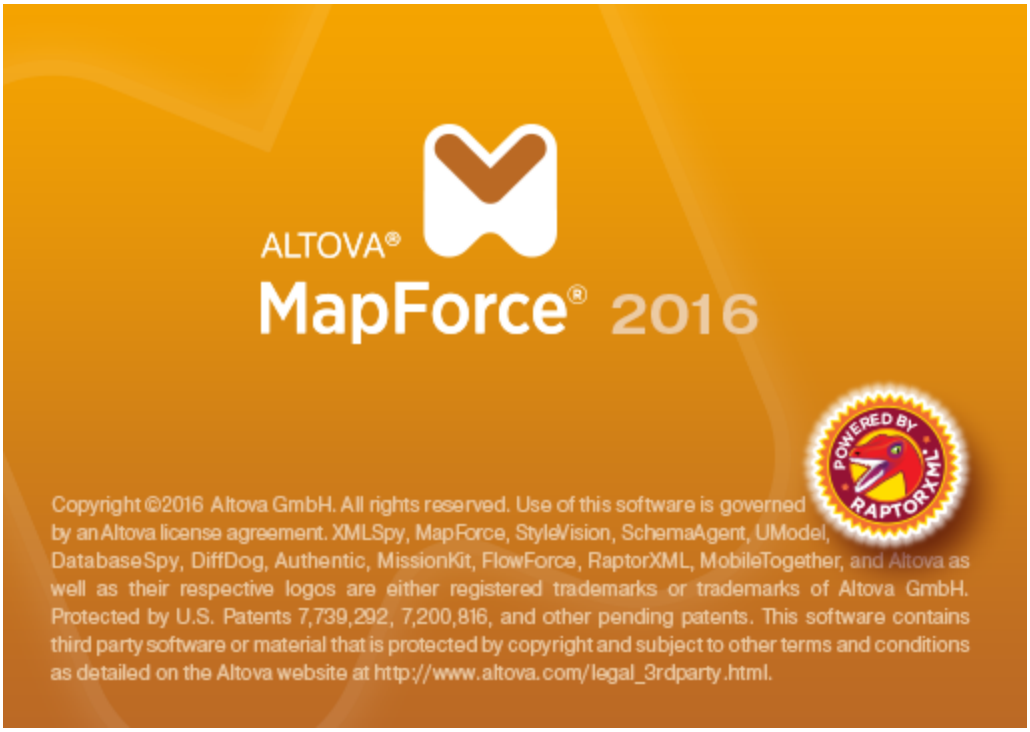
User and Reference Manual



Altova MapForce 2016 User & Reference Manual

Published: 2016

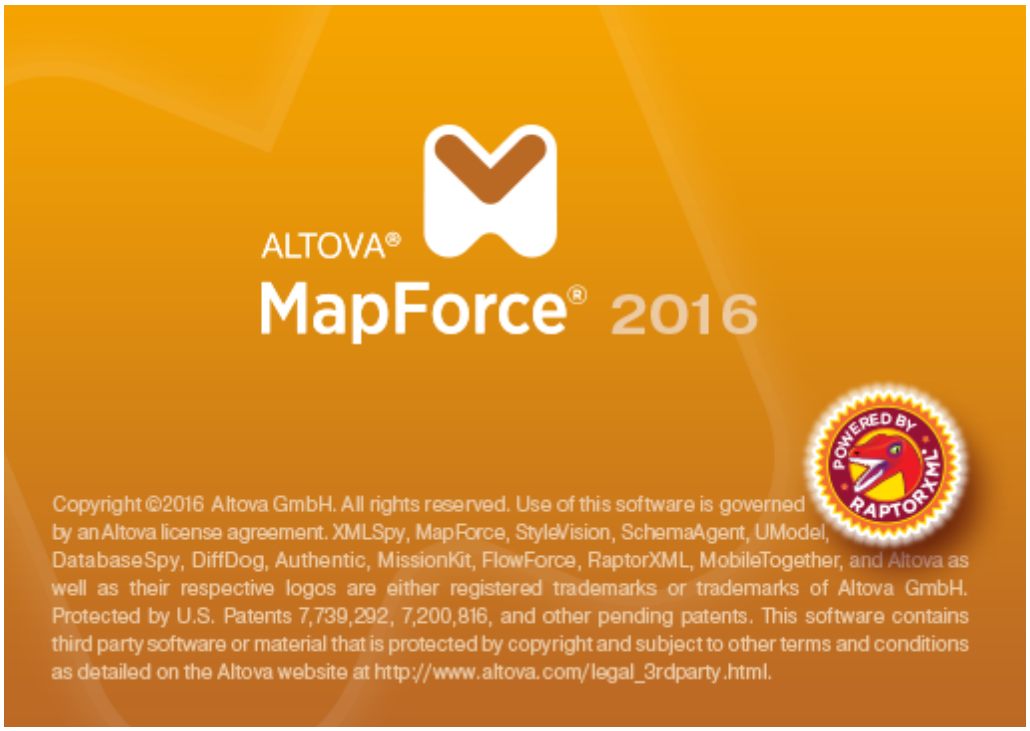
Table of Contents

Chapter 1

MapForce 2016

1 MapForce 2016

MapForce® 2016 Enterprise Edition is a visual data mapping tool for advanced data integration projects. MapForce® is a 32/64-bit Windows application that runs on Windows 10, Windows 8, Windows 7, Windows Vista, Windows XP, and Windows Server 2003/2008/2012. 64-bit support is available for the Enterprise and Professional editions.



Last updated: 03 February 2016

1.1 What's new...

New in MapForce 2016 R2:

- More intuitive code folding in the XSLT pane: collapsed text is displayed with an ellipsis symbol and can be previewed as a tooltip. The same rules apply for text in the XQuery pane and the SQL Editor
- You can search for all occurrences of a function within the active mapping (in the Libraries window, right-click the function, and select **Find All Calls**).
- You can call REST-style Web services from a mapping (see Adding a Web Service Call (REST-style). This adds to existing support for WSDL-style Web services.
- Support for the EDIFACT 2015A directory (see EDI)
- Support for Eclipse 4.5 (see MapForce Plug-in for Eclipse)
- Internal updates and optimizations

New features in MapForce 2016:

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

New features in MapForce Version 2015 R4:

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

New features in MapForce Version 2015 R3 include:

- Option to suppress the `<?xml ... ?>` declaration in XML output
- When calling Web services, you can use preemptive HTTP authentication, HTTPS, and basic WS-Security (see Setting HTTP Security and Setting WS-Security)
- Support for reading Strict Open XML Excel 2013 files
- FlexText split and switch using regular expressions (see FlexText and Regular Expressions)
- Text-based components (including EDI, CSV, fixed-length field, JSON, and XML) can parse and serialize strings in addition to plain files

- SQLite database support (see Setting up a SQLite connection)
- New string padding functions: pad-string-left and pad-string-right
- New component type: Simple Output
- Internal updates and optimizations

New features in MapForce Version 2015 include:

- New `language` argument available in the format-date and format-dateTime functions
- New sequence function: replicate-item
- XBRL table linkbase support
- New mode for FlexText "Split Once" option: Delimited (line starts with)
- Map to or from JSON files

New features in MapForce Version 2014 R2 include:

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

New features in MapForce Version 2014 include:

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

New features in MapForce Version 2013 R2 SP1 include:

- New super-fast transformation engine RaptorXML Server

New features in MapForce Version 2013 R2 include:

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

New features in MapForce Version 2013 include:

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

New features in MapForce Version 2012 R2 include:

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

New features in MapForce Version 2012 include:

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

New features in MapForce Version 2011R3 include:

- Intermediate variables
- Support for multiple row ranges in Excel components
- Support for EDI X12 - HIPAA Implementation Guides
- Generation of X12 EDI 999 Implementation Acknowledgement component
- XML Digital signatures
- Support for US-GAAP 2011
- Support for .NET Framework 4.0 assembly files
- Ability to output StyleVision formatted documents from the command line

New features in MapForce Version 2011R2 include:

- Built-in Execution Engine now supports streaming output
- Find function capability in Library window
- Reverse mapping
- Extendable IF-ELSE function
- Node Name and parsing functions in Core Library
- New EDI format IATA PADIS

- Ability to process multiple EDI messages per component
- Improved database table actions dialog with integrated key generation settings
- New option of using StyleVision Power Stylesheets when documenting a mapping

New features in MapForce Version 2011 include:

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

New features in MapForce Version 2010 Release 3 include:

- Support for generation of Visual Studio 2010 project files for C# and C++ added
- Ability to define a worksheet row as column names in an Excel component
- Support for MSXML 6.0 in generated C++ code
- Support for Nillable values, and xsi:nil attribute in XML instance files
- Ability to disable automatic casting to target types in XML documents
- Support for SAP IDocs

New features in MapForce Version 2010 Release 2 include:

- 64-bit MapForce Enterprise / Professional editions on 64-bit operating systems: Windows Server 2003/2008, Windows XP, Windows Vista and Windows 7
- Support for Excel 2010
- Automatic connection of identical child connections when moving a parent connection
- Support for fields in the SQL Where component
- Ability to add compiled Java .class and .NET assembly files
- Ability to tokenize input strings for further processing
- UN/EDIFACT and ANSI X12 EDI source file validation
- Generation of X12 EDI 997 Functional Acknowledgement component

New features in MapForce Version 2010 include:

- Multiple input/output files per component
- Upgraded relative path support
- xsi:type support allowing use of derived types
- New internal data type system
- Improved user-defined function navigation
- Validation of EDI output in generated code
- Support of EDIFACT service messages CONTRL, AUTACK and KEYMAN
- Support for Web services defined using WSDL 2.0
- Enhanced handling of mixed content in XML elements

New features in MapForce Version 2009 SP1 include:

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

New features in MapForce Version 2009 include:

- Support for XBRL and XBRL dimension instance files, as well as XBRL taxonomies as source and target components
- EDI HL7 versions 2.2 to 2.6 components as source and target components
- EDI HL7 versions 3.x XML as source and target components
- Documentation of mapping projects
- Native support for XML fields in SQL Server
- Grouping of nodes or node content
- Ability to filter data based on a nodes position in a sequence
- QName support
- Item/node search in components

New features in MapForce Version 2008 Release 2 include:

- Ability to automatically generate XML Schemas for XML files
- Office Open XML Excel 2007 and higher (*.xslx) support as source and target components
- Support for stream objects as input/output in generated Java and C# code
- Generation of Visual Studio 2008 project files for C++ and C#
- Support for SOAP version 1.2 in Web services
- New Repeated split option "Starts with..." in FlexText
- Ability to strip database schema names from generated code
- SQL SELECT Statements as virtual tables in database components
- Local Relations - on-the-fly creation of primary/foreign key relationships
- Support for Altova Global Resources
- Performance optimizations

New features in MapForce Version 2008 include:

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

New features in MapForce Version 2007 Release 3 include:

- XML data mapping to/from database fields (see Mapping XML Data to / from Database Fields)
- Direct querying of databases
- SQL-WHERE filter and SQL statement wizard
- Code generator optimization and improved documentation
- Full support for all EDI X12 releases from 3040 to 5030

- Full support for UN/EDIFACT messages of directories 93A to 06B

Chapter 2

Introduction

2 Introduction

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

2.1 What Is MapForce?

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

You can use MapForce to conveniently convert and transform data from and to a variety of file-based and other formats. Regardless of the technology you work with, MapForce determines automatically the structure of your data, or gives you the option to supply a schema for your data, or generate it automatically from a sample instance file. For example, if you have an XML instance file but no schema definition, MapForce can generate it for you, thus making the data inside the XML file available for mapping to other files or formats.

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

MapForce Basic Edition	MapForce Professional Edition	MapForce Enterprise Edition
• XML and XML schema • HL7 version 3.x (schema-based)	• XML and XML schema • Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format • Databases (all major relational databases, including Microsoft Access and SQLite databases)	• XML and XML schema • Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format • Data from legacy text files can be mapped and converted to other formats with MapForce FlexText • Databases (all major relational databases, including Microsoft Access and SQLite databases) • EDI family of formats (including UN/EDIFACT, ANSI X12, HL7, IATA PADIS, SAP IDoc, TRADACOMS) • JSON files • Microsoft Excel 2007 and later files • XBRL instance files and taxonomies

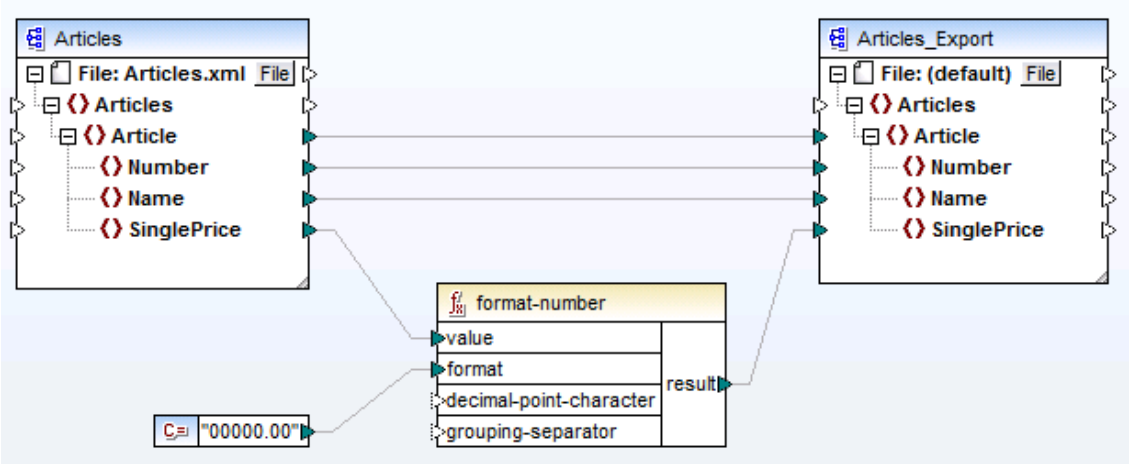
Based on the MapForce edition, you can choose the preferred language for your data transformation as follows.

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

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

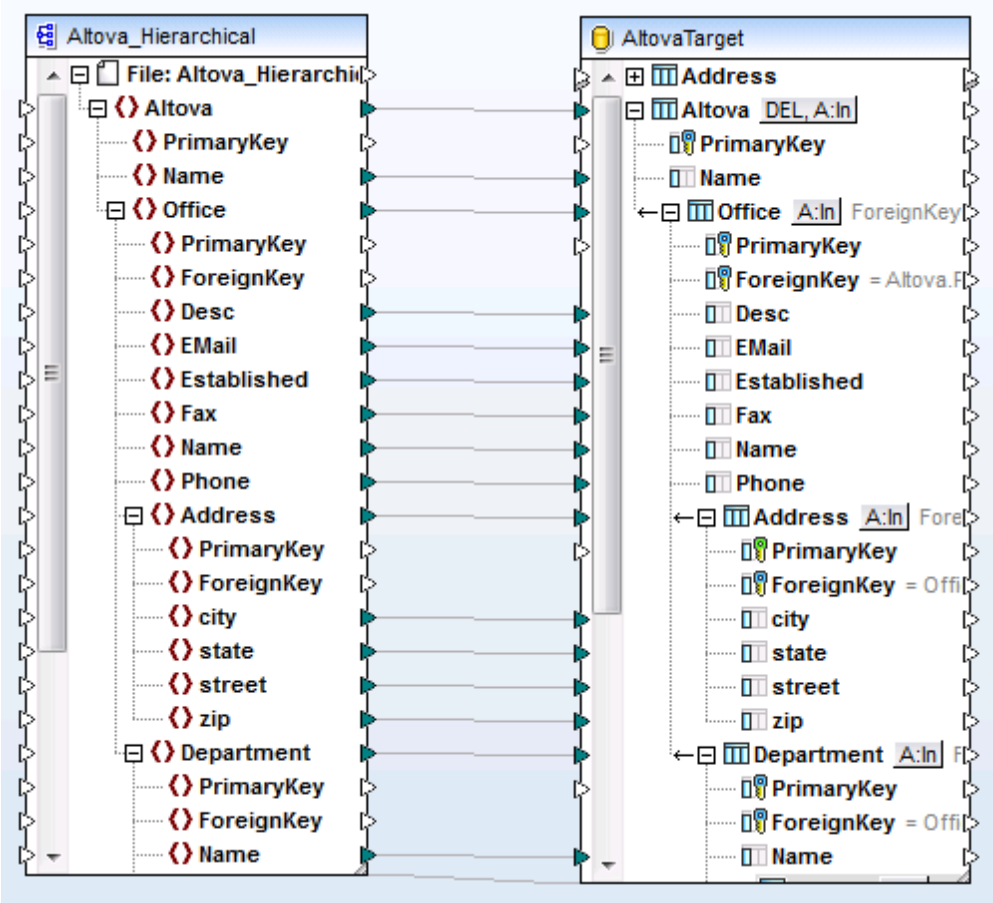
When you choose Java, C#, or C++ as transformation language, MapForce generates the required projects and solutions so that you can open them directly in Visual Studio or Eclipse, and run the generated data mapping program. For advanced data integration scenarios, you can also extend the generated program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

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



Sample data transformation between two XML files

Likewise, when working with databases in MapForce Professional or Enterprise Editions, you can see any database column in the MapForce mapping area and map data to or from it by making visual connections. As with other Altova MissionKit products, when setting up a database connection from MapForce, you can flexibly choose the database driver and the connection type (ADO, ODBC, or JDBC) according to your existing infrastructure and data mapping needs. Additionally, you can visually build SQL queries, use stored procedures, or query a database directly (support varies by database type, edition and driver).



Sample data transformation between an XML file and a database

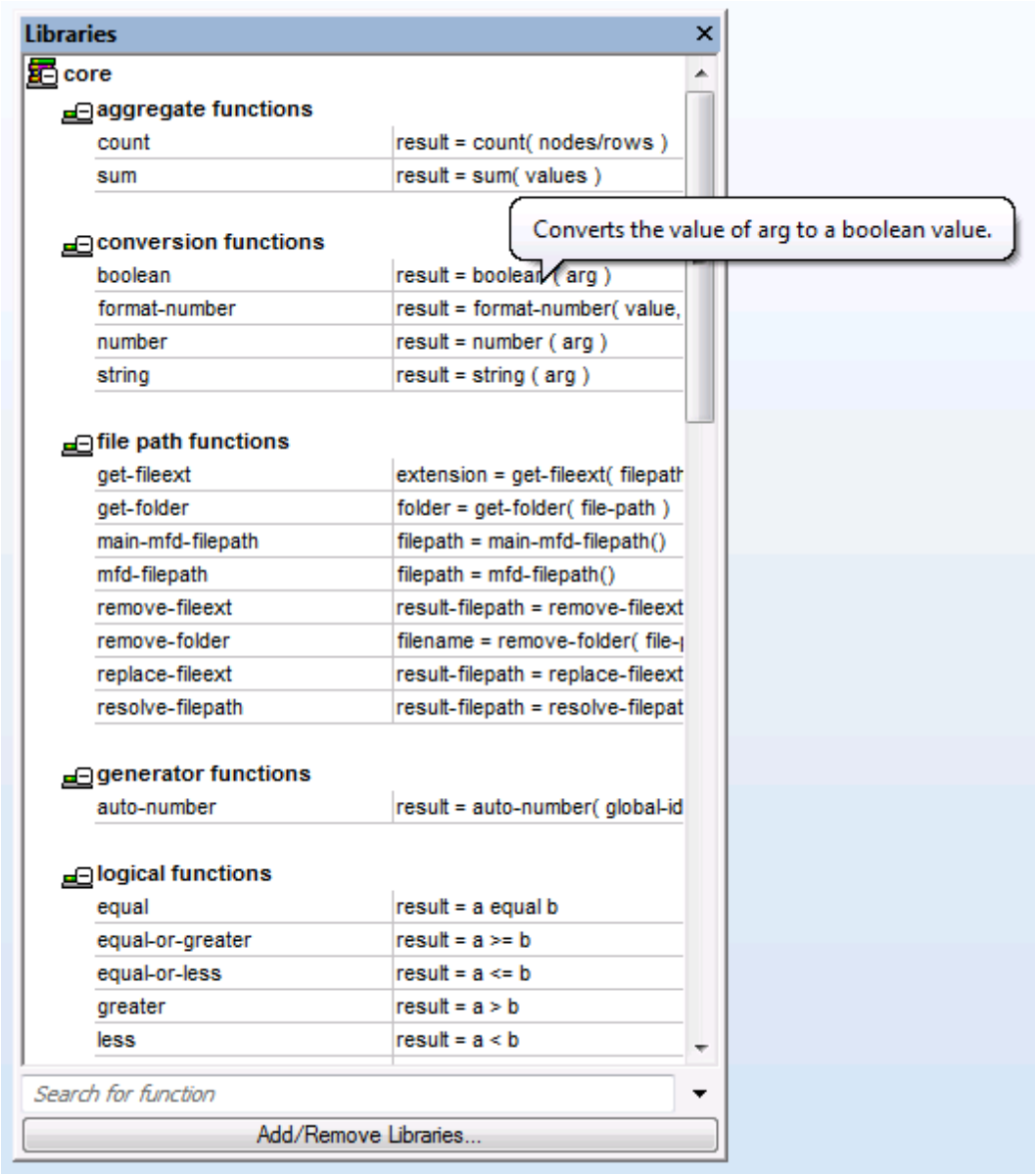
In a very simple scenario, a mapping design created with MapForce could be resumed as "read data from the source X and write it to target Y". However, you can easily design MapForce scenarios such as "read data from the source X and write it to target Y, and then read data from the source Y and write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

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

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

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

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



Libraries pane (MapForce Basic Edition)

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

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

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

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

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

<table>
<tr><th>MapForce Basic Edition</th><th>MapForce Professional Edition</th><th>MapForce Enterprise Edition</th></tr>
<tr><td colspan="3">You can choose to run the generated XSLT directly in MapForce and preview the data transformation result immediately. When you need increased performance, you can process the mapping using RaptorXML Server, an ultra-fast XML transformation engine.</td></tr>
<tr><td colspan="3">If XMLSpy is installed on the same machine, you can conveniently open and edit any supported file types, by opening XMLSpy directly from the relevant MapForce contexts (for example, the Component | Edit Schema Definition in XMLSpy menu command is available when you click an XML component).</td></tr>
<tr><td></td><td colspan="2">You can run data transformations either directly in MapForce, or deploy them to a different machine and even operating system for command-line or automated execution. More specifically, you can design mappings on Windows, and run them on a Windows, Linux, or Mac server machine which runs MapForce Server or FlowForce Server.</td></tr>
<tr><td></td><td colspan="2">If StyleVision is installed on the same machine, you can design or reuse existing StyleVision Power Stylesheets and preview the result of the mapping transformations as HTML, RTF, PDF, or Word 2007+ documents.</td></tr>
</table>

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

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

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

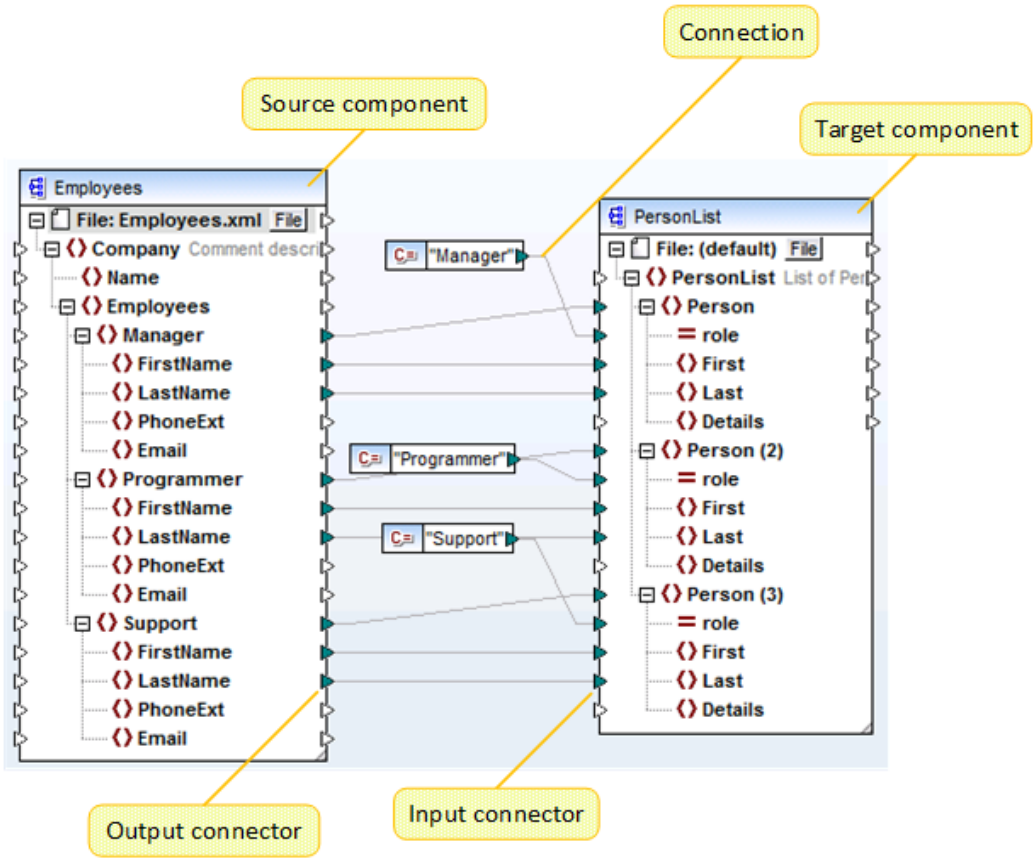
All editions of MapForce are available as a 32-bit application. The MapForce Professional and Enterprise editions are additionally available as a 64-bit application.

2.2 Basic Concepts

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

Mapping

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



Basic structure of a MapForce mapping

Component

In MapForce, the term "component" is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of

any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Databases
- Filters
- Conditions
- Flat files (CSV, fixed-length, and other text files)
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- Variables
- XBRL documents
- XML Schemas and DTDs

Connector

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

Connection

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

Source component

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

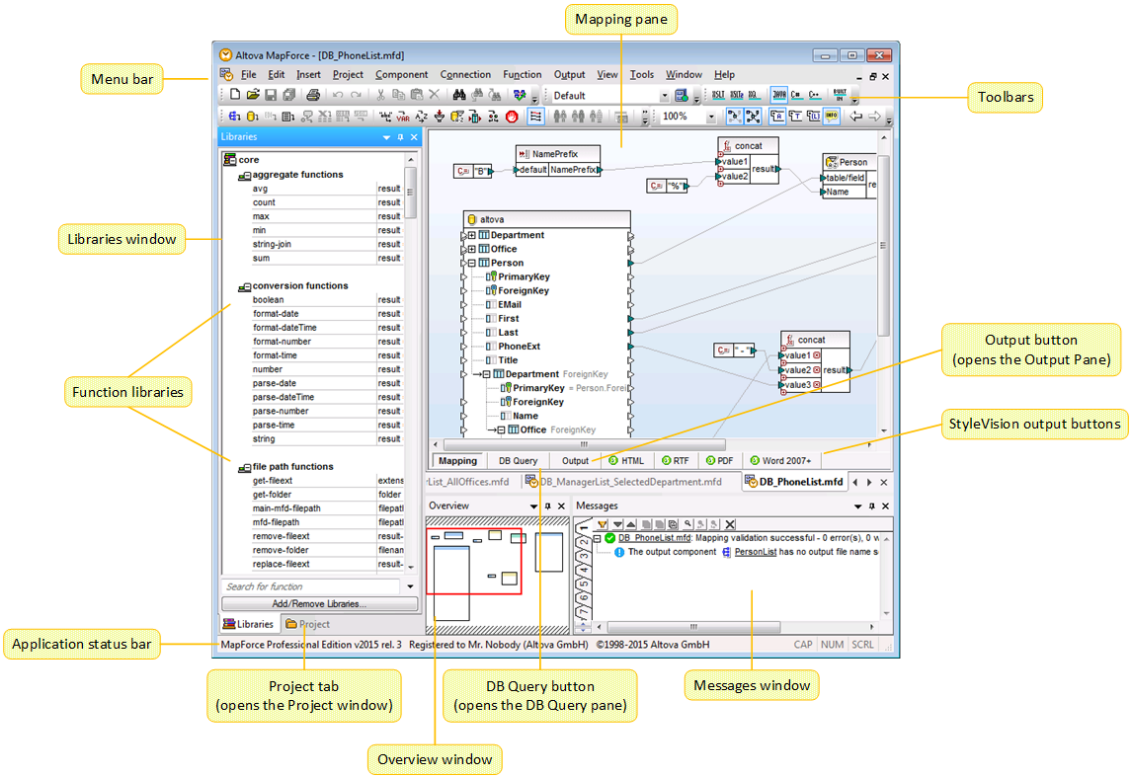
Target component

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

2.3 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**.



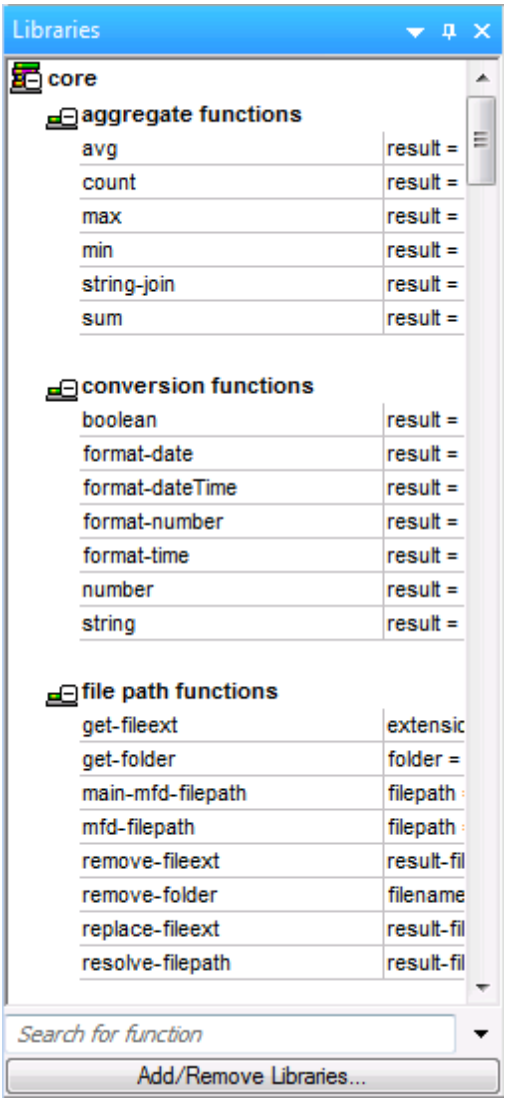
MapForce graphical user interface (MapForce Professional Edition)

Menu Bar and Toolbars

The Menu Bar displays the menu items. Each toolbar displays a group of buttons representing MapForce commands. You can reposition the toolbars by dragging their handles to the desired locations.

Libraries window

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

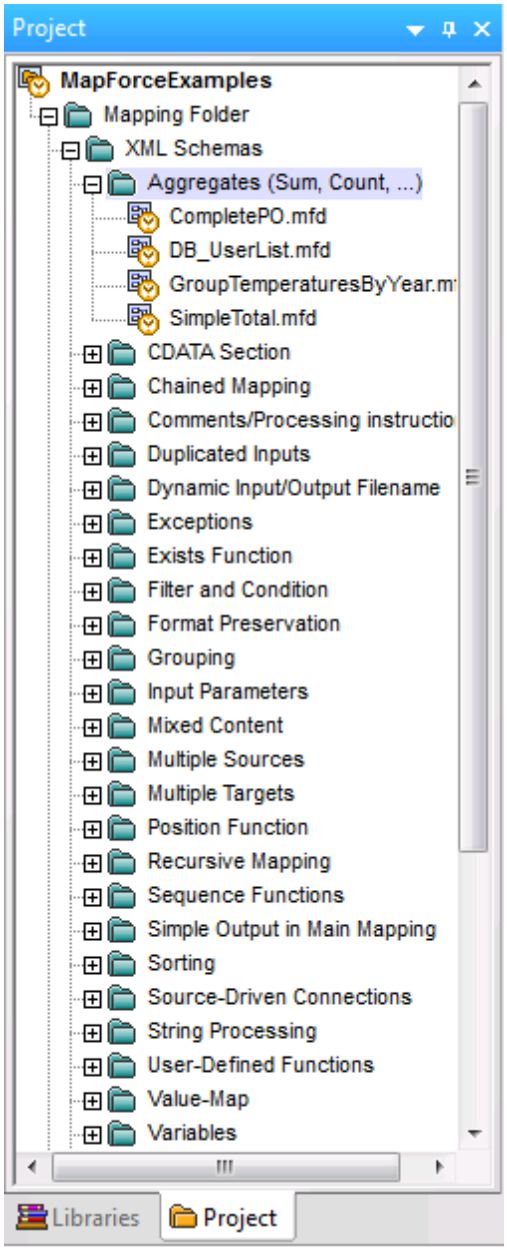


To search functions by name or by description, enter the search value in the text box at the bottom of the **Libraries** window. To find all occurrences of a function (within the currently active mapping), right-click the function, and select **Find All Calls** from the context menu. You can also view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

Project window

MapForce supports the Multiple Document Interface, and allows you to group your mappings into mapping projects. The Project window shows all files and folders that have

been added to the project. Project files have the extension *.mfp (MapForce Project). To search for mappings inside projects, click anywhere inside the Projects window, and press **CTRL + F**. For more information, see Working with Mapping Projects.



Mapping pane

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

XSLT (XSLT2) pane

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

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

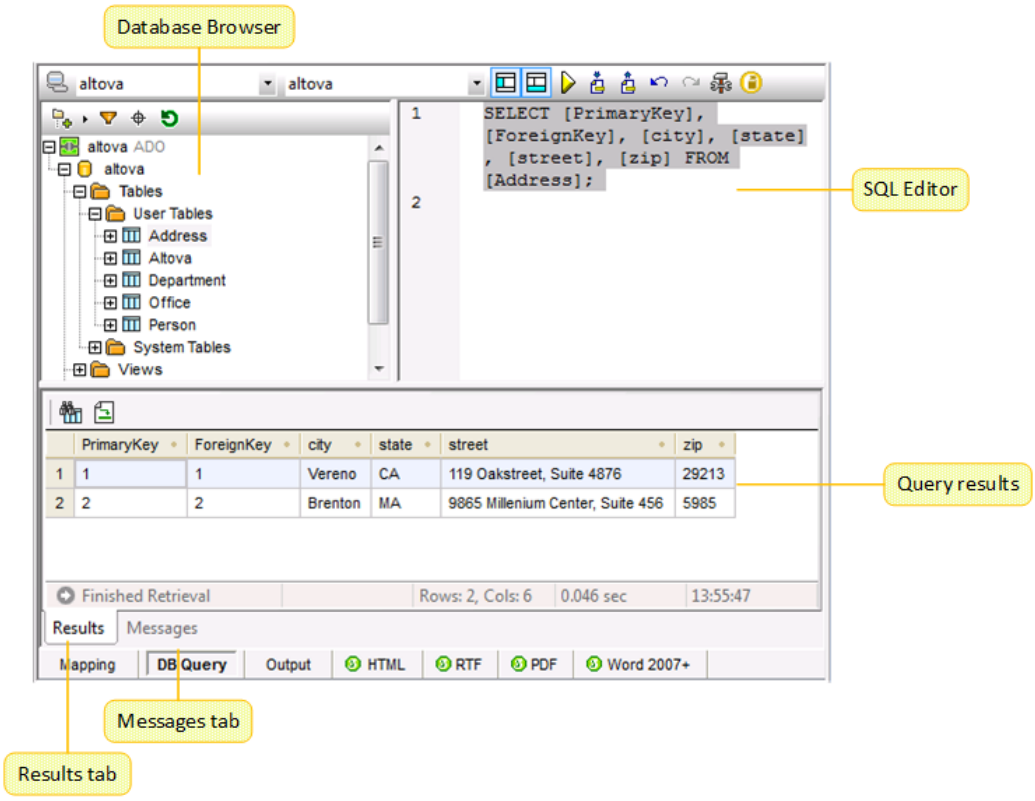
To configure the display settings (including indentation, end of line markers, and others), right-click the pane, and select **Text View Settings** from the context menu. Alternatively, click the **Text View Settings** ( ) toolbar button.

XQuery pane

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

DB Query pane

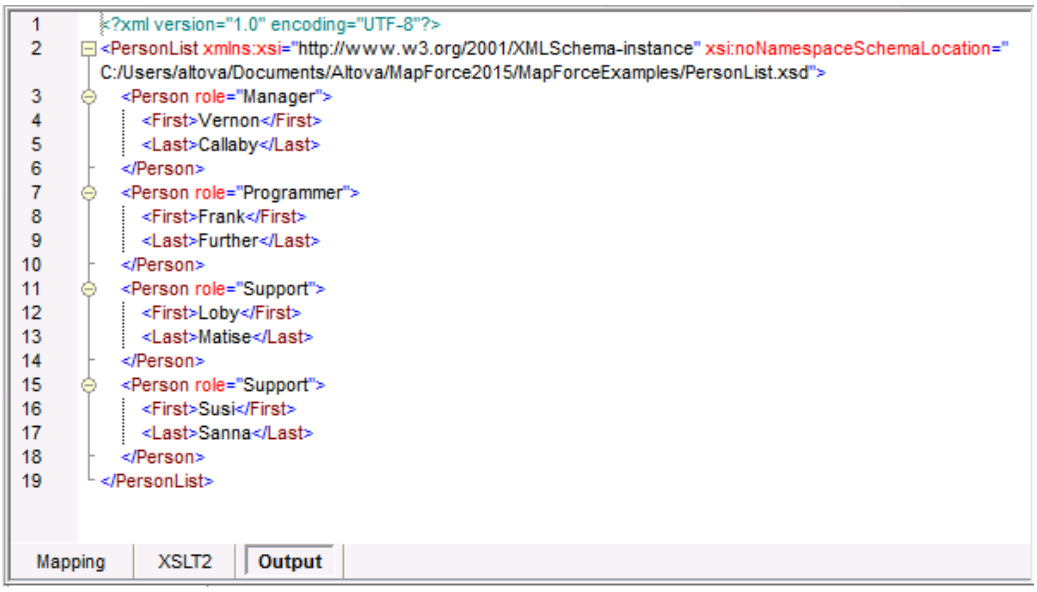
The DB Query pane allows you to directly query any major database. You can work with multiple active connections to different databases.



For more information, see Browsing and Querying Databases.

Output pane

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



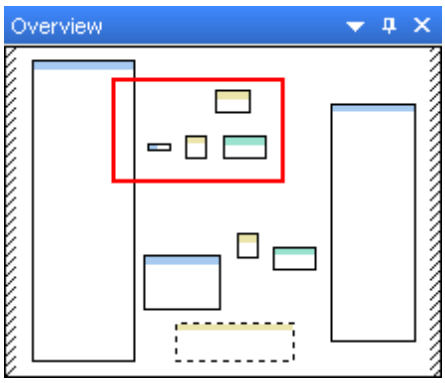
This pane also provides line numbering and code folding functionality, which works in a similar way as in the XSLT pane (see above).

StyleVision Output buttons

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

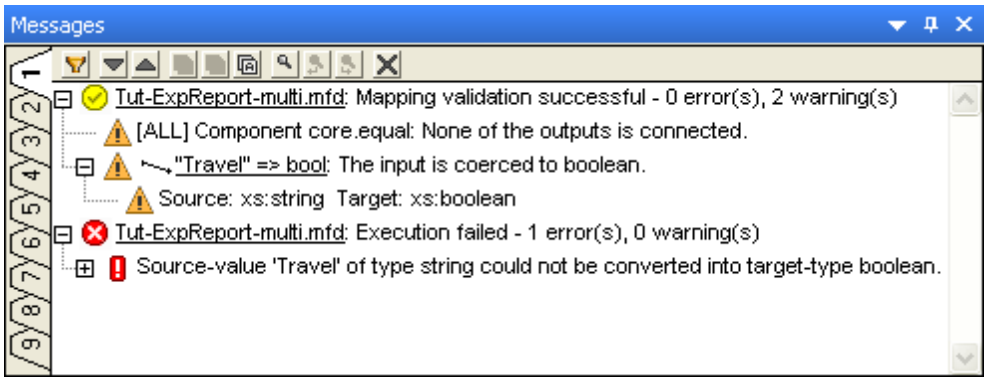
Overview window

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



Messages window

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



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

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

Icon	Description
	Operation completed successfully.
	Operation completed with warnings.
	Operation has failed.

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

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

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

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

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

Application status bar

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

2.4 Conventions

Example files

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

- (My) Documents\Altova\MapForce2016\MapForce Examples
- (My) Documents\Altova\MapForce2016\MapForce Examples\Tutorials

The location of **(My) Documents** folder depends on your version of Windows.

Windows 10 Windows 8 Windows 7 Windows Vista Windows Server 2008 Windows Server 2012	`C:\Users\<username>\Documents`
Windows XP Windows Server 2003	`C:\Documents and Settings\<username>\My Documents`

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

Graphical user interface

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

Chapter 3

Tutorials

3 Tutorials

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

Convert XML to New Schema

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

Map Multiple Sources to One Target

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

Work with Multiple Target Schemas

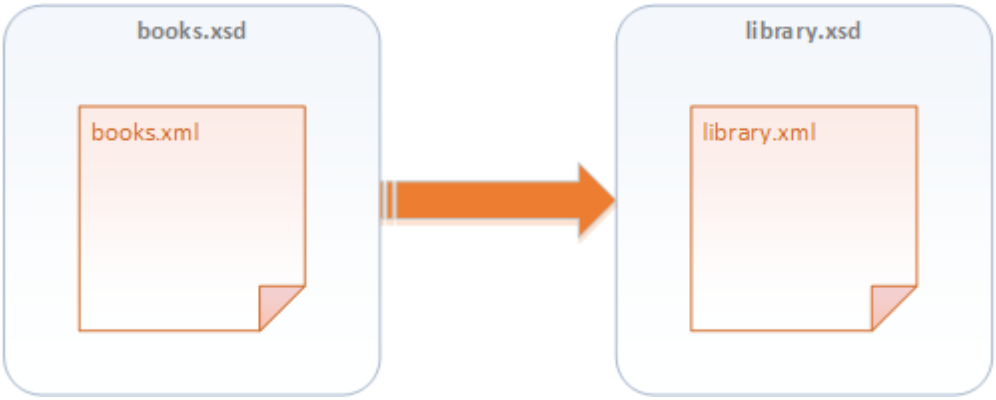
This tutorial shows you how to work with more complex mapping that produce two or more target outputs. More specifically, you will learn how to generate, in the same mapping, an XML file that stores a list of book records, and another XML file that contains only a subset of the books in the first file,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

Process and Generate Files Dynamically

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

3.1 Convert XML to New Schema

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



Abstract model of the data transformation

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

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

books.xml

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

```
<library>
   <last_updated>2015-06-02T16:26:55+02:00</last_updated>
   <publication>
      <id>1</id>
      <author>Mark Twain</author>
```

```
      <title>The Adventures of Tom Sawyer</title>
      <genre>Fiction</genre>
      <publish_year>1876</publish_year>
   </publication>
   <publication>
      <id>2</id>
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <genre>Fiction</genre>
      <publish_year>1912</publish_year>
   </publication>
</library>
```

library.xml

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

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

Step 1: Select XSLT2 as transformation language

You can do this in one of the following ways:

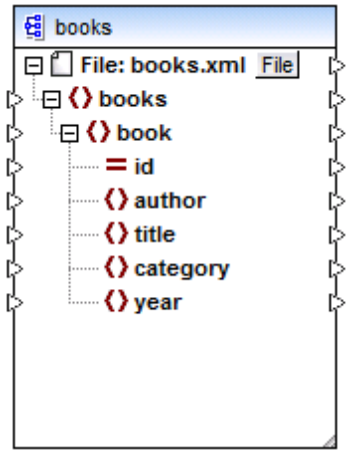
- Click the **XSLT2** () toolbar button.
- On the **Output** menu, click **XSLT 2.0**.

Step 2: Add the source XML file to the mapping

The source XML file for this mapping is located at the following path: **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial\books.xml**. You can add it to the mapping in one of the following ways:

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

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



Mapping component

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

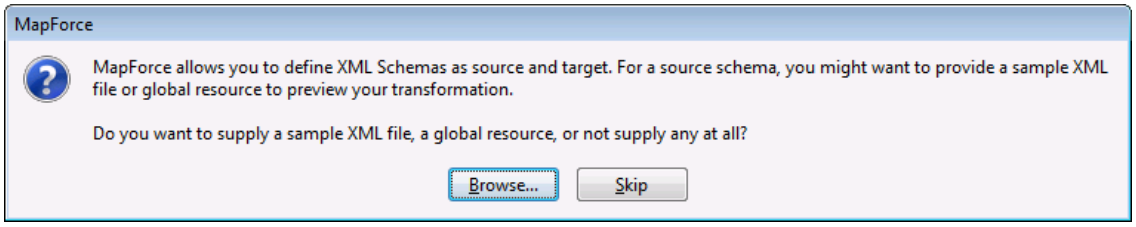
The top level node represents the file name; in this particular case, its title displays the name of the XML instance file. The XML elements in the structure are represented by the icon, while XML attributes are represented by the icon.

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

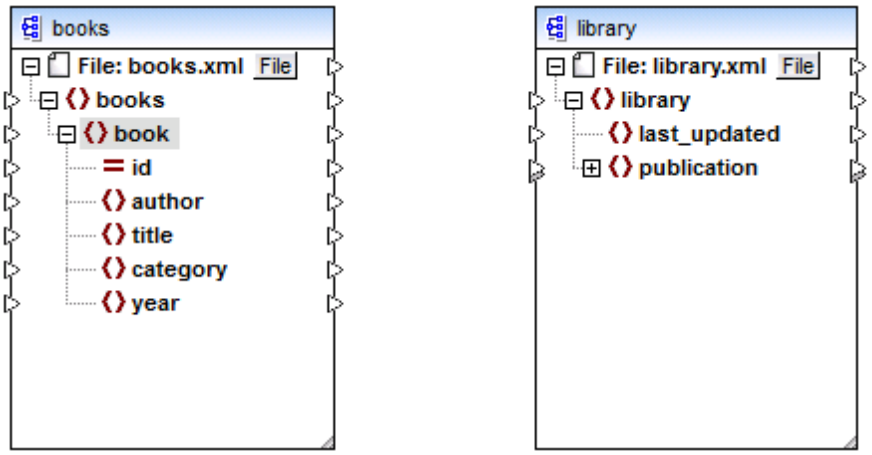
Step 3: Add the target XML schema to the mapping

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

For this particular example, we are using an existing schema file available at: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\library.xsd**. To add it to the mapping, follow the same steps as with the source XML file (that is, click the **Insert XML Schema/File** ( ) toolbar button). Click **Skip** when prompted by MapForce to supply an instance file.



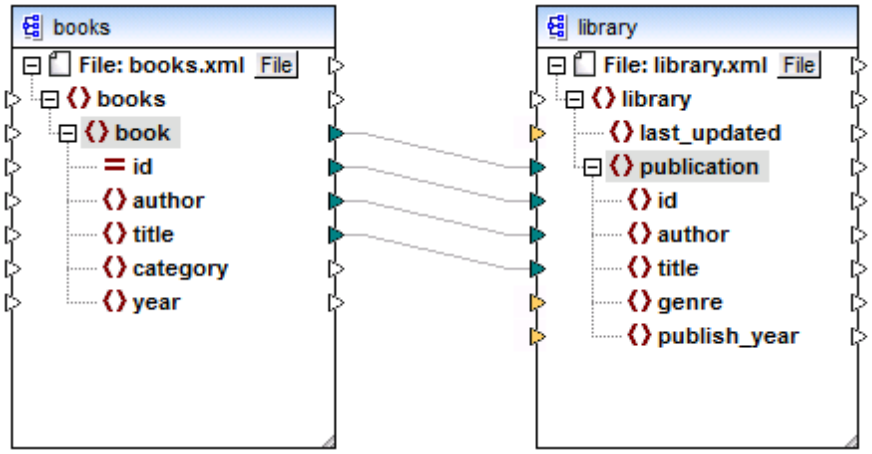
At this stage, the mapping design looks as follows:



Step 4: Make the connections

For each <book> in the source XML file, we want to create a new <publication> in the target XML file. We will therefore create a mapping connection between the <book> element in the source component and the <publication> element in the target component. To create the mapping connection, click the output connector (the small triangle) to the right of the <book> element and drag it to the input connector of the <publication> element in the target.

When you do this, MapForce may automatically connect all elements which are children of <book> in the source file to elements having the same name in the target file; therefore, four connections are being created simultaneously. This behavior is called "Auto Connect Matching Children" and it can be disabled and customized if necessary.



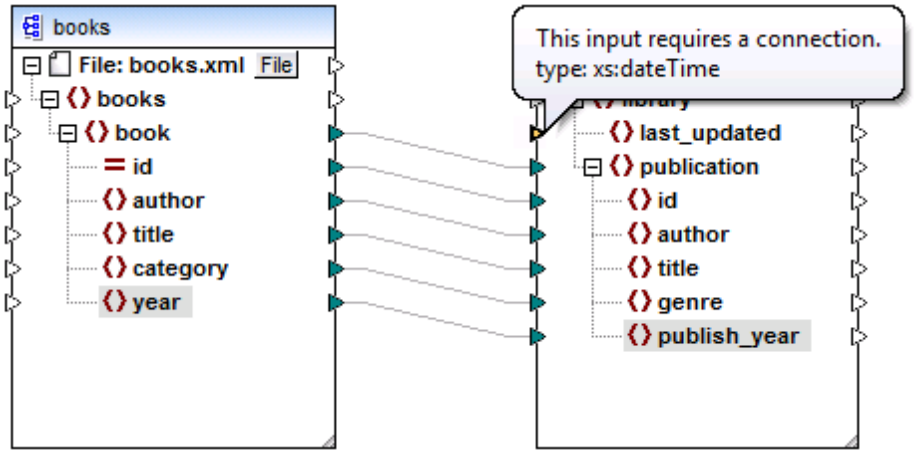
You can enable or disable the "Auto Connect Matching Children" behavior in one of the following ways:

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

Notice that some of the input connectors on the target component have been highlighted by MapForce in orange, which indicates that these items are mandatory. To ensure the validity of the target XML file, provide values for the mandatory items as follows:

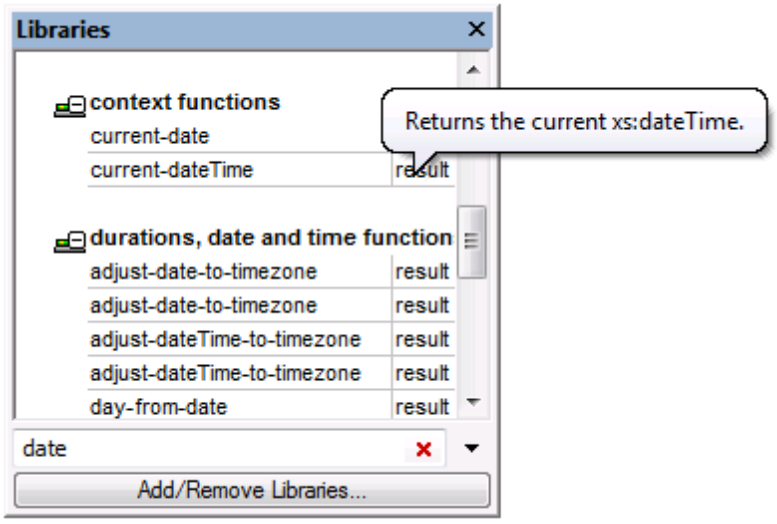
- Connect the `<category>` element in the source with the `<genre>` element in the target
- Connect the `<year>` element in the source with the `<publish_year>` element in the target

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



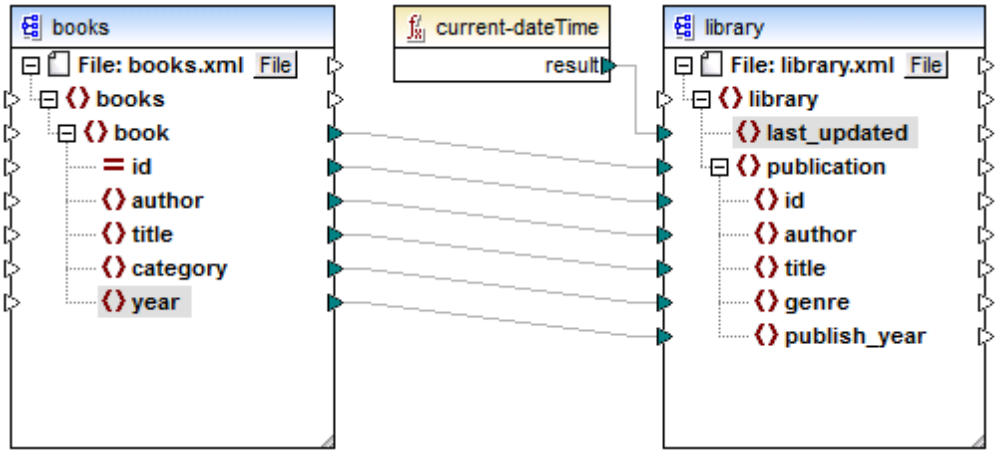
You can also make the data type of each item visible at all times, by clicking the **Show Data Types** ( ) toolbar button.

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

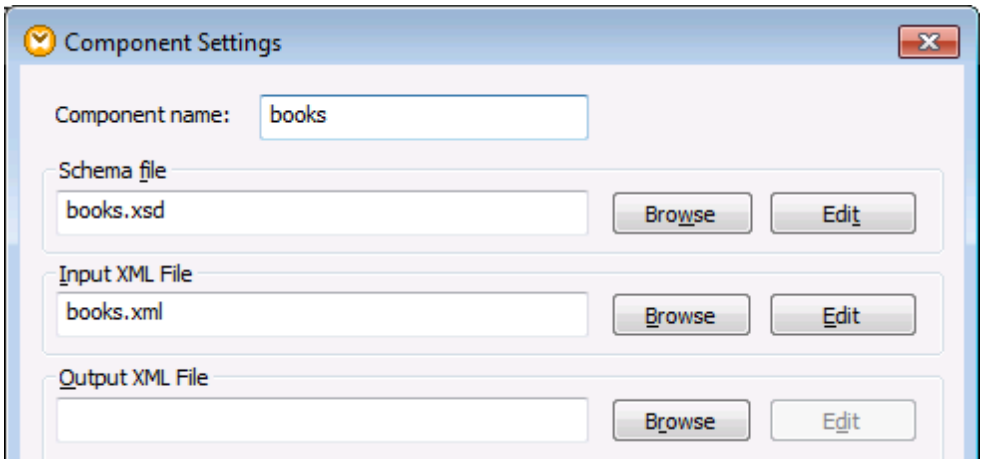


As shown above, if you move the mouse over the "result" part of the function, you can see its description. For tips to be displayed, make sure that the **Show tips** ( ) toolbar button is enabled.

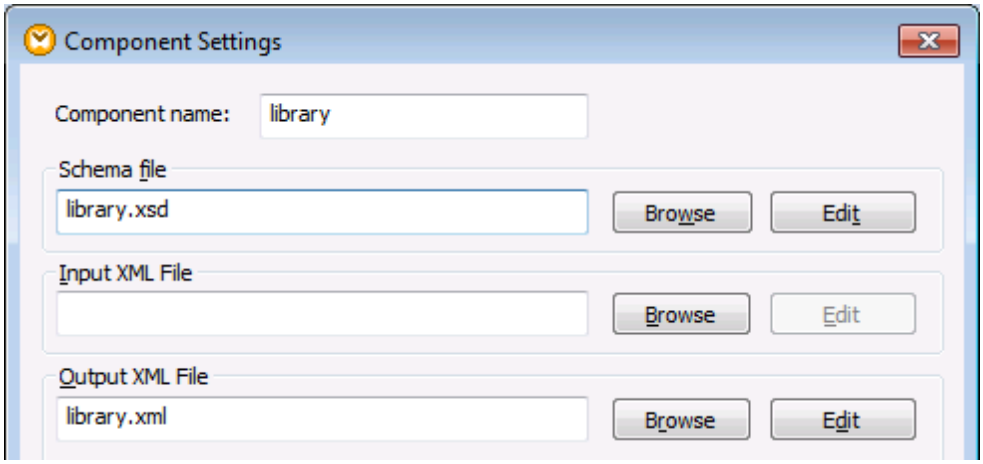
To add the function to the mapping, drag the function into the mapping pane, and connect its output to the input of the `<last_updated>` element.



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



Component settings for the source



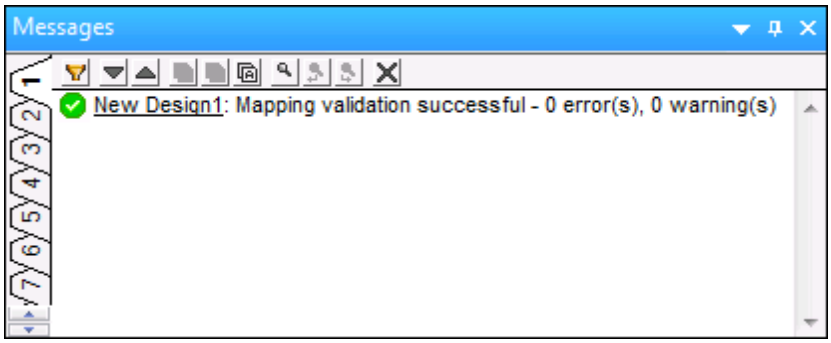
Component settings for the target

Step 5: Validate and save the mapping

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

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

The Messages window displays the validation results:



Messages window

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

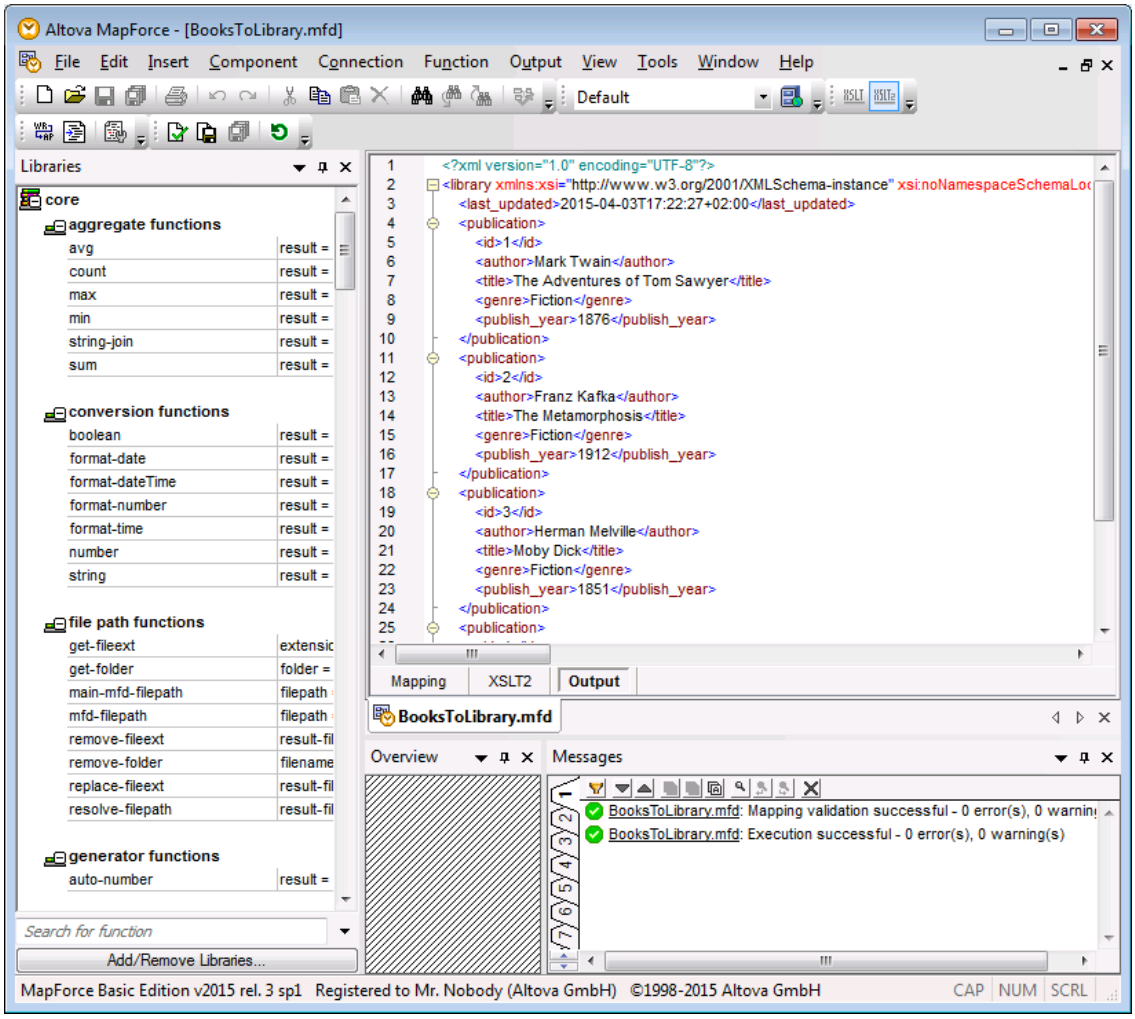
- On the **File** menu, click **Save**.
- Click the **Save** ( ) toolbar button.

For your convenience, the mapping created in this tutorial is available at the following path:

<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\\BooksToLibrary.mfd. Therefore, from this point onwards, you can either continue with the mapping file you created, or with the **BooksToLibrary.mfd** file.

Step 6: Preview the mapping result

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



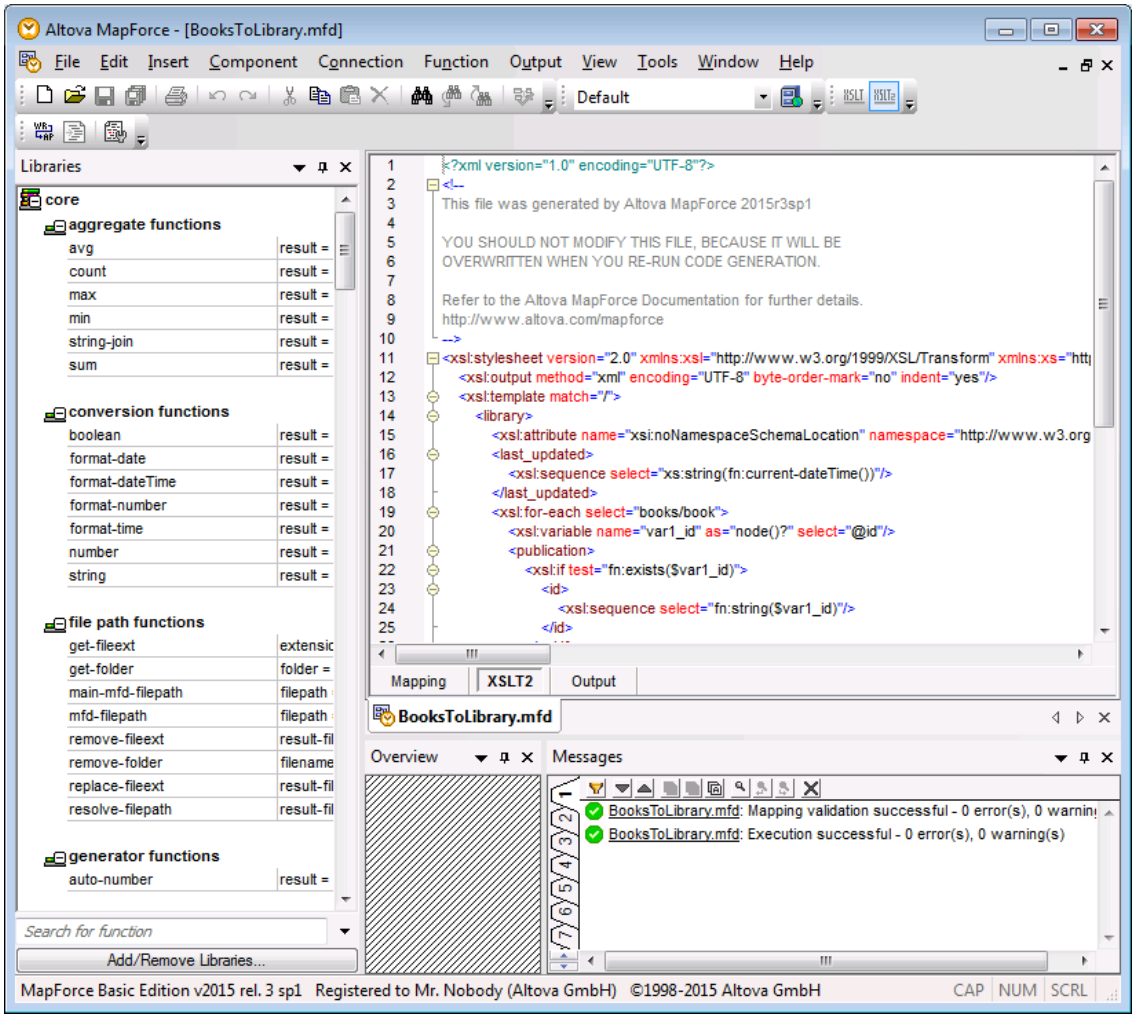
Output pane

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

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

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

You can also preview the generated XSLT code that performs the transformation. To preview the code, click the **XSLT2** button located in the lower area of the mapping pane.



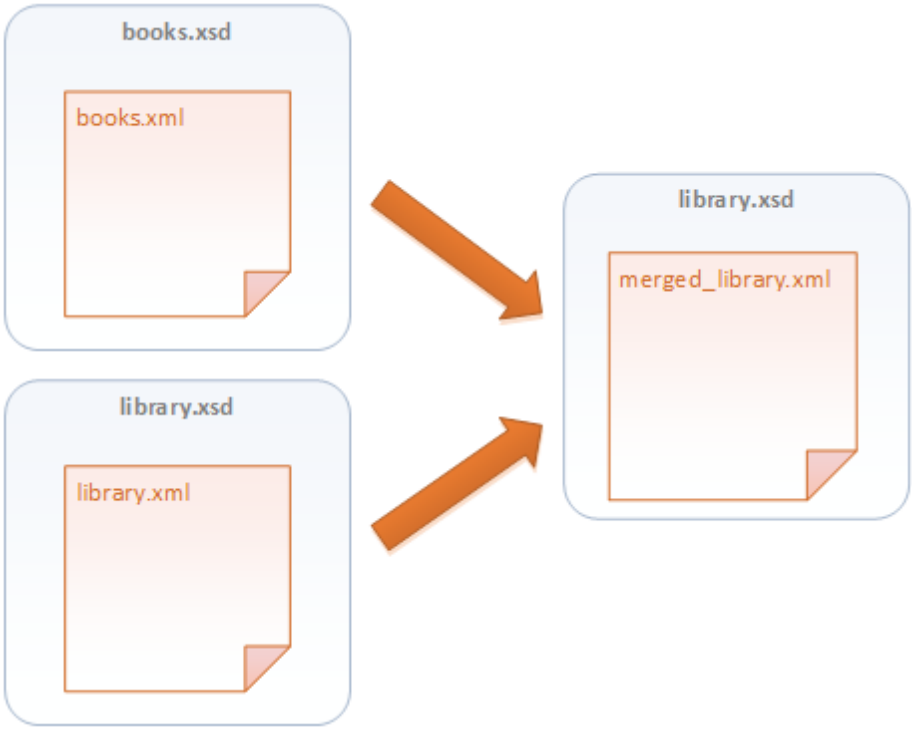
XSLT2 pane

To generate and save the XSLT2 code to a file, select the menu item **File | Generate Code in | XSLT 2.0**. When prompted, select a folder where the generated code must be saved. After code generation completes, the destination folder includes the following two files:

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

3.2 Map Multiple Sources to One Target

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



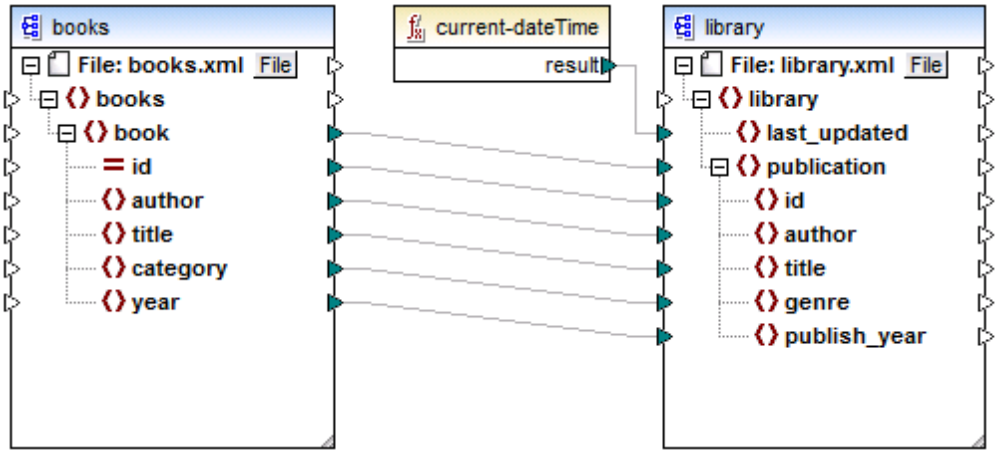
Abstract model of the data transformation

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

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **BooksToLibrary.mfd** mapping from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. You have already designed this mapping in the Convert XML to New Schema tutorial. To begin, open the **BooksToLibrary.mfd** file in MapForce, and save it with a new name.

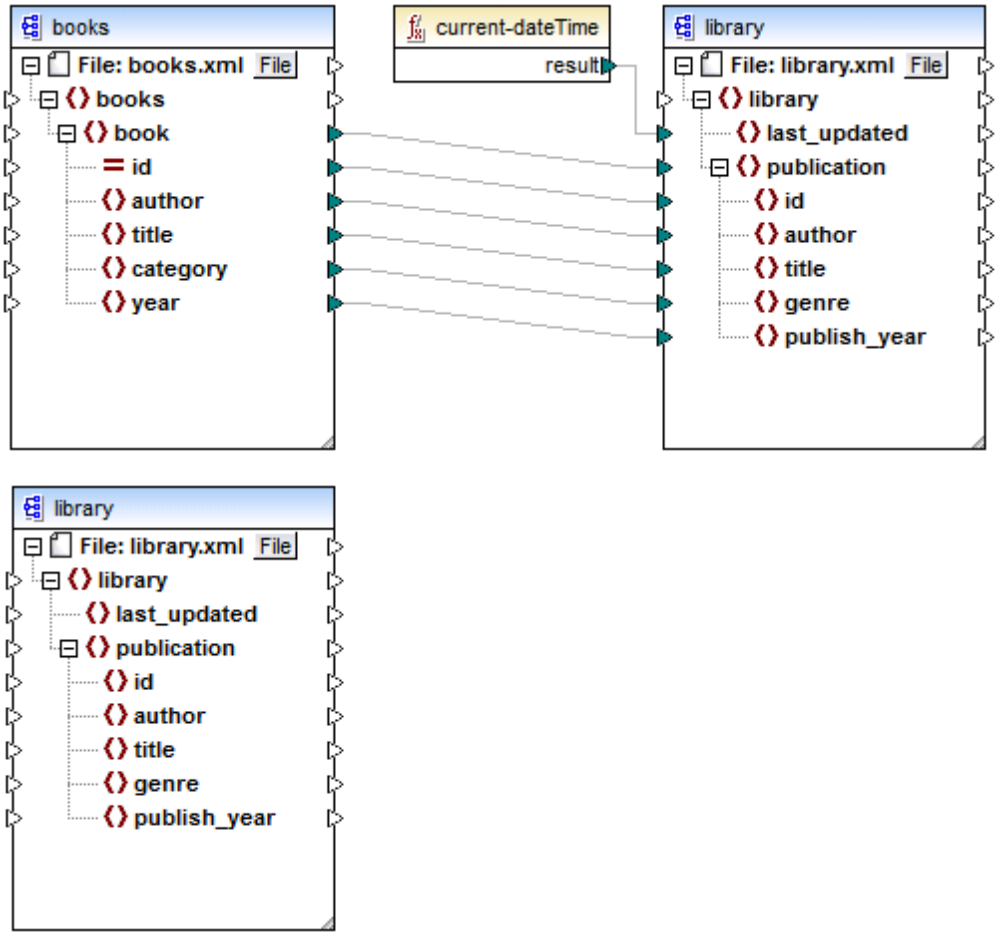
> Make sure to save the new mapping in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, because it references several files from it.



BooksToLibrary.mfd (MapForce Basic Edition)

Step 2: Create a second source component

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

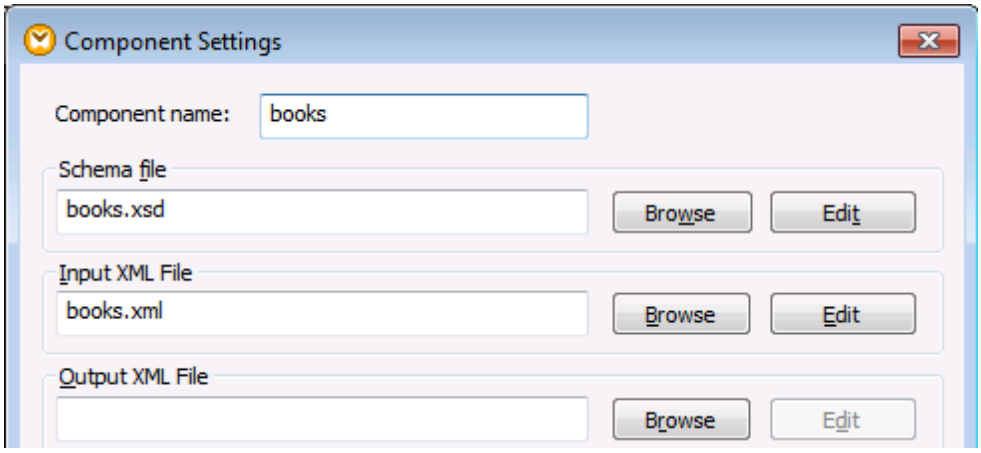


The mapping now has two source components: **books** and **library**, and one target component: **library**.

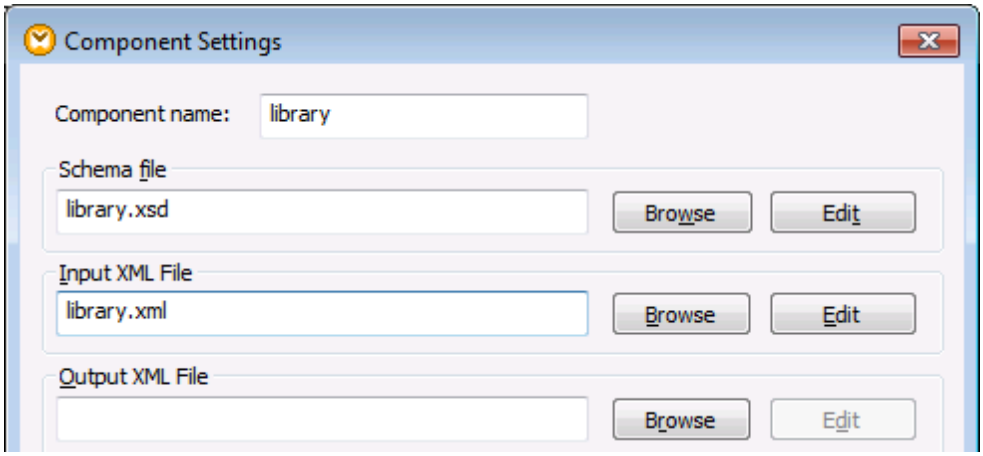
> You can always move the mapping components in any direction (left, right, top, bottom). Nevertheless, placing a source component to the left of a target component will make your mapping easier to read and understand by others. This is also the convention for all mappings illustrated in this documentation,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

Step 3: Verify and set the input/output files

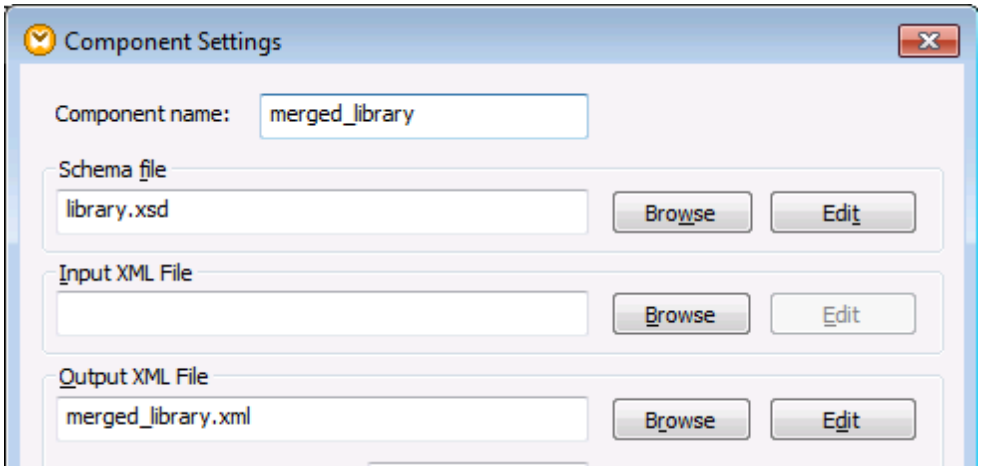
In the previous step, the new source component was copy-pasted from the target component, so it inherits the same settings. To ensure that the name input/output instance files are correctly set, double-click the header of each component, and, in the Component Settings dialog box, verify and change the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



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



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

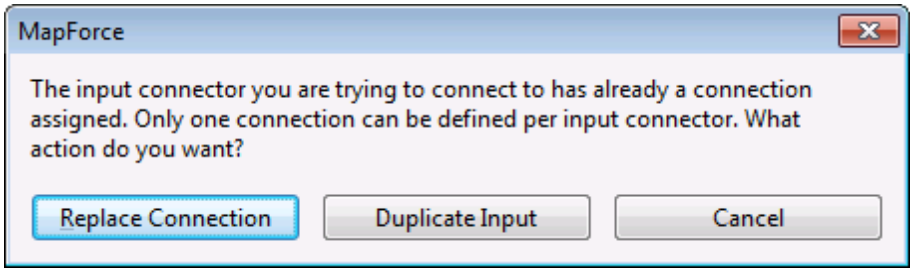


*Component settings for the target (**merged_library**)*

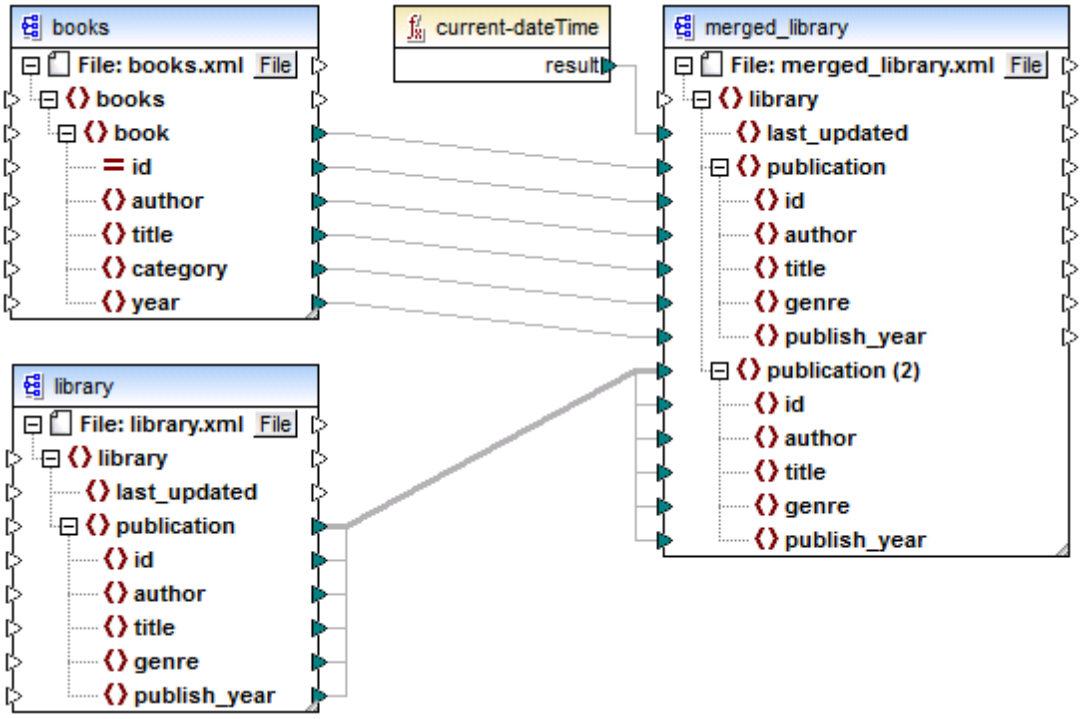
As shown above, the first source component reads data from **books.xml**. The second source component reads data from **library.xml**. Finally, the target component outputs data to a file called **merged_library.xml**.

Step 4: Make the connections

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



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



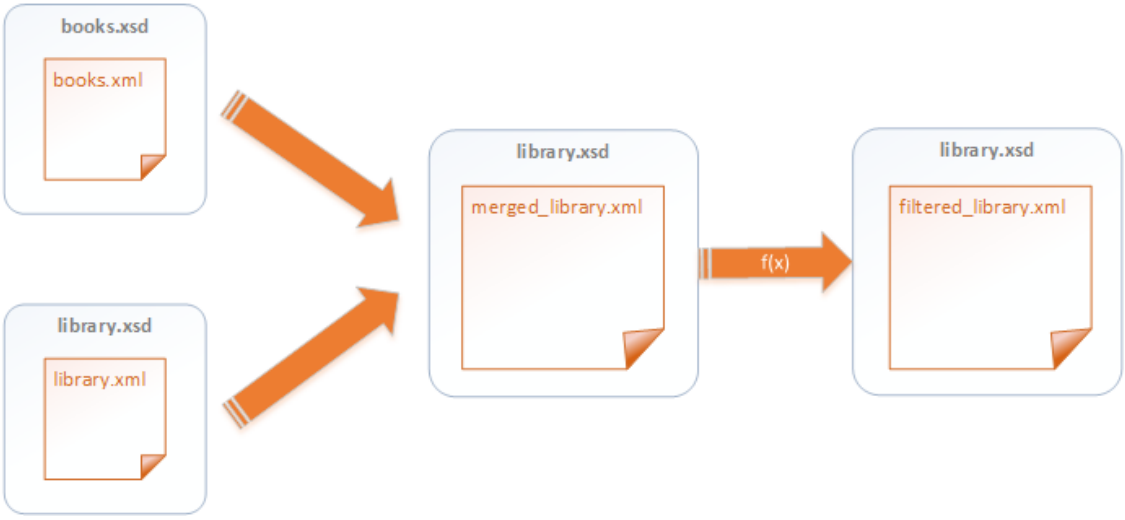
Notice that the `publication` item in the target component has now been duplicated. The new `publication(2)` node will accept data from the source **library** component. Importantly, even though the name of this node appears as `publication(2)` in the mapping, its name in the resulting XML file will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

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

3.3 Work with Multiple Target Schemas

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

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



Abstract model of the data transformation

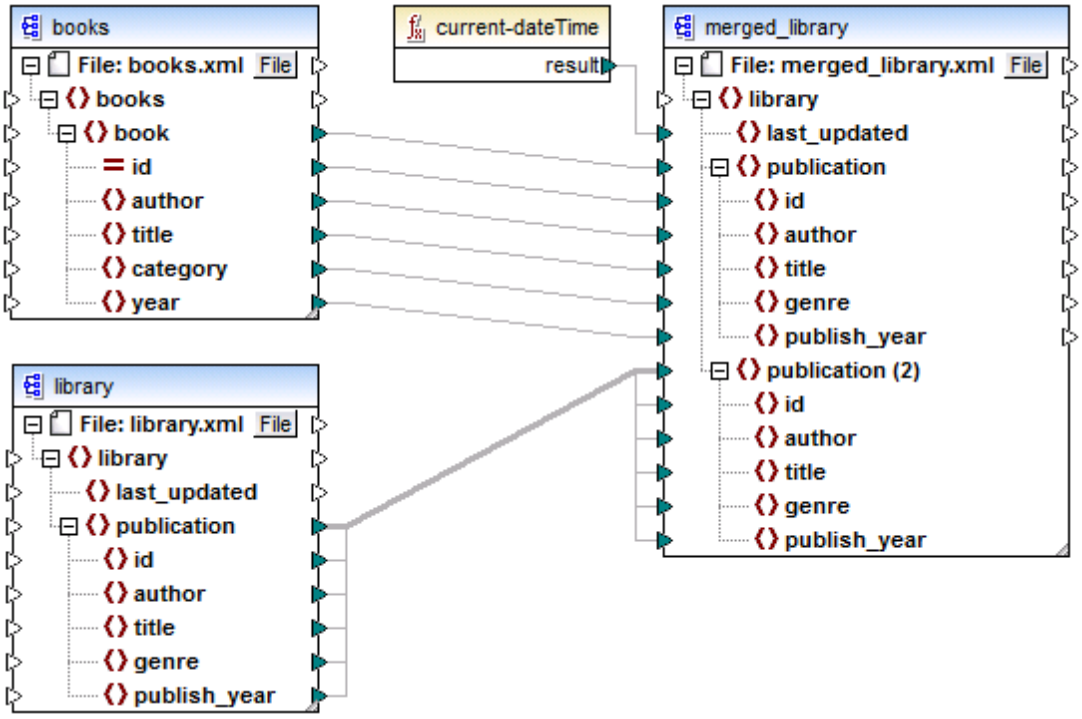
In the diagram above, the data is first merged from two different schemas (**books.xsd** and **library.xsd**) into a single XML file called **merged_library.xml**. Secondly, the data is transformed using a filtering function and passed further to the next component, which creates an XML file called **filtered_library.xml**.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

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

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **MultipleSourcesToOneTarget.mfd** mapping from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. You have already designed this mapping in the Map Multiple Sources to One Target tutorial. To begin, open the **MultipleSourcesToOneTarget.mfd** file in MapForce, and save it with a new name.

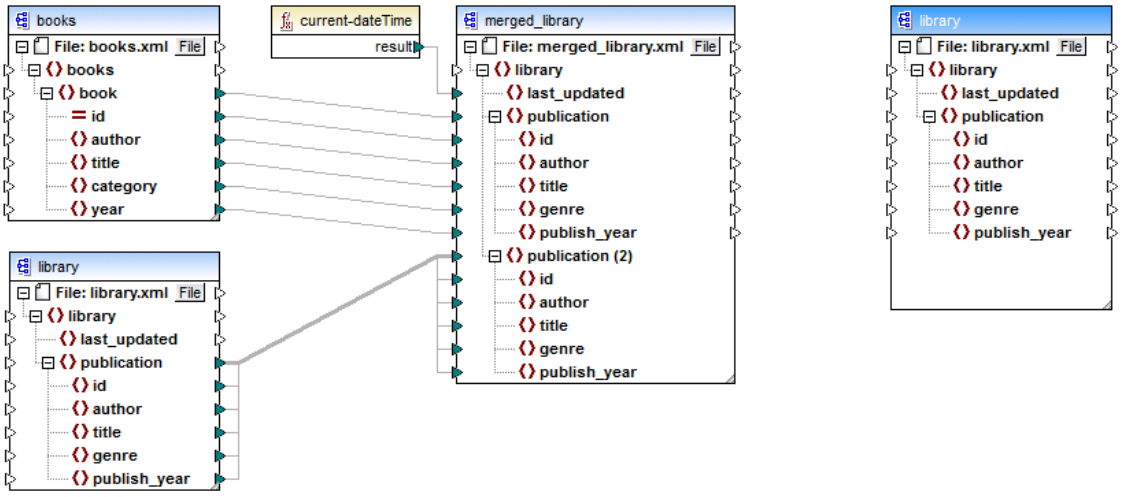
Make sure to save the new mapping in the **<Documents>\Altova\MapForce2016 \MapForceExamples\Tutorial** folder, because it references several files from it.



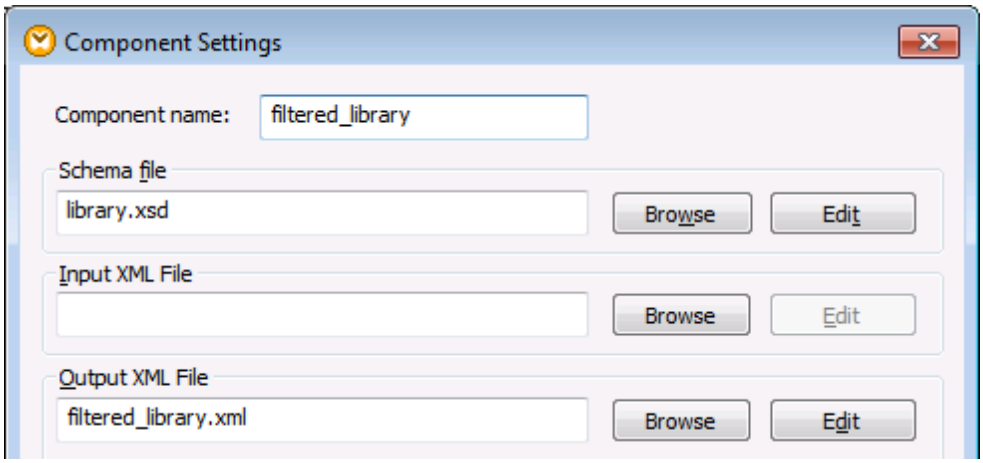
MultipleSourcesToOneTarget.mfd (MapForce Basic Edition)

Step 2: Add and configure the second target component

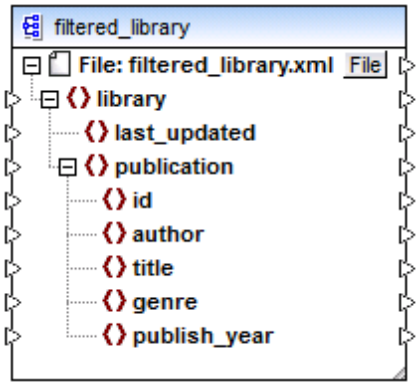
To add the second target component, click the **Insert XML Schema/File** ( ) toolbar button, and open the **library.xsd** file located in the **<Documents>\Altova\MapForce2016 \MapForceExamples\Tutorial** folder. Click **Skip** when prompted to supply a sample instance file. The mapping now looks as follows:



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

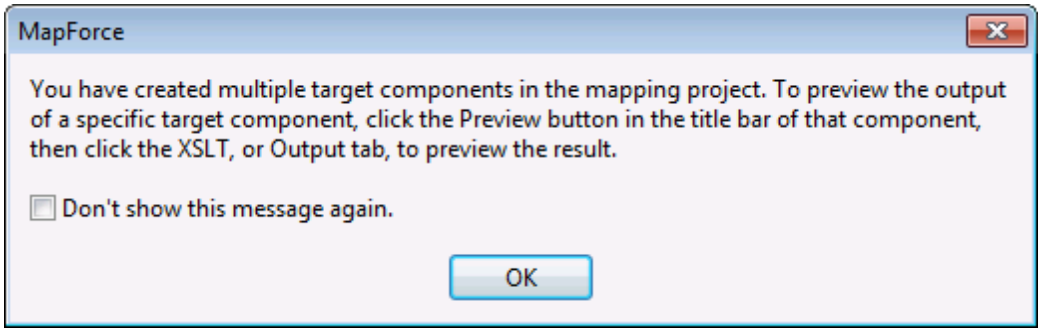


Notice that the new name of the component is **filtered_library**, and the output XML file is named **filtered_library.xml**.

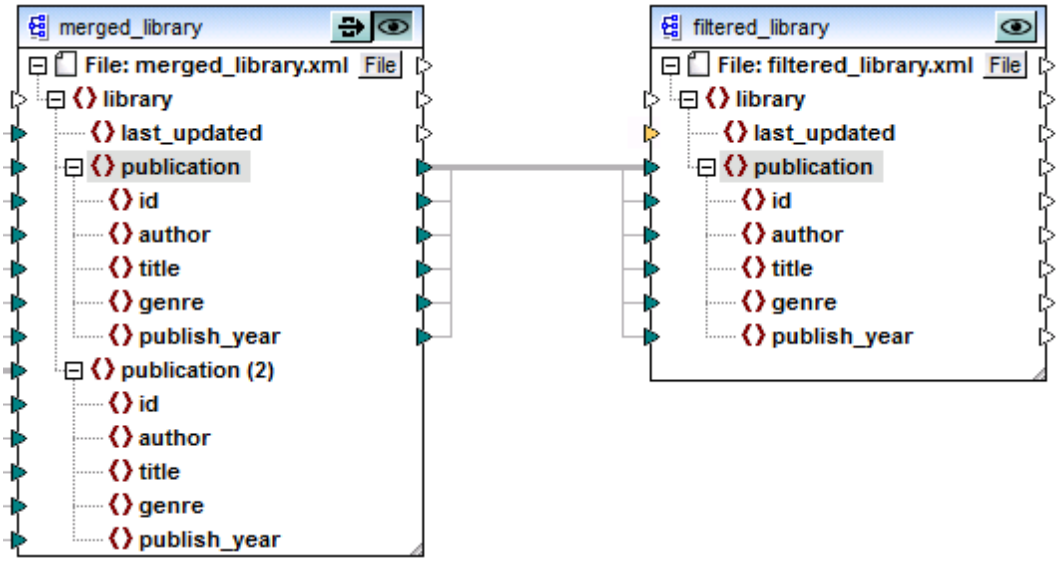


Step 3: Make the connections

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

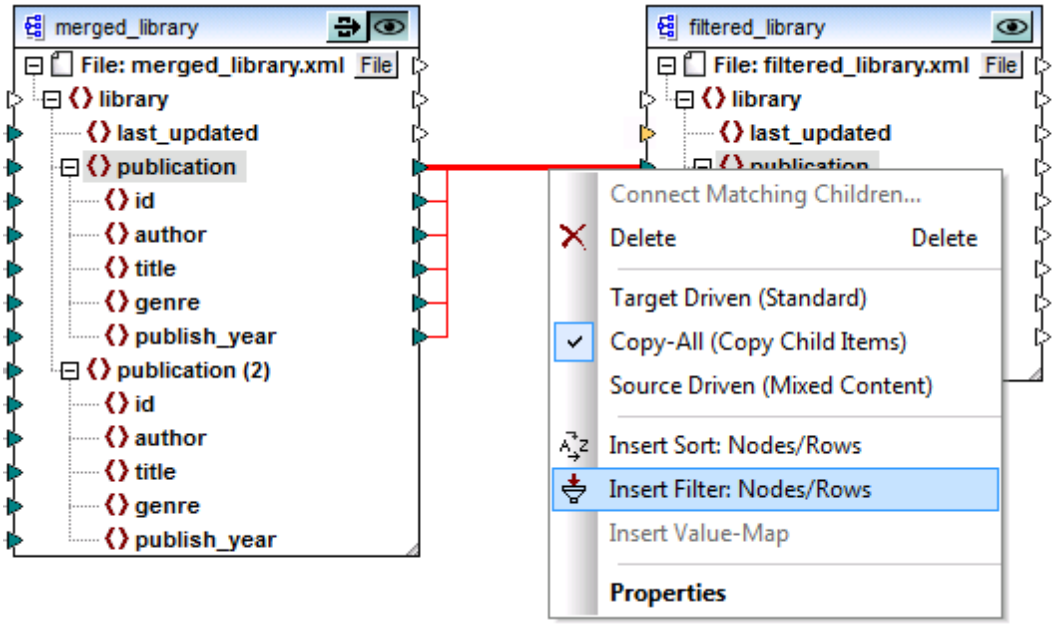


Click **OK**. Notice that new buttons are now available in the upper-right corner of both target components: **Preview** ( ) and **Pass-through** ( ). These buttons will be used and explained in the following steps.

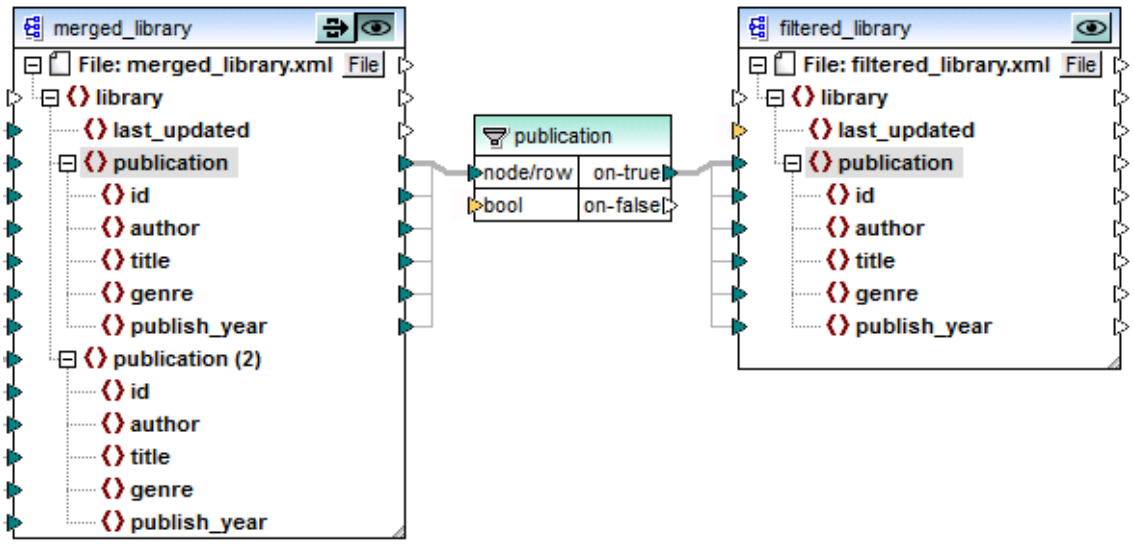


Step 4: Filter data

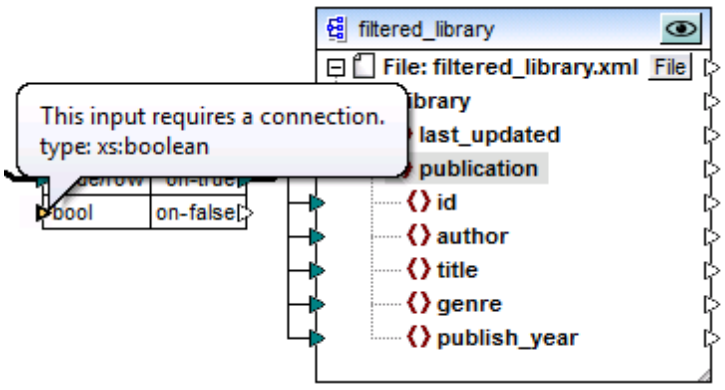
To filter data before supplying it to the **filtered_library**, we will use a **Filter** component. To add a filter component, right-click the connection between **merged_library** and **filtered_library**, and select **Insert Filter: Nodes/Rows** from the context menu.



The filter component has now been added to the mapping.



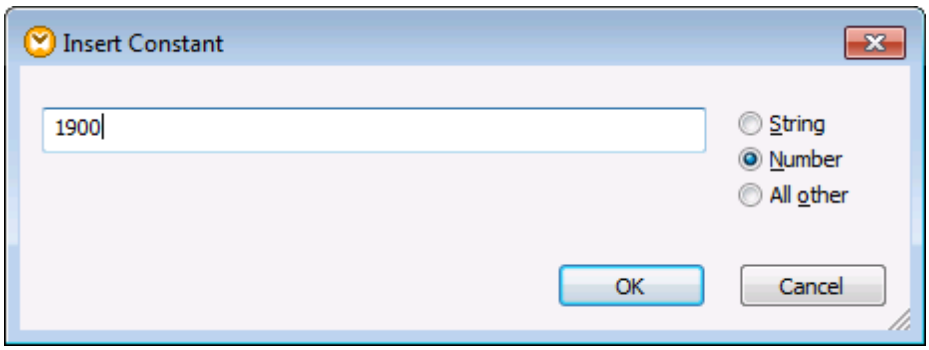
As shown above, the bool input connector is highlighted in orange, which suggests that an input is required. If you move the mouse over the connector, you can see that an input of type `xs:boolean` is required. Note that, for tips to be displayed, the **Show tips** ( ) toolbar button must be enabled.



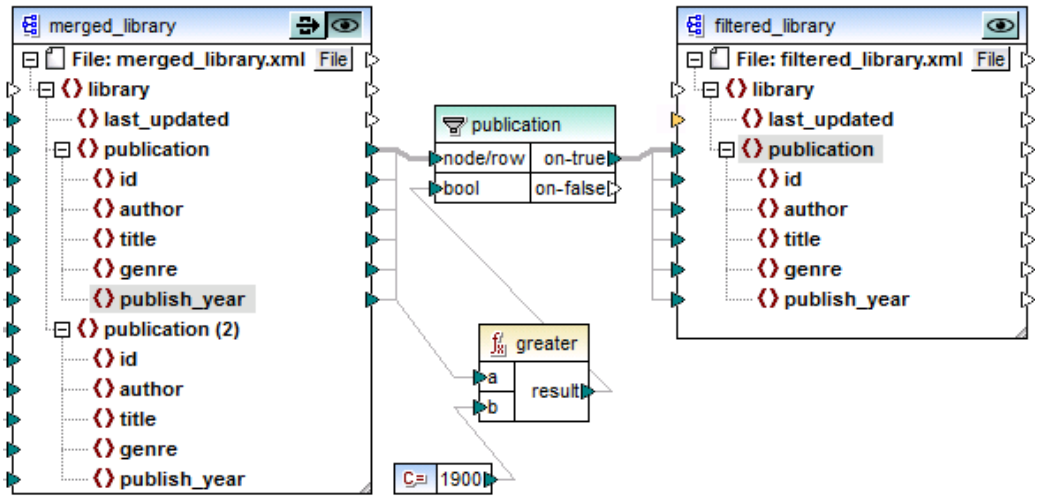
The filter component requires a condition that returns either `true` or `false`. When the Boolean condition returns `true`, data of the current **publication** sequence will be copied over to the target. When the condition returns `false`, data will not be copied.

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

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



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



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

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

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

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

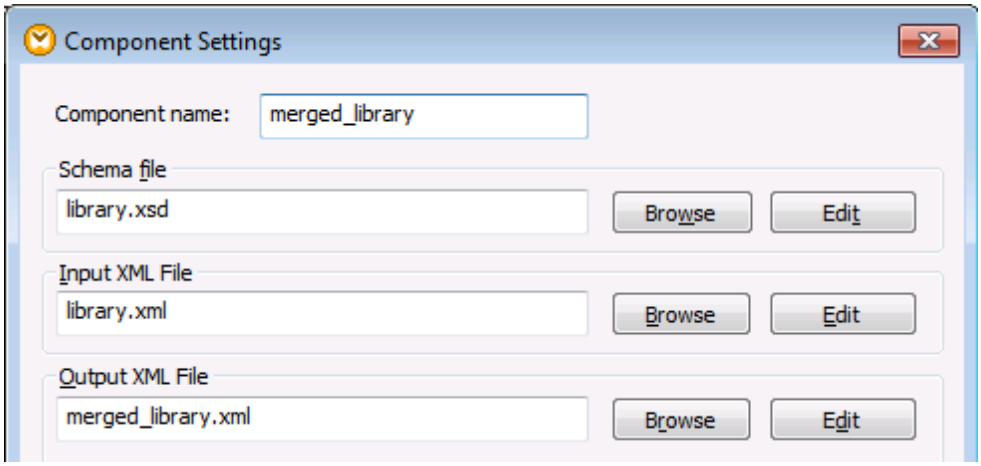
When you want to view and save the output of the **filtered_library** component :

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

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

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

and edit so as to supply a valid input XML file, as shown below:

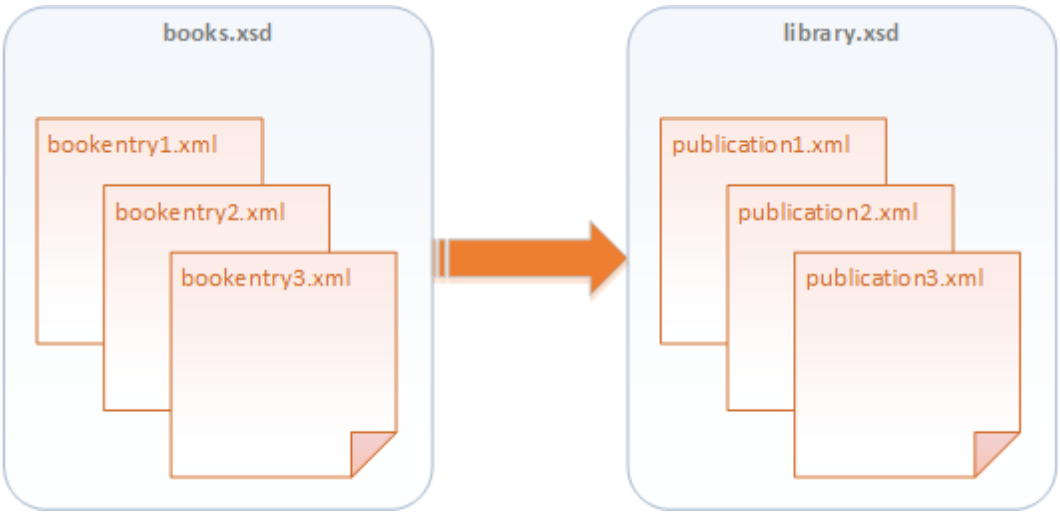


You have now finished designing a mapping which has multiple target components, and you can view and save the output of each target, which was the intended goal of this tutorial. For further information about working with pass-through components, see Chained mappings / pass-through components.

3.4 Process and Generate Files Dynamically

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

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



Abstract model of the data transformation

We will use three source XML files as example. The files are located in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, and they are named **bookentry1.xml**, **bookentry2.xml**, and **bookentry3.xml**. Each of the three files stores a single book.

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

bookentry1.xml

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

bookentry2.xml

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

bookentry3.xml

The source XML files use the **books.xsd** schema available in the following folder: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial**. To convert the source files to a new XML schema, we will use the **library.xsd** schema (available in the same folder). After the transformation, the mapping will generate three files according to this new schema (see the code listings below). We will also configure the mapping so that the name of the generated files will be: **publication1.xml**, **publication2.xml**, and **publication3.xml**. Notice that the XML declaration and the namespace declaration must be stripped.

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

publication1.xml

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

publication2.xml

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

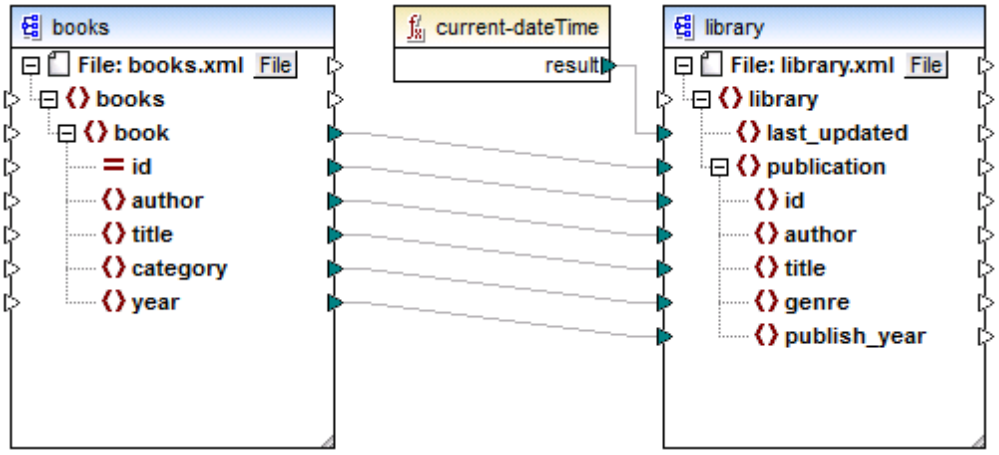
publication3.xml

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

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **BooksToLibrary.mfd** mapping from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. You have already designed this mapping in the Convert XML to New Schema tutorial. To begin, open the **BooksToLibrary.mfd** file in MapForce, and save it with a new name, in the same folder.

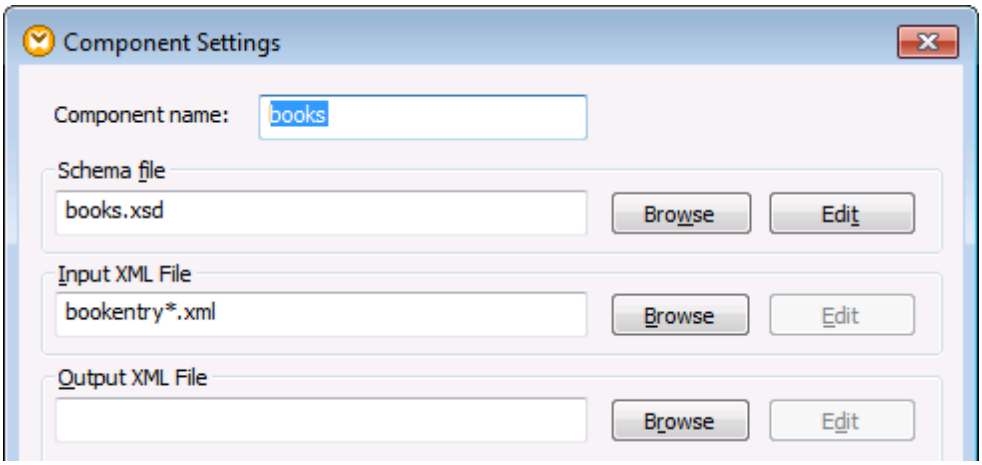
> Make sure to save the new mapping in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, because it references several files from it.



BooksToLibrary.mfd (MapForce Basic Edition)

Step 2: Configure the input

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



Component Settings dialog box

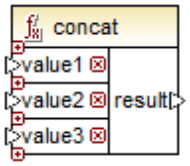
The asterisk (*) wildcard character in the file name instructs MapForce to use as mapping input all the files that have the **bookentry-** prefix. Because the path is a relative one, MapForce will look for all **bookentry-** files in the same directory as the mapping file. Note that you could also enter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

Step 3: Configure the output

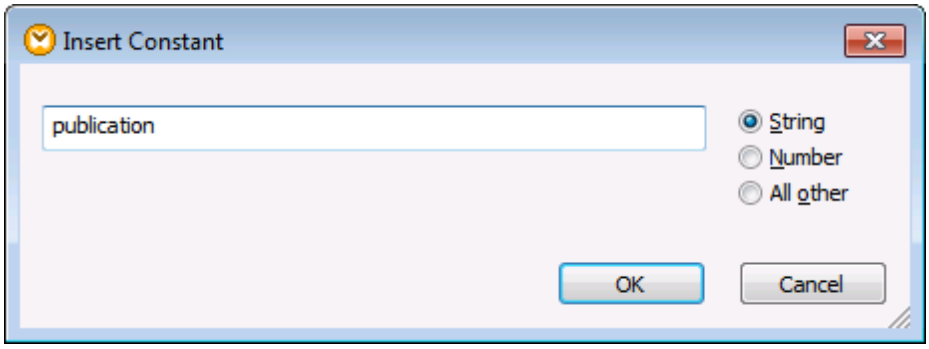
To create the file name of each output file, we will use the **concat** function. This function concatenates (joins) all the values supplied to it as argument.

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

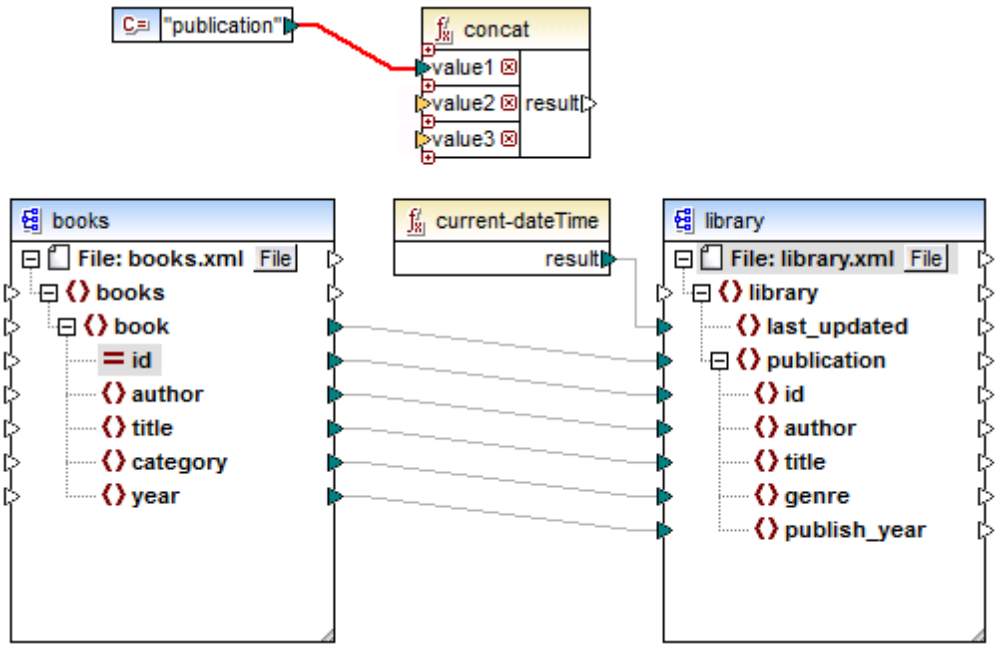
1. Search for the **concat** function in the Libraries window and drag it to the mapping area. By default, this function is added to the mapping with two parameters; however, you can add new parameters if necessary. Click the **Add parameter** (⊡) symbol inside the function component and add a third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



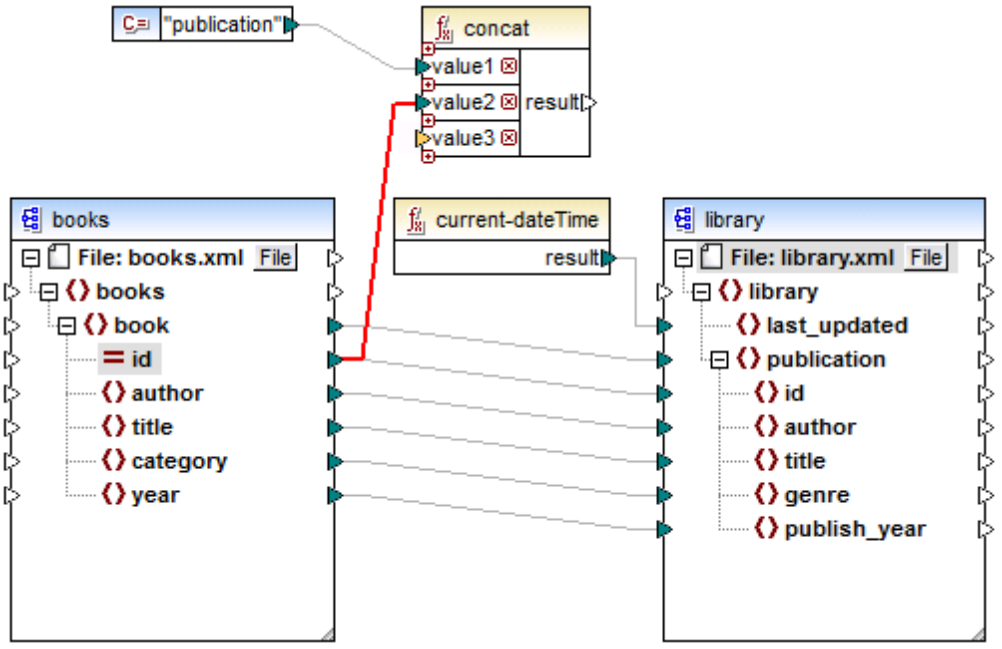
2. Insert a constant (on the **Insert** menu, click **Constant**). When prompted to supply a value, enter "publication" and leave the **String** option unchanged.



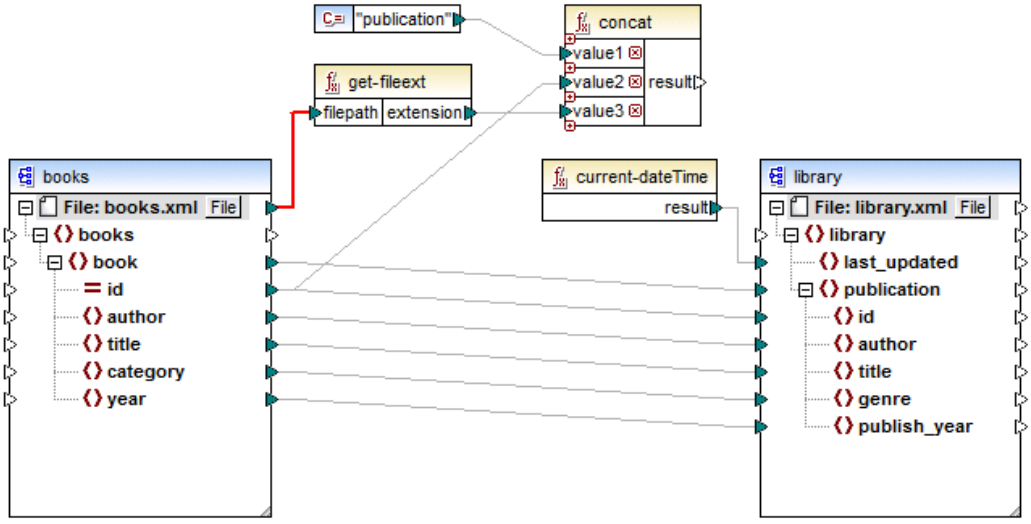
3. Connect the constant with **value1** of the **concat** function.



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



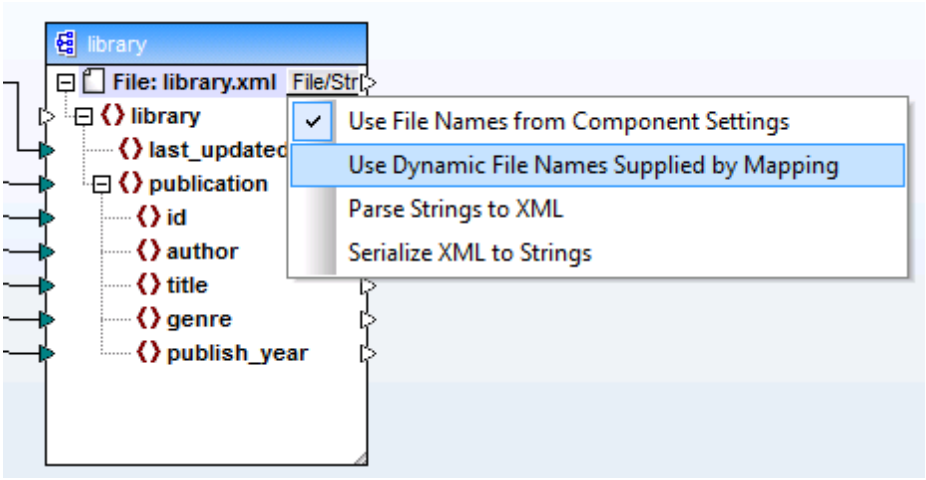
5. Search for the **get-fileext** function in the Libraries window and drag it to the mapping area. Create a connection from the top node of the source component (**File: books.xml**) to the **filepath** parameter of this function. Then create a connection from the result of the **get-fileext** function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



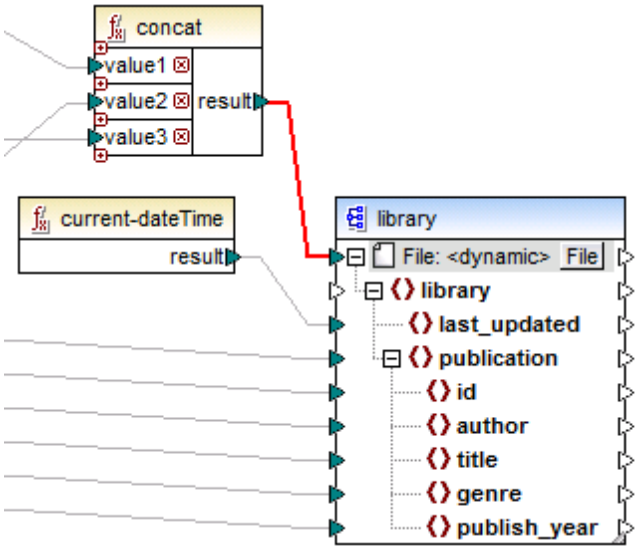
So far, you have provided as parameters to the **concat** function the three values which, when joined together, will create the generated file name (for example, **publication1.xml**):

Part	Example
The constant "publication" supplies the constant string value "publication".	publication
The attribute `id` of the source XML file supplies a unique identifier value for each file. This is to prevent all files from being generated with the same name.	1
The **`get-fileext`** function returns the extension of the file name to be generated.	.xml

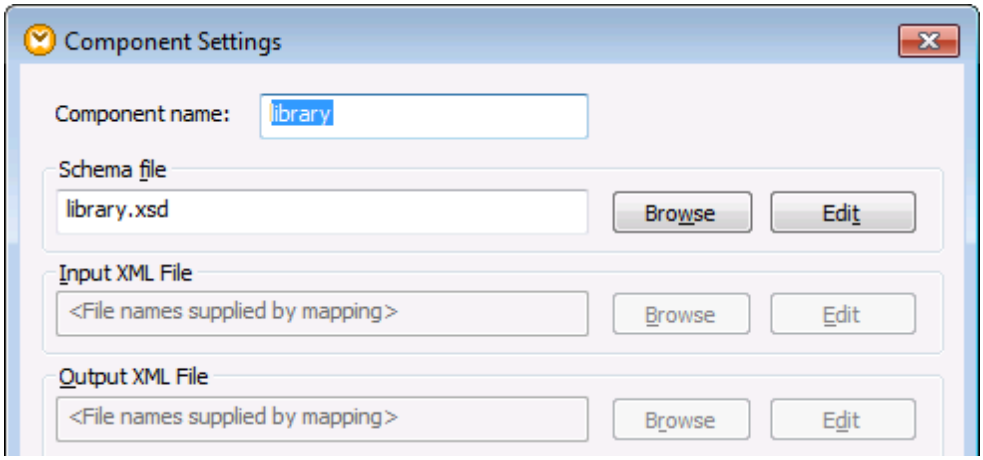
You can now instruct MapForce to actually build the file name when the mapping runs. To do this, click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.



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

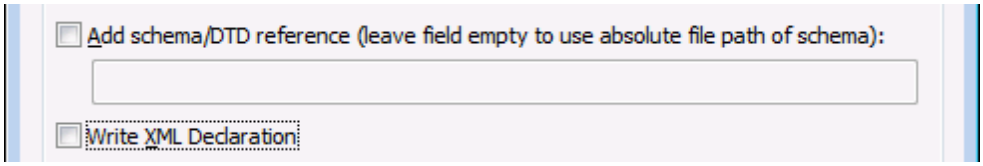


If you double-click the target component header at this time, you will notice that the **Input XML File** and **Output XML File** text boxes are disabled, and their value shows **<File names supplied by the mapping>**.

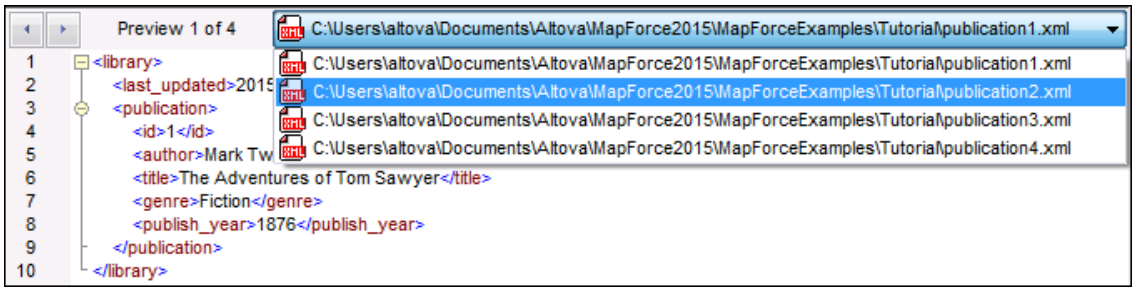


This serves as an indication that you have supplied the instance file names dynamically from a mapping, so it is no longer relevant to define them in the component settings.

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



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



Chapter 4

Common Tasks

4 Common Tasks

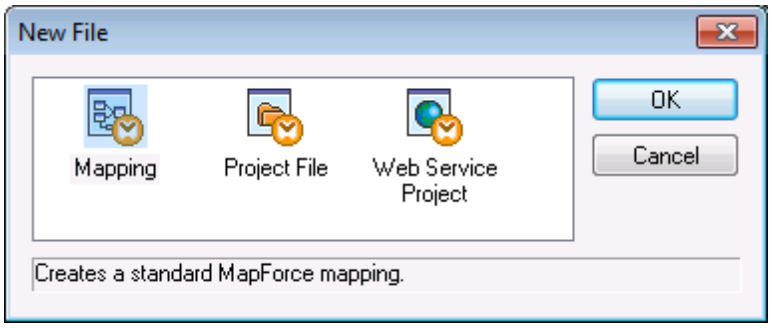
This section describes common MapForce tasks and concepts, such as working with mappings, components, connections, mapping projects, as well as using relative and absolute paths.

4.1 Working with Mappings

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

To create a new mapping:

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



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

4.1.1 Adding Components to the Mapping

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

- Constants
- Databases
- Filters
- Conditions
- Flat files (CSV, fixed-length, and other text files)
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)

- Excel 2007+ files
- Simple input components
- Simple output components
- Variables
- XBRL documents
- XML Schemas and DTDs

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

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

Insert Component toolbar (MapForce Enterprise Edition)

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

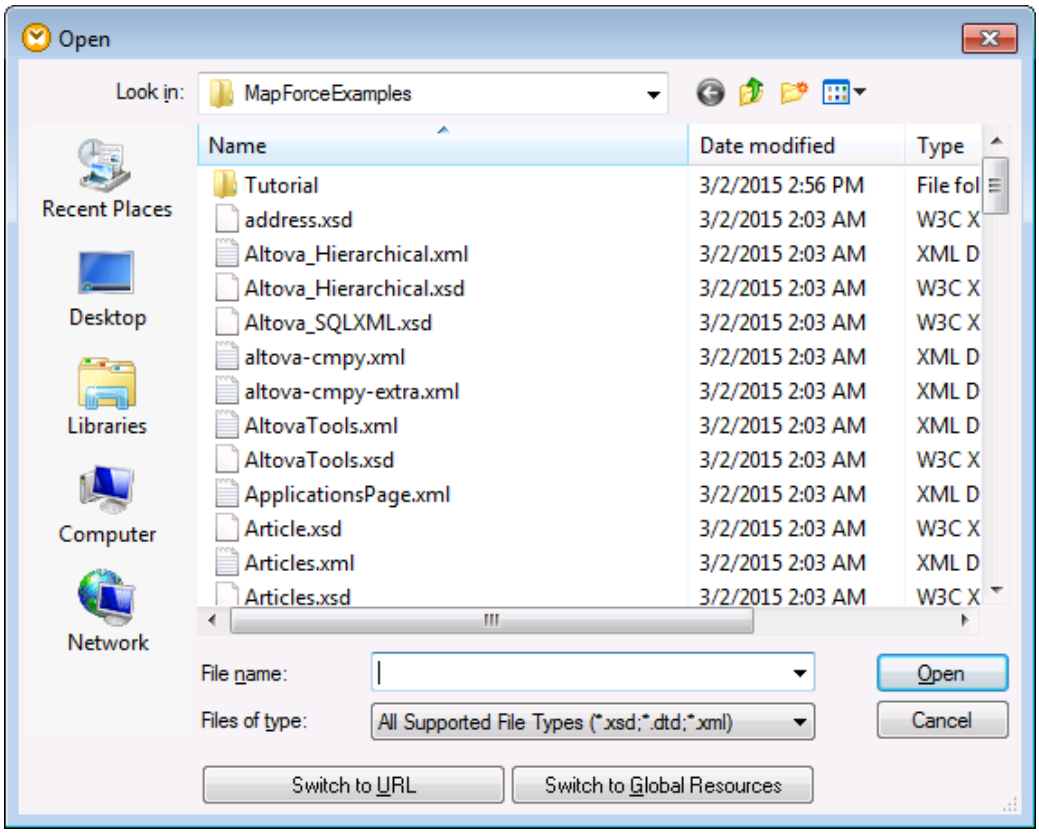
For an introduction to components, see Working with Components. For specific information about each technology supported as mapping source or target, see Data Sources and Targets. For information about MapForce built-in components used to store data temporarily or transform it (such as filtering or sorting),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

4.1.2 Adding Components from a URL

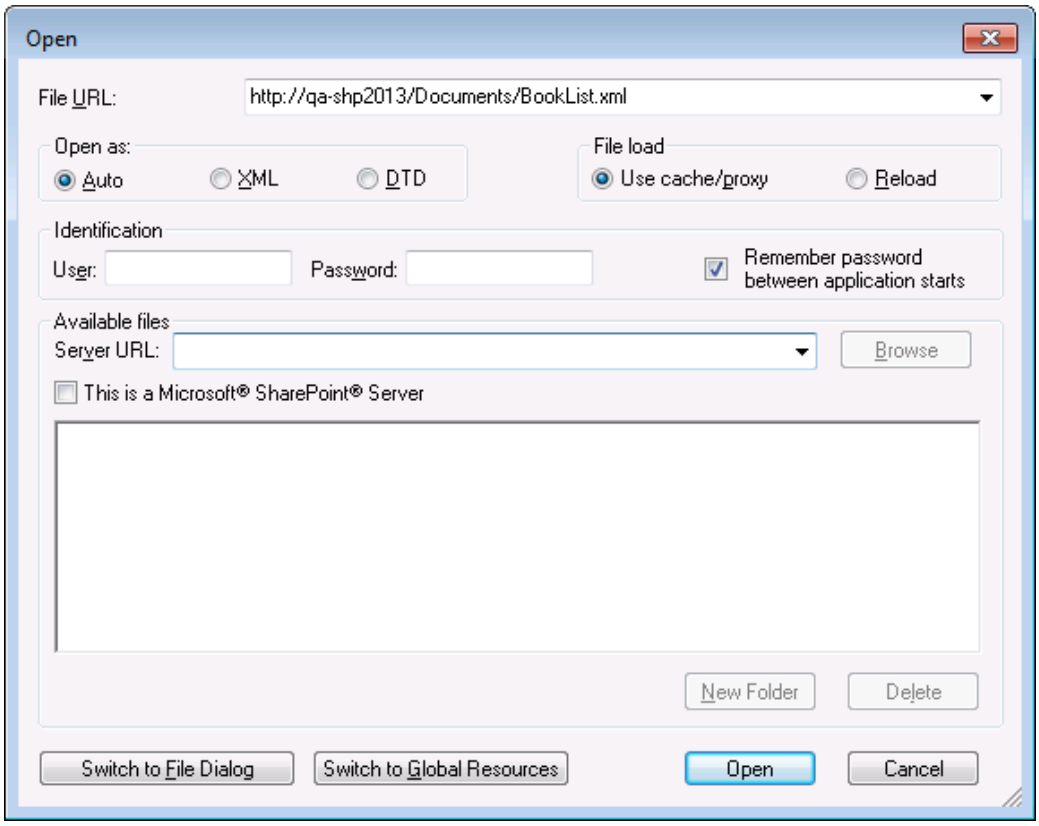
In addition to adding local files as mapping components, you can also add files from a URL. Note that this operation is supported when you add a component as source component (that is, your mapping reads data from the remote file).

To add a component from a URL:

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



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



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

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

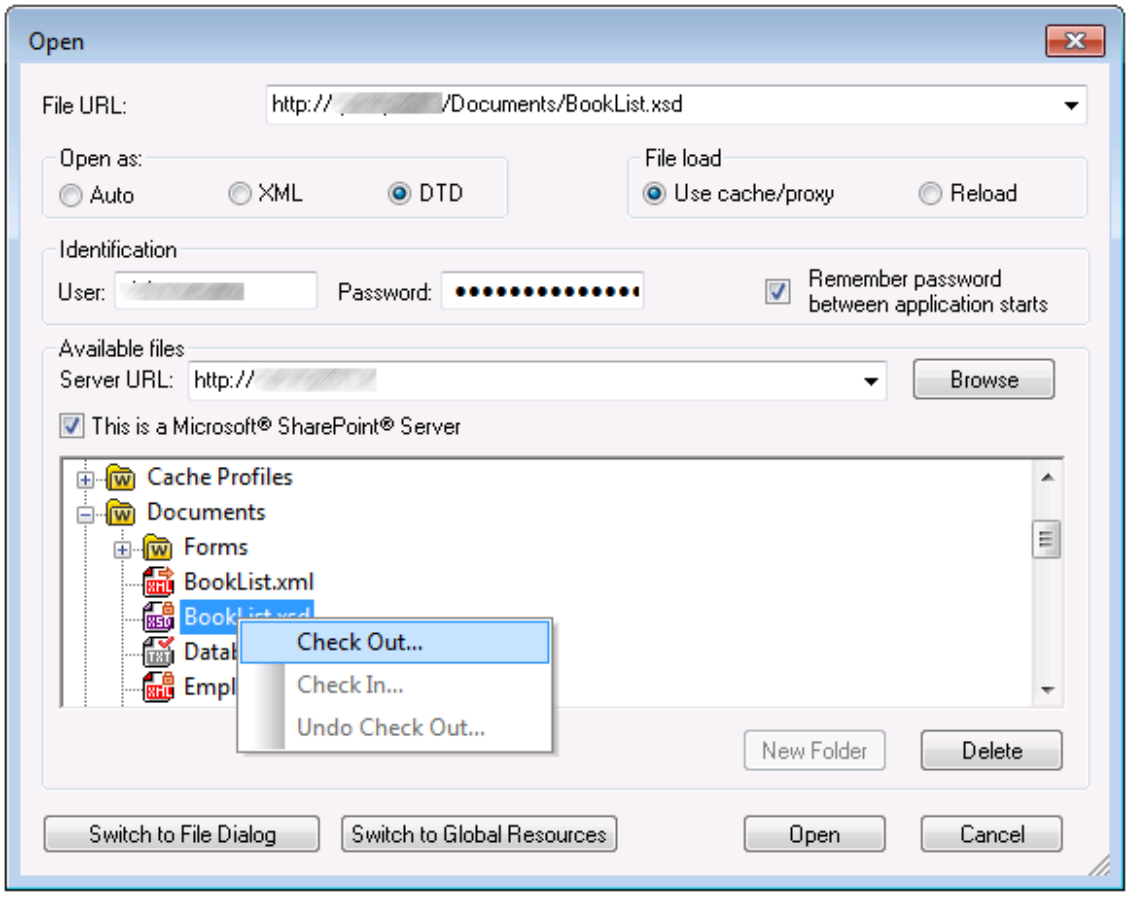
The **Open As** setting defines the grammar for the parser when opening the file. The default and recommended option is **Auto**.

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

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

If the server is a Microsoft SharePoint Server, select the **This is a Microsoft SharePoint Server** check box. Doing so displays the check-in or check-out state of the file in the preview area. If you want to make sure that no one else can edit the file on the server while you are using it in

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



Open dialog box (in Switch to URL mode)

4.1.3 About Data Streaming

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

Data streaming applies to the following data sources:

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

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

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

Memory usage considerations

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

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

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

4.1.4 Selecting a Transformation Language

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

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

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

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

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

- On the **Output** menu, click the name of the language you wish to use for transformation.
- Click the name of the language in the Language Selection toolbar.

Note: Some mapping inputs and outputs are not supported by certain languages. For example,

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

Using the BUILT-IN option

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

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

- As default option, when you do not necessarily need to use a specific language to transform data.
- If you are processing large files and memory usage is a concern.
- If you need to generate digital signatures in the output XML file.

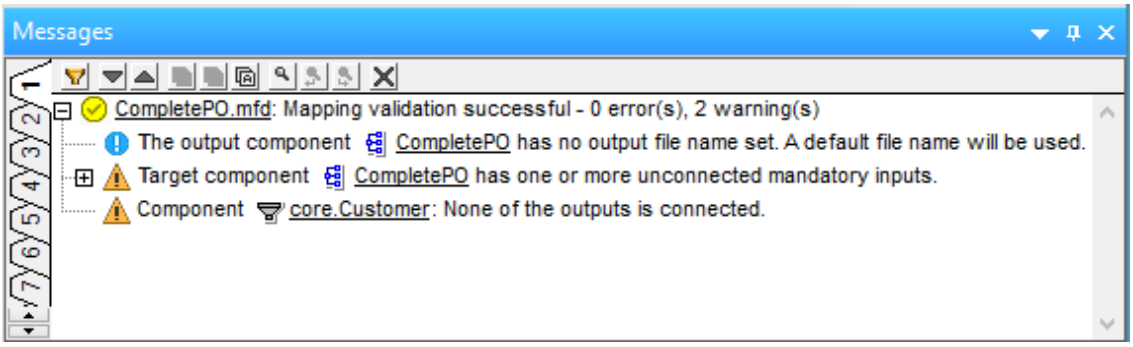
4.1.5 Validating Mappings

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

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

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

The Messages window displays the validation results, for example:



Messages window

When you validate a mapping, MapForce checks for the validity of the mapping (such as incorrect or missing connections, unsupported component kinds), and the validation result is then displayed in the Messages window with one of the following status icons:

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

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

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

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

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

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

To display the result of each validation in an individual tab, click the numbered tabs available on the left side of the Messages window. This may be useful, for example, if you work with multiple mapping files simultaneously

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

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

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

4.1.6 Validating the Mapping Output

After you click the **Output** tab to preview the mapping, the resulting output becomes available in the Output pane. You can validate this output against the schema associated with it. For example, if the mapping transformation generates an XML file, then the resulting XML document can be validated against the XML schema. If the target component is an EDI file, then the output is validated against the EDI specification (see UN/EDIFACT and ANSI X12 as target components). Likewise, if the target is a JSON object, the output is validated against the JSON schema.

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

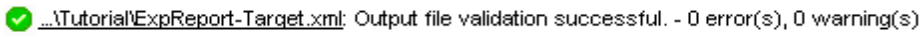
To validate the mapping output, do one of the following:

- Click the **Validate** toolbar button.

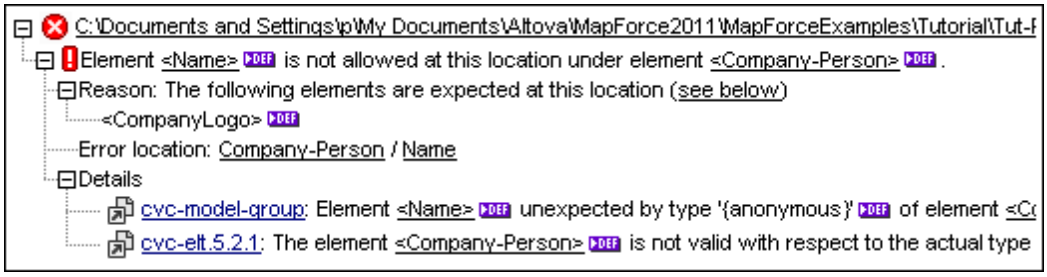
- On the **Output** menu, click **Validate Output File**.

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

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



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



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

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

4.1.7 Previewing the Output

When working with MapForce mappings, you can preview the resulting output without having to run and compile the generated code with an external processor or compiler. In general, it is a good idea to preview the transformation output within MapForce before attempting to process the generated code externally.

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

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

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

To preview the transformation output:

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

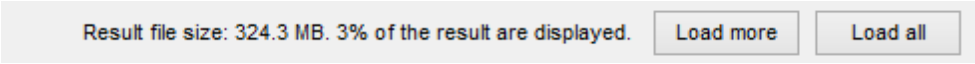
Note: If you select C++, C#, or Java as transformation language, MapForce executes the mapping using its built-in transformation engine. The result that appears in the Output pane is the same as if the Java, C++, or C# code had been generated, compiled and executed.

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

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

Partial output preview

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



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

You can configure the preview settings from the **General** tab of the **Options** dialog box.

4.1.8 Previewing the XSLT Code

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

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

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

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

4.1.9 Generating XSLT Code

To generate XSLT code:

1. Select the menu item **File | Generate code in | XSLT 1.0 (XSLT 2.0)**.
2. Select the folder you want to place the generated XSLT file in, and click **OK**. MapForce generates the code and displays the result of the operation in the Messages window.
3. Navigate to the designated folder and you will find the XSLT file name in the form: **MappingMapTo<TargetSchemaName>.xslt**.

The folder in which the XSLT file is placed also contains a batch file called **DoTransform.bat** which can be run with RaptorXML Server to transform the data.

To run the transformation with RaptorXML Server:

1. Download and install RaptorXML from the download page.
2. Start the **DoTransform.bat** batch file located in the previously designated output folder.

Note that you might need to add the RaptorXML installation location to the **path** variable of the Environment Variables. You can find the RaptorXML documentation on the website documentation

page.

4.1.10 Previewing the XQuery Code

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

To preview the generated XQuery code:

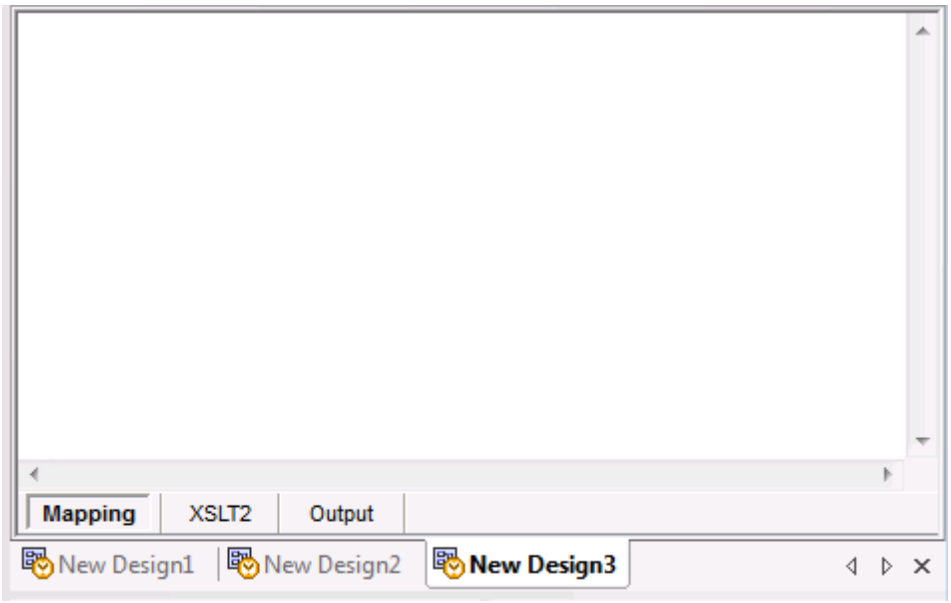
- Click the **XQuery** tab under the Mapping window.

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

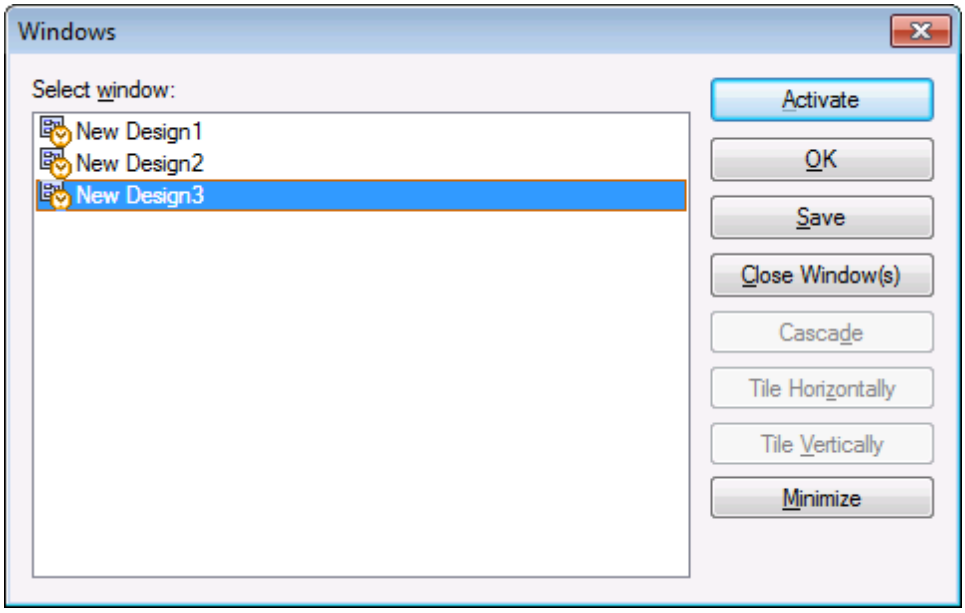
4.1.11 Working with Multiple Mapping Windows

MapForce uses a Multiple Document Interface (MDI). Each mapping file you open in MapForce has a separate window. This enables you to work with multiple mapping windows and arrange or resize them in various ways inside the main (parent) MapForce window. You can also arrange all open windows using the standard Windows layouts: Tile horizontally, Tile vertically, Cascade.

When multiple mappings are open in MapForce, you can quickly switch between them using the tabs displayed in the lower part of the Mapping pane.



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



Windows dialog box

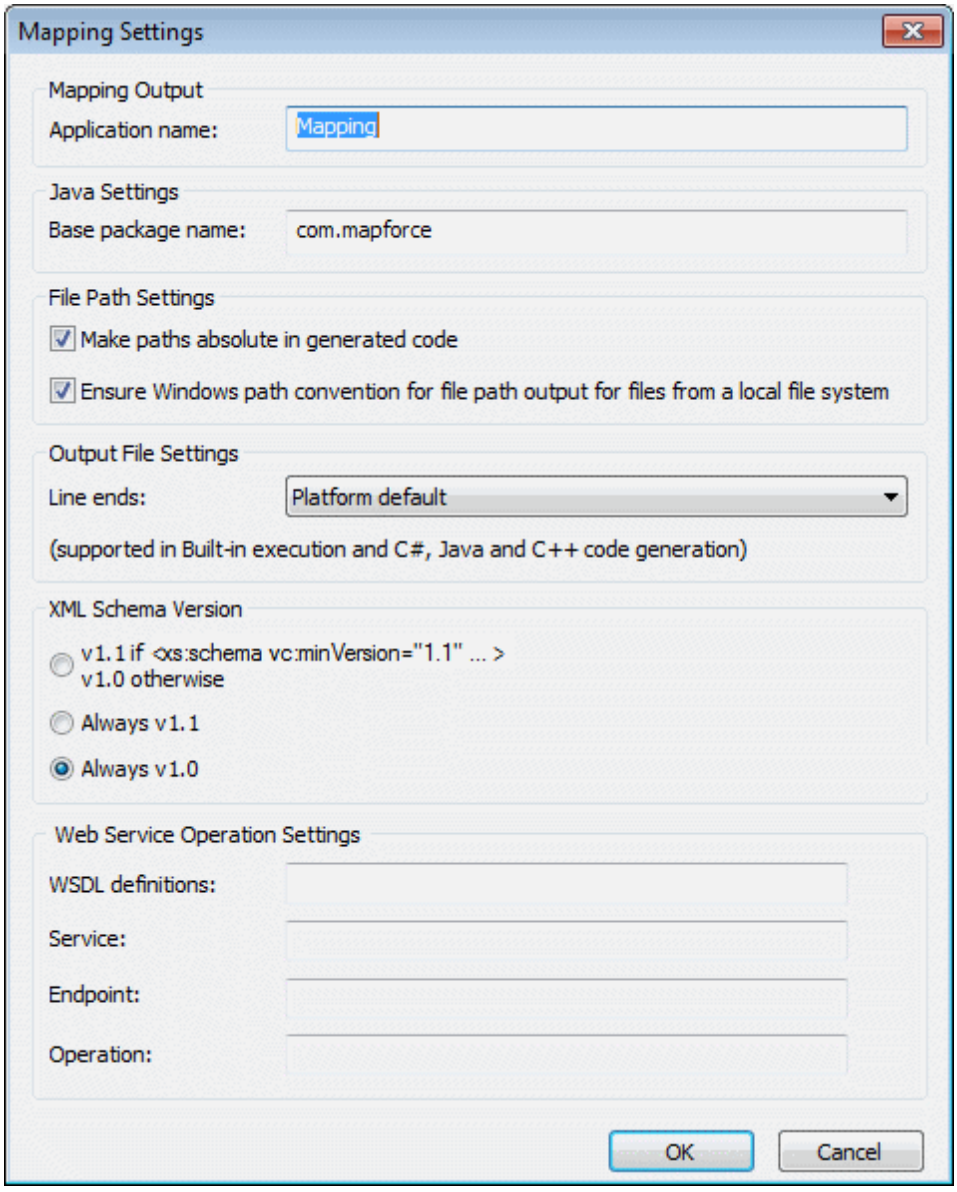
You can open the Windows dialog box using the menu command **Window | Windows...** To select multiple windows in the Windows dialog box, click the required entries while holding the **Ctrl** key pressed.

4.1.12 Changing the Mapping Settings

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

To open the Mapping Settings dialog box:

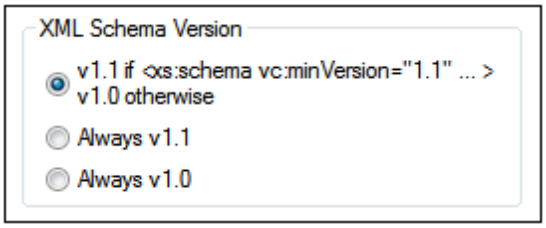
- On the **File** menu, click **Mapping Settings**.



Mapping Settings dialog box

The available settings are as follows.

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

Ensure Windows path convention for file path	The "Ensure Windows path convention...." check box makes sure that Windows path conventions are followed. When outputting XSLT2 (and XQuery), the currently processed file name is internally retrieved using the document-uri function, which returns a path in the form file:// URI for local files. When this check box is active, a file:// URI path specification is automatically converted to a complete Windows file path (e.g. "C:\...") to simplify further processing.
Line ends	This combo box allows you to specify the line endings of the output files. "Platform default" is the specific default for the target operating system, e.g. Windows (CR+LF), Mac OS X (LF), or Linux (LF). You can also select a specific line ending manually. The settings you select here are crucial when you deploy a mapping to FlowForce Server running on a different operating system.
XML Schema Version	Lets you define the XML Schema Version used in the mapping file. You can define if you always want to **load** the Schemas conforming to version 1.0 or 1.1. Note that not all version 1.1 specific features are currently supported. If the *xs:schema vc:minVersion="1.1"* declaration is present, then version 1.1 will be used; if not, version 1.0 will be used.  If the XSD document has no `vc:minVersion` attribute or the value of the `vc:minVersion` attribute is other than `1.0` or `1.1`, then XSD 1.0 will be the default mode. **Note:** Do not confuse the `vc:minVersion` attribute with the `xsd:version` attribute. The former holds the XSD version number, while the latter holds the document version number. Changing this setting in an existing mapping causes a reloading of all schemas of the selected XML schema version, and might also change its validity.
Web Service Operation Settings	The WSDL-Definitions, Service, Port and Operation fields are automatically filled if the mapping document is part of a Web service implementation.

4.2 Working with Components

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

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

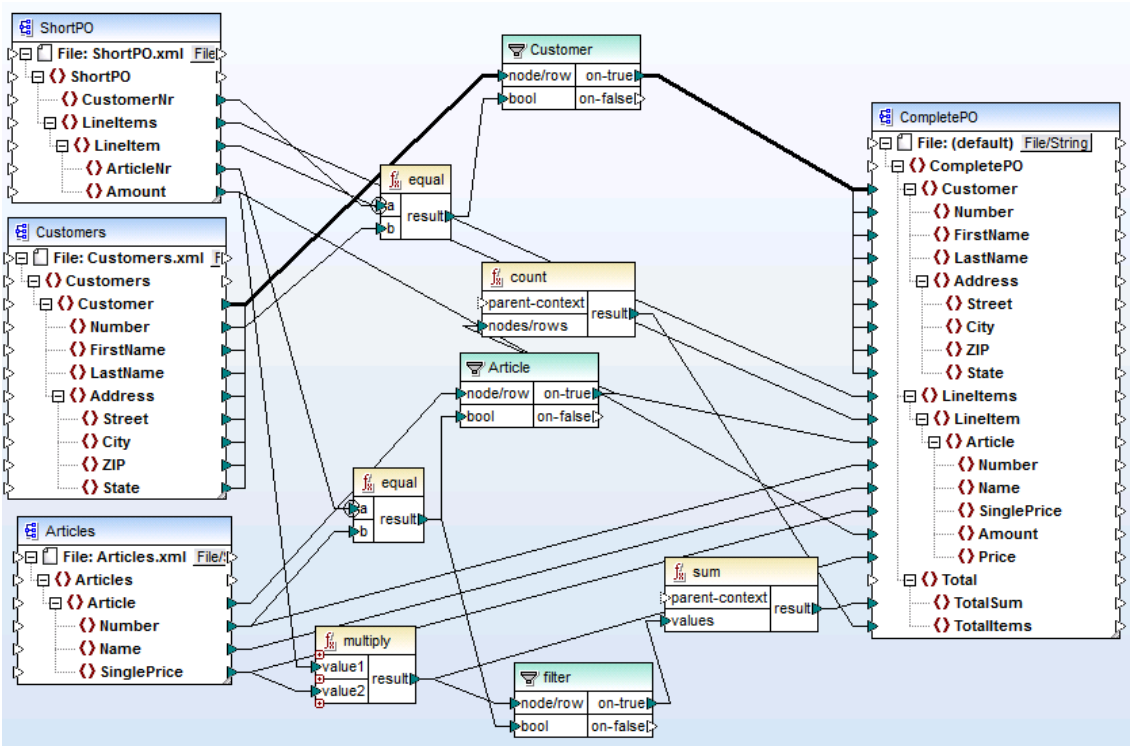
The structure components represent the abstract structure or schema of your data. For example, when you add an XML file to the mapping area (using the menu command **Insert | XML Schema/ File**), it becomes a mapping component. For further information about structure components and their specifics, see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

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

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

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

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



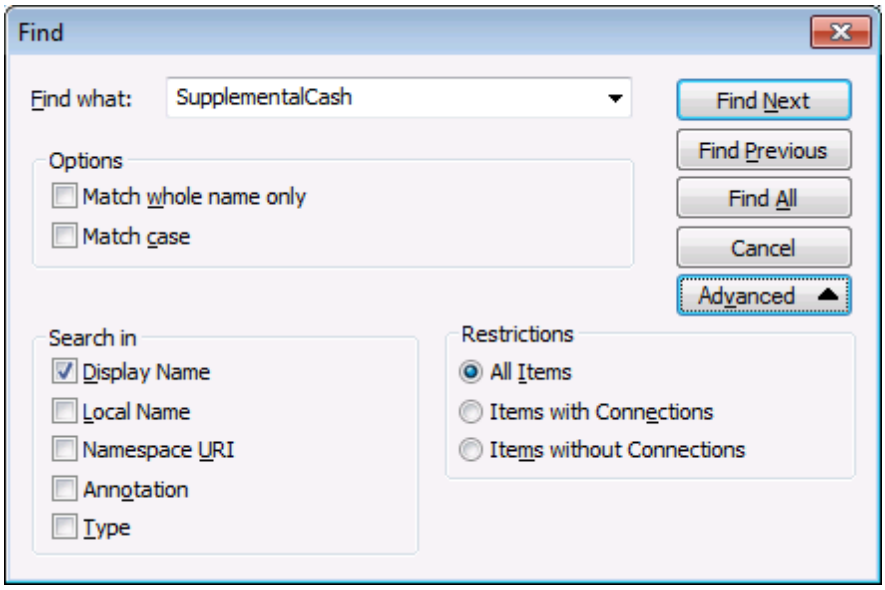
CompletePO.mfd

This mapping sample is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\CompletePO.mfd**.

4.2.1 Searching within Components

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

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

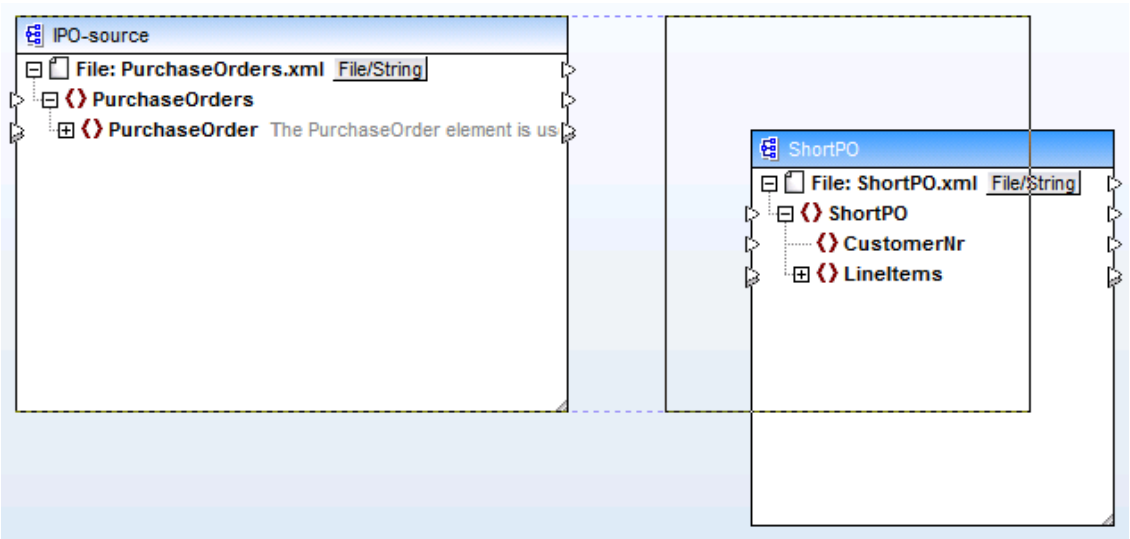


Use the Advanced options to define which items (nodes) are to be searched, as well as restrict the search options based on the specific connections.

4.2.2 Aligning Components

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

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



Component auto-alignment guide lines

To enable or disable this option:

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

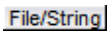
4.2.3 Changing the Component Settings

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

- Select the component and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

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

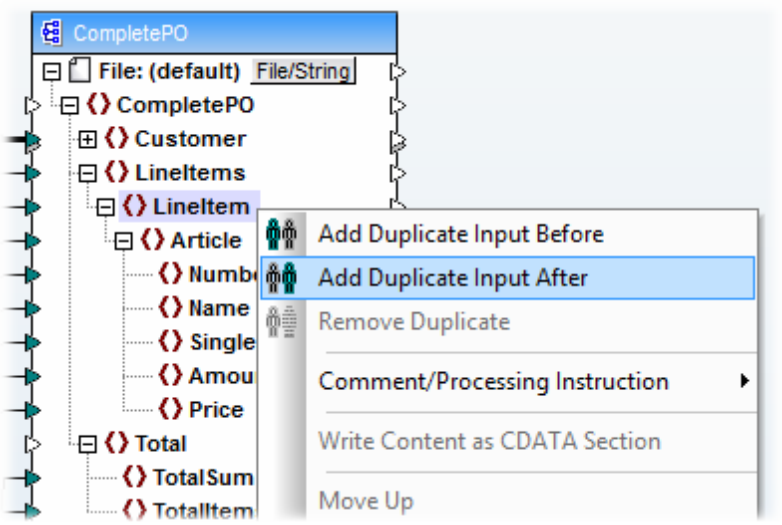
- XML Component Settings
- Database Component Settings
- CSV Component Settings
- Fixed-Length Field Component Settings
- FlexText Component Settings
- JSON Component Settings
- Excel 2007+ Component Settings
- XBRL Component Settings
- EDI Component Settings

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

4.2.4 Duplicating Input

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

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



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

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

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

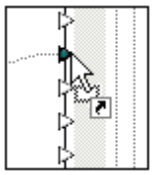
4.3 Working with Connections

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

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

To draw a connection between two items:

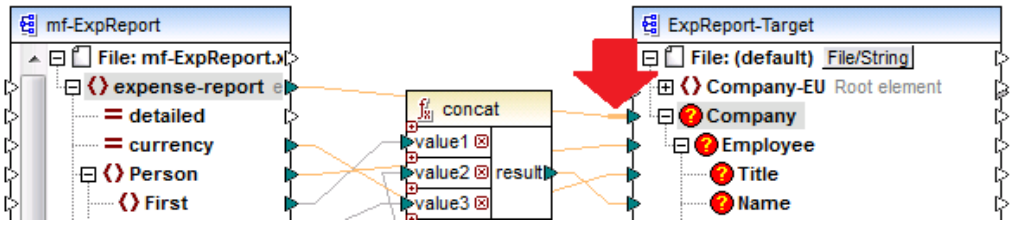
- Click the output connector of a source item and drag it to a destination item. When the drop action is allowed, a link tooltip appears next to the text cursor.



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

To move a connection to a different item:

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

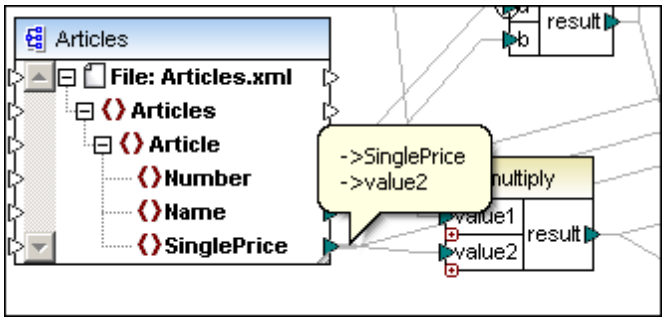


To copy a connection to a different item:

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

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

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



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

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

See also Connection Settings.

To delete a connection, do one of the following:

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

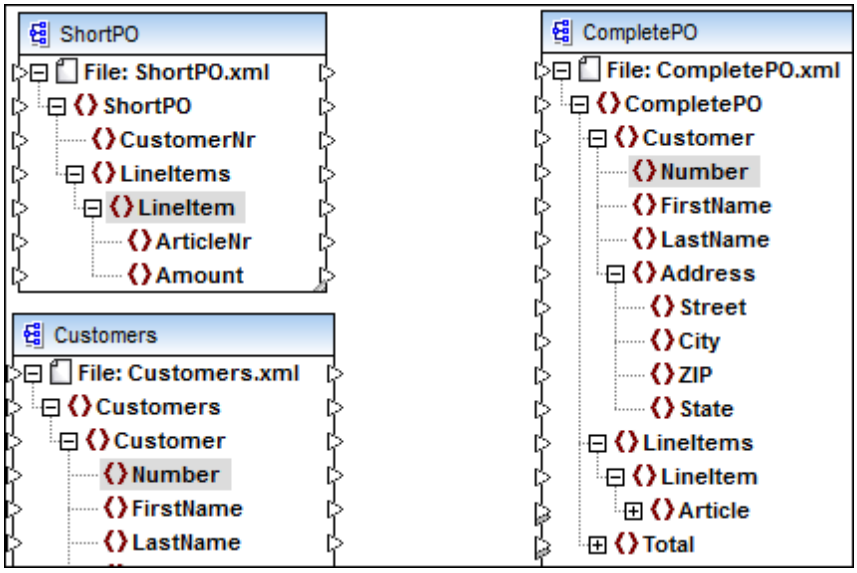
4.3.1 About Mandatory Inputs

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

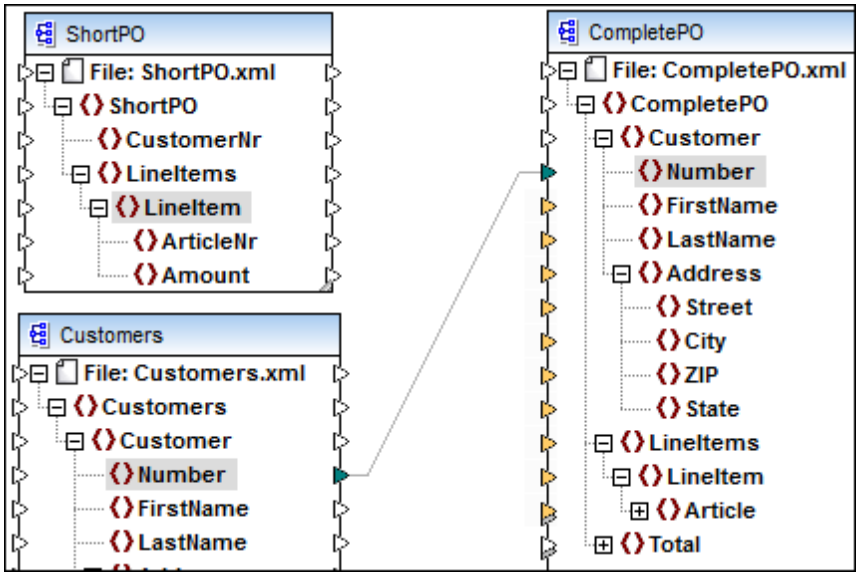
- In XML and EDI components these are items where the minOccurs parameter is equal/ greater than 1.
- In databases these are fields that have been defined as "not null"
- WSDL calls and WSDL response (all nodes)
- XBRL nodes that have been defined as mandatory
- In functions these are the specific mandatory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

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

inserted XML Schema files exist as shown below.



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



4.3.2 Changing the Connection Display Preferences

You can selectively view the connections in the mapping window.

Show selected component connectors switches between showing:

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

Show connectors from source to target switches between showing:

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

4.3.3 Annotating Connections

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

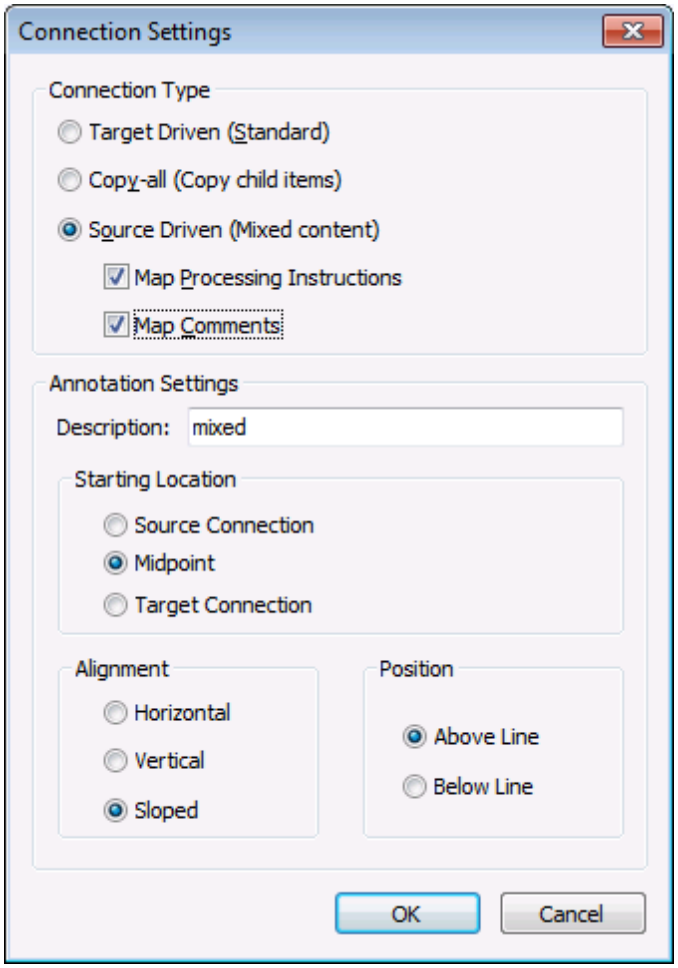
To annotate to a connection:

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

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

4.3.4 Connection Settings

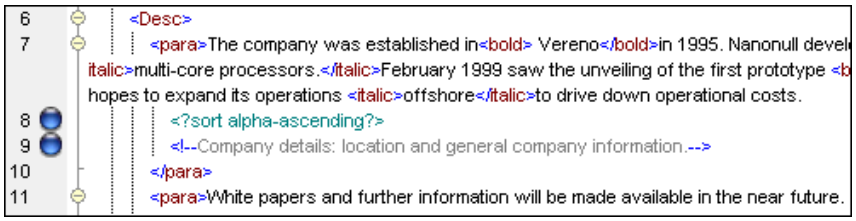
Right-clicking a connection and selecting **Properties** from the context menu, or double-clicking a connection, opens the Connection Settings dialog box in which you can define the settings of the current connection. Note that unavailable options are disabled.



Connection Settings dialog box

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

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

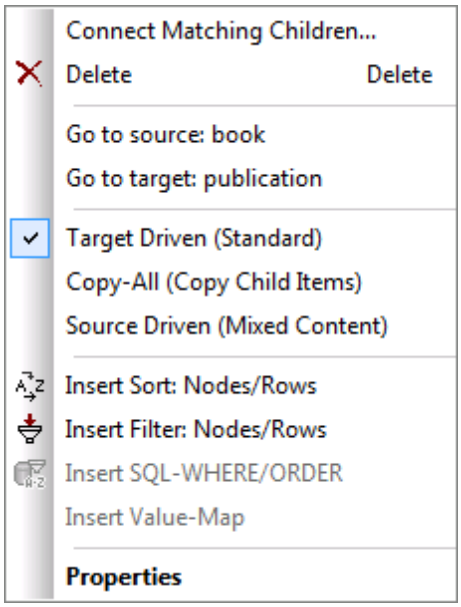


Note: CDATA sections are treated as text.

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

4.3.5 Connection Context Menu

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



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

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

4.3.6 Connecting Matching Children

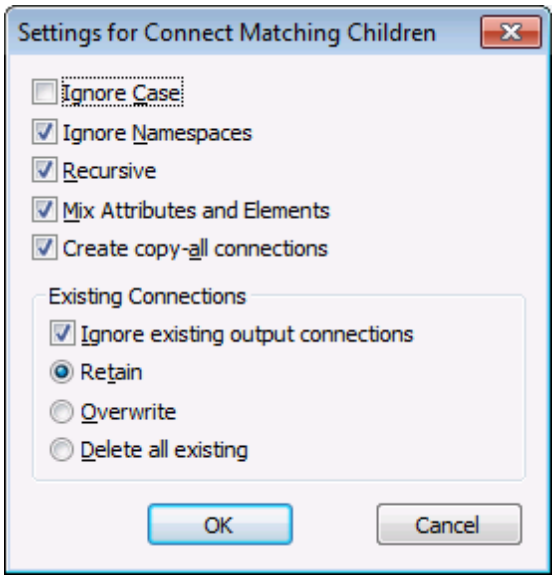
You can create multiple connections between items of the **same name** in both the source and target components. Note that a "Copy-all" connection (see Copy-all connections) is created by default.

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

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

To change the settings for "Connect Matching Children":

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



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

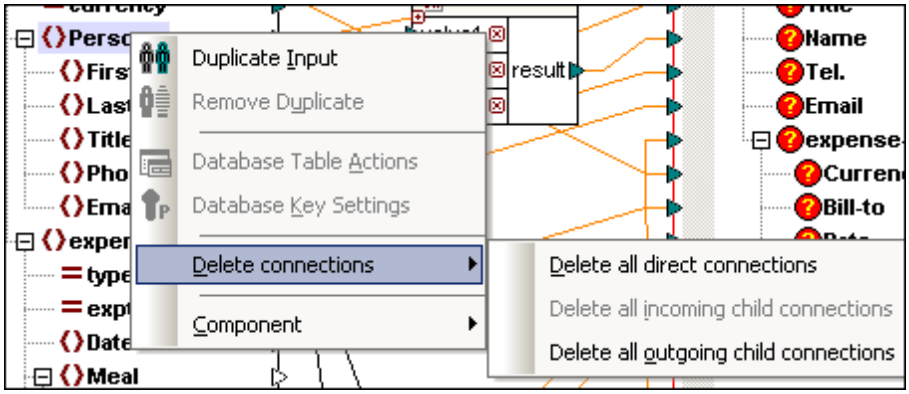
Note: The settings you define here are applied when connecting two items if the **Toggle auto connect of children** ( ) toolbar button is active.

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

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

Deleting connections

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



To delete connections:

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

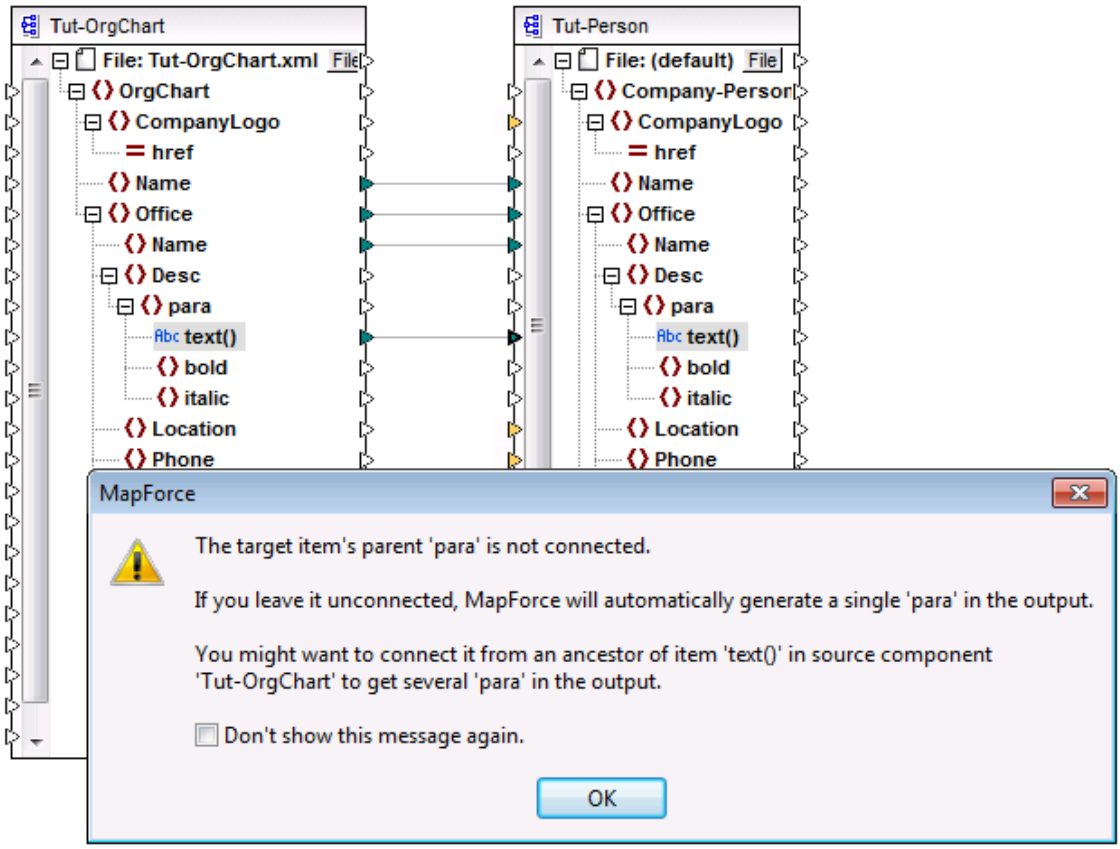
Delete all direct connections	Deletes all connections directly mapped to, or from, the current component to any other source or target components.
Delete all incoming child connections	Only active if you have right clicked an item in a target component. Deletes all incoming child connections.
Delete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

4.3.7 Notifications on Missing Parent Connections

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

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

To understand how this works, open the sample mapping **Tut-OrgChart.mfd** available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. If you connect the source `text()` item to the target `text()` item, a notification message appears, stating that the parent item "para" is not connected and will only be generated once in the output.



Tut-OrgChart.mfd (MapForce Basic Edition)

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

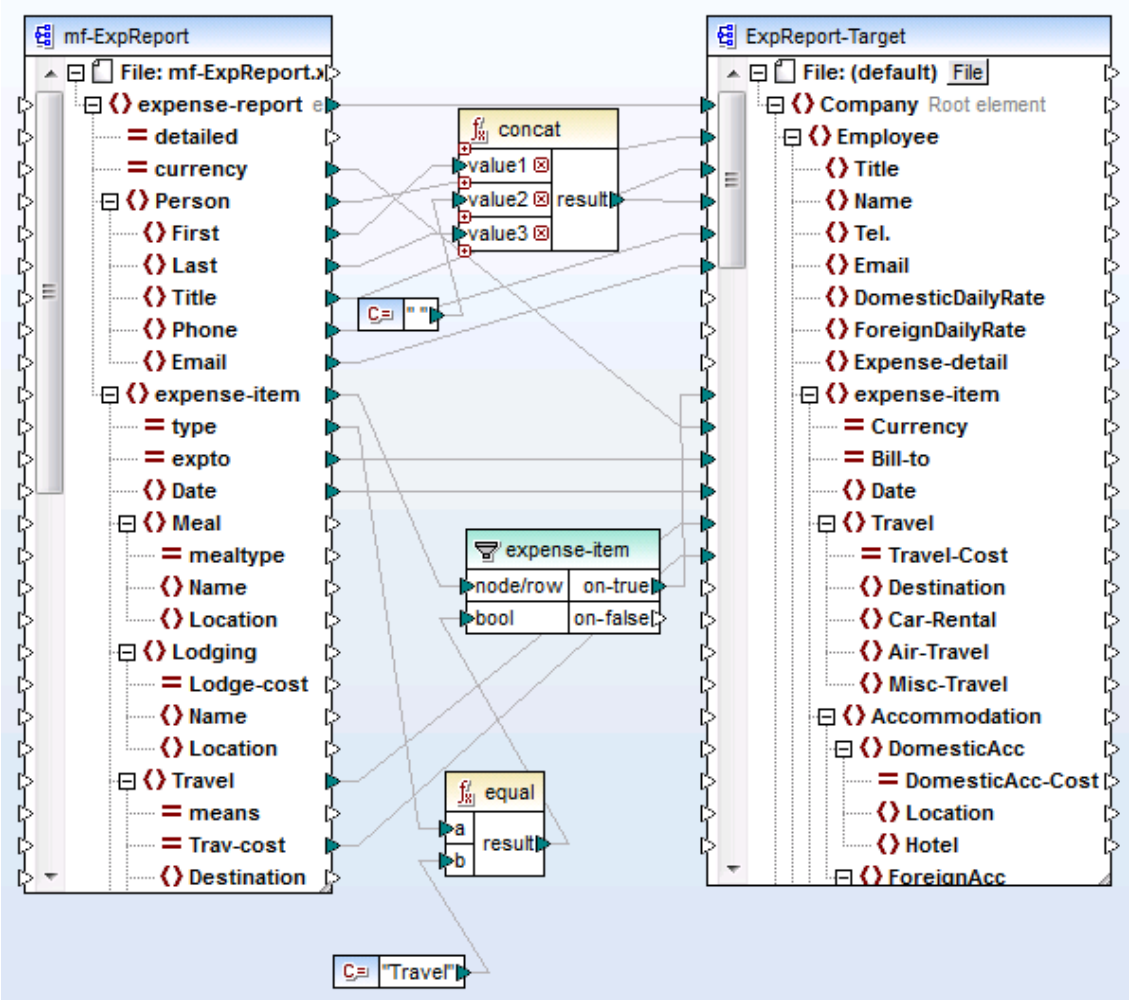
To disable such notifications, do the following:

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

4.3.8 Moving Connections and Child Connections

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

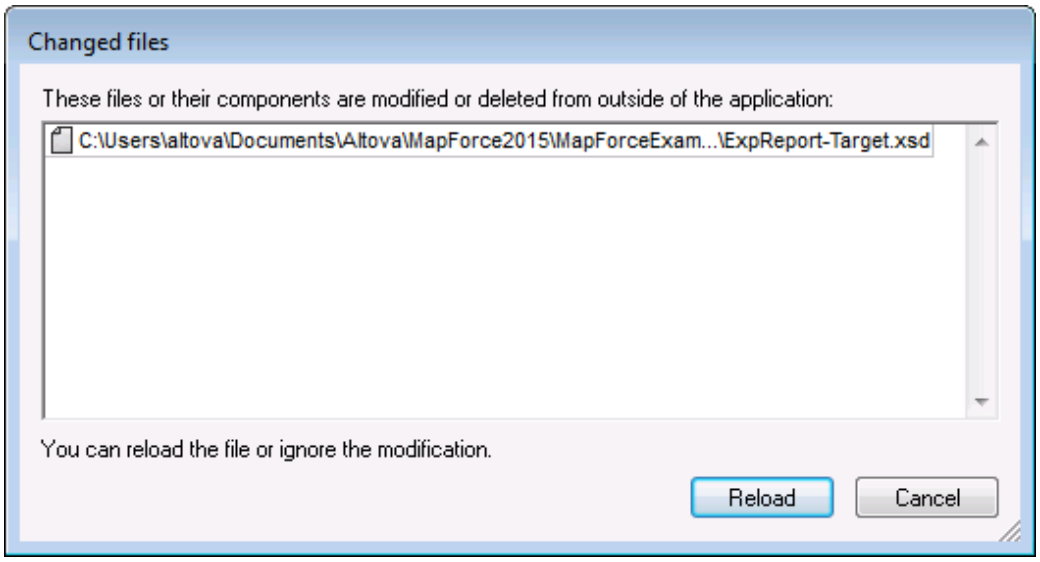
This example uses the **Tut-ExpReport.mfd** file available in the **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial** folder.



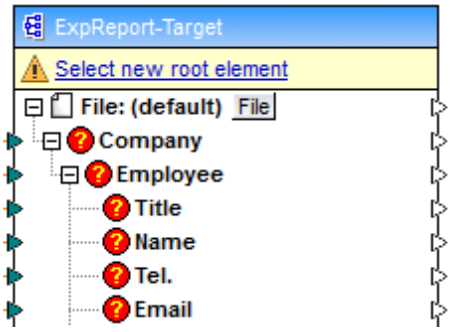
Tut-ExpReport.mfd (MapForce Basic Edition)

To understand how it works, do the following:

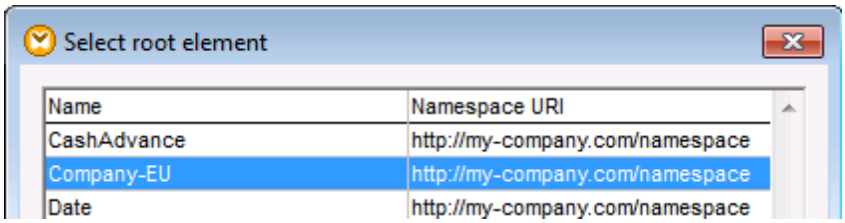
1. Open the **Tut-ExpReport.mfd** sample mapping.
2. Edit the **ExpReport-Target.xsd** schema outside MapForce so as to change the `Company` root element of the target schema to `Company-EU`. You do not need to close MapForce.
3. After you have changed the `Company` root element of the target schema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



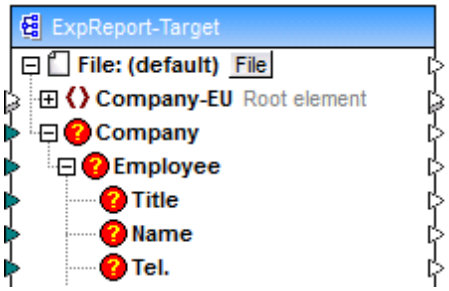
4. Click the **Reload** button to reload the updated Schema. Since the root element was deleted, the component displays multiple missing nodes.



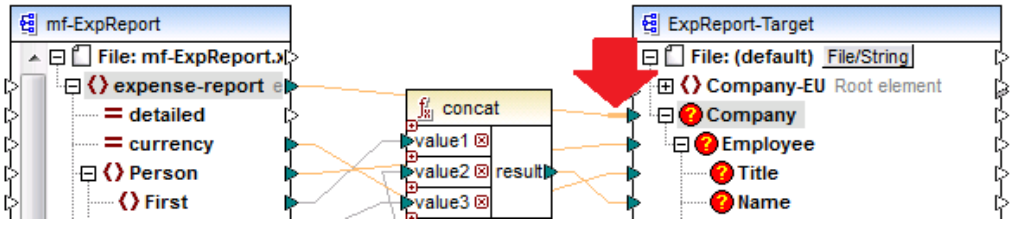
5. Click **Select new root element** at the top of the component. (You can also change the root element by right clicking the component header and selecting **Change Root Element** from the context menu.)



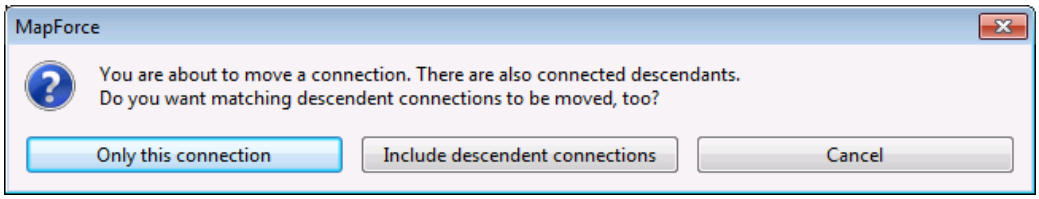
6. Select `Company-EU` as new root element and click OK to confirm. The `Company-EU` root element is now visible at the top of the component.



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



A notification dialog box appears.



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

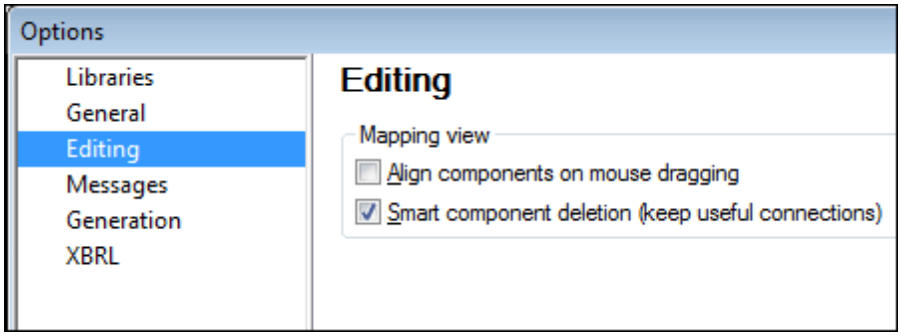
Note: If the node to which you are mapping has the same name as the source node but is in a different namespace, then the notification dialog box will contain an additional button: "Include descendants and map namespace". Clicking this button moves the child connections of the same namespace as the source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

4.3.9 Keeping Connections After Deleting Components

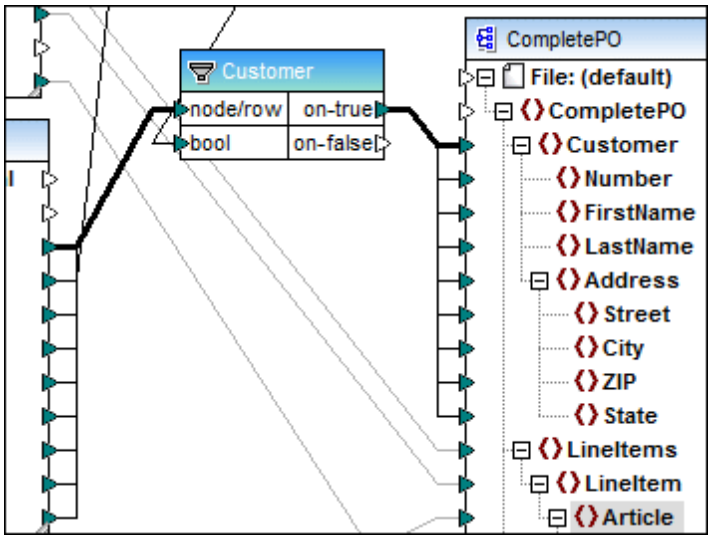
You can decide what happens when you delete a component that has multiple (child) connections to another component, e.g. a filter or sort component. This is very useful if you want to keep all the child connections and not have to restore each one individually.

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

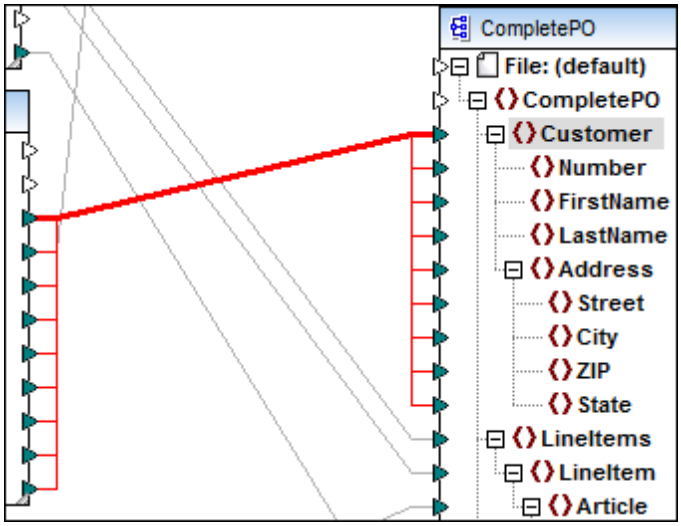
Select **Tools | Options | Editing** (tab) to see the current setting. The default setting for the check box is **inactive**, i.e. "Smart component deletion (keep useful connections)" is disabled.



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



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



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

Clicking anywhere in the mapping area deselects the connectors.

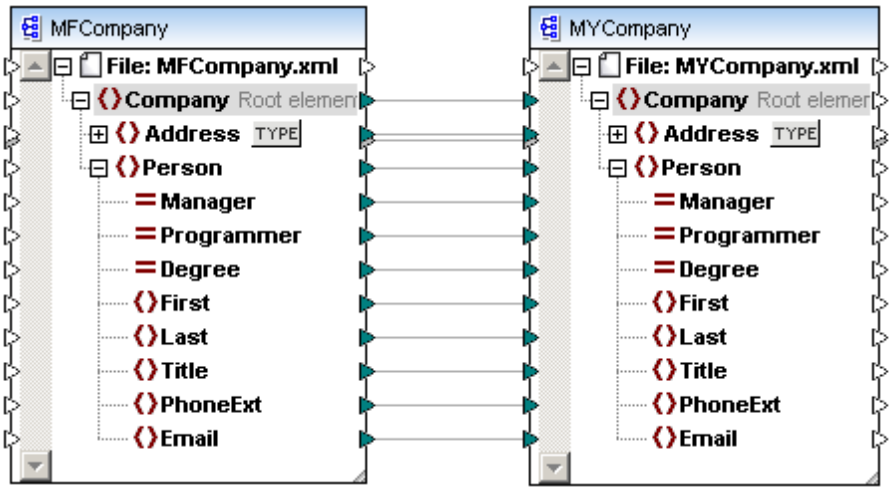
If the "Smart component deletion..." check box is **inactive**, then deleting the filter will delete all child connectors immediately.

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

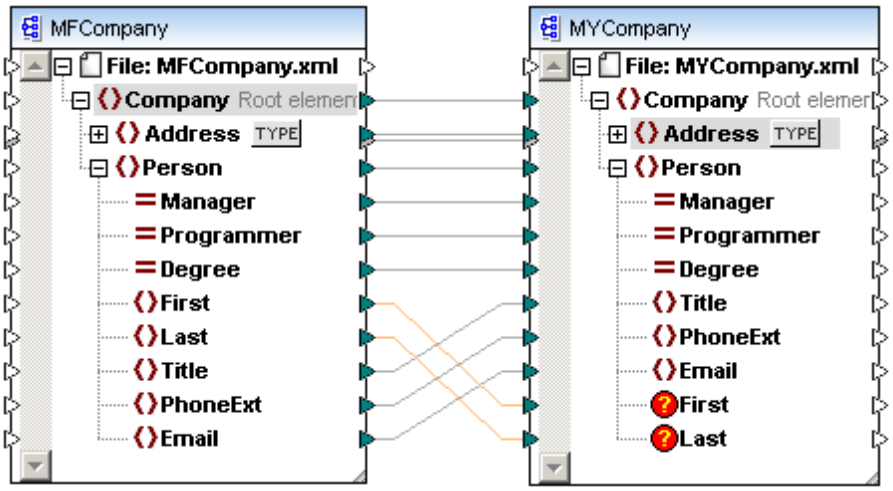
4.3.10 Dealing with Missing Items

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

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



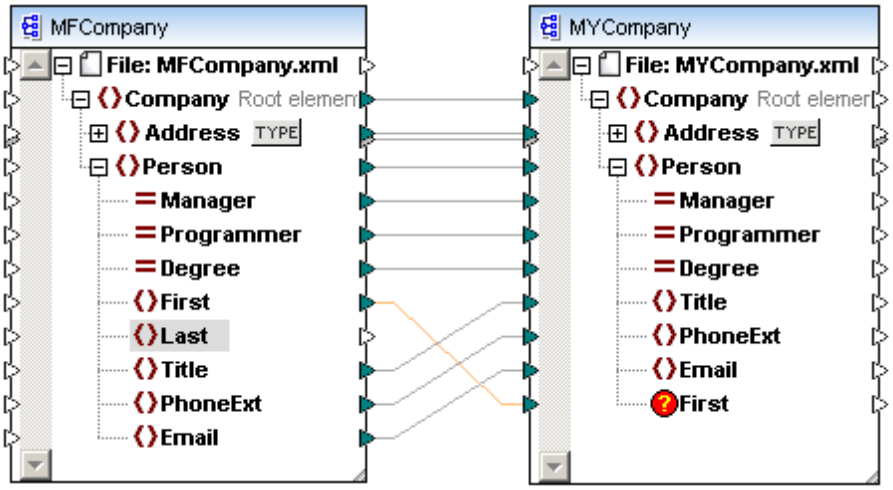
While editing MyCompany.xsd, in XMLSpy, the First and Last items in the schema are deleted. Returning to MapForce opens a Changed Files notification dialog box, prompting you to reload the schema. Clicking **Reload** updates the components in MapForce.



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

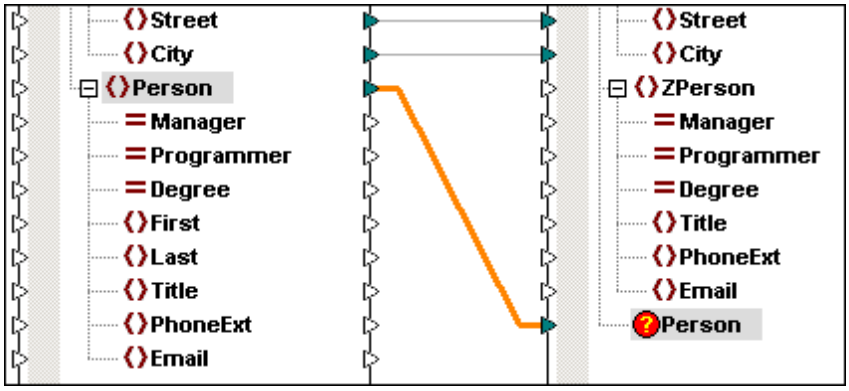
Note that you can still preview the mapping (or generate code), but warnings will appear in the Messages window if you do so at this point. All connections to, and from, missing items are ignored during preview or code-generation.

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



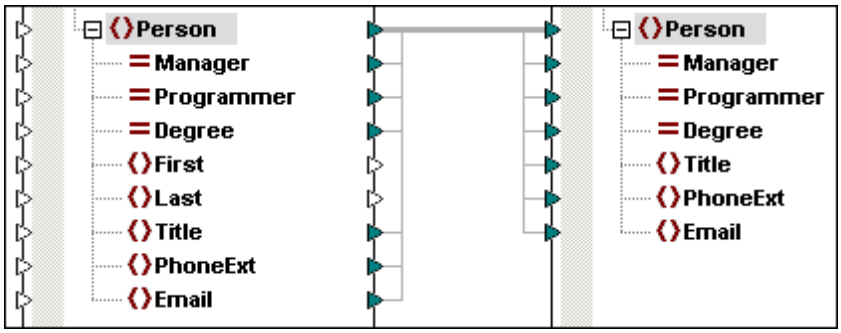
Renamed items

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



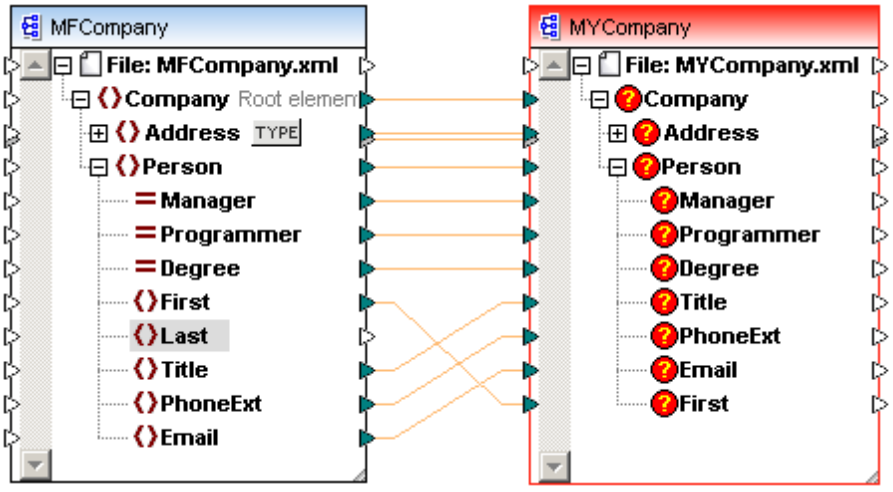
"Copy all" connectors and missing items

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

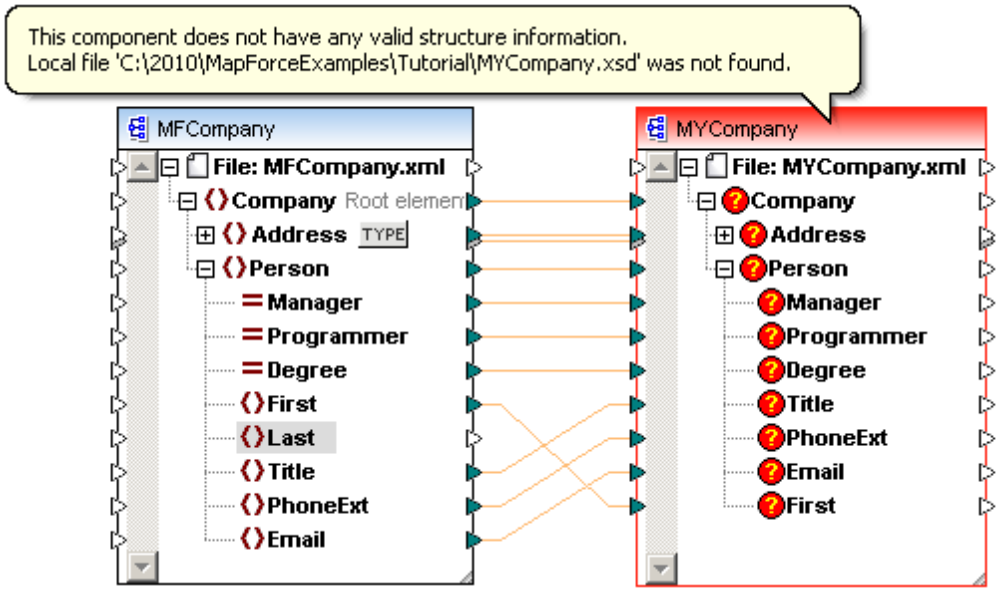


Renamed or deleted component sources

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

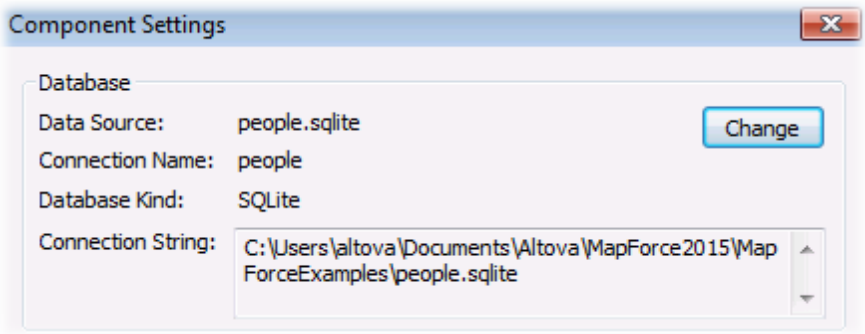


Placing the mouse cursor over the highlighted component, opens a popup containing pertinent information.



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

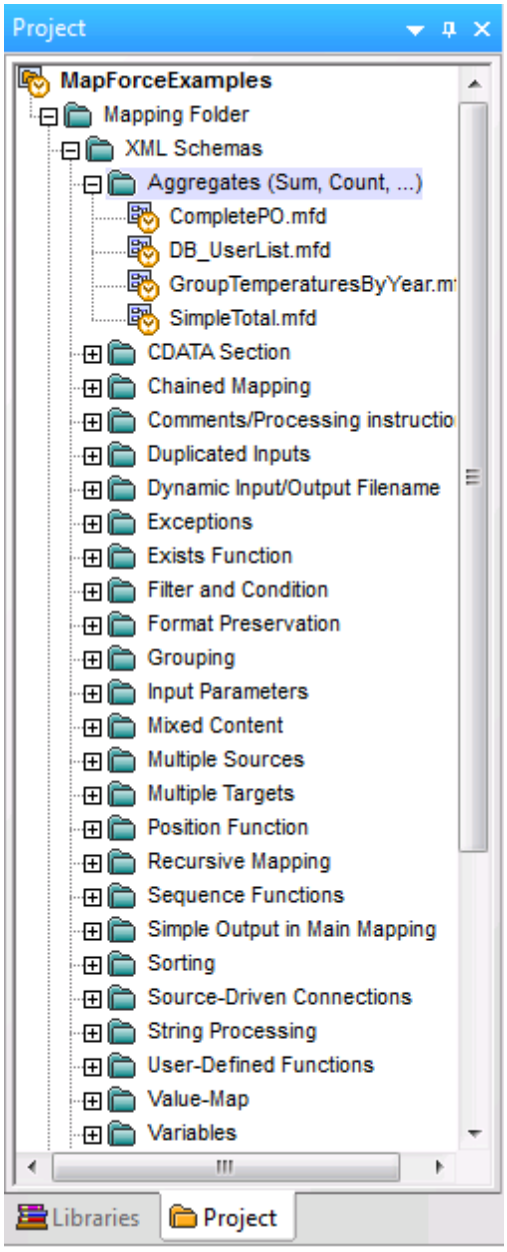
Clicking the **Change** button in the dialog box that opens if the component is a database, allows you to select a different database, or change the tables that appear in the database component. Connectors to tables of the same name will be retained.



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

4.4 Working with Mapping Projects

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



Project window (MapForce Enterprise Edition)

The main advantage of projects is that you can define common code generation settings (such as the target language and the output directory) for all the mapping files included in that particular project. You can also create folders inside projects, and specify custom code generation settings for each individual folder in a project. For more information about the MapForce-generated program

code (in C++, C#, and Java), see Code Generator.

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

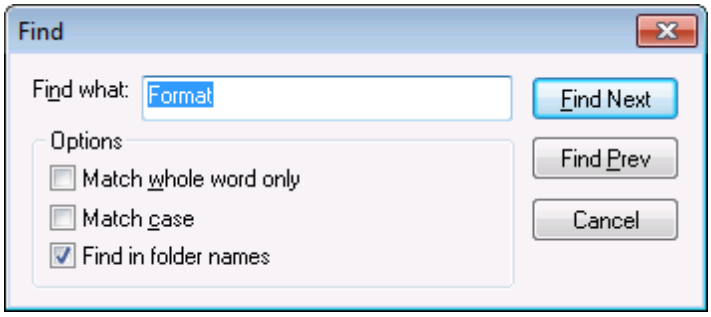
4.4.1 Opening, Searching, and Closing Projects

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

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

To search for files within a project:

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



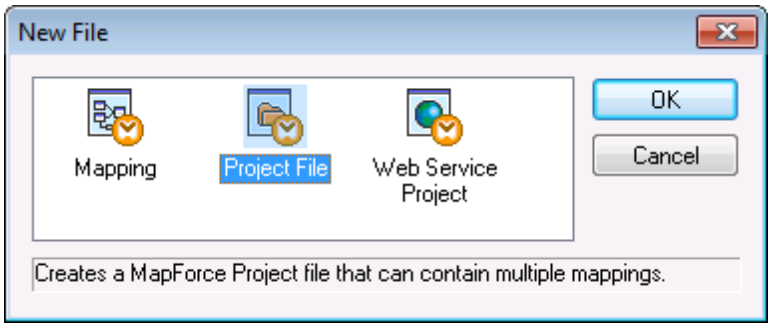
To close a project:

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

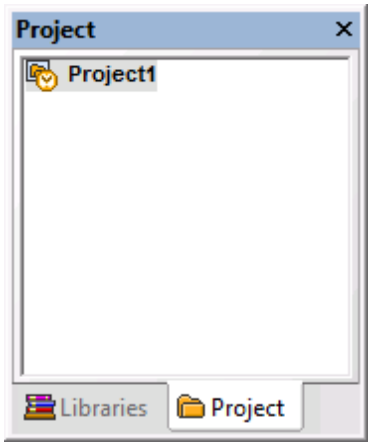
4.4.2 Creating a New Project

To create a new project:

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



3. Enter the project name in the **Save Project As** dialog box, and click **Save**. The new project is now displayed in the Project window.



You can now add mappings to the project.

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

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

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

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

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

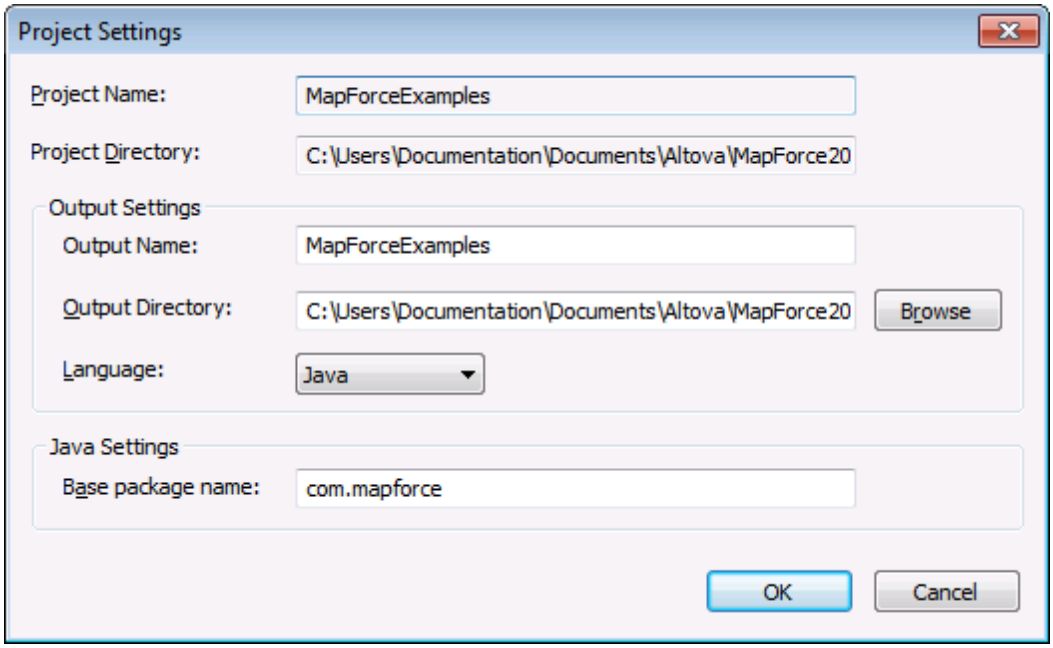
To remove a file or folder from a project, do one of the following:

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

4.4.3 Setting the Code Generation Settings

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

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



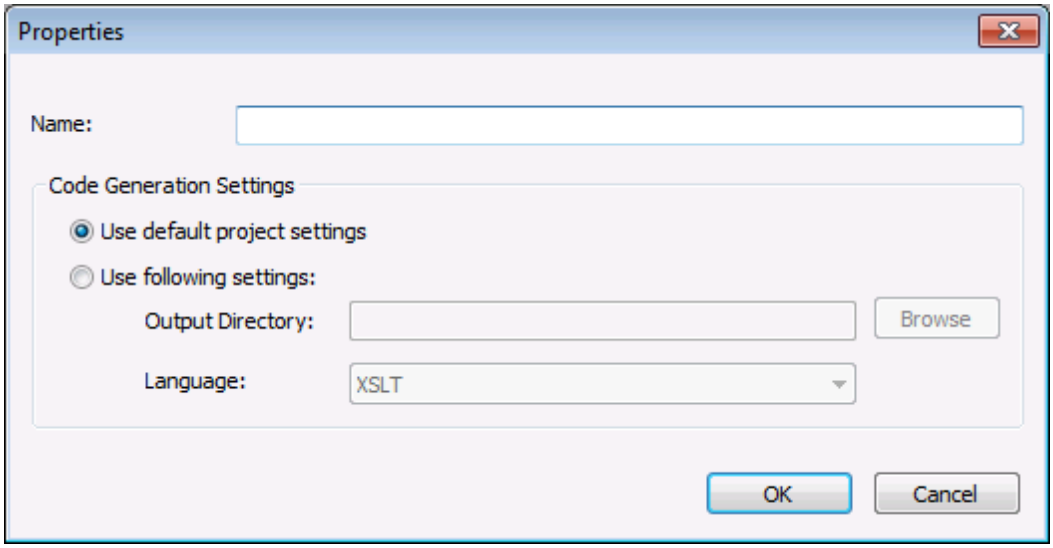
Project Settings dialog box

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

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

4.4.4 Managing Project Folders

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



Folder Properties dialog box

To create a folder inside a MapForce project:

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

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

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

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

Chapter 5

Designing Mappings

5 Designing Mappings

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

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

I want to...	Read this topic...
designing complex mappings.	

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

5.1 Using Relative and Absolute Paths

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

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

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

This section provides instructions for setting or changing the path to miscellaneous file types referenced by a mapping, and the implications of using relative versus absolute paths.

5.1.1 Using Relative Paths on a Component

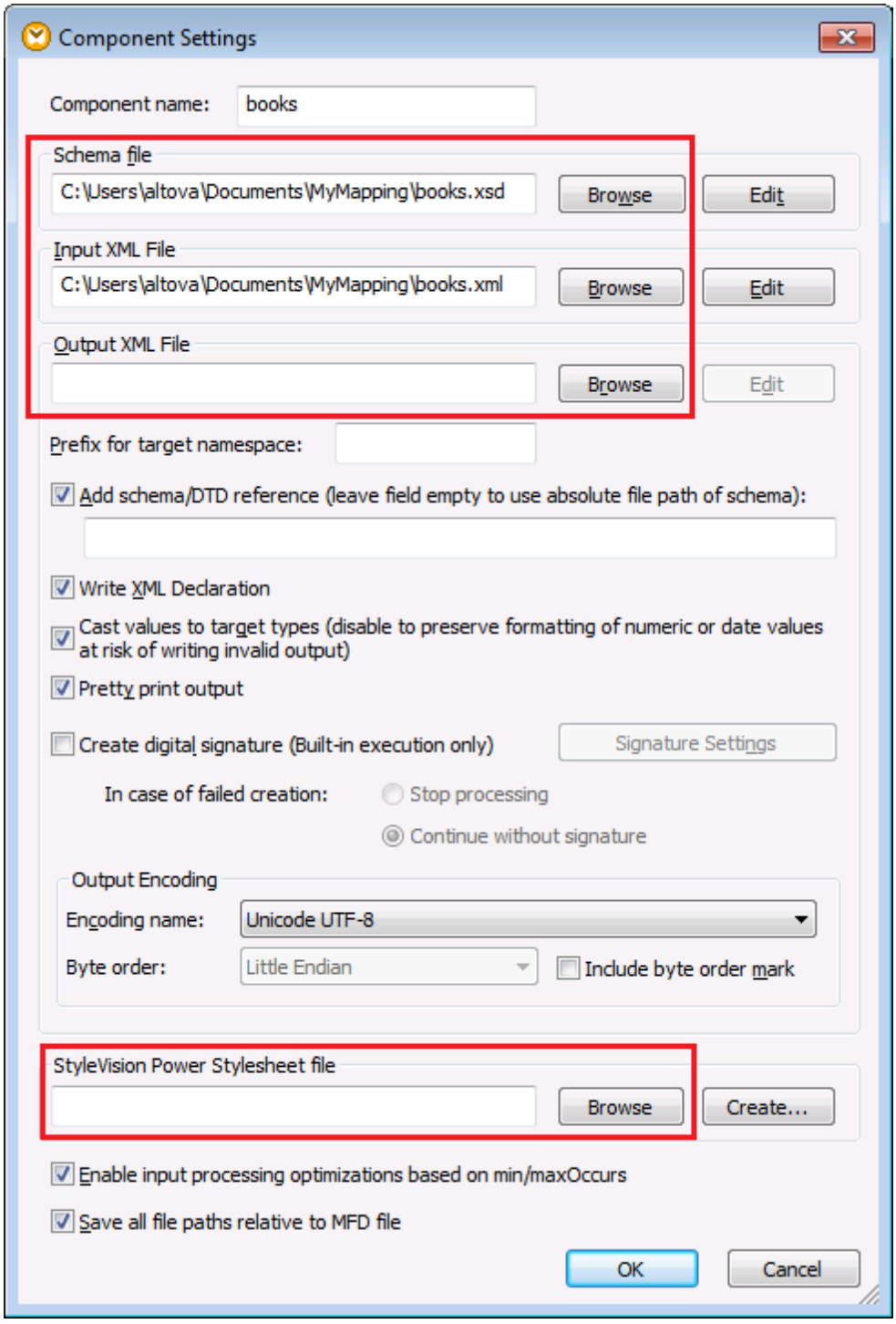
The Component Settings dialog box (illustrated below for an XML component) provides the option to specify either absolute or relative paths for various files which may be referenced by the component:

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

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

Before entering relative file paths, make sure to save the mapping file (.mfd) first. Otherwise, all relative paths are resolved against the personal application folder of Windows (Documents \Altova\MapForce2016), which may not be the intended behavior.

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



Component Settings dialog box

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

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

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

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

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

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

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

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

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

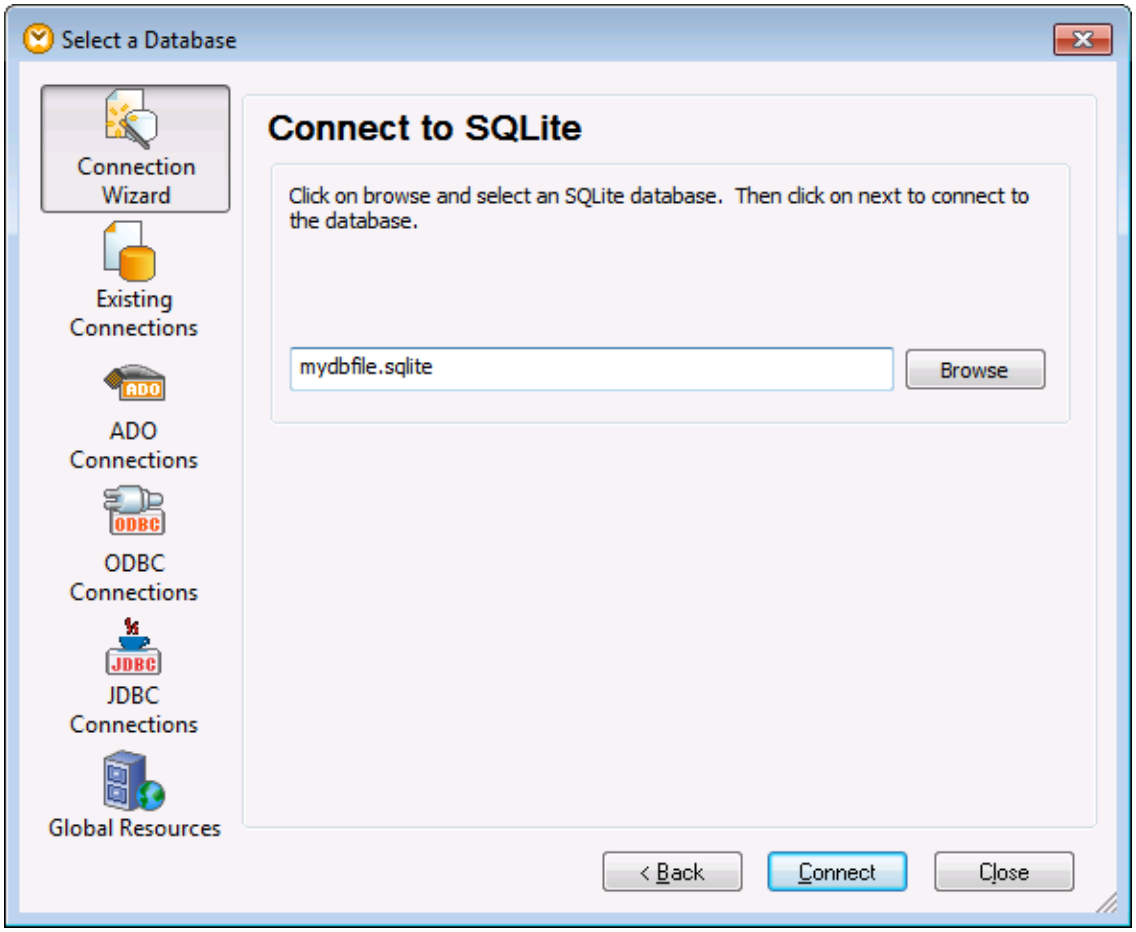
- The location of the mapping on your operating system is likely to change in future.
- The mapping is stored in a directory which is under source control (using a version control system such as TortoiseSVN, for example).
- You intend to deploy the mapping for execution to a different machine or even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

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

5.1.2 Setting the Path to File-Based Databases

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

Before entering relative file paths, make sure to save the mapping file (.mfd) first. Otherwise, all relative paths are resolved against the personal application folder of Windows (Documents \Altova\MapForce2016), which may not be the intended behavior.



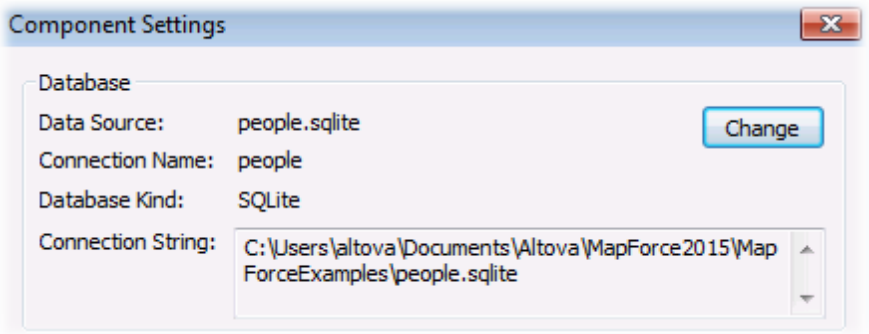
Database Connection Wizard

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

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

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

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



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

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

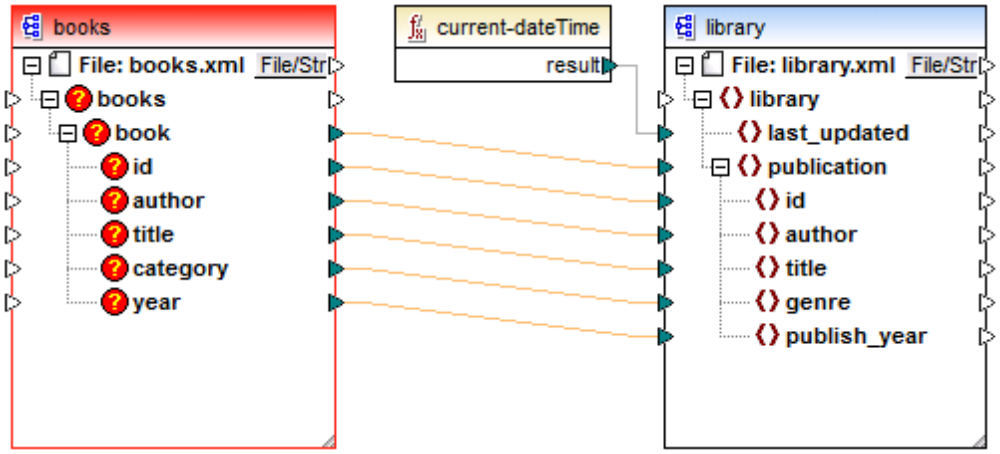
When you generate code, or when you compile MapForce Server execution files (.mfx), the path of the database will be absolute or relative, depending on the mapping settings (see About Paths in Generated Code).

5.1.3 Fixing Broken Path References

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

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

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



Broken path reference

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

5.1.4 About Paths in Generated Code

If you generate code from mappings, or if you compile mappings to MapForce Server execution files (.mfx), the generated files will no longer be run by MapForce, but by the target environment you have chosen (for example, RaptorXML Server, MapForce Server, or a C# application). The implication is that, for the mapping to run successfully, any relative paths must be meaningful in the environment where the mapping runs.

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

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

** Languages available in MapForce Professional and Enterprise editions*

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

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

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

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

When the check box is disabled, the file paths will be written in the generated code as they are defined in the component settings.

5.1.5 Copy-Paste and Relative Paths

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

5.2 Connection Types

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

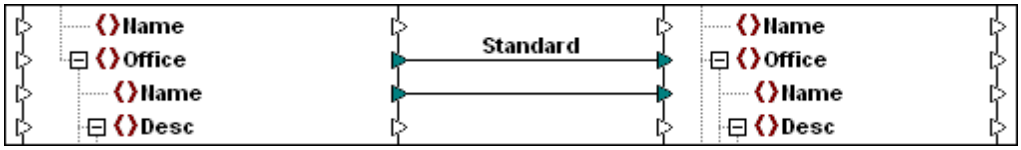
- Target Driven (Standard)
- Source Driven (Mixed Content)
- Copy-All (Copy Child Items).

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

5.2.1 Target-driven connections

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

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



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

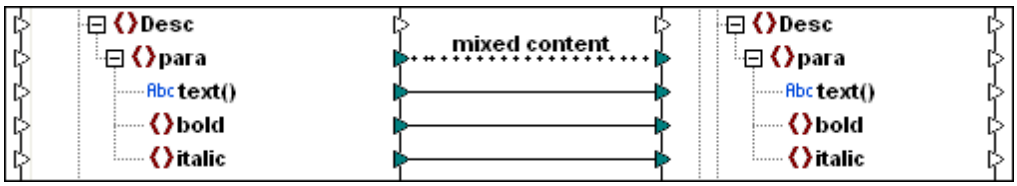
```
<p>This is our <i>best-selling</i> product.</p>
```

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

5.2.2 Source-driven connections

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

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



Mixed content mappings are shown with a dotted line.

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

Source-driven / mixed content mapping supports:

Mappings from

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

Note: CDATA sections are treated as text.

5.2.2.1 Mapping mixed content

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

Source XML instance

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp2 U (http://www.altova.com) by Mr. Nobody (Altova GmbH) -->
<OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Tut-OrgChart.xsd">
  <CompanyLogo href="nanonull.gif"/>
  <Name>Organization Chart</Name>
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Desc>
      <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull
develops nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999
saw the unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand
its operations <italic>offshore</italic>to drive down operational costs.
        <?sort alpha-ascending?>
        <!--Company details: location and general company information.-->
      </para>
      <para>White papers and further information will be made available in the near future.
    </Desc>
```

Please note the **sequence** of the text and bold/italic nodes of Nanonull., Inc in the XML instance

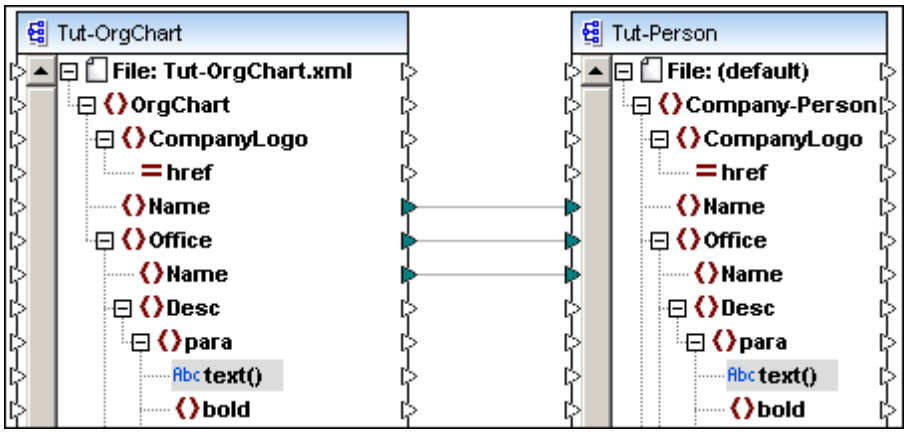
file, they are:

```
<para> The company...
    <bold>Vereno</bold>in 1995 ...
    <italic>multi-core...</italic>February 1999

    <bold>Nano-grid.</bold>The company ...
    <italic>offshore...</italic>to drive...
</para>
```

Initial mapping

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



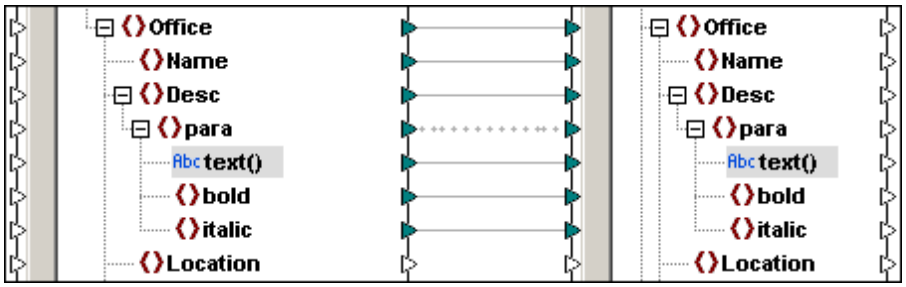
Output of above mapping

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

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
3    <Name>Organization Chart</Name>
4    <Office>
5      <Name>Nanonull, Inc.</Name>
6    </Office>
7    <Office>
8      <Name>Nanonull Europe, AG</Name>
9    </Office>
10 </Company-Person>
11
```

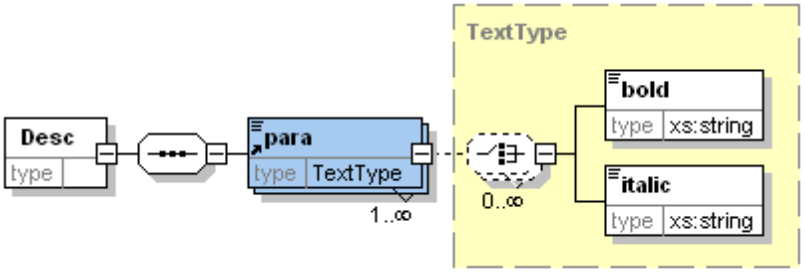
Mapping the para element

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



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

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



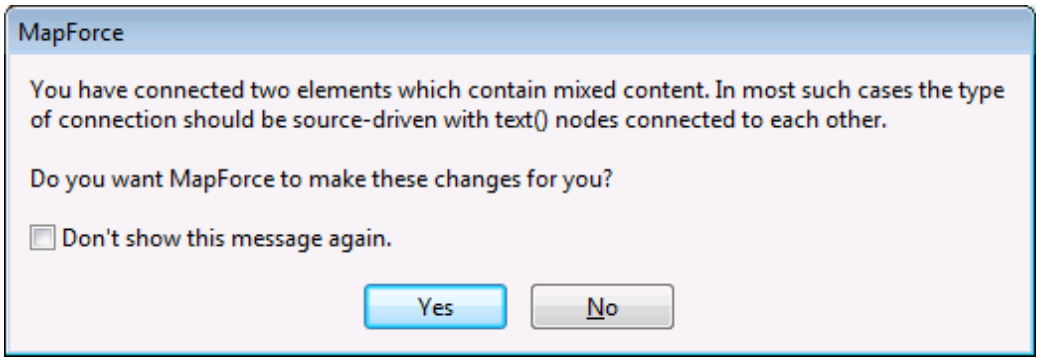
Note the following properties of the **para** element in the Content model:

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

To create mixed content connections between items:

1. Select the menu option **Connection | Auto Connect Matching Children** to activate this option, if it is not currently activated.
2. Connect the **para** item in the source schema, with the **para** item in the target schema.

   A message appears, asking if you would like MapForce to define the connectors as source driven.



