procedures are identified by the icon. In addition, tables, views and procedures are sorted alphabetically in the database component.


AdventureWorks2016
HumanResources.Department
HumanResources.Employee
HumanResources.EmployeeDepartmentHistory
HumanResources.EmployeePayHistory
HumanResources.JobCandidate
HumanResources.Shift
HumanResources.uspUpdateEmployeeHireInfo
HumanResources.uspUpdateEmployeeLogin
HumanResources.uspUpdateEmployeePersonalInfo
HumanResources.vEmployee
HumanResources.vEmployeeDepartment
HumanResources.vEmployeeDepartmentHistory
HumanResources.vJobCandidate
HumanResources.vJobCandidateEducation
HumanResources.vJobCandidateEmployment
Show Nodes as Source
Show Nodes as Target
Insert Call with Parameters
Edit Recordset Structures
Procedure Settings

The **Show Context Menu** button next to each stored procedure lets you configure how the stored procedure is to be called, and other procedure-related settings, as follows:

Option	Usage
Show Nodes as Source	Select this option if you want to call a stored procedure *without parameters* in order to retrieve data from a database and map it to another component supported by MapForce (XML, text, EDI, and so on). For an example, see Stored Procedures as Data Source 302.
Show Nodes as Target	Select this option if you want to call a stored procedure in order to modify the database or perform another specific action where you don't need the output of the stored procedure. For an example, see Stored Procedures in Target Components 309.
Insert Call with Parameters	Select this option if you want to call a stored procedure *with parameters* and want to map the returned data to another component supported by MapForce. For an example, see Stored Procedures with Input and Output Parameters 305.
Edit Recordset Structures	Applicable for stored procedures that return recordsets. Select this option to execute the stored procedure once, so that MapForce can determine the structure of the returned recordset and display it on the mapping. Alternatively, if you don't want to execute the stored procedure at design time, you can define the recordset structure manually.
Procedure Settings	Applicable only for stored procedures that were configured as "target" (that is, those that update the database). Select this option to configure additional procedure-related settings, such as running a custom

	SQL query before calling the procedure, or enabling database transactions.

4.2.6.2 Stored Procedures as Data Source

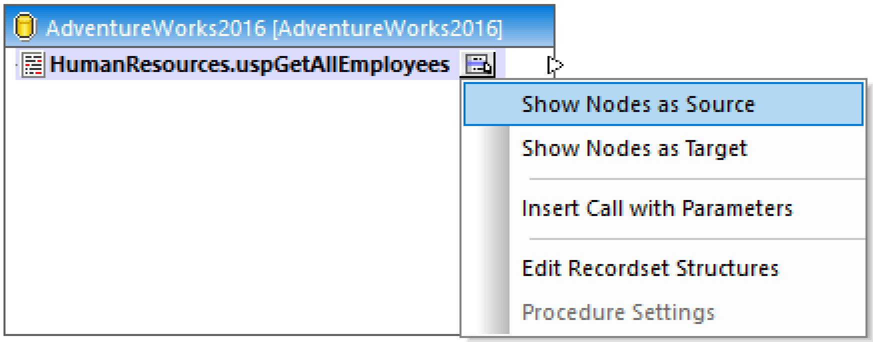
This example shows you how to call a procedure that takes no input parameters and just retrieves some data from the database. In this scenario, the stored procedure acts as a source component to the mapping, and you can map data retrieved by it to any other target component supported by MapForce. If you need to call a stored procedure with input parameters, see Stored Procedures with Input and Output 305.

Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** pane of MapForce, see Browsing and Querying Databases 277. In either case, make sure that your database user account has permission to create stored procedures.

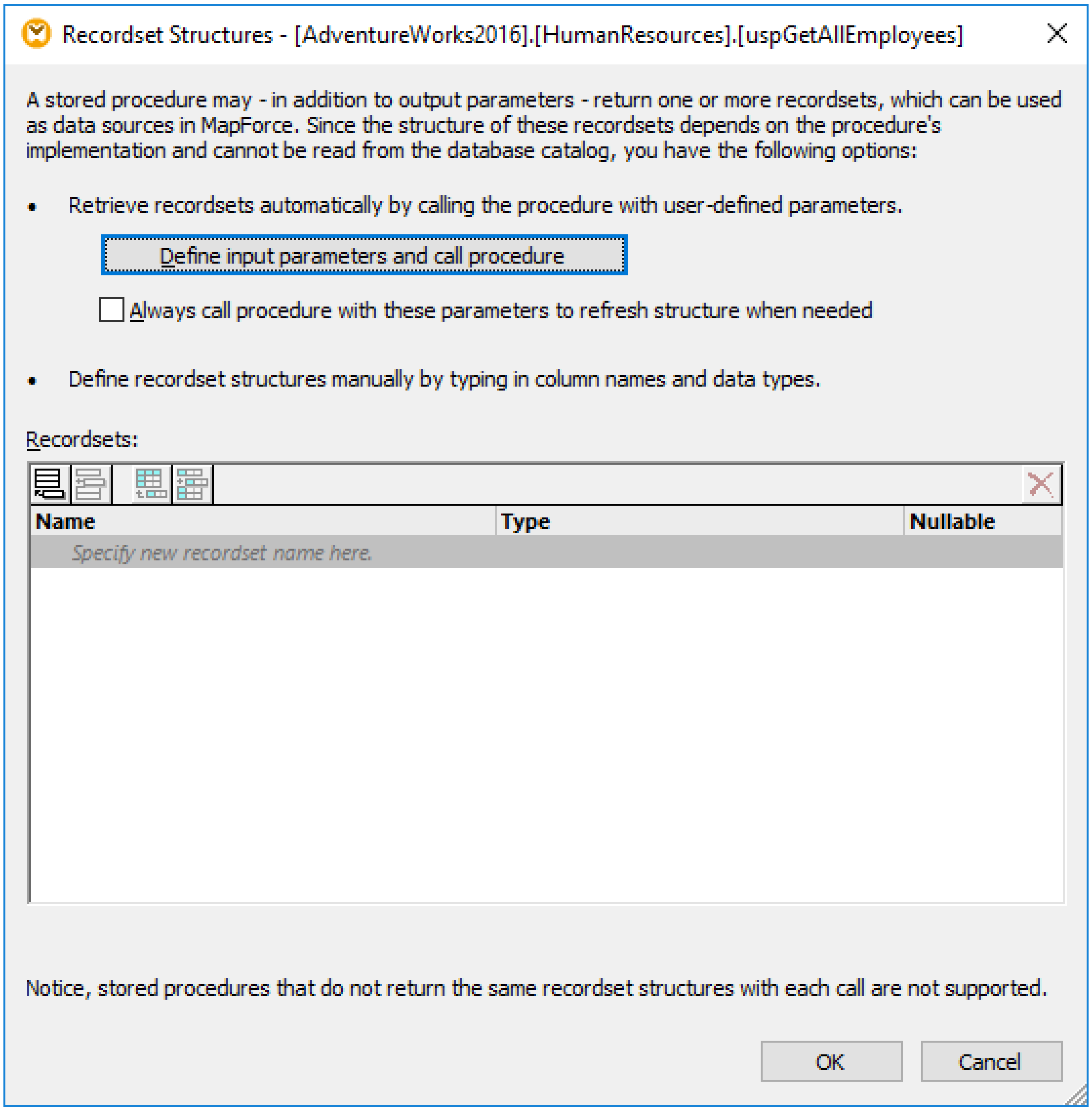
```
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment
```

The stored procedure above returns employee information from the **vEmployeeDepartment** view. The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping 300. Make sure that your database user account has permission to view and execute stored procedures.
2. Click the **Show Context Menu** button next to the stored procedure and select **Show Nodes as Source**.


AdventureWorks2016 [AdventureWorks2016]
HumanResources.uspGetAllEmployees
Show Nodes as Source
Show Nodes as Target
Insert Call with Parameters
Edit Recordset Structures
Procedure Settings

3. Click the **Show Context Menu** button again and select **Edit Recordset Structures**. The "Recordset Structures" dialog box appears.



Recordset Structures - [AdventureWorks2016].[HumanResources].[uspGetAllEmployees]

A stored procedure may - in addition to output parameters - return one or more recordsets, which can be used as data sources in MapForce. Since the structure of these recordsets depends on the procedure's 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.

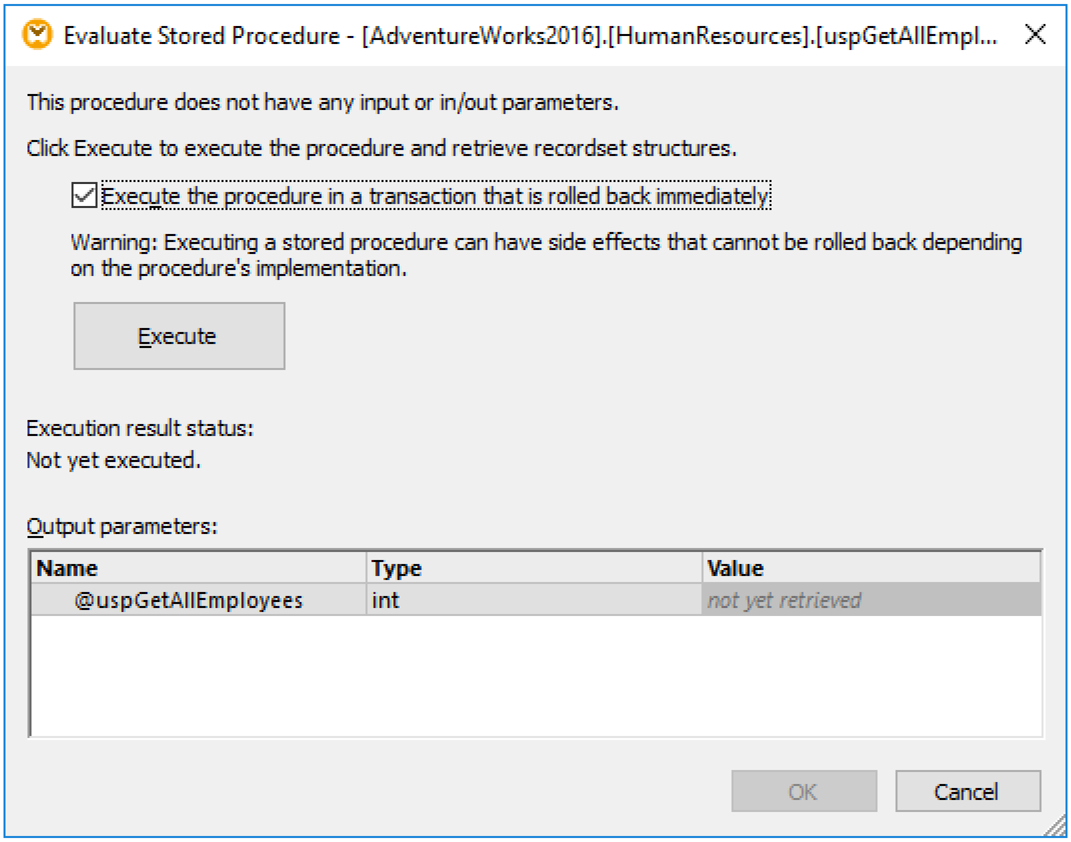
Recordsets:

Name	Type	Nullable
Specify new recordset name here.		

Notice, stored procedures that do not return the same recordset structures with each call are not supported.

Calling a stored procedure at design time may have side effects (depending on the procedure implementation). If you do not want to execute the stored procedure at design time, do not click **Execute**, as further described in subsequent steps. Instead, define the expected recordset in the "Recordset Structures" dialog box, by adding recordsets and their associated columns manually. Use the **Add recordset** or **Add column** buttons in the "Recordset Structures" dialog box.

4. Click **Define input parameters and call procedure**. The "Evaluate Stored Procedure" dialog box appears.


Evaluate Stored Procedure - [AdventureWorks2016].[HumanResources].[uspGetAllEmpl...

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

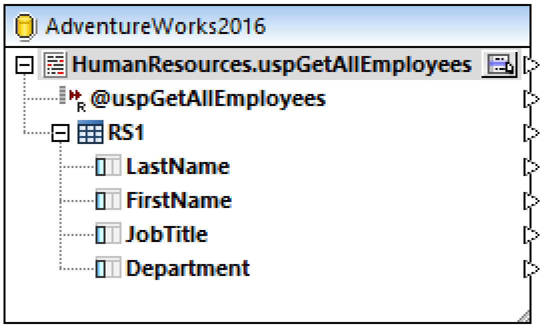
Warning: Executing a stored procedure can have side effects that cannot be rolled back depending on the procedure's implementation.

Execution result status:
Not yet executed.

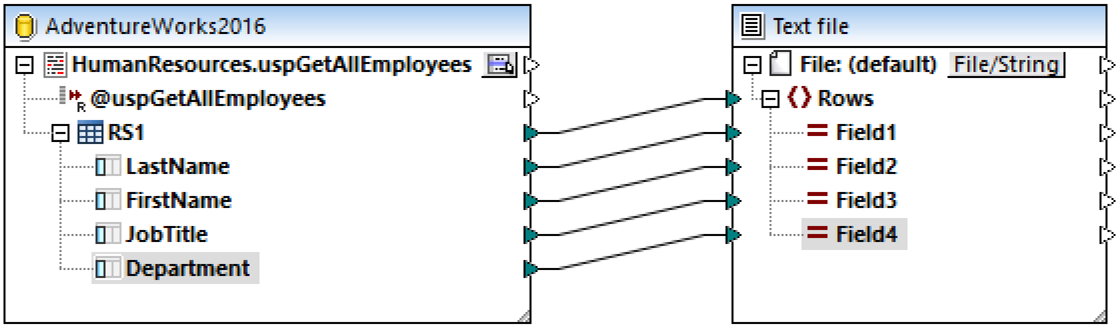
Output parameters:

Name	Type	Value
@uspGetAllEmployees	int	*not yet retrieved*

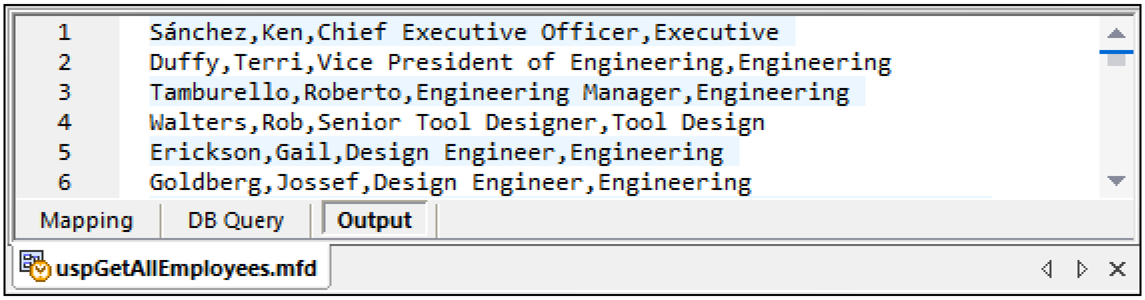
5. Click **Execute**, and then click **OK**. The recordset structure ("RS1") is now visible both on the "Recordset Structures" dialog box and on the mapping.



6. At this stage, you can add a target component where the retrieved data will be written. In this example, data will be written to a CSV file. On the **Insert** menu, click **Text File**, and add a CSV component to the mapping. For more information, see CSV and Text Files 323.



You can now preview the mapping. Click the **Output** button and observe the mapping result in the **Output** pane, for example:



```
1    Sánchez,Ken,Chief Executive Officer,Executive
2    Duffy,Terri,Vice President of Engineering,Engineering
3    Tamburello,Roberto,Engineering Manager,Engineering
4    Walters,Rob,Senior Tool Designer,Tool Design
5    Erickson,Gail,Design Engineer,Engineering
6    Goldberg,Jossef,Design Engineer,Engineering
```

4.2.6.3 Stored Procedures with Input and Output

This example shows you how to call a procedure that takes input parameters and also retrieves some output from the database. In this scenario, the stored procedure is called similar to a web service, or a function, and you can map data retrieved by it to any other target component supported by MapForce.

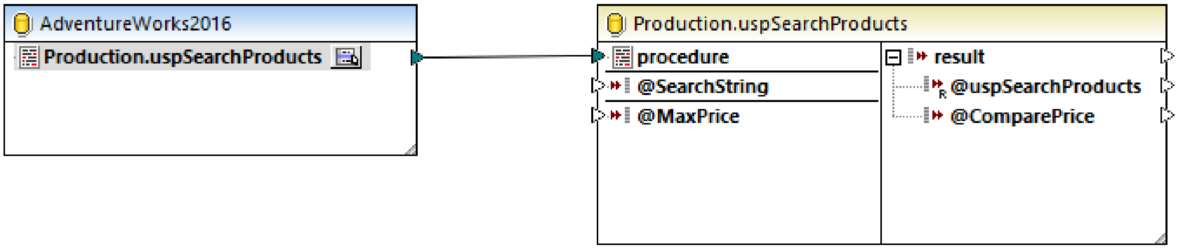
Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** pane of MapForce, see Browsing and Querying Databases 277. In either case, make sure that your database user account has permission to create stored procedures.

```
CREATE PROCEDURE Production.uspSearchProducts
   @SearchString nvarchar(50)
   ,@MaxPrice money
   ,@ComparePrice money OUTPUT
AS
BEGIN
   SET NOCOUNT ON
   SELECT pr.[Name], pr.ListPrice FROM [Production].[Product] pr
   WHERE pr.[Name] like @SearchString AND  pr.ListPrice < @MaxPrice

   SET @ComparePrice = @MaxPrice
   RETURN @ComparePrice
END
```

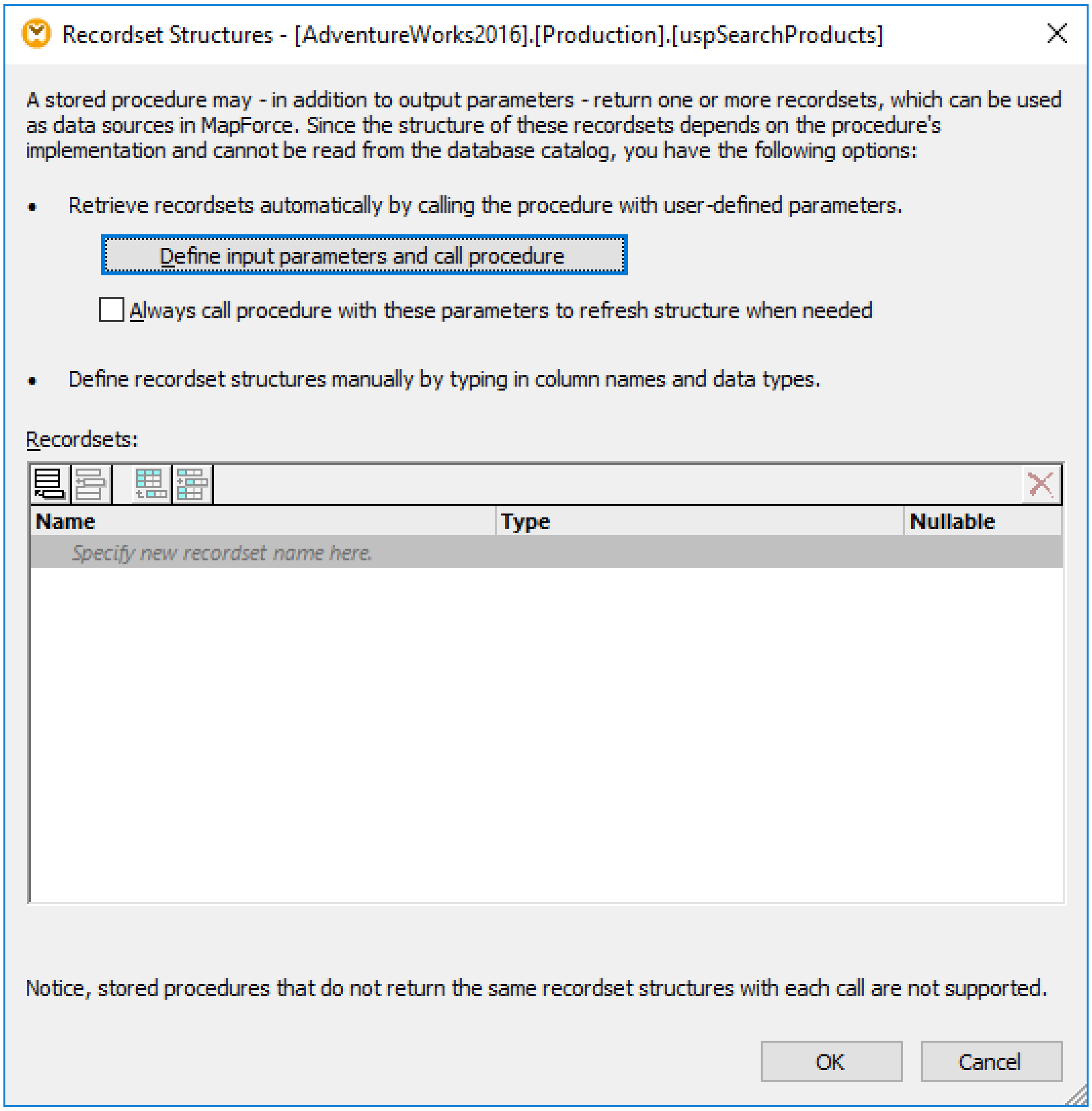
The stored procedure above retrieves a recordset containing product information. It takes as input two parameters: a string with the product name (`@SearchString`) and the maximum product price (`@MaxPrice`). In addition to the recordset and the default return parameter, it also retrieves an output parameter (`@ComparePrice`).

The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping 300. Make sure that your database user account has permission to view and execute stored procedures.
2. Click the **Show Context Menu** button next to the stored procedure and select **Insert Call with Parameters**. The stored procedure now appears in a separate component on the mapping, where the left side lists the input parameters, and the right side contains the return and the output parameters.



3. Click the **Show Context Menu** button again, and select **Edit Recordset Structures**. This is necessary so as to provide to MapForce information about the structure of the recordset returned by the procedure. The "Recordset Structures" dialog box appears.


Recordset Structures - [AdventureWorks2016].[Production].[uspSearchProducts]

A stored procedure may - in addition to output parameters - return one or more recordsets, which can be used as data sources in MapForce. Since the structure of these recordsets depends on the procedure's 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.

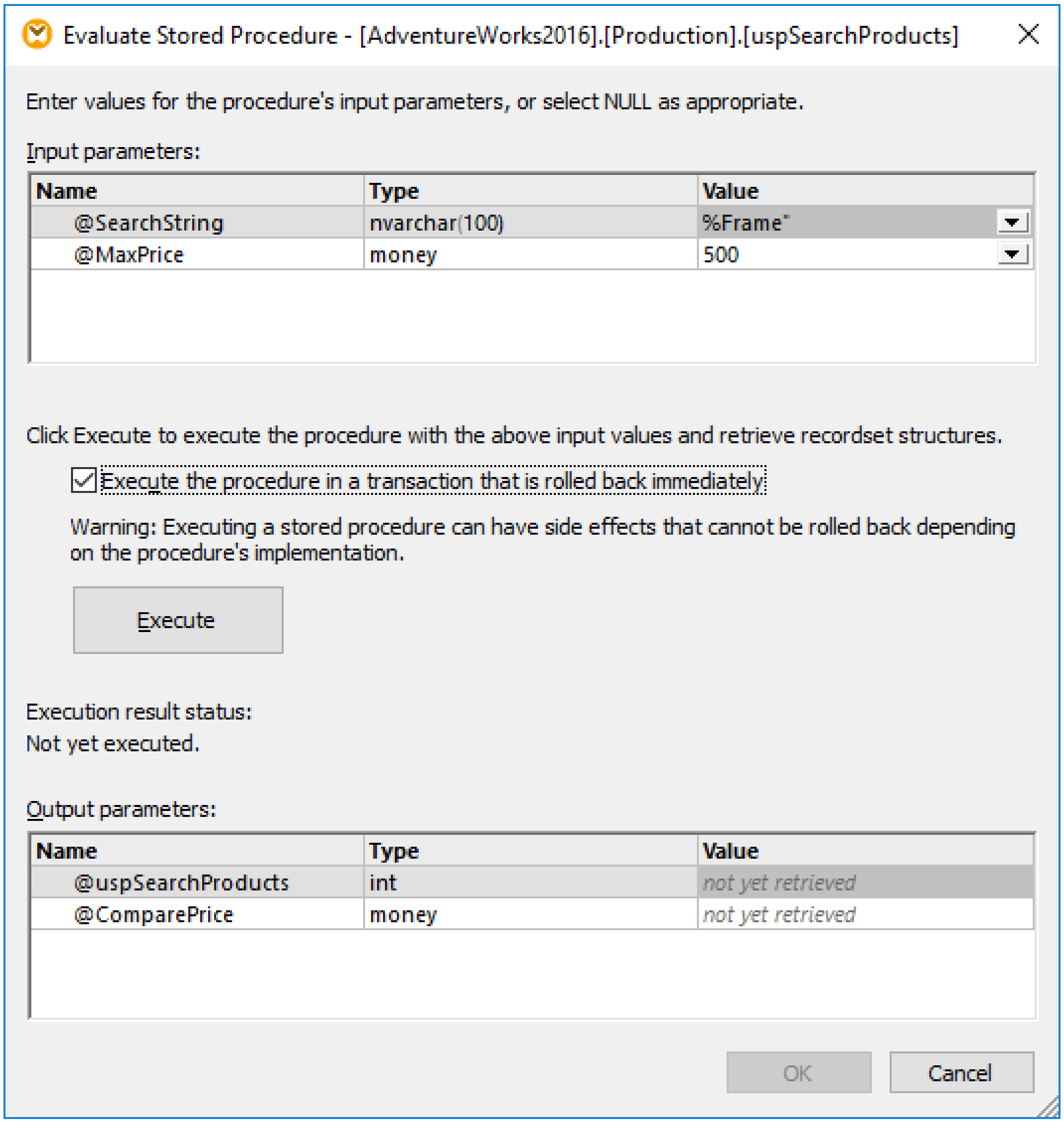
Recordsets:

Name	Type	Nullable
Specify new recordset name here.		

Notice, stored procedures that do not return the same recordset structures with each call are not supported.

> Calling a stored procedure at design time may have side effects (depending on the procedure implementation). If you do not want to execute the stored procedure at design time, do not click **Execute**, as further described in subsequent steps. Instead, define the expected recordset in the "Recordset Structures" dialog box, by adding recordsets and their associated columns manually. Use the **Add recordset** or **Add column** buttons in the "Recordset Structures" dialog box.

4. Click **Define input parameters and call procedure**. The "Evaluate Stored Procedure" dialog box appears.



Evaluate Stored Procedure - [AdventureWorks2016].[Production].[uspSearchProducts]

Enter values for the procedure's input parameters, or select NULL as appropriate.

Input parameters:

Name	Type	Value
@SearchString	nvarchar(100)	%Frame"
@MaxPrice	money	500

Click Execute to execute the procedure with the above input values 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.

Execute

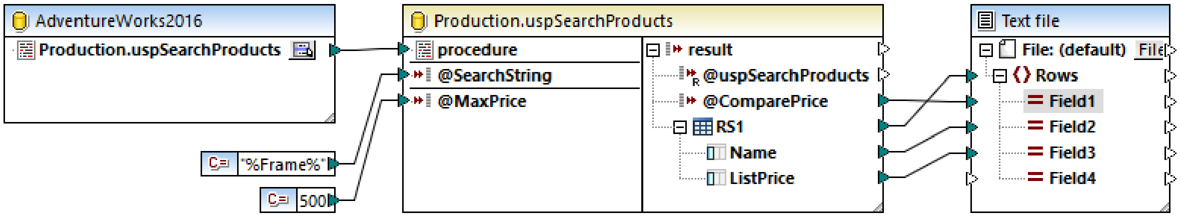
Execution result status:
Not yet executed.

Output parameters:

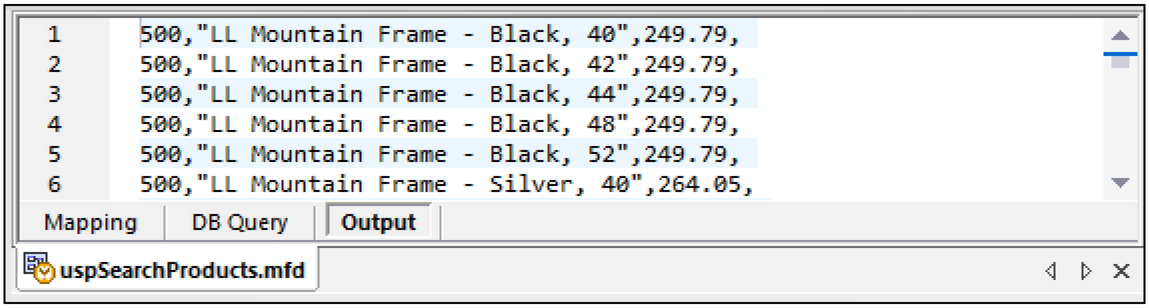
Name	Type	Value
@uspSearchProducts	int	not yet retrieved
@ComparePrice	money	not yet retrieved

OK Cancel

5. Fill in the parameter values as shown above, and click **Execute**.
6. Click **OK**. The recordset structure ("RS1") is now visible both on the "Recordset Structures" dialog box and on the mapping.
7. At this stage, you can add a target component where the retrieved data will be written. In this example, data will be written to a CSV file. On the **Insert** menu, click **Text File**, and add a CSV component to the mapping, see also CSV and Text Files 323. Next, draw the mapping connections as illustrated below. Notice that the procedure's input parameters are supplied by means of constants. For more information about constants, see Add a Constant to the Mapping 437.



You can now preview the mapping. Click the **Output** button and observe the mapping result in the **Output** pane, for example:



```
1    500,"LL Mountain Frame - Black, 40",249.79,
2    500,"LL Mountain Frame - Black, 42",249.79,
3    500,"LL Mountain Frame - Black, 44",249.79,
4    500,"LL Mountain Frame - Black, 48",249.79,
5    500,"LL Mountain Frame - Black, 52",249.79,
6    500,"LL Mountain Frame - Silver, 40",264.05,
```

4.2.6.4 Stored Procedures in Target Components

This example shows you how to call a procedure that takes input parameters and updates a database. Calling a procedure this way makes it possible to enable transactions and roll the action back in case of an error, or add a custom SQL statement to be executed before the procedure is called. This scenario implies that the stored procedure acts like a target component in MapForce and you are not interested in the output returned by it. For an example that illustrates how to pass parameters and also map data returned by a stored procedure, see Stored Procedures with Input and Output 305.

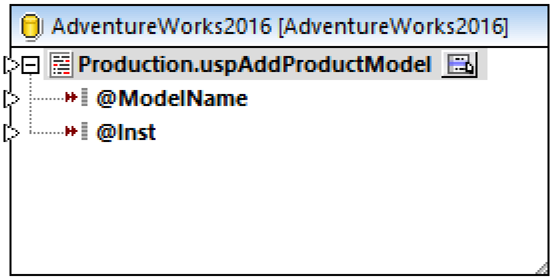
Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** pane of MapForce, see Browsing and Querying Databases 277. In either case, make sure that your database user account has permission to create stored procedures.

```
CREATE PROCEDURE Production.uspAddProductModel
   @ModelName nvarchar(50)
   ,@Inst xml
AS
BEGIN
INSERT INTO [Production].[ProductModel]
           ([Name]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           ,@Inst
           ,NEWID()
           ,GETDATE())
END
```

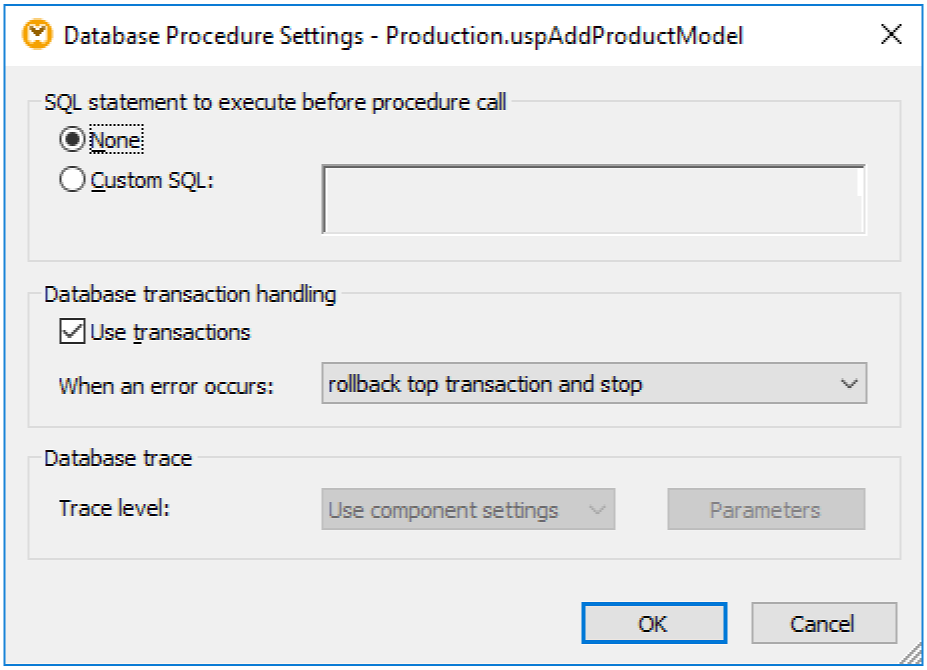
The stored procedure above takes two parameters (`@ModelName`, `@Inst`) as input and inserts the corresponding values into the `ProductModel` table of the AdventureWorks database, along with some database-generated data.

The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce and add the stored procedure to the mapping, as described in Adding Stored Procedures to the Mapping 300. Make sure that your database user account has permission to view and execute stored procedures.
2. Click the **Show Context Menu** button next to the stored procedure and select **Show Nodes As Target**. The stored procedure now appears as target component on the mapping, where the left side lists the input parameters.


AdventureWorks2016 [AdventureWorks2016]
Production.uspAddProductModel
@ModelName
@Inst

3. Click the **Show Context Menu** button again, and select **Procedure Settings**. This optional step enables you to execute the stored procedure inside a transaction that can be rolled back. You can also add a custom SQL statement to be executed before the procedure is called.
4. Select the **Use Transactions** check box.


Database Procedure Settings - Production.uspAddProductModel

SQL statement to execute before procedure call

● None
○ Custom SQL:

Database transaction handling

☑ Use transactions

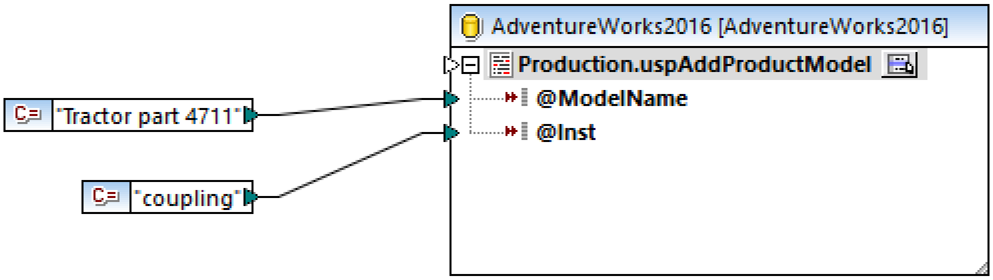
When an error occurs: rollback top transaction and stop

Database trace

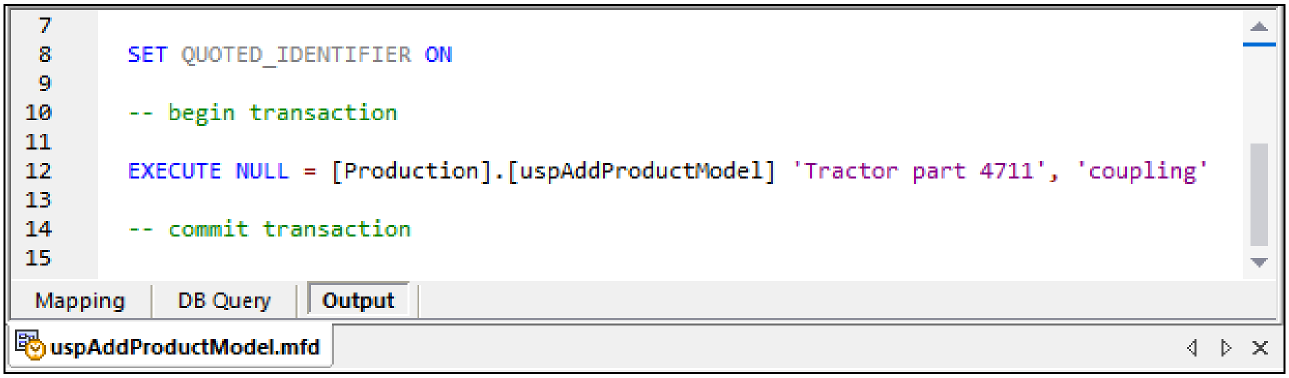
Trace level: Use component settings

Note: In this example, database tracing is disabled at database component level and no tracing is set to take place. However, you can enable database tracing for stored procedures if necessary.

5. Add the source component that provides data to be inserted into the database. In this example, the source data is supplied by constants; however, any other source component supported by MapForce could act as input. For more information about constants, see Add a Constant to the Mapping 437.


AdventureWorks2016 [AdventureWorks2016]
Production.uspAddProductModel
@ModelName
@Inst
"Tractor part 4711"
"coupling"

Since this mapping updates a database, you do not preview its output directly like with other mappings. Instead, click the **Output** button to display the pseudo-SQL containing hints about how the database will be modified. If you enabled transactions, these will occur as indicated by the comments.



```
SET QUOTED_IDENTIFIER ON

-- begin transaction

EXECUTE NULL = [Production].[uspAddProductModel] 'Tractor part 4711', 'coupling'

-- commit transaction

```

> The pseudo-SQL displayed in the **Output** pane does not show the actual transaction commands, only hints (as comments). The actual SQL commands are sent to the underlying database API, however.

To run the mapping against the database, do one of the following:

- On the **Output** menu, click **Run SQL-Script**.
- Click the **Run SQL-Script** toolbar button.

Stored procedures and duplicate inputs

If you need to map data from multiple sources on the mapping to the same stored procedure, you can duplicate the stored procedure so that it accepts multiple inputs. To do this, right-click the stored procedure item on the component and select **Add duplicate input** from the context menu, see also Duplicating Input [42]. When the mapping runs, such duplicate stored procedures will be called once for each duplicated input.

Note that the **Add duplicate input** command is disabled for the stored procedure parameters, because each parameter is an atomic value (and could also be "nullable").

Tracing at stored-procedure level

To enable tracing at stored-procedure level:

1. Make sure that the tracing level at database component level is set to either **Always** or **Error** (see above).
2. Do one of the following, click the **Show Context Menu** button, and then select **Procedure Settings** from the context menu.
3. Select the trace level. The **Use component settings** option inherits the same settings that were defined at the component level. The **Limit to errors** option restricts tracing only to error events. **Always disabled** means that no tracing will occur for this table or stored procedure.

4.2.6.5 Stored Procedures and Local Relations

Local relations are logical relationships between database fields that you can create in MapForce, saving you the need to change the underlying database, see also Defining Local Relationships[253]. You can define local relations not only for database fields, but also for stored procedures as well, both in source and target components.

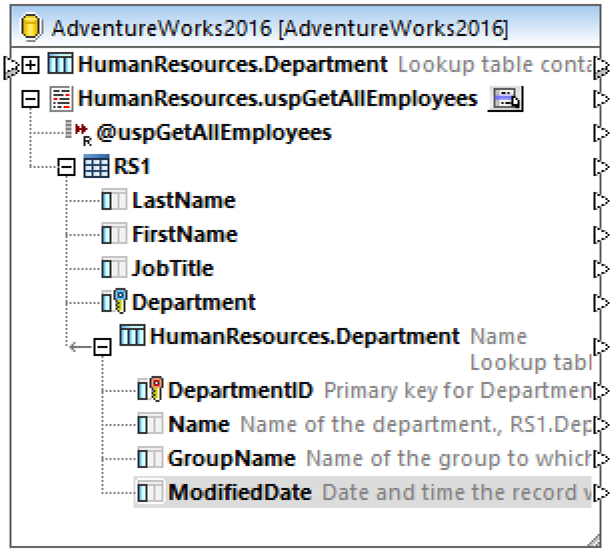
In source components, local relations make it easy to read data from related objects, for example, read IDs from a database table and call a stored procedure with each of these IDs to retrieve related information. It is also possible to call a stored procedure with data retrieved from another procedure.

In target components, local relations enable you to define a hierarchical order in which multiple related procedures are to be called. For example, you can first call a stored procedure that creates an ID value, and another one that inserts related information into a table. It is also possible to mix stored procedures and tables in local relations. For example, you can perform the insert directly on a related table instead of calling another procedure, see Using Stored Procedures to Generate Keys[317].

To create a local relation:

1. Right-click the title bar of a database component and select **Add/Remove/Edit Database Objects** from the context menu. The "Add/Remove/Edit Database Objects" dialog box opens.
2. Click **Add/Edit Relations**.
3. Click **Add Relation** and select the objects between which you want to create the relationship.

As illustrated above, a local relation consists of a **primary/unique key** object and a **foreign key** object. Think of it as a parent-child relationship. On the mapping component, the object (table, view, procedure, and so on) where the primary/unique key is will appear as a parent while the object where the foreign key is will appear nested under it. For example, in the database component illustrated below, a local relation was defined between a recordset column (**RS1.Department**) and a table column (**Department.Name**). Consequently, the **Department** table appears as a child of the stored procedure on the mapping. This example is discussed in more detail in Local Relations in Source Components[313].



AdventureWorks2016 [AdventureWorks2016]
HumanResources.Department Lookup table conta
HumanResources.uspGetAllEmployees
@uspGetAllEmployees
RS1
LastName
FirstName
JobTitle
Department
HumanResources.Department Name Lookup tabl
DepartmentID Primary key for Departmen
Name Name of the department., RS1.Dep
GroupName Name of the group to which
ModifiedDate Date and time the record v

The following table lists all the possible fields between which you can define local relations. Mixed relationships are possible (e.g., mapping the output of a stored procedure to a database column). The fields taking part in the relationship must have the same or compatible data types.

Primary/unique key	Foreign key
• Column of a database table or view • Output parameter or return value of a stored procedure (see also Stored Procedures 297) • Column of a recordset returned by a stored procedure. Applicable if the stored procedure is called as a data source (without parameters) or as a function (with input and output parameters). In order for the recordset to become available for selection, you must execute the stored procedure once, to retrieve the recordset. • Column of a user-defined SELECT statement (see also SQL SELECT Statements as Virtual Tables 243)	• Column of a database table or view • Input parameter of a stored procedure • Input parameter of a user-defined SELECT statement

4.2.6.6 Local Relations in Source Components

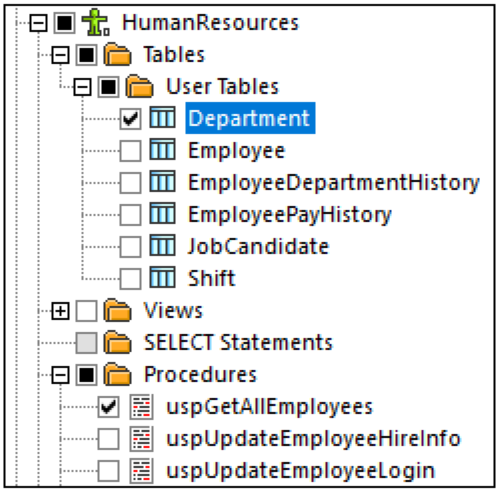
This example shows you how to combine data returned by a stored procedure with data from a table in the same database, with the help of local relations.

If you haven't done so already in a previous example, run the following script to create the demo stored procedure in the "AdventureWorks" database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** pane of MapForce, see Browsing and Querying Databases 277. In either case, make sure that your database user account has permission to create stored procedures.

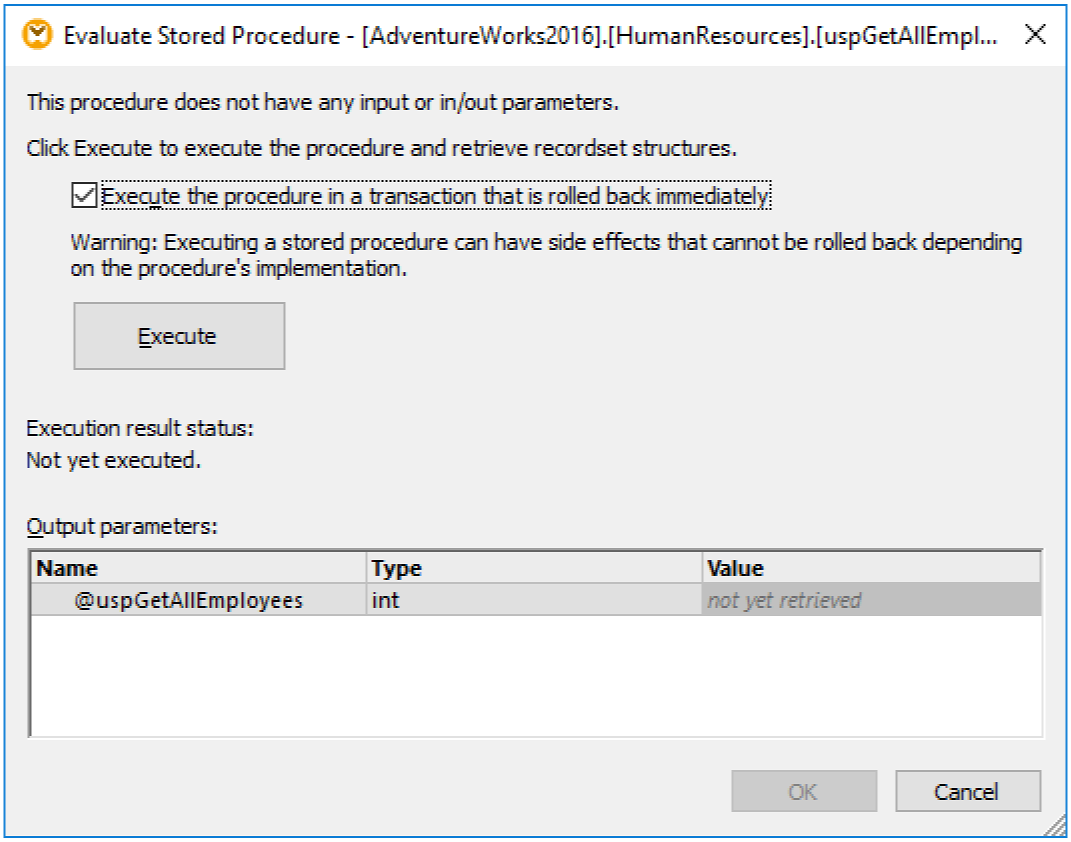
```
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment
```

The stored procedure above returns employee information from the **vEmployeeDepartment** view. The following steps show you how to create a mapping that consumes data returned by this procedure.

1. Connect to the "AdventureWorks" database from MapForce, as described in Adding Stored Procedures to the Mapping 300. Make sure that your database user account has permission to view and execute stored procedures.
2. When prompted to choose database objects, select the **Department** table and the **uspGetAllEmployees** stored procedure.



3. Click the **Show Context Menu** button next to the stored procedure and select **Show Nodes as Source**.
4. Click the **Show Context Menu** button again and select **Edit Recordset Structures**. The "Recordset Structures" dialog box appears.
5. Click **Define input parameters and call procedure**. The "Evaluate Stored Procedure" dialog box appears.



Evaluate Stored Procedure - [AdventureWorks2016].[HumanResources].[uspGetAllEmpl...

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

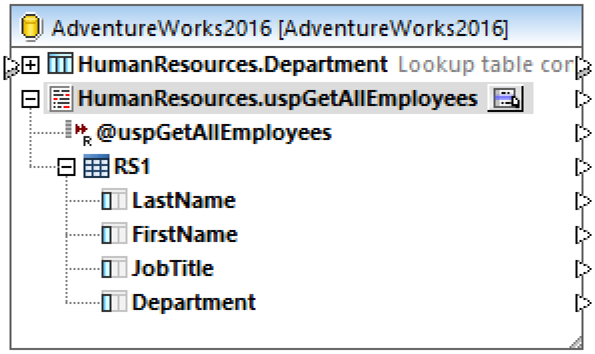
Warning: Executing a stored procedure can have side effects that cannot be rolled back depending on the procedure's implementation.

Execution result status:
Not yet executed.

Output parameters:

Name	Type	Value
@uspGetAllEmployees	int	*not yet retrieved*

6. Click **Execute**, and then click **OK**. The recordset structure ("RS1") is now visible both on the "Recordset Structures" dialog box and on the mapping.

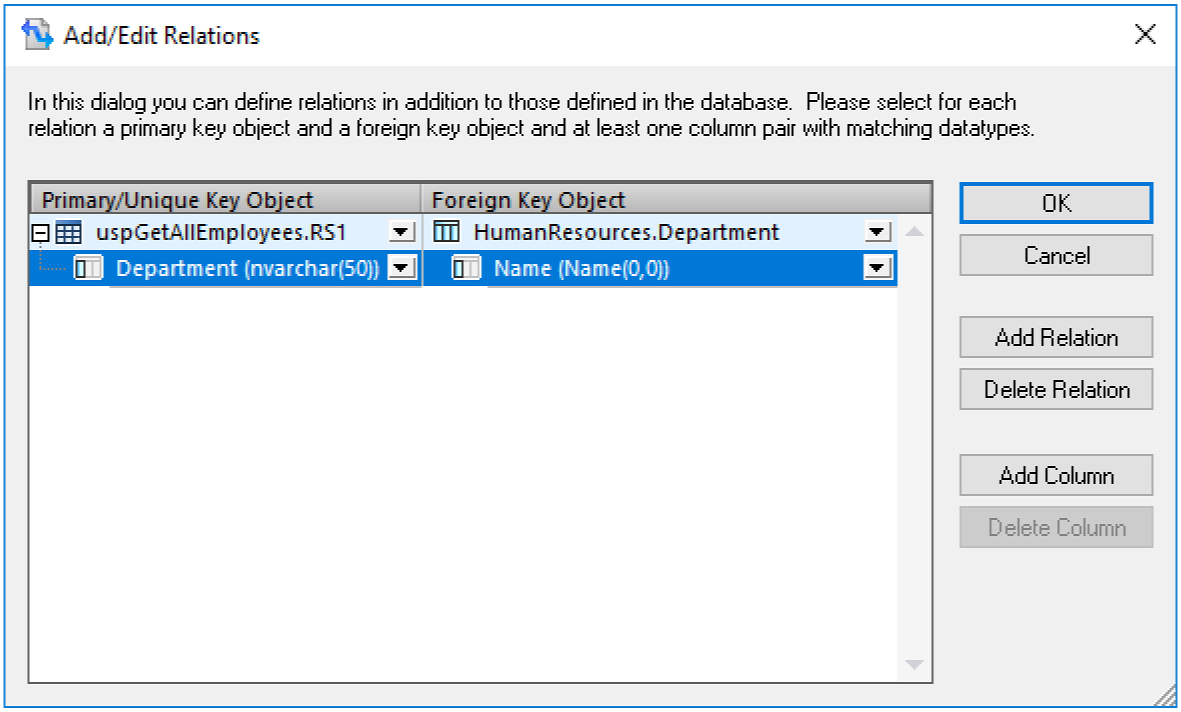


AdventureWorks2016 [AdventureWorks2016]
HumanResources.Department Lookup table cor
HumanResources.uspGetAllEmployees
@uspGetAllEmployees
RS1
LastName
FirstName
JobTitle
Department

Define the local relationships

Let's now define a local relationship between the **Department** column of the returned recordset and the **Name** column of the **Department** table.

1. Right-click the title bar of the database component and select **Add/Remove/Edit Database Objects** from the context menu.
2. Click **Add/Edit Relations**, and then click **Add Relation**. Define the relationships as shown below.

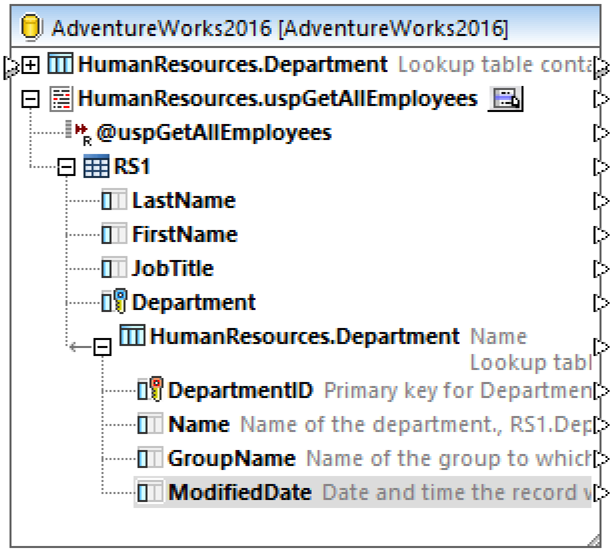


Add/Edit Relations

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
uspGetAllEmployees.RS1	HumanResources.Department
Department (nvarchar(50))	Name (Name(0,0))

3. Click **OK** to close the dialog box. Notice that the **Department** table has now become a child of the **RS1** recordset.



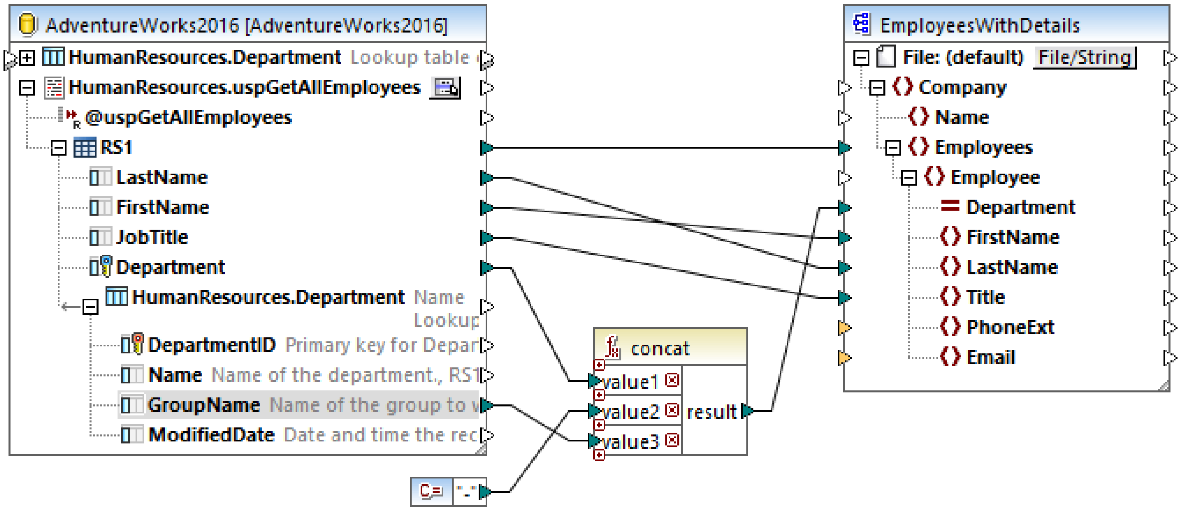
AdventureWorks2016 [AdventureWorks2016]
HumanResources.Department Lookup table conta
HumanResources.uspGetAllEmployees
@uspGetAllEmployees
RS1
LastName
FirstName
JobTitle
Department
HumanResources.Department Name
Lookup tabl
DepartmentID Primary key for Departmen
Name Name of the department., RS1.Dep
GroupName Name of the group to which
ModifiedDate Date and time the record w

Completing the mapping

Thanks to the relationship that was just created, it is now possible to map data from the recordset combined with data from the table. For the scope of this example, let's write data to a target XML file, as follows:

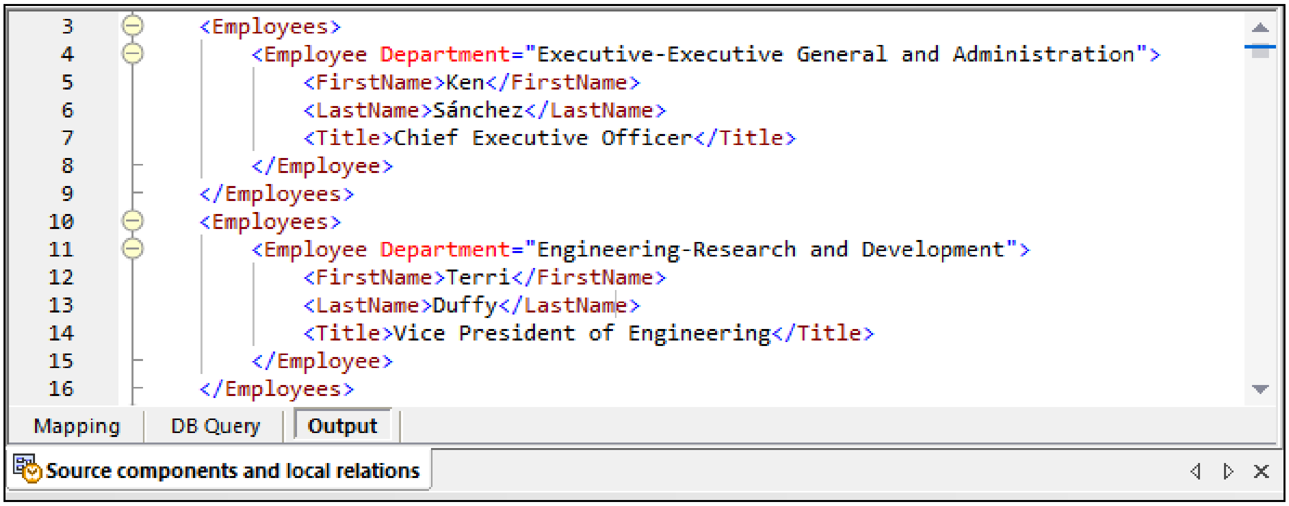
1. On the **Insert** menu, click **XML Schema/File** and select the following file: **<Documents>\Altova\MapForce2025\MapForceExamples\EmployeesWithDetails.xsd**.

2. When prompted to provide a sample XML instance file, click **Skip**.
3. Draw the mapping connections as shown below.


AdventureWorks2016 [AdventureWorks2016]
HumanResources.Department Lookup table
HumanResources.uspGetAllEmployees
@uspGetAllEmployees
RS1
LastName
FirstName
JobTitle
Department
HumanResources.Department Name Lookup
DepartmentID Primary key for Depar
Name Name of the department., RS1
GroupName Name of the group to
ModifiedDate Date and time the rec
"-"
concat
value1
value2
value3
result
EmployeesWithDetails
File: (default) File/String
Company
Name
Employees
Employee
Department
FirstName
LastName
Title
PhoneExt
Email

The mapping illustrated above writes data from the database to a target XML file. The source data is a combination of data extracted by the stored procedure with data extracted directly from a table. The mapping uses the **concat** 592 function to produce a string that includes the department name, followed by a dash character, followed by the group name.

To preview the mapping, click the **Output** button and observe the mapping result in the **Output** pane, for example:



```
<Employees>
    <Employee Department="Executive-Executive General and Administration">
        <FirstName>Ken</FirstName>
        <LastName>Sánchez</LastName>
        <Title>Chief Executive Officer</Title>
    </Employee>
</Employees>
<Employees>
    <Employee Department="Engineering-Research and Development">
        <FirstName>Terri</FirstName>
        <LastName>Duffy</LastName>
        <Title>Vice President of Engineering</Title>
    </Employee>
</Employees>
```

4.2.6.7 Using Stored Procedures to Generate Keys

This example shows you how to insert some key (ID) generated by a stored procedure into another table, with the help of local relations.

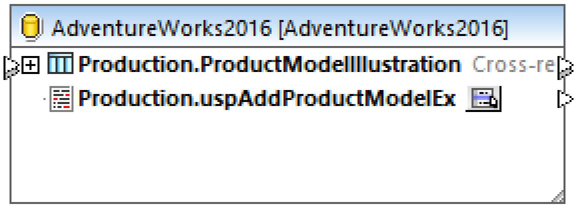
Let us first create the demo stored procedure in the "AdventureWorks" database. To do this, run the script below against the database. You can do this from a query window of **Microsoft SQL Server Management Studio**, or directly from the **DB Query** pane of MapForce, see Browsing and Querying Databases 277. In either case, make sure that your database user account has permission to create stored procedures.

```
CREATE PROCEDURE Production.uspAddProductModelEx
  @ModelName nvarchar(50)
  ,@Inst xml
  ,@ProductModelID int OUTPUT
AS
BEGIN
INSERT INTO [Production].[ProductModel]
           ([Name]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           ,@Inst
           ,NEWID()
           ,GETDATE())
   SELECT @ProductModelID = SCOPE_IDENTITY()
END
```

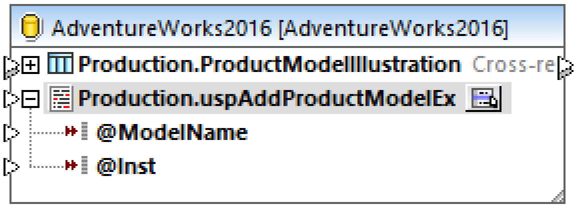
The stored procedure above takes two parameters (`@ModelName`, `@Inst`) as input and performs an `INSERT` operation into the **ProductModel** table. It then returns the generated `@ProductModelID` as output parameter. The requirement is to insert the `@ProductModelID` returned by the stored procedure into the **ProductModelIllustration** table.

The following steps show you how to create a mapping that satisfies the requirement above.

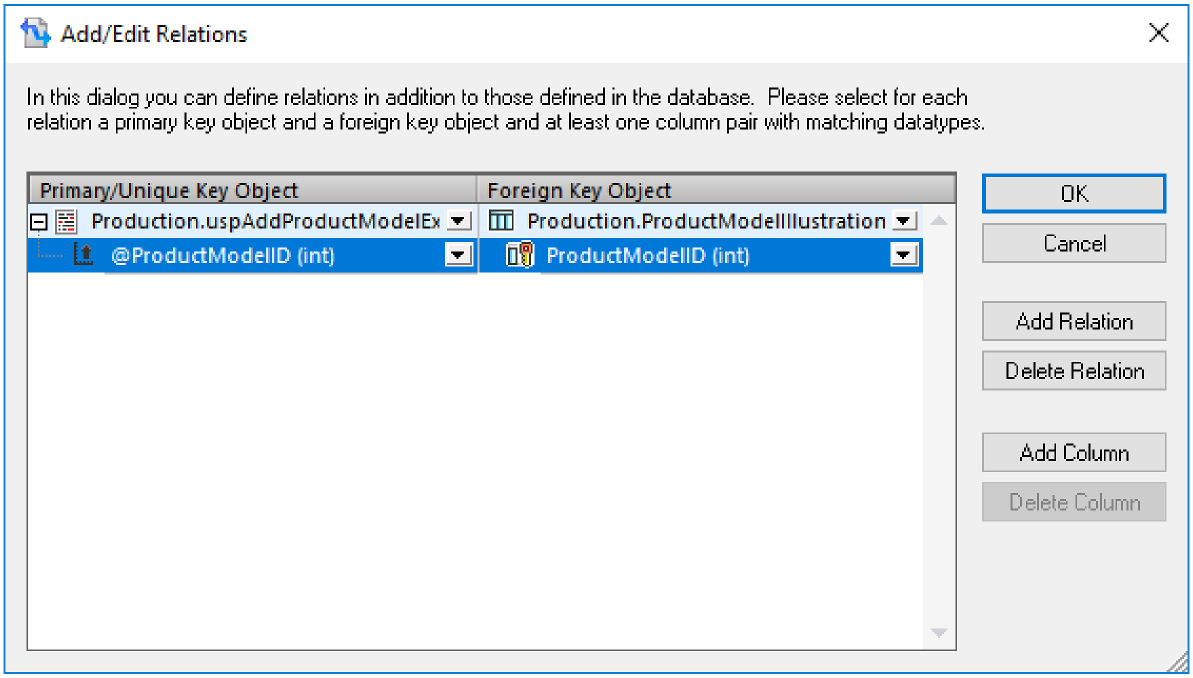
1. Connect to the "AdventureWorks" database from MapForce, as described in Adding Stored Procedures to the Mapping 300. Make sure that your database user account has permission to view and execute stored procedures.
2. When prompted to choose database objects, select the **ProductModelIllustration** table and the **uspAddProductModelEx** stored procedure.


AdventureWorks2016 [AdventureWorks2016]
Production.ProductModelIllustration Cross-re
Production.uspAddProductModelEx

3. Click the **Show Context Menu** button next to the stored procedure and select **Show Nodes As Target**. The stored procedure now appears as target component on the mapping, where the left side lists the input parameters.


AdventureWorks2016 [AdventureWorks2016]
Production.ProductModelIllustration Cross-re
Production.uspAddProductModelEx
@ModelName
@Inst

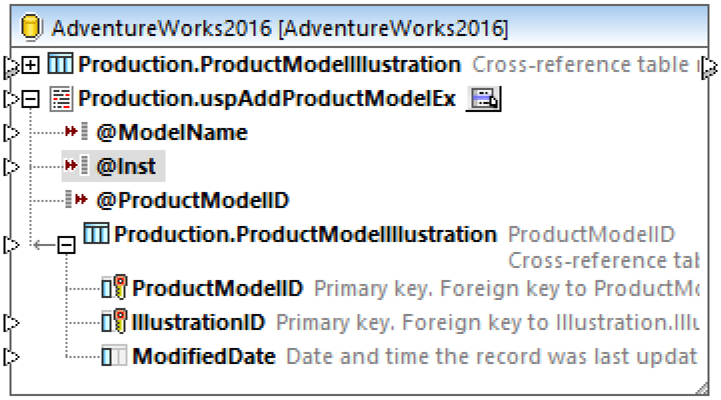
4. Optionally, if you want to execute the stored procedure inside a transaction, click the **Show Context Menu** button again, select **Procedure Settings**, and then select the **Use Transactions** check box. Defining the transaction for the stored procedure ensures that retrieving the key and inserting the record occur during the same transaction.
5. Right-click the title bar of the database component and select **Add/Remove/Edit Database Objects** from the context menu.
6. Click **Add/Edit Relations**, and then click **Add Relation**. Define the relationships as shown below.


Add/Edit Relations

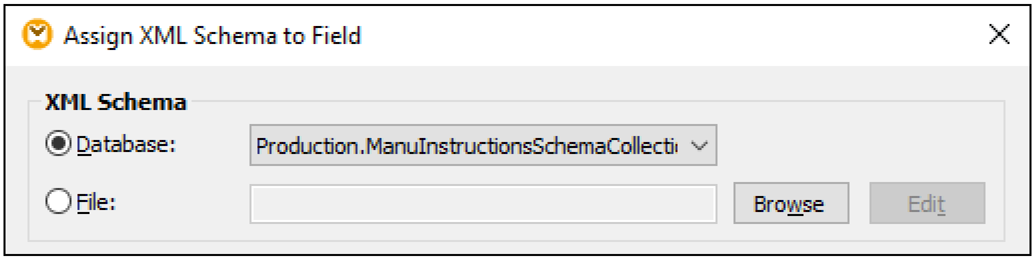
In this dialog you can define relations in addition to those defined in the database. Please select for each relation a primary key object and a foreign key object and at least one column pair with matching datatypes.

Primary/Unique Key Object	Foreign Key Object
Production.uspAddProductModelEx	Production.ProductModelIllustration
@ProductModelID (int)	ProductModelID (int)

7. Click **OK** to close the dialog box. Notice that the **ProductModelIllustration** table now appears as a child of the stored procedure. The stored procedure output parameter (`@ProductModelID`) is displayed as an indicator that it will be used in the local relation, but it does not have any input or output connectors.


AdventureWorks2016 [AdventureWorks2016]
Production.ProductModelIllustration Cross-reference table
Production.uspAddProductModelEx
@ModelName
@Inst
@ProductModelID
Production.ProductModelIllustration ProductModelID Cross-reference tal
ProductModelID Primary key. Foreign key to ProductMo
IllustrationID Primary key. Foreign key to Illustration.Illu
ModifiedDate Date and time the record was last updat

8. In this example, the `@Inst` parameter is of XML type. Right-click the `@Inst` parameter on the component and select **Assign XML Schema to Field** from the context menu. Next, select the **Production.ManuInstructionsSchemaCollection** schema from the database. When prompted to select a root element, leave the default value as is, and click **OK**. For more information about mapping data to database XML fields, see Mapping XML Data to / from Database Fields 287.

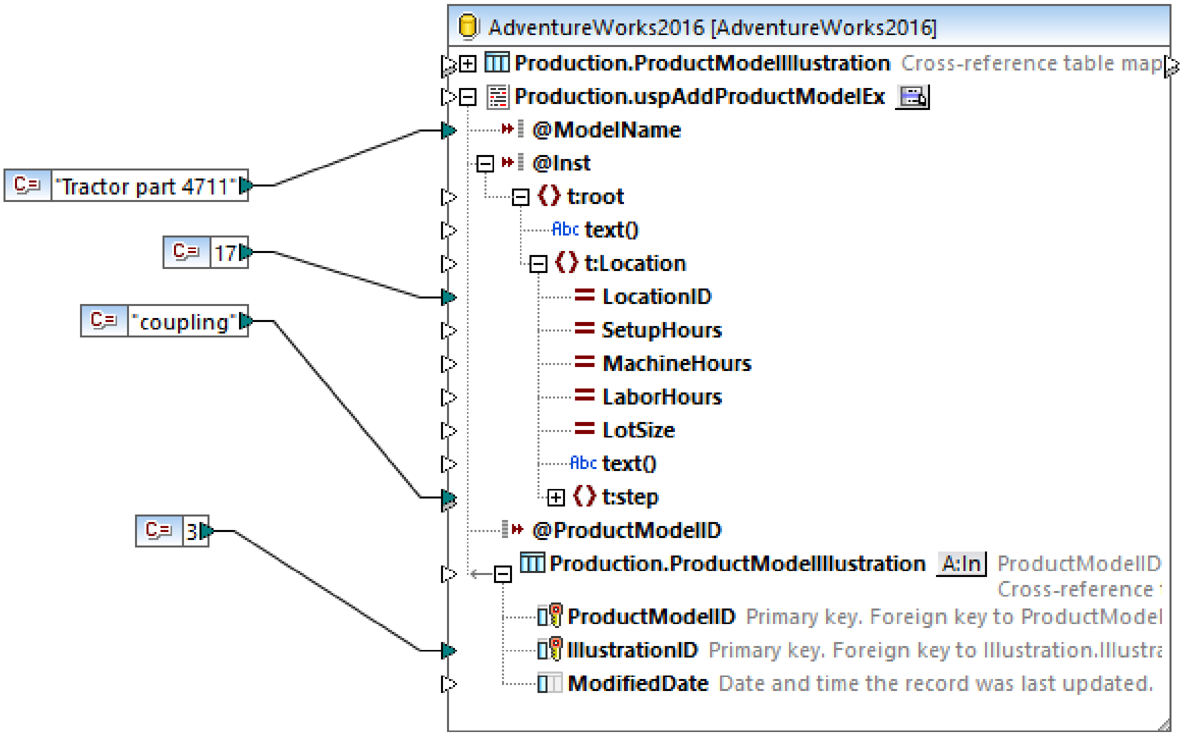

Assign XML Schema to Field

XML Schema

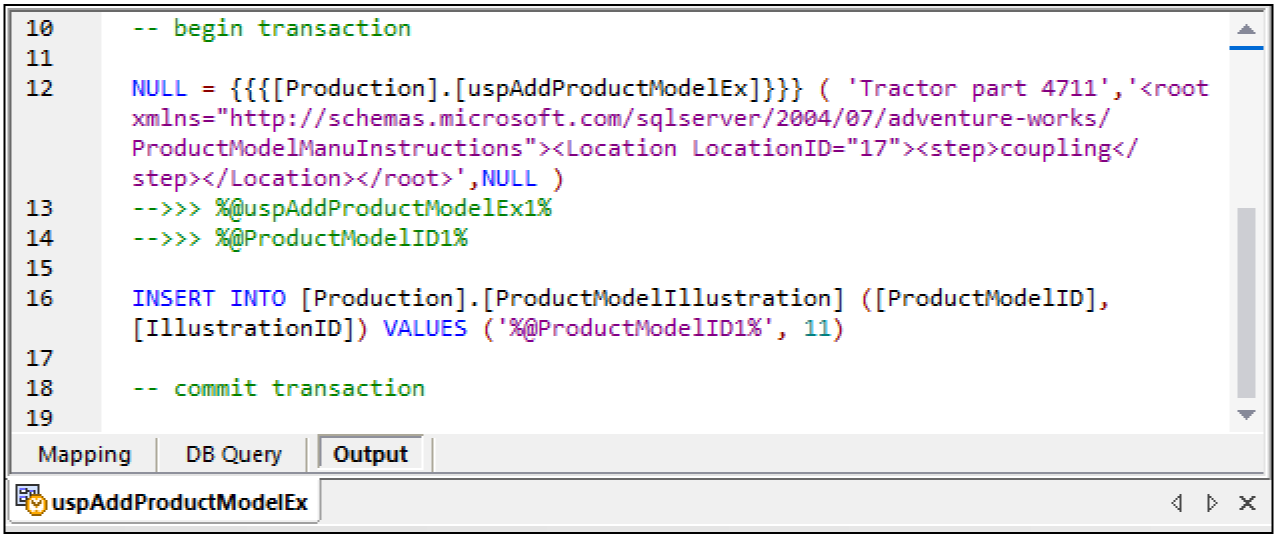
● Database: Production.ManuInstructionsSchemaCollecti

○ File:

9. Add the source components that provides data to be inserted into the database. In this example, the source data is supplied by constants; however, any other source component supported by MapForce could act as input. For more information about constants, see Add a Constant to the Mapping 437.


AdventureWorks2016 [AdventureWorks2016]
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 Primary key. Foreign key to ProductModel
IllustrationID Primary key. Foreign key to Illustration.Illustra
ModifiedDate Date and time the record was last updated.
"Tractor part 4711"
17
"coupling"
3

Since this mapping updates a database, you do not preview its output directly like with other mappings. Instead, click the **Output** button to display the pseudo-SQL containing hints about how the database will be modified. If you enabled transactions, these will occur as indicated by the comments.



```
-- begin transaction

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 )
-->>> %@uspAddProductModelEx1%
-->>> %@ProductModelID1%

INSERT INTO [Production].[ProductModelIllustration] ([ProductModelID], [IllustrationID]) VALUES ('%@ProductModelID1%', 11)

-- commit transaction

```

The pseudo-SQL displayed in the **Output** pane does not show the actual transaction commands, only hints (as comments). The actual SQL commands are sent to the underlying database API, however.

To run the mapping against the database, do one of the following:

- On the **Output** menu, click **Run SQL-Script**.
- Click the **Run SQL-Script** toolbar button.

4.3 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. In MapForce, CSV and FLF files are structural components 34 that can be used as data sources and targets.

An FLF is a common text format where data is conventionally separated into fields which have a fixed length (for example, the first 5 characters of every row represent a transaction ID, and the next 20 characters represent a transaction description).

Note that, in the 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.

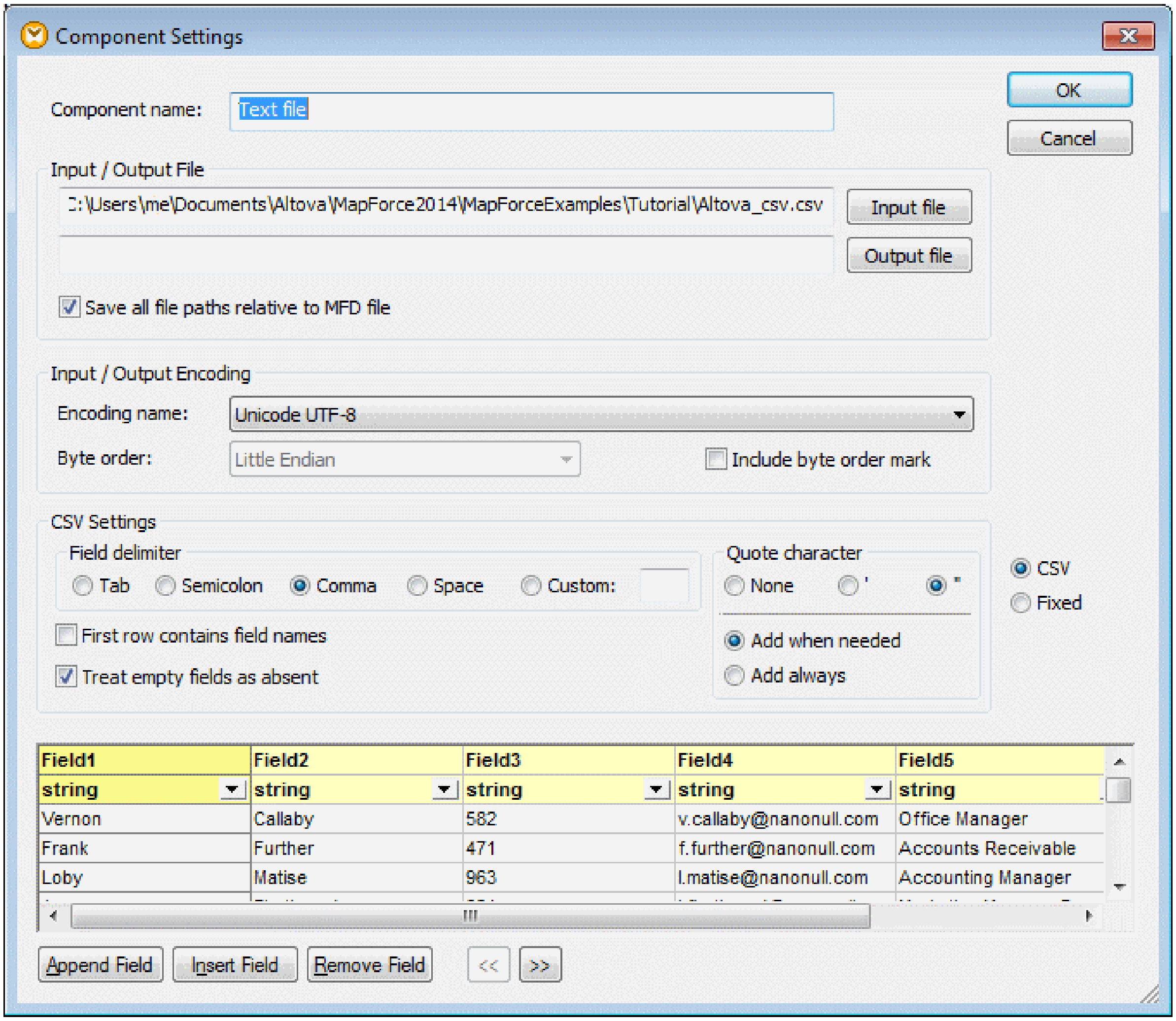
There are two ways that mapped flat file data can be generated:

- By clicking the Output pane which generates a preview using the Built-in execution engine. You can also save the mapping result by selecting the menu option **Output | Save output file**, or clicking the icon.
- By selecting **File | Generate code in | Java, C#, or C++** , and then compiling and executing the generated code.

4.3.1 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\MapForce2025\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

:\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

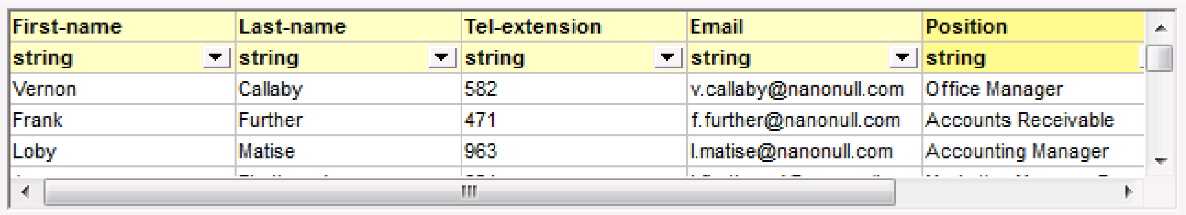
□ First row contains field names

☑ Treat empty fields as absent

● CSV ○ Fixed

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

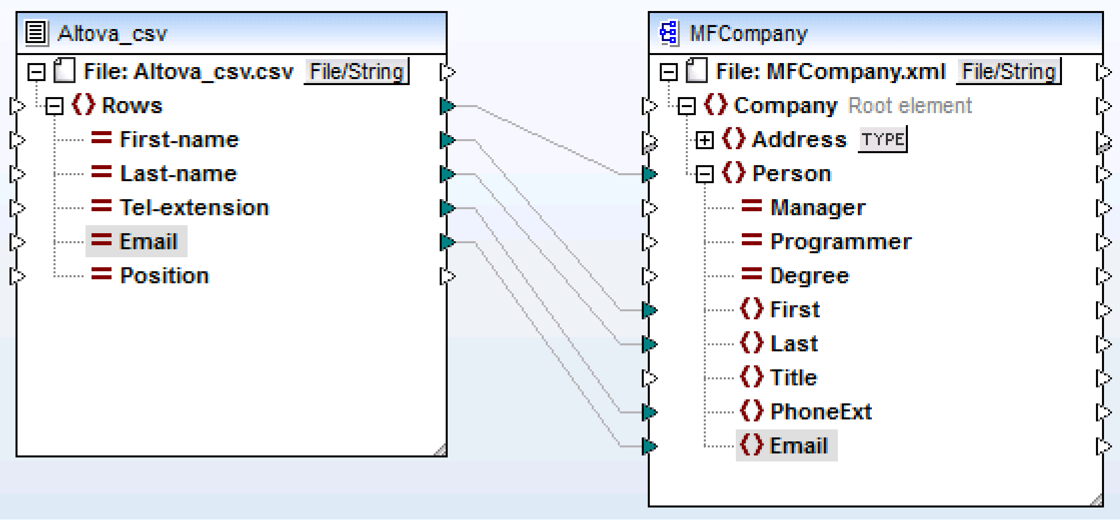
4. Click inside the **Field1** header and change the text to First-name. Do the same for all the other fields, as follows: Field 2 => Last-name, Field 3 =>Tel-extension, Field 4 => Email, Field 5 => Position. TIP: Press the **Tab** key to cycle through all the fields: header1, header2 etc.



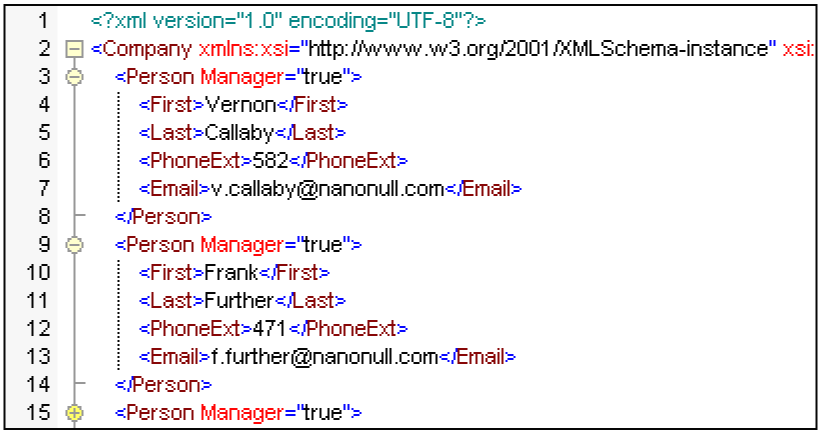
First-name	Last-name	Tel-extension	Email	Position
string	string	string	string	string
Vernon	Callaby	582	v.callaby@nanonull.com	Office Manager
Frank	Further	471	f.further@nanonull.com	Accounts Receivable
Loby	Matise	963	l.matise@nanonull.com	Accounting Manager

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 pane to see the result.

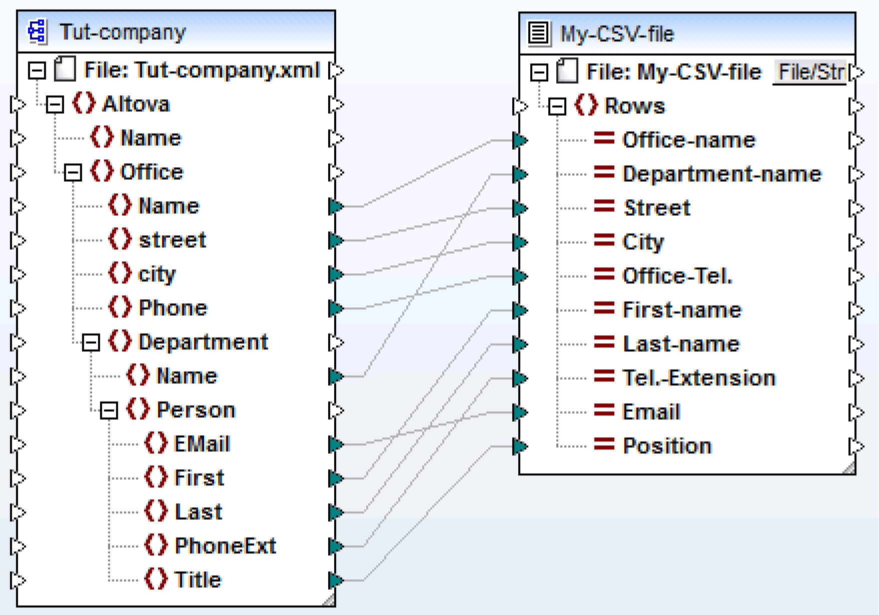


```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
  <Person Manager="true">
    <First>Vernon</First>
    <Last>Callaby</Last>
    <PhoneExt>582</PhoneExt>
    <Email>v.callaby@nanonull.com</Email>
  </Person>
  <Person Manager="true">
    <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.

4.3.2 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\MapForce2025\MapForceExamples\Tutorial\Tut-xml2csv.mfd**.

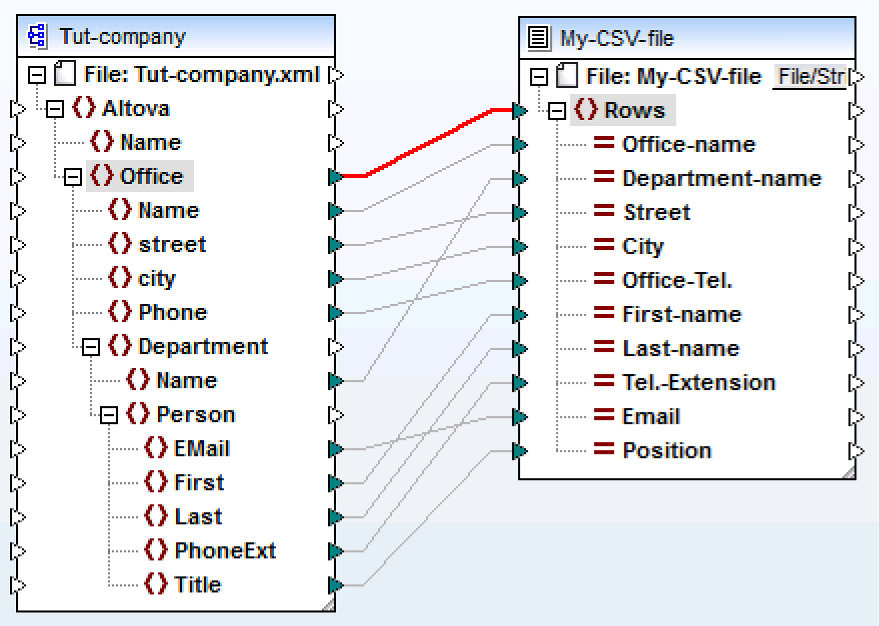


Tut-xml2csv.mfd

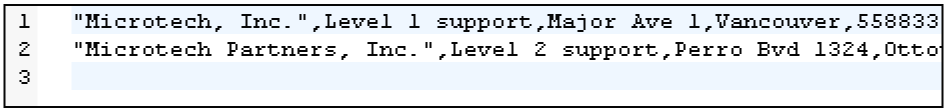
This mapping has been intentionally created as incomplete. If you attempt to validate the example file using the menu command **File | Validate Mapping**, you will notice that validation warnings occur. Also, if you preview the mapping output, a single row is produced, which may or may not be your intended goal.

Let's assume that your goal is to create multiple rows in the CSV file from a sequence of items in the XML file. You can achieve this by drawing a connection to the `Rows` item of the target CSV file.

For example, to iterate through all offices and have the output appear in the CSV file, it is necessary to connect `Office` to `Rows`. By doing this, you are instructing MapForce: for each `Office` item of the source XML, create a row in the target CSV file.

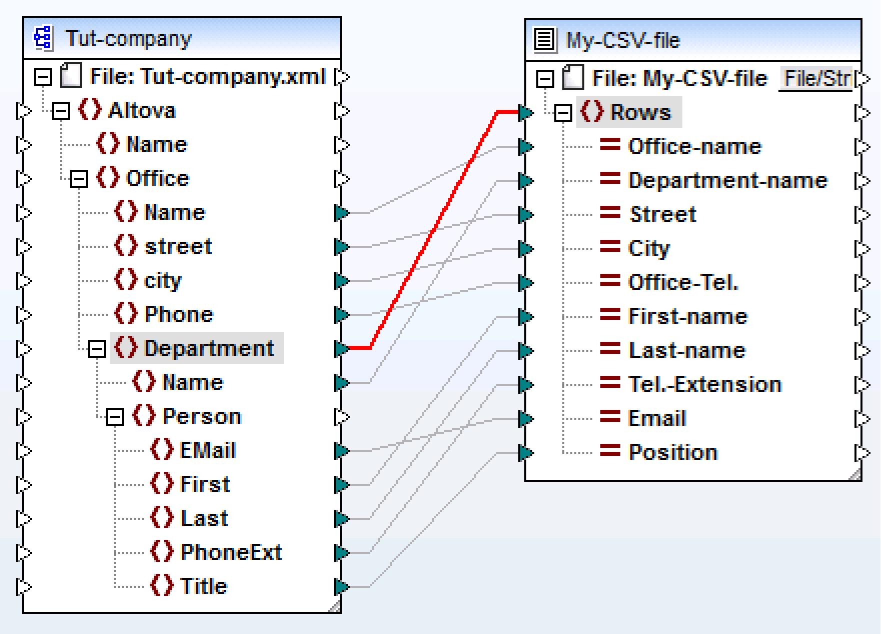


The Rows item in the CSV component acts as an iterator for the sequence of items connected to it. Therefore, if you connect the Office item, the output creates a row for each office found in the source XML.

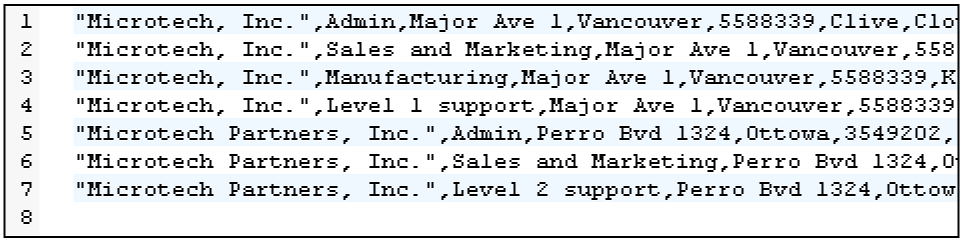


```
"Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,558833
"Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Otto
```

In a similar fashion, if you connect **Department** to the **Rows** item, a row will be produced for each department found in the source XML.



The output would then look as follows:



```
"Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Clive,Clo
"Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,558
"Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,K
"Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,5588339
"Microtech Partners, Inc.",Admin,Perro Bvd 1324,Ottowa,3549202,
"Microtech Partners, Inc.",Sales and Marketing,Perro Bvd 1324,O
"Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Ottow
```

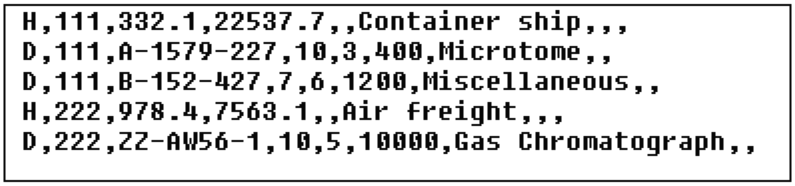
Finally, mapping `Person` to the `Rows` item results in all the Persons being output. In this case, MapForce will iterate through the records as follows: each Person within each Department, within each Office.

4.3.3 Example: Creating Hierarchies from CSV and Fixed-Length Text Files

This example is available at the following path: **<Documents>\Altova\MapForce2025\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.

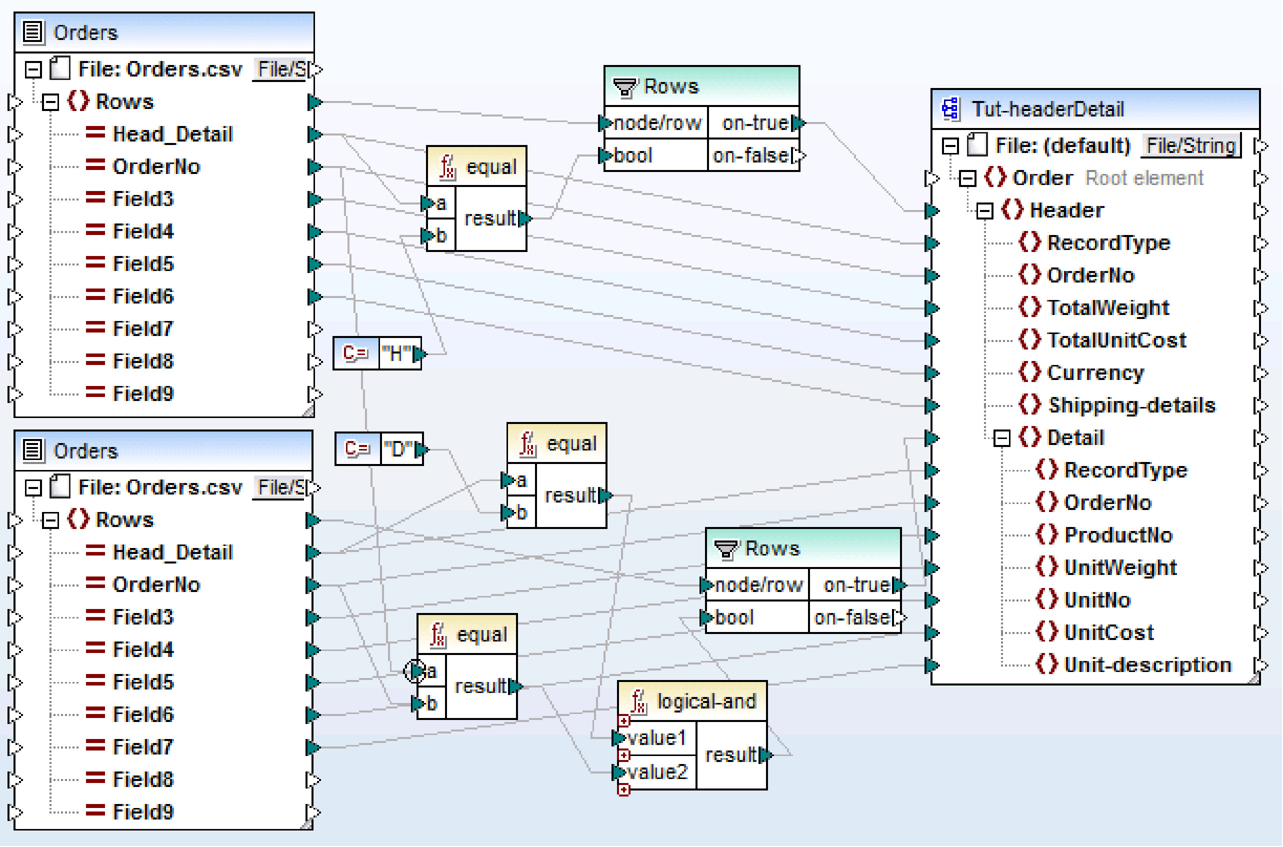
The contents of the Orders.csv file are shown below.



```
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.

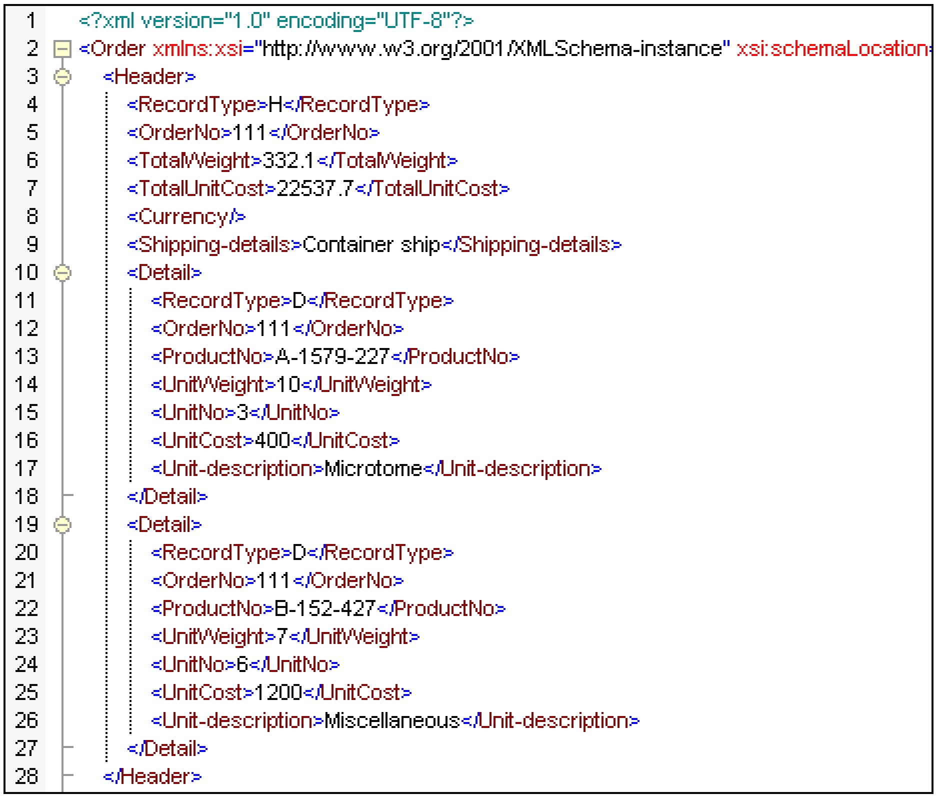
The first Orders.csv file supplies the **Header** records (and all mapped fields) to the Header item in the schema target file. The filter component is used to filter out all records other than those starting with H. The **Rows** item supplies these filtered records to the Header item in the schema file.

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 86 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 pane produces the XML file displayed below. Each Header record contains its data, and all associated Detail records that have the same Order No.

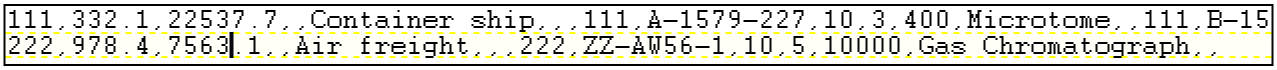


```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation
  <Header>
    <RecordType>H</RecordType>
    <OrderNo>111</OrderNo>
    <TotalWeight>332.1</TotalWeight>
    <TotalUnitCost>22537.7</TotalUnitCost>
    <Currency/>
    <Shipping-details>Container 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\MapForce2025\MapForceExamples\Tutorial\** folder as **Head-detail-inline.mfd**. The difference is that:

- No record designator (H, or D) is available

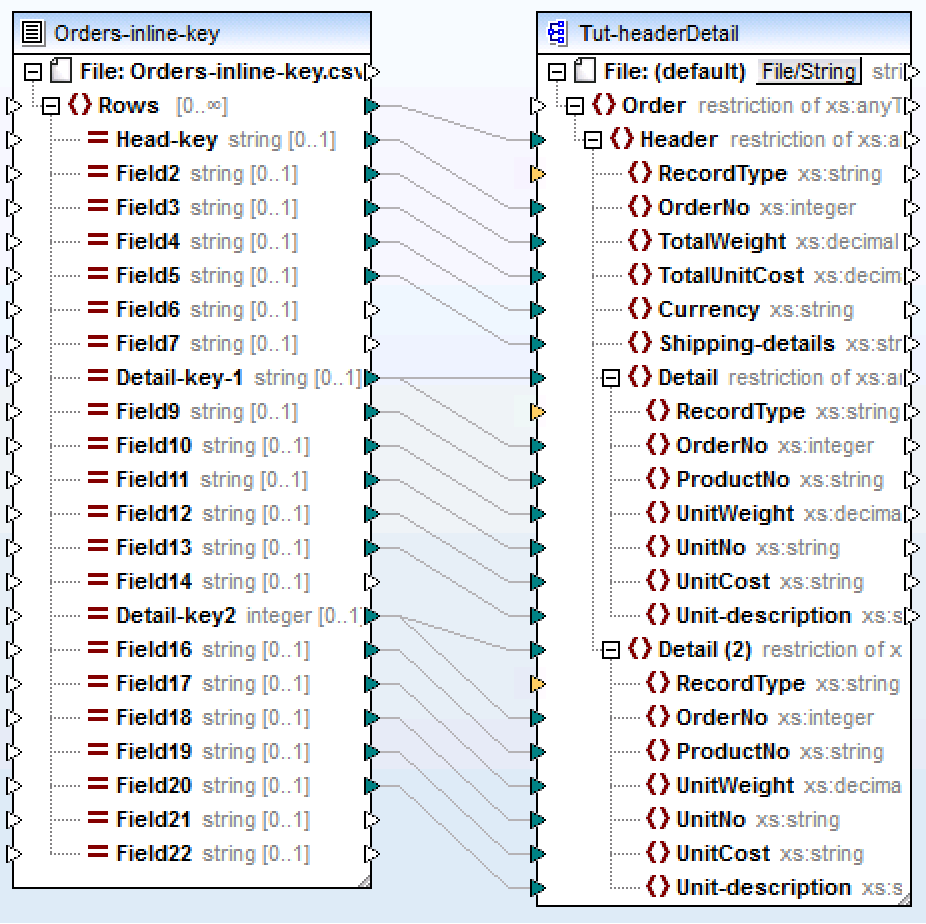
- A common key field, the first field of the CSV file, still exists for both header and detail records (Head-key, Detail-key...). The field is mapped to OrderNo in the schema target
- Header and all respective Detail fields are all on the same line.



```
111,332.1,22537.7,,Container ship,,,111,A-1579-227,10,3,400,Microtome,,111,B-15
222,978.4,7563.1,,Air freight,,,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,,
```

The mapping has been designed as follows:

- The key fields are mapped to the respective OrderNo items in the schema target.
- The Detail item in the schema target file has been duplicated, and is displayed as **Detail (2)**. This allows you to map the second set of detail records to the correct item.
- The result of this mapping is basically the same XML file that was produced in the first example.

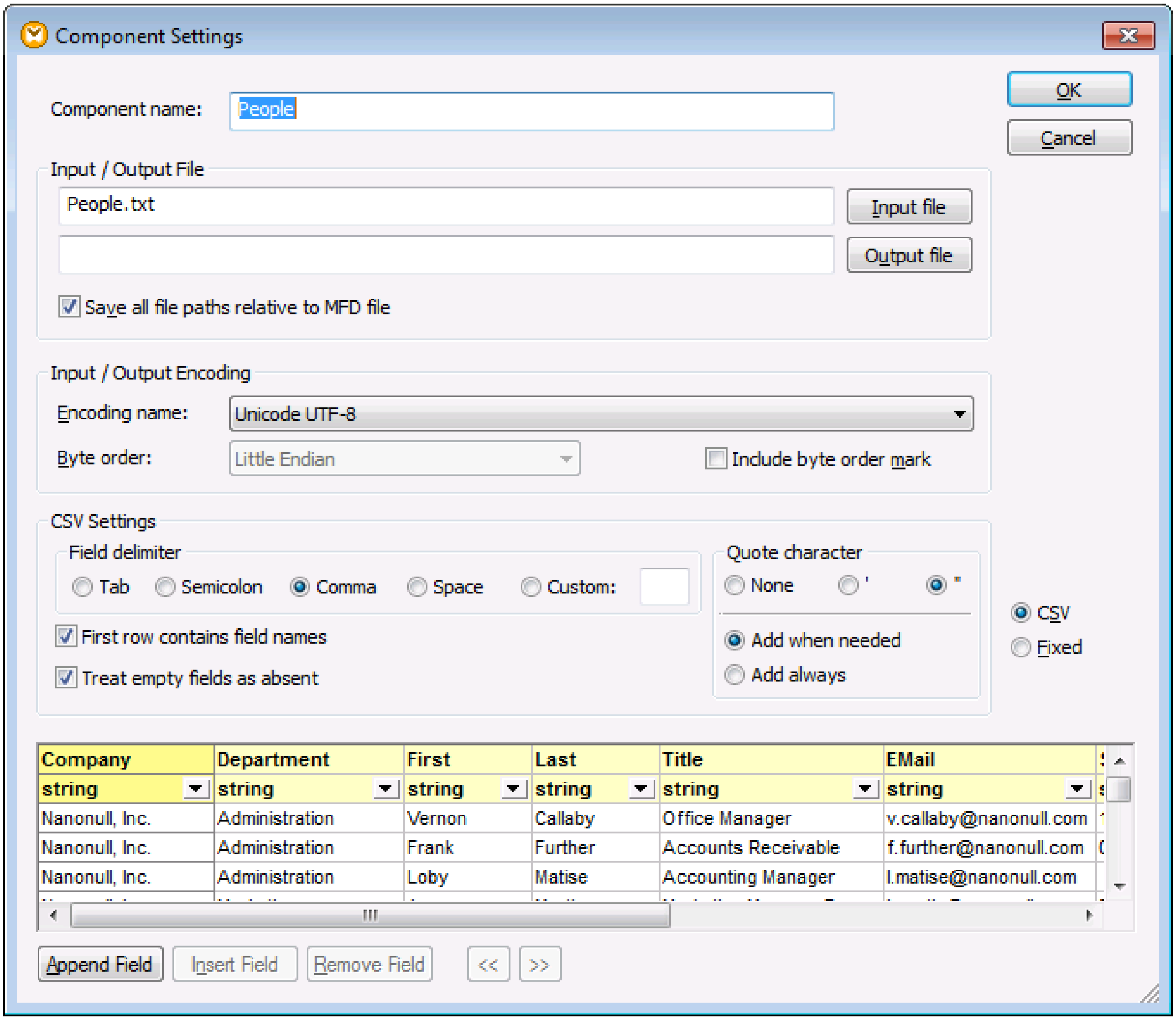


Head-detail-inline.mfd

4.3.4 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 Settings

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

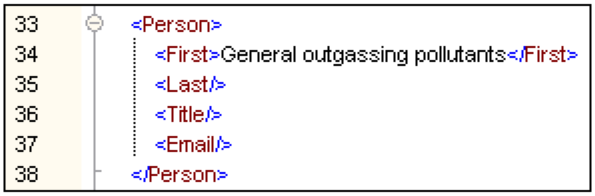
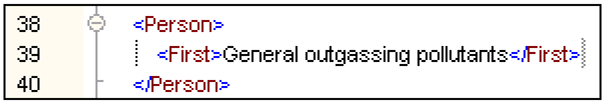
Company	Department	First	Last	Title	EMail
string	string	string	string	string	string
Nanonull, Inc.	Administration	Vernon	Callaby	Office Manager	v.callaby@nanonull.com
Nanonull, Inc.	Administration	Frank	Further	Accounts Receivable	f.further@nanonull.com
Nanonull, Inc.	Administration	Loby	Matise	Accounting Manager	l.matise@nanonull.com

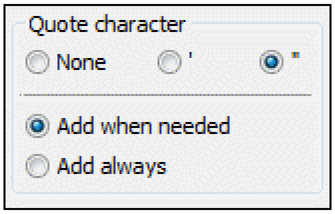
Text Component Settings dialog box (in CSV mode)

The available settings are as follows.

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

	• If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.
Input file	Specifies the file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The field can remain empty if you are using the text file component as a target for your mapping. In a source component, MapForce uses the value of this field to read column names and preview the contents of the instance text file. To select a new file, click **Input File**.
Output file	Specifies the file to which MapForce will write data. This field is meaningful for a target component. To select a new file, click **Output File**.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the input and output files used by the text component. See also Using Relative Paths on a Component 43.
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 only thos fields where the text contains the field delimite line breaks. **Add always** Adds the selected quote character to all fields 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 record.

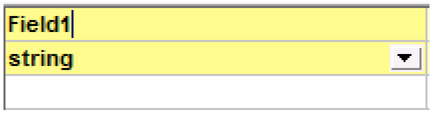
Insert field Creates a new field immediately before the currently selected CSV record.

Remove field Deletes the currently selected field.

<< Moves the currently selected field one position the left.

>> Moves the currently selected field one position the right.

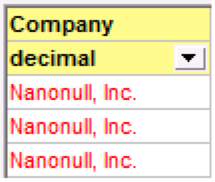
To change the name of a field, click the header (for example, **Field1**), and type the new value. Note that the field names are not editable when the **First row contains field names** option is enabled.



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**.

4.3.5 FLF to Database

This topic explains how to map data of a fixed-length text file (FLF) to an SQLite database. The files used in this topic are available in the **`Tutorial`** folder. The source text file and the target database store a list of employees. In the source file, the records are delimited by their size as follows:

Field position and name	Size (in characters)
Field 1 (First name)	8
Field 2 (Last name)	10
Field 3 (Phone extension)	3
Field 4 (Email)	25

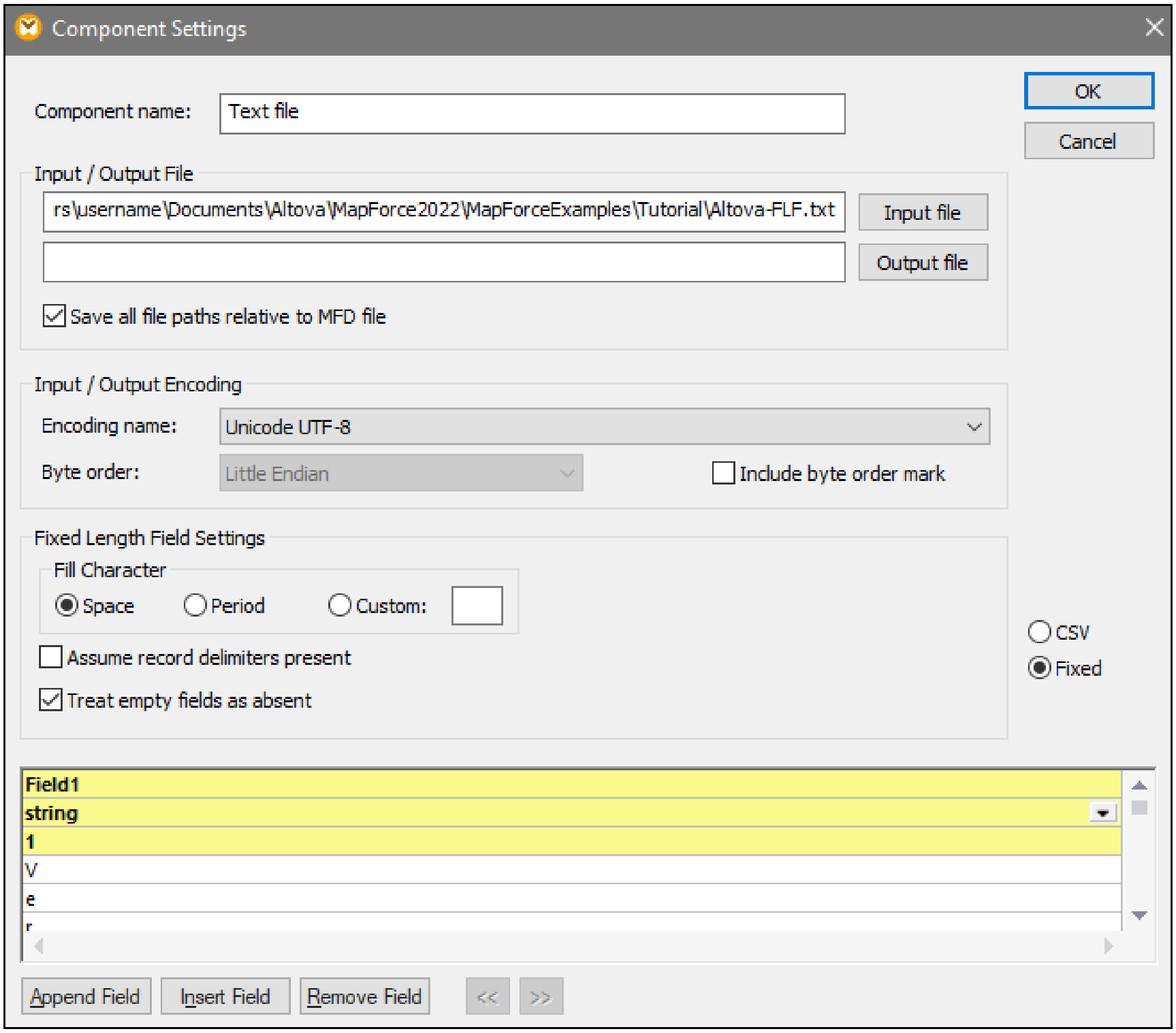
Field position and name	Size (in characters)
Field 5 (Position)	25

The goal of the mapping is to map the FLF data to the database component. We also want to map the phone extensions with a new prefix. To achieve this, take the steps below.

Step 1: Insert a text component

The first step is to add and configure a text component. Follow the instructions below:

1. Select the menu item **Insert | Text file** or click the toolbar button (**Insert Text file**).
2. Click **Input file** in the **Component Settings** dialog box (*see below*) and select **`Altova-FLF.txt`**.



Component Settings

Component name: Text file

Input / Output File

rs\username\Documents\Altova\MapForce2022\MapForceExamples\Tutorial\Altova-FLF.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

Fixed Length Field Settings

Fill Character

● Space ○ Period ○ Custom:

□ Assume record delimiters present

☑ Treat empty fields as absent

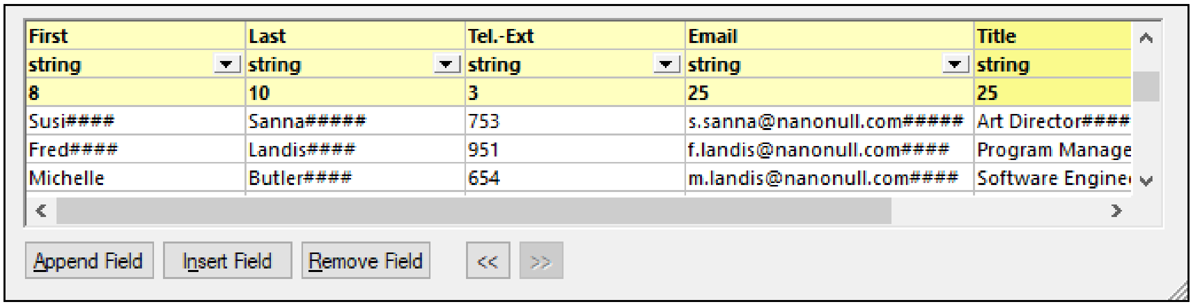
○ CSV

● Fixed

Field1
string
1
V
e
r

3. Select **Fixed**.
4. Clear the **Assume record delimiters present** check box.

5. The yellow rows are editable and enable you to specify i) the field name, ii) the data type, and iii) the field size. Set the field size to 8 (the third yellow line from the top) and press **Enter**. More data is now visible in the first column, which is now 8 characters wide.
6. Click **Append Field** to add a new field and set the length of the second field to 10 characters.
7. Use the same method to create three more fields of the following lengths: 3, 25, and 25 characters. Then change the field headers, as shown in the screenshot below.



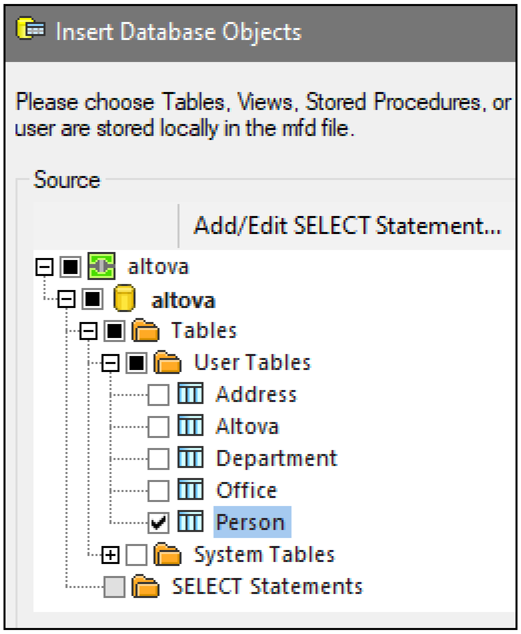
First	Last	Tel.-Ext	Email	Title
string	string	string	string	string
8	10	3	25	25
Susi####	Sanna#####	753	s.sanna@nanonull.com#####	Art Director####
Fred####	Landis####	951	f.landis@nanonull.com####	Program Manage
Michelle	Butler####	654	m.landis@nanonull.com####	Software Engine

8. In **Fixed Length Field Settings**, select **Custom** and type the hash (#) character. This instructs MapForce to treat the # character as a fill character.
9. Click **OK**.
10. MapForce will ask you if you would like to change the component name to match the instance files. Click **Change component name**. The `Altova-FLF` component appears in the mapping window.

Step 2: Insert a database component

The next step is to add a database component. Follow the steps below:

1. Go to **Insert | Database**, select **SQLite**, and click **Next**.
2. Select the **`Altova.sqlite`** database and click **Connect**.
3. Select the **Person** table (*see below*) and click **OK**.



Insert Database Objects

Please choose Tables, Views, Stored Procedures, or user are stored locally in the mfd file.

Source

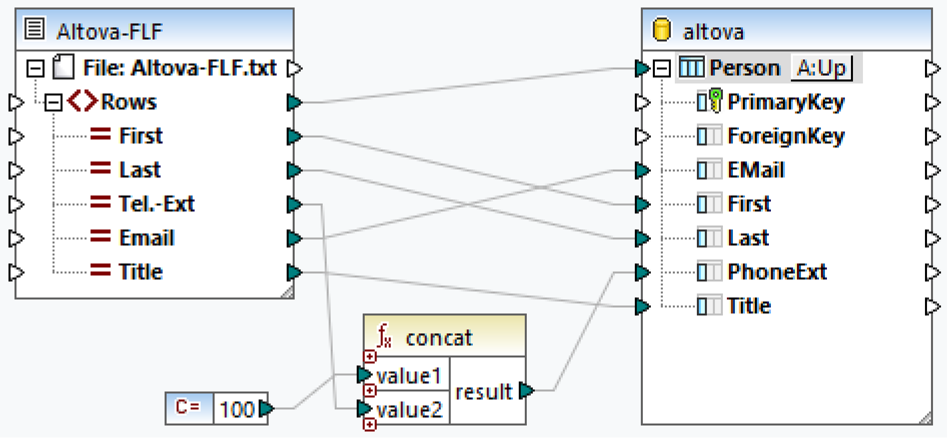
Add/Edit SELECT Statement...

- ■ altova
  - ■ altova
    - ■ Tables
      - ■ User Tables
        - ☐ Address
        - ☐ Altova
        - ☐ Department
        - ☐ Office
        - ☑ Person
      - ☐ System Tables
    - ☐ SELECT Statements

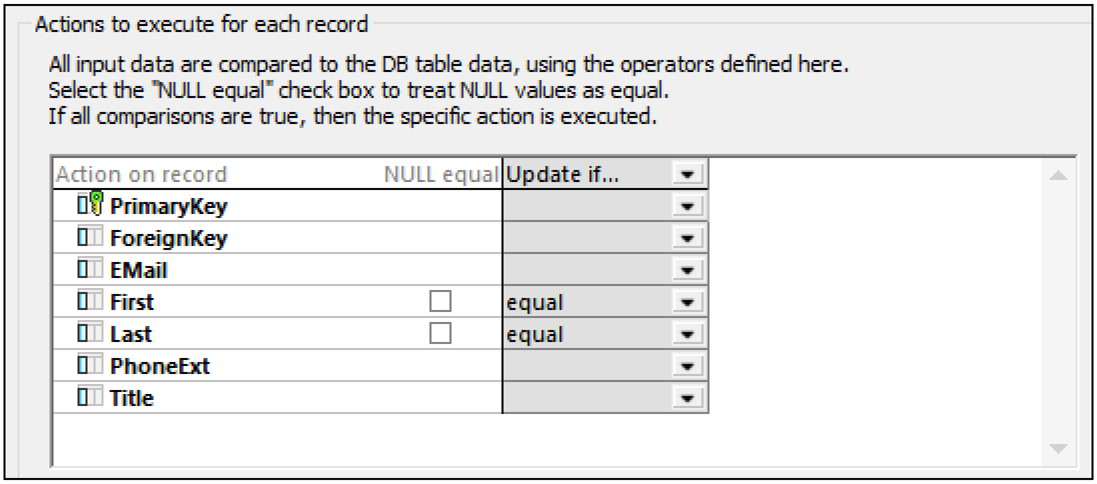
Step 3: Design the mapping

The next step is to create a mapping:

1. Drag the core | concat 592 function from the **Libraries** window into the mapping.
2. Go to **Insert | Constant**, select *Number* as a type, and enter 100 in the text field. This constant stores the new telephone extension prefix.
3. Create connections as shown below.



4. In the database component, click the **Table Action** button next to **Person**. This opens the **Database Table Actions** dialog box (*see screenshot below*).
5. Next to *Action on record data*, select **Update if**. Set the `equal` action for the **First** and **Last** fields. Click **OK**. MapForce will be instructed to update the `Person` table only if the first and last names in the source file are equal to the corresponding database fields. When this condition is true, the telephone extension will get a prefix 100 and will be copied to the `PhoneExt` field of the `Person` table.



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.

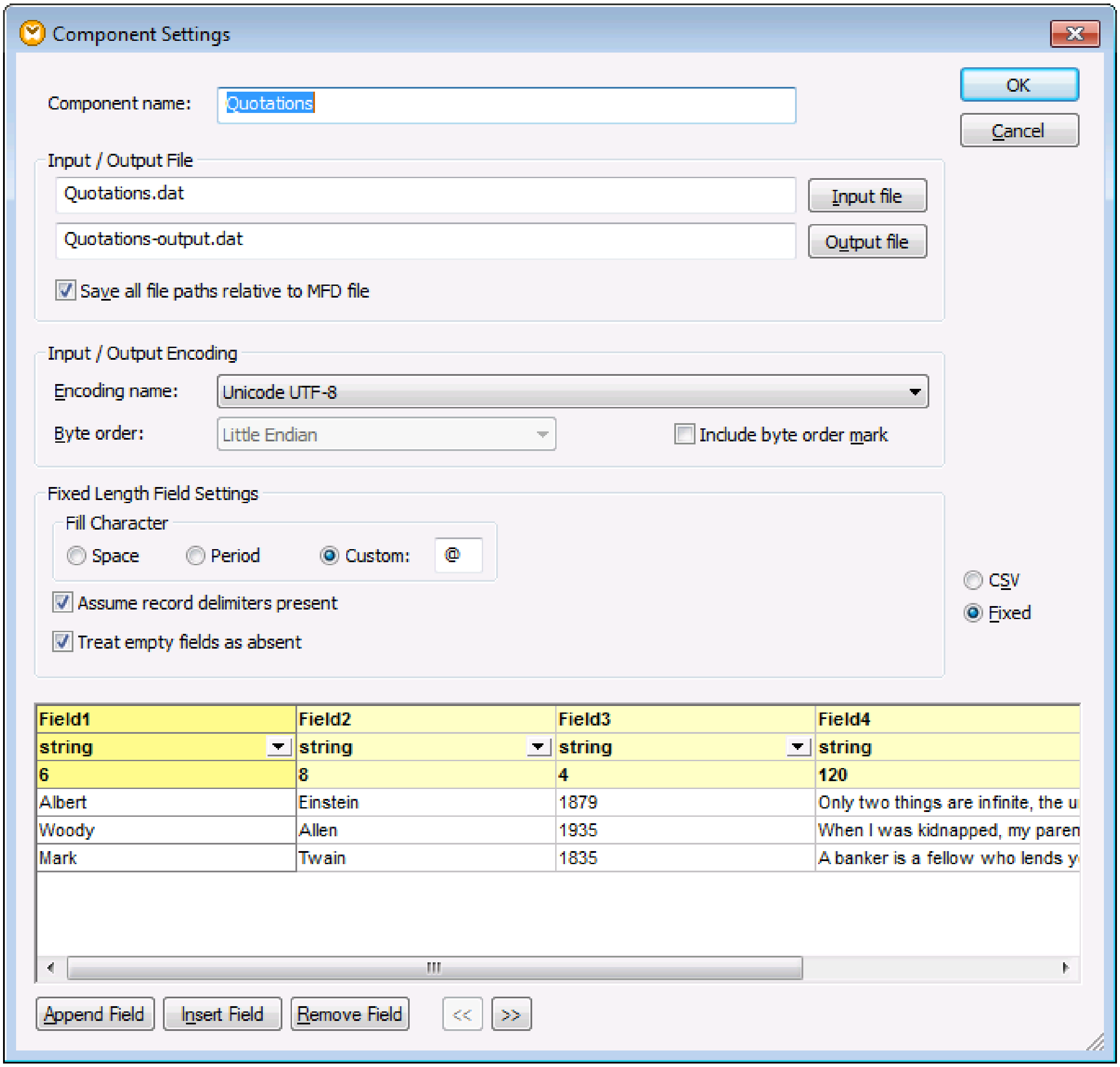
Action on record	NULL equal	Update if...
PrimaryKey		
ForeignKey		
EMail		
First	□	equal
Last	□	equal
PhoneExt		
Title		

6. To generate the SQL statements (for preview in MapForce), click the **Output** pane. To run the SQL statements against the database, click the **Run SQL-script** button.

4.3.6 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

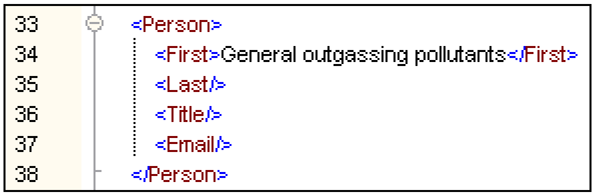
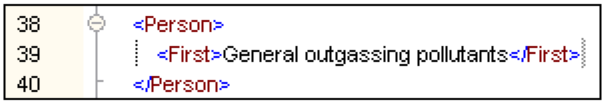
Field1	Field2	Field3	Field4
string	string	string	string
6	8	4	120
Albert	Einstein	1879	Only two things are infinite, the u
Woody	Allen	1935	When I was kidnapped, my paren
Mark	Twain	1835	A banker is a fellow who lends y

Text Component Settings dialog box (in fixed-length field mode)

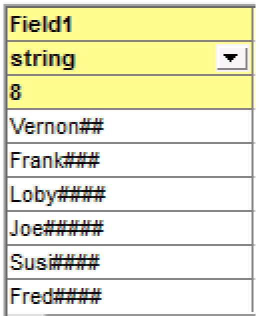
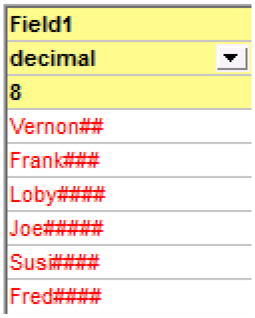
The available settings are as follows.

Component name	The component name is automatically generated when you create a component. However, you can change the name at any time. The component name can contain spaces and full stops. It may not contain slashes, backslashes, colons, double quotes, leading and trailing spaces. If you want to change the name of the component, be aware of the following: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have a different encoding in Windows and at the command line.
Input file	Specifies the file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The field can remain empty if you are using the text file component as a target for your mapping. In a source component, MapForce uses the value of this field to read column names and preview the contents of the instance text file. To select a new file, click **Input File**.
Output file	Specifies the file to which MapForce will write data. This field is meaningful for a target component. To select a new file, click **Output File**.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the input and output files used by the text component. See also Using Relative Paths on a Component[43].
Input / Output Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab.
Fill Character	This option allows you to define the characters that are to be used to complete, or fill in, the rest of the (fixed) field when the incoming data is shorter than the respective field definitions. The custom field allows you to define your own fill character.

	If the incoming data already contains specific fill characters, and you enter the same fill character in the Custom field, then the incoming data will be stripped of those fill characters!
Assume record delimiters present	This option is useful when you want to read data from a source flat file that does not contain record delimiters such as CR/LF, or when you want to produce a target flat FLF file without record delimiters. See the Understanding the "Assume record delimiters present" option 342 section below.

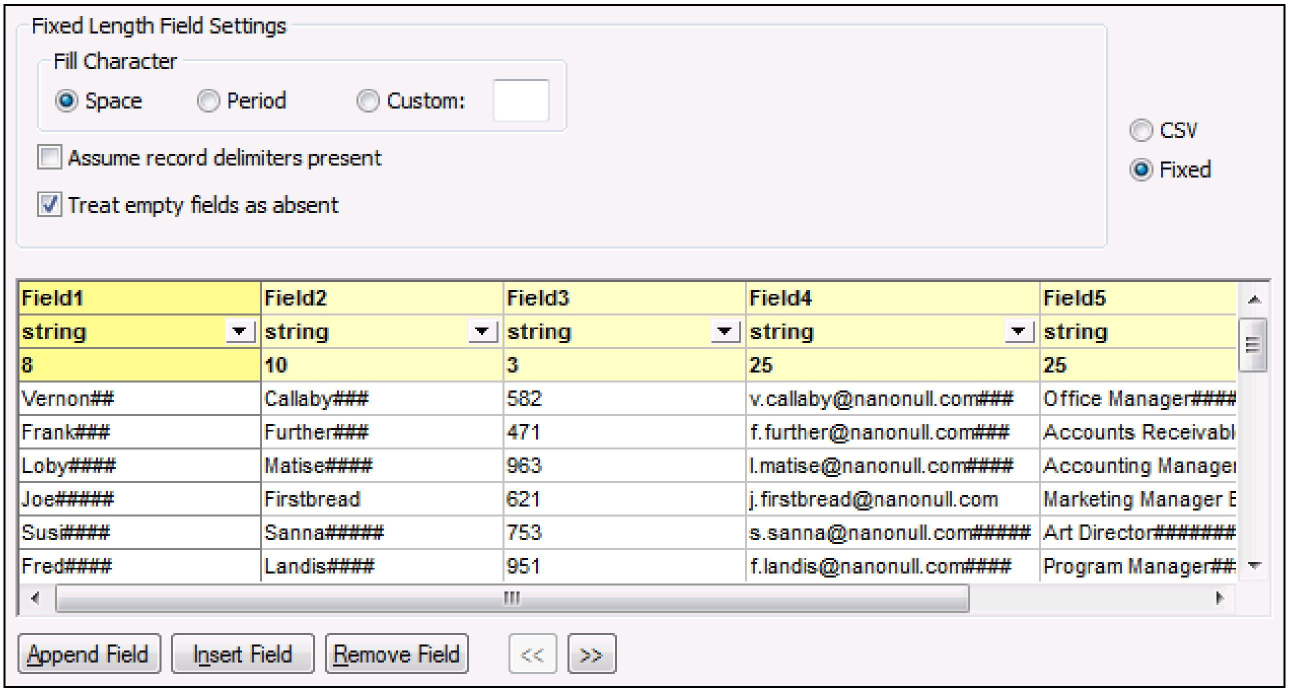
Treat empty fields as absent	When this option is enabled, empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file.
	Assuming that the output is an XML file, when this option is disabled, the empty fields will be created in the output with an empty value (in this example, the elements `Last`, `Title`, and `Email`):
	
	`<Person>`
	`  <First>General outgassing pollutants</First>`
	`  <Last/>`
	`  <Title/>`
	`  <Email/>`
	`</Person>`
	When this option is enabled, the empty fields will not be created in the output:
	
	`<Person>`
	`  <First>General outgassing pollutants</First>`
	`</Person>`

CSV / Fixed	Changes the component type to either CSV or FLF (fixed-length field).
Preview area	The lower part of the dialog box displays a preview of up to 20 rows of the file selected as input or output. If necessary, you can create the structure of the file (or change the structure of the existing one), as follows. **Append field** Creates a new field after the last record. **Insert field** Creates a new field immediately before the currently selected record. **Remove field** Deletes the currently selected field. **<<** Moves the currently selected field one position the left.

	>> Moves the currently selected field one position the right. To change the name of a field, click the header (in this example, **Field1**), and type the new value.  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\MapForce2025\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 335.



Fixed Length Field Settings

Fill Character

● Space ○ Period ○ Custom: □

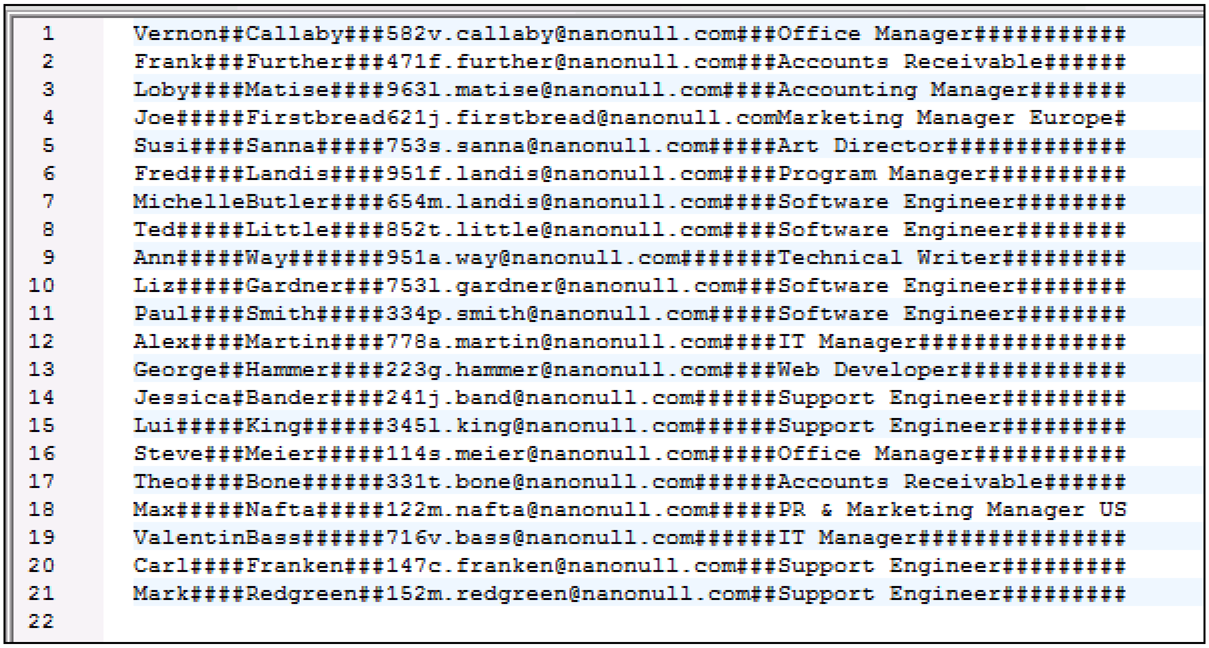
□ Assume record delimiters present

☑ 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 Manager
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##

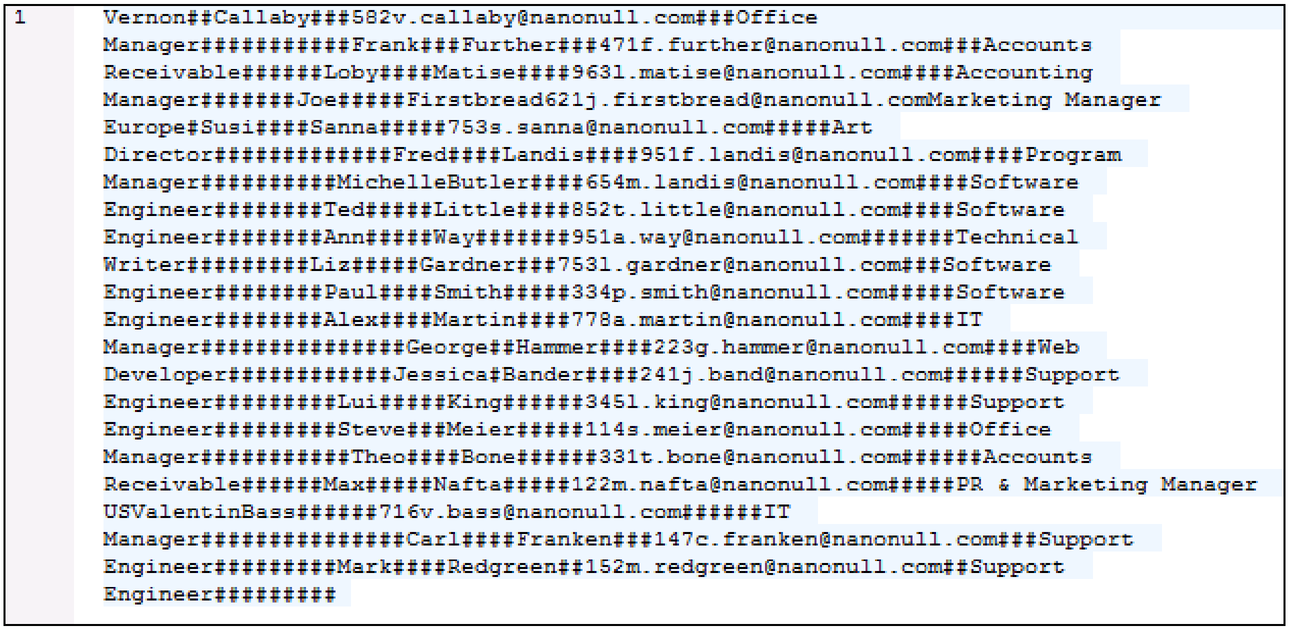
If you would need to write data from this file to a destination file which uses the same structure, then enabling **Assume record delimiters present** creates a new record after every 71 characters.



```
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####9631.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###7531.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######3451.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

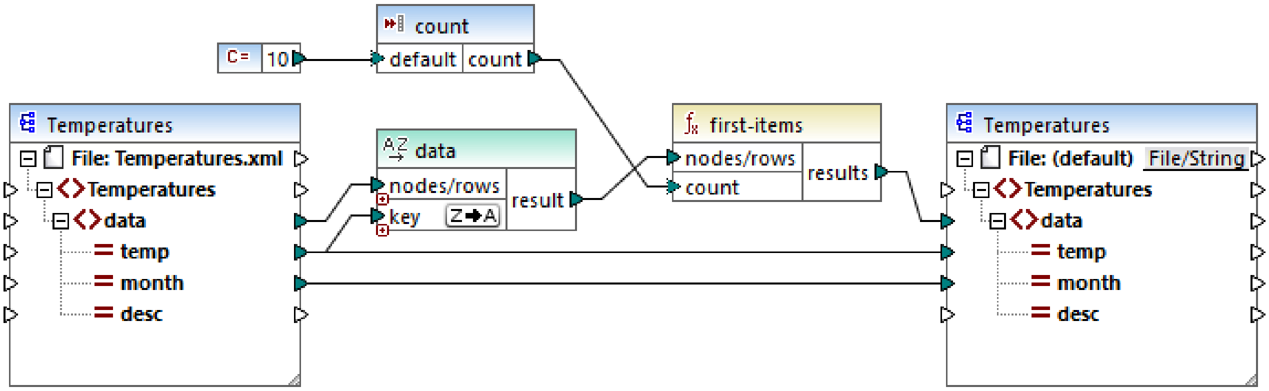
5 Transformation Components

This section describes transformation components that can be used to transform data or to store data temporarily for further processing. The list of transformation components is given below:

- Simple input 346
- Simple output 356
- Variables 360
- Join Components 373
- Sort Components 402
- Filters and Conditions 408
- Value-Maps 421
- Exceptions 432
- Functions 436

5.1 Simple Input

If you need to create a mapping that takes parameters as input, you can do so by adding a special component type called "simple input component". Simple input components always have a simple data type (for example, string, integer, and so on) instead of a structure of items and sequences. For example, in the mapping illustrated below, there is a simple input component **count**. Its role is to supply as parameter the maximum number of rows that should be retrieved from the source XML file (with value **10** as default). Importantly, the nodes supplied as input to the first-items 565 function are sorted with the help of a sort component, so the mapping outputs the highest *N* temperatures only, where *N* is the parameter's value.



FindHighestTemperatures.mfd

Another fairly common usage of simple input components is to supply a file name to the mapping. This is useful in mappings that read input files or write output files dynamically, see Processing Multiple Input or Output Files Dynamically 746.

You can use simple input components in any the following MapForce transformation languages:

- BUILT-IN (when you preview the mapping transformation directly in MapForce, from the **Preview** tab)
- BUILT-IN (when you run a compiled MapForce Server execution file)
- XSLT 1.0, XSLT 2.0, XSLT 3.0
- XQuery
- C++
- C#
- Java

In case of mappings executed with MapForce Server or by means of generated code, simple input components become command line parameters. In case of mappings generated as XSLT transformations, simple input components correspond to stylesheet parameters in the generated XSLT file.

You can create each simple input component (or parameter) as optional or mandatory, see Adding Simple Input Components 347. If necessary, you can also create default values for the mapping input parameters, see Creating a Default Input Value 349. This enables you to safely run the mapping even if you do not explicitly supply a parameter value at mapping execution time. For an example, see Example: Using File Names as Mapping Parameters 350.

Input parameters added on the main mapping area should not be confused with input parameters in user-defined functions 457. There are some similarities and differences between the two, as follows.

Input parameters on the mapping	Input parameters of user-defined functions
Added from **Function | Insert Input** menu.	Added from **Function | Insert Input** menu.
Can have simple data types (string, integer, and so on).	Can have simple as well as complex data types.
Applicable to the entire mapping.	Applicable only in the context of the function in which they were defined.

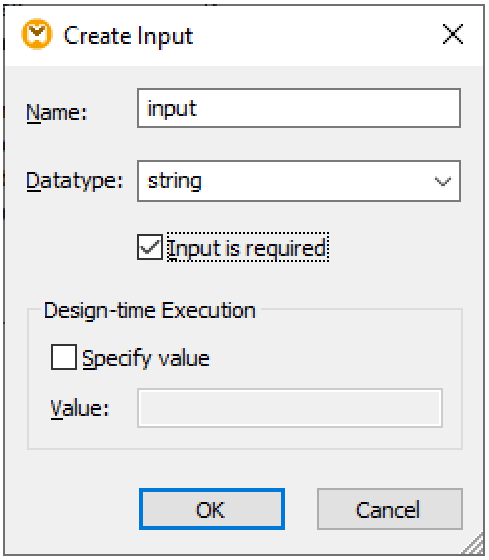
When you create a reversed mapping (using the menu command **Tools | Create Reversed Mapping**), a simple input component becomes a simple output component.

5.1.1 Adding Simple Input Components

To add a simple input to the mapping:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. Do one of the following:

   - On the **Function** menu, click **Insert Input**.
   - On the **Insert** menu, click **Insert Input**.
   - Click the **Insert Input** toolbar button.



Create Input

Name: input

Datatype: string

☑ Input is required

Design-time Execution

☐ Specify value

Value:

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 348.

Note: The parameter name can contain only letters, digits, and underscores; no other characters are allowed. This makes it possible for a mapping to work across all code generation languages.

4. Click **OK**.

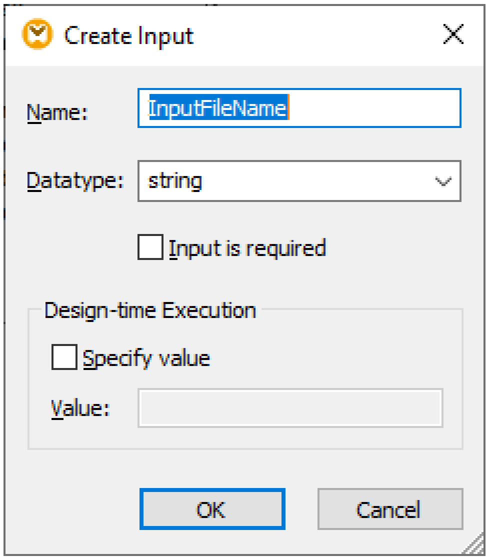
You can change later any of the settings defined here (see Simple Input Component Settings[348]).

5.1.2 Simple Input Component Settings

You can define the settings applicable to a simple input component when adding it to the mapping area. You can also change the settings at a later time, from the Edit Input dialog box.

To open the Edit Input dialog box, do one of the following:

- Select the component, and, on the **Component** menu, click **Properties**.
- Double-click the component.
- Right-click the component, and then click **Properties**.



Name: InputFileName

Datatype: string

□ Input is required

Design-time Execution

□ Specify value

Value:

Edit Input dialog box

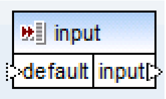
The available settings are as follows.

Name	Enter a descriptive name for the input parameter corresponding to this component. At mapping execution time, the value entered in this text box becomes the name of the parameter supplied to the mapping; therefore, no spaces or special characters are allowed.
Datatype	By default, any input parameter is treated as string data type. If the parameter should have a different data type, select the respective value from the list. When the mapping is executed, MapForce casts the input parameter to the data type selected here.

Input is required	When enabled, this setting makes the input parameter mandatory (that is, the mapping cannot be executed unless you supply a parameter value). Clear this check box if you want to specify a default value for the input parameter (see Creating a Default Input Value[349]).
Specify value	This setting is applicable only if you execute the mapping during design time, by clicking the **Preview** tab. It allows you to enter directly in the component the value to use as mapping input.
Value	This setting is applicable only if you execute the mapping during design time, by clicking the **Preview** tab. To enter a value to be used by MapForce as mapping input, select the **Specify Value** check box, and then type the required value. **Note:** If you click the **Specify value** check box and enter a value in the adjacent box, the entered value takes precedence over the default value when you preview the mapping (that is, at design-time execution). However, the design-time value has no effect in the generated XSLT, XQuery, or program code, in execution by MapForce Server, or deployment to FlowForce Server.

5.1.3 Creating a Default Input Value

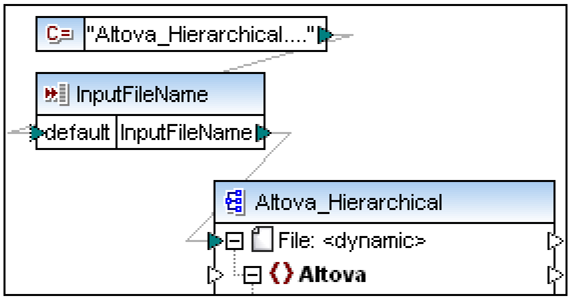
After you add an Input component to the mapping area, notice the **default** item to the left of the component.



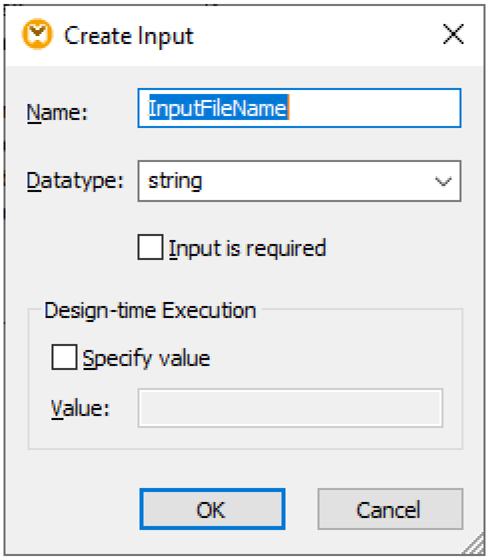
Simple input component

The **default** item enables you to connect an optional default value to this input component, as follows:

1. Add a constant component (on the **Insert** menu, click **Constant**), and then connect it to the **default** item of the input component.



2. Double-click the input component and clear the **Input is required** check box. When you create a default input value, this setting is not meaningful and causes mapping validation warnings.



Create Input

Name: InputFileName

Datatype: string

□ Input is required

Design-time Execution

□ Specify value

Value:

3. Click **OK**.

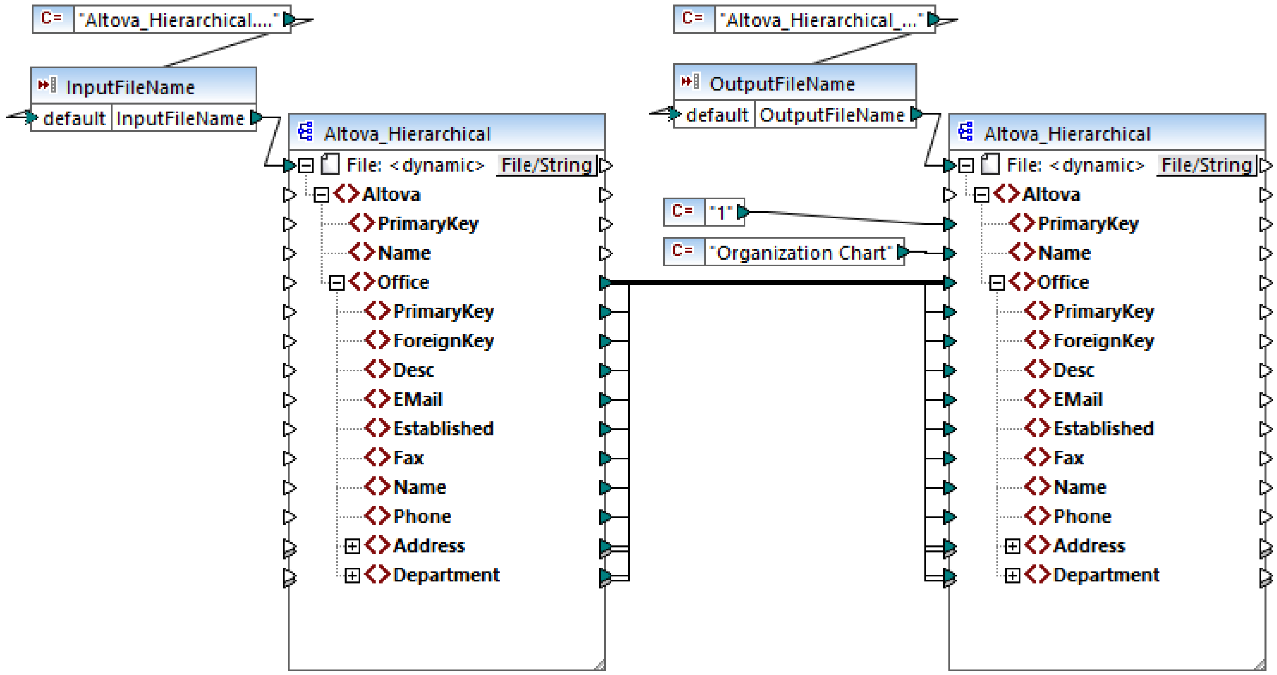
Note: If you click the **Specify value** check box and enter a value in the adjacent box, the entered value takes precedence over the default value when you preview the mapping (that is, at design-time execution). However, the design-time value has no effect in the generated XSLT, XQuery, or program code, in execution by MapForce Server, or deployment to FlowForce Server.

5.1.4 Example: Using File Names as Mapping Parameters

This example walks you through the steps required to execute a mapping that takes input parameters at runtime. The mapping design file used in this example is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\FileNamesAsParameters.mfd**.

This mapping reads data from a source XML file and writes it to a target XML file. The data is written to the target file almost unchanged; only the attributes **PrimaryKey** and **Name** are populated with some constant values from the mapping. The main goal of the mapping is to enable the caller to specify the name of the input file and the name of the output file, as mapping parameters, at mapping runtime.

To achieve this, the mapping has two input components: **InputFileName** and **OutputFileName**. These supply the input file name (and the output file name, respectively) of the source and target XML file. For this reason, they are connected to the **File: <dynamic>** item. You can switch a component to this mode by clicking the **File/String** (File/String) button, and selecting **Use Dynamic File Names Supplied by Mapping**.



FileNamesAsParameters.mfd (MapForce Enterprise Edition)

If you double-click the title bar of either of the **InputFileName** and **OutputFileName** components, you can view or edit their properties. For example, you can specify the data type of the input parameter or change the input parameter name, as described in Simple Input Component Settings 348. In this example, the input and output parameters are configured as follows:

- The **InputFileName** parameter is of type "string" and it has a default value supplied by a constant defined in the same mapping. The constant is of type "string" and its value is "Altova_Hierarchical.xml". Therefore, when this mapping runs, it will attempt to read data from a file called "Altova_Hierarchical.xml", assuming that you do not supply some other value as parameter.
- The **OutputFileName** parameter is of type "string" and it also has a default value supplied by a constant defined in the same mapping. The constant is of type "string" and its value is "Altova_Hierarchical_output.xml". Therefore, the mapping will create an XML output file called "Altova_Hierarchical_output.xml" when it runs, assuming that you do not supply some other value as parameter.

The following sections illustrate how to run the mapping and supply parameters in the following transformation languages:

- XSLT 2.0 352, using RaptorXML Server
- Built-in (MapForce Server Execution File) 352, using MapForce Server
- Java 353
- C# 354
- C++ 354

XSLT 2.0

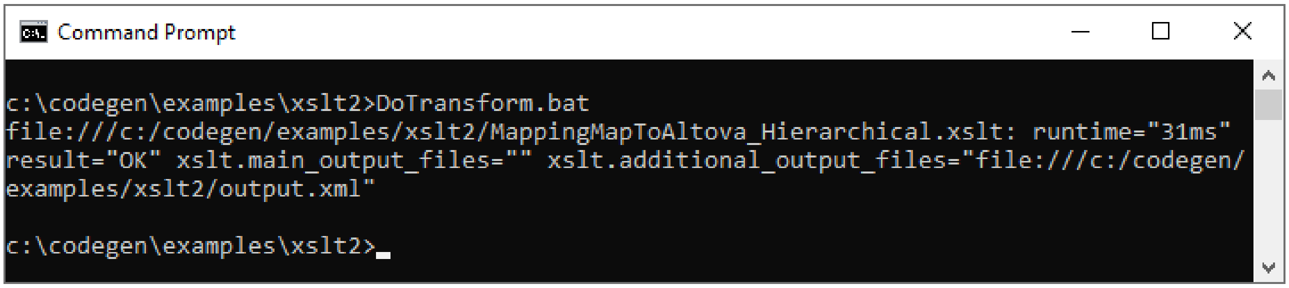
If you generate code in XSLT 1.0, XSLT 2.0, or XSLT 3.0, a **DoTransform.bat** batch file is generated in the chosen target directory, in addition to the XSLT file. The **DoTransform.bat** lets you execute the mapping with RaptorXML Server, see Automation with RaptorXML Server 792.

To use a different input (or output) file, edit the **DoTransform.bat** file to include the required parameters, as follows:

1. First, generate the XSLT code. For example, to generate XSLT 2.0, select the menu command **File | Generate Code In | XSLT 2.0**.
2. Copy the **Altova_Hierarchical.xml** file from **<Documents>\Altova\MapForce2025\MapForceExamples\** to the directory where you generated the XSLT 2.0 code (in this example, **c:\codegen\examples\xslt2**). As stated previously, the mapping will attempt to read this file if you do not supply a custom value to the **InputFileName** parameter.
3. Edit **DoTransform.bat** to include the custom input parameter either before or after `%*`. Note that the parameter value is enclosed with single quotes. The available input parameters are listed in the **rem** (Remark) section. Let's suppose that you would like to generate an output file called **output.xml**. To achieve this, change the **DoTransform.bat** file as follows:

```
@echo off

RaptorXML xslt --xslt-version=2
  --input="MappingMapToAltova_Hierarchical.xslt"
  --param=OutputFileName:'output.xml' %* "MappingMapToAltova_Hierarchical.xslt"
rem --param=InputFileName:
rem --param=OutputFileName:
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

When you run the **DoTransform.bat** file, RaptorXML Server completes the transformation using **Altova_Hierarchical.xml** as input. If you followed the steps above, the name of the generated output file will be **output.xml**.



```
c:\codegen\examples\xslt2>DoTransform.bat
file:///c:/codegen/examples/xslt2/MappingMapToAltova_Hierarchical.xslt: runtime="31ms"
result="OK" xslt.main_output_files="" xslt.additional_output_files="file:///c:/codegen/
examples/xslt2/output.xml"

c:\codegen\examples\xslt2>
```

MapForce Server Execution File

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

1. If you haven't done that already, open the **FileNamesAsParameters.mfd** example from the **<Documents>\Altova\MapForce2025\MapForceExamples\** directory.

2. On the **File** menu, click **Compile to MapForce Server Execution File**, see also Compiling Mappings to MapForce Server Execution Files 799. When prompted, save the .mfx execution file to a directory on your computer (in this example, **c:\codegen\examples\mfx**).
3. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2025\MapForceExamples\** directory to the directory where you saved the .mfx file.
4. Run MapForce Server with the following command:

```
MapForceServer.exe run "C:\codegen\examples\mfx\FileNamesAsParameters.mfx"
  -p=InputFileName:"C:\codegen\examples\mfx\Altova_Hierarchical.xml"
  -p=OutputFileName:"C:\codegen\examples\mfx\OutputFile.xml"
```

In the MapForce Server command above, `-p=InputFileName` and `-p=OutputFileName` are the input parameters to the mapping. You can use any file name as the value of **-OutputFileName**. However, the file name supplied in **-InputFileName** parameter must exist as a physical file; otherwise, the mapping will fail.

Note: If you see the message "MapForceServer.exe is not recognized as an internal or external command, operable program, or batch file", change the current directory to the one where the MapForce Server executable is installed. To avoid changing the path every time when you run a mapping, add to your operating system's PATH environment variable the path of the directory where the MapForce Server executable is installed (for example, **C:\Program Files (x86)\Altova\MapForceServer2025\bin**).

With MapForce Server, running a mapping is also possible by calling the MapForce Server API (which is invokable from languages such as C++, C#, or Java). For further information about this scenario, refer to the MapForce Server documentation (https://www.altova.com/documentation).

Java

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

1. If you haven't done that already, open the **FileNamesAsParameters.mfd** example from the **<Documents>\Altova\MapForce2025\MapForceExamples\** directory.
2. On the **File** menu, click **Generate Code In | Java**.
3. Compile the Java code into an executable JAR file. For an example of how to do this in Eclipse, see Example: Generate and Run Java Code.
4. Copy the **Altova_Hierarchical.xml** file from **<Documents>\Altova\MapForce2025\MapForceExamples\** to the directory where the .jar file is. As stated previously, the mapping will attempt to read this file if you do not supply a custom value to the **InputFileName** parameter.
5. Run the Java application with the following command:

```
java -jar Mapping.jar /OutputFileName "output.xml"
```

In the command above, the input parameter `/OutputFileName` supplies the name of the output file to be generated.

Note: If you use wildcards when passing parameters to .jar files, enclose the wildcard parameters within quotes, for example:

```
java -jar Mapping.jar /InputFileName "altova-*.xml"
```

C#

To supply a custom input parameter to a C# command line application generated by MapForce:

1. If you haven't done that already, open the **FileNamesAsParameters.mfd** example from the **<Documents>\Altova\MapForce2025\MapForceExamples\** directory.
2. On the **File** menu, click **Generate Code In | C#**, and select a target directory (**C:\codegen\examples\cs**, in this example).
3. Open the solution in Visual Studio and build it (**Ctrl + Shift + B**).
4. Copy the **Altova_Hierarchical.xml** file from **<Documents>\Altova\MapForce2025\MapForceExamples\** to the directory where **Mapping.exe** was generated (in this example, **C:\codegen\examples\cs\Mapping\bin\Debug**). As stated previously, the mapping will attempt to read this file if you do not supply a custom value to the **InputFileName** parameter.
5. Open a Command Prompt window and change to the directory where **Mapping.exe** is.

```
cd C:\codegen\examples\cs\Mapping\bin\Debug
```

6. Run the application with the following command:

```
Mapping.exe /OutputFileName output.xml
```

In the command above, the input parameter `/OutputFileName` supplies the name of the output file to be generated.

C++

To supply a custom input parameter to a C++ command line application generated by MapForce:

1. If you haven't done that already, open the **FileNamesAsParameters.mfd** example from the **<Documents>\Altova\MapForce2025\MapForceExamples\** directory.
2. On the **File** menu, click **Generate Code In | C++**, and select a target directory (**C:\codegen\examples\cpp**, in this example).
3. Open the solution in Visual Studio and build it (**Ctrl + Shift + B**).
4. Copy the **Altova_Hierarchical.xml** file from **<Documents>\Altova\MapForce2025\MapForceExamples\** to the directory where **Mapping.exe** was generated (in this example, **C:\codegen\examples\cpp\Mapping\Debug**). As stated previously, the mapping will attempt to read this file if you do not supply a custom value to the **InputFileName** parameter.
5. Open a Command Prompt window and change to the directory where **Mapping.exe** is.

```
cd C:\codegen\examples\cpp\Mapping\Debug
```

6. Run the application with the following command:

```
Mapping.exe /OutputFileName output.xml
```

In the command above, the input parameter `/OutputFileName` supplies the name of the output file to be generated.

5.2 Simple Output

An output component (or "simple output") is a MapForce component which enables you to return a string value from the mapping. Output components represent one possible type of target components[32], but should not be confused with the latter. Use a simple output component when you need to return a string value from the mapping. On the mapping area, simple output components play the role of a target component which has a string data type instead of a structure of items and sequences. Consequently, you can create a simple output component instead of (or in addition to) a file-based target component. For example, you can use a simple output component to quickly test and preview the output of a function (see Example: Testing Function Output[358]). 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[457]). 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** pane 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** pane 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[799]), the mapping output is returned in the standard output stream (stdout), so you can view it or redirect to a file. For example, assuming that the name of the MapForce server execution file is **MyMapping.mfx**, use the following syntax to redirect the mapping output to the **output.txt** file and any errors to the **log.txt** file: `MapForceServer.exe run MyMapping.mfx >output.txt 2>log.txt`
XSLT 1.0, XSLT 2.0, XSLT 3.0	In the generated XSLT files, a simple output component defined in the mapping becomes the output of the XSLT transformation.

Language	How it works
	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 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.

5.2.1 Adding Simple Output Components

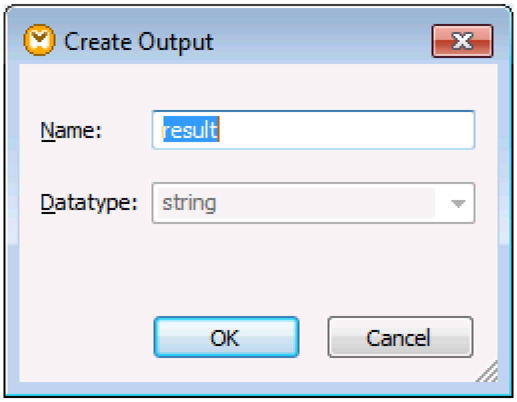
To add an output component to the mapping area:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. Do one of the following:

   a. On the **Function** menu, click **Insert Output**.

   b. Click the **Insert output** toolbar button.

3. Enter a name for the component.
4. Click **OK**.



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**.

5.2.2 Example: Previewing Function Output

This example illustrates how to preview the output returned by MapForce functions with the help of simple output components. You will make the most of this example if you already have a basic understanding of functions in general, and of MapForce functions in particular. If you are new to MapForce functions, you may want to refer to Using Functions 436 before continuing.

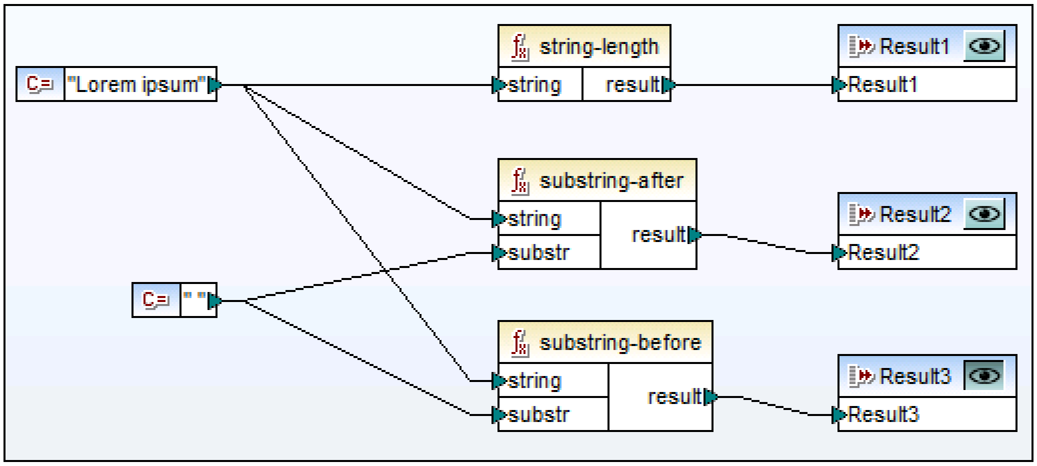
Our aim is to add a number of functions to the mapping area, and learn how to preview their output with the help of simple output components. In particular, the example uses a few simple functions available in the core library. Here is a summary of their usage:

`string-length` 595	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` 596	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` 597	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** pane on the Mapping window.

5.3 Variables

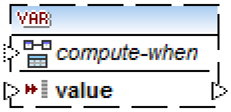
A variable is a special type of components used to store an intermediate mapping result for further processing. Variables can be of simple type (e.g., string, integer, boolean, etc) and complex type (a tree structure). See the examples of both types in the subtopics below.

One of the most important aspects of variables is that they are sequences and can be used to create sequences. The term sequence means a list of zero or more items. This makes it possible for a variable to process multiple items for the duration of the mapping lifetime. For more information, see also Mapping Rules and Strategies 77. However, it is also possible to assign a value to a variable once and keep this value the same for the rest of the mapping. For details, see Changing the Context and Scope of Variables 366.

For details about how to add a variable to a mapping, see Add a Variable 362.

Simple variables

A simple variable is built to represent atomic types such as strings, numbers, and booleans (*see screenshot below*).



Complex variables

A complex variable has a tree structure. The structures on which a complex variable can be based are summarized in the list below.

MapForce Basic Edition:

- XML Schema Structure

MapForce Professional Edition:

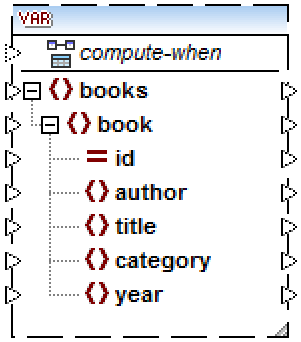
- XML Schema Structure
- Database Structure

MapForce Enterprise Edition:

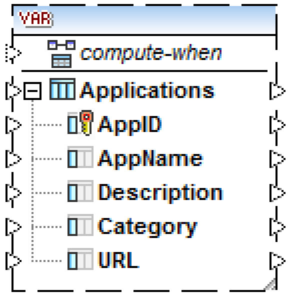
- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure
- JSON Schema Structure

Example 1: Variable based on XML Schema

You can create a complex variable by supplying an XML schema which defines the structure of the variable (*see screenshot below*). If the schema defines any elements globally, you can choose which one should become the root node of the variable structure. Note that a variable does not have an associated instance XML file. The data of the variable is computed at mapping runtime.



Example 2: Variable based on a database (MapForce Professional and Enterprise editions)
If you choose a database structure for your variable (*see screenshot below*), you can choose a specific database table as the root item for the variable structure. MapForce allows you to create DB-based variables with a tree of related tables.



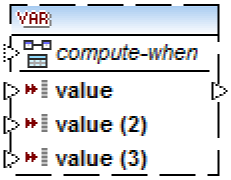
The tree of related tables represents an in-memory structure that has no connection to the database at runtime. The variable only describes the structure but does not do anything with the data. This implies the following:

- No validation based on the data type
- No possibility to use the SQL WHERE filter
- No table actions in parameters or variables
- No automatic handling of foreign keys
- No primary/foreign key checks

Compute-when
In both examples above, each variable has an item called `compute-when`. Connecting this item is optional: This enables you to control how the variable value should be computed in the mapping. For more information, see Changing the Context and Scope of Variables 366.

Variables with duplicated inputs

When necessary, items of a variable structure can be duplicated to accept data from more than one source connection. This is similar to duplicating inputs 42 in standard components. The screenshot below illustrates a simple variable with duplicated inputs.



Chained mappings vs. variables

Variables can be compared to intermediate components of a chained mapping 115. However, variables are more flexible and convenient if you do not need to produce intermediary files at each stage of the mapping. The table below outlines differences between variables and chained mappings.

Chained mappings	Variables
Chained mappings involve two independent steps. For example, a mapping has three components, namely A, B, and C. Step 1: mapping data A to B. Step 2: mapping data from B to C.	You can control when and how often the variable value is computed when the mapping is carried out. For details, see Changing the Context and Scope of Variables 366.
When the mapping is carried out, intermediate results are stored externally in files.	When the mapping is carried out, intermediate results are stored internally. No external files containing the results of a variable are produced.
The intermediate result can be previewed using the preview button.	The result of a variable cannot be previewed, since it is computed at mapping runtime.

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

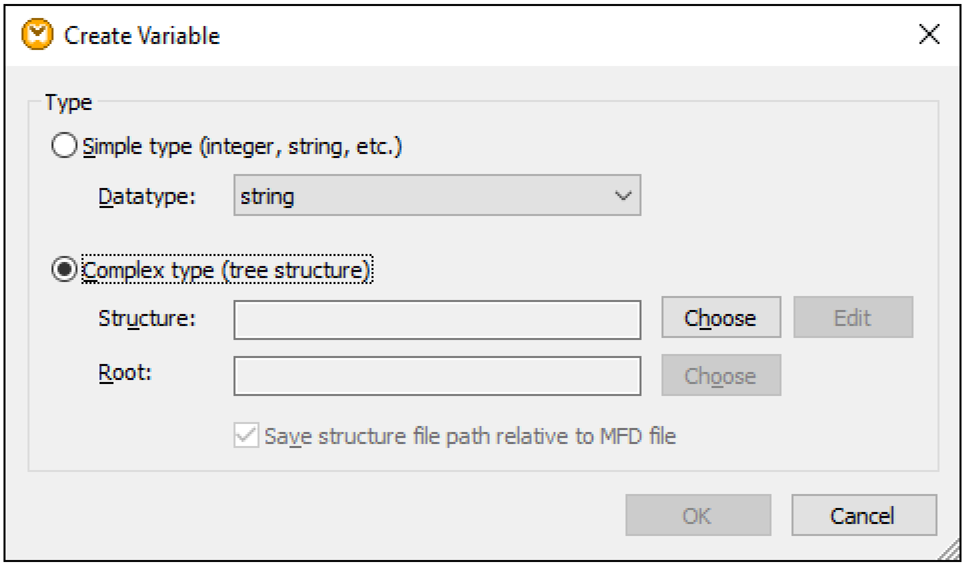
5.3.1 Add a Variable

This topic explains how to add a variable to a mapping. The first option is to add a variable via the menu or toolbar command. The second option allows you to add a variable via the context menu.

Option 1: via the menu or toolbar command

This option enables you to add a variable via the menu or toolbar command. Take the steps below:

1. Go to the **Insert** menu and click **Variable**. Alternatively, click the VAR toolbar button (**Variable**).



Create Variable

Type

○ Simple type (integer, string, etc.)

Datatype: string

● Complex type (tree structure)

Structure:

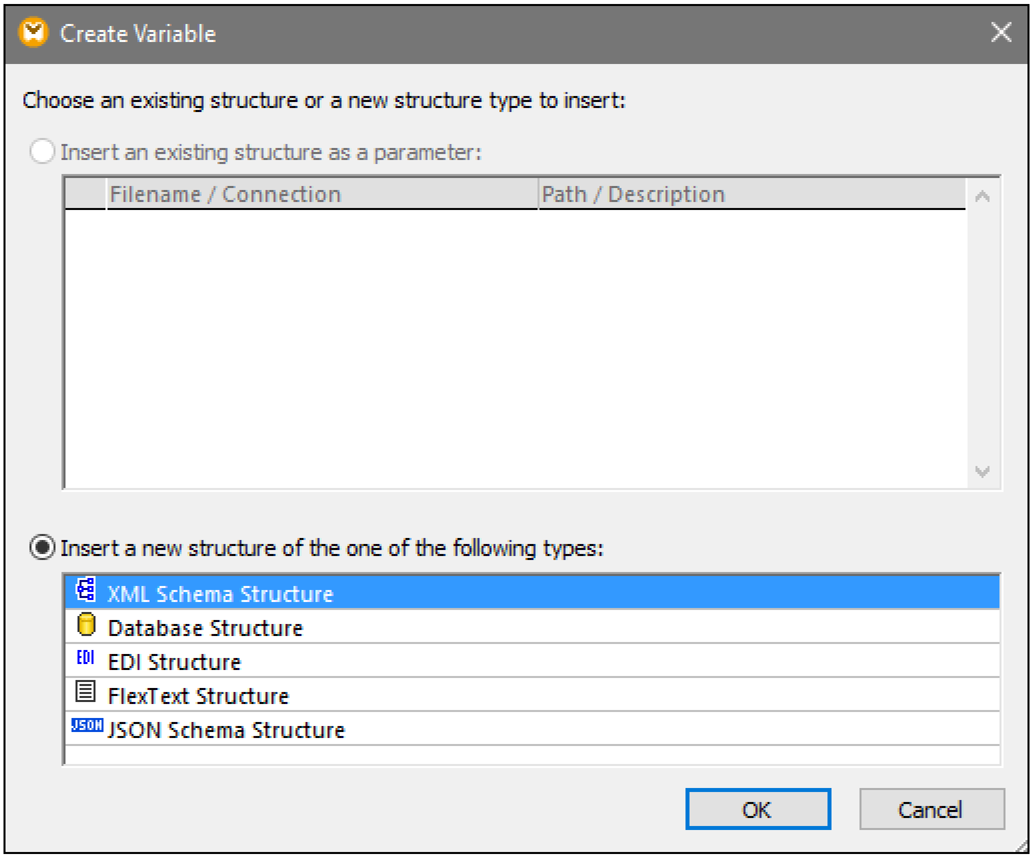
Root:

☑ Save structure file path relative to MFD file

2. Select the type of variable you want to insert (simple or complex type).

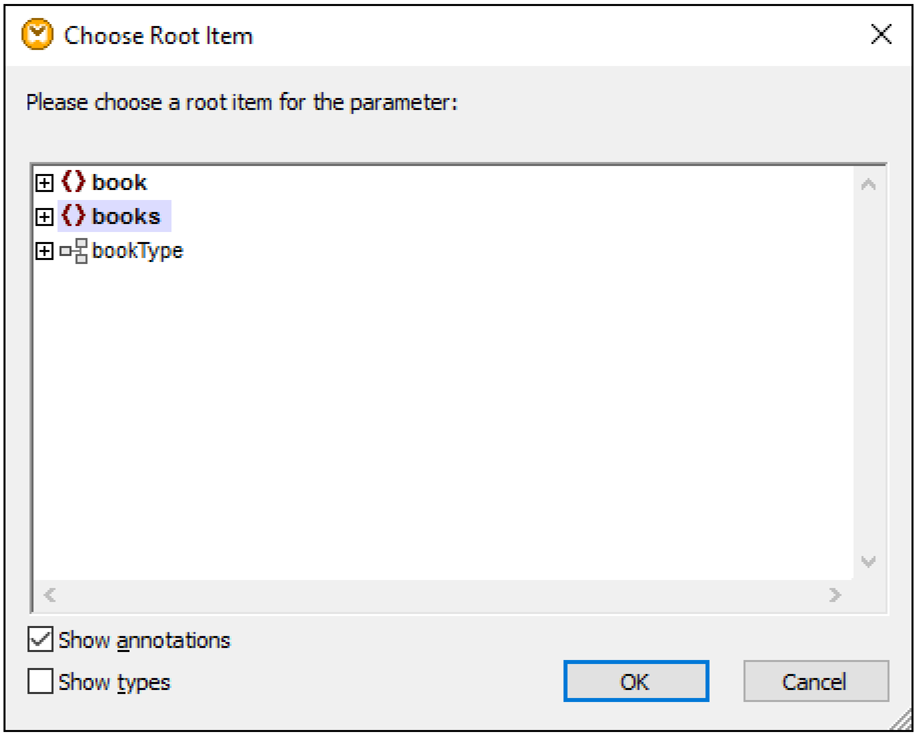
If you select **Complex type**, there are a few additional steps:

3. Click **Choose** to select the source which should provide the structure of the variable 360. The structures illustrated in the screenshot below only apply to MapForce Enterprise Edition. See the list of structures relevant to other MapForce editions in the previous topic 360.



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
JSON Schema Structure
OK
Cancel

4. When prompted, specify the root item of the structure of the variable. For example, in XML schemas, you can select any element or type from the selected source (*see screenshot below*).

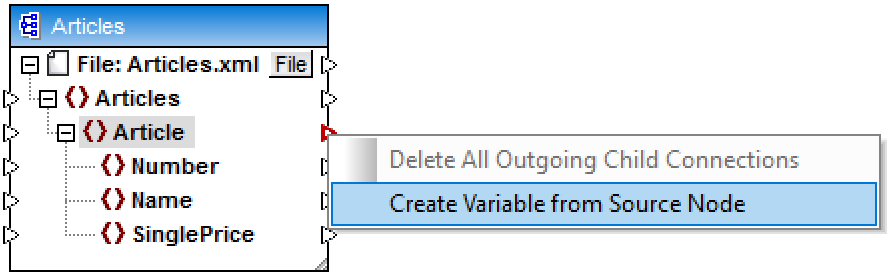


Option 2: via the context menu

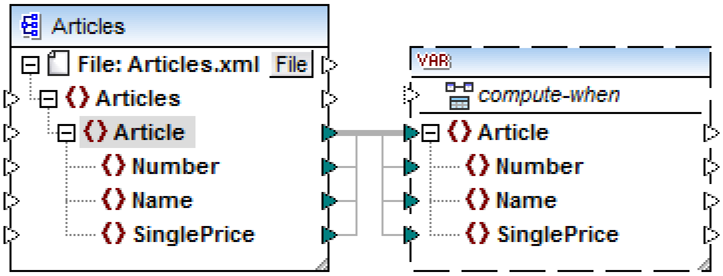
The second option allows you to create a variable using the context menu. The possible options are listed below.

Variable from a source node

To create a variable from a source node, right-click the output connector of a component (in this example, the output connector of the `<Article>` element) and select **Create Variable from Source Node** (*see screenshot below*).


Articles
File: Articles.xml
File
Articles
Article
Number
Name
SinglePrice
Delete All Outgoing Child Connections
Create Variable from Source Node

This creates a complex variable with the source schema of the `Articles` component. All the items are automatically connected with a copy-all connection 56 (*see screenshot below*).

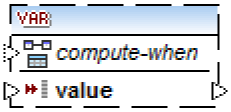


Variable from a target node
To create a variable from a target node, right-click the input connector of a target component and select **Create Variable for Target Node**. This creates a complex variable with the same schema as in the target. All the items are automatically connected with a copy-all connection.

Variable from a filter:
To create a variable using a filter, right-click the output connector of a filter component (`on-true/on-false`) and select **Create Variable from Source Node**. This creates a complex component with the source schema and automatically uses the item linked to the filter input as the root element of the intermediate component.

5.3.2 Scope and Context of Variables

Every variable has a `compute-when` input item (*see screenshot below*), which allows you to control the scope of the variable. This means that you can control when and how often the variable value is computed when the mapping is executed. You do not have to connect this input in many cases, but it can be essential to override the default context or to optimize the mapping performance.



The following terms are relevant to the discussion of the scope and context of variables: *subtree* and *variable value*. A subtree is a set of an item/node in a target component and all of its descendants: for example, a `<Person>` element with its `<FirstName>` and `<LastName>` child elements.

A variable value is the data that is available at the output side of the variable component.

- For simple variables, it is a sequence of atomic values that have the datatype specified in the component properties.
- For complex variables, it is a sequence of root nodes (of the type specified in the component properties), each one including all its descendant nodes.

The sequence of atomic values (or nodes) may contain one or even zero elements. This depends on what is connected to the input side of the variable, and to any parent items in the source and target components.

Compute-when is not connected (default)

If the `compute-when` input item is not connected to an output node of a source component, the variable value is computed *whenever it is first used in a target subtree* directly via a connector from the variable component to a node in the target component or indirectly via functions. The same variable value is also used for all target child nodes inside the subtree.

The actual variable value depends on any connections between parent items of the source and target components. This default behavior is the same as that of complex outputs of regular user-defined functions 460 and web service function calls. If the variable output is connected to multiple unrelated target nodes, the variable value is computed *separately for each of them*. This can produce different results in each case, because different parent connections influence the context in which the variable's value is evaluated.

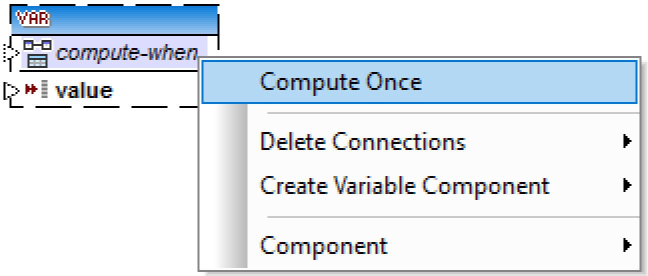
Compute-when is connected

By connecting an output connector of a source component to `compute-when`, the variable is computed *whenever that source item is first used in a target subtree*.

The variable actually acts as if it were a child item of the item connected to `compute-when`. This makes it possible to bind the variable to a specific source item. That is, at runtime the variable is re-evaluated whenever a new item is read from the sequence in the source component. This is related to the general rule governing connections in MapForce: For each source item, one target item is created. In this case, `compute-when` instructs MapForce to compute the variable value for each source item. For more information, see Mapping Rules and Strategies 77.

Compute-once

If necessary, you can choose to compute the variable value *once before each of the target components*, making the variable essentially a global constant for the rest of the mapping. To do this, right-click the `compute-when` item and select **Compute Once** from the context menu:

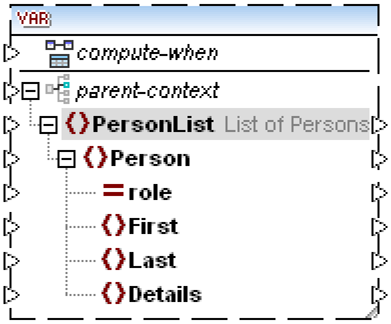


When you change the scope of a variable to `compute-when=once`, the input connector is removed from the `compute-when` item, since such a variable is only evaluated once. In a user-defined function, the `compute-when=once` variable is evaluated each time the function is called before the actual function result is evaluated.

Parent-context

The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., `min`, `max`, `avg`, `count`). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.

Adding a `parent-context` item may be necessary, for example, if your mapping uses multiple filters and you need an additional parent node to iterate over. For details, see Example: Changing the Parent Context 82. To add a parent-context to a variable, right-click the root node (in this example, `PersonList`) and select **Add Parent Context** from the context menu. This adds a new node, `parent-context`, to the existing hierarchy.

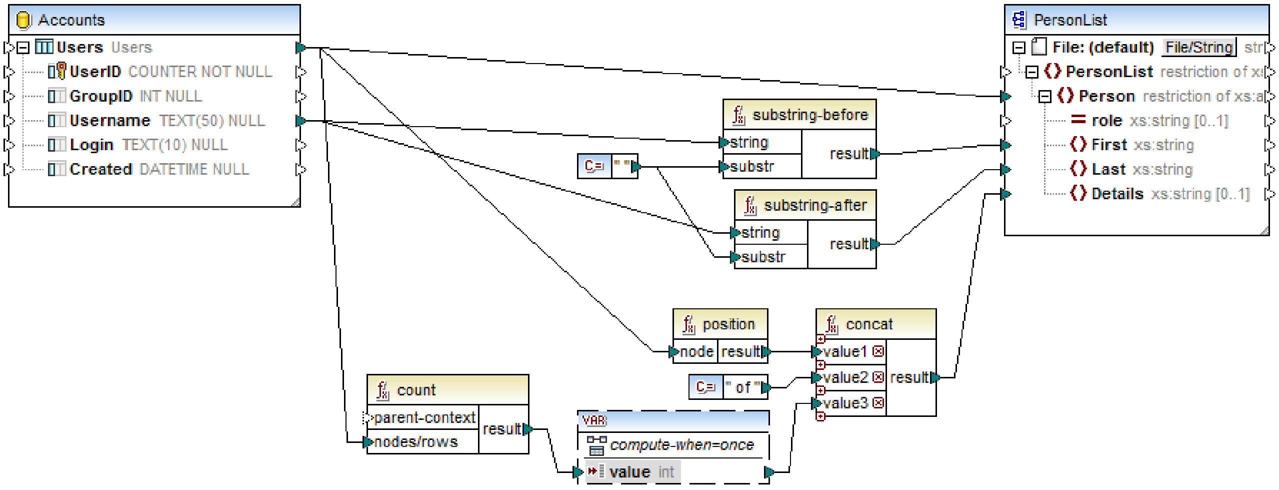


The parent context adds a virtual parent node to the hierarchy within the component. This allows you to iterate over an additional node in the same or in a different source component.

5.3.3 Example: Counting Database Table Rows

The mapping illustrated in this example is available as **DB_UserList.mfd** in the **<Documents>\Altova\MapForce2025\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.

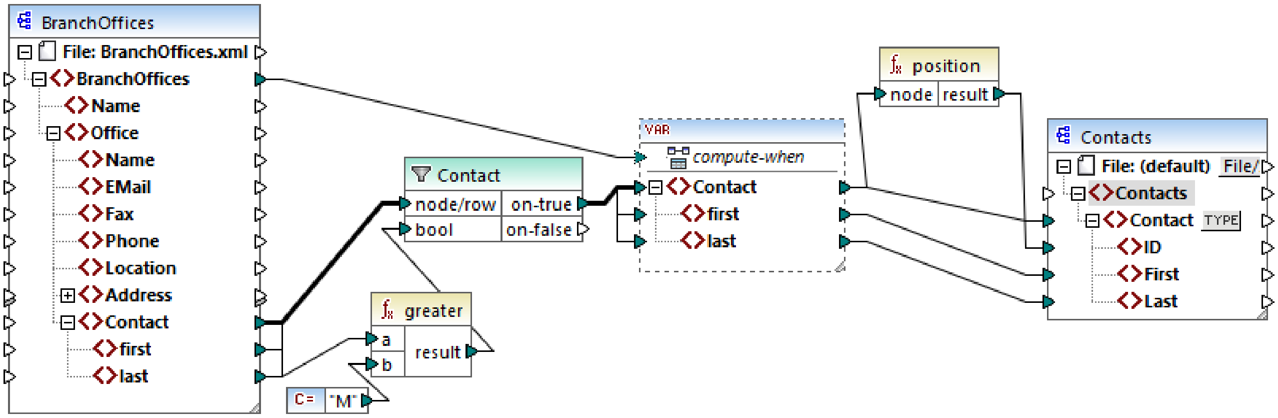
The value of the field "Username" is supplied to the **`substring-before`**[597] and **`substring-after`**[596] functions. These two functions extract the text before and after the space character (" "), respectively, which takes care of the second mapping goal.

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[366])

5.3.4 Example: Filtering and Numbering Nodes

The mapping illustrated in this example is available as **PositionInFilteredSequence.mfd** in the **<Documents>\Altova\MapForce2025\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 [408])
- A complex variable (see Adding Variables [362])
- The functions **greater** [545] and **position** [582] (see Add a Function to the Mapping [436])
- A constant (To add a constant, select the menu command **Insert | Constant**).

The variable uses the same schema as the source component. If you right-click the variable and select **Properties** from the context menu, notice that the node **BranchOffices/Office/Contact** is selected as root node for this variable structure.

First, data of the source component is passed on to the filter. The filter passes onwards to the variable only those records that meet the filter condition. Namely, the filter is configured to get only those `Contact` nodes where the last name is equal or greater than M. To achieve this, the function **greater** [545] 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.

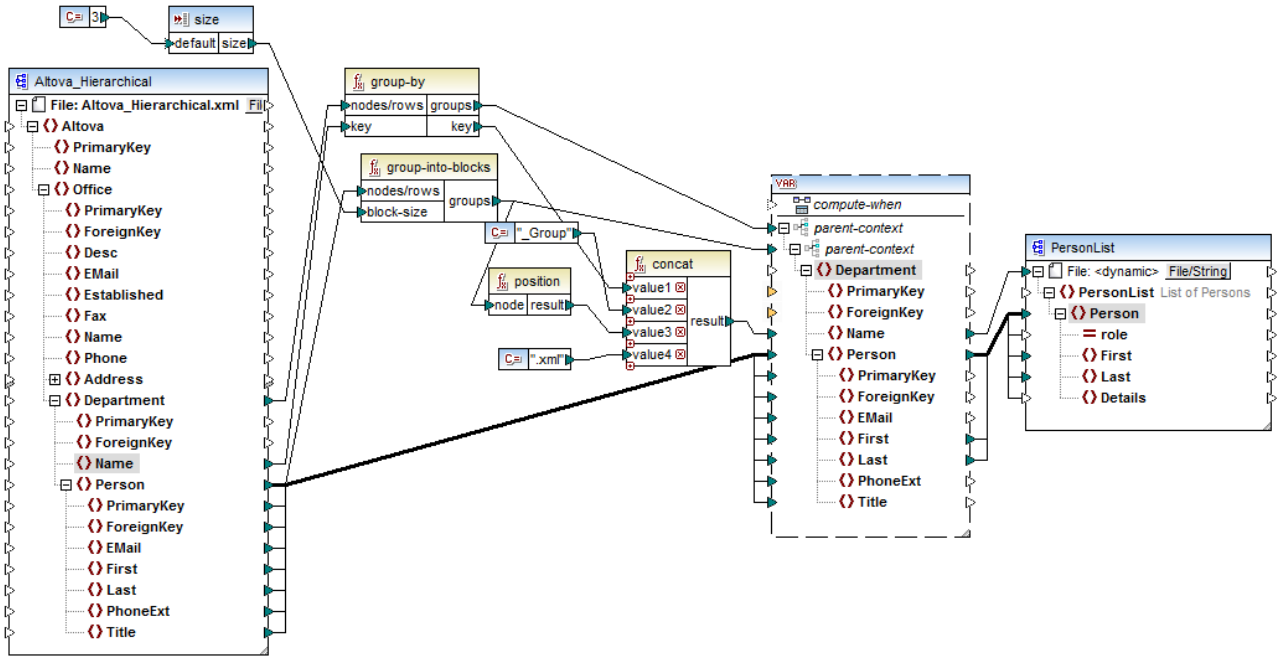
The **position** [582] functions returns, for each iteration of the variable, the number of the current sequence. Only eight contacts meet the filter condition; therefore, if you preview the mapping and look at the output, notice how IDs 1 through 8 were written to the `ID` element of the target component.

In case you were wondering why the variable was necessary at all, it is because of the requirement to number all records. Had we connected the filter result directly to the target component, there would have been no way to number each occurrence of `Contact`. The purpose of the variable in this mapping is, therefore, to store each instance of `Contact` temporarily on the mapping, so that it can be numbered before it is written to the target.

5.3.5 Example: Grouping and Subgrouping Records

The mapping illustrated in this example is available as **DividePersonsByDepartmentIntoGroups.mfd** in the **<Documents>\Altova\MapForce2025\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 366).

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`** 569 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`** 575 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 346.

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`** 592 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"

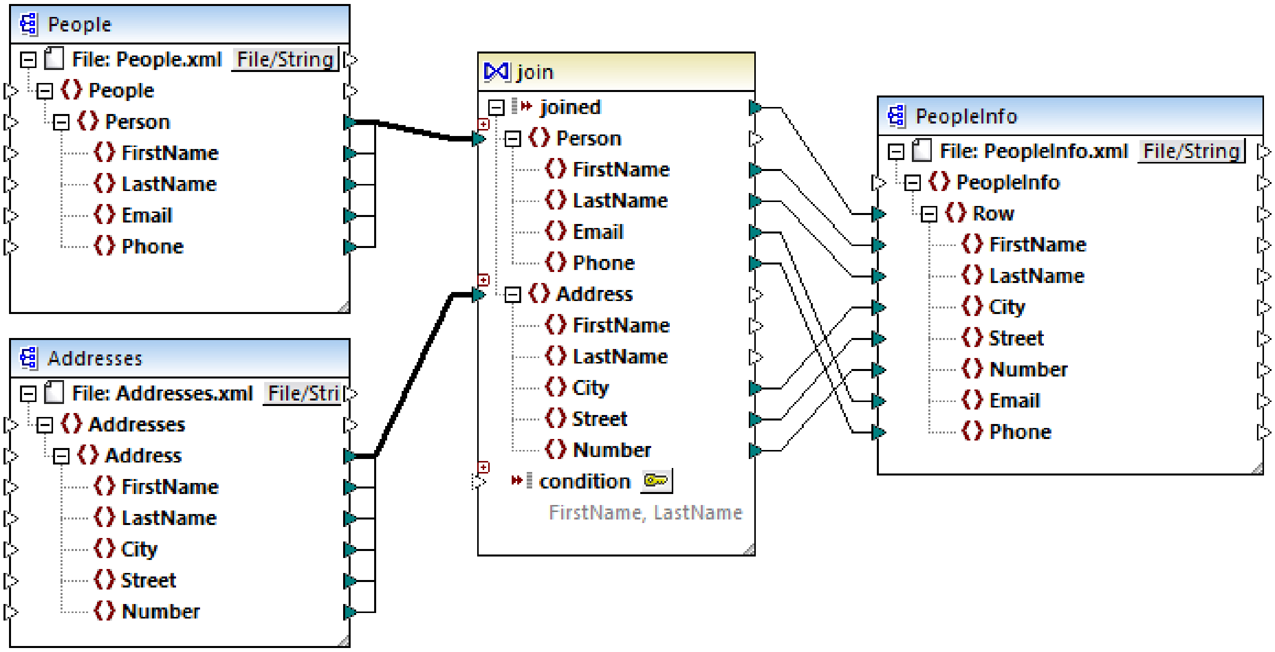
The result of this concatenation is stored in the `Name` item of the variable, and then supplied as a dynamic file name to the target component. This causes a new file name to be created for each received value. In this example, the variable computes eight groups in total, so eight output files are created when the mapping runs, as required. For more information about this technique, see Processing Multiple Input or Output Files Dynamically 746.

5.4 Join Components

Sometimes, you may need to combine data from two or more structures based on some condition (for example, if field A in the first structure has the same value as field B in the second structure). For such mapping requirements, a Join component can be used.

A Join component is a MapForce component which enables joining two or more structures on the mapping based on custom-defined conditions. It returns the association (joined set) of items that satisfy the condition. Joins are particularly useful to combine data from two structures which share a common field (such as an identity).

For example, on the mapping illustrated below, the middle component is a "Join" component. In this mapping, two XML structures (a list of people and a list of addresses) are being joined. The goal here is to get the full details of each person into a target XML file. The `FirstName` and `LastName` fields act as joining keys. Namely, if value of `FirstName` and `LastName` (under `Person`) is the same as that of `FirstName` and `LastName` (under `Address`), the address details belong to one and the same person and they become "joined". Any items from the joined structure can further be mapped to a subsequent target (in this case, an XML file). The join condition itself is defined in the properties of the Join component, by clicking the **Define Join Condition** ( ) button. This example is accompanied by a mapping sample and is explained in more detail in Example: Join XML Structures 379.



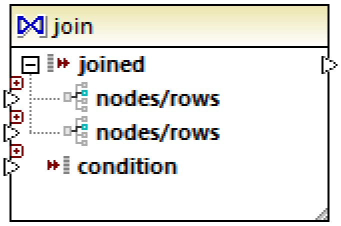
JoinPeopleInfo.mfd

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 56).

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 384.

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 378.



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 375.

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 64).
- Join components should not be connected to input parameters or results of inline user-defined functions. If such connections exist, validation errors will occur during mapping validation.
- When you connect eligible database components (such as tables or views) directly to a Join component, an **SQL mode** (SQL) button automatically appears at the top-right corner of the Join component. When enabled, this button provides special SQL features applicable to the join operation (see Joins in SQL Mode 386).
- 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 360 and Filters and Conditions 408, 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 384).

Adding a parent context

In some special cases, in order to achieve a specific mapping result, you can explicitly provide a mapping context (a so-called "parent context") for data connected to the Join component. To add a parent context, right-click the `joined` item of the Join component, and select **Add Parent Context** from the context menu. The Join component changes appearance to include an additional `parent-context` input where you can connect the required source item. For more information, see Example: Changing the Parent Context 82.

The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.

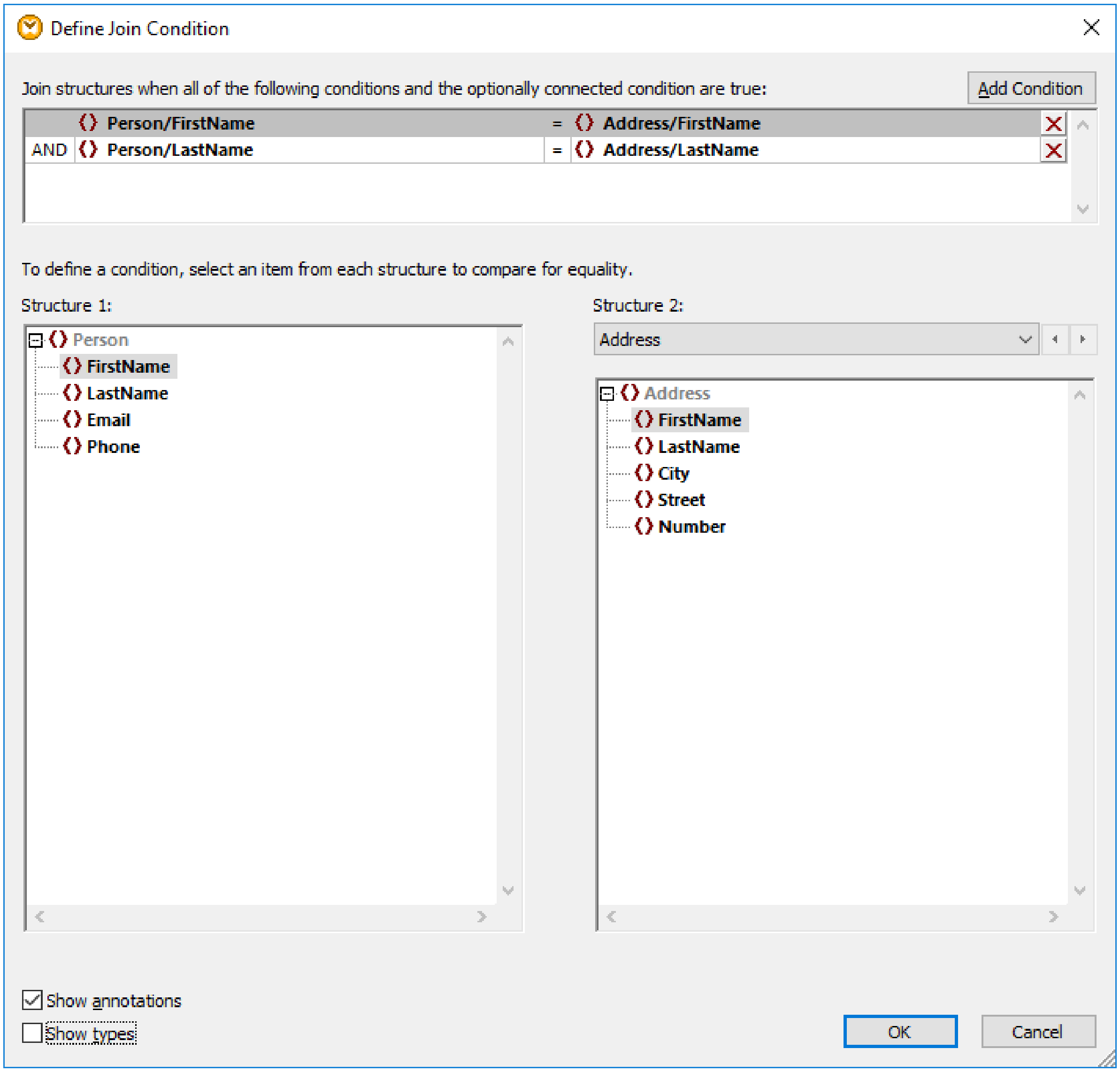
5.4.1 Adding Join Conditions

A join works by combining items of two or more structures according to a condition, so a join always requires at least one condition. There are several ways to add join conditions, as shown below.

Note: When database tables are joined in SQL mode, MapForce will create the join condition (or conditions) automatically, based on foreign key relationships detected between tables. For automatic join conditions to happen, the database tables must be in a child-parent relationship on the MapForce component (that is, one table must be "parent" or "child" of another one on the component), see Example: Join Tables in SQL Mode 389 .

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\MapForce2025\MapForceExamples\Tutorial\**. This mapping is discussed in more detail in Example: Join XML Structures 379.
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:

Structure 2: Address

☑ 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.

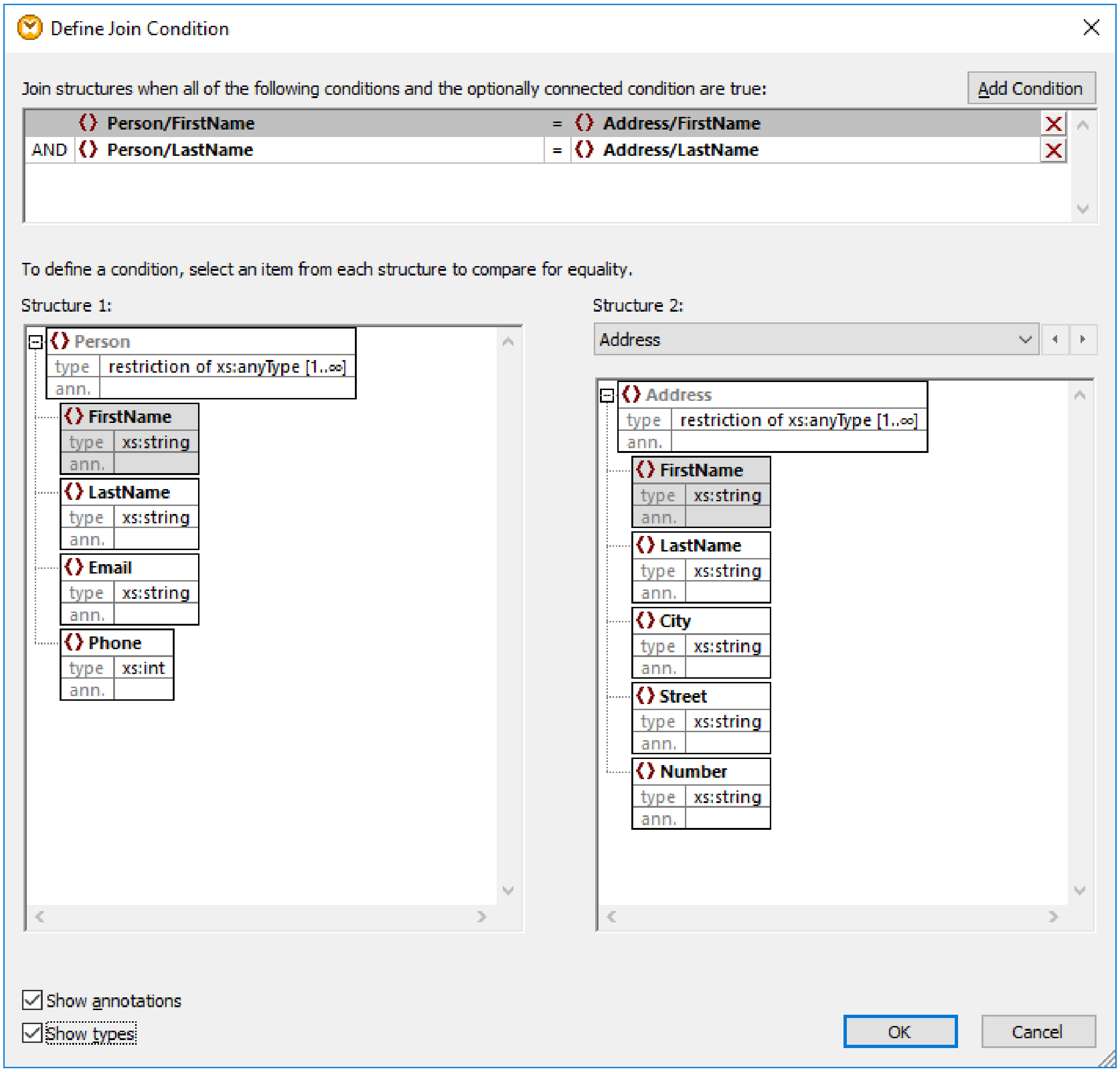
To remove a join condition, click the **Delete** ☒ button next to it.

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 397.

- 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:


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.

Structure 1:

Item	type	ann.
Person	restriction of xs:anyType [1..∞]	
FirstName	xs:string	
LastName	xs:string	
Email	xs:string	
Phone	xs:int	

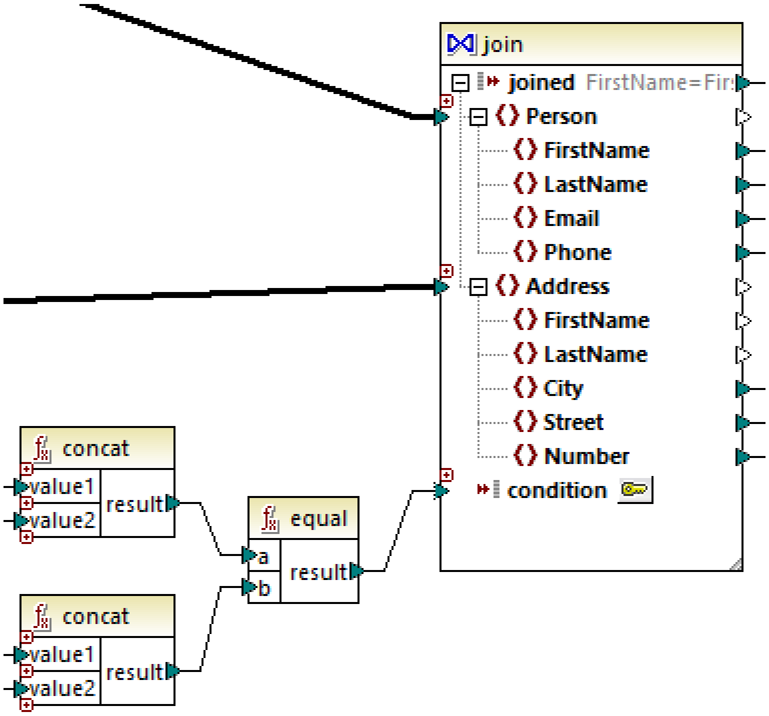
Structure 2: Address

Item	type	ann.
Address	restriction of xs:anyType [1..∞]	
FirstName	xs:string	
LastName	xs:string	
City	xs:string	
Street	xs:string	
Number	xs:string	

☑ 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.



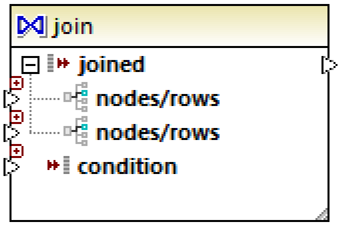
Note: If no condition is defined from the join component properties (Approach 1), the `condition` item of the join component must be connected (Approach 2).

Approach 3: Mixed approach

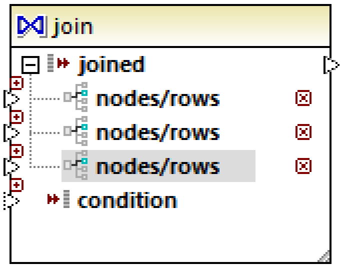
In the same mapping, it is possible to define some join conditions in component properties (Approach 1) and combine them with the one from the mapping (Approach 2). However, if you intend to join database tables in SQL mode, the conditions must be defined strictly using Approach 1, see also Joins in SQL Mode 386.

5.4.2 Joining Three or More Structures

When you add a Join component to the mapping using the menu command **Insert | Join**, it accepts two structures by default (that is, the component contains only two `nodes/rows` inputs).



If you need to join more than two structures, click the **Add input** (⊞) button and create as many `nodes/rows` as necessary. If you need to remove a `nodes/rows` input, click the **Delete input** (⊠) button. Note that a join requires at least two structures, so the ⊠ button is only available when more than two inputs exist.



When a join has multiple inputs, the join conditions must accordingly take into consideration each of the inputs that you want to be joined, see Adding Join Conditions 375. For a step-by-step example of how to join multiple database tables, see Example: Create CSV Report from Multiple Tables 397.

5.4.3 Example: Join XML Structures

This example shows you how to combine data from two XML structures conditionally, by using a join component. The example is accompanied by a mapping sample which is available at the following path: **<Documents>\Altova\MapForce2025\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

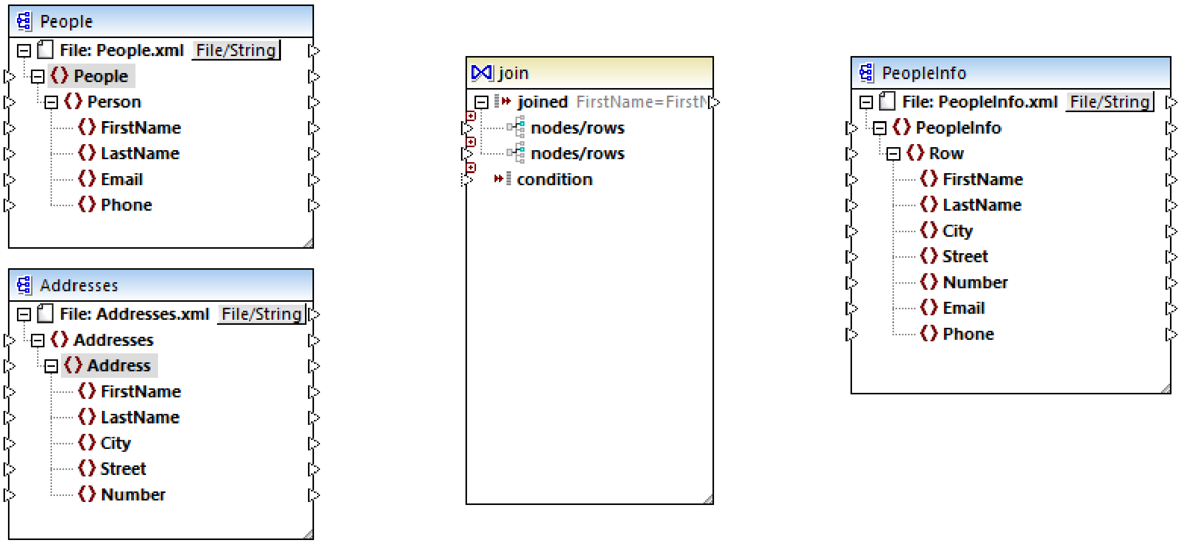
1. On the **Insert** menu, click **XML Schema/File**, and browse for the following source file: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\People.xml**.
2. Repeat the step above for **Addresses.xml** (the second source file).

Step 2: Add the target schema file to the mapping

- On the **Insert** menu, click **XML Schema/File**, and browse for **<Documents>\Altova\MapForce2025\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.

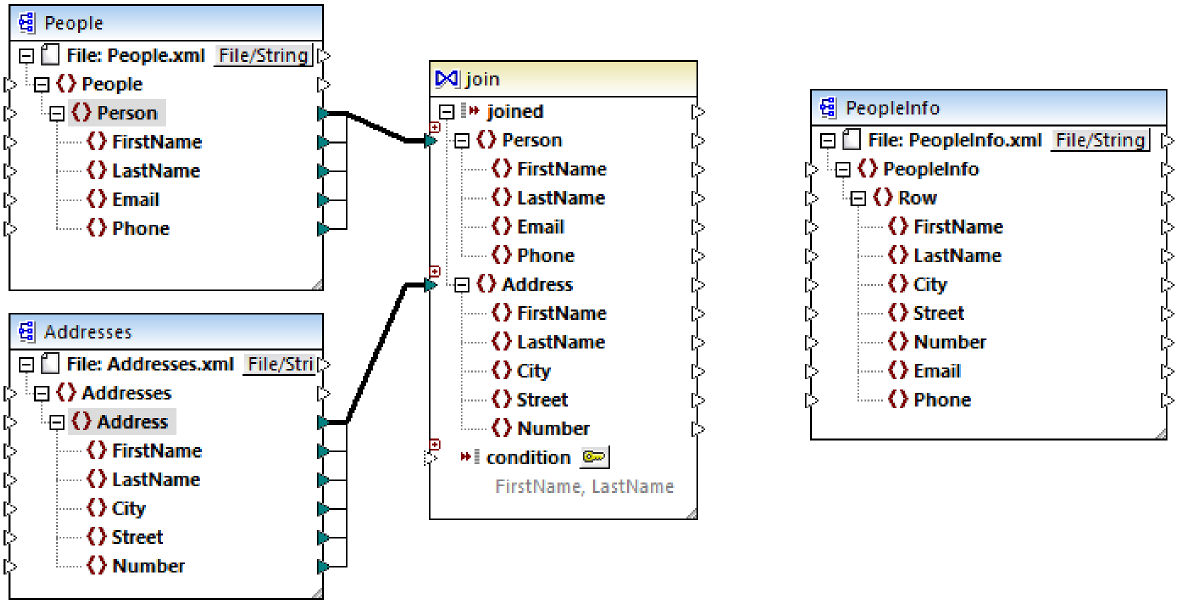
Step 3: Add the Join component

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):

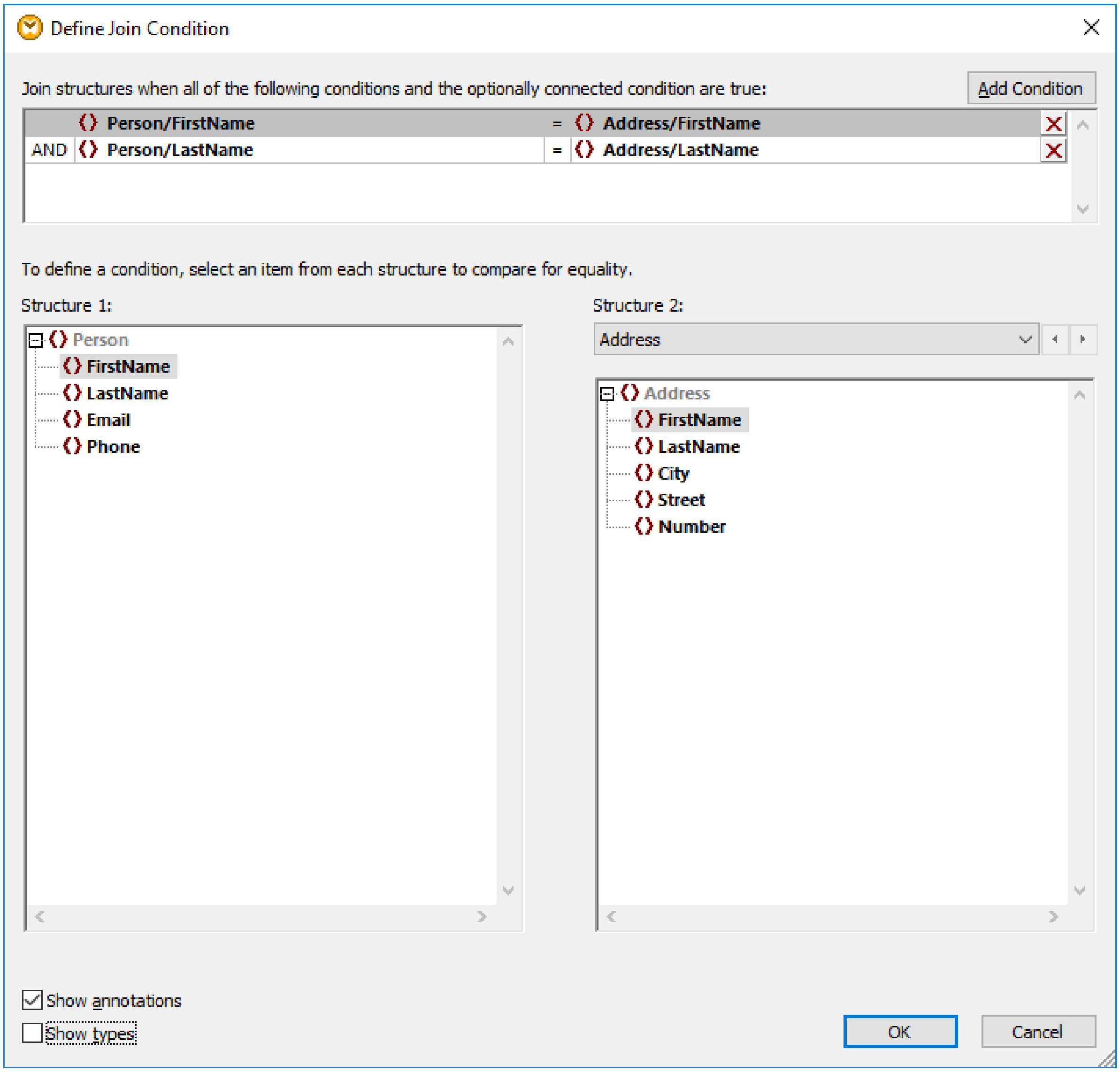


   Observe the structure of the Join component. It has two `nodes/rows` items, which makes it possible to connect to it the two structures that need to be compared (in this case, the `Person` and the `Address` structures).

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.

Structure 1:

Structure 2: Address

☑ Show annotations

□ Show types

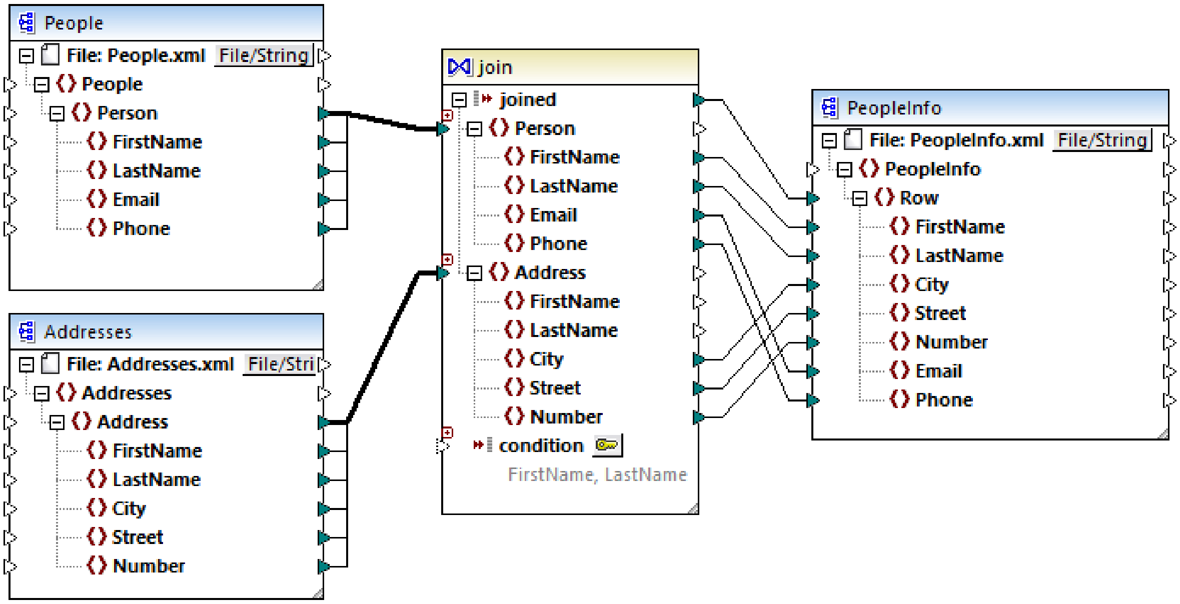
In some mappings, a condition consisting of one comparison may be sufficient to perform the join. However, in this example, it is important that two comparisons are created:

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** pane. As expected, each person record (`<Row>`) now includes the full address details, joined from two different sources.

5.4.4 Join Database Data

In mappings that read data from databases, you can join database objects such as tables or views by adding a Join component to the mapping. For example, you could combine data from two or more tables bound by foreign key relationships, which is the typical way data is stored in relational databases. The result would be the same as if you ran against the database an SQL query where two or more tables are joined by means of an INNER JOIN (or LEFT JOIN, if applicable) operation.

Depending on the kind of data connected to the join component, the join operation can happen either in standard (non-SQL) mode, or in SQL mode. Joins in non-SQL mode are undertaken by MapForce, while joins in SQL mode are undertaken by the database from which the mapping reads data.

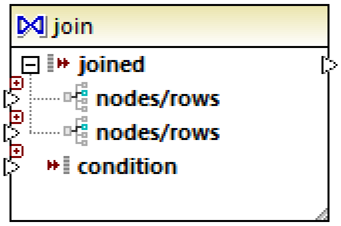
Joins in non-SQL mode are more flexible because they support more component types as input (for example, the join can be between tables from different databases, or between XML structures and database tables). For an example of a non-SQL join, see Example: Join XML Structures 379. On the other hand, a non-SQL join causes the mapping engine to perform memory-costly comparisons (because the total number of comparisons represents the cross-join, or Cartesian product, of all joined structures). Usually this process is very fast and causes negligible load in mappings which are not data-intensive; however, if the joined data sources consist of a huge number of records, then the mapping will require significant time to execute. If your mappings must process a very large number of records, consider licensing MapForce Server Advanced Edition, which includes dedicated join optimization to speed up the mapping execution.

A join in SQL mode accepts only eligible database objects as input (such as tables or views), so it is not as flexible as a non-SQL join. However, it offers better mapping performance because it is executed natively by the database. For further information, see Joins in SQL Mode 386.

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 243. 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 378.



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 375.

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 64).
- Join components should not be connected to input parameters or results of inline user-defined functions. If such connections exist, validation errors will occur during mapping validation.
- When you connect eligible database components (such as tables or views) directly to a Join component, an **SQL mode** (SQL) button automatically appears at the top-right corner of the Join component. When enabled, this button provides special SQL features applicable to the join operation (see Joins in SQL Mode 386).
- 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.

5.4.4.1 Joins in SQL Mode

When you connect eligible database components (such as tables or views) directly to a join component, an **SQL mode** SQL button appears at the top-right corner of the join component. When SQL mode is enabled, the join operation is undertaken by the database from where the mapping reads data. In other words, MapForce will internally send to the database a query with the appropriate SQL syntax to select and combine data from all tables that take part in the join. Importantly, you do not need to write any SQL; the required query is produced based on how you visually designed the Join component on the mapping, as you will see in subsequent examples.

For SQL mode to be possible, the following conditions must be met:

1. Both objects (tables or views) that are to be joined must be from the same database.
2. Both objects that are to be joined must originate from the same MapForce component. (Note that you can quickly add/remove database objects in a component as follows: right-click the database component, and select **Add/Remove/Edit Database Objects** from the context menu.)
3. The Join condition (or conditions) must be defined only from the component properties (by right-clicking the header of the join component, and selecting **Properties**), and not on the mapping (see also Adding Join Conditions 375).

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 389.

4. All database tables must not yet be in the current target context. When the join result is used in a target component, none of the joined tables may be connected directly or indirectly to any target parent nodes. For more information about how a mapping is executed, see Mapping Rules and Strategies 77.

You can view or control the SQL mode through the **SQL** (SQL) button at the top-right corner of the join component, as follows:

SQL	SQL mode is disabled (join will be executed by MapForce (or, if applicable, by MapForce Server).
SQL	SQL mode is enabled (join will be executed by the database).

If the SQL button is missing, this means that SQL mode is not meaningful or not supported for the data that is being joined.

In certain cases, the SQL mode must be explicitly disabled (SQL), for example:

- When your mapping requires join conditions outside of the join component properties (that is, conditions defined on the mapping and connected to the `condition` item of the join component).
- When you want to join tables from different databases. Use a standard (non-SQL) join if you need to join tables from different databases.

Changing the Join mode

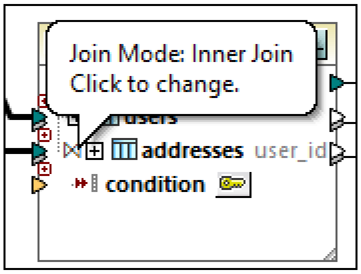
When the Join component is in **SQL mode** SQL, you can join database tables or views in one of the following ways:

- INNER JOIN - Only records which satisfy the condition in both input sets are returned by the Join component.
- LEFT OUTER JOIN - The Join component includes all records from the "leftmost" table (in MapForce, this is the topmost table of a Join component), plus those records from the subsequently joined table that satisfy the join condition.

To view the join mode of a table or view on the Join component, observe the icon shown in front of the joined table or view. One of the following icons can be shown for any joined table or view except the first one:

- Inner Join ⋈
- Left Join ⟕

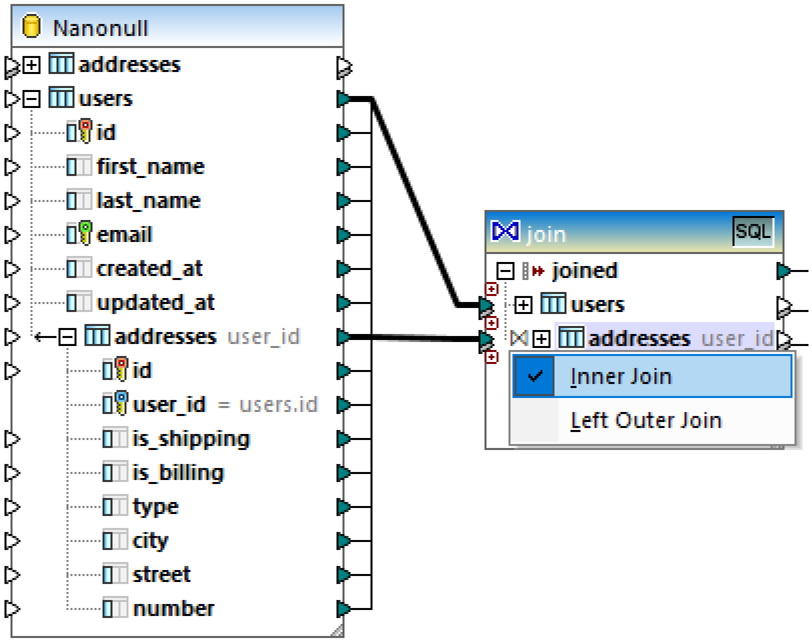
To display a tooltip with details about the join, move the cursor over the icon:



Join Mode: Inner Join
Click to change.

To change the join mode, do one of the following:

- Click the **Inner Join** ⋈ or **Left Join** ⟕ icon in front of each joined table or view, and select **Inner Join** or **Left Outer Join** from the context menu.
- Right-click the second (or third, fourth, etc) joined table or view on the Join component, and select **Join Type | Inner Join**, or **Join Type | Left Outer Join** from the context menu.



Note the following:

- If you changed the join mode to LEFT OUTER JOIN, then the upper table or view represents the "left" side of the join.
- Changing the join mode affects the data returned by the join component in the same way that INNER JOIN or LEFT JOIN affects the result of a SQL query in a database.

Alias names

It is often the case that joined database tables or views contain identical field names in both joined structures. When SQL mode is enabled, such items appear on the component prefixed by the keyword "AS". For example, if two joined tables contain an "id" field, this field appears as "id" on the first joined table and as "id AS id2" on the second joined table. Joined tables can also produce alias names, for example, if the same table is joined to itself.

The alias field or table names are important if you need to refer to them subsequently on a mapping. For example, imagine a case when you want to filter or sort the result of the join. To achieve this, the output of the join component can be connected to a SQL WHERE/ORDER component, where you would enter the SQL WHERE and ORDER BY clauses.

To refer to a field from the WHERE clause, write the table name, followed by a dot (.) character, followed by the field name. To refer to a table alias, use the alias name as it appears on the Join component. In the ORDER BY clause, you can either use the same technique (`table.field`), or write just the alias field name (the name that appears after "AS").

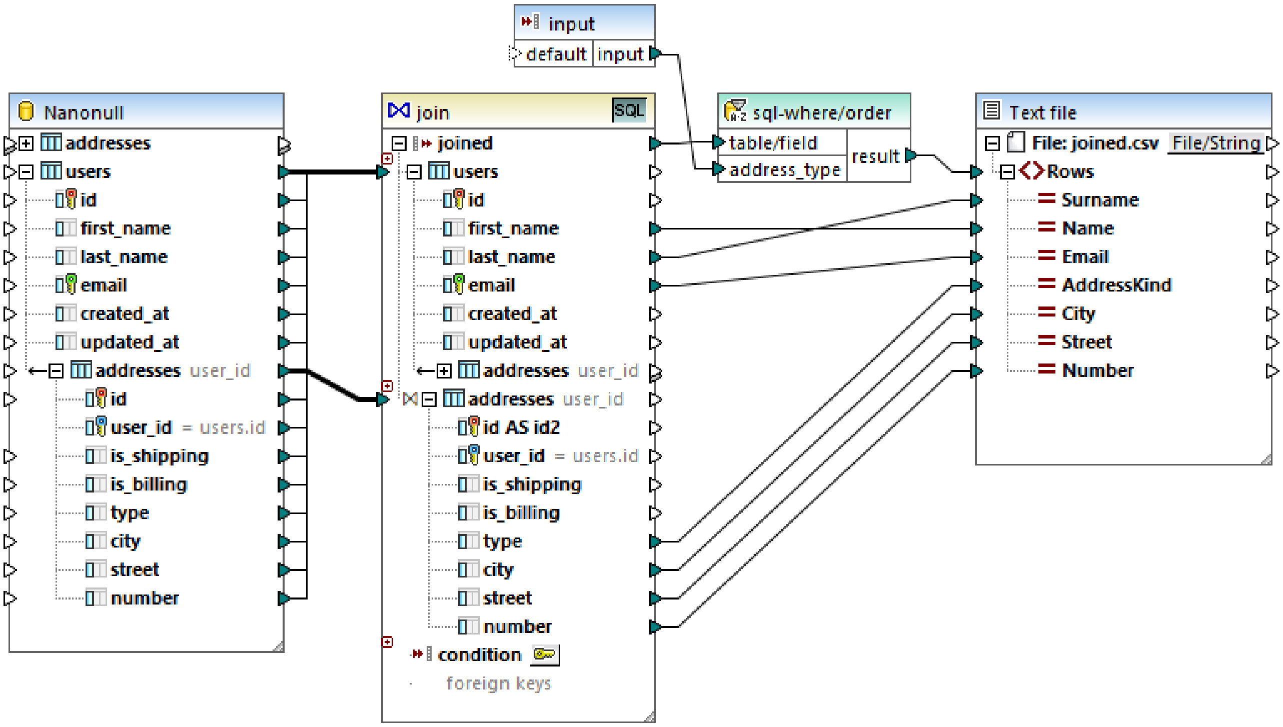
For an example mapping which uses SQL WHERE/ORDER clauses, see Example: Join Tables in SQL Mode 389.

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) 413.

5.4.4.2 Example: Join Tables in SQL Mode

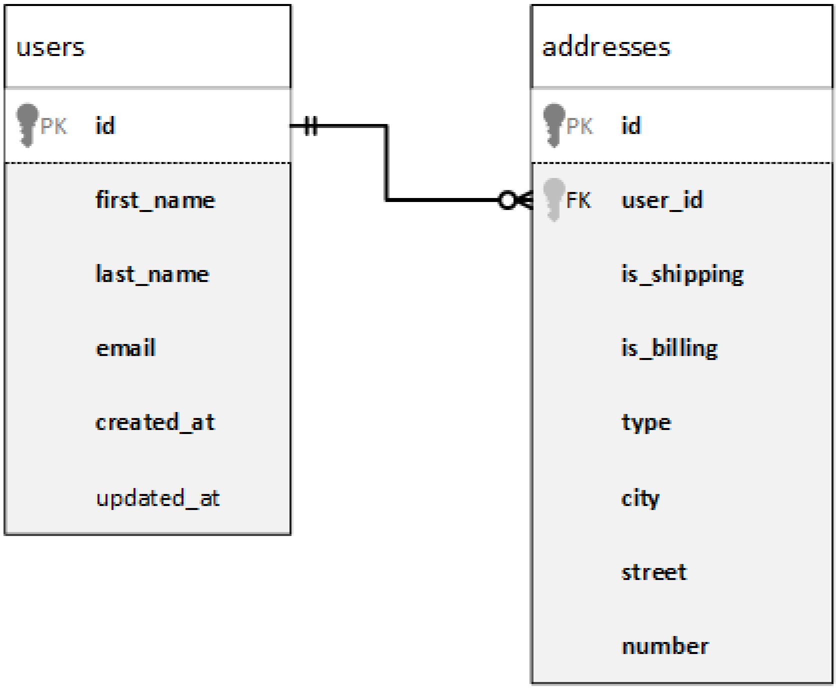
This example illustrates how to join data from two database tables, using a MapForce join component. The join operation is performed in SQL mode, as described in Joins in SQL Mode 386. Note that joining three or more tables works in a very similar way, see also Example: Create CSV Report from Multiple Tables 397.

The example is accompanied by a mapping sample which is available at the following path:
<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\JoinDatabaseTables.mfd.

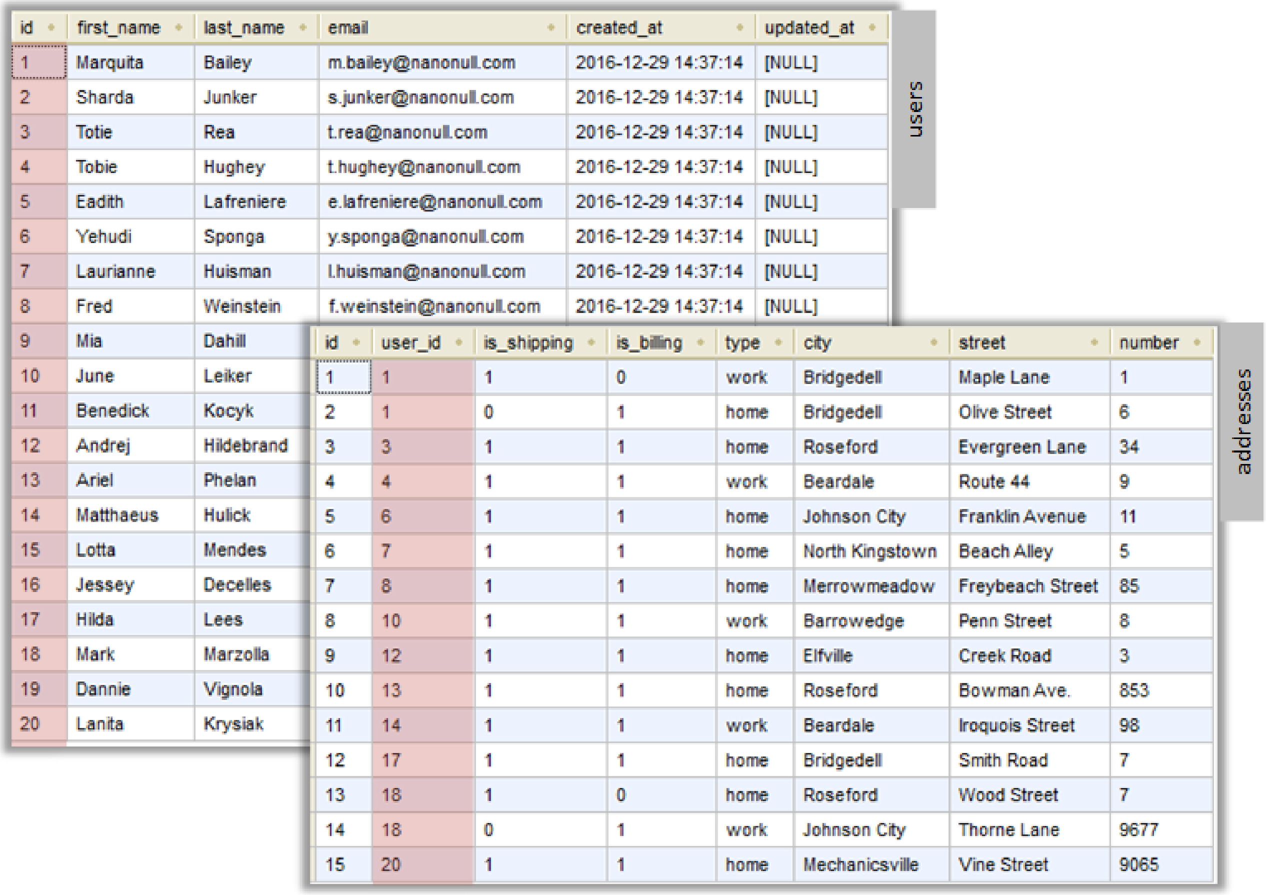


JoinDatabaseTables.mfd

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 names and emails, and the second table (`addresses`) stores people's 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.



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			

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.

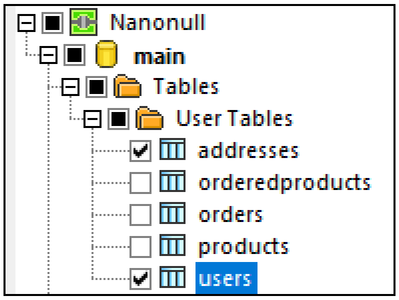
The goal of the mapping is to retrieve full data (name, surname, email, city, street, number) of all users that have at least one address in the addresses table. It should also be possible to easily retrieve only addresses of a specific kind (for example, only home addresses, or only work addresses). 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 243. 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.

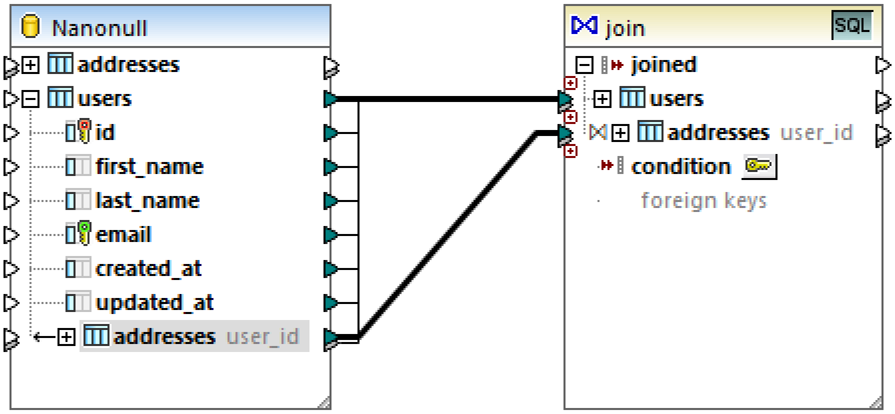
Step 1: Add the source database

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\MapForce2025\MapForceExamples\Tutorial\**, and click **Connect**.
4. When prompted, select the `addresses` and `users` tables.



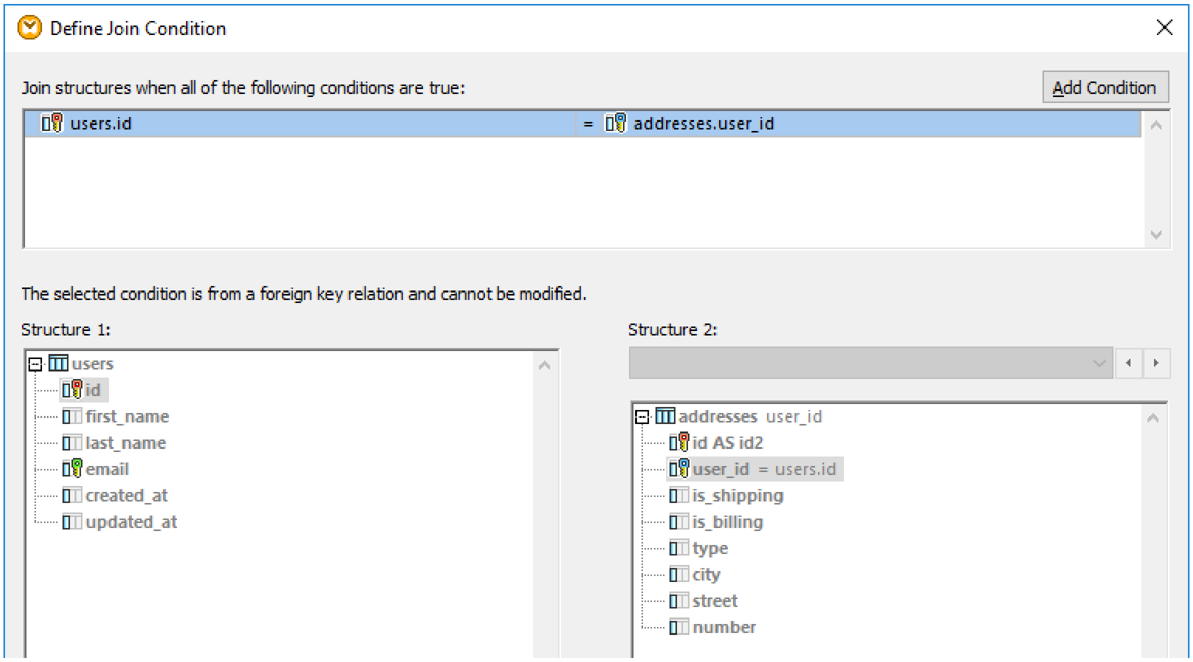
Step 2: Add the join component

1. On the **Insert** menu, click **Join**. (Alternatively, click the **Join** toolbar button).
2. Draw a connection from the `users` table to the first input of the join component.
3. Expand the `users` table and draw a connection from the `addresses` table (child of `users`) to the second input of join component. The ⊕ button enables you to add more tables if necessary; however, in this example, only two tables are being joined.



Note: It is also possible to add the connection directly from the `addresses` table (the one which is not child of `users`); however, in this case, the join conditions would have to be defined manually, as described in Adding Join Conditions 375. 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:

users.id = addresses.user_id

The selected condition is from a foreign key relation and cannot be modified.

Structure 1:

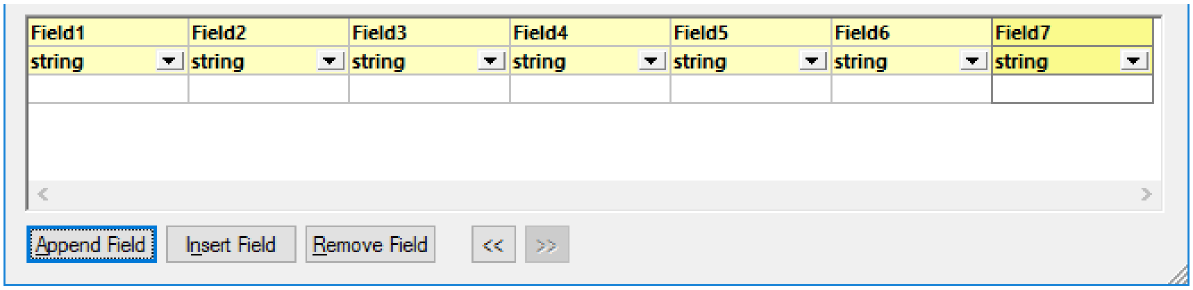
- users
  - id
  - first_name
  - last_name
  - email
  - created_at
  - updated_at

Structure 2:

- addresses user_id
  - id AS id2
  - user_id = users.id
  - is_shipping
  - is_billing
  - type
  - city
  - street
  - number

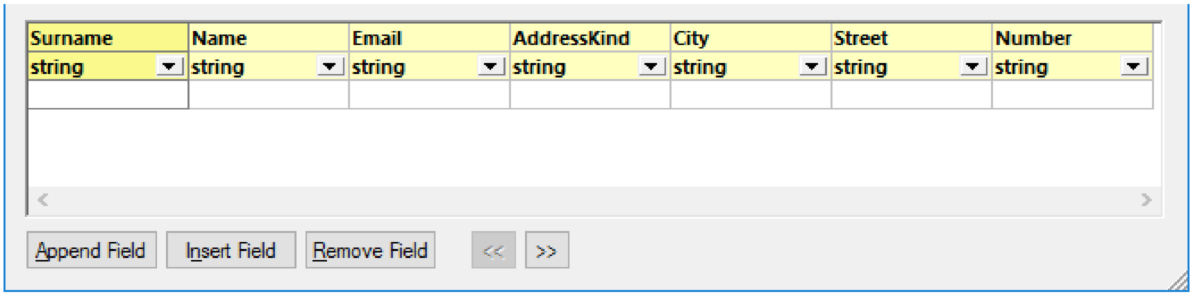
Step 3: Add the target CSV component

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.



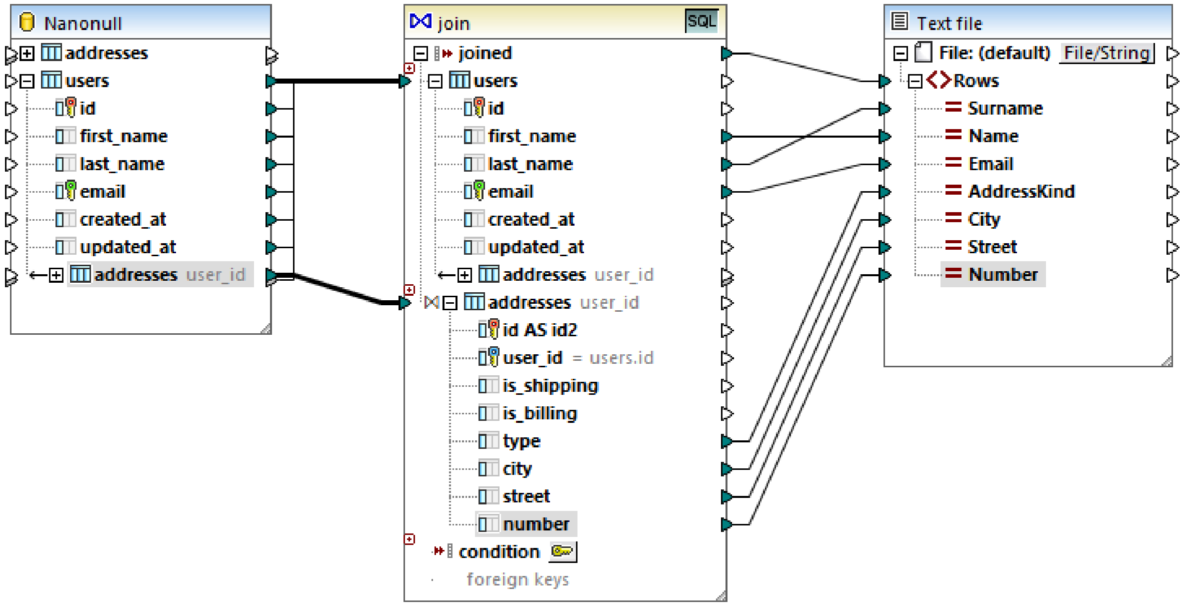
Field1	Field2	Field3	Field4	Field5	Field6	Field7
string	string	string	string	string	string	string

4. Double-click the title cell of each field to give it a descriptive name (this will make your mapping easier to read).



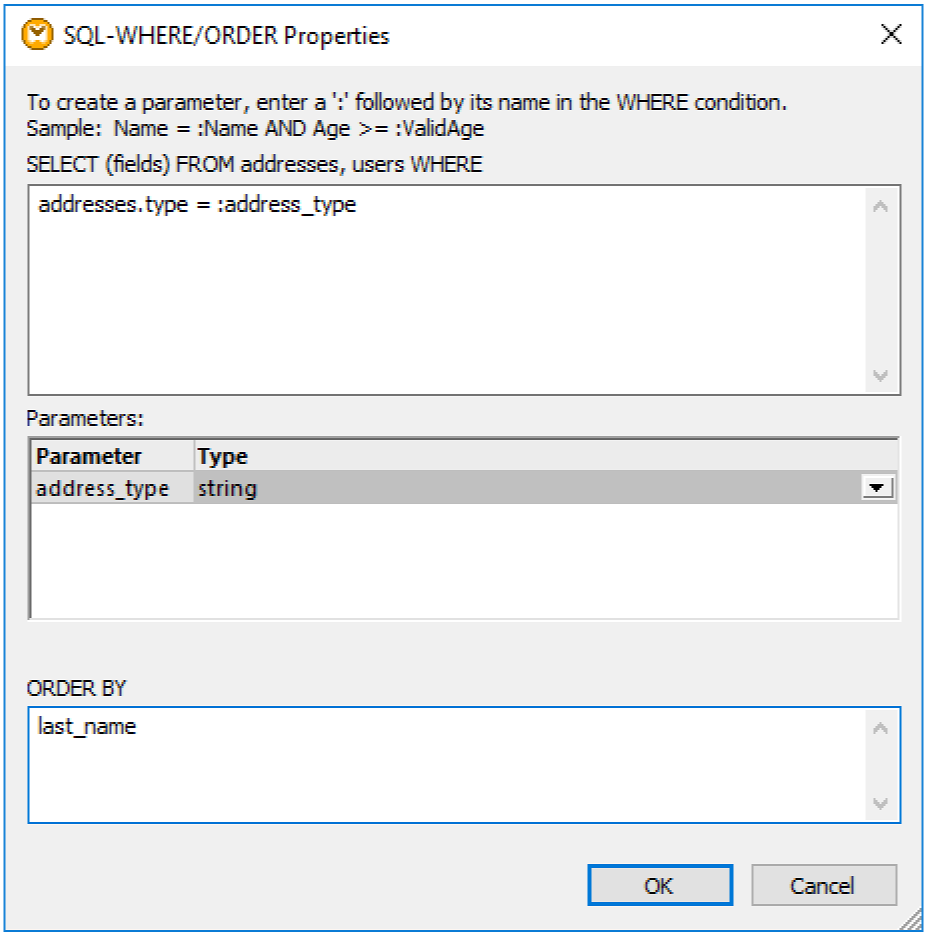
Surname	Name	Email	AddressKind	City	Street	Number
string	string	string	string	string	string	string

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
```

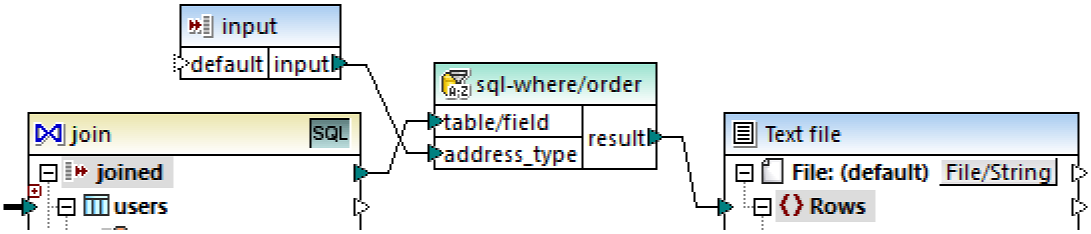
Parameters:

Parameter	Type
address_type	string

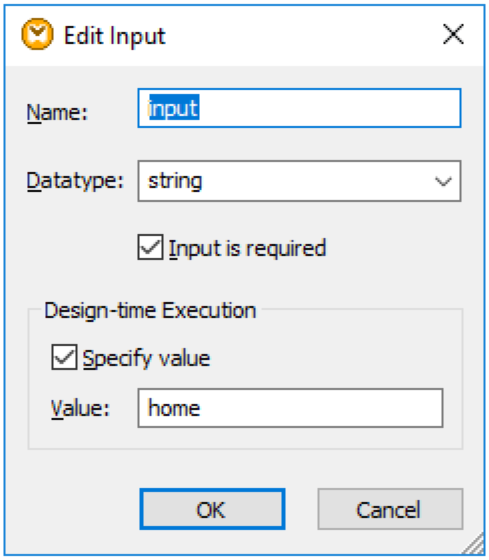
ORDER BY

```
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 246. Because this example has made use of the already existing table relationships, you did not have to define any join conditions manually. For an example that shows you how to define join conditions manually, see Example: Create CSV Report from Multiple Tables 397.

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 type of the join (INNER JOIN) is indicated by the **Inner Join** ⋈ icon in front of the **addresses** table on the Join component. You can also change the join type to LEFT OUTER JOIN ⟕, as described in Changing the Join Mode 387. Note, however, that changing the join mode does not affect the output of this example.

The SQL WHERE/ORDER component added in step 4 enables filtering (to retrieve either home or work addresses) and sorting the recordset. Notice that the WHERE clause created a parameter **`:address_type`** of type `string`. This parameter makes it possible to supply the address kind (home or work) from the mapping. For more information about SQL WHERE/ORDER, see Filtering and Sorting Database Data (SQL WHERE/ORDER) 413.

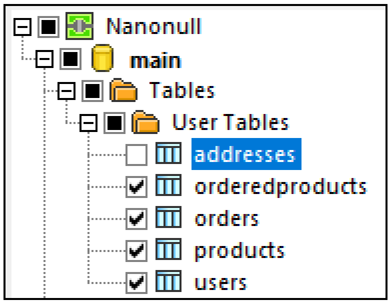
Finally, the input component makes it possible to supply the actual parameter value when the mapping runs. Note that, when the mapping runs outside MapForce (for example, when it is executed by MapForce Server on a different machine), the input must be supplied at mapping runtime as a command-line parameter, so the design-time value mentioned above is ignored. For more information, see Supplying Parameters to the Mapping 346.

5.4.4.3 Example: Create CSV Report from Multiple Tables

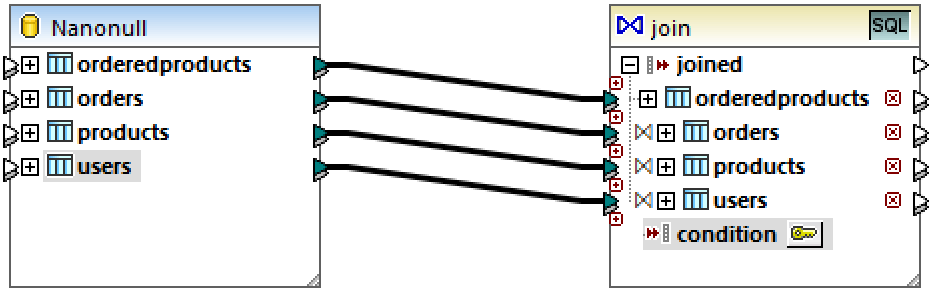
This example illustrates how to join multiple database tables for the purpose of extracting data into a single report in CSV format. The database used in this example is called **Nanonull.sqlite** and is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\**. This database stores information about a fictitious business (which includes orders, products, users and their addresses). As is typically the case with relational databases, the information is normalized and spread across multiple tables. For example, the `users` table stores user personal data (which includes first name, last name, and email). The database also stores information about products ordered by users, in two different tables: `orders` (which includes the unique ID of the order, and the time when it took place) and `orderedproducts` (which includes a list of products ordered, and their quantity). Furthermore, the names of the products themselves is stored in a separate table called `products`.

The goal of the example is to produce a report based on data extracted from various tables, so as to make it clear who ordered certain products, when, and in which quantity. To achieve the mapping goal, follow the steps below:

1. On the **Insert** menu, click **Database**.
2. When prompted to select a database kind, click **SQLite**, and then click **Next**.
3. Browse for the **Nanonull.sqlite** database mentioned above, and click **Connect**.
4. When prompted, select the tables `orderedproducts`, `orders`, `products`, and `users`, and click **OK**.



5. Add a Join component to the mapping and create four nodes/rows items by clicking the **Add input** (⊞) button.
6. Connect the four tables from the database component to the corresponding input items of the Join component.

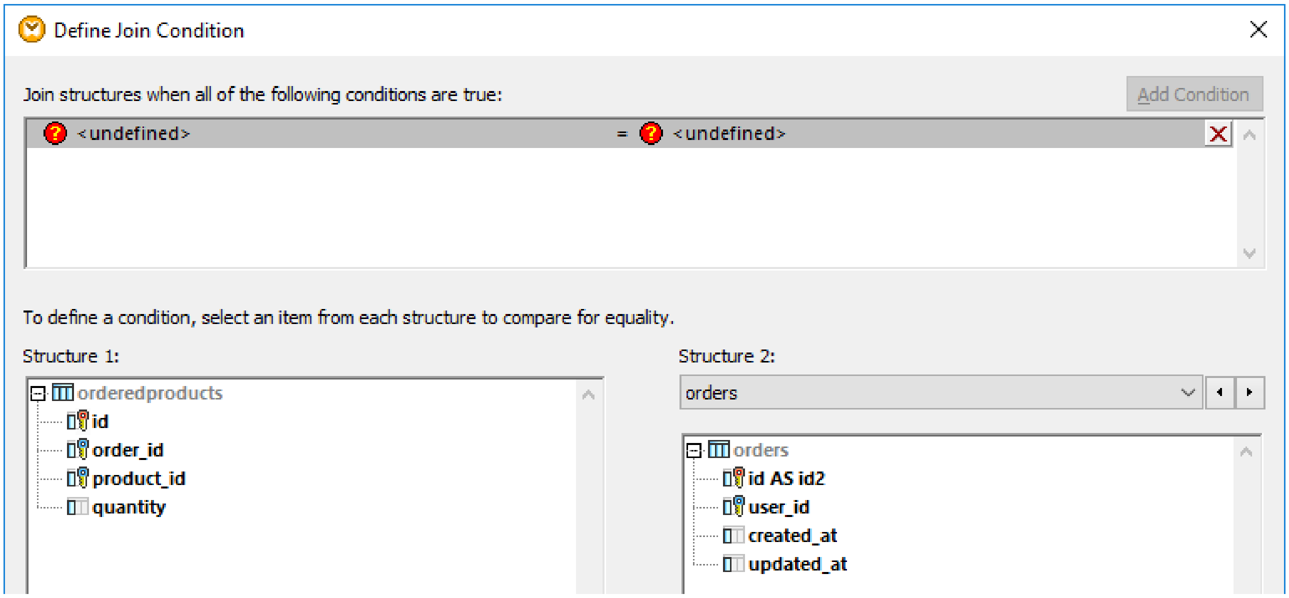


Note: In an alternative scenario, you could connect the table `orderedproducts` to the Join component, then the table `orders` (the one which is nested under it, not the one at the same level), and so on, so that all joined tables are nested under the same "root" table, see also Handling Database Relationships 246. The mapping result would be the same if you joined tables this way. The difference is that in this example the join conditions are created manually, whereas in the alternative scenario the join conditions would be created automatically by MapForce. For an example of joining tables without having to define join conditions manually, see Example: Join Tables in SQL Mode[389]. Another mapping where all joined tables are under the same "root" table is available at the following path: **<Documents>\Altova\MapForce2025\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**.


Define Join Condition

Join structures when all of the following conditions are true:

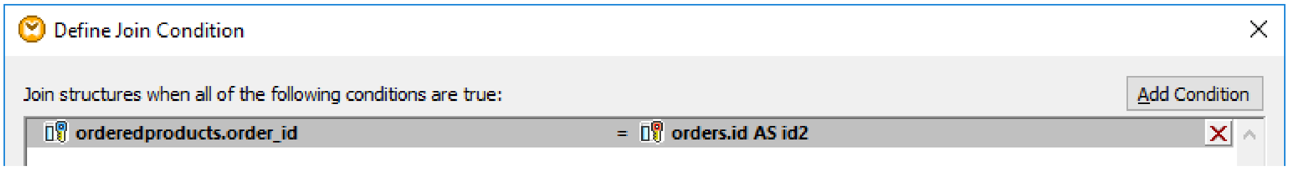
<undefined> = <undefined>

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 (id AS id2, user_id, created_at, updated_at)

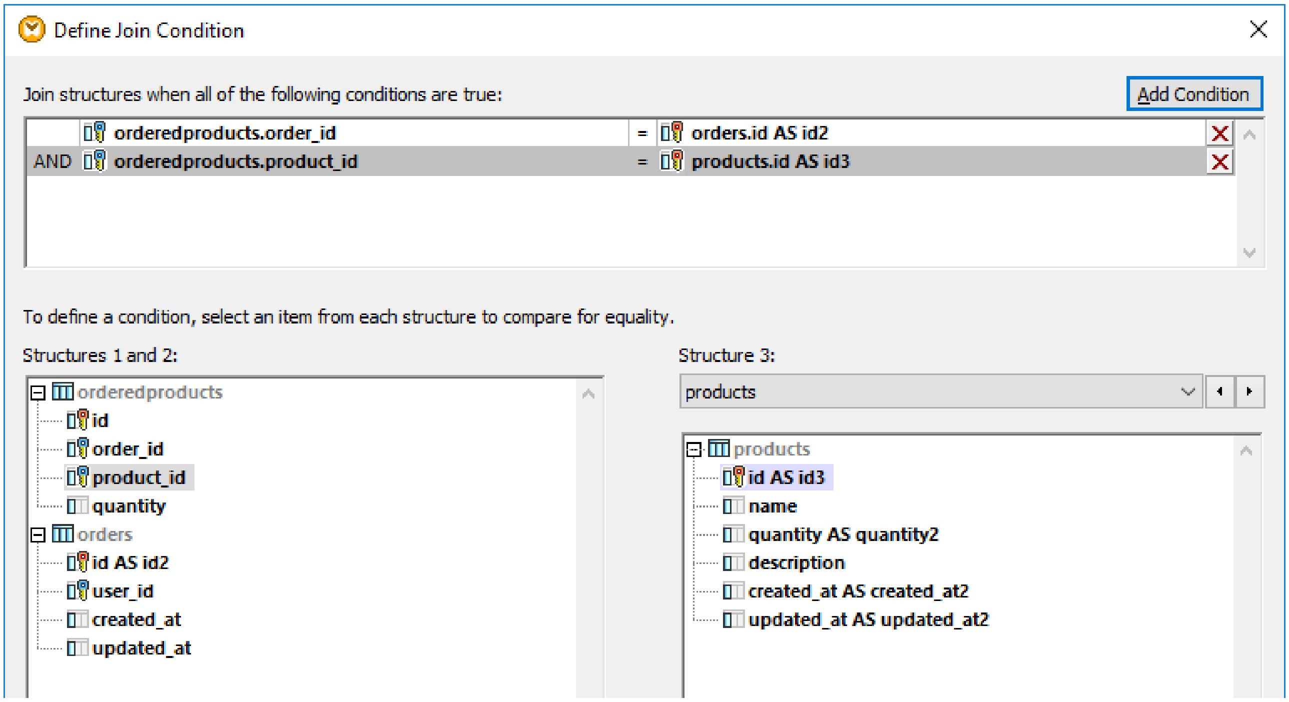
To define the first join condition, click the `order_id` item in the left pane and the `id` item in the right pane. Now the fields `orderedproducts.order_id` and `orders.id`. are paired:


Define Join Condition

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:

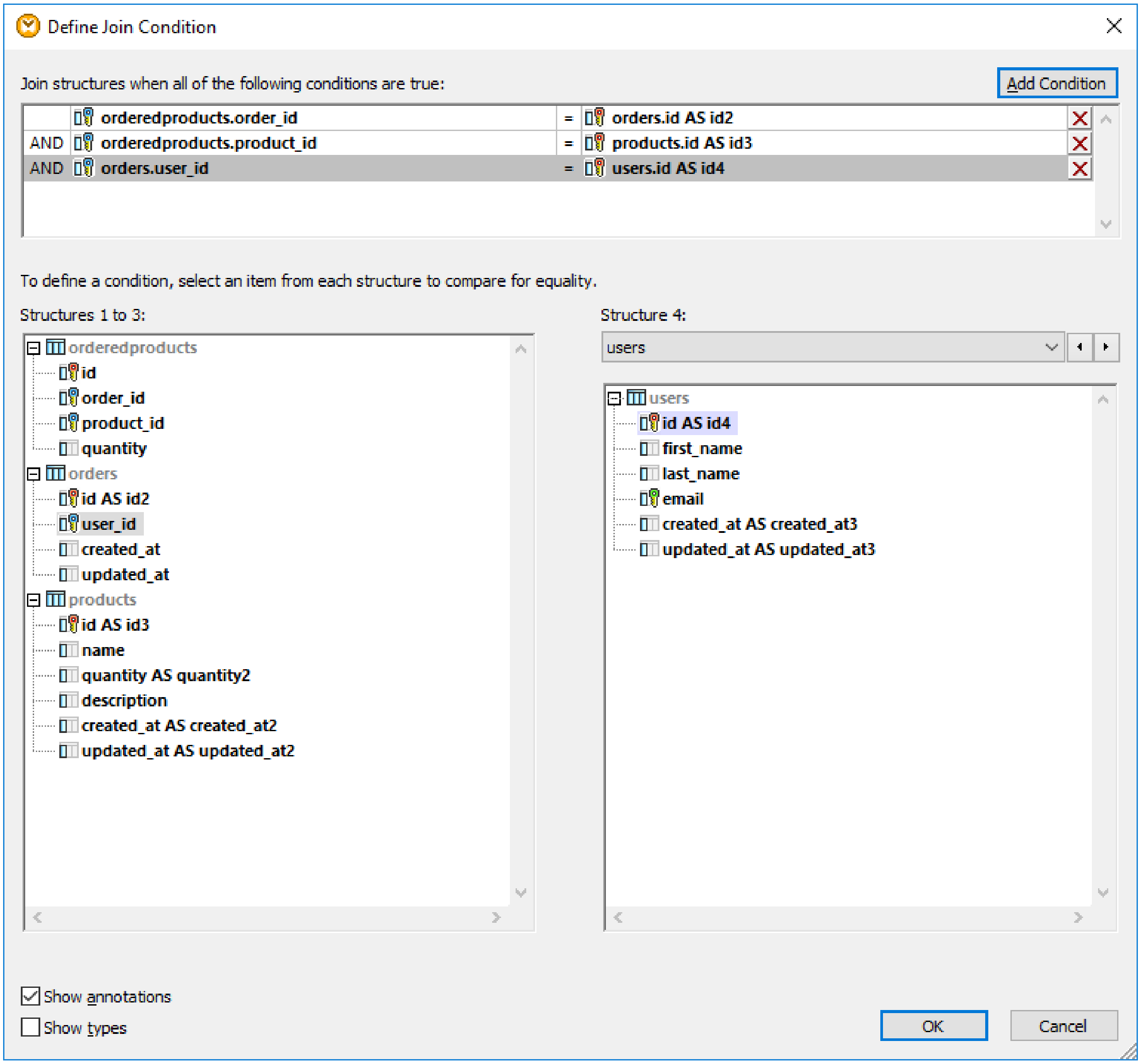
	orderedproducts.order_id	=	orders.id AS id2
AND	orderedproducts.product_id	=	products.id AS id3

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.

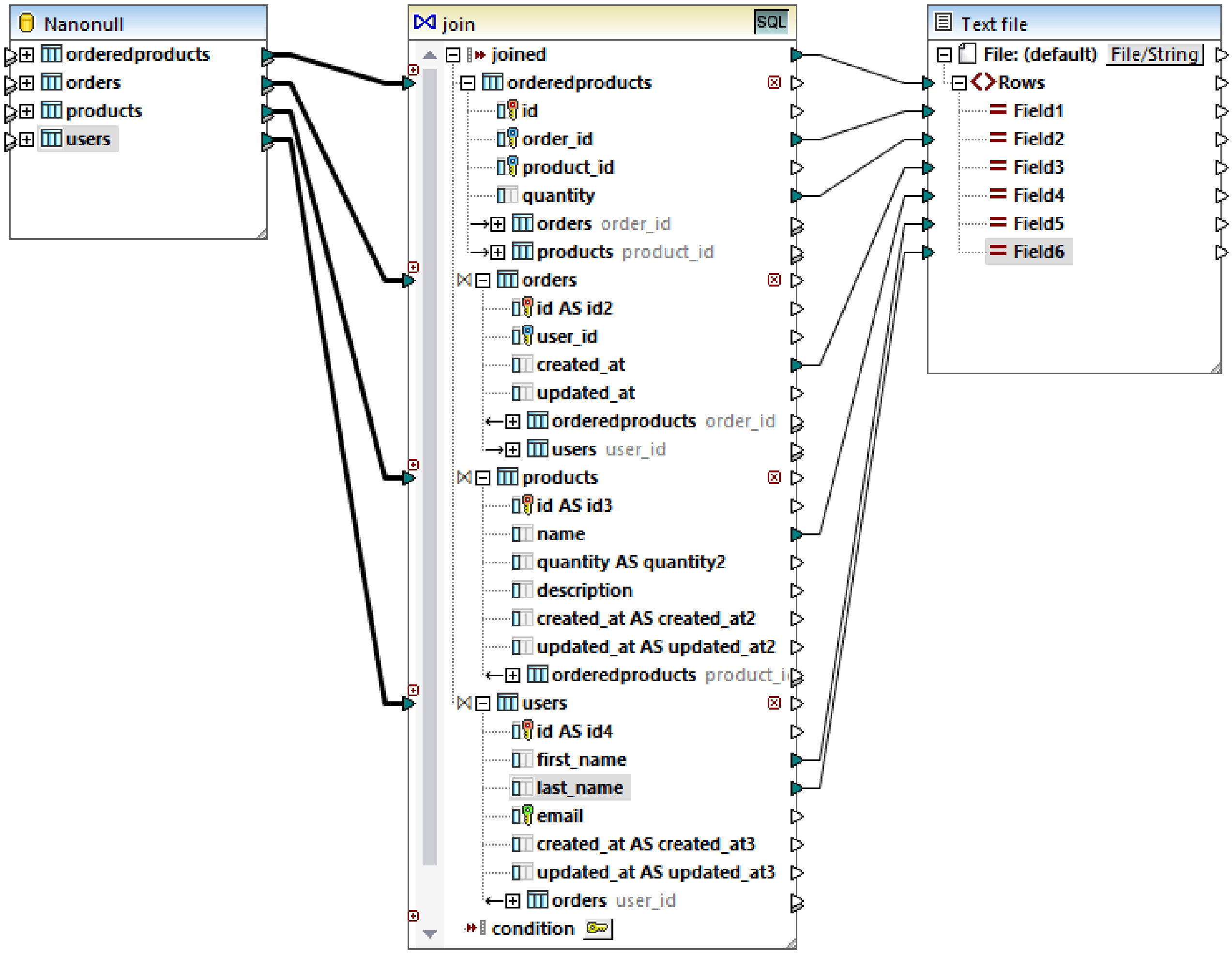
Structures 1 to 3:

Structure 4: users

☑ 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 323), and connect items from the Join component to the target CSV component as illustrated below:



The mapping illustrated above produces a report (in CSV format) compiled from all four tables involved in the join, as follows:

- ID of the order (taken from the `orderedproducts` table)
- Quantity of ordered items (taken from the `orderedproducts` table)
- Time when the order took place (taken from the `orders` table)
- Name of the product ordered (taken from the `products` table)
- First name and last name of the user who ordered the product (taken from the `users` table).

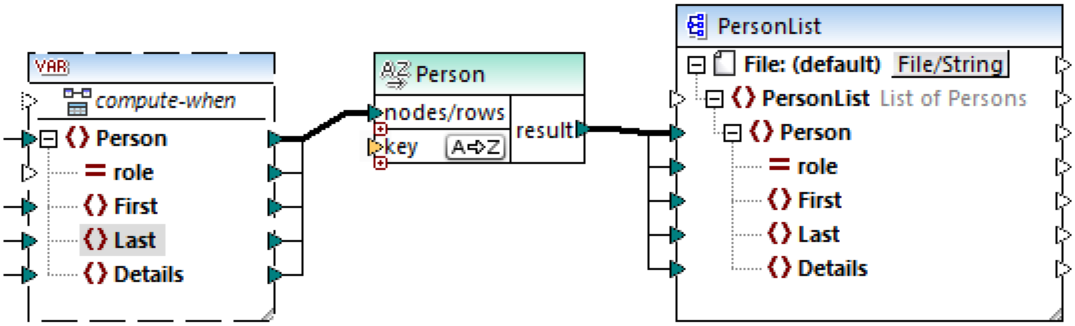
All the tables in this example are joined using INNER JOIN mode. For information about changing the join mode to LEFT OUTER JOIN, see Changing the join mode 387.

5.5 Sort Components

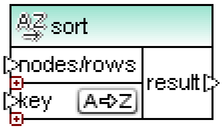
To sort input data based on a specific sort key, use a Sort component. The Sort component supports the following target languages: XSLT2, XQuery, and Built-in. When the transformation language is "Built-in", the Sort component can be used to sort database table data. Better performance is, however, achieved using an SQL-WHERE/ORDER component. For more details, see Filtering and Sorting Database Data (SQL WHERE/ORDER) 413 .

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).



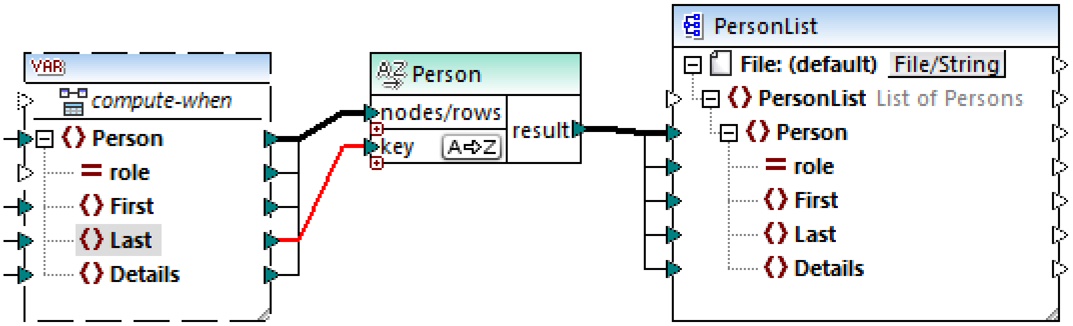
- On the **Insert** menu, click **Sort** (alternatively, click the **Sort** toolbar button). This inserts the Sort component in its "unconnected" form.



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`.

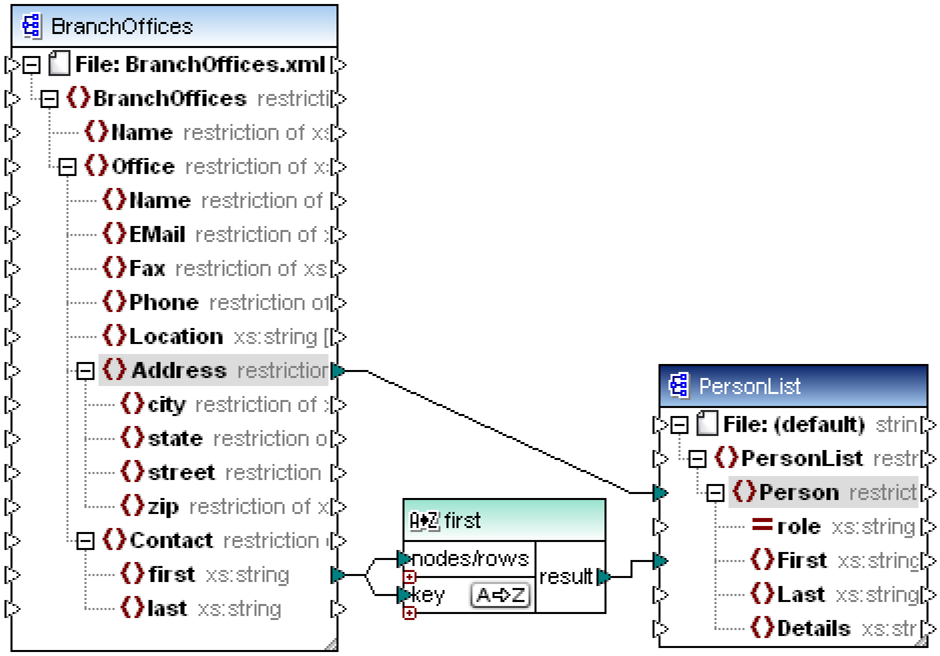


To change the sort order:

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

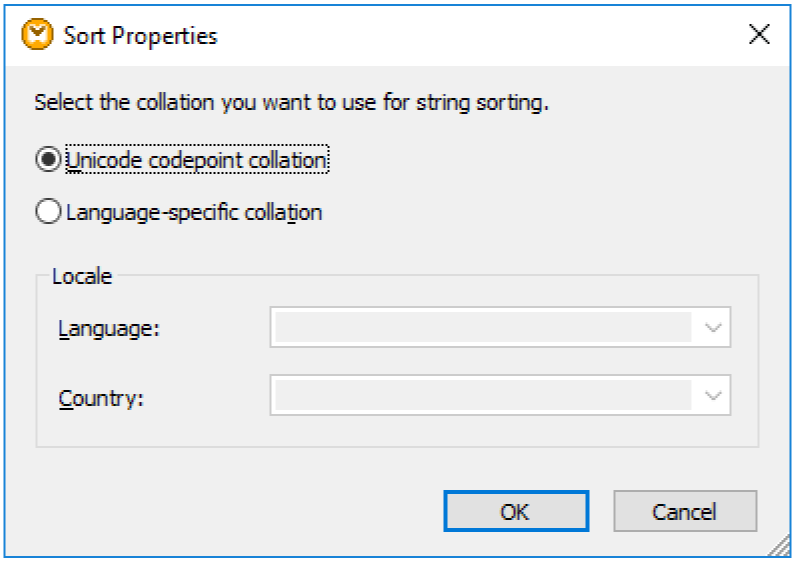
To sort input data consisting of simple type items:

- 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:

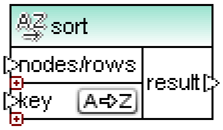
OK Cancel

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.

5.5.1 Sorting by Multiple Keys

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

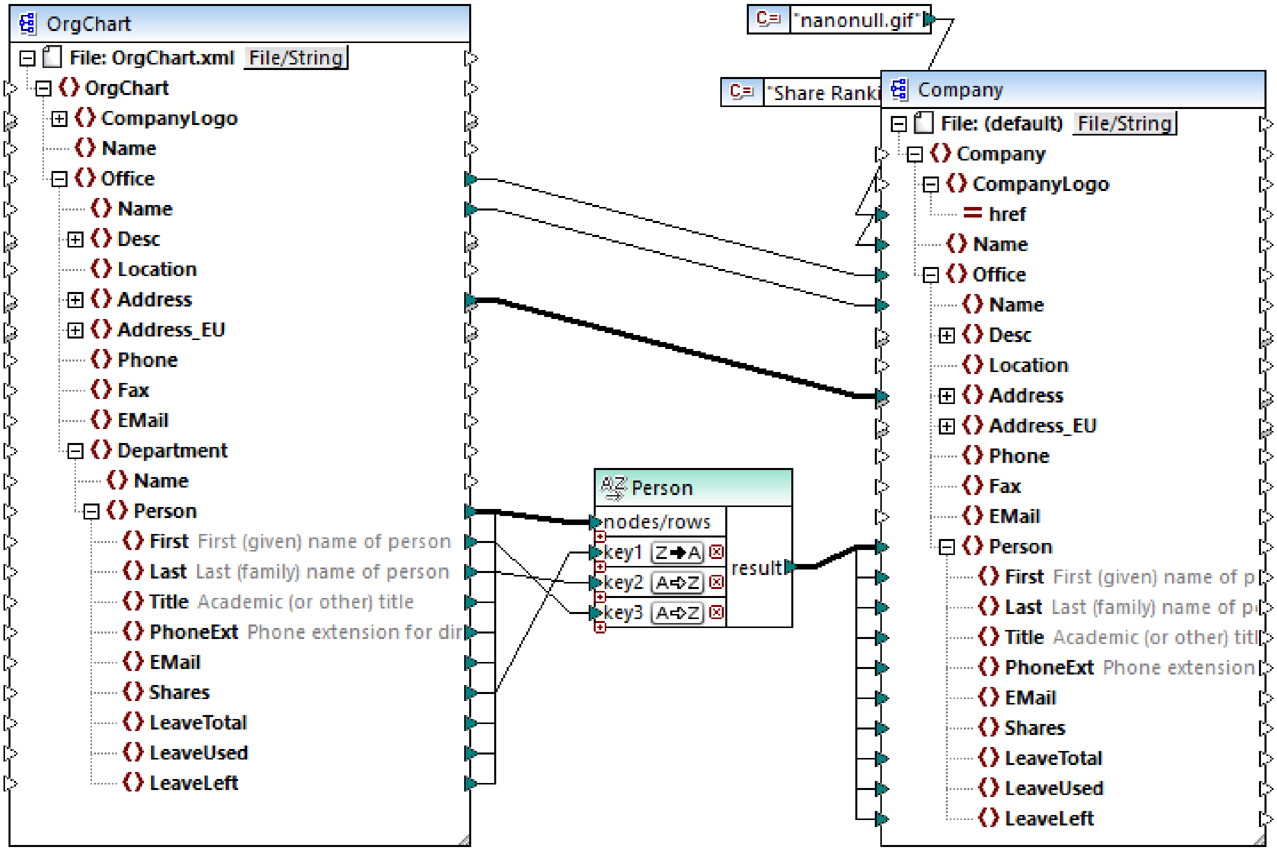


Default Sort component

If you want to sort by multiple keys, adjust the Sort component as follows:

- Click the **Add Key** (⊞) icon to add a new key (for example, key2 in the mapping below).
- Click the **Delete Key** (⊠) icon to delete a key.
- Drop a connection onto the ⊞ icon to add a key and also connect to it.

A mapping which illustrates sorting by multiple key is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\SortByMultipleKeys.mfd**.



SortByMultipleKeys.mfd

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

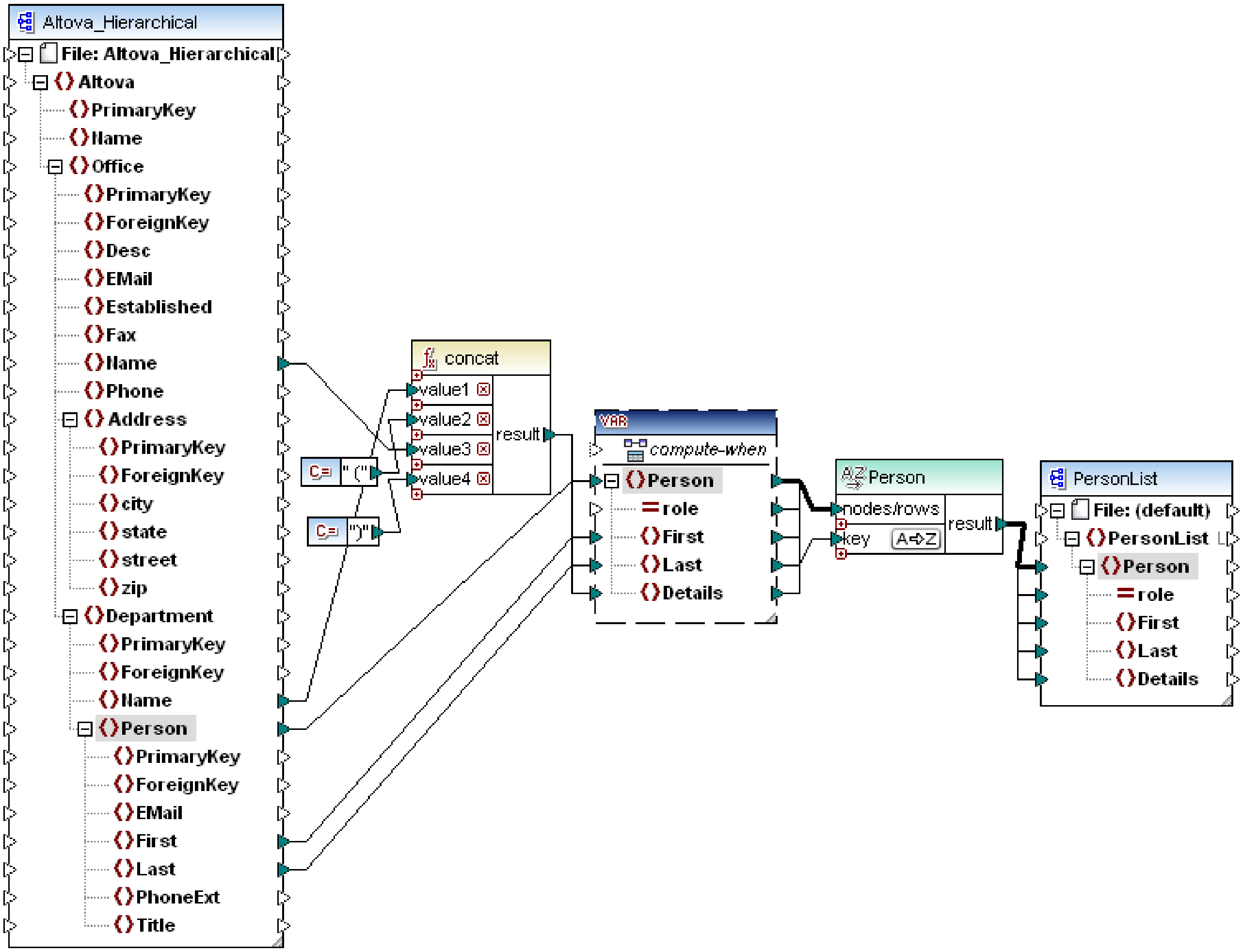
1. `Shares` (number of shares a person holds)
2. `Last` (last name)
3. `First` (first name)

Note that the position of the sorting key in the Sort component determines its sort priority. For example, in the mapping above, records are initially sorted by the number of shares. This is the sorting key with the highest priority. If the number of shares is the same, people are then sorted by their last name. Finally, when multiple people have the same number of shares and the same last name, the person's first name is taken into account.

The sort order of each key can be different. In the mapping above, the key `Shares` has a descending sort order (Z-A), while the other two keys have ascending sort order (A-Z).

5.5.2 Sorting with Variables

In some cases, it may be necessary to add intermediate variables to the mapping in order to achieve the desired result. This example illustrates how to extract records from an XML file, and sort them, with the help of intermediate variables. The example is accompanied by a mapping sample located at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\Altova_Hierarchical_Sort.mfd**.



Altova_Hierarchical_Sort.mfd

This mapping reads data from a source XML file called **Altova_Hierarchical.xml** and writes it to a target XML file. As shown above, the source XML contains information about a fictitious company. The company is divided into offices. Offices are sub-divided into departments, and departments are further divided into people.

The target XML component, `PersonList`, contains a list of `Person` records. The `Details` item is meant to store information about the office and department where the person belongs.

The aim is to extract all persons from the source XML and sort them alphabetically by last name. Also, the office and department name where each person belongs must be written to the `Details` item.

To achieve this goal, this example makes use of the following component types:

1. The **concat** function. In this mapping, this function returns a string in the format `Office(Department)`. It takes as input the office name, the department name, and two constants which supply the start and end brackets. See also Add a Function to the Mapping 436.
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 77). 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 360 and Copy-All Connections 56.

3. A Sort component, which performs the actual sorting. Notice that the key input of the `Sort` component is connected to the `Last` item of the variable, which sorts all person records by their last name.

5.6 Filters and Conditions

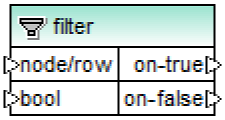
When you need to filter data, or get a value conditionally, you can use one of the following component types:

- Filter: Nodes/Rows ( )
- SQL WHERE/ORDER ( )
- If-Else Condition ( )

You can add these components to the mapping either from the **Insert** menu, or from the **Insert Component** toolbar. Importantly, each of the components above has specific behavior and requirements. The differences are explained in the following sections.

Filtering nodes or rows

When you need to filter data, including XML nodes or CSV rows, use a **Filter Nodes/Rows** component. The **Filter Nodes/Rows** component enables you to retrieve a subset of nodes from a larger set of data, based on a true or false condition. Its structure on the mapping area reflects this:



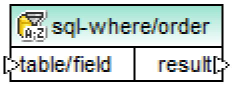
In the structure above, the condition connected to **bool** determines whether the connected **node/row** goes to the **on-true** or **on-false** output. Namely, if the condition is true, the **node/row** will be redirected to the **on-true** output. Conversely, if the condition is false, the **node/row**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

When your mapping needs to consume only items that *meet* the filter condition, you can leave the **on-false** output unconnected. If you need to process the items that *do not meet* the filter condition, connect the **on-false** output to a target where such items should be redirected. If you want to add an exception when the filter condition is not met, connecting the **on-false** output is mandatory (see Adding Exceptions 432).

For a step-by-step mapping example, see Example: Filtering Nodes 410.

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 413.

Returning a value conditionally

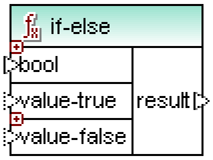
If you need to get a single value (not a node or row) conditionally, use an **If-Else Condition**. Note that If-Else conditions are not suitable for filtering nodes or rows. Unlike **Filter Nodes/Rows** components, an **If-Else**

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>
```

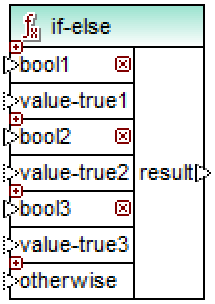
An **If-Else Condition** would enable you to return, for each item in the list, the value "high" if temperature exceeds 20 degrees Celsius, and value "low" if temperature is lower than 5 degrees Celsius.

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** (⊠) button. This feature enables you to check for multiple conditions and return a different value for each condition, if it is true.



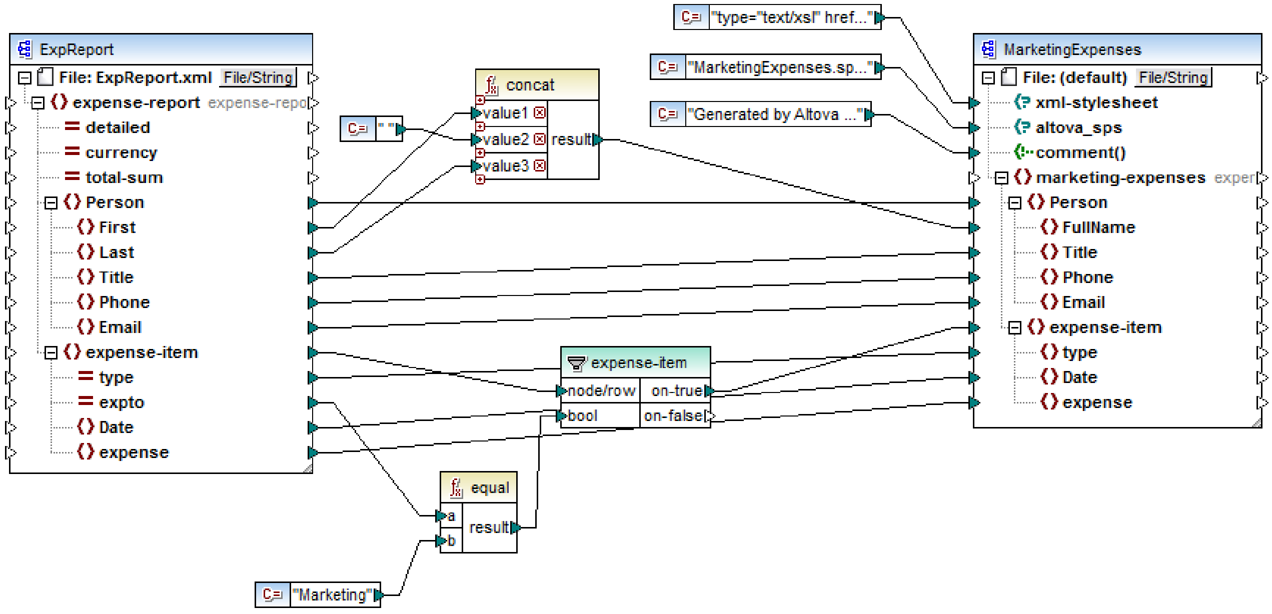
Expanded **If-Else Conditions** are evaluated from top to bottom (first conditions is checked first, then the second one, and so on). If you want to return a value when none of the conditions are true, connect it to **otherwise**.

For a step-by-step mapping example, see Example: Returning a Value Conditionally[412].

5.6.1 Example: Filtering Nodes

This example shows you how to filter nodes based on a true/false condition. A **Filter: Nodes/Rows** ( ) component is used to achieve this goal. The technique illustrated in this example works in the same way not only for XML, but also for other component types, such as CSV or text. In case of databases, although you can use a filter, it is recommended to use a SQL WHERE/ORDER component instead, for better performance (see SQL WHERE / ORDER Component 413).

The mapping described in this example is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\MarketingExpenses.mfd**.



As shown above, the mapping reads data from a source XML which contains an expense report ("ExpReport") and writes data to a target XML ("MarketingExpenses"). There are several other components between the target and source. The most relevant component is the **expense-item** filter ( ), which represents the subject of this topic.

The goal of the mapping is to filter out only those expense items that belong to the Marketing department. To achieve this goal, a filter component has been added to the mapping. (To add a filter, click the **Insert** menu, and then click **Filter: Nodes/Rows**.)

To identify whether each expense item belongs to Marketing, this mapping looks at the value of the "expto" attribute in the source. This attribute has the value "Marketing" whenever the expense is a marketing expense. For example, in the code listing below, the first and third expense item belongs to Marketing, the second belongs to Development, and the fourth belongs to Sales:

```
...
   <expense-item type="Meal" expto="Marketing">
      <Date>2003-01-01</Date>
      <expense>122.11</expense>
   </expense-item>
   <expense-item type="Lodging" expto="Development">
      <Date>2003-01-02</Date>
      <expense>122.12</expense>
   </expense-item>
   <expense-item type="Lodging" expto="Marketing">
      <Date>2003-01-02</Date>
      <expense>299.45</expense>
   </expense-item>
   <expense-item type="Entertainment" expto="Sales">
      <Date>2003-01-02</Date>
      <expense>13.22</expense>
   </expense-item>
...
```

XML input before the mapping is executed

On the mapping area, the **node/row** input of the filter is connected to the **expense-item** node in the source component. This ensures that the filter component gets the list of nodes that it must process.

To add the condition based on which filtering should occur, we have added the **`equal`** function from the MapForce core library, see also Add a Function to the Mapping 436. The **`equal`** function compares the value of the `expto` attribute to a constant which has the value `Marketing`. (To add a constant, click the **Insert** menu, and then click **Constant**.)

Since we need to filter only those items that satisfy the condition, we connected only the **on-true** output of the filter to the target component.

When you preview the mapping result, by clicking the **Output** pane, MapForce evaluates, for each expense-item node, the condition connected to the **bool** input of the filter. When the condition is true, the expense-item node is passed on to the target; otherwise, it is ignored. Consequently, only the expense items matching the criteria are displayed in the output:

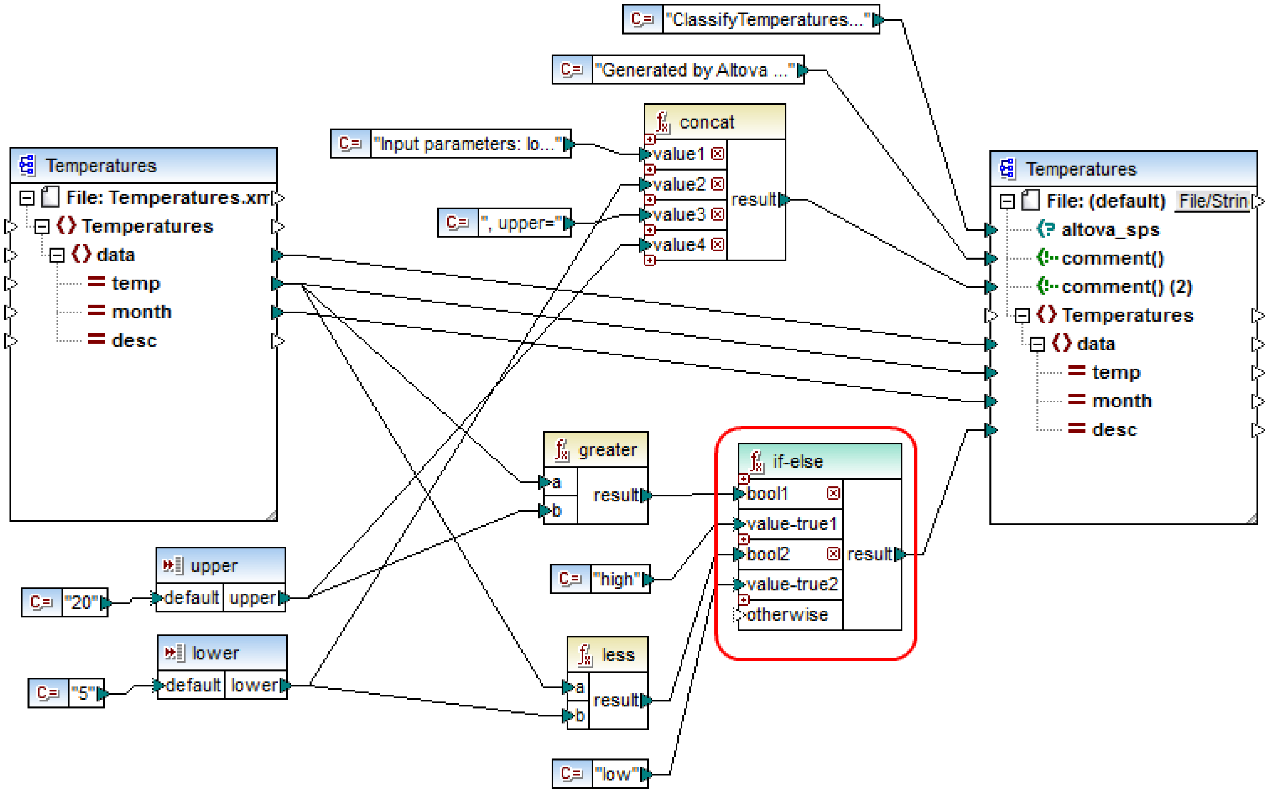
```
...
   <expense-item>
      <type>Meal</type>
      <Date>2003-01-01</Date>
      <expense>122.11</expense>
   </expense-item>
   <expense-item>
      <type>Lodging</type>
      <Date>2003-01-02</Date>
      <expense>299.45</expense>
   </expense-item>
...
```

XML output after the mapping is executed

5.6.2 Example: Returning a Value Conditionally

This example shows you how to return a simple value from a component, based on a true/false condition. An **If-Else Condition** ( ) is used to achieve the goal. Note that **If-Else Conditions** should not be confused with filter components. **If-Else Conditions** are only suitable when you need to process simple values conditionally (string, integer, etc.). If you need to filter complex values such as nodes, use a filter instead (see Example: Filtering Nodes 410).

The mapping described in this example is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\ClassifyTemperatures.mfd**.



This mapping reads data from a source XML which contains temperature data ("Temperatures") and writes data to a target XML which conforms to the same schema. There are several other components between the target and source, one of them being the **if-else** condition (highlighted in red), which is also the subject of this topic.

The goal of the mapping is to add short description to each temperature record in the target. Specifically, if temperature is above 20 degrees Celsius, the description should be "high". If the temperature is below 5 degrees Celsius, the description should be "low". For all other cases, no description should be written.

To achieve this goal, conditional processing is required; therefore, an If-Else Condition has been added to the mapping. (To add an If-Else Condition, click the **Insert** menu, and then click **If-Else Condition**.) In this mapping, the If-Else Condition has been extended (with the help of the ⊞ button) to accept two conditions: **bool1** and **bool2**.

The conditions themselves are supplied by the `greater` and `less` functions, which have been added from the MapForce core library, see also Add a Function to the Mapping[436]. 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[346].)

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** pane. 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

5.6.3 Filter and Sort Database Data

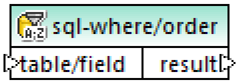
When you need to filter and sort database data, use the **SQL/NoSQL-WHERE/ORDER** component. This enables you to manually enter an SQL WHERE clause that filters data. Optionally, you can also specify an ORDER BY clause if you want to sort the record set by a particular database field, in ascending or descending order.

The **SQL/NoSQL-WHERE/ORDER** component must be connected to a table or field of a database mapping component. It is also possible to connect an **SQL/NoSQL-WHERE/ORDER** component with a **Join** component if you need to filter the joined set or records. For more information, see Joining Database Data[384].

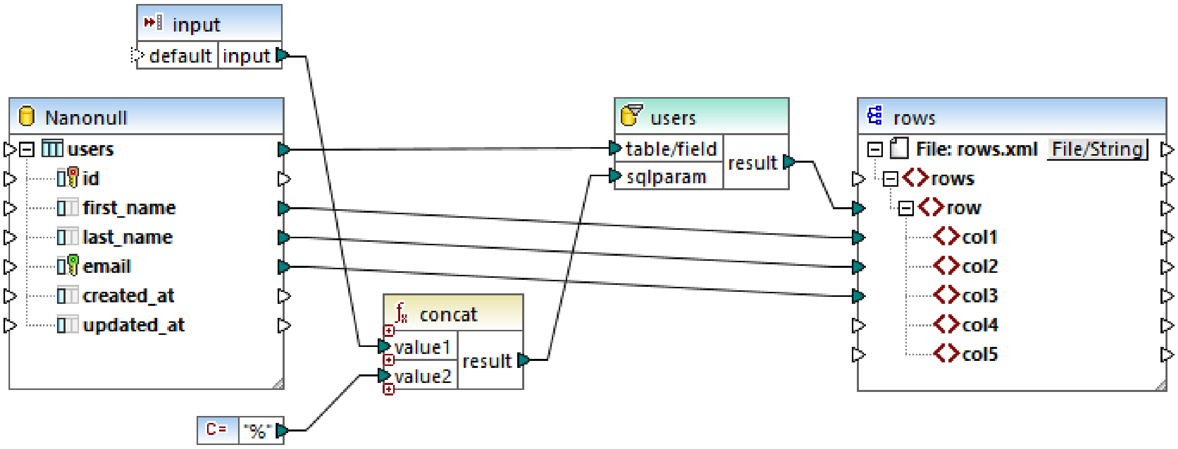
Add an SQL/NoSQL-WHERE/ORDER component

To add an **SQL/NoSQL-WHERE/ORDER** component to the mapping, follow the instructions below:

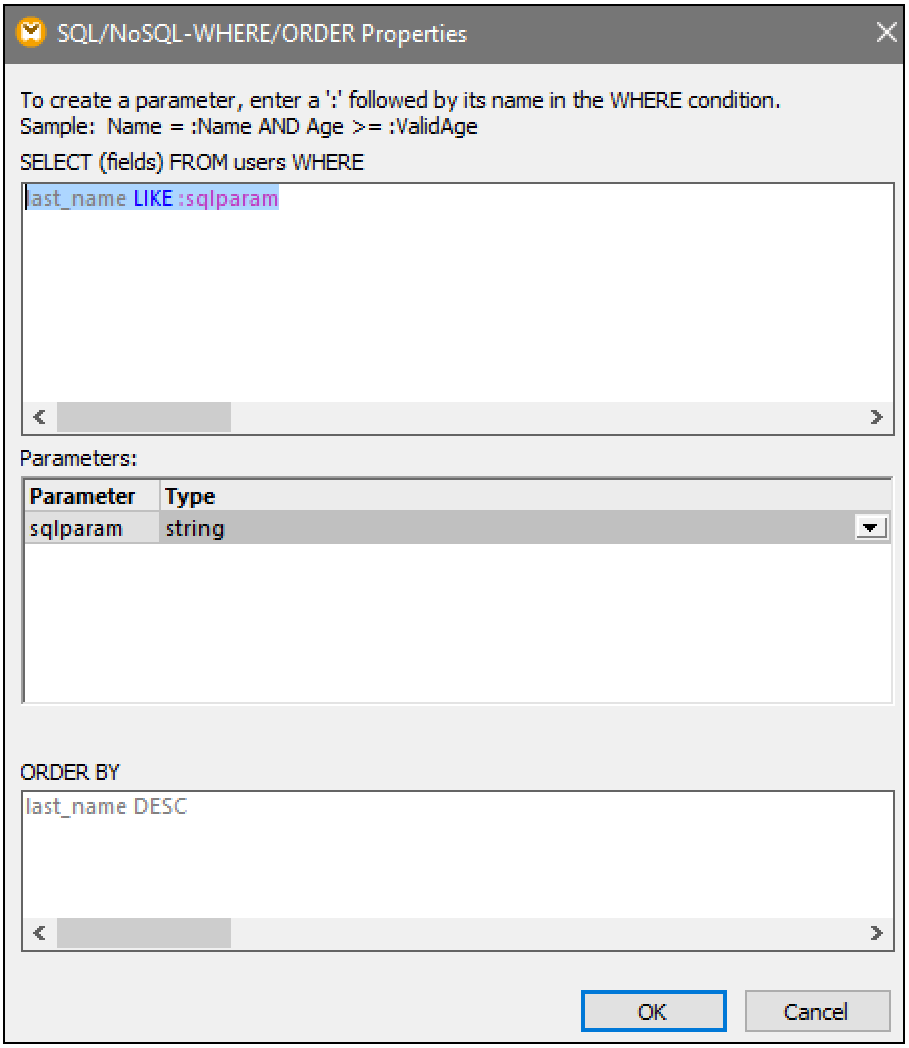
1. Go to the **Insert** menu and click **SQL/NoSQL-WHERE/ORDER**. By default, this component has the following structure:



2. Connect a source database table or field to the `table/field` item of the **SQL/NoSQL-WHERE/ORDER** component. You can find the sample mapping **`FilterDatabaseRecords.mfd`** (*see screenshot below*) in the following folder: **`<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\`**. In this mapping, the **SQL/NoSQL-WHERE/ORDER** component takes the data from the source table `users`, filters all its records and selects only those where the last name begins with the letter M (*see the explanation in the subsection below*).



3. Double-click the header of the **SQL/NoSQL-WHERE/ORDER** component. Alternatively, right-click it and select **Properties** from the context menu. This opens the dialog box **SQL/NoSQL-WHERE/ORDER Properties**.



To create a parameter, enter a ':' followed by its name in the WHERE condition.
Sample: Name = :Name AND Age >= :ValidAge

SELECT (fields) FROM users WHERE

```
last_name LIKE :sqlparam
```

Parameters:

Parameter	Type
sqlparam	string

ORDER BY

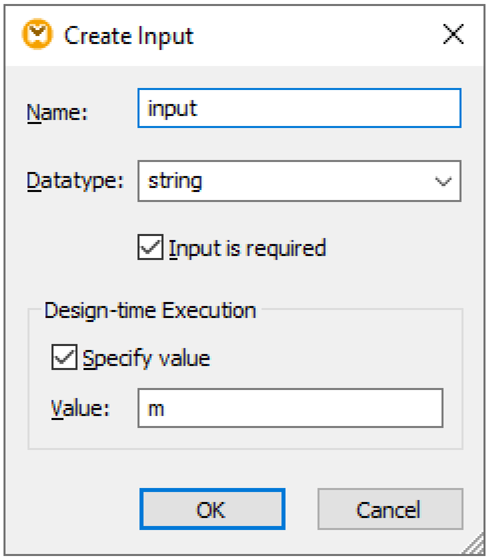
```
last_name DESC
```

4. Type an SQL WHERE clause in the text box at the top. In our example, the SQL Where clause is as follows: **`last_name LIKE :sqlparam`**. Optionally, type an ORDER BY clause. The image above illustrates the WHERE and ORDER BY clauses defined in the **`FilterDatabaseRecords.mfd`** mapping (these settings are further explained below). For more examples, see Creating WHERE and ORDER BY Clauses 417.

Parameters in SQL/NoSQL-WHERE/ORDER components

The **SQL/NoSQL-WHERE/ORDER** component used in the mapping **`FilterDatabaseRecords.mfd`** has the following WHERE clause: **`last_name LIKE :sqlparam`**, where `last_name` refers to the name of the database field in the connected table; `LIKE` is an SQL operator; `:sqlparam` creates a parameter called `sqlparam` in the mapping.

Parameters in the **SQL/NoSQL-WHERE/ORDER** component are optional. They are useful if you want to pass a value to the WHERE clause from the mapping. Without parameters, the WHERE clause above could have been written as follows: `Last LIKE "M%"`. This would retrieve all persons whose last name begins with the letter M. In order to make this query even more flexible, we have added a parameter instead of `"M%"`. This makes it possible to supply any other letter from the mapping: e.g., D, and thus retrieve people whose last name begins with D by changing a constant or a mapping input parameter. In the mapping above, the input letter comes from an input component called `input`. If you double-click the title bar of this component and open its properties, you will notice that `m` is given as a design-time execution value (*see screenshot below*).



Create Input

Name: input

Datatype: string

☑ Input is required

Design-time Execution

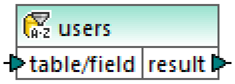
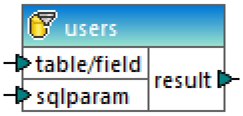
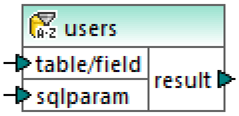
☑ Specify value

Value: m

In the mapping, the SQL wildcard character % is provided by a constant. This wildcard character is then concatenated with the parameter value with the help of the **concat** function. The advantage is that you do not have to type SQL wildcards in the command line if this mapping runs in another environment (e.g., MapForce Server).

Appearance of SQL/NoSQL-WHERE/ORDER components

SQL/NoSQL WHERE/ORDER components change their appearance depending on the settings defined in them. This way you can quickly view directly from the mapping what the **SQL/NoSQL WHERE/ORDER** component does (*see table below*).

	A WHERE clause has been defined.
	A WHERE clause with a parameter has been defined. The parameter name is visible under the `table/field` item.
	A WHERE clause with a parameter has been defined. Additionally, an ORDER BY clause has been defined. The sorting is indicated by the A-Z sort icon.

If you place the mouse cursor over the **SQL/NoSQL WHERE/ORDER** header, you will see a tooltip displaying the various clauses that have been defined.

Filtering data in Azure CosmosDB (Enterprise Edition)

If the name of a container in your CosmosDB has special characters (e.g., `+`, `-`, `@`, etc.) or is a keyword, this name will be unusable in queries, and you will get a syntax error from the database. To avoid potential issues, MapForce uses the following syntax for CosmosBD queries:

```
SELECT * FROM ROOT AS c WHERE c.<field> ORDER BY c.<field>
```

The `ROOT` keyword in the `FROM` clause references the current container you are querying. Since the `ROOT` keyword cannot be used to address fields in `WHERE` and `ORDER BY` clauses, the container is given the alias name `c` that is assigned to the container by means of the `AS` keyword. Assigning the alias name to the container makes it possible to filter and sort the container's contents.

Note that in the **WHERE/ORDER** dialog, all field references must be prefixed with the container alias name `c` (*see example below*).

```
SELECT * FROM ROOT AS c WHERE c.age > 20 ORDER BY c.name
```

5.6.3.1 Creating WHERE and ORDER BY Clauses

After an **SQL/NoSQL-WHERE/ORDER** component is added to the mapping, it needs a WHERE condition (clause) through which you specify how exactly you want to filter the data connected to the component. The WHERE condition must be entered in the dialog box **SQL/NoSQL-WHERE/ORDER Properties** (*see previous section*).

Writing a WHERE condition from MapForce is similar to writing the same SQL clause outside MapForce. Use the syntax applicable to the SQL dialect of the corresponding database. For example, you can use operators, wildcards, as well as sub-selects or aggregate functions. To create a parameter that you can pass from the mapping, enter a colon character (`:`) followed by the parameter's name.

Note: When you finish writing the WHERE clause and click OK, MapForce validates the integrity of the final SQL statement. A dialog box prompts you if there are syntax errors.

The table below lists some typical operators that can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal
<	Less than
>	Greater than
>=	Greater than/equal
<=	Less than/equal
`IN`	Retrieves a known value of a column

Operator	Description
`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 294.

Optionally, if you want to sort the retrieved recordset by a particular field, add an `ORDER BY` clause in the corresponding text box of the dialog box **SQL/NoSQL-WHERE/ORDER Properties**. To sort by multiple fields, separate the field names by commas. To change the sort order, use the `ASC` and `DESC` keywords. For example, the following `ORDER BY` clause retrieves records ordered by `lastname`, and then by `firstname`, in descending order:

```
lastname, firstname DESC
```

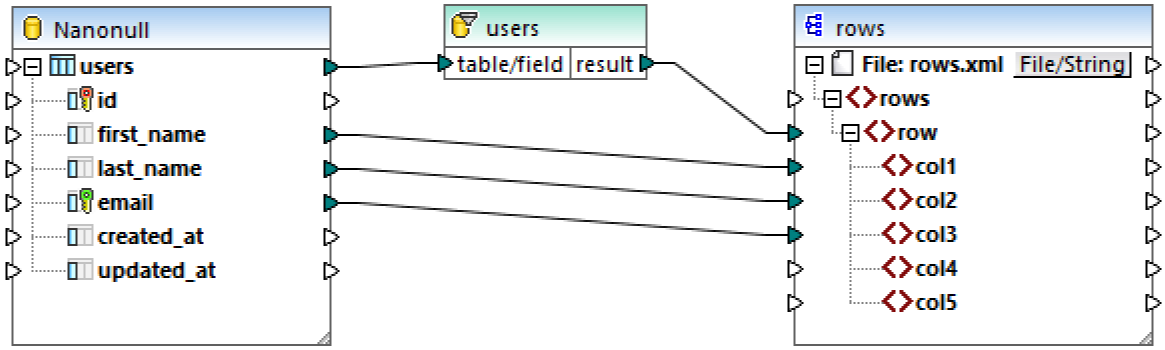
Example 1

The following WHERE condition is attached to the `users` table of the **`Nanonull.sqlite`** database component. It retrieves those records where `last_name` is greater than the letter M. In other words, it retrieves all names starting from user called Marzolla onwards.

```
last_name > "M"
```

Note how the connections are placed:

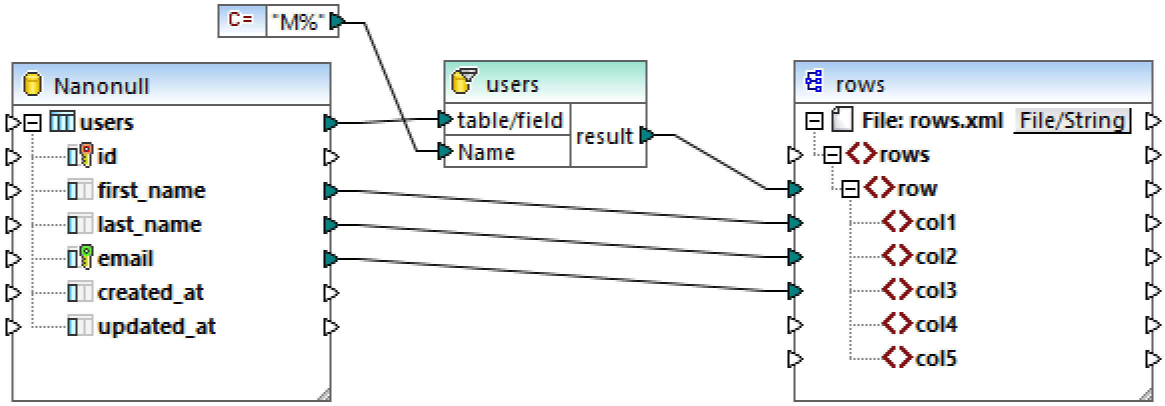
- The connection to `table/field` originates in the table that you want to query (`users` in this case).
- The `result` output is connected to a parent item of the fields that are queried/filtered (in this case the `row` item).



Example 2

The following WHERE condition creates a parameter `param` which then appears in the **SQL/NoSQL-WHERE/ORDER** component in the mapping.

```
last_name LIKE :param
```

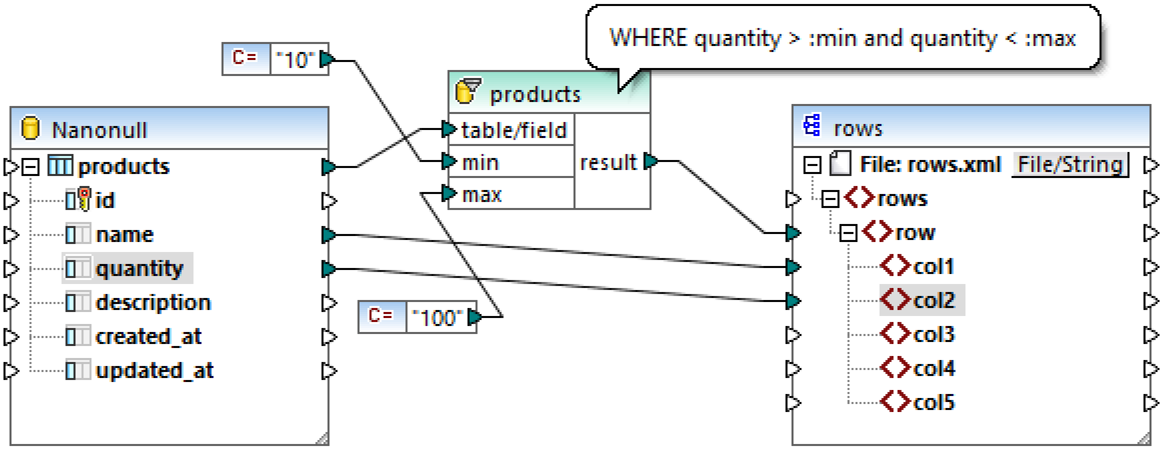


The constant component `%M` supplies the value of `param`. The wildcard % denotes any number of characters. This causes the mapping to search for a pattern in the column `last_name` (all last names starting with the letter M).

Example 3

The following WHERE condition creates two parameters, `min` and `max`, to which the current values of `quantity` are compared. The min and max values are supplied by two constant components from the mapping.

```
quantity > :min and quantity < :max
```



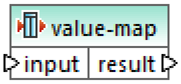
WHERE quantity > :min and quantity < :max

The WHERE condition in this example could also be written using the BETWEEN operator:

```
quantity BETWEEN :min and :max
```

5.7 Value-Maps

The Value-Map component (*screenshot below*) enables you to replace a value by another value with the help of a predefined look-up table. Such a component processes only one value at a time; therefore, it has one **input** and one **result** on the mapping.



A Value-Map is very useful when you would like to map individual items within two sets in order to replace items. For example, you could map the days of the week expressed as numbers (1, 2, 3, 4, 5, 6, and 7) to the name of each day of the week ("Monday", "Tuesday", and so on). Likewise, you could map the month names ("January", "February", "March", etc) to the numeric representation of each month (1, 2, 3, etc). At mapping run time, the matching values will be replaced according to your custom look-up table. The values in both sets can be of different type, but each set must store values of the same data type.

Value-Map components are suitable for simple look-ups, where each value in the first set corresponds to a single value in the second set. If a value is not found in the look-up table, you can either replace it with a custom value or an empty value, or pass it on as is. If you need to look up or filter values based on more complex criteria, use one of the filtering components 408 instead.

Importantly, when you generate code or compile a MapForce Server Execution file from the mapping, the look-up table data is embedded into the generated code or file. Consequently, defining a look-up table directly on the mapping is a good choice only if your data does not change frequently and is not very big (less than maybe a few hundred entries). If the look-up data changes regularly, you may find it difficult to maintain both the mapping and the generated code regularly—it is easier to maintain the look-up data as text, XML, database, or Excel.

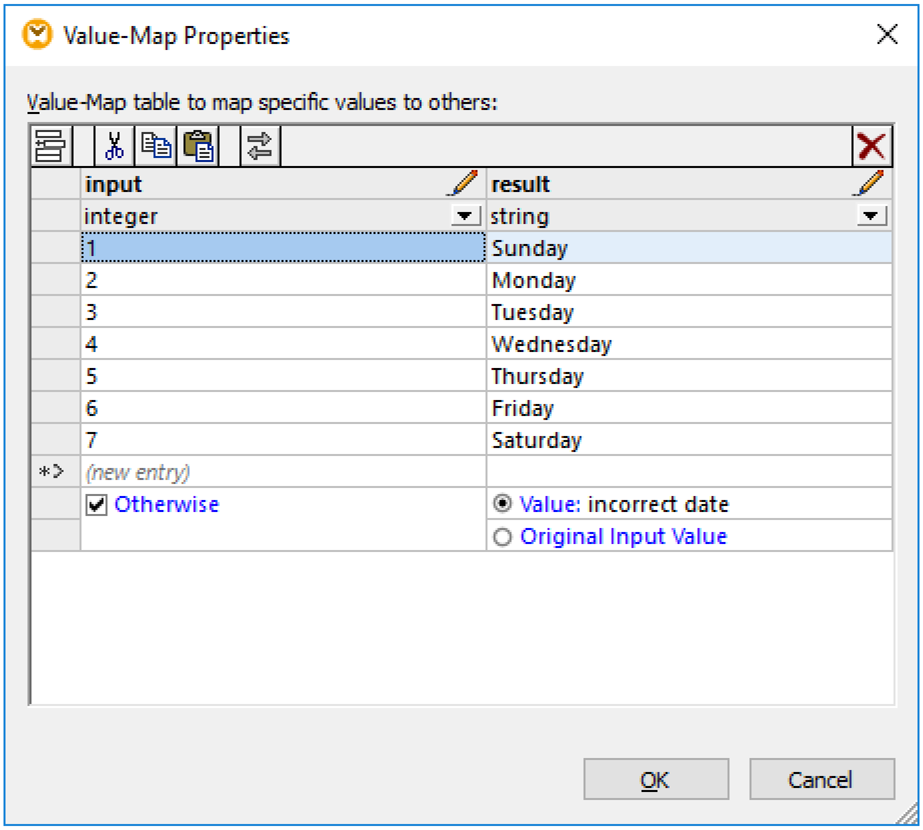
If the look-up table is huge, the mapping execution will be slowed down by the look-up table. In this case, it is recommended to use a database component with SQL-Where 413 instead. SQLite databases are good candidates for this, given their portability. On the server side, you can improve the performance of look-up tables by running a mapping with MapForce Server or MapForce Server Advanced Edition.

Create a Value-Map

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

- Click the **Insert Value-Map** toolbar button.
- On the **Insert** menu, click **Value-Map**.
- Right-click a connection, and select **Insert Value-Map** from the context menu.

This adds a new Value-Map component to the mapping. You can now start adding pairs of items to the look-up table. To do this, double-click the component's title bar or right-click it and select **Properties** from the context menu.



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	● Value: incorrect date
		○ Original Input Value

At mapping run time, MapForce checks each value that reaches the **input** of the Value-Map. If there is a matching value *in the left column* of the look-up table, then it replaces the original input value with the value *from the right column*. Otherwise, you can optionally configure it to return one of the following:

- A replacement value. In the example above, the replacement value is the text "incorrect date". You can also set the replacement value to be empty, by not entering any text at all.
- The original input value. This means that, if no match is found in the look-up table, the original input value will be passed further on to the mapping, unchanged.

> If you do not configure an "Otherwise" condition, the Value-Map returns an **empty node** whenever a match is not found. In this case, nothing will be passed to the target component and the output will contain missing fields. To prevent this from happening, you should either configure the "Otherwise" condition, or use the **`substitute-missing`** 589 function.

There is a difference between setting an empty replacement value and not specifying the "Otherwise" condition. In the first case, the field will be generated in the output, but it will have an empty value. In the latter case, the field (or XML element) enclosing the value will not be created at all. For more information, see Example: Replacing Job Titles 428.

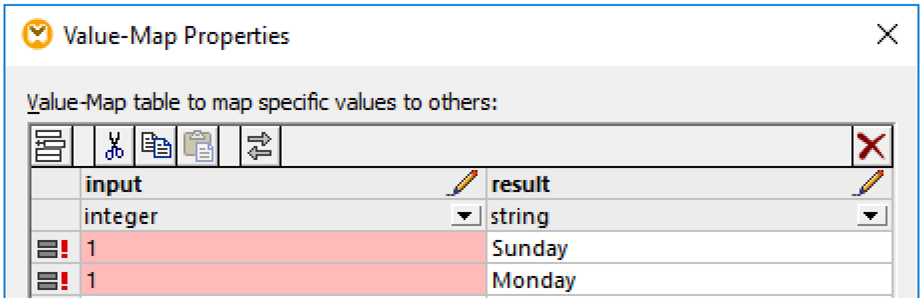
Populate a Value-Map

In a look-up table, you can define as many pairs of values as needed. You can enter the values manually, or copy-paste tabular data from text, CSV, or Excel files. Copy-pasting tables from an HTML page using a common browser will also work in most cases. You can also paste data from the database grid in the DB Query pane 277. If you copy data from text files, the fields must be separated by tab characters. In addition, MapForce will recognize text separated by commas or semicolons in most cases.

Keep in mind the following when creating look-up tables:

1. All items in the left column must be unique. Otherwise, it would not be possible to determine which item you want to match specifically.
2. Items that belong to the same column must be of the same data type. You can choose the data type from the drop-down list at the top of each column in the look-up table. If you need to convert Boolean types, enter the text "true" or "false" literally. For an illustration of this case, see Example: Replacing Weekdays 425.

If MapForce encounters invalid data in the look-up table, it displays an error message and highlights the invalid rows in pink color, for example:



Value-Map Properties

Value-Map table to map specific values to others:

input	result
integer	string
1	Sunday
1	Monday

To import data from an external source into the Value-Map component, take the steps below:

1. Select the cells of interest in the source program (for example, Excel). This can be either a single column of data or two adjacent columns.
2. Copy data to clipboard using the **Copy** command of the external program.
3. On the Value-Map component, click the row before which you would like to paste the data .
4. Click the **Paste table from clipboard** button on the Value-Map component. Alternatively, press **Ctrl+V** or **Shift+Insert**.

Note: The **Paste table from clipboard** button is enabled only if you have copied data from some source first (that is, if there is data on the clipboard).

When your clipboard data contains multiple columns, then only data from the first two columns are inserted into the look-up table; any other subsequent columns will be ignored. If you paste data from a single column on top of any existing values, a context menu appears, asking whether the clipboard data should be inserted as new rows or the existing rows should be overwritten. Therefore, if you need to overwrite existing values in the look-up table as opposed to inserting new rows, ensure that the clipboard contains only one column, not multiple.

To insert rows manually before an existing row, first click the row of interest, and then click the **Insert** button.

To move an existing row to some other position, drag the row to the new position (upwards or downwards) while holding the left mouse button pressed.

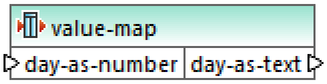
To copy or cut rows for subsequent pasting at some other position, first select the row, and then click the **Copy** button (or **Cut** button, respectively). You can also copy or cut multiple rows that are not necessarily consecutive. To select multiple rows, hold the **Ctrl** key pressed while clicking the rows. Note that the cut or copied text always contains values from both columns; you cannot cut or copy values from one column only.

To remove a row, click it, and then click the **Delete** button.

To swap the left and right columns, click the **Swap** button.

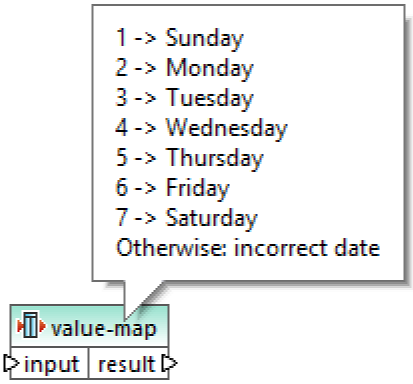
Rename Value-Map parameters

By default, the input parameter of a Value-Map component is called "input" and the output parameter is called "result". To make the mapping clearer, you can optionally rename any of these parameters by clicking the **Edit** button next to the respective name. The following is an example of a Value-Map with custom parameter names:



Preview a Value-Map

After you have finished creating a Value-Map, you can quickly preview its implementation directly from the mapping by holding the mouse over the component's title bar:

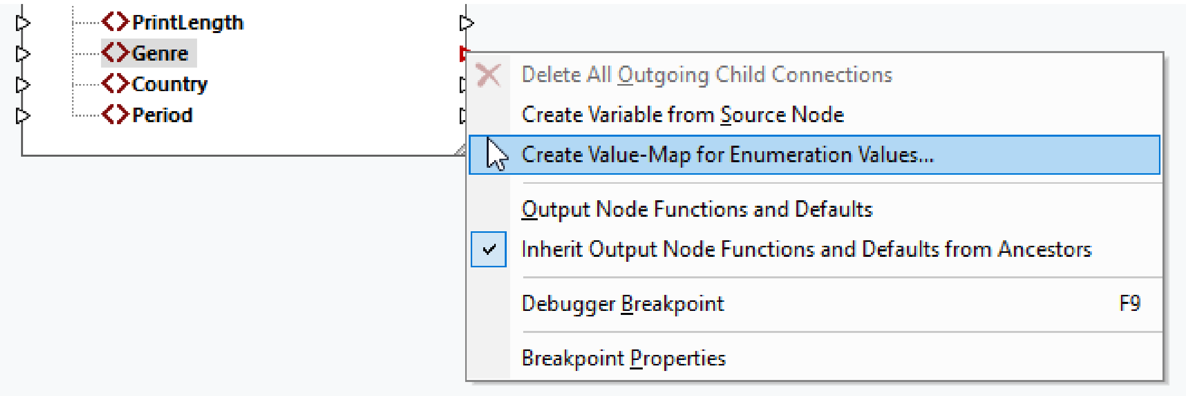

1 -> Sunday
2 -> Monday
3 -> Tuesday
4 -> Wednesday
5 -> Thursday
6 -> Friday
7 -> Saturday
Otherwise: incorrect date

Create a Value-Map from enumeration type

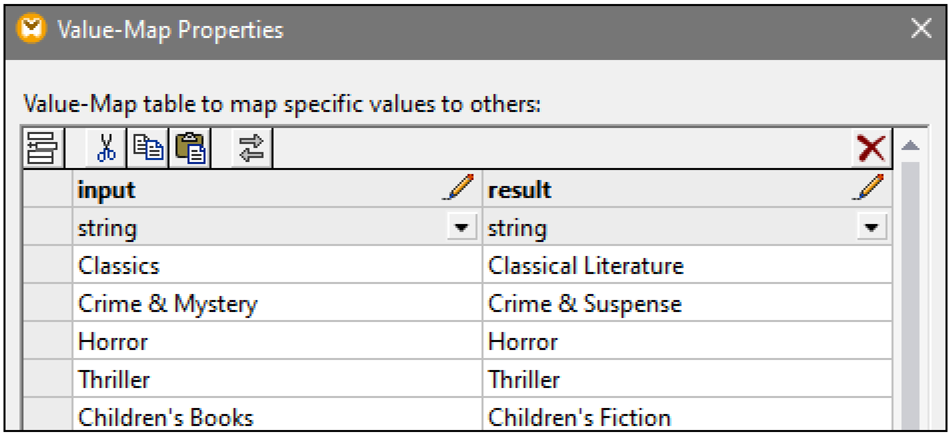
MapForce enables you to create a Value-Map from nodes with enumeration values. The feature is currently supported for XML components whose nodes have enumeration facets (*all editions*) and EDI components whose nodes have EDI codelists (*Enterprise Edition*). Depending on your needs, you can create such a Value-Map from a node's input or output connector.

To create a Value-Map from an enumeration type, follow the instructions below:

1. Depending on your goals, right-click the input or output connector of the node for the enumeration values of which you would like to create a Value-Map. In our example (*screenshot below*), we have selected the output connector of the `Genre` node.


PrintLength
Genre
Country
Period
Delete All Outgoing Child Connections
Create Variable from Source Node
Create Value-Map for Enumeration Values...
Output Node Functions and Defaults
Inherit Output Node Functions and Defaults from Ancestors
Debugger Breakpoint F9
Breakpoint Properties

2. Select **Create Value-Map for Enumeration Values** from the context menu.
3. The **Value-Map Properties** dialog appears. Both `input` and `result` parts of the Value-Map are pre-filled with the same enumeration values. You can now review and edit the values as necessary. The screenshot below shows the list of `Genre` values that the source XML file originally has (`input`) and the list of modified values we want to map (`result`).



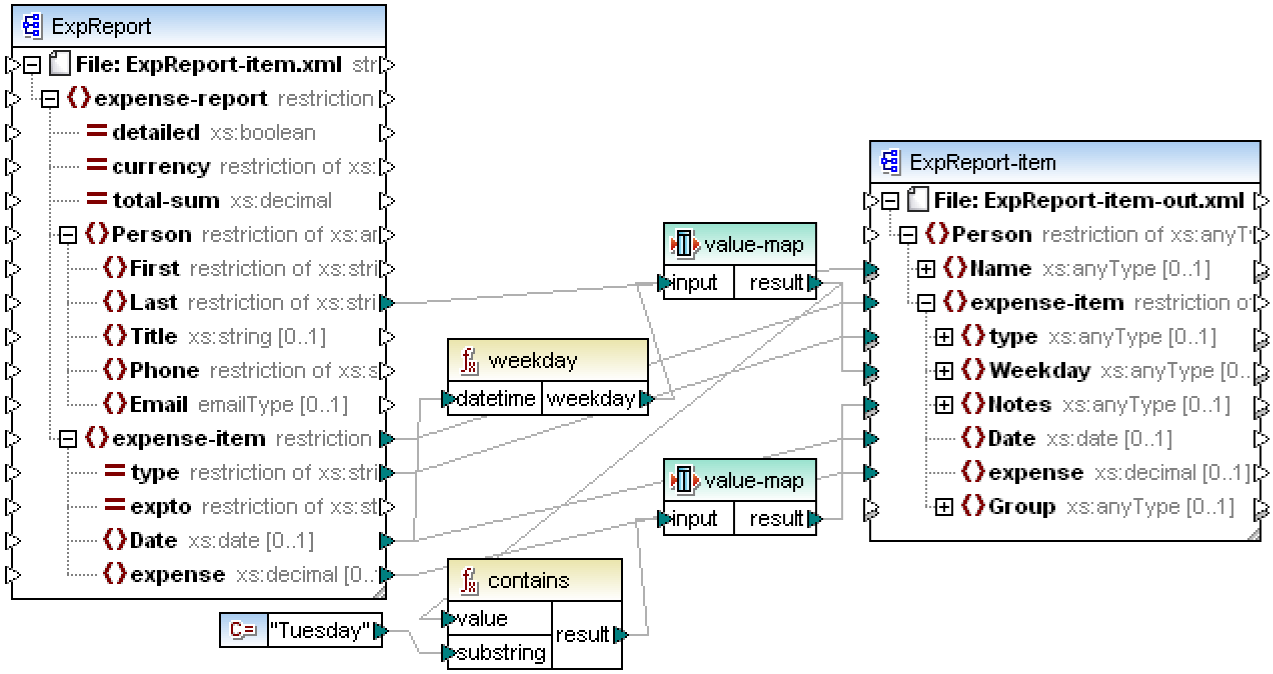
Value-Map table to map specific values to others:

input	result
string	string
Classics	Classical Literature
Crime & Mystery	Crime & Suspense
Horror	Horror
Thriller	Thriller
Children's Books	Children's Fiction

4. After reviewing the enumeration values, click **OK**. This action will add a Value-Map component to the mapping area. The Value-Map component will automatically be connected to the node from whose enumeration values the Value-Map was created.
5. Connect the other parameter of the Value-Map with the relevant node and continue designing your mapping as required.

5.7.1 Example: Replacing Weekdays

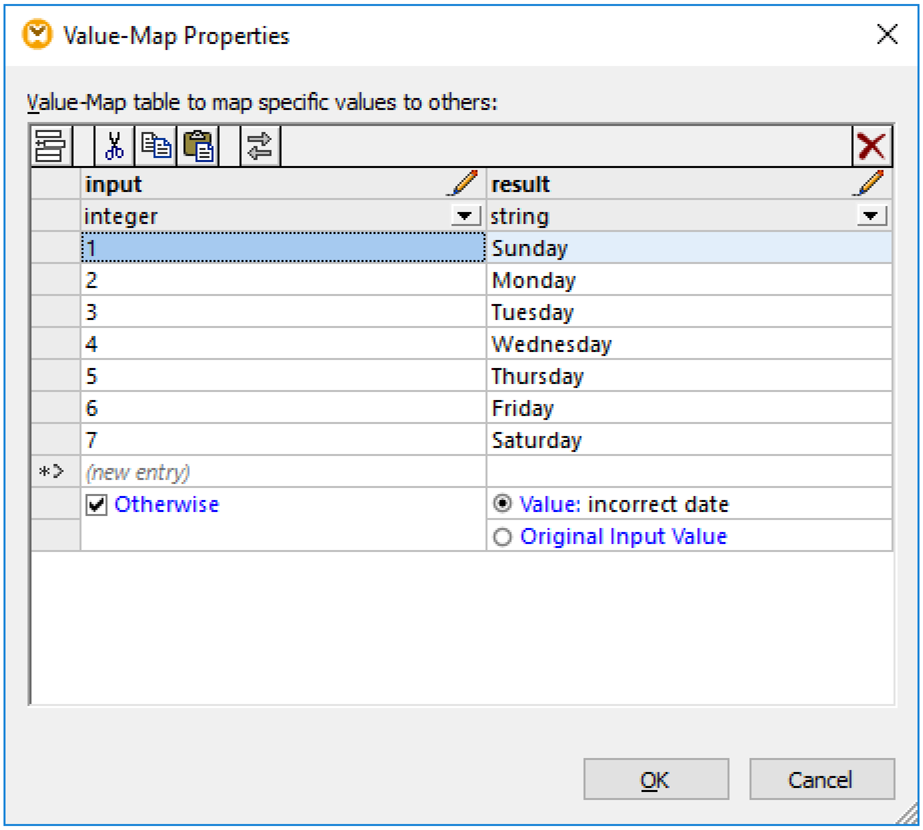
This example illustrates a Value-Map that replaces integer values with weekday names (1 = Sunday, 2 = Monday, and so on). This example is accompanied by a mapping which is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\Expense-valmap.mfd**.



Expense-valmap.mfd

This mapping extracts the day of the week from the **Date** item in the source file, converts the numerical value into text, and writes it to the **Weekday** item of the target component. More specifically, the following happens:

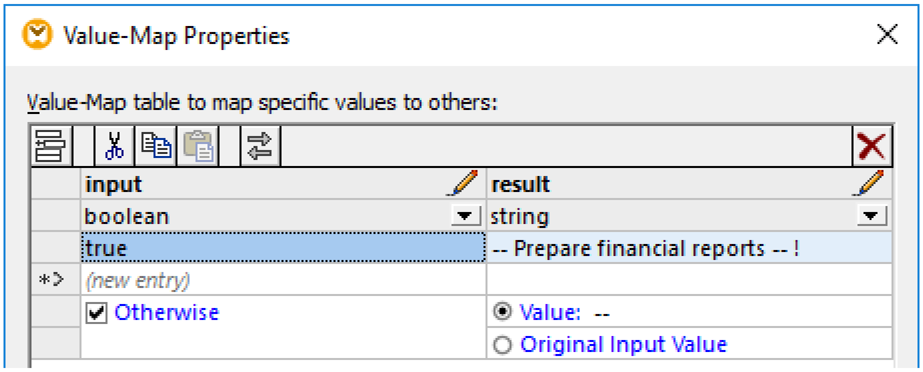
- The **weekday** function extracts the weekday number from the **Date** item in the source file. The result of this function are integers ranging from 1 to 7.
- The first Value-Map component transforms the integers into weekdays (1 = Sunday, 2 = Monday, and so on). If the component encounters an invalid integer outside of the 1-7 range, then it will return the text "incorrect date".



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	● Value: incorrect date
	○ Original Input Value

- If the weekday contains "Tuesday", then the text "Prepare Financial Reports" is written to the **Notes** item in the target component. This is achieved with the help of the `contains` function, which passes a Boolean **true** or **false** value to a second Value-Map component. The second Value-Map has the following configuration:



Value-Map table to map specific values to others:

input	result
boolean	string
true	-- Prepare financial reports -- !
(new entry)	
☑ Otherwise	● Value: --
	○ Original Input Value

The Value-Map illustrated above should be understood as follows:

- Whenever a Boolean **true** is encountered, convert it to the text "-- Prepare financial reports -- ! ". For all other cases, return the text "--".

Notice that the data type of the first column is set to "boolean". This ensures that the input Boolean value **true** is recognized as such.

5.7.2 Example: Replacing Job Titles

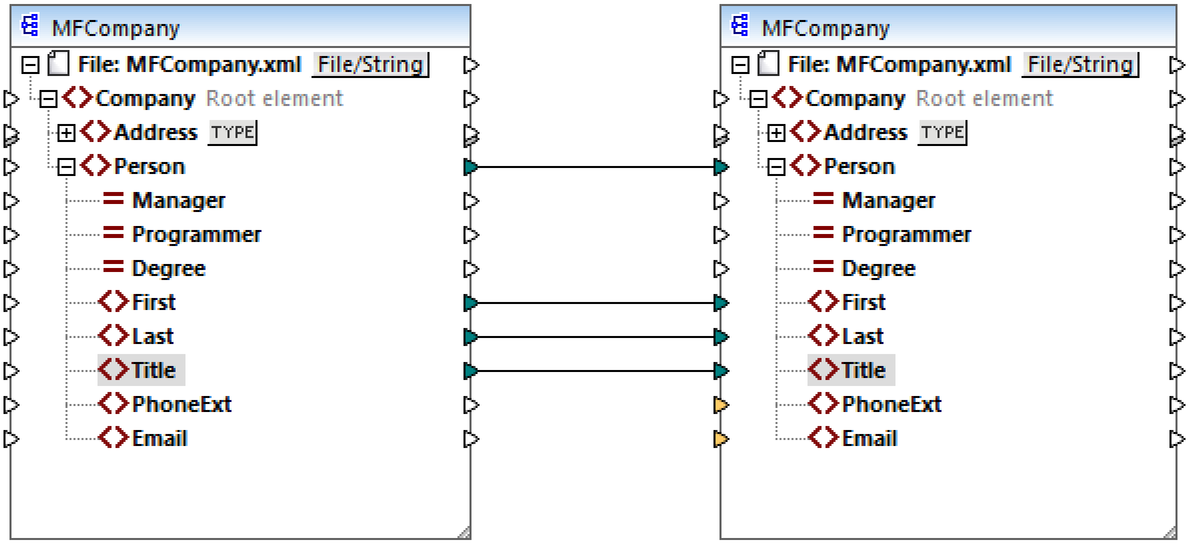
This example shows you how to replace values of specific elements in an XML file with the help of Value-Map components (that is, using a predefined look-up table).

The XML file required for this example is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\MFCompany.xml**. It stores, among other data, information about company employees and their job titles, for example:

```
<Person>
   <First>Michelle</First>
   <Last>Butler</Last>
   <Title>Software Engineer</Title>
</Person>
<Person>
   <First>Lui</First>
   <Last>King</Last>
   <Title>Support Engineer</Title>
</Person>
<Person>
   <First>Steve</First>
   <Last>Meier</Last>
   <Title>Office Manager</Title>
</Person>
```

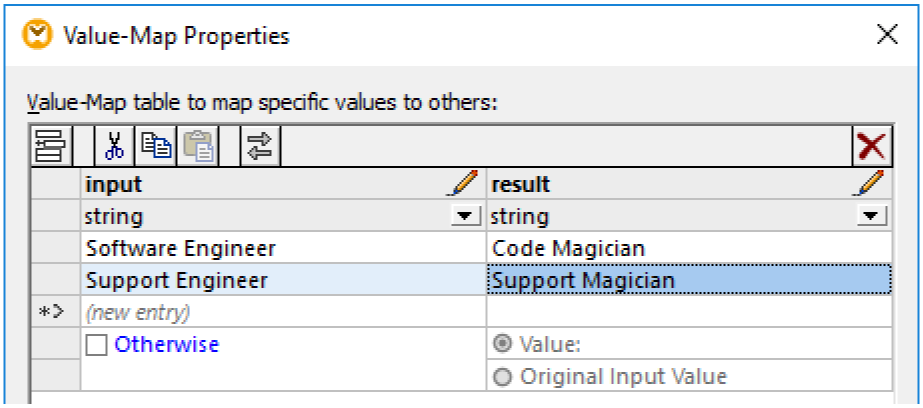
Let's assume that you need to replace some of the job titles in the XML file above. Specifically, the title "Software Engineer" must be replaced with "Code Magician". Also, the title "Support Engineer" must be replaced with "Support Magician". All the other job titles must remain unchanged.

To achieve the goal, add the XML file to the mapping area, by clicking the **Insert XML Schema/File** toolbar button or by running the **Insert | XML Schema/File** menu command. Next, copy-paste the XML component on the mapping and create the connections as shown below. Note that you might need to turn off the **Toggle auto-connect of children** toolbar option first, in order to prevent unnecessary connections from being created automatically.



The mapping created so far simply copies the **Person** elements to the target XML file, without making any changes to the **First**, **Last**, and **Title** elements.

To replace the required job titles, let's add a Value-Map component. Right-click the connection between the two **Title** elements, and select **Insert Value-Map** from the context menu. Set up the Value-Map properties as shown below:


Value-Map Properties

Value-Map table to map specific values to others:

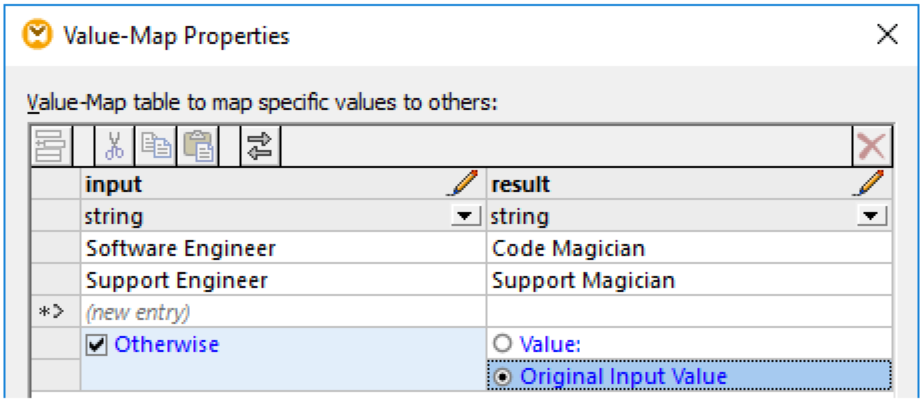
input	result
string	string
Software Engineer	Code Magician
Support Engineer	Support Magician
(new entry)	
□ Otherwise	● Value:
	○ Original Input Value

According to the setup above, each occurrence of "Software Engineer" will be replaced with "Code Magician", and each occurrence of "Support Engineer" will be replaced with "Support Magician". Notice that the **Otherwise** condition was not specified yet. For this reason, the Value-Map returns an *empty node* whenever the job title is other than "Software Engineer" and "Support Engineer". Consequently, if you click the **Output** pane and preview the mapping, some of the **Person** elements will have a missing a **Title**, for example:

```
<Person>
   <First>Vernon</First>
   <Last>Callaby</Last>
</Person>
<Person>
```

```
    <First>Frank</First>
    <Last>Further</Last>
</Person>
<Person>
    <First>Michelle</First>
    <Last>Butler</Last>
    <Title>Code Magician</Title>
</Person>
```

As stated before, empty nodes cause missing entries in the generated output; therefore, in the XML fragment above, only Michelle Butler had the title replaced, because her title was present in the look-up table. The configuration created so far still does not fulfill the original requirement. The correct setup is as follows:



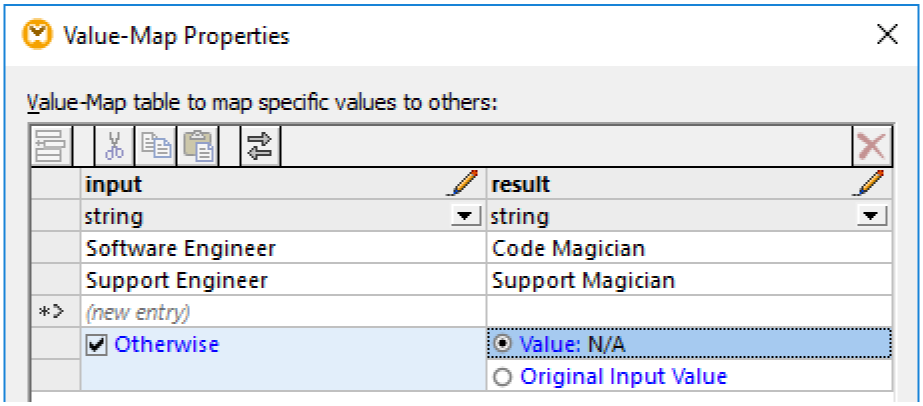
Value-Map table to map specific values to others:

	input	result
	string	string
	Software Engineer	Code Magician
	Support Engineer	Support Magician
*>	(new entry)	
	☑ Otherwise	○ Value:
		● Original Input Value

With the configuration above, the following happens at mapping run time:

- Each occurrence of "Software Engineer" will be replaced with "Code Magician"
- Each occurrence of "Support Engineer" will be replaced with "Support Magician"
- If the original title is not found in the look-up table, the Value-Map will return it unchanged.

For illustrative purposes only, we can also change all the job titles other than "Software Engineer" and "Support Engineer" to a custom value, for example "N/A". To achieve this, set the Value-Map properties as shown below:



Value-Map table to map specific values to others:

	input	result
	string	string
	Software Engineer	Code Magician
	Support Engineer	Support Magician
*>	(new entry)	
	☑ Otherwise	● Value: N/A
		○ Original Input Value

When you preview the mapping this time, each job title is present in the output, but those that were not matched have the "N/A" value, for example:

```
<Person>
   <First>Vernon</First>
   <Last>Callaby</Last>
   <Title>N/A</Title>
</Person>
<Person>
   <First>Frank</First>
   <Last>Further</Last>
   <Title>N/A</Title>
</Person>
<Person>
   <First>Michelle</First>
   <Last>Butler</Last>
   <Title>Code Magician</Title>
</Person>
```

This concludes the Value-Map example. By applying the logic above, you can now achieve the desired result in other mappings.

5.8 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[408]). For example, you may want to throw an exception if the value of some mapping item is greater than some custom threshold.

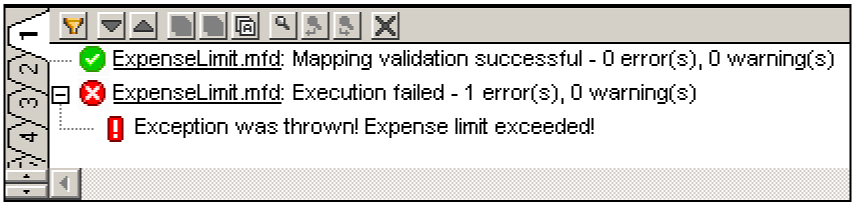
To add an exception to the mapping:

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:

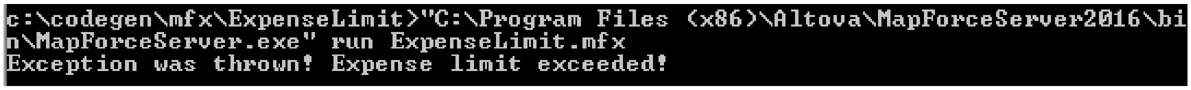
- In MapForce, the Messages window displays an error, and the exception text (in this case, "Expense limit exceeded").

  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!

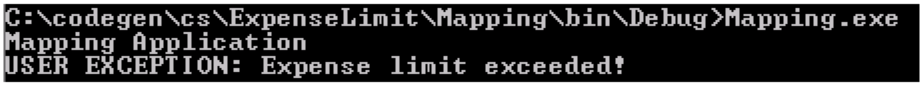
  If the mapping language is XSLT 2.0 or XQuery, an "Execution failed" error appears in the Messages window, and the respective XSLT2 or XQuery tab is opened. The error line is highlighted in the Messages window.

- 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!
  ```

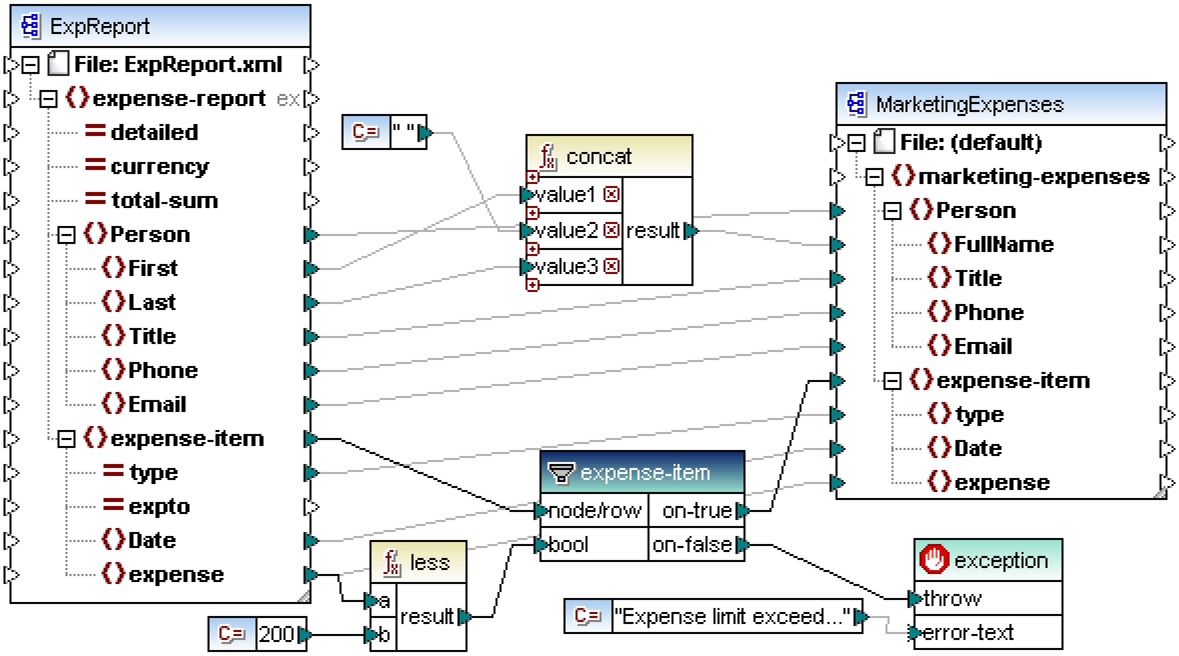
- If you run the mapping from the generated C#, C++, or Java code, the error "USER EXCEPTION" is returned, followed by the custom exception text you have defined in MapForce.

  ```
  C:\codegen\cs\ExpenseLimit\Mapping\bin\Debug>Mapping.exe
  Mapping Application
  USER EXCEPTION: Expense limit exceeded!
  ```

5.8.1 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\MapForce2025\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.

5.8.2 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.

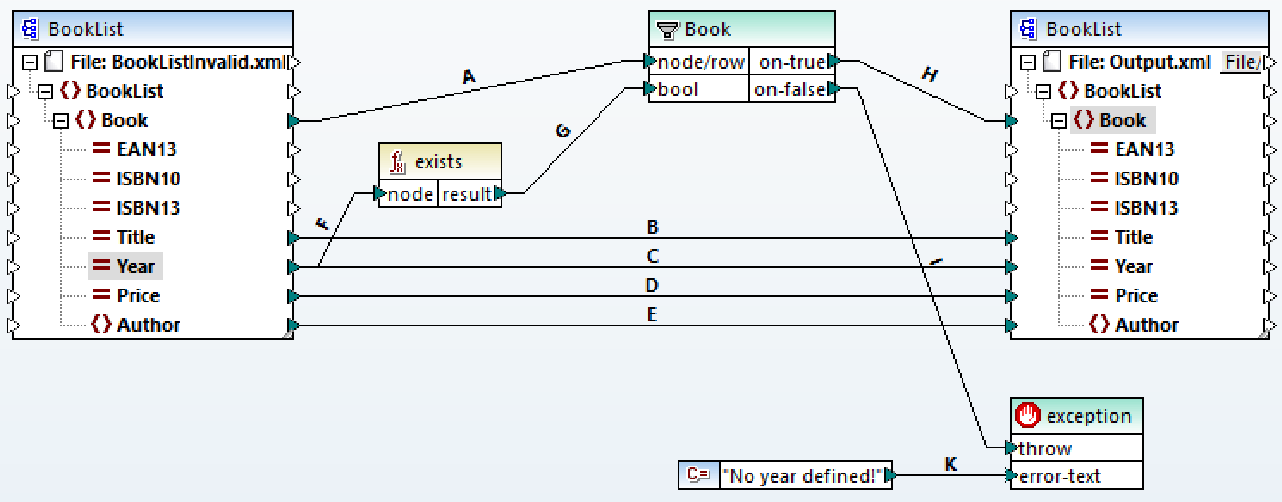
To add the source schema to the mapping:

1. On the **Insert** menu, click **XML Schema/File**, and browse for **<Documents>\Altova\MapForce2025\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 408)
- 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\MapForce2025\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.

5.9 Functions

In MapForce, you can use the following categories of functions to transform data:

- **MapForce built-in functions:** These functions are predefined in MapForce, and you can use them in your mappings to perform a wide range of processing tasks that involve strings, numbers, dates, and other types of data. You can also use them to perform grouping, aggregation, auto-numbering, and various other tasks. For reference to all available built-in functions, see Function Library Reference 511.

- **Node functions and defaults (*Professional and Enterprise editions*):** These are more specialized functions that let you create and apply custom processing logic to one or multiple descendant nodes in a mapping component. They enable you to process data either *before* it reaches a node of a mapping structure or *immediately after* it leaves a node. For more details, see Defaults and Node Functions 442.

- **User-defined functions (UDFs):** These are MapForce functions that you can create on your own, using various component kinds and built-in functions already available in MapForce (see User-Defined Functions 457).

- **Custom functions:** These are functions that you can import from external sources such as XSLT libraries, XQuery library modules, Java **`.class`** files, .NET **`.dll`** files, and adapt to MapForce. Note that, in order to be reusable in MapForce, your custom functions must return data of simple type (such as a string or integer), and they must also take parameters of simple type. For more information, see Importing Custom XSLT Functions 476, Importing Custom XQuery 1.0 Functions 483, and Importing Custom Java and .NET Libraries 487.

- **Web service calls (*Enterprise Edition*):** MapForce enables you to make a call to a previously defined Web service (a WSDL-based service or a generic HTTP API).

Note: You can import custom external libraries of functions either directly (no configuration required) or by configuring a MFF (MapForce Function File) recognized by MapForce. If you use the latter approach, you can also import C++ libraries, in addition to Java classes and .NET assemblies. Note that libraries imported via .mff files must meet the prerequisites mentioned in Referencing Java, C# and C++ Libraries Manually 494.

5.9.1 Functions Basics

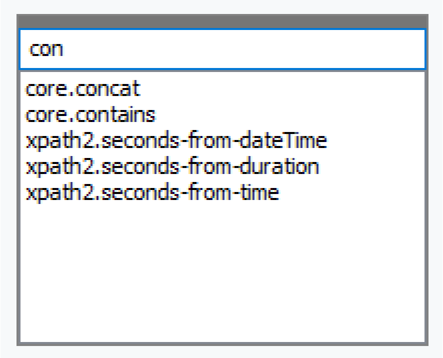
The subsections below give an overview of basic function-related actions. The functions you see in the Libraries window depend on the transformation language you have selected. For more information, see Transformation Languages 19.

Add a function

MapForce includes a large number of built-in functions that you can add to a mapping. For more information about all available built-in functions, see Function Library Reference 511. To add a function to a mapping, you can choose one of the following methods:

- Press and hold the required function in the Libraries window and drag it to the mapping area. To filter functions by name, start typing the function name in the text box at the bottom of the window.

- Double-click anywhere in the empty area of the mapping and start typing the function name (*see screenshot below*). To see a tooltip with more details about a function, click this function in the list. To add a function to your mapping, double-click the relevant function in the combo box.



con

core.concat
core.contains
xpath2.seconds-from-dateTime
xpath2.seconds-from-duration
xpath2.seconds-from-time

Add a UDF
You can also add user-defined functions (UDFs) to your mapping using the same approaches as described above if (i) a UDF has already been created in the same mapping or (ii) you have imported a mapping that contains a UDF as a local or global library.

Add a constant

Constants enable you to supply custom text and numbers to a mapping. To add a constant to your mapping, you can choose one of the following options:

- Right-click anywhere in the empty mapping area and select **Insert Constant** from the context menu. Enter the value and select one of the data types: *String*, *Number*, or *All other*.
- Select the **Insert | Constant** menu command. Enter the value and select one of the data types: *String*, *Number*, or *All other*.
- Click the **Constant** toolbar command. Enter the value and select one of the data types: *String*, *Number*, or *All other*.
- Double-click anywhere in the empty mapping area. Type the double quotation mark followed by the constant value. The closing double quotation mark is optional. To add a numeric constant, just type the number.

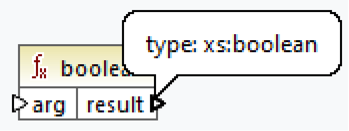
Search for a function

To search for a function in the **Libraries** window, start typing the function name in the text field at the bottom of the window. By default, MapForce searches by function name and description text. If you want to exclude the function description from the search, click the down-arrow and disable the *Search in function descriptions* option. To cancel the search, press the **Esc** key or click ✖.

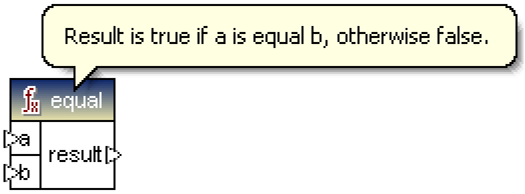
To find all occurrences of a function within the currently active mapping, right-click the function name in the **Libraries** window and select **Find All Calls** from the context menu. The search results are displayed in the **Messages** window.

View a function's type and description

To view the data type of a function's input/output argument, move your mouse over the argument part of a function (*see screenshot below*). Make sure that the INFO (**Show tips**) toolbar button is enabled.

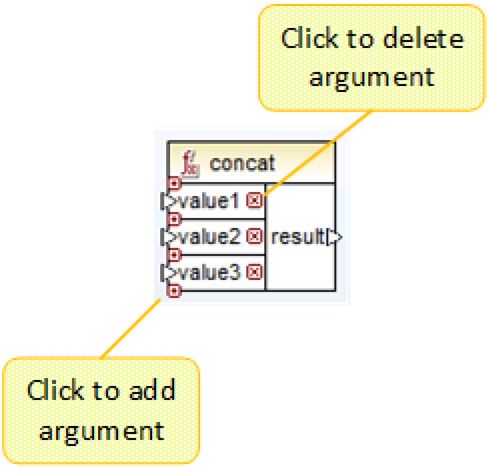


To view the description of a function, move the mouse over the function's header (*see screenshot below*). Make sure that the INFO (**Show tips**) toolbar button is enabled.


Result is true if a is equal b, otherwise false.

Add/delete function arguments

For some MapForce built-in functions, it is possible to add as many parameters as you need for your mapping purposes. One such example is the **concat** 592 function. To add or delete function arguments (for functions that support that), click **Add parameter** (⊡) or **Delete parameter** (☒) next to the parameter you want to add or delete, respectively (*see below*). Moving a connection to the ⊡ symbol adds another parameter and connects this parameter.


Click to delete argument

Click to add argument

5.9.2 Manage Function Libraries

In MapForce, you can import and use the following kinds of libraries in a mapping:

- Any mapping design files (*.mfd) that contain user-defined functions (UDFs). This specifically refers to mapping files that contain UDFs created with MapForce, using the MapForce built-in functions and components as building blocks. For further information, see Creating User-Defined Functions 458.

- Custom XSLT files that contain functions. This refers to XSLT functions written outside of MapForce that qualify for import into MapForce as described in Importing Custom XSLT Functions 476.
- Custom XQuery 1.0 files that contain functions. This refers to XQuery functions written outside of MapForce that qualify for import into MapForce as described in Importing Custom XQuery 1.0 Functions 483.
- Java .class files and .NET .dll libraries which qualify for import into MapForce as described in Importing Custom Java and .NET Libraries 487.

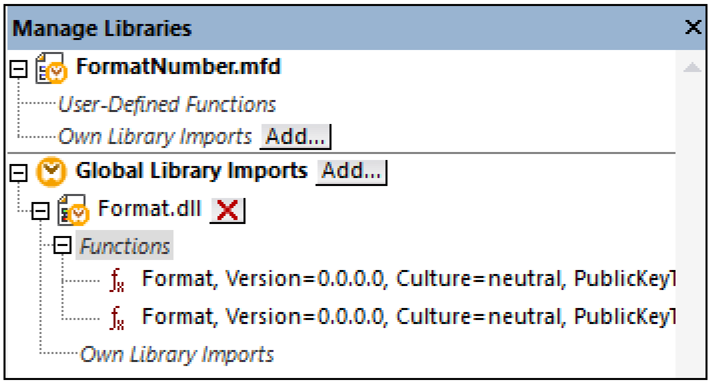
Note: You can import custom external libraries of functions either directly (no configuration required) or by configuring a MFF (MapForce Function File) recognized by MapForce. If you use the latter approach, you can also import C++ libraries, in addition to Java classes and .NET assemblies. Note that libraries imported via .mff files must meet the prerequisites mentioned in Referencing Java, C# and C++ Libraries Manually 494.

Manage Libraries window

You can view and manage all libraries used by a mapping file from the Manage Libraries window. This includes UDFs and custom libraries.

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

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



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

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

Data displayed in the Manage Libraries window is organized as a tree hierarchy as follows:

- Any currently open mapping documents are displayed as top-level entries. Each entry has two branches: **User-Defined Functions** and **Own Library Imports**.
  - o The **User-Defined Functions** branch displays any UDFs contained in that document.
  - o The **Own Library Imports** branch displays libraries imported *locally* into the current mapping document. The term "libraries" means other mapping documents (.mfd files that contain user-defined functions) or custom external libraries written in XSLT 1.0, XSLT 2.0, XQuery 1.0*, Java*, C#*, or .mff files mentioned previously. Note that the **Own Library Imports** structure could be

several levels deep, since any mapping document may import any other mapping document as a library.

- The **Global Library Imports** entry encloses any custom libraries that you have imported *globally* at application level. Again, in case of .mfd files, the structure could be several levels deep, for the reasons mentioned above.

** These languages are supported only in MapForce Professional or Enterprise edition.*

Note: The XSLT, XQuery, C#, and Java libraries may have dependencies of their own. Such dependencies are not displayed in the Libraries window.

Context menu commands

You can quickly perform various operations against objects in the Manage Libraries window by right-clicking an object and selecting one of the following context menu options:

Command	Description	Applicable for
Open	Opens the mapping.	Mappings
Add	Opens a dialog box where you can browse for a custom library of functions.	Own Library Imports
Locate Function in Libraries Window	Changes focus to the Libraries window, and selects the function.	Functions
Cut, Copy, Delete	These standard Windows commands are applicable only to MapForce user-defined functions. You cannot copy-paste functions from external XSLT files or other library kinds.	User-defined functions
Paste	Lets you paste a user-defined function that was previously copied to clipboard into the current library.	Libraries (UDF)
Options	Opens a dialog box where you can set or change options for the current library.	Libraries
Show All Open Documents	When this option is switched on, the Manage Libraries window will display all currently open mappings. This is typically useful if you need to copy-paste functions between mappings. Otherwise, only the mapping that is currently in focus is shown.	Always
Show File Paths	When this option is switched on, objects in the Manage Libraries window are displayed with their full path. Otherwise, only the object name is shown.	Always

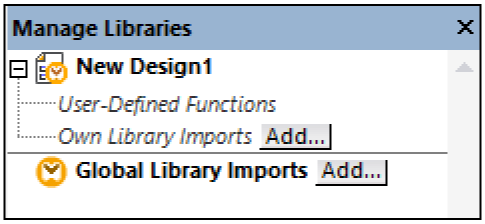
5.9.2.1 Local and Global Libraries

You can import libraries *locally* or *globally*. Global imports are at application level. If a library was imported globally, you can use its functions from any mapping.

Local imports are at mapping file level. For example, let's suppose that, while working on mapping **A.mfd**, you decide to import all user-defined functions from mapping **B.mfd**. In this case, mapping **B.mfd** is considered to be imported as a local library into **A.mfd** and you can use functions from **B.mfd** in **A.mfd** as well. Likewise, if you import functions from an XSLT file into **A.mfd**, this is also a local import.

You can view and manage all local and global imports from the Manage Libraries window. To import a library, do one of the following:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[23]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

Conflicting function names

You may come across situations where the same function name is defined at any of the following levels:

- in the main mapping
- in a library that was imported locally
- in a library that was imported globally

When it encounters such cases, MapForce will attempt to call the function exactly in the order above, to prevent ambiguity. That is, the function defined directly in the mapping takes precedence if the same function name exists in a locally imported library. Also, the function imported locally takes precedence over the function imported globally (assuming that both functions have the same name).

If multiple functions with the same name exist, only the "winning" function will be called, according to the rule above; any other ambiguous function names will be blocked. Such blocked functions appear as grayed out in the Libraries window, and it is not be possible to use them in the mapping.

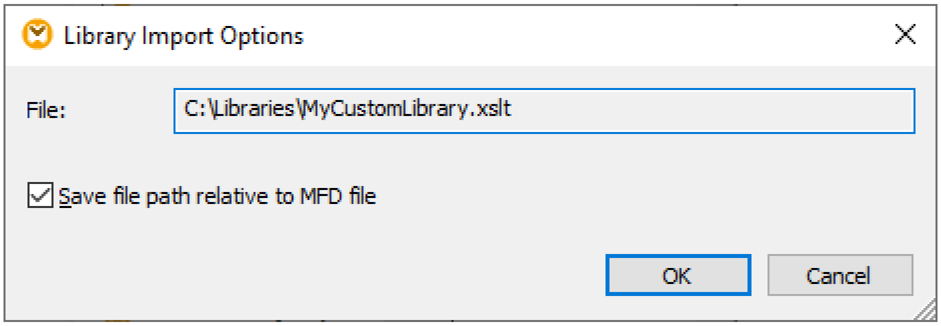
5.9.2.2 Relative Library Paths

You can set the path of any imported library file to be relative to the mapping design file (.mfd), provided that the library was imported locally (not globally), as described in Local and Global Libraries[441].

Setting a relative library path is applicable only for those libraries that were imported *locally* at document level. If a mapping was imported *globally* at program level, its path is always absolute.

To set a library path as relative to the mapping design file:

1. Click **Add/Remove Libraries** at the base of the Libraries window. The Manage Libraries window[25] opens.
2. Click **Options** next to the library of interest. (Alternatively, right-click the library, and select **Options** from the context menu.)



Library Import Options

File: C:\Libraries\MyCustomLibrary.xslt

☑ Save file path relative to MFD file

3. Select the **Save file path as relative to MFD file** check box.

Note: If the check box is grayed out, make sure that the library was indeed imported locally, and not globally.

When the check box is selected, MapForce will keep track and update the path to any referenced library files when you save the mapping file to a new directory using the **Save as** menu command. Also, if the library files are in the same directory as the mapping file, the path reference will not be broken when you move the entire directory to a new location on the disk, see also Using Relative Paths on a Component[43].

Note that the **Save file path as relative to MFD file** check box specifies that paths are *relative to the mapping file*, and it does not affect paths in generated code. For information about how library references are handled in generated code, see Paths in Various Execution Environments[46].

5.9.3 Defaults and Node Functions

Defaults and node functions are particularly useful when you want to apply the same processing logic to multiple descendant items in a structure. Usually you would need to copy-paste the same function multiple times in the mapping. This could clutter the mapping and make it more difficult to understand. Defaults and node functions can apply to a single item or to multiple items at once. Defaults replace empty sequences. If the connection transports a value, the default will be ignored.

Defaults and node functions are suitable for most components that have a tree with nodes (e.g., XML, EDI, Join components, variables).

Defaults and node functions are compatible only with the Built-In transformation language. Running such mappings from generated C#, C++, Java program code or with generated XSLT/XQuery transformations is not supported. On the server side, you can execute such mappings with MapForce Server Advanced Edition.

Advantages of node functions and defaults

Creating node functions and defaults means defining a *rule*. Rules have the following important characteristics that make them flexible and easy to use:

- *Inheritance.* When you define a rule on an item that has descendants, the rule will be inherited by descendants by default unless you choose to disable this option. If the item for which you define the function has multiple levels of child items nested under it, you can choose to apply the rule only to direct child items or to all descendant items.
- *Filtering.* MapForce applies rules conditionally, based on the data type of each item. This makes it possible, for example, to apply a certain default value or function to all items of type `string` and a different default or function to all items of type `decimal`. For details, see *Scenario 2* in Use-Case Scenarios 447. You can also define more advanced filtering options: For example, you can specify a data type that your function must match (this could be a category of data types such as `numeric`) and then filter the nodes of this data type based on the node name or node type (e.g., `integer`). For details, see *Scenario 5* 452.

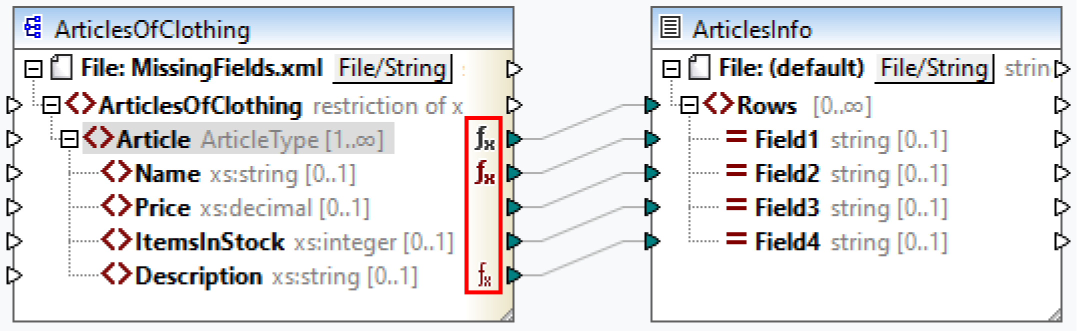
For example, you can instruct MapForce to do the following:

- Every time an empty or null value is encountered, replace it with some other value and do this recursively for all descendant items.
- Every time a specific value is encountered, replace it with some other value (or with an empty string) and do this recursively for all descendant items.
- Replace all database null values with empty strings or custom text.
- Append a custom prefix or suffix to all values that are written to a target file or database.

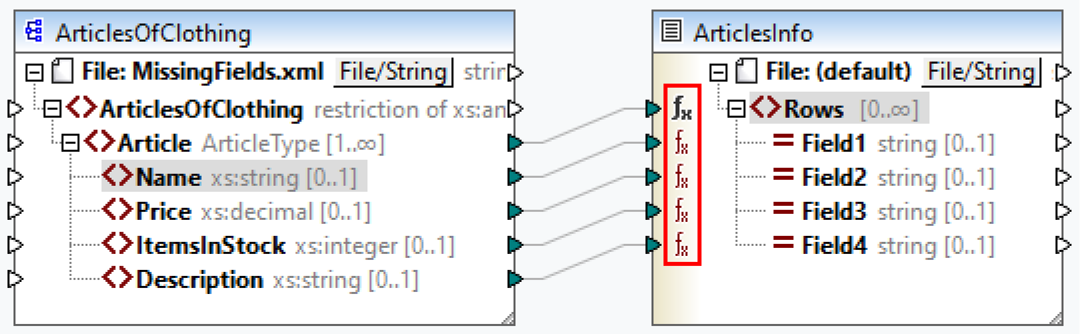
Output vs. input side

You can define node functions and defaults on the input side, output side, or on both sides of a component, depending on your needs. In MapForce, a mapping works in the following way: (i) it first reads data from a source component (e.g., an XML file), (ii) then optionally processes the data in some way (e.g., using a function), and (iii) finally writes data to some target component (e.g., a database). Considering this basic principle, you can set node functions and defaults at various stages:

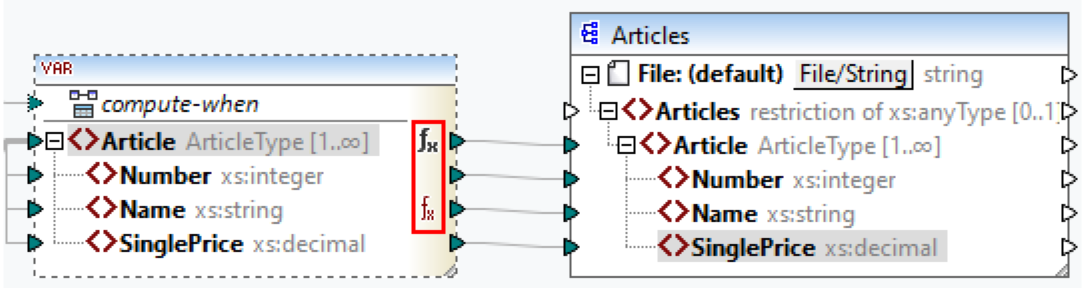
- Immediately *after* data is read from the source, but before it is further processed by your mapping, which means the function/default is defined on the output side of the source component (*see example below*).


ArticlesOfClothing
File: MissingFields.xml File/String
ArticlesOfClothing restriction of x
Article ArticleType [1..∞]
Name xs:string [0..1]
Price xs:decimal [0..1]
ItemsInStock xs:integer [0..1]
Description xs:string [0..1]
ArticlesInfo
File: (default) File/String strin
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

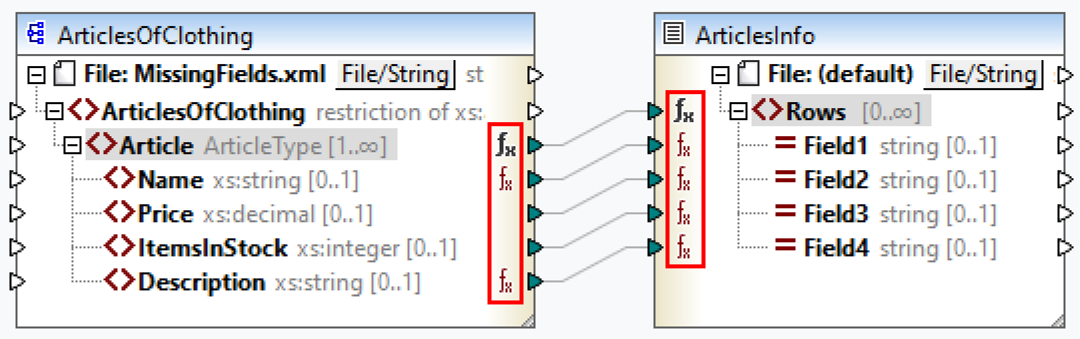
- Immediately *before* data is written to the target component (and after it has finished all intermediary processing), which means the function/default is defined on the input side of the target component (*see example below*).


ArticlesOfClothing
File: MissingFields.xml File/String string
ArticlesOfClothing restriction of xs:an
Article ArticleType [1..∞]
Name xs:string [0..1]
Price xs:decimal [0..1]
ItemsInStock xs:integer [0..1]
Description xs:string [0..1]
ArticlesInfo
File: (default) File/String
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

- At an intermediary stage in the mapping process. For example, if the mapping contains an intermediary variable of complex type (e.g., an XML structure), you can trim all values before they are supplied to the XML structure or immediately after they are returned by the XML structure (*see example below*).



- On the output side of a source component and on the input side of a target component. In the example below, a default has been defined for all the nodes of type `string` in the `ArticlesOfClothing` component. In the `ArticlesInfo` component, we have defined a node function that will transform all values of string nodes into uppercase characters.


ArticlesOfClothing
File: MissingFields.xml File/String st
ArticlesOfClothing restriction of xs:
Article ArticleType [1..∞]
Name xs:string [0..1]
Price xs:decimal [0..1]
ItemsInStock xs:integer [0..1]
Description xs:string [0..1]
ArticlesInfo
File: (default) File/String
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

In this section

This section explains how to configure a rule, describes real-life scenarios in which defaults and node functions can be useful, and shows how to add node metadata to node functions. The section is organized into the following topics:

- Rule Configuration 445
- Use-Case Scenarios 447
- Node Metadata in Node Functions 454

5.9.3.1 Rule Configuration

You can create node functions and defaults for almost any item in the mapping. Creating node functions and defaults means defining a *rule*. To create a rule, select an item (node or field) for which you want to define a rule. This can be a single node or a node with child nodes. If you choose to create a rule for a node with children, the rule will apply to all the children unless you disable this option explicitly.

Important aspects of defaults and node functions

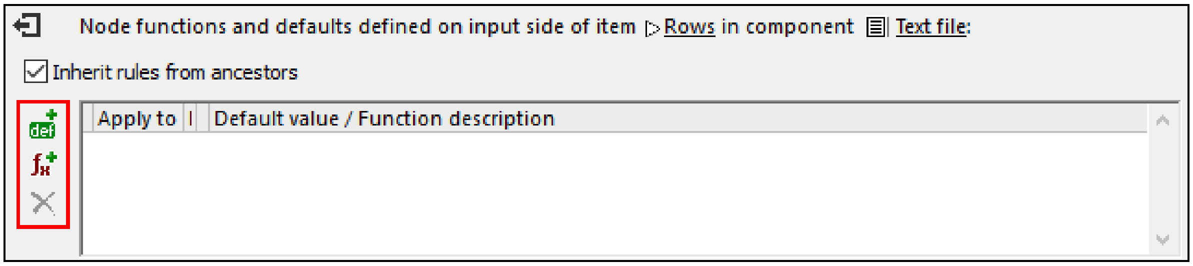
Defaults and node functions have the following important aspects:

- You can create defaults and node functions on the input side of a target component or on the output side of a source component. To establish which side is right for your needs, see Input vs. Output Side 443.
- When multiple rules exist for one and the same item, MapForce will apply the rule that is closer to that item. To find out how to override rules, see *Scenario 4* in Use-Case Scenarios 451.
- Defaults and node functions can be created if the connection type between the source and target is source-driven or target-driven. However, copy-all connections are not supported. Specifically, node functions and defaults do not apply to descendants of copy-all connections. The parent node with the copy-all connection can have node functions and defaults but only if this parent node has a simple value, for example, an XML element with simple-type content and attributes. For more information about connection types, see Connection Types 51.
- Creating defaults and node functions is not supported for the `File` node.
- Inside a node function, only certain components meaningful in this context are supported (e.g., built-in functions, variables, if-else conditions). Complex structures such as XML, JSON, EDI, or databases are not supported. Adding inline user-defined functions and join components to a node function is not supported.
- A node function can have one input parameter or no parameter at all. The input parameter is always called `raw_value`. Do not delete the input parameter even if you do not need an input for your function; otherwise, validation errors will occur when you run the mapping. The same applies to the function's output. Should you need to restore an accidentally deleted input component, run the menu command **Function | Insert Input**.
- MapForce allows you to supply node metadata (e.g., a node name, an annotation) to a node function. For details, see Node Metadata in Node Functions 454.
- The **Filter Node Functions and Defaults** dialog box allows you to choose from an extensive range of data types, some of which are categories of types. This means that they will match a larger selection of types. For example, the type `string` matches various other data types derived from `string`, such as `normalizedString`, `token`, `NCName`, `NMTOKEN`, `IDREF`, `ENTITY`, and others. Likewise, the type `decimal` will match the derived types `integer`, `long`, `short`, and others. The hierarchy of types is specified in the XML Schema W3C Recommendation.

Create a rule

To create a rule, take the following steps:

1. Right-click the node of interest and select **Node Functions and Defaults | Input/Output Node Functions and Defaults** from the context menu. Whether it is *Input* or *Output Node Functions and Defaults* depends on the side of the component on which you need to create a node function or a default. Alternatively, right-click a connector and select the node function command for that side. The **Mapping** pane will display the Node Functions window, in which you can define defaults and node functions (*red rectangle in screenshot below*).


Node functions and defaults defined on input side of item ▷ Rows in component Text file:
☑ Inherit rules from ancestors
Apply to
Default value / Function description
def
fx

   If a parent node has rules defined for it, and you want to transfer these rules from the parent to a child, select the *Inherit rules from ancestors* check box (*screenshot above*). For more information about inheritance, see Use-Case Scenarios 447.

2. The next step is to define whether you want to add a default (def) or a function (fx). As soon as you click the relevant option, a new rule will be created (a row in the grid in the Node Functions window). For information about rule configuration, see *Rule Configuration* below. If you are defining a function, the mapping area will display the function's input and output parameters.

Edit/delete rules

To view, modify, or delete a rule, click the fx icon (black or red) next to a node of interest. The Node Functions window will display all the rules defined for this node. If you have added a node function, you will see the Edit button in the grid, which allows you to change the implementation of the function. If the Edit button is not present, the function is most likely defined on some ancestor. In this case, click the fx icon near the item for which the rule has been defined.

To delete a rule, select it from the grid in the Node Functions window and then click the × button.

Configure a rule

As soon as you select **Node Functions and Defaults | Input/Output Node Functions and Defaults** from the context menu (*see instructions above*), you will see the Node Functions window in which you can configure node functions and defaults. The available settings are described below.

- *Apply to:* A rule can apply to the current item, its direct child items, or to all descendant items. If the item you have selected has no descendants, only the *Current item* option will be available.

- *Data type:* Click the ... button and select a data type from the dialog box. The rule will apply only to items of this data type (or a derived data type). If the item you have selected has no descendants, the item's data type is the only choice.

- *Default Value/Function Description:* If you are defining a default, type the default value that you wish to set for the selected item (and all descendants, if applicable). To set an empty string as a default, leave this field empty. If you are defining a function, this field is for information only: It displays a summary of the function.

To exit the Node Functions window, click the ⇦ button in the upper-left corner of this window or press **Escape**.

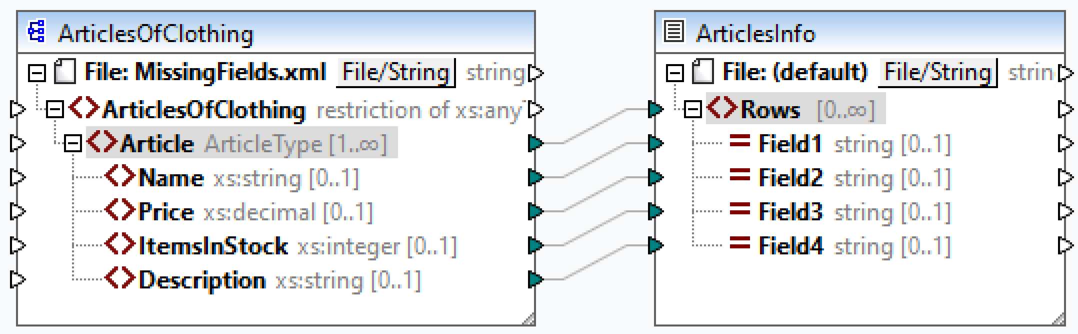
Visual clues

MapForce displays different visual clues to help you understand which rules are defined and whether they apply to a specific node (*see table below*).

Icon	Description
fx	This icon indicates that a rule has been defined for this item and may affect all its descendants. Click the icon to modify or delete the rule.
fx	This icon indicates that the item inherits the rule defined at ancestor level.
fx	This icon indicates that a rule is defined for and applies to this item. This icon usually appears when a function or a default is defined for a single node.
fx	This icon indicates that even though a rule applies to this item, it is deliberately blocked. This icon is displayed only if inheritance is blocked and no other rules are defined for this node. If a rule from an ancestor does apply, the fx icon has priority.
fx	This icon indicates that a rule defined for this item is inactive. For example, this icon may appear for child items that are not connected yet.
fx⚠	This icon indicates that the rule is currently inactive, because no matching and connected nodes have been identified. By *matching* the following is meant: Before applying a function or a default, MapForce looks for the data type you have defined; if you have also specified a filter that must match a specific node name, for example, MapForce will also look for the relevant node name.

5.9.3.2 Use-Case Scenarios

This topic describes different scenarios in which defaults and node functions can be useful. To test these scenarios, you will need the following sample mapping: **`Tutorial\MissingFields.mfd`**. Our starting point is the mapping illustrated below.


ArticlesOfClothing
File: MissingFields.xml File/String string
ArticlesOfClothing restriction of xs:any
Article ArticleType [1..∞]
Name xs:string [0..1]
Price xs:decimal [0..1]
ItemsInStock xs:integer [0..1]
Description xs:string [0..1]
ArticlesInfo
File: (default) File/String strin
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

Some of the nodes in the source file are empty. At this stage, the output file looks as follows:

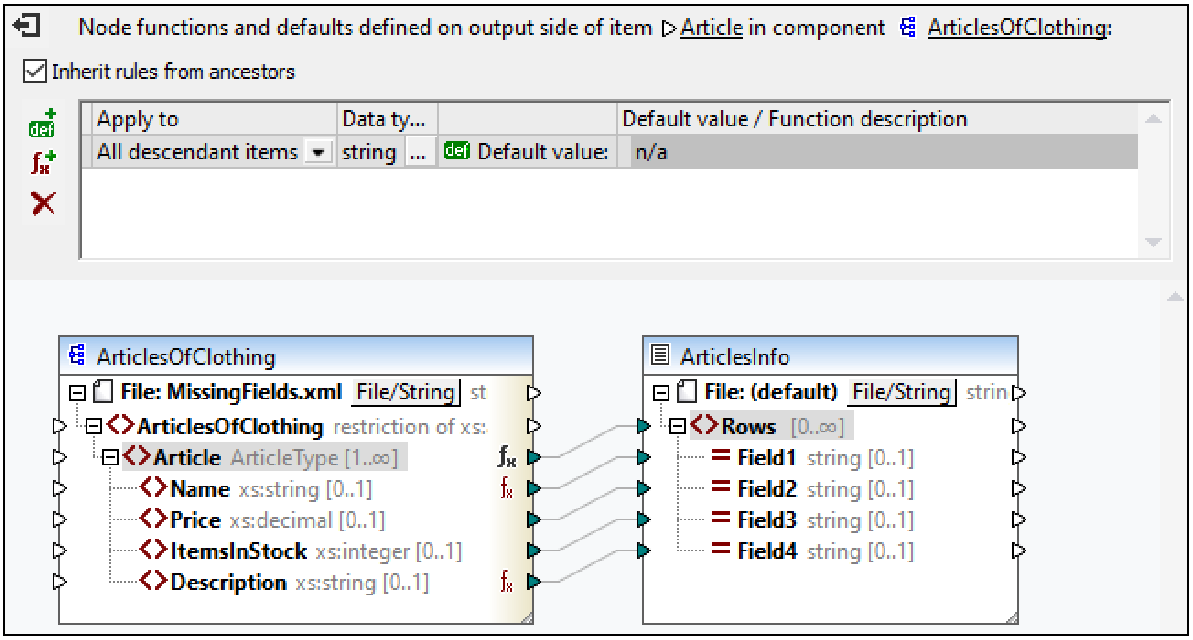
```
T-Shirt         25.3            20      Available in black, blue, and red
Shirt           70.3            60
Pants                                   Limited stock
Jacket          57.5            40
```

The scenarios described below will show you how to set a default for all nodes of type `string`, set defaults for different data types, block rules for specific nodes, override inherited rules, set a node function and conditionally apply this function to relevant nodes, and apply defaults to unconnected nodes.

Scenario 1: Set a default for all string nodes

In our mapping, some of the nodes of type `string` are empty. Our goal is to set a default value `n/a` instead of the empty strings. Take the following steps:

1. Right-click the `Article` element and select **Node Functions and Defaults | Output Node Functions and Defaults** from the context menu.
2. Click the def option in the Node Functions window and type `n/a` in the *Default value* field (*see screenshot below*).



Node functions and defaults defined on output side of item ▷Article in component ArticlesOfClothing:

☑ Inherit rules from ancestors

Apply to	Data ty...		Default value / Function description
All descendant items	string ...	Default value:	n/a

ArticlesOfClothing
File: MissingFields.xml File/String st
ArticlesOfClothing restriction of xs:
Article ArticleType [1..∞]
Name xs:string [0..1]
Price xs:decimal [0..1]
ItemsInStock xs:integer [0..1]
Description xs:string [0..1]
ArticlesInfo
File: (default) File/String strin
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

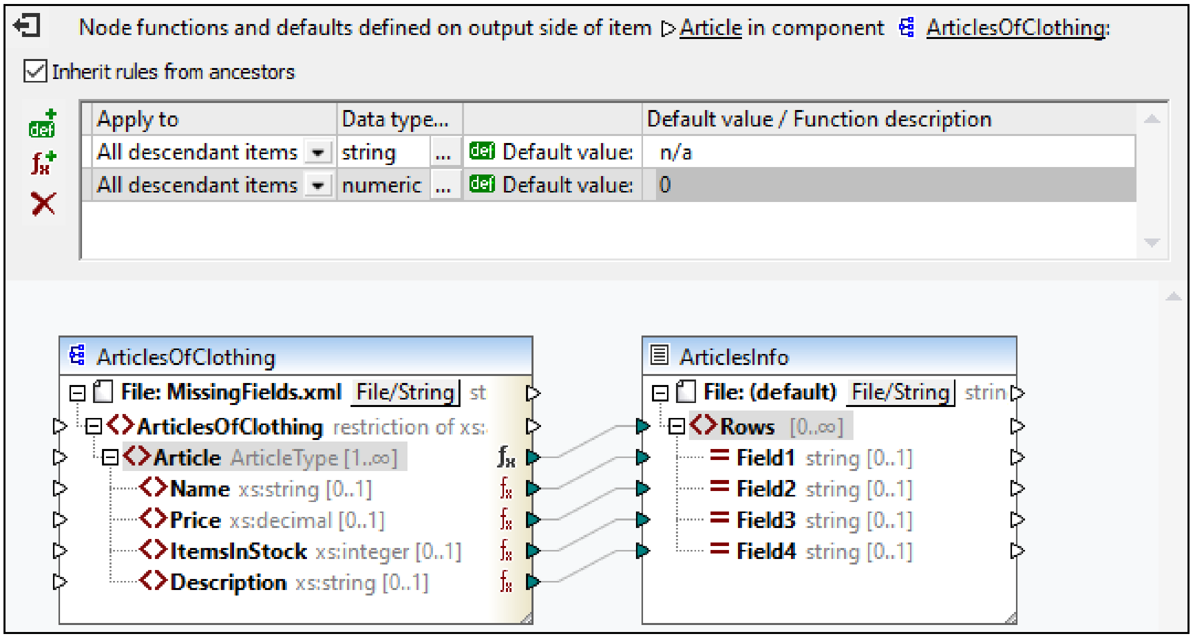
In the `ArticlesOfClothing` component, the `Name` and `Description` elements are of type `string`, which makes the default value suitable for both of them. Therefore, the fx icon appears next to these nodes. Since default values apply only to empty values, only two `Field4` nodes have received the `n/a` values in the output (*highlighted yellow below*).

```
T-Shirt                 25.3                    20          Available in black, blue, and red
Shirt                   70.3                    60          n/a
Pants                                                       Limited stock
Jacket                  57.5                    40          n/a
```

Scenario 2: Set defaults for different data types

Besides setting a default for nodes of one specific type, you can additionally set another default for some other data type. For example, one of the articles has empty values in `Price` and `ItemsInStock`, both of which are of numeric type. Now our goal is to replace these empty values with `0`. Take the steps below:

1. Repeat the steps from *Scenario 1*. This will add a default value for all nodes of type `string` that have empty values.
2. Add another default and select `numeric` as a data type.
3. Type `0` in the *Default value* field (*see screenshot below*).



Node functions and defaults defined on output side of item ▷Article in component ArticlesOfClothing:

☑ 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

ArticlesOfClothing
File: MissingFields.xml File/String st
ArticlesOfClothing restriction of xs:
Article ArticleType [1..∞]
Name xs:string [0..1]
Price xs:decimal [0..1]
ItemsInStock xs:integer [0..1]
Description xs:string [0..1]
ArticlesInfo
File: (default) File/String strin
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

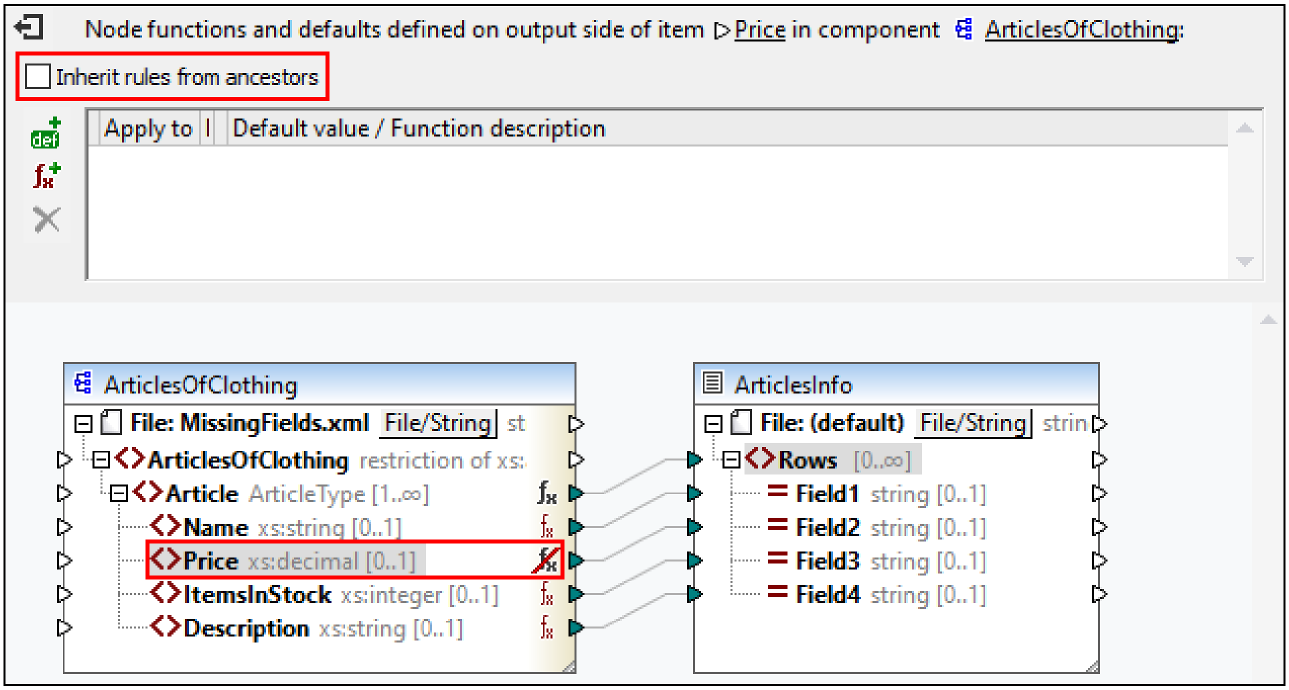
The output now looks as follows (*note the highlighted parts*):

```
T-Shirt         25.3            20      Available in black, blue, and red
Shirt           70.3            60      n/a
Pants           0               0       Limited stock
Jacket          57.5            40      n/a
```

Note: According to the XML Schema Specification, `integer` is derived from `decimal`, and both of them belong to the numeric data type. In our example, the rule will apply to both `ItemsInStock` and `Price` elements if you select `numeric` as a data type in the Node Functions window. If you select `decimal` as a data type, the rule will still apply to both elements. However, if you select `integer` as a data type, the rule will apply only to the `ItemsInStock` element.

Scenario 3: Block rules for specific nodes

In this scenario, we want to apply defaults for all string and numeric nodes, but we do not want to set a default for the `Price` item. To achieve the goal, reproduce the steps from *Scenario 2*, click the `Price` element, and clear the check box *Inherit rules from ancestors*. In the mapping below, the `Price` element no longer inherits rules from its parent, `Article`. Therefore, the fx icon appears next to the `Price` element (*see screenshot below*).



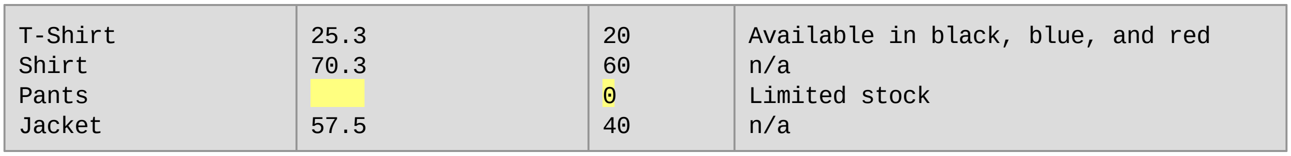
Node functions and defaults defined on output side of item ▷Price in component ArticlesOfClothing:

☐ Inherit rules from ancestors

Apply to	I	Default value / Function description

ArticlesOfClothing
File: MissingFields.xml File/String st
ArticlesOfClothing restriction of xs:
Article ArticleType [1..∞]
Name xs:string [0..1]
Price xs:decimal [0..1]
ItemsInStock xs:integer [0..1]
Description xs:string [0..1]
ArticlesInfo
File: (default) File/String strin
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
Field4 string [0..1]

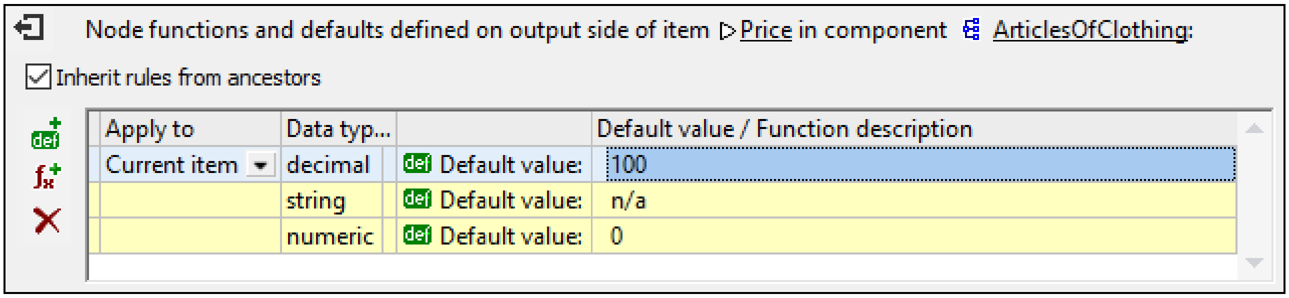
The output below shows that the empty value of the `Price` element, for which the rule has been blocked, has been mapped to `Field2` (*note the highlighted parts*).