3. Click Yes to create a mixed content connection.

Please note:

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

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

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


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

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

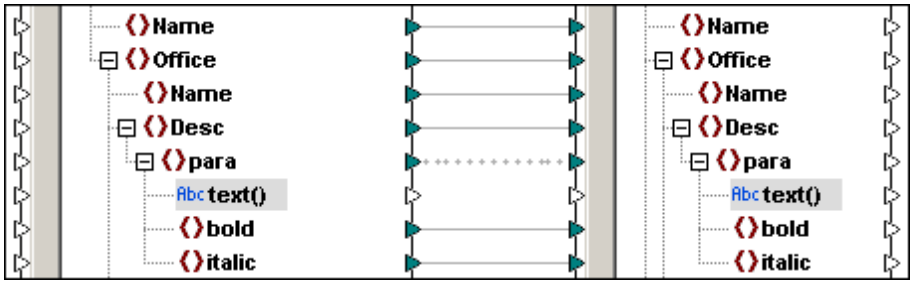
```
  <Name>Organization Chart</Name>
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Desc>
      <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull develops
nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999 saw the
unveiling of the first prototype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

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

6. Switch back to the Mapping view.

To remove text nodes from mixed content items:

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



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

```
5     <Name>Nanonull, Inc.</Name>
6     <Desc>
7       <para>
8         <bold> Vereno</bold>
9         <italic>multi-core processors.</italic>
10        <bold>Nano-grid.</bold>
11        <italic>offshore</italic>
12      </para>
13      <para/>
14    </Desc>
15  </Office>
16  <Office>
17    <Name>Nanonull Europe, AG</Name>
18    <Desc>
19      <para>
20        <italic>Europe</italic>
21        <bold> five research scientists </bold>
22      </para>
23    </Desc>
```

Result:

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

To map the Processing Instructions and Comments:

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

5.2.2.2 Mixed content example

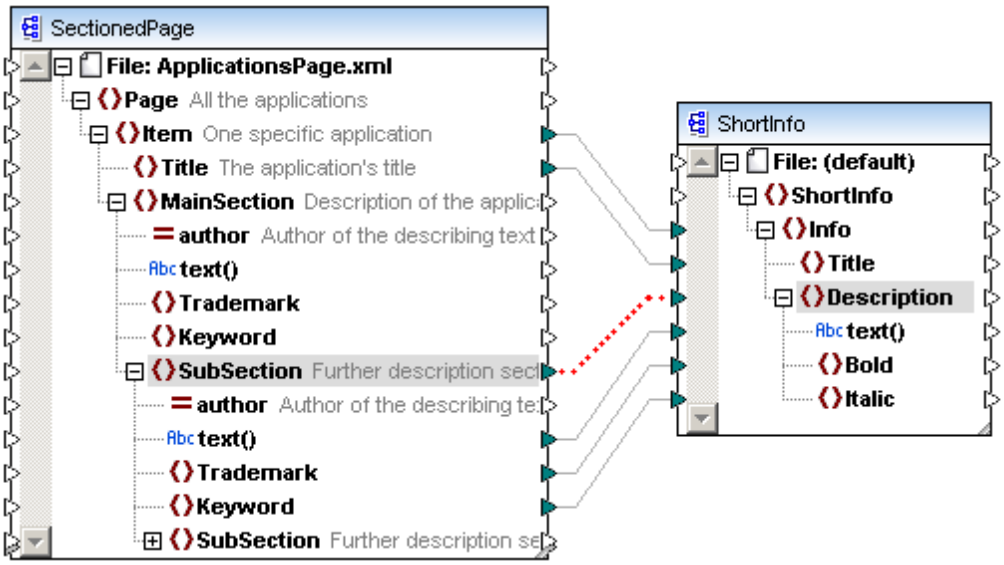
The following example is available as "**ShortApplicationInfo.mfd**" in the **...\MapForceExamples** folder.

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

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

The mapping is shown below. Please note that:

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



Mapping result

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

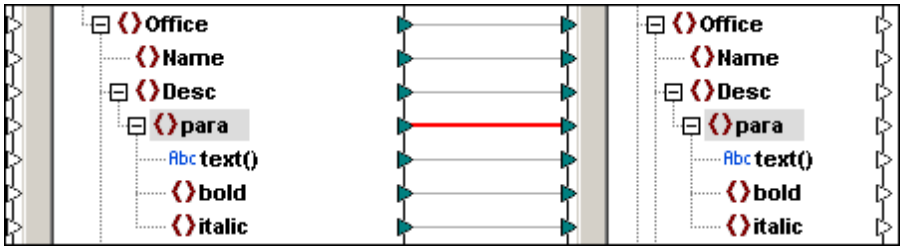
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ShortInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="
   C:/PROGRA~1/Altova/MapForce2005/MapForceExamples/ShortInfo.xsd">
3      <Info>
4          <Title>XMLSpy</Title>
5          <Description>Altova <Bold>XMLSpy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

5.2.2.3 Using standard mapping on mixed content items

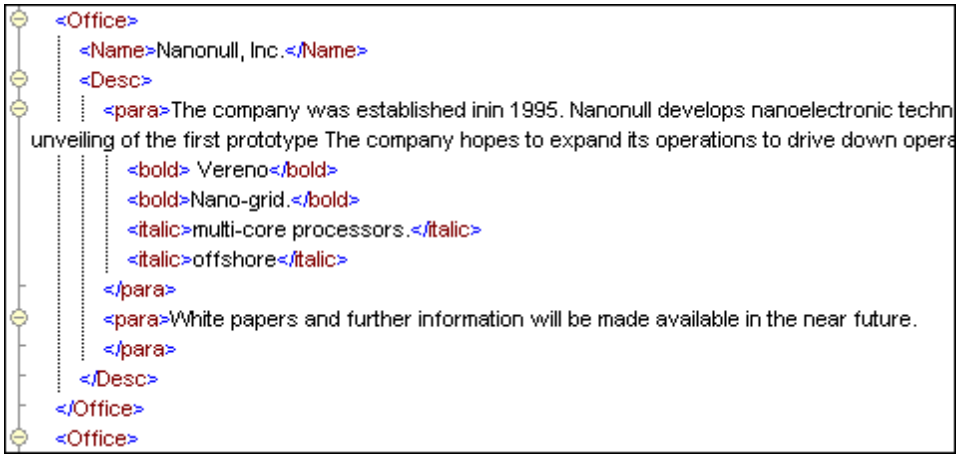
This section describes the results when defining **standard** mappings (or using standard connectors) on **mixed content** nodes. The files used in the following example (**Tut-OrgChart.mfd**) are available in the ...\MapForceExamples\Tutorial\ folder.

To create standard connections between mixed content items:

1. Create a connector between the two **para** items.
   A message appears, asking if you would like MapForce to define the connectors as source driven.
2. Click No to create a standard mapping.



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



Result

Mapping mixed content items using standard mapping produces the following result:

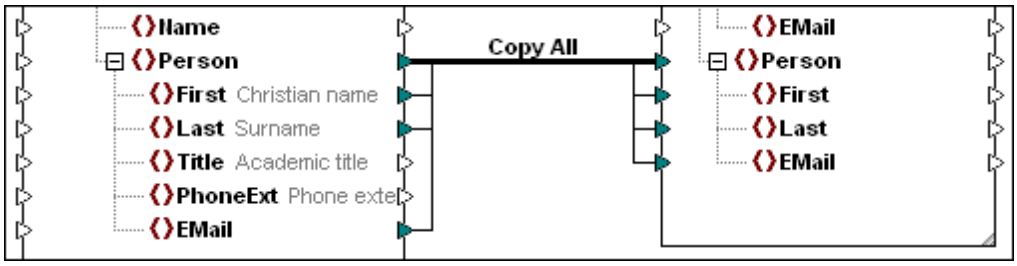
- Text() **content** is supported/mapped.
- The start/end tags of the child nodes, bold and italic, are removed from the text node.
- The child nodes appear after the mixed content node text.
- The **sequence** of child nodes depends on the sequence in the **target** XML/schema file.

That is:
For each **para** element, map the text() node, then **all bold** items, finally **all italic** items. This results in the child item sequence shown above: bold, bold - italic, italic. The content of each item is mapped if a connector exists.

5.2.3 Copy-all connections

This type of connection allows you to simplify your workspace and automatically connect **all** identical items in source and target components, meaning that, depending on the source and target **type**, all source child items are **copied** to the target component, if either the source and target **types** are **identical**, or if the target type is xs:anyType.

If the source and target **types** are **not identical,** and if the target type is not xs:anyType, the source data is transferred/mapped to the respective target items of the same name and the same hierarchy level. If the names of the target items differ, then the mapping to the target item is not created.



Connections of type "Copy-All" are shown with a single bold line that connects the various identical items of source and target components.

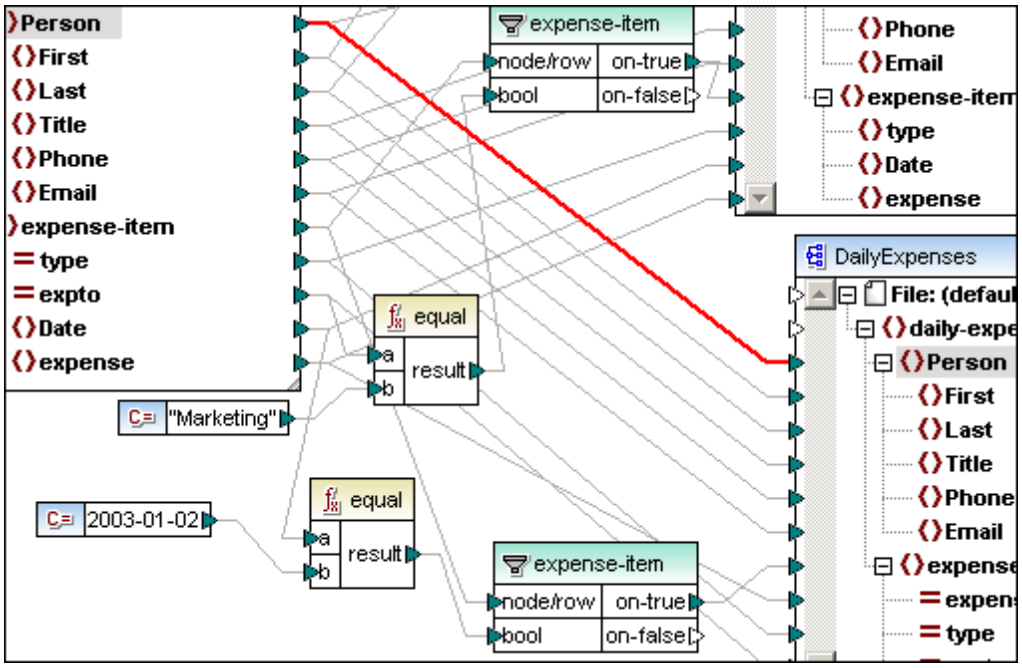
Note that only the names of the child items, but not their individual types, are compared/matched.

Currently, "Copy-All" connections are supported (i) between XML schema complex types, and (ii) between complex components (XML schema, database, EDI) and complex user-defined functions/ components containing the same corresponding complex parameters.

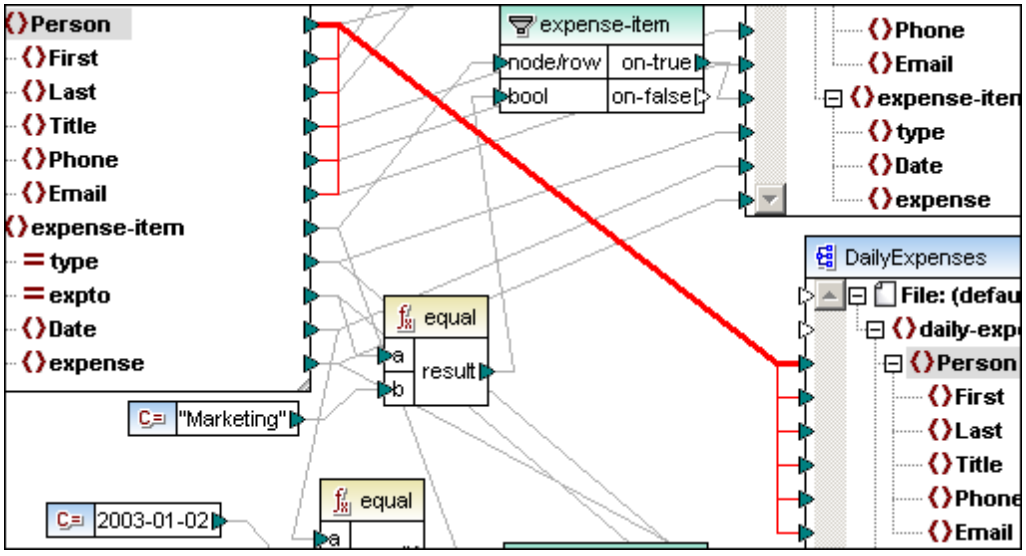
The example below uses the **MarketingAndDailyExpenses.mfd** sample available in the **...\MapForceExamples** folder.

To create a "Copy-All" connection:

1. Right-click an existing connection which has child items, and select **Copy-All (Copy Child Items)** from the context menu. A dialog box appears asking for your confirmation.



2. Click OK to create Copy-all connectors. If any child items have identical names in the source and target, connections between them are created automatically.



Please note:

- When the existing target connections are deleted, connections from other source components functions are also deleted.
- A "Copy-all" connection cannot be created between an item and the root element of a schema component.
- Individual connections cannot be deleted or reconnected from the Copy-all group, once you have used this method.

To resolve/delete "Copy-All" connections:

1. Connect any item to a child item of the copy-all connection at the target component.

You are notified that only one connector can exist at the target item. Click Replace to replace the connector.

2. Click the **Resolve copy-all connection** button in the next message box that opens. The copy-all connection is replaced by individual connectors to the target component.

Copy-all connections and user-defined functions

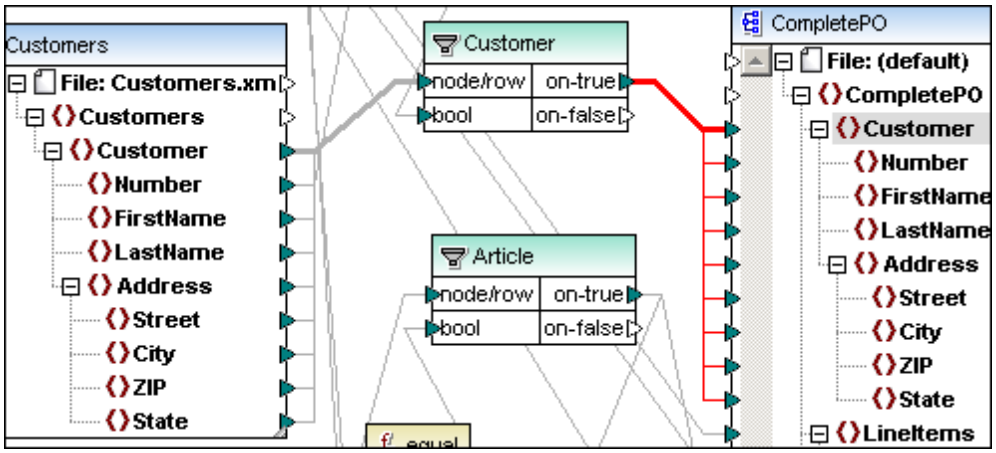
When creating Copy-all connections between a schema and a user-defined function parameter, the two components must be based on the same schema. It is not necessary that they both have the same root elements, however. Please see "Complex output components - defining" for an example.

Copy-all connections and filters

Copy-all connections can also be created through filter components if the source component:

- consists of structured data, meaning a schema, database, or EDI component,
- receives data through a complex output parameter of a user-defined function, or Web service,
- receives data through another filter component.

Only the filtered data is passed on to the target component.



To create a copy-all connection through a filter component:

1. Create a connector from the `on-true/on-false` item to the target item, e.g. Customer.
2. Right-click the connector and select **Copy-all (Copy child items)** from the context menu.

To influence what happens when filter components are deleted, see Moving Connections and Child Connections.

5.3 Chained Mappings

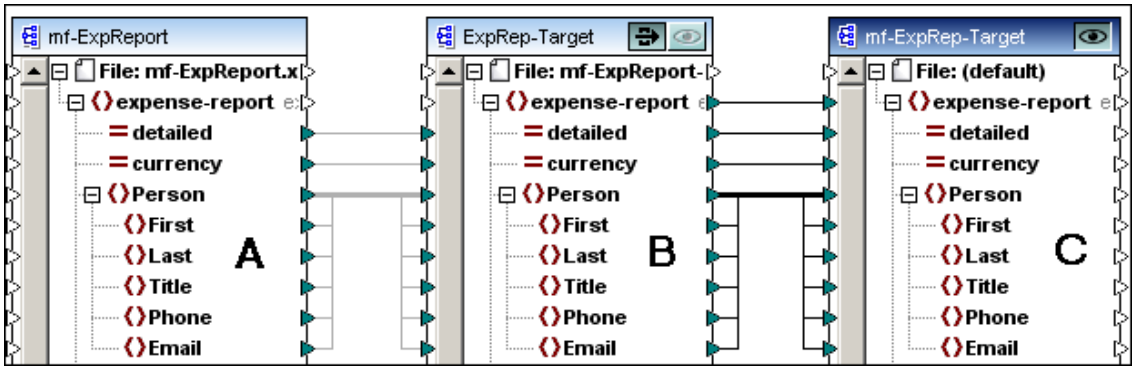
MapForce supports mappings that consist of multiple components in a mapping chain. Chained mappings are mappings where at least one component acts both as a source and a target. Such a component creates output which is later used as input for a following mapping step in the chain. Such a component is called an "intermediate" component.

Chained mappings introduce a feature called "pass-through", which allows you to create intermediate file outputs of "intermediate" components for preview, command line execution and in code generation. "Pass-through" is a preview capability allowing you to view the various stages of a chained mapping in the Output window.

If the mapping is executed from the command line, or generated code, then regardless of the entries in the Input/Output XML File fields of the "intermediate" component, the full mapping chain is executed, and the output of a previous step of a mapping chain is forwarded as input to the following mapping step.

Note: The "pass-through" feature is available only for file-based components (for example, XML, CSV, and text). Database components can be intermediate, but the pass-through button is not shown. The intermediate component is always regenerated from scratch when previewing or generating code. This would not be feasible with a database as it would have to be deleted prior to each regeneration.

The screenshot below shows three components A, B, and C, where C is the target component. Component B (ExpRep-Target) is the "intermediate" component, as it has both input and output connections.



Note that when executing a chained mapping using the command line, or executing the generated code, the mapping executes all steps in the correct order and generates the necessary output files.

Preview button

Both the component B and the component C have preview buttons. This allows you to preview the intermediate mapping result of B, as well as the final result of the chained mapping of component C in the Built-in execution engine. Click the preview button of the respective component, then click Output to see the mapping result.

"Intermediate" components with the pass-through button active cannot be previewed, since the preview button is automatically disabled. To see the output of such a component, click the "pass-

through" button to deactivate it, and then click the preview button of the intermediate component.

Pass-through button

The intermediate component B has an extra button in the component title bar called "pass-through".

If the pass-through button is **active** , MapForce maps all data into the preview window in one go; from component A to component B, then on to component C. Two result files will be created:

- the result of mapping component A to intermediate component B
- the result of the mapping from the intermediate component B, to target component C.

If the pass-through button is **inactive** , MapForce will execute only parts of the full mapping chain. Data is generated depending on which Preview buttons are active on the components B or C:

- If the Preview button of component B is active, then the result of mapping component A to component B is generated. The mapping chain actually stops at component B. Component C is not involved in the preview at all.
- If the Preview button of component C is active, then the result of mapping intermediate component B to the component C is generated. When pass-through is inactive, automatic chaining has been interrupted for component B. Only the right part of the mapping chain is executed. Component A is not used.

If the mapping is executed from the command line, or generated code, then, regardless of the settings of the pass-through button of component B, as well as the currently selected preview component, the output of all components is generated.

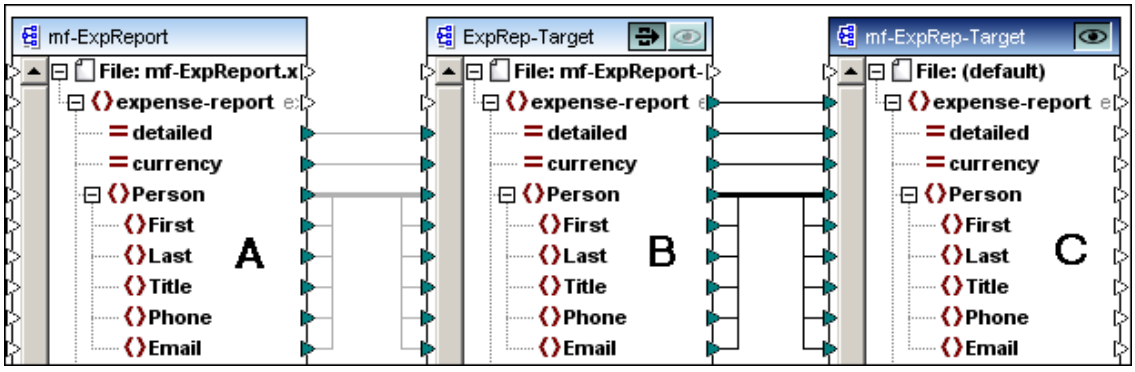
In our sample, two result files will be generated. This is the case because MapForce automatically analyzes the dependency of all components and generates all outputs of intermediate and final target components in the correct order.

Since the "pass-through" setting is currently inactive, it is vital that the intermediate component B has identical file names in the "Input XML file" and "Output XML file" fields.

Please see the following sections for more on this example, and how the source data is transferred differently when the pass-through button is active or inactive. Please see Chained mapping example for a more plausible example.

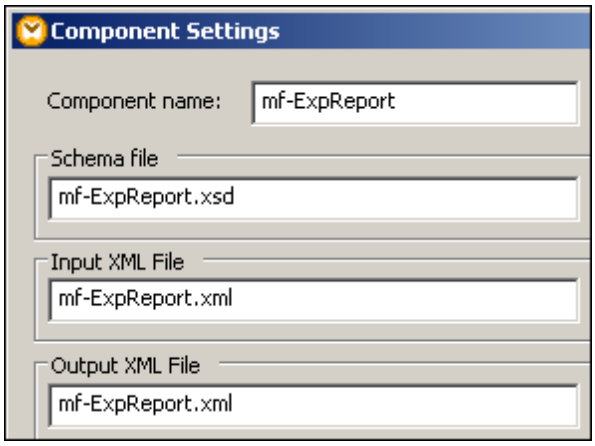
5.3.1 Chained mappings - Pass-through active

The files used in the following example (**Tut-ExpReport-chain.mfd**) are available in the **...\MapForceExamples\Tutorial** folder.



The Tut-ExpReport-chain.mfd example (*screenshot above*) is set up as follows:

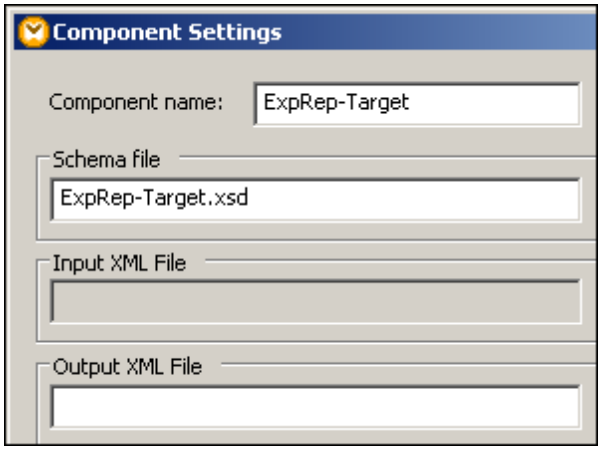
- **Component A** supplies all the mapping data, using a sample XML file. The XML file (mf-ExpReport.xml) appears in the *Input XML File* field of the **Component Settings** dialog box. The Output XML File, of the same name, is automatically inserted when you define an Input XML file.



- Intermediate **component B** "pass-through" active:
  When pass-through is active, the *Input XML File* field of the intermediate component, is automatically deactivated. A file name need not exist for the mapping to execute, as intermediate data is stored in temp files.

  If no Output XML File is defined, a default file name will be automatically used. If an Output XML File entry exists, then it is used for the file name of the intermediate output file.

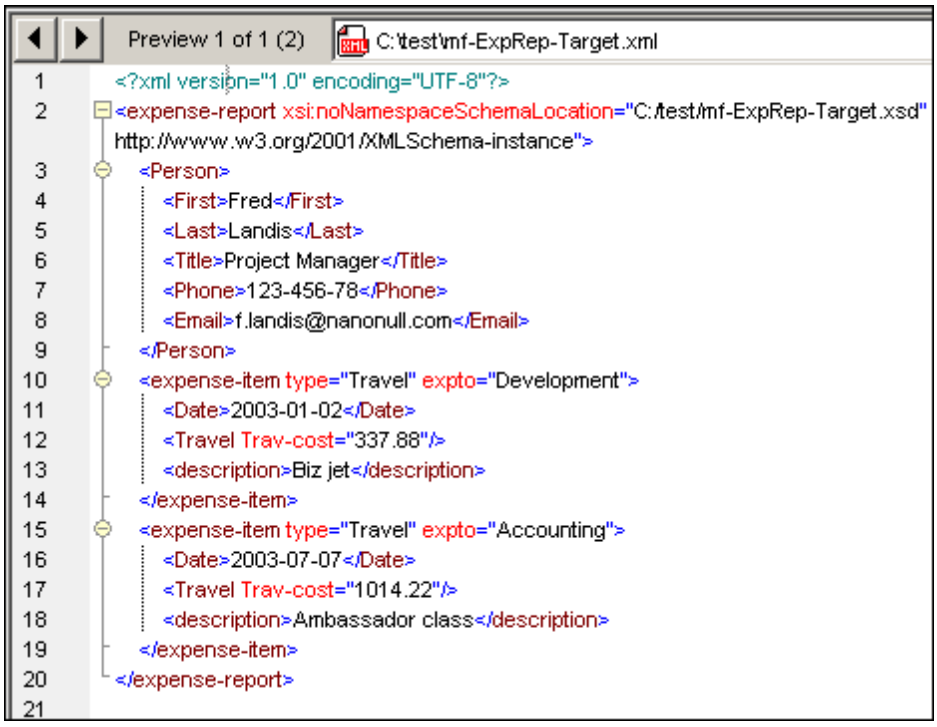
  Note that it is also possible for intermediate components to have dynamic file names i.e. connectors to the "File:" item of a component (or even file name wildcards). See Processing Multiple Input or Output Files Dynamically.



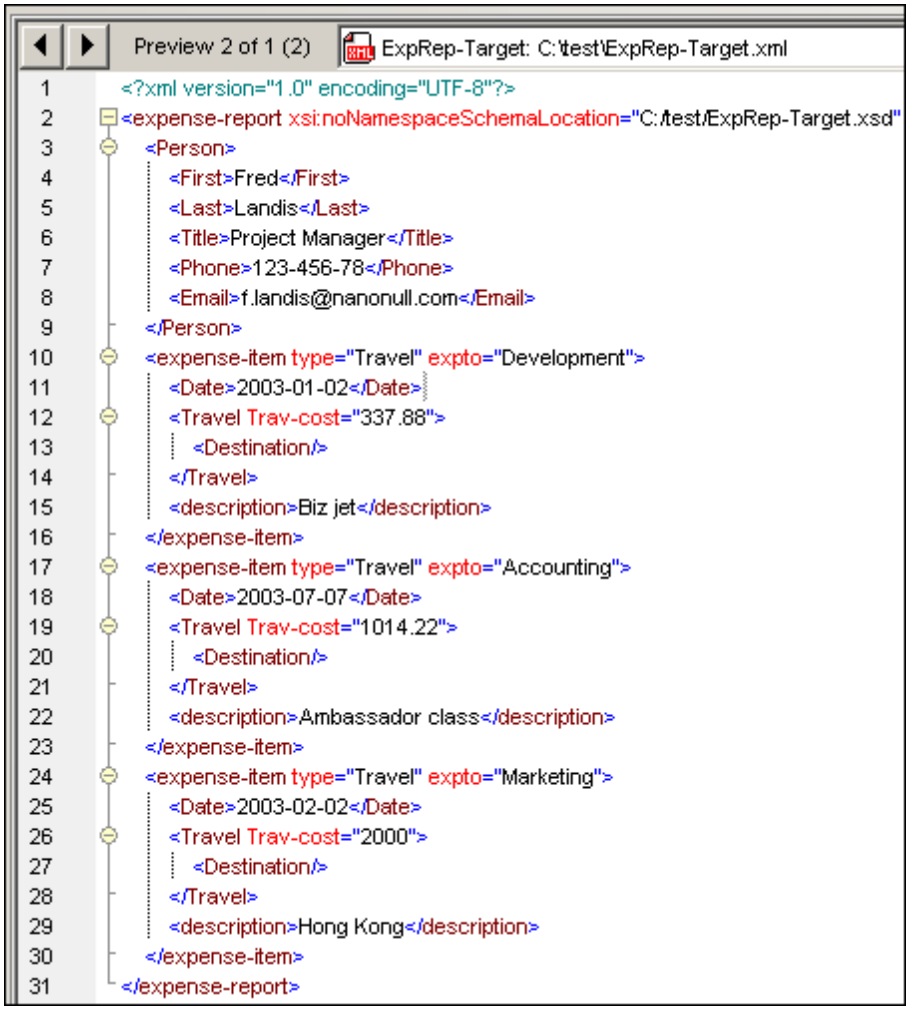
- Final **component C** does **not** have an Output XML File assigned to it. The preview button of component C is active.

Click the Output button to preview the results in the Built-in execution engine.

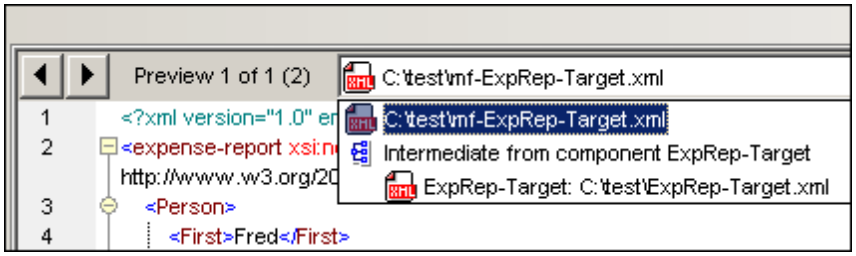
Preview 1:
The final result of the mapping from component A via intermediate component B, to target component C. These are the Travel expenses below 1500.



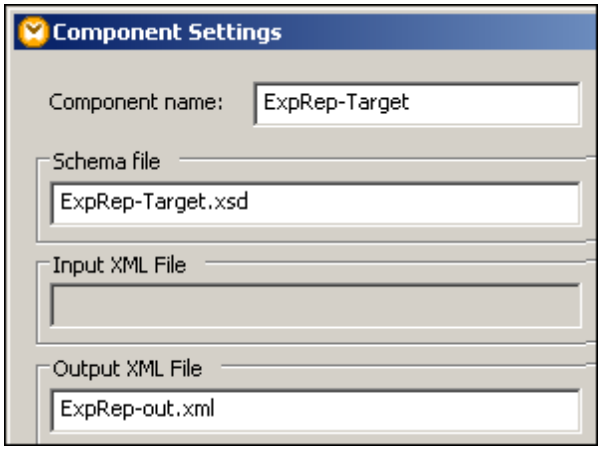
Preview 2:
The result of the mapping from component A to the intermediate component B, i.e. all Travel expense items. ExpRep-Target.xml is a default file name which is automatically generated because a file name was not entered in the Output XML file field.



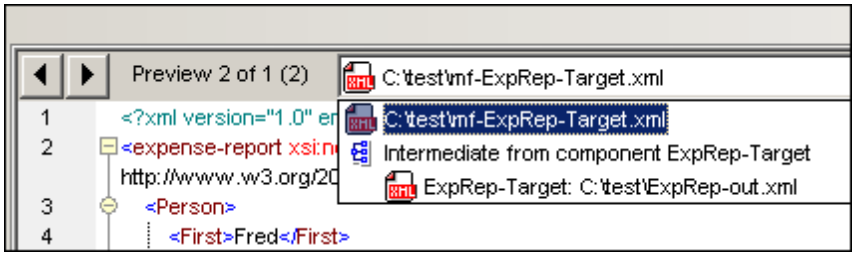
Please note:
Each mapping result is displayed in its own Preview window. Click the scroll button(s) to see the next/previous result.



Clicking the File name combo box displays the result files(s) in a hierarchy. The final target result is shown at the top, with the intermediate result file(s) shown below. Click a file name to select it, or use the keyboard keys to navigate through the file list and press Enter.



Setting an Output XML File in the intermediate component B, e.g. to "ExpRep-out.xml", causes the intermediate data of component B to be saved in a file of that name, e.g. ExpRep-out.xml.



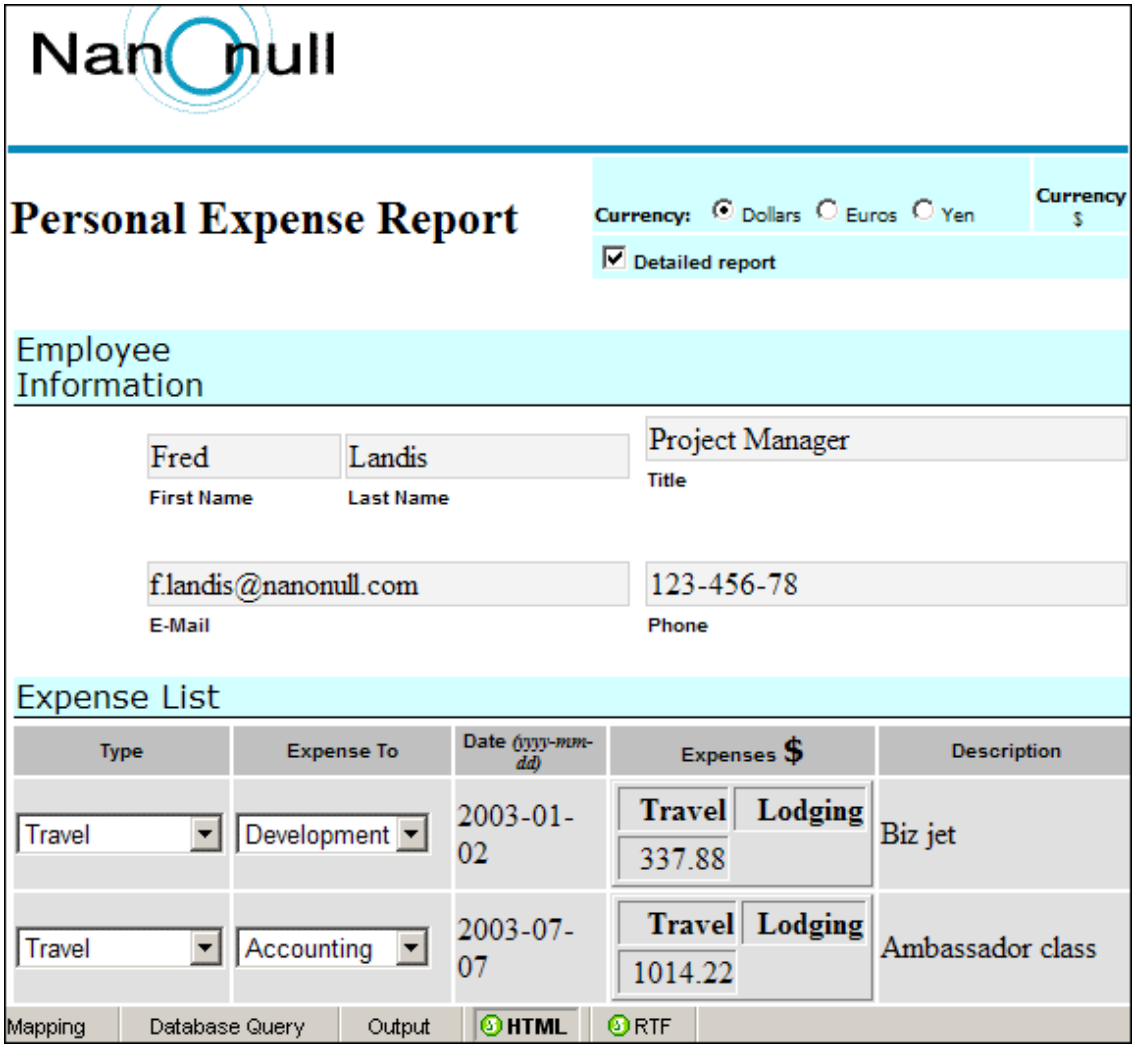
When "pass-through" is active, files created by an intermediate component are **automatically** saved as a temp files and used for further processing of that components output.

The setting "Write directly to final output file" (Tools | Options | General) determines whether the intermediate files are saved as temporary files or as physical files. For intermediate components a default file name is used to save the intermediate result, unless a dynamic file name is supplied/ mapped.

The Preview XX **of 1** means the number of final targets from the selected target component, one in this case. The Preview ... **(2**) refers to the total number of results including all intermediate components.

Displaying the result with StyleVision

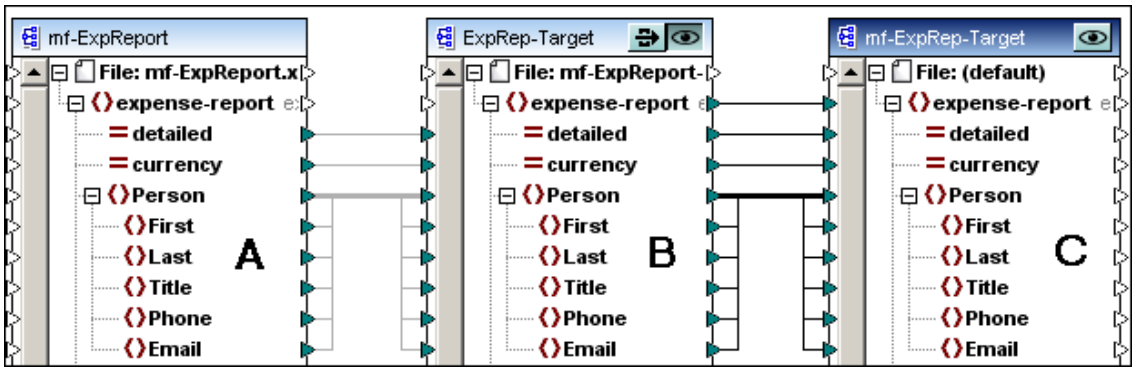
If an SPS file has been assigned to a target component, then clicking the HTML, RTF , PDF, or Word tab, will show the resulting data in the respective StyleVision tab in MapForce.



Note that only outputs of final target components of a mapping chain are shown in the StyleVision tab in MapForce. StyleVision outputs of intermediate components can not be shown.

5.3.2 Chained mappings - Pass-through inactive

The Tut-ExpReport-chain.mfd example works differently if the pass-through button is **inactive** on component B.



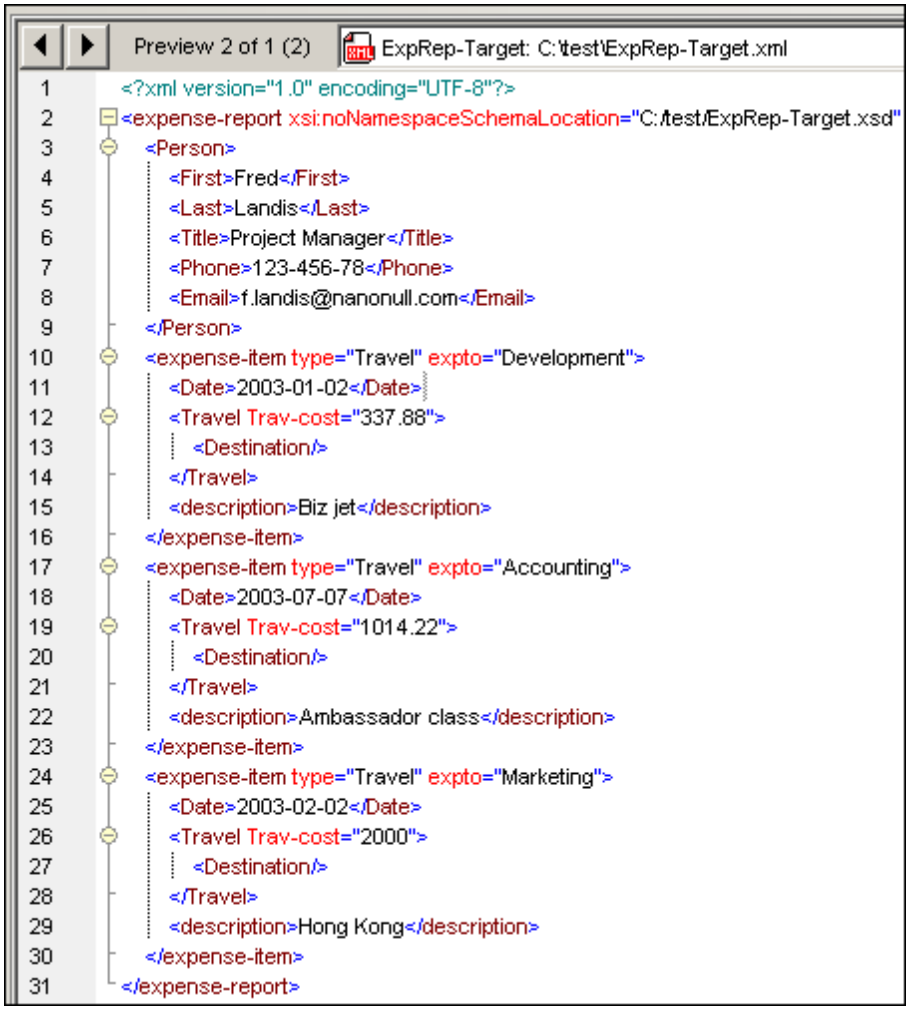
The automatic transfer of data from component A via component B and further to component C, has been interrupted by disabling the pass-through button. The Preview buttons of components B and C determine which part of the mapping chain is generated.

MapForce generates the output for the component where the Preview button is active.

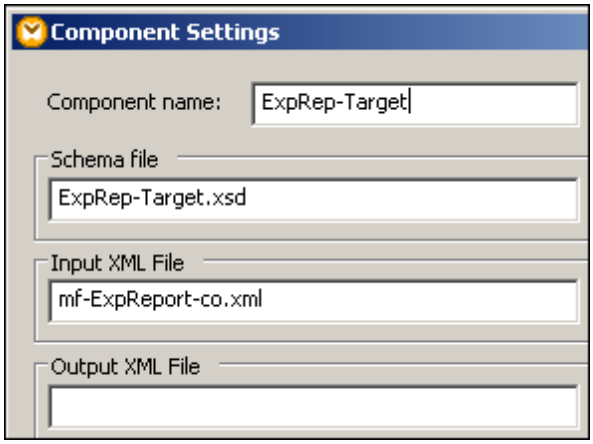
- If the Preview button of component **B** is active, then the result of mapping component A to component B is generated. Component C is ignored.

  Clicking the Output button previews the results in the Built-in execution engine.

Preview:
The result of the mapping from component A to the intermediate component B, i.e. all Travel expense items.



- If the Preview button of component **C** is active, MapForce maps the data from the intermediate component B to component C. Component A is ignored. Component B has an Input XML File, mf-ExpReport-co.xml, assigned to it, see Saving an intermediate mapping result in the text below.

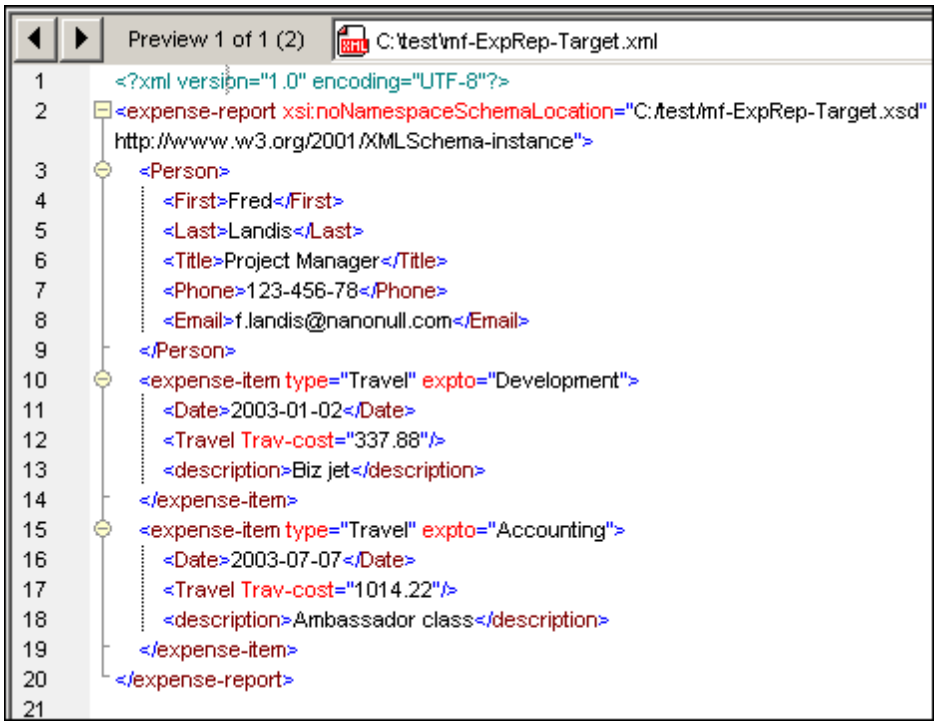


MapForce opens the intermediate file and maps its data to component C. If the input file of component B exists, this mapping will produce output. This file entry must exist here for the mapping to execute. MapForce displays an error message if the input file is missing.

When "pass-through" is inactive, the *Input XML File* field of the intermediate component is enabled, as shown above.

Note the difference to the case where component B had the "pass-through" button active, in that case the Input XML file field is automatically disabled.

Preview:
The result of the mapping from intermediate component B to the target component C, i.e. Travel expenses below 1500.



Note:
If this mapping is executed from the command line, or generated code, then regardless of the state of the pass-through button in component B and the selected preview component, MapForce attempts to generate the output of component B and component C. The setting of the preview button has no effect.

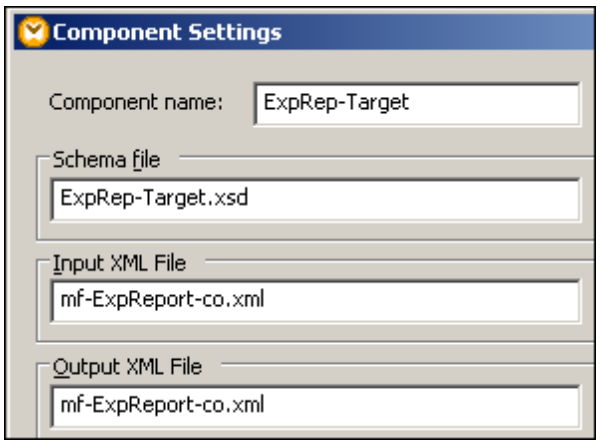
Since the Input XML file entry is different from the Output XML file entry (which is empty) the mapping chain is broken and the output for component C cannot be generated. Both the Input XML File field and Output XML File field have to be identical for code generation to succeed.

Saving the intermediate mapping result

To make the input file of the intermediate component accessible, when "pass-through" is inactive, the result of the mapping of component A to B must be saved. This file name is then placed in the *Input XML File* of component B. Only then can data be displayed in the final component C.

To save the mapping result of component B to a file:

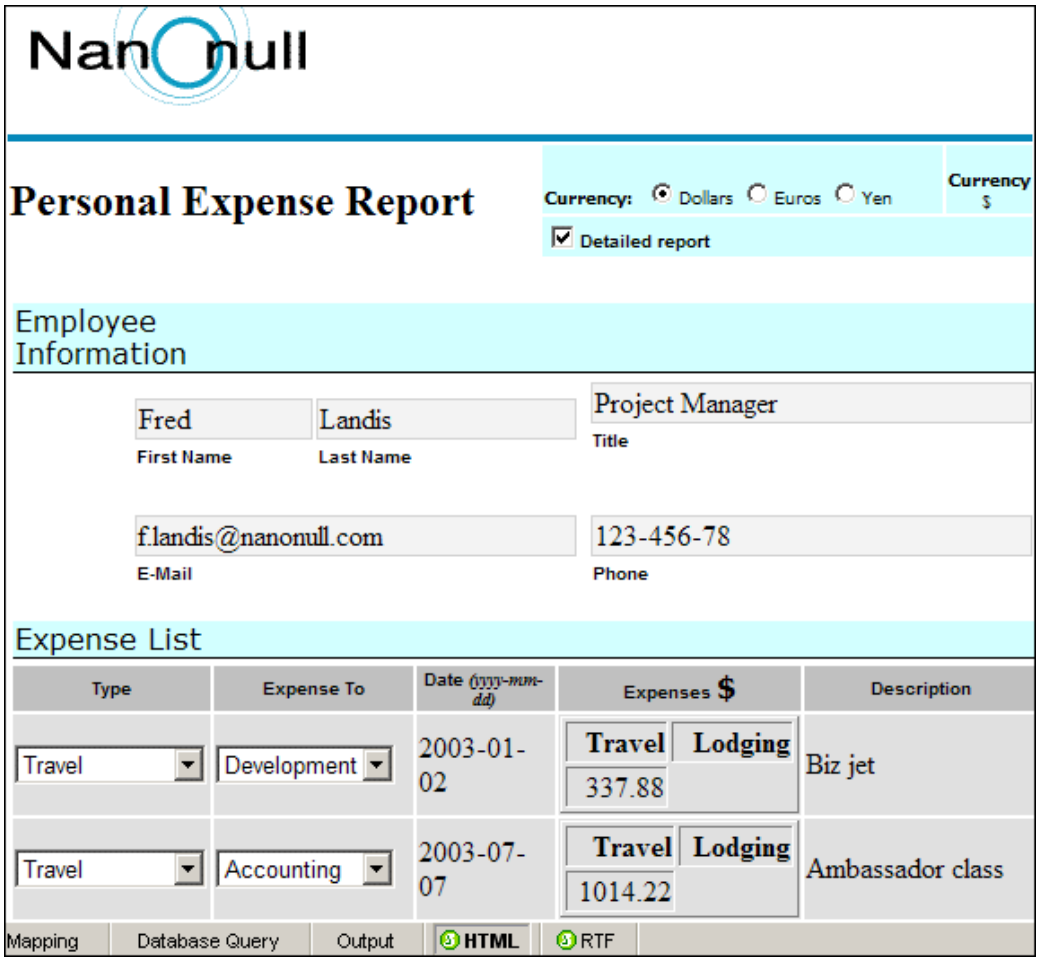
1. Click the Preview button of component B to make it active, then click the Output button.
2. Click the **Save generated output** button in the Output Preview tool bar and give the XML file a name, e.g. `mf-ExpReport-co.xml`.
3. Double click the header of component B to open the **Component Settings** dialog box, and copy the file name into the *Input XML File* field and click **OK**.



Please note: Both the Input and Output file names **must** be identical (and present) for **code generation** and execution from the command line to occur.

Displaying the result with StyleVision

If an SPS file has been assigned to a target component, then clicking the HTML, RTF , PDF, or Word tab, will show the resulting data in the respective StyleVision tab in MapForce.

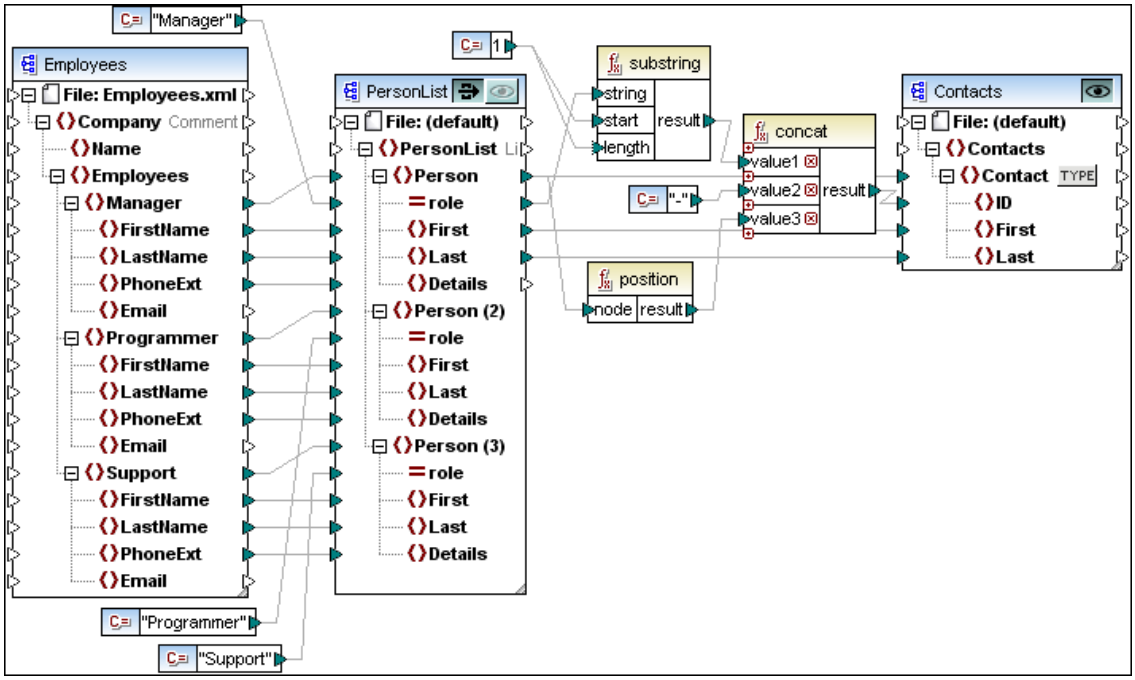


5.3.3 Chained mapping example

The example shown below is available as **ChainedPersonList.mfd** in the ...\MapForceExamples folder.

Aim:
To create two sets of employee documents, one for human resources and the other for bookkeeping.

- The document for the bookkeeping department assigns an unique ID to the employee.
- The document for the HR department has the person details, and additionally the telephone extension.



Components:

Employees:
The Employees.xml instance file contains four people with the roles in the sequence: manager, programmer and support.

PersonList: (output will be the HR document) - Intermediate component

- A **role** attribute is added to the person data, and the position hierarchy that exists in the Employees component is removed.
- The "pass-through" button is active.

Contacts: (output will be the bookkeeping document)

- An ID element is added to the Contact data to make sure that person data is unique.

How it works:

PersonList:

- The person element is **duplicated** twice to allow for the three types of roles that exist within the company.
- The **role** names are added as strings, using constant components, in the same sequence as in the Employees component.

Contacts:

- The **substring** function splits off the first character of the role attribute and forwards it to the concat function.
- The **position** function iterates over all the Person nodes, assigns a sequential number (starting at 1) and forwards it to the concat function.
- The **concat** function combines the substring character, a hyphen (from a constant component) and the position number and forwards it to the ID element of the Contacts

component.

Result:

PersonList component: (output: HR document)

Preview 2 of 1 (2) PersonList: C:\Documents and Settir

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <PersonList xsi:noNamespaceSchemaLocation="C:/DOCUME~1
   http://www.w3.org/2001/XMLSchema-instance">
3    <Person role="Manager">
4      <First>Vernon</First>
5      <Last>Callaby</Last>
6      <Details>582</Details>
7    </Person>
8    <Person role="Programmer">
9      <First>Frank</First>
10     <Last>Further</Last>
11     <Details>471</Details>
12   </Person>
13   <Person role="Support">
14     <First>Loby</First>
15     <Last>Matise</Last>
16     <Details>963</Details>
17   </Person>
18   <Person role="Support">
19     <First>Susi</First>
20     <Last>Sanna</Last>
21     <Details>753</Details>
22   </Person>
23 </PersonList>
```

Contacts component: (output: bookkeeping document)

Preview 1 of 1 (2) C:\Documents and Settings\ply\My

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Contacts xsi:noNamespaceSchemaLocation="C:/DOCUME~1
   http://www.w3.org/2001/XMLSchema-instance">
3    <Contact>
4      <ID>M-1</ID>
5      <First>Vernon</First>
6      <Last>Callaby</Last>
7    </Contact>
8    <Contact>
9      <ID>P-2</ID>
10     <First>Frank</First>
11     <Last>Further</Last>
12   </Contact>
13   <Contact>
14     <ID>S-3</ID>
15     <First>Loby</First>
16     <Last>Matise</Last>
17   </Contact>
18   <Contact>
19     <ID>S-4</ID>
20     <First>Susi</First>
21     <Last>Sanna</Last>
22   </Contact>
23 </Contacts>
```

5.4 Processing Multiple Input or Output Files Dynamically

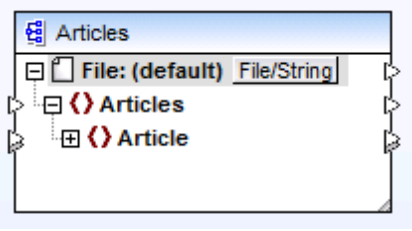
You can configure MapForce to process multiple files (for example, all files in a directory) when the mapping runs. Using this feature, you can solve tasks such as:

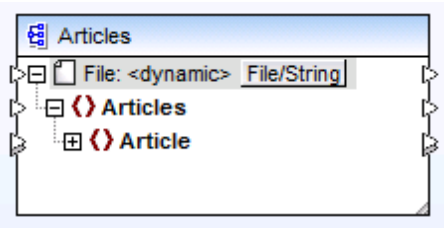
- Supply to the mapping a list of input files to be processed
- Generate as mapping output a list of files instead of a single output file
- Generate a mapping application where both the input and output file names are defined at runtime
- Convert a set of files to another format
- Split a large file (or database) into smaller parts
- Merge multiple files into one large file (or load them into a database)

You can configure a MapForce component to process multiple files in one of the following ways:

- Supply the path to the required input or output file(s) using wildcard characters instead of a fixed file name, in the component settings (see Changing the Component Settings). Namely, you can enter the wildcards * and ? in the Component Settings dialog box, so that MapForce resolves the corresponding path when the mapping runs.
- Connect to the root node of a component a sequence which supplies the path dynamically (for example, the result of the `replace-fileext` function). When the mapping runs, MapForce will read dynamically all the input files or generate dynamically all the output files.

Depending on what you want to achieve, you can use either one or both of these approaches on the same mapping. However, it is not meaningful to use both approaches at the same time on the same component. To instruct MapForce which approach you want to use for a particular component, click the **File** (File) or **File/String** (File/String) button available next to the root node of a component. This button enables you to specify the following behavior:

Use File Names from Component Settings	If the component should process one or several instance files, this option instructs MapForce to process the file name(s) defined in the Component Settings dialog box. If you select this option, the root node does not have an input connector, as it is not meaningful.  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.  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>**.  This option is mutually exclusive with the *Use File Names from Component Settings* option.
Parse Strings to XML, Parse Strings to JSON, Parse Strings to CSV, Parse Strings to FLF, Parse Strings to EDI	When switched on, this option enables the component to accept a string value as input to the root node, and convert it to an XML, JSON, CSV, FLF, or EDI structure, respectively. For more information, see Parsing and Serializing Strings.
Serialize XML to Strings, Serialize JSON to Strings, Serialize CSV to Strings,	When switched on, this option enables the component to accept a structure as input, and

Serialize FLF to Strings, Serialize EDI to Strings	convert it to string. The input structure can be XML, JSON, CSV, Fixed-length Field, or EDI, respectively. For more information, see Parsing and Serializing Strings.

Multiple input or output files can be defined for the following components:

- XML files
- Text files (CSV*, FLF* files and FlexText** files)
- EDI documents**
- Excel spreadsheets**
- XBRL documents**

** Requires MapForce Professional Edition*
*** Requires MapForce Enterprise Edition*

The following table illustrates support for dynamic input and output file and wildcards in MapForce languages.

Target language	Dynamic input file name	Wildcard support for input file name	Dynamic output file name
XSLT 1.0	*	Not supported by XSLT 1.0	Not supported by XSLT 1.0
XSLT 2.0	*	*(1)	*
XQuery	*	*(1)	Not supported by XQuery
C++	*	*	*
C#	*	*	*
Java	*	*	*
BUILTIN	*	*	*

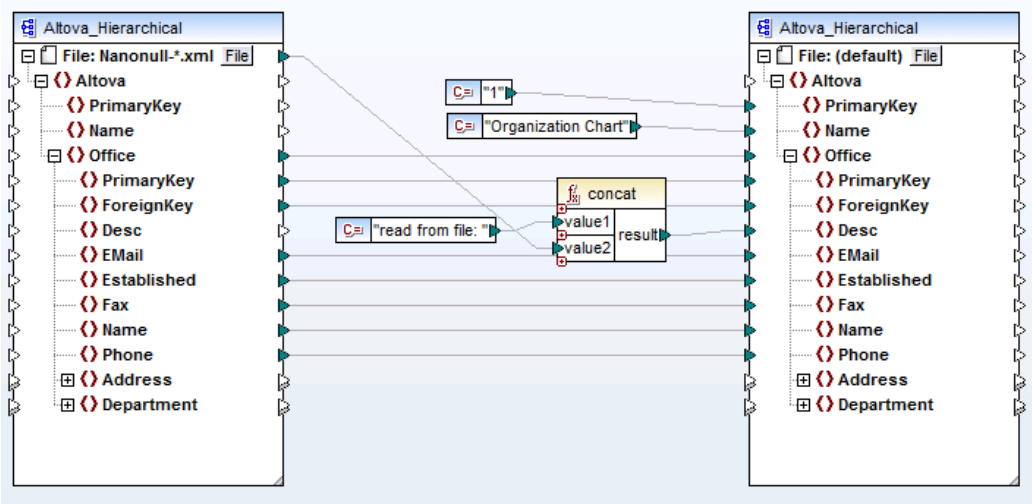
Legend:

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

5.4.1 Mapping Multiple Input Files to a Single Output File

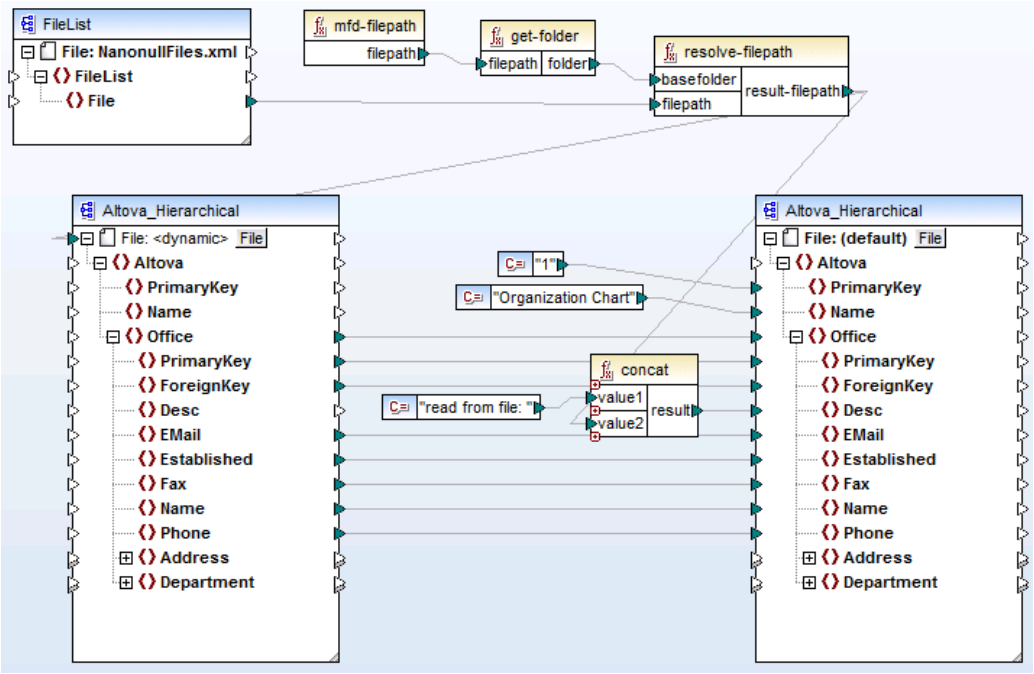
To process multiple input files, do one of the following:

- Enter a file path with wildcards (* or ?) as input file in the Component Settings dialog box. All matching files will be processed. The example below uses the ***** wildcard character in the Input XML file field to supply as mapping input all files whose name begins with "Nanonull-". Multiple input files are being merged into a **single** output file because there is no dynamic connector to the target component, while the source component accesses multiple files using the wildcard *. Notice that the name of the root node in the target component is **File: <default>**, indicating that no output file path has been defined in the Component Settings dialog box. The multiple source files are thus appended in the target document.



MergeMultipleFiles.mfd (MapForce Basic Edition)

- Map a **sequence** of strings to the *File* node of the source component. Each string in the sequence represents one file name. The strings may also contain wildcards, which are automatically resolved. A sequence of file names can be supplied by components such as an XML file , database text fields, text files (CSV or fixed length), or an Excel sheet.

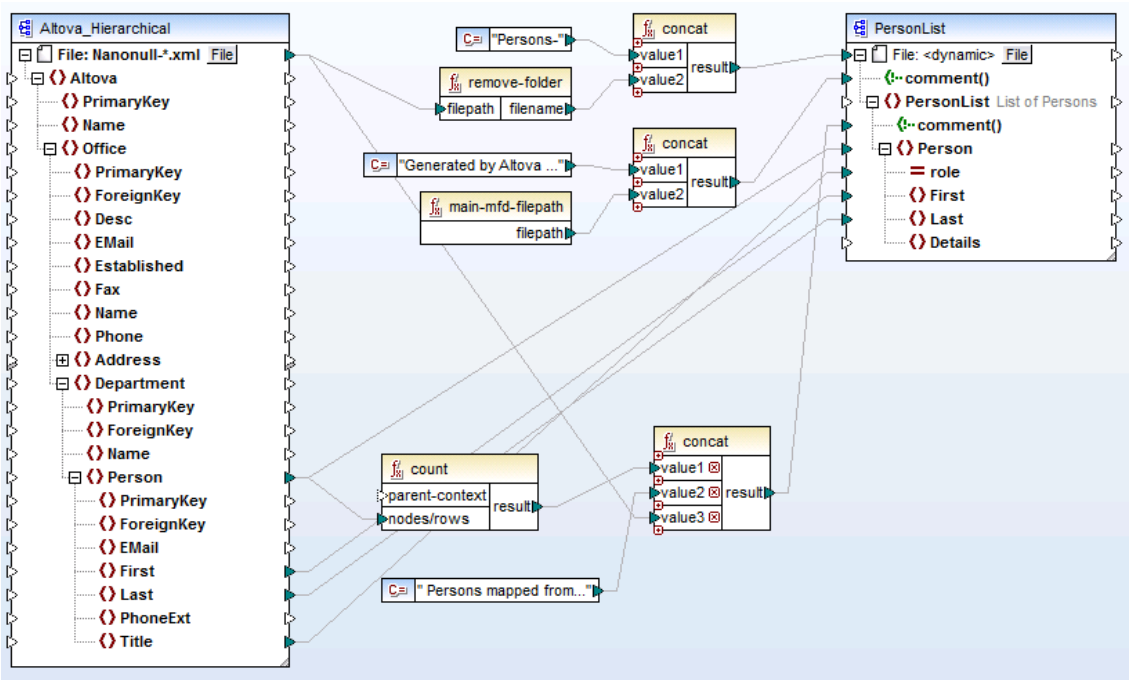


MergeMultipleFiles_List.mfd (MapForce Basic Edition)

5.4.2 Mapping Multiple Input Files to Multiple Output Files

To map multiple files to multiple target files, you need to generate unique output file names. In some cases, the output file names can be derived from strings in the input data, and in other cases it is useful to derive the output file name from the input file name, e.g. by changing the file extension.

In the following mapping, the output file name is derived from the input file name, by adding the prefix "Persons-" with the help of the **`concat`** function.



MultipleInputToMultipleOutputFiles.mfd (MapForce Basic Edition)

Note: Avoid simply connecting the input and output root nodes directly, without using any processing functions. Doing this will overwrite your input files when you run the mapping. You can change the output file names using functions such as the **concat** function, as shown above.

The menu option **File | Mapping Settings** allows you to define globally the file path settings used by the mapping (see Changing the mapping settings).

5.4.3 Supplying File Names as Mapping Parameters

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

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

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

5.4.4 Previewing Multiple Output Files

Click the Output tab to display the mapping result in a preview window. If the mapping produces multiple output files, each file has its own numbered pane in the Output tab. Click the arrow

buttons to see the individual output files.

MultipleInputToMultipleOutputFiles.mfd

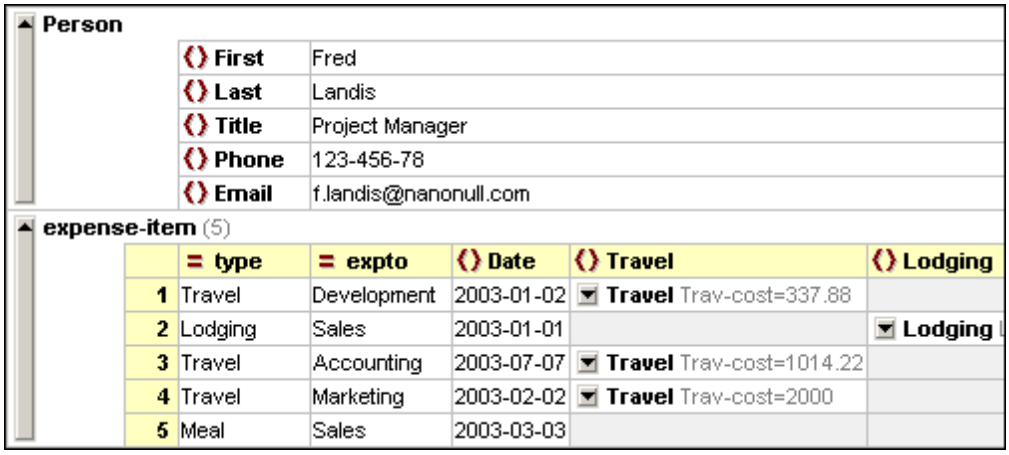
To save the generated output files, do one of the following:

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

5.4.5 Example: Split One XML File into Many

This example shows you how to generate dynamically multiple XML files from a single source XML file. The accompanying mapping for this example is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\Tut-ExpReport-dyn.mfd**.

The source XML file (available in the same folder as the mapping) consists of the expense report for a person called "Fred Landis" and contains five expense items of different types. The aim of the example is to generate a separate XML file for each of the expense items listed below.



Person

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

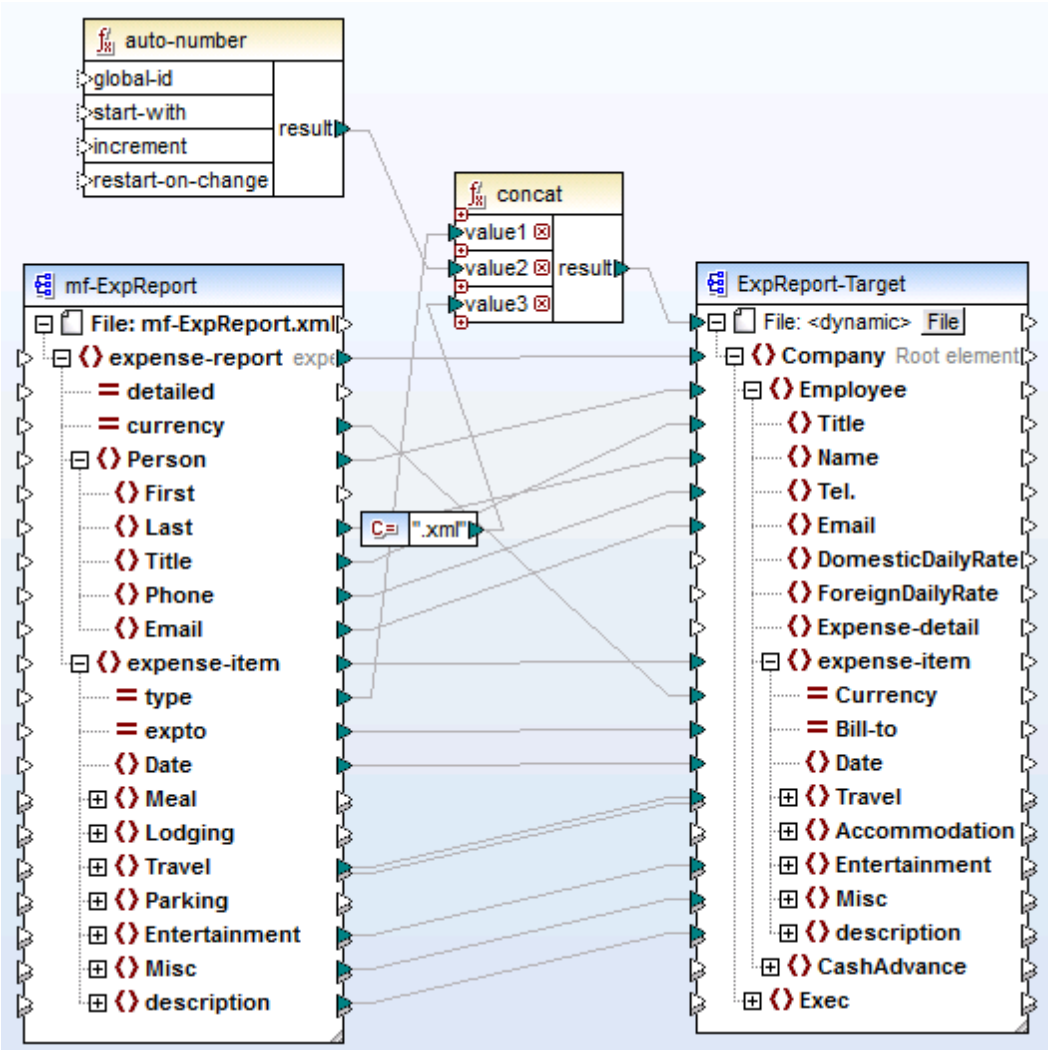
expense-item (5)

	type	expto	Date	Travel	Lodging
1	Travel	Development	2003-01-02	**Travel** Trav-cost=337.88	
2	Lodging	Sales	2003-01-01		**Lodging**
3	Travel	Accounting	2003-07-07	**Travel** Trav-cost=1014.22	
4	Travel	Marketing	2003-02-02	**Travel** Trav-cost=2000	
5	Meal	Sales	2003-03-03		

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

As the `type` attribute defines the specific expense item type, this is the item we will use to split up the source file. To achieve the goal of this example, do the following:

1. Insert a **`concat`** function (you can drag it from the **core | string functions** library of the Libraries pane).
2. Insert a constant (on the **Insert** menu, click **Constant**) and enter ".xml" as its value.
3. Insert the **`auto-number`** function (you can drag it from the **core | generator functions** library of the Libraries pane).
1. Click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.
4. Create the connections as shown below and then click the **Output** tab to see the result of the mapping.



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

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

- The `type` attribute supplies the first part of the file name (for example, "Travel").
- The **`auto-number`** function supplies the sequential number of the file (for example, "Travel1", "Travel2", and so on).
- The constant supplies the file extension, which is ".xml", thus "Travel1.xml" is the file name of the first file.

5.4.6 Example: Split Database Table into Many XML Files

This example shows you how to generate dynamically multiple XML files, one for each record of a database table. The accompanying mapping for this example is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\PersonDB-dyn.mfd**.

The source database file (available in the same folder as the mapping) includes a Person table

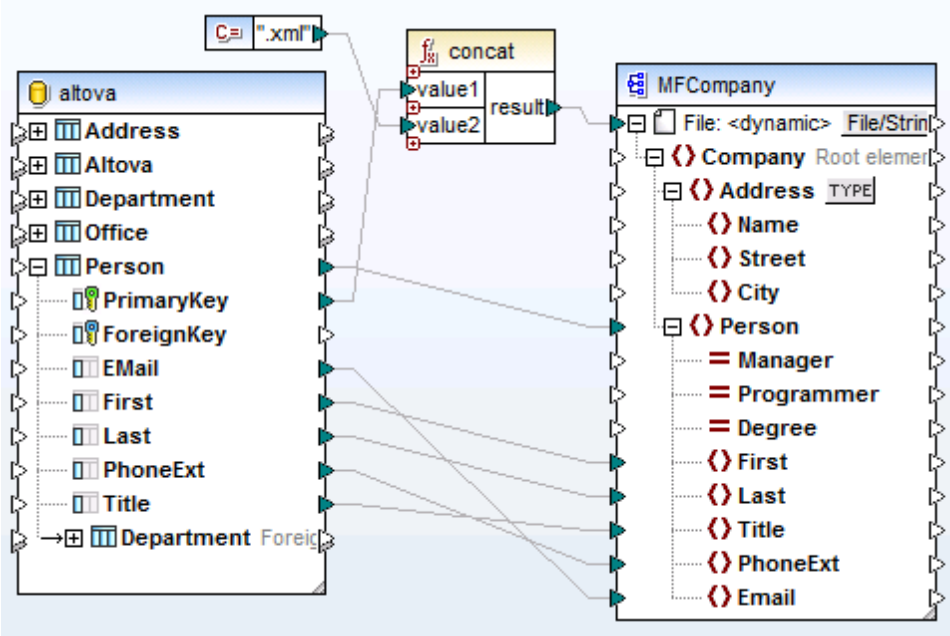
which contains 21 records. The aim of the example is to generate a separate XML file for each record in the Person table.

Person

PrimaryKey	ForeignKey	EMail	First	Last
1	1	v.callaby@nanonu	Vernon	Callaby
2	1	f.further@nanonu	Frank	Further
3	1	l.matise@nanonu	Loby	Matise
4	2	j.firstbread@nanc	Joe	Firstbread
5	2	s.sanna@nanonul	Susi	Sanna
6	3	f.landis@nanonul	Fred	Landis
7	3	m.landis@nanonu	Michelle	Butler
8	3	t.little@nanonull.	Ted	Little

As the "PrimaryKey" field uniquely identifies each person in the table, this is the item we will use to split up the source database into separate files. To achieve the goal of this example, do the following:

1. Insert a **concat** function (you can drag it from the **core | string functions** library of the Libraries pane).
2. Insert a constant (on the **Insert** menu, click **Constant**) and enter ".xml" as its value.
3. Click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.
4. Create the connections as shown below and then click the **Output** tab to see the result of the mapping.



PersonDB-dyn.mfd (MapForce Professional Edition)

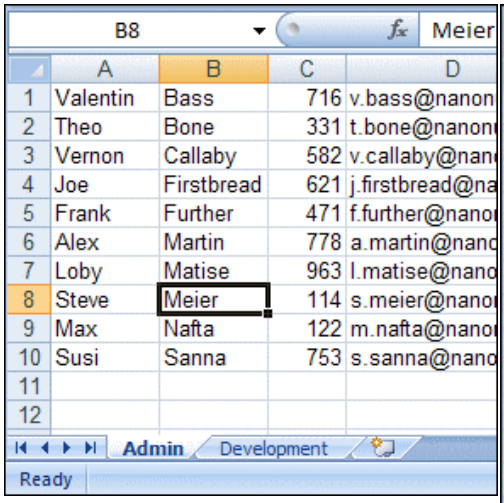
Note that the resulting output files are named dynamically as follows:

- The **PrimaryKey** field supplies the first part of the file name (for example, "1").
- The constant supplies the file extension (".xml"), thus "1.xml" is the file name of the first file.

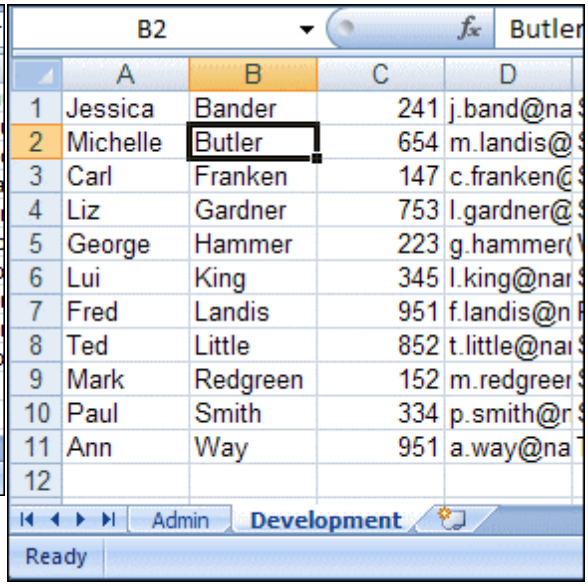
5.4.7 Multiple XML files from Excel rows

The content of the **altova.xlsx** spreadsheet file, available in the ...\MapForceExamples\Tutorial folder, is shown below. It consists of two worksheets: Admin with 10, and Development with 11, rows of data. This example is available as **Excel-Mapping-dyn.mfd** in the ...\Tutorial folder.

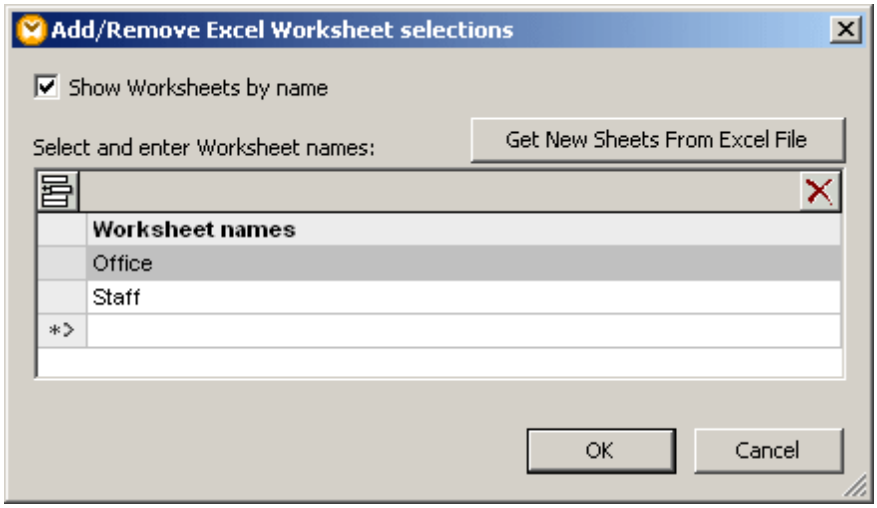
Admin worksheet



Development worksheet

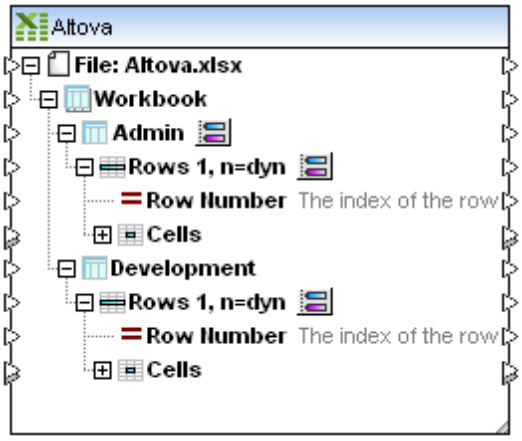


MapForce is able to display, and map, Excel components in two different ways depending on the component options. The default settings are shown in the dialog box below. The "Show Worksheets by name" check box is active when you first insert the Excel component.

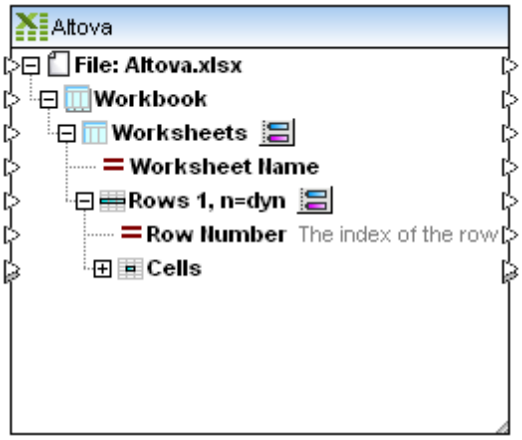


To access a Workbook as if it were a single Worksheet:

1. Click the icon next to **Admin** in the Excel component.



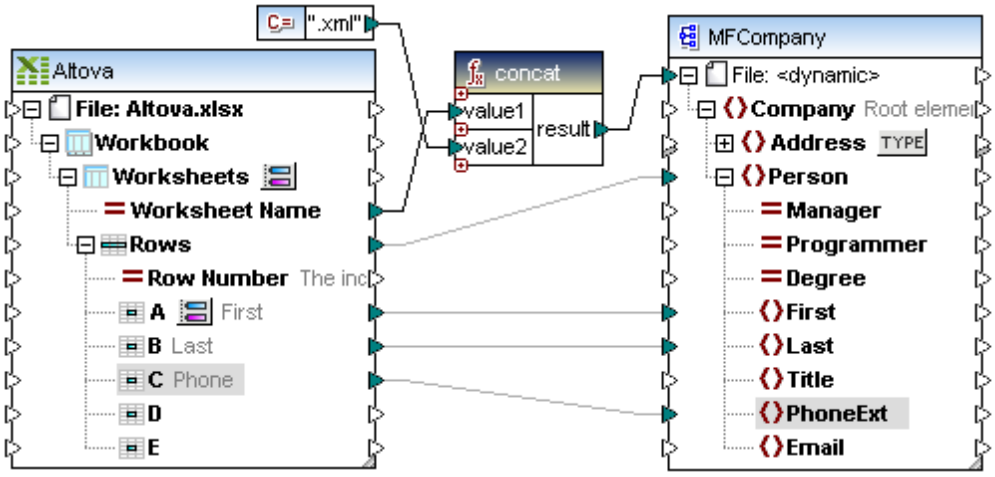
2. Click the "Show Worksheets by name" check box to deselect it.
   The named Worksheets are not visible anymore, as shown in the screenshot below, but a Worksheet Name item is now available.



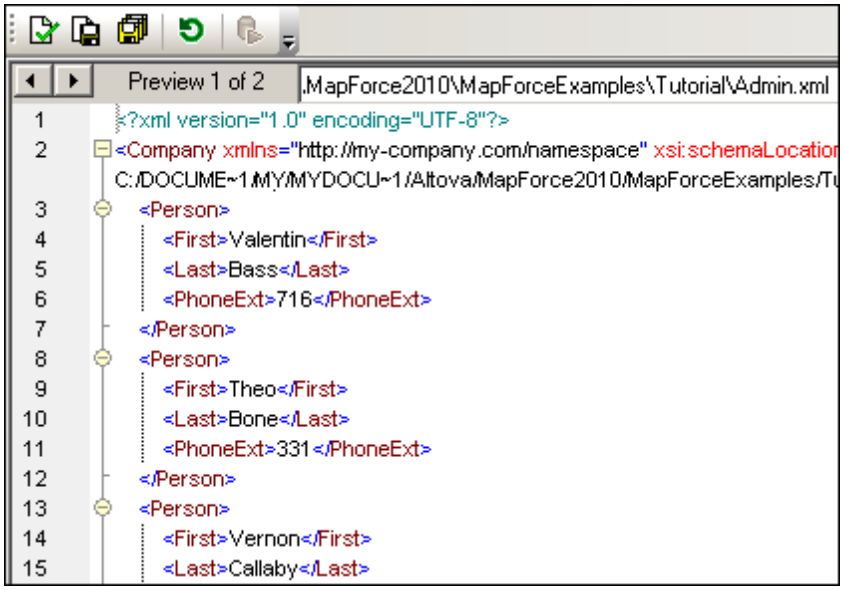
Aim 1: To generate separate files for each Worksheet containing the person records of each.

The "Worksheet Name" item determines the specific worksheets in the workbook, so this is the item we will use to split up the source workbook into separate files.

1. Insert a concat and constant function from the libraries pane.



2. Create the connections as shown above: Worksheet Name to value1 and the constant to value2.
3. Connect the **result** parameter of the concat function to the **File:** item of the target component. Note that File: <dynamic> is now displayed.
4. Define the remaining connections as needed.
5. Click the Output tab to see the result of the mapping.



Each record is now visible in its own Preview tab, the first one is shown above.

Clicking the Next/Previous arrows allows you to see each of the files in the Output tab.

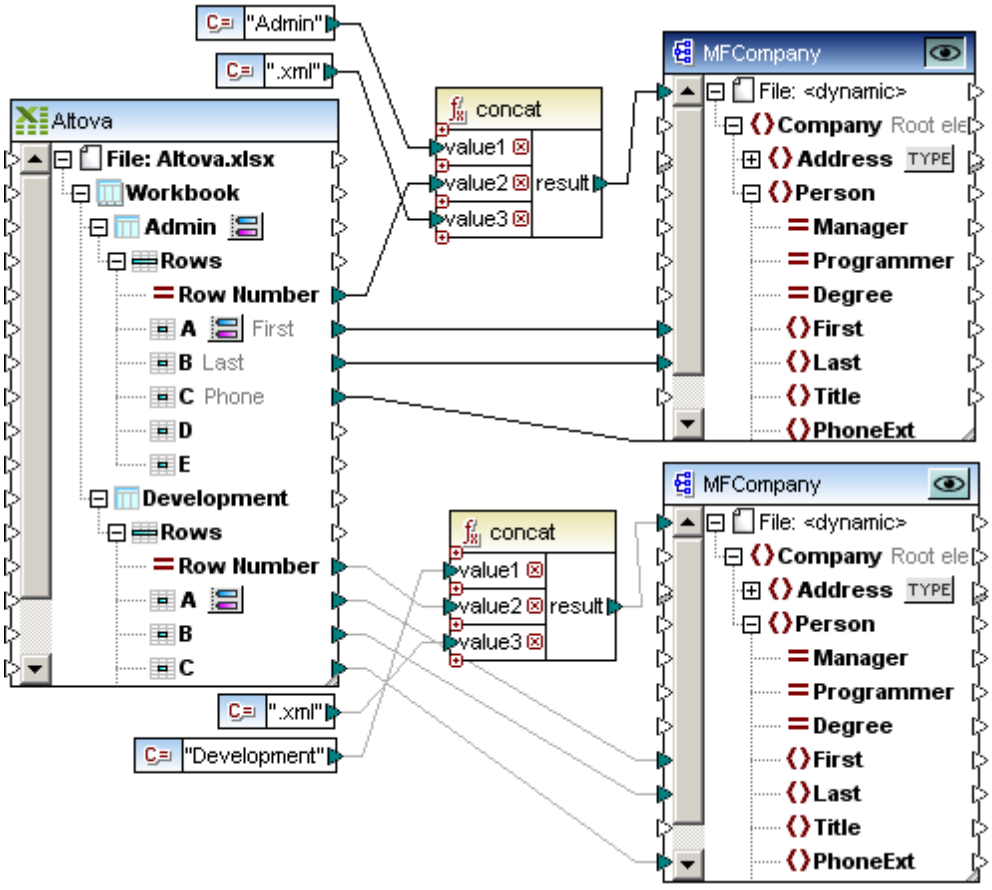
Note:

- The **WorkSheet** name field supplies the first part of the file name e.g. Admin.
- The **constant** component supplies the file extension i.e. **.xml**, thus Admin.xml is the file name of the first file. Admin.xml contains all the rows of that Excel tab. Development.xml contains the other rows.
- Clicking the Save All icon allows you to save the individual files directly from the

Output tab, without having to generate code.

Aim 2: To generate separate files for each Person in each Worksheet

1. Click the icon next to Worksheets if you followed the section above, and activate the "Show Worksheets by name" check box.
   Each of the separate Worksheets are now visible in the component: Admin and Development.

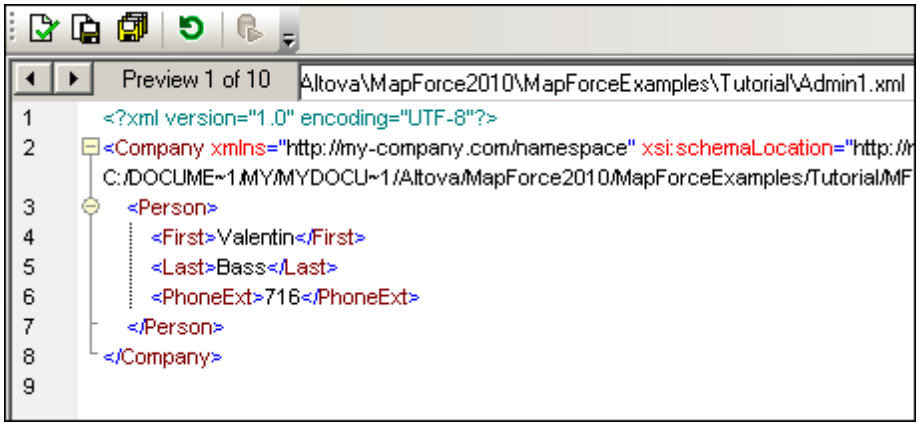


2. Insert the concat and constant components as shown.
3. Insert a second XML Schema file of the same name and create the connections as shown.

   As the "Row number" element determines the specific person rows in the worksheet, this is the item we will use to split up each worksheet into separate files.

For the **Admin** worksheet item in the Altova XLSX component:

1. Create the connections as shown above: Admin constant to value1, **Row Number** to value2, and .xml constant to value3.
2. Connect the **result** parameter of the concat function to the **File:** item of the target component. Note that File: <dynamic> is now displayed.
3. Define the remaining connections as needed.
4. Click the Output tab to see the result of the mapping.



Each row is now visible in its own Preview tab, the first one is shown above. All 10 records of the Admin worksheet have been split into separate files.

Clicking the Next/Previous arrows allows you to see each of the files in the Output tab.

Notes:

- The Admin **constant** supplies the first part of the file name e.g. Admin.
- The **Row Number** item supplies the row number from the Excel worksheet e.g. 1.
- The .xml **constant** supplies the file name extension used when saving the file which is Admin1.xml, as shown above.
- Clicking the Save All icon allows you to save the individual files directly from the Output tab, without having to generate code.

To see the output of the Development worksheet, click the Preview button of that component, then click the Output tab to see the result of the mapping. 11 records have been split into separate files.

5.5 Supplying Parameters to the Mapping

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

You can use simple input components in any the following MapForce transformation languages:

- BUILT-IN (when you preview the mapping transformation directly in MapForce, from the **Preview** tab)
- BUILT-IN (when you run a compiled MapForce Server execution file)
- XSLT 1.0, XSLT 2.0
- XQuery
- C++
- C#
- Java

In case of mappings executed with MapForce Server or by means of generated code, simple input components become command line parameters. In case of mappings generated as XSLT transformations, simple input components correspond to stylesheet parameters in the generated XSLT file.

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

Input parameters added on the main mapping area should not be confused with input parameters in user-defined functions (see User-defined functions). There are some similarities and differences between the two, as follows.

Input parameters on the mapping	Input parameters of user-defined functions
Added from **Function | Insert Input** menu.	Added from **Function | Insert Input** menu.
Can have simple data types (string, integer, and so on).	Can have simple as well as complex data types.
Applicable to the entire mapping.	Applicable only in the context of the function in which they were defined.

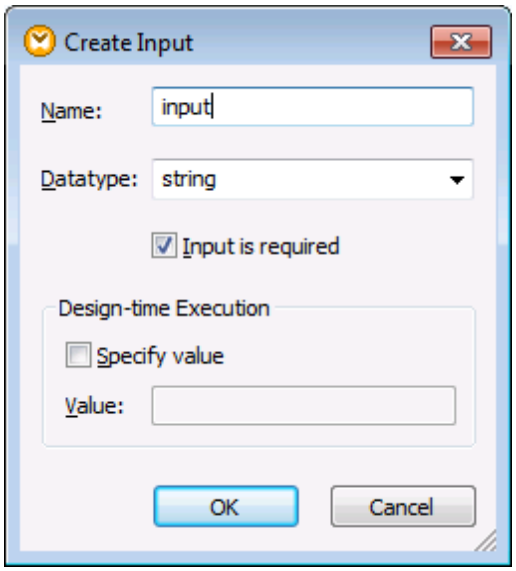
When you create a reversed mapping (using the menu command **Tools | Create Reversed Mapping**), a simple input component becomes a simple output component.

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

5.5.1 Adding Simple Input Components

To add a simple input to the mapping:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. On the **Function** menu, click **Input**.
3. Enter a name and select the data type required for this input. If the input should be treated as a mandatory mapping parameter, select the **Input is required** check box. For a complete list of settings, see Simple Input Component Settings.
4. Click **OK**.

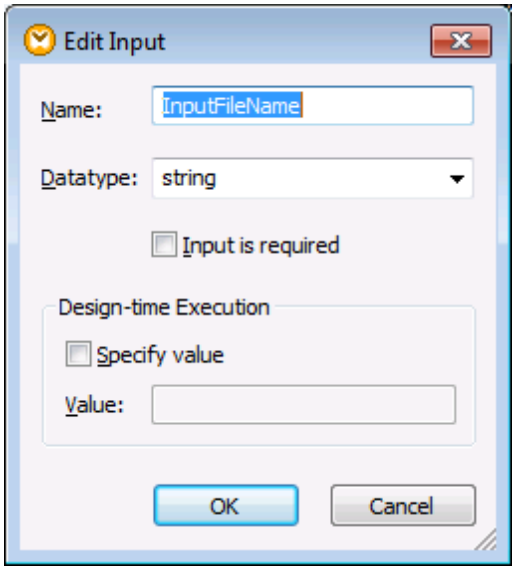


Create Input dialog box

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

5.5.2 Simple Input Component Settings

You can define the settings applicable to a simple input component either when adding it to the mapping area. You can also change the setting at a later time, from the Edit Input dialog box.



Edit Input dialog box

To open the Edit Input dialog box, do one of the following:

- Select the component, and, on the **Component** menu, click **Properties**.
- Double-click the component.
- Right-click the component, and then click **Properties**.

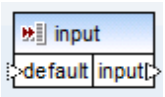
The available settings are as follows.

Name	Enter a descriptive name for the input parameter corresponding to this component. At mapping execution time, the value entered in this text box becomes the name of the parameter supplied to the mapping; therefore, no spaces or special characters are allowed.
Datatype	By default, any input parameter is treated as string data type. If the parameter should have a different data type, select the respective value from the list. When the mapping is executed, MapForce casts the input parameter to the data type selected here.
Input is required	When enabled, this setting makes the input parameter mandatory (that is, the mapping cannot be executed unless you supply a parameter value). Disable this check box if you want to specify a default value for the input parameter (see Creating a Default Input Value).
Specify value	This setting is applicable only if you execute the mapping during design time, by clicking the **Preview** tab. It allows you to enter directly in the component the value to use as mapping input.
Value	This setting is applicable only if you execute the mapping during design time, by clicking the **Preview** tab. To enter a value to be used by MapForce as mapping input, select the Specify Value check box, and then

	type the required value.

5.5.3 Creating a Default Input Value

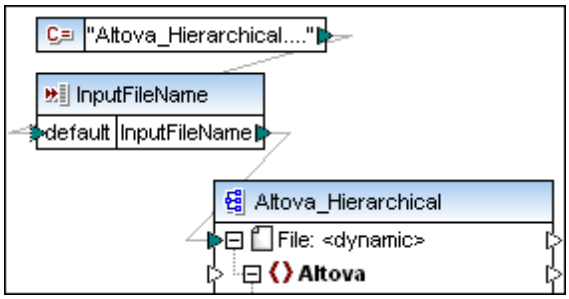
After you add an Input component to the mapping area, notice the **default** item to the left of the component.



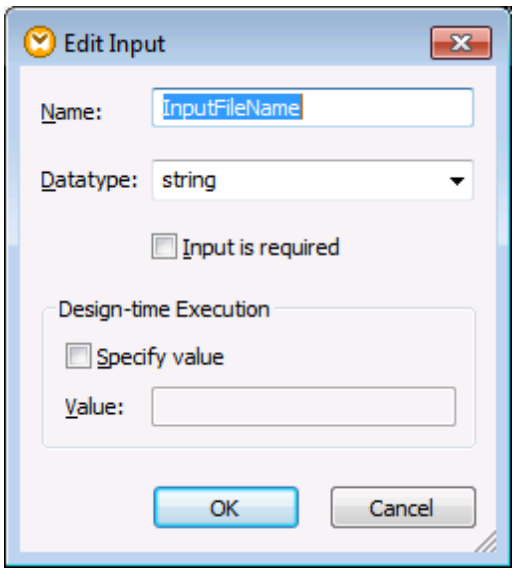
Simple input component

The default item enables you to connect an optional default value to this input component, as follows:

1. Add a constant component (on the **Insert** menu, click **Constant**), and then connect it to the **default** item of the input component.



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



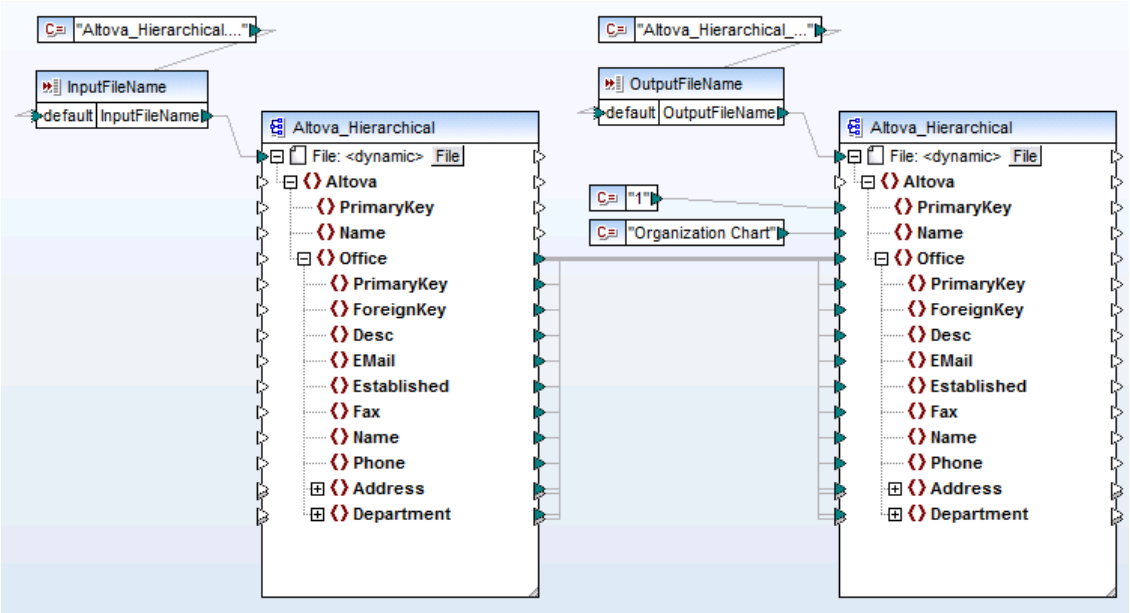
3. Click **OK**.

Note: If you click the **Specify value** check box and enter a value in the adjacent box, the entered value takes precedence over the default value when you preview the mapping (that is, at design-time execution). However, the same value has no effect in the generated code.

5.5.4 Example: Using File Names as Mapping Parameters

This example walks you through the steps required to execute a mapping that takes input parameters at runtime. The mapping design file used in this example is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\FileNamesAsParameters.mfd**.

The mapping uses two input components: **InputFileName** and **OutputFileName**. These supply the input file name (and the output file name, respectively) of the source and target XML file. For this reason, they are connected to the **File: <dynamic>** item.



FileNamesAsParameters.mfd (MapForce Basic Edition)

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

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

XSLT 2.0

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

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

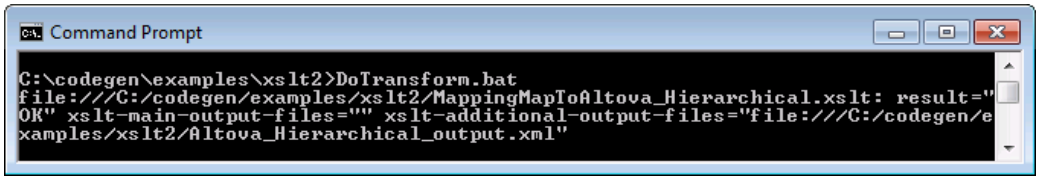
To supply a custom input parameter in the **DoTransform.bat** file:

1. Generate the XSLT 2.0 code (**File | Generate Code In | XSLT 2.0**) from the **FileNamesAsParameters.mfd** sample.
2. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2016 \MapForceExamples** directory to the directory where you generated the XSLT 2.0 code (in this example, **c:\codegen\examples\xslt2**). This file will act as custom parameter.
3. Edit **DoTransform.bat** to include the custom input parameter either before or after %* (as highlighted below). Note that the parameter value is enclosed with single quotes. The available input parameters are listed in the **rem** (Remark) section.

```
@echo off

RaptorXML xslt --xslt-version=2 --
input="MappingMapToAltova_Hierarchical.xslt" --
param=InputFileName:'Altova_Hierarchical.xml' %*
"MappingMapToAltova_Hierarchical.xslt"
rem --param=InputFileName:
rem --param=OutputFileName:
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

When you run the DoTransform.bat file, RaptorXML Server completes the transformation using **Altova_Hierarchical.xml** as input parameter.



MapForce Server Execution File

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

1. Compile the **FileNamesAsParameters.mfd** to a MapForce Server execution file (see Compiling a MapForce mapping). When prompted, save the .mfx execution file to a directory on your computer (in this example, **c:\codegen\examples\mfx**).
2. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2016 \MapForceExamples** directory to the directory where you saved the .mfx file. This file will act as the custom parameter supplied to the mapping execution file.
3. Run MapForce Server with the following command:

```
MapForceServer.exe run "C:\codegen\examples\mfx
\FileNamesAsParameters.mfx" -p=InputFileName:"C:\codegen\examples\mfx
\Altova_Hierarchical.xml" -p=OutputFileName:"C:\codegen\examples\mfx
\OutputFile.xml"
```

In the MapForce Server command above, `-p=InputFileName` and `-p=OutputFileName` are the input parameters to the mapping. You can use any file name as the value of **-OutputFileName**. However, the file name supplied in **-InputFileName** parameter must exist as a physical file; otherwise, the mapping will fail.

Note: If you see the message "MapForceServer.exe is not recognized as an internal or external command, operable program, or batch file", change the current directory to the one where the MapForce Server executable is installed. To avoid changing path every time when you run a mapping, add to your operating system's PATH environment variable the path of the directory where MapForce Server executable is installed (for example, **C:\Program Files (x86)\Altova\MapForceServer2016\bin**) .

Java

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

1. Generate the Java code (**File | Generate Code In | Java**) from the **FileNamesAsParameters.mfd** sample.
2. Compile the Java code into an executable JAR file (for instructions on how to do this in Eclipse, see Example: Build a Java application with Eclipse and Ant).
3. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2016\MapForceExamples** directory to the directory where the .jar file is. This file will act as the custom parameter supplied to the Java mapping application.
4. At the command line, enter: `java -jar Mapping.jar /InputFileName "InputFile.xml"`

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


5.6 Returning String Values from a Mapping

Use a simple output component when you need to return a string value from the mapping. On the mapping area, simple output components play the role of a target component which has a string data type instead of a structure of items and sequences. Consequently, you can create a simple output component instead of (or in addition to) a file-based target component. For example, you can use a simple output component to quickly test and preview the output of a function (see Example: Testing Function Output). This technique is also useful for mappings which use string serialization (see String Parsing and Serialization). The main purpose of a simple output component is, however, to get back a string when calling the MapForce Server API, without writing any files.

Simple output components should not be confused with output parameters of user-defined functions (see User-defined functions). There are some similarities and differences between the two, as follows.

Output components	Output parameters of user-defined functions
Added from **Function | Insert Output** menu.	Added from **Function | Insert Output** menu.
Have "string" as data type.	Can have simple as well as complex data types.
Applicable to the entire mapping.	Applicable only in the context of the function in which they were defined.

If necessary, you can add multiple simple output components to a mapping. You can also use simple output components in combination with file-based and database target components. When your mapping contains multiple target components, you can preview the data returned by a particular component by clicking the **Preview** () button in the component title bar, and then clicking the **Output** tab on the Mapping window.

You can use simple output components as follows in MapForce transformation languages:

Language	How it works
BUILT-IN (when previewing the mapping transformation)	You can preview Output components in the same way as you would preview a file-based mapping output—by clicking the **Output** tab on the Mapping window.
BUILT-IN (when running the MapForce Server execution file)	When you run a compiled MapForce Server execution file (see Compiling a MapForce mapping), the mapping output is returned in the standard output stream (stdout), so you can view it or redirect to a file. For example, assuming that the name of the MapForce server execution file is **MyMapping.mfx**, use the following syntax to redirect the mapping output to **output.txt** file and any errors to the **log.txt** file: `MapForceServer.exe run MyMapping.mfx >output.txt`

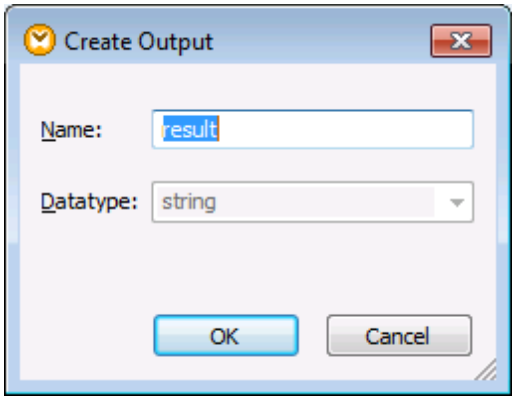
	`2>log.txt`
XSLT 1.0, XSLT 2.0	If the generated XSLT files, a simple output components defined in the mapping becomes the output of the XSLT transformation. If you are using RaptorXML Server, you can instruct RaptorXML Server to write the mapping output to the file passed as value to the `--output` parameter. To write the output to a file, add or edit to the `--output` parameter in the **DoTransform.bat** file. For example, the following **DoTransform.bat** file has been edited to write the mapping output to the **Output.txt** file (see highlighted text). `RaptorXML xslt --xslt-version=2 --input="MappingMapToResult1.xslt" --output="Output.txt" %* "MappingMapToResult1.xslt"` If an `--output` parameter is not defined, the mapping output will be written to the standard output stream (stdout) when the mapping is executed.
C++, C#, Java	In the generated C++, C#, and Java code, the mapping output is written to the standard output of the generated application. If the mapping contains multiple target components, the generated application concatenates the standard output of each target component and returns it as one unified standard output.

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

5.6.1 Adding Simple Output Components

To add an Output component to the mapping area:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. On the **Function** menu, click **Output**.
3. Enter a name for the component.
4. Click **OK**.



Create Output dialog box

You can change the component name at any time later, in one of the following ways:

- Select the component, and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

5.6.2 Example: Previewing Function Output

This example illustrates how to preview the output returned by MapForce functions with the help of simple output components. You will make the most of this example if you already have a basic understanding of functions in general, and of MapForce functions in particular. If you are new to MapForce functions, you may want to refer to Using Functions before continuing.

Our aim is to add a number of functions to the mapping area, and learn how to preview their output with the help of simple output components. In particular, the example uses a few simple functions available in the core library. Here is a summary of their usage:

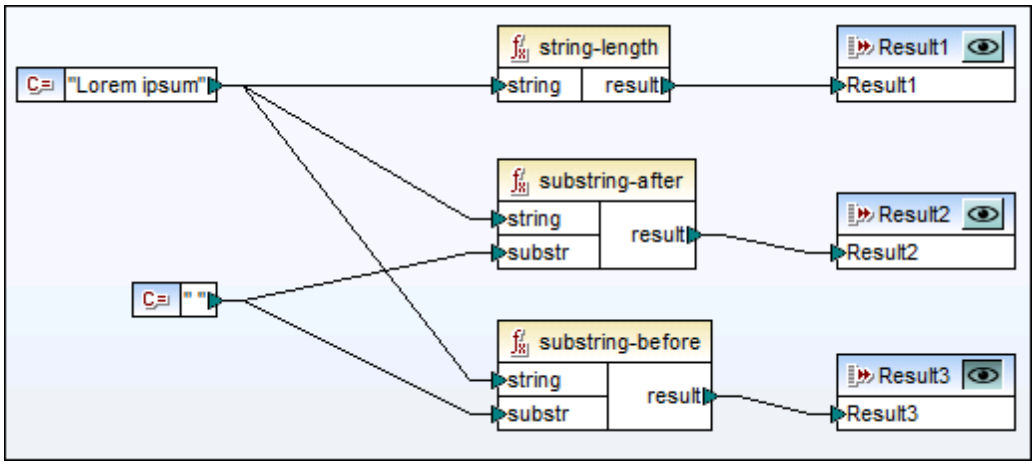
`string-length`	Returns the number of characters in the string provided as argument. For example, if you pass to this function the value "Lorem ipsum", the result is "11", since this is the number of characters that the text "Lorem ipsum" takes.
`substring-after`	Returns the part of the string that occurs after the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "ipsum".
`substring-before`	Returns the part of the string that occurs before the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "Lorem".

To test each of these functions against a custom text value ("Lorem ipsum", in this example), follow the steps below:

1. Add a constant with the value "Lorem ipsum" to the mapping area (use the menu command **Insert | Constant**). The constant will be the input parameter for each of the

functions to be tested.

2. Add the **string-length**, **substring-after**, and **substring-before** functions to the mapping area, by dragging them to the mapping area from the core library, **string functions** section.
3. Add a constant with an empty space (" ") as value. This will be the separator parameter required by the **substring-after** and **substring-before** functions.
4. Add three simple output components (use the menu command **Function | Insert Output**). In this example, they have been named *Result1*, *Result2*, and *Result3*, although you can give them another title.
5. Connect the components as illustrated below.



Testing function output with simple output components

As shown in the sample above, the "Lorem ipsum" string acts as input parameter to each of the **string-length**, **substring-after**, and **substring-before** functions. In addition to this, the the **substring-after** and **substring-before** functions take a space value as second input parameter. The *Result1*, *Result2*, and *Result3* components can be used to preview the result of each function.

To preview the output of any function

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

5.7 Using Variables

Intermediate variables are a special type of component used to solve various advanced mapping problems. They store an intermediate mapping result for further processing.

- Variables work in all languages except XSLT1.0.
- Variable results are always sequences, i.e. a delimited list of values, and can also be used to create sequences.
- Variables are structural components, with a root node, and do not have instances (XML files etc.) associated to them.
- Variables make it possible to compare items of one sequence, to other items within the same sequence.
- Variables can be used to build intermediate sequences. Records can be filtered before passing them on to a target, or filtered after the variable by using the position function for example.

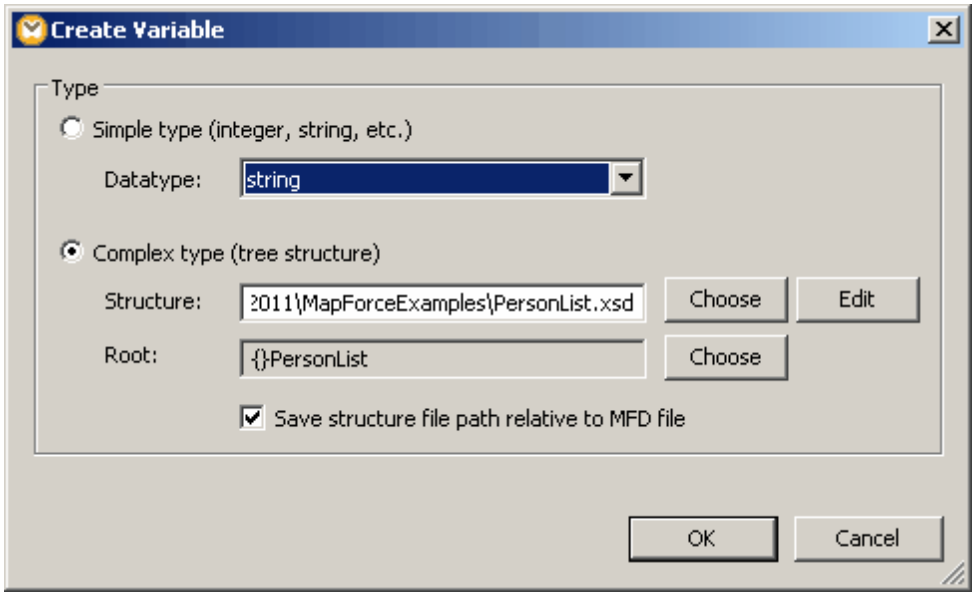
The following table outlines differences between variables and chained mappings.

Chained mappings	Variables
Involve two totally independent steps.	Evaluated depending on context / scope. Controlled by "compute-when" connection
Intermediate results are stored externally in XML files when mapping is executed.	Intermediate results are stored internally, not in any physical files, when mapping is executed.
Intermediate result can be previewed using preview button.	Variable result cannot be previewed.

Inserting intermediate variables

There are several ways of inserting intermediate variables: Using the menu option, by clicking the Var. icon, or by right clicking input/output icons and creating variables based on the input/output components.

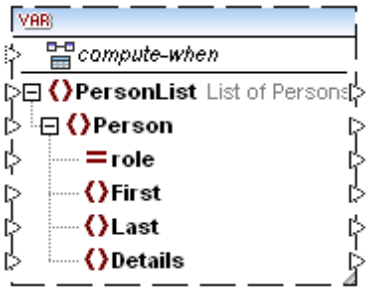
1. Select **Insert | Variable** or click the Variable icon  in the icon bar. You can now select if you want to insert a simple or complex variable.



2. Click the radio button for the type of variable you want to insert, i.e. Simple type, or Complex type.

   If you clicked the "Complex type" radio button:
3. Click the "Choose" button to select XML Schema for example, and select the Root item from the next dialog box.
4. Click OK to insert the variable.

Complex variable:

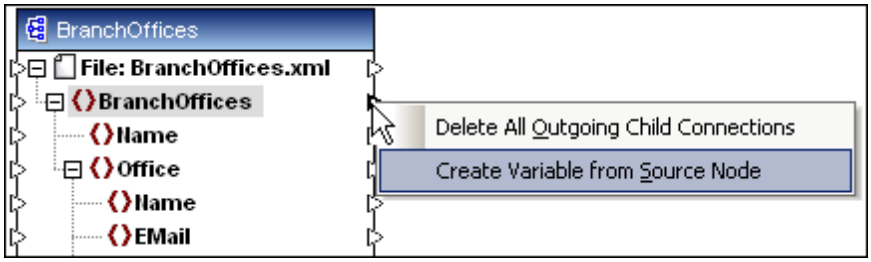


Simple variable:

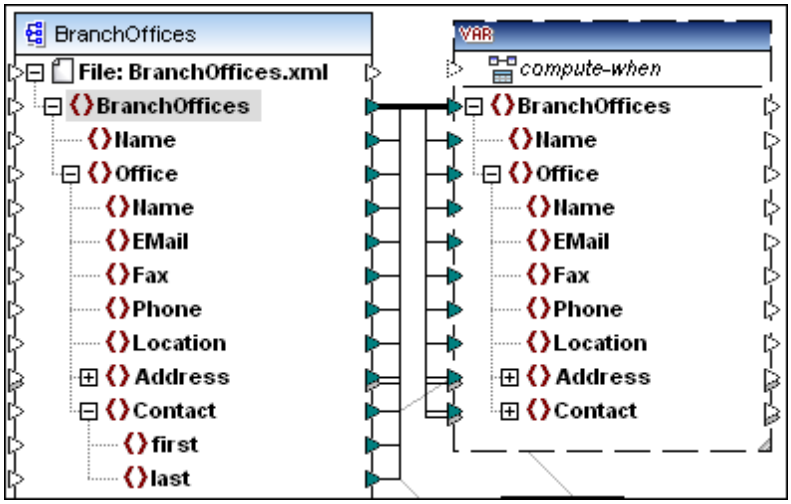
Have a single mappable item/value e.g. string, integer. Note that the "value" item can be duplicated.

Alternate methods of inserting variables:

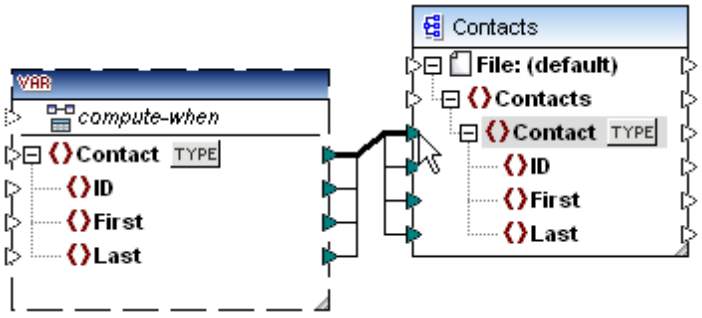
- Right click an **output** icon of a component (e.g. BranchOffices) and select "Create Variable from Source node".



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

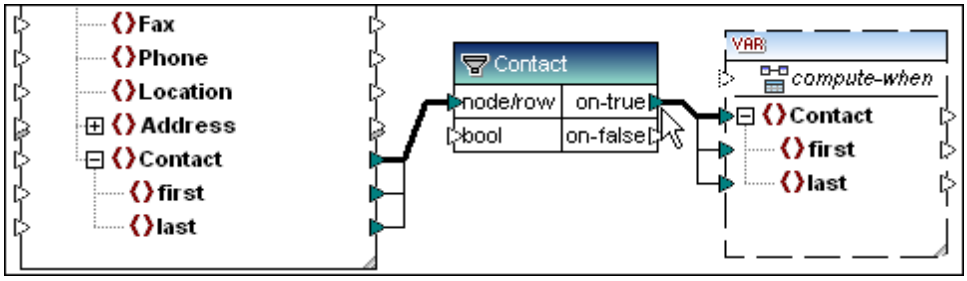


- Right click an **input** icon of a target component (e.g. Contact) and select "Create Variable for Target Node".



This creates a complex variable using the same schema as the target, with the Contact item as the root node, and automatically connects all items with a copy-all connection.

- Right click an **output** icon of a filter component (on-true/on-false) and select "Create Variable from Source node".



This creates a complex component using the source schema, and automatically uses the item linked to the filter input node/row, i.e. Contact, as the root element of the intermediate component.

Compute-when 

The compute-when input item allows you to control the **scope** of the variable; in other words when and how often the variable value is computed when the mapping is executed. You do not have to connect this input in many cases, but it can be essential to override the default context, or to optimize mapping performance.

A **subtree** in the following text means the set of an item/node in a target component and all of its descendants, e.g. a <Person> element with its <FirstName> and <LastName> child elements.

Variable value means the data that is available at the **output** side of the variable component.

- For simple variables, it is a sequence of atomic values that have the datatype specified in the component properties.
- For complex variables, it is a sequence of root nodes (of the type specified in the component properties), each one including all its descendant nodes.

The sequence of atomic values (or nodes) may contain one or even zero elements. This depends on what is connected to the input side of the variable component, and to any parent items in the source and target components.

Compute-when 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 (via a connector from the variable component directly to a node in the target component, or indirectly via functions). The same variable value is also used for all target child nodes inside the subtree.

The actual variable value depends on any connections between parent items of the source and target components.

This default behavior is the same as that of complex outputs of regular user-defined functions and Web service function calls.

If the variable output is connected to multiple **unrelated** target nodes, the variable value is computed separately for each of them. This can produce different results in each case, because different parent connections influence the context in which the variable's value is evaluated.

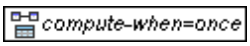
Compute-when - connected

By connecting compute-when to an **output node** of a source component, 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, i.e. at runtime the variable is re-evaluated whenever a new item is read from the sequence in the source component.

This relates to the general rule governing connections in MapForce - "for each source item, create one target item". With compute-when, it means "for each source item, compute the variable value".

Compute-once 

Right clicking the "compute-when" icon and selecting "Compute Once" from the context menu changes the icon to "compute-when=once" and also removes the input icon.

This setting causes the variable value to be computed once **before** each of the target components, making the variable essentially a global constant for the rest of the mapping.

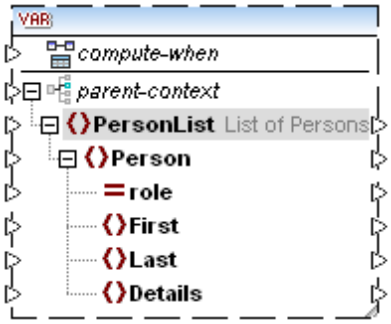
In a user-defined function, the variable is evaluated each time the function is called, before the actual function result is evaluated.

Parent-context

The main use of adding a parent-context is when using multiple filters and you need an additional parent node to iterate over.

To add a parent-context to a variable:

- Right click the root node, e.g. PersonList and select "Add Parent Context" from the context menu.
  This adds a new node, "parent-context", to the existing hierarchy.

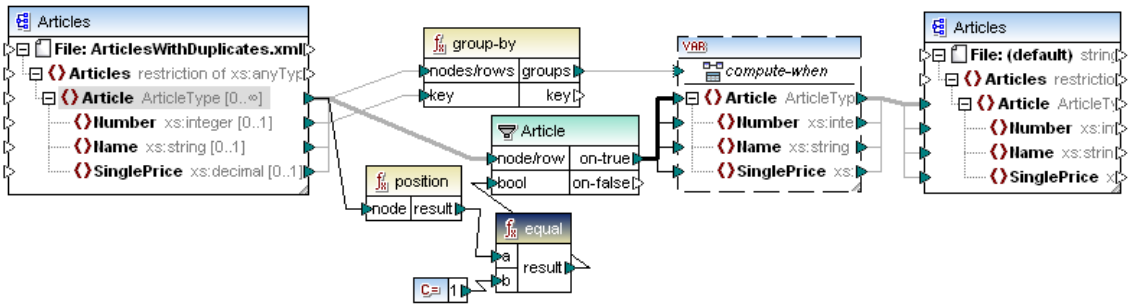


The parent context adds a virtual "parent" node to the hierarchy within the component. This allows you to iterate over an additional node in the same, or different source component.

5.7.1 Variables - use cases

Filtering out multiple instances of the same record from a source instance

Source data can often contain multiple instances of the same record. In most cases you would only want one of these records to be mapped to the target component. The example below is available as **DistinctArticles.mfd** in the ...\MapForceExamples folder.



The ArticlesWithDuplicates.xml file contains two articles both having the same article number (two with article no. 1 and two with article no 3).

Articles

xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespac...	Articles.xsd

Article (6)

	Number	Name	SinglePrice
1	1	T-Shirt	25
2	1	T-Shirt Duplicate	25
3	2	Socks	2.30
4	3	Pants	34
5	4	Jacket	57.50
6	3	Pants Duplicate	34

The article Number is used as the key in the **group-by** function, so it creates one group per unique article number. Each group thus contains one article and all the unwanted duplicates of that article. The next step is to extract the first item of each group and discard the rest.

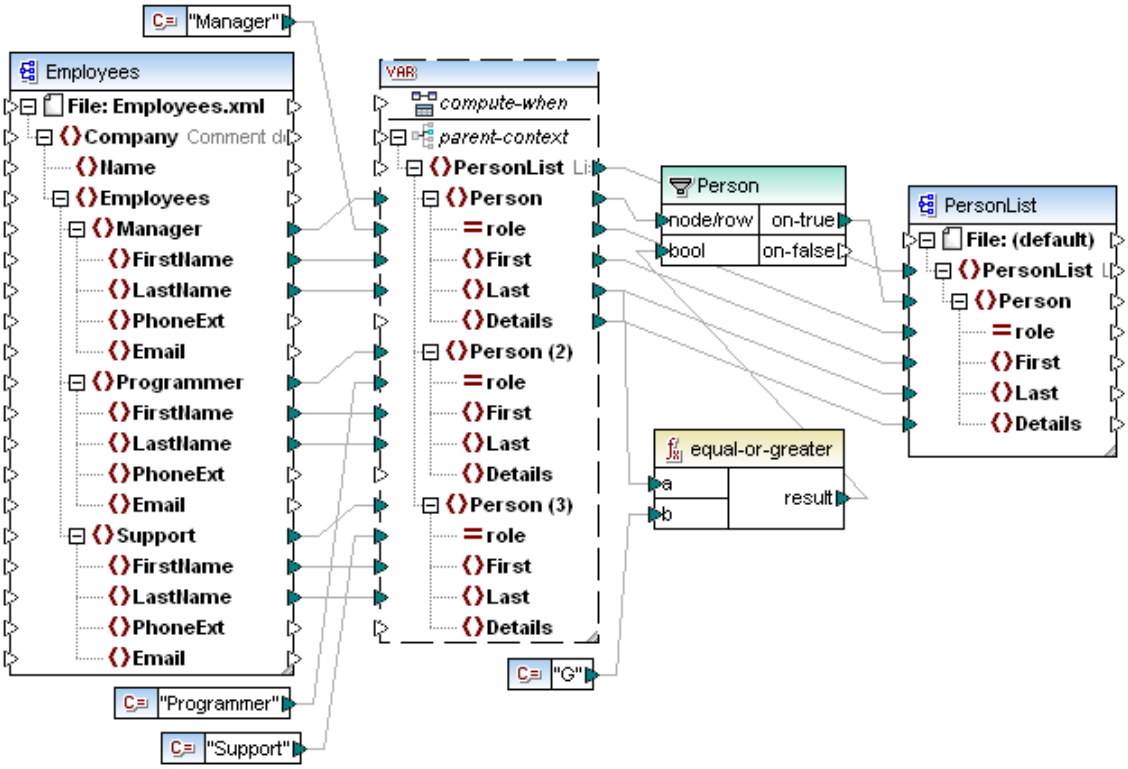
The connection of the group output to "compute-when" causes the variable to be evaluated once for each group, in the context of that group. This establishes an additional context level, as if we had connected a parent element of the target Article element.

To select the first article of each group, we use the position function and a filter component, which are connected to the variable input.

Applying filters to intermediate sequences:

Nodes in variable components can be duplicated as in any other type of component. This allows you to build sequences from multiple different sources and then further process the sequence.

The screenshot below shows how PersonList.mfd could be modified using an intermediate variable, and how constant components can also act as source items.

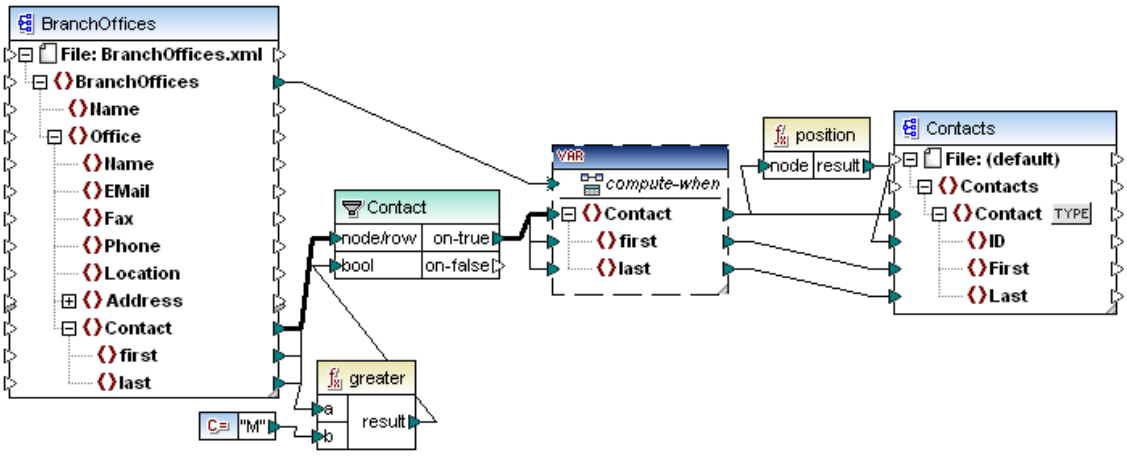


The Person node of the **variable** has been duplicated twice, and a filter has been added to filter those persons whose Last name starts with a letter higher than "G".

Numbering nodes of a filtered sequence

This example is available as **PositionInFilteredSequence.mfd** in ...\MapForceExamples folder and uses the variable to collect the filtered contacts where the last name starts with a letter higher than M.

The contacts are then passed on (from the variable) to the target component, with the position function numbering these contacts sequentially.

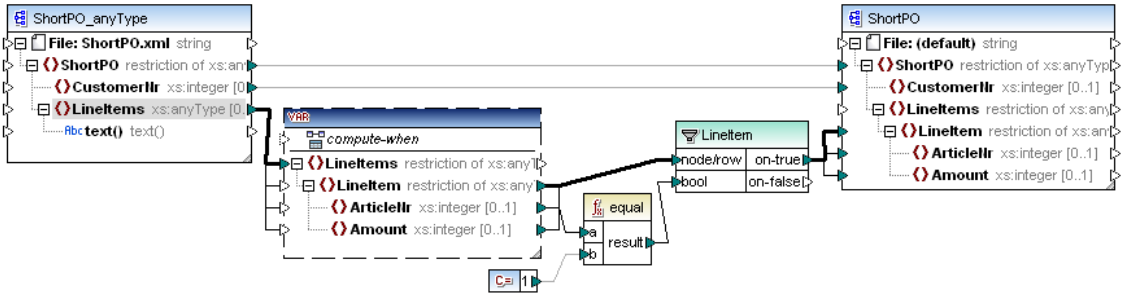


The variable acts like another source component allowing the position function to process the

filtered sequence.

Extract specific data from a source components' anyType node

This example consists of a source component containing anyType elements from which we want to filter specific data.



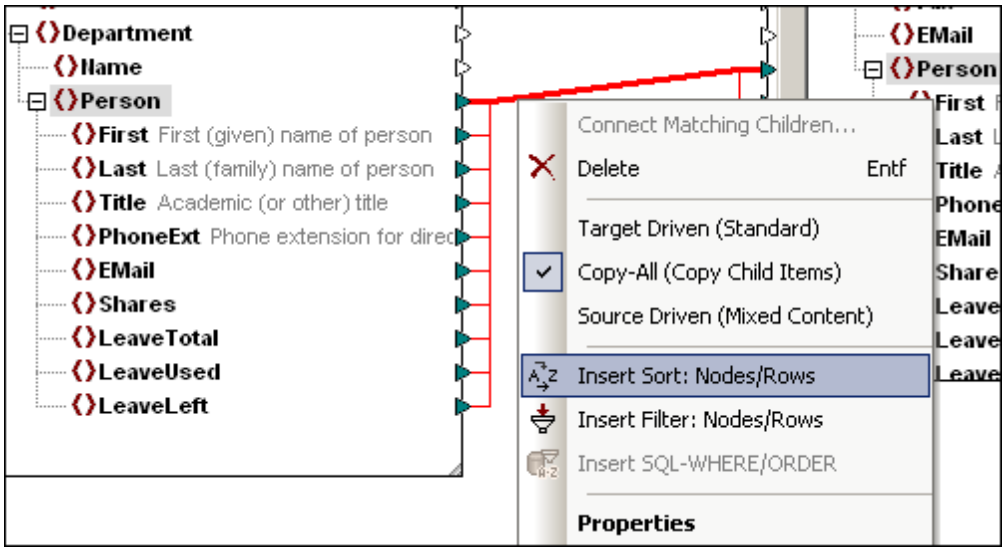
The intermediate variable is based on a schema that has nodes of the specific type of data that we want to map, i.e. ArticleNr and Amount are both of type integer. That specific data is filtered by ArticleNr. and passed on to the target component.

5.8 Sorting Data

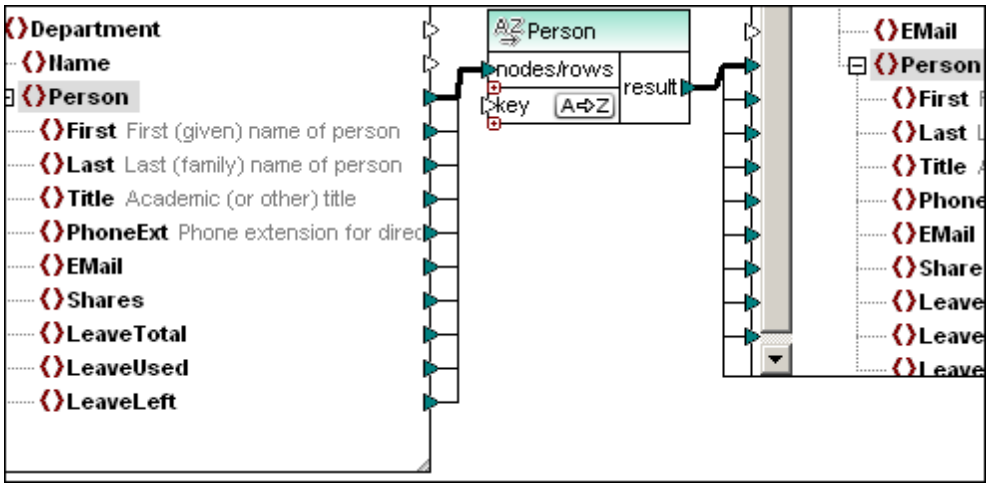
To sort input data based on a specific sort key, use the **Sort** component. The sort component currently supports the targets: XSLT2, XQuery, and the Built-in execution engine.

To insert a Sort component:

1. Right click a connector that exists between the nodes that you want to sort.



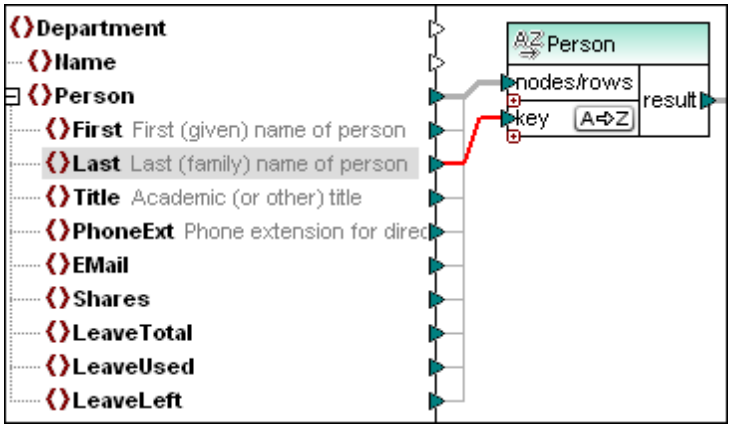
2. Click the **Insert Sort: Nodes/Rows** item from the context menu.



This inserts, and automatically connects, the sort component to the source and target components.

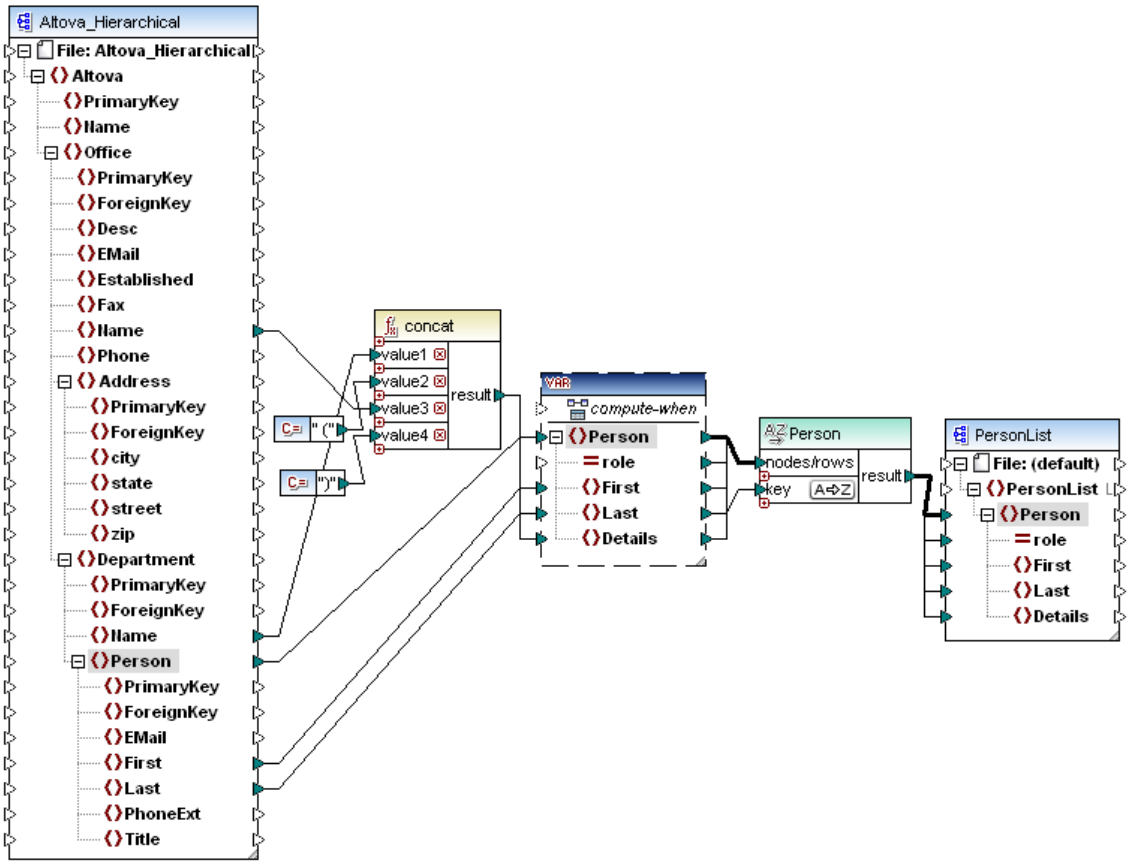
To define which item you want to sort by:

- Connect the item you want to sort by, e.g. **Last**, to the **key** parameter of the sort component, now named "Person".



The Persons wil now be sorted by Last in the output tab.

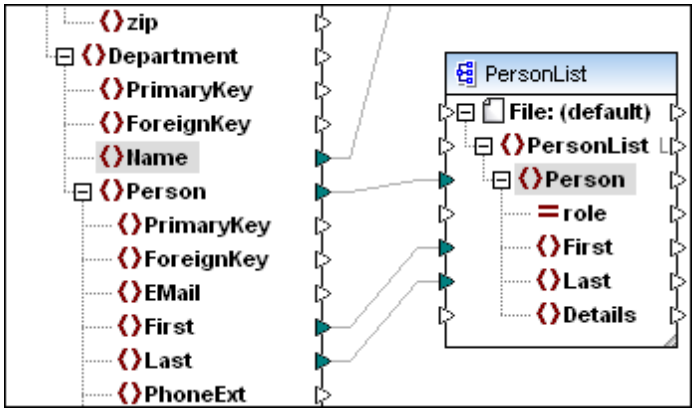
Example: **Altova_Hierarchical_Sort.mfd** in the MapForceExamples folder.



The aim is to have the persons of each of the branch offices alphabetically sorted, and also include detailed information on the office and department names. This example makes use of the Variable component which allows access to otherwise unavailable parent items. In this case parent items of the Person node.

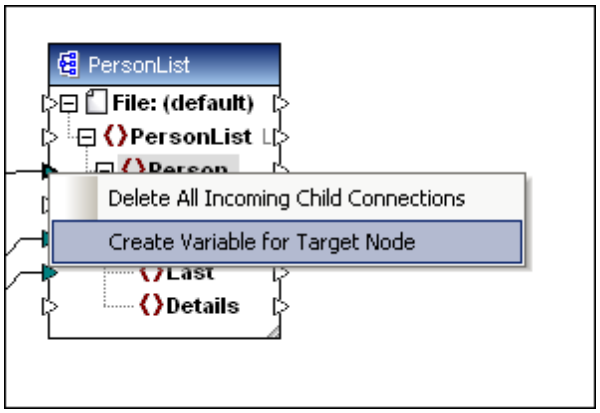
How this mapping was created:

The initial stage of the mapping is shown below.

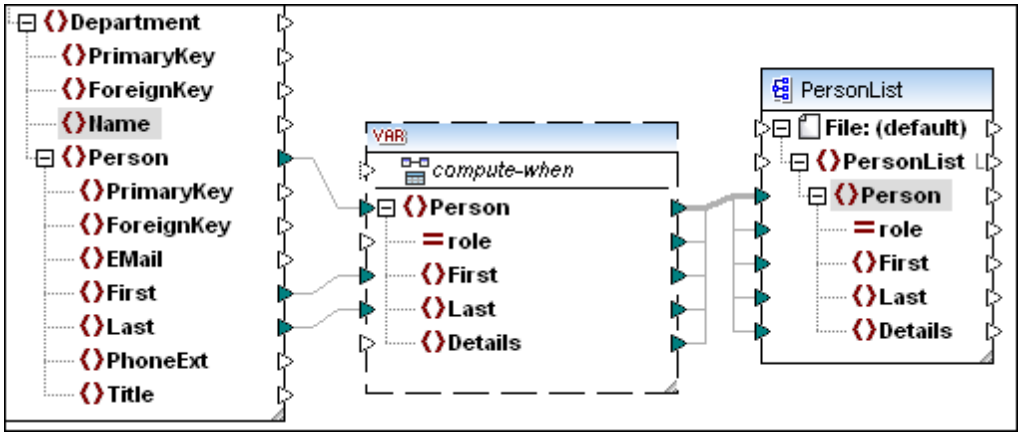


As we want to sort a Person item, the Person items in both components are connected, which also connects the identically named child items.

1. Right click the input icon (exactly on the triangle) next to the Person item of the target component.
2. Select the "Create Variable for Target Node" item in the context menu.

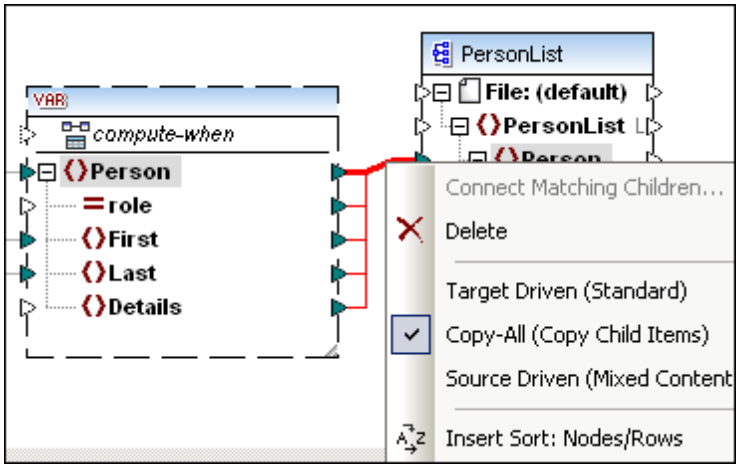


This inserts a complex Intermediate variable between the components and connects the identical items. See Intermediate variables for more specific information.

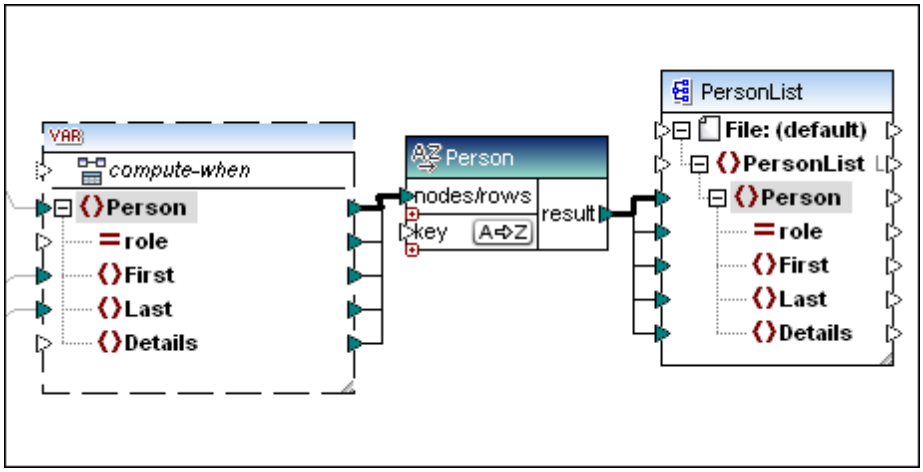


As we want the persons to be sorted by their last names, we now need to insert the sort component.

3. Right click the Copy-All connector and select Insert Sort: Nodes/Rows.

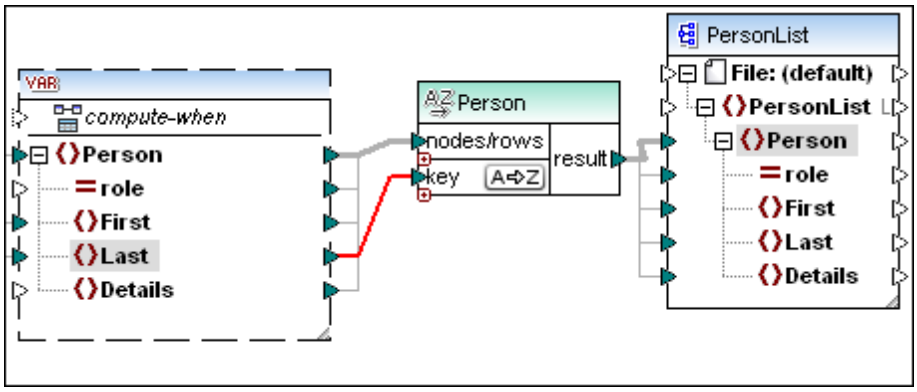


This inserts the sort component and automatically connects all the relevant items.



The only thing left to do is to define the key that we want to sort by.

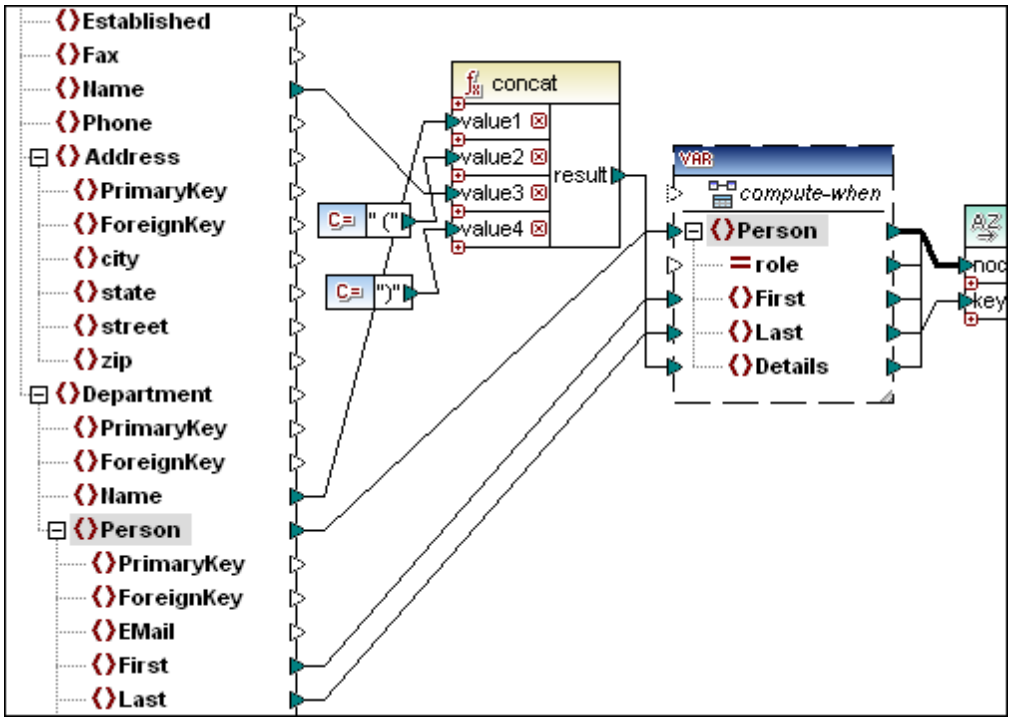
4. Connect the Last item of the intermediate variable to the **key** parameter of the sort component.



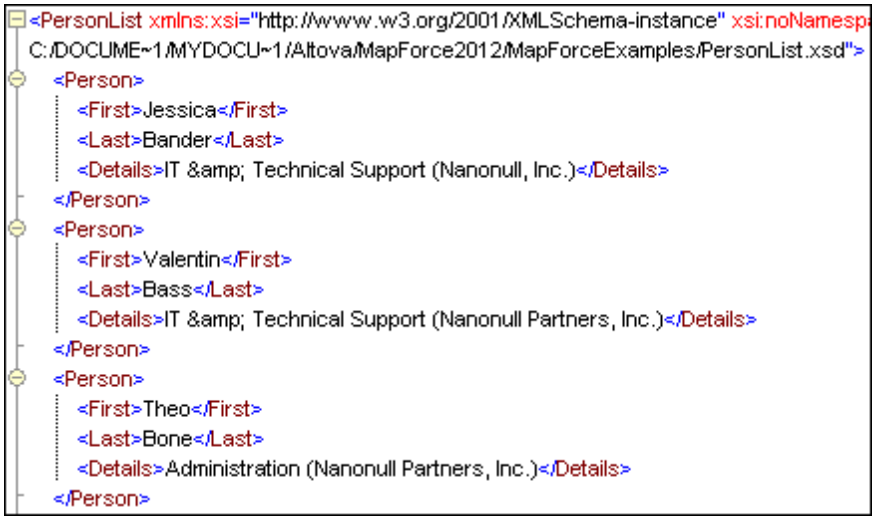
5. Click the Output button to see a preview of the result.
   The persons are now sorted by last name. Click the Mapping button to return to the design window.
6. Using the concat function, and the constant components (to supply the parenthesis)

connect up the other items as shown below.

7. Connect the result parameter of the concat function, to the Details item of the intermediate variable.



8. Click the Ouput button to see the result.



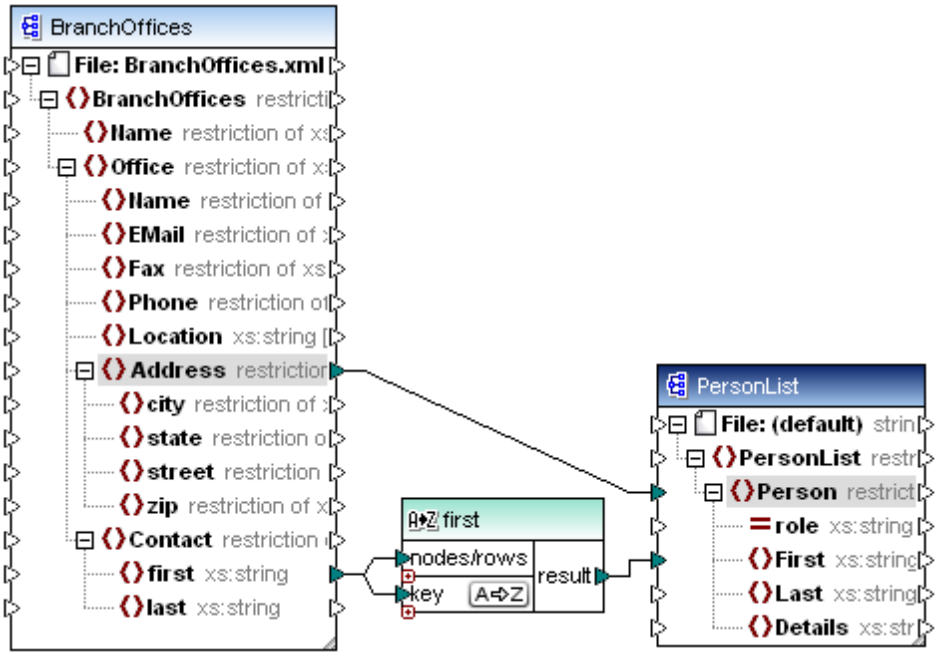
The persons are still sorted by last name but additional info, supplied by the details field, has been added to each person. The correct office and department names are now available to each person, because the intermediate variable makes it possible to access parent data from a child node. This is only possible by using the intermediate variable.

To reverse the sort sequence:

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

To sort input data consisting of simple type items:

- Connect the simple content item, e.g. first, to **both** the nodes/row and key parameters of the sort component.

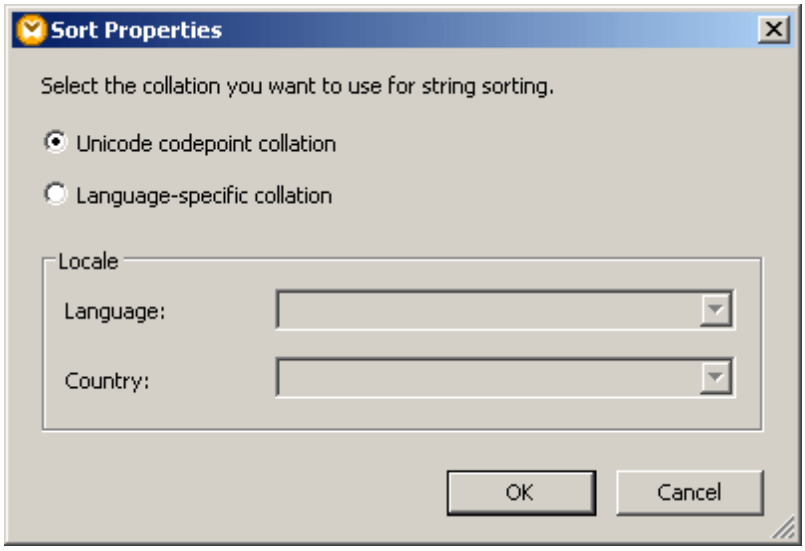


Sorting database data - target Built-in (execution engine)

The Sort component can also be used to sort table data. Better performance is however achieved using the SQL-WHERE/ORDER component.

To sort strings using language-specifc rules:

Double click the title bar/header of the inserted Sort component to open the Sort Properties dialog box.



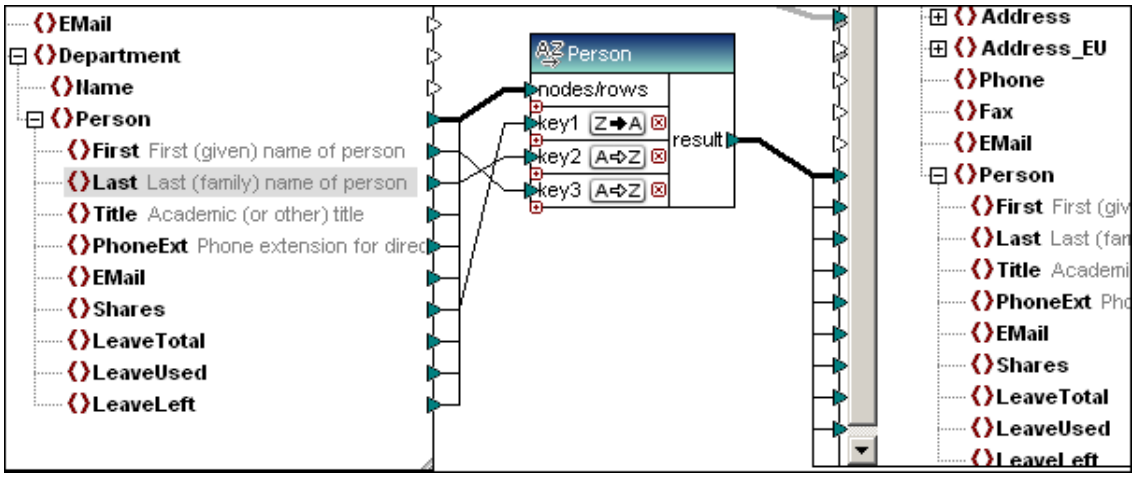
Unicode code point collation: This (default) option compares/orders strings based on code point values. Code point values are integers that have been assigned to abstract characters in the Universal Character Set adopted by the Unicode Consortium. This option allows sorting across many languages and 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 BUILTIN execution engine, and for XSLT support depends on the specific engine used to execute the code.

To sort by multiple keys:

The sort component allows you to define multiple keys.

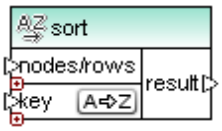
- Clicking the "+" icon adds a new key to the sort component, i.e. key2
- Clicking the "x" icon deletes a that specific key.
- Dropping a connector onto a + icon automatically inserts/adds the parameter and connects to it.



Note that key1 has a higher sort priority than key2, and key2 higher than key3.

To insert a Sort component conventionally:

- Click the Sort icon  in the icon bar to insert the component
  This inserts the sort component in its "unconnected" form where "sort" is visible in the title bar of the component.



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.

5.9 Filtering Data

The "filter" component is very important when querying database data, as it allows you to work on large amounts of data efficiently. When working with database tables containing thousands of rows, filters reduce table access and efficiently structure the way data is extracted. The way filters are used, directly affects the speed of the mapping generation.

This section will deal with methods enabling you to optimize data access and generally speed up the mapping process.

In general, use as few filter components as possible, and:

1. Avoid concatenating filter-components
2. Connect the "on-true/on-false" parameters, to parent items if possible, instead of child items directly
3. Connect the "on-false" parameter to map the complement node set, delivered by the on-true parameter
4. Don't use filters to map to child data, if the parent item is mapped
5. Use the "Priority context" to prioritize execution of unrelated items

Avoid concatenating filter components
Every filter-component leads to a loop through the source data, thus accessing the source **n** times. When you concatenate two filters, it loops **n*n** times.

Solution:
Use "**logical-and**" components to combine the boolean expressions of two filter-components. The result is a single filter component looping only **n**-times.

Connect the "on-true/on-false" parameter of the filter component, to target parent items
Filter components work best when they are connected to parent items containing child items, instead of individual items directly.

The filter **boolean** expression is therefore evaluated against the parent, **before** looping through the child elements. Using filters mapped from a database table will generate:

- "SELECT * FROM table WHERE <expression>" if the **parent** item is mapped, or
- "SELECT * FROM table", and then evaluate for each row, if child items are mapped

Please note:
when connecting a filter from a source parent item, it is also necessary to connect the on-true/on-false parameter to the parent target element. If this cannot be done, then do not apply this rule.

Connect the "on-false" parameter to map the complement node set
Connecting this parameter allows you quick access to the complement node set defined by the current mapping. The same tips apply when using this parameter, connect to parent items etc.

Don't use filters to map to child data, if the parent item is mapped
Using a filter to map data from a source parent to a target parent, automatically applies the **same filter** to **every child** item of the particular parent.

Filter components do not have to be used to supply filtered data to child items, if the parent item

can be mapped! You can therefore map child data directly.

Use priority-context to prioritize execution when mapping unrelated items
Mappings are always executed top-down; if you loop/search through two tables then each loop is processed consecutively. When mapping unrelated elements, without setting the priority context, MapForce does not know which loop needs to be executed first, it therefore automatically selects the first table, or data source.

Solution:
Decide which table, or source data is to be looped/searched first, and then set the priority context on the connector to that table. Please see Priority Context node/item for a more concrete example.

To define a priority context:

- Right click an input icon and select "Priority Context" from the pop-up menu.
  If the option is not available, mapping the remaining input icons of that component will make it accessible.

Filters and source-driven / mixed content mapping

Source-driven mappings only work with direct connections between source and target components. Connections that exist below a source-driven connection, are not taken as source-driven and the items will be handled in target component item/node order.

A single filter where both outputs are connected to same/separate targets, acts as if there were two separate filter components, one having a negated condition.

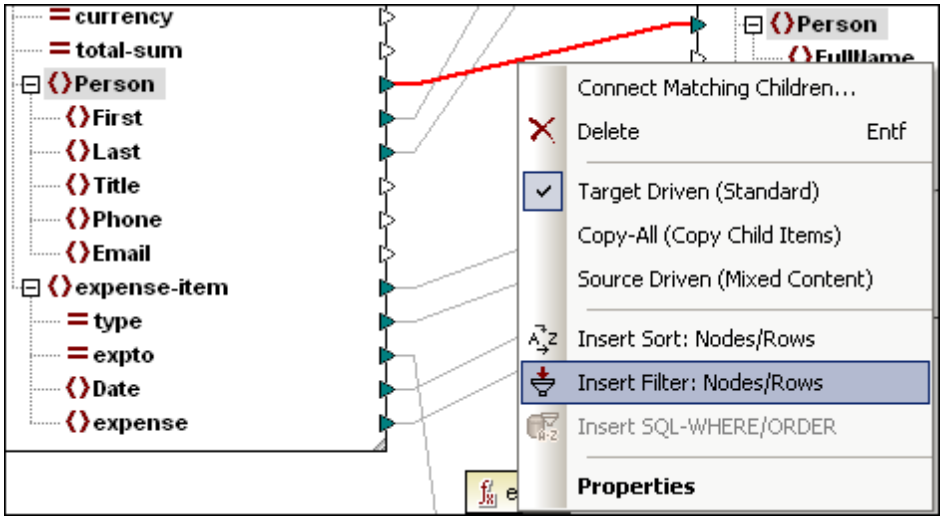
If an exception component is connected to one of the filter outputs, the exception condition is checked when the mappings to the other filter output are executed.

5.9.1 Example: Filter Database Data

Use the **Filter Nodes/Rows** component filter data based on specific criteria.

To insert a Filter component

1. Right click a connector that exists between the nodes/items you want to filter.



2. Click the **Insert Filter: Nodes/Rows** from the context menu.
   This inserts, and automatically connects, the filter component to the source and target components.
3. Define the condition you want to filter by, e.g. PrimaryKey=4, and connect the result to the bool parameter of the filter.

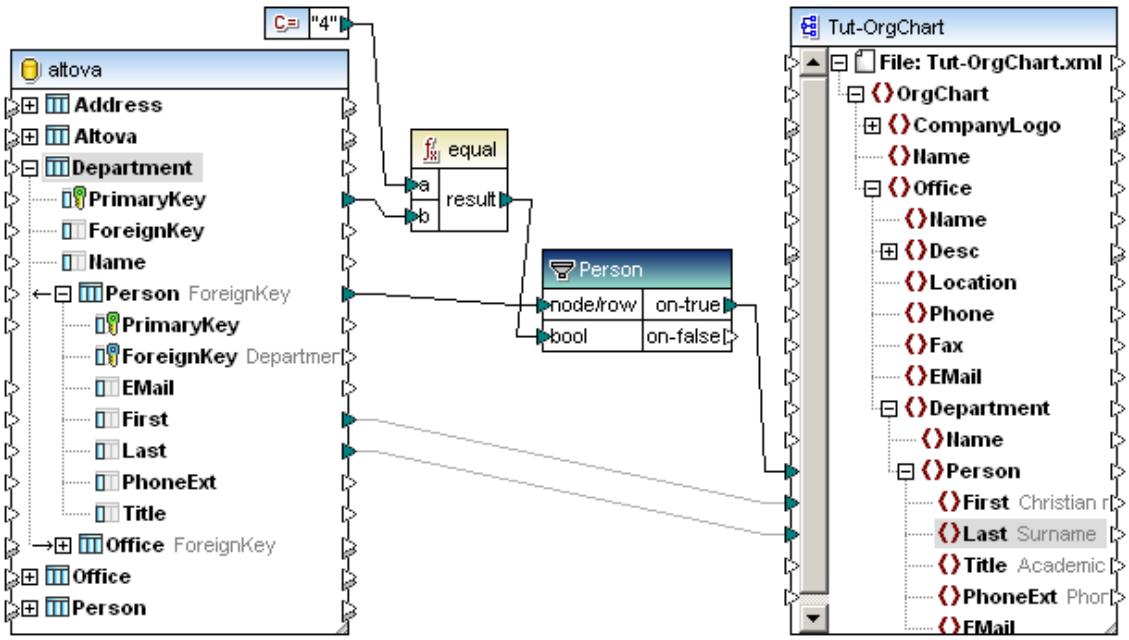
Alternatively:

- Click the filter icon  in the icon bar to insert the component.
  This inserts the filter component in its "unconnected" form where "filter" is visible in the title bar of the component.

  As soon as a connection is made to the source component, the title bar name changes to that of the item connected to the node/row item, in this case to Person.

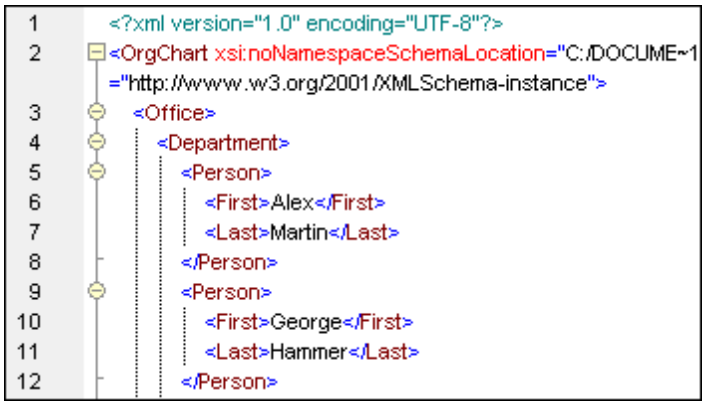
Goal: to map all First/Last names from the **Person** table, where the Department **primary** key is equal to 4.

- The value of the PrimaryKey is compared to the value 4, supplied by the Constant component, using the equal function.
- **PrimaryKey** is mapped from the Department "root" table.
- **First** and **Last** are mapped from the Person table, **which is a child** of the Department "root" table.
- Mappings defined under the **same** "root" table, in this case **Department**, maintain the relationships between their tables.



A filter has two input and output parameters: **node/row** and **bool**, and **on-true**, **on-false**. If the Bool(ean) condition is **true**, then the content of the **child** items/nodes connected to the **node/row** parameter, of the source component, are forwarded to the target component. If the Boolean is false, then the complement values can be used by connecting to the on-false parameter.

The first and last names of the persons in the IT Department with the primary key of 4, are displayed in the Output tab.



For more examples on filtering data, see:

- Filtering XML data: Map Multiple Sources to One Target
- Filtering CSV files by header and detail records: Creating hierarchies from CSV and fixed length text files
- Table relationships and filters: Database relationships and how to preserve or discard them
- Defining SQL-Where filters: SQL WHERE Component / condition

5.10 Using Value-Maps

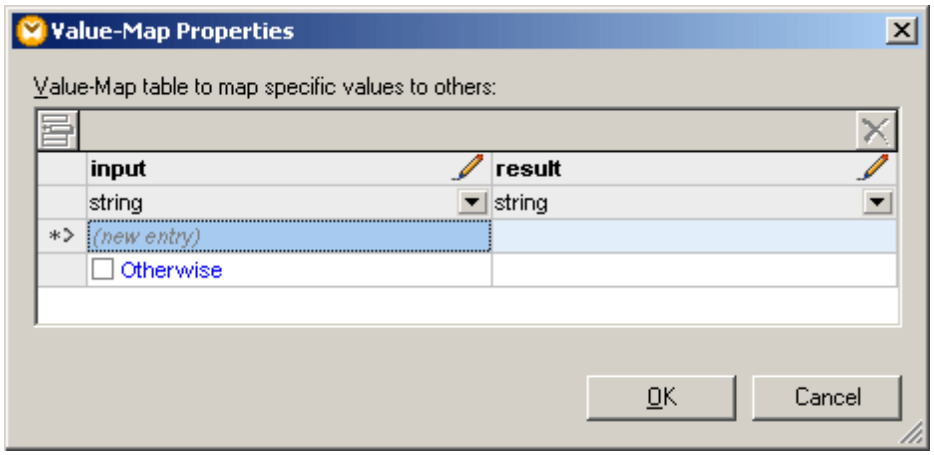
The Value-Map component allows you to transform an input value to a different output value using a lookup table. This is useful for converting different enumeration types. The component only has one input and output item.

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

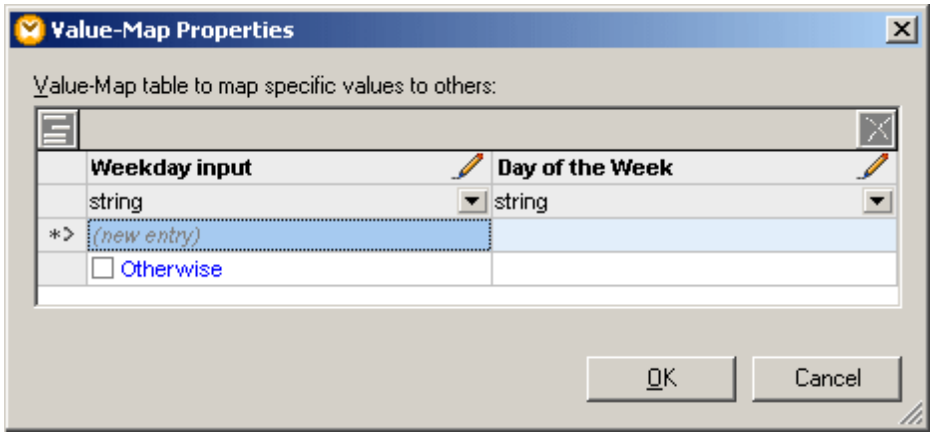
To use a Value-Map component:

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

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



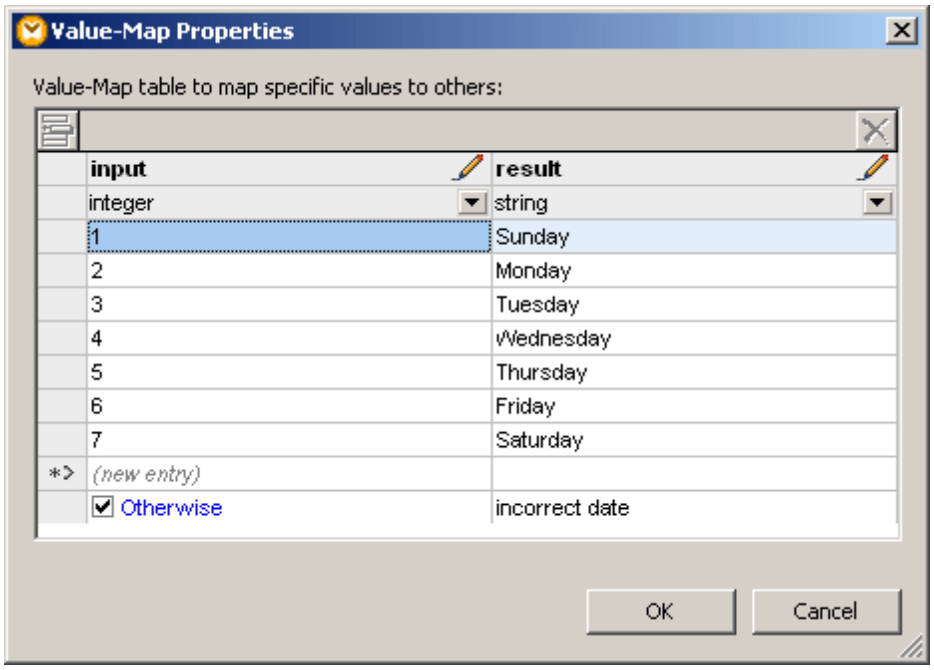
3. Click into the column headers and enter **Weekday input** in the first, and **Day of the Week** in the second.



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

column.

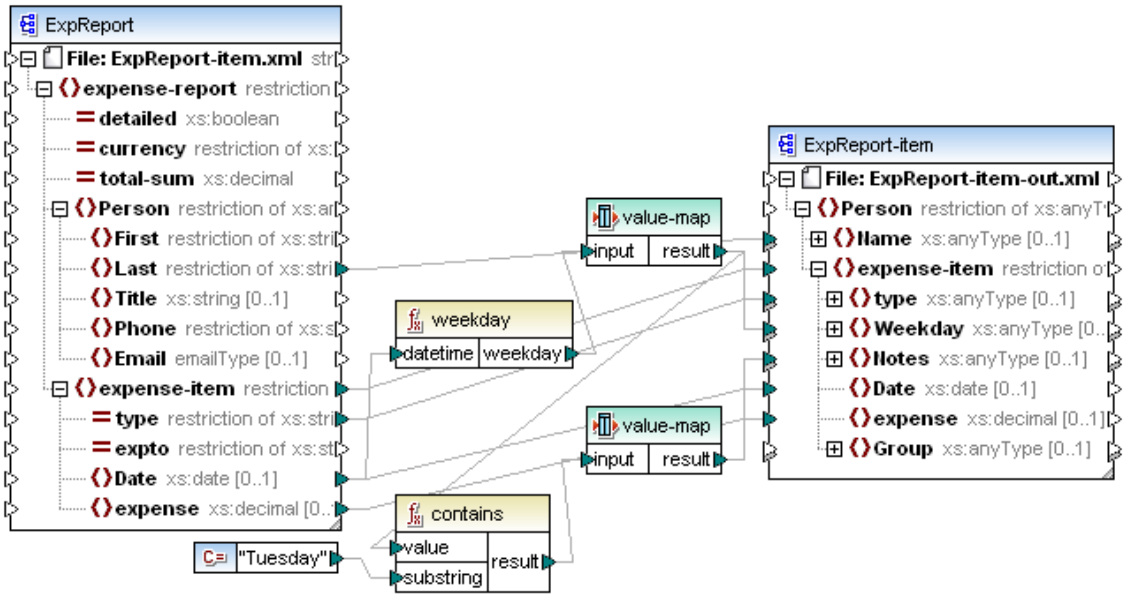
6. Simply type in the ***(new entry)*** input field to enter a new value pair.
7. Click the **datatype** combo box, below the column header to select the input and output datatypes, e.g. integer and string.



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

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

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



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

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

```
3    <Name>Landis</Name>
4    <expense-item>
5      <type>Meal</type>
6      <Weekday>Tuesday</Weekday>
7      <Notes>-- Prepare financial reports -- !</Notes>
8      <Date>2003-01-01</Date>
9      <expense>122.11</expense>
10   </expense-item>
11   <expense-item>
12     <type>Lodging</type>
13     <Weekday>Monday</Weekday>
14     <Notes> --</Notes>
15     <Date>2003-01-14</Date>
16     <expense>122.12</expense>
17   </expense-item>
```

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

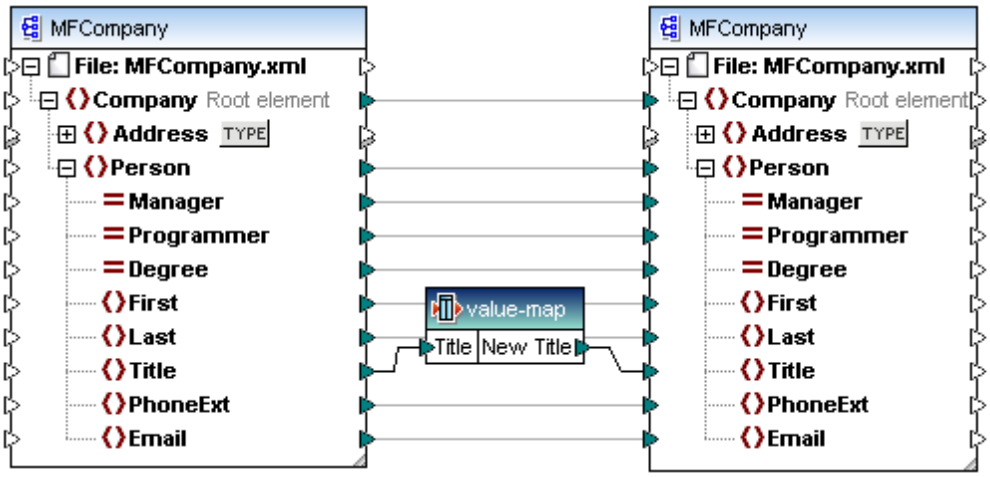
The output from various types of logical, or string functions, can only be a boolean "**true**"

or "**false**" value. The value you want to test for, must thus be entered into the **input** field of the value map table e.g. "true".

5.10.1 Passing data through a Value-Map unchanged

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

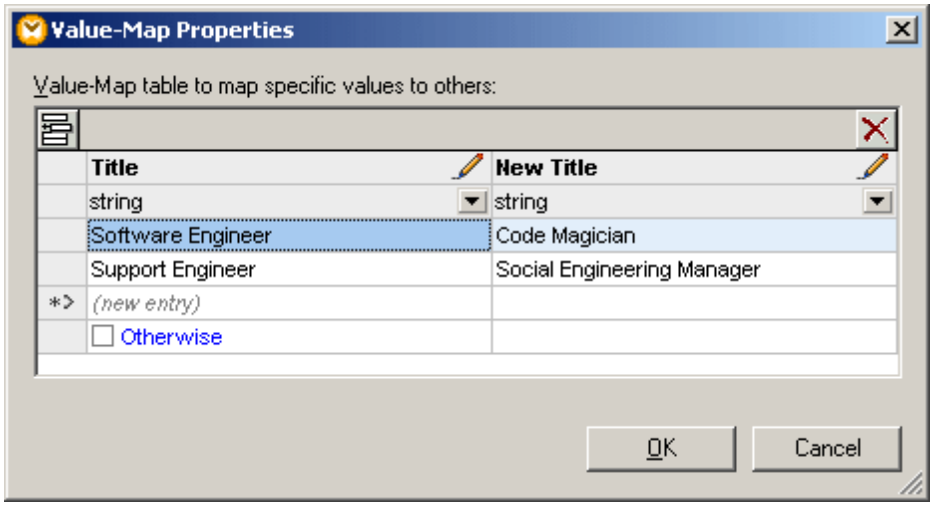
An example of this would be a company that changes some of the titles in a subsidiary. In this case it might change two title designations and want to keep the rest as they currently are.



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

```
33  <Person>
34    <First>Fred</First>
35    <Last>Landis</Last>
36    <PhoneExt>951</PhoneExt>
37    <Email>f.landis@nanonull.com</Email>
38  </Person>
39  <Person>
40    <First>Michelle</First>
41    <Last>Butler</Last>
42    <Title>Code Magician</Title>
43    <PhoneExt>654</PhoneExt>
44    <Email>m.landis@nanonull.com</Email>
45  </Person>
```

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

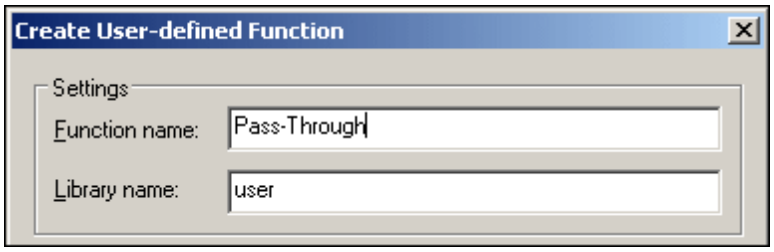


Possible alternative:
Clicking the **Otherwise** check box and entering a substitute term, does make the Title node reappear in the output file, but it now contains the same **New Title** for all other persons of the company.

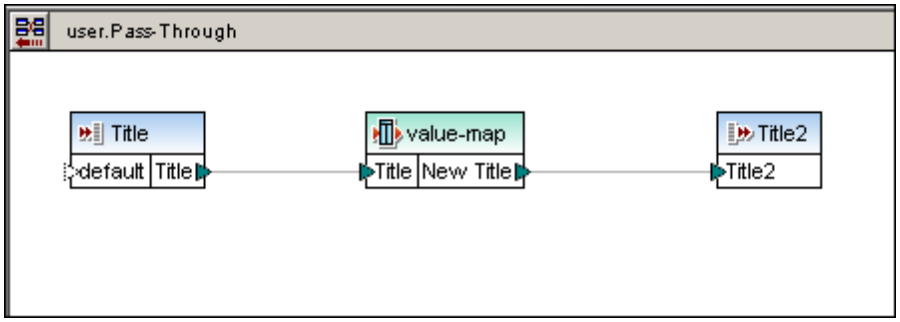
Solution:

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

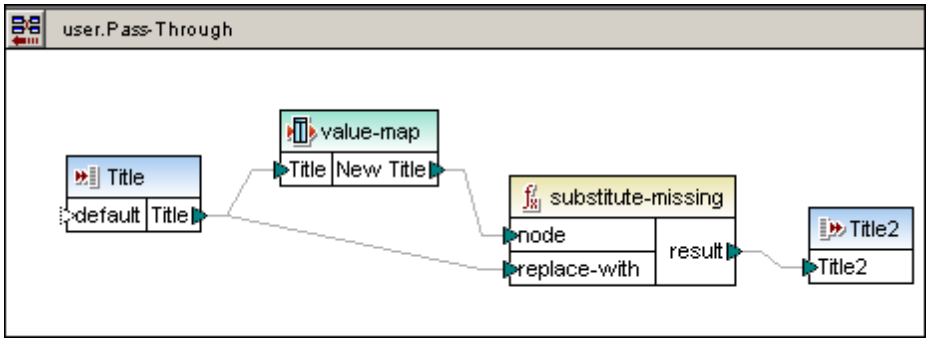
1. Click the value-map component and select **Function | Create user-defined function from Selection**.



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



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



4. Click the Ouput tab to see the result:

Result of the mapping:

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

```
38  <Person>
39    <First>Fred</First>
40    <Last>Landis</Last>
41    <Title>Program Manager</Title>
42    <PhoneExt>951</PhoneExt>
43    <Email>f.landis@nanonull.com</Email>
44  </Person>
45  <Person>
46    <First>Michelle</First>
47    <Last>Butler</Last>
48    <Title>Code Magician</Title>
49    <PhoneExt>654</PhoneExt>
50    <Email>m.landis@nanonull.com</Email>
51  </Person>
```

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

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

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

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

5.10.2 Value-Map component properties

Actions:

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

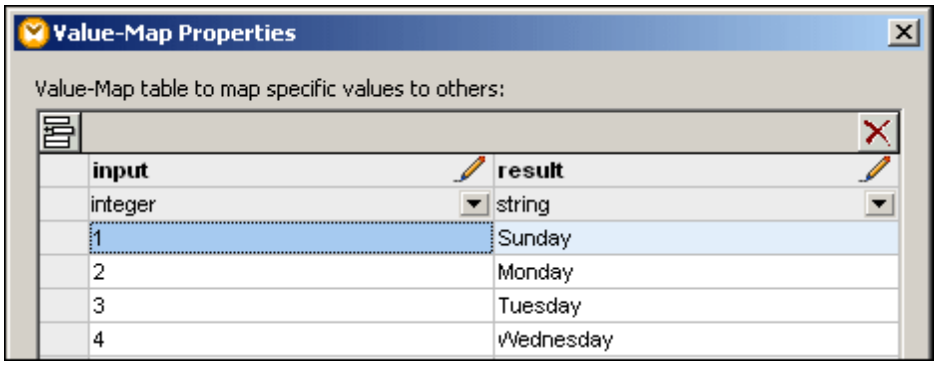
 Click the delete icon to **delete** the currently active row.

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

You can also reorder lines by dragging them.

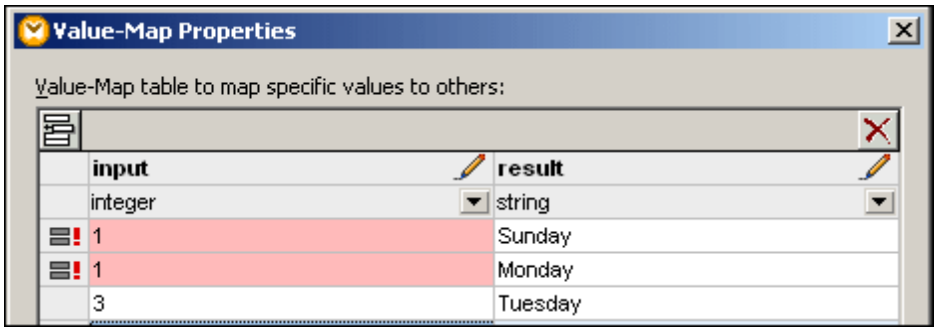
Changing the column header:

Double clicking the column header, or clicking the pencil icon, allows you to edit the column name and change it to something more meaningful. This will make it easier to identify the purpose of the component, as the column names are also displayed in the mapping.



Using unique Input values:

The values entered into the input column must be unique. If you enter two identical values, both are automatically highlighted for you to enable you to correct one of them.

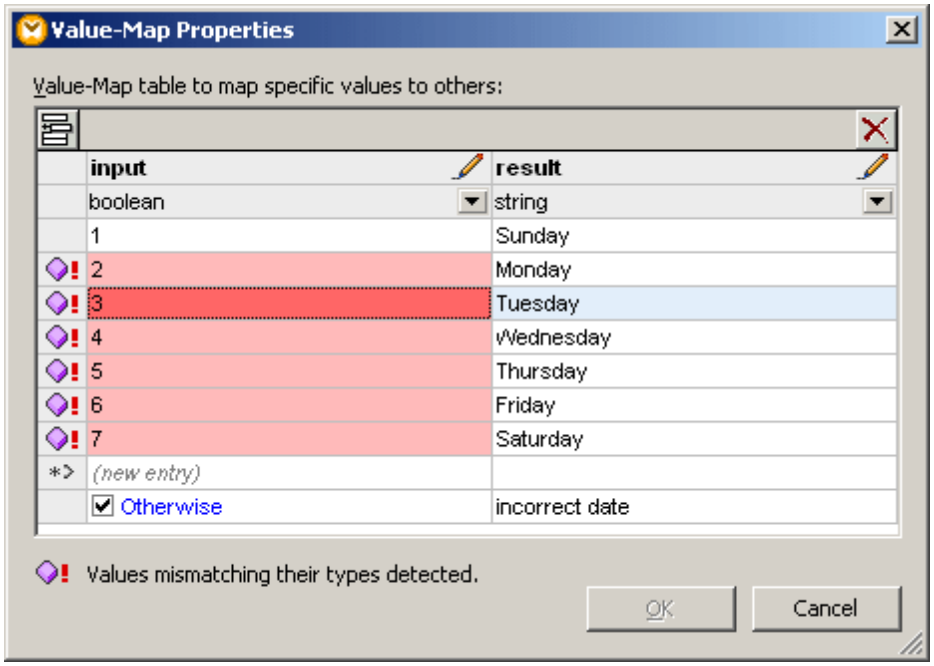


Having corrected one of the values, the OK button is again enabled.

Input and output datatypes

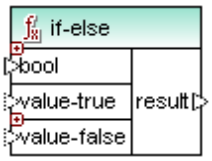
The input and result datatypes are automatically checked when a selection is made using the combo box. If a mismatch occurs, then the respective fields are highlighted and the OK button is disabled. Change the datatype to one that is supported.

In the screenshot below a boolean and string have been selected.



5.11 Using If-Else Conditions

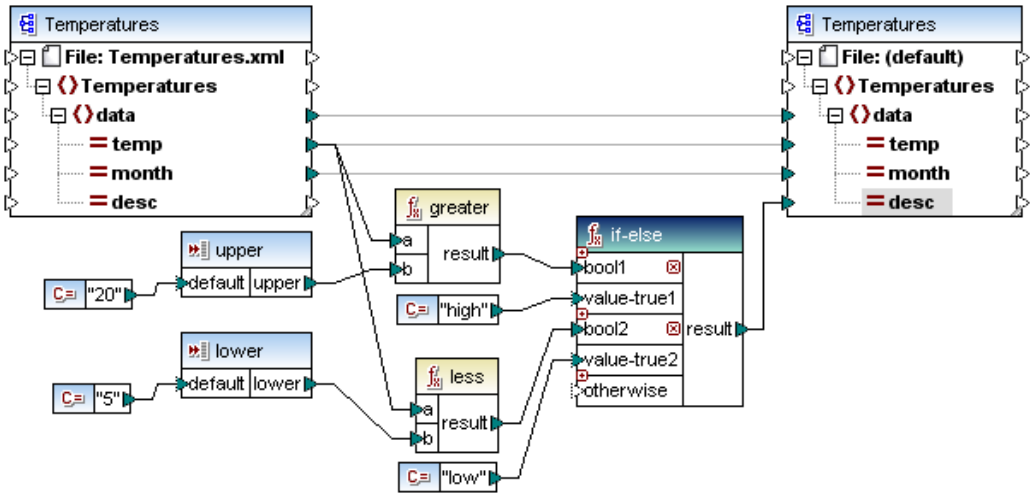
An **If-Else Condition** is a special component type which allows you to pass on different sets of data depending on the outcome of a preset condition. The component header displays the text **if-else**.



- The first input parameter is a **bool**, which contains the data you are checking against.
- The **value-true** input parameter supplies the data to be passed on, as a result, if the condition is true.
- The **value-false** supplies the data to be passed on if the condition is false.
- The **result** parameter outputs the data supplied by the value-true/false input parameters.

The IF-Else function is **extendable**. This means that you can check for multiple conditions and use the **otherwise** parameter to output the Else condition/value.

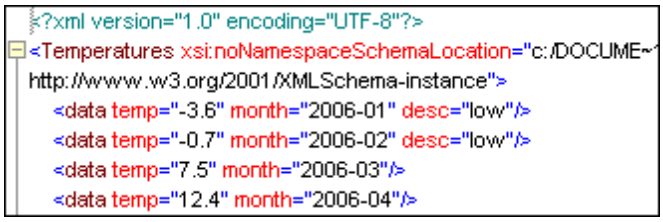
Clicking the "plus" icon inserts or appends a new if-else **pair**, i.e. boolX and value-trueX, while clicking the "x" deletes the parameter pair.



In the example above, the temperature data is analyzed:

- If temp is **greater** than 20, then true is passed on to **bool1** and the result is "**high**" from **value-true1.**
- Else, If temp is **less** than 5, then true is passed on to **bool2** and the result is "**low**" from **value-true2.**
- Otherwise, nothing (an empty sequence) is the result of the component, since there is no connection to the "otherwise" input.

Result of the mapping:

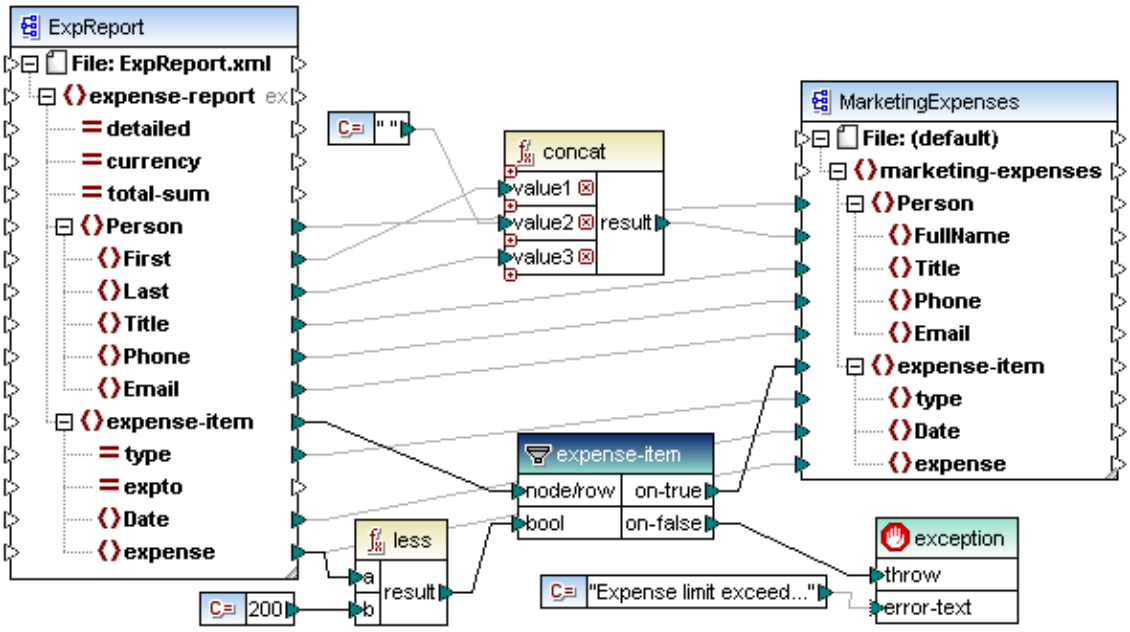


5.12 Adding Exceptions

MapForce provides support for the definition of exceptions. You can define the condition that will throw an error. When the condition is satisfied, a user-defined message appears and the mapping process is stopped. The **ExpenseLimit.mfd** file in the **...\MapForceExamples** folder is a sample mapping that contains an exception function.

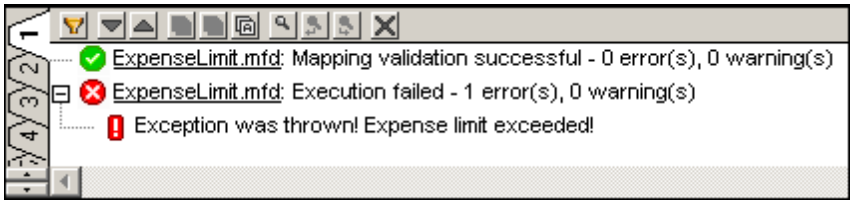
To insert an exception component:

- Select the menu option **Insert | Exception**, or click the Exception icon in the icon bar.



The example above shows how exceptions are defined in mappings. The exception should be triggered when the expense limit is exceeded, i.e. higher than 200 in this example.

- The **less** component checks to see if expense is less than 200 with the bool result being passed on to the filter component.
- When the bool result is **false**, i.e. expense is greater than, or equal to, 200, the **on-false** parameter of the filter component activates the exception and the mapping process is stopped. (Note that you can also connect the exception to the on-true parameter, if that is what you need.)
- The error text (Expense limit exceeded!) supplied by the **constant** component is mapped to the **error-text** parameter of the exception function.
- The error text appears in the Output tab, and also when running the compiled code.



Please note:
It is very important to note the filter placement in the example:

- **Both parameters** of the **filter** component, on-true and on-false, must be mapped! One of them needs to be mapped to the **exception** component, and the other, to the target component that receives the filtered source data. If this is not the case, the exception component will never be triggered.
- The **exception** and **target** components must be **directly connected** to the **filter** component. Functions, or other components, may not be placed between the filter and either the exception, or target components.
- When generating **XSLT 2.0** and **XQuery** code, an "Execution failed" error appears in the Messages window, and the respective XSLT2 or XQuery tab is opened with the error line automatically highlighted.

5.13 Parsing and Serializing Strings

String parsing and serialization is an advanced mapping technique that enables you to configure the component to either parse data from a string, or serialize data to a string. This technique can be regarded as an alternative to reading data from (or writing data to) files. MapForce components which parse strings or serialize data to strings can be useful in a variety of situations, for example:

- You need to insert structures such as XML, text, or JSON files into database fields or Excel spreadsheet cells.
- You need to convert XML fragments stored in database fields into standalone XML files.
- You have legacy data stored as text (for example, fixed-length content in a single database field), and you would like to convert this data into a fully sortable, field-based structure (using FlexText, for example)

String parsing and serialization is available for the following MapForce component types:

- Text (CSV, fixed-length field text, EDI, and MapForce FlexText templates)
- JSON schema files
- XML schema files

String parsing and serialization is supported in MapForce target languages as follows.

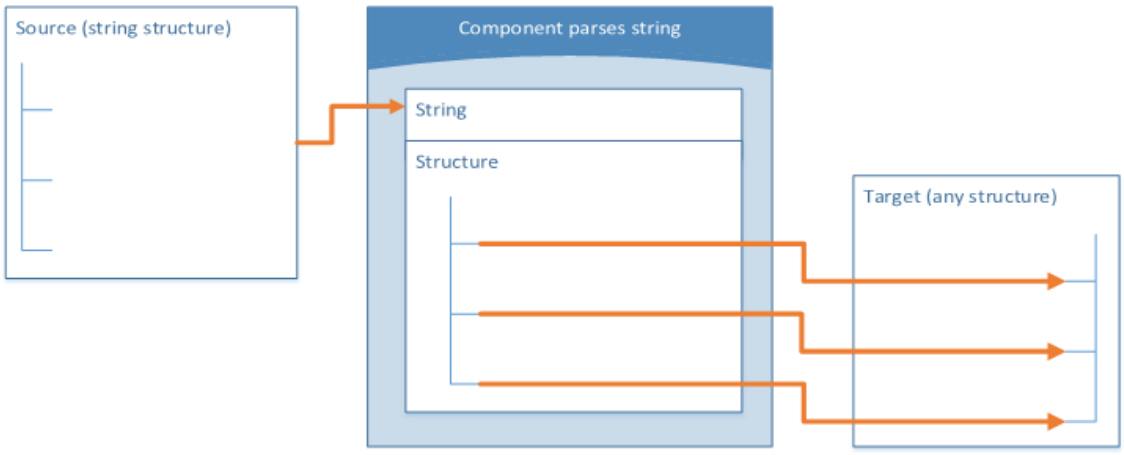
Language	Reading	Writing
BUILT-IN (preview the mapping transformation)	Yes	Yes
BUILT-IN (run the MapForce Server execution file)	Yes	Yes

This section includes the following topics:

5.13.1 About the Parse/Serialize Component

A Parse/Serialize component in MapForce is a hybrid component which is neither a source nor a target component. Given the role they play in the mapping design, such components must be placed in between other source and target components.

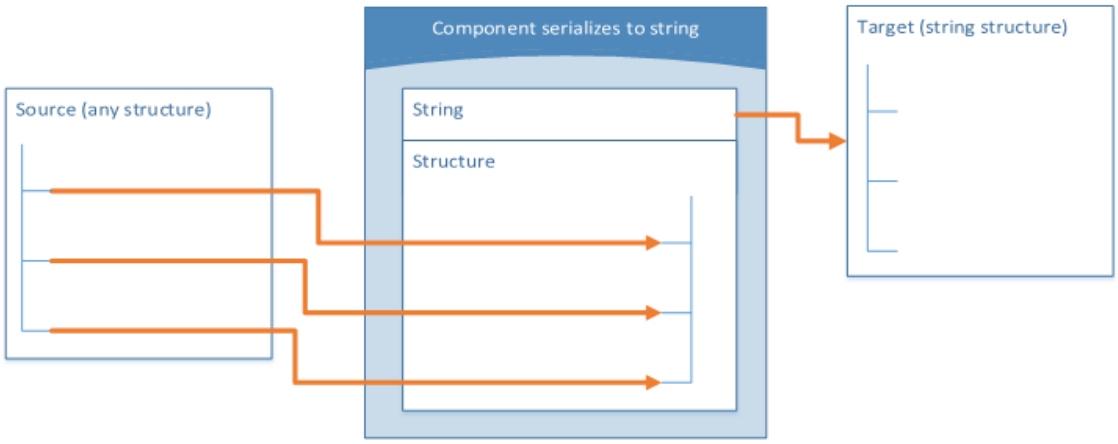
You can use a "Parse/Serialize String" component for string parsing when, for some reason, you need to convert a string that has structure (for example, some XML stored as string in a database) into another format. Parsing data from the source string to the "Parse/Serialize" component means that the source string is turned into a MapForce structure, and, thus, you get access to any element or attribute of the source XML stored as string.



Generic "Parse String" component

The diagram above illustrates the typical structure of a MapForce component which parses a string. Note that the "Parse/Serialize String" component is placed in between the source and target of the mapping. What this component does is accept some string structure as input, by means of a single MapForce connector which is connected to its top **String** node. The output structure can be any of the data targets supported by MapForce. For an example, see Example: Parse String (Fixed-Length Text to Excel).

When you serialize data from a component to string, the reverse happens. Specifically, the entire structure of the MapForce component becomes a string structure which you can further manipulate as necessary. For example, this enables you to write an XML file (or XML fragment) to a database field or to a single cell of an Excel spreadsheet.

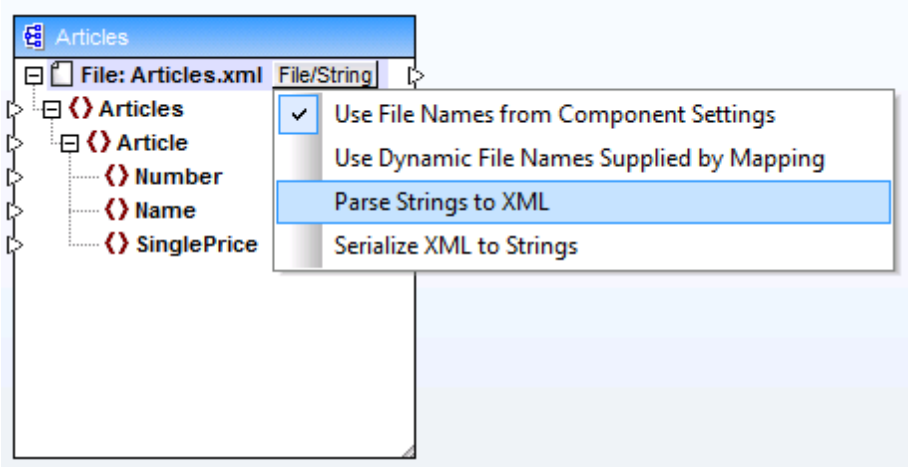


Generic "Serialize to String" component

The diagram above illustrates a generic MapForce "Serialize to String" component. What this component does is accept as input any data source supported by MapForce (by means of standard MapForce connectors). The output structure is a string which you can pass further by means of a single MapForce connector drawn from the top **String** node of the component to a target component item (for example, a spreadsheet cell). For an example, see Example: Serialize

to String (XML to Database).

You can designate a component for string parsing or serialization at any time from the mapping window. To do so, click the **File/String** (File/String) button adjacent to the root node, and then select the desired option.



Changing the component mode

Note: A "Parse/Serialize String" component cannot read data from a string and write to a string simultaneously. Therefore, the root node can have either an incoming connector or an outgoing connector (not both). An error will be generated if you attempt to use the same component for both operations.

When you designate a component for string parsing or serialization, the appearance of component changes as follows:

- The component gets the **parse** or **serialize** prefix in the title.
- The title bar has yellow background color, similar to function components.
- The top node begins with the **String:** prefix and is identified by the icon.
- If the component parses a string, the output connector from the root node is not meaningful and thus it is not available.
- If the component serializes to a string, the input connector to the root node is not meaningful and thus it is not available.

When a component is in "Parse/Serialize String" mode, you can change its settings in a similar way as if it were in a file-based mode (see Changing the Component Settings). Note that not all component settings are available when a component is in either "Parse" or "Serialize" mode.

5.13.2 Example: Parse String (Fixed-Length Text to Excel)

This example walks you through the steps required to create a mapping design which parses string data. The example is accompanied by a sample file. If you want to look at the sample file before starting this example, you can open it from the following path: **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial\ParseString.mfd**.

Let's assume a scenario where you have some legacy text data stored as a single database field.

The text data is a list of employees, stored in the RESOURCE column, and formatted as fixed-length fields, as follows:

```
|----|-----------------------------------------------------------------------------------------|------------|
| ID | RESOURCE                                                                                | TYPE       |
|----|-----------------------------------------------------------------------------------------|------------|
| 1  | P00001Callaby        Vernon         582Office Manager           v.callaby@nanonull.com    | Employees  |
|    | P00002Further        Frank          471Accounts Receivable      f.further@nanonull.com    |            |
|    | P00003Matise         Loby           963Accounting Manager       l.matise@nanonull.com     |            |
|    | P00004Firstbread     Joe            621Marketing Manager Europe j.firstbread@nanonull.com |            |
|    | P00005Sanna          Susi           753Art Director             s.sanna@nanonull.com       |            |
|    | P00006Landis         Fred           951Program Manager          f.landis@nanonull.com      |            |
|    | P00007Butler         Michelle       654Software Engineer        m.landis@nanonull.com      |            |
|    | P00008Little         Ted            852Software Engineer        t.little@nanonull.com      |            |
|    | P00009Way            Ann            951Technical Writer         a.way@nanonull.com         |            |
|    | P00010Gardner        Liz            753Software Engineer        l.gardner@nanonull.com     |            |
|    | P00011Smith          Paul           334Software Engineer        p.smith@nanonull.com       |            |
|----|-----------------------------------------------------------------------------------------|------------|
```

Because the text data is stored as a single database field, you cannot easily access and manipulate each individual employee record. This makes it difficult to add or remove new employees, or to sort data. For the purpose of this example, your goal is to extract the text data from the RESOURCE database field and split it into a structure so that you can easily process the records.

This task can be accomplished by using a "Parse/Serialize String" component. First, the "Parse/Serialize String" component will take the text data as input. Then it will parse it and convert it into a structure. Finally, it will write the structure to a target format. In this example, the target format is an Excel spreadsheet; however, in general, it can be any other output format supported by MapForce.

To summarize, the mapping described in this example will convert the contents of he RESOURCE database field to a table. After the mapping is executed, each table row will correspond to an employee and each column will correspond to one of the fixed-length fields, in this order: ID, Last Name, First Name, Extension, Job Title, Email.

	A	B	C	D	E	F	G
1	P00001	Callaby	Vernon	582	Office Manager	v.callaby@nanonull.com	
2	P00002	Further	Frank	471	Accounts Receivable	f.further@nanonull.com	
3	P00003	Matise	Loby	963	Accounting Manager	l.matise@nanonull.com	
4	P00004	Firstbread	Joe	621	Marketing Manager Europe	j.firstbread@nanonull.com	
5	P00005	Sanna	Susi	753	Art Director	s.sanna@nanonull.com	
6	P00006	Landis	Fred	951	Program Manager	f.landis@nanonull.com	
7	P00007	Butler	Michelle	654	Software Engineer	m.landis@nanonull.com	
8	P00008	Little	Ted	852	Software Engineer	t.little@nanonull.com	
9	P00009	Way	Ann	951	Technical Writer	a.way@nanonull.com	
10	P00010	Gardner	Liz	753	Software Engineer	l.gardner@nanonull.com	
11	P00011	Smith	Paul	334	Software Engineer	p.smith@nanonull.com	
12							
13							

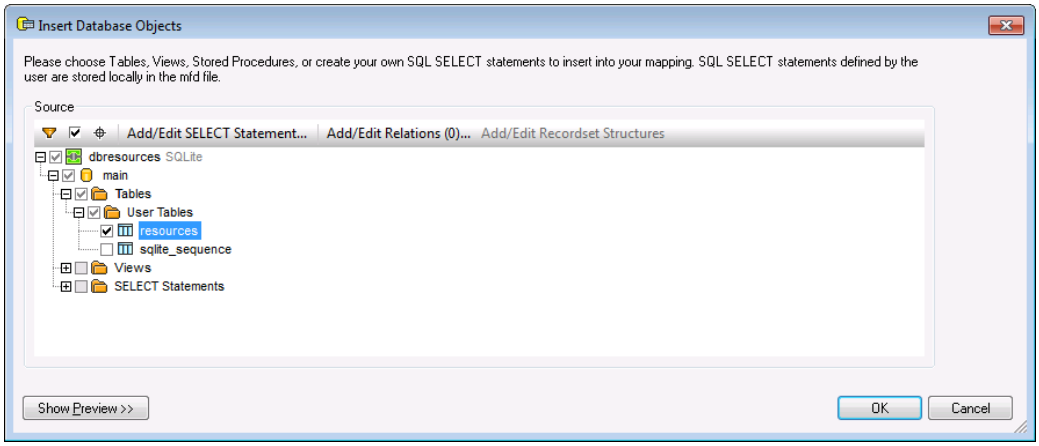
Expected output after parsing the string

To accomplish the goal, follow the steps below:

1. Add to the mapping area the source database. The database is available as a standalone SQLite database file at the following path: **<Documents>\Altova\MapForce2016**

\MapForceExamples\Tutorial\\dbresources.db. (To add the database component, use the **Insert | Database** menu command, see also Connecting to a Database).

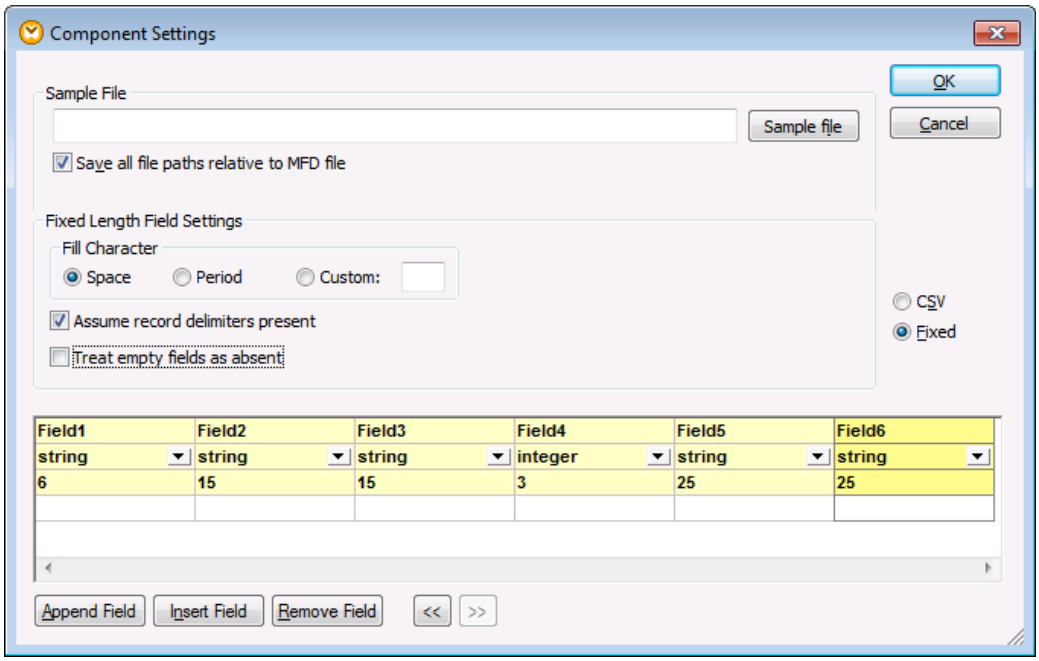
2. When prompted to insert the database objects, select the **resources** table.



3. Add to the mapping area a Text component (use the **Insert | Text File** menu command). Since the source data is field-delimited text, choose **Use simple processing...** when prompted by MapForce.
4. Configure the structure of the Text component to map to the structure of the source text data stored in the database. As you may have already noticed, in this example, the source text consists of six fixed-length fields of fixed size, as follows:

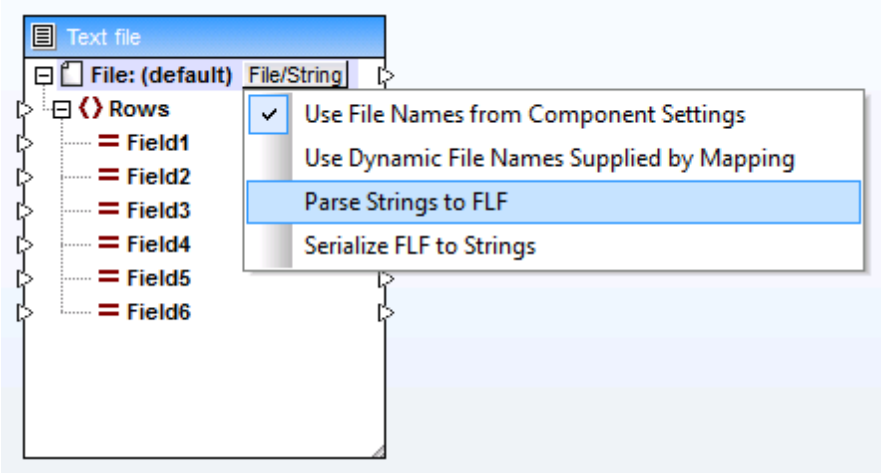
Size in characters	Field
6	ID
15	Surname
15	First Name
3	Extension
25	Job Title
25	Email

To achieve this, declare the Text component as **Fixed**, and add to it six fields that correspond to positions above. (To open the Component Settings dialog box, right-click the component, and then select **Properties** from the context menu.)

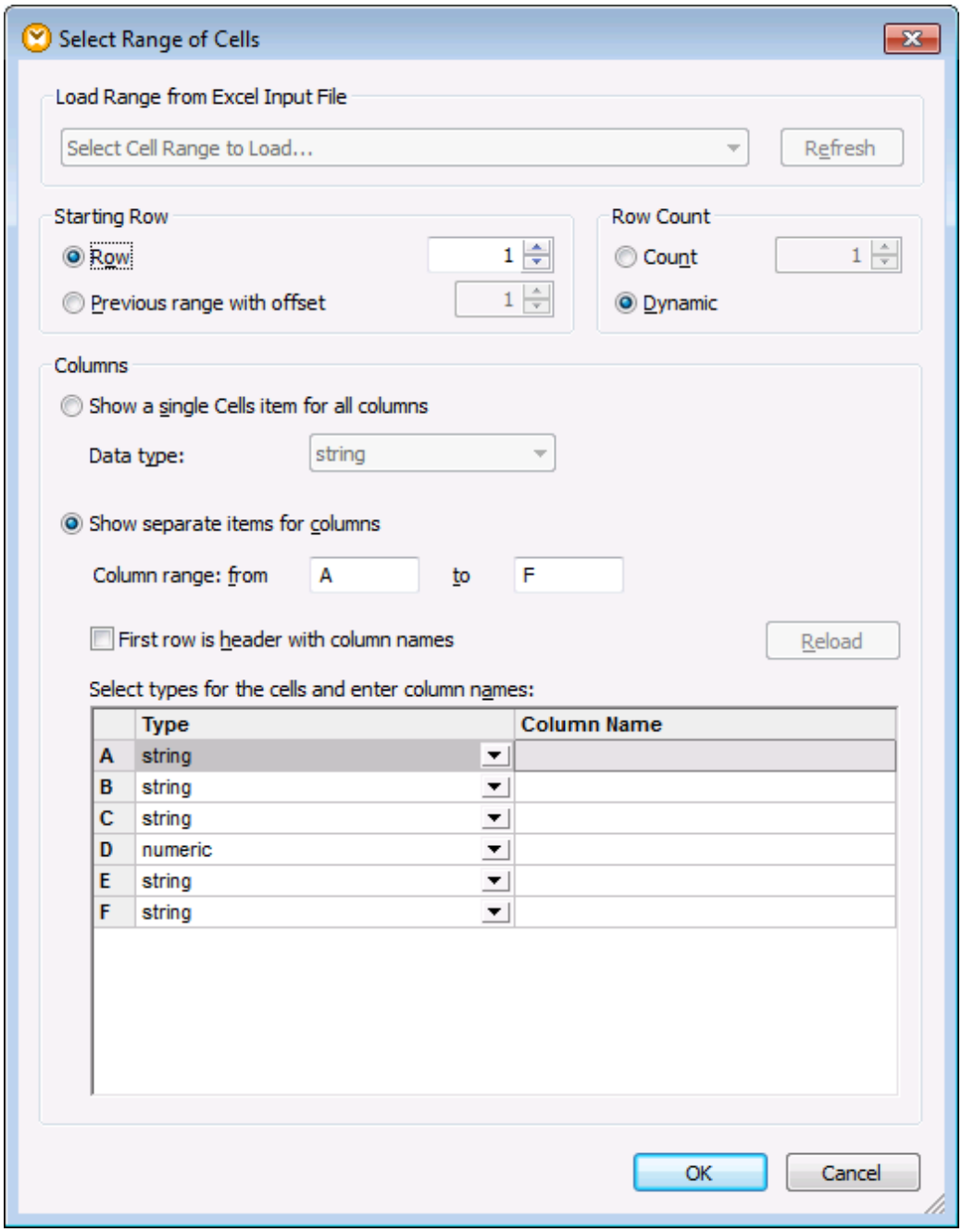


Observe the **Field4** field, which uses **integer** as data type. Although declaring the **Field4** as numeric type is optional for the scope of the current example, this ensures that the phone extension (**Field4**) extracted from the source text is validated as a numeric value.

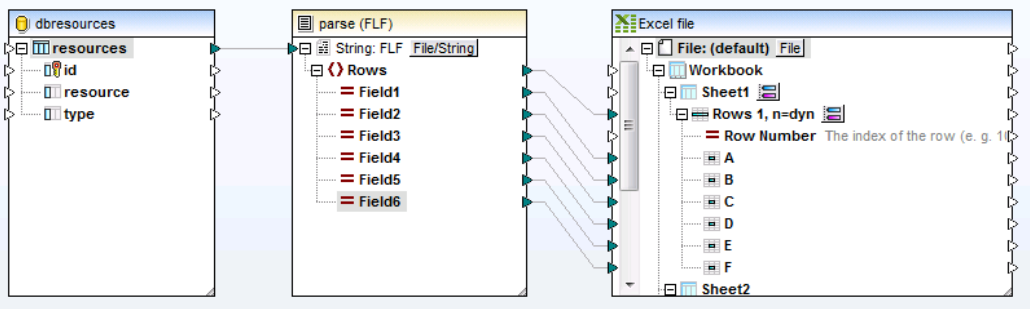
5. Click File/String the, and then select **Parse Strings to FLF** from the context menu. This instructs MapForce that this component will parse a string to fixed-length field (FLF) format.



6. Add to the mapping area the target Excel 2007+ component. When prompted to select a sample file, click **Skip**. (You can add the Excel component using the **Insert | Excel 2007 +** menu command, see also Adding Excel 2007+ Files as Mapping Components).
7. Click the button next to **Row 1, n=dyn** node, and configure the Excel component to write a row for each text field, starting with the first row, as shown below. (For more information about Excel 2007+ component types and their configuration, see About the Excel 2007+ Component).



8. Draw the connections between component items, as shown below.



On the left side of the mapping, the contents of the resource database column is being converted (parsed) from a string value to a MapForce structure. On the right side of the mapping, the items of the *Parse (Text file)* component are connected to individual Excel columns, thus splitting the source string into individual sortable cells.

You have now finished creating a MapForce design file which parses string data and creates a structure from it. If you click the **Output** tab, the legacy text data is now converted to individual rows and columns of the Excel spreadsheet, which was the intended goal of this mapping.

5.13.3 Example: Serialize to String (XML to Database)

This example walks you through the steps required to create a mapping design which serializes data to a string. The example is accompanied by a sample file. If you want to look at the sample file before starting this example, you can open it from the following path: **<Documents>\Altova \MapForce2016\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>
```

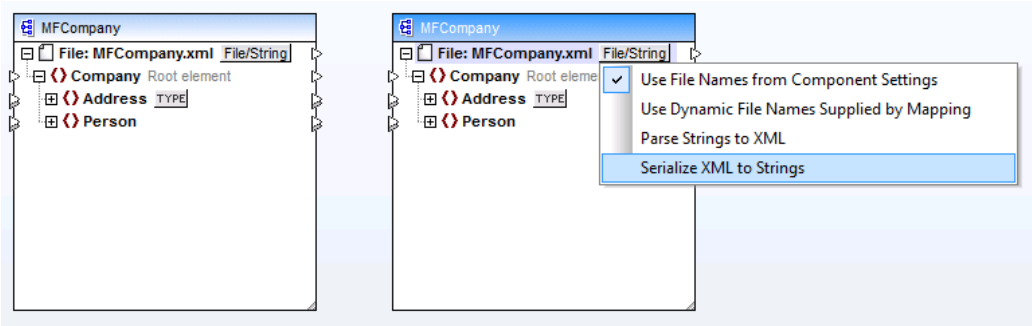
Your goal is to extract each `<Person>` element from the XML file and insert it literally (including XML tags) as a new database record in the `PEOPLE` table of a SQLite database. The `PEOPLE` table contains only two columns: `ID` and `PERSON`. Its full definition is as follows:

```
CREATE TABLE PEOPLE (ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, PERSON
TEXT);
```

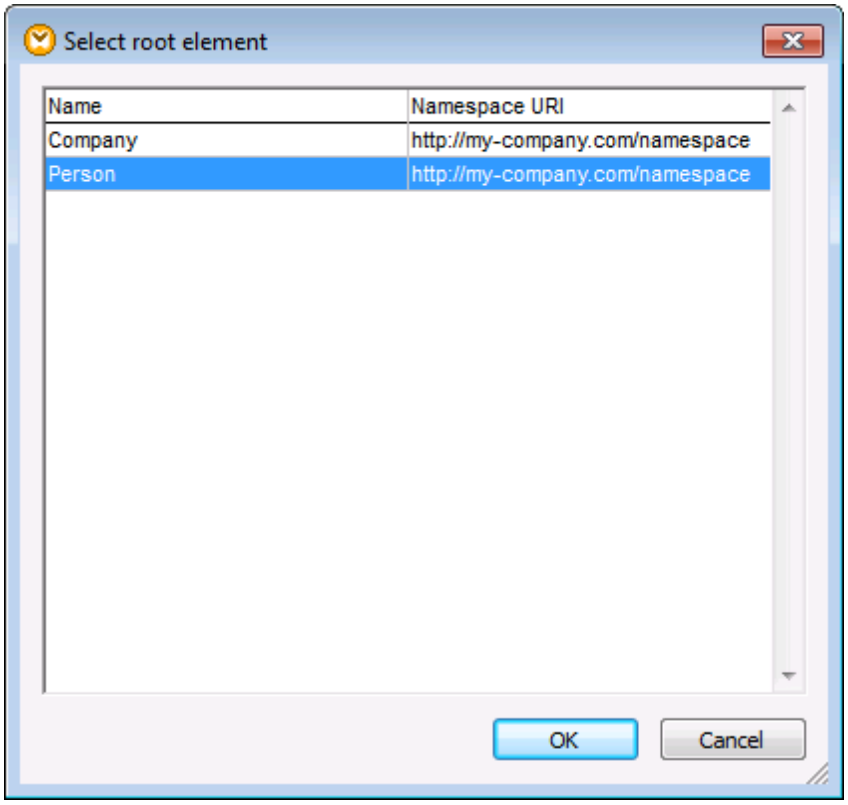
After the mapping is executed, the expected result is that the `PEOPLE` table will have the same number of rows as the number of <Person> elements in the XML file.

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\MapForce2016\MapForceExamples\Tutorial\MFCompany.xml**.
2. Duplicate (copy-paste) the XML component.
3. On the duplicated XML component, click File/String, and then select **Serialize XML to Strings**.

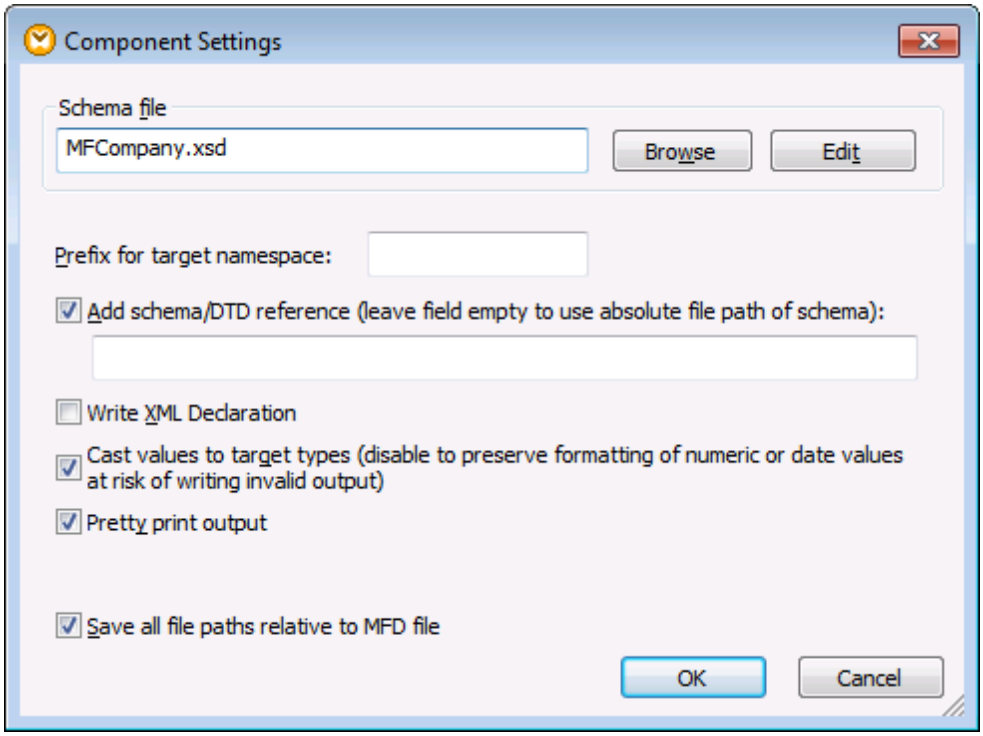


4. Right-click the duplicated component and select **Change Root Element** from the context menu. Then change the root element to `<Person>`.

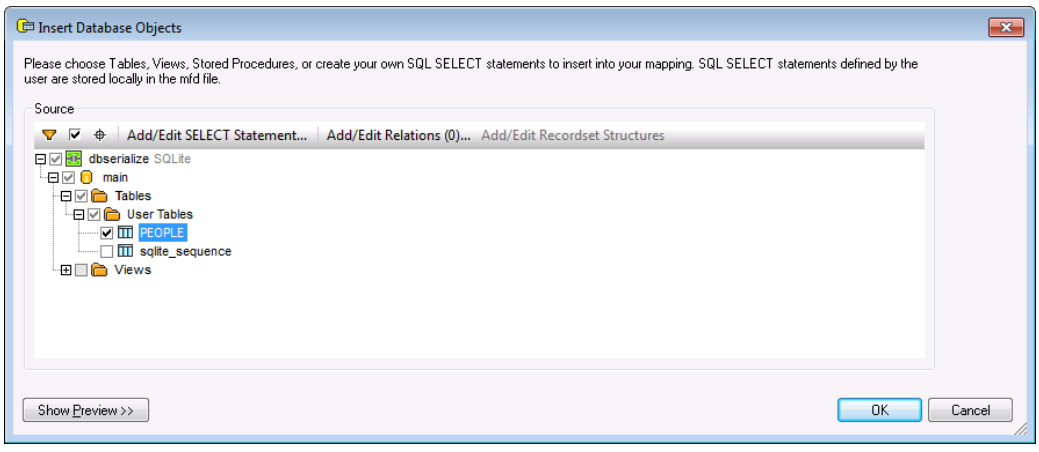


In general, you can change the root element to any element that has a global (not local) declaration in the XML schema. Any elements that are not defined globally in your schema are not listed in the "Select Root Element" dialog box.

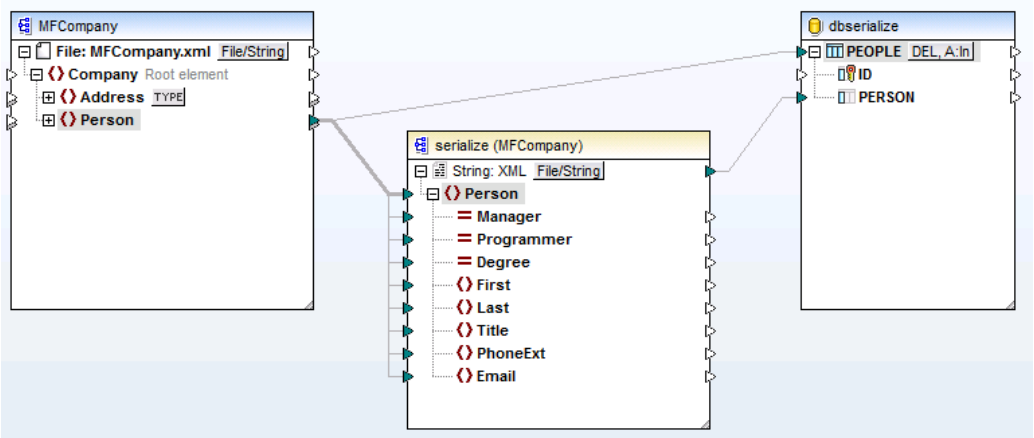
5. Double-click the component and disable the **Write XML Declaration** option. This prevents the XML declaration from being written for each `<Person>` element.



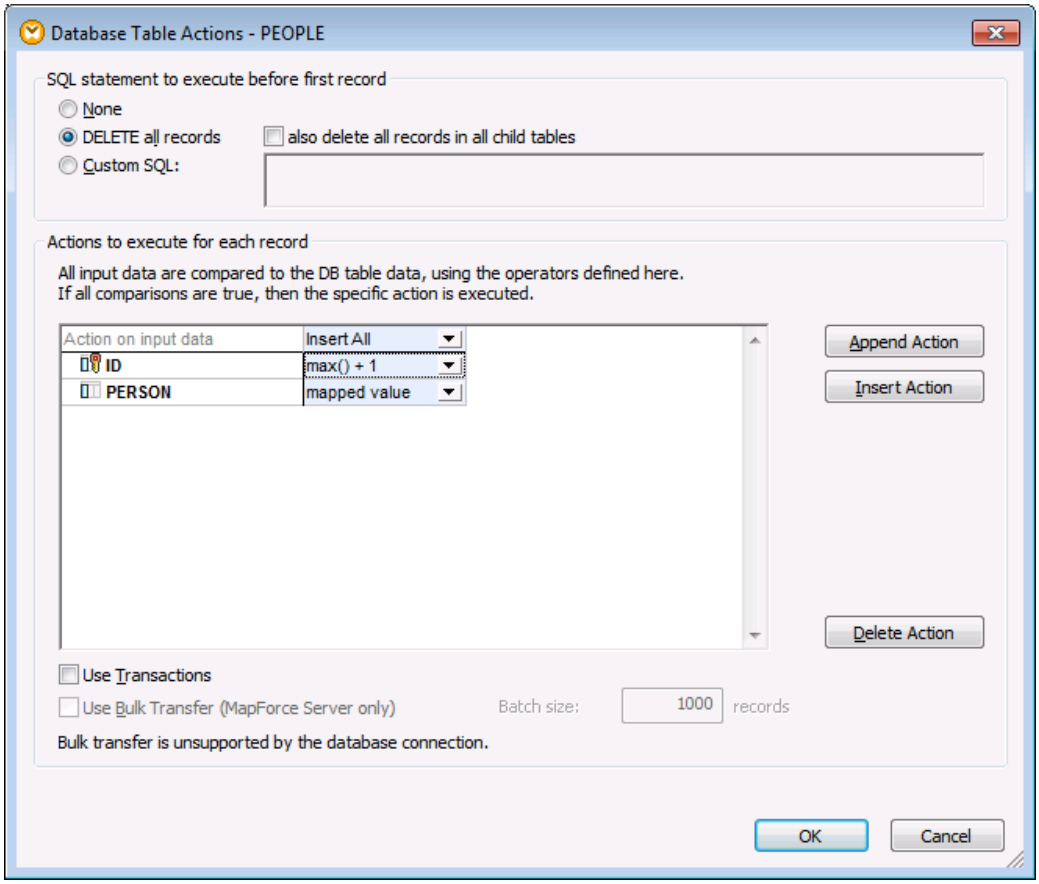
6. Add to the mapping area the target SQLite database component, from the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\\dbserialize.db**. (To add the database component, use the **Insert | Database** menu command, see also Connecting to a Database). When prompted to insert a database object, select the `PEOPLE` table.



7. Link the components as shown below. On the left side of the mapping, the `<Person>` element maps to the serialization component. On the right side of the mapping, the serialized string value is inserted into the `PERSON` column of the `PEOPLE` database table. Finally, the connector drawn from `<Person>` to `PEOPLE` table instructs MapForce to create a new record for each `<Person>` element encountered.



8. Click the **A:In** button on the database component, and instruct MapForce to perform the following actions every time when the mapping transformation runs:
   a. Delete all records from the table;
   b. Increment the value of ID by 1.



Observe the **max()+1** action selected for the ID column. This instructs MapForce to analyze what is the maximum ID value already existing in the database, and insert the next available integer, incremented by 1.

You have now created a mapping design which serializes data to string. If you click the **Output** tab, the preview SQL query indicates that separate records will be inserted into the database for each `<Person>` element in the XML file, which was the goal of this mapping.

5.14 Mapping Rules and Strategies

MapForce generally maps data in an intuitive way, but you may come across situations where the resulting output seems to have too many, or too few items. This topic is intended to help you avoid such mapping problems.

General rule

Generally, every connection between a source and target item means: for each source item, create one target item. If the source node contains simple content (for example, string or integer) and the target node accepts simple content, then MapForce copies the content to the target node and, if necessary, converts the data type.

This generally holds true for all connections, with the following exceptions:

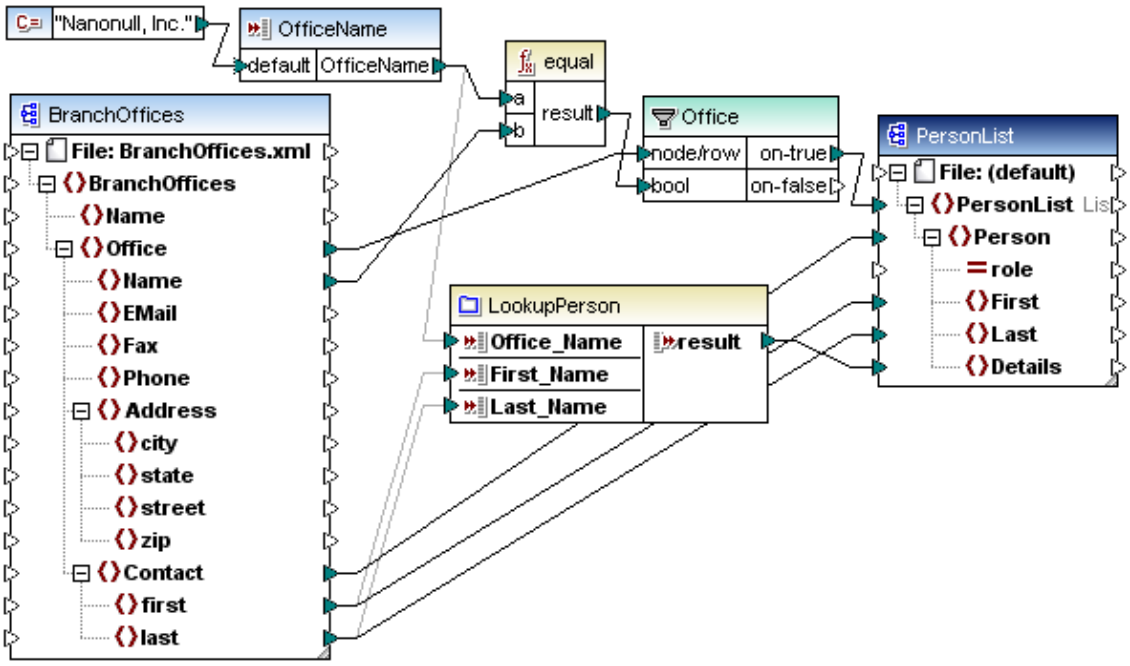
- A target XML root element is always created once and only once (see also Mapping to the Root Element). If you connect a sequence to it, only the contents of the element will be repeated, but not the root element itself, and the result might not be schema-valid. If attributes of the root element are also connected, the XML serialization will fail at runtime, so you should avoid connecting a sequence to the root element. If what you want to achieve is creating multiple output files, connect the sequence to the "File" node instead, via some function that generates file names.
- Some nodes accept a single value, not a sequence (for example, XML attributes, database fields , and output components in user-defined functions).

The "context" and "current" items

MapForce displays the structure of a schema, database, or EDI file as a hierarchy of mappable items in the component. Each of these nodes may have many instances (or none) in the instance file or database.

Example: If you look at the source component in **PersonListByBranchOffice.mfd**, there is only a single node **first** (under **Contact**). In the **BranchOffices.xml** instance file, there are multiple **first** nodes and **Contact** nodes having different content, under different **Office** parent nodes.

It depends on the current **context** (of the **target** node) which source nodes are actually selected and have their data copied, via the connector, to the target component/item.



PersonListByBranchOffice.mfd

This context is defined by the **current target node** and the connections to its ancestors:

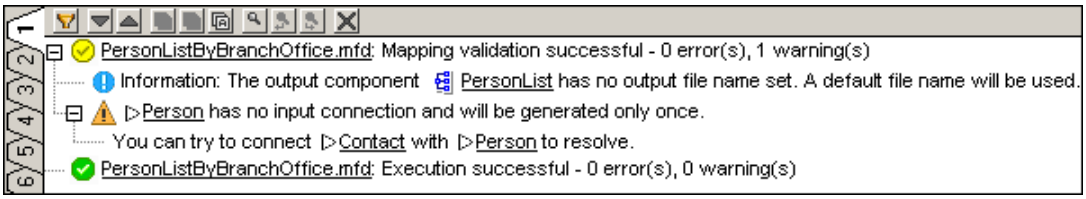
- Initially the context contains only the source components, but no specific nodes. When evaluating the mapping, MapForce processes the **target root** node first (PersonList), then works down the hierarchy.
- The connector to the **target** node is traced back to all source items directly or indirectly connected to it, even via functions that might exist between the two components. The source items and functions results are added to the context for this node.
- For each new target node a new context is established, that initially contains all items of the parent node's context. Target sibling nodes are thus independent of each other, but have access to all source data of their parent nodes.

Applied to the example mapping above (**PersonListByBranchOffice.mfd**):

- The connection from **Office** through the filter (Office) to **PersonList** defines a **single** office as the context for the whole target document (because PersonList is the root element of the target component). The office name is supplied by the input component, which has a default containing "Nanonull, Inc."
- All connections/data to the **descendants** of the root element PersonList, are automatically affected by the filter condition, because the selected single office is in the context.
- The connection from **Contact** to **Person** creates one target Person **per** Contact item of the source XML (general rule). For each Person one specific Contact is added to the context, from which the children of Person will be created.
- The connector from **first** to **First** selects the first name of the current Contact and writes it to the target item First.

Leaving out the connector from **Contact** to **Person** would create only **one** Person with multiple

First, Last, and Detail nodes, which is not what we want here. In such situations, MapForce issues a warning and a suggestion to fix the problem: "You can try to connect Contact with Person to resolve":



Sequences

MapForce displays the structure of a schema, database, or EDI file as a hierarchy of mappable items in the component.

Depending on the (target) context, each **mappable item** of a **source** component can represent:

- a **single instance** node of the assigned input file (or database)
- a **sequence** of 0 to **multiple instance** nodes of the input file (or database)

If a sequence is connected to a **target** node, a loop is created to create as many target nodes as there are source nodes.

If a **filter** is placed between the sequence and target node, the bool condition is checked for each input node i.e. each item in the sequence. More exactly, a check is made to see if there is at least one bool in each sequence that evaluates to true. The priority context setting can influence the order of evaluation, see below.

As noted above, filter conditions automatically apply to all descendant nodes.

Note: If the source schema specifies that a specific node occurs exactly once, MapForce may remove the loop and take the first item only, which it knows must exist. This optimization can be disabled in the source Component Settings dialog box (check box "Enable input processing optimizations based on min/maxOccurs").

Function inputs (of normal, non-sequence functions) work similar to target nodes: If a sequence is connected to such an input, a loop is created around the function call, so it will produce as **many results** as there are items in the sequence.

If a sequence is connected to **more than one** such function input, MapForce creates nested loops which will process the **Cartesian product** of all inputs. Usually this is not desired, so only one single sequence with multiple items should be connected to a function (and all other parameters bound to singular current items from parents or other components).

Note: If an empty sequence is connected to such a function (e.g. concat), you will get an **empty sequence** as result, which will produce no output nodes at all. If there is no result in your target output because there is no input data, you can use the “substitute-missing” function to insert a substitute value.

Functions with **sequence inputs** are the only functions that can produce a result if the input sequence is **empty**:

- **`exists`**, **`not-exists`** and **`substitute-missing`** (also, **`is-not-null`**, **`is-null`** and **`substitute-null`**, which are aliases for the first three)

- aggregate functions (**sum**, **count**, etc.)
- regular user-defined functions that accept sequences (i.e. non-inlined functions)

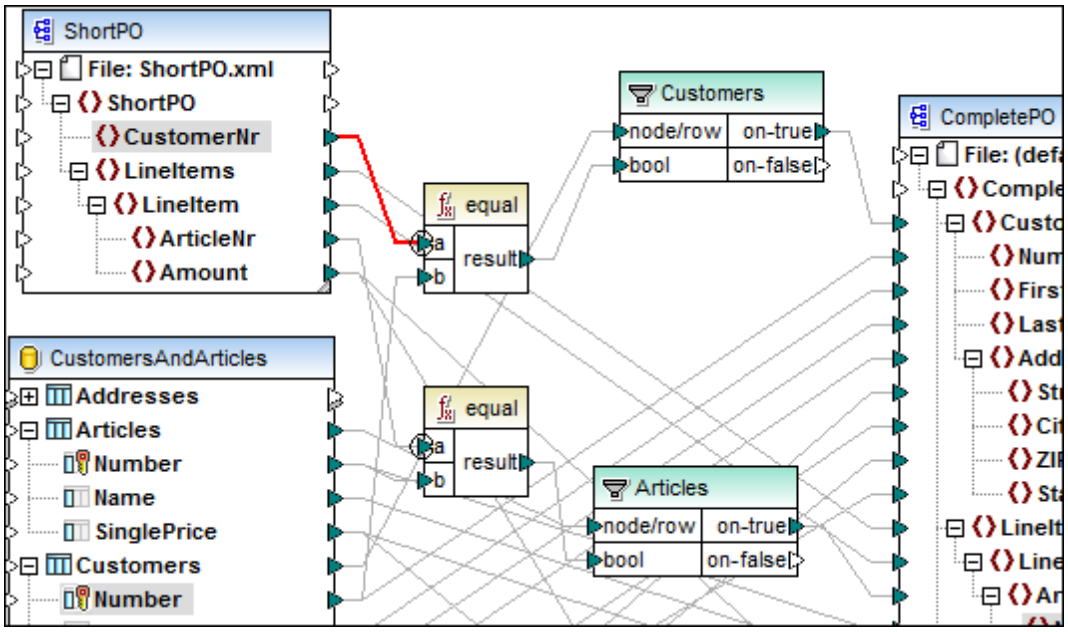
The sequence input to such functions is always evaluated independently of the current target node in the context of its ancestors. This also means that any filter or SQL-Where components connected to such functions, do not affect any other connections.

Priority context

Usually, function parameters are evaluated from top to bottom, but its is possible to define one parameter to be evaluated before all others, using the **priority context** setting.

In functions connected to the bool input of **filter** conditions, the priority context affects not only the comparison function itself but also the evaluation of the filter, so it is possible to join together two source sequences (see CompletePO.mfd, CustomerNo and Number).

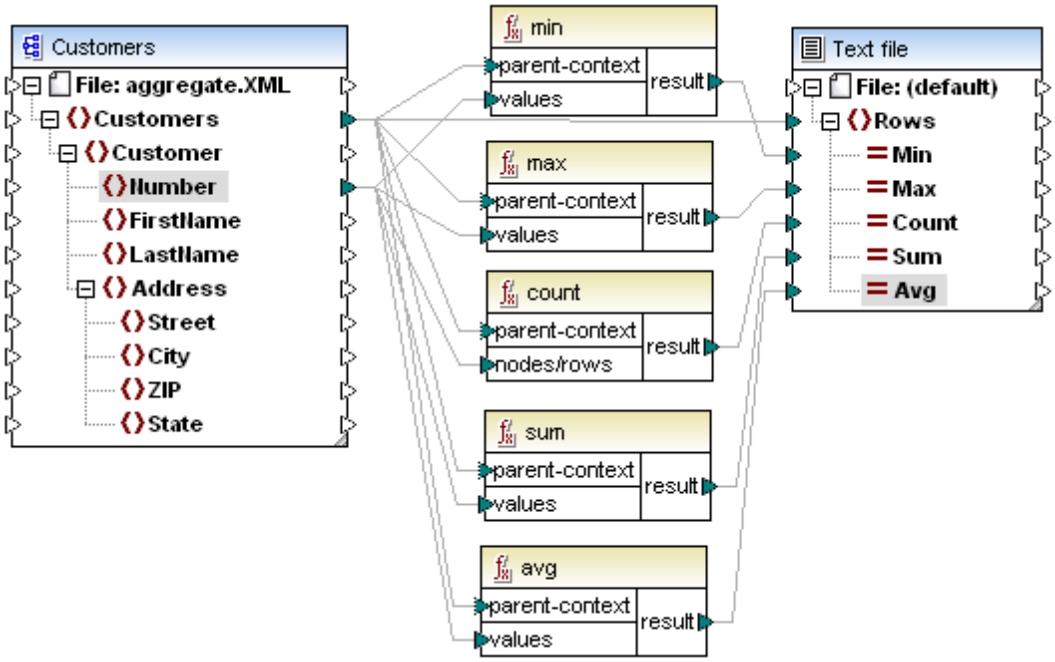
See example Priority Context node/item



Overriding the context

Some aggregate functions have an optional “parent-context” input.

If this input is not connected, it has no effect and the function is evaluated in the normal context for sequence inputs (that is, in the context of the target node's parent).



If the parent-context input is connected to a source node, the function is evaluated **for each** "parent-context" node and will produce a separate result for each occurrence.

Bringing multiple nodes of the same source component into the context
This is required in some special cases and can be done with Intermediate variables.

5.14.1 Changing the Processing Order of Mapping Components

MapForce supports mappings that have several target components. Each of the target components has a preview button allowing you to preview the mapping result for that specific component.

If the mapping is executed from the command line or from generated code, then, regardless of the currently active preview, the full mapping is executed and the output for each target component is generated.

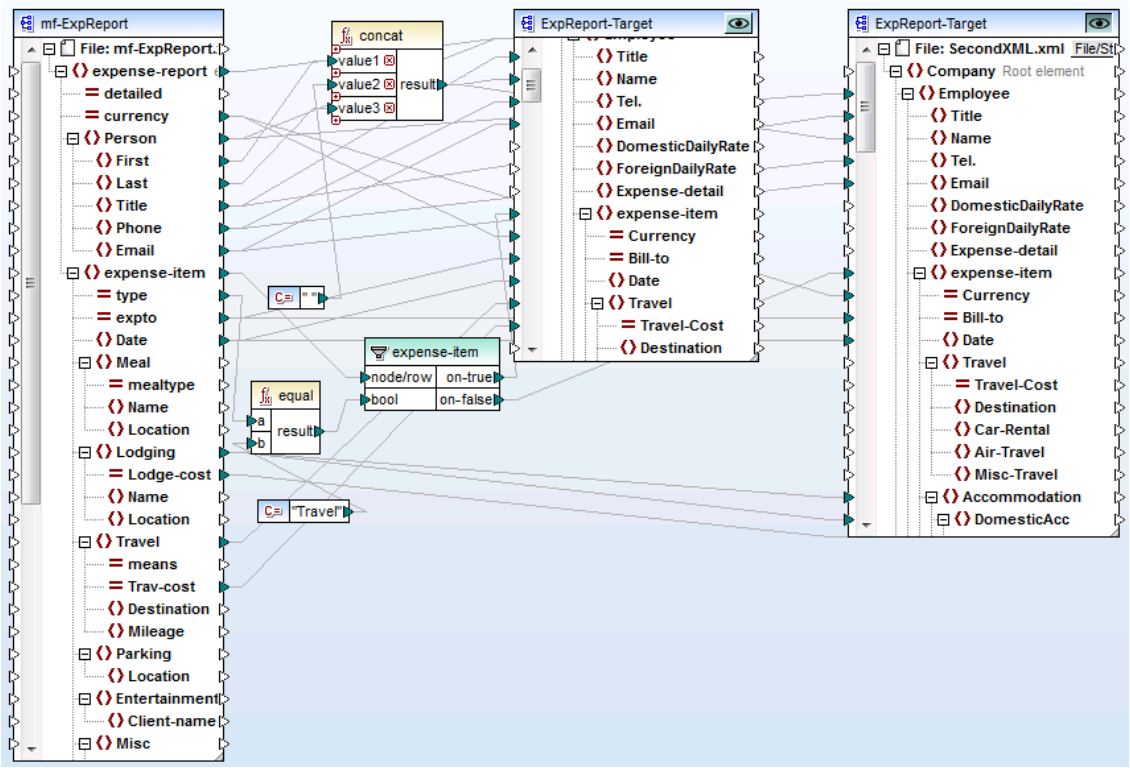
The order in which the target components are processed can be directly influenced by changing the position of target components in the mapping window. The **position** of a component is defined as its top left corner.

Target components are processed according to their Y-X position on screen, from top to bottom and left to right.

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

The screenshot below shows the tutorial sample **Tut-ExpReport-multi.mfd** available in the

<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial folder. Both target components (ExpReport-Target) have the same **vertical** position, and the preview button is active on the right hand target component.



Tut-ExpReport-multi.mfd (MapForce Enterprise Edition)

Having selected XSLT2 and generated the code:

- The leftmost target component is processed first and generates the **ExpReport.xml** file.
- The component to the right of it is processed next and generates the **SecondXML.xml** file.

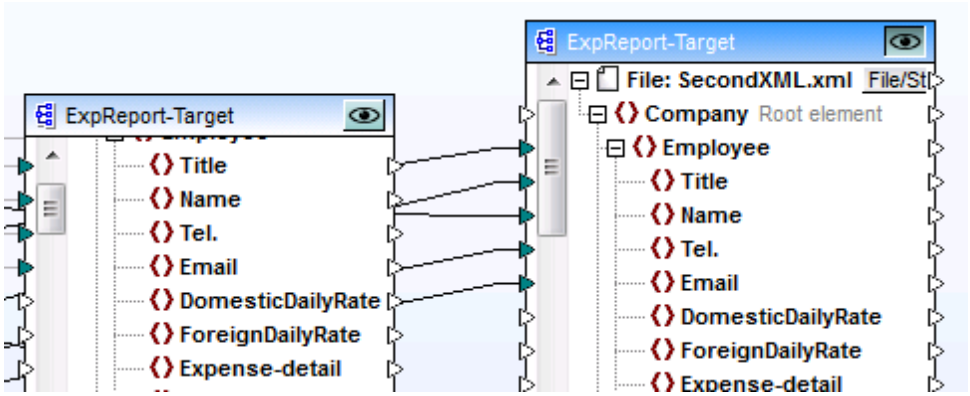
You can check that this is the case by opening the **DoTransform.bat** file (in the output folder you specified) and see the sequence the output files are generated. **ExpReport-Target.xml** is the first output to be generated by the batch file, and **SecondXML.xml** the second.