```
T-Shirt      25.3     20   Available in black, blue, and red
Shirt        70.3     60   n/a
Pants                 0    Limited stock
Jacket       57.5     40   n/a
```

Scenario 4: Override inherited rules

In this scenario, we want to set defaults for all string and numeric nodes; however, for the `Price` element exclusively, we want to set `100` as a default value. To achieve this, repeat the steps outlined in *Scenario 2*, click `Price` in the `ArticlesOfClothing` component, add a new default and type `100`, as shown in the screenshot below. The inherited rules have a yellow background.



Node functions and defaults defined on output side of item ▷Price in component ArticlesOfClothing:

☑ Inherit rules from ancestors

Apply to	Data typ...		Default value / Function description
Current item ▾	decimal	def Default value:	100
	string	def Default value:	n/a
	numeric	def Default value:	0

The rule defined directly on a node has priority over the rules created at parent level. Therefore, the output now looks as follows (*note the highlighted part*):

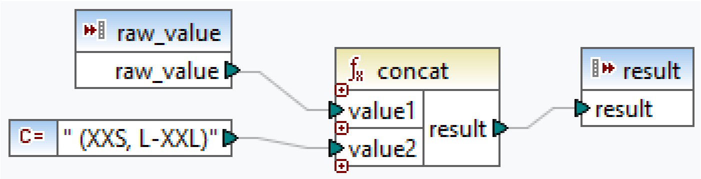
T-Shirt	25.3	20	Available in black, blue, and red
Shirt	70.3	60	n/a
Pants	100	0	Limited stock
Jacket	57.5	40	n/a

If you have defined more than one rules for the same node, the rule at the top of the grid will apply to the current node. You can swap the order of these rules by manually dragging the rules in the grid.

Scenario 5: Create a node function and apply it conditionally

This scenario will explain how to create a node function and apply it conditionally. We will use the **concat** function to add some text to nodes of type `string`. Follow the instructions below.

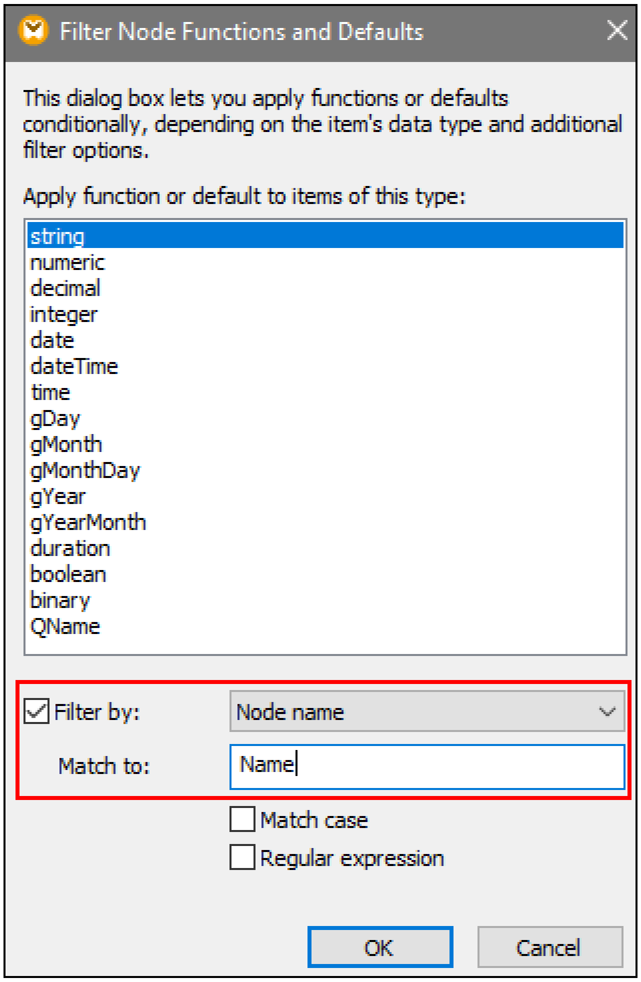
1. Right-click the `Article` element and select **Node Functions and Defaults | Output Node Functions and Defaults** from the context menu.
2. Add a node function by clicking the icon and create the function shown below.



At this stage, we have not filtered the nodes yet, and the output looks as follows (*note the highlighted parts*):

T-Shirt (XXS, L-XXL)	25.3	20	Available in black, blue, and red (XXS, L-XXL)
Shirt (XXS, L-XXL)	70.3	60	
Pants (XXS, L-XXL)			Limited stock (XXS, L-XXL)
Jacket (XXS, L-XXL)	57.5	40	

The output shows that our rule has been applied to both elements of type `string`: `Name` and `Description`. Now we want the rule to apply only to the `Name` element. To achieve this, go to the Node Functions window and click the ... button in the *Data type and filter* column. This opens the **Filter Node Functions and Defaults** dialog box, in which you can change data types and specify filtering options. Define the filter as shown in the screenshot below.



Filter Node Functions and Defaults

This dialog box lets you apply functions or defaults conditionally, depending on the item's data type and additional filter options.

Apply function or default to items of this type:

- string
- numeric
- decimal
- integer
- date
- dateTime
- time
- gDay
- gMonth
- gMonthDay
- gYear
- gYearMonth
- duration
- boolean
- binary
- QName

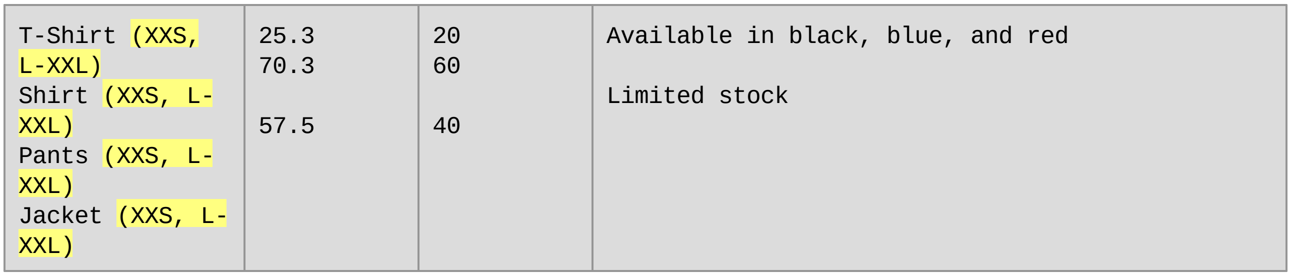
☑ Filter by: Node name

Match to: Name

□ Match case

□ Regular expression

The output now looks as follows:



T-Shirt (XXS, L-XXL) Shirt (XXS, L-XXL) Pants (XXS, L-XXL) Jacket (XXS, L-XXL)	25.3 70.3 57.5	20 60 40	Available in black, blue, and red Limited stock

Filter by node type

This option allows you to narrow down the data type you have selected in the **Filter Node Functions and Defaults** dialog. For example, you have chosen to apply your node function to numeric nodes. Then you can specify a subtype (e.g., *Filter by/Node name:* `xs:decimal`).

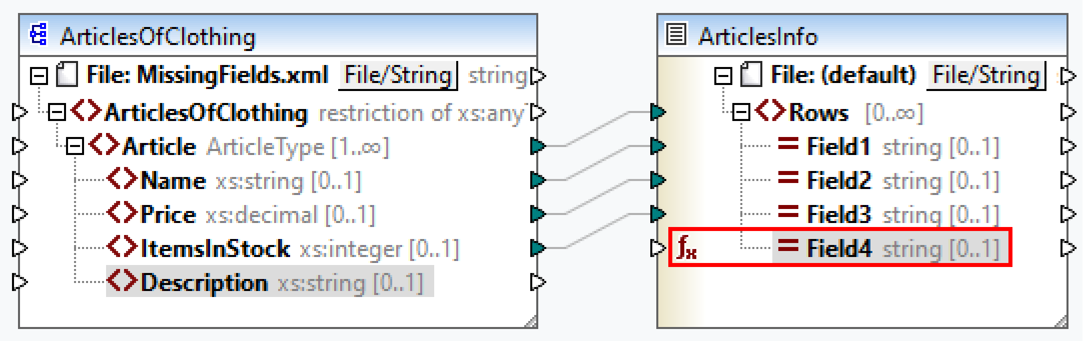
Filter with regular expressions

For more advanced filtering options, you can use regular expressions (e.g., to match multiple node names or type names). For more information, see Regular Expressions 508. Note the following points:

- The anchors `^` and `$` are implicit and must not be entered in the *Match to* box.
- Case sensitivity can be specified in the *Match Case* check box. Therefore, the `i` flag is not supported.
- Matching on multiple lines is not meaningful for node filtering. Therefore, the `m` flag is not supported.

Scenario 6: Create defaults for unconnected nodes

MapForce allows you to apply defaults to unconnected nodes but only if the parent node and/or a sibling are connected. Consider the following example:


ArticlesOfClothing
File: MissingFields.xml File/String string
ArticlesOfClothing restriction of xs:any
Article ArticleType [1..∞]
Name xs:string [0..1]
Price xs:decimal [0..1]
ItemsInStock xs:integer [0..1]
Description xs:string [0..1]
ArticlesInfo
File: (default) File/String
Rows [0..∞]
Field1 string [0..1]
Field2 string [0..1]
Field3 string [0..1]
fx
Field4 string [0..1]

In the mapping above, we have defined a default value for the `Field4` node in the `ArticlesInfo` component. At this stage, `Field4` does not have an input connection. The parent and sibling nodes are connected. If `Field4` remains unconnected, the default value will overwrite all the `Description` values in the output (*see below*). If you connect `Description` and `Field4`, the default value will apply only to empty sequences.

T-Shirt Shirt Pants Jacket	25.3 70.3 57.5	20 60 40	n/a n/a n/a n/a

Note: Defaults do not apply to unconnected nodes in JSON components.

5.9.3.3 Node Metadata in Node Functions

There might be cases in which you want a node function to do something based on *node metadata*. By node metadata, we understand miscellaneous information about the node on which the function applies (e.g., a node name, value length, and precision), which is stored in the schema. The table below lists all possible metadata parameters that you can use in a node function. Note that some metadata parameters apply only to specific data types. MapForce will display a warning when you attempt to use metadata that is incompatible with the current node.

Metadata parameters	Description
`node_name`	Supplies the name of the current node as it is shown in the component. This metadata is supported by all nodes. In XML files, `node_name` refers to the name of the current element or attribute. In CSV components,

Metadata parameters	Description
	node_name refers to the name of a CSV field. In databases, node_name refers to the name of a table column.
node_annotation	Displays annotation text next to the node value in the output. This metadata is supported by all nodes.
node_minLength	Supplies the value of the minLength facet of the node's data type. Applicable to XML and text nodes with appropriate types. For more information about length-constraining facets, see the XML Schema Recommendation.
node_maxLength	Supplies the value of the maxLength facet of the node's data type. Applicable to XML and text nodes with appropriate types. For more information about length-constraining facets, see the XML Schema Recommendation.
node_totalDigits	Supplies the value of the totalDigits facet of the node's data type. Applicable to XML nodes with appropriate types. For details about the totalDigits facet, see the XML Schema Recommendation.
node_fractionDigits	Supplies the value of the fractionDigits facet of the node's data type. Applicable to XML nodes with appropriate types. For details about the fractionDigits facet, see the XML Schema Recommendation.
node_length	Supplies the length (in bytes) of a number. Applicable to database fields with appropriate types. To find out more about length, see the Microsoft documentation.
node_precision	Supplies the number of all digits of a number. Applicable to database fields with appropriate types. For more information, see the Microsoft documentation.
node_scale	Supplies the number of digits to the right of the decimal point of a number. Applicable to database fields with appropriate types. For more information, see the Microsoft documentation.

Note: The data type of a metadata parameter must coincide with the data type of the relevant node. Otherwise, the function execution will fail.

Add metadata to a node function

To add a metadata parameter to a node function, you need to create a node function[446] and then click the **Add Node Specifics** option, which is located under the Node Functions window. This will open the **Insert Input with Node Specific** dialog box, in which you can select a relevant metadata parameter. Instead of the **Add Node Specifics** option, you can also choose one of the following options:

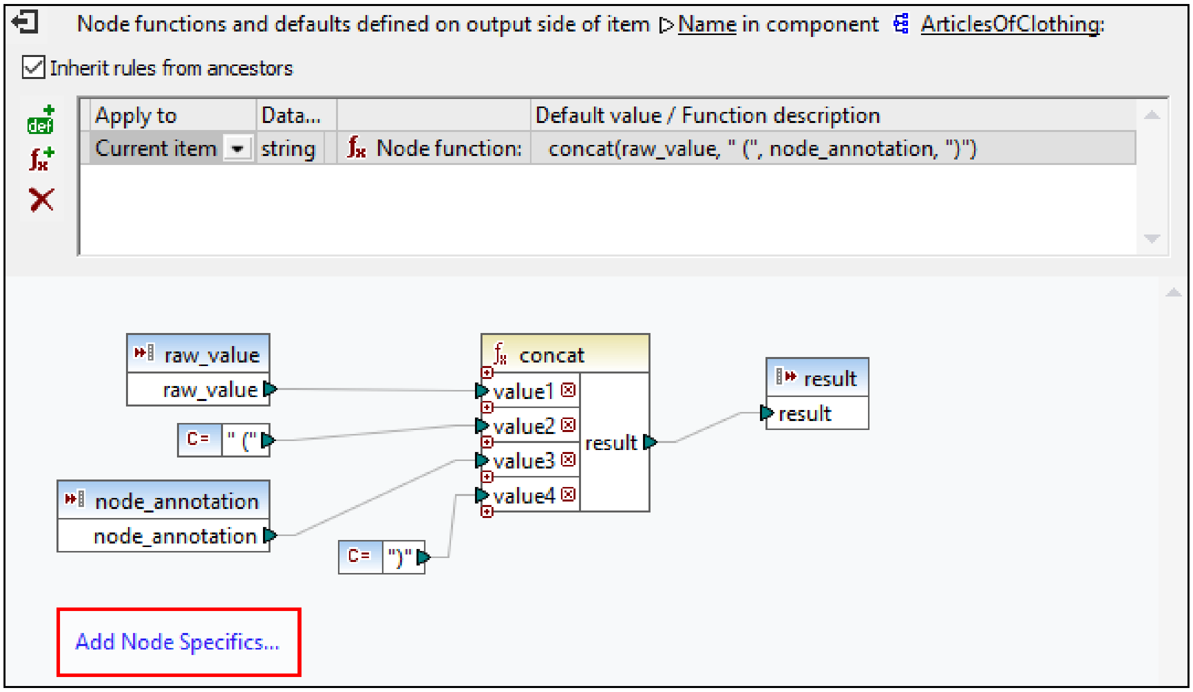
- Right-clicking an empty area in the mapping and selecting **Insert Input** from the context menu.
- Clicking the toolbar command.
- Clicking **Insert Input** in the **Function** menu.

When metadata is not supported by a node
When a metadata parameter is not supported by a node, you can instruct MapForce to return an empty sequence or not to apply the node function (both options are available in the **Insert Input with Node Specific** dialog box). The empty sequence must be handled; otherwise, the node function might not return a value at all. You would typically need to use sequence functions (e.g., `substitute-missing` 589 and `exists` 563) or other component types to process the empty sequence further.

Example

The example below explains how to supply annotation text to a node function. For this example, you will need the following mapping:
`<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\MissingFields.mfd`. In **`MissingFields.xsd`** (on which **`MissingFields.xml`** is based), we have added the following annotation for the `Name` element: `Article name from catalog Spring 2022`. Our goal is to supply this annotation to a node function and to see the annotation in the output. Follow the instructions below.

1. Create a node function 446 for the `Name` element in the `ArticlesOfClothing` component.
2. Replicate the function shown below. To add a `node_annotation` parameter, click **Add Node Specifics** (*see red rectangle below*) and select this parameter from the list. To find out how to add and manipulate functions, see Functions Basics 436.



Node functions and defaults defined on output side of item ▷ Name in component ArticlesOfClothing:

☑ Inherit rules from ancestors

Apply to	Data...		Default value / Function description
Current item	string	Node function:	concat(raw_value, " (", node_annotation, ")")

Note: By default, deeply nested structures are not fully scanned so as to preserve memory and improve the user experience. If the component where you apply the node function has such deeply nested structures, you can expand the relevant nodes in the mapping to make MapForce aware of them. In this case, MapForce will take the expanded nodes into account when you add a new metadata parameter, and the warning may disappear. The node function must be connected in order to take effect; expanding unconnected items is not relevant.

Output
The listing below shows that the annotation has been added to the values of the `Name` elements.

```
T-Shirt (Article name from catalog Spring 2022)   25.5  20    Available in black, blue, 
and red
Shirt (Article name from catalog Spring 2022)     70.3  60

Pants (Article name from catalog Spring 2022)                 Limited
stock
Jacket (Article name from catalog Spring 2022)    57.5  40
```

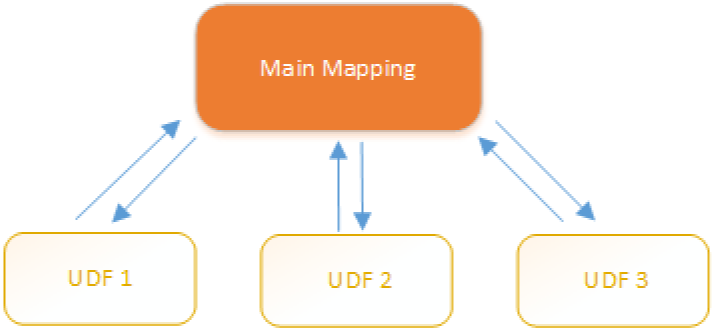
5.9.4 User-Defined Functions

User-Defined Functions (UDFs for short) are custom functions that are defined once and can be reused multiple times within the same mapping or across multiple mappings. UDFs are like mini-mappings themselves: They typically consist of one or more input parameters, some intermediary components to process data, and an output to return data to the caller. The caller is the main mapping or another UDF.

Advantages of UDFs

UDFs have the following advantages:

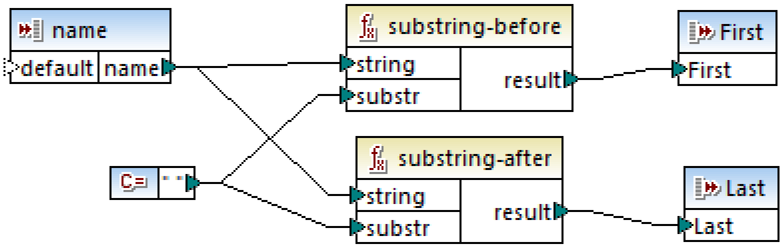
- UDFs are reusable within one mapping or across multiple mappings.
- UDFs can make your mapping easier to read: For example, you can package parts of the mapping into smaller components and abstract away the implementation details. The diagram below illustrates this principle.



- UDFs are flexible functions that enable you to process strings, numbers, dates, and other data in a custom way that extends the built-in MapForce functions. For example, you might want to concatenate or split text in a particular way, perform some advanced calculations, manipulate dates and times, and so on.
- Another common use of UDFs is to look up a field in an XML file, database or some other data format supported by your MapForce edition and present this data in a convenient way. For details, see Look-up Implementation[472].
- UDFs can be called recursively (i.e., the UDF calls itself). This requires that the UDF be defined as a regular (not inline) function[460]. Recursive UDFs[467] can fulfill various advanced mapping requirements, such as iterating over data structures having a depth of *N* children, where *N* is not known in advance.

Example

Below is an example of a simple UDF that splits a string into two separate strings. This UDF is part of a larger mapping called **`MapForceExamples\ContactsFromPO.mfd`**. The UDF takes the name as a parameter (e.g., `Helen Smith`), applies the built-in functions **`substring-before`** and **`substring-after`**, and then returns two values: `Helen` and `Smith`.



In this section

This section explains how to work with UDFs and is organized into the following topics:

- UDF Basics 458
- UDF Parameters 463
- Recursive UDFs 467
- Look-up Implementation 472

5.9.4.1 UDF Basics

This topic explains how to create, import, edit, copy-paste, and delete user-defined functions (UDFs for short).

Create a UDF

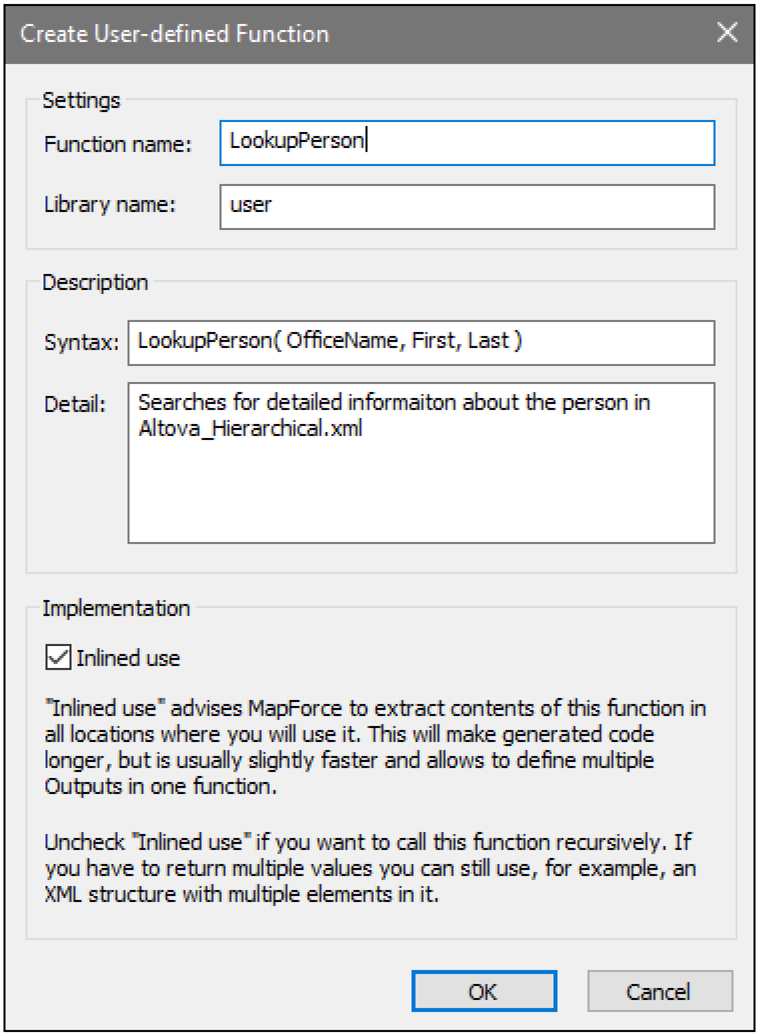
This subsection explains how to create a UDF from scratch and from already existing components. The minimum requirement is one output component to which some data is connected. For input parameters, a function can have zero, one or more inputs. The input and output parameters can be of simple type (e.g., a string) or complex type (a structure). For more information about simple and complex parameters, see UDF Parameters 463.

UDF from scratch

To create a UDF from scratch, follow the instructions below:

1. Select **Function | Create User-Defined Function**. Alternatively, click the toolbar button.

2. Enter the relevant information into the **Create User-defined Function** dialog (*see screenshot below*).



Function name: LookupPerson

Library name: user

Syntax: LookupPerson(OfficeName, First, Last)

Detail: Searches for detailed informaiton about the person in Altova_Hierarchical.xml

☑ 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, an XML structure with multiple elements in it.

The available options are listed below.

- *Function Name:* Mandatory field. For the name of your UDF, you can use the following characters: alphanumeric characters (a-z, A-Z, 0-9), an underscore (_), a hyphen/dash (-), and a colon (:).
- *Library Name:* Mandatory field. This is the name of a function library (in the Libraries window 23) in which your function will be saved. If you do not specify the name of a library, the function will be placed into a default library called `user`.
- *Syntax:* Optional field. Enter some text that concisely describes the syntax of the function (e.g., expected parameters). This text will be displayed next to the function in the **Libraries** window and does not affect the implementation of the function.
- *Detail:* Optional field. This description will be displayed when you move the cursor over the function in the **Libraries** window or in other contexts.
- *Inlined use:* Select this check box if the function should be created as inline. For more information, see *Regular vs. Inline UDFs* below.

3. Click **OK**. The function becomes immediately visible in the **Libraries** window under the library name specified above. The mapping window is now redrawn to allow you to create a new function (this is a standalone mapping referred to as *the function's mapping*). The function's mapping includes an output component by default.

4. Add all the required components to the function's mapping. You can do this in the same way as for a standard mapping.

To use the UDF in a mapping, drag the UDF from the **Libraries** window onto the main mapping area. See also *Call and import UDFs* below.

UDF from existing components
To create a UDF from existing components, take the steps below:

1. Select the relevant components on the mapping by making a rectangular selection with the mouse. You can also select multiple components by clicking each one while holding the **Ctrl** key pressed.
2. Select the menu command **Function | Create User-Defined Function from Selection**. Alternatively, click the toolbar button.
3. Follow Steps 2-4 from *UDF From Scratch*.

Regular vs. Inline UDFs
There are two types of UDFs: *regular* and *inline*. You can specify the type of your UDF when you create it. Inline and regular functions behave differently in terms of code generation, recursiveness, and the ability to have multiple output parameters. The table below summarizes the main differences between regular and inline UDFs.

Inline functions (dashed border)	**Regular functions (solid border)**
With inline functions, the UDF code is inserted at all locations where the function is called. If the UDF is called several times, the generated program code would be significantly longer.	The code for the UDF is generated once, and inputs to it are passed as parameter values. If the UDF is called several times, the UDF is evaluated each time with the corresponding parameter values.
Inline functions can have multiple outputs and thus return multiple values.	Regular functions can have only one output. To return multiple values, you can declare the output to be of complex type (e.g., an XML structure), which would allow you to pass multiple values to the caller.
Inline functions cannot be called recursively.	Regular functions can be called recursively.
Inline functions do not support setting a priority context[86] on a parameter.	Regular functions support setting a priority context on a parameter.
Inline UDFs affect and are affected by the context in the mapping that calls the UDF.	Regular UDFs have their own local context.

Note: Switching a UDF from inline to regular, and vice versa, may affect the mapping context[79], and this may cause the mapping to produce a different result.

Call and import UDFs

After you have created a UDF, you can call it from the same mapping where you created it or from any other mapping.

Call UDF from the same mapping

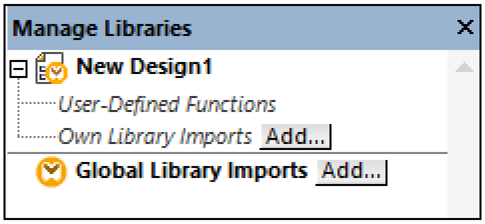
To call a UDF from the same mapping, take steps below:

1. Find the relevant function in the **Libraries** window under the library that you specified when you created the function. To do that, start typing the name in the **Libraries** window.
2. Drag the function from the **Libraries** window into the mapping. You can now connect all the required parameters to the function.

Import UDF from a different mapping

To import a UDF from another mapping, follow the instructions below:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window 23. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for the **`.mfd`** file that contains the UDF and click **Open**. A message box will inform you that a new library has been added and can be accessed in the **Libraries** window.

You can now use any of the imported functions in the current mapping by dragging them from the **Libraries** window into the mapping. For more information about viewing and organizing function libraries, see Manage Function Libraries 438.

Mapping with credentials (Enterprise Edition)

If the imported **`.mfd`** file contains credentials, these are shown as imported (with a yellow background) in the Credentials Manager. By default, imported credentials are not saved with the mapping, but you can optionally create a local copy and save them in the mapping.

Edit UDFs

To edit a UDF, take the steps below:

1. Open the mapping that contains a UDF.
2. Double-click the title bar of the UDF in the mapping to see the function's contents where you can add, edit, or remove components as required.

3. To change the function's properties (e.g., the name or description), right-click an empty area in the mapping and select **Function Settings** from the context menu. Alternatively, click the toolbar button.

You can also edit a function by double-clicking its name in the **Libraries** window. However, only functions in the currently active document can be opened this way. Double-clicking a UDF that was created in another mapping opens that mapping in a new window. If you edit or delete a UDF that was imported into multiple mappings, all these mappings will be affected by the change.

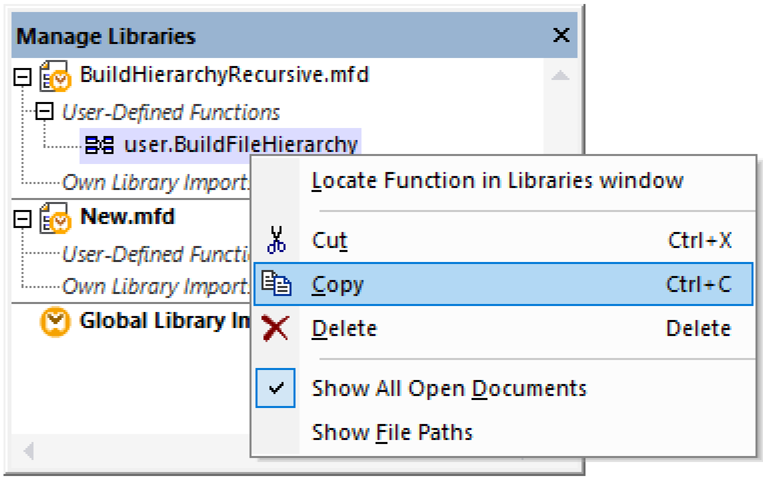
To go back to the main mapping, click the button in the top-left corner of the mapping window.

MapForce allows you to navigate through different mappings and UDFs by using the ⇦ and ⇨ toolbar buttons. The corresponding keyboard shortcuts for these buttons are **Alt+Left** and **Alt+Right**, respectively.

Copy-paste UDFs

To copy a UDF and paste it into another mapping, follow the steps below:

1. Open the Manage Libraries window 438.
2. Right-click inside an empty area in the **Libraries** window and select the option **Show All Open Documents**.
3. Open both the source and destination mappings. Make sure that both mappings are saved to disk. This ensures correct resolution of paths. See also Copy-Paste and Relative Paths 44.
4. Right-click the relevant UDF from the source mapping in the **Manage Libraries** window and select **Copy** from the context menu (*see screenshot below*) or press **Ctrl+C**. Leave the **Manage Libraries** window open.



5. Switch to the destination mapping (and the **Manage Libraries** window will change accordingly), right-click *User-Defined Functions*, and select **Paste** from the context menu.

Delete UDFs

To delete a UDF, take the steps below:

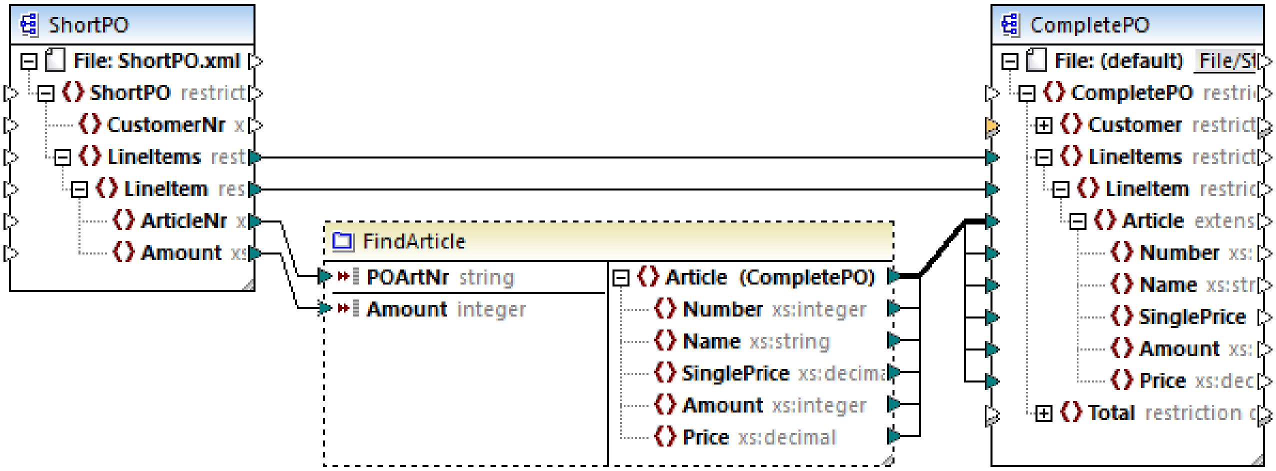
1. Double-click the title bar of the UDF in the mapping.
2. Click the button in the top-right corner of the Mapping window.

3. If the function is used in the currently open mapping, MapForce will ask whether you want to replace all instances with internal components. Click **Yes** if you want to delete the function and replace all instances where it is called with the function's components. This lets you keep the main mapping valid even if the function is deleted. However, if the deleted function is used in any other external mappings, those will not be valid. Click **No** if you want to delete the function and all its internal components permanently. In this case, all the mappings where the function is used will not be valid.

5.9.4.2 UDF Parameters

When you create a UDF, you must specify what input parameters it should take (if any) and what output it should return. While input parameters are sometimes not necessary, an output parameter is mandatory in all cases. Function parameters can be of simple type (e.g., `string` or `integer`) or a complex structure 464. For example, the `FindArticle` UDF illustrated below has two input and one output parameters:

- `POArtNr` is an input parameter of type `string`.
- `Amount` is an input parameter of type `integer`.
- `CompletePO` is an output parameter that has a complex XML structure.



Parameter order
When a UDF has multiple input or output parameters, you can change the order in which parameters will appear to the callers of this function. The order of parameters in the function's mapping (starting from the top) dictates the order in which they appear to the callers of this function.

Important

- Input and output parameters are sorted by their position from top to bottom. Therefore, if you move the `input3` parameter to the top in the function's mapping, it will become the first parameter of this function.
- If two parameters have the same vertical position, the leftmost takes precedence.
- In the unusual case that two parameters have exactly the same position, the internal component ID is automatically used.

Complex-type structures

The structures on which a parameter in a UDF can be based are summarized in the list below.

MapForce Basic Edition
- XML Schema Structure

MapForce Professional Edition
- XML Schema Structure
- Database Structure

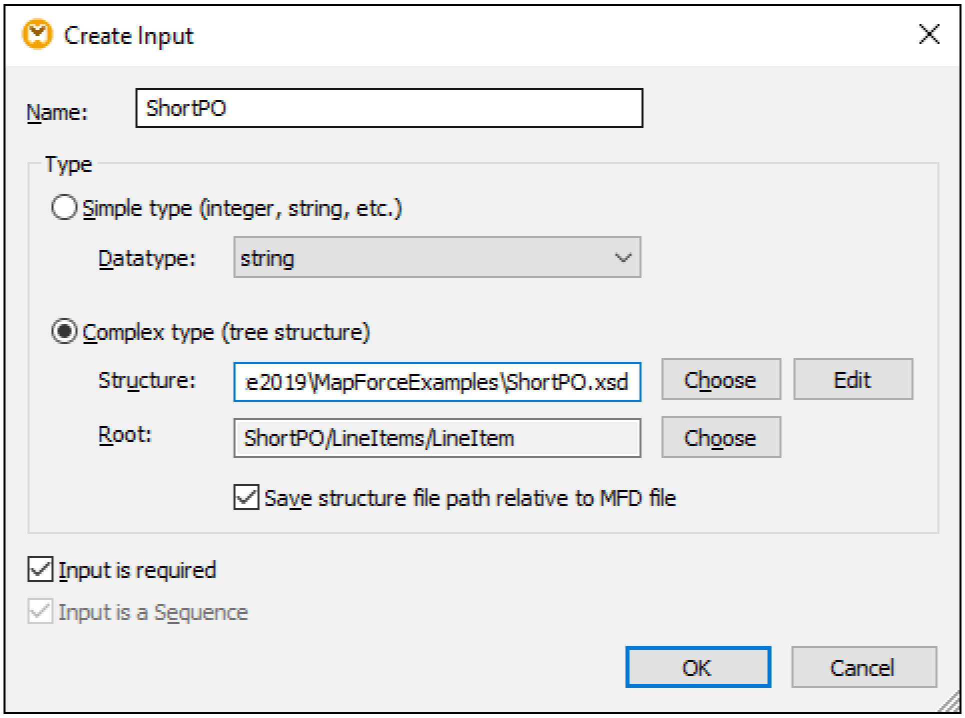
MapForce Enterprise Edition
- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure
- JSON Schema Structure

UDFs based on database structures (Professional and Enterprise editions)
MapForce allows you to create DB-based UDF parameters with a tree of related tables. The tree of related tables represents an in-memory structure that has no connection to the database at runtime. This also means that there is no automatic handling of foreign keys and no table actions in parameters or variables.

Add Parameters

To add an input or output parameter, take the following steps:

1. Create a UDF 458 or open an existing one 461.
2. Run the menu command **Function | Insert Input** or **Function | Insert Output** (*see screenshot below*). Alternatively, click (**Insert Input**) or (**Insert Output**) in the toolbar.



Create Input

Name: ShortPO

Type

○ Simple type (integer, string, etc.)

Datatype: string

● Complex type (tree structure)

Structure: e2019\MapForceExamples\ShortPO.xsd

Root: ShortPO/LineItems/LineItem

☑ Save structure file path relative to MFD file

☑ Input is required

☑ Input is a Sequence

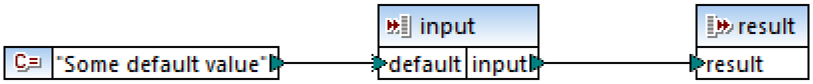
3. Choose whether input or output parameters should be of simple or complex type (*see dialog box above*). See the list of available complex structures above. For example, to create a parameter that is a complex XML type, click **Choose** next to *Structure* and browse for the XML schema that describes the required structure.

If the function's mapping already includes XML schemas, they become available for selection as structures. Otherwise, you can select a new schema that will provide the structure of the parameter. The same is true for databases and other complex structures if they are supported by your MapForce edition. With XML structures, it is possible to select the root element for your structure if the XML schema allows it. To specify the root element, click **Choose** next to *Root* and select the root item from the dialog box that opens.

If selected, the check box *Save structure file path relative to MFD file* will change the absolute path of the structure into a path relative to the current mapping, when you save the mapping. For more information, see Relative and Absolute Paths 43. The check boxes *Input is required* and *Input is a Sequence* are explained below.

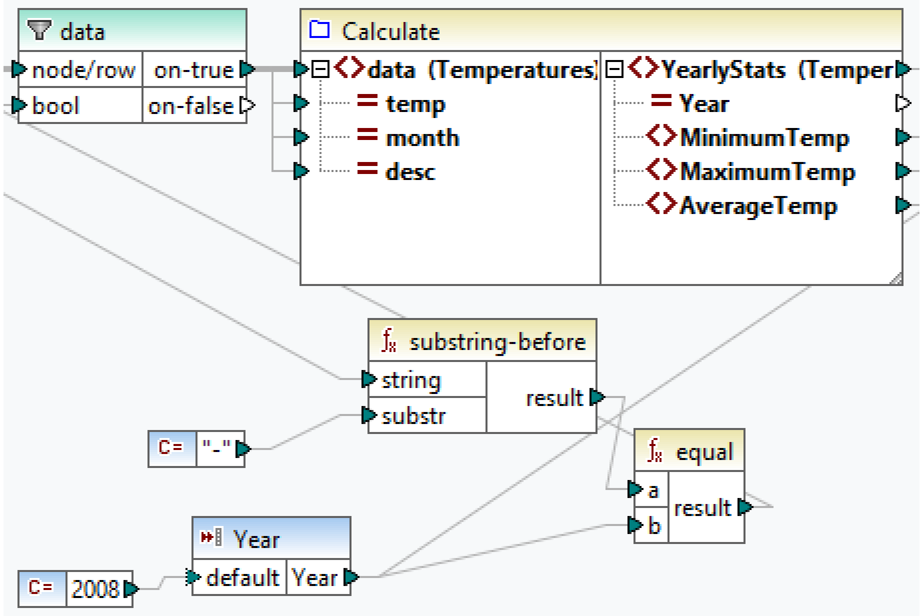
Input is required
To make a parameter mandatory in a UDF, select the *Input is required* check box (*see dialog box above*). If you clear the *Input is required* check box, the parameter will become optional and have a dashed border in the mapping.

You can also specify a default parameter value by connecting it to the `default` input of a parameter (*see example below*). The default value will apply only if there is no other value. If the optional parameter receives a value when the function is called, that value takes precedence over the default.

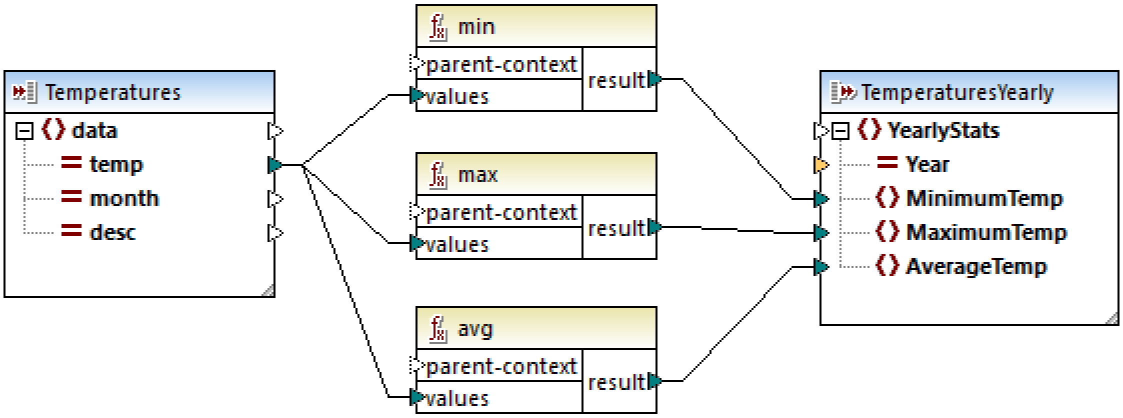


Input is a sequence
You can optionally define whether a function's parameter should be treated as a single value (default option) or as a sequence. A sequence is a range of zero or more values. A sequence might be useful when your UDF expects input data as a sequence in order to calculate values in that sequence, for example, by calling functions such as **`avg`**, **`min`**, **`max`**. To treat the input of the parameter as a sequence, select the *Input is sequence* check box. Note that this check box is enabled only if the UDF is regular 460.

The usage of a sequence is illustrated in the following mapping: **`MapForceExamples\InputIsSequence.mfd`**. In the extract of this mapping (*see screenshot below*), the `data` filter is connected to the UDF called `Calculate`. The filter's output is a sequence of items. Therefore, the input parameter of the function is set to be a sequence.



Illustrated below is the implementation of the `Calculate` function that aggregates all the sequence values: It runs the **avg**, **min**, and **max** functions on the input sequence. To see the internal structure of the `Calculate` function, double-click the header of the `Calculate` component in the mapping above.



As a rule of thumb, the input data (sequence or non-sequence) determines how often the function is called:

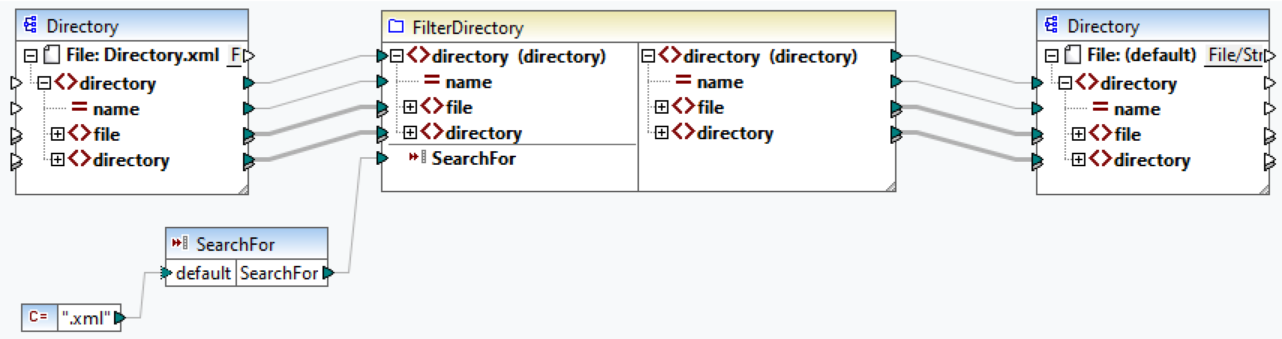
- When input data is connected to a *sequence* parameter, the UDF is called only *once*, and the complete sequence is passed into the UDF.
- When input data is connected to a *non-sequence* parameter, the UDF is called *once for each single item* in the sequence.
- If you connect an empty sequence to a non-sequence parameter, the function is not called at all. This can happen if the source structure has optional items or when a filter condition returns no matching items. To avoid this, use the `substitute-missing` 589 function before the function input to ensure that the sequence is never empty. Alternatively, set the parameter to a sequence and add handling for the empty sequence inside the function.

The *Output is a Sequence* check box may be required for output parameters, too. When a function passes a sequence of multiple values to its output component, and the output component is not set to sequence, the function will return only the first item in the sequence.

5.9.4.3 Recursive UDFs

This topic explains how to search for data in a source XML file with the help of a recursive UDF. To test the recursive UDF, you will need the following mapping: **`MapForceExamples\RecursiveDirectoryFilter.mfd`**. In the mapping below, the `FilterDirectory` UDF receives data from the source file `Directory.xml` and the simple input component `SearchFor` that supplies the `.xml` extension. After the data has been processed by the UDF, it is mapped to the target file.

The main mapping (*see screenshot below*) describes the general mapping layout. The way the UDF processes the data is defined separately, in the function's mapping (*See UDF Implementation below*).



Goal

Our goal is to list files with a **`.xml`** extension in the output while preserving the entire directory structure. The subsections below explain the mapping process in detail.

Source file

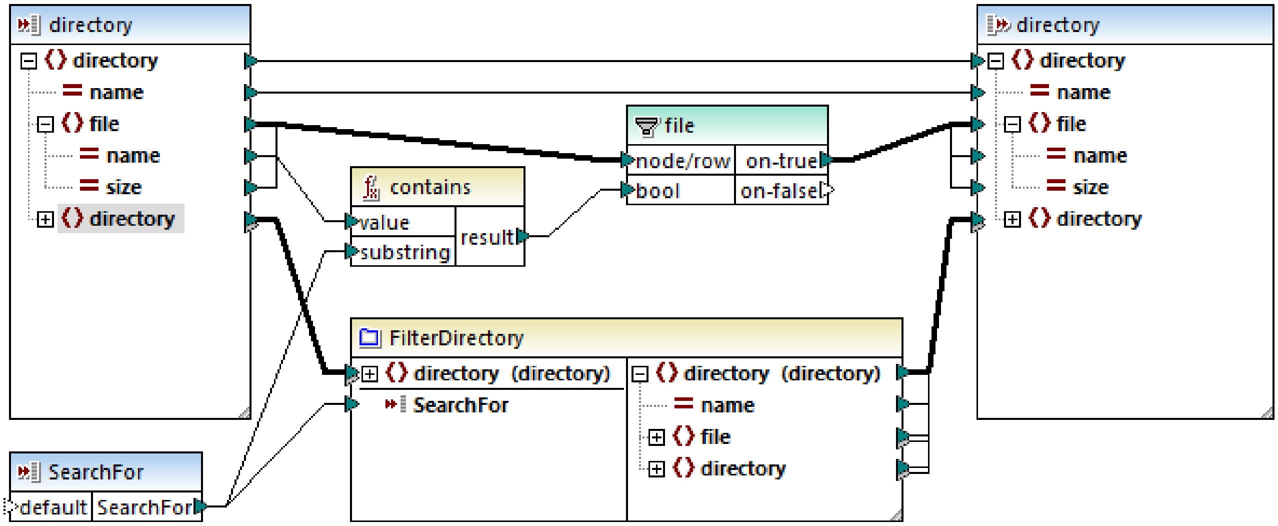
Below is an extract from the source XML file (**`Directory.xml`**) that contains information about files and directories. Note that the files in this listing have different extensions (e.g., **`.xml`**, **`.dtd`**, **`.sps`**). According to the schema (**`Directory.xsd`**), the `directory` element can have `file` children and `directory` children. All `directory` elements are recursive.

```
<directory name="ExampleSite">
  <file name="blocks.sps" size="7473"/>
  <file name="blocks.xml" size="670"/>
  <file name="block_file.xml" size="992"/>
  <file name="block_schema.xml" size="1170"/>
  <file name="contact.xml" size="453"/>
  <file name="dictionaries.xml" size="206"/>
  <file name="examplesite.dtd" size="230"/>
  <file name="examplesite.spp" size="1270"/>
  <file name="examplesite.sps" size="20968"/>
  ...
  <directory name="output">
     <file name="examplesite1.css" size="3174"/>
```

```
        <directory name="images">
          <file name="blank.gif" size="88"/>
          <file name="block_file.gif" size="13179"/>
          <file name="block_schema.gif" size="9211"/>
          <file name="nav_file.gif" size="60868"/>
          <file name="nav_schema.gif" size="6002"/>
        </directory>
      </directory>
     </directory>
```

UDF implementation

To see the internal implementation of the UDF, double-click its header in the main mapping. The UDF is recursive, that is, it includes a call to itself. Because it is connected to the recursive element `directory`, this function will be called as many times as there are nested `directory` elements in the source XML instance. To support recursive calls, the function must be regular 460.



The implementation of the UDF consists of two parts: (i) defining the files and (ii) defining the directory to be searched.

Defining files

The UDF processes the files as follows: The **contains** function checks whether the first string (the file name) contains the substring `.xml` (supplied by the simple input component `SearchFor`). If the function returns `true`, the file name with a `.xml` extension is written to the output.

Processing child directories

Child directories of the current directory are sent as input to the current UDF. The UDF thus iterates through all `directory` elements and check whether files with the **`.xml`** extension exist.

Output

When you click the **Output** pane, MapForce will display only files with the **`.xml`** extension (*see extract below*).

```
<directory name="ExampleSite">
  <file name="blocks.xml" size="670"/>
```

```
    <file name="block_file.xml" size="992"/>
    <file name="block_schema.xml" size="1170"/>
    <file name="contact.xml" size="453"/>
    ...
    <directory name="output">
      <directory name="images"/>
    </directory>
  </directory>
```

5.9.4.4 Context in UDFs

User-defined functions (UDFs) are custom functions embedded into the mapping, where you define the inputs, outputs, and processing logic. Each user-defined function may contain the same component kinds as a main mapping, including Web services and databases.

> By default, if a UDF contains a database or a web service component, and if the input data to the UDF is a sequence of multiple values, then each input value will call the UDF and consequently will result in a database or web service call.

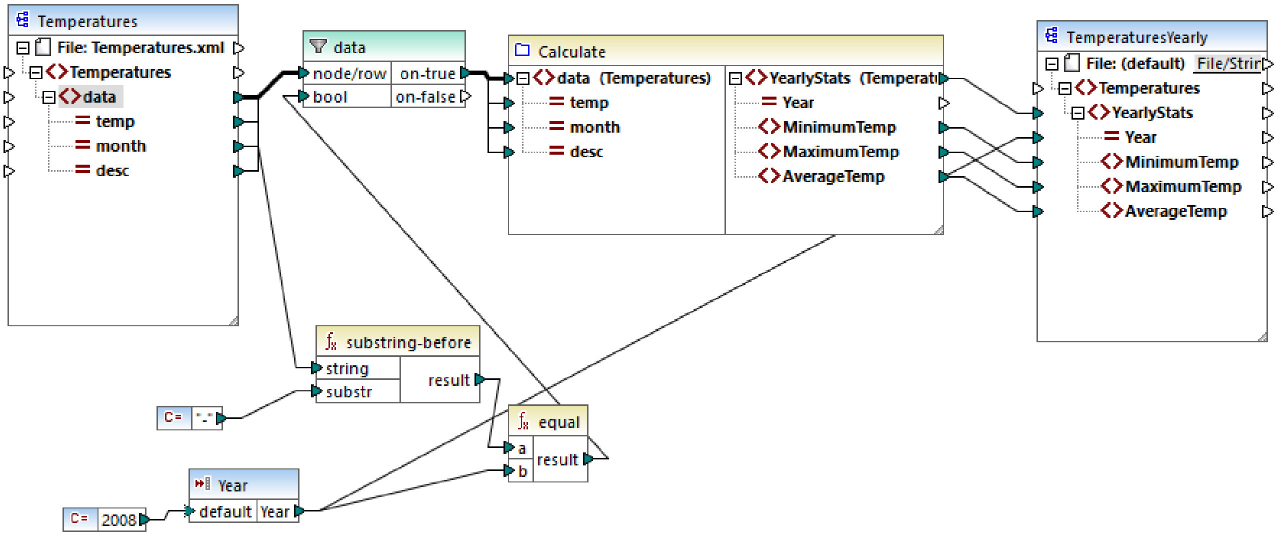
The behavior above may be acceptable for those mappings where you really need the UDF to be called *as many times as there are input values* and there is simply no other alternative way.

If you do not want the above to happen, you can configure the UDF so that it is called only once even if gets a sequence of values as input. You will typically want to do this for those UDFs that operate on a set of values before they can return (such as functions that calculate averages or totals).

Configuring a UDF to accept multiple input values in the same function call is possible if the UDF is of type "regular", not "inlined". (For details, see the User-Defined Functions 457 chapter.) With regular functions, you can specify that the input parameter is a sequence by selecting the **Input is a sequence** check box. This check box is visible on the component settings, after you double-click the title bar of an input parameter. The check box affects how often the function is called, as follows:

- When input data is connected to a **sequence** parameter, the user-defined function is called *only once* and the complete sequence is passed into the user-defined function.
- When input data is connected to a **non-sequence** parameter, the user-defined function is called *once for each single item in the sequence*.

For an example, open the following demo mapping:
<Documents>\Altova\MapForce2025\MapForceExamples\InputIsSequence.mfd.

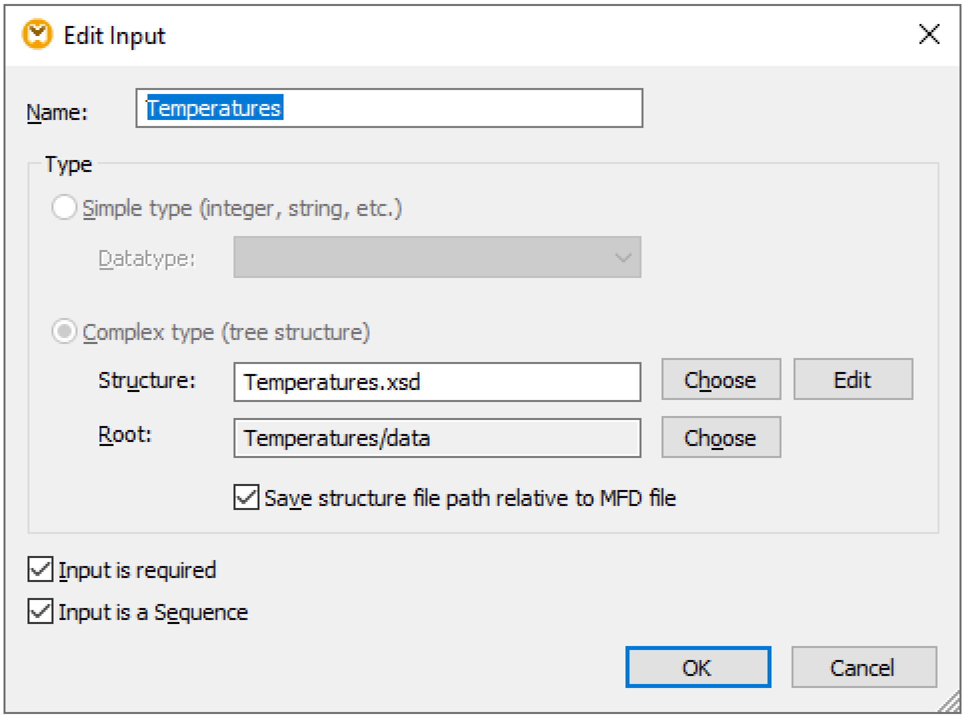


The mapping above illustrates a typical case of a UDF that operates on a set of values and thus requires all the input values in one call. Specifically, the **`Calculate`** user-defined function returns the minimum, maximum and average temperatures, taking as input data from a source XML file. The expected mapping output is as follows:

```
<Temperatures>
    <YearlyStats Year="2008">
        <MinimumTemp>-0.5</MinimumTemp>
        <MaximumTemp>24</MaximumTemp>
        <AverageTemp>11.6</AverageTemp>
    </YearlyStats>
</Temperatures>
```

As usual, the mapping execution begins with the top item of the target component (**YearlyStats**, in this example). To populate this node, the mapping attempts to obtain source data from the UDF, which in its turn, triggers the filter. The filter's role in this mapping is to pass onto the UDF only temperatures from year 2008.

The check box **Input is sequence** was selected for the input parameter of the UDF (To view this check box, double-click the title bar of the **`Calculate`** function to enter the function's mapping; then double-click the title bar of the input parameter). As mentioned before, the **Input is sequence** option causes the complete sequence of values to be supplied as input to the function and the function is called only once.



Edit Input

Name: Temperatures

Type

○ Simple type (integer, string, etc.)

Datatype:

● Complex type (tree structure)

Structure: Temperatures.xsd

Root: Temperatures/data

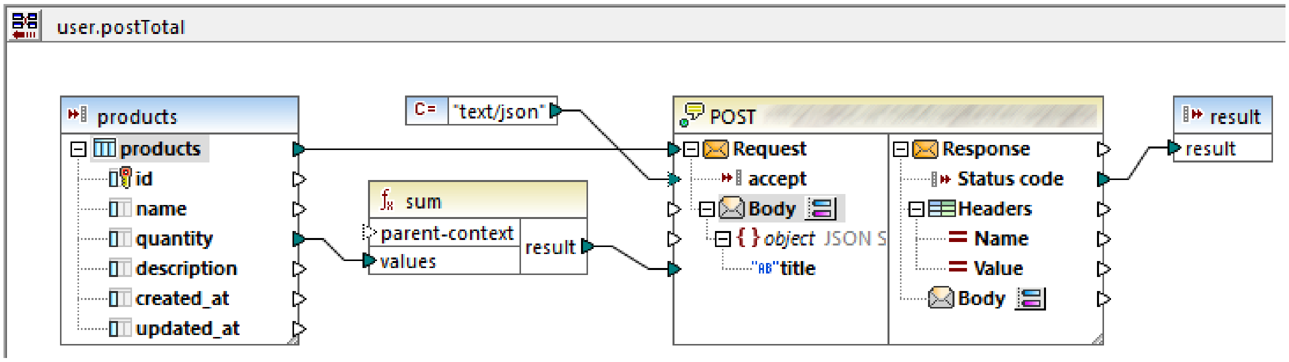
☑ Save structure file path relative to MFD file

☑ Input is required

☑ Input is a Sequence

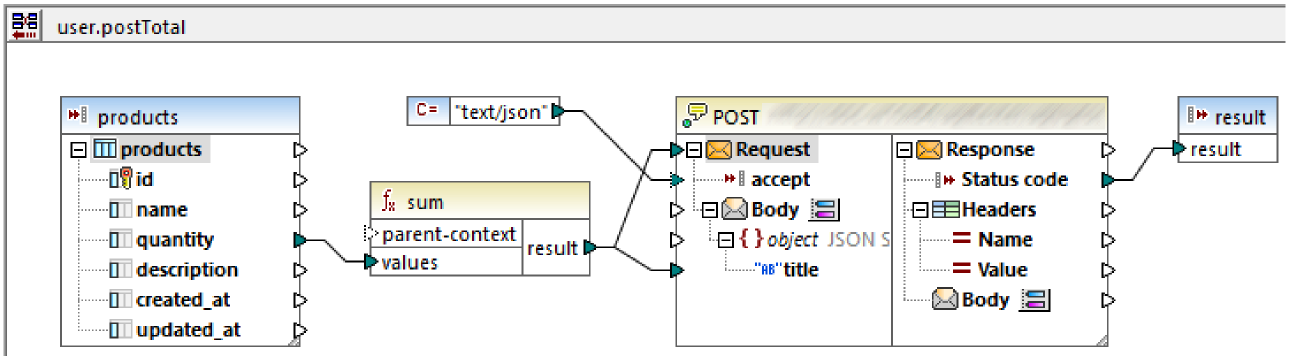
Had the **Input is sequence** check box not been selected, the UDF would have been called for each value in the source. As a result, the minimum, maximum and average would be calculated for each single value individually and incorrect output would be produced.

By applying the same logic in more complex UDFs that include database or web service calls, it may be possible to optimize the execution and avoid unnecessary calls to the database or web service. Nevertheless, be aware that the **Input is sequence** check box does not control what happens to the sequence of values *after* it enters the function. In other words, there is nothing to prevent you from connecting the incoming sequence of values to the input of a web service and thus call it multiple times. Consider the following example:



The UDF illustrated above receives a sequence of values from the external mapping. Specifically, the data supplied to the input parameter originates from a database. The input parameter has the option **Input is sequence** selected, so the entire sequence is supplied to the function in one call. The function is supposed to add up multiple **quantity** values and post the result to a web service. Exactly one web service call is expected. However, the web service will be incorrectly called multiple times when the mapping runs. The reason is that the **Request** input of the web service receives *a sequence of values*, not a single value.

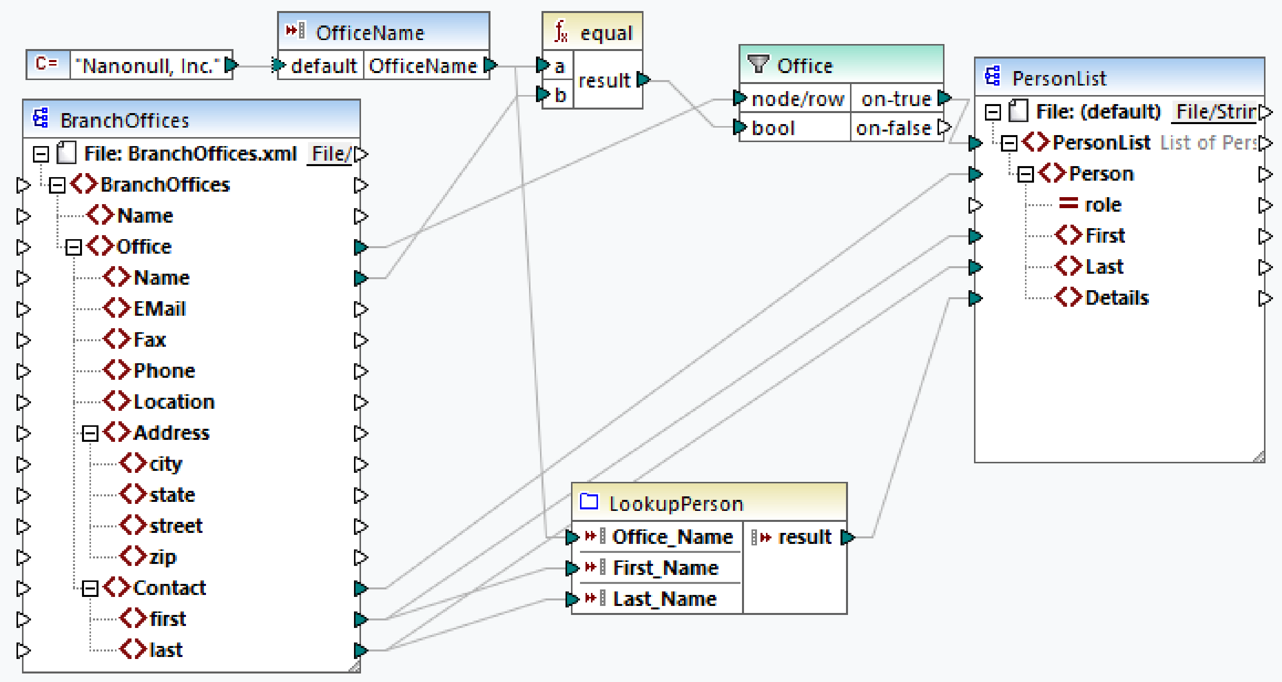
To fix this problem, connect the **Request** input of the web service to the result of the **sum** function. The function produces one single value, so the web service will also be called once:



Normally, aggregate functions like **sum**, **count**, etc produce a single value. Nevertheless, if there is a parent connection that allows it, they may produce a sequence of values as well, as described further in the Example: Changing the Parent Context 82.

5.9.4.5 Look-up Implementation

This topic explains how to look up data about employees and present this information in a suitable way. To test the look-up implementation, you will need the following mapping: **MapForceExamples\PersonListByBranchOffice.mfd**.



Goals
Our goals are as follows:

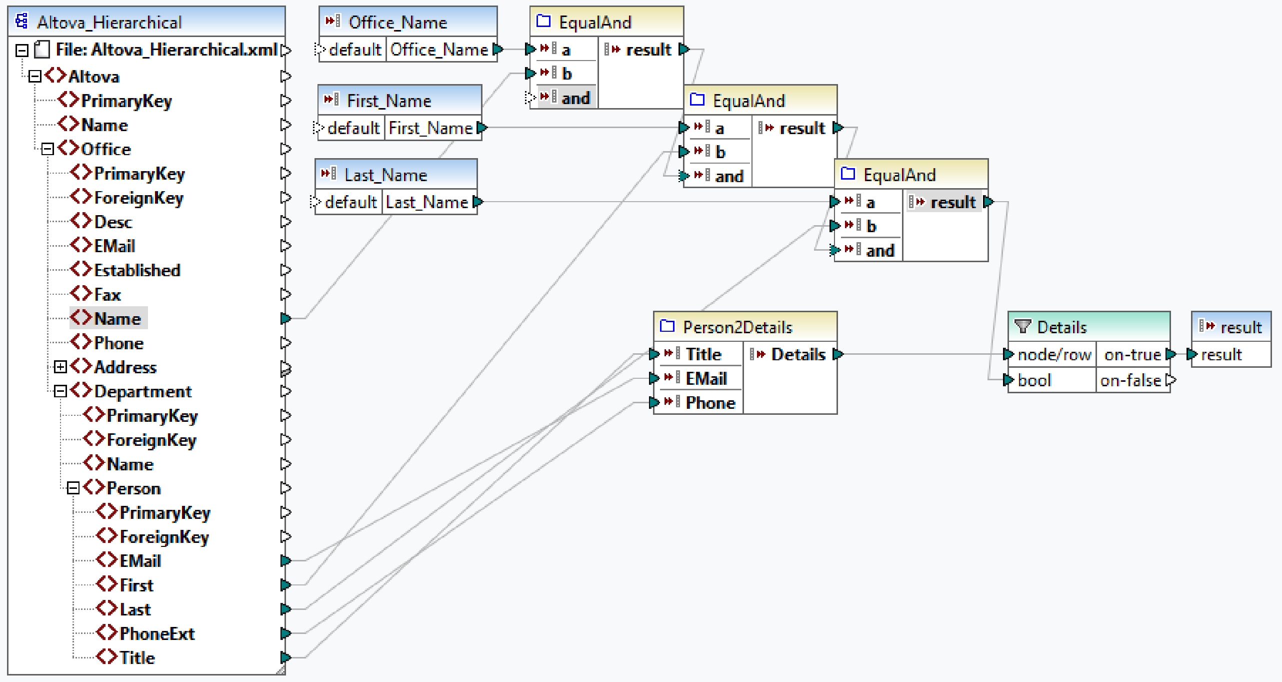
- To look up data about each employee (their phone extension, email address, and title) in a separate XML file.
- To present this data as a comma-separated list and map this list to the `Details` element of the target XML file.
- To extract information about employees only from one branch office called Nanonull, Inc.

To achieve these goals, we have designed our mapping in the following way:

- To filter only employees from Nanonull, Inc., the mapping uses the `Office` filter.
- To look up information about employees in a different XML file, the mapping calls the `LookupPerson` UDF. The implementation of this UDF is described in the subsection below.
- To process employee data, the `LookupPerson` function internally calls other functions that retrieve and concatenate information about each employee. All these operations are in the function's mapping and not visible in the main mapping. The `LookupPerson` function then maps the employee data to the `Details` element in `PersonList`.

LookupPerson: Internal structure

The look-up functionality is provided by the `LookupPerson` function, whose definition is illustrated below. To see the internal implementation of the UDF, double-click its header in the main mapping.

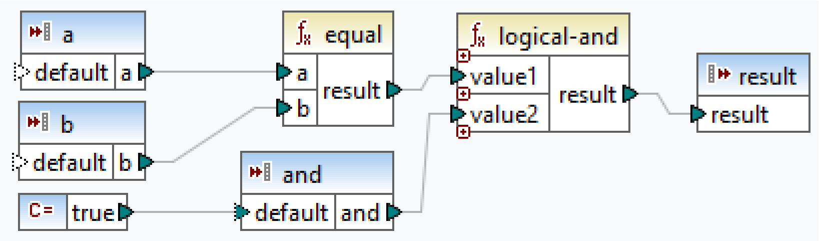


The UDF is defined as follows:

- The data is retrieved from the XML file **`Altova_Hierarchical.xml`**: (i) the name of the office and first and last names of employees, which are used to select employees only from Nanonull, Inc., and (ii) the email, title, and phone extension that are concatenated into one string. The definitions of the `EqualAnd` and `Person2Detail` functions are described below.
- The UDF also has three input parameters that provide the look-up values `Office_Name`, `First_Name`, and `Last_Name`. The value of the `Office_Name` parameter is retrieved from the `OfficeName` input from the main mapping, and the values of `First_Name` and `Last_Name` are supplied by the `BranchOffices` component from the main mapping.
- The value of the `EqaulAnd` function (`true` or `false`) is passed to the `Details` filter each time a new employee's details (title, email, phone) are processed. When the `Details` filter gets the value `true`, the look-up operation is successful and the employee's details are retrieved and returned to the main mapping. Otherwise, the next item in the context is examined, and this procedure continues until the loop finishes.

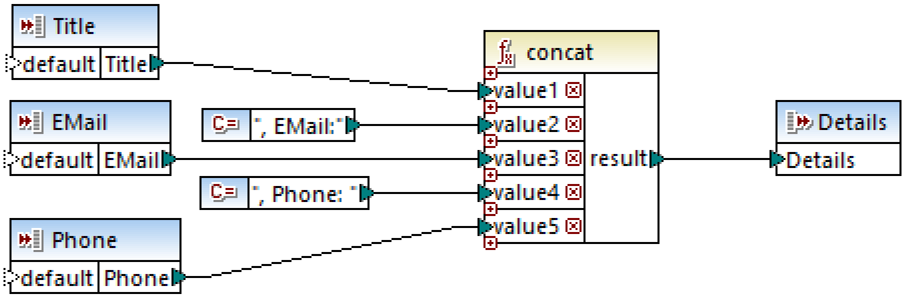
EqualAnd implementation

The `EqualAnd` function (*see below*) is a separate UDF defined inside the `LookupPerson` UDF. To see the internal structure of the `EqualAnd` UDF, double-click the function's header.



The EqualAnd UDF first checks whether a equals b; if the result is true, it is passed as the first parameter of the **logical-and** function. If both values are true in the **logical-and** function, the result is also true and is passed on to the next EqualAnd function. The result of the third EqaulAnd function (*see LookupPerson UDF above*) is passed on to the Details filter.

Person2Detail implementation
The Person2Details UDF is another function inside the LookupPerson UDF. The Person2Details UDF (*see below*) concatenates three values (retrieved from **Altova_Hierarchical.xml**) and two text constants.



Output

When you click the **Output** pane, MapForce will display first and last names, and details of employees only from Nanonull, Inc (*see extract below*).

```
<PersonList>
  <Person>
     <First>Vernon</First>
     <Last>Callaby</Last>
     <Details>Office Manager, EMail:v.callaby@nanonull.com, Phone: 582</Details>
  </Person>
  <Person>
     <First>Frank</First>
     <Last>Further</Last>
     <Details>Accounts Receivable, EMail:f.further@nanonull.com, Phone: 471</Details>
  </Person>
  ...
</PersonList>
```

5.9.5 Custom Functions

This section explains how to import custom XSLT 476, XQuery 483, Java and .NET 487 functions. The section also shows how to reference C#, C++ and Java libraries manually 494.

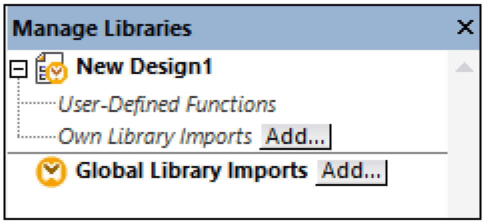
5.9.5.1 Import Custom XSLT Functions

You can extend the XSLT 1.0, XSLT 2.0, and XSLT 3.0 function libraries available in MapForce with your own custom functions, provided that your custom functions return simple types.

> Only custom functions that return simple data types (for example, strings) are supported.

To import functions from an XSLT file:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[23]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for the .xsl file that contains the functions, and click **Open**. A message box appears informing you that a new library has been added.

Imported XSLT files appear as libraries in the Libraries window, and display all named templates as functions below the library name. If you do not see the imported library, ensure you have selected XSLT as a transformation language[19]. See also Managing Function Libraries[438].

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[66].

Data types in XPath 2.0

If your XML document references an XML Schema and is valid according to it, you must explicitly construct or cast datatypes that are not implicitly converted to the required datatype by an operation.

In the XPath 2.0 Data Model used by the Altova XSLT 2.0 Engine, all **atomized** node values from the XML document are assigned the `xs:untypedAtomic` datatype. The `xs:untypedAtomic` type works well with implicit type conversions.

For example,

- the expression `xs:untypedAtomic("1") + 1` results in a value of 2 because the `xdt:untypedAtomic` value is *implicitly* promoted to `xs:double` by the addition operator.
- Arithmetic operators implicitly promote operands to `xs:double`.
- Value comparison operators promote operands to `xs:string` before comparing.

See also:

Example: Adding Custom XSLT Functions 477
Example: Summing Node Values 479
XSLT 1.0 engine implementation 1101
XSLT 2.0 engine implementation 1101

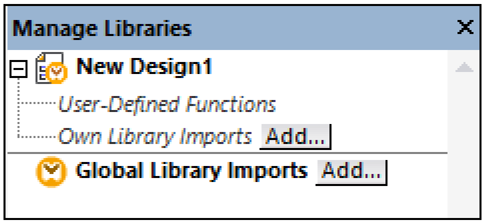
5.9.5.1.1 Example: Adding Custom XSLT Functions

This example illustrates how to import custom XSLT 1.0 functions into MapForce. The files needed for this example are available in the following folder: **`C:\Users\<username>\Documents\Altova\MapForce2025\MapForceExamples`**.

- `Name-splitter.xslt`. This XSLT file defines a named template called **tokenize** with a single parameter `string`. The template works through an input string and separates capitalized characters with a space for each occurrence.
- `Name-splitter.xml` (the source XML instance file to be processed)
- `Customers.xsd` (the source XML schema)
- `CompletePO.xsd` (the target XML schema)

To add a custom XSLT function:

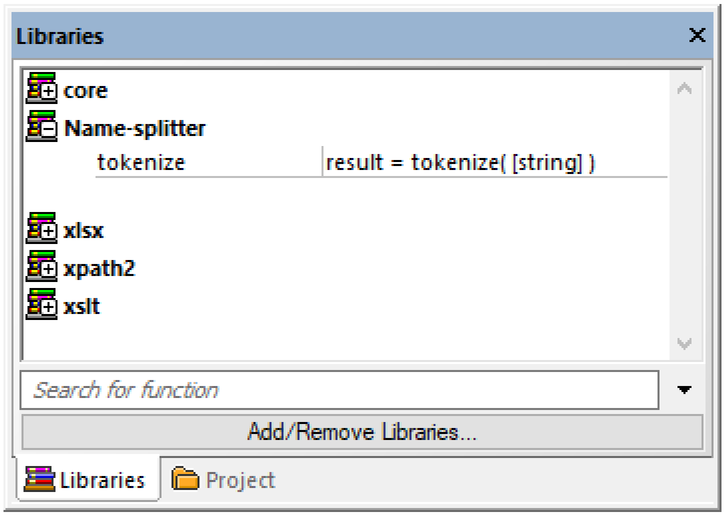
1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window 23. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be

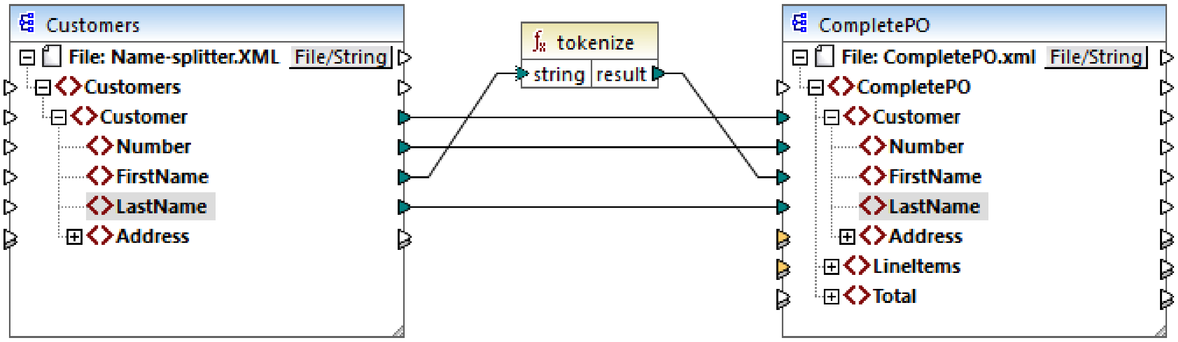
relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for the `.xsl` or `.xslt` file that contains the named template you want to act as a function, in this case `Name-splitter.xslt`, and click **Open**. A message box appears informing you that a new library has been added, and the XSLT file name appears in the **Libraries** window, along with the functions defined as named templates (in this example, **Name-splitter** with the **`tokenize`** function).

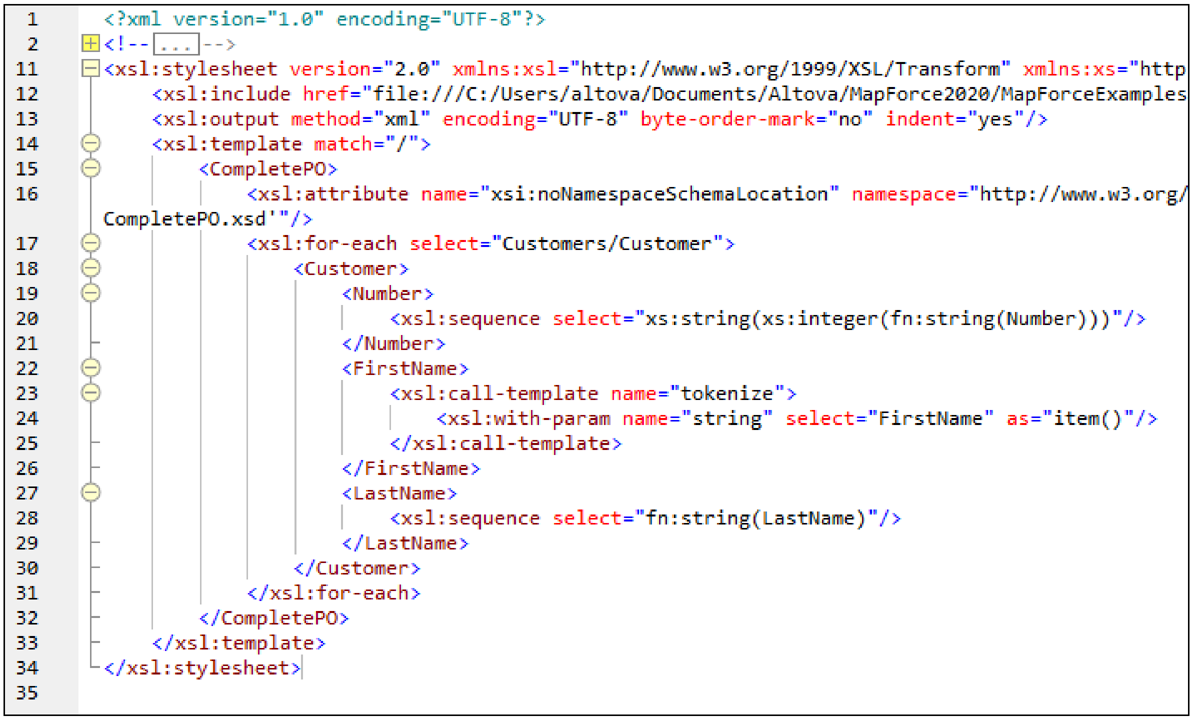


To use the XSLT function in your mapping:

1. Drag the **`tokenize`** function into the Mapping window and map the items as show below.



2. Click the **XSLT** pane to see the generated XSLT code.



```
<?xml version="1.0" encoding="UTF-8"?>
<!--[...]-->
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xs="http
    <xsl:include href="file:///C:/Users/altova/Documents/Altova/MapForce2020/MapForceExamples
    <xsl:output method="xml" encoding="UTF-8" byte-order-mark="no" indent="yes"/>
    <xsl:template match="/">
        <CompletePO>
            <xsl:attribute name="xsi:noNamespaceSchemaLocation" namespace="http://www.w3.org/
CompletePO.xsd'"/>
            <xsl:for-each select="Customers/Customer">
                <Customer>
                    <Number>
                        <xsl:sequence select="xs:string(xs:integer(fn:string(Number)))"/>
                    </Number>
                    <FirstName>
                        <xsl:call-template name="tokenize">
                            <xsl:with-param name="string" select="FirstName" as="item()"/>
                        </xsl:call-template>
                    </FirstName>
                    <LastName>
                        <xsl:sequence select="fn:string(LastName)"/>
                    </LastName>
                </Customer>
            </xsl:for-each>
        </CompletePO>
    </xsl:template>
</xsl:stylesheet>
```

Note: As soon as a named template is used in a mapping, the XSLT file containing the named template is **included** in the generated XSLT code (**xsl:include href...**), and is **called** using the command **xsl:call-template**.

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

To remove custom XSLT libraries from MapForce:

1. Click **Add/Remove Libraries** at the base of the Libraries window. The Manage Libraries window opens.
2. Click **Delete Library** ☒ next to the library that is to be deleted.

5.9.5.1.2 Example: Summing Node Values

This example shows you how to process multiple nodes of an XML document and have the result mapped as a single value to a target XML document. Specifically, the goal of the mapping is to calculate the price of all products in a source XML file and write it as a single value to an output XML file. The files used in this example are available in the **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\** folder:

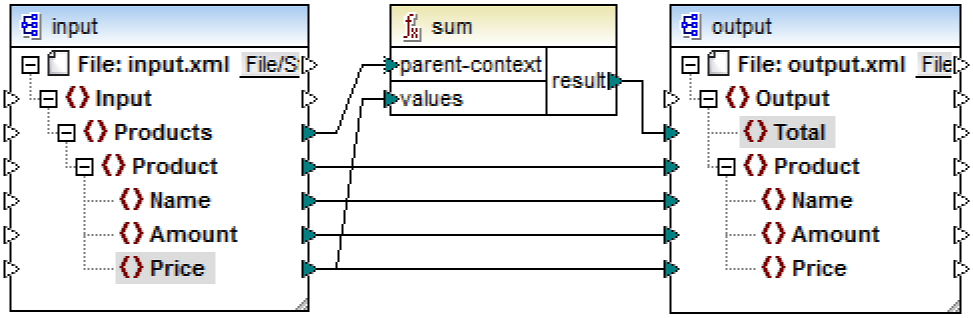
- **Summing-nodes.mfd** — the mapping file
- **input.xml** — the source XML file
- **input.xsd** — the source XML schema
- **output.xsd** — the target XML schema
- **Summing-nodes.xslt** — A custom XSLT stylesheet containing a named template to sum the individual nodes.

There are two different ways to achieve the goal of the mapping:

- By using the sum[521] function. This MapForce built-in function is available in the Libraries window.
- By importing a custom XSLT stylesheet into MapForce.

Solution 1: Using the "sum" aggregate function

To use the **sum** function in the mapping, drag it from the Libraries window into the mapping. Note that the functions available in the Libraries window depend on the XSLT language version you selected (XSLT 1 or XSLT 2). Next, create the mapping connections as shown below.



For more information about aggregate functions of the core library, see also core | aggregate functions[514].

Solution 2: Using a custom XSLT Stylesheet

As mentioned above, the aim of the example is to sum the Price fields of products in the source XML file, in this case products A and B.

```
<?xml version="1.0" encoding="UTF-8"?>
<Input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="input.xsd">
   <Products>
      <Product>
         <Name>ProductA</Name>
         <Amount>10</Amount>
         <Price>5</Price>
      </Product>
      <Product>
         <Name>ProductB</Name>
         <Amount>5</Amount>
         <Price>20</Price>
      </Product>
   </Products>
</Input>
```

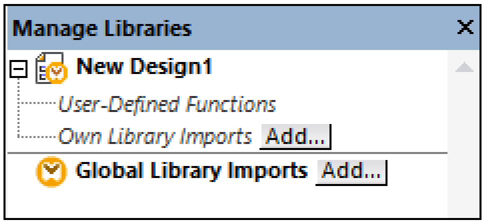
The code listing below shows a custom XSLT stylesheet which uses the named template "Total" and a single parameter string. The template works through the XML input file and sums all the values obtained by the XPath expression /Product/Price.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

   <xsl:template match="*">
      <xsl:for-each select=".">
      <xsl:call-template name="Total">
         <xsl:with-param name="string" select="."/>
      </xsl:call-template>
      </xsl:for-each>
   </xsl:template>

   <xsl:template name="Total">
   <xsl:param name="string"/>
      <xsl:value-of select="sum($string/Product/Price)"/>
   </xsl:template>
</xsl:stylesheet>
```

Note: To sum the nodes in XSLT 2.0, change the stylesheet declaration to `version="2.0"`.

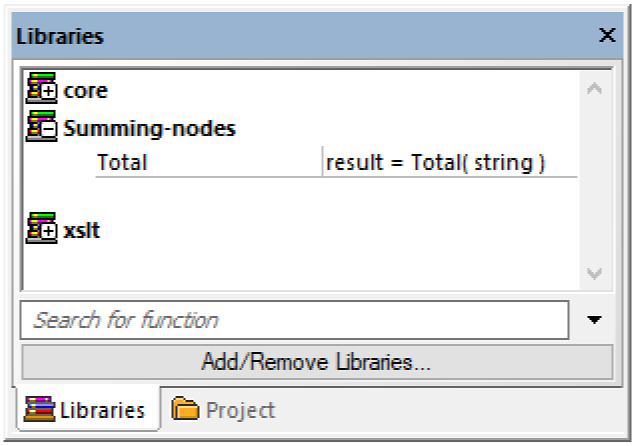
Before importing the XSLT stylesheet into MapForce, select XSLT 1.0 as a transformation language 19. You are now ready to import the custom function, as follows:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window 23. The **Manage Libraries** window opens (*see screenshot below*).

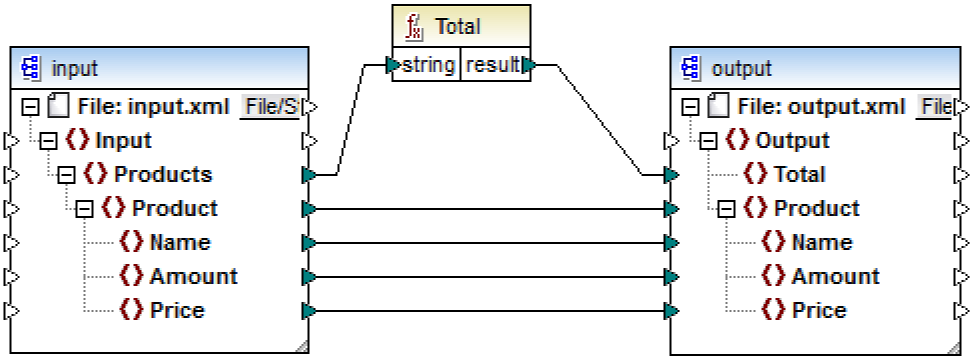


2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\Summing-nodes.xslt**, and click **Open**. A message box appears informing you that a new library has been added, and the new library appears in the Libraries window.



4. Drag the `Total` function from the Libraries into the mapping, and create the mapping connections as shown below.



To preview the mapping result, click the **Output** pane. 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>
```

5.9.5.2 Import Custom XQuery 1.0 Functions

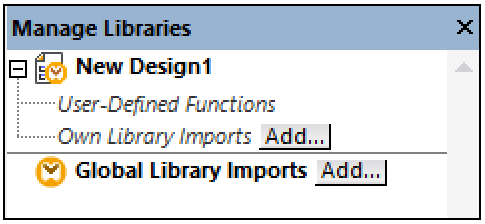
When XQuery is selected as mapping transformation language, MapForce displays the built-in function libraries available for XQuery in the Libraries window. If necessary, you can extend this list with custom XQuery functions, by importing custom XQuery 1.0 library modules into MapForce.

To be eligible for import into MapForce, an XQuery file must satisfy the following requirements:

- It must be a valid library module according to XQuery specification. In other words, it must start with a module declaration such as `module namespace <prefix>="<namespace name"`
- All functions declared in the imported library module must return atomic data types (for example, `xs:string`, `xs:boolean`, `xs:integer`, etc). Function parameters must also have atomic types.

To import an XQuery library module:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window 23. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for the .xq or .xquery library file, and click **Open**.

The imported library modules appear in the Libraries window, and then you can drag specific functions into the mapping area and use them like any other MapForce function component, see also Add a Function to the Mapping 436.

If you do not see the imported XQuery library module, make sure that XQuery is selected as a transformation language 19.

See also:

XQuery engine implementation 1103

5.9.5.2.1 Example: Import Custom XQuery Function

This example shows you how to import a demo XQuery library module into MapForce and call its functions from a mapping. The demo module in this example consists of only one function which calculates tax on decimal amounts as 20% of the amount. In a production scenario, an XQuery module may contain multiple functions.

> All functions declared in the XQuery module must return atomic types and their parameters must also be of atomic data types. Otherwise, the module is not eligible for import into MapForce.

You can find the demo XQuery module file at the following path relative to your personal "Documents" folder on the computer where MapForce is installed: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\module.xq**.

```
xquery version "1.0";

module namespace demo="http://www.altova.com/mapforce/demo";

declare function demo:calculatetax($val as xs:decimal) as xs:decimal {
   $val*0.2
};
```

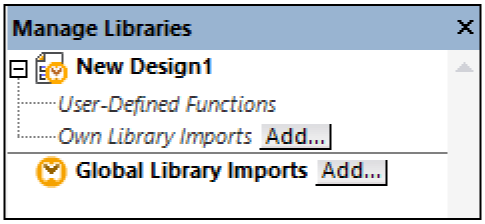
module.xq

After you import the XQuery module file into MapForce, you will be able to call the `demo:calculatetax` function from a mapping. Note that calculating the tax amount with the help of an XQuery function is just for demo purposes—you can achieve the same result by using MapForce built-in functions.

A demo mapping which calls the `demo:calculatetax` function above is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\CalculateTax_XQuery.mfd**. When you open this mapping initially, MapForce displays a warning that it contains one or more components that are not available in XQuery. This warning is normal and it occurs because the mapping references a function from a custom XQuery library module that was not imported yet. To remove the warning and run the mapping, we will import the missing XQuery module into MapForce as shown below.

To import the XQuery module into MapForce:

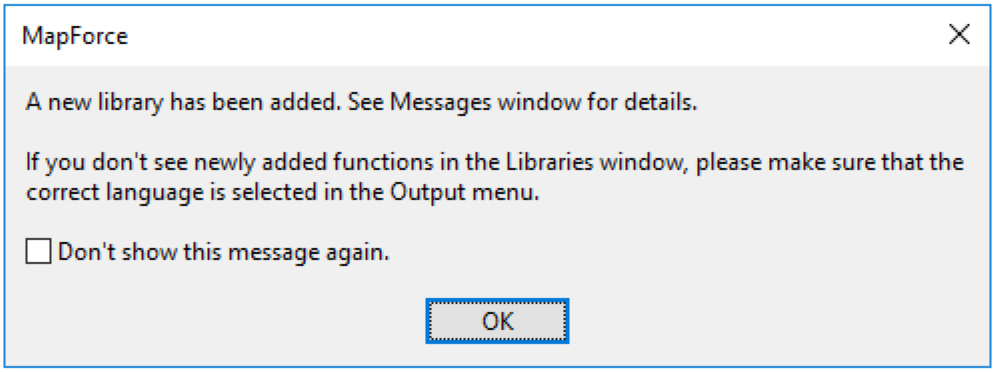
1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[23]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to

Global Library Imports. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Browse for **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\module.xq** and click **Open**. A message box appears informing you that a new library has been added.



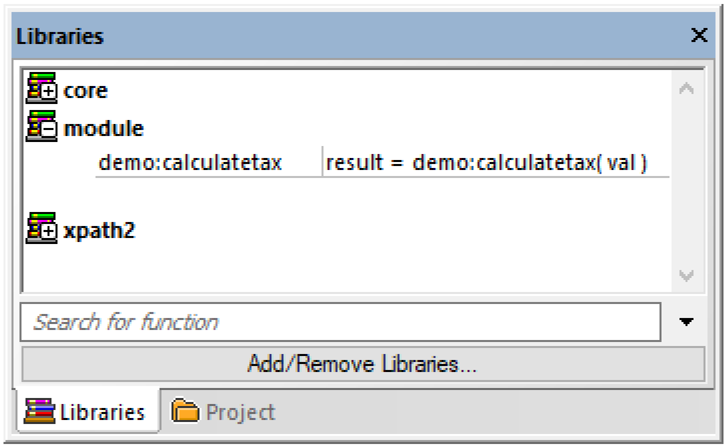
MapForce

A new library has been added. See Messages window for details.

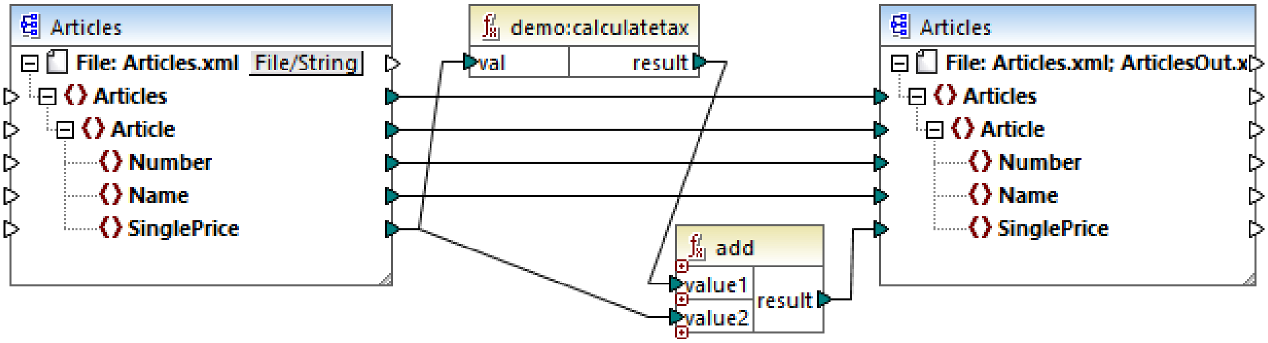
If you don't see newly added functions in the Libraries window, please make sure that the correct language is selected in the Output menu.

□ Don't show this message again.

The imported library and the **demo:calculatetax** function are now visible in the Libraries window.



Also, the mapping can now be validated and run without warnings. The **demo:calculatetax** function illustrated in the image below originates from the imported XQuery module, and it can be added to the mapping just like any other built-in function, see Add a Function to the Mapping 436.



CalculateTax_XQuery.mfd

The mapping explained

The **CalculateTax_XQuery.mfd** mapping illustrated above takes as input an XML file that stores articles. Each article has a single price expressed as a decimal value, for example:

```
<Articles>
   <Article>
      <Number>1</Number>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>
   <Article>
      <Number>2</Number>
      <Name>Socks</Name>
      <SinglePrice>2.30</SinglePrice>
   </Article>
   <Article>
      <Number>3</Number>
      <Name>Pants</Name>
      <SinglePrice>34</SinglePrice>
   </Article>
   <Article>
      <Number>4</Number>
      <Name>Jacket</Name>
      <SinglePrice>57.50</SinglePrice>
   </Article>
</Articles>
```

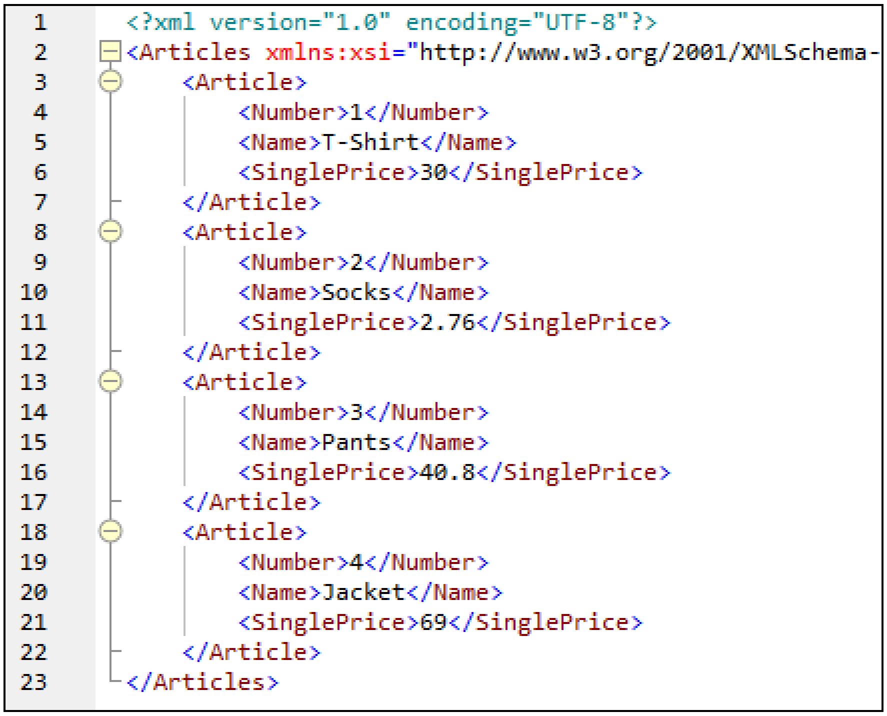
Articles.xml

The mapping produces an XML file which abides by the same schema as the source XML file. Therefore, both the source and target components have the same structure on the mapping. As the mapping connections suggest, nearly all elements are mapped in a straightforward way from target to source—for example, for each **Article** in the source there will be an **Article** in the target. The values of all items are copied verbatim from the source XML, except for **SinglePrice**. The value of **SinglePrice** is calculated with the help of two functions:

- The **demo:calculatetax** XQuery function calculates the tax amount by taking the original **SinglePrice** as input.
- The MapForce built-in **add** function adds the tax amount to the original **SinglePrice** amount and returns the final amount.

Importantly, the data type of the **SinglePrice** item is `xs:decimal`—which corresponds to the input parameter type and the return type of the XQuery function.

The output produced by the mapping when you click the **Output** pane is illustrated below. Notice the increase of 20% applied to each price compared to the source XML.



```
<?xml version="1.0" encoding="UTF-8"?>
<Articles xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    <Article>
        <Number>1</Number>
        <Name>T-Shirt</Name>
        <SinglePrice>30</SinglePrice>
    </Article>
    <Article>
        <Number>2</Number>
        <Name>Socks</Name>
        <SinglePrice>2.76</SinglePrice>
    </Article>
    <Article>
        <Number>3</Number>
        <Name>Pants</Name>
        <SinglePrice>40.8</SinglePrice>
    </Article>
    <Article>
        <Number>4</Number>
        <Name>Jacket</Name>
        <SinglePrice>69</SinglePrice>
    </Article>
</Articles>
```

5.9.5.3 Import Custom Java and .NET Libraries

This section explains how to import compiled Java class files and .NET DLL assemblies (including .NET 4.0 assemblies) into MapForce. If the imported libraries contain functions that use basic data types as parameters and return simple types, such functions appear in the **Libraries** window and can be used in mappings as any other function available in MapForce. The mapping output of imported Java and .NET functions can be previewed in the **Output** pane, and the functions are available in the generated code. To find out more about importing custom libraries, see the examples provided in Import Custom Java Class 491 and Import Custom .NET DLL Assembly 492.

Important:

- To import custom Java or .NET functions, you need compiled Java classes (`.class`) or .NET.dll assembly files. The import of Java `.jar` files or `.dll` files that are not a .NET assembly is not supported.

- .NET assembly files are supported when the mapping language is set to C#. The .NET assemblies may be written in .NET languages other than C# (e.g., C++.NET or VB.NET) if they use only the basic data types from the System Assembly as parameters and return types. For details, see .NET Function Support 489.

- If you want to use custom .NET functions in the built-in output preview (in the **Output** pane), these functions need to be compiled for .NET Framework 4.x or .NET Standard 2.0.

- Compiled Java class (`.class`) files are supported when the mapping language is set to Java. Java Runtime Environment 7 or later must be installed on your computer. Only specific types and members are supported (see Java function support 488).

- You cannot set the mapping language to C++ if the mapping uses imported Java `.class` or .NET DLL assemblies.

- You cannot set the mapping language to XSLT if the mapping uses imported Java `.class` or .NET DLL assemblies (a custom XSLT function that acts as an adapter would have to be written).

- The import of functions from native C++ DLLs is limited and requires a special approach. For more information, see Reference Java, C# and C++ Libraries Manually 494.

- All functions called from a MapForce mapping should return the same value each time the function is called with the same input parameters. The exact order and the number of times a function is called by MapForce is undefined.

- In the case of Java, the imported class files and their packages do not need to be added to the `CLASSPATH` variable since the Built-in execution engine and generated Java code will automatically add imported packages to the Java engine's classpath or to Ant, respectively. However, any dependencies of the imported class files and packages will not be handled automatically. Therefore, if imported Java class files or packages depend on other class files, make sure to add the parent directories of all dependent packages to the `CLASSPATH` environment variable.

Java function support

Top-level classes, static member classes and non-static member classes are supported:

- `new <classname>(<arg1>, <arg2>, ...)`
- `<object>.new <member-class>(<arg1>, <arg2>, ...)`

Member functions and static functions are supported:

- `<function>(<arg1>, <arg2>, ...)`
- `<object>.<method>(<arg1>, ...)`

Supported connections between XML Schema and Java types:

Schema type	Java type
xs:string	String
xs:byte	byte
xs:short	short

Schema type	Java type
`xs:int`	`int`
`xs:long`	`long`
`xs:boolean`	`boolean`
`xs:float`	`float`
`xs:double`	`double`
`xs:decimal`	`java.math.BigDecimal`
`xs:integer`	`java.math.BigInteger`

Connections in both directions are possible. Other Java types (including array types) are not supported. Methods using such parameters or return values, will be ignored. Object types are supported by calling their constructor or as a return value of a method. They can be mapped to other Java methods. Manipulating the object using MapForce means is not possible.

.NET function support

Top-level classes and member classes are supported:

- `new <classname>(<arg1>, <arg2>, ...)`

Member functions and static functions are supported:

- `<function>(<arg1>, <arg2>, ...)`
- `<object>.<method>(<arg1>, ...)`

Supported connections between XML Schema and .NET/C# types:

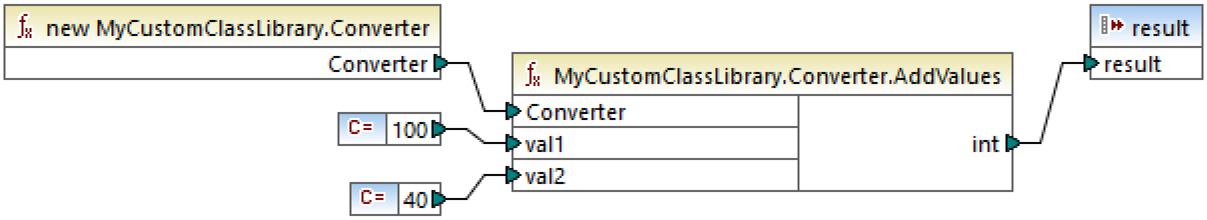
Schema type	.NET type	C# type
`xs:string`	`System.String`	`string`
`xs:byte`	`System.SByte`	`sbyte`
`xs:short`	`System.Int16`	`short`
`xs:int`	`System.Int32`	`int`
`xs:long`	`System.Int64`	`long`
`xs:unsignedByte`	`System.Byte`	`byte`
`xs:unsignedShort`	`System.UInt16`	`ushort`
`xs:unsignedInt`	`System.UInt32`	`uint`
`xs:unsignedLong`	`System.UInt64`	`ulong`
`xs:boolean`	`System.Boolean`	`bool`

Schema type	.NET type	C# type
`xs:float`	`System.Single`	`float`
`xs:double`	`System.Double`	`double`
`xs:decimal`	`System.Decimal`	`decimal`

Connections in both directions are possible. Other .NET/C# types (including array types) are not supported. Methods using such parameters or return values will be ignored. Object types are supported by calling their constructor or as a return value of a method. They can be mapped to other .NET methods. Manipulating the object using MapForce means is not possible.

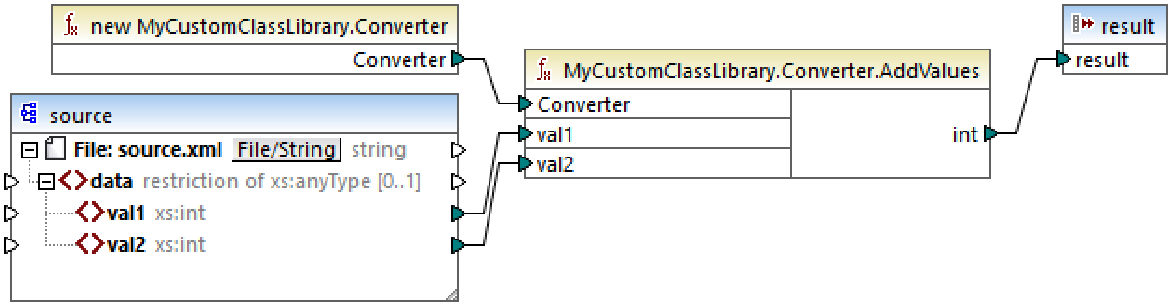
Data type issues and workarounds

When a function in your custom library expects integer types, connecting constants of type `Number` to the function's arguments may cause a type mismatch error similar to this one: `No match for MyCustomClassLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null.MyCustomClassLibrary.Converter.AddValues(MyCustomClassLibrary.Converter, xs:decimal, xs:decimal). Check argument types.` This issue is specific to constants of type `Number` only. A sample mapping that could generate this error is shown below. In this mapping, two constants of type `Number` are connected to the function's arguments of type `Integer`.


new MyCustomClassLibrary.Converter
Converter
C= 100
C= 40
MyCustomClassLibrary.Converter.AddValues
Converter
val1
val2
int
result
result

The possible workarounds are described below:

1. Change the constant type from **Number** to **All other**. You can do this after double-clicking the title bar of the constant component.
2. Instead of a constant, use a source component (for example, an XML file) that provides values of the data type expected by the function.


new MyCustomClassLibrary.Converter
Converter
source
File: source.xml File/String string
data restriction of xs:anyType [0..1]
val1 xs:int
val2 xs:int
MyCustomClassLibrary.Converter.AddValues
Converter
val1
val2
int
result
result

3. In your external code, create a wrapper function that accepts a decimal value and returns an integer value. The wrapper solution may be imported as a separate library. Therefore, you do not need to change the original source code of the target function to use this approach.

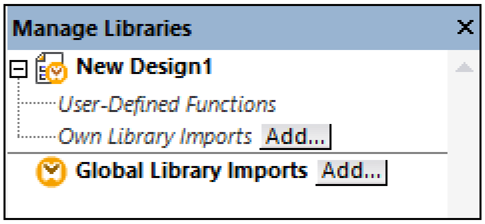
5.9.5.3.1 Example: Import Custom Java Class

This example shows how to import a custom Java `.class` file into MapForce.

Note: Java SE 8 Runtime Environment or later is required to complete this example.

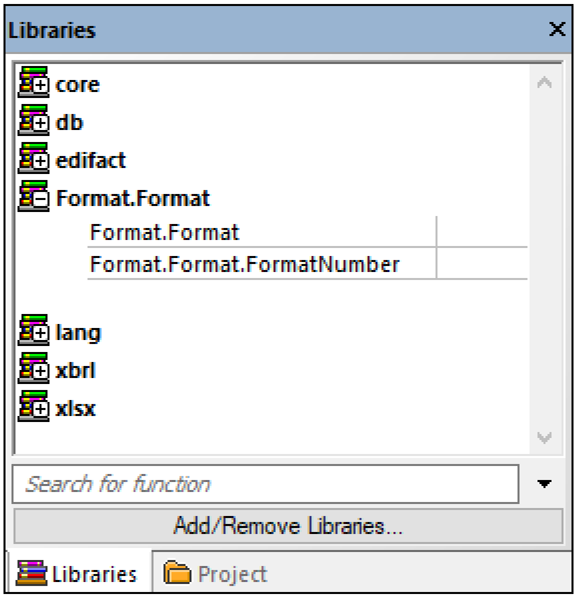
Java .class import
To add a Java `.class` file as a MapForce library, take the following steps:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[23]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

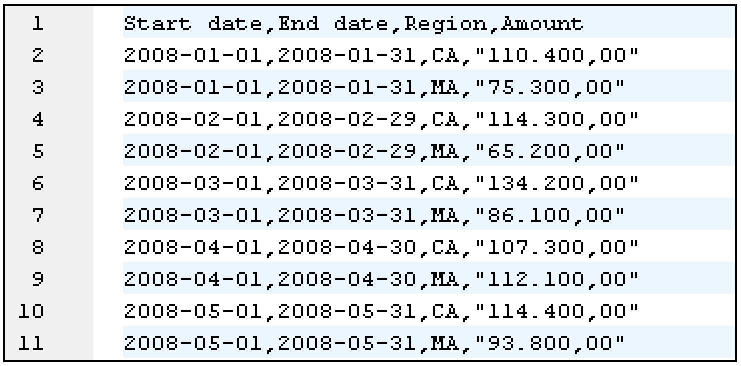
3. Search for the following file: `<Documents>\Altova\MapForce2025\MapForceExamples\Java\Format\Format.class`. A message appears informing you that a new library has been added. The imported library is now visible in the **Libraries** window (*see screenshot below*).



If you do not see the newly imported library in the **Libraries** window, make sure that the transformation language[19] is set to Java. To add the function to the mapping, drag it from the **Libraries** window into the mapping area. For details, see Add a Function to the Mapping[436].

Mapping output
To preview the mapping output in MapForce, take the following steps:

1. Open the following mapping: **`<Documents>\Altova\MapForce2025\MapForceExamples\Java\FormatNumber.mfd`**. This is a complete mapping that already imports the Java `.class` library mentioned above.
2. Click the **Output** button to see the result of the mapping (*see screenshot below*).



```
Start date,End date,Region,Amount
2008-01-01,2008-01-31,CA,"110.400,00"
2008-01-01,2008-01-31,MA,"75.300,00"
2008-02-01,2008-02-29,CA,"114.300,00"
2008-02-01,2008-02-29,MA,"65.200,00"
2008-03-01,2008-03-31,CA,"134.200,00"
2008-03-01,2008-03-31,MA,"86.100,00"
2008-04-01,2008-04-30,CA,"107.300,00"
2008-04-01,2008-04-30,MA,"112.100,00"
2008-05-01,2008-05-31,CA,"114.400,00"
2008-05-01,2008-05-31,MA,"93.800,00"
```

Mapping in Java
To run the mapping in Java, follow the instructions below:

1. Click **Generate Code In | Java** in the **File** menu.
2. Select a target directory where the code should be generated and click **OK**.
3. Import the generated libraries into your Java project and build the Java application. For more information, see Example: Generate and Run Java Code.

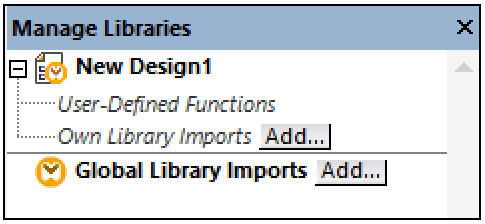
5.9.5.3.2 Example: Import Custom .NET DLL Assembly

This example shows how to import a custom .NET DLL assembly created in C# into MapForce. The source code of this sample is available at the following path: **`<Documents>\Altova\MapForce2025\MapForceExamples\C#\Format`**. The `.dll` assembly file that will be imported into MapForce is in the **`..\bin\Debug`** directory. You can also open the `.sln` solution file in Visual Studio and compile a new `.dll` file.

Note: If you want to use custom .NET functions in the built-in output preview (in the **Output** pane), these functions need to be compiled for .NET Framework 4.x or .NET Standard 2.0.

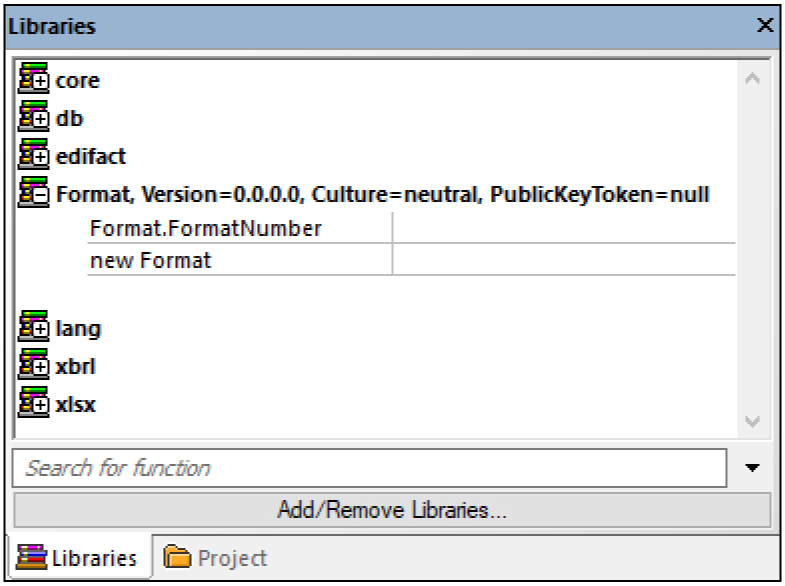
.NET assembly import
To import a .NET assembly file, take the following steps:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window[23]. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

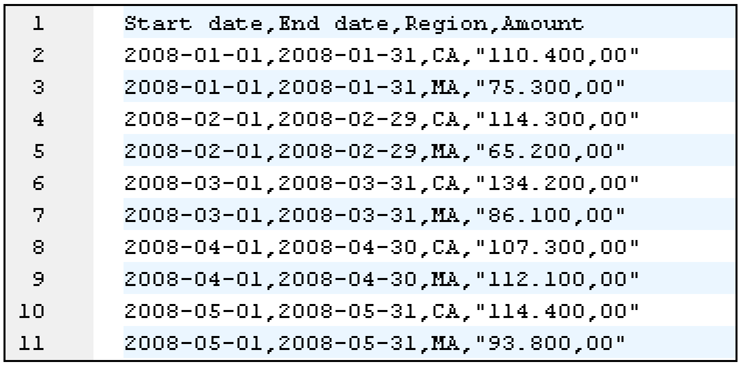
3. Search for **`Format.dll`** in the following directory: **`...\MapForceExamples\C#\Format\bin\Debug\`**. A message appears informing you that a new library has been added. The imported library is now visible in the **Libraries** window.



If you do not see the newly imported library in the **Libraries** window, make sure that the transformation language[19] is set to C#. To add the function to the mapping, drag it from the **Libraries** window into the mapping area. For more information, see Add a Function to the Mapping[436].

Mapping output
To preview the mapping output, take the following steps:

1. Open the **`FormatNumber.mfd`** file available in the following folder: **`...\MapForceExamples\C#`**. This is a sample mapping that has an imported `.dll` library mentioned above.
2. Click the **Output** button to see the result of the mapping (*see screenshot below*).



```
Start date,End date,Region,Amount
2008-01-01,2008-01-31,CA,"110.400,00"
2008-01-01,2008-01-31,MA,"75.300,00"
2008-02-01,2008-02-29,CA,"114.300,00"
2008-02-01,2008-02-29,MA,"65.200,00"
2008-03-01,2008-03-31,CA,"134.200,00"
2008-03-01,2008-03-31,MA,"86.100,00"
2008-04-01,2008-04-30,CA,"107.300,00"
2008-04-01,2008-04-30,MA,"112.100,00"
2008-05-01,2008-05-31,CA,"114.400,00"
2008-05-01,2008-05-31,MA,"93.800,00"
```

Mapping in C#

To run the mapping from a custom C# application, follow the instructions below:

1. Click **Generate Code In | C#** in the **File** menu.
2. Select a target directory where the code should be generated and click **OK**.
3. Build the application with Visual Studio and run the generated console application. For more details, see Generating C# code.

5.9.5.4 Reference C#, C++ and Java Libraries Manually

This section explains how to reference custom libraries in a `.mff` file (MapForce Function File). The `.mff` file containing the reference can then be imported as a MapForce library. A `.mff` file is an XML file in which you manually define the link between class definitions in your custom code and MapForce. Once you create a custom `.mff` file, you can import it into MapForce, which is similar to importing a .NET DLL or Java class file.

Important:

- If you want to use custom .NET functions in the built-in output preview (in the **Output** pane), these functions need to be compiled for .NET Framework 4.x or .NET Standard 2.0.

- You can import a function into MapForce only if its return type and parameters are of simple type. To find out more about a list of data types available for each language, see Data Type Mapping 500.

- When you import function libraries from custom `.mff` files, the preview of the mapping directly in MapForce (in the **Output** pane) is limited. For libraries written in C++, the preview of the mapping in MapForce is not supported. For Java and C#, the preview is available when your library uses native language types, but it is not available if your library imports the Altova generated classes. However, you can generate code in the specific language targeted by your library. The custom functions will be available in the generated code, enabling you to run the mapping from the generated code.

- The exact order in which functions are called by the generated mapping code is undefined. MapForce may need to cache calculated results for reuse or evaluate expressions in any order. Therefore, it is recommended to use only custom functions that have no side effects.

- It is important to distinguish between user-defined functions and custom function libraries. User-defined functions are created graphically in a mapping; they cannot and need not be saved to a `.mff` file, because they are saved together with the `.mfd` file in which they are created. For more information, see Call and Import UDFs[461].

- If you are upgrading from a MapForce version earlier than 2010, you may need to update the data types used in your custom functions. For details, see Data Type Mapping[500].

To find out more about how to create and configure a custom `.mff` file, see Configure .mff File[495]. The examples are provided in the following topics:

- Reference C# in .mff[503]
- Reference C++ in .mff[504]
- Reference Java in .mff[506]

5.9.5.4.1 Configure .mff File

This topic provides instructions on how to configure a MapForce Function File file (`.mff`). A `.mff` file is a configuration file in XML format that allows importing functions from custom Java, C#, or C++ libraries into MapForce so that they appear in the **Libraries** window. A `.mff` file is an intermediary between your custom libraries and MapForce. The `.mff` file must be configured to specify i) the interfaces for the custom functions and ii) where the implementation can be found in the generated code.

Important:

- The `*.mff` library files must be valid against the following schema: **`C:\Program Files\MapForceLibraries\mff.xsd`**. The `mff.xsd` schema defines the custom library configuration and is for internal use only. Altova GmbH retains the right to change this file format with new releases.
- It is only possible to define one C#, C++, or Java class per `.mff` file.

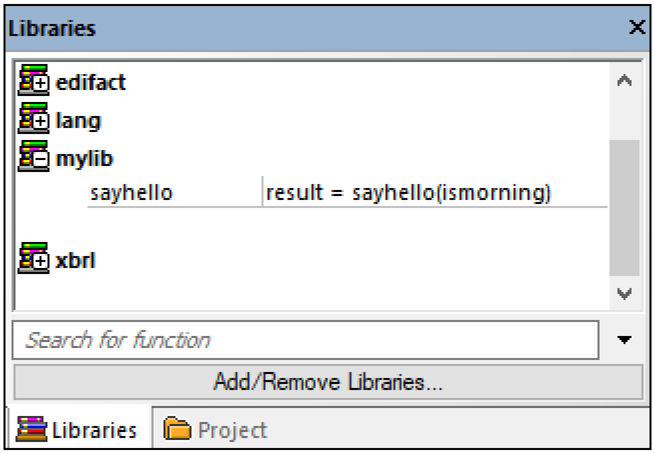
Sample .mff for C#

The following code listing illustrates a sample `.mff` file for C++:

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
   <implementations>
      <implementation language="cpp">
         <setting name="namespace" value="mylib"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Libraries\cpp"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
      </implementation>
   </implementations>
   <group name="greetings">
      <component name="sayhello">
         <sources>
            <datapoint name="ismorning" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="cpp">
               <function name="SayHello"/>
            </implementation>
         </implementations>
         <description>
            <short>result = sayhello(ismorning)</short>
            <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
         </description>
      </component>
   </group>
</mapping>
```

Imported custom library

The image below shows how a custom `.mff` file may look after being imported into MapForce. Notice that the custom library `mylib` appears as a library entry (sorted alphabetically), containing the `sayhello` string function.



Configuration steps

To configure the `.mff` file, follow the instructions below.

Step 1. Configure the library name

The library name can be found in the `.mff` file (*see below*). By convention, library names are written in lowercase in MapForce; however, you can also use uppercase letters.

```
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
```

In the sample above, the entry that will appear in the **Libraries** window is called `mylib`.

Step 2. Configure the language implementations

The `<implementations>` element is a mandatory element which specifies which languages your library should support, and it must be added as a child element of `<mapping>` (*see example below*).

```
<!-- ... -->
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
   <implementations>
      <implementation language="cpp">
         <setting name="namespace" value="mylib"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Libraries\cpp"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
      </implementation>
   </implementations>
<!-- ... -->
```

The settings within each `<implementation>` element allow the generated code to call the specific functions defined in Java, C++ or C#. A `.mff` file can be written so that it targets more than one programming language. In this case, every additional language must contain an additional `<implementation>` element. The specific settings for each programming language are discussed below.

Java library reference

```
<!-- ... -->
<implementation language="java">
   <setting name="package" value="com.hello.functions"/>
   <setting name="class" value="Greetings"/>
</implementation>
<!-- ... -->
```

It is important for the generated code to be able to find your **`Greetings.class`** file. Therefore, make sure to add a reference to your class to the Java class path.

C# library reference

```
<!-- ... -->
      <implementation language="cs">
         <setting name="namespace" value="MyLibrary" />
         <setting name="class" value="Greetings" />
         <setting name="reference" value="C:
\Libraries\cs\MyLibrary\bin\debug\MyLibrary.dll" />
      </implementation>
<!-- ... -->
```

For C#, it is important that the namespace in the code should correspond to the namespace defined in the `.mff` file (in the code listing above, the namespace is **`MyLibrary`**). The same is true for the class name (in the code listing above, the class name is **`Greetings`**). The third setting, `reference`, provides the path of the `dll` that is to be linked to the generated code.

C++ library reference

```
<!-- ... -->
   <implementation language="cpp">
         <setting name="namespace" value="MyLibrary"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Libraries\cpp"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
```

```
      </implementation>
   <!-- ... -->
```

For C++, note the following:

- namespace is the namespace in which your Greetings class will be defined. It must be equal to the library attribute in mapping element.
- path is the path in which the include and the source files are to be found.
- When code for a mapping is generated, the include and source files will be copied to the directory **targetdir/libraryname**, which is defined when you select the menu command **File | Generate code in | C++**, and included in the project file.

All the include files you supply will be included in the generated algorithm.

Step 3. Add a component

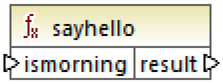
In the **Libraries** window, each function appears nested under a function group, for example, string functions. In the .mff file, a function corresponds to a <component> element. Conversely, each <component> must be nested under a <group> element, for example:

```
<!-- ... -->
<group name="string functions">
   <component name="sayhello">
   <!-- ... -->
   </component>
</group>
<!-- ... -->
```

The code shown below defines a sample function (component) called **sayhello**.

```
<!-- ... -->
<component name="sayhello">
   <sources>
      <datapoint name="ismorning" type="xs:boolean"/>
   </sources>
   <targets>
      <datapoint name="result" type="xs:string"/>
   </targets>
   <implementations>
   <!-- ... -->
   </implementations>
   <description>
     <short>result = sayhello(ismorning)</short>
     <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
   </description>
</component>
<!-- ... -->
```

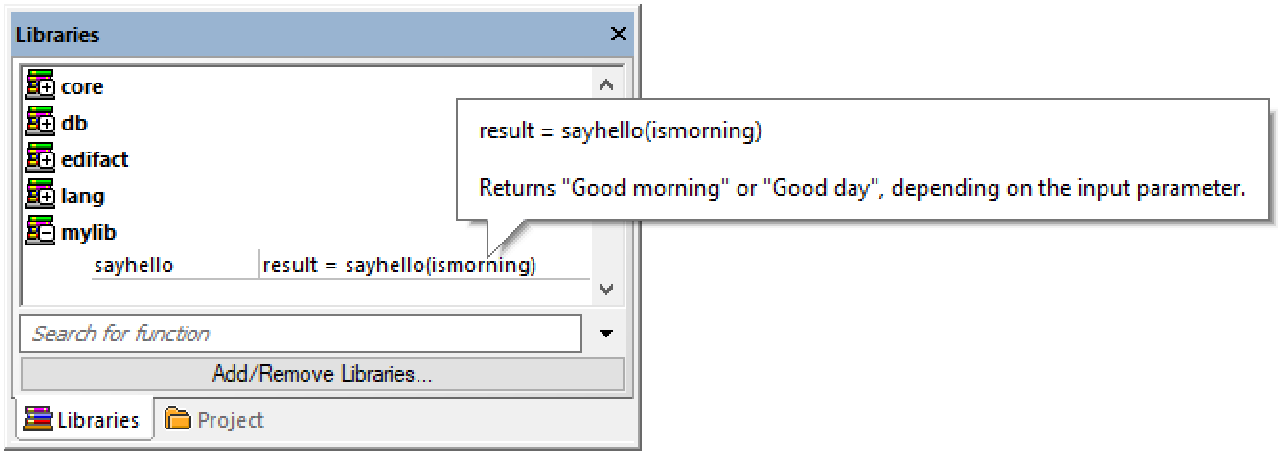
This is how the component above would look in MapForce:



In the code listing above, the `<datapoint>` element can be loosely defined as the input or output parameter of a function (also known as an input or output connector). The `type` argument of the `<datapoint>` element specifies the data type of the parameter or the data type of the return value. Only one target datapoint is allowed for each function. The number of source datapoints you can define is not limited.

The data type of each datapoint must be one of the XML Schema types (e.g., `xs:string`, `xs:integer`, etc.) These data types must correspond to the data types of the function's parameters you defined in your Java, C++ or C# library. To find out more about mapping of XML Schema datatypes to language types, see Data Type Mapping 500.

Functions are accompanied by short and long descriptions in the **Libraries** window. The short description is always shown to the right of the function name, while the long description is displayed as a tooltip when you place the mouse cursor over the short description (*see screenshot below*).



result = sayhello(ismorning)

result = sayhello(ismorning)

Returns "Good morning" or "Good day", depending on the input parameter.

Step 4. Define language implementations
We can now connect the function in the **Libraries** window with the function in the custom Java, C# or C++ classes. This is achieved through the `<implementation>` element. One function may have multiple `<implementation>` elements—one for each supported programming language. A function may be called `Hello` in Java or `SayHello` in C++. This is why you need to specify a separate function name for each programming language. A function for each of the three programming languages might look as follows:

```
<!-- ... -->
<component name="sayhello">
<!-- ... -->
   <implementations>
      <implementation language="cs">
         <function name="HelloFunction"/>
      </implementation>
      <implementation language="java">
         <function name="Hello"/>
      </implementation>
      <implementation language="cpp">
         <function name="SayHello"/>
      </implementation>
   </implementations>
<!-- ... -->
```

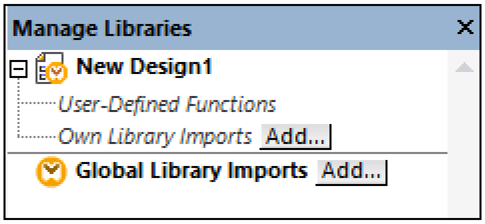
```
</component>
<!-- ... -->
```

The value you supply as a function name must match the name of the method in the Java, C# or C++ class.

5.9.5.4.2 Import .mff Libraries

After you have created 495 a custom `.mff` file, you can import it into MapForce as follows:

1. Click the **Add/Remove Libraries** button at the bottom of the Libraries window 23. The **Manage Libraries** window opens (*see screenshot below*).



2. To import functions as a *local* library (in the scope of the current document only), click **Add** under the current mapping name. To import functions as a *global* library (at program level), click **Add** next to **Global Library Imports**. When you import a library *locally*, you can set the path of the library file to be relative to the mapping file. With globally imported libraries, the path of the imported library is always absolute.

3. Search for the custom `.mff` file and click **Open**.

The imported library becomes visible in the **Libraries** window after you set the mapping language to a language targeted by the custom library.

If you save the `*.mff` file in **`...\Altova\MapForce2025\MapForceLibraries`**, which is relative to the **Program Files** (or **Program Files (x86)** folder), the library is automatically loaded into the **Libraries** window when you start MapForce. Libraries and their functions can be toggled on or off, by deleting or adding the respective library file (`*.mff`).

5.9.5.4.3 Data Type Mapping

The table below lists the data types supported as function return types and parameter types when you create custom `.mff` files that reference your Java, C#, and C++ libraries. The table lists both native and non-native data types. If you need support for non-native data types such as Altova date, time and duration types, your custom Java and C# libraries must include a reference to the Altova libraries. In the case of C++, the Altova libraries must always be imported. For information about how to generate the Altova libraries, see Code Generator 868.

XML Schema Type	Java Type	C# Type	C++ Type
anyAtomicType	String	string	string_type

XML Schema Type	Java Type	C# Type	C++ Type
anySimpleType	String	string	string_type
anyURI	String	string	string_type
base64Binary	byte[]	byte[]	altova::mapforce::blob
boolean	boolean	bool	bool
byte	int	int	int
date	com.altova.types.DateTime 959	Altova.Types.DateTime 944	altova::DateTime 929
dateTime	com.altova.types.DateTime 959	Altova.Types.DateTime 944	altova::DateTime 929
dayTimeDuration	com.altova.types.Duration 963	Altova.Types.Duration 949	altova::Duration 932
decimal	java.math.BigDecimal	decimal	double
double	double	double	double
duration	com.altova.types.Duration 963	Altova.Types.Duration 949	altova::Duration 932
ENTITIES	String	string	string_type
ENTITY	String	string	string_type
float	double	double	double
gDay	com.altova.types.DateTime 959	Altova.Types.DateTime 944	altova::DateTime 929
gMonth	com.altova.types.DateTime 959	Altova.Types.DateTime 944	altova::DateTime 929
gMonthDay	com.altova.types.DateTime 959	Altova.Types.DateTime 944	altova::DateTime 929
gYear	com.altova.types.DateTime 959	Altova.Types.DateTime 944	altova::DateTime 929
gYearMonth	com.altova.types.DateTime 959	Altova.Types.DateTime 944	altova::DateTime 929
hexBinary	byte[]	byte[]	altova::mapforce::blob
ID	String	string	string_type
IDREF	String	string	string_type

XML Schema Type	Java Type	C# Type	C++ Type
IDREFS	String	string	string_type
int	int	int	int
integer	java.math.BigInteger	decimal	__int64
language	String	string	string_type
long	long	long	__int64
Name	String	string	string_type
NCName	String	string	string_type
negativeInteger	java.math.BigInteger	decimal	__int64
NMTOKEN	String	string	string_type
NMTOKENS	String	string	string_type
nonNegativeInteger	java.math.BigInteger	decimal	unsigned __int64
nonPositiveInteger	java.math.BigInteger	decimal	__int64
normalizedString	String	string	string_type
NOTATION	String	string	string_type
positiveInteger	java.math.BigInteger	decimal	unsigned __int64
QName	javax.xml.namespace.QName	Altova.Types.QName	altova::QName
short	int	int	int
string	String	string	string_type
time	com.altova.types.DateTime 959	Altova.Types.DateTime 944	altova::DateTime 929
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 963	Altova.Types.Duration 949	altova::Duration 932

5.9.5.4.4 Reference C# Library in .mff

This example shows how to create a sample C# library and reference it in a MapForce Function File (`.mff`). The `.mff` file can then be imported as a MapForce library. Referencing a C# library in a `.mff` file is one of the ways to import C# libraries into MapForce. A simpler alternative is to import .NET assemblies directly. For more information, see Example: Import Custom .NET DLL Assembly 492.

Configuration steps

To reference a C# library in a `.mff` file, follow the instruction below.

Note: If you want to use custom .NET functions in the built-in output preview (in the **Output** pane), these functions need to be compiled for .NET Framework 4.x or .NET Standard 2.0.

Step 1. Create a new class library in VS

Create a new class library project in Visual Studio. Notice that the function has been defined as `public static`.

```
namespace MyLibrary
{
    public class Greetings
    {
        public static string SayHello(bool isMorning)
        {
            if (isMorning)
                return "Good morning!";
            return "Good Day!";
        }
    }
}
```

Step 2. Add a reference to Altova.dll

If you need special XML Schema types (such as date and duration), add a reference from your Visual Studio project to the `Altova.dll` library. To obtain this library, generate C# code from a mapping without custom functions. The `Altova.dll` file will be located in the **`..\Altova\bin\debug`** directory relative to the directory where the code was generated. To add the reference to `Altova.dll` in Visual Studio, click **Add Reference** in the **Project** menu and search for `Altova.dll`. Then add the line **`using Altova.Types;`** to your code. For information about how XML Schema types map to C# types, see Data Type Mapping 500.

Step 3. Build your VS project

Build your Visual Studio project. The `MyLibrary.dll` file is generated in your project output directory.

Step 4.Create .mff and reference your C# library

Using an XML editor, create a new `.mff` file and validate it against the following schema: **`C:\Program Files\MapForceLibraries\mff.xsd`**. Make sure that all references under **`implementation language="cs"`** point to the correct C# members and paths created previously. The line **`function name="SayHello"`** must refer to the function name exactly as it was defined in C#. For details, see Configure .mff File 495.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
```

```
   <implementations>
      <implementation language="cs">
         <setting name="namespace" value="MyLibrary" />
         <setting name="class" value="Greetings" />
         <setting name="reference" value="C:
\Libraries\cs\MyLibrary\bin\debug\MyLibrary.dll" />
      </implementation>
   </implementations>
   <group name="string functions">
      <component name="sayhello">
         <sources>
            <datapoint name="ismorning" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="cs">
               <function name="SayHello"/>
            </implementation>
         </implementations>
         <description>
            <short>result = sayhello(ismorning)</short>
            <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
         </description>
      </component>
   </group>
</mapping>
```

Step 5. Import .mff as a library
Now that your custom library is referenced in the `.mff` file, you can import the `.mff` file into MapForce as a library. For more information, see Import.mff Libraries 500.

5.9.5.4.5 Reference C++ in .mff

This example shows how to create a sample C++ library and reference it in a MapForce Function File (`.mff`). The `.mff` file can then be imported as a MapForce library.

Configuration steps

To reference a C++ library in a `.mff` file, follow the instructions below.

Step 1. Create a header file
Create a header (`.h`) file for your class library. The following code listing illustrates a sample header file called **`Greetings.h`**.

```
#ifndef MYLIBRARY_GREETINGS_H_INCLUDED
#define MYLIBRARY_GREETINGS_H_INCLUDED

#if _MSC_VER > 1000
#pragma once
```

```
#endif // _MSC_VER > 1000

using namespace altova;

namespace mylib {

class ALTOVA_DECLSPECIFIER Greetings
{
   public:
    static string_type SayHello(bool isMorning);
};

} // namespace mylib

#endif // MYLIBRARY_GREETINGS_H_INCLUDED
```

Notice that the function is declared as static and that the namespace `altova` is imported. Remember to write `ALTOVA_DECLSPECIFIER` in front of the class name: this ensures that your classes will be compiled correctly—whether you use dynamic or static linkage in the generated code.

Step 2. Create a .cpp file

Create a `.cpp` file with the same name as the header file. The `.cpp` file must be in the same directory as the `.h` file. The following code listing illustrates a sample `.cpp` file called **`Greetings.cpp`** that includes the **`Greetings.h`** file created previously:

```
#include "StdAfx.h"
#include "../Altova/Altova.h"
#include "../Altova/AltovaException.h"
#include "../Altova/SchemaTypes.h"

#include "Greetings.h"

namespace mylib {

   string_type Greetings::SayHello(bool isMorning)
   {
      if( isMorning )
         return _T("Good morning!");
      return _T("Good day!");
   }
}
```

Notice the lines that import **`StdAfx.h`** and several Altova libraries. These lines must be left unchanged. If the paths to the Altova libraries are correct in the generated code, these paths will point to the respective files. In contrast to Java or C#, you do not need to compile your source C++ files. They will be copied to the generated code and compiled with the rest of the generated mapping code.

Step 3. Create .mff and reference your C++ library

Using an XML editor, create a new `.mff` file and validate it against the following schema: **`C:\Program Files\MapForceLibraries\mff.xsd`**. Make sure that the namespace, function names and data types defined here correspond to those in the C++ code, as described in Configure .mff File 495. For information about data type support, see Data Type Mapping 500.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="mylib" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
   <implementations>
      <implementation language="cpp">
         <setting name="namespace" value="mylib"/>
         <setting name="class" value="Greetings"/>
         <setting name="path" value="C:\Libraries\cpp"/>
         <setting name="include" value="Greetings.h"/>
         <setting name="source" value="Greetings.cpp"/>
      </implementation>
   </implementations>
   <group name="greetings">
      <component name="sayhello">
         <sources>
            <datapoint name="ismorning" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="cpp">
               <function name="SayHello"/>
            </implementation>
         </implementations>
         <description>
            <short>result = sayhello(ismorning)</short>
            <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
         </description>
      </component>
   </group>
</mapping>
```

Step 4. Import .mff as a library
Now that your custom library is referenced in the `.mff` file, you can import the `.mff` file into MapForce as a library. For more information, see Import .mff File 500.

C++ compiler errors
In order to execute mappings that use native C++ libraries, you will need to generate C++ code and run the mapping from your C++ code or application, as described in Generating C++ code. If you get a compiler error in **`#import "msado15.dll" rename("EOF", "EndOfFile")`**, modify the project properties to include a reference to **`msado15.dll`** in **`C:\Program Files\Common Files\System\ADO`**.

5.9.5.4.6 Reference Java in .mff

This example shows how to create a sample Java library and reference it in a MapForce Function File (`.mff`). The `.mff` file can then be imported as a MapForce library. Referencing a Java library in a `.mff` file is one of the ways to import Java libraries into MapForce. A simpler alternative is to import Java `.class` files directly. For more information, see Example: Import Custom Java Class 491.

Configuration steps

To reference a C# library in a `.mff` file, follow the instruction below.

Step 1. Create a new project
Create a new Java project in your preferred development environment (for example, Eclipse).

Step 2. Add the com.mylib package
Add to the project a new package called **`com.mylib`** which consists of a class called **`Greetings`**. In the code listing below, notice that the `SayHello` function has been defined as `public static`.

```
package com.mylib;

public class Greetings {

   public static String SayHello ( boolean isMorning ) {
      if( isMorning )
         return "Good Morning!";
      return "Good Day!";
   }

}
```

Step 3. Import com.altova.types
Optionally, if your project needs support for special schema types such as date, time, and duration, import the `com.altova.types` package. To obtain this package, generate Java code from a mapping without custom functions: **`import com.altova.types.*;`**.

Step 4. Compile your custom library
Compile your custom library to a class file and add it to the Java classpath.

Step 5. Create .mff and reference your Java library
Using an XML editor, create a new `.mff` file and validate it against the following schema: **`C:\Program Files\MapForceLibraries\mff.xsd`**. Make sure that all references under **`implementation language="java"`** point to the correct Java members created previously. Also, the line **`function name="SayHello"`** must refer to the function name exactly as it was defined in Java. For more detail, see Configure .mff File 495.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping version="9" library="custom" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd">
   <implementations>
      <implementation language="java">
         <setting name="package" value="com.mylib"/>
         <setting name="class" value="Greetings"/>
      </implementation>
   </implementations>
   <group name="greetings">
      <component name="sayhello">
         <sources>
            <datapoint name="ismorning" type="xs:boolean"/>
         </sources>
         <targets>
            <datapoint name="result" type="xs:string"/>
         </targets>
         <implementations>
            <implementation language="java">
               <function name="SayHello"/>
            </implementation>
         </implementations>
         <description>
            <short>result = sayhello(ismorning)</short>
            <long>Returns "Good morning" or "Good day", depending on the input
parameter.</long>
         </description>
      </component>
   </group>
</mapping>
```

Step 6. Import .mff as a library
Now that your custom library is referenced in the `.mff` file, you can import the `.mff` file into MapForce as a library. For more information, see Import .mff Libraries[500].

5.9.6 Regular Expressions

When designing a MapForce mapping, you can use regular expressions ("regex") in the following contexts:

- In the **pattern** parameter of the `match-pattern`[653] and `tokenize-regexp`[603] functions
- To filter the nodes on which a node function should apply. For more information, see Applying Node Functions and Defaults Conditionally[452].

The regular expression syntax and semantics for XSLT and XQuery are as defined in Appendix F of "XML Schema Part 2: Datatypes Second Edition".

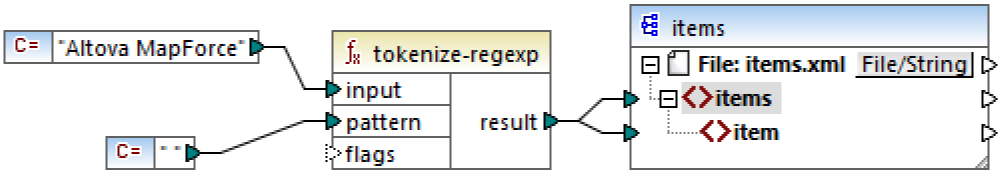
Note: When generating C++, C#, or Java code, the advanced features of the regular expression syntax might differ slightly. See the regex documentation of each language for more information.

Terminology

Let's examine the basic regular expression terminology by analyzing the **`tokenize-regexp`** function as an example. This function splits text into a sequence of strings, with the help of regular expressions. To achieve this, the function takes the following input parameters:

input	The input string to be processed by the function. The regular expression will operate on this string.
pattern	The actual regular expression pattern to be applied.
flags	This is an optional parameter that defines additional options (flags) that determine how the regular expression is interpreted, see "Flags" below.

In the mapping below, the input string is "Altova MapForce". The **pattern** parameter is a space character, and no regular expression flags are used.



This causes the text to be split whenever the space character occurs, so the mapping output is:

```
<items>
   <item>Altova</item>
   <item>MapForce</item>
</items>
```

Note that the **tokenize-regexp** function excludes the matched characters from the result. In other words, the space character in this example is omitted from the output.

The example above is very basic and the same result can be achieved without regular expressions, with the tokenize 598 function. In a more practical scenario, the **pattern** parameter would contain a more complex regular expression. The regular expression can consist of any of the following:

- Literals
- Character classes
- Character ranges
- Negated classes
- Meta characters
- Quantifiers

Literals

Use literals to match characters exactly as they are written (literally). For example, if input string is **abracadabra**, and **pattern** is the literal **br**, the output is:

```
<items>
   <item>a</item>
   <item>acada</item>
   <item>a</item>
</items>
```

The explanation is that the literal **br** had two matches in the input string **abracadabra**. After removing the matched characters from the output, the sequence of three strings illustrated above is produced.

Character classes

If you enclose a set of characters in square brackets (**[** and **]**), this creates a character class. One and only one of the characters inside the character class is matched, for example:

- The pattern **[aeiou]** matches any lowercase vowel.
- The pattern **[mj]ust** matches "must" and "just".

Note: The pattern is case sensitive, so a lowercase "a" does not match the uppercase "A". To make the matching case insensitive, use the **i** flag, see below.

Character ranges

Use `[a-z]` to create a range between the two characters. Only one of the characters will be matched at one time. For example, the pattern `[a-z]` matches any lowercase character between "a" and "z".

Negated classes

Using the caret (`^`) as the first character after the opening bracket negates the character class. For example, the pattern `[^a-z]` matches any character not in the character class, including newline characters.

Matching any character

Use the dot (`.`) meta character to match any single character, except for newline character. For example, the pattern `.` matches any single character.

Quantifiers

Within a regular expression, quantifiers define how many times the preceding character or sub-expression is allowed to occur in order for the match to take place.

?	Matches zero or one occurrences of the immediately preceding item. For example, the pattern `mo?` will match "m" and "mo".
+	Matches one or more occurrences of the immediately preceding item. For example, the pattern `mo+` will match "mo", "moo", "mooo", and so on.
*****	Matches zero or more occurrences of the immediately preceding item.
{min,max}	Matches any number of occurrences between *min* and *max*. For example, the pattern `mo{1,3}` matches "mo", "moo", and "mooo".

Parentheses

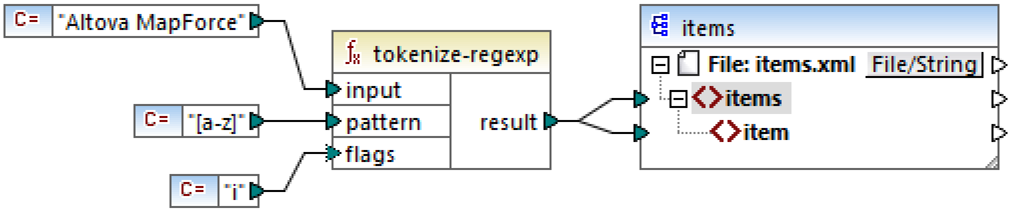
Parentheses `(` and `)` are used to group parts of a regex together. They can be used to apply quantifiers to a sub-expression (as opposed to just one character), or with alternation (see below).

Alternation

The vertical bar (pipe) character `|` means "or". It can be used to match any of the several sub-expressions separated by `|`. For example, the pattern `(horse|make) sense` will match both "horse sense" and "make sense".

Flags

These are optional parameters that define how the regular expression is to be interpreted. Each flag corresponds to a letter. Letters may be in any order and can be repeated.

s	If this flag is present, the matching process operates in the "dot-all" mode. If the input string contains "hello" and "world" on two *different* lines, the regular expression `hello*world` will only match if the **s** flag is set.
m	If this flag is present, the matching process operates in multi-line mode. In multi-line mode, the caret `^` matches the start of any line, i.e. the start of the entire string and the first character after a newline character. The dollar character `$` matches the end of any line, i.e. the end of the entire string and the character immediately before a newline character. Newline is the character **#x0A**.
i	If this flag is present, the matching process operates in case-insensitive mode. For example, the regular expression `[a-z]` plus the **i** flag matches all letters a-z and A-Z.  C= "Altova MapForce" C= "[a-z]" C= "i" tokenize-regexp input pattern flags result items File: items.xml File/String items item
x	If this flag is present, whitespace characters are removed from the regular expression prior to the matching process. Whitespace characters are **#x09**, **#x0A**, **#x0D** and **#x20**. **Note:** Whitespace characters within a character class are not removed, for example, `[#x20]`.

5.9.7 Function Library Reference

This reference section describes the MapForce built-in functions available in the Libraries window[23]. The functions are organized by library. The availability of function libraries in the **Libraries** window depends on the transformation language you choose for your mapping. To find out more about the list of available transformation languages, see this topic[19].

The information about the compatibility of functions and transformation languages is provided in the subsections below.

core functions

The lists below summarize the compatibility of core functions with transformation languages.
core | aggregate functions

- **`avg`**, **`max`**, **`max-string`**, **`min`**, **`min-string`**: XSLT 2.0, XSLT 3.0, XQuery 1.0, C#, C++, Java, Built-In;
- **`count`**, **`sum`**: all transformation languages.

core | conversion functions

- **boolean**, string, **number**: all transformation languages;
- **format-date**, **format-dateTime**, **format-time**: XSLT 2.0, XSLT 3.0, C#, C++, Java, Built-In;
- **format-number**: XSLT 1.0, XSLT 2.0, XSLT 3.0, C#, C++, Java, Built-In;
- **parse-date**, **parse-dateTime**, **parse-number**, **parse-time**: C#, C++, Java, Built-In.

core | file path functions
All the file path functions are compatible with all the transformation languages.

core | generator functions
The **auto-number** function is available for all the transformation languages.

core | logical functions
The logical functions are compatible with all the transformation languages.

core | math functions

- **add**, **ceiling**, **divide**, **floor**, **modulus**, **multiply**, **round**, **subtract**: all transformation languages;
- **round-precision**: C#, C++, Java, Built-In.

core | node functions

- **is-xsi-nil**, **local-name**, **static-node-annotation**, **static-node-name**: all transformation languages;
- **node-name**, **set-xsi-nil**, **substitute-missing-with-xsi-nil**: XSLT 2.0, XSLT 3.0, XQuery 1.0, C#, C++, Java, Built-In.

core | QName functions
The QName functions are compatible with all the transformation languages except for XSLT1.0.

core | sequence functions

- **exists**, **not-exists**, **position**, **substitute-missing**: all transformation languages;
- **distinct-values**, **first-items**, **generate-sequence**, **item-at**, **items-from-till**, **last-items**, **replicate-item**, **replicate-sequence**, **set-empty**, **skip-first-items**: XSLT 2.0, XSLT 3.0, XQuery 1.0, C#, C++, Java, Built-In;
- **group-adjacent**, **group-by**, **group-ending-with**, **group-into-blocks**, **group-starting-with**: XSLT 2.0, XSLT 3.0, C#, C++, Java, Built-In.

core | string functions

- **concat**, **contains**, **normalize-space**, **starts-with**, **string-length**, **substring**, **substring-after**, **substring-before**, **translate**: all transformation languages;
- **char-from-code**, **code-from-char**, **tokenize**, **tokenize-by-length**, **tokenize-regexp**: XSLT 2.0, XSLT 3.0, XQuery 1.0, C#, C++, Java, Built-In.

bson functions (MapForce Enterprise Edition only)

All the BSON functions are compatible only with Built-In.

db functions (MapForce Professional and Enterprise editions)

The `db` functions are compatible with C#, C++, Java, Built-In.

edifact functions (MapForce Enterprise Edition only)

The `edifact` functions are compatible with C#, C++, Java, Built-In.

lang functions (MapForce Professional and Enterprise editions)

The lists below summarize the compatibility of `lang` functions with transformation languages.

lang | datetime functions

The `lang | datetime` functions are compatible with C#, C++, Java, Built-In.

lang | file functions

The functions **`read-binary-file`** and **`write-binary-file`** are compatible only with Built-In.

lang | generator functions

The **`create-guid`** function is available for C#, C++, Java, Built-In.

lang | logical functions

The `lang | logical` functions are available for C#, C++, Java, Built-In.

lang | math functions

The `lang | math` functions are available for C#, C++, Java, Built-In.

lang | QName functions

The `lang | QName` functions are compatible with C#, C++, Java, Built-In.

lang | string functions

- **`charset-decode`**, **`charset-encode`**: Built-In;
- **`match-pattern`**: C#, Java, Built-In.
- **`capitalize`**, **`count-substring`**, **`empty`**, **`find-substring`**, **`format-guid-string`**, **`left`**, **`left-trim`**, **`lowercase`**, **`pad-string-left`**, **`pad-string-right`**, **`repeat-string`**, **`replace`**, **`reversefind-substring`**, **`right`**, **`right-trim`**, **`string-compare`**, **`string-compare-ignore-case`**, **`uppercase`**: C#, C++, Java, Built-In.

mime functions (MapForce Enterprise Edition only)

The `mime` functions are available for Built-In only.

xbrl functions (MapForce Enterprise Edition only)

The `xbrl` functions are compatible with C#, C++, Java, Built-In.

xlsx functions (MapForce Enterprise Edition only)

The `xlsx` functions are compatible with XSLT 2.0, XSLT 3.0, C#, Java, and Built-In.

xpath2 functions

All the `xpath2` functions are compatible with XSLT 2.0, XSLT 3.0, and XQuery 1.0.

xpath3 functions

All the `xpath3` functions are compatible only with XSLT 3.0.

xslt10 functions

The lists below summarize the compatibility of `xslt10` functions with transformation languages.
xslt10 | xpath functions

- **`local-name`**, **`name`**, **`namespace-uri`**: XSLT 1.0, XSLT 2.0, and XSLT 3.0.
- **`lang`**, **`last`**, **`position`**: XSLT 1.0.

xslt10 | xslt functions

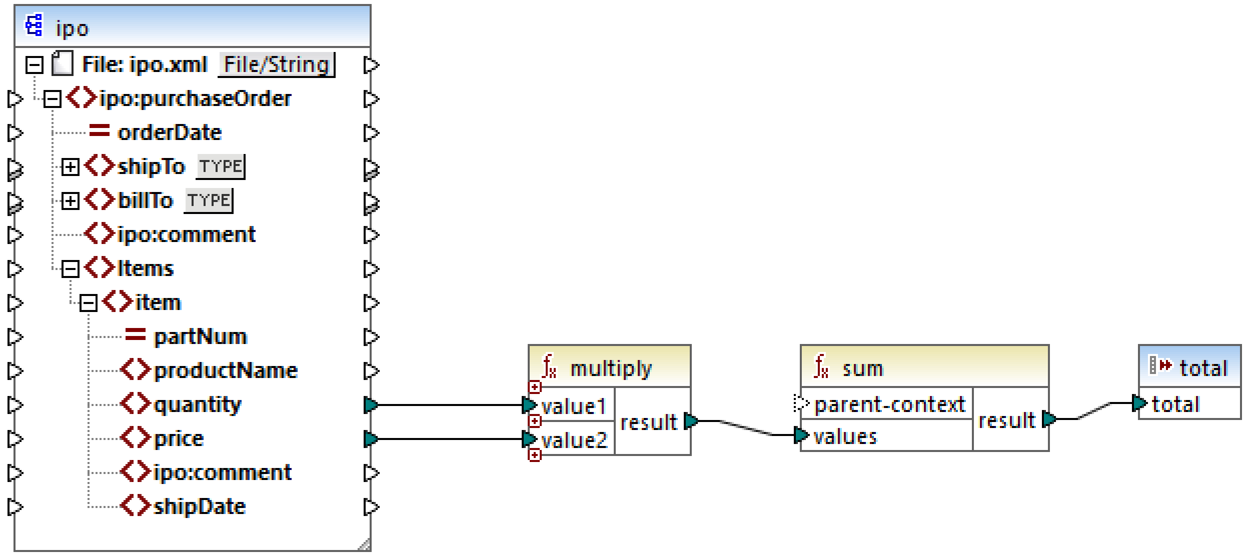
- **`generate-id`**, **`system-property`**: XSLT 1.0, XSLT 2.0, and XSLT 3.0.
- **`current`**, **`document`**, **`element-available`**, **`function-available`**, **`unparsed-entity-uri`**: XSLT 1.0.

5.9.7.1 core | aggregate functions

"Aggregating" means processing multiple values of the same type so as to obtain a single result, such as a sum, a count, or an average. You can perform data aggregation in MapForce with the help of aggregation functions, such as `avg`, `count`, `max`, and others.

The following two arguments are common to all aggregation functions:

1. **parent-context**. This argument is optional; it lets you override the default mapping context (and thus change the scope of the function, or the values that the function must iterate over). For a worked example, see Example: Changing the Parent Context 82.
2. **values**. This argument must be connected to a source item that supplies the values to be processed. For example, in the mapping illustrated below, the **`sum`** function takes as input a sequence of numeric values that originates from a source XML file. For each item in the source XML file, the **`multiply`** function gets the item's price times quantity, and passes the result to the **`sum`** function. The **`sum`** function will aggregate all input values and produce a total result that is also the output of the mapping. You can find this mapping in the **`MapForceExamples`** folder.

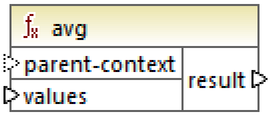


SimpleTotal.mfd

Some aggregate functions, such as `min`, `max`, `sum`, and `avg`, work exclusively with numeric values. The input data of these functions is converted to the **decimal** data type for processing.

5.9.7.1.1 avg

Returns the average value of all values within the input sequence. The average of an empty set is an empty set.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context 82. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.

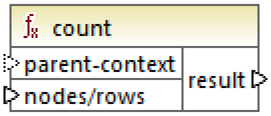
Argument	Description
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

Example

See Example: Grouping Records by Key 569.

5.9.7.1.2 count

Returns the number of individual items making up the input sequence. The count of an empty set is zero.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Note that this function has limited functionality in XSLT 1.0.

Parameters

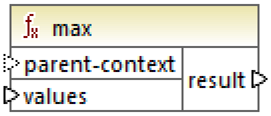
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context 82. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
nodes/rows	This argument must be connected to the source item to be counted.

Example

See Example: Changing the Parent Context 82, Example: Counting Database Table Rows 368.

5.9.7.1.3 max

Returns the maximum value of all numeric values in the input sequence. The maximum of an empty set is an empty set.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

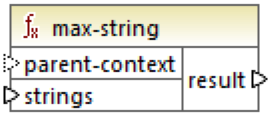
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context 82. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the maximum from a sequence of strings, use the max-string 517 function.

Example

See Example: Grouping Records by Key 569.

5.9.7.1.4 max-string

Returns the maximum value of all string values in the input sequence. For example, **`max-string("a", "b", "c")`** returns **`"c"`**. The function returns an empty set if the **strings** argument is an empty set.



Languages

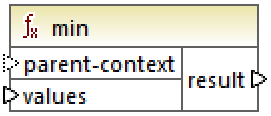
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context 82. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

5.9.7.1.5 min

Returns the minimum value of all numeric values in the input sequence. The minimum of an empty set is an empty set.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

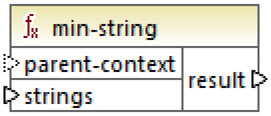
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context 82. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the minimum from a sequence of strings, use the min-string 519 function.

Example

See Example: Grouping Records by Key 569.

5.9.7.1.6 min-string

Returns the minimum value of all string values in the input sequence. For example, `min-string("a", "b", "c")` returns `"a"`. The function returns an empty set if the **strings** argument is an empty set.



Languages

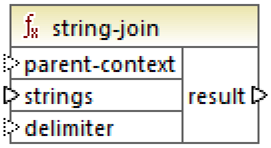
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context 82. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

5.9.7.1.7 string-join

Concatenates all the values of the input sequence into one string delimited by whatever string you choose to use as the delimiter. The function returns an empty string if the **strings** argument is an empty set.



Languages

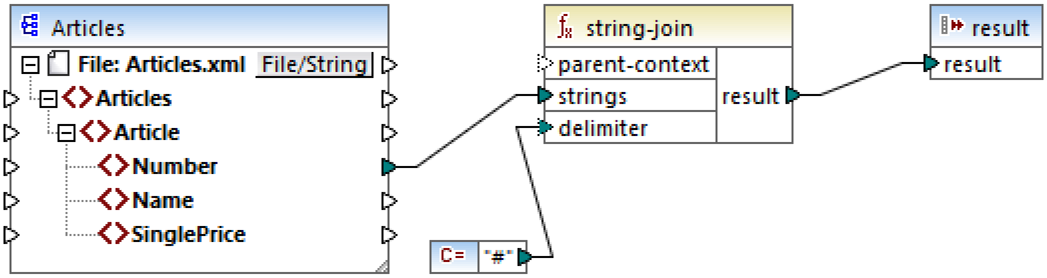
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context 82. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.
delimiter	Optional argument. Specifies the delimiter to be inserted between any two consecutive strings.

Example

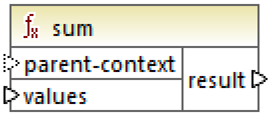
In the example below, the source XML file contains four **Article** items, with the following numbers: 1, 2, 3, and 4.



The constant supplies the character "#" as the delimiter. The mapping result is, therefore, **`1#2#3#4`**. If you do not supply a delimiter, then the result becomes **`1234`**.

5.9.7.1.8 sum

Returns the arithmetic sum of all values in the input sequence. The sum of an empty set is zero.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Example: Changing the Parent Context[82]. The `parent-context` argument is an optional argument in some MapForce core aggregation functions (e.g., **`min`**, **`max`**, **`avg`**, **`count`**). In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

Example

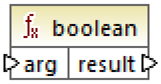
See Example: Summing Node Values[479].

5.9.7.2 core | conversion functions

To support explicit data type conversion, several type conversion functions are available in the **conversion** library. Note that the conversion functions are not always necessary because, in most cases, MapForce creates the necessary conversions automatically. Conversion functions are typically useful to format date and time values, or to compare values. For example, if some mapping items are of differing types (such as integer and string), you can use the number[530] conversion function to force a numeric comparison.

5.9.7.2.1 boolean

Converts the value of **arg** to a Boolean value. This may be useful for working with logical functions (such as **`equal`**, **`greater`**, and so on), as well as filters and if-else conditions[408]. To get a Boolean **false**, supply an empty string or numeric 0 as argument. To get a Boolean **true**, supply a non-empty string or numeric 1 as argument.



Languages

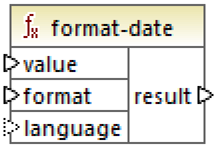
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameter

Argument	Description
arg	Mandatory argument. Supplies the value to be converted.

5.9.7.2.2 format-date

Converts a date value of type `xs:date` to a string and formats it according to specified options.



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

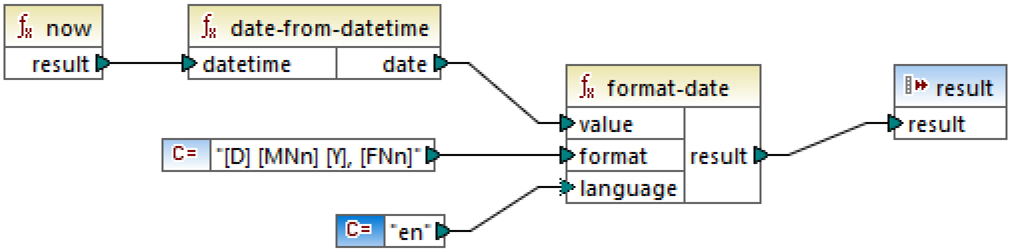
Parameters

Argument	Description
value	The `xs:date` value to be formatted.
format	A format string identifying the way in which the date is to be formatted. This argument is used in the same way as the **format** argument in the format-dateTime 523 function.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values:

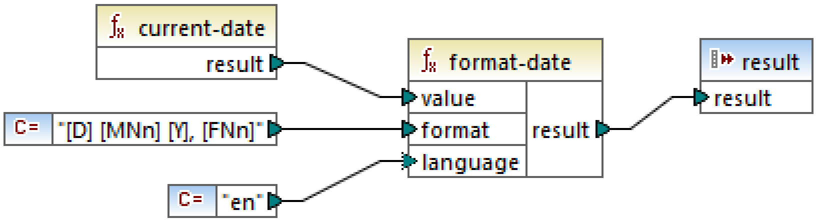
Argument	Description
	de German **en (default)** English **es** Spanish **fr** French **ja** Japanese

Example

The following mapping outputs the current date in a format like: "25 March 2020, Wednesday". To translate this value to Spanish, set the value of the **language** argument to **es**.

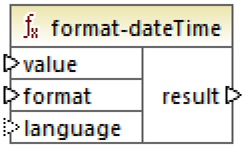


Note that the mapping above is designed for the Built-in, C++, C#, or Java transformation languages. In XSLT 2.0, the same result can be achieved by the following mapping:



5.9.7.2.3 format-dateTime

Converts a value of type `xs:dateTime` to a string. The string representation of date and time is formatted according to the value of the `format` argument.



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
`value`	The `xs:dateTime` value to be formatted.
`format`	A format string identifying the way in which `value` is to be formatted. See *Formatting* below.
`language`	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values are as follows: `de` German `en` (default) English `es` Spanish `fr` French `ja` Japanese

Note: If the function's output (result) is connected to an item of type other than `string`, the formatting may be lost as the value is cast to the target type. To disable this automatic cast, clear the *Cast target values to target types* check box in the Component Settings 41 of the target component.

Formatting

The `format` argument consists of a string that contains *variable markers* enclosed in square brackets (e.g., **`[Y]/[M]/[D]`**). Characters outside the square brackets are literal characters. If square brackets are needed as literal characters in the result, then they should be doubled.

Each variable marker consists of the following parts:

- a specifier identifying which part of the data type is to be displayed (i.e., date or time),
- an optional formatting modifier,
- an optional presentation modifier, and
- an optional width modifier, which must be preceded by a comma if it is present.

```
[variable marker] = [specifier(format)?(presentation)?(width)?]

where

width := , min-width ("-" max-width)?
```

Specifiers
The available specifiers are described in the table below.

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

Formatting modifiers
The formatting modifier can be any of the following:

Formatting modifier	Description	Example
1	Decimal numeric format with no leading zeros	1, 2, 3
01	Decimal format, two digits	01, 02, 03
N	Name of component, upper case	MONDAY, TUESDAY
n	Name of component, lower case	monday, tuesday
Nn	Name of component, upper/lower case	Monday, Tuesday

Important

The **N**, **n**, and **Nn** modifiers are supported by the following specifiers: **M**, **F**, and **P**. The **N**, **n**, and **Nn** modifiers are also supported by the **z** specifier but only in Built-In and code generation (*Professional and Enterprise editions*).

If you need a width modifier, put a comma before it. The width modifier is a digit that expresses the minimum width. Optionally, you can add a dash and a digit that expresses the maximum width. For example:

- **`[D,2]`** is the day of the month, with leading zeros (two digits).
- **`[MNn,3-3]`** is the name of the month, written as three characters, e.g., *Jan*, *Feb*, *Mar*, and so on.

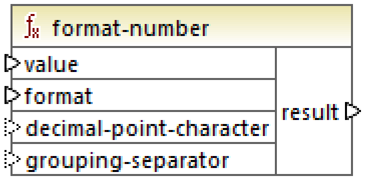
Examples

The table below illustrates some examples of formatting `xs:dateTime` values with the help of the **`format-dateTime`** function. The *Value* column specifies the value supplied to the `value` argument. The *Format* column specifies the value of the `format` argument. The *Result* column shows 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
2010-06-02T08:02:12.054	`[Y] [MNn] [D01] [FNn,3-3] [d] [H]:[m]:[s].[f]`	2010 June 02 Wed 153 8:02:12.054
2010-06-02T08:02:12.054+02:00	`[Y] [MNn] [D01] [FNn,3-3] [d] [H]:[m]:[s].[f] [z]`	2010 June 02 Wed 153 8:02:12.054 GMT+02:00
2010-06-02T08:02:12.054+02:00	`[Y] [MNn] [D1] [FNn] [H]:[m]:[s].[f] [Z]`	2010 June 2 Wednesday 8:02:12.054 +02:00
2010-06-02T08:02:12.054	`[Y] [MNn] [D] [FNn,3-3] [H01]:[m]:[s]`	2010 June 2 Wed 08:02:12

5.9.7.2.4 format-number

Converts a number into a string and formats it according to the specified options.



Languages

Built-in, C++, C#, Java, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value	Mandatory argument. Supplies the number to be formatted.
format	Mandatory argument. Supplies a format string that identifies the way in which the number is to be formatted. See "Remarks" below.
decimal-point-format	Optional argument. Supplies the character to be used as the decimal point character. The default value is the full stop (.) character.
grouping-separator	Optional argument. Supplies the character used to separate groups of numbers. The default value is the comma (,) character.

Note: If the function's output (result) is connected to an item of type other than string, the formatting may be lost as the value is cast to the target type. To disable this automatic cast, clear the **Cast target values to target types** check box in the Component Settings 41 of the target component.

Remarks

The **format** argument takes the following form:

```
format := subformat (;subformat)?
  subformat := (prefix)? integer (.fraction)? (suffix)?
  prefix := any characters except special characters
  suffix := any characters except special characters
  integer := (#)* (0)* ( allowing ',' to appear)
  fraction := (0)* (#)* (allowing ',' to appear)
```

The first *subformat* is used for formatting positive numbers, and the second subformat for negative numbers. If only one *subformat* is specified, then the same subformat will be used for negative numbers, but with a minus sign added before the *prefix*.

Special Character	Default	Description
zero-digit	0	A digit will always appear at this point in the result
digit	#	A digit will appear at this point in the result string unless it is a redundant leading or trailing zero
decimal-point	.	Separates the integer and the fraction part of the number.
grouping-separator	,	Separates groups of digits.

Special Character	Default	Description
percent-sign	%	Multiplies the number by 100 and shows it as a percentage.
per-mille	‰	Multiplies the number by 1000 and shows it as per-mille.

The table below illustrates examples of format strings and their result.

Note: The rounding method used by the **format-number** function is "half up", which means that the value gets rounded up if the fraction is greater than or equal to 0.5. The value gets rounded down if the fraction is less than 0.5. This method of rounding applies only to generated program code and the built-in execution engine. In XSLT 1.0, the rounding mode is undefined. In XSLT 2.0, the rounding mode is "round-half-to-even".

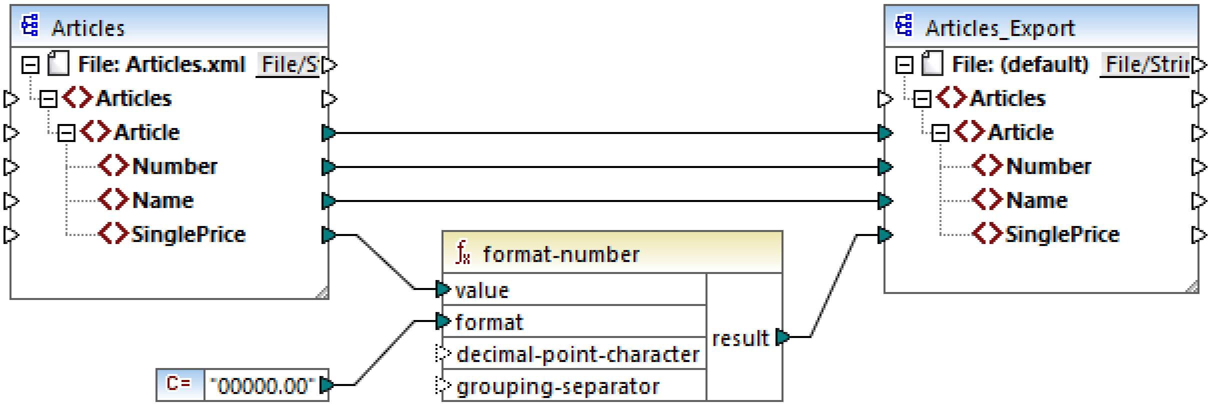
Number	Format String	Result
1234.5	#,##0.00	1,234.50
123.456	#,##0.00	123.46
1000000	#,##0.00	1,000,000.00
-59	#,##0.00	-59.00
1234	###0.0###	1234.0
1234.5	###0.0###	1234.5
.00025	###0.0###	0.0003
.00035	###0.0###	0.0004
0.25	#00%	25%
0.736	#00%	74%
1	#00%	100%
-42	#00%	-4200%
-3.12	#.00;(#.00)	(3.12)
-3.12	#.00;#.00CR	3.12CR

Example

The mapping illustrated below reads data from source XML and writes it to a target XML. There are multiple **SinglePrice** elements in the source that contain the following decimal values: **25**, **2.30**, **34**, **57.50**. The mapping has two goals:

1. Pad all values with zeros to the left so that the significant part takes 5 digits exactly
2. Pad all values with zeros to the right so that the decimal part takes 2 digits exactly

To achieve this, the format string `00000.00` was supplied as argument to the `format-number` function.



PreserveFormatting.mfd

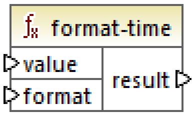
Consequently, the values in the target have become:

```
00025.00
00002.30
00034.00
00057.50
```

You can find the mapping design file at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\PreserveFormatting.mfd**.

5.9.7.2.5 format-time

Converts an `xs:time` input value into a string.



Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

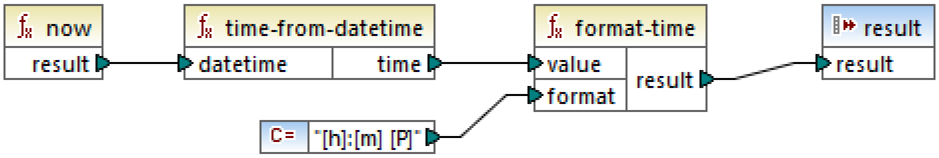
Parameters

Argument	Description
value	Mandatory argument. Supplies the `xs:time` value to be formatted.
format	Mandatory argument. Supplies a format string. This argument is used in the same way as the **format** argument in the format-dateTime 523 function.

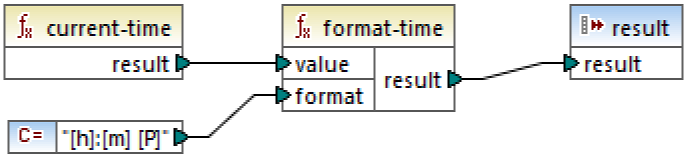
Example

The following mapping outputs the current time in a format like `2:15 p.m.` . To achieve this, it uses the format string `[h]:[m] [P]`, where:

- **[h]** is the current hour in 12-hour format
- **[m]** is the current minute
- **[P]** is the "a.m." or "p.m." part

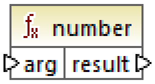


Note that the mapping above is designed for the Built-in, C++, C#, or Java transformation languages. In XSLT 2.0, the same result can be achieved by the following mapping:



5.9.7.2.6 number

Converts the value of **arg** into a number, where **arg** is a string or Boolean value. If **arg** is a string, MapForce will attempt to parse it as a number. For example, a string like `"12.56"` is converted to the decimal value `12.56`. If **arg** is Boolean **true**, it is converted to numeric **1**. If **arg** is Boolean **false**, it is converted to numeric **0**.



Languages

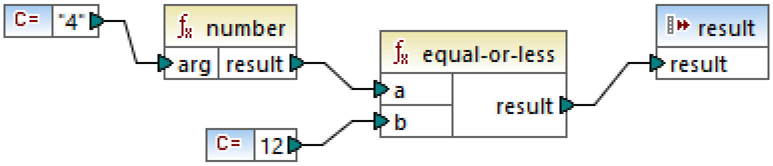
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
arg	Mandatory argument. Supplies the value to be converted.

Example

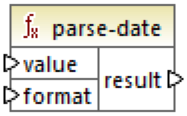
In the example below, the first constant is of type `string` and it contains the string "4". The second constant contains the numeric constant 12. In order for the two values to be compared as numbers, the types must agree.



Adding a **number** function to the first constant converts the string "4" to the numeric value of 4. The result of the comparison is then "true". If the **number** function were not used (that is, if "4" was connected directly to **a**), a string comparison would occur, with the result being "false".

5.9.7.2.7 parse-date

Converts a string into a date. This function uses the parse-dateTime 532 function as a basis, while ignoring the time component. The result is of type `xs:date`.



Languages

Built-in, C++, C#, Java.

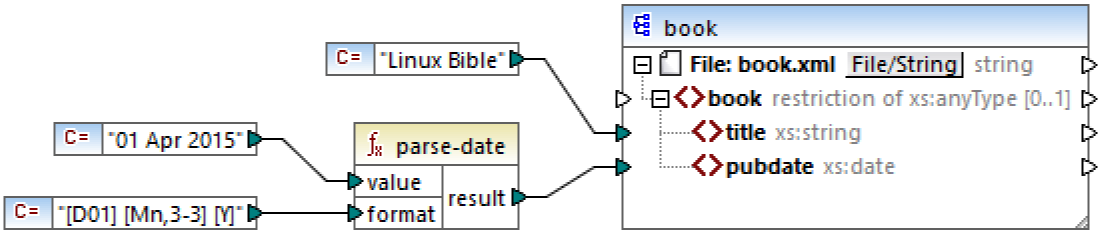
Parameters

Argument	Description
value	Mandatory argument. Supplies the string value to be converted.
format	Mandatory argument. Supplies a format string. This argument is used in the same way as the **format** argument in the parse-dateTime 532 function.

Example

The mapping below parses the string **`"01 Apr 2015"`**, converts it to a date and writes the result to a target item (**pubdate**) of type `xs:date`. This was achieved by using the format **`[D01] [MNn,3-3] [Y]`**, where:

- **`[D01]`** is the date of the month, expressed as two digits
- **`[MNn,3-3]`** is the month name, with a minim and maximum width of 3 characters
- **`[Y]`** is the year

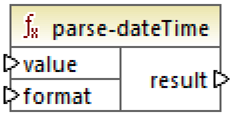


The result is as follows (excluding the XML and namespace declarations):

```
<book>
   <title>Linux Bible</title>
   <pubdate>2015-04-01</pubdate>
</book>
```

5.9.7.2.8 parse-dateTime

Converts a date and time value expressed as a string into a value of type `xs:dateTime`.



Languages

Built-in, C++, C#, Java.

Parameters

Argument	Description
value	The string value to be converted.
format	Specifies the format mask to apply to **value**.

Remarks

A format mask can consist of the following components:

Component	Description	Default Presentation
Y	year (absolute value)	four digits (2010)
M	month of the year	1-12

Component	Description	Default Presentation
D	day of month	1-31
d	day of year	1-366
H	hour (24 hours)	0-23
h	hour (12 hour)	1-12
P	A.M. or P.M.	alphabetic (language dependent)
m	minutes in hour	00-59
s	seconds in minute	00-59
f	fractional seconds	numeric, one decimal place
Z	timezone as a time offset from UTC	+08:00
z	timezone as a time offset using GMT	GMT+n

Some of the components above take modifiers (for example, they can be used to interpret a date either as a single digit or as two digits):

Modifier	Description	Example
1	decimal numeric format with no leading zeros: 1, 2, 3, ...	1, 2, 3
01	decimal format, two digits: 01, 02, 03, ...	01, 02, 03
N	name of component, upper case	FEBRUARY, MARCH
n	name of component, lower case	february, march
Nn	name of component, title case	February, March

Note: *N*, *n*, and *Nn* modifiers support only the component *M* (month).

If you need a width modifier, put a comma before it. The width modifier is a digit that expresses the minimum width. Optionally, you can add a dash and a digit that expresses the maximum width. For example:

- **`[D,2]`** is the day of the month, with leading zeros (two digits).
- **`[MNn,3-3]`** is the name of the month, written as three characters, e.g., *Jan*, *Feb*, *Mar*, and so on.

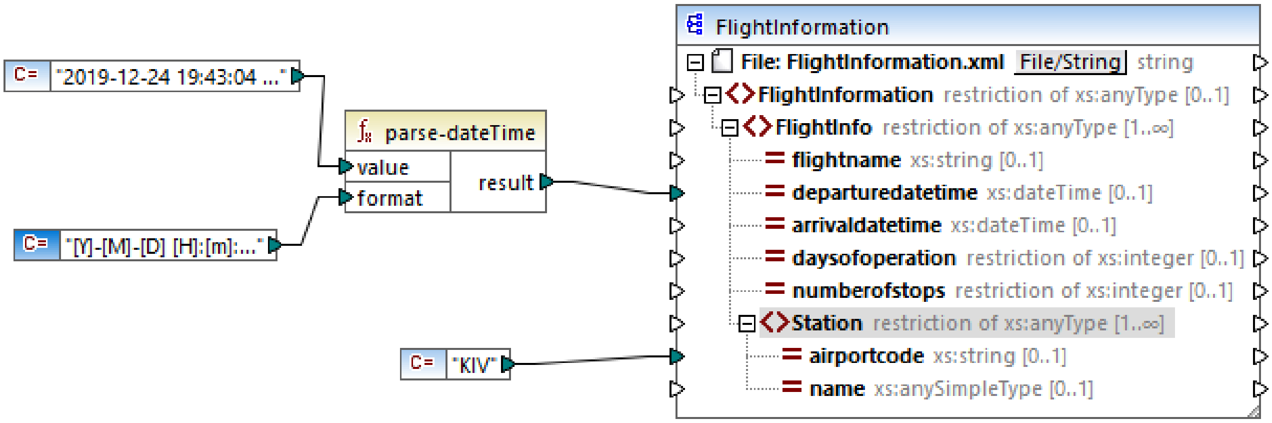
The table below lists some format examples:

Value	Format	Result
21-03-2002 16:21:12.492 GMT+02:00	`[D]-[M]-[Y] [H]:[m]:[s].[f] [z]`	2002-03-21T16:21:12.492+02:00
315 2004 +01:00	`[d] [Y] [Z]`	2004-11-10T00:00:00+01:00

Value	Format	Result
1.December.10 03:2:39 p.m. +01:00	`[D].[MNn].[Y,2-2] [h]:[m]:[s] [P] [Z]`	2010-12-01T15:02:39+01:00
20110620	`[Y,4-4][M,2-2][D,2-2]`	2011-06-20T00:00:00

Example

In the mapping below, the string value **`2019-12-24 19:43:04 +02:00`** is converted into its `dateTime` equivalent, by applying the format mask **`[Y]-[M]-[D] [H]:[m]:[s] [Z]`**.



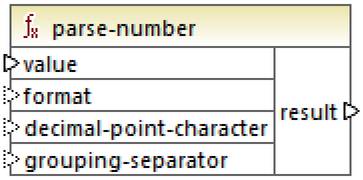
C= "2019-12-24 19:43:04 ..."
parse-dateTime
value
format
result
C= "[Y]-[M]-[D] [H]:[m]:..."
C= "KIV"
FlightInformation
File: FlightInformation.xml File/String string
FlightInformation restriction of xs:anyType [0..1]
FlightInfo restriction of xs:anyType [1..∞]
flightname xs:string [0..1]
departuredatetime xs:dateTime [0..1]
arrivaldatetime xs:dateTime [0..1]
daysofoperation restriction of xs:integer [0..1]
numberofstops restriction of xs:integer [0..1]
Station restriction of xs:anyType [1..∞]
airportcode xs:string [0..1]
name xs:anySimpleType [0..1]

The result is as follows (excluding the XML and namespace declarations):

```
<FlightInformation>
   <FlightInfo departuredatetime="2019-12-24T19:43:04+02:00">
      <Station airportcode="KIV"/>
   </FlightInfo>
</FlightInformation>
```

5.9.7.2.9 parse-number

The **`parse-number`** function (*see below*) converts a string into a decimal number according to a specified pattern. The function uses the pattern to determine only the prefix, suffix and digit grouping. The actual length of the number is not checked against the pattern. If the input value is longer or shorter than specified by the pattern, this will be ignored in the checking.



Languages

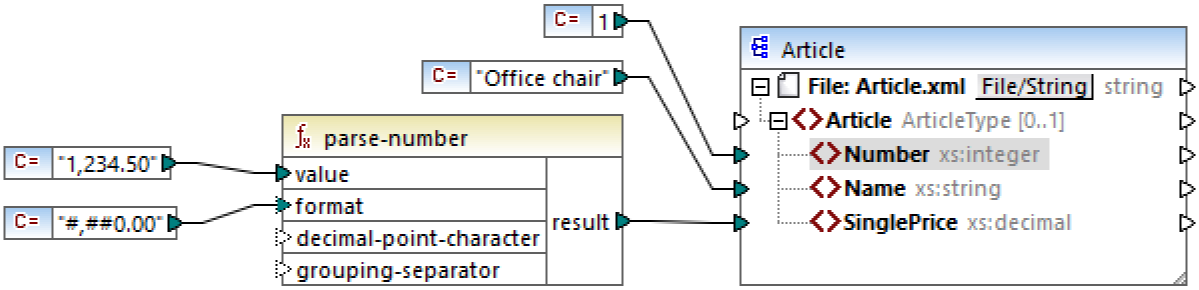
Built-in, C++, C#, Java.

Parameters

Argument	Description
`value`	The string to be converted to a number.
`format`	Optional argument. A format string that identifies the way in which the number is currently formatted. The format string is the same as that used in format-number 526. Default is "#,##0.#"
`decimal-point-character`	Optional argument. Specifies the character to be used as the decimal point character. Default is the '.' character.
`grouping-separator`	Optional argument. Specifies the separator/delimiter used to separate groups of numbers. Default is the "," character.

Example

The following mapping parses the string value **`"1,234.50"`** to a decimal equivalent, by using the format mask **`#,##0.00`**. In this mapping , there is no need to connect the `decimal-point-character` and `grouping-separator` arguments, since their default values match the format of the input string.

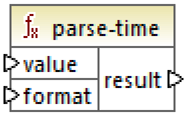


The output (excluding the XML and namespace declarations) is shown below:

```
<Article>
   <Number>1</Number>
   <Name>Office chair</Name>
   <SinglePrice>1234.5</SinglePrice>
</Article>
```

5.9.7.2.10 parse-time

Converts a string into an `xs:time` value. This function uses the parse-dateTime[532] function as a basis, while ignoring the date component.



Languages

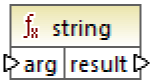
Built-in, C++, C#, Java.

Parameters

Argument	Description
value	Mandatory argument. Supplies the string value to be converted.
format	Mandatory argument. Supplies a format string. This argument is used in the same way as the **format** argument in the parse-dateTime[532] function.

5.9.7.2.11 string

Converts an input value into a string. The function can also be used to retrieve the text content of a node. If the input node is an XML complex type, then all descendants are also output as a single string.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

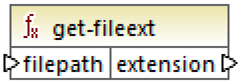
Argument	Description
arg	Mandatory argument. Supplies the value to be converted.

5.9.7.3 core | file path functions

The **file path** functions allow you to directly access and manipulate file path data, such as folders, file names, and extensions for further processing in your mappings. They can be used in all languages supported by MapForce.

5.9.7.3.1 get-fileext

Returns the extension of the file path including the dot "." character.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

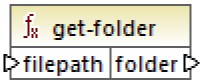
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.

Example

If you supply "c:\data\Sample.mfd" as argument, the result is `.mfd`.

5.9.7.3.2 get-folder

Returns the folder name of the file path including the trailing slash, or backslash character.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

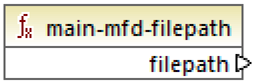
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.

Example

If you supply "c:\data\Sample.mfd" as argument, the result is **`c:\data\`**.

5.9.7.3.3 main-mfd-filepath

Returns the full path of the mapping design file (.mfd) containing the main mapping. An empty string is returned if the .mfd is currently not saved.

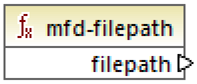


Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

5.9.7.3.4 mfd-filepath

If the function is called in the main mapping, it returns the same as the main-mfd-filepath[538] function, i.e. the full path of the .mfd file containing the main mapping. An empty string is returned if the .mfd file is currently not saved. If called within a user-defined function which is *imported* by an .mfd file, it returns the full path of the *imported* .mfd file that contains the definition of the user-defined function.

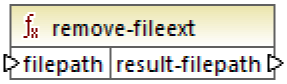


Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

5.9.7.3.5 remove-fileext

Removes the extension of the file path, including the dot character.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

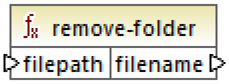
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.

Example

If you supply "c:\data\Sample.mfd" as argument, the result is `c:\data\Sample`.

5.9.7.3.6 remove-folder

Removes the directory of the file path, including the trailing slash, or backslash character.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

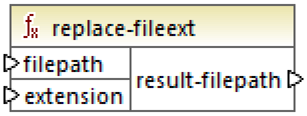
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.

Example

If you supply "c:\data\Sample.mfd" as argument, the result is `Sample.mfd`.

5.9.7.3.7 replace-fileext

Replaces the extension of the file path supplied by the **filepath** parameter with the one supplied by the connection to the **extension** parameter.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

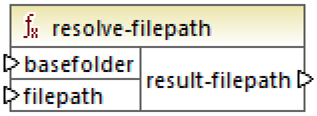
Argument	Description
filepath	Mandatory argument. Supplies the file path to be processed.
extension	Mandatory argument. Supplies the new extension to use.

Example

If you supply "c:\data\Sample.log" as **filepath**, and ".txt" as **extension**, the result is `c:\data\Sample.txt`.

5.9.7.3.8 resolve-filepath

Resolves a relative file path against a base folder. The function supports '.' (current directory) and '..' (parent directory).



Languages

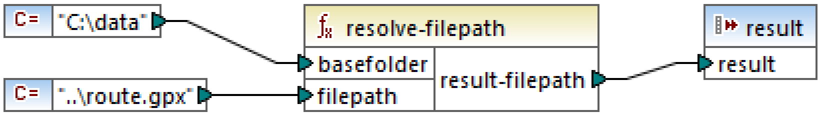
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
basefolder	Mandatory argument. Supplies the base directory relative to which the path should be resolved. This can be an absolute or relative path.
filepath	Mandatory argument. Supplies the relative file path to be resolved.

Examples

In the mapping below, the relative file path **..\route.gpx** is resolved against the **C:\data** directory.



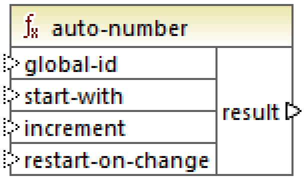
The mapping result is `C:\route.gpx`.

5.9.7.4 core | generator functions

The **core / generator** functions library includes functions which generate values.

5.9.7.4.1 auto-number

Generates integer numbers in a sequence (for example, 1,2,3,4, ...). It is possible to set the starting integer, the increment value, and other options by means of parameters.



> The exact order in which functions are called by the generated mapping code is undefined. MapForce may need to cache calculated results for reuse, or evaluate expressions in any order. Also, unlike other functions, the `auto-number` function returns a different result when called multiple times with the same input parameters. Therefore, it is strongly recommended to use the `auto-number` function cautiously. In some cases, it is possible to achieve the same result by using the position 582 function instead.

Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

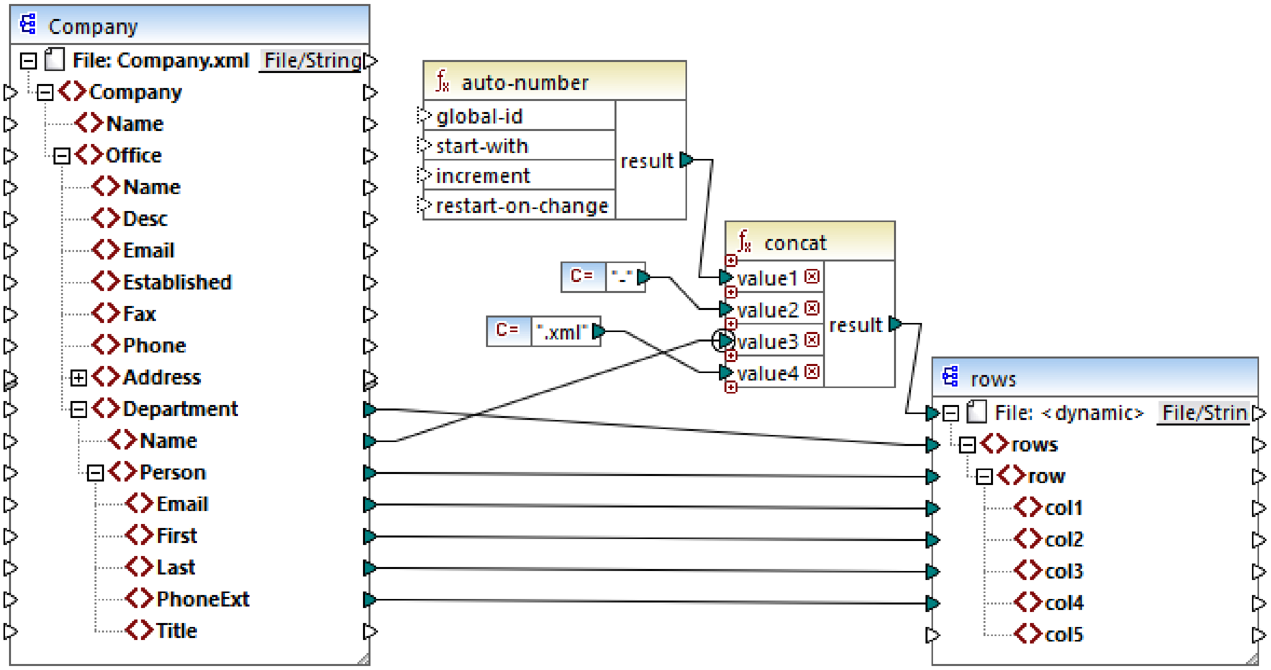
Parameters

Argument	Description
global-id	Optional parameter. If a mapping design contains multiple `auto-number` functions, they will generate sequences with duplicate (overlapping) numbers. To make all `auto-number` functions aware of each other, and thus generate sequences that do not overlap, connect a common string (for example, a constant) to the **global-id** input of each `auto-number` function.
start-with	Optional parameter. Specifies the integer with which the generated sequence begins. The default value is **1**.
increment	Optional parameter. Specifies the increment value. The default value is **1**.
restart-on-change	Optional parameter. Resets the counter to **start-with**, when the content of the connected item changes.

Example

The following mapping is a variation of the **ParentContext.mfd** mapping discussed in the Example: Changing the Parent Context 82.

The goal of the mapping illustrated below is to generate multiple XML files, one for each department in the source XML file. There are some departments with the same name (that's because they belong to different parent offices). For this reason, each generated file name must begin with a sequential number, for example **1-Administration.xml**, **2-Marketing.xml**, and so on.



To achieve the mapping goal, the **auto-number** function was used. The result of this function is concatenated with a dash character, followed by the department name, followed by the ".xml" string in order to create the unique name of the generated file. Importantly, the third parameter of the **concat** function (the department name) has a priority context 86 applied. This has the effect that the **auto-number** function is called in the context of each department, and produces the required sequential values. If priority context were not used, the **auto-number** function would keep generating number 1 (in the absence of any context), and duplicate file names would be generated as a consequence.

5.9.7.5 core | logical functions

Logical functions are (generally) used to compare input data and return a Boolean `true` or `false`. They are generally used to test data before passing on a subset to the target component using a filter 408. Nearly all logical functions have the following structure:

- input parameters: `a | b` or `value1 | value2`
- output parameter: `result`

The evaluation result depends on the input values as well as the data types used for the comparison. For example, the less than comparison of the integer values **4** and **12** yields the boolean value `true`, since 4 is less than 12. If the two input parameters contain string values **4** and **12**, the lexical analysis results in the output value `false`, since **4** is alphabetically greater than the first character **1** of the second operand (**12**).

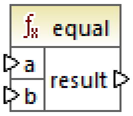
If all input values are of the same data type, then the comparison is done for the common type. If input values are of different types (for example, `integer` and `string`, or `string` and `date`), then the data type used for the comparison is the most general (least restrictive) of the two.

Before comparing two values of different types, all input values are converted to a common data type. Using the previous example, the data type `string` is less restrictive than `integer`. Comparing the integer value **4** with the string **12** converts the integer value **4** to the string **4**, which is then compared with the string **12**.

Note: Logical functions cannot be used to test the existence of null values. If you supply a null value as an argument to a logical function, it returns a null value. For more information about handling null values, see Nil Values / Nillable 139.

5.9.7.5.1 equal

The **equal** function (*see screenshot below*) returns Boolean `true` if **a** is the same as **b**; `false` otherwise. The comparison is case-sensitive.



Example:

```
a = hi
b = hi
```

In this example, both values are the same. Therefore, the result is `true`. If, for instance, `b` equaled `Hi`, the function would return `false`.

Languages

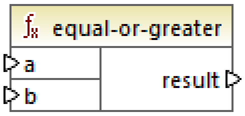
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

5.9.7.5.2 equal-or-greater

Returns Boolean **true** if *a* is equal to or greater than *b*; **false** otherwise.



Languages

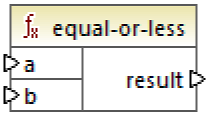
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

5.9.7.5.3 equal-or-less

Returns Boolean **true** if *a* is equal to or less than *b*; **false** otherwise.



Languages

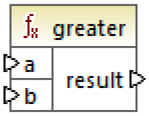
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

5.9.7.5.4 greater

Returns Boolean **true** if *a* is greater than *b*; **false** otherwise.



Languages

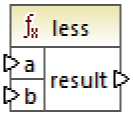
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

5.9.7.5.5 less

Returns Boolean **true** if *a* is less than *b*; **false** otherwise.



Languages

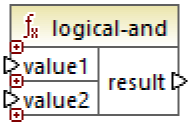
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

5.9.7.5.6 logical-and

Returns Boolean **true** only if each input value is true; **false** otherwise. You can connect the result to another **logical-and** function and thus join an arbitrary number of conditions with logical AND, in order to test that they all return **true**. Also, this function can be extended to take additional arguments, see Add or Delete Function Arguments 438.



Languages

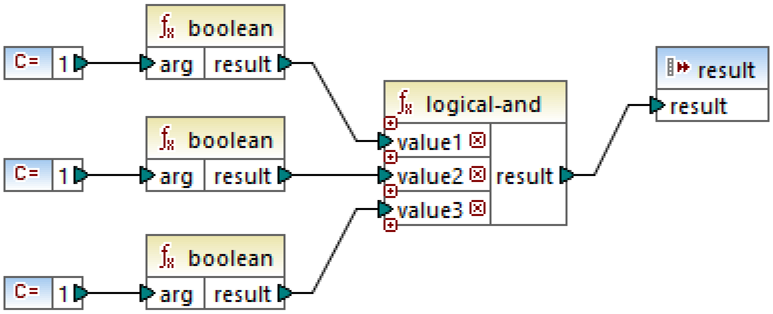
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first value to compare.
value2	Mandatory parameter. Provides the second value to compare.

Example

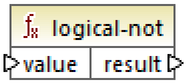
The mapping illustrated below returns **true** because all input values to the **logical-and** function are **true** as well. If any of the input values were **false**, then the mapping's result would be **false** as well.



See also Example: Look-up and Concatenation 472.

5.9.7.5.7 logical-not

Inverts or flips the logical result of the input value. For example, if *value* is **true**, the function's result is false. If *value* is **false**, then result is true.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

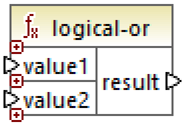
Parameters

Argument	Description
value	Mandatory parameter. Provides the input value.

5.9.7.5.8 logical-or

This function requires both input values to be Boolean. If at least one of the input values is **true**, then the result is **true**. Otherwise, the result is **false**.

This function can be extended to take additional arguments, see Add or Delete Function Arguments 438.



Languages

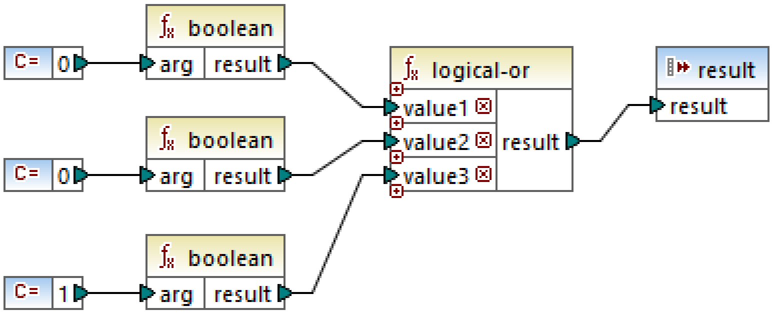
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first value to compare.
value2	Mandatory parameter. Provides the second value to compare.

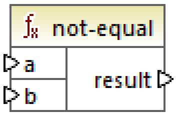
Example

The result of the mapping below is **true**, because at least one of the function's arguments is **true**.



5.9.7.5.9 not-equal

Returns Boolean **true** if *a* is not equal to *b*; **false** otherwise.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

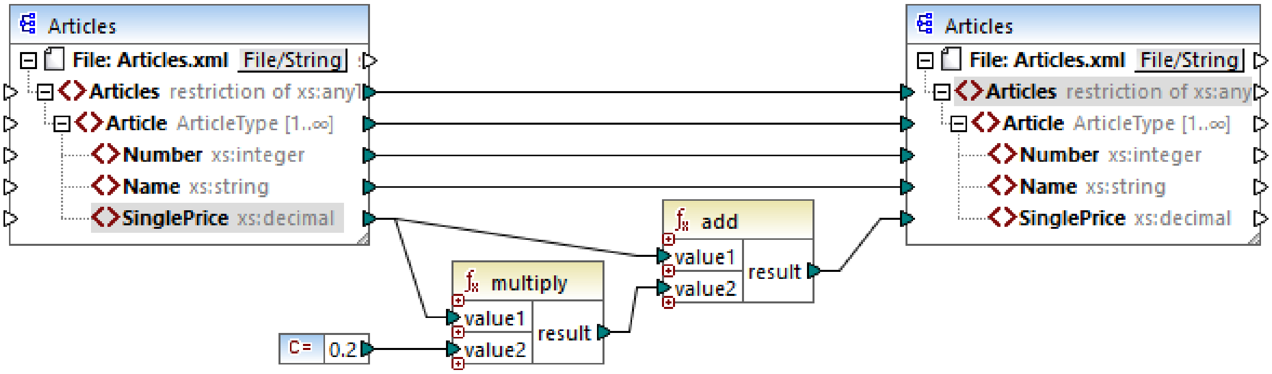
Parameters

Argument	Description
a	Mandatory parameter. Provides the first value to compare.
b	Mandatory parameter. Provides the second value to compare.

5.9.7.6 core | math functions

Math functions are used to perform basic mathematical operations on data. Note that they cannot be used to perform computations on durations or `datetime` values.

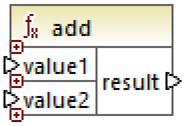
Most math functions take two input parameters (**value1**, **value2**) that are operands of the mathematical operation. The input values are automatically converted to `decimal` type for further processing. The result of math functions is also of `decimal` type.



The example shown above adds 20% sales tax to each of the articles mapped to the target component.

5.9.7.6.1 add

Adds **value1** to **value2** and returns the result as a decimal value. This function can be extended to take additional arguments, see Add or Delete Function Arguments 438.



Languages

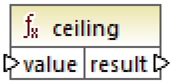
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

5.9.7.6.2 ceiling

Returns the smallest integer that is greater than or equal to **value**.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

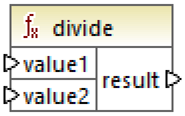
Argument	Description
value	Mandatory parameter. Provides the function's input value.

Example

If the input value is **11.2**, then applying the `ceiling` function to it makes the result **12**, i.e. the smallest integer that is greater than **11.2**.

5.9.7.6.3 divide

Divides **value1** by **value2** and returns the result as decimal value. The result precision depends on the target language. Use the round-precision 553 function to define the precision of result.



Languages

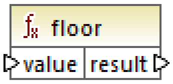
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

5.9.7.6.4 floor

Returns the greatest integer that is less than or equal to **value**.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

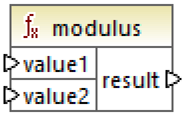
Argument	Description
value	Mandatory parameter. Provides the function's input value.

Example

If the input value is **11.7**, then applying the `floor` function to it makes the result **11**, i.e. the greatest integer than is less than **11.7**.

5.9.7.6.5 modulus

Returns the remainder of dividing **value1** by **value2**.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

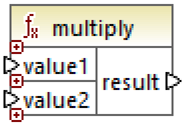
Example

If the input values are **1.5** and **1**, then the result of the `modulus` function is **0.5**. The explanation is that `1.5 / 1` leaves a remainder of **0.5**.

If the input values are **9** and **3**, then the result is **0**, since `9 / 3` leaves no remainder.

5.9.7.6.6 multiply

Multiplies **value1** by **value2** and returns the result as a decimal value.



Languages

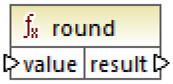
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

5.9.7.6.7 round

Returns the value rounded to the nearest integer. When the value is exactly in between two integers, the "Round Half Towards Positive Infinity" algorithm is used. For example, the value "10.5" gets rounded to "11", and the value "-10.5" gets rounded to "-10".



Languages

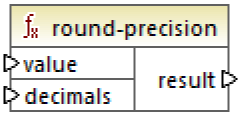
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
value	Mandatory parameter. Provides the function's input value.

5.9.7.6.8 round-precision

Rounds the input value to *N* decimal places, where *N* is the **decimals** argument.



Languages

Built-in, C++, C#, Java.

Parameters

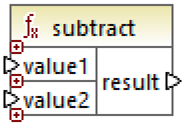
Argument	**Description**
value	Mandatory parameter. Provides the function's input value.
decimals	Mandatory parameter. Specifies the number of decimals to round to.

Example

Rounding the value **2.777777** to 2 decimals yields **2.78**. Rounding the value **0.1234** to 3 decimals yields **0.123**.

5.9.7.6.9 subtract

Subtracts **value2** from **value1** and returns the result as decimal value.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	**Description**
value1	Mandatory parameter. Provides the first operand.
value2	Mandatory parameter. Provides the second operand.

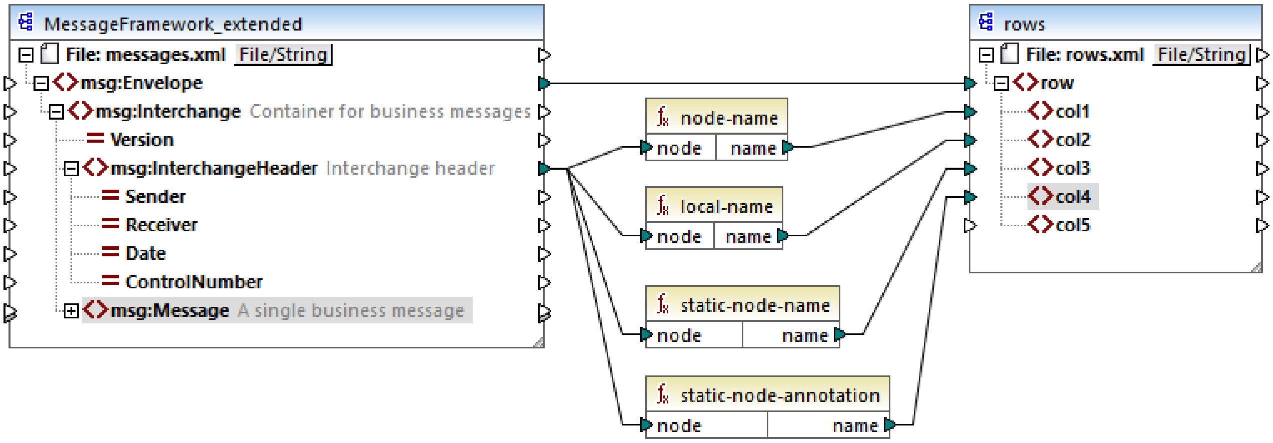
5.9.7.7 core | node functions

The functions from the **core | node functions** library allow you to access information about nodes on a mapping component (such as the node name or annotation), or to process nillable elements, see also Nil Values / Nillable 139.

Be aware that there is an alternative way to access node names, which does not require node functions at all, see Mapping Node Names 726.

The mapping illustrated below shows a few node functions that get information from the **msg:InterchangeHeader** node of the source XML file. More specifically, the following information is extracted:

1. The **node-name** function returns the qualified name of the node, which includes the node prefix.
2. The **local-name** function returns just the local part.
3. The **static-node-name** function is similar to the **node-name** function, but is available in XSLT 1.0 as well.
4. The **static-node-annotation** function gets the element's annotation as it was defined in the XML schema.

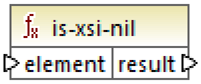

MessageFramework_extended
File: messages.xml File/String
msg:Envelope
msg:Interchange Container for business messages
Version
msg:InterchangeHeader Interchange header
Sender
Receiver
Date
ControlNumber
msg:Message A single business message
node-name
node name
local-name
node name
static-node-name
node name
static-node-annotation
node name
rows
File: rows.xml File/String
row
col1
col2
col3
col4
col5

The output of the mapping is as follows (excluding the XML and namespace declarations):

```
<row>
   <col1>msg:InterchangeHeader</col1>
   <col2>InterchangeHeader</col2>
   <col3>msg:InterchangeHeader</col3>
   <col4>Interchange header</col4>
</row>
```

5.9.7.7.1 is-xsi-nil

Returns **true** if the **element** node has the `xsi:nil` attribute set to **true**.



Languages

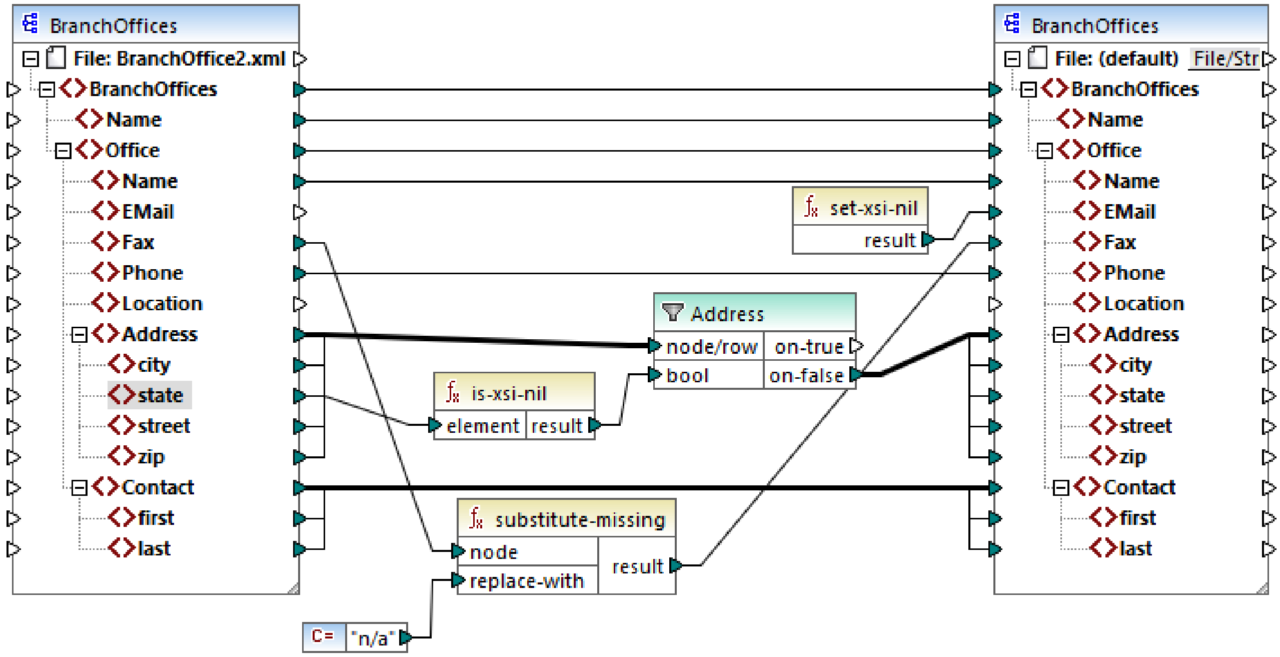
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
element	Mandatory parameter. Must be connected to the source node that is to be checked.

Example

The mapping design illustrated below copies data from a source to a target XML file conditionally, and also illustrates the usage of several functions, including **`is-xsi-nil`**. This mapping is called **HandlingXsiNil.mfd** and can be found in the **<Documents>\Altova\MapForce2025\MapForceExamples\** directory.



As illustrated above, the **is-xsi-nil** function checks whether the `xsi:nil` attribute is "true" for the **state** item in the source file. If this attribute is "false", the filter will copy the parent **Address** element to the target. The source XML file looks as follows (excluding the XML and namespace declarations):

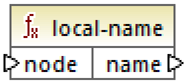
```
<BranchOffices>
   <Name>Nanonull</Name>
   <Office>
      <Name>Nanonull Research Outpost</Name>
      <EMail>sp@nanonull.com</EMail>
      <Fax xsi:nil="true"/>
      <Phone>+8817 3141 5926</Phone>
      <Address>
         <city>South Pole</city>
         <state xsi:nil="true"/>
         <street xsi:nil="true"/>
         <zip xsi:nil="true"/>
      </Address>
      <Contact>
         <first>Scott</first>
         <last>Amundsen</last>
      </Contact>
   </Office>
</BranchOffices>
```

The result of the mapping is that no **Address** is copied to the target at all, because there is only one **Address** in the source, and the `xsi:nil` attribute is set to "true" for the **state** element. Consequently, the mapping output is as follows:

```
<BranchOffices>
   <Name>Nanonull</Name>
   <Office>
      <Name>Nanonull Research Outpost</Name>
      <EMail xsi:nil="true"/>
      <Fax>n/a</Fax>
      <Phone>+8817 3141 5926</Phone>
      <Contact>
         <first>Scott</first>
         <last>Amundsen</last>
      </Contact>
   </Office>
</BranchOffices>
```

5.9.7.7.2 local-name

Returns the local name of the node. Unlike the node-name[557] function, **local-name** does not return the node's prefix. If the node does not have a prefix, then **local-name** and **node-name** return the same value.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

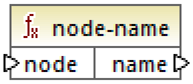
Parameters

Argument	Description
node	Mandatory parameter. Connect this input to the node whose name you want to get.

5.9.7.7.3 node-name

Returns the qualified name (QName) of the connected node. If the node is an XML **text()** node, an empty QName is returned. This function works only on those nodes that have a name. If XSLT 2.0 is the target language (which calls `fn:node-name`), the function returns an empty sequence for nodes which have no names.

Note: Getting the node name is not supported for "File input" nodes, database tables or fields, XBRL, Excel, JSON, or Protocol Buffers fields.



Languages

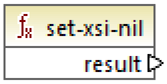
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
node	Mandatory parameter. Connect this input to the node whose name you want to get.

5.9.7.7.4 set-xsi-nil

Sets the target node to xsi:nil.



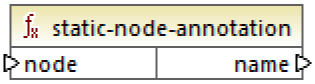
Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.7.5 static-node-annotation

Returns the string with annotation of the connected node. The input must be: (i) a source component node, or (ii) a user-defined function of type "inline[460]" that is directly connected to a parameter[463], which in turn is directly connected to a node in the calling mapping.

The connection must be direct. It cannot pass through a filter or a regular (not "inline") user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

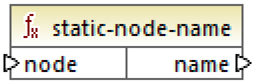
Parameters

Argument	Description
node	Mandatory parameter. Connect this input to the node whose annotation you want to get.

5.9.7.7.6 static-node-name

Returns the string with the name of the connected node. The input must be: (i) a source component node, or (ii) a user-defined function of type "inline[460]" that is directly connected to a parameter[463], which in turn is directly connected to a node in the calling mapping.

The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.



Languages

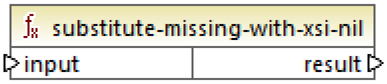
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Argument	Description
node	Mandatory parameter. Connect this input to the node whose name you want to get.

5.9.7.7.7 substitute-missing-with-xsi-nil

For nodes with simple content, this function substitutes any missing (or null values) of the source component, with the `xsi:nil` attribute in the target node.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

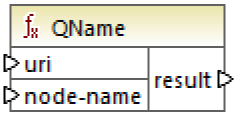
Argument	Description
input	Mandatory parameter. Connect this input to the node whose name you want to get.

5.9.7.8 core | QName functions

QName functions provide ways to manipulate the Qualified Names (QName) in XML documents.

5.9.7.8.1 QName

Constructs a QName from a namespace URI and a local part. Use this function to create a QName in a target component. The **uri** and **node-name** parameters can be supplied by a constant function.



Languages

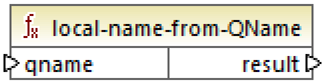
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
uri	Mandatory. Provides the URI.
node-name	Mandatory. Provides the name of the node.

5.9.7.8.2 local-name-from-QName

Extracts the local name part from a value of type `xs:QName`. Note that, unlike the local-name[556] function which returns the local name of the *node*, this function processes the *content* of the item connected to the **qname** input.



Languages

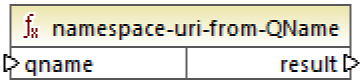
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
qname	Mandatory. Provides the function's input value, of type `xs:QName`.

5.9.7.8.3 namespace-uri-from-QName

Returns the namespace URI part of the QName value supplied as argument.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

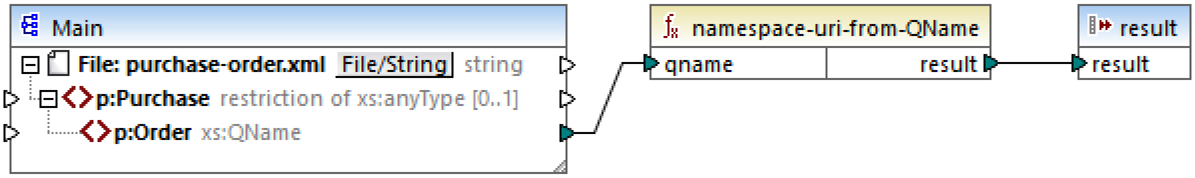
Name	Description
qname	Mandatory. Provides the function's input value.

Example

The following XML file contains a QName value, **o:name**. Note that the prefix "o" is mapped to the namespace **http://NamespaceTest.com/Order**.