```
@echo off

RaptorXML xslt --xslt-version=2 --
input="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\m
f-ExpReport.xml" --
output="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
ExpReport-Target.xml" %* "MappingMapToExpReport-Target.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --
input="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\m
f-ExpReport.xml" --
output="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
SecondXML.xml" %* "MappingMapToExpReport-Target2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```


Changing the mapping processing sequence:

1. Click the left target component and move it below the one at right.



2. Regenerate your code and take a look at the **DoTransform.bat** file.

```
@echo off

RaptorXML xslt --xslt-version=2 --
input="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
mf-ExpReport.xml" --
output="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial
\SecondXML.xml" %* "MappingMapToExpReport-Target.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --
input="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
mf-ExpReport.xml" --
output="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial
\ExpReport-Target.xml" %* "MappingMapToExpReport-Target2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

SecondXML.xml is now the first output to be generated by the batch file, and **ExpReport-Target.xml** the second.

Chained mappings

The same processing sequence as described above is followed for chained mappings. The chained mapping group is taken as one unit however. Repositioning the intermediate or final target component of a single chained mapping has no effect on the processing sequence.

Only if multiple "chains" or multiple target components exist in a mapping does the position of the **final** target components of each group determine which is processed first.

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

Chapter 6

Debugging Mappings

6 Debugging Mappings

MapForce includes a mapping debugger available for the MapForce BUILT-IN transformation language. The mapping debugger helps you achieve the following goals:

- View and analyze the values produced by the mapping at each individual connector level.
- Highlight on the mapping the context (set of nodes) responsible for producing a particular value.
- Execute a mapping step-by-step, in order to see how MapForce processes or computes each value in real time, and preview the mapping output as it is being generated.
- Set milestones (breakpoints) at which the mapping execution should stop and display the value(s) currently being processed.
- View the history of values processed by a connector since mapping execution began up until the current execution position.

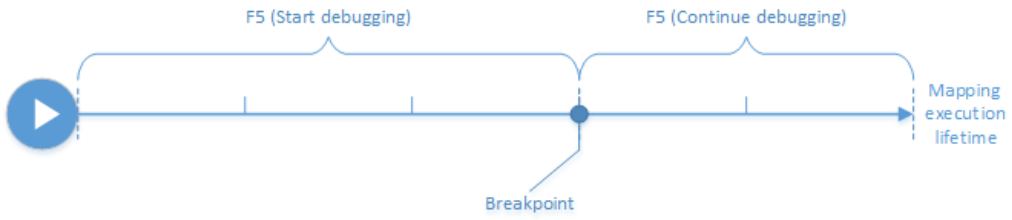
The mapping debugger is available when the transformation language of the mapping is BUILT-IN. If you start debugging a mapping designed for a different language, you will be prompted to change the mapping language to BUILT-IN. You can also convert a mapping to BUILT-IN by selecting the menu command **Output | Built-in Execution Engine**. In either case, the conversion to BUILT-IN will be successful if the mapping does not include components that are not available in the BUILT-IN language (for example, XSLT functions).

The MapForce debugger is unlike a traditional debugger in that it does not traverse your program code line by line (since you do not write any code with MapForce). Instead, the debugger exposes the results of MapForce-generated code produced from the mappings you design. More specifically, the debugger logs values that are passed from and to mapping components through their input and output connectors. The logged values are then available for your analysis directly on the mapping or through dedicated windows.

The following sections highlight various ways in which you can use the mapping debugger.

Debug with breakpoints

When you need to stop the debugging execution at a particular place in the mapping, you can set breakpoints, similar to how you would do that in a traditional development environment. The difference is that breakpoints are added not to a line of code, but to an input or output connector of a mapping component. You can also add conditions to breakpoints (this can be useful if you want to stop the execution only if the set condition is satisfied).

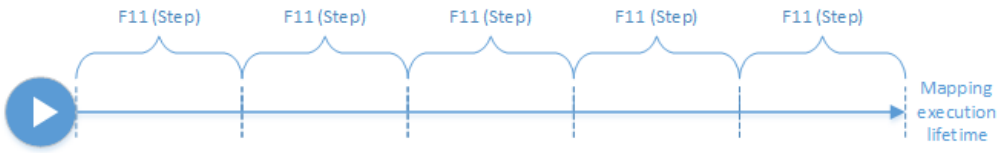


You can define breakpoints on the desired connectors and execute the mapping up to the

first encountered breakpoint, then go to the next one, and so on. This way you can analyze the mapping context and values associated with chosen connectors. You can also speed up or slow down the execution by means of the **Step Into**, **Step Out**, **Step Over**, and **Minimal Step** commands provided by the debugger. These commands enable you to skip portions of the mapping, or, on the contrary, execute portions of the mapping in a more granular way if necessary.

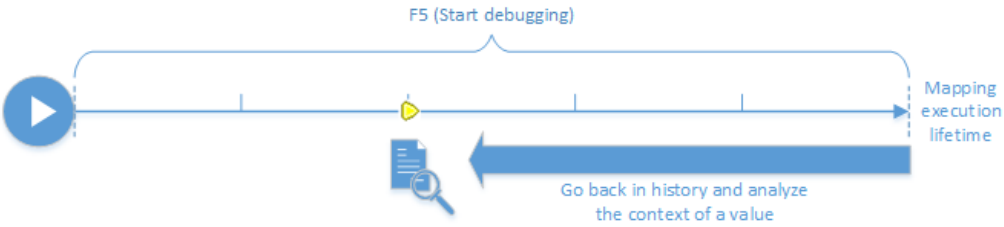
Debug step-by-step

You can debug a mapping step-by-step, and analyze the mapping context and values associated with each step. This scenario is similar to the previous one, in that you can speed up or slow down execution using the **Step Into**, **Step Out**, **Step Over**, and **Minimal Step** commands.



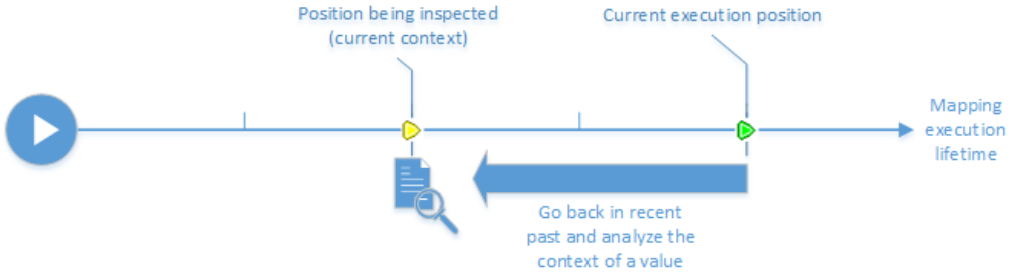
Analyze the log of values

You can configure MapForce to remember the log of all values (trace history) that were processed by all connectors while you debug a mapping. Keeping the full trace history may not be suitable for mappings that are data-intensive, so this option can be disabled if necessary. When the option is enabled, you can analyze the full log of values processed by each connector up until the current execution position. You can also instruct MapForce to recreate the mapping context associated with any particular value, which would help you understand why that value was produced.



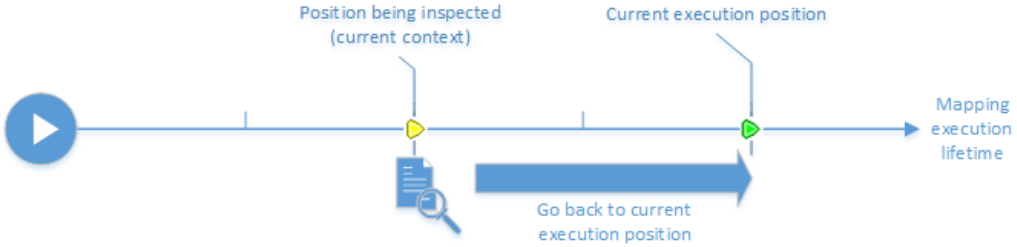
Set the context to a value related to the current execution position

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

- Currently, the mapping debugger is not supported in the MapForce plug-in for Visual Studio or Eclipse.
- When MapForce executes a mapping, it may internally optimize code (for example, by caching data, or by calculating intermediate results at arbitrary points). This may cause certain connectors (and thus breakpoints) to be unreachable for debugging, in which case MapForce displays a notification. Note that the MapForce code optimizations (and, consequently, the behavior exposed by the debugger) may be different from one MapForce release to the other, even though the mapping output is the same for a given mapping.
- The debugger can debug the output generation for one target component at a time. If there are multiple target components on the mapping, you will need to select which one should be executed by the debugger.
- Currently, debugging is not supported for the database table actions (such as "Insert All", "Update If", etc.) of database components.
- Breakpoints cannot be added on any of the following entities: constants, the `core | position` function, descendent items of "Copy-all" connections, parameters of "inline" user-defined functions.

6.1 Debugger Preparation

Debugging preparation is primarily required for big data mappings that are likely to need a lot of system memory to execute. This is the case of mappings that either process very big input or output files, or repeatedly iterate through large collections of data.

To make debugging faster and reduce memory requirements, it is recommended to do the following before you start debugging:

- If the mapping is complex, remove or disconnect parts of the mapping that need not be debugged.
- If the mapping uses big input files, replace them with files of smaller size.
- Ensure that the **Keep full trace history** option is disabled (see Debugger Settings)

Also, to ensure you are debugging the right output, check the following if applicable:

- If the mapping has multiple target components, select the target component to be debugged by clicking the **Preview** button ().
- If the mapping is a chained mapping (see Chained Mappings), release the **Pass-Through** () button on the intermediary component. Debugging Pass-Through components is currently not supported.

Optionally, if you want the debugger to stop at some important connectors whose value you want to analyze, add breakpoints to these connectors (see Adding and Removing Breakpoints).

6.2 Debugger Commands

You can access the debugger commands as follows:

- In the **Debug** menu
- As keyboard shortcuts
- In the Debug toolbar.

Menu Command	Keyboard Shortcut	Toolbar button	Description
Debug | Start debugging	**F5**		Starts or continues debugging until a breakpoint is hit or the mapping finishes.
Debug | Stop debugging	**Shift + F5**		Stops debugging. This command exits the debug mode and switches MapForce back to standard mode.
Debug | Step Into	**F11**		Executes the mapping until a single step is finished anywhere in the mapping. In the mapping debugger, a step is a logical group of dependent computations which normally produce a single item of a sequence. Depending on the mapping context, this command roughly translates into "go to the left/go to target child/go to source parent".
Debug | Step Over	**F10**		Continues execution until the current step finishes (or finishes again for another item of the sequence), or an unrelated step finishes. This command steps over computations that are inputs of the current step.
Debug | Step Out	**Shift + F11**		Continues execution until the result of the current step is consumed or a step is executed that is not an input or child of the consumption. This command steps out of the current computation. Depending on the mapping context, this command roughly translates into "go to the right/go to target parent/go to source child".
Debug | Minimal Step	**Ctrl + F11**		Continues execution until a value is produced or consumed. This command subdivides a step and will typically stop twice for each connection: once when its source produces a value and once when its target consumes it. MapForce does not

Menu Command	Keyboard Shortcut	Toolbar button	Description
			necessarily compute values in the order the mapping would suggest, so production and consumption events do not always follow each other.

Debug toolbar

6.3 About the Debug Mode

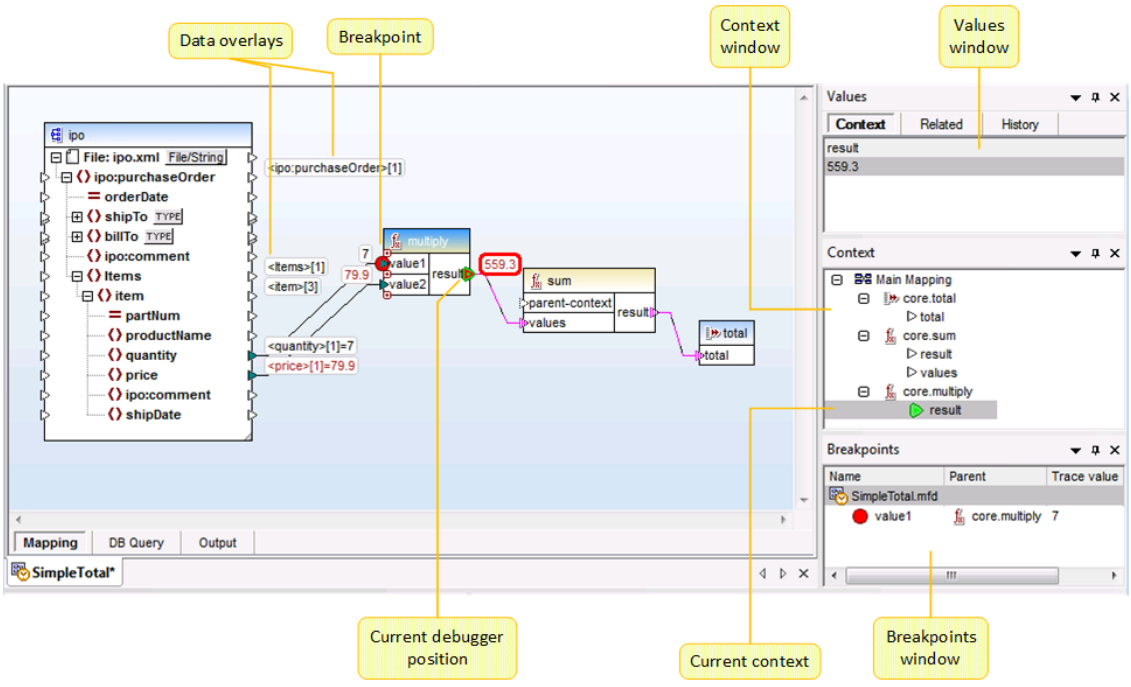
When you start debugging (by pressing **F5**, or **F11**, or **Ctrl + F11**), MapForce executes the mapping in debug mode.

> While MapForce is in debug mode, the mapping is read-only. Although you can move components on the mapping area, most commands are not available. This includes commands such as mapping validation and deployment, code generation, documenting mappings, adding new components to the mapping area or reloading existing ones, and others.

The debug mode enables you to analyze the context responsible for producing a particular value. This information is available directly on the mapping, as well as in the Values, Context, and Breakpoints windows. By default, these windows are displayed when you start debugging and are hidden when you stop debugging.

MapForce is in debug mode (and the mapping is read-only) until you stop debugging, by pressing **Shift + F5** (or by clicking the **Stop debugging** toolbar button).

The following image illustrates a sample mapping (**SimpleTotal.mfd**, from the **<Documents>\Altova\MapForce2016\MapForceExamples** directory) that is debugged in steps (by pressing **F11** to advance a step).



The MapForce development environment in debug mode

The visual clues and other information provided by MapForce while in debug mode are described below.

The mapping pane

While debugging, the mapping pane displays additional information:

- Data overlays (see below) show the current value and related values near their connectors.
- The current context (shown as a structure in the Context window) is highlighted as follows:
  - Connectors in the context are striped magenta ( ).
  - Connectors in ambiguous context are dotted magenta ( ).
  - Connections in the context are striped magenta.
  - Connections in ambiguous context are striped magenta but lighter.
- The current execution location is displayed with a green connector icon ( ).

Data overlays

The values processed by each connector are displayed as data overlays (small rectangles) near their corresponding connector. A currently selected data overlay is displayed with thick red border. Values changed from the last step are displayed in dark red. For nodes with simple content, the data overlay combines two values - the node name and the value. If the node name has been iterated multiple times before the current execution position, the index of the current iteration is indicated by the number in square brackets.

Data overlays have the following behavior:

- Pointing the mouse to a data overlay brings it temporarily to the foreground, clicking it does it permanently. Clicking also selects the corresponding connector.
- Data overlays can be moved by dragging.
- Data overlays move when a component is moved. Therefore, if the data overlays appear stacked because the components are too close to each other, drag the components around the mapping area to make more space, and the data overlays will move together with the component.
- Clicking a data overlay shows its value in the Values window.
- Clicking a connector also selects its data overlay.

Breakpoints

Breakpoints are designated milestones at which the mapping should break during execution in debug mode. This term may be already familiar to you by analogy with other integrated development environments. Unlike other development environments where you add breakpoints to a line of code, a breakpoint in MapForce can be added to an input or output connector (small triangle to the left or right of the connection). On the mapping pane, breakpoints are represented as red circles. Any defined breakpoints are also displayed in the Breakpoints window. See also Adding and Removing Breakpoints.

Current debugger position

The green triangle () indicates the position of the debugger. This position is either an input or an output connector of any given component.

The value currently being processed is also displayed in the Values window, on the **Context** tab.

The set of connections and/or connectors colored in striped magenta indicate the current mapping context. The same information is also displayed as a hierarchical structure in the Context window (see Using the Context Window).

When you set manually the context of a value, the current debugger position is in a position in the past relative to the most current execution position. To help you distinguish between the most current execution position and the one in the past, the "current position" connector may appear with the following colors in the debugger interface.

	Green is "the present"; it indicates the current execution position (see Viewing the Current Value of a Connector).
	Yellow is "the past"; it indicates that you are looking at some connector in the past, relative to the current execution position. This may happen after you set a context manually (see Setting the Context to a Value).

Values window

The Values window provides information about the values processed by the mapping. It enables you to see what the mapping processes at the current execution position, or in a particular context that you can set yourself. See also Using the Values Window.

Context window

The Context window provides a hierarchical view of the set of nodes and functions that are relevant for the current debugger position. See also Using the Context Window.

Breakpoints window

The Breakpoints window displays the list of debugging breakpoints created since MapForce was started. If you have defined breakpoints on multiple mappings, all of them appear in the Breakpoints window. See also Using the Breakpoints Window.

6.4 Adding and Removing Breakpoints

Breakpoints are designated milestones at which the mapping should break during execution in debug mode. Any breakpoints you create are stored globally for all mappings and are displayed in the Breakpoints window. Breakpoints are valid until you either explicitly delete them, or close MapForce.

Note: Breakpoints cannot be added on any of the following entities: constants, the `core | position` function, descendent items of "Copy-all" connections, parameters of "inline" user-defined functions.

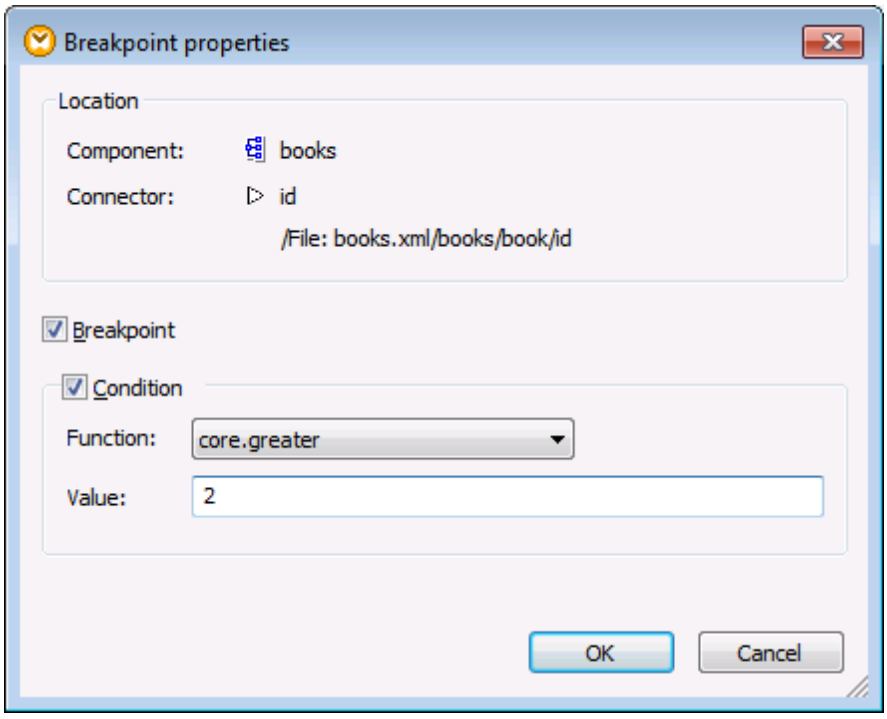
Breakpoints can be simple or conditional. Simple breakpoints stop the mapping execution unconditionally. Conditional breakpoints stop the mapping execution only when the condition assigned to them is satisfied. Conditions take the form of MapForce built-in library functions to which you supply custom values. In other words, if the condition returns true, the breakpoint will stop the mapping execution.

To create a simple breakpoint, do one of the following:

- Right-click an input or output connector (the small triangles to the left or right of a component), and select **Debugger Breakpoint**.
- Click an input or output connector, and then press **F9**.

To create a conditional breakpoint:

1. Right-click a connector, and select **Breakpoint properties**.



2. Click to select both the **Breakpoint** and **Condition** check boxes.
3. Select the required function from the list, and enter the function value (if applicable). For example, in the example above, the breakpoint will stop the mapping execution if the value passing through it is greater than 2.

> If the data type of the connector where you add the conditional breakpoint does not match the type(s) expected by the function, MapForce will attempt to convert the data type automatically. If automatic conversion is not possible, mapping execution will fail. To avoid this, make sure to use compatible data types. For example, the function `core.starts-with` expects a string value, so the breakpoint's connector must have the same type.

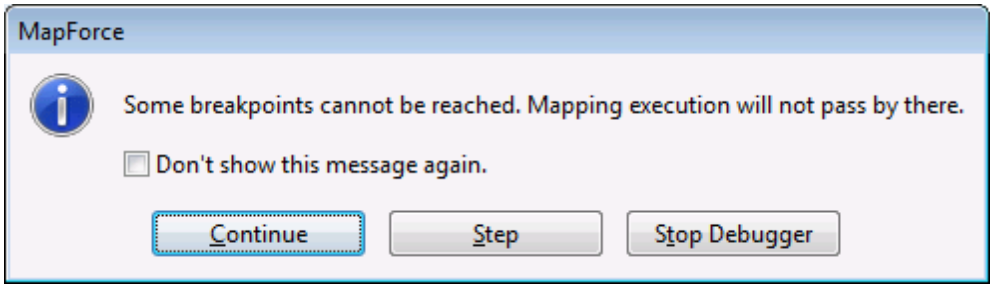
Removing breakpoints

To remove a breakpoint, right-click the connector on which the breakpoint exists, and select **Debugger Breakpoint**. Alternatively, click the input or output connector on which the breakpoint exists, and then press **F9**.

You can also remove breakpoints from the Breakpoints window (see Using the Breakpoints Window).

Unreachable breakpoints

There may be cases when MapForce displays a "Breakpoints cannot be reached" message:



This indicates that breakpoints cannot be reached by the debugger, because of one of the following reasons:

- A breakpoint has been defined on a connector that does not take part in the mapping.
- The breakpoint cannot be reached by MapForce because of execution optimizations (see Limitations).

Click **Continue** to advance to the next defined breakpoint (or go to the end of debugging execution). Click **Step** to start debugging in steps.

You can disable notifications about unreachable breakpoint encountered by the debugger, either by clicking **Don't show this message again**, or as follows:

1. On the **Tools** menu, click **Options**.
2. Click **Messages**.

3. Click to clear the **Inform about unreachable breakpoints** check box.

6.5 Using the Values Window

The Values window displays information about the values processed by the mapping when in debug mode. The information displayed in the Values window depends on the current debugger position, and on the user interface elements that you clicked. The Values window contains the following tabs:

The "Context" tab

The **Context** tab displays the value currently being processed (the same value whose context is shown in the Context window). This is either the value at the current execution position of the debugger, or the value of a connector processed in the past. MapForce helps you distinguish between the two using colors:

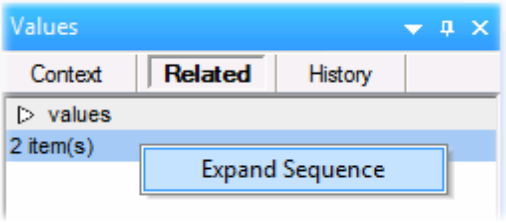
	Green is "the present"; it indicates the current execution position (see Viewing the Current Value of a Connector).
	Yellow is "the past"; it indicates that you are looking at some connector in the past, relative to the current execution position. This may happen after you set a context manually (see Setting the Context to a Value).

The "Related" tab

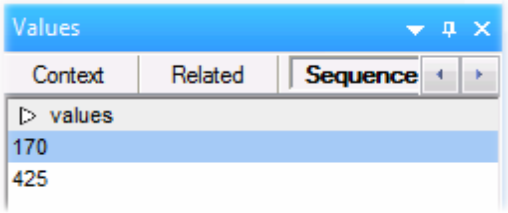
The **Related** tab displays values that are related to (or represent the "near past" of) the currently processed value. Normally, you do need to explicitly click this tab; MapForce switches to it automatically when you click the data overlay of a connector that is related to the current execution position of the debugger. See Stepping back into Recent Past.

The "Sequence" tab

When present, the **Sequence** tab enables you to get access to the values of a connector that processes a sequence. This tab is visible only when a connector has processed a sequence of items (for example, an aggregate function such as `sum` or `count` does that). When you click the data overlay of a connector that processed a sequence of items, the Values window displays an entry in the format "**n items**", where **n** is the number of items processed by the connector. To get access to each value, double-click this entry (or right-click it, and select **Expand Sequence** from the context menu).



The values are then displayed in the **Sequence** tab.



The "History" tab

The **History** tab displays values have been processed by a particular node since debugging started and up to the current execution position. See Viewing the History of Values Processed by a Connector.

6.6 Using the Context Window

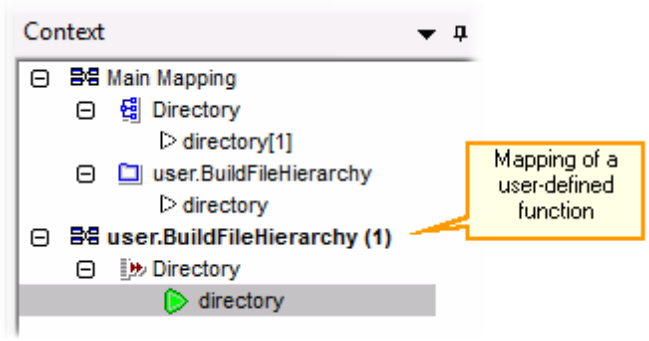
While MapForce is in debug mode, the Context window displays a structure of connectors that are relevant to the current position of the debugger. In other words, it provides the mapping context responsible for producing the current mapping value.

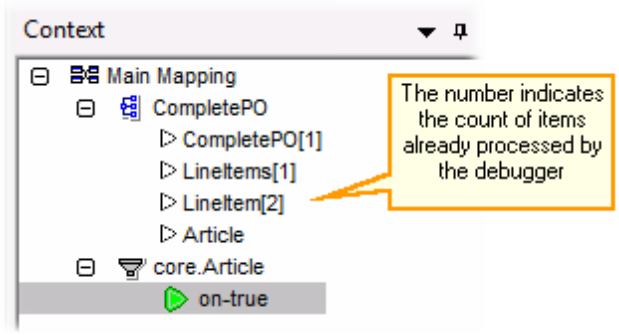
MapForce builds the current context as follows:

1. Start with the root node of the target structure.
2. Descend to the current target node.
3. From the current target node, move left inside the mapping through any components that lead to the current position. These components may be filter or sort components, built-in or user-defined functions, variables, and so on.

The Context window serves both as informational and as a navigational aid. To select a particular node in the mapping directly from the current context, right-click the node in the Context window, and click **Select in mapping**. This might be especially useful when the mapping is large, so as to avoid extensive scrolling.

The Context window may display the following special icons and notation:

Icon	Description
	Represents the mapping to which the context belongs. This can be either the main mapping or the mapping of a user-defined function. 
▷	Represents a connector. The target nodes processed so far have their position displayed in square brackets.

Icon	Description
	
	Represents the current connector (the most recent execution position). This is the source of the current value in the Values window. In some rare situations, it is possible that a computed value is used for multiple connectors. In this case, multiple green icons may appear.
	Represents the current connector when the debugger is at some position in the past relative to the most recent execution post. This may happen after you set the context to a value (see Setting the Context to a Value).

In addition to the icons above, the Context window includes the standard icons of any component types that are present in the mapping.

Context window and user-defined functions

If the current context includes any user-defined functions, they are displayed in the Context window as well. Note that if the current context is for computing an input value of a user-defined function, the context is determined as follows:

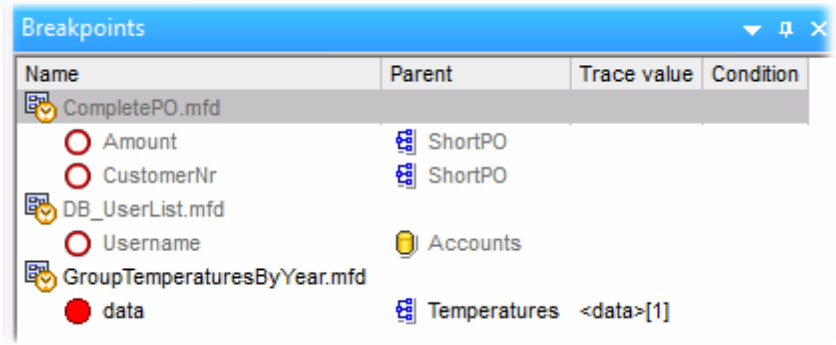
1. From the target to the output connector of the user-defined function to the input connector of the user-defined function
2. From there further to the left.

Note: A user-defined function may occur multiple times in the context. This happens either because several function calls are chained or because the user-defined function is defined as recursive.

6.7 Using the Breakpoints Window

The Breakpoints window enables you to view and manage breakpoints globally. By default, the Breakpoints window is displayed when MapForce is in debug mode. To make the Breakpoints window visible at all times, select the menu command **View | Debug Windows | Breakpoints**.

The Breakpoints window displays all breakpoints created since you started MapForce, grouped by the mapping file to which they belong. While MapForce is open, any breakpoints associated with any mapping are "remembered" by MapForce and displayed in the Breakpoints window, even if you closed the mapping file in the meanwhile. The mapping that is currently being debugged is represented with standard text color in the Breakpoints window, while other mappings (the ones that are closed or not active) are grayed out.



You can quickly open any mapping by double-clicking it (or any of its breakpoints) in the Breakpoints window.

Note: Once you close or restart MapForce, all breakpoints are removed.

Information about breakpoints is displayed as a grid with the following columns:

Column	Description
Name	The name of the node where the breakpoint belongs.
Parent	The name of the mapping component where the breakpoint belongs.
Trace value	The value that passes through the connector on which the breakpoint is. The trace value is displayed during debugging execution.
Condition	If the breakpoint is conditional, this column displays the condition of the breakpoint.

Breakpoints may be associated with any of the following icons.

Icon	Description
●	Active breakpoint. Denotes a breakpoint from the mapping that is currently being debugged.

Icon	Description
	Inactive breakpoint. Denotes a breakpoint from a mapping that is open, but is not currently being debugged.
	Inaccessible breakpoint. Denotes a breakpoint that cannot be reached by the debugger.
	Conditional breakpoint. Denotes a breakpoint with a condition attached to it.

To view or change the properties of a breakpoint:

- Right-click it, and select **Breakpoint Properties** from the context menu.

To delete a breakpoint:

- Right-click the breakpoint you want to delete, and then select **Delete Breakpoint** from the context menu.
- Click a breakpoint, and then press **Delete**.

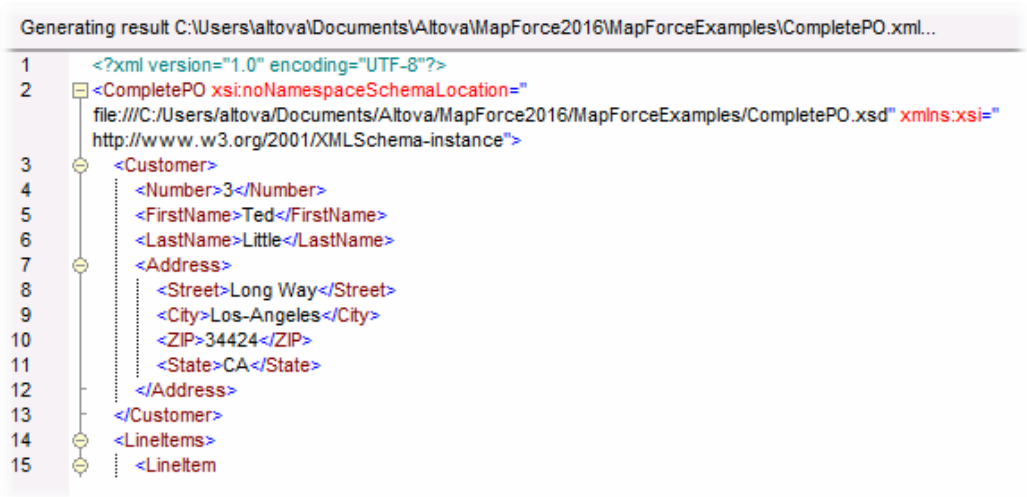
The context command **Delete All Breakpoints** removes all breakpoints displayed in the Breakpoints window, regardless of the mapping where they belong.

See also: Adding and Removing Breakpoints

6.8 Previewing Partially Generated Output

When you are debugging in steps or using breakpoints, you can view the mapping output generated up to the current debugger position. Previewing partially generated output is supported by XML, flat text, and EDI target components.

By default, when you press **F5** (without having defined any breakpoints), MapForce executes the entire mapping in debug mode, and then switches to the **Output** tab, displaying the final generated output. However, if you have defined breakpoints, or if you are debugging in steps (**F11**, or **Ctrl + F11**), the debugger execution stops while the mapping output is still being generated. Even if the mapping output is partially written at this stage, you can still click to the **Output** tab, and preview it.



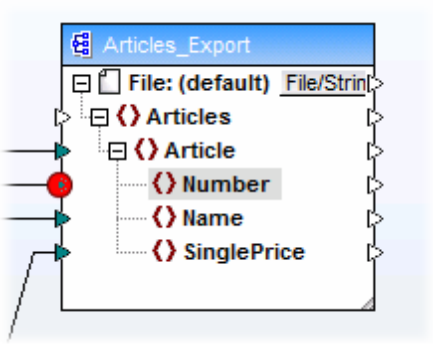
Limitations

- The currently computed target node is not always displayed in the Output tab. For example, XML attributes are collected internally and written at once.
- If the output produces multiple files, only the currently written file can be displayed; switching to another output file is disabled.

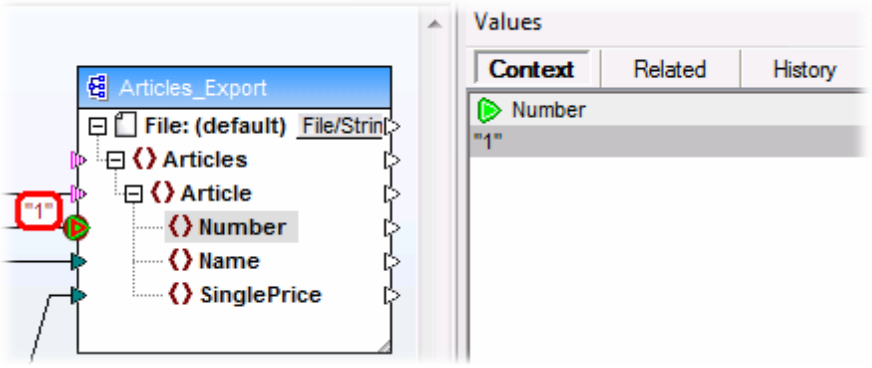
6.9 Viewing the Current Value of a Connector

When the current execution position of the debugger ( ) is on a particular connector (either because you are debugging in steps, or because there is a breakpoint defined on the connector), the current value processed by the connector is displayed in the **Context** tab of the Values window. This is the value that is about to be written to the output, that is, "the present". It is also the value whose context is displayed in the Context window (see Using the Context Window).

To understand this case, open the **PreserveFormatting.mfd** sample from the **<Documents>\Altova\MapForce2016\MapForceExamples** directory. Click the input connector of the `Number` node on the target component, and press **F9** to add a breakpoint on it.



Then press **F5** to start debugging and observe the results.



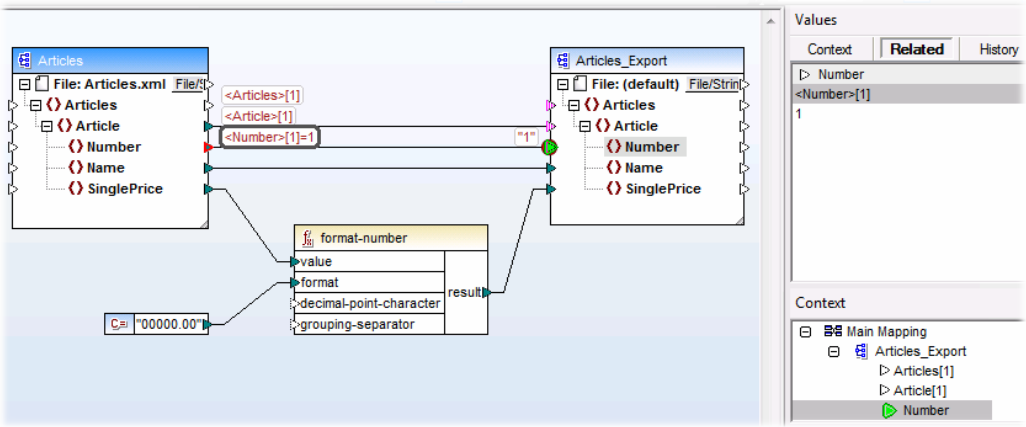
As shown in the image, the current debugger position (and the breakpoint) is on the `Number` node of the target component. The Values window indicates that this node processes the value "1" (this value is also highlighted with a thick red border on the mapping).

6.10 Stepping back into Recent Past

When you click a data overlay (small rectangular box) next to a mapping connector, the **Values** window displays the name and, optionally, the value associated with the selected connector. The focus now is no longer on the current debugger position, but on the selected data overlay. You can consider this view as stepping slightly back in the debugging history. This is the "near" past, since the mapping displays data overlays only for the last few connectors related to the current debugger position. When you click such a "related" data overlay, the Values window switches automatically to the **Related** tab.

For an illustration of this scenario, open the mapping **PreserveFormatting.mfd** from the **<Documents>\Altova\MapForce2016\MapForceExamples** directory.

After opening the mapping, click the connector next to the `Number` node on the target component, and press **F9** to add a breakpoint on it. Press **F5** to start debugging, and then click the data overlay (small rectangular box) next to the `Number` node of the source component.



Because a connector is typically iterated multiple times for the lifetime of a mapping, the current index of the iteration is displayed enclosed with square brackets: **<Number>[1]**. Also, because the connector carries a value, its value is also represented after the equal sign: **<Number>[1]=1**. The same value is displayed on a new row in the Values window, as shown below.

If you need additional information about a particular value, remember that you can recreate the context that produced it (see Setting the Context to a Value).

6.11 Viewing the History of Values Processed by a Connector

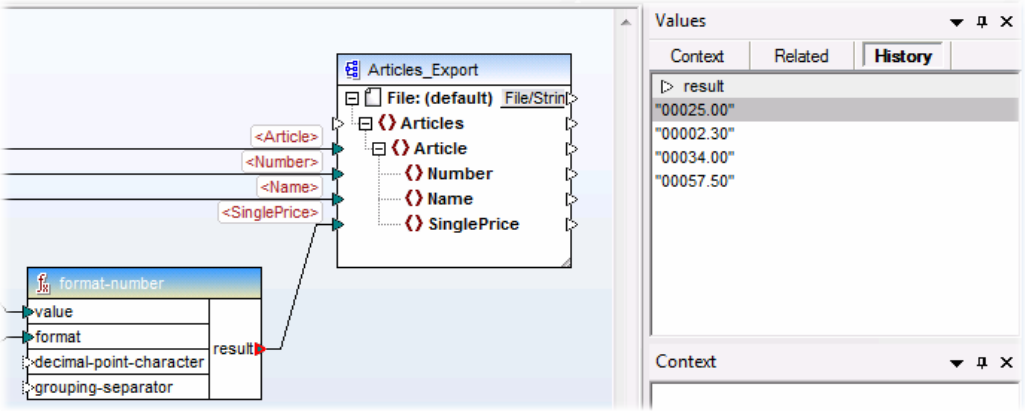
If the option **Keep full trace history** is enabled (see Debugger Settings), you can view the history of all values that were processed by that connector (up to the current execution position).

The history is displayed when you click a connector, and then click the **History** tab of the Values window. Note that this operation is meaningful only for connectors that have processed values since the beginning of mapping execution until the current debugger position.

To illustrate this case, let's debug a mapping from begging till end without using any breakpoints, and then watch the history of all values that were processed by a particular connector. First, open the mapping **PreserveFormatting.mfd** from the **<Documents>\Altova\MapForce2016 \MapForceExamples** directory. If it is already open, make sure to do the following:

- Clear any breakpoints, if such exist (see Adding and Removing Breakpoints)
- Stop debugging if it is currently in progress, by pressing **Shift + F5**.

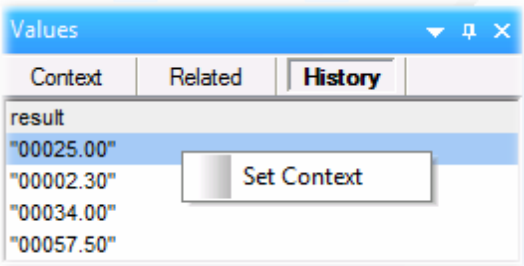
When ready, press **F5** start a new debugging operation. When you press **F5**, MapForce executes the mapping in debug mode, and switches to the **Output** tab. Click the **Mapping** tab to go back to the main mapping window, and then click the `result` node of the `format-number` function (highlighted in red in the image below). Finally, click the **History** tab of the Values window, and notice the displayed values.



As shown in the image above, this particular node (`result`) has processed four values in total. If you need additional information about a particular value, remember that you can recreate the context that produced it (see Setting the Context to a Value).

6.12 Setting the Context to a Value

Setting the context to a value is an action that can be compared to stepping into the past, in order to view more details about the mapping context that produced that value. You can set the context to any value displayed in the Values window (in the **Related** tab, **Sequence** tab, or **History** tab). If you have enabled the **Keep full trace history** option (see Debugger Settings), the **History** tab displays all values processed by the currently selected connector; therefore, in this case, you can additionally set the context to any value in the past for that connector.

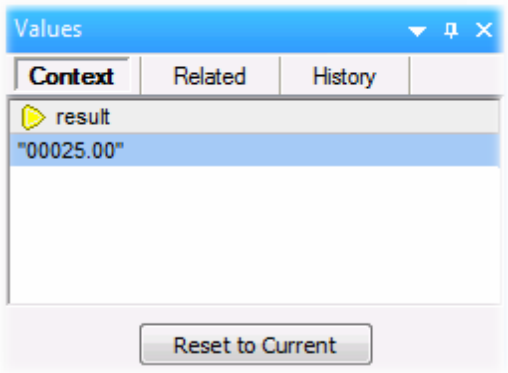


To set the context to a value, do one of the following:

- Right-click the value, and select **Set Context** from the context menu.
- Double-click the value.

When you set the context to a value, MapForce highlights the mapping area so as to recreate the situation that produced that value, and populates the **Values** window and the **Context** window according to the selected context. For a legend to visual clues used on the mapping area while in a context, see About the Debug Mode. For information about the context itself, see Using the Context Window.

The connector of a manually-set context is yellow ( ), which indicates that you are no longer at the most recent execution position. To switch back to the most recent execution position (when applicable), click the **Reset to Current** button on the **Context** tab of the Values window.



6.13 Debugger Settings

To access the settings applicable to the MapForce debugger, select the menu command **Tools | Options**, and then click **Debugger**. The available settings are as follows:

Maximum storage length of values

Defines the string length of values displayed in the Values window (at least 15 characters). Note that setting the storage length to a high value may deplete available system memory.

Keep full trace history

Instructs MapForce to keep the history of all values processed by all connectors of all components in the mapping for the duration of debugging. If this option is enabled, all values processed by MapForce since the beginning of debug execution will be stored in memory and available for your analysis in the Values window, until you stop debugging. It is not recommended to enable this option if you are debugging data-intensive mappings, since it may slow down debugging execution and deplete available system memory. If this option is disabled, MapForce keeps only the most recent trace history for nodes related to the current execution position.

Chapter 7

Data Sources and Targets

7 Data Sources and Targets

This section describes the various source and target component types that MapForce can map from/to.

- XML and XML schema
- Databases and MapForce
- Mapping CSV and Text files
- MapForce Flextext
- EDI - UN/EDIFACT, ANSI X12, HL7, IATA PADIS, SAP IDoc
- JSON
- Microsoft OOXML Excel 2007+
- XBRL
- HL7 v3.x to/from XML schema

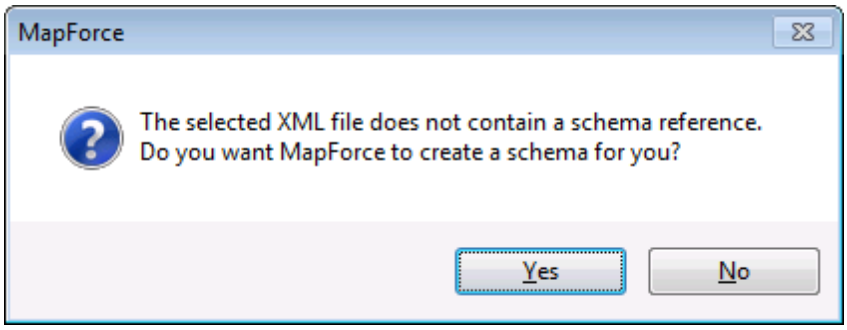
7.1 XML and XML schema

In the introductory part of this documentation, you have seen examples of simple mappings that use XML and XML schema files as source or target components. This section provides further information about using XML components in your mappings. It includes the following topics:

- XML Component Settings
- Using DTDs as "schema" components
- Derived XML Schema types - mapping to
- QName support
- Nil Values / Nillable
- Comments and Processing Instructions
- CData sections
- Wildcards - xs:any

7.1.1 Generating an XML Schema

MapForce can automatically generate an XML schema based on an existing XML file if the XML Schema is not available. When you add to the mapping area an XML file without a schema, the following dialog box appears.

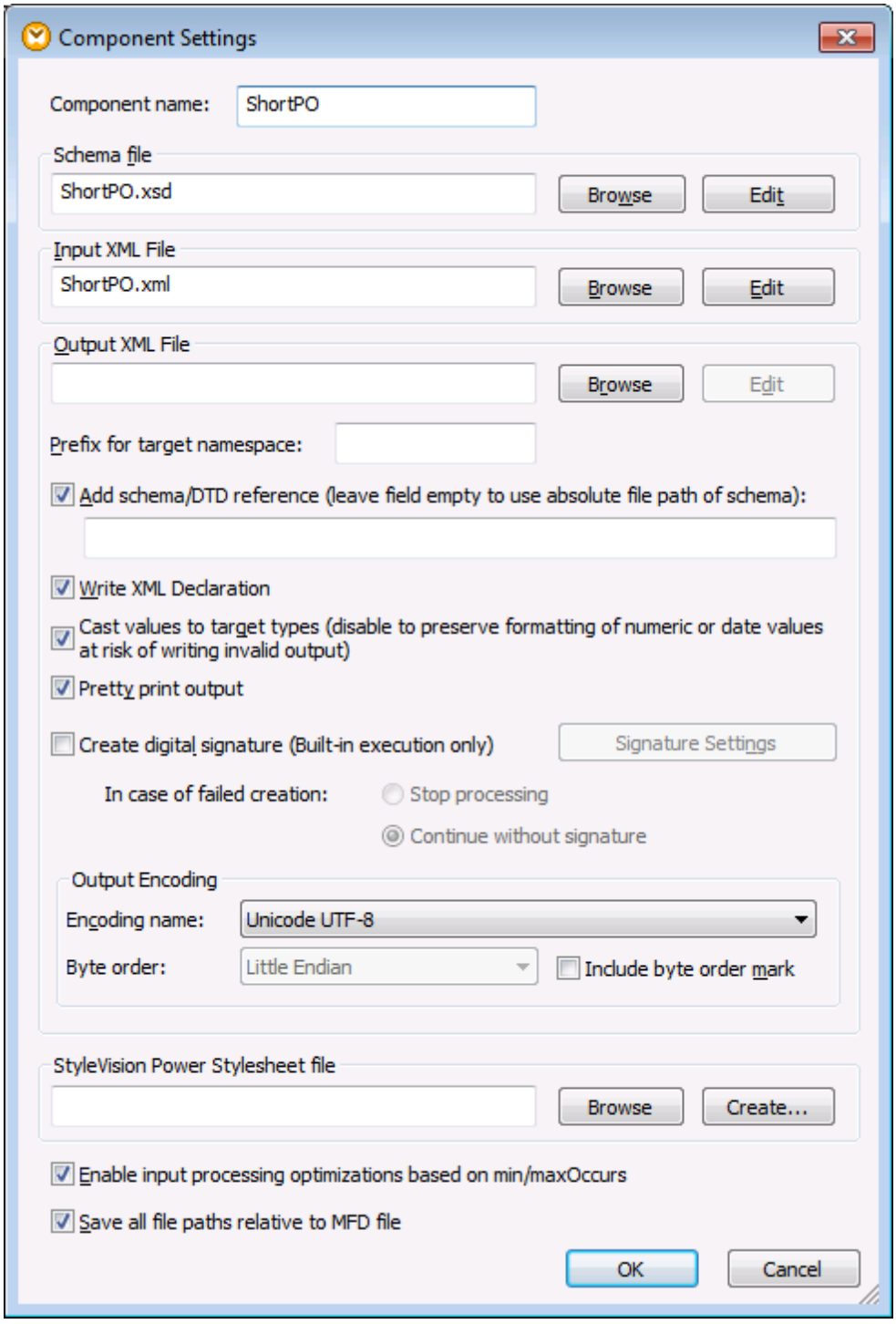


When MapForce generates a schema from an XML file, data types for elements/attributes must be inferred from the XML instance document and may not be exactly what you expect. It is recommended that you check whether the generated schema is an accurate representation of the instance data.

7.1.2 XML Component Settings

After you add an XML component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component on the mapping, and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.



XML Component Settings dialog box

The available settings are as follows.

Component name	The component name is automatically generated when you

	create the component. You can however change the name at any time. If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well. The component name can contain spaces (for example, "Source XML File") and full stop characters (for example, "Orders.EDI"). The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications when changing the name of the component: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have different encodings in Windows and at the command line.
Schema file	Specifies the name or path of the XML schema file used by MapForce to validate and map data. To change the schema file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.
Input XML file	Specifies the XML instance file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it an XML instance file. In a source component, the instance file name is also used to detect the XML root element and the referenced schema, and to validate against the selected schema. To change the location of the file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.
Output XML file	Specifies the XML instance file to which MapForce will write data. This field is meaningful for a target component. To change the location of the file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.
Prefix for target namespace	Allows you to enter a prefix for the target namespace. Ensure that the target namespace is defined in the target schema, before assigning the prefix.
Add schema/DTD reference	Adds the path of the referenced XML Schema file to the root element of the XML output. The path of the schema entered in this field is written into the generated target instance files in the `xsi:schemaLocation` attribute, or into the `DOCTYPE`

<table>
<tr><td></td><td>declaration if a DTD is used.

Note that, if you generate code in XQuery or C++, adding the DTD reference is not supported.

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

Deactivating this option allows you to decouple the XML instance from the referenced XML Schema or DTD (for example, if you want to send the resulting XML output to someone who does not have access to the underlying XML Schema).</td></tr>
<tr><td>Write XML declaration</td><td>This option enables you to suppress the XML declaration from the generated output. By default, the option is enabled, meaning that the XML declaration is written to the output.

This feature is supported as follows in MapForce target languages and execution engines.

<table>
<tr><th>Target language / Execution engine</th><th>When output is a file</th><th>When output is a string</th></tr>
<tr><td>Built-in</td><td>Yes</td><td>Yes</td></tr>
<tr><td>MapForce Server</td><td>Yes</td><td>Yes</td></tr>
<tr><td>XSLT, XQuery</td><td>Yes</td><td>No</td></tr>
<tr><td>Code generator (C++, C#, Java)</td><td>Yes</td><td>Yes</td></tr>
</table>
</td></tr>
<tr><td>Cast values to target types</td><td>Allows you to define if the target XML schema types should be used when mapping, or if all data mapped to the target component should be treated as 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.</td></tr>
</table>

	Note that disabling this option will also disable the detection of invalid values, e.g. writing letters into numeric fields.
Pretty print output	Reformats the output XML document to give it a structured look. Each child node is offset from its parent by a single tab character.
Create digital signature	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 transformation language (see also Digital Signatures).
Output Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab. If the mapping generates XSLT 1.0/2.0, activating the **Byte Order Mark** check box does not have any effect**,** as these languages do not support Byte Order Marks.
StyleVision Power Stylesheet file	This option allows you to select or create an Altova StyleVision stylesheet file. Such files enable you to output data from the XML instance file to a variety of formats suitable for reporting, such as HTML, RTF, and others. See also Using Relative Paths on a Component.
Enable input processing optimizations based on min/ maxOccurs	This option allows special handling for sequences that are known to contain exactly one item, such as required attributes or child elements with `minOccurs` and `maxOccurs="1"`. In this case, the first item of the sequence is extracted, then the item is directly processed as an atomic value (and not as a sequence). If the input data is **not valid** against the schema, an empty sequence might be encountered in a mapping, which stops the mapping with an error message. To allow the processing of such **invalid input**, disable this check box.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative Paths on a Component.

7.1.3 Using DTDs as "Schema" Components

Starting with MapForce 2006 SP2, namespace-aware DTDs are supported for source and target components. The namespace-URIs are extracted from the DTD "xmlns"-attribute declarations, to make mappings possible.

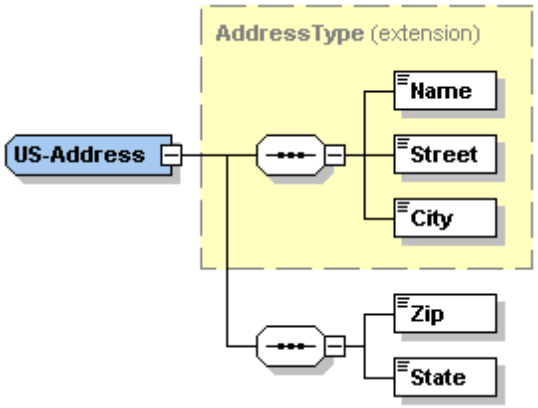
However, some DTDs contain xmlns*-attribute declarations without namespace-URIs (for example, DTDs used by StyleVision). Such DTDs have to be extended to make them useable in MapForce. Specifically, you can make such DTDs useable by defining the xmlns-attribute with the namespace-URI, as shown below:

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

7.1.4 Derived XML Schema Types

MapForce supports the mapping to/from derived types of a complex type. Derived types are complex types of an XML Schema that use the **xsi:type** attribute to identify the specific derived types.

The screenshot below shows the definition of a derived type called `US-Address`, in XMLSpy. The base type (or originating complex type) is `AddressType`. Two extra elements were added to create the derived type `US-Address`: `Zip` and `State`.

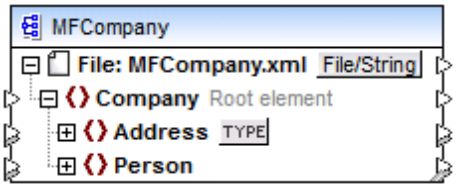


Sample derived type (XMLSpy schema view)

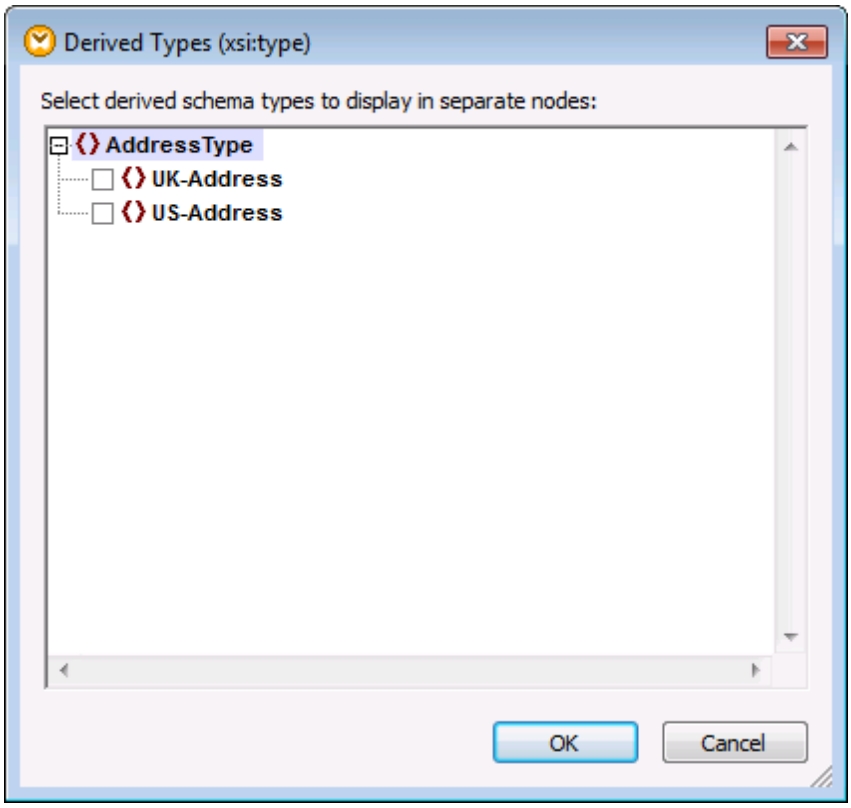
The following example shows you how to map data to or from derived XML schema types.

1. On the **Insert** menu, click **XML Schema/File**, and open the following XML Schema: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial \MFCompany.xsd**.
2. When prompted to supply an instance file, click **Skip**, and then select `Company` as the

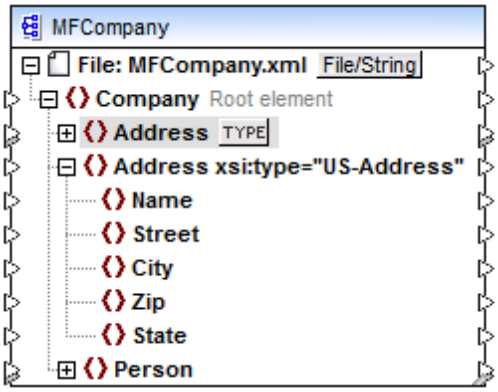
root element.



3. Click the TYPE button next to the `Address` element. This button indicates that derived types exist for this element in the schema.



4. Select the check box next to the derived type you want to use (`US-Address`, in this case), and confirm with OK. A new element `Address xsi:type="US-Address"` has been added to the component.



You can now map data to or from the `US-Address` derived type.

Note that you can also include multiple derived types by selecting them in the Derived Types dialog box. In this case, each would have its own `xsi:type` element in the component.

7.1.5 QNames

MapForce resolves QName (qualified name) prefixes (http://www.w3.org/TR/xml-names/#ns-qualnames) when reading data from XML files at mapping execution run-time.

QNames are used to reference and abbreviate namespace URIs in XML and XBRL instance documents. There are two types of QNames: Prefixed and Unprefixed QNames.

PrefixedName	Prefix	':'	LocalPart
UnPrefixedName			LocalPart where LocalPart is an Element or Attribute name.

For example, in the listing below, `<x:p/>` is a QName, where:

- the prefix "x" is an abbreviation of the namespace "**http://myCompany.com**".
- `p` is the local part.

```
<?xml version='1.0'?>
<doc xmlns:x="http://myCompany.com">
   <x:p/>
</doc>
```

MapForce also includes several QName-related functions in the **xpath2 | qname-related functions** library. The **lang | QName** library (available in the MapForce Professional and Enterprise Editions) additionally includes QName-related functions that can be used in code generation languages and the BUILT-IN transformation language.

7.1.6 Nil Values / Nillable

The XML Schema specification allows for an element to be valid without content if the `nillable="true"` attribute has been defined for that specific element in the schema. In the instance XML document, you can then indicate that the value of an element is nil by adding the `xsi:nil="true"` attribute to it. This section describes how MapForce handles nil elements in source and target components.

'xsi:nil' versus 'nillable'

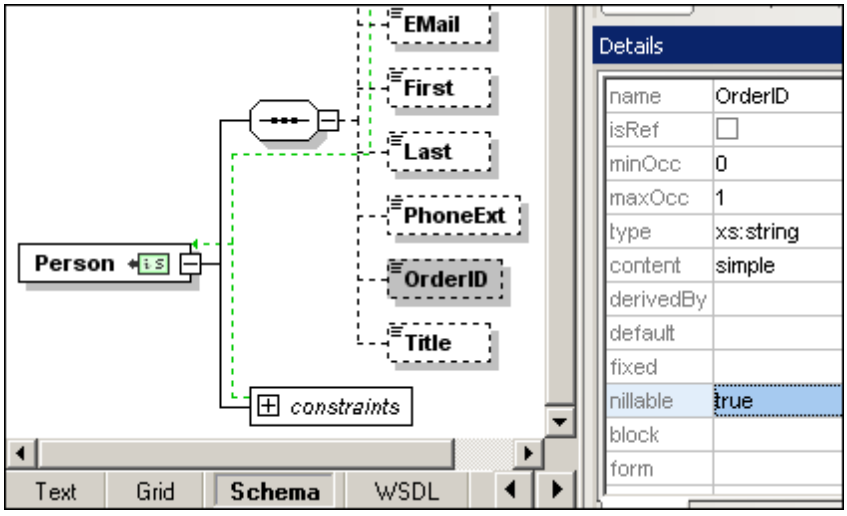
The `xsi:nil="true"` attribute is defined in the XML **instance** document.

```
14  <Person>
15    <PrimaryKey>2</PrimaryKey>
16    <ForeignKey>1</ForeignKey>
17    <EMail>biff@amail.com</EMail>
18    <First>biff</First>
19    <Last>bander</Last>
20    <PhoneExt>22</PhoneExt>
21    <OrderID xsi:nil="true"/>
22    <Title>IT services</Title>
23  </Person>
```

The `xsi:nil="true"` attribute indicates that, although the element exists, it has no content. Note that the `xsi:nil="true"` attribute applies to element values, and not to attribute values. An element with `xsi:nil="true"` may still have other attributes, even if it does not have content.

The `xsi:nil` attribute is not displayed explicitly in the MapForce graphical mapping, because it is handled automatically in most cases. Specifically, a "nilled" node (one that has the `xsi:nil="true"` attribute) exists, but its content does not exist.

The `nillable="true"` attribute is defined in the XML **schema**. In MapForce, it can be present in both the source and target components.



Nillable elements as mapping source

MapForce checks the `xsi:nil` attribute automatically, whenever a mapping reads data from nilled XML or XBRL elements. If the value of `xsi:nil` is `true`, the content will be treated as non-existent.

When you create a **Target-driven** mapping from a nillable source element to a nillable target element with **simple content** (a single value with optional attributes, but without child elements), where `xsi:nil` is set on a source element, MapForce adds the `xsi:nil` attribute to the target element (for example, `<OrderID xsi:nil="true"/>`).

When you create a **Copy-All** mapping from a nillable source element to a nillable target element, where `xsi:nil` is set on a source element, MapForce adds the `xsi:nil` attribute to the target element (for example, `<OrderID xsi:nil="true"/>`).

To check explicitly whether a source element has the `xsi:nil` attribute set to `true`, use the **`is-xsi-nil`** function. It returns TRUE for nilled elements and FALSE for other nodes.

To substitute a nilled (non-existing) source element value with something specific, use the **`substitute-missing`** function.

Notes:

- Connecting the **`exists`** function to a nilled source element returns TRUE, since the element node actually exists, even if it has no content.
- Using functions that expect simple values (such as **`multiply`** and **`concat`**) on elements where `xsi:nil` has been set does not yield a result, as no element content is present and no value can be extracted. These functions behave as if the source node did not exist.

Nillable elements as mapping target

When you create a **Target-driven** mapping from a nillable source element to a nillable target element with **simple content** (a single value with optional additional attributes, but without child elements), where `xsi:nil` is set on a source element, MapForce inserts the `xsi:nil` attribute into the target element (for example, `<OrderID xsi:nil="true"/>`). If the `xsi:nil="true"` attribute has not been set in the XML source element, then the element content is mapped to the target element in the usual fashion.

When mapping to a nillable target element with **complex type** (with child elements), the `xsi:nil` attribute will **not** be written automatically, because MapForce cannot know at the time of writing the element's attributes if any child elements will follow. For such cases, define a **Copy-All** connection to copy the `xsi:nil` attribute from the source element.

When mapping an **empty sequence** to a target element, the element will not be created at all, independent of its nillable designation.

To force the creation of an empty target element with `xsi:nil` set to `true`, connect the **`set-xsi-nil`** function directly to the target element. This works for target elements with simple and complex types.

If the node has simple type, use the **`substitute-missing-with-xsi-nil`** function to insert

`xsi:nil` in the target if no value from your mapping source is available. This can happen if the source node does not exist at all, or if a calculation (for example, multiply) involved a nilled source node and therefore yielded no result.

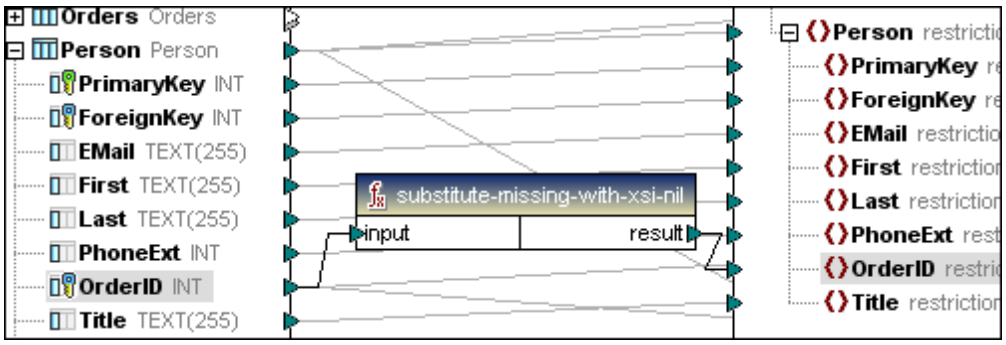
Note:

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

Mapping NULL database fields to xsi:nil

If you map a NULL database field to an nillable element of an XML schema, MapForce generates only those target elements which actually contain database data. Elements of NULL database fields are not created in the target component. Connecting the **`exists`** node function to such a source element results in `false` for the NULL fields.

To force the creation of all elements in the target component, use the **`substitute-missing-with-xsi-nil`** function from the node functions of the core library.



The screenshot above illustrates how the **`substitute-missing-with-xsi-nil`** function is used to create target elements for all database fields:

- All missing/NULL database fields contain <OrderID xsi:nil="true"/> in the target element.
- Existing data from database fields is mapped directly to the target element e.g. <OrderID>1</OrderID>.

To see the NULL fields of a database component, click the **Database Query** button and run a query on the database table(s). Null fields are shown as *[NULL]* in the Results window.

	EMail	First	Last	PhoneExt	OrderID	Title
1	v.callaby@nanonull.com	Vernon	Callaby	582	*[NULL]*	Office Manager
2	f.further@nanonull.com	Frank	Further	471	*[NULL]*	Accounts Recei
3	l.matise@nanonull.com	Loby	Matise	963	1	Accounting Mar
4	j.firstbread@nanonull.com	Joe	Firstbread	621	*[NULL]*	Marketing Manag

Mapping xsi:nil to NULL database fields

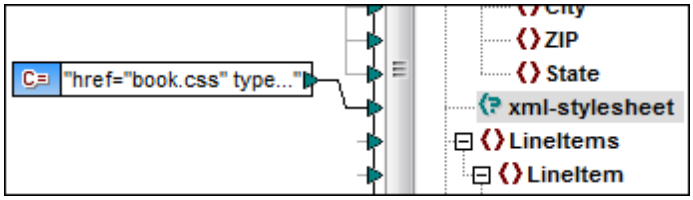
If you map a nilled XML element to a database column, MapForce writes a NULL value to the database. You can also use the `set-null` function if you want to set a database field to NULL unconditionally.

7.1.7 Comments and Processing Instructions

Comments and Processing Instructions can be inserted into target XML components. Processing instructions are used to pass information to applications that further process XML documents. Note that Comments and Processing instructions cannot be defined for nodes that are part of a copy-all mapped group.

To insert a Processing Instruction:

1. Right-click an element in the target component and select Comment/Processing Instruction, then one of the Processing Instruction options from the menu (Before, After)
2. Enter the Processing Instruction (target) name in the dialog and press OK to confirm, e.g. xml-stylesheet.
   This adds a node of this name to the component tree.

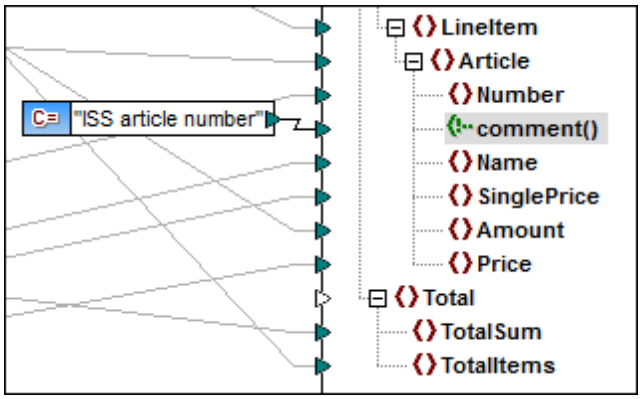


3. You can use, for example, a constant component to supply the value of the Processing Instruction attribute, e.g. `href="book.css" type="text/css".`

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

To insert a comment:

1. Right-click an element in the target component and select Comment/Processing Instruction, then one of the Comment options from the menu (Before, After).



This adds the comment node (`<!--comment()`) to the component tree.

2. Use a constant component to supply the comment text, or connect a source node to the comment node.

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

To delete a Comment/Processing Instruction:

- Right-click the respective node, select Comment/Processing Instruction, then select Delete Comment/Processing Instruction from the flyout menu.

7.1.8 CDATA Sections

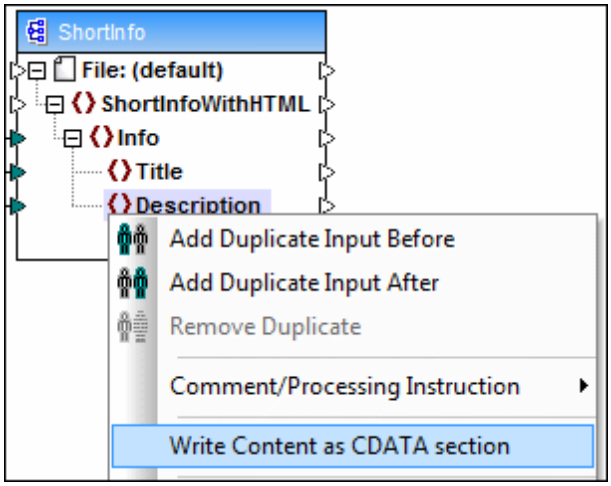
CDATA sections are used to escape blocks of text containing characters which would normally be interpreted as markup. CDATA sections start with "<![CDATA[" and end with the "]]>".

Target nodes can now write the input data that they receive as CDATA sections. The target node components can be:

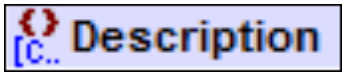
- XML data
- XML data embedded in database fields
- XML child elements of typed dimensions in an XBRL target

To create a CDATA section:

1. Right-click the target node that you want to define as the CDATA section and select "Write Content as CDATA section".



A prompt appears warning you that the input data should not contain the CDATA section close delimiter ']]>', click OK to close the prompt.
The [C.. icon shown below the element tag shows that this node is now defined as a CDATA section.

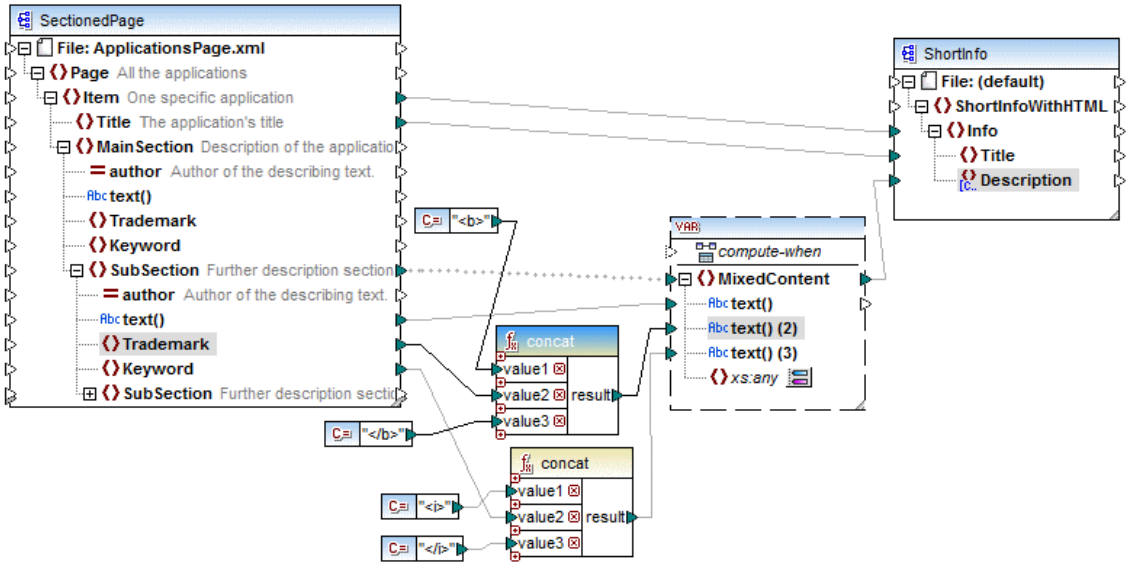


Note:
CDATA sections can also be defined on duplicate nodes, and xsi:type nodes.

Example:
The **HTMLinCDATA.mfd** mapping file available in the ...\MapForceExamples folder shows an example of where CDATA sections can be very useful.

In this example:

- Bold start () and end () tags are added to the content of the **Trademark** source element.
- Italic start () and end () tags are added to the content of the **Keyword** source element.
- The resulting data is passed on to duplicate **text()** nodes in the order that they appear in the source document, due to the fact the Subsection element connector, has been defined as a Source Driven (Mixed content) node.
- The output of the MixedContent node is then passed on to the **Description** node in the ShortInfo target component, which has been defined as a CDATA section.

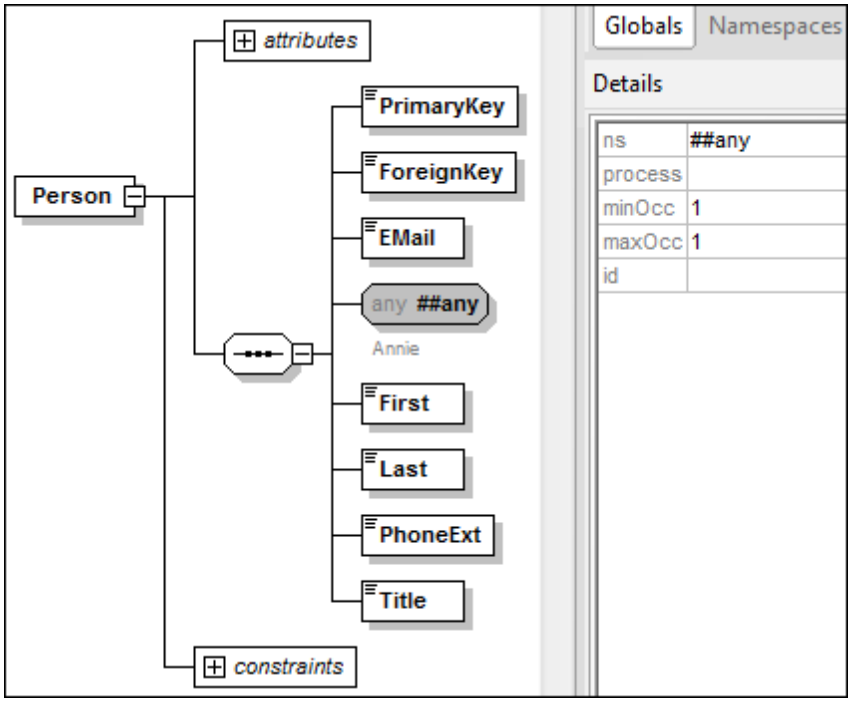


Clicking the Output button shows the CDATA section containing the marked-up text.

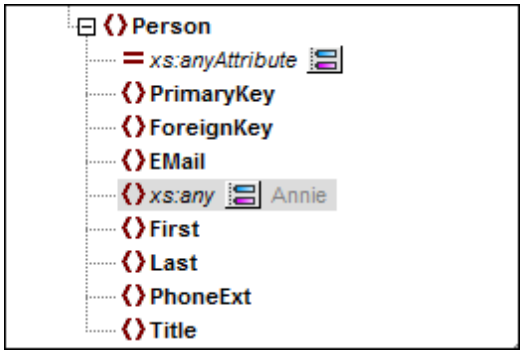
```
7    <Info>
8      <Title>MapForce</Title>
9      <Description><![CDATA[Altova <b>MapForce</b> 2014 Enterprise Edition is the premier <i>XML</i>
/ <i>database</i> / <i>flat file</i> / <i>EDI</i> data mapping tool that auto-generates mapping code in
<i>XSLT</i> 1.0/2.0, <i>XQuery</i>, <i>Java</i>,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

7.1.9 Wildcards - xs:any / xs:anyAttribute

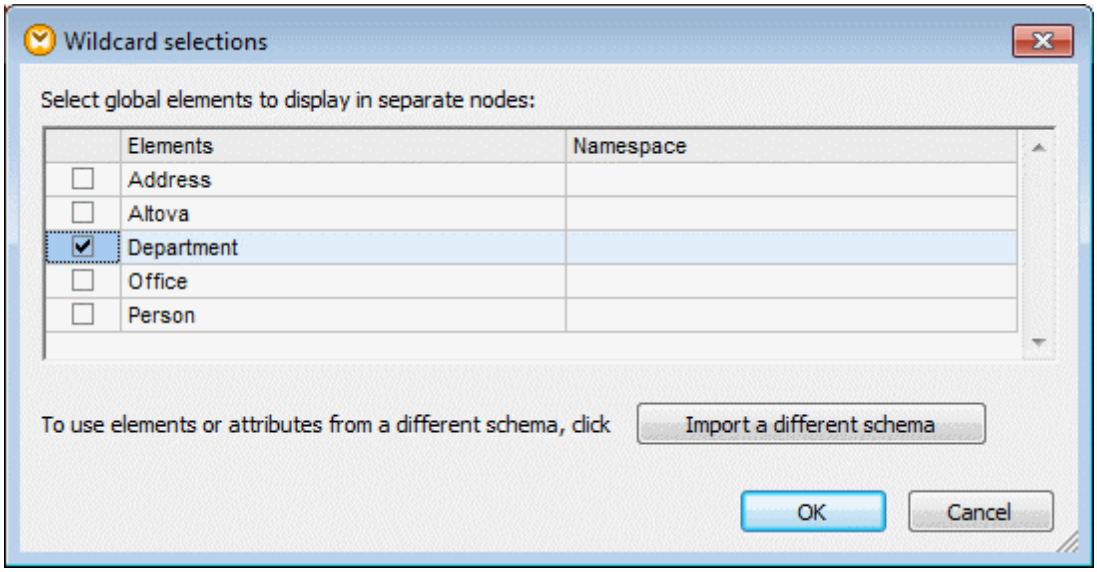
The wildcards xs:any (and xs:anyAttribute) allow you to use any elements/attributes from schemas . The screenshot shows the "any" element in the Schema view of XMLSpy.



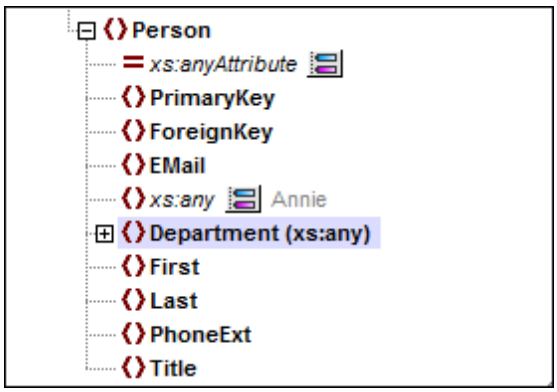
In MapForce the schema structure is shown as below with a "selection" button to the right of the xs:any element (and xs:anyAttribute).



Clicking the xs:any selection button opens the "Wildcard selections" dialog box. The entries in this listbox show the global elements/attributes declared in the current schema.



Clicking one, or more of the check boxes and confirming with OK, inserts that element/attribute (and any other child nodes) into the component at that position.



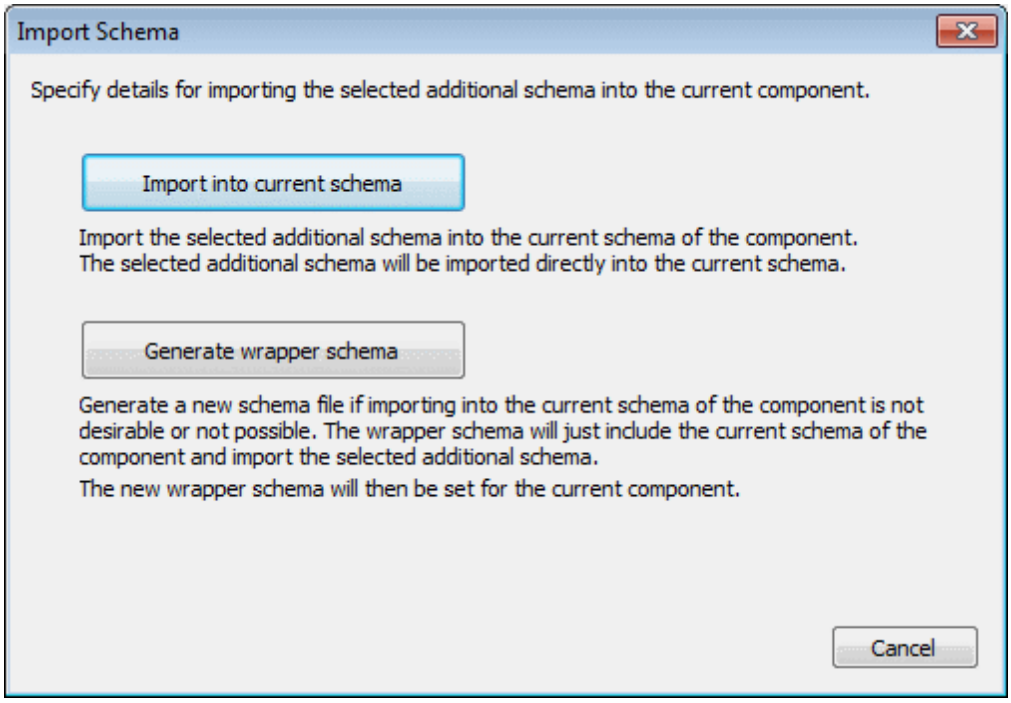
You can now map to/from these nodes as with any other element.

To remove a wildcard element:

- Click the selection button, then deselect the global elements.

To use elements from a different schema:

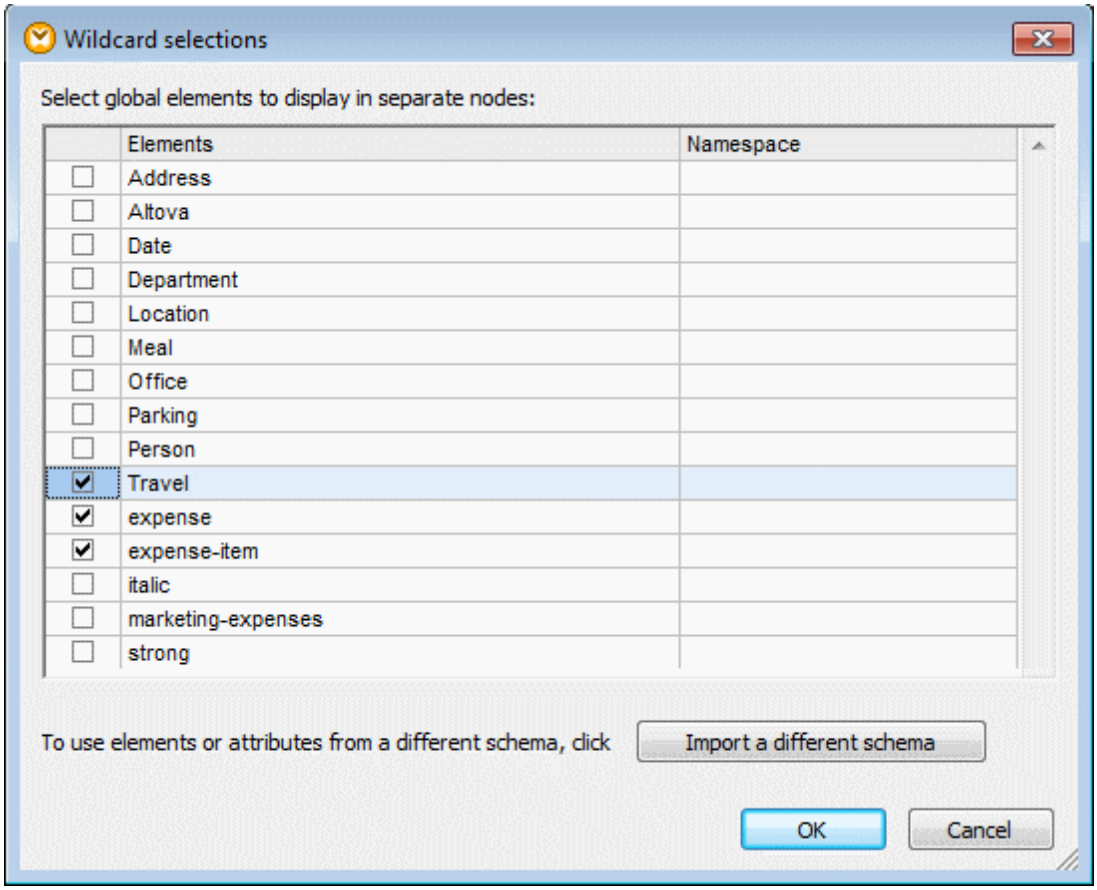
1. Click the "Import a different schema" button in the "Wildcard selections" dialog box.
2. Select the schema you want to import the elements from.
   You can now define if you want to import the other schema into the currently open one, or generate a new "wrapper" schema which contains references to both schemas.



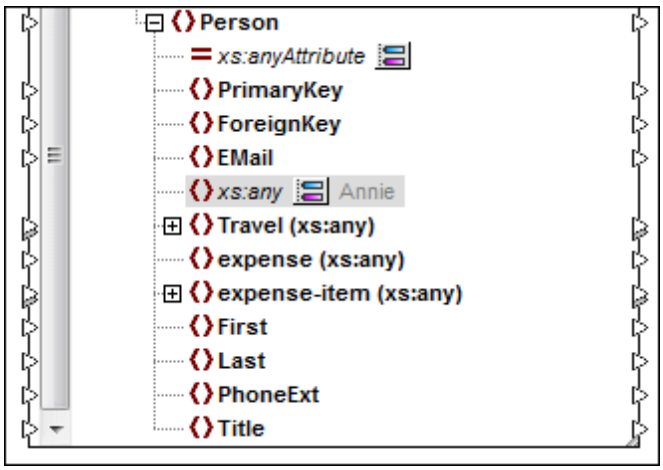
3. Click the button to select which of the two options you want.

Import into current schema

The screenshot shows the global elements available if the imported schema is the **HasExpenses.xsd** file available in the ...\MapForceExamples folder. Three expense related elements have been selected.



Having clicked OK, each of the elements (and any child elements they may have) are added to the component below the xs:any wildcard node.



Each of the imported nodes has a (xs:any) annotation to show that it has been imported.

Note:

Importing into the current schema, **overwrites** the schema at that location. You must therefore have the user rights to do so. A remote schema that you might have opened using the "Switch to URL" button cannot be overwritten, and cannot be imported in this way.

Generate wrapper schema

1. Click the "Generate wrapper schema" button in the "Wildcard selections" dialog box, enter the name of the new wrapper schema and click Save.
   A default name is supplied in the form XXXX-wrapper.xsd. This new wrapper schema will include the current schema and import the other one.

   A prompt appears asking if you want the Component Setting schema location to reference the wrapper schema, or if you want to change it to reference the previous main schema.
2. Click No to keep the reference to the wrapper schema, or click Yes to change the schema location to the previous main schema.
   The imported nodes are shown as before in the component.

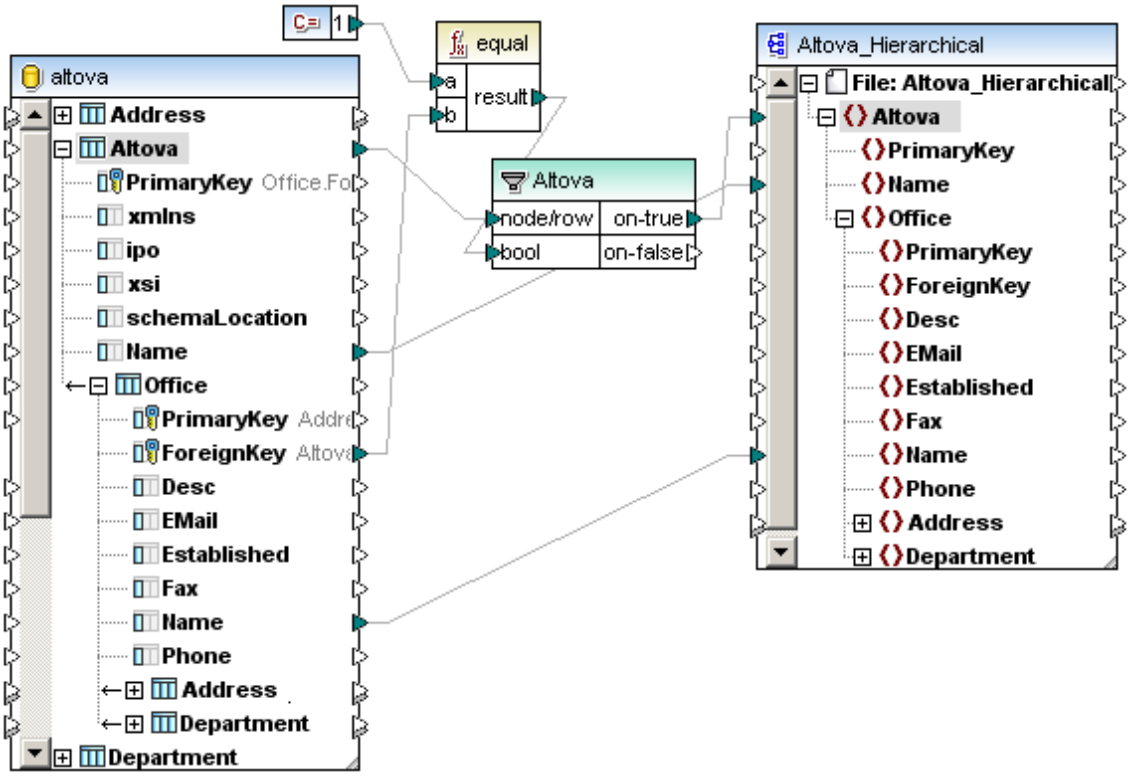
Note:
Using the generate wrapper option allows you to use remote schemas, with a URL, and add elements from another schema to the current component.

7.1.10 Mapping to the Root Element

An XML document may have only one root element. Therefore, when drawing a connection to the root element of the target Schema/XML file, make sure that the connection passes only one item to the target XML component. Otherwise, you will create well-formed, but not valid, XML files.

Use a filter component (see Filtering) to limit the mapped data to a single element or record.

In the example below, the `ForeignKey` is checked to see if it is 1, and only then is one Altova element passed on to the target root element. If no mappings exist from any of the source items to the target root element, then the root element of the target schema is inserted automatically.

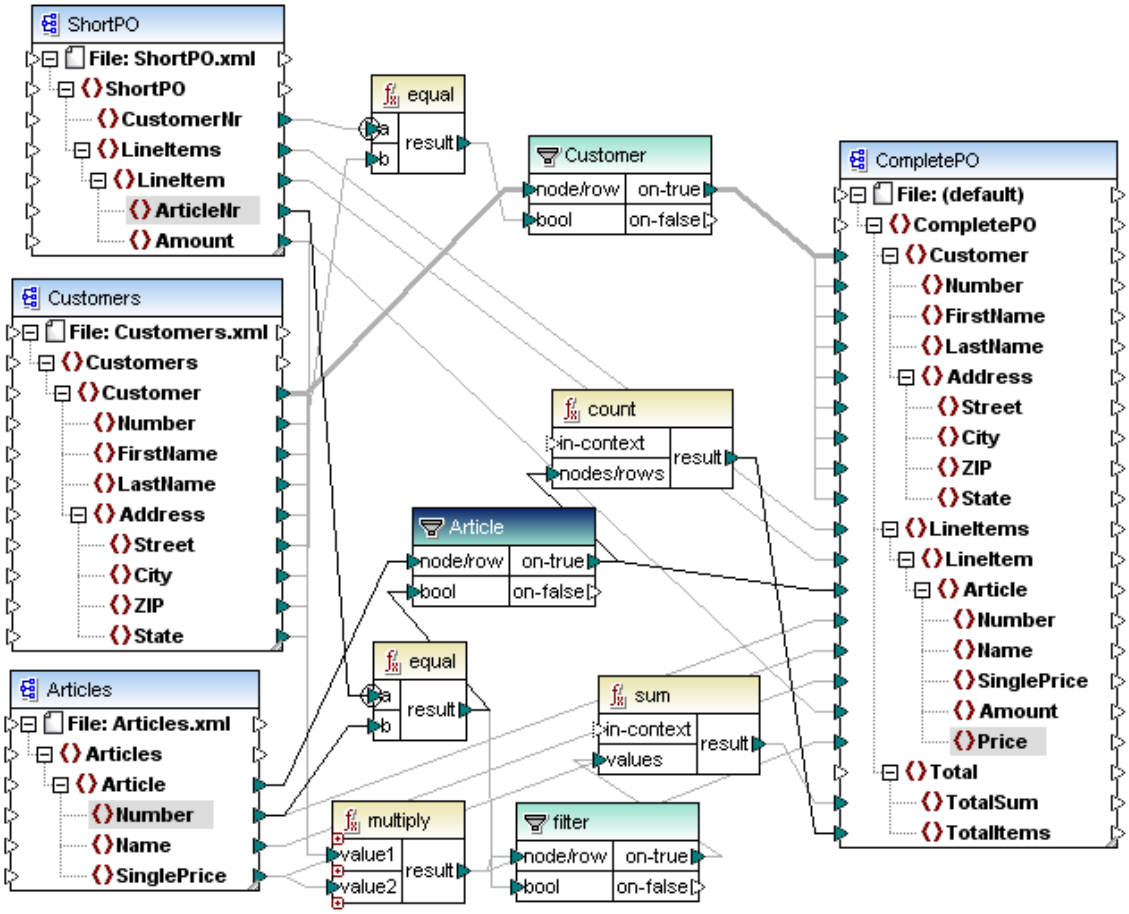


7.1.11 Merging Data from Multiple Schemas

MapForce allows you to merge multiple files into a single target file.

This example merges multiple source components with different schemas to a target schema. To merge an arbitrary number of files using the same schema, see Processing Multiple Input or Output Files Dynamically.

The **CompletePO.mfd** file available in the **...\MapForceExamples** folder shows how three XML files are merged into one purchasing order XML file.



Note that multiple source component data are combined into one target XML file - CompletePO

- **ShortPO** is a schema with an associated XML instance file and contains only customer number and article data, i.e. Line item, number and amount.
- **Customers** is a schema with an associated XML instance file and contains customer number and customer information details, i.e. Name and Address info. (There is only one customer in this file with the Customer number of 3)
- **Articles** is a schema with an associated XML instance and contains article data, i.e. article name number and price.
- **CompletePO** is a schema file without an instance file as all the data is supplied by the three XML instance files. The hierarchical structure of this file makes it possible to merge and output all XML data.

This schema file has to be created in an XML editor such as XMLSpy, it is not generated by MapForce (although it would be possible to create if you had an CompletePO.xml instance file).

The stucture of CompletePO is a combination of the source XML file structures.

The **filter** component (Customer) is used to find/filter the data where the customer numbers are identical in both the ShortPO and Customers XML files, and pass on the associated data to the target CompletePO component.

- The **CustomerNr** in ShortPO is compared with the **Number** in Customers using the "equal" function.
- As ShortPO only contains one customer (number 3), only customer and article data for

customer number 3, can be passed on to the filter component.
- The **node/row** parameter, of the filter component, passes on the **Customer** data to "on-true" when the bool parameter is true, i.e. when the same number has been found, in this case customer number 3.
- The rest of the customer and article data are passed on to the target schema through the two other filter components.

7.1.12 Digital Signatures

Digital signatures are a W3C specification to digitally sign an XML document with an encrypted code that can be used to verify that the XML document has not been altered. The XML Signature feature in MapForce supports only certificates of type RSA-SHA1 and DSA-SHA1.

For more details about XML signatures, see the W3C specification for XML signatures at http://www.w3.org/TR/xmldsig-core/

MapForce supports creating XML digital signatures for XML and XBRL output files. Digital signatures can only be generated when the output target is BUILTIN and only in the preview. A signature is created for the generated result file, when the output button is pressed, and the result file is saved.

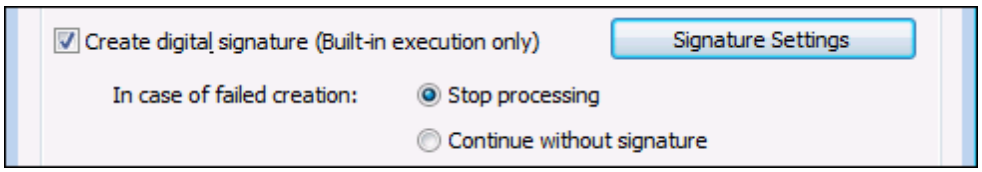
Note: MapForce Server does not support digital signatures.

Digital signatures can be embedded as the last element of the output document or stored in a separate signature file.

- If "Enveloped" is selected, then the signature is the last child element below the root element of the XML file.
- If "Detached" is selected, then the signature file is generated as a separate document.

To activate generation of digital signatures:

1. Open the Component Settings dialog box of the output component, by double-clicking its header, or by selecting **Component | Properties**.
2. Select the **Create digital signature** check box.



3. The XML Signature Settings dialog box opens, where you can define the required settings (see XML Signature Settings).

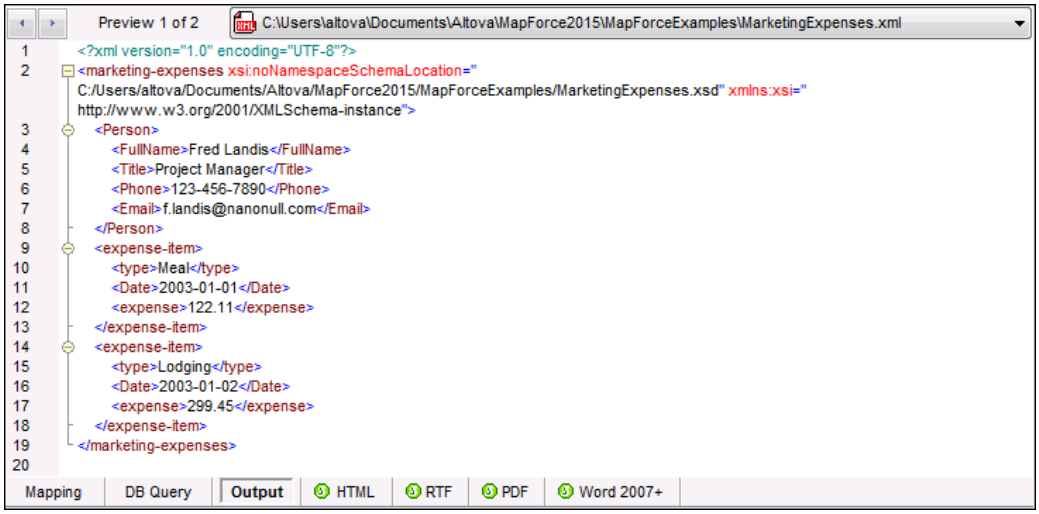
To change settings for digital signatures:

1. Open the Component Settings dialog box of the output component, by double-clicking its header, or by selecting **Component | Properties**.
2. Click the "Signature Settings" button to open the XML Signature Settings dialog box.
3. Enter settings and click OK.

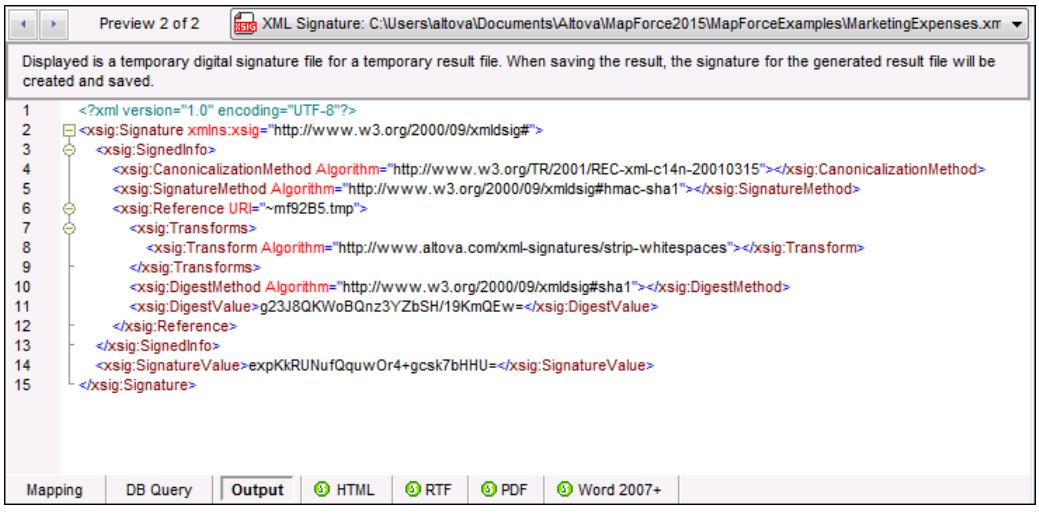
Using the **MarketingExpenses_DetachedSignature.mfd** file in the ...\MapForceExamples folder, as an example:

1. Double click the MarketingExpenses target component, then click the "Signature Settings" button. The selected options are shown.
2. Click OK to close the dialog box.
3. Click the Output button to see the mapping result.

   Two files are generated in the preview window. The first file, **MarketingExpenses.xml**, is the mapping result of that target component.

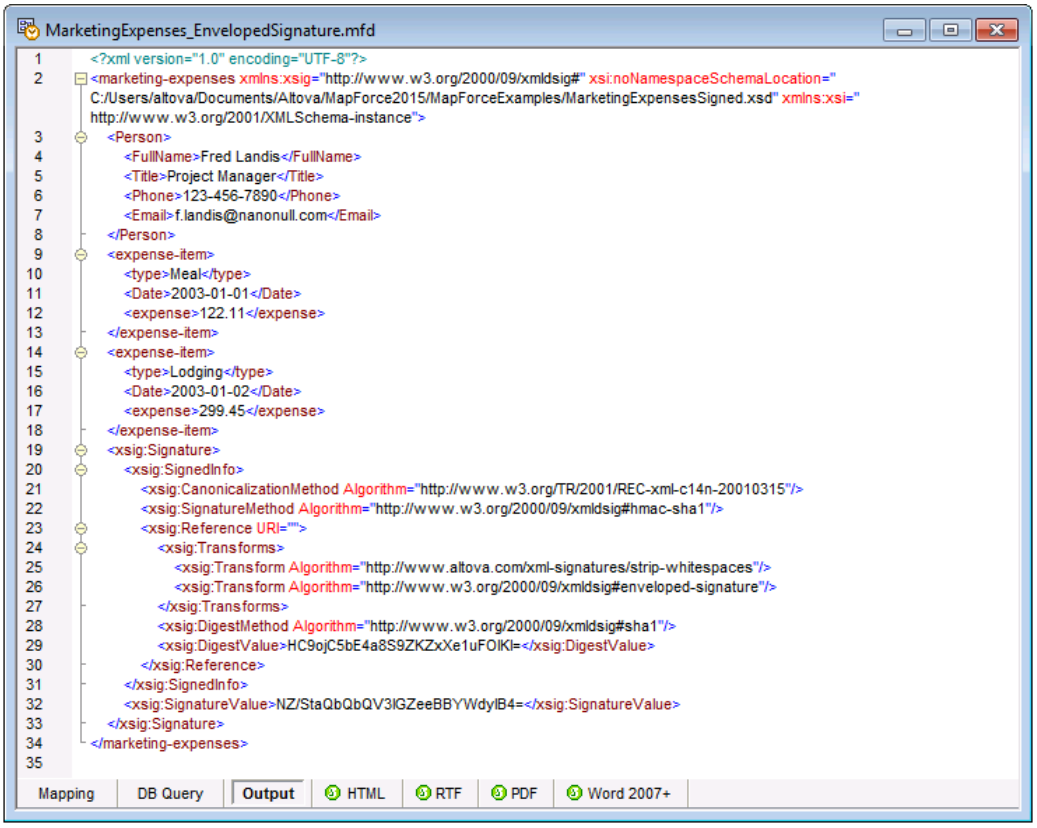


   The second file, **MarketingExpenses.xml.xsig**, is the temporary digital signature file generated by the target component.



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

The **MarketingExpenses_EnvelopedSignature.mfd** file in the ...\MapForceExamples folder shows the result when the signature placement is "Enveloped".



XML document validity

If an XML signature is embedded in the XML document, a `Signature` element in the namespace `http://www.w3.org/2000/09/xmldsig#` is added to the XML document. In order for the document to remain valid according to a schema, the schema must contain the appropriate element declarations. MapForce embeds signatures using the Enveloped option:

- *Enveloped:* The `Signature` element is created as the last child element of the root (or document) element.

  If you do not wish to modify the schema of the XML document, the XML signature can be created in an external file using the "Detached" option.

Given below are excerpts from XML Schemas that show how the `Signature` element of an enveloped signature can be allowed. You can use these examples as guides to modify your own schemas.

In the first of the two listings below, the XML Signature Schema is imported into the user's schema. The XML Signature Schema is located at the web address: `http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

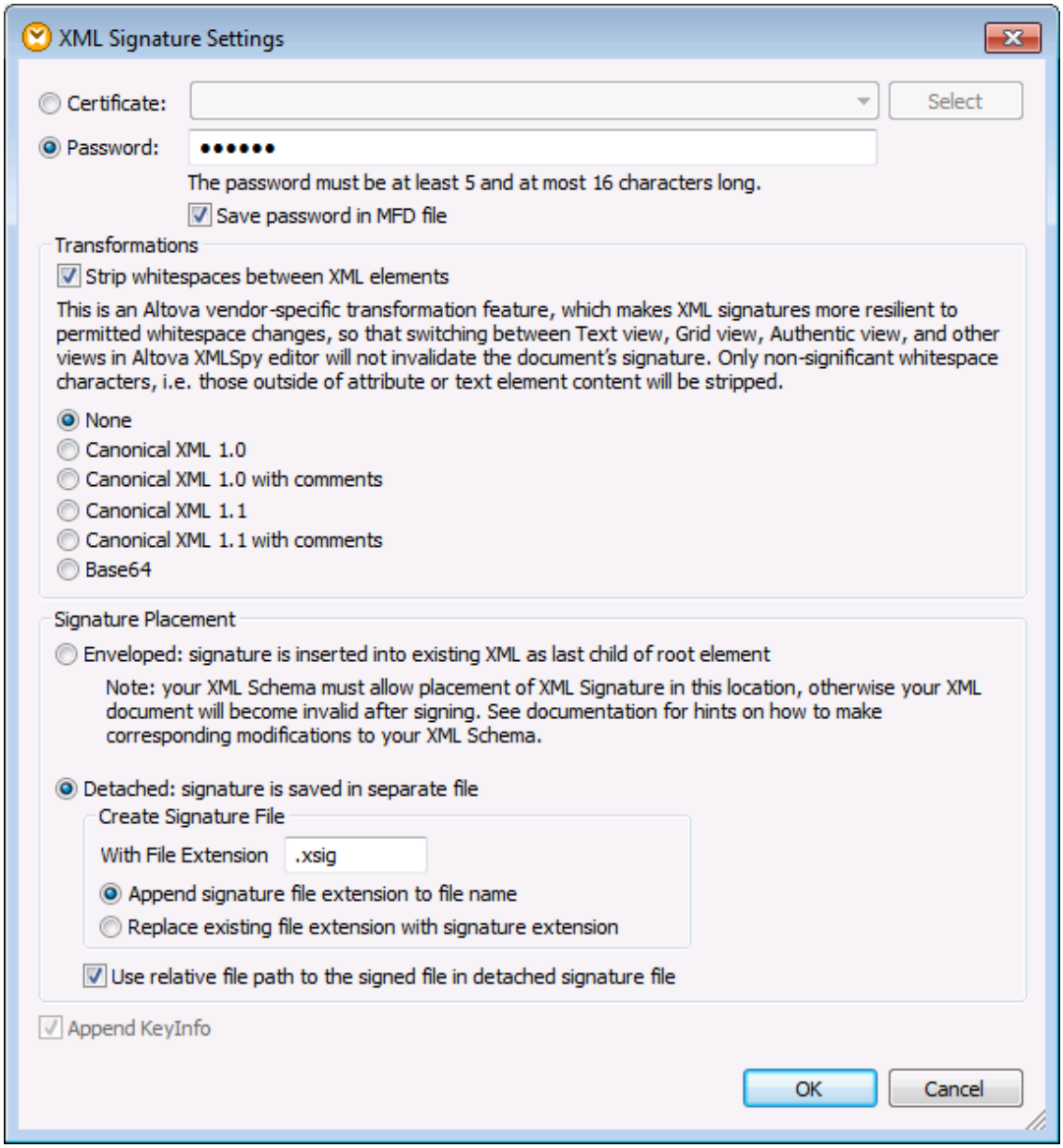
```
            xmlns:xsig="http://www.w3.org/2000/09/xmldsig#"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">
   <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
              schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-
schema.xsd"/>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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
   <xs:element name="Root">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="selection"/>
            <xs:element ref="newsitems" minOccurs="0"/>
            <xs:element ref="team" minOccurs="0"/>
            <xs:any namespace="##other" minOccurs="0" processContents="lax"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   ...
</xs:schema>
```

7.1.12.1 XML Signature Settings

Signature settings are stored for each component individually in the component settings dialog box, and are all stored in the MFD file when it is saved.



XML Signature Settings dialog box

Authentication method: Certificate or Password

The signature can be based on a certificate or a password. Select the radio button of the method you wish to use.

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

certificate.

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

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

Transformations

The XML data is transformed and the result of the transformation is used for the creation of the signature. You can specify the canonicalization algorithm to be applied to the file's XML data (the `SignedInfo` content) prior to performing signature calculations. Significant points of difference between the algorithms are noted below:

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

  Note:
  "...with comments" is only available for "Detached" placement.

- *Base64:*
  The root (or document) element of the XML document is considered to be Base64 encoded, and is read in its binary form. If the root element is not Base64, an error is returned or the element is read as empty, depending on what type of element is encountered.

- *None:*
  No transformation is carried out and the XML data from the binary file saved on disk, is passed directly for signature creation. Any subsequent change in the data will result in a failed verification of the signature.

  However, if the **Strip Whitespace between XML elements** check box option is selected, then all whitespace is stripped and changes in whitespace will be ignored.

  A major difference between the *None* option and a *Canonicalization* option is that canonicalization produces an XML data stream, in which some differences, such as attribute order, are normalized. As a result, a canonicalization transformation will normalize any changes such as that of attribute order (so verification will succeed), while no-transformation will reflect such a change (verification will fail).

  Note, however, that a default canonicalization is performed if the signature is embedded (enveloped). So the XML data will be used as is (i.e. with no transformation) when: the signature is detached, *None* is selected, and the *Strip Whitespaces* checkbox is unchecked.

Signature Placement

The signature can be placed within the XML file or be created as a separate file. The following options are available:

- *Enveloped:*
  The signature element is created as the last child element of the root (document) element. Note: the associated XML Schema must contain the signature definition elements for the output XML to be valid. Please see the top of this section for more information.
- *Detached:*
  The XML signature is created as a separate file. In this case, you can specify the file extension of the signature file and whether the file name is created with: (i) the extension appended to the name of the XML file (for example, `test.xml.xsig`), or (ii) the extension replaces the XML extension of the XML file (for example, `test.xsig`). You can also specify whether, in the signature file, the reference to the XML file is a relative or an absolute path.

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

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

Append KeyInfo

The *Append Keyinfo* option is available when the signature is certificate-based. It is unavailable if the signature is password-based.

If Append KeyInfo is active/checked, public-key information is placed inside the signature, otherwise key information is not included in the signature. The advantage of including key information is that the certificate itself (specifically the public-key information in it) will not be required for the verification process (since the key information is present in the signature).

Invalid signature settings

MapForce cannot digitally sign an output if the signature settings are invalid. Signature settings are invalid if:

- The selected certificate is not accessible, or is not suitable for signing xml documents, or
- No password is set, e. g. the option "Save password in mfd file" is not checked.

When clicking the Output button MapForce prompts -
for the password with:

> Please specify a password to sign the output of component "MarketingExpensesSigned"

for the certificate with:

> Please choose the store containing the certificate you want to use to sign the output of component "MarketingExpensesSigned".

If no password or certificate is chosen, then the processing is either stopped, or continued without a signature. You can determine this behavior in the Component Settings dialog box via the "Stop processing" or "Continue without signature" radio buttons.

If the mapping is executed from the command line, no prompt dialog box appears. The mapping execution either stops with an error, or continues without signature.

7.2 Databases and MapForce

Altova web site: Mapping Database data

MapForce 2016 provides powerful support for mapping databases to XML, flat files, and other database formats. With MapForce Enterprise edition, you can additionally map databases to EDI formats, Excel 2007+, JSON, XBRL, and Web services.

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

Database	Root Object	Notes
Firebird 2.5.4	database	
IBM DB2 8.x, 9.1, 9.5, 9.7, 10.1, 10.5	schema	
IBM DB2 for i 6.1, 7.1	schema	Logical files are supported and shown as views.
IBM Informix 11.70	database	Informix supports connections via ADO, JDBC and ODBC. The implementation does not support large object data types in any of the code generation languages. MapForce will generate an error message (during code generation) if any of these data types are used.
Microsoft Access 2003, 2007, 2010, 2013	database	
Microsoft SQL Server 2005, 2008, 2012, 2014	database	
MySQL 5.0, 5.1, 5.5, 5.6	database	
Oracle 9i, 10g, 11g, 12c	schema	
PostgreSQL 8.0, 8.1, 8.2, 8.3, 9.0.10, 9.1.6, 9.2.1, 9.4	database	
SQLite 3.x	database	SQLite connections are supported as native, direct connections to the SQLite database file. No separate drivers are required.
Sybase ASE15	database	

Databases are supported as mapping components in all MapForce transformation languages, with

the exception of SQLite. SQLite components are supported only in the Built-in transformation language.

If your mappings include database connections, and then you run the mapping execution file (.mfx) on a server machine with MapForce Server, you will need to create database connections on the target server as well. If the target operating system is Linux or Mac OS, connections through ADO and ODBC are not supported (see also Database Connections on Linux and Mac).

To save time and reduce complexity, you may want to define database connections as global resources (see Global Resources).

7.2.1 Connecting to a Database

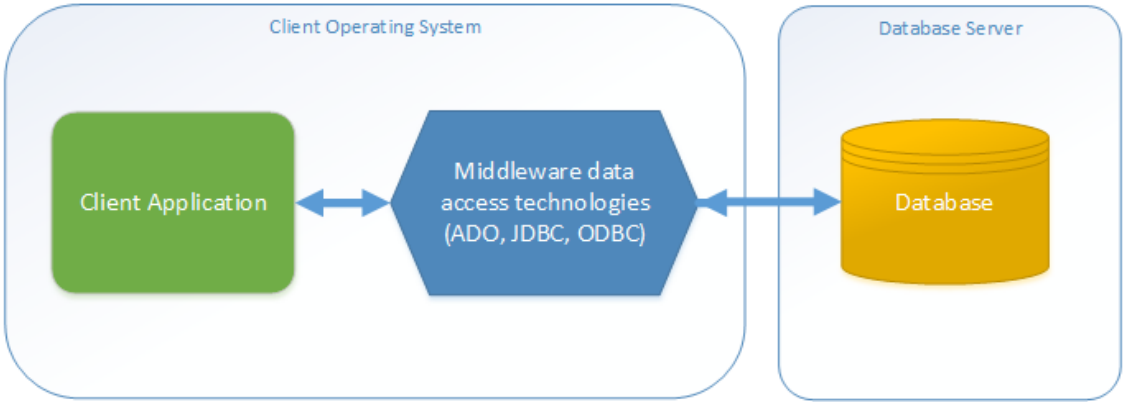
A database typically resides on a database server (either local or remote) which does not necessarily use the same operating system as the application that connects to it and consumes data. For example, while MapForce runs on a Windows operating system, the database from which you want to access data (for example, MySQL) might run on a Linux machine.

MapForce uses a database connection mechanism which 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. More specifically, 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
- JDBC (Java Database Connectivity)
- ODBC (Open Database Connectivity)

Direct native connections to SQLite databases are also supported. To connect to a SQLite database, no additional drivers are required to be installed on your system.

The following diagram illustrates a simplified, generic representation of the typical data exchange between a client application such as MapForce and a database.



Typical data exchange between a client application and a database server

Whether you should use ADO, ODBC or JDBC as a data connection interface largely depends on your existing software infrastructure. You will typically choose the data access technology and the database driver which integrates tighter (preferably natively) 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 or ODBC drivers 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 OLE DB, 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.

This section contains the following topics:

7.2.1.1 Starting the Database Connection Wizard

Whenever you take an action that requires a database connection, a wizard appears that guides you through the steps required to set up the connection.

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

To add the database as a source or target component on a mapping:

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

To add the database as a reusable global resource:

1. On the **Tools** menu, click **Global Resources**.
2. Click **Add**, and then click Database.
3. Click **Choose Database**.

After you select a database type and click **Next**, the on-screen instructions will depend on the database kind, technology (ADO, ODBC, JDBC) and driver used.

For examples applicable to each database type, see the Examples section. For instructions applicable to each database access technology, refer to the following topics:

- Setting up an ADO Connection
- Setting up an ODBC Connection
- Setting up a JDBC Connection

7.2.1.2 Database Drivers Overview

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

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

With some exceptions, most database vendors provide database client software which normally includes any required database drivers, or provide 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. If you have not installed your database client software yet, it is strongly recommended to read carefully the installation and configuration instructions of the database client, since they typically vary for each database version and for each Windows version.

Database	ODBC	JDBC	ADO
Firebird	Firebird ODBC driver (http://www.firebirdsql.org/en/odbc-driver/)	Firebird JDBC driver (http://www.firebirdsql.org/en/jdbc-driver/)	
IBM DB2	IBM DB2 ODBC Driver	IBM Data Server Driver for JDBC and SQLJ	IBM OLE DB Provider for DB2
IBM DB2 for i	iSeries Access ODBC Driver	IBM Toolbox for Java JDBC Driver	• 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
IBM Informix	IBM Informix ODBC Driver	IBM Informix JDBC Driver	IBM Informix OLE DB Provider
Microsoft Access	• **Microsoft Access Driver**		• **Microsoft Jet OLE DB Provider** • **Microsoft Access Database Engine OLE DB Provider**
Microsoft SQL Server	• **SQL Server Native Client**	• Microsoft JDBC Driver for SQL Server (http://msdn.microsoft.com/en-us/data/aa937724.aspx)	• **Microsoft OLE DB Provider for SQL Server** • **SQL Server Native Client**
MySQL	Connector/ODBC (http://dev.mysql.com/downloads/connector/odbc/)	Connector/J (http://dev.mysql.com/downloads/connector/j/)	
Oracle	• **Microsoft ODBC for Oracle** • Oracle ODBC Driver (typically installed	• JDBC Thin Driver • JDBC Oracle Call Interface (OCI) Driver These drivers are	• **Microsoft OLE DB Provider for Oracle**

Database	ODBC	JDBC	ADO
	during the installation of your Oracle database client)	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.	
PostgreSQL	psqlODBC (https://odbc.postgresql.org/)	Postgre JDBC Driver (https://jdbc.postgresql.org/download.html)	
Sybase	Sybase ASE ODBC Driver	jConnect™ for JDBC	Sybase ASE OLE DB Provider

** The drivers highlighted in bold are Microsoft-supplied. If not already available on Windows system, they can be downloaded from the official Microsoft website.*

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, consider the following general notes and recommendations:

- Since 32-bit and 64-bit drivers may not be compatible, make sure to install and configure the driver version applicable to your Altova application. For example, if you are using a 32-bit Altova application on a 64-bit operating system, set up your database connection using the 32-bit driver version.
- The latest driver versions may provide features not available in older editions.
- When setting up an ODBC data source, it is generally recommended to create the data source name (DSN) as *System DSN* instead of *User DSN*.
- When setting up a JDBC data source, ensure that JRE (Java Runtime Environment) is installed and that the CLASSPATH environment variable of the operating system is configured.
- For the support details and known issues applicable to Microsoft-supplied database drivers, refer to the MSDN documentation.
- 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. Whether you are using an official or third party database driver, the most comprehensive information and the configuration procedures applicable to that specific driver on your specific operating system is normally part of the driver installation package.

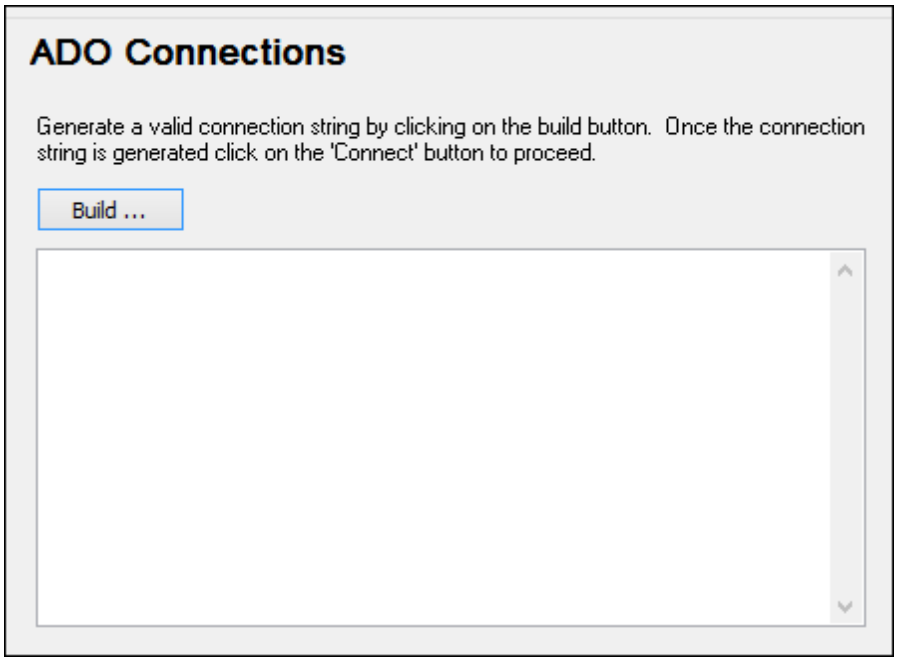
7.2.1.3 Setting up an ADO Connection

Microsoft ActiveX Data Objects (ADO) is a data access technology that enables you to connect to a variety of data sources through OLE DB. OLE DB is an alternative interface to ODBC or JDBC; it provides uniform access to data in a COM (Component Object Model) environment. ADO is the typical choice for connecting to Microsoft native databases such as Microsoft Access or

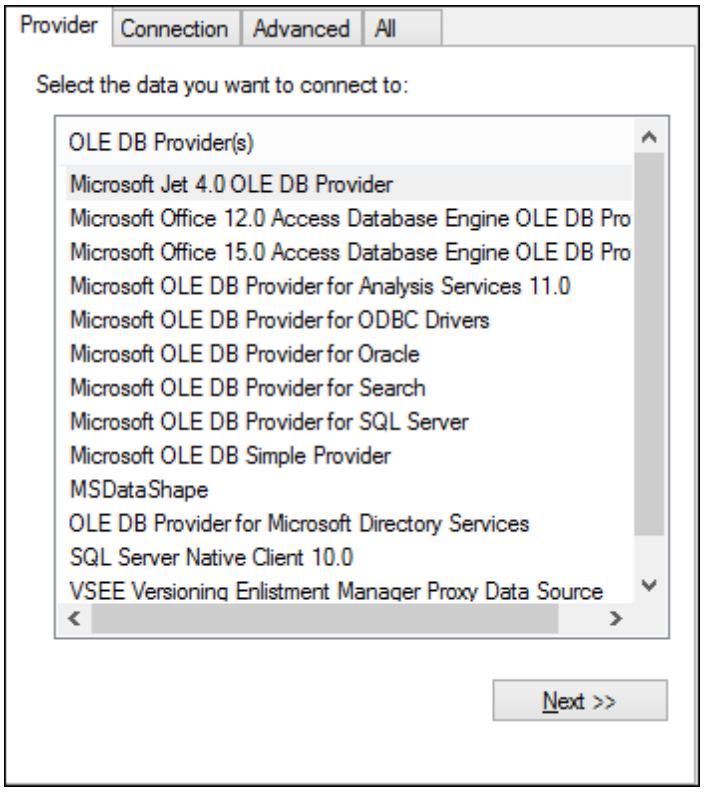
SQL Server, although you can also use it for other data sources.

To set up an ADO connection:

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



3. Click **Build**.



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

To connect to this database...	Use this provider...
Microsoft Access	• **Microsoft Office Access Database Engine OLE DB Provider** When connecting to Microsoft Access 2003, you can also use the **Microsoft Jet OLE DB Provider**.
SQL Server	• **SQL Server Native Client** • **Microsoft OLE DB Provider for SQL Server**
Other database	Select the provider applicable to your database. If an OLE DB provider to your database is not available, install the required driver from the database vendor (see Database Drivers Overview). Alternatively, set up an ODBC or JDBC connection. If the operating system has an ODBC driver to the required database, you can also use the **Microsoft OLE DB Provider for ODBC Drivers**.

5. Click **Next** and complete the wizard.

The subsequent wizard steps are specific to the provider you chose. For SQL Server, you will need to provide or select the host name of the database server, as well as the database username and password. For Microsoft Access, you will be asked to browse for or provide the path to the database file.

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

- Setting up the SQL Server Data Link Properties
- Setting up the Microsoft Access Data Link Properties

Connecting to an Existing Microsoft Access Database

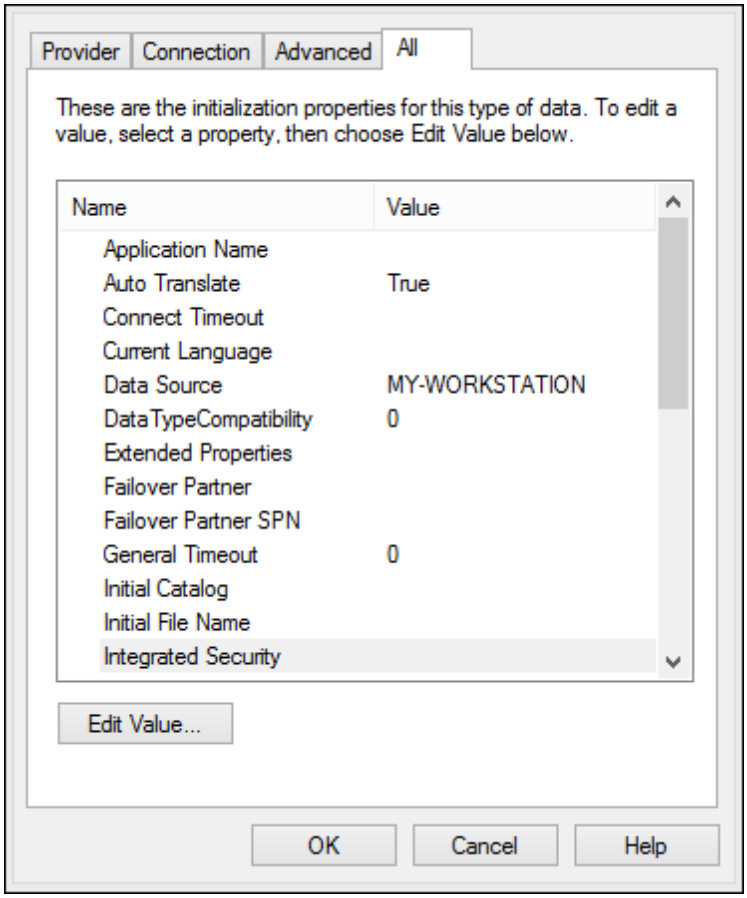
This approach is suitable when you want to connect to a Microsoft Access database which is not password-protected. If the database is password-protected, set up the database password as shown in Connecting to Microsoft Access (ADO).

To connect to an existing Microsoft Access database:

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

Setting up the SQL Server Data Link Properties

When you connect to a Microsoft SQL Server database through ADO (see Setting up an ADO Connection), ensure that the following data link properties are configured correctly in the **All** tab of the Data Link Properties dialog box.

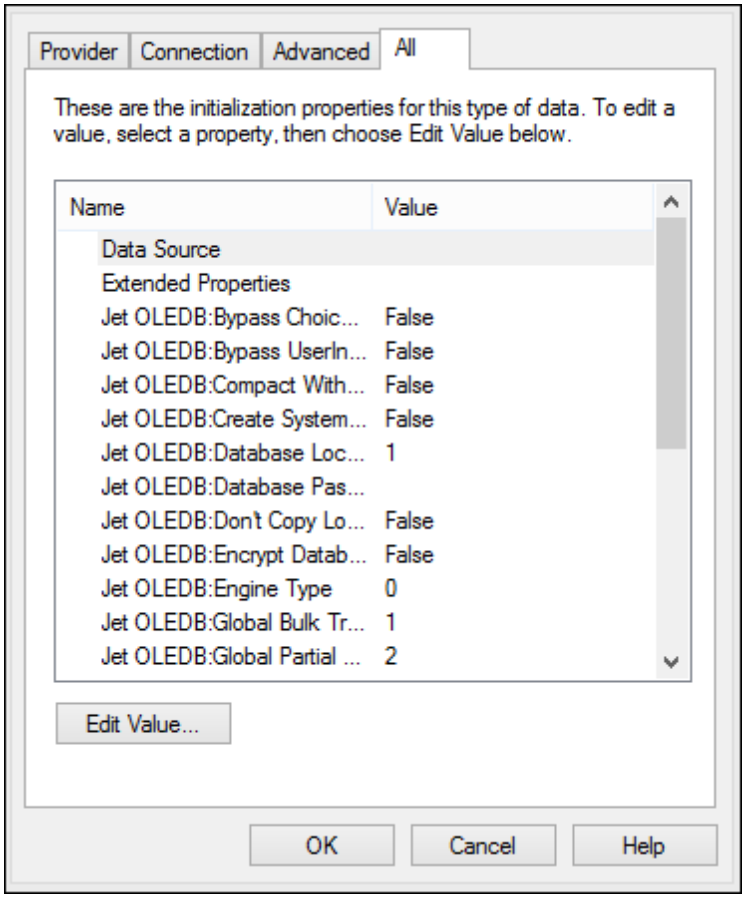


Data Link Properties dialog box

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

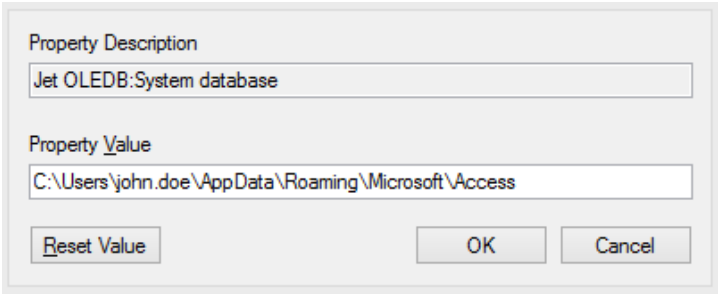
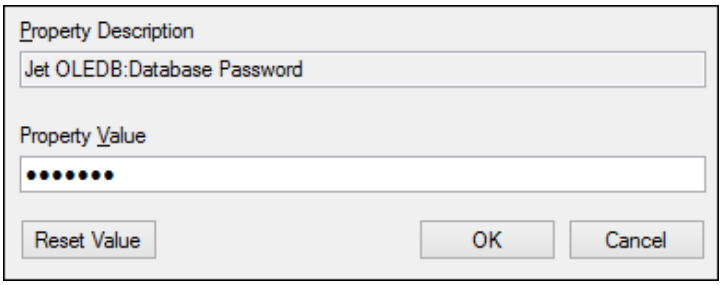
Setting up the Microsoft Access Data Link Properties

When you connect to a Microsoft Access database through ADO (see Setting up an ADO Connection), ensure that the following properties are configured correctly in the **All** tab of the Data Link Properties dialog box.



Data Link Properties dialog box

Property	Notes
Data Source	This property stores the path to the Microsoft Access database file. To avoid database connectivity issues, it is recommended to use the UNC (Universal Naming Convention) path format, for example: `\\anyserver\share$\filepath`
Jet OLEDB:System Database	This property stores the path to the workgroup information file. You may need to explicitly set the value of this property before you can connect to a Microsoft Access database. If you cannot connect due to a "workgroup information file" error, locate the workgroup information file (**System.MDW**) applicable to your user profile (see http://support.microsoft.com/kb/305542 for instructions), and set the property value to the path of the **System.MDW** file.

Property	Notes
	
Jet OLEDB:Database Password	If the database is password-protected, set the value of this property to the database password. 

7.2.1.4 Setting up an ODBC Connection

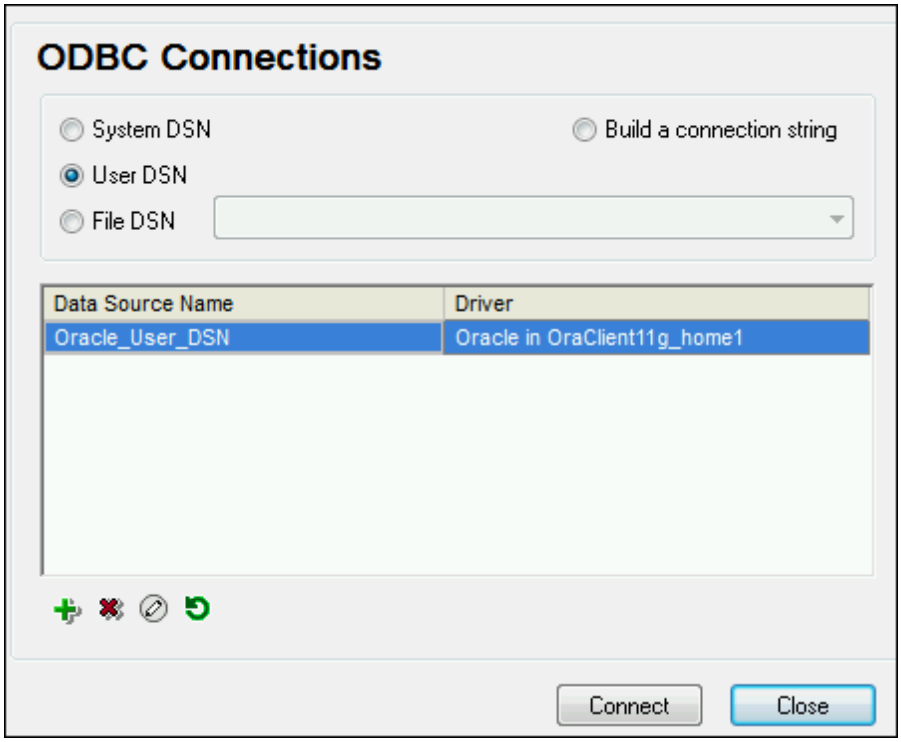
ODBC (Open Database Connectivity) is a widely used data access technology that enables you to connect to a database from MapForce. It can be used either as primary means to connect to a database, or as an alternative to OLE DB- or JDBC-driven connections.

To connect to a database through ODBC, first you need to create an ODBC data source name (DSN) on the operating system. This step is not required if the DSN has already been created, perhaps by another user of the operating system. The DSN represents a uniform way to describe the database connection to any ODBC-aware client application on the operating system, including MapForce. DSNs can be of the following types:

- System DSN
- User DSN
- File DSN

A *System* data source is accessible by all users with privileges on the operating system. A *User* data source is available to the user who created it. Finally, if you create a *File DSN*, the data source will be created as a file with the .dsn extension which you can share with other users, provided that they have installed the drivers used by the data source.

Any DSNs already available on your machine are listed by the database connection dialog box when you click **ODBC connections** on the ODBC connections dialog box.



ODBC Connections dialog box

If a DSN to the required database is not available, the MapForce database connection wizard will assist you to create it; however, you can also create it directly on your Windows operating system. In either case, before you proceed, ensure that the ODBC driver applicable for your database is in the list of ODBC drivers available to the operating system (see Viewing the Available ODBC Drivers).

To connect by using a new DSN:

1. Start the database connection wizard.
2. On the database connection dialog box, click **ODBC Connections**.
3. Select a data source type (User DSN, System DSN, File DSN).

   To create a System DSN, you need administrative rights on the operating system.

4. Click **Add** .
5. Select a driver, and then click **User DSN** or **System DSN** (depending on the type of the DSN you want to create). If the driver applicable to your database is not listed, download it from the database vendor and install it (see Database Drivers Overview).
6. On the dialog box that pops up, fill in any driver specific connection information to complete the setup.

For the connection to be successful, you will need to provide the host name (or IP address) of the database server, as well as the database username and password. There may be other optional connection parameters—these parameters vary between database providers. For detailed

information about the parameters specific to each connection method, consult the documentation of the driver provider. Once created, the DSN becomes available in the list of data source names. This enables you to reuse the database connection details any time you want to connect to the database. Note that User DSNs are added to the list of User DSNs whereas System DSNs are added to the list of System DSNs.

To connect by using an existing DSN:

1. Start the database connection wizard.
2. Click **ODBC Connections**.
3. Choose the type of the existing data source (User DSN, System DSN, File DSN).
4. Click the existing DSN record, and then click **Connect**.

To build a connection string based on an existing .dsn file:

1. Start the database connection wizard.
2. Click **ODBC Connections**.
3. Select **Build a connection string**, and then click **Build**.
4. If you want to build the connection string using a File DSN, click the **File Data Source** tab. Otherwise, click the **Machine Data Source** tab. (System DSNs and User DSNs are known as "Machine" data sources.)
5. Select the required .dsn file, and then click **OK**.

To connect by using a prepared connection string:

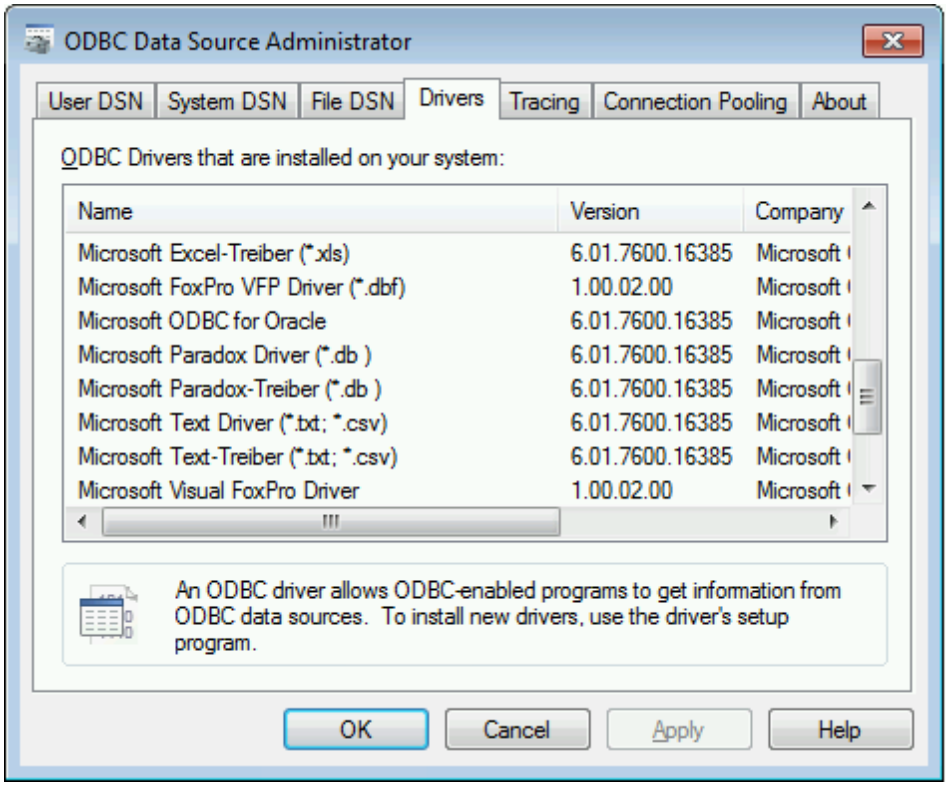
1. Start the database connection wizard.
2. Click **ODBC Connections**.
3. Select **Build a connection string**.
4. Paste the connection string into the provided box, and then click **Connect**.

Viewing the Available ODBC Drivers

You can view the ODBC drivers available on your operating system in the ODBC Data Source Administrator. You can access the ODBC Data Source Administrator (**Odbcad32.exe**) from the Windows Control Panel, under **Administrative Tools**. On 64-bit operating systems, there are two versions of this executable:

- The 32-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\SysWoW64** directory (assuming that **C:** is your system drive).
- The 64-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\System32** directory.

Any installed 32-bit database drivers are visible in the 32-bit version of ODBC Data Source Administrator, while 64-bit drivers—in the 64-bit version. Therefore, ensure that you check the database drivers from the relevant version of ODBC Data Source Administrator.



ODBC Data Source Administrator

If the driver to your target database does not exist in the list, or if you want to add an alternative driver, you will need to download it from the database vendor (see Database Drivers Overview). Once the ODBC driver is available on your system, you are ready to create ODBC connections with it (see Setting up an ODBC Connection).

7.2.1.5 Setting up a JDBC Connection

JDBC (Java Database Connectivity) is a database access interface which is part of the Java software platform from Oracle. JDBC connections are generally more resource-intensive than ODBC connections but may provide features not available through ODBC. It is generally recommended to use a JDBC connection if you are using database features not available through an ODBC connector, for example, support for the XML DB technology in Oracle databases. Due to insufficient information returned by the drivers, JDBC connections have the following limited functionality: (i) data editing is not possible for tables without a primary key; (ii) the **Execute for Data Editing** command in SQL Editor will not work.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. If you have not installed it already, check the official Java website for the download package and installation instructions.
- The JDBC drivers from the database vendor must be installed. If you are connecting to an Oracle database, note that some Oracle drivers are specific to certain JRE versions and

may require additional components and configuration. The documentation of your Oracle product (for example, the "Oracle Database JDBC Developer's Guide and Reference") includes detailed instructions about the configuration procedure for each JDBC driver.

- The operating system's `PATH` environment variable must include the path to the `bin` directory of the JRE or JDK installation directory, for example `C:\Program Files (x86)\Java\jre1.8.0_51\bin`.
- The `CLASSPATH` environment variable must include the path to the JDBC driver on your Windows operating system. When you install some database clients, the installer may configure this variable automatically. The documentation of the JDBC driver will typically include step-by-step instructions on setting the CLASSPATH variable (see also Configuring the CLASSPATH).

To set up a JDBC connection:

1. Start the database connection wizard.
2. Click **JDBC Connections**.
3. Do one of the following:
   a. Select a JDBC driver from the Driver list. This list contains any JDBC drivers configured through the `CLASSPATH` environment variable (see Configuring the CLASSPATH).
   b. Enter a Java class name.
4. Enter the username and password to the database in the corresponding boxes.
5. In the Database URL text box, enter the JDBC connection string in the format specific to your database type (see the JDBC connection formats below).
6. Click **Connect**.

JDBC connection formats

The following table describes the syntax of JDBC connection strings for common database types.

Database	JDBC Connection Format
Firebird	`jdbc:firebirdsql://<host>[:<port>]/<database path or alias>`
IBM DB2	`jdbc:db2://hostName:port/databaseName`
IBM Informix	`jdbc:informix-sqli://hostName:port/databaseName:INFORMIXSERVER=myserver`
Microsoft SQL Server	`jdbc:sqlserver://hostName:port;databaseName=name`
MySQL	`jdbc:mysql://hostName:port/databaseName`
Oracle	`jdbc:oracle:thin:@//hostName:port:databaseName`
Oracle XML DB	`jdbc:oracle:oci:@//hostName:port:databaseName`
PostgreSQL	`jdbc:postgresql://hostName:port/databaseName`
Sybase	`jdbc:sybase:Tds:hostName:port/databaseName`

Note: Syntax variations to the formats listed above are also possible (for example, the database URL may exclude the port or may include the username and password to the database).

Check the documentation of the database vendor for further details.

Configuring the CLASSPATH

The CLASSPATH environment variable is used by the Java Runtime Environment (JRE) to locate Java classes and other resource files on your operating system. When you connect to a database through JDBC, this variable must be configured to include the path to the JDBC driver on your operating system, and, in some cases, the path to additional library files specific to the database type you are using.

The following table lists sample file paths that must be typically included in the CLASSPATH variable. Importantly, you may need to adjust this information based on the location of the JDBC driver on your system, the JDBC driver name, as well as the JRE version present on your operating system. To avoid connectivity problems, check the installation instructions and any pre-installation or post-installation configuration steps applicable to the JDBC driver installed on your operating system.

Database	Sample CLASSPATH entries
Firebird	`C:\Program Files\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar`
IBM DB2	`C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;`
IBM Informix	`C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;`
Microsoft SQL Server	`C:\Program Files\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\enu\sqljdbc.jar`
MySQL	`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`
Sybase	`C:\sybase\jConnect-7_0\classes\jconn4.jar`

- Changing the CLASSPATH variable may affect the behavior of Java applications on your machine. To understand possible implications before you proceed, refer to the Java documentation.
- Environment variables can be user or system. To change system environment variables, you need administrative rights on the operating system.
- After you change the environment variable, restart any running programs for settings to take effect. Alternatively, log off or restart your operating system.

To configure the CLASSPATH on Windows 7:

1. Open the **Start** menu and right-click **Computer**.
2. Click **Properties**.
3. Click **Advanced system settings**.
4. In the **Advanced** tab, click **Environment Variables**,
5. Locate the CLASSPATH variable under user or system environment variables, and then click Edit. If the CLASSPATH variable does not exist, click **New** to create it.
6. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator (;).

To configure the CLASSPATH on Windows 8:

1. Right-click the Windows Start button, and then click **System**.
2. Click **Advanced System Settings**.
3. Click **Environment Variables**.
4. Locate the CLASSPATH variable under user or system environment variables, and then click **Edit**. If the CLASSPATH variable does not exist, click **New** to create it.
5. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator (;).

7.2.1.6 Setting up a SQLite Connection

SQLite (http://www.sqlite.org) 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. Also, note the following:

- SQLite databases are supported in the MapForce BUILT-IN transformation language (either when you preview the mapping or when you run a MapForce Server execution file).
- SQLite databases are not supported in user-defined functions (UDF).
- On Linux, statement execution timeout for SQLite databases is not supported.
- Full text search tables are not supported
- SQLite allows values of different data types in each row of a given table. In MapForce, all processed values must be compatible with the declared column type; therefore, run-time errors may occur if your SQLite database has row values which are not the same as the declared column type.
- If your mapping should write data to a SQLite database, and if you don't have the target database file already, you will need to create it separately. In this case, you can either create it with a tool such as DatabaseSpy (http://www.altova.com/databasespy.html) or download the SQLite command-line shell from the official website, and create the database file from the command line (see also Example: Mapping data from XML to SQLite). For complete reference to SQLite command syntax, refer to the official SQLite documentation.

Connecting to an Existing SQLite Database

To connect to an existing SQLite database:

1. Run the database connection wizard (see Starting the Database Connection Wizard).
2. Select **SQLite**, and then click **Next**.
3. Browse for the SQLite database file, or enter the path (either relative or absolute) to the database. The **Connect** button becomes enabled once you enter the path to a SQLite database file.
4. Click **Connect**.

Example: Mapping data from XML to SQLite

This example walks you through the steps required to create a MapForce mapping which reads data from an XML file and writes it to a SQLite database. The example is accompanied by a sample mapping design (.mfd) file. If you want to look at the sample file before starting this example, you can open it from the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\XMLtoSQLite.mfd**.

The goal of the example is to insert data from an XML file into a SQLite database. To accomplish the goal of the example, you will need an empty SQLite database to which data will be written. For the scope of this example, you will create the SQLite database with the command-line shell available from the official SQLite website, although this can be done with other tools as well (for example, Altova DatabaseSpy).

To create the SQLite database:

1. Download the SQLite command-line shell for Windows from the SQLite download page (http://www.sqlite.org/download.html) and unpack the .zip archive to a directory on your local machine (for the scope of this example, use **c:\sqlite**).
2. Run **c:\sqlite\sqlite3.exe** and enter the following statement:

   ```
   create table articles (number smallint, name varchar(10), singleprice
   real);
   ```

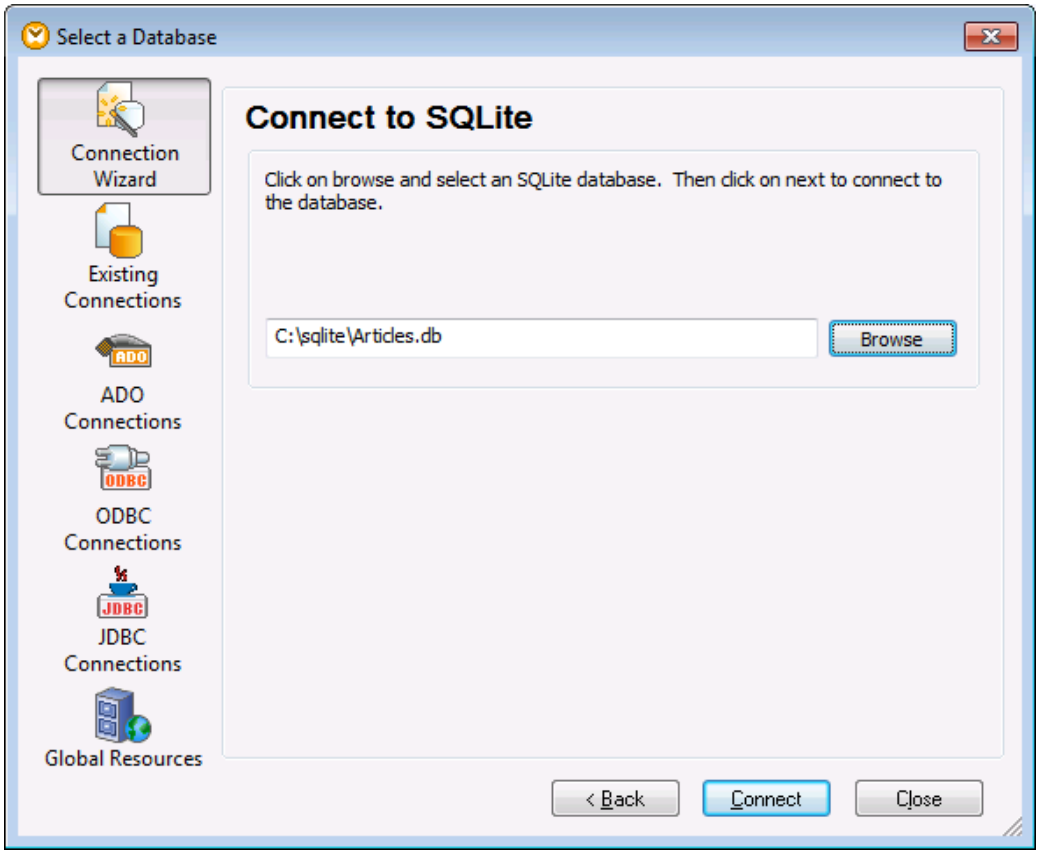
   This creates the table `articles` in the in-memory database. The table `articles` consists of three columns: *number*, *name*, and *singleprice*. The purpose of these columns is to store data from the elements with the same name defined in the source XML schema. Each column is declared with a data type suitable for the data expected to be stored in that column.
3. Run the command:

   ```
   .save Articles.db
   ```

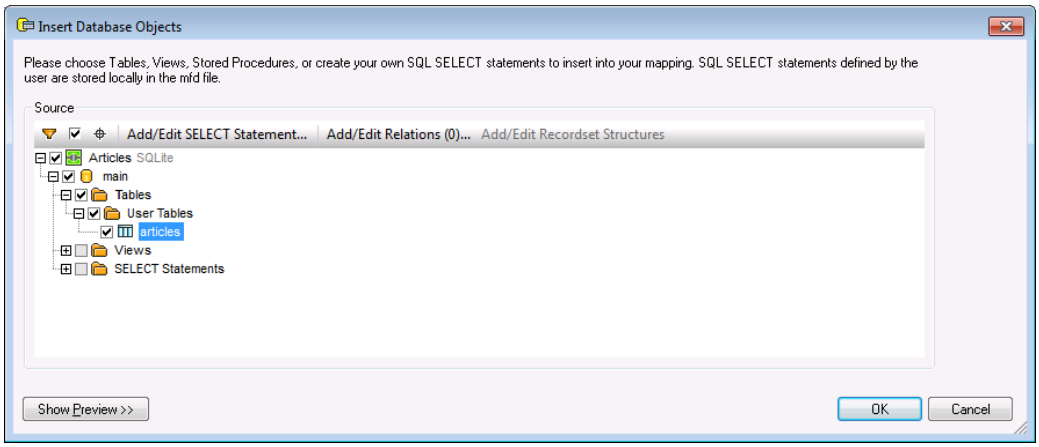
   This saves the in-memory database to the current working path: **c:\sqlite\Articles.db**. Note that you will need to refer to this path in subsequent steps.

To create the XML to SQLite mapping design:

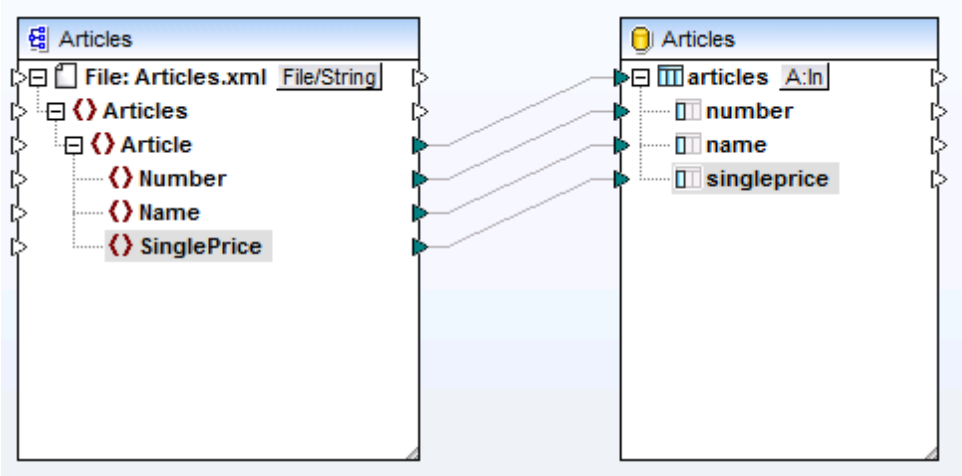
1. Run MapForce and make sure that the transformation language is set to BUILT-IN (use the menu command **Output | Built-in Execution Engine**).
2. Add to the mapping area the file **Articles.xml** located in the **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial** folder (use the menu command **File | Insert XML Schema/File**).
3. Add to the mapping area the database **Articles.db** created in previous steps (use the menu command **File | Insert Database**), and then select **SQLite**.



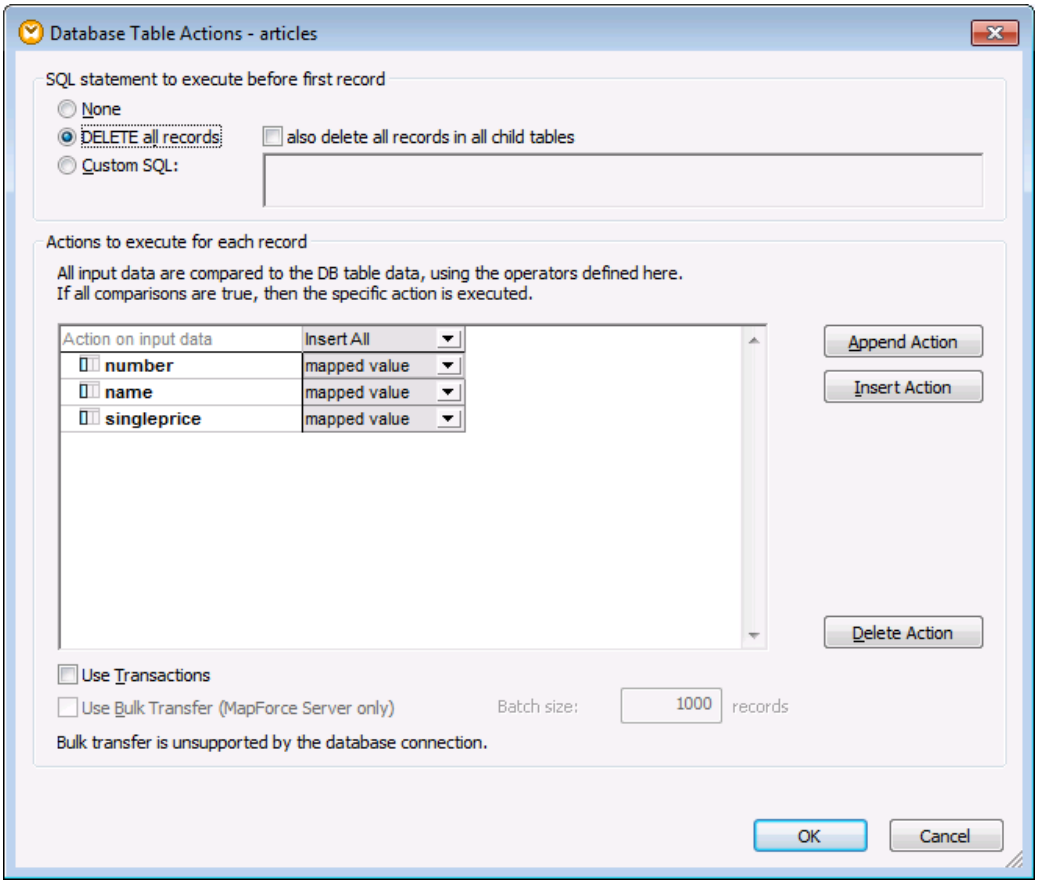
4. When prompted to choose the database objects, select the `articles` table.



5. Draw the connections as shown below:



6. Click the **A:In** button on the database component and select the **Delete All records** option. This ensures that, every time the mapping is executed, all existing database rows are first deleted, in order to prevent duplication.



7. Click the **Output** tab of the main mapping window. MapForce executes the mapping using the built-in execution engine and displays the create SQL query in the Output window.

```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
C:\sqlite\Articles.db
*/

PRAGMA foreign_keys = ON;

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (1, 'T-Shirt', 25)

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (2, 'Socks', 2.3)

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (3, 'Pants', 34)

INSERT INTO "articles" ("number", "name", "singleprice") VALUES (4, 'Jacket', 57.5)

```

8. Run the SQL script to populate the database (use the menu command **Output | Run SQL-Script)**. If MapForce does not encounter any runtime errors, the records are inserted into the SQLite database.

To check whether data was correctly inserted into the SQLite database:

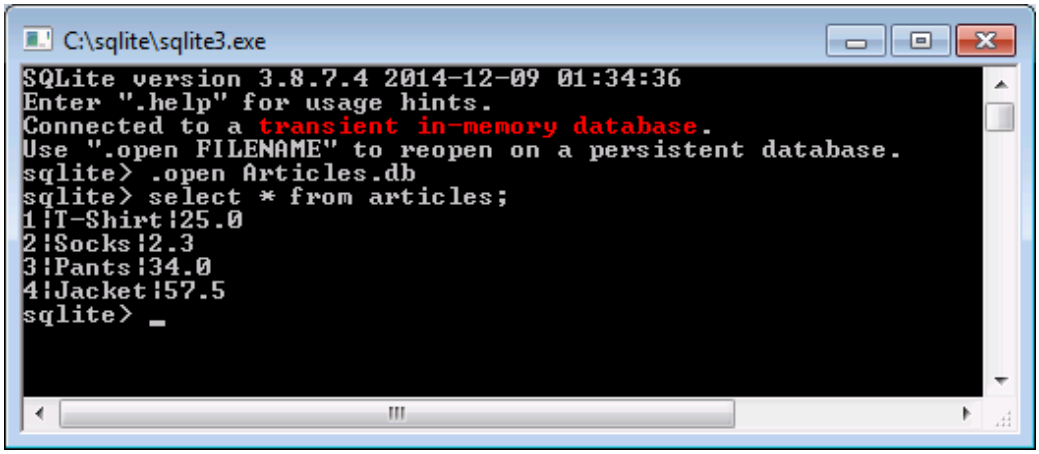
1. Run the file **c:\sqlite\sqlite3.exe** and open the database with the command:

```
.open Articles.db
```

2. Run the following select statement:

```
select * from articles;
```

This query returns all the records in the `articles` table, separated by the broken bar (¦) character, as follows:



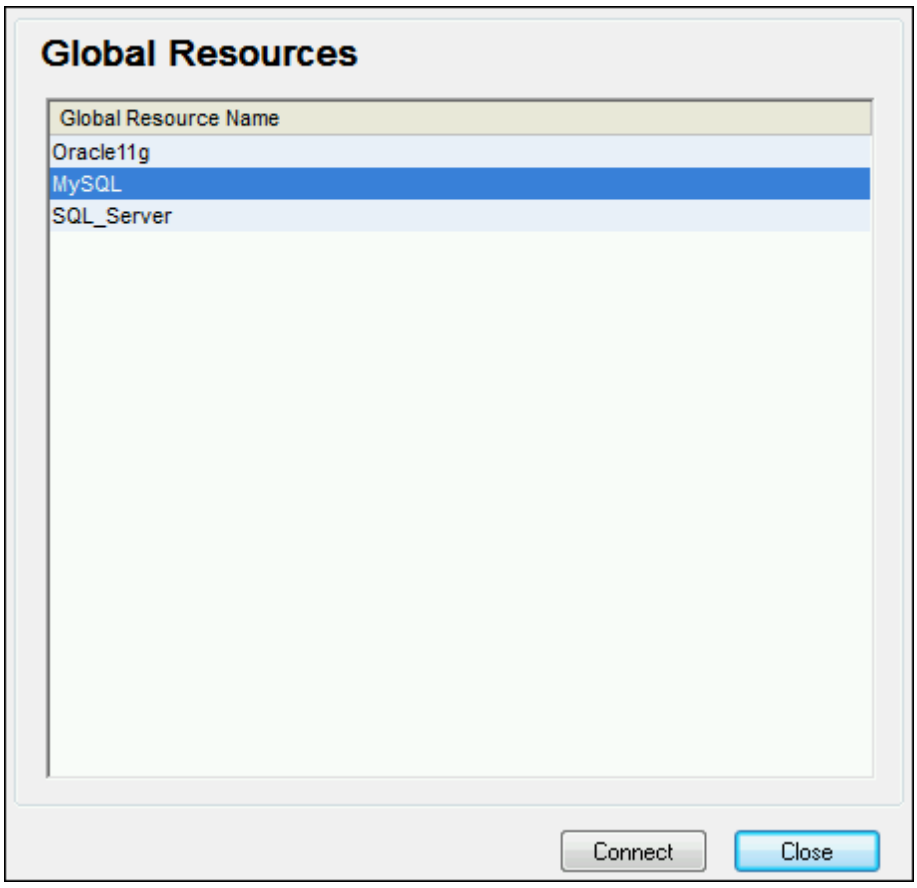
As shown above, the query returns four rows. This corresponds to the number of records in the source XML file, which was the intended goal of this example.

7.2.1.7 Using a Connection from Global Resources

If you have previously configured a database connection to be available as a global resource, you can reuse the connection at any time (even across different Altova applications).

To use a database connection from Global Resources:

1. Start the database connection wizard.
2. Click **Global Resources**. Any database connections available as global resources are listed.



3. Select the database connection record, and click **Connect**.

Tip: To get additional information about each global resource, move the mouse cursor over the global resource.

7.2.1.8 Examples

This section includes sample procedures for connecting to a database from MapForce. Note that your Windows machine, the network environment, and the database client or server software is likely to have a configuration that is not exactly the same as the one presented in the following examples.

Note: For most database types, it is possible to connect using more than one data access technology (ADO, 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.

This section includes the following topics:

Connecting to Firebird (ODBC)

This topic provides sample instructions for connecting to a Firebird 2.5.4 database running on a Linux server.

Prerequisites:

- The Firebird database server is configured to accept TCP/IP connections from clients.
- The Firebird ODBC driver must be installed on your operating system. This example uses the Firebird ODBC driver version 2.0.3.154 downloaded from the Firebird website (http://www.firebirdsql.org/).
- The Firebird client must be installed on your operating system. Note that there is no standalone installer available for the Firebird 2.5.4 client; the client is part of the Firebird server installation package. You can download the Firebird server installation package from the Firebird website (http://www.firebirdsql.org/), look for "Windows executable installer for full Superclassic/Classic or Superserver". To install only the client files, choose "**Minimum client install - no server, no tools**" when going through the wizard steps.

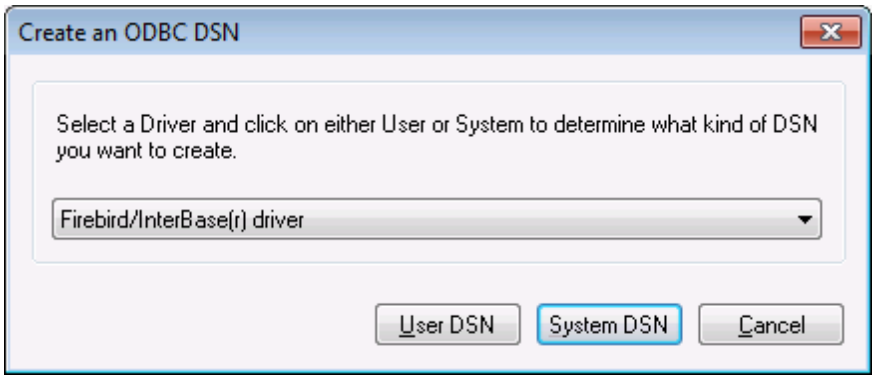
  Important:

  - The platform of both the Firebird ODBC driver and client (32-bit or 64-bit) must correspond to that of MapForce.
  - The version of the Firebird client must correspond to the version of Firebird server to which you are connecting.

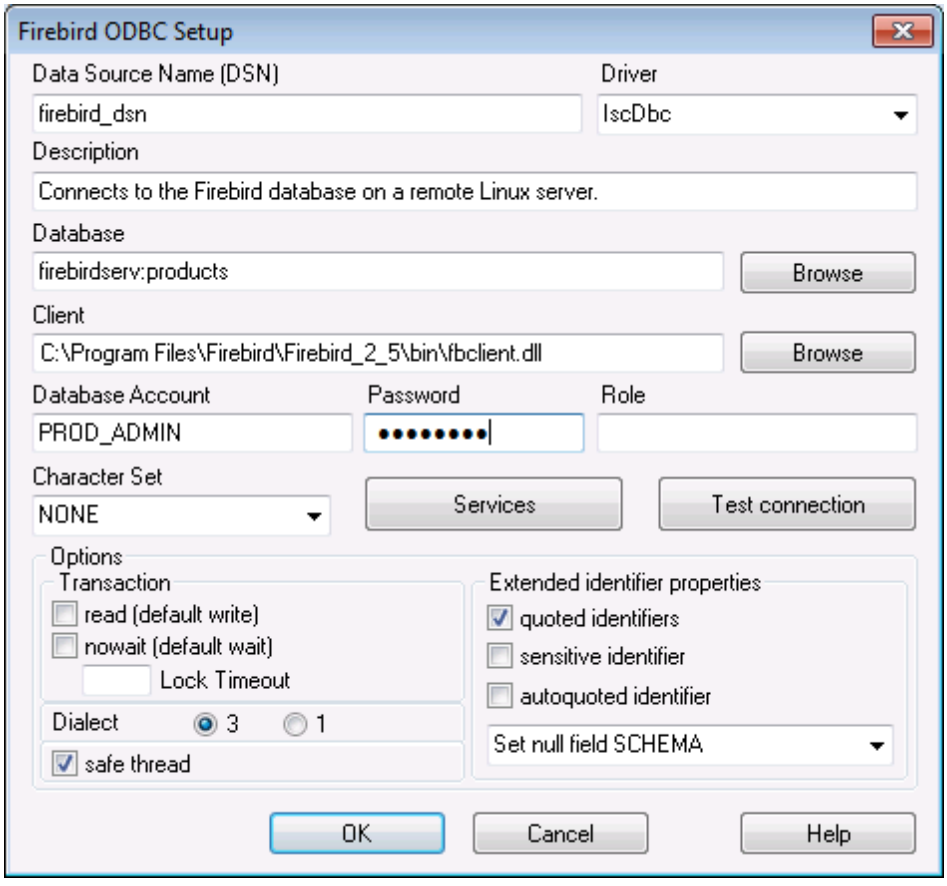
- You have the following database connection details: server host name or IP address, database path (or alias) on the server, user name, and password.

To connect to Firebird via ODBC:

1. Start the database connection wizard.
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .



4. Select the Firebird driver, and then click **User DSN** (or **System DSN**, depending on what you selected in the previous step). If the Firebird driver is not available in the list, make sure that it is installed on your operating system (see also Viewing the Available ODBC Drivers).



5. Enter the database connection details as follows:

Data Source Name (DSN)	Enter a descriptive name for the data source you are creating.

Database	Enter the server host name or IP address, followed by a colon, followed by the database alias (or path). In this example, the host name is `firebirdserv`, and the database alias is `products`, as follows:
	
	`firebirdserv:products`
	
	Using a database alias assumes that, on the server side, the database administrator has configured the alias *products* to point to the actual Firebird (.fdb) database file on the server (see the Firebird documentation for more details).
	
	You can also use the server IP address instead of the host name, and a path instead of an alias; therefore, any of the following sample connection strings are valid:
	
	`firebirdserver:/var/Firebird/databases/butterflies.fdb`
	`127.0.0.1:D:\Misc\Lenders.fdb`
	
	If the database is on the local Windows machine, click **Browse** and select the Firebird (.fdb) database file directly.

Client	Enter the path to the **fbclient.dll** file. By default, this is the `bin` subdirectory of the Firebird installation directory.
Database Account	Enter the database user name supplied by the database administrator (in this example, `PROD_ADMIN`).
Password	Enter the database password supplied by the database administrator.

6. Click **OK**.

Connecting to Firebird (JDBC)

This topic provides sample instructions for connecting to a Firebird database server through JDBC.

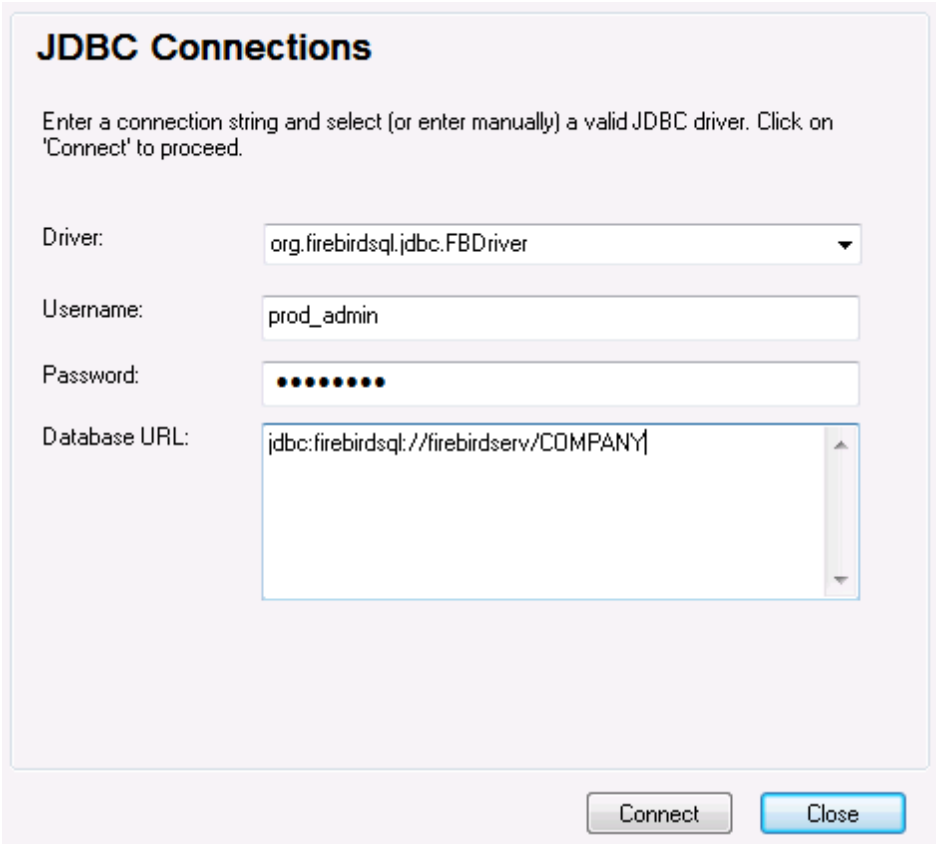
Prerequisites:

- Java Runtime Environment (JRE) or Java Development Kit (JDK) must be installed on your operating system.
- The operating system's `PATH` environment variable must include the path to the `bin` directory of the JRE or JDK installation directory, for example `C:\Program Files (x86)\Java\jre1.8.0_51\bin`.

- The Firebird JDBC driver must be installed on your operating system. This example uses *Jaybird 2.2.8* downloaded from the Firebird website (http://www.firebirdsql.org/).
- The operating system's CLASSPATH environment variable must include the path to the Jaybird driver, for example **C:\jdbc\firebird\jaybird-full-2.2.8.jar**. See also Configuring the CLASSPATH.
- You have the following database connection details: host, database path or alias, username, and password.

To connect to Firebird through JDBC:

1. Start the database connection wizard.
2. Click **JDBC Connections**.
3. In the Driver box, select **org.firebirdsql.jdbc.FBDriver**. If the entry is not available, check if the CLASSPATH and PATH environment variables are set correctly (see the prerequisites above).



4. Enter the username and password to the database in the corresponding text boxes.
5. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:firebirdsql://<host>[:<port>]/<database path or alias>
```

6. Click **Connect**.

Connecting to IBM DB2 (ODBC)

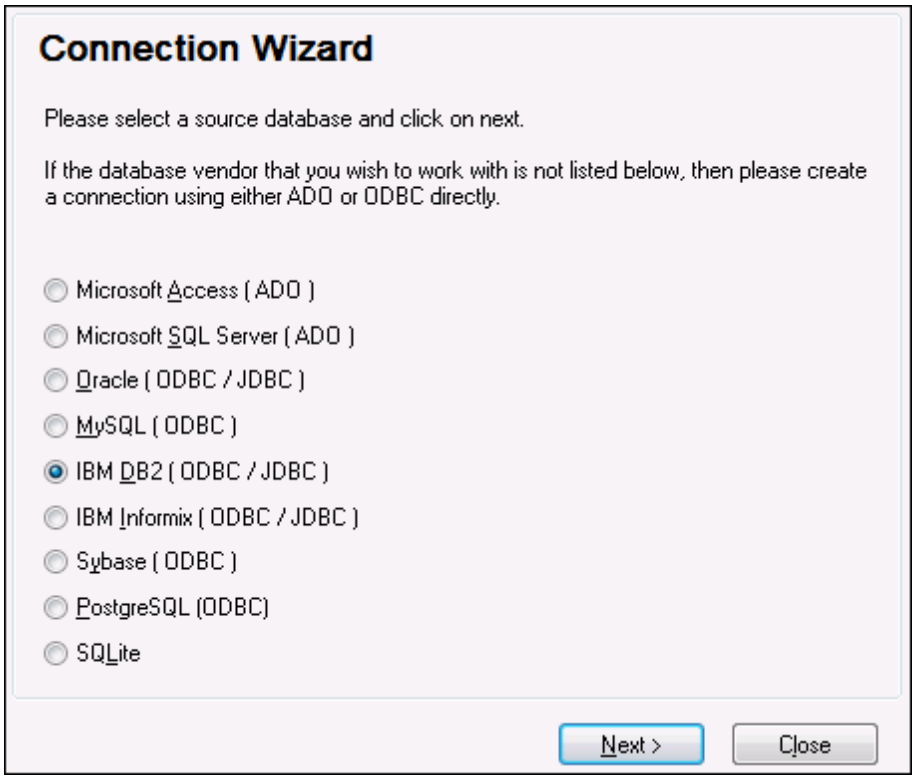
This topic provides sample instructions for connecting to an IBM DB2 database through ODBC.

Prerequisites:

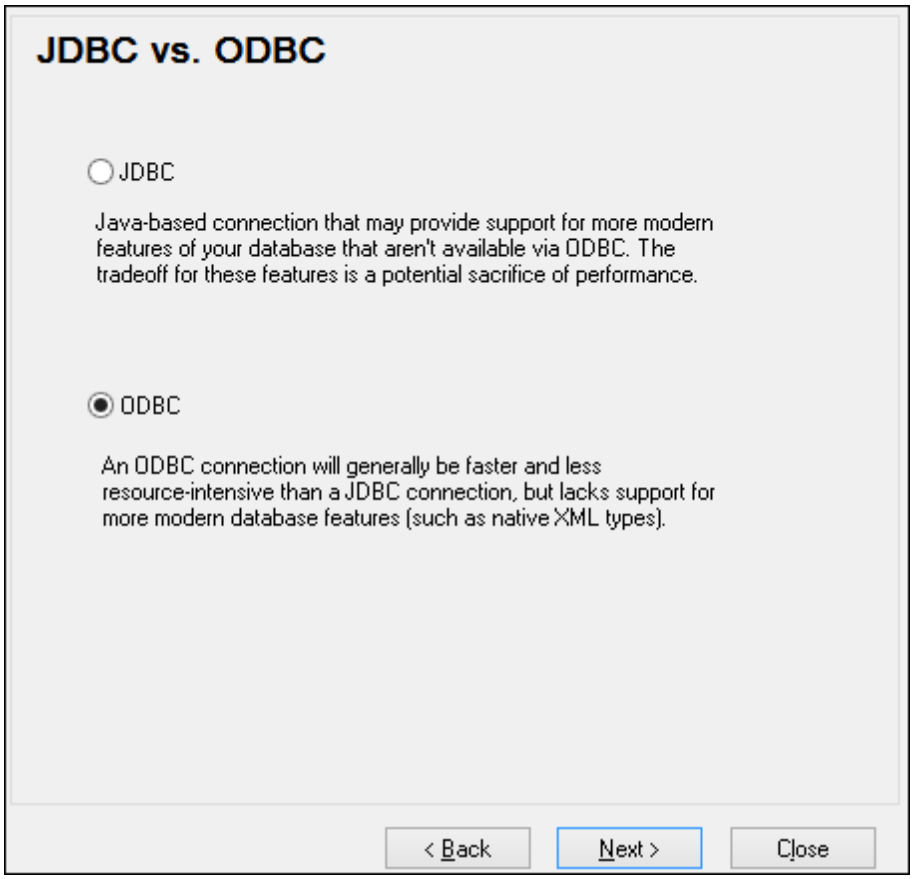
- IBM Data Server Client must be installed and configured on your operating system (this example uses IBM Data Server Client 9.7). For installation instructions, check the documentation supplied with your IBM DB2 software. After installing the IBM Data Server Client, check if the ODBC drivers are available on your machine (see Viewing the Available ODBC Drivers).
- Create a database alias. There are several ways to do this:
  - 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.

To connect to IBM DB2:

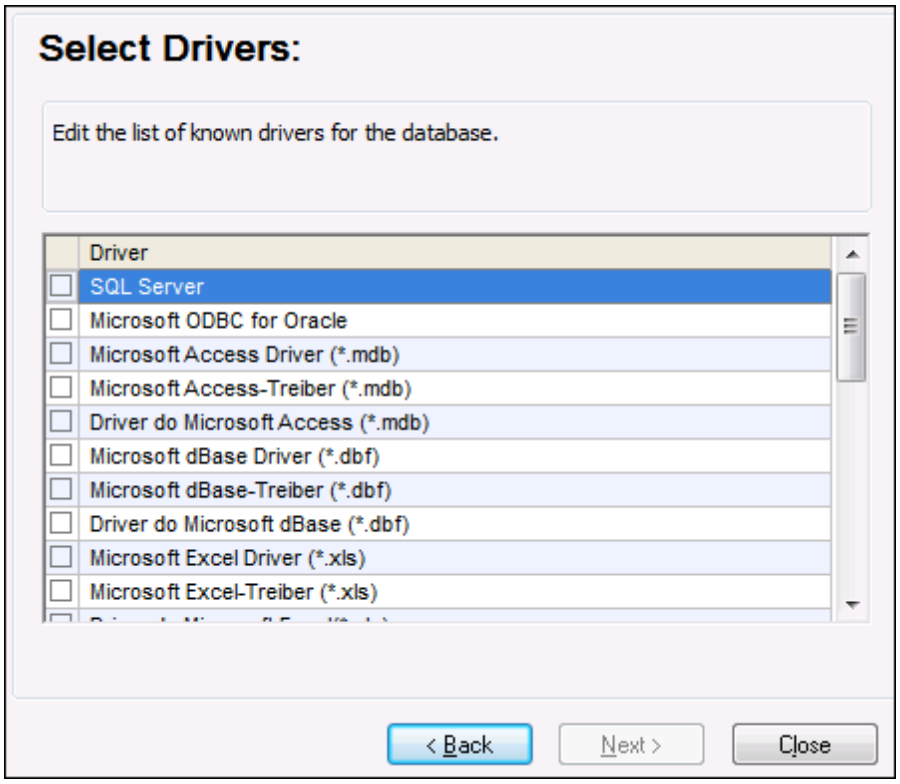
1. Start the database connection wizard and select **IBM DB2 (ODBC/JDBC)**.



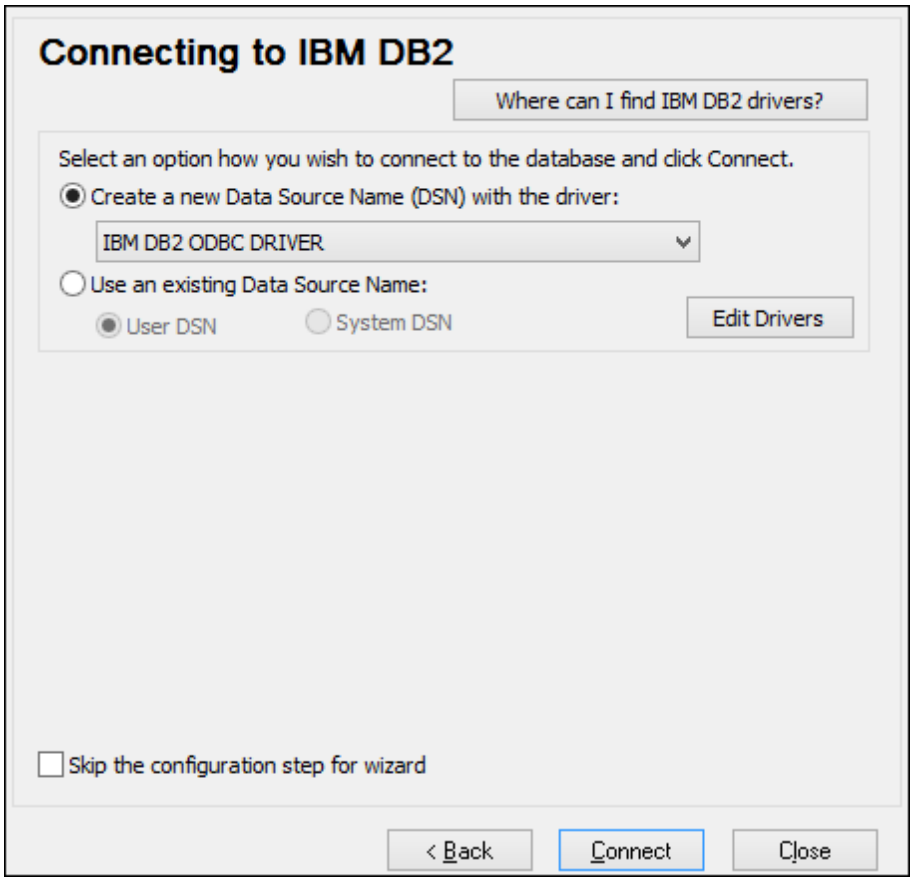
2. Click **Next**.



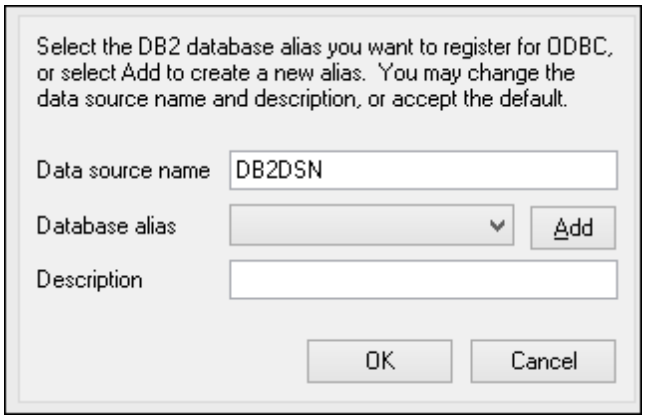
3. Select **ODBC**, and click **Next**. If prompted to edit the list of known drivers for the database, select the database drivers applicable to IBM DB2 (see Prerequisites), and click **Next**.



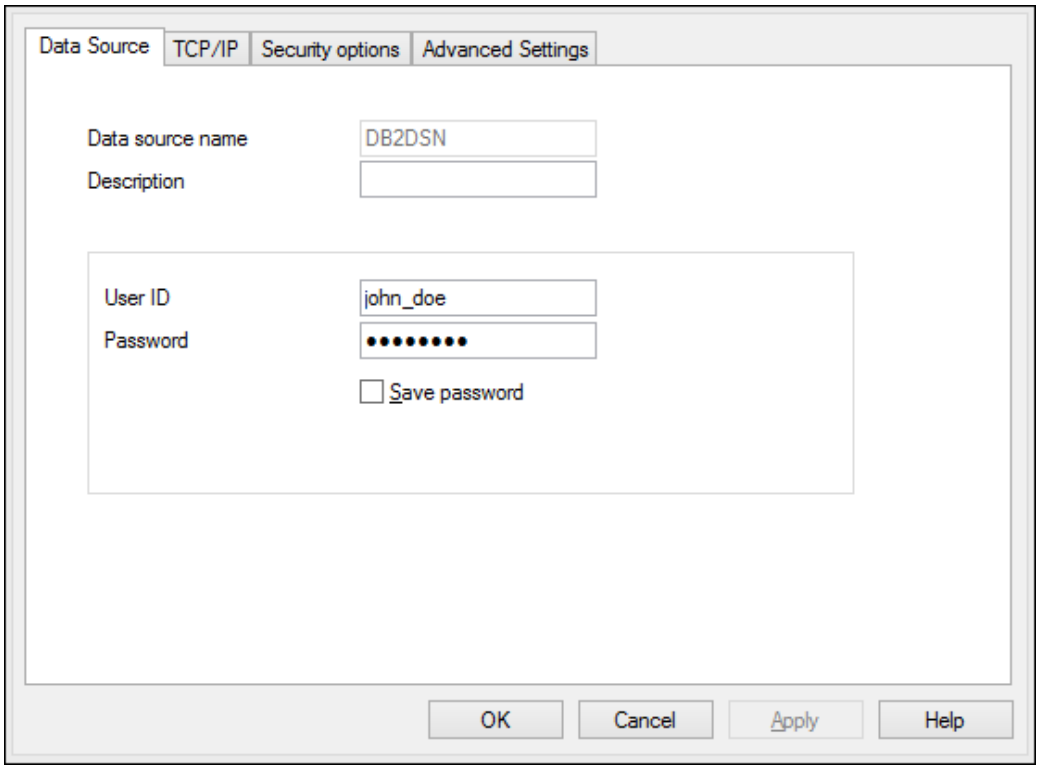
4. Select the IBM DB2 driver from the list, and then click **Connect**. (To edit the list of available drivers, click **Edit Drivers**, and then check or uncheck the IBM DB2 drivers you wish to add or remove, respectively.)



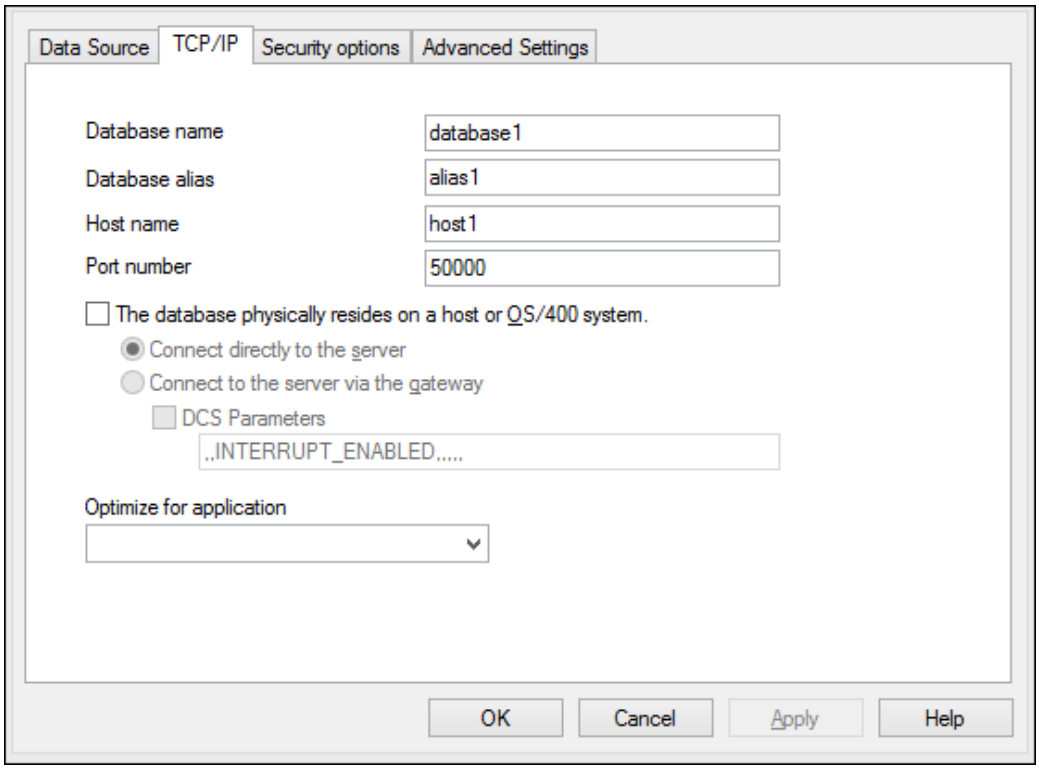
5. Enter a data source name (in this example, **DB2DSN**), and then click **Add**.



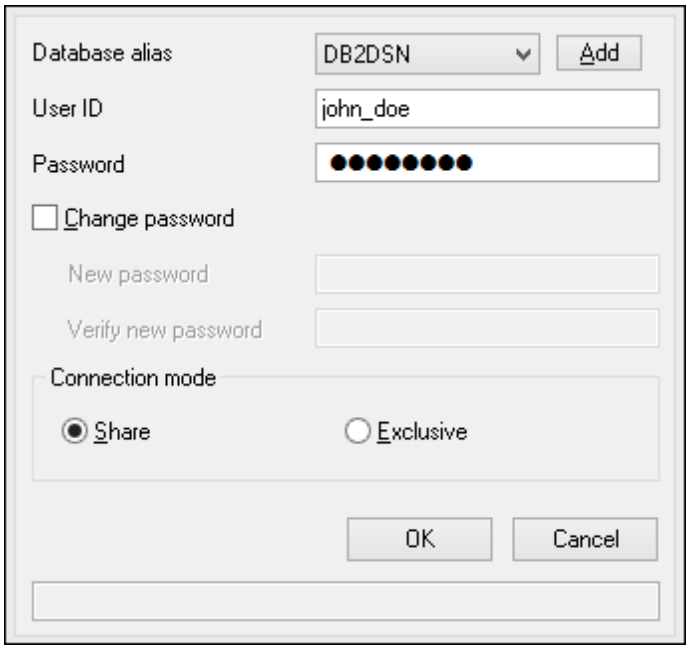
6. On the **Data Source** tab, enter the user name and password to the database.



7. On the **TCP/IP** tab, enter the database name, a name for the alias, the host name and the port number, and then click OK.



8. Enter again the username and password, and then click **OK**.

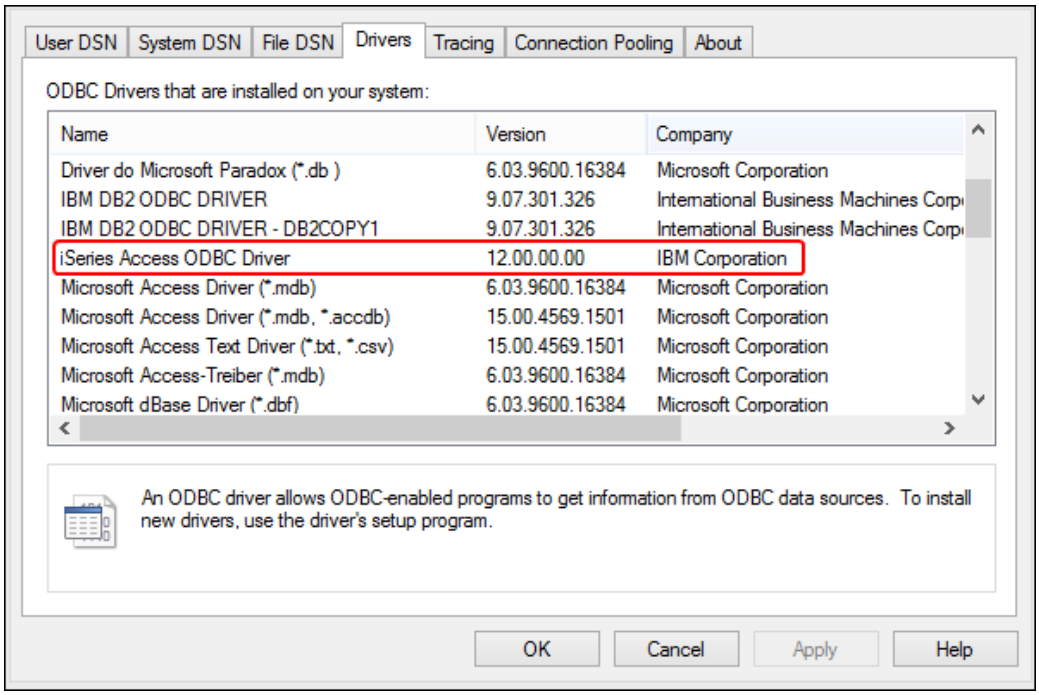


Connecting to IBM DB2 for i (ODBC)

This topic provides sample instructions for connecting to an *IBM DB2 for i* database through ODBC.

Prerequisites:

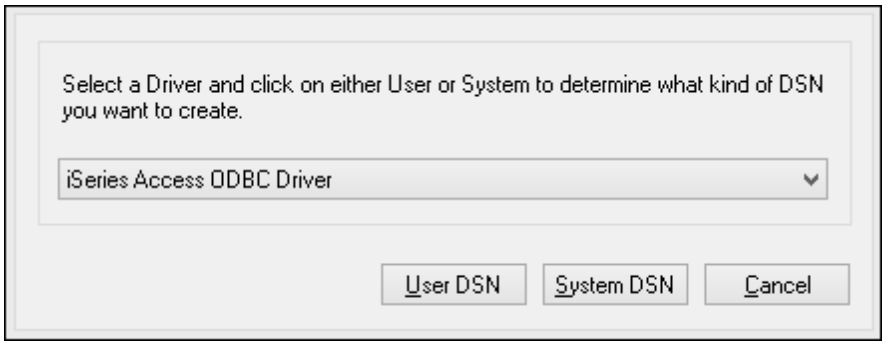
- *IBM System i Access for Windows* must be installed on your operating system (this example uses *IBM System i Access for Windows V6R1M0*). For installation instructions, check the documentation supplied with your *IBM DB2 for i* software. After installation, check if the ODBC driver is available on your machine (see Viewing the Available ODBC Drivers).



- You have the following database connection details: the I.P. address of the database server, database user name, and password.
- Run *System i Navigator* and follow the wizard to create a new connection. When prompted to specify a system, enter the I.P. address of the database server. After creating the connection, it is recommended to verify it (click on the connection, and select **File** > **Diagnostics** > **Verify Connection**). If you get connectivity errors, contact the database server administrator.

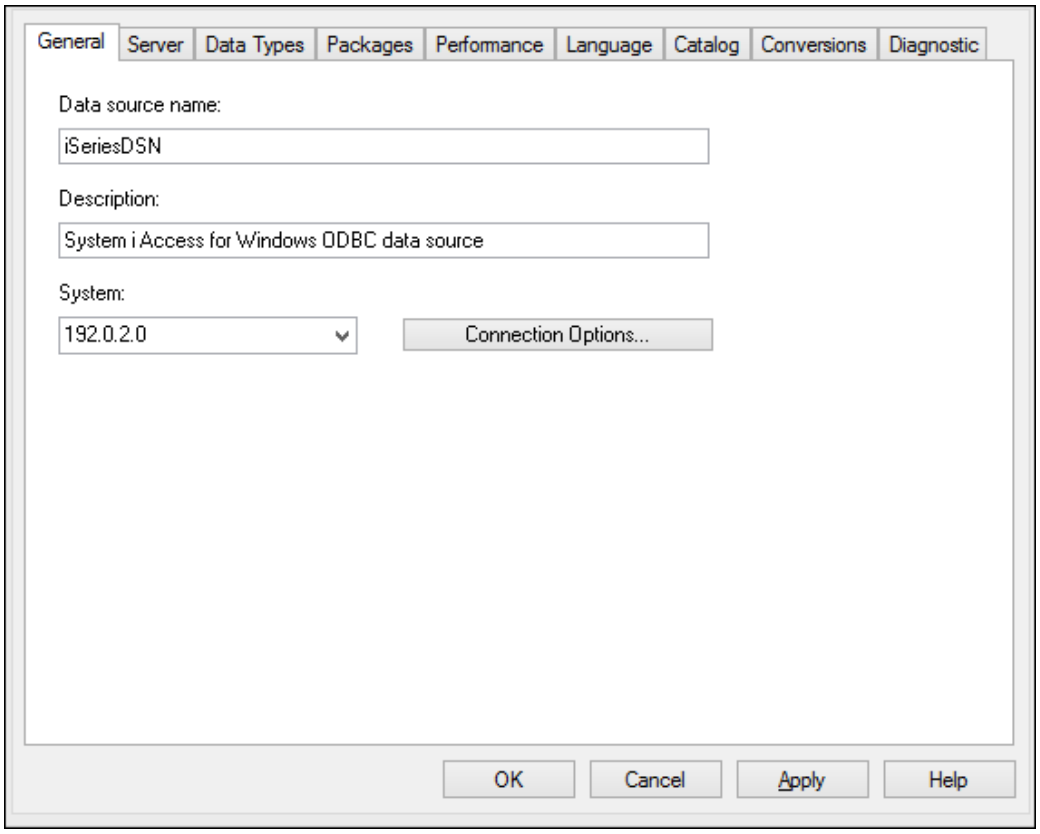
To connect to IBM DB2 for i:

1. Start the database connection wizard.
2. Click **ODBC connections**.
3. Click **User DSN** (alternatively, click **System DSN**, or **File DSN**, in which case the subsequent instructions will be similar).
4. Click **Add** .
5. Select the **iSeries Access ODBC Driver** from the list, and click **User DSN** (or **System DSN**, if applicable).

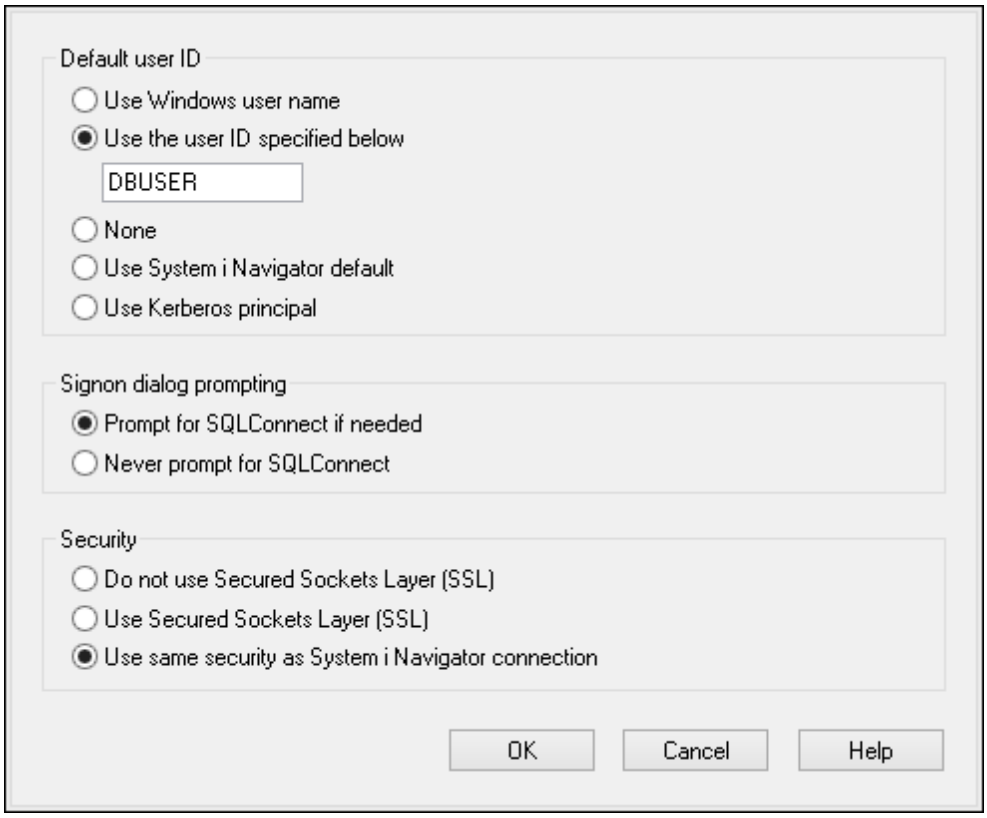


6. Enter a data source name and select the connection from the System combo box. In this

example, the data source name is **iSeriesDSN** and the System is **192.0.2.0**.



7. Click **Connection Options**, select **Use the User ID specified below** and enter the name of the database user (in this example, **DBUSER**).



8. Click **OK**. The new data source becomes available in the list of DSNs.
9. Click **Connect**.
10. Enter the user name and password to the database when prompted, and then click **OK**.

Connecting to IBM Informix (JDBC)

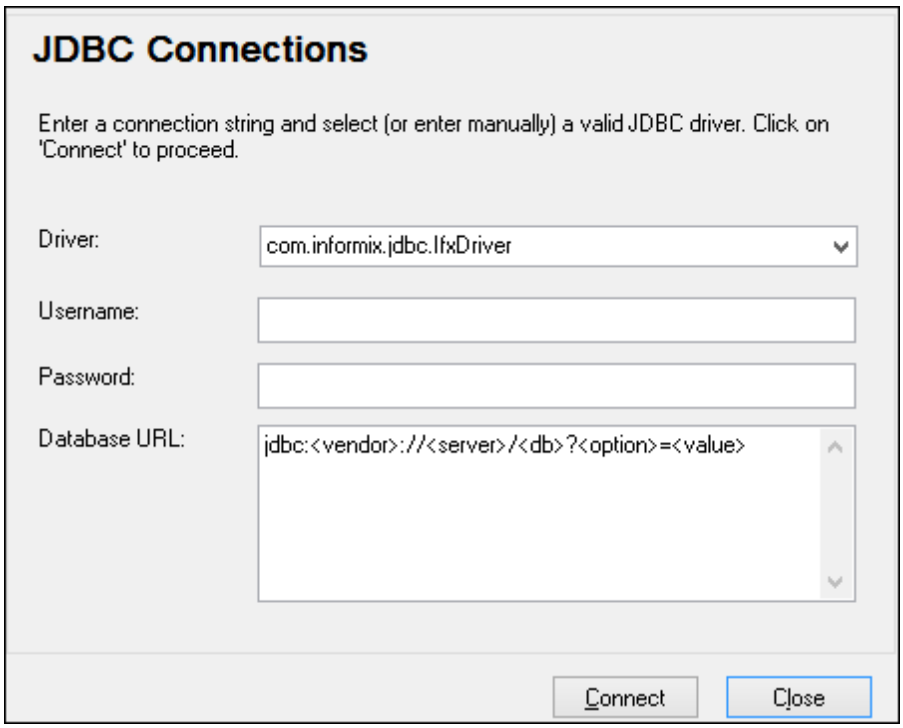
This topic provides sample instructions for connecting to an IBM Informix database server through JDBC.

Prerequisites:

- Java Runtime Environment (JRE) must be installed on your operating system.
- The JDBC driver must be installed 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").
- The operating system's CLASSPATH environment variable includes the path where the Informix JDBC driver (**ifxjdbc.jar**) was installed. In this example, the Informix JDBC driver is installed in the directory **C:\Informix_JDBC_Driver**, and the value of CLASSPATH variable is **C:\Informix_JDBC_Driver\lib\ifxjdbc.jar**. For more information, see Configuring the CLASSPATH.
- You have the following database connection details: host, name of the Informix server, database, port, username, and password.

To connect to IBM Informix through JDBC:

1. Start the database connection wizard.
2. Click **JDBC Connections**.
3. Select the Informix JDBC driver from the list of available JDBC drivers (in this example, **com.informix.jdbc.IfxDriver**). If the list does not contain an Informix driver, it is either not installed correctly, or not included in the CLASSPATH variable (see the list of prerequisites above).



4. Enter the username and password to the database in the corresponding text boxes.
5. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:informix-sqli://hostName:port/
databaseName:INFORMIXSERVER=myserver;
```

6. Click **Connect**.

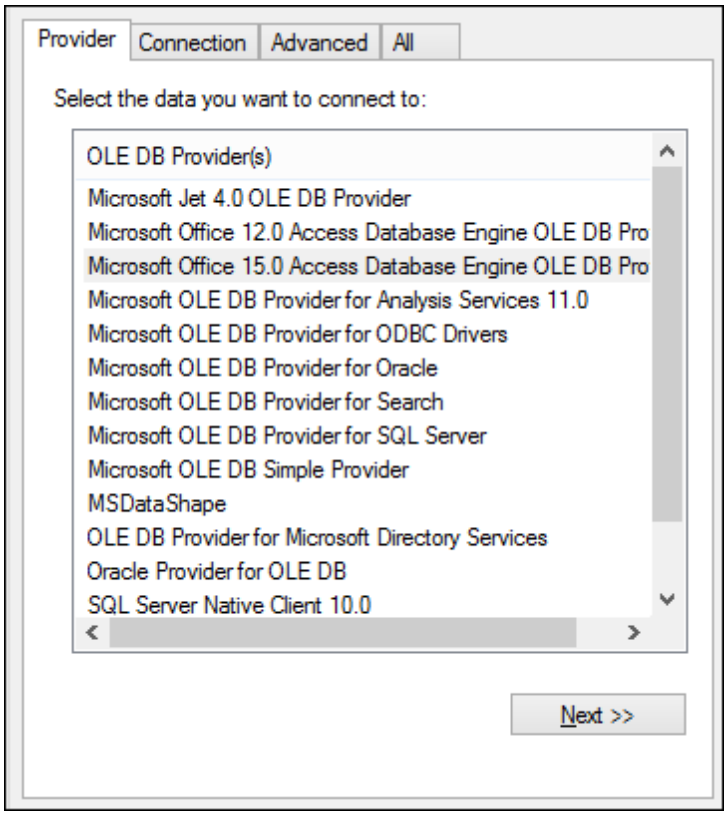
Connecting to Microsoft Access (ADO)

A simple way to connect to a Microsoft Access database is to follow the wizard and browse for the database file, as shown in Connecting to an Existing Microsoft Access Database. An alternative approach is to set up an ADO connection explicitly, as shown in this topic. This approach is useful if your database is password-protected.

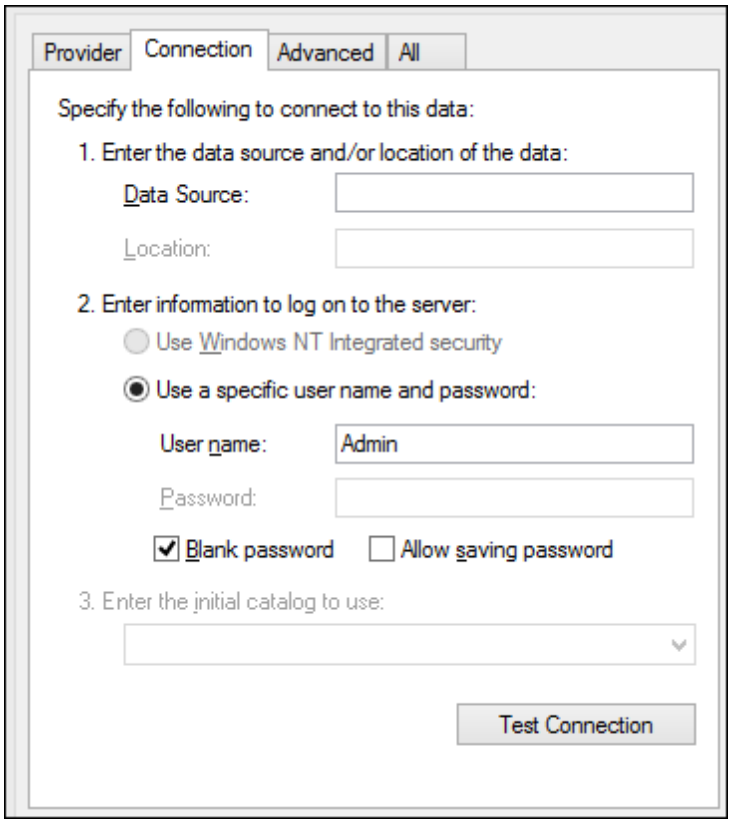
It is also possible to connect to Microsoft Access through an ODBC connection, but there are some limitations in this scenario, so it is best to avoid it.

To connect to a password-protected Microsoft Access database:

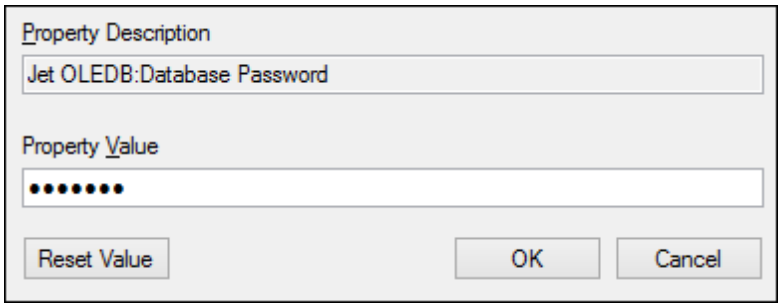
1. Start the database connection wizard.
2. Click **ADO Connections**.
3. Click **Build**.



4. Select the **Microsoft Office 15.0 Access Database Engine OLE DB Provider**, and then click **Next**.



5. In the Data Source box, enter the path to the Microsoft Access file. Because the file is on the local network share **U:\Departments\Finance\Reports\Revenue.accdb**, we will convert it to UNC format, and namely **\\server1\\dfs\Departments\Finance\Reports\Revenue.accdb**, where **server1** is the name of the server and **dfs** is the name of the network share.
6. On the **All** tab, double click the **Jet OLEDB:Database Password** property and enter the database password as property value.

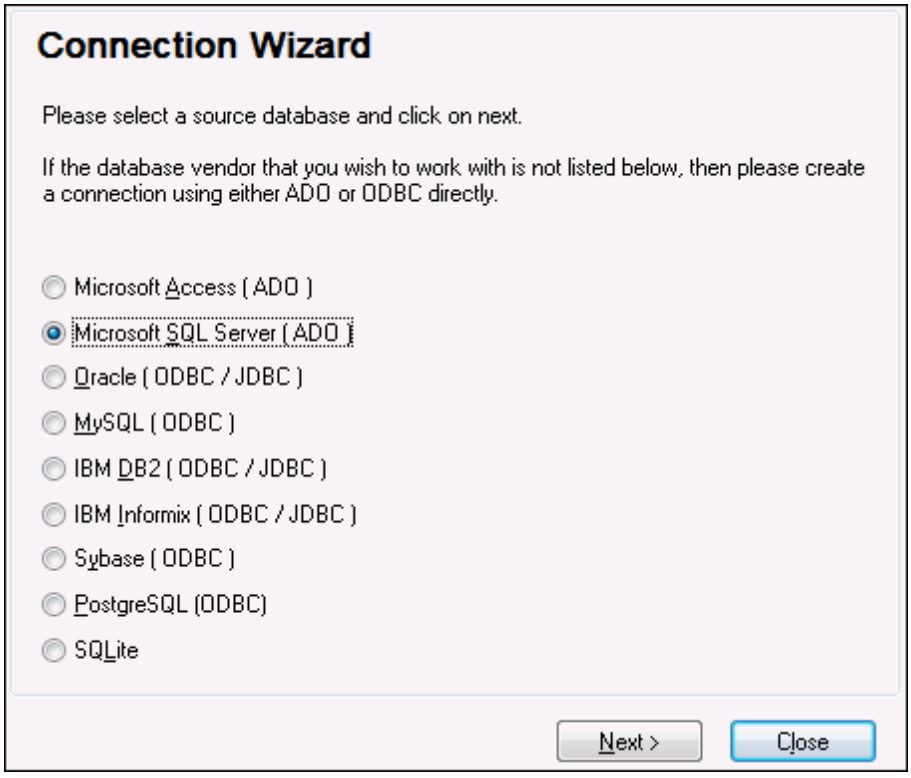


Note: If you are still unable to connect, locate the workgroup information file (**System.MDW**) applicable to your user profile (see http://support.microsoft.com/kb/305542 for instructions), and set the value of the **Jet OLEDB: System database** property to the path of the **System.MDW** file.

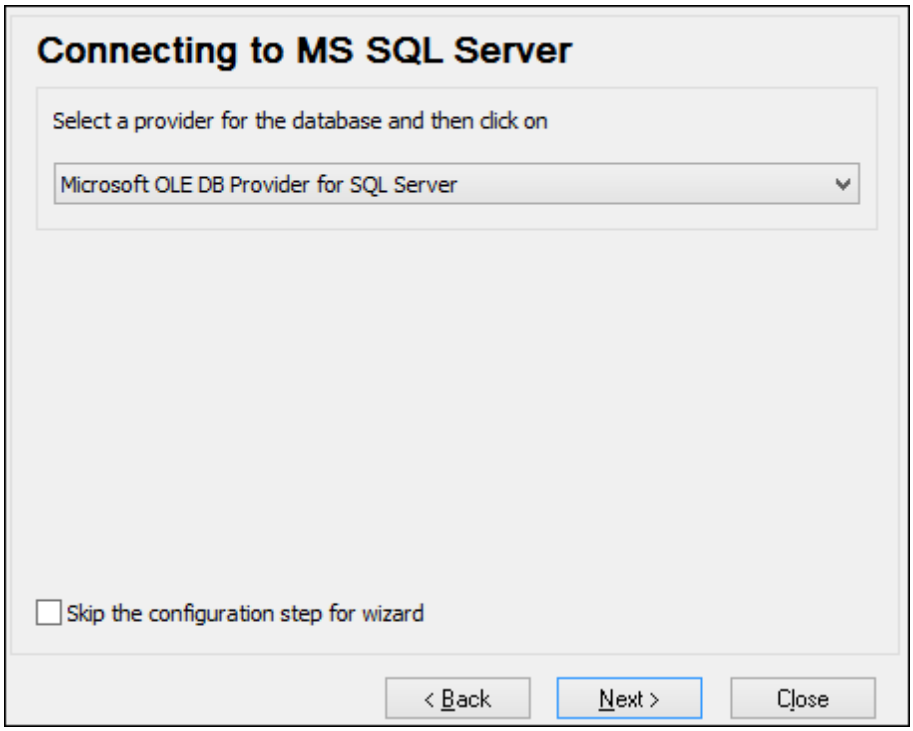
Connecting to Microsoft SQL Server (ADO)

To connect to SQL Server using the Microsoft OLE DB Provider:

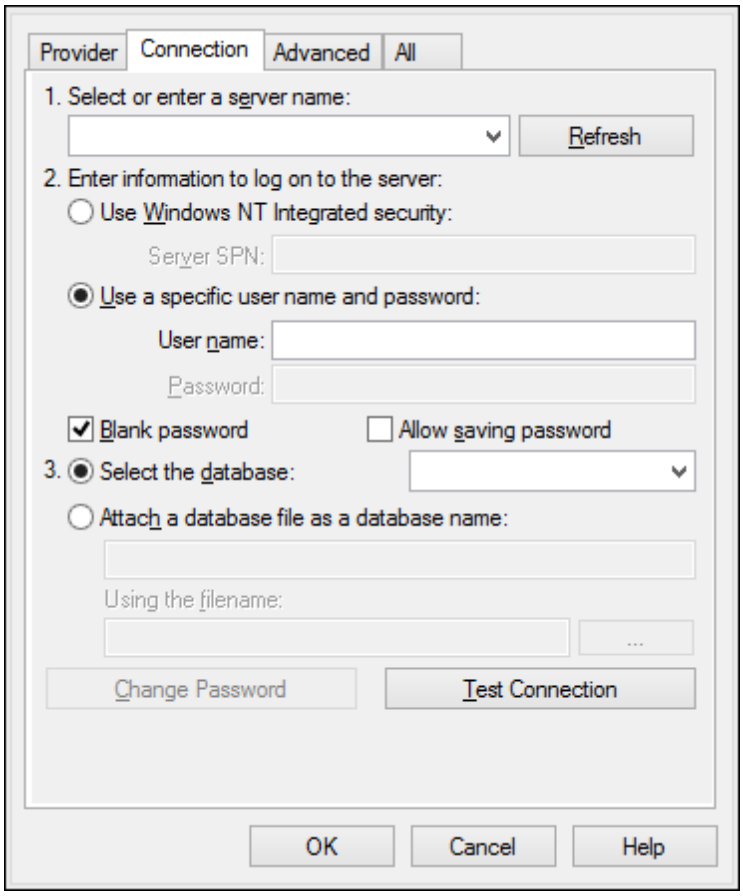
1. Start the database connection wizard.



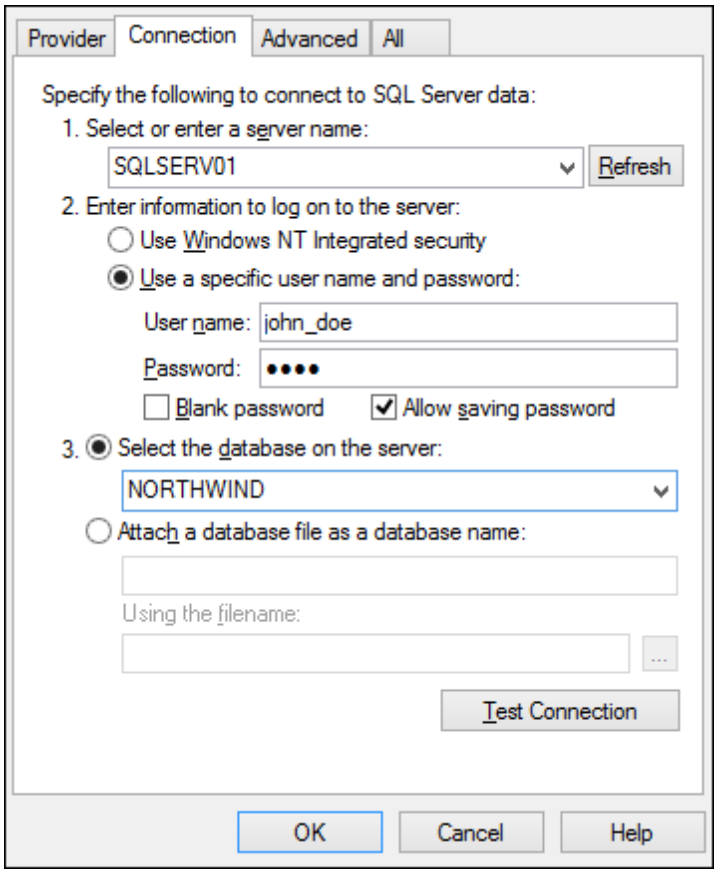
2. Select **Microsoft SQL Server (ADO)**, and then click **Next**. The list of available ADO drivers is displayed.



3. Select **Microsoft OLE DB Provider for SQL Server**, and then click **Next**.



4. Select or enter the name of the database server (in this example, **SQLSERV01**). To view the list of all servers on the network, expand the drop-down list.
5. If the database server was configured to allow connections from users authenticated on the Windows domain, select **Use Windows NT integrated security**. Otherwise, select **Use a specific user name and password**, and type them in the relevant boxes.
6. Select the database to which you are connecting (in this example, **NORTHWIND**).
7. To test the connection at this time, click **Test Connection**. This is an optional, recommended step.
8. Do one of the following:
   a. Select the **Allow saving password** check box.
   b. On the **All** tab, change the value of the **Persist Security Info** property to **True**.

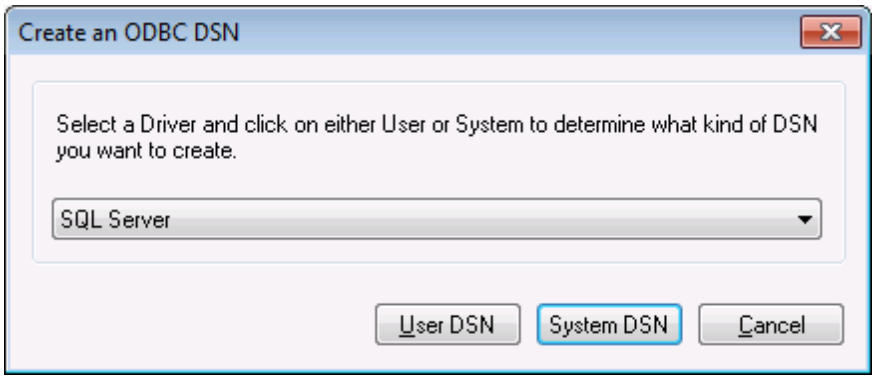


9. Click **OK**.

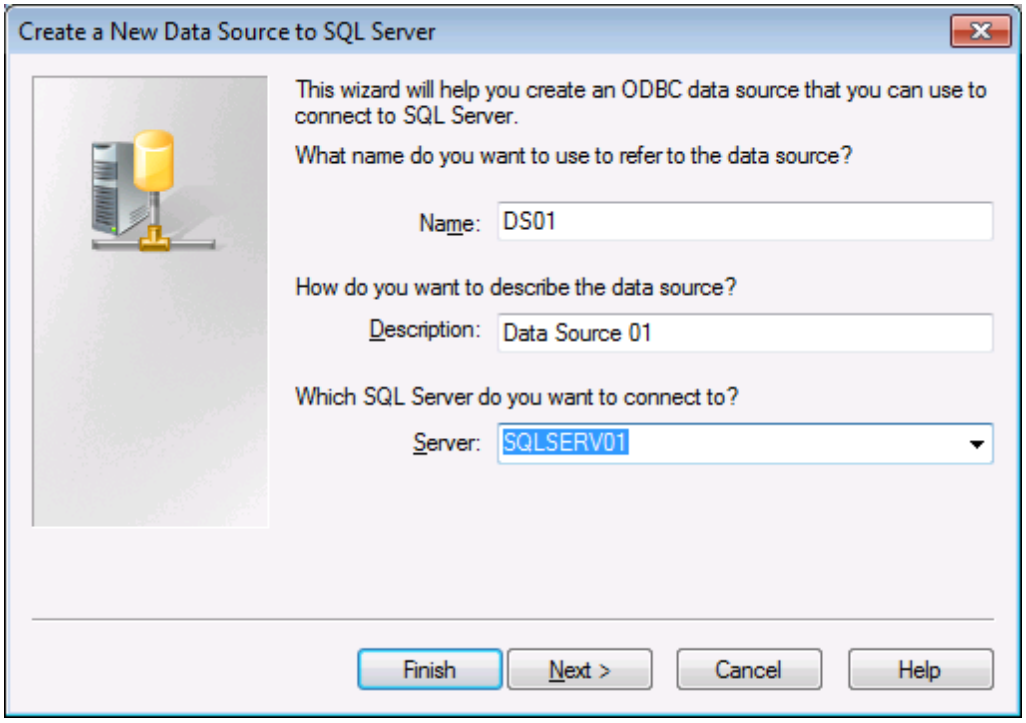
Connecting to Microsoft SQL Server (ODBC)

To connect to SQL Server using ODBC:

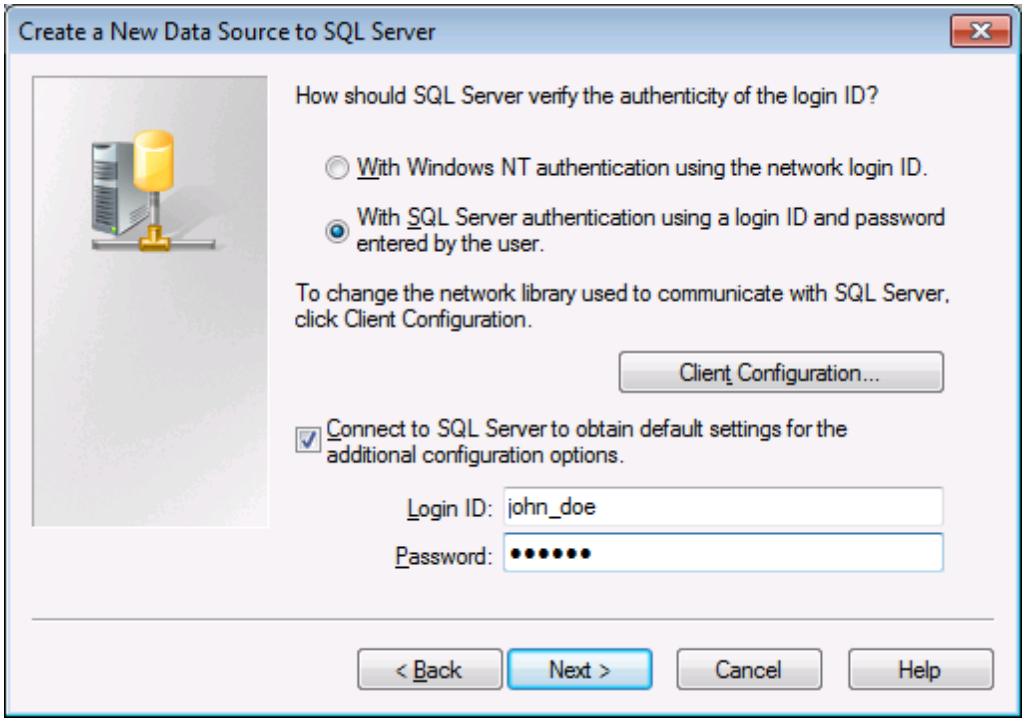
1. Start the database connection wizard.
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .



4. Select **SQL Server** (or **SQL Server Native Client**, if available), and then click **User DSN** (or **System DSN** if you are creating a System DSN).

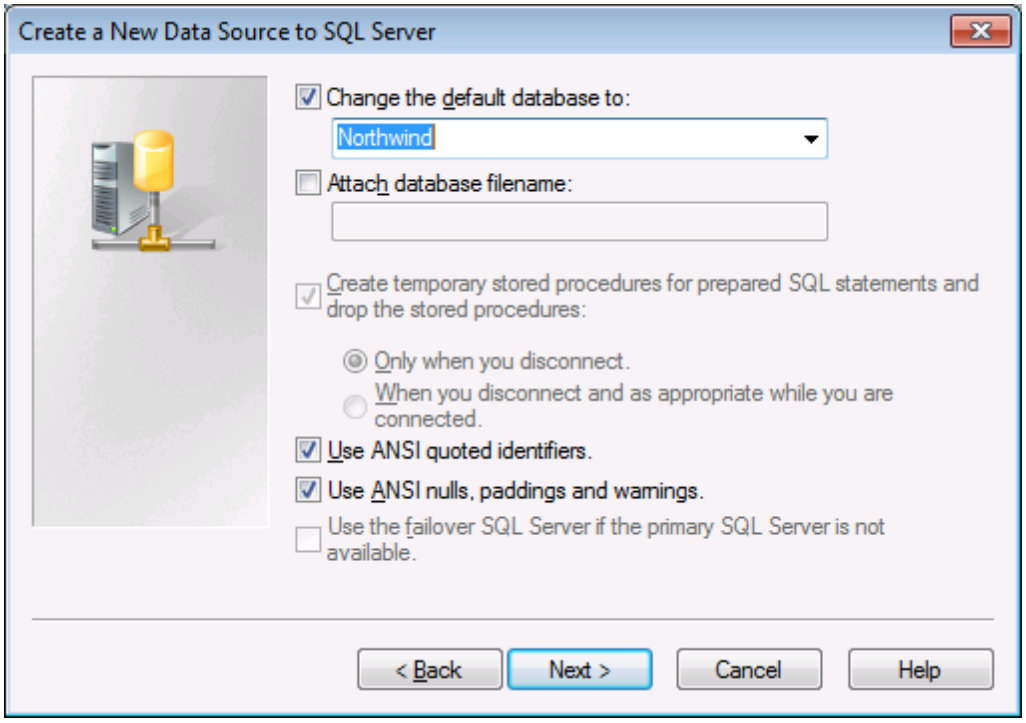


5. Enter a name and description to identify this connection, and then select from the list the SQL Server to which you are connecting (**SQLSERV01** in this example).



6. If the database server was configured to allow connections from users authenticated on

the Windows domain, select **With Windows NT authentication**. Otherwise, select **With SQL Server authentication...** and type the user name and password in the relevant boxes.



7. Select the name of the database to which you are connecting (in this example, **Northwind**).
8. Click **Finish**.

Connecting to MySQL (ODBC)

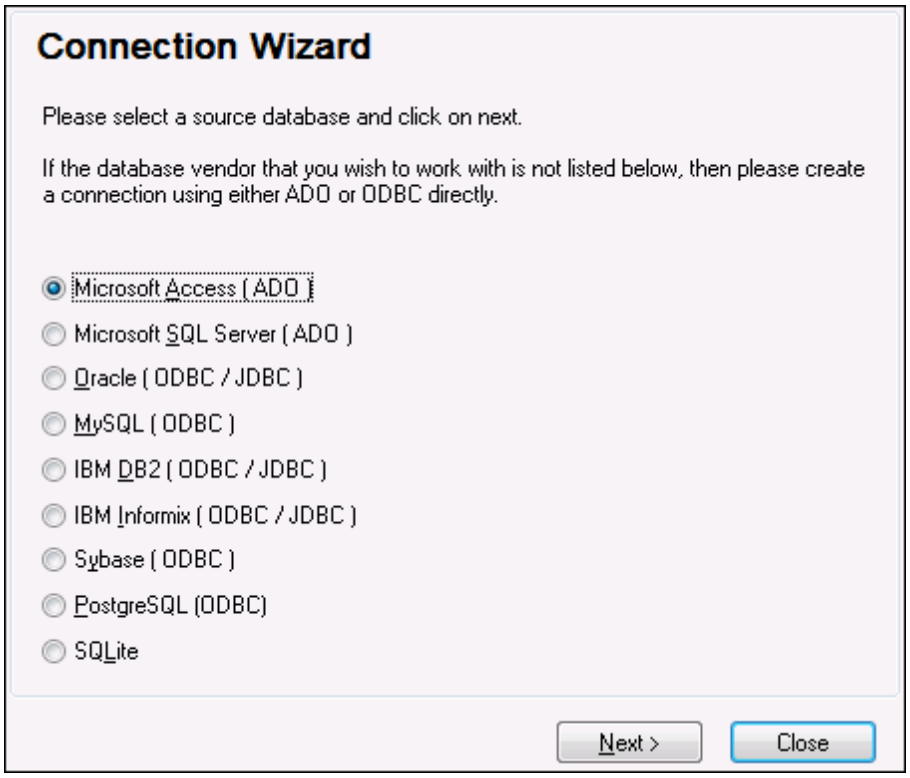
This topic provides sample instructions for connecting to a MySQL database server from a Windows machine through the ODBC driver. The MySQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses MySQL ODBC driver version 5.3.4 downloaded from the official website (see also Database Drivers Overview).

Prerequisites:

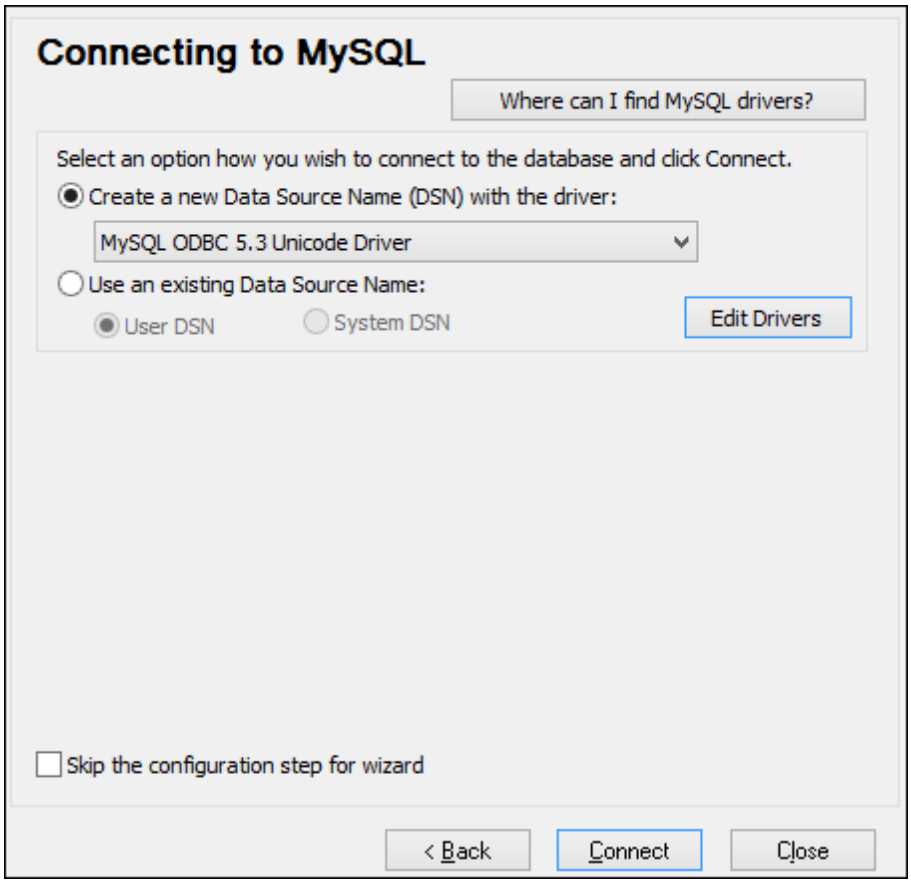
- MySQL ODBC driver must be installed on your operating system (for installation instructions, check the documentation supplied with the driver).
- You have the following database connection details: host, database, port, username, and password.

To connect to MySQL via ODBC:

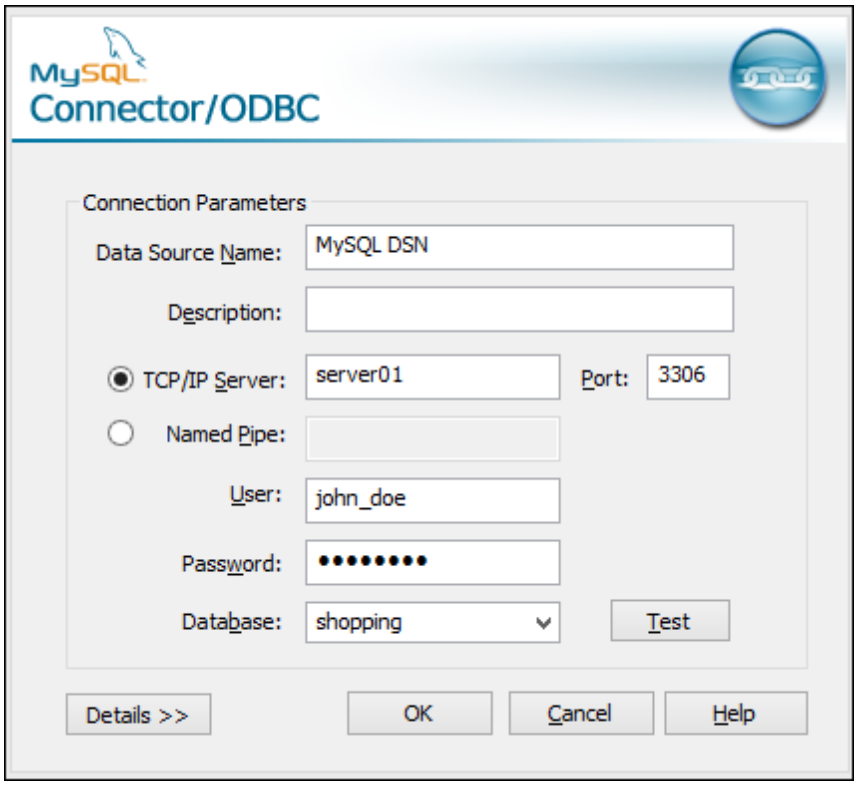
1. Start the database connection wizard.



2. Select **MySQL (ODBC)**, and then click **Next**.



3. Select **Create a new Data Source Name (DSN) with the driver**, and select a MySQL driver. If no MySQL driver is available in the list, click **Edit Drivers**, and select any available MySQL drivers (the list contains all ODBC drivers installed on your operating system).
4. Click **Connect**.



5. In the Data Source Name box, enter a descriptive name that will help you identify this ODBC data source in future.
6. Fill in the database connection credentials (TCP/IP Server, User, Password), select a database, and then click **OK**.

Note: If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address. Also, if you click **Details>>**, there are several additional parameters available for configuration. Check the driver's documentation before changing their default values.

Connecting to Oracle (ODBC)

This example illustrates a common scenario where you connect from MapForce to an Oracle database server on a network machine, through an Oracle database client installed on the local operating system.

The example includes instructions for setting up an ODBC data source (DSN) using the database connection wizard in MapForce. If you have already created a DSN, or if you prefer to create it directly from ODBC Data Source administrator in Windows, you can do so, and then select it when prompted by the wizard. For more information about ODBC data sources, see Setting up an ODBC Connection.

Prerequisites:

- The Oracle database client (which includes the ODBC Oracle driver) must be installed and configured on your operating system. For instructions on how to install and configure

an Oracle database client, refer to the documentation supplied with your Oracle software.

- The **tnsnames.ora** file located in Oracle home directory contains an entry that describes the database connection parameters, in a format similar to this:

```
ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = server01)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = orcl)
      (SERVER = DEDICATED)
    )
  )
```

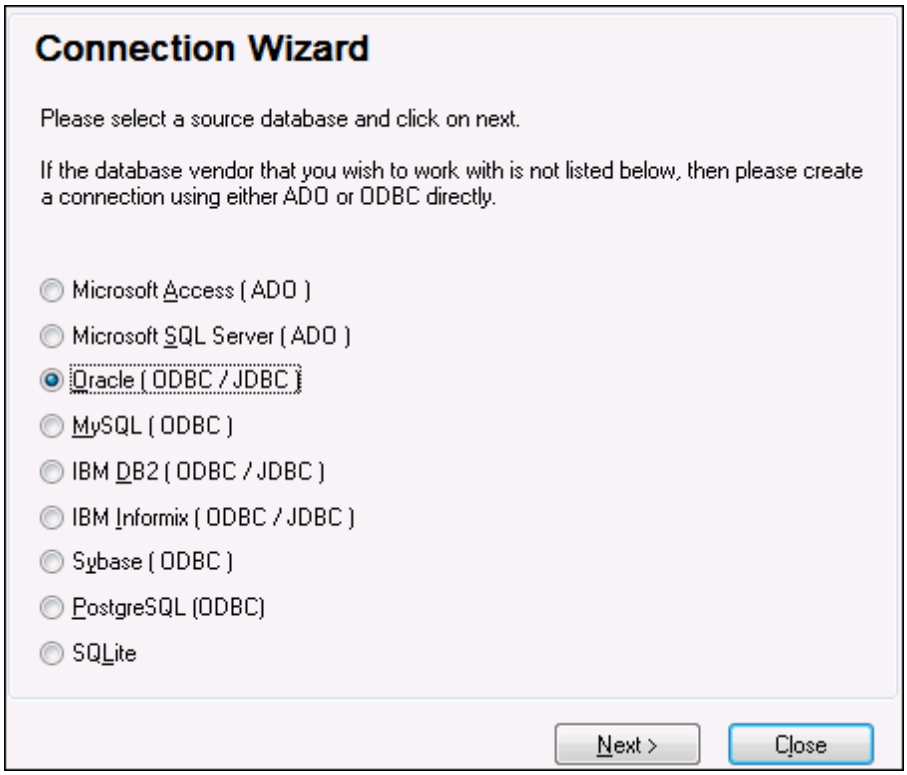
The path to the **tnsnames.ora** file depends on the location where Oracle home directory was installed. For Oracle database client 11.2.0, the default Oracle home directory path could be as follows:

```
C:\app\username\product\11.2.0\client_1\network\admin\tnsnames.ora
```

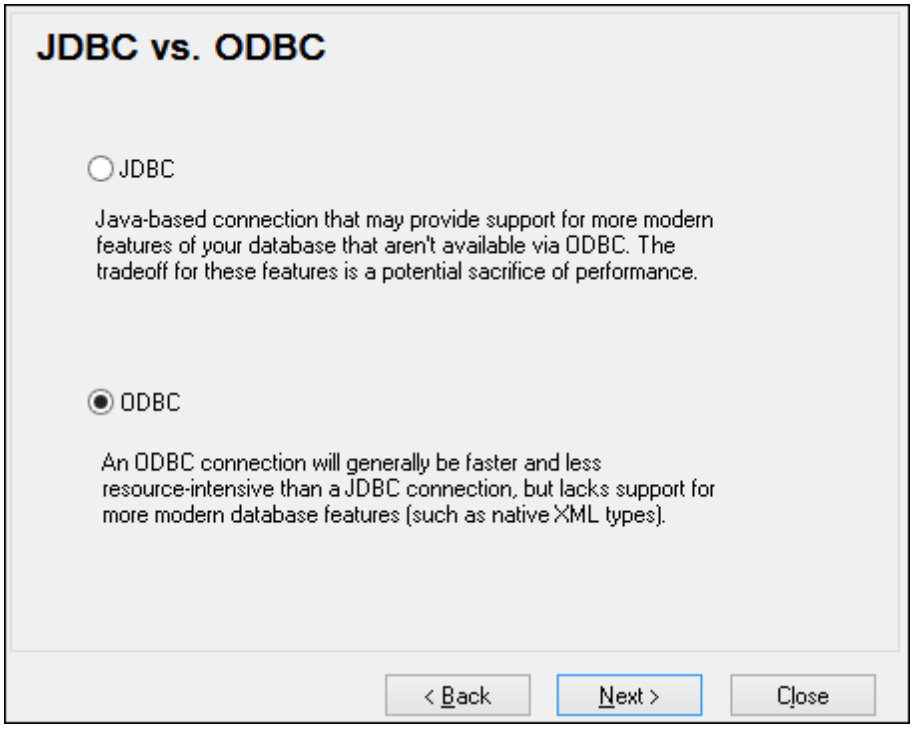
You can add new entries to the **tnsnames.ora** file either by pasting the connection details and saving the file, or by running the Oracle *Net Configuration Assistant* wizard (if available).

To connect to Oracle using ODBC:

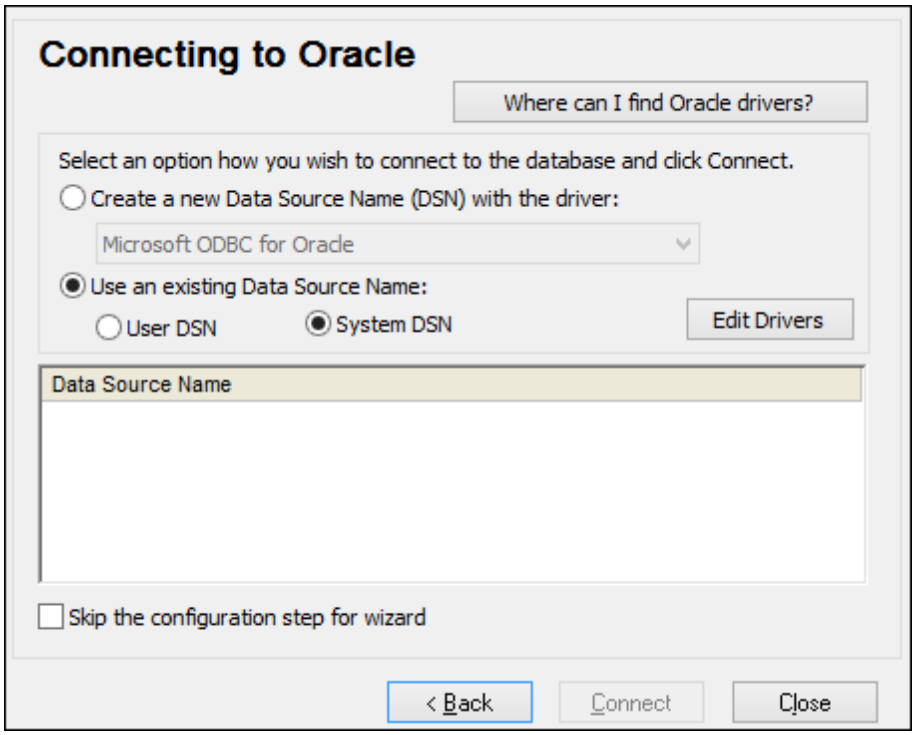
1. Start the database connection wizard.



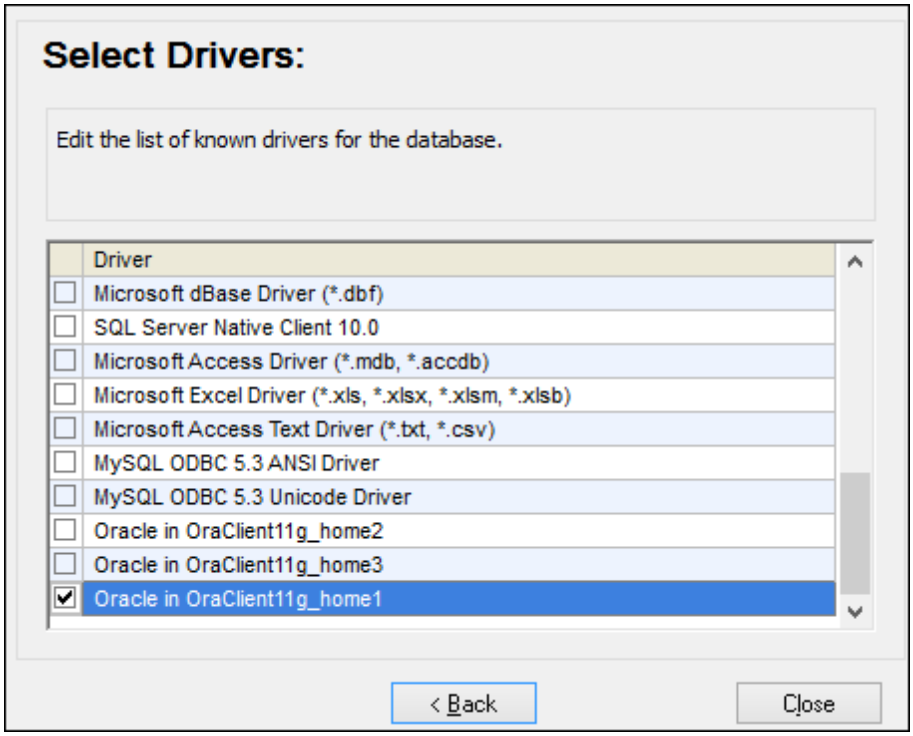
2. Select **Oracle (ODBC / JDBC)**, and then click **Next**.



3. Select **ODBC**.

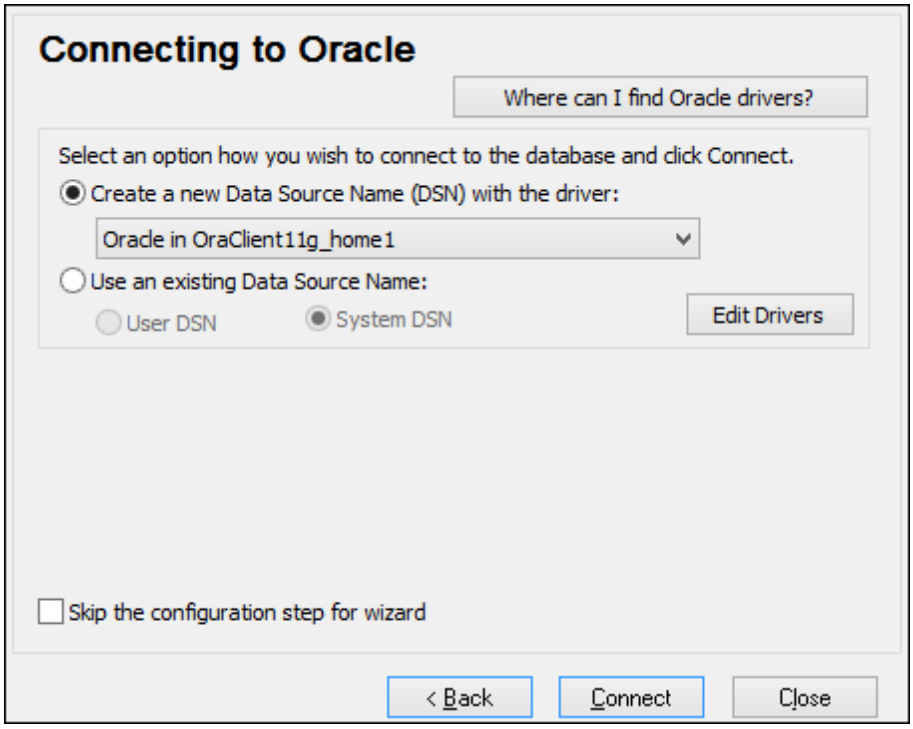


4. Click **Edit Drivers**.



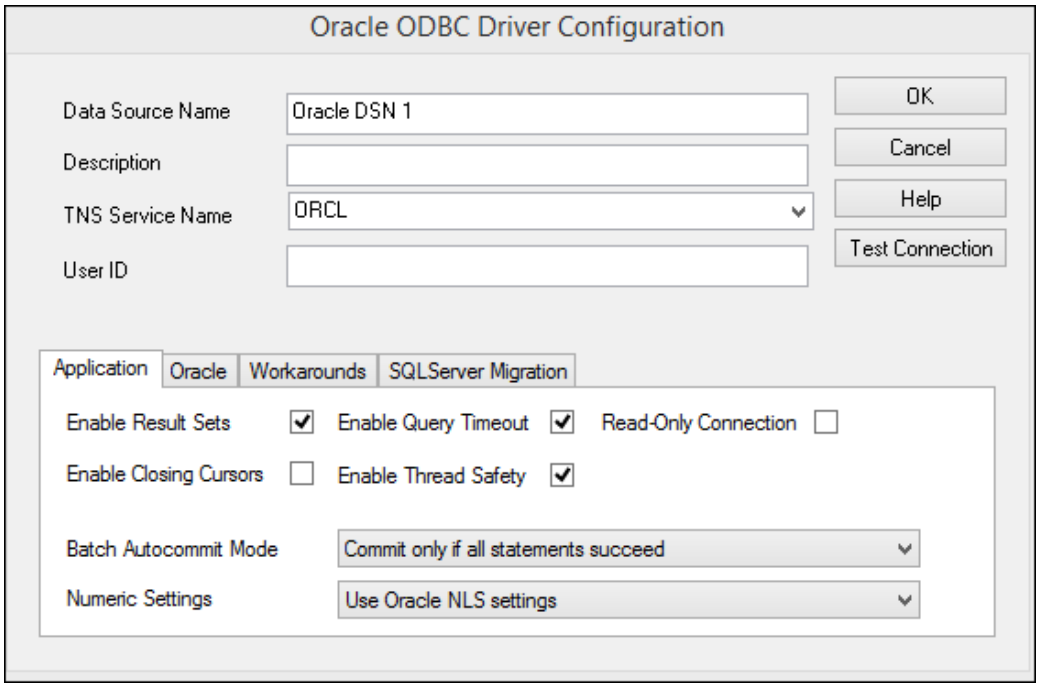
5. Select the Oracle drivers you wish to use (in this example, **Oracle in OraClient11g_home1**). The list displays the Oracle drivers available on your system after installation of Oracle client.

6. Click **Back**.
7. Select **Create a new data source name (DSN) with the driver**, and then select the Oracle driver chosen in step 4.

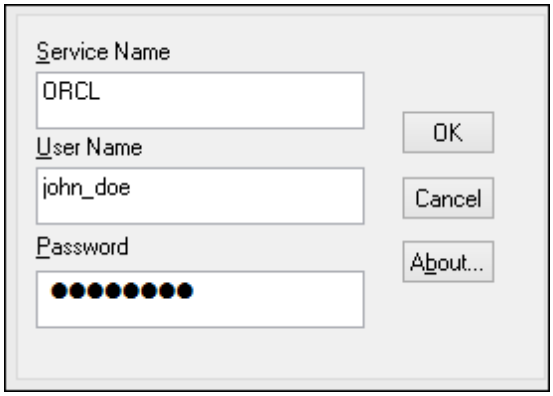


> Avoid using the Microsoft-supplied driver called **Microsoft ODBC for Oracle** driver. Microsoft recommends using the ODBC driver provided by Oracle (see http://msdn.microsoft.com/en-us/library/ms714756%28v=vs.85%29.aspx)

8. Click **Connect**.



9. In the Data Source Name text box, enter a name to identify the data source (in this example, **Oracle DSN 1**).
10. In the TNS Service Name box, enter the connection name as it is defined in the **tnsnames.ora** file (see prerequisites). In this example, the connection name is **ORCL**.
11. Click **OK**.



12. Enter the username and password to the database, and then click OK.

Connecting to PostgreSQL (ODBC)

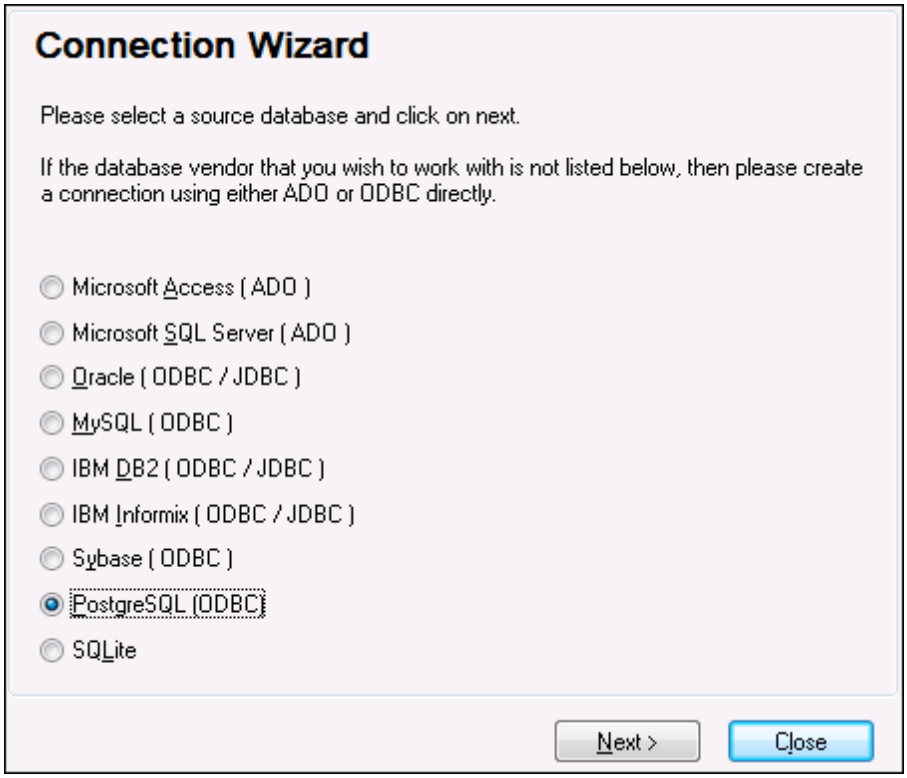
This topic provides sample instructions for connecting to a PostgreSQL database server from a Windows machine through the ODBC driver. The PostgreSQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses the psqlODBC driver (version 09_03_300-1) downloaded from the official website (see also Database Drivers Overview).

Prerequisites:

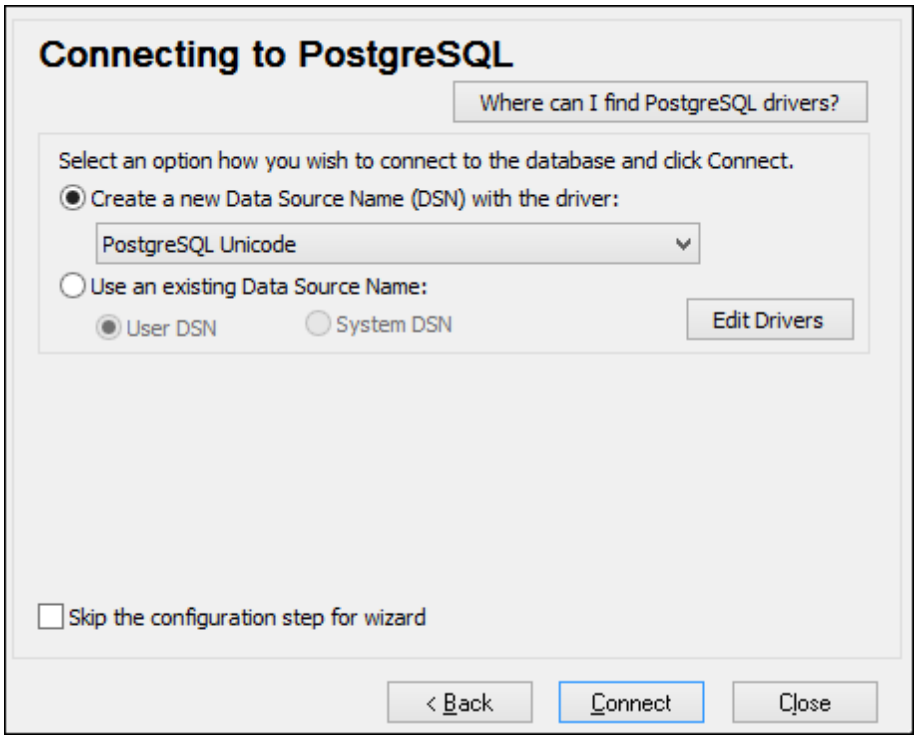
- *psqlODBC* driver must be installed on your operating system (for installation instructions, check the documentation supplied with the driver).
- You have the following database connection details: server, port, database, user name, and password.

To connect to PostgreSQL using ODBC:

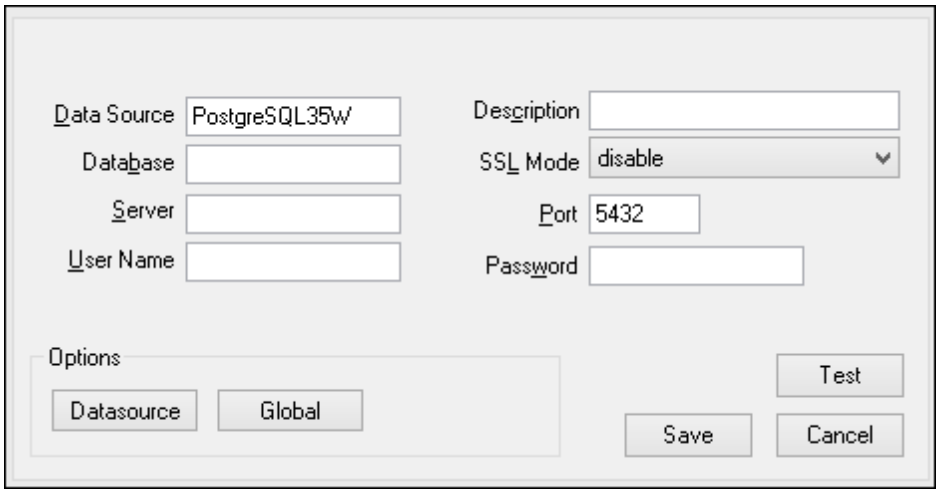
1. Start the database connection wizard.



2. Select **PostgreSQL (ODBC)**, and then click **Next**.



3. Select **Create a new Data Source Name (DSN) with the driver**, and select the PostgreSQL driver. If no PostgreSQL driver is available in the list, click **Edit Drivers**, and select any available PostgreSQL drivers (the list contains all ODBC drivers installed on your operating system).
4. Click **Connect**.



5. Fill in the database connection credentials (Database, Server, Port, User Name, Password), and then click **OK**.

Connecting to Sybase (JDBC)

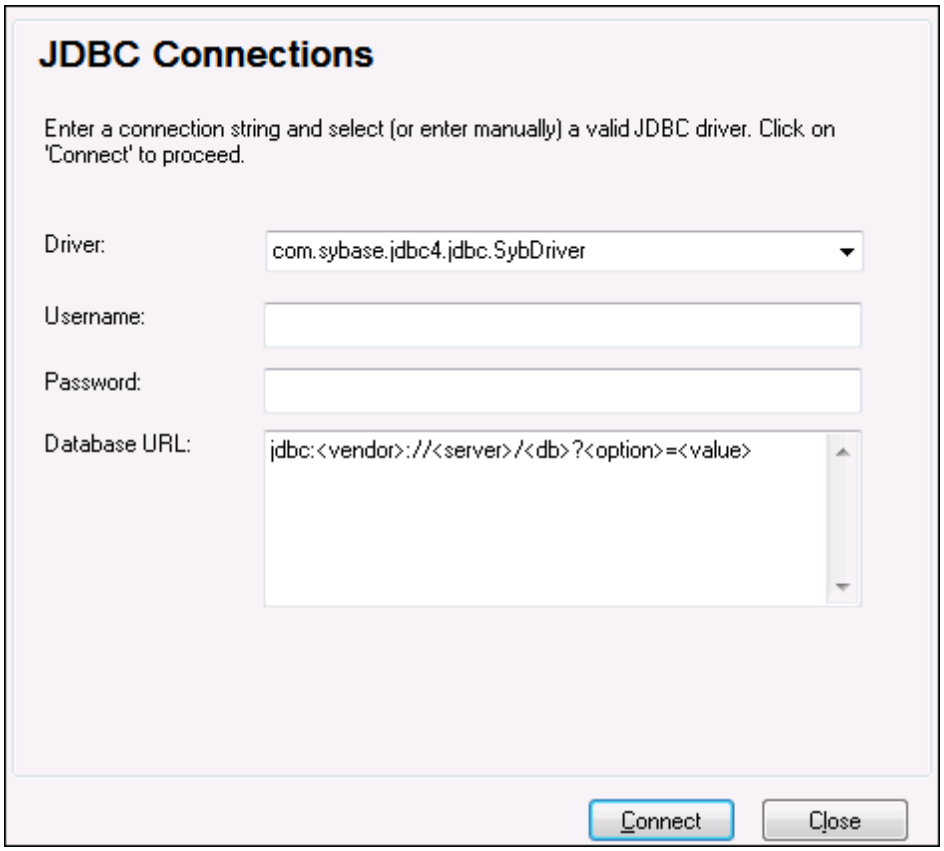
This topic provides sample instructions for connecting to a Sybase database server through JDBC.

Prerequisites:

- Java Runtime Environment (JRE) must be installed on your operating system.
- Sybase *jConnect* component must be installed on your operating system (in this example, *jConnect 7.0* is used, installed as part of the *Sybase Adaptive Server Enterprise PC Client* installation). For the installation instructions of the database client, refer to Sybase documentation.
- The operating system's CLASSPATH environment variable includes the path where the Sybase JDBC driver was installed. In this example, the JDBC driver is installed in the directory **C:\Sybase**, and the value of CLASSPATH variable was configured to include the path **C:\sybase\jConnect-7_0\classes\jconn4.jar**. For more information, see Configuring the CLASSPATH.
- You have the following database connection details: host, port, database name, username, and password.

To connect to Sybase through JDBC:

1. Start the database connection wizard.
2. Click **JDBC Connections**.
3. Select the Sybase JDBC driver from the list of available JDBC drivers (in this example, **com.sybase.jdbc4.jdbc.SybDriver**). If the list does not contain a Sybase driver, it is either not installed correctly, or not included in the CLASSPATH variable (see the list of prerequisites above).



4. Enter the username and password to the database in the corresponding text boxes.
5. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:sybase:Tds:hostName:port/databaseName
```

6. Click **Connect**.

7.2.2 Database Connections on Linux and Mac

If you have licensed any of the following Altova server products—MobileTogether Server, MapForce Server, or StyleVision Server, a common scenario is to design MobileTogether designs, MapForce mappings, or StyleVision transformations on a Windows desktop machine, and then deploy them to a server machine (either Windows, Linux, or Mac) to automate their execution.

In this documentation, the term "server execution files" is used to denote the following file types:

- MapForce Server execution files (.mfx)
- MobileTogether design files (.mtd)
- StyleVision transformations (.sps) packaged as Portable XML Forms (.pxf).

The following scenarios are possible when deploying server execution files:

1. **"Design and execute on Windows"**. In this scenario, you design the MobileTogether

designs, MapForce mappings, or StyleVision transformations on Windows, and then run their corresponding server execution files on a Windows system as well (which can either be the same Windows machine, or a remote Windows server).

2. **"Design on Windows, execute on Linux or Mac"**. In this scenario, you design all of the above files on Windows, and then deploy their corresponding server execution files to Linux and Mac for execution.

> In the **"Design and execute on Windows"** scenario, the selection of available database technologies and drivers comprises any of ADO, ODBC, JDBC, as well as SQLite connections (see Database Drivers Overview).
>
> In the **"Design on Windows, execute on Linux or Mac"** scenario, ADO and ODBC connections are not supported. In this scenario, you can use direct SQLite connections (see SQLite connections) and JDBC connections (see JDBC connections).

When you deploy server execution files to a server, databases are not included in the deployed package (this also applies to file-based databases such as SQLite and Microsoft Access), so a connection to them must be set up on the deployment server as well. In other words, the same database configuration must be in place both on the operating system where you design and on the server to which you deploy the files.

In general, the scenario in which you deploy server execution files to a different operating system is slightly more complex, since it requires that the same database configuration exist on both machines. To bypass complexity while designing locally and deploying remotely, consider using the Global Resources feature available in MapForce, MobileTogether Designer, and StyleVision.

For example, you can define two different Global Resource configurations to connect to the same database: one which would specify the connection settings using the Windows-style path conventions, and another one—using Linux-style path conventions. You could then use the first connection to test your files during the design phase, and the second connection to run the execution file on the Linux server.

7.2.2.1 SQLite connections on Linux and Mac

There is no need to separately install SQLite on Linux and Mac since support for it is integrated into Altova server products as well. Therefore, if your server execution files include calls to a SQLite database, you will be able to run them without having to install SQLite first. You need to ensure, however, that the server execution files use the correct path to the database file on the Linux or Mac machine. That is, before running the server execution files on the Linux or Mac server, make sure that the SQLite database file is referenced through a path which is POSIX (Portable Operating System Interface) compliant. This assumes that no Windows-style drive letters are used in the path, and directories are delimited by the forward slash character (/). For example, the path **/usr/local/mydatabase.db** is POSIX compliant, while the path **c:\sqlite \mydatabase.db** isn't.

7.2.2.2 JDBC connections on Linux and Mac

To set up a JDBC connection on Linux or Mac:

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 Mac OS, 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.

7.2.2.3 Oracle Connections on Mac OS X Yosemite

On Mac OS X Yosemite, you can connect to an Oracle database through the **Oracle Database Instant Client**. Note that, if you have a Mac OS with a Java version prior to Java 8, you can also connect through the **JDBC Thin for All Platforms** library, in which case you may disregard the instructions in this topic.

You can download the Oracle Instant Client from the Oracle official download page. Note that there are several Instant Client packages available on the Oracle download page. Make sure to select a package with Oracle Call Interface (OCI) support, (for example, Instant Client Basic). Also, make sure to select the 32-bit version if your operating system runs on 32-bit, and the 64-bit version if your operating system runs on 64-bit.

Once you have downloaded and unpacked the Oracle Instant Client, edit the property list (.plist) file shipped with the installer so that the following environment variables point to the location of the corresponding driver paths, for example:

Variable	Sample Value
`CLASSPATH`	`/opt/oracle/instantclient_11_2/ojdbc6.jar:/opt/oracle/instantclient_11_2/ojdbc5.jar`
`TNS_ADMIN`	`/opt/oracle/NETWORK_ADMIN`
`ORACLE_HOME`	`/opt/oracle/instantclient_11_2`
`DYLD_LIBRARY_PATH`	`/opt/oracle/instantclient_11_2`
`PATH`	`$PATH:/opt/oracle/instantclient_11_2`

Note: Edit the sample values above to fit the paths where Oracle Instant Client files are installed on your operating system.

7.2.3 Tutorial: Mapping XML data to databases

In this tutorial, you will learn how to:

- Set up the XML-to-database mapping
- Insert data into database tables
- Update
- Delete
- Ignore
- Generate database output values

Tutorial example files

The database tutorial makes use of the following files:

- Altova_Hierarchical.xsd The hierarchical schema file, containing identity constraints
- Altova-cmpy.xml The Altova company data file which supplies the XML data
- Altova.mdb The Altova MS-Access database file, which functions as the target database

All these example files are available in the ...**MapForceExamples\Tutorial** folder. **Please note**: This section makes heavy use of the **Altova.mdb** database, to show the database-as-target functionality of MapForce. Make sure you backup the file before you try any of the examples shown here. To produce the same results as shown in the tutorial examples, you should begin each section from scratch and without any previous updates in the database!

MapForce is able to map from password protected Access files, if they are added via the **Any ODBC** option in the "Select a source database" dialog box. The user and password settings can then be entered in the wizard.

7.2.3.1 Setting up the XML-to-database mapping

Setting up an XML to database mapping, is in no way different from the methods previously described. **Please note**: Creating mappings between database components is not possible if you select XSLT, XSLT2, or XQuery as the target language. XSLT does not support database queries.

Objective

In this section of the tutorial, you will learn how to add a database component and connect to the database it represents.

Commands used in this section

Built-in Execution Engine: This command is located in the Language Selection toolbar and in the **Output** menu. Click this command to select the built-in execution engine as the preferred output format.

Insert XML Schema/File: Click this icon to open the standard Windows **Open** dialog box and select the file from your file system.

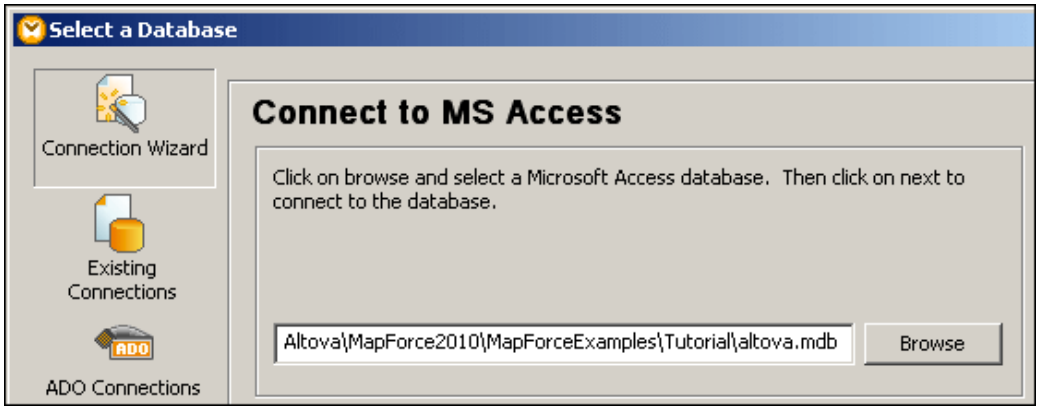
Insert database: This command is located in the Insert Component toolbar and in the **Insert** menu. Click this command to open the **Select a Database** dialog box where you

can define a database connection and select the database tables to be included in the MapForce mapping.

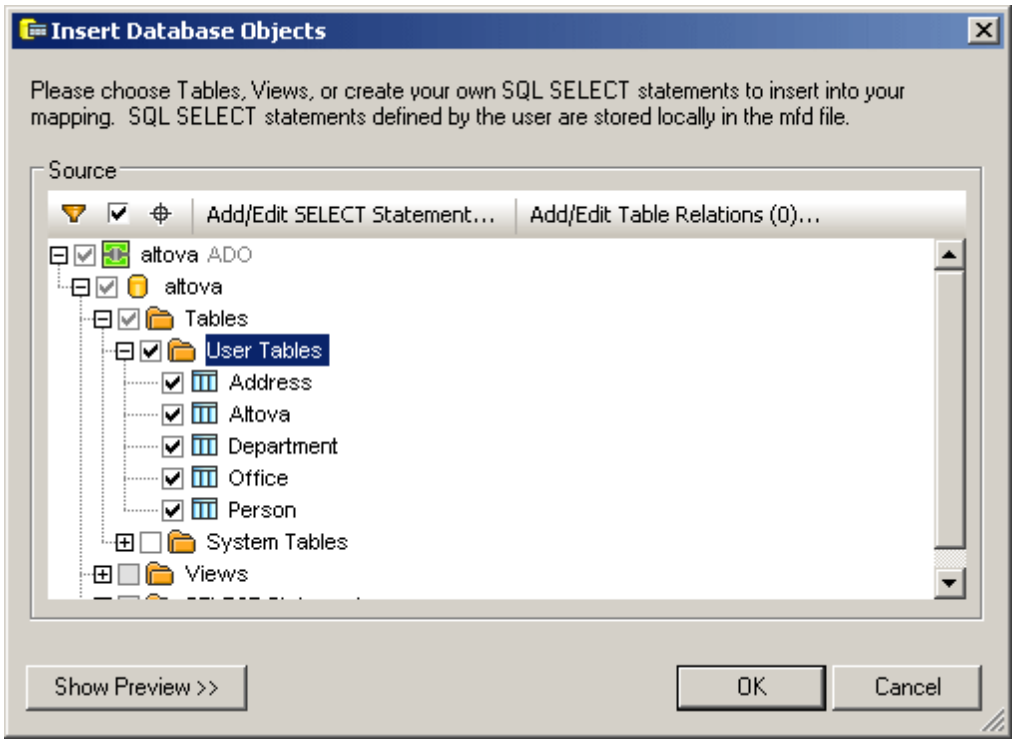
To set up the mapping environment and connect to an MS Access database:

1. Click a programming language icon (Java, C#, C++, or BUILTIN) in the icon bar to specify the language the generated code should support. This setting also loads the language related library into the Libraries window.
2. Click the **Insert Schema | XML Schema/File** icon, and select the **Altova_Hierarchical.xsd** from the MapForceExamples folder.
3. In the message box that pops up, click the **Browse**... button and select the **Altova-cmpy.xml** file as the XML instance file.
4. Click the **Altova** entry in the Altova_Hierarchical component of the mapping window, and hit the ***** key on the numeric keypad to view the items; resize the component if necessary.
5. Click the **Insert Database** icon and, in the **Select a Database** dialog box, select the **Connection Wizard** icon.
6. Select the **Microsoft Access (ADO)** entry and click **Next**.

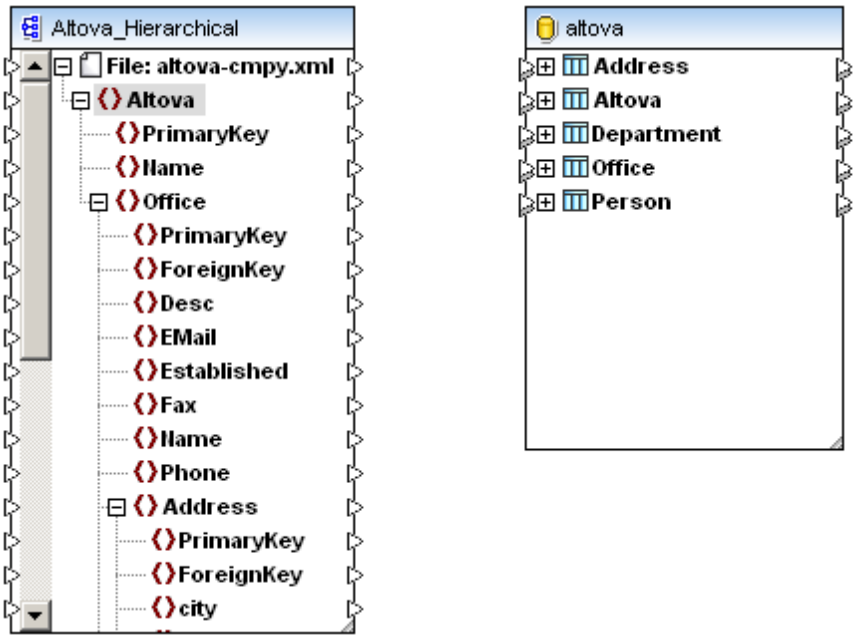
7. Click the **Browse** button to select the **altova.mdb** database available from the ...\MapForceExamples\Tutorial\ folder, and click **Connect**.



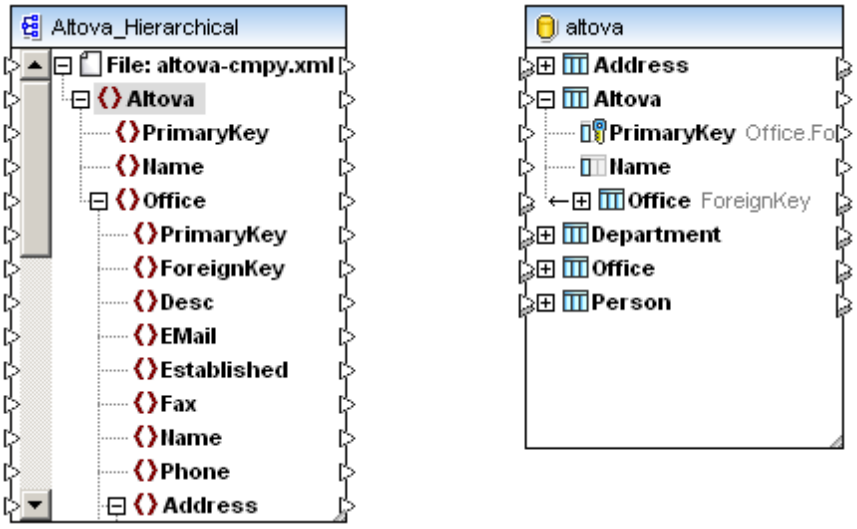
The Insert Database Objects dialog box that pops up allows you to define the specific Tables, Views or System tables that you want to appear in the Database component. You can preview the selected table by clicking the **Show Preview** button in the Preview group box below.



8. Click the check box to the left of User Tables to select them all, and click **OK** to insert the database.



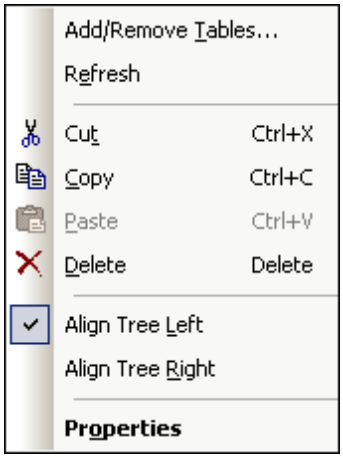
9. Click the + expand icon of the **Altova** item, to display the Altova table fields.



Changing the database settings

Database settings can be changed by right clicking the database and selecting:

- **Add/Remove tables**, allows you to add or delete tables to/from the current component.
- **Properties**, allows you to change the component database by clicking the Change button, and using the wizard to select a different database.



7.2.3.2 Inserting databases - table preview customization

The **Insert Database Objects** dialog box contains an icon bar which allows you customize, or find specific items in the Source group box.

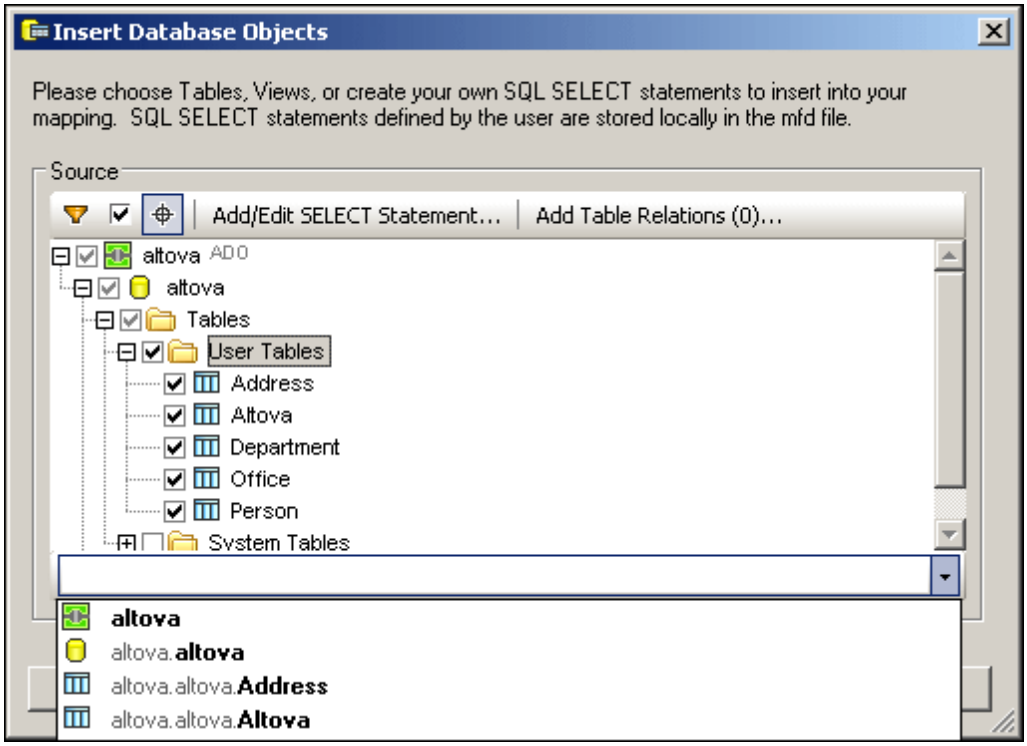
The **Object Locator** allows you to find specific database items.

The **Filter** allows you to restrict tables by existing characters.

The **Show checked objects only** icon displays those items where a check box is active.

Finding database elements using the Object Locator:

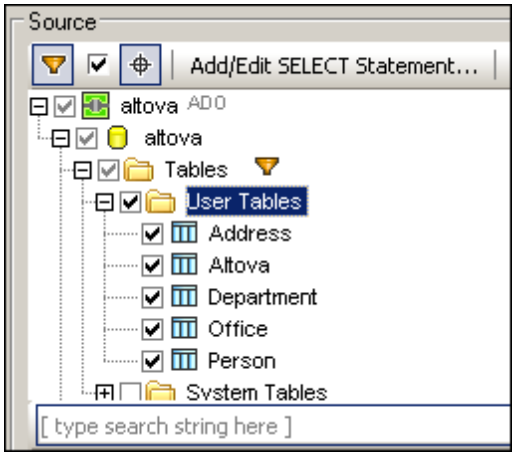
1. Click the **Object Locator**  icon or press **Ctrl+L** to search for specific database items. A drop-down list containing all selectable items appears at the bottom of the window.



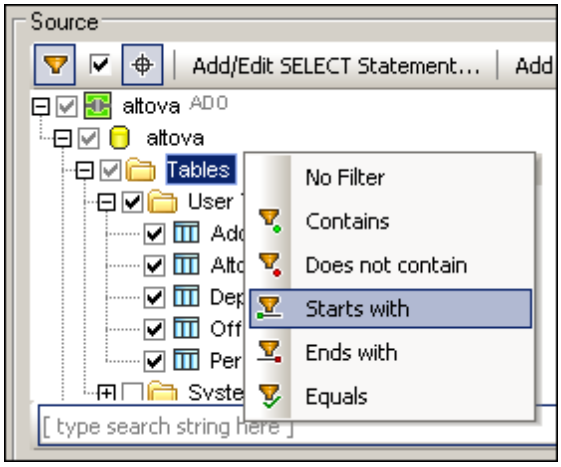
2. Enter the string you want to search for, or select the item from the drop-down list.

Filtering objects in the preview window:
Schemas, tables, and views can be filtered by name or part of a name. Filtering is case-insensitive.

1. Click the **Filter Folder contents** icon in the toolbar to activate filtering. Filter icons appear next to folders.

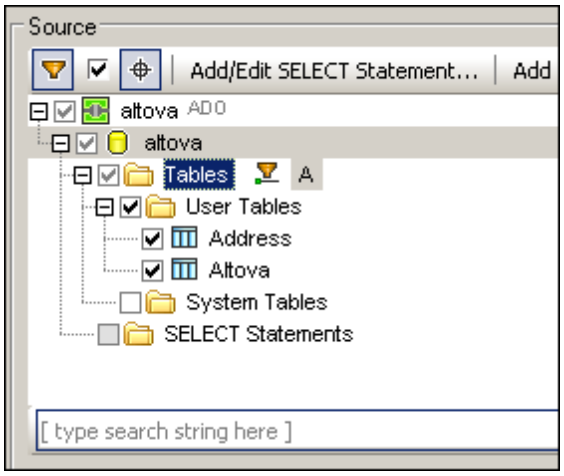


2. Click the filter icon next to the folder, and database objects, you want to filter. Select the filtering option from the popup menu that appears.



An empty field appears next to the filter icon.

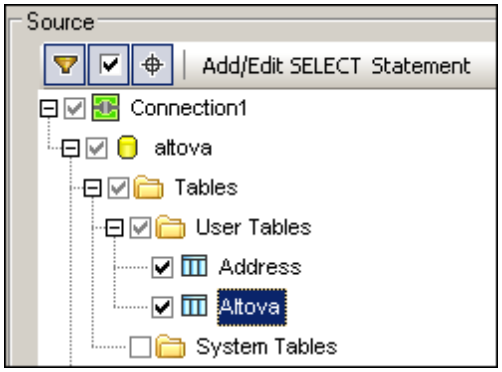
3. Enter the characters you want to act as the filter e.g. A.



The results are automatically updated as you type.

Showing checked objects only

1. Click the **Show checked objects only** icon in the toolbar to have only those tables displayed, where the check mark is present.



7.2.3.3 Components and table relationships

Table relationships are easily recognized in the database component. The database component displays **each table** of a database, as a "**root**" table with all other related tables beneath it in a tree view.

Let us call the table names visible in the above diagram "**root**" tables, i.e. they are the top level, or **root** of the tree view. Expanding a table name displays all the tables related to it. The "root" tables are usually displayed in alphabetical sort order; this has no bearing on the actual table relationships however.

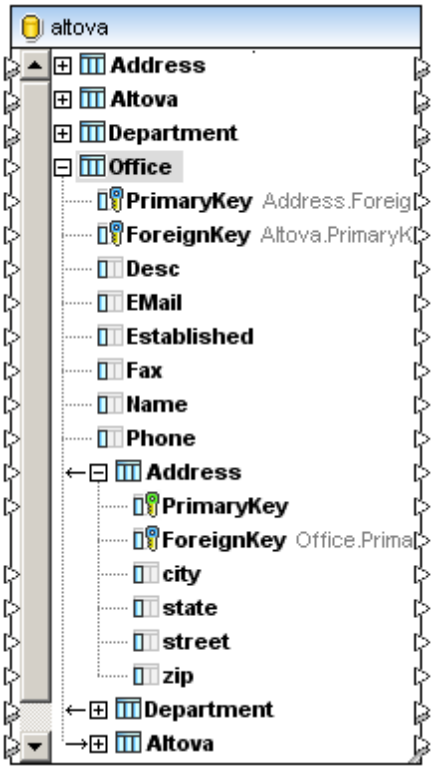
When creating queries/mappings of databases with relations, including flat format SQL/XML databases, make sure that you create mappings **between tables** that appear under **one of the "root" tables**, if you want the table relationships to be maintained i.e. when creating queries that make use of joins.

The graphic below, shows the expanded **Office** "root" table of the Altova database. The arrows to the left of the expand/contract icons of each table name, as well as the indentation lines, show the table relationships.

Starting from the **Office** table and going down the tree view:

- **Arrow left,** denotes a child table of the table above, **Address** is a child table of Office. Department is also a child of Office, as well as a "sibling" table of Address, both have the same indentation line.
- Person is also a child table of Department.

Arrow right, denotes a parent of the table above, **Altova** is the parent of the Office table.



Which "root" tables should I use when I am mapping data?
When creating mappings to database tables, make sure you create mappings using the **specific** "root" table as the top level table.

E.g.
suppose you only want to insert or update Person table data. You should then create mappings using the Person table as the "root" table, and create mappings between the source and target items of the Person fields you want to update.

If you want to update Department and Person data, while retaining database relationships between them, use the Department table as the "root" table, and create mappings between the source and target items of both tables.

7.2.3.4 Database action: Insert

This section of the tutorial deals with inserting data into a database. The source data from an XML file and its corresponding XML Schema will be used to generate SQL scripts that perform the relevant database actions.

Objective

In this section of the tutorial, you will learn how to use MapForce to insert data into a database. Specifically, you will learn how to:

- Add a new office orgchart to the Altova table in your example database
- Insert related office tables to the new orgchart record.

Commands used in this section

Insert XML Schema/File: Click this icon to open the standard Windows **Open** dialog box and select the file from your file system.

Insert database: This command is located in the Insert Component toolbar and in the **Insert** menu. Click this command to open the **Select a Database** dialog box where you can define a database connection and select the database tables to be included in the MapForce mapping.

Database Table Actions: This command is located in the **Component** menu of a database component and appears as a button to the right of the table name in the database component. Click this command to open the **Database Table Actions** dialog box where you can define the actions to be executed for each database record.

Run SQL Script: This command is located in the Output Preview toolbar and in the **Output** menu. Click this command to execute the SQL script that has been generated in the Output pane.

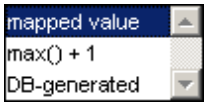
Regenerate Output: This command is located in the Output Preview toolbar and in the **Output** menu. Click this command to reset the SQL script so that it can be run for a second time.

Inserting data into a database table

The first example in this section, deals with the simple task of **adding** a new office orgchart to the Altova table. For each record, a new primary key will be generated.

Primary key settings

The primary key settings for the Insert action, are set using the combo boxes to the right of each field.



You can choose from among the following options:

- **mapped value**: allows source data to be mapped to the database field directly, and is the standard setting for all database fields. It is also possible to use a stored procedure to supply a key value by defining a relation, see Using stored procedures to generate primary keys.
- **Max() + 1**: Generates the key values based on the existing keys in the database, so that new records are automatically appended to existing ones.
- **DB-generated**: The database uses the **Identity function** to generate key values.

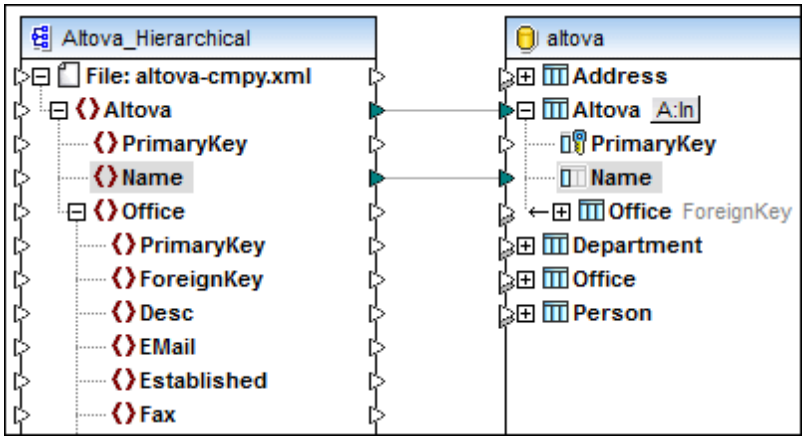
To insert values from an XML file into a database table:

1. Open a new mapping window and insert **Altova_Hierarchical.xsd**, select the **Altova-cmpy.xml** file as the XML instance file, and insert the **altova.mdb** database as described in the Setting up the XML to database mapping section.
2. Create the following mappings:

   Altova to Altova
   Name to Name

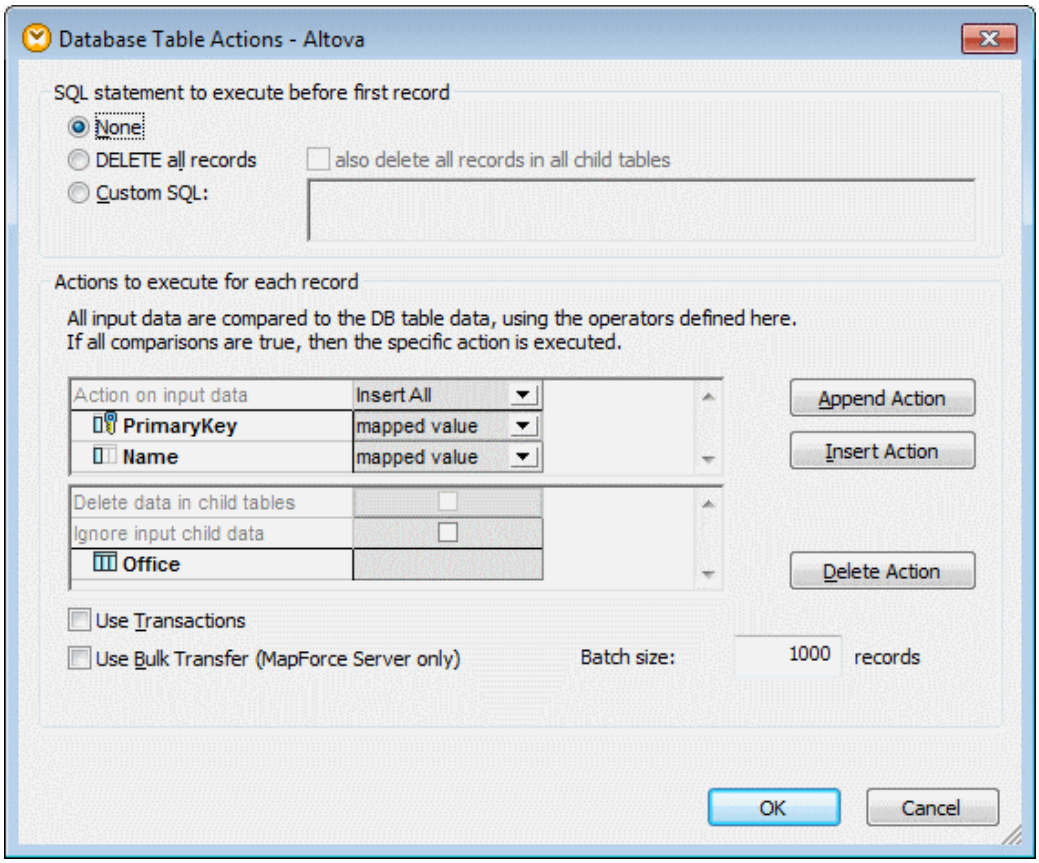
   Please note: If all Altova, Office etc. items are automatically mapped, the option "Auto-

connect children" is active. Select the menu option **Edit | Undo**, and then **Connection | Auto-connect matching children,** to disable this option.



The icon to the right of the Altova table (**A:In** in the screen shot above) displays the currently defined **table action** defined for that table, i.e. Action:Insert. Clicking the icon opens the **Database Table Actions** dialog box where the various table actions can be defined.

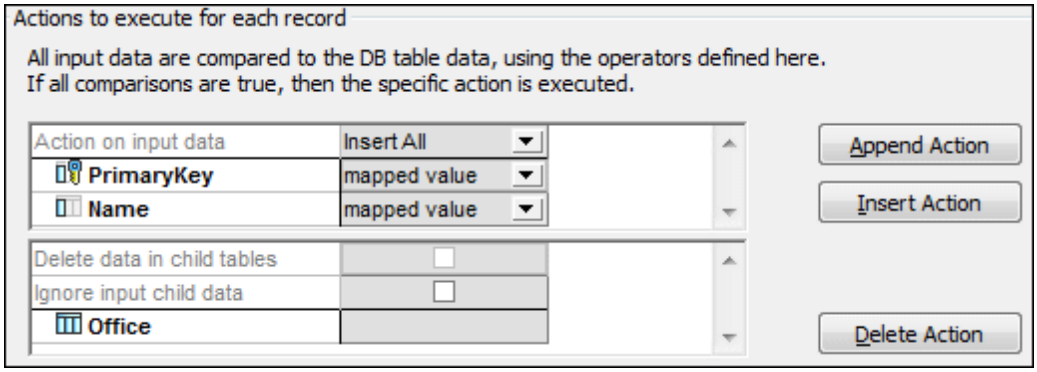
3. Click the **Table Actions** A:In icon next to the **Altova** table entry.
There is currently only one table action column defined in this dialog box, **Insert All**.



The table action, Insert All, will insert records with all **mapped fields** of the current table into the database. We now have to define how to generate the new PrimaryKey field for

this action.

4. Click the combo box to the right of the **PrimaryKey** field and select **max() + 1**.



4. Click **OK** to confirm.
6. Click the Output tab at the bottom of the mapping window to see the SQL script that this mapping produces.

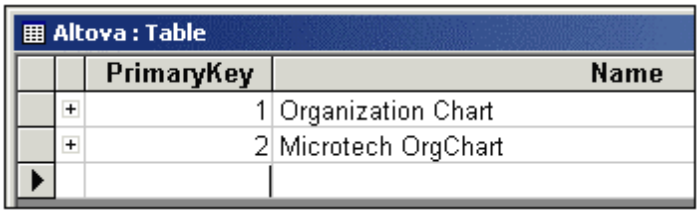
```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Documents and Settings\...\My Documents\
Altova\MapForce2012\MapForceExamples\altova.mdb;
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey]
FROM [Altova]
-->>> %PrimaryKey1%

INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Microtech OrgChart', '%PrimaryKey1%')
```

The script executes the mapping to the target database, taking the defined table actions into account. You can rerun SQL scripts from the Output pane by clicking the **Regenerate Output** icon.

7. Click the **Run SQL-Script** icon in the function bar to run the script and insert the table data into the database. If the script was successful, a confirmation message appears in the Message window. Click **OK** to confirm.
8. Open the Altova database in DatabaseSpy or Access to see the effect.



A new Microtech OrgChart record has been added to the Altova table with the new PrimaryKey 2. The data for this record originated in the input XML instance.

9. Switch back to MapForce.

In the Message window, you will now see a record of what happened when the SQL script was processed.

```
SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey]
FROM [Altova]
-->>> OK. One or more rows.

INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Microtech OrgChart', 2)
-->>> OK. 1 row(s).
```

Note:
To speed up inserting large amounts of rows into a database, you can activate the Bulk insert option.

Inserting tables and related child tables

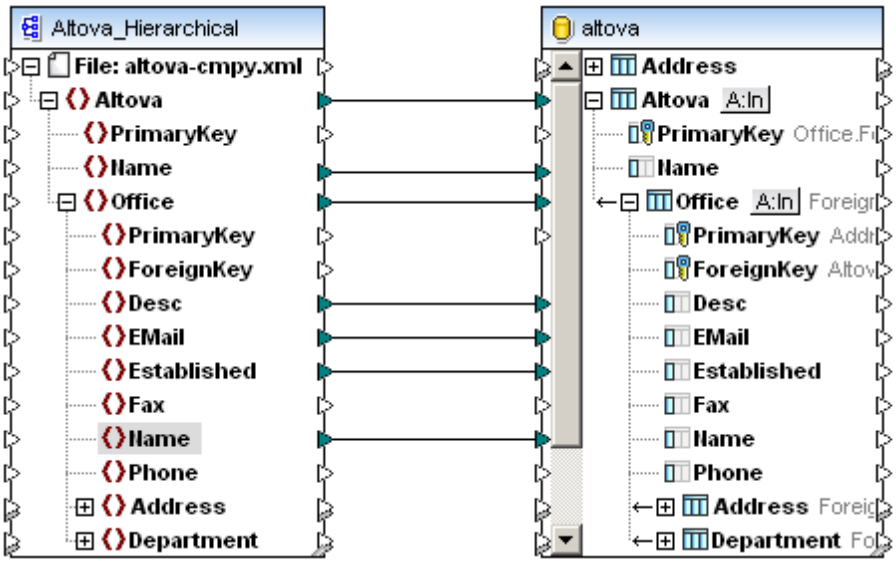
This example uses the previous example as a basis, and extends it by inserting related Office child tables to the Altova parent table.

Table relationships are only generated automatically, when mappings are created **between** child tables of a "root" table. In this case, mappings are created between the Office fields that appear directly under the Altova parent (or "root") table.

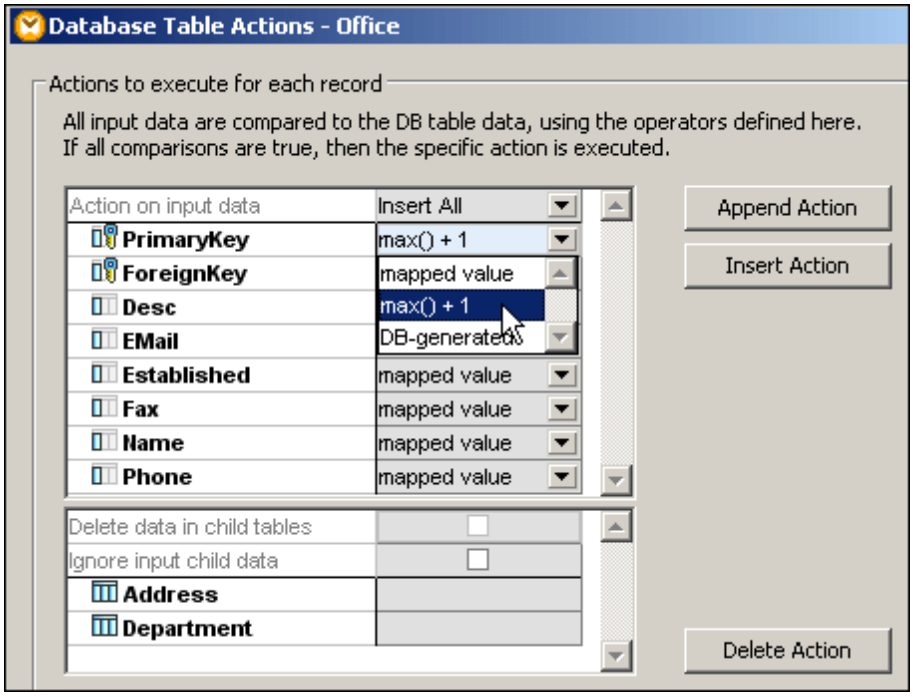
To insert tables and related child tables:

1. Open a new mapping window and insert **Altova_Hierarchical.xsd**, select the **Altova-cmpy.xml** file as the XML instance file, and insert the **altova.mdb** database as described in the Setup pf XML to database mapping section.
2. Create mappings between the Altova and Name items as described in the previous section.
3. Click the + expand icon of the **Office** item in both components, to display the Office table fields.
4. Create the following mappings between the two components:

Office	to	Office
Desc	to	Desc
Email	to	Email
Established	to	Established
Name	to	Name



5. Click the **Table Actions** A:In icon next to the **Office** table entry.
   The **Insert All..**. table action is selected by default, you do not have to make any changes here.
6. Click the **PrimaryKey** combo box and select the **max()+1** entry, then click **OK** to confirm.



7. Click the Output button to see the SQL script.

```
/*
The following SQL statements are only for preview and may not be executed in another SQL query tool!
To execute these statements use function "Run SQL-script" from menu "Output".
Connect to database using the following connection-string:
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Documents and Settings\...\My Documents\Altova\
MapForce2012\MapForceExamples\altova.mdb;
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM
 [Altova]
-->>> %PrimaryKey1%

INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Microtech OrgChart', '%PrimaryKey1%')

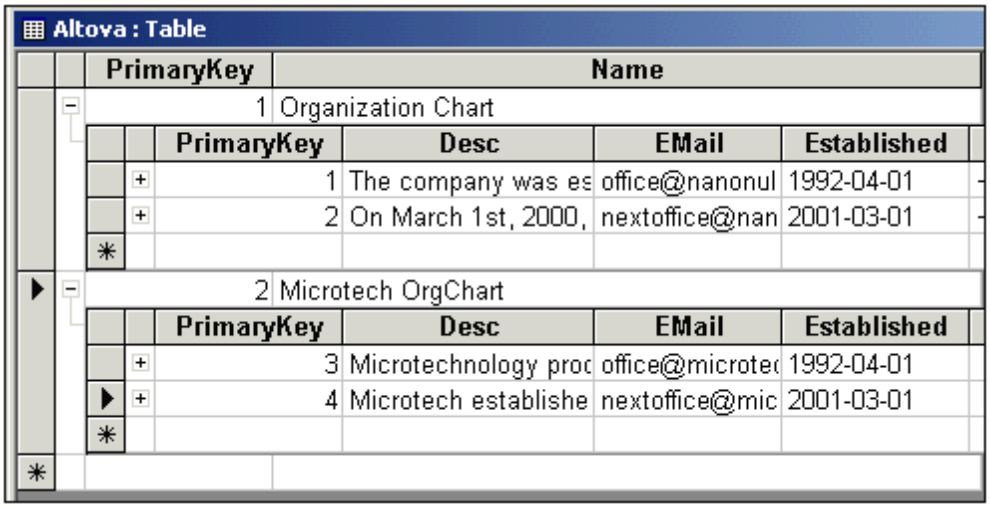
SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM
[Office]
-->>> %PrimaryKey2%

INSERT INTO [Office] ([ForeignKey], [Desc], [EMail], [Established], [Name], [PrimaryKey]) VALUES (
'%PrimaryKey1%', 'Microtechnology products are currently the bleeding edge of computer technology.',
'office@microtech.com', '1992-04-01', 'Microtech, Inc.', '%PrimaryKey2%')

SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM
[Office]
-->>> %PrimaryKey3%

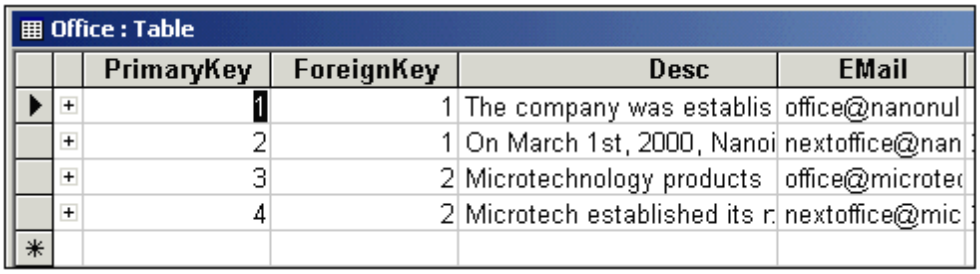
INSERT INTO [Office] ([ForeignKey], [Desc], [EMail], [Established], [Name], [PrimaryKey]) VALUES (
'%PrimaryKey1%', 'Microtech established its new office on Feb 30', 'nextoffice@microtech.com',
'2001-03-01', 'Microtech Partners, Inc.', '%PrimaryKey3%')
```

8. Click the Run SQL script icon  to run the script and insert the new tables.
9. Open the database in Ms Access and double-click the **Altova** table to see the effect.



Two new offices have been added to the Microtech OrgChart.

10. Double click the **Office** table to see the effect in greater detail.



Office : Table

	PrimaryKey	ForeignKey	Desc	EMail
▶ +	1	1	The company was establis	office@nanonul
+	2	1	On March 1st, 2000, Nanoi	nextoffice@nan
+	3	2	Microtechnology products	office@microte
+	4	2	Microtech established its r	nextoffice@mic
*				

The new offices have been added with primary keys of 3 and 4 respectively. Both these new offices are related to the Altova table by their foreign key 2, which references the Microtech OrgChart record.

```
SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM [Altova]
-->>> OK. One or more rows.

INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Microtech OrgChart', 2)
-->>> OK. 1 row(s).

SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM [Office]
-->>> OK. One or more rows.

INSERT INTO [Office] ([ForeignKey], [Desc], [EMail], [Established], [Name], [PrimaryKey]) VALUES (2,
'Microtechnology products are currently the bleeding edge of computer technology.', 'office@microtech.com',
'1992-04-01', 'Microtech, Inc.', 3)
-->>> OK. 1 row(s).

SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM [Office]
-->>> OK. One or more rows.

INSERT INTO [Office] ([ForeignKey], [Desc], [EMail], [Established], [Name], [PrimaryKey]) VALUES (2, 'Microtech
established its new office on Feb 30', 'nextoffice@microtech.com', '2001-03-01', 'Microtech Partners, Inc.', 4)
-->>> OK. 1 row(s).
```

7.2.3.5 Database action: Update

This section deals with updating databases and the various database actions MapForce provides for this purpose.

Objective

In this section of the tutorial, you will learn how to use MapForce to update data in a database table. Specifically, you will learn how to do the following:

- Update the fields of a specific table
- Add new records to a table
- Use the Update if... condition

Commands used in this section

Insert XML Schema/File: Click this icon to open the standard Windows **Open** dialog box and select the file from your file system.

Insert database: This command is located in the Insert Component toolbar and in the **Insert** menu. Click this command to open the **Select a Database** dialog box where you can define a database connection and select the database tables to be included in the

MapForce mapping.

Auto Connect Matching Children: Click this icon to toggle the automatic connection of matching child nodes, on and off.

Database Table Actions: This command is located in the **Component** menu of a database component and appears as a button to the right of the table name in the database component. Click this command to open the **Database Table Actions** dialog box where you can define the actions to be executed for each database record.

Run SQL Script: This command is located in the Output Preview toolbar and in the **Output** menu. Click this command to execute the SQL script that has been generated in the Output pane.

Files used in this example

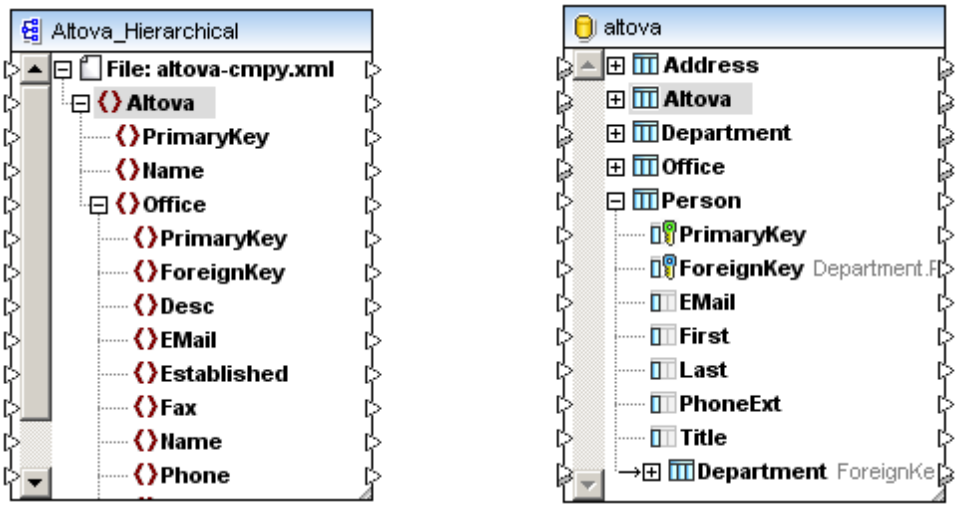
- Altova_Hierarchical.xsd
- altova-cmpy.xml
- altova-cmpy-extra.xml
- altova.mdb

Updating database fields

The first example deals with the simple task of updating existing Person records. Mappings are created from the XML data source to the "root" table Person.

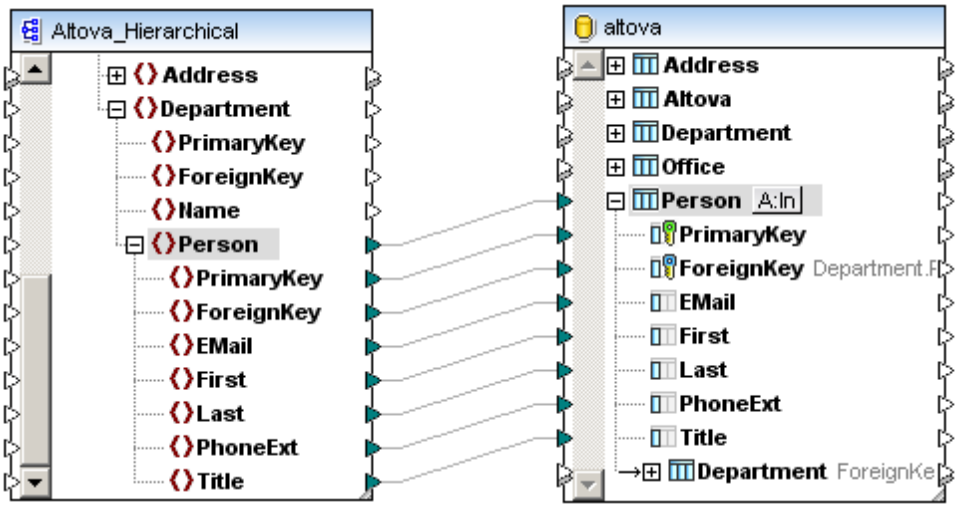
To update the Person table:

1. Insert the Altova_Hierarchical schema (and assign **altova-cmpy.xml** as the input XML instance) as described in the previous section.
2. Insert the MS Access database **altova.mdb** into the mapping as described in the previous section.

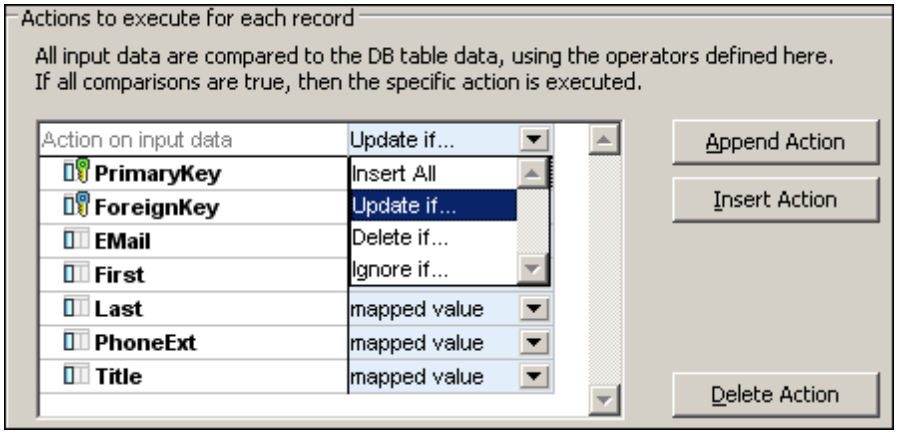


3. Activate the **Auto connect matching children** icon.
4. In the XML Schema component, expand the Office and Department items, click the **Person** item and drag the connector to the Person item of the database. Make sure that you connect to the "**root**" table, Person.

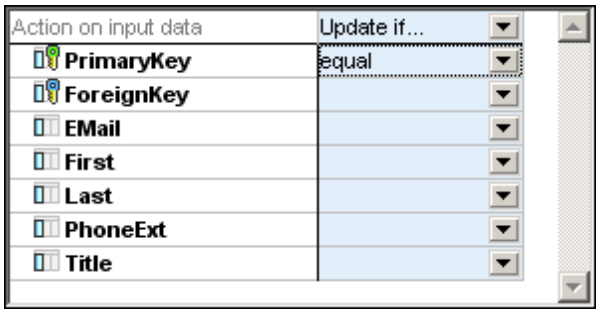
All matching child items are mapped automatically.



5. Click the **Person** Table Actions icon A:In to open the dialog box.
6. Click the Action on input data combo box, the topmost entry, and select **Update if...**.

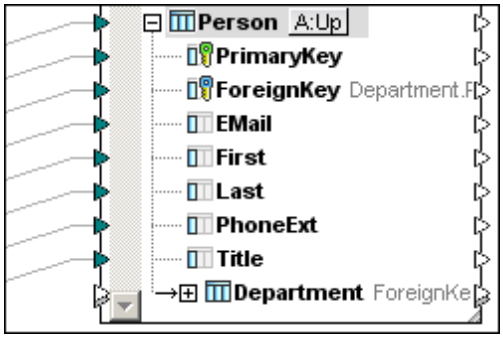


7. Click the combo box in-line with the PrimaryKey entry, and select the **equal** entry, click **OK** to confirm.



The source and target primary keys are compared using the "equal" operator. If the two keys are identical, then all the mapped fields of the Person tables are updated.

The Table Actions icon has now changed to **A:Up**, showing that the table action "Update" is selected.



8. Click the Output button at the bottom of the mapping window to see the SQL script that this mapping produces. The script executes the mapping to the target database, taking the defined table actions into account.
9. Click the **Run SQL-Script** icon in the function bar to run the script and update the database records. If the script was successful, a confirmation message appears in the Messages window.
10. Open the **Altova** database in MS Access, and double-click the **Person** table to see the effect.
    All the person records of the database have been updated.

Person : Table

PrimaryKey	ForeignKey	EMail	First	Last	PhoneExt
1	1	A.Aldrich@micı	Albert	Aldrich	582
2	1	b.bander@micr	Bert	Bander	471
3	1	c.clovis@micro	Clive	Clovis	963
4	2	d.Durnell@micr	Dave	Durnelll	621
5	2	e.ellas@microt	Eve	Ellas	753
6	3	f.fortunas@micı	Fred	Fortunas	951
7	3	g.gundall@micr	Gerry	Gundall	654
8	3	h.hardy@micro	Harry	Hardy	852
9	3	i.idilko@microte	Ingrid	Idilko	951
10	3	j.judy@microte	June	Judy	753
11	3	k.krove@microt	Karl	Krove	334

Updating and adding new records

This slightly more complex example, attempts to update records in both the Department and Person tables, as well as add any new Person records which might exist in the XML input file. The "**root**" table used in this example is thus the **Department** table.

Files used in this example:

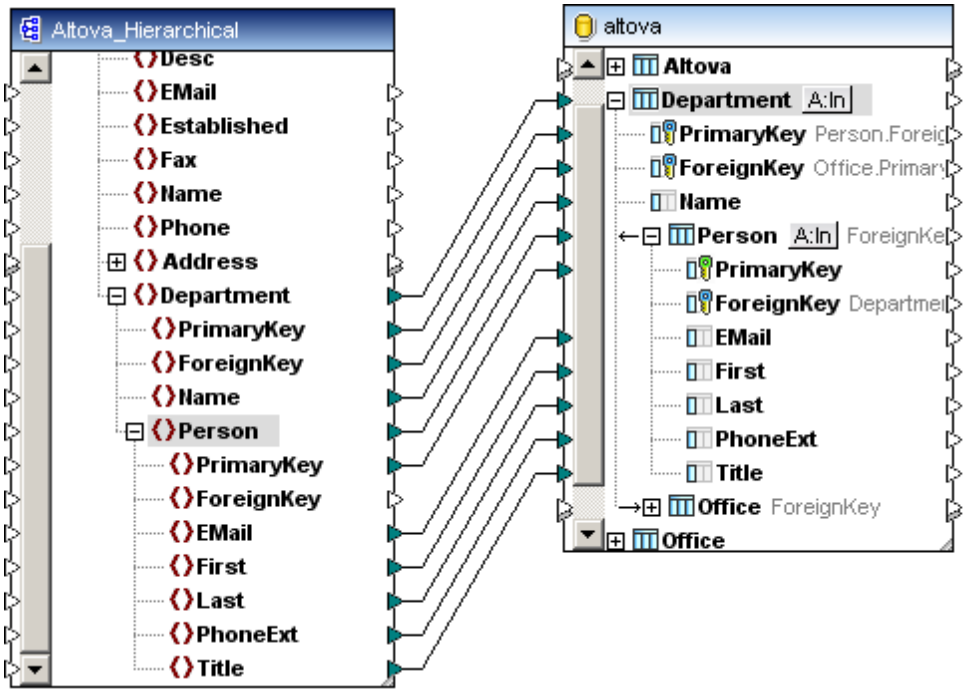
- Altova_Hierarchical.xsd
- altova-cmpy-**extra**.xml (is the XML instance for Altova_hierarchical.xsd)
- altova.mdb

Aim:

- to **update** the Department Name records
- to **update** existing Person records
- **insert** any new Person records

To map the Department and Person tables:

1. Insert the Altova_Hierarchical schema as described in the previous section,and assign **altova-cmpy-extra.xml** as the input XML instance.
2. Insert the MS Access database **altova.mdb** into the mapping as described in the previous section.
3. Activate the **Auto connect matching children** icon.
4. In the database component, expand the Department item and its child Person item.
5. In the XML Schema component, expand the Office and Department items, click the **Person** item and drag the connector to the Person item of the database. Make sure that you connect to the Person table that is nested inside the Department table.
   All matching child items are mapped automatically.

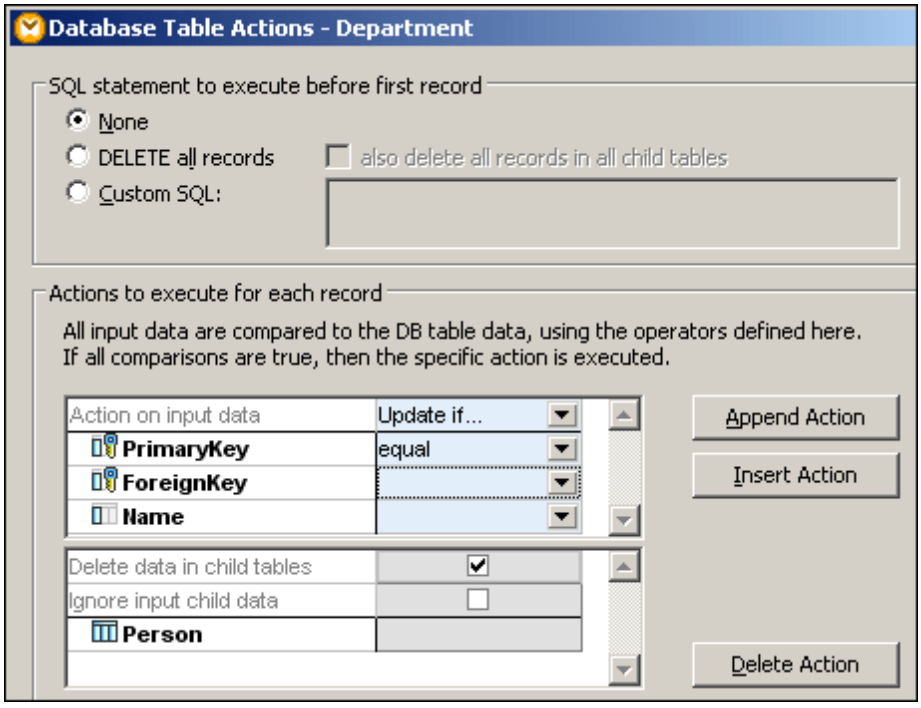


Defining the table actions

The source and target **primary keys** of both tables are compared using the "equal" operator. If the two keys are identical, then the **mapped** fields of the Department and Person tables are updated. If the comparison fails (in the Person table), then the next table action is processed, i.e. Insert Rest.

To define the table action for the Department table:

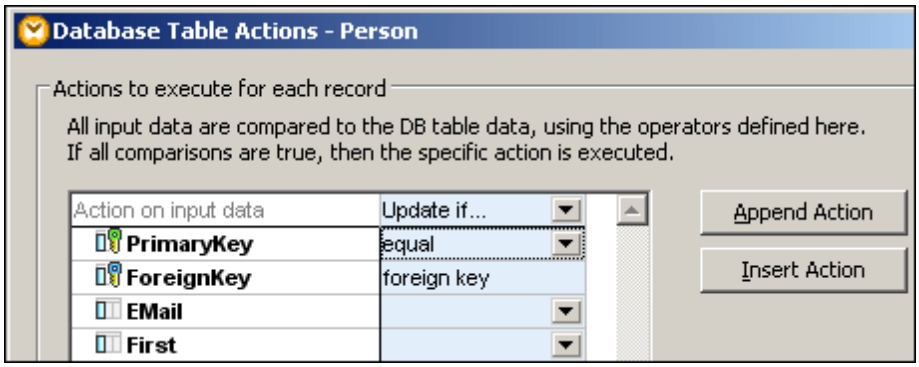
1. Click the **Department** Table Actions icon A:In to open the dialog box.
2. Click the Action on input data combo box, the topmost entry, and select **Update if...**.
3. Click the combo box in-line with the PrimaryKey entry, and select the **equal** entry, click **OK** to confirm.



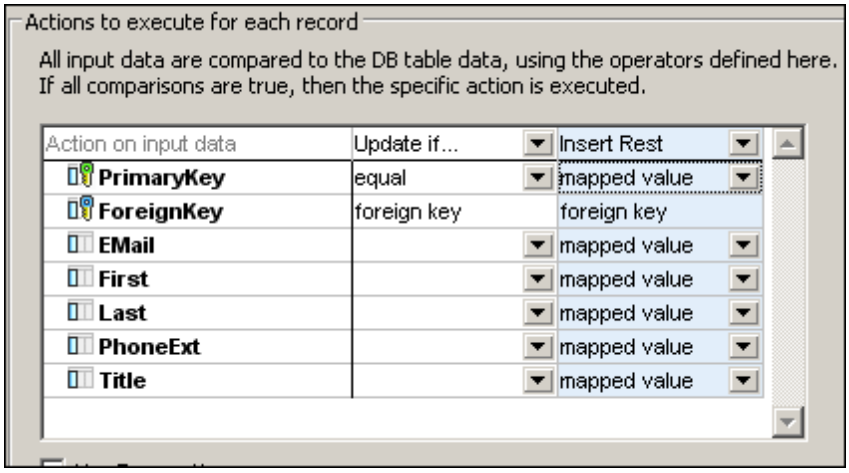
The source and target primary keys are compared using the "equal" operator. If the two keys are identical, then all the mapped fields of the Department tables are updated.

To define the table action for the Person table:

1. Click the **Person** Table Actions icon A:In to open the dialog box.
2. Click the Action on input data combo box, the topmost entry, and select **Update if...**.
3. Click the combo box in-line with the PrimaryKey entry, and select the **equal** entry



4. Click the **Append Action** button to append a new Table action column.
   The table action **Insert Rest** is automatically inserted as the second table action. If the Update if... actions fails then this table action is processed.



5. Click **OK** to complete the table actions definition. Note that the table actions icon has now been updated to **A:Up,In**.

6. Click the Output button to see the SQL script.

```
UPDATE [Department] SET [ForeignKey] = 1, [Name] = 'Admin' WHERE ([Department].[PrimaryKey]=1)

SELECT [ForeignKey], [PrimaryKey] FROM [Department] WHERE ([PrimaryKey]=1)
-->>> %ForeignKey1%

-->>> %PrimaryKey1%

DELETE FROM [Person] WHERE EXISTS(SELECT * FROM [Department] WHERE [Department].[PrimaryKey] =
[Person].[ForeignKey] AND ([Department].[PrimaryKey]='%PrimaryKey1%'))

UPDATE [Person] SET [EMail] = 'A.Aldrich@microtech.com', [First] = 'Albert', [Last] = 'Aldrich', [PhoneExt] = 582
, [Title] = 'Manager' WHERE ([Person].[ForeignKey] = '%PrimaryKey1%') AND ([Person].[PrimaryKey]=1)
```

Please note: The script executes the mapping to the target database, taking the defined table actions into account.

Processing sequence

The **Update if...** condition checks whether or not the primary keys of the source and target items are identical.

Department table:

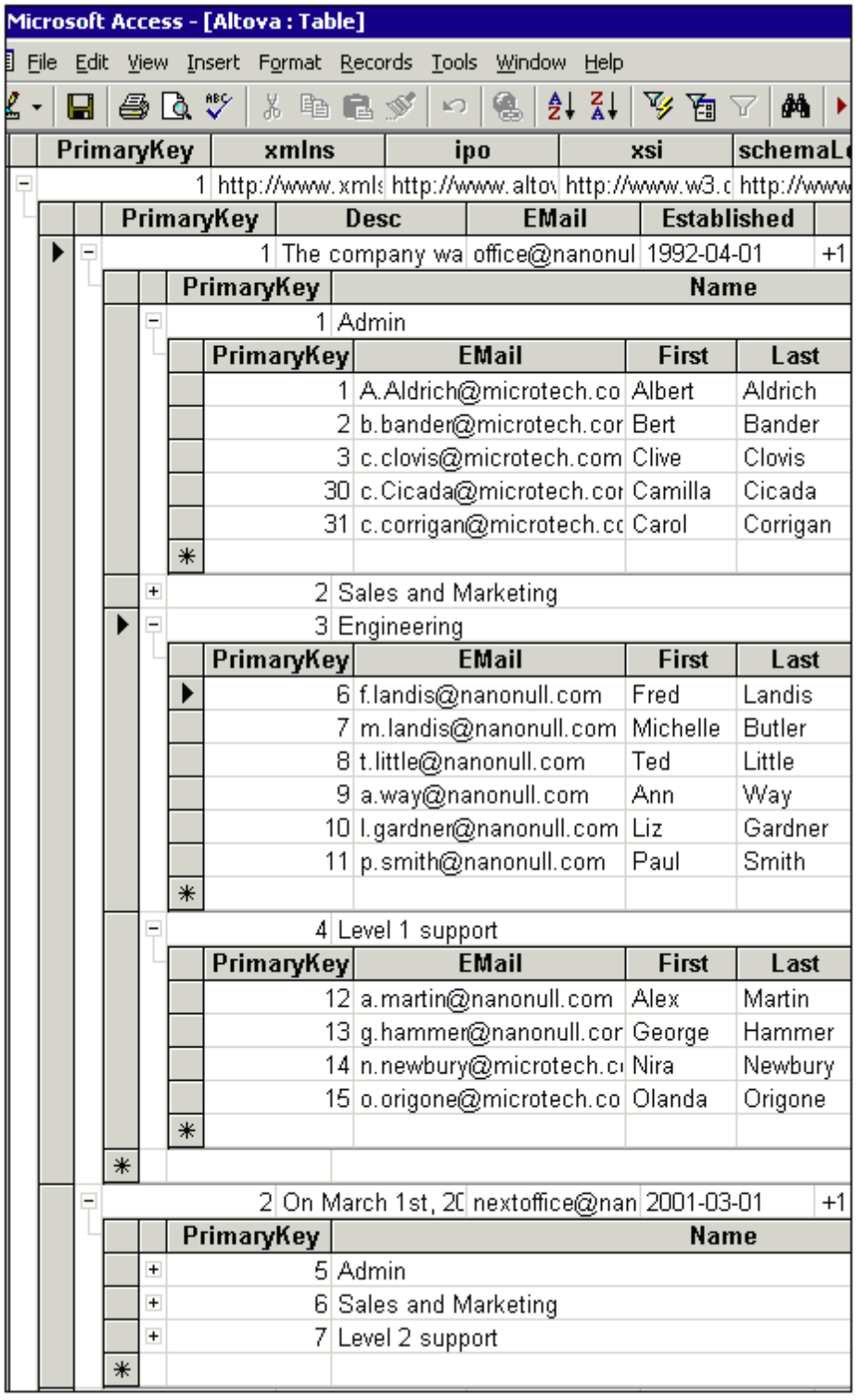
- If the **condition is true**, then each Department record where the keys are identical is updated. If records exist in the database with no counterpart in the source file, then these records are retained and remain unchanged (in this example the Engineering table).
- If the **condition is false**, i.e. source keys exist which have no match in the target database, none of the Department records are updated.

Person table:

- If the **condition is true**, then each Person record where the keys are identical is

updated. If records exist in the database with no counterpart in the source file, then these records are retained and remain unchanged.

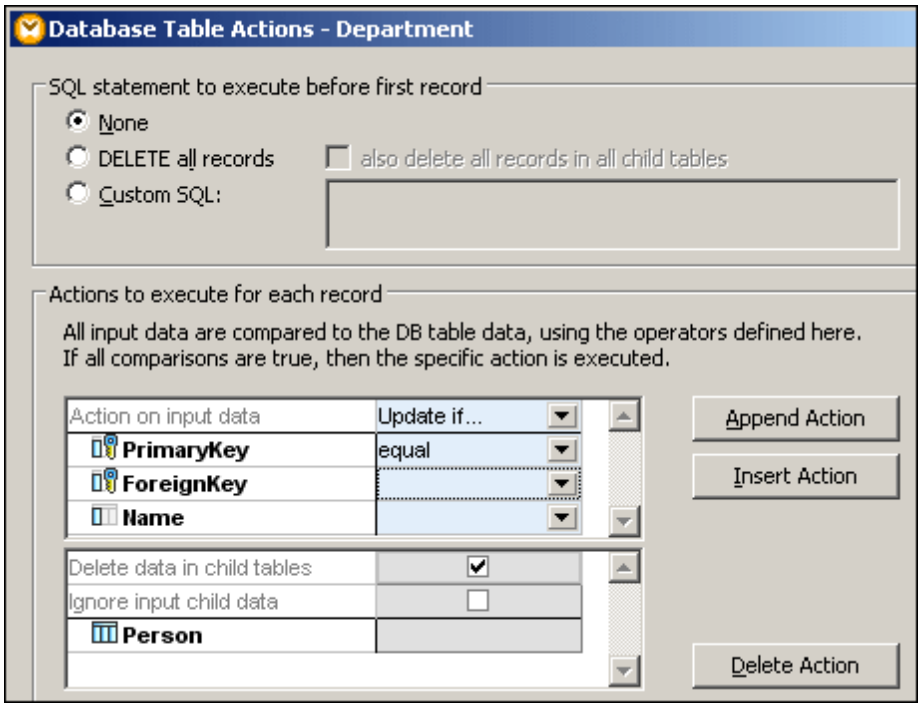
- If the **condition is false**, i.e. source keys exist which have no match in the target database, then MapForce moves on to the next Table Action column: **Insert Rest**. This action inserts the new Person records into the Person table if any exist. In this example, two new person records are added to the Admin department, with the person primary keys of 30, and 31, respectively.



Updating and deleting child data

This section describes the effect of the **Update if...** condition on a parent table combined with each of the possible table actions (i.e., Insert All, Update if..., Delete if...) defined for related child

tables. The "**Delete data in child tables option**" is active in all **but one** of these examples. You can continue to use the mapping from the previous section, for this section.



Files used to illustrate this example:

- Altova_hierarchical.xsd
- Altova-cmpy-**extra**.xml
- Altova.mdb

The settings for database actions of the **parent table (Department)** are as follows:

Action on input data	Update if...
PrimaryKey	equal
ForeignKey	
Name	
Delete data in child tables	☑*
Ignore input child data	☐

* For the Delete if... action, this check box may also be deactivated (*see table below*).

The result of the mapping depends on the "Action on input data" you select for the **child table (Person), i.e.**:

Insert all...	**Update if...**	**Delete if...**
Updates parent table data (Department records).		
Deletes child data of those tables which satisfy the Update if... condition (Person records).	If the "Delete data in child tables" option is **active**, deletes child data (Person records) from all Departments. All Person records are	If the "Delete data in child tables" option is **inactive**, deletes child data of those tables which satisfy the Update if... condition

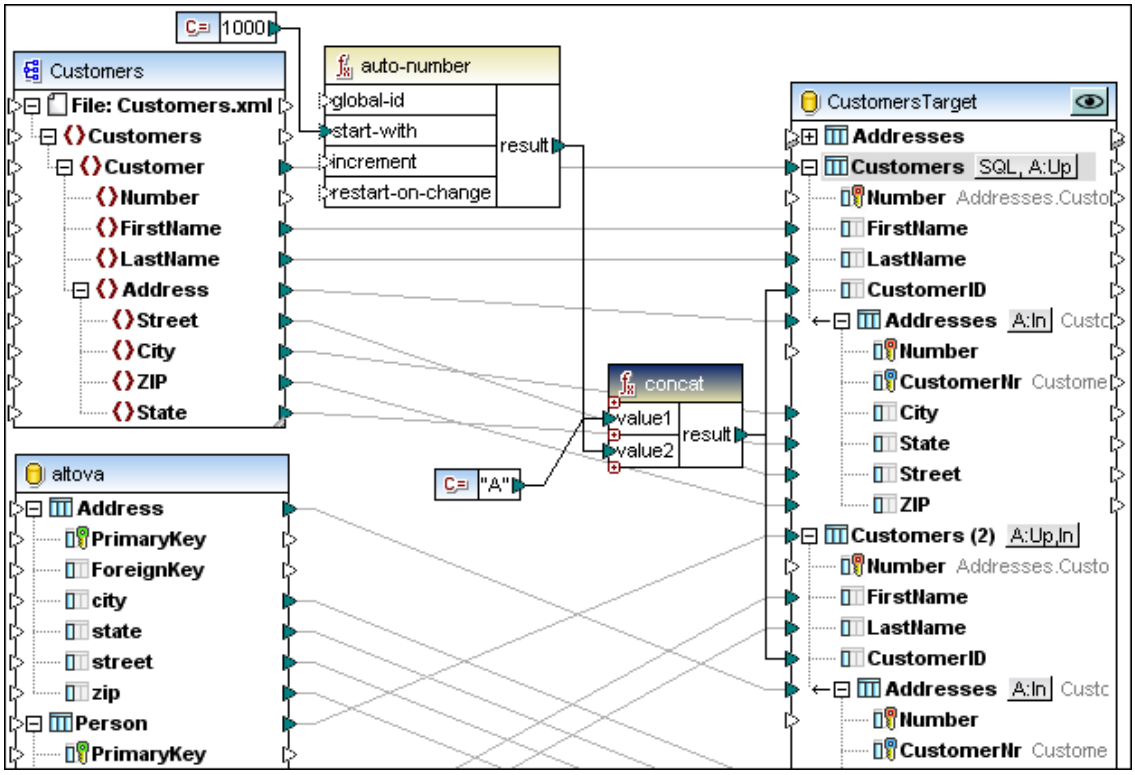
		deleted for each Department which has a corresponding PrimaryKey in the source XML. I.e. even **Person** records of the database which have no counterpart in the source XML, are deleted.	(Person records).
Retains existing database tables (Engineering in this case) which do not have a counterpart in the input XML file (no source keys for the Update if... comparison).			
Inserts all Person records from the input XML instance. This also includes new records that might not already exist in the database.	Update if... condition, defined for the Person table, fails because all Person records in the database have been deleted by the "Delete data in child tables" option. There is no way to compare the database and XML data primary keys, as the database keys have been deleted. No records are updated.	The child table data (Person records) are deleted before the Table action, Delete if..., is executed, no records are deleted.	Database records which do not have the corresponding Person key, are retained.

To see a further example involving duplicate items, Insert, Update and transactions, please see the **Customers_DB.mfd** sample file available in the ...\MapForceExamples folder. The example shows how XML schemas and database sources can be mapped to target databases.

In the example:

- XML Schema to database:
  Customers and Addresses exist in the target database. These entries are updated with the new data from the source XML Schema/document. The FirstName and LastName items are used to find the correct rows in the database.
- Database to database:
  Address and Person data are supplied by the database source and are inserted into the target database. The target table (Customers) is duplicated.

  CustomerID for each record are created anew, with the initial value being A1000.



7.2.3.6 *Database action: Delete*

The table action Delete if... is used to selectively delete data from tables. This is achieved by selecting specific items/fields of the source and target components which are to be compared. The specific table action is then executed depending on the outcome of this comparison.

Please note: This table action should not be confused with the "**Delete data in child tables**" option, available in the table action dialog box. The Delete if... table action only affects the table for which the action is defined, no other tables are affected.

Objective

In this section of the tutorial. you will learn how to use MapForce to delete data from database tables. Specifically, you will learn how to do the following:

- Delete the existing Person records in the database
- Insert new Person records from the input XML file

Commands used in this section

Insert XML Schema/File: Click this icon to open the standard Windows **Open** dialog box and select the file from your file system.

Insert database: This command is located in the Insert Component toolbar and in the **Insert** menu. Click this command to open the **Select a Database** dialog box where you can define a database connection and select the database tables to be included in the MapForce mapping.

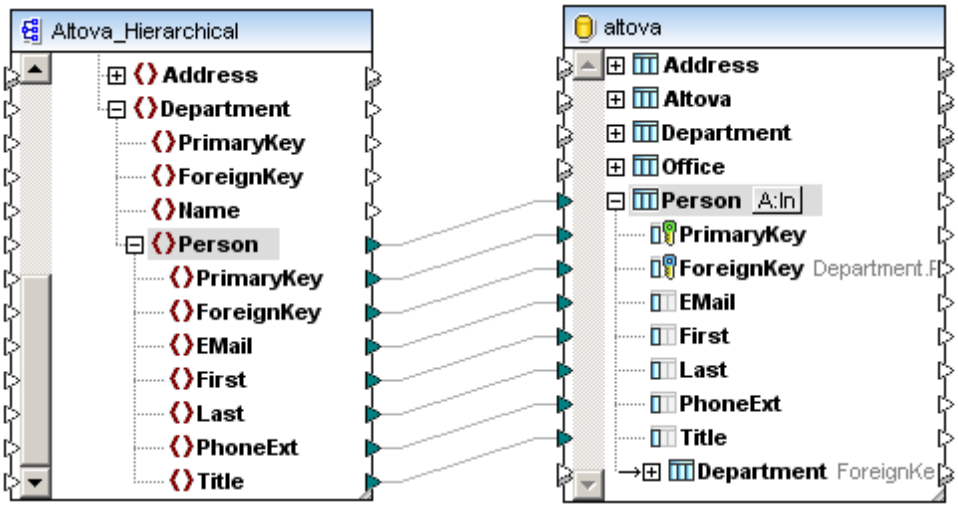
Auto Connect Matching Children: Click this icon to toggle the automatic connection of matching child nodes, on and off.

Database Table Actions: This command is located in the **Component** menu of a database component and appears as a button to the right of the table name in the database component. Click this command to open the **Database Table Actions** dialog box where you can define the actions to be executed for each database record.

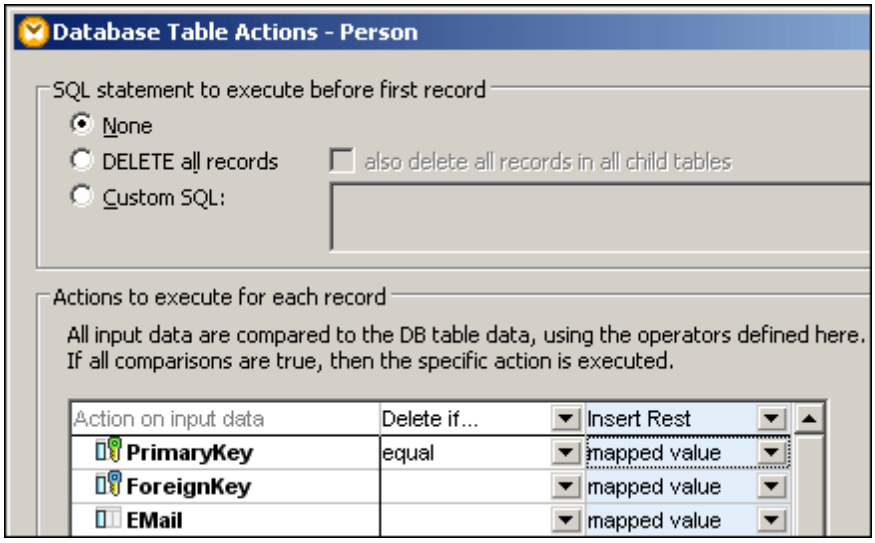
Run SQL Script: This command is located in the Output Preview toolbar and in the **Output** menu. Click this command to execute the SQL script that has been generated in the Output pane.

To set up the mapping and define the database table actions for Delete if...:

1. Insert the Altova_Hierarchical schema (and assign **altova-cmpy-extra.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping and select all User Tables.



3. Select the menu option **Connection | Auto Connect matching children** (if not already active).
4. Click the Person item in the XML source component and drag the connector to the Person item of the database. Make sure that you connect to the "root" table, **Person**. All matching child items are mapped automatically.
5. Click the **Person** Table Actions icon A:In to open the **Database Table Actions** dialog box.
6. Click the "Action on input data" combo box and select **Delete if...**
7. Click the **Append Action** button.
   This automatically inserts a new Table action column with the table action "Insert Rest".



8. Click the "PrimaryKey" combo box and select **equal**.

   The source and target primary keys are compared using the "equal" operator. If the two keys are identical, then the record is deleted.

   Insert Rest creates new records in the Person table, based on the source XML file data, which do not have a counterpart key/field in the database.

9. Click **OK** to close the dialog box.
   The Table Action icon now displays **A:De,In**.

   Note: if a combo box down arrow is not available for the PrimaryKey or ForeignKey entries, you have to change the specific key handling properties, please see Table actions, Key settings.

10. Click the Output button at the bottom of the mapping window to see the SQL script that this mapping produces. The script executes the mapping to the target database, taking the defined table actions into account.
11. Click the Run-SQL-Script icon  in the function bar to run the script and update the database records.
12. Open the Altova database in MS Access and double click the **Person** table to see the effect.

Person : Table

	PrimaryKey	ForeignKey	EMail	First	Last
▶	6	3	f.landis@nanonull.com	Fred	Landis
	7	3	m.landis@nanonull.co	Michelle	Butler
	8	3	t.little@nanonull.com	Ted	Little
	9	3	a.way@nanonull.com	Ann	Way
	10	3	l.gardner@nanonull.cc	Liz	Gardner
	11	3	p.smith@nanonull.cor	Paul	Smith
	12	4	a.martin@nanonull.co	Alex	Martin
	13	4	g.hammer@nanonull.c	George	Hammer
	30	1	c.Cicada@microtech.c	Camilla	Cicada
	31	1	c.corrigan@microtech	Carol	Corrigan
*					

Processing sequence

The **Delete if...** condition checks whether or not the primary keys of the source and target items are identical.

Person table:

- If the **condition is true**, then each Person record where the keys are identical is deleted. If records exist in the database with no counterpart in the source file, then these records are not deleted and remain unchanged.
- If the **condition is false**, i.e. source keys exist which have no match in the target database, then MapForce moves on to the next Table Action column: **Insert Rest**. This action inserts the new Person records into the Person table if any exist. In this example, two new person records are added to the Administration department, with the person primary keys of 30, and 31, respectively.

Two additional examples of the Delete if... table action can be viewed in the Update if... combinations section.

7.2.3.7 Database action: Ignore

The table action Ignore if... is used to selectively ignore specific records created by the mapping. A SELECT statement is generated with the specified condition. If this statement finds the identical data, the corresponding mapped input record is ignored.

This action can be used together with a following "Insert Rest" action to ignore all input records that already exist in the database, and to insert any new ones.

Objective

In this section of the tutorial, you will learn how to:

- Ignore duplicate Person records in the database that originate from the Input XML file
- Insert new Person records from the Input XML file

Commands used in this section

Insert XML Schema/File: Click this icon to open the standard Windows **Open** dialog

box and select the file from your file system.

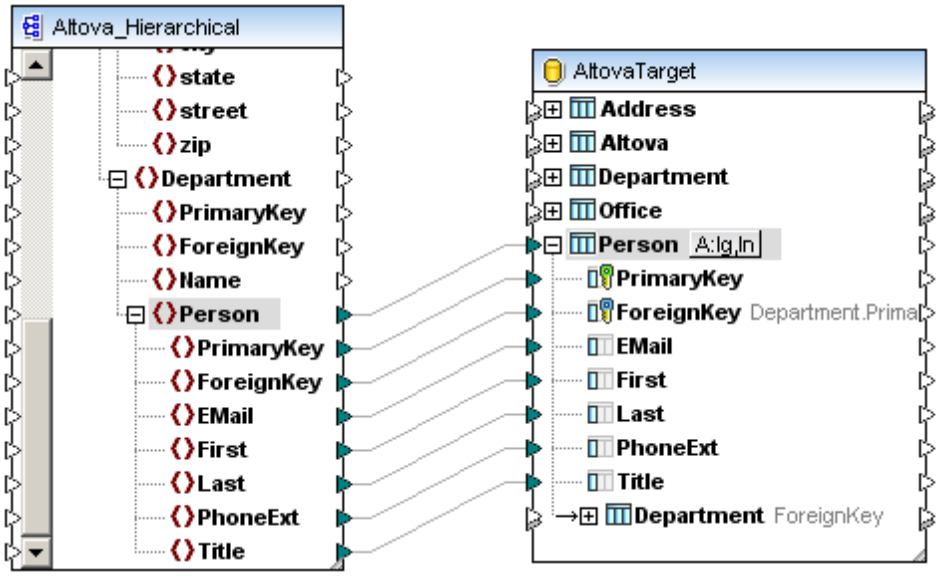
Insert database: This command is located in the Insert Component toolbar and in the **Insert** menu. Click this command to open the **Select a Database** dialog box where you can define a database connection and select the database tables to be included in the MapForce mapping.

Database Table Actions: This command is located in the **Component** menu of a database component and appears as a button to the right of the table name in the database component. Click this command to open the **Database Table Actions** dialog box where you can define the actions to be executed for each database record.

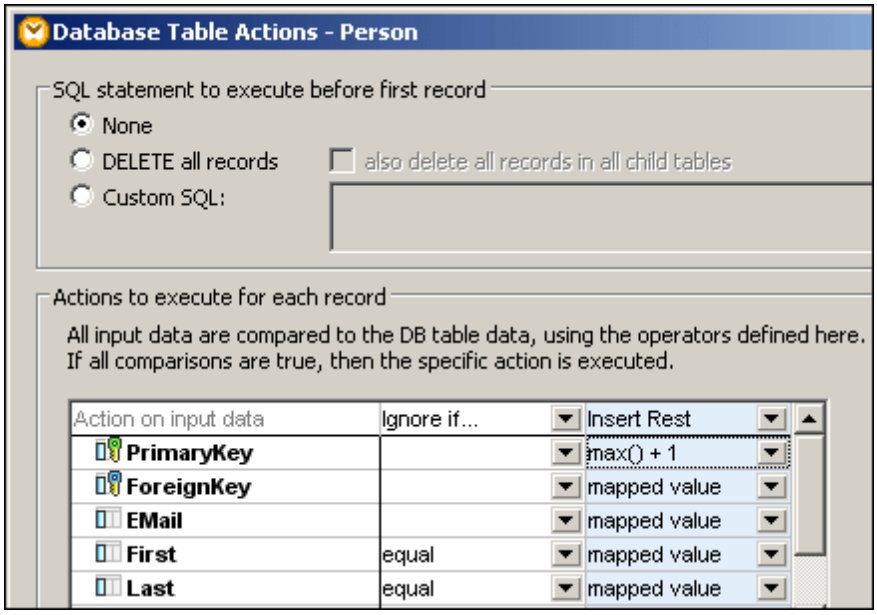
Auto Connect Matching Children: Click this icon to toggle the automatic connection of matching child nodes, on and off.

To set up the mapping and define the database table actions for Ignore if...:

1. Insert the Altova_Hierarchical schema (and assign **altova-cmpy-extra.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping and select all User Tables.



3. Select the menu option **Connection | Auto Connect matching children**.
4. Click the Person item in the XML source file and drag the connector to the Person item of the database. Make sure that you connect to the "root" table, **Person**.
   All matching child items are mapped automatically.
5. Click the **Person** Table Actions icon A:In to open the **Database Table Actions** dialog box.
6. Click the "Action on input data" combo box and select **Ignore if...**
7. Click the **Append Action** button.
   This automatically inserts a new Table action column with the table action "Insert Rest".



8. Click the "PrimaryKey" combo box and select **max() + 1**.
9. Click the "First" combo box and select "equal", then do the same for the "Last" combo box.
10. Click **OK** to close the dialog box.

Processing sequence

The **Ignore if..**. table action compares the First and Last fields of the source XML with the same fields in the database:

- If the data of both fields are found in the database, then the **mapped data** is **ignored** and the Insert Rest... table action is not processed.
- If the comparison on either the First or Last fields fails, then a new record is generated in the database, using the max()+ 1 PrimaryKey entry.

7.2.3.8 Generating database output values

Generator functions for the programming languages, as well as the Built-in execution engine, can generate values for database fields, which do not have any input data from the Schema, database or EDI source component.

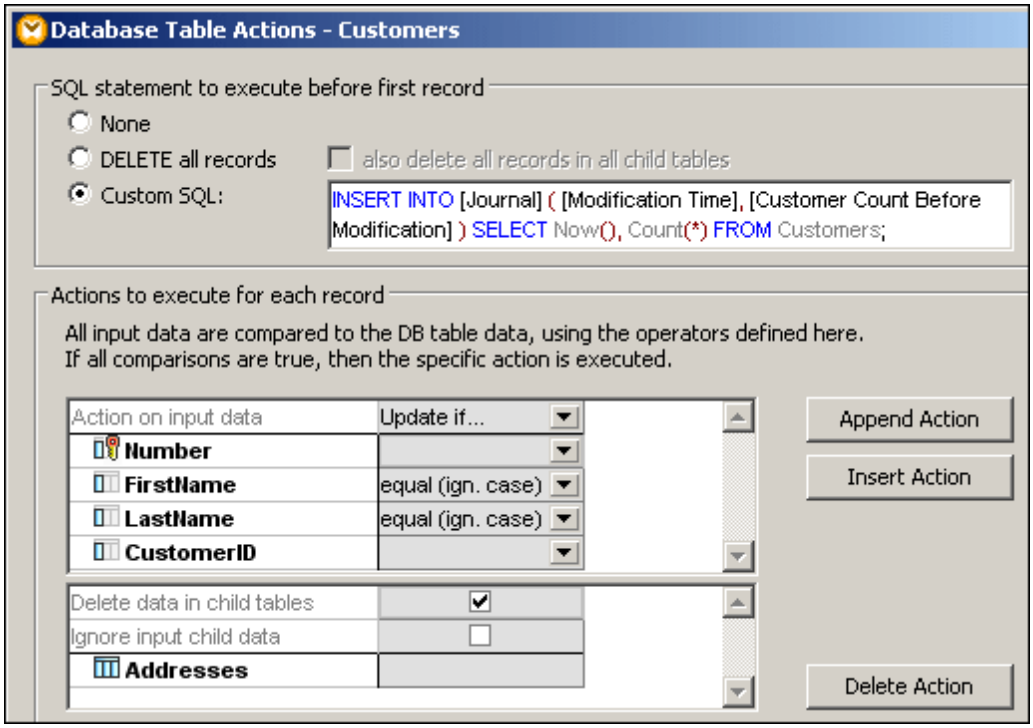
Auto-number and create-guid can both generate values for fields.

auto-number (core | generator functions) - also available for Built-in execution engine.
is generally used to generate primary key values for a numeric field.

create-guid (lang | generator functions)
Creates a globally-unique identifier (as a hex-encoded string) for the specific field.

7.2.4 Table Actions, key settings, transaction processing

The Database Table Actions dialog box lets you specify (i) an optional SQL statement to be executed prior to any other table action, (ii) the table actions to be performed for each record delivered to the target table by the mapping, and (iii) the database key settings for inserting new records.



SQL statement to execute before first record

In this group box you can define SQL statements that are executed before processing the records produced by the mapping. Select the desired radio button:

- **None** (default setting): No action is carried through
- **DELETE all records**: All records from the selected table are deleted before any specific table action defined in the Actions to execute for each record group box is performed. Activate the **also delete all records in all child tables** check box if you also want to get rid of the data stored in child tables of the selected table.
- **Custom SQL**: Write a custom SQL statement to affect the complete table.

Note: Support for multiple SQL statements in one query depends on the database, connection method, and the driver used (see Connecting to a Database).

Actions to execute for each record

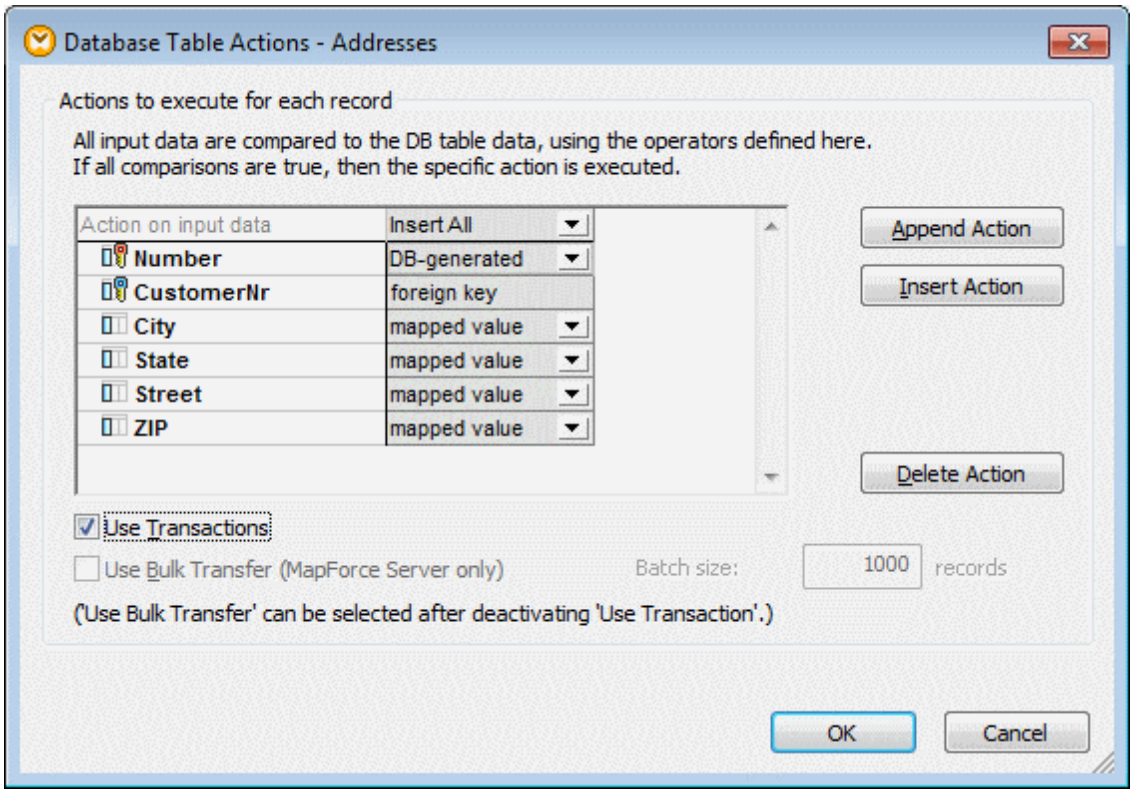
The Table Actions define how the mapped records are processed. Several options exist: Insert, Update, Delete and Ignore are selected using the combo boxes entries of "Action on input data".

See Table actions for more information.

7.2.4.1 Table actions

In the Actions to execute for each record group box, you define the table actions that determine how the mapped records are processed. MapForce supports the table actions: Insert, Update, Delete and Ignore.

The Update, Delete and Ignore table actions require a **condition** that defines for which records they should apply. One or more fields are used to compare source and target data to determine if the table action is to be executed. The Insert action is unconditional but it has **key settings** to define how to generate the primary key.



For each mapped database column in the Table Action dialog, you define:

- fields that will be compared (e.g. PrimaryKey, FirstName etc.)
- operators used for the comparison (equal, equal ignoring case), and
- action taken, when all conditions of each column are fulfilled.

Table Actions are processed from left to right. In the example above, the **Update if...** column is processed first. If the update condition is not satisfied then the following action is processed e.g. the **Insert Rest** column.

- **All** the defined conditions of one column must be satisfied if the table action is to be executed. When this is the case, all those fields are updated where a **mapping exists**, i.e. a connector exists between the source and target items in the Mapping window. Any following table actions (to the right of an action whose condition matched) are ignored for

that record.

- If a condition is not satisfied, then the table action for that column is skipped, and the next column is processed.
- If none of the conditions are "true", no table action takes place.

Please note:
Any table actions defined after **Insert All/Insert Rest,** will never be executed as there are no column conditions for insert actions. A dialog box will appear if this is the case, stating that the subsequent table action columns will be deleted.

Delete data in child tables:

- Standard setting when you select the **Update if...** action.
- Necessary if the no. of records in the source file might be different from the no. of records in the target database.
- Helps keep the database synchronized (no orphaned data in child tables)

Effect:

- The Update if... condition is satisfied when a corresponding key (or any other field) exists in the source XML file. All **child data** of the parent table are deleted.
- Update if... selects the parent table, and thus the child tables related to it, on which the "Delete data in child tables" works.
- If the update condition (on the parent) is not satisfied, i.e. no corresponding key/field in source XML file exists, then child data are not deleted.
- Existing database records, that do not have a counterpart in the source file, are not deleted from the database, they are retained.

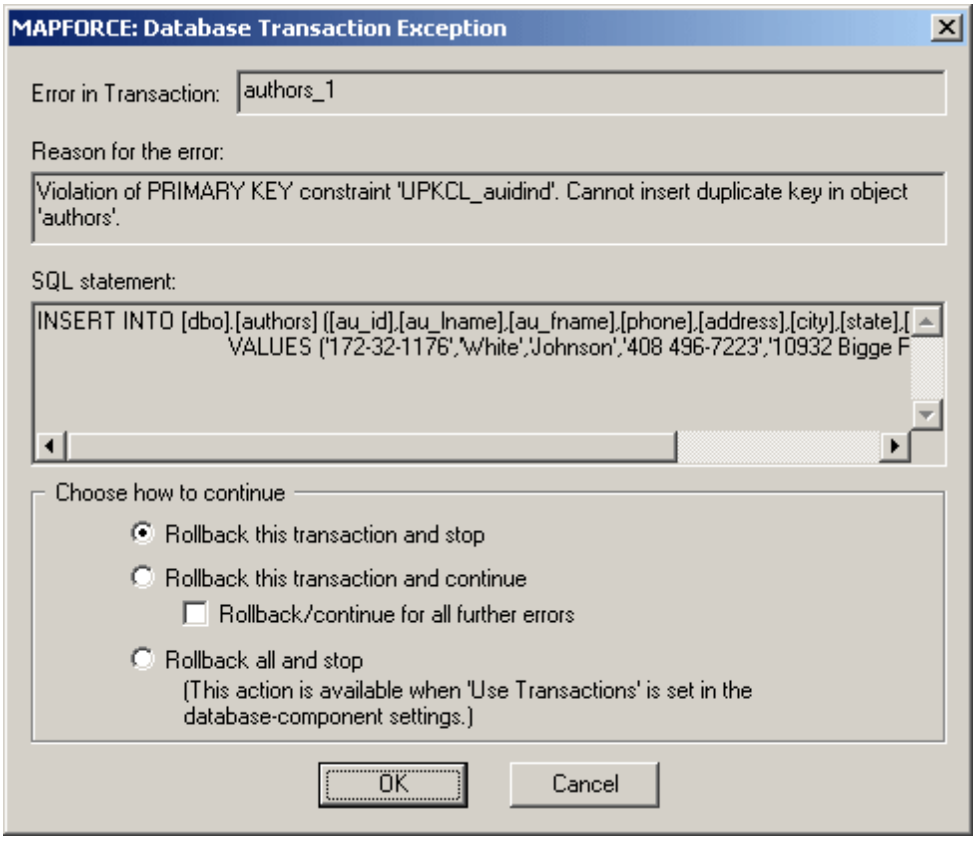
Ignore input child data:
Use this option when you want to update specific table data, without affecting any of the child tables/records of that table.

For example, your mapping setup might consist of 3 source records and 2 target database records.
You would therefore need to:

- define an **Update if...** condition, to update the existing records
- activate the **Ignore input child data** check box, of the **Update if... column**, to ignore the related child records, and
- define an **Insert Rest...** condition for any new records, that have to be inserted.

Use Transactions:
The "Use Transaction" check box allows you to define what is to happen if a database action does not succeed for whatever reason. When such an exception occurs, a dialog box opens prompting you for more information on how to proceed. You then select the specific option and click OK to proceed. Activating this option for a specific table (using the table action dialog box), allows that specific database table to be rolled back when an error occurs.

The transaction settings can also activated for the database component, by activating "Use Transactions" in the Component Settings dialog box of the respective database component, (double click to open the dialog box). In this case, all tables can be rolled back.



No Transaction options set:
If the transaction check box has not been activated in the table options, or in the component settings, and an error occurs:

- Execution stops at the point the error occurs. All previously successful SQL statements are executed and the results are stored in the database.

Transaction option set at **database component** level:

- Execution stops at the point the error occurs. All previously successful SQL statements are rolled back. No changes are made in the database. All previously successful SQL statements for that for the **database** and all its tables can be rolled back.

Transaction option set at **Table Actions** level:

- The Transaction exception dialog box appears with the "Rollback all and stop" option **disabled**. The failed SQL statement for that specific **table** can be rolled back.

Transaction option set at both **database component** and **table action** level:

- The Transaction exception dialog box appears with the "Rollback all and stop" option **enabled**. All previously successful SQL statements for that for the **database** and all its tables can be rolled back.

Hitting the **Cancel** button, rolls back the current SQL statement and stops.

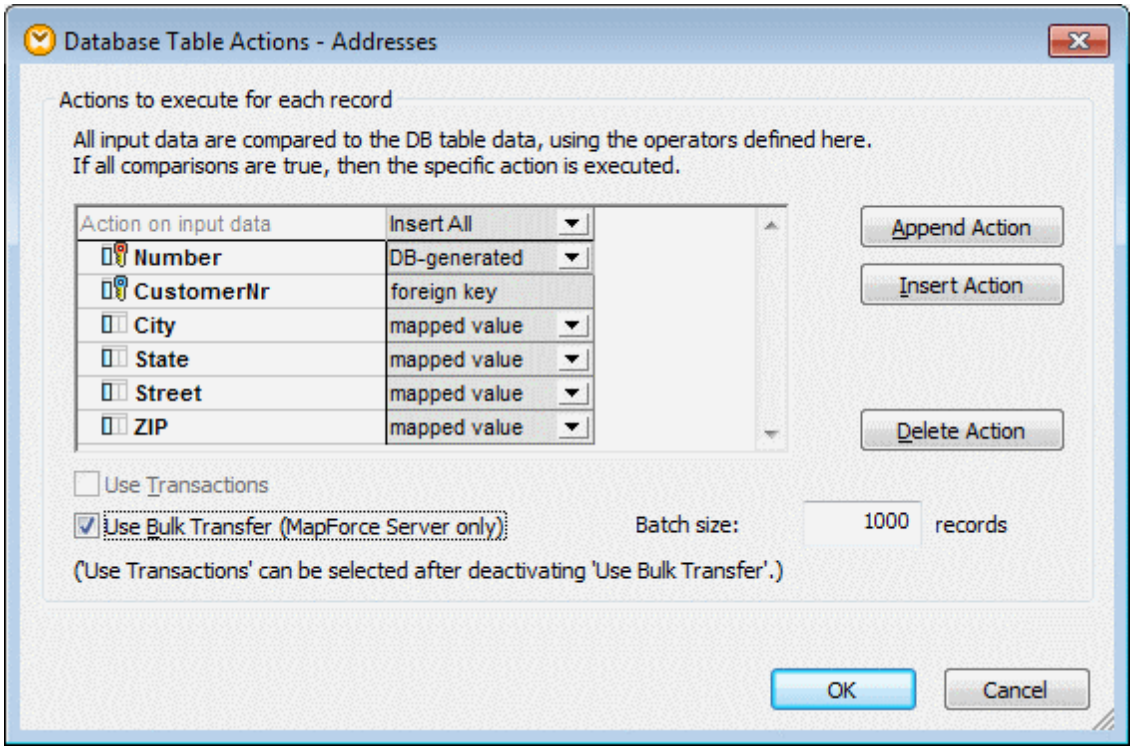
Please note:
The transaction prompts are only displayed when the transformation is performed

interactively!

Generated code performs a rollback (and stop) when the first error is encountered.

Use Bulk Transfer

The Bulk Insert command allows you to load data from a data file (TXT, CSV, DAT, etc.) into a database table and is only available when Built-in is used as the mapping target, and the mapping is executed with MapForce Server. Using this option dramatically speeds-up the Insert process as only one statemement needs to be executed.



This option can only be selected if the "Use Transactions" dialog box has been disabled. Please see Bulk Inserts (MapForce Server) for more information.

7.2.4.2 *Bulk Inserts (MapForce Server)*

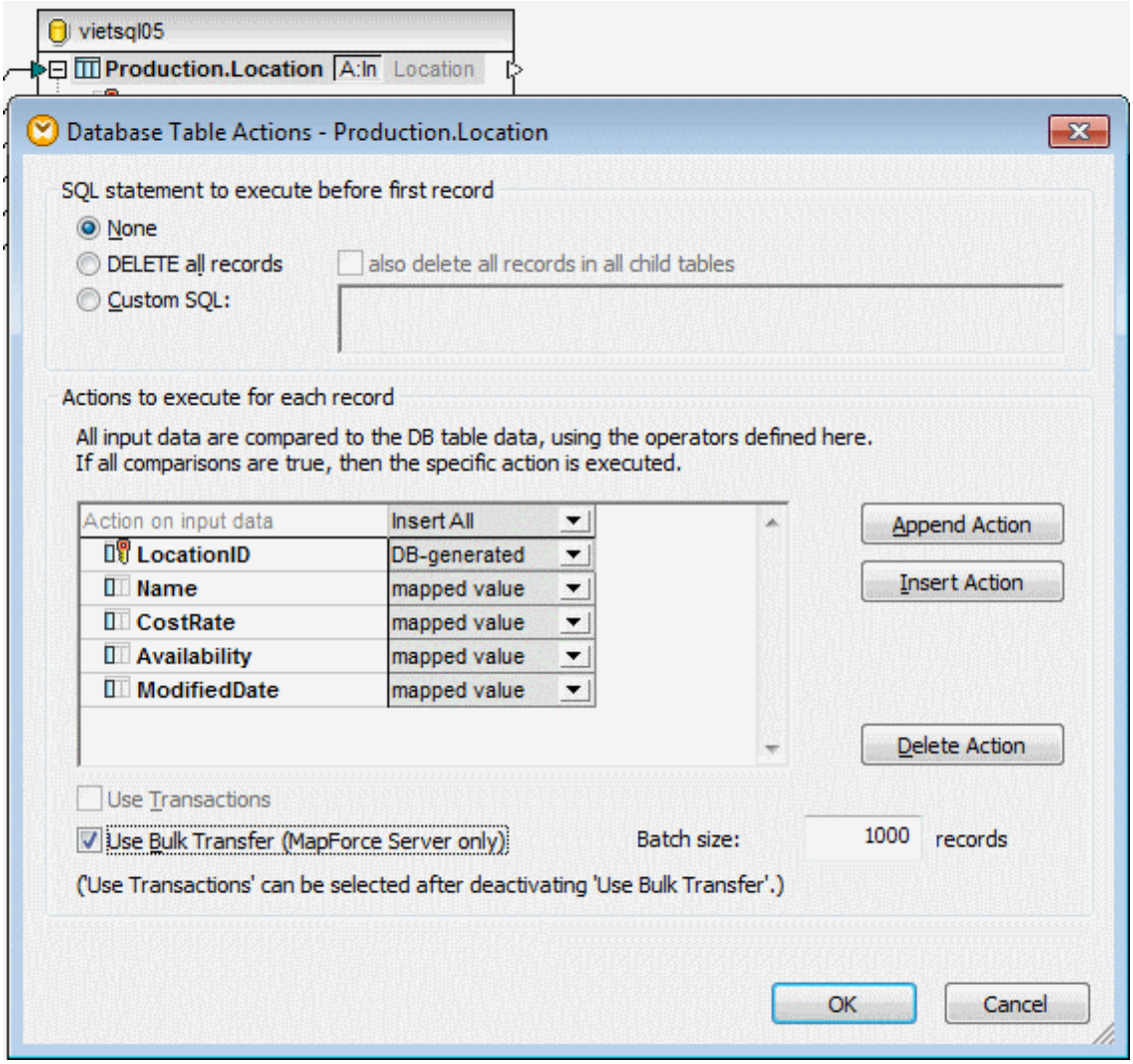
The **Bulk Insert** command allows you to insert data at very high speed from a MapForce component (TXT, CSV, DAT, etc.) into a database table. Using this option dramatically speeds-up the Insert process, as only one statement needs to be executed.

Note: If you are connecting to Microsoft Access and MySQL through ODBC, bulk inserts are not supported by the ODBC driver.

- Bulk inserts can only be defined with **Built-In** as the output target in MapForce
- Bulk Insert can only be **run** on MapForce Server (the mapping needs to be supplied/ deployed using the "Compile to MapForce Server execution file option".
- If the mapping is deployed to FlowForce Server, using the Deploying a MapForce mapping option then you can define a job that will execute the mapping depending on the

triggers defined in FlowForce. FlowForce then executes the job on MapForce Server.
- The MapForce Server license cannot be limited to "single thread execution" on a multi-core machine
- Bulk Inserts can be performed for "Insert all" actions
- The table into which the data is to be bulk loaded must be a "leaf table", i.e. on the lowest hierarchy of the database. There should not be any related tables, views, or stored procedures referencing the table in the mapping.

The "Use Bulk Transfer" option is available by clicking the Table Action icon A:I in a database target component as shown below (next to the Production.Location table). The "Batch size" field defines the number of records to be inserted per action.



Bulk inserts for "Insert all", are supported for those drivers that support Bulk Inserts on WHERE conditions. Please note: When selecting "Use Bulk Transfer", this automatically deselects the "Use Transactions" option. You have to deselect the Bulk Transfer check box to be able to enable transaction processing.

SQL Server is the only database that supports ADO, ODBC and JDBC connections. All other

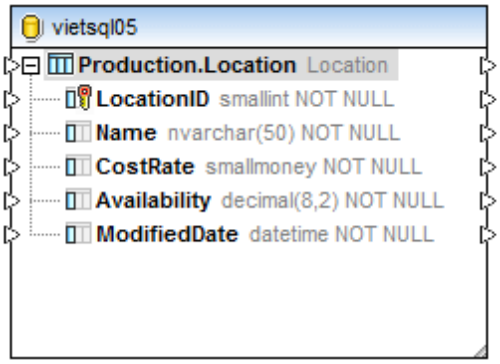
databases support ODBC connections.

Example

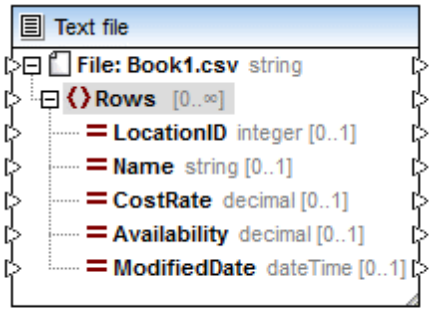
This example uses SQL Server 2008 and the **AdventureWorks** database that can be downloaded from the CodePlex website to show how Bulk inserts can be defined in MapForce.

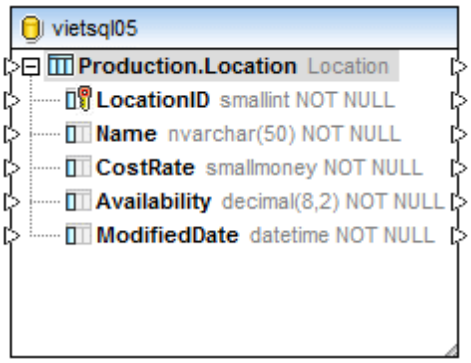
To define a bulk insert:

1. Make sure that the Built-In target output icon  is active.
2. Insert a database component into a mapping using Insert | Database, e.g. the Production.Location table.

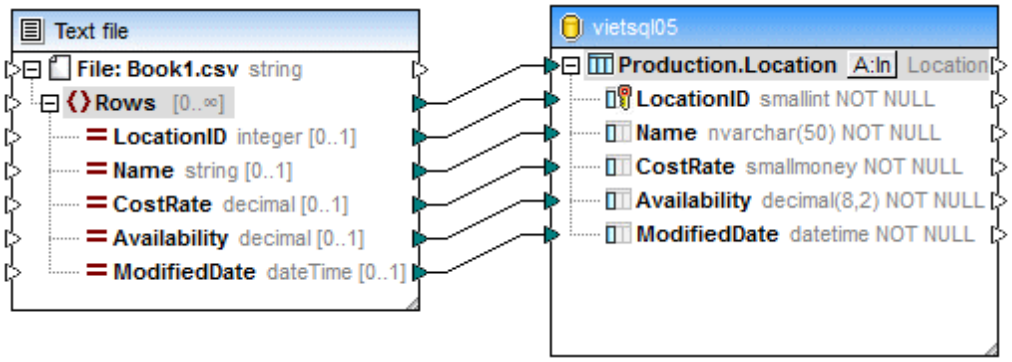


3. Insert the data file that you intend to use as the source for the Bulk insert, e.g Insert Text File. Make sure that the datatypes of both the source and the target components are identical.

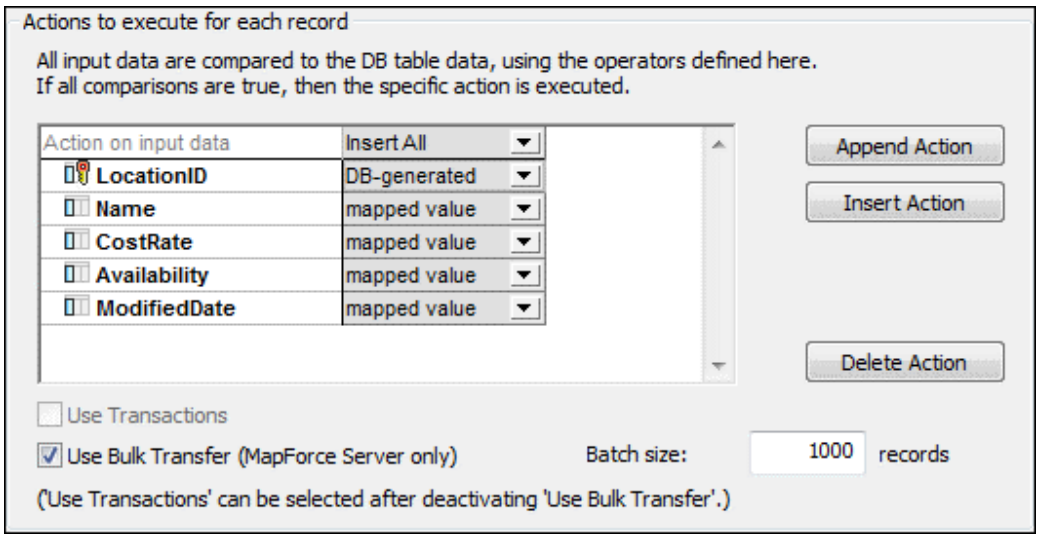




4. Connect the Rows item with the table name (Production.Location) in the database component. All the other items are automatically connected if the **Autoconnect Matching Children** option is active (see Connecting matching children). Note that the "Database Actions" button [A:In] is now visible to the right of the table name.



5. Click the Database Actions button A:In, activate the "Use Bulk Transfer" check box and click OK to confirm.



Once you defined a bulk insert action, the next step is to compile it to a MapForce Server Execution file.

To compile a mapping via the MapForce command line:

- Execute MapForce and specify the mapping file and the `/COMPILE` command line option. The MapForce Server Execution file will be created in the same directory as the mapping file.

To compile a mapping using the MapForce GUI:

1. Open a mapping in MapForce e.g. **myMapping.mfd**.
2. Select the menu option **File | 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 at that location.

For information on how to run the MapForce Server Execution file, see the Run command of the MapForce Server documentation.

Note: To successfully execute the "run" command on MapForce Server, the **Limit to single thread execution** check box in the "Server Management" tab of Altova LicenseServer must be **inactive**. If the Run command executed successfully in MapForce Server, the Bulk Insert command will have inserted the specified records.

7.2.5 Database relationships - preserve / discard

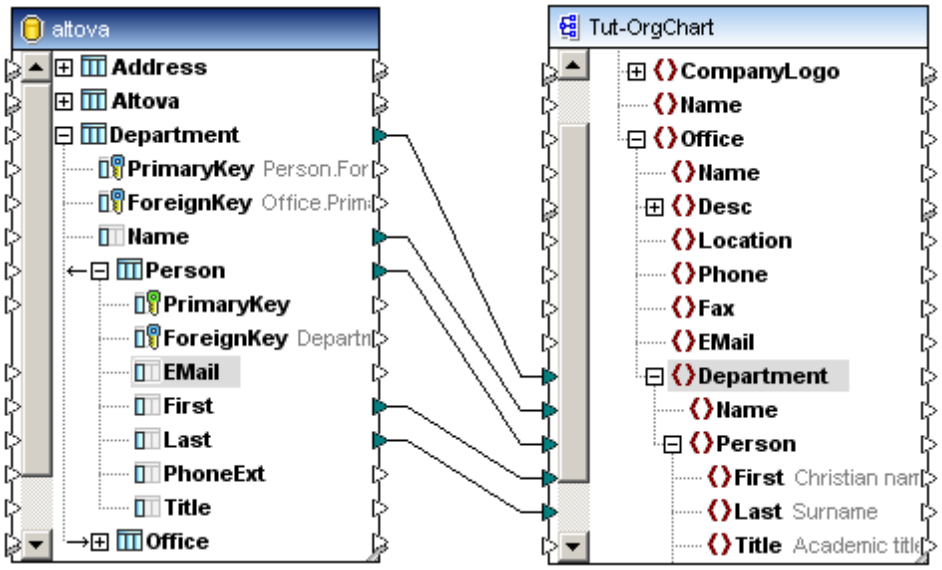
The following section describes how to keep database relationships between tables intact, or how to discard those table relationships.

Maintaining/Using database relationships

To maintain the relationship between database tables, you need to create mappings, between the various fields, below **one** of the "**root**" tables. (e.g. Department). The complete database structure is shown hierarchically below each of the "root" tables in the database component.

See Components and table relationships for more information on the hierarchical table structure of database components.

- **Department | Name** is mapped from under the **Department** "root" table.
- **Person | First and Last** are mapped from the **Person** table which is below the **Department** "root" table in the table hierarchy.



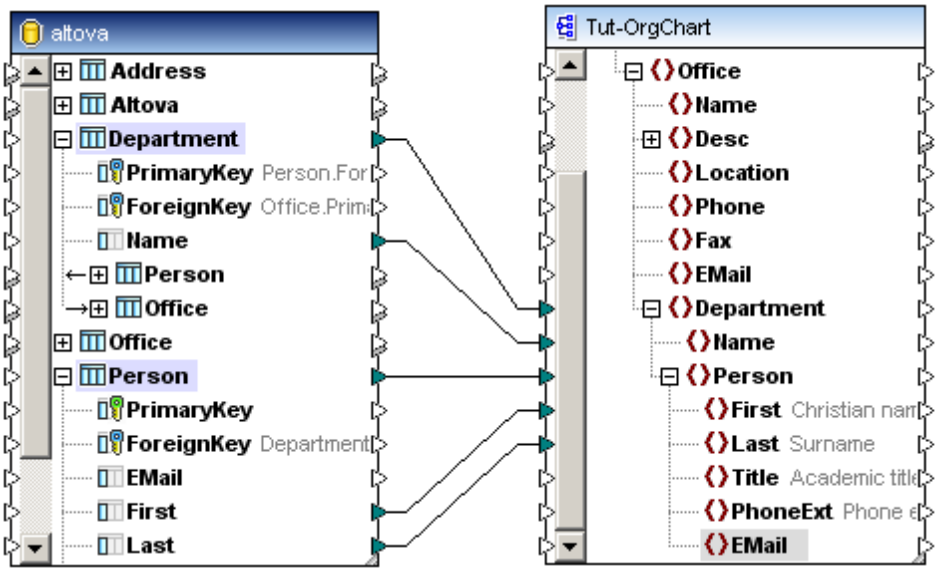
Result of the above mapping:
The names of the persons in each of the departments is shown in the output component.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <OrgChart xsi:noNamespaceSchemaLocation="
   C:/DOCUME~1/MYDOCU~1/Altova/MapForce2010/MapForceExamples
   http://www.w3.org/2001/XMLSchema-instance">
3    <Office>
4      <Department>
5        <Name>Administration</Name>
6        <Person>
7          <First>Vernon</First>
8          <Last>Callaby</Last>
9        </Person>
10       <Person>
11         <First>Frank</First>
12         <Last>Further</Last>
13       </Person>
14       <Person>
15         <First>Loby</First>
16         <Last>Matise</Last>
17       </Person>
```

Discarding database relationships

This method requires that you create mappings between the various fields, beneath separate "**root**" tables of the database component.

- Department | **Name** is mapped from the **Department** "root" table.
- **Person | First and Last** are mapped from the **Person** "root" table.



Result of the above mapping:

The result outputs the Person names of all 21 persons for each, and every, department. The table relationships have been ignored, for each department all persons are output.

Administration department:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <OrgChart xsi:noNamespaceSchemaLocation="
    C:/DOCUME~1/MYDOCU~1/Altova/MapForce2010/MapForceExamples
    http://www.w3.org/2001/XMLSchema-instance">
3     <Office>
4       <Department>
5         <Name>Administration</Name>
6         <Person>
7           <First>Vernon</First>
8           <Last>Callaby</Last>
9         </Person>
10        <Person>
11          <First>Frank</First>
12          <Last>Further</Last>
13        </Person>
14        <Person>
15          <First>Loby</First>
16          <Last>Matise</Last>
17        </Person>
```

Marketing department:

```
91        <Department>
92          <Name>Marketing</Name>
93          <Person>
94            <First>Vernon</First>
95            <Last>Callaby</Last>
96          </Person>
97          <Person>
98            <First>Frank</First>
99            <Last>Further</Last>
100         </Person>
```

7.2.6 Local Relations - creating database relationships

MapForce allows you to query or create related database data, even if no such relationships explicitly exist in the database. You can define the primary/foreign key relations between tables in a component, without affecting the underlying database relationships in any way.

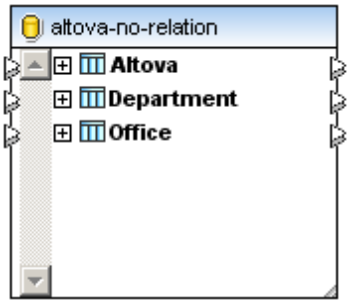
These on-the-fly relationships are called **Local Relations** in MapForce.

- any database fields can be used as primary or foreign keys
- new relations can be created that do not currently exist in the database

Local relations can be defined for:

- Database tables
- Database views
- Stored procedures and functions
- User-defined SELECT statements

The MS Access **altova-no-relation.mdb** database used in this example, is a simplified version of the Altova.mdb database supplied with MapForce. The Person and Address tables, as well as all remaining table relationships have been removed in MS Access.



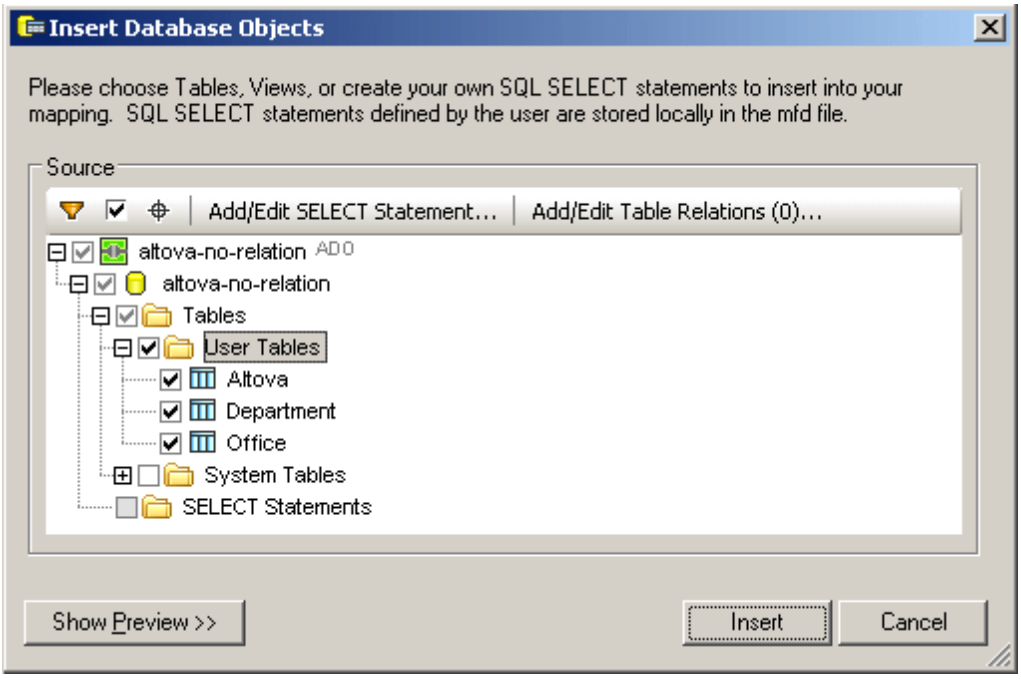
The aim of this example is to display the offices of Altova and show the departments in each.

None of the tables visible in the **altova-no-relation** tree have any child tables, all tables are on the same "**root**" level. The content of each table is limited to the fields it contains. We can however, use MapForce to extract related database data, even though relationships have not been explicitly defined.

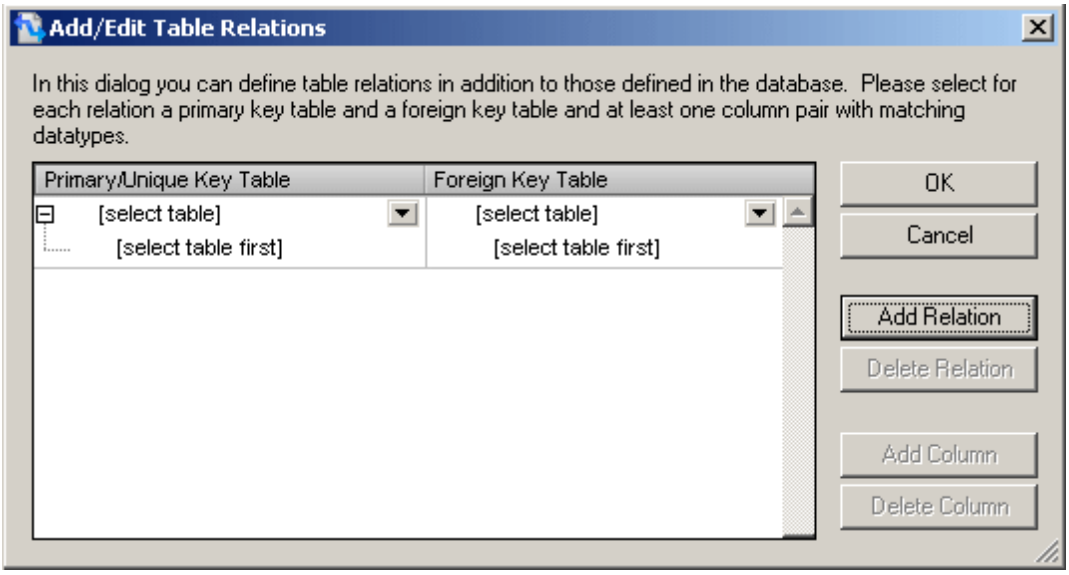
To create local relations:

Local relations can be defined while inserting a database, or by right clicking an existing database component and selecting the Add/Remove Tables from the context menu.

1. Insert the **altova-no-relation** database.
2. In the connection wizard click Microsoft Access, then click Next.
3. Click Browse, select Altova.mdb then click the **User Tables** checkbox to select all three tables.

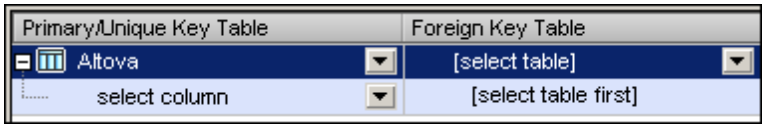


4. Click the **Add/Edit Relations** button in the icon bar.
   This opens the Add/Edit Relations dialog box.
5. Click the **Add Relation** button.

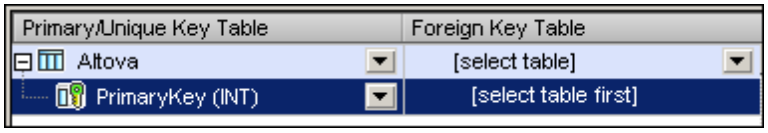


The two combo boxes allow you to select the tables or database objects you want to create relations for. The left combo box is the Primary/Unique Key Table, the right one, the Foreign Key Table. The Primary/Unique Key object will be the parent object in MapForce, and the Foreign Key object will be shown as child in the database component.

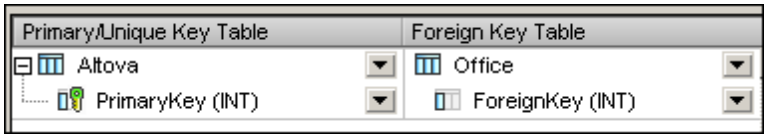
6. Click the left combo box and select the **Altova** table.



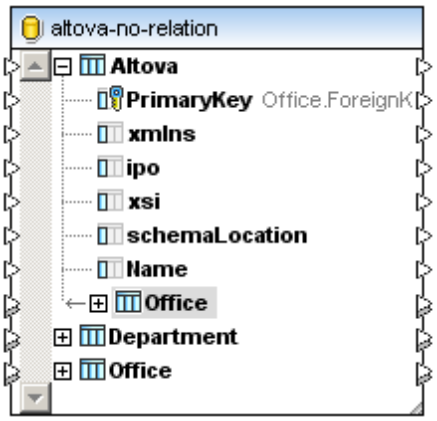
7. Click the "select column" combo box below it, and select the **Primary Key** entry.



8. Select the **Office** table and **ForeignKey** column for the Foreign Key table.



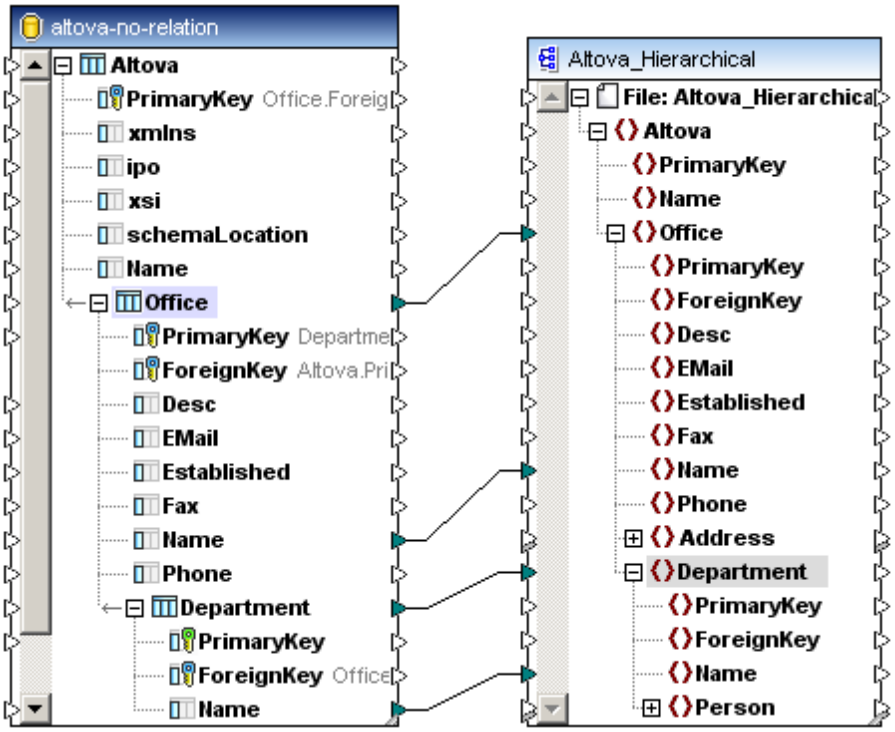
9. Click the OK button to complete the local relation defintion, then click the **OK** button to insert the database into the mapping area.



Clicking the expand icon of the Altova table shows that there is a relationship between the Altova and Office tables. The Office table is shown as a related table below the Altova table with its own expand icon.
Use the same method to create a relationship between the **Office** and **Department** tables.

10. **Right click** the database component and select **Add/Remove Tables** from the context menu, then click the Add/Edit Table Relations button in the icon bar.

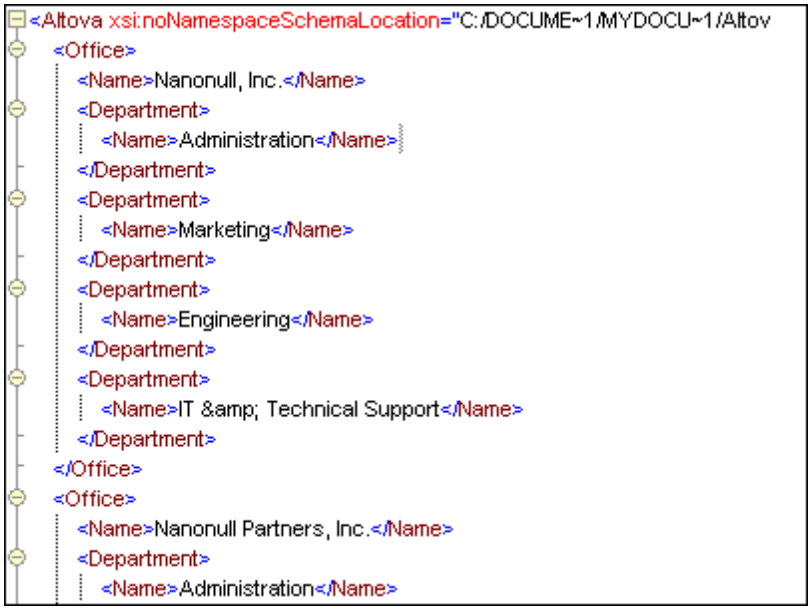
When creating the mapping it is important to remember that to preserve relationships between tables, connectors below **one** of the "root" tables must be used, i.e. Altova in this case.



Having defined the mapping as shown above, click the Output tab, to preview the result immediately. Database data cannot be previewed if the target language is XSLT,XSLT2, or XQuery; a message will appear and the database component will be greyed out.

The mapping result shows:

- "for each Office element, output the office name and then all departments in that office"



7.2.7 Mapping large databases with MapForce

When using databases with many tables in mappings, MapForce displays all database relations between the imported tables, of the whole database. This is due to the fact that the application cannot automatically decide which tables will be used in the mapping process, all possibilities have to be covered.

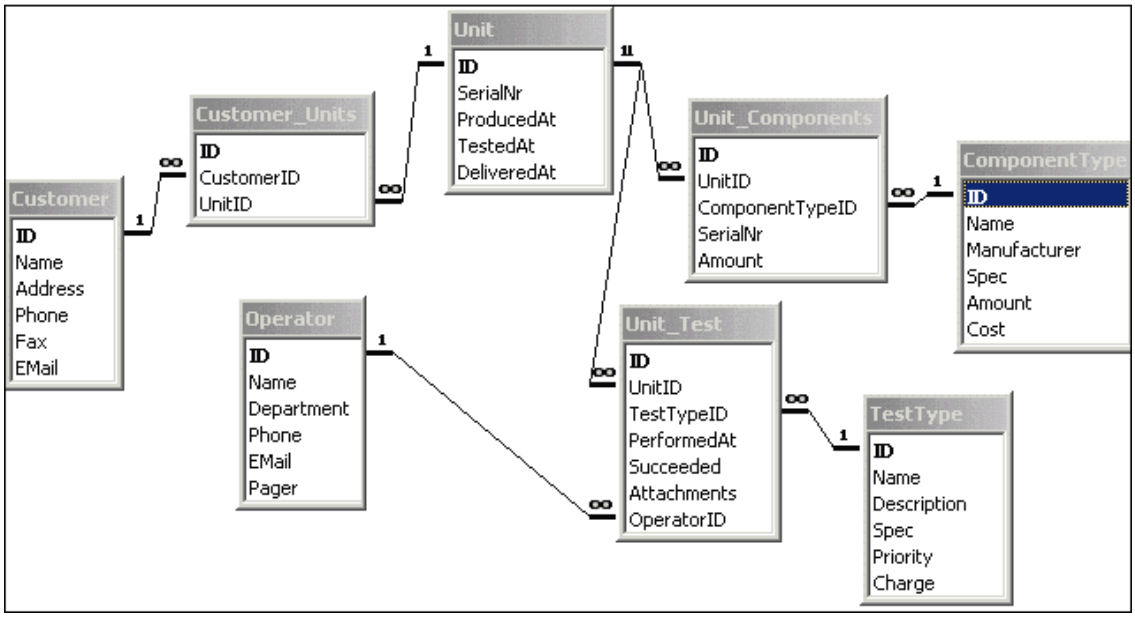
This may lead to inconveniently large tree structures if all tables are selected in a single database component. It is however possible to create multiple database components, of the same database, which only use/import those tables that are needed for the mapping process. This method also makes for a more intuitive mapping.

E.g.
In a production company, various components are assembled to produce customer defined units. Before delivery, the units undergo a unit test and all results are stored in a database.

At some point during the prototype testing phase, it is discovered that a batch of components are faulty, and a recall has to be initiated. The goal of the mapping is to generate a list of all affected customers to whom a letter must be sent.

In this case the mapping defines:
For the ComponentType name = "Prototype" AND the Manufacturer = "Noname",
Select all related Customers and their requisite details.

The relationship diagram of the example database discussed in this section, is shown below:



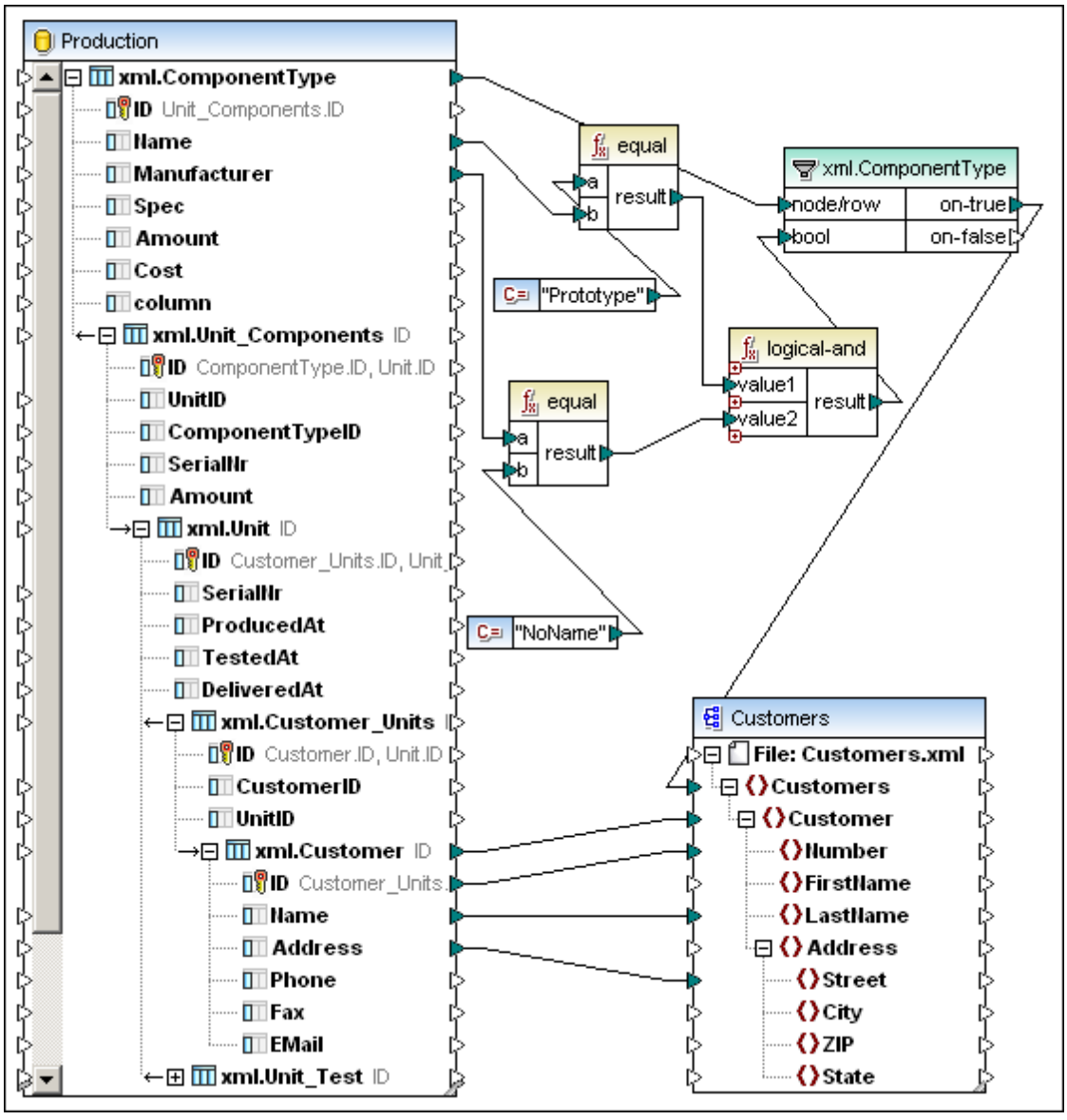
7.2.7.1 Complete database import

Option 1:
Import the complete database i.e. with **all** the tables it contains. The Production database component therefore contains all tables, with each table appearing as a "root" table along with all

its related tables.

Using the ComponentType table as the root table: the mappings filter by:

- the Component Name "Prototype" AND
- the Manufacturer "NoName", along with
- the related Customer ID and address data



7.2.7.2 *Partial database import*

Option 2:
Import only those tables that are necessary to extract the necessary information i.e.:

- retrieve all defective units
- retrieve all customers to whom these units were supplied

Insert two database components, from the same database, importing different sets of tables

Component 1, insert the following tables:

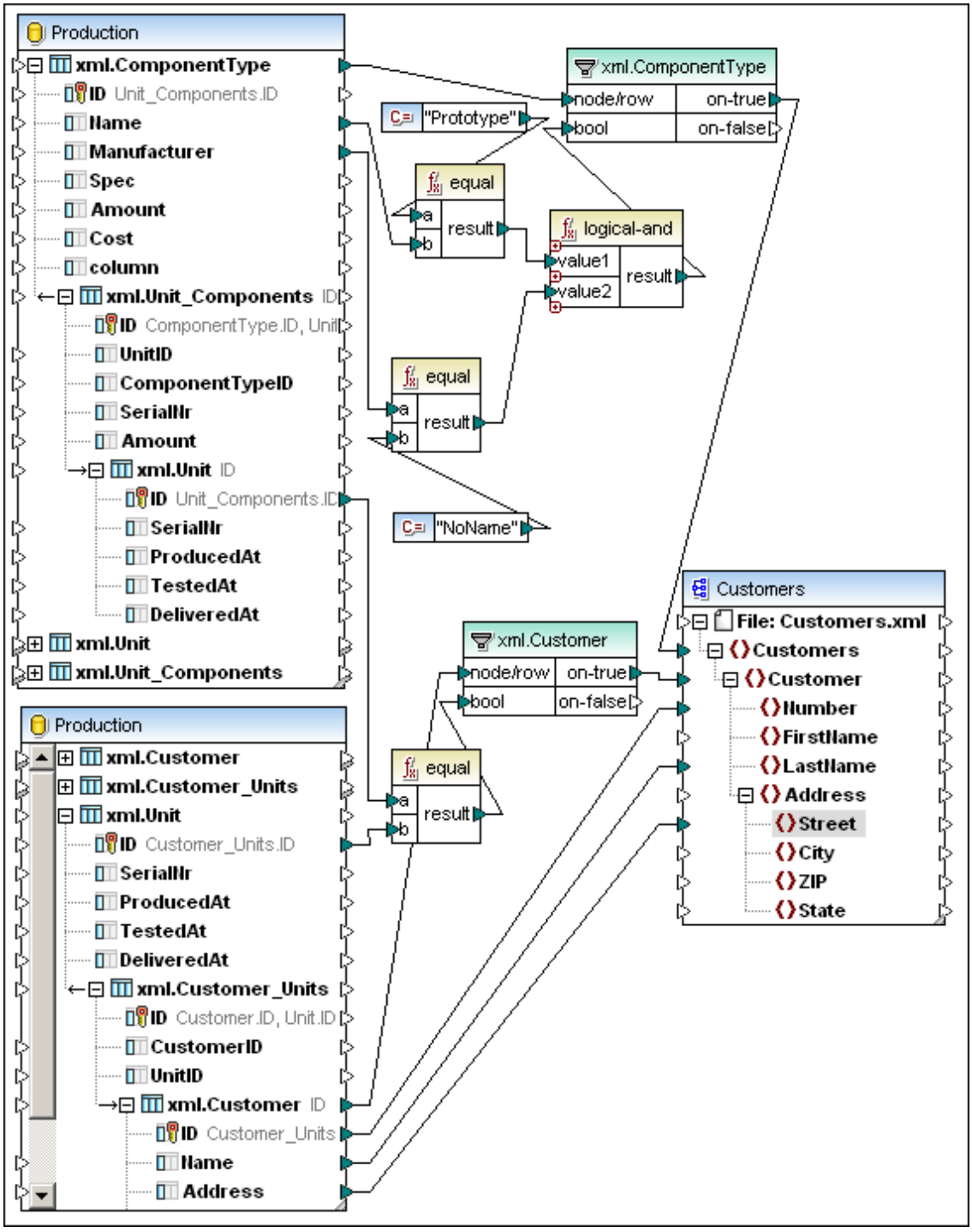
- ComponentType
- Unit_Components
- Unit

Component 2, insert:

- Unit
- Customer_Units
- Customer

Mapping process:

- filter by the Component Name "Prototype" AND the Manufacturer "NoName" (component 1)
- use the "equal" function compare the unit ID from component 1 with the unit ID from component 2
- if the IDs are equal, use the filter component to pass on the associated customer data from component 2 to the Customers XML file.



7.2.8 SQL WHERE / ORDER Component

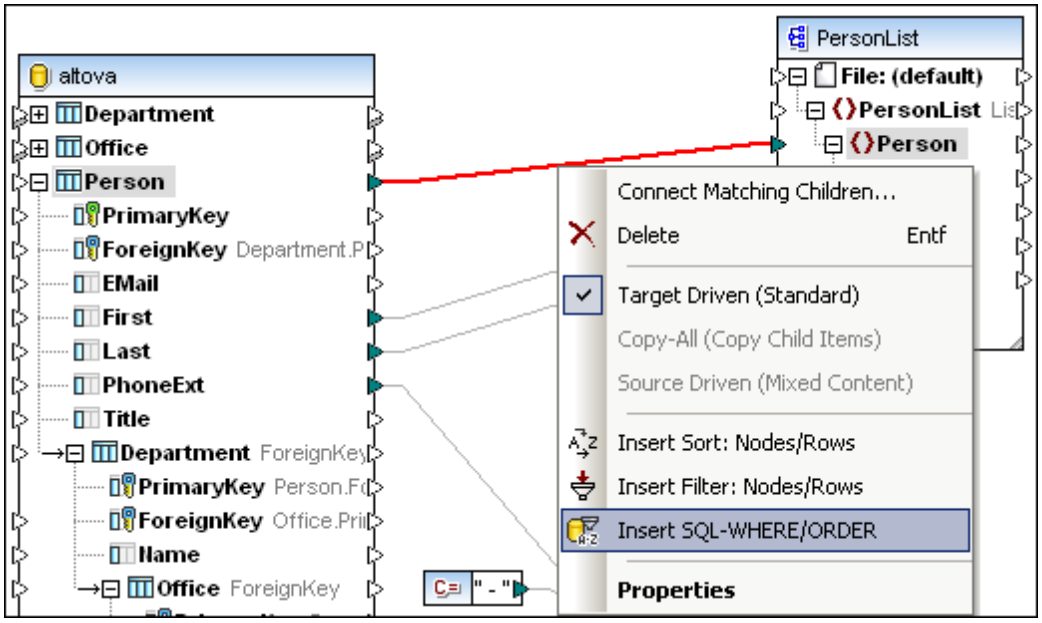
MapForce allows you to filter and sort database data using the SQL WHERE/ORDER component. The example discussed here is available as **DB_PhoneList.mfd** in the ...\MapForceExamples folder.

The SQL WHERE/ORDER component is comprised of several parts:

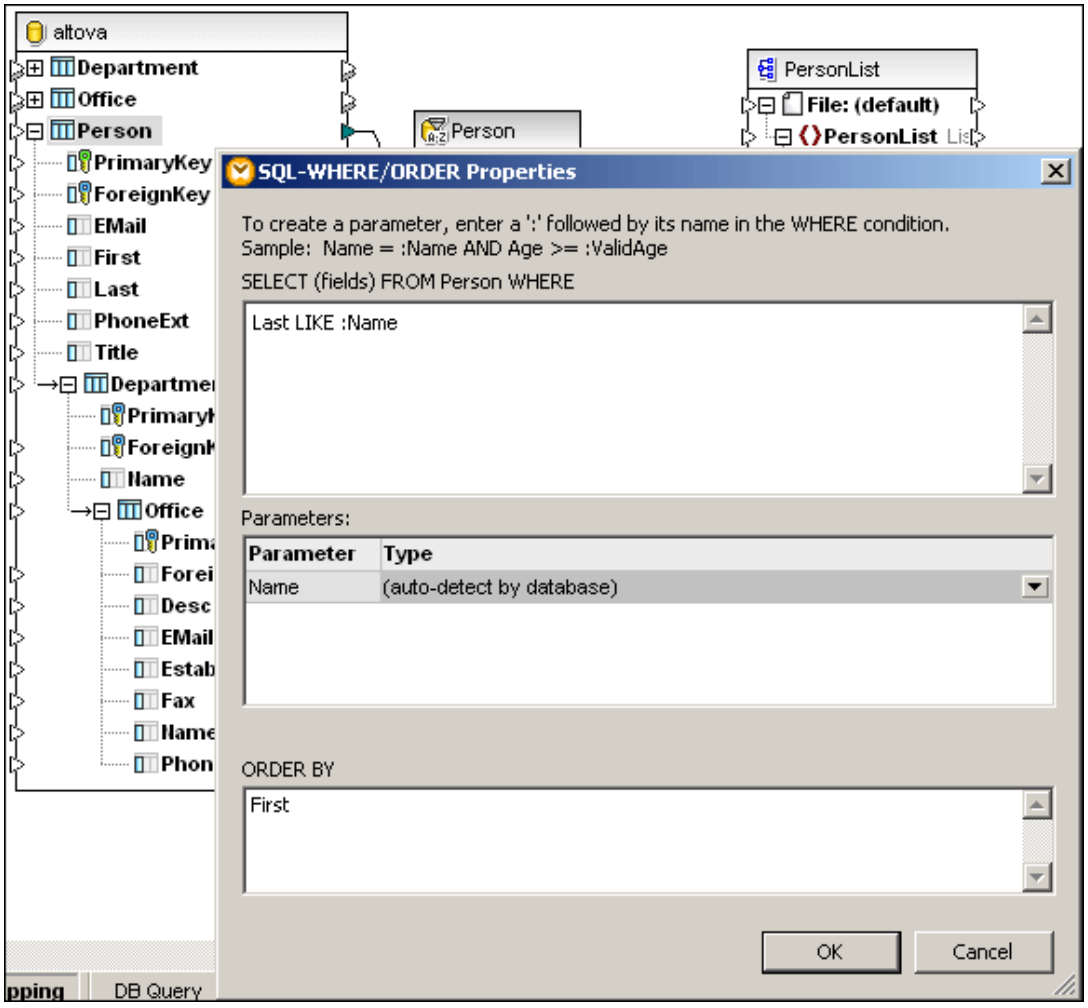
- The **SELECT** statement that is automatically generated when you connect to a database table or field.
- The **WHERE** clause that you manually enter in the SELECT text box. Note that the foreign keys are automatically included in the WHERE condition.
- The **ORDER BY** clause

To insert an SQL WHERE/ORDER component:

1. Right-click a connector that exists between a table and the target node.



2. Click the Insert SQL-WHERE/ORDER item from the context menu. This opens the SQL-WHERE/ORDER Properties dialog box.
3. Click in the top field to write the WHERE query in the text box. The SELECT statement, just above the text box, is automatically generated for you when a connection is made from a table, or field, to the **table/field** input icon of the SQL WHERE/ORDER component.

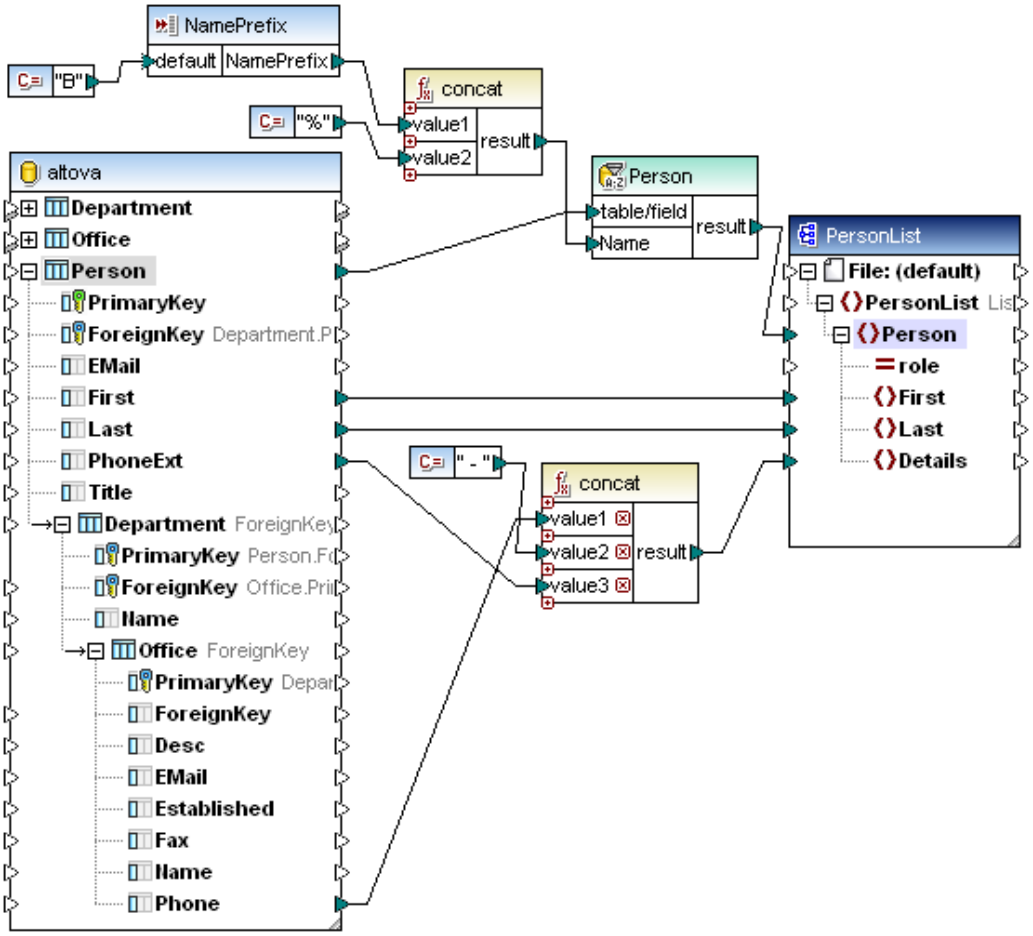


The WHERE statement in the SELECT text box, defines a parameter called "Name", which uses the LIKE keyword to find a pattern in the "Last" field of the Person table. In this case a constant component supplies the search string, **B**, which results in all persons being found whose last name starts with a "B".

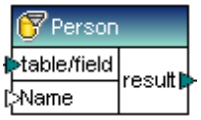
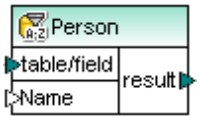
The wildcard character "%" (in the constant component) denotes any number of characters.

The Parameters pane allows you to define the datatype of a parameter defined in the query (a warning message is displayed in the dialog box while the component is not connected to a database).

4. Enter "First" in the ORDER BY pane to have the resulting records sorted by the person first name - First.



The appearance of the SQL-WHERE/ORDER component provides information about its contents, for example:

	A WHERE clause has been defined, for example: `First > "C" AND Last > "C"`
	A WHERE clause with a parameter has been defined, for example: `Last LIKE :Name` The parameter "Name" is visible under the "table/field" parameter.
	A WHERE clause with a parameter has been defined. Additionally, an ORDER BY clause has been defined. The sorting is indicated by the A-Z sort icon.

Placing the mouse cursor over the SQL WHERE/ORDER header, opens a screen tip displaying the various clauses that have been defined.

All persons whose last name begins with "B" are now sorted by their first name in the Output tab.

```
<Person>
  <First>Jessica</First>
  <Last>Bander</Last>
  <Details>+1 (321) 555 5155 - 241</Details>
</Person>
<Person>
  <First>Michelle</First>
  <Last>Butler</Last>
  <Details>+1 (321) 555 5155 - 654</Details>
</Person>
<Person>
  <First>Theo</First>
  <Last>Bone</Last>
  <Details>+1 (927) 555 0094 - 331</Details>
</Person>
```

7.2.8.1 SQL WHERE / ORDER operators

The following operators are supported within the WHERE component:

Operator	Description
=	Equal
<>	Not equal
<	Less that
>	Greater than
>=	Greater than/equal
<=	Less than/equal
IN	Retrieves a known value of a column
LIKE	Searches for a specific pattern
BETWEEN	Searches between a range

Wildcards:

The **%** wildcard is used to define any number of characters in a pattern (equivalent to * in other programs) e.g. %r retrieves all records ending in "r".

XML Data:

XQuery commands are also supported when querying databases that support storing and querying of XML database data (for example, IBM DB2, Oracle, SQL Server).

E.g. **xmlexists**('$c/Client/Address[zip>"55116"]' passing USER.CLIENTS.CONTACTINFO AS "c")

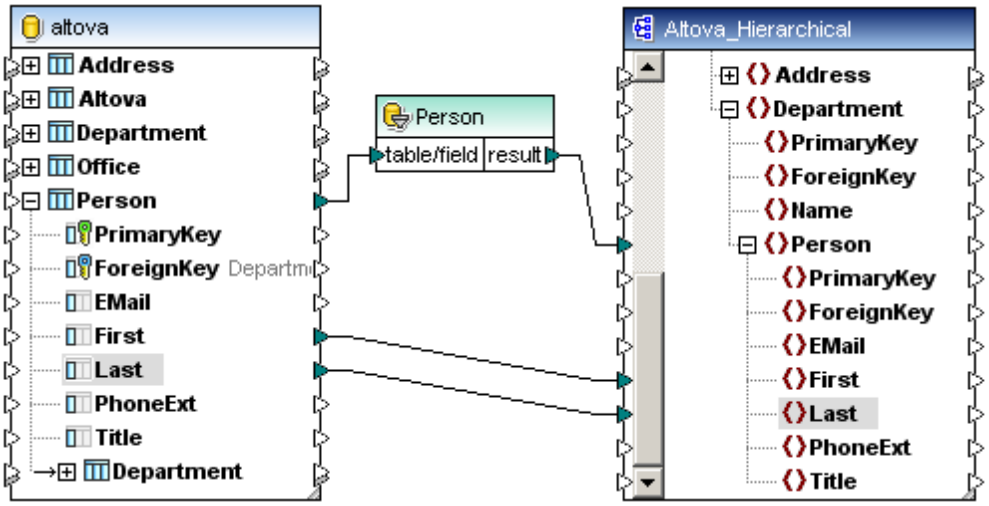
Examples

The following WHERE condition is attached to the `Person` table of the **altova.mdb** database component. It retrieves those records where `First` and `Last` are greater than the letter "C". In other words, it retrieves all names from "Callaby" onwards.

```
First > "C" AND Last > "C"
```

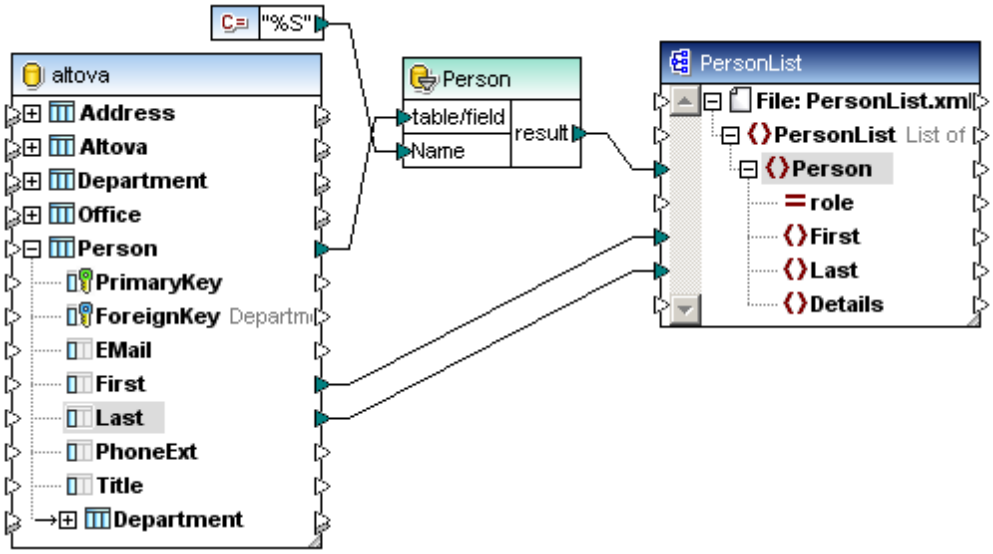
Note how the connectors are placed:

- The connector to the **table/field** parameter is connected to the table that you want to query, "Person" in this case.
- The **result** parameter is connected to a "parent" item of the fields that are queried/filtered, in this case the Person item. The first and last fields are connected to sub-items in the target component for them to appear in the result.



The following WHERE condition creates a parameter **Name** which appears as a parameter in the SQL WHERE/ORDER component. In this case it is used to search for a pattern in the column "Last".

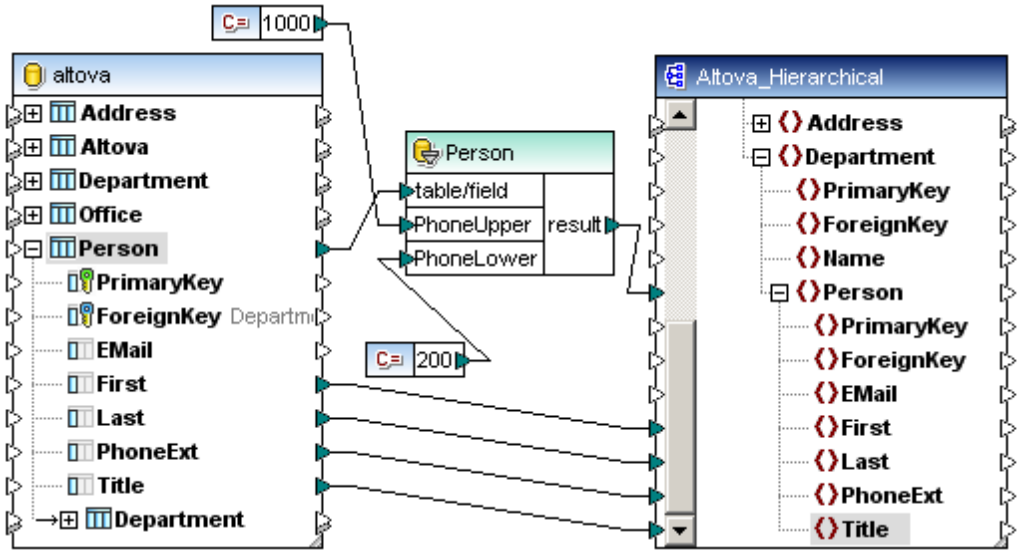
```
Last LIKE :Name
```



The Constant component supplies the values needed to search for all Last names ending in "S", i.e. %S. The wildcard % denotes any number of characters.

The following WHERE condition creates two parameters, `PhoneUpper` and `PhoneLower`, to which the current values of `PhoneExt` are compared. The upper and lower values are supplied by two constant components shown in the diagram below.

```
PhoneExt < :PhoneUpper and PhoneExt > :PhoneLower
```



The WHERE condition in this example could also be written using the BETWEEN operator:

```
PhoneExt BETWEEN :PhoneUpper and :PhoneLower
```


7.2.9 IBM DB2 - Mapping XML data to / from databases

MapForce supports the mapping of XML data to / from IBM DB2 databases. The examples in this section assume that you have access to an IBM DB2 database; all other necessary files are supplied in the ..**Tutorial** folder.

For the XML nodes to appear in the database component, which allows mapping to and from the XML items, a schema must be **assigned**. Once this is done the mapping process proceeds as usual.

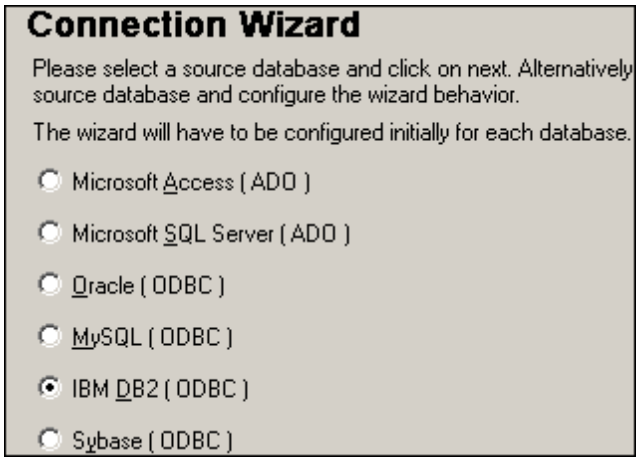
MapForce currently supports the assigning of **one** XML Schema per database column, and the selection of a **single** "root" element of that schema.

Please note:
When adding an ODBC Data Source for the IBM iSeries (formerly AS/400), a default flag is set which enables query timeouts. This setting must be **disabled** for MapForce to correctly load mapping files.

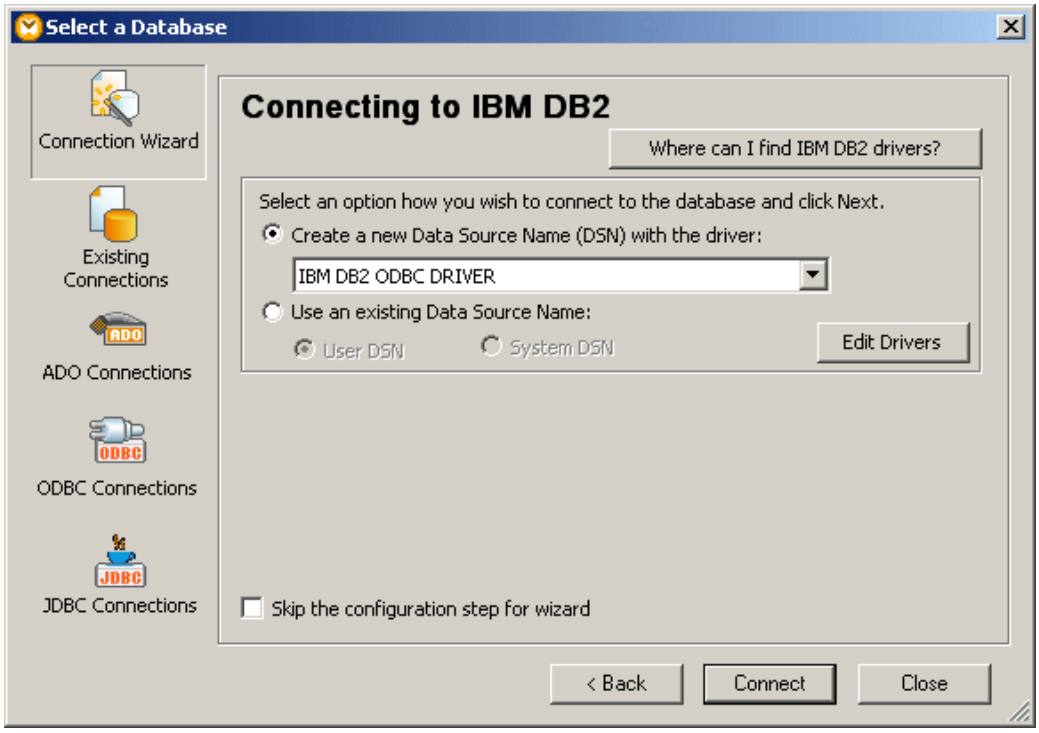
When adding an ODBC data source for iSeries Access ODBC driver, the "iSeries Access for Windows ODBC Setup" dialog box is opened. Select the "Performance" tab, click the "Advanced" button and **uncheck** the "Allow query timeout" check box option.

Configuring and inserting an IBM DB2 database:

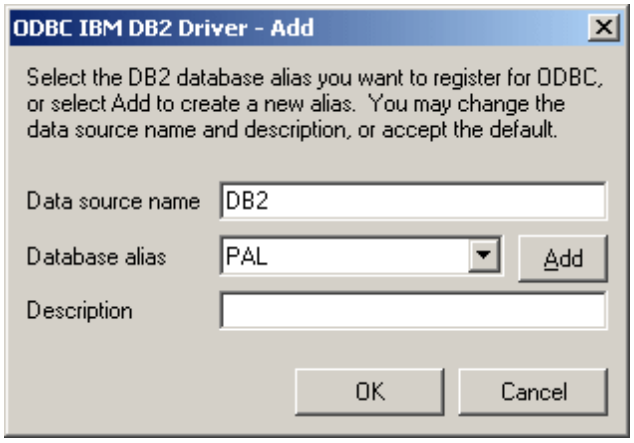
1. Click one of the icons in the icon bar: Java, C#, C++, or BUILTIN.
2. Click the **Insert Database** icon  in the icon bar.



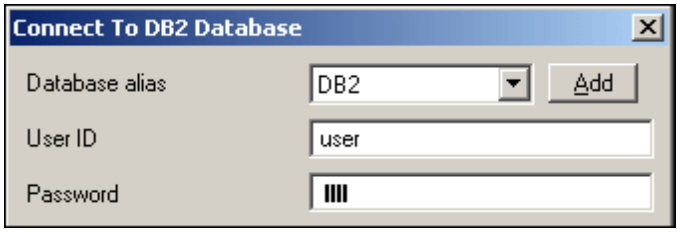
3. Click the IBM DB2 (ODBC) radio button and click Next.
4. Select a driver from the Driver list, by selecting the appropriate entry from the combo box.



5. Click **Next** to define a new Data Source Name (DSN).
6. Enter the Data source name, select (or Add) the Database alias, then click OK.



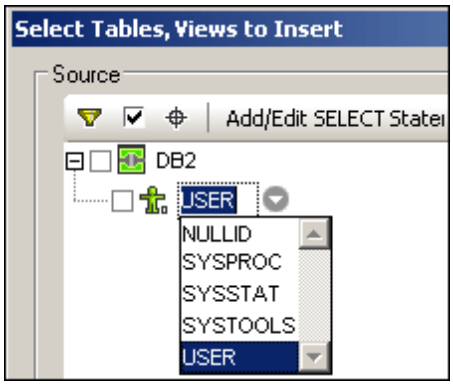
7. Enter the login data and click OK to connect to the database.



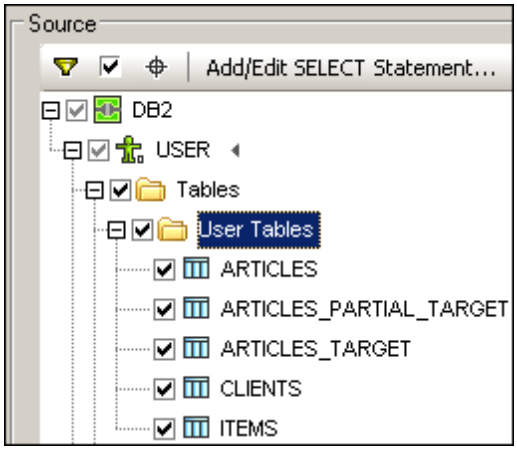
Entering the required login info and clicking OK opens a further dialog box allowing you to choose how the datasource will communicate with the server; natively or via the ODBC API.

This opens the Insert Database Objects dialog box.

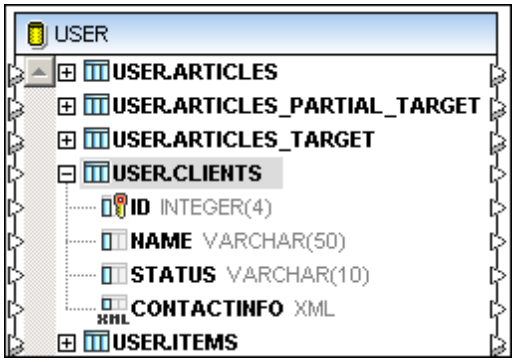
8. Click the database schema icon and select the correct database schema e.g. USER.



All USER tables are now visible.

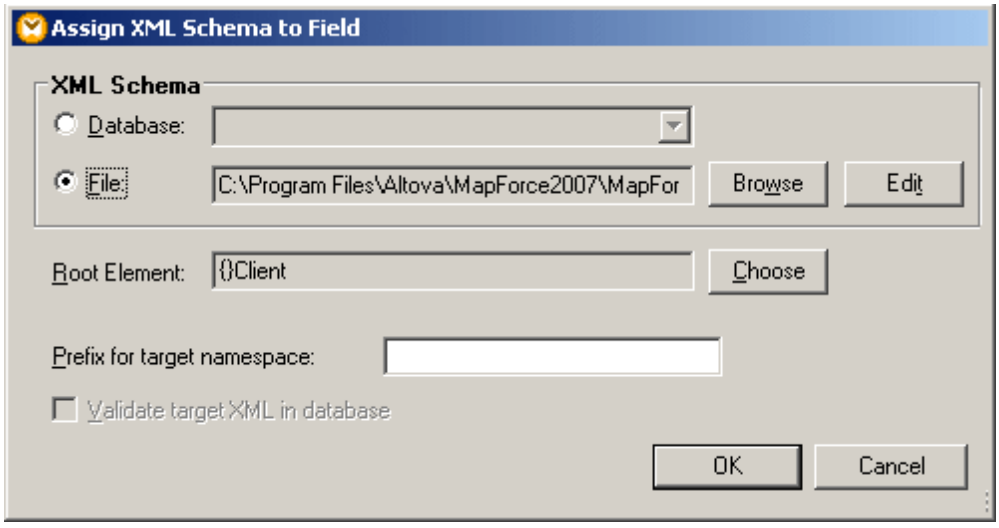


9. Click the check box next to "User Tables" to select all child tables, then click OK to insert the database component.
10. Click the expand icon next to the USER.CLIENTS table to see its contents.
Note that the CONTACTINFO column is of type **XML**.

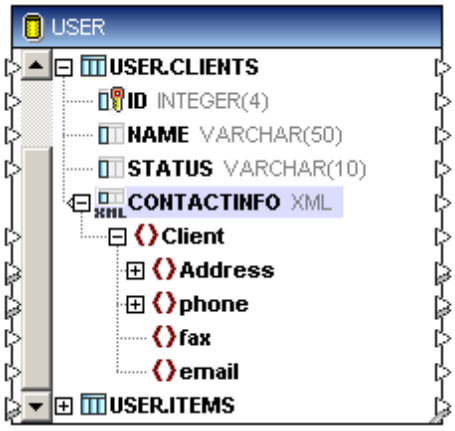


Assigning an XML Schema to an XML file:

1. Right click the CONTACTINFO column, under the USER.CLIENTS table, and select "**Assign XML Schema to field...**".
2. Clicking the **Database** radio button allows you to select from schemas that have been saved (registered) in the **database**, while **File** allows you to select a local one. The XML Schema **DB2Client.xsd** available in the ...**Tutorial** folder was selected for this example.

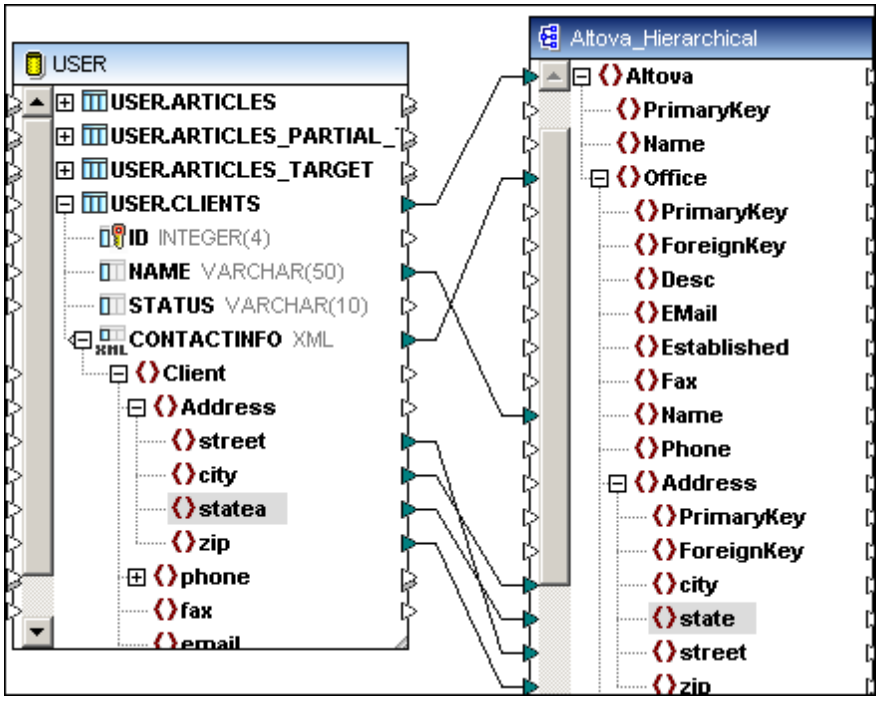


3. Choose the Root element of the schema that is to appear in the component e.g. Client, and click OK to confirm.

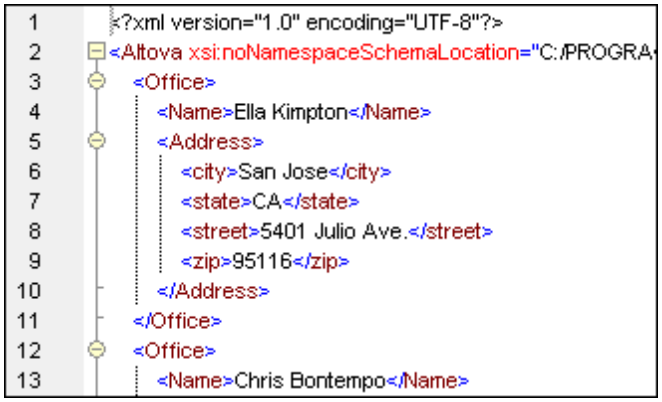


The "Client" item appears below CONTACTINFO; click the expand icons to see the schema structure.

4. Map connectors from the schema items to the target component, which is an XML document in this case.

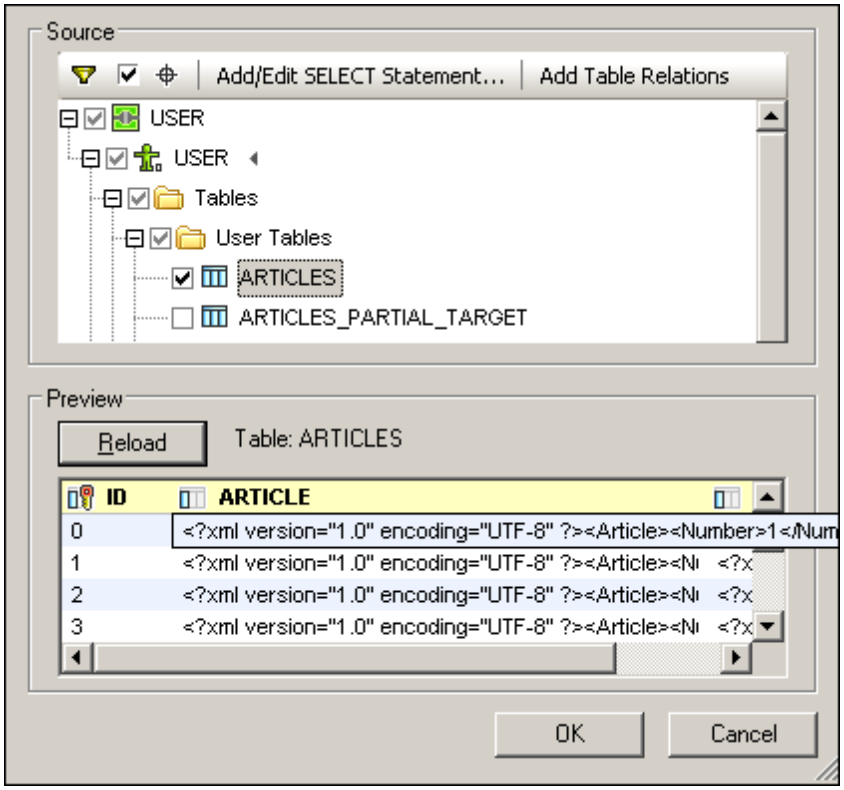


5. Click the Output button to see the result of the mapping.



Previewing table content

Clicking the Preview button, while the **Select Tables / Views to insert** dialog box is open in the connection wizard, displays the table data in the preview window. Clicking the Preview button changes it to Reload. If a table column is of type XML, then placing the mouse cursor over the XML column opens a popup displaying the XML content.



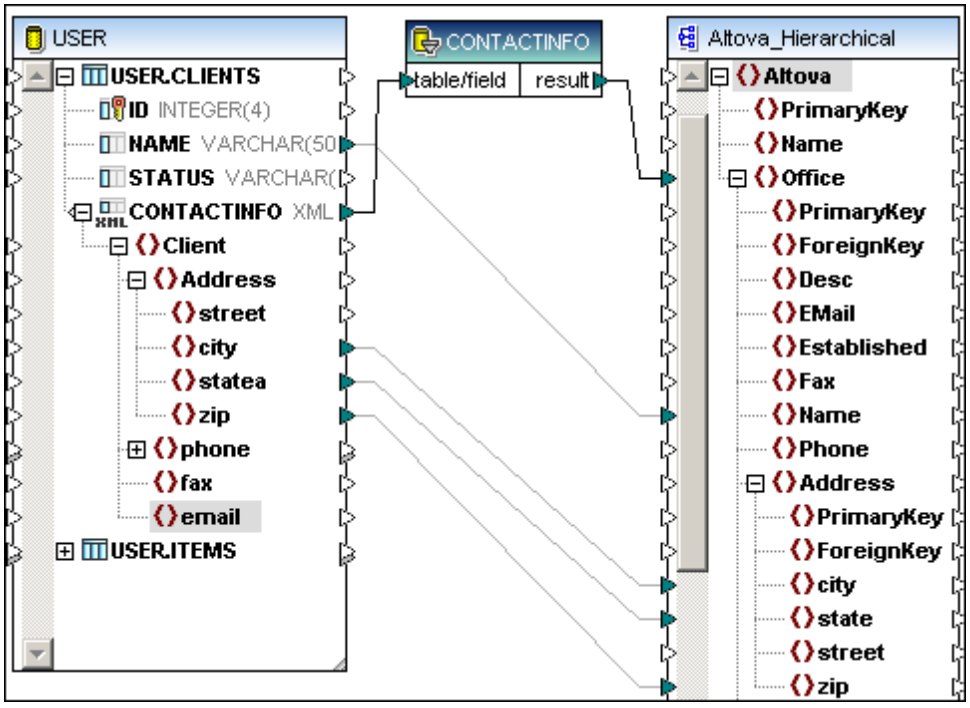
7.2.9.1 Querying and mapping XML data in IBM DB2

MapForce allows you to query XML data in IBM DB2 databases using the SQL WHERE component and map the record set data to other components. Please see SQL WHERE Component / condition for more information on how to insert and use the SQL WHERE component. This section discusses how to query, and map, XML data from an IBM DB2 database.

Please note that you can also query databases directly using the Database Query tab, please see Querying databases directly - Database Query tab for more information.

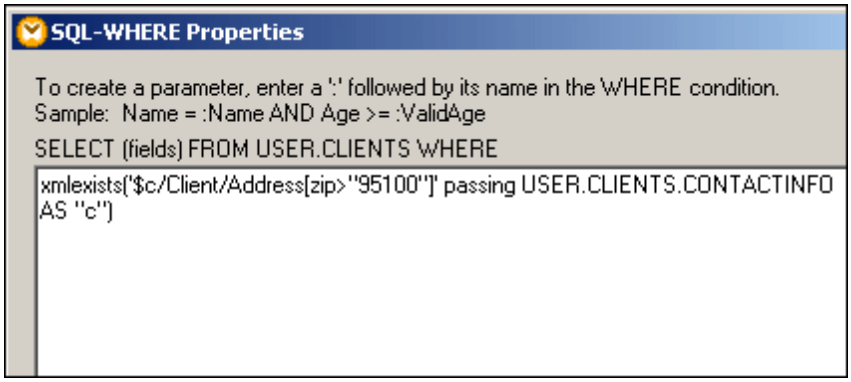
Having inserted the DB2 database and assigned the XML schema to the CONTACTINFO item:

1. Click the SQL WHERE icon in the icon bar to insert it.
2. Connect the **CONTACTINFO** item, of the database source, to the **table** item of the component.
3. Connect the **result** item to an item in the target component e.g. Office.



This updates the **name** of the SQL WHERE component to CONTACTINFO.

4. Double click the CONTACTINFO component to create the query.
5. Enter the SQL/XML WHERE query e.g. **xmlexists('$c/Client/Address[zip>"95100"]' passing USER.CLIENTS.CONTACTINFO AS "c")**



Note that the first part of the Select statement, SELECT (fields) FROM USER CLIENTS WHERE, is automatically generated for you when you connect the input and output connectors to the database and target component.

This query outputs those records where the zip code in the XML file is greater than 95100.

The **xmlexists** function allows you to navigate an XML document using an XPath expression, e.g. '$c**/Client/Address[zip>"95100"]**', and test a condition. For more information on SQL/XML functions please see the DB2 Information Centre web page.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Altova xsi:noNamespaceSchemaLocation="C:/PROGRA~1/Altova
3    <Office>
4      <Name>Ella Kimpton</Name>
5      <Address>
6        <city>San Jose</city>
7        <state>CA</state>
8        <zip>95116</zip>
9      </Address>
10   </Office>
11   <Office>
12     <Name>Chris Bontempo</Name>
13     <Address>
14       <city>San Jose</city>
15       <zip>95124</zip>
16     </Address>
17   </Office>
```

7.2.9.2 *Mapping XML data - IBM DB2 as target*

This section discusses how to map XML data to a target XML document stored in an IBM DB2 database. The example assumes that you have access to an IBM DB2 database; all other necessary files are available in the ...**Tutorial** folder.