```
<?xml version="1.0" encoding="utf-8"?>
<p:Purchase xsi:schemaLocation="http://NamespaceTest.com/Purchase Main.xsd"
          xmlns:p="http://NamespaceTest.com/Purchase"
          xmlns:o="http://NamespaceTest.com/Order"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <p:Order>o:name</p:Order>
</p:Purchase>
```

A mapping that processes the QName value and gets the namespace URI is illustrated below:



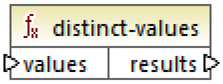
The output of this mapping is **http://NamespaceTest.com/Order**.

5.9.7.9 core | sequence functions

Sequence functions allow processing of input sequences[78] and grouping of their content.

5.9.7.9.1 distinct-values

Processes the sequence of values connected to the **values** input and returns only the distinct values, as a sequence. This is useful when you need to remove duplicate values from a sequence and copy only the unique items to the target component.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

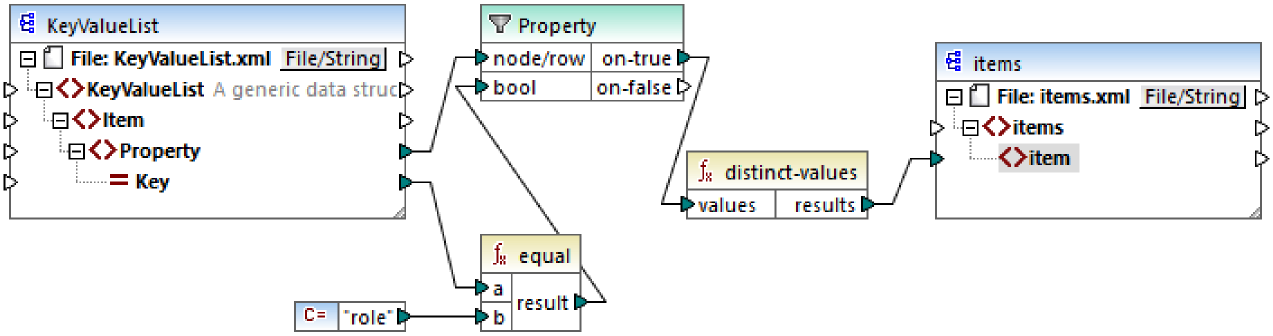
Name	Description
values	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.

Example

The following XML file contains information about employees of a demo company. Some employees have the same role; therefore, the "role" attribute role contains duplicate values. For example, both "Loby Matise" and "Susi Sanna" have the role "Support".

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyValueList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="KeyValueList.xsd">
   <Item>
      <Property Key="role">Manager</Property>
      <Property Key="First">Vernon</Property>
      <Property Key="Last">Callaby</Property>
   </Item>
   <Item>
      <Property Key="role">Programmer</Property>
      <Property Key="First">Frank</Property>
      <Property Key="Last">Further</Property>
   </Item>
   <Item>
      <Property Key="role">Support</Property>
      <Property Key="First">Loby</Property>
      <Property Key="Last">Matise</Property>
   </Item>
   <Item>
      <Property Key="role">Support</Property>
      <Property Key="First">Susi</Property>
      <Property Key="Last">Sanna</Property>
   </Item>
</KeyValueList>
```

Let's suppose that you need to extract a list of all *unique* role names that occur in this XML file. This can be achieved with a mapping like the one below:



In the mapping above, the following happens:

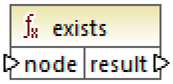
- Each **Property** element from the source XML file is processed by a filter.
- The connection to the filter's **bool** input ensures that only **Property** elements where the **Key** attribute is equal to "role" are supplied to the target component. The string "role" is provided by a constant. Note that the filter's output still produces duplicates at this stage (since there are two "Support" properties that meet the filter's condition).
- The sequence produced by the filter is processed by the `distinct-values` function, which excludes any duplicate values.

As a result, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>Manager</item>
   <item>Programmer</item>
   <item>Support</item>
</items>
```

5.9.7.9.2 exists

Returns **true** if the connected node exists; **false** otherwise. Since it returns a Boolean value, this function is typically used with filters 408, to filter out only records which have (or perhaps do not have) a child element or attribute.



Languages

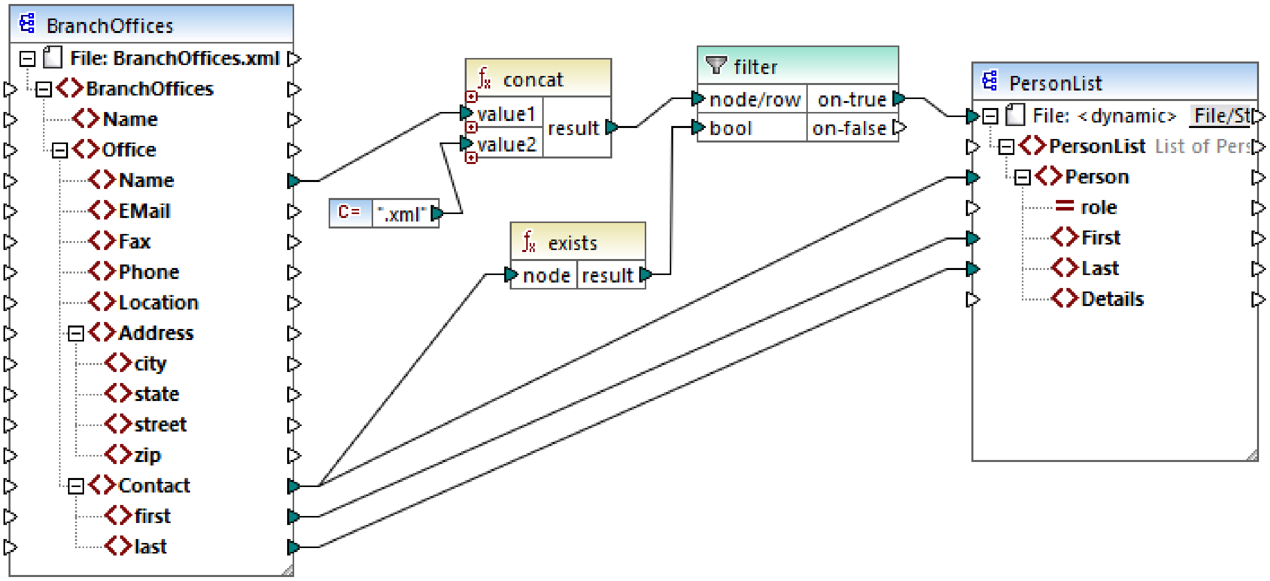
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node	The node to be tested for existence.

Examples

The following mapping illustrates how to filter data with the help of the `exists` function. This mapping is called **PersonListsForAllBranchOffices.mfd** and it can be found in the **<Documents>\Altova\MapForce2025\MapForceExamples\** directory.



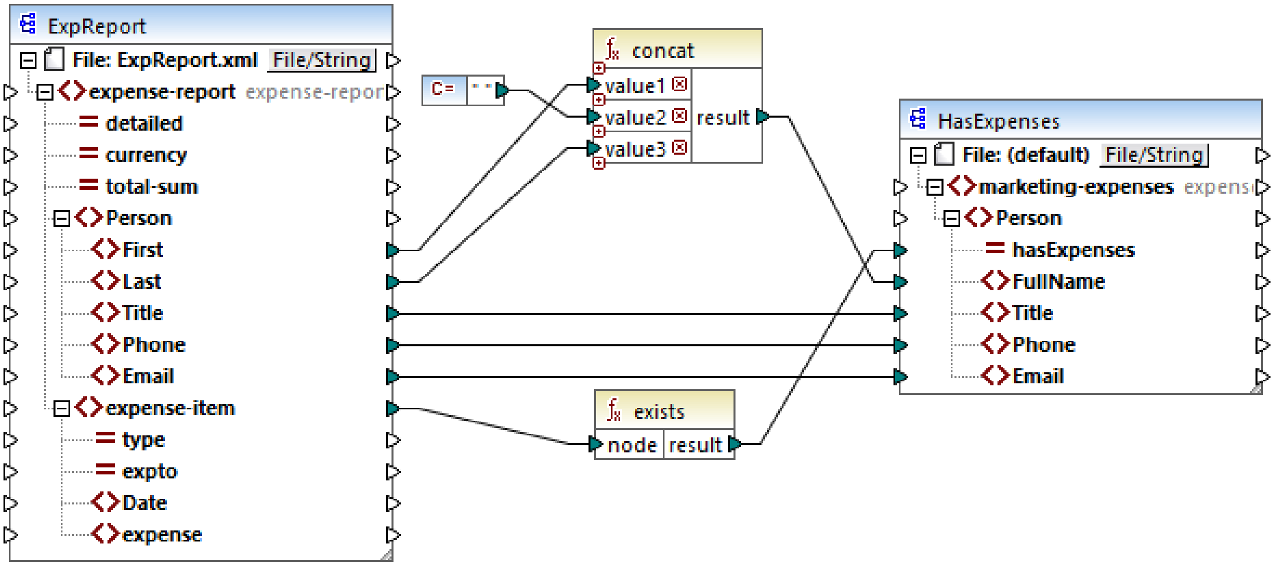
PersonListsForAllBranchOffices.mfd

In the source file **BranchOffices.xml**, there are three **Office** elements. Notably, one of the offices does not have any **Contact** child elements. The goal of the mapping is many-fold:

a) for each office, extract a list of contacts that exist in that office
b) for each office, create a separate XML file with the same name as the office
c) do not generate the XML file if the office has no contacts.

To achieve these goals, a filter was added to the mapping. The filter passes on to the target only those **Office** items where at least one **Contact** item exists. This Boolean condition is provided by the `exists` function. If the function's result is true, then the name of the office is concatenated with the string `.xml` in order to produce the target file name. For more information about generating file names from the mapping, see Processing Multiple Input or Output Files Dynamically 746 .

Another example is the following mapping: **<Documents>\Altova\MapForce2025\MapForceExamples\HasMarketingExpenses.mfd**. Here, if an **expense-item** exists in the source XML, then the **hasExpenses** attribute is set to **true** in the target XML file.

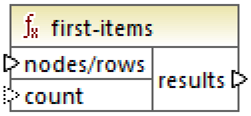


HasMarketingExpenses.mfd

See also Example: Exception When Node Does Not Exist 433.

5.9.7.9.3 first-items

Returns the first *N* items of the input sequence, where *N* is supplied by the **count** parameter.



Languages

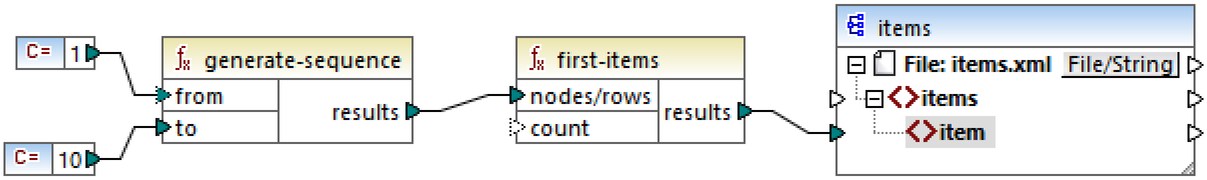
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence 78 of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Optional parameter. Specifies how many items should be retrieved from the input sequence. The default value is **1**.

Example

The following mock-up mapping generates a sequence of 10 values. The sequence is processed by the `first-items` function and the result is written to a target XML file.



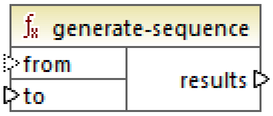
Because the **count** argument has no value, the default value of **1** applies. As a result, only the first value from the sequence is generated in the mapping output:

```
<items>
   <item>1</item>
</items>
```

For a more realistic example, see the **FindHighestTemperatures.mfd** mapping discussed in Supplying Parameters to the Mapping 346.

5.9.7.9.4 generate-sequence

Creates a sequence of integers using the "from" and "to" parameters as the boundaries.



Languages

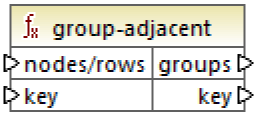
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
from	Optional parameter. Specifies the integer that the sequence should start with (lower boundary). The default value is **1**.
to	Mandatory parameter. Specifies the integer that the sequence should end with (upper boundary).

5.9.7.9.5 group-adjacent

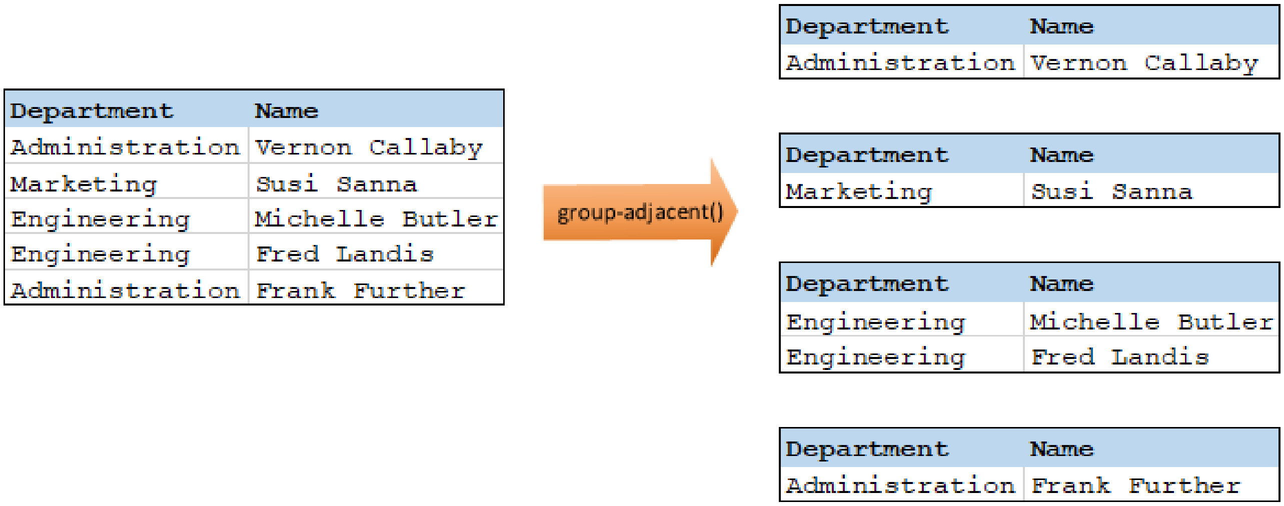
The **group-adjacent** function groups the items connected to the **nodes/rows** input by the key connected to the **key** input. Note that this function places items that share the same key into separate groups if they are not adjacent. If multiple consecutive (adjacent) items share the same key, they are placed into the same group.



For example, in the abstract transformation illustrated below, the grouping key is "Department". The left side of the diagram shows the input data while the right side shows the output data after grouping. The following takes place when the transformation runs:

- Initially, the first key, "Administration", creates a new group.
- The next key is different, so a second group is created, "Marketing".
- The third key is also different, so another group is created, "Engineering".
- The fourth key is the same as the third; therefore, this record is placed in the already existing group.
- Finally, the fifth key is different from the fourth, and this creates the last group.

As illustrated below, "Michelle Butler" and "Fred Landis" were grouped together because they have the same key and are adjacent. However, "Vernon Callaby" and "Frank Further" are in separate groups because they are not adjacent, even though they have the same key.



Department	Name
Administration	Vernon Callaby
Marketing	Susi Sanna
Engineering	Michelle Butler
Engineering	Fred Landis
Administration	Frank Further

Department	Name
Administration	Vernon Callaby

Department	Name
Marketing	Susi Sanna

Department	Name
Engineering	Michelle Butler
Engineering	Fred Landis

Department	Name
Administration	Frank Further

Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

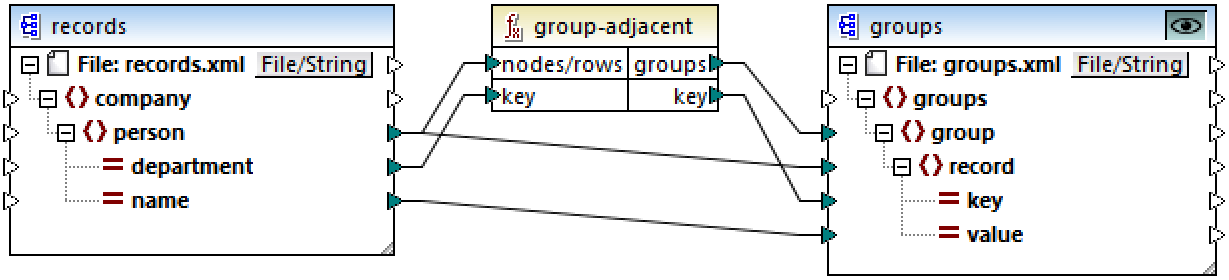
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence 78 of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
key	The key by which to group items.

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records by department, provided they are adjacent. To achieve this, the following mapping invokes the `group-adjacent` function, and supplies **department** as **key**.



The mapping result is as follows:

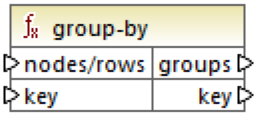
```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
   </group>
   <group>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
```

```
    </group>
    <group>
        <record key="Administration" value="Frank Further"/>
    </group>
</groups>
```

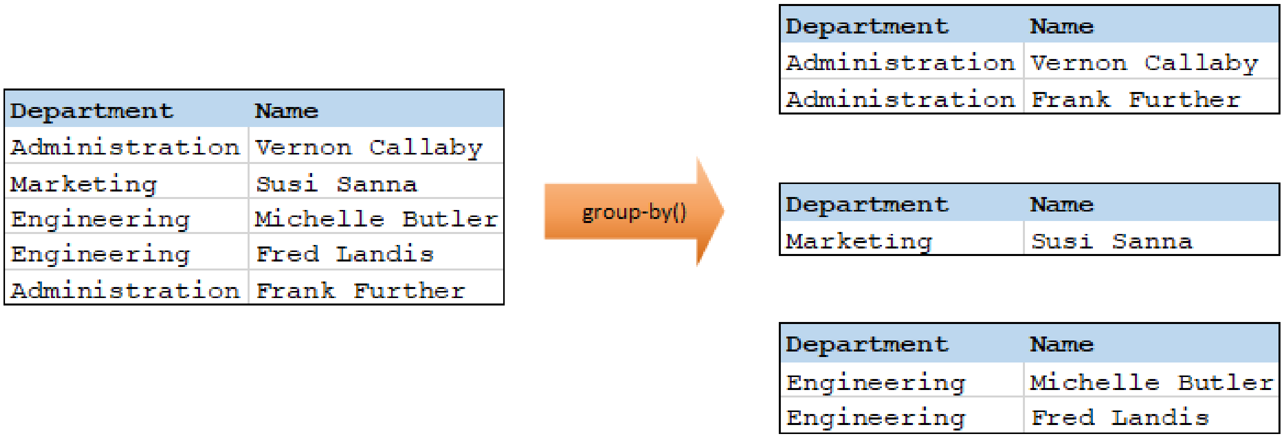
This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview** button applicable to the function you want to preview, before clicking the **Output** pane.

5.9.7.9.6 group-by

The **group-by** function creates groups of records according to some grouping key that you specify.



For example, in the abstract transformation illustrated below, the grouping key is "Department". Since there are three unique departments in total, applying the group-by function would create three groups:



Department	Name
Administration	Vernon Callaby
Marketing	Susi Sanna
Engineering	Michelle Butler
Engineering	Fred Landis
Administration	Frank Further

Department	Name
Administration	Vernon Callaby
Administration	Frank Further

Department	Name
Marketing	Susi Sanna

Department	Name
Engineering	Michelle Butler
Engineering	Fred Landis

Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so

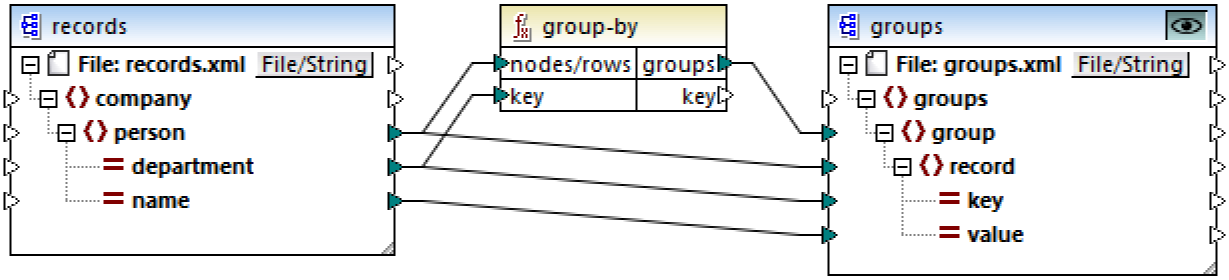
Name	Description
	on.
key	The key by which to group items.

Example 1

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records by department. To achieve this, the following mapping invokes the **group-by** function, and supplies **department** as key.



The mapping result is as follows:

```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
      <record key="Administration" value="Frank Further"/>
   </group>
   <group>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
   </group>
</groups>
```

This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview** 👁 button applicable to the function you want to preview, before clicking the **Output** pane.

Example 2

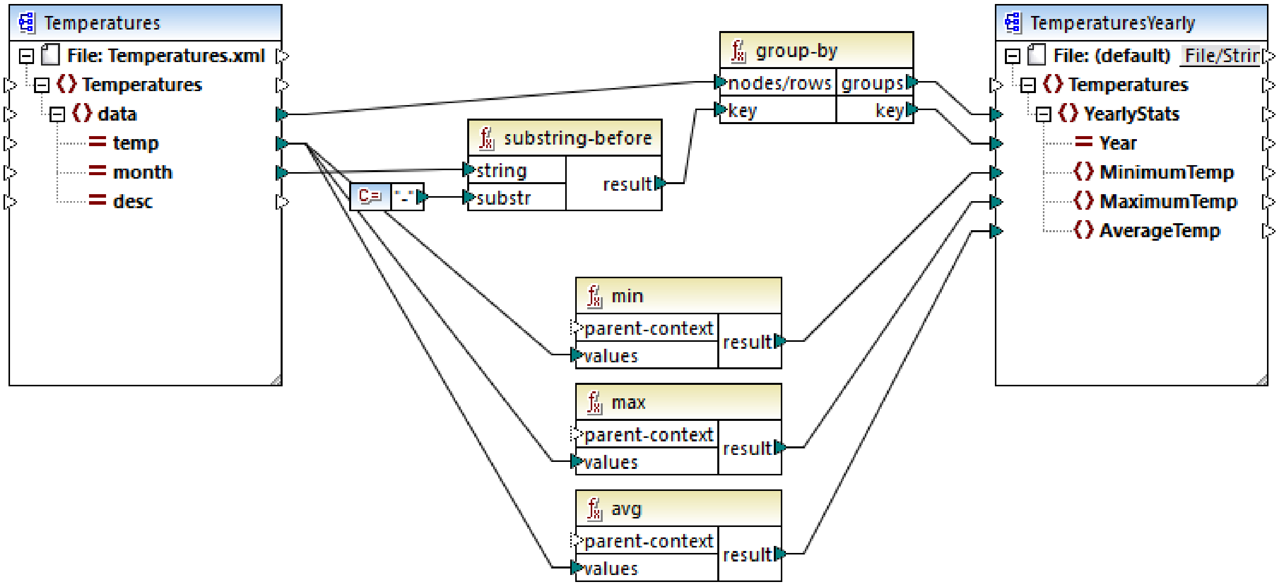
This example shows you how to group records with the help of the `group-by` function, and also illustrates how to aggregate data. This example is accompanied by a demo mapping available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\GroupTemperaturesByYear.mfd**. This mapping reads data from an XML file that contains a log of monthly temperatures, as illustrated in the code listing below:

```
<Temperatures>
   <data temp="-3.6" month="2006-01" />
   <data temp="-0.7" month="2006-02" />
   <data temp="7.5" month="2006-03" />
   <data temp="12.4" month="2006-04" />
   <data temp="16.2" month="2006-05" />
   <data temp="19" month="2006-06" />
   <data temp="22.7" month="2006-07" />
   <data temp="23.2" month="2006-08" />
   <data temp="18.7" month="2006-09" />
   <data temp="11.2" month="2006-10" />
   <data temp="9.1" month="2006-11" />
   <data temp="0.8" month="2006-12" />
   <data temp="-3.2" month="2007-01" />
   <data temp="-0.3" month="2007-02" />
   <data temp="6.5" month="2007-03" />
   <data temp="10.6" month="2007-04" />
   <data temp="19" month="2007-05" />
   <data temp="20.3" month="2007-06" />
   <data temp="22.3" month="2007-07" />
   <data temp="20.7" month="2007-08" />
   <data temp="19.2" month="2007-09" />
   <data temp="12.9" month="2007-10" />
   <data temp="8.1" month="2007-11" />
   <data temp="1.9" month="2007-12" />
</Temperatures>
```

The business requirement of this mapping is two-fold:

1. Group temperatures of each year together.
2. Find out the minimum, maximum, and the average temperature of each year.

To achieve the first goal, we use the `group-by` function. To achieve the second goal, we use the `min`[518], `max`[516], and `avg`[515] aggregation functions.



GroupTemperaturesByYear.mfd

The way MapForce executes a mapping (and the recommended approach to start reading one) is by looking at the topmost item of the target component. In this example, an **YearlyStats** item will be created for each group returned by the **group-by** function. The **group-by** function takes as first argument all **data** items from the source and groups them by whatever is connected to the **key** input. Since the requirement is to group temperatures by year, the year must be obtained first. To achieve this, the **substring-before** 597 function extracts the year part from the **month** attribute of each **data** element. Namely, it takes as argument the value of **month** and returns the part before the first occurrence of **substr**. As illustrated above, in this example, **substr** is set to the dash character; therefore, if given the value "2006-01", the function will return "2006".

Finally, the values of **MinimumTemp**, **MaximumTemp**, and **AverageTemp** are obtained by connecting these items with the respective aggregate functions: **min**, **max**, and **avg**. All three functions take as input the sequence of temperatures read from the source component. These functions do not need a **parent-context** argument, because they already work in the context of each group. In other words, there is a parent connection —from **data** to **YearlyStats**— which provides the context for each aggregation function to work on.

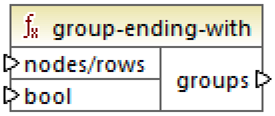
To preview the mapping output, click the **Output** pane. Notice that the number of groups coincides with the number of years obtained by reading the source file, for example:

```
<Temperatures>
   <YearlyStats Year="2006">
      <MinimumTemp>-3.6</MinimumTemp>
      <MaximumTemp>23.2</MaximumTemp>
      <AverageTemp>11.375</AverageTemp>
   </YearlyStats>
   <YearlyStats Year="2007">
      <MinimumTemp>-3.2</MinimumTemp>
      <MaximumTemp>22.3</MaximumTemp>
      <AverageTemp>11.5</AverageTemp>
   </YearlyStats>
</Temperatures>
```

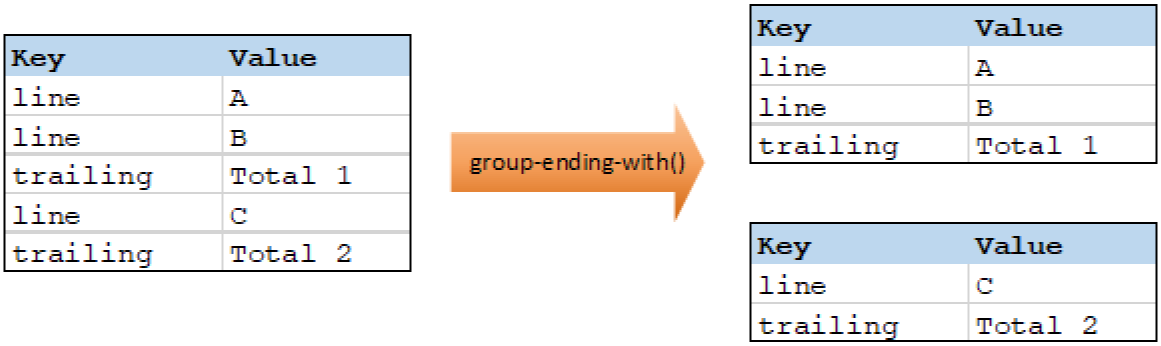
Note: For simplicity, the code listings above contain less data than the actual input and output used by the demo mapping.

5.9.7.9.7 group-ending-with

The **group-ending-with** function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, ending with the record that satisfies the condition.



In the example below, the condition is that "Key" must be equal to "trailing". This condition is true for the third and fifth records, so two groups are created as a result:



Key	Value
line	A
line	B
trailing	Total 1
line	C
trailing	Total 2

Key	Value
line	A
line	B
trailing	Total 1

Key	Value
line	C
trailing	Total 2

Note: One additional group is created if records exist after the last one that satisfies the condition. For example, if there were more "line" records after the last "trailing" record, these would all be placed into a new group.

Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

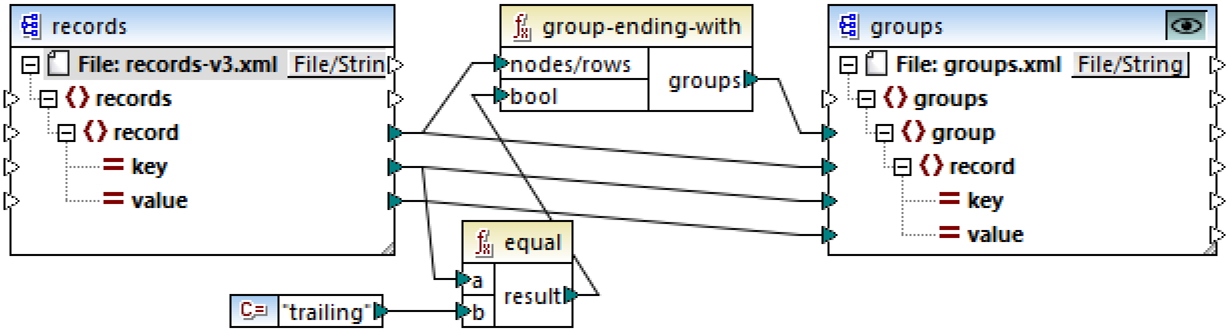
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
bool	Provides the Boolean condition that starts a new group when **true**.

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<records>
   <record key="line" value="A"/>
   <record key="line" value="B"/>
   <record key="trailing" value="Total 1"/>
   <record key="line" value="C"/>
   <record key="trailing" value="Total 2"/>
</records>
```

The business requirement is to create groups for each "trailing" record. Each group must also include any "line" records that precede the "trailing" record. To achieve this, the following mapping invokes the `group-ending-with` function. In the mapping below, whenever the **key** name is equal to "trailing", the argument supplied to **bool** becomes **true**, and a new group is created.



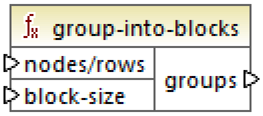
The mapping result is as follows:

```
<groups>
   <group>
      <record key="line" value="A"/>
      <record key="line" value="B"/>
      <record key="trailing" value="Total 1"/>
   </group>
   <group>
      <record key="line" value="C"/>
      <record key="trailing" value="Total 2"/>
   </group>
</groups>
```

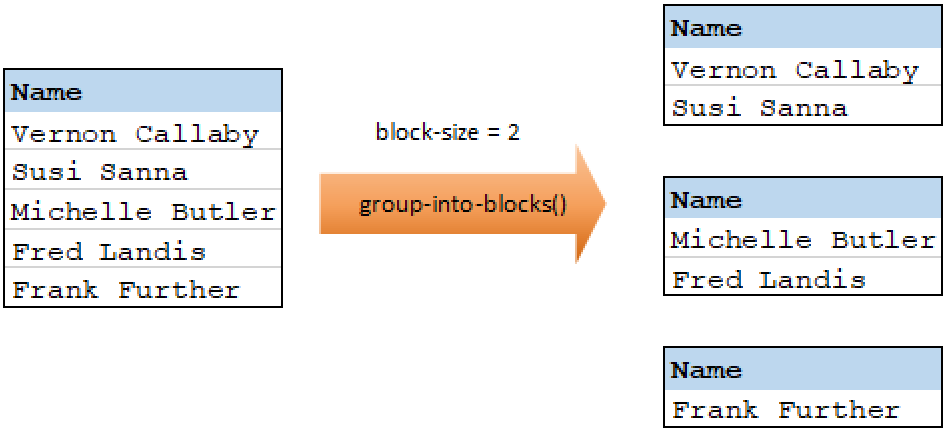
This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview** button applicable to the function you want to preview, before clicking the **Output** pane.

5.9.7.9.8 group-into-blocks

The **group-into-blocks** function creates equal groups that contain exactly N items, where N is the value you supply to the `block-size` argument. Note that the last group may contain N items or less, depending on the number of items in the source.



In the example below, `block-size` is 2. Since there are five items in total, each group contains exactly two items, except for the last one.



Name
Vernon Callaby
Susi Sanna
Michelle Butler
Fred Landis
Frank Further

Name
Vernon Callaby
Susi Sanna

Name
Michelle Butler
Fred Landis

Name
Frank Further

Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

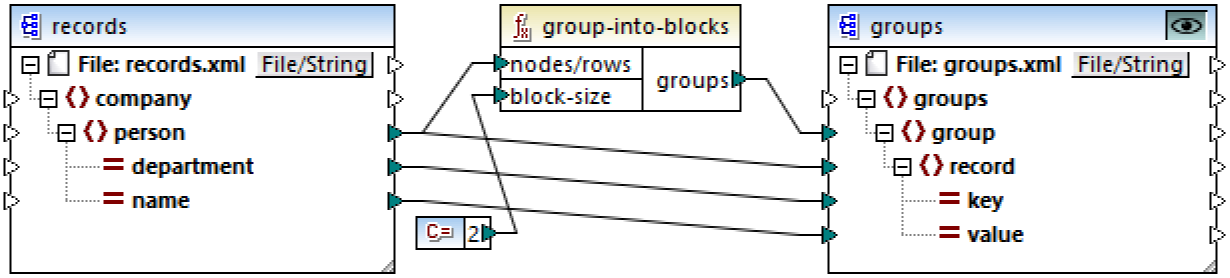
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
block-size	Specifies the size of each group

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

The business requirement is to group person records into blocks of two items each. To achieve this, the following mapping invokes the `group-into-blocks` function, and supplies the integer value "2" as **block-size**.



The mapping result is as follows:

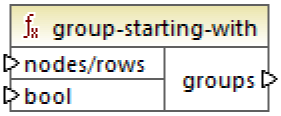
```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
   </group>
   <group>
      <record key="Administration" value="Frank Further"/>
   </group>
</groups>
```

Note that the last group contains only one item, since the total number of items (5) cannot be divided evenly by 2.

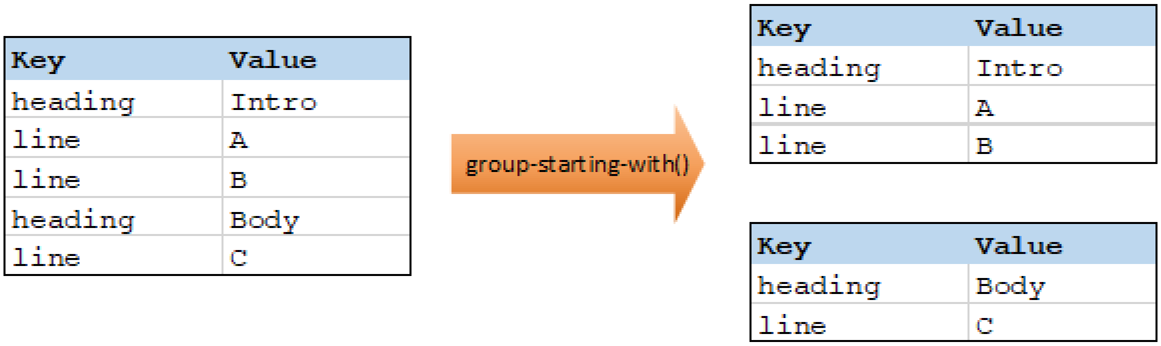
This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview** button applicable to the function you want to preview, before clicking the **Output** pane.

5.9.7.9.9 group-starting-with

The `group-starting-with` function takes a Boolean condition as argument. If the Boolean condition is true, a new group is created, starting with the record that satisfies the condition.



In the example below, the condition is that "Key" must be equal to "heading". This condition is true for the first and fourth records, so two groups are created as a result:



Key	Value
heading	Intro
line	A
line	B
heading	Body
line	C

Key	Value
heading	Intro
line	A
line	B

Key	Value
heading	Body
line	C

Note: One additional group is created if records exist before the first one that satisfies the condition. For example, if there were more "line" records before the first "heading" record, these would all be placed into a new group.

Languages

Built-in, C++, C#, Java, XSLT 2.0, XSLT 3.0.

Parameters

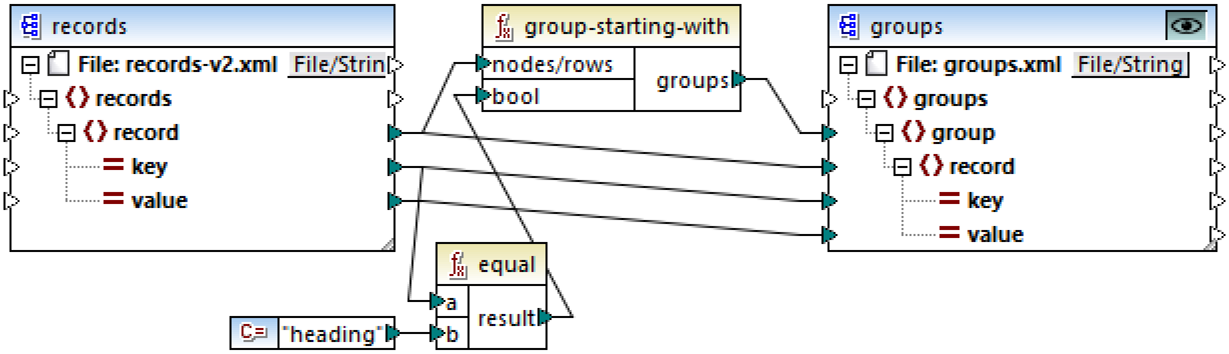
Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
bool	Provides the Boolean condition that starts a new group when **true**.

Example

Let's assume that your source data is an XML file with the following content (note that, in the code listing below, the namespace and XML declarations were removed for simplicity).

```
<records>
   <record key="heading" value="Intro"/>
   <record key="line" value="A"/>
   <record key="line" value="B"/>
   <record key="heading" value="Body"/>
   <record key="line" value="C"/>
</records>
```

The business requirement is to create groups for each "heading" record. Each group must also include any "line" records that follow the "heading" record. To achieve this, the following mapping invokes the `group-starting-with` function. In the mapping below, whenever the **key** name is equal to "heading", the argument supplied to **bool** becomes **true**, and a new group is created.



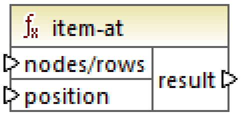
The mapping result is as follows:

```
<groups>
   <group>
      <record key="heading" value="Intro"/>
      <record key="line" value="A"/>
      <record key="line" value="B"/>
   </group>
   <group>
      <record key="heading" value="Body"/>
      <record key="line" value="C"/>
   </group>
</groups>
```

This example, together with other grouping examples, is part of the following mapping file: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\GroupingFunctions.mfd**. Remember to click the **Preview** button applicable to the function you want to preview, before clicking the **Output** pane.

5.9.7.9.10 item-at

Returns an item from the sequence of **nodes/rows** supplied as argument, at the position supplied by the **position** argument. The first item is at position **1**.



Languages

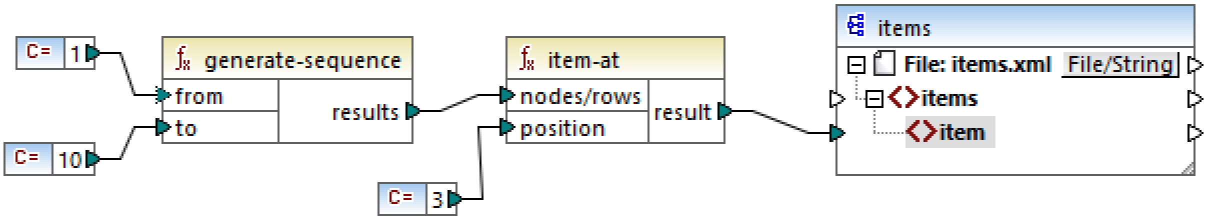
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
position	This integer specifies which item from the sequence of items is to be returned.

Example

The following mock-up mapping generates a sequence of 10 values. The sequence is processed by the `item-at` function and the result is written to a target XML file.

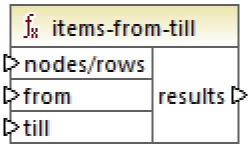


Because the **position** argument is set to **3**, only the third value from the sequence is passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>3</item>
</items>
```

5.9.7.9.11 items-from-till

Returns a sequence of **nodes/rows** using the "from" and "till" parameters as the boundaries. The first item is at position **1**.



Languages

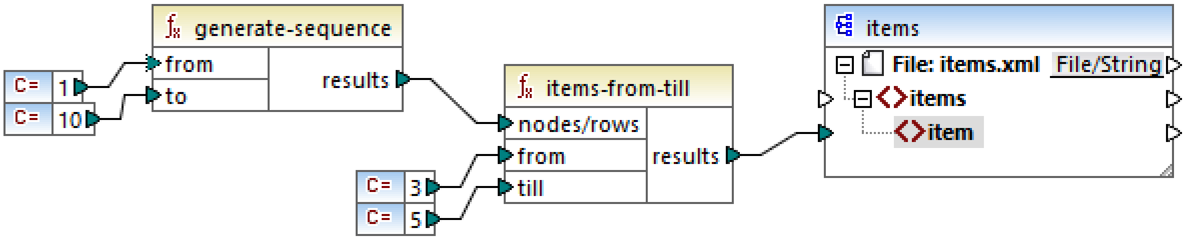
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
from	This integer specifies the starting position from which items must be retrieved.
till	This integer specifies the position up to which items must be retrieved.

Example

The following mock-up mapping generates a sequence of 10 values. The sequence is processed by the **items-from-till** function and the result is written to a target XML file.

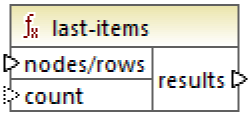


Because the **from** and **till** arguments are set to **3** and **5**, respectively, only the subset of values from **3** through **5** are passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>3</item>
   <item>4</item>
   <item>5</item>
</items>
```

5.9.7.9.12 last-items

Returns the last *N* items of the input sequence, where *N* is supplied by the **count** parameter. The first item is at position "1".



Languages

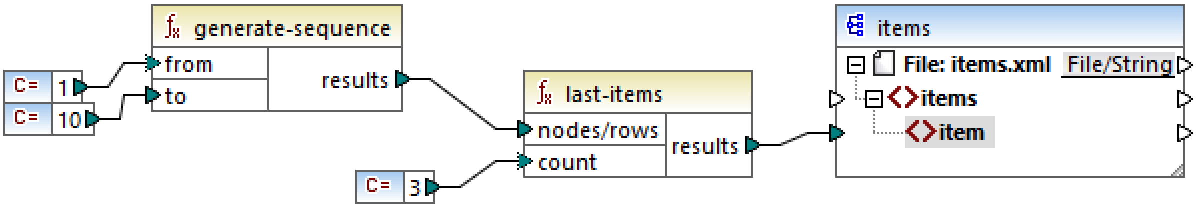
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
nodes/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Optional parameter. Specifies how many items should be retrieved from the input sequence. The default value is **1**.

Example

The following mock-up mapping generates a sequence of 10 values. The sequence is processed by the `last-items` function and the result is written to a target XML file.

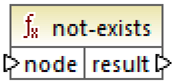


Because the count argument is set to **3**, only the last three values from the sequence are passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>8</item>
   <item>9</item>
   <item>10</item>
</items>
```

5.9.7.9.13 not-exists

Returns **false** if the connected node exists; **true** otherwise. This function is the opposite of exists[563] function, but, otherwise, it has the same use.



Languages

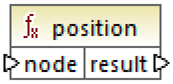
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node	The node to be tested for non-existence.

5.9.7.9.14 position

Returns the position of an item within the sequence of items currently being processed. This can be used, for example, to auto-number items sequentially.



Languages

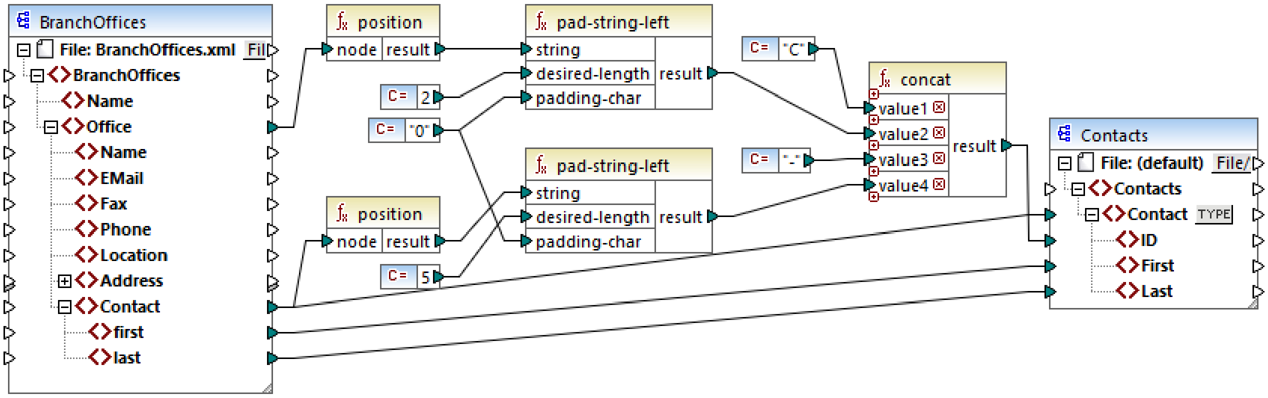
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.

Example

The following mapping illustrates using the `position` function in order to generate unique identification values in data generated by the mapping. This mapping is accompanied by a mapping design file that is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\ContactsFromBranchOffices.mfd**.



ContactsFromBranchOffices.mfd

In the mapping above, the source XML file contains three branch offices. A branch office may contain an arbitrary number of **Contact** child items. The goals of the mapping are as follows:

- Extract all **Contact** items from the source XML file and write them to the target XML file.
- Each contact must be assigned a unique identification number (the **ID** item in the target XML).
- The ID of each contact must take the form **`CXX-YYYYY`**, where X identifies the office number, and Y identifies the contact number. If the office number is less than two characters, it must be left-padded with zeros. Likewise, if the contact number takes less than five characters, it must be left-padded with zeros. Consequently, a valid identification number of the first contact from the first office should look like **`C01-00001`**.

To achieve the mapping goals, several MapForce functions have been used, including the **`position`** function. The upper **`position`** function gets the position of each office. The lower one gets the position of each contact, in the context of each office.

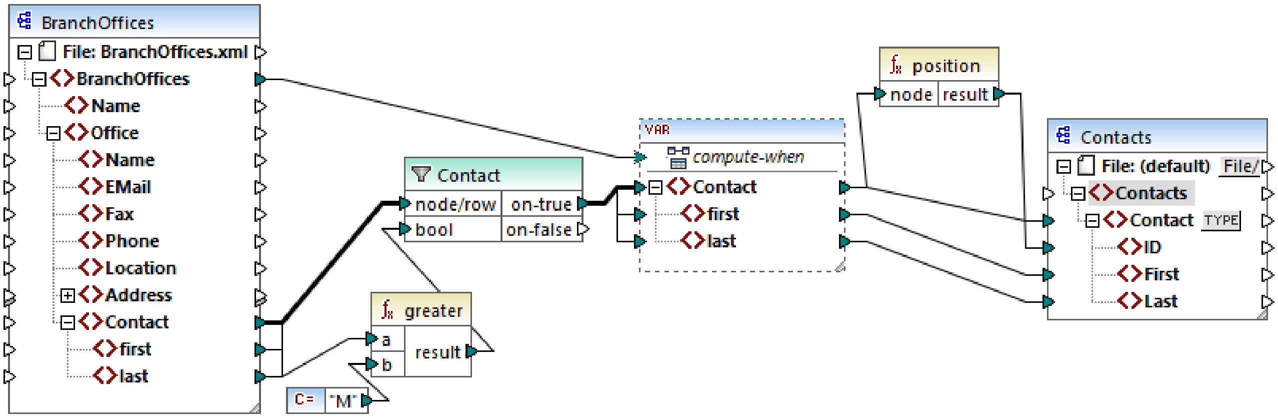
When using the **`position`** function, it is important to consider the current mapping context[79]. More specifically, when the mapping runs, the initial mapping context is established from the root item of the target component to the source item connected to it (even indirectly via functions). In this example, the upper **`position`** function processes *the sequence of all offices* and it initially generates the value 1, corresponding to the first office in the sequence. The lower **`position`** function generates sequential numbers corresponding to the contact's position *in the context of that office* (1, 2, 3, and so on). Note that this "inner" sequence will be reset (and thus start from 1 again) when the next office gets processed. Both **`pad-string-left`** functions apply padding to the generated numbers, according to the requirements stated previously. The `concat` function operates *in the context of each contact* (because of the parent connection from the source to the target **Contact**). It joins all the computed values and returns the unique identification number of each contact.

The output generated from the mapping above is shown below (note that some of the records were removed for readability):

```
<Contacts>
   <Contact>
      <ID>C01-00001</ID>
      <First>Vernon</First>
      <Last>Callaby</Last>
   </Contact>
```

```
    <Contact>
       <ID>C01-00002</ID>
       <First>Frank</First>
       <Last>Further</Last>
    </Contact>
    <!-- ... -->
    <Contact>
       <ID>C02-00001</ID>
       <First>Steve</First>
       <Last>Meier</Last>
    </Contact>
    <Contact>
       <ID>C02-00002</ID>
       <First>Theo</First>
       <Last>Bone</Last>
    </Contact>
    <!-- ... -->
 </Contacts>
```

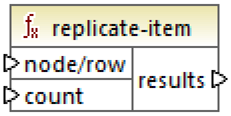
There may also be cases where you need to get the position of items resulting after applying a filter 408. Note that the filter component is not a sequence function, and it cannot be used *directly* in conjunction with the **position** function to find the position of filtered items. Indirectly, this is possible by adding a variable 360 component to the mapping. For example, the mapping below is a simplified version of the previous one. Its mapping design file is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\PositionInFilteredSequence.mfd**.



The result of variables in MapForce are always sequences. Therefore, in the mapping above, the **position** function iterates through the sequence created by the variable and returns the position of each item in that sequence. This mapping is discussed in more detail in Example: Filtering and Numbering Nodes 369.

5.9.7.9.15 replicate-item

Repeats every item in the input sequence the number of times specified in the **count** argument. If you connect a single item to the **node/row** input, the function returns *N* items, where *N* is the value of the **count** argument. If you connect a sequence of items to the **node/row** input, the function repeats each individual item in the sequence **count** times, processing one item at a time. For example, if count is **2**, then the sequence `1,2,3` produces `1,1,2,2,3,3`. It is also possible to supply a different **count** value for each item in the input sequence, as illustrated in the example below.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

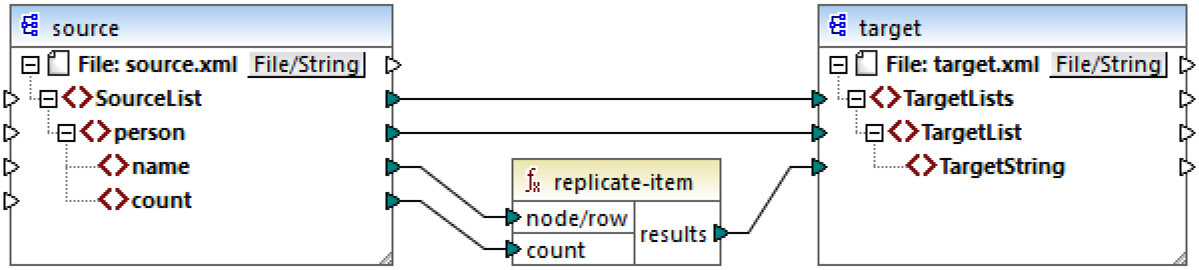
Name	Description
node/row	This input must receive a connection from a mapping item that provides a sequence 78 of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Specifies the number of times to replicate each item or sequence connected to **node/row**.

Example

Let's assume that you have a source XML file with the following structure:

```
<SourceList>
   <person>
      <name>Michelle</name>
      <count>2</count>
   </person>
   <person>
      <name>Ted</name>
      <count>4</count>
   </person>
   <person>
      <name>Ann</name>
      <count>3</count>
   </person>
</SourceList>
```

With the help of the `replicate-item` function, you can repeat each person name a different number of times in a target component. To achieve this, connect the **count** node of each person to the **count** input of the `replicate-item` function:

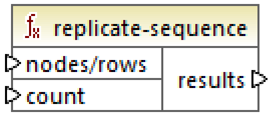


The output is as follows:

```
<TargetLists>
   <TargetList>
      <TargetString>Michelle</TargetString>
      <TargetString>Michelle</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
   </TargetList>
</TargetLists>
```

5.9.7.9.16 replicate-sequence

Repeats all items in the input sequence the number of times specified in the **count** argument. For example, if count is **2**, then the sequence `1,2,3` produces `1,2,3,1,2,3`.



Languages

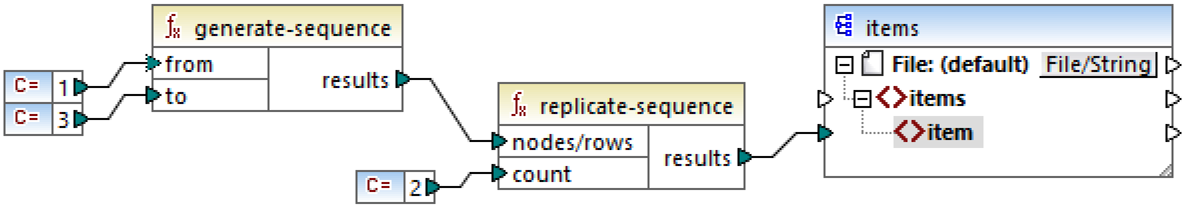
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Specifies the number of times to replicate the connected sequence.

Example

The following mock-up mapping generates the sequence `1,2,3`. The sequence is processed by the `replicate-sequence` function and the result is written to a target XML file.

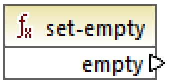


Because the **count** argument is set to **2**, the sequence is replicated twice and then passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>1</item>
   <item>2</item>
   <item>3</item>
   <item>1</item>
   <item>2</item>
   <item>3</item>
</items>
```

5.9.7.9.17 set-empty

Returns an empty sequence.

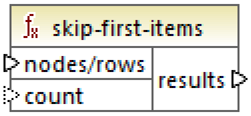


Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.9.18 skip-first-items

Skips the first N items of the input sequence, where N is supplied by the **count** argument, and returns the rest of the sequence.



Languages

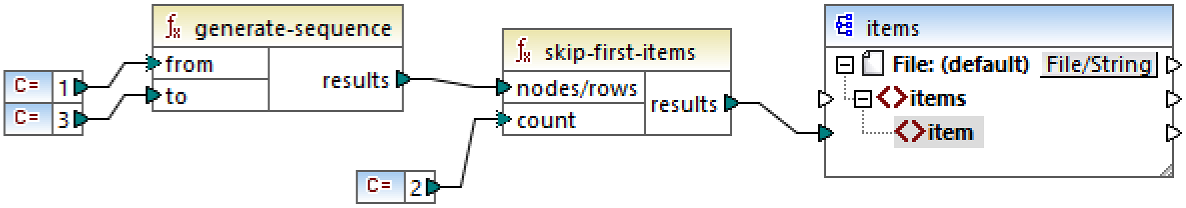
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
node/rows	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
count	Optional argument. Specifies the number of items to skip. The default value is **1**.

Example

The following mock-up mapping generates the sequence `1,2,3`. The sequence is processed by the `skip-first-items` function and the result is written to a target XML file.

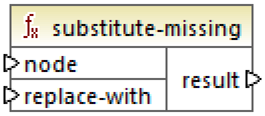


Because the **count** argument is set to **2**, the first two items are skipped and the remaining items are passed on to the target. Consequently, the mapping output is as follows (excluding the XML and schema declarations):

```
<items>
   <item>3</item>
</items>
```

5.9.7.9.19 substitute-missing

This function is a convenient combination of exists[563] function and if-else condition[412]. If the item connected to the **node** input exists, its content will be copied to the target. Otherwise, the content of the item connected to the **replace-with** input will be copied to the target.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

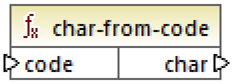
Name	Description
node	This input must receive a connection from a mapping item that provides a sequence[78] of zero or more values. For example, the connection may originate from a source XML item, a CSV field, a database record, and so on.
replace-with	This input must receive a connection from a mapping item that provides the replacement value.

5.9.7.10 core | string functions

The string functions allow you to manipulate string data so as to extract parts of strings, test for sub-strings, retrieve information from strings, split strings, and others.

5.9.7.10.1 char-from-code

Returns the character representation of the decimal Unicode value (code) supplied as argument. **Tip:** To find the Unicode decimal code of a character, you can use the **`code-from-char`**[591] function.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

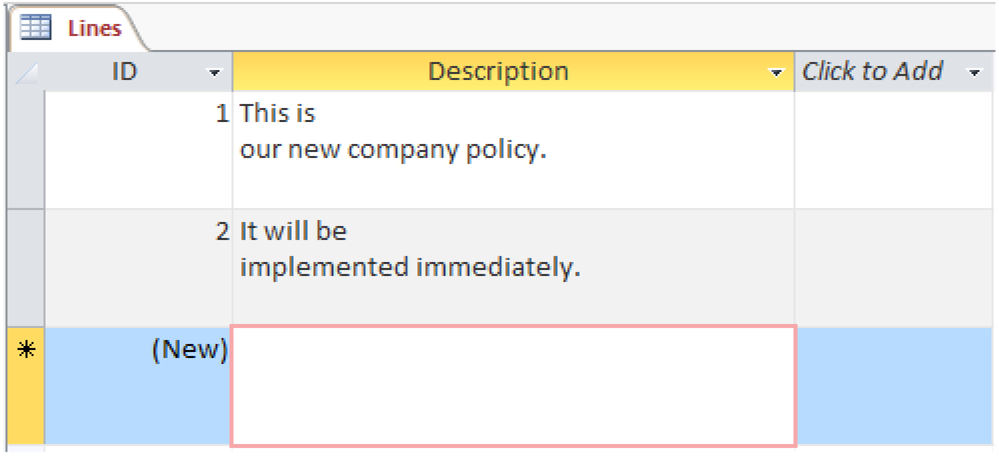
Name	Description
code	The Unicode value, as a decimal number.

Example 1

According to the charts available on the Unicode website (https://www.unicode.org/charts/), the exclamation mark character has the hexadecimal value of `0021`. The corresponding value in decimal format is **33**. Therefore, supplying `33` as argument to the `char-from-code` function will return the `!` character.

Example 2 (Professional and Enterprise editions)

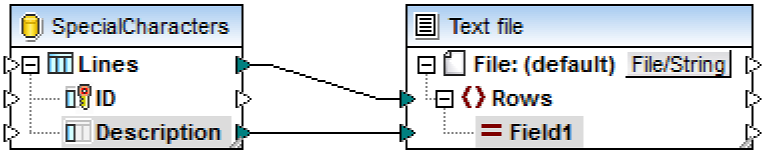
This example shows how to replace special characters in a database with space characters. Consider an SQLite database consisting of a table "Lines" which has two columns: "ID" and "Description".



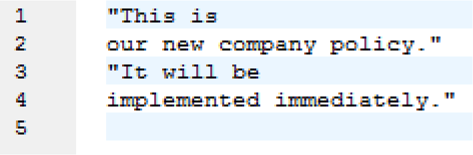
Lines

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:



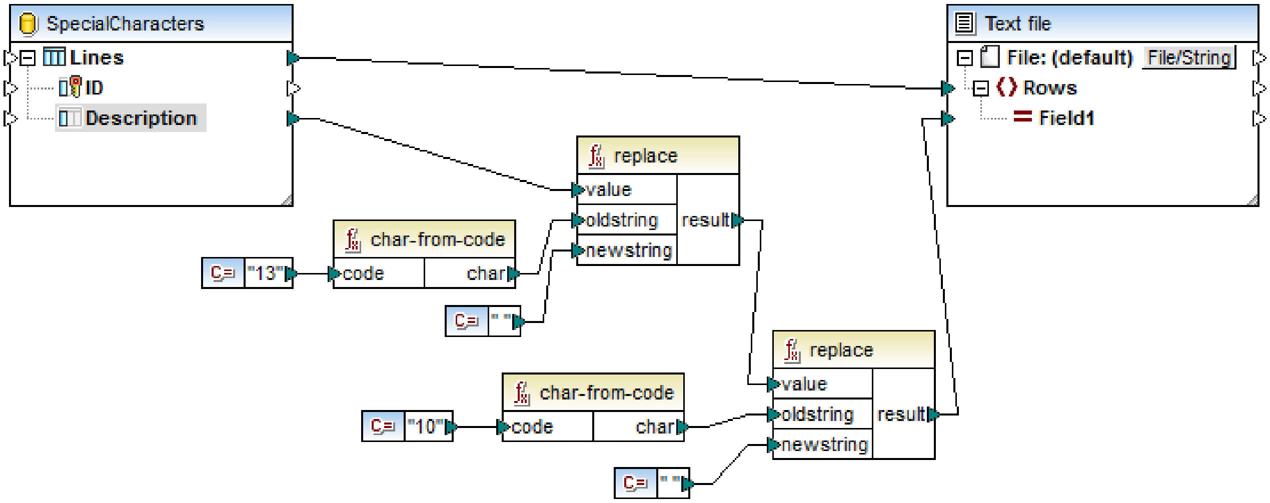
However, because each "Description" row in Access contains multiple lines separated by CR/LF characters, the mapping output includes line breaks also, which is not the intended result:



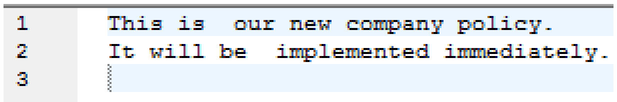
```
"This is
our new company policy."
"It will be
implemented immediately."

```

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. 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.



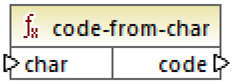
If you preview the mapping now, notice that the CR/LF characters within each database field have been replaced by a space.



```
This is  our new company policy.
It will be  implemented immediately.

```

5.9.7.10.2 code-from-char

Returns the decimal Unicode value (code) of the character supplied as argument. If the string supplied as argument has multiple characters, then the code of the first character is returned.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

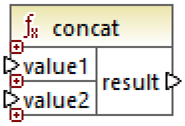
Name	Description
char	The input string value.

Example

If the input **char** is the **$** (dollar sign) character, the function returns **36** (which is the decimal Unicode value for this character).

5.9.7.10.3 concat

Concatenates (appends) two or more values into a single result string. All input values are automatically converted to type "string". By default, this function has only two parameters, but you can add more. Click **Add parameter** (⊞) or **Delete parameter** (⊠) to add or remove parameters.



Note: All the inputs to the `concat` function must have a value. If any of the inputs does not have a value, the function is not called and an error occurs. Be aware that an empty string is a valid input value; however, an empty sequence (such as the result of the **`set-empty`** function) is not a valid value and the function will fail as a result. To prevent this from happening, you can first process values with the substitute-missing 589 function and then supply the result as input to the **`concat`** function.

Languages

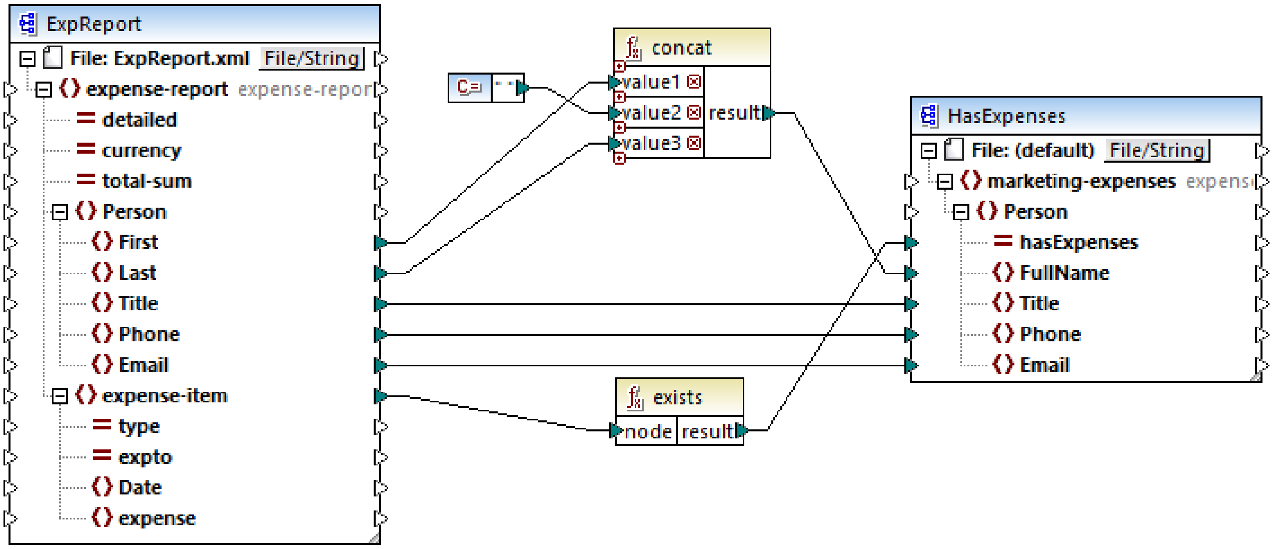
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
value1	The first input value.
value2	The second input value.
valueN	The *N* input value.

Example

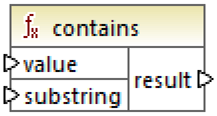
In the mapping illustrated below, the **concat** function joins the first name, the constant " ", and the last name. The returning value is then written to the **FullName** target item. The mapping of this function is available at the following path: **<Documents>\Altova\MapForce2025\MapForceExamples\HasMarketingExpenses.mfd**.



HasMarketingExpenses.mfd

5.9.7.10.4 contains

Returns Boolean **true** if the string value supplied as argument contains the sub-string supplied as argument.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

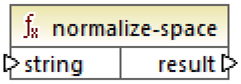
Name	Description
value	The input value (that is, the "haystack").
substring	The sub-string to look for (that is, the "needle").

Example

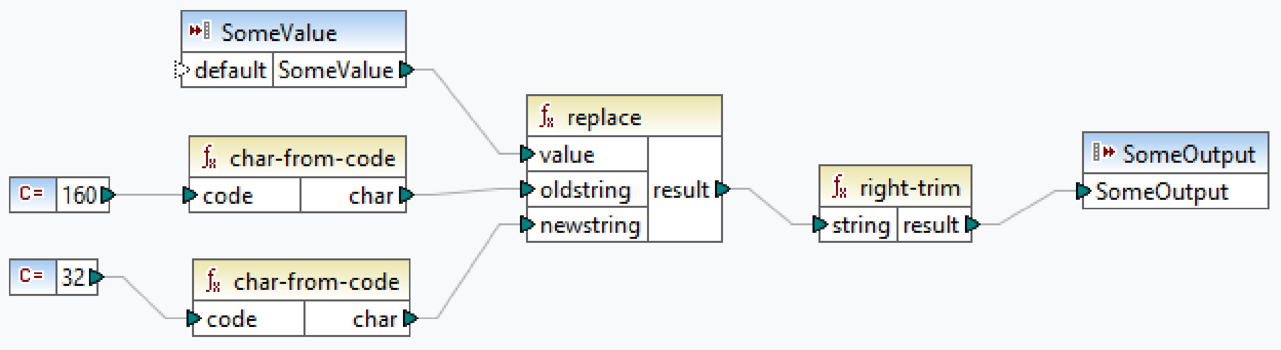
If the input **value** is "category" and **substring** is "cat", the function returns **true**.

5.9.7.10.5 normalize-space

The **normalize-space** function (*see screenshot below*) removes leading and trailing spaces of a string and replaces internal whitespaces with a single whitespace character. Whitespace includes space (U+0020), tab (U+0009), carriage return (U+000D), and line feed (U+000A) characters. For details about whitespaces, see the XML Recommendation.



About non-breaking spaces
The **left-trim**, **right-trim**, and **normalize-space** functions do not remove non-breaking spaces. One of the possible solutions could be to replace the non-breaking space character, whose decimal representation is 160, with the space character, whose decimal representation is 32. The mapping below shows that after the non-breaking space has been replaced, the trimmed `SomeValue` value will be mapped to the target.



If your source component is an Excel file, you can remove extra spaces in Excel using a combination of TRIM, CLEAN, and SUBSTITUTE functions. For details, see *Removing Spaces and Nonprinting Characters from Text*.

Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

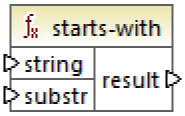
Name	Description
string	The input string to normalize.

Example

If the input string is "`  The quick  brown fox  `", the function returns "`The quick brown fox`".

5.9.7.10.6 starts-with

Returns Boolean **true** if the string supplied as argument starts with the sub-string supplied as argument; **false** otherwise.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

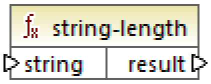
Name	Description
string	The input string.
substr	The sub-string to check for.

Example

If the input **string** is `category` and **substr** is `cat`, the function returns **true**.

5.9.7.10.7 string-length

Returns the number of characters in the string supplied as argument.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

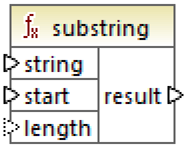
Name	Description
string	The input string.

Example

If the input string is **car**, the function returns **3**. If the input string is an empty string, the function returns **0**.

5.9.7.10.8 substring

Returns the portion of the string specified by the **start** and **length** parameters.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

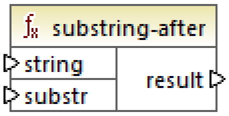
Name	Description
string	The input string.
start	Specifies the starting position (index) from which the sub-string should be retrieved. The first index is **1**.
length	Optional. Specifies the number of characters to retrieve. If the **length** parameter is not specified, the result is a fragment starting from **start** until the end of the string.

Example

If the input string is **MapForce**, start is **1**, and length is **3**, the function returns **Map**. If the input string is **MapForce**, start is **4**, and length is not provided, the function returns **Force**.

5.9.7.10.9 substring-after

Returns the portion of the string that occurs after the first occurrence of **substr**. If **substr** does not occur in **string**, the function returns an empty string.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

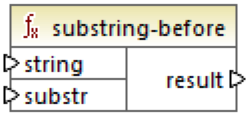
Name	Description
string	The input string.
substr	The sub-string. Any characters after the first occurrence of **substr** are the result of the function.

Example

If the input string is `MapForce`, and **substr** is `Map`, the function returns `Force`. If the input string is `2020/01/04` and **substr** is `/`, the function returns `01/04`.

5.9.7.10.10 substring-before

Returns the portion of the string that occurs before the first occurrence of **substr**. If **substr** does not occur in **string**, the function returns an empty string.



Languages

Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

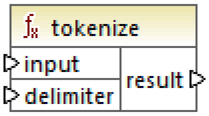
Name	Description
string	The input string.
substr	The sub-string. Any characters before the first occurrence of **substr** are the result of the function.

Example

If the input string is `MapForce`, and **substr** is `Force`, the function returns `Map`. If the input string is `2020/01/04` and **substr** is `/`, the function returns `2020`.

5.9.7.10.11 tokenize

Splits the input string into a sequence of strings using the delimiter supplied as argument.



Languages

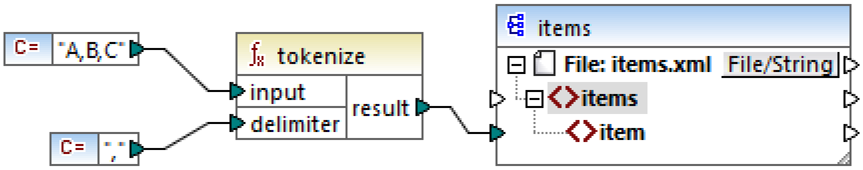
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
input	The input string.
delimiter	The delimiter to use.

Example

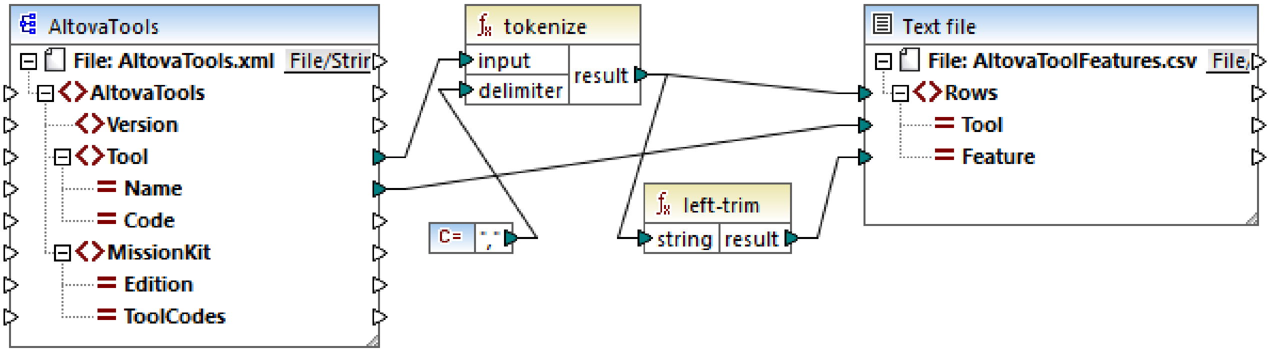
If the input string is `A,B,C` and the delimiter is `,` then the function returns a sequence of three strings: `A`, `B`, and `C`.



In the mock-up mapping illustrated above, the function's result is a sequence of strings. According to the general mapping rules 78, for each item in the source sequence, a new **item** is created in the target component. Consequently, the mapping output looks as follows:

```
<items>
   <item>A</item>
   <item>B</item>
   <item>C</item>
</items>
```

For a more elaborate example, see the **tokenizeString1.mfd** mapping available in the **<Documents>\Altova\MapForce2025\MapForceExamples\** folder.



tokenizeString1.mfd

A fragment from the source XML file is shown below. The **Tool** element has two attributes: **Name** and **Code**. The **Tool** element data consists of comma-delimited text.

```
<?xml version="1.0" encoding="UTF-8"?>
<AltovaTools xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AltovaTools.xsd">
   <Version>2010</Version>
   <Tool Name="XMLSpy" Code="XS">XML editor, XSLT editor, XSLT debugger, XQuery editor,
XQuery debugger, XML Schema / DTD editor, WSDL editor, SOAP debugger</Tool>
   <Tool Name="MapForce" Code="MF">Data integration, XML mapping, database mapping, text
conversion, EDI translator, Excel mapping, XBRL mapping, Web services</Tool>
   <Tool Name="StyleVision" Code="SV">Stylesheet designer, electronic forms, XSLT design,
XSL:FO design, database reporting, XBRL rendering</Tool>
   <Tool Name="UModel" Code="UM">UML modeling tool, code generation, reverse engineering,
UML, BPMN, SysML, project documentation, XMI interchange</Tool>
   <Tool Name="DatabaseSpy" Code="DS">Multi-database tool, SQL auto-completion, graphical
database design, table browser, content editor, database comparison tool</Tool>
   <!-- ... -->
</AltovaTools>
```

The mapping does the following:

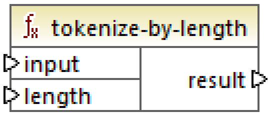
- The `tokenize` function receives data from the **Tool** source item and uses the comma `,` delimiter to split that data into separate chunks. The first chunk is "XML editor", the second one is "XSLT editor", and so on.
- For each chuck resulting from the `tokenize` function, a new row is generated in the target. This happens thanks to the connection between the function's result and the **Rows** item in the target component.
- The result of the `tokenize` function is also mapped to the `left-trim` function, which removes the leading white space of each chunk.
- The result of the `left-trim` function (each chunk) is written to the **Feature** item of the target component.
- The target component output file has been defined as a CSV file (**AltovaToolFeatures.csv**) with the field delimiter being a semicolon (double click component to see settings).

The result of the mapping is that, for each chunk created by the **tokenize** function, a new row is created in the target CSV file. A fragment of the mapping output looks as follows:

```
Tool;Feature
XMLSpy;XML editor
XMLSpy;XSLT editor
XMLSpy;XSLT debugger
XMLSpy;XQuery editor
XMLSpy;XQuery debugger
XMLSpy;XML Schema / DTD editor
XMLSpy;WSDL editor
XMLSpy;SOAP debugger
MapForce;Data integration
MapForce;XML mapping
MapForce;database mapping
MapForce;text conversion
MapForce;EDI translator
MapForce;Excel mapping
```

5.9.7.10.12 tokenize-by-length

Splits the input string into a sequence of strings. The size of each resulting string is determined by the **length** parameter.



Languages

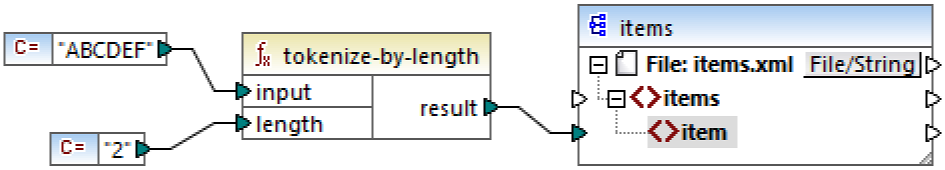
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
input	The input string.
length	Determines the length of each string in the generated sequence of strings.

Example

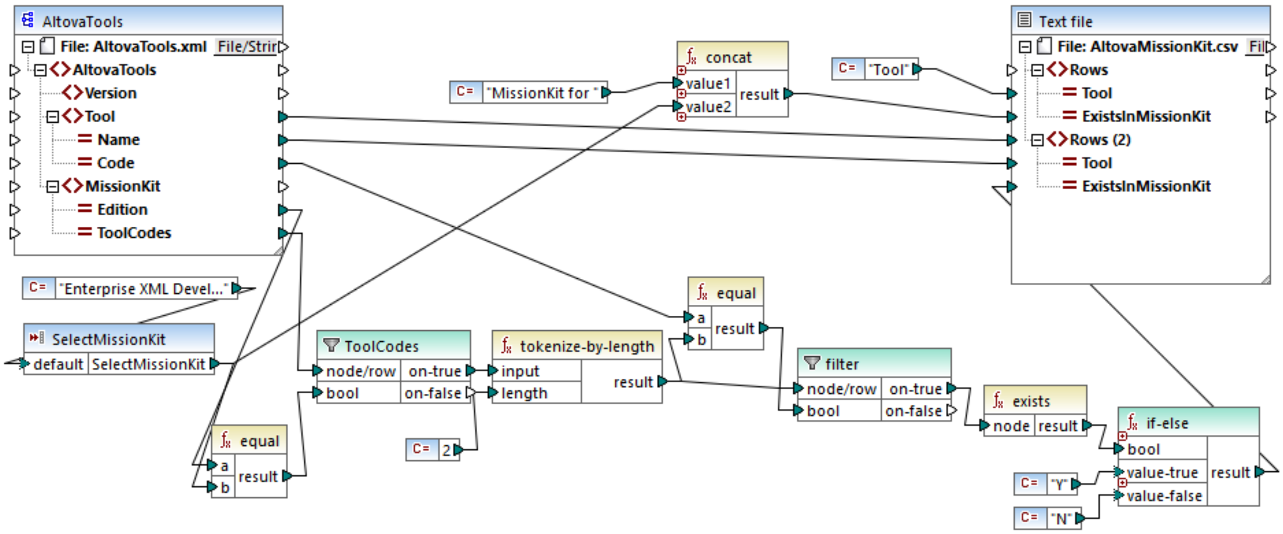
If the input string is **ABCDEF** and the length is **2**, then the function returns a sequence of three strings: **AB**, **CD**, and **EF**.



In the mock-up mapping illustrated above, the function's result is a sequence of strings. According to the general mapping rules 78, for each item in the source sequence, a new **item** is created in the target component. Consequently, the mapping output looks as follows:

```
<items>
   <item>AB</item>
   <item>CD</item>
   <item>EF</item>
</items>
```

For a more elaborate example, see the **tokenizeString2.mfd** mapping available in the **<Documents>\Altova\MapForce2025\MapForceExamples\** folder.



tokenizeString2.mfd

The XML source file is shown below, and is the same as the one used in the previous example. The **MissionKit** element has two attributes: **Edition** and **ToolCodes**, but no MissionKit element content. Note that some of the XML content not relevant to this example has been removed.

```
<?xml version="1.0" encoding="UTF-8"?>
<AltovaTools xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AltovaTools.xsd">
   <Version>2010</Version>
   <Tool Name="XMLSpy" Code="XS"><!--...--></Tool>
   <Tool Name="MapForce" Code="MF"><!--...--></Tool>
   <Tool Name="StyleVision" Code="SV"><!--...--></Tool>
   <Tool Name="UModel" Code="UM"><!--...--></Tool>
   <Tool Name="DatabaseSpy" Code="DS"><!--...--></Tool>
   <Tool Name="DiffDog" Code="DD"><!--...--></Tool>
   <Tool Name="SchemaAgent" Code="SA"><!--...--></Tool>
   <Tool Name="SemanticWorks" Code="SW"><!--...--></Tool>
   <Tool Name="Authentic" Code="AU"><!--...--></Tool>
   <MissionKit Edition="Enterprise Software Architects" ToolCodes="XSMFSVUMDSDDSASW"/>
   <MissionKit Edition="Professional Software Architects" ToolCodes="XSMFSVUMDS"/>
   <MissionKit Edition="Enterprise XML Developers" ToolCodes="XSMFSVDDSASW"/>
   <MissionKit Edition="Professional XML Developers" ToolCodes="XSMFSV"/>
</AltovaTools>
```

The aim of the mapping is to generate a list showing which Altova tools are part of the respective MissionKit editions.

How the mapping works:

- The **SelectMissionKit** input component acts as a parameter to the mapping; it receives its default value from a constant, in this case "Enterprise XML Developers".
- The **equal** function compares the edition supplied as parameter with the **Edition** item from the source XML file and passes on the result to the **bool** parameter of the **ToolCodes** filter.
- The **node/row** input of the **ToolCodes** filter is supplied by the **ToolCodes** item of the source file. The value for the "Enterprise XML Developers" edition is: **XSMFSVDDSASW**.
- The **XSMFSVDDSASW** value is passed to the **on-true** parameter, and further to the **input** parameter of the **tokenize-by-length** function.
- The **tokenize-by-length** function splits the value **XSMFSVDDSASW** into multiple chunks of two characters each. The **length** parameter is **2**; therefore 6 chunks are created as a result.
- Each chunk is compared to the 2-character **Code** value from the source file (of which there are 9 items in total). The result of the comparison (true/false) is passed on to the **bool** parameter of the filter. Note that *all* chunks produced by the **tokenize-by-length** function are passed on to the **node/row** parameter of the filter.
- The **exists** functions now checks for existing/non-existing nodes passed on to it by the **on-true** parameter of the filter component. Existing nodes are those where *there is a match* between the **ToolCodes** chunk and the **Code** value. Non-existing nodes are those where there was no **ToolCodes** chunk to match a **Code** value.
- Each **bool** result of the **exists** function is passed on to the **if-else** component, which generates a "Y" in the target if the node exists, or an "N" if the node does not exist.

The result of the mapping is as follows:

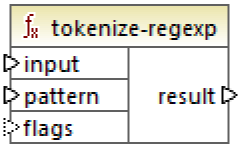
```
Tool;MissionKit for Enterprise XML Developers
XMLSpy;Y
MapForce;Y
StyleVision;Y
UModel;N
DatabaseSpy;N
DiffDog;Y
SchemaAgent;Y
```

```
SemanticWorks;Y
Authentic;N
```

5.9.7.10.13 tokenize-regexp

Splits the input string into a sequence of strings. Any substring that matches the regular expression **pattern** supplied as argument defines the separator. The matched (separator) strings are not included in the result returned by the function.

Note: When generating C++, C#, or Java code, the advanced features of the regular expression syntax might differ slightly. See the regex documentation of each language for more information.



Languages

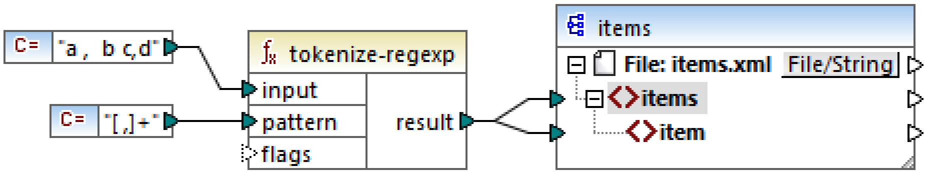
Built-in, C++, C#, Java, XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
input	The input string.
pattern	Provides a regular expression pattern. Any substring that matches the pattern will be treated as delimiter. For more information, see Regular expressions[508].
flags	Optional parameter. Provides the regular expression flags[510] to be used. For example, the flag "i" instructs the mapping process to operate in case-insensitive mode.

Example

The goal of the mapping illustrated below is to split the string `a ,   b c,d` into a sequence of strings, where each alphabetic character is an item in the sequence. Any redundant whitespace or commas must be removed.



To achieve this goal, the regular expression pattern `[ ,]+` was supplied as parameter to the `tokenize-regexp` function. This pattern has the following meaning:

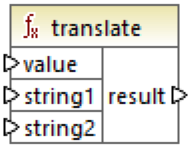
- It matches any of the characters inside the character class `[ ,]`. Therefore, a split will occur whenever a comma or a space is encountered in the input string.
- The quantifier `+` specifies that one or more occurrences of the preceding character class are to be matched. Without this quantifier, each occurrence of space or comma would create a separate item in the resulting sequence of strings, which is not the intended result.

The mapping output is as follows:

```
<items>
   <item>a</item>
   <item>b</item>
   <item>c</item>
   <item>d</item>
</items>
```

5.9.7.10.14 translate

Performs a character by character replacement. It looks in the **value** for characters contained in **string1**, and replaces each character with the one in the same position in the **string2**. When there are no corresponding characters in **string2**, the character is removed.



Languages

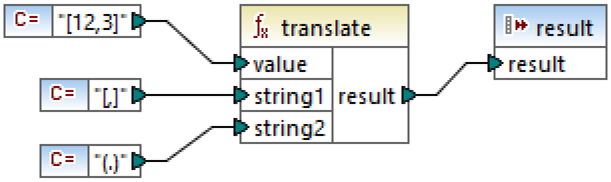
Built-in, C++, C#, Java, XQuery, XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Description
value	The input string.
string1	Provides a list of search characters. The position of each character inside the string is important.
string2	Provides a list of replacement characters. The position of each replacement character must correspond to the one in **string1**.

Example

Let's suppose you want to convert the string `[12,3]` to `(12.3)`. Namely, the square brackets must be replaced by round brackets, and any comma must be replaced by the dot character. To achieve this, you can call the `translate` function as follows:



In the mapping above, the first constant supplies the input string to be processed. The second and the third constant provide a list of characters as **string1** and **string2**, respectively.

string1	`[,]`
string2	`(.)`

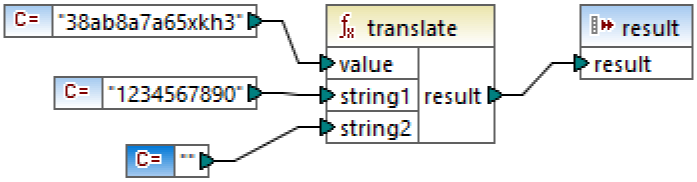
Notice that both **string1** and **string2** have the same number of characters. For each character in **string1**, the equivalent character at the same position from **string2** will be used as a replacement. Consequently, the following replacements will take place:

- Each `[` will be replaced by a `(`
- Each `,` will be replaced by a `.`
- Each `]` will be replaced by a `)`

The mapping output is as follows:

```
(12.3)
```

This function can also be used to strip certain characters selectively from a string. To achieve this, set the **string1** parameter to the characters you want to remove, and **string2** to an empty string. For example, the mapping below removes all digits from the string `38ab8a7a65xkh3`.



The mapping output is as follows:

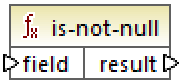
```
abaaxkh
```

5.9.7.11 db

The **db** library contains functions that allow you to define the mapping results when encountering null fields in databases.

5.9.7.11.1 is-not-null

Returns **false** if the field is null; **true** otherwise.



Languages

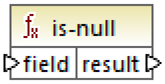
Built-in, C++, C#, Java.

Parameters

Name	Description
field	The database field.

5.9.7.11.2 is-null

Returns **true** if the field is null; **false** otherwise.



Languages

Built-in, C++, C#, Java.

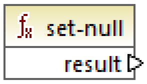
Parameters

Name	Description
field	The database field.

5.9.7.11.3 set-null

Sets a database field to null. This function will also overwrite a default value with null. If connected to something that is not a database field, it will behave like an empty sequence. Note the following:

- Connecting **set-null** to a different function will usually result in the other function not being called at all. Connecting **set-null** to a sequence function such as **count** will call the function with an empty sequence.
- Connecting **set-null** to filters and if-else conditions works as expected; fields are set to null. For filters, this means the "node/row" input.
- Using **set-null** as an input for a `simpleType` element will not create that element in the target component.

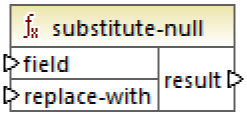


Languages

Built-in, C++, C#, Java.

5.9.7.11.4 substitute-null

Returns the field itself if it is not null; otherwise, **replace-with** is returned.



Languages

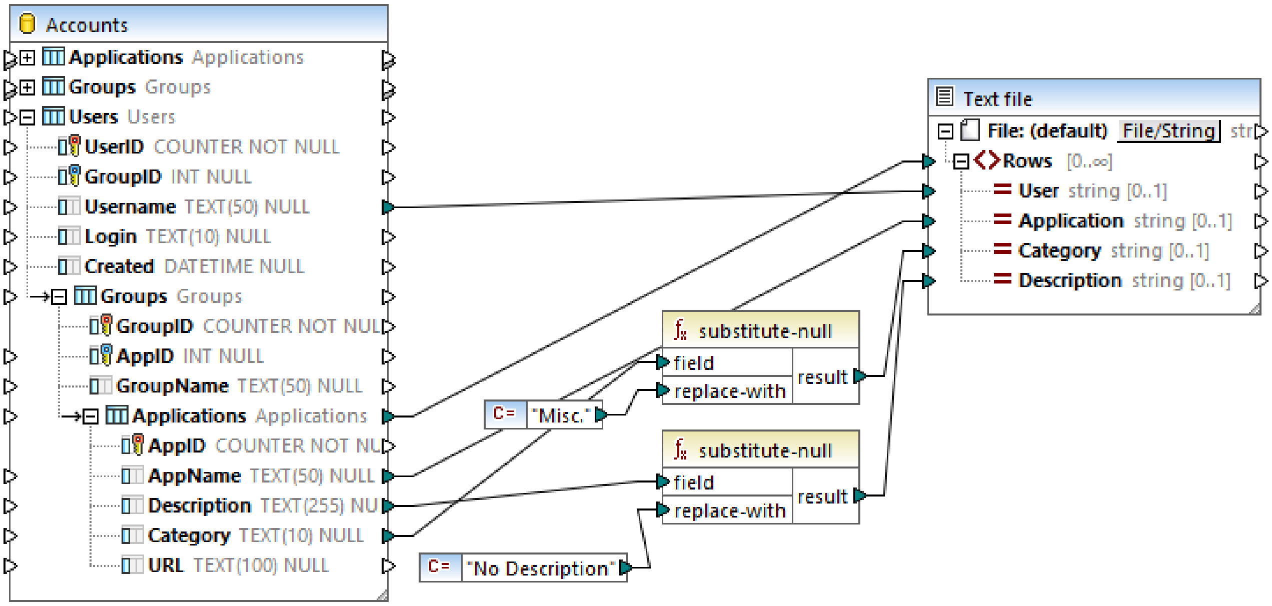
Built-in, C++, C#, Java.

Parameters

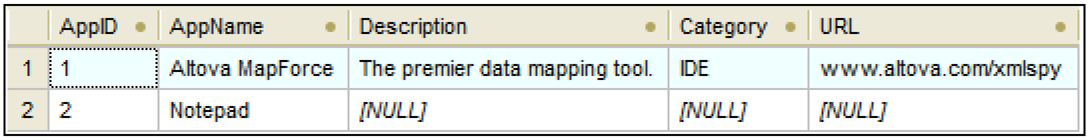
Name	Description
field	The database field.
replace-with	The replacement value.

Example

The mapping below shows an example of the **substitute-null** function in use. This mapping is called **DB-ApplicationList.mfd** and is available in the **<Documents>\Altova\MapForce2025\MapForceExamples\** folder.



The mapping reads data from an SQLite database which contains an "Applications" table.



	AppID	AppName	Description	Category	URL
1	1	Altova MapForce	The premier data mapping tool.	IDE	www.altova.com/xmlspy
2	2	Notepad	*[NULL]*	*[NULL]*	*[NULL]*

The first function checks if the **Category** field is null in the "Applications" table. Since this field is null for the Notepad application, the substitute value "Misc" is mapped to the **Category** item of the target text file.

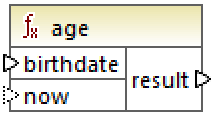
The second function checks if the **Description** field is null. Again, this field is null for the Notepad application, so the substitute value "No description" is mapped to the **Description** item of the target file.

5.9.7.12 lang | datetime functions

The date and time functions from the **lang** library can be used to manipulate dates, times, and durations. Unlike the date and time functions from the **core** library, these functions are available only when selecting Built-in, Java, C#, or C++ languages.

5.9.7.12.1 age

Returns the number of full years elapsed between the birth date supplied as argument and now.



Languages

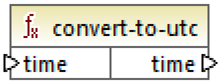
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
birthdate	`xs:date`	Mandatory. Provides the birth date as an `xs:date` value.
now	`xs:date`	Optional parameter. The default is the current system date. If a value is mapped to the now argument, the function returns the difference between the birth date and now, in full years.

5.9.7.12.2 convert-to-utc

Converts the time value supplied as argument to UTC (Coordinated Universal Time). The function takes the timezone component (for example, "+5:00") into account.



Languages

Built-in, C++, C#, Java.

Parameters

Name	Type	Description
time	`xs:dateTime`	Provides the `xs:dateTime` value to be converted.

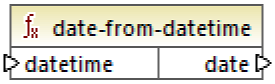
Example

If the input value is **`2001-12-17T09:30:02+05:00`**, the function's result is **`2001-12-17T04:30:02`**.

If the input value is **`2001-12-17T09:30:02Z`**, the function's result is **`2001-12-17T09:30:02`**. In this case, no conversion has taken place, because the trailing "Z" already defines this time to be "Zero" (or "Zulu") time, which is the same as UTC.

5.9.7.12.3 date-from-datetime

Returns the *date* part from the `xs:dateTime` value supplied as argument. The *time* part is set to zero. The timezone is not changed.



Languages

Built-in, C++, C#, Java.

Parameters

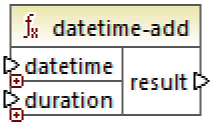
Name	Type	Description
datetime	**`xs:dateTime`**	Provides the `xs:dateTime` value to be processed.

Example

If the input value is **`2001-12-17T09:30:02+05:00`**, the function's result is **`2001-12-17+05:00`**.

5.9.7.12.4 datetime-add

Returns an `xs:dateTime` value obtained by adding a duration (the second argument) to a datetime (the first argument).



Languages

Built-in, C++, C#, Java.

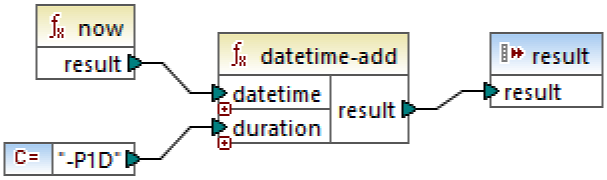
Parameters

Name	Type	Description
datetime	`xs:dateTime`	Provides the `xs:dateTime` value to be used as input.
duration	`xs:duration`	Provides the `xs:duration` value. An example duration is **`P1Y2M3DT04H05M59S`**, where: • "P" is the period designator, and is mandatory; • The rest of the characters denote, in this order: 1 Year, 2 Months, 3 Days, T (Time designator), 04 Hours, 05 Minutes, 59 Seconds. If the minus character appears before the "P" designator, this indicates a negative duration, for example: **`-P1D`**.

Example

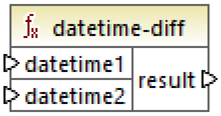
Let's assume that the input **datetime** value is **`2001-12-17T09:30:02+05:00`**. If the **duration** is **`P10D`** (10 days), the function's result is **`2001-12-27T09:30:02+05:00`**.

To obtain yesterday's date, connect the **now** function to the **datetime** input. In the mapping below, the period **`-P1D`** means "minus 1 day", so the mapping returns yesterday's date.



5.9.7.12.5 datetime-diff

Returns the duration obtained by subtracting **datetime2** (second argument) from **datetime1** (first argument). The result can be mapped to a string or duration data type.



Languages

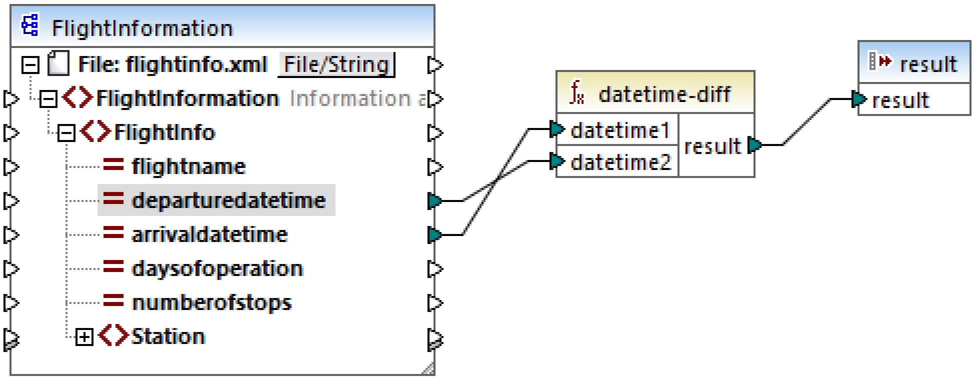
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
datetime1	`xs:dateTime`	Provides the first `xs:dateTime` value.
datetime2	`xs:duration`	Provides the second `xs:dateTime` value.

Example

In the mapping illustrated below, the **`datetime-diff`** function subtracts the flight departure datetime **`2001-12-17T09:30:02+05:00`** from the arrival datetime **`2001-12-17T19:30:02+05:00`**. Note that the arrival datetime is the greater value, so it is connected to the first input of the function.

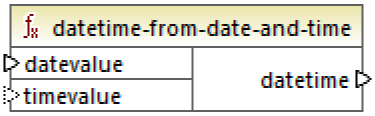


The mapping output is the difference between the two (a period of 10 hours):

```
PT10H
```

5.9.7.12.6 datetime-from-date-and-time

Returns an `xs:dateTime` value built from an `xs:date` value (first argument) and an `xs:time` value (second argument). The result can be mapped to a string or `xs:dateTime` data type.



Languages

Built-in, C++, C#, Java.

Parameters

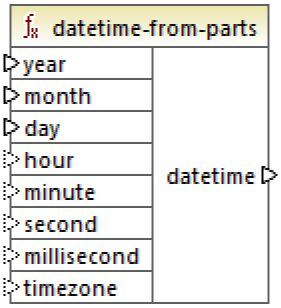
Name	Type	Description
datevalue	`xs:date`	Provides a value of type `xs:date`
timevalue	`xs:time`	Provides a value of type `xs:time`

Example

If the first argument is **`2012-06-29`** and the second argument is **`11:59:55`**, the function returns **`2012-06-29T11:59:55`**.

5.9.7.12.7 datetime-from-parts

Returns a value of type `xs:dateTime` built from any combination of the following parts as arguments: year, month, day, hour, minute, second, millisecond, and timezone. This function automatically normalizes the supplied parameters. For example, 32nd of January will automatically be changed to 1st of February.



Languages

Built-in, C++, C#, Java.

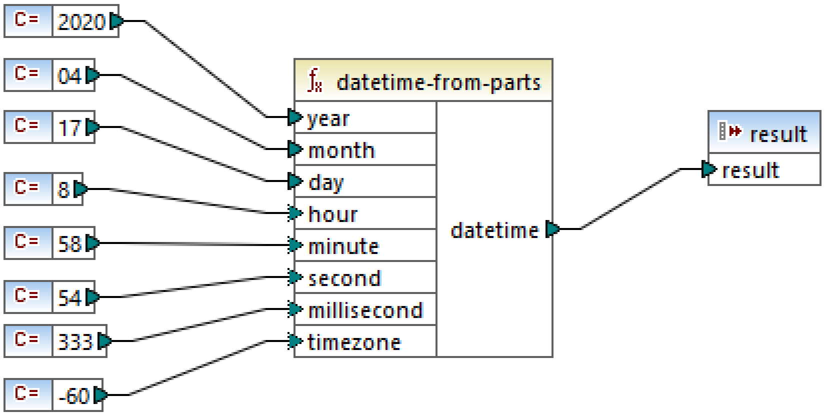
Parameters

Name	Type	Description
year	`xs:int`	Provides the year.
month	`xs:int`	Provides the month.
day	`xs:int`	Provides the day of the month.
hour	`xs:int`	Optional. Provides the hour.
minute	`xs:int`	Optional. Provides the minute.
second	`xs:int`	Optional. Provides the second.

Name	Type	Description
millisecond	`xs:decimal`	Optional. Provides the millisecond.
timezone	`xs:int`	Optional. Provides the timezone, in minutes. This value can be negative.

Example

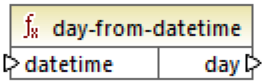
The following mapping constructs an `xs:dateTime` value from parts that are supplied by constants.



The mapping output is `2020-04-17T08:58:54.333-01:00`.

5.9.7.12.8 day-from-datetime

Returns the day, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

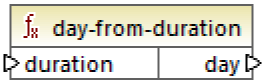
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T10:30:03+01:00`, then the function returns `17`.

5.9.7.12.9 day-from-duration

Returns the day, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

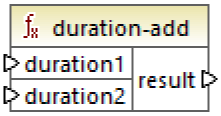
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is `P1Y2M3DT10H30M`, then the `day-from-duration` function returns `3`.

5.9.7.12.10 duration-add

Returns the duration obtained by adding two durations.



Languages

Built-in, C++, C#, Java.

Parameters

Name	Type	Description
duration1	`xs:duration`	Provides the first input value of type `xs:duration`.

Name	Type	Description
duration2	`xs:duration`	Provides the second input value of type `xs:duration`.

Example

If the first duration is **`P0Y0M3DT03H0M`** (3 days and 3 hours) and the second duration is **`P0Y0M3DT01H0M`** (3 days and 1 hour), then the function returns **`P6DT4H`** (6 days and 4 hours).

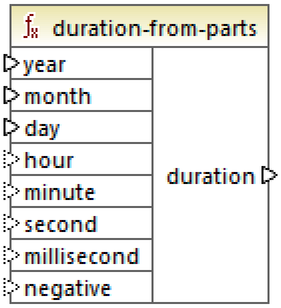
5.9.7.12.11 duration-from-parts

Returns a value of type `xs:duration` calculated by combining the following parts supplied as arguments: year, month, day, hour, minute, second, millisecond, negative.

An example duration is **`P1Y2M3DT04H05M59S`**, where:

- "P" is the period designator, and is mandatory;
- The rest of the characters denote, in this order: 1 Year, 2 Months, 3 Days, T (Time designator), 04 Hours, 05 Minutes, 59 Seconds.

If the minus character appears before the "P" designator, this indicates a negative duration, for example: **`-P1D`**.



Languages

Built-in, C++, C#, Java.

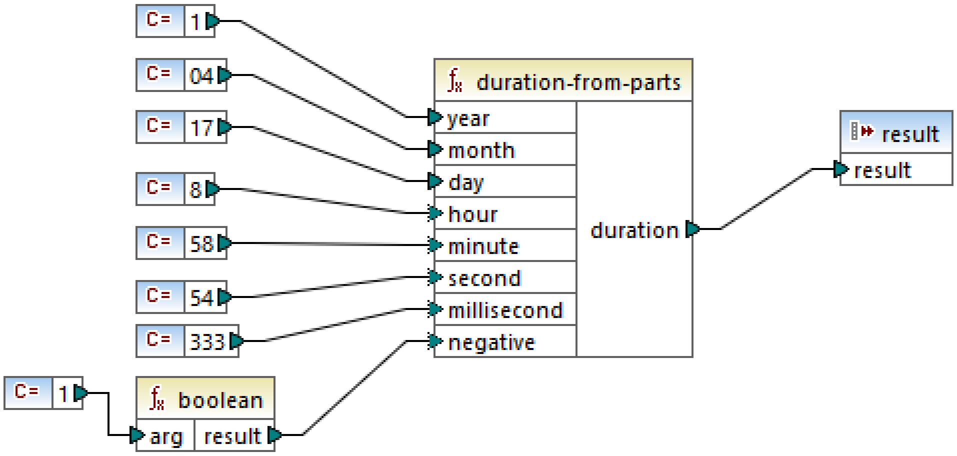
Parameters

Name	Type	Description
year	`xs:int`	Provides the year.
month	`xs:int`	Provides the month.
day	`xs:int`	Provides the day of the month.
hour	`xs:int`	Optional. Provides the hour.

Name	Type	Description
minute	`xs:int`	Optional. Provides the minute.
second	`xs:int`	Optional. Provides the second.
millisecond	`xs:decimal`	Optional. Provides the millisecond.
negative	`xs:boolean`	Optional. Must be **true** for a negative duration; **false** otherwise.

Example

The following mapping generates a negative duration of 1 year, 4 months, 17 days, 8 hours, 58 minutes, and 54.333 seconds.



The mapping output is **`-P1Y4M17DT8H58M54.333S`**.

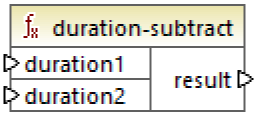
5.9.7.12.12 duration-subtract

Returns the `xs:duration` value obtained by subtracting **duration2** from **duration1**.

An example duration is **`P1Y2M3DT04H05M59S`**, where:

- "P" is the period designator, and is mandatory;
- The rest of the characters denote, in this order: 1 Year, 2 Months, 3 Days, T (Time designator), 04 Hours, 05 Minutes, 59 Seconds.

If the minus character appears before the "P" designator, this indicates a negative duration, for example: **`-P1D`**.



Languages

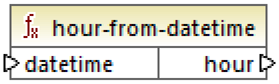
Built-in, C++, C#, Java.

Example

If **duration1** is `P0Y0M0DT05H07M` (5 hours and 7 minutes) and **duration2** is `PT1H` (1 hour), the function returns `PT4H7M` (4 hours and 7 minutes).

5.9.7.12.13 hour-from-datetime

Returns the hour, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

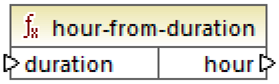
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T09:30:02+05:00`, then the function returns `9`.

5.9.7.12.14 hour-from-duration

Returns the hour, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

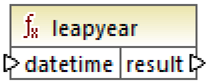
Name	Type	Description
duration	xs:duration	Provides the input value of type xs:duration.

Example

If **duration** is **P0Y0M0DT05H07M**, the function returns **5**.

5.9.7.12.15 leapyear

Returns Boolean **true** if the year of the xs:dateTime value supplied as argument is a leap year; **false** otherwise.



Languages

Built-in, C++, C#, Java.

Parameters

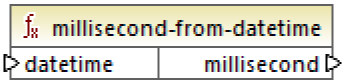
Name	Type	Description
datetime	xs:dateTime	Provides the input value of type xs:dateTime.

Example

If **datetime** is **2020-04-17T09:30:02+02:00**, the function returns **true**, since the year 2020 is a leap year.

5.9.7.12.16 millisecond-from-datetime

Returns the milliseconds, as an xs:decimal value, from the xs:dateTime value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

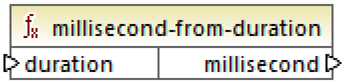
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T09:30:02.544+05:00`, the function returns `544`.

5.9.7.12.17 millisecond-from-duration

Returns the milliseconds, as an `xs:decimal` value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

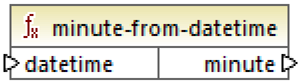
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is `P0Y0M0DT05H07M02.227S`, the function returns `227`.

5.9.7.12.18 minute-from-datetime

Returns the minutes, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

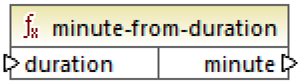
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T09:30:02.544+05:00`, the function returns `30`.

5.9.7.12.19 minute-from-duration

Returns the minutes, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

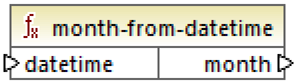
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is `P0Y0M0DT05H07M02.227S`, the function returns `7`.

5.9.7.12.20 month-from-datetime

Returns the month, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

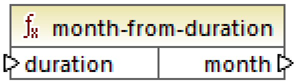
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T09:30:02.544+05:00`, the function returns `12`.

5.9.7.12.21 month-from-duration

Returns the month, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

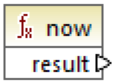
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is `P0Y04M0DT05H07M02.227S`, the function returns `4`.

5.9.7.12.22 now

Returns the current date and time (including timezone), as an `xs:dateTime` value.

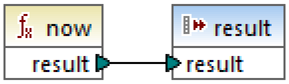


Languages

Built-in, C++, C#, Java.

Example

The following mapping outputs the current date and time. The output changes each time when the mapping runs.

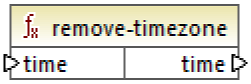


An example output would be `2020-04-17T11:42:34.684+02:00`.

For an example on how to extract yesterday's date, see the core | lang | datetime-add 610 function.

5.9.7.12.23 remove-timezone

Removes the timezone component from the **time** (of type `xs:dateTime`) input parameter.



Languages

Built-in, C++, C#, Java.

Parameters

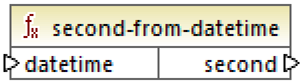
Name	Type	Description
time	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **time** is `2001-12-17T09:30:02+05:00`, the function returns `2001-12-17T09:30:02`.

5.9.7.12.24 second-from-datetime

Returns the seconds, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

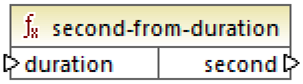
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T09:30:02.544+05:00`, the function returns `2`.

5.9.7.12.25 second-from-duration

Returns the seconds, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

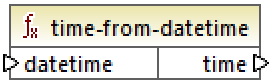
Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is `P0Y04M0DT05H07M02.227S`, the function returns `2`.

5.9.7.12.26 time-from-datetime

Returns the time component, as an `xs:time` value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

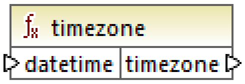
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T09:30:02+05:00`**, the function returns **`09:30:02+05:00`**.

5.9.7.12.27 timezone

Returns the timezone offset, in minutes, from the `xs:dateTime` value supplied as argument. Returns 0 for UTC.



Languages

Built-in, C++, C#, Java.

Parameters

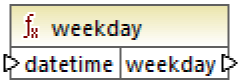
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is **`2001-12-17T09:30:02.544+05:00`**, the function returns **`300`**.

5.9.7.12.28 weekday

Returns the day of the week from the `xs:dateTime` value supplied as argument. The function will return value **1** for Monday, value **2** for Tuesday, and so on.



Languages

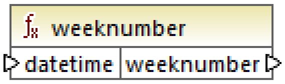
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

5.9.7.12.29 weeknumber

Returns the week number within the year from the `xs:dateTime` value supplied as argument. The function will return value **1** for the first week of the year, value **2** for the second week, and so on.



Languages

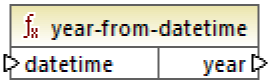
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

5.9.7.12.30 year-from-datetime

Returns the year, as an integer value, from the `xs:dateTime` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

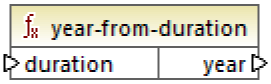
Name	Type	Description
datetime	`xs:dateTime`	Provides the input value of type `xs:dateTime`.

Example

If **datetime** is `2001-12-17T09:30:02.544+05:00`, the function returns `2001`.

5.9.7.12.31 year-from-duration

Returns the year, as an integer value, from the `xs:duration` value supplied as argument.



Languages

Built-in, C++, C#, Java.

Parameters

Name	Type	Description
duration	`xs:duration`	Provides the input value of type `xs:duration`.

Example

If **duration** is `P01Y04M0DT05H07M02.227S`, the function returns `1`.

5.9.7.13 lang | file functions

MapForce provides the ability to read BLOB (binary large object) data from binary files into a mapping, and then consume it without changing the internal structure of the binary data (raw). For example, you can save binary data to a database BLOB field, to a field of type `xs:base64Binary` in an XML file, or send it to a web service*.

** Web service calls are supported in MapForce Enterprise Edition only.*

You can also create mappings that read binary data from some source (such as a BLOB field in a database, a field of type `xs:base64Binary` in an XML file, or a web service) and then write binary files to the disk. The list below illustrates some of the possible scenarios in which binary files might be useful:

- Extract binary content encoded as base-64 data from an XML file and save it to the disk (for example, as a PDF file)
- Process image files stored on the disk and send them as base-64 encoded binary content to a web service
- Extract BLOB content from a database table and save it as image files to the disk (one image file for each row in the database table)
- Read image files from the disk and save them to a database table as BLOB data fields.

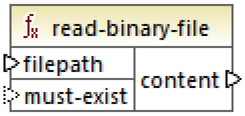
Note: Mapping data to or from binary files requires BUILT-IN 20 as a transformation language. You can preview the mapping in MapForce (and save the output files, if any) or choose to execute it with MapForce Server (licensed separately) on a different computer or platform. It is not supported to generate an executable C#, C++, or Java program from mappings that read or write binary files.

Read from and write to binary files
As such, there is no component kind associated with binary files in MapForce, like it is the case, for example, with XML, text, or JSON files. Instead, to help you accomplish goals such as the ones above, the following MapForce built-in functions are available:

- `read-binary-file` 628
- `write-binary-file` 630

5.9.7.13.1 read-binary-file

This function returns the content of the specified file as a BLOB (binary large object) of type `xs:base64Binary`. Note that even though the data type is called "base64Binary", the internal representation is just a BLOB. Only when you map the function's result to an XML node of type `xs:base64Binary` will it actually be base64-encoded. You could also map the function's result to `xs:hexBinary`, to a database blob, or to a binary field in a Protocol Buffers structure.



To read a binary file into a mapping, supply its path as input to the **filepath** argument. If the **filepath** is relative, then MapForce will look for the file in the same directory as the mapping. The **must-exist** argument is optional; if the file cannot be opened and this parameter is **true**, the mapping throws an error. If the file cannot be opened and this parameter is **false**, an empty binary is returned.

Languages

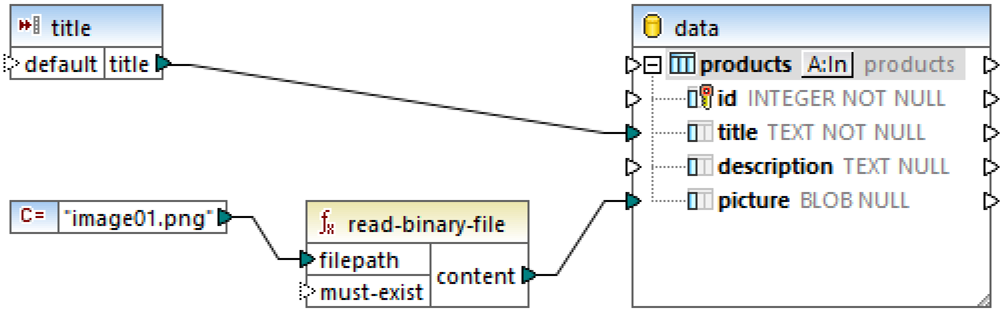
Built-in.

Parameters

Name	Type	Description
filepath	`xs:string`	The file path.
must-exist	`xs:boolean`	Optional parameter. If the file cannot be opened and this parameter is **true**, the mapping throws an error. If the file cannot be opened and this parameter is **false**, an empty binary is returned. The default value is **true**.

Example

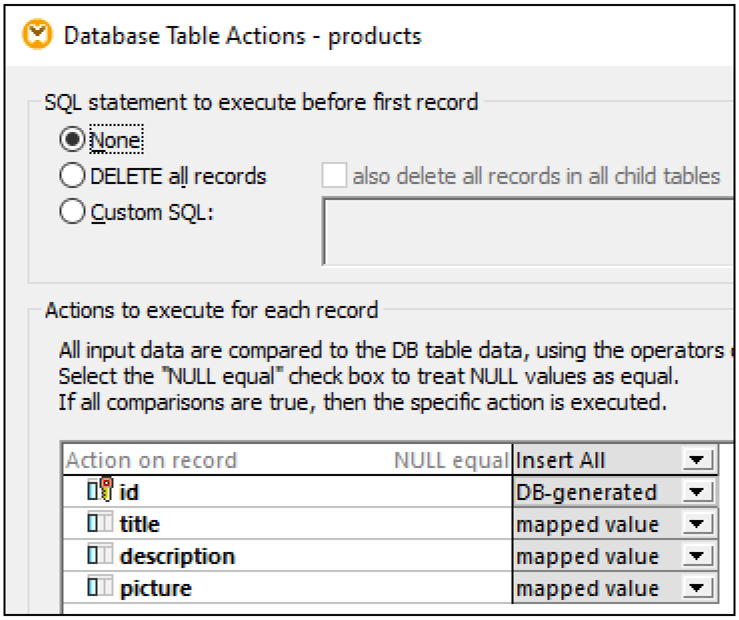
The mapping illustrated below reads data from an image file and writes it to a database table. The target database is SQLite. Notice that the data type of the **picture** database field is BLOB.



To extract binary content from the file, the `read-binary-file` function was used. In this example, the first argument, **filepath**, is supplied by a constant. Note that, because the path is relative, MapForce will look for the image file in the same directory as the mapping.

The mapping populates the following fields in the target database:

- **id** - In this example, the database component is configured so that **id** is database-generated rather than being supplied by the mapping. For more information, see Inserting Data into a Table[266].



Database Table Actions - products

SQL statement to execute before first record

● None

○ DELETE all records □ also delete all records in all child tables

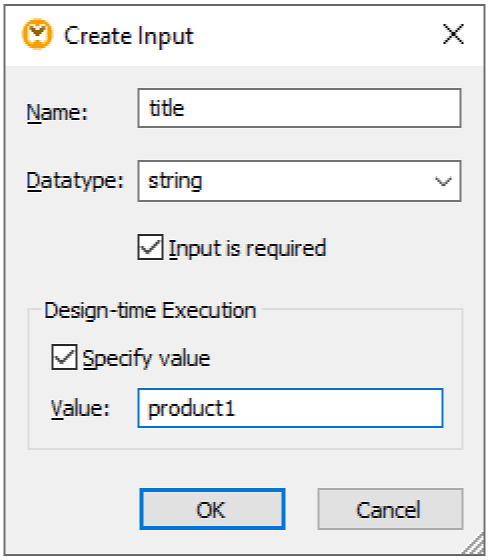
○ Custom SQL:

Actions to execute for each record

All input data are compared to the DB table data, using the operators
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
id		DB-generated
title		mapped value
description		mapped value
picture		mapped value

- **title** - This value is provided by a simple input component with the same name. Note that a design-time execution value is set ("product1") in order to make it possible to preview the mapping. For more information, see Supplying Parameters to the Mapping[346].


Create Input

Name: title

Datatype: string

☑ Input is required

Design-time Execution

☑ Specify value

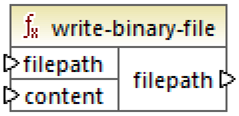
Value: product1

- **picture** - This field receives the direct output of the `read-binary-file` function.

Because the target component is a database, previewing the mapping generates a pseudo-SQL script that you can review, but does not send any changes to the database. To run the actual script against the database, select the menu command **Output | Run SQL-Script**.

5.9.7.13.2 write-binary-file

This function writes binary content to the specified file path and returns the path of the written file. If a binary file is the only desired output, connect the function's result to a simple output[357] component. Because this function writes a file whenever its output is used in the mapping, it is recommended to connect the function's result directly to a target component, without using other processing in between.



To write binary files, supply their path as input to the **filepath** argument. If **filepath** is relative, then MapForce will generate the file in the same directory as the mapping. The **content** argument must be connected to the actual binary content (for example, a BLOB field in a database).

When you preview the mapping in MapForce, the function generates temporary files by default, instead of writing files directly to the disk. To save the temporary files to disk, first click the **Output** pane, and then click the **Save generated output** or **Save all generated outputs** toolbar button, as applicable.

To configure MapForce to write output directly to final files instead of temporary, select the **Tools | Options** menu command, click **General**, and then select the **Write directly to final output files** option. Be aware that this option overwrites any existing files with the same name.

The function always returns the final (not temporary) file name, even when the final file is not saved to the disk yet (that is the case when you preview the mapping and the **Write directly to final output files** option is disabled).

> Note that it is not supported for a mapping to read back its own output file.

Languages

Built-In

Parameters

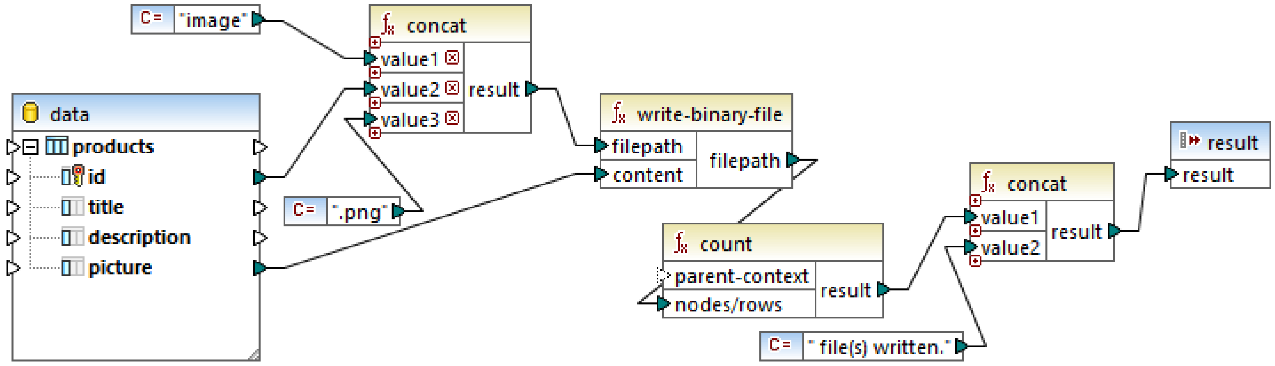
Name	Type	Description
filepath	`xs:string`	The input file path.
content	`xs:base64Binary`	The binary content of type `xs:base64Binary`.

Example

The mapping illustrated below reads BLOB values from a SQLite database and writes image files to the disk. The database has a table called **products**, which has the following columns (fields):

- **id** (*integer*, the unique permanent serial number of the record)
- **title** (*text*, the title of the product)
- **description** (*text*, the product's description)
- **picture** (*blob*, the product's image)

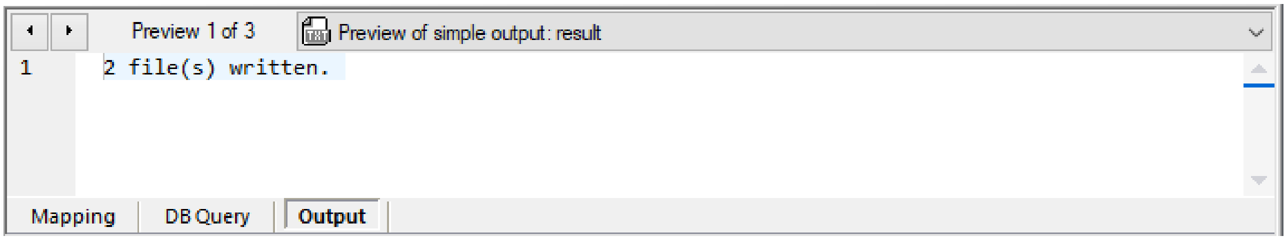
For the scope of this example, only the **id** and **picture** fields are relevant.



The goal of the mapping is to extract all pictures from the **products** table and write them as files to the disk. As illustrated above, the `write-binary-file` function is used for that purpose. The first argument, **filepath**, receives the file path for each image. The path must be unique to avoid overriding any files, so it is generated by concatenating the unique database **id** of each record with the word "image" and then adding the ".png" file extension (it is assumed that all pictures are in PNG format). Note that the path is relative, so the files will be written to the same directory as the mapping.

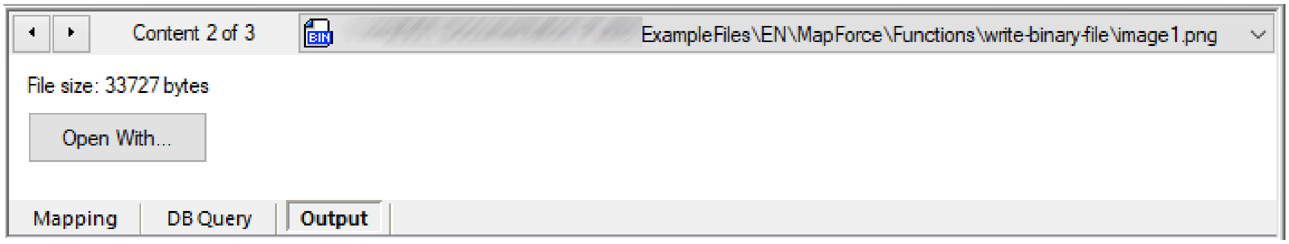
The second argument, **content**, receives the binary content stored in the database.

The `count` function returns a count of all generated files, and combines this number with the string "file(s) written". This provides a report as to how many files were actually generated by the mapping. In this example, the database contains only two product records, so the mapping output reflects this:



```
1    2 file(s) written.
```

As stated previously, this function generates temporary files when the mapping runs in preview execution. To preview each individual file, use the arrow buttons to navigate to the record of interest, click the **Open with** button, and select an image editor.


Content 2 of 3
ExampleFiles\EN\MapForce\Functions\write-binary-file\image1.png
Open With...
Mapping
DB Query
Output

File size: 33727 bytes

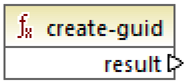
To save all files, click the **Save generated output** or **Save all generated outputs** toolbar button, as applicable.

5.9.7.14 lang | generator functions

The generator functions from the **lang** library are functions that generate values (currently, **`create-guid`** is the only such function).

5.9.7.14.1 create-guid

Creates a globally unique identifier (GUID), as a hex-encoded string. This function can be used to generate unique values, directly from the mapping, for database fields or other component types. See also the function format-guid-string 651.



Languages

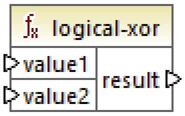
Built-in, C++, C#, Java.

5.9.7.15 lang | logical functions

Logical functions from the **lang** library include functions that evaluate miscellaneous value types using Boolean logic.

5.9.7.15.1 logical-xor

Returns **true** if **value1** is different from **value2**; **false** otherwise.



Languages

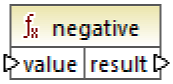
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value1	`xs:boolean`	The first input value.
value2	`xs:boolean`	The second input value.

5.9.7.15.2 negative

Returns **true** if the input value is negative (less than zero); **false** otherwise.



Languages

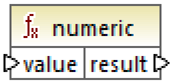
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:decimal`	The input value.

5.9.7.15.3 numeric

Returns **true** if the input value is a number or a string that can be parsed as a number; **false** otherwise.



Languages

Built-in, C++, C#, Java.

Parameters

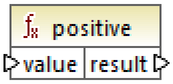
Name	Type	Description
value	`xs:decimal`	The input value.

Example

If the input value is the string `"4.33"`, the function returns **true**. If the input value is the string `"4.33 USD"`, the function returns **false**.

5.9.7.15.4 positive

Returns **true** if the input value is positive (equal to or greater than zero); **false** otherwise.



Languages

Built-in, C++, C#, Java.

Parameters

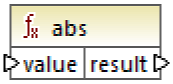
Name	Type	Description
value	`xs:decimal`	The input value.

5.9.7.16 lang | math functions

The math functions from the **lang** library can be used to perform various mathematical operations in the mapping.

5.9.7.16.1 abs

Returns the absolute value of the numeric value supplied as argument. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.



Languages

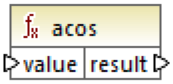
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:decimal`	The input value.

5.9.7.16.2 acos

Returns the arc cosine of **value**, in the range of `-pi/2` through `pi/2`.



Languages

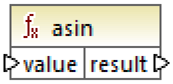
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.3 asin

Returns the arc sine of **value**, in the range of `-pi/2` through `pi/2`.



Languages

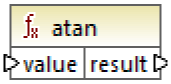
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.4 atan

Returns the arc tangent of **value**, in the range of `-pi/2` through `pi/2`.



Languages

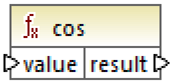
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.5 cos

Returns the trigonometric cosine of the angle given by **value**. The unit of value is radian.



Languages

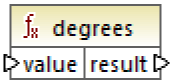
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.6 degrees

Converts an angle measured in radians to an approximately equivalent angle measured in degrees.



Languages

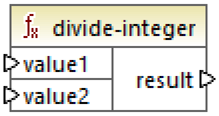
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.7 divide-integer

Returns the integer result of dividing **value1** by **value2**.



Languages

Built-in, C++, C#, Java.

Parameters

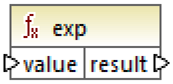
Name	Type	Description
value1	`xs:decimal`	The first input value.
value2	`xs:decimal`	The second input value.

Example

If the first value is **15** and the second value is **2**, the function returns **7**.

5.9.7.16.8 exp

Returns Euler's number **e** raised to the power of **value**.



Languages

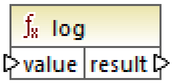
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.9 log

Returns the natural logarithm (base e) of **value**.



Languages

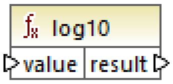
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.10 log10

Returns the decimal logarithm (base 10) of **value**.



Languages

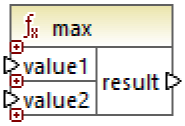
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.11 max

Returns the numeric value of the largest value supplied as argument. By default, this function has only two parameters, but you can add more. Click **Add parameter** (⊞) or **Delete parameter** (⊠) to add or remove parameters, see also Add or Delete Function Arguments 438.



Languages

Built-in, C++, C#, Java.

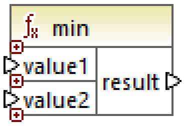
Parameters

Name	Type	Description
value1	`xs:decimal`	The first input value.

Name	Type	Description
value2	`xs:decimal`	The second input value.
valueN	`xs:decimal`	The *n*th input value.

5.9.7.16.12 min

Returns the numeric value of the smallest value supplied as argument. By default, this function has only two parameters, but you can add more. Click **Add parameter** (⊞) or **Delete parameter** (⊠) to add or remove parameters, see also Add or Delete Function Arguments 438.



Languages

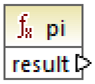
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value1	`xs:decimal`	The first input value.
value2	`xs:decimal`	The second input value.
valueN	`xs:decimal`	The *n*th input value.

5.9.7.16.13 pi

Returns the value of mathematical constant *pi*.

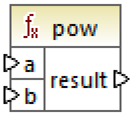


Languages

Built-in, C++, C#, Java.

5.9.7.16.14 pow

Returns the value of **a** raised to the power of **b**.



Languages

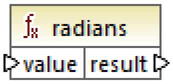
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
a	`xs:double`	Supplies value **a** (the base).
b	`xs:double`	Supplies value **b** (the power).

5.9.7.16.15 radians

Converts an angle measured in degrees to an approximately equivalent angle measured in radians.



Languages

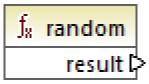
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.16 random

Returns a value with a positive sign, greater than or equal to 0.0 and less than 1.0. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

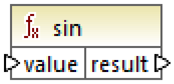


Languages

Built-in, C++, C#, Java.

5.9.7.16.17 sin

Returns the sine of **value**.



Languages

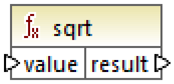
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.18 sqrt

Returns the correctly rounded positive square root of **value**.



Languages

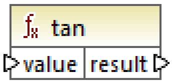
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.19 tan

Returns the tangent of **value**.



Languages

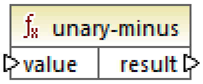
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.16.20 unary-minus

Returns the negation of the signed input value.



Languages

Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:decimal`	The input value.

Example

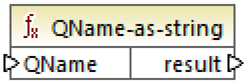
If the input value is **3**, the function returns **-3**. If the input value is **-3**, the function returns **3**.

5.9.7.17 lang | QName functions

The QName functions from the **lang** library convert Qualified Name (QName) values to strings, and vice versa. Unlike the functions from the **core** library, these functions are available only in the Built-in, Java, C#, or C++ languages.

5.9.7.17.1 QName-as-string

Returns the string representation of the QName value supplied as argument.



Languages

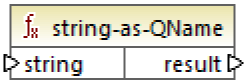
Built-in, C++, C#, Java.

Parameters

Name	Description
QName	The input `xs:QName` value.

5.9.7.17.2 string-as-QName

Converts the string representation of a QName back to a QName.



Languages

Built-in, C++, C#, Java.

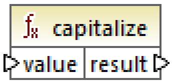
Name	Description
string	The input `string` value.

5.9.7.18 lang | string functions

The string functions from the `lang` library enable you to process strings (e.g., trim, pad, replace, convert strings to upper- or lower- case).

5.9.7.18.1 capitalize

Returns the input string **value,** where the first letter of each word is capitalized.



Languages

Built-in, C++, C#, Java.

Parameters

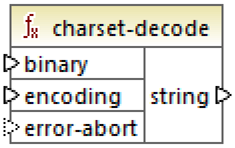
Name	Type	Description
value	`xs:string`	The input value.

Example

If the input value is `the quick brown fox`, the function returns `The Quick Brown Fox`.

5.9.7.18.2 charset-decode

The **`charset-decode`** function takes as input binary data encoded as Base64 text. It decodes data according to the specified character set (for example, "utf-8") and returns the resulting string value. If you need to encode binary data as Base64 text, use the charset-encode 647 function.



Languages

Built-in.

Parameters

Name	Type	Description
binary-data	`xs:base64Binary`	The binary data as Base64 text.
encoding	`xs:string`	The character set used for encoding (for example, "utf-8").

Name	Type	Description
error-abort	`xs:boolean`	Optional argument that specifies how processing should continue when errors are encountered. Valid values: - **true** - End processing with an exception on the invalid character. - **false** - Continue processing, and replace invalid characters with the replacement character �. The default value is **true**.

Example

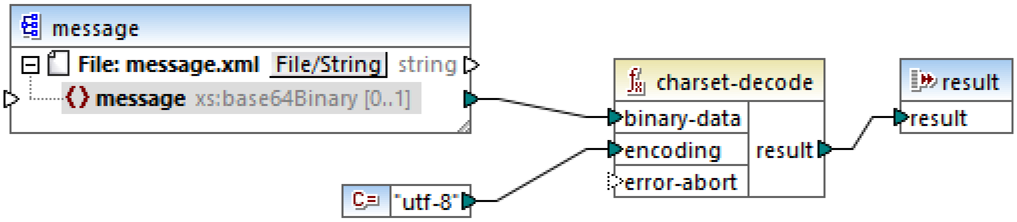
Let's suppose that you would like to decode binary data originating from the following source XML file. Notice that the **message** element contains binary data encoded as Base64 text.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<message xsi:noNamespaceSchemaLocation="message.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">TG9yZW0gaXBzdW0=</message>
```

The data type of the **message** element is `xs:base64Binary`, as illustrated by the schema:

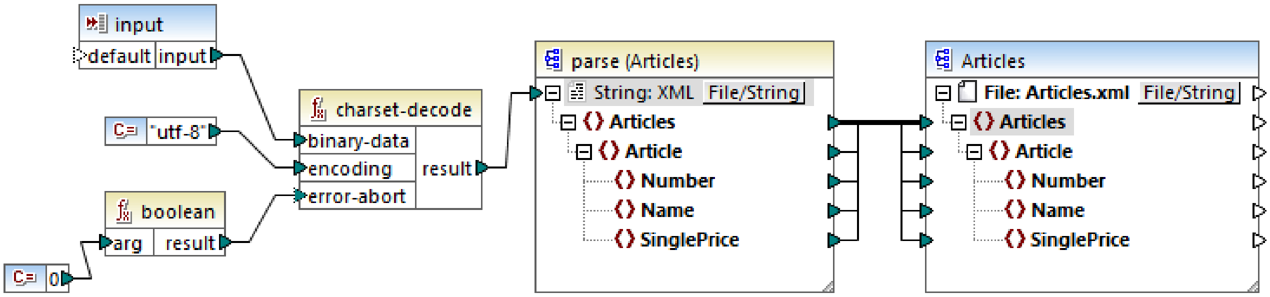
```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="message" type="xs:base64Binary"/>
</xs:schema>
```

A mapping that decodes the message above looks as follows:



The mapping in this example outputs the text "Lorem ipsum".

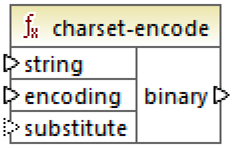
A mapping can also process text or XML files encoded as Base64 data, with the help of a MapForce serialization component. For example, the mapping illustrated below has an **input** parameter which expects Base64 text data. Assuming that the Base64 data was created from an XML file as shown in the charset-encode 647 example, you can recreate the original XML file as shown in the mapping below:



In this mapping, the **error-abort** argument gets a **false** value, which was produced with the help of the **boolean** built-in function. This ensures that processing will continue even if invalid characters are encountered. The string result of the function is then passed to an XML parsing component which converts it to an XML file. Note that, in order for XML parsing to be possible, you must have the XSD schema file. For more information, see Parsing and Serializing Strings 753.

5.9.7.18.3 charset-encode

The **charset-encode** function takes as input string data and encodes it as Base64 text. Data is encoded in the specified character set (for example, "utf-8") and returned as `xs:base64Binary` type. If you need to decode binary data previously encoded as Base64 text, use the charset-decode 645 function.



Languages

Built-in.

Parameters

Name	Type	Description
string-data	`xs:string`	The string data to be encoded.
encoding	`xs:string`	The character set used for encoding (for example, "utf-8").
substitute	`xs:string`	Optional argument that specifies a replacement character when invalid characters are encountered. This argument is applicable if you use a non-Unicode encoding. For

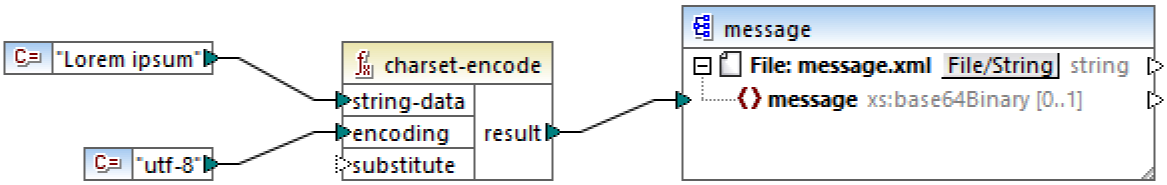
Name	Type	Description
		Unicode encodings, the replacement character is �.

Example

Let's suppose that you would like to encode the text "Lorem ipsum" as Base64 data, using the UTF-8 character set, and write it to a target XML file. The target XML file has a **message** element of `xs:base64Binary` type, as illustrated by the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="message" type="xs:base64Binary"/>
</xs:schema>
```

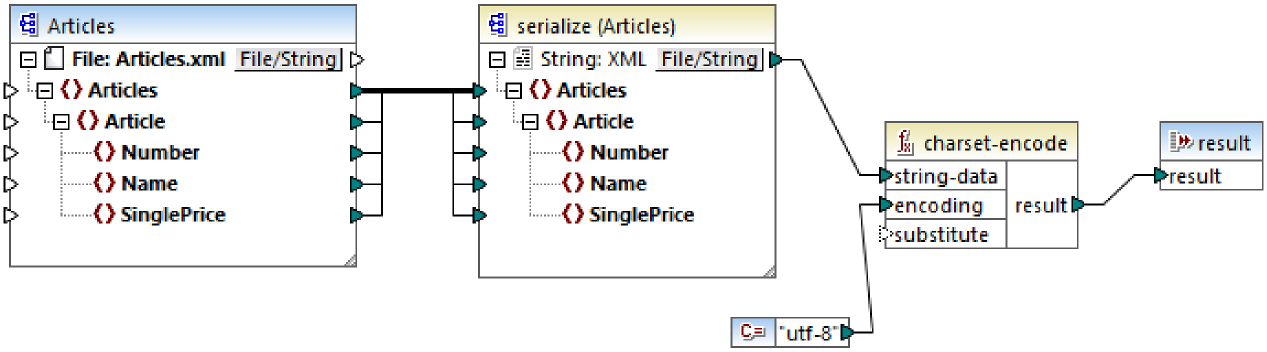
A mapping that performs the Base64 encoding looks as follows:



This mapping produces XML output like the one below (the schema references and XML declaration were skipped):

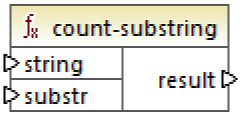
```
<message>TG9yZW0gaXBzdW0=</message>
```

You can also encode text or XML files as Base64, with the help of a MapForce serialization component. For example, the mapping illustrated below serializes a source XML file to a string. The resulting string is then supplied as argument to the **charset-encode** function. Finally, the function result is returned as mapping output, with the help of a simple output component, see Returning String Values from a Mapping[356]. For more information about serialization, see Parsing and Serializing Strings[753].



5.9.7.18.4 count-substring

Returns an integer value expressing the number of times that **substr** occurs in **string**.



Languages

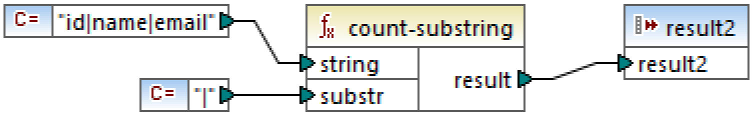
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	The input string.
substr	`xs:string`	The sub-string to test for.

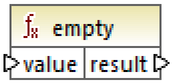
Example

The following mapping returns **2**. This is the number of times that the pipe separator occurs within the input string **`id|name|email`**.



5.9.7.18.5 empty

Returns **true** if the input string value is empty; **false** otherwise.



Languages

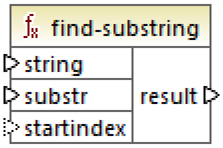
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
value	`xs:string`	The input value.

5.9.7.18.6 find-substring

Returns the position of the first occurrence of **substr** within **string**. By default, the function starts the search from the first character, which has position (index) 1, but you can optionally specify a specific starting index. If **substr** cannot be found, then the function returns **0**.



Languages

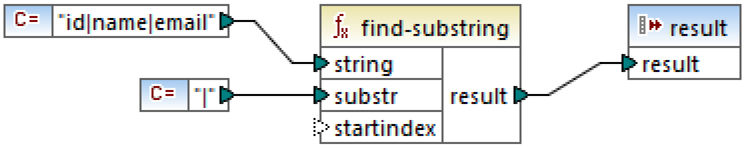
Built-in, C++, C#, Java.

Parameters

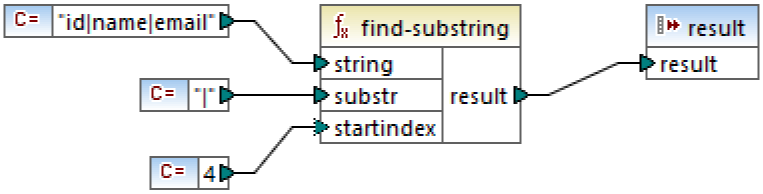
Name	Type	Description
string	`xs:string`	The input string.
substr	`xs:string`	The sub-string to search for.
startindex	`xs:int`	Optional. Specifies the starting position (index) of the search. If this parameter is not specified, the search starts at position 1.

Example

The following mapping outputs **3**, which is the position of the first occurrence of the pipe character in the input string **`id|name|email`**.

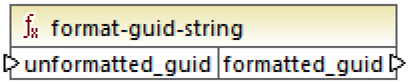


If you specify **4** as starting index, then the function starts searching from the fourth character. Consequently, the mapping below outputs **8**, which is the first occurrence of the pipe character after searching from the fourth character onwards.



5.9.7.18.7 format-guid-string

Returns a correctly formatted globally unique identifier (GUID) string, typically for use in database fields. See also the create-guid 632 function.



Languages

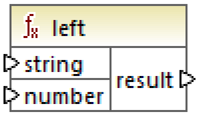
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
unformatted_guid	`xs:string`	The input HEX-encoded string to be formatted.

5.9.7.18.8 left

Returns a string containing the first **number** characters of the input string.



Languages

Built-in, C++, C#, Java.

Parameters

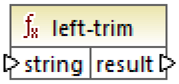
Name	Type	Description
string	`xs:string`	The input string.
number	`xs:int`	Specifies how many characters to return, starting from the beginning of the string.

Example

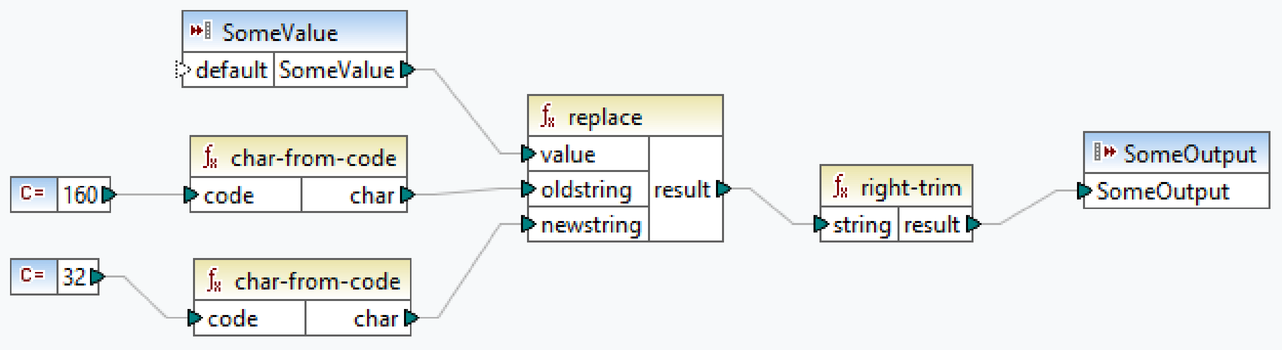
If the input string is `This is a sentence` and number is **4**, the function returns `This`.

5.9.7.18.9 left-trim

The **`left-trim`** function (*see screenshot below*) removes leading whitespace characters of a string. Whitespace includes space (U+0020), tab (U+0009), carriage return (U+000D), and line feed (U+000A) characters. For details about whitespaces, see the [XML Recommendation].



About non-breaking spaces
The **`left-trim`**, **`right-trim`**, and **`normalize-space`** functions do not remove non-breaking spaces. One of the possible solutions could be to replace the non-breaking space character, whose decimal representation is 160, with the space character, whose decimal representation is 32. The mapping below shows that after the non-breaking space has been replaced, the trimmed `SomeValue` value will be mapped to the target.



If your source component is an Excel file, you can remove extra spaces in Excel using a combination of TRIM, CLEAN, and SUBSTITUTE functions. For details, see *[Removing Spaces and Nonprinting Characters from Text]*.

Languages

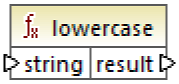
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	The input string.

5.9.7.18.10 lowercase

Converts the input **string** to lowercase. For Unicode characters, the corresponding lower-case characters (defined by the Unicode consortium) are used.



Languages

Built-in, C++, C#, Java.

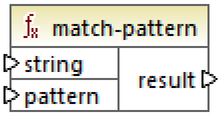
Parameters

Name	Type	Description
string	`xs:string`	The input string.

5.9.7.18.11 match-pattern

Returns Boolean **true** if the input string matches the regular expression defined by **pattern**; **false** otherwise. See also Regular expressions 508.

Note: When generating C++, C#, or Java code, the advanced features of the regular expression syntax might differ slightly. See the regex documentation of each language for more information.



Languages

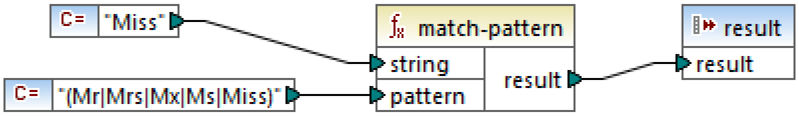
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	xs:string	The input string.
pattern	xs:string	The regular expression to match.

Example

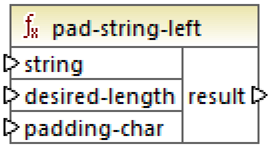
The following mapping validates various person titles. Specifically, the mapping will output **true** for any of the following titles: Mr, Mrs, Mx, Ms, Miss.



If the input string is other than any of the titles listed above, the mapping outputs **false**.

5.9.7.18.12 pad-string-left

Returns a string which is padded to the left by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.



Languages

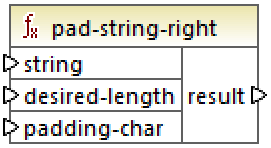
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	xs:string	Specifies the input string.
desired-length	xs:int	Defines the desired length of the string after padding.
padding-char	xs:string	Defines the character to use as padding character.

5.9.7.18.13 pad-string-right

Returns a string which is padded to the right by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.



Languages

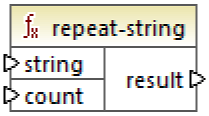
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	Specifies the input string.
desired-length	`xs:int`	Defines the desired length of the string after padding.
padding-char	`xs:string`	Defines the character to use as padding character.

5.9.7.18.14 repeat-string

Repeats the string supplied as argument *n* times. The **count** argument specifies the number of times to repeat the string.



Languages

Built-in, C++, C#, Java.

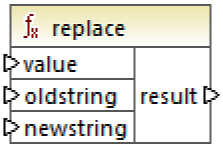
Parameters

Name	Type	Description
string	`xs:string`	The input string.

Name	Type	Description
count	xs:int	The number of times to repeat the string.

5.9.7.18.15 replace

Result is a new string where each instance of **oldstring**, in the input string **value**, is replaced by **newstring**.



Languages

Built-in, C++, C#, Java.

Parameters

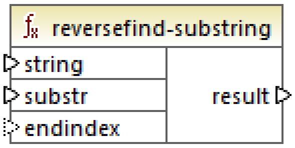
Name	Type	Description
value	xs:string	The input value.
oldstring	xs:string	The old string to be replaced.
newstring	xs:string	The new string to act as replacement.

Example

See Replacing Special Characters 589.

5.9.7.18.16 reversefind-substring

Returns the position of the last occurrence of **substr** within **string**. By default, the function starts the search from the first character, which has position (index) 1, and ends the search at the last character, but you can optionally specify an ending index. If **substr** cannot be found, then the function returns **0**.



Languages

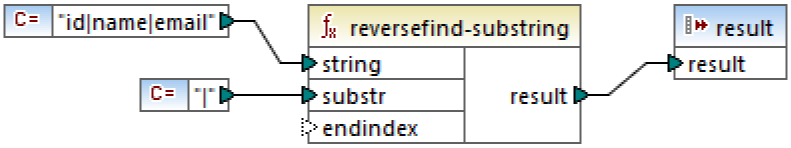
Built-in, C++, C#, Java.

Parameters

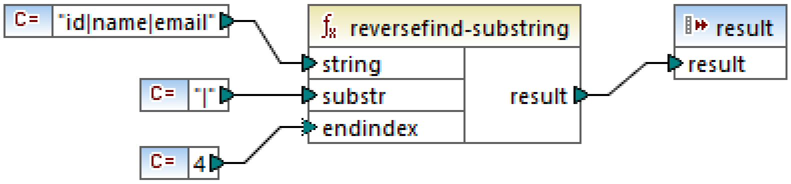
Name	Type	Description
string	`xs:string`	The input string.
substr	`xs:string`	The sub-string to search for.
endindex	`xs:int`	Optional. Specifies the ending position (index) of the search. If this parameter is not specified, the search ends after the last character in **string**.

Example

The following mapping outputs **8**, which is the position of the last occurrence of the pipe character in the input string `id|name|email`.

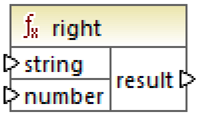


If you specify **4** as ending index, then the function searches up to the fourth character. Consequently, the mapping below outputs **3**.



5.9.7.18.17 right

Returns a string containing the last **number** characters of the input string.



Languages

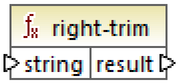
Built-in, C++, C#, Java.

Example

If the input string is `The brown red fox` and number is **3**, the function returns `fox`.

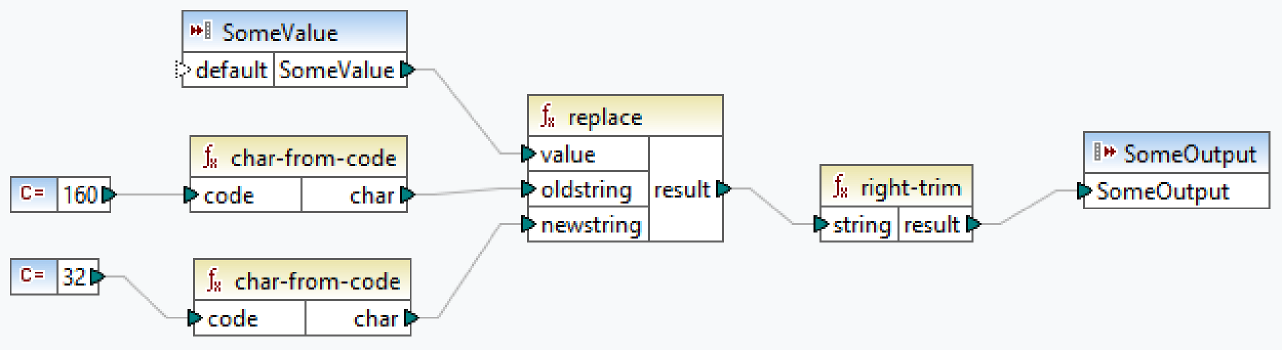
5.9.7.18.18 right-trim

The **`right-trim`** function (*see screenshot below*) removes trailing whitespace characters of a string. Whitespace includes space (U+0020), tab (U+0009), carriage return (U+000D), and line feed (U+000A) characters. For details about whitespaces, see the [XML Recommendation](#).



About non-breaking spaces

The **`left-trim`**, **`right-trim`**, and **`normalize-space`** functions do not remove non-breaking spaces. One of the possible solutions could be to replace the non-breaking space character, whose decimal representation is 160, with the space character, whose decimal representation is 32. The mapping below shows that after the non-breaking space has been replaced, the trimmed `SomeValue` value will be mapped to the target.



If your source component is an Excel file, you can remove extra spaces in Excel using a combination of TRIM, CLEAN, and SUBSTITUTE functions. For details, see *[Removing Spaces and Nonprinting Characters from Text](#)*.

Languages

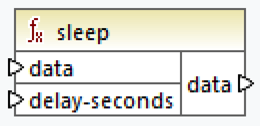
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string	`xs:string`	The input string.

5.9.7.18.19 sleep

The **sleep** function (*screenshot below*) delays the transmission of data for N seconds. Passing a sequence through the function will delay each item of the sequence for a specified time. The **sleep** function is compatible with the following transformation languages: Java, C#, C++, and Built-In. Code generation is supported in Java, C#, and C++. For more information about code generation, see Code Generator 868.



Parameters

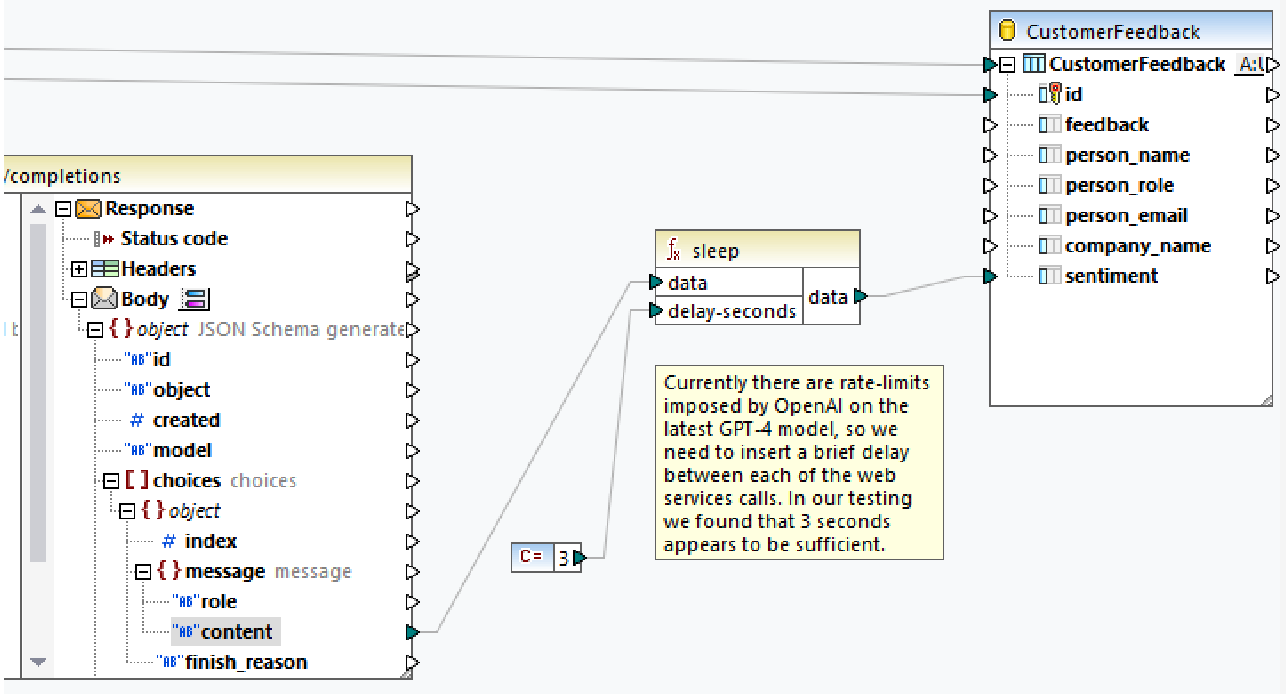
Name	**Type**	**Description**
`data`	`any node or atomic type`	The `data` input parameter accepts any value (e.g., `string`).
`delay-seconds`	`xs:double`	The `delay-seconds` input parameter delays the transmission of data for N seconds. Fractional seconds are also acceptable.
`data`	`any node or atomic type`	The `data` output parameter receives data from the input and passes this data on to a target node.

Example

For a possible use-case scenario, in which the **sleep** function is used, see the following mapping: **`MapForceExamples\SentimentAnalysis.mfd`**. An extract of this mapping is illustrated below. To be able to test the mapping, you will need your organization's login credentials.

Since OpenAI imposes rate limits on the API requests you can make, you may run into a `Too Many Requests` error. The **sleep** function enables you to bypass the rate limits by configuring a delay.

In the response structure of the web service call below, the `content` node receives data as a result of the request sent to the OpenAI API. Before each web service call, there is a delay for 3 seconds, and then the value of the `content` node is mapped to the `sentiment` column of the `CustomerFeedback` database.



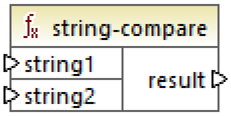
Currently there are rate-limits imposed by OpenAI on the latest GPT-4 model, so we need to insert a brief delay between each of the web services calls. In our testing we found that 3 seconds appears to be sufficient.

For more information about this example and AI functionality in MapForce, see the following articles:

- *Data Integration with AI*
- *AI-Based Support Request Sentiment Analysis Using MapForce and GPT-4*
- *AI-Based Database Image Classification with Altova MapForce*

5.9.7.18.20 string-compare

The **`string-compare`** function (*see screenshot below*) returns the result of a character by character comparison of two input strings: `string1` and `string2`. The comparison is based on the ASCII codes. Both `string1` and `string2` are of type `xs:string`. The function is case-sensitive. If the strings are equal, the result is 0. If `string1` is less than `string2`, the result is -1. If `string1` is greater than `string2`, the result is 1.



Example:

```
string1: hi
string2: Hit
```

The **`string-compare`** function compares the strings character by character. The comparison stops after the function has detected that the first character of `string1` and the first character of `string2` are different. The result is based on the comparison of the first character of each string. Since `h` is represented as a bigger ASCII code number (104 in the decimal system) than `H` (72 in the decimal system), `string1` is greater than `string2`, and the result of the string comparison is 1. If the first character of `string1` and the first character of `string2` were the same, the function would proceed to analyze the second character and so on.

For simple string comparison with a boolean result, see `core | logical functions | equal`543.

Languages

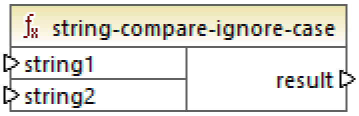
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
`string1`	`xs:string`	The first input string.
`string2`	`xs:string`	The second input string.

5.9.7.18.21 string-compare-ignore-case

The **`string-compare-ignore-case`** function (*see screenshot below*) returns the result of a character by character comparison of two input strings: `string1` and `string2`. Both `string1` and `string2` are of type `xs:string`. The function ignores case. The comparison is based on the ASCII codes. If the strings are equal, the result is 0. If `string1` is less than `string2`, the result is -1. If `string1` is greater than `string2`, the result is 1.



Example:

```
string1: hi
string2: Hit
```

The **`string-compare-ignore-case`** function compares the strings character by character. Even though `h` is represented as a bigger ASCII code number than `H`, these two characters are treated as equal in this function. The second character in both strings is the same. However, `string2` has a third character, whereas `string1` does not. The third character in `string1` has an empty value. The `t` value in `string2` is greater than the empty value in `string1`. Therefore, `string1` is less than `string2`, and the result equals -1.

Languages

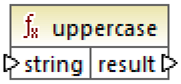
Built-in, C++, C#, Java.

Parameters

Name	Type	Description
string1	xs:string	The first input string.
string2	xs:string	The second input string.

5.9.7.18.22 uppercase

Converts the input **string** to uppercase. For Unicode characters, the corresponding upper-case characters (defined by the Unicode consortium) are used.



Languages

Built-in, C++, C#, Java.

Parameters

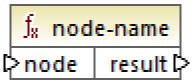
Name	Type	Description
string	**xs:string**	The input string.

5.9.7.19 xpath2 | accessors

Functions from the **xpath2 | accessors** sub-library retrieve information about XML nodes or items. These functions are available when either the XSLT2 or XQuery languages are selected.

5.9.7.19.1 base-uri

The **base-uri** function takes a node as input, and returns the URI of the XML resource containing the node. The output is of type xs:string.



Languages

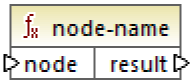
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
node	`mf:node`	The input node.

5.9.7.19.2 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`** 560 function.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

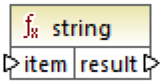
Parameters

Name	Type	Description
node	`mf:node`	The input node.

5.9.7.19.3 string

The **`string`** function works like the `xs:string` constructor: it converts its argument to `xs:string`.

When the input argument is a value of an atomic type (for example `xs:decimal`), this atomic value is converted to a value of `xs:string` type. If the input argument is a node, the string value of the node is extracted. (The string value of a node is a concatenation of the values of the node's descendant nodes.)



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
item	`mf:item`	The input value.

5.9.7.20 xpath2 | anyURI functions

The **xpath2 | anyURI** sub-library contains the **`resolve-uri`** function. This function is available when either the XSLT2 or XQuery languages are selected.

5.9.7.20.1 resolve-uri

The **`resolve-uri`** function takes a relative URI as its first argument and resolves it against the base URI in the second argument. The result is of data type `xs:string`. The function's implementation treats both inputs as strings; no checks are performed as to whether the resources identified by these URIs actually exist.

Languages

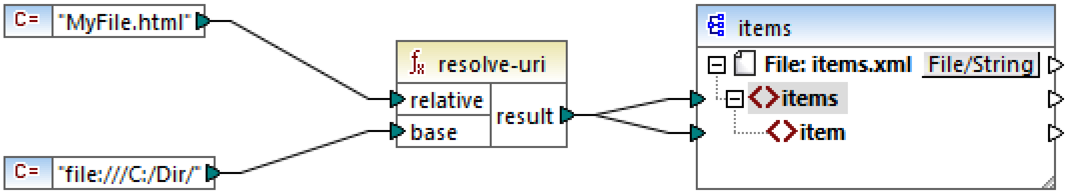
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
relative	`xs:string`	The relative URI to be resolved against the base.
base	`xs:string`	The base URI.

Example

In the mapping illustrated below, the first argument provides the relative URI **`MyFile.html`**, and the second argument provides the base URI **`file:///C:/Dir/`**. The resolved URI will be a concatenation of both, so **`file:///C:/Dir/MyFile.html`**.

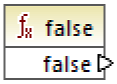


5.9.7.21 xpath2 | boolean functions

The Boolean functions `true` and `false` take no argument and return the boolean constant values **true** and **false**, respectively. They can be used where a constant boolean value is required.

5.9.7.21.1 false

Returns the Boolean value **false**.

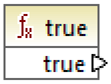


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.21.2 true

Returns the Boolean value **true**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.22 xpath2 | constructors

The functions in the "constructors" sub-library of the XPath 2.0 library construct specific data types from the input text. The following table lists the available constructor functions.

`xs:ENTITY`	`xs:double`	`xs:nonPositiveInteger`
`xs:ID`	`xs:duration`	`xs:normalizedString`
`xs:IDREF`	`xs:float`	`xs:positiveInteger`
`xs:NCName`	`xs:gDay`	`xs:short`
`xs:NMTOKEN`	`xs:gMonth`	`xs:string`
`xs:Name`	`xs:gMonthDay`	`xs:time`

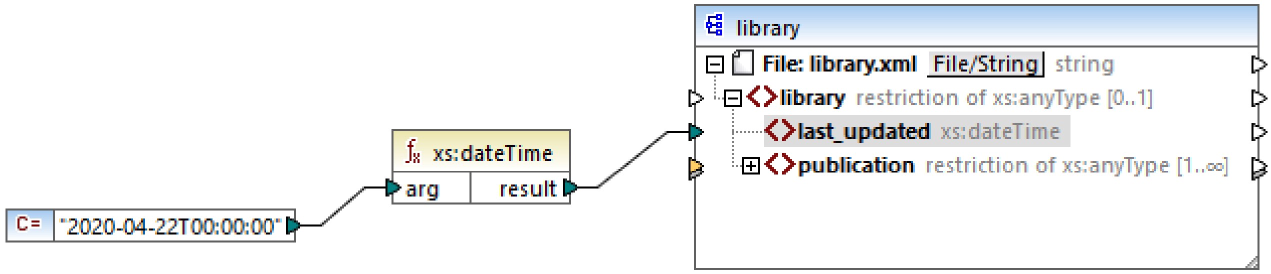
xs:QName	**xs:gYear**	**xs:token**
xs:anyURI	**xs:gYearMonth**	**xs:unsignedByte**
xs:base64Binary	**xs:hexBinary**	**xs:unsignedInt**
xs:boolean	**xs:int**	**xs:unsignedLong**
xs:byte	**xs:integer**	**xs:unsignedShort**
xs:date	**xs:language**	**xs:untypedAtomic**
xs:dateTime	**xs:long**	**xs:yearMonthDuration**
xs:dayTimeDuration	**xs:negativeInteger**	
xs:decimal	**xs:nonNegativeInteger**	

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Example

Typically, the lexical format of the input text must be the one expected of the data type to be constructed. Otherwise, the transformation will not be successful. For example, to construct an `xs:dateTime` value using the **`xs:dateTime`** constructor function, the input text must have the lexical format of the `xs:dateTime` data type, which is **`YYYY-MM-DDTHH:mm:ss`**.



In the mapping illustrated above, a string constant (**`"2020-04-28T00:00:00"`**) has been used to provide the input argument of the function. The input could also have been obtained from an item in the source document. The **`xs:dateTime`** function returns the value **`2020-04-28T00:00:00`** of type `xs:dateTime`.

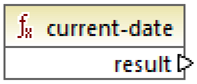
To view the expected data type of a mapping item (including the data type of function arguments), move the mouse cursor over the respective input or output connector.

5.9.7.23 xpath2 | context functions

The context functions from the **xpath2** library provide miscellaneous information about the current date and time, the default collation used by the processor, the size of the current sequence, and the position of the current node.

5.9.7.23.1 current-date

Returns the current date (`xs:date`) from the system clock.

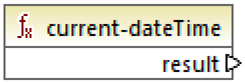


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.23.2 current-dateTime

Returns the current date and time (`xs:dateTime`) from the system clock.

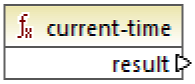


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.23.3 current-time

Returns the current time (`xs:time`) from the system clock.



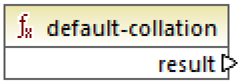
Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.23.4 default-collation

The **`default-collation`** function takes no argument and returns the default collation, that is, the collation that is used when no collation is specified for a function where one can be specified.

Comparisons, including for the **`max-string`** and **`min-string`** functions, are based on the default collation.

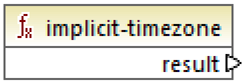


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.23.5 implicit-timezone

Returns the value of the "implicit timezone" property from the evaluation context.

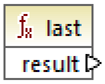


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.23.6 last

Returns the number of items in the sequence of items currently being processed. Importantly, the sequence of items is determined by the current mapping context[79], as described in the example below.



Languages

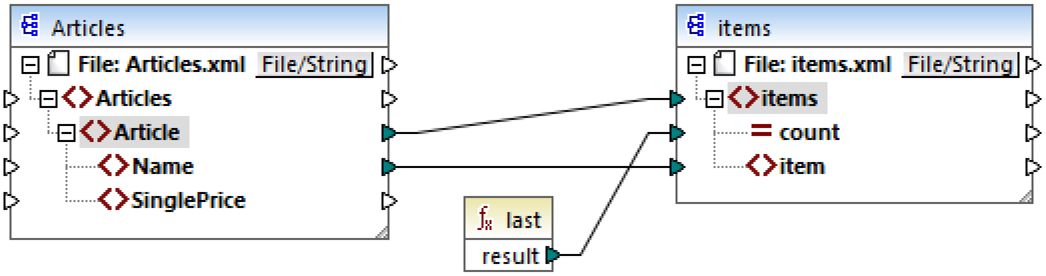
XQuery, XSLT 2.0, XSLT 3.0.

Example

Let's suppose that you have the following source XML file:

```
<Articles>
   <Article>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>
   <Article>
      <Name>Socks</Name>
      <SinglePrice>2.30</SinglePrice>
   </Article>
   <Article>
      <Name>Jacket</Name>
      <SinglePrice>57.50</SinglePrice>
   </Article>
</Articles>
```

Your goal is to copy data to an XML file with a different schema. Also, the count of all items must be saved to the target XML file. This can be achieved by a mapping like the one below:



In the example above, the `last` function returns the position of the last node in the current parent context and populates the **count** attribute with value **3**.

```
<items count="3">
   <item>T-Shirt</item>
   <item>Pants</item>
   <item>Jacket</item>
</items>
```

Note that value **3** is the position of the last item (and thus the count of all items) in the mapping context created by the connection between **Article** and **items**. If this connection did not exist, items would still be copied to the target, but the `last` function would return value **1** incorrectly, because it would have no parent context 82 to iterate over. (More precisely, it would use the default implicit context created between the root items of both components, which produces a sequence of 1 item, not 3 as expected).

It is generally advisable to use the `count` 516 function from the **core** library instead of the `last` function, because the former has a **parent-context** argument, which enables you to alter the mapping context explicitly.

5.9.7.24 xpath2 | durations, date and time functions

The duration, date and time functions from the **xpath2** library enable you to adjust the time zone in date and time values, extract particular components from date, time, and duration values, and subtract date and time values.

Adjusting the time zone

To adjust the time zone in date and time values, the following functions are available:

- **adjust-date-to-timezone**

- **adjust-date-to-timezone** (with timezone argument)
- **adjust-dateTime-to-timezone**
- **adjust-dateTime-to-timezone** (with timezone argument)
- **adjust-time-to-timezone**
- **adjust-time-to-timezone** (with timezone argument)

Each of these related functions takes an `xs:date`, `xs:time`, or `xs:dateTime` value as the first argument and adjusts the input by adding, removing, or modifying the time zone component depending on the value of the second argument (if one is present).

The following situations are possible when the first argument contains no time zone (for example, the date `2020-01` or the time `14:00:00`).

- If the **timezone** argument is present, the result will contain the time zone specified in the second argument. The time zone in the second argument is added.
- If the **timezone** argument is absent, the result will contain the implicit timezone, which is the system's time zone. The system's time zone is added.
- If the **timezone** argument is empty, the result will contain no time zone.

The following situations are possible when the first argument contains a time zone (for example, the date `2020-01-01+01:00` or the time `14:00:00+01:00`).

- If the **timezone** argument is present, the result will contain the time zone specified in the second argument. The original time zone is replaced by the timezone in the second argument.
- If the **timezone** argument is absent, the result will contain the implicit time zone, which is the system's time zone. The original time zone is replaced by the system's time zone.
- If the **timezone** argument is empty, the result will contain no time zone.

Extracting components of dates and times

To extract numeric values such as hours, minutes, days, months, and so on from date and time values, the following functions are available:

- **day-from-date**
- **day-from-dateTime**
- **hours-from-dateTime**
- **hours-from-time**
- **minutes-from-dateTime**
- **minutes-from-time**
- **month-from-date**
- **month-from-dateTime**
- **seconds-from-dateTime**
- **seconds-from-time**
- **timezone-from-date**
- **timezone-from-dateTime**
- **timezone-from-time**
- **year-from-date**
- **year-from-dateTime**

Each of these functions extracts a particular component from `xs:date`, `xs:time`, `xs:dateTime`, and `xs:duration` values. The result will be either `xs:integer` or `xs:decimal`.

Extracting components of durations

To extract time components from durations, the following functions are available:

- **days-from-duration**
- **hours-from-duration**
- **minutes-from-duration**
- **months-from-duration**
- **seconds-from-duration**
- **years-from-duration**

The duration must be specified either as `xs:yearMonthDuration` (for extracting years and months) or `xs:dayTimeDuration` (for extracting days, hours, minutes, and seconds). All functions returns a result of type `xs:integer`, with the exception of the **seconds-from-duration** function, which returns `xs:decimal`.

Subtracting date and time values

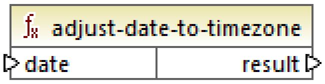
To subtract date and time values, the following functions are available:

- **subtract-dateTimes**
- **subtract-dates**
- **subtract-times**

Each of the subtraction functions enables you to subtract one time value from another and return a duration value.

5.9.7.24.1 adjust-date-to-timezone

Adjusts an `xs:date` value to the implicit time zone in the evaluation context (the system's time zone).



Languages

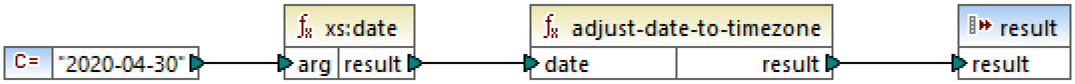
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**xs:date**	The input value of type `xs:date`.

Example

The following mapping constructs an `xs:date` from a string and supplies it as argument to the **adjust-date-to-timezone** function.

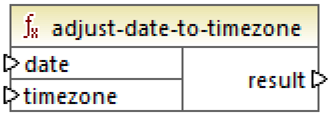


XSLT 2.0 mapping

If the mapping runs on a computer where the system time zone is +02:00, the function adjusts the date value to include the system's time zone. Consequently, the mapping output is **`2020-04-30+02:00`**.

5.9.7.24.2 adjust-date-to-timezone

Adjusts an `xs:date` value to a specific time zone, or to no time zone at all. If the **timezone** argument is an empty sequence, the function returns an `xs:date` without a time zone. Otherwise, it returns an `xs:date` with a time zone.



Languages

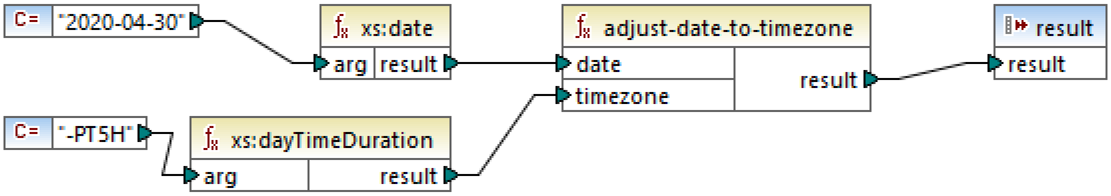
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**`xs:date`**	The input value of type `xs:date`.
timezone	**`xs:dayTimeDuration`**	The time zone expressed as an `xs:dayTimeDuration` value. The value can be negative. For example, a time zone value of -5 hours can be expressed as **`-PT5H`**.

Example

The following mapping constructs both parameters to the **`adjust-date-to-timezone`** function from strings, using the corresponding XPath 2 constructor 665 functions. The goal of the mapping is to adjust the time zone to -5 hours. This time zone can be expressed as **`-PT5H`**.

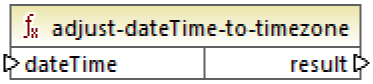


XSLT 2.0 mapping

The function adjusts the date value to the time zone supplied as argument. Consequently, the mapping output is **`2020-04-30-05:00`**.

5.9.7.24.3 adjust-dateTime-to-timezone

Adjusts an `xs:dateTime` value to the implicit time zone in the evaluation context (the system's time zone).



Languages

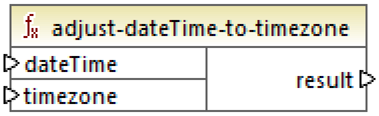
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**`xs:dateTime`**	The input value of type `xs:dateTime`.

5.9.7.24.4 adjust-dateTime-to-timezone

Adjusts an `xs:dateTime` value to a specific time zone, or to no time zone at all. If the **timezone** argument is an empty sequence, the function returns an `xs:dateTime` without a time zone. Otherwise, it returns an `xs:dateTime` with a time zone.



Languages

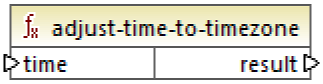
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**`xs:dateTime`**	The input value of type `xs:dateTime`.
timezone	**`xs:dayTimeDuration`**	The time zone expressed as an `xs:dayTimeDuration` value. The value can be negative. For example, a time zone value of -5 hours can be expressed as **`-PT5H`**.

5.9.7.24.5 adjust-time-to-timezone

Adjusts an `xs:time` value to the implicit time zone in the evaluation context (the system's time zone).



Languages

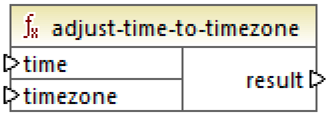
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	**`xs:time`**	The input value of type `xs:time`.

5.9.7.24.6 adjust-time-to-timezone

Adjusts an `xs:time` value to a specific time zone, or to no time zone at all. If the **timezone** argument is an empty sequence, the function returns an `xs:time` without a time zone. Otherwise, it returns an `xs:time` with a time zone.



Languages

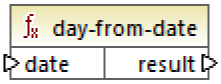
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	**xs:time**	The input value of type xs:time.
timezone	**xs:dayTimeDuration**	The time zone expressed as an xs:dayTimeDuration value. The value can be negative. For example, a time zone value of -5 hours can be expressed as **-PT5H**.

5.9.7.24.7 day-from-date

Returns an xs:integer representing the day part of the xs:date value supplied as argument.



Languages

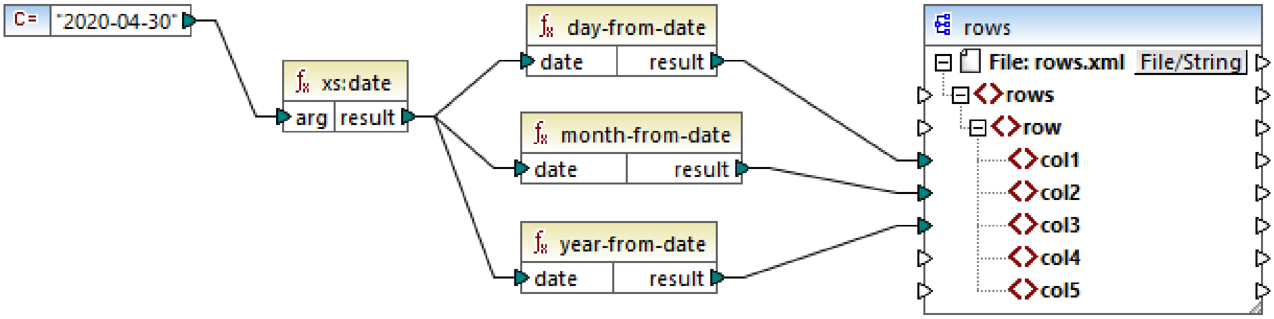
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**xs:date**	The input value of type xs:date.

Example

The following mapping converts a string to xs:date using the **xs:date** constructor function. The **day-from-date**, **month-from-date**, and **year-from-date** functions each extract the respective part of the date and write it to a separate item in the target XML file.



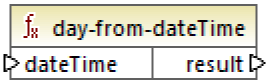
XQuery 1.0 mapping

The mapping output is as follows:

```
<rows>
   <row>
      <col1>30</col1>
      <col2>4</col2>
      <col3>2020</col3>
   </row>
</rows>
```

5.9.7.24.8 day-from-dateTime

Returns an `xs:integer` representing the day part of the `xs:dateTime` value supplied as argument.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	`xs:dateTime`	The input value of type `xs:dateTime`.

5.9.7.24.9 days-from-duration

Returns an `xs:integer` representing the "days" component of the canonical representation of the duration value supplied as argument.

Languages

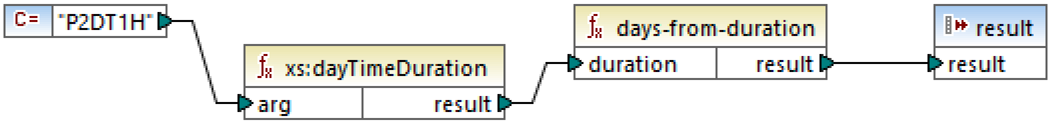
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	`xs:duration`	The input value of type `xs:duration`.

Example

The mapping illustrated below constructs the `xs:dayTimeDuration` of **`P2DT1H`** (2 days and 1 hours) and supplies it as input to the **`days-from-duration`** function. The result is **2**.



XSLT 2.0 mapping

Note: If the duration is **`P1DT24H`** (1 day and 24 hours), the function returns **2**, not **1**. This is because the canonical representation of **`P1DT24H`** is actually **`P2D`** (2 days).

5.9.7.24.10 hours-from-dateTime

Returns an `xs:integer` representing the hours part of the `xs:dateTime` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	`xs:dateTime`	The input value of type `xs:dateTime`.

5.9.7.24.11 hours-from-duration

Returns an `xs:integer` representing the hours component of the canonical representation of the duration value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	`xs:duration`	The input value of type `xs:duration`.

Example

If the duration is **`PT1H60M`** (1 hour and 60 minutes), the function returns **2**, not **1**. This is because the canonical representation of **`PT1H60M`** is actually **`PT2H`** (2 hours).

5.9.7.24.12 hours-from-time

Returns an `xs:integer` representing the hours part of the `xs:time` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	`xs:time`	The input value of type `xs:time`.

5.9.7.24.13 minutes-from-dateTime

Returns an `xs:integer` representing the minutes part of the `xs:dateTime` supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	`xs:dateTime`	The input value of type `xs:dateTime`.

5.9.7.24.14 minutes-from-duration

Returns an `xs:integer` representing the minutes component of the canonical representation of the duration supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	**`xs:duration`**	The input value of type `xs:duration`.

Example

If the duration is **`PT1M60S`** (1 minute and 60 seconds), the function returns **2**, not **1**. This is because the canonical representation of **`PT1M60S`** is actually **`PT2M`** (2 minutes).

5.9.7.24.15 minutes-from-time

Returns an `xs:integer` representing the minutes part of the `xs:time` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	**`xs:time`**	The input value of type `xs:time`.

5.9.7.24.16 month-from-date

Returns an `xs:integer` representing the month part of the `xs:date` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**xs:date**	The input value of type `xs:date`.

5.9.7.24.17 month-from-dateTime

Returns an `xs:integer` representing the month part of the `xs:dateTime` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**xs:dateTime**	The input value of type `xs:dateTime`.

5.9.7.24.18 months-from-duration

Returns an `xs:integer` representing the months component in the canonical representation of the duration value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	**xs:duration**	The input value of type `xs:duration`.

5.9.7.24.19 seconds-from-dateTime

Returns an xs:integer representing the seconds component in the localized value of dateTime.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	`xs:dateTime`	

5.9.7.24.20 seconds-from-duration

Returns an `xs:integer` representing the seconds component in the canonical representation of the duration value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	`xs:duration`	The input value of type `xs:duration`.

5.9.7.24.21 seconds-from-time

Returns an `xs:integer` representing the seconds part of the `xs:time` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	`xs:time`	The input value of type `xs:time`.

5.9.7.24.22 subtract-dateTimes

Returns the `xs:dayTimeDuration` that corresponds to the difference between the normalized value of **dateTime1** and the normalized value of **dateTime2**.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime1	`xs:dateTime`	The first input value.
dateTime2	`xs:dateTime`	The second input value.

5.9.7.24.23 subtract-dates

Returns the `xs:dayTimeDuration` that corresponds to the difference between the normalized value of **date1** and the normalized value of **date2**.

Languages

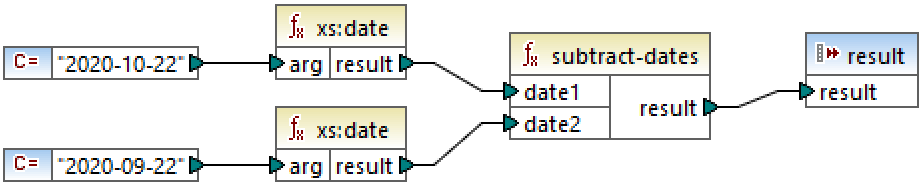
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date1	`xs:date`	The first input value.
date2	`xs:date`	The second input value.

Example

The mapping illustrated below subtracts two dates (2020-10-22 minus 2020-09-22). The result is the value **`P30D`** of type `xs:dayTimeDuration`., which represents a duration of 30 days.



5.9.7.24.24 subtract-times

Returns the `xs:dayTimeDuration` that corresponds to the difference between the normalized value of **time1** and the normalized value of **time2**.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time1	**xs:time**	The first input value.
time2	**xs:time**	The second input value.

5.9.7.24.25 timezone-from-date

Returns the timezone component of the date supplied as argument. The result is an `xs:dayTimeDuration` that indicates deviation from UTC; its value may range from +14:00 to -14:00 hours, both inclusive.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	**xs:date**	The input value of type `xs:date`.

5.9.7.24.26 timezone-from-dateTime

Returns the timezone component of the `xs:dateTime` value supplied as argument. The result is an `xs:dayTimeDuration` that indicates deviation from UTC; its value may range from +14:00 to -14:00 hours, both inclusive.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	**xs:dateTime**	The input value of type `xs:dateTime`.

5.9.7.24.27 timezone-from-time

Returns the timezone component of the `xs:time` value supplied as argument. The result is an `xs:dayTimeDuration` that indicates deviation from UTC; its value may range from +14:00 to -14:00 hours, both inclusive.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
time	`xs:time`	The input value of type `xs:time`.

5.9.7.24.28 year-from-date

Returns an `xs:integer` representing the year part of the `xs:date` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
date	`xs:date`	The input value of type `xs:date`.

5.9.7.24.29 year-from-dateTime

Returns an `xs:integer` representing the year part of the `xs:dateTime` value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
dateTime	`xs:dateTime`	The input value of type `xs:dateTime`.

5.9.7.24.30 years-from-duration

Returns an `xs:integer` representing the years component in the canonical lexical representation of the duration value supplied as argument.

Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
duration	`xs:duration`	The input value of type `xs:duration`.

5.9.7.25 xpath2 | node functions

The node functions from the **`xpath2`** library provide information about nodes (items) on a mapping component.

The **`lang`** function takes a string argument that identifies a language code (such as "en"). The function returns **true** or **false** depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function.

The **`local-name`**, **`name`**, and **`namespace-uri`** functions, return, respectively, the local name, name, and namespace URI of the input node. For example, for the node **altova:Products**, the local name is **`Products`**, the name is **`altova:Products`**, and the namespace URI is the URI of the namespace to which the altova: prefix is bound (see the example given for the **`local-name`**[687] function). Each of these three functions has two variants:

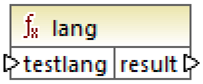
- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the **`lang`**[685] function).
- With an argument that must be a node: the function is applied to the connected node.

The **`number`** function takes a node as input, atomizes the node (that is, extracts its contents), and converts the value to a decimal and returns the converted value. There are two variants of the **`number`** function:

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the **`lang`**[685] function).
- With an argument that must be a node: the function is applied to the connected node.

5.9.7.25.1 lang

Returns **true** if the context node has an `xml:lang` attribute with a value that either matches exactly the **testlang** argument, or is a subset of it. Otherwise, the function returns **false**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

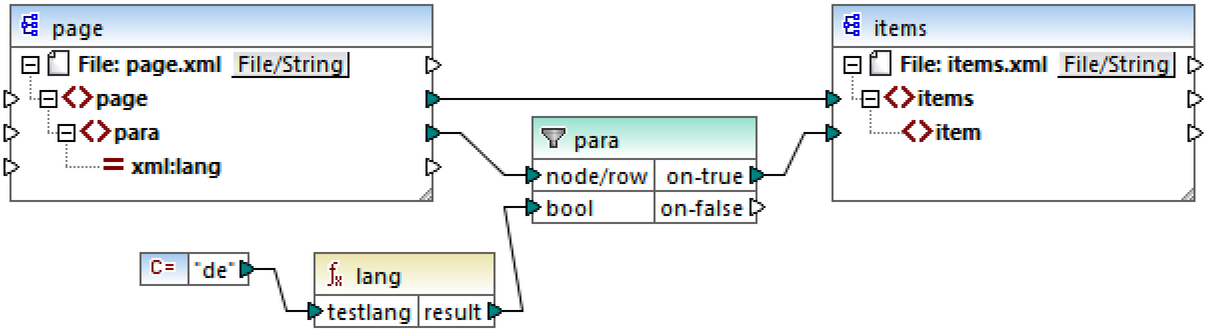
Name	Type	Description
testlang	`xs:string`	The language code to check, for example, "en".

Example

The following XML contains **para** elements with different values for the `xml:lang` attribute.

```
<page>
   <para xml:lang="en">Good day!</para>
   <para xml:lang="fr">Bonjour!</para>
   <para xml:lang="de-AT">Grüss Gott!</para>
   <para xml:lang="de-DE">Guten Tag!</para>
   <para xml:lang="de-CH">Grüezi!</para>
</page>
```

The mapping illustrated below filters only the German paragraphs, regardless of the country variant, with the help of the `lang` function.



XSLT 2.0 mapping

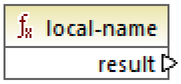
In the mapping above, for each **para** in the source, an **item** is created in the target, conditionally. The condition is provided by a filter which passes on to the target only those nodes where the `lang` function returns **true**. That is, only those nodes that have the `xml:lang` attribute set to "de" (or a subset of "de") will satisfy the filter's condition. Consequently, the mapping output is as follows:

```
<items>
   <item>Grüss Gott!</item>
   <item>Guten Tag!</item>
   <item>Grüezi!</item>
</items>
```

Note that the **lang** function operates in the context of each **para**, because of the parent connection between **para** and **item**, see also The Mapping Context[79].

5.9.7.25.2 local-name

Returns the local part of the name of the context node as an `xs:string`. This is a parameterless variant of the **local-name** function where the context node is determined by the connections in your mapping. To specify a node explicitly, use the **local-name**[687] function that takes an input node as parameter.

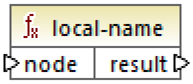


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.25.3 local-name

Returns the local part of the name of **node** as an `xs:string`.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
node	**node()**	The input node.

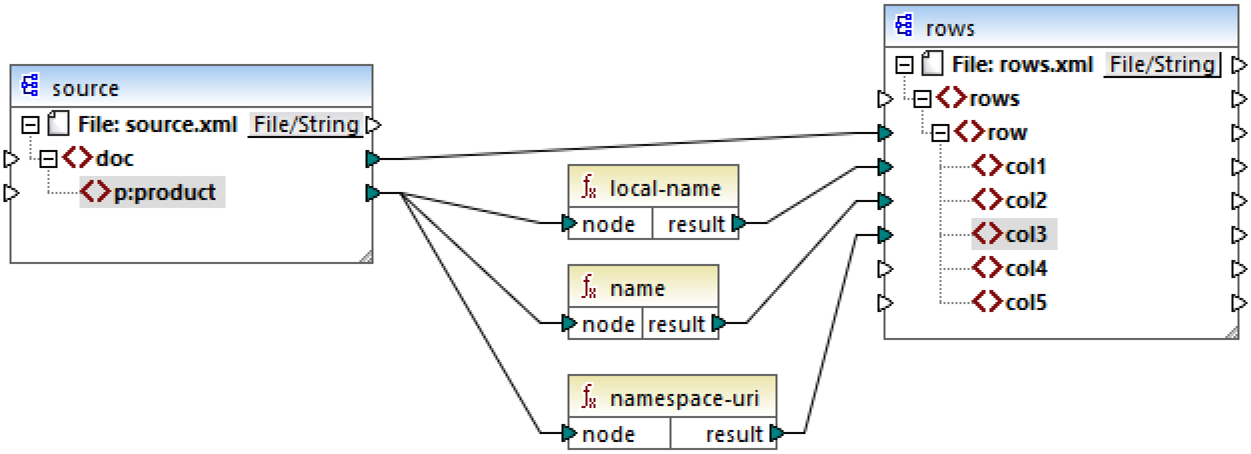
Example

In the following XML file, the name of the `p:product` element is a prefixed qualified name (QName). The prefix "p" is mapped to the namespace "http://mycompany.com".

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:p="http://mycompany.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation="source.xsd">
  <p:product/>
</doc>
```

The following mapping extracts the local name, the name, and the namespace URI of the node and writes these values to a target file:



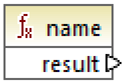
XSLT 2.0 mapping

The mapping output is displayed below. Each **col** item lists the result of the **local-name**, **name**, and **namespace-uri** functions, respectively.

```
<rows>
   <row>
      <col1>product</col1>
      <col2>p:product</col2>
      <col3>http://mycompany.com</col3>
   </row>
</rows>
```

5.9.7.25.4 name

Returns the name of the context node. This is a parameterless variant of the **name** function where the context node is determined by the connections in your mapping. To specify a node explicitly, use the **name** 689 function that takes an input node as parameter.

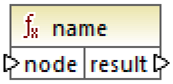


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.25.5 name

Returns the name of a node.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

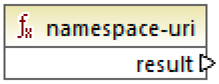
Name	Type	Description
node	`node()`	The input node.

Example

See the example given for the `local-name`[687] function.

5.9.7.25.6 namespace-uri

Returns the namespace URI of the QName of the context node, as an `xs:string`. This is a parameterless variant of the **`namespace-uri`** function where the context node is determined by the connections in your mapping. To specify a node explicitly, use the `namespace-uri`[689] function that takes an input node as parameter.

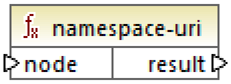


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.25.7 namespace-uri

Returns the namespace URI of the QName of **node**, as an `xs:string`.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
node	node()	The input node.

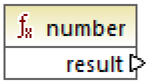
Example

See the example given for the local-name 687 function.

5.9.7.25.8 number

Returns the value of the context node, converted to an `xs:double`. This is a parameterless variant of the **number** function where the context node is determined by the connections in your mapping. To specify a node explicitly, use the number 690 function that takes an input node as parameter.

The only types that can be converted to numbers are Booleans, numeric strings, and other numeric types. Non-numeric input values (such as a non-numeric string) result in NaN (Not a Number).

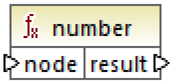


Languages

XQuery, XSLT 2.0, XSLT 3.0.

5.9.7.25.9 number

Returns the value of **node**, converted to an `xs:double`. The only types that can be converted to numbers are Booleans, numeric strings, and other numeric types. Non-numeric input values (such as a non-numeric string) result in NaN (Not a Number).



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

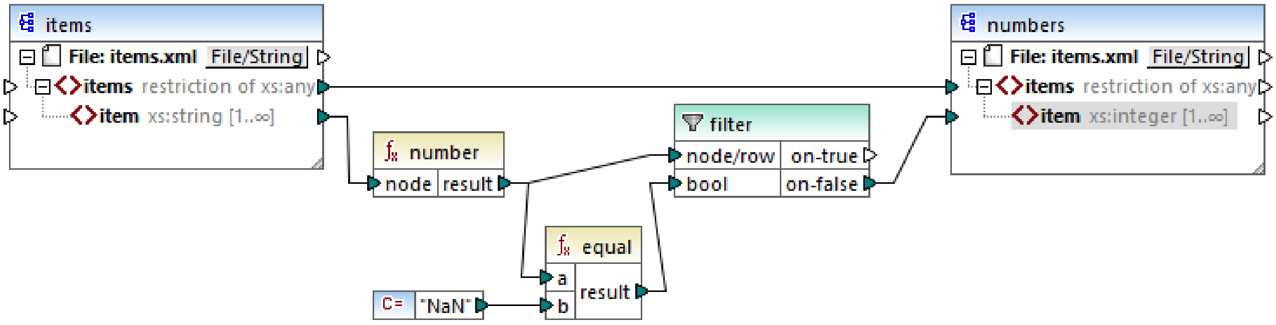
Name	Type	Description
node	`mf:atomic`	The input node.

Example

The following XML contains items of type `string`:

```
<items>
   <item>1</item>
   <item>2</item>
   <item>Jingle Bells</item>
</items>
```

The mapping illustrated below attempts to convert all these strings to numeric values and write them to a target XML file. Notice that the data type of **item** in the target XML component is `xs:integer` while the source **item** is of `xs:string` data type. If the conversion is not successful, the item must be skipped and not copied to the target file.



XSLT 2.0 mapping

To achieve the mapping goal, a filter was used. The **`equal`** function checks if the result of the conversion is "NaN". If this is false, this indicates a successful conversion, so the item is copied to the target. The output of the mapping is as follows:

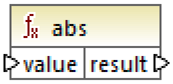
```
<items>
   <item>1</item>
   <item>2</item>
</items>
```

5.9.7.26 xpath2 | numeric functions

The numeric functions of the **xpath2** library include the **`abs`** and **`round-half-to-even`** functions.

5.9.7.26.1 abs

Returns the absolute value of the argument. For example, if the input argument is **-2** or **2**, the function returns **2**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

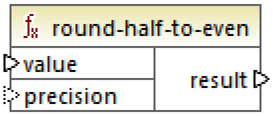
Parameters

Name	Type	Description
value	`xs:decimal`	The input value.

5.9.7.26.2 round-half-to-even

The **`round-half-to-even`** function rounds the supplied number (first argument) to the degree of precision (number of decimal places) supplied in the optional second argument. For example, if the first argument is **2.141567** and the second argument is **3**, then the first argument (the number) is rounded to three decimal places, so the result will be **2.142**. If no precision (second argument) is supplied, the number is rounded to zero decimal places, that is, to an integer.

The "even" in the name of the function refers to the rounding to an even number when a digit in the supplied number is midway between two values. For example, `round-half-to-even(3.475, 2)` would return **3.48**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:decimal`	Mandatory argument which provides the input value to be rounded.

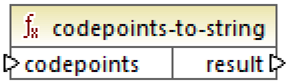
Name	Type	Description
precision	`xs:integer`	Optional argument which specifies the number of decimal places to round to. The default value is **0**.

5.9.7.27 xpath2 | string functions

The string functions of the **xpath2** library enable you to process strings (this includes comparing strings, converting strings to upper or lower case, extracting substrings from strings, and others).

5.9.7.27.1 codepoints-to-string

Creates a string from a sequence of Unicode code points. This function is the opposite of the **`string-to-codepoints`** 701 function.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
codepoints	`ZeroOrMore xs:integer`	This input must be connected to a sequence of items of integer type, where each integer specifies a Unicode code point.

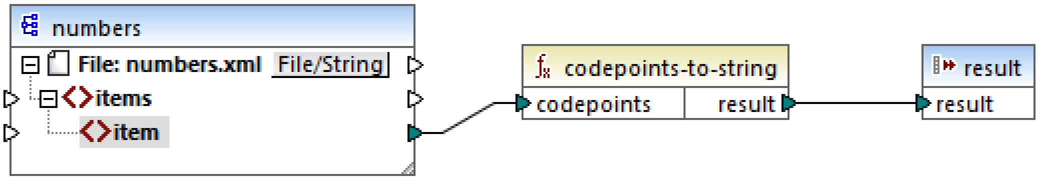
Example

The following XML contains multiple **item** elements that store each Unicode code point values.

```
<items>
   <item>77</item>
   <item>97</item>
   <item>112</item>
   <item>70</item>
   <item>111</item>
   <item>114</item>
   <item>99</item>
```

```
    <item>101</item>
</items>
```

The mapping illustrated below supplies the sequence of items as argument to the **codepoint-to-string** function.



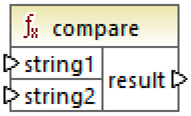
XSLT 2.0 mapping

The mapping output is **MapForce**.

5.9.7.27.2 compare

The **compare** function takes two strings as arguments and compares them for equality and alphabetically. If **string1** is alphabetically less than **string2** (for example the two string are "A" and "B"), then the function returns **-1**. If the two strings are equal (for example, "A" and "A"), the function returns **0**. If **string1** is greater than **string2** (for example, "B" and "A"), then the function returns **1**.

This variant of the function uses the default collation, which is Unicode. Another variant 695 of this function exists where you can supply the collation as argument.



Languages

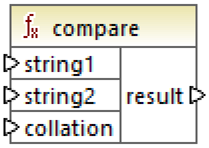
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string1	xs:string	The first input string.
string2	xs:string	The second input string.

5.9.7.27.3 compare

The **compare** function takes two strings as arguments and compares them for equality and alphabetically, using the collation supplied as argument. If **string1** is alphabetically less than **string2** (for example the two string are "A" and "B"), then the function returns **-1**. If the two strings are equal (for example, "A" and "A"), the function returns **0**. If **string1** is greater than **string2** (for example, "B" and "A"), then the function returns **1**.



Languages

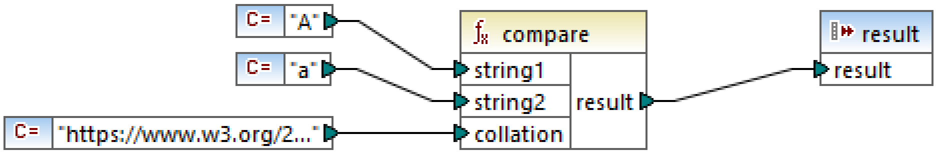
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string1	xs:string	The first input string.
string2	xs:string	The second input string.
collation	xs:string	Specifies the collation to use for string comparison. This input may originate from the output of the **default-collation** 667 function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

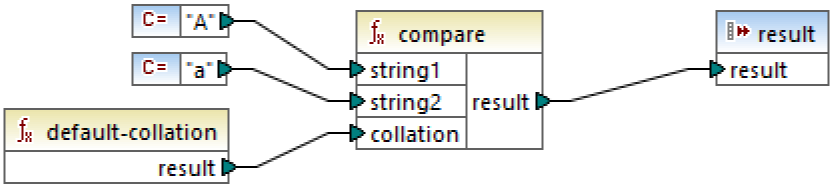
Example

The following mapping compares the strings "A" and "a" using the case insensitive collation `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`, which is supplied by a constant.



XSLT 2.0 Mapping

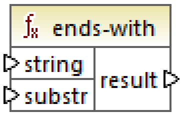
The result of the mapping above is **0**, meaning that both strings are treated as equal. However, if you replace the collation with the one provided by the **default-collation** function, the collation changes to the default Unicode code point collation, and the mapping result becomes **-1** ("A" is alphabetically less than "a").



5.9.7.27.4 ends-with

Returns **true** if **string** ends with **substr**; **false** otherwise. The returned value is of type `xs:boolean`.

This variant of the function uses the default collation, which is Unicode. Another variant [697] of this function exists where you can supply the collation as argument.



Languages

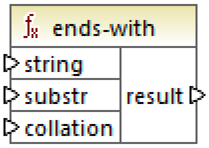
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	The input string (that is, the "haystack").
substr	`xs:string`	The substring (that is, the "needle").

5.9.7.27.5 ends-with

Returns **true** if **string** ends with **substr**; **false** otherwise. The returned value is of type `xs:boolean`.



Languages

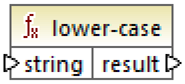
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	The input string (that is, the "haystack").
substr	`xs:string`	The substring (that is, the "needle").
collation	`xs:string`	Specifies the collation to use for string comparison. This input may originate from the output of the `default-collation` 667 function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

5.9.7.27.6 lower-case

Returns the value of **string** after translating every character to its lower-case correspondent.



Languages

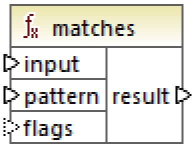
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	The input value.

5.9.7.27.7 matches

The **`matches`** function tests whether a supplied string (the first argument) matches a regular expression (the second argument). The syntax of regular expressions must be that defined for the `pattern` facet of XML Schema. The function returns **true** if the string matches the regular expression, **false** otherwise.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

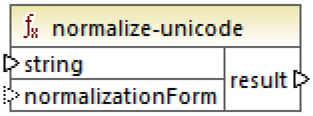
Parameters

Name	Type	Description
input	`xs:string`	The input string.
pattern	`xs:string`	The regular expression to match, see Regular Expressions 508.
flags	`xs:string`	Optional argument that influences the matching. This argument may supply any combination of the following flags: `i`, `m`, `s`, `x`. Multiple flags can be used, for example, `imx`. If no flag is used, the default values of all four flags are used. The four flags are as follows: `i` Use case-insensitive mode. The default is case-sensitive. `m` Use multi-line mode, in which the input string is considered to have multiple lines, each separated by a newline character (`x0a`). The meta characters **^** and **$** indicate the beginning and end of each line. The default is string mode, in which the string starts and ends with the meta characters **^** and **$**. `s` Use dot-all mode. The default is not-dot-all mode, in which the meta character **.** matches all characters except the newline character (`x0a`). In dot-all mode, the dot also matches the newline character.

Name	Type	Description
		x Ignore whitespace. By default, whitespace characters are not ignored.

5.9.7.27.8 normalize-unicode

Returns the value of **string** normalized according to the rules of the normalization form specified (the second argument). For more information about Unicode normalization, see §2.2 of https://www.w3.org/TR/charmod-norm/.



Languages

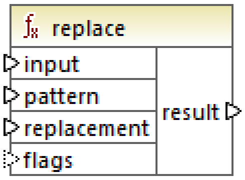
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	The string value to be normalized.
normalizationForm	`xs:string`	Optional argument which supplies the normalization form. The default is Unicode Normalization Form C (NFC). The normalization forms NFC, NFD, NFKC, and NFKD are supported.

5.9.7.27.9 replace

This function takes an input string, a regular expression, and a replacement string as arguments. It replaces all matches of the regular expression in the input string with the replacement string. If the regular expression matches two overlapping strings in the input string, only the first match is replaced.



Languages

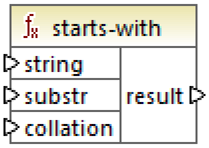
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
input	`xs:string`	The input string.
pattern	`xs:string`	The regular expression to match, see Regular Expressions 508.
replacement	`xs:string`	The replacement string.
flags	`xs:string`	Optional argument that influences the matching. This argument is used in the same way as the **flags** argument of the **`matches`** 698 function.

5.9.7.27.10 starts-with

Returns **true** if **string** starts with **substr**; **false** otherwise. The returned value is of type `xs:boolean`. String comparison takes place according to the specified collation.



Languages

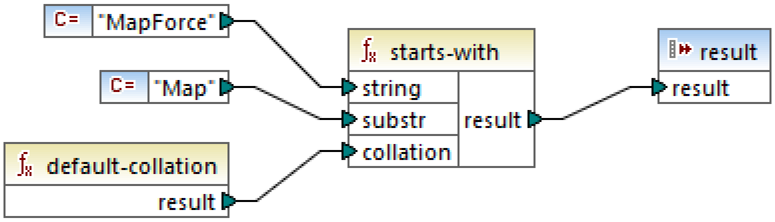
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	xs:string	The input string (that is, the "haystack").
substr	xs:string	The substring (that is, the "needle").
collation	xs:string	Specifies the collation to use for string comparison. This input may originate from the output of the **default-collation** 667 function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

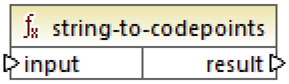
Example

The following mapping returns the value `true`, because the input string "MapForce" begins with the substring "Map", assuming that the default Unicode collation is used.



5.9.7.27.11 string-to-codepoints

Returns the sequence of Unicode code points (integer values) that constitute the string supplied as argument. This function is the opposite of the **codepoints-to-string** 693 function.



Languages

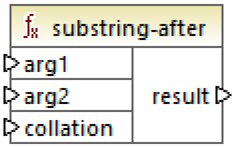
XQuery, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
input	`xs:string`	The input string

5.9.7.27.12 substring-after

Returns the part of string **arg1** that occurs after the string **arg2**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

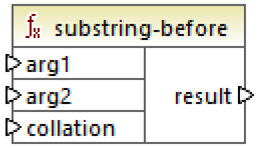
Name	Type	Description
arg1	`xs:string`	The input string (that is, the "haystack").
arg2	`xs:string`	The substring (that is, the "needle").
collation	`xs:string`	Specifies the collation to use for string comparison. This input may originate from the output of the **`default-collation`** 667 function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

Example

If **arg1** is "MapForce", **arg2** is "Map", and **collation** is **`default-collation`** 667, the function returns "Force".

5.9.7.27.13 substring-before

Returns the part of string **arg1** that occurs before the string **arg2**.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

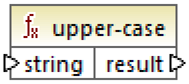
Name	Type	Description
arg1	`xs:string`	The input string (that is, the "haystack").
arg2	`xs:string`	The substring (that is, the "needle").
collation	`xs:string`	Specifies the collation to use for string comparison. This input may originate from the output of the `default-collation`[667] function or it may be a collation such as `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive`.

Example

If **arg1** is "MapForce", **arg2** is "Force", and **collation** is `default-collation`[667], the function returns "Map".

5.9.7.27.14 upper-case

Returns the value of **string** after translating every character to its upper-case correspondent.



Languages

XQuery, XSLT 2.0, XSLT 3.0.

Parameters

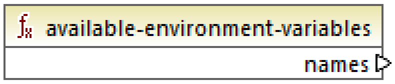
Name	Type	Description
string	`xs:string`	The input string.

5.9.7.28 xpath3 | external information functions

The external information functions of the **xpath3** library enable you to obtain information about the XSLT execution environment or retrieve data from external resources.

5.9.7.28.1 available-environment-variables

Returns a list of environment variable names that are suitable for passing to the **`environment-variable`** function, as a (possibly empty) sequence of strings.

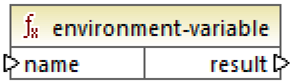


Languages

XSLT 3.0.

5.9.7.28.2 environment-variable

Returns the value of a system environment variable, if it exists. The return type is `xs:string`.



Languages

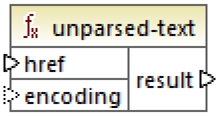
XSLT 3.0.

Parameters

Name	Type	Description
name	`xs:string`	The name of the environment variable.

5.9.7.28.3 unparsed-text

Reads an external resource (for example, a file) and returns a string representation of the resource.



Languages

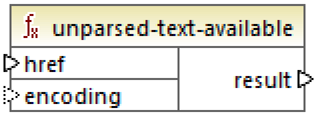
XSLT 3.0.

Parameters

Name	Type	Description
href	`xs:string`	A string in the form of a URI reference.
encoding	`xs:string`	Optional argument. Specifies the name of the encoding, for example "UTF-8", "UTF-16". If the encoding cannot be determined automatically, then UTF-8 is assumed.

5.9.7.28.4 unparsed-text-available

Determines whether a call to **`unparsed-text`** with particular arguments would succeed. The return type is `xs:boolean`.



Languages

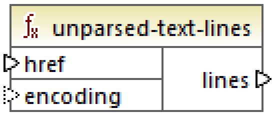
XSLT 3.0.

Parameters

Name	Type	Description
href	`xs:string`	A string in the form of a URI reference.
encoding	`xs:string`	Optional argument. Specifies the name of the encoding, for example "UTF-8", "UTF-16". If the encoding cannot be determined automatically, then UTF-8 is assumed.

5.9.7.28.5 unparsed-text-lines

Reads an external resource (for example, a file) and returns its contents as a sequence of strings, one for each line of text in the string representation of the resource.



Languages

XSLT 3.0.

Parameters

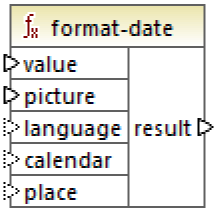
Name	Type	Description
href	`xs:string`	A string in the form of a URI reference.
encoding	`xs:string`	Optional argument. Specifies the name of the encoding, for example "UTF-8", "UTF-16". If the encoding cannot be determined automatically, then UTF-8 is assumed.

5.9.7.29 xpath3 | formatting functions

The formatting functions available of the **xpath3** library are used to format date, time and integer values.

5.9.7.29.1 format-date

Returns a string containing an `xs:date` value formatted for display.



Languages

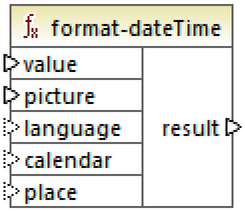
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:date`	The input `xs:date` value to be formatted. Mandatory parameter.
picture	`xs:string`	Mandatory parameter. See section 9.8.4.1 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
language	`xs:string`	Optional parameter. See section 9.8.4.8 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
calendar	`xs:string`	Same as above.
place	`xs:string`	Same as above.

5.9.7.29.2 format-dateTime

Returns a string containing an `xs:dateTime` value formatted for display.



Languages

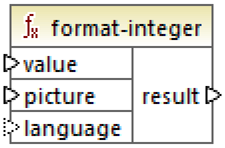
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:dateTime`	The input `xs:dateTime` value to be formatted.
picture	`xs:string`	Mandatory parameter. See section 9.8.4.1 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
language	`xs:string`	Optional parameter. See section 9.8.4.8 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
calendar	`xs:string`	Same as above.
place	`xs:string`	Same as above.

5.9.7.29.3 format-integer

Formats an integer according to a given picture string, using the conventions of a given natural language if specified.



Languages

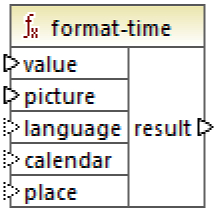
XSLT 3.0.

Parameters

Name	Type	Description
value	xs:integer	The input integer value to be formatted.
picture	xs:string	Mandatory parameter. See section 4.6.1 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
language	xs:string	Optional parameter. Specifies the natural language according to which the value should be formatted. If specified, this value must be either an empty string or any value that would be allowed for the `xml:lang` attribute according to the "Extensible Markup Language (XML) 1.0 W3C Recommendation (https://www.w3.org/TR/xml).

5.9.7.29.4 format-time

Returns a string containing an `xs:time` value formatted for display.



Languages

XSLT 3.0.

Parameters

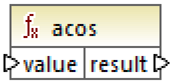
Name	Type	Description
value	`xs:time`	The input `xs:time` value to be formatted.
picture	`xs:string`	Mandatory parameter. See section 9.8.4.1 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
language	`xs:string`	Optional parameter. See section 9.8.4.8 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).
calendar	`xs:string`	Same as above.
place	`xs:string`	Same as above.

5.9.7.30 xpath3 | math functions

The math functions of the **xpath3** library are used to perform trigonometric and other mathematical calculations.

5.9.7.30.1 acos

Returns the arc cosine of an angle, in the range of **0** through **pi**.



Languages

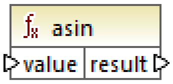
XSLT 3.0.

Parameters

Name	**Type**	**Description**
value	`xs:double`	The input value.

5.9.7.30.2 asin

Returns the arc sine of an angle, in the range of **-pi/2** through **pi/2**.



Languages

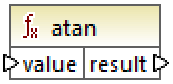
XSLT 3.0.

Parameters

Name	**Type**	**Description**
value	`xs:double`	The input value.

5.9.7.30.3 atan

Returns the arc tangent of an angle, in the range of **-pi/2** through **pi/2**.



Languages

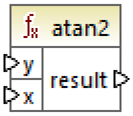
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.30.4 atan2

Returns the angle in radians subtended at the origin by the point on a plane with coordinates (x, y) and the positive x-axis.



Languages

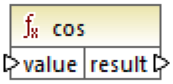
XSLT 3.0.

Parameters

Name	Type	Description
y	`xs:double`	The x coordinate.
x	`xs:double`	The y coordinate.

5.9.7.30.5 cos

Returns the trigonometric cosine of the angle given by value. The unit of value is radian.



Languages

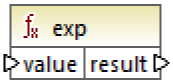
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.30.6 exp

Returns Euler's number *e* raised to the power of the value.



Languages

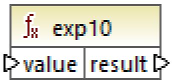
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.30.7 exp10

Returns 10 raised to the power of the value.



Languages

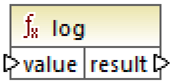
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.30.8 log

Returns the natural logarithm (base e) of a value.



Languages

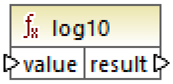
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.30.9 log10

Returns the decimal logarithm (base 10) of a value.



Languages

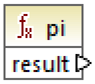
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.30.10 pi

Returns an approximation to the mathematical constant **pi**.

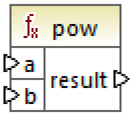


Languages

XSLT 3.0.

5.9.7.30.11 pow

Returns the value of **a** raised to the power of **b**.



Languages

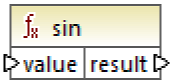
XSLT 3.0.

Parameters

Name	Type	Description
a	`xs:double`	The input value **a**.
b	`xs:double`	The input value **b**.

5.9.7.30.12 sin

Returns the trigonometric sine of the angle given by value. The unit of value is radian.



Languages

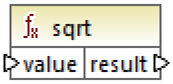
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.30.13 sqrt

Returns the non-negative square root of the argument.



Languages

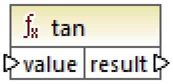
XSLT 3.0.

Parameters

Name	Type	Description
value	`xs:double`	The input value.

5.9.7.30.14 tan

Returns the trigonometric tangent of the angle given by value. The unit of value is radian.



Languages

XSLT 3.0.

Parameters

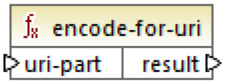
Name	Type	Description
value	`xs:double`	The input value.

5.9.7.31 xpath3 | URI functions

The URI functions in the **xpath3** library perform encoding, escaping, and conversion of values intended for use in URIs.

5.9.7.31.1 encode-for-uri

Encodes reserved characters in a string that is intended to be used in the path segment of a URI. For further information about this function, see section 6.2 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).



Languages

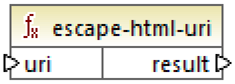
XSLT 3.0.

Parameters

Name	Type	Description
uri-part	`xs:string`	The input URI value to encode.

5.9.7.31.2 escape-html-uri

Escapes a URI in the same way that HTML user agents handle attribute values expected to contain URIs. For further information about this function, see section 6.4 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).



Languages

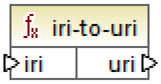
XSLT 3.0.

Parameters

Name	Type	Description
uri	`xs:string`	The input URI value to escape.

5.9.7.31.3 iri-to-uri

Converts a string containing an IRI (Internationalized Resource Identifier) into a URI (Uniform Resource Identifier). For further information about this function, see section 6.3 of the "XPath and XQuery Functions and Operators 3.1" W3C Recommendation (https://www.w3.org/TR/xpath-functions-31).



Languages

XSLT 3.0.

Parameters

Name	Type	Description
iri	`xs:string`	The input IRI value.

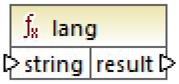
5.9.7.32 xslt | xpath functions

The functions in this sub-group are XPath 1.0 functions that retrieve information about mapping items (or nodes). Most of these functions take a node as argument and return information about that node. The `last` and `position` functions operate in the current mapping context[79], which is determined by the connections on your mapping.

Note: Additional XPath 1.0 functions can be found in the **core** library.

5.9.7.32.1 lang

Returns **true** if the context node has an `xml:lang` attribute with a value that either matches exactly the **string** argument, or is a subset of it. Otherwise, the function returns **false**.



Languages

XSLT 1.0.

Parameters

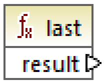
Name	Type	Description
string	`xs:string`	The language code to check, for example, "en".

Example

See the example given for the `lang`[685] function of the **xpath2** library.

5.9.7.32.2 last

Returns the position number of the last node in the processed node list.



Languages

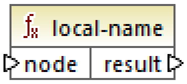
XSLT 1.0.

Example

See the example given for the `last`[668] function of the **xpath2** library.

5.9.7.32.3 local-name

Returns the local part of the name of the node supplied as argument.



Languages

XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

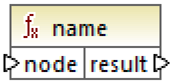
Name	Type	Description
node	`node()`	The input node.

Example

See the example given for the `local-name`[687] function of the **xpath2** library.

5.9.7.32.4 name

Returns the name of the node supplied as argument.



Languages

XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

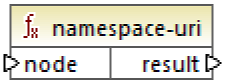
Name	Type	Description
node	`node()`	The input node.

Example

See the example given for the `local-name`[687] function of the **xpath2** library.

5.9.7.32.5 namespace-uri

Returns the namespace URI of the node supplied as argument.



Languages

XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

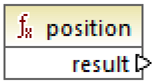
Name	Type	Description
node	`node()`	The input node.

Example

See the example given for the `local-name` 687 function of the **`xpath2`** library.

5.9.7.32.6 position

Returns the position of the current node in the node set that is being processed.



Languages

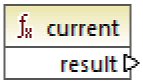
XSLT 1.0.

5.9.7.33 xslt | xslt functions

The functions in this group are miscellaneous XSLT 1.0 functions.

5.9.7.33.1 current

The **`current`** function takes no argument and returns the current node.

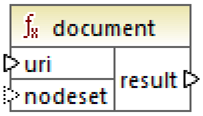


Languages

XSLT 1.0.

5.9.7.33.2 document

Accesses nodes from an external XML document. The result is output to a node in the output document.



Languages

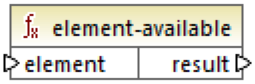
XSLT 1.0.

Parameters

Name	Type	Description
uri	`xs:string`	Mandatory. Specifies the path to the XML document. The XML document must be valid and parseable.
nodeset	`node()`	Optional. Specifies a node, the base URI of which is used to resolve the URI supplied as the first argument if it is relative.

5.9.7.33.3 element-available

The **`element-available`** function tests whether an element, entered as the only string argument of the function, is supported by the XSLT processor. The argument string is evaluated as a QName. Therefore, XSLT elements must have an `xsl:` prefix and XML Schema elements must have an `xs:` prefix—since these are the prefixes declared for these namespaces in the underlying XSLT that will be generated for the mapping. The function returns a Boolean.



Languages

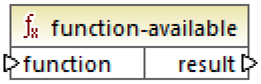
XSLT 1.0.

Parameters

Name	Type	Description
element	`xs:string`	The element name.

5.9.7.33.4 function-available

The **`function-available`** function is similar to the **`element-available`** function and tests whether the function name supplied as the function's argument is supported by the XSLT processor. The input string is evaluated as a QName. The function returns a Boolean.



Languages

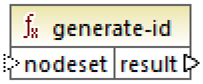
XSLT 1.0.

Parameters

Name	Type	Description
function	`xs:string`	The function name.

5.9.7.33.5 generate-id

The **`generate-id`** function generates a unique string that identifies the first node in the node set identified by the optional input argument. If no argument is supplied, the ID is generated on the context node. The result can be directed to any node in the output document.



Languages

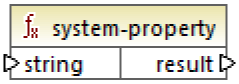
XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
nodeset	`node()`	Optional argument that supplies the input node.

5.9.7.33.6 system-property

The **`system-property`** function returns properties of the XSLT processor (the system). Three system properties, all in the XSLT namespace, are mandatory for XSLT processors. These are `xsl:version`, `xsl:vendor`, and `xsl:vendor-url`. The input string is evaluated as a QName and so must have the `xsl:` prefix, since this is the prefix associated with the XSLT namespace in the underlying XSLT stylesheet.



Languages

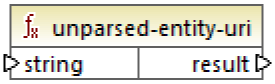
XSLT 1.0, XSLT 2.0, XSLT 3.0.

Parameters

Name	Type	Description
string	`xs:string`	Specifies the property name, which can be any of the following: `xsl:version, xsl:vendor`, `xsl:vendor-url`.

5.9.7.33.7 unparsed-entity-uri

If you are using a DTD, you can declare an unparsed entity in it. This unparsed entity (for example, an image) will have a URI that locates the unparsed entity. The input string of the function must match the name of the unparsed entity that has been declared in the DTD. The function then returns the URI of the unparsed entity, which can then be directed to a node in the output document, for example, to an **href** node.



Languages

XSLT 1.0.

Parameters

Name	Type	Description
string	`xs:string`	The name of the unparsed entity whose URI should be retrieved.

6 Advanced Mapping Procedures

Altova website: 🔗 Data integration tool

This section describes advanced mapping scenarios and includes the following topics:

- Mapping Node Names 726
- Batch-Process Files 746
- Parsing and Serializing Strings 753

6.1 Map Node Names

Most of the time when you create a mapping with MapForce, the goal is to read *values* from a source and write *values* to a target. However, there might be cases when you want to access not only the node *values* from the source, but also the node names. For example, you might want to create a mapping which reads the element or attribute names (not values) from a source XML and converts them to element or attribute values (not names) in a target XML.

Consider the following example: you have an XML file that contains a list of products. Each product has the following format:

```
 <product>
    <id>1</id>
    <color>red</color>
    <size>10</size>
 </product>
```

Your goal is to convert information about each product into name-value pairs, for example:

```
 <product>
    <attribute name="id" value="1" />
    <attribute name="color" value="red" />
    <attribute name="size" value="10" />
</product>
```

For such scenarios, you would need access to the node name from the mapping. With *dynamic* access to node names, you can perform data conversions such as the one above.

Note: You can also perform the transformation above by using the `node-name`[557] and `static-node-name`[558] core library functions. However, in this case, you need to know exactly what element names you expect from the source, and you need to connect every single such element manually to the target. Also, these functions might not be sufficient, for example, when you need to filter or group nodes by name, or when you need to manipulate the data type of the node from the mapping.

Accessing node names dynamically is possible not only when you need to read node names, but also when you need to write them. In a standard mapping, the name of attributes or elements in a target is always known before the mapping runs; it comes from the underlying schema of the component. With dynamic node names, however, you can create new attributes or elements whose name is not known before the mapping runs. Specifically, the name of the attribute or element is supplied by the mapping itself, from any source supported by MapForce.

For dynamic access to a node's children elements or attributes to be possible, the node must actually have children elements or attributes, and it must not be the XML root node.

Dynamic node names are supported when you map to or from the following component types:

- XML
- CSV/FLF*

* Requires MapForce Professional or Enterprise Edition.

Note: In case of CSV/FLF, dynamic access implies access to "fields" instead of "nodes", since CSV/FLF structures do not have "nodes".

> When the mapping target is a CSV or FLF (fixed-length field) file, the fields must be defined in the component settings (and it is not possible to change the name, order, or number of the target fields). Unlike XML, the format of text files is fixed, so only the actual field value can be manipulated, not the field name, number or order.

Dynamic node names are supported in any of the following mapping languages: Built-In*, XSLT 2.0, XSLT 3.0, XQuery*, C#*, C++*, Java*.

* These languages require MapForce Professional or Enterprise Edition.

For information about how dynamic node names work, see Getting Access to Node Names 727. For a step-by-step mapping example, see Example: Map Element Names to Attribute Values 738.

6.1.1 Get 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\MapForce2025\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\MapForce2025\MapForceExamples\Tutorial\Products.xml**. When prompted to select a root element, choose `products`.

To enable dynamic node names for the `product` node, right-click it and select one of the following context menu commands:

- **Show Attributes with Dynamic Name**, if you want to get access to the node's attributes
- **Show Child Elements with Dynamic Name**, if you want to get access to the node's child elements

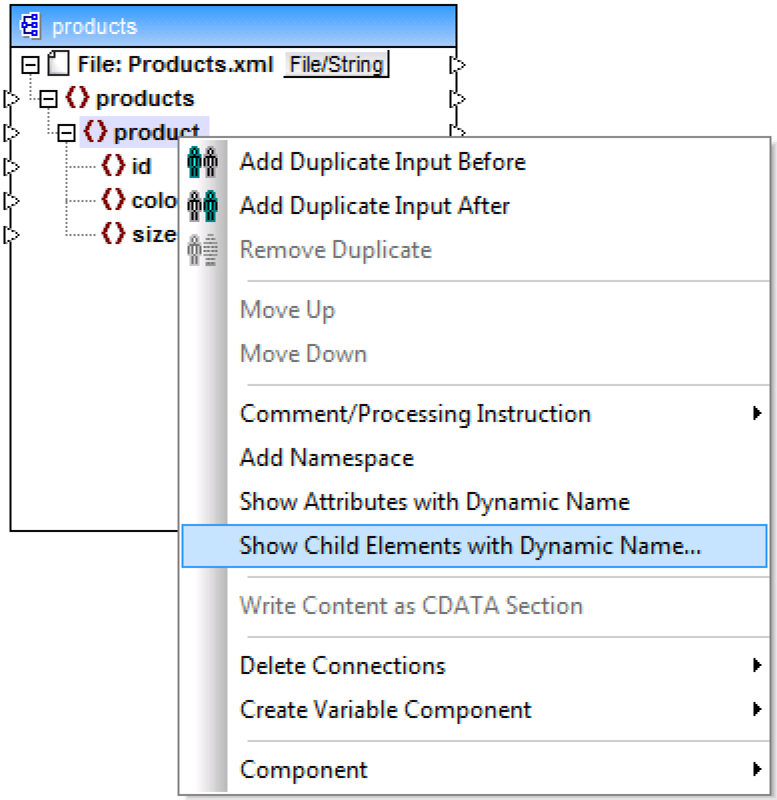
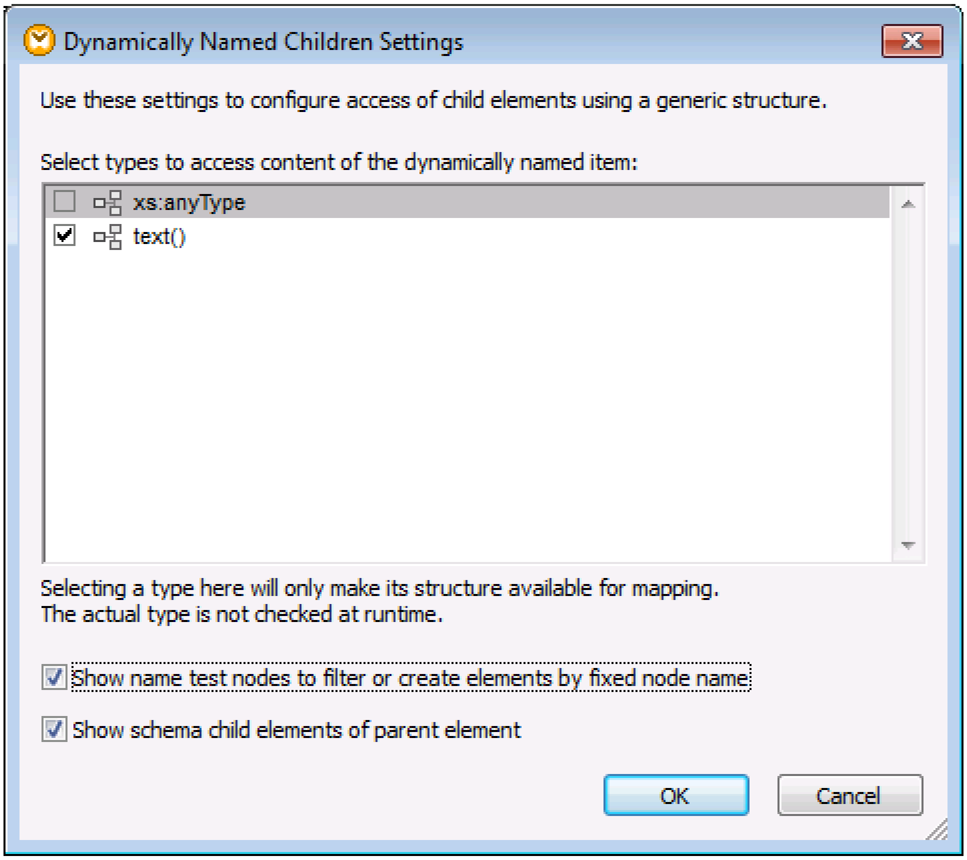

products
File: Products.xml
File/String
products
product
id
colo
size
Add Duplicate Input Before
Add Duplicate Input After
Remove Duplicate
Move Up
Move Down
Comment/Processing Instruction
Add Namespace
Show Attributes with Dynamic Name
Show Child Elements with Dynamic Name...
Write Content as CDATA 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 735.



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()

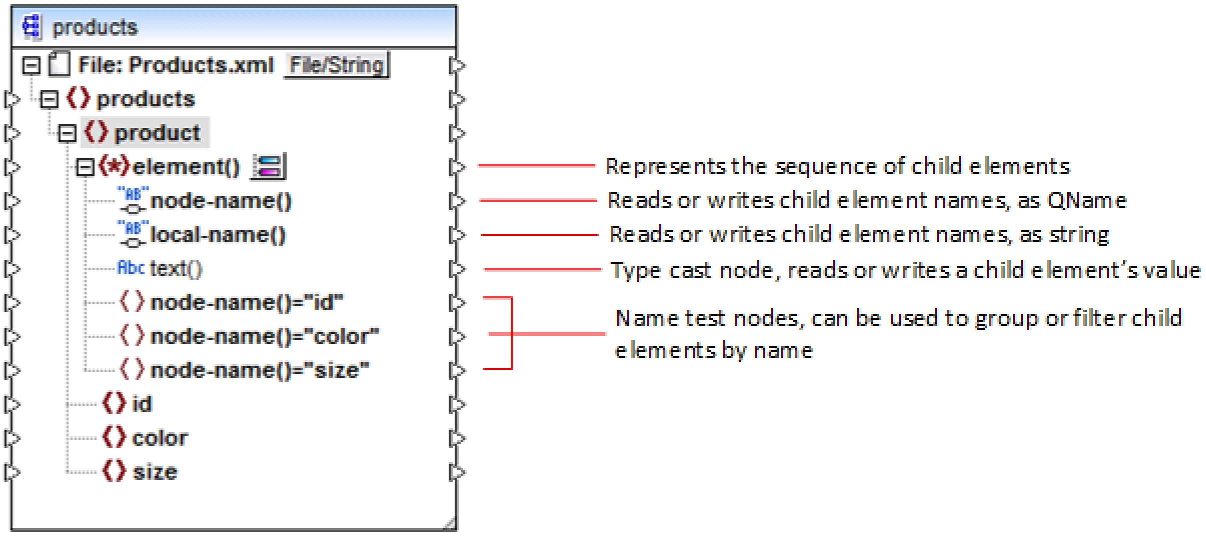
Selecting a type here will only make its structure available for mapping.
The actual type is not checked at runtime.

☑ 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.



Represents the sequence of child elements

Reads or writes child element names, as QName

Reads or writes child element names, as string

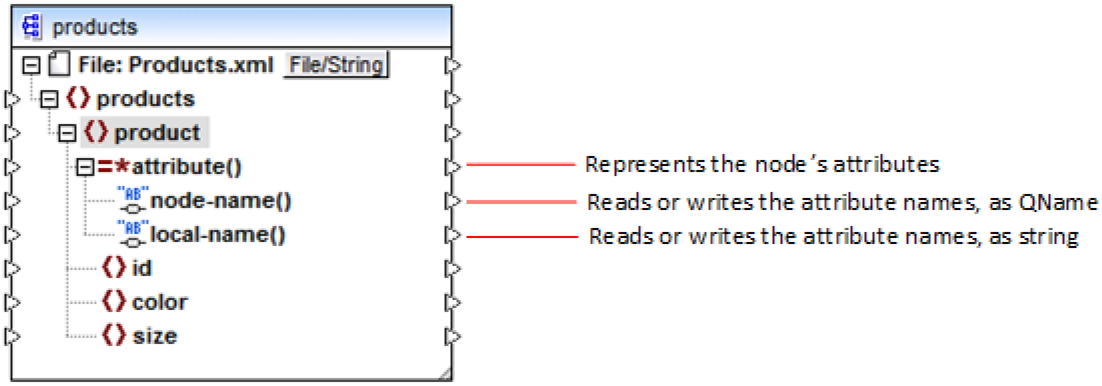
Type cast node, reads or writes a child element's value

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.



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 742.

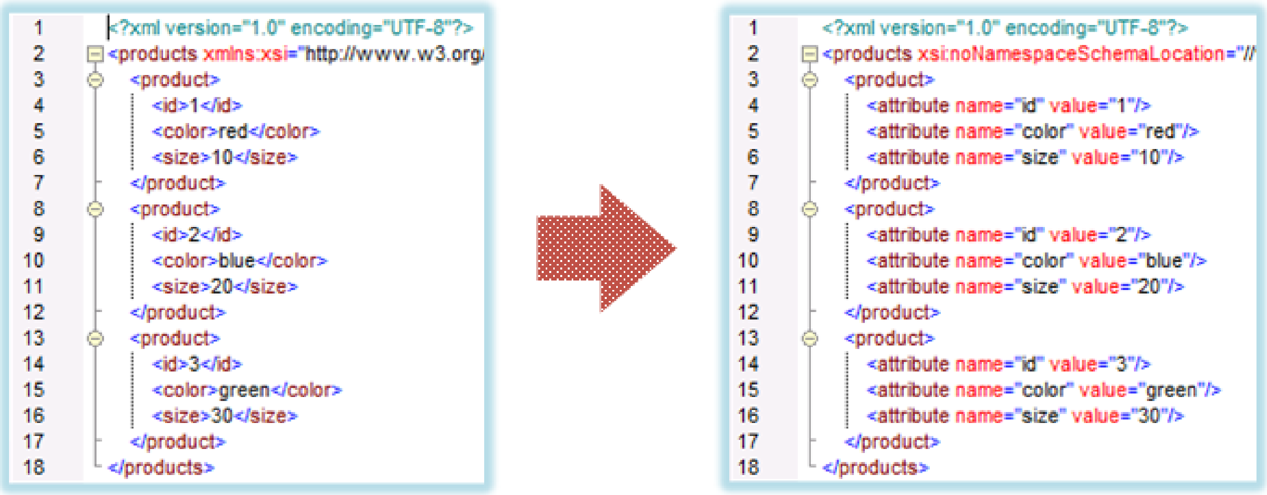
The node types that you can work with in "dynamic node name" mode are described below.

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:

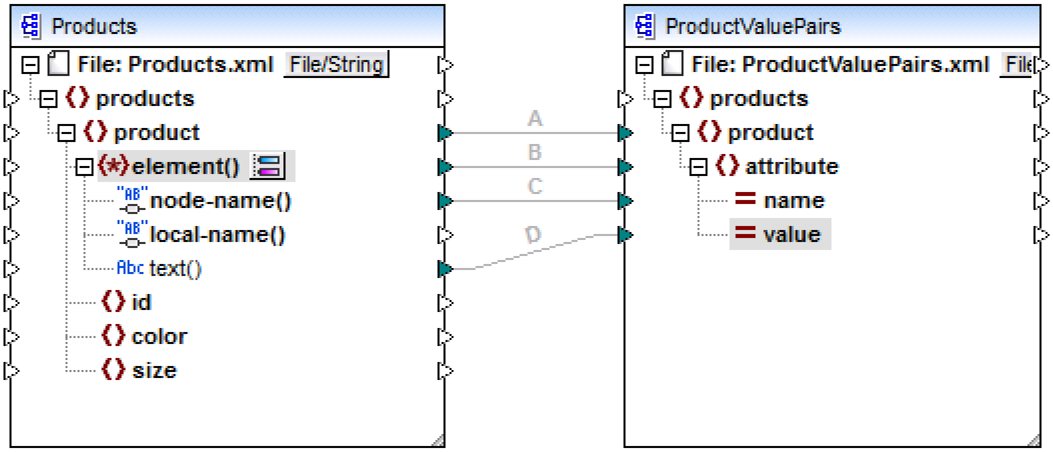


Fig. 7 Mapping XML element names to attribute values (in MapForce)

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 738.

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).

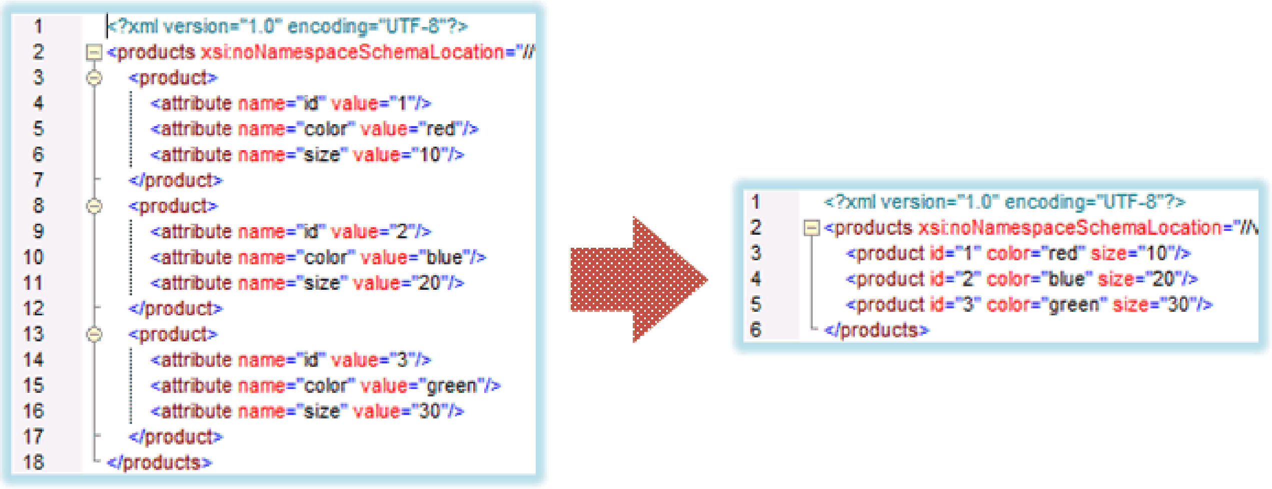
attribute()

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:

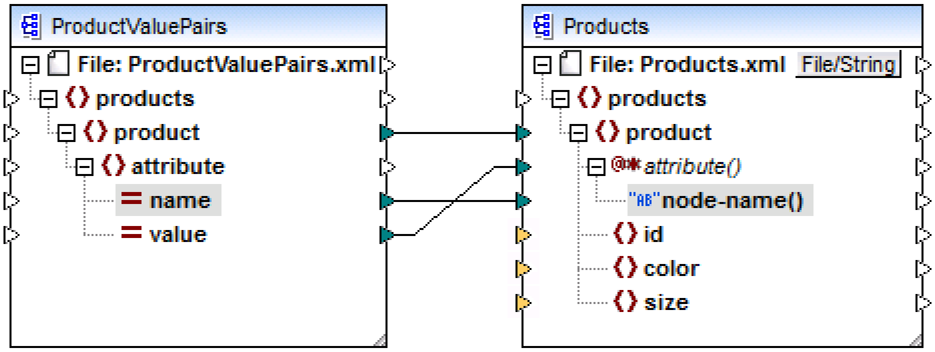


Fig. 9   Mapping attribute values to attribute names (in MapForce)

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\MapForce2025\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 644.

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 735.

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 735). For an example, see Example: Group and Filter Nodes by Name 742.

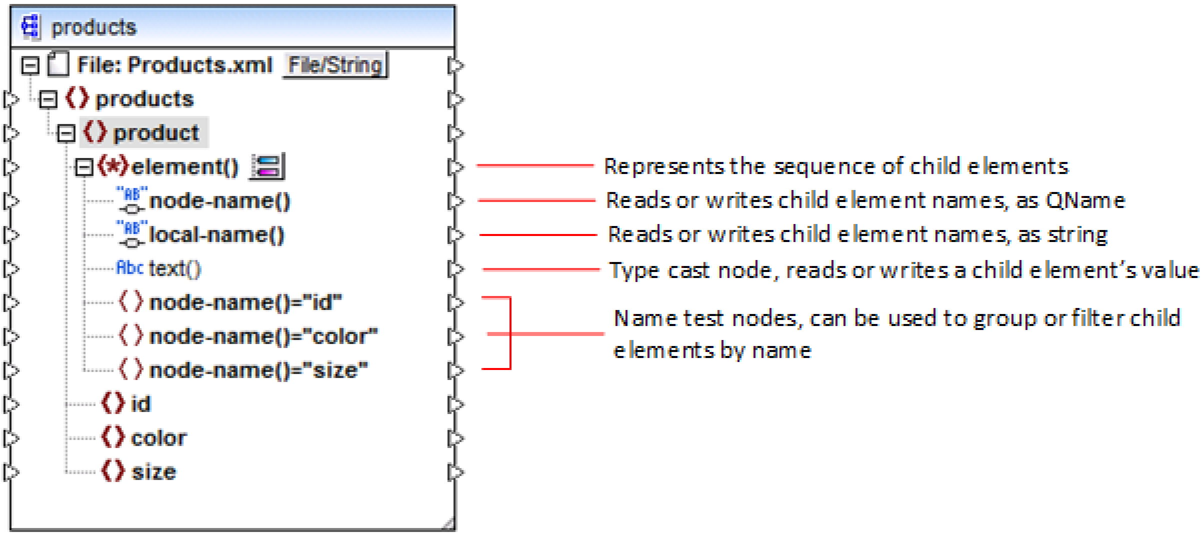
In general, the name test nodes work almost like normal element nodes for reading and writing values and subtree structures. However, because the mapping semantics is different when dynamic access is enabled, there are some limitations. For example, you cannot concatenate the value of two name test nodes.

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()`.

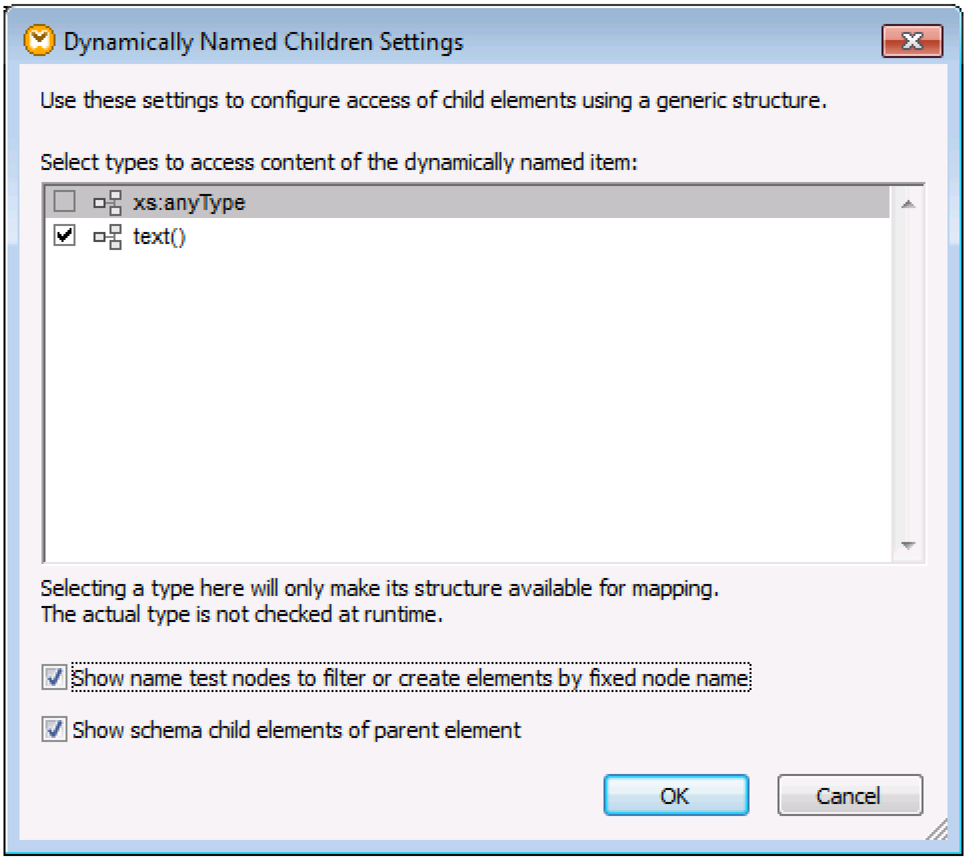
If necessary, you can hide the name test nodes from the component. To do this, click the **Change Selection** (▤) button next to the `element()` node. Then, in the "*Dynamically Named Children Settings*" dialog box (Fig. 2), clear the **Show name test nodes...** check box.

6.1.2 Get Access to Nodes of Specific Type

As mentioned in the previous section, Getting Access to Node Names 727, 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 738. 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**.

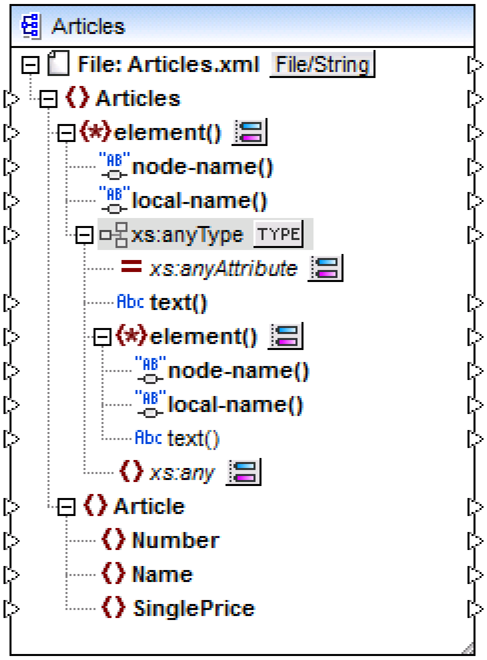
> Be aware that, at mapping runtime, MapForce (through the type cast node) has no information as to what the actual type of the instance node is. Therefore, your mapping must access the node content using the correct type. For example, if you expect that the node of a source XML instance may have children 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 742.

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 738, you don't need this technique because the mapping accesses only the immediate children of an XML node. However, if you need to access deeper structures dynamically, such as "grandchildren", "great-grandchildren", and so on, this is possible as shown below.

1. Create a new mapping.
2. On the Insert menu, click **Insert XML Schema/File** and browse for the XML instance file (in this example, the **Articles.xml** file from the **<Documents>\Altova\MapForce2025\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 137).
- 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 145). If using this option, make sure that the selected attribute or element can actually exist at that particular level according to the schema.

6.1.3 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\MapForce2025\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.

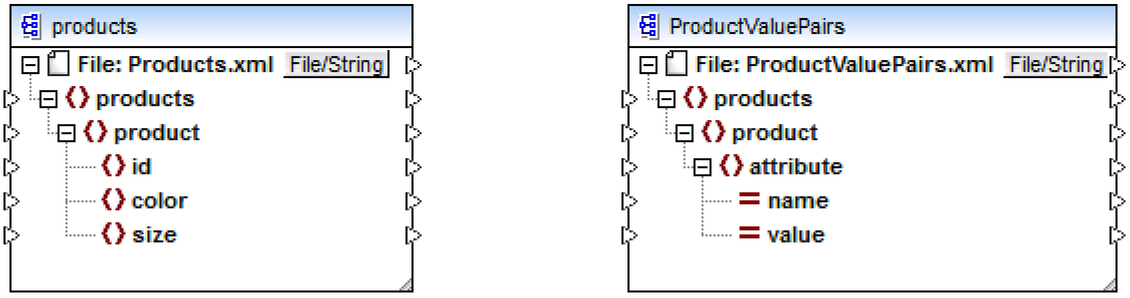
Step 1: Add the source XML component to the mapping

- On the **Insert** menu, click **XML Schema/File**, and browse for the following file: **<Documents>\Altova\MapForce2025\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

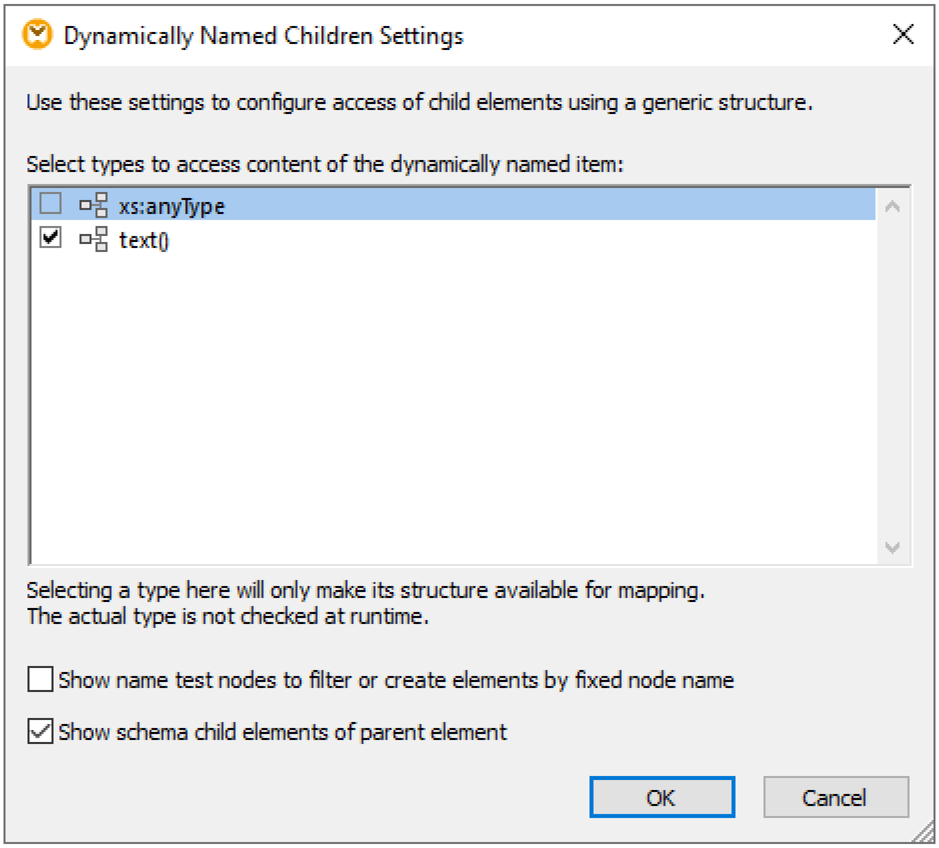
- On the **Insert** menu, click **XML Schema/File**, and browse for the following schema file: **<Documents>\Altova\MapForce2025\MapForceExamples\Tutorial\ProductValuePairs.xsd**. When prompted to supply an instance file, click **Skip**. When prompted to select a root element, select `products` as root element.

At this stage, the mapping should look as follows:



Step 3: Enable dynamic access to child nodes

1. Right-click the `product` node on the source component, and select **Show Child Elements with Dynamic Name** from the context menu.
2. In the dialog box which opens, select **text()** as type. Leave other options as is.



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:

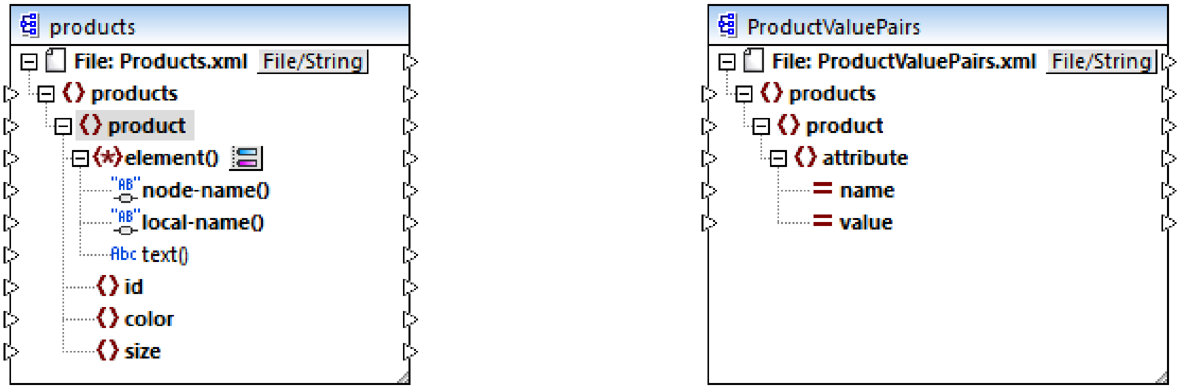
□ xs:anyType
☑ text()

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

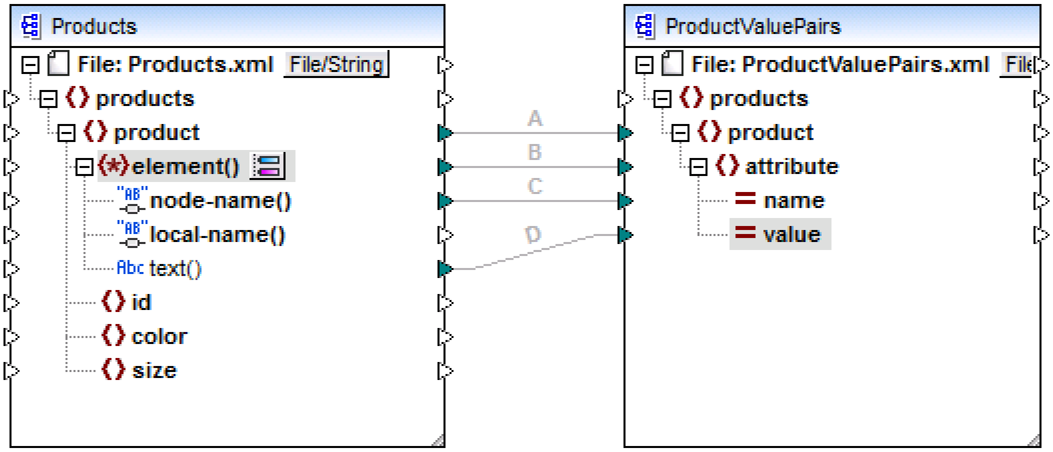
☑ Show schema child elements of parent element

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** pane and observe the generated XML. As expected, the output contains several products whose data is stored as name-value pairs, which was the intended goal of this mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<products xsi:noNamespaceSchemaLocation="ProductValuePairs.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <product>
      <attribute name="id" value="1"/>
      <attribute name="color" value="red"/>
      <attribute name="size" value="10"/>
   </product>
   <product>
      <attribute name="id" value="2"/>
      <attribute name="color" value="blue"/>
      <attribute name="size" value="20"/>
   </product>
   <product>
      <attribute name="id" value="3"/>
      <attribute name="color" value="green"/>
      <attribute name="size" value="30"/>
   </product>
</products>
```

Generated mapping output

6.1.4 Example: Group and Filter Nodes by Name

This example shows you how to design a mapping that reads key-value pairs from an XML property list (or XML plist) and writes them to a CSV file. (XML property lists represent a way of storing macOS and iOS object information in XML format, see https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/PropertyLists/UnderstandXMLPlist/UnderstandXMLPlist.html.) The example is accompanied by a mapping sample which is available at the following path: **<Documents>\Altova\MapForce2025\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 576 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".

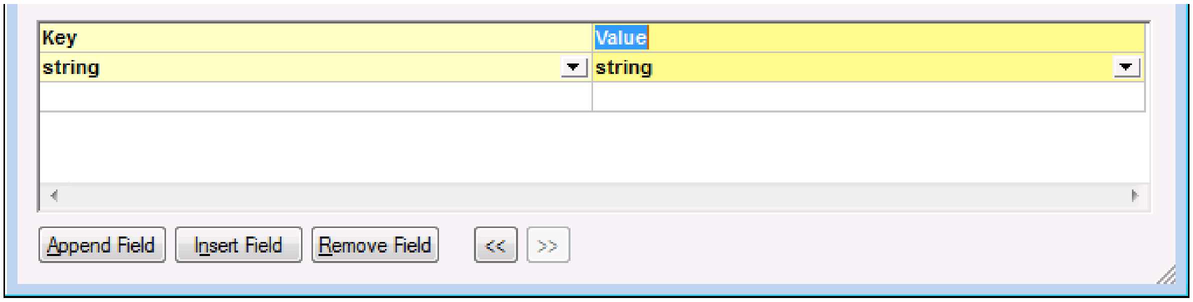
The following steps show how the required mapping can be created.

Step 1: Add the source XML component to the mapping

1. Set the mapping transformation language to BUILT-IN 20.
2. On the **Insert** menu, click **XML Schema/File**, and browse for the following file: **<Documents>\Altova\MapForce2025\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

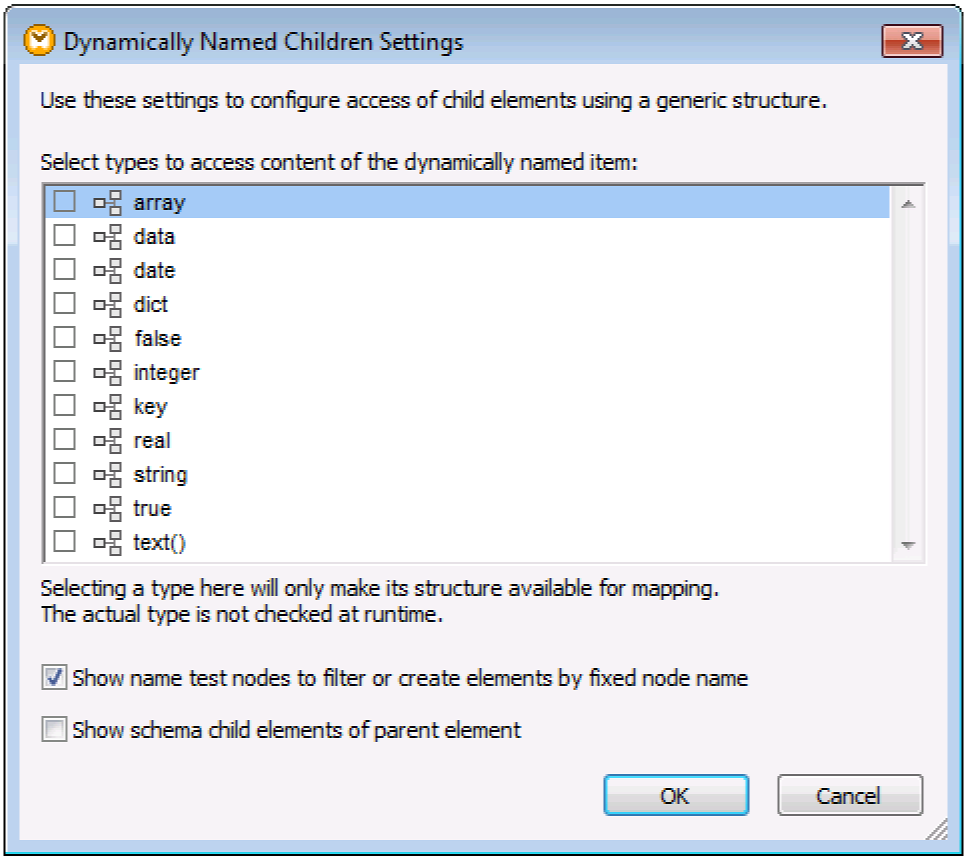
1. On the **Insert** menu, click **Text File**. When prompted, select the **Use simple processing for standard CSV...** option.
2. Add a CSV field to the component, by clicking **Append field**.
3. Double-click the name of each field, and enter "Key" as name of the first field, and "Value" as name of 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 323.



Key	Value
string	string

Step 3: Add the filter and functions

1. Drag the **equal**[543], **exists**[563] and **group-starting-with**[576] functions from the Libraries window into the mapping. For general information about functions, see Functions[436].
2. To add the filter, click the **Insert** menu, and then click **Filter: Nodes/Rows**. For general information about filters, see Filters and Conditions[408].
3. On the **Insert** menu, click **Constant**, and then enter the text "string".
4. In the source component, right-click the `dict` node and select **Show Child Elements with Dynamic Name** from the context menu. On the "Dynamically Named Children Settings" dialog box, make sure that the check box **Show name test nodes to filter or create elements by fixed node name** is selected.



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:

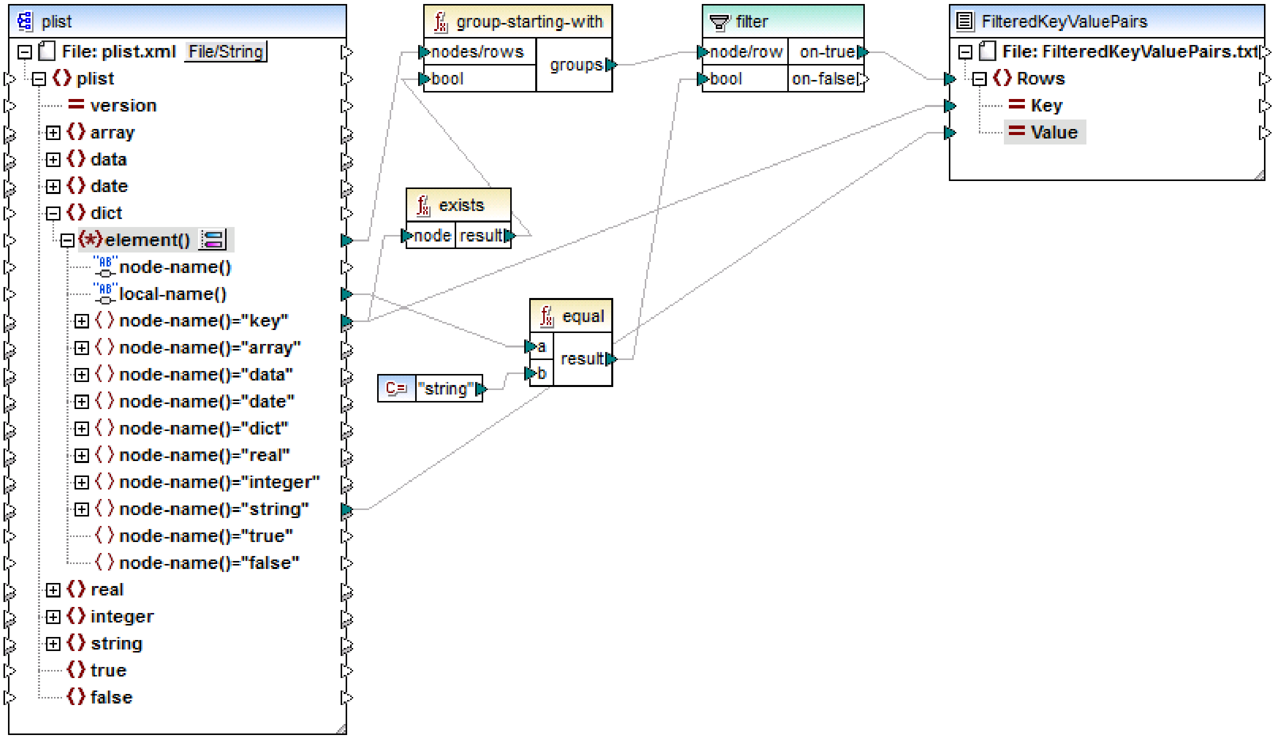
- ☐ array
- ☐ data
- ☐ date
- ☐ dict
- ☐ false
- ☐ integer
- ☐ key
- ☐ real
- ☐ string
- ☐ true
- ☐ text()

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.

6.2 Batch-Process Files

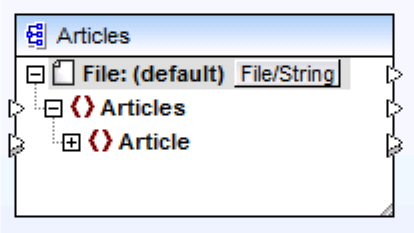
You can configure MapForce to process multiple files (for example, all files in a directory) when the mapping runs. Using this feature, you can solve tasks such as:

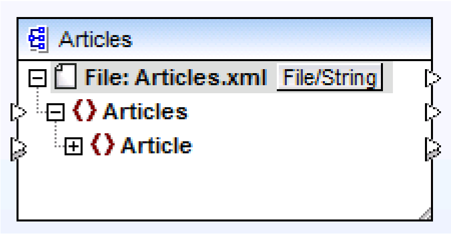
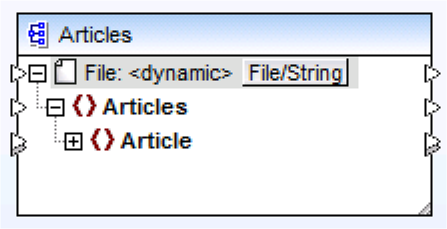
- Supply to the mapping a list of input files to be processed
- Generate as mapping output a list of files instead of a single output file
- Generate a mapping application where both the input and output file names are defined at runtime
- Convert a set of files to another format
- Split a large file (or database) into smaller parts
- Merge multiple files into one large file (or load them into a database)

You can configure a MapForce component to process multiple files in one of the following ways:

- Supply the path to the required input or output file(s) using wildcard characters instead of a fixed file name, in the component settings (see Changing the Component Settings 41). 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 753.
Serialize XML to Strings, Serialize JSON to Strings, Serialize CSV to Strings, Serialize FLF to Strings, Serialize EDI to Strings	When switched on, this option enables the component to accept a structure as input, and convert it to string. The input structure can be XML, JSON, CSV, Fixed-length Field, or EDI, respectively. For more information, see Parsing and Serializing Strings 753.

Multiple input or output files can be defined for the following components:

- XML files
- Text files (CSV*, FLF* files and FlexText** files)
- EDI documents**
- Excel spreadsheets**
- XBRL documents**
- JSON files**
- Protocol Buffers files**

** Requires MapForce Professional Edition*
*** Requires MapForce Enterprise Edition*

The following table illustrates support for dynamic input and output file and wildcards in MapForce languages.

Target language	Dynamic input file name	Wildcard support for input file name	Dynamic output file name
XSLT 1.0	*	Not supported by XSLT 1.0	Not supported by XSLT 1.0
XSLT 2.0	*	*(1)	*
XSLT 3.0	*	*(1)	*
XQuery	*	*(1)	Not supported by XQuery
C++	*	*	*
C#	*	*	*
Java	*	*	*
BUILT-IN	*	*	*

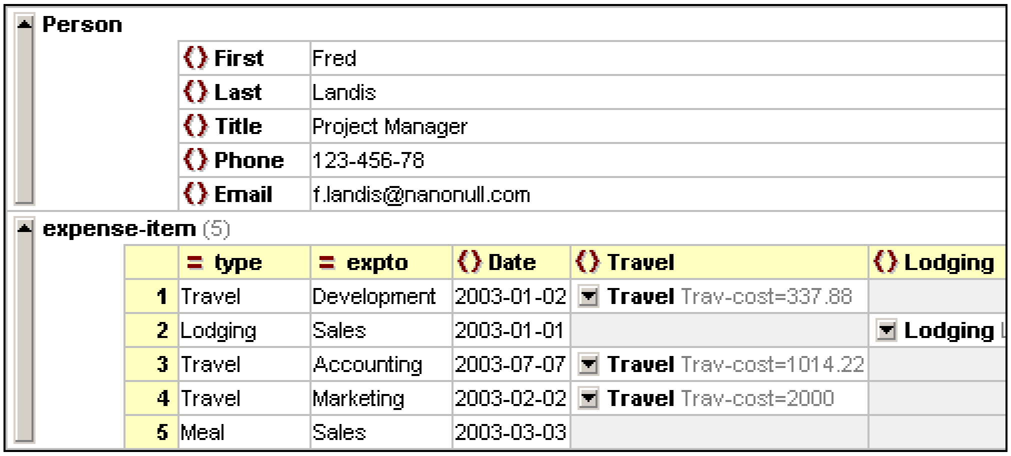
Legend:

*	Supported
(1)	XSLT 2.0, XSLT 3.0, and XQuery use the `fn:collection` function. The implementation in the Altova XSLT 2.0, XSLT 3.0, and XQuery engines resolves wildcards. Other engines may behave differently.

6.2.1 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\MapForce2025\MapForceExamples\Tutorial\Tut-ExpReport-dyn.mfd**.

The source XML file (available in the same folder as the mapping) consists of the expense report for a person called "Fred Landis" and contains five expense items of different types. The aim of the example is to generate a separate XML file for each of the expense items listed below.



Person

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

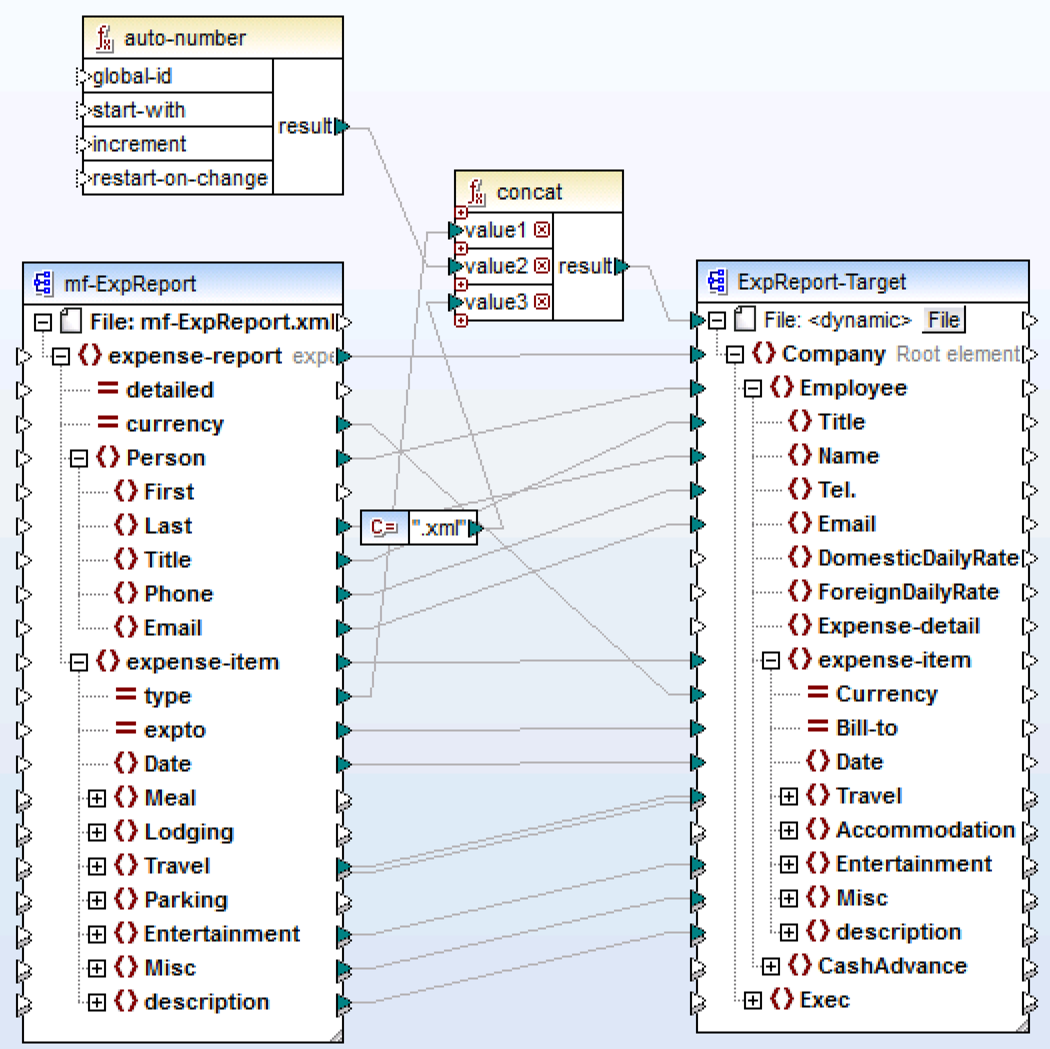
expense-item (5)

	type	**expto**	**Date**	**Travel**	**Lodging**
1	Travel	Development	2003-01-02	**Travel** Trav-cost=337.88	
2	Lodging	Sales	2003-01-01		**Lodging** L
3	Travel	Accounting	2003-07-07	**Travel** Trav-cost=1014.22	
4	Travel	Marketing	2003-02-02	**Travel** Trav-cost=2000	
5	Meal	Sales	2003-03-03		

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

As the `type` attribute defines the specific expense item type, this is the item we will use to split up the source file. To achieve the goal of this example, do the following:

1. Insert a **`concat`** function (you can drag it from the **core | string functions** library of the Libraries pane).
2. Insert a constant (on the **Insert** menu, click **Constant**) and enter ".xml" as its value.
3. Insert the **`auto-number`** function (you can drag it from the **core | generator functions** library of the Libraries pane).
4. Click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.
5. Create the connections as shown below and then click the **Output** pane to see the result of the mapping.



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

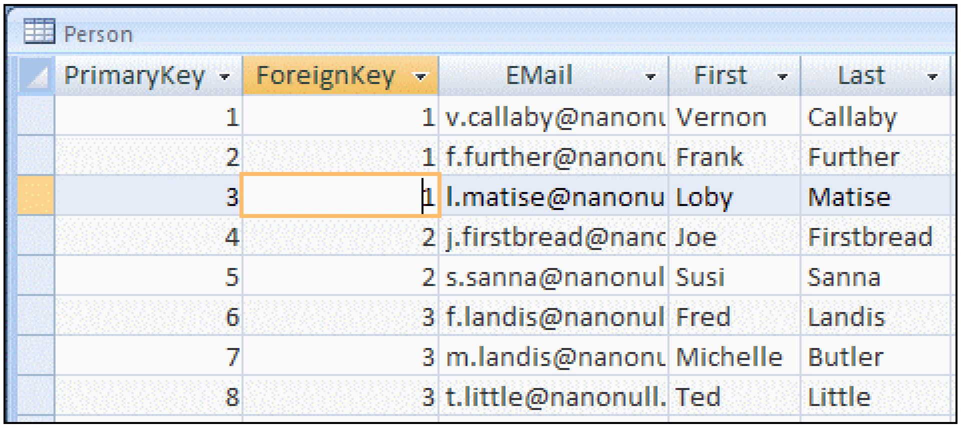
Note that the resulting output files are named dynamically as follows:

- 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.

6.2.2 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\MapForce2025\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.

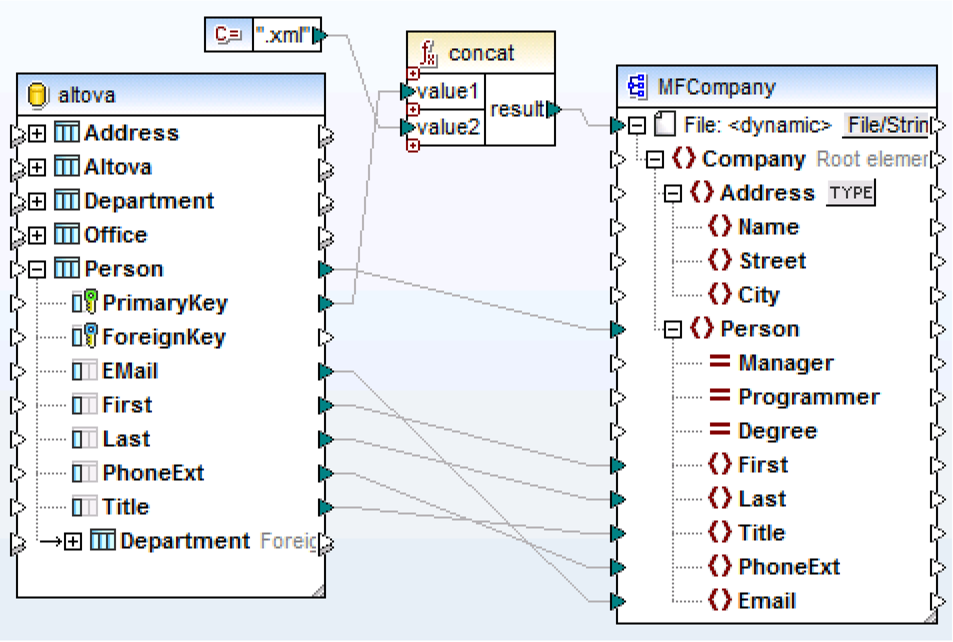


Person

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** pane 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.

6.3 Parse and Serialize 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:

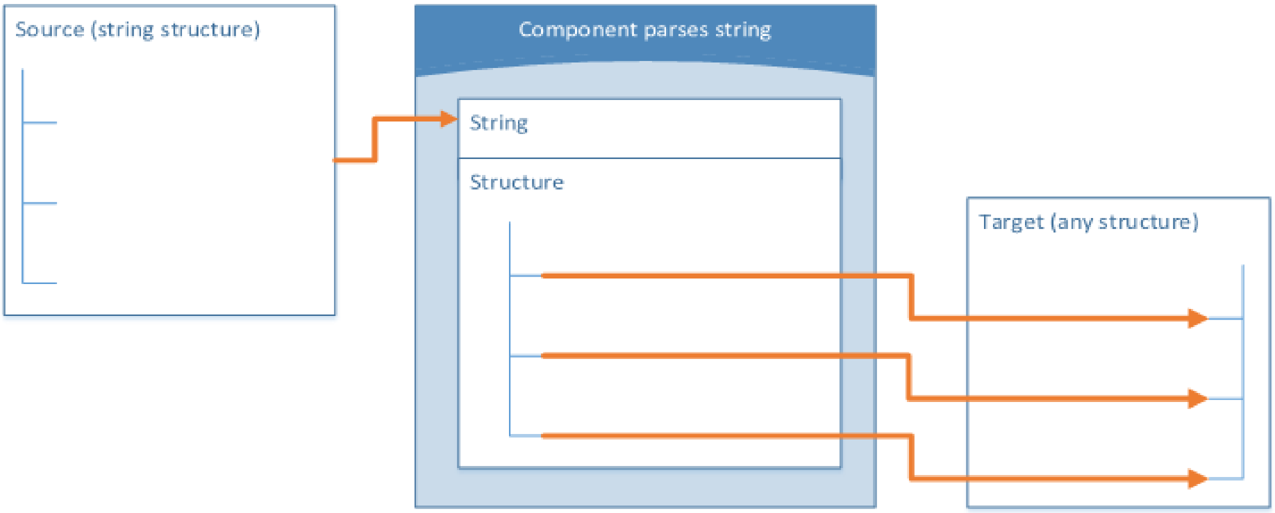
- Text (CSV, fixed-length field text)
- XML schema files

For all component types above, string parsing and serialization is supported in the BUILT-IN target language. In addition, parsing strings to JSON or serializing JSON from strings is supported in BUILT-IN, C#, and Java.

6.3.1 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.

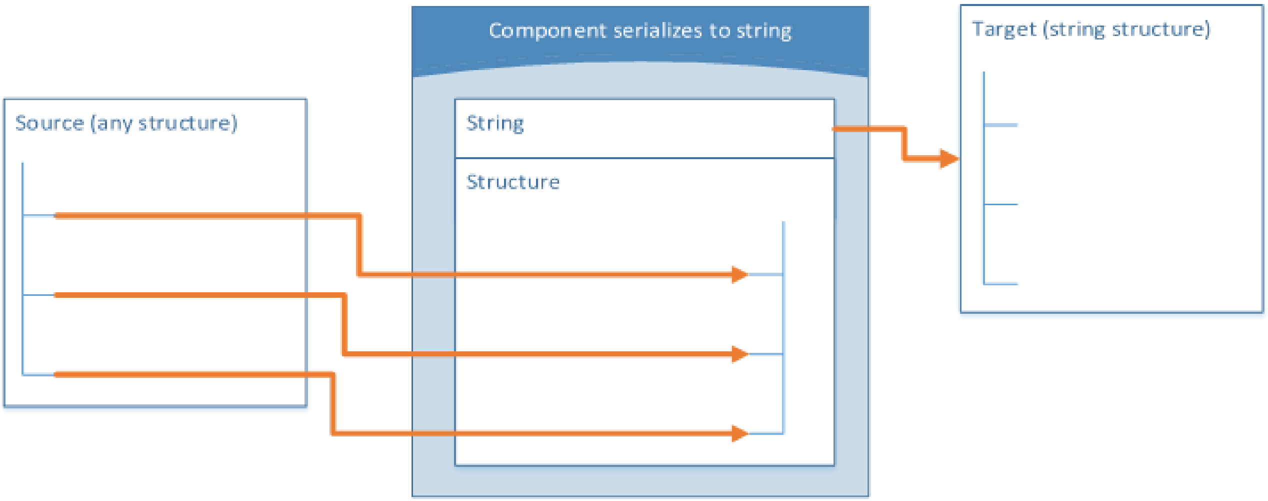
You can use a "Parse/Serialize String" component for string parsing when, for some reason, you need to convert a string that has structure (for example, some XML stored as string in a database) into another format. Parsing data from the source string to the "Parse/Serialize" component means that the 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.

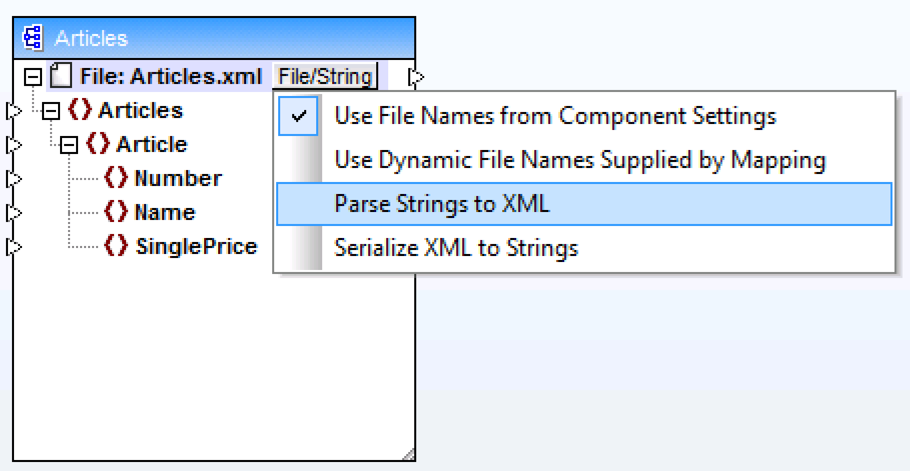
When you serialize data from a component to string, the reverse happens. Specifically, the entire structure of the MapForce component becomes a string structure which you can further manipulate as necessary. For example, this enables you to write an XML file (or XML fragment) to a database field or to 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)755.

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 Settings41). Note that not all component settings are available when a component is in either "Parse" or "Serialize" mode.

6.3.2 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\MapForce2025\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>
```

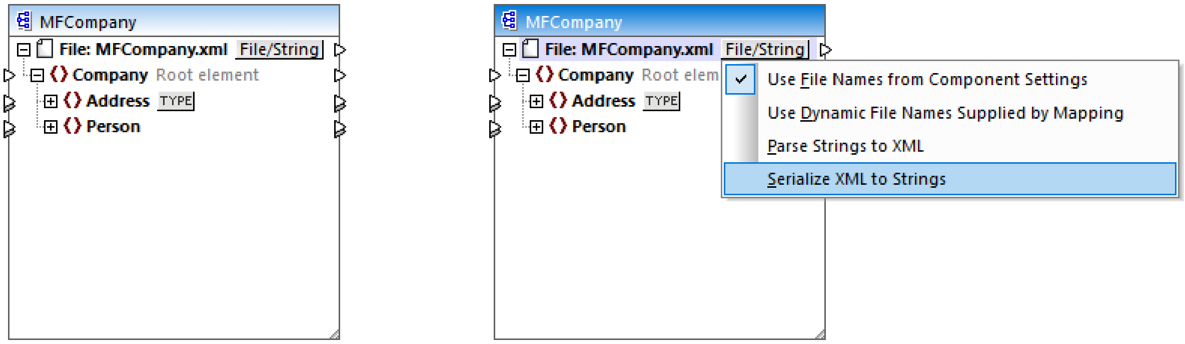
Your goal is to extract each <Person> element from the XML file and insert it literally (including XML tags) as a new database record in the PEOPLE table of a SQLite database. The PEOPLE table contains only two columns: ID and PERSON. Its full definition is as follows:

```
CREATE TABLE PEOPLE (ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, PERSON TEXT);
```

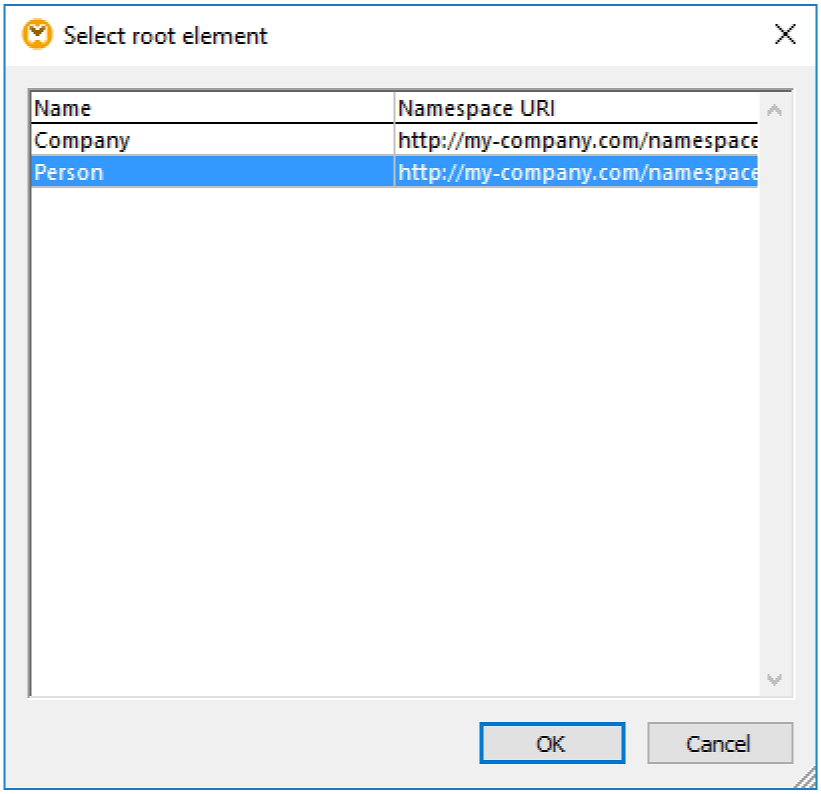
After the mapping is executed, the expected result is that the PEOPLE table will have the same number of rows as the number of <Person> elements in the XML file.

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\MapForce2025\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**.



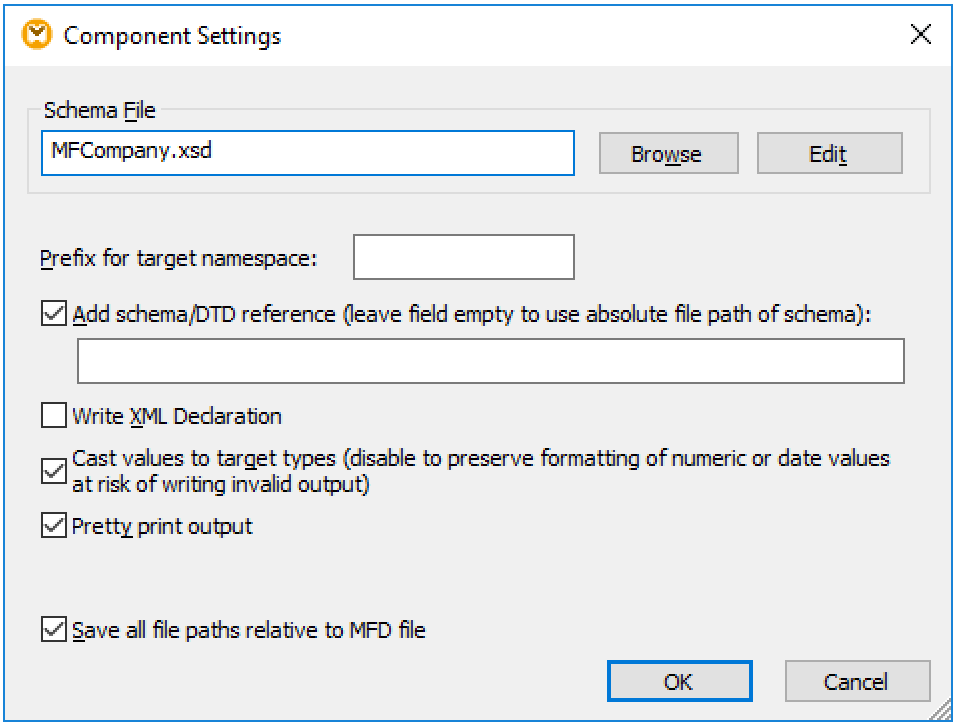
4. Right-click the duplicated component and select **Change Root Element** from the context menu. Then change the root element to <Person>.



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 clear the **Write XML Declaration** check box. 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):

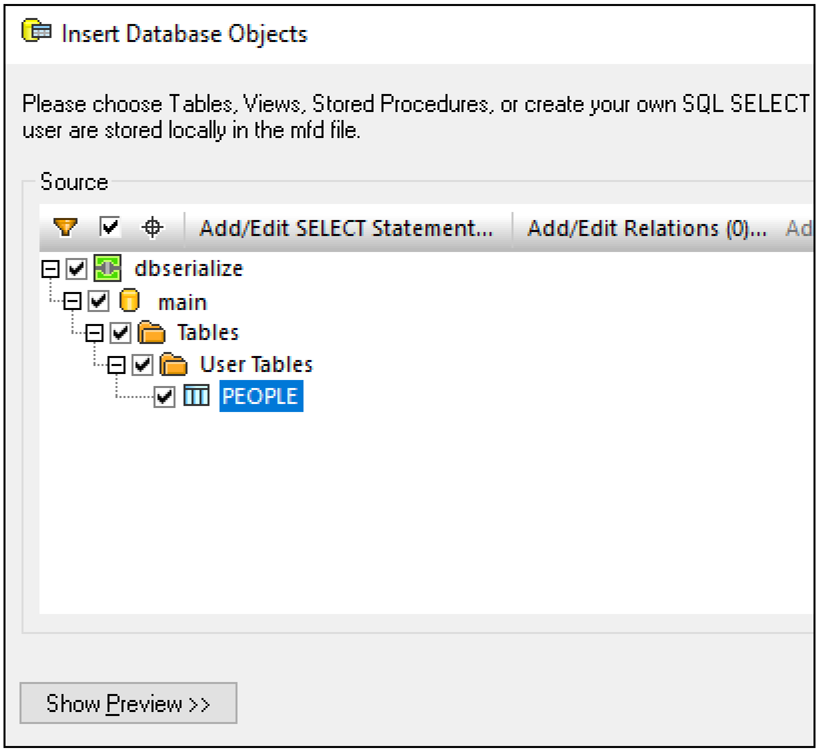
□ Write XML Declaration

☑ 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\MapForce2025\MapForceExamples\Tutorial\\dbserialize.db**. (To add the database component, use the **Insert | Database** menu command, see also Connecting to a Database 168). When prompted to insert a database object, select the PEOPLE table.



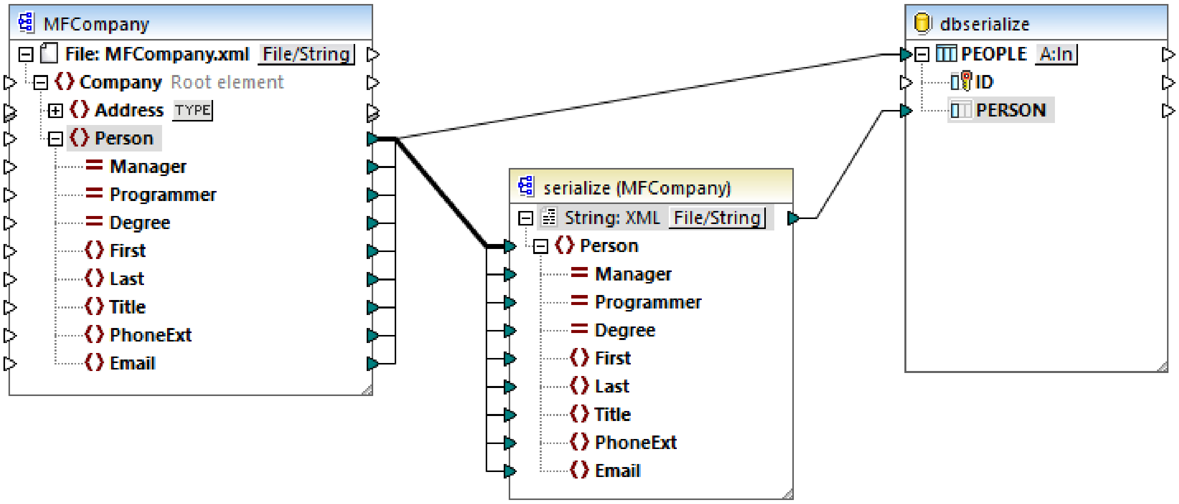
Insert Database Objects

Please choose Tables, Views, Stored Procedures, or create your own SQL SELECT user are stored locally in the mfd file.

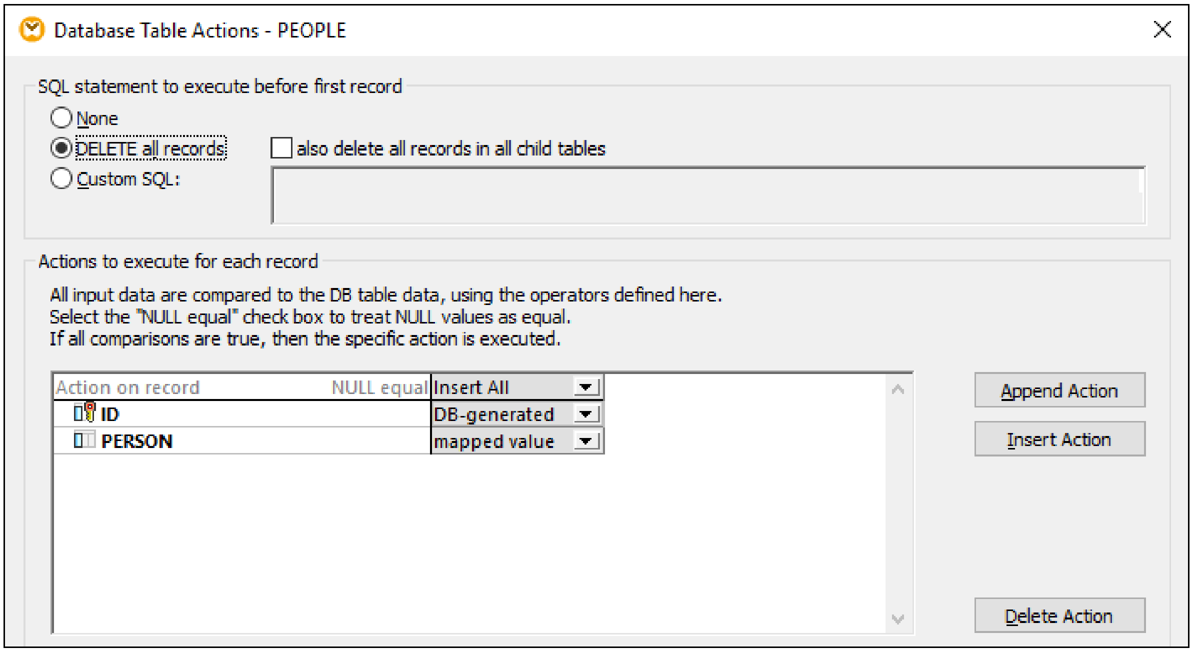
Source

☑ dbserialize
☑ main
☑ Tables
☑ User Tables
☑ PEOPLE

7. Link the components as shown below. On the left side of the mapping, the `<Person>` element maps to the serialization component. On the right side of the mapping, the serialized string value is inserted into the `PERSON` column of the `PEOPLE` database table. Finally, the connector drawn from `<Person>` to the `PEOPLE` table instructs MapForce to create a new record for each `<Person>` element encountered.



8. Click the **A:In** button on the database component, and do the following:
   a. Select the **Delete all records** option. At mapping runtime, this will delete any existing records from the database before new ones are inserted.
   b. Select the **DB-generated** option next to the **ID** column. This ensures that the ID of the record will be generated by the database. Note that the **DB-generated** option appears only if the column supports this option. For columns that are not an identity or auto-incremented field, the **max+1** option is available instead—this option will check what is the maximum value already existing in that column, and insert the next available integer, incremented by 1.


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.
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
ID		DB-generated
PERSON		mapped value

You have now created a mapping design which serializes data to string. If you click the **Output** pane, 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.

7 Mapping Documentation

You can generate detailed documentation about any mapping in HTML, Microsoft Word (.doc), or RTF format. If StyleVision is installed, you can additionally generate documentation in PDF format.

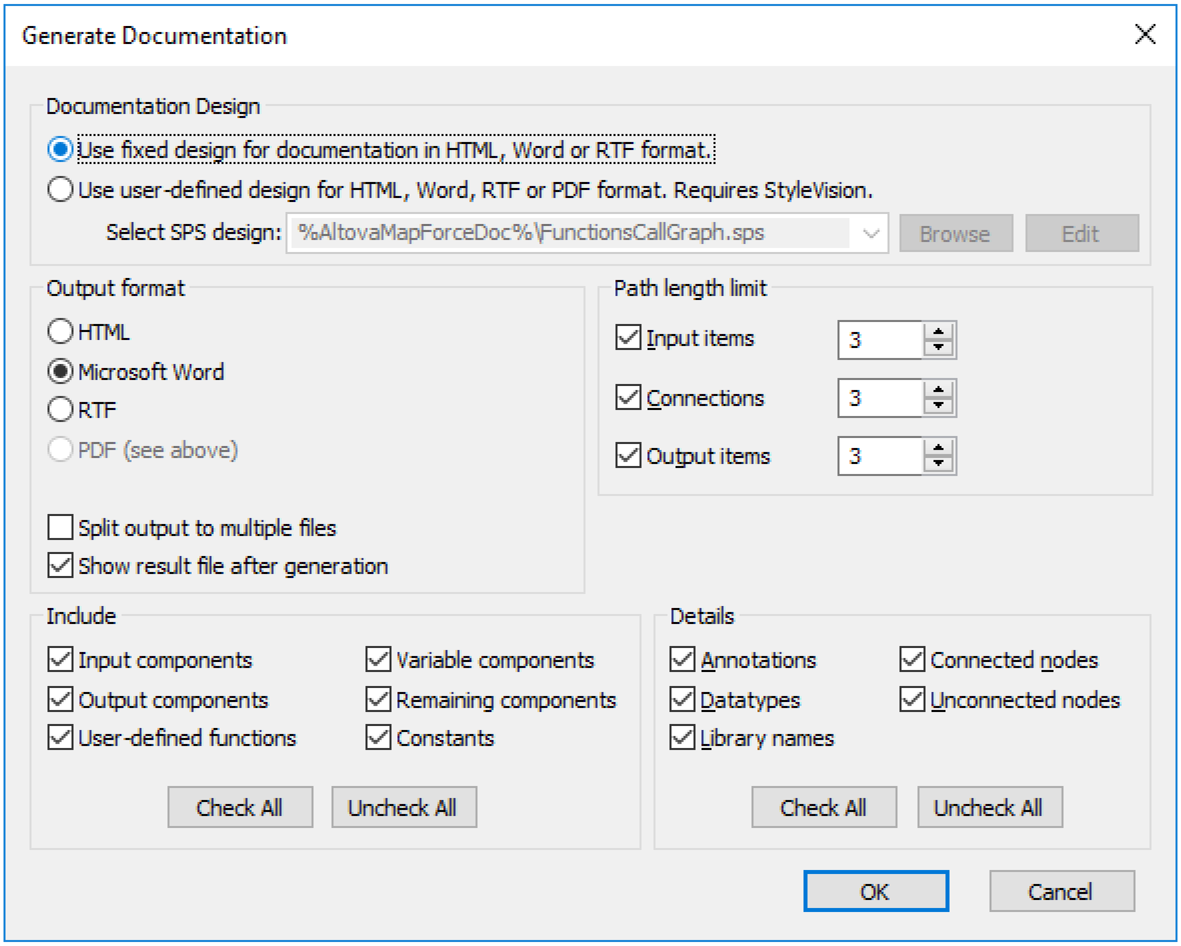
Prerequisites

- Microsoft Word 2000 or later must be installed if you would like to generate documentation in Microsoft Word format.
- StyleVision must be installed if you would like to generate documentation in PDF format or customize the design of the generated documentation.

By default, documentation is generated with a fixed design, where you can configure basic options such as the components to include, the depth of displayed paths, and other settings. If StyleVision is installed, you can additionally benefit from several included StyleVision Power Stylesheets (SPS) files, or even create your own design in StyleVision.

To generate mapping documentation:

1. On the **File** menu, click **Generate Documentation**. This opens the "Generate documentation" dialog box.



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

☑ Variable components

☑ Remaining components

☑ Constants

Details

☑ Annotations

☑ Datatypes

☑ Library names

☑ Connected nodes

☑ Unconnected nodes

2. Select the required settings and click **OK**.

The settings you can configure are described below.

Documentation Design

- Select "Use fixed design..." to use the built-in documentation template.
- Select "Use user-defined..." to use a predefined StyleVision Power Stylesheet created in StyleVision. The SPS files are available in the **...\Documents\Altova\MapForce2025\Documentation\MapForce\** folder. For details, see Predefined StyleVision Power Stylesheets 764.
- Click **Browse** to browse for a predefined SPS file.
- Click **Edit** to launch StyleVision and open the selected SPS in a StyleVision window.

Output Format

- Select one of the following output formats: HTML, Microsoft Word, RTF, or PDF. Microsoft Word documents are created with the **.doc** file extension when generated using a fixed design, and with a **.docx** file extension when generated using a StyleVision SPS. The PDF output format requires StyleVision and is available only if you selected a StyleVision SPS.
- Select **Split output to multiple files** if you would like to generate multiple documentation files, one file for each individual component such as input or output component. If using a fixed design, links between multiple documents are created automatically.

- If the **Show result file after generation** option is selected, MapForce will open the generated files in the default browser or application, as applicable.

Path length limit

Use these options to define the maximum path length to be shown for input or output items or connections. For example, with the default length **3**, an item path would be shown as **.../ShortPO/LineItems/LineItem**.

Include

Select here the specific components that should be included in the generated documentation.

Details

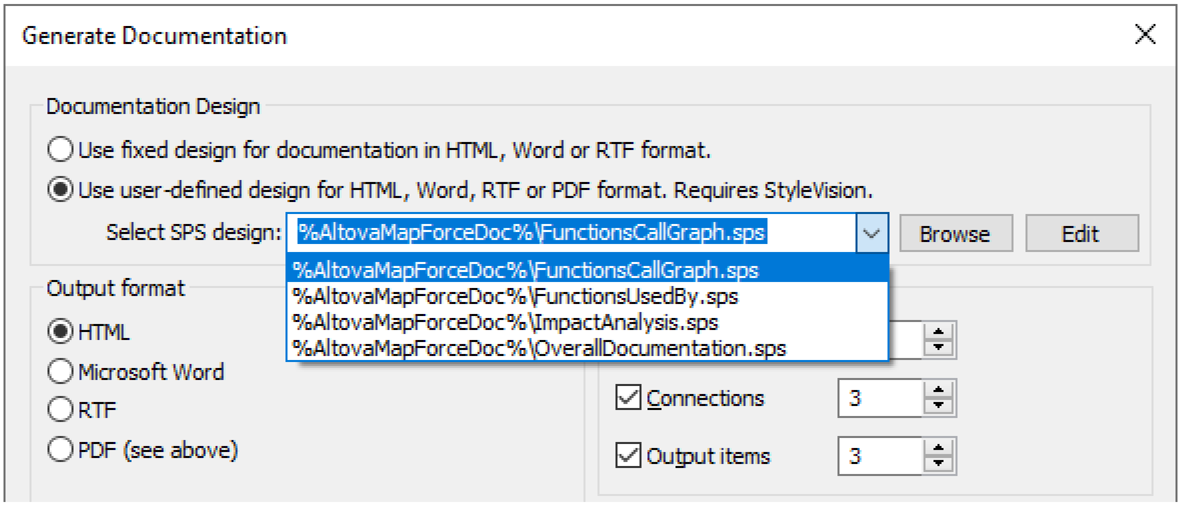
Use these options to customize the level of detail in the generated documentation. The **Library Names** option inserts the "core" prefix for functions.

7.1 Predefined StyleVision Power Stylesheets

When StyleVision is installed on your computer, you can generate mapping documentation by selecting one of the predefined StyleVision Power Stylesheet (SPS) files as template, instead of the built-in fixed design. The following predefined SPS stylesheets are available:

- **FunctionCallGraph.sps** - shows the call graph of the main mapping and any user-defined functions.
- **FunctionsUsedBy.sps** - shows which functions are used directly or indirectly in the mapping.
- **ImpactAnalysis.sps** - lists every source and target node, and the route taken via various functions to the target node.
- **OverallDocumentation.sps** - shows all nodes, connections, functions, and target nodes. This template outputs the maximum detail and is identical to the built-in "fixed design" output.

You can select the required stylesheet each time before generating documentation, as shown below. The files are located in the **...\MapForce2025\Documentation\MapForce** folder.



○ 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

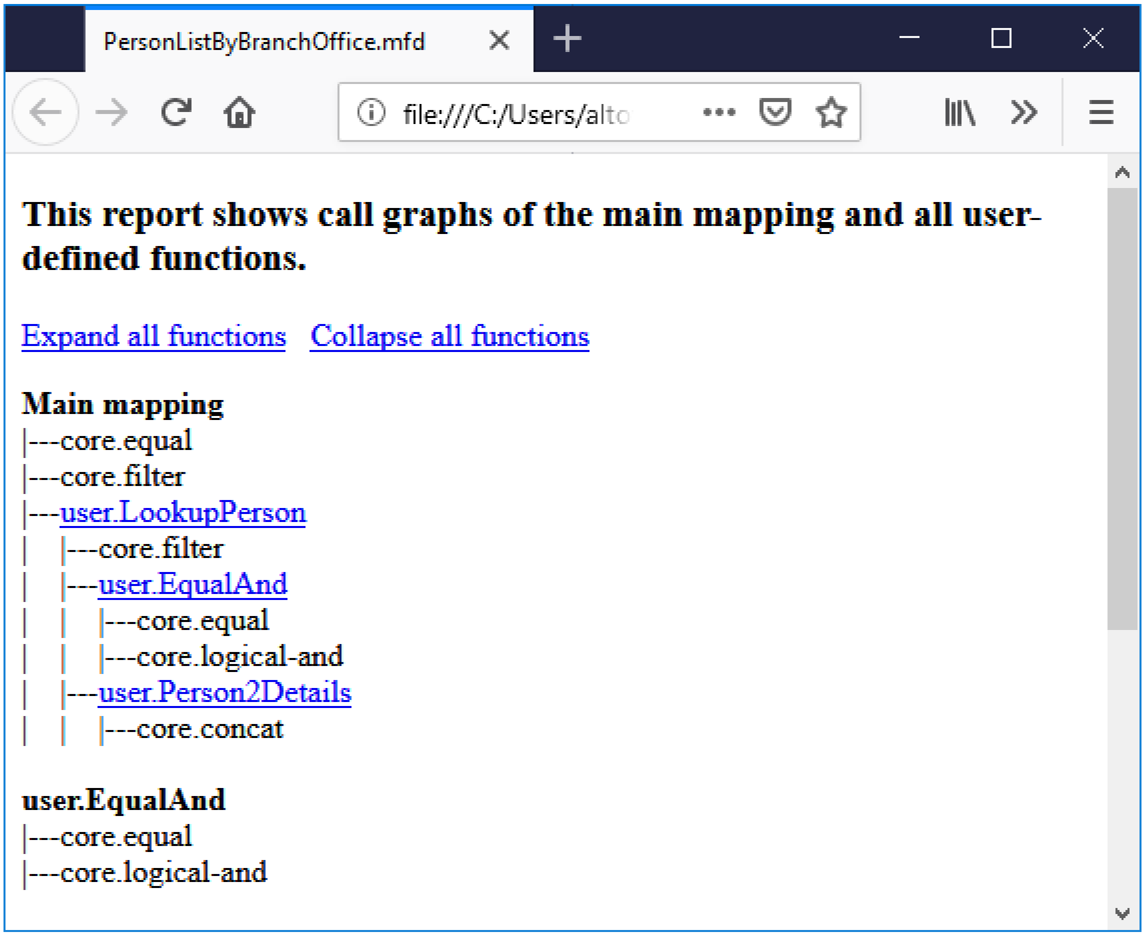
%AltovaMapForceDoc%\FunctionsCallGraph.sps
%AltovaMapForceDoc%\FunctionsUsedBy.sps
%AltovaMapForceDoc%\ImpactAnalysis.sps
%AltovaMapForceDoc%\OverallDocumentation.sps

● HTML
○ Microsoft Word
○ RTF
○ PDF (see above)

☑ Connections 3
☑ Output items 3

The examples below illustrate output produced by each of these stylesheets. The examples were generated from one of the demo mappings installed with MapForce, **PersonListByBranchOffice.mfd**. Although these examples illustrate HTML output specifically, the layout is similar with other formats. For information about creating or customizing SPS files, see Custom Stylesheets 769.

Stylesheet "FunctionCallGraph.sps"

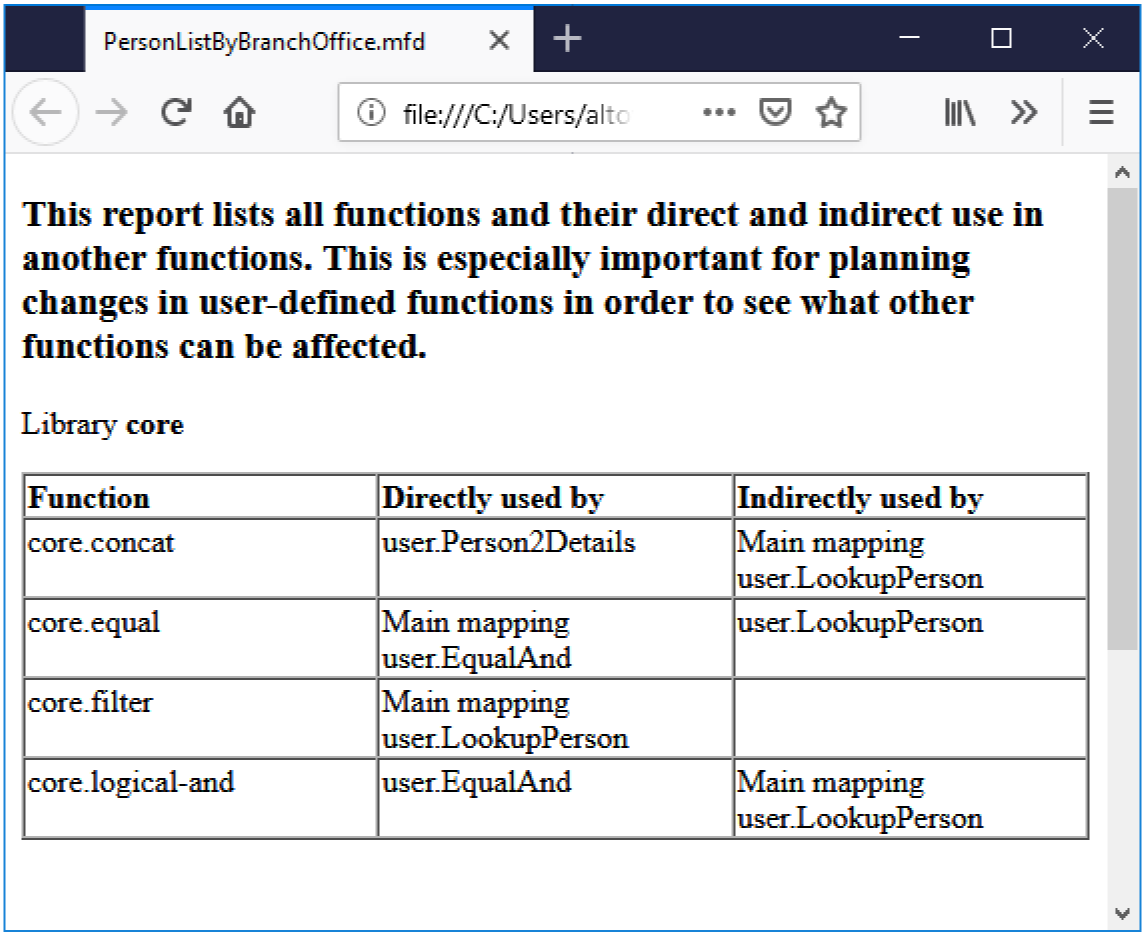


This report shows call graphs of the main mapping and all user-defined functions.

Expand 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.EqualAnd
|---core.equal
|---core.logical-and
```

Stylesheet "FunctionsUsedBy.sps"

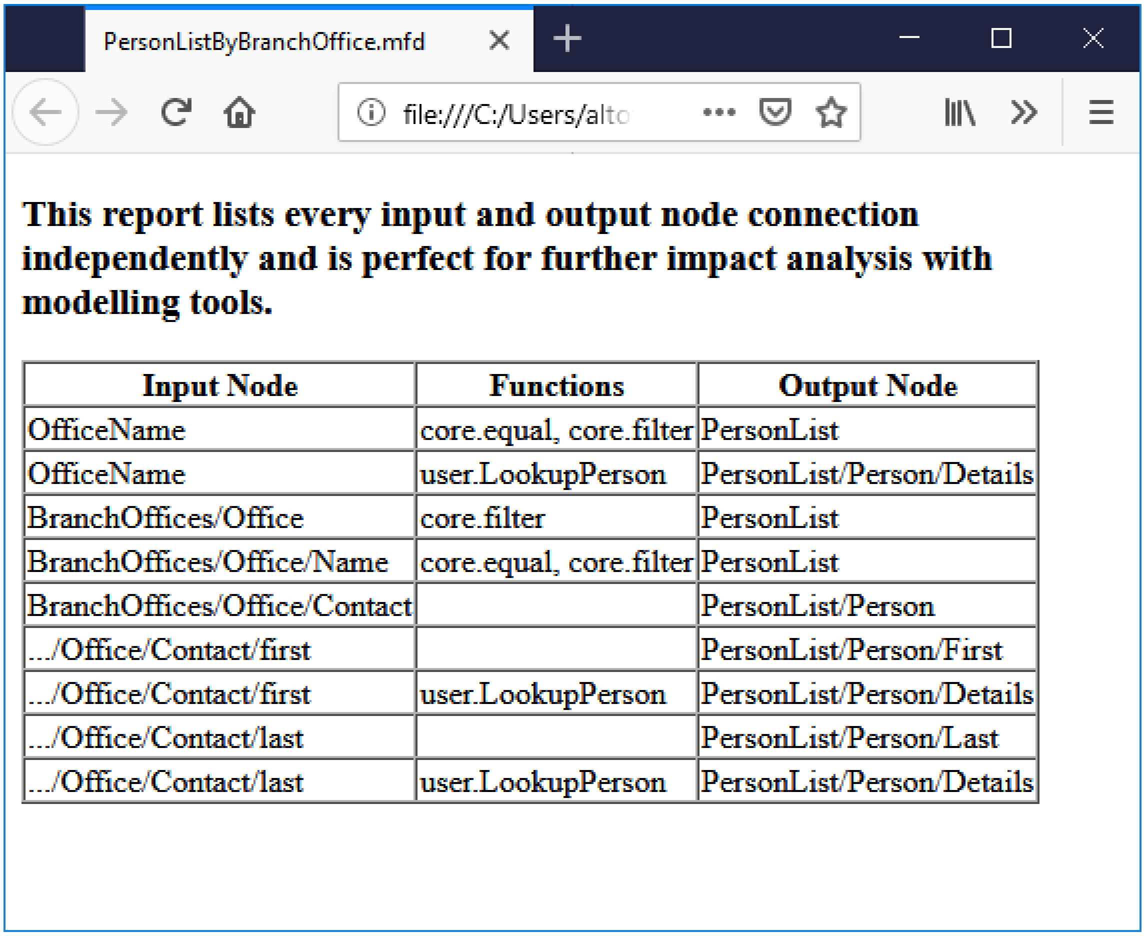


This report lists all functions and their direct and indirect use in another functions. This is especially important for planning changes in user-defined functions in order to see what other functions can be affected.

Library **core**

Function	**Directly used by**	**Indirectly used by**
core.concat	user.Person2Details	Main mapping user.LookupPerson
core.equal	Main mapping user.EqualAnd	user.LookupPerson
core.filter	Main mapping user.LookupPerson	
core.logical-and	user.EqualAnd	Main mapping user.LookupPerson

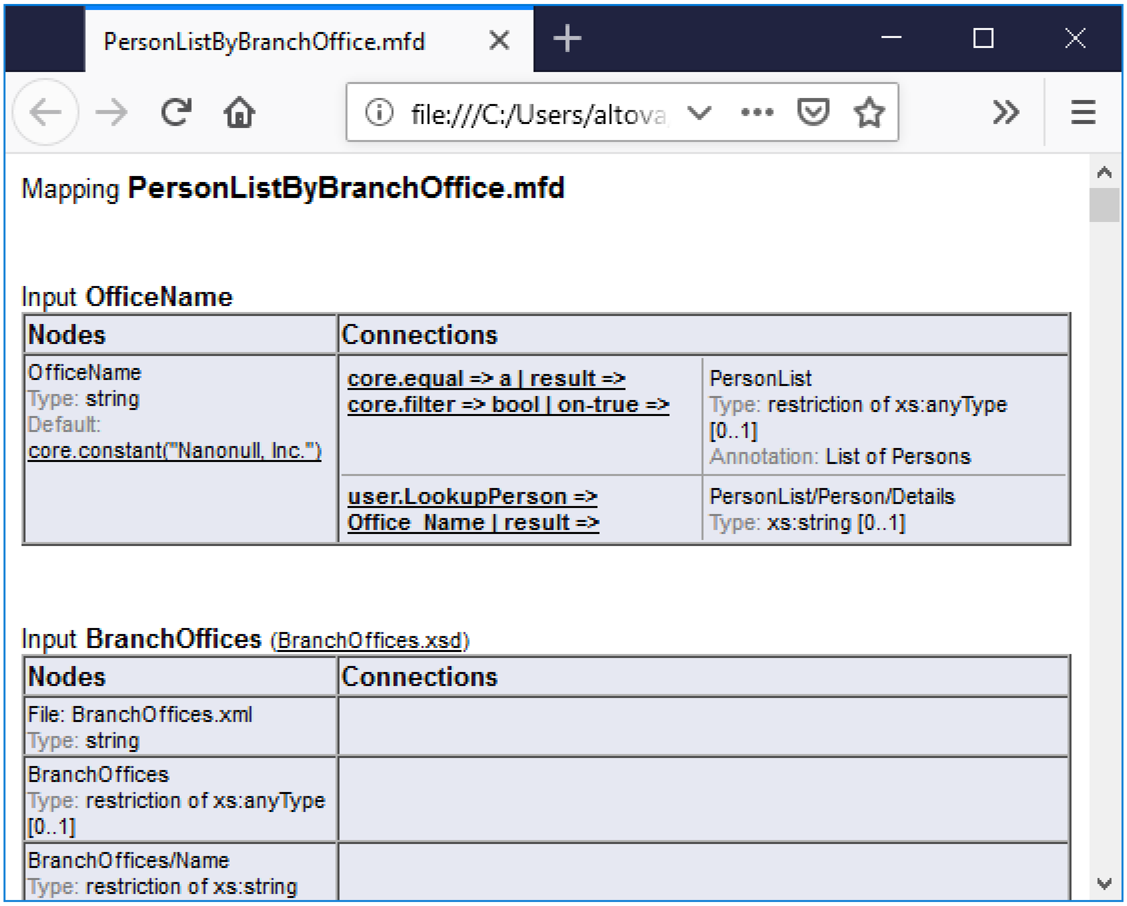
Stylesheet "ImpactAnalysis.sps"



This report lists every input and output node connection independently and is perfect for further impact analysis with modelling tools.

Input Node	Functions	Output Node
OfficeName	core.equal, core.filter	PersonList
OfficeName	user.LookupPerson	PersonList/Person/Details
BranchOffices/Office	core.filter	PersonList
BranchOffices/Office/Name	core.equal, core.filter	PersonList
BranchOffices/Office/Contact		PersonList/Person
.../Office/Contact/first		PersonList/Person/First
.../Office/Contact/first	user.LookupPerson	PersonList/Person/Details
.../Office/Contact/last		PersonList/Person/Last
.../Office/Contact/last	user.LookupPerson	PersonList/Person/Details

Stylesheet "OverallDocumentation.sps"



Mapping **PersonListByBranchOffice.mfd**

Input **OfficeName**

Nodes	Connections	
OfficeName Type: string Default: core.constant("Nanonull, Inc.")	**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	

7.2 Custom Stylesheets

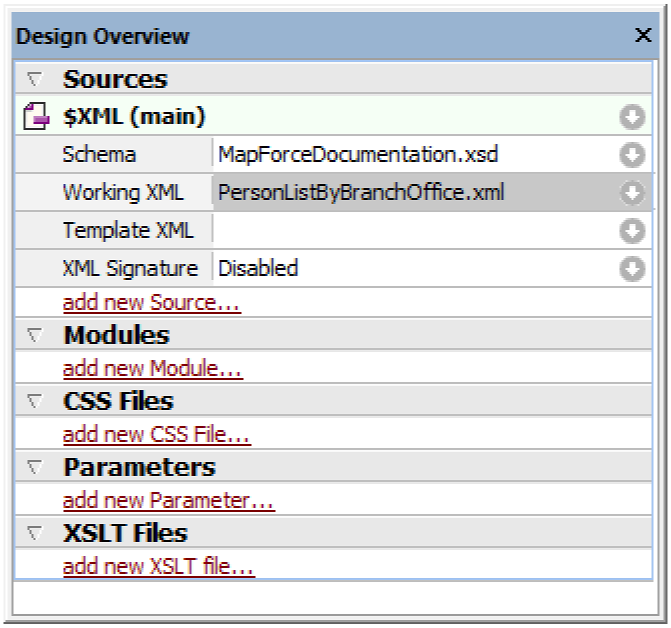
In addition to the built-in fixed design, you can create custom stylesheets for the generated mapping documentation with StyleVision (https://www.altova.com/stylevision). You can also change any of the predefined stylesheets 764, for example, by adjusting the fonts and other styles.

A custom design is a StyleVision Power StyleSheet (SPS). The advantage of using an SPS for generating mapping documentation is that you have complete control over the design of the documentation.

To create a custom SPS file, the following is required:

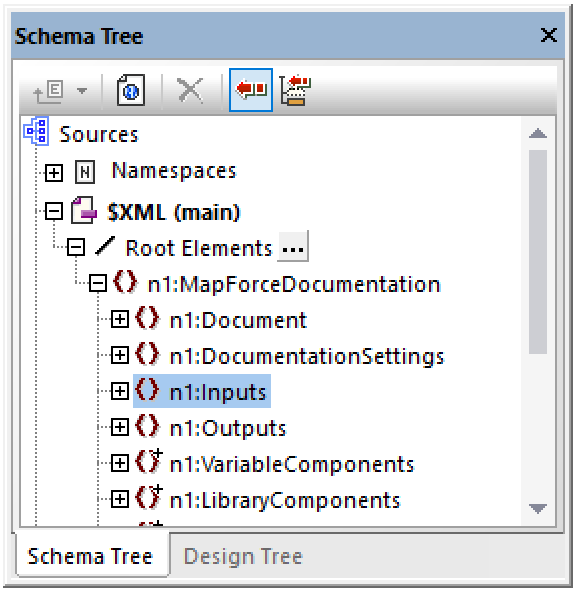
1. The XML Schema that provides the structure of the generated MapForce documentation. This schema is called **MapForceDocumentation.xsd** and is delivered with your MapForce installation package. It is stored in the **...\Documents\Altova\MapForce2025\Documentation\MapForce** folder. Note that the **MapForceDocumentation.xsd** includes the **Documentation.xsd** file located in the folder above it.
2. Some sample data to test and preview the custom design. You can use the following XML file as sample data : **...\Documents\Altova\MapForce2025\Documentation\MapForce\SampleData\PersonListByBranchOffice.xml**.

The files mentioned above must be referenced in the Design Overview window in StyleVision, for example:



Schema	MapForceDocumentation.xsd
Working XML	PersonListByBranchOffice.xml
Template XML	
XML Signature	Disabled

In StyleVision, you create a design by dragging nodes from the Schema Tree window onto the design area and assigning styles and properties to them.



You can also add additional components such as links and images to the SPS design. To preview the design in a specific format, click any of the following tabs: **HTML**, **RTF**, **PDF**, or **Word 2007+**. For more details, refer to StyleVision documentation (https://www.altova.com/documentation).

8 Debugger

MapForce includes a mapping debugger available for the MapForce BUILT-IN transformation language. The mapping debugger helps you achieve the following goals:

- View and analyze the values produced by the mapping at each individual connector level.
- Highlight on the mapping the context (set of nodes) responsible for producing a particular value.
- Execute a mapping step-by-step, in order to see how MapForce processes or computes each value in real time, and preview the mapping output as it is being generated.
- Set milestones (breakpoints) at which the mapping execution should stop and display the value(s) currently being processed.
- View the history of values processed by a connector since mapping execution began up until the current execution position.

The mapping debugger is available when the transformation language of the mapping is BUILT-IN. If you start debugging a mapping designed for a different language, you will be prompted to change the mapping language to BUILT-IN. You can also convert a mapping to BUILT-IN by selecting the menu command **Output | Built-in Execution Engine**. In either case, the conversion to BUILT-IN will be successful if the mapping does not include components that are not available in the BUILT-IN language (for example, XSLT functions).

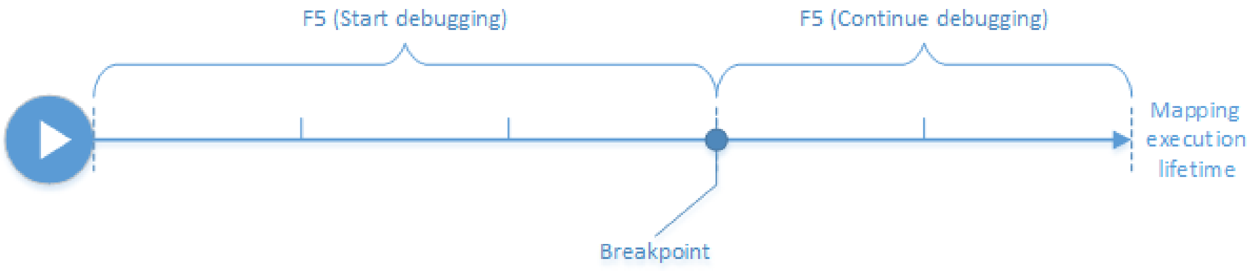
The MapForce debugger is unlike a traditional debugger in that it does not traverse your program code line by line (since you do not write any code with MapForce). Instead, the debugger exposes the results of MapForce-generated code produced from the mappings you design. More specifically, the debugger logs values that are passed from and to mapping components through their input and output connectors. The logged values are then available for your analysis directly on the mapping or through dedicated windows.

The following sections highlight various ways in which you can use the mapping debugger.

Debugger settings are available in the Options[1079] dialog box. The list of available debugging commands is available in Debug[1070].

Debug with breakpoints

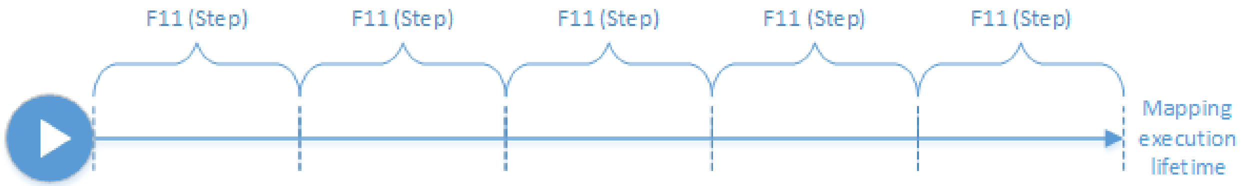
When you need to stop the debugging execution at a particular place in the mapping, you can set breakpoints, similar to how you would do that in a traditional development environment. The difference is that breakpoints are added not to a line of code, but to an input or output connector of a mapping 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.

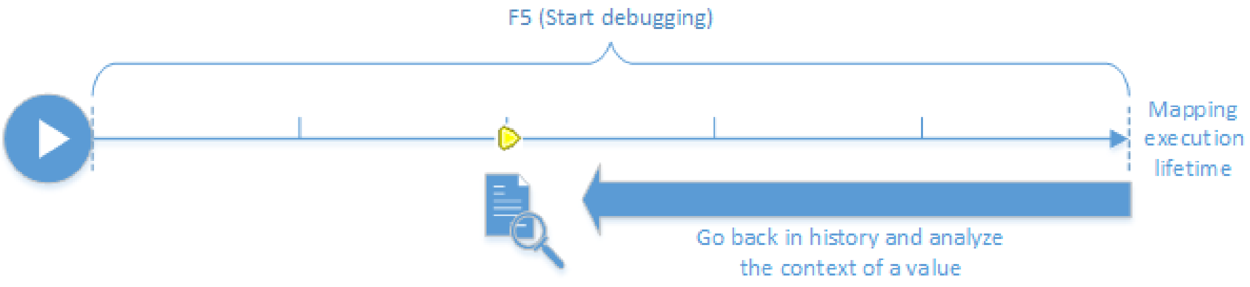
Debug step-by-step

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.



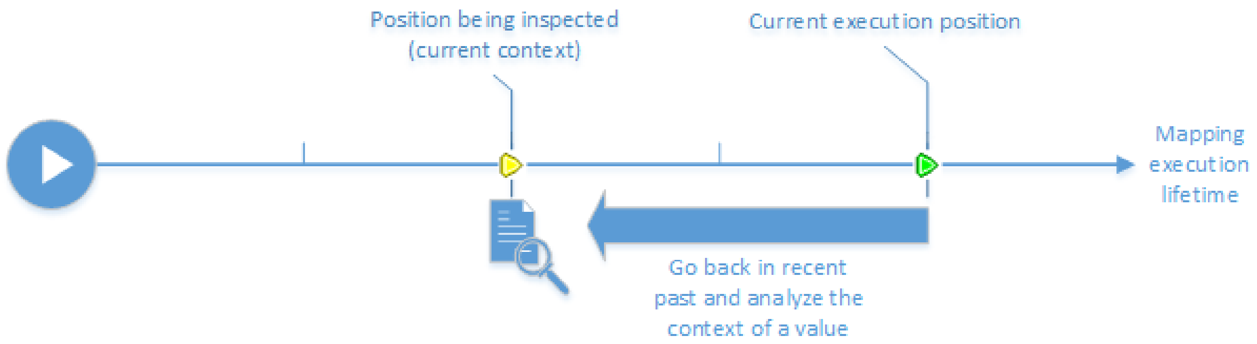
Analyze the log of values

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

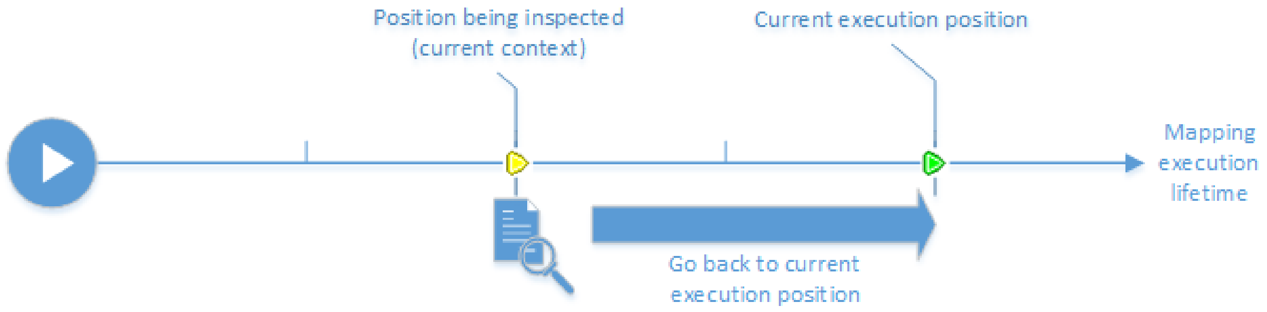
Set the context to a value related to the current execution position

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.

8.1 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[1079])

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 chained[115], 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[778]).

8.2 About the Debug Mode

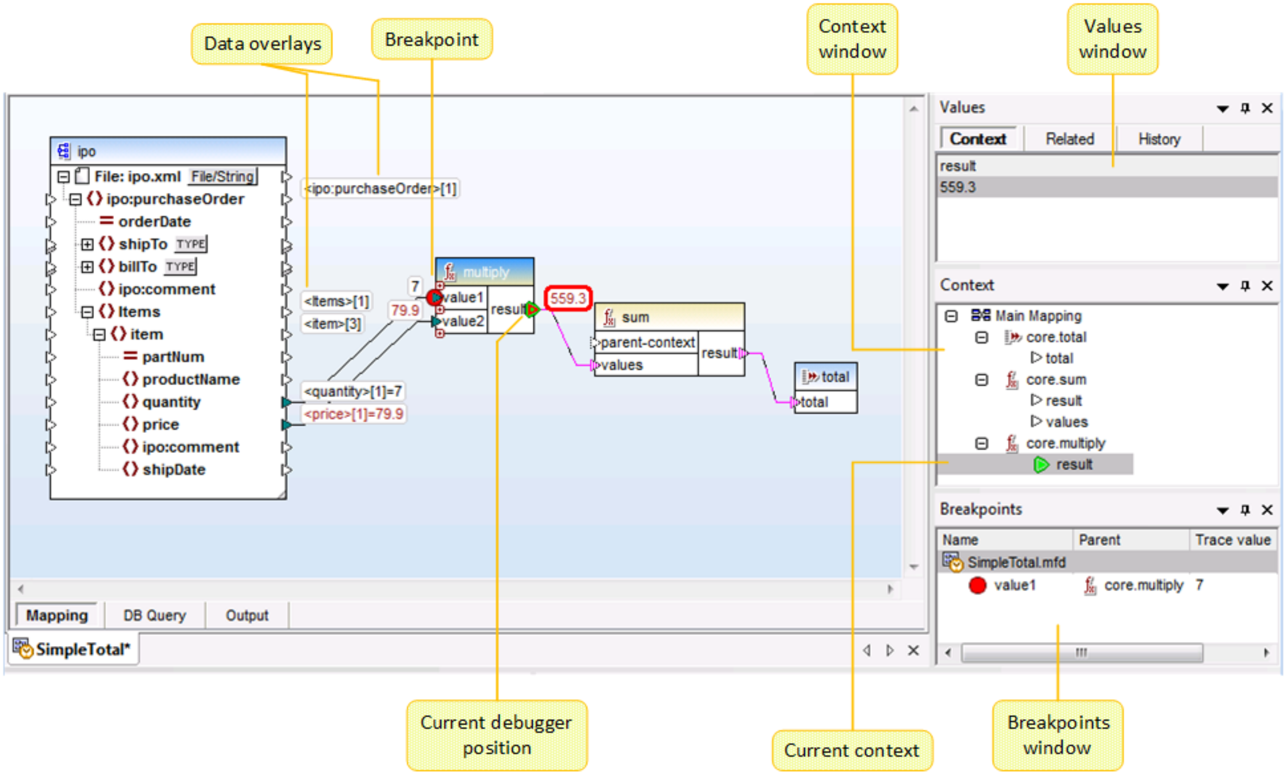
When you start debugging (by pressing **F5**, or **F11**, or **Ctrl + F11**), MapForce executes the mapping in debug mode.

> While MapForce is in debug mode, the mapping is read-only. Although you can move components on the mapping area, most commands are not available. This includes commands such as mapping validation and deployment, code generation, documenting mappings, adding new components to the mapping area or reloading existing ones, and others.

The debug mode enables you to analyze the context responsible for producing a particular value. This information is available directly on the mapping, as well as in the Values, Context, and Breakpoints windows. By default, these windows are displayed when you start debugging and are hidden when you stop debugging.

MapForce is in debug mode (and the mapping is read-only) until you stop debugging, by pressing **Shift + F5** (or by clicking the **Stop debugging** 🟥 toolbar button).

The following image illustrates a sample mapping (**SimpleTotal.mfd**, from the **<Documents>\Altova\MapForce2025\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:
  - o Connectors in the context are striped magenta ( ).
  - o Connectors in ambiguous context are dotted magenta ( ).
  - o Connections in the context are striped magenta.
  - o 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 778.

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 782).

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[787]).
▷	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[790]).

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[780].

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[782].

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[784].

8.3 Adding and Removing Breakpoints

Breakpoints are designated milestones at which the mapping should break during execution in debug mode. Any breakpoints you create are stored globally for all mappings and are displayed in the Breakpoints window. Breakpoints are valid until you either explicitly delete them, or close MapForce.

Note: Breakpoints cannot be added on any of the following entities: constants, the `core | position` function, descendent items of "Copy-all" connections, parameters of "inline" user-defined functions.

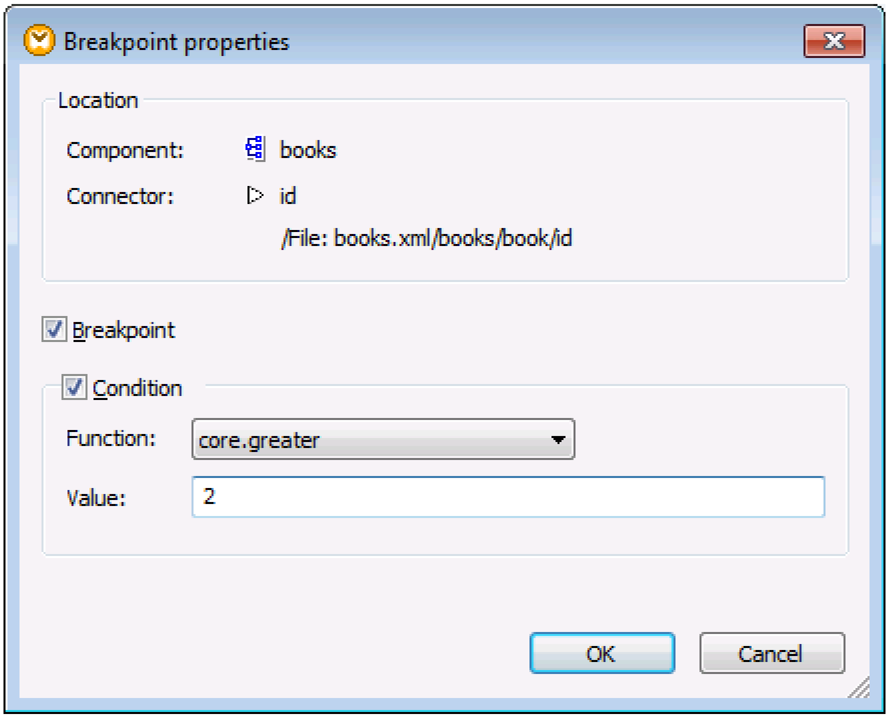
Breakpoints can be simple or conditional. Simple breakpoints stop the mapping execution unconditionally. Conditional breakpoints stop the mapping execution only when the condition assigned to them is satisfied. Conditions take the form of MapForce built-in library functions to which you supply custom values. In other words, if the condition returns true, the breakpoint will stop the mapping execution.

To create a simple breakpoint, do one of the following:

- Right-click an input or output connector (the small triangles to the left or right of a component), and select **Debugger Breakpoint**.
- Click an input or output connector, and then press **F9**.

To create a conditional breakpoint:

1. Right-click a connector, and select **Breakpoint properties**.



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.

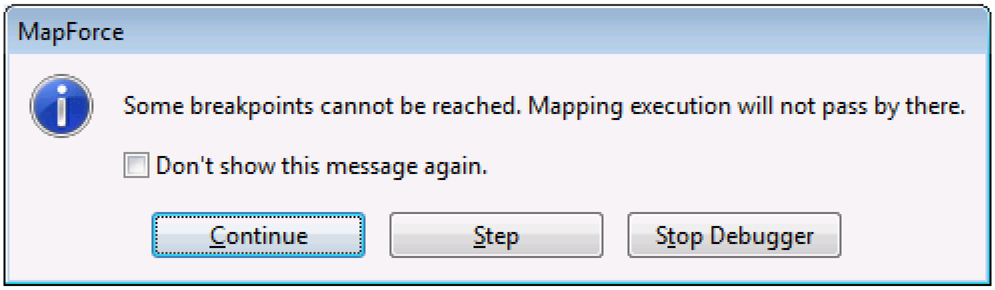
Removing breakpoints

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 784).

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 773).

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.

8.4 Using the Values Window

The Values window displays information about the values processed by the mapping when in debug mode. The information displayed in the Values window depends on the current debugger position, and on the user interface elements that you clicked. The Values window contains the following tabs:

The "Context" tab

The **Context** tab displays the value currently being processed (the same value whose context is shown in the Context window). This is either the value at the current execution position of the debugger, or the value of a connector processed in the past. MapForce helps you distinguish between the two using colors:

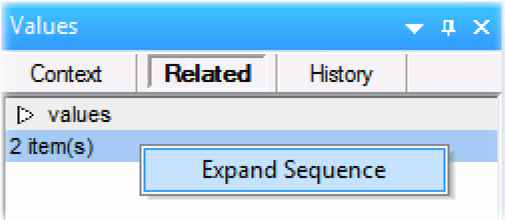
▷	Green is "the present"; it indicates the current execution position (see Viewing the Current Value of a Connector 787).
▷	Yellow is "the past"; it indicates that you are looking at some connector in the past, relative to the current execution position. This may happen after you set a context manually (see Setting the Context to a Value 790).

The "Related" tab

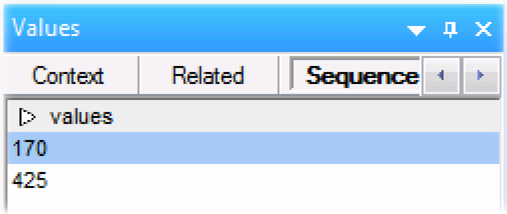
The **Related** tab displays values that are related to (or represent the "near past" of) the currently processed value. Normally, you do need to explicitly click this tab; MapForce switches to it automatically when you click the data overlay of a connector that is related to the current execution position of the debugger. See Stepping back into Recent Past 788.

The "Sequence" tab

When present, the **Sequence** tab enables you to get access to the values of a connector that processes a sequence. This tab is visible only when a connector has processed a sequence of items (for example, an aggregate function such as `sum` or `count` does that). When you click the data overlay of a connector that processed a sequence of items, the Values window displays an entry in the format "**n items**", where **n** is the number of items processed by the connector. To get access to each value, double-click this entry (or right-click it, and select **Expand Sequence** from the context menu).



The values are then displayed in the **Sequence** tab.



The "History" tab

The **History** tab displays values have been processed by a particular node since debugging started and up to the current execution position. See Viewing the History of Values Processed by a Connector 789.

8.5 Using the Context Window

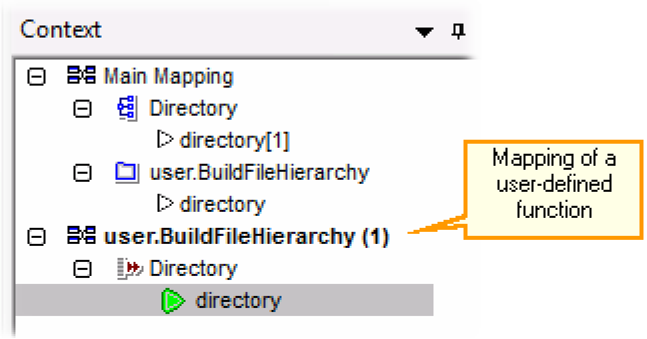
While MapForce is in debug mode, the Context window displays a structure of connectors that are relevant to the current position of the debugger. In other words, it provides the mapping context responsible for producing the current mapping value.

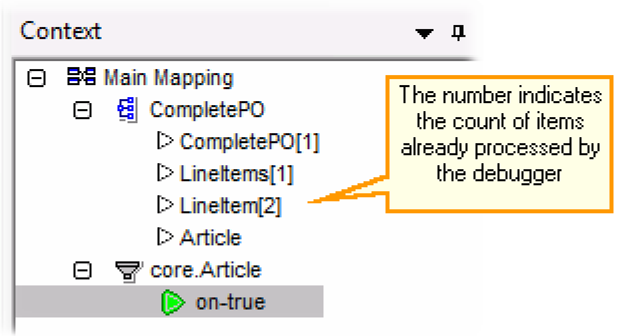
MapForce builds the current context as follows:

1. Start with the root node of the target structure.
2. Descend to the current target node.
3. From the current target node, move left inside the mapping through any components that lead to the current position. These components may be filter or sort components, built-in or user-defined functions, variables, and so on.

The Context window serves both as informational and as a navigational aid. To select a particular node in the mapping directly from the current context, right-click the node in the Context window, and click **Select in mapping**. This might be especially useful when the mapping is large, so as to avoid extensive scrolling.

The Context window may display the following special icons and notation:

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 790).

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:

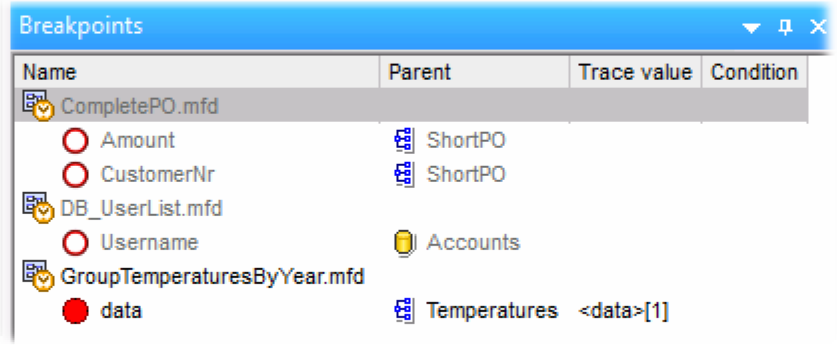
1. From the target to the output connector of the user-defined function to the input connector of the user-defined function
2. From there further to the left.

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.

8.6 Using the Breakpoints Window

The Breakpoints window enables you to view and manage breakpoints globally. By default, the Breakpoints window is displayed when MapForce is in debug mode. If you want to make the Breakpoints window visible at all times, select the menu command **View | Debug Windows | Breakpoints** when MapForce is not in debug mode.

The Breakpoints window displays all breakpoints created since you started MapForce, grouped by the mapping file to which they belong. While MapForce is open, any breakpoints associated with any mapping are "remembered" by MapForce and displayed in the Breakpoints window, even if you closed the mapping file in the meanwhile. The mapping that is currently being debugged is represented with standard text color in the Breakpoints window, while other mappings (the ones that are closed or not active) are grayed out.



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.
○	Inactive breakpoint. Denotes a breakpoint from a mapping that is open, but is not currently being debugged.

Icon	Description
⚠	Inaccessible breakpoint. Denotes a breakpoint that cannot be reached by the debugger.
⊕	Conditional breakpoint. Denotes a breakpoint with a condition attached to it.

To view or change the properties of a breakpoint:

- Right-click it, and select **Breakpoint Properties** from the context menu.

To delete a breakpoint:

- Right-click the breakpoint you want to delete, and then select **Delete Breakpoint** from the context menu.
- Click a breakpoint, and then press **Delete**.

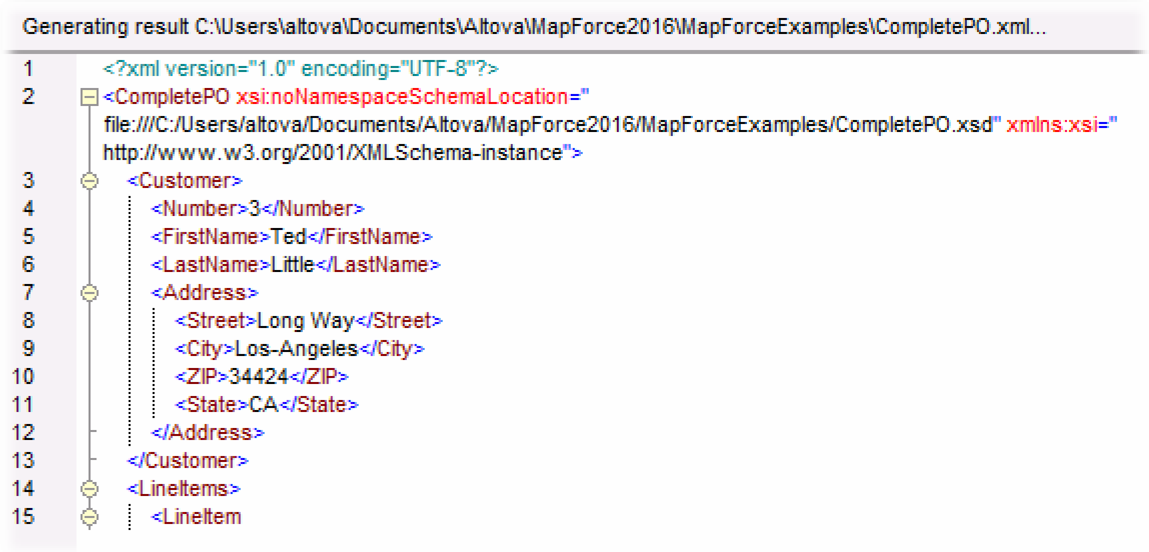
The context command **Delete All Breakpoints** removes all breakpoints displayed in the Breakpoints window, regardless of the mapping where they belong.

See also: Adding and Removing Breakpoints 778

8.7 Previewing Partially Generated Output

When you are debugging in steps or using breakpoints, you can view the mapping output generated up to the current debugger position. Previewing partially generated output is supported by XML, flat text, and EDI target components.

By default, when you press **F5** (without having defined any breakpoints), MapForce executes the entire mapping in debug mode, and then switches to the **Output** pane, 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** pane, 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
```

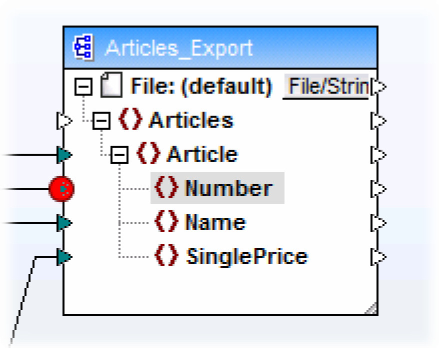
Limitations

- The currently computed target node is not always displayed in the Output pane. 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.

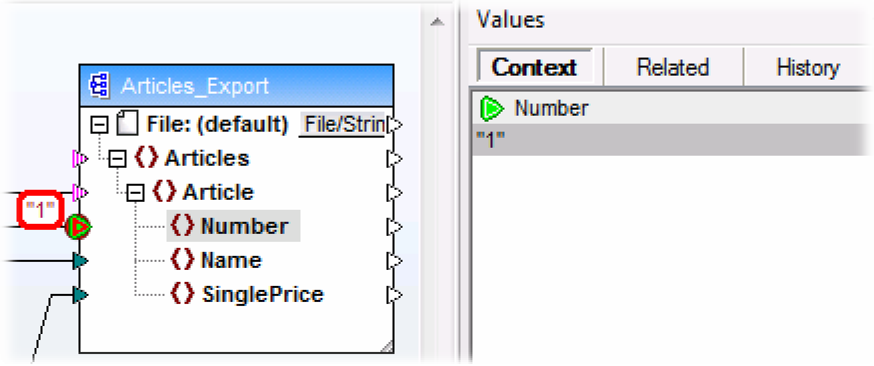
8.8 Viewing the Current Value of a Connector

When the current execution position of the debugger (▷) is on a particular connector (either because you are debugging in steps, or because there is a breakpoint defined on the connector), the current value processed by the connector is displayed in the **Context** tab of the Values window. This is the value that is about to be written to the output, that is, "the present". It is also the value whose context is displayed in the Context window (see Using the Context Window 782).

To understand this case, open the **PreserveFormatting.mfd** sample from the **<Documents>\Altova\MapForce2025\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.



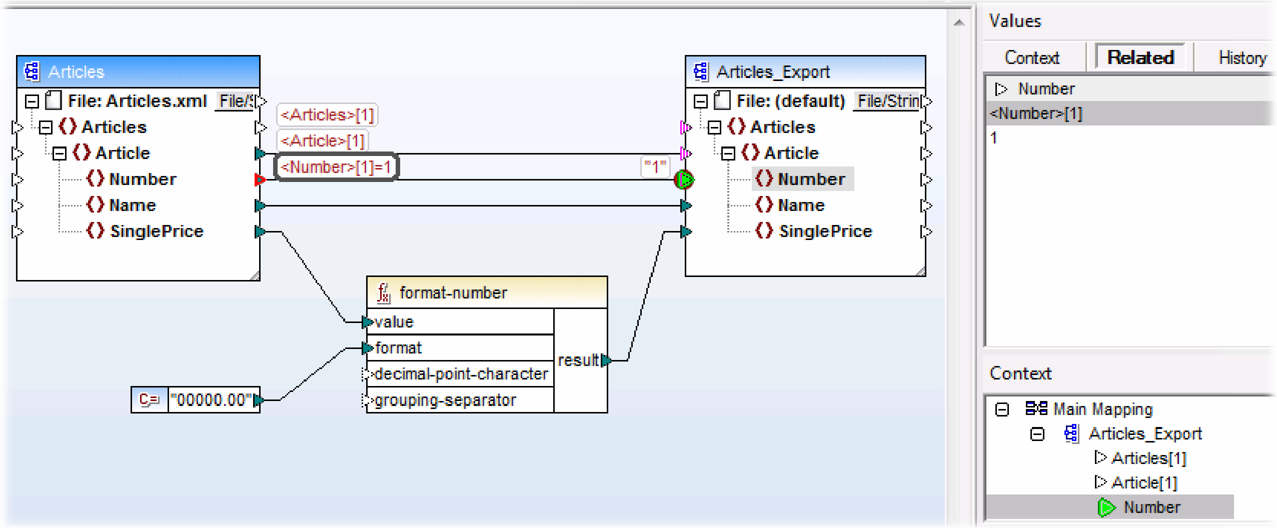
As shown in the image, the current debugger position ▷ (and the breakpoint ●) is on the `Number` node of the target component. The Values window indicates that this node processes the value "1" (this value is also highlighted with a thick red border on the mapping).

8.9 Stepping back into Recent Past

When you click a data overlay (small rectangular box) next to a mapping connector, the **Values** window displays the name and, optionally, the value associated with the selected connector. The focus now is no longer on the current debugger position, but on the selected data overlay. You can consider this view as stepping slightly back in the debugging history. This is the "near" past, since the mapping displays data overlays only for the last few connectors related to the current debugger position. When you click such a "related" data overlay, the Values window switches automatically to the **Related** tab.

For an illustration of this scenario, open the mapping **PreserveFormatting.mfd** from the **<Documents>\Altova\MapForce2025\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 790).

8.10 Viewing the History of Values Processed by a Connector

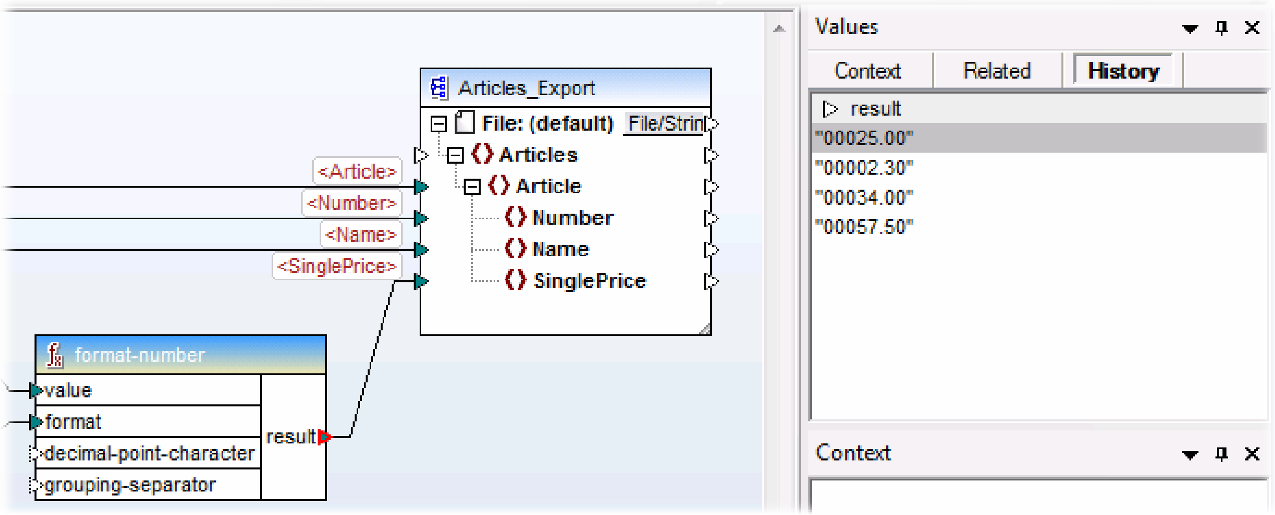
If the option **Keep full trace history** is enabled (see Debugger Settings 1079), you can view the history of all values that were processed by that connector (up to the current execution position).

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\MapForce2025\MapForceExamples\** directory. If it is already open, make sure to do the following:

- Clear any breakpoints, if such exist (see Adding and Removing Breakpoints 778)
- 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** pane. Click the **Mapping** pane 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.

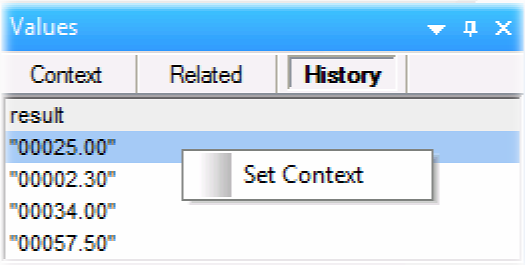


```
result
"00025.00"
"00002.30"
"00034.00"
"00057.50"
```

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 790).

8.11 Setting the Context to a Value

Setting the context to a value is an action that can be compared to stepping into the past, in order to view more details about the mapping context that produced that value. You can set the context to any value displayed in the Values window (in the **Related** tab, **Sequence** tab, or **History** tab). If you have enabled the **Keep full trace history** option (see Debugger Settings[1079]), 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.



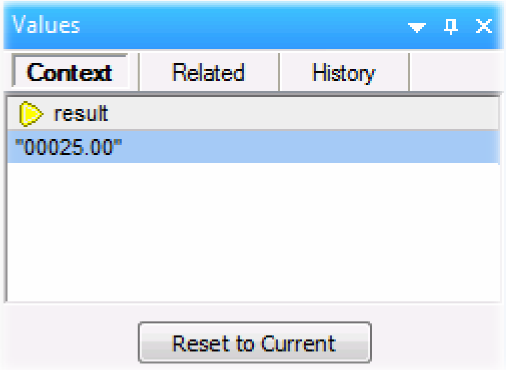
result
"00025.00"
"00002.30"
"00034.00"
"00057.50"

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[775]. For information about the context itself, see Using the Context Window[782].

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"

9 Automation with Altova Products

Mappings designed with MapForce can be executed in a server environment (including Linux and macOS servers), and with server-level performance, by the following Altova transformation engines (licensed separately):

- *RaptorXML Server*. Running a mapping with this engine is suitable if the transformation language of the mapping is XSLT 1.0, XSLT 2.0, XSLT 3.0, or XQuery. See Automation with RaptorXML Server 792.
- *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 793.

** 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 811.

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 is optimized for the latest standards and parallel computing environments. Designed to be highly cross-platform capable, the engine takes advantage of today's ubiquitous multi-core computers to deliver lightning-fast processing of XML and XBRL data.

RaptorXML is available in two editions which can be downloaded from the Altova download page (https://www.altova.com/download-trial-server.html):

- RaptorXML Server is a very fast XML processing engine with support for XML, XML Schema, XSLT, XPath, XQuery, and more.
- RaptorXML+XBRL Server supports all the features of RaptorXML Server with the additional capability of processing and validating the XBRL family of standards.

If you generate code in XSLT or in XQuery, MapForce creates a batch file called **DoTransform.bat** which is placed in the output folder that you choose upon generation. Executing the batch file calls RaptorXML Server and executes the XSLT (or XQuery) transformation on the server.

If you intend to execute or automate MapForce mappings for other outputs on a server, see Automation with MapForce Server[793].

Note: You can also preview the XSLT[66] 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 macOS operating systems. The role of MapForce Server is to execute mappings in a server environment (including on non-Windows platforms) and with server-level performance. Any MapForce mapping where the target execution language is BUILT-IN qualifies for server execution (see also Selecting a Transformation Language 19). 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 799.

If MapForce Server runs under FlowForce Server management, the mapping can be deployed to a target machine through an HTTP (or SSL/HTTPS) connection directly from MapForce. On the server, the mapping can then be executed as a triggered or scheduled job, or through a Web service call defined from the FlowForce Server administration interface. For further information, see Deploying Mappings to FlowForce Server 802.

There are two editions of MapForce Server:

- MapForce Server
- MapForce Server Advanced Edition

MapForce Server Advanced Edition provides the same features as MapForce Server, and additionally includes optimization features for mappings which qualify for optimization. This is the case of mappings which join or filter large amounts of data, and where it is possible to apply join optimization so as to increase the execution speed. Unlike MapForce Server, MapForce Server Advanced Edition can execute mappings where node functions are present, see Defaults and Node Functions 442.

Limitations:

- XML digital signatures are not supported
- ADO, ADO.NET, and ODBC database connections are supported only on Windows (for other operating systems, see Preparing Mappings for Server Execution 794).

For more information about MapForce Server, refer to its accompanying documentation (https://www.altova.com/documentation).

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 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 64).

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 799.
- 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:

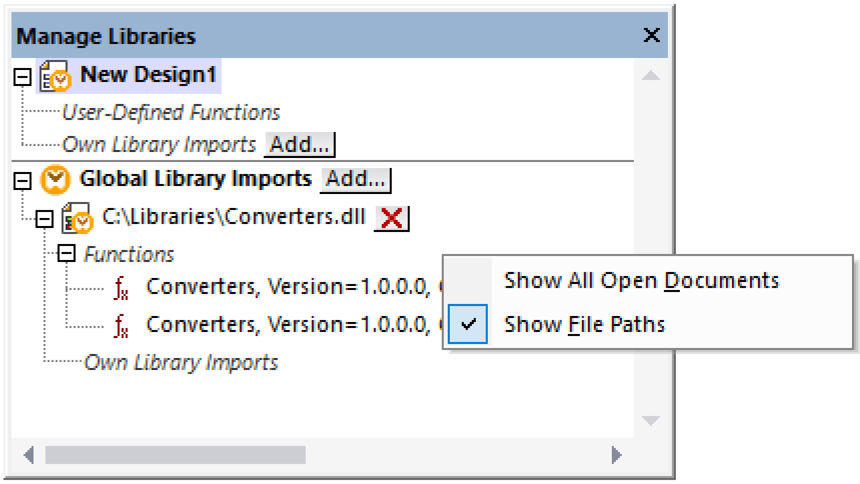
- On the source machine, MapForce Enterprise or Professional edition is required to design the mapping and deploy it to a target machine, see Deploying Mappings to FlowForce Server 834.
- Both MapForce Server and FlowForce Server must be licensed on the target machine. The role of MapForce Server is to run the mapping; the role of FlowForce is to make the mapping available as a job which benefits from features such as scheduled or on demand execution, execution as a Web service, error handling, conditional processing, email notifications, and others.
- FlowForce Server must be up and running at the configured network address and port. Namely, the "FlowForce Web Server" service must be started and configured to accept connections from HTTP clients (or HTTPS if configured) and must not be blocked by the firewall. The "FlowForce Server" service must also be started and running at the designated address and port.
- You have a FlowForce Server user account with permissions to one of the containers (by default, the **/public** container is accessible to any authenticated user).

General considerations

- If you intend to run the mapping on a target machine with standalone MapForce Server, all input files referenced by the mapping must be copied to the target machine as well. If MapForce Server runs under FlowForce Server management, there is no need to copy files manually. In this case, the instance and schema files are included in the package deployed to the target machine, see Deploying Mappings to FlowForce Server 834.
- If the mapping includes database components which require specific database drivers, such drivers must be installed on the target machine as well. For example, if your mapping reads data from a Microsoft Access database, then Microsoft Access or Microsoft Access Runtime

(https://www.microsoft.com/en-us/download/details.aspx?id=50040) must be installed on the target machine as well.

- When you deploy a mapping to non-Windows platforms, ADO, ADO.NET and ODBC database connections are automatically changed to JDBC. Native SQLite and native PostgreSQL connections are preserved as such and require no additional configuration. See also "Database connections" below.
- If the mapping contains custom function calls (for example, to .dll or .class files), such dependencies are not deployed together with the mapping, since they are not known before runtime. In this case, copy them manually to the target machine. The path of the .dll or .class file on the server must be the same as in the "Manage Libraries" window in MapForce, for example:



- Some mappings read multiple input files using a wildcard path (see Processing Multiple Input or Output Files Dynamically 746). 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 1077).
- 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.
- If the mapping connects to file-based databases such as Microsoft Access and SQLite, the database file must be manually transferred to the target machine or saved to a shared directory which is accessible to both the source and the target machine and referenced from there, see "File-based databases" below.

Making paths portable

If you intend to run the mapping on a server, ensure that the mapping follows the applicable path conventions and uses a supported database connection.

To make paths portable to non-Windows operating systems, use relative instead of absolute paths when designing the mapping in MapForce:

1. Open the desired mapping design file (.mfd) with MapForce on Windows.
2. On the **File** menu, select **Mapping Settings**, and clear the **Make paths absolute in generated code** check box if it is selected.
3. For each mapping component, open the **Properties** dialog box (by double-clicking the component's

title bar, for example), and change all file paths from absolute to relative. Also, select the **Save all file paths relative to MFD file** check box. For convenience, you can copy all input files and schemas into the same folder as the mapping itself, and reference them just by the file name.

For more information about dealing with relative and absolute paths while designing mappings, , see Using Relative and Absolute Paths 43.

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 46. The working directory is specified at mapping runtime, as follows:

- In FlowForce Server, by editing the "Working-directory" parameter of any job.
- In MapForce Server API, through the `WorkingDirectory` property of the COM and .NET API, or through the `setWorkingDirectory` method of the Java API.
- In MapForce Server command line, the working directory is the current directory of the command shell.

Database connections

Be aware that ADO, ADO.NET, and ODBC connections are not supported on Linux and macOS machines. Therefore, if the target machine is Linux or macOS, such connections are converted to JDBC when you deploy the mapping to FlowForce or when you compile the mapping to a MapForce Server execution file. In this case, you have the following options before deploying the mapping or compiling it to a server execution file:

- In MapForce, create a JDBC connection to the database (see Setting up a JDBC Connection 187)
- In MapForce, fill the JDBC database connection details in the "JDBC-specific Settings" section of the database component (see Database Component Settings 235).

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 166. If the mapping connects to a file-based database, such as Microsoft Access and SQLite, additional configuration is required, see "File-based databases" below.

Running mappings with JDBC connections requires that the Java Runtime Environment or Java Development Kit be installed on the server machine. This may be either Oracle JDK or an open source build such as Oracle OpenJDK.

- The `JAVA_HOME` environment variable must point to the JDK installation directory.
- On Windows, a Java Virtual Machine path found in the Windows registry will take priority over the `JAVA_HOME` variable.
- The JDK platform (64-bit, 32-bit) must be the same as that of MapForce Server. Otherwise, you may get an error with the reason: "JVM is inaccessible".

To set up a JDBC connection on Linux or macOS:

1. Download the JDBC driver supplied by the database vendor and install it on the operating system. Make sure to select the 32-bit version if your operating system runs on 32-bit, and the 64-bit version if your operating system runs on 64-bit.
2. Set the environment variables to the location where the JDBC driver is installed. Typically, you will need to set the CLASSPATH variable, and possibly a few others. To find out which specific environment variables must be configured, check the documentation supplied with the JDBC driver.

Note: On macOS, the system expects any installed JDBC libraries to be in the **/Library/Java/Extensions** directory. Therefore, it is recommended that you unpack the JDBC driver to this location; otherwise, you will need to configure the system to look for the JDBC library at the path where you installed the JDBC driver.

Oracle Instant Client connections on macOS

These instructions are applicable if you connect to an Oracle database through the **Oracle Database Instant Client**, on macOS. Prerequisites:

- Java 8.0 or later must be installed. If the Mac machine runs a Java version prior to Java 8, you can also connect through the **JDBC Thin for All Platforms** library, and disregard the instructions below.
- Oracle Instant Client must be installed. You can download the Oracle Instant Client from the Oracle official download page. Note that there are several Instant Client packages available on the Oracle download page. Make sure to select a package with Oracle Call Interface (OCI) support, (for example, Instant Client Basic). Also, make sure to select the 32-bit version if your operating system runs on 32-bit, and the 64-bit version if your operating system runs on 64-bit.

Once you have downloaded and unpacked the Oracle Instant Client, edit the property list (.plist) file shipped with the installer so that the following environment variables point to the location of the corresponding driver paths, for example:

Variable	Sample Value
`CLASSPATH`	`/opt/oracle/instantclient_11_2/ojdbc6.jar:/opt/oracle/instantclient_11_2/ojdbc5.jar`
`TNS_ADMIN`	`/opt/oracle/NETWORK_ADMIN`
`ORACLE_HOME`	`/opt/oracle/instantclient_11_2`
`DYLD_LIBRARY_PATH`	`/opt/oracle/instantclient_11_2`
`PATH`	`$PATH:/opt/oracle/instantclient_11_2`

Note: Edit the sample values above to fit the paths where Oracle Instant Client files are installed on your operating system.

File-based databases

File-based databases such as Microsoft Access and SQLite are not included in the package deployed to FlowForce Server or in the compiled MapForce Server execution file. Therefore, if the source and target machine are not the same, take the following steps:

1. In MapForce, right-click the mapping and clear the check box **Make paths absolute in generated code** (see Changing the Mapping Settings 75).
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 43. 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.
- 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 43).

Global Resources

If a mapping includes references to Global Resources instead of direct paths or database connections, you will be able to use Global Resources on the server side as well. When you compile a mapping to a MapForce Server execution file (.mfx), the references to Global Resources will be kept intact, so that you can provide these on the server side, at mapping runtime. When deploying a mapping to FlowForce Server, you can optionally choose whether it should use resources on the server.

For mappings (or mapping functions, in case of FlowForce Server) to run successfully, the actual file, folder, or database connection details that you supply as Global Resources must be compatible with the server environment. For example, files and folders paths must use the Linux convention for paths if the mapping will run on a Linux server. Likewise, Global Resources defined as database connections must be possible on the server machine.

For further information, see Global Resources in MapForce Server 833 and Global Resources in FlowForce Server 833..

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[793]). There are two ways to execute a mapping with MapForce Server:

- If MapForce Server runs in standalone mode (that is, no FlowForce Server is installed), the mapping must be compiled to a server execution file (.mfx), as shown below. You can then run the .mfx file at the command line, using the command `run`. You can also run the mapping by invoking the `run` method of the MapForce Server API. For further information, see the MapForce Server documentation (https://www.altova.com/documentation).
- If MapForce Server runs under FlowForce Server management, you can deploy the mapping to a machine where both MapForce Server and FlowForce Server run. For more information about this scenario, see Deploying Mappings to FlowForce Server[802].

Prerequisites

See Preparing Mappings for Server Execution[794].

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\MapForce2025\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\MapForce2025\MapForce.exe" "C:
\Users\altova\Documents\Altova\MapForce2025\MapForceExamples\SimpleTotal.mfd" /COMPILE
"C:\Users\altova\Desktop"
```

See also the MapForce Command Line Interface[811].

What's included in the .mfx file

The .mfx file includes the following data:

- The mapping algorithm, which includes all user-defined functions (UDFs) imported from other mappings.

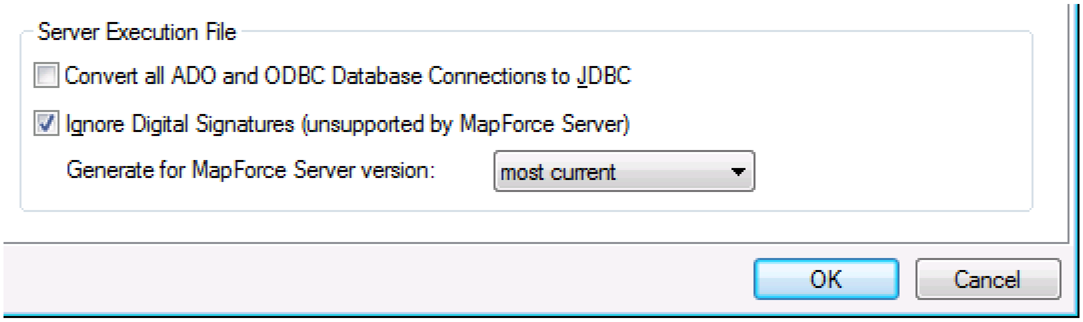
- Input and output file names referenced from components. Paths are absolute or relative depending on the mapping settings, see Paths in Various Execution Environments 46.
- If the mapping contains XML components, information about the XML schema needed to execute the mapping is encoded into the mapping algorithm.
- The database connection details, if the mapping includes database connections. Passwords are encrypted.

The input instance files (XML, CSV, Text) that are used by the mapping are not included in the compiled .mfx file. The same is true for file-based databases such as Access or SQLite. For details, see Preparing Mappings for Server Execution 794.