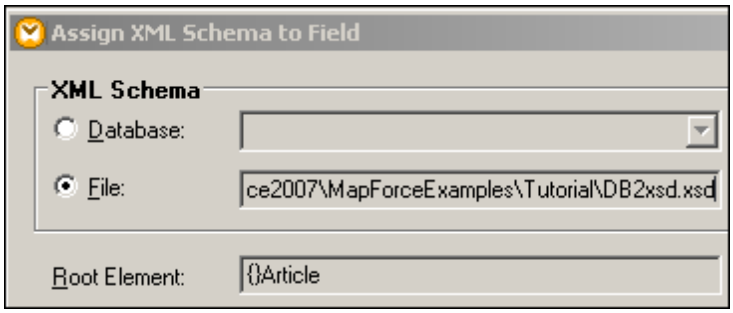
Please note that you can also query databases directly using the Database Query tab, please see Querying databases directly - Database Query tab for more information.

To insert the data source component:

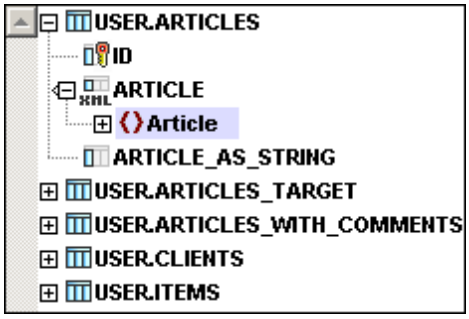
1. Click the **Insert XML Schema/File** icon , and select the **DB2asTarget.xsd** schema.
2. Click Browse when the prompt for a sample XML file appears, and select **DB2asTarget.xml**
3. Select **Articles** as the root element and expand it.

To insert the database target and map data to it:

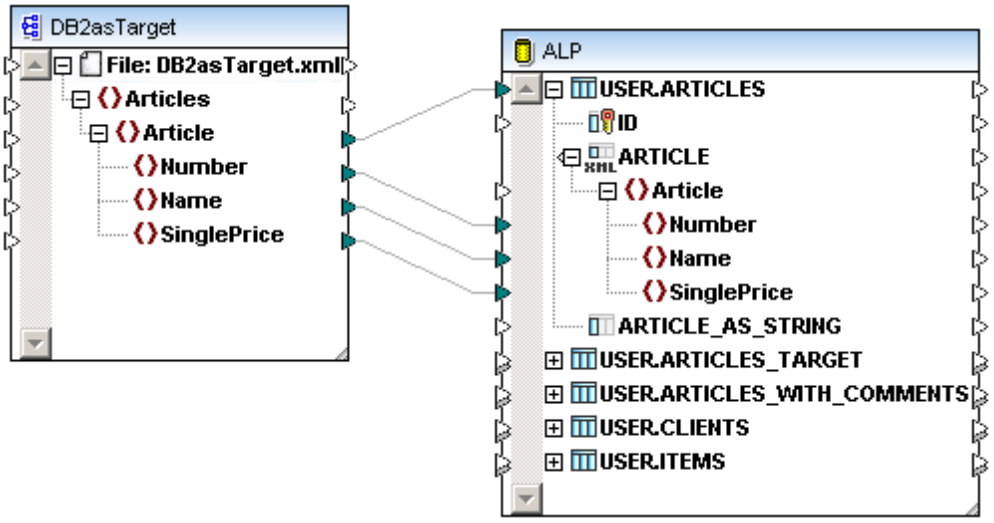
1. Using the method discussed in insert the IBM DB2 database, to insert the database.
2. Right click the ARTICLE item/column in the database target and select **Assign XML Schema to Field...**
3. Click the **File** radio button, select **DB2xsd.xsd** and click OK.



4. Expand the **Article** item to be able to create connectors to the respective items.



5. Create connections between the source and target components as shown in the screenshot below.



6. Click the **Output** tab, then the Word Wrap icon to see more of the SQL script.

```
8   INSERT INTO "USER"."ARTICLES" ("ARTICLE")
9       VALUES ('<?xml version="1.0" encoding="UTF-8"?><Article
    xsi:noNamespaceSchemaLocation="C:\Program
    Files\Altova\MapForce2007\MapForceExamples\Tutorial\DB2xsd.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Number>11</Number><Name>
    sneakers</Name><SinglePrice>99</SinglePrice></Article>')
10
11  INSERT INTO "USER"."ARTICLES" ("ARTICLE")
12      VALUES ('<?xml version="1.0" encoding="UTF-8"?><Article
    xsi:noNamespaceSchemaLocation="C:\Program
    Files\Altova\MapForce2007\MapForceExamples\Tutorial\DB2xsd.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Number>22</Number><Name>
    band</Name><SinglePrice>2.9</SinglePrice></Article>')
```

This gives you a preview of the XML data that will be inserted into the database.

7. Click the "Run SQL-script" icon to insert the data.

```
/*
The following SQL statements were executed during "Generate output" function.
Every single result is written right to the "-->>>" string.
These statements are only for preview and may not be executed in another SQL query tool!
*/

INSERT INTO "USER"."ARTICLES" ("ARTICLE")
    VALUES ('<?xml version="1.0" encoding="UTF-8"?><Article
xsi:noNamespaceSchemaLocation="C:\Program
Files\Altova\MapForce2007\MapForceExamples\Tutorial\DB2xsd.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Number>11</Number><Name>s
ers</Name><SinglePrice>99</SinglePrice></Article>')
-->>> OK. 1 row(s).
```

The output window now shows if the commands were executed successfully.

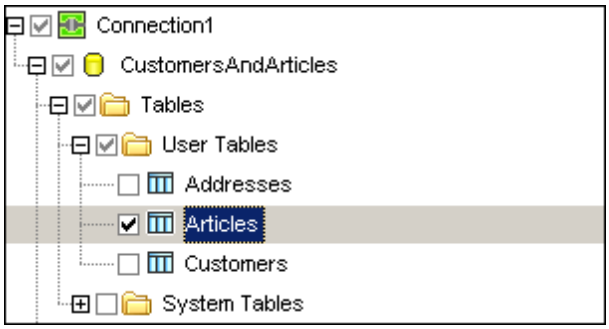
7.2.9.3 Mapping data - database to database

This section discusses how to map XML data from an MS Access database to a target XML document stored in an IBM DB2 database. The example assumes that you have access to an IBM DB2 database; all other necessary files are available in the ...**MapforceExampes**, or ... **MapforceExampes****Tutorial** folder.

Please note that you can also query databases directly using the Database Query tab, please see Querying databases directly - Database Query tab for more information.

To insert the MS Access source database:

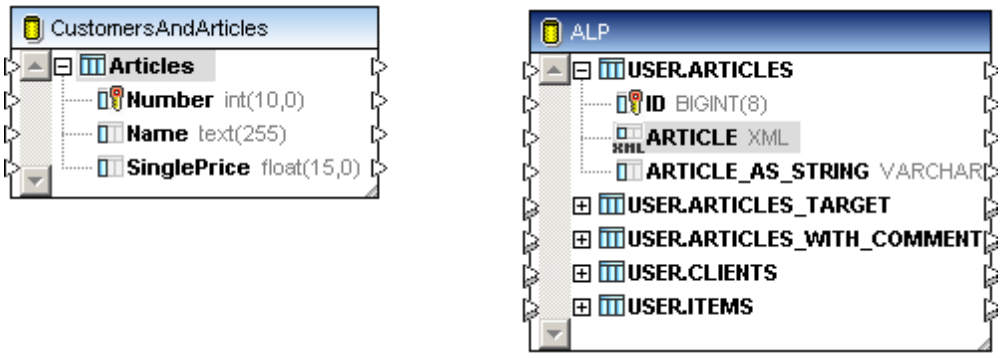
1. Click the **Insert Database** icon , and select the **CustomersAndArticles.mdb** database from the ...\MapForceExamples folder.
2. Select the **Articles** table and click OK to insert.



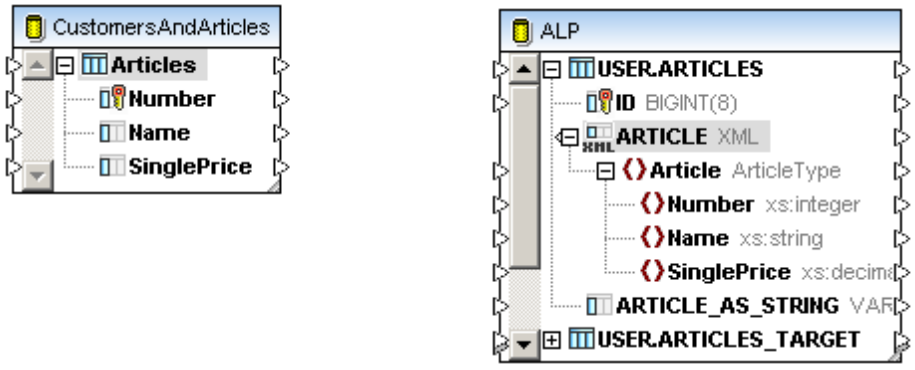
The database table is now visible as a database component.

To insert the IBM DB2 target database and map data to it:

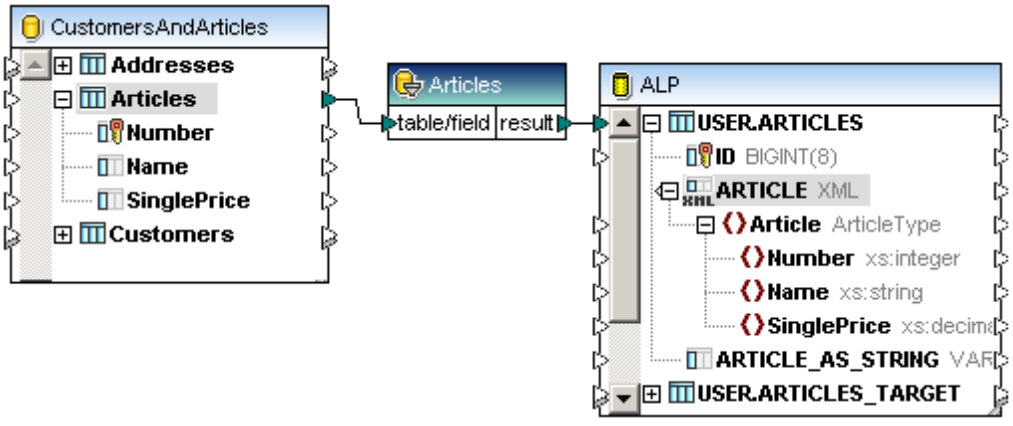
1. Using the method discussed in insert the IBM DB2 database to insert the database.



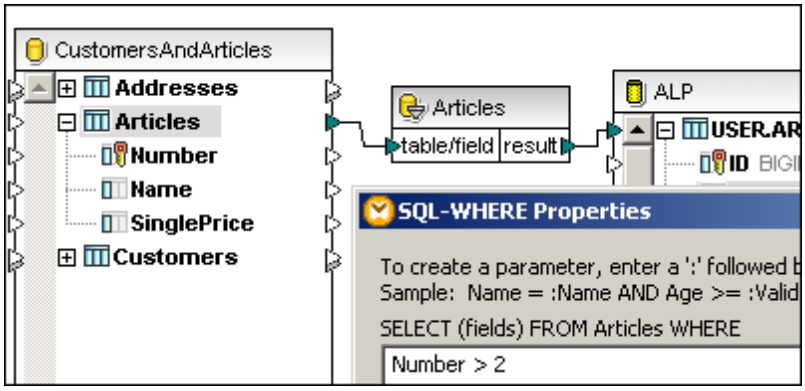
2. Right click the ARTICLE item/column in the database target and select **Assign XML Schema to Field...**
3. Click the **File** radio button, select the **DB2xsd.xsd** schema file, then click OK.



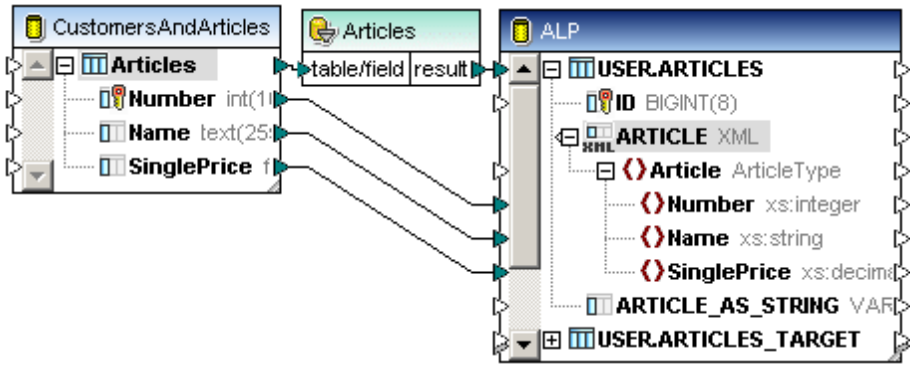
4. Click the SQL WHERE icon  in the icon bar to insert it.
5. Connect the **Articles** item, of the database source, to the **table** item component.
6. Connect the **result** item to the USER.ARTICLES item in the target component.



7. Double click the SQL WHERE **Articles** component, enter **Number > 2** as the Where clause, and click OK.



8. Map the Number, Name and SinglePrice items from the source to the target database.



9. Click the Output tab to see a preview, then click the "Run SQL-script" icon  to insert the data.

```
INSERT INTO "USER"."ARTICLES" ("ARTICLE")
    VALUES ('<?xml version="1.0" encoding="UTF-8"?><Article
xsi:noNamespaceSchemaLocation="C:\Program
Files\Altova\MapForce2007\MapForceExamples\Tutorial\DB2xsd.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Number>3</Number><Name>Pants<
/Name><SinglePrice>34</SinglePrice></Article>')

INSERT INTO "USER"."ARTICLES" ("ARTICLE")
    VALUES ('<?xml version="1.0" encoding="UTF-8"?><Article
xsi:noNamespaceSchemaLocation="C:\Program
Files\Altova\MapForce2007\MapForceExamples\Tutorial\DB2xsd.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Number>4</Number><Name>Jacket
</Name><SinglePrice>5750</SinglePrice></Article>')
```

7.2.10 Mapping XML Data to / from Database Fields

MapForce enables you to map data to or from database fields (columns) that store XML content. This means that XML data stored by the database field (column) can be extracted and written to any other structure supported by MapForce, and the other way round. You can map data as follows:

1. To or from fields of a dedicated XML type (for example, `Xml` in SQL Server, `XMLType` in

Oracle). Reading or writing XML to/from dedicated XML fields is applicable to databases that have native support for XML (such as IBM DB2, Oracle, and SQL Server).

2. To or from text fields storing XML content (for example, `Text, Varchar`). This applies to any database where the text field has sufficient length to store an XML document.

In either of the cases, a valid XML schema must exist for each database column to/from which you want to map data. When a database column stores XML, MapForce provides you with the choice to assign an XML schema directly from the database (if supported by the database), or select a schema from an external file. You can assign one XML schema per database column. If the schema has multiple root elements, you can select a single root element of that schema.

When XML is stored as a string field in a database, the character encoding of the XML document is that of the underlying string field. If the database field does not store text as Unicode, some characters cannot be represented.

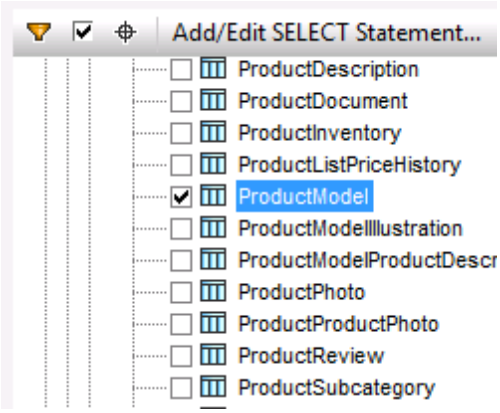
Some databases support XML encoding for XML fields (which may not necessarily be the same as that of the database character set). If supported by the database, the XML document encoding declaration is assumed to be the one declared in the XML field. For information about the XML encoding support provided by various databases, refer to their documentation.

7.2.10.1 Assigning an XML Schema to a Database Field

This topic illustrates how to assign a schema to a field that is natively defined as XML type in the database. The instructions below use SQL Server 2014 and the Adventure Works 2014 database. The latter can be downloaded from the CodePlex website (https://msftdbprodsamples.codeplex.com/). Note that mapping of data to or from XML fields works in the same way with other database types that support XML fields.

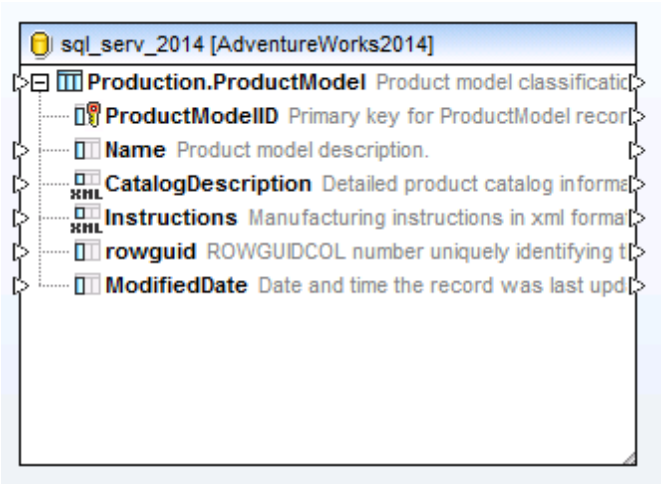
To add the Adventure Works 2014 database as a mapping component:

1. On the **Insert** menu, click **Database**, and follow the wizard to connect to the database using your preferred method (ADO or ODBC). For more information, see Connecting to Microsoft SQL Server (ADO) and Connecting to Microsoft SQL Server (ODBC). NOTE: If you use the **SQL Server Native Client** driver, you might need to set the **Integrated Security** property to a space character (see Setting up the SQL Server Data Link Properties).
2. On the **Insert Database Object** dialog box, expand the **Production** schema, and then select the **ProductModel** table.

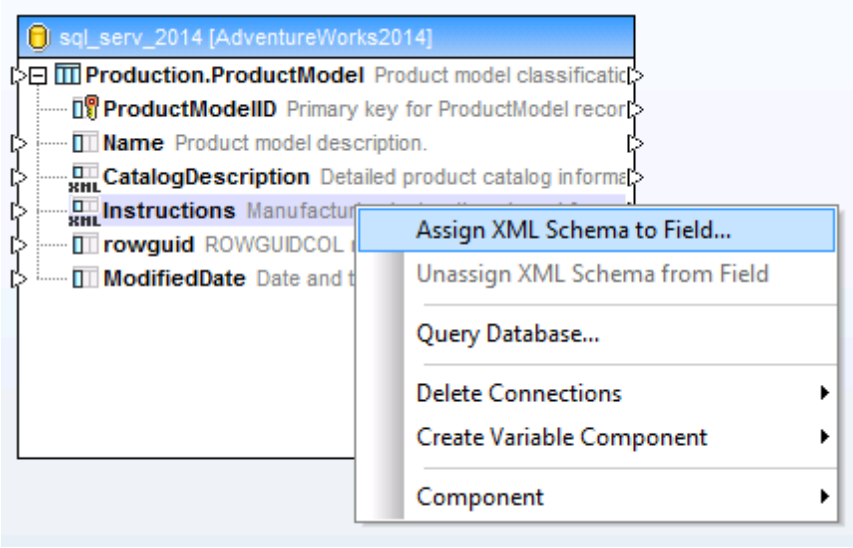


3. Click OK.

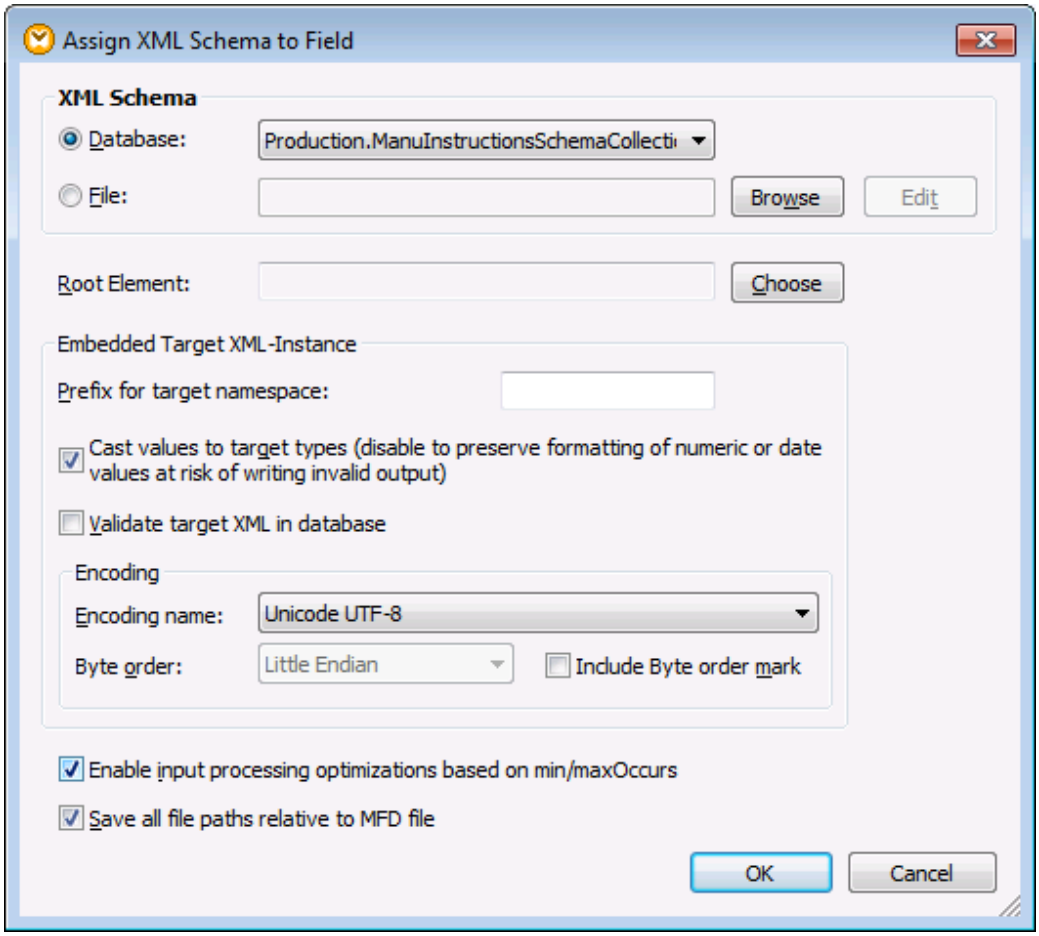
The database table has now been added to the mapping area. Notice that this table has two fields of XML type: **CatalogDescription** and **Instructions**:



For the structure of the XML fields to appear on the mapping, the XML schema of the field content is required. Right-click the **Instructions** field and select **Assign XML Schema to Field** from the context menu.

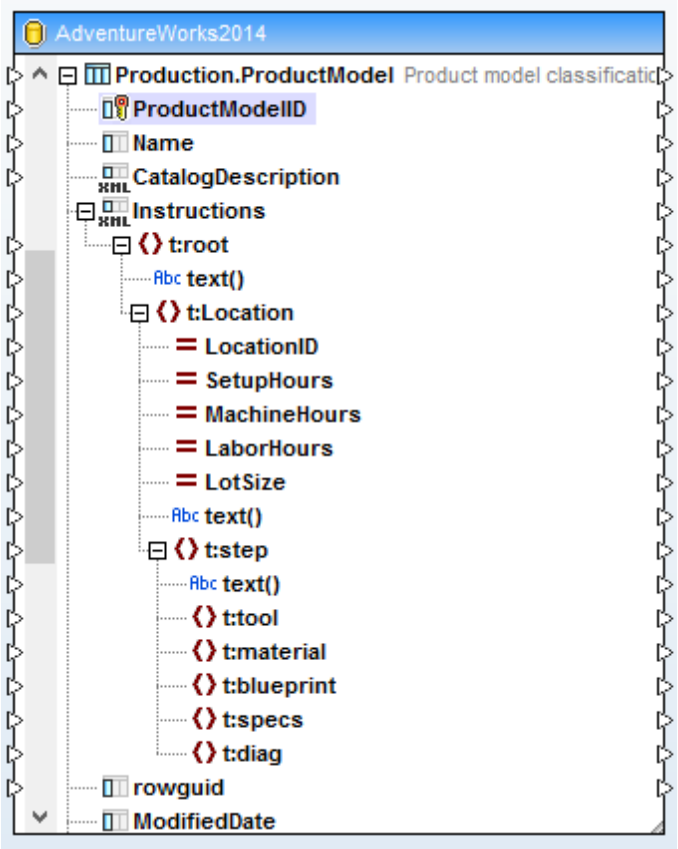


In this particular example, you will assign a schema to the **Instructions** field directly from the database. To do this, select the **Production.ManuInstructionsSchemaCollection** item next to the **Database** option, and then click OK.



The structure of the XML field now appears on the component. You can now draw connections

(and map data) to or from this field.



7.2.10.2 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 \MapForce2016\MapForceExamples\Tutorial**. The file names are as follows:

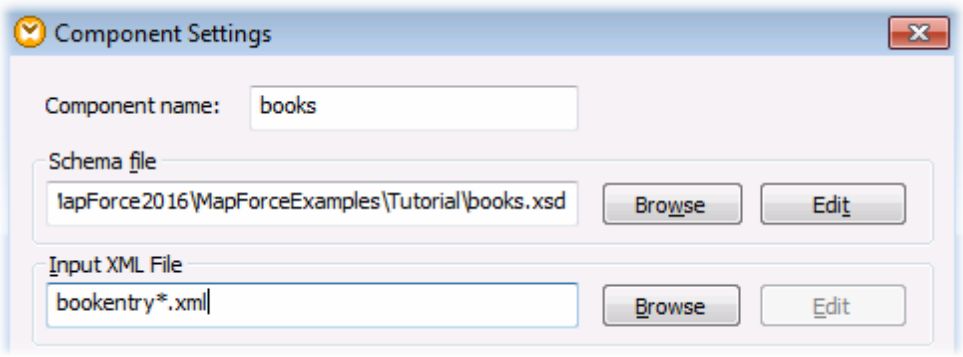
The mapping design file	• **XmlToSqliteField.mfd**
The source XML files	• **bookentry1.xml** • **bookentry2.xml** • **bookentry3.xml**
The XML schema used for validation	• **books.xsd**
The target SQLite database	• **Library.sqlite**

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.

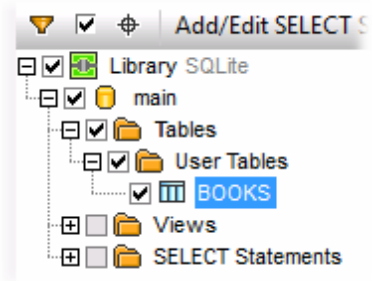
Step 1: Add the XML component

1. On the **Insert** menu, click **XML Schema/File** and browse for the **books.xsd** schema located in the **<Documents>\Altova\MapForce2016\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.

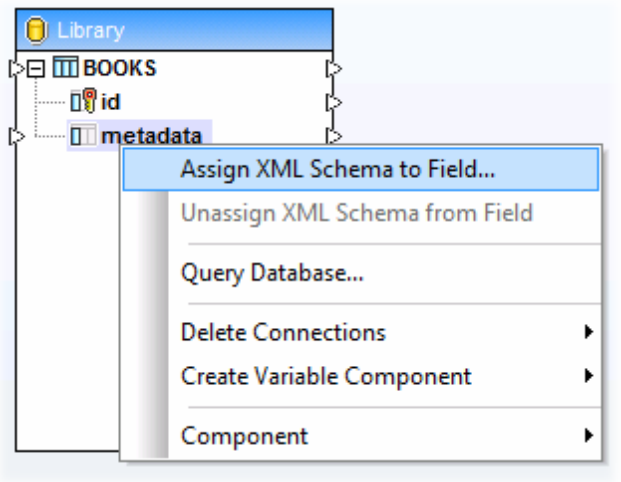


Step 2: Add the SQLite component

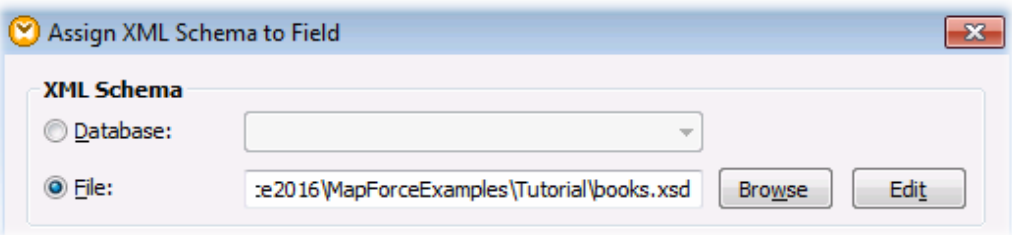
On the **Insert** menu, click **Database**, and follow the wizard to connect to the **Library.sqlite** database file from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** directory (see also Connecting to an Existing SQLite Database). When prompted to select the database objects, select the BOOKS table.



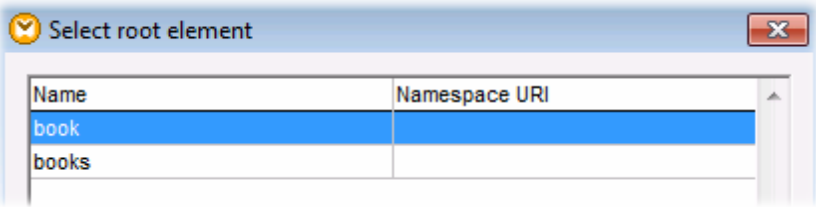
The database field where XML content will be written is called `metadata`. To assign an XML schema to this field, right-click it and select **Assign XML Schema to Field** from the context menu.



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\MapForce2016\MapForceExamples\Tutorial** directory:

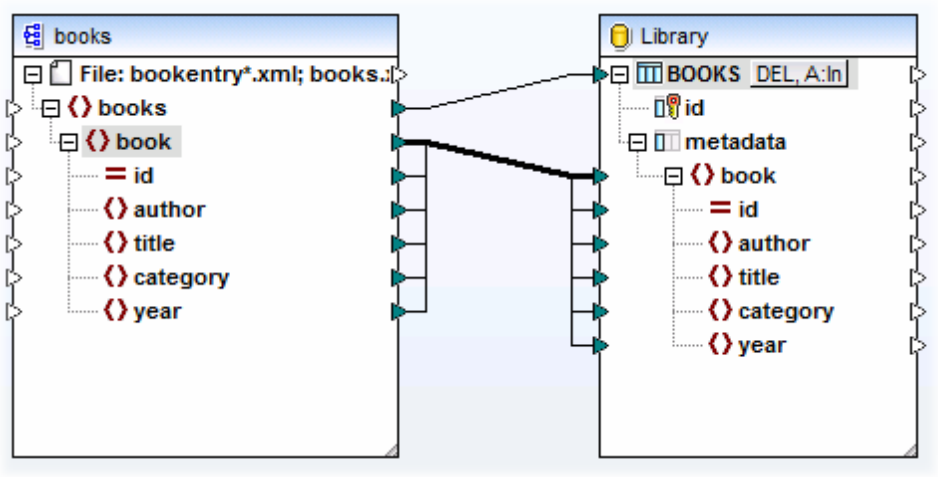


The **books.xsd** schema has two elements with global declaration: `book` and `books`. In this example, we will set `book` as the root element of the XML written to the database field. Click **Choose**, and select `book` as root element:



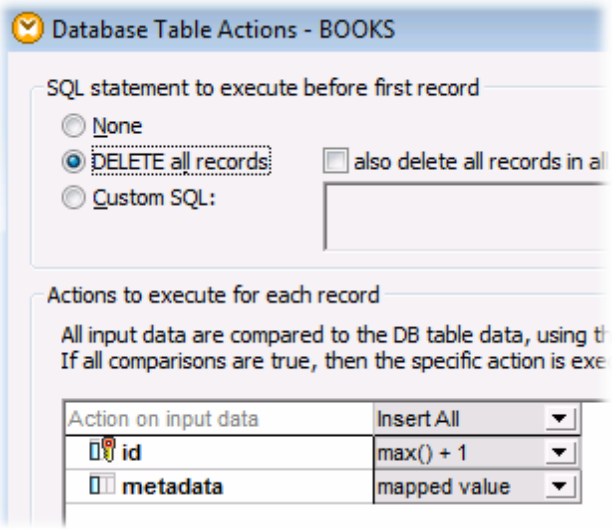
Step 3: Create the mapping connections and configure the database INSERT action

Create the mapping connections as follows:



As shown above, the connection from `book` to `book` is a "Copy-All" connection, since both the source and target use the same schema and the names of child elements are the same. For more information about such connections, see Copy-all connections.

The topmost connection (`books` to `BOOKS`) iterates through each book element in the source and writes a new record in the BOOKS table. Click the `A:In` button on the database component and set the database update settings as shown below:



The **DELETE all records** option instructs MapForce to delete the contents of the `BOOKS` table before inserting any records.

The **Insert All** actions specify that a database `INSERT` query will take place. The field `id` is generated from the database itself, while the field `metadata` will be populated with the value provided by the mapping.

Make sure to save the mapping before running it.

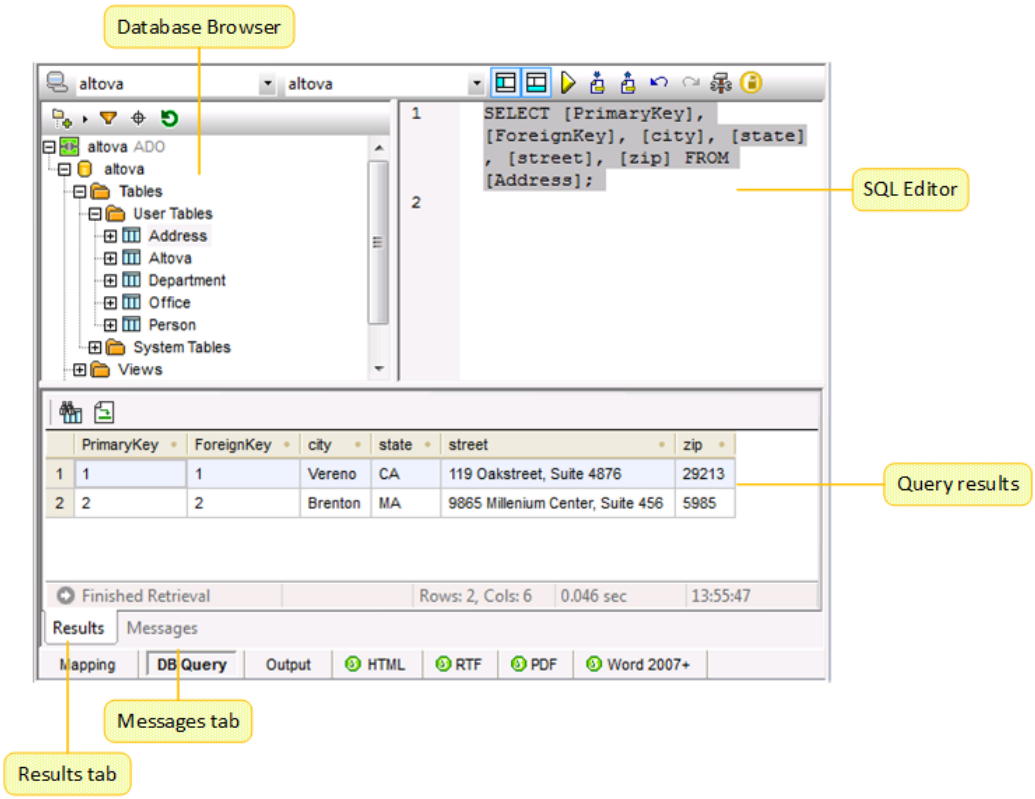
To run the mapping and view the generated output, click the **Output** tab. Note that this action does not update the database immediately. When you are ready to run the generated database

script, select the menu command **Output | Run SQL Script** (or click the  toolbar button).

7.2.11 Browsing and Querying Databases

MapForce has a dedicated Database Query pane (also called **DB Query**) that allows you to query a database independently of the mapping process. Such direct queries are not saved together with the mapping *.mfd file but provide a convenient way to browse or modify the contents of a database directly from MapForce.

A separate **DB Query** pane exists for each currently active mapping. You can create multiple active connections, to different databases, within each **DB Query** pane. Note that the connections created from the **DB Query** pane are not part of the mapping and thus are not preserved after you close MapForce, unless you define them as Global Resources (see Global Resources - Databases).



The Database Query pane consists of the following parts:

- **Database Browser**, which displays connection info and database tables
- **SQL Editor**, in which you write your SQL queries
- **Results tab**, which displays the query results in tabular form
- **Messages tab,** which displays warnings or error messages.

The upper area of the Database Query pane contains the connection controls allowing you to define the working databases, as well as the connection and database schemas.

7.2.11.1 Selecting or Connecting to a Database

For each database that you want to query, a database connection must be created. If your mapping already includes a database component, you can select the existing database connection from the upper area of the **DB Query** pane (by default, the connection is "Offline") and start exploring the database objects and run queries.

If your mapping does not include any database component, or if you want to connect to a new database, click **Quick Connect** ( ) and follow the wizard steps to create a new database connection (see Examples). You can also select an existing database connection from Global Resources, if one has been defined as such (see Global Resources - Databases).

Once you are connected to the database, you can create database queries using one of the following methods:

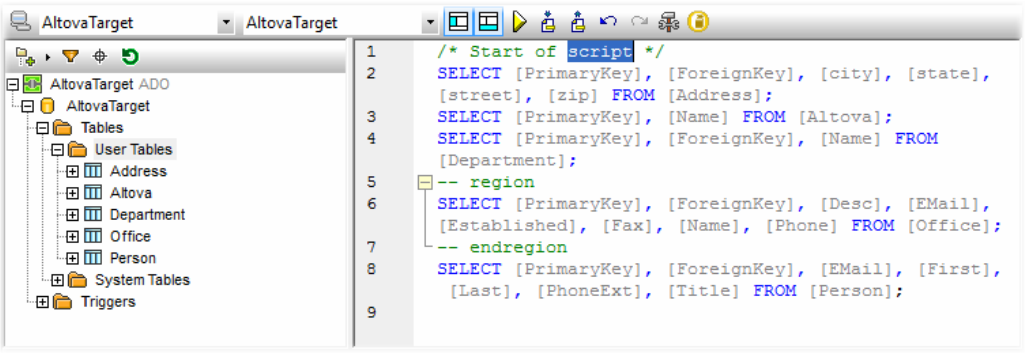
- Import the SQL query into the SQL Editor pane from an existing SQL file.
- Write the query in the SQL Editor pane.
- Right-click an object in the Database Browser pane and generate a query (typically, SELECT).

When you are ready to run the query displayed in the SQL Editor pane, click the **Execute** button. The database data is retrieved and displayed in the Results tab in tabular form. Note that the status bar displays the "Finished Retrieval" message (Finished Retrieval), and other pertinent information about the query results.

Once the "Finished Retrieval" message is displayed, you can search, sort, or copy to clipboard the search results (see Database Query - Results & Messages tab).

7.2.11.2 Creating and Editing SQL Statements

The SQL Editor is used to write and execute SQL statements. It displays any SQL statements that you may have generated automatically, loaded from existing SQL scripts, or written manually. The SQL Editor supports autocompletion, regions, and line or block comments.



The SQL Editor toolbar provides the following buttons:

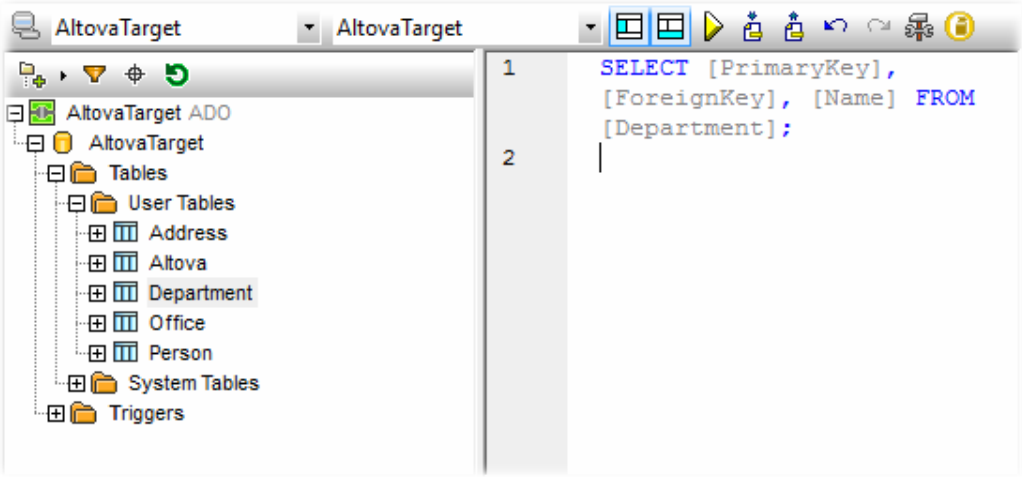
	Toggle Browser: Toggles the Browser pane on and off.
	Toggle Result: Toggles the Result pane on and off.
	Execute (F5): Clicking this button executes the SQL statements that are currently selected. If multiple statements exist and none are selected, then all are executed.
	Undo: Allows you to undo an unlimited number of edits in the SQL window.
	Redo: Allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.
	Import SQL file: Opens an SQL file in the SQL Editor, which can then be executed.
	Export SQL file: Saves SQL queries for later use.
	Open SQL script in DatabaseSpy: Starts DatabaseSpy and opens the script in the SQL Editor.
	Options: Opens the Options dialog box allowing you to define general database query settings as well as SQL Editor settings.

Generating SQL Statements

SQL statements can be generated automatically from the Database Browser, loaded from scripts, or entered manually.

To generate SQL SELECT statements from the Database Browser, do one of the following:

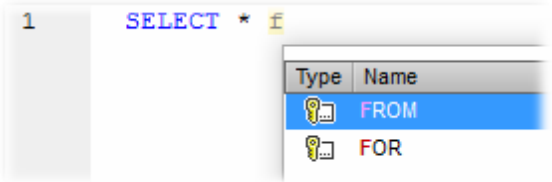
- Click a database object (such as a table or view), or a folder, in the Database Browser and drag it into the SQL Editor.



- Right-click a database object in the Database Browser and select **Show in SQL Editor | Select**.

To create SQL statements manually using autocompletion:

1. Start entering the SQL statement in the SQL Editor. If autocompletion is set to occur automatically, a drop-down list with suggestions appears while you enter statement.



2. Use the cursor **Up** and **Down** keys to select a suggestion, and then press an autocompletion key (for example, **Space**) to insert the highlighted option. The autocompletion settings, including the keys that trigger autocompletion, are configurable (see Autocompletion).

Executing SQL Statements

The SQL statements that appear in the SQL Editor can be executed against the database, with immediate effect. The result of the SQL query and the number of affected rows is displayed in the **Messages** pane of the **DB Query** pane.

When multiple SQL statements appear in the SQL Editor, only the selected statements will be executed. You can select individual SQL statements as follows:

- Holding the left mouse button clicked, drag the cursor over a specific statement.
- Click a line number in the SQL Editor.
- Triple-click a specific statement.

To execute a SQL statement:

1. Enter or select the SQL statement in the SQL Editor (see Generating SQL Statements).
2. Click the **Execute** ( ) button.

Importing and Exporting SQL Scripts

You can save any SQL that appears in an SQL Editor window to a file and re-use the script file later on.

To export the contents of the SQL Editor pane to a file:

- Click **Export SQL file** ( ), and enter a name for the SQL script.

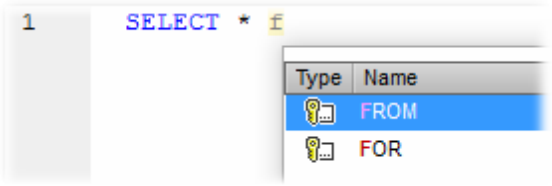
To import a previously saved SQL file:

- Click **Import SQL file** ( ), and select the SQL file you want to open.

Using Autocompletion

When entering an SQL statement in the SQL Editor, autocompletion helps you by suggesting the appropriate keywords, data types, identifiers, separators, and operators depending on the type of statement you are entering.

If autocompletion is set to occur automatically (this is the default option), a list with suggested entries appears while you enter a statement in the SQL Editor.



To insert the highlighted option, press any of the keys defined as **Completion Keys** in autocompletion settings (see SQL Autocompletion Settings).

Adding and Removing SQL Comments

The SQL Editor allows you to comment out statements, parts of statements, or groups of statements. These statements, or the respective parts of them, are skipped when the SQL script is being executed.

To comment out a section of text:

1. Select a statement or part of a statement.
2. Right-click the selected statement and select **Insert / Remove Block Comment**.

```
1
2   /*
3   SELECT [PrimaryKey], [ForeignKey], [city], [state],
    [street], [zip] FROM [Address];
4   SELECT [PrimaryKey], [Name] FROM [Altova];
5   SELECT [PrimaryKey], [ForeignKey], [Name] FROM
    [Department];*/
6
7   SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail],
    [Established], [Fax], [Name], [Phone] FROM [Office];
8   SELECT [PrimaryKey], [ForeignKey], [EMail], [First],
    [Last], [PhoneExt], [Title] FROM [Person];
9
```

To comment out text line by line:

- Right-click at the position you want to comment out the text and select **Insert / Remove Line Comment**. The statement is commented out from the current position of the cursor to the end of the statement.

To remove a block comment or a line comment:

1. Select the part of the statement that is commented out. If you want to remove a line comment, it is sufficient to select only the comment marks - - before the comment.
2. Right-click and select **Insert / Remove Block (or Line) Comment**.

Using Bookmarks

Bookmarks are used to mark items of interest in long scripts.

To add a bookmark:

- Right-click the line you want to have bookmarked and select **Insert/Remove Bookmark** from the context menu.

```
1
2   SELECT [PrimaryKey], [ForeignKey], [city], [state],
    [street], [zip] FROM [Address];
3   SELECT [PrimaryKey], [Name] FROM [Altova];
4   SELECT [PrimaryKey], [ForeignKey], [Name] FROM
    [Department];
5   SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail],
    [Established], [Fax], [Name], [Phone] FROM [Office];
6   SELECT [PrimaryKey], [ForeignKey], [EMail], [First],
    [Last], [PhoneExt], [Title] FROM [Person];
7
```

A bookmark icon is displayed in the margin at the beginning of the bookmarked line.

To remove a bookmark:

- Right-click the line from where you want to remove the bookmark and select **Insert/ Remove Bookmark** from the context menu.

To navigate between bookmarks:

- To move the cursor to the next bookmark, right-click and select **Go to Next Bookmark**.
- To move the cursor to the previous bookmark, right-click and select **Go to Previous Bookmark**.

To remove all Bookmarks:

- Right-click and select **Remove all Bookmarks**.

Inserting Regions

Regions are sections of text that you mark and declare as a unit to structure your SQL scripts. Regions can be collapsed and expanded to display or hide parts of SQL scripts. It is also possible to nest regions within other regions.

When you insert a region, an expand/collapse icon and a `--region` comment are inserted above the selected text.

Note: You can change the name of a region by appending descriptive text to the --region comment. The word "region" must not be deleted, e.g. --region DB2query.

To create a region:

1. In the SQL Editor, select the statements you want to make into a region.
2. Right-click and select **Add Region** from the context menu. The selected statements become a region which can be expanded or collapsed.

```
1
2     SELECT [PrimaryKey], [ForeignKey], [city], [state],
      [street], [zip] FROM [Address];
3     -- region
4     SELECT [PrimaryKey], [Name] FROM [Altova];
5     SELECT [PrimaryKey], [ForeignKey], [Name] FROM
      [Department];
6     -- endregion
7     SELECT [PrimaryKey], [ForeignKey], [Desc], [EMail],
      [Established], [Fax], [Name], [Phone] FROM [Office];
8     SELECT [PrimaryKey], [ForeignKey], [EMail], [First],
      [Last], [PhoneExt], [Title] FROM [Person];
9
```


3. Click the + or - box to expand or collapse the region.

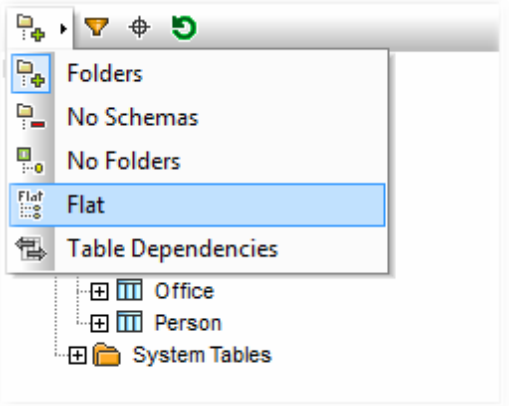
To remove a region:

- Delete the `-- region` and `-- endregion` comments.

7.2.11.3 Browsing Database Objects

When you are connected to one or several databases, the **Database Browser** pane gives a full overview of the objects in each database, including tables, views, procedures, and so on, up to the most detailed level. For databases with XML support, the **Database Browser** additionally shows registered XML schemas in a separate folder.

For custom navigation through database objects, the **Database Browser** pane includes several predefined database display layouts. The predefined layouts are available in the top area of the Database Browser.



To select a layout, click the **Folders Layout** () drop-down button and select an entry from the list. Note that the button changes with the selected layout.

- The **Folders** layout organizes database objects into folders based on object type in a hierarchical tree, this is the default setting.
- The **No Schemas** layout is similar to the Folders layout, except that there are no database schema folders; tables are therefore not categorized by database schema.
- The **No Folders** layout displays database objects in a hierarchy without using folders.
- The **Flat** layout divides database objects by type in the first hierarchical level. For example, instead of columns being contained in the corresponding table, all columns are displayed in a separate Columns folder.
- The **Table Dependencies** layout categorizes tables according to their relationships with other tables. There are categories for tables with foreign keys, tables referenced by foreign keys and tables that have no relationships to other tables.

In addition to layout navigation, you can use the **Database Browser** for the following tasks:

- Generate SQL statements (see Generating SQL Statements).
- Filter and search the displayed database objects (see Filtering and Searching Database Objects).

- Sort the tables into "System" and "User" tables.
- Refresh the root object of the active data source.

To sort tables into User and System tables:

- In the **Database Browser**, right-click the "Tables" folder, and then select **Sort into User and System Tables**.

  Note: This function is available when one of the following layouts is selected: **Folders**, **No Schemas** or **Flat**.

To refresh the root object of the active data source:

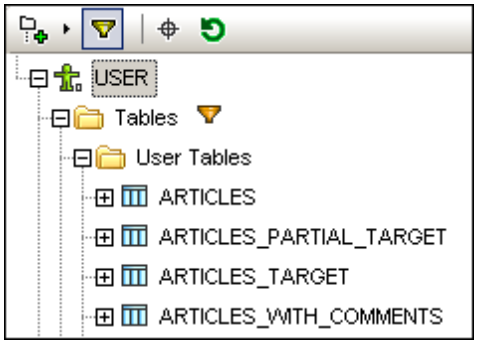
- At the top of the **Database Browser**, click **Refresh** ( ).

Filtering and Searching Database Objects

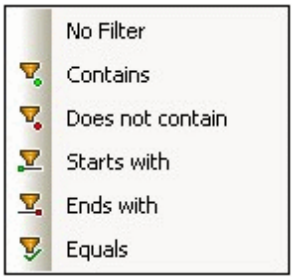
You can filter any database objects (schemas, tables, views, etc) displayed in the **Database Browser** by name or part of a name. Objects are filtered as you type in the characters. Filtering is case-insensitive by default. Filtering is not supported if you have selected the "No Folders" layout.

Filtering database objects

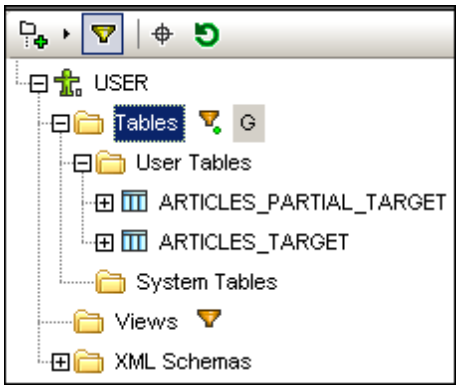
1. At the top of the Database Browser, click **Filter Folder contents** ( ). Filter icons appear next to all folders in the currently selected layout.



2. Click the filter icon next to the folder you want to filter, and select the filtering option from the context menu (for example, **Contains**).



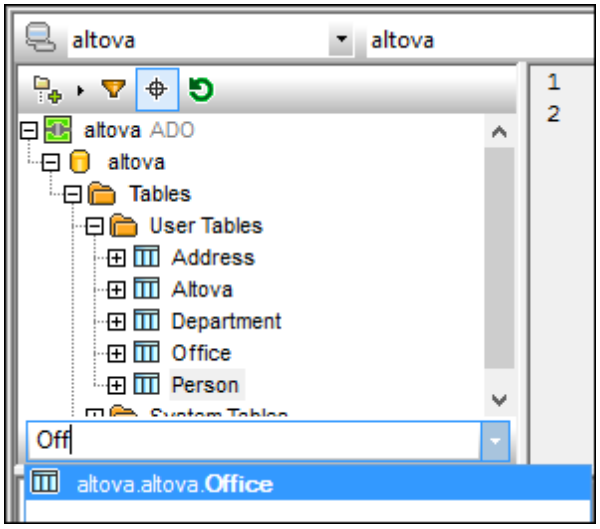
3. In the empty field which appears next to the filter icon, enter the search text (for example, "G"). The results are adjusted as you type.



Searching database objects

To find a specific database item by its name, you can either use filtering functions or the **Object Locator**. To find database elements using the Object Locator:

1. At the top of the **Database Browser**, click **Object Locator** ( ) .
2. In the drop-down list that appears, enter the search text (for example, "Off").



3. Click an object in the list to select it in the **Database Browser**.

Context Options in Database Browser

The context menu options available in the **Database Browser** depend on the object you have selected, for example:

- Right-clicking the "root" object allows you to **Refresh** the database.
- Right-clicking a folder always presents the same choices: **Expand | Siblings | Children** and **Collapse | Siblings | Children**.
- Right-clicking a database object reveals the **Show in SQL Editor** command and the submenu items discussed below.

To select multiple database objects, press either **Shift + Click** or **Ctrl + Click**.

Note: The syntax of the SQL statements may vary depending on the database you are using. The syntax below applies to Microsoft SQL Server 2014.

The following options are available under the **Show in SQL Editor** context menu for the root object:

- **CREATE**: Creates a CREATE statement for the selected database root object, for example:
  ```
  CREATE DATABASE [MYDB]
  ```
- **DROP**: Creates a DROP statement for the selected database root object, for example:
  ```
  DROP DATABASE [MYDB]
  ```

The following options are available under the **Show in SQL Editor** context menu for tables and views:

- **SELECT**: Creates a SELECT statement that retrieves data from all columns of the source table, for example:
  ```
  SELECT [DepartmentID], [Name], [GroupName], [ModifiedDate] FROM [MYDB].[HumanResources].[Department]
  ```
- **Name**: Returns the name of the table.
- **Path**: Returns the full path of the tables, in the format `DataSourceName.DatabaseName.SchemaName.TableName`.

If you selected multiple tables, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for columns:

- **SELECT**: Creates a SELECT statement that retrieves data from the selected column(s) of the parent table, for example:
  ```
  SELECT [DepartmentID] FROM [MYDB].[HumanResources].[Department]
  ```
- **Name**: Returns the name of the selected column.
- **Path**: Returns the full path of the column, in the format `DataSourceName.DatabaseName.SchemaName.TableName.ColumnName`.

If you selected multiple columns, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for constraints:

- **Name**: Returns the name of the selected constraint.
- **Path**: Returns the full path of the constraint, in the format `DataSourceName.DatabaseName.SchemaName.TableName.ConstraintName`.

If you selected multiple constraints, the names or paths are printed on separate lines, separated by commas.

The following options are available under the **Show in SQL Editor** context menu for indexes:

- **Name**: Returns the name of the selected index.
- **Path**: Returns the full path of the index, in the format `DataSourceName.DatabaseName.SchemaName.TableName.IndexName`.

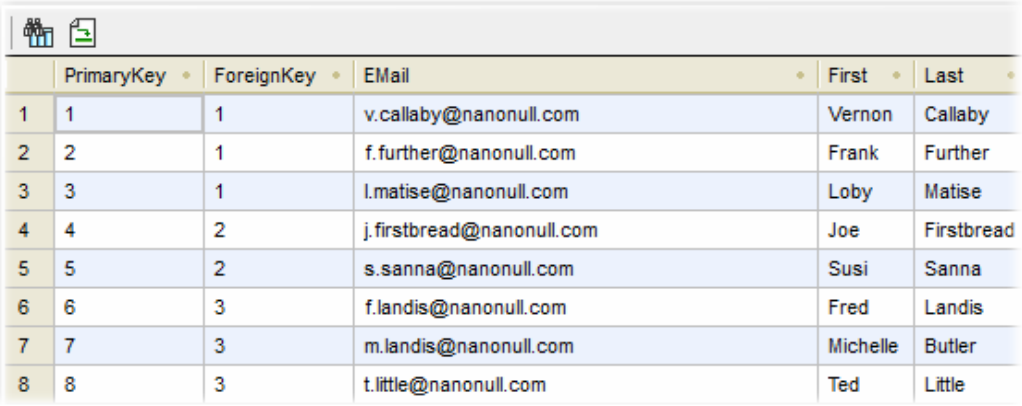
If you selected multiple indexes, the names or paths are printed on separate lines, separated by commas.

If the database has support for XML Schemas, the following options are available for every schema displayed under the "XML Schemas" folder:

- **View in XMLSpy**: Opens the database schema in XMLSpy, provided that the latter is installed.
- **Manage XML Schemas**: Opens a dialog box where you can add new or drop existing database XML schemas.

7.2.11.4 Copying, Sorting, and Searching the Query Results

The **Results** tab of the **DB Query** pane shows the recordset retrieved as a result of a database query.



	PrimaryKey	ForeignKey	EMail	First	Last
1	1	1	v.callaby@nanonull.com	Vernon	Callaby
2	2	1	f.further@nanonull.com	Frank	Further
3	3	1	l.matise@nanonull.com	Loby	Matise
4	4	2	j.firstbread@nanonull.com	Joe	Firstbread
5	5	2	s.sanna@nanonull.com	Susi	Sanna
6	6	3	f.landis@nanonull.com	Fred	Landis
7	7	3	m.landis@nanonull.com	Michelle	Butler
8	8	3	t.little@nanonull.com	Ted	Little

The toolbar buttons enable navigation between results and SQL statements and facilitate searching within the query results.

	**Find:** Searches a specific text within the displayed results. Press **F3** to go to the next occurrence of the search term.
	**Go to statement**: Jumps to the SQL Editor and highlights the SQL statement that produced the current result. This might be particularly useful when the SQL Editor contains multiple statements.

To select cells from the query results:

- Click a column header to select the entire column
- Click a row number to select the entire row
- Click individual cells. Holding down the **Ctrl** key while clicking allows you to make multiple selections. If a column or cell contains XML data then this data can also be copied.

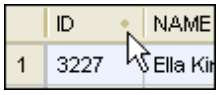
  Note: The context menu can also be used to select data, **Selection | Row | Column | All.**

To copy the selected cells to clipboard:

- Right-click and select **Copy selected cells** from the context menu.

To sort data:

- Right-click anywhere in the column to be sorted and select **Sorting | Ascending** or **Descending**
- Click the sort icon in the column header



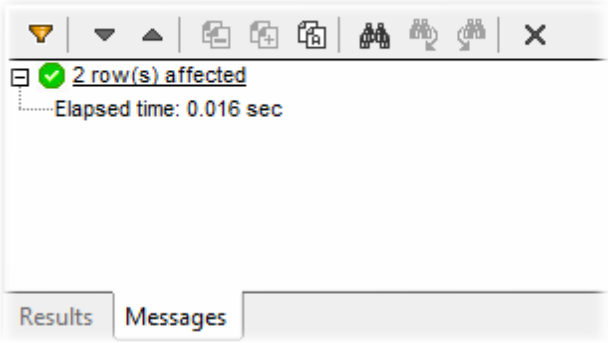
The data is sorted according to the contents of the sorted column.

To restore the default sort order:

- Right-click anywhere in the table and choose **Sorting | Restore default** from the context menu.

7.2.11.5 Viewing the Status of Executed Queries

The **Messages** tab of the **DB Query** pane provides specific information about the last executed SQL statement and reports errors or warning messages.



You can use different filters to customize the view of the **Messages** tab or use the **Next** and **Previous** buttons to browse data row by row. The buttons at the top are used to navigate the messages, copy text to clipboard, and hide certain parts of the message. These options are also available in the context menu, when you right-click anywhere inside the **Messages** tab.

	Filter: Opens a pop-up menu from where you can select the individual message parts (**Summary**, **Success**, **Warning**, **Error**) for display. Furthermore, you can check all or none of these options with a single mouse click by selecting either **Check All** or **Uncheck All** from the pop-up menu.
	Next: Jumps to and highlights the next message.
	Previous: Jumps to and highlights the previous message.
	Copy selected message to the clipboard
	Copy selected message including its children to the clipboard
	Copy all messages to the clipboard
	Find: Opens the **Find** dialog box.
	Find previous: Jumps to the previous occurrence of the string specified in the **Find** dialog box.
	Find next: Jumps to the next occurrence of the string specified in the **Find** dialog box.
	Clear: Removes all messages from the Message tab of the SQL Editor window.

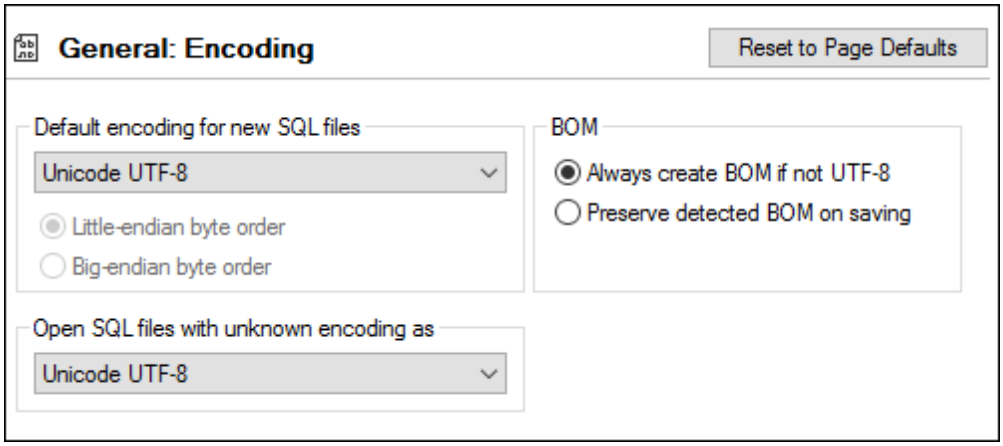
7.2.11.6 Database Query Settings

This section includes information about configuring miscellaneous settings applicable to SQL statements entered or loaded in SQL Editor, as well as the query results displayed after a query is executed.

SQL File Encoding Settings

You can specify the encoding options for SQL files created or opened with SQL Editor as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options**, and then click **General | Encoding**.



Default encoding for new SQL files

Define the default encoding for new files so that each new document includes the encoding specification that you specify here. If a two- or four-byte encoding is selected as the default encoding (for example, UTF-16, UCS-2, or UCS-4), you can also choose between little-endian and big-endian byte ordering for the SQL files.

The encoding of existing files is not affected by this setting.

Open SQL files with unknown encoding as

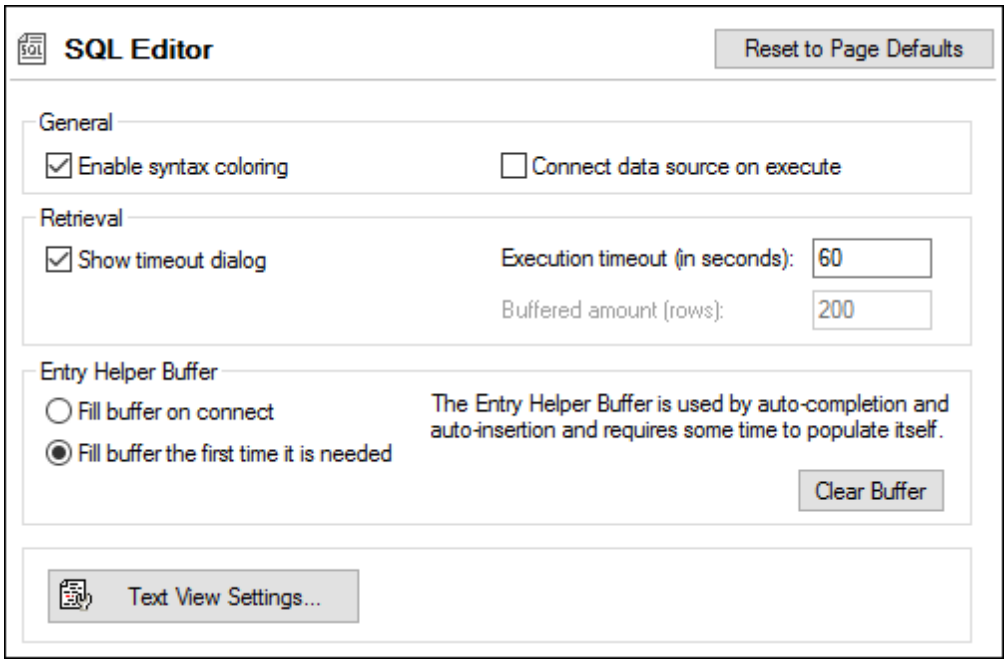
You can select the encoding with which to open an SQL file with no encoding specification or where the encoding cannot be detected.

Note: SQL files which have no encoding specification are saved with a UTF-8 encoding.

SQL Editor General Settings

You can change the general settings applicable to the SQL Editor as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options**, and then click **SQL Editor**.



General

Syntax coloring emphasizes different elements of SQL syntax using different colors.

Activate the **Connect datasource on execute** check box to connect to the corresponding data source automatically whenever a SQL statement is executed and its data source is not connected.

Retrieval

Specify the maximum amount of time permissible for SQL execution (Execution timeout) in seconds.

Activating the **Show timeout dialog** check box allows you to change the time-out settings when the permissible execution period is exceeded.

Entry Helper Buffer

Allows you to define how MapForce should fill the entry helper buffer, on connection or only the first time it is needed.

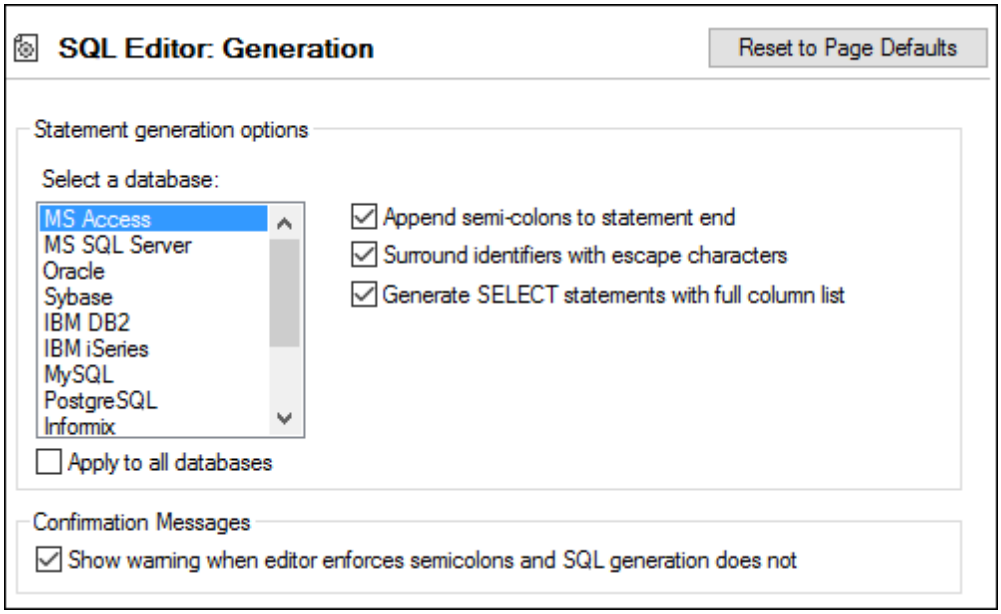
Text View Settings

Allows you to define the specific Text view settings: Margins, Tabs, Visual aids, as well as showing you the Text view navigation hotkeys.

SQL Statement Generation Settings

You can specify the SQL statement generation syntax for various database kinds as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options**, and then click **SQL Editor**.



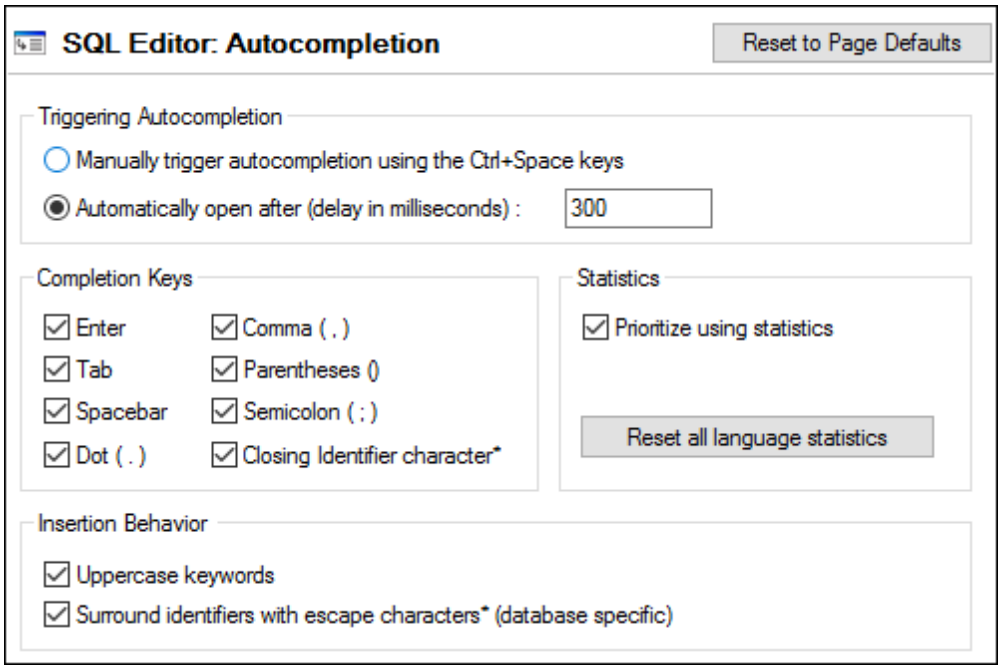
To define the syntax preferences for a specific database, select it from the list, and then enable or disable the three check boxes to the right.

To define a unique syntax for all databases, select **Apply to all databases**, and then enable or disable the three check boxes to the right. Note that using common settings for all databases may cause inability to edit data in Oracle and IBM DB2 and iSeries databases via a JDBC connection.

SQL Autocompletion Settings

You can change the autocompletion settings applicable to the SQL Editor as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options**, and then click **SQL Editor | Autocompletion**.



Autocompletion can be triggered manually or automatically.

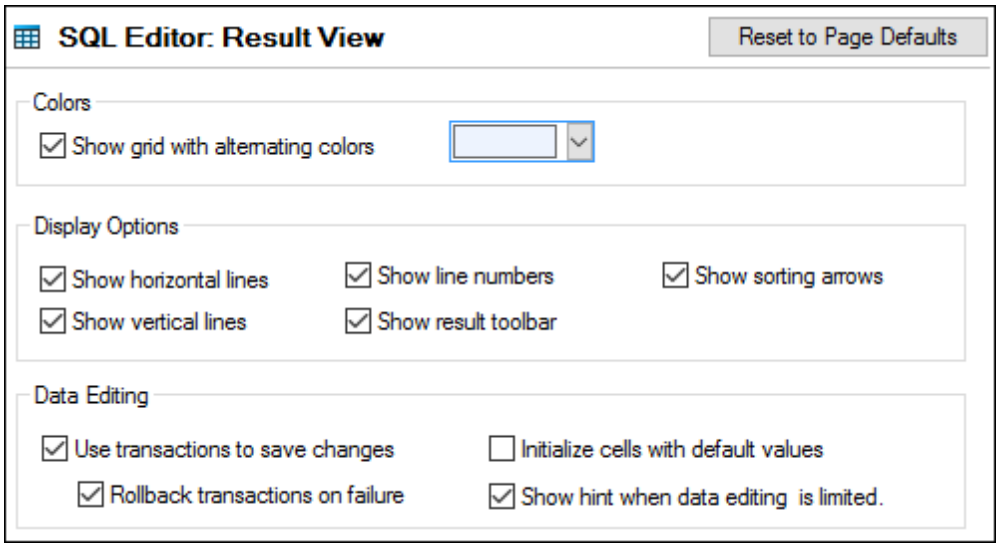
The Completion Keys section allows you to specify the specific characters that insert the specific keyword and close the autocompletion window.

Insertion Behaviour lets you define upper/lower case and if the identifiers are to be surrounded with escape characters.

Query Result View Settings

You can configure the appearance of the **Results** tab of the **DB Query** pane as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options**  , and then click **SQL Editor | Result View**.



Select the **Show grid with alternating colors** check box to display rows in Result tabs as simple grid or with alternating white and colored rows. The alternating color is configurable.

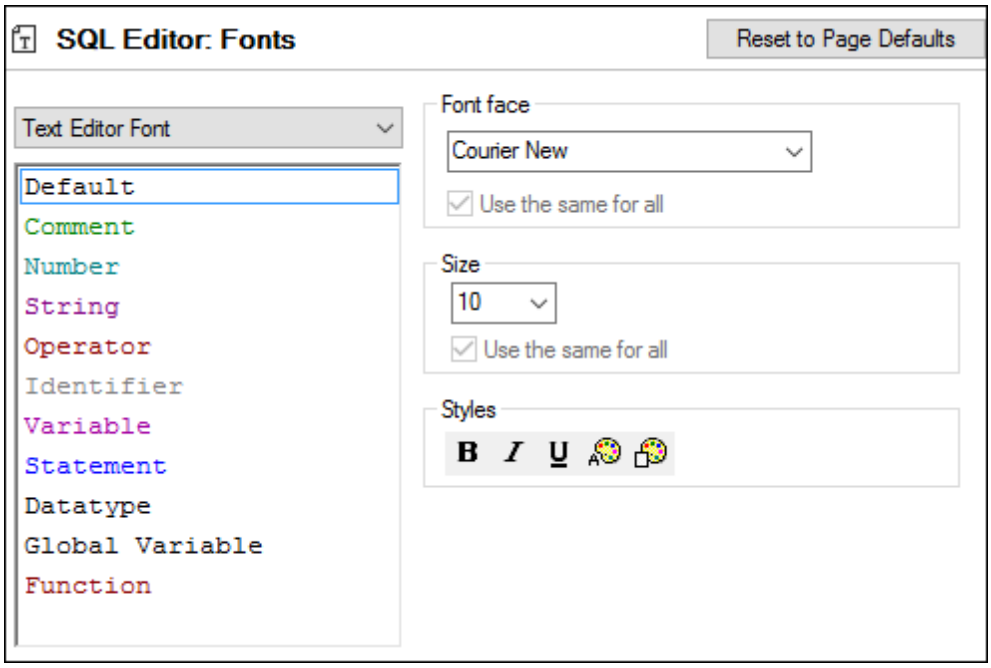
The **Display Options** group lets you define how horizontal and vertical grid lines, as well as line numbers and the **Result** toolbar, are displayed. You can switch any of these options off by deactivating the respective check box.

The **Data Editing** group lets you define the transaction settings, if the cells are to be filled with default values and if a hint is to be displayed when data editing is limited.

SQL Editor Font Settings

You can configure color and font settings of SQL statements that appear in SQL Editor as follows:

1. Click the **DB Query** tab.
2. At the top of the pane, click **Options**  , and then click **SQL Editor | Fonts**.



The font settings listed in the Font Settings list box are elements of SQL statements. You can choose the common font face, style, and size of all text that appears in SQL Editor. Note that the same font and size is used for all text types.

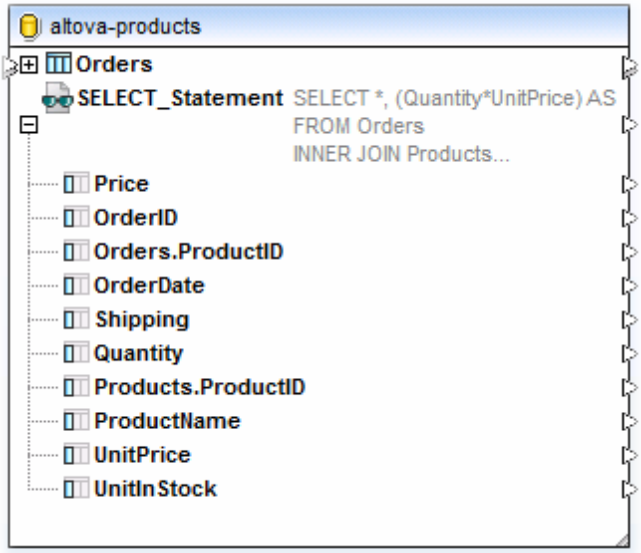
Only the style can be changed for individual text types. This enables the syntax coloring feature. Click the **Reset to default** button to restore the original settings.

7.2.12 SQL SELECT Statements as Virtual Tables

MapForce supports the creation of SQL SELECT statements with parameters in database components. These are table-like structures that contain the fields of the result set generated by the SELECT statement. These structures can then be used as a mapping data source, like any table or view defined in the database.

- When using Inner/Outer **joins** in the SELECT statement, fields of all tables are included in the component.
- Expressions with correlation names (using the SQL "AS" keyword) also appear as a mappable items in the component.
- Database views can also be used in the FROM clause.
- SELECT statements can contain parameters which use the same syntax as the SQL WHERE/ORDER component.

Once the SELECT statement has been added to a database component, the fields returned by it are available for mapping, for example:



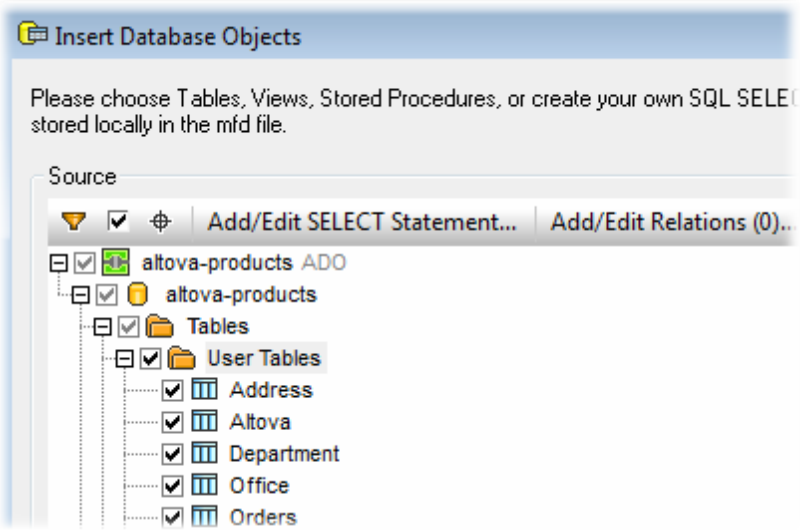
The number of visible lines of the SELECT statement is configurable. To define the number of lines you want to see on the component, select the menu command **Tools | Options**, click the **General** tab and enter the number of lines in the Mapping View group.

7.2.12.1 Creating SELECT Statements

You can create SELECT statements on any mapping which contains a database component. If your mapping does not contain a database yet, add a database first (see Connecting to a Database). For the scope of this example, select the menu command **Insert | Insert Database** and follow the wizard steps to connect to the **altova-products.mdb** file available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder.

To create a SELECT statement:

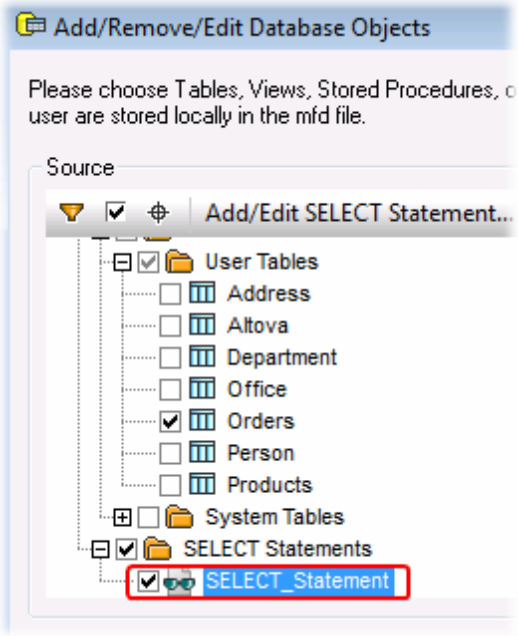
1. Right-click the title of the database component, and select **Add/Remove/Edit Database Objects**. (As an alternative, select the database component, and then select the menu command **Component | Add/Remove/Edit Database Objects**).



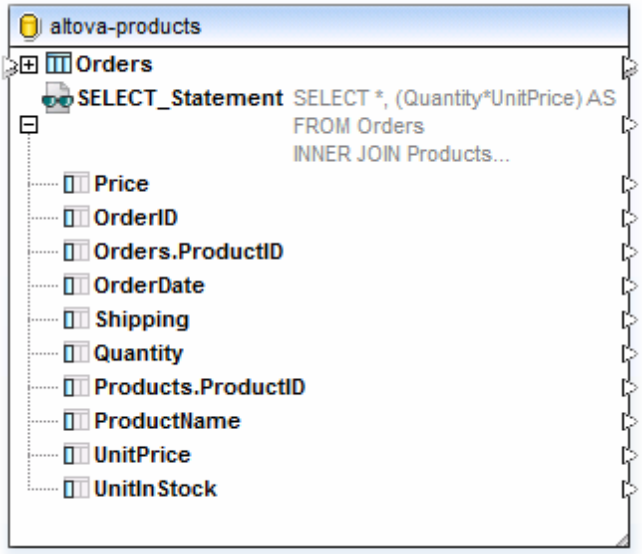
2. Do one of the following:
   - To generate the SELECT statement from an existing table, right-click any table and select **Generate and add an SQL statement** from the context menu. You will be able to edit the generated statement afterwards.
   - To write a custom SELECT statement, click the **Add/Edit SELECT Statement** button.
3. Edit or create the statement as required. For example, the SELECT statement below is valid for the **altova-products.mdb** file available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. The **Price** field is the product of the two fields, **Quantity** and **UnitPrice**, and is declared as a correlation name (`AS Price`).

```
SELECT *, (Quantity*UnitPrice) AS Price
From Orders
INNER JOIN Products
ON Orders.ProductID = Products.ProductID
Where Orders.Quantity > 2
```

4. Click **Add SELECT Statement**. Notice that the SELECT statement is now visible as a database object, similar to how tables, views, and procedures are visible.



5. Click OK. The SELECT statement is also displayed on the database component, and you can map data from any of the fields returned by the SELECT query.



Important notes:

- All calculated expressions in the SELECT statement must have a unique correlation name (like "AS Price" in this example) to be available as a mappable item.
- If you connect to an Oracle or IBM DB2 database using JDBC, the SELECT statement must have no final semicolon.

To remove a previously added SELECT statement:

1. Right-click the title of the database component, and select **Add/Remove/Edit Database Objects**.
2. Right-click the SELECT statement you want to delete, and select **Remove Select Statement**.

7.2.12.2 Example: SELECT with Parameters

This example shows you how to create a MapForce mapping which reads data from a Microsoft Access database and writes it to a CSV file. In particular, the mapping described in this example uses a custom database SELECT query with a parameter. The SELECT statement combines data from multiple tables. Then, the results are supplied to the mapping for further processing.

The example is accompanied by a mapping design (.mfd) file available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\select-component.mfd**. You might want to open this sample file and analyze it first, or follow the steps below to create it from scratch.

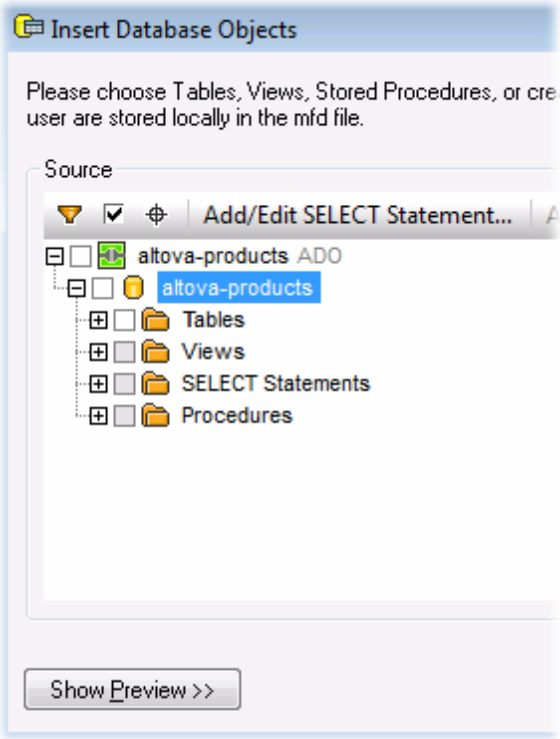
Although this example uses a Microsoft Access database, the process works in the same way for other database types. For information about connecting to other databases, see Connecting to a Database.

The goals are as follows:

1. We must select from the database only those orders where the number of ordered items exceeds a custom value. This custom value should be supplied as a parameter to the mapping. To achieve this goal, we will create a custom database SELECT statement that takes an input parameter.
2. In the Access database, the date format is `YYYY-MM-DD HH-MI-SS`. In the CSV file, the time part must be stripped, so the format should be `YYYY-MM-DD`. To achieve this goal, we use the date-from-datetime function available in MapForce.
3. The resulting CSV file must have the name **OrdersReport.csv**.

Step 1: Add the SELECT structure

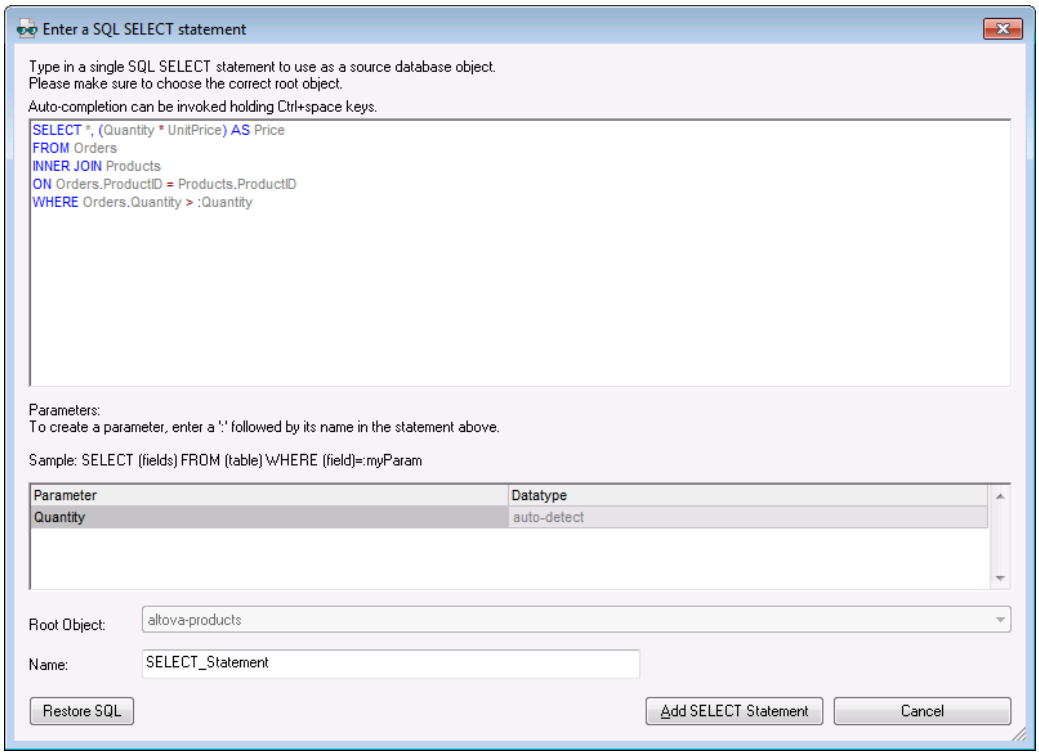
1. On the **Insert** menu, click **Database**.
2. Select **Microsoft Access (ADO)**, and follow the wizard steps to connect to the **altova-products.mdb** file available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder.



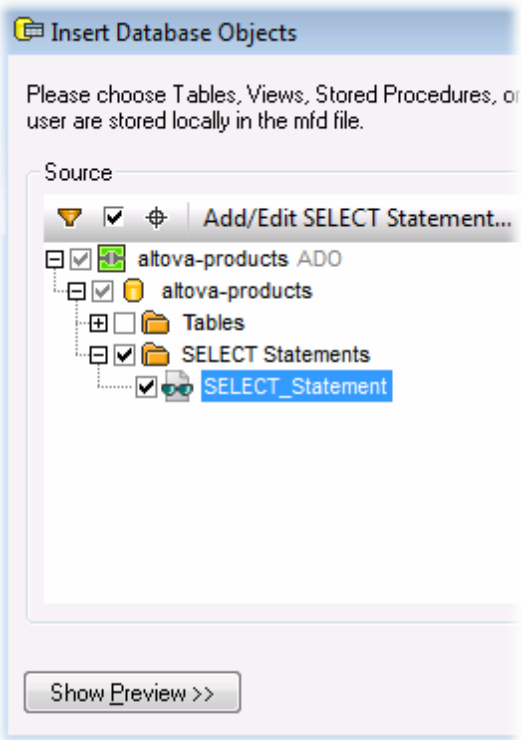
3. On the Insert Database Objects dialog box, click **Add/Edit SELECT Statement**, and enter the following query:

```
SELECT *, (Quantity * UnitPrice) AS Price
FROM Orders
INNER JOIN Products
ON Orders.ProductID = Products.ProductID
WHERE Orders.Quantity > :Quantity
```

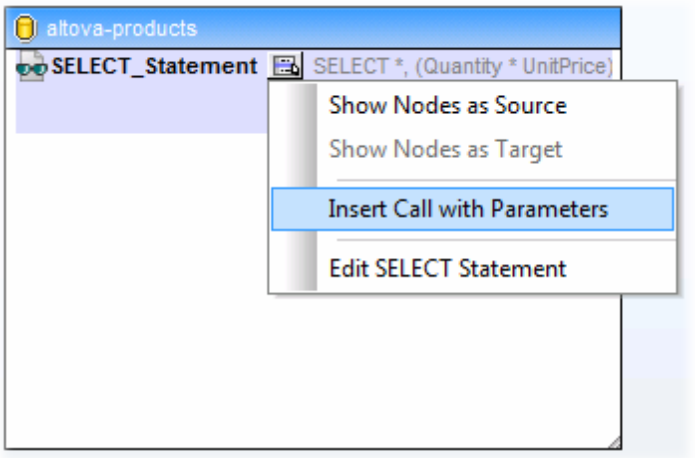
This query uses a join between the Orders and Products tables, and retrieves all fields (*), and a computed value (`AS Price`). The query also specifies the **:Quantity** parameter in the WHERE clause.



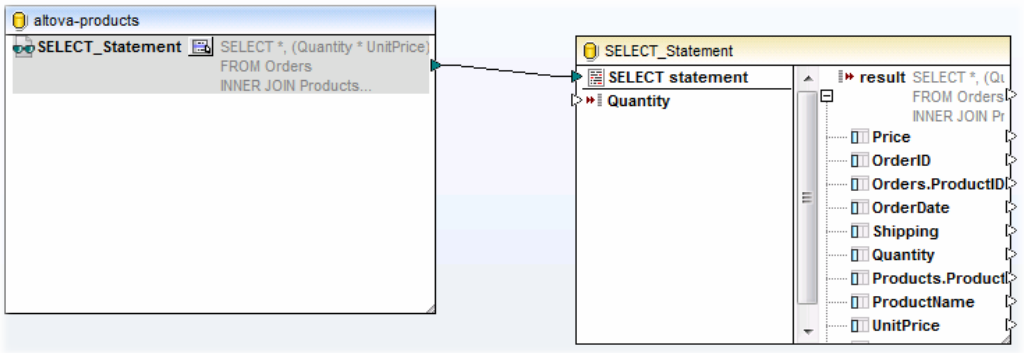
4. Click **Add SELECT statement**.



5. Click OK. The **altova-products** component has now been added to the mapping area.
6. On the **altova-products** component, click and select **Insert Call with Parameters**.

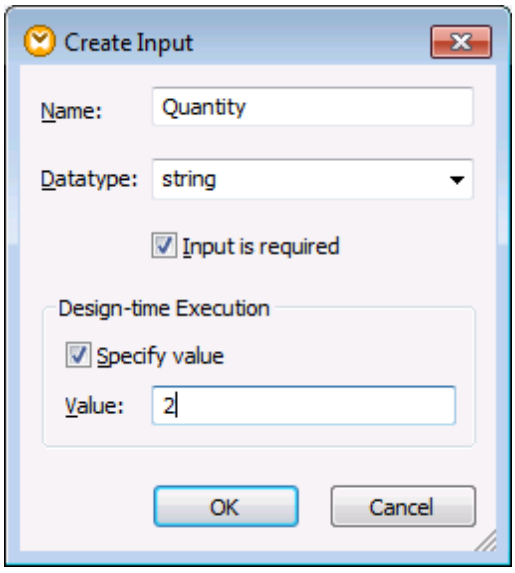


A new structure (SELECT_Statement) is now available on the mapping. It is split into two parts: the left part supplies input connectors and the right part supplies output connectors. Notice that the left part also includes the **Quantity** parameter defined previously.

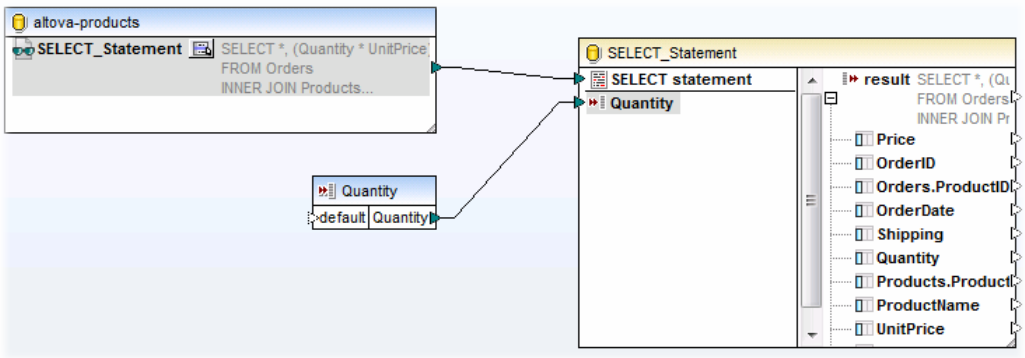


Step 2: Add the input parameter

1. On the **Insert** menu, click **Insert Input**.
2. Type "Quantity" as name.
3. Under **Design-time Execution**, enter a parameter value to be used for executing the mapping during the design phase (in this example, "2"). For more information, see Supplying Parameters to the Mapping.

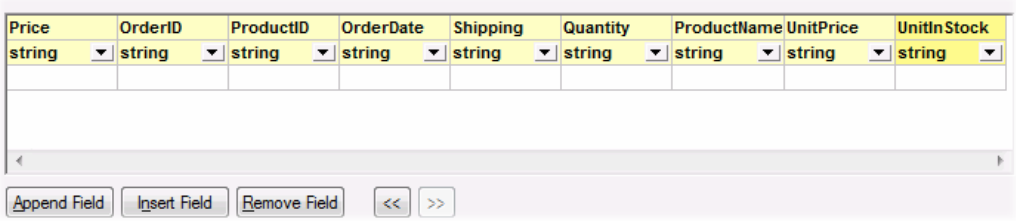


You can now connect the input parameter to the database call structure, as shown below.



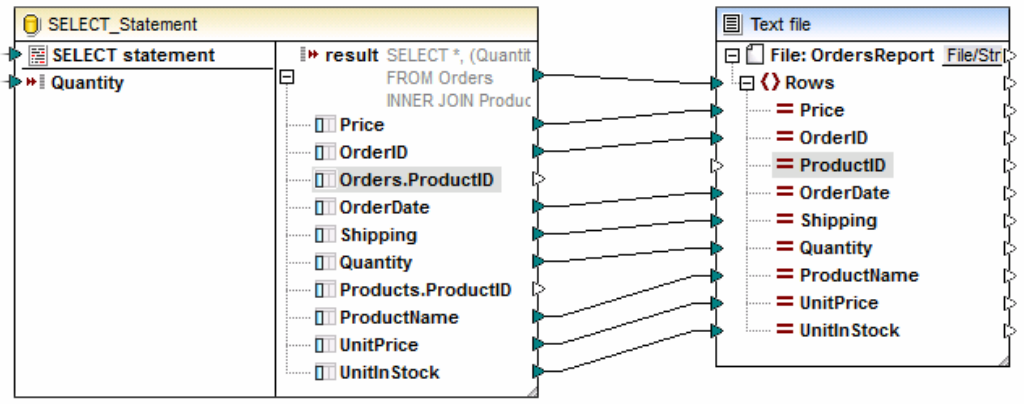
Step 3: Add the target CSV component

1. On the **Insert** menu, click **Text File**.
2. Select **Use simple processing for standard CSV...**, and then click **Continue**.
3. On the Component Settings dialog box, click **Append Field** and add nine new fields. It is recommended that you give to the CSV fields the same name as the name of the database fields, as shown below. This will help you save time later when drawing mapping connections. For more information about these settings, see Setting the CSV Options.



4. Create a connection between the **result** node of the SELECT structure and the **Rows**

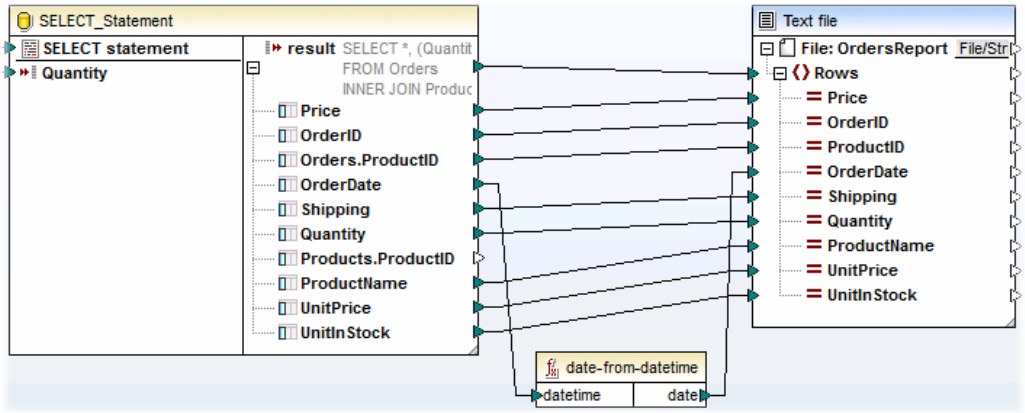
node of the CSV component.



Because most of the fields in the CSV component have the same name as their database equivalent, mapping connections will likely be drawn automatically when you connect **result** to **Rows**. If this does happen, select the **Connection** menu and make sure that the **Auto Connect Matching Children** option is enabled. The only mapping item that you have to connect manually is **ProductID**, since there is no field with this name in the SELECT structure.

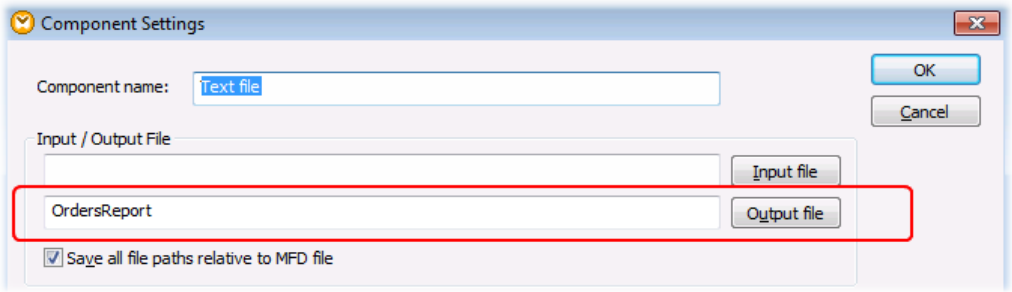
Step 4: Convert the date

In the Libraries window, search for the `date-from-datetime` function and drag it to the mapping area. Then connect its input and output as shown below.



Step 5: Set the name of the output file

To set the name of the output file to **OrdersReport.csv**, double-click the CSV component, and enter the value in the Output File box.



7.2.13 Stored Procedures

Stored procedures are programs that are hosted and run on a database server. Stored procedures can be called by client applications and they are often written in some extended dialect of SQL. Some databases support also implementations in Java, .NET CLR, or other programming languages.

Typical uses of stored procedures include querying a database and returning data to the calling client, or performing modifications to the database after additional validation of input parameters. Stored procedures can also perform other actions outside the database, e.g. send e-mails.

Stored procedures in MapForce:

- Can be present (and called) in both source and target database components.
- Can have data be mapped to them by input parameters, as well as mapped from them, by output parameters.
- Can be inserted as a function-like call. This allows you to provide input data, execute the stored procedure, and read/map the output data to other components.
- Are visible with their unique name and a clickable button, inside the database component once the database has been inserted into the mapping area.
- Cannot be edited from within MapForce
- Can only be used in the BUILTIN execution engine. Code generation in C++, C#, or Java is not supported.

Note: To illustrate how MapForce implements stored procedures, this section uses Microsoft SQL Server 2008 and the "AdventureWorks" database. The latter can be downloaded from the CodePlex website (http://sqlserversamples.codeplex.com) .

Stored procedure support

- Input/output parameter types: user-defined types, cursor types, variant types and many "exotic" database-specific data types (e.g. arrays, geometry, CLR types, etc.) are generally not supported.
- 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/JDBC). Only procedures that return the same number of recordsets with a fixed column structure are supported.

- Stored procedures in Microsoft Access databases have very limited functionality and are not supported in MapForce.

Drivers

- 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.

Firebird 2.5

- Supported in MapForce: stored procedures, table-valued functions

IBM DB2 8, 9, 10, IBM DB2 for i 6.1, 7.1

- Supported in MapForce: stored procedures, scalar functions, table-valued functions.
- Row-valued functions (RETURNS ROW) are not supported.
- Its 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.

Informix 11.7

- Supported in MapForce: stored procedures, table-valued functions.

Microsoft SQL Server 2005, 2008, 2012, 2014

- Supported in MapForce: stored procedures, scalar functions, table-valued functions.
- It is recommended to use the latest **SQL Server Native Client** driver instead of the **Microsoft OLE DB Provider for SQL Server**.
- The ADO API has limited support for some data types introduced with SQL Server 2008 (`datetime2`, `datetimeoffset`). If you encounter data truncation issues with these temporal types when using ADO with the SQL Server Native Client, you can set the connection string argument `DataTypeCompatibility=80` or use ODBC.

MySQL 5.5, 5.6

- 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 9i, 10g, 11g, 12c

- Supported in MapForce: stored procedures, scalar functions, table-valued functions.
- It is recommended to use a native Oracle driver instead of the **Microsoft OLE DB Provider for Oracle**.

- Oracle has a special way to return result sets to the client by using output parameters of type REF CURSOR. This is supported by MapForce for stored procedures, but not for functions. The names and number of recordsets is therefore always fixed for Oracle stored procedures.

PostgreSQL 8, 9

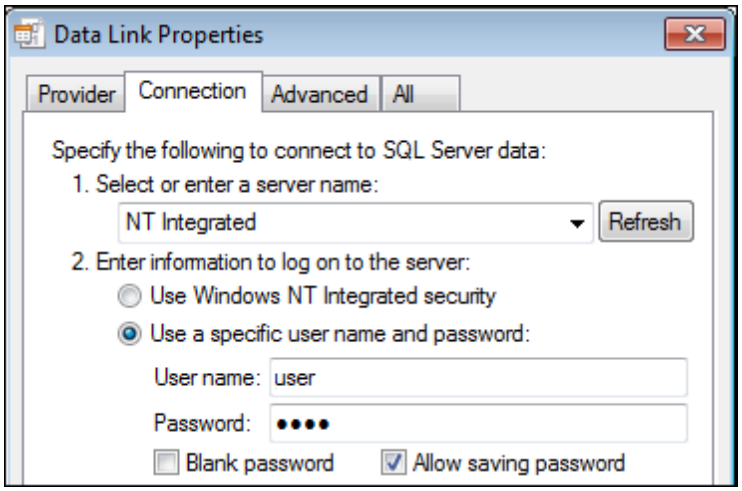
- Supported in MapForce: scalar functions, row-valued functions, table-valued functions.
- In PostgreSQL, any output parameters defined in a function describe the columns of the result set. This information is automatically used by MapForce - no detection by execution or manual input of recordsets is needed. Parameters of type refcursor are not supported.

7.2.13.1 Inserting stored procedures in database components

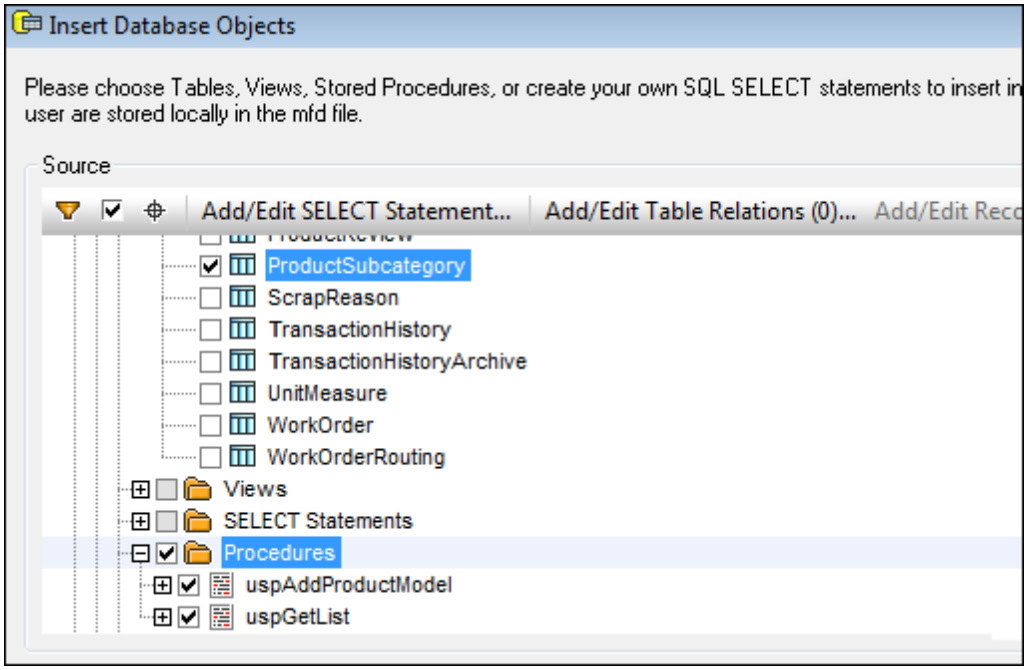
Stored procedures can be incorporated into a database component when inserting it into the mapping area. This follows the usual sequence of inserting a database component into MapForce.

To insert a database component containing stored procedures:

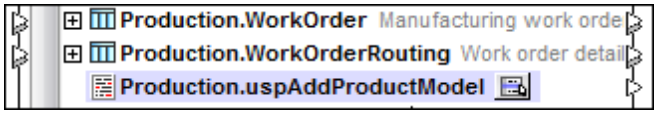
1. Click the Insert Database icon, or select the menu option **File | Insert Database**.
2. Use the Connection Wizard to connect to the database.
3. Having filled in the Connection tab of the Data Link Properties dialog box, click the OK button.



4. Click the expand button to select the database tables you want to insert, and select the specific tables.

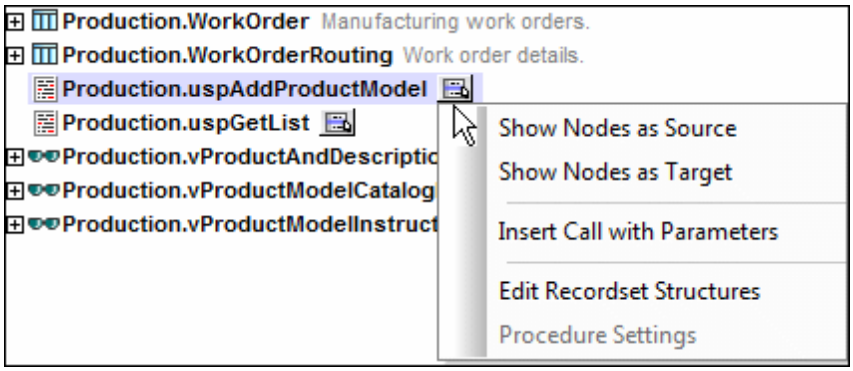


5. Click the expand button of the Procedures folder to select the stored procedures that you want to insert along with the tables, then click OK.



The database component is inserted and shows the selected tables followed by the stored procedures that you selected.

- Tables, views and procedures are sorted alphabetically in the database component.
- Each stored procedure is shown as an item in the database component containing the procedure name and a clickable button. The button allows you to select if the procedure is to be used as a source or target, as well as other procedure settings.
- At this point, MapForce has no specific information if the parameters of the stored procedure are to be used as source or target parameters. This is achieved by clicking the stored procedure button and selecting the specific option.



7.2.13.2 Use cases

The following uses cases should cover most common types of stored procedures and how to define them in MapForce.

I want to:	Read this section
I want to call a stored procedure to **retrieve** data from a database and map it to another component. E.g. I want to use a stored procedure as a data source to write the resulting data into another file (XML, TXT, EDI, etc.).	Stored procedures in Source components
I want to call a stored procedure to **modify** the database or perform another specific action.	Stored procedures in Target components
I want to use stored procedures to **generate** one or more values/keys for an Insert statement in the same database.	Using stored procedures to generate primary keys

7.2.13.3 Stored procedures and local relations

By using local relations, you can define a hierarchical order in which to call stored procedures or perform actions (insert, update, ...) on database tables. They can be used in source and target components.

A local relation always has a parent object (containing a primary/unique key) and a child object (containing a foreign key).

Possible parent objects and their fields used in a relation are:

- Database table or view (column)
- Stored procedure (output parameter or return value)
- Recordset of a stored procedure (column) - only for source and procedure call components
- User-defined SELECT statement (column)

Possible child objects and their fields are:

- Database table or view (column, produces a WHERE condition)
- Stored procedure (input parameter)
- User-defined SELECT statement (input parameter)

In source components, this makes it easy to read data from related objects, e.g. read IDs from a database table and call a stored procedure with each of these IDs to retrieve related information. It is also possible to call a stored procedure with data retrieved from another procedure.

In target components, local relations allow defining a clear order in which multiple related procedures are to be called, e.g. one that creates an ID value, and another that inserts related

information into another table. It is also possible to mix stored procedures and tables in local relations, e.g. perform the insert directly on the related table instead of calling another procedure.

7.2.13.4 Stored procedures as a data source

The output of a stored procedure can be zero or more output or return parameters, and zero or more recordsets from SELECT statements embedded inside the stored procedure. A recordset or result set is the output of such a SELECT statement, similar to a table or view. Output parameters and recordsets can be mapped to target components.

The column structures of these recordsets cannot be directly read from the database catalog, they must therefore be detected by executing the stored procedure at design time or by being defined manually - see Defining recordsets for details.

Depending on whether the stored procedure has input parameters or not, the handling in MapForce is different:

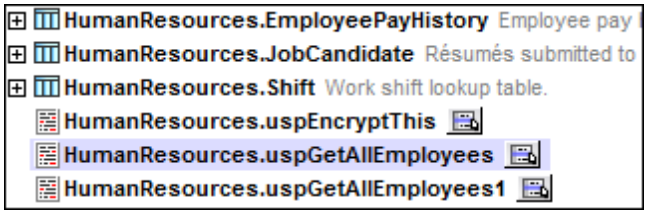
The stored procedure has no input parameters	Stored procedures without input parameters
I want to supply the values for the procedure's input parameters by mapping from an XML, Text, or other type of file, or from mapping input parameters or constants	Call with parameters - input and output
I want to supply the values for the procedure's input parameters from a table or view in the same database, or from the output of another stored procedure	Source components and Local Relations

Stored procedures without input parameters

Use this option (for example) if you want to use the stored procedure in a source component **without** having any **input** parameters.

E.g. this could be a stored procedure that is a pure SELECT-type query without any input parameters, where you want to map the result of the SELECT statement to a target component.

E.g. HumanResources.uspGetAllEmployees of the AdventureWorks database.



Stored procedure:

```
PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
```

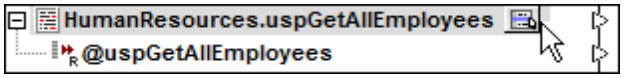


```
FROM HumanResources.vEmployeeDepartment;
```

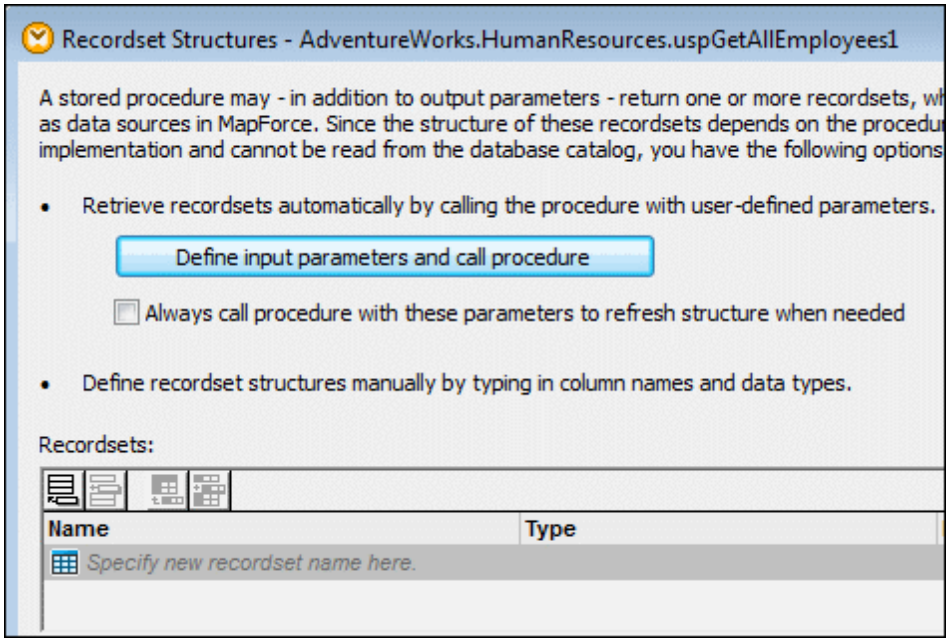
The columns of the recordset cannot be directly read from the database catalog by MapForce, they must therefore be detected by executing the stored procedure once or by being defined manually.

Defining the output recordset of a source component:
Having inserted the AdventureWorks database component and selected the HumanResources tables and included stored procedures:

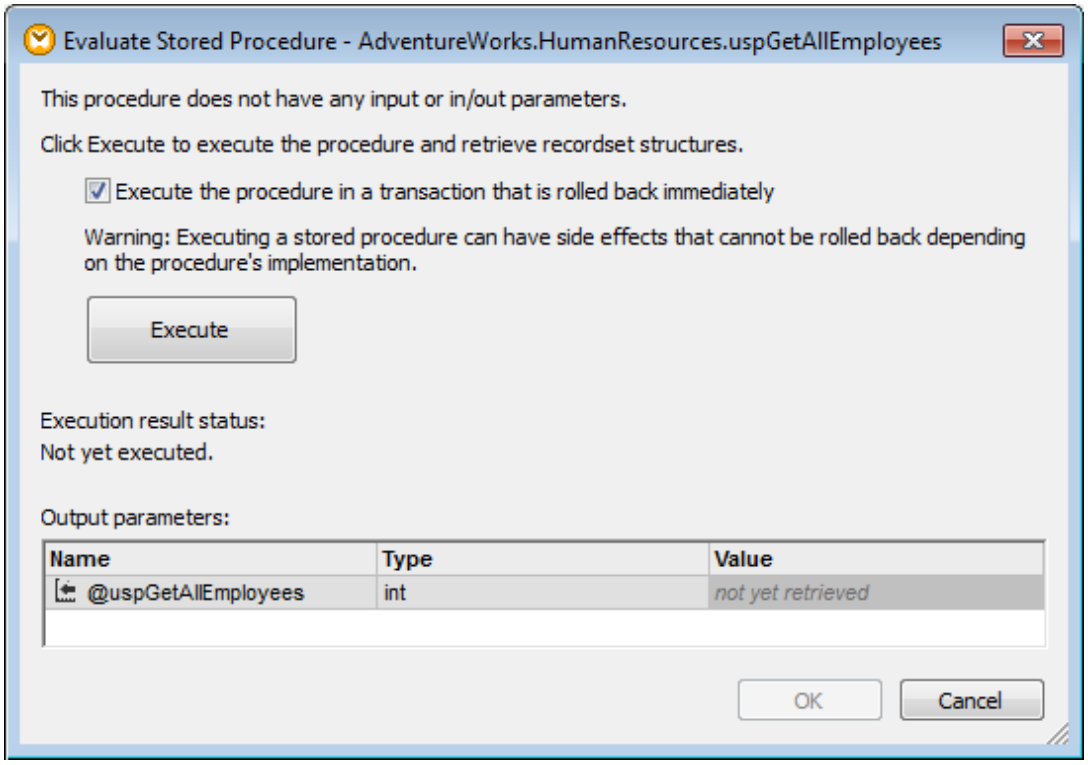
1. Click the "stored procedure" button and select the option "Show nodes as Source".



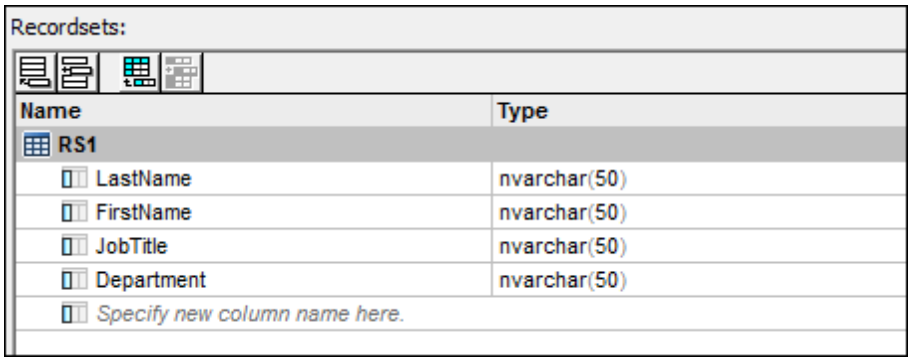
   The Return value node is inserted below the stored procedure name. Since we want to read the recordset and not the return value, click the stored procedure button again and select **Edit RecordSet structures**.
2. Click the "Define input parameters and call procedure" button.



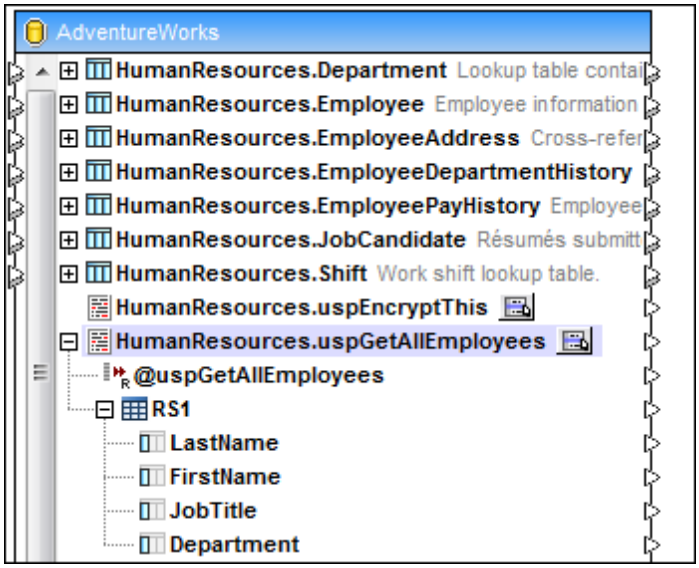
   This opens the "Evaluate Stored Procedure" dialog box.



3. Click the "Execute" button, then click OK.
   The recordset fields are now visible in the Recordsets section of the dialog box.

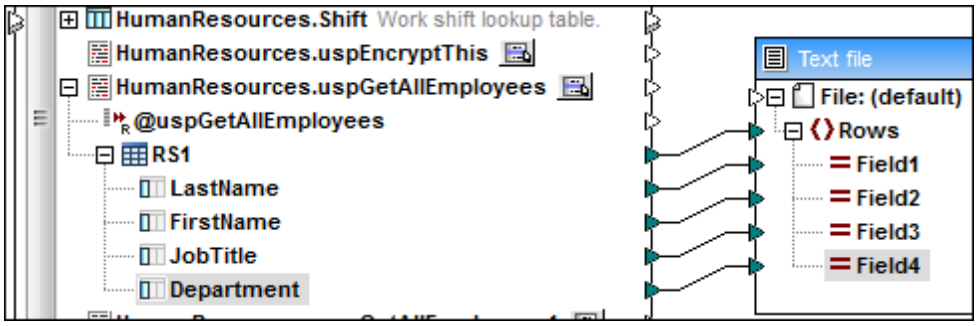


4. Click the OK button again to complete the recordset definition.
   The columns, LastName etc., are shown as nodes below the recordset node **RS1**. (Click the "+" button to expand the recordset if not visible).



Completing the mapping:

1. Insert a Text file component and map the output icons to the text file.



2. Click the Output button to see the result.

```
1   Gilbert,Guy,Production Technician - WC60,Production
2   Brown,Kevin,Marketing Assistant,Marketing
3   Tamburello,Roberto,Engineering Manager,Engineering
4   Walters,Rob,Senior Tool Designer,Tool Design
5   D'Hers,Thierry,Tool Designer,Tool Design
6   Bradley,David,Marketing Manager,Marketing
7   Dobney,JoLynn,Production Supervisor - WC60,Production
```

Note:
If executing the stored procedure has side effects (depending on the procedure implementation) that you want to avoid at design time, recordsets can be also be defined manually in the Recordset Structures dialog box, by adding recordsets and their associated columns. Click the Add recordset, or Add column buttons in the Recordset Structures dialog box.

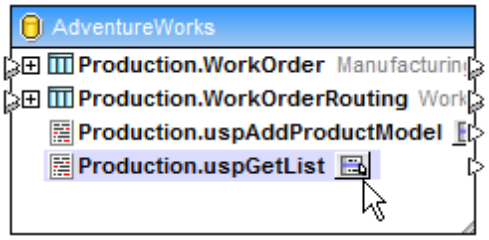
Call with parameters - input and output

Stored procedures can also be used as a function-like call. This allows you to:

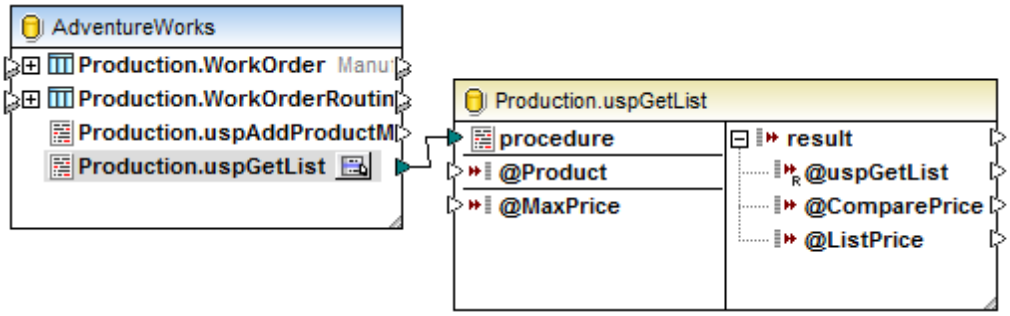
- provide input data to the procedure
- execute the procedure
- map the procedure output data to other components

To use a stored procedure as a function-like call:
Having inserted the AdventureWorks database component and selected the Production tables and included stored procedures:

1. Click the "stored procedure" button of Production.uspGetList, to open the menu.



2. Select the option "Insert Call with Parameters".

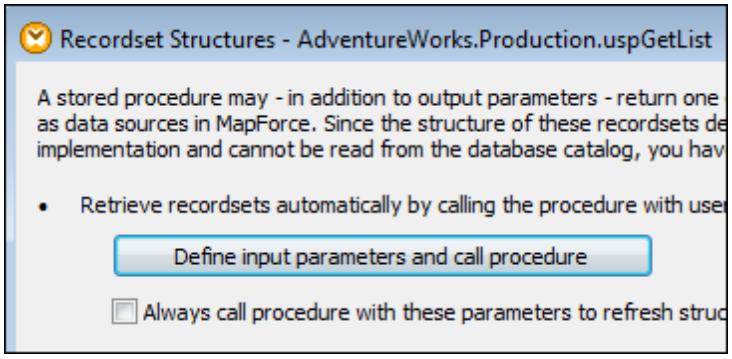


This inserts the procedure component into the mapping. The component looks and works similar to a web service, or user-defined, component. The procedure name is automatically connected to the "procedure" item of the component.

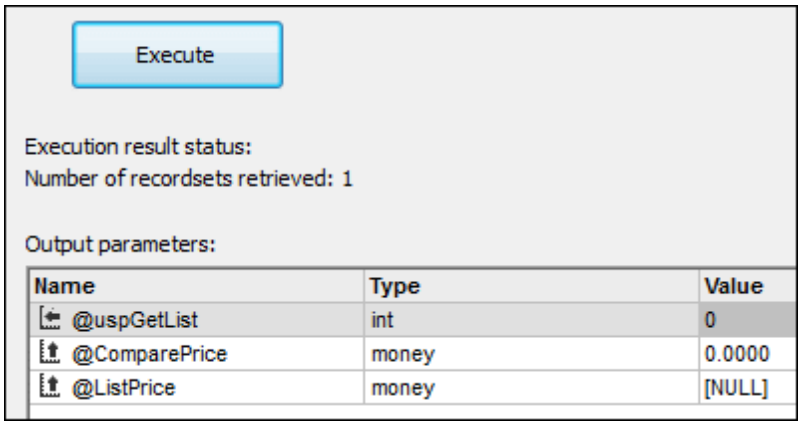
The procedure input parameters are shown on the left, while the output parameters are shown at right. This particular stored procedure returns output parameters and also a recordset, however we must define its structure before we can see and use it in MapForce:

To define the recordset structure:

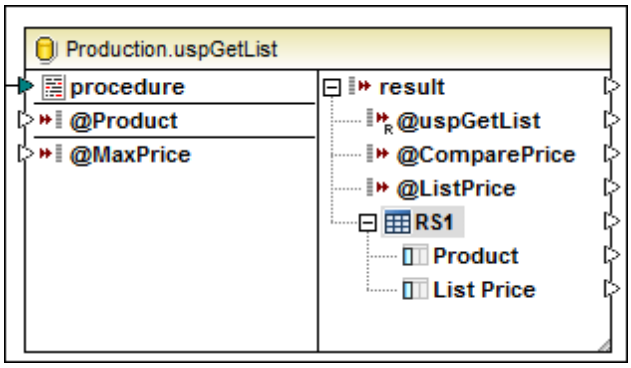
1. Click the "stored procedure" button, of uspGetList, and select "Edit recordset structures".
2. Click the "Define input parameters and call procedure" button, then click Execute in the dialog box that opens.



This writes the returned output parameter values into the table below and displays that one recordset was retrieved.



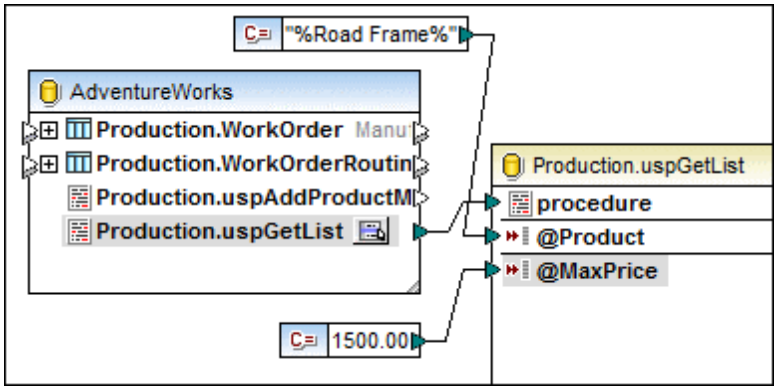
3. Click the OK button to confirm, then click OK to close the Recordset dialog box.



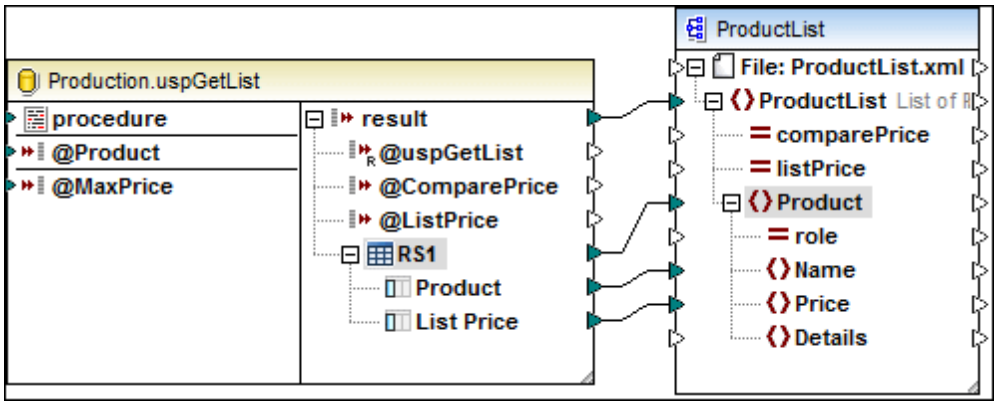
The recordset has been added to the output section of the stored procedure component.

Using the call parameter component:

1. Define the components you want to use to supply the input parameters, e.g. two constant components as shown in the screen shot, and connect them to the input parameters.



2. Define and insert the target component which will be used to contain the stored procedure output, e.g. an XML document as shown below.



3. Click the Output button to see the result of the mapping.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ProductList xsi:noNamespaceSchemaLocation="C:/Tools/sp/ProductList.xsd"
3    <Product>
4      <Name>HL Road Frame - Black, 58</Name>
5      <Price>1431.5</Price>
6    </Product>
7    <Product>
8      <Name>HL Road Frame - Red, 58</Name>
9      <Price>1431.5</Price>
10   </Product>
11   <Product>
12     <Name>HL Road Frame - Red, 62</Name>
13     <Price>1431.5</Price>
14   </Product>
15   <Product>
16     <Name>HL Road Frame - Red, 44</Name>
17     <Price>1431.5</Price>
```

The various road frame products are listed.

Source components and Local Relations

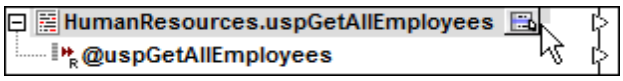
Use this option if you want to combine data supplied by a stored procedure recordset with data from another table, to which there is no direct relationship in the database.


```
PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT LastName, FirstName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment;
```

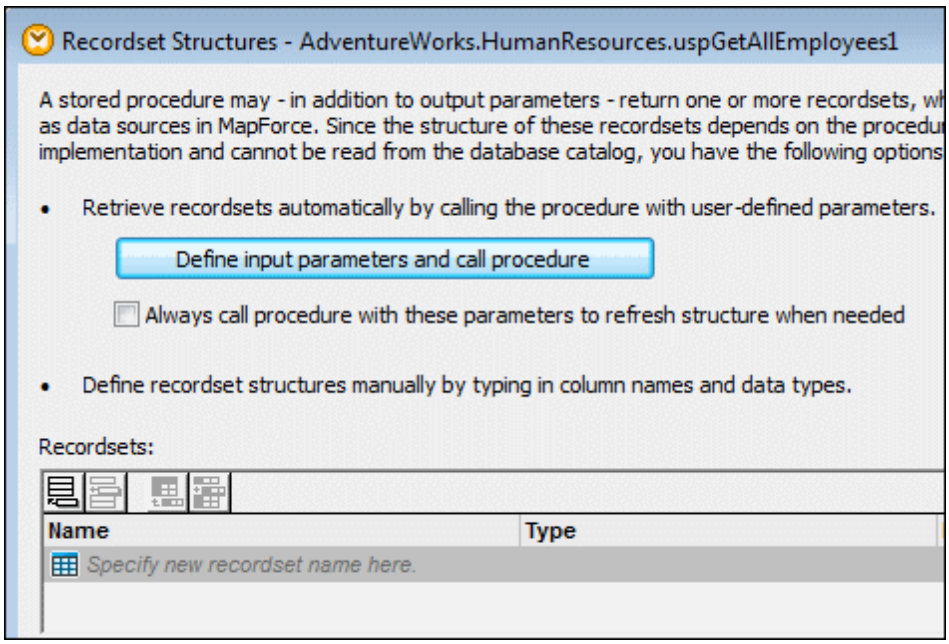
The columns of the recordset cannot be directly read from the database catalog by MapForce, they must therefore be detected by executing the stored procedure once or by being defined manually.

Defining the output recordset of a source component:
Having inserted the AdventureWorks database component and selected the HumanResources tables and included stored procedures:

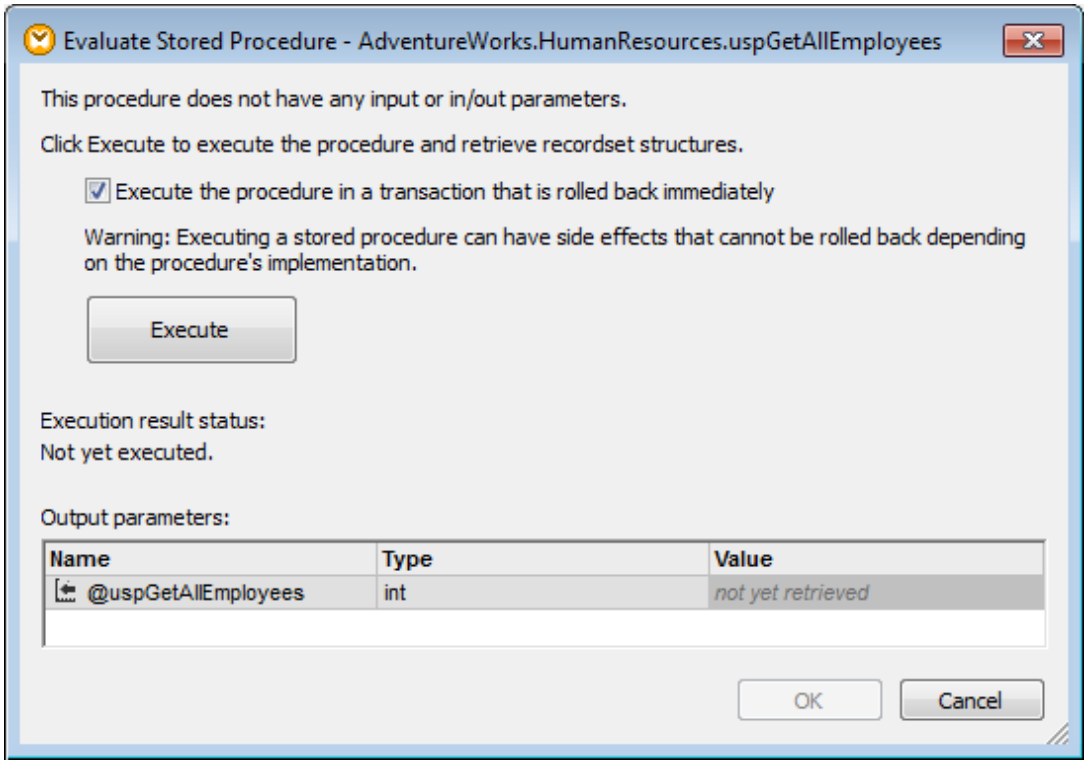
1. Click the "stored procedure" button and select the option "Show nodes as Source".



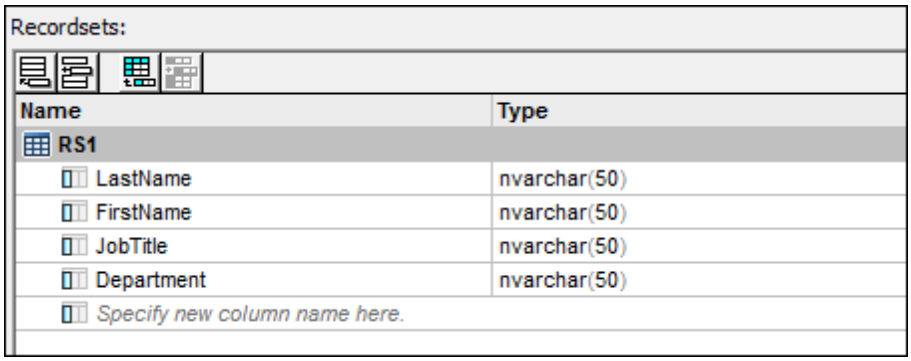
   The Return value node is inserted below the stored procedure name. Since we want to read the recordset and not the return value, click the stored procedure button again and select **Edit RecordSet structures**.
2. Click the "Define input parameters and call procedure" button.



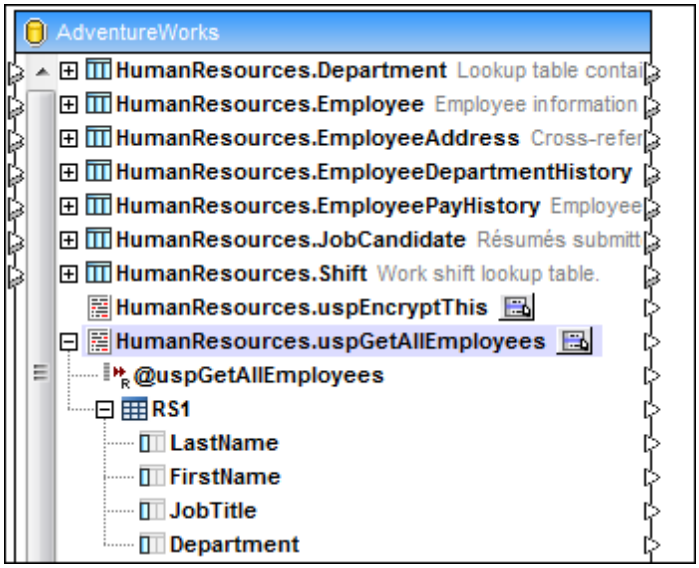
   This opens the "Evaluate Stored Procedure" dialog box.



3. Click the "Execute" button, then click OK.
   The recordset fields are now visible in the Recordsets section of the dialog box.

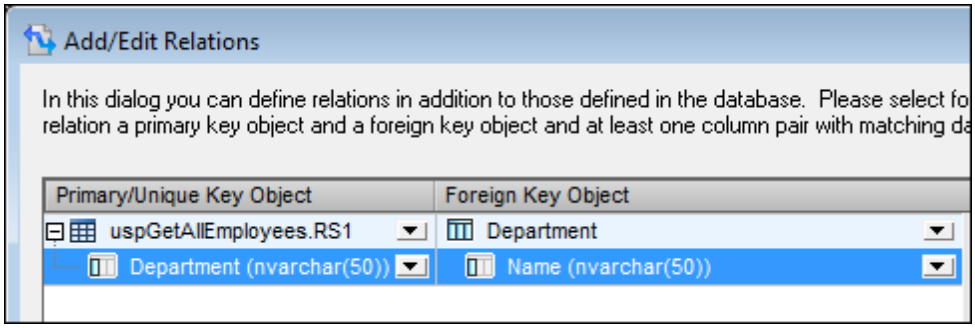


4. Click the OK button again to complete the recordset definition.
   The columns, LastName etc., are shown as nodes below the recordset node **RS1**. (Click the "+" button to expand the recordset if not visible).

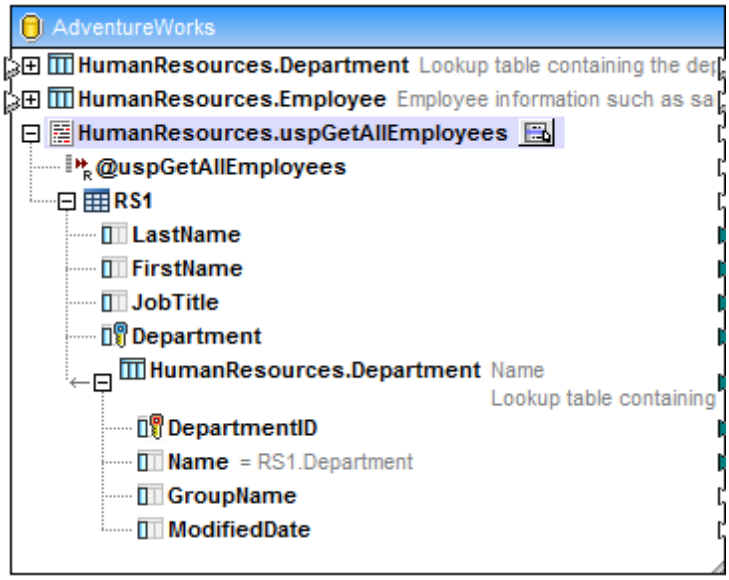


Defining a Local relation to a different table:

1. Right click the Component header, and select Add/Remove/Edit Database Objects.
2. Click the Add/Edit Relations button to open the Add/Edit Relations dialog box, then click the "Add Relation" button.
3. Define the Primary/Unique Key Object as the stored procedure **uspGetAllEmployees.RS1** and the column as the **@Department** parameter.
4. Define the Foreign Key Object as **Department** and the column as **Name**.



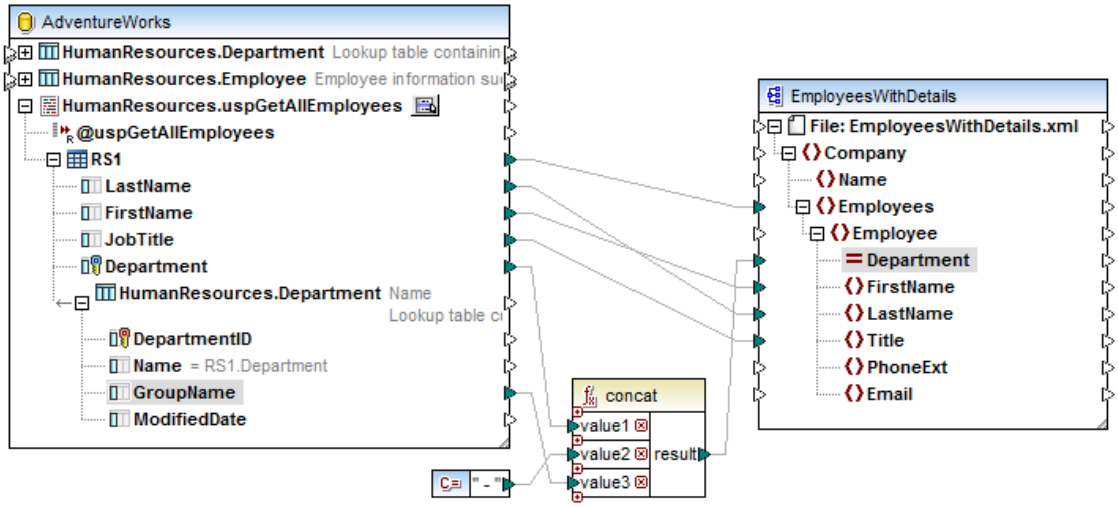
5. Click the OK button in the various dialog boxes.



The Department table is now displayed as a child of the stored procedure.

Completing the mapping

1. Insert the target schema to which you want to map the source database data, and add the connections as shown below.



2. Click the Output button to see the result.

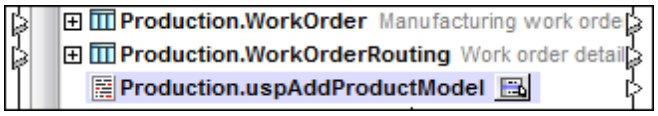

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xsi:noNamespaceSchemaLocation="C:/Users/Documents/Altova/Ma
3    <Employees>
4      <Employee Department="Production - Manufacturing">
5        <FirstName>Guy</FirstName>
6        <LastName>Gilbert</LastName>
7        <Title>Production Technician - WC60</Title>
8      </Employee>
9    </Employees>
10   <Employees>
11     <Employee Department="Marketing - Sales and Marketing">
12       <FirstName>Kevin</FirstName>
13       <LastName>Brown</LastName>
14       <Title>Marketing Assistant</Title>
15     </Employee>
16   </Employees>
17   <Employees>
18     <Employee Department="Engineering - Research and Development">
19       <FirstName>Roberto</FirstName>
20       <LastName>Tamburello</LastName>
21       <Title>Engineering Manager</Title>
22     </Employee>
```

7.2.13.5 Stored procedures in Target components

Choose this option when the stored procedure makes changes to the database, e.g. add/update/ delete etc., and you are not interested in any stored procedure output.

To use stored procedures in a target component:
This option adds the child nodes of the **input** parameters (as well as in/out parameters) under the stored procedure item in the target database component.

E.g.: You want to add a new product model to the database, using the uspAddProductModel stored procedure of the AdventureWorks database.



Stored procedure:

```
PROCEDURE Production.uspAddProductModel
@ModelName nvarchar(50),
@Inst xml

as
INSERT INTO [AdventureWorks].[Production].[ProductModel]
           ([Name]
           --,[CatalogDescription]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           --,<CatalogDescription, ProductDescriptionSchemaCollection,>
           ,@Inst
```

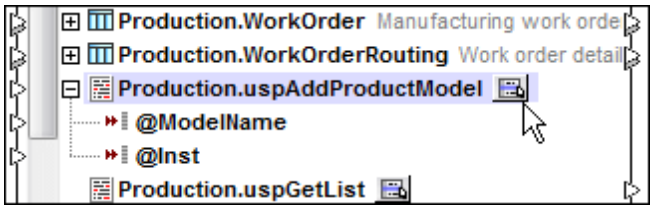
```
,NEWID()
,GETDATE());
```

At runtime, MapForce executes the stored procedure using all the mapped **input** parameters while ignoring the stored procedure data output.

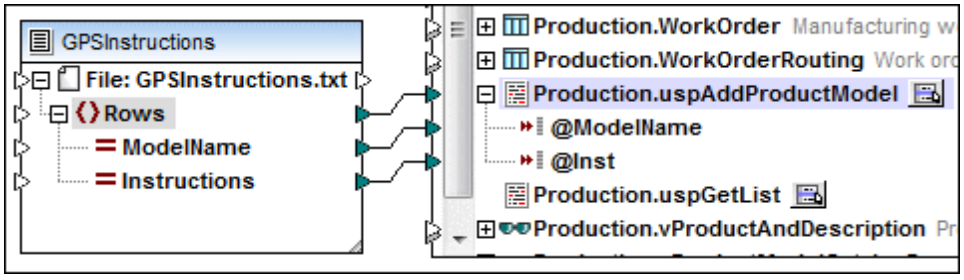
To create the input parameter items in a target component:

Having inserted the AdventureWorks database component and selected the Production tables and included stored procedures:

1. Click the "stored procedure" button and select the option "Show nodes as Target".

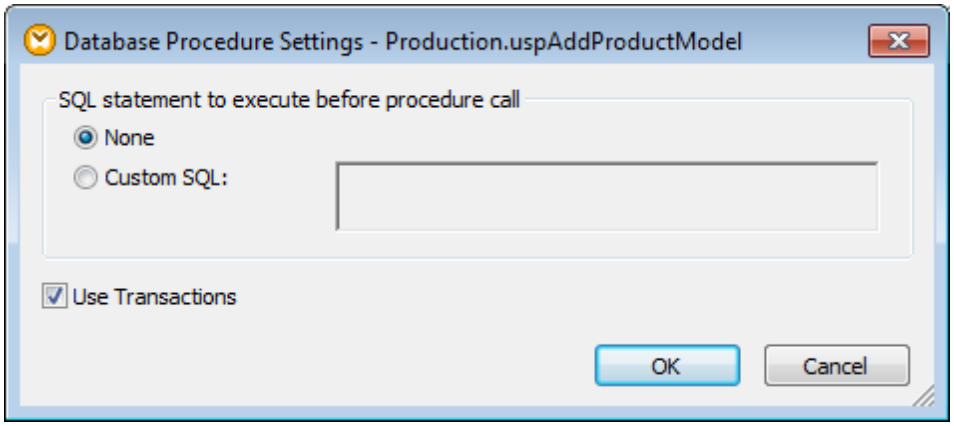


   This inserts the @ModelName and @Inst input parameters below the stored procedure name. Only the **input** icons of the input parameters are available in the target component.
2. Insert a source component, e.g. text file, XML file, etc., and map the items that are to supply the input parameter data, to the input icons of the stored procedure.

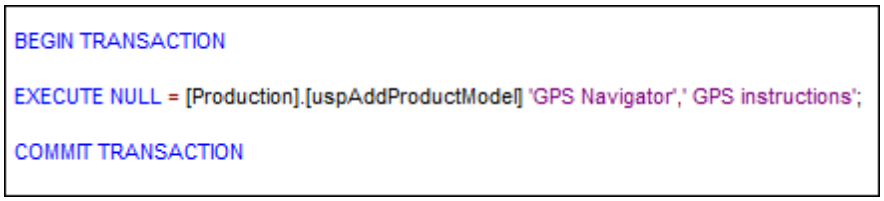


To define transactions for a stored procedure:

1. Click the "stored procedure" button and select the option "Procedure settings". This opens the Database Procedure Settings dialog box.



2. Click the "Use Transactions" check box and click OK to confirm.
   The transaction setting makes sure that the procedure commands can be rolled back if an error occurs during execution.
3. Click the Output button to see the commands that will be sent to the database.



   This dialog box also allows you to define SQL statements to be executed before the stored procedure is called.

Notes:
The "Add Duplicate input..." context menu options are disabled for the stored procedure **parameters**, as each parameter is an atomic value (and could also be "nullable").

The "Add duplicate input..." context menu options are however available for a stored **procedure** item. This would call the stored procedure for each duplicated item/node.

Using stored procedures to generate primary keys

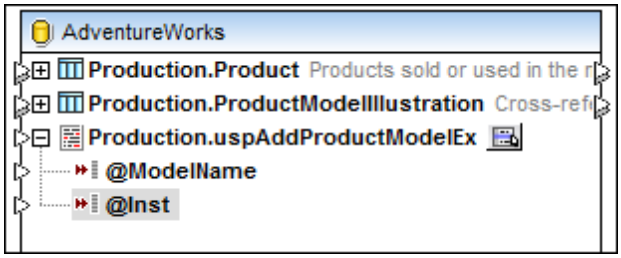
Choose this option when the stored procedure makes changes to a database table, and you also want to use the procedure output parameter to generate a primary key in a different table.

The uspAddProductModelEx procedure is a variation of the uspAddProductModel stored procedure in the AdventureWorks database.

```
procedure Production.uspAddProductModelEx
@ModelName nvarchar(50),
@Inst xml,
@ProductModelID int OUTPUT
as begin
INSERT INTO [AdventureWorks].[Production].[ProductModel]
           ([Name]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           ,@Inst
           ,NEWID()
           ,GETDATE());
   SELECT @ProductModelID = SCOPE_IDENTITY()
end;
```

Having inserted the AdventureWorks database component, selected the Production tables and included stored procedures:

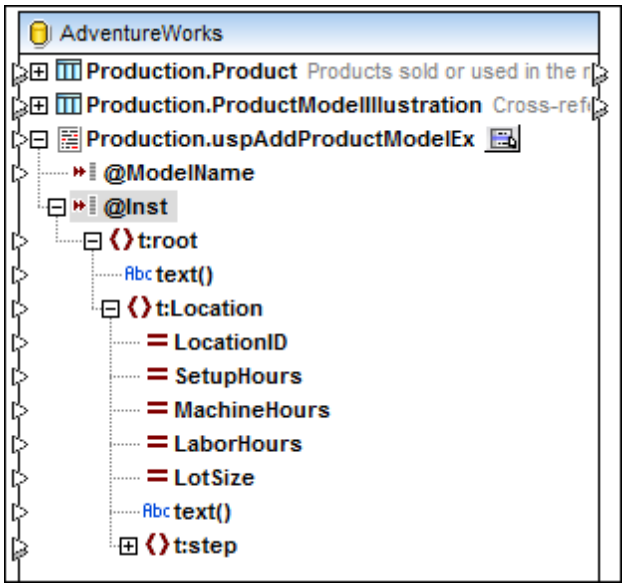
1. Click the "stored procedure" button and select the option "Show nodes as Target".



This inserts the ModelName and Inst parameters below the procedure name. Only the input parameters of the stored procedure are visible in the component.

As the Inst parameter is of type XML, we need to assign it a relevant XML Schema to supply the XML data.

2. Right click the Inst parameter and select "Assign XML Schema to field...".
3. Select the provided "Production.ManuInstructionsSchemaCollection in the "Database" combo box, and click OK.

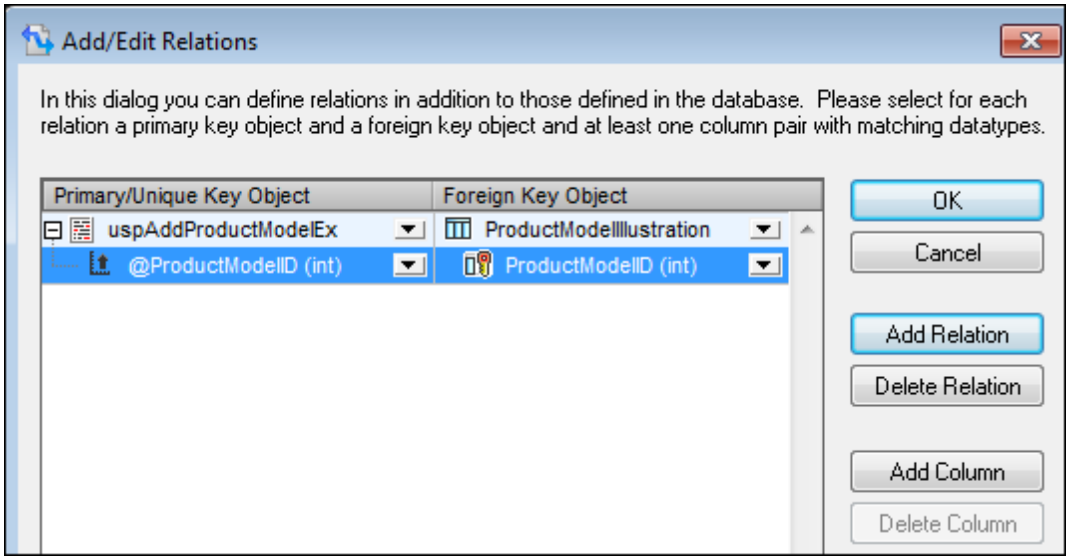


This adds the XML Schema elements and attributes to the component. The ModelName parameter and all the Inst parameters are now available in the component.

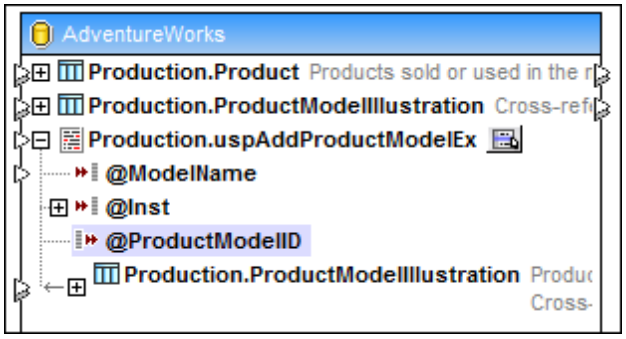
We now want to define a Local relation to a table that has no direct connection to the table referenced by the stored procedure parameters (production.product).

Defining a Local relation to a table in which you want to generate a primary key:

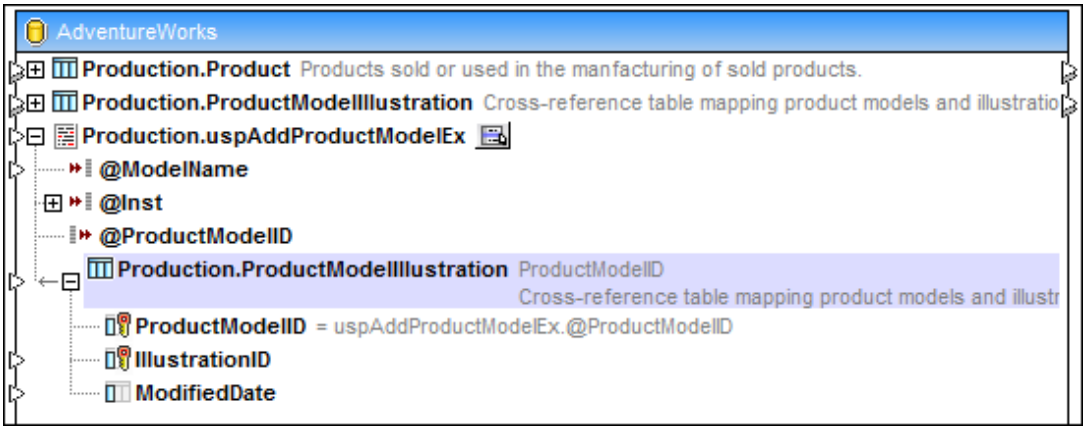
1. Right click the Component header, and select Add/Remove/Edit Database Objects.
2. Click the Add/Edit Relations button to open the Add/Edit Relations dialog box, then click the "Add Relation" button.
3. Define the Primary/Unique Key Object as the stored procedure **uspAddProductModelEx** and the column as the **@ProductModelID** parameter.
4. Define the Foreign Key Object as **ProductModelIllustration** and the column as **ProductModelID**.



5. Click the OK button in the various dialog boxes.

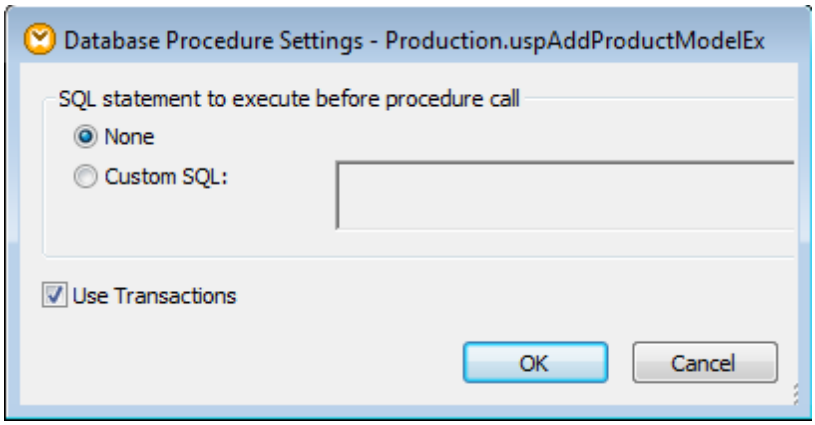


- The stored procedure output parameter (ProductModelID) has been added to the stored procedure as an indicator that it will be used in the local relation, but does not have any input or output icons.
- The table ProductModelIllustration has also been added as a child item to the stored procedure.
- Expanding the table shows the keys and columns of the table. Note that ProductModelID key shows the stored procedure and parameter name it is related to.
- Local relations that use the (output) recordset of the stored procedure, cannot be used here.
- Clicking the stored procedure button and selecting "Procedure Settings" allows you to define an SQL Statement to be run before the procedure is called, as well as activate transaction settings.



Defining a transaction for a stored procedure:

1. Click the stored procedure icon and select "Procedure Settings".
2. Click the Use Transactions check box, then click OK to confirm.

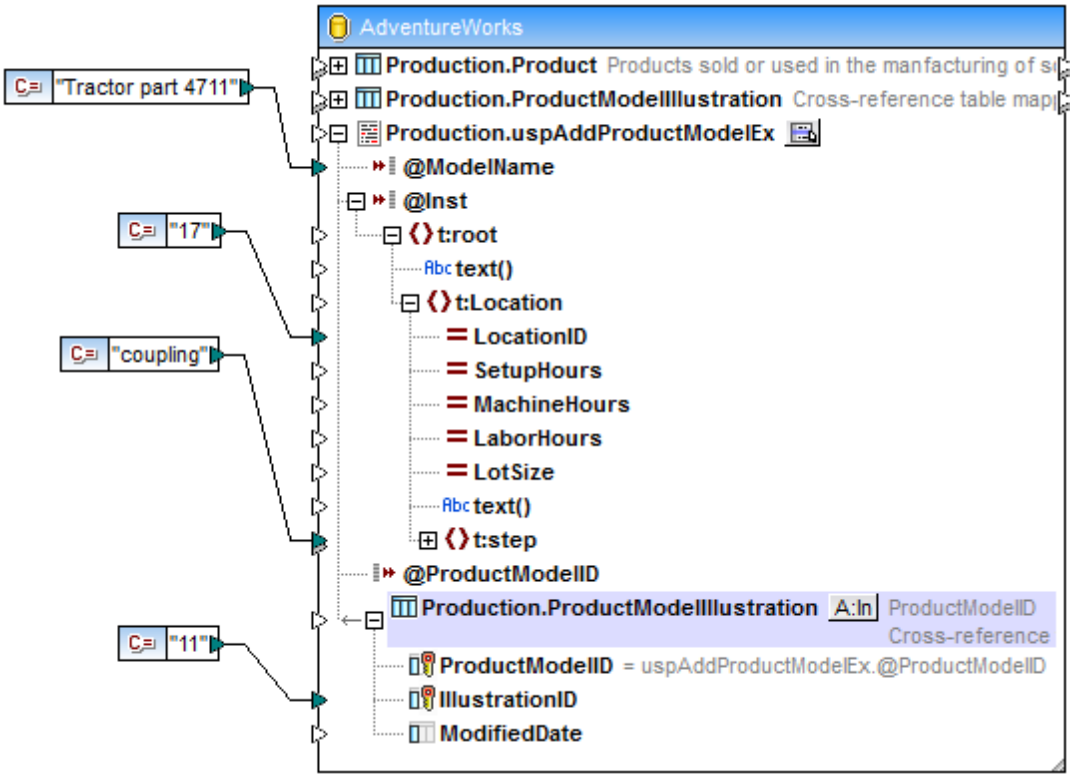


Defining the transaction for the stored procedure ensures that both retrieving the key and inserting the record both occur during the same transaction.

Completing the mapping:

The screenshot below shows only a subset of the data you would normally map.

1. Map the data source items to the target database; in this case constants.



2. Click the Output button to see the pseudo SQL that will be sent to the database.

```
BEGIN TRANSACTION

EXECUTE NULL = [Production].[uspAddProductModelEx] 'Tractor part 4711','<root
xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/ProductModelManuInstructions
"><Location LocationID="17"><step>coupling</step></Location></root>',NULL OUT;
-->>> %@uspAddProductModelEx1%
-->>> %@ProductModelID1%

INSERT INTO [Production].[ProductModelIllustration] ([ProductModelID], [IllustrationID]) VALUES (
'%@ProductModelID1%', 11)

COMMIT TRANSACTION
```

MapForce automatically calls the stored procedure for each record before the Insert action.

7.2.14 Mapping multiple tables to one XML file

Mapping multiple hierarchical tables to one XML output file

- You have a database and want to extract/map a certain number of tables into an XML file.
- Primary and foreign-key relationships exist between the tables
- Related tables are to appear as child elements in the resulting XML file.

The "**DB_Altova_Hierachical.mfd**" sample file in the **...\MapForceExamples** folder shows how this can be achieved when mapping from an hierarchical database. The Altova_Hierarchical.xsd

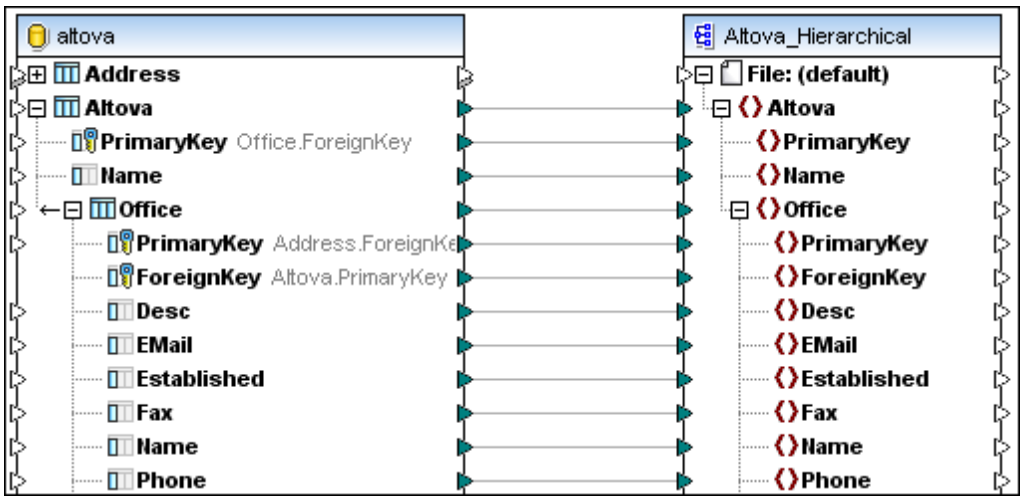
schema is also supplied in the same folder. The schema structure is practically identical to the Access database hierarchy. (The same method can also be used to map flat format XML/SQL databases.)

The MS Access database, **Altova.mdb**, is supplied in the ...\MapForceExamples\Tutorial\ folder.

Schema prerequisites:

- All tables related to Altova, appear as child items of the target root element.
- To preserve the table relationships all mappings have been created under the Altova table in the database component.

  The diagram below shows the mapping of the hierarchical Access database to Altova_Hierarchical.xsd.

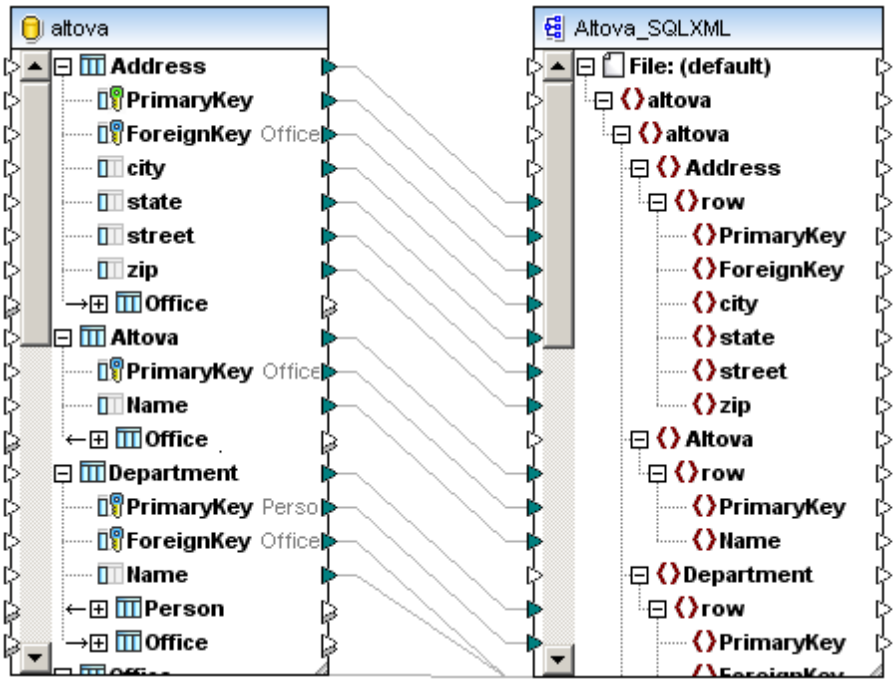


Mapping multiple flat file tables to one XML output file

The following diagram shows the same type of mapping to a flat file SQL/XML database schema.

Schema prerequisites:

- The **schema** structure has to follow the SQL/XML specifications.
- XMLSpy has the ability to create such an SQL/XML conformant file from an SQL database, by using the menu option **Convert | Create Database Schema**. You can then use the **schema** as the target in MapForce.
- In this case each table name is mapped to the row child element, of the same element name in the schema, i.e. Address is mapped to the **row** child element of the **Address** element

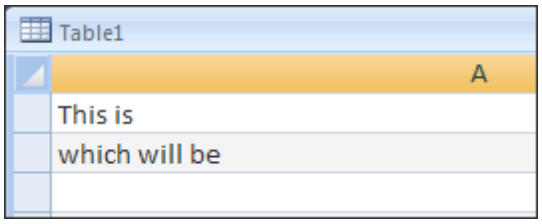


- Please note that the above example **DB_Altova_SQLXML.mfd**, does not preserve the table relationships, as mappings are created from several different "root" tables.

7.2.15 Replacing special characters in database data

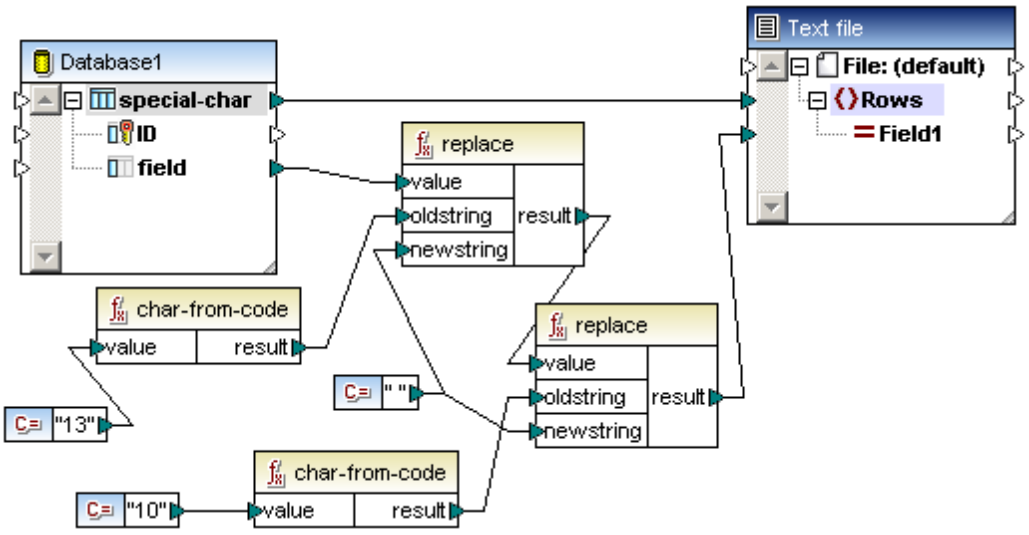
When transforming database data you might need to remove specific "special" characters e.g. newline (line break), CR/LF (hex 0D/0A) from the data source. This can be achieved by using the char-from-code function of the string function library.

In the mapping shown below the data source is an MS Access database consisting of a single table with two rows. The text of each of the row has multiple lines separated by line feeds (LF) hex 0D, decimal 13.



What the mapping does is:

- Convert the 0D and 0A hex characters (13 and 10 decimal) to a string to allow further processing by the replace function.
- Use the replace function to replace the oldstring parameter, supplied by char-from-code (0D/0A), with the "space" character supplied by the constant component as the newstring parameter.
- Place the converted data into a text component.



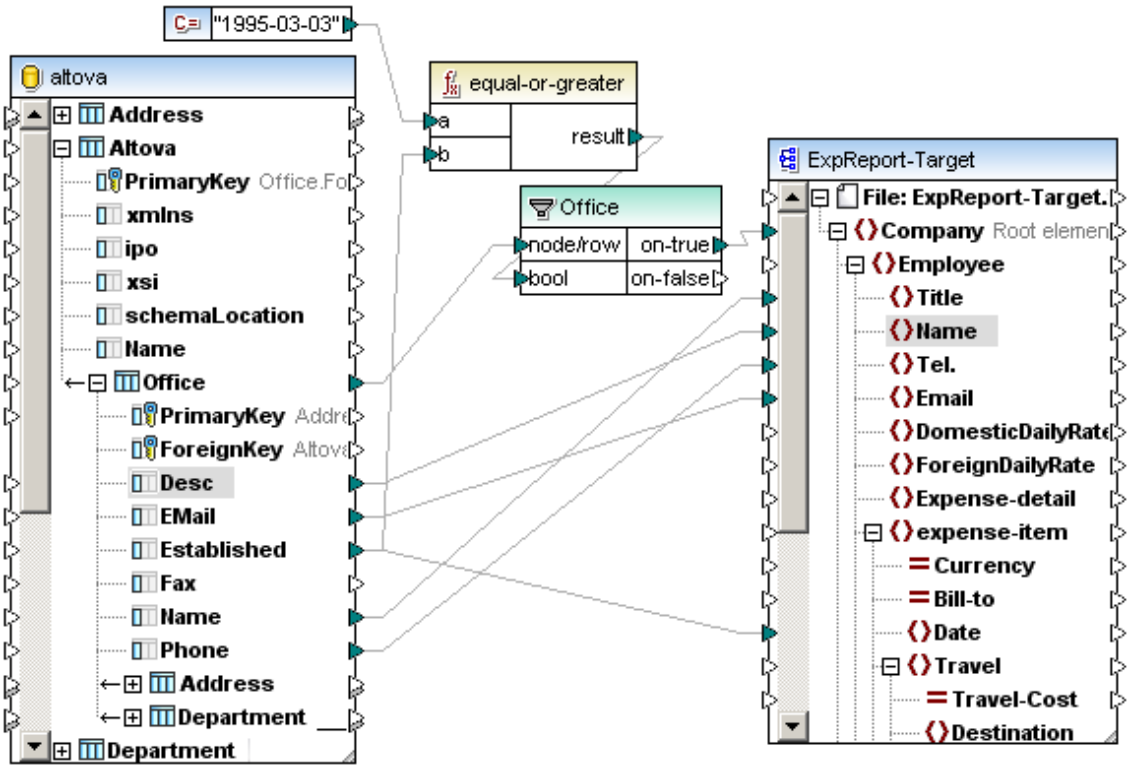
Clicking the Output tab shows the result in the preview window. The (CR) LF characters within each database field have been replaced by a space.

```
1   This is  our new  company policy
2   which will be  implemented immediately.
3
4
5
6
```

7.2.16 Filtering database data by date

The example below shows how you can use the filter component to filter database records according to a specific date.

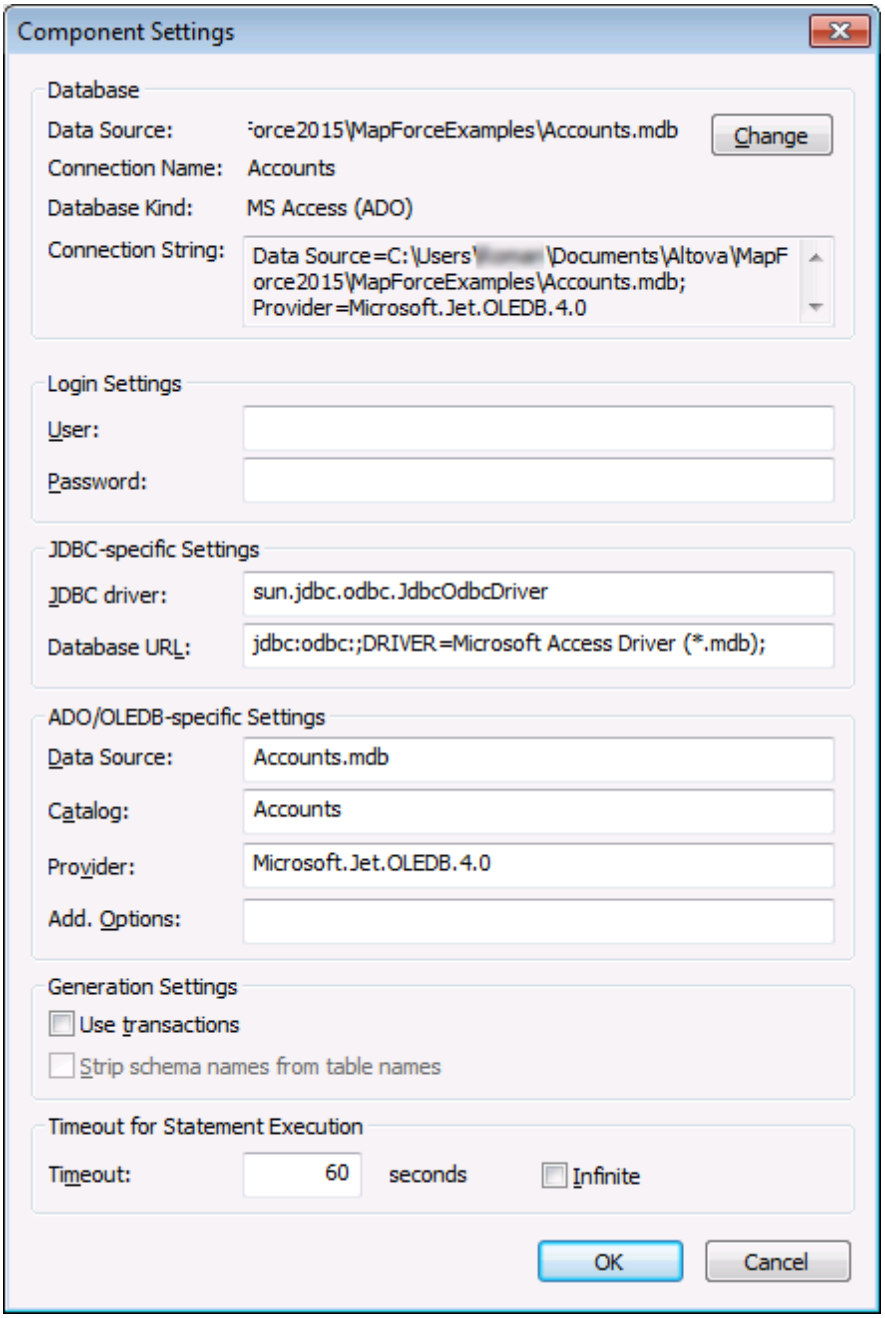
- The Established field is defined as a Date/Time field in the database.
- The comparison date is entered into a Constant component.
- If the date record is greater than 1995-03-03, only then are the respective Office data passed on to the target file by the filter component. Note: use the "**All other**" datatype for the constant component.



7.2.17 Database Component Settings

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

- Select the component, and then, on the **Component** menu, click **Properties**.
- Double-click the component.
- Right-click the component, and then click **Properties**.



Database Component Settings dialog box

The available settings are as follows.

Database

This group displays database connection information. Click **Change** to select a different database, or to redefine the database objects in the existing database component. Connectors to tables of the same name will be retained. You can also change the tables in the component, by right clicking a database component and selecting **Add/Remove/Edit Database Objects**.

Data Source	Specifies the name of the current data source. For file-based databases, this can be a path on the file system.
Connection Name	Specifies the name of the connection (this is the same as the data source name if the database uses a data source)
Database Kind	Specifies the kind of the database.
Connection String	Displays the current database connection string. This read-only field is generated based on the information you supply when creating or changing the database connection.

Login Settings

The login settings are used for all code generation targets and the built-in execution engine.

User	Enables you to change the user name for connecting to the database. Mandatory if the database requires a user name to connect.
Password	Enables you to change the password for connecting to the database. Mandatory if the database requires a password to connect.

JDBC-specific Settings

JDBC specific settings are only used for Java code generation.

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

ADO/OLEDB settings are only used for C++ and C# code generation. The Datasource and Catalog settings are not used by the built-in execution engine.

Data Source	Displays the name of the ADO data source.
Catalog	Displays the name of the ADO catalog.
Provider	Displays the currently active provider for the database component.
Add. Options	Displays any additional database options.

Generation Settings

Generation settings apply to all code generation targets as well as the built-in execution engine.

Use transactions	Enables transaction processing when using a database as a target. A dialog box opens when an error is encountered allowing you to choose how to proceed. Transaction processing is enabled for all tables of the database component when you select this option.
Strip schema names from table names	Allows you to strip database schema names from generated code, only retaining the table names for added flexibility. Note that this option only works for SQL Select statements generated by MapForce. User-defined SQL-Statements, when creating virtual tables, will not be modified.

Timeout for Statement Execution

When a database is used as a target component, execution timeouts can occur due to server availability, traffic, long-running triggers, and other factors. This setting allows you to define how long the timeout period can be before the database connection is closed. The setting takes effect when querying database data as well as in generated C#, Java, and C++ code.

Timeout	Defines the time period, in seconds, that the execution engine must wait for a database response before aborting the execution of the database statement. The default setting for the execution timeout is 60 seconds.
Infinite	When enables, this option instructs the execution engine to never time out.

Note: Timeout for statement execution is not applicable to SQLite databases.

7.3 CSV and Text Files

MapForce includes support for mapping data to or from text-based file formats such as CSV (comma-separated values) and FLF (Fixed-Length Field) text files. For example, you can create data transformations such as:

- XML schema to/from flat file formats
- Database to/from flat file formats
- UN/EDIFACT and ANSI X12 to/from flat file formats or databases

Note that, in case of CSV, your files can have as delimiter not only commas, but also tabs, semicolons, spaces, or any other custom values.

In addition to CSV and FLF files, mapping to or from text files with more complex or custom structures is possible using MapForce FlexText (this module is available in MapForce Enterprise Edition). FlexText essentially enables you to define the structure of your custom text data (using a so-called "FlexText template"), for the purpose of mapping it to other formats. For more information, see MapForce FlexText.

Mapping data to or from text files is supported in any one of the following languages: Java, C#, C++, or BUILT-IN.

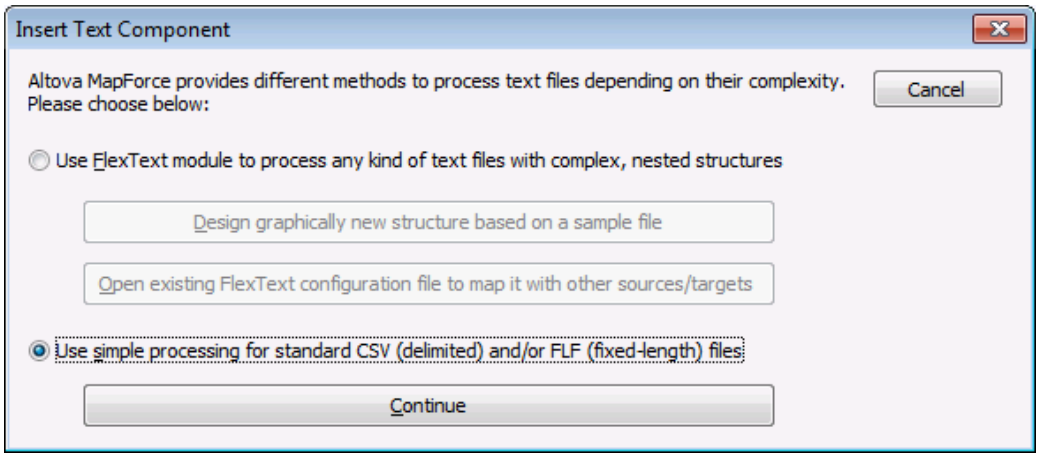
There are two ways that mapped flat file data can be generated:

- By clicking the Output tab which generates a preview using the Built-in execution engine. You can also save the mapping result by selecting the menu option **Output | Save output file**, or clicking the  icon.
- By selecting **File | Generate code in | Java, C#, or C++** , and then compiling and executing the generated code.

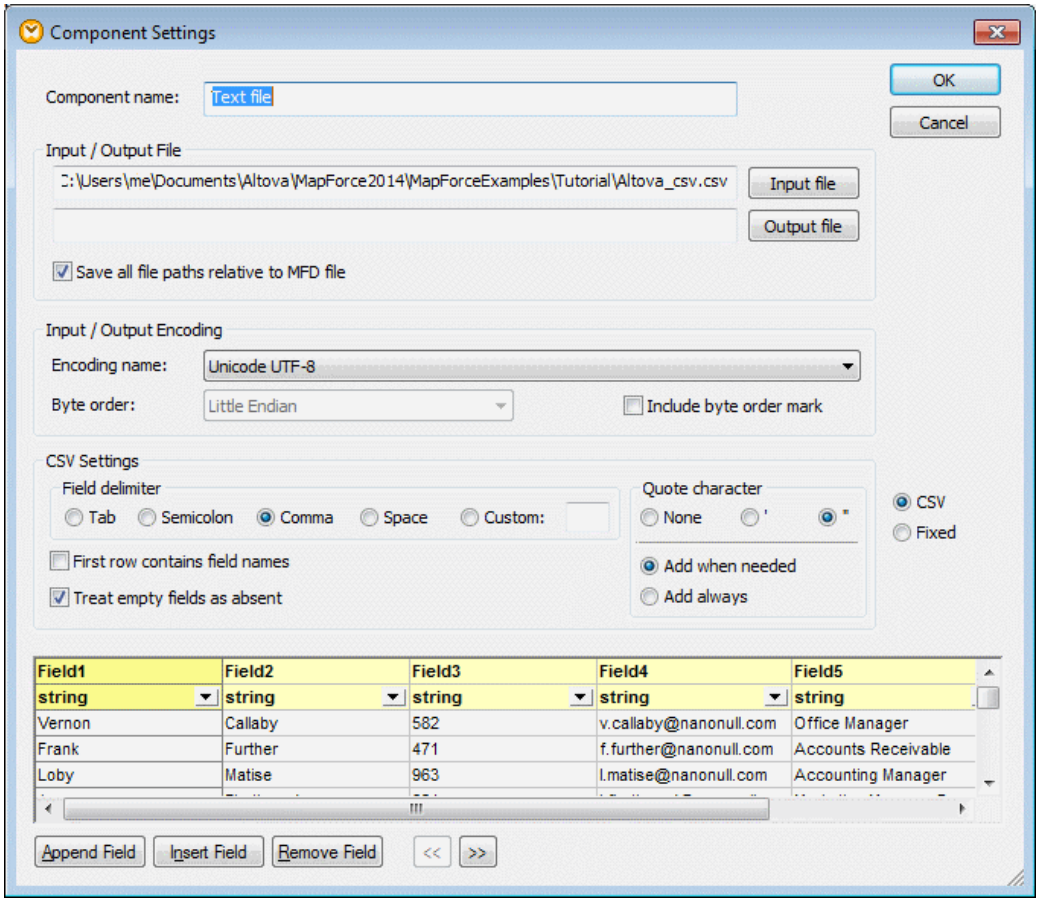
7.3.1 Example: Mapping CSV Files to XML

The goal of this example is to create a mapping which reads data from a simple CSV file and writes it to an XML file. The files used in the example are available in the **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial** folder.

1. Select one of the following as transformation language: Java, C#, C++, or BUILT-IN.
2. Add a Text file component to the mapping area (on the **Insert** menu, click **Text File**, or click the **Insert Text file** toolbar button ().



3. Select the **Use simple processing** ... option, and then click **Continue**.
4. On the Component Settings dialog box, click **Input file** and browse for the **Altova_csv.csv** file. The file contents are now visible in the lower part of the dialog box. Note that only the first 20 rows of the text file are displayed when in preview mode.

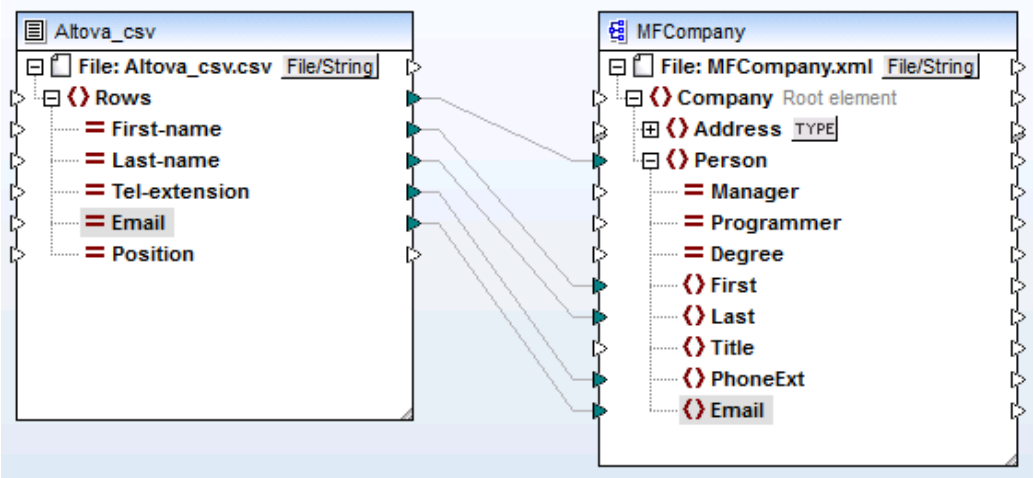


5. Click inside the **Field1** header and change the text to First-name. Do the same for all the other fields, as follows: Field 2 => Last-name, Field 3 =>Tel-extension, Field 4 => Email, Field 5 => Position. TIP: Press the **Tab** key to cycle through all the fields: header1, header2 etc.

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

6. Click OK.
7. When prompted to change the component name, click "Change component name". The CSV component is now visible in the mapping.
8. Add **MFCompany.xsd** as the target XML component of the mapping (on the **Insert** menu, click **XML/Schema file**).
9. Click **Skip** when prompted to supply a sample XML file, and select `Company` as the root element.
10. Map the corresponding items of both components, making sure to map the **Rows** item to the **Person** item in the schema target.

> The connector from the `Rows` item in the CSV component to the `Person` item in the schema is essential, as it defines which elements will be iterated through. That is, for each row in the CSV file, a new `Person` element will be created in the XML output file.



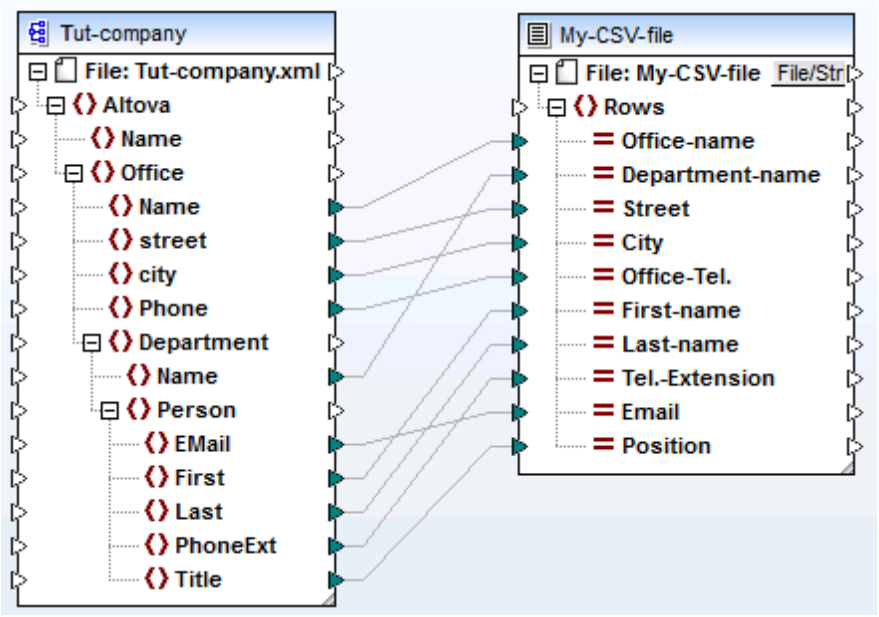
11. Click the Output tab to see the result.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
3    <Person Manager="true">
4      <First>Vernon</First>
5      <Last>Callaby</Last>
6      <PhoneExt>582</PhoneExt>
7      <Email>v.callaby@nanonull.com</Email>
8    </Person>
9    <Person Manager="true">
10     <First>Frank</First>
11     <Last>Further</Last>
12     <PhoneExt>471</PhoneExt>
13     <Email>f.further@nanonull.com</Email>
14   </Person>
15   <Person Manager="true">
```

The data from the CSV file is now successfully mapped to an XML file.

7.3.2 Example: Iterating Through Items

This example illustrates how to create iterations (multiple rows) in a target CSV file. The mapping design file accompanying this example is available at the following path: **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial\Tut-xml2csv.mfd**.

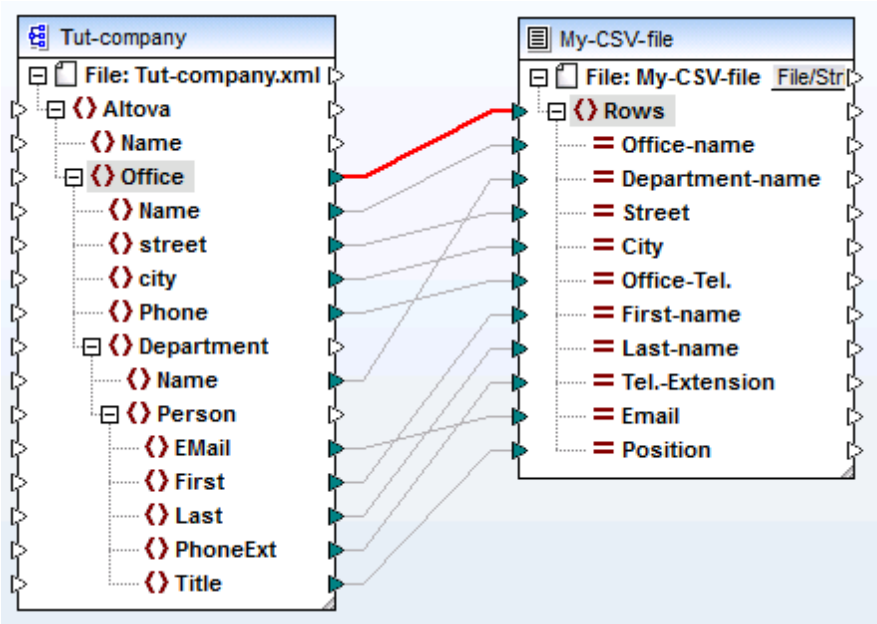


Tut-xml2csv.mfd

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.

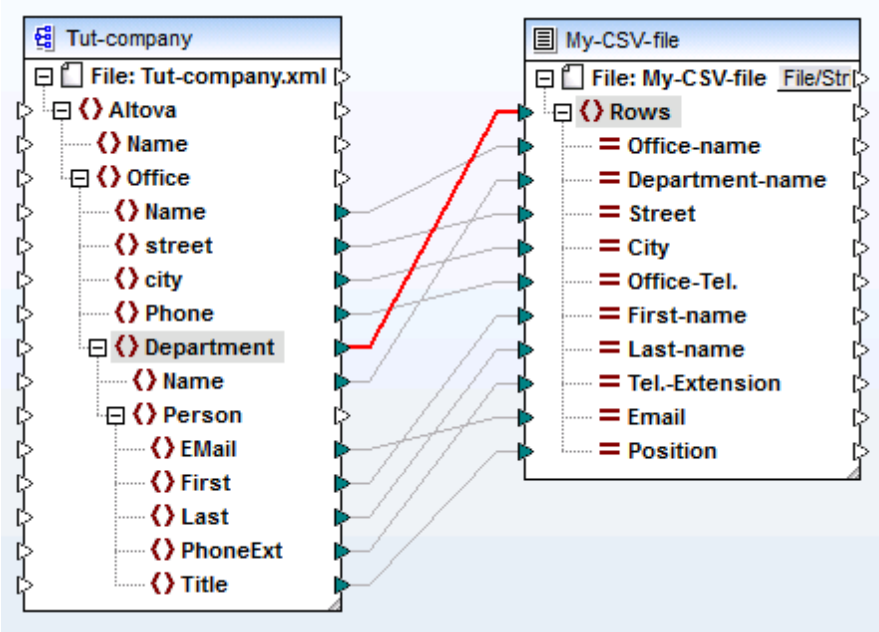
For example, to iterate through all offices and have the output appear in the CSV file, it is necessary to connect `Office` to `Rows`. By doing this, you are instructing MapForce: for each `Office` item of the source XML, create a row in the target CSV file.



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.

```
1   "Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,558833
2   "Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Otto
3
```

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:

```
1  "Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Clive,Clo
2  "Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,558
3  "Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,K
4  "Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,5588339
5  "Microtech Partners, Inc.",Admin,Perro Bvd 1324,Ottowa,3549202,
6  "Microtech Partners, Inc.",Sales and Marketing,Perro Bvd 1324,O
7  "Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Ottow
8
```

Finally, mapping `Person` to the `Rows` item results in all the Persons being output. In this case, MapForce will iterate through the records as follows: each Person within each Department, within each Office.

7.3.3 Example: Creating Hierarchies from CSV and Fixed-Length Text Files

This example is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\Tut-headerDetail.mfd**. The example uses a CSV file (Orders.csv) which has the following format:

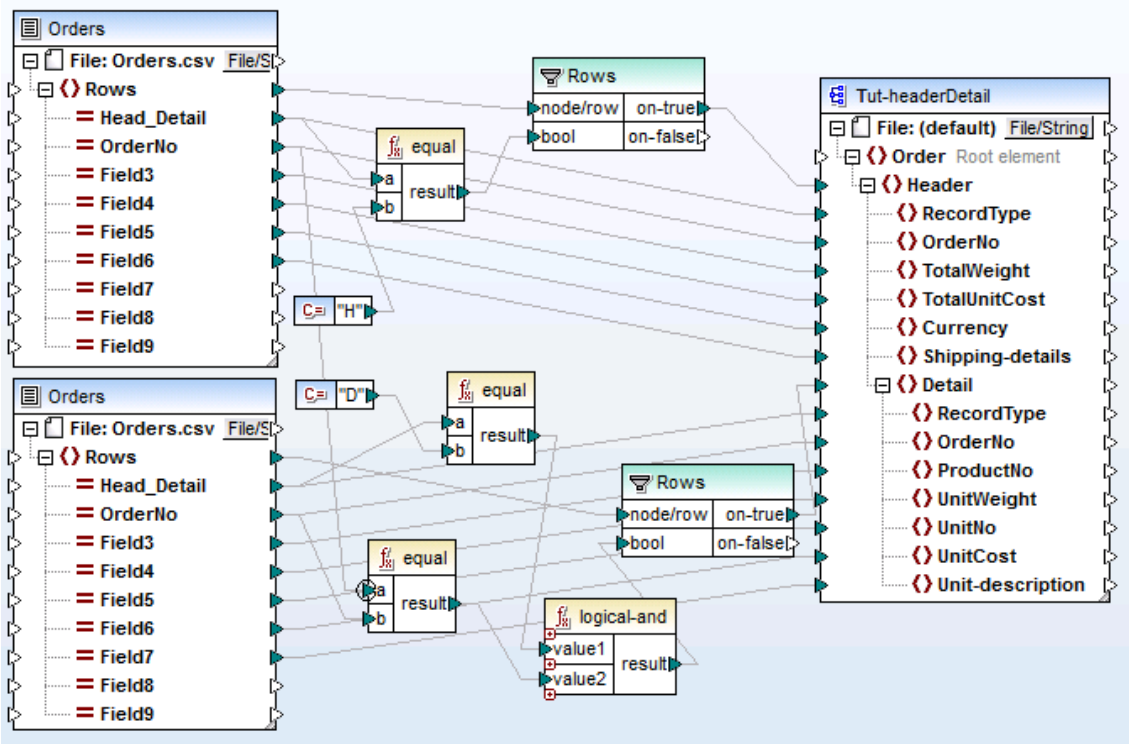
- Field 1: H defines a header record and D a detail record.
- Field 2: A common key for both header and detail records.
- Each Header or Detail record is on a separate line.

The contents of the Orders.csv file are shown below.


```
H,111,332.1,22537.7,,Container ship,,,
D,111,A-1579-227,10,3,400,Microtome,,
D,111,B-152-427,7,6,1200,Miscellaneous,,
H,222,978.4,7563.1,,Air freight,,,
D,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,,
```

The aim of the mapping is as follows:

- Map the flat file CSV to an hierarchical XML file
- Filter the Header records, designated with an H
- Associate the respective detail records, designated with a D, with each of the header records.



tut-headerDetail.mfd

For this to be achieved, the header and detail records must have one common field. In this case the common field, or key, is the second field of the CSV file, i.e. OrderNo. In the CSV file, both the first header record and the following two detail records contain the common value 111.

The Orders.csv file has been inserted twice to make the mapping more intuitive.

The **Tut-headerDetail.xsd** schema file has a hierarchical structure: Order is the root element, with Header as its child element, and Detail being a child element of Header.

The first Orders.csv file supplies the **Header** records (and all mapped fields) to the Header item in

the schema target file. The filter component is used to filter out all records other than those starting with H. The **Rows** item supplies these filtered records to the Header item in the schema file.

The second Orders.csv file supplies the **Detail** records (and all mapped fields) by filtering out the Detail records that match the OrderNo key of the Header record. This is achieved by:

- Comparing the **OrderNo** field of the Header record with the same field of the Detail records, using the **equal** function (the priority context is set on the **a** parameter for enhanced performance).
- Using the **Logical-and** function to only supply those Detail records containing the same OrderNo field, as the Header record.

The **Rows** item supplies these filtered records to the Header and Detail items in the schema file, through the on-true parameter of the filter component.

Clicking the Output tab produces the XML file displayed below. Each Header record contains its data, and all associated Detail records that have the same Order No.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation
3    <Header>
4      <RecordType>H</RecordType>
5      <OrderNo>111</OrderNo>
6      <TotalWeight>332.1</TotalWeight>
7      <TotalUnitCost>22537.7</TotalUnitCost>
8      <Currency/>
9      <Shipping-details>Container ship</Shipping-details>
10     <Detail>
11       <RecordType>D</RecordType>
12       <OrderNo>111</OrderNo>
13       <ProductNo>A-1579-227</ProductNo>
14       <UnitWeight>10</UnitWeight>
15       <UnitNo>3</UnitNo>
16       <UnitCost>400</UnitCost>
17       <Unit-description>Microtome</Unit-description>
18     </Detail>
19     <Detail>
20       <RecordType>D</RecordType>
21       <OrderNo>111</OrderNo>
22       <ProductNo>B-152-427</ProductNo>
23       <UnitWeight>7</UnitWeight>
24       <UnitNo>6</UnitNo>
25       <UnitCost>1200</UnitCost>
26       <Unit-description>Miscellaneous</Unit-description>
27     </Detail>
28   </Header>
```

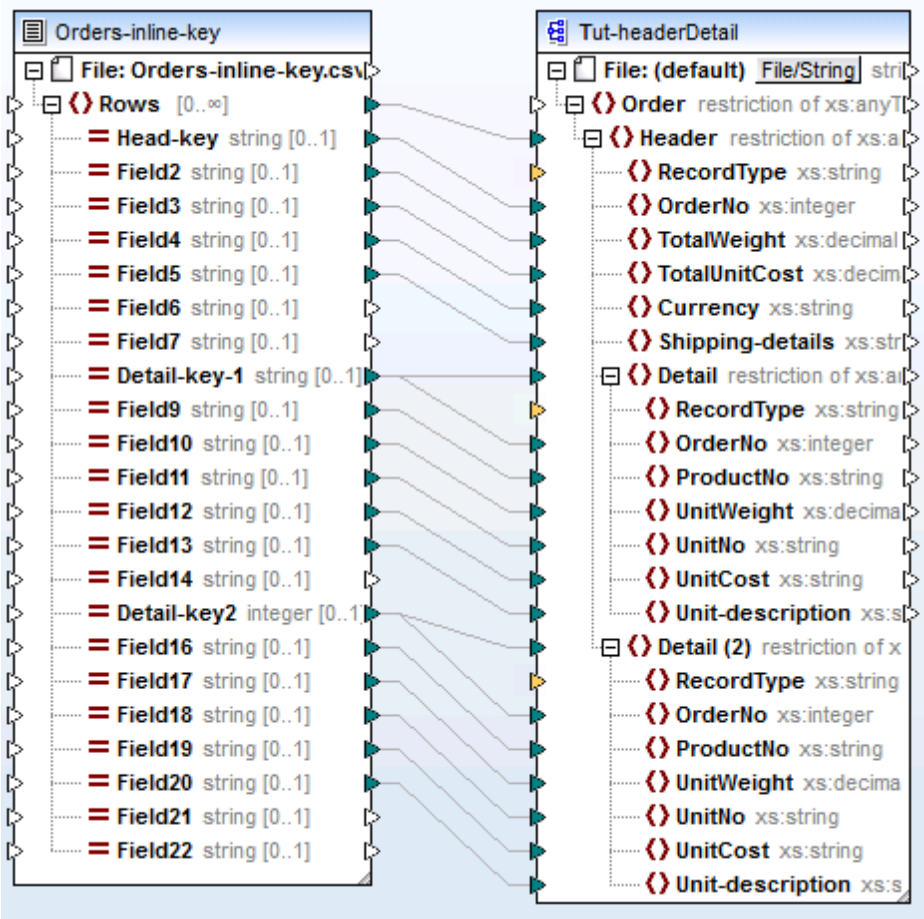
Let's now have a look at another example, which uses a slightly different CSV file and is available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder as **Head-detail-inline.mfd**. The difference is that:

- No record designator (H, or D) is available
- A common key field, the first field of the CSV file, still exists for both header and detail records (Head-key, Detail-key…). The field is mapped to OrderNo in the schema target
- Header and all respective Detail fields are all on the same line.


```
111,332.1,22537.7,,Container ship,,,111,A-1579-227,10,3,400,Microtome,,111,B-15
222,978.4,7563.1,,Air freight,,,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,,,,
```

The mapping has been designed as follows:

- The key fields are mapped to the respective OrderNo items in the schema target.
- The Detail item in the schema target file has been duplicated, and is displayed as **Detail (2)**. This allows you to map the second set of detail records to the correct item.
- The result of this mapping is basically the same XML file that was produced in the first example.

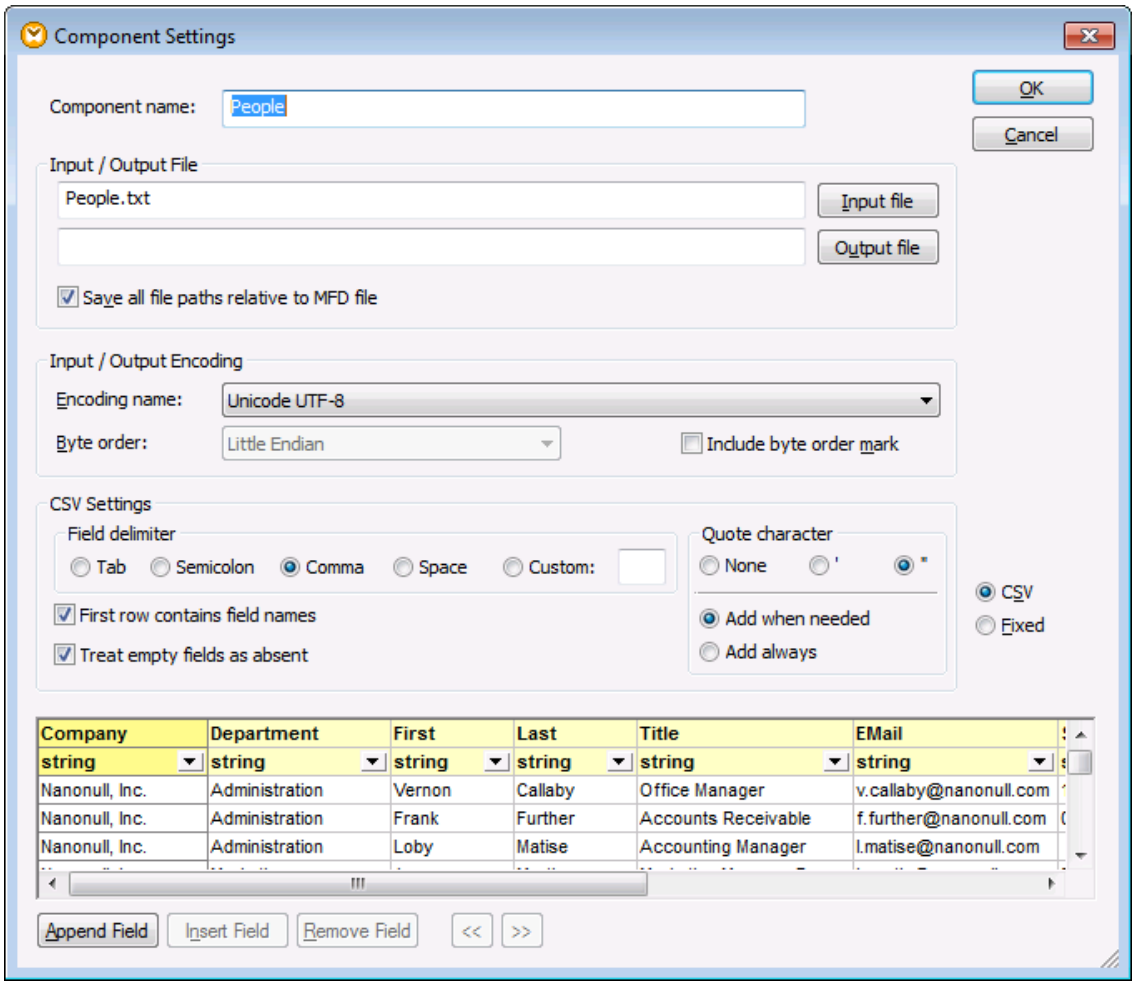


Head-detail-inline.mfd

7.3.4 Setting the CSV Options

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

- Select the component and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

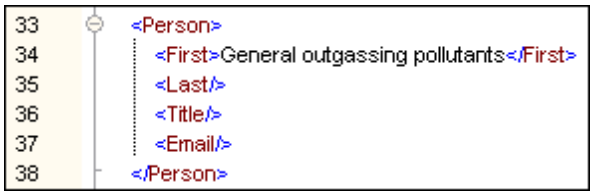
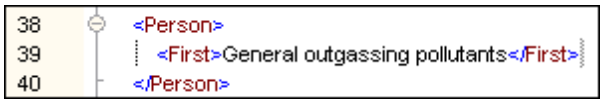
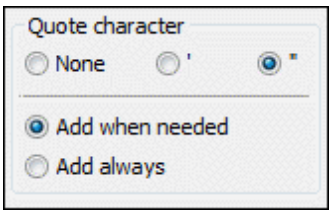


Text Component Settings dialog box (in CSV mode)

The available settings are as follows.

Component name	The component name is automatically generated when you create the component. You can however change the name at any time. If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well. The component name can contain spaces and full stop characters. The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications

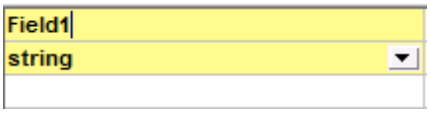
	when changing the name of the component: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have different encodings in Windows and at the command line.
Input file	Specifies the file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The field can remain empty if you are using the text file component as a target for your mapping. In a source component, MapForce uses the value of this field to read column names and preview the contents of the instance text file. To select a new file, click **Input File**.
Output file	Specifies the file to which MapForce will write data. This field is meaningful for a target component. To select a new file, click **Output File**.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the input and output files used by the text component. See also Using Relative Paths on a Component.
Input / Output Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab.
Field delimiter	CSV files are comma delimited "," by default. This option enables you to select the **Tab**, **Semicolon**, or **Space** characters as delimiters. You can also enter a custom delimiter in the **Custom** field.
First row contains field names	Select this option to instruct MapForce to treat the values in the first record of the text file as column headers. The column headers then appear as item names on the mapping.
Treat empty fields as absent	When this option is enabled, empty fields in the source file

	will not produce a corresponding empty item (element or attribute) in the target file. For example, the CSV record "`General outgassing pollutants,,,,`" consists of four fields, the last three of which are empty. Assuming that the output is an XML file, when this option is disabled, the empty fields will be created in the output with an empty value (in this example, the elements `Last`, `Title`, and `Email`):  When this option is enabled, the empty fields will not be created in the output: 
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.  For output files, you can specify additional settings: **Add when needed** — Adds the selected quote character to fields where the text contains the field line breaks. **Add always** — Adds the selected quote character to the generated CSV file.
CSV / Fixed	Changes the component type to either CSV or FLF (fixed-length field).
Preview area	The lower part of the dialog box displays a preview of up to 20 rows of the file selected as input or output.

If necessary, you can create the structure of the file (or change the structure of the existing one), as follows.

Append field	Creates a new field after the last CSV
Insert field	Creates a new field immediately befor currently selected CSV record.
Remove field	Deletes the currently selected field.
<<	Moves the currently selected field one the left.
>>	Moves the currently selected field one the right.

To change the name of a field, click the header (for example, **Field1**), and type the new value. Note that the field names are not editable when the **First row contains field names** option is enabled.



To change the data type of a field, select the required value from the drop-down list. MapForce checks the data type, so if the input data and the field format to do not agree, then the data is highlighted in red.

The field types are based on the default XML schema data types. For example, the Date type is in the form **YYYY-MM-DD**.

7.3.5 Example: Mapping Fixed-Length Text Files to Databases

This example illustrates a data mapping operation between a fixed-length text file (FLF) text file and a Microsoft Access database. The files used in the example are available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. Both the source text file and the target database store a list of employees. In the source file, the records are implicitly delimited by their size, as follows:

Field position and name	Size (in characters)
Field 1 (First name)	8

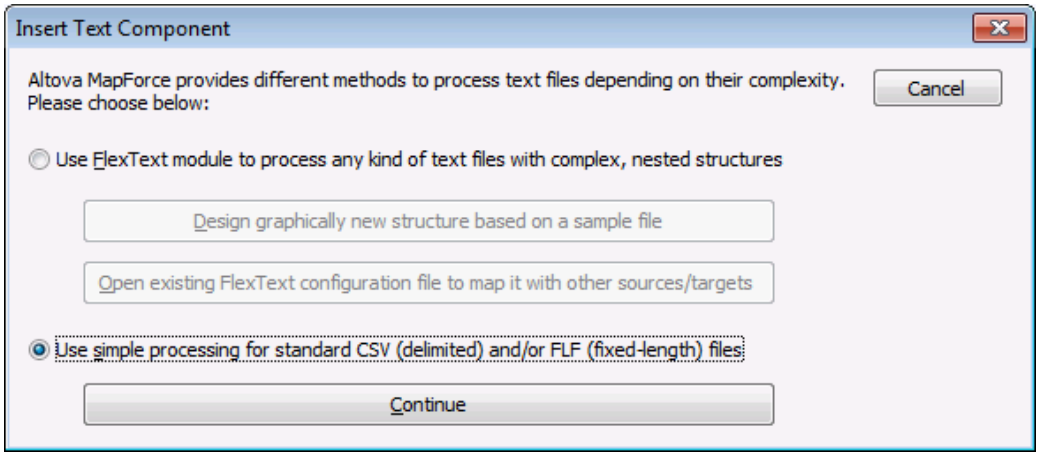
Field position and name	Size (in characters)
Field 2 (Last name)	10
Field 3 (Phone extension)	3
Field 4 (Email)	25
Field 5 (Position)	25

The goals of the mapping is to update the phone extension of each employee in the database to the one existing in the source file, while adding the prefix "100" to each extension. To achieve the goal, take the following steps:

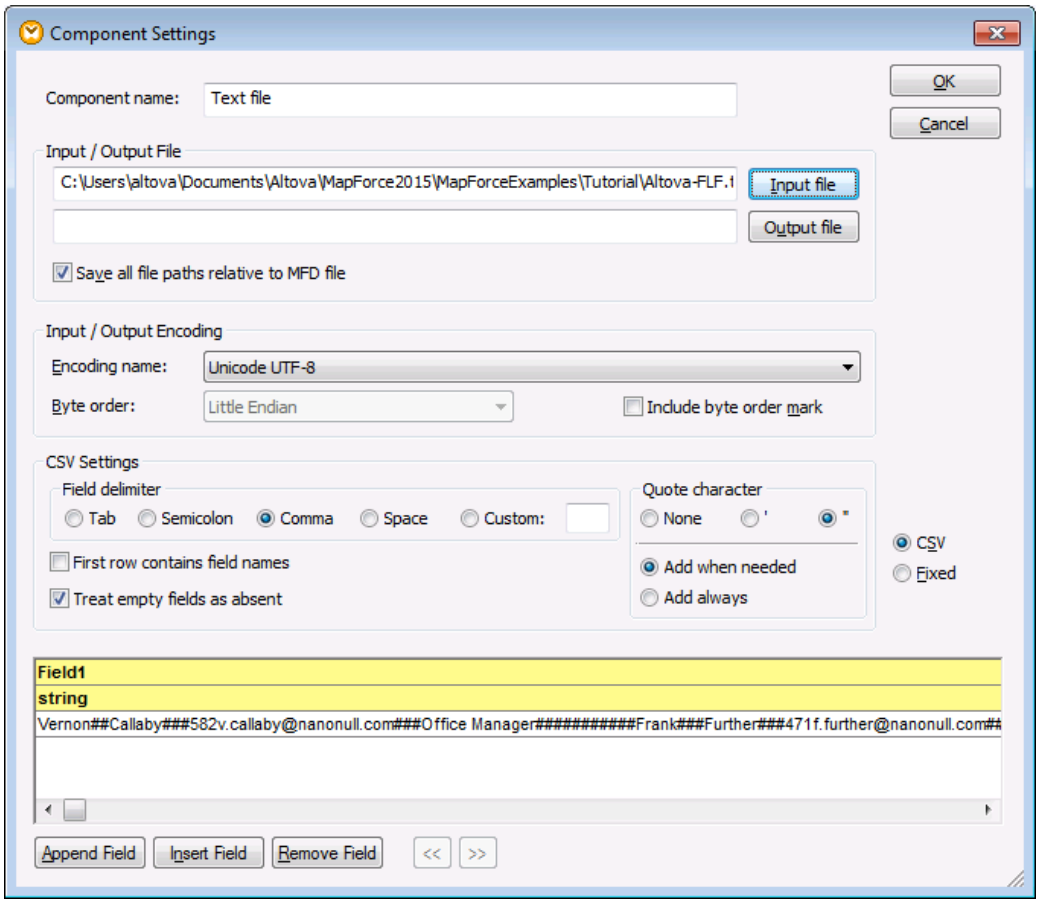
Step 1: Insert and configure the text component
Step 2: Insert the database component
Step 3: Design the mapping
Step 4: Run the mapping

Step 1: Insert and configure the text component

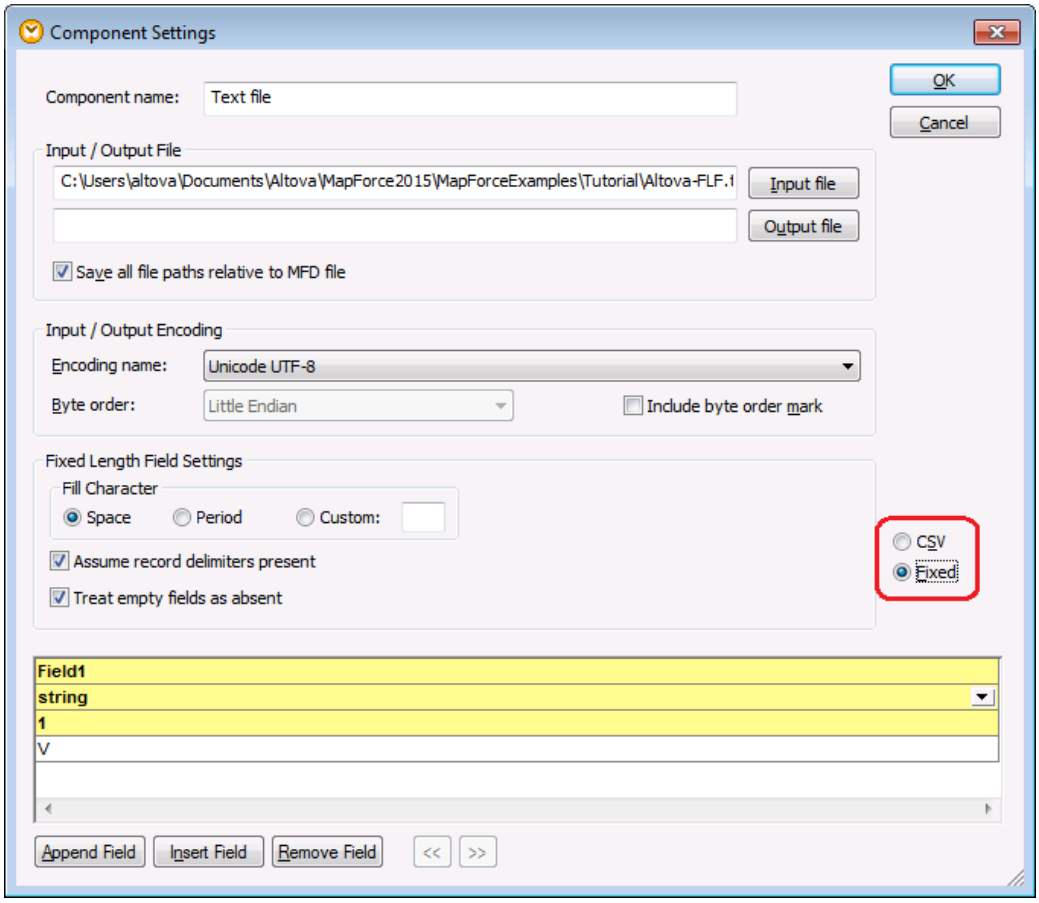
1. Select the menu option **Insert | Text file**, or click the insert Text file icon  . This opens the "Insert Text Component" dialog box.



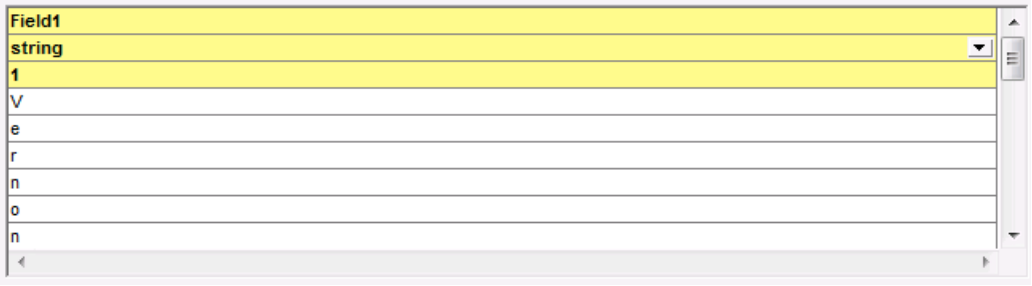
   Select **Use simple processing...** and click **Continue**.
2. Click the **Input file** button and select the file **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\Altova-FLF.txt** file. You will notice that the file is made up of a single string, and contains fill characters of type #.



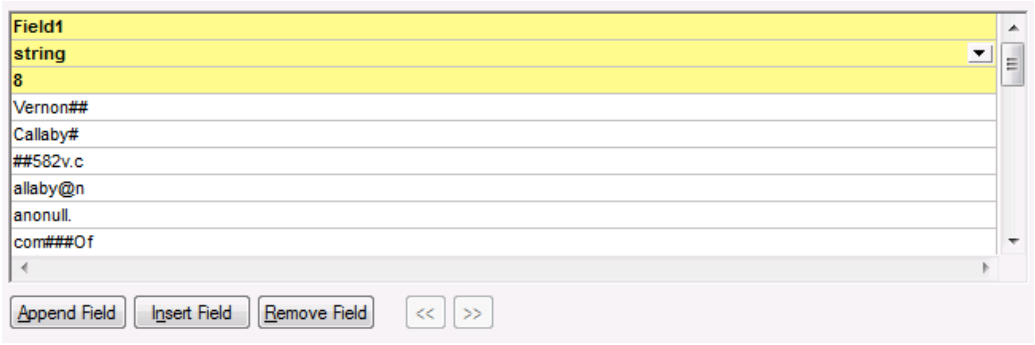
3. Select **Fixed**.



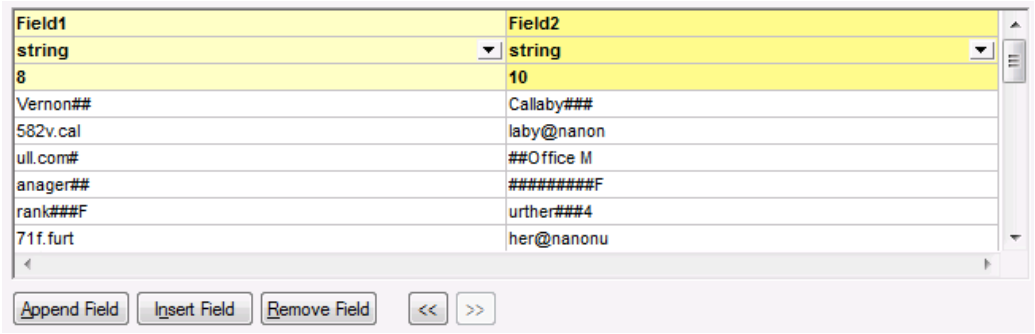
4. Uncheck the **Assume record delimiters present** check box.



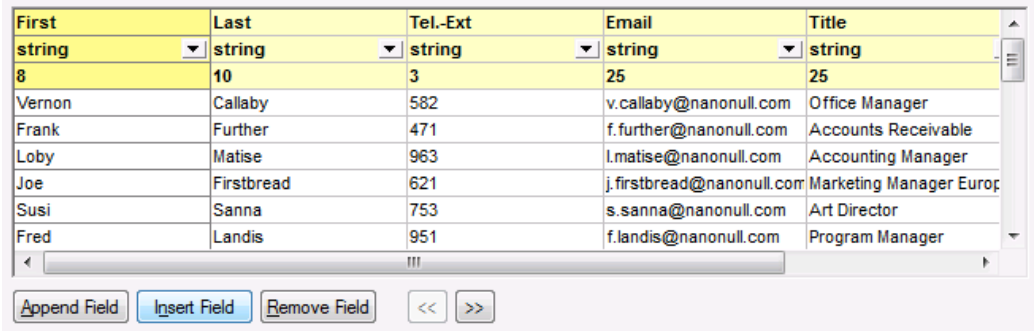
5. The three rows highlighted in yellow are editable, and enable you to specify i) the field name ii) the data type and iii) the field size. Type **8** as the new field size, and press **Enter**. More data is now visible in the first column, which is now defined as 8 characters wide.



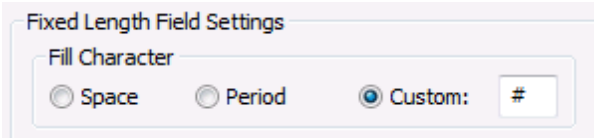
6. Click **Append Field** to add a new field, and set the length of the second field to 10 characters.



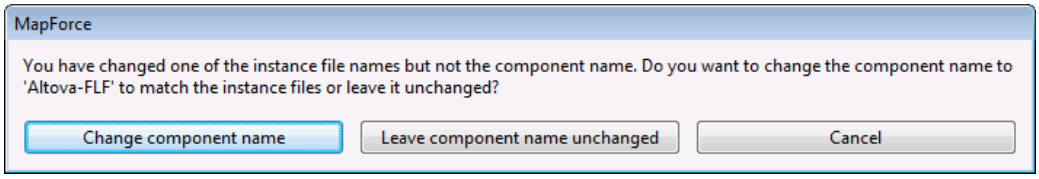
7. Use the same method to create three more fields of the following lengths: 3, 25, and 25 characters, and change the field headers to make them easier to map: First, Last, Tel.-Ext, Email, Title. The preview will then look like this:



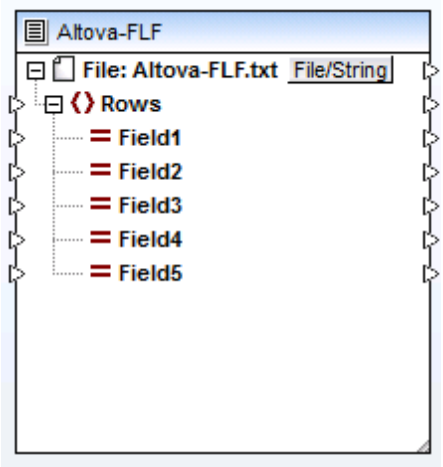
8. In the Fixed Length Field Settings group, select Custom, and type the hash (#) character. This instructs MapForce to treat the # character as fill character.



9. Click **OK**.

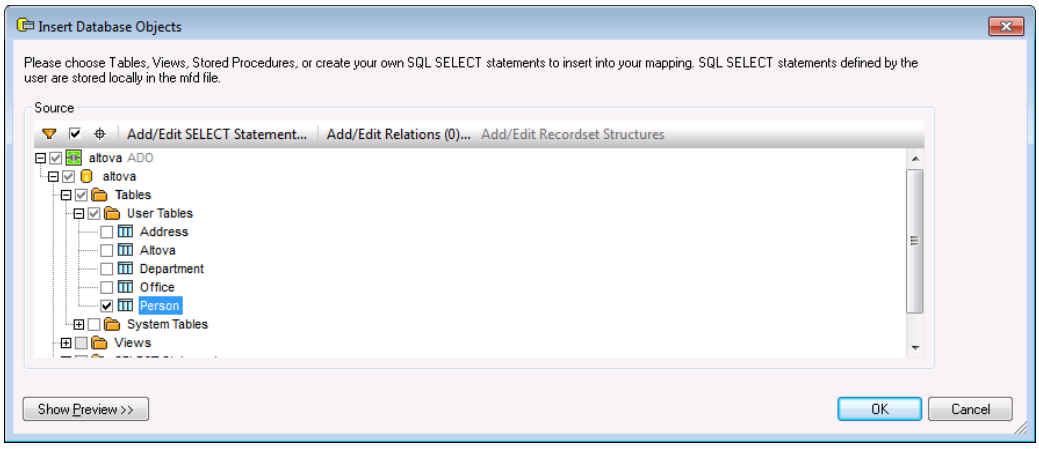


10. Click **Change component name**. The Text file component appears in the Mapping window. Data can now be mapped to and from this component.



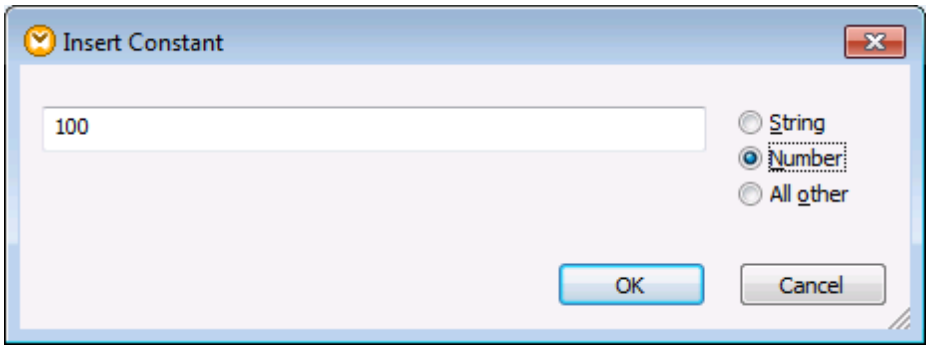
Step 2: Insert the database component

1. Select the menu command **Insert | Database**, select **Microsoft Access**, and then click **Next**.
2. Select the **altova.mdb** database available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, and click **Connect**.
3. Select the **Person** table and click **OK**.