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\MapForce2025\MapForce.exe" /COMPILE /MFXVERSION:2025
```

See also the MapForce Command Line Interface 811.

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 235). The JDBC connection will be used implicitly if the target machine is a Linux or macOS server.
Ignore Digital Signatures (unsupported by MapForce Server)	This option is applicable only to MapForce Enterprise. It is enabled by default. If the mapping uses XML digital signatures, it skips the digital signature information, since MapForce Server does not support XML digital signatures.

To view or change these options:

- On the **Tools** menu, click **Options**, and then click **Generation**.

These options are also available from the command line interface. See also the MapForce Command Line Interface 811.

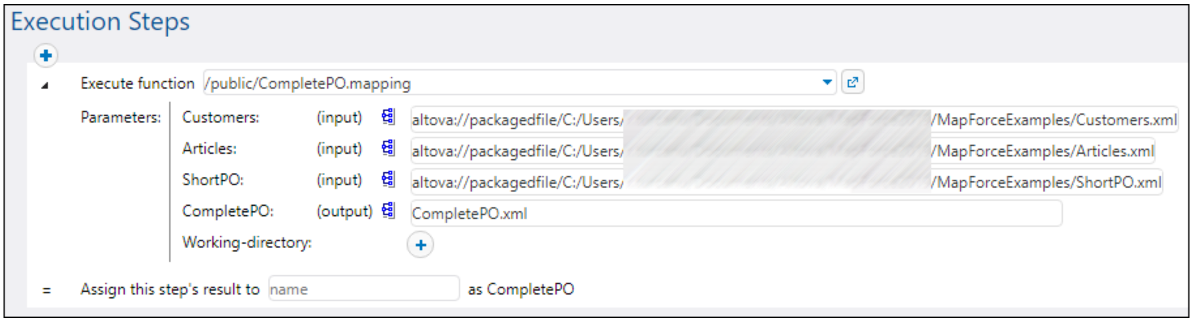
9.5 Deploying Mappings to FlowForce Server

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 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.



Execution Steps

Execute function /public/CompletePO.mapping

Parameters:
Customers: (input) altova://packagedfile/C:/Users/[redacted]/MapForceExamples/Customers.xml
Articles: (input) altova://packagedfile/C:/Users/[redacted]/MapForceExamples/Articles.xml
ShortPO: (input) altova://packagedfile/C:/Users/[redacted]/MapForceExamples/ShortPO.xml
CompletePO: (output) CompletePO.xml
Working-directory:

Assign this step's result to name as CompletePO

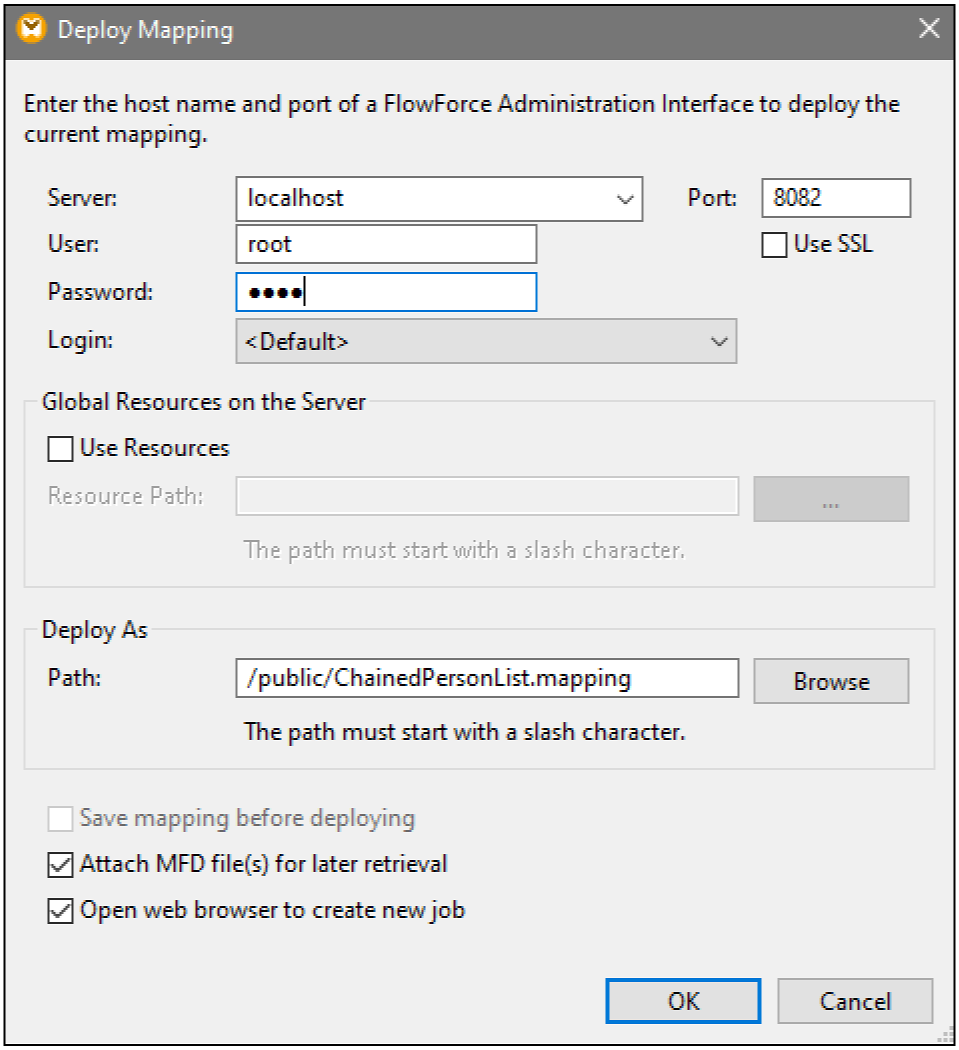
- All kinds of input instance files (XML, CSV, Text) that are used by the mapping.

Prerequisites

See Preparing Mappings for Server Execution 794.

Deploying the mapping to FlowForce Server

1. Run MapForce and ensure that the transformation language is set to Built-In.
2. In the **File** menu, click **Deploy to FlowForce Server**. The **Deploy Mapping** dialog box opens (*see below*).



Deploy Mapping

Enter the host name and port of a FlowForce Administration Interface to deploy the current mapping.

Server: localhost

Port: 8082

□ Use SSL

User: root

Password: ●●●●

Login: <Default>

Global Resources on the Server

□ Use Resources

Resource Path:

The path must start with a slash character.

Deploy As

Path: /public/ChainedPersonList.mapping

The path must start with a slash character.

□ Save mapping before deploying

☑ Attach MFD file(s) for later retrieval

☑ Open web browser to create new job

3. Enter your deployment settings (as described below) and click OK. If you select 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 table below lists the mapping-deployment settings available in the **Deploy Mapping** dialog box.

Setting	Description
Server, Port, Use SSL	Enter the server host name (or IP address) and port of FlowForce Server. These could be **localhost** and **8082** if FlowForce Server is running on the same machine at the default port. When in doubt, log on to FlowForce Server Web administration interface and check the I.P. address and port displayed in the Web browser's address bar. If you encounter connectivity errors, ensure that the machine on which FlowForce Server runs is configured to allow incoming connections on the designated address and port.

Setting	Description
	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 (https://www.altova.com/documentation).
User and Password	The user name and password to be entered depends on the value of the Login drop-down list (see next option). If the Login drop-down list is set to **<Default>** or **Directly**, enter your FlowForce Server user name and password. Otherwise, enter your domain user name and password, and select the domain name from the Login drop-down list.
Login	If Directory Service integration is enabled in FlowForce Server, select the domain name from this drop-down list, and enter your domain credentials in the User and Password fields (see previous option).
Use Resources, Resource Path	Select the **Use Resources** check box if the mapping function should use Resources 815 after it is deployed to the server. If you select the check box, you must also enter the path of the respective resource on the server in the **Resource Path** text box. To select the resource, click the **Ellipsis** button. If there are no resources on the server yet to choose from, click **Deploy Global Resources** and deploy the required Global Resource to the server. For more information, see Deploying Global Resources to FlowForce Server 834. If you do not select the **Use Resources** check box, any Global Resources will be resolved, based on the currently selected configuration. On the server, the mapping function will no longer require Global Resources, but will use the resolved value instead.
Path	Click **Browse**, and select the path where the mapping function should be saved in the FlowForce Server container hierarchy. By default, the path is set to the **/public** container of FlowForce Server. From the dialog box, you can also create new containers or delete existing containers and mappings, provided that you have the required FlowForce Server permissions and privileges.
Save mapping before deploying	This option is available if you are deploying an unsaved mapping. Select this check box to save the mapping before deployment.
Attach MFD files for later retrieval	This option enables you to deploy the MFD file together with its dependent input files (e.g., source XML file(s)), except for structure-defining files (e.g., XSD schemas). When you open the deployed mapping in FlowForce

Setting	Description
	Server, the *Deployed Files* section will list all the files that you can download.
Open browser to create new job	If you select this check box, the FlowForce Server Web administration interface opens in the browser after deployment, and you can start creating a FlowForce Server job immediately.

Troubleshooting

The following table lists problems that you might encounter when deploying a mapping, and their solution.

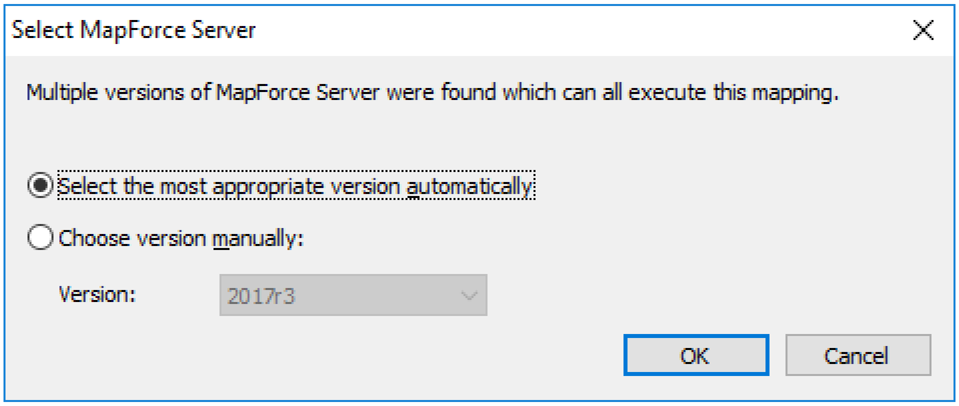
Problem	Solution

Deploying the mapping returns the following error:
`I/O operation on file ... failed.`
`I/O Error 28: Failed to connect to <server> port 8082. Timed out`
`System error 10060: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.`
	Make sure that, on the target machine, the *FlowForce Web Server* service is running and configured to listen for connections on the specified port (**8082**, by default). Also, make sure that the firewall does not block incoming connections through this port.
	The *FlowForce Server* service must be running as well in order for the deployment to be possible.

Deploying the mapping returns the following error:
`I/O operation on file ... failed.`
`I/O Error 413: Payload Too Large`
	This error may occur if an input file of the deployed mapping exceeds the maximum size limit of HTTP requests allowed by FlowForce Server (roughly 100 MB). You can increase the limit by setting the `max_request_body_size` option (in bytes) in the **flowforceweb.ini** and **flowforce.ini** files. For details, see the FlowForce Server documentation.

Selecting the server version (Windows only)

If the server where you deploy the mapping has multiple versions of MapForce Server running under FlowForce Server management (applicable to Windows servers only), then you are additionally prompted to specify the version of MapForce Server with which you want this mapping to be executed.


Select MapForce Server

Multiple versions of MapForce Server were found which can all execute this mapping.

● Select the most appropriate version automatically

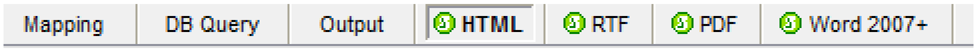
○ Choose version manually:

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\FlowForceServer2025\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\MapForceServer2025\etc**.

9.6 StyleVision Output Panes

In mappings where the target component is XML , it is possible to preview and save the mapping output as HTML, RTF, PDF, Word 2007+, and Text documents if Altova StyleVision is installed on your computer. If you are using StyleVision Enterprise Edition, charts will also be rendered in these previews. When a mapping supports preview in any of these formats, additional panes become available next to the **Output** pane (*see screenshot below*).



Important

- When StyleVision Professional is installed, it is possible to preview **HTML**, **RTF**, and **Text** outputs.
- With StyleVision Enterprise, it is possible to preview **HTML**, **RTF**, **PDF**, **Word 2007+**, and **Text** outputs.
- Previewing the mapping output as PDF requires Java, Acrobat Reader, and FOP (Formatting Objects Processor) version 0.93 or 1.0. FOP is installed together with StyleVision unless you opted not to install it when installing StyleVision.
- In the 64-bit edition of MapForce, the Word 2007+ and RTF previews are opened as non-embedded applications.
- If your mapping contains components that act both as sources and targets (pass-through components), the StyleVision preview will be possible only for those components where the **Preview** button of the component has been activated. For more information about such mappings, see Chained Mapping 115.

To preview your mapping data in the StyleVision output panes, the following is required:

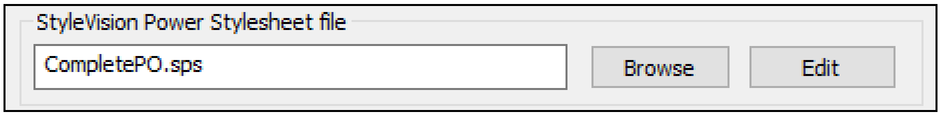
- Altova StyleVision must be installed on your computer either as a standalone installation or as part of Altova MissionKit.
- The target component must have a StyleVision Power Stylesheet (SPS) file associated with it. The stylesheet file can be created or edited with StyleVision. You cannot edit or change the stylesheet in MapForce directly, but you can open it via MapForce in StyleVision. Once the stylesheet is ready, you can assign it to a target MapForce component, as shown below.

StyleVision output panes configuration

The instructions below will help you set up StyleVision output panes.

Assign a StyleVision Power Stylesheet to a target component
To assign an SPS file to a target component, take the following steps:

1. In StyleVision, create the required stylesheet file. Make sure to use the same XML schema as a source as that of the MapForce component.
2. In MapForce, right-click the target XML component and select **Properties**.
3. In the **Component Settings** dialog box, next to **StyleVision Power Stylesheet file**, browse for the stylesheet file created previously (*see screenshot below*).



StyleVision Power Stylesheet file

CompletePO.sps

Note: The path to the StyleVision Power Stylesheet file can be absolute or relative. For details, see Relative and Absolute Paths[43].

Save the StyleVision-generated output

You can save the StyleVision-generated output to a file in a similar way as saving the result of any other mapping: Click the 💾 toolbar button (**Save generated output**) or go to the **Output** menu and click **Save Output File**.

Automate generation of different formats with Altova products

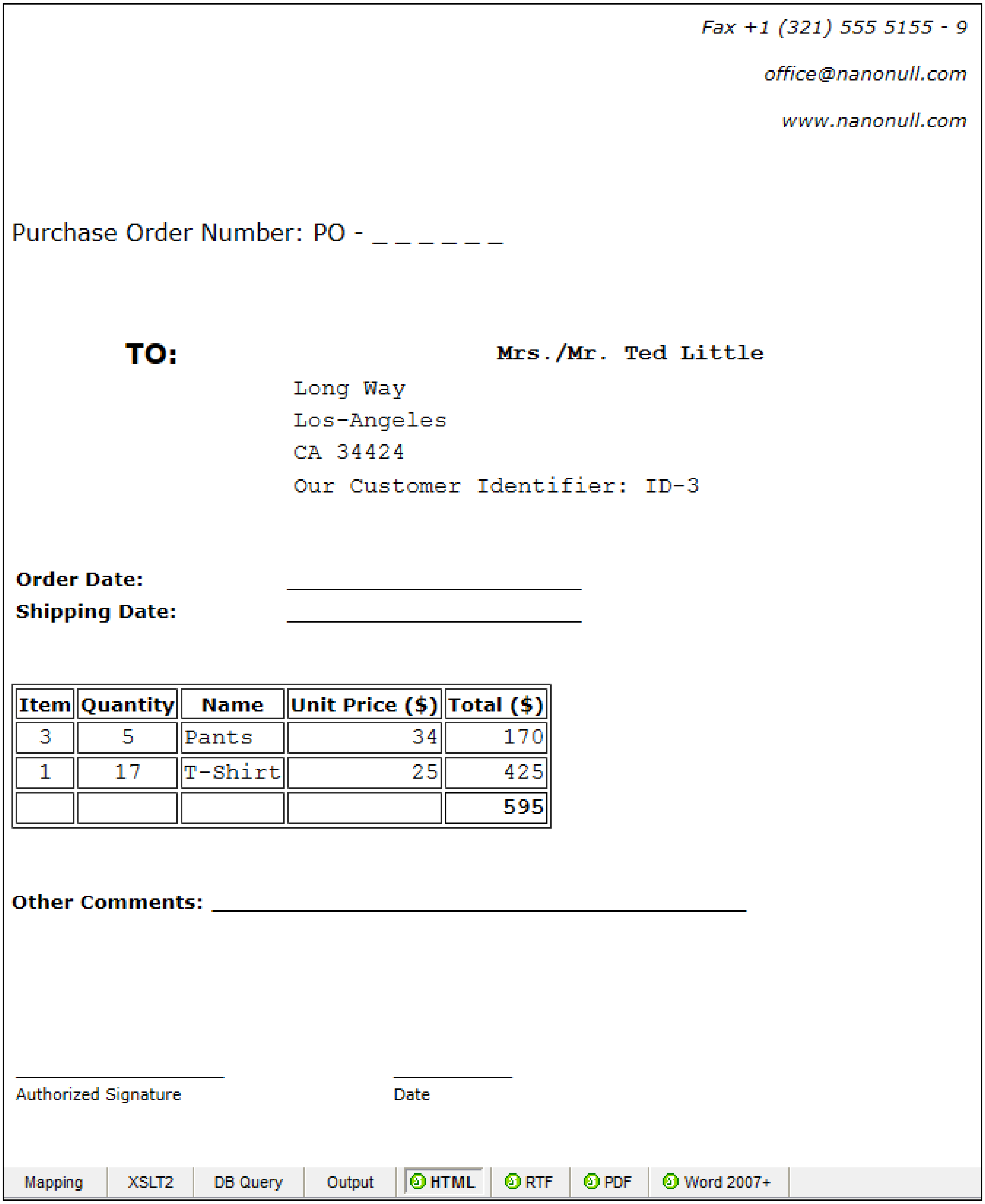
If you need your mapping to generate HTML, PDF, RTF, Word 2007+, and Text files automatically (either on the same or on a different computer or even platform), you can use MapForce Server or StyleVision Server. These are separately licensed server products that extend the functionality of MapForce and StyleVision, respectively. In this scenario, each application plays its own specific role:

- MapForce enables you to design a mapping (**.mfd**) which defines the data transformation inputs and outputs (for example, database to XML)
- MapForceServer runs the executable mapping (**.mfx**) from the command line or an API (on the same or a different operating system)
- StyleVision enables you to design the stylesheet (**.sps**) required to transform the mapping output to HTML, PDF, RTF, Word 2007+, and Text files.
- StyleVision Server runs the **.sps** stylesheet which transforms the mapping output to a desired format. This happens in the command line or from an API (on the same or a different operating system).
- Both StyleVision Server and MapForce Server can optionally run under the management of FlowForce Server (licensed separately). In this scenario, MapForce mappings and StyleVision transformations can run as scheduled, triggered, or on-demand jobs. This means that these MapForce mappings and StyleVision transformations can be fully automated.

Examples

The example below (**MapForceExamples\CompletePO.mfd**) shows the output in the StyleVision output pane called **HTML**. This mapping produces a purchase order in XML format. Right-click the target component, select **Properties**, and notice that it has a **.sps** file assigned to it.

If you click the **HTML** pane, you will see the following output:



Fax +1 (321) 555 5155 - 9

office@nanonull.com

www.nanonull.com

Purchase Order Number: PO - _ _ _ _ _ _

TO: **Mrs./Mr. Ted Little**

Long Way
Los-Angeles
CA 34424
Our Customer Identifier: ID-3

Order Date: ____________________

Shipping Date: ____________________

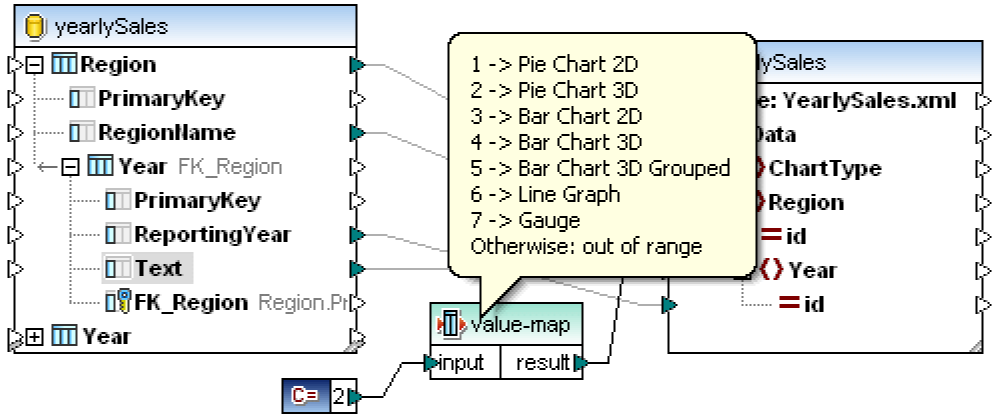
Item	Quantity	Name	Unit Price ($)	Total ($)
3	5	Pants	34	170
1	17	T-Shirt	25	425
				595

Other Comments: ____________________________________

Authorized Signature

Date

Another example is **`Tutorial\YearlySales.mfd`**. The stylesheet assigned to this mapping was designed in StyleVision in such a way that it is possible to control the type of the chart by changing the value of the `ChartType` element. This makes it possible to change the chart type directly from the mapping: You can change the default value of the constant to any value from 1 to 7. If you place the mouse cursor over the `value-map` component, you will see the possible values (*see screenshot below*).



1 -> Pie Chart 2D
2 -> Pie Chart 3D
3 -> Bar Chart 2D
4 -> Bar Chart 3D
5 -> Bar Chart 3D Grouped
6 -> Line Graph
7 -> Gauge
Otherwise: out of range

The default value of the constant is 2, which generates a 3D pie chart in the output. To display other chart types, change this value to any other allowed value and click the **Output** pane to see the changes.

9.7 MapForce Command Line Interface

The general syntax of a MapForce command at the command line is as follows:

```
MapForce.exe <filename> [/{target} [[<outputdir>] [/options]]]
```

For more information about each parameter of the command, see *Parameters* below.

Notation
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

Parameters
`<filename>`
The mapping design (**`.mfd`**) or mapping project (**`.mfp`**) (*Professional and Enterprise editions*) from which code is to be generated. To generate code for the whole project, set the target `/GENERATE` (*see /{target} below*) and enter the project path as `<filename>` (e.g., **`MapForceExamples.mfp`**).

`/{target}`
Specifies the target language or environment for which code is to be generated. The following code generation targets are supported:

- `/XSLT`
- `/XSLT2`
- `/XSLT3`
- The `/COMPILE[:compileoptions]` command compiles a mapping to a MapForce Server execution file (**`.mfx`**). You can also supply the following options separated by a comma:
  - o The `JDBC` option transforms all database connections to JDBC using the JDBC driver and the database URL defined in the Database Component Settings 235 dialog box.
  - o The `NOXMLSIGNATURES` option suppresses the generation of digital signatures in the MapForce Server Execution file. Note that digital signatures are not supported by MapForce Server.
- The `/GENERATE` command generates project code for all mappings in the project file, using the current folder settings (see Managing Project Folders 98). If you select this target, make sure to supply a MapForce project (**`.mfp`**) as `<filename>`.
- `/XQuery`
- `/JAVA`

- The /CS command generates C# code. This command allows setting the following code generation options:

  ```
  /CS[:{VS2013|VS2015|VS2017|VS2019|VS2022|DOTNETCORE31|DOTNET50|DOTNET60}]
  ```

  If no Visual Studio version is specified, code will be generated using the Visual Studio version defined in the code generation options.

- The /CPP command generates C++ code. This command allows setting the following code generation options:

  ```
  /CPP[:{VS2013|VS2015|VS2017|VS2019|VS2022|DOTNETCORE31|DOTNET50|DOTNET60},{MSXML|
  XERCES3},{LIB|DLL},{MFC|NoMFC}]
  ```

  The first group of options specifies the development environment (e.g., VS2022 stands for Visual Studio 2022).

  The second group of options specifies the XML library targeted by the generated code. The following values are valid:

  o MSXML (generates code for MSXML 6.0)
  o XERCES3 (generates code for Xerces 3)

  The third group of options specifies whether static or dynamic libraries should be generated. Valid values include the following:

  o LIB (generates static LIB libraries)
  o DLL (generates DLL libraries)

  The fourth group of options specifies whether code should be generated with or without MFC support. Valid values include the following:

  o MFC (enables MFC support)
  o NoMFC (disables MFC support)

  If the options above are not specified, code will be generated using the Visual Studio version defined in the code generation 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 the target is /GENERATE and the <outputdir> parameter is not set, the code generation language and the output path of each mapping are supplied by the settings defined for each folder inside the project (see Managing Project Folders 98).

When the 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:

- The /GLOBALRESOURCEFILE <filename> option is applicable if the mapping uses Global Resources to resolve an input or output file, folder paths, or databases. For more information, see Altova Global Resources 815. The /GLOBALRESOURCEFILE option specifies the path to a Global Resource XML file. Note that, if /GLOBALRESOURCEFILE is set, then /GLOBALRESOURCECONFIG must also be set.

- The /GLOBALRESOURCECONFIG <config> option specifies the name of the Global Resource configuration (*see also the previous option*). If /GLOBALRESOURCEFILE is set, then /GLOBALRESOURCECONFIG must also be set.

- The /LOG <logfilename> option generates a log file at the specified path. The <logfilename> path can be an absolute path. If a full path is supplied, the directory must exist for the log file to be generated. If you specify only the file name, the file will be placed in the current directory of the Windows command prompt.

- The [/MFXVERSION[:<version>] option is applicable if the target is /COMPILE. This option compiles the MapForce Server Execution (.mfx) file for a particular version of MapForce Server. You can supply any version of MapForce Server, starting with 2013r2 onwards. See also Compiling mappings for a specific MapForce Server version 800.

- The /LIBRARY <libname> (...) option is used together with a code generation target language to specify additional function libraries. This option can be specified more than once to load multiple libraries. See also Managing Function Libraries 438.

Notes

- 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 (e.g., **"C:\My directory\"**). These two characters are interpreted by the command line parser as a literal double quotation mark. It is recommended to avoid using spaces and quotes. If spaces occur in the command line and you need the quotes, use the double backslash (e.g., **"c:\My Directory\\"**).

Examples

1) To start MapForce and open the mapping <filename>.mfd, use the following command:

```
MapForce.exe <filename>.mfd
```

2) To generate XSLT 2.0 code and create a log file with the name <logfilename>, use the following command:

```
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 the following command:

```
Mapforce.exe <filename>.mfd /XSLT2 <outputdir> /GLOBALRESOURCEFILE
<grfilename> /GLOBALRESOURCECONFIG <grconfigname>
```

Examples for Professional and Enterprise editions

1) To generate a C# application for Visual Studio 2022 and output a log file, use the following command:

```
MapForce.exe <filename>.mfd /CS:VS2022 <outputdir> /LOG <logfilename>
```

2) To generate a C++ application using the code generation settings defined in **Tools | Options** and output a log file, use the following command:

```
MapForce.exe <filename>.mfd /CPP <outputdir> /LOG <logfilename>
```

3) To generate a C++ application for Visual Studio 2022, MSXML, with static libraries, MFC support, and no log file, use the following command:

```
MapForce.exe <filename>.mfd /CPP:VS2022,MSXML,LIB,MFC
```

4) To generate a C++ application for Visual Studio 2022, Xerces, with dynamic libraries, no MFC support, and a log file, use the following command:

```
MapForce.exe <filename>.mfd /CPP:VS2022,XERCES,DLL,NoMFC <outputdir> /LOG
<logfilename>
```

5) To generate a Java application and output a log file, use the following command:

```
MapForce.exe <filename>.mfd /JAVA <outputdir> /LOG <logfilename>
```

6) 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 the following command:

```
MapForce.exe <filename>.mfp /GENERATE /LOG <logfilename>
```

7) To generate Java code for all mappings in the project file, use the following command:

```
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.

8) To supply input and output files at the command line for a previously compiled Java mapping, use the following command:

```
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 346).

9) To compile a mapping to a MapForce Server execution file for MapForce Server version 2025 and suppress XML signatures, use the following command:

```
MapForce.exe <filename>.mfd /COMPILE:NOXMLSIGNATURES
<outputdir> /MFXVERSION:2025 /LOG <logfilename>
```

10 Altova Global Resources

Altova Global Resources are aliases for file, folder, and database resources. Each alias can have multiple configurations, and each configuration maps to a single resource. Therefore, when you use a global resource, you can switch between its configurations. For example, you could create a database resource with two configurations: `development` and `production`. Depending on your goals, you can switch between these configurations. In MapForce, you can retrieve data from the `development` or `production` database by choosing the desired configuration from the drop-down list before previewing the mapping.

Global resources can be used across different Altova applications (*see subsection below*).

Global resources in other Altova products

When stored as global resources, files, folders, and database connection details become reusable across multiple Altova applications. For example, if you often need to open the same file in multiple Altova desktop applications, you can define this file as a global resource. If you need to change the file path, you will need to change it only in one place. Currently, global resources can be defined and used in the following Altova products:

- Altova Authentic
- DatabaseSpy
- MobileTogether Designer
- MapForce
- StyleVision
- XMLSpy
- FlowForce Server
- MapForce Server
- RaptorXML Server/RaptorXML+XBRL Server

In this section

This section explains how to create and configure different types of global resources. The section is organized into the following topics:

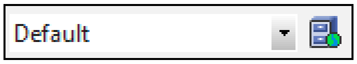
- Global Resource Setup Part 1 816
- Global Resource Setup Part 2 818
- XML Files as Global Resources 822
- Folders as Global Resources 824
- Databases as Global Resources 826
- Transformation Results as Global Resources 828
- Global Resources in Execution Environments 832

10.1 Global Resource Setup Part 1

The global resource setup consists of two parts: (i) creating a global resource in the **Manage Global Resources** dialog box (*see below*) and (ii) defining the properties of this global resource in the **Global Resource** dialog. The second part is discussed in the next topic[818].

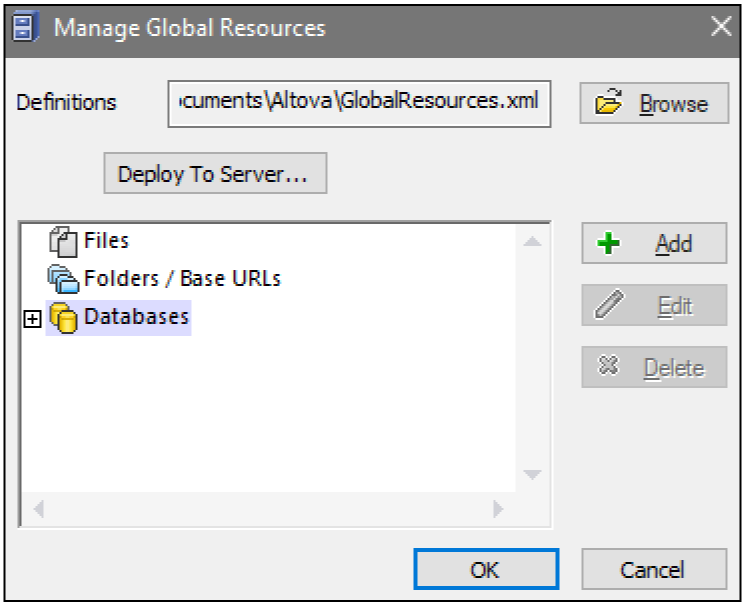
Altova Global Resources are defined in the **Manage Global Resources** dialog, which can be accessed in two ways:

- Click the menu command **Tools | Global Resources**.
- Click the **Manage Global Resources** icon in the Global Resources toolbar (*screenshot below*).



Global Resources Definitions file

Information about global resources is stored in an XML file called the Global Resources Definitions file. This file is created when the first global resource is defined in the **Manage Global Resources** dialog box (*screenshot below*) and saved.



Definitions ıcuments\Altova\GlobalResources.xml

When you open the **Manage Global Resources** dialog box for the first time, the default location and name of the Global Resources Definitions file is specified in the *Definitions* text box (*see screenshot above*):

```
C:\Users\<username>\Documents\Altova\GlobalResources.xml
```

This file is set as the default Global Resources Definitions file for all Altova applications. A global resource can be saved from any Altova application to this file and will immediately be available to all other Altova applications as a global resource. To define and save a global resource to the Global Resources Definitions file, add the global resource in the **Manage Global Resources** dialog and click **OK** to save.

To select an already existing Global Resources Definitions file to be the active definitions file of a particular Altova application, browse for it via the **Browse** button of the *Definitions* text box (*see screenshot above*).

The **Manage Global Resources** dialog box also allows you to edit and delete existing global resources.

Notes:

- You can give any name to the Global Resources Definitions file and save it to any location accessible to your Altova applications. All you need to do in each application, is specify this file as the Global Resources Definitions file for that application (in the *Definitions* text box). The resources become global across Altova products when you use a single definitions file across all Altova products.

- You can also create multiple Global Resources Definitions files. However, only one of these can be active at any time in a given Altova application, and only the definitions contained in this file will be available to the application. The availability of resources can therefore be restricted or made to overlap across products as required.

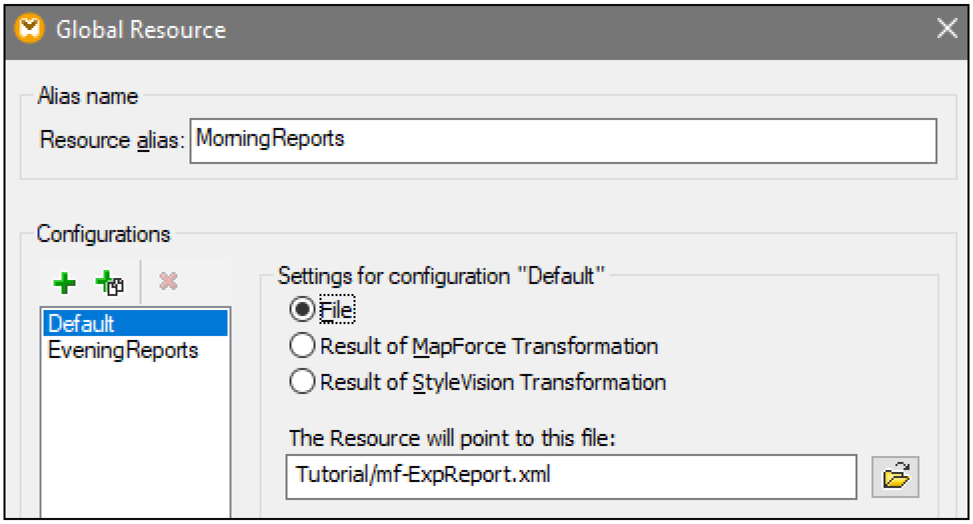
10.2 Global Resource Setup Part 2

The second part of the global resource setup consists in defining properties of a global resource in the **Global Resource** dialog box. The properties depend on the type of a global resource (*see subsections below*). You can access the **Global Resource** dialog box by clicking the **Add** button in the Manage Global Resources dialog box 816.

To find out more about setting up different types of global resources, see the following examples: XML Files as Global Resources 822, Folders as Global Resources 824, Databases as Global Resources 826.

Files

The file-specific properties are shown in the **Global Resource** dialog box below. The setup has three major parts: (i) the name of the file, (ii) the location of this file, and (iii) the list of configurations defined for this file alias.


Global Resource

Alias name

Resource alias: MorningReports

Configurations

Default
EveningReports

Settings for configuration "Default"

● File
○ Result of MapForce Transformation
○ Result of StyleVision Transformation

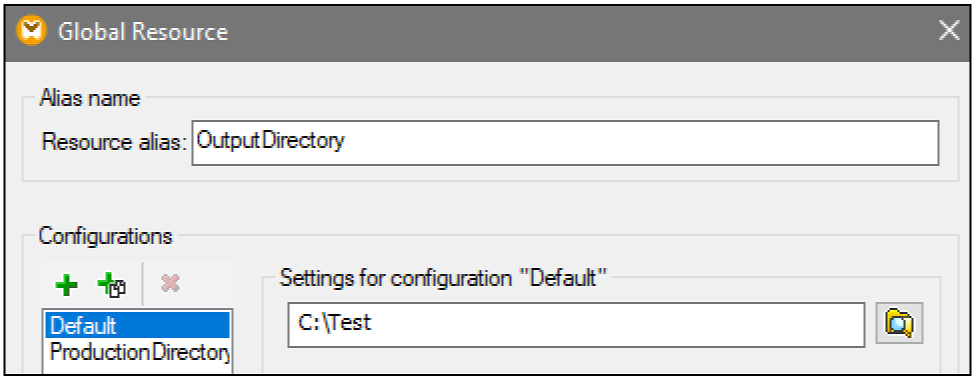
The Resource will point to this file:

Tutorial/mf-ExpReport.xml

The settings *Result of MapForce Transformation* and *Result of StyleVision Transformation* are discussed in Transformation Results as Global Resources 828.

Folders

The folder-specific properties are shown in the **Global Resource** dialog box below. The setup has three major parts: (i) the name of the folder, (ii) the location of this folder, and (iii) the list of configurations defined for this folder alias.



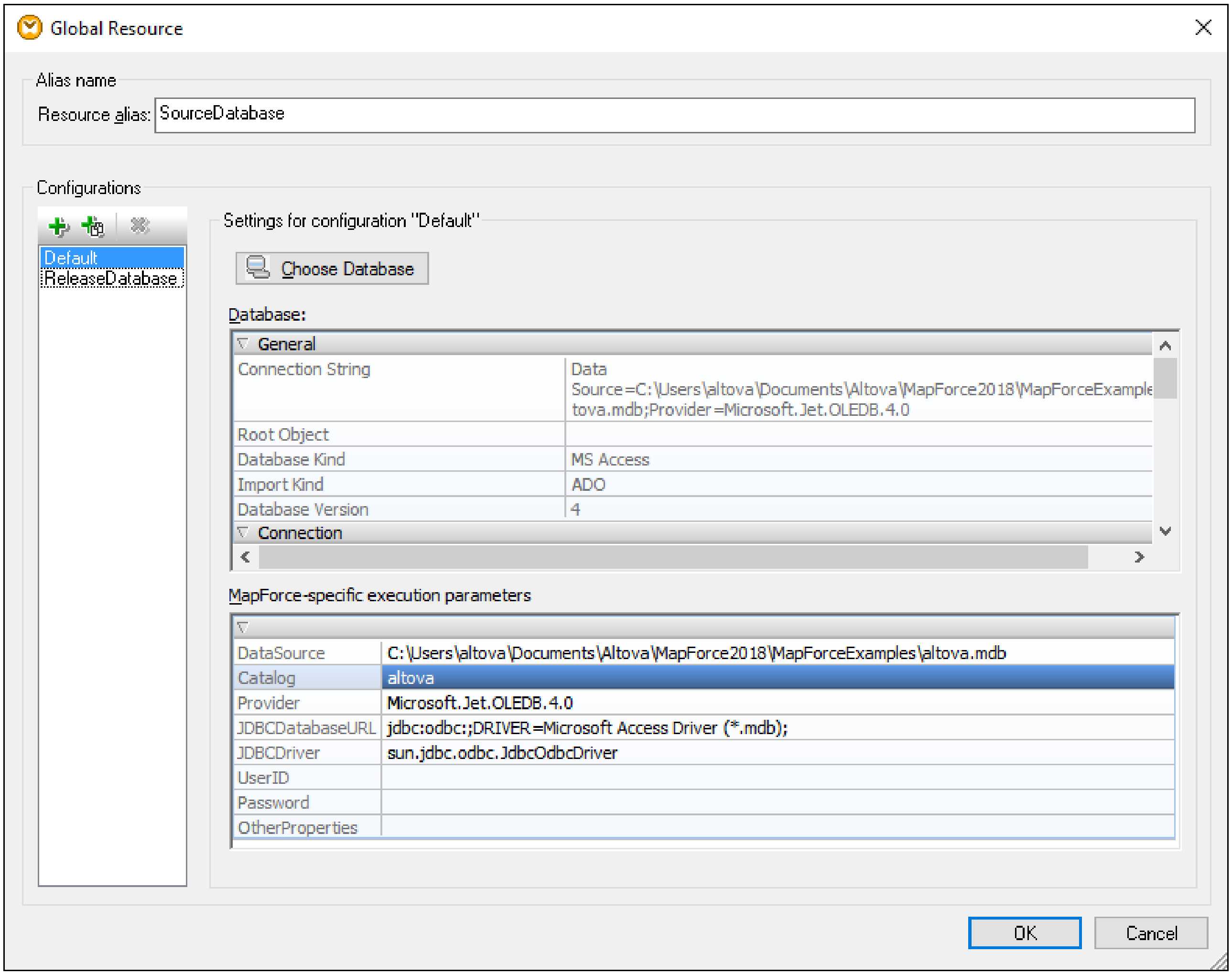
Resource alias: OutputDirectory

Settings for configuration "Default"

C:\Test

Databases

When you add a database connection as a global resource, a connection wizard guides you through the steps required to set up the connection. For more information, see Start Database Connection Wizard[170]. Once you complete the wizard, the database connection parameters are displayed in the **Global Resource** dialog box (*see screenshot below*).



Global Resource

Alias name

Resource alias: SourceDatabase

Configurations

Default
ReleaseDatabase

Settings for configuration "Default"

Choose Database

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	

OK Cancel

In the **Global Resource** dialog box, it is possible to edit some of the database connection parameters. The parameters are grouped into two categories: database parameters and MapForce-specific execution parameters (*see below*).

Database

These parameters are shared among Altova applications. In MapForce, they are used at design time, that is, when the mapping is loaded, or when you click the **Output** pane in MapForce to preview the mapping.

MapForce-specific execution parameters

These parameters are applicable when you generate program code or compile a mapping to MapForce Server execution file (**`.mfx`**). They are used at mapping runtime as follows:

- In generated C++, C#, or Java program code.
- If you compile the mapping to a MapForce Server execution file, and automatic JDBC conversion takes place. For more information about automatic JDBC conversion, see Database mappings in various execution environments 166.

If a mapping uses a Global Resource to connect to a database, the database connection details in the **Global Resource** dialog box take precedence over those defined in the mapping. The Component Settings dialog box informs you that the connectivity parameters are defined as a Global Resource. To change the database component to connect to the database directly (without using Global Resources), click **Change**, and follow the wizard steps to reconnect to the database.

Global Resource dialog icons

✚	*Add Configuration:* Pops up the Add Configuration dialog in which you enter the name of the configuration to be added.
✚	*Add Configuration as Copy:* Pops up the Add Configuration dialog in which you can enter the name of the configuration to be created as a copy of the selected configuration.
☒	*Delete:* Deletes the selected configuration.
📂	*Open:* Browse for the file to be created as the global resource.
📁	*Open:* Browse for the folder to be created as the global resource.

Global resource setup: General procedures

The broad procedure of creating and configuring global resources is described below:

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add** and select the resource type you wish to create (file, folder, database). The **Global Resource** dialog box will appear.
3. Enter a descriptive name in the **Resource alias** text box (e.g., `InputFile`).
4. Setting up the default configuration depends on the type of the global resource: (i) For a file or folder, browse for the file or folder to which this resource should point by default; (ii) for a database connection, click **Choose Database** and follow the Database Connection Wizard to connect to the database (see Connecting to a Database 168). This database connection will be used by default when you run the mapping.
5. If you need an additional configuration (e.g., an additional output folder), click the ✚ button in the **Global Resource** dialog box, enter the name of this configuration, and specify the path to this configuration.
6. Repeat the previous step for each additional configuration required.

Note: Database connections are supported as global resources only in MapForce Professional and Enterprise editions.

10.3 XML Files as Global Resources

This topic explains how to use XML files as global resources. There are situations in which you may need to change an input XML file multiple times per day. For example, every morning you need to run a particular mapping and generate a report by using one XML file as a mapping input, and every evening the same report must be generated from another XML file. Instead of editing the mapping multiple times per day (or keeping multiple copies of it), you could configure the mapping to read from a file defined as a global resource (the so-called *file alias*). In this example, our file alias will have two configurations:

1. The `Default` configuration will supply a morning XML file as a mapping input.
2. The `EveningReports` configuration will supply an evening XML file as a mapping input.

To create and configure the file alias, take the steps below.

Step 1: Create a global resource

First, we need to create a file alias. Follow the instructions below:

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add | File** and enter a name in the **Resource alias** text box. In this example, we call our default configuration `MorningReports`.
3. Click the **Browse** button near the text field **The Resource will point to this file** and select **`Tutorial\mf-ExpReport.xml`**.
4. Click + in the **Configurations** section and name the second configuration `EveningReports`.
5. Click **Browse** and select **`Tutorial\mf-ExpReport2.xml`**.

Step 2: Use the global resource in the mapping

Now we can use the newly created global resource in our mapping. To make the mapping read data from the global resource, take the steps below:

1. Open the **`Tutorial\Tut-ExpReport.mfd`** mapping.
2. Double-click the header of the source component to open the **Component Settings** dialog box.
3. Next to **Input XML file**, click **Browse**, then click **Global Resources** and select the file alias `MorningReports`. Click **Open**.
4. Open the **Component Settings** dialog box again: The input XML file path has now become **`altova://file_resource/MorningReports`**, which indicates that the path uses a global resource.

Step 3: Run the mapping with the desired configuration

You can now switch between the input XML files before running the mapping:

- To use **`mf-ExpReport.xml`** as an input, select the menu item **Tools | Active Configuration | Default**.
- To use **`mf-ExpReport2.xml`** as an input, select the menu item **Tools | Active Configuration | EveningReports**.

Alternatively, select the required configuration from the **Global Resources** drop-down list in the toolbar (*see screenshot below*).

To preview the mapping result with either configuration, click the **Output** pane.

10.4 Folders as Global Resources

This topic explains how to use folders as global resources. There are situations in which you may need to generate the same output in different directories. To make this possible, we will create a folder alias with two configurations:

1. The `Default` configuration will generate the output in **`C:\Test`**.
2. The `Production` configuration will generate the output in **`C:\Production`**.

To create and configure the folder alias, take the steps below.

Step 1: Create a global resource

First, we need to create a folder alias. Follow the instructions below:

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add | Folder** and enter a name in the **Resource alias** text box. In this example, we call our default configuration `OutputDirectory`.
3. Click the **Browse** button near the text field **Settings for configuration "Default"** and select **`C:\Test`**. Make sure this folder already exists on your operating system.
4. Click ➕ and enter a name of the second configuration. In this example, we call our second configuration `ProductionDirectory`.
5. Click **Browse** and select the **`C:\Production`** folder. Make sure that this folder already exists on your operating system.

Step 2: Use the global resource in the mapping

The next step is to make the mapping use the folder alias we have just created. Take the steps below:

1. Open the **`Tutorial\Tut-ExpReport.mfd`** mapping.
2. Double-click the header of the target component to open the **Component Settings** dialog box.
3. Click **Global Resources** and then click **Save**.
4. Save the output XML file as **`Output.xml`**. The output XML file path has now become **`altova://folder_resource/OutputDirectory/Output.xml`**, which indicates that the path is defined as a global resource.

Step 3: Run the mapping with the desired configuration

You can now switch between the output folders before running the mapping:

- To use **`C:\Test`** as the output directory, select the menu item **Tools | Active Configuration | Default**.
- To use **`C:\Production`** as the output directory, select the menu item **Tools | Active Configuration | ProductionDirectory**.

By default, the mapping output is written as a temporary file unless you explicitly configure MapForce to write output to permanent files. To configure MapForce so that it generates permanent files, do the following:

1. Go to the **Tools** menu and click **Options**.
2. In the **General** section, select the option **Write directly to final output files**.

10.5 Databases as Global Resources

This topic explains how to use databases as global resources. There are situations in which you may need to map data from databases with the same structure but different data. Using a database resource will allow you to switch between databases without editing your mapping. To achieve this, we need to create a database alias with two configurations:

1. The `Default` configuration will point to the `DevelopmentDatabase`: **`MapForceExamples\Altova.sqlite`**.
2. The `ReleaseDatabase` configuration will point to **`Tutorial\Altova.sqlite`**.

To create and configure the database alias, take the steps below.

Step 1: Create a global resource

The fist step is to create a database alias. Take the steps bellow:

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add | Database** and enter a descriptive name in the **Resource alias** text box. In this example, we call the default configuration `DevelopmentDatabase`.
3. Click **Choose Database**, select **SQLite**, and browse for **`MapForceExamples\Altova.sqlite`**.
4. Click ✚ and name the second configuration `ReleaseDatabase`.
5. Click **Choose Database**, select **SQLite**, and browse for **`Tutorial\Altova.sqlite`**.

Step 2: Use the global resource in the mapping

The next step is to configure the mapping so that it can use the database alias:

1. Open the **`Tutorial\PersonDB.mfd`** mapping.
2. Double-click the database component to open the **Component Settings** dialog box. Click **Change**.
3. Select **Global Resources** in the **Select a Database** dialog box and select the `DevelopmentDatabase` alias. Click **Connect**.
4. When you are prompted to select database objects, leave the default selection as is and click **OK**.

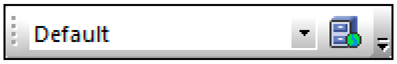
The connectivity parameters can be changed by clicking the toolbar button.

Step 3: Run the mapping with the desired configuration

You can now easily switch between the databases before running the mapping:

- To use the `DevelopmentDatabase` configuration, select the menu item **Tools | Active Configuration | Default**.
- To use the `ReleaseDatabase` configuration, select the menu item **Tools | Active Configuration | ReleaseDatabase**.

Alternatively, select the required configuration from the **Global Resources** drop-down list (*see screenshot below*).



When you switch between configurations, the **Configuration switch** dialog box informs you that the resource has been modified. Click **Reload**.

Note: The databases used in this example contain the same data. Therefore, there are no differences in the generated output.

10.6 Transformation Results as Global Resources

You can use the result of a MapForce mapping or StyleVision transformation as a global resource. This topic shows how to create a global resource from the transformation result and use this global resource across different Altova applications.

In order to make a mapping output available as a global resource, the transformation language of the mapping must be set to Built-In, or the mapping must contain only components which are supported by the Built-In language.

Important:

- The workflows mentioned above are meaningful between Altova desktop applications installed on the same computer.
- It is **not** possible to use the result of MapForce and StyleVision transformations as global resources in Altova server products and in MapForce Basic Edition.

The example below shows how to use the result of a MapForce transformation as a global resource.

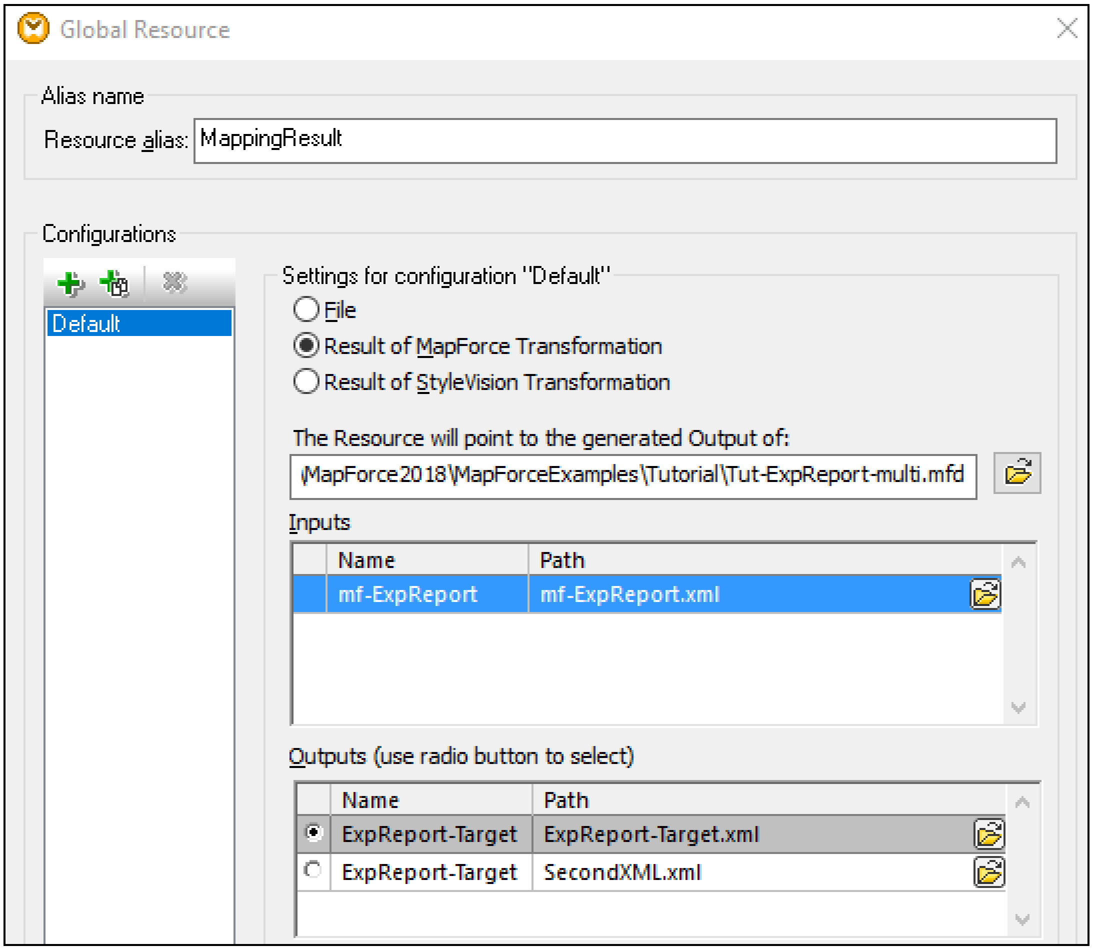
Example: Result of MapForce Transformation

This example illustrates how to create a workflow between Altova MapForce and Altova XMLSpy. More specifically, the example shows how to create a global resource from a MapForce mapping, trigger the execution of this mapping in XMLSpy, and view the outputs in XMLSpy, which were generated by MapForce.

Step 1: Create a global resource

You can take this step using MapForce or XMLSpy.

1. Click the toolbar button (**Manage Global Resource**). Alternatively, go to the **Tools** menu and click **Global Resources**.
2. Click **Add | File** and enter a descriptive name in the **Resource alias** text box. In this example, we call our default configuration `MappingResult`.
3. Select the option **Result of MapForce Transformation**.
4. Click **Browse** and select the mapping **`Tutorial\Tut-ExpReport-multi.mfd`**. As shown below, this mapping has one input and two outputs.



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

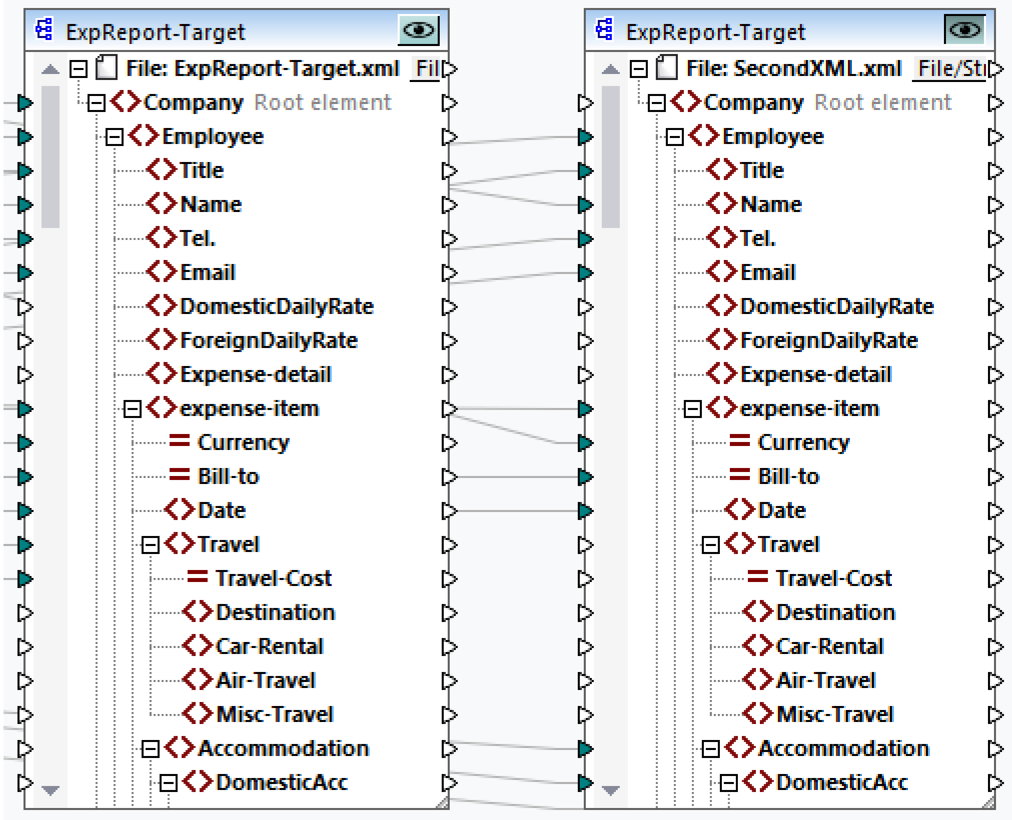
Inputs

Name	Path
mf-ExpReport	mf-ExpReport.xml

Outputs (use radio button to select)

	Name	Path
●	ExpReport-Target	ExpReport-Target.xml
○	ExpReport-Target	SecondXML.xml

The screenshot below illustrates the two outputs listed in the **Global Resource** dialog box. We will generate each output file separately in the **`C:\temp`** folder (*see Step 2 below*).



Step 2: Generate output files

At this stage, we would like to generate each of the two output files (*see screenshot above*) in the **`C:\temp`** folder and change the file names. To achieve this, we will create a configuration for each output. Take the steps below:

1. In the *Outputs* section of the **Global Resource** dialog box, click **Browse** next to the first output and enter **`C:\temp\file1.xml`** as a destination file name. This is the default configuration which will produce the first output file.
2. Click under *Configurations* and enter a name for the new configuration (in this example, `Output2`). In the *Outputs* section, click the radio button near the second file (**`SecondXML.xml`**).
3. In the *Outputs* section, click **Browse** next to the second output and enter **`C:\temp\file2.xml`** as a destination file name. This is the second configuration which produces the second output file.
4. Click **OK**.

Step 3: Use the global resource

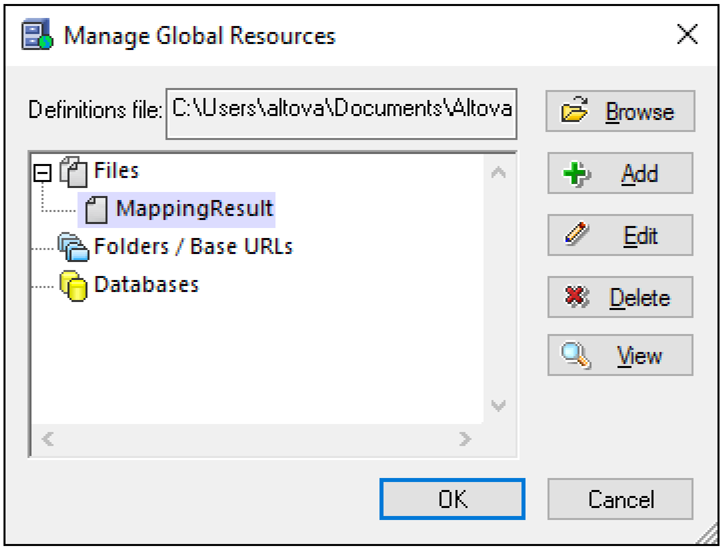
The instructions below show how to use the global resource we created in the previous step.

Default configuration

To use the default configuration in XMLSpy, take the steps below.

1. Run XMLSpy.

2. Go to the **Tools** menu and click **Global Resources**.
3. In the *Files* section, click the `MappingResult` global resource and then click **View** (*see screenshot below*). This executes the mapping, produces the default output (**`file1.xml`**) and loads it into the main pane of XMLSpy. The file is saved as **`C:\temp\file1.xml`**.



Second configuration
To trigger the mapping execution with the second configuration, do the following:

1. Go to the **Tools** menu in XMLSpy and click **Active Configuration | Output2**.
2. Click **Reload** when prompted.

As a result, the second output file is loaded into the main window of XMLSpy. The file is saved as **`C:\temp\file2.xml`**.

10.7 Global Resources in Execution Environments

This subsection explains how to work with global resources in different execution environments. The subsection is organized into the following topics:

- Global Resources in Generated Code 832
- Global Resources in MapForce Server 833
- Global Resources in FlowForce Server 833

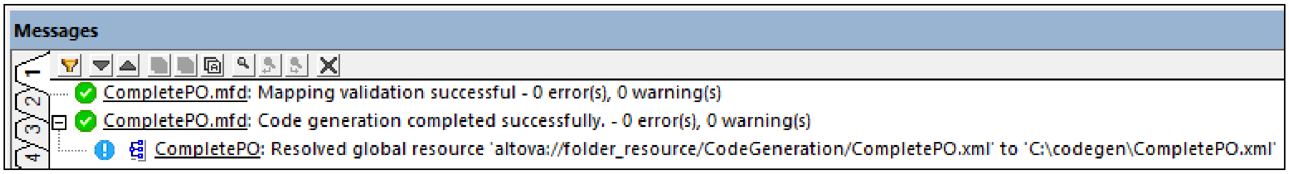
10.7.1 Global Resources in Generated Code

This topic explains how global resources are used in generated code. For more information, see the subsections below.

Global Resources in XSLT, XSLT 2, XQuery
When you generate XSLT or XSLT2 code and the mapping uses global resources, this does not affect the generated XSLT stylesheet in any way. With or without global resources, you can flexibly specify the input and output files when you run the XSLT stylesheet in your XSLT processor. The same applies to generated XQuery code.

An exception is the **DoTransform.bat** file generated for RaptorXML execution. Global resources used by the mapping will be resolved in actual paths in **DoTransform.bat**. The configuration which is currently selected from the global resources drop-down list will be taken into account. For information about supplying global resources to RaptorXML, see the RaptorXML documentation.

Global Resources in C++, C#, Java
When you generate C#, C++, or Java program code, global resources used by the mapping will be resolved. For example, a file or folder alias defined as a global resource will be converted to a file or folder path. If a particular global resource configuration is selected from the global resources drop-down list, the code will be generated for the selected configuration. The **Messages** window shows information about how exactly a global resource was resolved (*see screenshot below*).



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 resources drop-down list before generating code. Alternatively, if you generate code from the command line, supply the **GLOBALRESOURCEFILE** and **GLOBALRESOURCECONFIG** parameters at the command line. For details, see MapForce Command Line Interface 811.

It is not possible to switch or refer to global resources from generated code. Instead, you can modify the code to change the input or output file path.

Note: In C# or Java, you can change the path and the data type of input or output.

10.7.2 Global Resources in MapForce Server

When you compile a mapping to a MapForce Server execution file (**.mfx**), any global resource references used by the mapping are preserved, not resolved. This means that you will need to provide these references on the server side in order to run the mapping successfully. In MapForce Server, the following is required to run a **.mfx** file which uses global resources:

1. *The Global Resources Definitions file*. On the machine where MapForce is installed, the file is called **GlobalResources.xml**. You can find this file in the **Documents\Altova** folder. You can copy this file to the machine where MapForce Server runs and create multiple such files if necessary. See also Global Resources Setup Part 1 816.
2. *The Global Resource configuration name*. Each Global Resource has a default configuration. You can also create additional configurations. For more information, see Global Resources Setup Part 2 818.

In MapForce, the Global Resource Definitions file and the Global Resource configuration name are set or changed from the graphical user interface. In MapForce Server, these are specified at mapping runtime (*see below*).

- If you run the mapping through the command line interface, set the options **--globalresourceconfig** and **--globalresourcefile** after the run command, for example:

```
C:\Program Files (x86)\Altova\MapForceServer2025\bin\MapForceServer.exe run
SomeMapping.mfx --globalresourcefile="C:
\Users\me\Documents\Altova\GlobalResources.xml" --globalresourceconfig="Default"
```

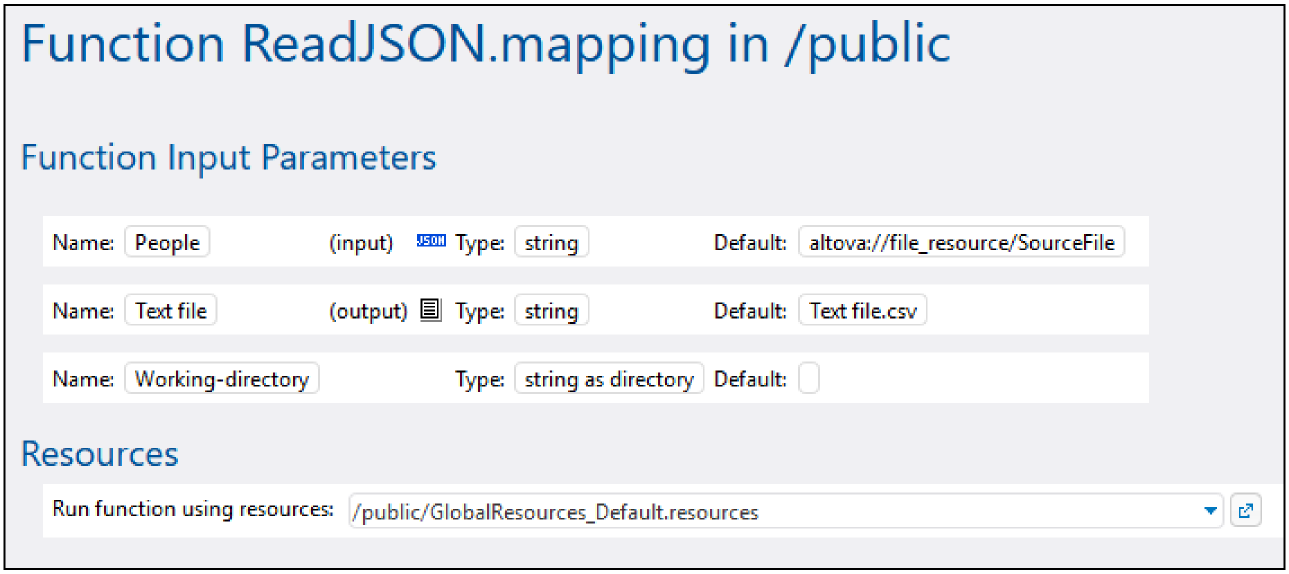
- If you run the mapping through the MapForce Server API, call the method **SetOptions** twice before calling the **Run** method. The first call is required to supply the Global Resource Definitions file path as an option, and the second call is required to supply the Global Resource configuration name.

For more information, see the MapForce Server documentation.

10.7.3 Global Resources in FlowForce Server

In FlowForce Server, global resources are not stored in one XML file as in desktop applications. In FlowForce, each resource is a reusable object that may contain file or folder paths or database connection details. Resources can be copied, exported, and imported, and are subject to the same user access mechanism as other FlowForce Server objects. This means that any FlowForce user can use any resource in their mapping functions if they have the required permissions.

Once you have created a mapping with global resources in MapForce, you can deploy it to FlowForce Server. At deployment time, if you want your mapping to use global resources, select the **Use Resources** check box in the deployment dialog box. If you do not select the check box, any global resources used by the mapping will be resolved, based on the currently selected configuration. If you have selected the check box, the mapping function will require resources in FlowForce Server as well. The screenshot below is an example of a mapping function deployed to FlowForce that requires resources to run. Notice that the first parameter gets the default file path from a resource.



Function ReadJSON.mapping in /public

Function Input Parameters

Name:	People	(input)	Type:	string	Default:	altova://file_resource/SourceFile
Name:	Text file	(output)	Type:	string	Default:	Text file.csv
Name:	Working-directory		Type:	string as directory	Default:	

Resources

Run function using resources: /public/GlobalResources_Default.resources

In FlowForce Server, it is the mapping function that uses the global resources, not the job. The mapping function reads the path of the first input file from the resource. This means that all jobs using this function will use the same path unless you override the path from the job configuration page.

You can also deploy global resources to FlowForce Server as standalone objects. This means there is no need to deploy a mapping first in order to be able to deploy a global resource. For more information, see *Deploying Resources to FlowForce Server* below.

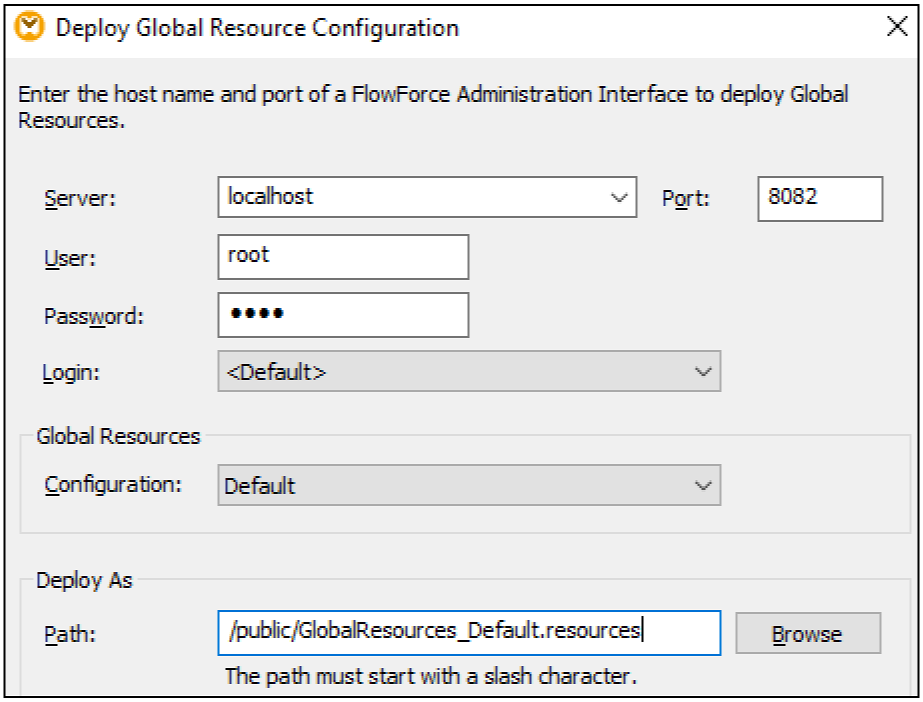
For information about consuming resources in FlowForce Server, see the FlowForce Server documentation.

Deploying resources to FlowForce Server

You can deploy global resources created with MapForce to FlowForce Server. Upon deployment, you must choose the configuration with which the resource should be deployed to the server. If you need all configurations of the same global resource on the server, you can deploy this global resource multiple times and select the desired configuration each time upon deployment. You can also change the name of each global resource on the server and choose the destination container on the server.

You can deploy global resources to FlowForce Server at the same time when you deploy the mapping or separately. To deploy global resources to FlowForce Server, take the steps below:

1. Run MapForce.
2. Click the **Manage Global Resources** toolbar button. Alternatively, go to the **Tools** menu and click **Global Resources**.
3. Click **Deploy To Server**. This opens the **Deploy Global Resource Configuration** dialog box (*see screenshot below*).



Deploy Global Resource Configuration

Enter the host name and port of a FlowForce Administration Interface to deploy Global Resources.

Server: localhost

Port: 8082

User: root

Password: ●●●●

Login: <Default>

Global Resources

Configuration: Default

Deploy As

Path: /public/GlobalResources_Default.resources

The path must start with a slash character.

4. Enter the connection details to FlowForce Server (server, port, user, password, login method). These parameters are the same as the ones required when you deploy mappings to FlowForce Server 802.
5. Select a configuration from the **Configuration** list. This list includes all configurations from the current Global Resources Definitions file 816. Note that only one global resource configuration can be deployed at a time. You can deploy the same resource multiple times, with a different name, if you need all the configurations on the server.
6. Select a target path where the resource should be saved on the server. Click **Browse** to select a target FlowForce container or create a new one if required.
7. Click **OK**.

You can see information about deploying the global resource to FlowForce Server in the **Messages** window.

Note: Global resources that run other Altova applications are *not* supported in a server environment. For more information, see Transformation Results as Global Resource 828.

11 Catalogs in MapForce

MapForce supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables MapForce to retrieve commonly used schemas (as well as other files) from local user folders. This increases the overall processing speed, enables users to work offline (that is, not connected to a network), and improves the portability of documents (because URIs would then need to be changed only in the catalog files.)

The catalog mechanism in MapForce works as outlined in this section:

- How Catalogs Work 837
- Catalog Structure in MapForce 839
- Customizing Your Catalogs 841
- Environment Variables 843

For more information on catalogs, see the XML Catalogs specification.

11.1 How Catalogs Work

Catalogs can be used to redirect both DTDs and XML Schemas. While the concept behind the mechanisms of both cases is the same, the details are different and are explained below.

DTDs

Catalogs are commonly used to redirect a call to a DTD to a local URI. This is achieved by mapping, in the catalog file, public or system identifiers to the required local URI. So when the **DOCTYPE** declaration in an XML file is read, its public or system identifier locates the required local resource via the catalog file mapping.

For popular schemas, the **PUBLIC** identifier is usually pre-defined, thus requiring only that the URI in the catalog file map the **PUBLIC** identifier to the correct local copy. When the XML document is parsed, the **PUBLIC** identifier in it is read. If this identifier is found in a catalog file, then the corresponding URL in the catalog file will be looked up and the schema will be read from this location. So, for example, if the following SVG file is opened in MapForce:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="20" height="20" xml:space="preserve">
  <g style="fill:red; stroke:#000000">
     <rect x="0" y="0" width="15" height="15"/>
     <rect x="5" y="5" width="15" height="15"/>
  </g>
</svg>
```

The catalog is searched for the **PUBLIC** identifier of this SVG file. Let's say the catalog file contains the following entry:

```
<catalog>
  ...
    <public publicId="-//W3C//DTD SVG 1.1//EN" uri="schemas/svg/svg11.dtd"/>
  ...
</catalog>
```

In this case, there is a match for the **PUBLIC** identifier. As a result, the lookup for the SVG DTD is redirected to the URL **schemas/svg/svg11.dtd** (which is relative to the catalog file). This is a local file that will be used as the DTD for the SVG file. If there is no mapping for the **Public** ID in the catalog, then the URL in the XML document will be used (in the SVG fie example above, this is the Internet URL: **http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd**).

XML Schemas

In MapForce, you can also use catalogs with **XML Schemas**. In the XML instance file, the reference to the schema will occur in the **xsi:schemaLocation** attribute of the XML document's top-level element. For example,

```
xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart OrgChart.xsd"
```

The value of the **xsi:schemaLocation** attribute has two parts: a namespace part (green above) and a URI part (highlighted). The namespace part is used in the catalog to map to the alternative resource. For example, the following catalog entry redirects the schema reference above to a schema at an alternative location.

```
<uri name="http://www.xmlspy.com/schemas/orgchart" uri="C:\MySchemas\OrgChart.xsd"/>
```

Normally, the URI part of the **`xsi:schemaLocation`** attribute's value is a path to the actual schema location. However, if the schema is referenced via a catalog, the URI part need not point to an actual XML Schema but must exist so that the lexical validity of the **`xsi:schemaLocation`** attribute is maintained. A value of `foo`, for example, would be sufficient for the URI part of the attribute's value to be valid.

11.2 Catalog Structure in MapForce

When MapForce starts, it loads a file called **`RootCatalog.xml`** (*structure shown in listing below*), which contains a list of catalog files that will be looked up. You can modify this file and enter as many catalog files to look up as you like, each of which is referenced in a **`nextCatalog`** element. These catalog files are looked up and the URIs in them are resolved according to their mappings.

Listing of RootCatalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
   xmlns:spy="http://www.altova.com/catalog_ext"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog Catalog.xsd">
  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/CustomCatalog.xml"/>
  <!-- Include all catalogs under common schemas folder on the first directory level -->
  <nextCatalog spy:recurseFrom="%CommonSchemasFolder%" catalog="catalog.xml"
spy:depth="1"/>
  <nextCatalog spy:recurseFrom="%ApplicationWritableDataFolder%/pkgs/.cache"
catalog="remapping.xml" spy:depth="0"/>
  <nextCatalog catalog="CoreCatalog.xml"/>
</catalog>
```

The listing above references a custom catalog (named **`CustomCatalog.xml`**) and a set of catalogs that locate commonly used schemas (such as W3C XML Schemas and the SVG schema).

- **`CustomCatalog.xml`** is located in your Personal Folder (located via the variable **`%PersonalFolder%`**). It is a skeleton file in which you can create your own mappings. You can add mappings to **`CustomCatalog.xml`** for any schema you require that is not addressed by the catalog files in the Common Schemas Folder. Do this by using the supported elements of the OASIS catalog mechanism (*see next section*).
- The Common Schemas Folder (located via the variable **`%CommonSchemasFolder%`**) contains a set of commonly used schemas. Inside each of these schema folders is a **`catalog.xml`** file that maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.
- **`CoreCatalog.xml`** is located in the MapForce application folder, and is used to locate schemas and stylesheets used by MapForce-specific processes, such as StyleVision Power Stylesheets which are stylesheets used to generate Altova's Authentic View of XML documents.

Location variables

The variables that are used in **`RootCatalog.xml`** (*listing above*) have the following values:

`%PersonalFolder%`	Personal folder of the current user, for example `C:\Users\<name>\Documents`
`%CommonSchemasFolder%`	`C:\ProgramData\Altova\Common2025\Schemas`
`%ApplicationWritableDataFolder%`	`C:\ProgramData\Altova`

Location of catalog files and schemas

Note the locations of the various catalog files.

- **`RootCatalog.xml`** and **`CoreCatalog.xml`** are in the MapForce application folder.
- **`CustomCatalog.xml`** is located in your **`MyDocuments\Altova\MapForce`** folder.
- The **`catalog.xml`** files are each in a specific schema folder, these schema folders being inside the Common Schemas Folder.

11.3 Customizing Your Catalogs

When creating entries in **`CustomCatalog.xml`** (or any other catalog file that is to be read by MapForce), use only the following elements of the OASIS catalog specification. Each of the elements below is listed with an explanation of their attribute values. For a more detailed explanation, see the XML Catalogs specification. Note that each element can take the **`xml:base`** attribute, which is used to specify the base URI of that element.

- `<public publicId="PublicID of Resource" uri="URL of local file"/>`
- `<system systemId="SystemID of Resource" uri="URL of local file"/>`
- `<uri name="filename" uri="URL of file identified by filename"/>`
- `<rewriteURI uriStartString="StartString of URI to rewrite" rewritePrefix="String to replace StartString"/>`
- `<rewriteSystem systemIdStartString="StartString of SystemID" rewritePrefix="Replacement string to locate resource locally"/>`

Note the following points:

- In cases where there is no public identifier, as with most stylesheets, the system identifier can be directly mapped to a URL via the **`system`** element.
- A URI can be mapped to another URI using the `uri` element.
- The `rewriteURI` and `rewriteSystem` elements enable the rewriting of the starting part of a URI or system identifier, respectively. This allows the start of a filepath to be replaced and consequently enables the targeting of another directory. For more information on these elements, see the XML Catalogs specification.

From release 2014 onwards, MapForce adheres closely to the XML Catalogs specification (OASIS Standard V1.1, 7 October 2005) specification. This specification strictly separates external-identifier look-ups (those with a Public ID or System ID) from URI look-ups (URIs that are not Public IDs or System IDs). Namespace URIs must therefore be considered simply URIs—not Public IDs or System IDs—and must be used as URI look-ups rather than external-identifier look-ups. In MapForce versions prior to version 2014, schema namespace URIs were translated through **`<public>`** mappings. From version 2014 onwards, **`<uri>`** mappings have to be used.

Prior to v2014: `<public publicID="http://www.MyMapping.com/ref" uri="file:///C:/MyDocs/Catalog/test.xsd"/>`
V-2014 onwards: `<uri name="http://www.MyMapping.com/ref" uri="file:///C:/MyDocs/Catalog/test.xsd"/>`

How MapForce finds a referenced schema

A schema is referenced in an XML document via the **`xsi:scemaLocation`** attribute (*shown below*). The value of the **`xsi:schemaLocation`** attribute has two parts: a namespace part (green) and a URI part (highlighted).

```
xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart OrgChart.xsd"
```

Given below are the steps, followed sequentially by MapForce, to find a referenced schema. The schema is loaded at the first successful step.

1. Look up the catalog for the URI part of the **`xsi:schemaLocation`** value. If a mapping is found, including in **`rewriteURI`** mappings, use the resulting URI for schema loading.
2. Look up the catalog for the namespace part of the **`xsi:schemaLocation`** value. If a mapping is found,

including in **`rewriteURI`** mappings, use the resulting URI for schema loading.

3. Use the URI part of the **`xsi:schemaLocation`** value for schema loading.

XML Schema specifications

XML Schema specification information is built into MapForce and the validity of XML Schema (`.xsd`) documents is checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema that defines the XML Schema specification.

The **`catalog.xml`** file in the `%AltovaCommonSchemasFolder%\Schemas\schema` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against these schemas. The referenced files are included solely to provide MapForce with entry helper info for editing purposes should you wish to create documents according to these older recommendations.

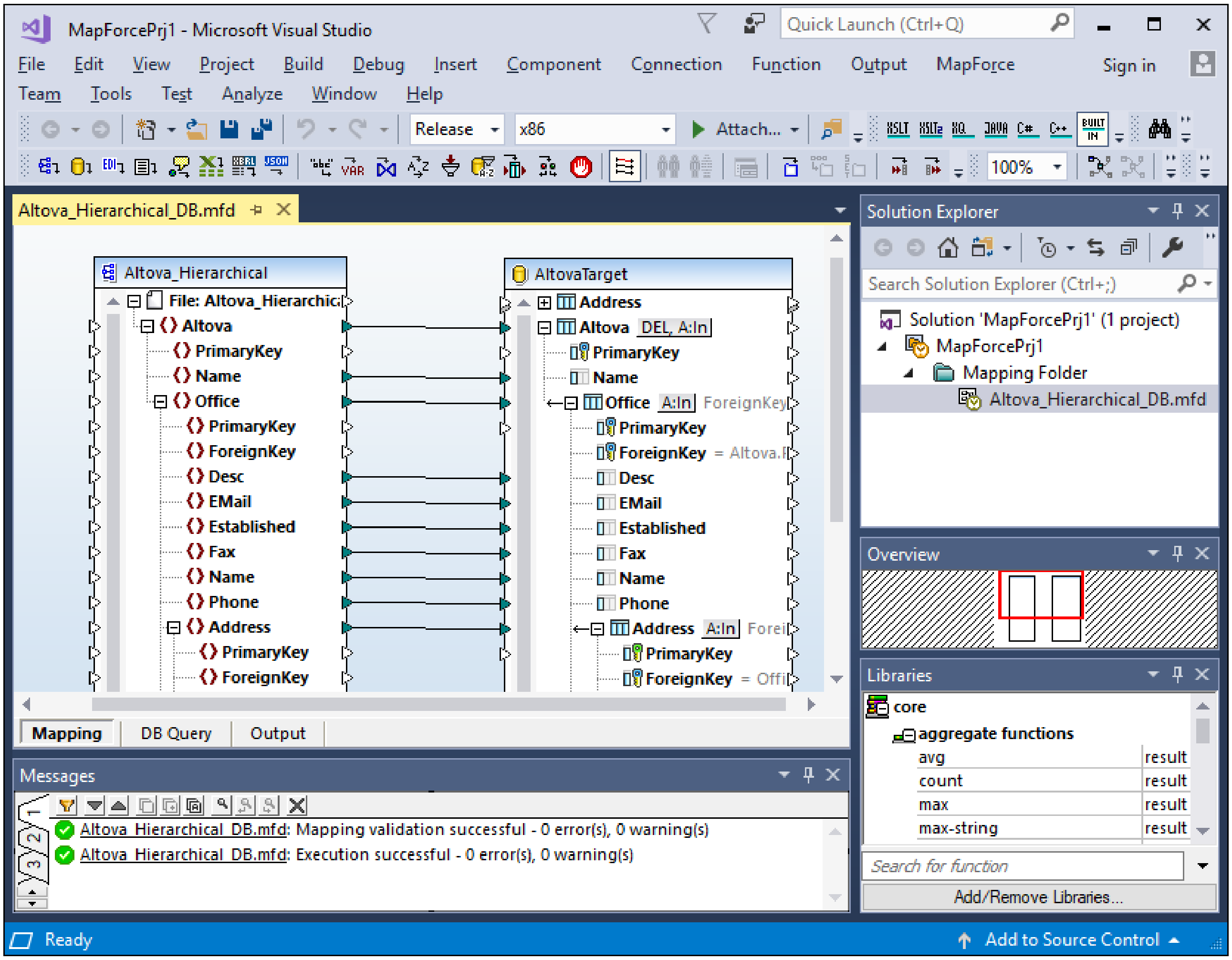
11.4 Environment Variables

Shell environment variables can be used in the `nextCatalog` element to specify the path to various system locations (*see RootCatalog.xml listing above*). The following shell environment variables are supported:

`%PersonalFolder%`	Full path to the Personal folder of the current user, for example `C:\Users\<name>\Documents`
`%CommonSchemasFolder%`	`C:\ProgramData\Altova\Common2025\Schemas`
`%ApplicationWritableDataFolder%`	`C:\ProgramData\Altova`
`%AltovaCommonFolder%`	C:\Program Files\Altova\Common2025
`%DesktopFolder%`	Full path to the Desktop folder of the current user.
`%ProgramMenuFolder%`	Full path to the Program Menu folder of the current user.
`%StartMenuFolder%`	Full path to Start Menu folder of the current user.
`%StartUpFolder%`	Full path to Start Up folder of the current user.
`%TemplateFolder%`	Full path to the Template folder of the current user.
`%AdminToolsFolder%`	Full path to the file system directory that stores administrative tools of the current user.
`%AppDataFolder%`	Full path to the Application Data folder of the current user.
`%CommonAppDataFolder%`	Full path to the file directory containing application data of all users.
`%FavoritesFolder%`	Full path of the Favorites folder of the current user.
`%PersonalFolder%`	Full path to the Personal folder of the current user.
`%SendToFolder%`	Full path to the SendTo folder of the current user.
`%FontsFolder%`	Full path to the System Fonts folder.
`%ProgramFilesFolder%`	Full path to the Program Files folder of the current user.
`%CommonFilesFolder%`	Full path to the Common Files folder of the current user.
`%WindowsFolder%`	Full path to the Windows folder of the current user.
`%SystemFolder%`	Full path to the System folder of the current user.
`%LocalAppDataFolder%`	Full path to the file system directory that serves as the data repository for local (nonroaming) applications.
`%MyPicturesFolder%`	Full path to the MyPictures folder.

12 MapForce Plug-in for Visual Studio

You can integrate MapForce 2025 into the Microsoft Visual Studio versions 2012/2013/2015/2017/2019/2022. This integration helps combine the mapping capabilities of MapForce with the development environment of Visual Studio. When the MapForce plug-in is enabled, you can create mappings and mapping projects directly in Visual Studio (*see screenshot below*).



Altova_Hierarchical_DB.mfd: Mapping validation successful - 0 error(s), 0 warning(s)

Altova_Hierarchical_DB.mfd: Execution successful - 0 error(s), 0 warning(s)

Installation

To install the MapForce Plug-in for Visual Studio, take the steps below:

1. Install Microsoft Visual Studio 2012/2013/2015/2017/2019/2022. Note that from Visual Studio 2022 onwards, Visual Studio is being made available only as a 64-bit application.
2. Install MapForce (Enterprise or Professional Edition). If you have installed Visual Studio 2022+, then you must install the 64-bit version of MapForce.
3. Download and run the MapForce integration package for Microsoft Visual Studio. This package is available on the MapForce (Enterprise and Professional Editions) download page at www.altova.com.

Once the integration package has been installed, you will be able to use MapForce in the Visual Studio environment.

> **Important**
>
> You must use the integration package corresponding to your MapForce version (current version is 2025). The integration package is not edition-specific and can therefore be used for both Enterprise and Professional editions.

Information about menus and functions

When the MapForce plug-in for Visual Studio is enabled, you can access different MapForce menus and functions (*see below*). You can customize MapForce menus and toolbars from the **Tools | Customize** menu of Visual Studio.

Note: In Visual Studio 2019 and later, MapForce functionality can be accessed in the **Extensions** menu of Visual Studio. In earlier versions of Visual Studio, MapForce features are available in the top-level menus of Visual Studio.

⊟ Themes

You can select MapForce themes in the **MapForce** menu of Visual Studio. The options are Classic, Light, and Dark themes.

⊟ Create and open files/projects

When the MapForce plug-in for Visual Studio is enabled, you can create, open, and work with mappings and mapping projects directly in Visual Studio. To create a new mapping design file in Visual Studio, use the **File | New** menu command. To create a new project, use the **File | New Project** menu command. To open existing mapping files or projects, you can use the following Visual Studio menus: **File | Open | File** or **File | Open | Project/Solution**. Then you can look for the MapForce-related file types.

⊟ Global Resources

MapForce Global Resources[815] are available in the **MapForce | Manage Global Resources** menu of Visual Studio. In Visual Studio 2019 onwards, the corresponding menu is **Extensions | MapForce | Manage Global Resources**.

⊟ Debugging

After you have opened a mapping file, the mapping debugging commands become available in the Debug menu[1070] and in the **Debug** toolbar. In Visual Studio 2019 onwards, the corresponding menu is **Extensions | MapForce | Debug**.

⊟ MapForce options

MapForce options are available in the **Tools | MapForce Options** menu of Visual Studio.

⊟ Mapping pane customization

When you open a mapping in the main pane of Visual Studio, the **View | MapForce** menu becomes available. It includes the same options as the **View** menu of the standalone version of MapForce.

⊟ Libraries window

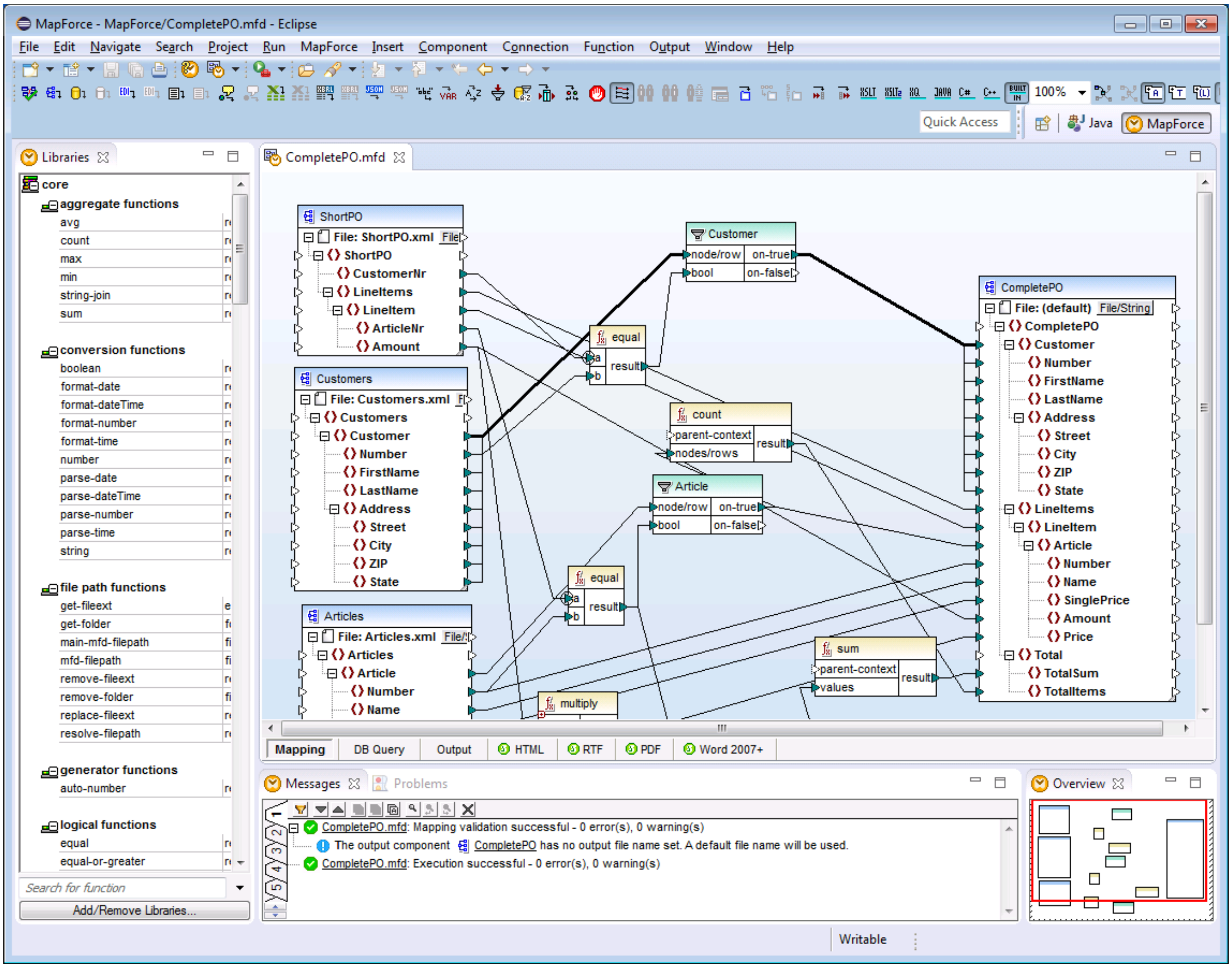
If the MapForce **Libraries** window is not visible in Visual Studio, you can enable it from the **View | MapForce | Libraries Window** menu (this menu becomes available in Visual Studio when a mapping file is open). Once the **Libraries** window is enabled, you can dock it to a particular position in the interface.

⊟ Help and Support

The MapForce menus **Help**, **Support Center**, **Check for Updates** and **About** are available in the **Help | MapForce Help** menu of Visual Studio.

13 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 2024-12 (4.34), 2024-09 (4.33), 2024-06 (4.32), 2024-03 (4.31) 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 848
- The MapForce Perspective 850
- Accessing Common Menus and Functions 853
- Working with Mapping and Projects 857
- Extending MapForce Plug-in for Eclipse 867

13.1 Installing the MapForce Plug-in for Eclipse

Prerequisites

- Eclipse 2024-12 (4.34), 2024-09 (4.33), 2024-06 (4.32), 2024-03 (4.31) (http://www.eclipse.org), 64-bit.
- A Java Runtime Environment (JRE) or Java Development Kit (JDK) for the 64-bit platform.
- MapForce Enterprise or Professional Edition 64-bit.

Note: All the prerequisites listed above must have the 64-bit platform. Integration with older Eclipse 32-bit platforms is no longer supported, although it may still work.

After the prerequisites listed above are in place, you can install the MapForce Integration Package (64-bit) to integrate MapForce in Eclipse. The integration can be carried out either during the installation of the Integration Package or manually from Eclipse after the Integration Package has been installed. The MapForce Integration Package is available for download at https://www.altova.com/components/download.

Note: Eclipse must be closed while you install or uninstall the MapForce Integration Package.

Integrate MapForce during installation of the Integration Package

You can integrate MapForce in Eclipse during the installation of the MapForce Integration Package. Do this as follows:

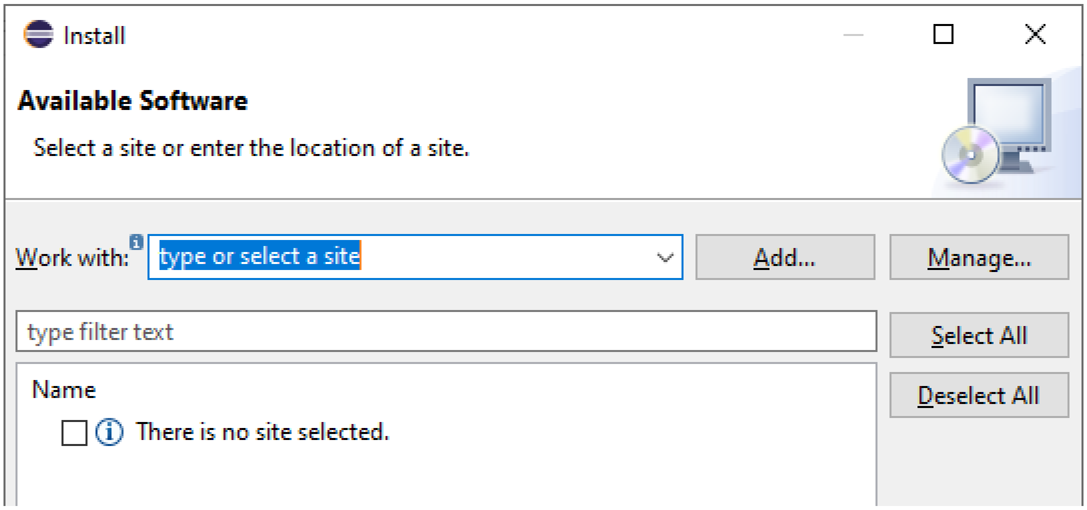
1. Run the MapForce Integration Package to start the installation wizard.
2. Go through the initial steps of the installation with eth wizard.
3. In the Integration step, select *Let this wizard integrate Altova MapForce plug-in into Eclipse*, and browse for the directory where the Eclipse executable (`eclipse.exe`) is located.
4. Click **Next** and complete the installation.

The MapForce perspective and menus will be available in Eclipse the next time you start it.

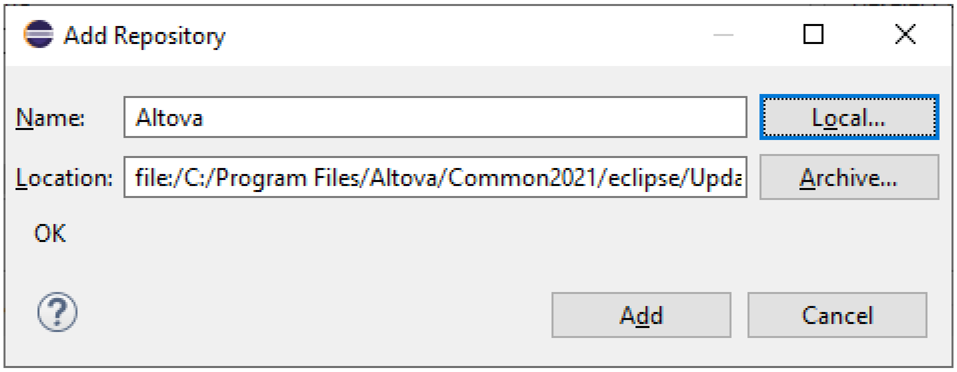
Integrate MapForce in Eclipse manually

After you have installed the MapForce Integration Package, you can manually integrate MapForce in Eclipse as follows:

1. In Eclipse, select the menu command **Help | Install New Software**.
2. In the Install dialog box, click **Add**.



3. In the Add Repository dialog box, click **Local**. Browse for the folder `C:\Program Files\Altova\Common2025\eclipse\UpdateSite`, and select it. Provide a name for the site (such as "Altova").



Name: Altova

Location: file:/C:/Program Files/Altova/Common2021/eclipse/Upda

OK

4. Repeat the steps 2-3 above, this time selecting the folder `C:\Program Files\Altova\<%APPNAMESHORT%>\eclipse\UpdateSite` and providing a name such as "Altova MapForce".
5. On the Install dialog box, select *Only Local Sites*. Next, select the "Altova category" folder and click **Next**.
6. Review the items to be installed and click **Next** to proceed.
7. To accept the license agreement, select the respective check box.
8. Click **Finish** to complete the installation.

Note: If there are problems with the plug-in (missing icons, for example), start Eclipse from the command line with the `-clean` flag.

13.2 The MapForce Perspective

In Eclipse, a perspective is a GUI view that is configured with the functionality of a specific application. After MapForce has been integrated in Eclipse, a new perspective, named MapForce, becomes available in Eclipse. This perspective is a GUI that resembles the MapForce GUI and includes a number of its components.

When a file having a filetype associated with MapForce is opened (`.mfd`), this file can be edited in the MapForce perspective. Similarly, a file of another filetype can be opened in another perspective in Eclipse. Additionally, for any active file, you can switch the perspective (*see below*), thus allowing you to edit or process that file in another environment.

There are therefore two main advantage of perspectives:

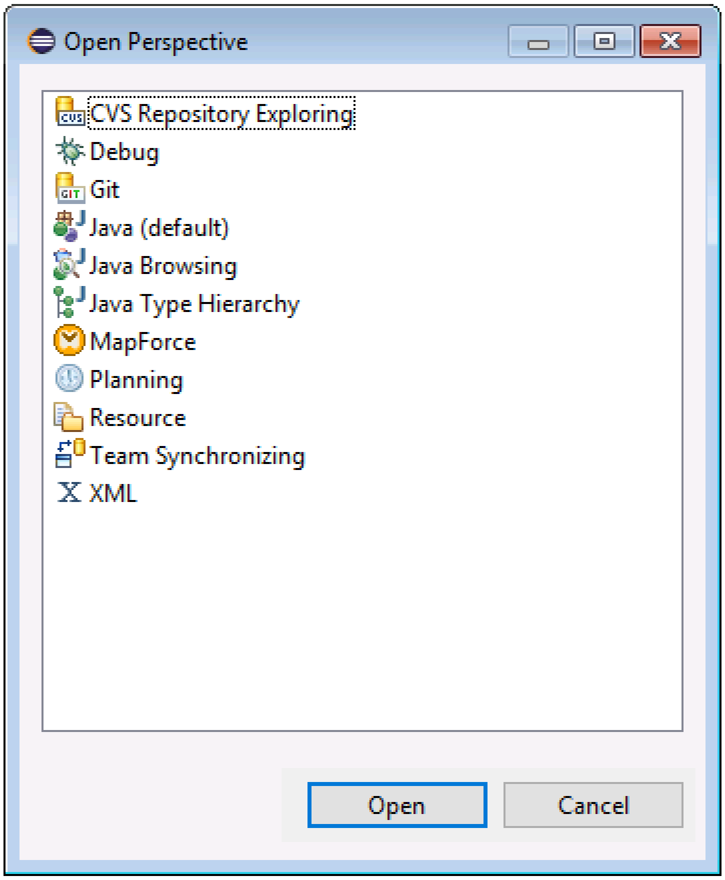
1. Being able to quickly change the working environment of the active file, and
2. Being able to switch between files without having to open a new development environment (the associated environment is available in a perspective)

Working with the MapForce perspective involves the following key procedures, which are described further below:

- Switching to the MapForce perspective.
- Setting preferences for the MapForce perspective.
- Customizing the MapForce perspective.

Switch to the MapForce perspective

In Eclipse, select the command **Window | Perspective | Open Perspective | Other**. In the dialog that appears (*screenshot below*), select **MapForce**, and click **Open**.



The empty window or the active document will now have the MapForce perspective. This is how the user switches the perspective via the menu. To access a perspective faster from another perspective, you can set the required perspective to be listed in the **Open Perspective** submenu, above the **Other** item. This setting is in the customization dialog (*see further below*).

Perspectives can also be switched when a file is opened or made active. The perspective of the application associated with a file's filetype will be automatically opened when that file is opened for the first time. Before the perspective is switched, a dialog appears asking whether you wish to have the default perspective automatically associated with this filetype. Check the *Do Not Ask Again* option if you wish to associate the perspective with the filetype without having to be prompted each time a file of this filetype is opened and then click **OK**.

Preferences for the MapForce perspective

To access the Preferences of a perspective, select the command **Window | Preferences**. In the list of perspectives in the left pane, select MapForce, then select the required preferences. Finish by clicking **OK**.

The preferences of a perspective include:

- To automatically switch to the MapForce perspective when a file of an associated filetype is opened (*see above*)
- Options for including or excluding individual MapForce toolbars
- Access to MapForce options.

Customize the MapForce perspective

The customization options enable you to determine what shortcuts and commands are included in the perspective. To access the Customize Perspective dialog of a perspective, make that perspective the active perspective and select the command **Window | Perspective | Customize Perspective**.

- In the *Toolbar Visibility* and *Menu Visibility* tabs, you can specify which toolbars and menus are to be displayed.
- In the *Action Set Availability* tab, you can add action sets to their parent menus and to the toolbar. If you wish to enable an action group, check its check box.
- In the *Shortcuts* tab of the Customize Perspective dialog, you can set shortcuts for submenus. Select the required submenu in the Submenus combo box. Then select a shortcut category, and check the shortcuts you wish to include for the perspective.

Click **Apply and Close** to complete the customization and for the changes to take effect.

Overview of the MapForce perspective

By default, the MapForce perspective in Eclipse is organized as follows:

- The mapping design window is available as an Eclipse editor. It has the same tabs and functionality as in the standalone edition of MapForce.
- The Libraries window is available as an Eclipse view, to the left of the main mapping editor. If this view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Libraries**. The Libraries view enables you to work with predefined or custom-defined functions and function libraries.
- The Messages pane is available as an Eclipse view, under the main mapping editor. If the Message view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Messages**. The messages view displays validation messages, errors, and warnings.
- The Overview pane is available as an Eclipse view. If the Overview view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Overview**. This view enables you to quickly navigate to a particular region on the mapping design area when it is very big.

13.3 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 850).

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 1055.

General MapForce commands

In the standalone edition of MapForce, the commands applicable to mapping design files (such as **Validate**, **Deploy to FlowForce Server**, **Generate Code**, and others) are available in the **File** menu. In Eclipse, these commands are available in the **MapForce** menu, or in the MapForce toolbar. Note that the commands for opening or saving files (including MapForce project files) are available in the **File** menu of Eclipse.

MapForce themes can be selected in the **MapForce** menu.

The MapForce toolbar in Eclipse

The toolbar button opens the MapForce help file.

The toolbar button displays commands specific to MapForce files. When you expand this button, the available commands depend on the kind of file currently active in the Eclipse editor. For example, the commands specific to mapping design (.mfd) files are available when such a file is active (in focus) in the Eclipse editor.

Global Resources

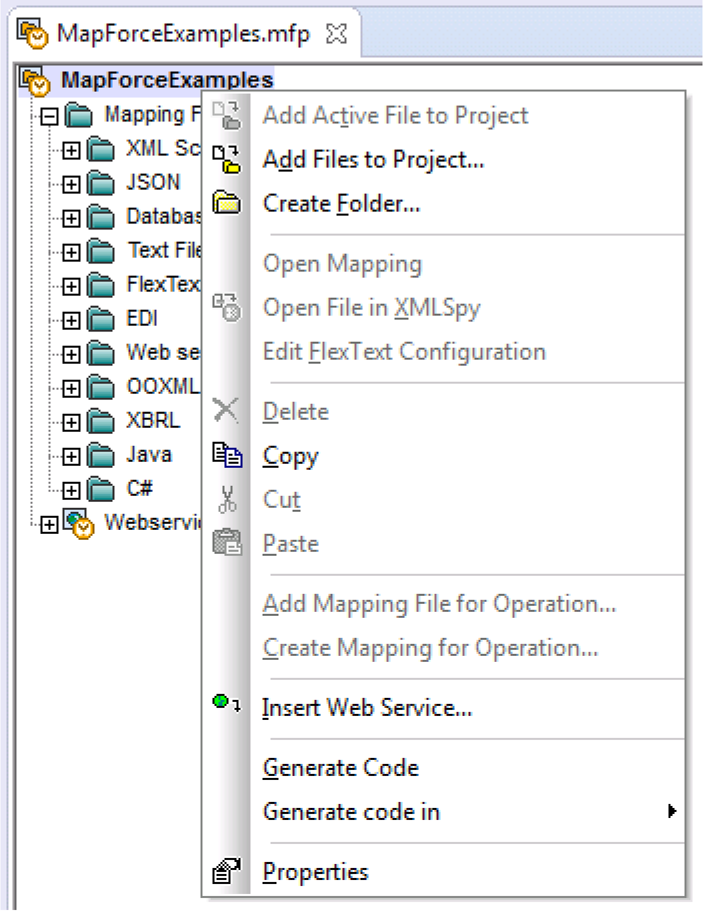
To access or manage Global Resources, do one of the following:

- 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:

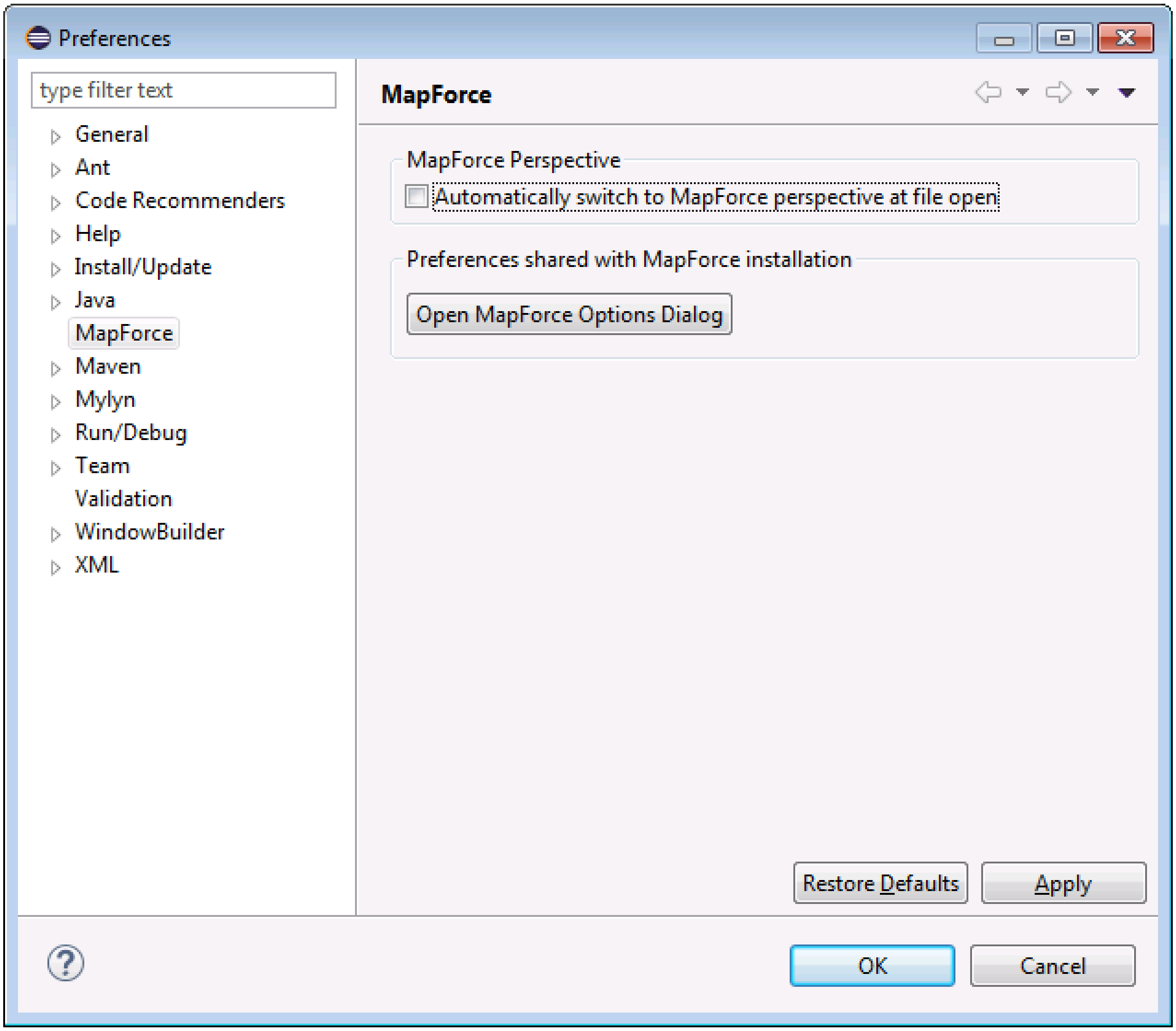
- The commands to open or save a project are available from the Eclipse **File** menu.
- Other project-specific commands are available as context commands. To display the context commands, create or open a MapForce project (.mfp) file in Eclipse, and then right-click the project.



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 857.

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**.



MapForce Perspective

☐ Automatically switch to MapForce perspective at file open

Preferences shared with MapForce installation

Preferences dialog box

Libraries window

In Eclipse, the MapForce Libraries window is available as a view. This view is by default located to the left of the main editor window. (All MapForce-related views become visible in the Eclipse interface when the MapForce perspective is switched on, see also The MapForce Perspective[850]).

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.

13.4 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.

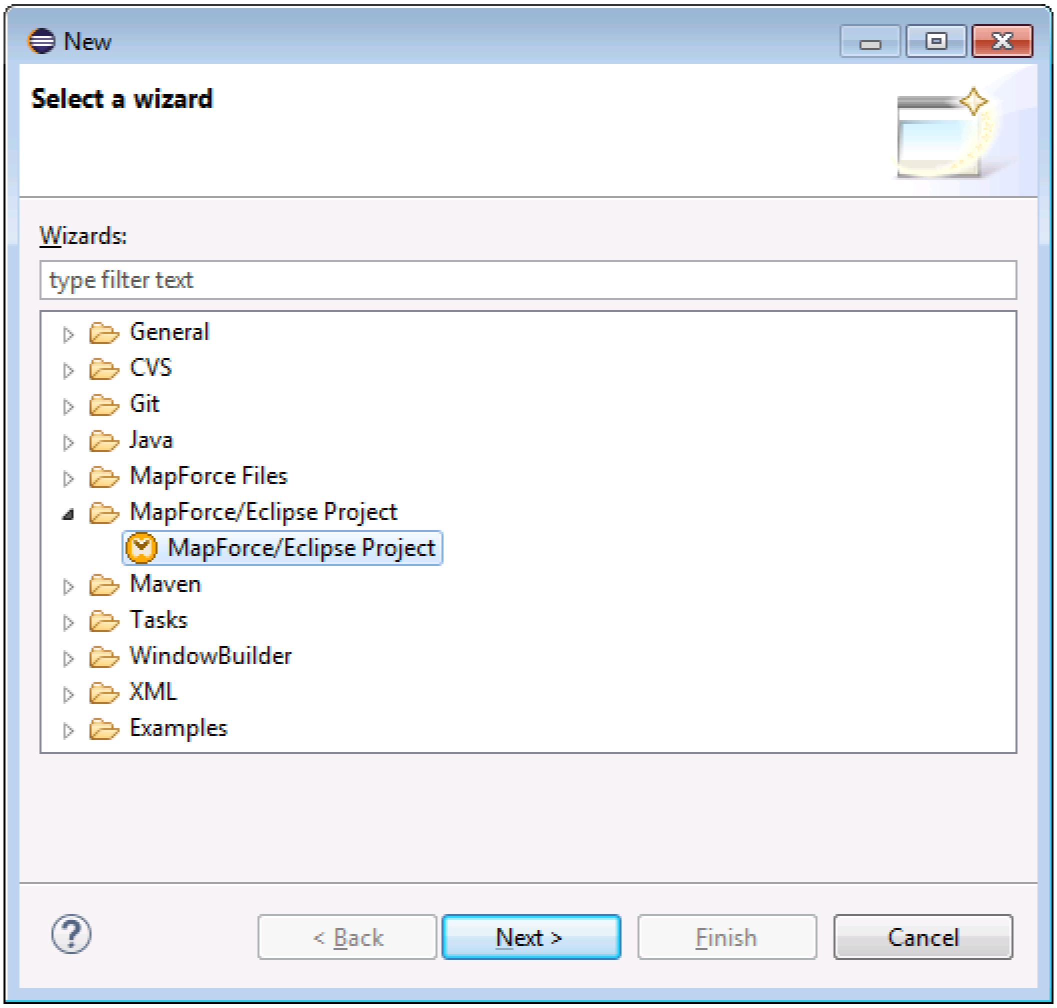
This section contains the following topics:

- Creating a MapForce/Eclipse Project 857
- Creating New Mappings 859
- Importing Existing Mappings into an Eclipse Project 861
- Configuring Automatic Build and Generation of MapForce Code 864

13.4.1 Creating a MapForce/Eclipse Project

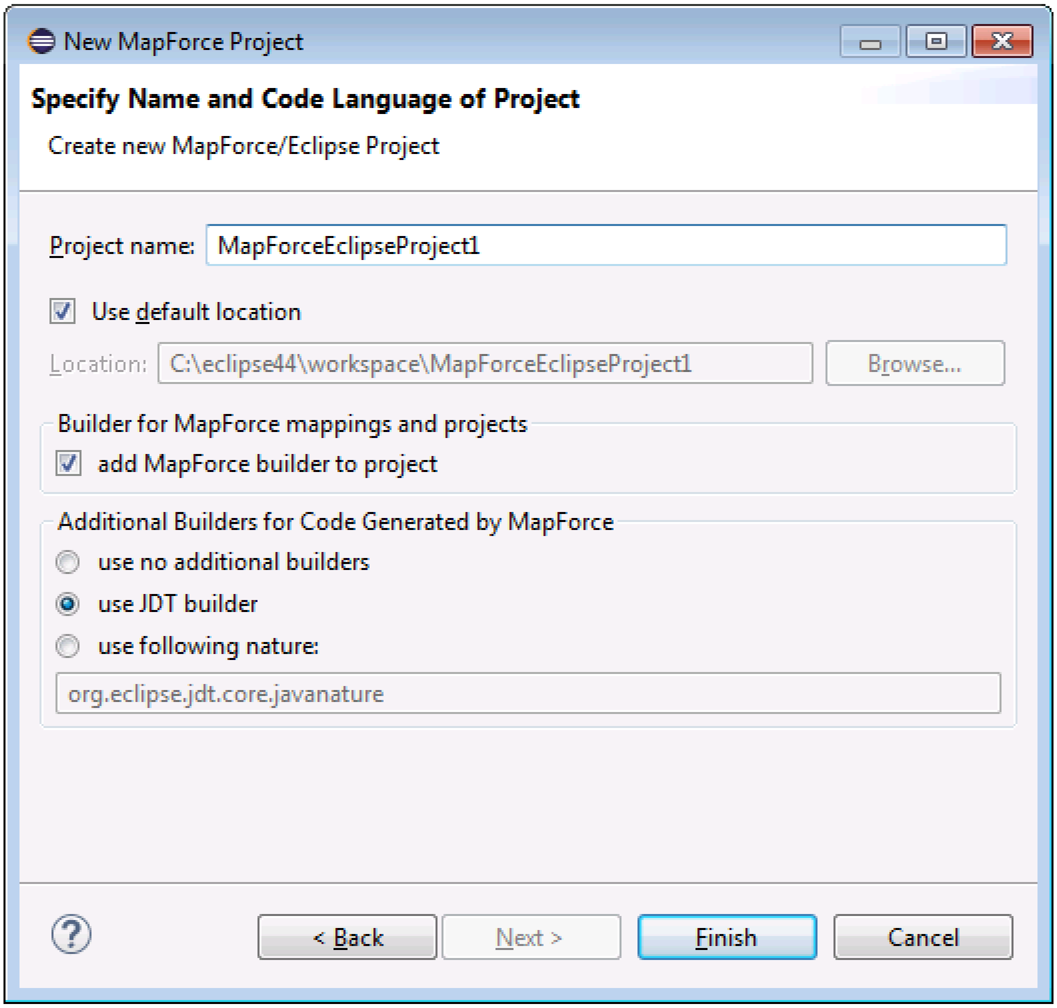
To create a MapForce/Eclipse project, take the steps below:

1. On the **File** menu, click **New | Other**.
2. Select the **MapForce/Eclipse Project** category.



New
Select a wizard
Wizards:
type filter text
General
CVS
Git
Java
MapForce Files
MapForce/Eclipse Project
MapForce/Eclipse Project
Maven
Tasks
WindowBuilder
XML
Examples
< Back
Next >
Finish
Cancel

3. Click **Next**.



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**.

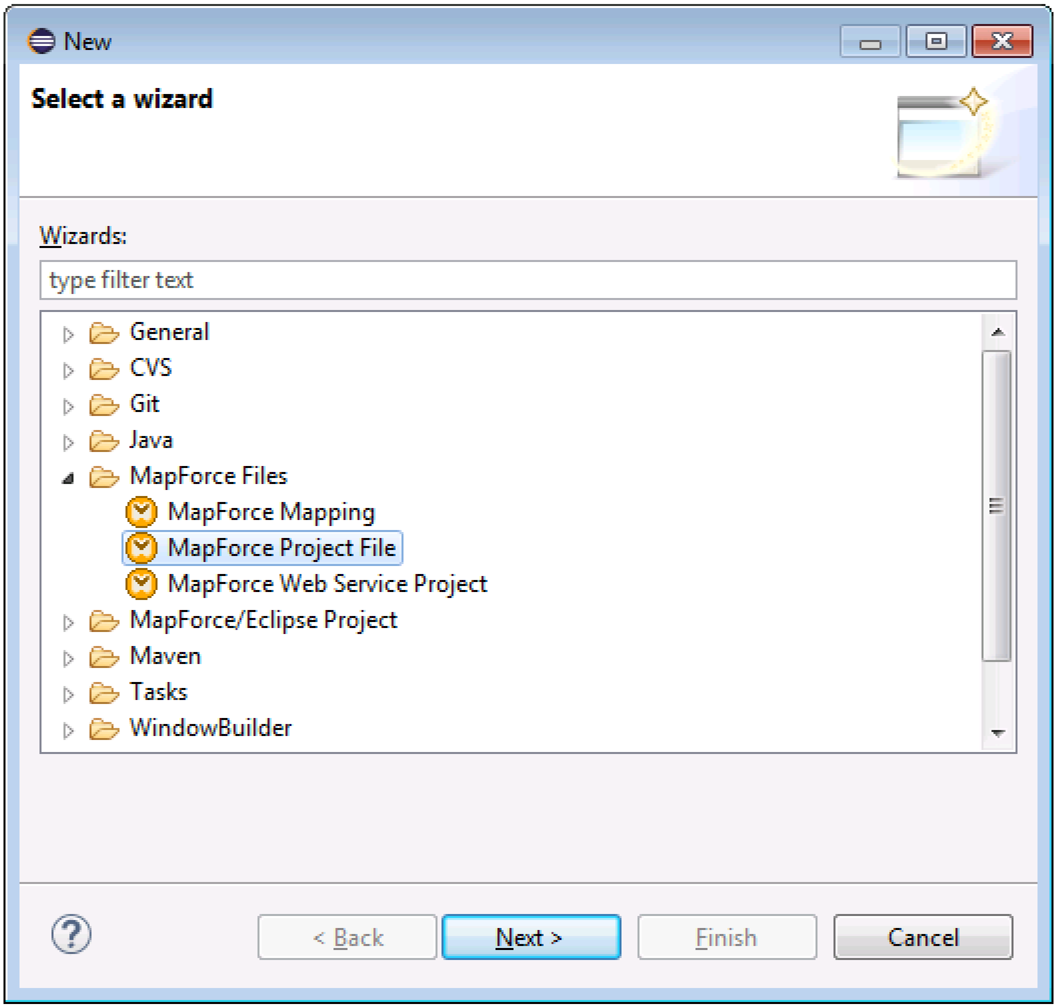
13.4.2 Creating New Mappings

You can create the following MapForce file types within an Eclipse project:

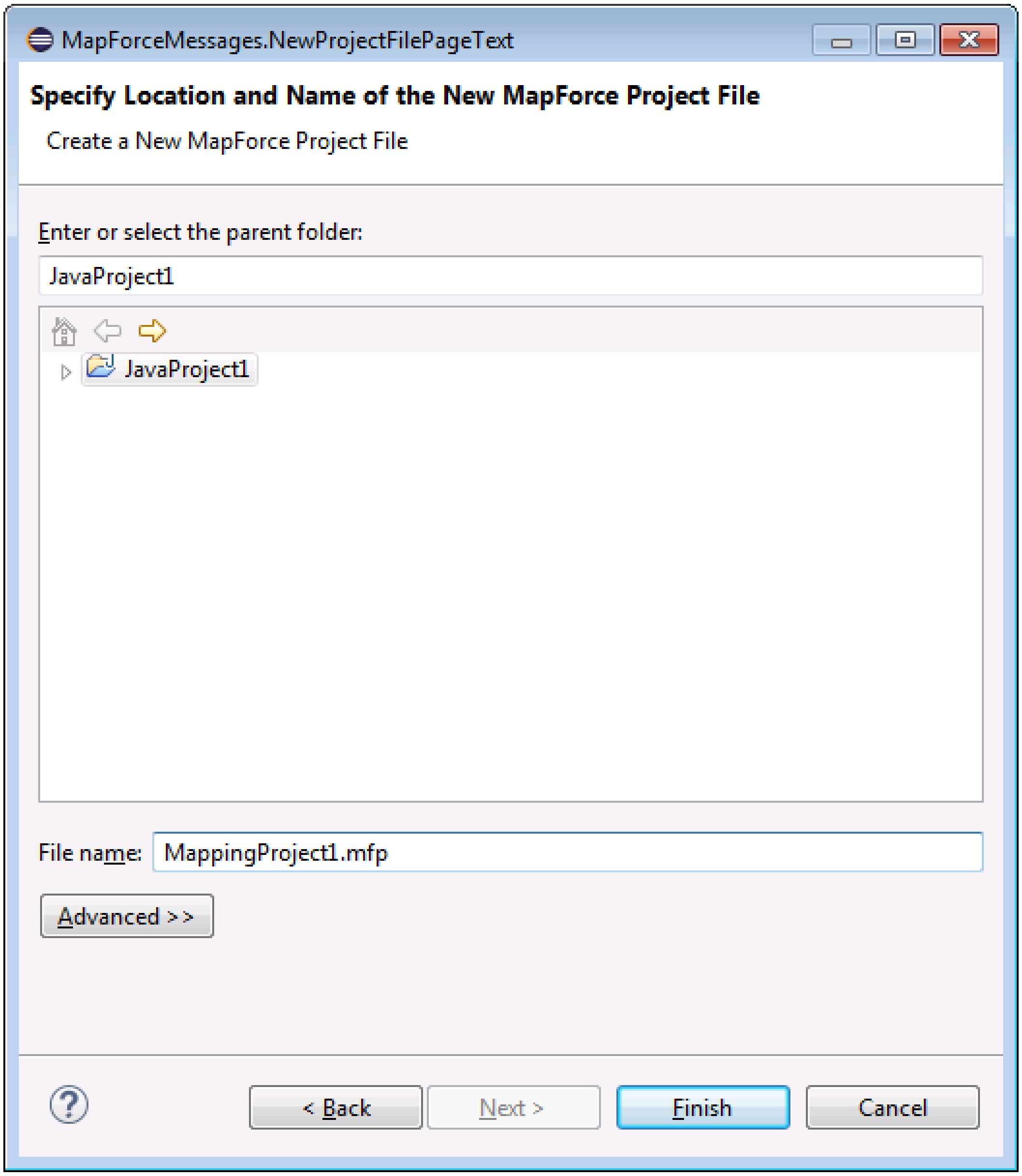
- MapForce mappings
- MapForce project files
- MapForce Web Service projects (available in MapForce Enterprise Edition)

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

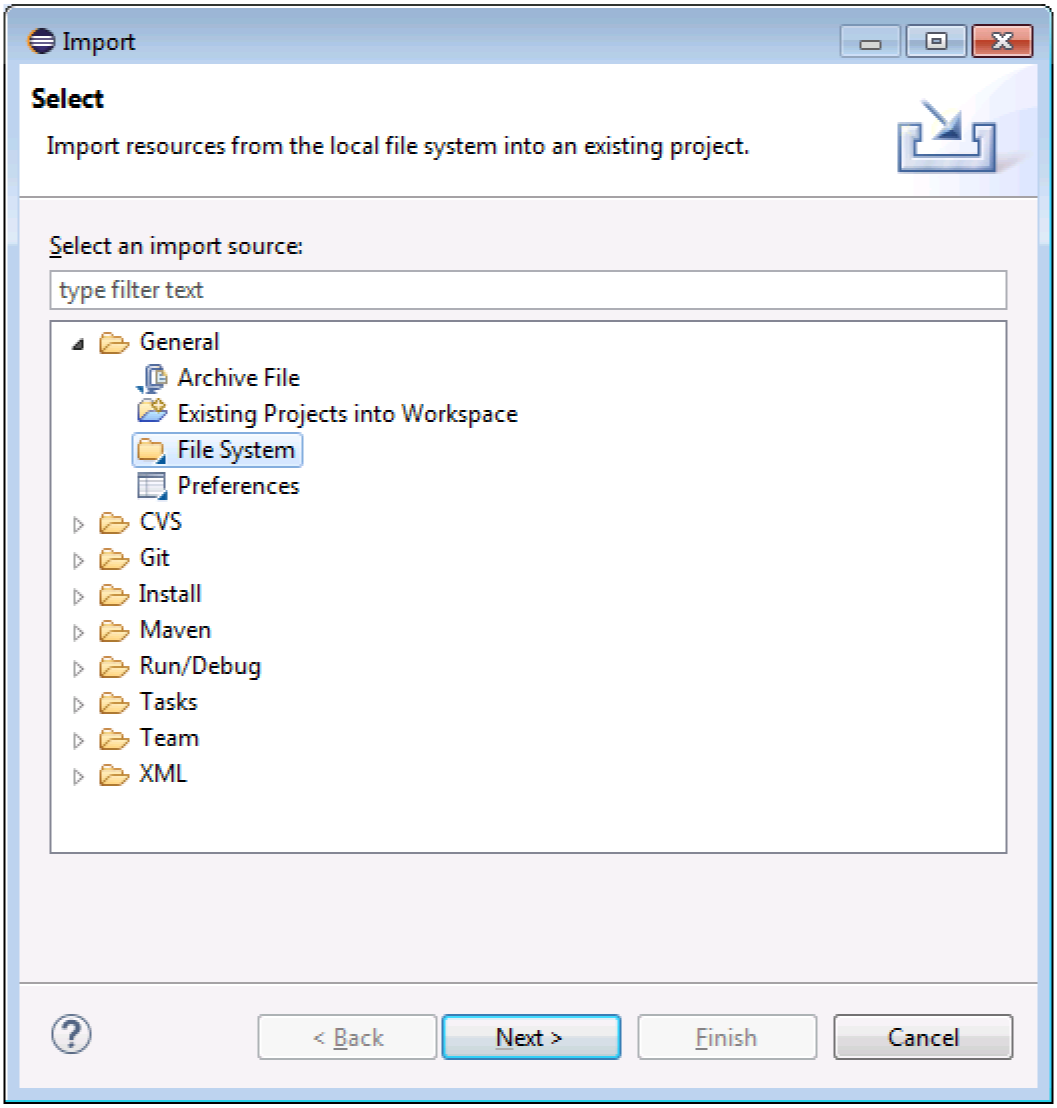
File name: MappingProject1.mfp

4. Select a parent folder in your existing project, and then click **Finish**.

13.4.3 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

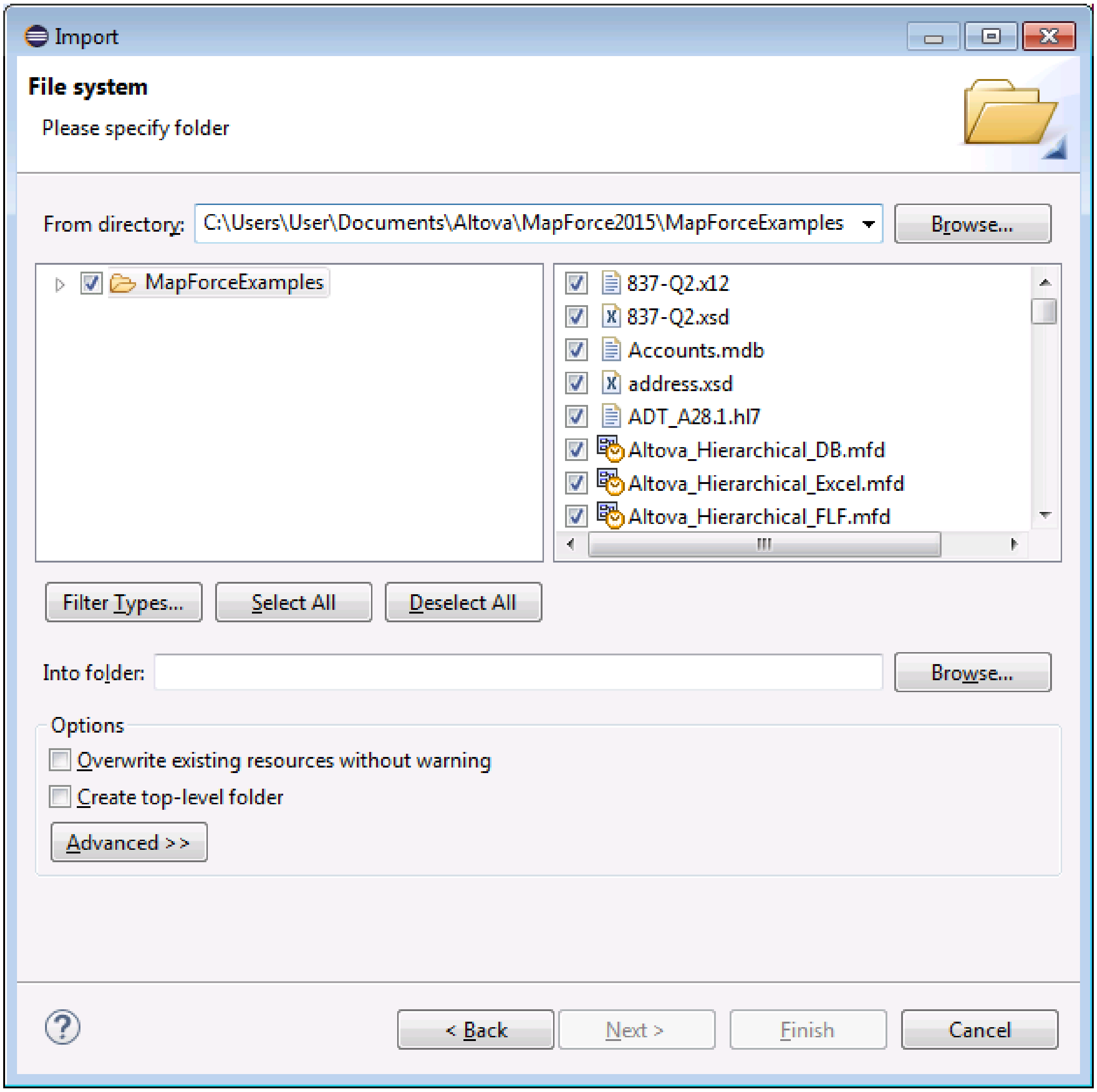
Import resources from the local file system into an existing project.

Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- CVS
- Git
- Install
- Maven
- Run/Debug
- Tasks
- Team
- XML

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

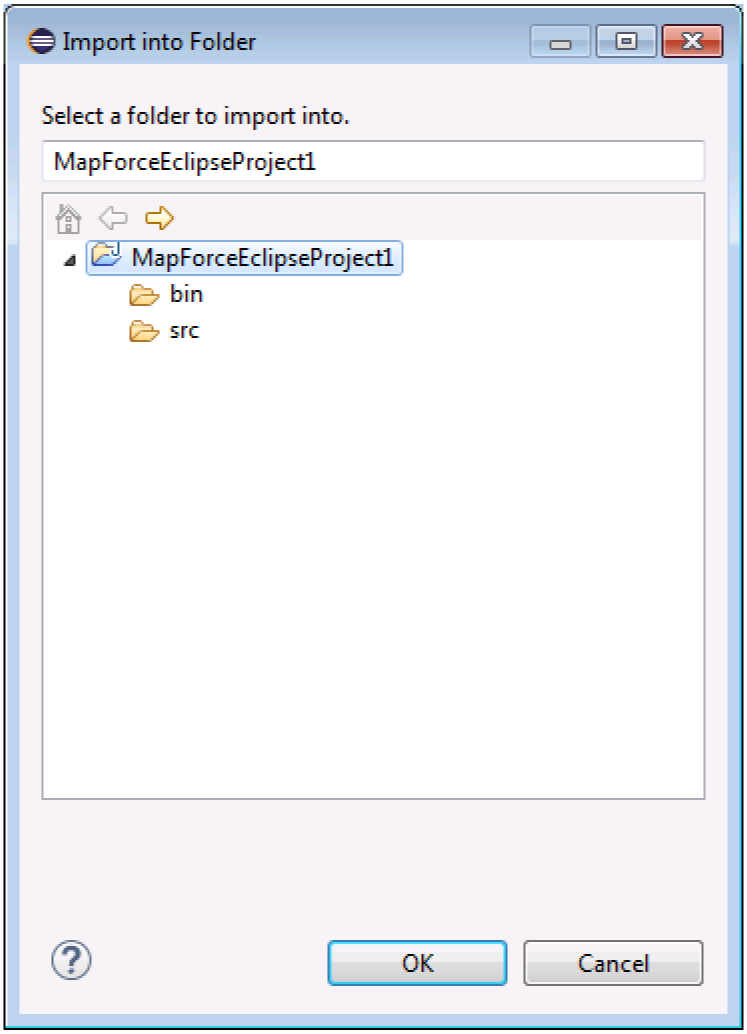
Into folder:

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**.

13.4.4 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 857). 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>
```

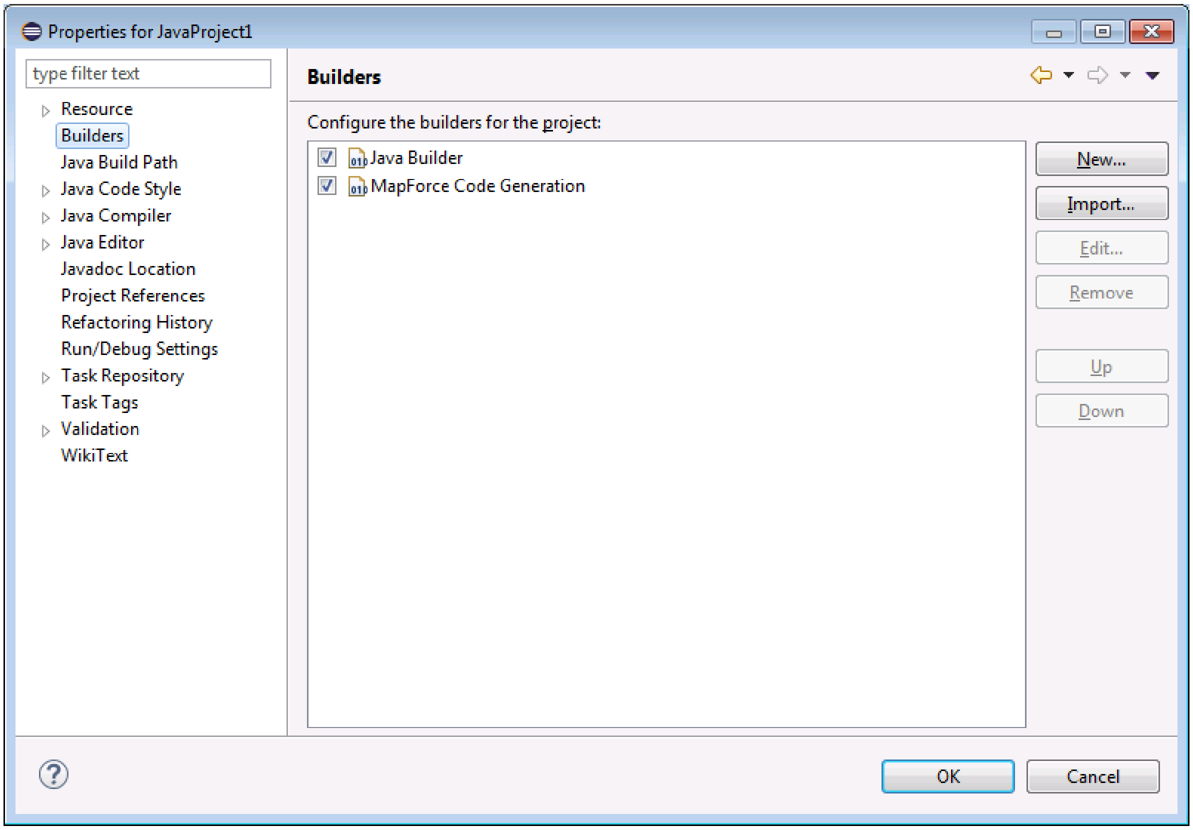
Tip: You can quickly open the **.project** file from the Navigator view of Eclipse (To enable this view, select the menu command **Window | Show View | Navigator**).

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**.



Configure the builders for the project:

☑ Java Builder

☑ MapForce Code Generation

3. Click to clear the **MapForce Code Generation** check box.

13.5 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 (**C:\Program Files\Altova\MapForce2025\eclipse\docs\**).

Before you install and run the sample MapForce plug-in, ensure that the following prerequisites are met:

- You are using 64-bit Java, 64-bit Eclipse, 64-bit MapForce and 64-bit MapForce Integration Package.
- The JDT (Java Development Tools) plug-in is installed.
- The Eclipse PDE (plug-in development environment) is installed.

To import the sample MapForce plug-in project into your workspace:

1. Start Eclipse.
2. On the **File** menu, click **Import**.
3. Select **General | Existing projects into Workspace**, and click **Next**.
4. Click the Browse... button next to the "'Select root directory" field and choose the sample project directory e.g. **C:\Program Files\Altova\MapForce2025\eclipse\workspace\MapForceExtension**.
5. Select the **Copy projects into workspace** option, and then click **Finish**. A new project named "MapForceExtension" has been created in your workspace.

To run the sample extension plug-in:

1. Switch to the Java perspective.
2. In the **Run** menu, click **Run Configurations**.
3. Right click **Eclipse Application** and select **New**. (If you cannot see "Eclipse application" in the list, the Eclipse Plug-In Development Tools are not installed in your Eclipse environment. To install Eclipse Plug-in Development Tools, click **Install New Software** in the **Help** menu. and install "Eclipse Plugin Development Tools" from "The Eclipse Project Updates" download site.)
4. Enter a name for your new configuration (in this example, SampleMapForcePlugin), and then click **Apply**.
5. Check that the MapForceClient workspace plug-in is selected in the 'Plug-ins' tab.
6. Click **Run**. A new Eclipse Workbench opens.
7. Open any MapForce mapping in the new Workbench. It will now open with a zoom level of 70%.

14 Code Generator

Code Generator is a MapForce built-in feature which enables you to generate code from mapping files. The result is a fully-featured and complete application which performs the mapping operation for you. After you have generated 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 and extend the code with your own functionality.

Support information
The table below summarizes support information about C++, C#, and Java.

Target Language	C++	C#	Java
Development environments	Microsoft Visual Studio 2013, 2015, 2017, 2019, 2022	Microsoft Visual Studio 2013, 2015, 2017, 2019, 2022 Target frameworks: • .NET Framework • .NET Core 3.1 • .NET 5.0 • .NET 6.0 • .NET 8.0	Java SE JDK 8, 11, 17, 21 (including OpenJDK) Eclipse 4.4 or later Apache Ant (build.xml file)
XML DOM implementations	MSXML 6.0 Apache Xerces 3	System.Xml	JAXP
Database API	ADO	ADO.NET	JDBC

Note: The MapForce-generated code can be considered thread-safe only if the underlying third-party XML DOM and database API libraries are. Although the thread safety of the generated code cannot be realistically proven or guaranteed, it is likely that multiple concurrent instances of the mapping code will run successfully in most cases.

C++
You can configure whether the C++ generated output should use MSXML 6.0 or Apache Xerces 3. MapForce generates complete project (**`.vcproj`**) and solution (**`.sln`**) files for all supported versions of Visual Studio (*see table above*). The generated code optionally supports MFC.

Note the following prerequisites:

- To compile the generated C++ code, Windows SDK must be installed on your computer.
- To use Xerces 3 for C++, you will need to install and build it using the instructions on the Apache Xerces page. Make sure to add the XERCES3 environment variable that points to the directory where Xerces is installed (e.g., `C:\xerces-c-3.2.2`). Also, the PATH environment variable must include the path where the Xerces binaries are (e.g., `%XERCES3%\bin`).
- When you build C++ code for Visual Studio and use a Xerces library precompiled for Visual C++, you will need to change the compiler setting in all the projects of the solution. Follow the steps below:

  a) Select all projects in the Solution Explorer.
  b) Click **Properties** in the **Project** menu.

c) Click **Configuration Properties | C/C++ | Language**.
d) In the list of configurations, select *All Configurations*.
e) Change *Treat wchar_t as Built-in Type* to *No (/Zc:wchar_t-)*.

C#
The generated C# code can be used from any .NET capable programming language, such as VB.NET, Managed C++, or J#. Project files can be generated for all supported versions of Visual Studio (*see table above*).

Java
The generated Java output is written against the Java API for XML Processing (JAXP) and includes an Ant build file and project files for supported versions of Java and Eclipse (*see table above*).

Generate, build, run, integrate code

For instructions on how to generate, build, and run code, see Generate, Build, and Run Code 870. For details about integrating MapForce-generated code into your custom code, see Integrate Generated Code 876.

Code-generation templates

The generated code is built via a template that is written in a template language called SPL 973 (Spy Programming Language). You can customize the template used for code-generation. This can be useful, for example, when you want to customize your code in accordance with your company's writing conventions or replace specific libraries in the generated code.

Examples

For examples illustrating code generation capabilities, see Example: Book Library 898 and Example: Purchase Order 922.

14.1 Generate, Build, and Run Code

This topic explains how to generate code from a mapping and a project, build the generated code, and run it. There are situations in which you might need to modify your generated C#/C++/Java code to integrate it into your custom code. For details, see Integrate Generated Code[876].

Generate code from a mapping

To generate code from a mapping design (**.mfd**), follow the instructions below:

1. Select the relevant code-generation options in the *Generation* section of the **Options** dialog (applicable to C# and C++) and in the Mapping Settings[75]. For details about the code-generation settings in the **Options** dialog, see Generation[1080].
2. Click **File | Generate code in** and select the relevant transformation language. Alternatively, you can select **File | Generate Code in Selected Language**. In this case, code will be generated in the language selected in the toolbar.
3. Select a destination directory for the generated files and then click **OK** to confirm. MapForce generates the code and displays the result of the operation in the Messages window[26].

Generate code from a project (Professional and Enterprise editions)

You can generate code from a mapping project (**.mfp**) that consists of multiple mapping design files (**.mfd**). Note that all mapping design files in the project must qualify for generation, which means that all their components must be supported in the selected transformation language (see Supported features in generated code[1098]).

To generate code from a mapping project, follow the instructions below.

1. Open the relevant mapping project, for which you wish to generate code.
2. Right-click the project name in the Project window and then select **Properties** from the context menu. Alternatively, click the project name and select the **Project | Properties** menu item.
3. Review and change the project settings if required. In particular, ensure that the target language and the output directory are set correctly. Then click **OK**.
4. Click **Generate Code for Entire Project** in the **Project** menu.

Irrespective of the language selected in the **Project Properties** dialog, you can always choose to generate project code in a different language, by selecting the menu command **Project | Generate Code in | <language>**.

The progress and result of the code generation process 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.

You can change the output directory and/or the name of the project in the **Project Properties** dialog. If your MapForce project contains folders, you can configure the code generation settings for each individual folder: Right-click a folder of interest and select **Properties** from the context menu. Otherwise, all project folders inherit the settings defined at top level. For more information about projects and project-related settings and procedures, see Projects[95].

Language-specific information

This subsection describes the peculiarities of generating code in different transformation languages. This subsection goes on to explain how to build the generated C++, C#, and Java code and run the application. You can also generate code in XSLT 1-3 and XQuery. For details, see Code Generation[68].

C++ and C# code

Generating, building, and running C++ and C# code follow the same logic. The broad procedures are outlined in the subsections below.

After you have generated C++ or C# code, the solution will include the following components:

- Solution (**.sln**) and project (**.vcxproj** for C++ and **.csproj** for C#) files that can be opened in Visual Studio
- Several Altova-signed libraries required by the mapping (all prefixed with `Altova`)
- The main mapping project (called `Mapping` by default), which includes the mapping application and its dependent files

Note that you can change the default name of the main mapping project in the Mapping Settings[75] dialog box.

After you have generated the C++/C# code, the next steps would be to build the code and run the application. There are two major approaches to building the generated code: (i) in Visual Studio and (ii) at the command line (*see details below*). Note that, to build C# code, you must have the relevant SDK and a compatible Visual Studio version installed. For the download package for your operating system and platform, refer to the Microsoft website.

Build generated code in Visual Studio

To build the generated C++/C# code, follow the instructions below:

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 with the generated code.
2. Select the required build configuration (e.g., Debug). Note about C++ code: 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.
3. Click **Build Solution** in the **Build** menu.

As a result of building the code, a command-line application called **Mapping.exe** and its related files will be created. The mapping application will be located in one of the subdirectories relative to the **.sln** file. The name of the subdirectory depends on the selected build configuration: e.g., **Debug** (C++), **bin\Debug** (C#).

Build generated code at command line

To build the generated code at the command line, switch to the directory with the generated code and run the following command:

```
devenv Mapping.sln /Build "Debug|AnyCPU" /Project Mapping
```

This command calls Visual Studio and specifies the name of the solution file to build (**Mapping.sln** in our case), the desired configuration (Debug and any CPU in our case), and the name of the project the solution file belongs to (`Mapping`). As a result of building the code, a command-line application called **`Mapping.exe`** and its related files will be created. The mapping application will be located in one of the subdirectories relative to the **`.sln`** file. The name of the subdirectory depends on the selected build configuration: e.g., **`Debug`** (C++), **`bin\Debug`** (C#).

Notes about building C# code

Besides calling Visual Studio, you can also call .NET to build the generated C# code. Follow the instructions below:

1. Make sure to select the correct platform to target in the *Generation* section of the **Options** dialog (**Tools** menu). For details, see Generation 1080.
2. If you target .NET/.NET Core, configure the PATH and DOTNET_ROOT environment variables so that they point to the location where .NET/.NET Core is installed. This will prevent possible problems with CLI commands.
3. Open a command prompt and switch to the directory with the generated code.
4. Run the following command:

```
dotnet build Mapping\Mapping.sln --configuration Release
```

This command calls .NET to build the solution file called `Mapping.sln` with the `Release` configuration and creates an executable called **`Mapping.exe`** and its related files at the destination folder. The name of the destination folder depends on the selected configuration and the .NET version used. In our example, the executable will be saved in the **`bin\Release\net8.0`** folder.

Run application

After you have built the code in Visual Studio directly or at the command line, you can proceed to run the application. To run the application, double-click **`Mapping.exe`** or call the executable from the command line. After you have run the executable, the result of the mapping transformation will be output to the destination folder (by default, this is the folder where the executable is stored).

If you build the generated code on Linux, the generated executable will be called `Mapping`, without any extension. To run the executable, you may need to use the following command:

```
./Mapping
```

Java code

After you have generated Java code, the Java project will include the following components:

- Several Altova-signed Java packages required by the mapping (all prefixed with `com.altova`)
- The `com.mapforce` package including the mapping application and its dependent files, among which the two most important files that specify the entry points of the application are the following:
  - The Java mapping application as a dialog application (`MappingApplication.java`)
  - The Java mapping application as a console application (`MappingConsole.java`)
- The `build.xml` file which you can execute with Apache Ant to compile the project and generate JAR files

The default names of the mapping application and its dependent files in the `com.mapforce` package are prefixed with **`Mapping`**. You can change this and other settings in the Mapping Settings 75 dialog box.

After you have generated Java code, the next steps would be to build and run the code. There are two major approaches to building the generated code and running the application: (i) in Eclipse and (ii) at the command line, using Apache Ant. The broad procedures are described in the subsections below.

Build generated code and run application in Eclipse

This approach makes use of the Eclipse workflow. Note the following prerequisites:

- Java Development Kit (JDK), Eclipse, and Apache Ant must be installed on your system. Eclipse typically includes a bundled version of Ant, but you can also install Ant separately.
- To run Eclipse with OpenJDK, you need to set the PATH environment variable so that it includes the path to the JDK **`bin`** directory (e.g., **`C:\Java\jdk-11.0.1\bin`**).
- The JAVA_HOME environment variable must point to the location of JDK.
- The ANT_HOME environment variable must point to the location of Apache Ant.

After you have generated Java code, the next step is to import the generated Java code into Eclipse. Follow the steps below:

1. In the **File** menu, click **Import** and select **General | Existing Projects into Workspace**.
2. Click **Next**.
3. Provide the path to the generated code and click **Finish**. The Java project created by MapForce is now available in the Package Explorer view. If you cannot see the Package Explorer view, use the menu command **Window | Show View | Package Explorer**.

Note that, by default, code is built automatically each time a change is detected. You can also disable this functionality and build code when necessary (see the Eclipse documentation for details).

After you have imported and built your code, the next step is to run the application. This topic discusses some of the possible approaches to running an application.

Approach 1: Run project as an application

This method allows you to run your Java project as a GUI application. Take the steps below:

1. In the Package Explorer view of Eclipse, right-click the **`MappingApplication.java`** file in the **`com.mapforce`** package.
2. Select **Run As | Java application** from the context menu.
3. In the MapForce application window that pops up, click **Start** to execute the mapping.

Approach 2: Run project as a console application

This method enables you run your Java project as a console (command-line) application. Take the steps below:

1. In the Package Explorer view of Eclipse, right-click the **`MappingConsole.java`** file in the **`com.mapforce`** package.
2. Select **Run As | Java application** from the context menu.

As a result of running the application, irrespective of the approach selected, the Java application will execute the mapping transformation and generate output file(s) at the destination folder.

Build generated code and run application at command line

To be able to build and run your generated code at the command line, you must have the following components installed and environment variables set:

- Java Development Kit (JDK) and Apache Ant must be installed on your system.
- The location of the Ant **`bin`** directory (e.g., **`C:\apache-ant-1.10.5\bin`**) should be added to the PATH environment variable. This will enable you to conveniently run Ant without having to type the full path to the executable at the command line.
- The JAVA_HOME environment variable must point to the location of JDK.
- The ANT_HOME environment variable must point to the location of Apache Ant.

To build the generated code with Apache Ant, follow the instructions below:

1. Open a command prompt and switch to the directory where the generated code, including the build file (**`build.xml`**), is stored.
2. Run the following command:

   ```
   ant jar
   ```

   This command will build the generated code and create a JAR file (called **`Mapping.jar`** by default). The purpose of the JAR file is to package Java **`.class`** files and their related metadata and resources. In our case, the JAR archive is intended to be used as an executable Java program; therefore, the archive's manifest file includes the entry point of the application (by default, `com.mapforce.MappingConsole`).

To run the Java application, run the following command in the directory where the JAR archive is located:

```
java com.mapforce.MappingConsole Mapping.jar
```

This command starts the Java Virtual Machine, launches the main class called `com.mapforce.MappingConsole`, which refers to the entry point of the Java application, and executes the program contained in the JAR file called **`Mapping.jar`**. As a result, the Java application will execute the mapping transformation and generate output file(s) at the destination folder. If you want to launch the application as a GUI application, pass the following value for the main class argument: `com.mapforce.MappingApplication`. This will open a pop-up window, in which you will be able to start the mapping transformation.

Preventing possible out-of-memory issues

Complex mappings with large schemas can produce a large amount of code, which might cause a `java.lang.OutofMemory` exception during compilation in Ant. To prevent possible out-of-memory issues, take the steps below:

1. Add the `ANT_OPTS` environment variable, which sets specific Ant options such as the memory to be allocated to the compiler, and set its value as follows: `-server -Xmx512m -Xms512m`.
2. 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 you run the `ant jar` command, you may get an error message similar to "`[...] archive contains more than 65535 entities`". To prevent this, it is recommended to use Ant 1.9 or later and to add `zip64mode="as-needed"` to the `<jar>` element in **`build.xml`**.

Preventing possible issues with JDBC connections

If you have generated Java code from a mapping that connects to a database through JDBC, you may need to add the JDBC driver as a classpath entry to the current configuration. Otherwise, running the application could result in an error. For details, see Databases 167.

14.2 Integrate Generated Code

Even though the result of code generation is a complete and fully-functioning application, you may need to adapt MapForce-generated code to be able to integrate it into your custom code. Some typical code-modification scenarios include the following:

- Modifying source and target files for the mapping application 876
- Defining custom error-handling code 876
- Changing the data type of the mapping input in C#- and Java-generated code (for example, from string to stream) 879
- Generating schema wrapper libraries that you can integrate in your custom application in order to read, modify, or write XML documents programmatically 885

14.2.1 Modify Input/Output, Define Error Handling

This topic explains how to modify source and target files and define error handling for the mapping application in Java, C#, and C++ code. To illustrate these procedures, we use the **`MapForceExamples\CompletePO.mfd`** sample mapping. The mapping consists of three source components (**`ShortPO.xml`**, **`Customers.xml`**, and **`Articles.xml`**) and one target component (**`CompletePO.xml`**).

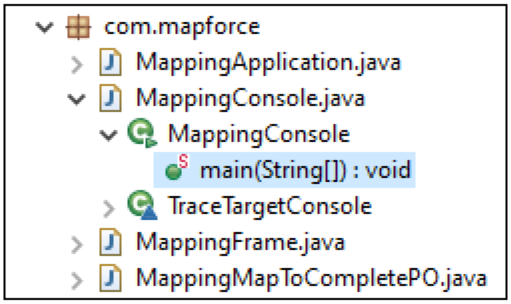
In the generated code, these sources and targets will translate to three input and one output parameters supplied to the **`Run`** method which executes the mapping. Note the following basic points about code generation:

- The number of sources 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.

Java

This example uses Eclipse as a Java IDE. To begin, generate Java code from the **`MapForceExamples\CompletePO.mfd`** sample mapping and then import the project into Eclipse. For information about generating Java code and importing it into Eclipse, see Generate, Build, and Run Code 872.

To edit the generated Java console application, locate the **`main`** method of your generated application in the Project Explorer of Eclipse (*screenshot below*). By default, this method is located 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. By default, the **run** method is invoked from the class called `MappingFrame.java` of the `com.mapforce` package (*screenshot above*).

Modify sources and targets

The code listing below illustrates an extract from the **main** method in the generated Java console application. The paths to the sources and targets are shown below and are defined as parameters to the **run** method. If you need to change the sources and/or targets, change the values of the parameters shown below.

```
com.altova.io.Input Customers2Source =
com.altova.io.StreamInput.createInput("C:/Users/<UserName>/Documents/Altova/MapForce2024/
MapForceExamples/Customers.xml");
                     com.altova.io.Input Articles2Source =
com.altova.io.StreamInput.createInput("C:/Users/<UserName>/Documents/Altova/MapForce2024/
MapForceExamples/Articles.xml");
                     com.altova.io.Input ShortPO2Source =
com.altova.io.StreamInput.createInput("C:/Users/<UserName>/Documents/Altova/MapForce2024/
MapForceExamples/ShortPO.xml");
                     com.altova.io.Output CompletePO2Target = new
com.altova.io.FileOutput("CompletePO.xml");
```

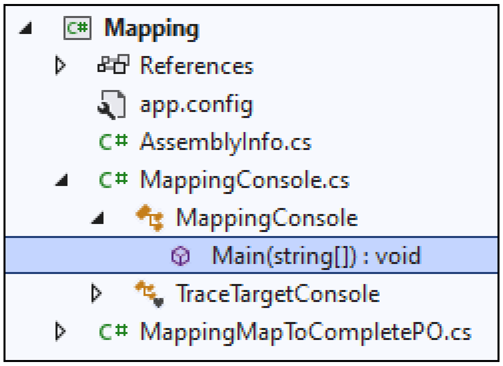
Define custom error handling

If you need to add your custom error-handling code, modify the `catch` statement in the **main** method (console application) or in `MappingFrame.java` (GUI application).

C#

This example uses Microsoft Visual Studio as a C# IDE. To begin, generate C# code from the **MapForceExamples\CompletePO.mfd** sample mapping and then open the solution in Visual Studio. For information about generating code and importing it into Visual Studio, see Generate, Build, and Run Code 871.

To edit the generated C# application, navigate to the **Main** method of your generated application in the Solution Explorer of Visual Studio (*screenshot below*). By default, the solution file is called **Mapping.sln** and is located in the **Mapping** subdirectory relative to the directory where you saved the generated code.



Modify sources and targets

The code listing below illustrates an extract from the **Main** method in the generated C# application. The paths to the sources and targets are shown below and are defined as parameters to the **Run** method. If you need to change the sources and/or targets, change the values of the parameters shown below.

```
Altova.IO.Input Customers2Source =
Altova.IO.StreamInput.createInput("C:/Users/<UserName>/Documents/Altova/MapForce2024/MapF
orceExamples/Customers.xml");
                           Altova.IO.Input Articles2Source =
Altova.IO.StreamInput.createInput("C:/Users/<UserName>/Documents/Altova/MapForce2024/MapF
orceExamples/Articles.xml");
                           Altova.IO.Input ShortPO2Source =
Altova.IO.StreamInput.createInput("C:/Users/<UserName>/Documents/Altova/MapForce2024/MapF
orceExamples/ShortPO.xml");
                           Altova.IO.Output CompletePO2Target = new
Altova.IO.FileOutput("CompletePO.xml");
```

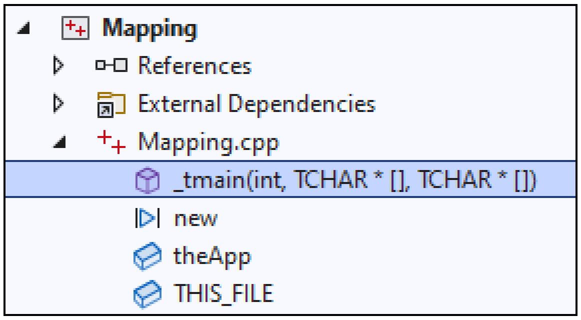
Define custom error handling

If you need to add your custom error-handling code, modify the `catch` statement in the **Main** method.

C++

This example uses Microsoft Visual Studio as a C++ IDE. To begin, generate C++ code from the **MapForceExamples\CompletePO.mfd** sample mapping and then open the solution in Visual Studio. For information about generating code and importing it into Visual Studio, see Generate, Build, and Run Code 871.

To edit the generated C++ application, navigate to the **_tmain** method of your generated application in the Solution Explorer of Visual Studio (*screenshot below*). By default, the solution file is called **Mapping.sln** and is located in the **Mapping** subdirectory relative to the directory where you saved the generated code.



Modify sources and targets
The code listing below illustrates an extract from the `_tmain` method in the generated C++ application. The paths to the sources and targets are shown below and are defined as parameters to the `Run` method. If you need to change the sources and/or targets, change the values of the parameters shown below.

```
MappingMapToCompletePO MappingMapToCompletePOObject;
                    MappingMapToCompletePOObject.Run(

_T("C:/Users/<UserName>/Documents/Altova/MapForce2024/MapForceExamples/Customers.xml"),

_T("C:/Users/<UserName>/Documents/Altova/MapForce2024/MapForceExamples/Articles.xml"),

_T("C:/Users/<UserName>/Documents/Altova/MapForce2024/MapForceExamples/ShortPO.xml"),
              _T("CompletePO.xml"));
```

Define custom error handling
If you need to add your custom error-handling code, modify the `catch` statement in the `_tmain` method.

14.2.2 Change Data Type of Input/Output

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. After generating Java or C# code 870 with MapForce, you can optionally change the data type of the mapping input or output by editing the generated code. More specifically, you can use as mapping parameters objects of types other than those generated by default. For example, instead of having the mapping read the input from a file on the disk, you can provide a string or a stream object as input. Note that this feature is specific to code generated in C# or Java only.

The object types supported as input or output are listed in the first column of the table below. Each subsequent column specifies data formats where that specific type is supported. For a more precise definition of each type, see the "Type definitions" section below.

	XML	JSON*	Microsoft Excel*	EDI (includes X12, HL7)*	FlexText*	CSV/Text
Files	Yes	Yes	Yes	Yes	Yes	Yes
Streams	Yes	Yes	Yes	Yes	Yes	Yes
Strings	Yes	Yes	–	Yes	Yes	Yes
Reader/Writer	Yes	Yes	–	Yes	Yes	Yes
DOM documents	Yes	–	–	–	–	–

** Formats supported only in MapForce Enterprise Edition*

To change the data type of the mapping input or output:

1. Generate C# or Java code from a mapping.
2. In the generated code, find the call to the **run** method (in Java) or **Run** method (in C#), as follows:

   a. If using C#, open the **MappingConsole.cs** file.
   b. If using Java, open the **MappingConsole.java** (the console program) or the **MappingFrame.java** file (the GUI program).

Note: The name of the file may be different if you have changed the application name in the mapping settings 75. For example, if you changed it to "MyApp", then name of the generated file becomes **MyAppConsole.js** and **MyAppConsole.java**, and **MyAppFrame.java**, respectively.

3. Create an instance of the required type (see the "Type definitions" section).
4. Supply the declared objects as parameters to the **run** method (in Java) or **Run** method (in C#), as shown in the examples below.

The **run** method is the most important method of generated mapping classes. It has one parameter for each *static* source or input component in the mapping, and a final parameter for the output component. If your mapping contains components that process multiple files dynamically 746, the respective parameters do not appear in generated code, because in this case the file names are processed dynamically inside the mapping.

Type definitions

In C#, the types that you can provide as parameters to the **Run** method are classes defined in the `Altova.IO` namespace. The base classes are `Altova.IO.Input` and `Altova.IO.Output`, respectively.

C# types

Files	`Altova.IO.FileInput(string filename)`
	`Altova.IO.FileOutput(string filename)`

Streams	`Altova.IO.StreamInput(System.IO.Stream stream)`
	`Altova.IO.StreamOutput(System.IO.Stream stream)`

Strings	`Altova.IO.StringInput(string content)`
	`Altova.IO.StringOutput(System.Text.StringBuilder sbuilder)`

Reader/Writer	`Altova.IO.ReaderInput(System.IO.TextReader reader)`
	`Altova.IO.WriterOutput(System.IO.TextWriter writer)`

DOM documents	`Altova.IO.DocumentInput(System.Xml.XmlDocument document)`
	`Altova.IO.DocumentOutput(System.Xml.XmlDocument document)`

In Java, the types that you can provide as parameters to the **`run`** method are classes defined in the `com.altova.io` package. The base classes are `com.altova.io.Input` and `com.altova.io.Output`, respectively.

Java types

Files	`com.altova.io.FileInput(String filename)`
	`com.altova.io.FileOutput(String filename)`

Streams	`com.altova.io.StreamInput(java.io.InputStream stream)`
	`com.altova.io.StreamOutput(String filename)`

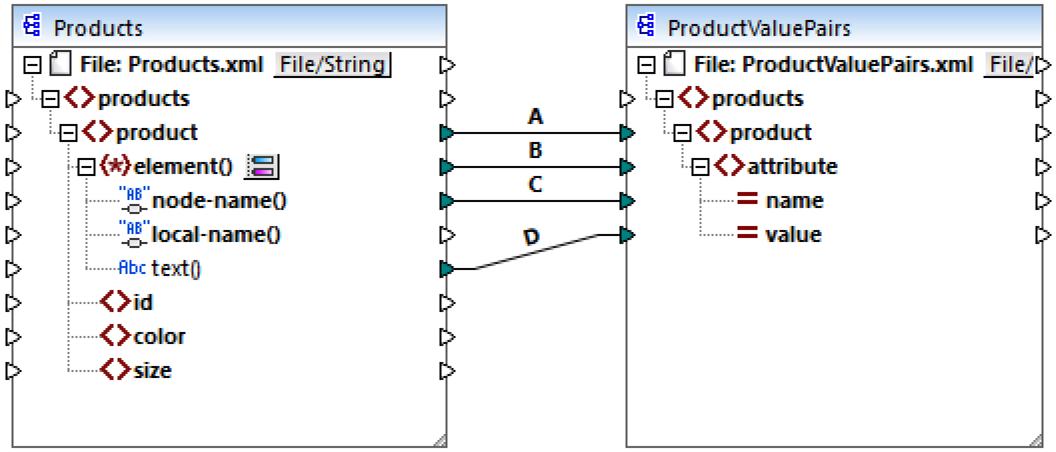
Strings	`com.altova.io.StringInput(String content)`
	`com.altova.io.StringOutput()`

Reader/Writer	`com.altova.io.ReaderInput(java.io.Reader reader)`
	`com.altova.io.WriterOutput(java.io.Writer writer)`

DOM documents	`com.altova.io.DocumentInput(org.w3c.dom.Document document)`
	`com.altova.io.DocumentOutput(org.w3c.dom.Document document)`

Example

To illustrate changing the input and output programmatically, we will use the **ConvertProducts.mfd** mapping as a model. After installing MapForce and running it at least once, you can find this mapping in the following directory: **C:\Users\<username>\Documents\Altova\MapForce2025\MapForceExamples\Tutorials**.

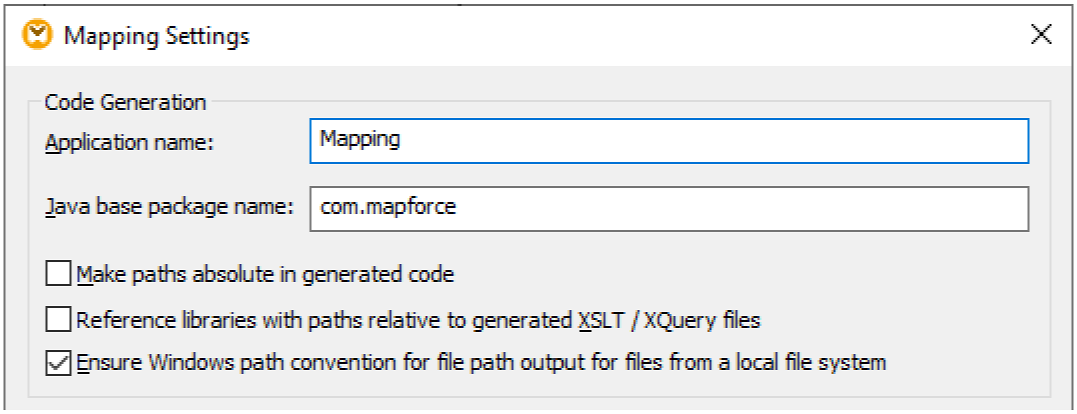


ConvertProducts.mfd

As illustrated above, the mapping converts data from a source XML document to another XML document. Our goals are as follows:

1. Generate Java and C# program code from this mapping.
2. Change the data type of the source component to a string type.
3. Change the data type of the target component to a string writer type.

To generate the program code, open the **ConvertProducts.mfd** mapping and run the **File | Generate code in | C#** (or **Java**) command. For the scope of this example, we will assume that the mapping settings of **ConvertProducts.mfd** are the default ones.



Application name: Mapping

Java base package name: com.mapforce

□ Make paths absolute in generated code

□ Reference libraries with paths relative to generated XSLT / XQuery files

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

This example uses the following target directories for the generated code (feel free to change the path if necessary):

- **C:\codegen\cs\ConvertProducts**, for C#
- **C:\codegen\java\ConvertProducts**, for Java

Having generated the program code, open the **MappingConsole.cs** (in C#) or **MappingConsole.java** (in Java) and find the following lines:

C#

```
Altova.IO.Input Products2Source = Altova.IO.StreamInput.createInput("Products.xml");
Altova.IO.Output ProductValuePairs2Target = new
Altova.IO.FileOutput("ProductValuePairs.xml");
```

Java

```
com.altova.io.Input Products2Source =
com.altova.io.StreamInput.createInput("Products.xml");
com.altova.io.Output ProductValuePairs2Target = new
com.altova.io.FileOutput("ProductValuePairs.xml");
```

Comment out the lines above and change the code as follows:

C#

```
//Altova.IO.Input Products2Source = Altova.IO.StreamInput.createInput("Products.xml");
```

```
//Altova.IO.Output ProductValuePairs2Target = new
Altova.IO.FileOutput("ProductValuePairs.xml");

Altova.IO.Input Products2Source = new Altova.IO.StringInput("<?xml version=\"1.0\"
encoding=\"UTF-8\"?>\r\n" +
                                       "<products
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\"products.xsd\">\r\n" +
                                       "   <product>\r\n" +
                                       "      <id>100</id>\r\n" +
                                       "      <color>blue</color>\r\n" +
                                       "      <size>XXL</size>\r\n" +
                                       "   </product>\r\n" +
                                       "</products>\r\n");

System.IO.StringWriter writer = new System.IO.StringWriter(new
System.Text.StringBuilder());
Altova.IO.Output ProductValuePairs2Target = new Altova.IO.WriterOutput(writer);

try
{
    MappingMapToProductValuePairsObject.Run(Products2Source, ProductValuePairs2Target);

    // Print out the writer object
    Console.Write(writer.ToString());
}
finally
{
    Products2Source.Close();
    ProductValuePairs2Target.Close();
}
```

Java

```
//com.altova.io.Input Products2Source =
com.altova.io.StreamInput.createInput("Products.xml");
//com.altova.io.Output ProductValuePairs2Target = new
com.altova.io.FileOutput("ProductValuePairs.xml");

com.altova.io.Input Products2Source = new com.altova.io.StringInput("<?xml
version=\"1.0\" encoding=\"UTF-8\"?>\r\n" +
               "<products xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\"products.xsd\">\r\n" +
               "   <product>\r\n" +
               "      <id>100</id>\r\n" +
               "      <color>blue</color>\r\n" +
               "      <size>XXL</size>\r\n" +
               "   </product>\r\n" +
               "</products>\r\n");

java.io.StringWriter writer = new java.io.StringWriter();
com.altova.io.Output ProductValuePairs2Target = new com.altova.io.WriterOutput(writer);
```

```
try {
   MappingMapToProductValuePairsObject.run(Products2Source, ProductValuePairs2Target);

   // Print out the writer object
   System.out.print(writer.toString());

   } finally {
      (Products2Source).close();
      ProductValuePairs2Target.close();
}
```

In the C# and Java code listings above, the following happens:

- The two original lines that provide the input and output to the **run** method were commented out. Consequently, the mapping application no longer reads data from **Products.xml**. In fact, we did not even need to copy this file to the program's working directory.
- The type `Products2Source` has been declared as a `StringInput` that provides the content of the input XML file to be processed.
- The type `ProductValuePairs2Target` has been declared as a `WriterOutput` type that takes a string writer as argument.
- After the mapping completes running, the contents of the string writer is printed out to the console.

Usage guidelines for streams and Reader/Writer objects

When using binary streams or Reader/Writer objects as input or output to the mapping, note the following:

- Binary stream objects and Reader/Writer objects are expected to be opened and ready to use before calling the **run** method.
- By default, the **run** method closes the stream when finished. To prevent this behavior, insert (or uncomment) the following line before calling the **run** method:

C#

```
MappingMapToSomething.CloseObjectsAfterRun = false;
```

Java

```
MappingMapToSomething.setCloseObjectsAfterRun(false);
```

Note: Make sure to change `MappingMapToSomething` to the name of the mapping object as applicable to your generated code.

Usage guidelines for strings

In Java, the constructor of `StringOutput` does not take an argument. The string content produced by the mapping can be accessed by calling the `getString()` method, for example:

Java

```
com.altova.io.Input Products2Source =
com.altova.io.StreamInput.createInput("Products.xml");
com.altova.io.StringOutput ProductValuePairs2Target = new com.altova.io.StringOutput();

try {
   // Run the mapping
   MappingMapToProductValuePairsObject.run(Products2Source, ProductValuePairs2Target);
   // Get the string object
   String str = ProductValuePairs2Target.getString().toString();
}
```

In C#, the constructor of `StringOutput` takes a parameter of type `StringBuilder` which you need to declare beforehand. If the `StringBuilder` object already contains data, the mapping output will be appended to it.

C#

```
Altova.IO.Input Products2Source = Altova.IO.StreamInput.createInput("Products.xml");
System.Text.StringBuilder sb = new System.Text.StringBuilder();
Altova.IO.Output ProductValuePairs2Target = new Altova.IO.StringOutput(sb);

try
{
   // Run the mapping
   MappingMapToProductValuePairsObject.Run(Products2Source, ProductValuePairs2Target);
   // Get the string output
   String str = sb.ToString();
}
```

To run these code listings, you can use the same generated project as in the previous example. Make sure, however, to copy the file **Products.xml** from **C:\Users\<username>\Documents\Altova\MapForce2025\MapForceExamples\Tutorials\** to your program's working directory, since the mapping code reads data from this file.

Usage guidelines for DOM documents

When using DOM documents as mapping input or output, note the following:

- The document instance supplied as parameter to the `DocumentOutput` constructor must be empty.
- After calling **`run`**, the DOM Document generated by the constructor of `DocumentOutput` already contains the mapping output, and you can manipulate the document as necessary.

14.2.3 Generate Code from XML Schemas or DTDs

When you generate code from a mapping, MapForce generates a complete application that executes all steps of the mapping automatically. Optionally, you can generate libraries for all the XML schemas used in the mapping. These allow your code to easily create or read XML instances that are used or created by the mapping code.

To generate libraries for all the XML schemas used in the mapping, select the **Generate Wrapper Classes** check box from the code generator options[1080]. Next time when you generate code, MapForce will create not only the mapping application, but also wrapper classes for all schemas used in the mapping, as follows:

C++ or C#	Java	Purpose
Altova	`com.altova`	Base library containing common runtime support, identical for every schema.
AltovaXML	`com.altova.xml`	Base library containing runtime support for XML, identical for every schema.
[YourSchema]	`com.YourSchema`	A library containing declarations generated from the input schema, named as the schema file or DTD. This library is a DOM (W3C Document Object Model) wrapper that allows you to read, modify and create XML documents easily and safely. All data is held inside the DOM, and there are methods for extracting data from the DOM, and to update and create data into the DOM. The generated C++ code supports either Microsoft MSXML or Apache Xerces 3. The syntax for using the generated code is generally similar for both DOM implementations, except for a few slight differences (for example, Xerces supports more overloaded functions). The generated C# code uses the .NET standard System.XML library as the underlying DOM implementation. The generated Java code uses JAXP (Java API for XML Processing) as the underlying DOM interface.

While prototyping an application from a frequently changing XML schema, you may need to frequently generate code to the same directory, so that the schema changes are immediately reflected in the code. Note that the generated test application and the Altova libraries are overwritten every time when you generate code into the same target directory. Therefore, do not add code to the generated test application. Instead, integrate the Altova libraries into your project (see Integrating Schema Wrapper Libraries 895).

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 973 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 Reference 973.

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++) 929
- Reference to Generated Classes (C#) 944
- Reference to Generated Classes (Java) 959

For information about type conversion and other details applicable to each language, see:

- About Schema Wrapper Libraries (C++) 888
- About Schema Wrapper Libraries (C#) 891
- About Schema Wrapper Libraries (Java) 893

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 973, but is not enforced by generated classes.
- Default and fixed values for attributes. These are available in SPL 973, 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`)

14.2.3.1 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`	

XML Schema	C++	Remarks
`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` 929	
`xs:duration`	`altova::Duration` 932	
`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` 938
`altova::meta::ComplexType` 936

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]` 939.

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` 942
`[YourSchema]::MemberElement` 943

Note: The actual class names depend on the name of the schema attribute or element.

See also Example: Using the Schema Wrapper Libraries 898.

Error Handling

Errors are reported by exceptions. The following exception classes are defined in the namespace `altova`:

Class	Base Class	Description
`Error`	`std::logic_error`	Internal program logic error (independent of input data).
`Exception`	`std::runtime_error`	Base class for runtime errors.
`InvalidArgumentsException`	`Exception`	A method was called with invalid argument values.
`ConversionException`	`Exception`	Exception thrown when a type conversion fails.
`StringParseException`	`ConversionException`	A value in the lexical space cannot be converted to value space.
`ValueNotRepresentableException`	`ConversionException`	A value in the value space cannot be converted to lexical space.
`OutOfRangeException`	`ConversionException`	A source value cannot be represented in target domain.
`InvalidOperationException`	`Exception`	An operation was attempted that is not valid in the given context.
`DataSourceUnavailableException`	`Exception`	A problem occurred while loading an XML instance.
`DataTargetUnavailableException`	`Exception`	A problem occurred while saving an XML instance.

All exception classes contain a message text and a pointer to a possible inner exception.

Method	Purpose
`string_type message()`	Returns a textual description of the exception.
`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 936
altova::meta::ComplexType 936
altova::meta::Element 937
altova::meta::SimpleType 938

14.2.3.2 About Schema Wrapper Libraries (C#)

The default mapping of XML Schema types to C# data types is as follows.

XML Schema	C#	Remarks
`xs:string`	`string`	
`xs:boolean`	`bool`	
`xs:decimal`	`decimal`	xs:decimal has unlimited range and precision, mapped to decimal for efficiency reasons.
`xs:float, xs:double`	`double`	
`xs:long`	`long`	
`xs:unsignedLong`	`ulong`	
`xs:int`	`int`	
`xs:unsignedInt`	`uint`	
`xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth`	`Altova.Types.DateTime` 944	
`xs:duration`	`Altova.Types.Duration` 949	
`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 953

Altova.Xml.Meta.ComplexType 952

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] 954.

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 957
[YourSchemaType].MemberElement 957

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 951
Altova.Xml.Meta.ComplexType 952
Altova.Xml.Meta.Element 953
Altova.Xml.Meta.SimpleType 953

The properties that return one of the metadata classes return null if the respective property does not exist.

14.2.3.3 About Schema Wrapper Libraries (Java)

The default mapping of XML Schema types to Java data types is as follows:

XML Schema	Java	Remarks
xs:string	String	
xs:boolean	boolean	
xs:decimal	java.math.BigDecimal	
xs:float, xs:double	double	
xs:integer	java.math.BigInteger	
xs:long	long	
xs:unsignedLong	java.math.BigInteger	Java does not have unsigned types.
xs:int	int	
xs:unsignedInt	long	Java does not have unsigned types.
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	com.altova.types.DateTime 959	
xs:duration	com.altova.types.Duration 963	
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` 968
`com.altova.xml.meta.ComplexType` 967

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]` 969.

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` 972
`com.[YourSchema].[YourSchemaType].MemberElement` 972

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

Class	Description
`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` 967
`com.altova.xml.meta.ComplexType` 967
`com.altova.xml.meta.Element` 968
`com.altova.xml.meta.SimpleType` 968

The properties that return one of the metadata classes return null if the respective property does not exist.

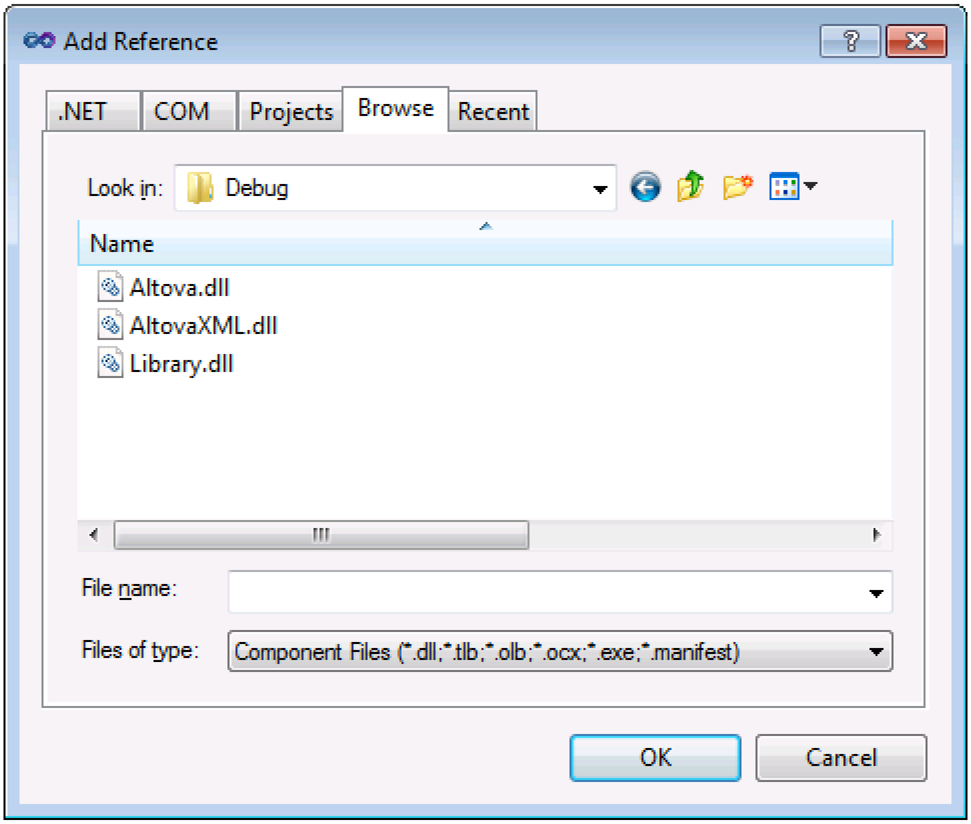
14.2.3.4 Integrate Schema Wrapper Libraries

To use the Altova libraries in your custom project, refer to the libraries from your project or include them in your project, as shown below for each language.

C#

To integrate the Altova libraries into an existing C# project:

1. After MapForce generates code from a schema (for example, **YourSchema.xsd**), build the generated **YourSchema.sln** solution in Visual Studio. This solution is in a project folder with the same name as the schema.
2. Right-click your existing project in Visual Studio, and select **Add Reference**.
3. On the Browse tab, browse for the following libraries: **Altova.dll**, **AltovaXML.dll**, and **YourSchema.dll** located in the output directory of the generated projects (for example, **bin\Debug**).



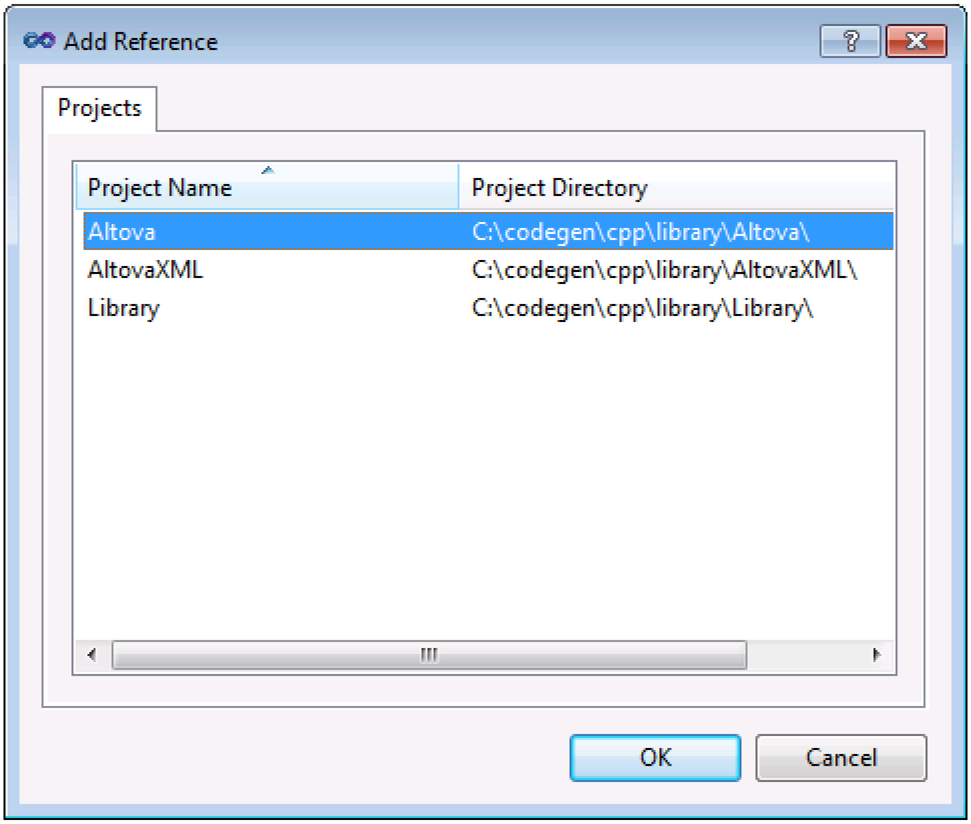
C++

The easiest way to integrate the libraries into an existing C++ project is to add the generated project files to your solution. For example, let's assume that you generated code from a schema called **Library.xsd** and selected **c:\codegen\cpp\library** as target directory. The generated libraries in this case are available at:

- c:\codegen\cpp\library\Altova.vcxproj
- c:\codegen\cpp\library\AltovaXML\AltovaXML.vcxproj
- c:\codegen\cpp\library\Library.vcxproj

First, open the generated **c:\codegen\cpp\library\Library.sln** solution and build it in Visual Studio.

Next, open your existing Visual Studio solution (in Visual Studio 2010, in this example), right-click it, select **Add | Existing Project**, and add the project files listed above, one by one. Be patient while Visual Studio parses the files. Next, right-click your project and select **Properties**. In the Property Pages dialog box, select **Common Properties | Framework and References**, and then click **Add New Reference**. Next, select and add each of the following projects: *Altova*, *AltovaXML*, and *Library*.



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

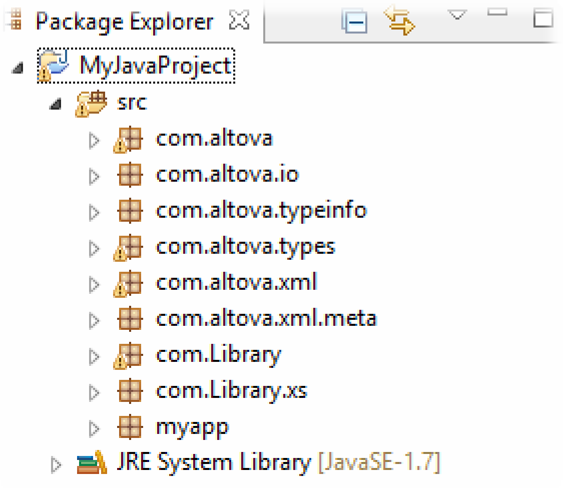
The option to generate static or dynamic libraries is available in the code generation options (see Generation 1080).

Java

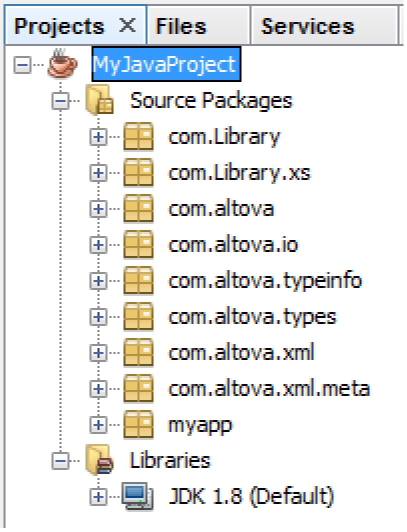
One of the ways to integrate the Altova packages into your Java project is to copy the **com** directory of the generated code to the directory which stores the source packages of your Java project (for example, **C:\Workspace\MyJavaProject\src**). For example, let's assume that you generated code in **c:\codegen\java\library**. The generated Altova classes in this case are available at **c:\codegen\java\library\com**.

After copying the libraries, refresh the project. To refresh the project in Eclipse, select it in the Package Explorer, and press **F5**. To refresh the project in NetBeans IDE 8.0, select the menu command **Source | Scan for External Changes**.

Once you perform the copy operation, the Altova packages are available in the Package Explorer (in case of Eclipse), or under "Source Packages" in the Projects pane (in case of NetBeans IDE).



Altova packages in Eclipse 4.4



Altova packages in NetBeans IDE 8.0.2

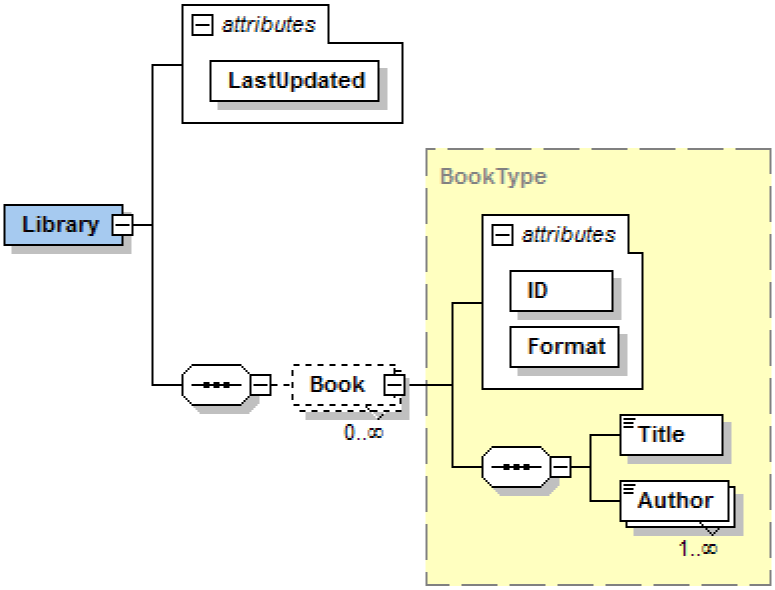
14.2.3.5 Example: Book Library

This example illustrates how to use the generated schema wrapper libraries in order to write or read programmatically XML documents conformant to the schema. Before using the sample code, take some time to understand the structure of the schema below.

The schema used in this example describes a library of books. The complete definition of the schema is shown below. Save this code listing as **`Library.xsd`** if you want to get the same results as this example. You will need this schema to generate the code libraries used in this example.

```xml
<?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>
```

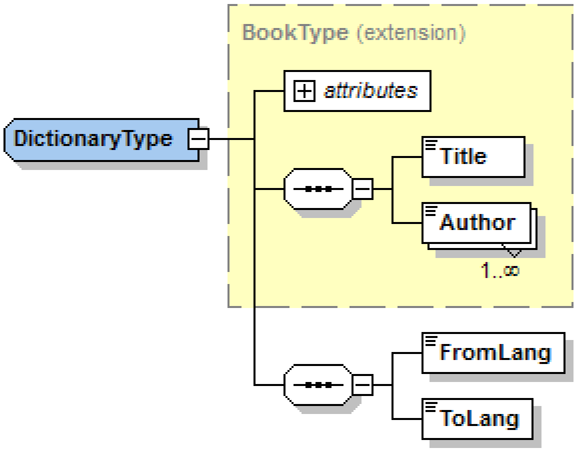
Library is a root element of a `complexType` which can be graphically represented as follows in the schema view of XMLSpy:



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>
```

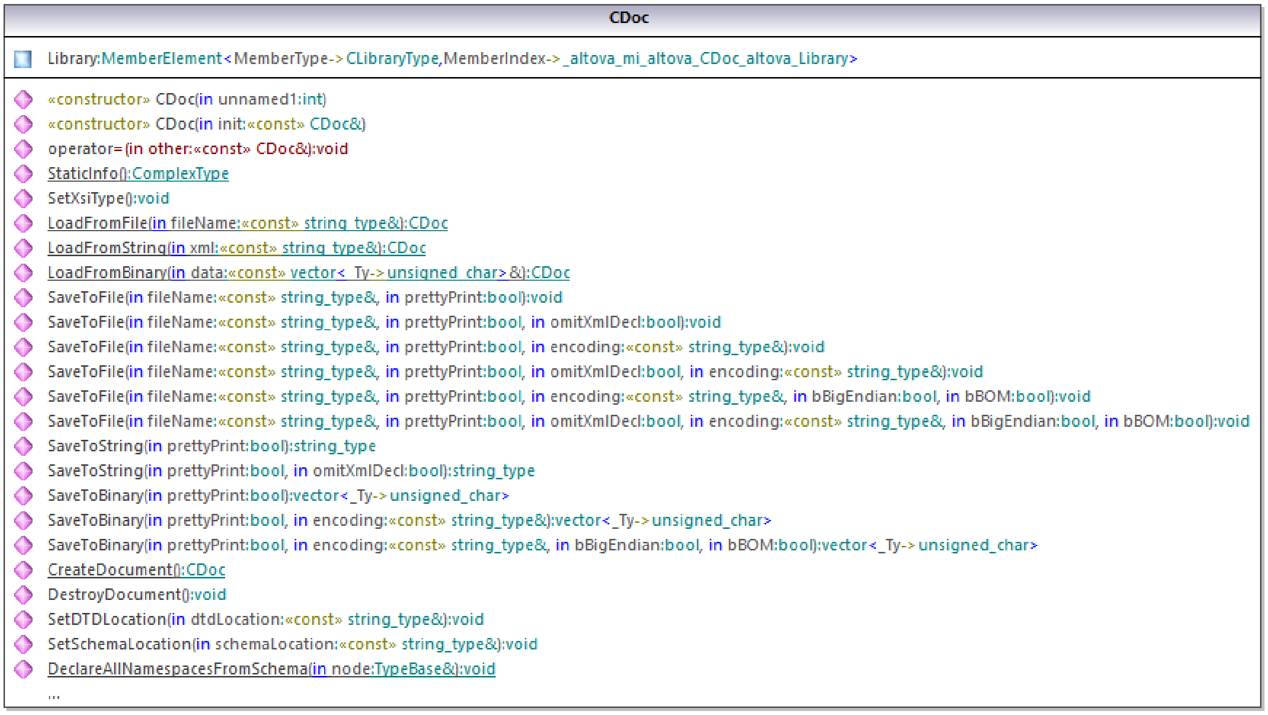
The next topics illustrate how to read from such a file programmatically, or write to such a file programmatically. To begin, generate the schema wrapper code from the schema above, using the steps described in Generating Code from XML Schemas or DTD[885].

14.2.3.5.1 Reading and Writing XML Documents (C++)

After you generate code from the example schema[898], a test C++ application is created, along with several supporting Altova libraries.

About the generated C++ libraries

The central class of the generated code is the `CDoc` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name. As shown in the diagram, this class provides methods for loading documents from files, binary streams, or strings (or saving documents to files, streams, strings). For a description of all members exposed by this class, see the class reference (`[YourSchema]::[CDoc]`[939]).



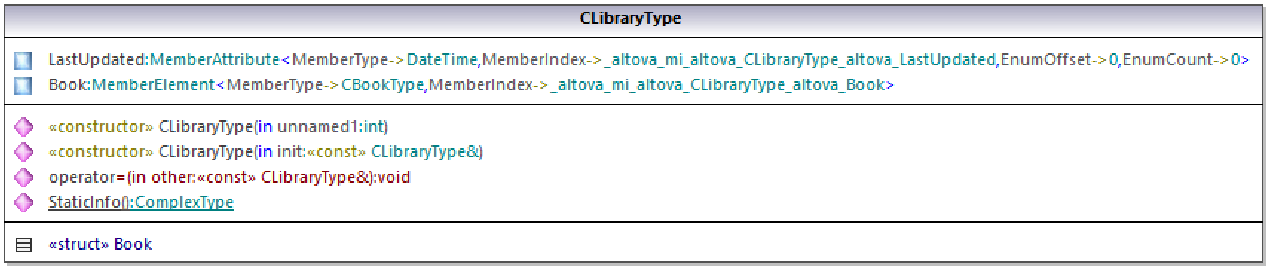
CDoc

Library:MemberElement<MemberType->CLibraryType,MemberIndex->_altova_mi_altova_CDoc_altova_Library>

«constructor» CDoc(in unnamed1:int)
«constructor» CDoc(in init:«const» CDoc&)
operator=(in other:«const» CDoc&):void
StaticInfo():ComplexType
SetXsiType():void
LoadFromFile(in fileName:«const» string_type&):CDoc
LoadFromString(in xml:«const» string_type&):CDoc
LoadFromBinary(in data:«const» vector<_Ty->unsigned_char>&):CDoc
SaveToFile(in fileName:«const» string_type&, in prettyPrint:bool):void
SaveToFile(in fileName:«const» string_type&, in prettyPrint:bool, in omitXmlDecl:bool):void
SaveToFile(in fileName:«const» string_type&, in prettyPrint:bool, in encoding:«const» string_type&):void
SaveToFile(in fileName:«const» string_type&, in prettyPrint:bool, in omitXmlDecl:bool, in encoding:«const» string_type&):void
SaveToFile(in fileName:«const» string_type&, in prettyPrint:bool, in encoding:«const» string_type&, in bBigEndian:bool, in bBOM:bool):void
SaveToFile(in fileName:«const» string_type&, in prettyPrint:bool, in omitXmlDecl:bool, in encoding:«const» string_type&, in bBigEndian:bool, in bBOM:bool):void
SaveToString(in prettyPrint:bool):string_type
SaveToString(in prettyPrint:bool, in omitXmlDecl:bool):string_type
SaveToBinary(in prettyPrint:bool):vector<_Ty->unsigned_char>
SaveToBinary(in prettyPrint:bool, in encoding:«const» string_type&):vector<_Ty->unsigned_char>
SaveToBinary(in prettyPrint:bool, in encoding:«const» string_type&, in bBigEndian:bool, in bBOM:bool):vector<_Ty->unsigned_char>
CreateDocument():CDoc
DestroyDocument():void
SetDTDLocation(in dtdLocation:«const» string_type&):void
SetSchemaLocation(in schemaLocation:«const» string_type&):void
DeclareAllNamespacesFromSchema(in node:TypeBase&):void
...

The `Library` field of the `CDoc` class represents the actual root of the document. **Library** is an element in the XML file, so in the C++ code it has a template class as type (`MemberElement`). The template class exposes methods and properties for interacting with the **Library** element. In general, each attribute and each element of a type in the schema is typed in the generated code with the `MemberAttribute` and `MemberElement` template classes, respectively. For more information, see `[YourSchema]::MemberAttribute` 942 and `[YourSchema]::MemberElement` 943 class reference.

The class `CLibraryType` is generated from the **LibraryType** complex type in the schema. Notice that the `CLibraryType` class contains two fields: `Book` and `LastUpdated`. According to the logic already mentioned above, these correspond to the **Book** element and **LastUpdated** attribute in the schema, and enable you to manipulate programmatically (append, remove, etc) elements and attributes in the instance XML document.



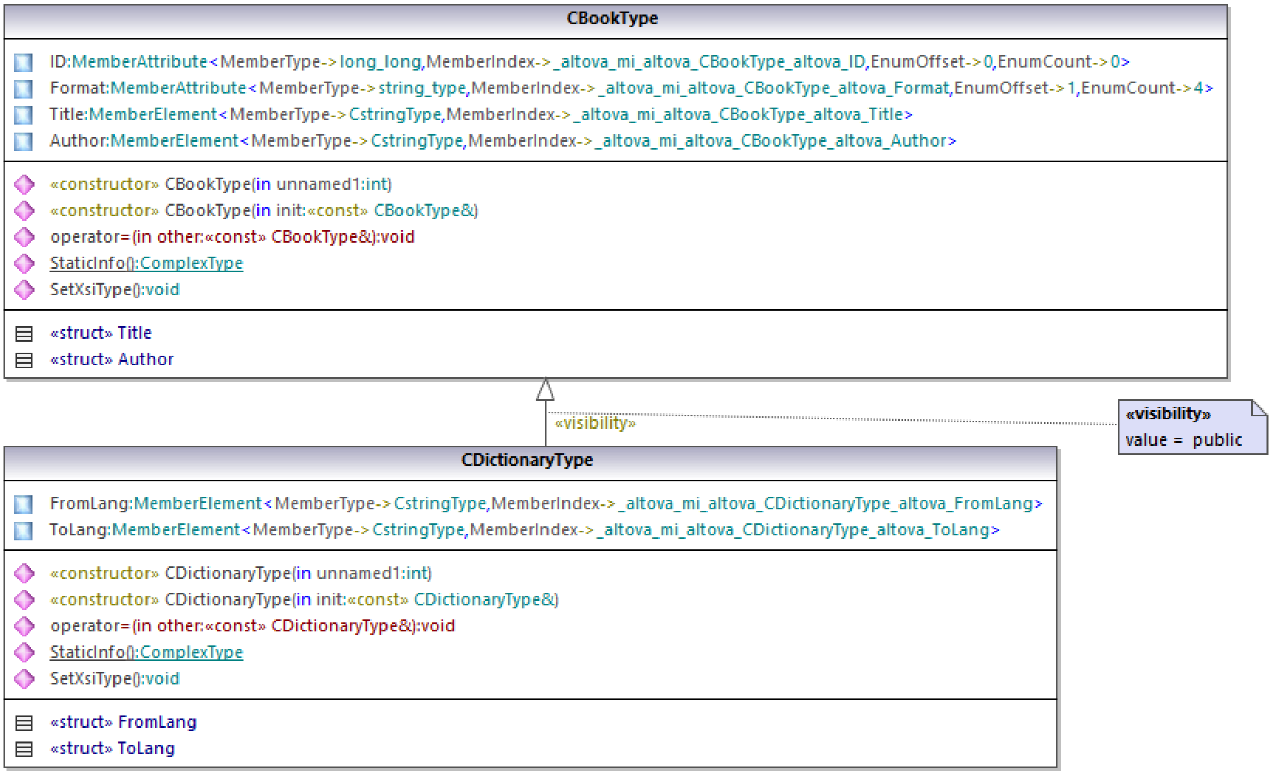
CLibraryType

LastUpdated:MemberAttribute<MemberType->DateTime,MemberIndex->_altova_mi_altova_CLibraryType_altova_LastUpdated,EnumOffset->0,EnumCount->0>
Book:MemberElement<MemberType->CBookType,MemberIndex->_altova_mi_altova_CLibraryType_altova_Book>

«constructor» CLibraryType(in unnamed1:int)
«constructor» CLibraryType(in init:«const» CLibraryType&)
operator=(in other:«const» CLibraryType&):void
StaticInfo():ComplexType

«struct» Book

The `DictionaryType` is a complex type derived from **BookType** in the schema, so this relationship is also reflected in the generated classes. As illustrated in the diagram, the class `CDictionaryType` inherits the `CBookType` class.


CBookType
ID:MemberAttribute<MemberType->long_long,MemberIndex->_altova_mi_altova_CBookType_altova_ID,EnumOffset->0,EnumCount->0>
Format:MemberAttribute<MemberType->string_type,MemberIndex->_altova_mi_altova_CBookType_altova_Format,EnumOffset->1,EnumCount->4>
Title:MemberElement<MemberType->CstringType,MemberIndex->_altova_mi_altova_CBookType_altova_Title>
Author:MemberElement<MemberType->CstringType,MemberIndex->_altova_mi_altova_CBookType_altova_Author>
«constructor» CBookType(in unnamed1:int)
«constructor» CBookType(in init:«const» CBookType&)
operator=(in other:«const» CBookType&):void
StaticInfo():ComplexType
SetXsiType():void
«struct» Title
«struct» Author
«visibility»
«visibility»
value = public
CDictionaryType
FromLang:MemberElement<MemberType->CstringType,MemberIndex->_altova_mi_altova_CDictionaryType_altova_FromLang>
ToLang:MemberElement<MemberType->CstringType,MemberIndex->_altova_mi_altova_CDictionaryType_altova_ToLang>
«constructor» CDictionaryType(in unnamed1:int)
«constructor» CDictionaryType(in init:«const» CDictionaryType&)
operator=(in other:«const» CDictionaryType&):void
StaticInfo():ComplexType
SetXsiType():void
«struct» FromLang
«struct» ToLang

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 895).

2. In Solution Explorer, open the **LibraryTest.cpp** file, and edit the `Example()` method as shown below.

```
#include <ctime> // required to get current time
using namespace Doc; // required to work with Altova libraries

void Example()
{
   // Create a new, empty XML document
   CDoc libDoc = CDoc::CreateDocument();

   // Create the root element <Library> and add it to the document
   CLibraryType lib = libDoc.Library.append();

   // Get current time and set the "LastUpdated" attribute using Altova classes
   time_t t = time(NULL);
   struct tm * now = localtime( & t );
   altova::DateTime dt = altova::DateTime(now->tm_year + 1900, now->tm_mon + 1, now-
>tm_mday, now->tm_hour, now->tm_min, now->tm_sec);
   lib.LastUpdated = dt;

   // Create a new <Book> and add it to the library
   CBookType book = lib.Book.append();

   // Set the "ID" attribute of the book
   book.ID = 1;

   // Set the "Format" attribute of the <Book> using an enumeration constant
   book.Format.SetEnumerationValue( CBookFormatType::k_Paperback );

   // Add the <Title> and <Author> elements, and set values
   book.Title.append() = _T("The XML Spy Handbook");
   book.Author.append() = _T("Altova");

   // Append a dictionary (book of derived type) and populate its attributes and elements
   CDictionaryType dictionary = CDictionaryType(lib.Book.append().GetNode());
   dictionary.ID = 2;
   dictionary.Format.SetEnumerationValue( CBookFormatType::k_E_book);
   dictionary.Title.append() = _T("English-German Dictionary");
   dictionary.Author.append() = _T("John Doe");
   dictionary.FromLang.append() = _T("English");
   dictionary.ToLang.append() = _T("German");

   // Since dictionary a derived type, set the xsi:type attribute of the book element
   dictionary.SetXsiType();

   // Optionally, set the schema location
   libDoc.SetSchemaLocation(_T("Library.xsd"));

   // Save the XML document to a file with default encoding (UTF-8),
   // "true" causes the file to be pretty-printed.
   libDoc.SaveToFile(_T("GeneratedLibrary.xml"), true);

   // Destroy the document
   libDoc.DestroyDocument();
}
```

3. Press **F5** to start debugging. If the code was executed successfully, a **GeneratedLibrary.xml** file is created in the solution output directory.

Reading an XML document

1. Open the **LibraryTest.sln** solution in Visual Studio.
2. Save the code below as **Library1.xml** to a directory that can be read by the program code (for example, the same directory as **LibraryTest.sln**).

```
<?xml 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 Doc;
void Example()
{
   // Load XML document
   CDoc libDoc = CDoc::LoadFromFile(_T("Library1.xml"));

   // Get the first (and only) root element <Library>
   CLibraryType lib = libDoc.Library.first();

   // Check whether an element exists:
   if (!lib.Book.exists())
   {
      tcout << "This library is empty." << std::endl;
      return;
   }

   // iteration: for each <Book>...
   for (Iterator<CBookType> itBook = lib.Book.all(); itBook; ++itBook)
   {
      // output values of ISBN attribute and (first and only) title element
      tcout << "ID: " << itBook->ID << std::endl;
      tcout << "Title: " << tstring(itBook->Title.first()) << std::endl;

      // read and compare an enumeration value
      if (itBook->Format.GetEnumerationValue() == CBookFormatType::k_Paperback)
```

```
         tcout << "This is a paperback book." << std::endl;

      // for each <Author>...
      for (CBookType::Author::iterator itAuthor = itBook->Author.all(); itAuthor; +
+itAuthor)
         tcout << "Author: " << tstring(itAuthor) << std::endl;

      // alternative: use count and index
      for (unsigned int j = 0; j < itBook->Author.count(); ++j)
         tcout << "Author: " << tstring(itBook->Author[j]) << std::endl;
   }

   // Destroy the document
   libDoc.DestroyDocument();
}
```

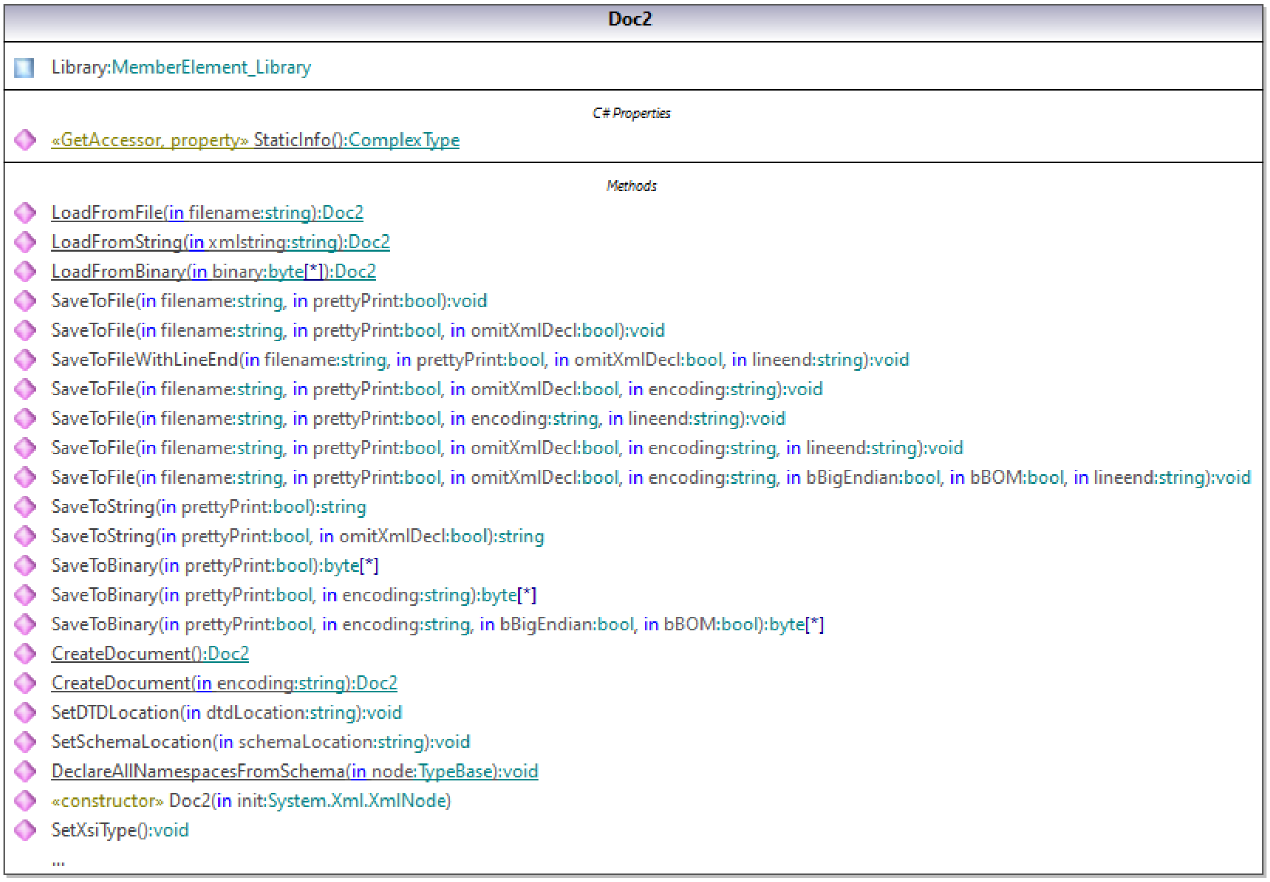
4. Press **F5** to start debugging.

14.2.3.5.2 Reading and Writing XML Documents (C#)

After you generate code from the example schema[898], a test C# application is created, along with several supporting Altova libraries.

About the generated C# libraries

The central class of the generated code is the `Doc2` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name. Note that this class is called `Doc2` to avoid a possible conflict with the namespace name. As shown in the diagram, this class provides methods for loading documents from files, binary streams, or strings (or saving documents to files, streams, strings). For a description of this class, see the class reference ([YourSchema].[Doc][954]).



Doc2

Library:MemberElement_Library

C# Properties

«GetAccessor, property» StaticInfo():ComplexType

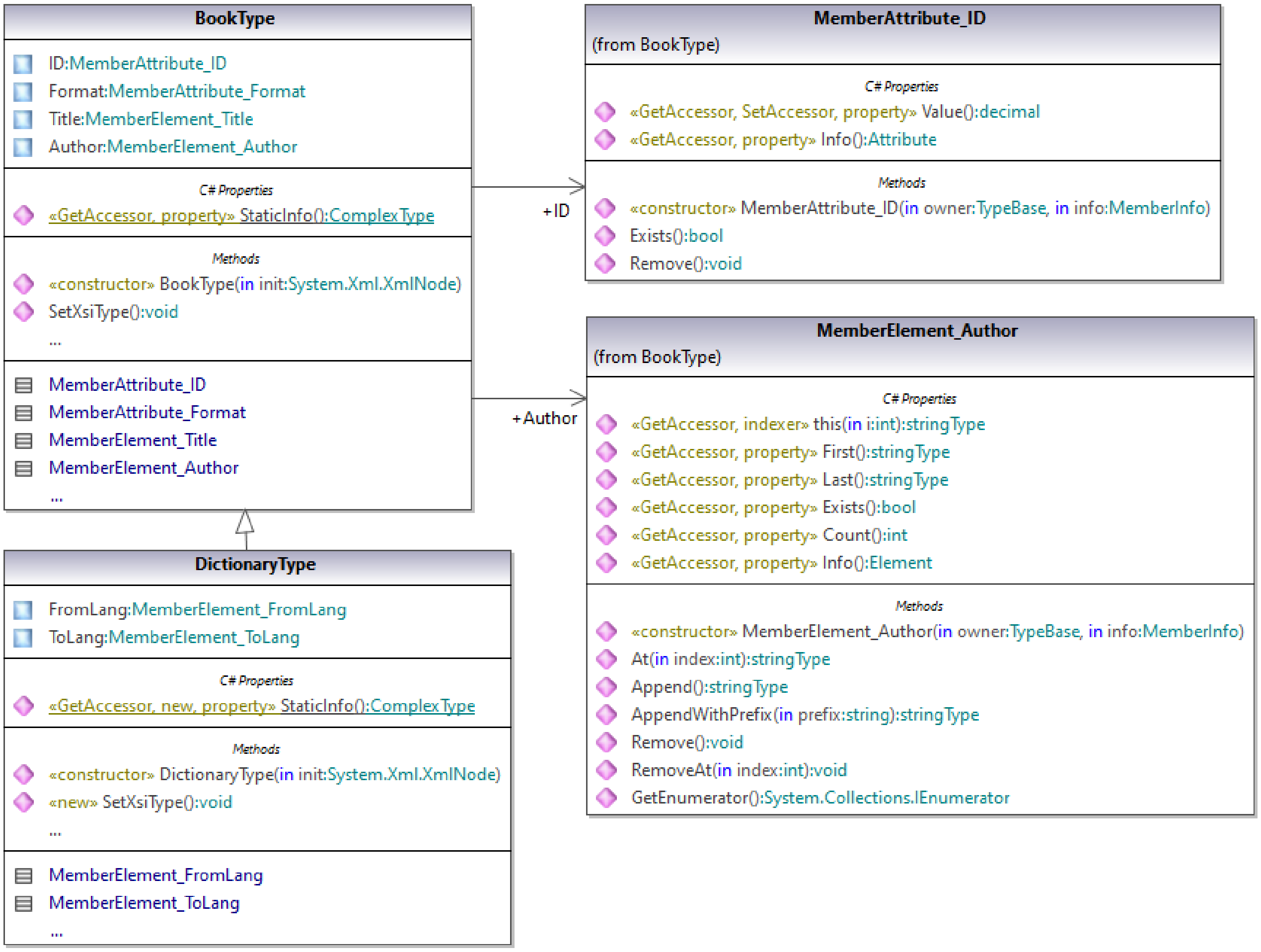
Methods

LoadFromFile(in filename:string):Doc2
LoadFromString(in xmlstring:string):Doc2
LoadFromBinary(in binary:byte[*]):Doc2
SaveToFile(in filename:string, in prettyPrint:bool):void
SaveToFile(in filename:string, in prettyPrint:bool, in omitXmlDecl:bool):void
SaveToFileWithLineEnd(in filename:string, in prettyPrint:bool, in omitXmlDecl:bool, in lineend:string):void
SaveToFile(in filename:string, in prettyPrint:bool, in omitXmlDecl:bool, in encoding:string):void
SaveToFile(in filename:string, in prettyPrint:bool, in encoding:string, in lineend:string):void
SaveToFile(in filename:string, in prettyPrint:bool, in omitXmlDecl:bool, in encoding:string, in lineend:string):void
SaveToFile(in filename:string, in prettyPrint:bool, in omitXmlDecl:bool, in encoding:string, in bBigEndian:bool, in bBOM:bool, in lineend:string):void
SaveToString(in prettyPrint:bool):string
SaveToString(in prettyPrint:bool, in omitXmlDecl:bool):string
SaveToBinary(in prettyPrint:bool):byte[*]
SaveToBinary(in prettyPrint:bool, in encoding:string):byte[*]
SaveToBinary(in prettyPrint:bool, in encoding:string, in bBigEndian:bool, in bBOM:bool):byte[*]
CreateDocument():Doc2
CreateDocument(in encoding:string):Doc2
SetDTDLocation(in dtdLocation:string):void
SetSchemaLocation(in schemaLocation:string):void
DeclareAllNamespacesFromSchema(in node:TypeBase):void
«constructor» Doc2(in init:System.Xml.XmlNode)
SetXsiType():void
...

The `Library` member of the `Doc2` class represents the actual root of the document.

According to the code generation rules mentioned in About Schema Wrapper Libraries (C#) 891, member classes are generated for each attribute and for each element of a type. In the generated code, the name of such member classes is prefixed with `MemberAttribute_` and `MemberElement_`, respectively. Examples of such classes are `MemberAttribute_ID` and `MemberElement_Author`, generated from the **Author** element and **ID** attribute of a book, respectively (in the diagram below, they are classes nested under `BookType`). Such classes enable you to manipulate programmatically the corresponding elements and attributes in the instance XML document (for example, append, remove, set value, etc). For more information, see the `[YourSchemaType].MemberAttribute` 957 and `[YourSchemaType].MemberElement` 957 class reference.

Since the **DictionaryType** is a complex type derived from **BookType** in the schema, this relationship is also reflected in the generated classes. As illustrated in the diagram below, the class `DictionaryType` inherits the `BookType` class.


BookType
ID:MemberAttribute_ID
Format:MemberAttribute_Format
Title:MemberElement_Title
Author:MemberElement_Author
C# Properties
«GetAccessor, property» StaticInfo():ComplexType
Methods
«constructor» BookType(in init:System.Xml.XmlNode)
SetXsiType():void
...
MemberAttribute_ID
MemberAttribute_Format
MemberElement_Title
MemberElement_Author
...
+ID
+Author
MemberAttribute_ID
(from BookType)
C# Properties
«GetAccessor, SetAccessor, property» Value():decimal
«GetAccessor, property» Info():Attribute
Methods
«constructor» MemberAttribute_ID(in owner:TypeBase, in info:MemberInfo)
Exists():bool
Remove():void
MemberElement_Author
(from BookType)
C# Properties
«GetAccessor, indexer» this(in i:int):stringType
«GetAccessor, property» First():stringType
«GetAccessor, property» Last():stringType
«GetAccessor, property» Exists():bool
«GetAccessor, property» Count():int
«GetAccessor, property» Info():Element
Methods
«constructor» MemberElement_Author(in owner:TypeBase, in info:MemberInfo)
At(in index:int):stringType
Append():stringType
AppendWithPrefix(in prefix:string):stringType
Remove():void
RemoveAt(in index:int):void
GetEnumerator():System.Collections.IEnumerator
DictionaryType
FromLang:MemberElement_FromLang
ToLang:MemberElement_ToLang
C# Properties
«GetAccessor, new, property» StaticInfo():ComplexType
Methods
«constructor» DictionaryType(in init:System.Xml.XmlNode)
«new» SetXsiType():void
...
MemberElement_FromLang
MemberElement_ToLang
...

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 895).

2. In Solution Explorer, open the **LibraryTest.cs** file, and edit the `Example()` method as shown below.

```
protected static void Example()
{
    // Create a new XML document
    Doc2 doc = Doc2.CreateDocument();
    // Append the root element
    LibraryType root = doc.Library.Append();

    // Create the generation date using Altova DateTime class
    Altova.Types.DateTime dt = new Altova.Types.DateTime(System.DateTime.Now);
    // Append the date to the root
    root.LastUpdated.Value = dt;

    // Add a new book
    BookType book = root.Book.Append();
    // Set the value of the ID attribute
    book.ID.Value = 1;
    // Set the format of the book (enumeration)
    book.Format.EnumerationValue = BookFormatType.EnumValues.eHardcover;
    // Set the Title and Author elements
    book.Title.Append().Value = "The XMLSpy Handbook";
    book.Author.Append().Value = "Altova";

    // Append a dictionary (book of derived type) and populate its attributes and
elements
    DictionaryType dictionary = new DictionaryType(root.Book.Append().Node);
    dictionary.ID.Value = 2;
    dictionary.Title.Append().Value = "English-German Dictionary";
    dictionary.Format.EnumerationValue = BookFormatType.EnumValues.eE_book;
    dictionary.Author.Append().Value = "John Doe";
    dictionary.FromLang.Append().Value = "English";
    dictionary.ToLang.Append().Value = "German";
    // Since it's a derived type, make sure to set the xsi:type attribute of the
book element
    dictionary.SetXsiType();

    // Optionally, set the schema location (adjust the path if
    // your schema is not in the same folder as the generated instance file)
    doc.SetSchemaLocation("Library.xsd");

    // Save the XML document with the "pretty print" option enabled
    doc.SaveToFile("GeneratedLibrary.xml", true);
}
```

3. Press **F5** to start debugging. If the code was executed successfully, a **GeneratedLibrary.xml** file is created in the solution output directory (typically, **bin/Debug**).

Reading an XML document

1. Open the **LibraryTest.sln** solution in Visual Studio.
2. Save the code below as **Library.xml** to the output directory of the project (by default, **bin/Debug**). This is the file that will be read by the program code.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.nanonull.com/LibrarySample"
xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" LastUpdated="2016-
02-03T17:10:08.4977404">
   <Book ID="1" Format="E-book">
      <Title>The XMLSpy Handbook</Title>
      <Author>Altova</Author>
   </Book>
   <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample"
xsi:type="n1:DictionaryType">
      <Title>English-German Dictionary</Title>
      <Author>John Doe</Author>
      <FromLang>English</FromLang>
      <ToLang>German</ToLang>
   </Book>
</Library>
```

3. In Solution Explorer, open the **LibraryTest.cs** file, and edit the `Example()` method as shown below.

```
protected static void Example()
{
    // Load the XML file
    Doc2 doc = Doc2.LoadFromFile("Library.xml");
    // Get the root element
    LibraryType root = doc.Library.First;

    // Read the library generation date
    Altova.Types.DateTime dt = root.LastUpdated.Value;
    string dt_as_string = dt.ToString(DateTimeFormat.W3_dateTime);
    Console.WriteLine("The library generation date is: " + dt_as_string);

    // Iteration: for each <Book>...
    foreach (BookType book in root.Book)
    {
      // Output values of ID attribute and (first and only) title element
      Console.WriteLine("ID:    " + book.ID.Value);
      Console.WriteLine("Title: " + book.Title.First.Value);

      // Read and compare an enumeration value
      if (book.Format.EnumerationValue == BookFormatType.EnumValues.ePaperback)
         Console.WriteLine("This is a paperback book.");

      // Iteration: for each <Author>
      foreach (xs.stringType author in book.Author)
         Console.WriteLine("Author: " + author.Value);

      // Determine if this book is of derived type
      if (book.Node.Attributes.GetNamedItem("xsi:type") != null)
      {
         // Find the value of the xsi:type attribute
         string xsiTypeValue =
book.Node.Attributes.GetNamedItem("xsi:type").Value;
```

```
         // Get the namespace URI and the lookup prefix of this namespace
         string namespaceUri = book.Node.NamespaceURI;
         string prefix = book.Node.GetPrefixOfNamespace(namespaceUri);

         // if this book has DictionaryType
         if (namespaceUri == "http://www.nanonull.com/LibrarySample" &&
xsiTypeValue.Equals(prefix + ":DictionaryType"))
         {
            // output additional fields
            DictionaryType dictionary = new DictionaryType(book.Node);
            Console.WriteLine("Language from: " +
dictionary.FromLang.First.Value);
            Console.WriteLine("Language to: " + dictionary.ToLang.First.Value);
         }
         else
         {
            throw new Exception("Unexpected book type");
         }
      }
    }

    Console.ReadLine();
}
```

4. Press **F5** to start debugging. If the code was executed successfully, **Library.xml** will be read by the program code, and its contents displayed as console output.

Reading and writing elements and attributes

Values of attributes and elements can be accessed using the `Value` property of the generated member element or attribute class, for example:

```
// Output values of ID attribute and (first and only) title element
Console.WriteLine("ID:    " + book.ID.Value);
Console.WriteLine("Title: " + book.Title.First.Value);
```

To get the value of the **Title** element in this particular example, we also used the `First()` method, since this is the first (and only) **Title** element of a book. For cases when you need to pick a specific element from a list by index, use the `At()` method.

The class generated for each member element of a type implements the standard `System.Collections.IEnumerable` interface. This makes it possible to loop through multiple elements of the same type. In this particular example, you can loop through all books of a Library object as follows:

```
// Iteration: for each <Book>...
foreach (BookType book in root.Book)
{
    // your code here...
}
```

To add a new element, use the `Append()` method. For example, the following code appends the root element to the document:

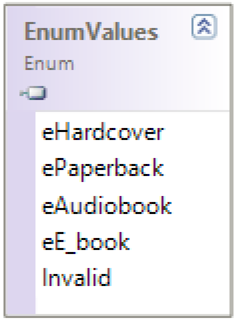
```
// Append the root element to the library
LibraryType root = doc.Library.Append();
```

You can set the value of an attribute (like ID in this example) as follows:

```
// Set the value of the ID attribute
book.ID.Value = 1;
```

Reading and writing enumeration values

If your XML schema defines simple types as enumerations, the enumerated values become available as `Enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `Enum`:



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` 944 or `Altova.Types.Duration` 949 object (either from `System.DateTime`, or by using parts such as hours and minutes, see `Altova.Types.DateTime` 944 and `Altova.Types.Duration` 949 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` 944 or `Altova.Types.Duration` 949 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` 944 and `Altova.Types.Duration` 949 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:

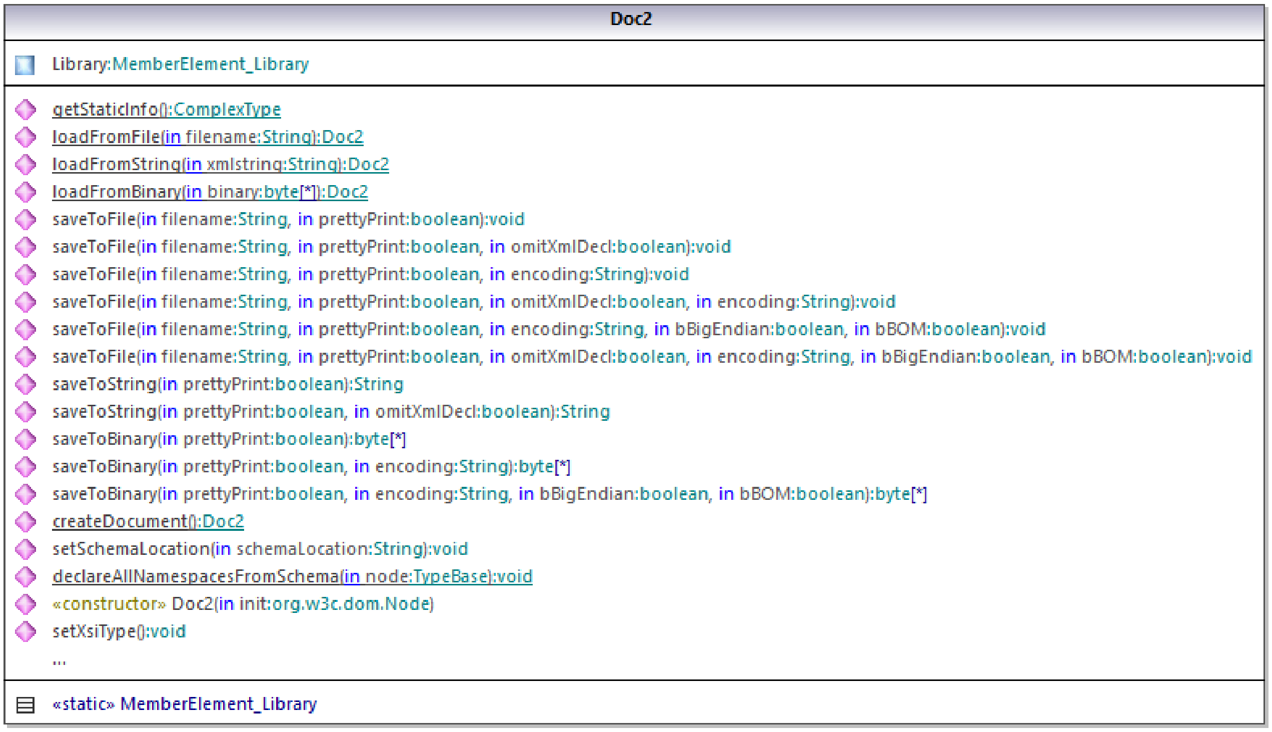
```
      // Determine if this book is of derived type
      if (book.Node.Attributes.GetNamedItem("xsi:type") != null)
      {
          // Find the value of the xsi:type attribute
          string xsiTypeValue = book.Node.Attributes.GetNamedItem("xsi:type").Value;
          // Get the namespace URI 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");
          }
      }
```

14.2.3.5.3 Reading and Writing XML Documents (Java)

After you generate code from the example schema 898, a test Java project is created, along with several supporting Altova libraries.

About the generated Java libraries

The central class of the generated code is the `Doc2` class, which represents the XML document. Such a class is generated for every schema and its name depends on the schema file name. Note that this class is called `Doc2` to avoid a possible conflict with the namespace name. As shown in the diagram, this class provides methods for loading documents from files, binary streams, or strings (or saving documents to files, streams, strings). For a description of this class, see the `com.[YourSchema].[Doc]` 969 class reference.

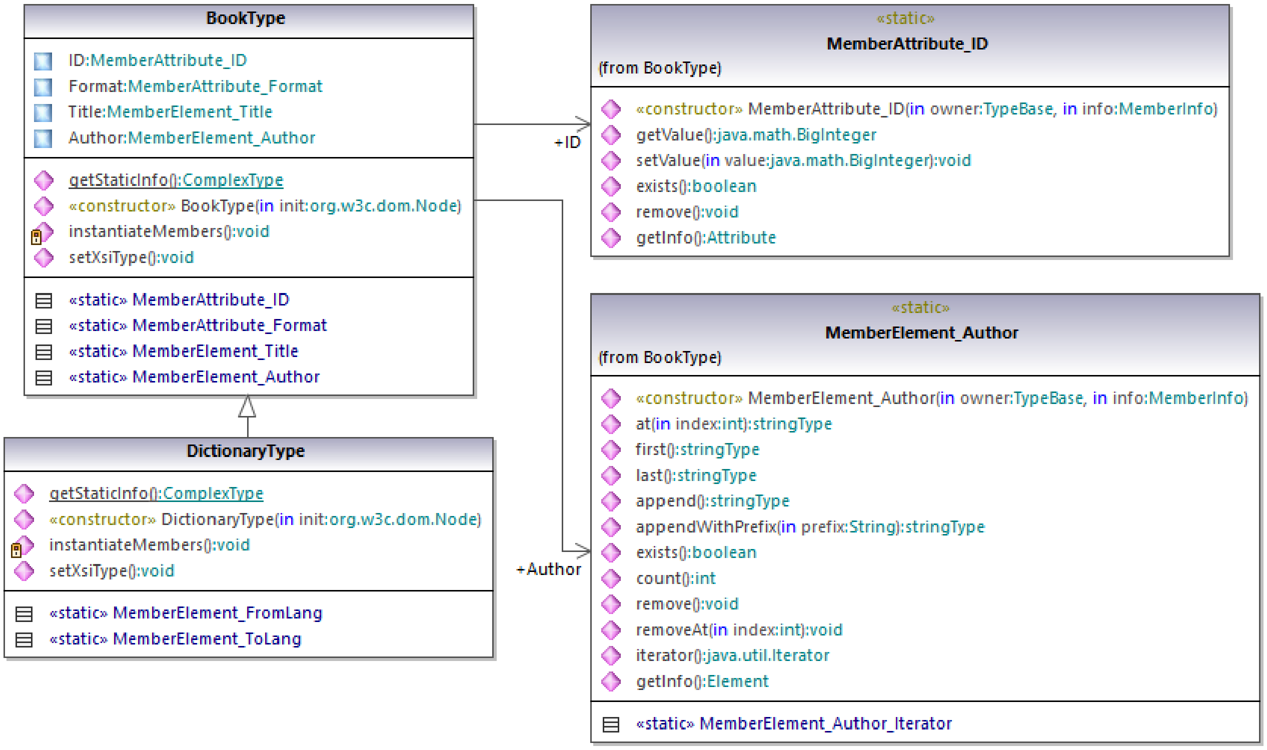


```
Doc2
Library:MemberElement_Library
getStaticInfo():ComplexType
loadFromFile(in filename:String):Doc2
loadFromString(in xmlstring:String):Doc2
loadFromBinary(in binary:byte[*]):Doc2
saveToFile(in filename:String, in prettyPrint:boolean):void
saveToFile(in filename:String, in prettyPrint:boolean, in 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 encoding:String):void
saveToFile(in filename:String, in prettyPrint:boolean, in encoding:String, in bBigEndian:boolean, in bBOM:boolean):void
saveToFile(in filename:String, in prettyPrint:boolean, in omitXmlDecl:boolean, in encoding:String, in bBigEndian:boolean, in bBOM:boolean):void
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 bBOM:boolean):byte[*]
createDocument():Doc2
setSchemaLocation(in schemaLocation:String):void
declareAllNamespacesFromSchema(in node:TypeBase):void
«constructor» Doc2(in init:org.w3c.dom.Node)
setXsiType():void
...
«static» MemberElement_Library
```

The `Library` member of the `Doc2` class represents the actual root of the document.

According to the code generation rules mentioned in About Generated Java Code 893, member classes are generated for each attribute and for each element of a type. In the generated code, the name of such member classes is prefixed with `MemberAttribute_` and `MemberElement_`, respectively. In the diagram below, examples of such classes are `MemberAttribute_ID` and `MemberElement_Author`, generated from the **Author** element and **ID** attribute of a book, respectively. Such classes enable you to manipulate programmatically the corresponding elements and attributes in the instance XML document (for example, append, remove, set value, etc). For more information, see the `com.[YourSchema].[YourSchemaType].MemberAttribute` 972 and `com.[YourSchema].[YourSchemaType].MemberElement` 972 class reference.

Since the **DictionaryType** is a complex type derived from **BookType** in the schema, this relationship is also reflected in the generated classes. As illustrated in the diagram below, the class `DictionaryType` inherits the `BookType` class.


BookType
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
DictionaryType
getStaticInfo():ComplexType
«constructor» DictionaryType(in init:org.w3c.dom.Node)
instantiateMembers():void
setXsiType():void
«static» MemberElement_FromLang
«static» MemberElement_ToLang
+ID
+Author
«static»
MemberAttribute_ID
(from BookType)
«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
«static»
MemberElement_Author
(from BookType)
«constructor» MemberElement_Author(in owner:TypeBase, in info:MemberInfo)
at(in index:int):stringType
first():stringType
last():stringType
append():stringType
appendWithPrefix(in prefix:String):stringType
exists():boolean
count():int
remove():void
removeAt(in index:int):void
iterator():java.util.Iterator
getInfo():Element
«static» MemberElement_Author_Iterator

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 895).

4. Edit the `Example()` method as shown below.

```
protected static void example() throws Exception {
      // create a new, empty XML document
      Doc2 libDoc = Doc2.createDocument();

      // create the root element <Library> and add it to the document
      LibraryType lib = libDoc.Library.append();

      // set the "LastUpdated" attribute
      com.altova.types.DateTime dt = new com.altova.types.DateTime(DateTime.now());
      lib.LastUpdated.setValue(dt);

      // create a new <Book> and populate its elements and attributes
      BookType book = lib.Book.append();
      book.ID.setValue(java.math.BigInteger.valueOf(1));
      book.Format.setEnumerationValue( BookFormatType.EPAPERBACK );
      book.Title.append().setValue("The XML Spy Handbook");
      book.Author.append().setValue("Altova");

      // create a dictionary (book of derived type)  and populate its elements and
attributes
      DictionaryType dict = new DictionaryType(lib.Book.append().getNode());
      dict.ID.setValue(java.math.BigInteger.valueOf(2));
      dict.Title.append().setValue("English-German Dictionary");
      dict.Format.setEnumerationValue(BookFormatType.EE_BOOK);
      dict.Author.append().setValue("John Doe");
      dict.FromLang.append().setValue("English");
      dict.ToLang.append().setValue("German");
      dict.setXsiType();

      // set the schema location (this is optional)
      libDoc.setSchemaLocation("Library.xsd");

      // save the XML document to a file with default encoding (UTF-8). "true" causes the
file to be pretty-printed.
      libDoc.saveToFile("Library1.xml", true);
   }
```

5. Build the Java project and run it. If the code is executed successfully, a **Library1.xml** file is created in the project directory.

Reading an XML document

1. On the **File** menu of Eclipse, click **Import**, select **Existing Projects into Workspace**, and click **Next**.
2. Next to **Select root directory**, click **Browse**, select the directory to which you generated the Java code, and then click **Finish**.
3. Save the code below as **Library1.xml** to a local directory (you will need to refer to the path of the **Library1.xml** file from the sample code below).

```
<?xml version="1.0" encoding="utf-8"?>
<Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.nanonull.com/LibrarySample"
xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" LastUpdated="2016-
02-03T17:10:08.4977404">
```

```
   <Book ID="1" Format="E-book">
      <Title>The XMLSpy Handbook</Title>
      <Author>Altova</Author>
   </Book>
   <Book ID="2" Format="Paperback" xmlns:n1="http://www.nanonull.com/LibrarySample"
xsi:type="n1:DictionaryType">
      <Title>English-German Dictionary</Title>
      <Author>John Doe</Author>
      <FromLang>English</FromLang>
      <ToLang>German</ToLang>
   </Book>
</Library>
```

4. In the Eclipse Package Explorer, expand the **com.LibraryTest** package and open the **LibraryTest.java** file.
5. Edit the `Example()` method as shown below.

```
protected static void example() throws Exception {
     // load XML document from a path, make sure to adjust the path as necessary
     Doc2 libDoc = Doc2.loadFromFile("Library1.xml");

     // get the first (and only) root element <Library>
     LibraryType lib = libDoc.Library.first();

     // check whether an element exists:
     if (!lib.Book.exists()) {
        System.out.println("This library is empty.");
        return;
     }

     // read a DateTime schema type
     com.altova.types.DateTime dt = lib.LastUpdated.getValue();
     System.out.println("The library was last updated on: " + dt.toDateString());

     // iteration: for each <Book>...
     for (java.util.Iterator itBook = lib.Book.iterator(); itBook.hasNext();) {
        BookType book = (BookType) itBook.next();
        // output values of ID attribute and (first and only) title element
        System.out.println("ID: " + book.ID.getValue());
        System.out.println("Title: " + book.Title.first().getValue());

        // read and compare an enumeration value
        if (book.Format.getEnumerationValue() == BookFormatType.EPAPERBACK)
           System.out.println("This is a paperback book.");

        // for each <Author>...
        for (java.util.Iterator itAuthor = book.Author.iterator(); itAuthor
              .hasNext();)
           System.out.println("Author: " + ((com.Doc.xs.stringType)
itAuthor.next()).getValue());

        // find the derived type of this book
        // by looking at the value of the xsi:type attribute, using DOM
        org.w3c.dom.Node bookNode = book.getNode();
        if (bookNode.getAttributes().getNamedItem("xsi:type") != null) {
           // Get the value of the xsi:type attribute
           String xsiTypeValue =
bookNode.getAttributes().getNamedItem("xsi:type").getNodeValue();

           // Get the namespace URI and lookup prefix of this namespace
           String namespaceUri = bookNode.getNamespaceURI();
           String lookupPrefix = bookNode.lookupPrefix(namespaceUri);

           // If xsi:type matches the namespace URI and type of the book node
           if (namespaceUri == "http://www.nanonull.com/LibrarySample"
                 && (  xsiTypeValue.equals(lookupPrefix + ":DictionaryType" )))    {
              // ...then this is a book of derived type (dictionary)
              DictionaryType dictionary = new DictionaryType(   book.getNode());
              // output the value of the "FromLang" and "ToLang" elements
              System.out.println("From language: " +
dictionary.FromLang.first().getValue());
              System.out.println("To language: " + dictionary.ToLang.first().getValue());
           }
           else
           {
              // throw an error
              throw new java.lang.Error("This book has an unknown type.");
           }
        }
     }
   }
```

6. Build the Java project and run it. If the code is executed successfully, **Library1.xml** will be read by the program code, and its contents displayed in the Console view.

Reading and writing elements and attributes

Values of attributes and elements can be accessed using the `getValue()` method of the generated member element or attribute class, for example:

```
// output values of ID attribute and (first and only) title element
System.out.println("ID: " + book.ID.getValue());
System.out.println("Title: " + book.Title.first().getValue());
```

To get the value of the **Title** element in this particular example, we also used the `first()` method, since this is the first (and only) **Title** element of a book. For cases when you need to pick a specific element from a list by index, use the `at()` method.

To iterate through multiple elements, use either index-based iteration or `java.util.Iterator`. For example, you can iterate through the books of a library as follows:

```
// index-based iteration
for (int j = 0; j < lib.Book.count(); ++j ) {
   // your code here
}

// alternative iteration using java.util.Iterator
for (java.util.Iterator itBook = lib.Book.iterator(); itBook.hasNext();) {
      // your code here
   }
```

To add a new element, use the `append()` method. For example, the following code appends an empty root **Library** element to the document:

```
// create the root element <Library> and add it to the document
LibraryType lib = libDoc.Library.append();
```

Once an element is appended, you can set the value of any of its elements or an attributes by using the setValue() method.

```
// set the value of the Title element
book.Title.append().setValue("The XML Spy Handbook");
// set the value of the ID attribute
book.ID.setValue(java.math.BigInteger.valueOf(1));
```

Reading and writing enumeration values

If your XML schema defines simple types as enumerations, the enumerated values become available as `Enum` values in the generated code. In the schema used in this example, a book format can be hardcover, paperback, e-book, and so on. Therefore, in the generated code, these values would be available through an `Enum` (see the `BookFormatType` class diagram above). To assign enumeration values to an object, use code such as the one below:

```
// set an enumeration value
book.Format.setEnumerationValue( BookFormatType.EPAPERBACK );
```

You can read such enumeration values from XML instance documents as follows:

```
// read an enumeration value
if (book.Format.getEnumerationValue() == BookFormatType.EPAPERBACK)
         System.out.println("This is a paperback book."
```

When an "if" condition is not enough, create a switch to determine each enumeration value and process it as required.

Working with xs:dateTime and xs:duration types

If the schema from which you generated code uses time and duration types such as `xs:dateTime`, or `xs:duration`, these are converted to Altova native classes in generated code. Therefore, to write a date or duration value to the XML document, do the following:

1. Construct a `com.altova.types.DateTime` 959 or `com.altova.types.Duration` 963 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` 959 or `com.altova.types.Duration` 963 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` 959 and `com.altova.types.Duration` 963 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.");
            }
         }
```

14.2.3.6 Example: Purchase Order

This example illustrates how to work with program code generated from a "main" XML schema that imports other schemas. Each of the imported schema has a different target namespace. The goal here is to create programmatically an XML document where all elements are prefixed according to their namespace. More specifically, the XML document created from your C++, C#, or Java code should look like the one below:

```
<?xml version="1.0" encoding="utf-8"?>
<p:Purchase xsi:schemaLocation="http://NamespaceTest.com/Purchase Main.xsd"
            xmlns:p="http://NamespaceTest.com/Purchase"
            xmlns:o="http://NamespaceTest.com/OrderTypes"
            xmlns:c="http://NamespaceTest.com/CustomerTypes"
            xmlns:cmn="http://NamespaceTest.com/CommonTypes"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <p:OrderDetail>
        <o:Item>
            <o:ProductName>Lawnmower</o:ProductName>
            <o:Quantity>1</o:Quantity>
            <o:UnitPrice>148.42</o:UnitPrice>
        </o:Item>
    </p:OrderDetail>
    <p:PaymentMethod>VISA</p:PaymentMethod>
    <p:CustomerDetails>
        <c:Name>Alice Smith</c:Name>
        <c:DeliveryAddress>
            <cmn:Line1>123 Maple Street</cmn:Line1>
            <cmn:Line2>Mill Valley</cmn:Line2>
        </c:DeliveryAddress>
        <c:BillingAddress>
            <cmn:Line1>8 Oak Avenue</cmn:Line1>
            <cmn:Line2>Old Town</cmn:Line2>
        </c:BillingAddress>
    </p:CustomerDetails>
</p:Purchase>
```

The main schema used in this example is called **Main.xsd**. As illustrated in the code listing below, it imports three other schemas: **CommonTypes.xsd**, **CustomerTypes.xsd**, and **OrderTypes.xsd**. To get the same results as in this example, save all the code listings below to files, and use the same file names as above. Notice that the schema maps each of the prefixes `ord`, `pur`, `cmn`, and `cust` to some namespace (Order types, Purchase types, Common types, and Customer types, respectively). This means that, in the generated code, the classes corresponding to Orders, Purchases, Customers, and so on, will be available under their respective namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://NamespaceTest.com/Purchase"
      xmlns:ord="http://NamespaceTest.com/OrderTypes"
      xmlns:pur="http://NamespaceTest.com/Purchase"
      xmlns:cmn="http://NamespaceTest.com/CommonTypes"
      xmlns:cust="http://NamespaceTest.com/CustomerTypes"
      elementFormDefault="qualified">
    <xs:import schemaLocation="CommonTypes.xsd"
namespace="http://NamespaceTest.com/CommonTypes" />
    <xs:import schemaLocation="CustomerTypes.xsd"
namespace="http://NamespaceTest.com/CustomerTypes" />
    <xs:import schemaLocation="OrderTypes.xsd"
namespace="http://NamespaceTest.com/OrderTypes" />
    <xs:element name="Purchase">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="OrderDetail" type="ord:OrderType" />
                <xs:element name="PaymentMethod" type="cmn:PaymentMethodType" />
                <xs:element ref="pur:CustomerDetails" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="CustomerDetails" type="cust:CustomerType" />
</xs:schema>
```

Main.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://NamespaceTest.com/CommonTypes"
      elementFormDefault="qualified">
   <xs:complexType name="AddressType">
      <xs:sequence>
         <xs:element name="Line1" type="xs:string"/>
         <xs:element name="Line2" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:simpleType name="PriceType">
      <xs:restriction base="xs:decimal">
         <xs:fractionDigits value="2"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="PaymentMethodType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="VISA"/>
         <xs:enumeration value="MasterCard"/>
         <xs:enumeration value="Cash"/>
         <xs:enumeration value="AMEX"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

CommonTypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://NamespaceTest.com/CustomerTypes"
      xmlns:cmn="http://NamespaceTest.com/CommonTypes"
      elementFormDefault="qualified">
    <xs:import schemaLocation="CommonTypes.xsd"
namespace="http://NamespaceTest.com/CommonTypes" />
    <xs:complexType name="CustomerType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" />
            <xs:element name="DeliveryAddress" type="cmn:AddressType" />
            <xs:element name="BillingAddress" type="cmn:AddressType" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

CustomerTypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://NamespaceTest.com/OrderTypes"
      xmlns:cmn="http://NamespaceTest.com/CommonTypes"
      elementFormDefault="qualified">
    <xs:import schemaLocation="CommonTypes.xsd"
namespace="http://NamespaceTest.com/CommonTypes" />
    <xs:complexType name="OrderType">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" name="Item">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="ProductName" type="xs:string" />
                        <xs:element name="Quantity" type="xs:int" />
                        <xs:element name="UnitPrice" type="cmn:PriceType" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

OrderTypes.xsd

To complete this example, take the following steps:

1. Save all schemas from the code listings above to files on the disk, making sure that you preserve the indicated file names.
2. Generate the schema wrapper code from the **Main.xsd** schema above, using the steps described in Generating Code from XML Schemas or DTD 885. After completing this step, you should have generated a compilable program in the language of your choice (C++, C#, or Java).
3. Add code to your C++, C#, or Java program from one the following example code listings, as required:

    - XML Namespaces and Prefixes (C++) 925
    - XML Namespaces and Prefixes (C#) 926
    - XML Namespaces and Prefixes (Java) 928

14.2.3.6.1 XML Namespaces and Prefixes (C++)

After you generate code from the example schema 922, a test C++ application is created, along with several supporting Altova libraries. Recall that the example schema (**Main.xsd**) has multiple namespace declarations. Consequently, the generated code includes namespaces that correspond to namespace aliases (prefixes) from the schema, namely: `Main::ord`, `Main::pur`, `Main::cmn`, and `Main::cust`.

In general, in order to control XML namespaces and prefixes with the help of the schema wrapper libraries, you have the following methods at your disposal:

- DeclareAllNamespacesFromSchema() 939. Call this method if you want to declare the same namespaces in your XML instance as in the schema. Otherwise, if you need different namespaces as in this example, then `DeclareNamespace()` should be used. The method `DeclareAllNamespacesFromSchema()` is not used in this example because we specifically want to create XML elements with prefixes that are slightly different from those declared in the schema.
- DeclareNamespace() 941. Call this method to create or override the existing namespace prefix attribute on an element. The element must already be created using either the `append()` or `appendWithPrefix()` methods, as further illustrated below.
- appendWithPrefix(). 943 Use this method to append an instance element with a specific prefix. To create the XML instance illustrated in this example, it was sufficient to call this method for the root element only. All other elements were appended using just append() 943, and their prefixes were added automatically based on their namespaces, according to the rules above.

The code listing below shows you how to create an XML document with multiple namespace declarations and prefixed element names. Specifically, it generates a Purchase Order instance as illustrated in the Example: Purchase Order 922. Importantly, for illustrative purposes, some prefixes are overridden in the XML instance (that is, they are not exactly the same as the ones declared in the schema).

```
void Example()
{
   // Create the XML document and append the root element
   Main::pur::CMain doc = Main::pur::CMain::CreateDocument();
   Main::pur::CPurchaseType purchase = doc.Purchase.appendWithPrefix(_T("p"));

   // Set schema location
   doc.SetSchemaLocation(_T("Main.xsd"));

   // Declare namespaces on root element
   purchase.DeclareNamespace(_T("o"), _T("http://NamespaceTest.com/OrderTypes"));
   purchase.DeclareNamespace(_T("c"), _T("http://NamespaceTest.com/CustomerTypes"));
   purchase.DeclareNamespace(_T("cmn"), _T("http://NamespaceTest.com/CommonTypes"));

   // Append the OrderDetail element
   Main::ord::COrderType order = purchase.OrderDetail.append();
   Main::ord::CItemType item = order.Item.append();
   item.ProductName.append() = _T("Lawnmower");
   item.Quantity.append() = 1;
   item.UnitPrice.append() = 148.42;

   // Append the PaymentMethod element
   Main::cmn::CPaymentMethodTypeType paymentMethod = purchase.PaymentMethod.append();
   paymentMethod.SetEnumerationValue(Main::cmn::CPaymentMethodTypeType::k_VISA);

   // Append the CustomerDetails element
   Main::cust::CCustomerType customer = purchase.CustomerDetails.append();
   customer.Name.append() = _T("Alice Smith");
   Main::cmn::CAddressType deliveryAddress = customer.DeliveryAddress.append();
   deliveryAddress.Line1.append() = _T("123 Maple Street");
   deliveryAddress.Line2.append() = _T("Mill Valley");
   Main::cmn::CAddressType billingAddress = customer.BillingAddress.append();
   billingAddress.Line1.append() = _T("8 Oak Avenue");
   billingAddress.Line2.append() = _T("Old Town");

   // Save to file and release object from memory
   doc.SaveToFile(_T("Main1.xml"), true);
   doc.DestroyDocument();
}
```

14.2.3.6.2 XML Namespaces and Prefixes (C#)

After you generate code from the example schema[922], a test C# application is created, along with several supporting Altova libraries. Recall that the example schema (**Main.xsd**) has multiple namespace declarations. Consequently, the generated code includes namespaces that correspond to namespace aliases (prefixes) from the schema, namely: **`Main.ord`**, **`Main.pur`**, **`Main.cmn`**, and **`Main.cust`**.

In general, in order to control XML namespaces and prefixes with the help of the schema wrapper libraries, you have the following methods at your disposal:

- DeclareAllNamespacesFromSchema()[954]. Call this method if you want to declare the same namespaces in your XML instance as in the schema. Otherwise, if you need different namespaces as in this example, then **DeclareNamespace()** should be used. The method **DeclareAllNamespacesFromSchema()** is not used in this example because we specifically want to create XML elements with prefixes that are slightly different from those declared in the schema.
- DeclareNamespace()[956]. Call this method to create or override the existing namespace prefix attribute on an element. The element must already be created using either the **Append()** or **AppendWithPrefix()** methods, as further illustrated below.
- AppendWithPrefix().[957] Use this method to append an instance element with a specific prefix. To create the XML instance illustrated in this example, it was sufficient to call this method for the root element only. All other elements were appended using just Append()[957], and their prefixes were added automatically based on their namespaces, according to the rules above.

The code listing below shows you how to create an XML document with multiple namespace declarations and prefixed element names. Specifically, it generates a Purchase Order instance as illustrated in the Example: Purchase Order[922]. Importantly, for illustrative purposes, some prefixes are overridden in the XML instance (that is, they are not exactly the same as the ones declared in the schema).

```
protected static void Example()
{
   // Create the XML document and append the root element
   pur.Main2 doc = pur.Main2.CreateDocument();
   pur.PurchaseType purchase = doc.Purchase.AppendWithPrefix("p");

   // Set schema location
   doc.SetSchemaLocation(@"Main.xsd");

   // Declare namespaces on root element
   purchase.DeclareNamespace("o", "http://NamespaceTest.com/OrderTypes");
   purchase.DeclareNamespace("c", "http://NamespaceTest.com/CustomerTypes");
   purchase.DeclareNamespace("cmn", "http://NamespaceTest.com/CommonTypes");

   // Append the OrderDetail element
   ord.OrderType order = purchase.OrderDetail.Append();
   ord.ItemType item = order.Item.Append();
   item.ProductName.Append().Value = "Lawnmower";
   item.Quantity.Append().Value = 1;
   item.UnitPrice.Append().Value = 148.42M;

   // Append the PaymentMethod element
   cmn.PaymentMethodTypeType paymentMethod = purchase.PaymentMethod.Append();
   paymentMethod.EnumerationValue = cmn.PaymentMethodTypeType.EnumValues.eVISA;

   // Append the CustomerDetails element
   cust.CustomerType customer = purchase.CustomerDetails.Append();
   customer.Name.Append().Value = "Alice Smith";
   cmn.AddressType deliveryAddress = customer.DeliveryAddress.Append();
   deliveryAddress.Line1.Append().Value = "123 Maple Street";
   deliveryAddress.Line2.Append().Value = "Mill Valley";
   cmn.AddressType billingAddress = customer.BillingAddress.Append();
   billingAddress.Line1.Append().Value = "8 Oak Avenue";
   billingAddress.Line2.Append().Value = "Old Town";

   // Save to file
   doc.SaveToFile("PurchaseOrder.xml", true);
}
```

14.2.3.6.3 XML Namespaces and Prefixes (Java)

After you generate code from the example schema[922], a test Java application is created, along with several supporting Altova libraries. Recall that the example schema (**Main.xsd**) has multiple namespace declarations. Consequently, the generated code includes namespaces that correspond to namespace aliases (prefixes) from the schema, namely: **`com.Main.ord`**, **`com.Main.pur`**, **`com.Main.cmn`**, and **`com.Main.cust`**.

In general, in order to control XML namespaces and prefixes with the help of the schema wrapper libraries, you have the following methods at your disposal:

- declareAllNamespacesFromSchema()[969]. Call this method if you want to declare the same namespaces in your XML instance as in the schema. Otherwise, if you need different namespaces as in this example, then **`declareNamespace()`** should be used. The method **`declareAllNamespacesFromSchema()`** is not used in this example because we specifically want to create XML elements with prefixes that are slightly different from those declared in the schema.
- declareNamespace()[971]. Call this method to create or override the existing namespace prefix attribute on an element. The element must already be created using either the **`append()`** or **`appendWithPrefix()`** methods, as further illustrated below.
- appendWithPrefix().[972] Use this method to append an instance element with a specific prefix. To create the XML instance illustrated in this example, it was sufficient to call this method for the root element only. All other elements were appended using just append()[972], and their prefixes were added automatically based on their namespaces, according to the rules above.

The code listing below shows you how to create an XML document with multiple namespace declarations and prefixed element names. Specifically, it generates a Purchase Order instance as illustrated in the Example: Purchase Order[922]. Importantly, for illustrative purposes, some prefixes are overridden in the XML instance (that is, they are not exactly the same as the ones declared in the schema).

```
protected static void example() throws Exception {
   // Create the XML document and append the root element
   com.Main.pur.Main2 doc = com.Main.pur.Main2.createDocument();
   com.Main.pur.PurchaseType purchase = doc.Purchase.appendWithPrefix("p");

   // Set schema location
   doc.setSchemaLocation("Main.xsd");

   // Declare namespaces on root element
   purchase.declareNamespace("o", "http://NamespaceTest.com/OrderTypes");
   purchase.declareNamespace("c", "http://NamespaceTest.com/CustomerTypes");
   purchase.declareNamespace("cmn", "http://NamespaceTest.com/CommonTypes");

   // Append the OrderDetail element
   com.Main.ord.OrderType order = purchase.OrderDetail.append();
   com.Main.ord.ItemType item = order.Item.append();
   item.ProductName.append().setValue("Lawnmower");
   item.Quantity.append().setValue(1);
   java.math.BigDecimal price = new java.math.BigDecimal("148.42");
   item.UnitPrice.append().setValue(price);

   // Append the PaymentMethod element
   com.Main.cmn.PaymentMethodTypeType paymentMethod = purchase.PaymentMethod.append();
   paymentMethod.setEnumerationValue(com.Main.cmn.PaymentMethodTypeType.EVISA);

   // Append the CustomerDetails element
   com.Main.cust.CustomerType customer = purchase.CustomerDetails.append();
   customer.Name.append().setValue("Alice Smith");
   com.Main.cmn.AddressType deliveryAddress = customer.DeliveryAddress.append();
   deliveryAddress.Line1.append().setValue("123 Maple Street");
   deliveryAddress.Line2.append().setValue("Mill Valley");
   com.Main.cmn.AddressType billingAddress = customer.BillingAddress.append();
   billingAddress.Line1.append().setValue("8 Oak Avenue");
   billingAddress.Line2.append().setValue("Old Town");

   // Save to file
   doc.saveToFile("PurchaseOrder.xml", true);
}
```

14.2.4 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 885). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code does include other supporting classes, which are not listed here and are subject to modification.

14.2.4.1 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.

Name	Description
`DateTime(int year, unsigned char month, unsigned char day, unsigned char hour, unsigned char minute, double second, short timezone)`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, second and timezone supplied as argument. The timezone is expressed in minutes and can be positive or negative. For example, the timezone "UTC-01:00" is expressed as "-60".

Methods

Name	Description
`unsigned char Day() const`	Returns the day of month of the current `DateTime` object. The return values range from 1 through 31.
`int DayOfYear() const`	Returns the day of year of the current `DateTime` object. The return values range from 1 through 366.
`bool HasTimezone() const`	Returns Boolean **true** if the current `DateTime` object has a timezone defined; **false** otherwise.
`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-

Name	Description
	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; }
}
```

14.2.4.2 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.

Name	Description
`Duration(const DayTimeDuration& dt)`	Initializes a new instance of the `Duration` class to a duration defined by the `dt` argument (see altova::DayTimeDuration[934]).
`Duration(const YearMonthDuration& ym)`	Initializes a new instance of the `Duration` class to the duration defined by the `ym` argument (see altova::YearMonthDuration[935]).
`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[935] and altova::DayTimeDuration[934]).

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[934]).
`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[935]).
`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;
}
```

14.2.4.3 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.

14.2.4.4 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.

Name	Description
`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.

14.2.4.5 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.

14.2.4.6 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.

14.2.4.7 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.

Name	Description
`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.

14.2.4.8 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.

Name	Description
`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.

14.2.4.9 [YourSchema]::[CDoc]

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the following methods. Note that, in the method names below, "CDoc" stands for the name of the generated document class itself.

Methods

Name	Description
`static CDoc CreateDocument()`	Creates a new, empty XML document. Must be released using DestroyDocument().
`static void DeclareAllNamespacesFromSchema(ElementType& node)`	Declares all namespaces from the XML Schema on the element supplied as argument (typically, the XML root element). Calling this method is useful if your schema has multiple namespace declarations, each mapped to a prefix, and you would like to declare all of them on the element supplied as argument.
`void DestroyDocument()`	Destroys a document. All references to the document and its nodes are invalidated. This must be called when you finish working with a document.
`static CDoc LoadFromBinary(const std:vector<unsigned char>& xml)`	Loads an XML document from a byte array.

Name	Description
`static CDoc LoadFromFile(const string_type& fileName)`	Loads an XML document from a file.
`static CDoc LoadFromString(const string_type& xml)`	Loads an XML document from a string.
`std::vector<unsigned char> SaveToBinary(bool prettyPrint)`	Saves an XML document to a byte array. When set to true, the `prettyPrint` argument re-formats the XML document for better readability.
`std::vector<unsigned char> SaveToBinary(bool prettyPrint, const string_type & encoding)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding.
`std::vector<unsigned char> SaveToBinary(bool prettyPrint, const string_type & encoding, bool bBigEndian, bool bBOM)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
`void SaveToFile(const string_type & fileName, bool prettyPrint)`	Saves an XML document to a file, with optional "pretty-print" formatting.
`void SaveToFile(const string_type & fileName, bool omitXmlDecl)`	Saves an XML document to a file. If the `omitXmlDecl` argument is set to true, the XML declaration will not be written.
`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 Generation 1080).
`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 Generation 1080).
`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.

Name	Description
`void SaveToFile(const string_type& fileName, bool prettyPrint, const string_type & encoding, bool bBigEndian, bool bBOM)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
`void SaveToFile(const string_type& fileName, bool prettyPrint, const string_type & encoding, bool bBigEndian, bool bBOM, const string_type & lineend)`	Saves an XML document to a file with the specified encoding and the specified line end. Byte order and Unicode byte-order mark can be specified for Unicode encodings. This method is only available if you generated the code for the Xerces3 XML library (see Generation[1080]).
`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 Generation[1080]).
`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.

14.2.4.10 [YourSchema]::[ElementType]

This class provides methods for manipulating XML elements from your schema. Methods of this class can be called on elements, not on the XML document itself. Note that, in order to call methods of this class, you don't need to instantiate the class directly. Any element created using the **`append()`** or **`appendWithPrefix()`** methods is of `[ElementType]` type.

Methods

Name	Description
`void DeclareNamespace(const string_type prefix, const string_type nsURI)`	This method takes two arguments that are both of string type: the prefix and the namespace URI that you want to use. The prefix supplied as argument will be mapped to the namespace

Name	Description

	URI value supplied as argument. If the prefix supplied as argument is empty, the method creates or overrides the default namespace declaration in the element.
	For example, let's assume that the XML document has an XML element called "purchase". If you call
	`purchase.DeclareNamespace(_T("ord"), _T("http://OrderTypes"));`
	then the XML document becomes
	`<purchase xmlns:ord="http://OrderTypes" />`
	Another example, if you call:
	`purchase.DeclareNamespace(_T(""), _T("http://OrderTypes"));`
	then the XML document becomes
	`<purchase xmlns="http://OrderTypes" />`
	Note: The declared namespace is used when appending subsequent child elements or attributes, according to the following rules:
	1. If the child namespace is the default, then use empty prefix.
	2. If the child namespace is equal to the parent one, then use the parent prefix.
	3. Otherwise, search for nearest prefix from parent to top, using the lookup algorithm described in section "B.2: Namespace Prefix Lookup" at https://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20021022/namespaces-algorithms.html.
	4. If there is no prefix for element namespace found, then use empty prefix.

14.2.4.11 [YourSchema]::MemberAttribute

When code is generated from an XML schema, a class such as this one is created for each member attribute of a type.

Methods

Name	Description
`bool exists()`	Returns true if the attribute exists.
`int GetEnumerationValue()`	Generated for enumeration types only. Returns one of the constants generated for the possible values, or "Invalid" if the value does not match any of the enumerated values in the schema.
`altova::meta::Attribute info()`	Returns an object for querying schema information (see `altova::meta::Attribute` 936).
`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.

14.2.4.12 [YourSchema]::MemberElement

When code is generated from an XML schema, a class such as this one is created for each member element of a type. In the descriptions below, "MemberType" stands for the name of the member element itself.

Methods

Name	Description
`Iterator<MemberType> all()`	Returns an object for iterating instances of the member element.
`MemberType append()`	Creates a new element and appends it to its parent.
`MemberType appendWithPrefix(string_type prefix)`	Creates a new element having the prefix supplied as argument, and appends it to its parent. For an example, see Example: Purchase Order 922.
`unsigned int count()`	Returns the count of elements.
`int GetEnumerationValue()`	Generated for enumeration types only. Returns one of the constants generated for the possible values, or **Invalid** if the value does not match any of the enumerated values in the schema.
`bool exists()`	Returns true if at least one element exists.
`MemberType first()`	Returns the first instance of the member element.
`MemberType operator[](unsigned int index)`	Returns the member element specified by the index.

Name	Description
`altova::meta::Element info()`	Returns an object for querying schema information (see `altova::meta::Element`[937]).
`MemberType last()`	Returns the last instance of the member element.
`void remove()`	Deletes all occurrences of the element from its parent.
`void removeAt(unsigned int index)`	Deletes the occurrence of the element specified by the index.
`void SetEnumerationValue(int)`	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.

14.2.5 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[885]). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code does include other supporting classes, which are not listed here and are subject to modification.

14.2.5.1 Altova.Types.DateTime

This class enables you to process XML attributes or elements that have date and time types, such as `xs:dateTime`.

Constructors

	Name	Description
◆	`DateTime(DateTime obj)`	Initializes a new instance of the `DateTime` class to the `DateTime` object supplied as argument.
◆	`DateTime(System.DateTime newvalue)`	Initializes a new instance of the `DateTime` class to the `System.DateTime` object supplied as argument.
◆	`DateTime(int year, int month, int day, int hour, int minute, double second, int offsetTZ)`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, second, and timezone offset supplied as arguments.
◆	`DateTime(int year, int month, int day, int hour, int minute, double second)`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, and second supplied as arguments.
◆	`DateTime(int year, int month, int day)`	Initializes a new instance of the `DateTime` class to the year, month and day supplied as arguments.

Properties

	Name	Description
▤	`bool HasTimezone`	Gets a Boolean value which indicates if the `DateTime` has a timezone.
▤	`static DateTime Now`	Gets a `DateTime` object that is set to the current date and time on this computer.
▤	`short TimezoneOffset`	Gets or sets the timezone offset, in minutes, of the `DateTime` object.
▤	`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.

	Name	Description
		Note that this method is static and can only be called on the `Altova.Types.DateTime` class itself, not on an instance of the class.
◆	`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 947. 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 947.

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));
}
```

14.2.5.2 Altova.Types.DateTimeFormat

The `DateTimeFormat` enum type has the following constant values:

Value	Description	Example
S_DateTime	Formats the value as standard dateTime, with a precision of a ten-millionth of a second, including timezone.	`2015-11-12 12:19:03.9019132+01:00`
S_Days	Formats the value as number of days elapsed since the UNIX epoch.	`735913.6318973451087962962963`
S_Seconds	Formats the value as number of seconds elapsed since the UNIX epoch, with a precision of a ten-millionth of a second.	`63582937678.0769062`
W3_date	Formats the value as schema date.	`2015-11-12`
W3_dateTime	Formats the value as schema dateTime.	`2015-11-12T15:12:14.5194251`
W3_gDay	Formats the value as schema gDay.	`---12` (assuming that the date is 12th of the month)
W3_gMonth	Formats the value as schema gMonth.	`--11` (assuming that the month is November)
W3_gMonthDay	Formats the value as schema gMonthDay.	`--11-12` (assuming that the date is 12th of November)
W3_gYear	Formats the value as schema gYear.	`2015` (assuming that the year is 2015)
W3_gYearMonth	Formats the value as schema gYearMonth.	`2015-11` (assuming that the year is 2015 and the month is November)
W3_time	Formats the value as schema time, with a precision of a ten-millionth of a second.	`15:19:07.5582719`

14.2.5.3 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:

	Name	Description
		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);
}
```

14.2.5.4 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.

	Name	Description
	`XmlQualifiedName QualifiedName`	Returns the qualified name of the attribute.
	`bool Required()`	Returns true if the attribute is required.

14.2.5.5 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.

14.2.5.6 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.

14.2.5.7 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.

	Name	Description
	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.

14.2.5.8 [YourSchema].[Doc]

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the members listed below. Note that, in the method names below, "Doc" stands for the name of the generated document class itself.

Methods

	Name	Description
	static Doc CreateDocument()	Creates a new, empty XML document.
	static Doc CreateDocument(**string** encoding)	Creates a new, empty XML document, with encoding of type "encoding".
	static void DeclareAllNamespacesFromSchema(**Altova.Xml.ElementType** node)	Declares all namespaces from the XML Schema on the element supplied as argument (typically, the XML root element). Calling this method is useful if your schema has multiple namespace declarations, each mapped to a prefix, and you would like to declare all of them on the element supplied as argument.
	static Doc LoadFromBinary(**byte**[] binary)	Loads an XML document from a byte array.
	static Doc LoadFromFile(**string** filename)	Loads an XML document from a file.
	static Doc LoadFromString(**string** xmlstring)	Loads an XML document from a string.

	Name	Description
◆	**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** fileName, **bool** prettyPrint, **string** encoding, **string** lineend)	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding, and line ending character(s).
◆	**void** SaveToFile(**string** fileName, **bool** prettyPrint, **bool** omitXmlDecl, **string** encoding, **string** lineend)	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding, and line ending character(s). When `omitXmlDecl` is true, the XML declaration will not be written.
◆	**void** SaveToFile(**string** fileName, **bool** prettyPrint, **bool** omitXmlDecl, **string** encoding, **bool** bBigEndian, **bool** bBOM, **string** lineend)	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding, byte order, BOM (Byte Order Mark), and line ending character(s). When `omitXmlDecl` is true, the XML declaration will not be written.
◆	**void** SaveToFileWithLineEnd(**string** fileName, **bool** prettyPrint, **bool** omitXmlDecl, **string** lineend)	Saves an XML document to a file, with optional "pretty-print" formatting, and line ending character(s). When `omitXmlDecl` is true, the XML declaration will not be written.
◆	**string** SaveToString(**bool** prettyPrint)	Saves an XML document to a file, with optional "pretty-print" formatting.
◆	**string** SaveToString(**bool** prettyPrint, **bool** omitXmlDecl)	Saves an XML document to a file, with optional "pretty-print" formatting. When `omitXmlDecl` is true, the XML declaration will not be written.

	Name	Description
◆	**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.

14.2.5.9 [YourSchema].[ElementType]

This class provides methods for manipulating XML elements from your schema. Methods of this class can be called on elements, not on the XML document itself. Note that, in order to call methods of this class, you don't need to instantiate the class directly. Any element created using the **`Append()`** or **`AppendWithPrefix()`** methods is of `[ElementType]` type.

Methods

	Name	Description

◆	void DeclareNamespace(string prefix, string nsURI)	This method takes two arguments that are both of string type: the prefix and the namespace URI that you want to use. The prefix supplied as argument will be mapped to the namespace URI value supplied as argument. If the prefix supplied as argument is empty, the method creates or overrides the default namespace declaration in the element.
		For example, let's assume that the XML document has an XML element called "purchase". If you call
		`purchase.DeclareNamespace("ord", "http://OrderTypes");`
		then the XML document becomes
		`<purchase xmlns:ord="http://OrderTypes" />`
		Another example, if you call:
		`purchase.DeclareNamespace("", "http://OrderTypes");`
		then the XML document becomes
		`<purchase xmlns="http://OrderTypes" />`
		Note: The declared namespace is used when appending subsequent child elements or attributes, according to the

	Name	Description
		following rules: 1. If the child namespace is the default, then use empty prefix. 2. If the child namespace is equal to the parent one, then use the parent prefix. 3. Otherwise, search for nearest prefix from parent to top, using the lookup algorithm described in section "B.2: Namespace Prefix Lookup" at https://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20021022/namespaces-algorithms.html. 4. If there is no prefix for element namespace found, then use empty prefix.

14.2.5.10 [YourSchemaType].MemberAttribute

When code is generated from an XML schema, a class is created for each member attribute of a type. In the descriptions below, "AttributeType" stands for the type of the member attribute itself.

Methods

	Name	Description
◆	`bool Exists()`	Returns true if the attribute exists.
◆	`void Remove()`	Removes the attribute from its parent element.

Properties

	Name	Description
▤	**`int`** `EnumerationValue`	Generated for enumeration types only. Sets or gets the attribute value using one of the constants generated for the possible values. Returns **Invalid** if the value does not match any of the enumerated values in the schema.
▤	`Altova.Xml.Meta.Attribute Info`	Returns an object for querying schema information (see `Altova.Xml.Meta.Attribute` 951).
▤	`AttributeType Value`	Sets or gets the attribute value.

14.2.5.11 [YourSchemaType].MemberElement

When code is generated from an XML schema, a class with the following members is created for each member element of a type. The class implements the standard `System.Collections.IEnumerable` interface, so it can be used with the `foreach` statement.

In the descriptions below, "MemberType" stands for the type of the member element itself.

Methods

	Name	Description
◆	MemberType Append()	Creates a new element and appends it to its parent.
◆	MemberType AppendWithPrefix(string prefix)	Creates a new element having the prefix supplied as argument, and appends it to its parent. For an example, see Example: Purchase Order 922.
◆	MemberType At(**int** index)	Returns the member element specified by the index.
◆	System.Collections.IEnumerator GetEnumerator()	Returns an object for iterating instances of the member element.
◆	**void** Remove()	Deletes all occurrences of the element from its parent.
◆	**void** RemoveAt(**int** index)	Deletes the occurrence of the element specified by the index.

Properties

	Name	Description
▤	**int** Count	Returns the count of elements.
▤	**int** EnumerationValue	Generated for enumeration types only. Sets or gets the element value using one of the constants generated for the possible values. Returns **Invalid** if the value does not match any of the enumerated values in the schema.
▤	**bool** Exists	Returns true if at least one element exists.
▤	MemberType First	Returns the first instance of the member element.
▤	Altova.Xml.Meta.Element Info	Returns an object for querying schema information (see Altova.Xml.Meta.Element 953).
▤	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).

14.2.6 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 885). You can integrate these classes into your code to read, modify, and write XML documents.

Note: The generated code does include other supporting classes, which are not listed here and are subject to modification.

14.2.6.1 com.altova.types.DateTime

This class enables you to process XML attributes or elements that have date and time types, such as `xs:dateTime`.

Constructors

	Name	Description
● c	**public** DateTime()	Initializes a new instance of the `DateTime` class to an empty value.
● c	**public** DateTime(DateTime newvalue)	Initializes a new instance of the `DateTime` class to the `DateTime` value supplied as argument.
● c	**public** DateTime(**int** newyear, **int** newmonth, **int** newday, **int** newhour, **int** newminute, **int** newsecond, **double** newpartsecond, **int** newoffsetTZ)	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, second, the fractional part of the second, and timezone supplied as arguments. The fractional part of the second newpartsecond must be between 0 and 1. The timezone offset newoffsetTZ can be either positive or negative and is expressed in minutes.
● c	**public** DateTime(**int** newyear, **int** newmonth, **int** newday, **int** newhour, **int** newminute, **int** newsecond, **double** newpartsecond)	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, second, and the fractional part of a second supplied as arguments.
● c	**public** DateTime(**int** newyear, **int** newmonth, **int** newday)	Initializes a new instance of the `DateTime` class to the year, month, and day supplied as arguments.
● c	**public** DateTime(Calendar newvalue)	Initializes a new instance of the `DateTime` class to the `java.util.Calendar` value supplied as argument.

Methods

	Name	Description
● s	**static** DateTime now()	Returns the current time as a `DateTime` object.

	Name	Description

● 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: `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.

<table>
<tr><th></th><th>Name</th><th>Description</th></tr>
<tr><td>●</td><td><b>void</b> setHasTimezone(<b>int</b> nHasTZ)</td><td>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:<table>
<tr><td>CalendarBase.TZ_MISSING</td><td>Set the timezone offset to undefined.</td></tr>
<tr><td>CalendarBase.TZ_UTC</td><td>Set the timezone to UTC.</td></tr>
<tr><td>CalendarBase.TZ_OFFSET</td><td>If the current object has a timezone offset, leave it unchanged.</td></tr>
</table></td></tr>
<tr><td>●</td><td><b>void</b> setHour(<b>int</b> nHour)</td><td>Sets the hour of the current DateTime instance to the value supplied as argument.</td></tr>
<tr><td>●</td><td><b>void</b> setMinute(<b>int</b> nMinute)</td><td>Sets the minute of the current DateTime instance to the value supplied as argument.</td></tr>
<tr><td>●</td><td><b>void</b> setMonth(<b>int</b> nMonth)</td><td>Sets the month of the current DateTime instance to the value supplied as argument.</td></tr>
<tr><td>●</td><td><b>void</b> setPartSecond(<b>double</b> nPartSecond)</td><td>Sets the fractional part of the second of the current DateTime instance to the value supplied as argument.</td></tr>
<tr><td>●</td><td><b>void</b> setSecond(<b>int</b> nSecond)</td><td>Sets the second of the current DateTime instance to the value supplied as argument.</td></tr>
<tr><td>●</td><td><b>void</b> setTimezoneOffset(<b>int</b> nOffsetTZ)</td><td>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.</td></tr>
<tr><td>●</td><td><b>void</b> setYear(<b>int</b> nYear)</td><td>Sets the year of the current DateTime instance to the value supplied as argument.</td></tr>
<tr><td>●</td><td>String toString()</td><td>Returns the string representation of the current DateTime instance, for example:

2015-11-24T15:50:56.968+01:00</td></tr>
</table>

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());
}
```

14.2.6.2 com.altova.types.Duration

This class enables you to process XML attributes or elements of type `xs:duration`.

Constructors

	Name	Description
● c	`Duration(Duration newvalue)`	Initializes a new instance of the `Duration` class to the `Duration` object supplied as argument.
● c	`Duration(`**`int`**` newyear, `**`int`**` newmonth, `**`int`**` newday, `**`int`**` newhour, `**`int`**` newminute, `**`int`**` newsecond, `**`double`**` newpartsecond, `**`boolean`**` newisnegative)`	Initializes a new instance of the `Duration` class to a duration built from parts supplied as arguments.

Methods

	Name	Description
● s	**`static`**` Duration getFromDayTime( `**`int`**` newday,`	Returns a `Duration` object created from the number of days, hours, minutes, seconds, and fractional second parts supplied as

	Name	Description
	`int newhour, int newminute, int newsecond, double newpartsecond )`	argument.
● 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** May be used when the string **s** consists of any of the following: days, hours, minutes, seconds, fractional second parts, for example -P4DT4H4M4.774S. **ParseType.DURATION** May be used when the string **s** consists of any of the following: years, months, days, hours, minutes, seconds, fractional second parts, for example P1Y1M1DT1H1M1.333S. **ParseType.YEARMONTH** May be used when the string **s** consists of any of the following: years, months. For example: P3Y2M.
●	`int getDay()`	Returns the number of days in the current `Duration` instance.
●	`long getDayTimeValue()`	Returns the day and time value (in milliseconds) of the current `Duration` instance. Years and months are ignored.
●	`int getHour()`	Returns the number of hours in the current `Duration` instance.
●	`int getMillisecond()`	Returns the number of milliseconds in the current `Duration` instance.
●	`int getMinute()`	Returns the number of minutes in the current `Duration` instance.
●	`int getMonth()`	Returns the number of months in the current `Duration` instance.
●	`double getPartSecond()`	Returns the number of fractional second parts in the current `Duration` instance.

	Name	Description
●	**`int`** `getSecond()`	Returns the number of seconds in the current `Duration` instance.
●	**`int`** `getYear()`	Returns the number of years in the current `Duration` instance.
●	**`int`** `getYearMonthValue()`	Returns the year and month value (in months) of the current `Duration` instance. Days, hours, seconds, and milliseconds are ignored.
●	**`boolean`** `isNegative()`	Returns Boolean **true** if the current `Duration` instance is negative.
●	**`void`** `setDayTimeValue(`**`long`** `l)`	Sets the duration to the number of milliseconds supplied as argument, affecting only the day and time part of the duration.
●	**`void`** `setNegative(` **`boolean`** `isnegative )`	Converts the current `Duration` instance to a negative duration.
●	**`void`** `setYearMonthValue(`**`int`** `l)`	Sets the duration to the number of months supplied as argument. Only the years and months part of the duration is affected.
●	`String toString()`	Returns the string representation of the current `Duration` instance, for example: `-P4DT4H4M4.774S`
●	`String toYearMonthString()`	Returns the string representation of the YearMonth part of the current `Duration` instance, for example: `P1Y2M`

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
import com.altova.types.*;
import com.altova.types.Duration.ParseType;
```

The following code listing illustrates various ways to create `Duration` objects:

```
protected static void ExampleDuration()
{
   // Create a negative duration of 1 year, 1 month, 1 day, 1 hour, 1 minute, 1 second,
   // and 0.333 fractional second parts
   Duration dr = new Duration(1, 1, 1, 1, 1, 1, .333, true);

   // Create a duration from an existing Duration object
   Duration dr1 = new Duration(dr);

   // Create a duration of 4 days, 4 hours, 4 minutes, 4 seconds, .774 fractional second
parts
   Duration dr2 = Duration.getFromDayTime(4, 4, 4, 4, .774);

   // Create a duration of 3 years and 2 months
   Duration dr3 = Duration.getFromYearMonth(3, 2);

   // Create a duration from a string
   Duration dr4 = Duration.parse("-P4DT4H4M4.774S");

   // Create a duration from a string, using specific parse formats
   Duration dr5 = Duration.parse("-P1Y1M1DT1H1M1.333S", ParseType.DURATION);
   Duration dr6 = Duration.parse("P3Y2M", ParseType.YEARMONTH);
   Duration dr7 = Duration.parse("-P4DT4H4M4.774S", ParseType.DAYTIME);
}
```

The following code listing illustrates getting and setting the value of `Duration` objects:

```
protected static void DurationExample2()
{
   // Create a duration of 1 year, 2 month, 3 days, 4 hours, 5 minutes, 6 seconds,
       // and 333 milliseconds
   Duration dr = new Duration(1, 2, 3, 4, 5, 6, .333, false);
   // Output the number of days in this duration
   System.out.println(dr.getDay());

   // Create a positive duration of one year and 333 milliseconds
   Duration dr1 = new Duration(1, 0, 0, 0, 0, 0, .333, false);
   // Output the day and time value in milliseconds
   System.out.println(dr1.getDayTimeValue());

   // Create a positive duration of 1 year, 1 month, 1 day, 1 hour, 1 minute, 1 second,
       // and 333 milliseconds
   Duration dr2 = new Duration(1, 1, 1, 1, 1, 1, .333, false);
   // Output the year and month value in months
   System.out.println(dr2.getYearMonthValue());

   // Create a positive duration of 1 year and 1 month
   Duration dr3 = new Duration(1, 1, 0, 0, 0, 0, 0, false);
   // Output the value
   System.out.println("The duration is now: " + dr3.toString());
   // Set the DayTime part of duration to 1000 milliseconds
   dr3.setDayTimeValue(1000);
   // Output the value
   System.out.println("The duration is now: " + dr3.toString());
   // Set the YearMonth part of duration to 1 month
   dr3.setYearMonthValue(1);
   // Output the value
   System.out.println("The duration is now: " + dr3.toString());
   // Output the year and month part of the duration
   System.out.println("The YearMonth part of the duration is: " +
dr3.toYearMonthString());
}
```

14.2.6.3 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.

14.2.6.4 com.altova.xml.meta.ComplexType

This class enables you to access schema information about classes generated from complex types. Note that this class is not meant to provide dynamic information about particular instances of a complex type in an XML document. Instead, it enables you to obtain programmatically information about a particular complex type defined in the XML schema.

Methods

	Name	Description
●	`Attribute findAttribute(String localName, String namespaceURI)`	Finds the attribute with the specified local name and namespace URI.
●	`Element findElement(String localName, String namespaceURI)`	Finds the element with the specified local name and namespace URI.
●	`Attribute[] GetAttributes()`	Returns a list of all attributes.
●	`ComplexType getBaseType()`	Returns the base type of this type.
●	`SimpleType getContentType()`	Returns the simple type of the content.
●	`Element[] GetElements()`	Returns a list of all elements.
●	`String getLocalName()`	Returns the local name of the type.
●	`String getNamespaceURI()`	Returns the namespace URI of the type.

14.2.6.5 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.

14.2.6.6 com.altova.xml.meta.SimpleType

This class enables you to access schema information about classes generated from simple types. Note that this class is not meant to provide dynamic information about particular instances of simple types in an XML document. Instead, it enables you to obtain programmatically information about a particular simple type defined in the XML schema.

Methods

	Name	Description
●	`SimpleType getBaseType()`	Returns the base type of this type.
●	`String[] getEnumerations()`	Returns an array of all enumeration facets.
●	`int getFractionDigits()`	Returns the value of this facet.
●	`int getLength()`	Returns the value of this facet.
●	`String getLocalName()`	Returns the local name of the type.
●	`String getMaxExclusive()`	Returns the value of this facet.
●	`String getMaxInclusive()`	Returns the value of this facet.
●	`int getMaxLength()`	Returns the value of this facet.

	Name	Description
●	String getMinExclusive()	Returns the value of this facet.
●	String getMinInclusive()	Returns the value of this facet.
●	**int** getMinLength()	Returns the value of this facet.
●	String getNamespaceURI()	Returns the namespace URI of the type.
●	String[] getPatterns()	Returns an array of all pattern facets.
●	**int** getTotalDigits()	Returns the value of this facet.
●	**int** getWhitespace()	Returns the value of the whitespace facet, which is one of: com.altova.typeinfo.WhitespaceType.Whitespace_Unknown com.altova.typeinfo.WhitespaceType.Whitespace_Preserve com.altova.typeinfo.WhitespaceType.Whitespace_Replace com.altova.typeinfo.WhitespaceType.Whitespace_Collapse

14.2.6.7 com.[YourSchema].[Doc]

When code is generated from an XML Schema, the generated code provides a document class with the same name as the schema. This class contains all possible root elements as members, as well as the members listed below. Note that, in the method names below, "Doc" stands for the name of the generated document class itself.

Methods

	Name	Description
● S	static Doc createDocument()	Creates a new, empty XML document.
● S	static void declareAllNamespacesFromSchema(com.altova.xml.ElementType node)	Declares all namespaces from the XML Schema on the element supplied as argument (typically, the XML root element). Calling this method is useful if your schema has multiple namespace declarations, each mapped to a prefix, and you would like to declare all of them on the element supplied as argument.
● S	static Doc loadFromBinary(byte[] xml)	Loads an XML document from a byte array.
● S	static Doc loadFromFile(String fileName)	Loads an XML document from a file.
● S	static Doc loadFromString(String xml)	Loads an XML document from a string.
●	byte[] saveToBinary(boolean prettyPrint)	Saves an XML document to a byte array, with optional "pretty-print" formatting.

	Name	Description
●	`byte[] saveToBinary(boolean prettyPrint, String encoding)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding.
●	`byte[] saveToBinary(boolean prettyPrint, String encoding, boolean bigEndian, boolean writeBOM)`	Saves an XML document to a byte array, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
●	`void saveToFile(String fileName, boolean prettyPrint)`	Saves an XML document to a file, with optional "pretty-print" formatting.
●	`void saveToFile(String fileName, boolean prettyPrint, boolean omitXmlDecl)`	Saves an XML document to a file, with optional "pretty-print" formatting, with UTF-8 encoding. When `omitXmlDecl` is true, the XML declaration will not be written.
●	`void saveToFile(String fileName, boolean prettyPrint, boolean omitXmlDecl, String encoding)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. When `omitXmlDecl` is true, the XML declaration will not be written.
●	`void saveToFile(String fileName, boolean prettyPrint, boolean omitXmlDecl, String encoding, boolean bBigEndian, boolean bBOM)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. When `omitXmlDecl` is true, the XML declaration will not be written. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
●	`void saveToFile(String fileName, boolean prettyPrint, String encoding)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding.
●	`void saveToFile(String fileName, boolean prettyPrint, String encoding, boolean bBigEndian, boolean bBOM)`	Saves an XML document to a file, with optional "pretty-print" formatting, with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
●	`String saveToString(boolean prettyPrint)`	Saves an XML document to a string, with optional "pretty-print" formatting.
●	`String saveToString(boolean prettyPrint, boolean omitXmlDecl)`	Saves an XML document to a string, with optional "pretty-print" formatting. When `omitXmlDecl` is true, the XML declaration will not be written.
●	`void setSchemaLocation(String schemaLocation)`	Adds an `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` attribute to the root element. A root element must already exist.

14.2.6.8 com.[YourSchema].[ElementType]

This class provides methods for manipulating XML elements from your schema. Methods of this class can be called on elements, not on the XML document itself. Note that, in order to call methods of this class, you don't need to instantiate the class directly. Any element created using the **`append()`** or **`appendWithPrefix()`** methods is of `[ElementType]` type.

Methods

	Name	Description

●	`void declareNamespace(String prefix, String nsURI)`	This method takes two arguments that are both of string type: the prefix and the namespace URI that you want to use. The prefix supplied as argument will be mapped to the namespace URI value supplied as argument. If the prefix supplied as argument is empty, the method creates or overrides the default namespace declaration in the element.
		For example, let's assume that the XML document has an XML element called "purchase". If you call
		`purchase.declareNamespace("ord", "http://OrderTypes");`
		then the XML document becomes
		`<purchase xmlns:ord="http://OrderTypes" />`
		Another example, if you call:
		`purchase.declareNamespace("", "http://OrderTypes");`
		then the XML document becomes
		`<purchase xmlns="http://OrderTypes" />`
		Note: The declared namespace is used when appending subsequent child elements or attributes, according to the following rules:
		1. If the child namespace is the default, then use empty prefix.
		2. If the child namespace is equal to the parent one, then use the parent prefix.
		3. Otherwise, search for nearest prefix from parent to

	Name	Description
		top, using the lookup algorithm described in section "B.2: Namespace Prefix Lookup" at https://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20021022/namespaces-algorithms.html. 4. If there is no prefix for element namespace found, then use empty prefix.

14.2.6.9 com.[YourSchema].[YourSchemaType].MemberAttribute

When code is generated from an XML schema, a class is created for each member attribute of a type. In the descriptions below, "AttributeType" stands for the type of the member attribute itself.

Methods

	Name	Description
●	**boolean** exists()	Returns true if the attribute exists.
●	**int** getEnumerationValue()	Generated for enumeration types only. Returns one of the constants generated for the possible values, or **Invalid** if the value does not match any of the enumerated values in the schema.
●	com.altova.xml.meta.Attribute getInfo()	Returns an object for querying schema information (see com.altova.xml.meta.Attribute[967]).
●	AttributeType getValue()	Gets the attribute value.
●	**void** remove()	Removes the attribute from its parent element.
●	**void** setEnumerationValue(**int**)	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.
●	**void** setValue(AttributeType value)	Sets the attribute value.

14.2.6.10 com.[YourSchema].[YourSchemaType].MemberElement

When code is generated from an XML schema, a class with the following members is created for each member element of a type. In the descriptions below, "MemberType" stands for the type of the member element itself.

Methods

	Name	Description
●	MemberType append()	Creates a new element and appends it to its parent.

	Name	Description
●	MemberType appendWithPrefix(String prefix)	Creates a new element having the prefix supplied as argument, and appends it to its parent. For an example, see Example: Purchase Order 922.
●	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 968).
●	MemberType getValue()	Gets the element content (only generated if element can have simple or mixed content).
●	java.util.Iterator iterator()	Returns an object for iterating instances of the member element.
●	MemberType last()	Returns the last instance of the member element.
●	**void** remove()	Deletes all occurrences of the element from its parent.
●	**void** removeAt(**int** index)	Deletes the occurrence of the element specified by the index.
●	**void** setEnumerationValue(**int** index)	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.
●	**void** setValue(MemberType value)	Sets the element content (only generated if element can have simple or mixed content).

14.2.7 SPL Reference

This section gives you an overview of SPL (Spy Programming Language), 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 applications's **spl** folder. You can use these files as a guide to developing your own templates.

How code generator works

Code is generated on the basis of the template files (**.spl** files) and the object model provided by MapForce. The template files contain the code of the target programming language combined with SPL instructions for creating files, reading information from the object model, and performing calculations.

The template file is interpreted by the code generator and outputs the source-code files of the target language/s (that is, the non-compiled code files) and any other relevant project file or template-dependent file. The source code can then be compiled into an executable file that accesses the XML data described by the schema file.

SPL files have access to a wide variety of information that is collated from the source schemas. Note that an SPL file is not tied to a specific schema, but allows access to all schemas. So, make sure you write your SPL files generically and avoid structures which apply to specific schemas.

Note about method names
When you customize code generation using the supplied SPL files, it might be necessary to reserve names to avoid collisions with other symbols. Follow the instructions below:

1. Navigate to the program installation directory, for example, `C:\Program Files\Altova\MapForce2025`.
2. In the **spl** subdirectory, locate the directory corresponding to the programming language, for example, **..\spl\java**.
3. Open the **settings.spl** file and insert a new line into the `reserve` section, for example, **reserve "myReservedWord"**.
4. Regenerate the program code.

Example: Creating a new file in SPL

This is a very basic SPL file. It creates a file named **test.cpp**, and places the include statement within it. The close command completes the template.

```
[create "test.cpp"]
#include "stdafx.h"
[close]
```

14.2.7.1 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 979 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 **]**.

14.2.7.2 Declarations

The following statements are evaluated while parsing the SPL template file. They are **not** affected by flow control statements like conditions, loops or subroutines, and are always evaluated exactly once.

These keywords, like all keywords in SPL, are not case sensitive.

Remember that all of these declarations must be inside a block delimited by square brackets.

map ... to ...

```
map mapname key to value [, key to value ]...
```

This statement adds information to a map. See below for specific uses.

```
map schemanativetype schematype to typespec
```

The specified built-in XML Schema type will be mapped to the specified native type or class, using the specified formatter. This setting applies only to code generation for version 2007r3 and higher. Typespec is a native type or class name, followed by a comma, followed by the formatter class instance.

Example:

```
map schemanativetype "double" to "double,Altova::DoubleFormatter"
```

map type ... to ...

```
map type schematype to classname
```

The specified built-in XML Schema type will be mapped to the specified class. This setting applies only to code generation for version 2007 or lower.

Example:

```
map type "float" to "CSchemaFloat"
```

default ... is ...

```
default setting is value
```

This statement allows you to affect how class and member names are derived from the XML Schema. Note that the setting names are case sensitive.

Example:

```
default "InvalidCharReplacement" is "_"
```

Setting name	**Explanation**
ValidFirstCharSet	Allowed characters for starting an identifier
ValidCharSet	Allowed characters for other characters in an identifier
InvalidCharReplacement	The character that will replace all characters in names that are not in the ValidCharSet
AnonTypePrefix	Prefix for names of anonymous types*
AnonTypeSuffix	Suffix for names of anonymous types*
ClassNamePrefix	Prefix for generated class names
ClassNameSuffix	Suffix for generated class names
EnumerationPrefix	Prefix for symbolic constants declared for enumeration values
EnumerationUpperCase	"on" to convert the enumeration constant names to upper case
FallbackName	If a name consists only of characters that are not in ValidCharSet, use this one

* Names of anonymous types are built from AnonTypePrefix + element name + AnonTypeSuffix

reserve

```
reserve word
```

Adds the specified word to the list of reserved words. This ensures that it will never be generated as a class or member name.

Example:

```
reserve "while"
```

include

includes the specified file as SPL source. This allows you to split your template into multiple files for easier editing and handling.

```
include filename
```

Example:

```
include "Module.cpp"
```

14.2.7.3 Variables

Any non-trivial SPL file will require variables. Some variables are predefined[978] 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[983] 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.

14.2.7.4 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 987	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

Name	Type	Description
$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 **2010** Visual Studio 2010 **2013** Visual Studio 2013 **2015** Visual Studio 2015 **2017** Visual Studio 2017 **2019** Visual Studio 2019

For C# generation only:

Name	Type	Description
$VSVersion	integer	Specifies the Visual Studio version. Valid values: **0** No Visual Studio project **2010** Visual Studio 2010 **2013** Visual Studio 2013 **2015** Visual Studio 2015 **2017** Visual Studio 2017 **2019** Visual Studio 2019

14.2.7.5 Creating output files

These statements are used to create output files from the code generation. Remember that all of these statements must be inside a block delimited by square brackets.

create

```
create filename
```

creates a new file. The file has to be closed with the **close** statement. All following output is written to the specified file.

Example:

```
[create $outputpath & "/" & $JavaPackageDir & "/" & $application.Name & ".java"]
package [=$JavaPackageName];

public class [=$application.Name]Application {
...
}
[close]
```

close

closes the current output file.

```
=$variable
```

writes the value of the specified variable to the current output file.

Example:

```
[$x = 20+3]
The result of your calculation is [=$x] - so have a nice day!
```

The file output will be:

```
The result of your calculation is 23 - so have a nice day!
```

write

```
write string
```

writes the string to the current output file.

Example:

```
[write "C" & $name]
```

This can also be written as:

```
C[=$name]
```

filecopy ... to ...

```
filecopy source to target
```

copies the source file to the target file, without any interpretation.

Example:

```
filecopy "java/mapforce/mapforce.png" to $outputpath & "/" & $JavaPackageDir &
"/mapforce.png"
```

14.2.7.6 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

14.2.7.7 Conditions

SPL allows you to use standard "if" statements. The syntax is as follows:

```
if condition
   statements
else
   statements
endif
```

or, without else:

```
if condition
   statements
endif
```

Note: There are no round brackets enclosing the condition.

As in any other programming language, conditions are constructed with logical and comparison operators 981.

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.

14.2.7.8 Collections and foreach

Collections and iterators

A collection contains multiple objects - like a ordinary array. Iterators solve the problem of storing and incrementing array indexes when accessing objects.

Syntax:

```
foreach iterator in collection
   statements
next
```

Example:

```
[foreach $class in $classes
   if not $class.IsInternal
      ]  class [=$class.Name];
[  endif
next]
```

Example 2:

```
[foreach $i in 1 To 3
      Write "// Step " & $i & "\n"
      ‘ Do some work
next]
```

In the first line:

$classes is the **global object**[978] 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]
```

14.2.7.9 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)

14.2.7.9.1 Subroutine declaration

Subroutines

Syntax example:

```
Sub SimpleSub()
       ... lines of code
 EndSub
```

- **Sub** is the keyword that denotes the procedure.
- **SimpleSub** is the name assigned to the subroutine.
- Round **parenthesis** can contain a parameter list.
- The code block of a subroutine starts immediately after the closing parameter parenthesis.
- **EndSub** denotes the end of the code block.

Note: Recursive or cascaded subroutine **declaration** is not permitted, i.e. a subroutine may not contain another subroutine.

Parameters

Parameters can also be passed by procedures using the following syntax:

- All parameters must be variables
- Variables must be prefixed by the **$** character
- Local variables are defined in a subroutine
- Global variables are declared explicitly, outside of subroutines
- Multiple parameters are separated by the comma character "**,**" within round parentheses
- Parameters can pass values

Parameters - passing values

Parameters can be passed in two ways, by value and by reference, using the keywords **ByVal** and **ByRef** respectively.

Syntax:

```
' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
] ...
```

- **ByVal** specifies that the parameter is passed by value. Note that most objects can only be passed by reference.
- **ByRef** specifies that the parameter is passed by reference. This is the default if neither ByVal nor ByRef is specified.

Function return values

To return a value from a subroutine, use the **return** statement. Such a function can be called from within an expression.

Example:

```
' define a function
[Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )
if $namespacePrefix = ""
  return $localName
else
  return $namespacePrefix & ":" & $localName
endif
EndSub
]
```

14.2.7.9.2 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")
```

14.2.7.9.3 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
]
```

14.2.7.10 Built in Types

The section describes the properties of the built-in types used in the predefined variables[978] which describe the parsed schema.

14.2.7.10.1 Library

This object represents the whole library generated from the XML Schema or DTD.

Property	Type	Description
SchemaNamespaces	Namespace[987] 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)

14.2.7.10.2 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[987] collection	All types contained in this namespace
Library	Library[987]	Library containing this namespace

14.2.7.10.3 Type

This object represents a complex or simple type. It is used to generate a class in the target language. There is one additional type per library that represents the document, which has all possible root elements as members.

Anonymous types have an empty LocalName.

Property	Type	Description
CodeName	string	Name for generated code (derived from local name or parent declaration)
LocalName	string	Original name in the schema
Namespace	Namespace[987]	Namespace containing this type
Attributes	Member[989] collection	Attributes contained in this type*
Elements	Member[989] 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[987]	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[990]	Native binding for this type
Facets	Facets[990]	Facets of this type

Property	Type	Description
Whitespace	string	Shortcut to the Whitespace facet

* Complex types with text content (these are types with mixed content and complexType with simpleContent) have an additional unnamed attribute member that represents the text content.

14.2.7.10.4 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 987	Type originally declaring the member (equal to ContainingType for non-inherited members)
ContainingType	Type 987	Type where this is a member of
DataType	Type 987	Data type of this member's content
Library	Library 987	Library containing this member's DataType
IsAttribute	boolean	True for attributes, false for elements
IsOptional	boolean	True if minOccurs = 0 or optional attribute
IsRequired	boolean	True if minOccurs > 0 or required attribute
IsFixed	boolean	True for fixed attributes, value is in Default property
IsDefault	boolean	True for attributes with default value, value is in Default property
IsNillable	boolean	True for nillable elements
IsUseQualified	boolean	True if NamespaceURI is not empty
MinOccurs	integer	minOccurs, as in schema. 1 for required attributes
MaxOccurs	integer	maxOccurs, as in schema. 0 for prohibited attributes, -1 for unbounded

Property	Type	Description
Default	string	Default value

14.2.7.10.5 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

14.2.7.10.6 Facets

This object represents all facets of a simple type. Inherited facets are merged with the explicitly declared facets. If a Length facet is in effect, MinLength and MaxLength are set to the same value.

Property	Type	Description
DeclaringType	Type	Type facets are declared on
Whitespace	string	"preserve", "collapse" or "replace"
MinLength	integer	Facet value
MaxLength	integer	Facet value
MinInclusive	integer	Facet value
MinExclusive	integer	Facet value
MaxInclusive	integer	Facet value
MaxExclusive	integer	Facet value
TotalDigits	integer	Facet value
FractionDigits	integer	Facet value
List	Facet collection	All facets as list

Facet

This object represents a single facet with its computed value effective for a specific type.

Property	Type	Description
LocalName	string	Facet name
NamespaceURI	string	Facet namespace
FacetType	string	one of "normalization", "lexicalspace", "valuespace-length", "valuespace-enum" or "valuespace-range"
DeclaringType	Type 987	Type this facet is declared on
FacetCheckerName	string	Name of facet checker (from schemafacet map)
FacetValue	string or integer	Actual value of this facet

15 MapForce COM API

The COM-based API of MapForce enables other applications to use the functionality of MapForce. This makes it possible to automate a wide range of tasks, for example, defining various options, generating output, and many others.

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 such as JScript and VBScript.

MapForce API documentation

The MapForce API documentation can be accessed here: https://www.altova.com/manual/en/api/mapforceapi/index.html.

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#, and VB.NET. 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[994]).

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[998] and Integration at Document Level[1001] describe the key steps at these respective levels. The ActiveX Integration Examples[1004] section provides examples in C# and Java. Looking through these examples will help you to make the right decisions quickly. The Object Reference[1034] 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[844].

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 2012/2013/2015/2017/2019/2022
- MapForce 32-bit and MapForce Integration Package 32-bit
- The ActiveX controls must be added to the Visual Studio toolbox (see Adding the ActiveX Controls to the Toolbox 996).

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 1004.

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.

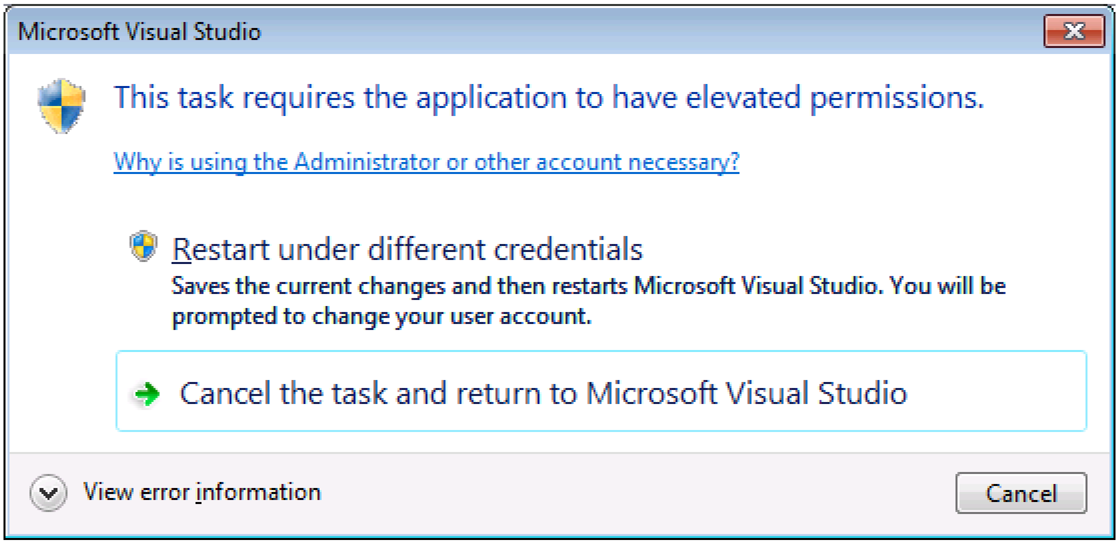
16.2 Adding the ActiveX Controls to the Toolbox

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 994.
2. If prompted to restart Visual Studio with elevated permissions, click **Restart under different credentials**.


Microsoft Visual Studio

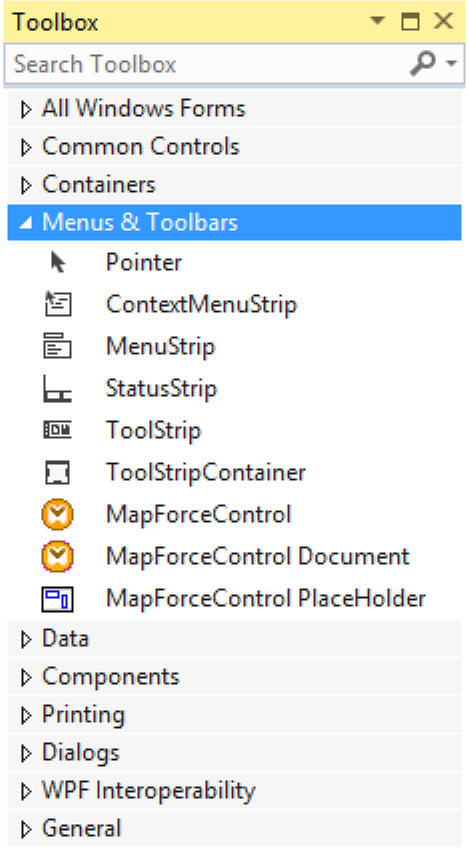
This task requires the application to have elevated permissions.

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 998). The **MapForceControl Document** and **MapForceControl Placeholder** controls are used for document-level integration (see Integration at Document Level 1001).

16.3 Integration at Application Level

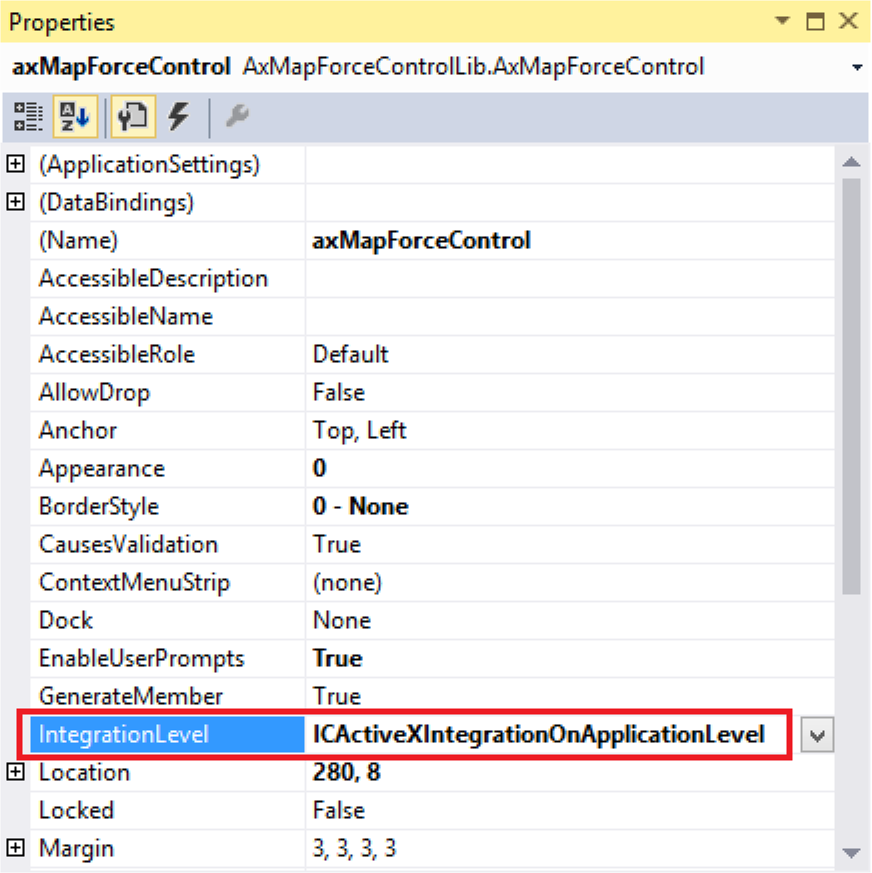
Integration at application level allows you to embed the complete interface of MapForce into a window of your application. With this type of integration, you get the whole user interface of MapForce, including all menus, toolbars, the status bar, document windows, and helper windows. Customization of 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`[1037]. Do not instantiate or access `MapForceControlDocument`[1045] or `MapForceControlPlaceHolder`[1051] 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`[1037]. Consider using `MapForceControl.Application`[1038] for more complex access to MapForce functionality.

In C# or VB.NET with Visual Studio, the steps to create a basic, one-form application which integrates the MapForce ActiveX controls at application level are as follows:

1. Check that all prerequisites are met (see Prerequisites[994]).
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[996]).
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**.



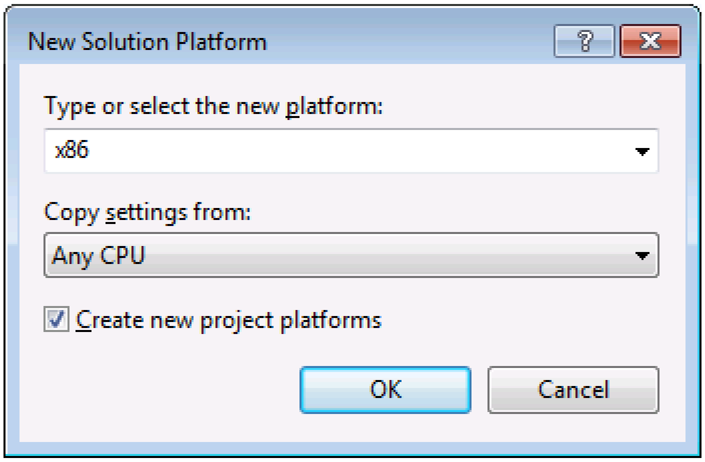
Properties

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 998, for an ActiveX integration at application level, only one control is required, namely the **MapForceControl**. However, for an ActiveX integration at document level, MapForce functionality is provided by the following ActiveX controls:

- MapForceControl 1037
- MapForceControl Document 1045
- MapForceControl Placeholder 1051

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 996).

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 1039 property to ICActiveXIntegrationOnDocumentLevel (or "1"). To hide the control from the user, set its Visible property to False. Note that, 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.

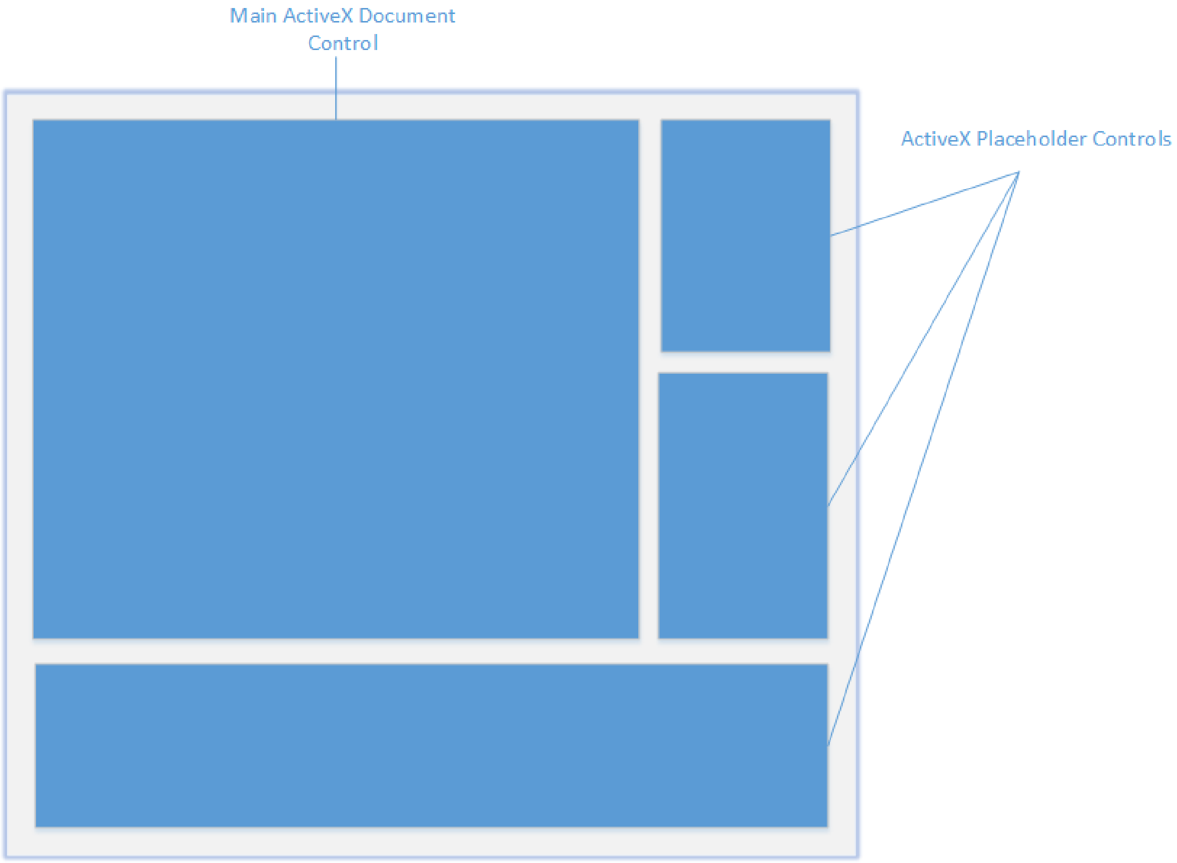
2. Create at least one instance of MapForceControl Document in your application. This control supplies the document editing window of MapForce to your application and can be instantiated multiple times if necessary. Use the method Open to load any existing file. To access document-related functionality, use the Path and Save or methods and properties accessible via the property Document. Note that 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 1054. 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 996).
3. Drag the MapForceControl 1037 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 1045 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 1051 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 1054.
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 1039 property of the MapForceControl.
- To retrieve commands organized according to their menu structure, use the MainMenu 1040 property.
- To retrieve commands organized by the toolbar in which they appear, use the Toolbars 1040 property.
- To send commands to MapForce, use the Exec 1041 method.
- To query if a command is currently enabled or disabled, use the QueryStatus 1042 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 1024 section.

General considerations

To automate the behaviour of MapForce, use the properties, methods, and events described for the MapForceControl 1037, MapForceControl Document 1045, and MapForceControl Placeholder 1051.

For more complex access to MapForce functionality, consider using the following properties:

- `MapForceControl.Application` 1038
- `MapForceControlDocument.Document` 1046
- `MapForceControlPlaceHolder.Project` 1052

These properties give you access to the MapForce automation interface (MapForceAPI)

Note: To open a document, always use MapForceControlDocument.Open 1048 or MapForceControlDocument.New 1048 on the appropriate document control. To open a project, always use MapForceControlPlaceHolder.OpenProject 1053 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 1004.

16.5 ActiveX Integration Examples

This section contains examples of MapForce document-level integration using different container environments and programming languages. Source code for all examples is available in the folder **`<ApplicationFolder>\Examples\ActiveX`** of your MapForce installation.

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 1004).

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 994.

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

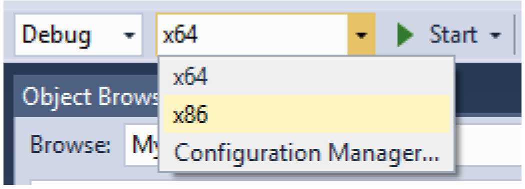
The sample application contains one instance of MapForceControlDocument 1045 and several instances of MapForceControlPlaceHolder 1051 controls. Double-check that the following properties of these controls are set as shown in the table below:

Control name	Property	Property value
axMapForceControl	IntegrationLevel	ICActiveXIntegrationOnDocumentLevel
axMapForceControlLibrary	PlaceholderWindowID	0
axMapForceControlOutput	PlaceholderWindowID	2
axMapForceControlPreview	PlaceholderWindowID	1

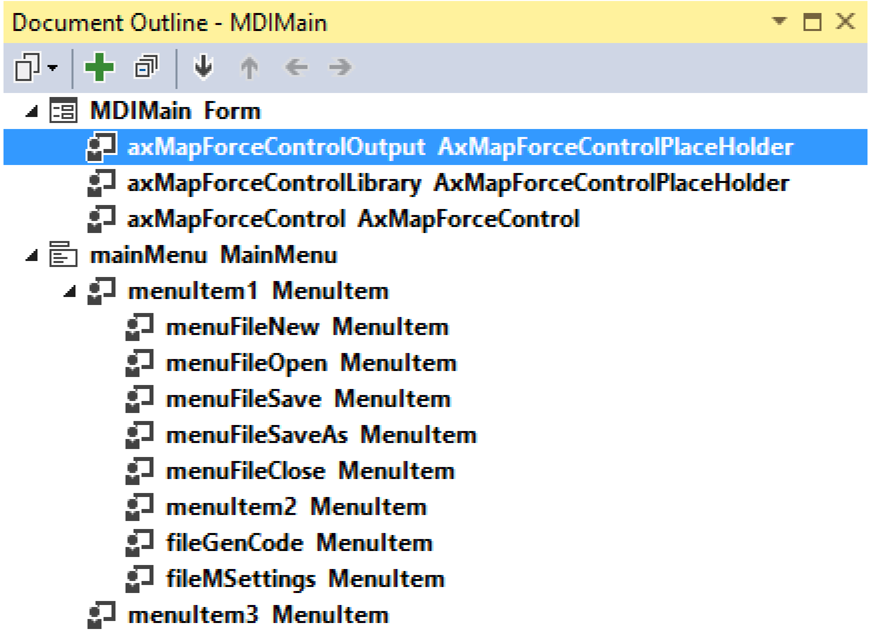
Here is how you can view or set the properties of an ActiveX control:

1. Open the **MDIMain.cs** form in the designer window.

Note: On 64-bit Windows, it may be necessary to change the build configuration of the Visual Studio solution to "x86" **before** opening the designer window. If you need to build the sample as a 64-bit application, see Prerequisites 994.

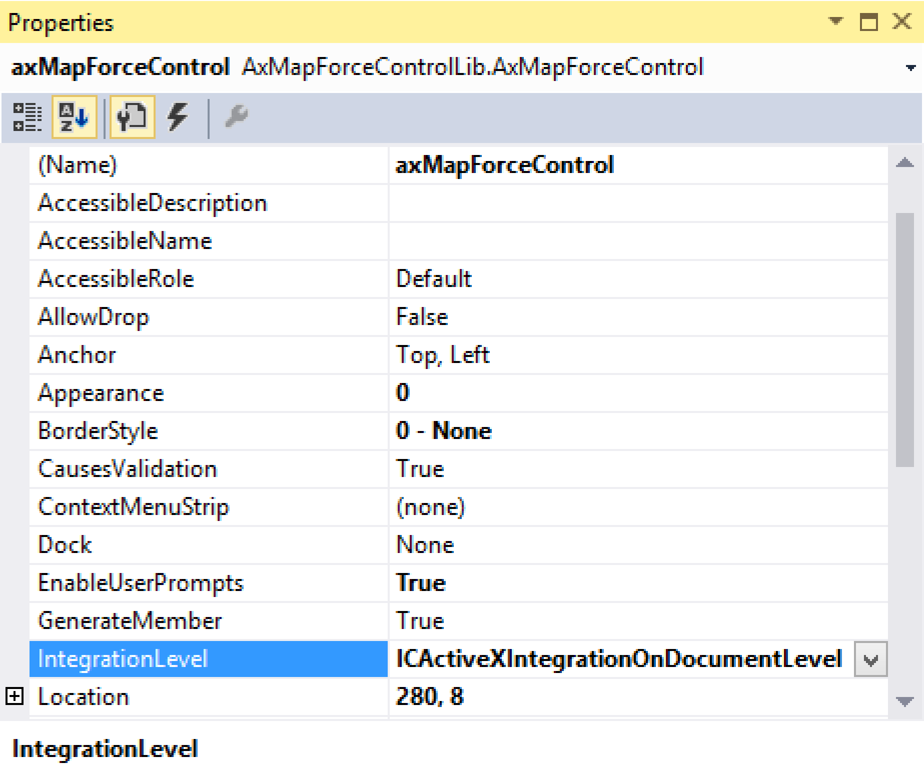


2. Open the **Document Outline** window of Visual Studio (On the **View** menu, click **Other Windows | Document Outline**).


Document Outline - MDIMain

- MDIMain Form
  - axMapForceControlOutput AxMapForceControlPlaceHolder
  - axMapForceControlLibrary AxMapForceControlPlaceHolder
  - axMapForceControl AxMapForceControl
- mainMenu MainMenu
  - menuItem1 MenuItem
    - menuFileNew MenuItem
    - menuFileOpen MenuItem
    - menuFileSave MenuItem
    - menuFileSaveAs MenuItem
    - menuFileClose MenuItem
    - menuItem2 MenuItem
    - fileGenCode MenuItem
    - fileMSettings MenuItem
  - menuItem3 MenuItem

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

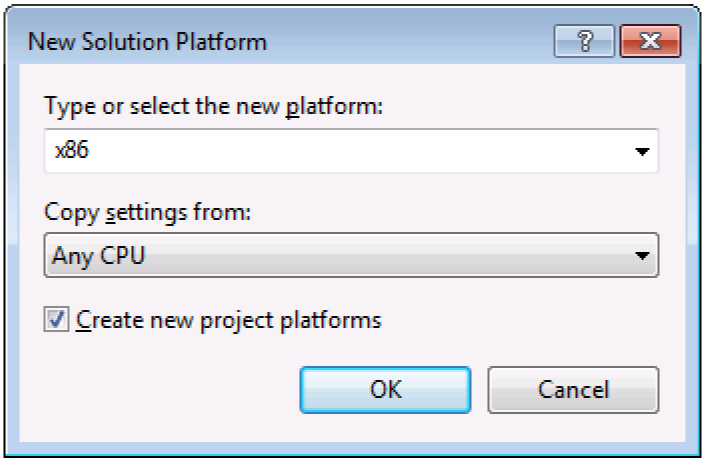
(Name)	**axMapForceControl**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
Appearance	**0**
BorderStyle	**0 - None**
CausesValidation	True
ContextMenuStrip	(none)
Dock	None
EnableUserPrompts	**True**
GenerateMember	True
IntegrationLevel	**ICActiveXIntegrationOnDocumentLevel**
⊞ 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

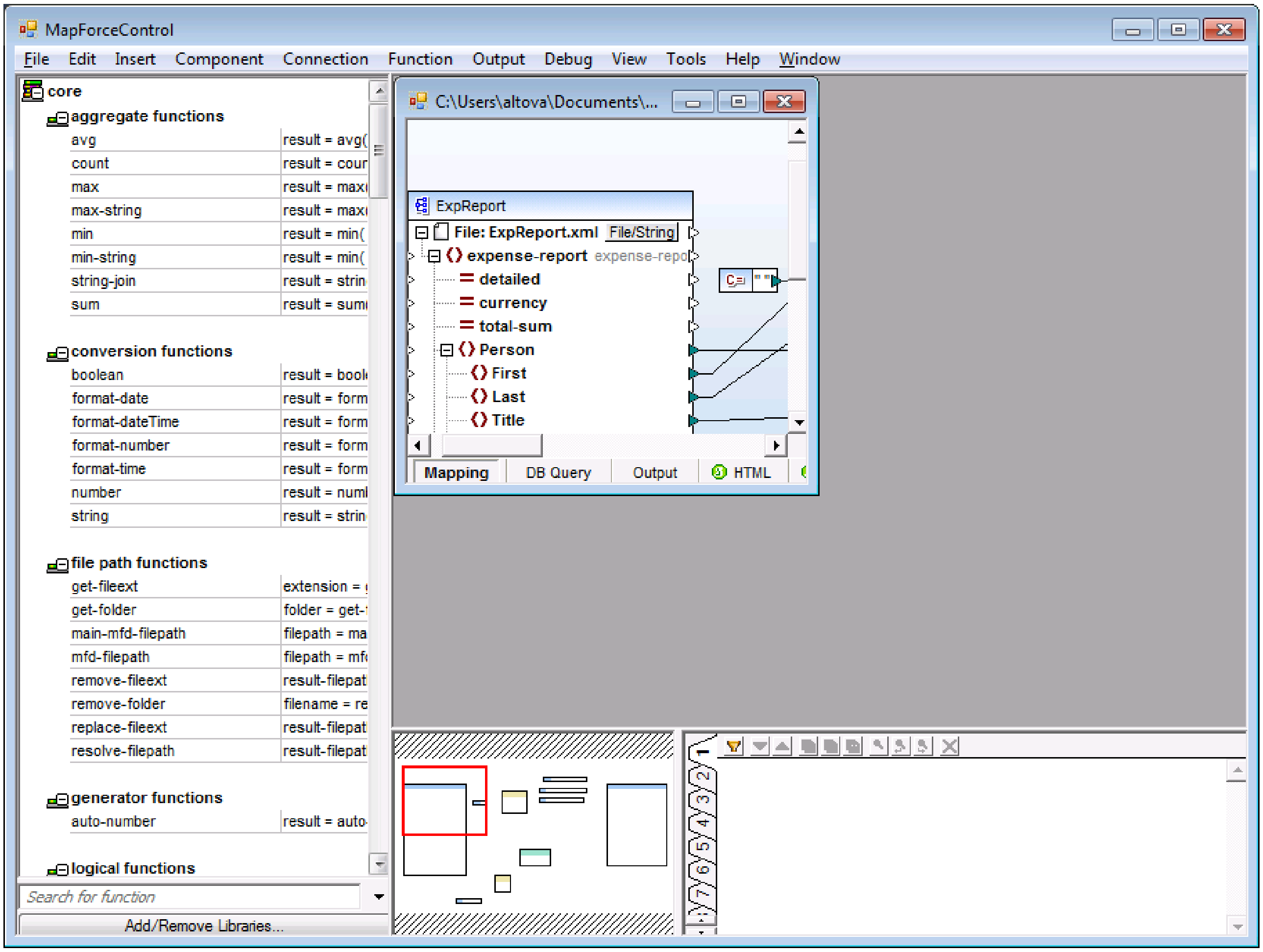
Type or select the new platform: x86

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:



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.

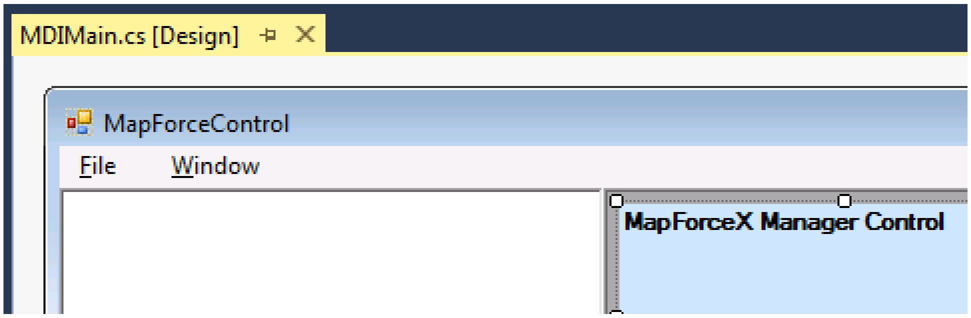
16.5.1.2 Retrieving Command Information

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);
  }
}
```

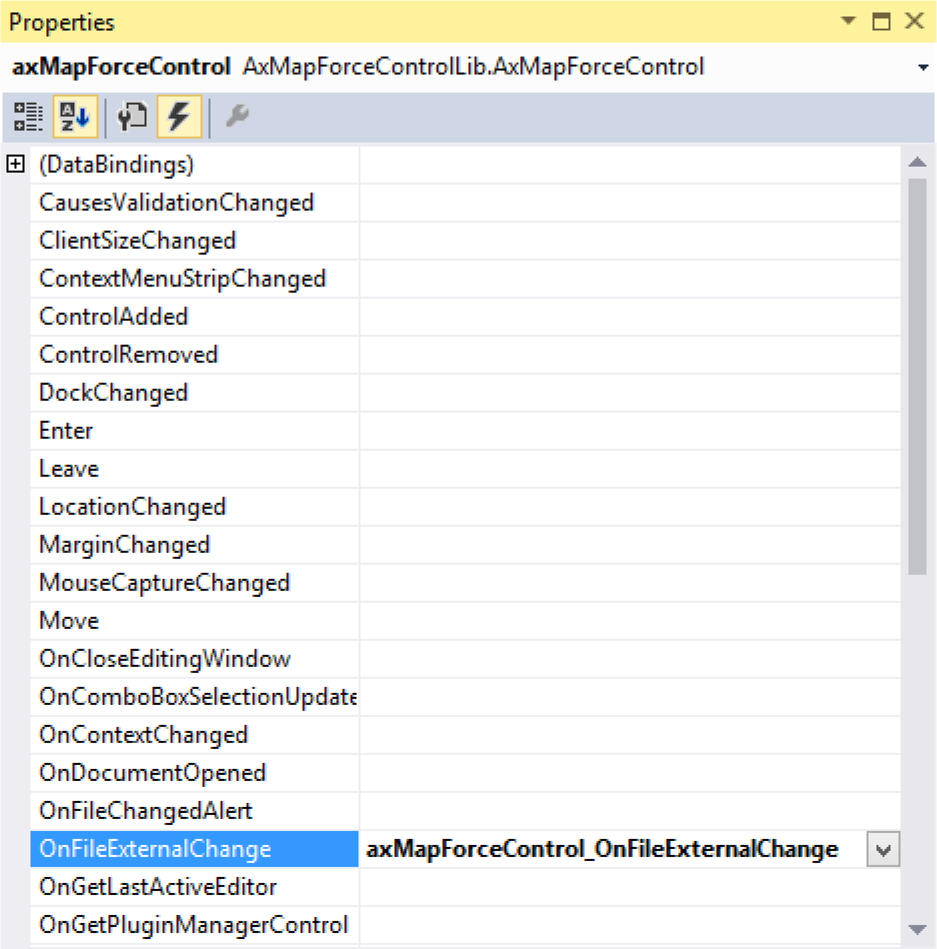
If the command is a container for other commands (that is, if it has child commands), it gets the event handler **`AltovaSubMenu_Popup`**. This handler queries the status of each child command and enables or disables it as required. This ensures that each command is enabled only when that is meaningful (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:



Properties

axMapForceControl AxMapForceControlLib.AxMapForceControl

⊞ (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 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 994).

- `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 1012.

Rules for mapping the ActiveX Control names to Java

For the documentation of ActiveX controls, see Object Reference 1034. 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 1012
- Creating the ActiveX Controls 1014
- Loading Data in the Controls 1015
- Basic Event Handling 1015
- Menus 1016
- UI Update Event Handling 1018
- Creating a MapForce Mapping Table 1018

16.5.2.1 Example Java Project

The MapForce installation package contains an example Java project, located in the ActiveX Examples folder of the application folder: `<ApplicationFolder>\Examples\ActiveX\Java\`.

The Java example shows how to integrate the MapForceControl in a common desktop application created with Java. You can test it directly from the command line using the batch file `BuildAndRun.bat`, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list

The Java examples folder contains all the files required to run the example project. These files are listed below:

`.classpath`	Eclipse project helper file
`.project`	Eclipse project file
`AltovaAutomation.dll`	Java-COM bridge: DLL part (for the 32-bit installation)
`AltovaAutomation_x64.dll`	Java-COM bridge: DLL part (for the 64-bit installation)

`AltovaAutomation.jar`	Java-COM bridge: Java library part
`BuildAndRun.bat`	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
`MapForceActiveX.jar`	Java classes of the MapForce ActiveX control
`MapForceActiveX_JavaDoc.zip`	Javadoc file containing help documentation for the Java API
`MapForceContainer.java`	Java example source code
`MapForceContainerEventHandler.java`	Java example source code
`MapForceTable.java`	Java example source code

What the example does

The example places one MapForce document editor window, the MapForce project window, the MapForce library window and the MapForce validation window in an AWT frame window. It reads out the main menu defined for MapForce and creates an AWT menu with the same structure. You can use this menu or the project window to open and work with files in the document editor.

You can modify the example in any way you like.

The following specific features are described in code listings:

- Creating the ActiveX Controls 1014: Starts MapForce, which is registered as an automation server, or activates MapForce if it is already running.
- Loading Data in the Controls 1015: Locates one of the example documents installed with MapForce and opens it.
- Basic Event Handling 1015: Changes the view of all open documents to Text View. The code also shows how to iterate through open documents.
- Menus 1016: Validates the active document and shows the result in a message box. The code shows how to use output parameters.
- UI Update Event Handling 1018: Shows how to handle MapForce events.
- Creating a MapForce Mapping Table 1018: 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\
\MapForce2025\\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 994).
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 994).
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**.

16.5.2.2 Creating the ActiveX Controls

The code listing below show how ActiveX controls can be created. The constructors will create the Java wrapper objects. Adding these Canvas-derived objects to a panel or to a frame will trigger the creation of the wrapped ActiveX object.

```
/**
 * MapForce manager control - always needed
 */
public static MapForceControl         mapForceControl = null;

/**
 * MapForceDocument editing control
 */
public static MapForceControlDocument     mapForceDocument = null;

/**
 * Tool windows - MapForce place-holder controls
 */
private static MapForceControlPlaceHolder   mapForceProjectToolWindow = null;
private static MapForceControlPlaceHolder   mapForceValidationToolWindow = null;
private static MapForceControlPlaceHolder   mapForceLibraryToolWindow = null;
```

```

// Create the MapForce ActiveX control; the parameter determines that we want
// to place document controls and place-holder controls individually.
// It gives us full control over the menu, as well.
   mapForceControl = new MapForceControl(
   ICActiveXIntegrationLevel.ICActiveXIntegrationOnDocumentLevel.getValue(), false );

   mapForceDocument = new MapForceControlDocument();
   frame.add( mapForceDocument, BorderLayout.CENTER );


// Create a project window and open the sample project in it
   mapForceProjectToolWindow = new MapForceControlPlaceHolder(
   MapForceControlPlaceholderWindow.MapForceXProjectWindow.getValue(),
   strExamplesFolder + "MapForceExamples.mfp" ) ;
   mapForceProjectToolWindow.setPreferredSize( new Dimension( 200, 200 ) );
```

16.5.2.3 Loading Data in the Controls

The code listing below show how data can be loaded in the ActiveX controls.

```
// Locate samples installed with the product.
   final String strExamplesFolder = System.getenv( "USERPROFILE" ) +
   "\\Documents\\Altova\\MapForce2025\\MapForceExamples\\";
   mapForceProjectToolWindow = new
MapForceControlPlaceHolder( MapForceControlPlaceholderWindow.MapForceXProjectWindow.getValu
e(), strExamplesFolder + "MapForceExamples.mfp" ) ;
```

16.5.2.4 Basic Event Handling

The code listing below shows how basic events can be handled. When calling the MapForceControl’s `open` method, or when trying to open a file via the menu or Project tree, the `onOpenedOrFocused` event is sent to the attached event handler. The basic handling for this event is opening the file by calling the MapForceDocumentControl’s `open` method.

```
    // Open the Marketing file when button is pressed
    btnMarkExp.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          // Instruct the Document control to open the file - avoid calling the open
method of MapForceControl (see help)
          mapForceDocument.open( strExamplesFolder + "MarketingExpenses.mfd" );
          mapForceDocument.requestFocusInWindow();
        } catch (AutomationException e1) {
          e1.printStackTrace();
        }
      }
    } );
    public void onOpenedOrFocused( String i_strFileName, boolean i_bOpenWithThisControl, boolean i_bFileAlreadyOpened ) throws AutomationException
{
  // Handle the New/Open events coming from the Project tree or from the menus
  if ( !i_bFileAlreadyOpened )
  {
    // This is basically an SDI interface, so open the file in the already existing document control
    try {
      MapForceContainer.mapForceDocument.open( i_strFileName );
      MapForceContainer.mapForceDocument.requestFocusInWindow();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

16.5.2.5 Menus

The code listing below shows how menu items can be created. Each `MapForceCommand` object gets a corresponding `MenuItem` object, with the `ActionCommand` set to the ID of the command. The actions generated by all menu items are handled by the same function, which can perform specific handlings (like reinterpreting the closing mechanism) or can delegate the execution to the `MapForceControl` object by calling its `exec` method. The `menuMap` object that is filled during menu creation is used later (see section UI Update Event Handling 1018).

```

    // 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();
  }

}
```

16.5.2.6 UI Update Event Handling

The code listing below shows how a UI-Update event handler can be created.

```
/**
   * Call-back from the MapForceControl.
   * Called to enable/disable commands
   */
  @Override
  public void onUpdateCmdUI() throws AutomationException
  {
    // A command should be enabled if the result of queryStatus contains the Supported (1) and Enabled (2) flags
    for ( java.util.Map.Entry<Integer, MenuItem> pair : menuMap.entrySet() )
      pair.getValue().setEnabled( MapForceContainer.mapForceControl.queryStatus( pair.getKey() ) > 2 );
  }
 /**
   * Call-back from the MapForceControl.
   * Usually called while enabling/disabling commands due to UI updates
   */
  @Override
  public boolean onIsActiveEditor( String i_strFilePath ) throws AutomationException
  {
    try {
      return MapForceContainer.mapForceDocument.getDocument().getFullName().equalsIgnoreCase( i_strFilePath );
    } catch ( Exception e ) {
      return false;
    }
  }
```

16.5.2.7 Listing the Properties of a MapForce Mapping

The listing below shows how a Mapping object in MapForce can be loaded as a table and prepared for modal activation.

```
//access MapForce Java-COM bridge
import com.altova.automation.MapForce.*;
import com.altova.automation.MapForce.Component;
import com.altova.automation.MapForce.Enums.ENUMComponentUsageKind;

//access AWT and Swing components
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;


/**
 * A simple example of a table control loading the structure from a Mapping object.
 * The class receives an Mapping object, loads its components in a JTable, and prepares
 * for modal activation.
 *
 * Feel free to modify and extend this sample.
 *
 * @author Altova GmbH
 */
class MapForceTable extends JDialog
{
  /**
   * The table control
   */
  private JTable myTable;

  /**
   * Constructor that prepares the modal dialog containing the filled table control
   * @param mapping The data to be displayed in the table
   * @param parent  Parent frame
   */
  public MapForceTable( Mapping mapping, Frame parent )
  {
    // Construct the modal dialog
    super( parent, "MapForce component table", true );
    // Build up the tree
    fillTable( mapping );
    // Arrange controls in the dialog
    setContentPane( new JScrollPane( myTable ) );
  }

  /**
   * Loads the components of a Mapping object in the table
   * @param mapping Source data
   */
  private void fillTable( Mapping mapping)
  {
    try
    {
      // count how many Instance components do we have
      int size = 0;
      for (Component comp : mapping.getComponents())
        if ( comp.getUsageKind() == 
ENUMComponentUsageKind.eComponentUsageKind_Instance )
```

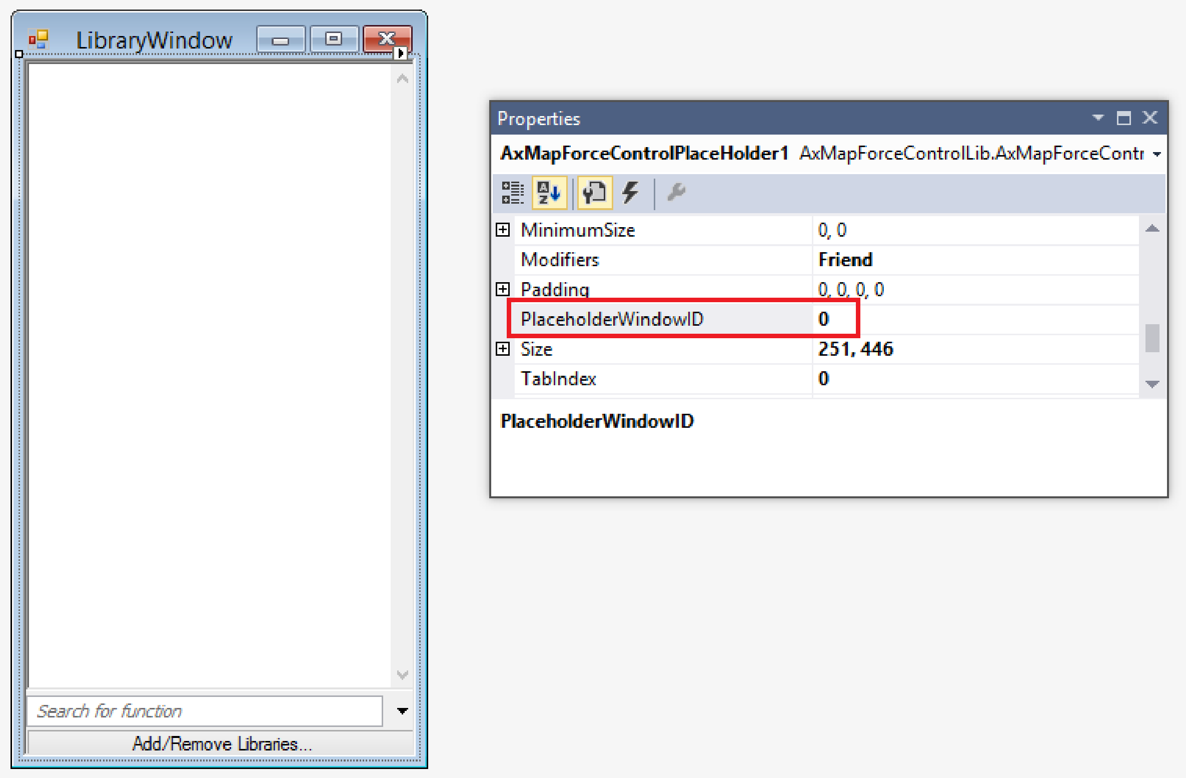
```
     ++size;

   // Prepare data
   final String[] columnNames = { "Component", "Has inputs", "Has outputs", "Input file", "Output file", "Schema" };
   final Object[][] data = new Object[size ][ 7 ] ;
   int index = 0 ;
   for (Component comp : mapping.getComponents())
    if ( comp.getUsageKind() == ENUMComponentUsageKind.eComponentUsageKind_Instance )
    {
     int i = 0;
     data[ index ][ i++ ] = comp.getName() ;
     data[ index ][ i++ ] = new Boolean( comp.getHasIncomingConnections() );
     data[ index ][ i++ ] = new Boolean( comp.getHasOutgoingConnections() );
     data[ index ][ i++ ] = comp.getInputInstanceFile();
     data[ index ][ i++ ] = comp.getOutputInstanceFile();
     data[ index++ ][ i ] = comp.getSchema() ;
    }

   // Set up table
   myTable = new JTable( new AbstractTableModel() {
     public String getColumnName(int col) { return columnNames[col]; }
     public int getRowCount() { return data.length; }
     public int getColumnCount() { return columnNames.length; }
     public Object getValueAt(int row, int col) { return data[row][col]; }
     public boolean isCellEditable(int row, int col) { return false; }
     public Class getColumnClass(int c) { return getValueAt(0, c).getClass(); }

   } );

   // Set width
   for( index = 0 ; index < columnNames.length ; ++index )
    myTable.getColumnModel().getColumn( index ).setMinWidth( 80 );
   myTable.getColumnModel().getColumn( 5 ).setMinWidth( 400 );
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

}
```

16.5.3 VB.NET

Source code which illustrates integration of MapForceControl into a VB.NET application can be found in the folder `<ApplicationFolder>\Examples\ActiveX\VB.NET` of your MapForce installation. The solution consists of three windows, as follows:

1. **`MainWindow.vb`** - the main document window, which also includes a basic application menu.

2. **`LibraryWindow.vb`** - the Library window. The contents of this window is populated by a Placeholder control which has the **`PlaceholderWindowID`** property set to 0 (this value instructs the control to display specifically the Library window).

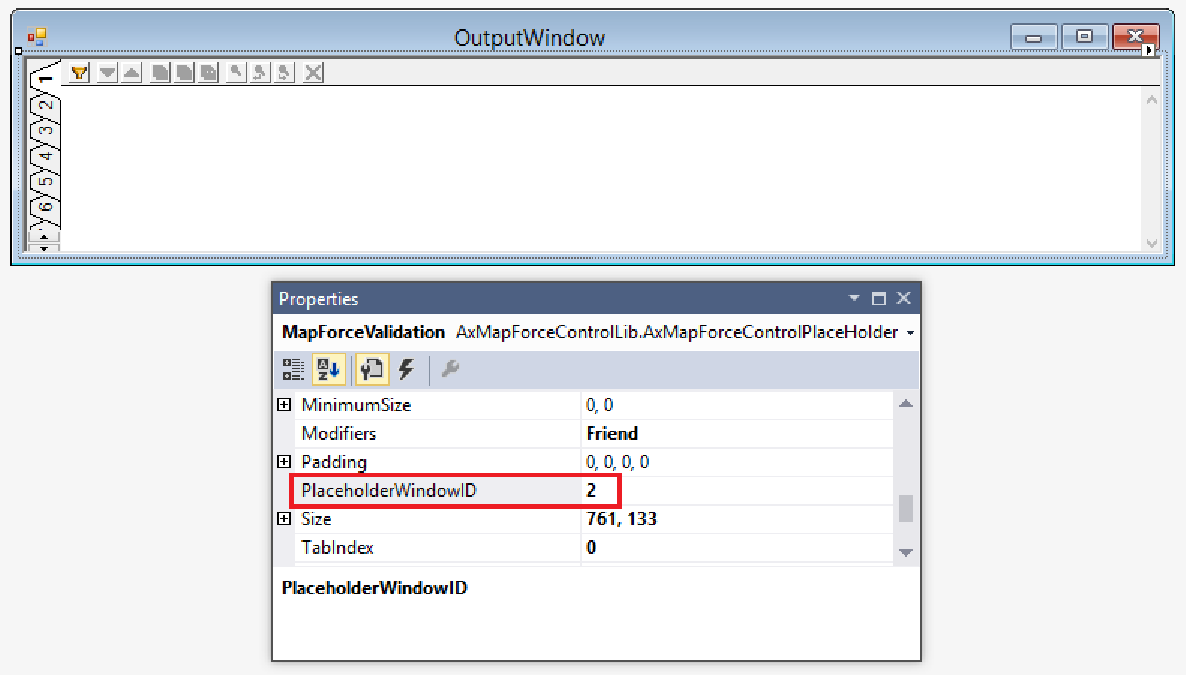


AxMapForceControlPlaceHolder1 AxMapForceControlLib.AxMapForceContr

⊞ 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 994.

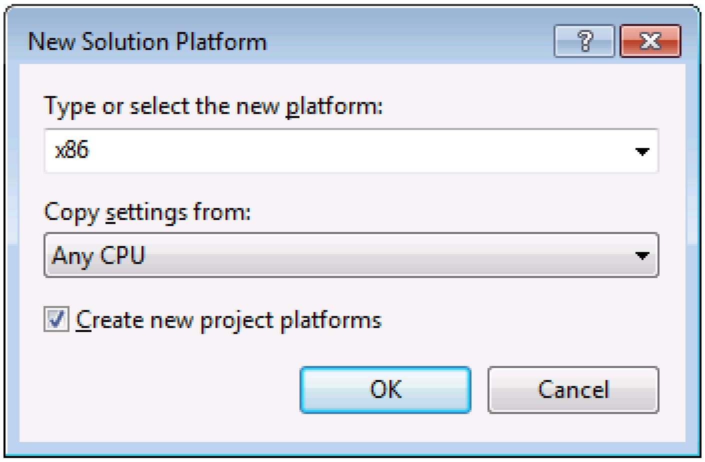
Step 2: Copy the sample to a directory where you have write permissions

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:

1. Right-click the solution in Visual Studio, and select **Configuration Manager**.
2. Under **Active solution platform**, select **New...** and then select the x86 or x64 configuration (in this example, **x86**).



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.

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`[1041] or the `MapForceControlDocument.Exec`[1047] methods. To query the status of a command, use the `MapForceControl.QueryStatus`[1042] or `MapForceControlDocument.QueryStatus`[1048] 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

Menu item	Command name	ID
Generate Code in Selected Language	ID_FILE_GENERATE_SELECTED_CODE	32362
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_CHANGEROOTELEMENT	32334

Menu item	Command name	ID
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

Menu item	Command name	ID
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	32966
Index...	IDC_HELP_INDEX	32967
Search...	IDC_HELP_SEARCH	32969
Software Activation...	IDC_ACTIVATION	32970
Order Form...	IDC_OPEN_ORDER_PAGE	32971
Registration...	IDC_REGISTRATION	32972
Check for Updates...	IDC_CHECK_FOR_UPDATES	32973
MapForce Product Comparison...	IDC_PRODUCT_COMPARISON	32955
Support Center...	IDC_OPEN_SUPPORT_PAGE	32961
FAQ on the Web...	IDC_OPEN_FAQ_PAGE	32962

Menu item	Command name	ID
Download Components and Free Tools…	IDC_OPEN_COMPONENTS_PAGE	32963
MapForce on the Internet..	IDC_OPEN_HOME_PAGE	32964
MapForce Training…	IDC_OPEN_TRAINING_PAGE	32965
About MapForce...	ID_APP_ABOUT	57664

16.7 Object Reference

Objects:
MapForceCommand 1034
MapForceCommands 1036
MapForceControl 1037
MapForceControlDocument 1045
MapForceControlPlaceHolder 1051

To give access to standard MapForce functionality, objects of the **MapForce automation interface** can be accessed as well. See MapForceControl.Application 1038, MapForceControlDocument.Document 1046 and MapForceControlPlaceHolder.Project 1052 for more information.

16.7.1 MapForceCommand

Properties:
`ID` 1035
`Label` 1035
`Name` 1035
`IsSeparator` 1035
`ToolTip` 1036
`StatusText` 1036
`Accelerator` 1035
`SubCommands` 1036

Description:
A command object can be one of the following: an executable command, a command container (for example, a menu, submenu, or toolbar), or a menu separator. To determine what kind of information is stored in the current `Command` object, query its `ID`, `IsSeparator`, and `SubCommands` properties, as follows.

The Command object is...	When...
An executable command	• `ID` is greater than zero • `IsSeparator` is false • `SubCommands` is empty
A command container	• `ID` is zero • `IsSeparator` is false • `SubCommands` contains a collection of `Command` objects.
Separator	• `ID` is zero • `IsSeparator` is true

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` 1041) or query its status (using `QueryStatus` 1042). 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 1034.

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 (1036)

Description:
The `SubCommands` property gets the collection of `Command` (1034) 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` (1037)
`Item` (1037)

Description:
Collection of `Command` (1034) objects to get access to command labels and IDs of the MapForceControl. Those commands can be executed with the `Exec` (1041) method and their status can be queried with `QueryStatus` (1042).

16.7.2.1 Count

Property: `Count` as `long`

Description:
Number of `Command`[1034] objects on this level of the collection.

16.7.2.2 Item

Property: `Item` (n as `long`) as `Command`[1034]

Description:
Gets the command with the index `n` in this collection. Index is 1-based.

16.7.3 MapForceControl

Properties:
`IntegrationLevel`[1039]
`Appearance`[1038]
`Application`[1038]
`BorderStyle`[1038]
`CommandsList`[1039]
`EnableUserPrompts`[1039]
`MainMenu`[1040]
`Toolbars`[1040]

Methods:
`Open`[1042]
`Exec`[1041]
`QueryStatus`[1042]

Events:
`OnUpdateCmdUI`[1044]
`OnOpenedOrFocused`[1044]
`OnCloseEditingWindow`[1043]
`OnFileChangedAlert`[1043]
`OnDocumentOpened`[1043]
`OnValidationWindowUpdated`[1044]

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` 1039
`EnableUserPrompts` 1039
`Appearance` 1038
`BorderStyle` 1038

Command related properties:
`CommandsList` 1039
`MainMenu` 1040
`Toolbars` 1040

Access to MapForceAPI:
`Application` 1038

16.7.3.1.1 Appearance

Property: `Appearance` as `short`

Dispatch Id: *-520*

Description:
A value not equal to `0` displays a client edge around the control. Default value is `0`.

16.7.3.1.2 Application

Property: `Application` as `Application`

Dispatch Id: *1*

Description:
The `Application` property gives access to the `Application` object of the complete MapForce automation server API. The property is read-only.

16.7.3.1.3 BorderStyle

Property: `BorderStyle` as `short`

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

16.7.3.1.4 CommandsList

Property: `CommandList` as [`Commands`] 1036 (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`] 1040. To get toolbar commands, use [`Toolbars`] 1040.

```
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

16.7.3.1.5 EnableUserPrompts

Property: `EnableUserPrompts` as `boolean`

Dispatch Id: *1006*

Description:
Setting this property to *false*, disables user prompts in the control. The default value is *true*.

16.7.3.1.6 IntegrationLevel

Property: `IntegrationLevel` as [`ICActiveXIntegrationLevel`] 1054

Dispatch Id: *1000*

Description:
The `IntegrationLevel` property determines the operation mode of the control. See also [Integration at Application Level] 998 and [Integration at Document Level] 1001 for more information.

Note: It is important to set this property immediately after the creation of the `MapForceControl` object.

16.7.3.1.7 MainMenu

Property: `MainMenu as Command`[1034] (read-only)

Dispatch Id: *1003*

Description:
This property provides information about the structure and commands available in the MapForceControl main menu, as a `Command` object. The `Command` object contains all available submenus of MapForce (for example "File", "Edit", "View" etc.). To access the submenu objects, use the `SubCommands` property of the `MainMenu` property. Each submenu is also a `Command` object. For each submenu, you can then further iterate through their `SubCommands` property in order to get their corresponding child commands and separators (this technique may be used, for example, to create the application menu programmatically). Note that some menu commands act as containers ("parents") for other menu commands, in which case they also have a `SubCommands` property. To get the structure of all menu commands programmatically, you will need a recursive function, as illustrated for C# in Retrieving Command Information[1007].

```
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

16.7.3.1.8 Toolbars

Property: `Toolbars` as `Commands`[1036] (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` 1042

`Exec` 1041

`QueryStatus` 1042

16.7.3.2.1 Exec

Method: `Exec` (`nCmdID` as `long`) as `boolean`

Dispatch Id: *6*

Description:
This method calls the MapForce command with the ID `nCmdID`. If the command can be executed, the method returns `true`. To get a list of all available commands, use `CommandsList` 1039. To retrieve the status of any command, use `QueryStatus` 1042.

16.7.3.2.2 Open

Method: `Open` (`strFilePath` as `string`) as `boolean`

Dispatch Id: *5*

Description:
The result of the method depends on the extension passed in the argument `strFilePath`. If the file extension is `.sps`, a new document is opened. If the file extension is `.svp`, the corresponding project is opened. If a different file extension is passed into the method, the control tries to load the file as a new component into the active document.

Do not use this method to load documents or projects when using the control in document-level integration mode. Instead, use `MapForceControlDocument.Open` 1048 and `MapForceControlPlaceHolder.OpenProject` 1053.

16.7.3.2.3 QueryStatus

Method: `QueryStatus` (`nCmdID` as `long`) as `long`

Dispatch Id: *7*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if `QueryStatus` returns `0` the command ID is not recognized as a valid MapForce command. If `QueryStatus` returns a value of `1` or `5`, the command is disabled.

16.7.3.3 Events

The MapForceControl ActiveX control provides the following connection point events:

`OnUpdateCmdUI` 1044

`OnOpenedOrFocused` 1044

`OnCloseEditingWindow` 1043

`OnFileChangedAlert` 1043

`OnDocumentOpened` 1043

`OnValidationWindowUpdated` 1044

16.7.3.3.1 OnCloseEditingWindow

Event: `OnCloseEditingWindow` (i_strFilePath as `String`) as `boolean`

Dispatch Id: *1002*

Description:
This event is triggered when MapForce needs to close an already open document. As an answer to this event, clients should close the editor window associated with *i_strFilePath*. Returning *true* from this event indicates that the client has closed the document. Clients can return *false* if no specific handling is required and MapForceControl should try to close the editor and destroy the associated document control.

16.7.3.3.2 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`1050 instead.

16.7.3.3.3 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.

16.7.3.3.4 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).

16.7.3.3.5 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.

16.7.3.3.6 OnToolWindowUpdated

Event: `OnToolWindowUpdated(pToolWnd as long )`

Dispatch Id: *1006*

Description:
This event is triggered when the tool window is updated.

16.7.3.3.7 OnUpdateCmdUI

Event: `OnUpdateCmdUI()`

Dispatch Id: *1003*

Description:
Called frequently to give integrators a good opportunity to check status of MapForce commands using `MapForceControl.QueryStatus` 1042. Do not perform long operations in this callback.

16.7.3.3.8 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` 1046
`BorderStyle` 1046
`Document` 1046
`IsModified` 1046
`Path` 1047
`ReadOnly` 1047

Methods:
`Exec` 1047
`New` 1048
`Open` 1048
`QueryStatus` 1048
`Reload` 1048
`Save` 1049
`SaveAs` 1049

Events:
`OnDocumentOpened` 1050
`OnDocumentClosed` 1050
`OnModifiedFlagChanged` 1051
`OnFileChangedAlert` 1050
`OnActivate` 1049

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` 1047
`IsModified` 1046
`Path` 1047
`Appearance` 1046
`BorderStyle` 1046

Access to MapForceAPI:

Document 1046

16.7.4.1.1 Appearance

Property: `Appearance` as `short`

Dispatch Id: *-520*

Description:
A value not equal to `0` displays a client edge around the document control. Default value is `0`.

16.7.4.1.2 BorderStyle

Property: `BorderStyle` as `short`

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

16.7.4.1.3 Document

Property: `Document` as `Document`

Dispatch Id: *1*

Description:
The `Document` property gives access to the `Document` object of the MapForce automation server API. This interface provides additional functionality which can be used with the document loaded in the control. The property is read-only.

16.7.4.1.4 IsModified

Property: `IsModified` as `boolean` (read-only)

Dispatch Id: *1006*

Description:
`IsModified` is *true* if the document content has changed since the last open, reload or save operation. It is *false*, otherwise.

16.7.4.1.5 Path

Property: `Path` as `string`

Dispatch Id: *1005*

Description:
Sets or gets the full path name of the document loaded into the control.

16.7.4.1.6 ReadOnly

Property: `ReadOnly` as `boolean`

Dispatch Id: *1007*

Description:
Using this property you can turn on and off the read-only mode of the document. If `ReadOnly` is `true` it is not possible to do any modifications.

16.7.4.2 Methods

The following methods are defined:

Document handling:
`New` 1048
`Open` 1048
`Reload` 1048
`Save` 1049
`SaveAs` 1049

Command Handling:
`Exec` 1047
`QueryStatus` 1048

16.7.4.2.1 Exec

Method: `Exec` (`nCmdID` as `long`) as `boolean`

Dispatch Id: *8*

Description:
`Exec` calls the MapForce command with the ID `nCmdID`. If the command can be executed, the method returns `true`. This method should be called only if there is currently an active document available in the application.

To get commands organized according to their menu structure, use the `MainMenu` 1040 property of MapForceControl. To get toolbar commands, use the `Toolbars` 1040 property of the MapForceControl.

16.7.4.2.2 New

Method: `New` () as `boolean`

Dispatch Id: *1000*

Description:
This method initializes a new document inside the control.

16.7.4.2.3 Open

Method: `Open` (`strFileName` as `string`) as `boolean`

Dispatch Id: *1001*

Description:
`Open` loads the file `strFileName` as the new document into the control.

16.7.4.2.4 QueryStatus

Method: `QueryStatus` (`nCmdID` as `long`) as `long`

Dispatch Id: *9*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if `QueryStatus` returns `0` the command ID is not recognized as a valid MapForce command. If `QueryStatus` returns a value of `1` or `5` the command is disabled. The client should call the `QueryStatus` method of the document control if there is currently an active document available in the application.

16.7.4.2.5 Reload

Method: `Reload()` as `boolean`

Dispatch Id: *1002*

Description:
`Reload` updates the document content from the file system.

16.7.4.2.6 Save

Method: `Save()` as `boolean`

Dispatch Id: *1003*

Description:
`Save` saves the current document at the location `Path`1047.

16.7.4.2.7 SaveAs

Method: `SaveAs` (`strFileName` as `string`) as `boolean`

Dispatch Id: *1004*

Description:
`SaveAs` sets `Path`1047 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`1050
`OnDocumentClosed`1050
`OnModifiedFlagChanged`1051
`OnFileChangedAlert`1050
`OnActivate`1049
`OnSetEditorTitle`1051

16.7.4.3.1 OnActivate

Event: `OnActivate` ()

Dispatch Id: *1005*

Description:
This event is triggered when the document control is activated, has the focus, and is ready for user input.

16.7.4.3.2 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.

16.7.4.3.3 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.

16.7.4.3.4 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.

16.7.4.3.5 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.

16.7.4.3.6 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.

16.7.4.3.7 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` 1052

Properties for project placeholder window:
`Project` 1052

Methods for project placeholder window:
`OpenProject` 1053
`CloseProject` 1053

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` 1052

Access to MapForceAPI:
`Project` 1052

16.7.5.1.1 Label

Property: `Label` as `String` (read-only)

Dispatch Id: *1001*

Description:
This property gives access to the title of the placeholder. The property is read-only.

16.7.5.1.2 PlaceholderWindowID

Property: `PlaceholderWindowID as MapForceControlPlaceholderWindow`[1054]

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`[1054] enumeration. The control changes its state immediately and shows the new MapForce window.

16.7.5.1.3 Project

Property: `Project` as Project (read-only)

Dispatch Id: *2*

Description:
The `Project` property gives access to the `Project` object of the MapForce automation server API. This interface provides additional functionality which can be used with the project loaded into the control. The property will return a valid project interface only if the placeholder window has `PlaceholderWindowID`[1052] with a value of `MapForceXProjectWindow (=3)`. The property is read-only.

16.7.5.2 Methods

The following method is defined:

`OpenProject`[1053]
`CloseProject`[1053]

16.7.5.2.1 OpenProject

Method: `OpenProject` (`strFileName` as `string`) as `boolean`

Dispatch Id: *3*

Description:
`OpenProject` loads the file `strFileName` as the new project into the control. The method will fail if the placeholder window has a **`PlaceholderWindowID`** 1052 different to `MapForceXProjectWindow (=3)`.

16.7.5.2.2 CloseProject

Method: `CloseProject ()`

Dispatch Id: *4*

Description:
`CloseProject` closes the project loaded by the control. The method will fail if the placeholder window has a `PlaceholderWindowID` 1052 different to `MapForceXProjectWindow (=3)`.

16.7.5.3 Events

The MapForceControlPlaceholder ActiveX control provides following connection point events:

`OnModifiedFlagChanged` 1053

16.7.5.3.1 OnModifiedFlagChanged

Event: `OnModifiedFlagChanged` (`i_bIsModified` as `boolean`)

Dispatch Id: *1*

Description:
This event gets triggered only for placeholder controls with a `PlaceholderWindowID` 1052 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.

16.7.5.3.2 OnSetLabel

Event: `OnSetLabel(i_strNewLabel as string)`

Dispatch Id: *1000*

Description:
Raised when the title of the placeholder window is changed.

16.7.6 Enumerations

The following enumerations are defined:

ICActiveXIntegrationLevel 1054
MapForceControlPlaceholderWindow 1054

16.7.6.1 ICActiveXIntegrationLevel

Possible values for the IntegrationLevel 1039 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
```

17 Menu Commands

This section describes the MapForce menu commands. The following menu commands are available:

- File 1056
- Edit 1058
- Insert 1059
- Project 1062
- Component 1064
- Connection 1066
- Function 1067
- Output 1068
- Debug 1070
- View 1071
- Tools 1073
- Window 1090
- Help 1091

17.1 File

This topic lists all the menu commands available in the **File** menu.

New

Creates a new mapping document. In Professional and Enterprise editions, you can also create a mapping project (`.mfp`). For details, see Projects 95.

Open

Opens the previously saved mapping design (`.mfd`). In Professional and Enterprise editions, you can also open a mapping project (`.mfp`). For details, see Projects 95.

Save/Save As/Save All

The **Save** option saves the currently active mapping with its current name. The **Save As** option allows you to save the currently open mapping with a different name. The **Save All** command saves all currently open mapping files.

Reload

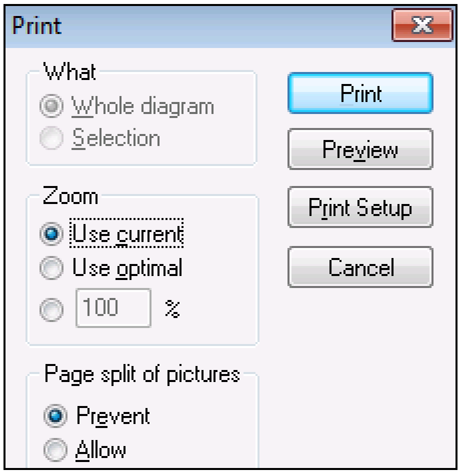
Reloading the currently active mapping reverts your last changes.

Close/Close All

The **Close** command closes the currently active mapping. The **Close All** command closes all currently open mappings. You are asked if you want to save any of the unsaved files.

Print/Print Preview/Print Setup

The **Print** command opens the **Print** dialog box (*see below*) that enables you to print out your mapping. **Use current** retains the currently defined zoom factor of the mapping. **Use optimal** scales the mapping to fit the page size. You can also specify the zoom factor numerically. Component scrollbars are not printed. You can also specify if you want to allow graphics to be split over several pages or not.



Print

What
● Whole diagram
○ Selection

Zoom
● Use current
○ Use optimal
○ 100 %

Page split of pictures
● Prevent
○ Allow

The **Preview** command opens the same **Print** dialog box with the same settings as described above. The **Print Setup** command opens the **Print Setup** dialog box in which you can select a printer and configure paper settings.

⊟ Validate Mapping

The **Validate Mapping** command checks whether all mappings are valid and displays relevant information messages, warnings, and errors. For details, see Validation 64.

⊟ Mapping Settings

Opens the Mapping Settings dialog box 75 where you can define document-specific settings.

⊟ Open Credentials Manager (*Enterprise Edition*)

Opens **Credentials Manager** that allows you to manage credentials required in mappings that perform basic HTTP authentication or OAuth 2.0 authorization.

⊟ Generate Code in Selected Language/Generate Code in

The **Generate Code in Selected Language** command generates code in language selected in the toolbar. The **Generate Code in <language>** command enables you to generate code in XSLT 1-3 (*all editions*), XQuery, Java, C#, and C++ (*Professional and Enterprise editions*). Selecting either command opens the **Browse for Folder** dialog box in which you need to select the location of the generated files. The names of the generated files are defined in the Mapping Settings 75 dialog box.

For more information about the available transformation languages, see Transformation Languages 19. For more information about generating code, see Code Generation 66.

⊟ Compile to MapForce Server Execution File (*Professional and Enterprise editions*)

Generates a file that can be executed by MapForce Server to run the mapping transformation. For details, see Compiling a MapForce mapping 799.

⊟ Deploy to FlowForce Server (*Professional and Enterprise editions*)

Deploys the currently active mapping to FlowForce Server. For details, see Deploying a MapForce mapping 802.

⊟ Generate Documentation (*Professional and Enterprise editions*)

Generates documentation of your mapping projects in great detail in various output formats. For more information, see Generating Mapping Documentation 761.

⊟ Recent files

Displays the list of the most recently opened files.

⊟ Exit

Exits the application. You are asked if you want to save any unsaved files.

17.2 Edit

This topic lists all the menu commands available in the **Edit** menu. Most of the commands in this menu become active when you view the result of a mapping in the **Output** pane or preview code, for example, in the **XSLT** pane.

Undo

MapForce has an unlimited number of undo steps that you can use to retrace your mapping steps. You can also use the ↶ toolbar command to undo actions.

Redo

The redo command allows you to redo previously undone actions. You can step backward and forward through the undo history using both these commands. You can also use the ↷ toolbar command to redo actions.

Find

Allows you to search for specific text in any of the **XQuery** (*Professional and Enterprise editions*), **XSLT**, **XSLT2**, **XSLT3**, and **Output** panes. You can also do a search using the 🔍 toolbar command.

Find Next

Searches for the next occurrence of the same search string. You can also search for the next occurrence using the 🔍 toolbar button.

Find Previous

Searches for the previous occurrence of the same search string. Searching for the previous occurrence is also possible with the 🔍 toolbar command.

Cut/Copy/Paste/Delete

The standard windows Edit commands that allow you to cut, copy, paste, and delete any components or functions visible in the mapping window.

Select all

Selects all components in the **Mapping** pane or the text/code in the **XQuery** (*Professional and Enterprise editions*), **XSLT**, **XSLT2**, **XSLT3**, and **Output** panes.

17.3 Insert

This topic lists all the menu commands available in the **Insert** menu.

XML Schema/File

Adds an XML schema or instance file to the mapping. If you select an XML file without a schema reference, MapForce can generate a matching XML schema 132. If you select an XML schema file, you are prompted to include an XML instance file which supplies the data for preview. You can also add an XML/XSD file via the toolbar command.

Database (*Professional and Enterprise editions*)

Adds a database component. See Databases 165. You can also add a database component via the toolbar command. In MapForce Enterprise Edition, you can also add NoSQL databases as components.

EDI (*Enterprise Edition*)

Adds an EDI document. You can also add an EDI component via the EDI toolbar command.

Text File (*Professional and Enterprise editions*)

Adds a flat file document such as CSV or a fixed-length text file. For details, see CSV and Text Files 323. You can also add a text file via the toolbar command. MapForce Enterprise Edition also allows you to process text files with FlexText.

Web Service Function (*Enterprise Edition*)

Adds a call to a Web service. You can also add a Web service via the toolbar button.

Excel 2007+ File (*Enterprise Edition*)

Adds a Microsoft Excel 2007+ (`.xlsx`) file. If you do not have Excel 2007+ installed on your machine, you can still map to or from Excel 2007+ files. In this case, you cannot preview the result in the **Output** pane, but you can still save the result. You can also add an Excel file via the toolbar command.

XBRL Document (*Enterprise Edition*)

Adds an XBRL instance or taxonomy document. You can also add an XBRL component via the XBRL toolbar command.

JSON Schema/File (*Enterprise Edition*)

Adds a JSON schema or file. You can also add a JSON component via the JSON toolbar command.

Protocol Buffers File (*Enterprise Edition*)

Adds a binary file encoded in Protocol Buffers format. You can also add a file in Protocol Buffers format via the toolbar command.

Insert PDF Document (*Enterprise Edition*)

Adds a PDF document. You can also insert a PDF document via the toolbar.

Insert Input

Simple-Input components can be used as input parameters that are relevant to the entire mapping or only in the context of user-defined functions. For more information, see Simple Input[346] and Parameters in UDFs[463]. You can also insert a Simple-Input component using the toolbar command.

Insert Output

Simple-Output components can be used as output components in mappings and as output parameters of user-defined functions. For more information, see Simple Output[356] and Parameters in UDFs[463]. You can also insert a Simple-Output component using the toolbar command.

Constant

Inserts a constant which supplies fixed data to an input connector. You can select the following types of data: `String`, `Number` and `All other`. You can also insert a constant using the toolbar command.

Variable

Inserts a variable[360], which is equivalent to a regular (non-inline) user-defined function. A variable is a special type of components used to store an intermediate mapping result for further processing. You can also add a variable using the toolbar command.

Join (*Professional and Enterprise editions*)

The Join component allows you to join data in SQL and non-SQL modes. You can also add a Join component using the toolbar command. For details, see Joining Data[373].

Sort: Nodes/Rows

Inserts a component which allows you to sort nodes (see Sort Nodes/Rows[402]). You can also add a Sort component using the toolbar command.

Filter: Nodes/Rows

Inserts a Filter component that can filter data from any other component structure supported by MapForce, including databases. For more information, see Filters and Conditions[408]. You can also add a filter using the toolbar command.

SQL/NoSQL-WHERE/ORDER (*Professional and Enterprise editions*)

Inserts a component which allows you to filter database data conditionally. For more information, see

SQL/NoSQL-WHERE/ORDER Component 413. You can also access the SQL/NoSQL-WHERE/ORDER component via the toolbar command.

Value-Map

Inserts a component that transforms an input value to an output value using a lookup table. This is useful when you need to map a set of values to another set of values (e.g., month numbers to month names).

For more information, see Value-Maps 421. You can also insert a Value-Map using the toolbar command.

IF-Else Condition

Inserts an If-Else Condition that is suitable for scenarios where you need to process a simple value conditionally. For more information, see Filters and Conditions 408. You can also add an If-Else Condition using the toolbar command.

Exception (*Professional and Enterprise editions*)

The exception component allows you to interrupt a mapping process when a specific condition is met.

You can also add an Exception component using the toolbar command. In MapForce Enterprise Edition, this component also allows you to define Fault messages in WSDL mapping projects. For more information about Exception components, see Exceptions 432.

Comment

This menu command allows you to insert sticky-note-style comments as free-standing components. For details, see Comments 36. You can also add a comment by clicking the toolbar command.

17.4 Project

MapForce allows you to group your mappings into mapping projects 95. This topic lists all the menu commands available in the **Project** menu.

⊟ Reload Project

Reloads the currently active project and switches to the **Project** window.

⊟ Close Project

Closes the currently active project.

⊟ Save Project

Saves the currently active project.

⊟ Add Files to Project

Allows you to add mappings to the current project.

⊟ Add Active File to Project

Adds the currently active file to the currently open project.

⊟ Create Folder

This option adds a new folder to the current project. See Project Folders 98.

⊟ Create Web Link

This command enables you to create a link to an external Web resource and add this link to your project. For details, Projects 95.

⊟ Open Mapping (*Enterprise Edition*)

Opens the currently highlighted/selected mapping in the **Project** window.

⊟ Create Mapping for Operation (*Enterprise Edition*)

Creates a mapping file for the currently selected operation of the WSDL project.

⊟ Add Mapping File for Operation (*Enterprise Edition*)

Allows you to add a previously saved mapping file to the currently active WSDL operation.

⊟ Insert Web Service (*Enterprise Edition*)

Allows you to insert a Web Service based on an existing WSDL file.

⊟ Open in XMLSpy

Opens the selected WSDL file in Altova XMLSpy.

⊟ Generate Code for Entire Project

Generates code for the entire project currently visible in the **Project** window. Code is generated in the selected language for all the `.mfd` files in each folder.

⊟ Generate Code in

Generates project code in the language you select from the context menu.

⊟ Properties

Opens a dialog box where you can define project settings 97.

⊟ Recent projects

Displays a list of the recently opened projects.

17.5 Component

This topic lists all the menu commands available in the **Component** menu.

⊟ Change Root Element

Allows you to change the root element of an XML instance document.

⊟ Edit Schema Definition in XMLSpy

To be able to edit a schema in Altova XMLSpy, you need to click an XML component and then select the option **Edit Schema Definition in XMLSpy**.

⊟ Edit FlexText Configuration (*Enterprise Edition*)

This command enables you to edit a FlexText file.

⊟ Add/Remove/Edit Database Objects (*Professional and Enterprise editions*)

Allows you to add, remove, and change database objects in a database component. See Databases[165].

⊟ Create Mapping to EDI X12 997 (*Enterprise Edition*)

The X12 997 Functional Acknowledgment reports the status of the EDI interchange. All errors encountered during processing of the document are reported in it. MapForce can automatically generate a X12 997 document that you will be able to send to the recipient.

⊟ Create Mapping to EDI X12 999 (*Enterprise Edition*)

The X12 999 Implementation Acknowledgment Transaction Set reports HIPAA implementation guide non-compliance or application errors. MapForce can automatically generate an X12 999 component and automatically create the necessary mapping connections.

⊟ Refresh (*Professional and Enterprise editions*)

Reloads the structure of the currently active database component.

⊟ Add Duplicate Input Before/After

Inserts a copy of the selected item before/after this item. Duplicated input cannot be used as a data source. For more information, see Duplicate Input[42].

⊟ Remove Duplicate

Removes a duplicated item.

⊟ Comment/Processing Instructions

This option enables you to insert comments and processing instructions[143] into XML components.

⊟ Write Content as CDATA Section

This command creates a CDATA section[143] that is used to represent parts of a document as character data which would normally be interpreted as markup.

⊟ Database Table Actions (*Professional and Enterprise editions*)

Allows you to configure database insert, update, and delete actions, and other options for database records. See Database Table Actions Settings 259 for more information.

⊟ Query Database (*Professional and Enterprise editions*)

Creates a SELECT statement based on the table/field you clicked in the database component. Clicking a table/field makes this command active, and the SELECT statement is automatically placed into the **Select** window.

⊟ Align Tree Left

Makes the tree of a component left-justified.

⊟ Align Tree Right

Makes the tree of a component right-justified.

⊟ Edit Comment

If you have a comment component in your mapping, you can edit it, by clicking on it and then selecting the **Edit Comment** command. Alternatively, you can double-click inside the comment component and edit the text directly in the comment box. For more information about comment components and their types, see Comment 36.

⊟ Properties

Displays the settings of the currently selected component. See Change Component Settings 41.

17.6 Connection

This topic lists all the menu commands available in the **Connection** menu.

Auto-Connect Matching Children

Activates/deactivates the **Auto Connect Matching Children** option. For more information about connections and their types, see Connections 48.

Settings for Connect Matching Children

Helps you define matching-children connections. For details, see Matching-Children Connections 54.

Connect Matching Children

This command allows you to create multiple connections for items with the same names in source and target components. The settings you define in this dialog box apply if the (**Auto connect child items**) toolbar command has been enabled. For more information, see Matching-Children Connections 54.

Target-Driven (Standard)

Changes the connector type to a standard mapping. For further information, see Target-driven vs. source-driven connections 51.

Copy-All (Copy Child Items)

Creates connections for all matching child items. The main benefit of copy-all connections is that they visually simplify the mapping workspace: One connection, represented by a thick line, is created instead of multiple connections. For details, see Copy-All Connections 56.

Source-Driven (Mixed Content)

Changes the connection type to a source-driven connection that enables you to automatically map mixed content (text and child nodes) in the same order as in the XML *source* file. For more information, see Source-Driven Connections 52.

Properties

Opens the **Connection Settings** dialog box which allows you to define connection types and annotation settings. For more information, see Connection Settings 58.

17.7 Function

This topic lists all the menu commands available in the **Function** menu.

⊟ Create User-Defined Function

Creates a user-defined function[457] (UDF). You can also create a UDF using the toolbar command.

⊟ Create User-Defined Function from Selection

Creates a user-defined function based on the currently selected elements in the mapping window. For details, see Create UDFs[460]. You can also create a UDF from selection using the toolbar command.

⊟ Function Settings

Opens the **Edit User-defined Function** dialog box that allows you to change a UDF's settings. For details, see Edit UDFs[461].

⊟ Remove Function

Deletes the currently active user-defined function if you are working in a context which allows this.

⊟ Insert Input

Simple-Input components can be used as input parameters that are relevant to the entire mapping or only in the context of user-defined functions. For more information, see Simple Input[346] and Parameters in UDFs[463]. You can also insert a Simple-Input component using the toolbar command.

⊟ Insert Output

Simple-Output components can be used as output components in mappings and as output parameters of user-defined functions. For more information, see Simple Output[356] and Parameters in UDFs[463]. You can also insert a Simple-Output component using the toolbar button.

17.8 Output

This topic lists all the menu commands available in the **Output** menu.

⊟ XSLT 1.0/XSLT 2.0/XSLT 3.0/XQuery/Java/C#/C++/Built-In

Sets the transformation language in which the mapping should be executed. The selection of transformation languages depends on your MapForce edition. For details, see Transformation Languages[19]. You can also select a transformation language in the toolbar.

⊟ Validate Output File

Validates the output XML file against the referenced schema. See Validation[64].

⊟ Save Output File

Saves the data in the **Output** pane to a file.

⊟ Save All Output Files

Saves all the generated output files of dynamic mappings[746]. See also Tutorial 4[123].

⊟ Regenerate Output

Reloads the data in the **Output** pane.

⊟ Run SQL/NoSQL-Script (*Professional and Enterprise editions*)

If an SQL/NoSQL script is currently visible in the **Output** pane, the script executes the mapping to the target database, taking the defined table actions into account. To find out more about databases, see Databases[165].

⊟ Insert/Remove Bookmark

Inserts/removes a bookmark at the cursor position in the **Output** pane.

⊟ Next/Previous Bookmark

Navigates to the next/previous bookmark in the **Output** pane.

⊟ Remove All Bookmarks

Removes all currently defined bookmarks in the **Output** pane.

⊟ Pretty-Print XML Text

Reformats your XML document in the **Output** pane so that the document has a structured display: Each child node is offset from its parent by a single tab character. In the **Output** pane, the Tab size settings defined in the Text View Settings[68] dialog (Tabs group) take effect.

⊟ Text View Settings

Displays the **Text View Settings** dialog box that allows you to customize text view settings in the **XQuery** pane (*Professional and Enterprise editions*), the **Output** pane, and the **XSLT** pane. The dialog also shows the currently defined hotkeys. For more information, see Text View Features 68.

17.9 Debug

This topic lists all the menu commands available in the **Debug** menu.

⊟ Start Debugging

Starts or continues debugging until a breakpoint is hit or the mapping finishes. You can also start debugging using the ▶ toolbar command. For more information about debugging mode, see Debugger[771].

⊟ Stop Debugging

Stops debugging. This command exits the debug mode and switches MapForce back to standard mode. You can also stop debugging using the ■ toolbar command. For more information about debugging mode, see Debugger[771].

⊟ Step Into

Executes the mapping until a single step is finished anywhere in the mapping. In the mapping debugger, a step is a logical group of dependent computations which normally produce a single item of a sequence. Depending on the mapping context, this command roughly says the following: *go to the left/go to target child/go to source parent*. You can also access this command using the toolbar button. For more information about debugging mode, see Debugger[771].

⊟ Step Over

Continues execution until the current step finishes (or finishes again for another item of the sequence) or an unrelated step finishes. This command steps over computations that are inputs of the current step. You can also access this command using the toolbar button. For more information about debugging mode, see Debugger[771].

⊟ Step Out

Continues execution until the result of the current step is consumed or a step is executed that is not an input or child of the consumption. This command steps out of the current computation. Depending on the mapping context, this command roughly says the following: *go to the right/go to target parent/go to source child*. You can also access this command using the toolbar button. For more information about debugging mode, see Debugger[771].

⊟ Minimal Step

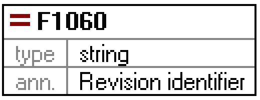
Continues execution until a value is produced or consumed. This command subdivides a step and will typically stop twice for each connection: (i) once when its source produces a value and (ii) once when its target consumes it. MapForce does not necessarily compute values in the order the mapping would suggest, so production and consumption events do not always follow each other. You can also access this command using the toolbar button. For more information about debugging mode, see Debugger[771].

17.10 View

This topic lists all the menu commands available in the **View** menu.

⊟ Show Annotations

Displays annotations in the component. You can also enable this option by clicking the toolbar button. If the **Show Types** icon is also active, both sets of information are shown in grid form (*see screenshot below*). You can also use annotations to label connections. For details, see Connection Settings 59.



= F1060	
type	string
ann.	Revision identifier

⊟ Show Types

Displays data types in a component. You can also enable this option by clicking the toolbar button. If the **Show Annotations** icon is also active, then both sets of information are shown in grid form (*see Show Annotations above*).

⊟ Show Library in Function Header

Displays the library name in the function's header. You can also enable this option by clicking the toolbar button.

⊟ Show Tips

When you place the cursor over a function's header, you will see a tooltip summarizing what this function does. With the **Show Tips** option enabled, you can also see information about datatypes in a component.

⊟ XBRL Display Options (*Enterprise Edition*)

MapForce enables you to configure the following XBRL settings:

- The label language of XBRL items and their annotations
- The preferred label roles for XBRL item names
- The specific type of label roles of annotations for XBRL items
- Custom XBRL Taxonomy Packages

⊟ Show Selected Components Connectors/Connections from Source to Target

These options allow you to highlight connections selectively. To find out how these options work, see Connections 50.

⊟ Zoom

Opens the **Zoom** dialog box. You can enter the zoom factor numerically or drag the slider to change the zoom factor interactively.

Back/Forward

The **Back** and **Forward** commands allow you to switch between the previous and next mappings you have been working on, relative to the currently open mapping.

Status Bar

Switches on/off the **Status Bar** visible below the **Messages** window.

Libraries/Manage Libraries

Click **Libraries** to switch on/off the **Libraries** window. Click **Manage Libraries** to switch on/off the **Manage Libraries** window.

Messages

Switches on/off the Messages window[26]. When code is generated, the **Messages** window is automatically activated to show the validation result.

Overview

Switches on/off the Overview window[26]. Drag the rectangle to navigate your way through the mapping.

Project Window (*Professional and Enterprise editions*)

Switches on/off the **Project** window. To find out more about projects, see Projects[95].

Debug Windows (*Professional and Enterprise editions*)

The debug mode enables you to analyze the context in which a particular value is produced. This information is available directly in the mapping and in the **Values**, **Context**, and **Breakpoints** windows. For more information, see About Debug Mode[775].

17.11 Tools

This topic lists all the menu commands available in the **Tools** menu.

Global Resources

Opens the **Manage Global Resources** dialog box that enables you to add, edit, and delete settings applicable across multiple Altova applications. For more information, see Altova Global Resources[815].

Active Configuration

Allows you to select the currently active global resource configuration from a list of configurations. To create and configure different types of global resources, see Altova Global Resources[815].

Create Reversed Mapping

Creates a reversed mapping from the currently active mapping, which means the source component becomes the target component, and the target component becomes the source. Note that only direct connections between components are retained in the reversed mapping. It is likely that the new mapping will not be valid or suitable for preview in the **Output** pane. Therefore, the new mapping would require manual editing.

The following data is retained:

- Direct connections between components
- Direct connections between components in a chained mapping
- The type of connection[51]: Standard, Mixed-Content, Copy-All
- Pass-through component settings
- Database components (*Professional and Enterprise editions*)

The following data is *not* retained:

- Connections via functions, filters, etc.
- User-defined functions
- Web service components (*Enterprise Edition*)

XBRL Taxonomy Manager (*Enterprise Edition*)

XBRL Taxonomy Manager is a tool that allows you to install and manage XBRL taxonomies.

XML Schema Manager

XML Schema Manager is an Altova tool that provides a centralized way to install and manage XML schemas (DTDs for XML and XML Schemas) for use across all Altova's XBRL-enabled applications. For more information, see Schema Manager[149].

Customize

This option allows you to customize the MapForce graphical user interface. This includes showing/hiding toolbars as well as customizing menus[1074] and keyboard shortcuts[1075].

Restore Toolbars and Windows

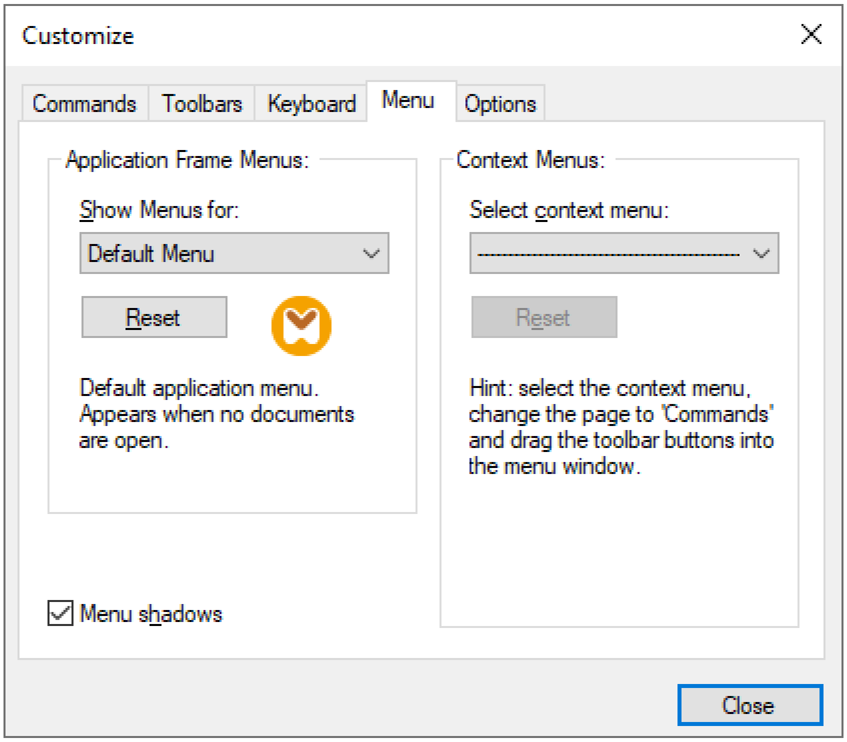
Resets the toolbars, entry helper windows, docked windows etc. to their defaults. You need to restart MapForce so that the changes take effect.

⊟ Options

Opens the **Options** dialog box that enables you to change the default MapForce settings. For more information, see Options 1077.

17.11.1 Customize Menus

You can customize standard MapForce menus and context menus (e.g., to add, change, or remove commands). You can also revert your changes to the default state (**Reset**). To customize menus, go to **Tools | Customize** and click the **Menu** tab (*see screenshot below*).



Default application menu. Appears when no documents are open.

Hint: select the context menu, change the page to 'Commands' and drag the toolbar buttons into the menu window.

☑ Menu shadows

Default Menu vs. MapForce Design

The *Default Menu* bar is displayed when no document is open in the main window. The *MapForce Design* menu bar is displayed when one or more mappings are open. Each menu bar can be customized separately. Customization changes made to one menu bar do not affect the other.

To customize a menu bar, select it from the *Show Menus For* drop-down list. Then click the **Commands** tab and drag commands from the *Commands* list box to the menu bar or into any of the menus.

Delete commands from menus

To delete an entire menu or a command inside a menu, do the following:

1. Select *Default Menu* or *MapForce Design* from the *Show Menus for* drop-down list.
2. With the **Customize** dialog open, select a toolbar command you want to delete or a command you want to delete from one of the menus.
3. Drag the toolbar command from the toolbar or the command from the menu. Alternatively, right-click the toolbar command or menu command and select **Delete**.

You can reset any menu bar to its default state by selecting it from the *Show Menus For* drop-down list and clicking the **Reset** button.

Customize context menus
Context menus appear when you right-click certain objects in the application's interface. Each of these context menus can be customized in the following way:

1. Select a context menu from the *Select context menu* drop-down list. This opens the respective context menu.
2. Open the **Commands** tab and drag a command from the *Commands* list box into the context menu.
3. To delete a command from the context menu, right-click that command and select **Delete**. Alternatively, drag the command out of the context menu.

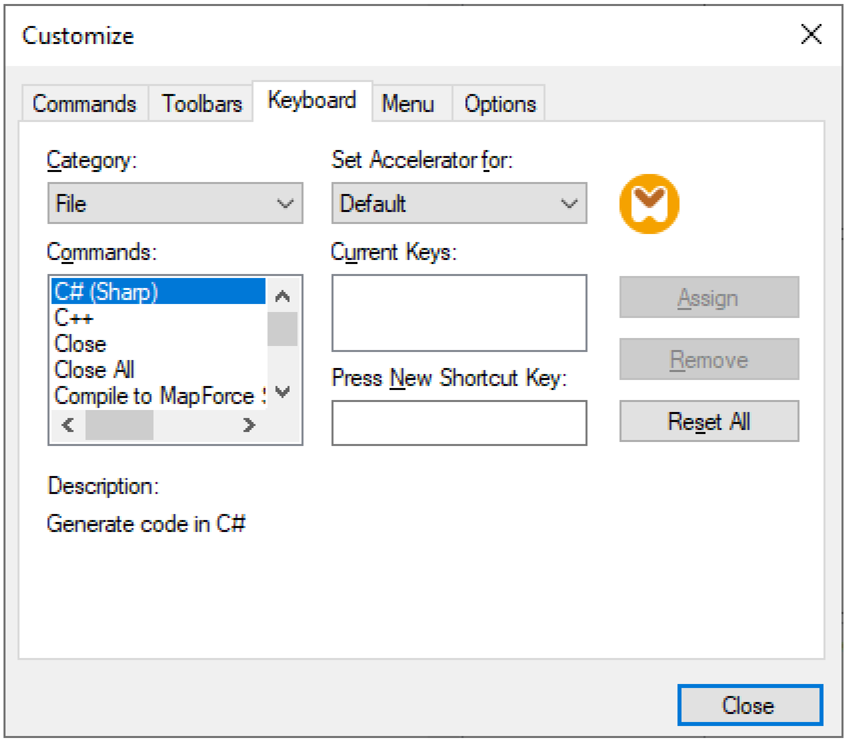
You can reset any context menu to its default state by selecting it in the *Select context menu* drop-down list and clicking the **Reset** button.

Menu shadows
Select the *Menu shadows* check box to give all menus shadows.

17.11.2 Customize Shortcuts

You can define or change keyboard shortcuts in MapForce as follows: Select the **Tools | Customize** and click the **Keyboard** tab. To assign a new shortcut to a command, take the following steps:

1. Select the **Tools | Customize** command and click the **Keyboard** tab (*see screenshot below*).
2. Click the *Category* combo box to select the menu name.
3. In the *Commands* list box, select the command you want to assign a new shortcut to.
4. Type in new shortcut keys in the *Press New Shortcut Key* text box and click **Assign**.



Category: File

Set Accelerator for: Default

Commands: C# (Sharp)

Current Keys:

Press New Shortcut Key:

Description:

Generate code in C#

To clear the entry in the *Press New Shortcut Key* text box, press any of the control keys: **Ctrl**, **Alt** or **Shift**. To delete a shortcut, click the shortcut you want to delete in the *Current Keys* list box and click **Remove**.

Note: The *Set accelerator for* does not currently have any function.

Keyboard shortcuts

By default, MapForce provides the following keyboard shortcuts:

F1	Help Menu
F2	Next bookmark (in output window)
F3	Find Next
F10	Activate menu bar
Num +	Expand current item node
Num -	Collapse item node
Num *	Expand all from current item node
CTRL + TAB	Switches between open mappings
CTRL + F6	Cycle through open windows
CTRL + F4	Closes the active mapping document
Alt + F4	Closes MapForce
Alt + F, F, 1	Opens the last file
Alt + F, T, 1	Opens the last project
CTRL + N	File New
CTRL + O	File Open
CTRL + S	File Save

CTRL + P	File Print
CTRL + A	Select All
CTRL + X	Cut
CTRL + C	Copy
CTRL + V	Paste
CTRL + Z	Undo
CTRL + Y	Redo
Del	Delete component (with prompt)
Shift + Del	Delete component (no prompt)
CTRL + F	Find
F3	Find Next
Shift + F3	Find Previous
Arrow keys	
(up / down)	Select next item of component
Esc	Abandon edits/close dialog box
Return	Confirms a selection
Output window hotkeys	
CTRL + F2	Insert/Remove Bookmark
F2	Next Bookmark
SHIFT + F2	Previous Bookmark
CTRL + SHIFT + F2	Remove All Bookmarks
Zooming hotkeys	
CTRL + mouse wheel forward	Zoom In
CTRL + mouse wheel back	Zoom Out
CTRL + 0 (Zero)	Reset Zoom

17.11.3 Options

You can change general and other preferences in MapForce by selecting the **Tools | Options** command. The available options are described below.

⊟ General

In the *General* section, you can define the following options:

- *Show logo | Show on start:* Shows or hides an image (splash screen) when MapForce starts.

- The *Mapping view* section allows you to set the following parameters:

  - You can enable/disable the gradient background in the Mapping pane (*Show gradient background*).
  - You can limit the display of annotations to N lines (*Limit annotation display*). For example, if you have set this option to 2, and your annotation text contains 3 lines, only the first two lines of the annotation text will be displayed in the mapping. This setting also applies to SELECT statements visible in a component.

  - You can also limit the display of a component's comment 34 to N lines (*Limit comment display*). For example, if you have limited the comment display to 1 line, and you comment contains more than one line, the comment box will display only the first line. Setting the property to `0` will suppress the display of component comments completely. Note that the *Limit comment display* option has no effect on comment components 36.

- *Default encoding for new components.*

  Encoding name: The default encoding for new XML files can be set by selecting an option from the dropdown list. If a two- or four-byte encoding is selected as the default encoding (i.e. UTF-16, UCS-2, or UCS-4), you can also choose between little-endian and big-endian byte-ordering. This setting can also be changed individually for each component (see Change Component Settings 41).

  Byte order: When a document with two-byte or four-byte character encoding is saved, the document can be saved either with little-endian or big-endian byte-ordering. You can also specify whether a byte order mark should be included.

- *Preview Settings:* The *Use execution timeout* option sets an execution timeout when you preview the mapping result in the **Output** pane.

- *On activating Output pane:* You can generate output to temporary files or write the output directly to an output file (*see below*).

  Generate output to temporary files: This is the default option. If the output file path contains folders that do not exist yet, MapForce will create these folders. For Professional and Enterprise editions: If you intend to deploy the mapping to a server for execution, any directories in the path must exist on the server; otherwise, an execution error will occur. See also Preparing Mappings for Server Execution 794.

  Write directly to final output files: If the output file path contains folders that do not exist yet, an error will occur. This option overwrites any existing output files without requesting further confirmation.

- *Display text in steps of N million characters:* Specifies the maximum size of the text displayed in the **Output** pane when you preview mappings that generate large XML and text files. If the output text exceeds this value, you will need to click the **Load more** button to load the next chunk. For more information, see Preview and validate output 64.

Editing

In the *Editing* section, you can define mapping view options:

- *Align components on mouse dragging:* Specify whether components or functions should be aligned with other components, while you drag them in the Mapping window. For more information, see Align Components 41.

- *Smart component deletion:* MapForce allows you to keep connections even after deleting some transformation components 34. Keeping connections might be particularly useful with multiple child connections, because you will not have to restore every single child connection manually after deleting a transformation component. For details, see Keep Connections after Deleting Components 62.

Messages

The *Messages* section allows you to switch on message notifications such as suggesting connecting ancestor items, informing about the creation of multiple target components, and so on.

Generation (*Professional and Enterprise editions*)

The *Generation* section allows you to define settings for program-code generation and MapForce Server Execution files. For more information, see Code Generation 66, Generation 1080, and Compiling Mappings to MapForce Server Execution Files 799.

Java

You may need to add a custom Java VM path, for example, if you are using a Java virtual machine which does not have an installer and does not create registry entries (e.g., Oracle's OpenJDK). You might also want to set this path if you need to override any Java VM path detected automatically by MapForce. For details, see Java 1082.

XBRL (*Enterprise Edition*)

MapForce enables you to configure the following general (application-wide) XBRL settings:

- The label language of XBRL items and their annotations
- The preferred label roles for XBRL item names
- The specific type of label roles of annotations for XBRL items
- Custom XBRL Taxonomy Packages

Debugger (*Professional and Enterprise editions*)

In the *Debugger* section, you can define the following debugging settings:

- *Maximum storage length of values:* Defines the string length of values displayed in the **Values** window (at least 15 characters). Note that setting the storage length to a high value may deplete available system memory.

- *Keep full trace history:* Instructs MapForce to keep the history of all values processed by all connectors of all components in the mapping for the duration of debugging. If this option is enabled, all values processed since the beginning of debug execution will be stored in memory and available for your analysis in the **Values** window until you stop debugging. It is not recommended to enable this option if you are debugging data-intensive mappings, since it may slow down debugging execution and deplete available system memory. If this option is disabled, MapForce keeps only the most recent trace history for nodes related to the current execution position.

Database (*Professional and Enterprise editions*)

In the *Database* section, you can define database query settings. For details, see Database Query Settings 1083.

Network Proxy

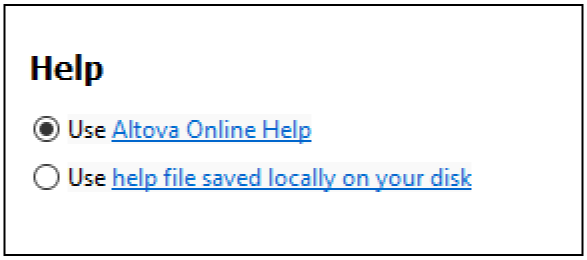
The *Network Proxy* section enables you to configure custom proxy settings. These settings affect how the application connects to the Internet. By default, the application uses the system's proxy settings, so you should not need to change the proxy settings in most cases. For more details, see Network Proxy 1088.

⊟ Help

MapForce provides Help (the user manual) in two formats:

- Online Help, in HTML format, which is available at the Altova website. In order to access the Online Help you will need Internet access.
- A Help file in PDF format, which is installed on your machine when you install MapForce. It is named `MapForce.pdf` and is located in the application folder (in the Program Files folder). If you do not have Internet access, you can always open this locally saved Help fie.

The Help option (*screenshot below*) enables you to select which of the two formats is opened when you click the **Help (F1)** command in the **Help** menu.



Help

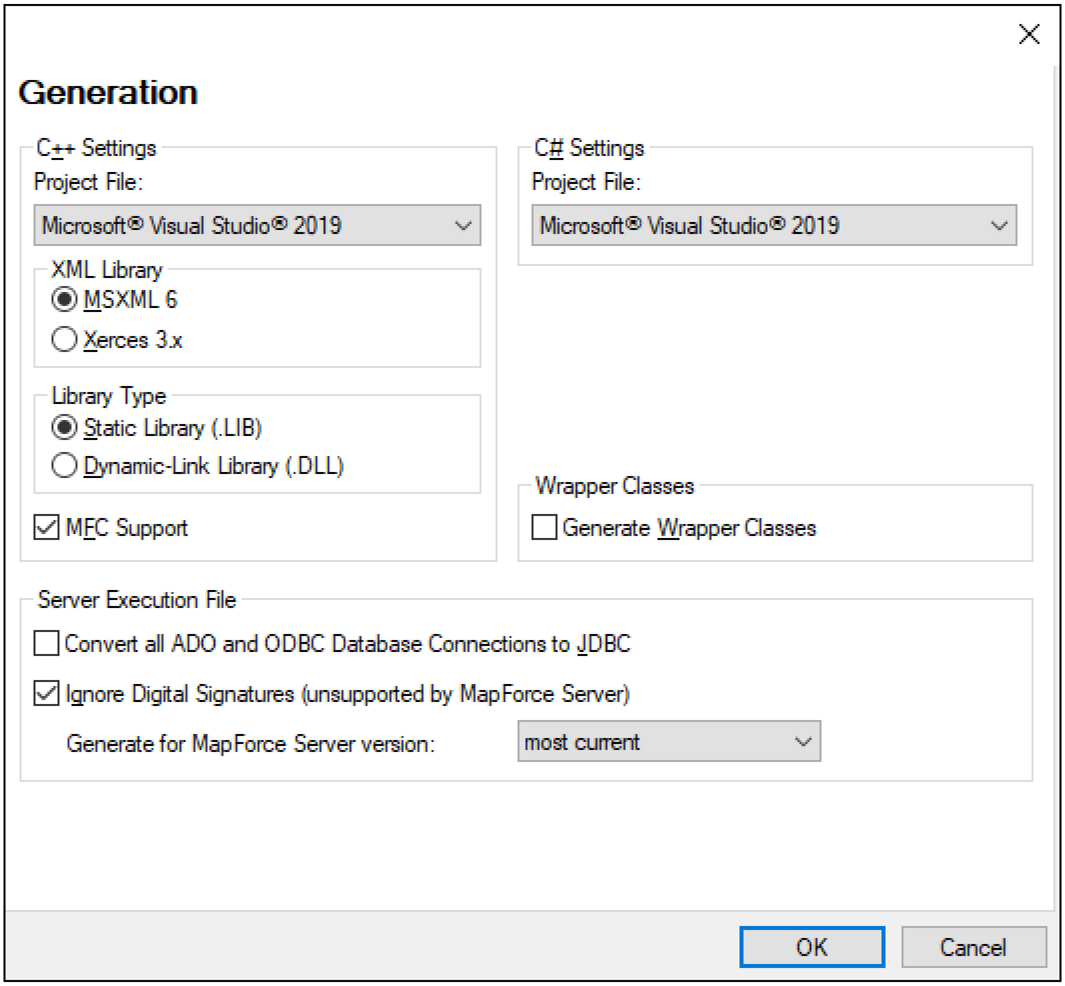
● Use Altova Online Help

○ Use help file saved locally on your disk

You can change this option at any time for the new selection to take effect. The links in this section (*see screenshot above*) open the respective Help format.

17.11.3.1 Generation

The *Generation* section of the **Options** dialog enables you to configure various code-generation and server-execution settings (*see below*).



Generation

C++ Settings
Project File: Microsoft® Visual Studio® 2019

XML Library
● MSXML 6
○ Xerces 3.x

Library Type
● Static Library (.LIB)
○ Dynamic-Link Library (.DLL)

☑ MFC Support

C# Settings
Project File: Microsoft® Visual Studio® 2019

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 described below.

⊟ C++ Settings

This section define the following compiler settings for the C++ environment:

- The Visual Studio version (2013, 2015, 2017, 2019, 2022)
- The XML library (MSXML, Xerces 3.x)
- Whether static or dynamic libraries must be generated
- Whether code must be generated with or without MFC support

⊟ C# Settings

If you need to target a specific platform, select the relevant *Microsoft .NET <version>* option from the drop-down list. If you need to target the .NET Framework platform for a specific Visual Studio version, select the relevant *Microsoft Visual Studio <version>* option.

⊟ Wrapper Classes

Allows you to generate wrapper classes for XML schemas. For details, see Generating Code from XML

Schemas or DTDs 885.

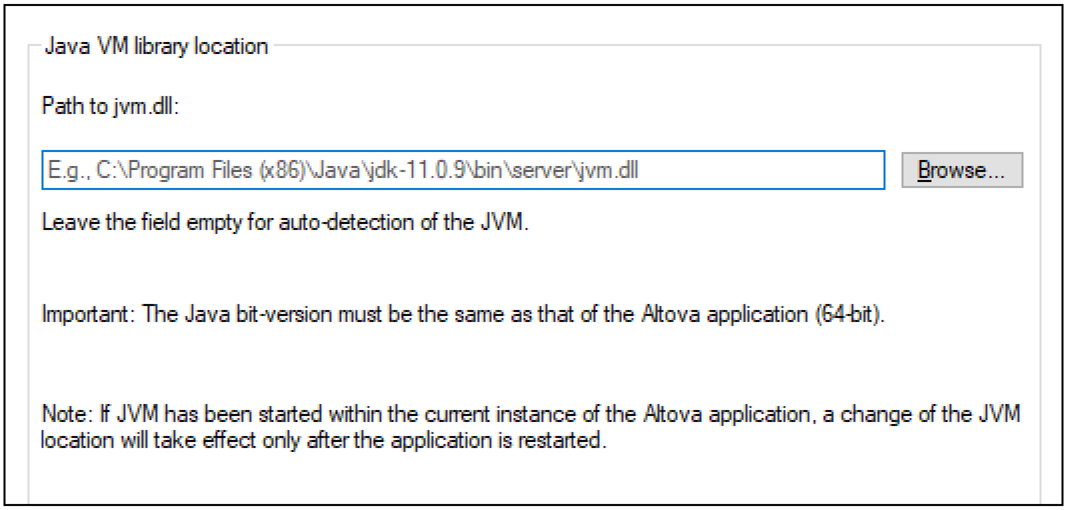
⊟ Server Execution File

These options are relevant when you compile mappings to MapForce Server execution files. For more information, see Compiling Mappings to MapForce Server Execution Files 799.

17.11.3.2 Java

In the *Java* section (*see screenshot below*), you can optionally enter the path to a Java VM (Virtual Machine) on your file system. Note that adding a custom Java VM path is not always necessary. By default, MapForce attempts to detect the Java VM path automatically by reading (in this order) the Windows registry and the JAVA_HOME environment variable. The custom path added in this dialog box will take priority over any other Java VM path detected automatically.

You may need to add a custom Java VM path, for example, if you are using a Java virtual machine which does not have an installer and does not create registry entries (e.g., Oracle's OpenJDK). You might also want to set this path if you need to override, for whatever reason, any Java VM path detected automatically by MapForce.



Java VM library location

Path to jvm.dll:

E.g., C:\Program Files (x86)\Java\jdk-11.0.9\bin\server\jvm.dll

Leave the field empty for auto-detection of the JVM.

Important: The Java bit-version must be the same as that of the Altova application (64-bit).

Note: If JVM has been started within the current instance of the Altova application, a change of the JVM location will take effect only after the application is restarted.

Note the following:

- The Java VM path is shared between Altova desktop (not server) applications. Consequently, if you change it in one application, it will automatically apply to all other Altova applications.
- The path must point to the `jvm.dll` file from the `\bin\server` or `\bin\client` directory, relative to the directory where the JDK was installed.
- The MapForce platform (32-bit, 64-bit) must be the same as that of the JDK.
- After changing the Java VM path, you may need to restart MapForce for the new settings to take effect.

Changing the Java VM path affects the following areas:

- JDBC connectivity

- Java extension functions for XSLT/XPath

This setting does not affect Java code generation.

17.11.3.3 Database

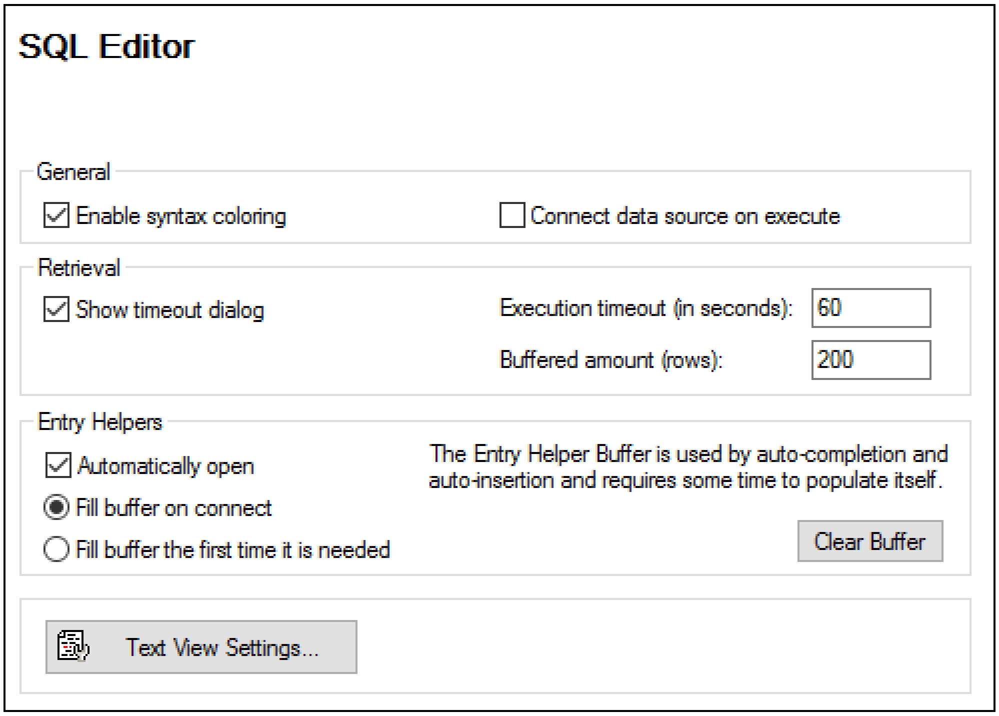
This section explains how to configure various SQL editing settings. You can access the settings in one of the following ways:

- By opening the **DB Query** pane and clicking (**Options**).
- By selecting **Tools | Options | Database** and then the relevant section.

In the *Database* section, you can define general SQL editing settings, encoding and result view options, SQL generation parameters, and fonts. For details, see the subsections below.

SQL Editor

The *SQL Editor* section allows you to change general SQL editing settings (*see screenshot below*). The available settings are described below.



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.* To see different elements of SQL syntax in distinct colors, enable syntax coloring. Select the *Connect data source on execute* check box to connect to the relevant data source automatically whenever an SQL statement is executed.

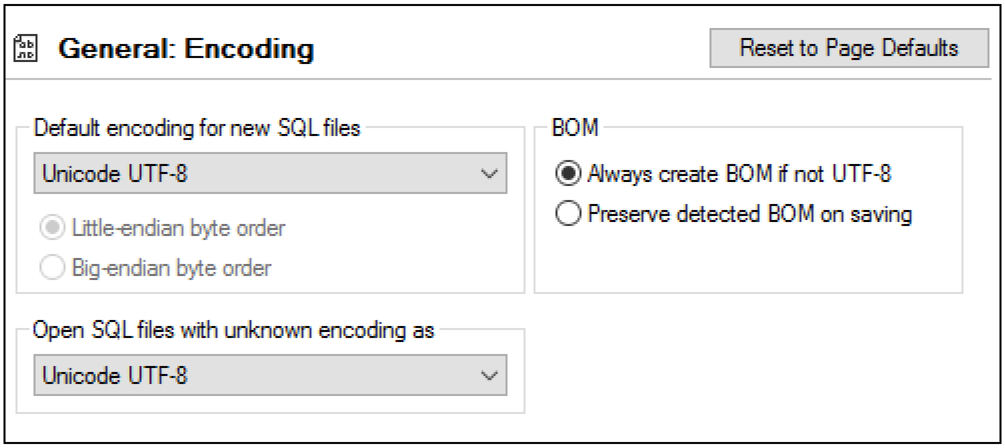
- *Retrieval.* Activating the *Show timeout dialog* check box allows you to change the timeout settings when the permissible execution period is exceeded. You can specify the maximum amount of time allowed for SQL execution (*Execution timeout*) in seconds. You can also define the number of rows that will be put into a buffer.

- *Entry Helpers.* To enable auto-completion suggestions as you start typing SQL statements, select the *Automatically open* check box (see also Auto-Completion 235). You can choose when to fill in the entry helper buffer: when you connect to a data source or when the buffer is used for the first time. Note that filling the buffer may take some time. Use the **Clear Buffer** button to reset the buffer.

- *Text View Settings:* Allows you to define different Text View settings such as margins, tabs, visual aids, auto-highlighting options, and Text View navigation hotkeys. For more information, see Text View Features 68.

Encoding

In the *Encoding* section, you can specify encoding options for SQL files created or opened with SQL Editor (*see screenshot below*).



General: Encoding

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 default encoding for new files so that each new document includes the same encoding. If a two- or four-byte encoding is selected as the default encoding (e.g., UTF-16, UCS-2, or UCS-4), you can also choose between little-endian and big-endian byte ordering for SQL files. The encoding of existing files is not affected by this setting.

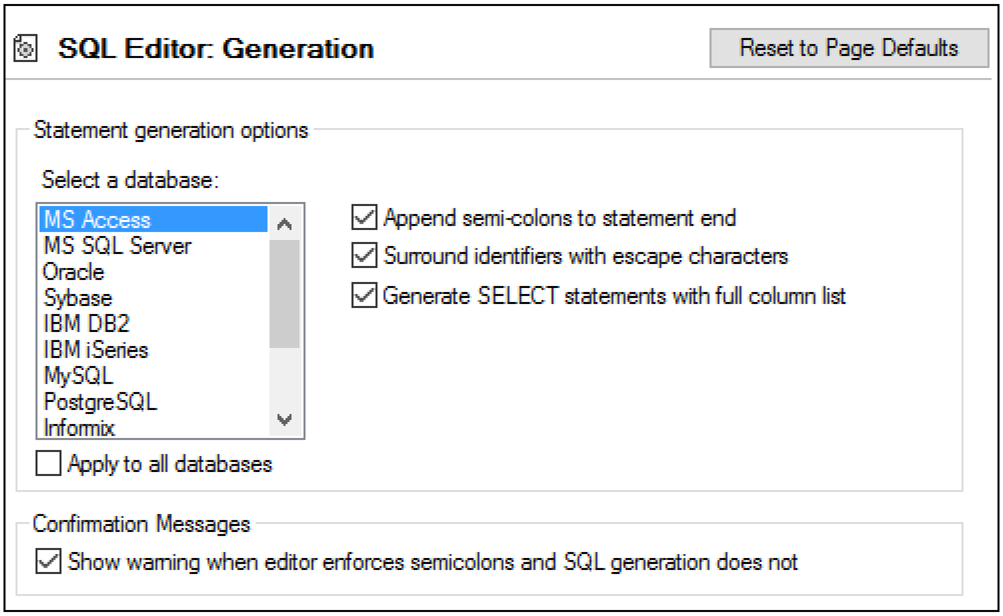
- *Open SQL files with unknown encoding as:* You can select the encoding with which to open an SQL file with unknown encoding.

- *BOM:* Choose whether to create the byte order mark (BOM) character for encoding other than UTF-8 or whether to keep the detected BOM.

Note: SQL files which have no encoding specification are saved with UTF-8 encoding.

SQL Generation

The *Generation* section enables you to specify SQL statement generation syntax for various database kinds (*see screenshot below*).



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

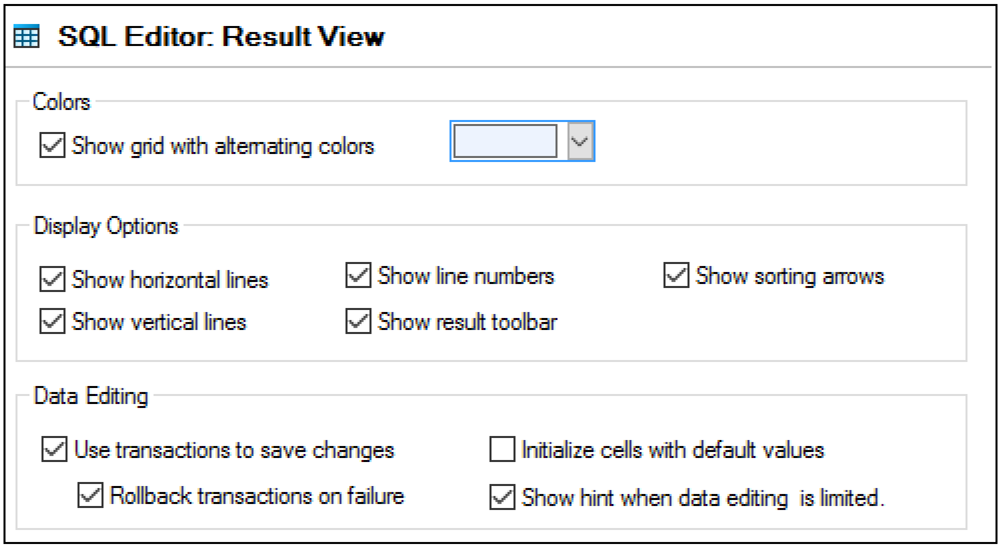
Confirmation Messages

☑ 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 options to the right. You can also choose to apply the same set of settings to all databases (the *Apply to all databases* check box). Note that using common settings for all databases may prevent you from editing data in Oracle, IBM DB2, and iSeries databases via a JDBC connection.

Result View

You can configure the appearance of the **Results** tab of the **DB Query** pane in the *Result View* section (*see screenshot below*).



SQL Editor: Result View

Colors

☑ Show grid with alternating colors

Display Options

☑ Show horizontal lines
☑ Show vertical lines
☑ Show line numbers
☑ Show result toolbar
☑ Show sorting arrows

Data Editing

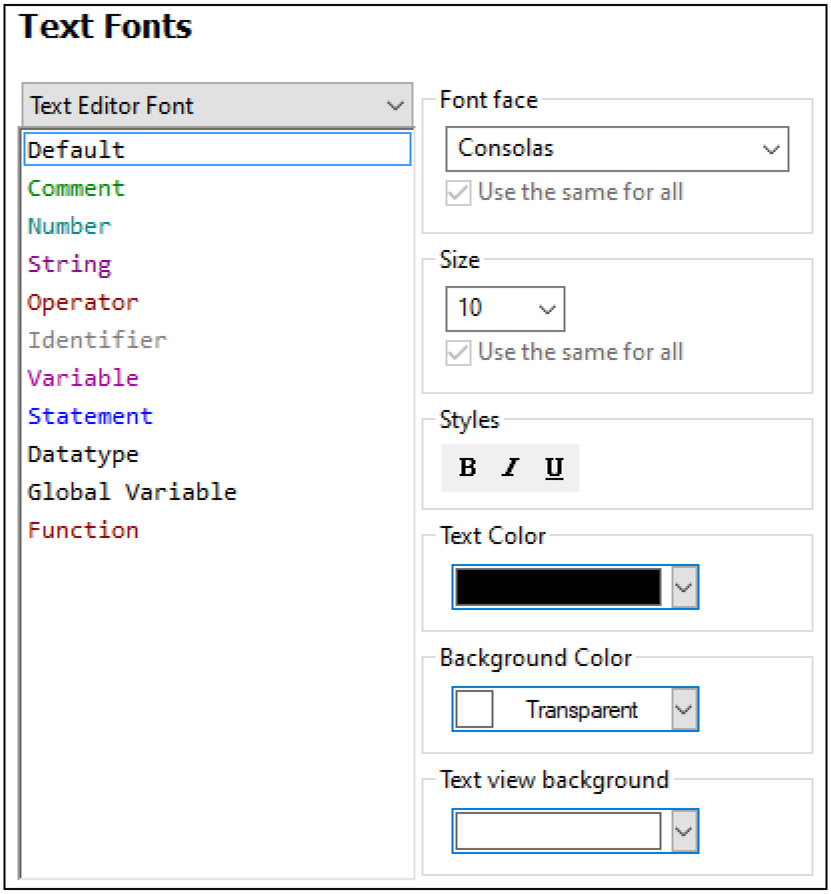
☑ Use transactions to save changes
☑ Rollback transactions on failure
□ Initialize cells with default values
☑ Show hint when data editing is limited.

Select the *Show grid with alternating colors* check box to display rows in **Result** tabs as a simple grid or a grid with alternating white and colored rows. The alternating color is configurable. The *Display Options* group allows you to define how to display horizontal and vertical grid lines, line numbers, and the **Result** toolbar.

The *Data Editing* group allows you to define transaction settings if the cells are to be filled with default values, and if a hint is to be displayed when data editing is limited.

Text Fonts

In the *Text Fonts* section, you can configure color and font settings of SQL statements (*see screenshot below*). You can choose the common font face, style, size, and syntax coloring of various text types that appear in SQL Editor. Note that the same font and size are used for all text types. Only the style can be changed for individual text types. If you access the **Text Fonts** dialog box from the **DB Query** pane, you can click the **Reset to Page Defaults** button to restore the original settings.



Text Fonts

Text Editor Font

- Default
- Comment
- Number
- String
- Operator
- Identifier
- Variable
- Statement
- Datatype
- Global Variable
- Function

Font face: Consolas

☑ Use the same for all

Size: 10

☑ Use the same for all

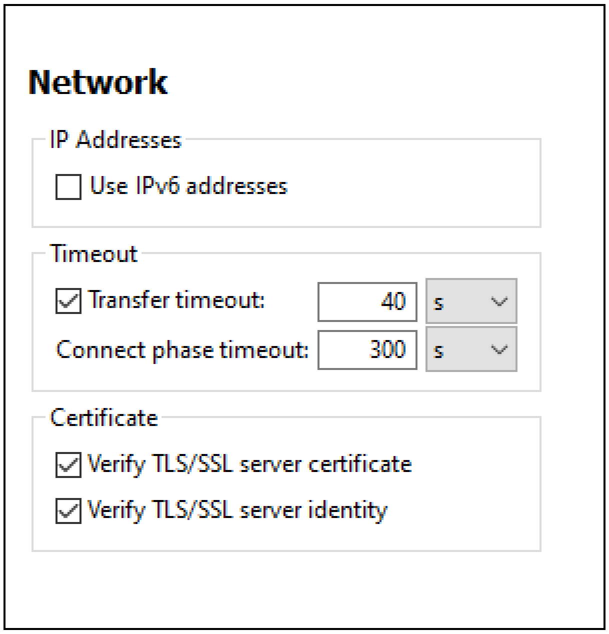
Text Color: ■

Background Color: Transparent

Text view background

17.11.3.4 Network

The **Network** section (*screenshot below*) enables you to configure important network settings.



Network

IP Addresses

□ Use IPv6 addresses

Timeout

☑ Transfer timeout: 40 s

Connect phase timeout: 300 s

Certificate

☑ Verify TLS/SSL server certificate

☑ Verify TLS/SSL server identity

IP addresses
When host names resolve to more than one address in mixed IPv4/IPv6 networks, selecting this option causes the IPv6 addresses to be used. If the option is not selected in such environments and IPv4 addresses are available, then IPv4 addresses are used.

Timeout

- *Transfer timeout:* If this limit is reached for the transfer of any two consecutive data packages of a transfer (sent or received), then the entire transfer is aborted. Values can be specified in seconds [s] or milliseconds [ms], with the default being 40 seconds. If the option is not selected, then there is no time limit for aborting a transfer.
- *Connection phase timeout:* This is the time limit within which the connection has to be established, including the time taken for security handshakes. Values can be specified in seconds [s] or milliseconds [ms], with the default being 300 seconds. This timeout cannot be disabled.

Certificate

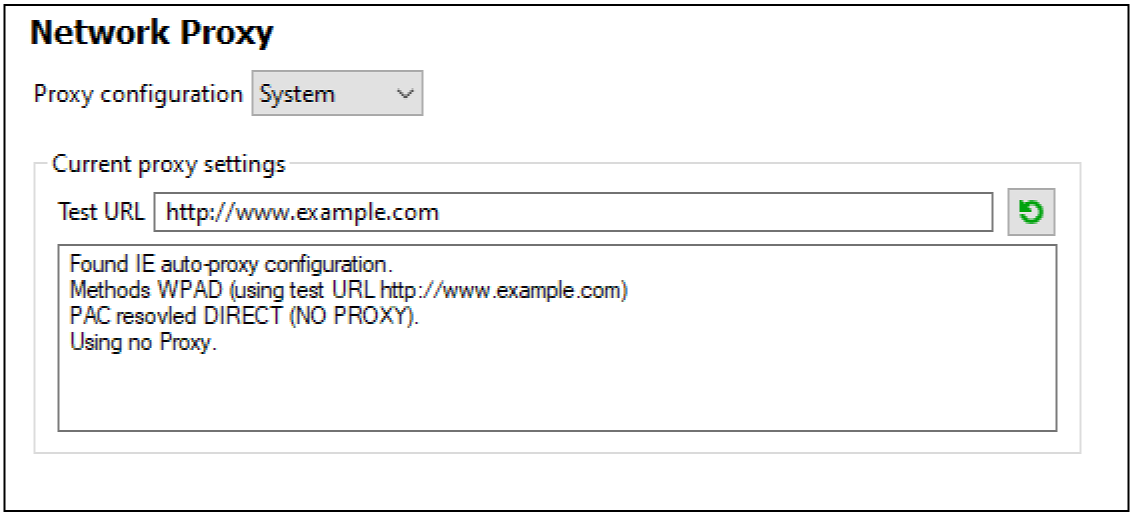
- *Verify TLS/SSL server certificate:* If selected, then the authenticity of the server's certificate is checked by verifying the chain of digital signatures until a trusted root certificate is reached. This option is enabled by default. If this option is not selected, then the communication is insecure, and attacks (for example, a man-in-the-middle attack) would not be detected. Note that this option does not verify that the certificate is actually for the server that is communicated with. To enable full security, both the certificate and the identity must be checked (*see next option*).
- *Verify TLS/SSL server identity:* If selected, then the server's certificate is verified to belong to the server we intend to communicate with. This is done by checking that the server name in the URL is the same as the name in the certificate. This option is enabled by default. If this option is not selected, then the server's identify is not checked. Note that this option does not enable verification of the server's certificate. To enable full security, both the certificate as well as the identity must be checked (*see*

previous option).

17.11.3.5 Network Proxy

The *Network Proxy* section enables you to configure custom proxy settings. These settings affect how the application connects to the Internet (for XML validation purposes, for example). By default, the application uses the system's proxy settings, so you should not need to change the proxy settings in most cases. If necessary, however, you can set an alternative network proxy by selecting, in the *Proxy Configuration* combo box, either *Automatic* or *Manual* to configure the settings accordingly.

Note: The network proxy settings are shared among all Altova MissionKit applications. So, if you change the settings in one application, all MissionKit applications will be affected.


Network Proxy

Proxy configuration: System

Current proxy settings

Test URL: http://www.example.com

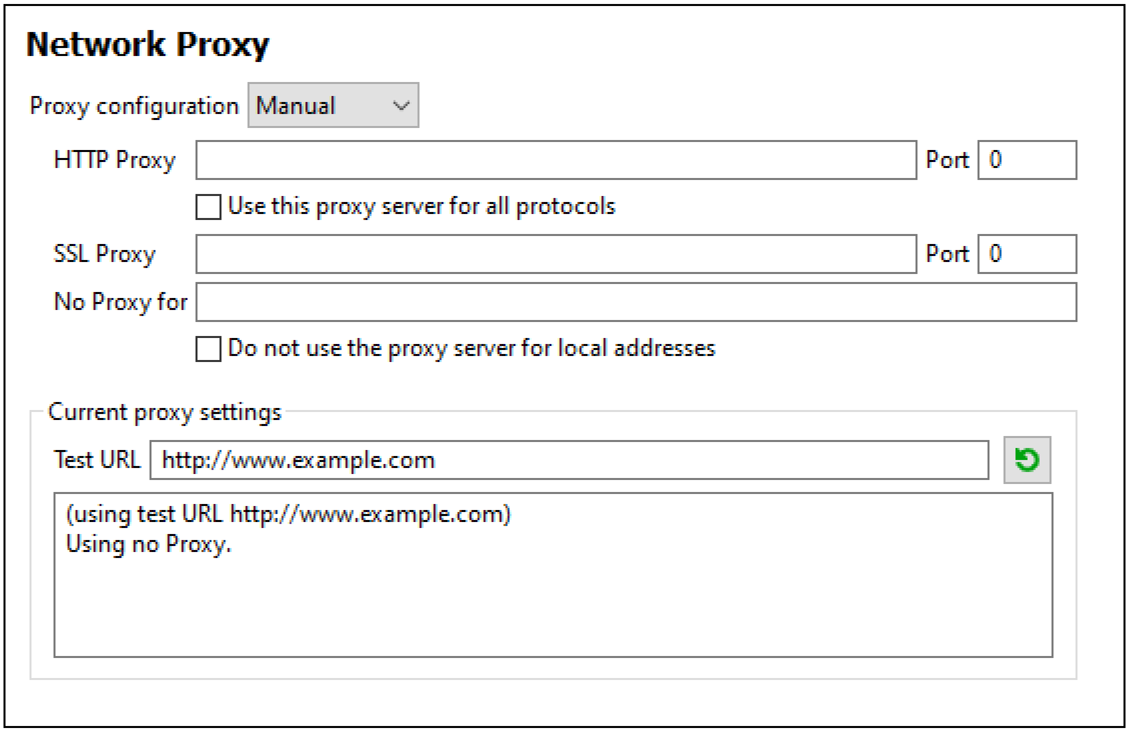
```
Found IE auto-proxy configuration.
Methods WPAD (using test URL http://www.example.com)
PAC resovled DIRECT (NO PROXY).
Using no Proxy.
```

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.



Network Proxy

Proxy configuration: Manual

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