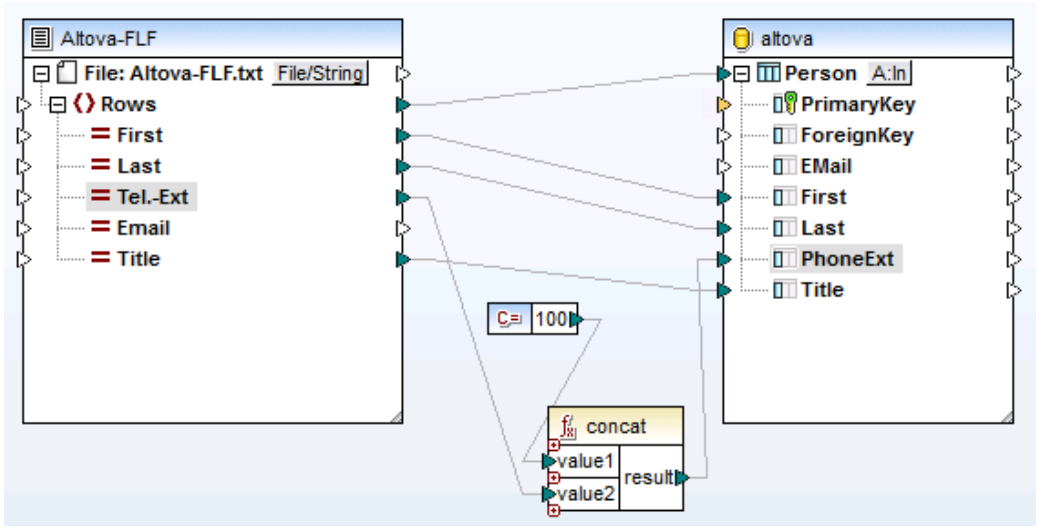


Step 3: Design the mapping

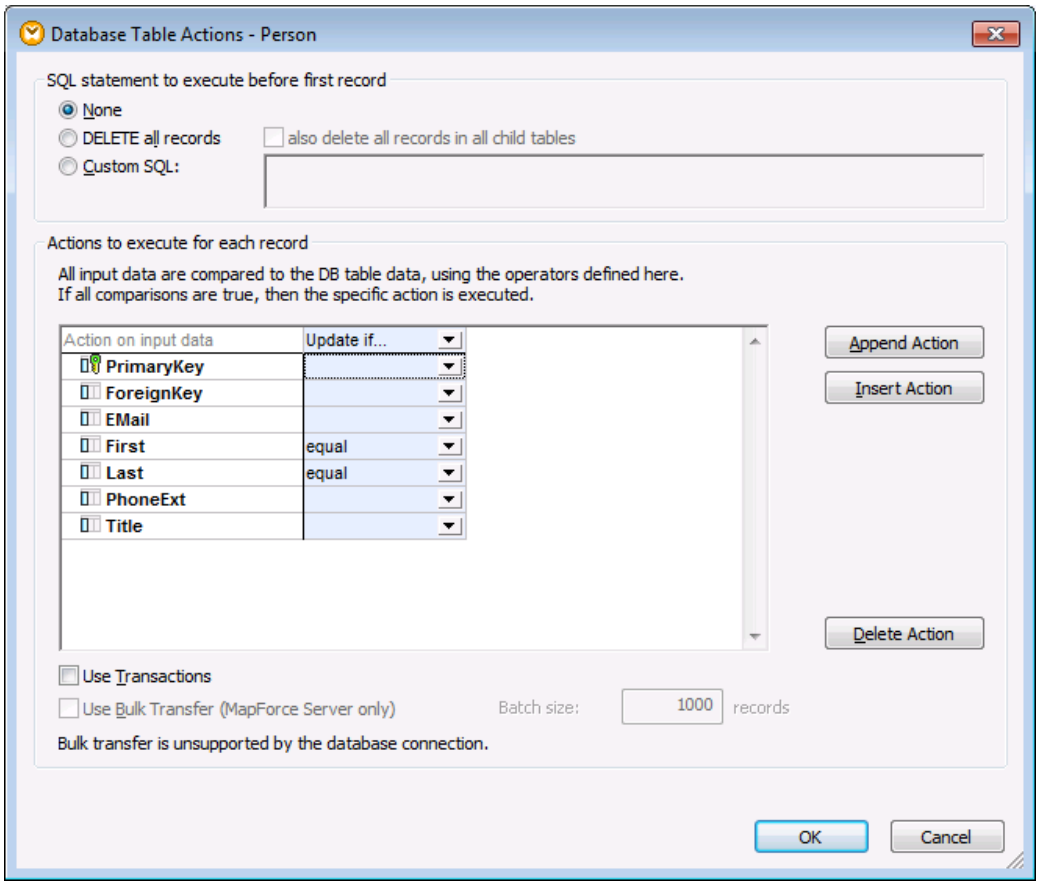
1. Drag the `core | concat` function from the Libraries window into the mapping.
2. Select the menu command **Insert | Constant**, select Number as type, and enter 100 as value. This constant stores the new telephone extension prefix.



3. Create the mapping as shown below.



4. On the database component, click the **Table Action** button A:In next to **Person**.
5. Next to **Action on input data**, select **Update If**, and ensure that the action for **First** and **Last** fields is set to **equal**. This instructs MapForce to update the Person table only if the first and last name in the source file is equal to the corresponding database field. When this condition is true, the action taken is defined by the mapping. In this case, the telephone extension is prefixed by 100, and copied to the **PhoneExt** field of the Person table.



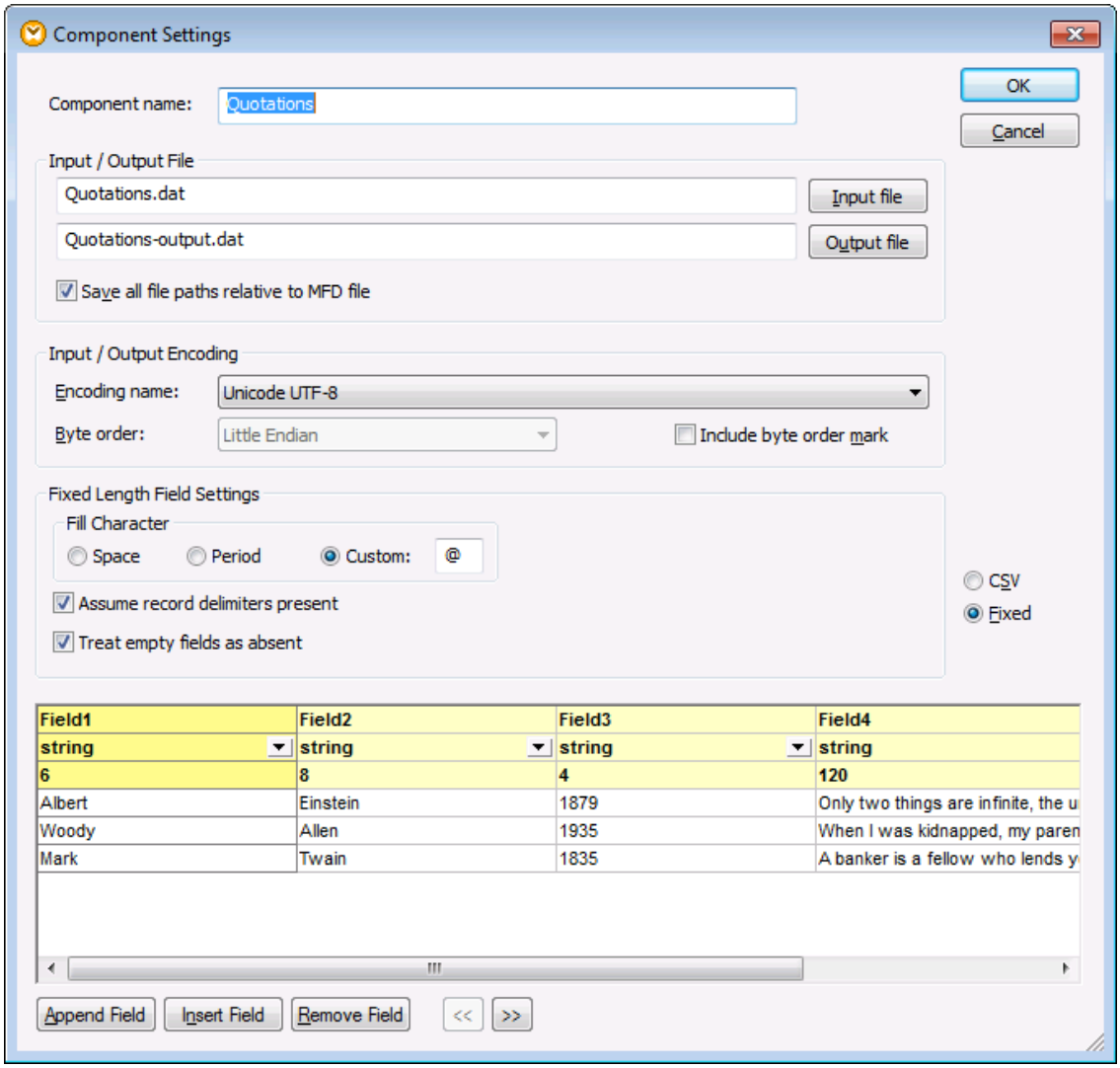
Step 4: Run the mapping

To generate the SQL statement (for preview in MapForce), click the Output tab. To run the SQL statements against the database, click the Run SQL-script button  .

7.3.6 Setting the FLF Options

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

- Select the component and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

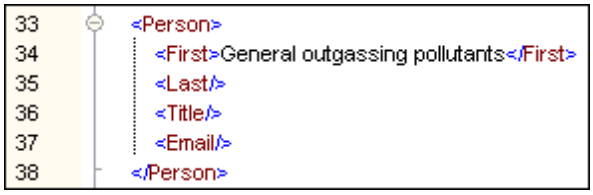
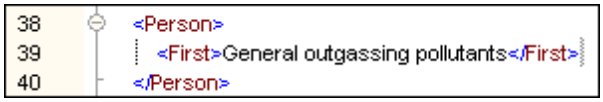


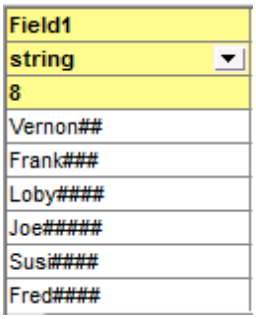
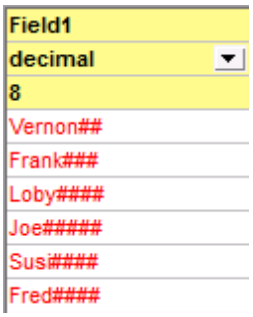
Text Component Settings dialog box (in fixed-length field mode)

The available settings are as follows.

Component name	The component name is automatically generated when you create the component. You can however change the name at any time. If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well. The component name can contain spaces and full stop characters. The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications

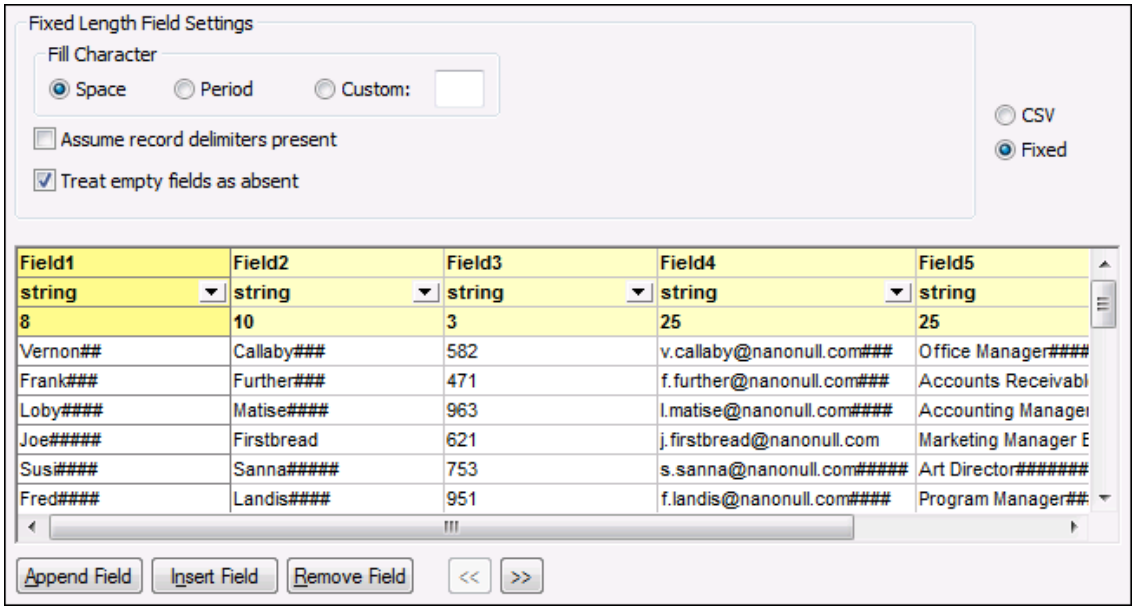
	when changing the name of the component: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have different encodings in Windows and at the command line.
Input file	Specifies the file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The field can remain empty if you are using the text file component as a target for your mapping. In a source component, MapForce uses the value of this field to read column names and preview the contents of the instance text file. To select a new file, click **Input File**.
Output file	Specifies the file to which MapForce will write data. This field is meaningful for a target component. To select a new file, click **Output File**.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the input and output files used by the text component. See also Using Relative Paths on a Component.
Input / Output Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab.
Fill Character	This option allows you to define the characters that are to be used to complete, or fill in, the rest of the (fixed) field when the incoming data is shorter than the respective field definitions. The custom field allows you to define your own fill character in the Custom field. If the incoming data already contains specific fill characters, and you enter the same fill character in the Custom field, then the incoming data will be stripped of those fill

	characters!
Assume record delimiters present	This option is useful when you want to read data from a source flat file that does not contain record delimiters such as CR/LF, or when you want to produce a target flat FLF file without record delimiters. See the Understanding the "Assume record delimiters present" option section below.
Treat empty fields as absent	When this option is enabled, empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file. Assuming that the output is an XML file, when this option is disabled, the empty fields will be created in the output with an empty value (in this example, the elements `Last`, `Title`, and `Email`):  When this option is enabled, the empty fields will not be created in the output: 
CSV / Fixed	Changes the component type to either CSV or FLF (fixed-length field).
Preview area	The lower part of the dialog box displays a preview of up to 20 rows of the file selected as input or output. If necessary, you can create the structure of the file (or change the structure of the existing one), as follows. **Append field** — Creates a new field after the last reco **Insert field** — Creates a new field immediately befor currently selected record. **Remove field** — Deletes the currently selected field. **<<** — Moves the currently selected field one the left. **>>** — Moves the currently selected field one the right.

	To change the name of a field, click the header (in this example, **Field1**), and type the new value.  To change the data type of a field, select the required value from the drop-down list. MapForce checks the data type, so if the input data and the field format to do not agree, then the data is highlighted in red.  To set the size of the field in characters, enter the field size in the third row from the top.

Understanding the "Assume record delimiters present" option

To better understand this option, open the **Altova-FLF.txt** file available in the **<Documents>\Altova\MapForce2016\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.



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.

```
 1  Vernon##Callaby###582v.callaby@nanonull.com###Office Manager##########
 2  Frank###Further###471f.further@nanonull.com###Accounts Receivable######
 3  Loby####Matise####963l.matise@nanonull.com####Accounting Manager#######
 4  Joe#####Firstbread621j.firstbread@nanonull.comMarketing Manager Europe#
 5  Susi####Sanna#####753s.sanna@nanonull.com#####Art Director#############
 6  Fred####Landis####951f.landis@nanonull.com####Program Manager##########
 7  MichelleButler####654m.landis@nanonull.com####Software Engineer########
 8  Ted#####Little####852t.little@nanonull.com####Software Engineer########
 9  Ann#####Way#######951a.way@nanonull.com#######Technical Writer#########
10  Liz#####Gardner###753l.gardner@nanonull.com###Software Engineer########
11  Paul####Smith#####334p.smith@nanonull.com#####Software Engineer########
12  Alex####Martin####778a.martin@nanonull.com####IT Manager###############
13  George##Hammer####223g.hammer@nanonull.com####Web Developer############
14  Jessica#Bander####241j.band@nanonull.com######Support Engineer#########
15  Lui#####King######345l.king@nanonull.com######Support Engineer#########
16  Steve###Meier#####114s.meier@nanonull.com#####Office Manager##########
17  Theo####Bone######331t.bone@nanonull.com######Accounts Receivable######
18  Max#####Nafta#####122m.nafta@nanonull.com#####PR & Marketing Manager US
19  ValentinBass######716v.bass@nanonull.com######IT Manager###############
20  Carl####Franken###147c.franken@nanonull.com###Support Engineer#########
21  Mark####Redgreen##152m.redgreen@nanonull.com##Support Engineer#########
22
```

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.

```
1   Vernon##Callaby###582v.callaby@nanonull.com###Office
    Manager###########Frank###Further###471f.further@nanonull.com###Accounts
    Receivable######Loby####Matise####963l.matise@nanonull.com####Accounting
    Manager#######Joe#####Firstbread621j.firstbread@nanonull.comMarketing Manager
    Europe#Susi####Sanna#####753s.sanna@nanonull.com#####Art
    Director#############Fred####Landis####951f.landis@nanonull.com####Program
    Manager##########MichelleButler####654m.landis@nanonull.com####Software
    Engineer########Ted#####Little####852t.little@nanonull.com####Software
    Engineer########Ann#####Way#######951a.way@nanonull.com#######Technical
    Writer#########Liz#####Gardner###753l.gardner@nanonull.com###Software
    Engineer########Paul####Smith#####334p.smith@nanonull.com#####Software
    Engineer########Alex####Martin####778a.martin@nanonull.com####IT
    Manager###############George##Hammer####223g.hammer@nanonull.com####Web
    Developer############Jessica#Bander####241j.band@nanonull.com######Support
    Engineer#########Lui#####King######345l.king@nanonull.com######Support
    Engineer#########Steve###Meier#####114s.meier@nanonull.com#####Office
    Manager###########Theo####Bone######331t.bone@nanonull.com######Accounts
    Receivable######Max#####Nafta#####122m.nafta@nanonull.com#####PR & Marketing Manager
    USValentinBass######716v.bass@nanonull.com######IT
    Manager###############Carl####Franken###147c.franken@nanonull.com###Support
    Engineer#########Mark####Redgreen##152m.redgreen@nanonull.com##Support
    Engineer#########
```

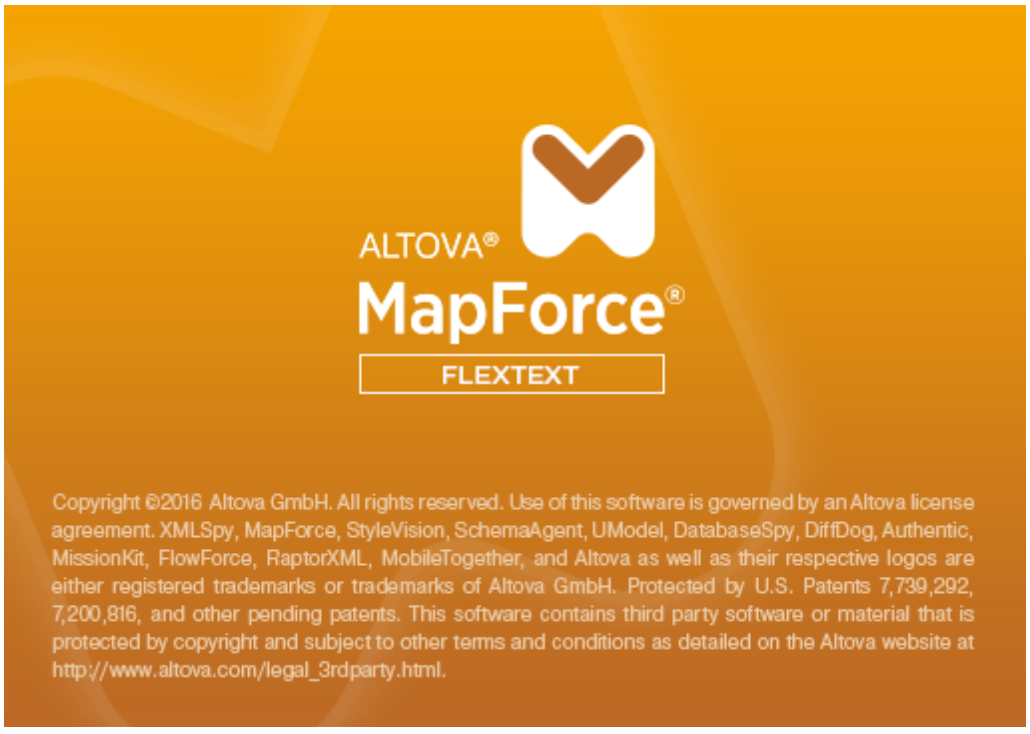
The mapping result when "Assume record delimiters present "is disabled

7.4 MapForce FlexText

Altova web site: Mapping complex text files - FlexText

FlexText is a MapForce module which enables you to convert data from non-standard or legacy text files of high complexity to other formats supported by MapForce. While XML files have a schema or a structure from which MapForce can derive the schema, this is not the case of text files, especially when they have a complex and unique structure that does not consistently fit into CSV or fixed-length field patterns. Moreover, sometimes you need to extract only portions of useful data from a legacy text file. FlexText solves these problems by helping you define and test visually, in real-time, the rules, or the template, according to which text data must be split down into mappable items. A FlexText template essentially defines the structural model of your custom text data, according to criteria you specify, for the purpose of mapping it to other formats.

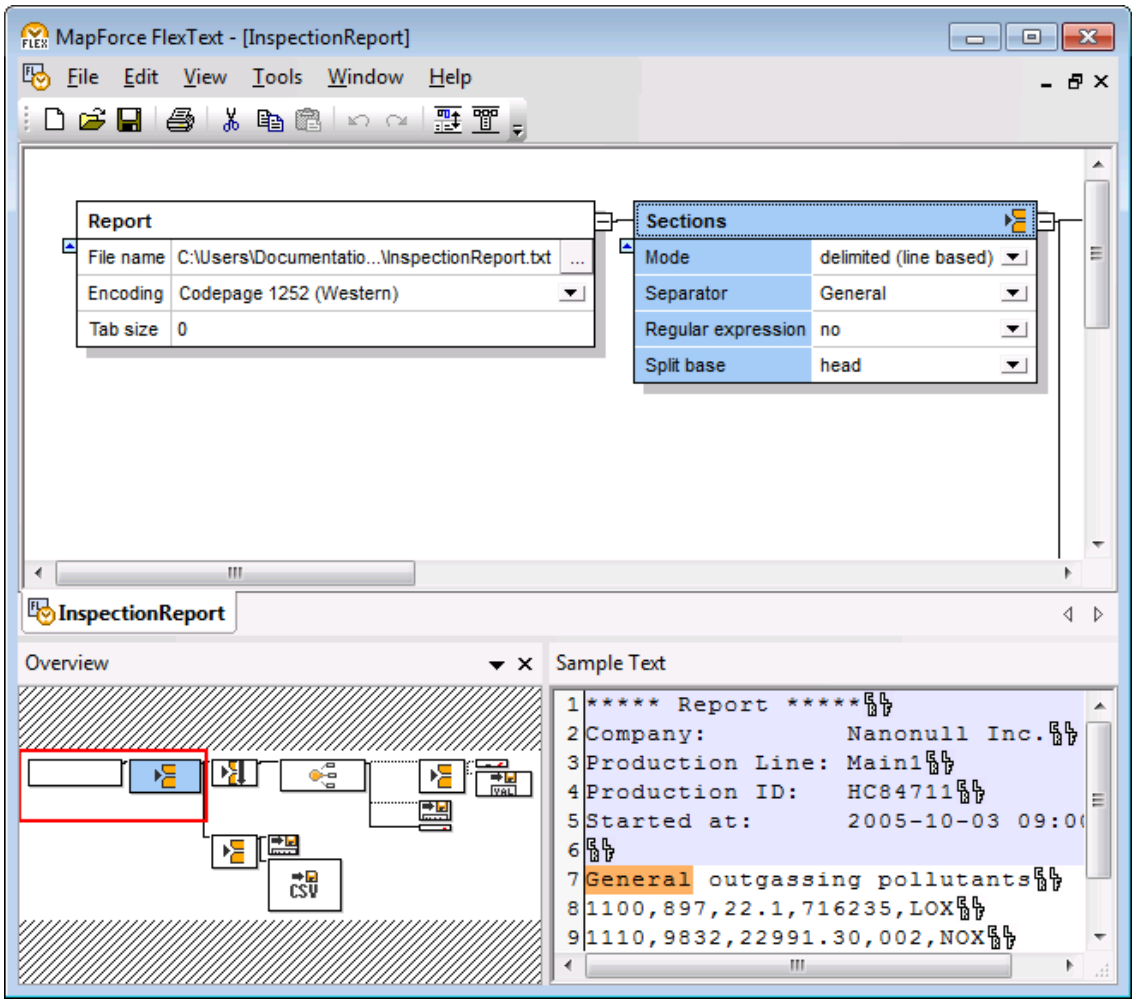
Once you define a FlexText template, you can add it to the MapForce mapping area as a source or target component, and thus map complex and non-standard text data with any other format supported by MapForce. FlexText template files have the .mft (MapForce FlexText) extension. You can reuse the same FlexText template in multiple mappings, to process any number of text files.



7.4.1 Overview

You can start FlexText directly from MapForce, when adding text files as mapping components (use the **Insert | Text File** menu command). You can also start FlexText as a standalone program, by running the *Altova MapForce FlexText* executable available in the MapForce installation directory.

The screen shot below illustrates a sample FlexText template (**InspectionReport.mft**) which is available, along with several other FlexText samples, in the **<Documents>\Altova \MapForce2016\MapForceExamples** directory.



The FlexText graphical Interface

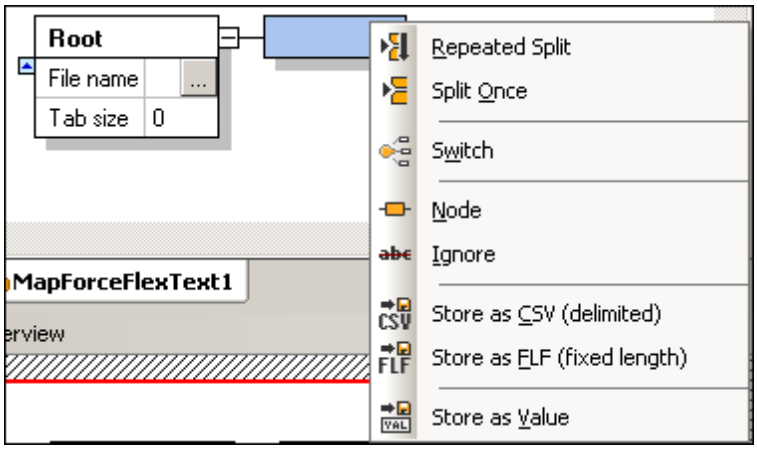
The FlexText interface consists of the following panes: Design pane, Overview pane and Sample Text pane. As you will see next, the position and appearance of these panes can be customized according to your preferences.

Design pane

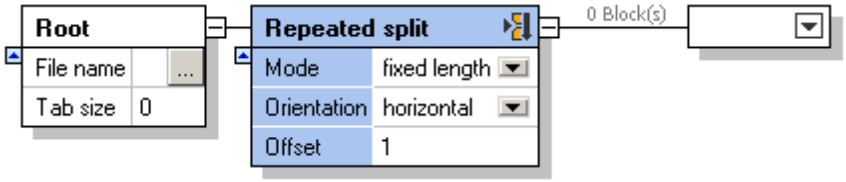
The Design pane is the working area where you define the structure of your text, by means of containers. Containers define the rules for handling text data according to criteria you specify. The first container of any FlexText template is the *Root* container, which represents the entire text file. All other containers essentially define the subsequent text processing logic (typically, splitting text into meaningful units from which you can map data). For example, the *Split Once* container splits a fragment of text into exactly two fragments. By default, the name of any container describes its function (for example, *Repeated Split*); however, you can change it if necessary. You typically define as many containers as demanded by the structure of the text file with which you are working.

The FlexText Reference section of this documentation covers containers in more detail. However, to begin with, note the following about containers:

- Clicking the icon in the top-right corner of a container opens a pop-up menu from which you can select the container type.



- Each container has a number of options which dictate what happens to text data at that particular place in the structure. These options determine the content of the container, and enable you to refine it before providing it to the MapForce component.



- To show or hide container contents, click the **Expand** or **Collapse** icon, respectively.
- To collapse containers as a group, press the **Shift** key. Two chevrons appear. Clicking the handle collapses the section of the container tree to the right of the one clicked.
- To preview the text created by a particular container directly in the design pane, enable the **Node Text in Design view** toolbar button.
- When the **Auto-collapse unselected node text** toolbar button is enabled, the full contents of a container is expanded when you select it. All other containers are collapsed.

Overview pane

The Overview pane gives a bird's-eye view of the Design pane. To navigate the Design pane, click and drag the red rectangle. To detach the Overview pane and reposition it elsewhere in the interface, click its title bar and drag the pane to the desired location.

Sample Text pane

The Sample Text pane displays the contents of the currently selected container. (Note that this pane is not shown if the **Node Text in Design view** toolbar button is enabled. Instead, the contents of the currently selected container is shown in the Design pane.)

To help you quickly identify the **Tab** and **Carriage Return/Line Feed** characters in the preview text, FlexText displays such characters using the → and ¶ symbols, respectively.

7.4.2 FlexText Tutorial

The tutorial will show you how to use the most common, and most powerful, features of FlexText to process a text file and map its output in various ways in MapForce.

The example uses the **Flex-tutorial.txt** file available in the **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial** folder. The .txt file has the following format:

```
111,332.1,22537.7,5,Container ship,Mega,
111,A1579227,10,3,400,Microtome,
111,B152427,7,6,1200,Miscellaneous,
222,978.4,7563.1,69,Air freight,Mini,
222,ZZAW561,10,5,10000,Gas Chromatograph,,

General outgassing pollutants
1100,897,22.1,716235,LOX
1110,9832,22991.30,002,NOX
1120,1213,33.01,008,SOX
```

Aim of the tutorial:

- To separate out the records containing 111, and 222 keys, into separately mappable items.
- To discard the plain text record.
- To create a CSV file of the remaining records.

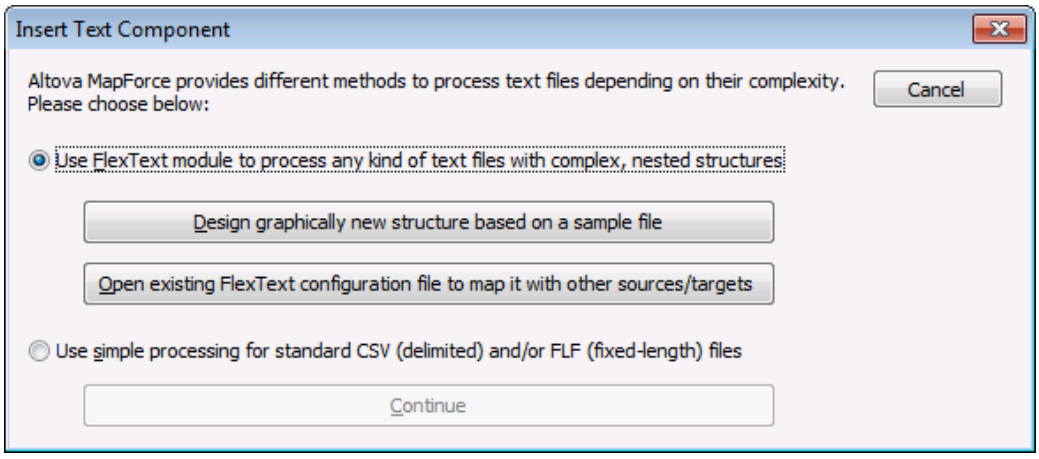
The tutorial is organized into the following parts:

- Step 1: Create the FlexText Template
- Step 2: Define Split Conditions
- Step 3: Define Multiple Conditions per Container
- Step 4: Create the Target MapForce Component
- Step 5: Use the FlexText Template in MapForce

7.4.2.1 Step 1: Create the FlexText Template

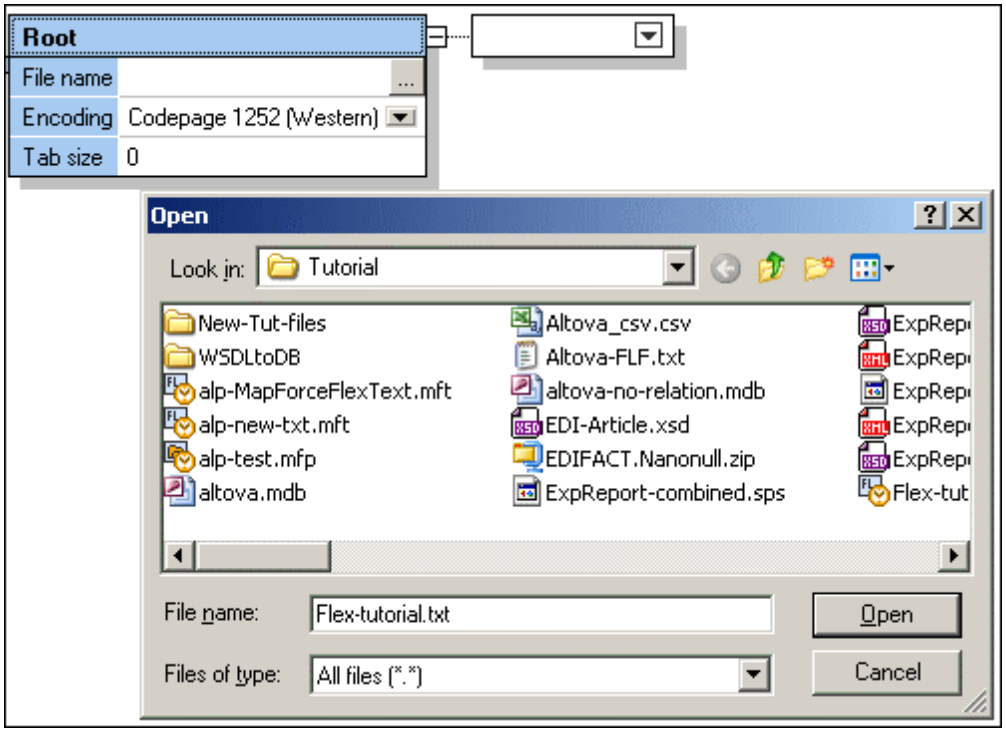
To create the FlexText template:

1. Start MapForce, and open a new mapping file.
2. Select **Insert | Text file**, or click the Insert Text file icon  .
3. Click the "**Design graphically new structure ...**" button.

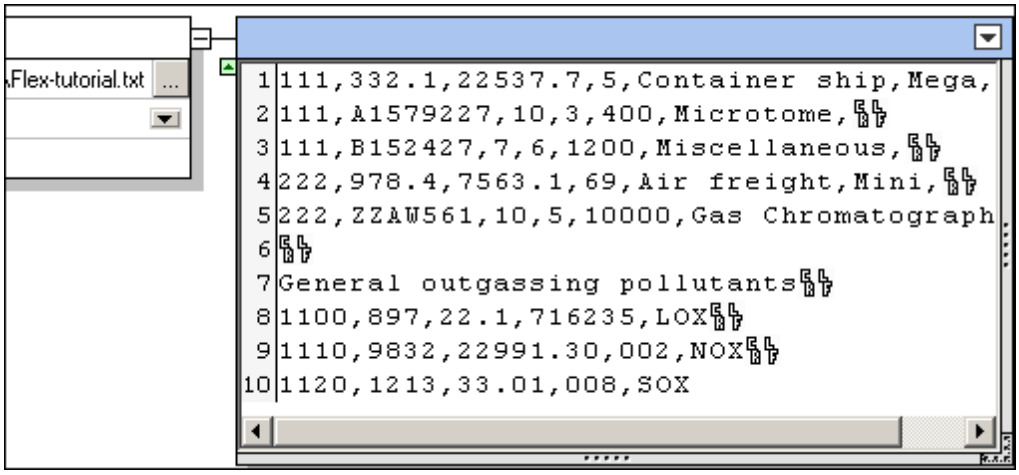


4. Enter a name for your FlexText template, and click Save to continue (e.g. Flex-tutorial.mft).

An empty design, along with the "Open" dialog box are displayed.



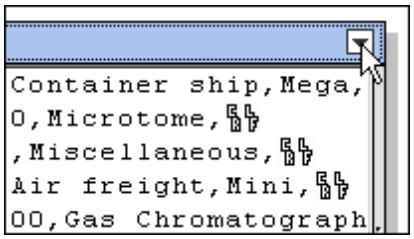
5. Select the **Flex-tutorial.txt** file in the **...\MapForceExamples\Tutorial** folder, and confirm by clicking Open.



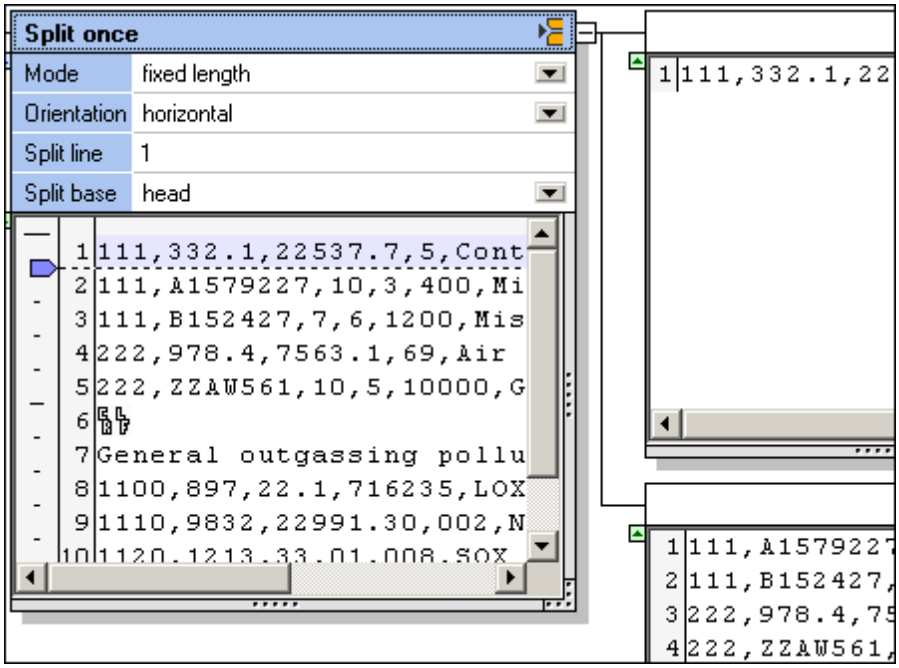
The text file contents are now visible.

Clicking the "Node Text in Design view" icon  , displays the active container contents, in the Sample Text pane.

Activating "Auto-collapse unselected node text"  , displays the content in the active container, all other containers which contain content, are collapsed.



6. Click the container icon at the top right, and select **Split once** from the pop-up menu. Two new containers appear next to the Split once container. For more information on the Split once condition, please see: Split once.



The default settings of the Split once container are visible: fixed length, horizontal and split line=1.

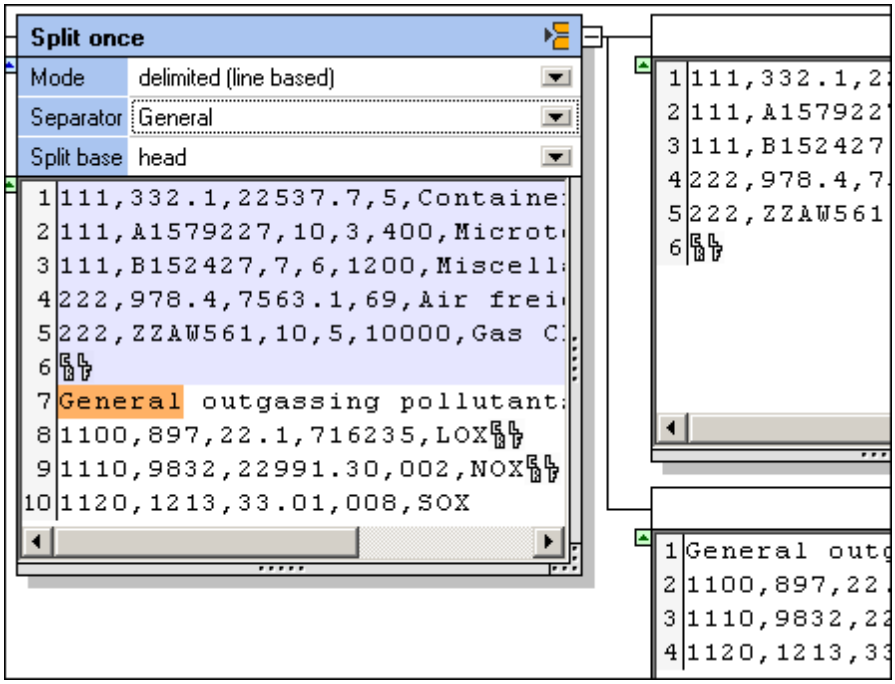
The result of these default settings are also visible:

- The top container contains the first line of the text file, highlighted in the **Split once** container.
- The lower container contains the rest of the text file.

7.4.2.2 Step 2: Define Split Conditions

Split conditions allow you to segment text fragments in various ways. To define split conditions:

1. Click the **Mode** combo box and select "delimited (line based)".
2. Double click the Separator field and enter "General".

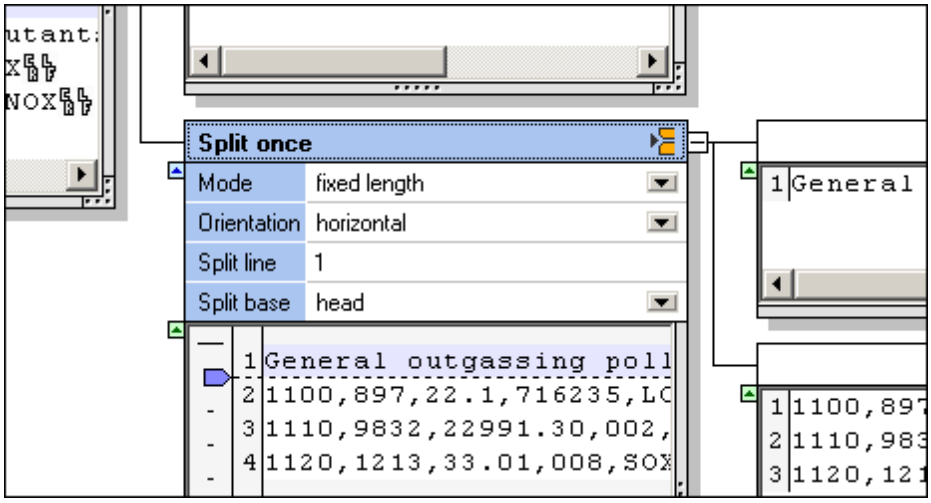


The text fragments in the respective containers have now changed.

Entering "General" and using delimited (line based), allows you to split off that section of text that contains the string "General", into the lower container. The text fragment up to the separator, is placed in the top container.

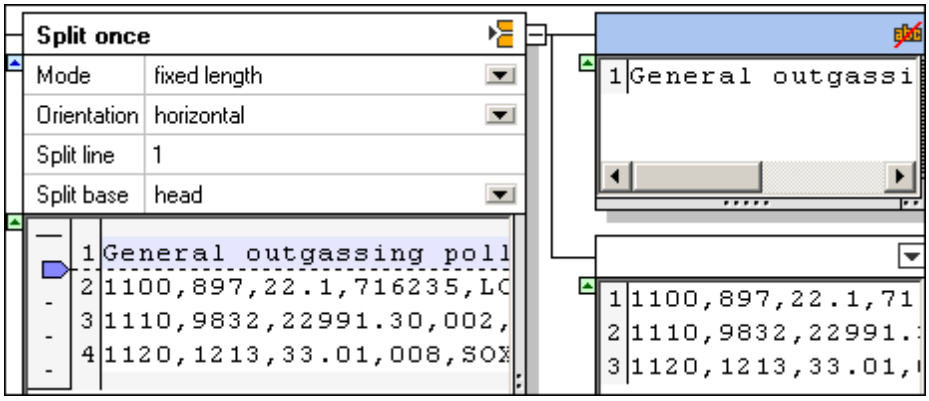
What we want to do now, is work on the lower container to produce a CSV file containing the records with 1100 and up.

3. Click the lower container and change it to **Split once**.



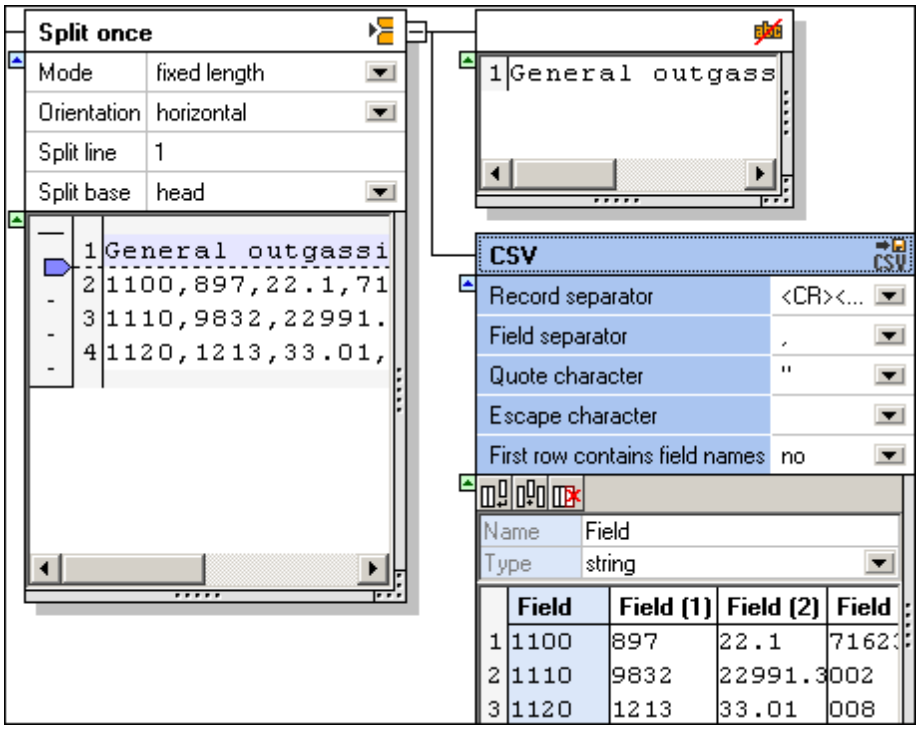
Two new containers are created. The default settings can remain as they are, because we now want to split off the first line of this text fragment, and ignore it. The remaining fragment in the lower container will be made into a CSV file.

4. Click the top container and change it to **Ignore**.



The text fragment, and thus mapping item, of this container has now been made unavailable for mapping in MapForce.

5. Click the lower container icon and change it to **Store as CSV**.



The container now shows the text fragment in a tabular form. The default settings can be retained.

Configuring the CSV file:
If you want to change the field names, click the field, in the table, and then change the entry in the **Name** field. Columns can also be appended, inserted and deleted in this container, please see "Store as CSV" for more information.

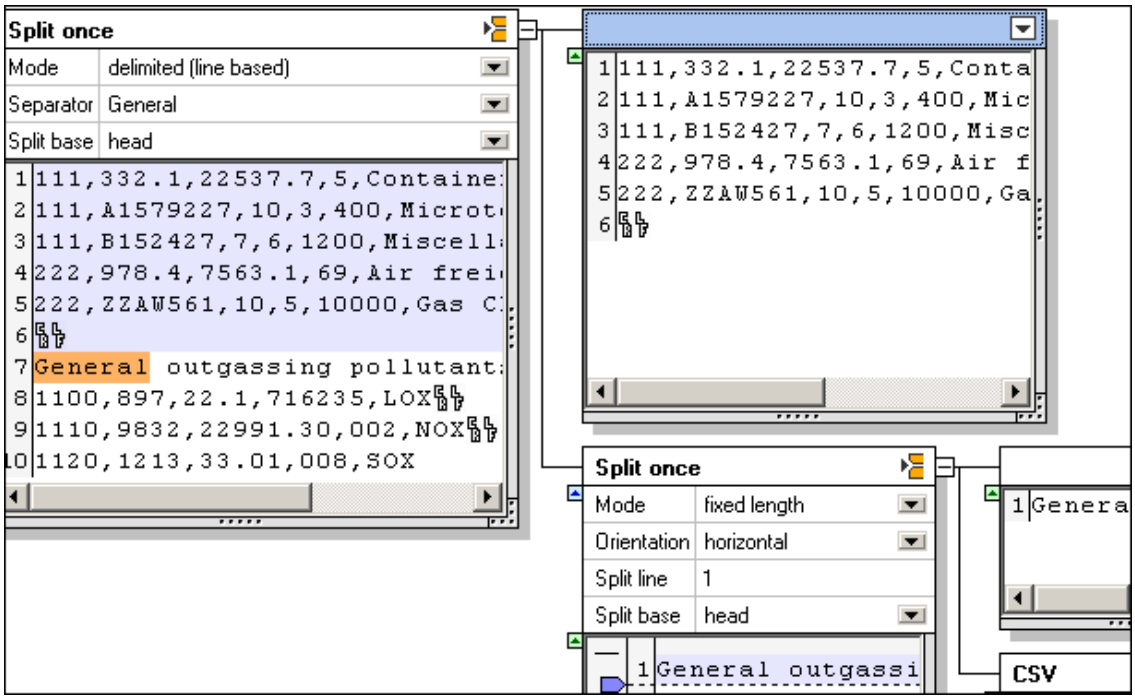
We can now continue with defining the remaining text fragment.

7.4.2.3 Step 3: Define Multiple Conditions per Container

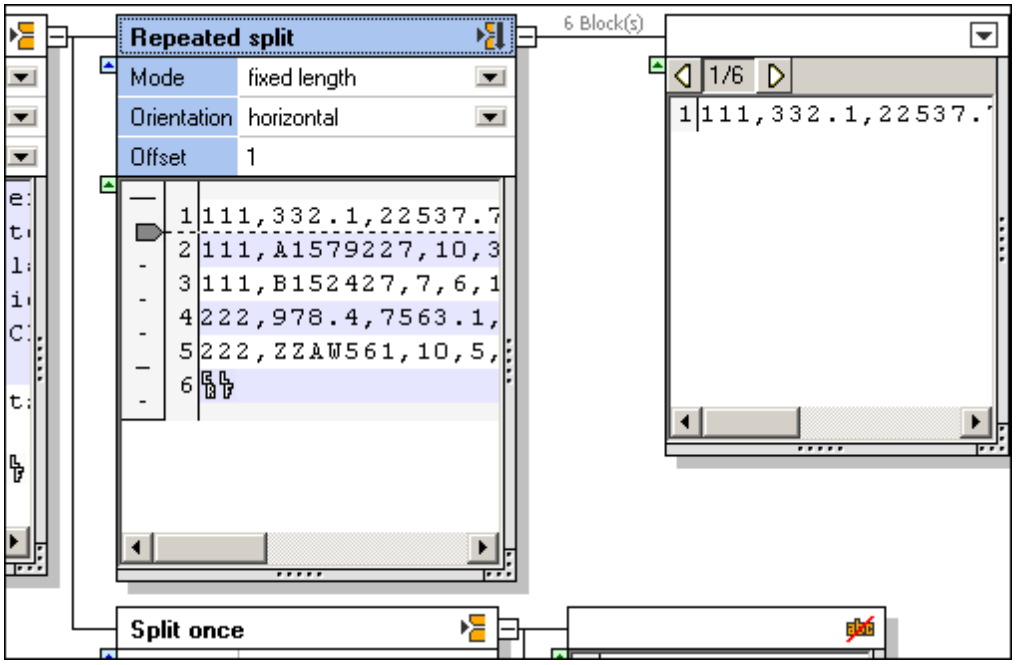
FlexText allows you to define multiple conditions per text fragment, using the Switch container. An associated container is automatically allocated to each condition that you define.

The current state of the tutorial at this point is that lower text fragment, of the first **Split once** container, has been defined:

- A Split once container splits off the first line into an Ignore container.
- The remaining segment is defined/stored as a CSV file.



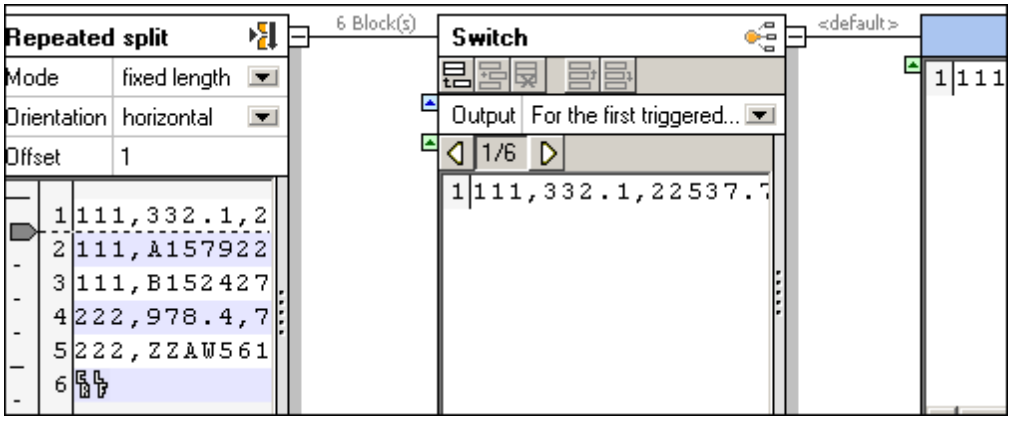
1. Click the top container icon and change it to **Repeated split**.



The default settings are what we need at this point. The text fragment is split into multiple text blocks of a single line each. The associated container shows a preview of each of the text blocks.

Clicking the Next text block icon , allows you to cycle through all the text fragments, of which there are 6.

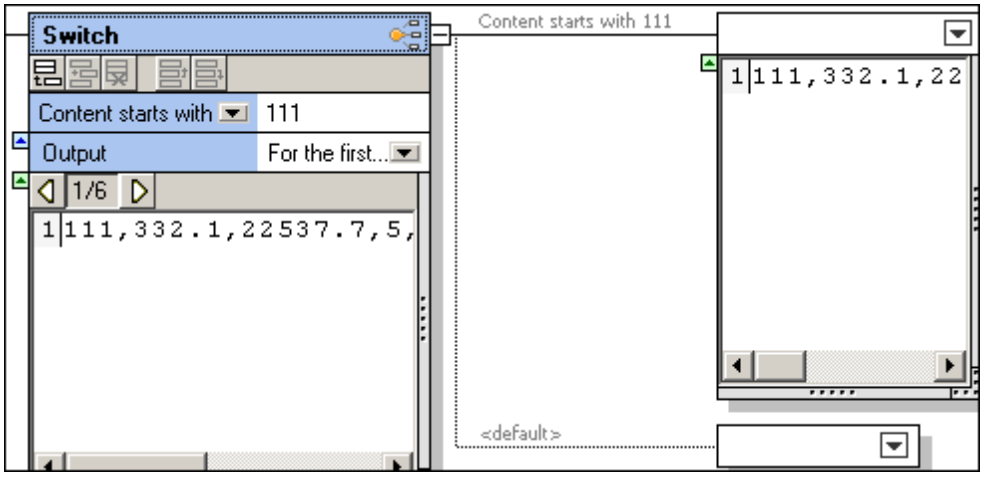
2. Click the new container and change it to **Switch**.



The initial state of the Switch container is shown above.

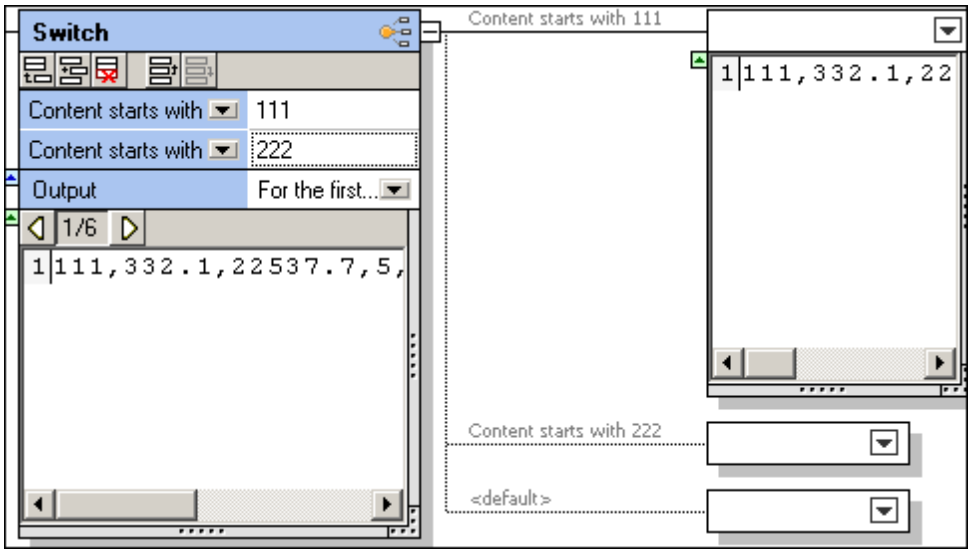
- An associated container "default", has been added.
- The content of the first record 1/6, is displayed in the default container.

3. Click the Append condition icon in the "Switch" title bar, to add a new condition.
4. Double click in the field "Content starts with", and enter 111.



This defines the first condition. An associated container (Content starts with 111) has been added above the "default" container.

5. Click the append icon again, and enter "222" in the Content starts with field.

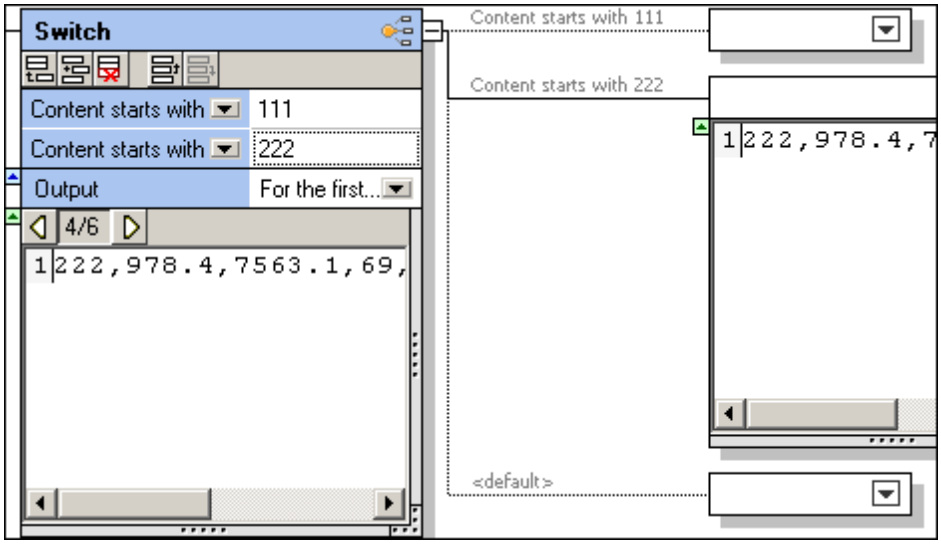


A third container has been added (Content starts with 222).

Please note:

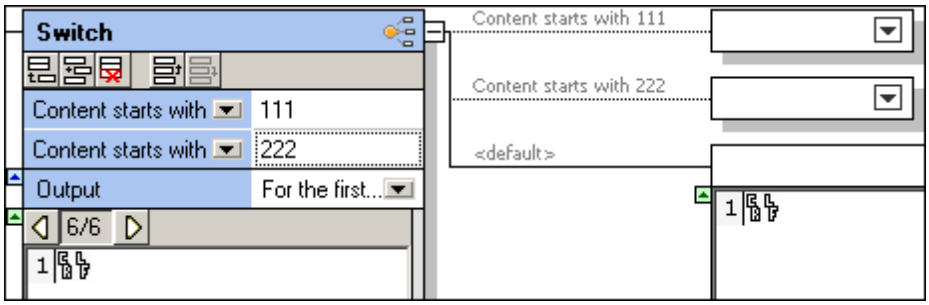
Clicking the "Contents starts with" combo box, allows you to select the "Contains" option. This allows you to specify a "string" which can occur anywhere in the text fragment.

6. Click the Next text block icon , several times to see the effect.



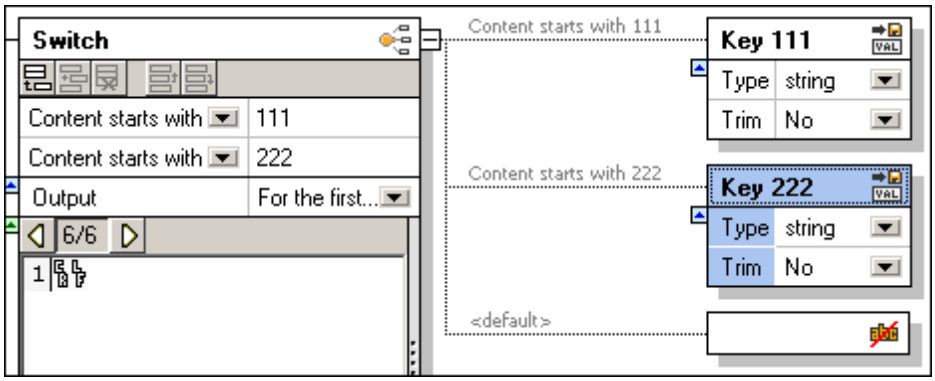
Upon reaching record 4 of 6, container 222 opens up, and displays its content.

7. Continue clicking, till you reach record 6 of 6. A single CR / LF character is displayed in the default container.



If a data fragment in the current block satisfies a condition, then the **complete data** of that block is passed on to the associated container. Data is not split up in any way, it is just routed to the associated container, or to the default container if it does not satisfy any of the defined conditions.

8. Click the first two containers and change them to **Store as value**. Click the last container and change it to **Ignore**.
9. Double click the "Store" text, and add descriptive text e.g Key 111 and Key 222.

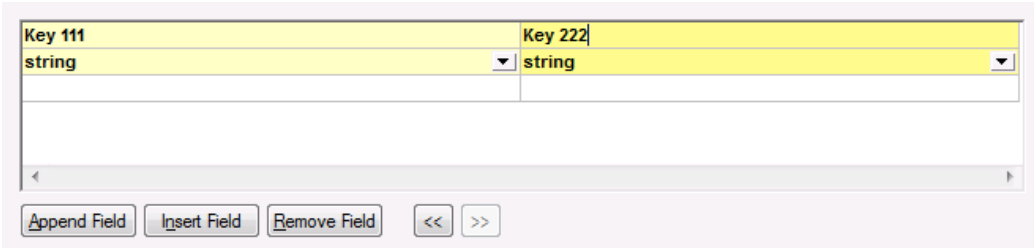


10. Save the FlexText template, e.g. Flex-Tutorial.mft.

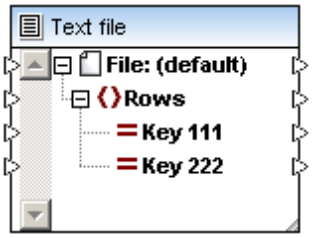
7.4.2.4 Step 4: Create the Target MapForce Component

To create the target component for use with the FlexText source:

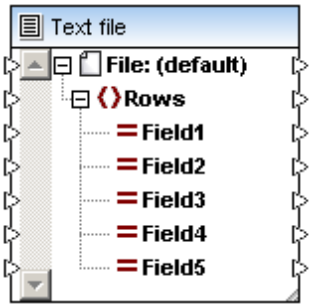
1. Select **Insert | Text file**, or click the Insert Text file icon  .
2. Click the "**Use simple processing...**" radio button and click **Continue**. This opens the Component Settings dialog box.
3. Click the **Append Field** button to add a new field.
4. Double click the Field1 field name and change it to **Key 111**.



5. Do the same for the other field, and name it **Key 222** and click OK to confirm. A text component with two fields has now been created.



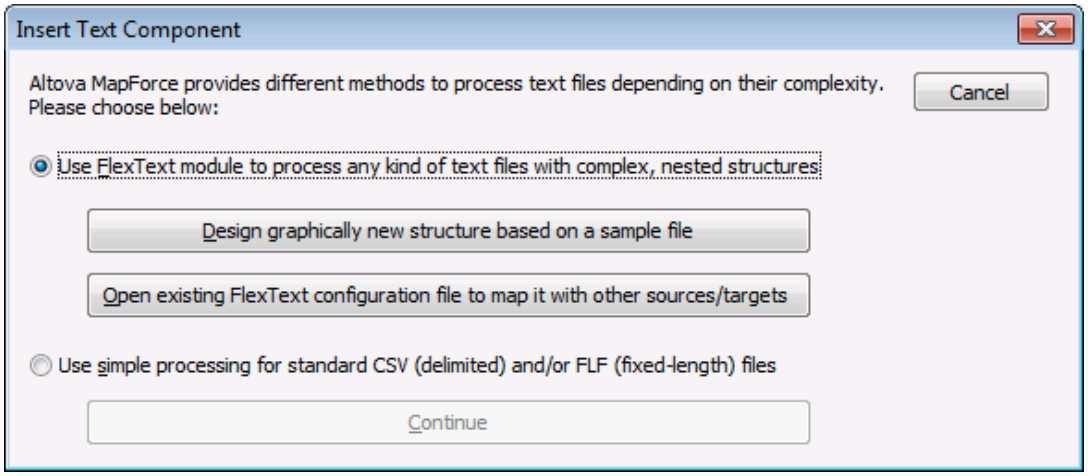
6. Use the same method to create a second text component that consists of five fields.



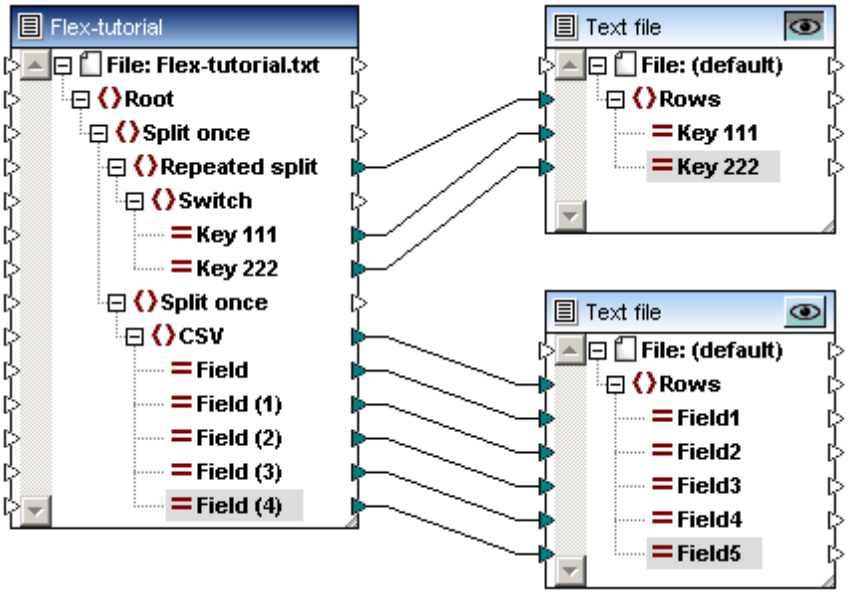
7.4.2.5 Step 5: Use the FlexText Template in MapForce

To use the FlexText template in MapForce:

1. Start, or switch back to MapForce, and select **Insert | Text file**.



2. Click the **Open existing FlexText configuration file...** button and select the previously defined FlexText template (**Flex-tutorial.mft**).
   The structure of the MapForce component mirrors that of the containers in Design view in FlexText.



3. Map the various items to the previously defined target components, and click the Output tab to preview the results.

Mapping preview of the top text component:

```
1  "111,332.1,22537.7,5,Container ship,Mega,
2  ",
3  "111,A1579227,10,3,400,Microtome,
4  ",
5  "111,B152427,7,6,1200,Miscellaneous,
6  ",
7  ,"222,978.4,7563.1,69,Air freight,Mini,
8  "
9  ,"222,ZZAW561,10,5,10000,Gas Chromatograph,,
10 "
11
```

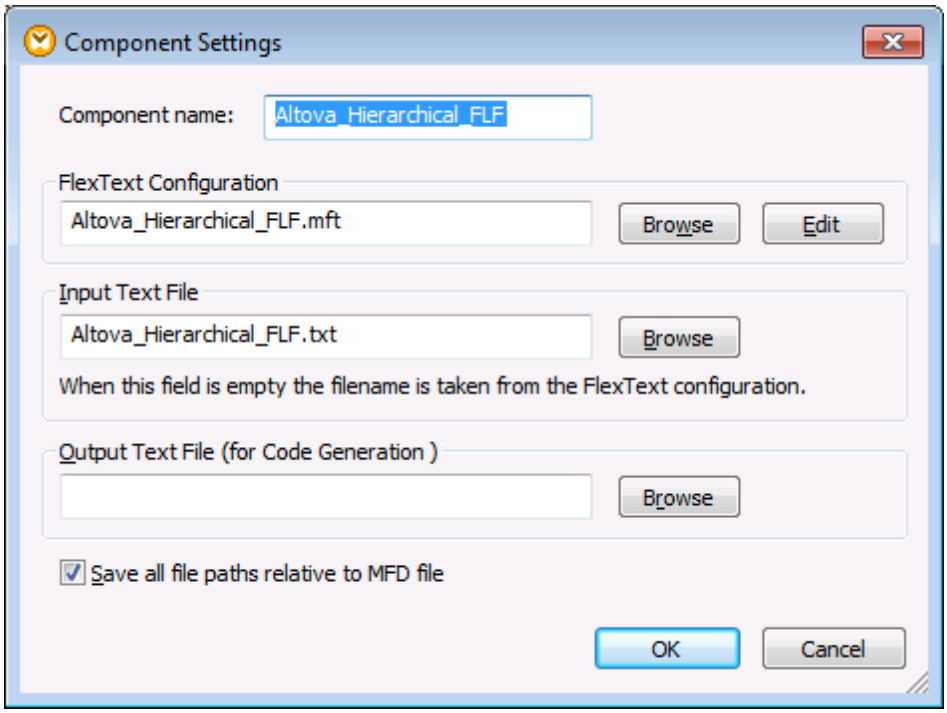
Mapping preview of the lower text component:

```
1   1100,897,22.1,716235,LOX
2   1110,9832,22991.30,002,NOX
3   1120,1213,33.01,008,SOX
4
```

7.4.3 FlexText Component Settings

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

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



FlexText Component Settings dialog box

The available settings are as follows.

Component name	The component name is automatically generated when you create the component. You can however change the name at any time. If the component name was automatically generated and you

	select an instance file after that, MapForce will prompt you to optionally update the component name as well. The component name can contain spaces and full stop characters. The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications when changing the name of the component: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have different encodings in Windows and at the command line.
FlexText Configuration	Specifies the name or path of the FlexText template (.mft) used by the component. To select a new template, click **Browse**. To edit the template in FlexText, click **Edit**.
Input Text File	Specifies the text file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it a text file. The source text file specified here takes precedence over the one defined in the FlexText template. If this field is empty, the filename defined in the FlexText template is used as source.
Output Text File (for Code Generation)	Specifies the name or path of the text file to which MapForce will write data. This field is meaningful for a target component when generating code. Entering a full path allows you to specifically define the target directory, for example, **c:\myfiles\sequence.txt**.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file.

7.4.4 Using FlexText as a Target Component

The main use of FlexText components is to reduce, or split off sections of text files and map the relevant items of the FlexText component to other target components. FlexText can, however, also be used as target component to assemble or reconstitute separate files into a single file, although this can entail a fair amount of trial and error to achieve the desired results.

Using a FlexText component as a **target** reverses the operations previously defined in it. Instead of splitting a file into various subsections, you assemble/reconstitute a file.

In general, the inverse of each operation defined in the FlexText template is carried out (in a bottom-up fashion) when using it as a target:

- "Split" becomes "merge", e.g. mapping to a repeated split delimited by "," becomes a merge items separated by ",".
- "Store" becomes "load".
- "Switch" becomes "choose the first match".

Note the following when using FlexText as target components:

- As soon as a connection is made between a data source component and one of the input items of a FlexText component, the FlexText component data source is ignored. The data provided by the newly mapped source component now takes precedence.
- If text is mapped to a "Store as..." (Store as CSV and FLF) container, then the separator is retained. However, text might be truncated if a fixed length split occurs in a node above the "Store as..." node.
- Fixed Width Splits truncate the left/top section if Split Base=Head, or the right/bottom section if Split Base=tail, to the predefined length. The truncated section is then as long as the defined length in characters. If the text is too short, then space characters are inserted to pad the section.
- FlexText would normally insert separators (or white space for fixed splits) between the items of a split operation, but this is not the case for 'delimited (line based)' splits. The 'delimited (line based)' operation is not a perfectly reversible operation. The "Delimited" text may occur anywhere in the first line and is included in the text, and therefore an automatic process cannot reliably add it.
  - Delimited (line based), will not add a separator to the first line if it is missing.
  - Delimited (floating), will add a separator between two sections.
- The switch operation cannot be inverted in a meaningful way except for simple cases. The switch scans its branches for the first branch that contains data, and uses/inserts this data. Only the first connection of a switch operation is mapped. To transfer data to the remaining switch containers, filters have to be defined for the remaining connectors and the duplication of the switch parent item is necessary, so that each switch item returns a single item which is then fed to a repeated split item to merge all of them.
- Mapping to a child of a single split container discards all mapping results except for the last item. Only a single result is retained, even if multiple results were generated.

The following analogy to the XML Schema content model gives some idea of FlexText's behavior when used as a target:

- A repeated split is a repeatable element.
- A single split forms part of a 'sequence' content model group.
- A switch forms a 'choice' content model group, each case being a possible child element.
- A store creates an element of simple type.

7.4.5 FlexText Reference

The reference section describes the various features of FlexText and shows how to use them to achieve specific results.

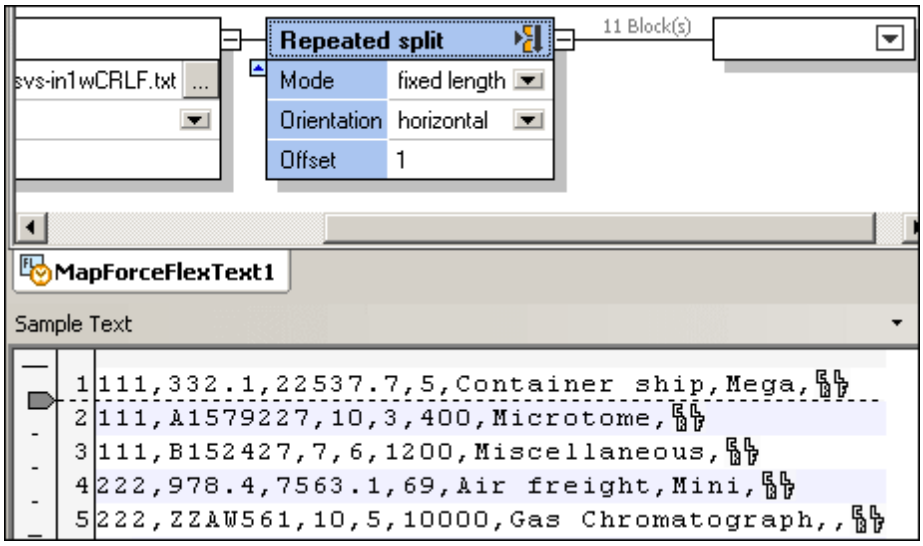
7.4.5.1 Repeated split

Using this option initially creates a single container. The container contains the text defined by the condition set in **Repeated Split**. There are several versions of the Repeated split option: Fixed length, Delimited Floating, Delimited Line based, and Delimited Starts with....

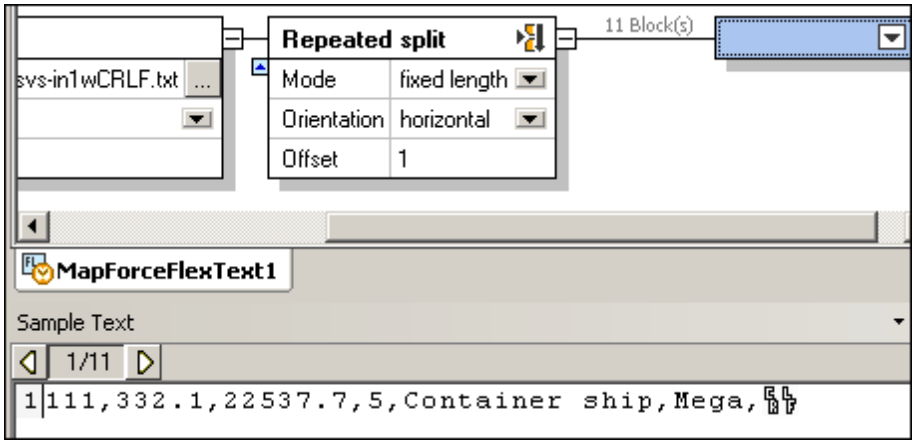
When you first select this option, default parameters are automatically set and the resultant fragments appear in the associated container. Note that the Repeated Split container is currently active, and the preview displays all current records/lines, in the Sample Text pane.

Container **default settings**:

Mode	fixed length
Orientation	horizontal
Offset	1



Default result:
Each line of text appears as a line/record in the new container, as the Offset is 1. Click the new container to preview its contents. The Sample Text **scroll arrows**, let you scroll through each of the 11 blocks/fragments produced by these settings.



Mode - Fixed length

Use the **Repeated split (fixed length)** mode when you want to split text into multiple horizontal or vertical fragments of fixed length. The settings applicable to the **Repeated split (fixed length)** mode are described below.

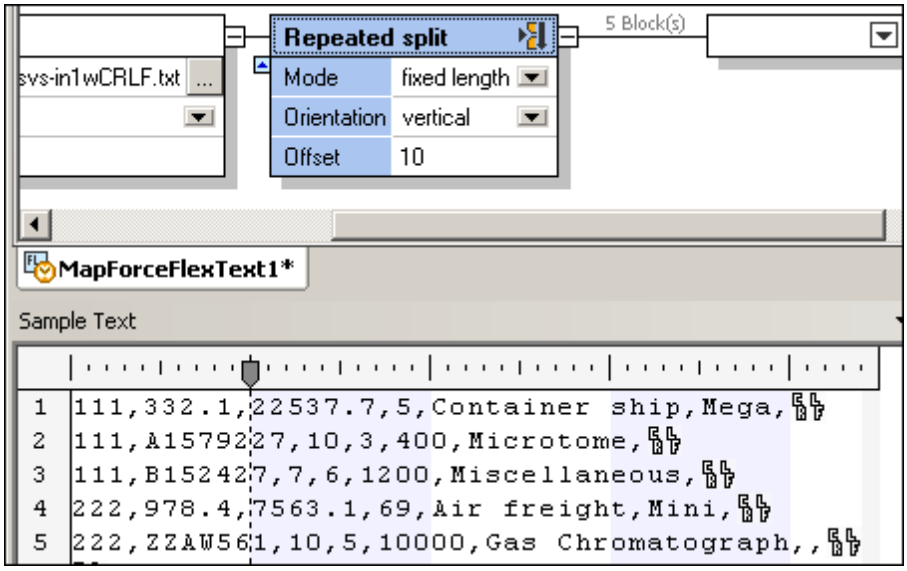
Orientation

Specifies if the text fragment is to be split horizontally or vertically. Choose "Horizontal" to split the fragment into multiple horizontal sections. Choose "Vertical" to split the fragment into multiple vertical columns. The default orientation is "Horizontal".

Offset

If orientation is set to "Horizontal", this setting specifies the number of lines that each fragment should contain. If orientation is set to "Vertical", this setting specifies the width in characters of each fragment. The default offset is "1". To modify the offset, do one of the following:

- Enter a value into the Offset field
- Drag the tab on the vertical or horizontal ruler.



Mode - Delimited (floating)

Use the **Repeated split with delimited (floating)** mode in the following situations:

- To split text where the separator characters that you specify must be stripped out from the resulting fragments
- To split text where the separators are in-line (for example, text that doesn't contain CR/LF

characters)

Note: A fragment is defined as the text between the first character after the separator, up to the last character before the next instance of the same separator. An exception to this rule are the first and last fragments, as shown in the example below.

The settings applicable to the **Repeated split with delimited (floating)** mode are described below.

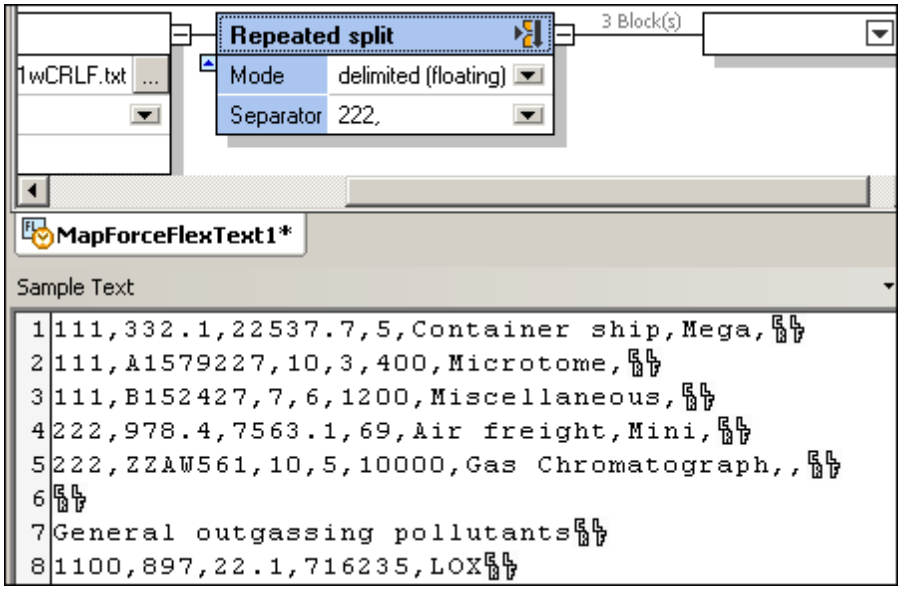
Regular expression

This is an optional setting which splits text into fragments whenever there is a regular expression match (see Splitting Text with Regular Expressions). The default value is "no".

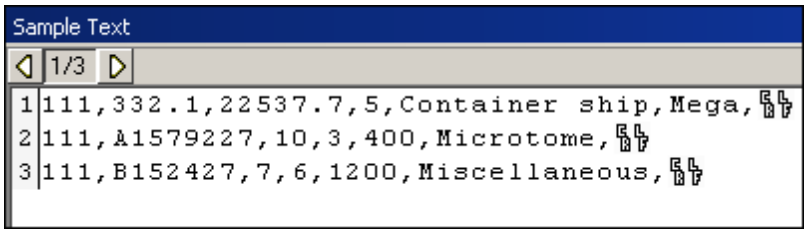
Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, using the separator "**222,**" against the text shown below produces three separate fragments.



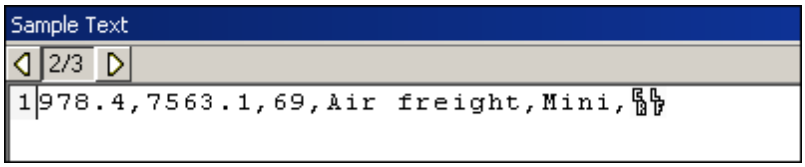
The first fragment contains all characters from the start of the fragment to the start of the first separator ("222,"), that is, from "111" to "Miscellaneous,".



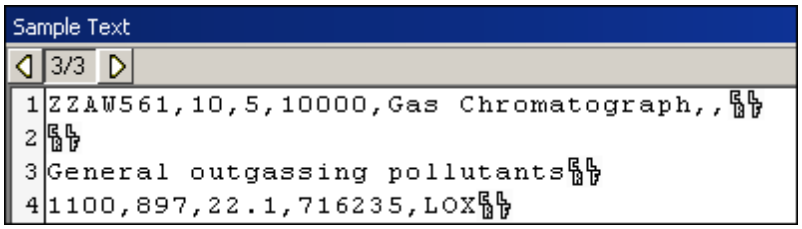
If the separator is not the first set of characters of the first line in the fragment, as in this example, then the first fragment includes all the text up to the first instance of the separator (for example, "222").

If "111" were the separator, then the first fragment would be a zero-length string, as the separator appears at the beginning of the first line of the source fragment.

The second fragment contains the first line containing the separator 222, without the separator.



The third fragment contains the next line containing the separator 222, without the separator itself, up to the end of the text file/fragment.



Mode - Delimited (line based)

Use the **Repeated split with delimited (line based)** mode to split text into multiple fragments, with the following behaviour:

- This mode creates multiple fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- A fragment is defined as the entire line containing the separator, up to the next line containing the same separator.
- If the separator does not appear in the first line, then the first fragment contains the line(s) up to the first line containing the separator.

The settings applicable to the **Repeated split with delimited (line based)** mode are listed below.

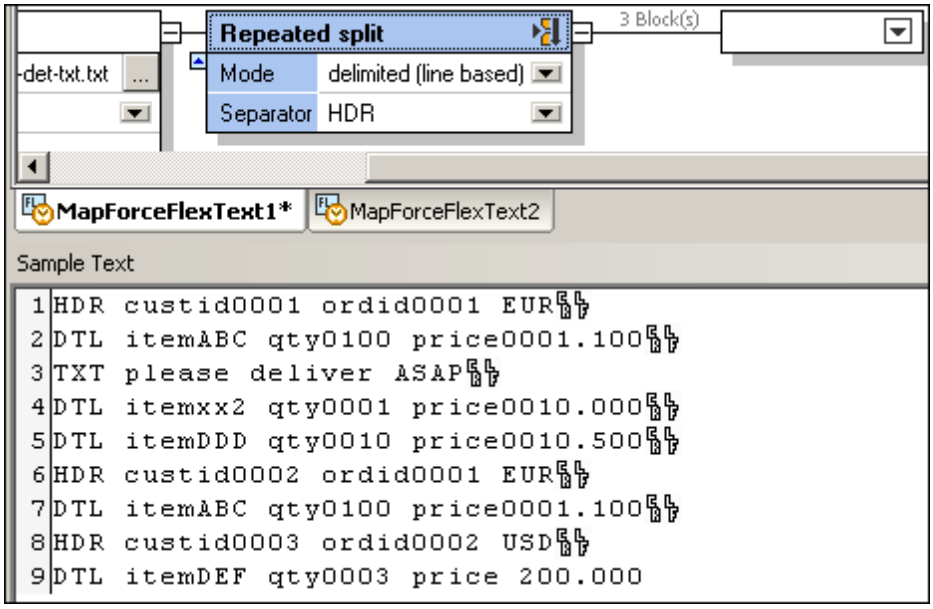
Regular expression

This is an optional setting which splits text into fragments whenever there is a regular expression match (see Splitting Text with Regular Expressions). The default value is "no".

Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, using the separator "**HDR**" against the text shown below produces three separate fragments.



The first fragment contains all characters from the start of the file/fragment, including all lines up to the next line containing the same separator.

```
1 HDR custid0001 ordid0001 EUR
2 DTL itemABC qty0100 price0001.100
3 TXT please deliver ASAP
4 DTL itemxx2 qty0001 price0010.000
5 DTL itemDDD qty0010 price0010.500
```

Note that this option allows you access to any number of lines between two separators. This is useful in files with record types that are optional or not in sequence (such as "DTL" or "TXT" in this example).

The second fragment contains all text from the second occurrence of "HDR" up to the next occurrence of "HDR".

```
1 HDR custid0002 ordid0001 EUR
2 DTL itemABC qty0100 price0001.100
```

The third fragment contains all text from the third occurrence of "HDR" up to the end.

```
1 HDR custid0003 ordid0002 USD
2 DTL itemDEF qty0003 price 200.000
```

Mode - Delimited (line starts with)

Use the **Repeated split with delimited (line starts with)** mode to split text into multiple fragments, with the following behaviour:

- This mode creates multiple fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- A fragment is defined as the entire line, starting with the separator, up to the next line containing the same separator at the start of the line.
- If the separator does not appear in the first line, then the first fragment contains the line(s) up to the first line containing the separator.

The settings applicable to the **Repeated split with delimited (line starts with)** mode are listed below.

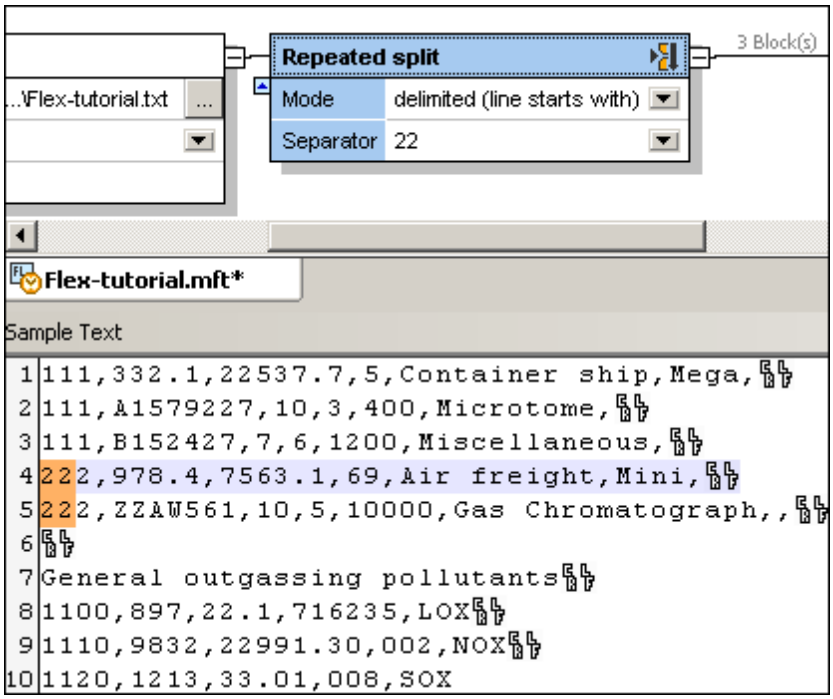
Regular expression

This is an optional setting which splits text into fragments whenever there is a regular expression match (see Splitting Text with Regular Expressions). The default value is "no".

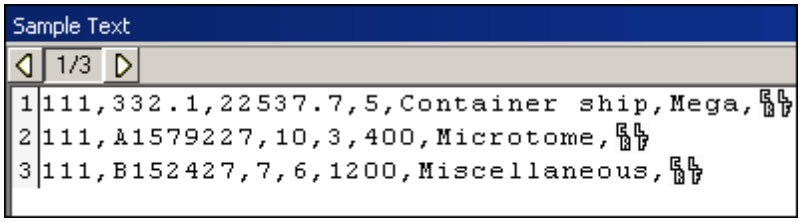
Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

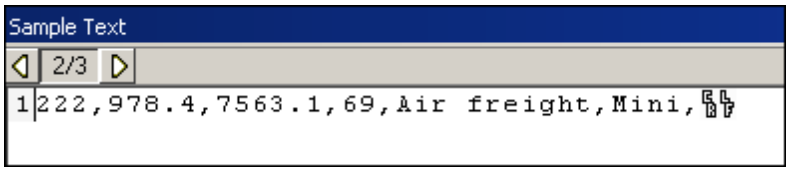
For example, using the separator "**22**" against the text below produces three separate fragments:



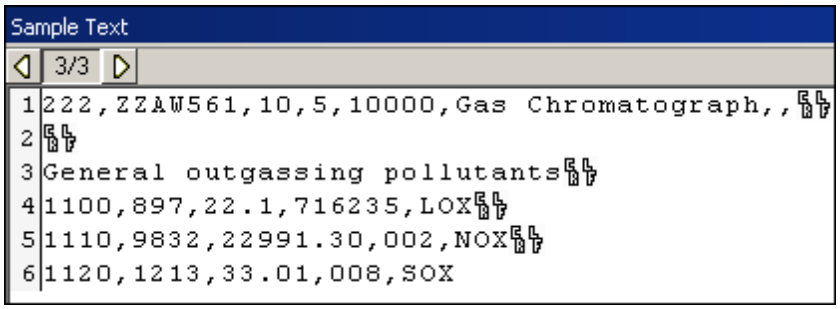
The first fragment contains all characters from the start of the file/fragment, including all lines up to the line containing the separator "22".



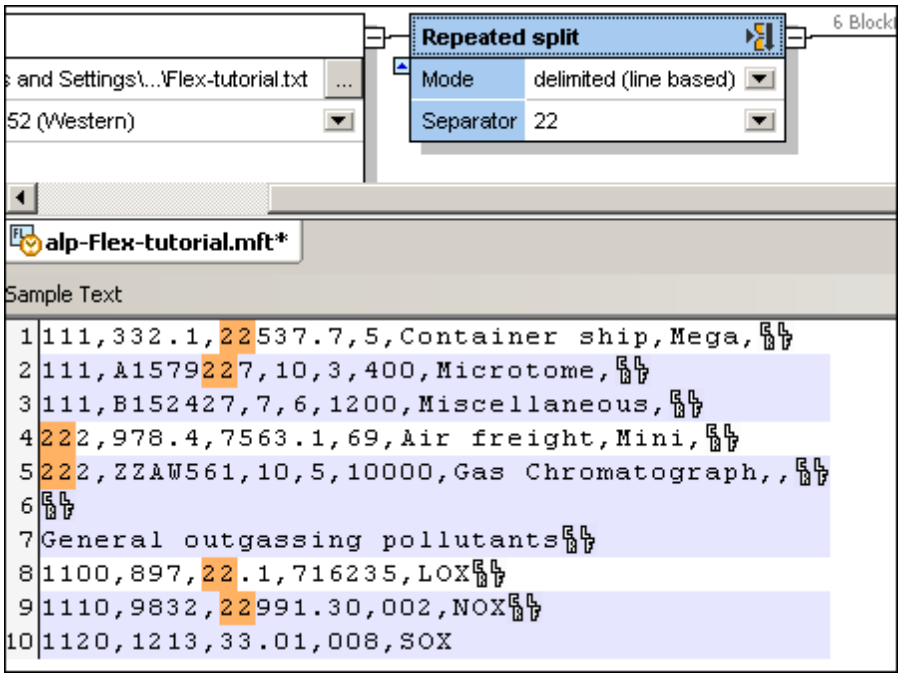
The second fragment contains all characters/lines from the second occurrence of "22", up to the next occurrence of "22", which in this case is only one line.



The third fragment contains all characters/lines from the third occurrence of "22", up to the end.



By contrast, here is what would happen if we used the **delimited (line based)** mode and separator as "22":



There would be six fragments, composed of lines that contained 22 anywhere in that line.

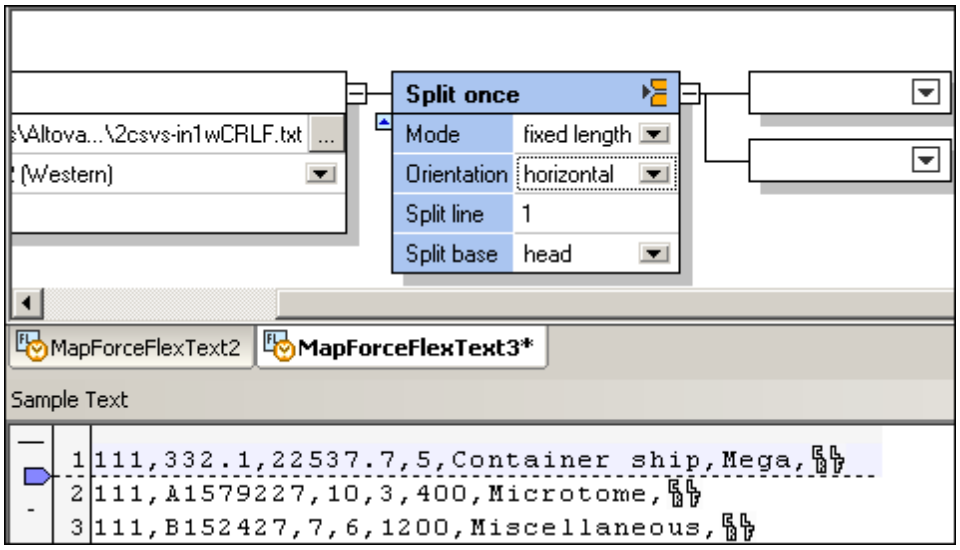
7.4.5.2 Split once

Using this option creates two vertically aligned containers. The top container contains the text defined by the condition set in the **Split once** container. The bottom container contains the rest of the text file/fragment. There are several versions of the Split once option: Fixed length, Delimited Floating, and Delimited Line Based.

When you first select this option, default parameters are automatically set, and the resultant fragments appears in both containers. Note that the **Split once** container is currently active, and displays a preview of all current records/lines, in the Sample Text pane.

Container **default settings** are:
Mode fixed length

Orientation horizontal
Split line 1
Split base head



Default result:
The first line of text appears in the top container. The bottom container contains the rest of the text file/fragment.

Mode - Fixed length

Use the **Split once (fixed length)** mode when you want to split text into two horizontal or vertical fragments, at a particular line or column relative to the beginning or end of the text. The settings applicable to the **Split once (fixed length)** mode are described below.

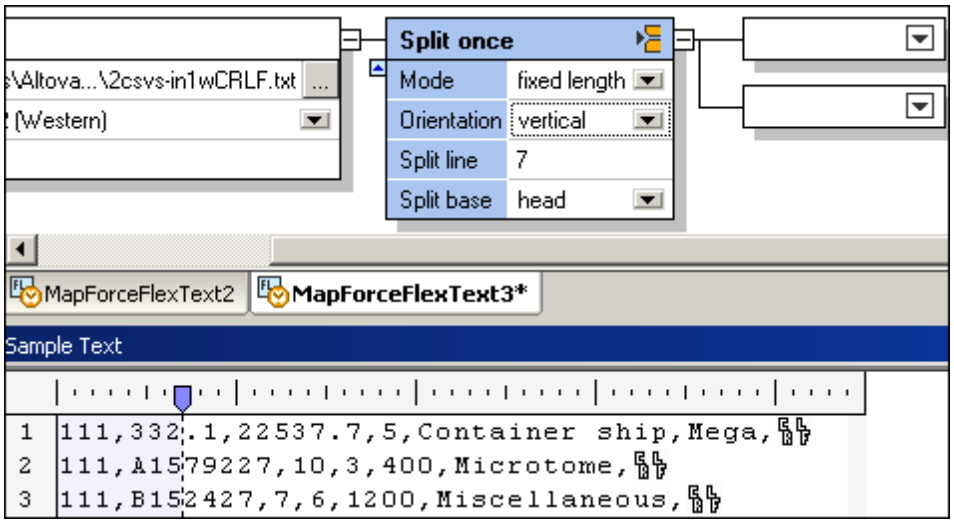
Orientation

Specifies if the text fragment is to be split horizontally or vertically. Choose "Horizontal" to split the fragment into two horizontal sections. Choose "Vertical" to split the fragment into two vertical sections. The default orientation is "Horizontal".

Split Line

Specifies the number of lines (or columns) after which the fragment should be divided into two. The default offset is "1". To modify this setting, do one of the following:

- Enter a value into the **Split line** field
- Drag the tab on the vertical or horizontal ruler.



Split base

Head	Look for the first occurrence of the separator starting from the beginning of text.
Tail	Look for the first occurrence of the separator starting from the end of text.

Mode - Delimited (floating)

Use the **Split once with delimited (floating)** mode to split text into two fragments, using a custom separator that is anywhere in the text. This is generally useful in files that do not contain CR, or LF characters, and you want to split the fragment into two, at some specific in-line location. Note the following:

- This mode creates two fragments defined by separator characters that you enter in the Separator field.
- The separator characters are not included in the fragment.
- The first fragment is defined as the text between the first character of the file/fragment, up to the last character before the separator.
- The second fragment is defined as the first character after the separator, up to the last character in the file/fragment.
- If the separator appears in the first/last position of the file/fragment, then the first of the two resulting containers remains empty.

The settings applicable to the **Split once with delimited (floating)** mode are listed below.

Regular expression

This is an optional setting which splits text into two fragments when there is a regular expression match (see Splitting Text with Regular Expressions). The default value is "no".

Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

Split base

Head	Look for the first occurrence of the separator starting from the beginning of text.
Tail	Look for the first occurrence of the separator starting from the end of text.

Mode - Delimited (line based)

Use the **Split once with delimited (line based)** mode to split text into two fragments, where the separator is anywhere in one of the lines. The line containing the separator is not split, but is retained whole. This is generally useful in files containing record delimiters (CR/LF), and you want to split the fragment into two separate fragments. Note the following:

- This mode creates two fragments defined by separator characters that you enter in the Separator field.
- The separator characters are included in the fragment.
- The first fragment is defined as all the text up to the line containing the separator.
- The second fragment is defined as the text, and line, including the separator up to the end of the file/fragment.
- If the separator appears in the first/last line, of the file/fragment, then the top container remains empty.

The settings applicable to the **Split once with delimited (line based)** mode are listed below.

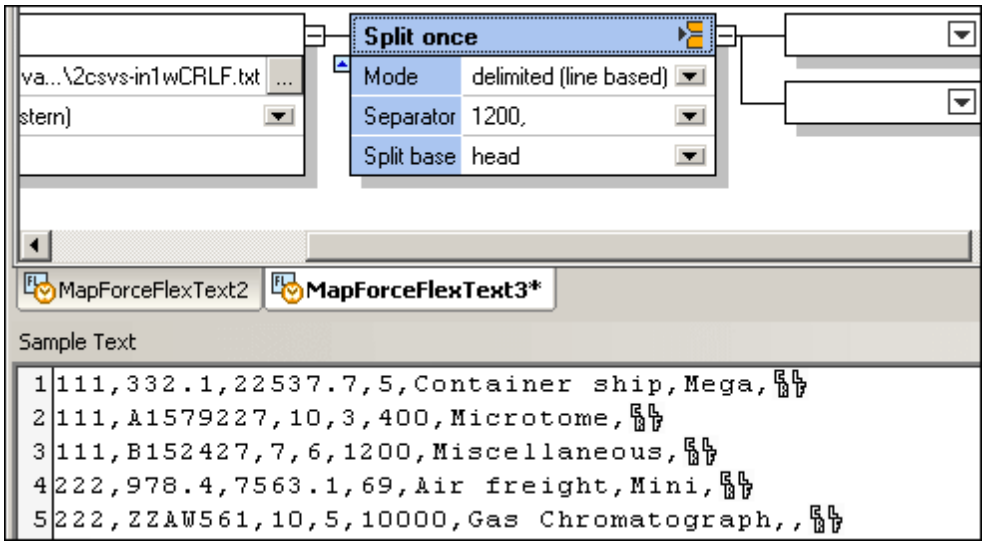
Regular expression

This is an optional setting which splits text into two fragments when there is a regular expression match (see Splitting Text with Regular Expressions). The default value is "no".

Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

For example, if you use the separator "1200," against the text below, two fragments are created.



The first fragment contains the text up to the line containing the separator.

```
1 111,332.1,22537.7,5,Container ship,Mega,
2 111,A1579227,10,3,400,Microtome,
```

The second fragment contains the entire line containing the separator, and all remaining lines up to the end.

```
1 111,B152427,7,6,1200,Miscellaneous,
2 222,978.4,7563.1,69,Air freight,Mini,
3 222,ZZAW561,10,5,10000,Gas Chromatograph,,
```

Split base

Head	Look for the first occurrence of the separator starting from the beginning of text.
Tail	Look for the first occurrence of the separator starting from the end of text.

Mode - Delimited (line starts with)

Use the **Split once with delimited (line starts with)** mode if the split should occur at the first line which begins with the specified separator. When you select this mode, two fragments of text are created, as follows:

- The first fragment contains all the text up to the line where the separator is.
- The second fragment contains the remaining text, including the line where the separator is.
- If the separator is at the beginning of the first line, no split occurs (that is, the first resulting fragment remains empty).

The settings applicable to the are listed below.

Regular expression

This is an optional setting which splits text into two fragments when there is a regular expression match (see Splitting Text with Regular Expressions). The default value is "no".

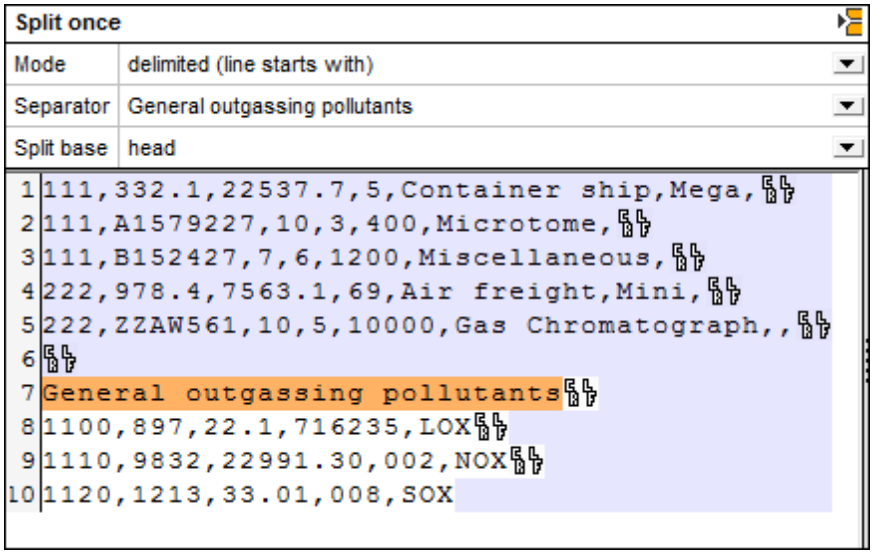
Separator

Specifies the character(s) to be used as separator. The default value is "none" (no separator).

Split base

Head	Look for the first occurrence of the separator starting from the beginning of text.
Tail	Look for the first occurrence of the separator starting from the end of text.

For example, in the image below, the split occurs at the first line which begins with the specified separator ("General outgassing pollutants"). If the Split base option was set to **tail**, then the split would have occurred at the first line which begins with the specified separator, starting from the end (which in this example happens to be the same line).



The first resulting fragment is as follows:

```
111,332.1,22537.7,5,Container ship,Mega,
111,A1579227,10,3,400,Microtome,
111,B152427,7,6,1200,Miscellaneous,
222,978.4,7563.1,69,Air freight,Mini,
222,ZZAW561,10,5,10000,Gas Chromatograph,,
```

The second resulting fragment is as follows:

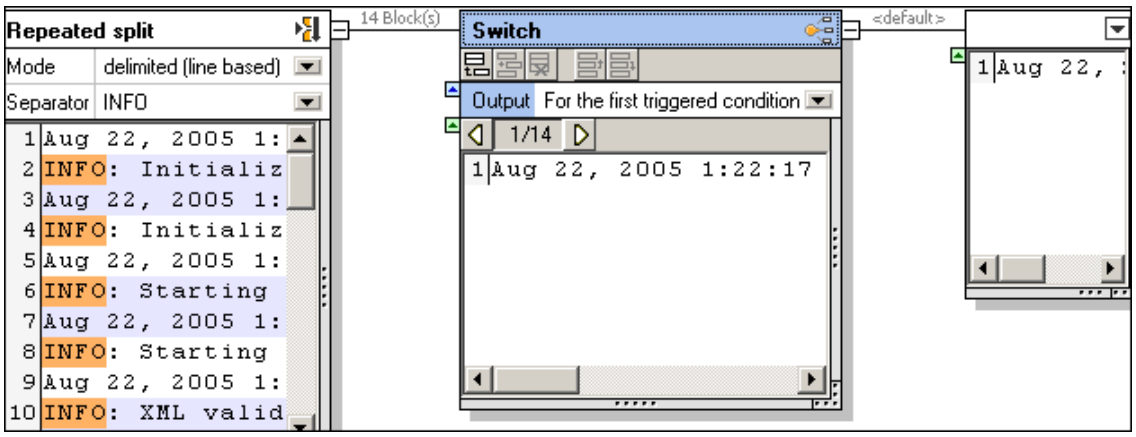
```
General outgassing pollutants
1100,897,22.1,716235,LOX
1110,9832,22991.30,002,NOX
1120,1213,33.01,008,SOX
```

7.4.5.3 Switch

Using the Switch option allows you to define multiple keywords, or conditions, for a single text fragment. Each keyword you define, has its own associated container which receives data only if the specific condition is satisfied, i.e. true. If none of the conditions are satisfied, then the specific fragment is mapped to a "default" container.

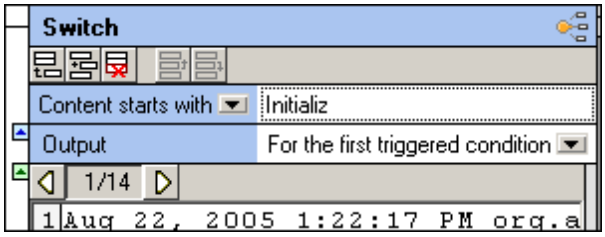
Container **default settings** are:
Output For the first triggered condition.

The example below processes a Tomcat log file, where the individual processes are to be separated out, and made mappable. When you first define a Switch container, only the **default** container appears to the right of the Switch container. All data is automatically passed on to it.



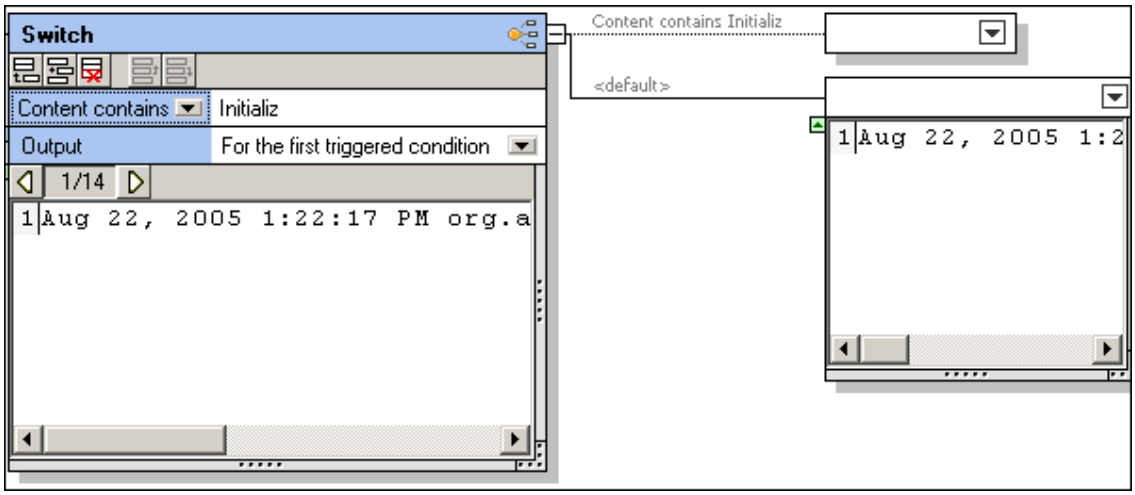
The repeated split container, using delimited (line based), separates all INFO sections out of the log file and passes them on to the Switch container.

1. Click the append icon  to add a new condition to the Switch container.
2. Double click in the "**Content starts with**" field, enter "**Initializ**" and hit Return.

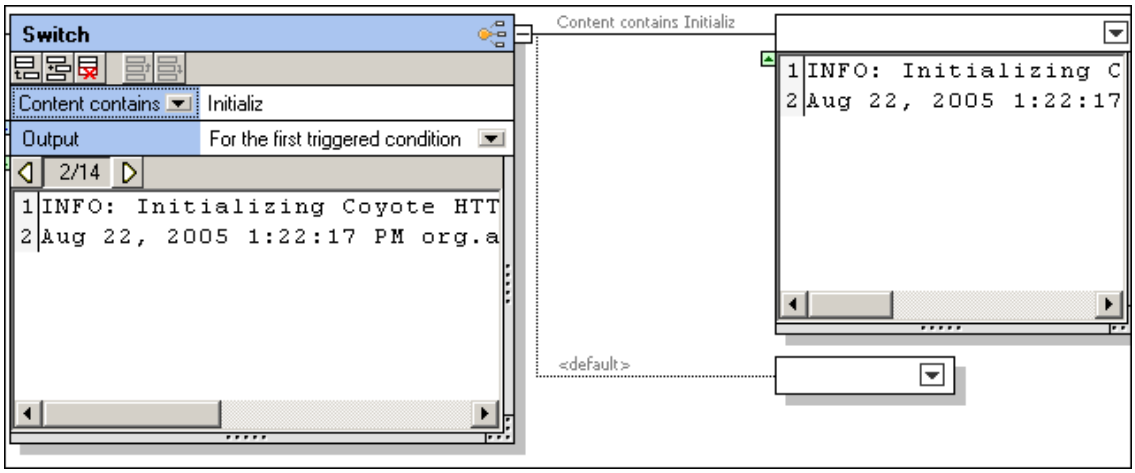


A new container is added. Data will be forwarded to this container if the condition is true. If not, the data is forwarded to the default container.

3. Click the "**Content starts with**" combo box, and change it to "Content contains".
   The first condition has now been defined and you can see the result below.
   The first fragment does not contain "Initializ", and its contents are therefore forwarded to the **default** container.

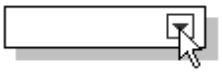


4. Click the Display next block icon  , to see the next text fragment.

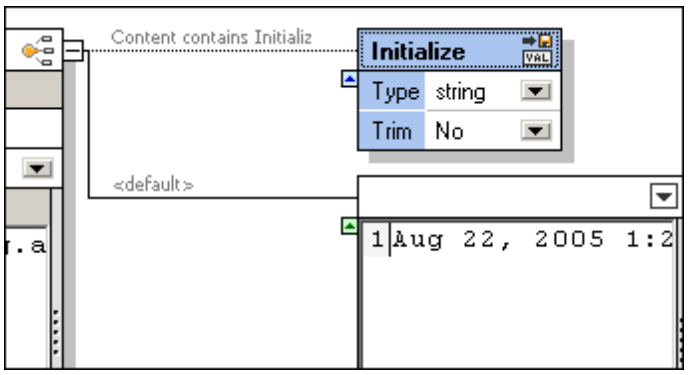


The Initializing... fragment now appears in its associated container, and the default container is empty. Stepping through the fragments gives you a preview of what the individual containers hold.

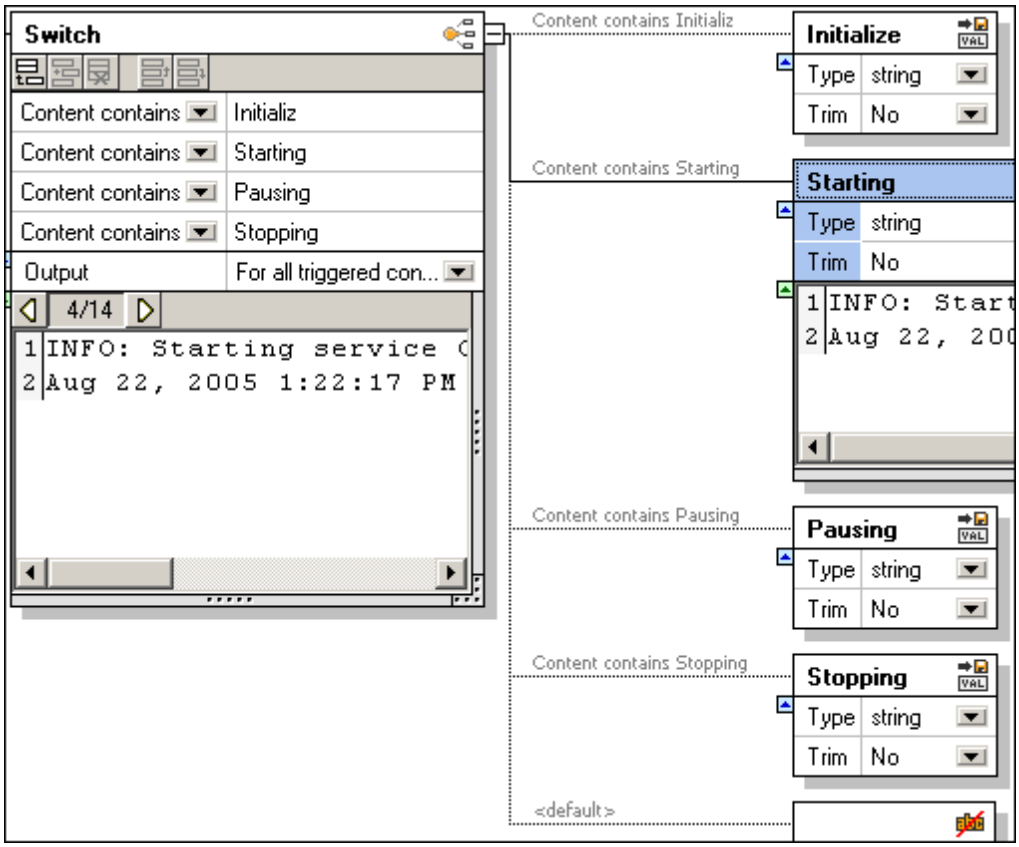
5. Click the container icon button, and select **Store as value**.



6. Double click in the "Store" title bar and change the text e.g. Initialize.



7. Click the append icon  to add a new condition to the Switch container.
8. Double click in the "**Content starts with**" field, enter "Starting" and hit Return.
   You can add as many conditions as you need e.g. Pausing, and Stopping.
   Give each of the associated containers a name, to make recognition in MapForce easier.

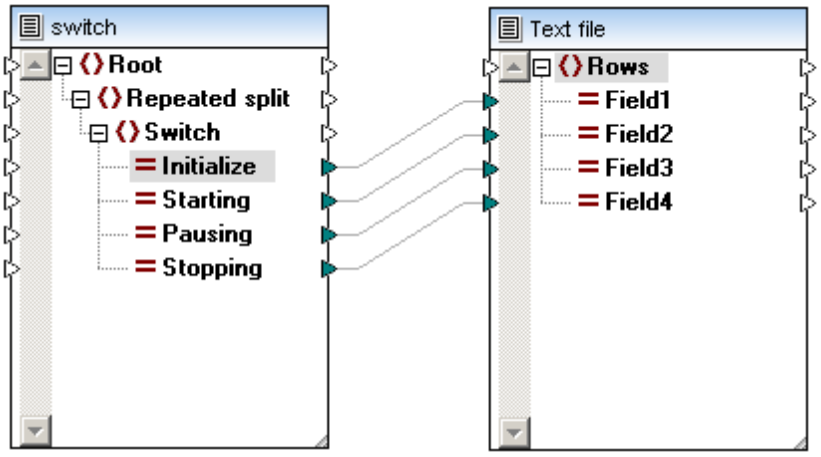


The screenshot above shows all four conditions, and the contents of the "Starting" container at block/fragment no 4. The associated containers have all been renamed to make identification in the MapForce component easier.

Note that conditions can be moved up and down in the condition list, using the respective

Move Up/Down buttons , or .

9. Save the template and insert it in MapForce.



Please note:

If a text fragment in the current fragment satisfies a condition, then the **complete data** of that fragment is passed on to the associated container. Data is not split up in any way, it is just routed to the associated containers, or to the default container if it does not satisfy any of the defined conditions.

The associated containers produced by Switch, can be used for further processing. You can change such a container to Split once, Repeated split, or anything else if you wish.

Content starts with:
Data is only passed to the associated container, if the condition string appears at the start of the text fragment.

Content contains:
Data is passed on to the associated container, if the condition string appears anywhere in the text fragment.

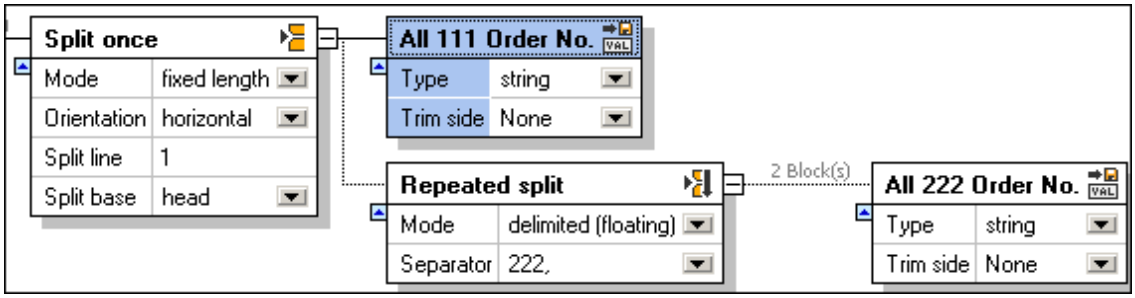
For the first triggered condition:
Data is passed on when **one** of the **conditions** in the condition list is **true**. Any other conditions that are true are ignored, and no data is passed on to any of the associated containers.

For all triggered conditions:
Outputs data for **every** condition that is true in the condition list. This makes it possible to have multiple occurrences of the same data/fragment in multiple associated containers simultaneously. This might occur if a text fragment contains text that satisfies two conditions simultaneously e.g. "initializing starting sequence" in the example above.

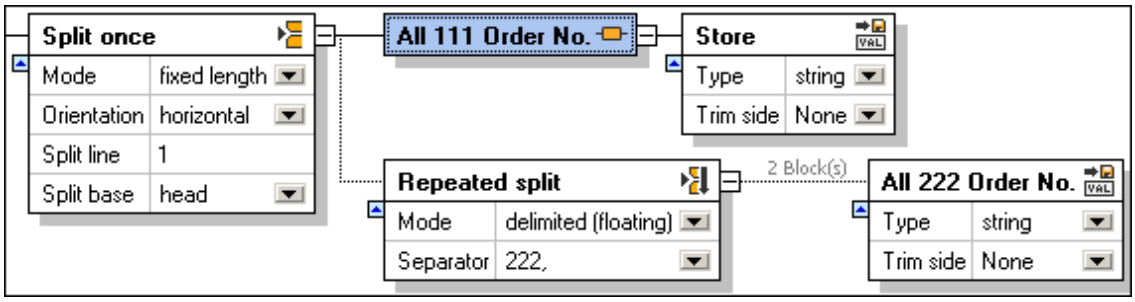
7.4.5.4 Node

Allows you to add a new hierarchical level to the FlexText, and MapForce tree structures. The data that the following node/container contains, is passed on as is.

In the screenshot below, the "All 111 Order No." container is the last container in the top branch.



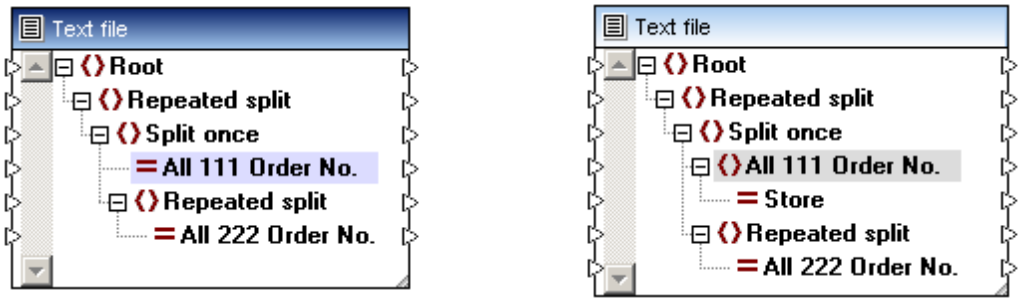
Click the top-right icon of the container, and select **Node** from the context menu.



A new container has been added to the right of the current one.

Please note:
The automatically appended container was then manually defined as "**Store as value**".

The screenshot below shows both template structures as they appear when inserted into MapForce.

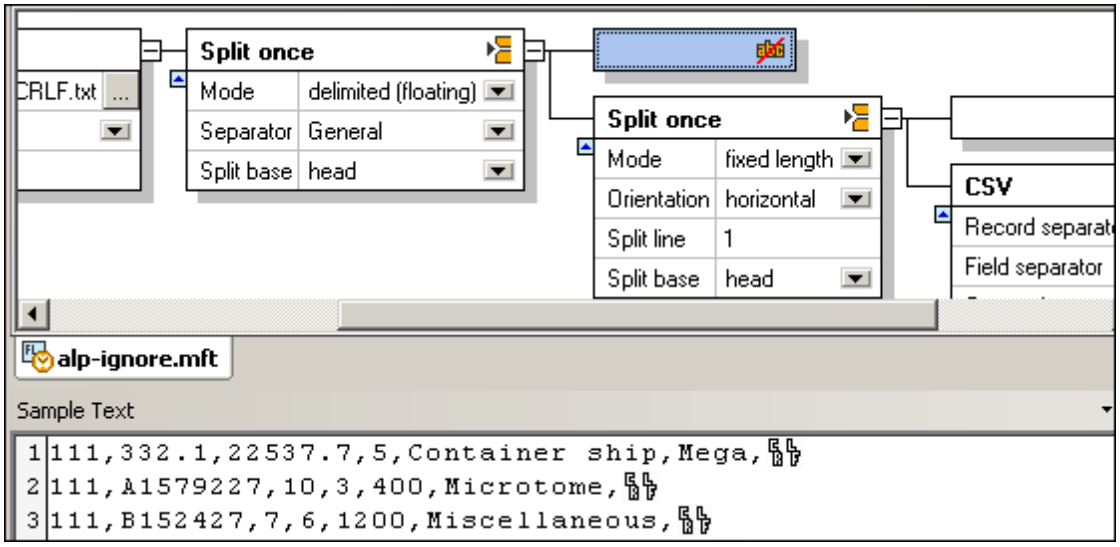


The left component shows the initial structure before adding the new Node.

The right component shows how the component structure has changed. "All 111..." is now a parent item, and a new child item "Store" has been added below it.

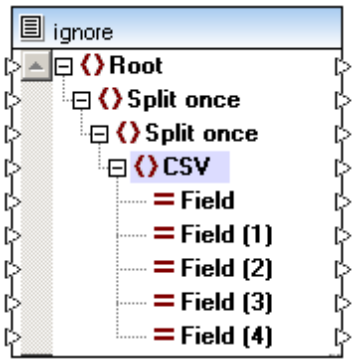
7.4.5.5 Ignore

Allows you to suppress the output of a specific text fragment. What this means, is that the container and any data it may contain, will not be made available as a mappable item in the FlexText component in MapForce.

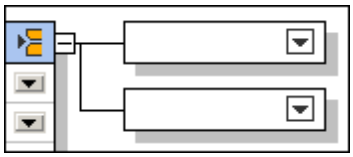


In the example shown above, the active container has been set to "Ignore". The Sample text that it contains will therefore not appear as a mappable item in MapForce.

The text template when inserted into MapForce, has the structure shown below. There is no mappable item between the two "Split once" items.



Please note:
Default "ignore" containers also exist. These are the new containers that are automatically appended when selecting "Split once" and "Repeated split" etc.



The contents of these containers are not initially mappable/available to MapForce when the template is inserted. You have to select one of the container options in FlexText: Store as value, Store as CSV etc., to be able to map them.

7.4.5.6 Store as CSV (delimited)

Store CSV allows you to store text fragments as CSV text, and map individual columns to

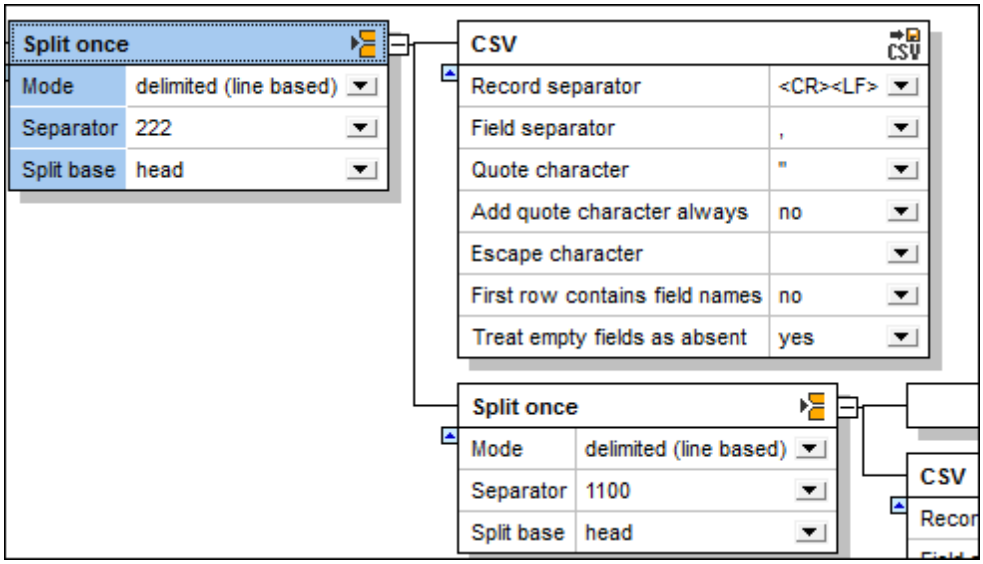
MapForce. Any number of CSV containers/components can be created in FlexText, and each of the CSV containers may have different separators.

The Sample Text pane provides an overview of the current CSV fragment, and also allows you to specify individual field names, and field types. Each column appears as a mappable item in the FlexText component in MapForce.

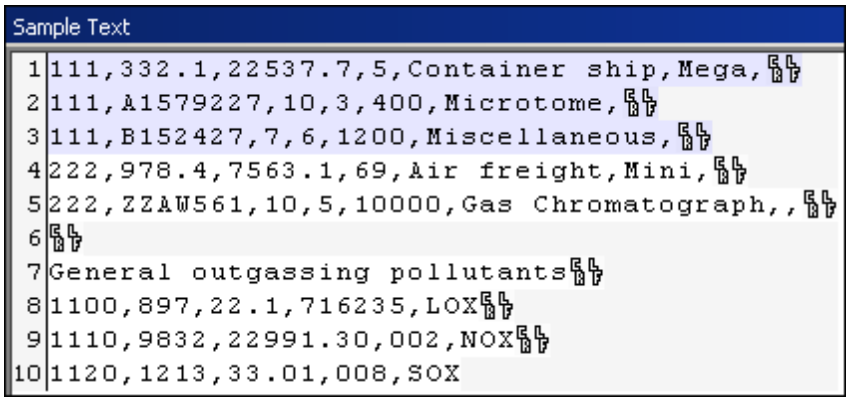
Container **default settings** are:

Record separator	CR LF
Field separator	,
Quote character	"
Add quote character always	no
Escape character	(none)
First row contains field names	no
Treat empty fields as absent	yes

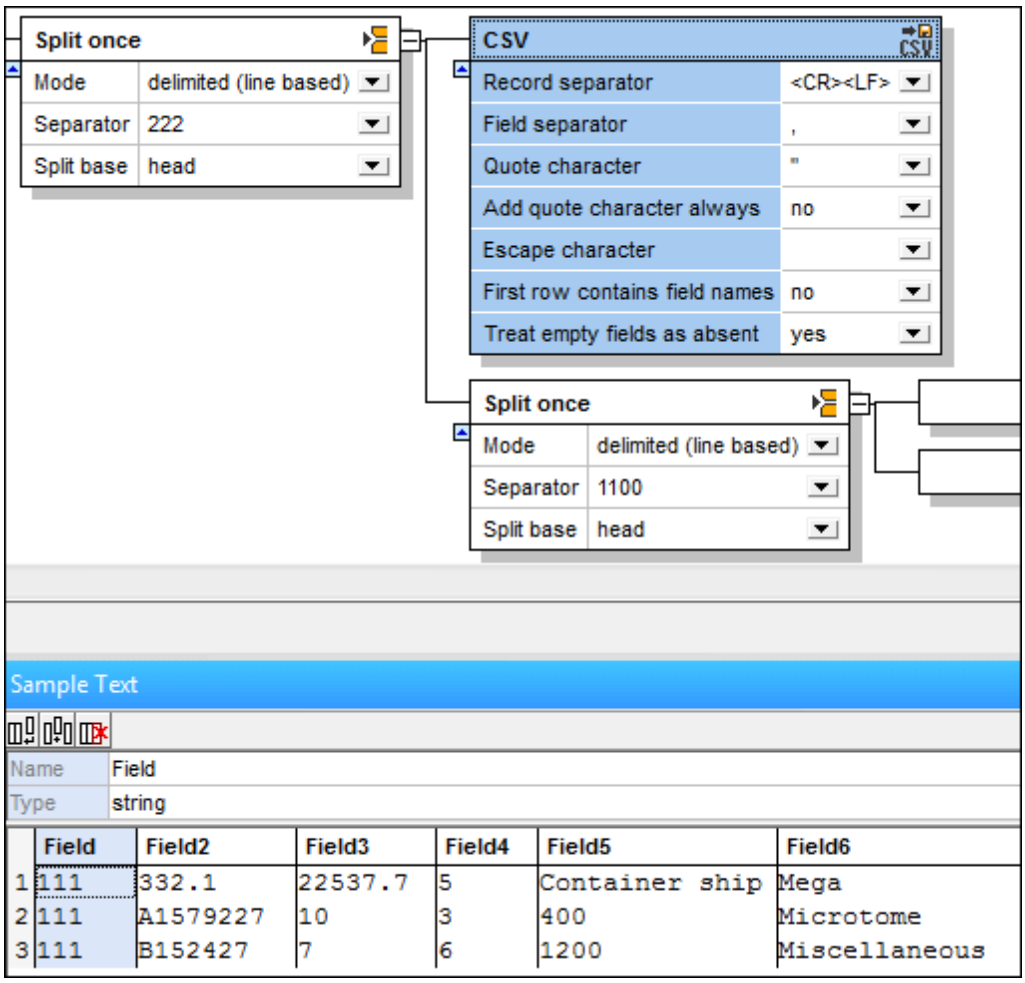
The following example shows how data in a small text file is split up into two CSV files, and mapped to separate XML files in MapForce.



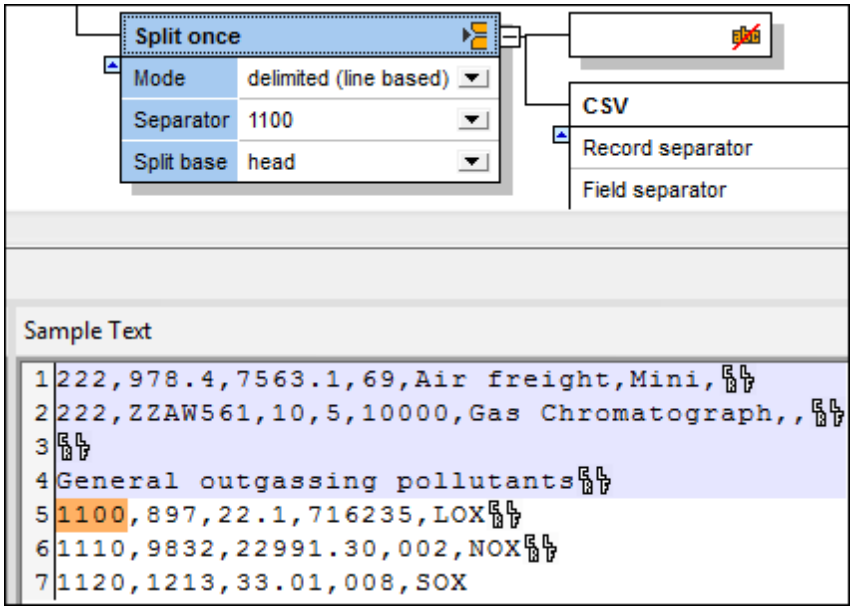
The **Split once** container shown above, is used to create two containers. The **delimited (line based)** function with the separator 222, is used to achieve this. All records up to the first occurrence of 222, are passed to the CSV container. The first, consisting of all records containing 111, is then defined as a CSV container. The Sample Text pane shows the contents of the currently active container "Split once".



The default CSV settings have not been changed. Clicking the CSV container shows its contents in tabular form.

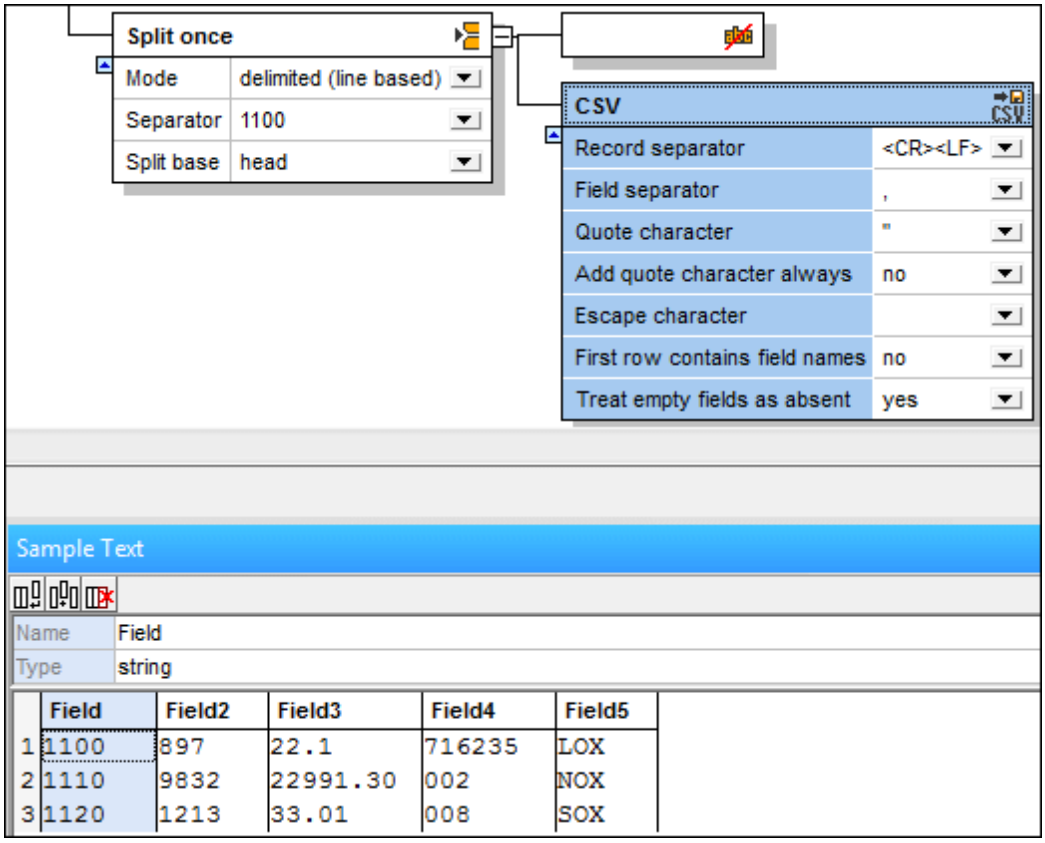


The second container holds the remaining data, and is made into another **Split once** container. This creates two more containers, one of which will be the second CSV. Clicking the Split once container, shows the current contents.

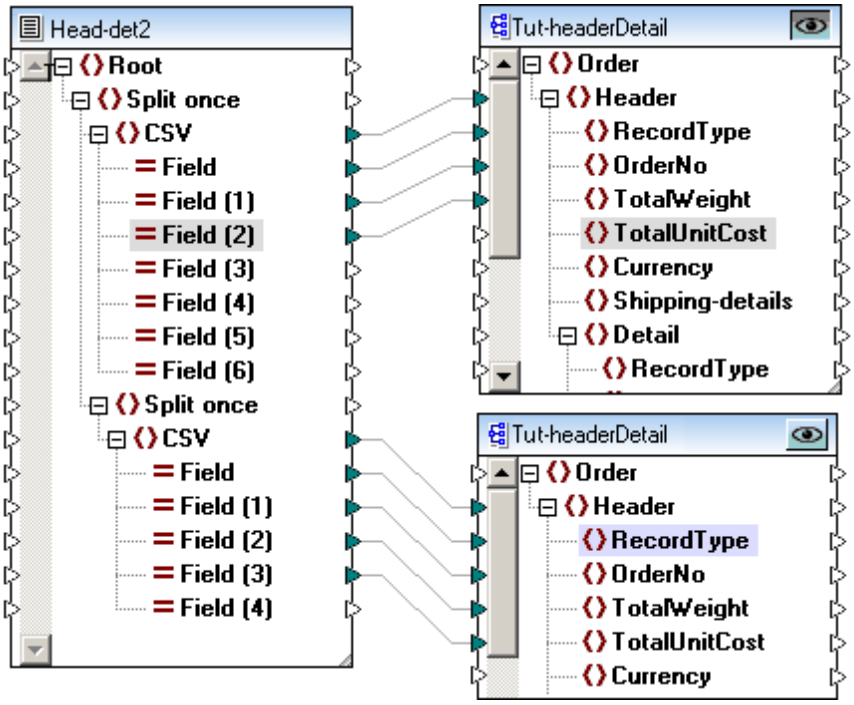


The delimited (line based) function, using 1100 as the separator, is used to split the remaining data into two sections.

- All records up to the first occurrence of 1100, are passed to the first container which is made non-mappable, by defining it as "Ignore" .
- The second container is then defined as CSV. The default settings have not been changed. Clicking the CSV container shows the contents in tabular form.



Inserting the FlexText template into MapForce allows you to map the data to any of the supported target files. In this example, each of the CSV items are mapped to two separate XML files.



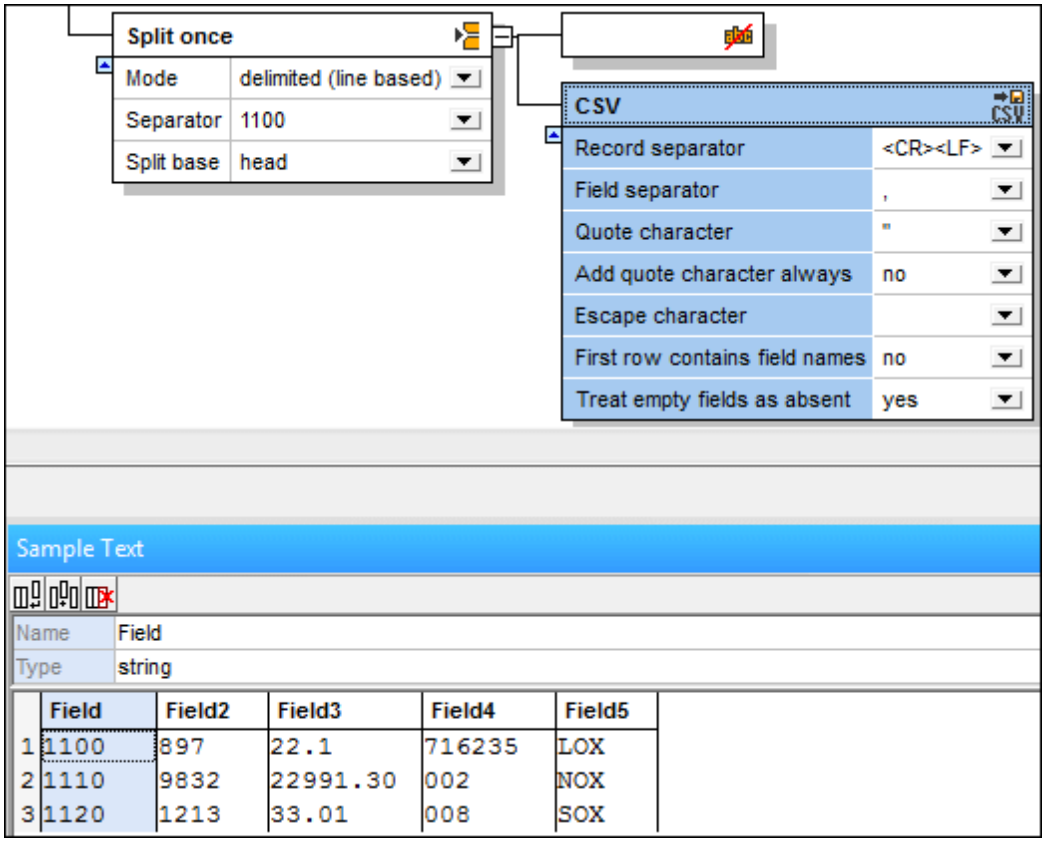
Note that not all of the items in the CSV sections are mapped to the target files.
The first XML file contains all 111 record types.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>332</OrderNo>
    <TotalWeight>22537.7</TotalWeight>
  </Header>
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>0</OrderNo>
    <TotalWeight>10</TotalWeight>
  </Header>
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>0</OrderNo>
    <TotalWeight>7</TotalWeight>
  </Header>
</Order>
```

The second XML file contains all records starting with 1100.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Header>
    <RecordType>1100</RecordType>
    <OrderNo>897</OrderNo>
    <TotalWeight>22.1</TotalWeight>
    <TotalUnitCost>716235</TotalUnitCost>
  </Header>
  <Header>
    <RecordType>1110</RecordType>
    <OrderNo>9832</OrderNo>
    <TotalWeight>22991.3</TotalWeight>
    <TotalUnitCost>2</TotalUnitCost>
  </Header>
  <Header>
    <RecordType>1120</RecordType>
    <OrderNo>1213</OrderNo>
    <TotalWeight>33.01</TotalWeight>
    <TotalUnitCost>8</TotalUnitCost>
  </Header>
</Order>
```

Configuring the CSV container/data:



Clicking a field in the Sample Text pane highlights it, allowing you to configure it further.

- Click in the **Name** field to edit the default text that is presented.
- Click in the **Type** field to define the field data type: string, boolean, decimal etc.
- Click the append icon to append a new field.
- Click the insert icon to insert a field before the currently active field.
- Click the delete icon to delete the currently active field.

Please note:
The field boundaries can be dragged by the mouse to display the data.

Add quote character always

Allows you define if the specified quote character is to be added to all fields of the generated CSV file.

Treat empty fields as absent

Allows you to define that empty fields in the source file, will not produce a corresponding empty item (element or attribute) in the target file.

Note that the delimiters for the empty fields in the source file must exist however, e.g. "General outgassing pollutants**,,,,**".

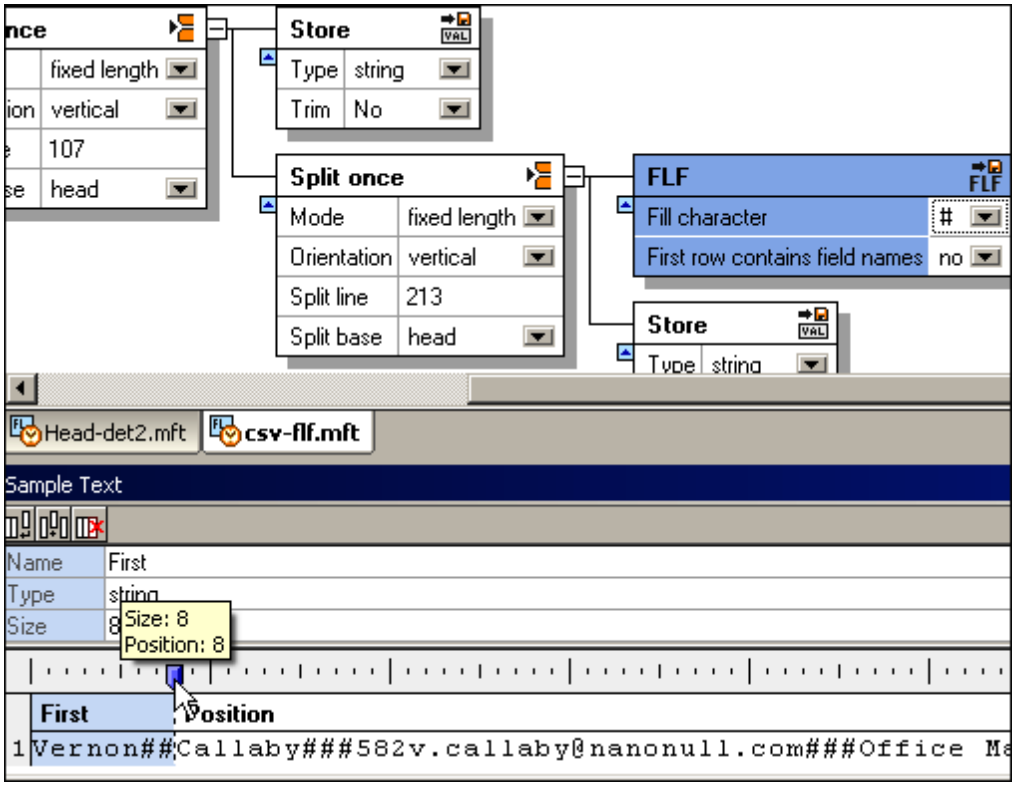
7.4.5.7 Store as FLF (fixed length)

Store FLF allows you to store text fragments as fixed length text, and map individual columns to MapForce. Any number of FLF containers/components can be created in FlexText, and each of the FLF containers may have different fill characters.

The Sample Text pane provides an overview of the current FLF fragment, and also allows you to specify field names, lengths, and widths. Each column appears as a mappable item in the text component in MapForce.

Container **default settings** are:

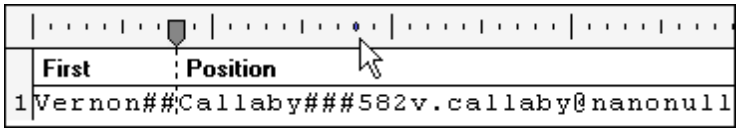
Fill character	(none)
First row contains field names	no
Treat empty fields as absent	yes



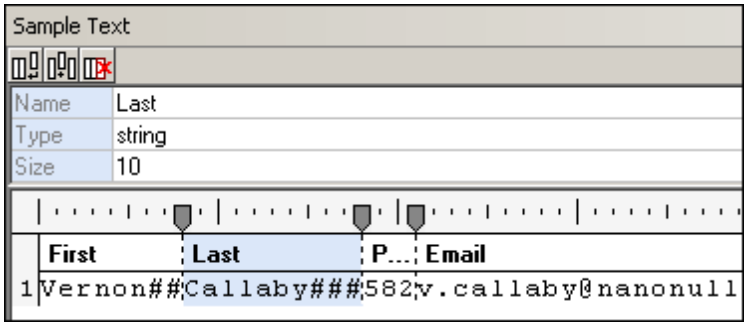
Configuring the FLF container/data

Having defined a container as "Store FLF", the Sample Text pane appears as shown in the screenshot above. A default field of width 10 is automatically inserted.

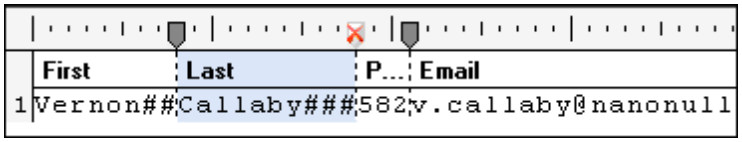
- Click the tab icon on the ruler and drag to reposition it. A tip appears showing you the current position.
- Positioning the cursor over the ruler displays a "dot"; clicking places a new tab at the click position.



- Having defined the new position, click the field to select it, and edit the name in the Name field.



- To **remove** a field, click the tab icon and drag it off the ruler. The tab icon changes when this action can be successfully completed.

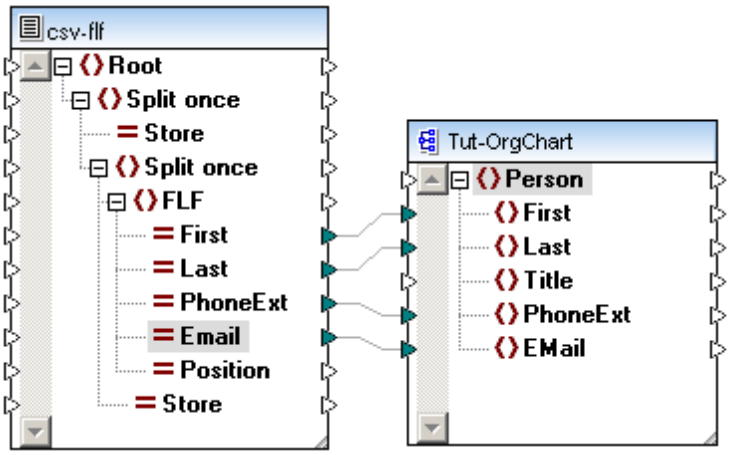


Note: When you drag a tab on the ruler, all tabs to the right of it will be automatically repositioned. To retain the other tab positions, hold down the **Shift** key before moving the tab.

Clicking a field in the Sample Text pane highlights it, allowing you to further configure it.

- Click the append icon to append a new field, of length 10.
- Click the insert icon to insert a field before the currently active field, length 10.
- Click the delete icon to delete the currently active field.
- Click in the **Name** field to edit the default text that is presented.
- Click in the **Type** field to define the field data type: string, boolean, decimal etc.

Inserting the FlexText template into MapForce allows you to map the data to any of the supported target files. In this example, FLF items are mapped to XML items.



If the option **Treat empty fields as absent** is **yes**, then any empty fields in the source file will not produce a corresponding empty item (element or attribute) in the target file. A field is considered as absent if there is no data between two subsequent fill characters.

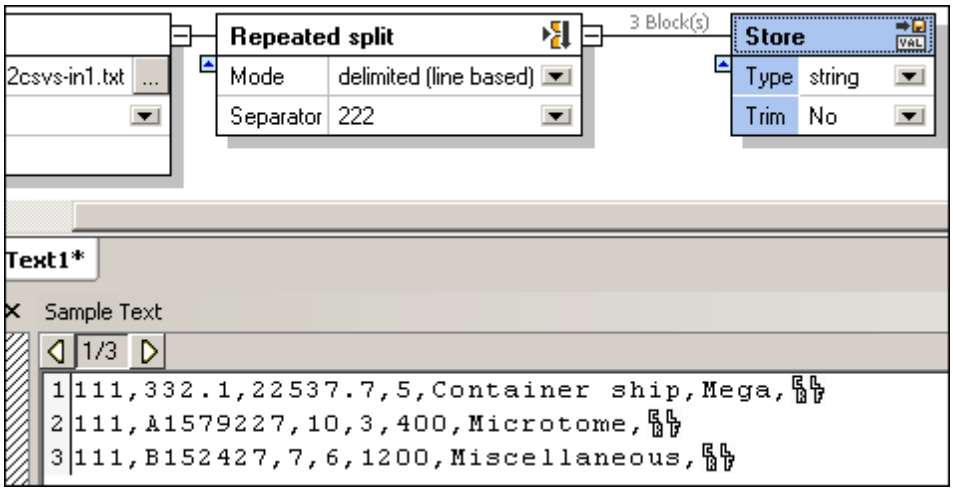
7.4.5.8 Store value

Allows you to define a container, which makes its data available as a mappable item, in MapForce. If you do not change the container name in FlexText, then the mappable item appears with the name "Store".

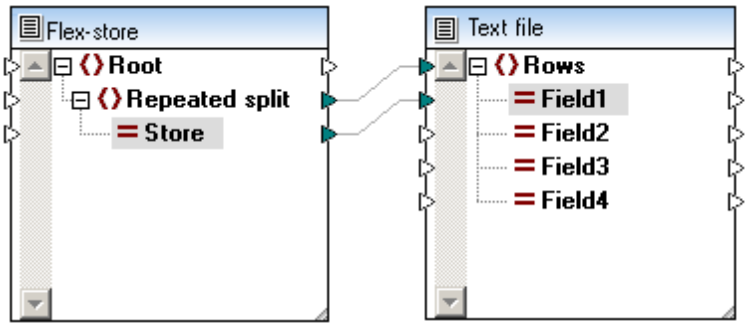
Container **default settings** are:

Type	string
Trim	no

The screenshot below shows the "Store" container with its contents visible in the Sample Text pane.



Saving this template and opening it in MapForce, allows you to map the Store item to other items in a target component.



Please note:

The field1 item in the target text file, will contain all 3 fragments supplied by the Store item, when you click the Output tab to preview the result.

Type
Allows you to define the data type of the text fragments.

Trim side
Defines the side from which the characters will be trimmed, left, right or both. Selecting Yes, activates the "Trim character set" option.

Trim character set
Defines the characters you want to trim from this text fragment. You can enter any number of characters here, by double clicking in the field. The characters you enter are removed from the Trim side(s) of the fragment.

7.4.6 FlexText and Regular Expressions

In MapForce FlexText, you can use regular expressions as follows:

1. To split text containers whenever a match is found (the matched text acts as separator). In this case, regular expressions are implicitly anchored; therefore, the caret (^) and the dollar sign ($) characters are not used. For example, to match any three consecutive digits, use `[0-9]{3}` instead of `^[0-9]{3}$`.
2. To redirect text from a Switch container if the text contains a regular expression match.

You can use regular expressions in FlexText components in any of the following MapForce target languages:

- Built-in (when previewing the mapping)
- Built-in (when running the MapForce Server execution file)
- Code generation languages (C++, C#, Java). Note that, in these languages, some advanced features of regular expressions may depend on the regular expressions implementation in that specific language.

The regular expression syntax and semantics in FlexText is based on http://www.w3.org/TR/xmlschema-2/#regexs, similar to the MapForce core function **`tokenize-regexp`**. Note the following:

- If the split condition matches two sequences following each other, FlexText creates an empty result in between (this behavior is the same when you are not using regular

expressions).
- No regular expression flags (http://www.w3.org/TR/xquery-operators/#flags) are supported.

This section includes the following topics:

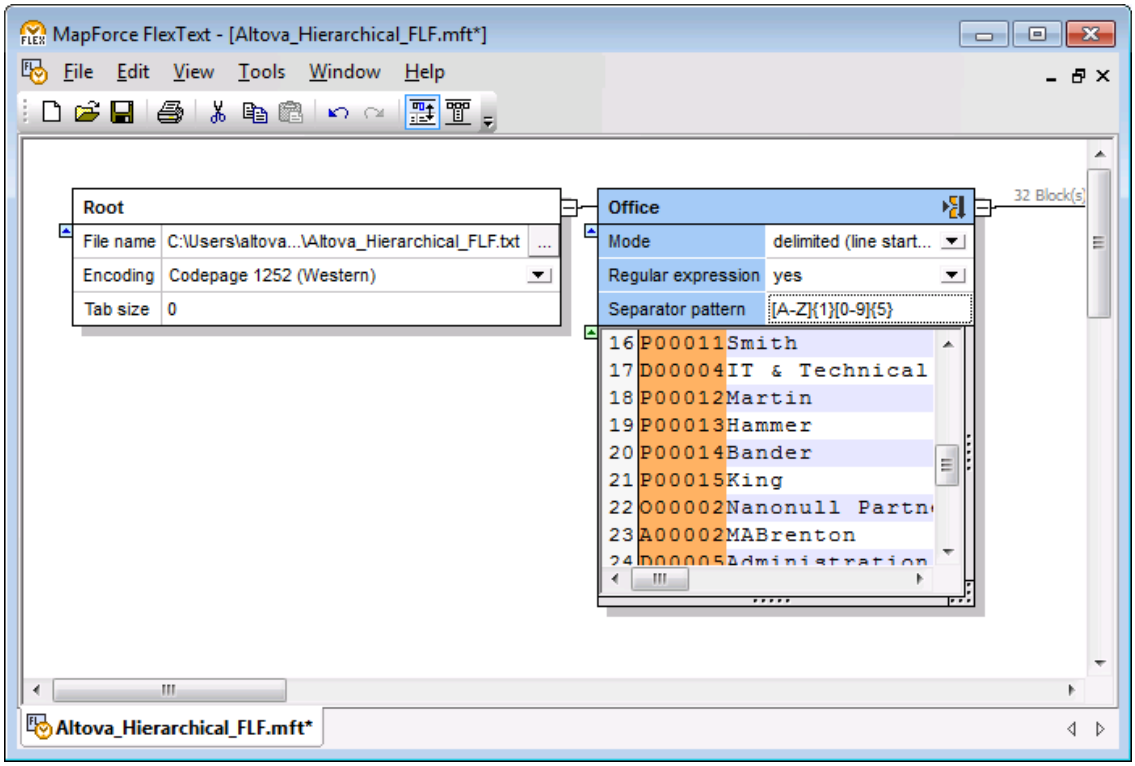
- Splitting Text with Regular Expressions.
- Using Regular Expressions in Switch Conditions

7.4.6.1 Splitting Text with Regular Expressions

When you need to split text into two or more fragments, you can optionally use a regular expression as separator. This is an alternative, more advanced option as compared to separating text by means of single or multiple consecutive characters.

The option to split text by means of regular expressions becomes available in FlexText when the following conditions are true:

- The container is of type **Split Once** or **Repeated Split**.
- The *Mode* option is set to **delimited (floating)**, **delimited (line based)**, or **delimited (line starts with)**.



Sample FlexText template which uses regular expressions to separate text

The options applicable to regular expressions are as follows.

Regular expression	To use a regular expression as text separator, switch this

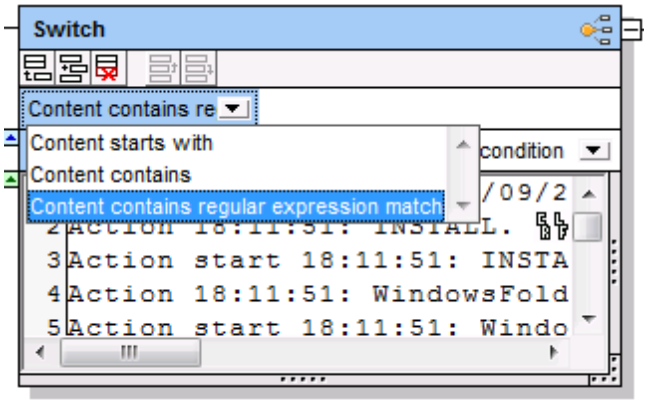
	option to **yes**. To use simple text as separator, switch this option to **no** (this is the default value).
Separator pattern	If the *Regular expression* option is switched to **yes**, a *Separator Pattern* text box becomes available where you can enter the regular expression that must act as separator. In the FlexText template illustrated above, the regular expression `[A-Z]{1}[0-9]{5}` matches exactly one alphanumeric character, followed by exactly five numeric characters (for example, "P00011"). All such occurrences are highlighted in the preview and act as text splitting separator.
Separator for writing	The *Separator for writing* option is meaningful when the following conditions are true: • The FlexText component is a target component • *Mode* is set to **delimited (floating)**. Enter in this field the string to be written to the target component at the occurrence where a regular expression match was found.

7.4.6.2 Using Regular Expressions in Switch Conditions

When working with Switch containers (see also Switch), you can optionally create a condition within the Switch container to redirect the text fragment if it contains a regular expression match. In such cases, you can use the the caret (^) and the dollar sign ($) characters to match the beginning or end of the text to be searched, except when C++ is set as target transformation language. (In C++, the caret and dollar sign characters are interpreted as the beginning or end of a line, not as the beginning or end of the whole text).

To use a regular expression in a switch condition:

1. Define the container type as Switch (click on the top-right corner of the container, and then click **Switch**).
2. Click the **Append Condition** () button to add a new condition.
3. Set the condition type to **Content contains regular expression match** and enter the regular expression in the adjacent text box.



Example

Let's assume that you need to map data from the database log file represented below (you can also find this file at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\SampleDatabaseLog.txt**).

```
Action 18:11:51: INSTALL.
Action start 18:11:51: INSTALL.
Action 18:11:51: WindowsFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Action start 18:11:51: WindowsFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: WindowsFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Return value 1.
Action 18:11:51: SystemFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Action start 18:11:51: SystemFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: SystemFolder.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E.
Return value 0.
Action 18:11:51: WindowsFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Action start 18:11:51: WindowsFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: WindowsFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Return value 0.
Action 18:11:51: SystemFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Action start 18:11:51: SystemFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: SystemFolder.9BAE13A2_E7AF_D6C3_FF1F_C8B3B9A1E18E.
Return value 1.
Action 18:11:51: WindowsFolder.63E949F6_03BC_5C40_FF1F_C8B3B9A1E18E.
Action start 18:11:51: WindowsFolder.63E949F6_03BC_5C40_FF1F_C8B3B9A1E18E.
Action ended 18:11:51: WindowsFolder.63E949F6_03BC_5C40_FF1F_C8B3B9A1E18E.
Return value 1.
```

Your goals are as follows:

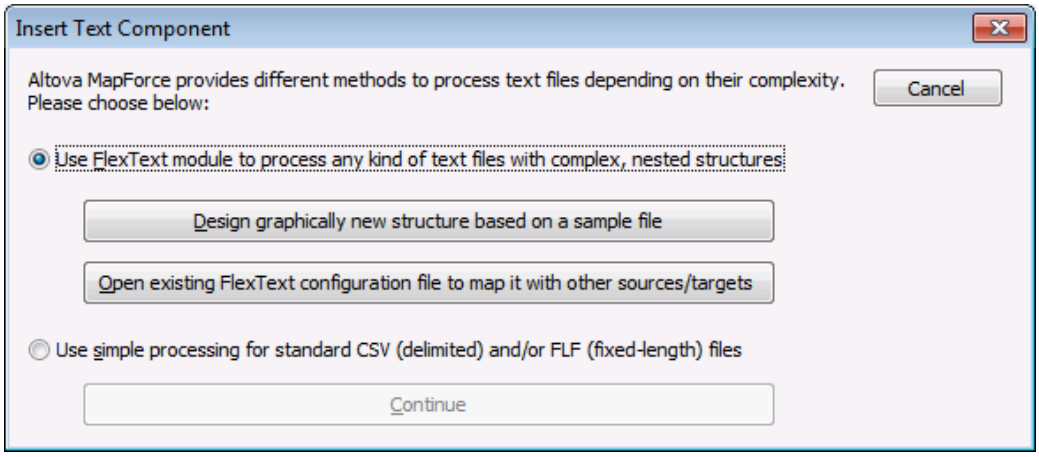
1. Collect in a list all rows where a return value is present. That is, the list must include every row which contains the value "Return value 1" or the value "Return value 0", or any other return value expressed as a digit.
2. Collect in another list all rows where the text contains the value "Action start".

To achieve these goals, you can use a *Repeated Split* container to split down the text into individual rows. After that, you can use a *Switch* container to redirect each row as required. Namely, the *Switch* container will consist of three conditions, as follows:

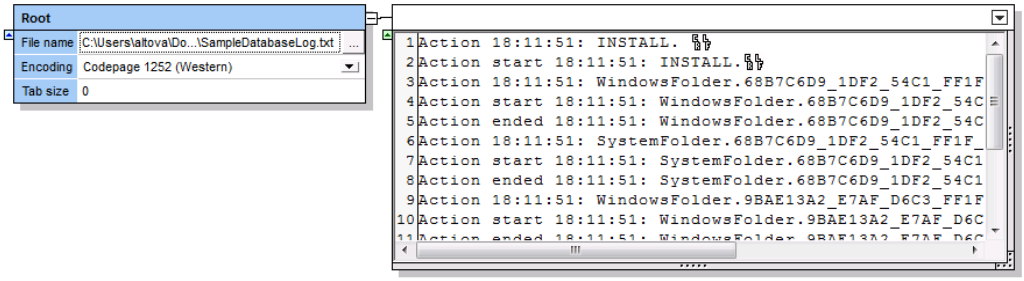
1. Redirect the current row to output A if it contains the value "Action started". You can find such rows by using a condition of type "Content starts with", and enter "Action started" as value.
2. Redirect the current row to output B if it contains a return value. You can find such rows by using the regular expression `Return value [0-9]\.` . This regular expression will return a match if the row contains the text "Return value", followed by a single digit, followed by a full stop. The backslash (\ ) before the full stop acts as an escape character, to denote that the full stop must be treated as normal character, not as a metacharacter.
3. Redirect the current row to output C (<default>) if the row satisfies neither of the conditions above.

To create the FlexText template which performs the tasks above:

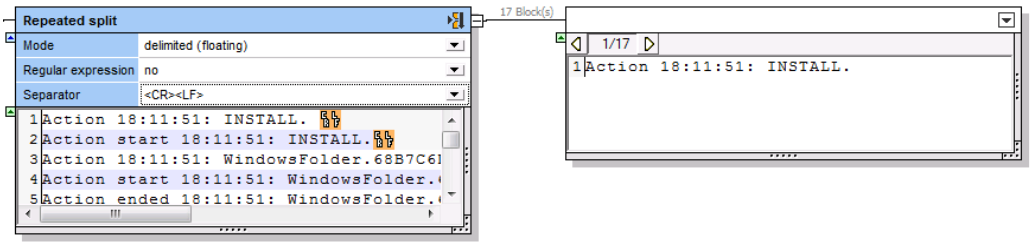
1. On the **Insert** menu, click **Text File**.



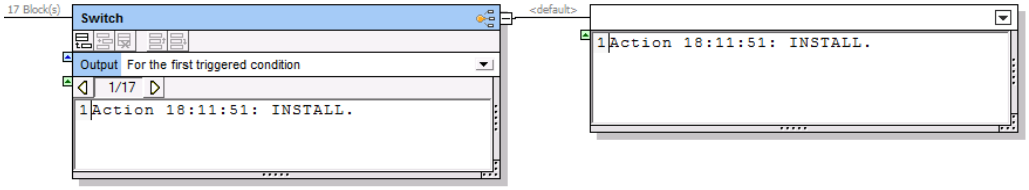
2. Click **Design graphically new structure based on a sample file** and save the FlexText .mft template to a directory of your choice.
3. When prompted to open a text file, browse for the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\SampleDatabaseLog.txt** file.



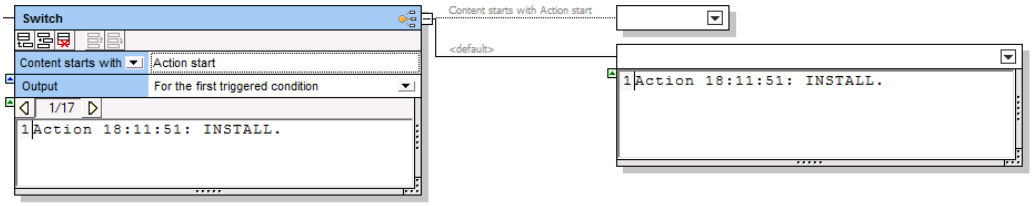
4. Click the top-right corner of the output container and select **Repeated Split**. Because we are using carriage return as split character, choose **delimited (floating)** mode, and **<CR><LF>** as separator. This creates a new output container which consists of 17 blocks (one for each row).



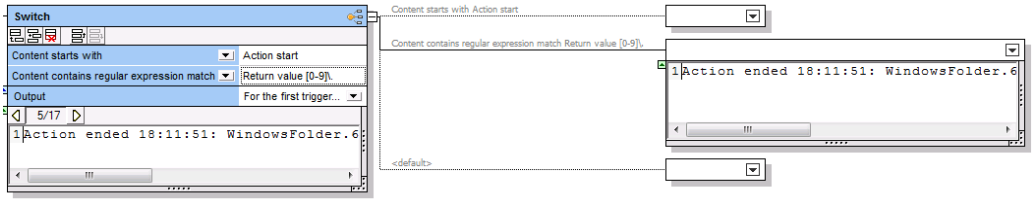
5. Click the top-right corner of the new output container and select **Switch**. Now FlexText will treat the contents of the container as a switch. As shown below, one <default> switch condition was created automatically—this condition redirects to a new container any text that does not match other conditions. At this stage, there are no other conditions defined, therefore all text is currently being redirected to the **<default>** output.



6. Click the **Append Condition** ( ) button and add the condition of type **Content starts with** with the value "Action start", as shown below. This condition redirects to a new container any text that begins with "Action start".



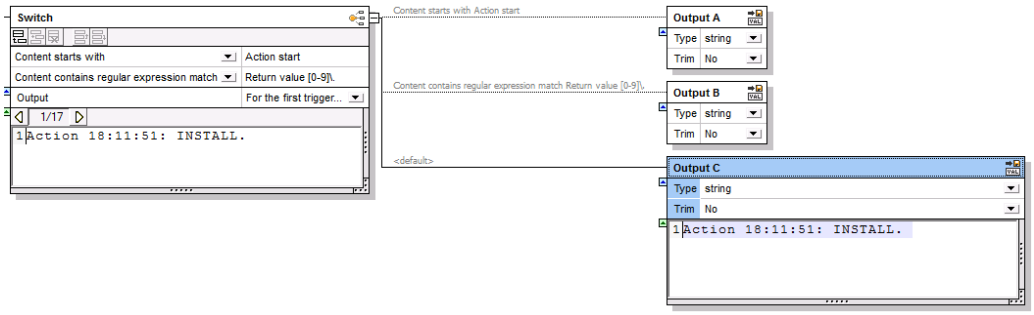
7. Click the **Append Condition** ( ) button and add the condition of type **Content contains regular expression match** with the value `Return value [0-9]\.` . If you now navigate to block 5 out of 17, you can see that this condition redirects that block to a new container, since the block contains a match for the regular expression.



You have now configured the Switch container so that it redirects text to a different output based on a conditions. There are three switch conditions and three possible outputs (one output for each condition). The remaining steps of this tutorial show how you can write each of output to a separate text file.

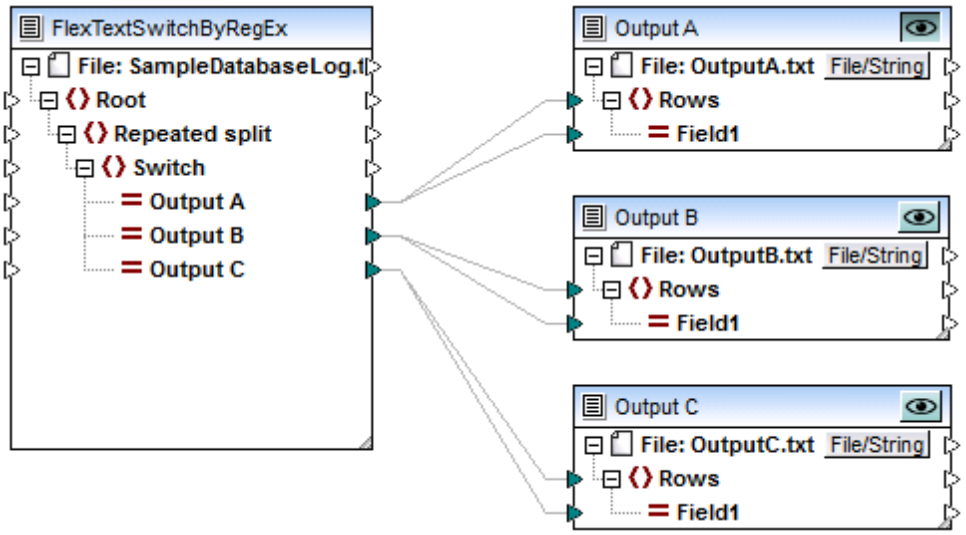
8. Click the top-right corner of each output container and choose **Store as Value**.

Additionally, double-click the title bar of each output and enter a descriptive name: **Output A**, **Output B**, and **Output C**, respectively.



9. Save and close the FlexText template.

The required FlexText template has now been created. You can see how this template works by opening the following tutorial file: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\FlexTextSwitchByRegEx.mfd**.



FlexTextSwitchByRegEx.mfd

The **FlexTextSwitchByRegEx.mfd** mapping illustrated in the screen shot reads text data from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\SampleDatabaseLog.txt** file and converts it into three separate text files: Output A, Output B, and Output C. Each of the target text files contains only the rows that satisfy one of the three conditions defined previously in the FlexText template. To view the output of a particular target component, click the **Preview Component** ( ) button in the top-right corner of the component, and then click the **Output** tab.

7.5 EDI

Altova web site: EDI Translator

EDI (Electronic Data Interchange) is a family of standards which enable electronic exchange of data between organizations or businesses.

MapForce supports translating data to or from EDI formats in any of the following transformation languages: BUILT-IN, C#, C++, Java (see also Selecting a Transformation Language). The EDI data can be mapped to or from any of the formats supported by MapForce, including databases, flat files, XML documents, and others.

MapForce supports the following flavours of EDI:

Standard	Description
ASC X12	ASC X12 is an industry standard for document interchange. MapForce supports versions: 3040, 3050, 3060, 3070, 4010, 4020, 4030, 4040, 4041, 4042, 4050, 4051, 4052, 4060, 5010, 5011, 5012, 5020, 5030, 5040, 5050, 6010, 6020, 6030, and 6040. **The default ASC X12 version in MapForce is 6040.** ASC X12 components have "virtual" nodes into which EDI parser error information/data is written depending on the settings you select in the EDI Validation Settings dialog box (see EDI component validation). An X12 997 Functional Acknowledgement can be generated from any X12 document. For more information about ASC X12, see http://www.x12.org/.
HIPAA X12	HIPAA is based on the X12 EDI 5010 standard, but has its own specialized versions which are natively supported by MapForce 2011 Release 3 or later. **The default HIPAA X12 version in MapForce is release A2 of the HIPAA implementation specs (TR3).** Previous releases are available for download on the MapForce Components page of the Altova website (http://www.altova.com/components_mapforce.html).
HL7	HL7 is an industry standard for data exchange between medical applications and is an abbreviation of "Health Level Seven". MapForce supports versions 2.2 to 2.6. **The default HL7 version in MapForce is 2.6.** A separate installer for the additional HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the MapForce Components page of the Altova website (http://www.altova.com/components_mapforce.html). The XML-based HL7 version 3.x is supported in MapForce 2016 using XML

Standard	Description
	schema components. The **MapForceExamples** project contains a sample that maps a HL7 V2.6 to a HL7 V3 XML file (HL7V260_to_HL7V3.mfd). For more information about HL7, see http://www.hl7.org/.
IATA PADIS	PADIS (Passenger and Airport Data Interchange Standards) is an industry standard for the exchange of passenger and airport data using EDI documents. MapForce supports versions: 00.1 to 08.1. **The default PADIS version in MapForce is 08.1.** For more information about IATA PADIS, see http://www.iata.org/Pages/default.aspx.
SAP IDocs	SAP IDocs (intermediate documents) documents are used to exchange business data between SAP R/3 and non-SAP applications. The documents are a form of intermediate data storage which can be exchanged between different systems. For more information about SAP IDocs, see http://help.sap.com/saphelp_nw70/helpdata/en/0b/2a6095507d11d18ee90000e8366fc2/frameset.htm.
TRADACOMS	TRADACOMS (Trading Data Communications) is a UK-specific Electronic Data Interchange standard used in the retail business. MapForce implements the base TRADACOMS specification as laid out in the "TRADACOMS Manual of Standards for Electronic Data Interchange", published in January 1993 by the Article Numbering Association (ANA) UK, now known as GS1 UK (https://www.gs1uk.org). For other TRADACOMS versions, MapForce can be customized to process new message types, data elements, and code values, by means of configuration files.
UN/EDIFACT	UN/EDIFACT is a de-facto financial industry standard for document interchange (also a UN standard). MapForce supports the messages contained in directories D93A - D15A. **The default UN/EDIFACT version in MapForce is D15A.** A separate installer for all previous versions back to D93A is available on the MapForce Components page of the Altova website (http://www.altova.com/components_mapforce.html). For more information about UN/EDIFACT, see http://www.unece.org/trade/untdid/welcome.htm.

Each MapForce release is typically supplied with the most recent version of the EDI

configuration files for each supported EDI flavour (highlighted in bold in the table above). If necessary, you can install previous versions of EDI configuration files from the MapForce Components page of the Altova website (http://www.altova.com/components_mapforce.html). After installation, the installed EDI versions become available in MapForce in addition to the default versions.

For a single EDI component that you have added to the mapping area, MapForce can process multiple message types of any single standard EDI release. For example, this enables you to read data from an interchange file which includes more than one message type, or to generate as mapping output an interchange file which contains multiple message types. You can also process data from multiple instance files containing different message types (see also Processing Multiple Input or Output Files Dynamically). Note that operations such as the ones described above are only meaningful when they are supported by the underlying specification.

If your organization is using EDI specifications that are customizations of those already supported by MapForce, it is possible to adapt MapForce to your custom EDI specification by means of configuration files. For example, you can add custom messages, segments, data elements, code lists, and so on (or change the existing ones), thus creating a custom EDI collection similar to any of the EDI-related directories that are already supported by MapForce. (The EDI directories supported by MapForce are available in the **Program Files\MapForce2016\MapForceEDI** directory and will include the latest versions of the corresponding specification, plus any additional versions that you may have downloaded and installed from http://www.altova.com/components_mapforce.html). Note that customization is only possible when the custom specification that you want to accommodate in MapForce is based on EDIFACT (ISO 9735), X12, HL7, or TRADACOMS.

7.5.1 EDI Terminology

The following short list describes the main UN/EDIFACT terms and their counterparts in ANSI X12, the US related standard.

Messages (ANSI X12 - Transactions)
A message (or transaction set) is a specific sequence of segments that represents a business document. MapForce supports one or multiple different message types in a single EDI component.

Segment
A single "record" contained in a message.

Segments are identified by a two or three character ID at the beginning of the segment. A group of related elements comprise a segment tag (or segment ID - ANSI X12). Segments of a transaction can be defined as mandatory or conditional (optional).

Element
An individual data field within a segment. An element can be thought of as a field, i.e. it contains one type of data, a name, or an address, for example. Elements can be further subdivided into composite elements, consisting of component elements or subelements.

Separators
Elements are delimited by "separator characters". In UN/EDIFACT these are either default characters, or defined in an optional UNA control segment.

Default characters

colon	:	component element separator
plus	+	data element separator
apostrophe	'	segment terminator

HL7 allows for an additional sub-subfield separator where the default is **&**. This separator is called the **Subcomponent Separator** in the Component Settings dialog box.

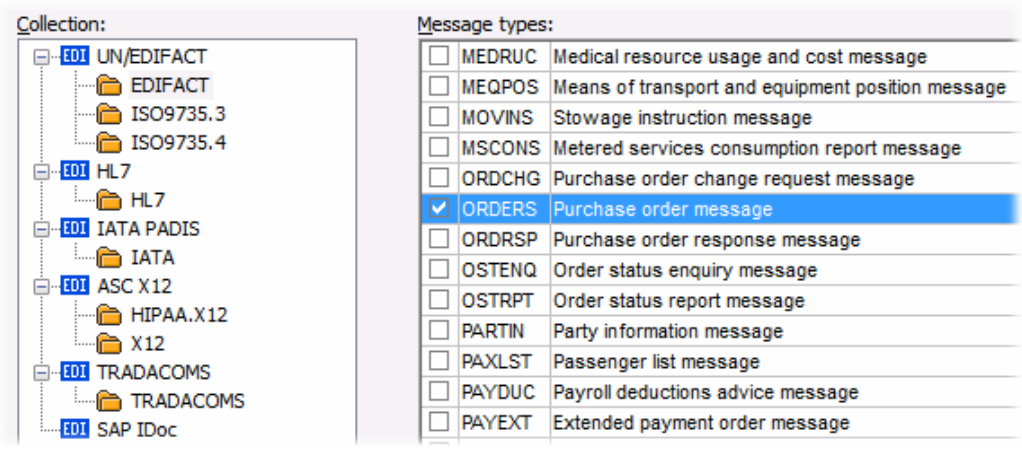
Interchange
These are a collection of Group envelopes and/or messages for the same recipient.

7.5.2 UN/EDIFACT to XML Schema mapping

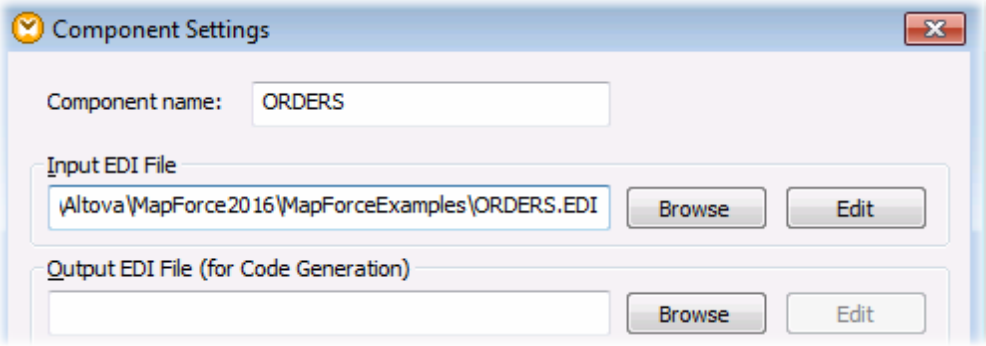
This example shows how to map data from UN/EDIFACT messages to an XML schema, in order to produce an XML instance file for further processing. The mapping created in this example is available in the **<Documents>\Altova\MapForce2016\MapForceExamples** directory as **EDI_Order.mfd**.

Step 1: Add the UN/EDIFACT component to the mapping

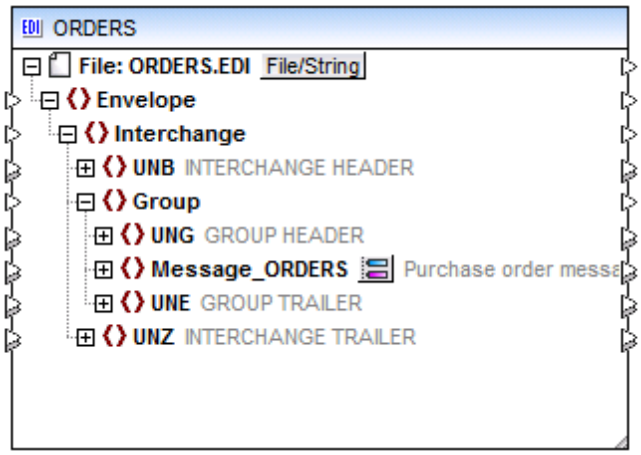
1. Create a new mapping and select one of the following transformation languages: Java, C#, C++, or BUILT-IN. In this example, Java is selected as transformation language.
2. On the **Insert** menu, click **EDI**.
3. On the EDI collections dialog box, select the EDIFACT collection, then select the **ORDERS** message type, and click **OK**.



4. When prompted to supply a sample EDI file, click **Browse** and open the **ORDERS.EDI** file from the **<Documents>\Altova\MapForce2016\MapForceExamples** directory. After you open the file, the Component Settings dialog box opens. This enables you to review the settings of the EDI component before adding it to the mapping. You can change these settings at any time later if required (see EDI Component Settings). Notice that the **ORDERS.EDI** file appears as Input EDI file.

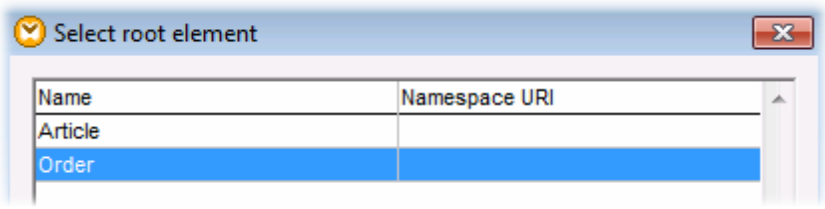


5. Click **OK**. The EDI component is now displayed in the mapping area. Double-click the **Message_ORDERS** node to view its children items. To resize the component, click and drag the lower-right corner of the component window.

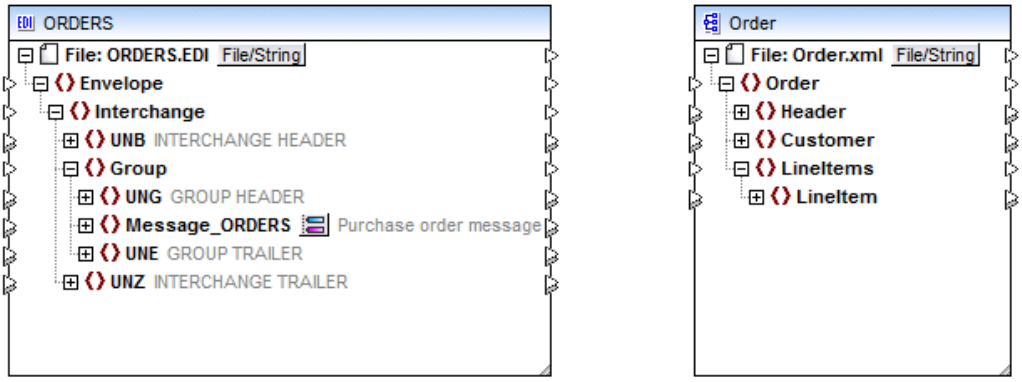


Step 2: Add the target schema component to the mapping

1. On the **Insert** menu, click the **XML Schema/File**, and open the **Order.xsd** file from the **<Documents>\Altova\MapForce2016\MapForceExamples** directory.
2. When prompted to supply a sample XML file, click **Skip** and select **Order** as the root of the target document.



At this point, both the source EDI component and the target XML schema are on the mapping area, so we are ready to start drawing the mapping connections.



Step 3: Map the EDI items

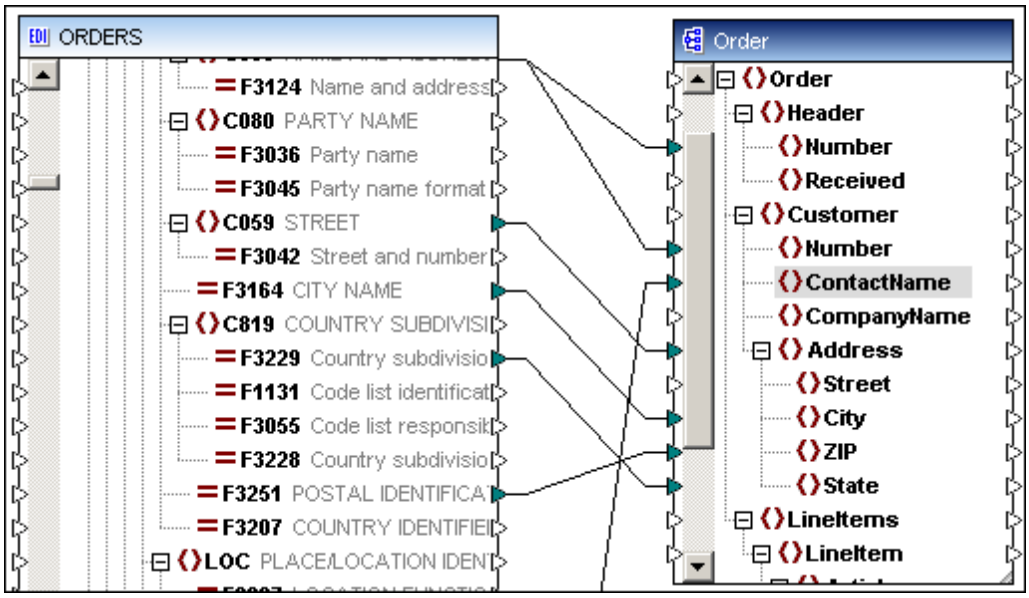
The EDI component displays the structure of a message based on the collection (ORDERS) we selected. Typically, not all of the nodes will actually contain data, so you must be sufficiently familiar with the EDI documents being worked on, to locate the relevant nodes.

In MapForce, you map a source item and a target item by drawing a connection between them. For step-by-step instructions on how to create mapping connections in MapForce, see Working with Connections.

In this example, the following nodes (starting from the **Group/Message_ORDERS** node) will be mapped (connected) first:

Source	Target
BGM/C106/F1004	Order/Header/Number
SG2/NAD/C082/F3055	Order/Customer/Number
SG2/NAD/C080/F3036	Order/Customer/CompanyName
SG2/NAD/C059/F3042	Order/Customer/Address/Street
SG2/NAD/F3164	Order/Customer/Address/City
SG2/NAD/C819/F3229	Order/Customer/Address/State
SG2/NAD/F3251	Order/Customer/Address/ZIP
SG2/SG5/CTA/C056/F3412	Order/Customer/ContactName

At this stage, the mapping should look similar to the image below:



Continue the mapping process and map:

Source	Target
SG28	Order/LineItems
SG28/LIN/C212/F7140	Order/LineItems/LineItem/Article/Number
SG28/IMD/C273/F7008	Order/LineItems/LineItem/Article/Name
SG28/QTY/C186/F6060	Order/LineItems/LineItem/Article/Amount
SG28/SG32/PRI/C509/F5118	Order/LineItems/LineItem/Article/SinglePrice

Step 4: Format the date

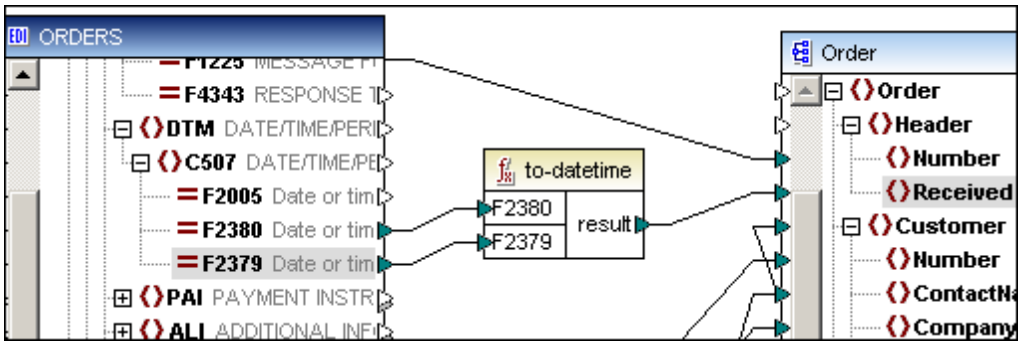
Drag the **to-datetime** function from the **edifact** library into the Mapping area. For instructions on how to work with functions in MapForce, see Working with Functions.

By supplying as arguments to this function the F2380 and F2379 components of the DTM/C507 element, we can create an appropriately formatted **Received** datetime.

We therefore map the following fields:

Source	Target
DTM/C507/F2380	The F2380 input of the **to-datetime** function
DTM/C507/F2379	The F2379 input of the **to-datetime** function

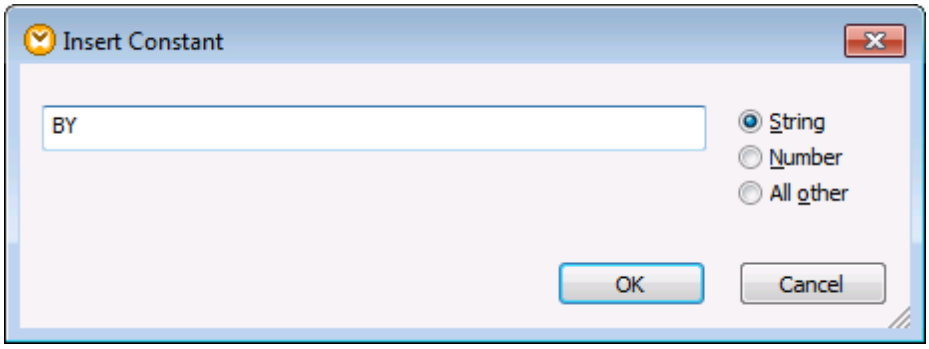
Source	Target
The **result** of the `to-datetime` function	`Order/Header/Received`



Step 5: Filter the buyer purchase orders

At this point we want to filter the "Buyer" purchase orders. These can be identified by the party function code qualifier of the NAD (Name and address) segment. In this case, the value 'BY' indicates a "Buyer" (Party to whom merchandise and/or service is sold).

1. Drag the `equal` function from the **core** library into the Mapping area.
2. Add a filter to the mapping (On the **Insert** menu, click **Filter: Nodes/Rows**).
3. Add a constant to the mapping (On the **Insert** menu, click **Constant**). Assign to the constant the value "BY" by entering "BY" into the text field:

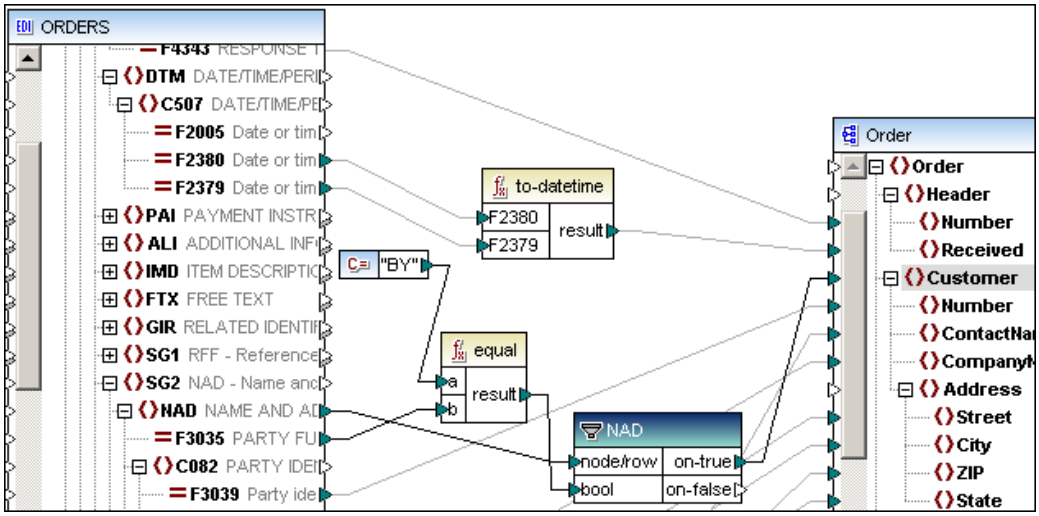


Map the following items:

Source	Target
`SG2/NAD/F3035`	The **b** input of the `equal` function
The constant "BY"	The **a** input of the `equal` function
The result of the `equal` function	The **bool** input of the **filter** component

Source	Target
SG2/NAD	The **node/row** input of the filter component
The result of the **filter** component	Order/Customer in the schema

The aim here is to only map data if the NAD node refers to a 'Buyer', as identified by the party function code qualifier 'BY'.



Step 6: Calculate pricing and tax

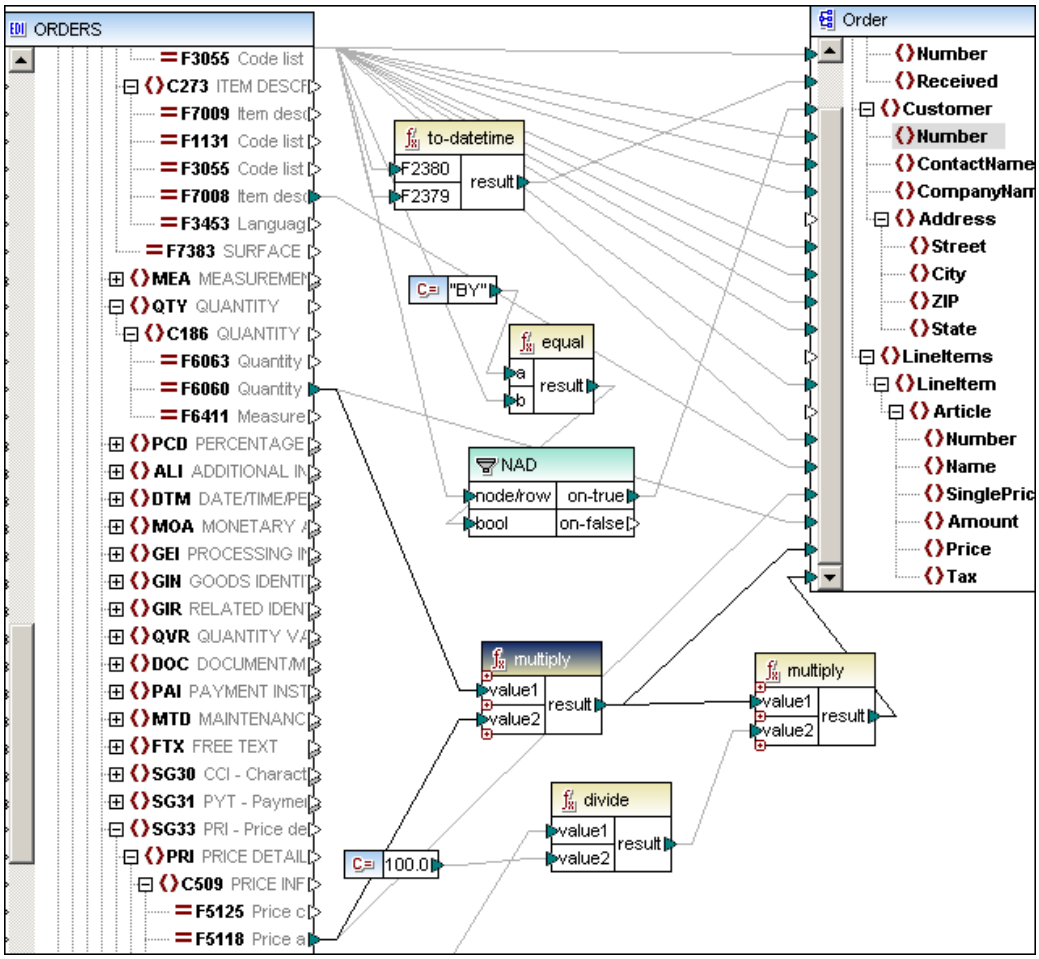
The final step in this task is to calculate the pricing and tax costs.

1. From the core library, drag two **multiply** and one **divide** function into the Mapping area.
2. Insert a Constant component (**Insert | Constant**). Make sure "Number" is selected as type, and enter 100.0 into the text field.
3. Map the following items:

Source	Target
SG28/QTY/C186/F6060	**value1** of the first **multiply** function
SG28/SG32/PRI/C509/F5118	**value2** of the first **multiply** function
The result of the first **multiply** function	Order/LineItems/LineItem/Article/Price
SG28/SG38/TAX/C243/F5278	**value1** of the **divide** function
The constant "100.0"	**value2** of the **divide** function
The result of the first **multiply** function	**value1** of the second **multiply** function
The result of **divide** function	**value2** of the second **multiply** function

Source	Target
The result of the second **multiply** function	Order/LineItems/LineItem/Article/Tax

Your mapping should now look like this:



Clicking the output tab performs an "on the fly" transformation and presents you with the XML document result:

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Header>
    <Number>ABC123456XYZ</Number>
    <Received>2004-04-30T17:42:00-09:00</Received>
  </Header>
  <Customer>
    <Number>123</Number>
    <ContactName>Michelle Butler</ContactName>
    <CompanyName>Nanonull, Inc.</CompanyName>
    <Address>
      <Street>119 Oakstreet Suite 4876</Street>
      <City>Vereno</City>
      <ZIP>29213</ZIP>
      <State>CA</State>
    </Address>
  </Customer>
  <LineItems>
    <LineItem>
      <Article>
        <Number>42</Number>
        <Name>Pizza Pepperoni</Name>
        <SinglePrice>7.2</SinglePrice>
        <Amount>1</Amount>
        <Price>7.2</Price>
        <Tax>0.648</Tax>
      </Article>
    </LineItem>
```

7.5.3 EDI component validation

MapForce is capable of validating all supported EDI **source** and **target** documents when the mapping is executed, e.g. by clicking the Output button. The validation process is, however, not a full EDI syntax or semantic validation. The **X12_To_XML_Order.mfd** project file available in the ...\MapForceExamples folder shows an example of this type of mapping.

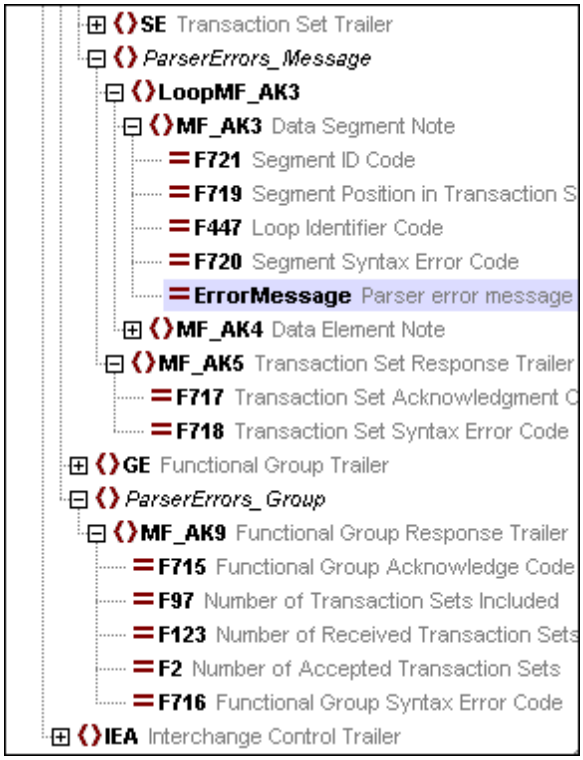
The validation parameters can be user-defined to suit your specific needs. You can choose to ignore errors, stop the validation process on a specific error, or accept/reject and report them.

When running generated program code, parsing errors will throw an exception that stops the mapping, and validation errors will display a message at the standard output.

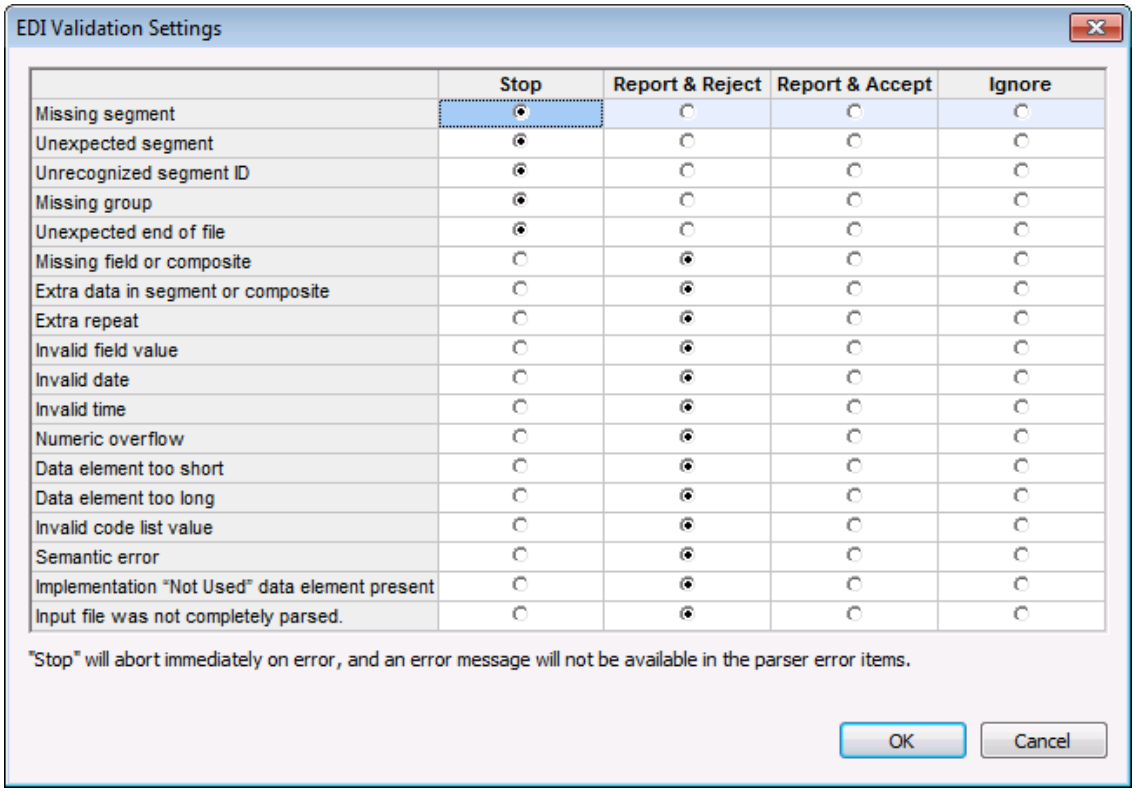
X12 Acknowledgment

For source EDI components, the validation results (report) are placed in "virtual" items at the base of the EDI component **(**under ***ParserError_Message*** and ***ParserErrors_Group).*** These can be used when generating an X12 997 or 999 Acknowledgement.

To create an X12 Acknowledgment file that reports the status of the interchange, please see X12 997 - Functional Acknowledgment or X12 999 - Implementation Acknowledgment.



To customize the validation settings, double-click an EDI component and click the **Validation** button in the Component Settings dialog box. The default settings are shown below.



EDI Validation Settings dialog box

Stop is used to catch fatal errors and stops the execution of the mapping without generating a report message.

Report & Reject and **Report & Accept** provide information in the Parser_Errors_Message and Parser_Errors_Group items of the EDI component which can be mapped further.

Depending on the setting **Reject** or **Accept,** the Functional Group Acknowledge Code F715 and the Transaction Set Acknowledgment Code F717, will contain either:

the value 'R' for *'Rejected'* or

the value 'E' for '*Accepted, but errors were noted*'. The errors also appear in the Messages window.

Ignore ignores the specific error. No information is provided within the Parser_Errors_Message and Parser_Errors_Group items.

The EDI validation settings are as follows:

Name	Description	Error might occur for a
Missing segment	A mandatory segment is missing or the occurrence is less than a specified minimum.	source component

Name	Description	Error might occur for a
Unexpected segment	A segment is defined in the specification but not in this message.	source component
Unrecognized segment ID	A segment was found which is not defined in the specification.	source component
Missing group	A mandatory group is missing or the occurrence is less than the specified minimum.	source component
Unexpected end of file	The instance can not be parsed since some data is missing.	source component
Missing field or composite	A mandatory field or composite is missing, or the occurrence is less than the specified minimum.	source or target component
Extra data in segment or composite	The input instance contains additional data that is not expected by the syntax description.	source component
Extra repeat	The actual number of fields within a segment/composite exceeds the specified maximum number.	source or target component
Invalid field value	A numeric field contains an invalid character.	source component
Invalid date	A date field contains either an invalid character or the values for the month or the day are invalid.	source component
Invalid time	A time field contains either an invalid character or the value for the hours or the minutes are invalid.	source component
Numeric overflow	A numeric values overflows its defined domain. This error is only supported within the generated code.	source component
Data element too short	The length of a data element is less than the specified minimum value.	source or target component
Data element too long	The length of a data element is greater than the specified maximum limit.	source or target component
Invalid code list value	A code list value does not match the set of specified values.	source or target component
Semantic error	A semantic error has occurred.	source or target component
Implementation "Not Used" data element	An element exists in the input file which is not allowed by the HIPAA configuration file	source component

Name	Description	Error might occur for a
present	(maxOccurs="0").	
Input file was not completely parsed	The input file was not completely parsed.	source component

Having created your mapping to the target component click the Output button to see the mapping result.

If there are any errors in the source or target EDI document, when executing the mapping, the errors will appear in the Messages window. An error in the source EDI does not automatically mean that the mapping process will fail, output could still be generated in the Output tab.

7.5.4 X12 997 - Functional Acknowledgment

The X12 997 Functional Acknowledgment reports the status of the EDI interchange. All errors encountered during processing of the document are reported in it. MapForce can automatically generate a X12 997 component in the main mapping area, and automatically create the necessary connectors.

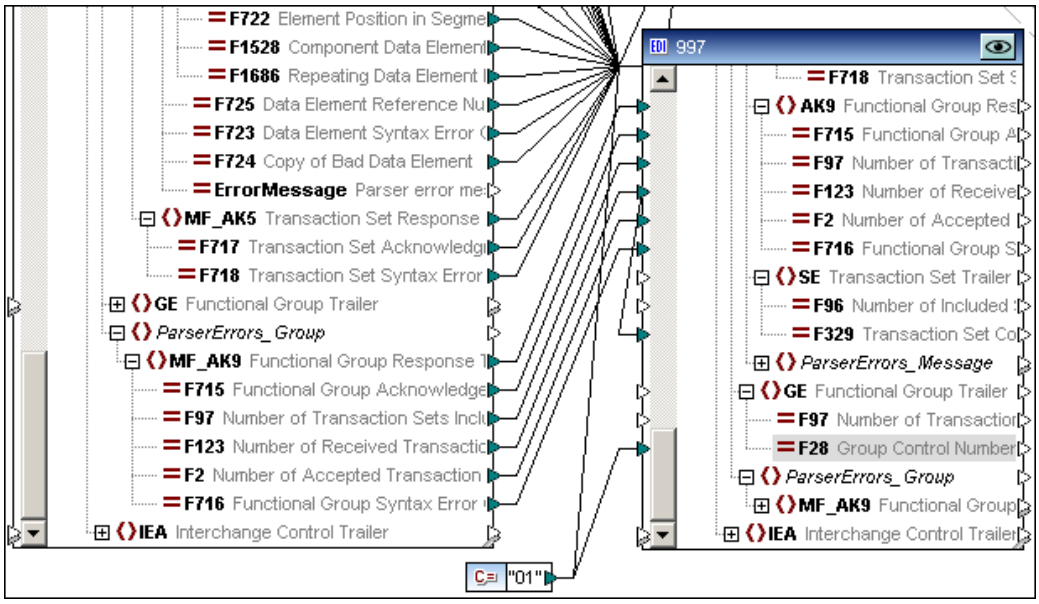
The **X12_To_XML_Order.mfd** mapping design file available in the **<Documents>\Altova \MapForce2016\MapForceExamples** folder shows an example of this type of mapping.

For more information about the structure of X12 997, see https://msdn.microsoft.com/en-us/library/bb226316(BTS.20).aspx.

Note that you can also generate an X12 999 - Implementation Acknowledgment instead.

To generate the EDI 997 Functional Acknowledgment:

1. Right-click the source EDI component and select "**Create mapping to EDI X12 997**".



This creates an EDI 997 component and automatically connects the items needed to generate the X12 997 acknowledgment.

2. Click the Preview button of the EDI 997 component, then click the Output tab, to see the generated acknowledgment file.

```
1   ISA+00+       +00+       +ZZ+ReceiverID    +ZZ+SenderID      +100128+1113+!+00505+000000000+1+P+:'
2   GS+01+ReceiverID+SenderID+20100128+11130544+1+X+05012'
3   ST+997+0001'
4   AK1+PO+1+05012'
5   AK2+850+12345'
6   AK5+A'
7   AK9+A+1+1+1'
8   SE+6+0001'
9   GE+1+1'
10  IEA+1+000000000'
11
```

To save the 997 ACK file:

- Either enter a file name in the "Output EDI File" field of the Component Settings dialog box, if you are generating code, or
- Click the "Save generated output" icon  while viewing the result in the Output window, and enter path and file name.

7.5.5 X12 999 - Implementation Acknowledgment

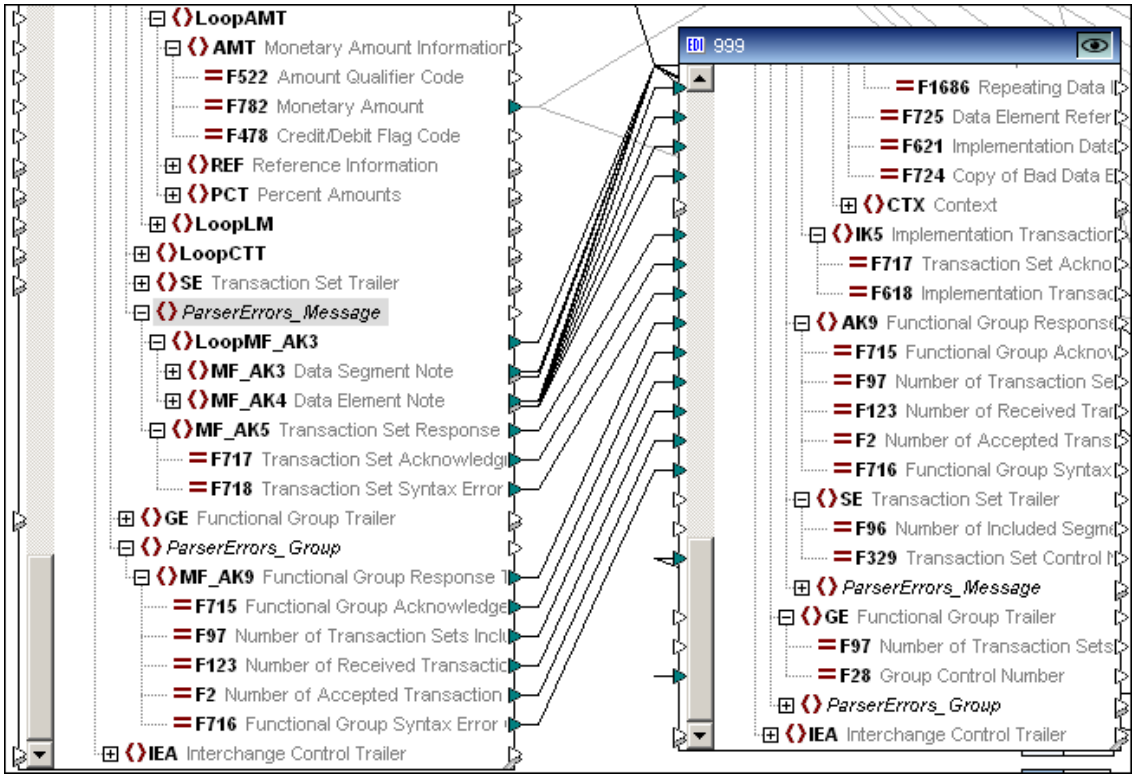
The X12 999 Implementation Acknowledgment Transaction Set reports X12 implementation guide non-compliance, or application errors.

As it is a super-set of the 997 Functional Acknowledgement, 999 can be used instead of 997 to accept, or reject, transaction sets based on either X12 or HIPAA Implementation Guide syntax requirements.

All errors encountered during processing of the document are reported in it. E.g. "Required Segment Missing", "Required Data Element Missing", "Code Value Not Used in Implementation", etc. MapForce can automatically generate a X12 999 component in the main mapping area, and automatically create the necessary connectors.

To generate the EDI 999 Implementation Acknowledgment:

1. Right click the source EDI component and select "**Create mapping to EDI X12 999**".



This creates an EDI 999 component and automatically connects the items needed to generate the X12 999 implementation acknowledgment.

2. Click the Preview button of the EDI 999 component, then click the Output tab, to see the generated acknowledgment file.

```
1  ISA+00+          +00+          +ZZ+ReceiverID    +ZZ+SenderID      +110502+1533+!+00505+000000000+1+P+:'
2  GS+FA+ReceiverID+SenderID+20110502+15333751+1+X+006020'
3  ST+999+0001'
4  AK1+PO+1+006020'
5  AK2+850+12345'
6  IK5+A'
7  AK9+A+1+1+1'
8  SE+6+0001'
9  GE+1+1'
10 IEA+1+000000000'
11
```

To save the 999 ACK file:

- Either enter a file name in the "Output EDI File" field of the Component Settings dialog box, if you are generating code, or
- Click the "Save generated output" icon  while viewing the result in the Output window, and enter path and file name.

7.5.6 TA1 Segment

The TA1 Segment is an optional segment and is used to acknowledge the reception of the interchange and the syntactical correctness of the envelope segments within it. The segment can

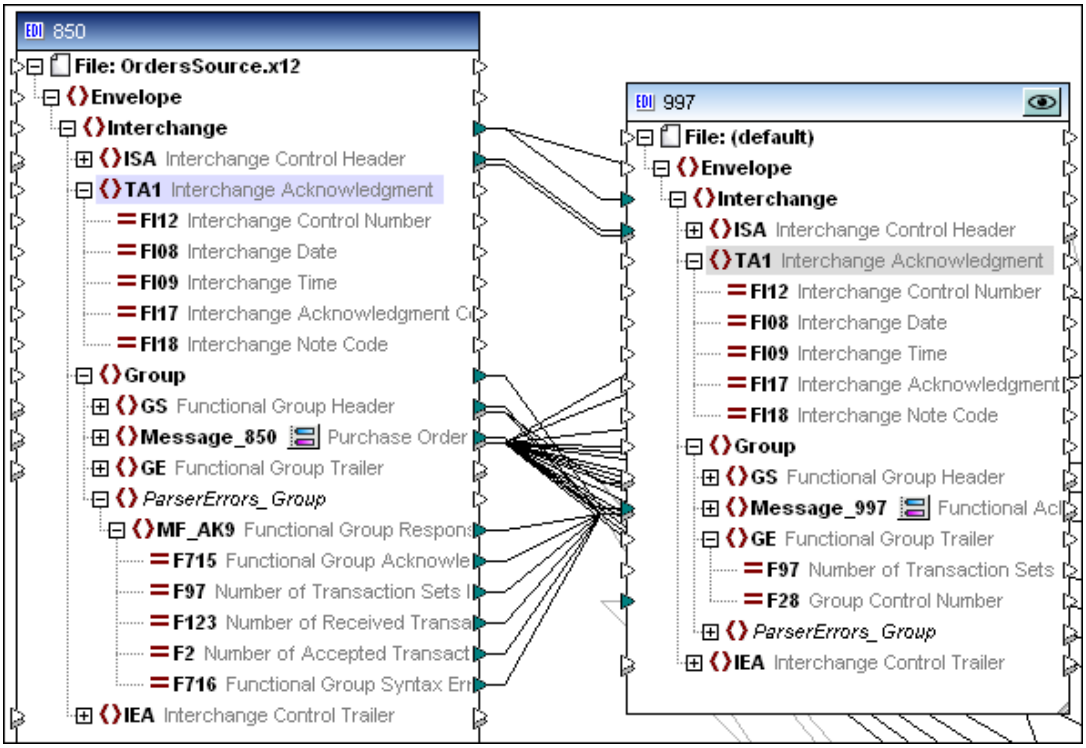
be added to EDI X12 and EDI HIPAA components.

To include the TA1 segment in an EDI component:

1. Browse to the folder containing the Envelope.Config file, e.g. c:\Program Files\Altova \MapForce2016\MapForceEDI\X12\ and edit the file.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nol
3    <Meta>
4      <Release>6020</Release>
5      <Agency>X12</Agency>
6    </Meta>
7    <Format standard="X12"/>
8    <Include href="X12.Segment"/>
9    <Include href="X12.Codelist"/>
10   <Include collection="EDI.Collection"/>
11   <Group name="Envelope">
12     <Group name="Interchange" maxOccurs="unbounded">
13       <Segment ref="ISA" minOccurs="0"/>
14       <Segment ref="TA1" minOccurs="0" maxOccurs="unbounded"/>
15       <Group name="Group" maxOccurs="unbounded">
```

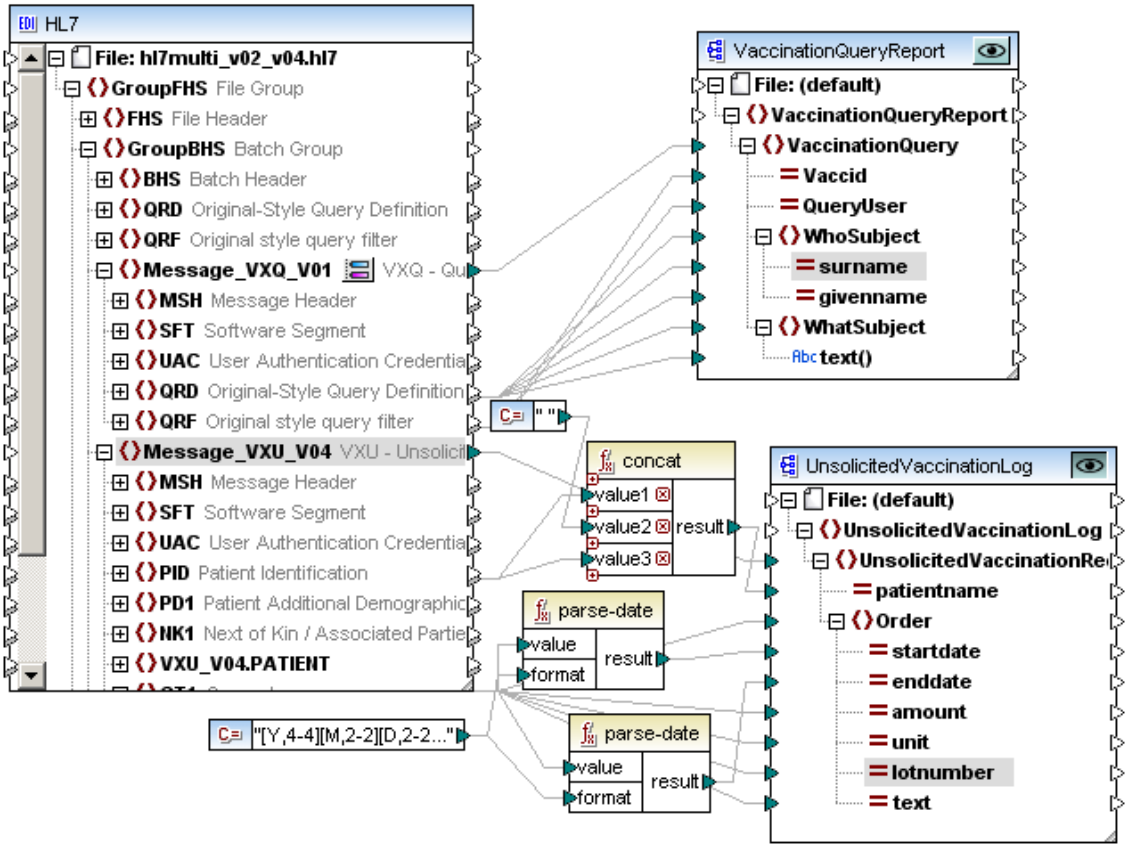
2. Add a reference to the TA1 segment by adding between the ISA and Group segments, then save the file.
3. Opening the X12_To_XML_Order.mfd file available in the MapForceExamples folder shows that the segment has been added to the 850 Purchase Order component as well as the 997 Functional Acknowledgment component.
   You can now choose which TA1 items you want to map to the target component.



7.5.7 Multiple EDI messages per component

MapForce is now able to process multiple message types per EDI component (of a single standard release). This allows you to process:

- multiple files (using wildcards) containing different messages as input instances
- different message types within the same file, i.e. Interchange



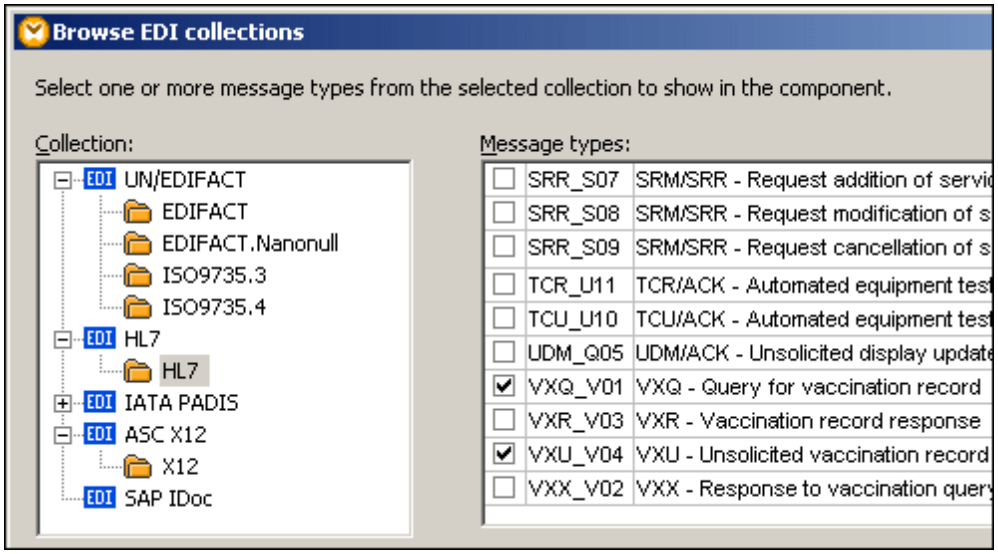
Please note:

the Select EDI message icon , of the EDI component, opens the "Browse EDI collections" dialog box, allowing you to select multiple messages per interchange (per EDI component).

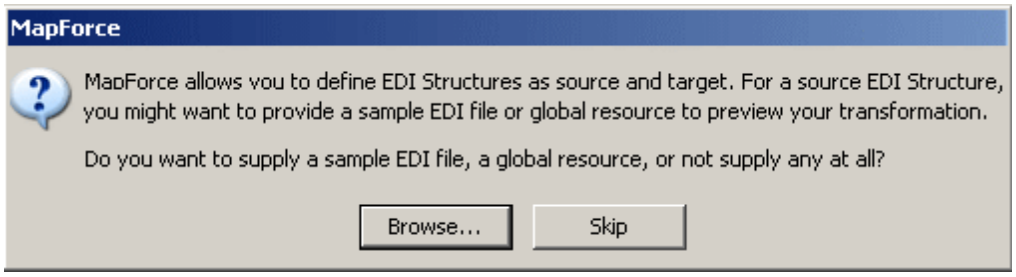
The example shown above is available as **HL7_MultiMessageTypes.mfd** in the **<Documents>\Altova\MapForce2016\MapForceExamples** folder. The section that follows uses the files available in that folder.

To create an EDI component which includes several message types:

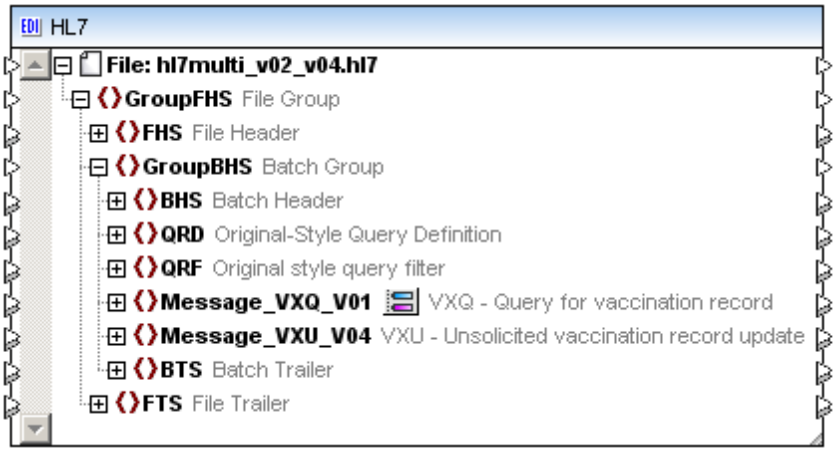
1. Create a new mapping and ensure that either Java, C#, C++, or BUILTIN is active. It is not possible to generate XSLT 1.0 / 2.0 or XQuery code when mapping from EDI files.
2. Select the menu option **Insert | EDI**, or click the EDI icon, and click the HL7 folder under the HL7 item.



3. Click the message types that should appear in the component, e.g. VXU_V01 and VXU_V04, then click OK.
   You are now prompted to supply a sample EDI file.



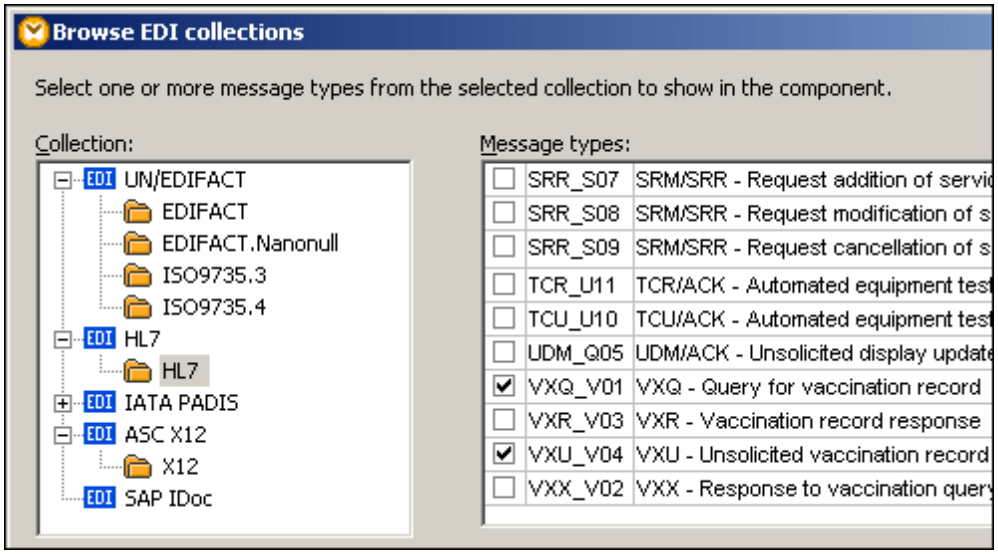
4. Click the Browse button and select the EDI file that should supply the mapping data, e.g. hl7multi_v02_v04.hl7, and click the Open button.
   This opens the Component Settings dialog box, in which you can change the EDI, or encoding settings.
5. Click OK to use the default settings and insert the EDI component.



The two messages selected in the Browse EDI Collections dialog box, are now shown as separate message items below the Batch Group item GroupBHS, eg. Message_VXQ_V01 and Message_VXQ_V04

To change the number of messages in a component:

1. Click the "Select EDI message" icon next to the first message in the component. This opens the "Browse EDI collections" dialog box.
2. Check (or uncheck) the message checkboxes to change the number of messages, then click OK to confirm.



7.5.8 UN/EDIFACT and ANSI X12 as target components

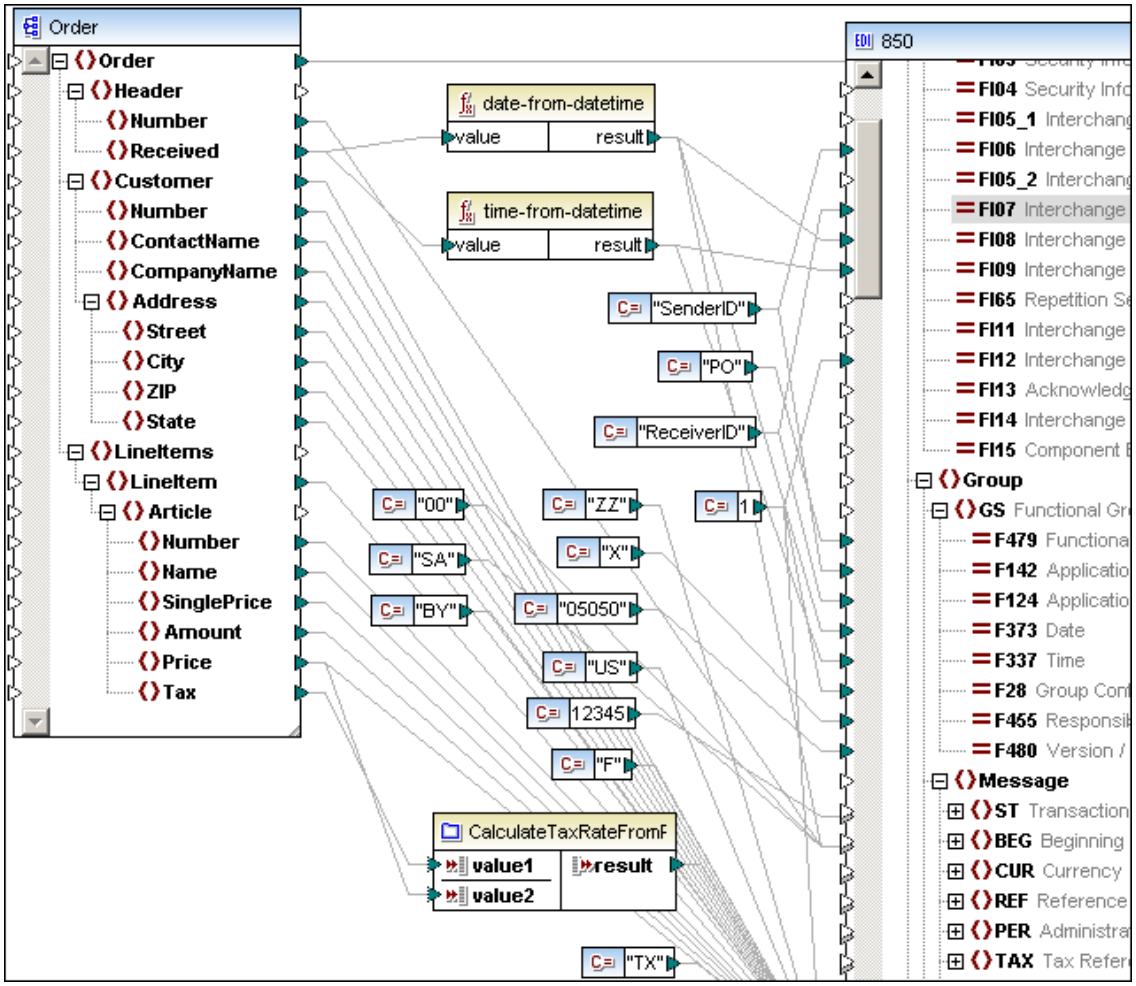
The **XML_To_X12_Order.mfd** file available in the **...\MapForceExamples** folder maps an XML file to an ANSI X12 EDI file. The resulting EDI output file can be validated against the X12 specification by clicking the "Validate output file" icon.

The following sections describe:

- the settings common to both EDI formats
- the validation rules and automatic data completion settings used for each of the specific EDI formats.

Please note:

If it is important to retain the element/item sequence present in the source component, please make sure you create connectors using the "Source Driven (mixed content)" option, see the section on Source driven and mixed content mapping for more information.



Automatic data completion
Automatic data completion does not generally change existing data. It does, however, create (specific) mandatory nodes and inserts data where necessary, after the mapping process, to produce a valid document.

Please note that fields not listed in the "Automatic data completion" sections that follow, are NOT inserted, or created. The correct values cannot be ascertained automatically.

Generic Settings:
Several settings can be defined that are applicable to all EDI documents:

- Auto-complete missing fields:
  should auto-completion be enabled or not. Default: **true**
- Begin new line after each segment:
  should a new line be appended to each segment for improved readability. The EDI standard ignores these lines if present in a message. Default: **true**
- Data element separator: see EDIFACT/X12 specification.
  Default: **+**
- Composite Separator: see EDIFACT/X12 specification.
  Default **:**

- Segment Terminator: see EDIFACT/X12 specification.
  Default **'**
- Decimal Notation: see EDIFACT/X12 specification.
  Default **.**
- Release Character: see EDIFACT/X12 specification.
  Default **?**

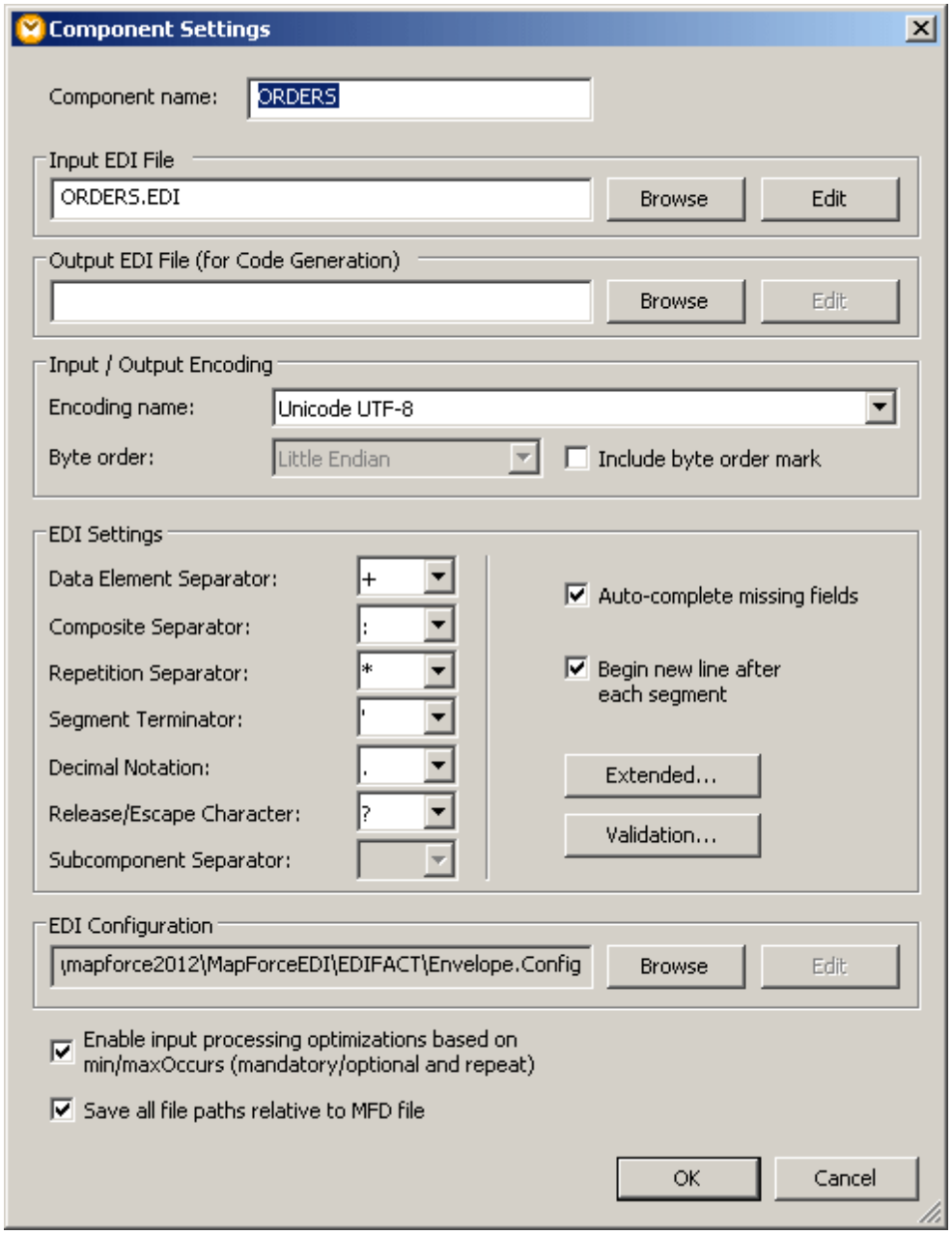
- HL7 allows for an additional sub-subfield separator where the default is **&**. This separator is called the **Subcomponent Separator** in the Component Settings dialog box.

  Right click an EDI component and select **Properties** to open the EDI component settings dialog box. The UN/EDIFACT settings are used as defaults for both EDI components.

Separator precedence when reading / writing EDI files

The EDI separators entered in this dialog box always take effect when **writing** EDI files. When **reading** in EDI files, the separators only take effect if the input file does not define/contain its own separators, e.g. EDIFACT files without the UNA "service string advice" segment.

If an EDI input component/file contains separator definitions, e.g. an X12 file with an ISA segment, then the existing separators override any separators defined in the Component Settings dialog box for that file.



Clicking the Extended... button, opens the respective extended EDI settings dialog box.

Defining non-printable characters:
You can use non-printable characters as separators by typing "x" followed by the hexadecimal ASCII character code into one of the combo boxes, e.g. "x1e" for the RS control character (ASCII record separator, decimal code 30).

Mapping date and time datatypes
Prior to MapForce 2006R3, date and time were of type "string" in EDIFACT and X12 components. From this release on, date and time can be mapped directly to/from **xsd:time**, or **xsd:date** (also from SQL date/time).

User defined functions prior to the 2006R3 version that convert xsd:date, or xsd:time into an EDI

format, generate an error message and cannot be used as they are. Please use the schema datatype **xsd:dateTime**, which can be mapped using the built-in functions **date-from-datetime** or **time-from-datetime,** in the datetime library.

If you need to map dates within user-defined functions, please make sure that they adhere to the ISO 8601 date/time format i.e. YYYY-MM-DD.

Validation

EDI Validation is supported in the Output tab (BUILTIN), as well as in the generated code. Select the menu item "**Output | Validate output XML file**" to validate the EDI Output file in the Output tab.

When running generated program code, parsing errors will throw an exception that stops the mapping, and validation errors will display a message at the standard output.

7.5.8.1 UN/EDIFACT target - validation

The following items are checked when a UN/EDIFACT document is validated:

- Whether a UNB and a UNZ segment exist.
- Whether UNB/S004 contains a valid date/time specification.
- Whether UNB/0020 and UNZ/0020 contain the same value.
- Whether UNZ/0036 contains the correct number; which is defined as the number of functional groups, if present, or the number of messages. If there are functional groups, this should be the number of functional groups, otherwise it should be the number of messages contained in the interchange.

Each **functional group** is checked:

- Whether it contains a matching UNG and UNE pair.
- Whether UNG/S004 contains a valid date/time specification.
- Whether UNE/0060 contains the correct number of messages contained in the functional group.

Each **message** is checked:

- Whether it contains a matching UNH and UNT pair.
- Whether UNH/S009/0052 contains the same value as UNG/S008/0052 of the enclosing functional group.
- Whether UNH/0062 and UNT/0062 contain the same value.
- Whether UNH/S009/0065 contains the correct message type specifier.
- Whether UNT/0074 contains the correct number of segments contained in the message.

Automatic data completion for EDIFACT makes sure:

- a UNB and a UNZ segment exist
- That if either UNG or UNE exist, that the other ID also exists
- That a UNH and a UNT segment exist

- That UNB/S001 exists. If it does not contain data, the syntax level and syntax version number from the user-defined settings are used. **See Extended | Syntax version number**.
- That UNB/S002 and UNB/S003 exist.
- That UNB/S004 exists. If it does not contain data, the current date/time in EDI format is inserted.
- That UNZ/0036 exists. If it does not contain data, the number of functional groups or messages is calculated and inserted.
- That UNZ/0020 exists. If it does not contain data, the value from UNB/0020 is copied.

Please note:

Any fields not mentioned in this section (Automatic data completion) are NOT inserted, or created. The correct values cannot be ascertained automatically.

Given a (target) parent element A (in the target EDI component) with child items x, y, and z - where y is mandatory, parent element A will only be created in the output file if the mandatory child element "y" in the target component has been mapped!

Functional group checking makes sure:

- That UNG/0038 exists. If it does not contain data, the name of the message is inserted.
- That UNG/S006 and UNG/S007 exist.
- That UNG/S004 exists. If it does not contain data, the current date/time in EDI format is inserted.
- That UNG/0051 exists. If it does not contain data, the first two characters of the controlling agency from the user-defined settings are inserted. See **Settings | Controlling agency**.
- That UNE/0060 exists. If it does not contain data, the number of messages in the group is calculated and inserted.
- That UNE/0048 exists. If it does not contain a value, the value from UNG/0048 is copied.

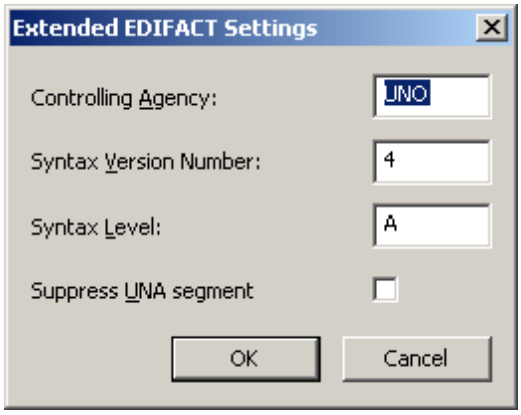
Message checking makes sure:

- That UNH/S009/0065 exists. If it does not contain data, the name of the message is inserted.
- That UNH/S009/0052 and UNH/S009/0054 exist.
- That UNH/S009/0051 exists. If it does not contain data, the first two characters of the controlling agency from the user-defined settings are inserted. See **Settings | Controlling agency**.
- That UNT/0074 exists. If it does not contain data, the number of segments in the message is calculated and inserted.
- That UNT/0062 exists. If it does not contain data, the value from UNH/0062 is copied.
- That UNH/0062 exists. If it does not contain data, the value from UNT/0062 is copied. (If only the trailer segment number is mapped, then the corresponding field in the header segment is supplied with the same value)

Settings

Clicking the Extended... button in the component settings dialog box opens the extended settings

dialog box. Please see the UN/EDIFACT specification for more details.



- Controlling agency: Default **UNO**
- Syntax version number: Default **4**
- Syntax level: Default **A**
- Suppress UNA segment: Default **unchecked.**

7.5.8.2 ANSI X12 target - validation

The following items are checked when a ANSI X12 document is validated:

- Whether an ISA and an IEA segment exist
- Whether ISA/I01 contains a legal authorization information qualifier.
- Whether ISA/I03 contains a legal security information qualifier.
- Whether the two ISA/I05 segments contain legal interchange ID qualifiers.
- Whether ISA/I08 contains a well-formed date value.
- Whether ISA/I09 contains a well-formed time value.
- Whether ISA/I13 contains a legal boolean value.
- Whether ISA/I14 contains a legal interchange usage indicator.
- Whether ISA/I12 and IEA/I12 contain the same value.
- Whether IEA/I16 contains the correct number of function groups in the interchange.

Each **function group** is checked:

- If there is a matching GS and GE pair.
- Whether GS/373 contains a well-formed date value.
- Whether GS/337 contains a well-formed time value.
- Whether GS/28 and GE/28 contain the same value.
- Whether GE/97 contains the correct number of messages in the function group.

Each **message** is checked:

- If there is a matching ST and SE pair.
- Whether ST/143 contains the correct message identifier.

- Whether ST/329 and SE/329 contain the same value.
- Whether SE/96 contains the correct number of segments in the message.

Automatic data completion for EDI/X12 makes sure:

- That an ISA and IEA pair exist on the interchange level.
- That if either GS or GE exist, the other ID also exists.
- That there is at least one ST/SE pair on the message level.
- That ISA/I01 and ISA/I03 exist. If they do not contain data, 00 is inserted.
- That ISA/I02 and ISA/I04 exist. If they do not contain data, ten blanks are inserted.
- That both ISA/I05 segments exist. If they do not contain data, ZZ is inserted.
- That ISA/I08 exists. If it does not contain data, the current date in EDI format is inserted.
- That ISA/I09 exists. If it does not contain data, the current time in EDI format is inserted.
- That ISA/I65 exists. If it does not contain data, the repetition separator is inserted.
- That ISA/I11 exists. If it does not contain data, the interchange control version number from the user-defined settings is inserted. See **Settings | Interchange control version-number**.
- That ISA/I12 exists.
- That ISA/I13 exists. If it does not contain data, the request acknowledgment setting is used. See **Settings | Request acknowledgement**.
- That ISA/I14 exists. If it does not contain data, P is inserted.
- That ISA/I15 exists. If it does not contain data, the composite separator from the user-defined settings is inserted. See **Settings | composite separator**.
- That IEA/I16 exists. If it does not contain data, the number of function groups in the interchange is calculated and inserted.
- That IEA/I12 exists. If it does not contain data, the value from ISA/I12 is copied.

Please note:

Any fields not mentioned in this section (Automatic data completion) are NOT inserted, or created. The correct values cannot be ascertained automatically.

Given a (target) parent element A (in the target EDI component) with child items x, y, and z - where y is mandatory, parent element A will only be created in the output file if the mandatory child element "y" in the target component has been mapped!

The potentially existing function group, is checked:

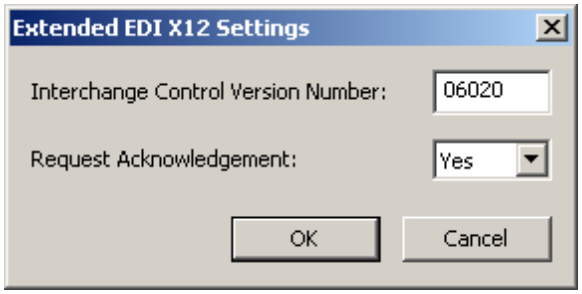
- That GS/373 exists. If it does not contain data, the current date in EDI format is inserted.
- That GS/337 exists. If it does not contain data, the current time in EDI format is inserted.
- That GE/97 exists. If it does not contain data, the number of messages in the function group are calculated and inserted.
- That GE/28 exists. If it does not contain data, the value from GS/28 is copied.

Message checking makes sure:

- That ST/143 exists. If it does not contain data, the name of the message is inserted.

- That SE/96 exists. If it does not contain data, the number of segments in the message is calculated and inserted.
- That ST/329 and SE/329 exist. If SE/329 does not contain data, the value from ST/329 is copied.

Settings
Clicking the Extended... button in the component settings dialog box opens the extended settings dialog box. Please see the EDI/X12 specification for more details:



- Interchange control version number: Default **06020**
- Request acknowledgement: Default **yes**

7.5.9 Customizing an EDIFACT message

MapForce allows you to customize EDIFACT messages to take different non-standard, or changed EDIFACT formats into account.

This example uses the **Orders-Custom-EDI.mfd** file available in the **...\MapForceExamples\Tutorial** folder. Please note that a ZIP file, **EDIFACT.Nanonull.zip**, is also included in the ...\Tutorial folder. The ZIP file contains the files final **result** of the customization procedure.

The EDIFACT file available in the ...\Tutorial folder, **Orders-Custom.EDI**, has been changed to include a new component element in the CTA segment:

- Line 9 contains a **Mr** entry, and
- Line 11 contains a **Mrs** entry.

```
 1    UNB+UNOB:1+003897733:01:MFGB+PARTNER ID:ZZ:ROUTING
      ADDR+970101:1230+00000000000001++ORDERS++++1'
 2    UNH+0001+ORDERS:S:93A:UN'
 3    BGM+221+ABC123456XYZ+9'
 4    DTM+4:200404301742PDT:303'
 5    FTX+PUR+3++Pizza purchase order'
 6    RFF+CT:123-456'
 7    RFF+CR:1122'
 8    NAD+SE+999::92++24h Pizza+Long Way+San-Francisco+CA+34424+US'
 9    CTA+SR+:Ted Little:Mr'
10    NAD+BY+123::92++Nanonull, Inc.+119 Oakstreet Suite 4876+Vereno+CA+29213+US'
11    CTA+PD+:Michelle Butler:Mrs'
12    NAD+ST+123::92++Nanonull, Inc.+119 Oakstreet Suite 4876+Vereno+CA+29213+US'
13    TAX+9++++++1-12345-6789-0'
14    CUX+2:USD:9'
15    PCD+12:2'
16    TDT+20++++:::24h Pizza Fast Carrier'
17    LOC+16+Nanonull Main Entrance'
```

The CTA (Contact Information) segment of the ORDERS message must be extended for the new data to be able to be mapped.

7.5.9.1 EDIFACT: customization setup

The text in the following sections describes how to customize the configuration files to be able to map the changed EDIFACT message to the **ORDER-EDI.xsd** schema. It assumes that the supplied ZIP file is **not** used.

Setting up the customizing example:

1. Create an EDIFACT.Nanonull folder below ***Program Files...****MapForceEDI*\.
2. Copy the following files from the ...**MapForceEDI\EDIFACT** folder into the EDIFACT.Nanonull folder:

   Admin.Segment
   EDI.Collection
   EDSD.Segment
   Envelope.Config
   ORDERS.Config
   UNCL.Codelist

3. Change the attributes of the files to **read-write** to make them editable.

Configuring the EDI.Collection file:

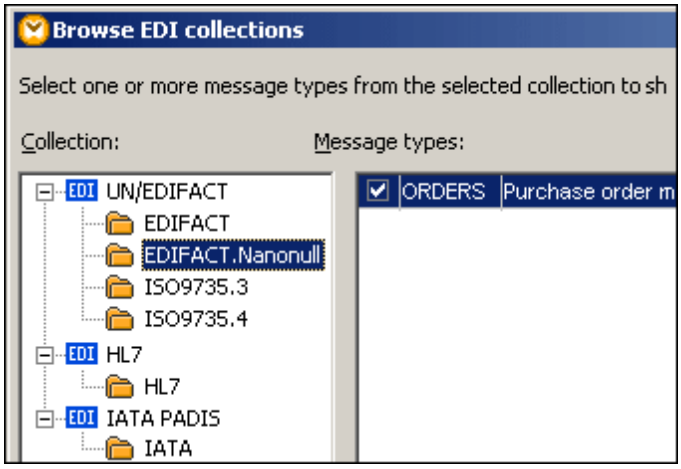
1. Open "EDI.Collection" file in XMLSpy, or in you preferred editor.
2. Remove all "Message" elements, except for the "ORDERS" message. Make sure you retain the <Messages> tags however!

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Messages Version="3">
3    <Meta>
4      <Version>D</Version>
5      <Release>10A</Release>
6      <Agency>UN</Agency>
7    </Meta>
8    <Root File="Envelope.Config"/>
9    <Message Type="ORDERS" File="ORDERS.Config" Description="Purchase order message"/>
10 </Messages>
```

3. Save the file.

To see the content of the collection file:

- Start MapForce, select **Insert | EDI**, or click the Insert EDI icon.
  The Browse EDI collections dialog box opens displaying a new folder named "EDIFACT.Nanonull". Only one entry is visible; the first column shows the message type "ORDERS", and the second the message description text.



Please note:
MapForce searches through all sibling subfolders under the "...**MapforceEDI**" directory and scans for a file called "EDI.Collection". The folder name containing an EDI Collection, appears as a folder in the dialog box. The listbox shows the current content of the collection file, which in our example only contains an ORDERS message entry.

The goal of this section is to redefine the CTA (Contact Information) segment by adding the X1000 field to make it available to individual messages. CTA consists of one field (F3139) and one composite (C056).

There are several ways the customization can be achieved:

- **Globally** by customizing the **EDSD.segment** file. All segments, in all messages that use composite C056, will contain/reference the new element.
- **Inline** by customizing the **ORDERS.Config** file. Only the customized segment (CTA) in the current message will contain the new element.

UNCL.Codelist

This file defines the various EDIFACT codes and the values that they may contain, and is used for validation of input/output files in MapForce. If your organization uses a special code not in the EDIFACT code list, add it here.

EDI.Collection
The Collection file contains a list of all messages in the current directory. It is used to provide a list of the available messages when inserting an EDIFACT file into MapForce. The following example contains only one message, namely "ORDERS".

Edit this file to contain only those messages relevant to your work.

ORDERS.Config
The configuration file for Purchase order message files. This file contains all groups and segment definitions used in Orders messages. Changes made to this file can define local or inline customizations.

Admin.Segment
"Admin.segment" describes the Interchange level administrative segments. They are all used to parse the EDIFACT file.

EDSD.Segment (Electronic Data Segment Definition)
This file defines the Segment, Composite and Field names of the EDIFACT files, and is used when parsing the EDIFACT file. Changes made to this file are global customizations, and apply to all segments and messages.

7.5.9.2 Global customization

- Changes only have to be made to the **EDSD.Segment** file to be able to access the new X1000 field globally.
- All **segments**, in **all messages** that use composite C056, will contain/reference the new element.

Composite redefinition in EDSD.Segment file:
Open the EDSD.Segment file in XMLSpy, or in you preferred editor, and navigate to **Config | Elements | Composite | C056**.

```
<Composite name="C056" info="DEPARTMENT OR EMPLOYEE DETAILS">
    <Data ref="F3413" minOccurs="0"/>
    <Data ref="F3412" minOccurs="0"/>
</Composite>
```

Insert the following line in the C056 segment, under F3412:

```
<Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
```

The composite definition appears as shown below:

```
<Composite name="C056" info="DEPARTMENT OR EMPLOYEE DETAILS">
    <Data ref="F3413" minOccurs="0"/>
    <Data ref="F3412" minOccurs="0"/>
    <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
</Composite>
```

Please note:
The new X1000 field is defined using the "name" attribute as opposed to other fields of the segment which use the "ref" attribute. The two other fields are defined at the beginning of the EDSD-Segment file, outside of the Composite section, (using the Data name element) and are

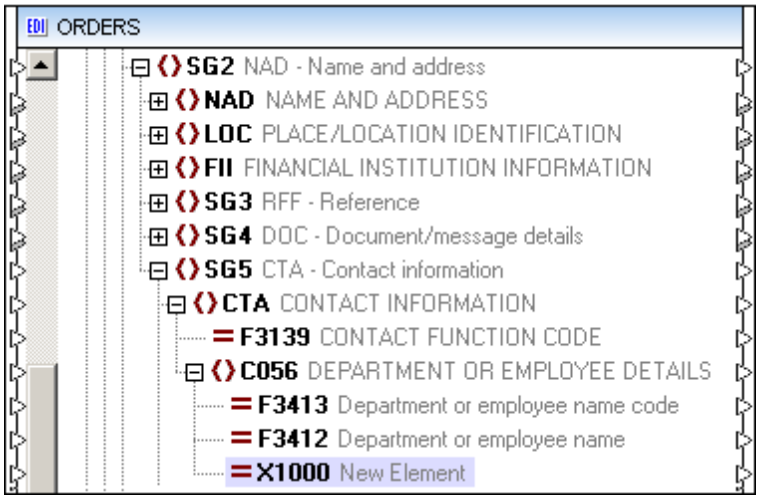
only referenced here. The new field can now be referenced from different Segments or Composites.

When customizing your EDI files the values that can be used for e.g. the "type" attribute are generally any of XML Schema types that are used in the delivered config files. Other XML Schema simple types can possibly be used, but cannot be guaranteed.

Simple types that are not supported are; "anyType", "ENTITIES" and "QName".

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI icon.
2. Click the "EDIFACT.Nanonull" tab, and select the ORDERS message.
3. Click the Skip button to skip the selection of the source EDI file for the moment. The ORDERS component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message_ORDERS/SG2/SG5/CTA/C056**, to see the new X1000 element.



7.5.9.3 *Inline customization*

- Changes only have to be made to the **ORDERS.config** file, at the specified location, to be able to have inline access to the new X1000 field.
- Only the redefined CTA segment in the **current** message, will contain/reference the new element.
- In other words, the CTA segment is redefined locally to contain a redefined Composite C056, with the local definition of the new field X1000.

Open the ORDERS.Config in XMLSpy, or in you preferred editor, and navigate to **Message | Envelope | Interchange | Group | Message | SG2 | SG5 | CTA** (or search for **SG5**).

```
47  <Group name="SG5" minOccurs="0" maxOccurs="5" info="CTA - Contact information">
48      <Segment ref="CTA"/>
49      <Segment ref="COM" minOccurs="0" maxOccurs="5"/>
50  </Group>
```

Replace the line :

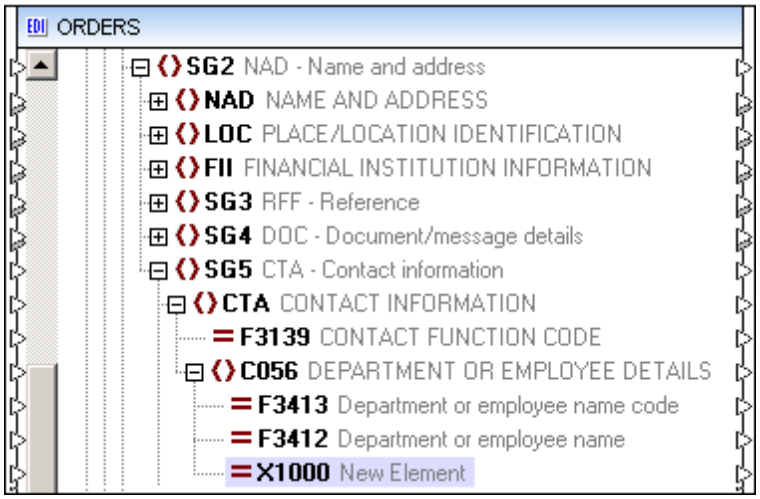
with the following lines:

```
<Segment name="CTA" id="CTA_ORDERS_SG5" info="CONTACT INFORMATION">
   <Data ref="F3139" minOccurs="0"/>
   <Composite name="C056" minOccurs="0" info="DEPARTMENT OR EMPLOYEE DETAILS">
         <Data ref="F3413" minOccurs="0"/>
         <Data ref="F3412" minOccurs="0"/>
         <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New
Element"/>
   </Composite>
</Segment>
```

```
47   <Group name="SG5" minOccurs="0" maxOccurs="5" info="CTA - Contact information">
48       <Segment name="CTA" id="CTA_ORDERS_SG5" info="CONTACT INFORMATION">
49       <Data ref="F3139" minOccurs="0"/>
50           <Composite name="C056" minOccurs="0" info="DEPARTMENT OR EMPLOYEE DETAILS">
51           <Data ref="F3413" minOccurs="0"/>
52           <Data ref="F3412" minOccurs="0"/>
53           <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Ele
54       </Composite>
55   </Segment>
56   <Segment ref="COM" minOccurs="0" maxOccurs="5"/>
57   </Group>
```

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "EDIFACT.Nanonull" tab, and select the ORDERS message.
3. Click the Skip button to skip the selection of the source EDI file for the moment. The ORDERS component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message_ORDERS/SG2/SG5/CTA/C056**, to see the new X1000 element.



7.5.9.4 *Customized Orders mapping example*

The mapping visible in the images below, Orders-Custom-EDI.mfd, is available in the ...\MapForceExamples\Tutorial\ directory. The example maps the ORDERS-Custom.EDI file to the

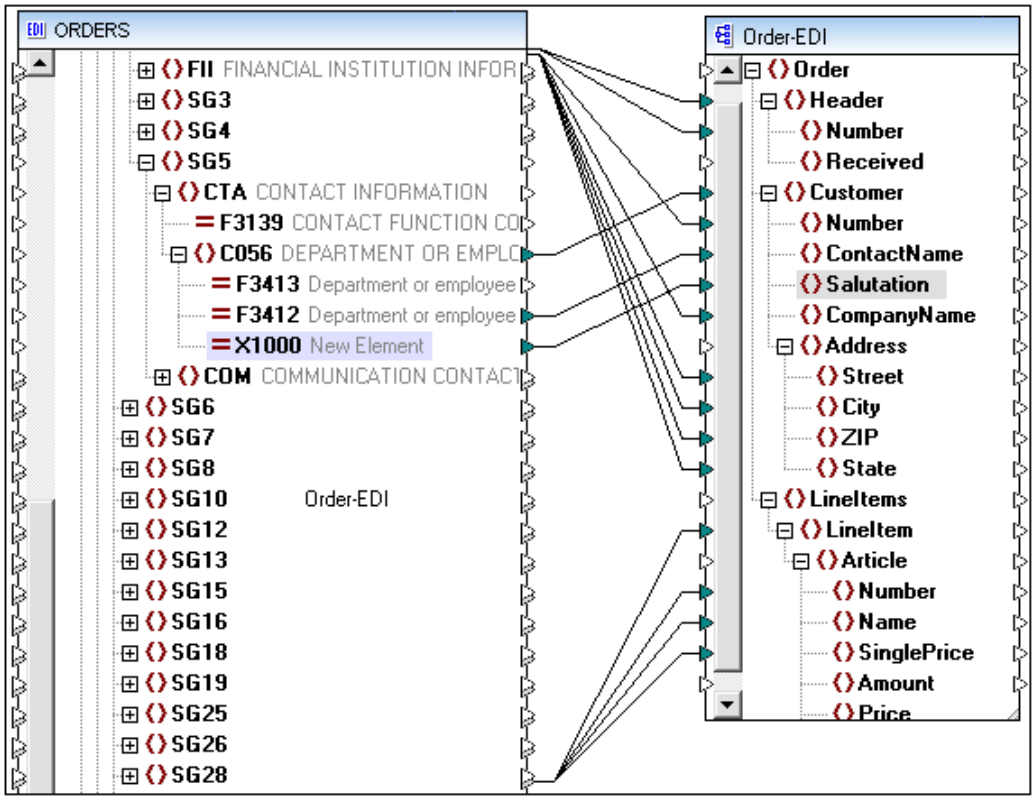
Order-EDI schema. The field that has been added to the EDI structure, X1000, has been mapped to the Salutation item.

To see the customization result:

1. Create a new folder under the "**...\MapForceEDI**" directory and name it e.g. "**EDIFACT.Nanonull**"
2. Unzip the supplied EDIFACT.Nanonull.zip file (from the ...\Tutorial folder) into the new folder. The ZIP file contains the follow files:

   Admin.Segment
   EDI.Collection
   EDSD.Segment
   Orders.config
   UNCL.Codelist

3. Open the **Orders-Custom-EDI.mfd file** and click the Output tab to see the result of the mapping.



Clicking the Output tab displays the mapping result shown below.


```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
3    <Header>
4      <Number>ABC123456XYZ</Number>
5    </Header>
6    <Customer>
7      <Number>92</Number>
8      <ContactName>Ted Little</ContactName>
9      <Salutation>Mr</Salutation>
10     <CompanyName>24h Pizza</CompanyName>
11     <Address>
12       <Street>Long Way</Street>
13       <City>San-Francisco</City>
14       <ZIP>34424</ZIP>
15       <State>CA</State>
16     </Address>
17   </Customer>
18   <Customer>
19     <Number>92</Number>
20     <ContactName>Michelle Butler</ContactName>
21     <Salutation>Mrs</Salutation>
22     <CompanyName>Nanonull, Inc.</CompanyName>
```

Code generation note:
When generating C++ code, a class named "CX1000Type" is generated which is accessible from the "CC056Type" class.

7.5.10 Customizing an ANSI X12 transaction

MapForce allows you to customize X12 transactions to take different nonstandard, or changed X12 formats into account.

This example uses the **Orders-Custom-X12.mfd** file available in the ...\MapForceExamples\Tutorial\ folder. Please note that a ZIP file, **X12.Nanonull.zip**, is also included in the ...\Tutorial folder. The ZIP file contains the files final result of the customization procedure.

The X12 source file available in the ...\Tutorial folder, **Orders-Custom.X12**, has been changed to include a new field in the N2 segment:

- Line 6 contains an additional ++**Mrs** entry

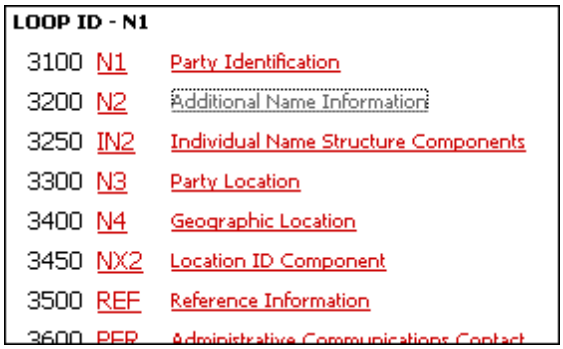
```
1   ISA+00+          +00+          +ZZ+SenderID       +ZZ+ReceiverID
2   GS+PO+SenderID+ReceiverID+20060308+182347+1+UN+050112'
3   ST+850+12345'
4   BEG+00+SA+ABC123456XYZ++20040430'
5   N1+1 +Nanonull, Inc.++123'
6   N2+Michelle Butler++Mrs'
7   N3+119 Oakstreet Suite 4876'
8   N4+Vereno+CA+29213'
9   PO1++1++7.2'
10  PO3+1 +++7.2++1+US'
11  PID+1++++Pizza Pepperoni'
12  TXI+1 ++9'
13  AMT+1+720'
14  PO1++2++13.2'
15  PO3+1 +++6.6++2+US'
```

Please note:
Customization information for EDIFACT and ANSI X12, is supplied by two structurally equivalent files that allow the use of multiple consecutive elements of the same name in each of the EDI formats. EDIFACT uses the **EDSD.segment** file while ANSI X12 uses the **X12.Segment** file. Please see "Multiple consecutive elements" for more information.

7.5.10.1 X12 source files and multiple identical fields

When customizing an X12 transaction it is important to take note that segments often allow the use of multiple **consecutive elements** of the same **name**.

Multiple consecutive elements of the same name
The goal of this section is to be able to map an ANSI X12 file that has been customized to an XML schema file. A new field has been added to the N2 segment i.e. "Mrs". This additional data field should be mapped to the Salutation field in the XML schema.



Adding a new field to an X12 file would normally entail adding a single separator character, generally the + character, followed by the data. The N2 segment specification, shown below, allows for two consecutive fields specified as 'Name'.

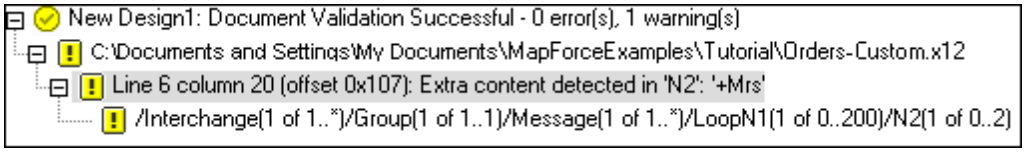
REF	ELE ID	NAME	RPT	ATTRIBUTES
01	93	Name	M	AN 1/60
02	93	Name	O	AN 1/60

The X12 source file therefore has to take this into account, by adding an "empty" field separator for the first (mandatory) occurrence, and a second one to separate the actual data. This means that **++Mrs** has to be entered for the data to adhere to the X12 specification.

You can find out the specific fields that have multiple entries by looking at the **X12.Segment file** supplied with MapForce, in the ...**MapForceEDI\X12** folder. Any segment that may contain multiple identical field entries (e.g. 93), is shown as the field number with a **mergedEntries** attribute which defines how many multiples may occur, in this case 2.

```
<Segment name="N2" info="Additional Name Information">
    <Data ref="F93" mergedEntries="2"/>
</Segment>
```

Inserting the X12 file with only one + separator, causes the following warning to appear, in the Messages window, when the EDI component is inserted and the sample EDI file has been assigned:



The Messages window supplies very specific help on the location and cause of a message. Clicking the respective message displays the specific line in the component if a connector is missing, or shows that extra content exists for one of the items.

7.5.10.2 X12 customization setup

The text in the following sections describes how to customize the configuration files to be able to map the changed X12 transaction to the **ORDER-x12.xsd** schema. It assumes that the supplied ZIP file is **not** used.

Setting up the customizing example:

1. copy the following files from the ...**MapForceEDI\X12** folder into the X12.Nanonull folder:

   EDI.Collection
   Envelope.Config
   850.Config
   X12.Segment
   X12.Codelist
2. Change the attributes of these files to **read-write,** to make them editable.

Configuring the EDI.Collection file:

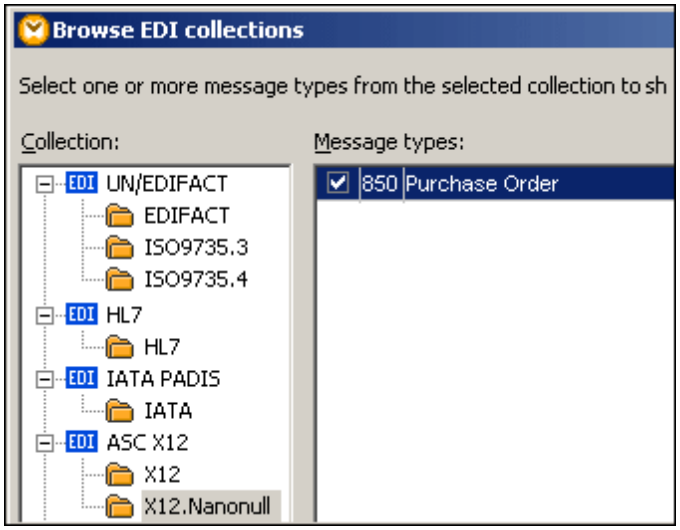
1. Open "EDI.Collection" file in XMLSpy, or in you preferred editor.
2. Remove all "Message" elements, except for the "850 Purchase Orders". Make sure you retain the <Messages> tags however!

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Messages Version="3">
3    <Meta>
4      <Release>6020</Release>
5      <Agency>X12</Agency>
6    </Meta>
7    <Root File="Envelope.Config"/>
8    <Message Type="850" File="850.Config" Description="Purchase Order"/>
9  </Messages>
```

3. Save the file.

To see the content of the collection file:

- Start MapForce, select **Insert | EDI**, or click the Insert EDI icon.
  The Browse EDI collections dialog box opens displaying a new folder named "X12.Nanonull". Only one entry is visible; the first column shows the message type "850", and the second the message description text "Purchase Order".



Please note:

MapForce searches through all sibling subfolders under the "...**MapforceEDI**" directory and scans for a file called "EDI.Collection". The folder name containing an EDI Collection, appears as a folder in the dialog box. The listbox shows the current content of the collection file, which in our example only contains a Purchase Order entry.

The goal of this section is to redefine the N2 "Additional Name Information" segment by adding the X1000 field to make it available to individual transactions. N2 currently consists of one field, "F93 Name".

There are several ways the customization can be achieved:

- **Globally** by customizing the **X12.Segment** file. All segments, in all transactions that use N2, will contain/reference the new element.
- **Inline** by customizing the **850.Config** file. Only the customized segment (N2) in the current transaction will contain the new element.

EDI.Collection
The Collection file contains a list of all transactions in the current directory. It is used to provide a list of the available transactions when inserting an X12 file into MapForce. The following example contains only one transaction, namely "Purchase Order".

850.Config
The configuration file for Purchase order transaction files. This file contains all groups and segment definitions used in purchase order transactions. Changes made to this file can define local or inline customizations.

X12.Segment
This file defines the Segment, Composite and Field names of the X12 files, and is used when parsing the file. Changes made to this file are global customizations, and apply to all segments and transactions.

7.5.10.3 Global customization

- Changes only have to be made to the **X12.Segment** file to be able to access the new X1000 field globally.
- All **segments**, in **all transaction** that use **N2**, will contain/reference the new element.

Redefinition in X12.Segment file:
Open the X12.Segment file and navigate to **Config | Elements | Segment name="N2" info="Additional Name Information"**.

```
<Segment name="N2" info="Additional Name Information">
    <Data ref="F93" mergedEntries="2"/>
</Segment>
```

Insert the following line under F93, and save the file:

```
<Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
```

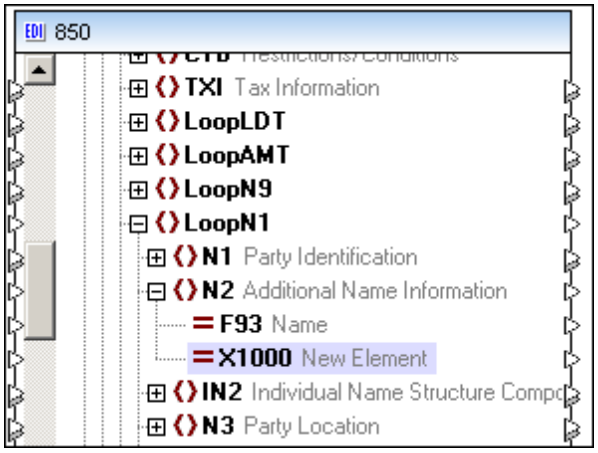
The definition appears as shown below:

```
<Segment name="N2" info="Additional Name Information">
    <Data ref="F93" mergedEntries="2"/>
    <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
</Segment>
```

Please note:
The new X1000 field is defined using the "name" attribute as opposed to other fields of the segment which use the "ref" attribute. The F93 field is defined at the beginning of the X12.Segment file using the Data name element and is only referenced here. The new field can now be referenced from different Segments or Composites.

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "X12.Nanonull" tab, and select the 850 Purchase Order transaction.
3. Click the Skip button to skip the selection of the source EDI file for the moment. The 850 component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message_850/LoopN1/N2**, to see the new X1000 element.



7.5.10.4 Inline customization

- Changes only have to be made to the **850.config** file, at the specified location, to be able to access the new X1000 field locally.
- Only the redefined segment N2 in the **current** transaction, will contain/reference the new X1000 field.
- In other words, the segment is redefined locally to contain the new field X1000.

Local customization - segment redefinition in 850.Config file:
Open the 850.Config file and navigate to **Config | Group | Message** | **Group name="LoopN1"** (or search for **LoopN1**).

```
73  <Group name="LoopN1" maxOccurs="200" minOccurs="0">
74      <Segment ref="N1"/>
75      <Segment ref="N2" minOccurs="0" maxOccurs="2"/>
76      <Segment ref="IN2" minOccurs="0" maxOccurs="unbounded"/>
77      <Segment ref="N3" minOccurs="0" maxOccurs="2"/>
78      <Segment ref="N4" minOccurs="0" maxOccurs="unbounded"/>
```

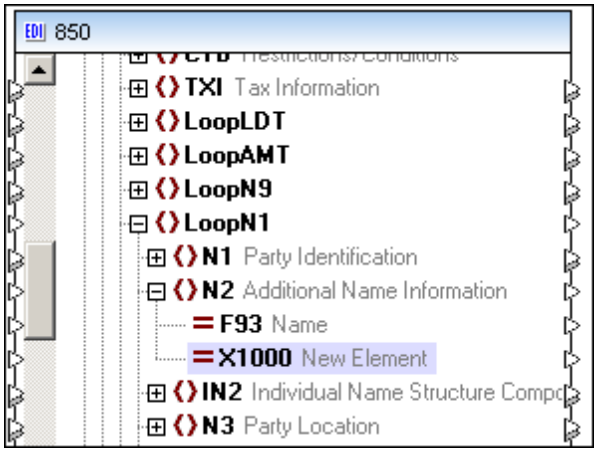
Replace the Segment ref="N2"... line with the following lines:

```
<Segment name="N2" info="Additional Name Information">
        <Data ref="F93" mergedEntries="2"/>
        <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
</Segment>
```

```
73  <Group name="LoopN1" maxOccurs="200" minOccurs="0">
74      <Segment ref="N1"/>
75      <Segment name="N2" info="Additional Name Information">
76          <Data ref="F93" mergedEntries="2"/>
77          <Data name="X1000" type="string" maxLength="35" minOcc
78      </Segment>
79      <Segment ref="IN2" minOccurs="0" maxOccurs="unbounded"/>
```

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "X12.Nanonull" tab, and select the 850 Purchase Order transaction.
3. Click the Cancel button to skip the selection of the source EDI file for the moment. The 850 component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message/LoopN1/N2**, to see the new X1000 element.

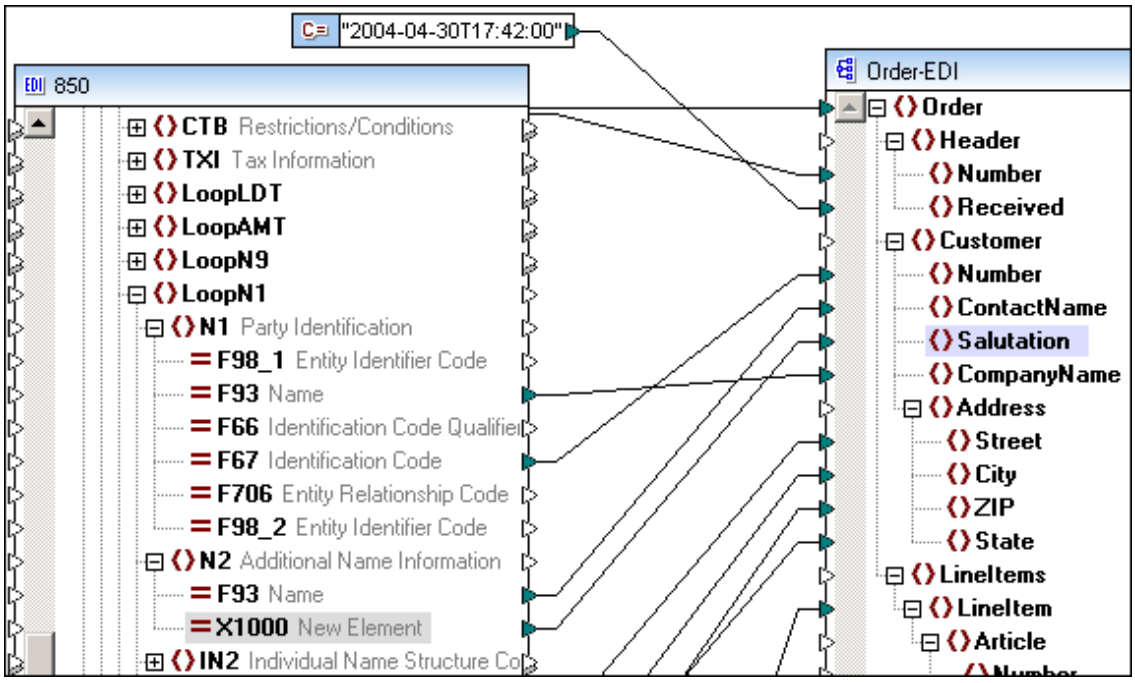


7.5.10.5 Customized X12 mapping example

The mapping visible in the images below, **Orders-Custom-X12.mfd**, is available in the [...\MapForceExamples\Tutorial\] directory.

The example maps the Orders-Custom.x12 file to the Order-X12 schema. The field that has been added to the EDI structure, X1000, has been mapped to the Salutation item.

1. Create a new folder under the "**...\MapForceEDI**" directory and name it e.g. "**X12.Nanonull**"
2. Unzip the supplied X12.Nanonull.zip file (from the ...\Tutorial folder) into the new folder.
3. Open the **Orders-Custom-X12.mfd file.**



Clicking the Output tab displays the mapping result shown below.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
3    <Header>
4      <Number>ABC123456XYZ</Number>
5      <Received>2004-04-30T17:42:00</Received>
6    </Header>
7    <Customer>
8      <Number>123</Number>
9      <ContactName>Michelle Butler</ContactName>
10     <Salutation>Mrs</Salutation>
11     <CompanyName>Nanonull, Inc.</CompanyName>
12     <Address>
13       <Street>119 Oakstreet Suite 4876</Street>
14       <City>Vereno</City>
15       <ZIP>29213</ZIP>
16       <State>CA</State>
17     </Address>
```

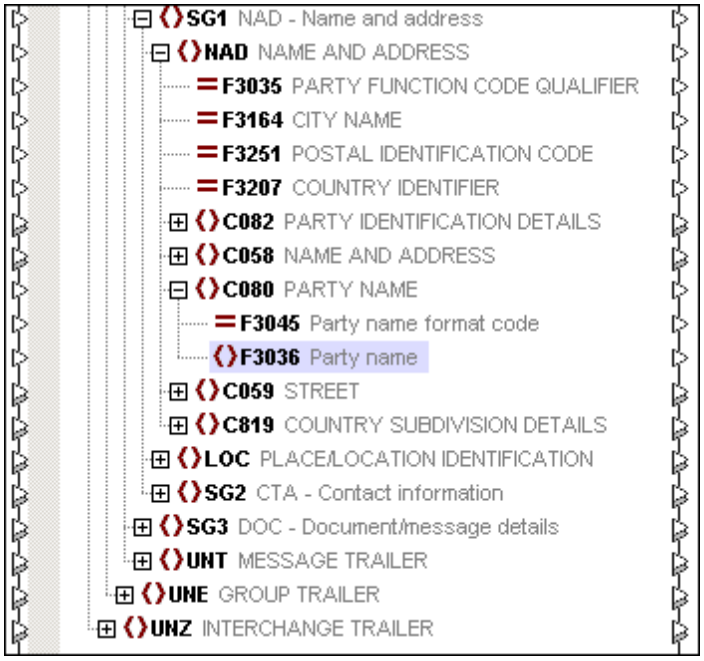
7.5.11 Splitting merged entries into separate nodes/items

When mapping to EDI components MapForce may automatically supply a field entry as a **single** mappable item/node, in the EDI component, even though it may consist of multiple consecutive occurrences of the same field.

Customization information for EDIFACT and ANSI X12, is supplied by two structurally equivalent files that allow the use of multiple consecutive elements of the same name in each of the EDI formats. EDIFACT uses the **EDSD.Segment** file while ANSI X12 uses the **X12.Segment** file.

Editing the EDSD.Segment file allows you to split the merged sequence into its constituent parts (occurrences) and create unique nodes/mappable items. Please make sure that you create a new directory for the EDSD.Segment file and any other EDI files that you might edit, as described here.

The OSTRPT component shown below contains the field F3036 Party name, which is shown as a single mappable item. The Party name field may contain five separate/unique Party Names per EDIFACT definition however.



Opening the EDSD.Segment file shows the definition of the C080 composite Party Name. A segment that contains multiple identical field entries (e.g. F3036), is shown as the field number with a **mergedEntries** attribute which defines how many multiples may occur, e.g. 5 as shown in the screenshot below.

```
<Composite name="C080" info="PARTY NAME">
    <Data ref="F3036" mergedEntries="5"/>
    <Data ref="F3045" minOccurs="0"/>
</Composite>
```

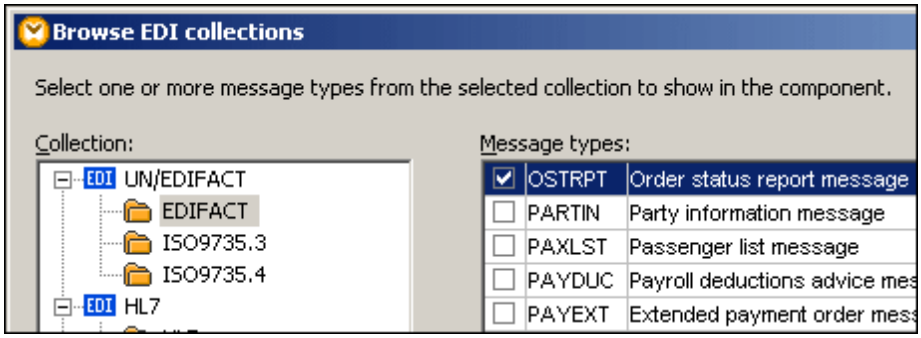
To create unique fields from a merged sequence:

1. Open the **EDSD.Segment** (X12.Segment) file that can be found in the folder c:\Program Files\Altova\MapForce2016\MapForceEDI\EDIFACT\ (or c:\Program Files\Altova \MapForce2016\MapForceEDI\X12\)
2. Remove the **mergedEntries**="5" attribute.
3. Copy the remaining <Data ref="F3036"/> element and insert it the number of times previously shown in the mergedEntries attribute, i.e. 5 times.
4. Add the **minOccurs="0"** attribute to all fields **except** for the **first** one.
5. Enter a unique node name for each of the fields e.g. **nodeName_1** etc.

```
<Composite name="C080" info="PARTY NAME">
    <Data ref="F3036" nodeName="F3036_1"/>
    <Data ref="F3036" nodeName="F3036_2" minOccurs="0"/>
    <Data ref="F3036" nodeName="F3036_3" minOccurs="0"/>
    <Data ref="F3036" nodeName="F3036_4" minOccurs="0"/>
    <Data ref="F3036" nodeName="F3036_5" minOccurs="0"/>
    <Data ref="F3045" minOccurs="0"/>
</Composite>
```

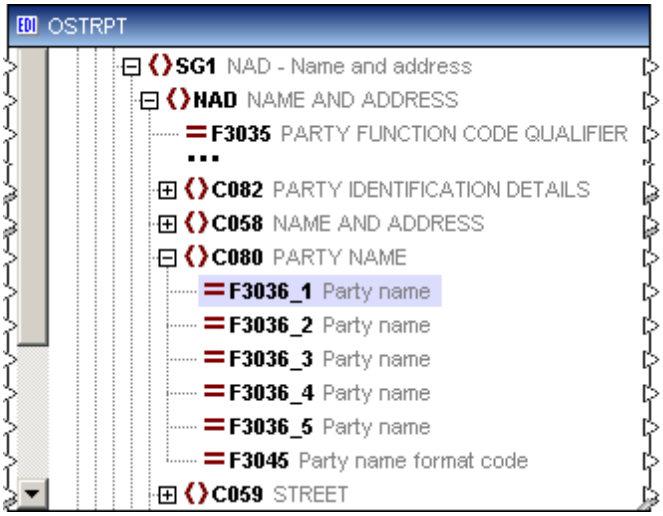
6. Save the edited EDSD.Segment file.
7. Open MapForce, click the Insert EDI  icon and insert the "Order Status Report

message" from the EDIFACT tab.



This creates the OSTRPT component.

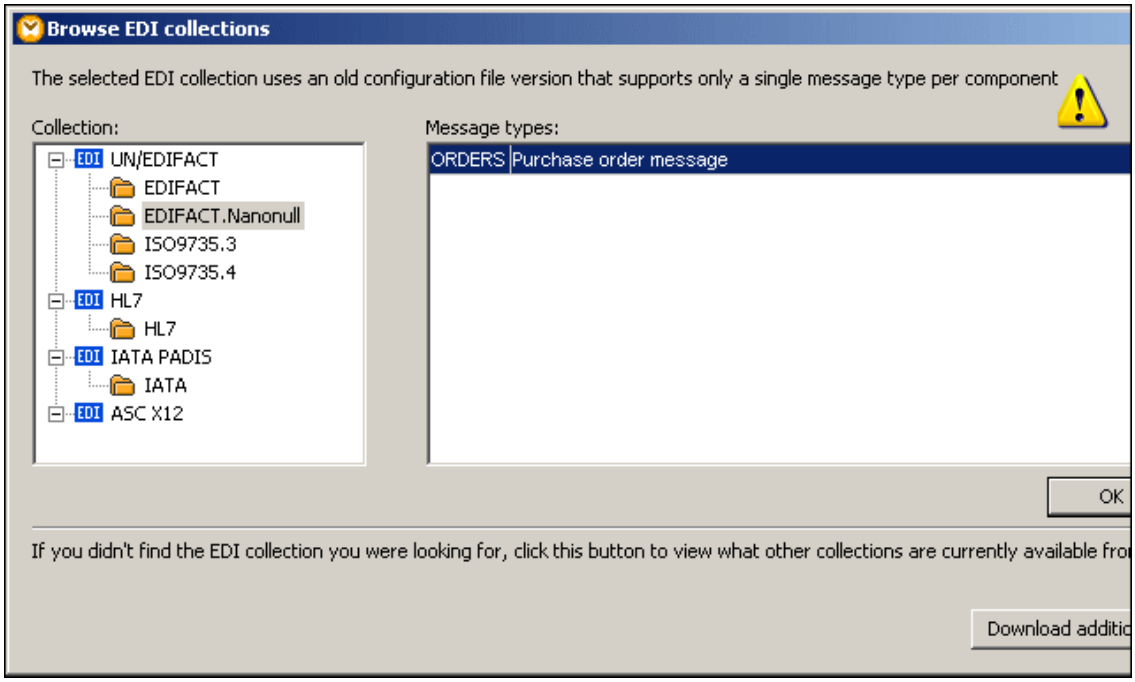
8. Expand the items down to **Envelope | Interchange | Group | Message_OSTRPT | SG1 NAD**.
9. Expand C080 to see the new unique Party name fields.
   You can now map to and from these individual Party name items/nodes.



Please see "Multiple consecutive elements" for more information.

7.5.12 Upgrading config files to support multiple messages

If you are using EDI configuration files prior to version 3, then the dialog box shown below will prompt you that you are using a configuration file that only supports a single message type per component.



To upgrade the configuration files to support multiple message types per component:

1. Copy Envelope.Config from the original config folder (e.g. EDIFACT) to the folder containing you customized config files (e.g. EDIFACT.Nanonull).
2. Edit **EDI.Collection**, and change the root element's Version attribute from 2 to 3.
3. Add "<Root File="Envelope.Config"/>" after the </Meta> tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="3">
        <Meta>
                <Version>D</Version>
                <Release>04B</Release>
                <Agency>UN</Agency>
        </Meta>
        <Root File="Envelope.Config"/>
        <Message Type="ORDERS" File="ORDERS.Config" Description="Purchase order message"/>
</Messages>
```

4. Edit **ORDERS.Config**, and change the root element's Version attribute from 2 to 3.
5. Add "<Format standard="EDIFACT"/>" (or X12, or HL7).
6. Rename "<Group name="Message"...>" to "<Group name="Message_ORDERS"...>" (or whatever the custom message type is), and
7. Remove the outer group levels ("Envelope", "Interchange, and their segments):

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="3">
        <Meta>
                <Version>D</Version>
                <Release>04B</Release>
                <Agency>UN</Agency>
```

```
                </Meta>
                <Format standard="EDIFACT"/>
                <Include href="Admin.Segment"/>
                <Include href="EDSD.Segment"/>
                <Include href="UNCL.Codelist"/>
                <Message>
                        <MessageType>ORDERS</MessageType>
                        <Description>Purchase order message</Description>
                        <Revision>14</Revision>
                        <Date>2004-11-23</Date>
                        <Group name="Envelope">
                                <Group name="Interchange" maxOccurs="unbounded">
                                        <Segment ref="UNA" minOccurs="0"/>
                                        <Segment ref="UNB" minOccurs="0"/>
                                        <Group name="Group"
maxOccurs="unbounded">
                                                <Segment ref="UNG"
minOccurs="0"/>
                                                        <Group
name="Message_ORDERS" maxOccurs="unbounded" info="UNH - Message header">
                                                                <Segment ref="UNH"/
>
                                                                <Segment ref="BGM"/
>
.
                                                                <Segment ref="UNT"/>
                                                        </Group>
                                                        <Segment ref="UNE"
minOccurs="0"/>
                                        </Group>
                                        <Segment ref="UNZ" minOccurs="0"/>
                                </Group>
                        </Group>
                </Message>
```

If a mapping was loaded that uses this configuration file while editing the configuration file, it should be reloaded. The connections will be automatically remapped from "Message" to "Message_ORDERS" item.

The same method is used to upgrade the EDI X12, or HL7 configuration files.

7.5.13 SAP IDocs

SAP IDocs (intermediate documents) documents are used to exchange business data between SAP and non-SAP applications. The documents are a form of intermediate data storage which can be exchanged between different systems.

An IDoc is structured as follows:

- **Control Record:** contains control information about the IDoc: sender, receiver, message type, and IDoc type. The control record format is similar for all IDoc types.

- **Data Segment:** contains the actual data of the segment as well as other metadata: header, segment no. and type as well as the fields containing the data.
- **Status Records:** contain info on the current status of the document, i.e. the currently processed stages, and the stages that still need to be processed. The status format is identical for all types of IDoc.

The version number in the port definition defines the systems you are communicating with. The major differences between the versions are the various name lengths used in the various elements and the use of extensions. SAP R3 version 4.X supports long names (as well as extensions) while the previous versions do not.

Port Version 1: Releases 2.1. and 2.2.
Port Version 2: Releases 3.0, 3.1 and R/2 systems.
Port Version 3: Release 4.x (default value)

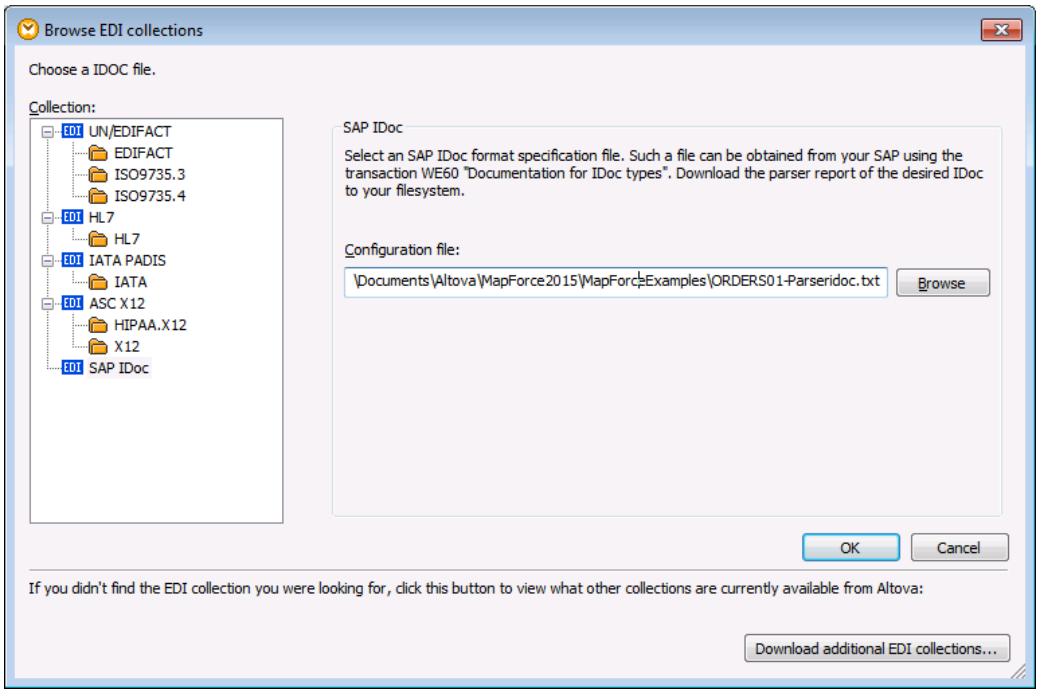
MapForce treats IDoc components as fixed-length files of length 30 char for Message type, 30 for IDoc type, and 27 for segment fields.

For an example which illustrates mapping data from SAP IDoc to XML, see **<Documents>\Altova\MapForce2016\MapForceExamples\IDoc_Order.mfd**.

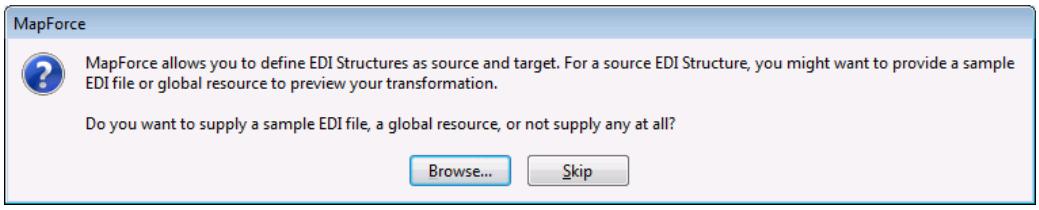
To add an SAP IDoc document as mapping component:

1. Select the menu option **Insert | EDI**. This opens the Browse EDI collections dialog box.
2. Click the **SAP IDoc** entry in the list box.
3. Click **Browse** and select the IDoc parser report (also known as "IDoc definition file").

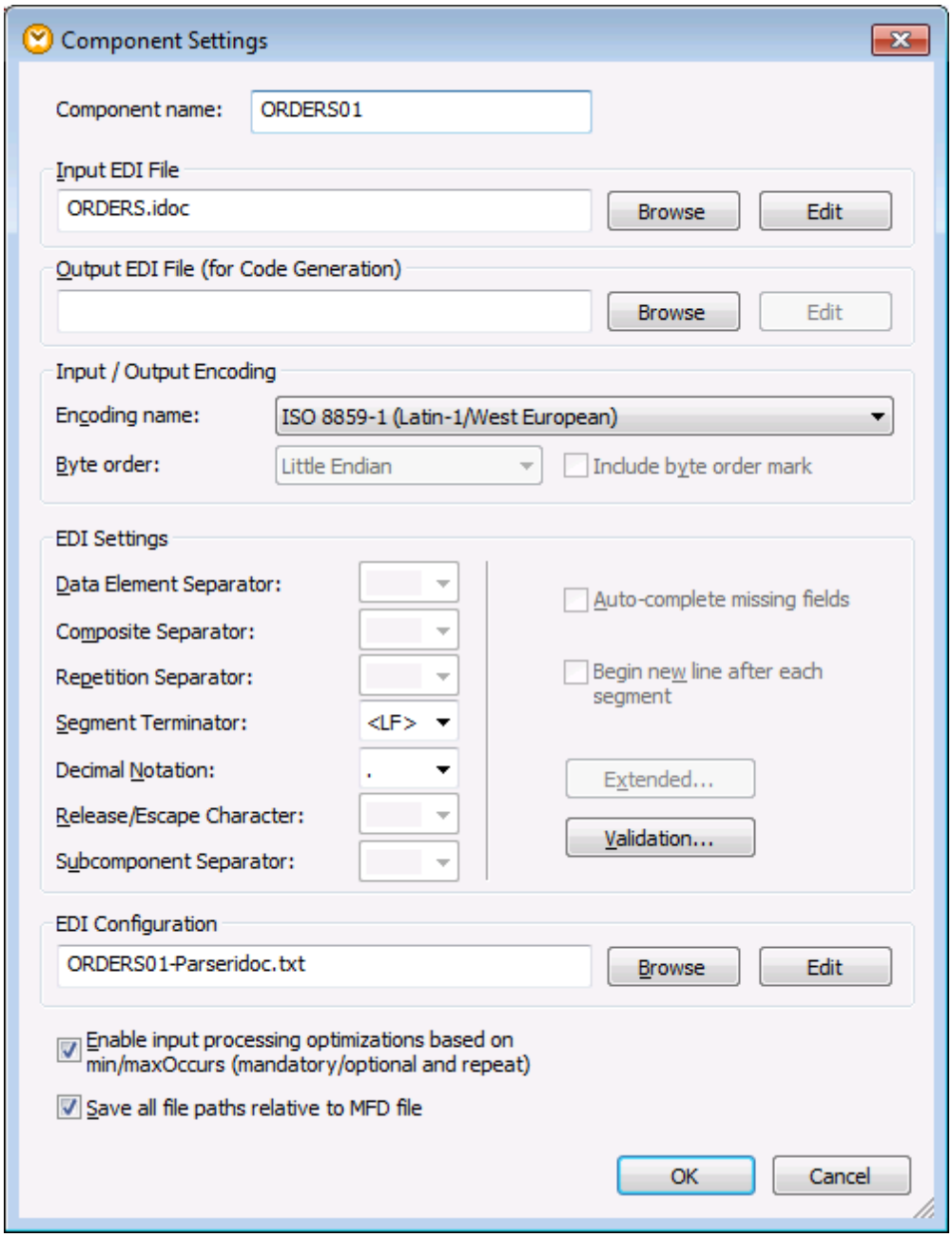
The IDoc parser report is created from the SAP system using the transaction WE60 ("Documentation for IDoc types"). Note that the file must be exported from SAP in uncompressed format. For information on how to invoke transaction WE60 and generate the parser report, refer to the SAP IDoc documentation. The MapForce Examples folder includes a sample parser report file (see **<Documents>\Altova\MapForce2016\MapForceExamples\ORDERS01-Parseridoc.txt**).



4. Click **OK**. MapForce prompts you to optionally select a sample EDI file.



5. If you would like to map data *from* an IDoc file, click **Browse** and select the IDoc (*.idoc) file that supplies the data. A sample IDoc file is available in the MapForce Examples folder (see **<Documents>\Altova\MapForce2016\MapForceExamples\ORDERS.idoc**). Otherwise, click **Skip**.
6. The Component Settings dialog box is displayed. This enables you to review the settings of the IDoc component before adding it to the mapping. You can change these settings at any time later if required (see Changing the Component Settings).



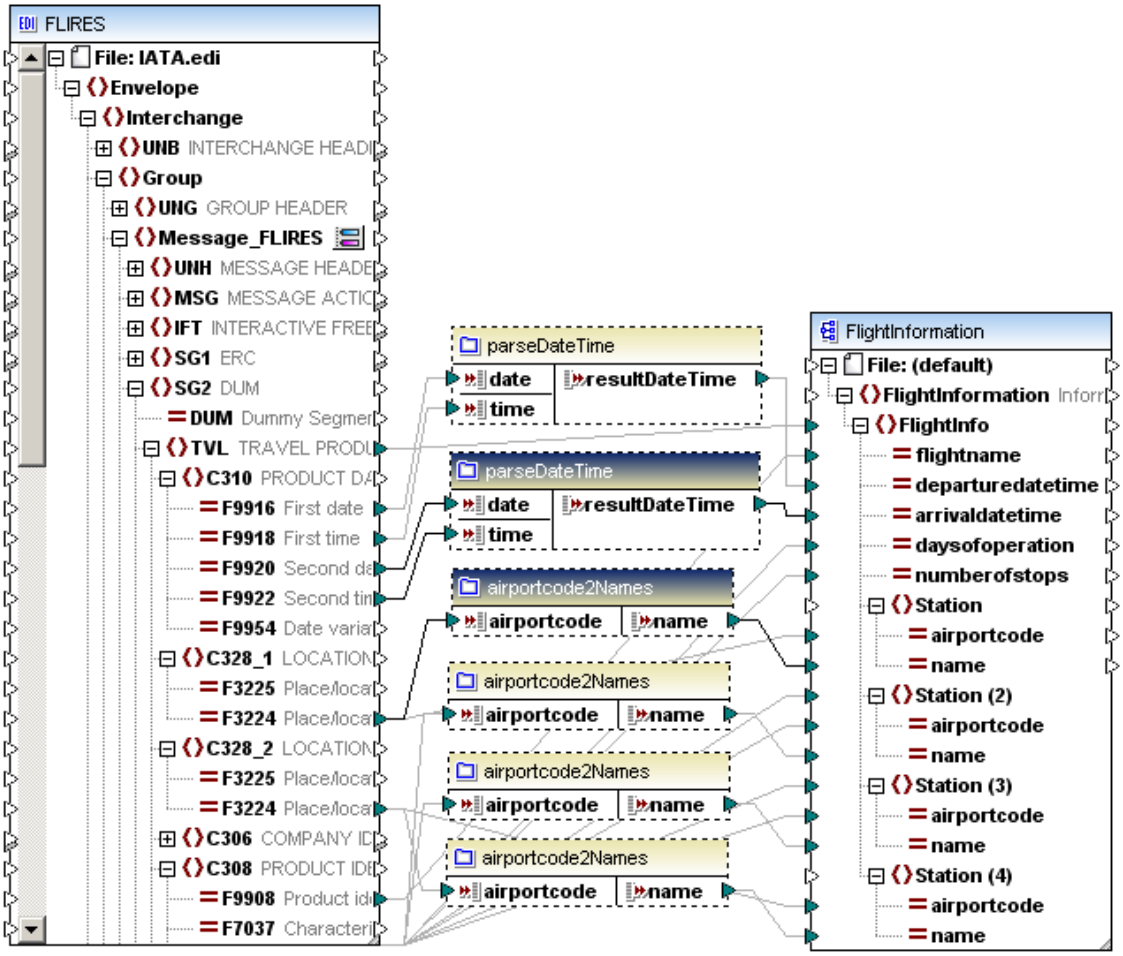
7. Click OK to close the Component Settings dialog box and add the IDoc component to the mapping area.

7.5.14 IATA PADIS

PADIS (Passenger and Airport Data Interchange Standards) are a set of messages using the EDIFACT (ISO 9735) syntax.

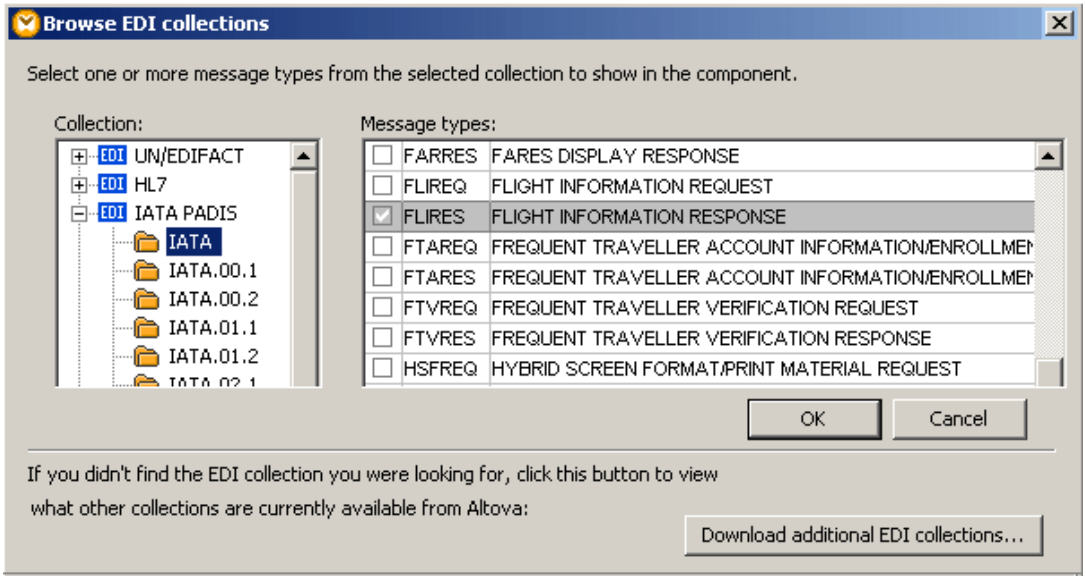
- MapForce currently only supports messages using the UNH/UNT message header and trailer segments.
- MapForce supports the collections IATA.00.1 to IATA.08.1.

The **IATA_FlightInformationReport.mfd** file available in the ...\MapForceExamples folder, shows an example mapping of an IATA PADIS file to an XML Schema target file.

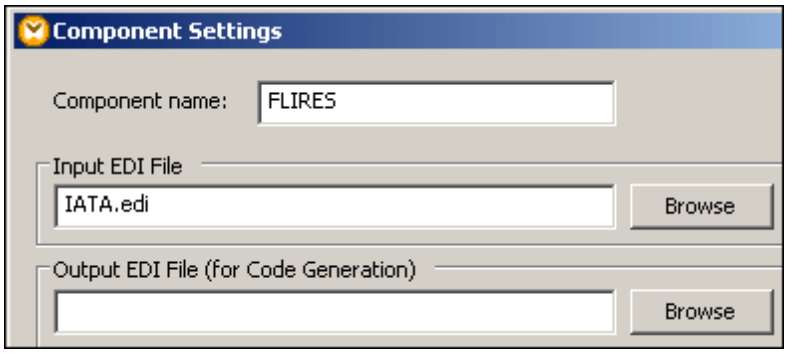


To insert an PADIS component:

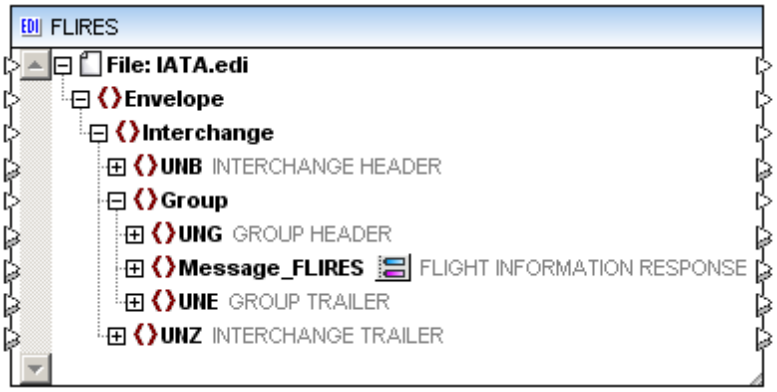
1. Select the menu option **Insert | EDI**.
   This opens the Browse EDI collections dialog box.
2. Click the IATA folder (or the specific IATA collection you use) under the IATA PADIS entry in the Collection list box.
3. Click the check box of the Message type(s) you want to insert, e.g. FLIRES, then click OK:



4. Click the Browse button of the following prompt to supply a sample EDI file, e.g. IATA.edi.
   The Component Settings dialog box is displayed.



   The IATA.edi file supplies the data for the source component. IATA.edi is available in the ...\MapForceExamples folder.
5. Click OK to close the Component Settings dialog box and insert the PADIS component.



Please note:
the Select EDI message icon , opens the "Browse EDI collections" dialog box, allowing you to select multiple messages per interchange (per EDI component).

7.5.15 HIPAA X12

HIPAA X12 is the latest version of the standard for electronic health care records established by the US Department of Health and Human Services for electronic medical data transactions between insurers, providers, and employers, based on EDI X12 version 5010.

MapForce supports the latest release, A2, of the HIPAA implementation specs (TR3). Older releases are downloadable as separate ZIP file from Altova website.

HIPAA components are similar to ordinary ANSI X12 components, and MapForce supports the following transactions:

X12 name	Message name
X279A1	"Health Care Eligibility Benefit Inquiry (270)"
X279A1	"Health Care Eligibility Benefit Response (271)"
X212	"Health Care Claim Status Request (276)"
X212	"Health Care Information Status Notification (277)"
X214	"Health Care Claim Acknowledgment (277)"
X217	"Health Care Services Review - Request for Review (278)"
X217	"Health Care Services Review - Response (278)"
X218	"Payroll Deducted and Other Group Premium Payment for Insurance Products (820)"
X220A1	"Benefit Enrollment and Maintenance (834)"
X221A1	"Health Care Claim Payment/Advice (835)"
X222A1	"Health Care Claim: Professional (837)"
X224A2	"Health Care Claim: Dental (837)"
X223A2	"Health Care Claim: Institutional (837)"
X231A1	"Implementation Acknowledgment For Health Care Insurance (999)"

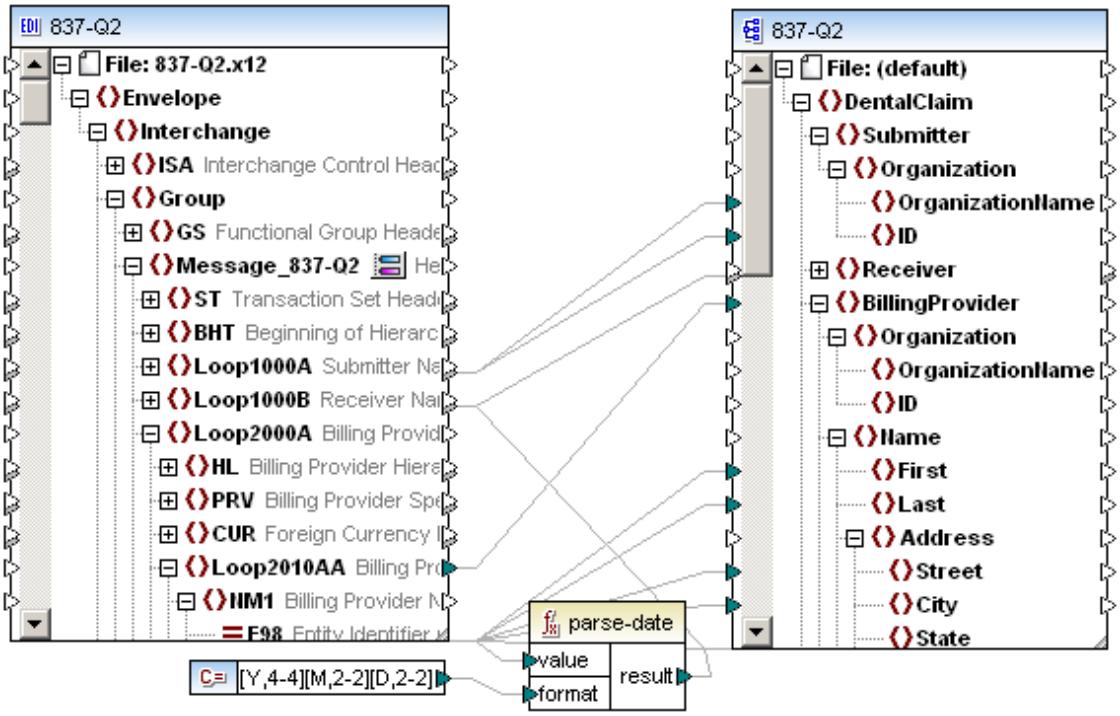
Differences to standard X12 message handling is that MapForce:

- automatically maintains the hierarchy of HL segments
- supports so called floating structures (in 837 messages)
- auto-completes and validates more fields.

MapForce also supports the auto-generation of X12 999 implementation acknowledgment, which is similar to the existing 999 and 997 functional acknowledgment in standard X12.

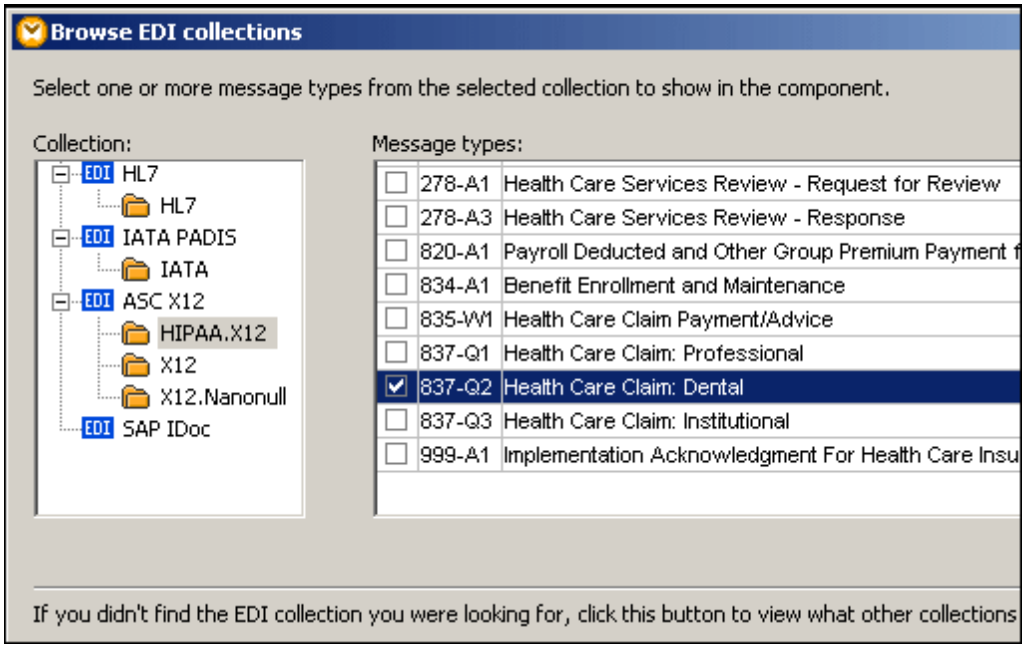
Multiple messages per interchange (i.e. EDI component) is also supported for HIPAA components.

The **HIPAA_837D.mfd** file available in the ...\MapForceExamples folder shows an example mapping of an 837-Q2 Health Care Claim: Dental file to an XML Schema target file.

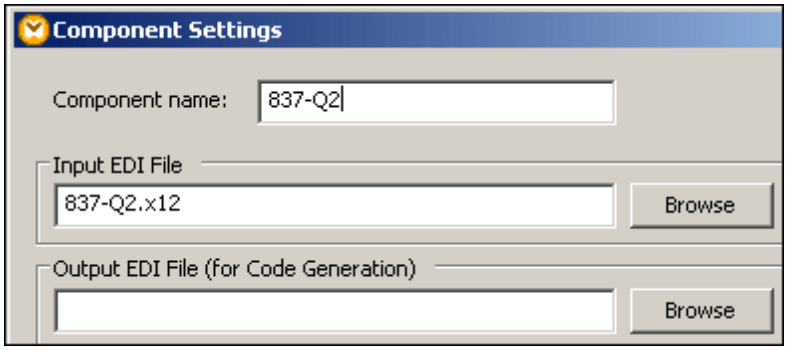


To insert an HIPAA X12 component:

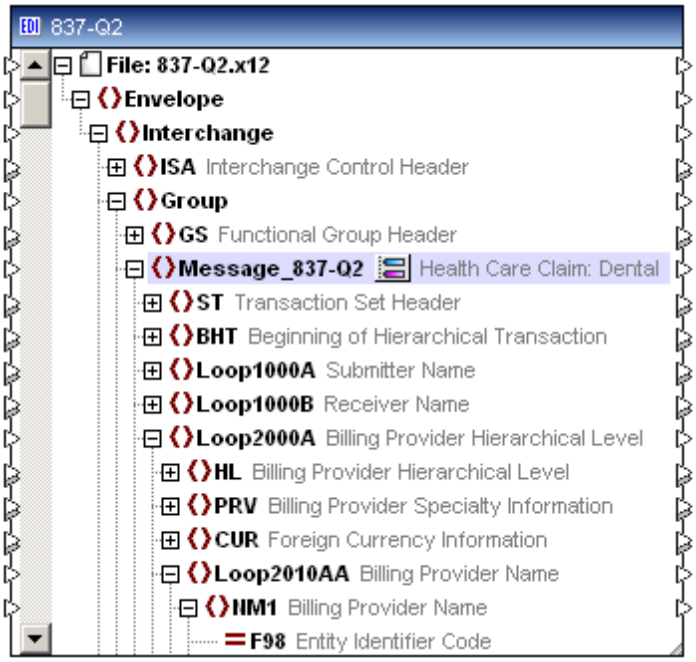
1. Select the menu option **Insert | EDI**.
   This opens the Browse EDI collections dialog box.
2. Click the HIPAA.X12 folder (or the specific HIPAA collection you use) under the ASC X12 entry in the Collection list box.
3. Click the check box of the Message type(s) you want to insert, e.g. 837-Q2: Health Care Claim: Dental, then click OK.



4. Click the Browse button of the following prompt to supply a sample EDI file, e.g.837-Q2.x12.
   The Component Settings dialog box is displayed. (You can adjust the separators at this point if necessary.)



   This file supplies the data for the source component. 837-Q2.x12 is available in the ... \MapForceExamples folder.
5. Click OK to close the Component Settings dialog box and insert the HIPAA component.

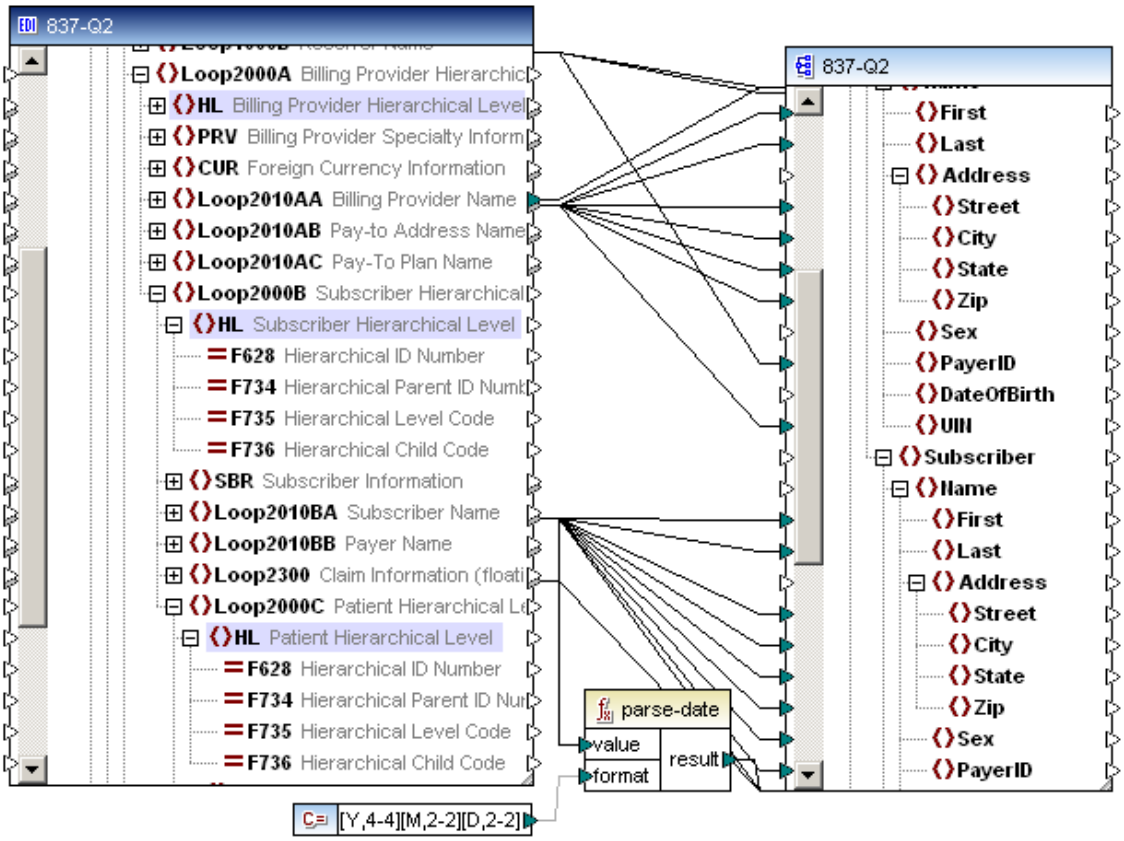


Please note:

the Select EDI message icon , opens the "Browse EDI collections" dialog box, allowing you to select multiple messages per interchange (per EDI component).

7.5.15.1 HIPAA Transactions

837 Dental transaction in MapForce



HL segment hierarchy

HL segments are generated automatically and do need to be mapped manually by the user. It is however possible to map values if there is such a need.

Autocompletion

The configuration files for HIPAA transactions contain many codes and are used for auto-completion of fields whenever possible. Autocompletion is context sensitive, so fields in different places may have different values.

Auto-completion can be disabled by unchecking the "Auto-complete missing fields" check box in the Component Settings dialog box.

Validation

Validation is similar to auto-completion in that it is also context sensitive. Validation errors are handled in the same way as per X12 validation.

Editing of configurations files

HIPAA configuration files are similar to X12 configuration files, with a few new additions:

- Values (code list)
- Conditions

- Completion flags
- “Not Used” Data Elements

Field instances can have a built-in code list which is used for validation and/or autocompletion. Values can be manually added, or removed.

```
<Data ref="F365" info="Communication Number Qualifier" nodeName="F365_1">
<Values>
        <Value Code="EM" />
        <Value Code="FX" />
        <Value Code="TE" />
</Values>
</Data>
```

Conditions

Conditions work in conjunction with values. As there can be multiple repetitions of a segment, or group (loop), they are identified by a specific condtion. A Condition requires a value to be present for it to be fulfilled, otherwise the specific group is not found.

For example, Loop1000 is repeated multiple times, but each instance has a different semantic meaning.

To tell which is which, MapForce uses a condition that specifies that Loop1000A must have a Value Code of “41”, in Field F98 of segment NM1. If this is not the case, then Group name Loop1000A is not found.

```
<Group name="Loop1000A" info="Submitter Name">
  <Segment name="NM1" info="Submitter Name">
    <Condition path="F98" />
    <Data ref="F98" info="Entity Identifier Code">
        <Values>
          <Value Code="41" />
        </Values>
    </Data>
(…)
```

Condition codes are auto generated in the target component, if they contain single value.

Configuration files are generated by a method that ensures that condition values are guaranteed to be unique across the sequence of repeating segments/loops.

If there is a need to edit conditions, or their values, the uniqueness constraint must be taken into account.

Completion flags

Autocompletion can be adjusted on the configuration file level. A new element "Completion", having three attributes, has been introduced to define this:

- singleConditions – auto-complete single conditions for all fields in the target component
- singleValues – auto-complete single values for all fields in the target component
- HL – generate appropriate fields in the HL segment in the target component (for all HL segments)

  (Where "1" means true)

```
<Message>
  <MessageType>837-Q2</MessageType>
  <Completion singleConditions="1" singleValues="1" HL="1" />
  <Description>Health Care Claim: Dental</Description>
(...)
```

"Not Used" Data Elements

HIPAA omits several optional elements that are present in standard X12 transactions. These optional elements are hidden in MapForce components.

The validation engine checks to make sure that these unused fields are not present in source files. Since these fields cannot be mapped to the target component (because they are hidden), they cannot appear in target output files either.

These fields are defined to have the maxOccurs attribute equal to 0, in the configuration files. If necessary, you can manually hide (or unhide) specific fields by using the maxOccurs attribute.

```
<Data ref="F1037" minOccurs="0" info="Submitter Middle Name or Initial" />
<Data ref="F1038" minOccurs="0" maxOccurs="0" />
<Data ref="F1039" minOccurs="0" maxOccurs="0" />
```

Scope of validation

Semantic validation is not supported in MapForce. This means that “situational” fields are simply treated as optional.

7.5.16 TRADACOMS

TRADACOMS (Trading Data Communications) is a UK-specific Electronic Data Interchange standard used in the retail business. It was introduced in 1982 as the first EDI standard for UK trade and industry. Although it has many similarities with the UN/EDIFACT standard, TRADACOMS is one of the precursors of EDIFACT and uses different structures within messages.

MapForce implements the base TRADACOMS specification as laid out in the "TRADACOMS Manual of Standards for Electronic Data Interchange", published in January 1993 by the Article Numbering Association (ANA) UK, now known as GS1 UK (https://www.gs1uk.org). For other TRADACOMS versions, MapForce can be customized to process new message types, data elements, and code values, by means of configuration files.

You can work with the TRADACOMS format in MapForce in the same way as with other supported EDI formats, as follows:

- You can map TRADACOMS files with any other data formats supported by MapForce, in both directions (either as data source or data target).
- Mapping data to or from TRADACOMS format is available in the BUILT-IN language (used when previewing mappings, or in MapForce Server and FlowForce Server execution) and in code generation languages (C++, C#, Java). When you deploy the mapping to a FlowForce Server running on a different machine, the deployed package includes automatically the configurations of selected TRADACOMS message types and all code lists that are used by their data elements. Likewise, in the generated code, MapForce generates classes for the configuration groups, segments, and data elements.
- You can flexibly define which validation events should stop the data conversion, which

ones should result in rejected (or accepted) records, and which ones should be ignored.
- You can enable or disable data auto-completion. When "Automatic data completion" (or "auto-completion") is enabled, MapForce fills some of the values automatically when generating TRADACOMS files.
- You can customize the message types, data elements, and code lists by means of configuration files, either globally or locally (see Configuration Files).
- You can set the encoding of parsed or generated files.

The following TRADACOMS file types and message types are available by default in MapForce:

File Type	Message Type	Version
THE PRODUCT INFORMATION FILE	PROHDR (Product File Header)	8
	PROINF (Product Details)	8
	PROTLR (Product File Trailer)	8
THE PRICE INFORMATION FILE	PRIHDR (Price File Header)	8
	PRIINF (Price Details)	8
	PRITLR (Price File Trailer)	8
THE CUSTOMER INFORMATION FILE	CUSHDR (Customer Information Header)	8
	CUSINF (Customer Information Details)	8
	CUSTLR (Customer Information Trailer)	8
THE ORDER FILE	ORDHDR (Order File Header)	9
	ORDERS (Order Details)	9
	ORDTLR (Order File Trailer)	9
THE PICKING INSTRUCTIONS FILE	PICHDR (Picking Instructions File Header)	4
	PICKER (Picking Instructions File Details)	4
	PICTLR (Picking Instructions File Trailer)	4
THE DELIVERY NOTIFICATION FILE	DELHDR (Delivery File Header)	9
	DELIVR (Delivery Details)	9
	DELTLR (Delivery File Trailer)	9
DELIVERY CONFIRMATION FILE	DLCHDR (Delivery Confirmation Header)	5
	DLCDET (Delivery Confirmation Details)	5
	DLCTLR (Delivery Confirmation Trailer)	5
THE INVOICE FILE	INVFIL (Invoice File Header)	9
	INVOIC (Invoice Details)	9

File Type	Message Type	Version
	VATTLR (VAT Trailer)	9
	INVTLR (Invoice File Trailer)	9
THE CREDIT NOTE FILE	CREHDR (Credit Note File Header)	9
	CREDIT (Credit Note Details)	9
	VATTLR (File VAT Trailer)	9
	CRETLR (Credit Note File Trailer)	9
STATEMENT/REMITTANCE FILE	SRMHDR (Statement/Remittance Details File Header)	9
	SRMINF (Statement/Remittance Line Details)	9
	SRMTLR (Statement/Remittance Details File Trailer)	9
UPLIFT INSTRUCTION FILE	UPLHDR (Uplift File Header)	4
	UPLIFT (Uplift File Details)	4
	UPLTLR (Uplift File Trailer)	4
UPLIFT CONFIRMATION FILE	UCNHDR (Uplift Confirmation Header)	3
	UCNDET (Uplift Confirmation Details)	3
	UCNTLR (Uplift Confirmation Trailer)	3
THE STOCK SNAPSHOT FILE	SNPHDR (Stock Snapshot Header)	3
	SNPSTS (Stock Snapshot Details)	3
	SNPTLR (Stock Snapshot Trailer)	3
THE STOCK ADJUSTMENT FILE	SADHDR (Stock Adjustment Header)	3
	SADDET (Stock Adjustment Details)	3
	SADTLR (Stock Adjustment Trailer)	3
AVAILABILITY REPORT FILE	AVLHDR (Availability File Header)	4
	AVLDET (Availability Report Details)	4
	AVLTLR (Availability File Trailer)	4
GENERAL COMMUNICATIONS FILE	GENHDR (General Communications File Header)	3
	GENRAL (General Communications Text)	3
	GENTLR (General Communications	3

File Type	Message Type	Version
	Trailer)	
COMPLEX ORDER FILE	CORHDR (Complex Order File Header)	6
	CORDER (Complex Order Details)	6
	CORTLR (Complex Order Trailer)	6
THE ACKNOWLEDGEMENT OF ORDER FILE	ACKHDR (Acknowledgement File Header)	4
	ACKMNT (Acknowledgement Details)	4
	ACKTLR (Acknowledgement Trailer)	4
PRODUCT PLANNING REPORT FILE	PPRHDR (Product Planning Report Header)	2
	PPRDET (Product Planning Report Details)	2
	PPRTLR (Product Planning Report Trailer)	2
THE PAYMENT ORDER FILE	PAYHDR (Payment Order File Header)	3
	PAYINF (Payment Order Line Details)	3
	PAYTLR (Payment Order File Trailer)	3
THE DEBIT ADVICE FILE	DRAHDR (Debit Advice File Header)	3
	DRAINF (Debit Advice Line Details)	3
	DRATLR (Debit Advice File Trailer)	3
THE CREDIT ADVICE FILE	CRAHDR (Credit Advice File Header)	3
	CRAINF (Credit Advice Line Details)	3
	CRATLR (Credit Advice File Trailer)	3
THE EXCEPTION CONDITION FILE	EXCHDR (Exception Condition File Header)	3
	EXCINF (Exception Condition Line Details)	3
	EXCTLR (Exception Condition File Trailer)	3
LOCATION PLANNING REPORT FILE	LPRHDR (Location Planning Report Header)	2
	LPRDET (Location Planning Report Details)	2
	LPRTLR (Location Planning Report Trailer)	2
THE UTILITY BILL FILE	UTLHDR (Utility Bill File Header)	3

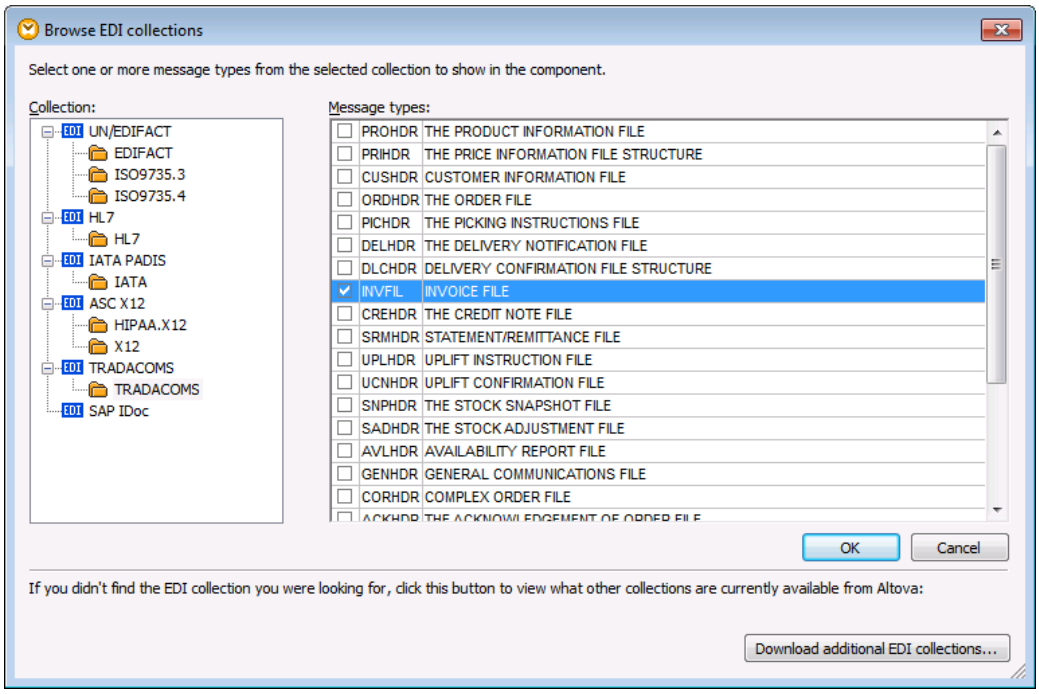
File Type	Message Type	Version
	UTLBIL (Utility Bill File Details)	3
	UVATLR (Utility Bill VAT Trailer)	3
	UTLTLR (Utility Bill File Trailer)	3

Note: The RSGRSG (Reconciliation Message) version 2 is also supported. This message type may occur in all file types.

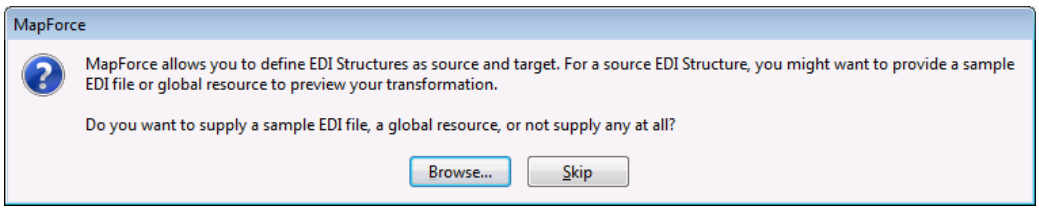
7.5.16.1 Adding TRADACOMS Files as Mapping Components

To add a TRADACOMS file as a data mapping source:

1. Do one of the following:
   - On the **Insert** menu, click **EDI**.
   - Click the **Insert EDI file** ( ) toolbar button.
2. On the Browse EDI Collections dialog box, click **TRADACOMS**, and select the message type(s) to be included in the message.



3. Click **OK**. You are now prompted to supply an instance file.



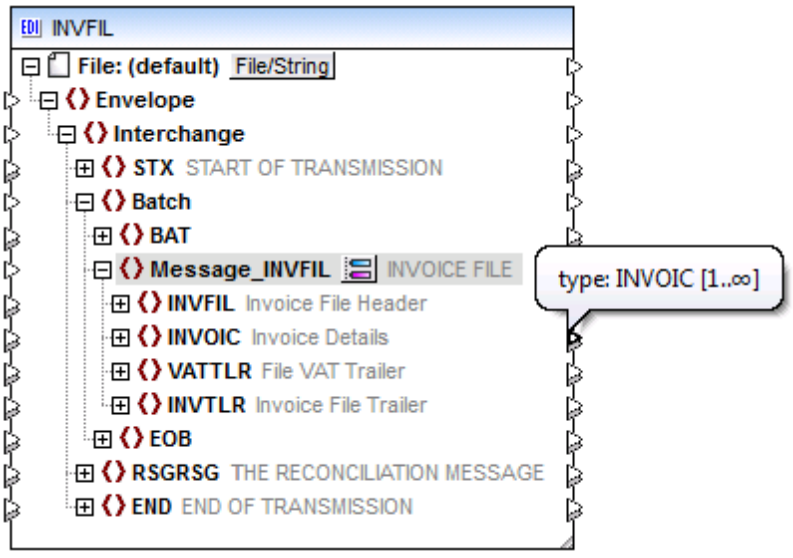
4. If you want to specify the source TRADACOMS file later, click **Skip**. Otherwise, click **Browse** and choose how you would like to open the TRADACOMS file:
   - If you want to open the file from your local drive or network, browse for the file, and then click **Open**.
   - If you want to open the file from a URL, click **Switch to URL** (for more information, see Adding Components from a URL)
   - If you have previously defined the file as a Global Resource, and would like to open it from Global Resources, click **Switch to Global Resources** (see Altova Global Resources).
5. On the Component Settings dialog box, set or change the settings if required (see EDI Component Settings).

To add a TRADACOMS file as a data mapping target:

1. Follow the steps 1 and 2 above.
2. When prompted to supply an instance file, click **Skip**.

7.5.16.2 The TRADACOMS Component in MapForce

In MapForce, the TRADACOMS component replicates the structure of a generic TRADACOMS transmission (with optional batching feature), while retaining all features common to other MapForce EDI-related components.



A sample TRADACOMS component in MapForce

The topmost item of the component, **File**, displays the name of the TRADACOMS interchange file being processed or generated by the component. "Default" indicates that no file has been assigned yet. The **File/String** (File/String) button displays advanced features common to all file-based MapForce components. It provides, among other options, the ability to read or process multiple files dynamically (for more information, see Processing Multiple Input or Output Files

Dynamically).

The "Envelope" and "Interchange" structures are generic for all MapForce EDI-related components. Where applicable, they provide the ability to process multiple interchanges within the same MapForce component.

The STX and END structures are specific to the TRADACOMS format. They denote the "Start of Transmission" and "End of Transmission" segments, respectively.

The BAT and EOB structures are specific to the TRADACOMS format. They denote the "Start of Batch" and "End of Batch" segments, respectively. Such segments are applicable if you are reading data from a TRADACOMS interchange file with batches, or if you want to generate a file with batches. Since there may exist multiple batches in the same transmission, a **Batch** sequence is available as parent of the BAT sequence.

The **Message_*Code*** structure (**Message_INVFIL**, in the sample above) corresponds to the TRADACOMS "file". The **Select EDI Message Types** button ( ) opens a dialog box where you can change the type of the file (for example, "Invoice File", "Orders File", and so on). By virtue of the existing MapForce EDI functionality, you can also include multiple file types in the same component. Note, however, that the TRADACOMS specification recommends one file and one type of file in each interchange transmission.

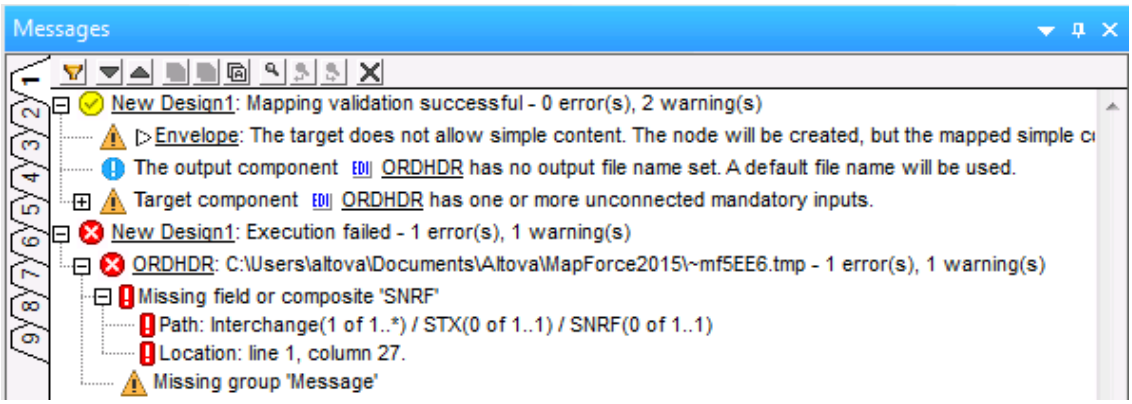
Any TRADACOMS file ("Invoice File", in this example) contains a Header Message, one or more Details Messages, and a Trailer Message. In the component above, these are the INVFIL (Invoice File Header), INVTLR (Invoice File Trailer), the VATTLR (File VAT Trailer) and multiple INVOIC (Invoice Details) messages.

As with any other MapForce component, the input and output connectors (small triangles) displayed on each side of the component provide the ability to map each individual data element or segment to or from other data types or formats supported by MapForce. Likewise, when you move the mouse over any item on the component, you can view additional information about it (such as the minimum and maximum allowed occurrences), provided that tips are enabled from the **View | Show Tips** menu.

Finally, you can change these and other settings by double-clicking the component head and opening the Component Settings dialog box (see EDI Component Settings). Note that this dialog box is generic for all EDI-related component types and, thus, some of the options might not be applicable to TRADACOMS.

7.5.16.3 Validation and Automatic Data Completion

When you run a mapping that reads data from or writes data to a TRADACOMS structure, MapForce performs structural data validation checks according to the TRADACOMS specification, and displays any validation errors in the Messages window.

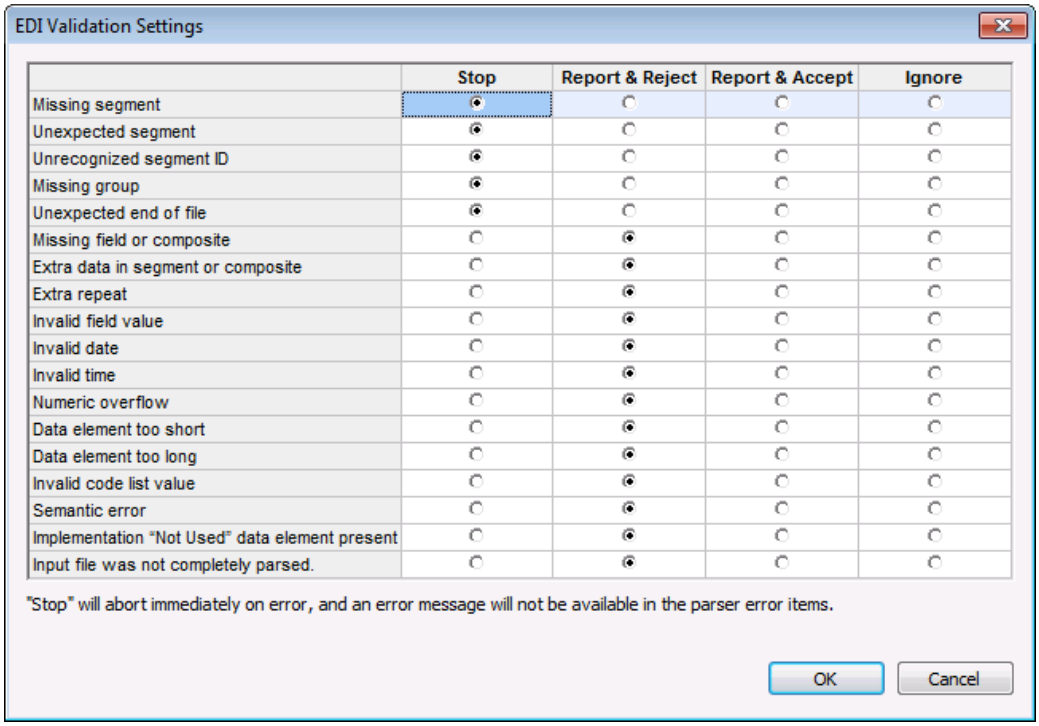


Messages window with validation errors

As shown in the sample above, the validation messages specific to TRADACOMS (information, warnings, and errors) are displayed in addition to the generic validation messages common to any MapForce mapping. For more information about MapForce validation in general, see Validating Mappings.

The following factors determine how MapForce validates the parsed or generated TRADACOMS files:

- The validation constraints defined in the configuration files available in the MapForce installation folder (subfolder **MapForceEDI\TRADACOMS**). These configuration files supply, on one hand, the default validation rules of the TRADACOMS specification. On the other hand, they provide the means to adapt the TRADACOMS format to custom requirements. In particular, it is possible to modify the data elements, segments, or code values defined in the configuration files, and thus influence both the outcome of validation and the mapping execution. For more information about the configuration files, see Configuration Files.
- The validation logic built into MapForce. This includes MapForce internal data integrity checks that may not be enforced by means of configuration files.
- Any custom validation settings that you have defined from the MapForce graphical user interface (see EDI component validation). To view or change the current EDI validation settings of any EDI component, including TRADACOMS, double-click the header of the component, and then click **Validation** on the Component Settings dialog box.



When writing to a TRADACOMS structure, MapForce automatically fills in the contents of those data elements for which the value can be calculated or is predefined. This is referred to as "automatic completion" (or "auto-completion"). To disable this behavior, clear the **Auto-complete missing fields** check box from the Component Settings dialog box (see EDI Component Settings).

The following TRADACOMS validation rules cause MapForce to either raise validation errors (during file parsing or generation) or to auto-complete missing fields (during file generation):

1. The segments STX (Start of Transmission) and END (End of Transmission) must exist.
2. If STDS-1 has the value 'ANAA' then a Reconciliation Message (RSGRSG) must exist before the end of transmission (END). Otherwise, no Reconciliation Message (RSGRSG) must be present.
3. If STDS-1 has the value 'ANAA' then:
   a. The value of RSGA in the Reconciliation Message must be equal to the value of SNRF in the STX segment
   b. The value of RSGB in the Reconciliation Message must be equal to the value of UNTO-1 in the STX segment.
4. TRDT-1 must contain the date (YYMMDD) and TRDT-2 must contain the time (HHMMSS) of transmission (current date and time).
5. If the Batch Header (BAT) is present then the Batch Trailer (EOB) must also be present, and the number of messages in the batch must be available in the NOLI (Number of Messages in Batch) data element.
6. The MSRF (Message Reference) data element in the Message Header (MHD) must contain the consecutive count of messages within the transmission, starting with 1.
7. The NOSG (Number of Segments in Message) data element in the Message Trailer (MTR) must contain the number of segments, including MHD and MTR.
8. When present, the Reconciliation Message (RSGRSG) must consist of one segment (RSG),

except the Message Header and the Message Trailer.

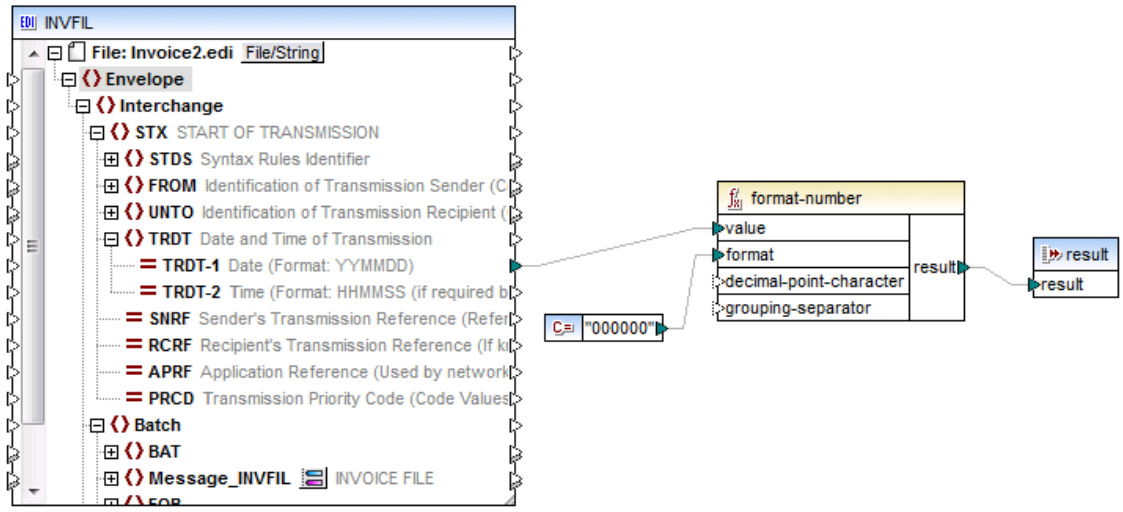
9. The NMST (Number of Messages in Transmission) data element in the END segment must contain the number of messages in interchange (count of MHD segments).
10. In general, when reading a TRADACOMS structure, MapForce expects that the interchange environment is of type "computer to computer" (or, in TRADACOMS terminology, "intelligent terminal to intelligent terminal"). Therefore, a segment such as **MHD = 12 + ORDHDR :3** would trigger a validation error, since it contains extra leading and trailing spaces.
11. String data must be in upper case. When generating TRADACOMS output, MapForce converts string data to upper case.

Additionally, as already mentioned, any validation rules defined in the configuration files will also affect TRADACOMS parsing and generation.

7.5.16.4 Preserving Leading Zeros During Conversion

In the TRADACOMS specification, the data type of the TRDT-1 and TRDT-2 (Date and Time of Transmission) data elements is defined as "decimal". This means that, by default, when you map data from the TRDT-1 or TRDT-2 fields, any leading zeros in these fields would be trimmed during conversion, and thus produce undesired results. This may also be the case of any other items which are defined as "decimal", but store values which are meant to be treated as string.

You can instruct MapForce to treat numeric fields as string (and thus preserve the leading zeros) by applying the **core | format-number** MapForce function. In the following example, the value of the TRDT-1 item in the source file is "020312". The normal output of this item would be "20312" (as a result of conversion to decimal), which is not the desired result. Therefore, to keep the leading zero, the **format-number** function has been added to the mapping. (For information about working with functions, see Working with Functions.)



Preserving leading zeros with the format-number function

As illustrated above, the function has the following two input arguments:

1. The value to be formatted (in this case, "020312").

2. The format mask "000000".

To test the output of the function, this example uses a simple output component (see Simple Output). When you click the **Output** button, the output of the mapping (which is the same as the result of the function) is "020312", as intended.

7.5.16.5 Configuration Files

By default, the MapForce configuration files and the underlying MapForce processing logic reflects the base version of the TRADACOMS standard (931). Therefore, if you use this TRADACOMS version, customization of the TRADACOMS configuration files is not necessary. However, if you want to make changes to the existing configuration for any reason, it is possible to do so. Note that any changes made to the standard TRADACOMS configuration files will take immediate effect and will affect the way MapForce parses or writes TRADACOMS structures.

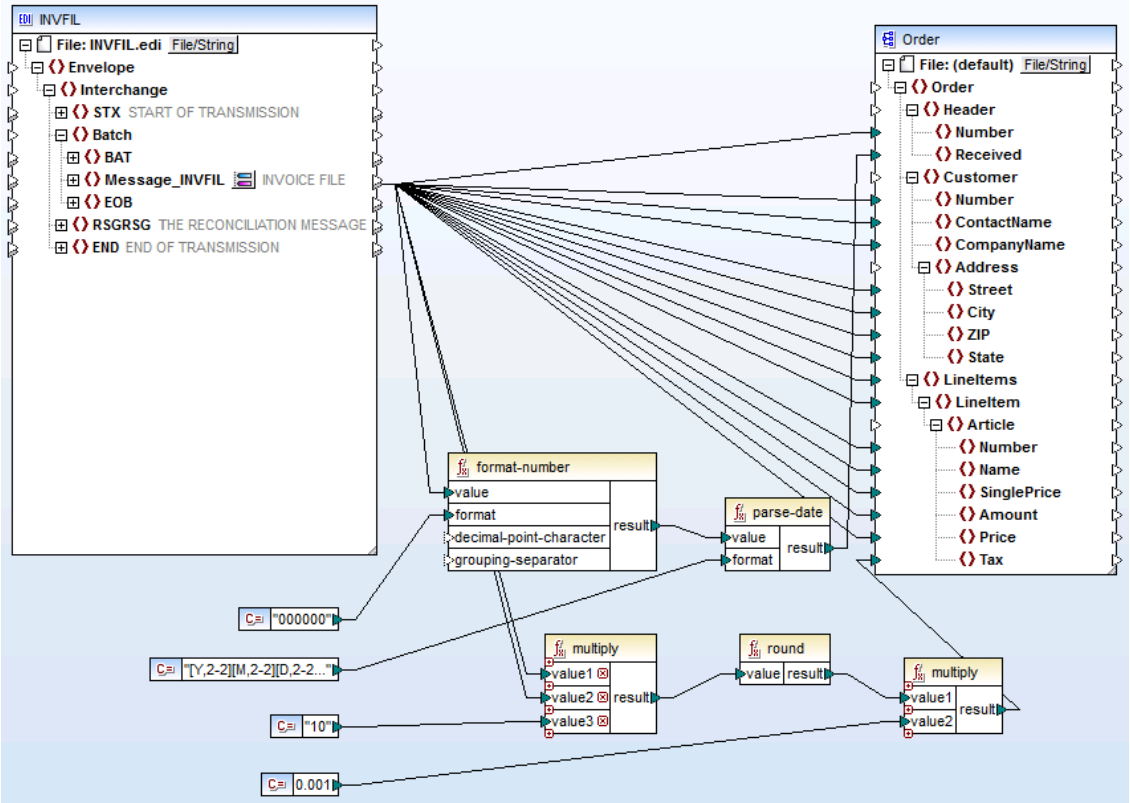
The MapForce configuration files applicable to TRADACOMS are stored in the following directory: **C:\Program Files\Altova\MapForce2016\MapForceEDI\TRADACOMS**. Should you need to make changes to the TRADACOMS configuration files, you can start by creating a sibling copy of the TRADACOMS directory, for example "TRADACOMS1", edit the directory permissions not to be "read-only", and then edit the files in it according to your custom requirements. All configuration files adhere to the XML syntax and, so it is recommended that you use an XML editor to modify the files.

File	Purpose
EDI.Collection	The **EDI.Collection** file contains a list of all messages in the current TRADACOMS configuration directory (for example, **C:\Program Files\Altova\MapForce2016\MapForceEDI\TRADACOMS**). Any messages listed in the collection file become available for selection in the "Browse EDI Collections" dialog box when you add a TRADACOMS component to the mapping (**Insert | EDI**). Therefore, if you want to suppress any message type on this dialog box, comment out or remove their respective XML element from the EDI.Collection file. For example, commenting out the following line makes the LPRHDR message type unavailable: `<!--<Message Type="LPRHDR" File="LPRHDR.Config" Description="LOCATION PLANNING REPORT FILE"/>-->`
Envelope.Config	Defines the structure of the TRADACOMS component in MapForce (that is, elements that are surrounded by the **Envelope** sequence). This file is validated against a schema handled by MapForce internally, and it should not be edited.
Tradacoms.Codelist	This file defines the TRADACOMS codes and the values that they may contain, and is used for validation of input/output TRADACOMS files. You can add custom code lists to this file, if

File	Purpose
	required.
Tradacoms.Segment	This file defines the Segment, Composite and Field names of the TRADACOMS files, and is used when parsing a TRADACOMS file. Changes made to this file are global customizations, and apply to all segments and messages.
[Message].Config	Defines the structure of a single TRADACOMS message. Changes made to this file are treated as local (inline) customizations.

7.5.16.6 Example: Converting a TRADACOMS Invoice to XML

This example provides instructions on how to create a mapping design that converts data from a TRADACOMS invoice file to XML format. The mapping design file created in this example is also available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\TRADACOMS_Invoice.mfd**.



TRADACOMS_Invoice.mfd

The mapping accomplishes the following goals:

1. Convert a source TRADACOMS file (**INVFIL.edi**) to an XML file which is valid against an

existing schema (**Order.xsd**). Both the source TRADACOMS file and the schema file (**Order.xsd**) are available in the MapForce Examples folder (**<Documents>\Altova \MapForce2016\MapForceExamples**).

2. The date in the TRADACOMS file has `decimal` format so it must be converted to `xs:dateTime` format in the target XML file.
3. For each line item in the purchase order, the tax amount must be calculated as a decimal value, rounded to three decimal places (thousandths). For the scope of this example, we will assume that the tax is calculated according to the following formula:

```
Tax = round(LEXC * VATP * 10) * 0.001
```

Where **round** rounds the value to the nearest integer.

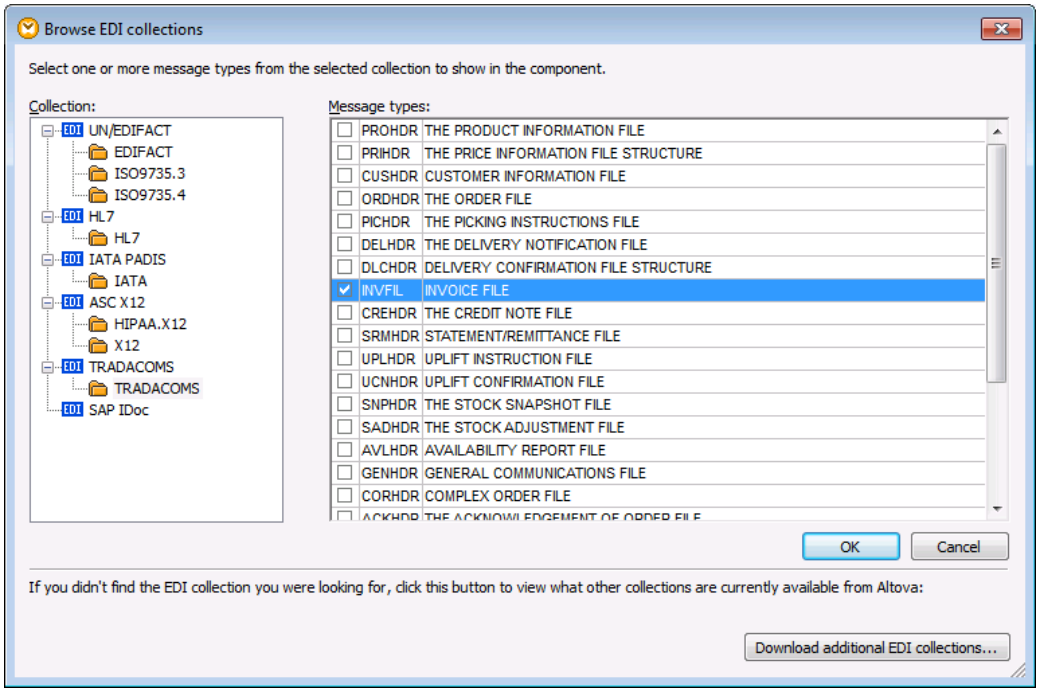
To accomplish the goals, take the following steps:

1. Add the source and target components to the mapping area and draw the mapping connections between them.
2. Use date functions to convert the date to the required format.
3. Use math functions to calculate the tax value.

Step 1: Add the components to the mapping area

To add the source TRADACOMS component:

1. On the **Insert** menu, click **EDI**.
2. Click the TRADACOMS collection folder, and, under Message Types, select the **INVOICE FILE**.



3. When prompted to supply a sample file, browse for the **INVFIL.edi** available in the

MapForce Examples folder (**<Documents>\Altova\MapForce2016\MapForceExamples**).

4. When prompted to enter the component settings, leave the default values as is, and click OK.

To add the target XML component:

1. On the **Insert** menu, click **XML Schema/File**.
2. When prompted to supply a sample file, click **Skip**. (Because this is a target component, the file will be generated, so there is no need to browse for an existing one.)
3. When prompted to select a root element, select "Order".

In this example, the data will be mapped from the source TRADACOMS file to the target XML file as follows:

Source (TRADACOMS)	Target (XML)
INVN	/Order/Header/Number
IVDT*	/Order/Header/Received
CIDN-2	/Order/Customer/Number
CADD-1	/Order/Customer/ContactName
CNAM	/Order/Customer/CompanyName
CADD-2	/Order/Customer/Address/Street
CADD-3	/Order/Customer/Address/City
CADD-5	/Order/Customer/Address/ZIP
CADD-4	/Order/Customer/Address/State
Group_IRF**	/Order/LineItems
Group_ILD**	/Order/LineItems/LineItem
SPRO-1	/Order/LineItems/LineItem/Article/Number
TDES-1	/Order/LineItems/LineItem/Article/Name
AUCT-1	/Order/LineItems/LineItem/Article/SinglePrice
QTYI-1	/Order/LineItems/LineItem/Article/Amount
LEXC	/Order/LineItems/LineItem/Article/Price
LEXC*, VATP*	/Order/LineItems/LineItem/Article/Tax

* Source data will be processed with functions before being written to the target
** These fields enable iteration through a group of fields of the same type

Using the table above as reference, you can now draw mapping connections from all of the above

items in the source component to their destination in the target component. For now, you can omit those fields that require calculated values (that is, the ones marked with * in the table); these will be handled in the next steps. For general information about drawing connections in MapForce, see Working with Connections.

Tip: To search for a field by its name inside the MapForce component, press **Ctrl + F**.

Step 2: Add the date conversion functions

To convert the value of the `IVDT` item from a decimal value to `xs:dateTime` type, the mapping uses the **`format-number`** function from the **`core | conversion functions`** library. For a worked example, see Preserving Leading Zeros During Conversion.

The role of the second function, **`parse-date`**, is to convert the YYMMDD string value returned by the **`format-number`** to `xs:dateTime` type. It has two input arguments: (a) the value to be formatted and (b) the format mask. The format mask essentially instructs MapForce to treat the year, month, and date as having a width of exactly two characters each.

To add the functions above to the mapping, drag them from the Libraries window to the mapping area. To supply an argument to a function, insert a constant (the menu command is **Insert | Constant**). For general information about working with functions in MapForce, see Working with Functions.

Step 3: Add the tax calculation functions

The functions used to calculate tax according to the formula are **`multiply`** and **`round`**, available in the **`core | math functions`** library. The multiply function is extendable (that is, it can take a variable number of arguments). In this example, it takes `LEXC`, `VATP`, and the integer 10 as arguments. The first two values are connected from the source TRADACOMS component, while the integer is supplied by a constant. The result of the **`multiply`** function is then rounded to the nearest integer using the **`round`** function. Finally, the result of the **`round`** function is multiplied by 0.001 by means of a second **`multiply`** function. This result is then connected to the destination item of the XML component (`/Order/LineItems/LineItem/Article/Tax`).

To preview the output of the mapping design, click the **Output** tab in the lower part of the mapping window.

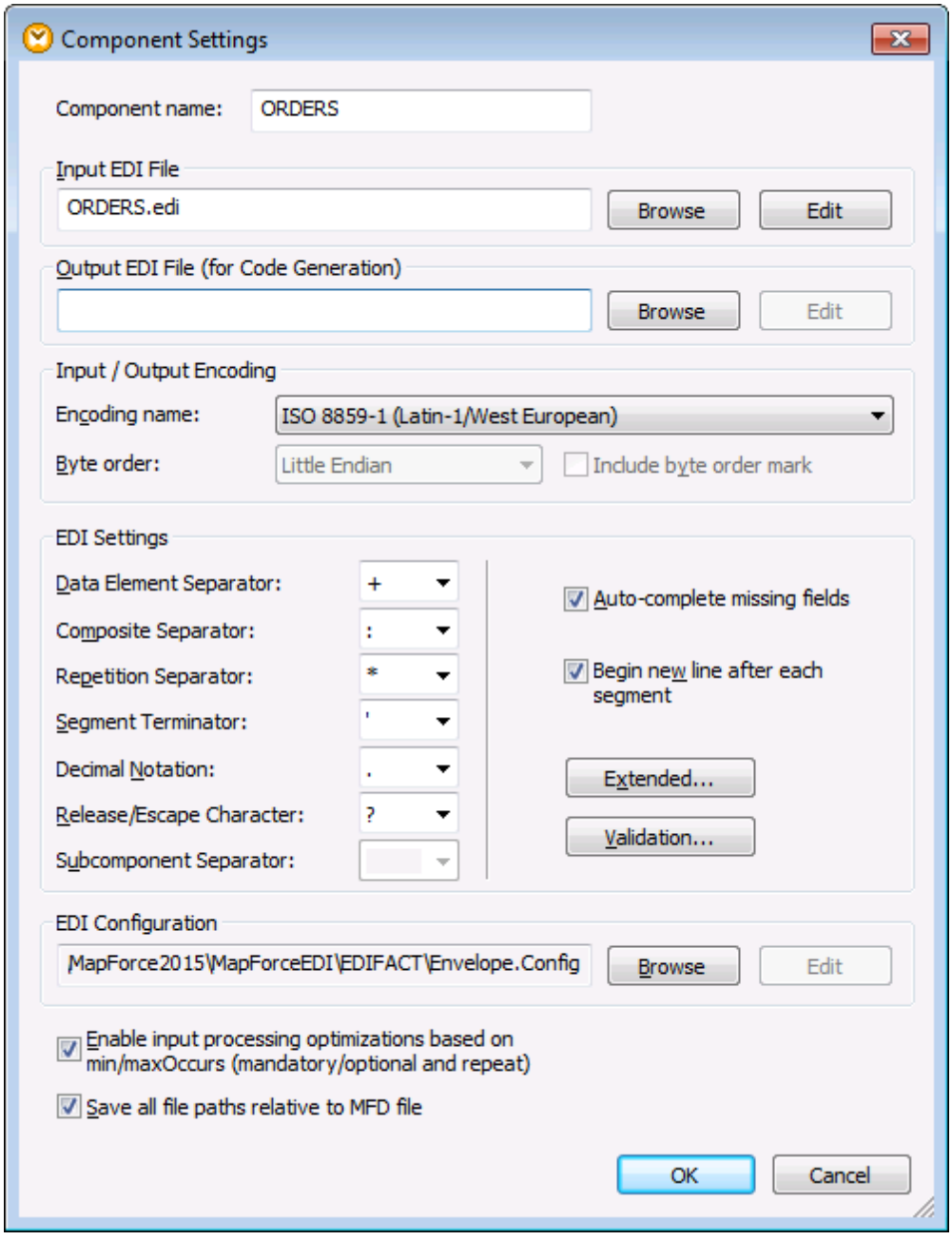
Step 4 (optional): Style the mapping output with StyleVision

Optionally, you can link the target XML component to a StyleVision Power StyleSheet (.sps) file. (For more information about StyleVision, see http://www.altova.com/stylevision.html.) This would enable you to generate the mapping output as HTML, RTF, PDF, or 2007+ file (provided that StyleVision is installed and the prerequisites required by each format are in place, see Styling Mapping Output with StyleVision). To take this optional step, double-click the header of the XML component, and, next to "StyleVision Power StyleSheet file", browse for the **Order.sps** file available in the MapForce Examples folder. Back on the mapping area, you can now click the HTML, PDF, RTF, and Word 2007+ tabs to view the mapping output in the corresponding format.

7.5.17 EDI Component Settings

After you add an EDI component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component on the mapping, and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.



EDI Component Settings dialog box

The available settings are listed below. Note that some settings are not available if the EDI flavour of the currently selected component does not support them.

Component name	The component name is automatically generated when you create the component. You can however change the name at any time.

	If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well. The component name can contain spaces (for example, "Source XML File") and full stop characters (for example, "Orders.EDI"). The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications when changing the name of the component: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have different encodings in Windows and at the command line.
Input EDI file	Specifies the EDI source file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it an EDI instance file.
Output EDI file	Specifies the EDI target file to which MapForce will write data. This field is meaningful for a target component.
Input/Output Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab.
EDI Settings	This group of settings enable you to define custom EDI delimiters, separators and terminators (note the settings are available only if supported by the EDI format).
Auto-complete missing fields	This option applies to target EDI components. When this check box is selected, MapForce fills in the values of some data fields automatically. This applies only to those fields where this operation would not contradict the specification of the currently selected EDI format. To disable such behaviour, clear this check box.
Begin new line after each segment	This option applies to target EDI components. When the check box is selected, MapForce adds a CR/LF (carriage return / line feed) character after each EDI segment.
Extended	Opens a dialog box where you can define additional settings

	for the current EDI component. The available settings vary by EDI flavour.
Validation	Opens a dialog box where you can define the validation settings for the current EDI component (see EDI component validation).
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**, clear 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. This setting affects the following files: • The input EDI file (if present) • The output EDI file (if present) See also Using Relative Paths on a Component.

7.6 JSON

You can add JSON (JavaScript Object Notation) instance and schema files as source or target components on a mapping. MapForce reads and writes JSON files based on the JSON Draft 04 Schema (http://tools.ietf.org/html/draft-zyp-json-schema-04).

When you add a JSON file to a mapping, MapForce checks if the file has a `$schema` property, in order to determine whether it is a schema or instance file. When the property is present and it is either "http://json-schema.org/schema#" or "http://json-schema.org/draft-04/schema#", the file is loaded as a schema. Otherwise, MapForce prompts you to indicate if the file is a schema or an instance. If you confirm that the loaded file is a JSON instance file, MapForce asks you either to browse for a schema or generate it automatically. MapForce uses the JSON schema to build the structure of the component.

MapForce recognizes JSON schema files with the **schema.json** extension (for example, **Example.schema.json**). At the time when this documentation is written, there is no formal convention for naming JSON schema files.

A JSON component in MapForce normally has the structure of a JSON instance file. In other words, the structure of nodes in the JSON component reflects the structure of the JSON instance file. The basic JSON types are conventionally represented in MapForce JSON components as shown below.

Type	MapForce representation
Array	[]
Boolean	01
Null	∅
Number or integer	#
Object	{}
String	"AB"

There are also special cases, as follows.

Case	Description
additionalProperties, patternProperties	The {▫} node appears on the JSON component under any object whose `additionalProperties` property is `true` or not present in the schema. It allows you to map to or from properties not explicitly listed in the schema (see also Example: Mapping from JSON to CSV). This node can also appear for objects having the `patternProperties` property.
Subtypes	JSON schema allows defining subtypes for objects and arrays

Case	Description
	(`anyOf`, `allOf`, `oneOf`). MapForce displays such subtypes using special structure nodes ( ) that do not have a direct representation in the JSON instance file. If your mapping is reading from a JSON file, such subtype nodes provide a value only if the current input value is valid according to the subtype schema. If your mapping is writing to a JSON file, make sure that you choose the correct subtype to fill. Filling multiple subtypes may lead to duplicate object properties and, thus, may result in invalid output JSON files.
Multiple types at the same location	JSON schema allows multiple types to occur at the same location. In such cases, the MapForce component displays separate structure nodes for all basic types that can occur at that location.
Type names	MapForce displays the `title` and `description` properties of types in the JSON schema in the "type" and "annotation" fields, if available. If `title` is absent, MapForce may also use part of the URI from the `$ref` property as a type name.
Arrays containing mixed item types	If an array has different types of items in the JSON schema (for example, both strings and numbers), MapForce displays an "item" node for each item type. When writing to a JSON file, this enables you to create arrays which contain items of different types.
Arrays defined as tuples	If an array has items whose type is assigned by position in the JSON schema, MapForce displays the zero-based index of the item as separate structure nodes (for example **item[0]**, **item[1]**, and so on. When writing to a JSON file, this enables you to specify the type of each individual item in the array by its zero-based index.

JSON support notes

To use JSON components in mappings, it is required to set the target language to "Built-in" (see Selecting a transformation language). JSON components cannot be used in any of the code generation languages (C#, C++, Java). Also, JSON components do not currently support the following:

- Complex parameters in user-defined functions
- Intermediate variable components.

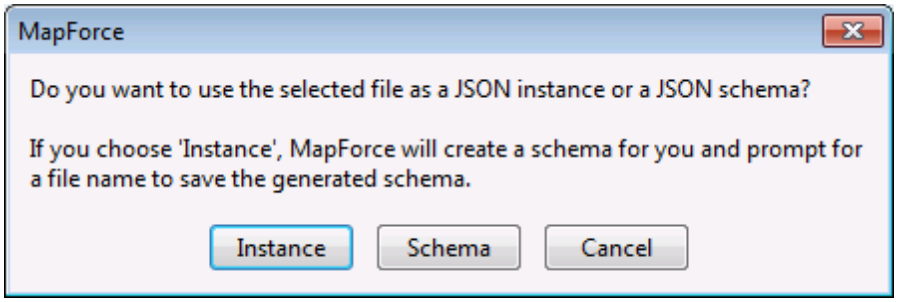
See also:

- Adding JSON Files as Mapping Components
- JSON Component Settings
- Example: Mapping from JSON to CSV

7.6.1 Adding JSON Files as Mapping Components

To add JSON files as mapping components

1. On the **Insert** menu, click **JSON Schema/File**. A dialog box prompts you to select the type of the JSON file (schema or instance).

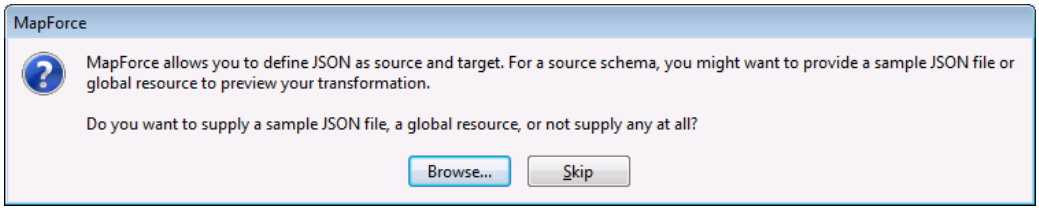


2. Select **Schema** or **Instance**, as required.

> The size of the JSON instance file must not exceed your system's available memory.

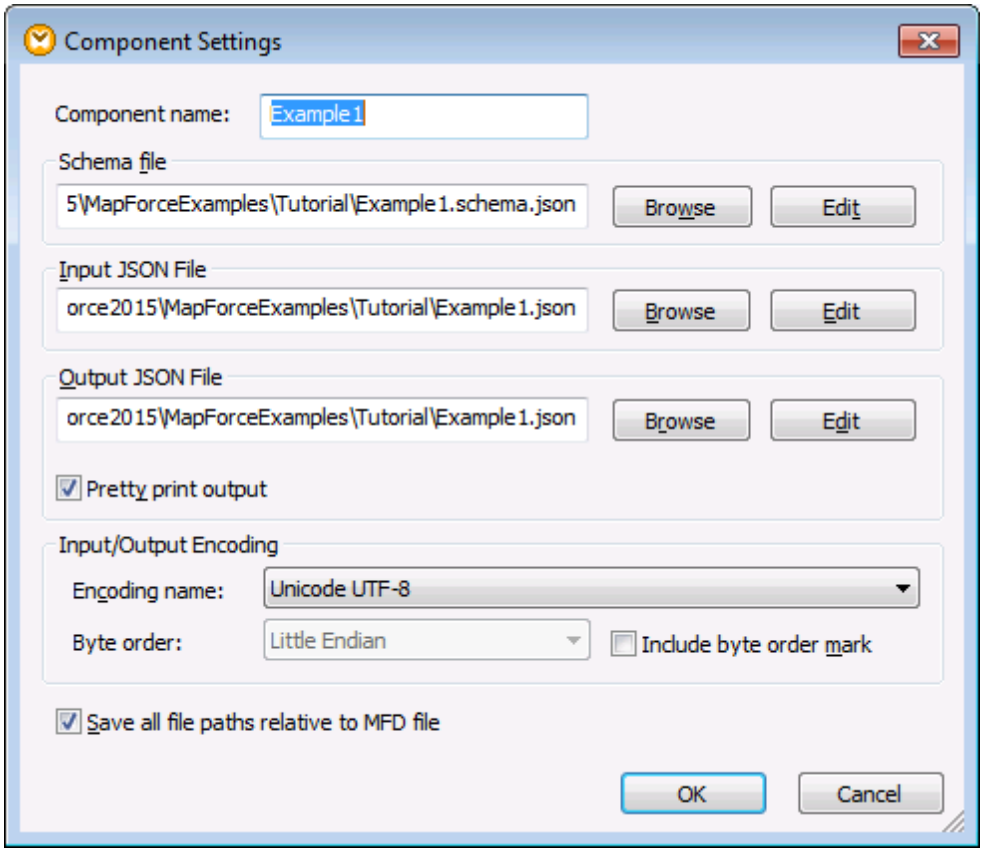
If you select **Instance**, MapForce generates automatically the schema based on the instance file, and prompts you for a location to save the schema.

If you select **Schema**, MapForce prompts you to specify a sample JSON file or a global resource (see Global Resources). This is required to preview the transformation.



7.6.2 JSON Component Settings

To change the settings of a JSON component, right-click the JSON component header, and then click **Properties** (alternatively, double-click the JSON component header).



JSON Component Settings dialog box

The available settings are as follows.

Component name	Allows you to change the display name of the JSON component.
Schema file	Specifies the file name and path of the schema file. To change the location of the file, click **Browse** and select the new file. To edit the file in your JSON editor (for example, XMLSpy), click **Edit**.
Input JSON File	Specifies the file name and path of the input JSON instance. To change the location of the file, click **Browse** and select the new file. To edit the file in your JSON editor (for example, XMLSpy), click **Edit**.
Output JSON File	Specifies the file name and path where the JSON target

	instance file is placed, when the mapping is executed by MapForce Server. This is also the default location when you save the output from the **Output** tab. To change the location of the file, click **Browse** and select the new file. To edit the file in your JSON editor (for example, XMLSpy), click **Edit**.
Pretty print output	Reformats your JSON document when the mapping is executed, in order to give it a structured display. Each child node is offset from its parent by a single tab character.
Encoding	Allows you specify the following settings of the output instance file: • Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. JSON files are expected to have UTF encoding (see http://tools.ietf.org/html/rfc7159#section-8.1). Other encodings are considered non-standard.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the following files: • The JSON schema file • The JSON input file • The JSON output file See also Using Relative Paths on a Component.

7.6.3 Example: Mapping from JSON to CSV

This example illustrates a mapping that converts data from a JSON instance file to a comma-separated text file. It also explains how to map values from additional properties that might be present in the JSON instance file, but not defined in the schema.

The JSON schema used in the example is represented below. As the `$schema` keyword indicates, the schema validates JSON instances with respect to Draft 04 JSON Schema. It describes a `people` array which consists of multiple `person` objects. The `people` object must contain at least one `person` object to be valid. Each `person` object has a `name`, `age`, and `email` address as properties. Note that `name` and `email` are of type `string`, while `age` is of type `integer`. Also, the

`name` and `email` properties are required, while `age` is optional.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "people",
  "type": "array",
  "items": {
       "title": "person",
       "type": "object",
    "required": [
      "name",
      "email"
    ],
    "properties": {
      "name": {
        "type": "string"
      },
      "email": {
        "type": "string",
        "format": "email"
      },
      "age": {
        "type": "integer"
      }
    }
  },
  "minItems": 1
}
```

The JSON instance file contains the people records that must be converted to CSV, as follows:

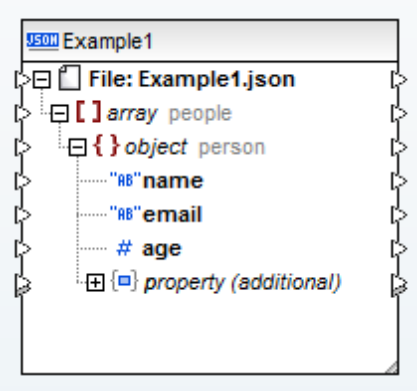
```
[
       {
              "name": "Alethia Alonso",
              "email": "altethia@example.com",
              "age": 35,
              "birthday": "4 July"
       }, {
              "name": "Klaus Mauer",
              "email": "klaus@example.com",
              "age": 57,
              "birthday": "31 August"
       }, {
              "name": "Natsuo Shinohara",
              "email": "natsuo@example.com",
              "age": 29
       }
]
```

The highlighted text shows that the first and the second person have an additional property that is not defined in the schema, namely `birthday`. Nevertheless, the JSON instance is valid, since the schema does not contain an `additionalProperties` property for the `person` object. When it is not present in the JSON schema, the `additionalProperties` property has the default value of **true**, which means that the object in the JSON instance can have as many additional properties

as required, and still be valid.

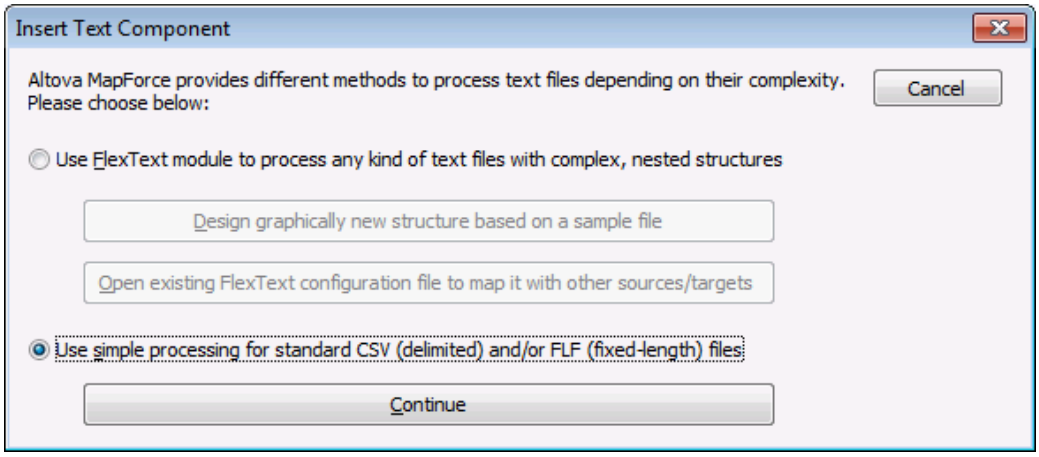
To create a JSON to CSV mapping

1. On the **Insert** menu, click **JSON Schema/File** and browse for the **Example1.schema.json** file available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, under "My Documents" folder. When prompted to specify an instance, browse for the **Example1.json** file. At this point, the MapForce component looks as follows:

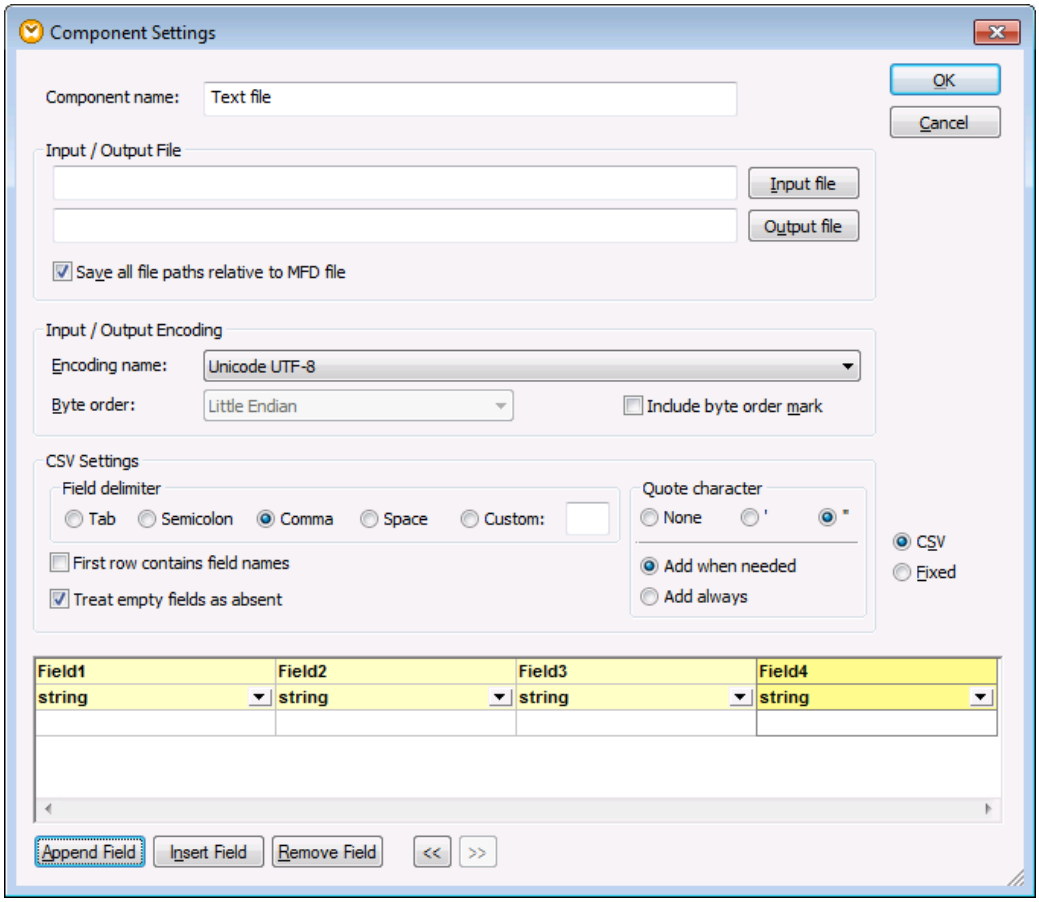


The structure of the MapForce component resembles that of the JSON file itself, with the exception of **property (additional)** node. This node indicates that the `additionalProperties` property of the `person` object is either missing or set to **true** in the schema. This means that the schema can contain custom additional properties, so MapForce displays the node in case you want to map from any additional properties of the object (the next steps show how to do this).

2. On the **Insert** menu, click **Text File**.

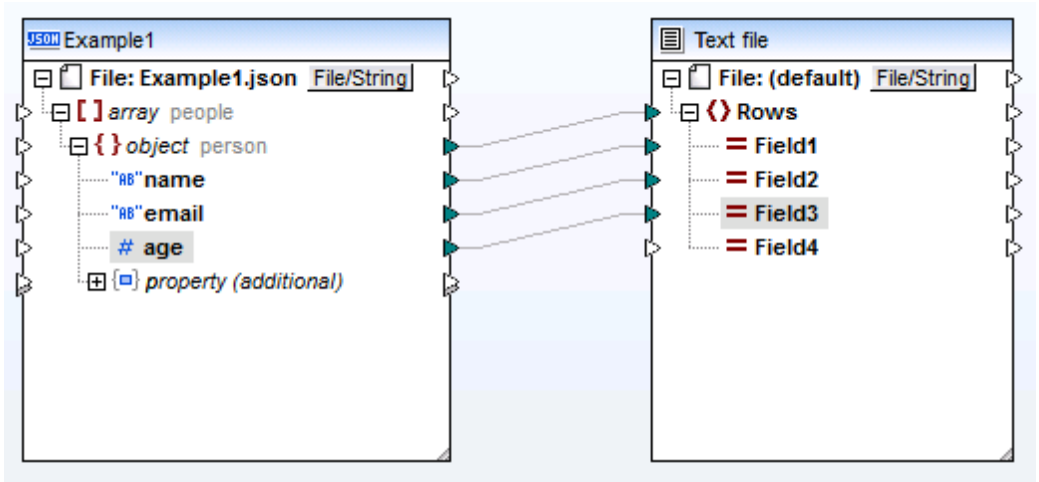


3. Select **Use simple processing for standard CSV**, and then click **Continue**.



4. Click **Append Field** four times to add four CSV fields on the text component, and then click **OK**. Three fields will map to the `name`, `email`, and `age` properties of the `person` object, and the fourth field will map to the `birthday` additional property.

5. On the mapping pane, connect the `person` object from the JSON component to the `Rows` node of the CSV component. Also, connect the `name`, `email`, and `age` properties of the person object to the first three fields on the text component. The mapping now looks as follows:



Optionally, if you give to CSV fields the same name as the object properties (that is, *name*, *email*, and *age*), you can use the **Auto-connect matching children** option to make all connections automatically (see Connecting matching children).

If you preview the transformation output at this time by clicking on the **Output** tab, the result is:

```
Alethia Alonso,altethia@example.com,35,
Klaus Mauer,klaus@example.com,57,
Natsuo Shinohara,natsuo@example.com,29,
```

As the output shows, for each person object in the JSON file, a new row is created, and object properties are comma-separated, which is the intended behavior.

However, we have not mapped yet the `birthday` additional property which exists in the JSON instance file, even if it does not exist in the schema.

To map from the additional property of a JSON object

1. Expand the **property (additional)** node under the `person` object. Notice the **name** item. This item allows you to access the additional property by its name. In our example, the name of the additional property is "birthday" and the type is "string".

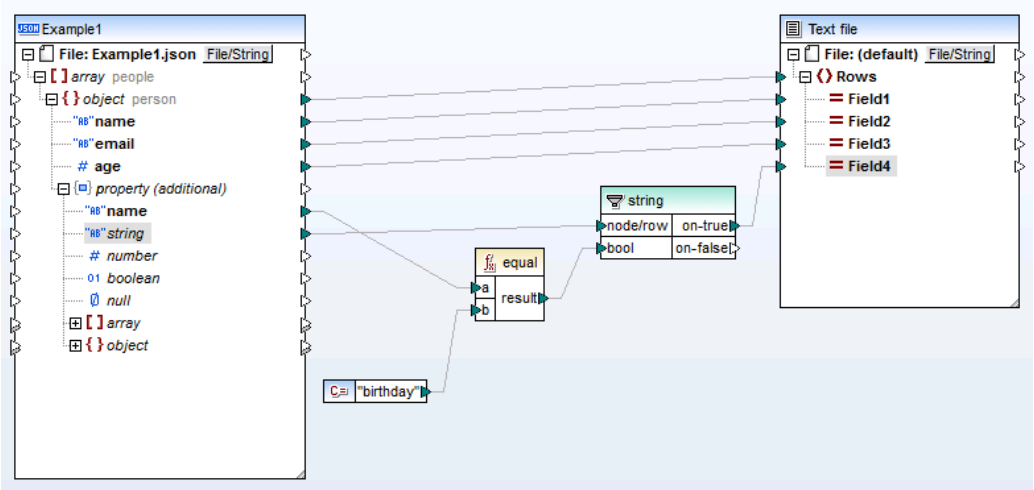
2. Insert the following onto the mapping area:

   a. A constant with the value "birthday".
   b. A "Filter: Nodes/Rows" component.
   c. The logical function **`equal`**.

3. Connect the components as follows:

   - From the name of the additional property to the "a" input of the equal function.
   - From the **birthday** constant to the "b" input of the **`equal`** function.
   - From the output of the **`equal`** function to the **bool** input of the "Filter: Nodes/Rows" component.
   - From the **string** item of the additional property to the **node/row** input of the "Filter: Nodes/Rows" component.
   - From the **on-true** output of the "Filter: Nodes/Rows" component to **Field4** of the Text component.

   Connecting components this way instructs MapForce to look for an additional property with the name **birthday**, and, if a string value is found, copy it to **Field4** of the text component.



If you preview the transformation output at this time, the result is:

```
Alethia Alonso,altethia@example.com,35,4 July
Klaus Mauer,klaus@example.com,57,31 August
Natsuo Shinohara,natsuo@example.com,29,
```

As the output shows, the fourth CSV field now includes the value of the only additional property of type "string", which is, in this case, `birthday`. Also, since the third person does not have a birthday, no value is available in the CSV at the corresponding position.

You can use the same technique to map any additional properties that are present in your JSON instance. As a general rule, it is recommended that your schema should include all properties that you intend to map. However, in the event that the JSON instance contains properties that are data-dependent (not fixed), use this technique.

7.7 Microsoft OOXML Excel 2007+

Altova web site: Mapping to/from Excel files

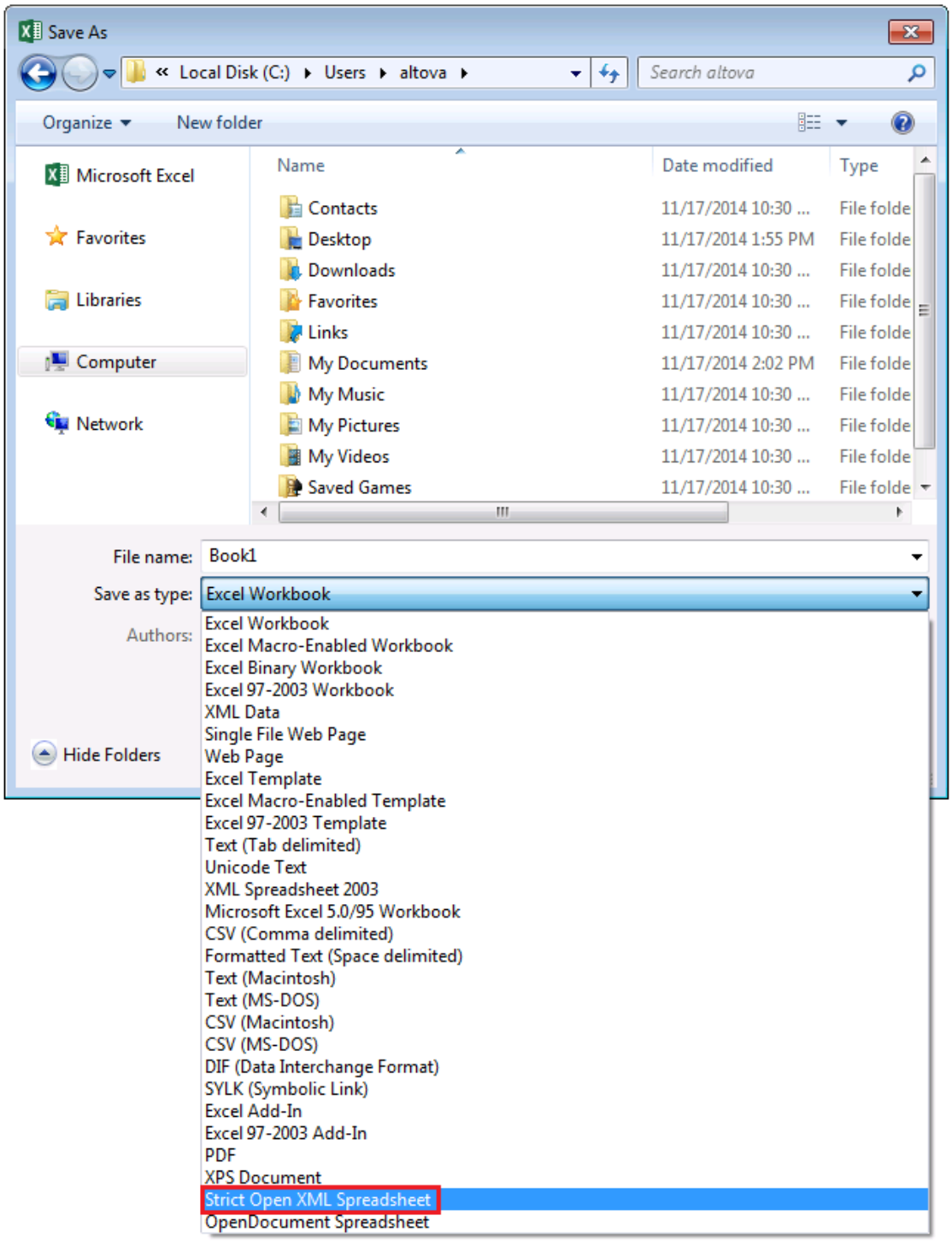
MapForce can read data from and write data to Microsoft Excel 2007+ workbooks, in the default *Office Open XML (OOXML)* format. This format was first introduced in Microsoft Office 2007 and, in case of Excel workbooks, is associated with the default .xlsx extension. In the MapForce interface, and in this documentation, Microsoft Excel 2007 and later files are generically referred to as "Excel 2007+" files.

Microsoft Office 2013 added support for *Strict Open XML Spreadsheet* format (ISO/IEC 29500 Strict). The *Strict Open XML Spreadsheet* format also has the .xlsx extension; however, technically this is a distinct format which adheres to stricter validation rules.

The following table illustrates how reading and writing data from/to Excel 2007+ workbooks is supported across MapForce transformation languages. Notice the differences between *Office Open XML* and *Strict Open XML Spreadsheet* formats.

Microsoft Excel Format	MapForce Language	Reading Support	Writing Support
Office Open XML	BUILT-IN	Yes	Yes
	C#	Yes	Yes
	Java	Yes	Yes
	XSLT2	Yes	No
Strict Open XML Spreadsheet	BUILT-IN	Yes	No
	C#	Yes	No
	Java	Yes	No
	XSLT2	Yes	No

If you need to convert any *Office Open XML* files generated by MapForce to *Strict Open XML Spreadsheet* format, open the workbook in Excel 2013, and then save it as *Strict Open XML Spreadsheet*.



Saving to Strict Open XML Spreadsheet format

As mapping components in MapForce, Excel 2007+ files have the following general behavior:

- You can map data from Excel 2007+ to any component supported in MapForce, and vice

versa, including XBRL taxonomy files (see Excel to XBRL example).

- If Microsoft Excel 2007 or later is installed on your computer, you can preview the transformation output immediately in the Output tab of the mapping window, and you can save it to a file. If you don't have Excel 2007 or later, you can still map to or from Excel 2007+ files. In this case, you cannot preview the result in the Output tab, but you can still save it, by clicking **Save Output File** on the **Output** menu.
- As an alternative to generating and saving the output manually, you can compile the mapping design to a MapForce Server execution file, or deploy it to a FlowForce Server, and execute it as and when required through FlowForce Server jobs. For further information, see Compiling a MapForce mapping and Deploying a MapForce mapping, respectively.
- You can generate mapping code (see the supported languages above) and execute the mapping from Visual Studio or from your custom application. Note that, when generating code for C#, OOXML support requires Visual Studio .NET 3.0 or later.

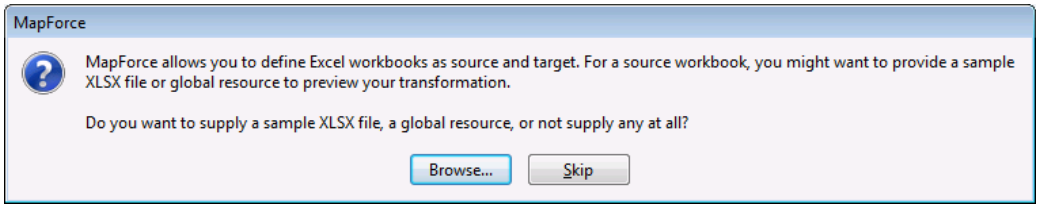
This section includes the following topics:

7.7.1 Adding Excel 2007+ Files as Mapping Components

To add a Microsoft Excel 2007+ (*.xlsx) component to the mapping area:

1. Do one of the following:

   - On the **Insert** menu, click **Excel 2007+ File**.
   - Click the **Insert Excel 2007+ File** ( ) toolbar button .

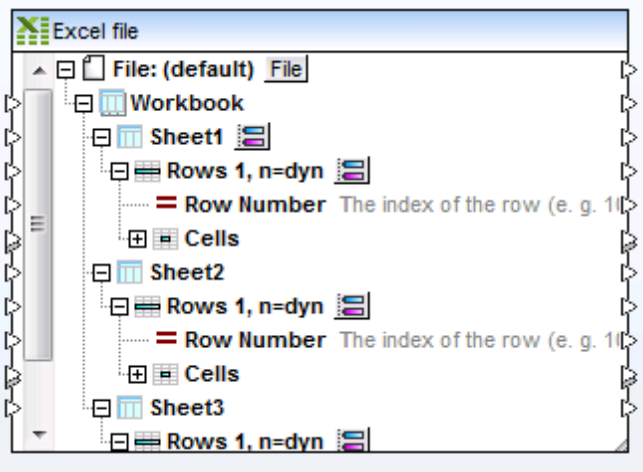


2. Do one of the following:

   - If you want to map data *from* a Microsoft Excel workbook, click **Browse** to select the .xlsx file from which you are mapping data. MapForce uses the sample Excel file to read worksheet names and cell ranges from it. If you have defined any Excel files as global resources, you can also select them from the Global Resources dialog box (see Global Resources). Click **Skip** if you would like to provide a sample file later.
   - If you want to map data *to* a Microsoft Excel workbook, click **Skip**. By default, when the mapping transformation runs, MapForce will generate an output Excel file named **xlsx-mapforce.xlsx** in the mapping folder. If required, you can change the name of the output file from the Excel 2007+ Component Settings (see About the Excel 2007 + Component).

7.7.2 About the Excel 2007+ Component

When you add an Excel 2007+ file to the mapping area without specifying a sample file (see Adding Microsoft Excel Files as Mapping Components), MapForce creates a default component which includes three worksheets (illustrated below). If you provide a sample file, MapForce reads the sample file and creates only the required worksheets.



Default Excel component

The structure of the Excel 2007+ component in MapForce reflects the structure of data in the Excel workbook, with the difference that in MapForce it is expressed in a tree structure which makes it possible to map each individual cell.

Before you can connect the Excel 2007+ component to any other component type, you will need to instruct MapForce precisely what are the columns and rows to be used in the mapping. Unlike other MapForce components such as XML or JSON, Excel 2007+ files do not have an explicit schema that MapForce can use to infer the structure of your data. Instead, MapForce provides you with settings from where you can define:

- What data precisely (such as worksheets, named ranges and tables, columns, rows) must be selected from your workbook (if you are reading from a workbook);
- To which worksheets, columns, and rows must MapForce write data (if you are writing to a workbook).

You can also configure the component to read from multiple locations within a workbook, or write to multiple locations, in the same mapping operation.

The required component configuration settings are available directly on the component. Use the following table to get started.

Component Item	Icon	Description
Workbook		Represents an Excel workbook.
Worksheet		Represents an Excel worksheet. The button displayed

<table>
<tr><th>Component Item</th><th>Icon</th><th>Description</th></tr>
<tr><td></td><td></td><td>next to the first worksheet lets you specify worksheet-related settings.</td></tr>
<tr><td>Rows</td><td>  </td><td>Represents a range of Excel rows. You can add multiple ranges of rows within a worksheet. This enables you read from (or write to) multiple ranges of cells in the same mapping operation.

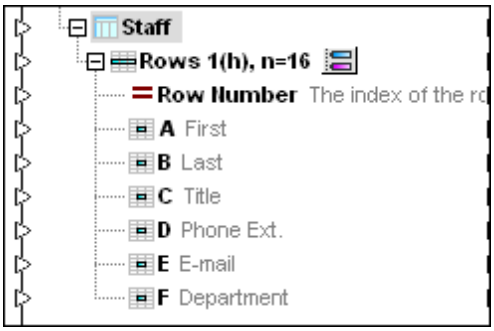
For each defined range, you can specify individual data selection options. For example, one range may begin at row 1 and include all columns of that row, while another range may begin at row 3, and consist of a dynamic number of rows, depending on the amount of data in the source Excel file.

To help you see all range settings at a glance, the component provides visual clues about them, as shown below.

Rows n Indicates a range which begins at row n.
Row n Indicates a single-line range of row n.
Rows prev+n Indicates a range which begins n rows a previous range.
Rows n(h) Indicates a range which begins at row n, the first row is designated as a header r
n=n Indicates a range which consists of exac rows.
n=dyn Indicates a dynamic range. Dynamic ran may have an unlimited number of rows.

The  button displayed next to each row range lets you specify advanced data selection settings for that range.</td></tr>
<tr><td>Cells</td><td>  </td><td>Represents all the cells (columns) of a particular row. This item appears if the component is configured to show a single cell for all columns (this is the default MapForce behavior).

Alternatively, you can configure a component to display each column separately, in which case it would look as shown in the following sample.</td></tr>
</table>

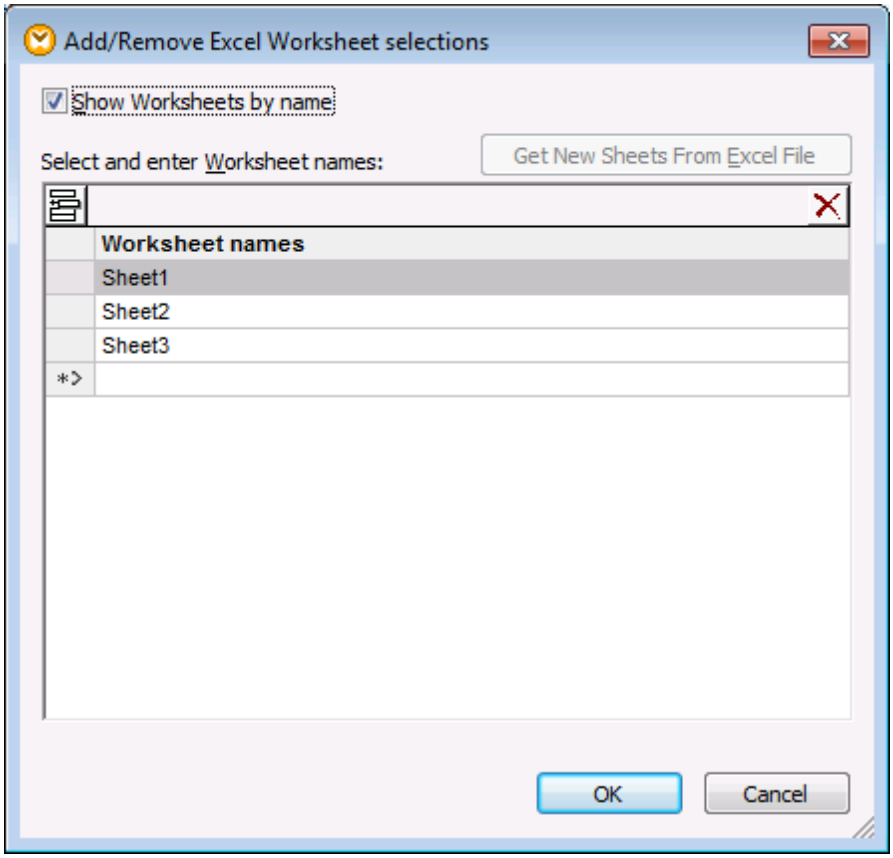
Component Item	Icon	Description
		
Change Selection		The  button displayed next to each worksheet or row lets you specify settings meaningful in that context. Using this button, you can modify the mapping structure of the Excel component as required. For example, if you are reading data from an Excel file, you can specify the worksheet, row and column from where MapForce should read data. If you are writing to an Excel file, you can specify the worksheet, row and column to which MapForce should write data.

The following sections describe the operations relevant to Excel 2007+ components:

- Adding and Removing Worksheets
- Adding and Removing Row Ranges
- Selecting Ranges of Cells
- Excel 2007+ Component Settings

7.7.3 Adding and Removing Worksheets

You can add or remove worksheets on the Excel 2007+ component, either manually, or by reloading them from an input .xlsx file. To do so, click the  button next to a worksheet node.



Add/Remove Excel Worksheet selections dialog box

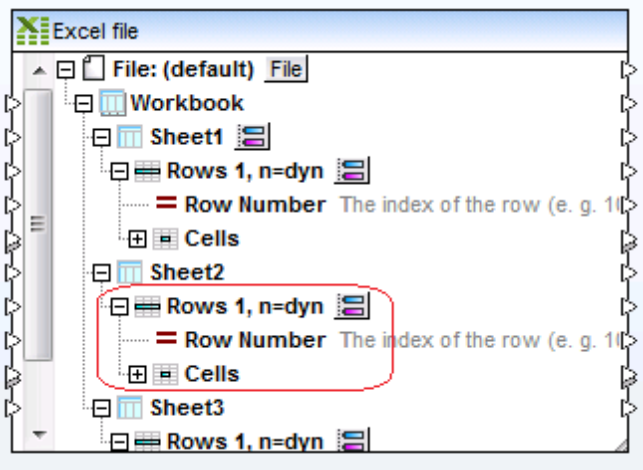
The options applicable to worksheets are as follows.

Show Worksheets by name	This option must be selected if each worksheet in your workbook has a different layout and therefore must appear as a separate item in MapForce. If all worksheets in your workbook have an identical structure, you can make this option inactive. This way, MapForce will collapse the worksheet items to a single item representing the ordered collection of all worksheets. If a workbook has multiple worksheets and the **Show Worksheets by name** option is inactive, then MapForce treats the workbook like a single worksheet. This allows a mapping to process any number of worksheets at once, but requires that all worksheets have the same structure.
Insert ( *)*	Inserts a new worksheet before the currently selected one.
Append ( *)*	Appends a new worksheet. Type a name into the text field to

	the right of the icon.
Delete ( )	Deletes the currently selected worksheet.
Get New Sheets From Excel File	Gets into the component the new worksheets found in the input Excel file. This setting is relevant and enabled in source components which have an input Excel file (see Excel 2007+ Component Settings).

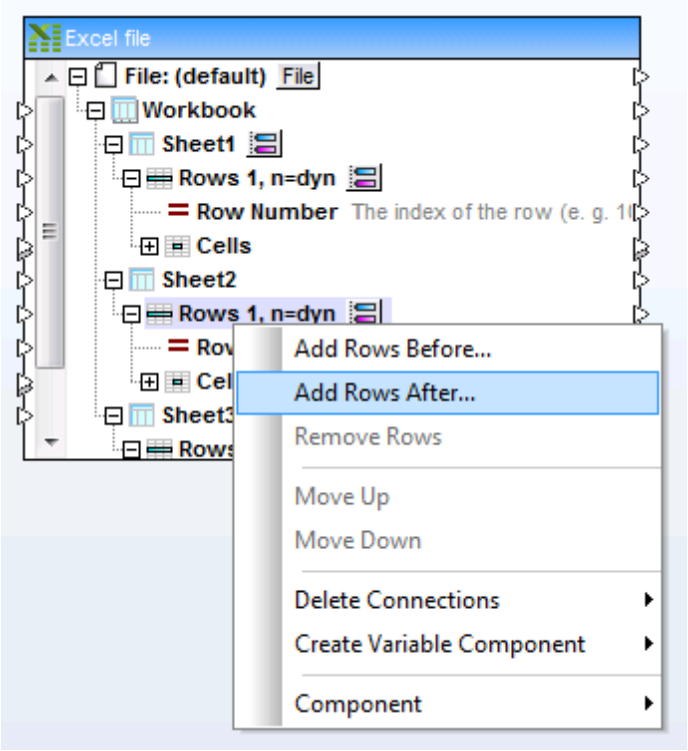
7.7.4 Adding and Removing Row Ranges

You can instruct MapForce to read (or write) a particular range of cells at a particular location within a worksheet. The following sample component illustrates a range which is available in Sheet2 of the workbook.



By default, any range is set to begin at Row 1 and iterate dynamically for n rows (**n=dyn**). However, you can change these and other settings if required (see Selecting Ranges of Cells).

You can create as many ranges of rows as required within the same worksheet, and remove the ones you do not need. To add a new range of rows, right-click any Rows ( ) node, and then select **Add Rows Before** or **Add Rows After**, respectively.



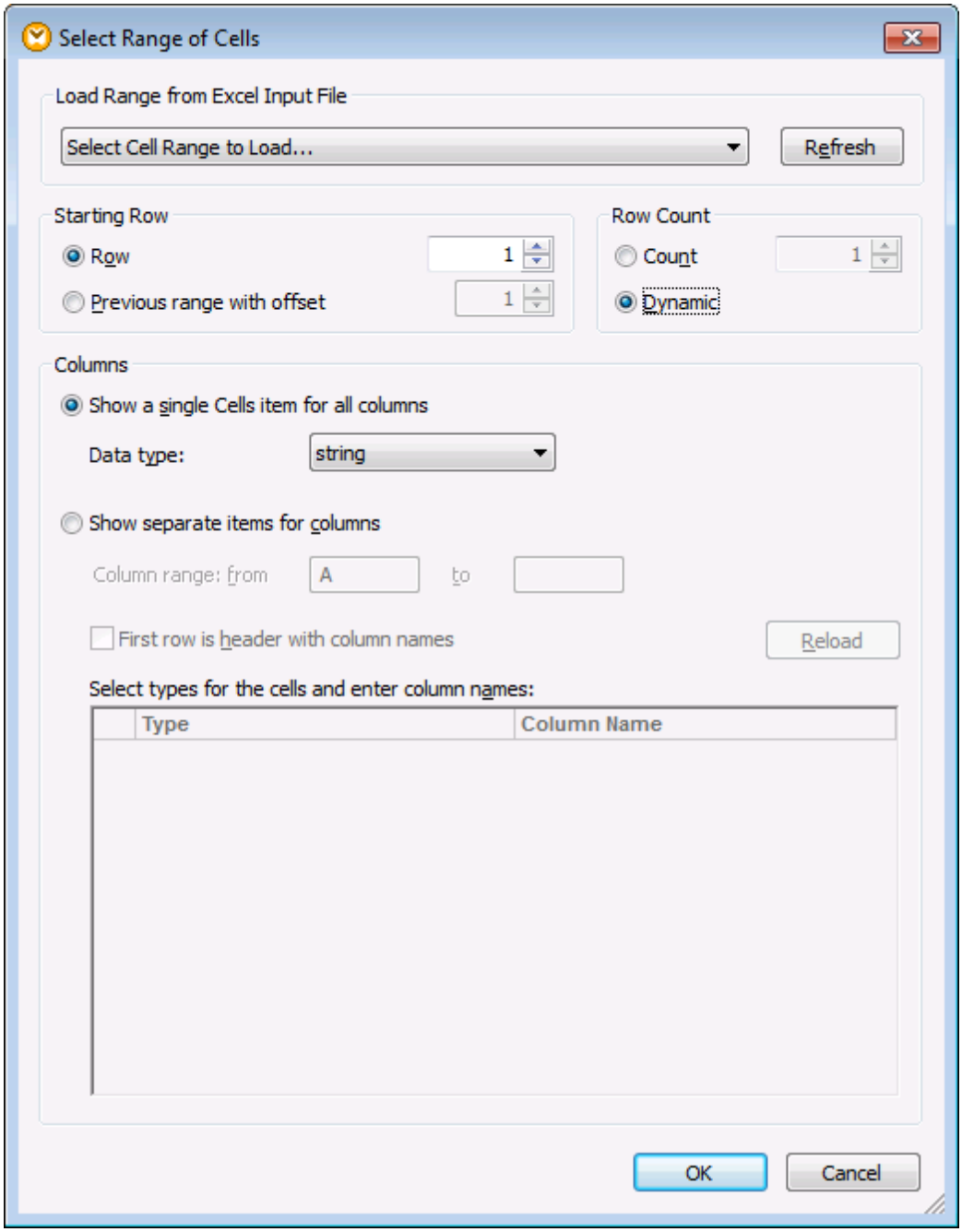
Adding cell ranges

To delete a range, right-click it, and then select **Remove Rows**.

To move a range up or down in the component, right-click it, and then select **Move Up** (or **Move Down**, respectively).

7.7.5 Selecting Ranges of Cells

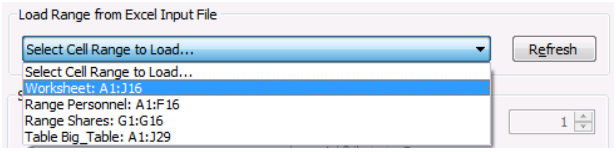
You can define what range of cells must be read by MapForce (when reading from a workbook) or written to (when writing to a workbook) from the "Select Range of Cells" dialog box. To open this dialog box, click the button next to a cell range on the component.

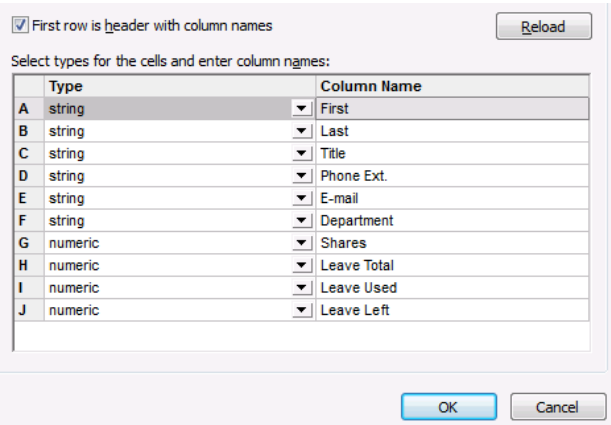
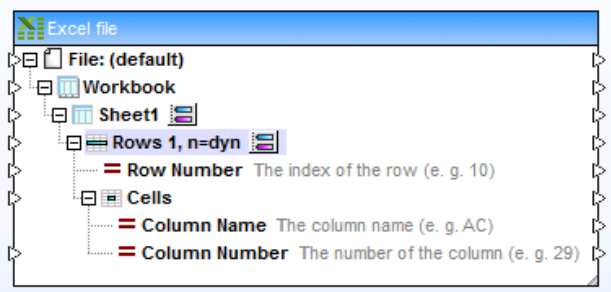


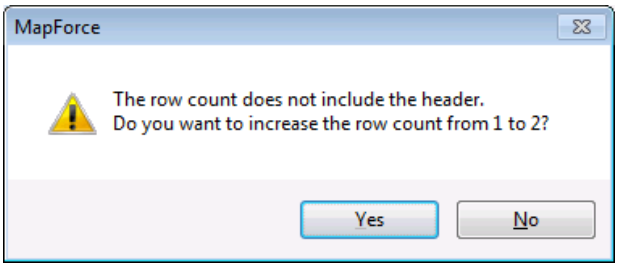
Select Range of Cells dialog box

The available settings are as follows.

Load Range from Excel Input File	**If you are reading from a workbook** Use this option to select a particular worksheet range, named range or table.

	 If the **Show Worksheets by name** option is disabled (see Adding and Removing Worksheets), data from all worksheets is visible in the list. The **Refresh** button updates the cell ranges from the input Excel file. Note that only rectangular ranges are currently supported. **If you are writing to a workbook** This option is not available.
Starting Row	The **Row** option lets you define the first row of data for the specific range. For example, if you enter "5" as starting row, MapForce will read (or write) beginning with the fifth row of the workbook. The **Previous range with offset** option is meaningful if there is another range in the same worksheet. It instructs MapForce to move the current range *N* rows down from the previously defined range. The minimum offset value is 1.
Row Count	**If you are reading from a workbook** **Count** defines the exact number of rows from which you want to read data, starting from the position defined in the **Starting Row** (see previous option). This value is automatically populated if you selected an Excel named range or table. The option **Dynamic** instructs MapForce to read all rows found in the source data beginning with **Starting Row**. Use this option only if your range is the last defined range of the worksheet, otherwise any subsequent range will not select data from the source Excel file. **If you are writing to a workbook** **Count** defines the exact number of rows to which data should be written, starting from the position defined in the **Starting Row** (see previous option). Note that if your input instance contains more rows than allowed by **Count**, MapForce writes only the number of rows defined by **Count**, and ignores the rest of data without any warning. The option **Dynamic** instructs MapForce to write all rows found in the input instance, beginning with **Starting Row**. If you defined a header row using the **First row is header with column names** option, **Count** does not take the header row into account (see the

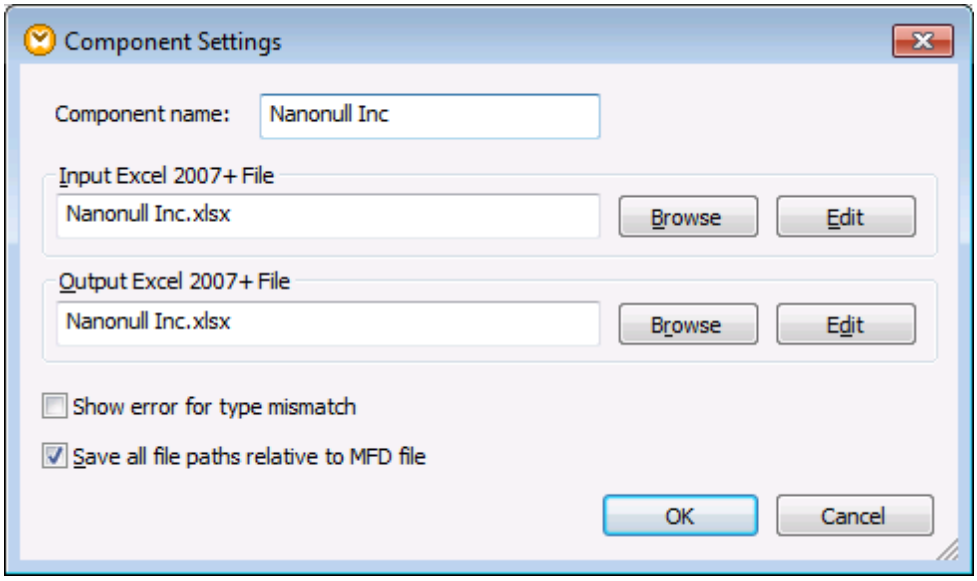
	Excel_Company_to_XML.mfd sample). 
Show a single Cells item for all columns	Collapses all cell items into a single mappable *Cells* item as shown below.  **If you are reading from a workbook** Use this option if you want to read all the cells of a particular row. For examples, see the **ExcelColumnsToRecords.mfd** and **ExcelWith2Dimensions.mfd** samples available in the **MapForceExamples** project, **OOXML Excel 2007+** folder. **If you are writing to a workbook** Use this option to write data to one or multiple cells in the same row. For an example, see the **Altova_Hierarchical_Excel.mfd** sample.
Show separate items for columns	This option enables you to access individual columns of the given row range. If you selected a worksheet range, named range or table, the column names are automatically populated. Otherwise, you can select specific column names by typing their corresponding alphabetic letter in the **from** and **to** text boxes. If the ranges in the input Excel file have changed, click **Reload** to update the component with the changes.

	To instruct MapForce to consider the first row of a range as the column header for that range, select the **First row is header with column names** option. When you activate or deactivate this option, and **Row Count** has been set, MapForce prompts you to optionally adjust the **Row Count** value. This prevents the **Row Count** from being one row too large, or too small.  Note that the **Row Count** setting does not take the header row into account.

7.7.6 Excel 2007+ Component Settings

After you add an "Excel 2007+" component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component Settings dialog box in one of the following ways:

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



Component Settings dialog box

The available settings are as follows.

Component name	The component name is automatically generated when you create the component. You can however change the name at any time. If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well. The component name can contain spaces (for example, "Source File") and full stop characters (for example, "Orders.xlsx"). The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications when changing the name of the component: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have different encoding in Windows and at the command line.
Input Excel 2007+ file	Specifies the Microsoft Excel file from which MapForce will read data. This field is meaningful in a source component, when MapForce uses it to read the Excel worksheet names and columns. To change the location of the file, click **Browse** and select

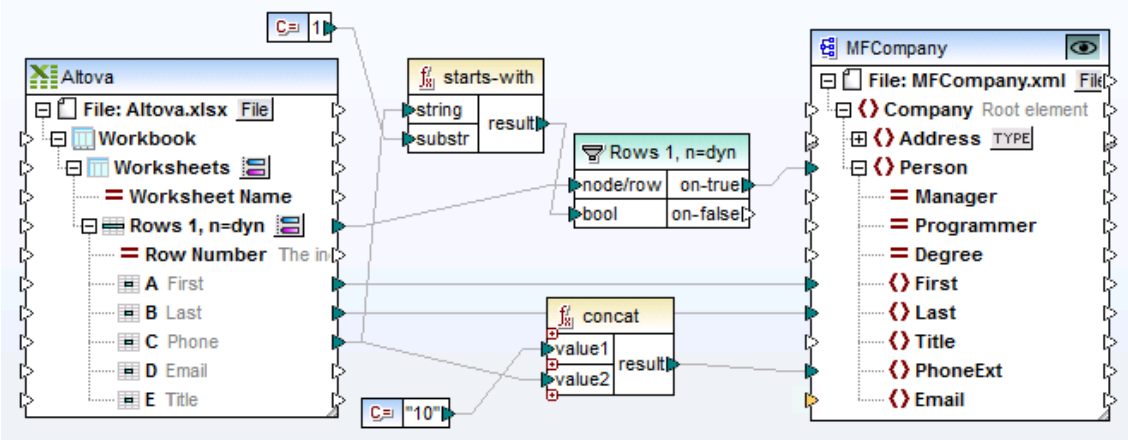
	the new file. To edit the file in Microsoft Excel, click **Edit**.
Output Excel 2007+ file	Specifies the name or path of the Microsoft Excel file to which MapForce will write data. This field is meaningful in a target component. To change the location of the file, click **Browse** and select the new file. To edit the file in Microsoft Excel, click **Edit**. If you do not specify an output file, MapForce generates the output to a file named **xlsx-mapfore.xlsx**. By default, this file is generated in the same folder as the .mfd file, unless you configured a different path. Any existing data in the output Excel file will be overwritten when the mapping transformation runs.
Show error for type mismatch	Enables error messages in the Messages window, when mismatches occur between a data type declared in the component and data in the input .xlsx file. For example, let's assume that this setting is enabled, and you declared a column as numeric in the *Select range of cells* dialog box. If the Excel file contains text data in this column, an error message is shown in the Messages window after you connect this column to some output and preview the output.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative Paths on a Component.

7.7.7 Example: Mapping Excel 2007+ to XML

The mapping file used in the following example is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\Excel-mapping.mfd**. When you open the sample file, you will notice that it contains three distinct mapping transformations. The top two transformations are discussed in this section.

The aim of the first mapping is to do the following:

- Select from the source Excel workbook only people whose phone extension (column C of the workbook) starts with a "1".
- Add the prefix "10" to the original number, and write it to a target XML file, along with the First and Last names of the respective persons.



Excel-mapping.mfd (sample 1)

The mapping is configured as follows:

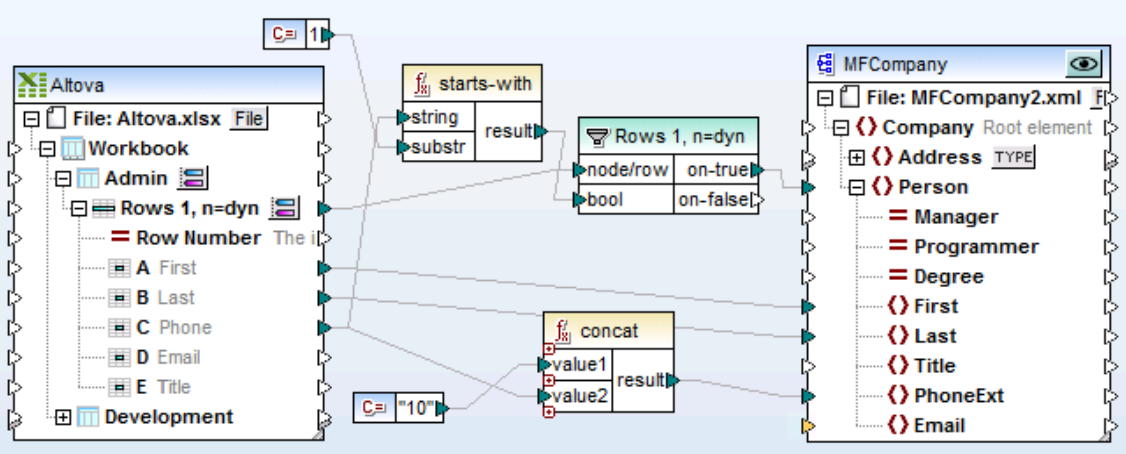
- **Altova.xlsx** is the source Excel 2007+ workbook. Columns A and B supply the First and Last names respectively. Column C supplies the phone extension number.
- Both worksheets of the workbook are shown as one node in the component (in other words, the **Show worksheets by name** option is disabled). This is indicated by the "Worksheets" node under the Workbook item.
- The `starts-with` function checks if the phone extension (col. C) starts with a "1", and if the result is true then those records are forwarded by the filter component.
- The **concat** filter adds the prefix "10" to each of the telephone extensions and writes it to the PhoneExt item.
- MFCompany.xsd is the target component and contains the filtered person details when data is output.

The result of the mapping is that four persons have been mapped to the XML file with their details.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xsi:schemaLocation="http://my-company.com/namespace
3    <Person>
4      <First>Steve</First>
5      <Last>Meier</Last>
6      <PhoneExt>10114</PhoneExt>
7    </Person>
8    <Person>
9      <First>Max</First>
10     <Last>Nafta</Last>
11     <PhoneExt>10122</PhoneExt>
12   </Person>
13   <Person>
14     <First>Carl</First>
15     <Last>Franken</Last>
16     <PhoneExt>10147</PhoneExt>
17   </Person>
18   <Person>
19     <First>Mark</First>
20     <Last>Redgreen</Last>
21     <PhoneExt>10152</PhoneExt>
22   </Person>
23 </Company>
```

The second mapping is identical with the first one, except that worksheets have been individually enabled using the **Show worksheets by name** option. This mapping is configured as follows:

- The **Admin** and **Development** worksheets are both visible under the Workbook item.
- Connectors have only been defined from the **Admin** worksheet to the target component.



Excel-mapping.mfd (Sample 2)

The result of the mapping is that only two persons have been mapped to the XML file with their details.

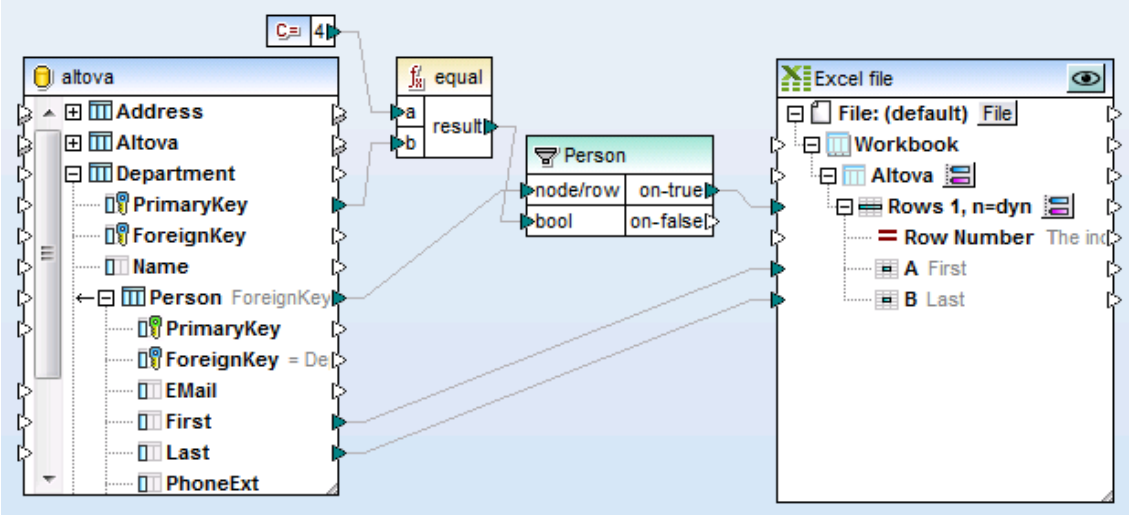
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xsi:schemaLocation="http://my-company.com/namespace
3    <Person>
4      <First>Steve</First>
5      <Last>Meier</Last>
6      <PhoneExt>10114</PhoneExt>
7    </Person>
8    <Person>
9      <First>Max</First>
10     <Last>Nafta</Last>
11     <PhoneExt>10122</PhoneExt>
12   </Person>
13 </Company>
```

7.7.8 Example: Mapping Database Data to Excel 2007+

The mapping file used in this example is available as **Excel-mapping.mfd** in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. The third mapping of the three is discussed here.

The aim of the mapping is as follows:

- Extract from the "altova" database only persons whose department primary key is equal to 4 (that is, those who are in the IT department).
- Write the extracted records to a default Excel 2007+ component.

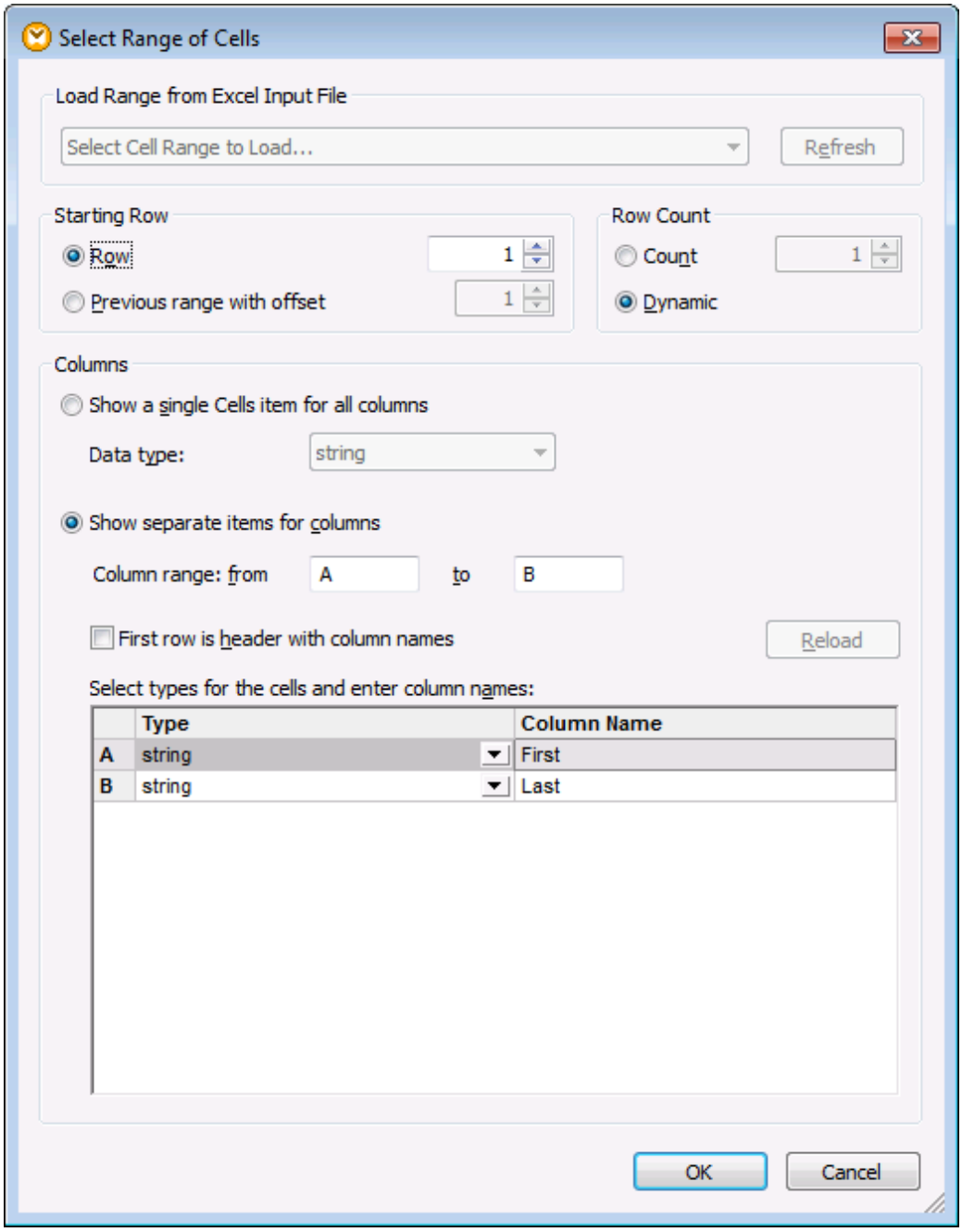


Excel-mapping.mfd (sample 3)

The mapping is configured as follows:

1. The database "altova" was added to the mapping area from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, using the **Insert | Database** menu command, and following the wizard for a Microsoft Access database.
2. The default Excel 2007+ component was added using the **Insert | Excel 2007+ file** menu command, and then skipping the option to supply a sample file.

3. The first worksheet (Sheet1) was renamed by clicking the button adjacent to it and then entering "Altova" as worksheet name.
4. The **Rows 1, n=dyn** range was configured by clicking the button adjacent to it. The cell range options were defined as follows:



5. Other options were defined as follows:

   - The value of the **PrimaryKey** is compared to the value "4", supplied by the **Constant** component, using the `equal` function.
   - The **filter** component passes on the First and Last fields if the Boolean condition is true (that is, if the department primary key is "4").

- The **on-true** item is connected to the **Rows 1, n=dyn** item in the Excel file.

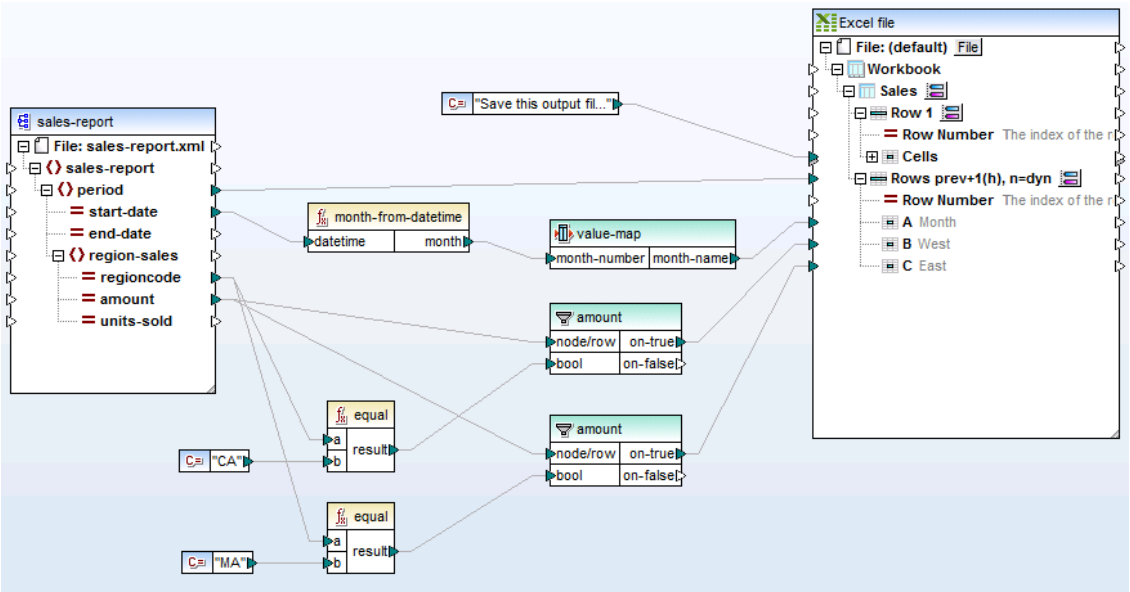
The result of the mapping is that four persons of the IT department are shown in the Excel workbook.

	A	B	C	D
1	Alex	Martin		
2	George	Hammer		
3	Jessica	Bander		
4	Lui	King		
5				
6				

Altova

7.7.9 Example: Supplying Data to Preformatted Excel Sheets

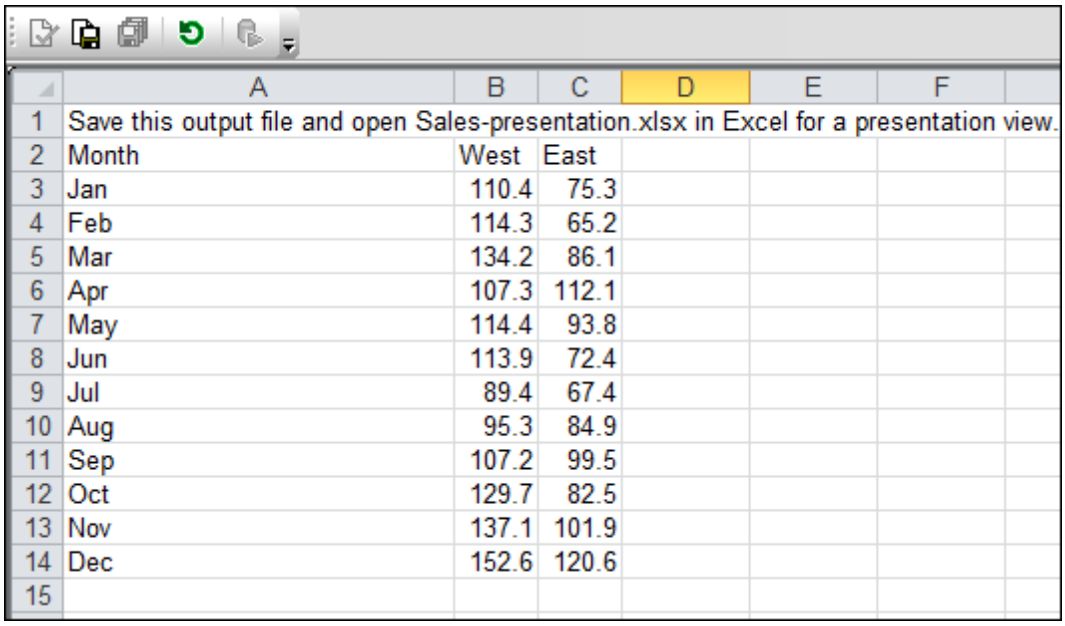
MapForce data that is output to an Excel sheet can be used as a data source for a preformatted Excel document. The screenshot below shows the **Sales_to_Excel.mfd** mapping available in the <Documents>\Altova\MapForce2016\MapForceExamples\ folder.



Sales_to_Excel.mfd

To save the mapping result as an unformatted Excel data source:

1. Click the BUILTIN icon to select the output language (you could also select one of the programming languages and compile).
2. Click the Output button to see the result as an embedded Excel sheet in MapForce.
3. Click the Save generated output icon and keep the suggested file name as "xlsx-mapforce.xlsx", then click Save to save the workbook to the ...\MapForceExamples folder.

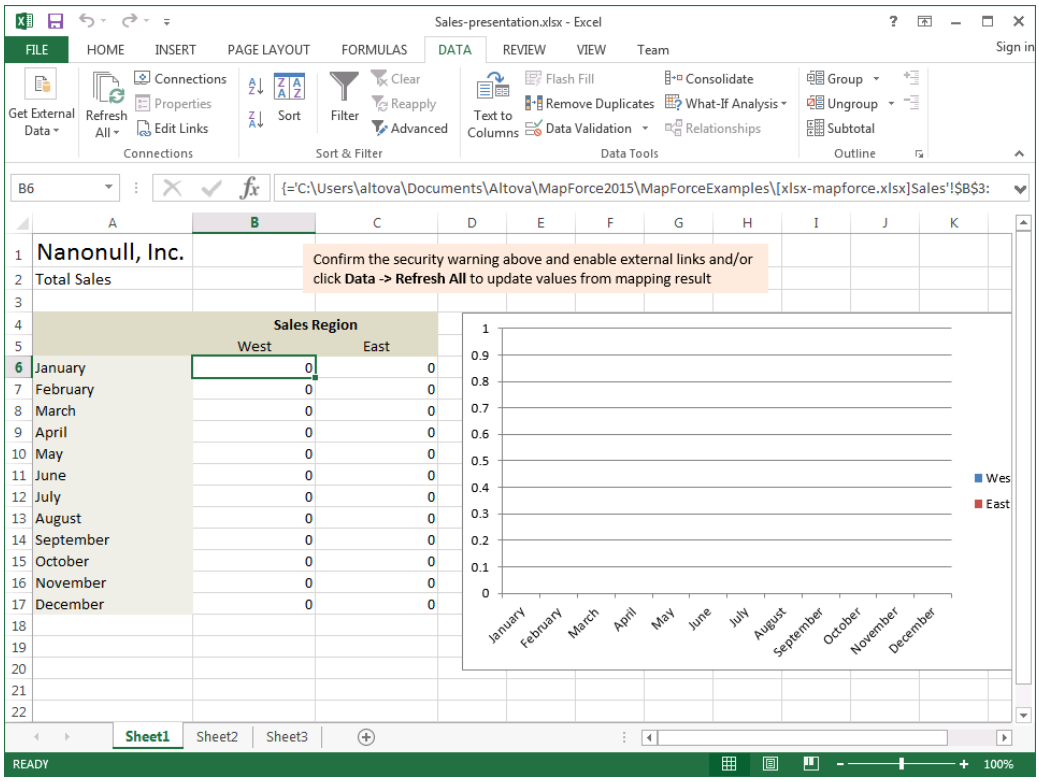


	A	B	C	D	E	F
1	Save this output file and open Sales-presentation.xlsx in Excel for a presentation view.					
2	Month	West	East			
3	Jan	110.4	75.3			
4	Feb	114.3	65.2			
5	Mar	134.2	86.1			
6	Apr	107.3	112.1			
7	May	114.4	93.8			
8	Jun	113.9	72.4			
9	Jul	89.4	67.4			
10	Aug	95.3	84.9			
11	Sep	107.2	99.5			
12	Oct	129.7	82.5			
13	Nov	137.1	101.9			
14	Dec	152.6	120.6			
15						

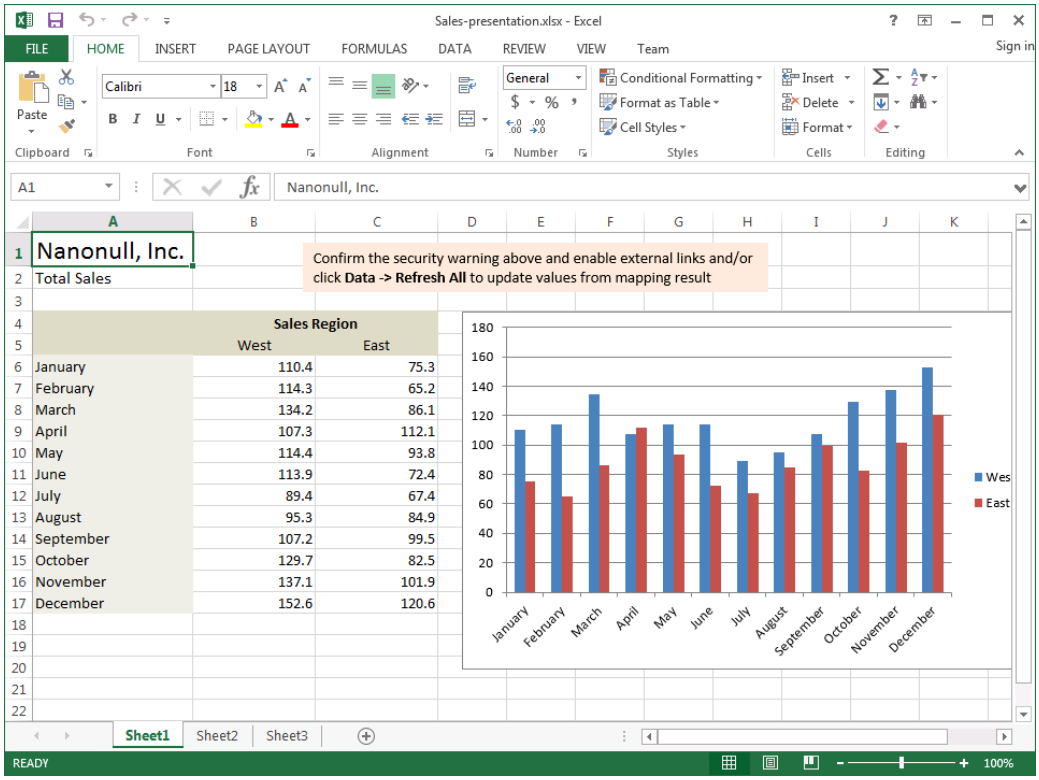
Formatting the Excel template file

You would normally now have to create a new Excel file that will be used to display the MapForce data. You could add formulas that reference the MapForce output file data, add a chart to the sheet and so on. There is no need to do this now as a preformatted Excel file called **Sales-presentation.xlsx** exists in the ...\MapForceExamples folder.

1. Double click the **Sales-presentation.xlsx file.** A warning appears that external links have been disabled and they need to be enabled to link to the source data. The current cursor position, B6, shows that the source data range is **Sales'!$B$3:$C$14** of the xlsx-mapforce.xlsx file.



2. Click the Enable button (or select the menu option Data | Refresh all). The external links are now enabled and the underlying source data is shown as a table and a graph.



7.8 XBRL

Altova web site: Mapping XBRL files

The Altova suite of products supports XBRL 2.1, XBRL Dimensions 1.0, and XBRL Table Linkbase 1.0.

Each Altova product takes care of a different aspect of XBRL, as follows:

- XMLSpy 2016 edits or creates new taxonomies and generates XBRL reports based on XSLT transformation files (XSLT transformation files can be created in StyleVision)
- StyleVision 2016 creates taxonomy stylesheets/templates, allowing you to generate XBRL reports
- MapForce 2016 maps data to or from XBRL instance files, and enables you to use XBRL taxonomies when designing the mapping structure. You can map XBRL data to or from Microsoft Excel 2007 and later, databases, or CSV files. You can thus create interim reports, or filter specific data from XBRL instance documents.

MapForce supports mapping to or from XBRL components based on taxonomies such as:

- US-GAAP (Generally Accepted Accounting Principles)
- IFRS (International Financial Reporting Standards)
- COREP / FINREP (Common Reporting and Financial Reporting taxonomies published by the European Banking Authority)

When mapping to or from XBRL components, it is assumed that the underlying XBRL taxonomy file is available from the taxonomy designer. Company specific taxonomy files are created or edited in XMLSpy with a specific existing taxonomy (for example, US-GAAP) as the basis of the new taxonomy. The new taxonomy file extends the base taxonomy and adds a new namespace attribute and prefix to it.

Note: By default, the MapForce installation package includes the most recent versions of the US-GAAP and IFRS taxonomies. Additional taxonomies, including earlier versions, are available on the Altova components download page (http://www.altova.com/download_components.html).

Conventions

This documentation makes references to XBRL terminology as defined by the following specifications.

Specification	URL
XBRL Specification 2.1	http://www.xbrl.org/Specification/XBRL-2.1/REC-2003-12-31/XBRL-2.1-REC-2003-12-31+corrected-errata-2013-02-20.html
Dimensions Specification 1.0	http://www.xbrl.org/specification/dimensions/rec-2012-01-25/dimensions-rec-2006-09-18+corrected-errata-2012-01-25-clean.html
Table Linkbase Specification 1.0	http://www.xbrl.org/Specification/table-linkbase/REC-2014-03-18/table-linkbase-REC-2014-03-18.html

In this documentation, references to the specifications above are as indicated by the Specification column. Additionally, the § character is used to denote a particular section number within the specification. For example, a reference to Section 5.4 of the XBRL Table Linkbase 1.0 specification looks as follows:

A table (Table Linkbase Specification 1.0, §5.1) is represented by the  *icon in the XBRL component.*

7.8.1 Adding XBRL Files as Mapping Components

Before you can map data to or from an XBRL document, the XBRL document must be added to the mapping area as a mapping component. There are two ways of adding XBRL documents as mapping components:

- As a flat XML document
- As a hierarchically structured XBRL document

The first option inserts the XBRL file as a flat XML file without any business logic concepts or specific XBRL hierarchical structure. Such a component is useful only for very simple mappings where no dimension handling or automatic XBRL context generation is required.

The second option inserts the XBRL taxonomy and displays its contents in a hierarchical fashion suitable for XBRL. In this scenario, MapForce resolves the Discoverable Taxonomy Set (DTS) references and builds automatically the derived XBRL structure. This means that the XBRL mapping component will include XBRL-specific items (which may be actual nodes in the XBRL file or "abstract" nodes derived from the DTS). For example, the following hierarchical structure is derived from the `xlink:from` and `xlink:to` presentation arcs in the presentation linkbase. (The term "linkbase" here has the meaning as found in the XLink specification (see http://www.w3.org/TR/xlink11/). Specifically, it represents a collection of links (or a link database). XLink linkbases are extended, among other uses, to XBRL, and provide additional information about concepts defined in a taxonomy.)

This example is only a basic illustration of how MapForce resolves the taxonomy in order to display data in a conceptual manner better suited for mapping operations than plain XML. In addition to the presentation linkbase, MapForce can also render structures derived from other linkbases, including the table linkbase (see Working with XBRL Tables).

For information about items that can appear on an XBRL component, including their conventional graphical representation, see About XBRL Component Items.

To open XBRL documents as flat XML

- On the **Insert** menu, click **XML/Schema File**.

To open XBRL documents with automatic context handling

1. On the **Insert** menu, click **Insert XBRL Document**.
2. Select the taxonomy file (*.xsd) or an XBRL instance file (*.xbrl or *.xml). If you select a taxonomy, a further dialog will prompt you to select a valid instance file.
3. Click **Browse...** if you intend to use this XBRL component as a source instance and select the XBRL instance file.

When you open an XBRL taxonomy, a dialog box opens prompting you to select the structure views to shown on the XBRL component. If you are not sure what structure view to select, leave the default option as is. By default, MapForce selects automatically one of the options by analyzing data from the loaded Discoverable Taxonomy Set. You can change the structure views later if necessary (see also Selecting Structure Views).

7.8.2 About XBRL Component Items

The following table gives an overview of the component items (nodes) which are typically part of an XBRL component. Note that some of the nodes are only available when you selected the relevant structure view (see Selecting Structure Views). For example, the nodes specific to the table linkbase are not visible if you choose to display XBRL data only from the presentation and definition linkbases.

Component Item	Icon	Description
Abstract item		Abstract items are used to organize related facts. They are either defined within the presentation linkbase or within domain member networks of definition linkbases. They do not actually exist in the XBRL instance file; they allow grouping of facts in an intuitive way.
Breakdown		As defined in the Table Linkbase Specification 1.0, §5.4.
Context		A context node is a container for all related business facts. Context nodes inside hypercubes manage their context elements and dimensions automatically. The `xbrli:context` node which is a child of the XBRL root element is used for manual dimension handling.
Dimension		Dimensions (XBRL Dimensions Specification 1.0, §2.5) are used to structure contextual information for business facts. Dimensions are defined in the taxonomy within a hypercube.
Explicit member		An explicit member is a member of a dimension which is defined by an enumeration of `QName` values (see also Showing Dimensions in a Component).
Explicit member value		The value of an explicit member.

Component Item	Icon	Description
Fact		Facts (XBRL Specification 2.1, §1.4) are the values of the XBRL items. They can be of the following types: • Monetary items ( ) • String items ( ) • Numeric items ( ) • General items ( ) • Shares items ( )
Footnote		Footnotes allow you to assign additional text information to facts.
Hypercube		Hypercubes (XBRL Dimensions Specification 1.0, §2.2) use information from the definition linkbase and the presentation linkbase to hierarchically structure dimensions, contexts and related XBRL concepts. See also Working with XBRL Hypercubes.
Root element		The `xbrli:xbrl` element is instance root element of every XBRL component.
Rule node		As defined in the Table Linkbase Specification 1.0, §6.6.
Structural node		As defined in the Table Linkbase Specification 1.0, §5.5.
Table		As defined in the Table Linkbase Specification 1.0, §5.1. MapForce displays the table structures in a hierarchy (see Working with XBRL Tables).
Table set		As defined in the Table Linkbase Specification 1.0, §5.2.
Tuple		Tuples (XBRL Specification 2.1, §1.4) are complex elements containing facts or other tuples as members.
Typed member		Typed members are members that are defined by an XML schema element (see http://www.xbrl.org/specification/dimensions/rec-2012-01-25/dimensions-rec-2006-09-18+corrected-errata-2012-01-25-clean.html#sec-typed-dimensions).
Unit		The unit element xbrli:unit contains units to which XBRL items refer. It is mandatory to define a value for the `xbrl:unit` element when mapping data. For example, for dollars, `UnitID` is `usd` and `Measure` is `iso4217:USD`. See also Working with XBRL Defaults.
View		Views represent extended link roles from the definition and presentation linkbase of an XBRL taxonomy.

Note that the icon is generic and is used with a variety of node types, including the XBRL root element.

Additionally, the following graphical user interface elements can appear on XBRL components if required by the context.

GUI element	Icon	Description
Additional information		This icon accompanies items that have an associated message (such as errors). Click on the icon to display additional information in the Messages pane.
Error		This icon indicates an error message.
Show Context menu		This icon accompanies items that have additional context menu options available for selection.
Unknown		This icon accompanies items that could not be resolved by MapForce due to invalid namespace references, or when the referenced item does not exist in the taxonomy.

7.8.3 Selecting Structure Views

You can select the XBRL structure views to be shown on the component, either when you add an XBRL document to the mapping area (see Adding XBRL Files as Mapping Components), or at any time later. The available structure views are as follows.

Structure View	Description
Tables from table linkbase	Shows the tables defined in the table linkbase of the taxonomy. This option is disabled if the taxonomy does not contain table definitions. This view is conditional. To proceed, select either this check box, or the **Views from presentation and definition linkbases** check box.
Views from presentation and definition linkbases	Shows data from the presentation and definition linkbases. The presentation linkbase includes hierarchies and the definition linkbase includes extended link roles with hypercubes and dimensions. This view is conditional. To proceed, select either this check box, or the **Table from table linkbase** check box.
All concepts (with context management)	Shows the additional node "All concepts" in the hierarchical structure. This node contains the hypercube "Dimensionless" which enables mappings of all concepts of the taxonomy regardless whether they are reported within hypercubes by means of the two default dimensions **identifier** and **period**. Automatic context handling is provided by the **context** node which contains all XBRL concepts of the taxonomy as children. Abstract items are not shown.

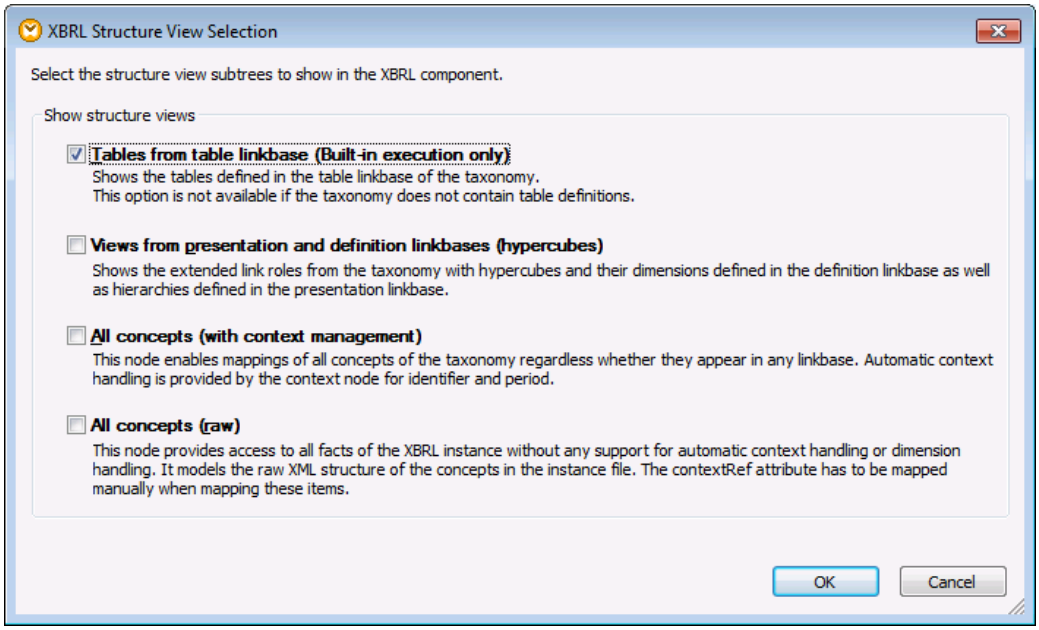
Structure View	Description
	This view is optional to display.
All concepts (raw)	Shows the additional node "All concepts (raw)" which provides access to all facts of the XBRL instance without any support for automatic context handling or dimension handling. It models the raw XML structure of the concepts in the instance file. The `contextRef` attribute has to be mapped manually when mapping these items. This view is optional to display.

Notes

- When reading data from an XBRL component (that is, if the XBRL component is a source component), you can choose any combination of structure views.
- When writing to an XBRL component (that is, if the XBRL component is a target component), the **Tables from table linkbase** cannot be used in combination with any other view.
- The **Tables from table linkbase** view requires that "Built-in" is set as transformation language (see also Selecting a transformation language).

To select the structure views to be shown on the XBRL component

1. On the root element of the XBRL component, click the **Show Context Menu** ( ) button, and then **Select Structure Views**.