```
(using test URL http://www.example.com)
Using no Proxy.
```

The following options are provided:

- *HTTP Proxy:* Uses the specified host name and port for the HTTP protocol. If *Use this proxy server for all protocols* is selected, then the specified HTTP proxy is used for all protocols.
- *SSL Proxy:* Uses the specified host name and port for the SSL protocol.
- *No Proxy for:* A semi-colon (`;`) separated list of fully qualified host names, domain names, or IP addresses for hosts that should be used without a proxy. IP addresses may not be truncated and IPv6 addresses have to be enclosed by square brackets (for example: `[2606:2800:220:1:248:1893:25c8:1946]`). Domain names must start with a leading dot (for example: `.example.com`).
- *Do not use the proxy server for local addresses:* If checked, adds `<local>` to the *No Proxy for* list. If this option is selected, then the following will not use the proxy: (i) `127.0.0.1`, (ii) `[::1]`, (iii) all host names not containing a dot character (`.`).

Note: If a proxy server has been set and you want to deploy a mapping to Altova FlowForce Server, you must select the option *Do not use the proxy server for local addresses*.

Current proxy settings

Provides a verbose log of the proxy detection. It can be refreshed with the **Refresh** button to the right of the *Test URL* field (for example, when changing the test URL, or when the proxy settings have been changed).

- *Test URL:* A test URL can be used to see which proxy is used for that specific URL. No I/O is done with this URL. This field must not be empty if proxy-auto-configuration is used (either through *Use system proxy settings* or *Authomatic proxy configuration*).

17.12 Window

This topic lists all the menu commands available in the **Window** menu.

Cascade

This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

Tile Horizontal

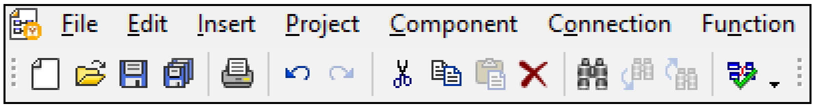
This command rearranges all open document windows as horizontal tiles, making them all visible at the same time.

Tile Vertical

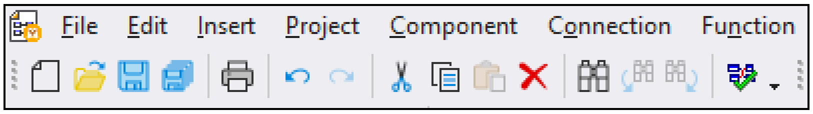
This command rearranges all open document windows as vertical tiles, making them all visible at the same time.

Classic/Light/Dark Theme

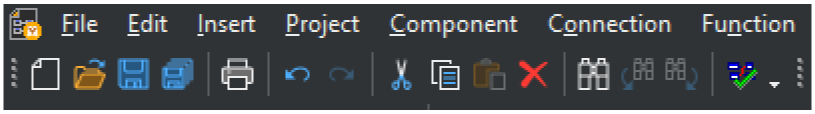
MapForce allows you to choose from the following themes: *Classic*, *Light*, and *Dark*. The examples of these themes are illustrated in the screenshots below. The default option is the Classic theme.



Classic Theme



Light Theme



Dark Theme

1 <MappingName>

Refers to the first open mapping design. If there are more mappings opened at the same time, they will be listed in the context menu, too.

Windows

This list shows all currently open windows and enables you to quickly switch between them. You can also use the **Ctrl-TAB** or **CTRL F6** keyboard shortcuts to switch between the open windows.

17.13 Help

This topic lists all the menu commands available in the **Help** menu.

⊟ Help (F1)

The **Help (F1)** command opens the application's Help documentation (its user manual). By default, the Online Help in HTML format at the Altova website will be opened.

If you do not have Internet access or do not want, for some other reason, to access the Online Help, you can use the locally stored version of the user manual. The local version is a PDF file named **`MapForce.pdf`** that is stored in the application folder (in the Program Files folder).

If you want to change the default format to open (Online Help or local PDF), do this in the Help section of the Options dialog (menu command **Tools | Options**).

⊟ Software Activation

License your product
After you download your Altova product software, you can license—or activate—it using either a free evaluation key or a purchased permanent license key.

- ***Free evaluation license.*** When you first start the software after downloading and installing it, the **Software Activation** dialog will pop up. In it is a button to request a free evaluation license. Click it to get your license. When you click this button, your machine-ID will be hashed and sent to Altova via HTTPS. The license information will be sent back to the machine via an HTTP response. If the license is created successfully, a dialog to this effect will appear in your Altova application. On clicking **OK** in this dialog, the software will be activated for a period of 30 days **on this particular machine**.

- ***Permanent license key.*** The **Software Activation** dialog allows you to purchase a permanent license key. Clicking this button takes you to Altova's online shop, where you can purchase a permanent license key for your product. Your license will be sent to you by e-mail in the form of a license file, which contains your license-data.

  There are three types of permanent license: *installed*, *concurrent user*, and *named user*. An installed license unlocks the software on a single computer. If you buy an installed license for *N* computers, then the license allows use of the software on up to *N* computers. A concurrent-user license for *N* concurrent users allows *N* users to run the software concurrently. (The software may be installed on 10N computers.) A named-user license authorizes a specific user to use the software on up to 5 different computers. To activate your software, click **Upload a New License**, and, in the dialog that appears, enter the path to the license file, and click **OK**.

Note: For multi-user licenses, each user will be prompted to enter his or her own name.

> *Your license email and the different ways to license (activate) your Altova product*
> The license email that you receive from Altova will contain your license file as an attachment. The license file has a **`.altova_licenses`** file extension.
>
> To activate your Altova product, you can do one of the following:

> - Save the license file (**`.altova_licenses`**) to a suitable location, double-click the license file, enter any requested details in the dialog that appears, and finish by clicking **Apply Keys**.
> - Save the license file (**`.altova_licenses`**) to a suitable location. In your Altova product, select the menu command **Help | Software Activation**, and then **Upload a New License**. Browse for or enter the path to the license file, and click **OK**.
> - Save the license file (**`.altova_licenses`**) to any suitable location, and upload it from this location to the license pool of your Altova LicenseServer. You can then either: (i) acquire the license from your Altova product via the product's Software Activation dialog (*see below*) or (ii) assign the license to the product from Altova LicenseServer. *For more information about licensing via LicenseServer, read the rest of this topic.*

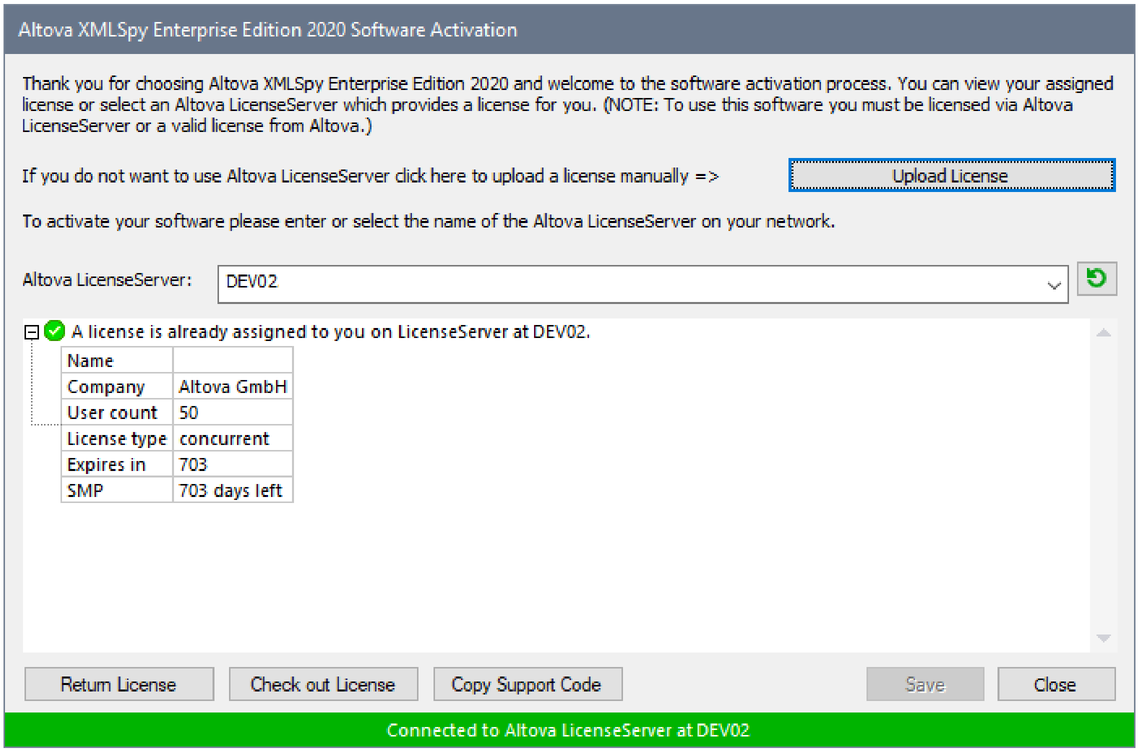
You can access the **Software Activation** dialog (*screenshot below*) at any time by clicking the **Help | Software Activation** command.

Activate your software

You can activate the software by registering the license in the Software Activation dialog or by licensing via Altova LicenseServer (*see details below*).

- *Registering the license in the Software Activation dialog*. In the dialog, click **Upload a New License** and browse for the license file. Click **OK** to confirm the path to the license file and to confirm any data you entered (your name in the case of multi-user licenses). Finish by clicking **Save**.

- *Licensing via Altova LicenseServer on your network:* To acquire a license via an Altova LicenseServer on your network, click **Use Altova LicenseServer**, located at the bottom of the **Software Activation** dialog. Select the machine on which the LicenseServer you want to use has been installed. Note that the auto-discovery of License Servers works by means of a broadcast sent out on the LAN. As these broadcasts are limited to a subnet, License Server must be on the same subnet as the client machine for auto-discovery to work. If auto-discovery does not work, then type in the name of the server. The Altova LicenseServer must have a license for your Altova product in its license pool. If a license is available in the LicenseServer pool, this is indicated in the **Software Activation** dialog (*see screenshot below showing the dialog in Altova XMLSpy*). Click **Save** to acquire the license.


Altova XMLSpy Enterprise Edition 2020 Software Activation

Thank you for choosing Altova XMLSpy Enterprise Edition 2020 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 from Altova.)

If you do not want to use Altova LicenseServer click here to upload a license manually =>

To activate your software please enter or select the name of the Altova LicenseServer on your network.

Altova LicenseServer: DEV02

A license is already assigned to you on LicenseServer at DEV02.

Name	
Company	Altova GmbH
User count	50
License type	concurrent
Expires in	703
SMP	703 days left

Connected to Altova LicenseServer at DEV02

After a machine-specific (aka installed) license has been acquired from LicenseServer, it cannot be returned to LicenseServer for a period of seven days. After that time, you can return the machine license to LicenseServer (click **Return License**) so that this license can be acquired from LicenseServer by another client. (A LicenseServer administrator, however, can unassign an acquired license at any time via the administrator's Web UI of LicenseServer.) Note that the returning of licenses applies only to machine-specific licenses, not to concurrent licenses.

Check out license

You can check out a license from the license pool for a period of up to 30 days so that the license is stored on the product machine. This enables you to work offline, which is useful, for example, if you wish to work in an environment where there is no access to your Altova LicenseServer (such as when your Altova product is installed on a laptop and you are traveling). While the license is checked out, LicenseServer displays the license as being in use, and the license cannot be used by any other machine. The license automatically reverts to the checked-in state when the check-out period ends. Alternatively, a checked-out license can be checked in at any time via the **Check in** button of the **Software Activation** dialog.

To check out a license, do the following: (i) In the **Software Activation** dialog, click **Check out License** (*see screenshot above*); (ii) In the **License Check-out** dialog that appears, select the check-out period you want and click **Check out**. The license will be checked out. After checking out a license, two things happen: (i) The **Software Activation** dialog will display the check-out information, including the time when the check-out period ends; (ii) The **Check out License** button in the dialog changes to a **Check In** button. You can check the license in again at any time by clicking **Check In**. Because the license automatically reverts to the checked-in status after the check-out period elapses, make sure that the check-out period you select adequately covers the period during which you will be working offline.

If the license being checked out is a Installed User license or Concurrent User license, then the license is checked out to the machine and is available to the user who checked out the license. If the license being checked out is a Named User license, then the license is checked out to the Windows account of the named user. License check-out will work for virtual machines, but not for virtual desktop (in a VDI). Note that, when a Named User license is checked out, the data to identify that license check-out is stored in the user's profile. For license check-out to work, the user's profile must be stored on the local machine that will be used for offline work. If the user's profile is stored at a non-local location (such as a file-share), then the checkout will be reported as invalid when the user tries to start the Altova application.

License check-ins must be to the same major version of the Altova product from which the license was checked out. So make sure to check in a license before you upgrade your Altova product to the next major version.

Note: For license check-outs to be possible, the check-out functionality must be enabled on LicenseServer. If this functionality has not been enabled, you will get an error message to this effect when you try to check out. In this event, contact your LicenseServer administrator.

Copy Support Code
Click **Copy Support Code** to copy license details to the clipboard. This is the data that you will need to provide when requesting support via the online support form.

Altova LicenseServer provides IT administrators with a real-time overview of all Altova licenses on a network, together with the details of each license as well as client assignments and client usage of licenses. The advantage of using LicenseServer therefore lies in administrative features it offers for large-volume Altova license management. Altova LicenseServer is available free of cost from the Altova website. For more information about Altova LicenseServer and licensing via Altova LicenseServer, see the Altova LicenseServer documentation.

⊟ Order Form

When you are ready to order a licensed version of the software product, you can use either the **Purchase a Permanent License Key** button in the **Software Activation** dialog (*see previous section*) or the **Order Form** command to proceed to the secure Altova Online Shop.

⊟ Registration

Opens the Altova Product Registration page in a tab of your browser. Registering your Altova software will help ensure that you are always kept up to date with the latest product information.

⊟ Check for Updates

Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly.

⊟ Support Center

A link to the Altova Support Center on the Internet. The Support Center provides FAQs, discussion forums where problems are discussed, and access to Altova's technical support staff.

⊟ Download Components and Free Tools

A link to Altova's Component Download Center on the Internet. From here you can download a variety of companion software to use with Altova products. Such software ranges from XSLT and XSL-FO processors to Application Server Platforms. The software available at the Component Download Center is typically free of charge.

MapForce on the Internet

A link to the Altova website on the Internet. You can learn more about MapForce, related technologies and products on the Altova website.

MapForce Training

A link to the Online Training page on the Altova website. Here you can select from online courses conducted by Altova's expert trainers.

About MapForce

Displays the splash window and version number of your product. If you are using the 64-bit version of MapForce, this is indicated with the suffix (x64) after the application name. There is no suffix for the 32-bit version.

18 Appendices

These appendices contain technical information about MapForce, its technical aspects and licensing. It also provides the list of key terms specific to MapForce and MapForce-related products. The section is organized into the following subsections:

- Support Notes 1097
- Engine Information 1101
- Technical Data 1202
- License Information 1204

18.1 Support Notes

MapForce® is a 32/64-bit Windows application that runs on the following operating systems:

- Windows 10, Windows 11
- Windows Server 2016 or newer

64-bit support is available for the Enterprise and Professional editions.

MapForce is optionally available as a plug-in to the following integrated development environments:

- Visual Studio 2012/2013/2015/2017/2019/2022. See MapForce Plug-in for Visual Studio 844.
- Eclipse 2024-12 (4.34), 2024-09 (4.33), 2024-06 (4.32), 2024-03 (4.31). See MapForce Plug-in for Eclipse 847.

MapForce can integrate with Microsoft Office products:

- It can map data to or from Access databases. For supported versions, see Databases and MapForce 165.
- It can generate mapping documentation in Word 2000 or later versions. See Generating and Customizing Mapping Documentation 761.

18.1.1 Supported Sources and Targets

When you change the transformation language of a MapForce mapping, certain features may not be supported for that specific language. The following table summarizes the compatibility of mapping formats and transformation languages in **MapForce Professional Edition**.

Remarks:

- *Built-in* means that you can execute the mapping by clicking the **Output** pane in MapForce or run it with MapForce Server.

Mapping format		XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
XML[1]		●	●	●	●	●	●	●	●
CSV and text						●	●	●	●
Binary files									●
Databases[2]	ADO					●	●		●
	ADO.NET						●		●
	JDBC							●	●

Mapping format		XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
	Native SQLite								●
	Native PostgreSQL								●
	ODBC					●	●	●	●

Footnotes:

1. XML with digital signatures processing is supported only by MapForce Enterprise Edition using BUILT-IN as a transformation language.
2. Limitations apply depending on the database type and the target environment. For more information, see Database mappings in various execution environments 166.

18.1.2 Supported Features in Generated Code

The following table lists the features relevant to code generation and the extent of support in each language in **MapForce Professional Edition**.

Feature	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
Supply parameters to the mapping 346	●	●	●	●	●	●	●	●
Supply the input file names dynamically from the mapping 746	●	●	●	●	●	●	●	●
Supply wildcard file names as mapping input 746 [1]		●	●	●	●	●	●	●
Generate the output file names dynamically from the mapping 746		●	●		●	●	●	●
Return string values from the mapping 356	●	●	●		●	●	●	●
Variables 360		●	●	●	●	●	●	●

Feature	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
Sort components 402		●	●	●				●
Grouping functions 561		●	●		●	●	●	●
Filters 408	●	●	●	●	●	●	●	●
Join components 373								●
Value-Map components 421	●	●	●	●	●	●	●	●
Defaults and node functions 442								●
Mapping exceptions 432		●	●	●	●	●	●	●
String parsing and serialization 753 [2]								●
Dynamic node names 726		●	●	●	●	●	●	●
Database bulk inserts 259								●
Database SQL SELECT without input parameters 243					●	●	●	●
Database SQL SELECT with input parameters 243								●
Database stored procedures 297								●
Database exception handling 271 [3]					●	●	●	●
Database tracing and error logging 235								●
Generate MapForce Server execution files 799								●

Feature	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery	C++	C#	Java	BUILT-IN
Deploy mappings to FlowForce Server 802								●
Read data from binary files 628								●
Write data to binary files 630								●

Footnotes:

1. XSLT 2.0, XSLT 3.0, and XQuery use the `fn:collection` function. The implementation in the Altova XSLT 2.0, XSLT 3.0, and XQuery engines resolves wildcards. Other engines may behave differently.
2. For JSON, parsing and serialization are additionally supported in Java and C#.
3. Database exception handling is possible when the mapping language is supported by the currently connected database driver, as indicated in the previous table.

18.2 Engine Information

This section contains information about implementation-specific features of the Altova XML Validator, Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery Engine.

18.2.1 XSLT and XQuery Engine Information

The XSLT and XQuery engines of MapForce follow the W3C specifications closely and are therefore stricter than previous Altova engines—such as those in previous versions of XMLSpy. As a result, minor errors that were ignored by previous engines are now flagged as errors by MapForce.

For example:

- It is a type error (`err:XPTY0018`) if the result of a path operator contains both nodes and non-nodes.
- It is a type error (`err:XPTY0019`) if `E1` in a path expression `E1/E2` does not evaluate to a sequence of nodes.

If you encounter this kind of error, modify either the XSLT/XQuery document or the instance document as appropriate.

This section describes implementation-specific features of the engines, organized by specification:

- XSLT 1.0 [1101]
- XSLT 2.0 [1101]
- XQuery 1.0 [1103]

18.2.1.1 XSLT 1.0

The XSLT 1.0 Engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999. Note the following information about the implementation.

Notes about the implementation

When the `method` attribute of `xsl:output` is set to HTML, or if HTML output is selected by default, then special characters in the XML or XSLT file are inserted in the HTML document as HTML character references in the output. For instance, the character U+00A0 (the hexadecimal character reference for a non-breaking space) is inserted in the HTML code either as a character reference (`&#160;` or `&#xA0;`) or as an entity reference, `&nbsp;` .

18.2.1.2 XSLT 2.0

This section:

- Engine conformance [1102]

- Backward compatibility 1102
- Namespaces 1102
- Schema awareness 1103
- Implementation-specific behavior 1103

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. Typically, the backwards compatibility of the XSLT 2.0 engine comes into play when using the XSLT 2.0 engine to process an XSLT 1.0 stylesheet or instruction. Note that there could be differences in the outputs produced by the XSLT 1.0 Engine and the backwards-compatible XSLT 2.0 engine.

Namespaces

Your XSLT 2.0 stylesheet should declare the following namespaces in order for you to be able to use the type constructors and functions available in XSLT 2.0. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
XPath 2.0 functions	`fn:`	`http://www.w3.org/2005/xpath-functions`

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   ...
</xsl:stylesheet>
```

The following points should be noted:

- The XSLT 2.0 engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.
- When using type constructors and types from the XML Schema namespace, the prefix used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).
- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string('Hello')`, the expression evaluates as `fn:string('Hello')`—not as `xs:string('Hello')`.

Schema-awareness

The XSLT 2.0 engine is schema-aware. So you can use user-defined schema types and the `xsl:validate` instruction.

Implementation-specific behavior

Given below is a description of how the XSLT 2.0 engine handles implementation-specific aspects of 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` argument 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`. Example: `xs:base64Binary(unparsed-text('chart.png', 'x-binarytobase64'))`.

`unparsed-text-available`
The `href` argument 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`.

18.2.1.3 XQuery 1.0

This section:

- Engine conformance 1104
- Schema awareness 1104
- Encoding 1104
- Namespaces 1102
- XML source and validation 1105
- Static and dynamic type checking 1105
- Library modules 1105
- External functions 1105
- Collations 1105
- Precision of numeric data 1106
- XQuery instructions support 1106
- Implementation-specific behavior 1106

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 (see `fn:`) 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[1106]. To use a specific collation, supply its URI as given in the list of supported collations[1106]. 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.

Implementation-specific behavior

Given below is a description of how the XQuery and XQuery Update 1.0 engines handle implementation-specific aspects of certain functions.

`unparsed-text`
The `href` argument 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`. Example: `xs:base64Binary(unparsed-text('chart.png', 'x-binarytobase64'))`.

`unparsed-text-available`
The `href` argument 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`.

18.2.2 XSLT and XPath/XQuery Functions

This section lists Altova extension functions and other extension functions that can be used in XPath and/or XQuery expressions. Altova extension functions can be used with Altova's XSLT and XQuery engines, and provide functionality additional to that available in the function libraries defined in the W3C standards.

This section describes XPath/XQuery extension functions that have been created by Altova to provide additional operations, as well as other extension functions 1184. These extension functions 1108 can be computed by Altova's XSLT and XQuery engines according to the rules described in this section. For information about the regular XPath/XQuery functions, see Altova's XPath/XQuery Function Reference.

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 Altova XSLT and XQuery engines read non-prefixed functions as belonging to the namespace `http://www.w3.org/2005/xpath-functions`, which is the default functions namespace specified in the XPath/XQuery functions specifications. If this namespace is explicitly declared in an XSLT or XQuery document, the prefix used in the namespace declaration can also optionally be used on function names.

- In general, if a function expects a sequence of one item as an argument, and a sequence of more than one item is submitted, then an error is returned.
- All string comparisons are done using the Unicode codepoint collation.
- Results that are QNames are serialized in the form `[prefix:]localname`.

Precision of xs:decimal

The precision refers to the number of digits in the number, and a minimum of 18 digits is required by the specification. For division operations that produce a result of type `xs:decimal`, the precision is 19 digits after the decimal point with no rounding.

Implicit timezone

When two `date`, `time`, or `dateTime` values need to be compared, the timezones of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `implicit-timezone()` function.

Collations

The default collation is the Unicode codepoint collation, which compares strings on the basis of their Unicode codepoint. The engine uses the Unicode Collation Algorithm. Other supported collations are the ICU collations listed below; to use one of these, supply its URI as given in the table below. Any string comparisons, including for the `max` and `min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Language	**URIs**
`da`: Danish	`da_DK`
`de`: German	`de_AT, de_BE, de_CH, de_DE, de_LI, de_LU`
`en`: English	`en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA, en_GB, en_GU, en_HK, en_IE, en_IN, en_JM, en_MH, en_MP, en_MT, en_MU, en_NA, en_NZ, en_PH, en_PK, en_SG, en_TT, en_UM, en_US, en_VI, en_ZA, en_ZW`
`es`: Spanish	`es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE`
`fr`: French	`fr_BE, fr_BF, fr_BI, fr_BJ, fr_BL, fr_CA, fr_CD, fr_CF, fr_CG, fr_CH, fr_CI, fr_CM, fr_DJ, fr_FR, fr_GA, fr_GN, fr_GP, fr_GQ, fr_KM, fr_LU, fr_MC, fr_MF, fr_MG, fr_ML, fr_MQ, fr_NE, fr_RE, fr_RW, fr_SN, fr_TD, fr_TG`
`it`: Italian	`it_CH, it_IT`
`ja`: Japanese	`ja_JP`
`nb`: Norwegian Bokmal	`nb_NO`

`nl`: Dutch	`nl_AW, nl_BE, nl_NL`
`nn`: Nynorsk	`nn_NO`
`pt`: Portuguese	`pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST`
`ru`: Russian	`ru_MD, ru_RU, ru_UA`
`sv`: Swedish	`sv_FI, sv_SE`

Namespace axis
The namespace axis is deprecated in XPath 2.0. Use of the namespace axis is, however, supported. To access namespace information with XPath 2.0 mechanisms, use the `in-scope-prefixes()`, `namespace-uri()` and `namespace-uri-for-prefix()` functions.

18.2.2.1 Altova Extension Functions

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

Functions defined in the W3C's XPath/XQuery Functions specifications can be used in: (i) XPath expressions in an XSLT context, and (ii) in XQuery expressions in an XQuery document. In this documentation we indicate the functions that can be used in the former context (XPath in XSLT) with an **XP** symbol and call them XPath functions; those functions that can be used in the latter (XQuery) context are indicated with an **XQ** symbol; they work as XQuery functions. The W3C's XSLT specifications—not XPath/XQuery Functions specifications—also define functions that can be used in XPath expressions in XSLT documents. These functions are marked with an **XSLT** symbol and are called XSLT functions. The XPath/XQuery and XSLT versions in which a function can be used are indicated in the description of the function (*see symbols below*). Functions from the XPath/XQuery and XSLT function libraries are listed without a prefix. Extension functions from other libraries, such as Altova extension functions, are listed with a prefix.

XPath functions (used in XPath expressions in XSLT):	**XP1 XP2 XP3.1**
XSLT functions (used in XPath expressions in XSLT):	**XSLT1 XSLT2 XSLT3**
XQuery functions (used in XQuery expressions in XQuery):	**XQ1 XQ3.1**

Usage of Altova extension functions

In order to use Altova extension functions, you must declare the Altova extension functions namespace (*first highlight in code listing below*) and then use the extension functions so that they are resolved as belonging to this namespace (*see second highlight*). The example below uses the Altova extension function named **age**.

```
<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:altova="http://www.altova.com/xslt-extensions">
   <xsl:output method="text" encoding="ISO-8859-1"/>
   <xsl:template match="Persons">
      <xsl:for-each select="Person">
         <xsl:value-of select="concat(Name, ': ')"/>
         <xsl:value-of select="altova:age(xs:date(BirthDate))"/>
         <xsl:value-of select="' years&#x0A;'"/>
      </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

XSLT functions 1109

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 1112
- Geolocation 1129
- Image-related 1141
- Numeric 1145
- Schema 1147
- Sequence 1167
- String 1175
- Miscellaneous 1181

18.2.2.1.1 XSLT Functions

XSLT extension functions can be used in XPath expressions in an XSLT context. They will not work in a non-XSLT context (for instance, in an XQuery context).

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

General functions

distinct-nodes [altova:]

altova:distinct-nodes(*node()**) **as node()*** XSLT1 XSLT2 XSLT3
Takes a set of one or more nodes as its input and returns the same set minus nodes with duplicate values. The comparison is done using the XPath/XQuery function `fn:deep-equal`.

Examples

- **altova:distinct-nodes**(`country`) returns all child `country` nodes less those having duplicate values.

evaluate [altova:]

altova:evaluate(XPathExpression *as xs:string***[, ValueOf$p1, ... ValueOf$pN])** XSLT1 XSLT2 XSLT3
Takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression. For example: **altova:evaluate('//Name[1]')** returns the contents of the first `Name` element in the document. Note that the expression `//Name[1]` is passed as a string by enclosing it in single quotes.

The `altova:evaluate` function can optionally take additional arguments. These arguments are the values of in-scope variables that have the names `p1`, `p2`, `p3`... `pN`. Note the following points about usage: (i) The variables must be defined with names of the form `pX`, where `X` is an integer; (ii) the `altova:evaluate` function's arguments (*see signature above*), from the second argument onwards, provide the values of the variables, with the sequence of the arguments corresponding to the numerically ordered sequence of variables: `p1` to `pN`: The second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on; (iii) The variable values must be of type `item*`.

Example

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
```
outputs `"hi 20 10"`

In the listing above, notice the following:

- The second argument of the `altova:evaluate` expression is the value assigned to the variable `$p1`, the third argument that assigned to the variable `$p2`, and so on.
- Notice that the fourth argument of the function is a string value, indicated by its being enclosed in quotes.
- The `select` attribute of the `xs:variable` element supplies the XPath expression. Since this expression must be of type `xs:string`, it is enclosed in single quotes.

Examples to further illustrate the use of variables

- `<xsl:variable name="xpath" select="'$p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath, //Name[1])" />`
  Outputs value of the first `Name` *element.*

- `<xsl:variable name="xpath" select="'$p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath, '//Name[1]')" />`
  Outputs "`//Name[1]`"

The `altova:evaluate()` extension function is useful in situations where an XPath expression in the XSLT stylesheet contains one or more parts that must be evaluated dynamically. For example, consider a situation in which a user enters his request for the sorting criterion and this criterion is stored in the attribute `UserReq/@sortkey`. In the stylesheet, you could then have the expression: `<xsl:sort select="`**`altova:evaluate(../UserReq/@sortkey)`**`" order="ascending"/>`. The `altova:evaluate()` function reads the `sortkey` attribute of the `UserReq` child element of the parent of the context node. Say the value of the `sortkey` attribute is `Price`, then `Price` is returned by the `altova:evaluate()` function and becomes the value of the `select` attribute: `<xsl:sort select="`**`Price`**`" order="ascending"/>`. If this `sort` instruction occurs within the context of an element called `Order`, then the `Order` elements will be sorted according to the values of their `Price` children. Alternatively, if the value of `@sortkey` were, say, `Date`, then the `Order` elements would be sorted according to the values of their `Date` children. So the sort criterion for `Order` is selected from the `sortkey` attribute at runtime. This could not have been achieved with an expression like: `<xsl:sort select="`**`../UserReq/@sortkey`**`" order="ascending"/>`. In the case shown above, the sort criterion would be the `sortkey` attribute itself, not `Price` or `Date` (or any other current content of `sortkey`).

Note: The static context includes namespaces, types, and functions—but not variables—from the calling environment. The base URI and default namespace are inherited.

⊟ *More examples*

- Static variables: `<xsl:value-of select="$i3, $i2, $i1" />`
  Outputs the values of three variables.

- Dynamic XPath expression with dynamic variables:
  `<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />`
  Outputs "`30 20 10`"

- Dynamic XPath expression with no dynamic variable:
  `<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath)" />`
  Outputs error: No variable defined for $p3.

▼ encode-for-rtf [altova:]

`altova:encode-for-rtf(input` *`as xs:string`*`,` **`preserveallwhitespace`** *`as xs:boolean`*`,` **`preservenewlines`** *`as xs:boolean`***`)`** **`as xs:string`** **XSLT2** **XSLT3**

Converts the input string into code for RTF. Whitespace and new lines will be preserved according to the boolean value specified for their respective arguments.

[Top 1109]

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 1109]

18.2.2.1.2 XPath/XQuery Functions: Date and Time

Altova's date/time extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data held as XML Schema's various date and time datatypes. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ Grouped by functionality

- Add a duration to xs:dateTime and return xs:dateTime 1114
- Add a duration to xs:date and return xs:date 1115
- Add a duration to xs:time and return xs:time 1117
- Format and retrieve durations 1116
- Remove timezone from functions that generate current date/time 1118

- Return days, hours, minutes, and seconds from durations 1119
- Return weekday as integer from date 1121
- Return week number as integer from date 1121
- Build date, time, or duration type from lexical components of each type 1123
- Construct date, dateTime, or time type from string input 1124
- Age-related functions 1126
- Epoch time (Unix time) functions 1127

▼ Listed alphabetically

`altova:add-days-to-date` 1115
`altova:add-days-to-dateTime` 1114
`altova:add-hours-to-dateTime` 1114
`altova:add-hours-to-time` 1117
`altova:add-minutes-to-dateTime` 1114
`altova:add-minutes-to-time` 1117
`altova:add-months-to-date` 1115
`altova:add-months-to-dateTime` 1114
`altova:add-seconds-to-dateTime` 1114
`altova:add-seconds-to-time` 1117
`altova:add-years-to-date` 1115
`altova:add-years-to-dateTime` 1114
`altova:age` 1126
`altova:age-details` 1126
`altova:build-date` 1123
`altova:build-duration` 1123
`altova:build-time` 1123
`altova:current-dateTime-no-TZ` 1118
`altova:current-date-no-TZ` 1118
`altova:current-time-no-TZ` 1118
`altova:date-no-TZ` 1118
`altova:dateTime-from-epoch` 1127
`altova:dateTime-from-epoch-no-TZ` 1127
`altova:dateTime-no-TZ` 1118
`altova:days-in-month` 1119
`altova:epoch-from-dateTime` 1127
`altova:hours-from-dateTimeDuration-accumulated` 1119
`altova:minutes-from-dateTimeDuration-accumulated` 1119
`altova:seconds-from-dateTimeDuration-accumulated` 1119
`altova:format-duration` 1116
`altova:parse-date` 1124
`altova:parse-dateTime` 1124
`altova:parse-duration` 1116
`altova:parse-time` 1124
`altova:time-no-TZ` 1118
`altova:weekday-from-date` 1121
`altova:weekday-from-dateTime` 1121
`altova:weeknumber-from-date` 1122
`altova:weeknumber-from-dateTime` 1122

[**Top** 1112]

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 1112]

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 1112]

Format and retrieve durations XP3.1 XQ3.1

These functions parse an input **xs:duration** or **xs:string** and return, respectively, an **xs:string** or **xs:duration**.

▼ format-duration [altova:]

altova:format-duration(Duration *as xs:duration*, **Picture** *as xs:string*) **as xs:string** **XP3.1** **XQ3.1**
Formats a duration, which is submitted as the first argument, according to a picture string submitted as the second argument. The output is a text string formatted according to the picture string.

⊟ *Examples*

- **altova:format-duration**(xs:duration("P2DT2H53M11.7S"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"
- **altova:format-duration**(xs:duration("P3M2DT2H53M11.7S"), "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "Months:03 Days:02 Hours:02 Minutes:53"

▼ parse-duration [altova:]

altova:parse-duration(InputString *as xs:string*, **Picture** *as xs:string*) **as xs:duration** **XP3.1** **XQ3.1**
Takes a patterned string as the first argument, and a picture string as the second argument. The input

string is parsed on the basis of the picture string, and an `xs:duration` is returned.

⊟ *Examples*

- **`altova:parse-duration`**`("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]")` returns `"P2DT2H53M11.7S"`
- **`altova:parse-duration`**`("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]")` returns `"P3M2DT2H53M"`

[Top 1112]

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 1112]

Remove the timezone part from date/time datatypes XP3.1 XQ3.1

These functions remove the timezone from the current **`xs:dateTime`**, **`xs:date`**, or **`xs:time`** values, respectively. Note that the difference between `xs:dateTime` and `xs:dateTimeStamp` is that in the case of the latter the timezone part is required (while it is optional in the case of the former). So the format of an `xs:dateTimeStamp` value is: `CCYY-MM-DDThh:mm:ss.sss±hh:mm`. or `CCYY-MM-DDThh:mm:ss.sssZ`. If the date and time is read from the system clock as `xs:dateTimeStamp`, the `current-dateTime-no-TZ()` function can be used to remove the timezone if so required.

▼ current-date-no-TZ [altova:]

`altova:current-date-no-TZ()` **`as xs:date`** XP3.1 XQ3.1
This function takes no argument. It removes the timezone part of `current-date()` (which is the current date according to the system clock) and returns an `xs:date` value.

⊟ *Examples*

If the current date is **`2014-01-15+01:00`**:

- **`altova:current-date-no-TZ`**`()` returns `2014-01-15`

▼ current-dateTime-no-TZ [altova:]

`altova:current-dateTime-no-TZ()` **`as xs:dateTime`** XP3.1 XQ3.1
This function takes no argument. It removes the timezone part of `current-dateTime()` (which is the current date-and-time according to the system clock) and returns an `xs:dateTime` value.

⊟ *Examples*

If the current dateTime is **`2014-01-15T14:00:00+01:00`**:

- **`altova:current-dateTime-no-TZ`**`()` returns `2014-01-15T14:00:00`

▼ current-time-no-TZ [altova:]

`altova:current-time-no-TZ()` **`as xs:time`** XP3.1 XQ3.1
This function takes no argument. It removes the timezone part of `current-time()` (which is the current time according to the system clock) and returns an `xs:time` value.

⊟ *Examples*

If the current time is **`14:00:00+01:00`**:

- **`altova:current-time-no-TZ`**`()` returns `14:00:00`

▼ date-no-TZ [altova:]

`altova:date-no-TZ(InputDate` *`as xs:date`***`)`** **`as xs:date`** XP3.1 XQ3.1
This function takes an `xs:date` argument, removes the timezone part from it, and returns an `xs:date` value. Note that the date is not modified.

⊟ *Examples*

- **altova:date-no-TZ**(xs:date("2014-01-15+01:00")) returns 2014-01-15

▼ dateTime-no-TZ [altova:]

altova:dateTime-no-TZ(InputDateTime *as xs:dateTime***) as xs:dateTime** **XP3.1** **XQ3.1**
This function takes an `xs:dateTime` argument, removes the timezone part from it, and returns an `xs:dateTime` value. Note that neither the date nor the time is modified.

⊟ *Examples*

- **altova:dateTime-no-TZ**(xs:date("2014-01-15T14:00:00+01:00")) returns 2014-01-15T14:00:00

▼ time-no-TZ [altova:]

altova:time-no-TZ(InputTime *as xs:time***) as xs:time** **XP3.1** **XQ3.1**
This function takes an `xs:time` argument, removes the timezone part from it, and returns an `xs:time` value. Note that the time is not modified.

⊟ *Examples*

- **altova:time-no-TZ**(xs:time("14:00:00+01:00")) returns 14:00:00

[Top1112]

Return the number of days, hours, minutes, seconds from durations XP3.1 XQ3.1

These functions return the number of days in a month, and the number of hours, minutes, and seconds, respectively, from durations.

▼ days-in-month [altova:]

altova:days-in-month(Year *as xs:integer***, Month** *as xs:integer***) as xs:integer** **XP3.1** **XQ3.1**
Returns the number of days in the specified month. The month is specified by means of the `Year` and `Month` arguments.

⊟ *Examples*

- **altova:days-in-month**(2018, 10) returns 31
- **altova:days-in-month**(2018, 2) returns 28
- **altova:days-in-month**(2020, 2) returns 29

▼ hours-from-dayTimeDuration-accumulated

altova:hours-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration***) as xs:integer** **XP3.1** **XQ3.1**
Returns the total number of hours in the duration submitted by the `DayAndTime` argument (which is of type `xs:duration`). The hours in the `Day` and `Time` components are added together to give a result that is an integer. A new hour is counted only for a full 60 minutes. Negative durations result in a negative hour value.

⊟ *Examples*

- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5D")) returns 120, which is the total number of hours in 5 days.

- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H"))` returns `122`, which is the total number of hours in 5 days plus 2 hours.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H60M"))` returns `123`, which is the total number of hours in 5 days plus 2 hours and 60 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H119M"))` returns `123`, which is the total number of hours in 5 days plus 2 hours and 119 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("P5DT2H120M"))` returns `124`, which is the total number of hours in 5 days plus 2 hours and 120 mins.
- `altova:hours-from-dayTimeDuration-accumulated(xs:duration("-P5DT2H"))` returns `-122`

▼ minutes-from-dayTimeDuration-accumulated

`altova:minutes-from-dayTimeDuration-accumulated(DayAndTime` *`as xs:duration`*`)` **`as xs:integer`** **XP3.1** **XQ3.1**

Returns the total number of minutes in the duration submitted by the `DayAndTime` argument (which is of type `xs:duration`). The minutes in the `Day` and `Time` components are added together to give a result that is an integer. Negative durations result in a negative minute value.

⊟ *Examples*

- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT60M"))` returns `60`
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT1H"))` returns `60`, which is the total number of minutes in 1 hour.
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("PT1H40M"))` returns `100`
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("P1D"))` returns `1440`, which is the total number of minutes in 1 day.
- `altova:minutes-from-dayTimeDuration-accumulated(xs:duration("-P1DT60M"))` returns `-1500`

▼ seconds-from-dayTimeDuration-accumulated

`altova:seconds-from-dayTimeDuration-accumulated(DayAndTime` *`as xs:duration`*`)` **`as xs:integer`** **XP3.1** **XQ3.1**

Returns the total number of seconds in the duration submitted by the `DayAndTime` argument (which is of type `xs:duration`). The seconds in the `Day` and `Time` components are added together to give a result that is an integer. Negative durations result in a negative seconds value.

⊟ *Examples*

- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1M"))` returns `60`, which is the total number of seconds in 1 minute.
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1H"))` returns `3600`, which is the total number of seconds in 1 hour.
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("PT1H2M"))` returns `3720`
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("P1D"))` returns `86400`, which is the total number of seconds in 1 day.
- `altova:seconds-from-dayTimeDuration-accumulated(xs:duration("-P1DT1M"))` returns `-86460`

Return the weekday from xs:dateTime or xs:date XP3.1 XQ3.1

These functions return the weekday (as an integer) from `xs:dateTime` or `xs:date`. The days of the week are numbered (using the American format) from `1` to `7`, with `Sunday=1`. In the European format, the week starts with Monday (=`1`). The American format, where `Sunday=1`, can be set by using the integer `0` where an integer is accepted to indicate the format.

▼ weekday-from-dateTime [altova:]

altova:weekday-from-dateTime(DateTime *as xs:dateTime***) as xs:integer** XP3.1 XQ3.1

Takes a date-with-time as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Sunday=1`. If the European format is required (where `Monday=1`), use the other signature of this function (*see next signature below*).

⊟ *Examples*

- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"))` returns `2`, which would indicate a Monday.

altova:weekday-from-dateTime(DateTime *as xs:dateTime***, Format** *as xs:integer***) as xs:integer** XP3.1 XQ3.1

Takes a date-with-time as its first argument and returns the day of the week of this date as an integer. If the second (integer) argument is `0`, then the weekdays are numbered `1` to `7` starting with `Sunday=1`. If the second argument is an integer other than `0`, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

⊟ *Examples*

- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 1)` returns `1`, which would indicate a Monday
- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 4)` returns `1`, which would indicate a Monday
- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 0)` returns `2`, which would indicate a Monday.

▼ weekday-from-date [altova:]

altova:weekday-from-date(Date *as xs:date***) as xs:integer** XP3.1 XQ3.1

Takes a date as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Sunday=1`. If the European format is required (where `Monday=1`), use the other signature of this function (*see next signature below*).

⊟ *Examples*

- `altova:weekday-from-date(xs:date("2014-02-03+01:00"))` returns `2`, which would indicate a Monday.

altova:weekday-from-date(Date *as xs:date***, Format** *as xs:integer***) as xs:integer** XP3.1 XQ3.1

Takes a date as its first argument and returns the day of the week of this date as an integer. If the second (`Format`) argument is `0`, then the weekdays are numbered `1` to `7` starting with `Sunday=1`. If the second argument is an integer other than `0`, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

⊟ *Examples*

- `altova:weekday-from-date(xs:date("2014-02-03"), 1)` returns `1`, which would indicate a

Monday
- `altova:weekday-from-date(xs:date("2014-02-03"), 4)` returns `1`, which would indicate a Monday
- `altova:weekday-from-date(xs:date("2014-02-03"), 0)` returns `2`, which would indicate a Monday.

[Top 1112]

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.

[Top1112]

Build date, time, and duration datatypes from their lexical components XP3.1 XQ3.1

The functions take the lexical components of the xs:date, xs:time, or xs:duration datatype as input arguments and combine them to build the respective datatype.

▼ build-date [altova:]

altova:build-date(Year *as xs:integer*, **Month** *as xs:integer*, **Date** *as xs:integer*) **as xs:date** XP3.1 XQ3.1

The first, second, and third arguments are, respectively, the year, month, and date. They are combined to build a value of xs:date type. The values of the integers must be within the correct range of that particular date part. For example, the second argument (for the month part) should not be greater than 12.

⊟ *Examples*

- **altova:build-date**(2014, 2, 03) returns 2014-02-03

▼ build-time [altova:]

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*) **as xs:time** XP3.1 XQ3.1

The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. They are combined to build a value of xs:time type. The values of the integers must be within the correct range of that particular time part. For example, the second (Minutes) argument should not be greater than 59. To add a timezone part to the value, use the other signature of this function (*see next signature*).

⊟ *Examples*

- **altova:build-time**(23, 4, 57) returns 23:04:57

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*, **TimeZone** *as xs:string*) **as xs:time** XP3.1 XQ3.1

The first, second, and third arguments are, respectively, the hour (0 to 23), minutes (0 to 59), and seconds (0 to 59) values. The fourth argument is a string that provides the timezone part of the value. The four arguments are combined to build a value of xs:time type. The values of the integers must be within the correct range of that particular time part. For example, the second (`Minutes`) argument should not be greater than `59`.

⊟ *Examples*

- `altova:build-time(23, 4, 57, '+1')` returns `23:04:57+01:00`

▼ build-duration [altova:]

`altova:build-duration(Years` *`as xs:integer`*, **`Months`** *`as xs:integer`***`) as xs:yearMonthDuration`** XP3.1 XQ3.1

Takes two arguments to build a value of type `xs:yearMonthDuration`. The first argument provides the `Years` part of the duration value, while the second argument provides the `Months` part. If the second (`Months`) argument is greater than or equal to `12`, then the integer is divided by `12`; the quotient is added to the first argument to provide the `Years` part of the duration value while the remainder (of the division) provides the `Months` part. To build a duration of type `xs:dayTimeDuration`., see the next signature.

⊟ *Examples*

- `altova:build-duration(2, 10)` returns `P2Y10M`
- `altova:build-duration(14, 27)` returns `P16Y3M`
- `altova:build-duration(2, 24)` returns `P4Y`

`altova:build-duration(Days` *`as xs:integer`*, **`Hours`** *`as xs:integer`*, **`Minutes`** *`as xs:integer`*, **`Seconds`** *`as xs:integer`***`)`** as **`xs:dayTimeDuration`** XP3.1 XQ3.1

Takes four arguments and combines them to build a value of type `xs:dayTimeDuration`. The first argument provides the `Days` part of the duration value, the second, third, and fourth arguments provide, respectively, the `Hours`, `Minutes`, and `Seconds` parts of the duration value. Each of the three Time arguments is converted to an equivalent value in terms of the next higher unit and the result is used for calculation of the total duration value. For example, 72 seconds is converted to `1M+12S` (1 minute and 12 seconds), and this value is used for calculation of the total duration value. To build a duration of type `xs:yearMonthDuration`., see the previous signature.

⊟ *Examples*

- `altova:build-duration(2, 10, 3, 56)` returns `P2DT10H3M56S`
- `altova:build-duration(1, 0, 100, 0)` returns `P1DT1H40M`
- `altova:build-duration(1, 0, 0, 3600)` returns `P1DT1H`

[Top 1112]

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 1112]

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 1112]

Epoch time (Unix time) functions XP3.1 XQ3.1

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. Altova's Epoch time extension functions convert **xs:dateTime** values to Epoch time values and vice versa.

▾ dateTime-from-epoch [altova:]

altova:dateTime-from-epoch(Epoch *as xs:decimal* **as xs:dateTime** **XP3.1** **XQ3.1**
Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **dateTime-from-epoch** function returns the **xs:dateTime** equivalent of an Epoch time, adjusts it for the local timezone, and includes the timezone information in the result.

The function takes an `xs:decimal` argument and returns an `xs:dateTime` value that includes a **TZ** (timezone) part. The result is obtained by calculating the UTC **dateTime** equivalet of the Epoch time, and adding to it the local timezone (taken from the system clock). For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then after the UTC **dateTime** equivalent has been calculated, one hour will be added to the result. The timezone information, which is an optional lexical part of the **xs:dateTime** result, is also reported in the **dateTime** result. Compare this result with that of **dateTime-from-epoch-no-TZ**, and also see the function **epoch-from-dateTime**.

⊟ *Examples*

The examples below assume a local timezone of UTC +01:00. Consequently, the UTC **dateTime** equivalent of the submitted Epoch time will be incremented by one hour. The timezone is reported in the result.

- **altova:dateTime-from-epoch**(34) returns 1970-01-01T01:00:34+01:00
- **altova:dateTime-from-epoch**(62) returns 1970-01-01T01:01:02+01:00

dateTime-from-epoch-no-TZ [altova:]

altova:dateTime-from-epoch-no-TZ(Epoch *as xs:decimal* **as xs:dateTime** **XP3.1** **XQ3.1**
Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **dateTime-from-epoch-no-TZ** function returns the **xs:dateTime** equivalent of an Epoch time, adjusts it for the local timezone, but does not include the timezone information in the result.

The function takes an xs:decimal argument and returns an xs:dateTime value that does not includes a **TZ** (timezone) part. The result is obtained by calculating the UTC **dateTime** equivalet of the Epoch time, and adding to it the local timezone (taken from the system clock). For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then after the UTC **dateTime** equivalent has been calculated, one hour will be added to the result. The timezone information, which is an optional lexical part of the **xs:dateTime** result, is not reported in the **dateTime** result. Compare this result with that of **dateTime-from-epoch**, and also see the function **epoch-from-dateTime**.

⊟ *Examples*

The examples below assume a local timezone of UTC +01:00. Consequently, the UTC **dateTime** equivalent of the submitted Epoch time will be incremented by one hour. The timezone is not reported in the result.

- **altova:dateTime-from-epoch**(34) returns 1970-01-01T01:00:34
- **altova:dateTime-from-epoch**(62) returns 1970-01-01T01:01:02

epoch-from-dateTime [altova:]

altova:epoch-from-dateTime(dateTimeValue *as xs:dateTime***)** **as xs:decimal** **XP3.1** **XQ3.1**
Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **epoch-from-dateTime** function returns the Epoch time equivalent of the **xs:dateTime** that is submitted as the argument of the function. Note that you might have to explicitly construct the **xs:dateTime** value. The submitted **xs:dateTime** value may or may not contain the optional **TZ** (timezone) part.

Whether the timezone part is submitted as part of the argument or not, the local timezone offset (taken from the system clock) is subtracted from the submitted **dateTimeValue** argument. This produces the equivalent UTC time, from which the equivalent Epoch time is calculated. For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then one hour is subtracted from the submitted **dateTimeValue** before the Epoch value is calculated. Also see the function **dateTime-from-epoch**.

⊟ *Examples*

The examples below assume a local timezone of UTC +01:00. Consequently, one hour will be subtracted from the submitted **dateTime** before the Epoch time is calculated.

- **altova:epoch-from-dateTime**(xs:dateTime("1970-01-01T01:00:34+01:00")) returns 34
- **altova:epoch-from-dateTime**(xs:dateTime("1970-01-01T01:00:34")) returns 34
- **altova:epoch-from-dateTime**(xs:dateTime("2021-04-01T11:22:33")) returns 1617272553

[Top 1112]

18.2.2.1.3 XPath/XQuery Functions: Geolocation

The following geolocation XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ format-geolocation [altova:]

altova:format-geolocation(Latitude *as xs:decimal*, **Longitude** *as xs:decimal*, **GeolocationOutputStringFormat** *as xs:integer*) **as xs:string** XP3.1 XQ3.1

Takes the latitude and longitude as the first two arguments, and outputs the geolocation as a string. The third argument, **GeolocationOutputStringFormat**, is the format of the geolocation output string; it uses integer values from 1 to 4 to identify the output string format (*see 'Geolocation output string formats' below*). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data 1141 function and the Exif metadata's attributes can be used to supply the input strings.

Examples

- `altova:format-geolocation(33.33, -22.22, 4)` returns the `xs:string` `"33.33 -22.22"`
- `altova:format-geolocation(33.33, -22.22, 2)` returns the `xs:string` `"33.33N 22.22W"`
- `altova:format-geolocation(-33.33, 22.22, 2)` returns the `xs:string` `"33.33S 22.22E"`
- `altova:format-geolocation(33.33, -22.22, 1)` returns the `xs:string` `"33°19'48.00"S 22° 13'12.00"E"`

Geolocation output string formats:

The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (1 to 4). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

1
Degrees, minutes, decimal seconds, with suffixed orientation (`N/S, E/W`) **`D°M'S.SS"N/S  D°M'S.SS"E/W`** *Example*: **`33°55'11.11"N  22°44'66.66"W`**

2
Decimal degrees, with suffixed orientation (`N/S`, `E/W`) **`D.DDN/S  D.DDE/W`** *Example*: **`33.33N  22.22W`**

3
Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); plus sign for (**`N/E`**) is optional **`+/-D°M'S.SS"  +/-D°M'S.SS"`** *Example*: **`33°55'11.11"  -22°44'66.66"`**

4
Decimal degrees, with prefixed sign (`+/-`); plus sign for (**`N/E`**) is optional **`+/-D.DD  +/-D.DD`** *Example*: **`33.33 -22.22`**

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: **`GPSLatitude`**, **`GPSLatitudeRef`**, **`GPSLongitude`**, **`GPSLongitudeRef`**, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

▼ parse-geolocation [altova:]

`altova:parse-geolocation(GeolocationInputString` *`as xs:string`***`)`** **`as xs:decimal+`** **`XP3.1`** **`XQ3.1`**
Parses the supplied `GeolocationInputString` argument and returns the geolocation's latitude and longitude (in that order) as a sequence two `xs:decimal` items. The formats in which the geolocation input string can be supplied are listed below.

Note: The `image-exif-data` 1141 function and the Exif metadata's `@Geolocation` 1141 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`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional

`+/-D°M.MM'  +/-D°M.MM'`
Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`
`33.33  22°44'55.25"W`
`33.33  22.45`

⊟ *Altova Exif Attribute: Geolocation*

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: **`GPSLatitude`**, **`GPSLatitudeRef`**, **`GPSLongitude`**, **`GPSLongitudeRef`**, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

▼ geolocation-distance-km [altova:]

`altova:geolocation-distance-km(GeolocationInputString-1` *`as xs:string`*, **`GeolocationInputString-2`** *`as xs:string`*) **`as xs:decimal`** **XP3.1** **XQ3.1**
Calculates the distance between two geolocations in kilometers. The formats in which the geolocation input string can be supplied are listed below. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data`[1141] function and the Exif metadata's `@Geolocation`[1141] attribute can be used to supply geolocation input strings.

⊟ *Examples*

- **`altova:geolocation-distance-km`**`("33.33  -22.22", "48°51'29.6""N  24°17'40.2""")`
  returns the `xs:decimal` `4183.08132372392`

⊟ *Geolocation input string formats:*

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`
`33.33  22°44'55.25"W`
`33.33  22.45`

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: **`GPSLatitude`**, **`GPSLatitudeRef`**, **`GPSLongitude`**, **`GPSLongitudeRef`**, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

▼ geolocation-distance-mi [altova:]

altova:geolocation-distance-mi(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) **as xs:decimal** **XP3.1** **XQ3.1**

Calculates the distance between two geolocations in miles. The formats in which a geolocation input string can be supplied are listed below. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data` 1141 function and the Exif metadata's `@Geolocation` 1141 attribute can be used to supply geolocation input strings.

⊟ *Examples*

- `altova:geolocation-distance-mi("33.33  -22.22", "48°51'29.6""N  24°17'40.2""")` returns the `xs:decimal` `2599.40652340653`

⊟ *Geolocation input string formats:*

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`
`33.33  22°44'55.25"W`
`33.33  22.45`

⊟ *Altova Exif Attribute: Geolocation*

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: **`GPSLatitude`**, **`GPSLatitudeRef`**, **`GPSLongitude`**, **`GPSLongitudeRef`**, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

▼ geolocations-bounding-rectangle [altova:]

`altova:geolocations-bounding-rectangle(Geolocations` *`as xs:sequence`*, **`GeolocationOutputStringFormat`** *`as xs:integer`*) **`as xs:string`** **XP3.1** **XQ3.1**
Takes a sequence of strings as its first argument; each string in the sequence is a geolocation. The function returns a sequence of two strings which are, respectively, the top-left and bottom-right geolocation coordinates of a bounding rectangle that is optimally sized to enclose all the geolocations submitted in the first argument. The formats in which a geolocation input string can be supplied are listed below (*see 'Geolocation input string formats'*). Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

The function's second argument specifies the format of the two geolocation strings in the output sequence. The argument takes an integer value from `1` to `4`, where each value identifies a different geolocation string format (*see 'Geolocation output string formats' below*).

Note: The `image-exif-data` 1141 function and the Exif metadata's attributes can be used to supply the input strings.

⊟ *Examples*

- **`altova:geolocations-bounding-rectangle`**`(("48.2143531 16.3707266", "51.50939 -0.11832"), 1)` returns the sequence `("51°30'33.804"N 0°7'5.952"W", "48°12'51.67116"N 16°22'14.61576"E")`
- **`altova:geolocations-bounding-rectangle`**`(("48.2143531 16.3707266", "51.50939 -0.11832", "42.5584577 -70.8893334"), 4)` returns the sequence `("51.50939 -70.8893334", "42.5584577 16.3707266")`

⊟ *Geolocation input string formats:*

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (**"**) while unit indicators that are escaped are highlighted in blue (**""**).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`
`33.33  22°44'55.25"W`
`33.33  22.45`

⊟ _Geolocation output string formats:_

The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (1 to 4). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

1
Degrees, minutes, decimal seconds, with suffixed orientation (`N/S, E/W`) **`D°M'S.SS"N/S  D°M'S.SS"E/W`**

Example: `33°55'11.11"N  22°44'66.66"W`

2
Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
`D.DDN/S  D.DDE/W`
Example: `33.33N  22.22W`

3
Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); plus sign for (**`N/E`**) is optional
`+/-D°M'S.SS"  +/-D°M'S.SS"`
Example: `33°55'11.11"  -22°44'66.66"`

4
Decimal degrees, with prefixed sign (`+/-`); plus sign for (**`N/E`**) is optional
`+/-D.DD  +/-D.DD`
Example: `33.33 -22.22`

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: **`GPSLatitude`**, **`GPSLatitudeRef`**, **`GPSLongitude`**, **`GPSLongitudeRef`**, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
`33 51 21.91`	`S`	`151 13 11.73`	`E`	`33°51'21.91"S 151°13'11.73"E`

geolocation-within-polygon [altova:]

`altova:geolocation-within-polygon(Geolocation` *`as xs:string`*, **`((PolygonPoint`** *`as xs:string`*`)+))` **`as xs:boolean`** **XP3.1** **XQ3.1**

Determines whether **`Geolocation`** (the first argument) is within the polygonal area described by the **`PolygonPoint`** arguments. If the `PolygonPoint` arguments do not form a closed figure (formed when the first point and the last point are the same), then the first point is implicitly added as the last point in order to close the figure. All the arguments (`Geolocation` and `PolygonPoint+`) are given by geolocation input strings (*formats listed below*). If the `Geolocation` argument is within the polygonal area, then the function returns `true()`; otherwise it returns `false()`. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data`[1141] function and the Exif metadata's `@Geolocation`[1141] attribute can be used to supply geolocation input strings.

⊟ *Examples*

- `altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48 24", "58 -32"))` returns `true()`
- `altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48 24"))` returns `true()`
- `altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48°51'29.6""N 24°17'40.2"""))` returns `true()`

⊟ *Geolocation input string formats:*

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
```

```
33.33  22.45
```

⊟ *Altova Exif Attribute: Geolocation*

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: **`GPSLatitude`**, **`GPSLatitudeRef`**, **`GPSLongitude`**, **`GPSLongitudeRef`**, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

geolocation-within-rectangle [altova:]

`altova:geolocation-within-rectangle(Geolocation` *`as xs:string`*, **`RectCorner-1`** *`as xs:string`*, **`RectCorner-2`** *`as xs:string`***`) as xs:boolean`** **XP3.1** **XQ3.1**

Determines whether **`Geolocation`** (the first argument) is within the rectangle defined by the second and third arguments, **`RectCorner-1`** and **`RectCorner-2`**, which specify opposite corners of the rectangle. All the arguments (`Geolocation`, **`RectCorner-1`** and **`RectCorner-2`**) are given by geolocation input strings (*formats listed below*). If the `Geolocation` argument is within the rectangle, then the function returns `true()`; otherwise it returns `false()`. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data` 1141 function and the Exif metadata's `@Geolocation` 1141 attribute can be used to supply geolocation input strings.

⊟ *Examples*

- **`altova:geolocation-within-rectangle(`**`"33 -22", "58 -32", "-48 24"`) returns `true()`
- **`altova:geolocation-within-rectangle(`**`"33 -22", "58 -32", "48 24"`) returns `false()`
- **`altova:geolocation-within-rectangle(`**`"33 -22", "58 -32", "48°51'29.6""S  24° 17'40.2"""`) returns `true()`

⊟ *Geolocation input string formats:*

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)

`D°M'S.SS"N/S  D°M'S.SS"W/E`
Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

⊟ *Altova Exif Attribute: Geolocation*

The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: **`GPSLatitude`**, **`GPSLatitudeRef`**, **`GPSLongitude`**, **`GPSLongitudeRef`**, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

[Top 1129]

18.2.2.1.4 XPath/XQuery Functions: Image-Related

The following image-related XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	**XP1 XP2 XP3.1**
XSLT functions (used in XPath expressions in XSLT):	**XSLT1 XSLT2 XSLT3**
XQuery functions (used in XQuery expressions in XQuery):	**XQ1 XQ3.1**

▼ suggested-image-file-extension [altova:]

`altova:suggested-image-file-extension(Base64String` *`as string`***`) as string?`** **XP3.1 XQ3.1**
Takes the Base64 encoding of an image file as its argument and returns the file extension of the image as recorded in the Base64-encoding of the image. The returned value is a suggestion based on the image type information available in the encoding. If this information is not available, then an empty string is returned. This function is useful if you wish to save a Base64 image as a file and wish to dynamically retrieve an appropriate file extension.

⊟ *Examples*

- **`altova:suggested-image-file-extension`**`(/MyImages/MobilePhone/Image20141130.01)` returns `'jpg'`
- **`altova:suggested-image-file-extension`**`($XML1/Staff/Person/@photo)` returns `''`

In the examples above, the nodes supplied as the argument of the function are assumed to contain a Base64-encoded image. The first example retrieves `jpg` as the file's type and extension. In the second example, the submitted Base64 encoding does not provide usable file extension information.

▼ image-exif-data [altova:]

`altova:image-exif-data(Base64BinaryString` *`as string`***`) as element?`** **XP3.1 XQ3.1**
Takes a Base64-encoded JPEG image as its argument and returns an element called **`Exif`** that contains the Exif metadata of the image. The Exif metadata is created as attribute-value pairs of the `Exif` element. The attribute names are the Exif data tags found in the Base64 encoding. The list of Exif-specification tags is given below. If a vendor-specific tag is present in the Exif data, this tag and its value will also be returned as an attribute-value pair. Additional to the standard Exif metadata tags (*see list below*), Altova-specific attribute-value pairs are also generated. These Altova Exif attributes are listed below.

Examples

- To access any one attribute, use the function like this:
  `image-exif-data(//MyImages/Image20141130.01)/@GPSLatitude`
  `image-exif-data(//MyImages/Image20141130.01)/@Geolocation`
- To access all the attributes, use the function like this:
  `image-exif-data(//MyImages/Image20141130.01)/@*`
- To access the names of all the attributes, use the following expression:
  `for $i in image-exif-data(//MyImages/Image20141130.01)/@* return name($i)`
  This is useful to find out the names of the attributes returned by the function.

Altova Exif Attribute: Geolocation

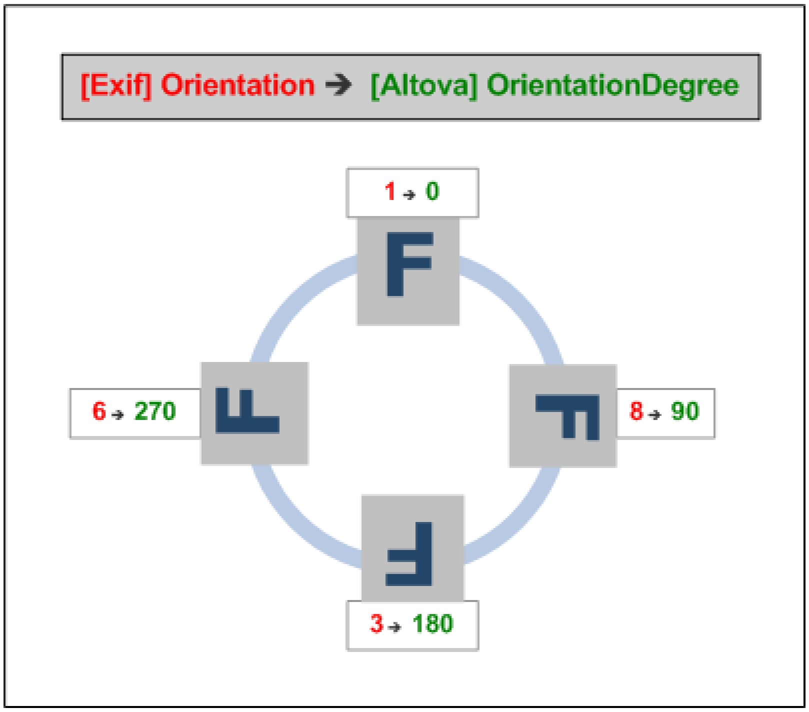
The Altova XPath/XQuery Engine generates the custom attribute Geolocation from standard Exif metadata tags. Geolocation is a concatenation of four Exif tags: **`GPSLatitude`**, **`GPSLatitudeRef`**, **`GPSLongitude`**, **`GPSLongitudeRef`**, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°13'11.73"E

Altova Exif Attribute: OrientationDegree

The Altova XPath/XQuery Engine generates the custom attribute **`OrientationDegree`** from the Exif metadata tag **`Orientation`**.

`OrientationDegree` translates the standard Exif tag **`Orientation`** from an integer value (**`1`**, **`8`**, **`3`**, or **`6`**) to the respective degree values of each (**`0`**, **`90`**, **`180`**, **`270`**), as shown in the figure below. Note that there are no translations of the **`Orientation`** values of **`2`**, **`4`**, **`5`**, **`7`**. (These orientations are obtained by flipping image 1 across its vertical center axis to get the image with a value of 2, and then rotating this image in 90-degree jumps clockwise to get the values of 7, 4, and 5, respectively).



⊟ *Listing of standard Exif meta tags*

- `ImageWidth`
- `ImageLength`
- `BitsPerSample`
- `Compression`
- `PhotometricInterpretation`
- `Orientation`
- `SamplesPerPixel`
- `PlanarConfiguration`
- `YCbCrSubSampling`
- `YCbCrPositioning`
- `XResolution`
- `YResolution`
- `ResolutionUnit`
- `StripOffsets`
- `RowsPerStrip`
- `StripByteCounts`
- `JPEGInterchangeFormat`
- `JPEGInterchangeFormatLength`
- `TransferFunction`
- `WhitePoint`
- `PrimaryChromaticities`
- `YCbCrCoefficients`
- `ReferenceBlackWhite`
- `DateTime`
- `ImageDescription`
- `Make`

- `Model`
- `Software`
- `Artist`
- `Copyright`

`------------------------------`

- `ExifVersion`
- `FlashpixVersion`
- `ColorSpace`
- `ComponentsConfiguration`
- `CompressedBitsPerPixel`
- `PixelXDimension`
- `PixelYDimension`
- `MakerNote`
- `UserComment`
- `RelatedSoundFile`
- `DateTimeOriginal`
- `DateTimeDigitized`
- `SubSecTime`
- `SubSecTimeOriginal`
- `SubSecTimeDigitized`
- `ExposureTime`
- `FNumber`
- `ExposureProgram`
- `SpectralSensitivity`
- `ISOSpeedRatings`
- `OECF`
- `ShutterSpeedValue`
- `ApertureValue`
- `BrightnessValue`
- `ExposureBiasValue`
- `MaxApertureValue`
- `SubjectDistance`
- `MeteringMode`
- `LightSource`
- `Flash`
- `FocalLength`
- `SubjectArea`
- `FlashEnergy`
- `SpatialFrequencyResponse`
- `FocalPlaneXResolution`
- `FocalPlaneYResolution`
- `FocalPlaneResolutionUnit`
- `SubjectLocation`
- `ExposureIndex`
- `SensingMethod`
- `FileSource`
- `SceneType`
- `CFAPattern`
- `CustomRendered`
- `ExposureMode`
- `WhiteBalance`
- `DigitalZoomRatio`
- `FocalLengthIn35mmFilm`
- `SceneCaptureType`

- `GainControl`
- `Contrast`
- `Saturation`
- `Sharpness`
- `DeviceSettingDescription`
- `SubjectDistanceRange`
- `ImageUniqueID`

`------------------------------`

- `GPSVersionID`
- `GPSLatitudeRef`
- `GPSLatitude`
- `GPSLongitudeRef`
- `GPSLongitude`
- `GPSAltitudeRef`
- `GPSAltitude`
- `GPSTimeStamp`
- `GPSSatellites`
- `GPSStatus`
- `GPSMeasureMode`
- `GPSDOP`
- `GPSSpeedRef`
- `GPSSpeed`
- `GPSTrackRef`
- `GPSTrack`
- `GPSImgDirectionRef`
- `GPSImgDirection`
- `GPSMapDatum`
- `GPSDestLatitudeRef`
- `GPSDestLatitude`
- `GPSDestLongitudeRef`
- `GPSDestLongitude`
- `GPSDestBearingRef`
- `GPSDestBearing`
- `GPSDestDistanceRef`
- `GPSDestDistance`
- `GPSProcessingMethod`
- `GPSAreaInformation`
- `GPSDateStamp`
- `GPSDifferential`

[**Top** 1141]

18.2.2.1.5 XPath/XQuery Functions: Numeric

Altova's numeric extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

Auto-numbering functions

▼ generate-auto-number [altova:]

`altova:generate-auto-number(ID` *`as xs:string`*`,` **`StartsWith`** *`as xs:double`*`,` **`Increment`** *`as xs:double`*`,` **`ResetOnChange`** *`as xs:string`***`)`** as `xs:integer` **XP1 XP2 XQ1 XP3.1 XQ3.1**

Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the `StartsWith` argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the `Increment` argument. In effect, the `altova:generate-auto-number` function creates a counter having a name specified by the `ID` argument, with this counter being incremented each time the function is called. If the value of the `ResetOnChange` argument changes from that of the previous function call, then the value of the number to be generated is reset to the `StartsWith` value. Auto-numbering can also be reset by using the `altova:reset-auto-number` function.

⊟ *Examples*

- **`altova:generate-auto-number`**`("ChapterNumber", 1, 1, "SomeString")` will return one number each time the function is called, starting with `1`, and incrementing by `1` with each call to the function. As long as the fourth argument remains `"SomeString"` in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called `ChapterNumber`) will reset to 1. The value of `ChapterNumber` can also be reset by a call to the `altova:reset-auto-number` function, like this: `altova:reset-auto-number("ChapterNumber")`.

▼ reset-auto-number [altova:]

`altova:reset-auto-number(ID` *`as xs:string`***`)`** **XP1 XP2 XQ1 XP3.1 XQ3.1**

This function resets the number of the auto-numbering counter named in the `ID` argument. The number is reset to the number specified by the `StartsWith` argument of the `altova:generate-auto-number` function that created the counter named in the `ID` argument.

⊟ *Examples*

- **`altova:reset-auto-number`**`("ChapterNumber")` resets the number of the auto-numbering counter named `ChapterNumber` that was created by the `altova:generate-auto-number` function. The number is reset to the value of the `StartsWith` argument of the `altova:generate-auto-number` function that created `ChapterNumber`.

[Top 1145]

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 1145]

[Top 1145]

18.2.2.1.6 XPath/XQuery Functions: Schema

The Altova extension functions listed below return schema information. Given below are descriptions of the functions, together with (i) examples and (ii) a listing of schema components and their respective properties.

They can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines and are available in XPath/XQuery contexts.

Schema information from schema documents
The function **`altova:schema`** has two arguments: one with zero arguments and the other with two arguments. The zero-argument function returns the whole schema. You can then, from this starting point, navigate into the schema to locate the schema components you want. The two-argument function returns a specific component kind that is identified by its QName. In both cases, the return value is a function. To navigate into the returned component, you must select a property of that specific component. If the property is a non-atomic item (that is, if it is a component), then you can navigate further by selecting a property of this component. If the selected property is an atomic item, then the value of the item is returned and you cannot navigate any further.

Note: In XPath expressions, the schema must be imported into the processing environment (for example, into XSLT) with the **`xslt:import-schema`** instruction. In XQuery expressions, the schema must be explicitly imported using a schema import.

Schema information from XML nodes
The function **`altova:type`** submits the node of an XML document and returns the node's type information from the PSVI.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, **`http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ Schema (zero arguments)

`altova:schema() as (function(xs:string) as item()*)?` XP3.1 XQ3.1
Returns the **schema** component as a whole. You can navigate further into the **schema** component by selecting one of the **schema** component's properties.

- If this property is a component, you can navigate another step deeper by selecting one of this component's properties. This step can be repeated to navigate further into the schema.
- If the component is an atomic value, the atomic value is returned and you cannot navigate any deeper.

The properties of the **schema** component are:

```
"type definitions"
"attribute declarations"
```

```
"element declarations"
"attribute group definitions"
"model group definitions"
"notation declarations"
"identity-constraint definitions"
```

The properties of all other component kinds (besides **`schema`**) are listed below.

Note: In XQuery expressions, the schema must be explicitly imported. In XPath expressions, the schema must have been imported into the processing environment, for example, into XSLT with the **`xslt:import`** instruction.

Examples

- `import schema "" at "C:\Test\ExpReport.xsd"; for $typedef in altova:schema() ("type definitions")`
  `return $typedef ("name")` returns the names of all simple types or complex types in the schema

- `import schema "" at "C:\Test\ExpReport.xsd";`
  `altova:schema() ("type definitions")[1]("name")` returns the name of the first of all simple types or complex types in the schema

Components and their properties

Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for	

	namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)

base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

⊟ Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	

substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint	string ("key"|"unique"|"keyRef")	

category		
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value

kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

Schema (two arguments)

`altova:schema(ComponentKind as xs:string, Name as xs:QName) as (function(xs:string) as item()*)?` XP3.1 XQ3.1
Returns the component kind that is specified in the first argument which has a name that is the same as the name supplied in the second argument. You can navigate further by selecting one of the component's

properties.

- If this property is a component, you can navigate another step deeper by selecting one of this component's properties. This step can be repeated to navigate further into the schema.
- If the component is an atomic value, the atomic value is returned and you cannot navigate any deeper.

Note: In XQuery expressions, the schema must be explicitly imported. In XPath expressions, the schema must have been imported into the processing environment, for example, into XSLT with the **`xslt:import`** instruction.

⊟ *Examples*

-
  ```
  import schema "" at "C:\Test\ExpReport.xsd";
  altova:schema("element declaration", xs:QName("OrgChart"))("type definition")
  ("content type")("particles")[3]!.("term")("kind")
  ```
  returns the `kind` property of the term of the third `particles` component. This particles component is a descendant of the element declaration having a `QName` of `OrgChart`

-
  ```
  import schema "" at "C:\Test\ExpReport.xsd";
  let $typedef := altova:schema("type definition", xs:QName("emailType"))
  for $facet in $typedef ("facets")
  return [$facet ("kind"), $facet("value")]
  ```
  returns, for each **`facet`** of each **`emailType`** component, an array containing that facet's kind and value

Components and their properties

⊟ Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

⊟ Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety":	

	"fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if

		anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

⊟ Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	

nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"

name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key"|"unique"|"keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

⊟ Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

⊟ Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

⊟ XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

▼ Type

altova:type(Node *as item?***) as (function(xs:string) as item()*)?** XP3.1 XQ3.1

The function **altova:type** submits an element or attribute node of an XML document and returns the node's type information from the PSVI.

Note: The XML document must have a schema declaration so that the schema can be referenced.

Examples

-
  ```
  for $element in //Email
  let $type := altova:type($element)
  return $type
  ```
  returns a function that contains the `Email` node's type information

-
  ```
  for $element in //Email
  let $type := altova:type($element)
  return $type ("kind")
  ```
  takes the `Email` node's type component (Simple Type or Complex Type) and returns the value of the component's **kind** property

The **"_props"** parameter returns the properties of the selected component. For example:

-
  ```
  for $element in //Email
  let $type := altova:type($element)
  return ($type ("kind"), $type ("_props"))
  ```
  takes the `Email` node's type component (Simple Type or Complex Type) and returns (i) the value of the component's **kind** property, and then (ii) the properties of that component.

Components and their properties

Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties	

	("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"

name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	**Property type**	**Property value**
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	

nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"

name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key"|"unique"|"keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

18.2.2.1.7 XPath/XQuery Functions: Sequence

Altova's sequence extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ attributes [altova:]

altova:attributes(AttributeName *as xs:string***) as attribute()*** XP3.1 XQ3.1

Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis. This means that the context node must be the parent element node.

⊟ *Examples*

- **altova:attributes**("MyAttribute") returns `MyAttribute()*`

altova:attributes(AttributeName *as xs:string*, **SearchOptions** *as xs:string***) as attribute()*** XP3.1 XQ3.1

Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis. The context node must be the parent element node. The second argument is a string containing option flags. Available flags are:
r = switches to a regular-expression search; `AttributeName` must then be a regular-expression search string;
f = If this option is specified, then `AttributeName` provides a full match; otherwise `AttributeName` need only partially match an attribute name to return that attribute. For example: if **f** is not specified, then `MyAtt` will return `MyAttribute`;
i = switches to a case-insensitive search;
p = includes the namespace prefix in the search; `AttributeName` should then contain the namespace prefix, for example: `altova:MyAttribute`.
The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed as the second argument.

⊟ *Examples*

- **altova:attributes**("MyAttribute", "rfip") returns `MyAttribute()*`

- **altova:attributes**("MyAttribute", "pri") returns MyAttribute()*
- **altova:attributes**("MyAtt", "rip") returns MyAttribute()*
- **altova:attributes**("MyAttributes", "rfip") returns no match
- **altova:attributes**("MyAttribute", "") returns MyAttribute()*
- **altova:attributes**("MyAttribute", "Rip") returns an unrecognized-flag error.
- **altova:attributes**("MyAttribute",) returns a missing-second-argument error.

▼ elements [altova:]

altova:elements(ElementName *as xs:string*) **as element()*** XP3.1 XQ3.1

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The context node must be the parent node of the element/s being searched for.

⊟ *Examples*

- **altova:elements**("MyElement") returns MyElement()*

altova:elements(ElementName *as xs:string*, **SearchOptions** *as xs:string*) **as element()*** XP3.1 XQ3.1

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The context node must be the parent node of the element/s being searched for. The second argument is a string containing option flags. Available flags are:
r = switches to a regular-expression search; `ElementName` must then be a regular-expression search string;
f = If this option is specified, then `ElementName` provides a full match; otherwise `ElementName` need only partially match an element name to return that element. For example: if **f** is not specified, then `MyElem` will return `MyElement`;
i = switches to a case-insensitive search;
p = includes the namespace prefix in the search; `ElementName` should then contain the namespace prefix, for example: `altova:MyElement`.
The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed.

⊟ *Examples*

- **altova:elements**("MyElement", "rip") returns MyElement()*
- **altova:elements**("MyElement", "pri") returns MyElement()*
- **altova:elements**("MyElement", "") returns MyElement()*
- **altova:elements**("MyElem", "rip") returns MyElement()*
- **altova:elements**("MyElements", "rfip") returns no match
- **altova:elements**("MyElement", "Rip") returns an unrecognized-flag error.
- **altova:elements**("MyElement",) returns a missing-second-argument error.

▼ find-first [altova:]

altova:find-first((Sequence *as item()**), **(Condition(Sequence-Item** *as xs:boolean*)) **as item()?** XP3.1 XQ3.1

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, `Condition`, is a reference to an XPath function that takes one argument (has an arity of **`1`**) and returns a boolean. Each item of **`Sequence`** is submitted, in turn, to the function referenced in `Condition`. (*Remember:* This function takes a single argument.) The first **`Sequence`** item that causes the function in **`Condition`** to evaluate to **`true()`** is returned as the result of **`altova:find-first`**, and the iteration stops.

⊟ *Examples*

- **`altova:find-first`**`(5 to 10, function($a) {$a mod 2 = 0})` returns `xs:integer 6`
The **`Condition`** argument references the XPath 3.0 inline function, **`function()`**, which declares an inline function named **`$a`** and then defines it. Each item in the **`Sequence`** argument of **`altova:find-first`** is passed, in turn, to **`$a`** as its input value. The input value is tested on the condition in the function definition (`$a mod 2 = 0`). The first input value to satisfy this condition is returned as the result of **`altova:find-first`** (in this case **`6`**).

- **`altova:find-first`**`((1 to 10), (function($a) {$a+3=7}))` returns `xs:integer 4`

Further examples
If the file **`C:\Temp\Customers.xml`** exists:

- **`altova:find-first`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `xs:string C:\Temp\Customers.xml`

If the file **`C:\Temp\Customers.xml`** does not exist, and **`http://www.altova.com/index.html`** exists:

- **`altova:find-first`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `xs:string http://www.altova.com/index.html`

If the file **`C:\Temp\Customers.xml`** does not exist, and **`http://www.altova.com/index.html`** also does not exist:

- **`altova:find-first`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns no result

Notes about the examples given above

- The XPath 3.0 function, `doc-available`, takes a single string argument, which is used as a URI, and returns `true` if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The `doc-available` function can be used for **`Condition`**, the second argument of `altova:find-first`, because it takes only one argument (arity=1), because it takes an `item()` as input (a string which is used as a URI), and returns a boolean value.
- Notice that the `doc-available` function is only referenced, not called. The `#1` suffix that is attached to it indicates a function with an arity of 1. In its entirety `doc-available#1` simply means: *Use the doc-availabe() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence*. As a result, each of the two strings will be passed to **`doc-available()`**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **`doc-available()`** evaluates to `true()` and that string is returned as the result of the **`altova:find-first`** function. *Note about the doc-available() function: Relative*

paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.

▼ find-first-combination [altova:]

altova:find-first-combination((Seq-01 *as item()**)**, (Seq-02** *as item()**)**, (Condition(Seq-01-Item, Seq-02-Item** *as xs:boolean*)**) as item()*** XP3.1 XQ3.1

This function takes three arguments:

- The first two arguments, **Seq-01** and **Seq-02**, are sequences of one or more items of any datatype.
- The third argument, **Condition**, is a reference to an XPath function that takes two arguments (has an arity of **2**) and returns a boolean.

The items of **Seq-01** and **Seq-02** are passed in ordered pairs (one item from each sequence making up a pair) as the arguments of the function in **Condition**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X1 Y2), (X1 Y3) ... (X1 Yn), (X2 Y1), (X2 Y2) ... (Xn Yn)
```

The first ordered pair that causes the **Condition** function to evaluate to **true()** is returned as the result of **altova:find-first-combination**. Note that: (i) If the **Condition** function iterates through the submitted argument pairs and does not once evaluate to **true()**, then **altova:find-first-combination** returns *No results*; (ii) The result of **altova:find-first-combination** will always be a pair of items (of any datatype) or no item at all.

⊟ *Examples*

- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns the sequence of `xs:integers (11, 21)`
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns the sequence of `xs:integers (11, 22)`
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 34})` returns the sequence of `xs:integers (11, 23)`

▼ find-first-pair [altova:]

altova:find-first-pair((Seq-01 *as item()**)**, (Seq-02** *as item()**)**, (Condition(Seq-01-Item, Seq-02-Item** *as xs:boolean*)**) as item()*** XP3.1 XQ3.1

This function takes three arguments:

- The first two arguments, **Seq-01** and **Seq-02**, are sequences of one or more items of any datatype.
- The third argument, **Condition**, is a reference to an XPath function that takes two arguments (has an arity of **2**) and returns a boolean.

The items of **Seq-01** and **Seq-02** are passed in ordered pairs as the arguments of the function in **Condition**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The first ordered pair that causes the **`Condition`** function to evaluate to **`true()`** is returned as the result of **`altova:find-first-pair`**. Note that: (i) If the **`Condition`** function iterates through the submitted argument pairs and does not once evaluate to **`true()`**, then **`altova:find-first-pair`** returns *No results*; (ii) The result of **`altova:find-first-pair`** will always be a pair of items (of any datatype) or no item at all.

⊟ *Examples*

- **`altova:find-first-pair`**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns the sequence of `xs:integers (11, 21)`
- **`altova:find-first-pair`**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns *No results*

Notice from the two examples above that the ordering of the pairs is: `(11, 21) (12, 22) (13, 23)...(20, 30)`. This is why the second example returns *No results* (because no ordered pair gives a sum of `33`).

▼ find-first-pair-pos [altova:]

`altova:find-first-pair-pos((Seq-01` *`as item()*`***`), (Seq-02`** *`as item()*`***`), (Condition( Seq-01-Item, Seq-02-Item`** *`as xs:boolean`***`)) as xs:integer`** **XP3.1** **XQ3.1**

This function takes three arguments:

- The first two arguments, **`Seq-01`** and **`Seq-02`**, are sequences of one or more items of any datatype.
- The third argument, **`Condition`**, is a reference to an XPath function that takes two arguments (has an arity of **`2`**) and returns a boolean.

The items of **`Seq-01`** and **`Seq-02`** are passed in ordered pairs as the arguments of the function in **`Condition`**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The index position of the first ordered pair that causes the **`Condition`** function to evaluate to **`true()`** is returned as the result of **`altova:find-first-pair-pos`**. Note that if the **`Condition`** function iterates through the submitted argument pairs and does not once evaluate to **`true()`**, then **`altova:find-first-pair-pos`** returns *No results*.

⊟ *Examples*

- **`altova:find-first-pair-pos`**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns `1`
- **`altova:find-first-pair-pos`**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns *No results*

Notice from the two examples above that the ordering of the pairs is: `(11, 21) (12, 22) (13,` `23)...(20, 30)`. In the first example, the first pair causes the **`Condition`** function to evaluate to **`true()`**, and so its index position in the sequence, **`1`**, is returned. The second example returns *No results* because no pair gives a sum of `33`.

▼ find-first-pos [altova:]

`altova:find-first-pos((Sequence` *`as item()*`*`), (`**`Condition( Sequence-Item`** *`as xs:boolean`*`))`
`as xs:integer` **XP3.1** **XQ3.1**

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, `Condition`, is a reference to an XPath function that takes one argument (has an arity of **`1`**) and returns a boolean. Each item of **`Sequence`** is submitted, in turn, to the function referenced in `Condition`. (*Remember:* This function takes a single argument.) The first **`Sequence`** item that causes the function in **`Condition`** to evaluate to **`true()`** has its index position in **`Sequence`** returned as the result of **`altova:find-first-pos`**, and the iteration stops.

⊟ *Examples*

- **`altova:find-first-pos`**`(5 to 10, function($a) {$a mod 2 = 0})` returns `xs:integer 2`
The **`Condition`** argument references the XPath 3.0 inline function, **`function()`**, which declares an inline function named **`$a`** and then defines it. Each item in the **`Sequence`** argument of **`altova:find-first-pos`** is passed, in turn, to **`$a`** as its input value. The input value is tested on the condition in the function definition (`$a mod 2 = 0`). The index position in the sequence of the first input value to satisfy this condition is returned as the result of **`altova:find-first-pos`** (in this case **`2`**, since **`6`**, the first value (in the sequence) to satisfy the condition, is at index position **`2`** in the sequence).

- **`altova:find-first-pos`**`((2 to 10), (function($a) {$a+3=7}))` returns `xs:integer 3`

Further examples

If the file **`C:\Temp\Customers.xml`** exists:

- **`altova:find-first-pos`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `1`

If the file **`C:\Temp\Customers.xml`** does not exist, and **`http://www.altova.com/index.html`** exists:

- **`altova:find-first-pos`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `2`

If the file **`C:\Temp\Customers.xml`** does not exist, and **`http://www.altova.com/index.html`** also does not exist:

- **`altova:find-first-pos`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns no result

Notes about the examples given above

- The XPath 3.0 function, `doc-available`, takes a single string argument, which is used as a URI, and returns `true` if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)

- The `doc-available` function can be used for **`Condition`**, the second argument of `altova:find-first-pos`, because it takes only one argument (arity=1), because it takes an `item()` as input (a string which is used as a URI), and returns a boolean value.
- Notice that the `doc-available` function is only referenced, not called. The `#1` suffix that is attached to it indicates a function with an arity of 1. In its entirety `doc-available#1` simply means: *Use the doc-availabe() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence*. As a result, each of the two strings will be passed to **`doc-available()`**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **`doc-available()`** function evaluates to `true()` and the index position of that string in the sequence is returned as the result of the **`altova:find-first-pos`** function. *Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

for-each-attribute-pair [altova:]

`altova:for-each-attribute-pair(Seq1` *`as element()?`*, **`Seq2`** *`as element()?`*, **`Function`** *`as function()`*`)` **`as item()*`** **XP3.1** **XQ3.1**

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then the pair is "disjoint", meaning that it consists of one member only. The function item (third argument `Function`) is applied separately to each pair in the sequence of pairs (joint and disjoint), resulting in an output that is a sequence of items.

Examples

- **`altova:for-each-attribute-pair`**`(/Example/Test-A, /Example/Test-B, function($a, $b){$a+b})` returns ...

  `(2, 4, 6)` if
  ```
  <Test-A att1="1" att2="2" att3="3" />
  <Test-B att1="1" att2="2" att3="3" />
  ```

  `(2, 4, 6)` if
  ```
  <Test-A att2="2" att1="1" att3="3" />
  <Test-B att3="3" att2="2" att1="1" />
  ```

  `(2, 6)` if
  ```
  <Test-A att4="4" att1="1" att3="3" />
  <Test-B att3="3" att2="2" att1="1" />
  ```

  Note: The result `(2, 6)` *is obtained by way of the following action:* `(1+1, ()+2, 3+3, 4+())`*. If one of the operands is the empty sequence, as in the case of items 2 and 4, then the result of the addition is an empty sequence.*

- **`altova:for-each-attribute-pair`**`(/Example/Test-A, /Example/Test-B, concat#2)` returns ...

`(11, 22, 33)` if

```
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />
```

`(11, 2, 33, 4)` if

```
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />
```

for-each-combination [altova:]

`altova:for-each-combination(FirstSequence` *`as item()*`*, **`SecondSequence`** *`as item()*`*, **`Function($i,$j){$i || $j} )`** `as item()*` XP3.1 XQ3.1

The items of the two sequences in the first two arguments are combined so that each item of the first sequence is combined, in order, once with each item of the second sequence. The function given as the third argument is applied to each combination in the resulting sequence, resulting in an output that is a sequence of items (*see example*).

Examples

- `altova:for-each-combination( ('a', 'b', 'c'), ('1', '2', '3'), function($i, $j) {$i || $j} )` returns `('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3')`

for-each-matching-attribute-pair [altova:]

`altova:for-each-matching-attribute-pair(Seq1` *`as element()?`*, **`Seq2`** *`as element()?`*, **`Function`** *`as function()`***`)`** `as item()*` XP3.1 XQ3.1

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then no pair is built. The function item (third argument `Function`) is applied separately to each pair in the sequence of pairs, resulting in an output that is a sequence of items.

Examples

- `altova:for-each-matching-attribute-pair(/Example/Test-A, /Example/Test-B, function($a, $b){$a+b})` returns ...

  `(2, 4, 6)` if

  ```
  <Test-A att1="1" att2="2" att3="3" />
  <Test-B att1="1" att2="2" att3="3" />
  ```

  `(2, 4, 6)` if

  ```
  <Test-A att2="2" att1="1" att3="3" />
  <Test-B att3="3" att2="2" att1="1" />
  ```

  `(2, 6)` if

  ```
  <Test-A att4="4" att1="1" att3="3" />
  <Test-B att3="3" att2="2" att3="1" />
  ```

- `altova:for-each-matching-attribute-pair(/Example/Test-A, /Example/Test-B, concat#2)` returns ...

`(11, 22, 33)` if

```
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />
```

`(11, 33)` if

```
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />
```

▼ substitute-empty [altova:]

altova:substitute-empty(FirstSequence *as item()**, **SecondSequence** *as item()*) **as item()***
XP3.1 **XQ3.1**
If `FirstSequence` is empty, returns `SecondSequence`. If `FirstSequence` is not empty, returns `FirstSequence`.

⊟ *Examples*

- `altova:substitute-empty( (1,2,3), (4,5,6) )` returns `(1,2,3)`
- `altova:substitute-empty( (), (4,5,6) )` returns `(4,5,6)`

18.2.2.1.8 XPath/XQuery Functions: String

Altova's string extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace**, **`http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	**XP1** **XP2** **XP3.1**
XSLT functions (used in XPath expressions in XSLT):	**XSLT1** **XSLT2** **XSLT3**
XQuery functions (used in XQuery expressions in XQuery):	**XQ1** **XQ3.1**

▼ camel-case [altova:]

altova:camel-case(InputString *as xs:string*) **as xs:string** **XP3.1** **XQ3.1**
Returns the input string **`InputString`** in CamelCase. The string is analyzed using the regular expression

`'\s'` (which is a shortcut for the whitespace character). The first non-whitespace character after a whitespace or sequence of consecutive whitespaces is capitalized. The first character in the output string is capitalized.

⊟ *Examples*

- `altova:camel-case("max")` returns `Max`
- `altova:camel-case("max max")` returns `Max Max`
- `altova:camel-case("file01.xml")` returns `File01.xml`
- `altova:camel-case("file01.xml file02.xml")` returns `File01.xml File02.xml`
- `altova:camel-case("file01.xml   file02.xml")` returns `File01.xml   File02.xml`
- `altova:camel-case("file01.xml -file02.xml")` returns `File01.xml -file02.xml`

altova:camel-case(InputString *as xs:string*, **SplitChars** *as xs:string*, **IsRegex** *as xs:boolean*) **as xs:string** **XP3.1** **XQ3.1**

Converts the input string **InputString** to camel case by using **SplitChars** to determine the character/s that trigger the next capitalization. **SplitChars** is used as a regular expression when **IsRegex = true()**, or as plain characters when **IsRegex = false()**. The first character in the output string is capitalized.

⊟ *Examples*

- `altova:camel-case("setname getname", "set|get", true())` returns `setName getName`
- `altova:camel-case("altova\documents\testcases", "\", false())` returns `Altova\Documents\Testcases`

▼ char [altova:]

altova:char(Position *as xs:integer*) **as xs:string** **XP3.1** **XQ3.1**

Returns a string containing the character at the position specified by the `Position` argument, in the string obtained by converting the value of the context item to `xs:string`. The result string will be empty if no character exists at the index submitted by the `Position` argument.

⊟ *Examples*

If the context item is **1234ABCD**:

- `altova:char(2)` returns `2`
- `altova:char(5)` returns `A`
- `altova:char(9)` returns the empty string.
- `altova:char(-2)` returns the empty string.

altova:char(InputString *as xs:string*, **Position** *as xs:integer*) **as xs:string** **XP3.1** **XQ3.1**

Returns a string containing the character at the position specified by the `Position` argument, in the string submitted as the `InputString` argument. The result string will be empty if no character exists at the index submitted by the `Position` argument.

⊟ *Examples*

- `altova:char("2014-01-15", 5)` returns `-`
- `altova:char("USA", 1)` returns `U`
- `altova:char("USA", 10)` returns the empty string.
- `altova:char("USA", -2)` returns the empty string.

▼ create-hash-from-string[altova:]

altova:create-hash-from-string(InputString *as xs:string***) as xs:string** XP2 XQ1 XP3.1 XQ3.1
altova:create-hash-from-string(InputString *as xs:string*, **HashAlgo** *as xs:string***) as xs:string** XP2 XQ1 XP3.1 XQ3.1
Generates a hash string from `InputString` by using the hashing algorithm specified by the `HashAlgo` argument. The following hashing algorithms may be specified (in upper or lower case): **MD5**, **SHA-1**, **SHA-224**, **SHA-256**, **SHA-384**, **SHA-512**. If the second argument is not specified (*see the first signature above*), then the **SHA-256** hashing algorithm is used.

⊟ *Examples*

- `altova:create-hash-from-string('abc')` returns a hash string generated by using the **SHA-256** hashing algorithm.
- `altova:create-hash-from-string('abc', 'md5')` returns a hash string generated by using the **MD5** hashing algorithm.
- `altova:create-hash-from-string('abc', 'MD5')` returns a hash string generated by using the **MD5** hashing algorithm.

▼ first-chars [altova:]

altova:first-chars(X-Number *as xs:integer***) as xs:string** XP3.1 XQ3.1
Returns a string containing the first `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

⊟ *Examples*

If the context item is **1234ABCD**:

- `altova:first-chars(2)` returns `12`
- `altova:first-chars(5)` returns `1234A`
- `altova:first-chars(9)` returns `1234ABCD`

altova:first-chars(InputString *as xs:string*, **X-Number** *as xs:integer***) as xs:string** XP3.1 XQ3.1
Returns a string containing the first `X-Number` of characters of the string submitted as the `InputString` argument.

⊟ *Examples*

- `altova:first-chars("2014-01-15", 5)` returns `2014-`
- `altova:first-chars("USA", 1)` returns `U`

▼ format-string [altova:]

altova:format-string(InputString *as xs:string*, **FormatSequence** *as item()****) as xs:string** XP3.1 XQ3.1
The input string (first argument) contains positional parameters (`%1`, `%2`, etc). Each parameter is replaced by the string item that is located at the corresponding position in the format sequence (submitted as the second argument). So the first item in the format sequence replaces the positional parameter `%1`, the second item replaces `%2`, and so on. The function returns this formatted string that contains the replacements. If no string exists for a positional parameter, then the positional parameter itself is returned. This happens when the index of a positional parameter is greater than the number of items in the format sequence.

⊟ *Examples*

- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe'))` returns `"Hello Jane, John, Joe"`
- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe', 'Tom'))` returns `"Hello Jane, John, Joe"`
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe', 'Tom'))` returns `"Hello Jane, John, Tom"`
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe'))` returns `"Hello Jane, John, %4"`

▼ last-chars [altova:]

`altova:last-chars(X-Number` *`as xs:integer`*) **`as xs:string`** **XP3.1** **XQ3.1**

Returns a string containing the last `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

⊟ *Examples*

If the context item is **`1234ABCD`**:

- `altova:last-chars(2)` returns `CD`
- `altova:last-chars(5)` returns `4ABCD`
- `altova:last-chars(9)` returns `1234ABCD`

`altova:last-chars(InputString` *`as xs:string`*, **`X-Number`** *`as xs:integer`*) **`as xs:string`** **XP3.1** **XQ3.1**

Returns a string containing the last `X-Number` of characters of the string submitted as the `InputString` argument.

⊟ *Examples*

- `altova:last-chars("2014-01-15", 5)` returns `01-15`
- `altova:last-chars("USA", 10)` returns `USA`

▼ pad-string-left [altova:]

`altova:pad-string-left(StringToPad` *`as xs:string`*, **`StringLength`** *`as xs:integer`*, **`PadCharacter`** *`as xs:string`*) **`as xs:string`** **XP3.1** **XQ3.1**

The `PadCharacter` argument is a single character. It is padded to the left of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad`. has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

⊟ *Examples*

- `altova:pad-string-left('AP', 1, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 2, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 3, 'Z')` returns `'ZAP'`
- `altova:pad-string-left('AP', 4, 'Z')` returns `'ZZAP'`
- `altova:pad-string-left('AP', -3, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 3, 'YZ')` returns a pad-character-too-long error

pad-string-right [altova:]

altova:pad-string-right(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string*) **as xs:string** **XP3.1** **XQ3.1**

The `PadCharacter` argument is a single character. It is padded to the right of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad` has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

Examples

- `altova:pad-string-right('AP', 1, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 2, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 3, 'Z')` returns `'APZ'`
- `altova:pad-string-right('AP', 4, 'Z')` returns `'APZZ'`
- `altova:pad-string-right('AP', -3, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 3, 'YZ')` returns a pad-character-too-long error

repeat-string [altova:]

altova:repeat-string(InputString *as xs:string*, **Repeats** *as xs:integer*) **as xs:string** **XP2** **XQ1** **XP3.1** **XQ3.1**

Generates a string that is composed of the first `InputString` argument repeated `Repeats` number of times.

Examples

- `altova:repeat-string("Altova #", 3)` returns "`Altova #Altova #Altova #`"

substring-after-last [altova:]

altova:substring-after-last(MainString *as xs:string*, **CheckString** *as xs:string*) **as xs:string** **XP3.1** **XQ3.1**

If `CheckString` is found in `MainString`, then the substring that occurs after `CheckString` in `MainString` is returned. If `CheckString` is not found in `MainString`, then the empty string is returned. If `CheckString` is an empty string, then `MainString` is returned in its entirety. If there is more than one occurrence of `CheckString` in `MainString`, then the substring after the last occurrence of `CheckString` is returned.

Examples

- `altova:substring-after-last('ABCDEFGH', 'B')` returns `'CDEFGH'`
- `altova:substring-after-last('ABCDEFGH', 'BC')` returns `'DEFGH'`
- `altova:substring-after-last('ABCDEFGH', 'BD')` returns `''`
- `altova:substring-after-last('ABCDEFGH', 'Z')` returns `''`
- `altova:substring-after-last('ABCDEFGH', '')` returns `'ABCDEFGH'`
- `altova:substring-after-last('ABCD-ABCD', 'B')` returns `'CD'`
- `altova:substring-after-last('ABCD-ABCD-ABCD', 'BCD')` returns `''`

substring-before-last [altova:]

altova:substring-before-last(MainString *as xs:string*, **CheckString** *as xs:string*) **as xs:string** **XP3.1** **XQ3.1**

If `CheckString` is found in `MainString`, then the substring that occurs before `CheckString` in `MainString` is returned. If `CheckString` is not found in `MainString`, or if `CheckString` is an empty string, then the empty string is returned. If there is more than one occurrence of `CheckString` in `MainString`, then the substring before the last occurrence of `CheckString` is returned.

⊟ *Examples*

- `altova:substring-before-last('ABCDEFGH', 'B')` returns `'A'`
- `altova:substring-before-last('ABCDEFGH', 'BC')` returns `'A'`
- `altova:substring-before-last('ABCDEFGH', 'BD')` returns `''`
- `altova:substring-before-last('ABCDEFGH', 'Z')` returns `''`
- `altova:substring-before-last('ABCDEFGH', '')` returns `''`
- `altova:substring-before-last('ABCD-ABCD', 'B')` returns `'ABCD-A'`
- `altova:substring-before-last('ABCD-ABCD-ABCD', 'ABCD')` returns `'ABCD-ABCD-'`

▼ substring-pos [altova:]

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string***) as xs:integer** XP3.1 XQ3.1

Returns the character position of the first occurrence of `StringToFind` in the string `StringToCheck`. The character position is returned as an integer. The first character of `StringToCheck` has the position `1`. If `StringToFind` does not occur within `StringToCheck`, the integer `0` is returned. To check for the second or a later occurrence of `StringToCheck`, use the next signature of this function.

⊟ *Examples*

- `altova:substring-pos('Altova', 'to')` returns `3`
- `altova:substring-pos('Altova', 'tov')` returns `3`
- `altova:substring-pos('Altova', 'tv')` returns `0`
- `altova:substring-pos('AltovaAltova', 'to')` returns `3`

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*, **Integer** *as xs:integer***) as xs:integer** XP3.1 XQ3.1

Returns the character position of `StringToFind` in the string, `StringToCheck`. The search for `StringToFind` starts from the character position given by the `Integer` argument; the character substring before this position is not searched. The returned integer, however, is the position of the found string within the *entire* string, `StringToCheck`. This signature is useful for finding the second or a later position of a string that occurs multiple times with the `StringToCheck`. If `StringToFind` does not occur within `StringToCheck`, the integer `0` is returned.

⊟ *Examples*

- `altova:substring-pos('Altova', 'to', 1)` returns `3`
- `altova:substring-pos('Altova', 'to', 3)` returns `3`
- `altova:substring-pos('Altova', 'to', 4)` returns `0`
- `altova:substring-pos('Altova-Altova', 'to', 0)` returns `3`
- `altova:substring-pos('Altova-Altova', 'to', 4)` returns `10`

▼ trim-string [altova:]

altova:trim-string(InputString *as xs:string***) as xs:string** XP3.1 XQ3.1

This function takes an `xs:string` argument, removes any leading and trailing whitespace, and returns a "trimmed" `xs:string`.

⊟ *Examples*

- `altova:trim-string("   Hello World   ")` returns `"Hello World"`
- `altova:trim-string("Hello World   ")` returns `"Hello World"`
- `altova:trim-string("   Hello World")` returns `"Hello World"`
- `altova:trim-string("Hello World")` returns `"Hello World"`
- `altova:trim-string("Hello   World")` returns `"Hello   World"`

▼ trim-string-left [altova:]

`altova:trim-string-left(InputString` *`as xs:string`***`) as xs:string`** **XP3.1** **XQ3.1**

This function takes an `xs:string` argument, removes any leading whitespace, and returns a left-trimmed `xs:string`.

⊟ *Examples*

- `altova:trim-string-left("   Hello World   ")` returns `"Hello World   "`
- `altova:trim-string-left("Hello World   ")` returns `"Hello World   "`
- `altova:trim-string-left("   Hello World")` returns `"Hello World"`
- `altova:trim-string-left("Hello World")` returns `"Hello World"`
- `altova:trim-string-left("Hello   World")` returns `"Hello   World"`

▼ trim-string-right [altova:]

`altova:trim-string-right(InputString` *`as xs:string`***`) as xs:string`** **XP3.1** **XQ3.1**

This function takes an `xs:string` argument, removes any trailing whitespace, and returns a right-trimmed `xs:string`.

⊟ *Examples*

- `altova:trim-string-right("   Hello World   "))` returns `"   Hello World"`
- `altova:trim-string-right("Hello World   "))` returns `"Hello World"`
- `altova:trim-string-right("   Hello World"))` returns `"   Hello World"`
- `altova:trim-string-right("Hello World"))` returns `"Hello World"`
- `altova:trim-string-right("Hello   World"))` returns `"Hello   World"`

18.2.2.1.9 XPath/XQuery Functions: Miscellaneous

The following general purpose XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace,** **`http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ decode-string [altova:]

altova:decode-string(Input *as xs:base64Binary***) as xs:string** XP3.1 XQ3.1
altova:decode-string(Input *as xs:base64Binary*, **Encoding** *as xs:string***) as xs:string** XP3.1 XQ3.1
Decodes the submitted base64Binary input to a string using the specified encoding. If no encoding is specified, then the UTF-8 encoding is used. The following encodings are supported: `US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4`

⊟ *Examples*

- `altova:decode-string($XML1/MailData/Meta/b64B)` returns the base64Binary input as a UTF-8 encoded string
- `altova:decode-string($XML1/MailData/Meta/b64B, "UTF-8")` returns the base64Binary input as a UTF-8-encoded string
- `altova:decode-string($XML1/MailData/Meta/b64B, "ISO-8859-1")` returns the base64Binary input as an ISO-8859-1-encoded string

▼ encode-string [altova:]

altova:encode-string(InputString *as xs:string***) as xs:base64Binaryinteger** XP3.1 XQ3.1
altova:encode-string(InputString *as xs:string*, **Encoding** *as xs:string***) as xs:base64Binaryinteger** XP3.1 XQ3.1
Encodes the submitted string using, if one is given, the specified encoding. If no encoding is given, then the UTF-8 encoding is used. The encoded string is converted to base64Binary characters, and the converted base64Binary value is returned. Initially, UTF-8 encoding is supported, and support will be extended to the following encodings: `US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4`

⊟ *Examples*

- `altova:encode-string("Altova")` returns the base64Binary equivalent of the UTF-8 encoded string `"Altova"`
- `altova:encode-string("Altova", "UTF-8")` returns the base64Binary equivalent of the UTF-8 encoded string `"Altova"`

▼ get-temp-folder [altova:]

altova:get-temp-folder() as `xs:string` XP2 XQ1 XP3.1 XQ3.1
This function takes no argument. It returns the path to the temporary folder of the current user.

⊟ *Examples*

- `altova:get-temp-folder()` would return, on a Windows machine, something like `C:\Users\<UserName>\AppData\Local\Temp\` as an `xs:string`.

▼ generate-guid [altova:]

`altova:generate-guid() as xs:string` XP2 XQ1 XP3.1 XQ3.1

Generates a unique string GUID string.

⊟ *Examples*

- `altova:generate-guid()` returns (for example) `85F971DA-17F3-4E4E-994E-99137873ACCD`

▼ high-res-timer [altova:]

`altova:high-res-timer() as xs:double` XP3.1 XQ3.1

Returns a system high-resolution timer value in seconds. A high-resolution timer, when present on a system, enables high precision time measurements when these are required (for example, in animations and for determining precise code-execution time). This function provides the resolution of the system's high-res timer.

⊟ *Examples*

- `altova:high-res-timer()` returns something like `'1.16766146154566E6'`

▼ parse-html [altova:]

`altova:parse-html(HTMLText` *`as xs:string`***`) as node()`** XP3.1 XQ3.1

The `HTMLText` argument is a string that contains the text of an HTML document. The function creates an HTML tree from the string. The submitted string may or may not contain the HTML element. In either case, the root element of the tree is an element named **`HTML`**. It is best to make sure that the HTML code in the submitted string is valid HTML.

⊟ *Examples*

- `altova:parse-html("<html><head/><body><h1>Header</h1></body></html>")` creates an HTML tree from the submitted string

▼ sleep[altova:]

`altova:sleep(Millisecs` *`as xs:integer`***`) as empty-sequence()`** XP2 XQ1 XP3.1 XQ3.1

Suspends execution of the current operation for the number of milliseconds given by the `Millisecs` argument.

⊟ *Examples*

- `altova:sleep(1000)` suspends execution of the current operation for 1000 milliseconds.

[Top 1181]

18.2.2.2 Miscellaneous Extension Functions

There are several ready-made functions in programming languages such as Java and C# that are not available as XQuery/XPath functions or as XSLT functions. A good example would be the math functions available in Java, such as `sin()` and `cos()`. If these functions were available to the designers of XSLT stylesheets and XQuery queries, it would increase the application area of stylesheets and queries and greatly simplify the tasks of stylesheet creators. The XSLT and XQuery engines used in a number of Altova products support the use of extension functions in Java 1184 and .NET 1193, as well as MSXSL scripts for XSLT 1199. 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 1184
- .NET Extension Functions 1193
- MSXSL Scripts for XSLT 1199

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.

18.2.2.2.1 Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- Java: Constructors 1189
- Java: Static Methods and Static Fields 1190
- Java: Instance Methods and Instance Fields 1191
- Datatypes: XPath/XQuery to Java 1191
- Datatypes: Java to XPath/XQuery 1192

Note the following

- If you are using an Altova desktop product, the Altova application attempts to detect the path to the Java virtual machine automatically, by reading (in this order): (i) the Windows registry, and (ii) the **`JAVA_HOME`** environment variable. You can also add a custom path in the Options dialog of the application; this entry will take priority over any other Java VM path detected automatically.
- If you are running an Altova server product on a Windows machine, the path to the Java virtual machine will be read first from the Windows registry; if this is not successful the **`JAVA_HOME`** environment variable will be used.

- If you are running an Altova server product on a Linux or macOS machine, then make sure that the **`JAVA_HOME`** environment variable is properly set and that the Java Virtual Machines library (on Windows, the **`jvm.dll`** file) can be located in either the **`\bin\server`** or **`\bin\client`** directory.

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with `java:` (*see below for examples*). The namespace declaration should identify a Java class, for example: `xmlns:myns="java:java.lang.Math"`. However, it could also simply be: `xmlns:myns="java"` (without a colon), with the identification of the Java class being left to the `fname()` part of the extension function.
- The `fname()` part identifies the Java method being called, and supplies the arguments for the method (*see below for examples*). However, if the namespace URI identified by the `prefix:` part does not identify a Java class (*see preceding point*), then the Java class should be identified in the `fname()` part, before the class and separated from the class by a period (*see the second XSLT example below*).

Note: The class being called must be on the classpath of the machine.

XSLT example

Here are two examples of how a static method can be called. In the first example, the class name (`java.lang.Math`) is included in the namespace URI and, therefore, must not be in the `fname()` part. In the second example, the `prefix:` part supplies the prefix `java:` while the `fname()` part identifies the class as well as the method.

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
     select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
     select="jmath:java.lang.Math.cos(3.14)" />
```

The method named in the extension function (`cos()` in the example above) must match the name of a public static method in the named Java class (`java.lang.Math` in the example above).

XQuery example

Here is an XQuery example similar to the XSLT example above:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

User-defined Java classes

If you have created your own Java classes, methods in these classes are called differently according to: (i) whether the classes are accessed via a JAR file or a class file, and (ii) whether these files (JAR or class) are located in the current directory (the same directory as the XSLT or XQuery document) or not. How to locate these files is described in the sections User-Defined Class Files 1186 and User-Defined Jar Files 1188. Note that paths to class files not in the current directory and to all JAR files must be specified.

18.2.2.2.1.1 User-Defined Class Files

If access is via a class file, then there are four possibilities:

- The class file is in a package. The XSLT or XQuery file is in the same folder as the Java package. (*See example below*[1186].)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. (*See example below*[1187].)
- The class file is in a package. The XSLT or XQuery file is at some random location. (*See example below*[1187].)
- The class file is not packaged. The XSLT or XQuery file is at some random location. (*See example below*[1188].)

Consider the case where the class file is not packaged and is in the same folder as the XSLT or XQuery document. In this case, since all classes in the folder are found, the file location does not need to be specified. The syntax to identify a class is:

```
java:classname
```

where

`java:` indicates that a user-defined Java function is being called; (Java classes in the current directory will be loaded by default)
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call.

Class file packaged, XSLT/XQuery file in same folder as Java package

The example below calls the `getVehicleType()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is also in the folder `JavaProject`.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:com.altova.extfunc.Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file referenced, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()`method of the `Car` class. Let us say that: (i) the `Car` class file is in the following folder: `JavaProject/com/altova/extfunc`, and (ii) that this folder is the current folder in the example below. The XSLT file is also in the folder `JavaProject/com/altova/extfunc`.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

where

`java:` indicates that a user-defined Java function is being called
`uri-of-package` is the URI of the Java package
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:com.altova.extfunc.Car?path=file:///C:/JavaProject/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>
```

```
</xsl:stylesheet>
```

Class file referenced, XSLT/XQuery file at any location

The example below calls the `getCarColor()`method of the `Car` class. Let us say that the `Car` class file is in the folder `C:/JavaProject/com/altova/extfunc`, and the XSLT file is at any location. The location of the class file must then be specified within the namespace URI as a query string. The syntax is:

`java:classname[?path=`*<uri-of-classfile>*`]`

where

`java:` indicates that a user-defined Java function is being called
`uri-of-classfile` is the URI of the folder containing the class file
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:Car?path=file:///C:/JavaProject/com/altova/extfunc/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

18.2.2.2.1.2 User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

`xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"`

The method is then called by using the prefix of the namespace URI that identifies the class:
`classNS:method()`

In the above:

`java:` indicates that a Java function is being called
`classname` is the name of the user-defined class

`?` is the separator between the classname and the path
`path=jar:` indicates that a path to a JAR file is being given
`uri-of-jarfile` is the URI of the jar file
`!/` is the end delimiter of the path
`classNS:method()` is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```
xmlns:ns1="java:docx.layout.pages?path=jar:file:///c:/projects/docs/docx.jar!/"
ns1:main()

xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
ns2:docx.layout.pages.main()
```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/2005/xpath-functions"
      xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
      <xsl:variable name="myCar" select="car:Car1.new('red')" />
      <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

18.2.2.2.1.3 Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be implicitly converted to XPath/XQuery datatypes[1192], 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*[1191]):
  ```
  <xsl:value-of select="date:toString(date:new())" xmlns:date="java:java.util.Date" />
  ```

18.2.2.2.1.4 Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples

Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`).) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery example

A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

18.2.2.2.1.5 Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()"/>
  <xsl:template match="/">
      <enrollment institution-id="Altova School"
                date="{date:toString($CurrentDate)}"
                type="{jlang:Object.toString(jlang:Object.getClass( date:new() ))}">
      </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the parameter `CurrentDate` takes the value returned by a constructor for the class `java.util.Date`. This value is then passed as an argument to the instance method `date:toString` in order to supply the value of `/enrollment/@date`.

18.2.2.2.1.6 Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected

method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (primitive), `java.lang.Boolean`
`xs:integer`	int, long, short, byte, float, double, and the wrapper classes of these, such as `java.lang.Integer`
`xs:float`	`float` (primitive), `java.lang.Float`, `double` (primitive)
`xs:double`	`double` (primitive), `java.lang.Double`
`xs:decimal`	`float` (primitive), `java.lang.Float`, `double`(primitive), `java.lang.Double`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

18.2.2.2.1.7 Datatypes: Java to XPath/XQuery

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.

One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

18.2.2.2.2 .NET Extension Functions

If you are working on the .NET platform on a Windows machine, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

- .NET: Constructors 1195
- .NET: Static Methods and Static Fields 1196
- .NET: Instance Methods and Instance Fields 1196
- Datatypes: XPath/XQuery to .NET 1197
- Datatypes: .NET to XPath/XQuery 1198

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.

18.2.2.2.2.1 .NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a `'No constructor found'` error is returned.

Constructors that return XPath/XQuery datatypes

If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes 1192, 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* 1191):
  `<xsl:value-of select="date:ToString(date:new(2008, 4, 29))" xmlns:date="clitype:System.DateTime" />`
- It can be converted to a string, number, or boolean:

- `<xsl:value-of select="xs:integer(date:get_Month(date:new(2008, 4, 29)))" xmlns:date="clitype:System.DateTime" />`

18.2.2.2.2.2 .NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

An XSLT example showing a call to a field (considered a method with no argument) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

An XSLT example showing a call to a property (syntax is `get_PropertyName()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length('my string')"
xmlns:string="clitype:System.String"/>
```

An XQuery example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

18.2.2.2.2.3 .NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:variable name="releasedate"
         select="date:new(2008, 4, 29)"
         xmlns:date="clitype:System.DateTime"/>
      <doc>
         <date>
            <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
               xmlns:date="clitype:System.DateTime"/>
         </date>
         <date>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008, 4, 29)`) as its first and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

18.2.2.2.2.4 Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

`xs:string`	`StringValue, string`
`xs:boolean`	`BooleanValue, bool`
`xs:integer`	`IntegerValue, decimal, long, integer, short, byte, double, float`
`xs:float`	`FloatValue, float, double`
`xs:double`	`DoubleValue, double`
`xs:decimal`	`DecimalValue, decimal, double, float`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

18.2.2.2.2.5 Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`.

When a .NET object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example `System.DateTime.ToString()`) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

18.2.2.2.3 MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (*see example below*).

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (*see below*).

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-prefix">

   function-1 or variable-1
   ...
   function-n or variable-n

</msxsl:script>
```

The `<msxsl:script>` element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the `<msxsl:script>` element. **The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.** Consequently, the .NET scripting languages can be used within the `<msxsl:script>` element.

The `language` attribute accepts the same values as the `language` attribute on the HTML `<script>` element. If the `language` attribute is not specified, then Microsoft JScript is assumed as the default.

The implements-prefix attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the `<msxsl:script>` element will be in the namespace identified by the prefix specified in the `implements-prefix` attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

Example

Here is an example of a complete XSLT stylesheet that uses a function defined within a `<msxsl:script>` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
    ' Input: A currency value: the wholesale price
  ' Returns: The retail price: the input value plus 20% margin,
  ' rounded to the nearest cent
  dim a as integer  = 13
  Function AddMargin(WholesalePrice) as integer
    AddMargin = WholesalePrice * 1.2 + a
  End Function
 ]]>
 </msxsl:script>

 <xsl:template match="/">
  <html>
   <body>
    <p>
     <b>Total Retail Price =
      $<xsl:value-of select="user:AddMargin(50)"/>
     </b>
     <br/>
     <b>Total Wholesale Price =
      $<xsl:value-of select="50"/>
     </b>
    </p>
   </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies

An assembly can be imported into the script by using the **`msxsl:assembly`** element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl:assembly` element is to be used.

```
<msxsl:script>
   <msxsl:assembly name="myAssembly.assemblyName" />
   <msxsl:assembly href="pathToAssembly" />

   ...

</msxsl:script>
```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral PublicKeyToken=a46b3f648229c514"
```

or a short name, such as "`myAssembly.Draw`".

Namespaces

Namespaces can be declared with the **`msxsl:using`** element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl:using` element is used so as to declare namespaces.

```
<msxsl:script>
   <msxsl:using namespace="myAssemblyNS.NamespaceName" />

   ...

</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.

18.3 Technical Data

This section contains information on some technical aspects of your software. This information is organized into the following sections:

- OS and Memory Requirements 1202
- Altova Engines 1202
- Unicode Support 1203
- Internet Usage 1203

18.3.1 OS and Memory Requirements

Operating System

Altova software applications are available for the following platforms:

- Windows 10, Windows 11
- Windows Server 2016 or newer

Memory

Since the software is written in C++ it does not require the overhead of a Java Runtime Environment and typically requires less memory than comparable Java-based applications. However, each document is loaded fully into memory so as to parse it completely and to improve viewing and editing speed. As a result, the memory requirement increases with the size of the document.

Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, available memory can rapidly be depleted.

18.3.2 Altova Engines

XML Validator

When opening an XML document, the application uses its built-in XML validator to check for well-formedness, to validate the document against a schema (if specified), and to build trees and infosets. The XML validator is also used to provide intelligent editing help while you edit documents and to dynamically display any validation error that may occur.

The built-in XML validator implements the Final Recommendation of the W3C's XML Schema 1.0 and 1.1 specifications. New developments recommended by the W3C's XML Schema Working Group are continuously being incorporated in the XML validator, so that Altova products give you a state-of-the-art development environment.

XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0, 2.0, and 3.0 Engines and the Altova XQuery 1.0 and 3.1 Engines. If one of these engines is included in the product, then documentation about implementation-specific behavior for each engine is given in the appendices of the documentation.

Note: Altova MapForce generates code using the XSLT 1.0, 2.0 and XQuery 1.0 engines.

18.3.3 Unicode Support

Altova's XML products provide full Unicode support. To edit an XML document, you will also need a font that supports the Unicode characters being used by that document.

Please note that most fonts only contain a very specific subset of the entire Unicode range and are therefore typically targeted at the corresponding writing system. If some text appears garbled, the reason could be that the font you have selected does not contain the required glyphs. So it is useful to have a font that covers the entire Unicode range, especially when editing XML documents in different languages or writing systems. A typical Unicode font found on Windows PCs is Arial Unicode MS.

In the `/Examples` folder of your application folder you will find an XHTML file called `UnicodeUTF-8.html` that contains the following sentence in a number of different languages and writing systems:

- *When the world wants to talk, it speaks Unicode*
- *Wenn die Welt miteinander spricht, spricht sie Unicode*
- 世界的に話すなら、Unicode です。)

Opening this XHTML file will give you a quick impression of Unicode's possibilities and also indicate what writing systems are supported by the fonts available on your PC.

18.3.4 Internet Usage

Altova applications will initiate Internet connections on your behalf in the following situations:

- If you click the "Request evaluation key-code" in the Registration dialog (**Help | Software Activation**), the three fields in the registration dialog box are transferred to our web server by means of a regular http (port 80) connection and the free evaluation key-code is sent back to the customer via regular SMTP e-mail.
- In some Altova products, you can open a file over the Internet (**File | Open | Switch to URL**). In this case, the document is retrieved using one of the following protocol methods and connections: HTTP (normally port 80), FTP (normally port 20/21), HTTPS (normally port 443). You could also run an HTTP server on port 8080. (In the URL dialog, specify the port after the server name and a colon.)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, the referenced schema document is also retrieved through a HTTP connection (port 80) or another protocol specified in the URL (see Point 2 above). A schema document will also be retrieved when an XML file is validated. Note that validation might happen automatically upon opening a document if you have instructed the application to do this (in the File tab of the Options dialog (**Tools | Options**)).
- In Altova applications using WSDL and SOAP, web service connections are defined by the WSDL documents.
- If you are using the **Send by Mail** command (**File | Send by Mail**) in XMLSpy, the current selection or file is sent by means of any MAPI-compliant mail program installed on the user's PC.
- As part of Software Activation and LiveUpdate as further described in the Altova Software License Agreement.

18.4 License Information

This section contains information about:

- the distribution of this software product
- software activation and license metering
- the license agreement governing the use of this product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

To view the terms of any Altova license, go to the Altova Legal Information page at the Altova website.

18.4.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge for 30 days before making a purchasing decision. *(Note: Altova MobileTogether Designer is licensed free of charge.)*
- Once you decide to buy the software, you can place your order online at the Altova website and get a fully licensed product within minutes.
- When you place an online order, you always get the latest version of our software.
- The product package includes an onscreen help system that can be accessed from within the application interface. The latest version of the user manual is available at www.altova.com in (i) HTML format for online browsing, and (ii) PDF format for download (and to print if you prefer to have the documentation on paper).

30-day evaluation period

After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into the evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you must purchase a product license, which is delivered in the form of a license file containing a key code. Unlock the product by uploading the license file in the Software Activation dialog of your product.

You can purchase product licenses at https://shop.altova.com/.

Helping Others within Your Organization to Evaluate the Software

If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may distribute only the installer file, provided that this file is not modified in any way. Any person who accesses the software installer that you have provided must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.

18.4.2 Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license

When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi-user license

If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (*see the IANA Service Name Registry for details*) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

Note about certificates

Your Altova application contacts the Altova licensing server (`link.altova.com`) via HTTPS. For this communication, Altova uses a registered SSL certificate. If this certificate is replaced (for example, by your IT department or an external agency), then your Altova application will warn you about the connection being insecure. You could use the replacement certificate to start your Altova application, but you would be doing this at your own risk. If you see a *Non-secure connection* warning message, check the origin of the certificate and consult your IT team (who would be able to decide whether the interception and replacement of the Altova certificate should continue or not).

If your organization needs to use its own certificate (for example, to monitor communication to and from client machines), then we recommend that you install Altova's free license management software, Altova LicenseServer, on your network. Under this setup, client machines can continue to use your organization's certificates, while Altova LicenseServer can be allowed to use the Altova certificate for communication with Altova.

18.4.3 Altova End-User License Agreement

- The Altova End-User License Agreement is available here: https://www.altova.com/legal/eula
- Altova's Privacy Policy is available here: https://www.altova.com/privacy

18.4.4 Packaging License Files with MapForce Installer

If you want to perform a silent installation of MapForce, you may want to modify the MSI database so that it includes your license file(s). This way, the installer will not only install the product but also license it. For details about how to achieve this, download this ZIP file from the Altova website and open the PDF document in it.

Index

.

.NET extension functions,
constructors, 1195
datatype conversions (.NET to XPath/XQuery), 1198
datatype conversions (XPath/XQuery to .NET), 1197
for XSLT and XQuery, 1193
instance methods, instance fields, 1196
static methods, static fields, 1196

A

A to Z,
sort component, 402
abs,
as MapForce function (in lang | math functions), 635
as MapForce function (in xpath2 | numeric functions), 691
Access database,
updating based on IF condition, 335
acos,
as MapForce function (in lang | math functions), 635
ActiveX,
integration at application level, 998
integration at document level, 1001
integration prerequisites, 994
ActiveX controls,
adding to the Visual Studio Toolbox, 996
add,
as MapForce function (in core | math functions), 549
ADO,
as data connection interface, 168
setting up a connection, 176
ADO.NET,
setting up a connection, 181
age,
as MapForce function (in lang | datetime functions), 609
Altova extensions,
chart functions (see chart functions), 1108
Altova XML Parser,
about, 1202
asin,
as MapForce function (in lang | math functions), 636
atan,
as MapForce function (in lang | math functions), 636
ATTLIST,
DTD namespace URIs, 132
Automated,
processing, 793
Automatic,
loading of libraries, 495
auto-number,
as MapForce function (in core | generator functions), 541
avg,
as MapForce function (in core | aggregate functions), 515
Azure SQL, 211

B

Background Information, 1202
Bars,
Application status, 23
Menu, 23
Toolbars, 23
base-uri,
as MapForce function (in xpath2 | accessors library), 662
Batch,
processing automation, 793
boolean,
as MapForce function (in core | conversion functions), 521
Breakpoints,
about, 775
adding, 778
removing, 778
Breakpoints window,
about, 775, 784

C

C#,
integration of MapForce, 1004
reference to generated classes, 944
C++,
reference to generated classes, 929
capitalize,

capitalize,
as MapForce function (in lang | string functions), 645
Catalogs, 836
customize, 841
environment variables, 843
in DTD, 837
in XML Schema, 837
structure, 839
CDATA,
section, 143
ceiling,
as MapForce function (in core | math functions), 549
char-from-code,
as MapForce function (in core | string functions), 589
charset-decode,
as MapForce function (in lang | string functions), 645
charset-encode,
as MapForce function (in lang | string functions), 647
Code,
built in types, 987
SPL, 973
Code Generation,
build, 66, 868
build code, 66, 868, 870
C#, 66, 868, 870, 876, 879
C++, 66, 868, 870, 876
change data type, 879
compile, 66, 868
generate code, 66, 868, 870
integrate generated code, 876, 879
Java, 66, 868, 870, 876, 879
modify generated code, 876, 879
modify input/output, 876
run application, 66, 868
run code, 870
XQuery, 66, 868
XSLT, 66, 868
Code generation settings,
defining globally for the entire project, 97
Code point,
collation, 402
code-from-char,
as MapForce function (in core | string functions), 591
Collation,
locale collation, 402
sort component, 402
unicode code point, 402
Command reference, 1024
Comments,
add to target files, 143
Complex type,
sorting, 402
Component,
sort data, 402
Components,
Add Duplicate Input After, 1064
Add Duplicate Input Before, 1064
Add/Remove/Edit Database Objects, 1064
adding to the mapping, 38
Align Tree Left, 1064
Align Tree Right, 1064
aligning, 40
basics, 40
Change Root Element, 1064
changing settings, 40
comment, 34
Constant, 1059
Create Mapping to EDI X12 997, 1064
Create Mapping to EDI X12 999, 1064
Database, 1059
Database Table Actions, 1064
databases, 165, 230, 235, 243, 246, 253, 257, 258, 259
deleted items, 60
deleting, 62
EDI, 1059
Edit FlexText Configuration, 1064
Edit Schema Definition in XMLSpy, 1064
Excel 2007+ File, 1059
Exception, 1059
Filter: Nodes/Rows, 1059
icon reference, 34
IF-Else Condition, 1059
Insert Input, 1059
Insert Output, 1059
Join, 1059
JSON Schema/File, 1059
menu commands, 1064
overview, 34
Properties, 1064
Protocol Buffers File, 1059
Query Database, 1064
Refresh, 1064
Remove Duplicate, 1064
searching, 40
settings, 40
Simple Input, 1059

Components,
- Simple Output, 1059
- Sort: Nodes/Rows, 1059
- SQL/NoSQL-WHERE/ORDER, 1059
- structural, 34, 131, 132, 165, 230, 235, 243, 246, 253, 257, 258, 259
- Text File, 1059
- transformation, 34, 345
- Value-Map, 1059
- Variable, 1059
- Web Service Function, 1059
- Write Content as CDATA Section, 1064
- XBRL Document, 1059
- XML, 133
- XML and XML Schema, 133, 137, 139, 143, 145, 147
- XML Schema, 133
- XML Schema/File, 1059

concat,
- (as function) example of usage, 335
- as MapForce function (in core | string functions), 592

Configure,
- mff file, 495

Connection type,
- copy-all, 56
- matching-children, 54
- mixed, 52
- source-driven, 52
- standard, 51
- standard with mixed content, 52
- target-driven, 51
- target-driven vs. source-driven, 52
- target-driven with mixed content, 52

Connections,
- annotation, 58
- Auto Connect Matching Children, 1066
- change, 48
- Connect Matching Children, 1066
- context menu, 59
- copy, 48
- copy-all, 51, 56
- Copy-all (Copy Child Items), 1066
- create, 48
- delete, 48
- fix, 60
- fixing after editing schema, 60
- highlight selectively, 48
- keeping after deleting components, 62
- mandatory inputs, 48
- matching-children, 51, 54
- missing parent connections, 48
- mixed, 51
- move, 48
- moving, 60
- Properties, 1066
- see connection tooltips, 48
- settings, 58
- Settings for Connect Matching Children, 1066
- Source Driven (Mixed Content), 1066
- source-driven, 51
- standard, 51
- Target Driven (Standard), 1066
- target-driven, 51
- types, 51, 58

Connector,
- viewing the history of processed values, 780

Constants,
- add, 436

contains,
- as MapForce function (in core | string functions), 593

Context window,
- about, 775, 782

Conventions, 17

convert-to-utc,
- as MapForce function (in lang | datetime functions), 609

Copyright information, 1204

cos,
- as MapForce function (in lang | math functions), 637

count,
- as MapForce function (in core | aggregate functions), 516

count-substring,
- as MapForce function (in lang | string functions), 649

create-guid,
- as MapForce function (in lang | generator functions), 632

CSV,
- as mapping source, 323
- creating hierarchies - keys, 328
- creating multiple rows, 325

CSV files,
- adding or removing fields in,, 331
- as source component, 331
- as target component, 331
- previewing data from,, 331
- setting the encoding of,, 331

current,
- as MapForce function (in xslt | xslt functions library), 721

current-date,

current-date,
as MapForce function (in xpath2 | context functions), 667
current-dateTime,
as MapForce function (in xpath2 | context functions), 667
current-time,
as MapForce function (in xpath2 | context functions), 667
Custom libraries,
add, 494
C#, 494
C++, 494
Java, 494
reference, 494
Customize,
commands, 1074
context menus, 1074
Default Menu vs. MapForce Design, 1074
delete commands, 1074
Keyboard, 1075
menu shadows, 1074
menus, 1074
reset menu bars, 1074
shortcuts, 1075

D

Data overlays,
about, 775
Data streaming,
definition, 38
Database,
assign XML schema to field, 287
Encoding, 1083
generate multiple XML files from, 750
Result View, 1083
SQL editing settings, 1083
SQL Editor, 1083
SQL Generation, 1083
Text Fonts, 1083
writing XML files to, 290
Database Browser,
database tree layout, 279
filter database objects, 279
folder layout, 279
object locator, 279
search database objects, 279
sort tables into User and System tables, 279
Database connection,
reusing from Global Resources, 195
setting up, 168
setup examples, 196
starting the wizard, 170
Database drivers,
overview, 172
Database transactions,
enabling for stored procedures, 309
Databases, 279
add, 230
add database objects, 230, 243
ADO/OLEDB-specific settings, 235
as data target, 258, 259
as global resources, 826
bulk insert, 259
bulk transfer, 259
child tables, 259
column icons, 230
component settings, 235, 243
connection name, 235
Custom SQL statement, 243, 266
data source, 235
DB-generated value, 259
Delete All, 266
delete child records, 259, 266
Delete If, 259, 266
delete SELECT statement, 243
edit, 230
edit database objects, 230, 243
edit SELECT statement, 243
efficient usage, 258
error logging, 235, 259
exception, 259
exceptions, 235, 271
filter, 230
foreign-key relationship, 246
functions, 257
generate sequential values, 257
generate unique values, 257
Ignore If, 259, 266
INSERT, 277
Insert All, 259, 266
insert database objects, 230
Insert Rest, 266
issues with JDBC driver, 165
JDBC, 235
JDBC-specific settings, 235

Databases, 279

local relationships, 253

login settings, 235

mapped value, 259

mapping context, 258

mappings in different execution environments, 165

max() +1, 259

MERGE, 277

modify, 277

Null equal, 259

null fields, 257

null values, 257

objects, 230

parameters, 243

parent table, 259

query, 277

relations, 230

relationships, 246

remove, 230

rollback, 271

save all file paths relative to MFD file, 235

scenarios, 246

SELECT statement, 243

SELECT statement with parameters, 243

settings, 230

SQL auto-completion, 230

SQL statements in Output, 258

table actions, 258, 259, 266, 277

timeout for statement execution, 235

tips, 258

trace file, 235

trace level, 235

tracing, 235, 259

transaction handling, 235, 259

transaction rollback, 259, 271

transactions, 271

UPDATE, 277

Update If, 259

use shared database connection, 235

date-from-datetime,

as MapForce function (in lang | datetime functions), 610

datetime-add,

as MapForce function (in lang | datetime functions), 610

datetime-diff,

as MapForce function (in lang | datetime functions), 611

datetime-from-date-and-time,

as MapForce function (in lang | datetime functions), 612

datetime-from-parts,

as MapForce function (in lang | datetime functions), 613

day-from-datetime,

as MapForce function (in lang | datetime functions), 614

day-from-duration,

as MapForce function (in lang | datetime functions), 615

DB,

ORDER BY, 413

DB Queary Pane,

Database Browser, 277

Messages tab, 277

Results tab, 277

SQL Editor, 277

DB Query Pane,

Database Browser, 279

Messages tab, 285

Results tab, 284

SQL Editor, 281

Debugger position,

viewing the current value of, 780

Debugging,

about, 775

limitations, 771

minimal step, 1070

preparation for, 774

start, 1070

step into, 1070

step out, 1070

step over, 1070

step-by-step, 771

stop, 1070

with breakpoints, 771

Default values, 442

create, 445

default-collation,

as MapForce function (in xpath2 | context functions), 667

Defaults and Node Functions,

annotation, 454

apply conditionally, 447

block rules, 447

configure, 445

create, 445

defaults for unconnected nodes, 447

filter, 447

fractionDigits, 454

function icons, 442

input side, 442

length, 454

maxLength, 454

Defaults and Node Functions,
metadata, 454
minLength, 454
output side, 442
override inherited rules, 447
precision, 454
rule configuration, 445
scale, 454
scenarios, 447
totalDigits, 454
visual clues, 445
degrees,
as MapForce function (in lang | math functions), 637
Delete,
missing items, 60
Delimiter,
changing in CSV files, 331
changing in flat text files, 339
Derived types,
map to/from, 137
xsi:type, 137
distinct-values,
as MapForce function (in core | sequence functions), 561
Distribution,
of Altova's software products, 1204
divide,
as MapForce function (in core | math functions), 550
divide-integer,
as MapForce function (in lang | math functions), 637
document,
as MapForce function (in xslt | xslt functions library), 721
Document-level,
examples of integration of MapForce, 1004
DoTransform.bat,
execute with RaptorXML Server, 792
DTD,
source and target, 132
duration-add,
as MapForce function (in lang | datetime functions), 615
duration-from-parts,
as MapForce function (in lang | datetime functions), 616
duration-subtract,
as MapForce function (in lang | datetime functions), 617

E

Edit,
Cut/Copy/Paste/Delete, 1058
Find, 1058
Find Next, 1058
Find Previous, 1058
Redo, 1058
Select all, 1058
Undo, 1058
element-available,
as MapForce function (in xslt | xslt functions library), 722
empty,
as MapForce function (in lang | string functions), 649
Empty fields,
in CSV files, 331
in flat text files, 339
Encoding,
changing in CSV files, 331
changing in flat text files, 339
End User License Agreement, 1204, 1206
Enumerations,
in MapForceControl, 1054
equal,
as MapForce function (in core | logical functions), 543
equal-or-greater,
as MapForce function (in core | logical functions), 544
equal-or-less,
as MapForce function (in core | logical functions), 544
Errors,
out-of-memory, 38
troubleshooting, 38
Evaluation period,
of Altova's software products, 1204
Exceptions,
adding, 432
example, 433
throwing when node is missing, 433
exists,
as MapForce function (in core | sequence functions), 563
exp,
as MapForce function (in lang | math functions), 638
Extension functions for XSLT and XQuery,
Altova extensions, 1108
Java extension functions, 1184, 1193

Extension functions for XSLT and XQuery,
see under .NET extension functions, 1193
see under Java extension functions, 1184
Extension Functions in MSXSL scripts, 1199

F

false,
as MapForce function (in xpath2 | boolean functions), 665
Faulty connections,
after changing schema, 60
in databases, 60
in XML files, 60
Field,
keys in text files, 328
File,
as a button in a component, 40
as button on components, 746
Close, 1056
Close All, 1056
Compile to MapForce Server Execution File, 1056
Deploy to FlowForce Server, 1056
Exit, 1056
Generate Code, 1056
Generate Documentation, 1056
Mapping Settings, 1056
New, 1056
Open, 1056
Open Credentials Manager, 1056
Print, 1056
Print Preview, 1056
Print Setup, 1056
Recent files, 1056
Reload, 1056
Save, 1056
Save All, 1056
Save As, 1056
Validate Mapping, 1056
File DSN,
setting up, 190
File paths,
absolute, 43
broken, 43
fix broken references, 43
in execution environments, 46
in generated code, 46
of file-based databases, 43
relative, 43
relative versus absolute, 46
File/String,
as a button in a component, 40
as button on components, 746
File: (default),
as name of root node, 746
File: <dynamic>,
as name of root node, 746
Fill character,
in flat text files, 339
Filtering,
data from components, 408
database tables, 408
in databases, 417
Filters,
adding to the mapping, 408
find-substring,
as MapForce function (in lang | string functions), 650
Firebird,
Connecting through JDBC, 196
Connecting through ODBC, 197
first-items,
as MapForce function (in core | sequence functions), 565
Fixed,
lenght files - mapping, 323
Flat file,
mapping, 323
floor,
as MapForce function (in core | math functions), 550
FlowForce Server,
deploying Global Resources to, 802, 833
deploying mappings to, 802
Global Resources in, 833
Folders,
as global resources, 824
format-date,
as MapForce function (in core | conversion functions), 522
format-dateTime,
as MapForce function (in core | conversion functions), 523
format-guid-string,
as MapForce function (in lang | string functions), 651
format-number,
as MapForce function (in core | conversion functions), 526
format-time,
as MapForce function (in core | conversion functions), 529
function-available,

function-available,
as MapForce function (in xslt | xslt functions library), 722
Functions, 436
add, 436
add parameters, 436
argument data type, 436
basics, 436
constants, 436
Create User-Defined Function, 1067
Create User-Defined Function from Selection, 1067
delete parameters, 436
description, 436
find in the Libraries window, 436
find occurences in active mapping, 436
Function Settings, 1067
Insert Input, 1067
Insert Output, 1067
lang | string functions, 644, 659
node, 442, 445
parameters, 436
Remove Function, 1067
search, 436
Fundamentals, 32, 34, 48, 64
context, 79
mapping rules, 77
mapping strategies, 77
multiple target components, 91
priority context, 86
processing order, 79
sequences, 78

G

General Procedures, 64
code generation, 75
generate code, 66
mapping settings, 75
output file settings, 75
paths in generated code, 75
text view, 68
text view search, 72
vaidation, 64
validate mapping, 64
validate output, 64
valide, 64
Generated code,
throwing exceptions from, 432
generate-id,
as MapForce function (in xslt | xslt functions library), 723
generate-sequence,
as MapForce function (in core | sequence functions), 566
get-fileext,
as MapForce function (in core | file path functions), 537
get-folder,
as MapForce function (in core | file path functions), 537
Global objects,
in SPL, 978
Global Resources,
creating, 816
databases as, 826
Definitions file, 816
deploying to FlowForce Server, 802, 833
folders as, 824
in execution environments, 832
in FlowForce Server, 833
introduction to, 815
setup, 816
XML Files as, 822
greater,
as MapForce function (in core | logical functions), 545
group-adjacent,
as MapForce function (in core | sequence functions), 567
group-by,
as MapForce function (in core | sequence functions), 569
group-ending-with,
as MapForce function (in core | sequence functions), 573
group-into-blocks,
as MapForce function (in core | sequence functions), 575
group-starting-with,
as MapForce function (in core | sequence functions), 576
GUI, 22
bars, 23
DB Query pane, 277, 281, 284, 285
Messages window, 26
panes, 28
windows, 23

H

Help,
About MapForce, 1091
Check for Updates, 1091

Help,
- Download Components and Free Tools, 1091
- FAQ on the Web, 1091
- Index, 1091
- MapForce on the Internet, 1091
- MapForce Training, 1091
- Order Form, 1091
- Registration, 1091
- Search, 1091
- Software Activation, 1091
- Support Center, 1091
- Table of Contents, 1091

Hierarchy,
- from text files, 328

hour-from-datetime,
- as MapForce function (in lang | datetime functions), 618

hour-from-duration,
- as MapForce function (in lang | datetime functions), 618

HTML,
- generate mapping documentation, 761
- preview mapping output as, 807

I

IBM DB2,
- connecting through JDBC, 199
- connecting through ODBC, 201
- reading from XML type fields, 287
- writing to XML type fields, 287

IBM DB2 for i,
- connecting through JDBC, 204
- connecting through ODBC, 205

IBM Informix,
- connecting through JDBC, 207

If-Else conditions,
- adding to the mapping, 408

implicit-timezone,
- as MapForce function (in xpath2 | context functions), 668

INNER JOIN,
- in Join components, 386

Input,
- duplicate, 40

Insert,
- SQL WHERE component, 413

Integrating,
- MapForce in applications, 993

Integration,
- with Altova products, 20

Internet usage,
- in Altova products, 1203

is-not-null,
- as MapForce function (in db functions), 606

is-null,
- as MapForce function (in db functions), 606

is-xsi-nil,
- as MapForce function (in core | node functions), 555

Item,
- missing, 60

item-at,
- as MapForce function (in core | sequence functions), 578

items-from-till,
- as MapForce function (in core | sequence functions), 579

J

Java, 1011
- reference to generated classes, 959
- VM library location, 1082

Java extension functions,
- constructors, 1189
- datatype conversions, Java to Xpath/XQuery, 1192
- datatype conversions, XPath/XQuery to Java, 1191
- for XSLT and XQuery, 1184
- instance methods, instance fields, 1191
- static methods, static fields, 1190
- user-defined class files, 1186
- user-defined JAR files, 1188

JDBC,
- as data connection interface, 168
- connect to Teradata, 227
- setting up a connection (Windows), 187

K

Key,
- fields in text files, 328
- sort key, 402

Key-value pairs,
- using on the mapping, 421

L

lang | string functions,
sleep, 659
last,
as MapForce function (in xpath2 | context functions), 668
last-items,
as MapForce function (in core | sequence functions), 581
leapyear,
as MapForce function (in lang | datetime functions), 619
left,
as MapForce function (in lang | string functions), 651
LEFT OUTER JOIN,
in Join components, 386
left-trim,
as MapForce function (in lang | string functions), 652
Legal information, 1204
less,
as MapForce function (in core | logical functions), 545
Library, 987
automatic loading of, 495
License, 1206
information about, 1204
License metering,
in Altova products, 1205
Licensing, 1091
package license files with installer, 1206
Local relations,
and stored procedures, 312, 313, 317
Locale collation, 402
local-name-from-QName,
as MapForce function (in lang | QName functions), 560
log,
as MapForce function (in lang | math functions), 638
log10,
as MapForce function (in lang | math functions), 639
logical-and,
as MapForce function (in core | logical functions), 546
logical-not,
as MapForce function (in core | logical functions), 546
logical-or,
as MapForce function (in core | logical functions), 547
logical-xor,
as MapForce function (in lang | logical functions), 633
Look-up tables,
using on the mapping, 421
lowercase,
as MapForce function (in lang | string functions), 653

M

main-mfd-filepath,
as MapForce function (in core | file path functions), 538
MapForce,
integration, 993
introduction, 17
model of data transformation, 17
overview, 17
MapForce COM API, 992
MapForce plug-in for Eclipse,
about, 847, 850, 857
accessing common menus and functions, 853
configuring for automatic code generation, 864
creating a MapForce/Eclipse project, 857
creating new mappings, 859
extending functionality, 867
extension point, 867
importing mappings into an Eclipse project, 861
installing, 848
switching to the MapForce perspective, 850
working with mappings and projects, 857
MapForce plug-in for Visual Studio,
about, 844
enabling, 844
information about menus and functions, 844
working with mappings and projects, 844
MapForce Server,
automating mappings, 793
compiling mappings for, 799
Global Resources in, 833
throwing exceptions from, 432
MapForceCommand,
in MapForceControl, 1034
MapForceCommands,
in MapForceControl, 1036
MapForceControl, 1037
documentation of, 993
examples of integration at document level, 1004
integration using C#, 1004
integration using Visual Basic, 1020
object reference, 1034

MapForceControlDocument, 1045

MapForceControlPlaceHolder, 1051

Mapping,

add a component, 103

add a function, 106

basics, 32

component types, 32

components, 32

connections, 32

connectors, 32

create, 102

creating, 32

debugging, 771

dynamic file names, 128

flat file format, 323

flat schema structure, 246

fundamentals, 32

generate code, 108

output file settings, 75

parts, 32

save, 102

save output, 108

scenarios, 18

select a transformation language, 102

settings, 75

source, 18

source-driven - mixed content, 52

SQL/XML Standard, 246

target, 18

terminology, 32

terms, 32

validate, 102

view structure, 103

XML schema version, 75

Mapping documentation,

generating, 761

Mapping input,

Supplying multiple files as, 746

Mapping output,

Generating multiple files as, 746

Mappings,

automated processing, 793

MariaDB,

connect through ODBC, 208

match-pattern,

as MapForce function (in lang | string functions), 653

max,

as MapForce function (in core | aggregate functions), 516

as MapForce function (in lang | math functions), 639

max-string,

as MapForce function (in core | aggregate functions), 517

Memory requirements, 1202

Menu commands, 1055

Component, 1064

Connection, 1066

Cutomize, 1074, 1075

Debug, 1070

Edit, 1058

File, 1056

Function, 1067

Help, 1091

Insert, 1059

Output, 1068

Project, 1062

Tools, 1073

Tools | Customize, 1074

Tools | Keyboard, 1075

Tools | Options, 1077, 1080

Tools | Options | Database, 1083

Tools | Options | Java, 1082

Tools | Options | Network Proxy, 1088

View, 1071

Windows, 1090

Menu reference, 1055

Messages tab,

clear, 285

copy message to clipboard, 285

copy messages, 285

errors, 285

filter, 285

find, 285

find next, 285

find previous, 285

summary, 285

warnings, 285

mfd-filepath,

as MapForce function (in core | file path functions), 538

mff,

library file, 494

mff.xsd file, 494

mff file,

configuring, 495

Microsoft Access,

connecting through ADO, 176, 209

Microsoft Azure SQL, 211

Microsoft SharePoint Server,

Microsoft SharePoint Server,
adding files as components from, 38
Microsoft SQL Server,
connecting through ADO, 211
connecting through ODBC, 212
millisecond-from-datetime,
as MapForce function (in lang | datetime functions), 619
millisecond-from-duration,
as MapForce function (in lang | datetime functions), 620
min,
as MapForce function (in core | aggregate functions), 518
as MapForce function (in lang | math functions), 640
min-string,
as MapForce function (in core | aggregate functions), 519
minute-from-datetime,
as MapForce function (in lang | datetime functions), 620
minute-from-duration,
as MapForce function (in lang | datetime functions), 621
Missing items, 60
Mixed,
content mapping, 52
source-driven mapping, 52
Mixed content,
mapping, 52
with standard connections, 52
with target-driven connections, 52
modulus,
as MapForce function (in core | math functions), 551
month-from-datetime,
as MapForce function (in lang | datetime functions), 621
month-from-duration,
as MapForce function (in lang | datetime functions), 622
msxsl:script, 1199
multiply,
as MapForce function (in core | math functions), 552
MySQL,
connecting through ODBC, 217

N

Namespace URI,
DTD, 132
Namespaces,
custom, 147
declare manually, 147
namespace-uri-form-QName,
as MapForce function (in lang | QName functions), 560
Native connections,
Azure CosmosDB, 194
CouchDB, 194
MariaDB, 194
MongoDB, 194
PostgreSQL, 194
SQLite, 194
negative,
as MapForce function (in lang | logical functions), 633
Network proxy,
automatic, 1088
configuration, 1088
manual, 1088
settings, 1088
system, 1088
New Features, 13
Version 2021, 16
Version 2022, 15
Version 2023, 14
Version 2024, 13
Node functions,
create, 445
Node names,
mapping data from/to, 726
node-name,
as MapForce function (in core | node functions), 557
as MapForce function (in xpath2 | accessors library), 663
node-name function,
alternatives to using, 726
normalize-space,
as MapForce function (in core | string functions), 594
not-equal,
as MapForce function (in core | logical functions), 548
not-exists,
as MapForce function (in core | sequence functions), 582
now,
as MapForce function (in lang | datetime functions), 622
NULL,
attribute, 139
values, 139
values in databases, 139
number,
as MapForce function (in core | conversion functions), 530
numeric,
as MapForce function (in lang | logical functions), 634

O

ODBC,

as data connection interface, 168

connect to MariaDB, 208

connect to Teradata, 228

setting up a connection, 190

ODBC Drivers,

checking availability of, 190

OLE DB,

as data connection interface, 168

OpenAI,

example, 659

use-case scenario, 659

OpenJDK,

as Java Virtual Machine, 187

Oracle,

reading from XML type fields, 287

writing to XML type fields, 287

Oracle database,

connecting through JDBC, 219

connecting through ODBC, 220

Oracle package,

stored procedures and functions, 300

ORDER BY,

SQL where component, 413

Ordering data,

sort component, 402

OS,

for Altova products, 1202

Output,

Built-in Execution Engine, 1068

C#, 1068

C++, 1068

Insert/Remove Bookmark, 1068

Java, 1068

Next Bookmark, 1068

Pretty-Print XML Text, 1068

Previous Bookmark, 1068

Regenerate Output, 1068

Remove All Bookmarks, 1068

Run SQL/NoSQL-Script, 1068

Save All Output Files, 1068

Save Output File, 1068

Text View Settings, 1068

Validate Output File, 1068

XQuery, 1068

XSLT 1.0, 1068

XSLT 2.0, 1068

XSLT 3.0, 1068

P

pad-string-left,

as MapForce function (in lang | string functions), 654

pad-string-right,

as MapForce function (in lang | string functions), 655

Panes,

DB Query, 28

Mapping, 28

Output, 28

StyleVision output, 28

XQuery, 28

XSLT, 28

Parameters,

supplying to the mapping, 346, 350

Parent context,

example, 82

parse-date,

as MapForce function (in core | conversion functions), 531

parse-dateTime,

as MapForce function (in core | conversion functions), 532

parse-number,

as MapForce function (in core | conversion functions), 534

Parser,

built into Altova products, 1202

parse-time,

as MapForce function (in core | conversion functions), 536

PDF,

generate mapping documentation, 761

preview mapping output as, 807

pi,

as MapForce function (in lang | math functions), 640

Platforms,

for Altova products, 1202

position,

as MapForce function (in core | sequence functions), 582

positive,

as MapForce function (in lang | logical functions), 634

PostgreSQL,

connecting through ODBC, 222

pow,
as MapForce function (in lang | math functions), 641
Priority context,
example, 88
Processing,
automating mappings, 793
Processing Instructions,
add to target files, 143
Processing Instructions and Comments,
mapping, 52
Progress OpenEdge database,
connecting through JDBC, 223
connecting through ODBC, 224
Project,
add mappings to, 95
add Web links, 95
basics, 95
closing, 95
code generation, 95
code generation settings, 97, 98
creating, 95
deleting, 95
folders, 98
generate code, 95
image preview, 95
new, 95
opening, 95
organization, 95
properties, 98
removing, 95
searching, 95
settings, 97
watch video tutorials, 95
Project files (.mfp), 95
Projects, 95
Add Active File to Project, 1062
Add Files to Project, 1062
Add Mapping File for Operation, 1062
Close, 1062
Create Folder, 1062
Create Mapping for Operation, 1062
Generate Code for Entire Project, 1062
Generate Code in, 1062
Insert Web Service, 1062
Open File in XMLSpy, 1062
Open Mapping, 1062
Properties, 1062
Reload, 1062
Save, 1062

Q

QName,
as MapForce function (in lang | QName functions), 559
QName-as-string,
as MapForce function (in lang | QName functions), 644
Question mark,
missing items, 60
Quote character,
in CSV files, 331

R

radians,
as MapForce function (in lang | math functions), 641
random,
as MapForce function (in lang | math functions), 641
RaptorXML Server,
executing a transformation, 792
read-binary-file,
as MapForce function (in lang | file functions), 628
Regular expressions,
using in mappings, 508
remove-fileext,
as MapForce function (in core | file path functions), 538
remove-folder,
as MapForce function (in core | file path functions), 539
remove-timezone,
as MapForce function (in lang | datetime functions), 623
repeat-string,
as MapForce function (in lang | string functions), 655
replace,
as MapForce function (in lang | string functions), 656
replace-fileext,
as MapForce function (in core | file path functions), 539
replicate-item,
as MapForce function (in core | sequence functions), 584
replicate-sequence,
as MapForce function (in core | sequence functions), 586
resolve-filepath,
as MapForce function (in core | file path functions), 540
resolve-uri,

resolve-uri,
as MapForce function (in xpath2 | anyURI functions), 664
Results tab,
copy, 284
find, 284
select, 284
sort, 284
reversefind-substring,
as MapForce function (in lang | string functions), 656
right,
as MapForce function (in lang | string functions), 657
right-trim,
as MapForce function (in lang | string functions), 658
round,
as MapForce function (in core | math functions), 552
round-half-to-even,
as MapForce function (in xpath2 | numeric functions), 691
round-precision,
as MapForce function (in core | math functions), 553
Rows,
mapping from - text files, 328
RTF,
generate mapping documentation, 761
preview mapping output as, 807

S

Schema,
generate, 132
industry standard, 132
pre-packaged, 132
Schema Manager,
CLI Help command, 158
CLI Info command, 159
CLI Initialize command, 159
CLI Install command, 160
CLI List command, 160
CLI overview, 158
CLI Reset command, 161
CLI Uninstall command, 162
CLI Update command, 163
CLI Upgrade command, 163
how to run, 152
installing a schema, 156
listing schemas by status in, 154
overview of, 149
patching a schema, 156
resetting, 157
status of schemas in, 154
uninstalling a achema, 157
upgrading a schema, 156
schemanativetype, 975
Search,
files in the Project window, 95
items within mapping components, 40
projects, 95
second-from-datetime,
as MapForce function (in lang | datetime functions), 623
second-from-duration,
as MapForce function (in lang | datetime functions), 624
set-empty,
as MapForce function (in core | sequence functions), 587
set-null,
as MapForce function (in db functions), 607
set-xsi-nil,
as MapForce function (in core | node functions), 557
Silent Installation,
modify MSI file, 1206
package license files with installer, 1206
Simple type,
sorting, 402
sin,
as MapForce function (in lang | math functions), 642
skip-first-items,
as MapForce function (in core | sequence functions), 588
Smart component deletion, 62
Software product license, 1206
Sort,
sort component, 402
Sort key,
sort component, 402
Sort order,
changing, 402
Sorting,
in databases, 417
Source-driven,
mixed-content mapping, 52
SPL, 973
code blocks, 974
conditions, 982
foreach, 983
global objects, 978
subroutines, 984
using files, 979

SPL, 973
variables, 977
SQL,
joining data, 386
SQL Azure, 211
SQL Editor,
add bookmarks, 281
add SQL comments, 281
block comment, 281
create statements, 281
execute SQL statements, 281
export SQL scripts, 281
import SQL scripts, 281
insert SQL regions, 281
line comment, 281
load from scripts, 281
toolbar buttons, 281
SQL Server,
connecting through ADO, 176
connecting through ADO.NET, 181
connecting via JDBC, 187
reading from XML type fields, 287
writing to XML type fields, 287
SQL WHERE,
component - insert, 413
ORDER BY, 413
SQL WHERE/ORDER,
as MapForce component, 417
SQLite,
changing database path to absolute in generated code, 46
mapping data to, 755
writing XML files to, 290
sqrt,
as MapForce function (in lang | math functions), 642
starts-with,
as MapForce function (in core | string functions), 595
static-node-annotation,
as MapForce function (in core | node functions), 558
static-node-name,
as MapForce function (in core | node functions), 558
Stored procedures,
adding to a mapping, 300
calling from a mapping, 297, 302, 305, 309, 313, 317
support notes, 297
String,
as MapForce function (in core | conversion functions), 536
as MapForce function (in xpath2 | accessors library), 663
parsing data from, 753
serializing data to, 753, 755
string-as-QName,
as MapForce function (in lang | QName functions), 644
string-compare,
as MapForce function (in lang | string functions), 660
string-compare-ignore-case,
as MapForce function (in lang | string functions), 661
string-join,
as MapForce function (in core | aggregate functions), 519
string-length,
as MapForce function (in core | string functions), 595
Structural components,
databases, 165, 230, 235, 243, 246, 253, 257, 258, 259
XML, 132
XML and XML Schema, 132
XML Schema, 132
Stylevision,
creating stylesheets with, 769
generating mapping documentation with, 764
substitute-missing,
as MapForce function (in core | sequence functions), 589
substitute-missing-with-xsi-nil,
as MapForce function (in core | node functions), 559
substitute-null,
as MapForce function (in db functions), 607
substring,
as MapForce function (in core | string functions), 596
substring-after,
as MapForce function (in core | string functions), 596
substring-before,
as MapForce function (in core | string functions), 597
subtract,
as MapForce function (in core | math functions), 553
sum,
as MapForce function (in core | aggregate functions), 521
Sybase,
connecting through JDBC, 226
System DSN,
setting up, 190
system-property,
as MapForce function (in xslt | xslt functions library), 723

T

Table data,
sorting, 402

tan,
as MapForce function (in lang | math functions), 643
Technical Information, 1202
Teradata,
connect through JDBC, 227
connect through ODBC, 228
Text,
files - defining key fields, 328
mapping text files, 323
Text files,
adding or removing fields in,, 339
as source component, 339
as target component, 339
mapping data from, 335
previewing data from,, 339
setting the encoding of,, 339
setting the fill character, 335
setting the fixed field size, 335
Text View,
bookmarks, 68
end-of-line markers, 68
folding margin, 68
indentation guides, 68
line numbers, 68
pretty-printing, 68
search, 72
source folding, 68
syntax coloring, 68
text highlighting, 68
whitespace markers, 68
word wrapping, 68
zooming, 68
Theme selection for MapForce in Eclipse, 853
time-from-datetime,
as MapForce function (in lang | datetime functions), 624
timezone,
as MapForce function (in lang | datetime functions), 625
tokenize,
as MapForce function (in core | string functions), 598
tokenize-by-length,
as MapForce function (in core | string functions), 600
tokenize-regexp,
as MapForce function (in core | string functions), 603
Tools,
Active Configuration, 1073
Create Reversed Mapping, 1073
Customize, 1073
Global Resources, 1073
menu command, 1073
Options, 1073
Restore Toolbars and Windows, 1073
XBRL Taxonomy Manager, 1073
Tools | Options,
code generation, 1080
Database, 1077
Debugger, 1077
Editing, 1077
General, 1077
Generation, 1077, 1080
generation settings, 1080
Java, 1077
Messages, 1077
Network Proxy, 1077
XBRL, 1077
Transformation languages,
BUILT-IN, 19
C#, 19
C++, 19
Java, 19
XQuery, 19
XSLT 1.0, 19
XSLT 2.0, 19
XSLT 3.0, 19
Transformations,
RaptorXML Server, 792
translate (in core | string functions),
as MapForce function, 604
true,
as MapForce function (in xpath2 | boolean functions), 665
Tutorials,
basic, 101
chained mapping, 115
duplicate input, 110
dynamic file names, 123
example files, 100
multiple sources to multiple targets, 123
multiple sources to one target, 110
one source to one target, 101
pass-through component, 115
transformation, 101
Types,
built in, 987

U

UDFs,
- and mapping context, 469

unary-minus,
- as MapForce function (in lang | math functions), 643

Unicode,
- code point collation, 402

Unicode support,
- in Altova products, 1203

unparsed-entity-uri,
- as MapForce function (in xslt | xslt functions library), 724

uppercase,
- as MapForce function (in lang | string functions), 662

URI,
- in DTDs, 132

URL,
- adding files as components from, 38

User DSN,
- setting up, 190

User interface, 22

User-defined functions,
- add parameters, 463
- advantages, 457
- call, 458
- call recursively, 467
- complex-type structures, 463
- copy-paste, 458
- create, 458
- delete, 458
- edit, 458
- example, 457
- examples, 467, 472
- import, 458
- inline, 458
- input parameters, 458
- lookup, 472
- navigate, 458
- of complex type, 463
- of simple type, 463
- output parameters, 458
- overview, 457
- parameter order, 463
- parameters, 463
- recursive, 467
- recursive search, 467
- regular, 458

V

Validator,
- in Altova products, 1202

Value-Map,
- as mapping component, 421
- examples, 425, 428

Values window,
- about, 775, 780
- Context tab, 780
- History tab, 780
- Related tab, 780

Variables,
- adding to the mapping, 362
- changing the scope of, 366
- complex, 360
- DB-based, 360
- examples of use, 368, 369, 371
- in SPL, 977
- simple, 360

View,
- Back, 1071
- Debug Windows, 1071
- Forward, 1071
- Libraries, 1071
- Manage Libraries, 1071
- menu command, 1071
- Messages, 1071
- Overview, 1071
- Project Window, 1071
- Show Annotations, 1071
- Show Connections from Source to Target, 1071
- Show Library in Function Header, 1071
- Show Selected Components Connectors, 1071
- Show Tips, 1071
- Show Types, 1071
- Status Bar, 1071
- XBRL Display Options, 1071
- Zoom, 1071

Visual Basic,
- integration of MapForce, 1020

Visual Studio,
- adding the MapForce ActiveX Controls to the toolbox, 996

Visual Studio plug-in,
running MapForce as, 844

W

WebDAV Server,
adding files as components from, 38
weekday,
as MapForce function (in lang | datetime functions), 625
weeknumber,
as MapForce function (in lang | datetime functions), 626
WHERE,
SQL WHERE component, 413
Wildcards,
import schema, 145
selections, 145
wrapper schema, 145
xs:any/xs:anyAtrribute, 145
Windows,
Cascade, 1090
Classic theme, 1090
Dark theme, 1090
Libraries, 23
Light theme, 1090
Manage Libraries, 23
Mapping, 23
Messages, 26
Multiple Mapping Windows, 23
Overview, 23
Project, 23
support for Altova products, 1202
Theme, 1090
Tile Horisontal/Vertical, 1090
Windows dialog, 1090
Word,
generate mapping documentation, 761
Word 2007+,
preview mapping output as, 807
write-binary-file,
as MapForce function (in lang | file functions), 630

X

XML,
add DTD reference, 133
add schema, 133
as mapping target, 323
BOM, 133
byte order, 133
cast values to target types, 133
component name, 133
component settings, 133
declaration, 133
digital signature, 133
encoding settings, 133
input XML file, 133
mapping data from CSV to, 323
min/maxOccurs, 133
output XML file, 133
pretty print, 133
save file paths relative to MFD file, 133
Schema file, 133
standalone, 133
standalone="yes", 133
StyleVision Power Stylesheet File, 133
writing to database field, 290
XML declaration, 133
XML data,
reading from database fields, 287
writing to database fields, 287
XML Files,
as global resources, 822
generate from database records, 750
generate from single XML source, 748
XML Parser,
about, 1202
XML Schema Manager, 132
XQuery,
adding custom functions, 483, 484
importing modules, 484
xs: any, 145
xs:anyAttribute, 145
xsi:nil,
as attribute in XML instance, 139
XSLT,
adding custom functions, 477
removing custom functions, 477
template namespace, 477

Y

year-from-datetime,
as MapForce function (in lang | datetime functions), 626

year-from-duration,
as MapForce function (in lang | datetime functions), 627

Z

Z to A,
sort component, 402