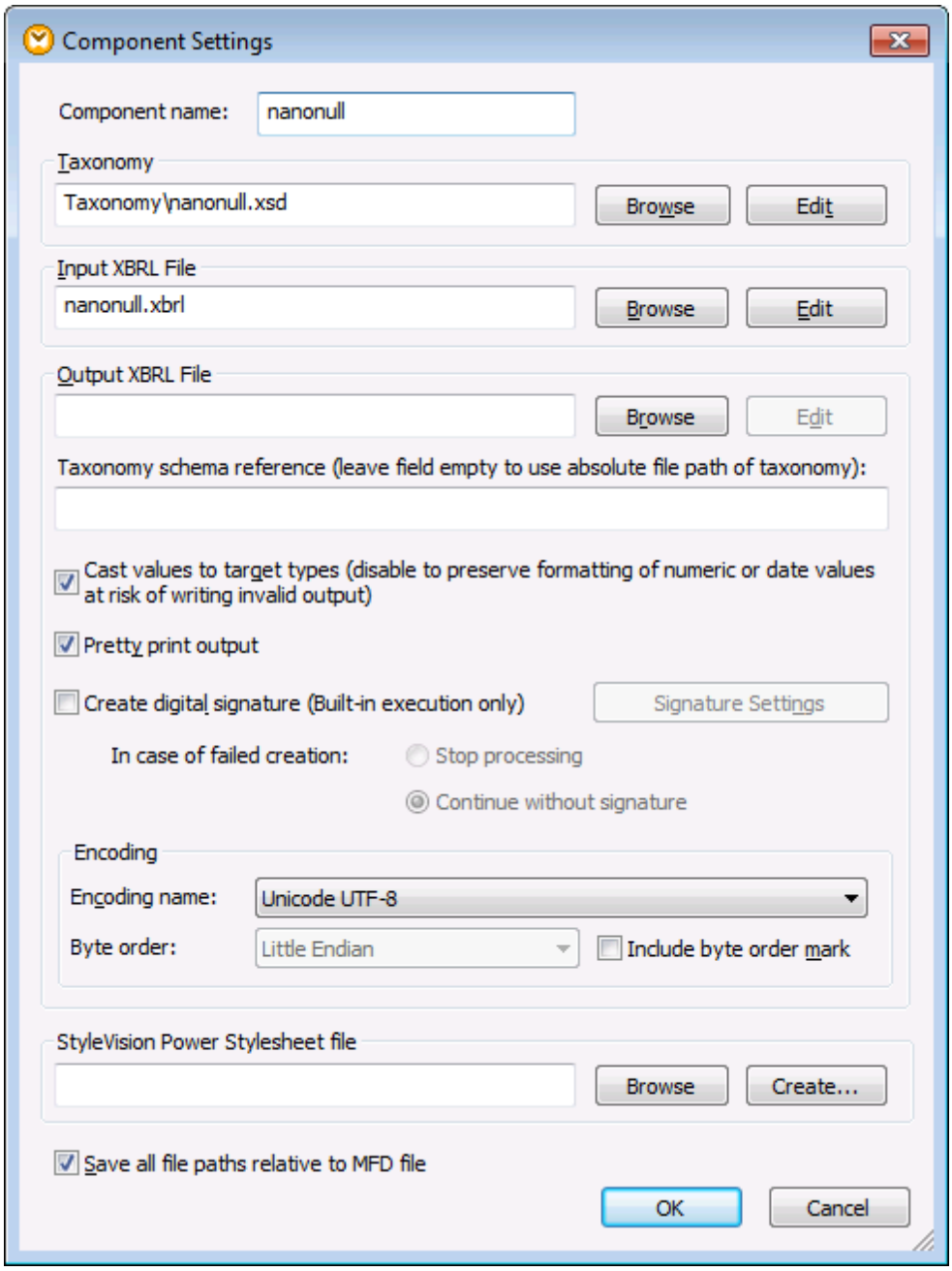
2. Select one or more structure views that must be visible in the XBRL component, and then click **OK**.

7.8.4 XBRL Component Settings

Once you added an XBRL document to the mapping area (see Adding XBRL Files as Mapping Components), you can configure the settings applicable to the component from the Component Settings dialog box.

To open the Component Settings dialog box

- Double-click the XBRL component.



The Component Settings dialog box includes the following settings.

Taxonomy	Specifies the file name and path of the main taxonomy file. To change the location of the file, click **Browse** and select the new file. To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.

Input XBRL File	Specifies the file name of the input XBRL instance for the currently selected XBRL component. This field is filled automatically when you first insert the XBRL component and assign an XBRL instance file. To change the location of the file, click **Browse** and select the new file. To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.
Output XBRL File	Specifies the file name and path where the XBRL target instance file is placed, if the component is used as a mapping target. The entry from the Input XBRL File field is automatically copied to this field when you assign the input XBRL instance file. If you do not assign an input XBRL instance file to the component, then this field contains the file name and path of the taxonomy file and the extension "xml". To change the location of the file, click **Browse** and select the new file. To edit the file in your XBRL editor (for example, XMLSpy), click **Edit**.
Taxonomy schema reference	The path of the referenced/associated taxonomy schema file relative to the MFD file. Use this field if you want to specify a different taxonomy location for validation. The taxonomy reference is written into the `href` attribute of the `link:schemaRef` element, for example: `<link:schemaRef xlink:type="simple" xlink:href="..\..\nanonull.xsd"/>`
Cast values to target types	Allows you to define if the target XML schema types should be used when mapping, or if all data mapped to the target component should be treated as **string** values. By default, this setting is enabled.
Pretty print output	Reformats the output XBRL document to give it a structured look. Each child node is offset from its parent by a single tab character.
Create digital signature	Allows you to add a digital signature to the XBRL output instance file. Adding a digital signature is possible when you select "Built-in" as transformation language (see also Digital Signatures).
Encoding	Allows you specify the following settings of the output instance file: • Encoding name

	• Byte order • Whether the byte order mark (BOM) character should be included.
StyleVision Power Stylesheet file	This option allows you to select or create an Altova StyleVision stylesheet file. Such files enable you to output data from the XBRL instance file to a variety of formats suitable for reporting, such as HTML, RTF, and others.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. This setting affects the following files: • The XBRL taxonomy file • The XBRL input file • The XBRL output file • The StyleVision stylesheet file See also Using Relative Paths on a Component.

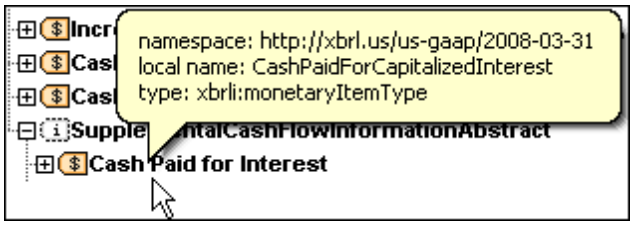
7.8.5 Setting XBRL Preferences

In MapForce, you can configure the XBRL-specific settings as follows:

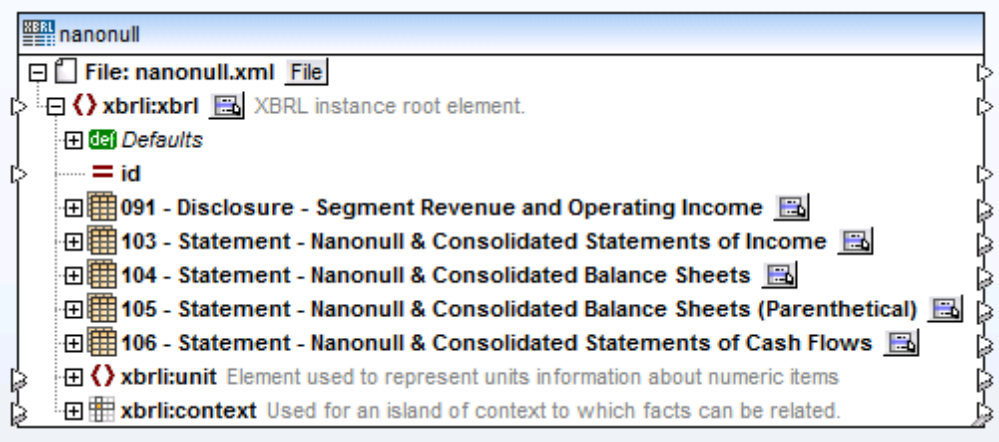
- View additional information about component items (see Enabling Tips and Annotations)
- Configure the general (application-wide) XBRL settings (see General XBRL Options).

7.8.5.1 Enabling Tips and Annotations

When tips are enabled, you can view additional information about each component item if you place the mouse cursor over it, as shown below.



You can also switch on or off the annotation text (if it exists). When enabled, annotations are displayed to the right of the item inside the component. For example, in the screen shot below, the annotation of the `xbrli:xbrl` root element is "XBRL instance root element".



To enable or disable tips, do one of the following:

- On the **View** menu, click **Show Tips**.
- Click the **Show Tips** ( ) toolbar button.

To enable or disable annotations, do one of the following:

- On the **View** menu, click **Show Annotations**.
- Click the **Show Annotations** ( ) toolbar button.

7.8.5.2 General XBRL Options

MapForce enables you to configure the following general (application-wide) XBRL settings:

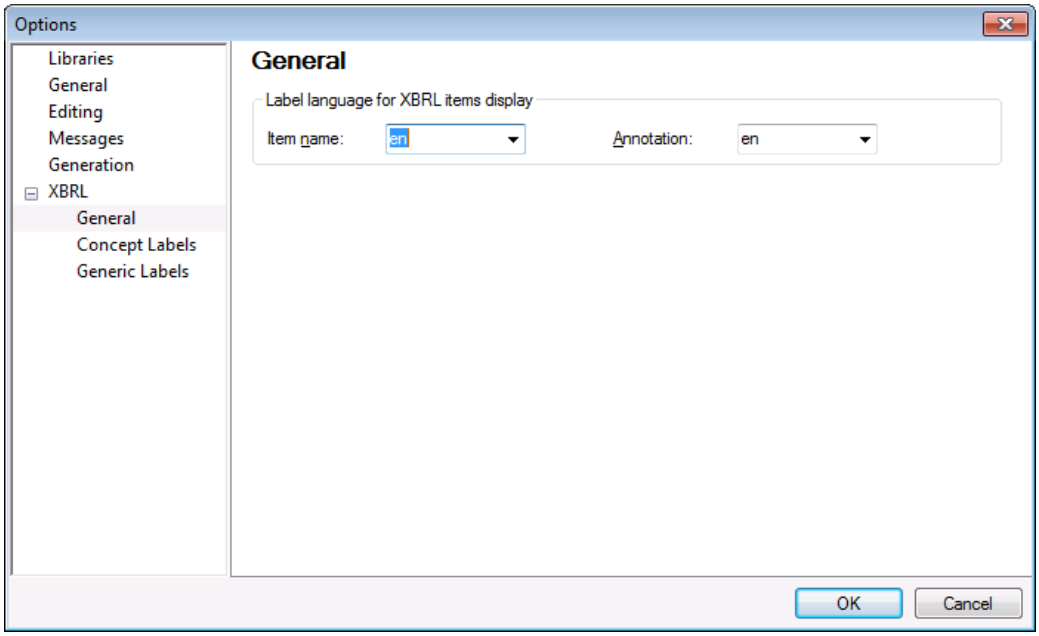
- The label language of XBRL items and their annotations
- The preferred label roles for XBRL item names
- The specific type of label roles of annotations for XBRL items

All of the above settings can be configured from the Options dialog box, XBRL section.

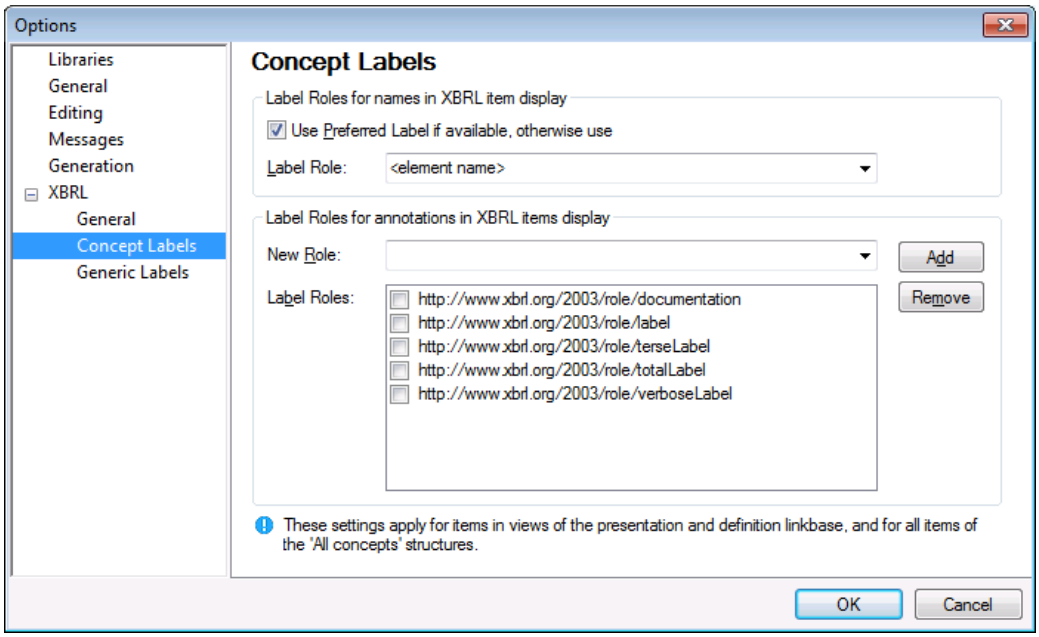
To change the general XBRL options

1. On the **View** menu, click **XBRL Display Options**.
2. In the list to the left of the dialog box, click the XBRL node. This node is divided into three sections: General, Concept Labels, and Generic Labels.
3. Do one of the following:

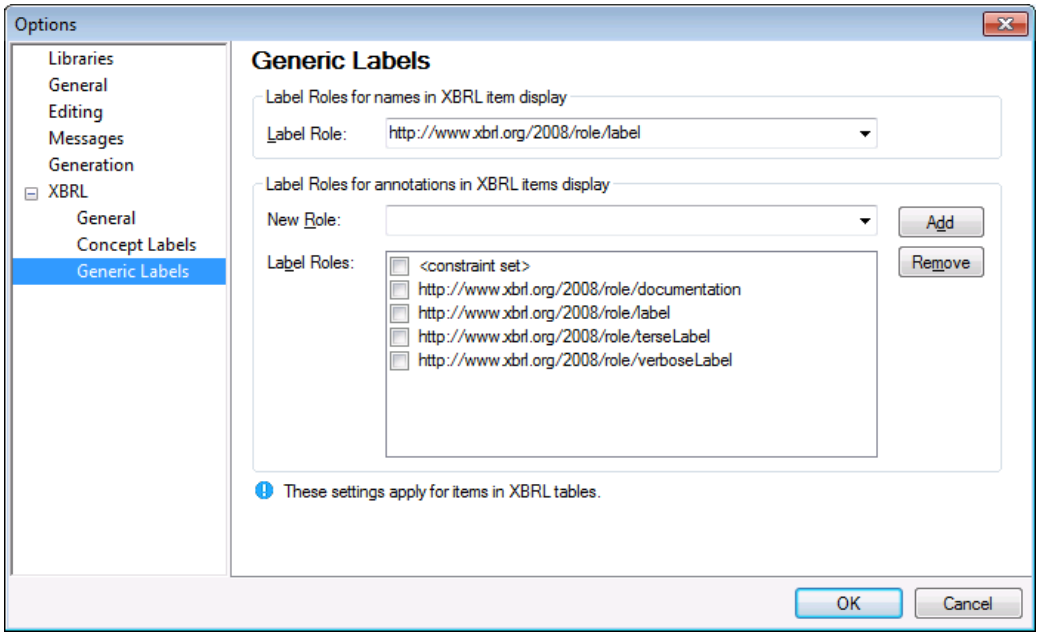
   - Click the **General** node to change the label language of XBRL items and their annotations.



- Click the **Concept Labels** node to change the preferred label roles and annotations applicable for items in the following structured views:
  - *Views from presentation and definition linkbases*
  - *All concepts (with context management)*
  - *All Concepts (raw)*

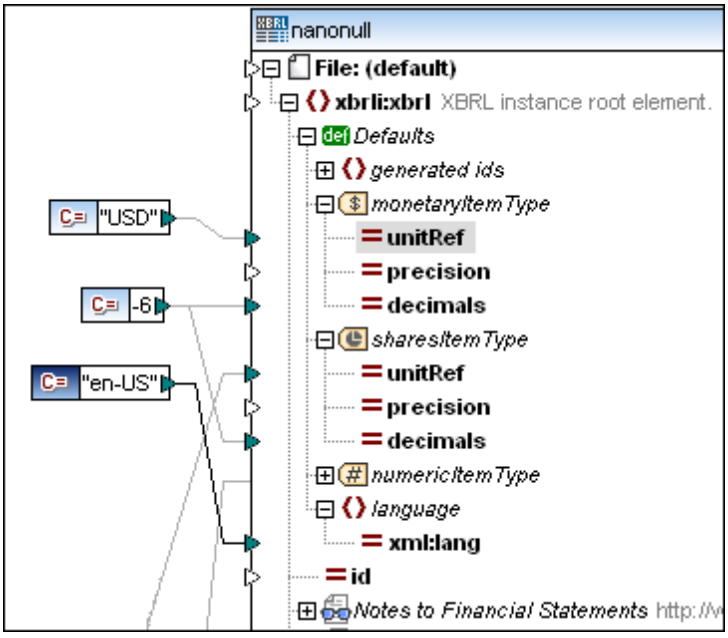


- Click the **Generic Labels** node to change the preferred label roles and annotations applicable for items in the *Tables from table linkbase* view.



7.8.6 Working with XBRL Defaults

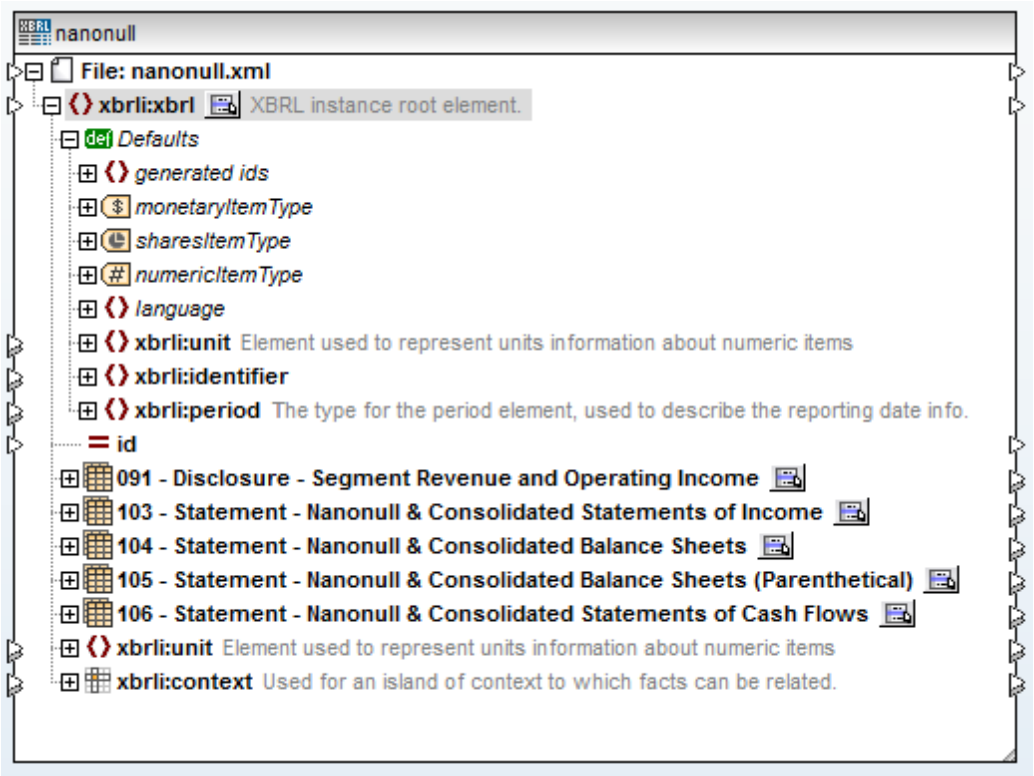
Defaults are a powerful way to assign values to attributes in the XBRL component without having explicit mapping connections to all of them. For example, if you assign a constant unit identifier to the attribute `unitRef` of the `monetaryItemType` item within the Defaults hierarchy (as shown in the following screen shot), this assigns the default to every monetary XBRL item unit identifier, except where its input is mapped explicitly by some other value. The screen shot below also illustrates the use of the default value for the `xml:lang` attribute which defines the language of a footnote.



By default, the "Defaults" node is visible in a new XBRL component; however, you can hide it if

you do not need to map to Defaults. You can display the "Defaults" node at the root level (for the whole XBRL document), or for individual nodes at any hierarchical level of the XBRL component. Note that, since Defaults can be defined at any level in the XBRL structure, different subtrees can have different default values.

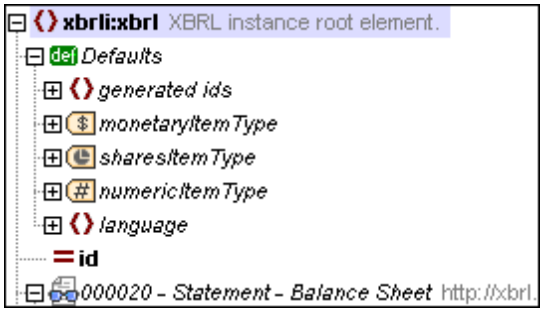
You can also map to aspect value defaults from the table linkbase, if your XBRL component uses the table linkbase view instead of the presentation/definition linkbase views. In the screen shot below, the `xbrli:unit`, `xbrli:identifier` and `xbrli:period` elements are aspect value defaults that you can use when mapping to XBRL tables.



To display the "Defaults" node for a particular item

1. Right click the item for which you want to display the default units, and select **XBRL | Show defaults**. (Alternatively, click the **Show Context Menu** ( ) button if available for the node, and then click **Show Defaults**.)

   This inserts a Defaults item to which you can connect your own default values for the various item types.

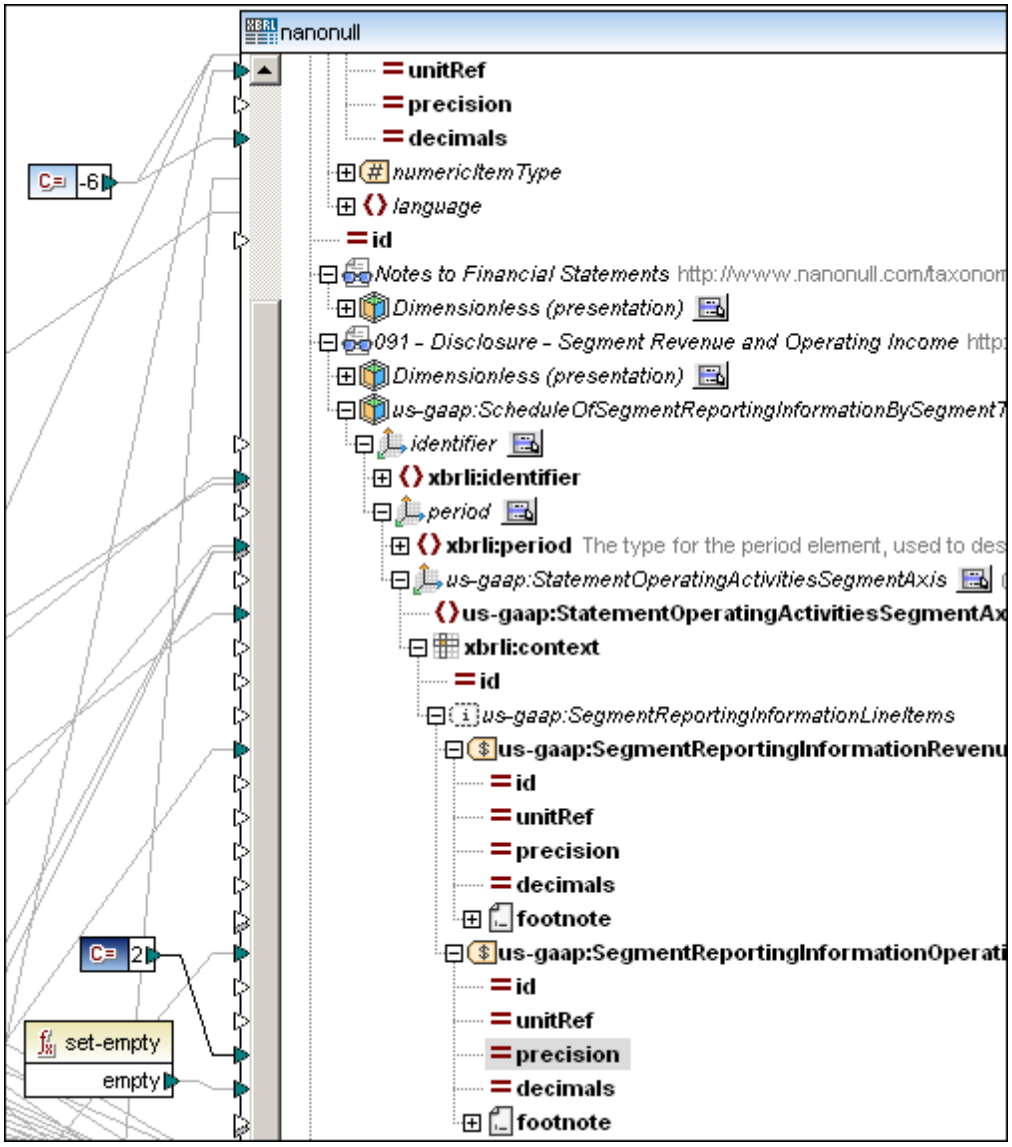


Replacing or de-activating a Default value

If a default value has been defined for some concept attribute, e. g. **decimals**, it is possible to remove this setting locally for each concept, by using the function **set-empty**.

In the screen shot below, the **set-empty** function "deactivates" the default value "-6" for the monetary item "Segment Reporting Information, Revenue".

This item will now be reported with the precision of "2" mapped to the precision attribute, while the other item, "Segment Reporting Information, Operating Income" will be reported with the default decimals value of "-6".



Context handling

The hierarchical structure within XBRL components allows automatic context handling. The generation of the `xbrli:context` in XBRL output instances is done automatically when reporting related concepts.

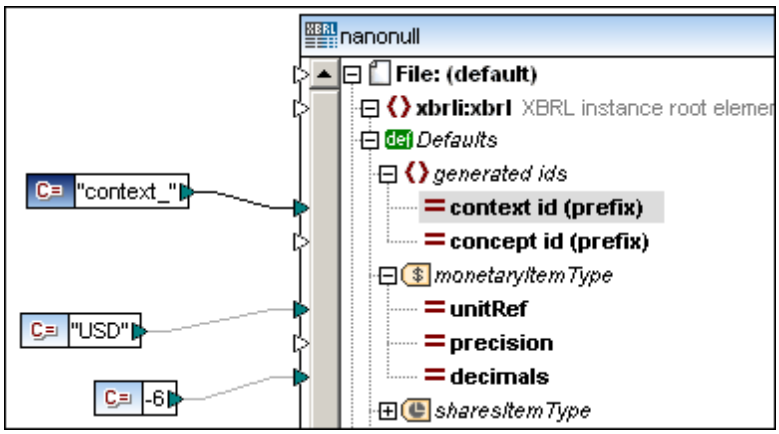
The value of the attribute `id` of a specific context in an XBRL instance is the value of the attribute `contextRef` in each related XBRL concept. MapForce automatically numbers all created contexts in an output instance.

Customization is possible by assigning text as a prefix into the node "context id (prefix)" under "generated ids".

For example, mapping the constant value "**context**_" as a default prefix creates consecutive context-ids in the output instance having the values "context_1", "context_2", "context_3", and so

on.

If no default value is defined, MapForce will create all context-ids with the prefix "**ctx**_".



If the output XBRL instance contains **footnotes**, the related concepts must have concept IDs to link to the automatically generated footnote links. These attributes are automatically generated. The item "**concept id (prefix)**" can be mapped to determine such a prefix.

If the prefix is not mapped within the XBRL component, MapForce will create all concept IDs with the prefix "**fact**_".

7.8.7 Working with XBRL Hypercubes

Hypercube enable automatic context handling. XBRL hypercubes can be of the following types:

- Defined by the taxonomy, e. g. *Statement [Table]*
- Generated by MapForce to simplify the default dimensions identifier and period, derived from the **Presentation** linkbase e. g. *Dimensionless (presentation)*
- Generated by MapForce within the **All concept node** *Dimensionless*

Every hypercube contains two default dimensions, identifier and period, that support the easy reading/writing of these two elements for each context. Additionally defined dimensions in the taxonomy are automatically related to the context elements `xbrli:segment` and `xbrli:scenario`.

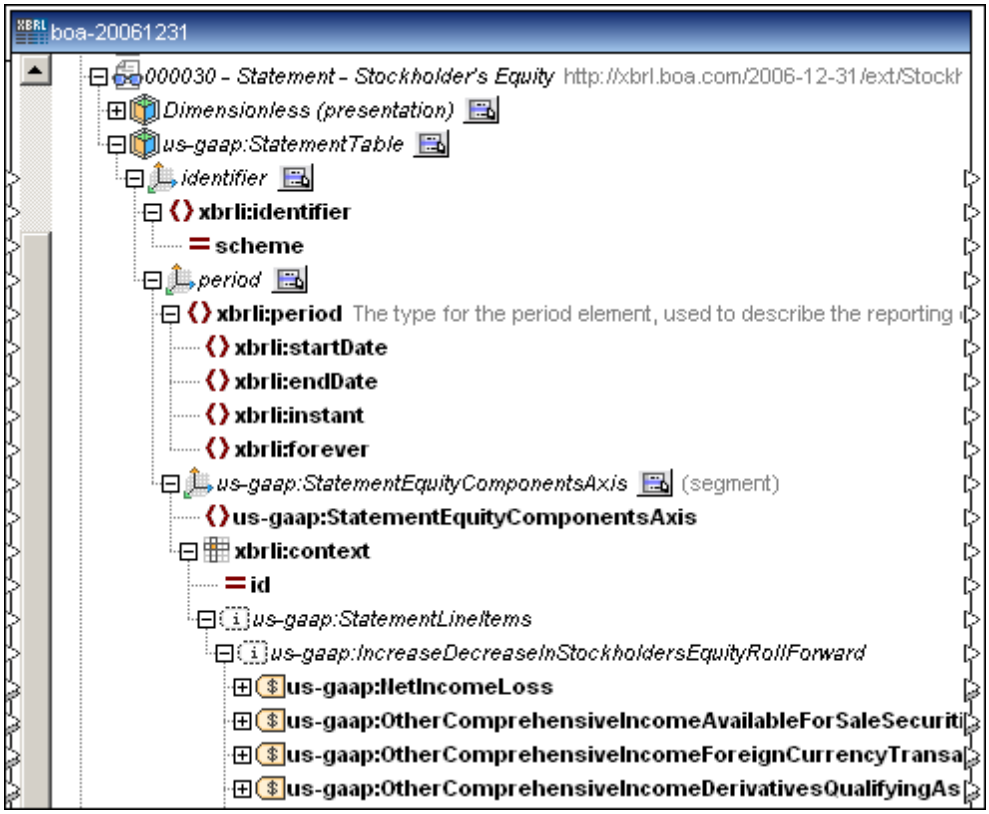
Hypercubes denoted as "**Dimensionless (presentation)**" use both default dimensions. The hierarchical order of concepts shown within its context node is taken from the presentation linkbase.

"**Dimensionless**" hypercube items also use both default dimensions, but do not have any hierarchical concept order and show only the raw list of all concepts defined in the taxonomy.

All other hypercubes are defined within the taxonomy and are designated according the name defined in the Label linkbase of the taxonomy.

Hypercubes as well as their dimensions (or Axes), each have a small icon which opens a pop-up menu allowing you to define the presentation of each of the dimensions in the component. The

screen shot below shows a sample taxonomy file which contains both generated hypercubes and hypercubes defined by the taxonomy.



Note: MapForce shows all hypercubes which have reportable concepts. If one of the related hypercube dimensions has no domain, it is not shown in the XBRL component.

7.8.7.1 Showing Dimensions in a Component

Dimension items in XBRL refer either to **explicit** or **typed** dimension values in the instance. The annotation of each dimension item shows in brackets whether the dimension is reported in the context elements `xbrli:segment` or `xbrli:scenario`.

Typed dimension items show the elements of their XML Schema type as children. Their values can be directly mapped.

Explicit dimensions in an XBRL taxonomy have a value of type `xs:QName` from a certain domain. This comprises of the XBRL domain member values and the value of the XBRL domain item itself. Explicit dimensions can be displayed in two different modes, depending on the mapping requirements and the other component/structure you are mapping to or from XBRL.

Initially, the explicit dimension is displayed with a **single** child node and can be mapped directly using this child, e. g. "Statement, Equity Components [Axis]".

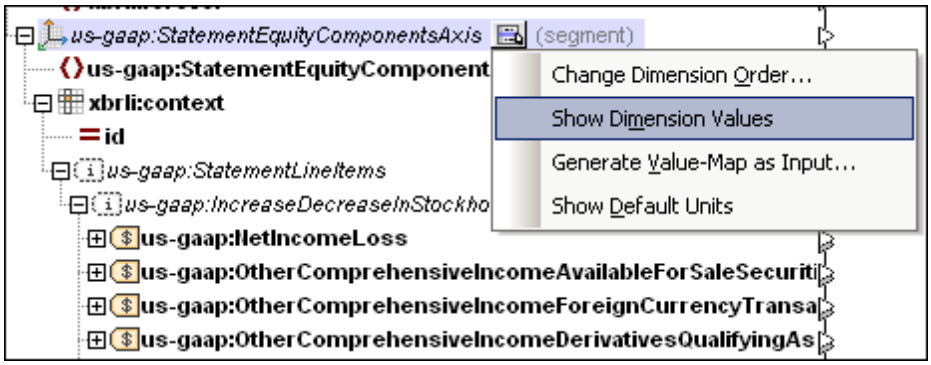
This is useful (for an XBRL target component) when the dimension values can be derived from a field in the source data, e.g. a database field, or a column in an Excel table. As the source data will generally not contain the required `QName` datatype, MapForce can automatically create them

using the value-map function (see Generating Value-Maps for Hypercube Dimensions).

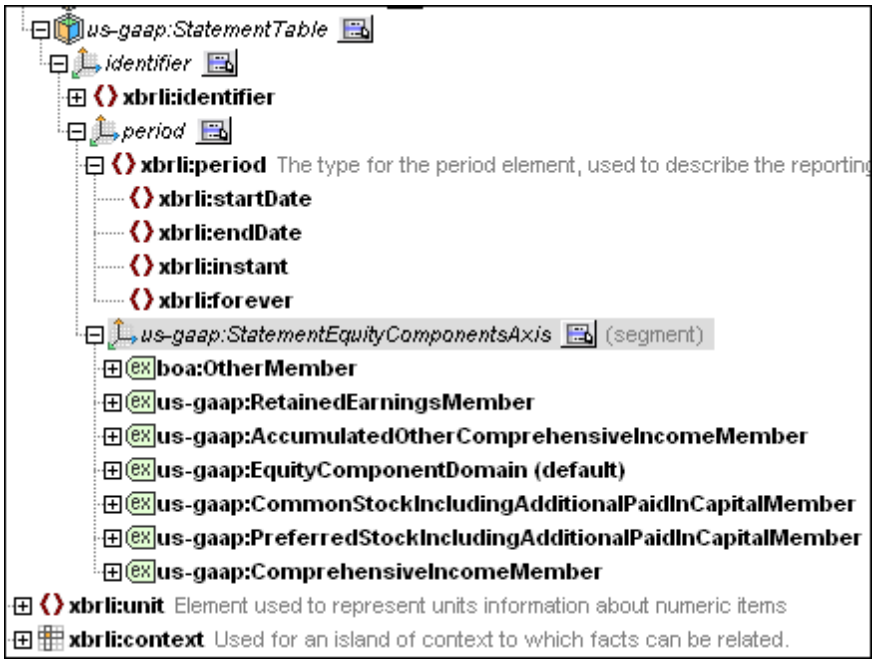
To allow different mappings for the facts related to each dimension member, you can display separate nodes for every single value of the dimension domain.

To show the dimension values in the component

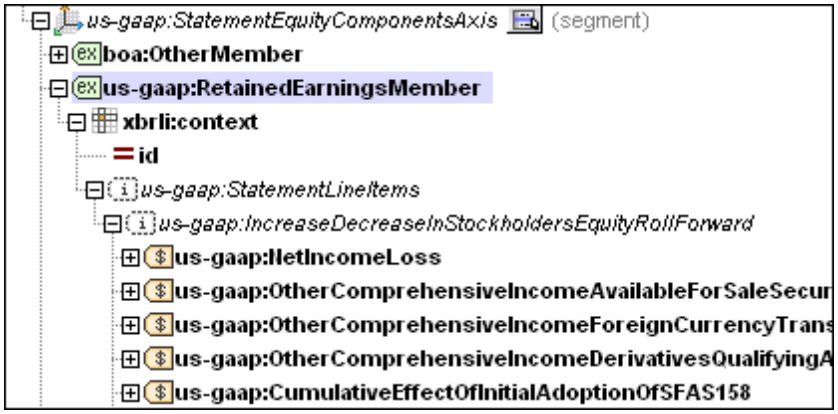
1. Click the icon of the dimension you want to see the values for, and select "Show Dimension Values".



This changes the items visible below the dimension name. The dimension **domain** and **member** items are now visible, each with a light green icon. These are all explicit members of a domain which is shown by the "ex" prefix in the item icon.



Each explicit member will now contain the same substructure, allowing different mappings for each.



When the output of a concept is mapped, only those values are used for which the related context element has the appropriate dimension value, e.g. the value of "Net Income (Loss)" in the instance, is mapped only for contexts which contain the dimension value "Comprehensive Income [Member]" for the dimension "Statement, Equity Components [Axis]". There is no additional filtering required.

When writing XBRL output instances, the automatic generation of proper dimension values within the context is supported. E.g. for every reported monetary item "Net Income (Loss)", the context node xbrli:context, acquires within its context element (xbrli:segment), an element for the explicit dimension "Statement, Equity Components [Axis]" containing the value "Comprehensive Income [Member]".

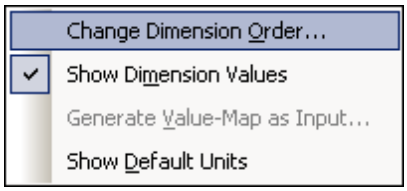
7.8.7.2 Changing the Order of Dimensions

Initially, MapForce displays all dimensions of a hypercube as nested child nodes, automatically creating a hierarchy. The hierarchical order of dimensions within the hypercube can be changed to match the other (non-XBRL) side of the mapping.

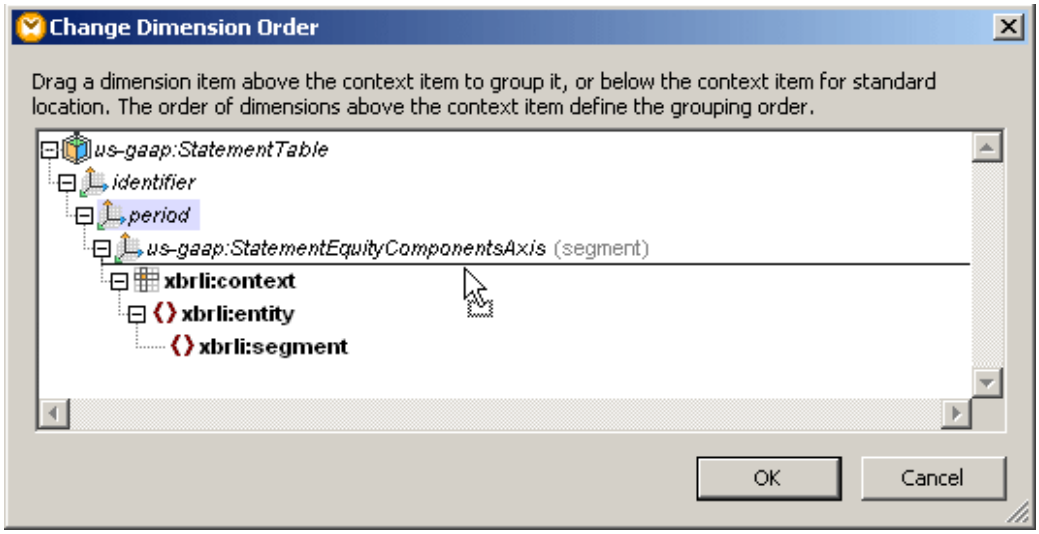
Furthermore, where dimension values have to be set specifically for some concepts, MapForce is able to display a dimension, without a hierarchy, and show it as a child element of the context node.

To change the order of dimension items

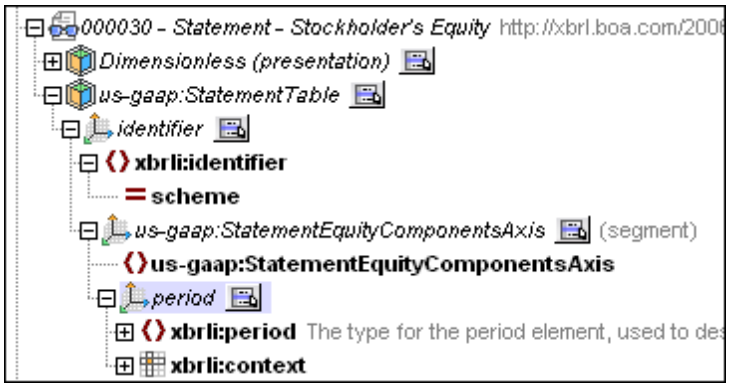
1. Click any one of the icons of the respective hypercube, and select "Change Dimension Order" entry in the popup menu.



   This opens a dialog box allowing you to reposition the various dimensions of a hypercube. Note that a hypercube has two **default dimensions**: **identifier** and **period** whose order in the hypercube can also be changed.
2. Click the hypercube dimension and use drag-and-drop to reposition it in the dialog box.



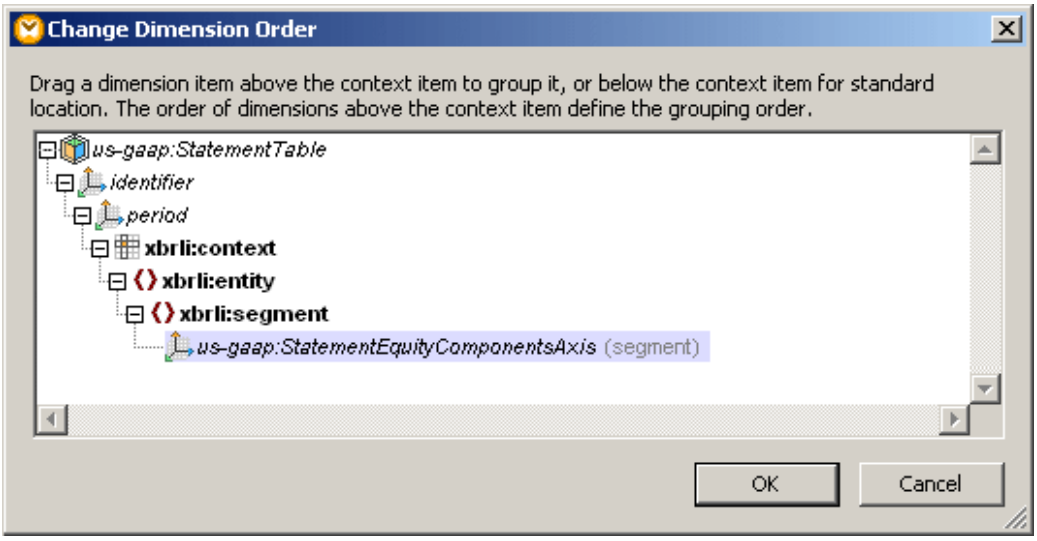
A line appears at a position where the dimension can be dropped.



3. Click OK to close the dialog and have the dimension repositioned in the component.

To exclude a dimension from the hierarchy

- Drag the dimension below the **xbrli:context** line, which will insert it into its context item.

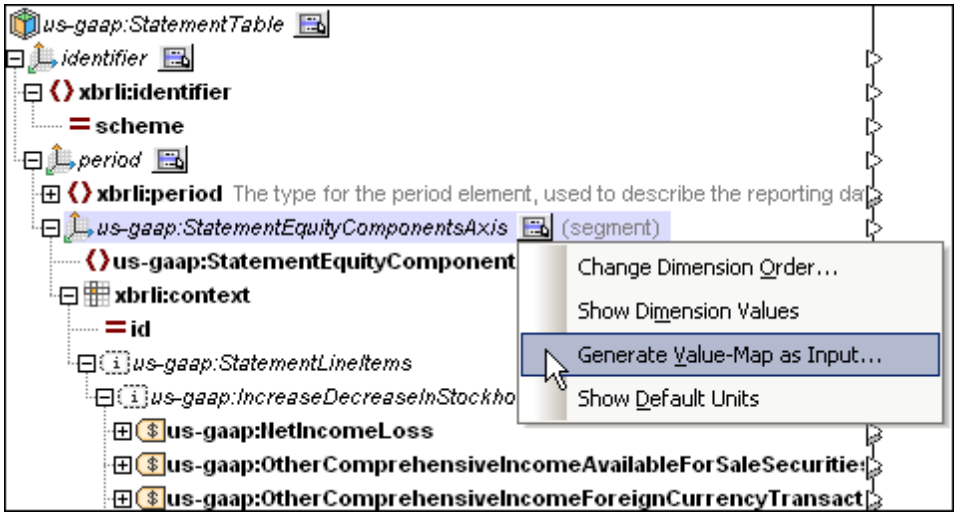


7.8.7.3 Generating Value-Maps for Hypercube Explicit Dimensions

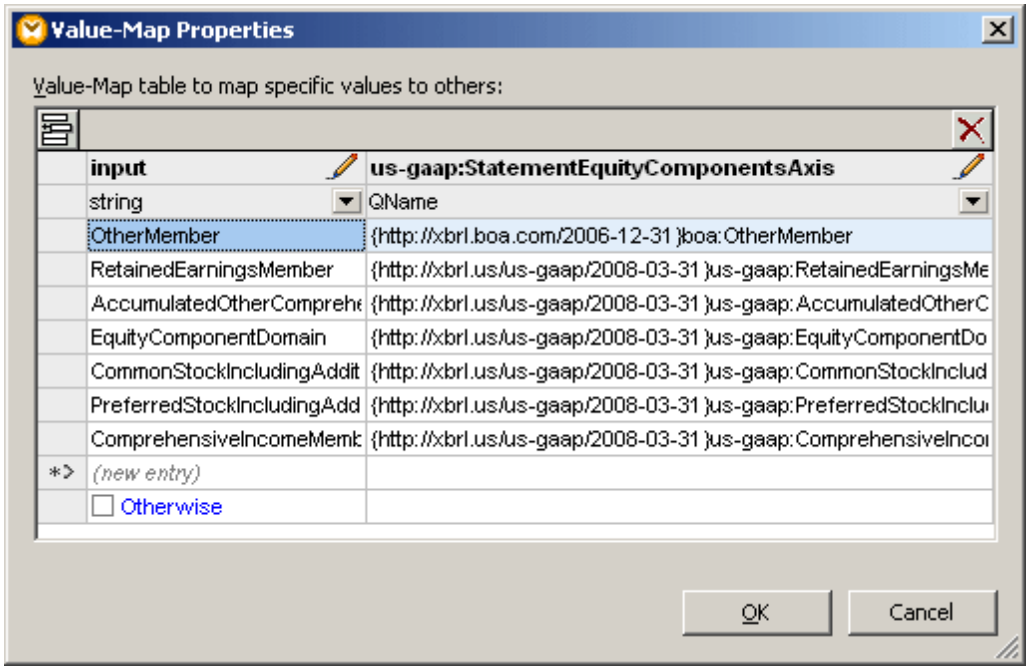
This option transforms input data of any type into a valid `QName` in the target component. In other words, the input string is converted into the prefixed name (`QName`).

To generate a value-map lookup table for the selected hypercube dimension

1. Make sure that the dimension values are not visible for the specific hypercube dimension.
2. Click the icon of the specific **dimension** and select the "Generate Value-Map as input" option in the popup menu.

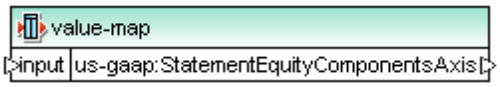


This opens the Value-Map Properties dialog box containing automatically generated input and output values based on the dimension default domain and domain members, as defined in the taxonomy.



3. Edit the input values if necessary, and click OK to insert the value-map component. Note that you can edit the column header text (by double clicking the header) to make them shorter, or more descriptive if you wish.

   This inserts the value-map function, showing the input and output parameter names. The output connector is automatically connected to the domain element of the target.



4. Connect the source item that contains the input data to be transformed, to the input parameter of the function.

For a general example of how to use the Value-Map function, see Value-Map - transforming input data.

7.8.8 Working with XBRL Tables

If your XBRL taxonomy references the table linkbase (see Table Linkbase Specification 1.0), MapForce can be configured to display the node types applicable to the table linkbase (such as tables or breakdowns). This enables you to map data to or from facts displayed in rendered tables. (For instructions on how to configure MapForce to display the information from the table linkbase, see Selecting Structure Views.)

To render information from the table linkbase, MapForce normally uses the structural model, with the following exceptions:

- Merged rule nodes are visible
- Roll-up nodes without siblings do not form a separate hierarchy level.

The structural model is one of the three data models defined by the Table Linkbase Specification 1.0, §4. To identify the parts of the structural model referenced by the mapping, MapForce also uses information from the definition model.

Since a MapForce component structure is one-dimensional, whereas tables can have up to three dimensions (x, y, z), MapForce displays table dimensions by nesting all breakdowns within each other, in the following default order: z, y, x. As such, the purpose of breakdowns nodes in MapForce is to inform you where each breakdown starts.

This section includes instructions on how to work with entities from the table linkbase, as follows:

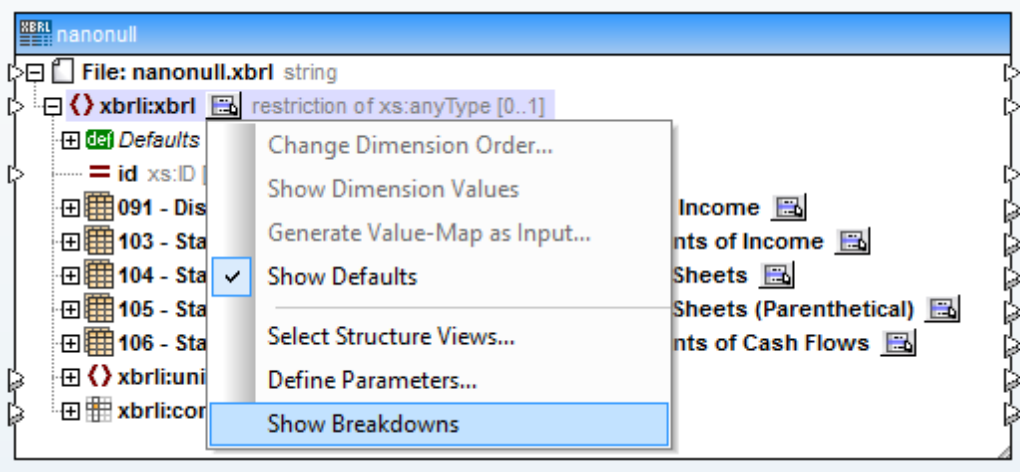
- Showing or Hiding Breakdowns
- Changing the Order of Breakdowns
- Working with Parameters

7.8.8.1 Showing or Hiding Breakdowns

In XBRL documents that reference the table linkbase, you can select whether to view or hide breakdowns in the component.

To show or hide breakdowns

1. In the XBRL component, locate a node that contains the **Show Context Menu** ( ) button to the right (for example, the root node).



2. Click the **Show Context Menu** ( ) button, and then click **Show Breakdowns**.

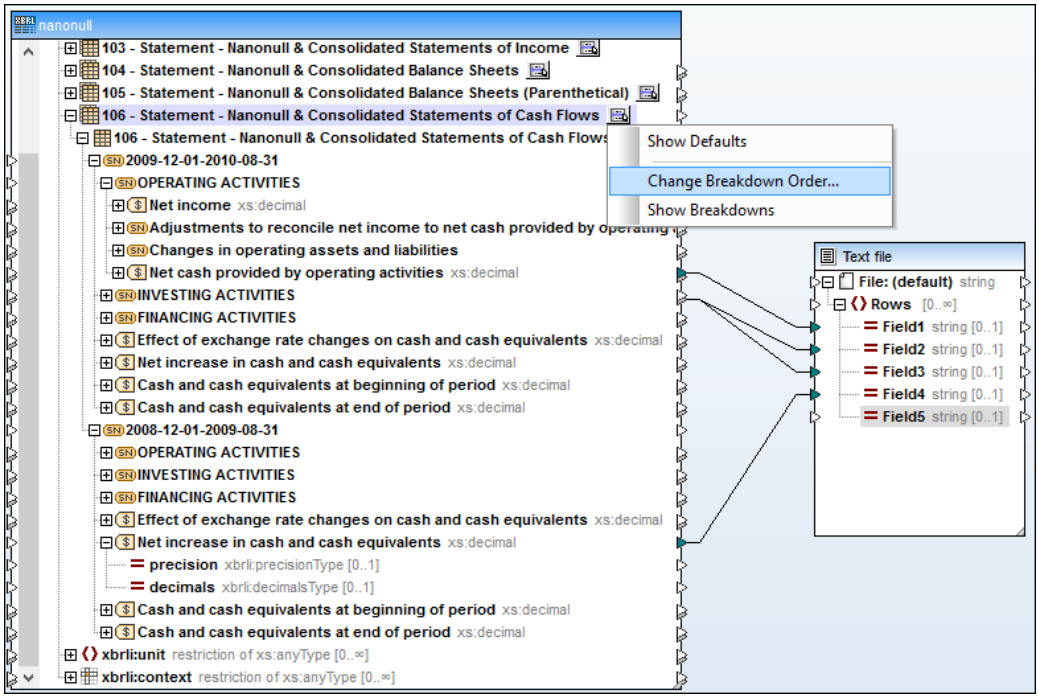
7.8.8.2 Changing the Order of Breakdowns

In XBRL documents that reference the table linkbase, you can change the order of breakdowns.

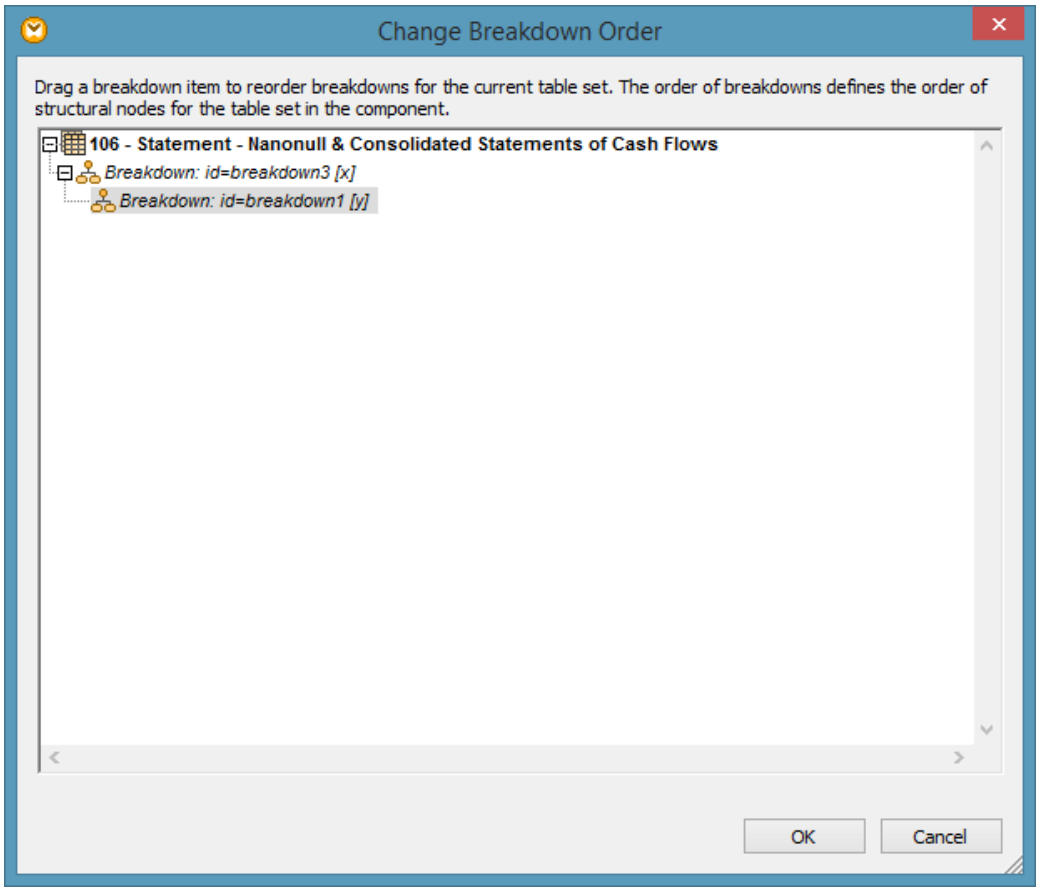
To change the order of breakdowns

1. On the XBRL component, locate the node for which you want to change the breakdown order.

2. Click the **Show Context Menu** ( ) button to the right of the node, and then click **Change Breakdown Order**. (If the **Change Breakdown Order** menu option is not available, this means that this option is not meaningful for the selected node.)



3. Drag the breakdowns to the desired location (for example, in the following screen shot, you can drag the *y* breakdown on top of the *x* breakdown).



4. Click **OK**.

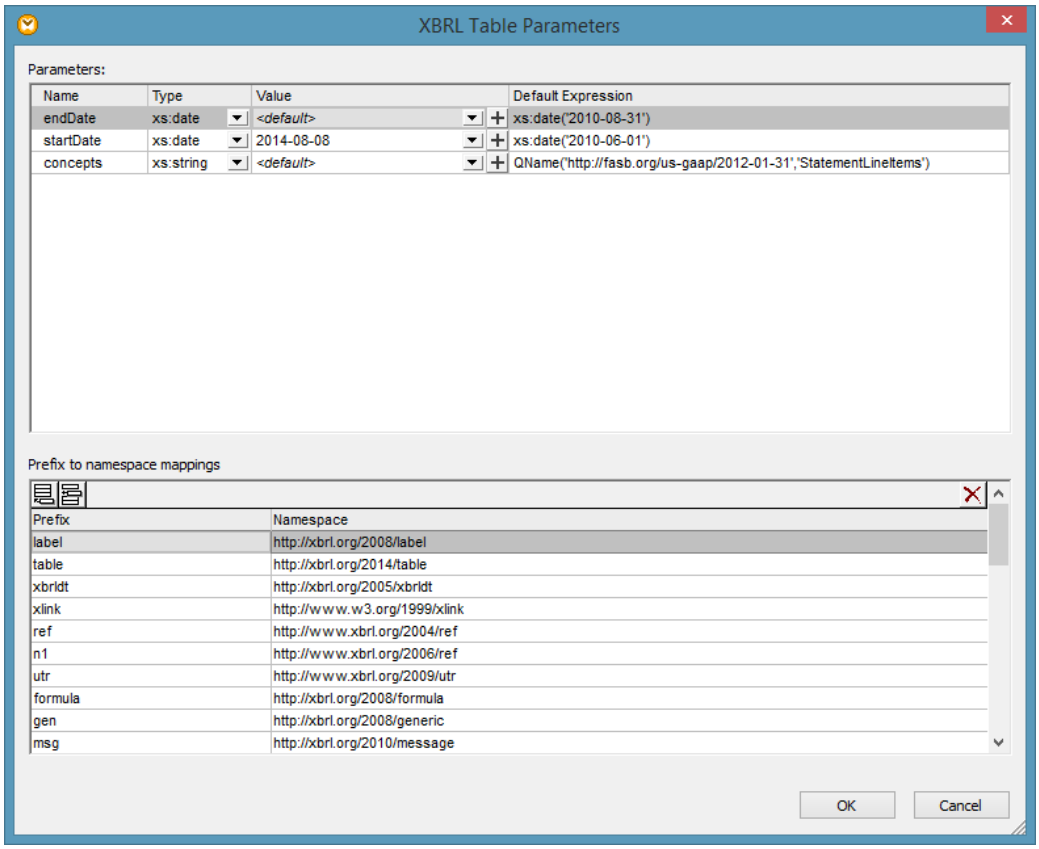
7.8.8.3 *Working with Parameters*

In XBRL documents that reference the table linkbase and contain parameters (Table Linkbase Specification 1.0, §5.3), you can change the parameter values. If supported by the context, you can also append new parameter values to existing parameters.

You can change parameter values from the XBRL Table Parameters dialog box (see instructions below). This dialog box displays parameters defined both at global level (anywhere in the Discoverable Taxonomy Set) and in individual tables. When you change any parameter value and close the dialog box, MapForce attempts to regenerate all tables in the component and merge the structure tree nodes. If the structure cannot be regenerated due to missing parameters, an error node is displayed in the relevant node of the component. For more detailed information about the error, check the Messages window.

To change parameter values

1. Click the **Show Context Menu** ( ) button to the right of the root node, and then click **Define Parameters**. (If the **Define Parameters** menu option is not available, the table linkbase does not contain parameters.)



2. Select the parameter record, and type the new value in the Value column.

To append parameter values to existing parameters

Select the parameter record, and then click the **Add Value** ( ) button under the Value column.

To remove an appended value

Click the **Remove Value** ( ) button next to the value you want to remove.

7.8.9 XBRL Mapping Examples

This section includes the following XBRL mapping examples:

- Microsoft Access to XBRL (shows how data from a Microsoft Access database is mapped to an XBRL taxonomy, producing a valid XBRL instance file)
- Microsoft Excel to XBRL (shows how Microsoft Excel spreadsheet data is mapped to a taxonomy, producing a valid XBRL instance file).

7.8.9.1 Microsoft Access to XBRL

This example is available as **DB_to_XBRL.mfd** in the ...\MapForceExamples folder, and uses various filters and functions to extract the database data.

The taxonomy **nanonull.xsd** is derived from US:GAAP. The mapping creates an XBRL output instance which contains all contexts, concepts, units and footnotes for one Disclosure and three Statements.

The report **"091 - Disclosure - Segment Revenue and Operating Income"** shows how MapForce can map dimension values. The hypercube "us-gaap:ScheduleOfSegmentReportingInformationBySegmentTable" contains an explicit dimension "us-gaap:StatementOperatingActivitiesSegmentAxis".

Its domain has been extended in the taxonomy by the three dimension values "nanonull:USA", "nanonull:Europe" and "nanonull:Asia" The mapping shows how a value-map maps the values of the database column "Name" of the table "Region" to the required dimension values of type QName.

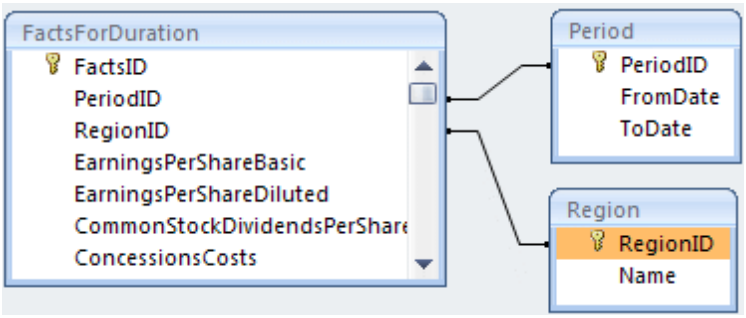
The report **"106 - Statement - Nanonull and Consolidated Statement of Cash Flows"** illustrates how MapForce can be used to write facts into the output instance which relates to both duration and instant periods.

As the mapping shows, proper reporting of facts such as "Cash and cash equivalents at beginning (end) of period" can be achieved by duplicating the period item in the hierarchy structure.

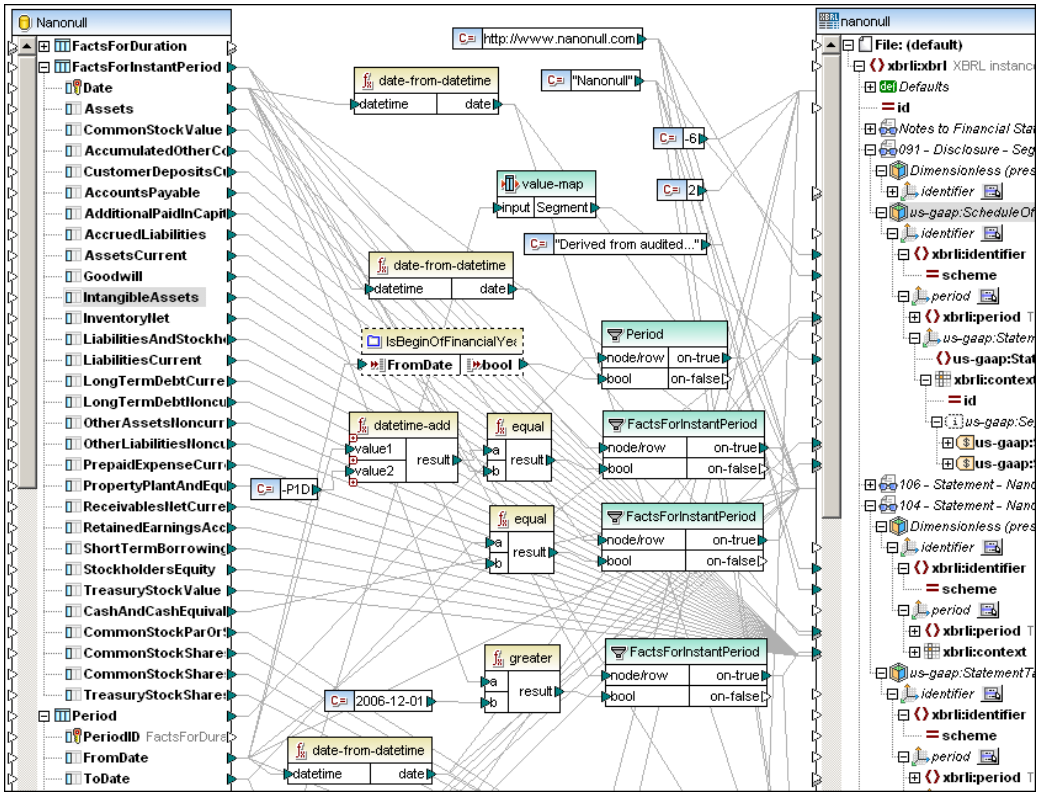
The mapping defines two units in the output instance, "USD" and "perShareItemType". The xbrli:unit element must be duplicated to do this. The related measure elements are created using the functions "xbrl-measure-currency" and "xbrl-measure-shares" from the XBRL library.

The facts in the database tables have been split up depending on whether they relate to an instant or duration period.

- The table **FactsForInstantPeriod** is a flat table of values.
- The table **FactsForDuration** is hierarchical and each fact it contains, relates to a specific PeriodID as well as a RegionID.



The Period table uses FromDate and ToDate fields to define the start and end period dates; while the Region table relates each of the facts to a specific region, i.e. Asia, Europe or USA.



Mandatory XBRL items needed in a XBRL instance file:

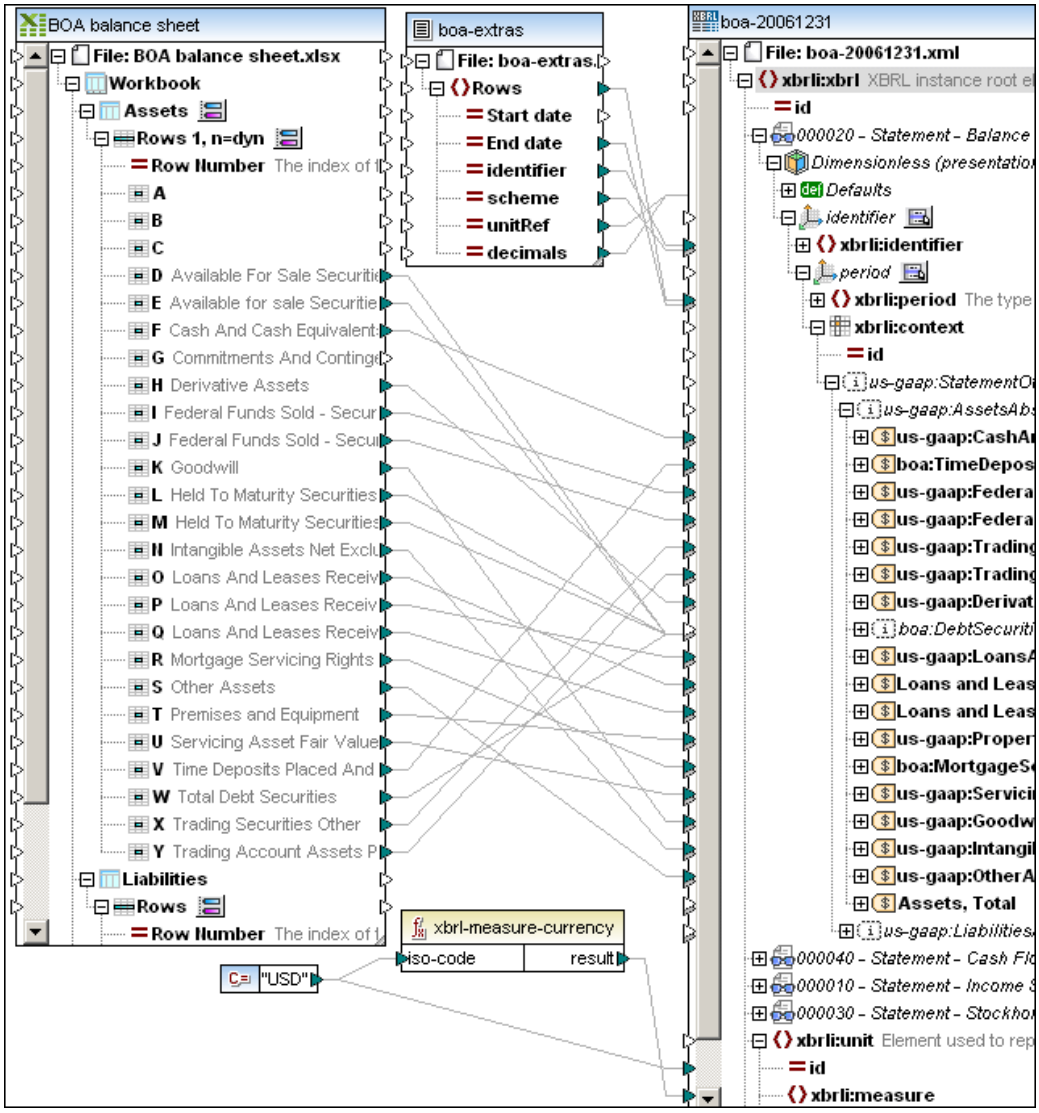
- **unitRef** and either **decimals** or **precision** in monetary concepts
- xbrli:**identifier** and **scheme** of the identifier dimension
- xbrli:**period** and either xbrli:**instant** or xbrli:**startDate**/xbrli:**endDate** elements
- xbrli:**id** and xbrli:**measure** in the xbrli:**unit** element

See also Microsoft Excel to XBRL.

7.8.9.2 Microsoft Excel to XBRL

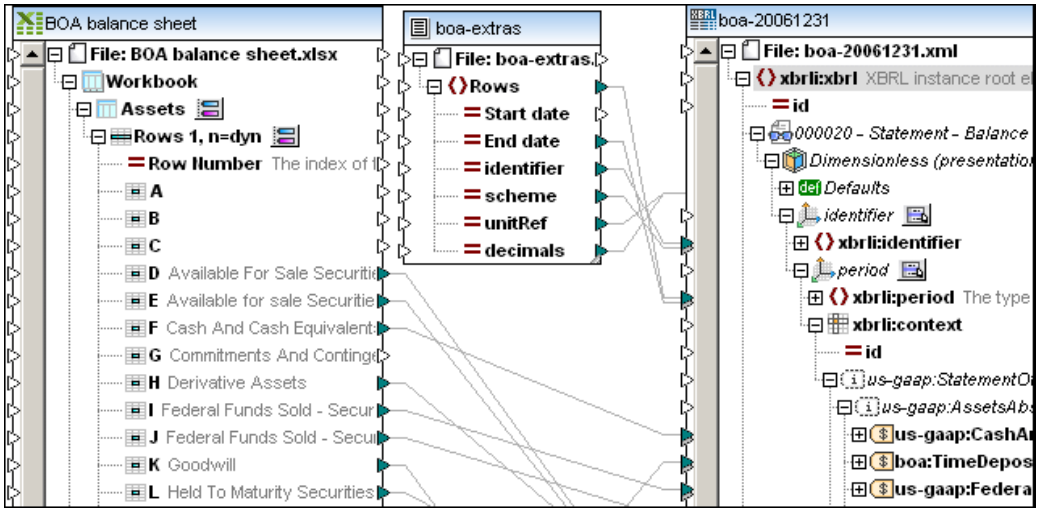
The mapping file used in the following example is available as **boa-balance-sheet.mfd** in the ...\MapForceExamples\Tutorial\ folder. The taxonomy files used in the example are available in the ..\Tutorial folder.

The example shows how data from an Excel sheet are mapped to an XBRL taxonomy to generate an XBRL instance file. Only data from the "Assets" worksheet have been mapped to keep the example simple. The result of the mapping is a valid XBRL instance document containing the Assets data for a particular instant, Dec. 31st 2006.



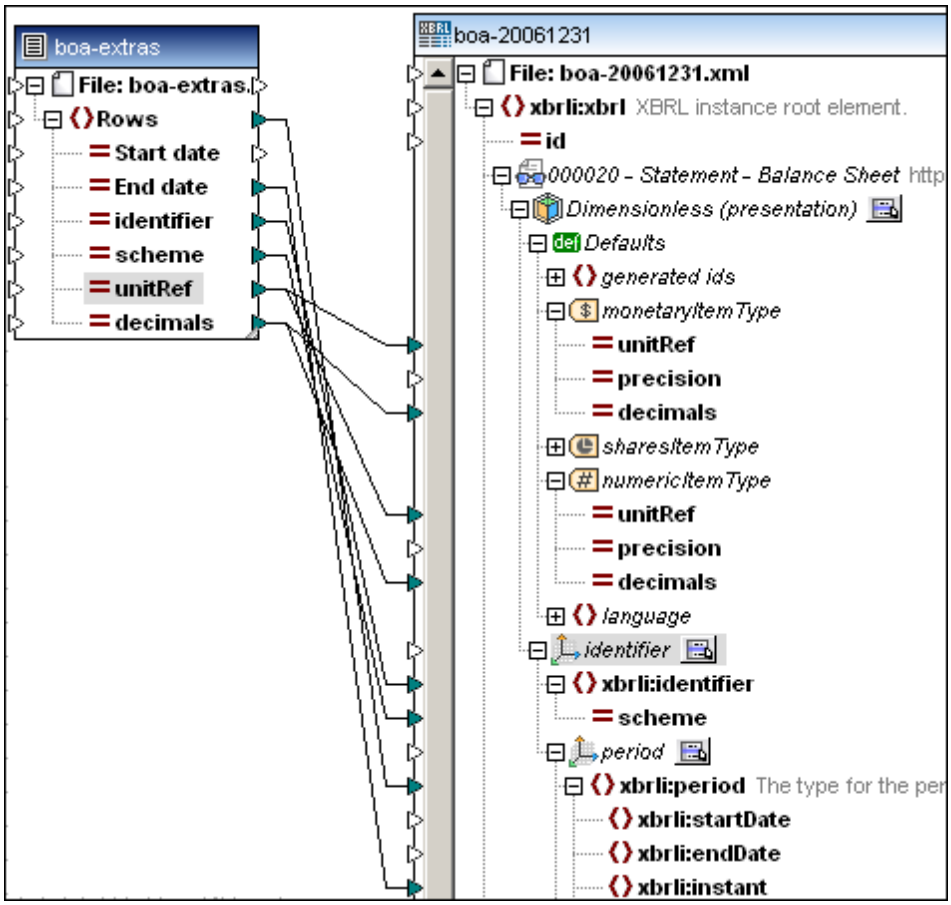
This example shows that the Excel columns, which are in alphabetical sort order, are mapped directly to the XBRL items/facts in the taxonomy component e.g. Cash and Cash Equivalents.

Note that the item names in the source and target components are not, and do not need to be, identical. The name of the target taxonomy item determines the name used in the resulting XBRL instance. However, having identical source and target item names does have the advantage that you can use the **Autoconnect Matching Children** option (see Connecting matching children).



There are a number of items that must be mapped to be able to generate a valid XBRL instance file. Some mandatory items are supplied by a text file (boa-extras) whose fields are mapped to child elements of the "Defaults" item in the Dimensionless (presentation) hypercube.

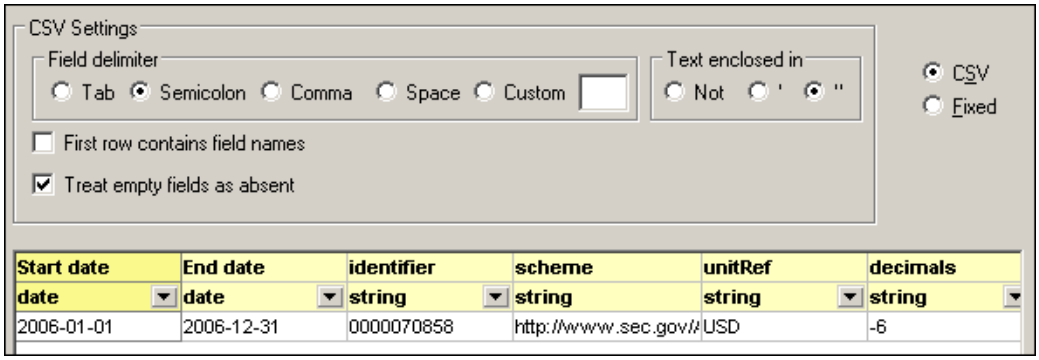
The Defaults item is inserted by right clicking a hypercube dimension and selecting **XBRL | Show default units,** or by clicking the icon and selecting the option there.



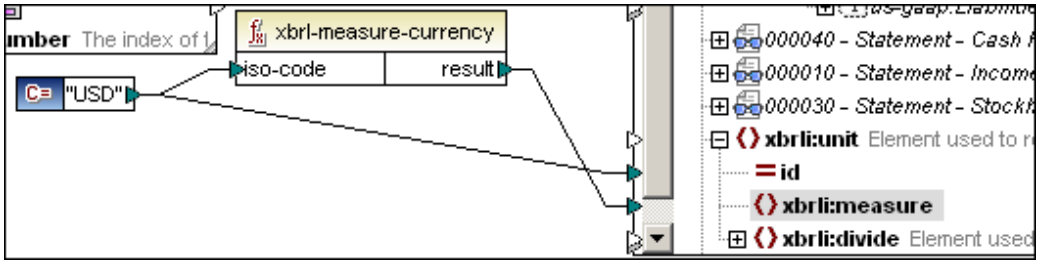
Mandatory XBRL items needed in a XBRL instance file:

- **unitRef** and either **decimals** or **precision** in monetary concepts
- xbrli:**identifier** and **scheme** of the identifier dimension
- xbrli:**period** and either xbrli:**instant** or xbrli:**startDate**/xbrli:**endDate** elements
- xbrli:**id** and xbrli:**measure** in the xbrli:**unit** element

The boa-extras text file supplies the data for some of the mandatory items, their values are shown below.



A constant component supplies the data for the **id** and **measure** items, which are at the base of the taxonomy component.



Clicking the **Output** button shows the resulting XBRL instance file.

- Clicking the "Validate Output" button of the Output toolbar, allows you to check the validity of the XBRL instance. Messages or warnings are displayed in the Messages window
- Clicking the Text view settings button allows you to define the output view settings


```
<?xml version="1.0" encoding="UTF-8"?>
<xbrl xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xl="http://www.xbrl.o
  <link:schemaRef xlink:type="simple" xlink:href="A:/AA-tutorial-xbrl-example/boa-20061231.xsd" />
  <context id="ctx1">
    <entity>
      <identifier scheme="http://www.sec.gov/Archives/edgar/data/70858/000119312507042036/d10k.htm">0000070858</iden
    </entity>
    <period>
      <instant>2006-12-31</instant>
    </period>
  </context>
  <us-gaap:CashAndCashEquivalentsAtCarryingValue contextRef="ctx1" unitRef="USD" decimals="-6">36429</us-gaap:CashAn
  <boa:TimeDepositsPlacedAndOtherShorttermInvestments contextRef="ctx1" unitRef="USD" decimals="-6">13952</boa:TimeDep
  <us-gaap:FederalFundsSoldAndSecuritiesPurchasedUnderAgreementsToResell contextRef="ctx1" unitRef="USD" decimals="-6
  <us-gaap:FederalFundsSoldAndSecuritiesPurchasedUnderAgreementsToResellPledgedAsCollateral contextRef="ctx1" unitRef=
  <us-gaap:TradingSecuritiesOther contextRef="ctx1" unitRef="USD" decimals="-6">153052</us-gaap:TradingSecuritiesOther>
  <us-gaap:TradingSecuritiesPledgedAsCollateral contextRef="ctx1" unitRef="USD" decimals="-6">922274</us-gaap:TradingSecu
  <us-gaap:DerivativeAssets contextRef="ctx1" unitRef="USD" decimals="-6">23439</us-gaap:DerivativeAssets>
  <us-gaap:AvailableForSaleSecuritiesDebtSecurities contextRef="ctx1" unitRef="USD" decimals="-6">192806</us-gaap:Availabl
  <boa:AvailableforsaleSecuritiesDebtSecuritiesCollateral contextRef="ctx1" unitRef="USD" decimals="-6">83875</boa:Availablef
  <us-gaap:HeldToMaturitySecurities contextRef="ctx1" unitRef="USD" decimals="-6">40</us-gaap:HeldToMaturitySecurities>
  <us-gaap:HeldToMaturitySecuritiesFairValue contextRef="ctx1" unitRef="USD" decimals="-6">40</us-gaap:HeldToMaturitySecu
  <boa:TotalDebtSecurities contextRef="ctx1" unitRef="USD" decimals="-6">192846</boa:TotalDebtSecurities>
  <us-gaap:LoansAndLeasesReceivableGrossCarryingAmount contextRef="ctx1" unitRef="USD" decimals="-6">706490</us-gaa
  <us-gaap:LoansAndLeasesReceivableAllowance contextRef="ctx1" unitRef="USD" decimals="-6">9016</us-gaap:LoansAndLe
  <us-gaap:LoansAndLeasesReceivableNetReportedAmount contextRef="ctx1" unitRef="USD" decimals="-6">697474</us-gaap:
  <us-gaap:PropertyPlantAndEquipmentNet contextRef="ctx1" unitRef="USD" decimals="-6">9255</us-gaap:PropertyPlantAndEqu
  <boa:MortgageServicingRights contextRef="ctx1" unitRef="USD" decimals="-6">3045</boa:MortgageServicingRights>
  <us-gaap:ServicingAssetAtFairValueAmount contextRef="ctx1" unitRef="USD" decimals="-6">20</us-gaap:ServicingAssetAtF
  <us-gaap:Goodwill contextRef="ctx1" unitRef="USD" decimals="-6">65662</us-gaap:Goodwill>
  <us-gaap:IntangibleAssetsNetExcludingGoodwill contextRef="ctx1" unitRef="USD" decimals="-6">9422</us-gaap:IntangibleAss
  <us-gaap:OtherAssets contextRef="ctx1" unitRef="USD" decimals="-6">119683</us-gaap:OtherAssets>
  <unit id="USD">
    <measure xmlns:iso4217="http://www.xbrl.org/2003/iso4217">iso4217:USD</measure>
  </unit>
</xbrl>
```

7.9 HL7 v3.x to/from XML schema mapping

Support for HL7 version **3.x** is automatically included in MapForce 2016 as it is XML based.

A separate installer for the HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the Libraries page of the Altova website (http://www.altova.com/components_mapforce.html) Select the Custom Setup in the installer, to only install the HL7 V3 components and XML Schemas.

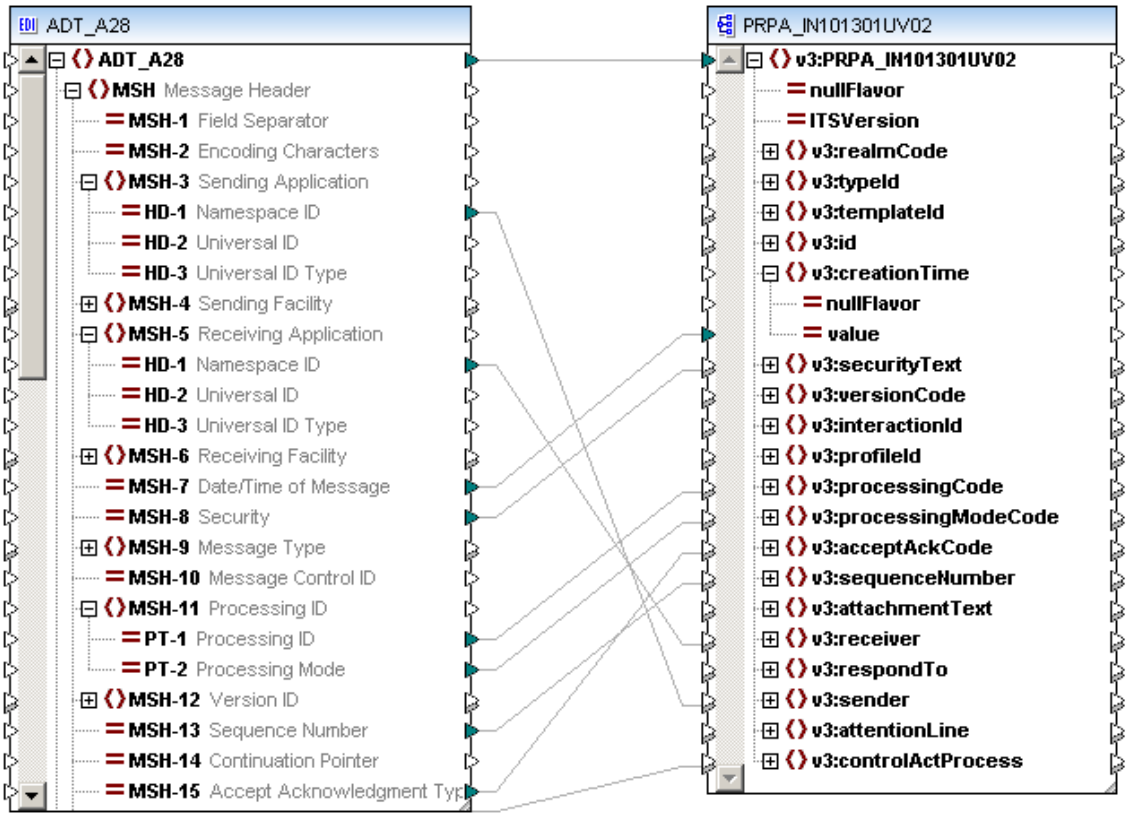
Location of HL7 XML Schemas after installation:

32-bit MapForce on 32-bit operating system, or 64-bit MapForce on 64-bit operating system	C:\Program Files\Altova\Common2016\Schemas\hl7v3
32-bit MapForce on 64-bit operating system	C:\Program Files(x86)\Altova\Common2016\Schemas \hl7v3

HL7 documents can be used as source and target components in MapForce. This data can also be mapped to any number of XML schema, database , EDI or other components.

This example, HL7V260_To_HL7V3.mfd available in the ...\MapForceExamples folder, partially maps an HL7 V2.6 document to an HL7 V3 XML-based document.

- The ADT_A28 EDI file is available in the ...**MapForceExamples** folder.
- The PRPA_IN101301UV02.xsd target XSD file is available in the ...**MapForceExamples \HL7V3_Example_Schemas** folder.



The resulting XML instance file is shown below.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <PRPA_IN101301UV02 xsi:schemaLocation="urn:hl7-org:v3
    C:/DOCUME~1/MY/MYDOCU~1/Altova/MapForce2009/MapForceExamples/HL7V3_Example_Schemas/
    http://www.w3.org/2001/XMLSchema" xmlns="urn:hl7-org:v3" xmlns:xsi="http://www.w3.org/2001/
3     <creationTime value="20070802153810000000000" />
4     <securityText>
5       <reference />
6     </securityText>
7     <processingCode code="P" />
8     <processingModeCode />
9     <acceptAckCode code="AL" />
10    <sequenceNumber />
11    <receiver>
12      <typeId assigningAuthorityName="Therapies" />
13    </receiver>
14    <sender>
15      <typeId assigningAuthorityName="RHAPSODY" />
16    </sender>
17    <controlActProcess>
18      <effectiveTime>
19        <high value="20070802153810000000000" />
20      </effectiveTime>
21      <authorOrPerformer>
22        <time>
23          <low />
24          <high value="20070802153810000000000" />
25        </time>
26      </authorOrPerformer>
```

Chapter 8

Functions

8 Functions

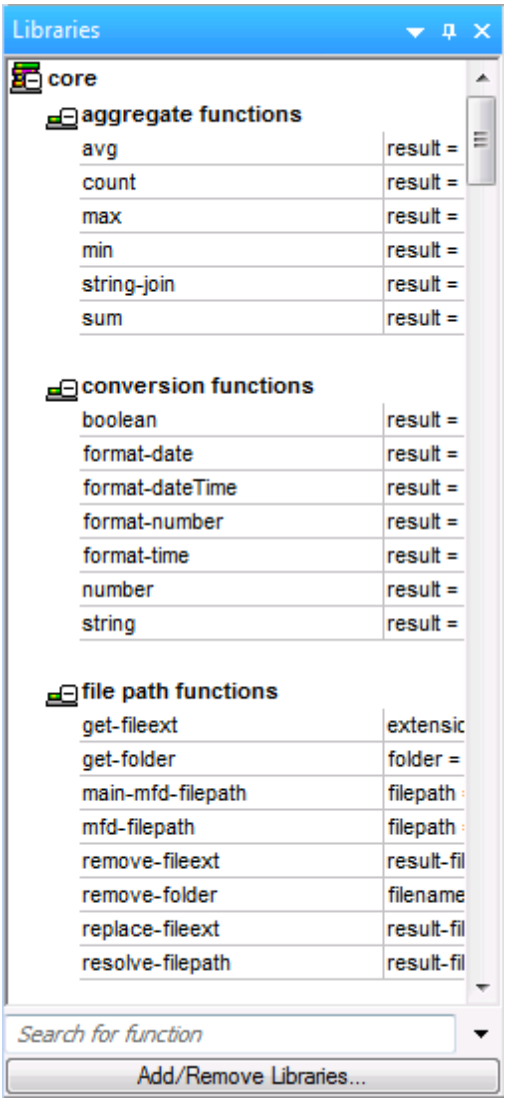
Functions represent a powerful way to transform data according to your specific needs. This section provides instructions on working with functions (regardless if they are built-in to MapForce, defined by you, or reused from external sources). Use the following roadmap for quick access to specific tasks related to functions:

I want to...	Read this topic...
Learn how to work with functions in MapForce	Working with Functions
Learn how to change the priority of operations inside a mapping	Priority Context node/item
Create my own functions in MapForce	User-Defined Functions
Use external XSLT functions	Adding custom XSLT functions
Use external .NET .dll and Java .class libraries	Adding custom Java, C# and C++ function libraries
Write my own Java library for use with MapForce	Create a Java library
Write my own C# library for use with MapForce	Create a C# library
Write my own C++ library for use with MapForce	Create a C++ library
View all built-in MapForce functions, or look up the description of a specific function.	Function Library Reference

8.1 Working with Functions

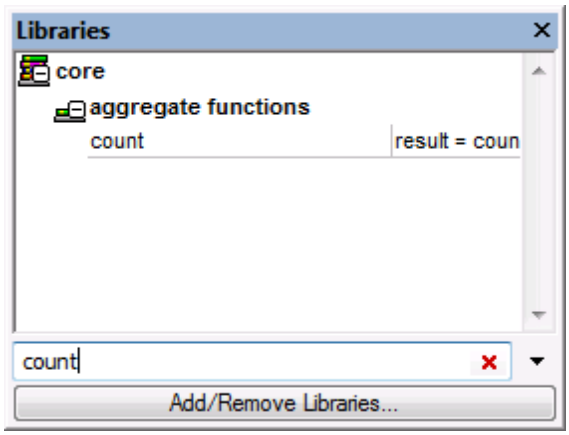
To use a function in a mapping:

1. Select the transformation language (see Selecting a transformation language).
2. Click the required function in the Libraries window and drag it to the mapping area.

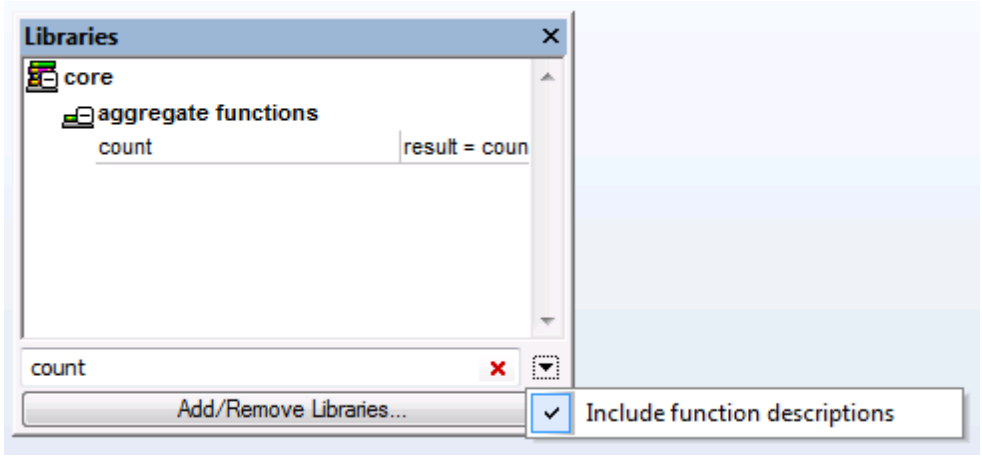


To search for a function in the Libraries window:

1. Start typing the function name in the text box located in the lower part of the Libraries window.



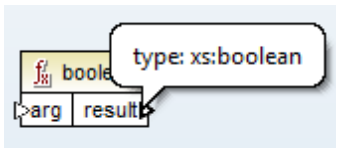
By default, MapForce searches by function name and description text. If you want to exclude the function description from the search, click the down-arrow and disable the **Include function descriptions** option.



To cancel the search, press the **Esc** key or click ×.

To view the data type of a function input or output argument:

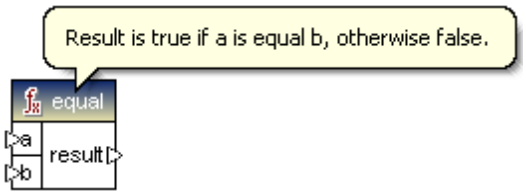
1. Make sure that the **Show tips** toolbar button is enabled.
2. Move your mouse over the argument part of a function.



To view the description of a function:

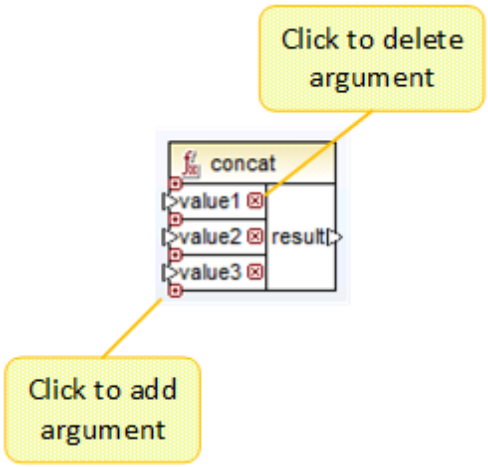
1. Make sure that the **Show tips** toolbar button is enabled.

2. Move your mouse of the function (this works both in the Libraries pane and on the mapping area)



To add or delete function arguments (for functions where that is applicable):

- Click **Add parameter** (⊡) or **Delete parameter** (⊠) next to the parameter you want to add or delete, respectively.



Dropping a connection on the ⊡ symbol automatically adds the parameter and connects it.

To find all occurences of a function within the currently active mapping:

- Right-click the function name in the Libraries window, and select **Find All Calls** from the context menu. The search results are displayed in the Messages window.

8.1.1 Priority Context node/item

When applying a function to different items in a schema or database, MapForce needs to know what the context node will be. All other items are then processed relative to this one. This is achieved by designating the item (or node) as the priority context.

Priority-context is used to prioritize execution when mapping unrelated items.

Mappings are always executed top-down; if you loop/search through two tables then each loop is processed consecutively. When mapping unrelated elements, without setting the priority context,

MapForce does not know which loop needs to be executed first, it therefore automatically selects the first table, or data source.

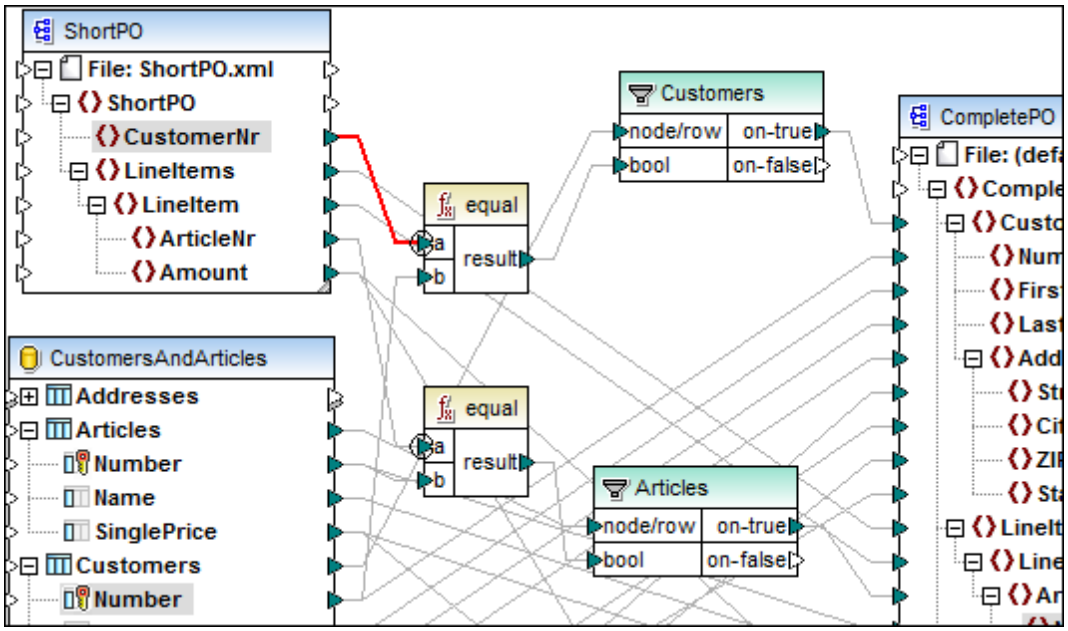
Solution:
Decide which table, or source data is to be looped/searched first, and then set the priority context on the connector to that table.

A simplified version of the complete **DB_CompletePO.mfd** file available in the **...\MapForceExamples** folder, is shown below.

Note that there are multiple source components in this example. **ShortPO** is a Schema with an associated XML instance file, while **CustomersAndArticles** is a database. The data from both are then mapped to the CompletePO schema / XML file. The priority context icon is enclosed in a circle as a visual indication.

Designating the **a** parameter of the equal function as the **priority context** would cause:

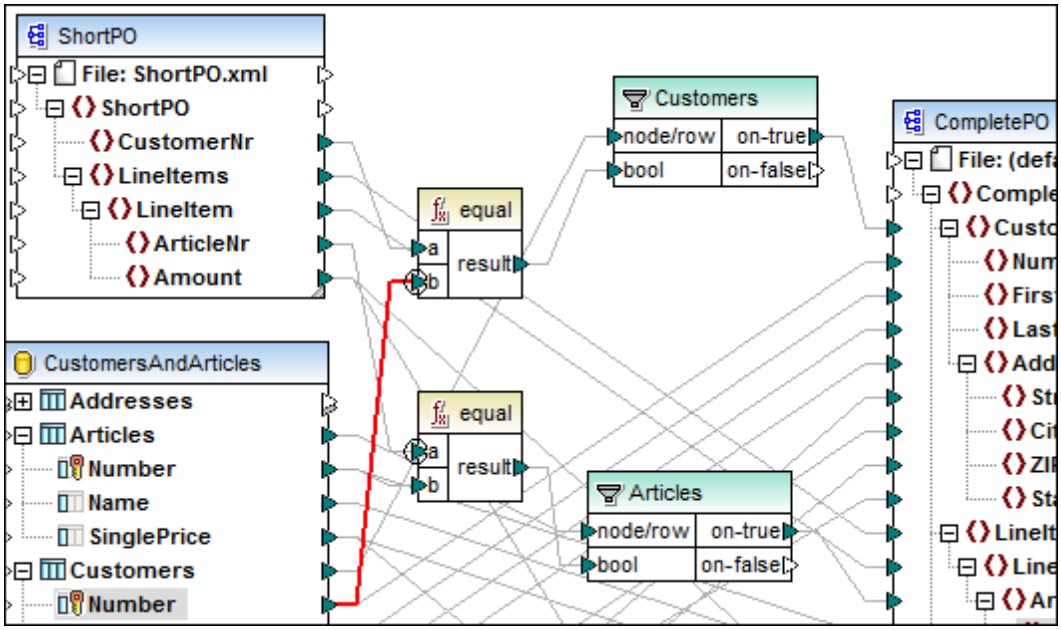
- The **CustomerNr** in ShortPO is compared with the item **Number** in the database.
- **CustomerNr** has been designated as the **priority context**, and is placed in the **a** parameter of the equal function.
- The **CustomersAndArticles** database is then searched (**once**) for the **same** number. The **b** parameter contains the Number item from the database.
- If the number is found, then the result is passed to the **bool** parameter of the **filter** component (Customers).
- The **node/row** parameter passes on the **Customer** data to "on-true" when the bool parameter is true, i.e. when the same number has been found.
- The rest of the customer data is then passed on as: Number, FirstName, LastName items, are all connected to the corresponding items in the target schema.



This means that the database is only searched once per CustomerNr supplied by ShortPO.

Designating the **b** parameter of the equal function as the **priority context** would cause:

- MapForce to search and load the first Number into the **b** parameter from the database
- Check against the **CustomerNr** in the **a** parameter of ShortPO
- If not equal, search through all CustomerNr of ShortPO
- Search the database and load the next Number into **b**, check against a, and
- Iterate through every Number in the database while trying to find that number in ShortPO.



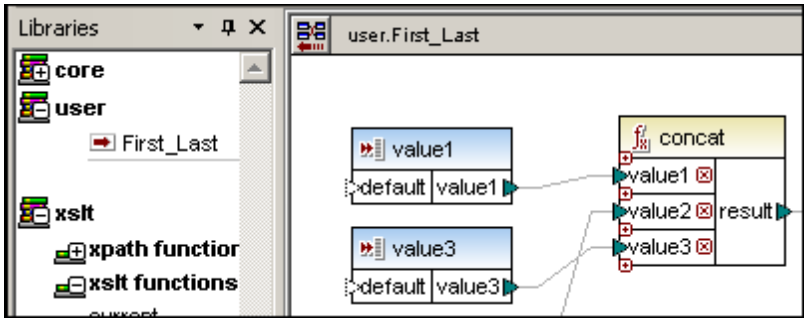
This means that a database query is generated for each Number and the result is then compared to every CustomerNr of ShortPO.

Priority context and user-defined functions:
If a user-defined function has been defined of type "inline", the default setting, then a priority context cannot be defined on one of the parameters of the user-defined function. The user-defined function can, of course, contain other regular (non-inlined) user-defined functions which have priority contexts set on their parameters.

8.2 User-Defined Functions

MapForce allows you to create user-defined functions visually, in the same way as in the main mapping window.

These functions are then available as function entries in the Libraries window (for example, "First_Last" in the image below), and are used in the same way as the currently existing functions. This allows you to organize your mapping into separate building blocks which are reusable across different mappings.



User-defined functions are stored in the ***.mfd** file, along with the main mapping.

A user-defined function uses **input** and **output components** to pass information from the main mapping (or another user-defined function) to the user-defined function and back.

User-defined functions can contain "local" source components (i.e that are within the user-defined function itself) such as XML schemas or databases, which are useful when implementing lookup functions.

User-defined functions can contain any number of input and outputs where any of these can be in the form of: simple values, XML nodes, databases, EDI files, or FlexText structure files.

User-defined functions are useful when:

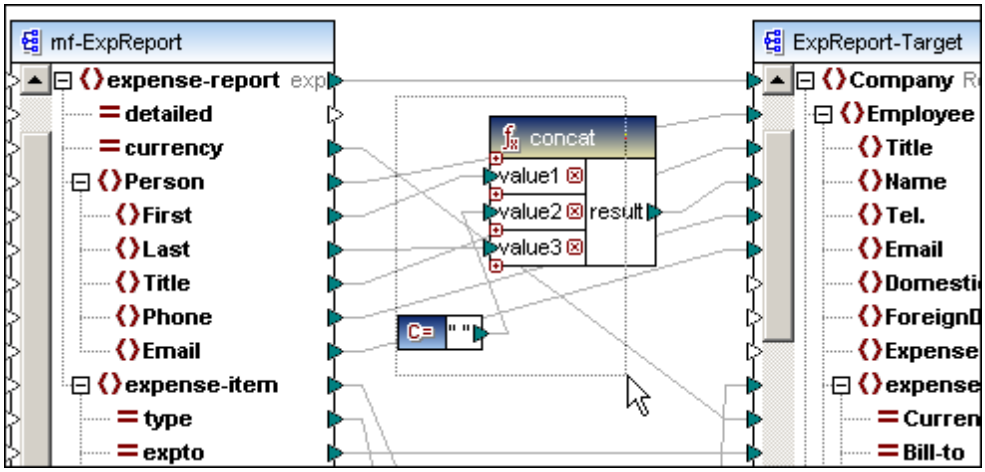
- combining multiple processing functions into a single component, e.g. for formatting a specific field or looking up a value
- reusing these components any number of times
- importing user-defined functions into other mappings (by loading the mapping file as a library)
- using inline functions to break down a complex mapping into smaller parts that can be edited individually
- mapping **recursive schemas** by creating recursive user-defined functions

User-defined functions can be either built from scratch, or from functions already available in the mapping tab.

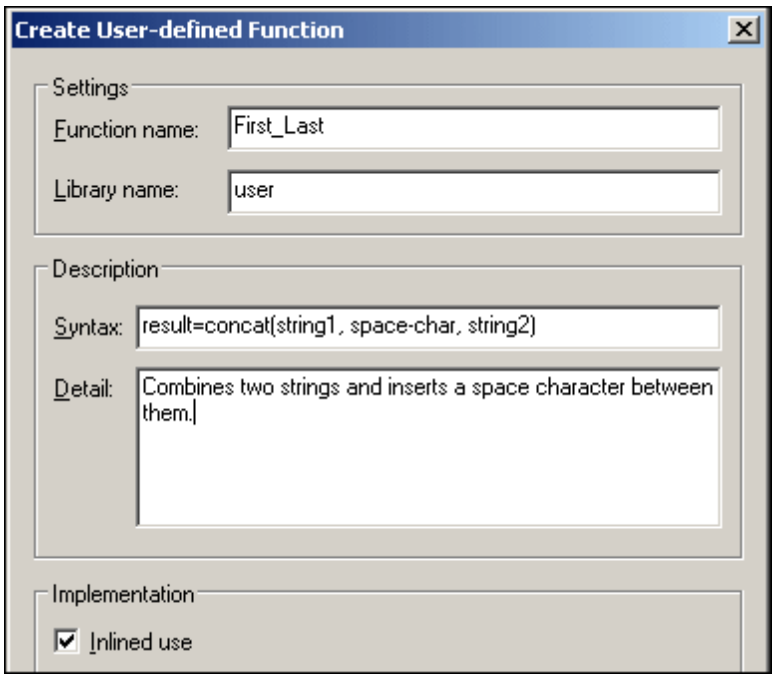
This example uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples\Tutorial\ folder.

Creating user-defined function from existing components

1. Drag to select both the "concat" and the constant components (you can also hold down the CTRL key and click the functions individually).

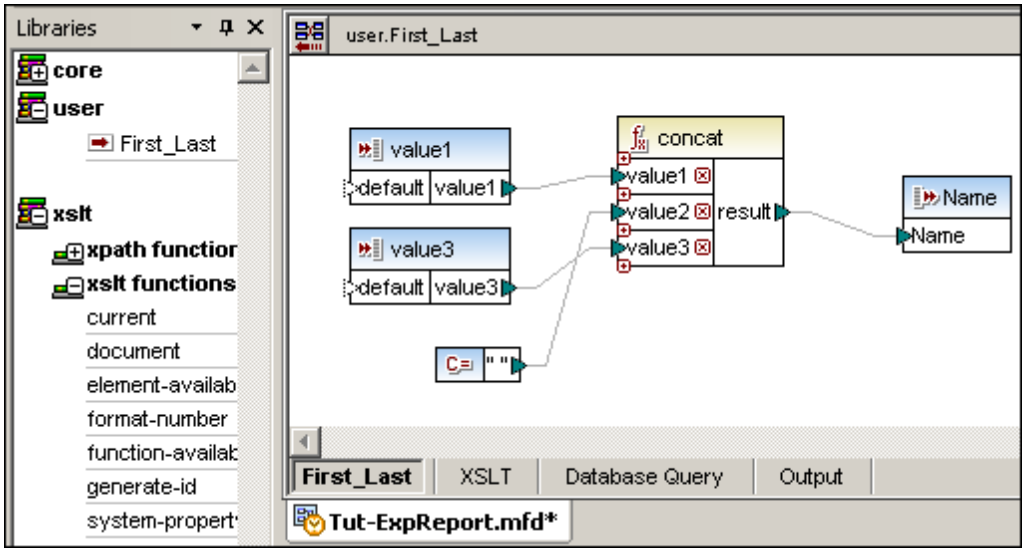


2. Select the menu option **Function | Create User-Defined Function from Selection**.
3. Enter the name of the new user-defined function (First_Last).
   Note: valid characters are: alphanumeric, a-z, A-Z, 0-9 as well as underscore "_", hyphen/dash "-" and colon ":".
4. Use the Syntax and Detail fields to add extra information on the new function, and click OK to confirm. The text you enter will appear as a tooltip when the cursor is placed over the function.
   The library name "user" is supplied as a default, you can of course define your own library name in this field.

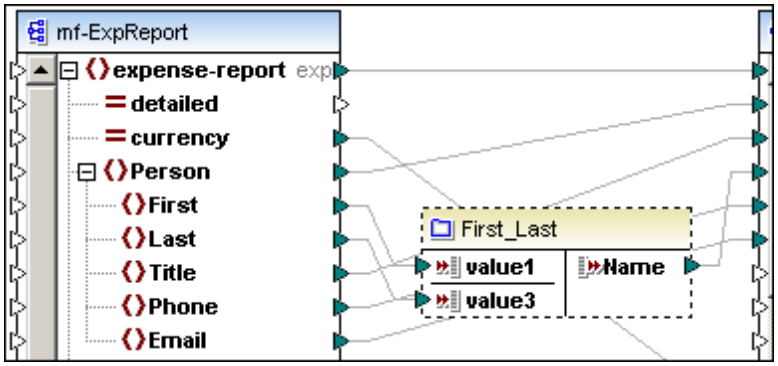


The individual elements that make up the function group appear in a tab with the function

name. The new library "user" appears in the Libraries pane with the function name "First_Last" below it.



Click the Home button  to return to the main mapping window. The components have now been combined into a single function component called First_Last. The input and output parameters have been automatically connected.



Note that inline user-defined functions are displayed with a dashed outline. See Inline user-defined functions for more information.

Dragging the function name from the Libraries pane and dropping it in the mapping window, allows you to use it anywhere in the current mapping. To use it in a different mapping, please see Reusing user-defined functions

Opening user-defined functions

To open a user-defined function, do one of the following:

- Double-click the title bar of a user-defined function component
- Double-click the specific user-defined function in the Libraries window.

This displays the individual components inside the function in a tab of that name. Click the Home

button  to return to the main mapping. Double-clicking a user-defined function of a different *.mfd file (in the main mapping window) opens that .mfd file in a new tab.

Navigating user-defined functions

When navigating the various tabs (or user-defined function tabs) in MapForce, a history is automatically generated which allows you to travel forward or backward through the various tabs, by clicking the back/forward icons. The history is session-wide, allowing you to traverse mutliple MFD files.

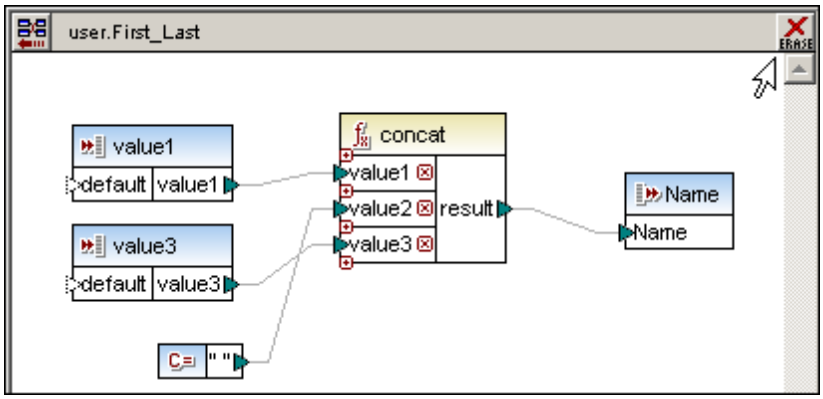
 The Home button returns you to the main mapping tab from within the user-defined function.

 The Back button takes you back through your history

 The Forward button moves you forward through your history

Deleting user-defined functions from a library

1. Double-click the specific user-defined function in the Libraries window.
2. Click the **Erase** button in the top right of the title bar.



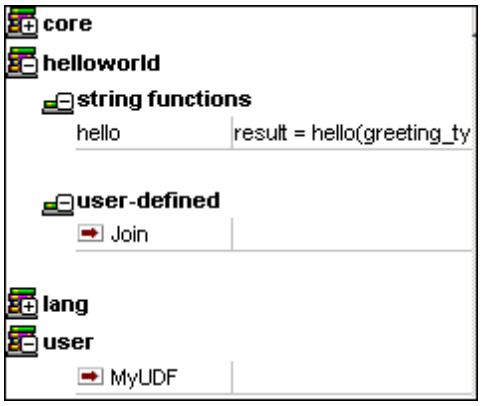
Reusing (importing) user-defined functions

User-defined functions defined in one mapping can be imported into any other mapping as follows:

1. Click the **Add/Remove Libraries** button at the base of the Libraries window.
2. Click **Add** and select the *.mfd file that contains the user-defined function(s) you want to import. Alternatively, you can import files of other types, if you have defined custom libraries previously (see Adding custom Java, C# and C++ function libraries). The user-defined function now appears in the Libraries window. The library name is "user" if you created the user-defined function with the default library name. Otherwise, look for the library name that you specified when creating the user-defined function.
2. Drag the imported function from the Libraries window into the mapping.

When creating user-defined functions, you can specify the same library name in multiple *.mfd files or custom libraries (see Adding custom libraries). In this case, functions from all available sources appear under the same library name in the Libraries window. However, only the functions in the currently active document can be edited by double-clicking.

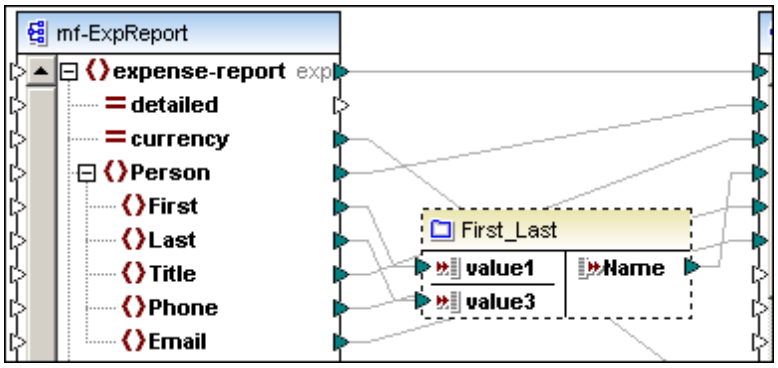
Consider the following example:



- The function "hello" in the "helloworld" library was imported from a custom library
- The function "Join" in the "helloworld" library is a user-defined function defined in the current *.mfd file
- The function "MyUDF" in the "user" library is also a user-defined function defined in the current *.mfd file

Note that possible changes in imported functions are applied to importing mappings when saving the library ***.mfd** file.

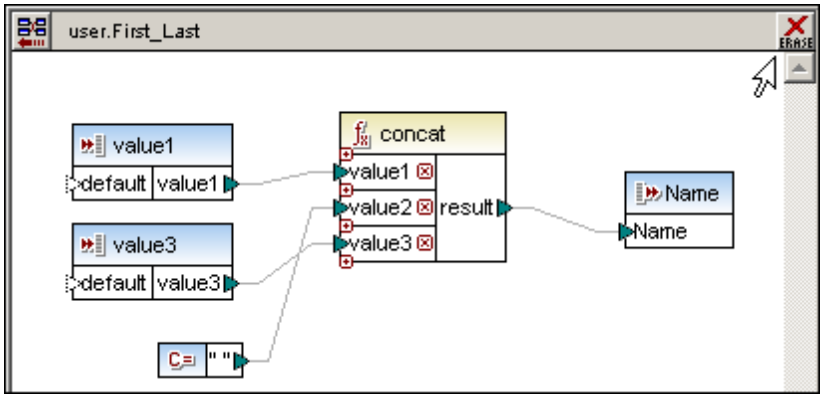
Parameter order in user-defined functions



The parameter order within user-defined functions can be directly influenced:

- Input and output parameters are sorted by their position from top to bottom (from the top left corner of the parameter component).
- If two parameters have the same vertical position, the leftmost takes precedence.

- In the unusual case that two parameters have exactly the same position, the internal component ID is automatically used.



Notes:

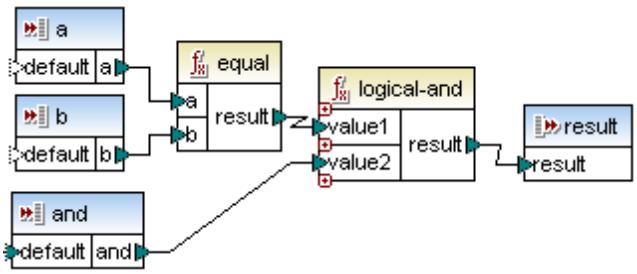
- The Component positioning and resizing actions are undoable.
- Newly added input or output components are created below the last input or output component.
- Complex and simple parameters can be mixed. The parameter order is derived from the component positions.

8.2.1 Function parameters

Function **parameters** are represented inside a user-defined function by **input** and **output components**.

Input components/parameters: **a**, **b**, **and**

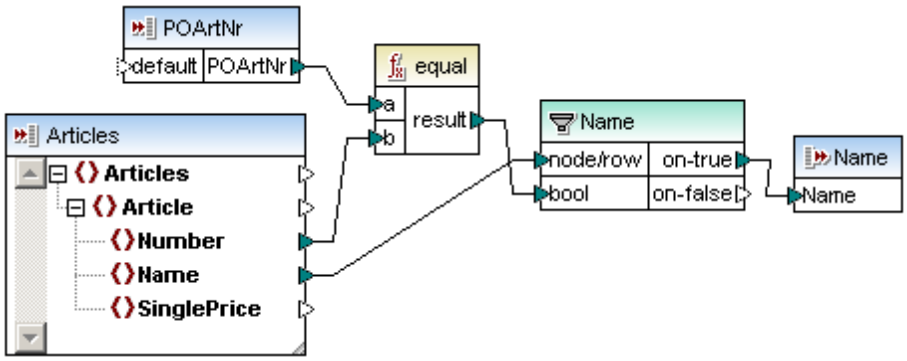
Output component/parameter: **result**



Input parameters are used to pass data from the main mapping into the user-defined function, while output parameters are used to return data back to the main mapping. Note that user-defined functions can also be called from other user-defined functions.

Simple and complex parameters
The input and output parameters of user-defined functions can be of various types:

- Simple values, e.g. string or integer
- Complex node trees, e.g. an XML element with attributes and child nodes



Input parameter **POArtNr** is a simple value of datatype "string"

Input parameter **Articles** is a **complex** XML document node tree

Output parameter **Name** is a simple value of type string

Note:

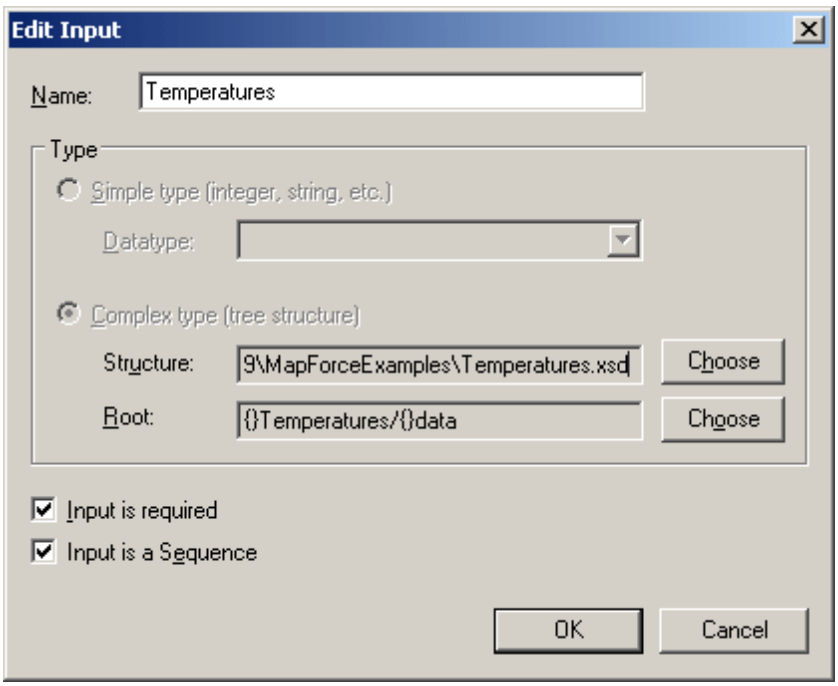
The user-defined functions shown above are all available in the **PersonListByBranchOffice.mfd** file available in the ...\MapForceExamples folder.

Sequences

Sequences are data consisting of a range, or sequence, of values. Simple and complex user-defined **parameters** (input/output) can be defined as sequences in the component properties dialog box.

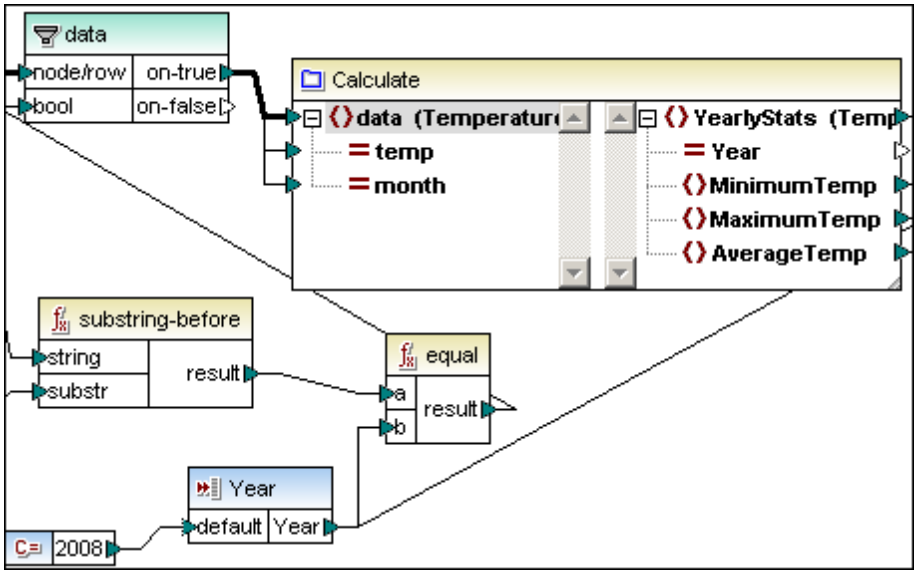
Aggregate functions, e.g. min, max, avg, etc., can use this type of input to supply a single specific value from the input sequence.

When the "**Input is a Sequence**" check box is active, the component handles the input as a sequence. When inactive, input is handled as a single value.

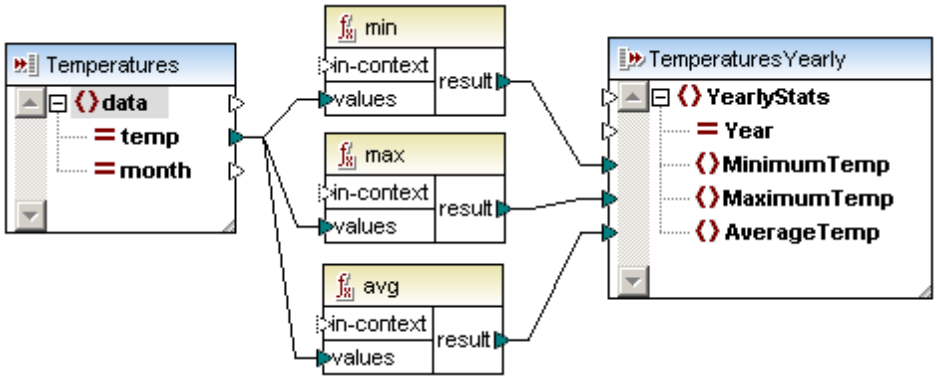


This type of input data, sequence or non-sequence, determines **how often** the function is called.

- When connected to a **sequence** parameter the user-defined function is called only **once** and the complete sequence is passed into the user-defined function.



The screenshot shows the user-defined function "Calculate" of the "**InputIsSequence.mfd**" mapping in the ...\MapForceExamples folder. The **Temperatures** input component (shown below) is defined as a sequence.



- When connected to a **non-sequence** parameter, the user-defined function is called **once** for **each** single item in the sequence.

Please note:

The sequence setting of input/output parameters is ignored when the user-defined function is of type inline.

> Connecting an empty sequence to a non-sequence parameter has the result that the function is not called at all.

This can happen if the source structure has **optional** items, or when a filter condition returns no matching items. To avoid this, either use the substitute-missing function before the function input to ensure that the sequence is never empty, or set the parameter to sequence, and add handling

for the empty sequence inside the function.

When a function passes a sequence of multiple values to its output component, and the output component is not set to sequence, only the first result is used when the function is called.

8.2.2 Inline and regular user-defined functions

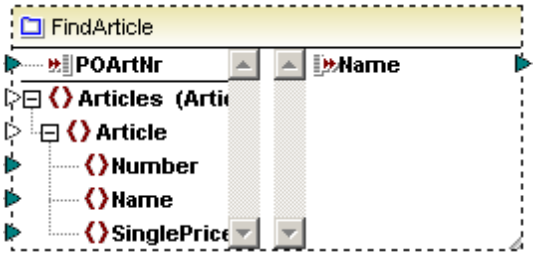
Inline functions differ fundamentally from regular functions, in the way that they are implemented when code is generated.

- The code for **inline** type functions is inserted at **all locations** where the user-defined functions are called/used
- The code of a **regular** function is implemented as a function call.

  Inline functions thus behave as if they had been replaced by their implementation. That makes them ideal for breaking down a complex mapping into smaller encapsulated parts.

Please note:
using **inline** functions can significantly increase the amount of generated program code! The user-defined function code is actually inserted at all locations where the function is called/used, and thus increases the code size substantially - as opposed to using a regular function.

INLINE user-defined functions are shown with a **dashed** outline:



Inline user-defined functions **support**:

- **Multiple output components** within a function

do not support:

- The setting of a priority context on a parameter
- Recursive calls to an inline user-defined function

REGULAR user-defined functions i.e. non-inline functions are shown with a **solid** outline:

Regular (non-inline) user-defined functions **support**:

- Only a **single output component** within a function

- **Recursive** calls (where the exit condition must be supplied, e.g. use an If-Else condition where one branch, or value, exits the recursion)
- Setting a priority context on a parameter

Please note:

Although regular functions **do not** support multiple output components, they **can be created** in this type of function. However, an error message appears when you try to generate code, or preview the result of the mapping.

If you are not using recursion in your function, you can change the type of the function to "inline".

do not support:

- Direct connection of filters to simple non-sequence **input** components
- Sequence or aggregate functions on simple input components (like exists, substitute-missing, sum, group-by, ...)

Code generation

The implementation of a regular user-defined function is generated only once as a callable XSLT template or function. Each user-defined function component generates code for a **function call**, where inputs are passed as parameters, and the output is the function (component) return value.

At runtime, all the input parameter values are evaluated first, then the function is called for each occurrence of the input data. See Function parameters for details about this process.

To change the user-defined function "type":

1. Double click the user-defined function to see its constituent components.
2. Select the menu option **Function | Function settings** and click the "Inlined use" checkbox.

User-defined functions and Copy-all connections

When creating Copy-all connections between a schema and a complex user-defined function parameter, the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components - defining" for an example.

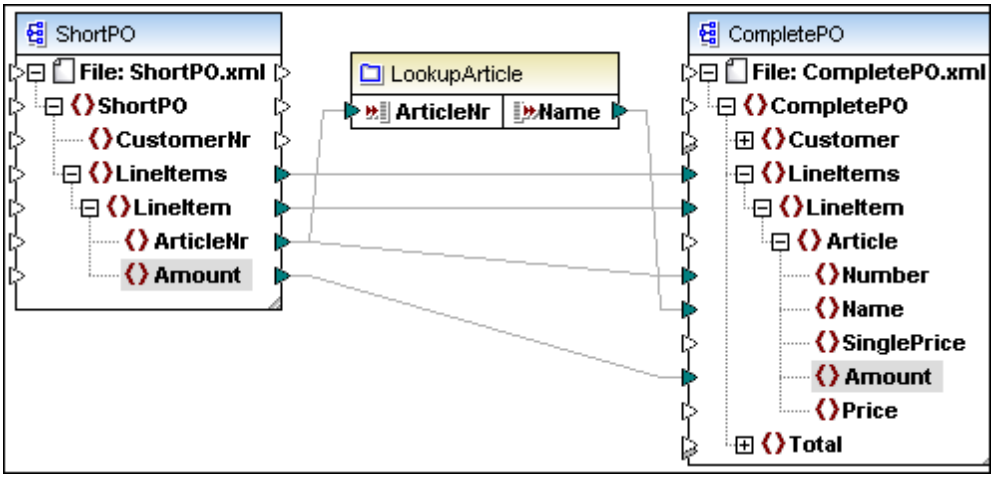
8.2.3 Creating a simple look-up function

This example is provided as the **lookup-standard.mfd** file available in the **...\MapForceExamples** folder.

Aim:

To create a generic look-up function that:

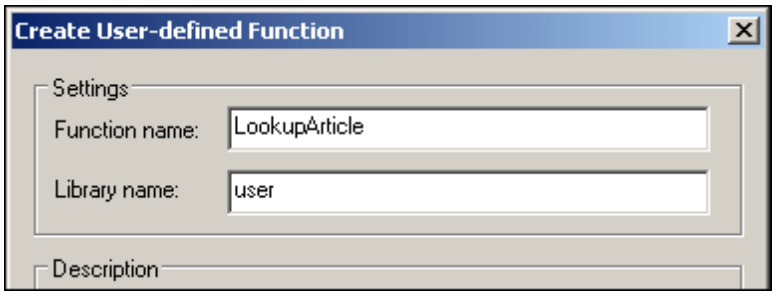
- supplies Articles/Number data from the Articles XML file, to be compared to Article numbers of a different XML file, ShortPO in this case.



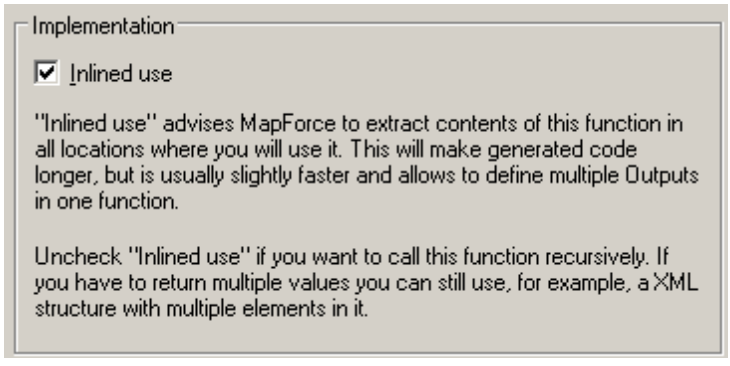
- Insert the ShortPO.xsd and assign ShortPO.xml as the source XML file.
- Insert the CompletePO.xsd schema file, and select CompletePO as the root element.
- Insert a new user-defined function using the method described below.

To create a user-defined function:

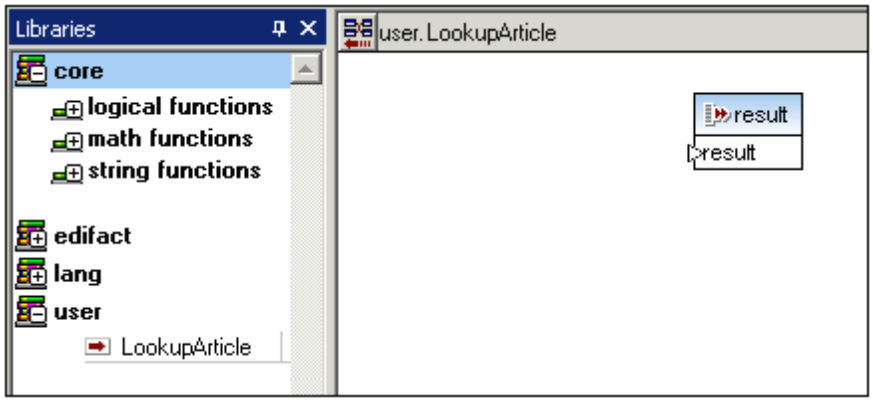
1. Select the menu option **Function | Create User-defined function**.
2. Enter the name of the function e.g. LookupArticle.



3. Uncheck the "Inlined use" checkbox and click OK to confirm



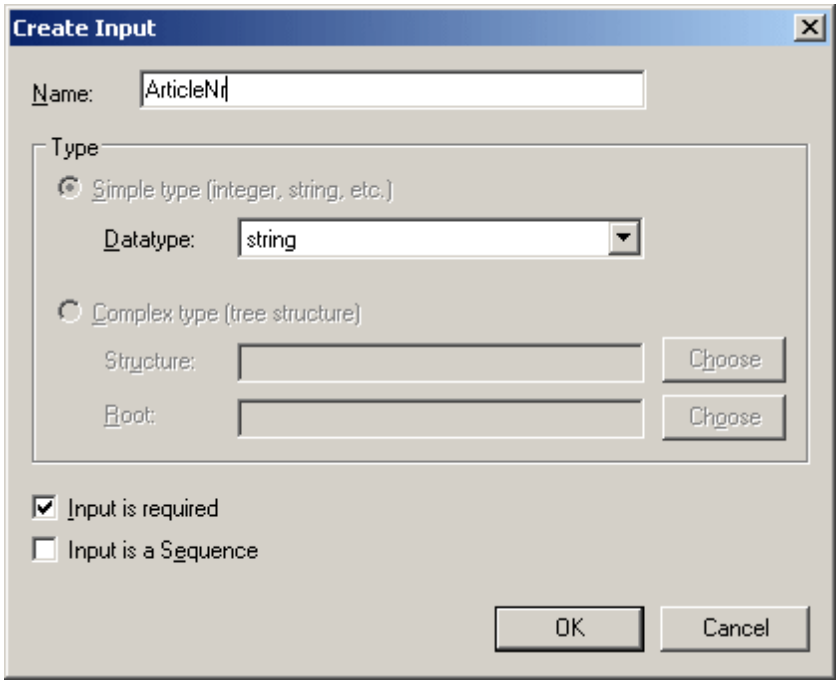
A tab only containing only one item, an output function currently called "result", is displayed.



This is the working area used to define the user-defined function.

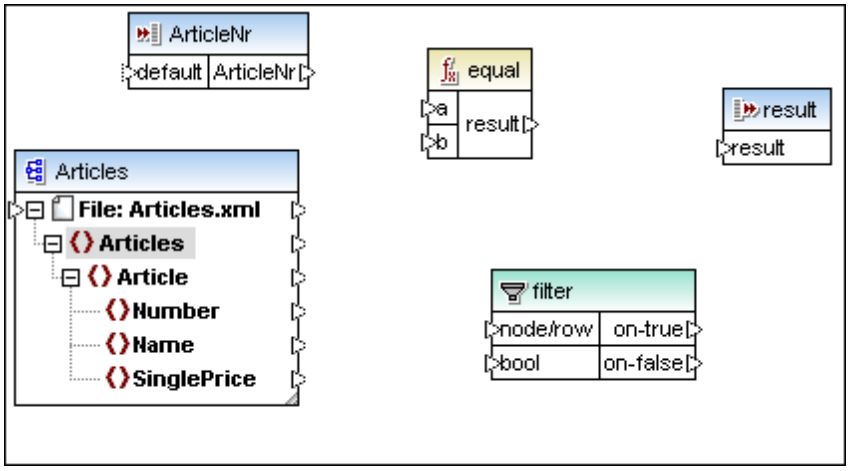
A new library has been created in the Libraries pane with the name "user" and the function name "LookupArticle".

3. Click the **Insert Schema/XML file** icon to insert the **Articles** schema and select the XML file of the same name to act as the data source.
4. Click the **Insert input component** icon to insert an input component.
5. Enter the name of the input parameter, ArticleNr in this case, and click OK.



This component acts as a data input to the user-defined function and supplies the input icon of the user-defined function.

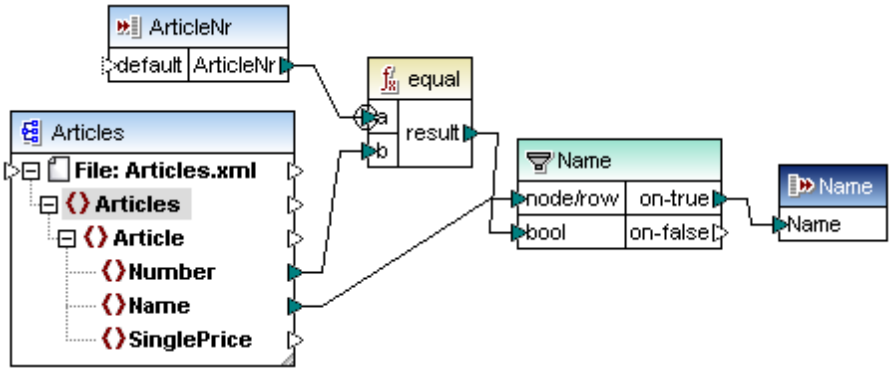
6. Insert an "**equal**" component by dragging it from the core library/logical functions group.
7. Insert a **filter** component by clicking the Insert Filter icon in the toolbar.



Use the diagram below as an aid to creating the mappings in the user-defined function, please take note of the following:

8 Right click the **a** parameter and select **Priority context** from the pop up menu.

9. **Double click** the output function and enter the name of the output parameter, in this case "**Name**".



This ends the definition of the user-defined function.

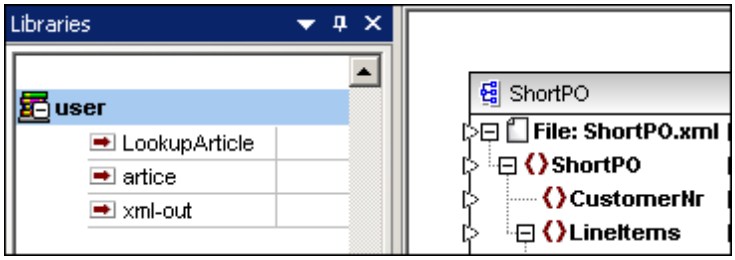
Please note:

Double clicking the **input** and **output** functions opens a dialog box in which you can change the name and datatype of the input parameter, as well as define if the function is to have an input icon (Input is required) and additionally if it should be defined as a sequence.

This user-defined function:

- has one **input** function, ArticleNr, which will receive data from the ShortPO XML file.
- **compares** the ShortPO ArticleNr, with the Article/Number from the **Articles** XML instance file, inserted into the user-defined function for this purpose.
- uses a **filter** component to forward the Article/Name records to the output component, if the comparison returns true.
- has one output function, Name, which will forward the Article Name records to the CompletePO XML file.

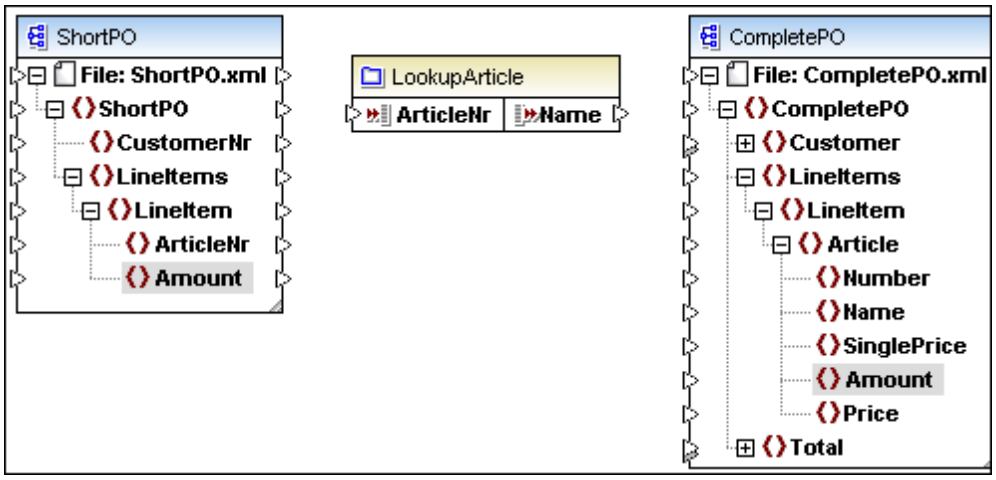
10. Click the Home icon  to return to the main mapping.
The LookupArticle user-defined function, is now available under the **user** library.



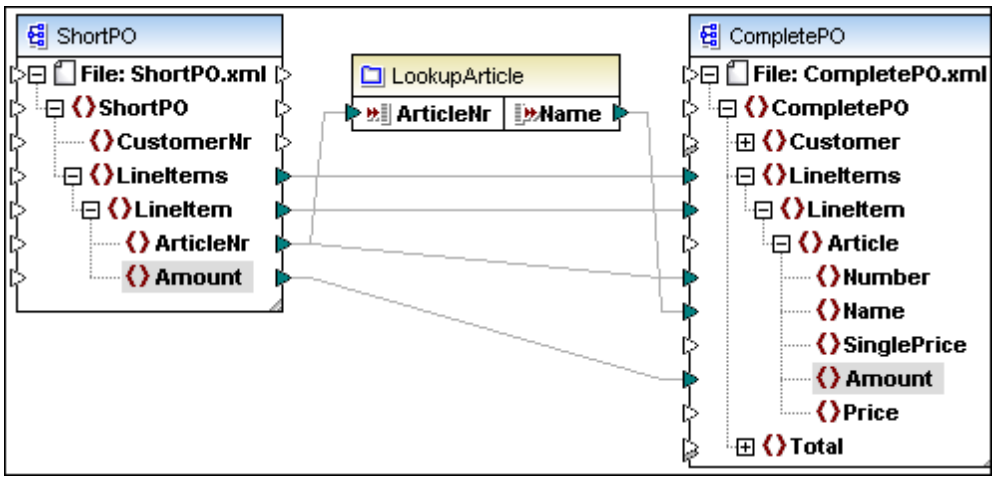
11. Drag the LookupArticle function into the Mapping window.

    The user-defined function is displayed:

- with its name "LookupArticle" in the title/function bar,
- with named input and output icons.



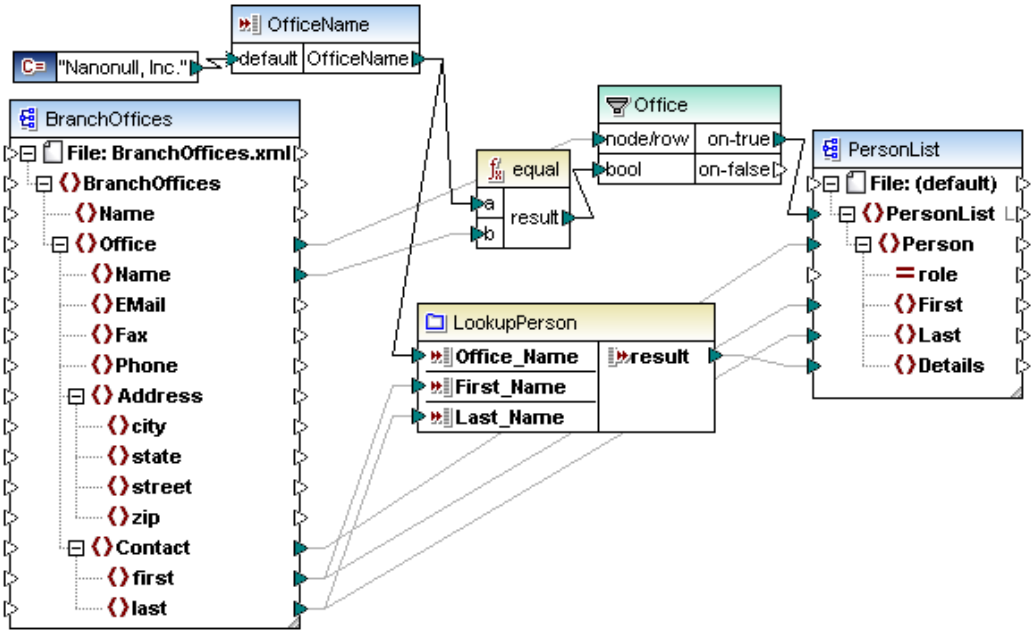
10. Create the connections displayed in the graphic below and click the Output tab to see the result of the mapping.



8.2.4 User-defined function - example

The **PersonListByBranchOffice.mfd** file available in the **<Documents>\Altova\MapForce2016\MapForceExamples** folder illustrates the following features:

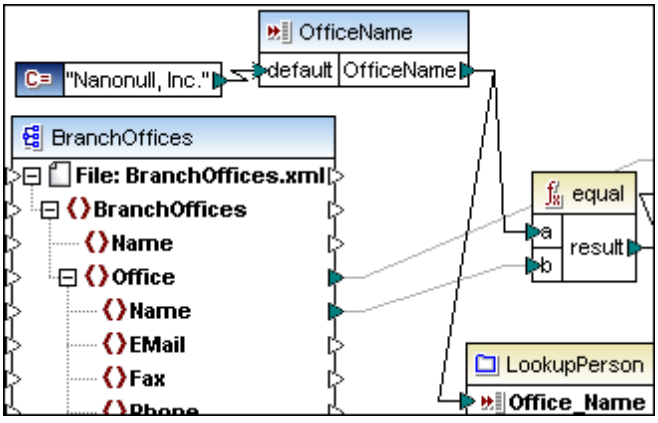
- Nested User-defined functions e.g. **LookupPerson**
- Look-up functions that generate a string output e.g. **LookupPerson**
- **Optional** input-parameters which can also supply a **default** value e.g. the **EqualAnd** component (contained in the LookupPerson component)
- **Configurable** input parameters, which can also double as a command line parameter(s) when executing the generated mapping code!



Configurable input parameters

The input component (OfficeName) receives data supplied when a mapping is executed. This is possible in two ways:

- as a **command line** parameter when executing the generated code, e.g. Mapping.exe / OfficeName "Nanonull Partners, Inc."
- as a **preview** value when using the Built-in execution engine to preview the data in the Output window.

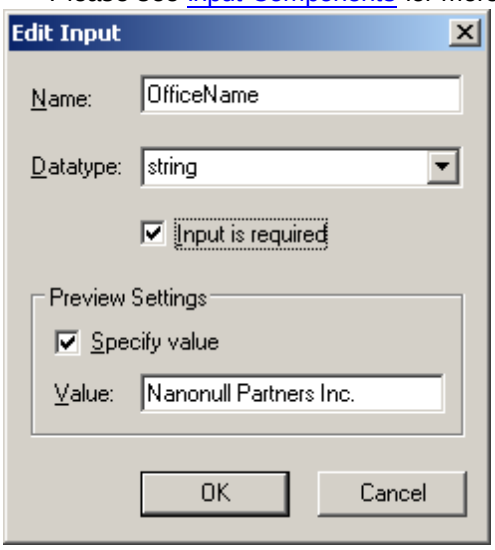


To define the Input value:

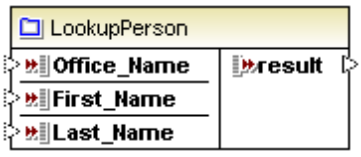
1. Double click the input component and enter a different value in the "Value" text box of the Preview Mode group e.g. "Nanonull Partners, Inc.", and click OK to confirm.
2. Click the Output tab to see the effect.
   A different set of persons are now displayed.

   Please note that the data entered in this dialog box is only used in "**preview**" mode i.e. when clicking the Output tab. If a value is not entered, or the check box is deactivated, then the data mapped to the input icon "default" is used.

   Please see Input Components for more information.



LookupPerson component

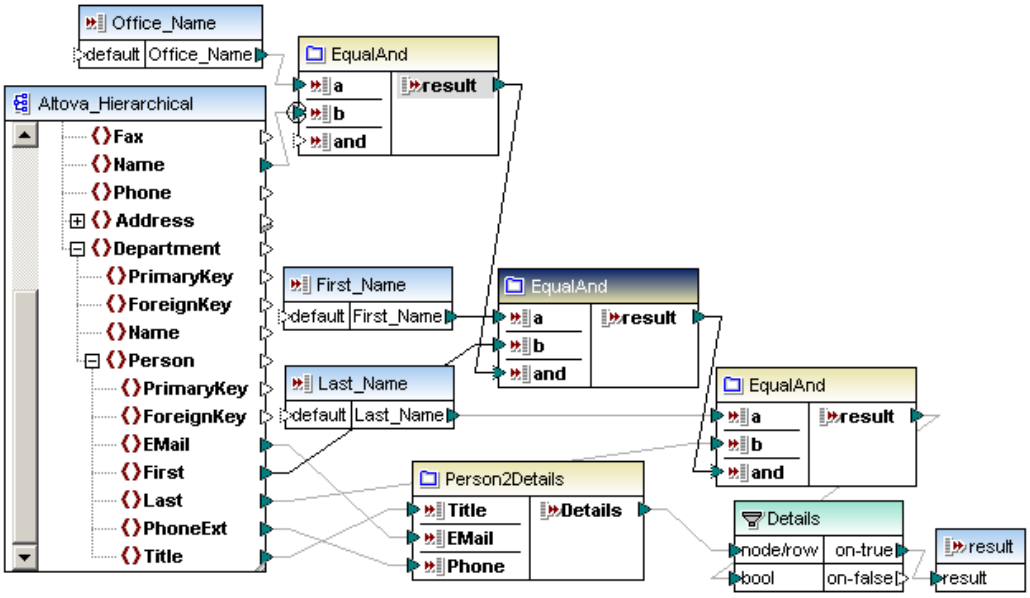


Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- **Compares** the Office, First, and Last names of BranchOffices.xml, with the same fields of the Altova_Hierarchical.xml file, using the **input** components and the **EqualAnd** user-defined components.
- **Combines** the Email, PhoneExt and Title items using the **Person2Details** user-defined function
- **Passes on** the combined person data to the **output** component if the previous EqualAnd comparisons are all true (i.e. supplied "true" to the filter component).

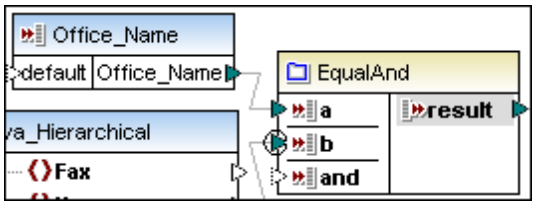
A user-defined function always outputs a value, which may even be an empty string! This would be the case if the filter component bool value is false. Only an empty string would be output instead

of data supplied by the Person2Details component.



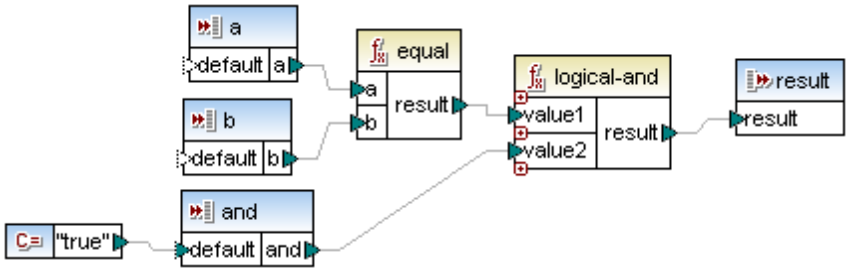
- The three **input** components, Office_Name, First_Name, Last_Name, receive their data from the BranchOffices.xml file.
- The **EqualAnd** component compares two values and provides an **optional** comparison value, as well as a default value.
- Person2Details combines three person fields and passes on the result to the filter component.

EqualAnd component



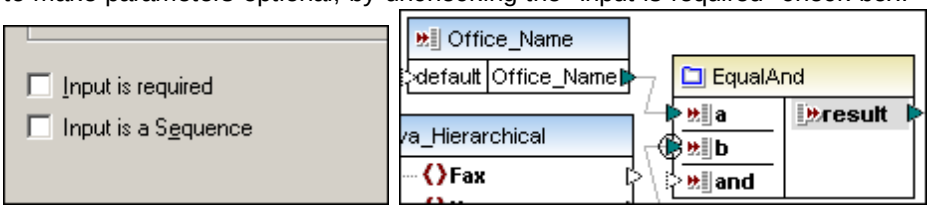
Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- Compare two input parameters **a** and **b**, and pass the result on to the logical-and component. Note that the **b** parameter has been defined as the **priority context** (right click the icon to do so). This ensures that the person data of the specific office, supplied by the input parameter **a**, is processed first.
- **Logical-and** the result of the first comparison, with an **optional** input parameter, "and".
- Pass on the boolean value of this comparison to the output parameter.



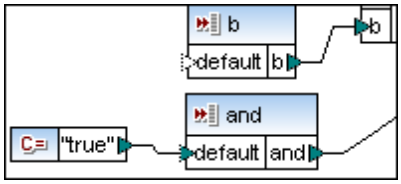
Optional parameters

Double clicking the "and" parameter, of the EqualAnd user-defined function shown above, allows you to make parameters optional, by unchecking the "Input is required" check box.



If "Input is required" is **unchecked**, then:

- A mapping connector is not required for the input icon of this user-defined function, e.g. the **and** parameter of the first EqualAnd function, does not have an input connector. The input icon has a dashed outline to show this visually.
- A **default** value can be supplied by connecting a component, within the user-defined function e.g. using a constant component containing the value "true".

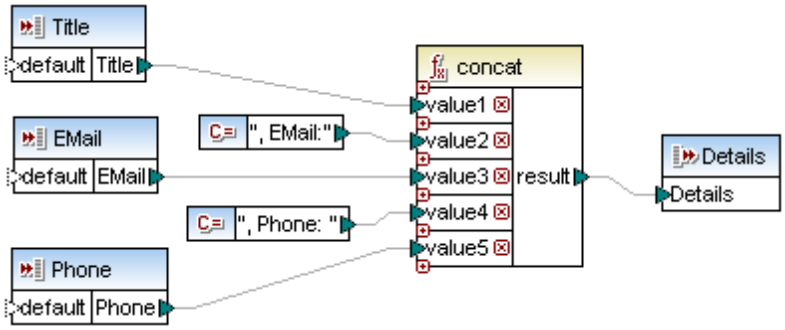


- A mapping from another item, mapped to the optional Input, takes precedence over the default value. E.g. the "and" parameter of second EqualAnd function, receives input data from the "result" parameter of the first EqualAnd user-defined function.

Person2Details component

Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- Concatenate three inputs and pass on the result string to the output parameter.
- Double clicking an output parameter allows you to change the parameter name (Details), and select the datatype (String).



8.2.5 Complex user-defined function - XML node as input

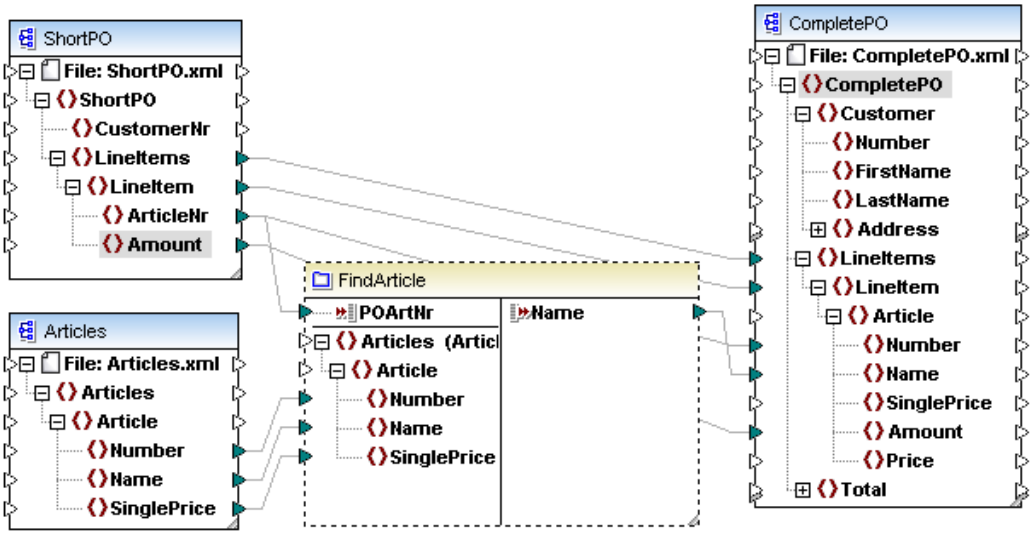
This example is provided as the **lookup-udf-in.mfd** file available in the ...\MapForceExamples folder. This section illustrates how to define an inline user-defined function that contains a complex input component.

Note that the user-defined function "FindArticle" consists of two halves.

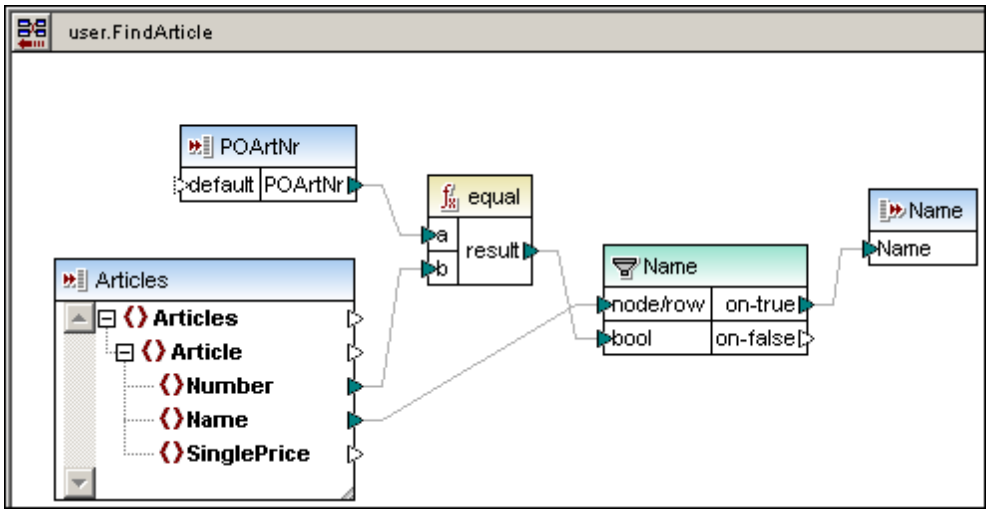
The left half contains the input parameters:

- a simple input parameter **POArtNr**
- a complex input component **Articles,** with mappings directly to its XML child nodes

The right half contains a simple output parameter called "**Name**".



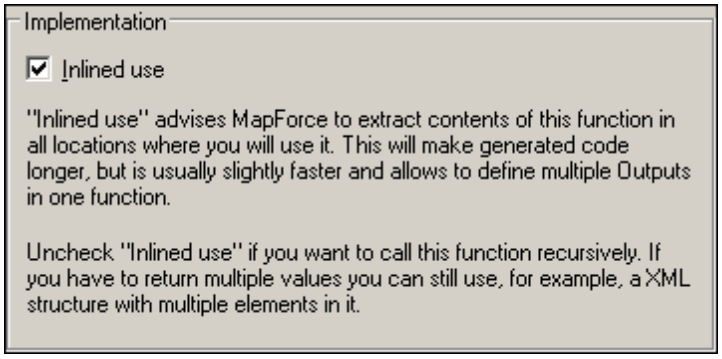
The screenshot below shows the constituent components of the user-defined function, the two input components to the left and the output component to the right.



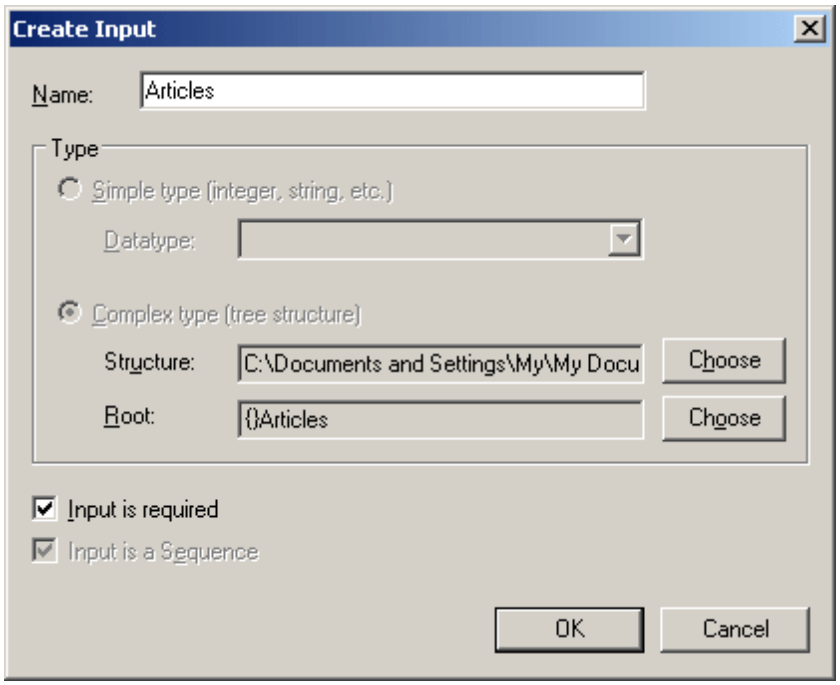
8.2.5.1 Defining Complex Input Components

Follow these steps to create a function that takes an XML structure as input parameter:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** and click OK to confirm. Note that the **Inlined use** check box is automatically selected.



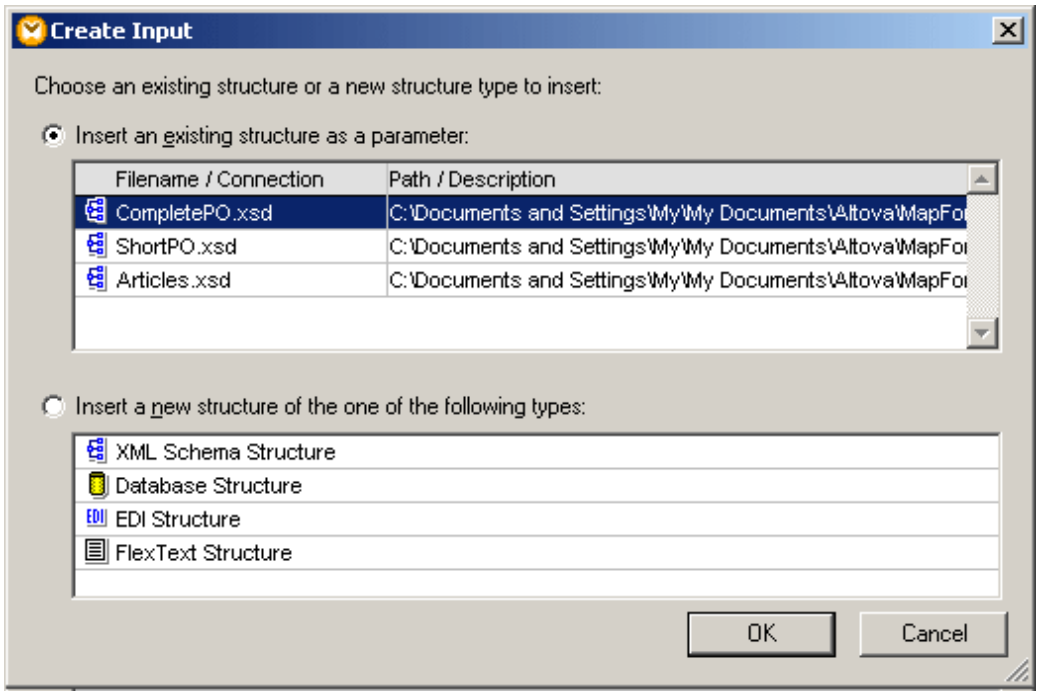
2. Click the **Insert input component** icon  in the icon bar.
3. Enter the name of the input component into the Name field.



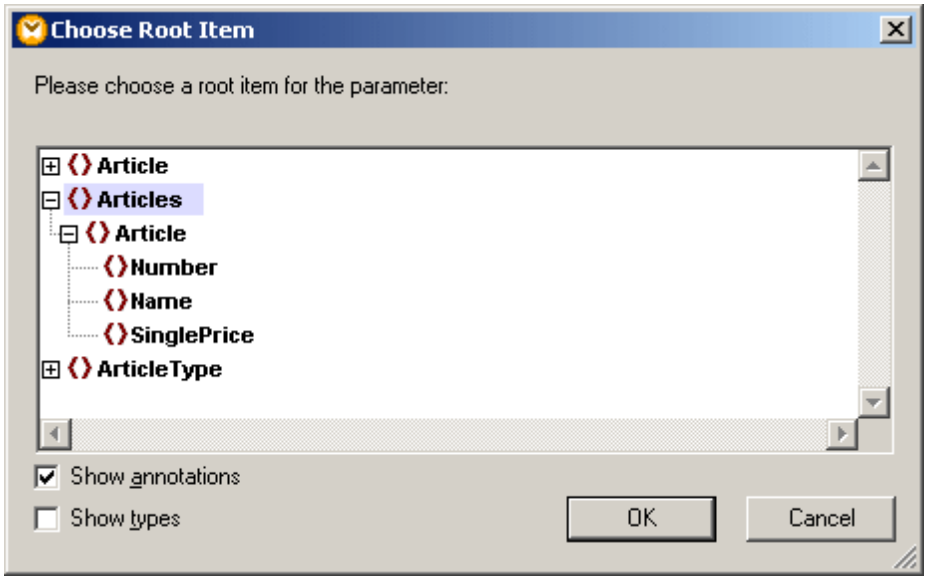
4. Click the **Complex type (tree structure)** radio button, then click the "**Choose**" button next to the Structure field. This opens another dialog box.

   The top list box displays the **existing** components in the mapping (three schemas if you opened the example mapping). Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML schema, database, EDI file, or FlexText structure file.

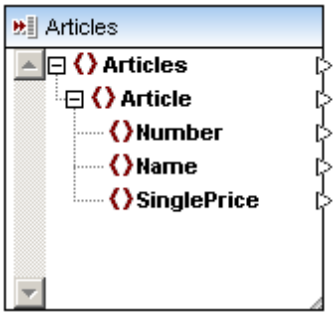
   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, EDI file, or FlexText structure file.



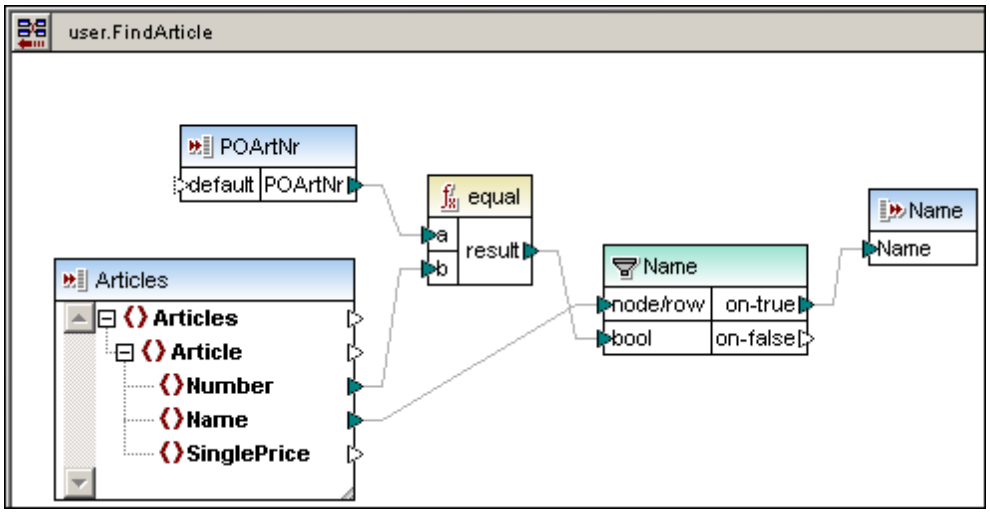
5. Click "Insert a new structure... " radio button, select the XML Schema Structure entry, and click OK to continue.
6. Select **Articles.xsd** from the "Open" dialog box.
7. Click the element that you would like to become the root element in the component, e.g. Articles, and click OK, then OK again to close both dialog boxes.



The Articles component is inserted into the user-defined function. Please note the input icon to the left of the component name. This shows that the component is used as a complex input component.

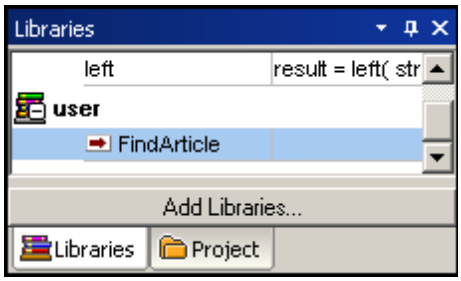


8. Insert the rest of the components as shown in the screenshot below, namely: a second "simple" input component (called POArtNr), filter, equal and output component (called Name), and connect them as shown.

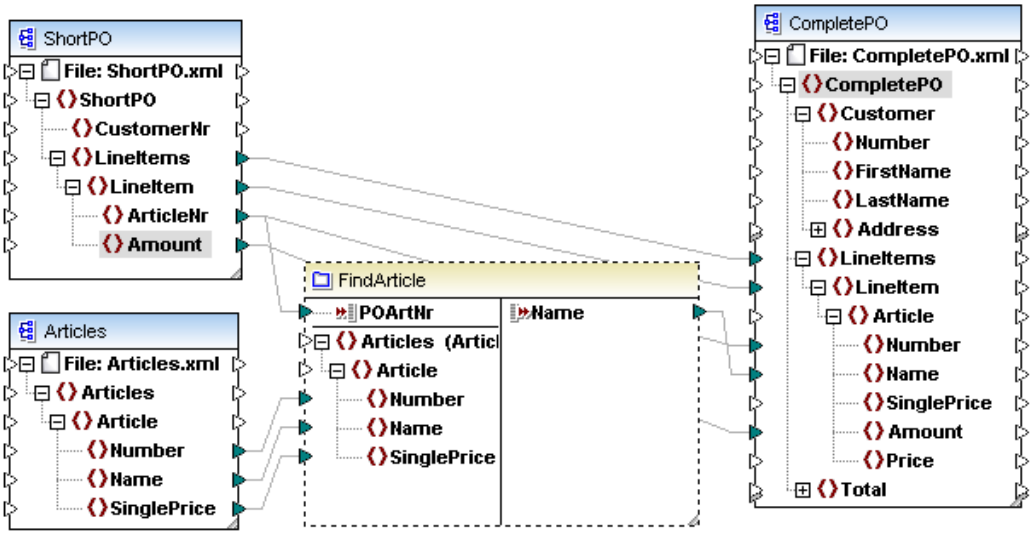


- The **Articles** input component receives its data from **outside** of the user-defined function. Input icons that allow mapping to this component, are available there.
- An XML **instance** file to provide data from within the user-defined function, cannot be assigned to a complex input component.
- The other input component POArtNr, supplies the ShortPO article number data to which the **Article | Number** is compared.
- The filter component filters the records where both numbers are identical, and passes them on to the output component.

10. Click the Home icon  to return to the mapping.
11. Drag the newly created user-defined component from the Libraries pane into the mapping.



12. Create the connections as shown in the screenshot below.



The left half contains the input parameters to which items from two schema/xml files are mapped:

- **ShortPO** supplies the data for the input component **POArtNr.**
- **Articles** supplies the data for the complex input component. The Articles.xml instance file was assigned to the Articles schema file when the component was inserted.
- The complex input component **Articles** with its XML child nodes, to which data has been mapped from the Articles component.

The right half contains a simple output parameter called "**Name"**, which passes the filtered line items which have the same Article number to the "Name" item of **CompletePO**.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    <LineItems>
4      <LineItem>
5        <Article>
6          <Number>3</Number>
7          <Name>Pants</Name>
8          <Amount>5</Amount>
9        </Article>
10     </LineItem>
11     <LineItem>
12       <Article>
13         <Number>1</Number>
14         <Name>T-Shirt</Name>
15         <Amount>17</Amount>
16       </Article>
17     </LineItem>
18   </LineItems>
19 </CompletePO>
```

Note: When creating **Copy-all** connections between a schema and a user-defined function parameter, the two components must be based on the same schema. It is not necessary that they both have the same root elements however.

8.2.6 Complex user-defined function - XML node as output

This example is provided as the **lookup-udf-out.mfd** file available in the **...\MapForceExamples** folder. What this section will show is how to define an inline user-defined function that allows a complex output component.

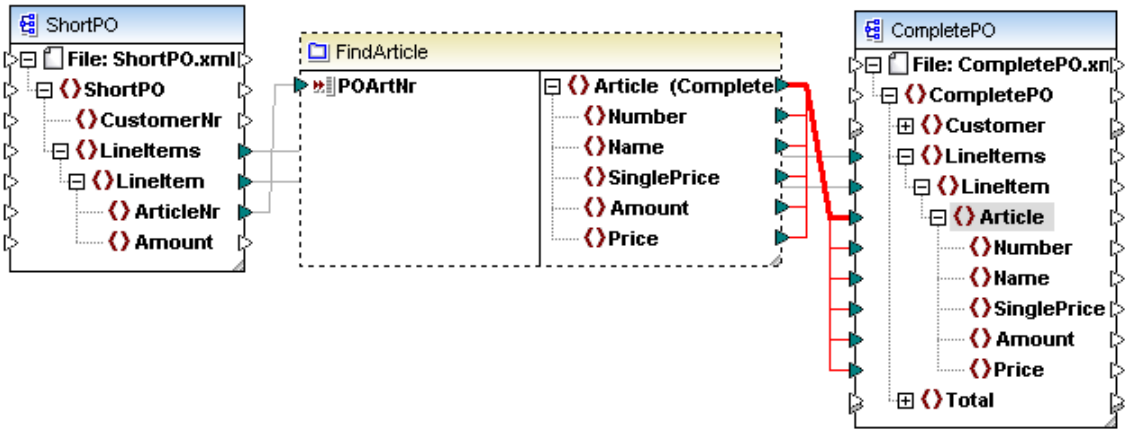
Note that the user-defined function FindArticle consists of two halves.

A left half which contains the input parameter:

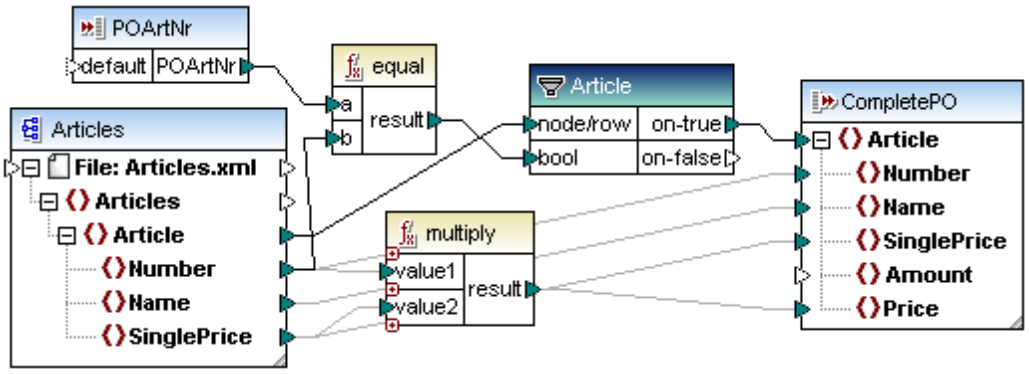
- a simple input parameter **POArtNr**

A right half which contains:

- a complex output component **Article (CompletePO)** with its XML child nodes mapped to CompletePO.



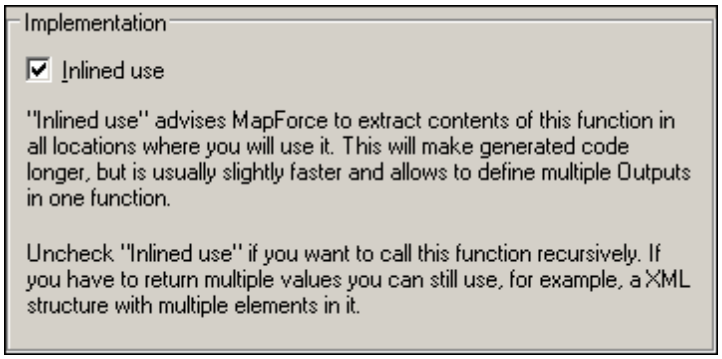
The screenshot below shows the constituent components of the user-defined function, the input components at left and the complex output component at right.



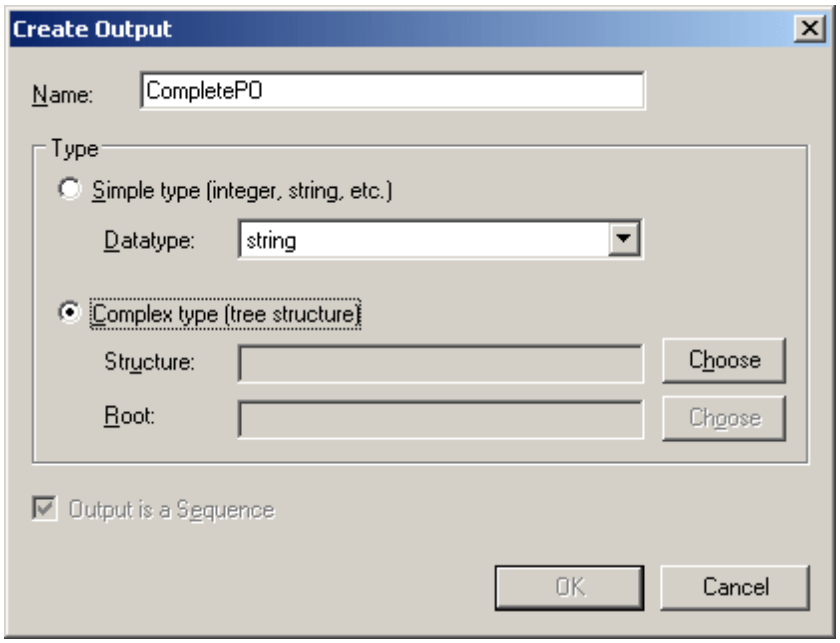
8.2.6.1 Defining Complex Output Components

Follow these steps to create a function that returns an XML structure as output parameter:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** name it **FindArticle**, and click OK to confirm. Note that the **Inline...** option is automatically selected.



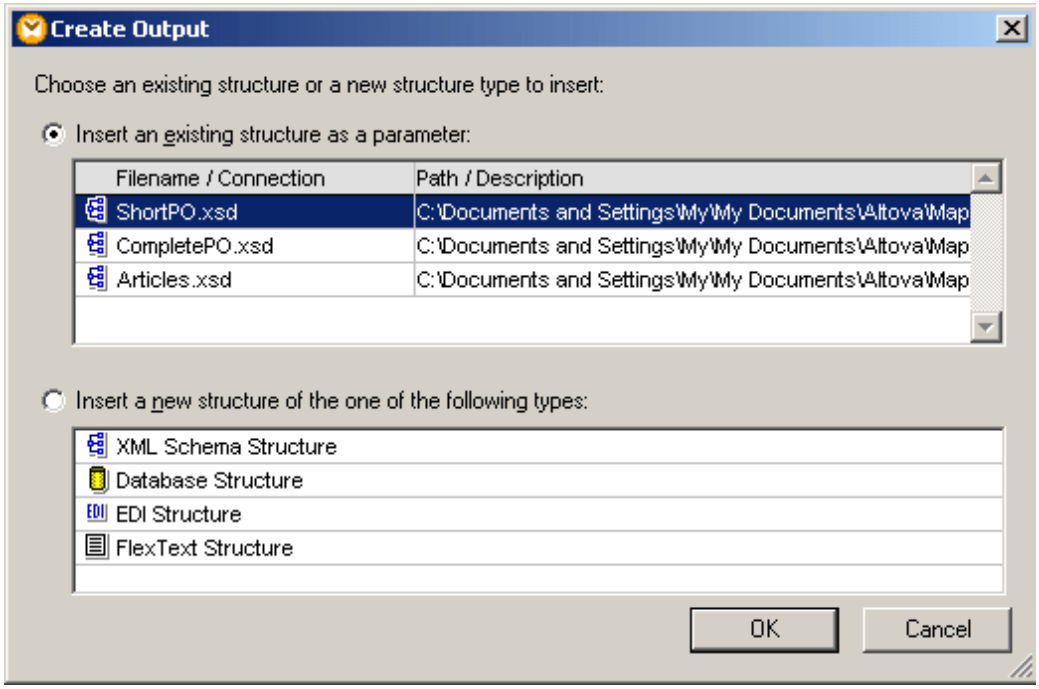
2. Click the Insert **Output** icon  in the icon bar, and enter a name e.g. CompletePO.



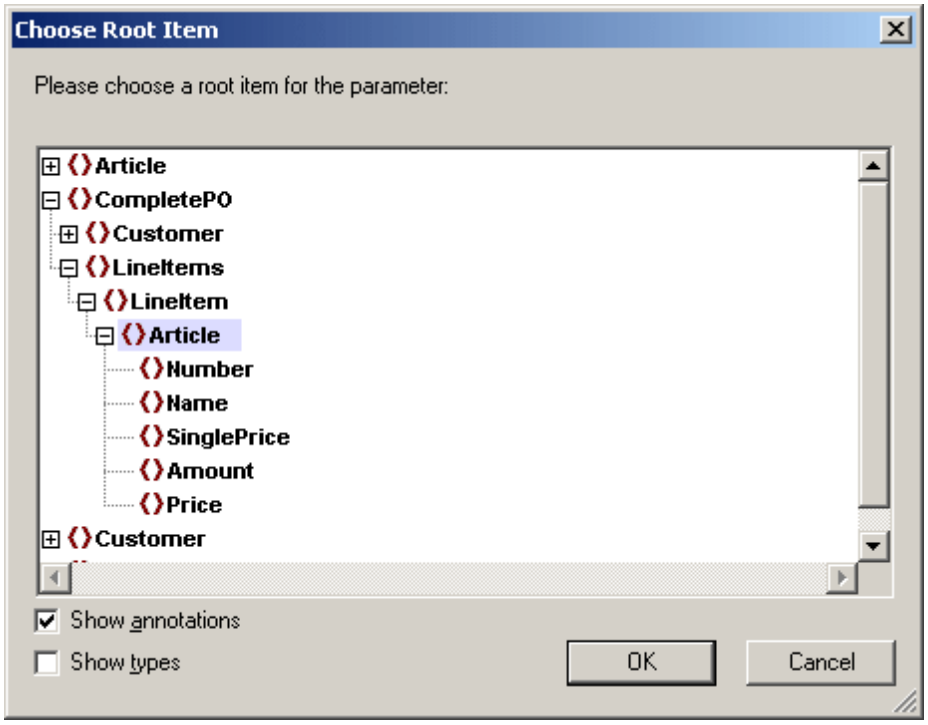
3. Click the **Complex type...** radio button, then click the "**Choose**" button.
This opens another dialog box.

   The top list box displays the **existing** components in the mapping, (three schemas if you opened the example file). Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML Schema , database, EDI file, or FlexText structure file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, EDI file, or FlexText structure file.

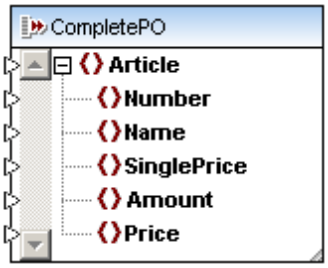


4. Click "**Insert new structure...** " radio button, select the XML Schema Structure entry, and click OK to continue.
5. Select the **CompletePO.xsd** from the "Open" dialog box.
6. Click the element that you would like to become the root element in the component, e.g. **Article**, and click OK, then OK again to close the dialog boxes.

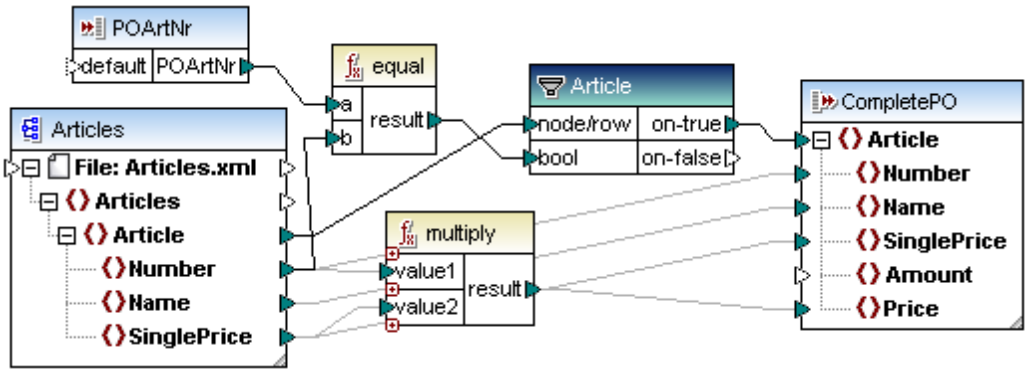


The CompletePO component is inserted into the user-defined function. Please note the

output icon to the left of the component name. This shows that the component is used as a complex output component.

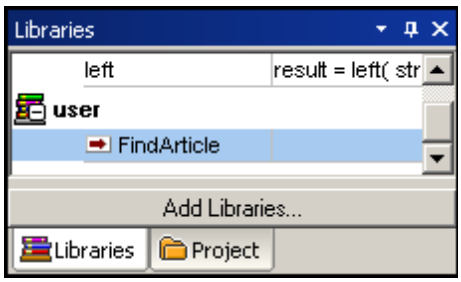


7. Insert the Articles schema/XML file into the user-defined function and assign the **Articles.xml** as the XML instance.
8. Insert the rest of the components as shown in the screenshot below, namely: the "simple" input components (POArtNr), filter, equal and multiply components, and connect them as shown.



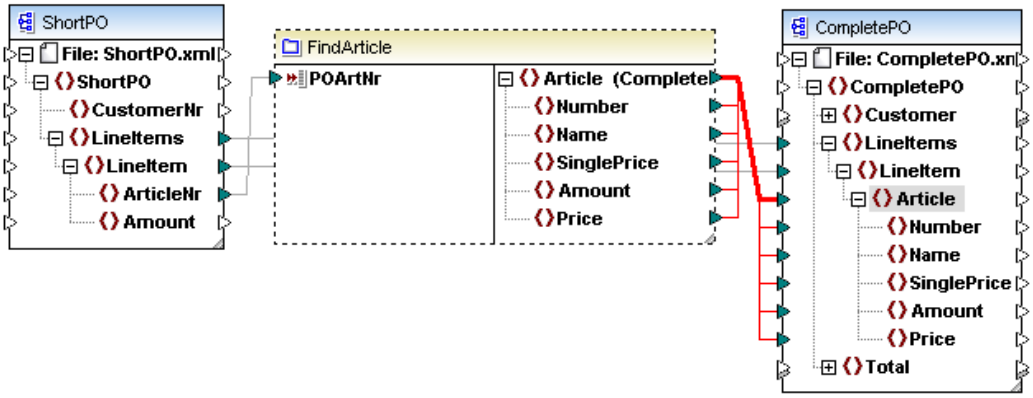
- The **Articles** component receives its data from the Articles.xml instance file, within the user-defined function.
- The input components supply the POArtNr (article number) and Amount data to which the Articles | Number & Price are compared.
- The filter component filters the records where both numbers are identical, and passes them on to the CompletePO output component.

9. Click the Home icon to return to the mapping.
10. Drag the newly created user-defined component from the Libraries pane into the mapping.



11. Create the connections as shown in the screenshot below.
Having created the Article (CompletePO) connector to the target, right click it and select

"Copy-all" from the context menu. The rest of the connectors are automatically generated, and are highlighted in the screenshot below.



Please note:
When creating **Copy-all** connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema. It is not necessary that they both have the same root elements however.

The left half contains the input parameter to which a single item is mapped; ShortPO supplies the article number to the **POArtNr** input component.

The right half contains a complex output component called "**Article (CompletePO)**" with its XML child nodes, which maps the filtered items, of the same Article number, to CompletePO.

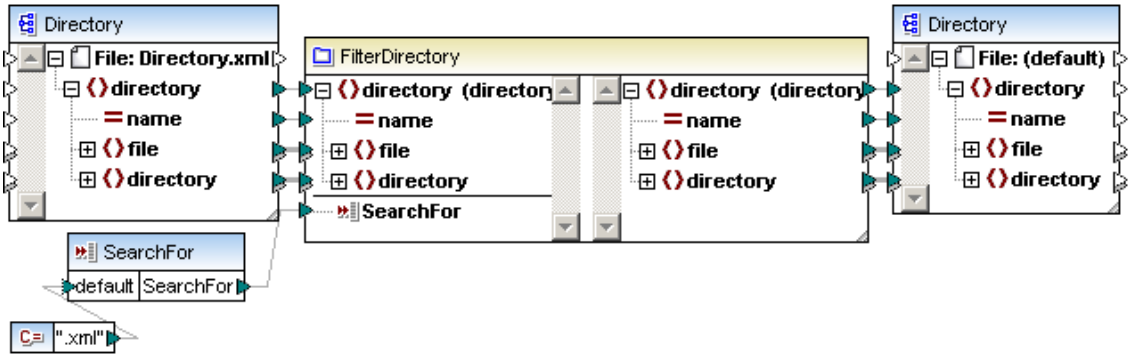
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
3    <LineItems>
4      <LineItem>
5        <Article>
6          <Number>3</Number>
7          <Name>Pants</Name>
8          <SinglePrice>34</SinglePrice>
9          <Price>102</Price>
10       </Article>
11     </LineItem>
12     <LineItem>
13       <Article>
14         <Number>1</Number>
15         <Name>T-Shirt</Name>
16         <SinglePrice>25</SinglePrice>
17         <Price>25</Price>
18       </Article>
```

8.2.7 Recursive user-defined mapping

This section will describe how the mapping **RecursiveDirectoryFilter.mfd**, available in the ... **\MapForceExamples** folder, was created and how recursive mappings are designed. The MapForceExamples project folder contains further examples of recursive mappings.

The screenshot below shows the finished mapping containing the recursive user-defined function

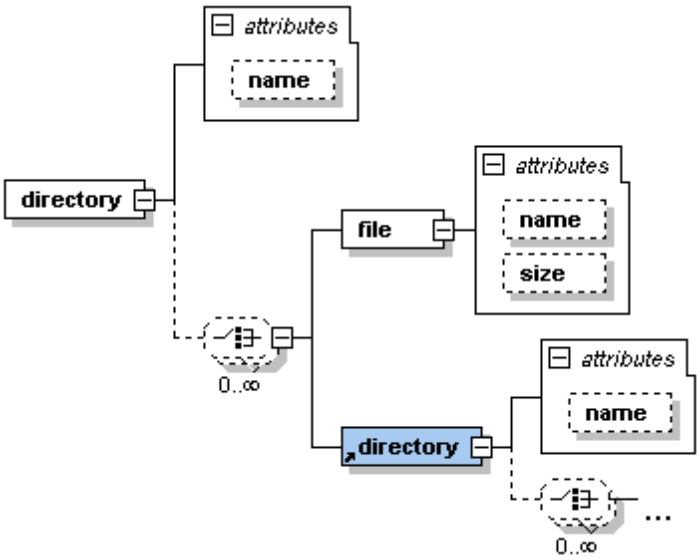
FilterDirectory, the aim being to filter a list of the .xml files in the source file.



The **source file** that contains the file and directory data for this mapping is Directory.xml. This XML file supplies the directory and file data in the hierarchical form you see below.

```
24    <directory name="output">
25      <file name="examplesite1.css" size="3174"/>
26      <directory name="images">
27        <file name="blank.gif" size="88"/>
28        <file name="block_file.gif" size="13179"/>
29        <file name="block_schema.gif" size="9211"/>
30        <file name="nav_file.gif" size="60868"/>
31        <file name="nav_schema.gif" size="6002"/>
32      </directory>
33    </directory>
34  </directory>
35  <directory name="Import">
36    <file name="altova.mdb" size="266240"/>
37    <file name="Data_shape.mdb" size="225280"/>
38  </directory>
39  <directory name="IndustryStandards">
40    <directory name="News">
41      <file name="high-tide.jpg" size="10793"/>
42      <file name="Newsml-example.xml" size="5004"/>
43      <file name="nitf-example.xml" size="9327"/>
44    </directory>
```

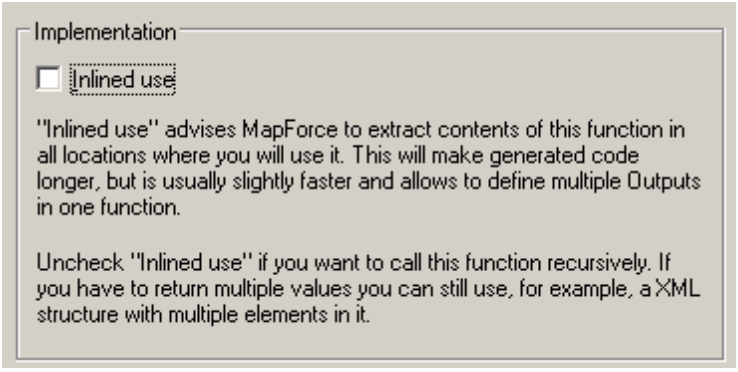
The XML schema file referenced by Directory.xml has a **recursive** element called "directory" which allows for any number of subdirectories and files below the directory element.



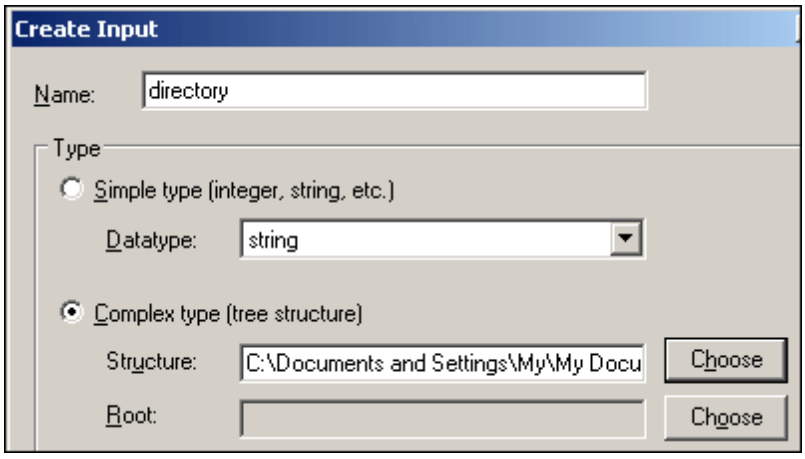
8.2.7.1 Defining a recursive user-defined function

Follow these steps to create a recursive user-defined function:

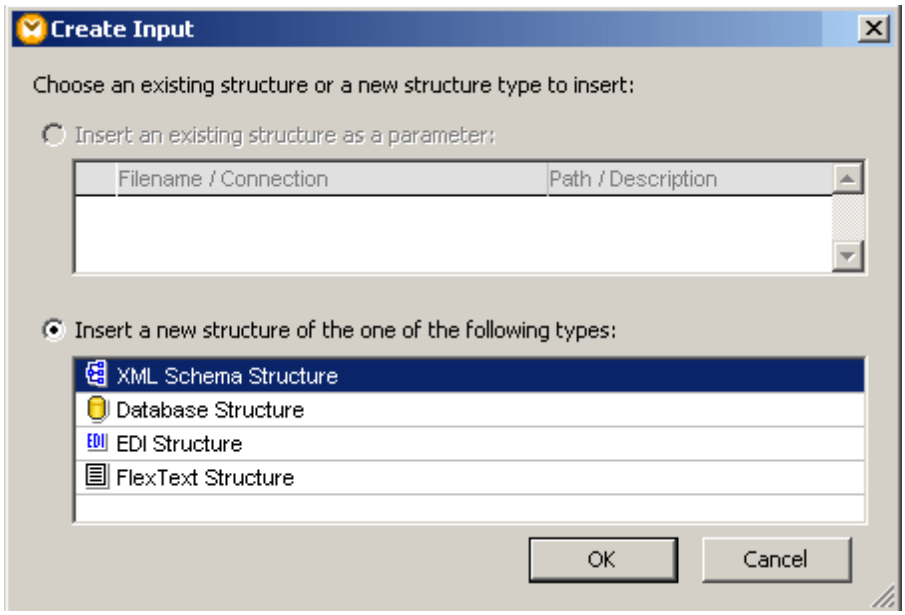
1. Select **Function | Create User defined Function** to start designing the function and enter a name e.g. FilterDirectory.
2. Make sure that you **deselect** the **Inlined Use** check box in the Implementation group, to make the function recursive, then click OK.



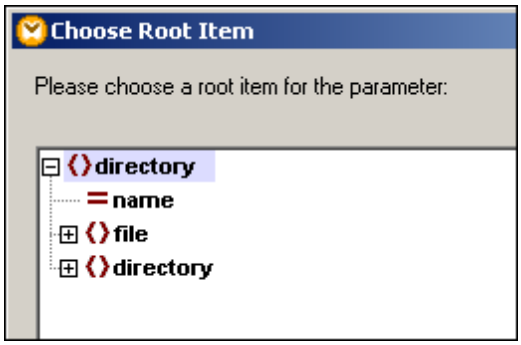
   You are now in the **FilterDirectory** window where you create the user-defined function.
3. Select **Function | Insert Input** to insert an input component.
4. Give the component a name e.g. "directory" and click on the **Complex Type** (tree structure) radio button.



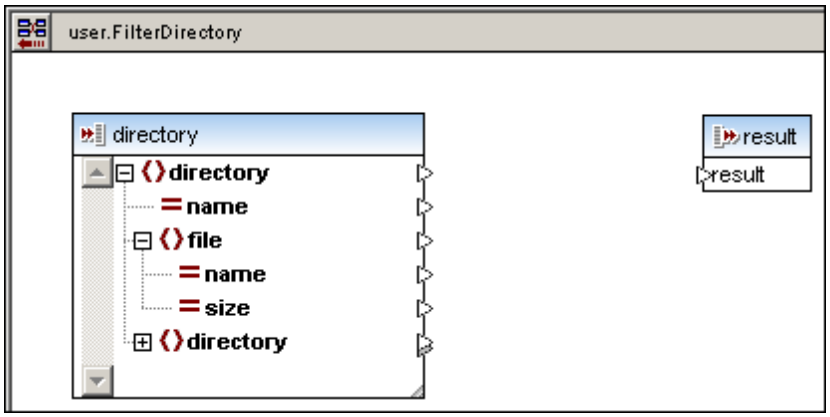
5. Click the **Choose** button, click the "XML Schema Structure" entry in the lower pane, then click OK.



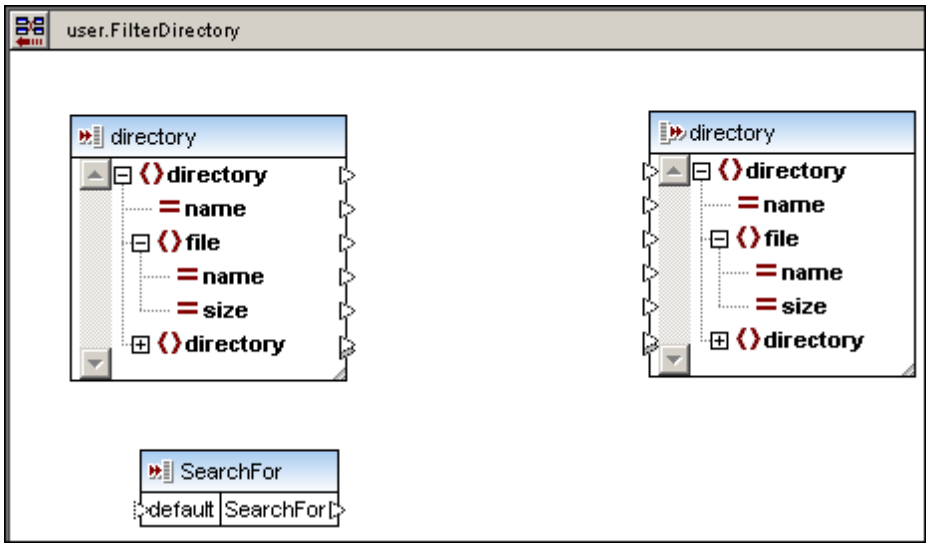
6. Select the **Directory.xsd** file in the ...\MapForceExamples folder and click the Open button.
7. Click OK again when asked to select the root item, which should be "directory" as shown below.



8. Click OK again to insert the complex input parameter.
   The user-defined function is shown below.



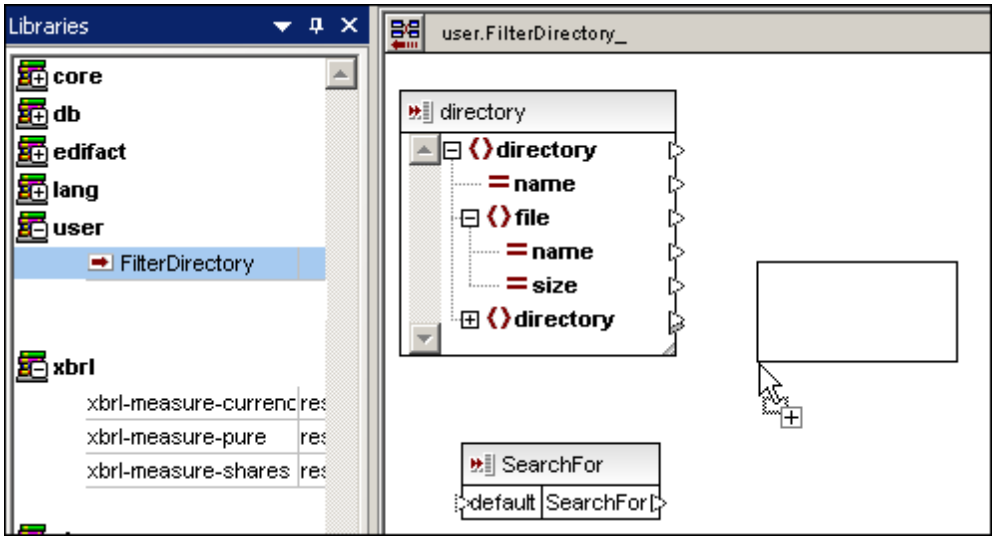
9. Delete the simple result output component, as we need to insert a complex output component here.
10. Select **Function | Insert Output...** to insert an **output** component and use the same method as outlined above, to make the output component, "directory", a complex type. You now have two complex components, one input and the other output.
11. Select **Function | Insert Input...** and insert a component of type Simple type, and enter a name e.g. **SearchFor**. Deselect the "Input is required" checkbox.



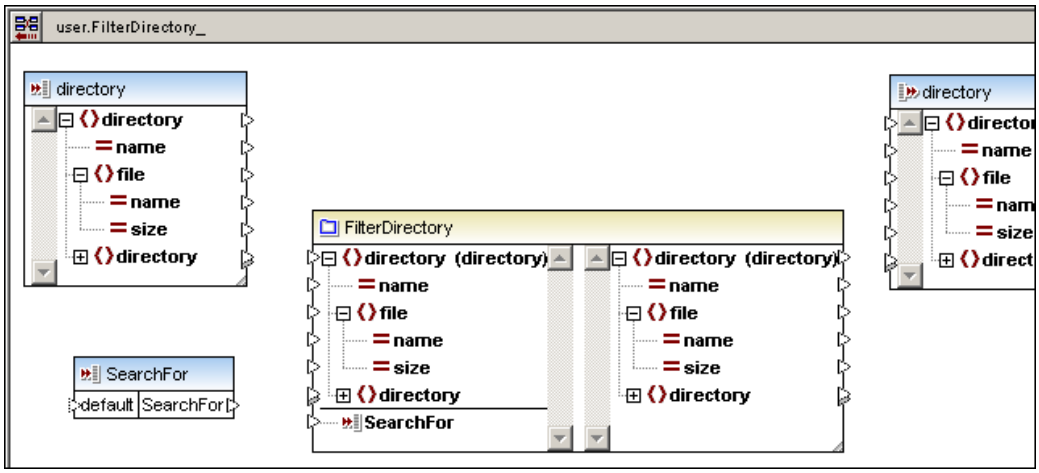
Inserting the recursive user-defined function

At this point, all the necessary input and output components have been defined for the user-defined function. What we need to do now is insert the "unfinished" function into the current user-defined function window. (You could do this at almost any point however.)

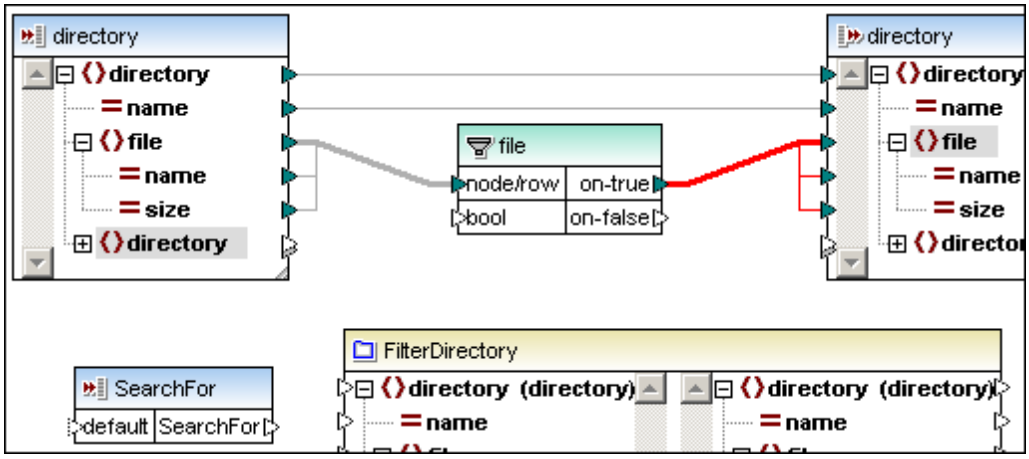
1. Find the FilterDirectory function in the **user** section of the **Libraries** window.
2. Click FilterDirectory then drag and drop it into the FilterDirectory window you have just been working in.



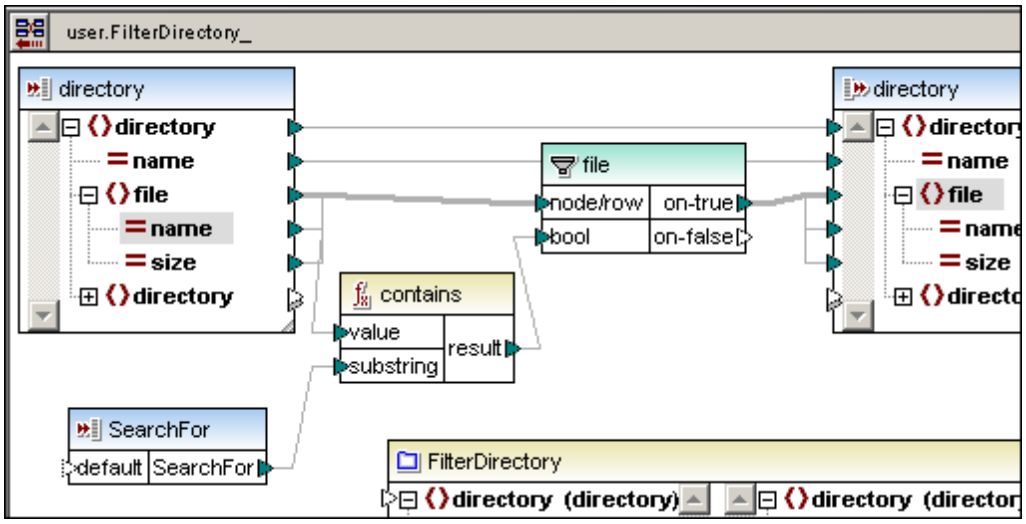
The user-defined function now appears in the user-defined function window as a recursive component.



3. Connect the **directory, name** and **file** items of the input component to the same items in the output component.
4. Right click the connector between the **file** items and select "Insert Filter" to insert a filter component.
5. Right click the on-true connector and select **Copy-All** from the context menu.
   The connectors change appearance to Copy-All connectors.



6. Insert a Contains function from the **Core | String functions** library.
7. Connect **name** to **value** and the **SearchFor** parameter to **substring**, then result to the bool item of the filter.



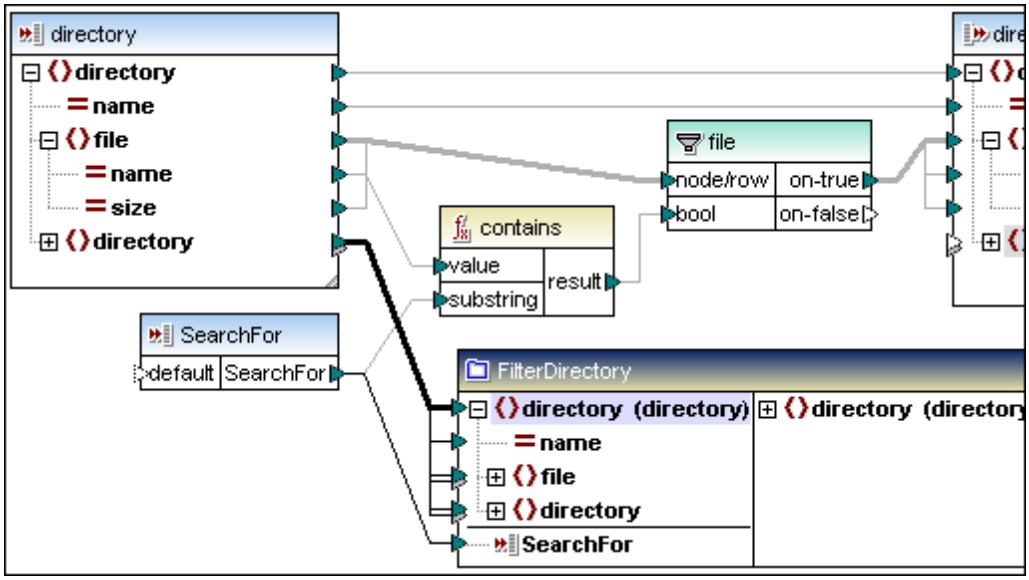
8. Connect the SearchFor item of the input component to the SearchFor item of the user-defined function.

Defining the recursion

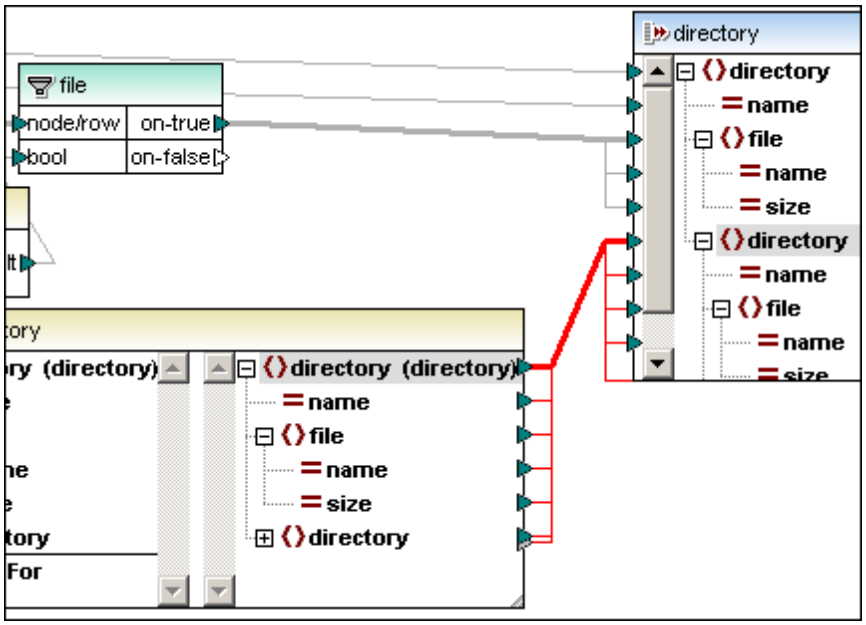
At this point, the mapping of a single directory recursion level is complete. Now we just need to define how to process a subdirectory.

Making sure that the Toggle Autoconnect icon is active in the icon bar:

1. Connect the lower **directory** item of the input component to the top **directory** item of the recursive user-defined function.



2. Connect the top output directory item of the user-defined function to the lower directory item of the output component.
3. Right click the top connector, select Copy-All from the context menu and click OK when prompted if you want to create Copy-All connection.

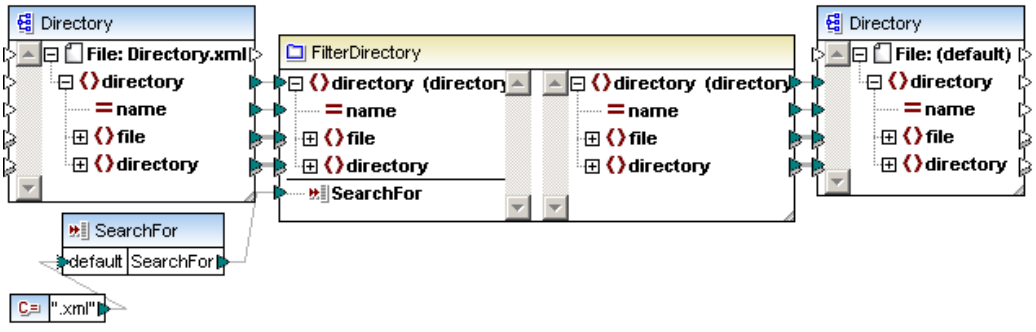


This completes the definition of the user-defined function in this window.

Click the Return to main mapping window icon,  to continue defining the mapping there.

Main Mapping window

1. Drag the FilterDirectory function from the **user** section of the Libraries window, into the **main** mapping area.
2. Use **Insert | XML Schema file** to insert Directory.xsd and select Directory.xml as the instance file.
3. Use the same method to insert Directory.xsd and select Skip, to create the output component.
4. Insert a constant component, then a Input component e.g. SearchFor.
5. Create the connections as shown in the screenshot below.
6. When connecting the top-level connectors, directory to directory, on both sides of the user-defined component, right click the connector and select **Copy-All** from the context menu.



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

Notes:
Double clicking the lowest "directory" item in the Directory component, opens a new level of recursion, i.e. you will see a new **directory | file | directory** sublevel. Using the Copy-all connector automatically uses all existing levels of recursion in the XML instance, you do not need expand the recursion levels manually.

8.3 Adding custom XSLT functions

MapForce allows you to extend the installed XSLT function libraries with your own custom functions. This option is made available when you select XSLT as transformation language.

XSLT files appear as libraries, and display all **named templates** as functions below the library name.

- Functions must be declared as Named Templates conforming to the XSLT specification in the XSLT file.
- If the imported XSLT file imports or includes other XSLT files, then these XSLT files and functions will be imported as well.
- Each named template appears as a function below each library name.
- The amount of mappable input icons depends on the number of parameters used in the template call; optional parameters are also supported.
- Updates to imported XSLT files occur at program start or whenever the files change.
- Namespaces are supported

Note: 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.

See also:

XSLT 1.0 engine implementation
XSLT 2.0 engine implementation

8.3.1 Adding custom XSLT 1.0 functions

The files needed for the simple example shown below, are available in the **...\MapForceExamples** directory.

- Name-splitter.xslt
- Name-splitter.xml (the XML instance file for Customers.xsd)
- Customers.xsd
- CompletePO.xsd

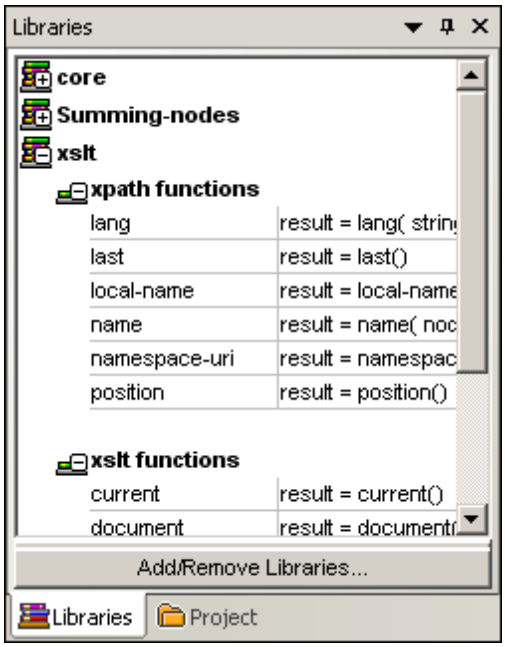
For an additional example of using named templates to sum nodes, see Aggregate functions.

To add a custom XSLT function:

1. Create an XSLT file that achieves the transformation/result you want.
   The example below, **Name-splitter.xslt**, shows a named template called "**tokenize**" with a single parameter "string". What the template does, is work through an input string and separate capitalized characters with a space for each occurrence.

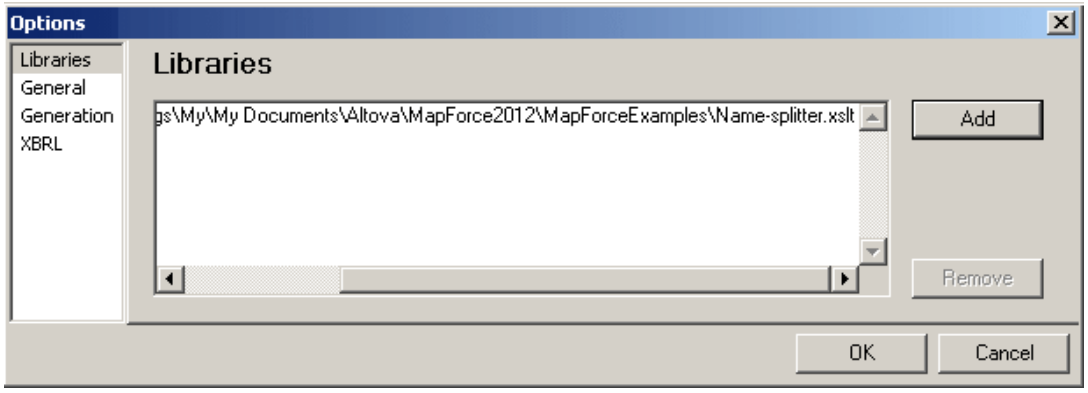
```
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
4
5    <xsl:template match="*">
6      <xsl:for-each select=".">
7      <xsl:call-template name="tokenize">
8      <xsl:with-param name="string" select="."/>
9      </xsl:call-template>
10     </xsl:for-each>
11   </xsl:template>
12
13   <xsl:template name="tokenize">
14     <xsl:param name="string" select="."/>
15     <xsl:variable name="caps" select="translate($string, '-abcdefghijklmnopqrstuv
16     <xsl:variable name="capscount" select="string-length($caps)"/>
17     <xsl:variable name="token">
```

2. Click the **Add/Remove Libraries** button, and then click the Add button in the following dialog box.

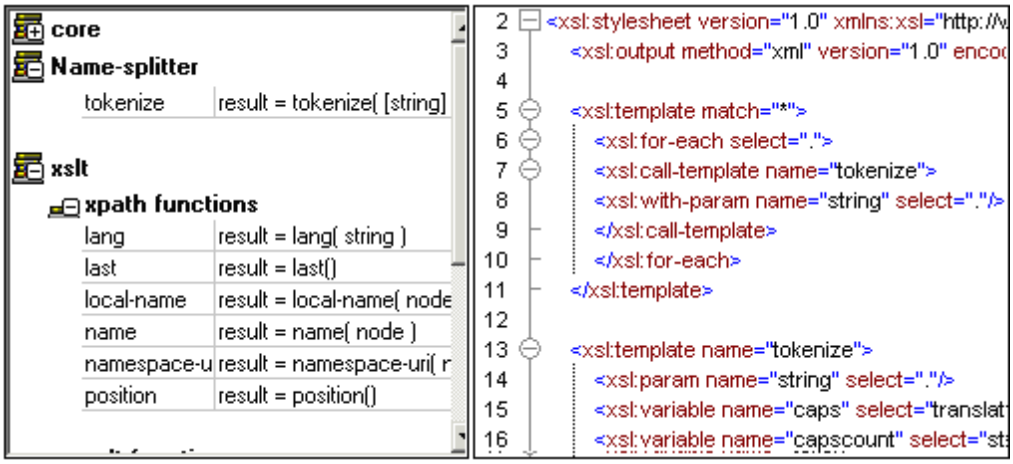


XSLT Selected

3. Select the XSL, or XSLT file, that contains the named template you want to act as a function, in this case **Name-splitter.xslt**. The XSLT file appears in the Libraries tab.



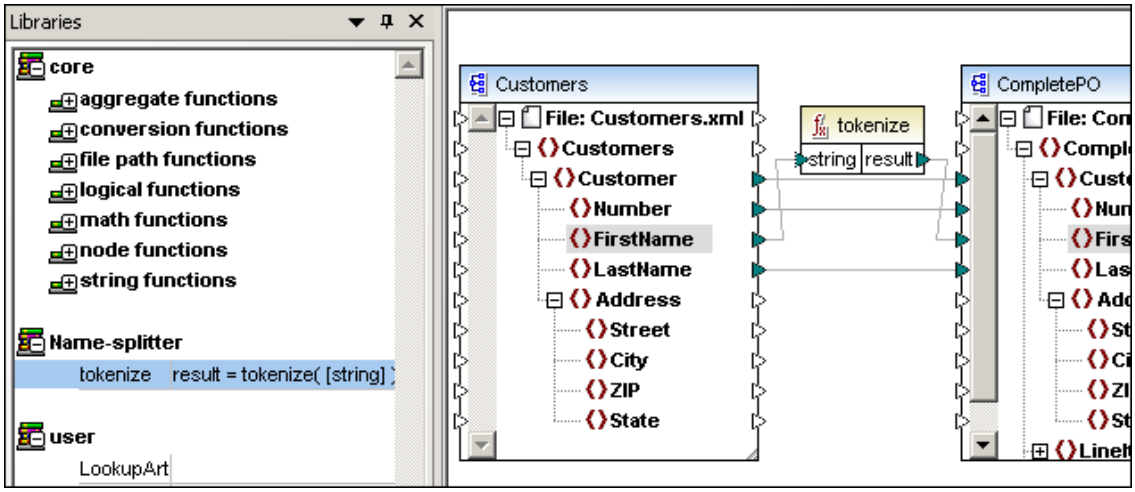
4. Click **OK** to insert the new function.



XSLT Selected

The XSLT file name appears in the library window, along with the function(s) defined as named templates, below it. In this example **Name-splitter** with the **tokenize** function.

5. Drag the function into the Mapping window, to use it in you current mapping, and map the necessary items, as show in the screenshot below.



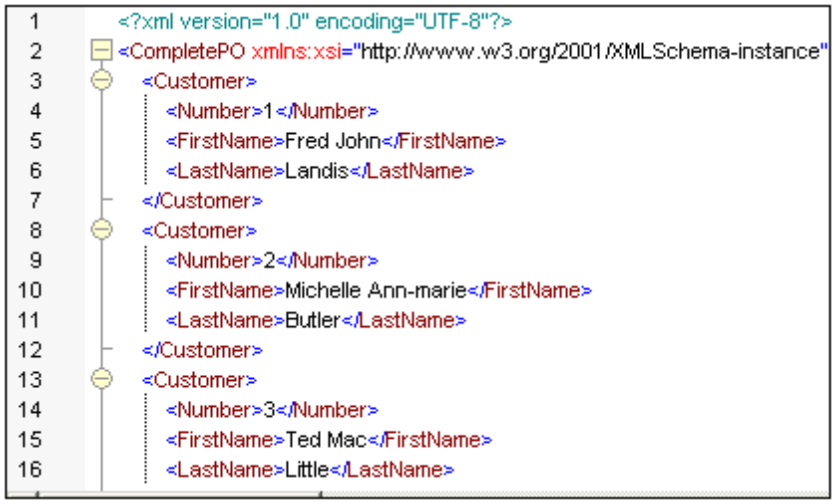
XSLT Selected

6. Click the XSLT tab to see the generated XSLT code.

Please 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**.

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



To delete custom XSLT functions:

1. Click the **Add/Remove Libraries** button.
2. Click to the specific XSLT **library name** in the Libraries tab
3. Click the Delete button, then click OK to confirm.

8.3.2 Adding custom XSLT 2.0 functions

MapForce also allows you to import XSLT 2.0 functions that occur in an XSLT 2.0 document in the form:

<xsl:function name="MyFunction">

For an additional example of using named templates to sum nodes, see Aggregate functions.

Datatypes in XPath 2.0
If your XML document references an XML Schema and is valid according to it, you must explicitly construct or cast datatypes that are not implicitly converted to the required datatype by an operation.

In the XPath 2.0 Data Model used by the Altova XSLT 2.0 Engine, all **atomized** node values from the XML document are assigned the `xs:untypedAtomic` datatype. The `xs:untypedAtomic` type works well with implicit type conversions.

For example,

- the expression `xs:untypedAtomic("1") + 1` results in a value of 2 because the `xdt:untypedAtomic` value is **implicitly** promoted to `xs:double` by the addition operator.
- Arithmetic operators implicitly promote operands to `xs:double`.
- Value comparison operators promote operands to `xs:string` before comparing.

See also:

XSLT 2.0 engine implementation

8.3.3 Aggregate functions - summing nodes in XSLT1 and 2

This section describes the method you can use to process multiple nodes of an XML instance document and have the result mapped as a single value to a target item. The files used in this example are available in the ...\MapForceExamples\Tutorial\ folder and consists of:

Summing-nodes.mfd	mapping file
input.xml	input XML file
input.xsd and output.xsd	source and target schema files
Summing-nodes.xslt	xslt file containing a named template to sum the individual nodes

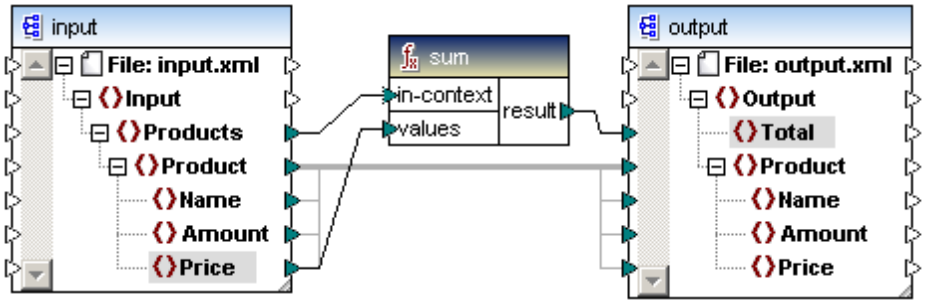
There are two separate methods of creating and using aggregate functions:

- Using the aggregate functions available in the **core library** of the Library pane
- Using a **Named Template**.

Aggregate functions - library

Depending on the XSLT library you select, XSLT 1 or XSLT 2, different aggregate functions are available in the core library. XSLT 1 supports count and sum, while XSLT 2 supports avg, count, max, min, string-join and sum.

Drag the aggregate function that you use from the library into the mapping area and connect the source and target components as shown in the screenshot below.



For more information on this type of aggregate function, please also see Aggregate functions.

Aggregate function - Named template

The screenshot below shows the **XML input** file. The aim of the example is to sum the Price fields of any number of products, in this case products A and B.

```
<?xml version="1.0" encoding="UTF-8"?>
<Input xmlns:xsi="http://www.w3.org/2001/XMLSchema
    <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>
```

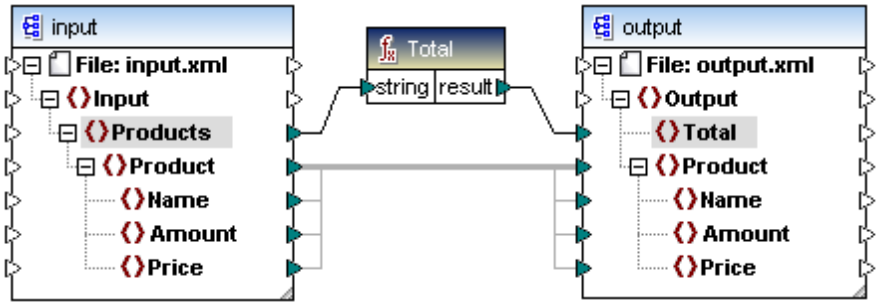
The screenshot below shows the XSLT stylesheet which uses the named template "**Total**" and a single parameter "string". What the template does, is work through the XML input file and sum all the values obtained by the XPath expression **/Product/Price**, in the document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/19
    <xsl:output method="xml" version="1.0" encoding="UTF-8" i

    <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>
```

1. Click the **Add/Remove Libraries** button, and select the Libraries tab of the Options dialog box.
2. Click the Add button and select the **Summing-nodes.xslt** file from the **<Documents> \Altova\MapForce2016\MapForceExamples** folder.
3. Drag in the Total function from the newly created Summing-nodes library and create the mappings as shown below.



4. Click the Output tab to preview the mapping result.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Output xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNa
3    <Total>25</Total>
4    <Product>
5      <Name>ProductA</Name>
6      <Amount>10</Amount>
7      <Price>5</Price>
8    </Product>
9    <Product>
10     <Name>ProductB</Name>
11     <Amount>5</Amount>
12     <Price>20</Price>
13   </Product>
14 </Output>
15
```

The two Price fields of both products have been added and placed into the Total field.

To sum the nodes in XSLT 2.0:

- Change the stylesheet declaration in the template to ... version="2.0".

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="2.0" xmlns:xs
3      <xsl:output method="xml" version="
```

8.4 Adding custom XQuery functions

MapForce allows you to import XQuery library modules.

See also:

XQuery engine implementation

8.5 Adding custom Java .class and .NET DLL functions

Compiled Java class files as well as .NET assembly files (including .NET 4.0 assemblies) can be added to the Libraries pane and used as any other function available in MapForce. The mapping output of these Java and .NET functions can be previewed in the Output pane and the functions are available in generated code.

Support notes:

- Compiled Java class (.class) files are supported when the Output language is set to Java.
- .NET assembly files are supported when the Output language is set to C#. .NET assemblies are generally supported irrespective of the originating language (C++ or VB.NET), provided they use only the basic data types from the System Assembly as parameters and return types.

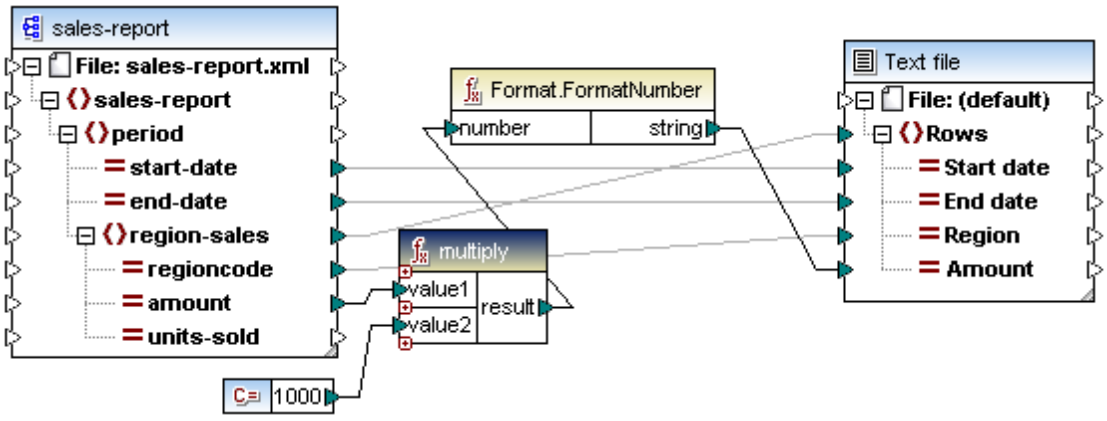
Not supported:

- Native C++ DLLs (or other DLLs that are not a .NET assembly)
- Using .class or DLL files/functions, while having the Output language set to C++
- Using Java or .NET functions directly in XSLT (a custom XSLT function that acts as an adapter, would have to be written)
- C++ functions cannot be previewed in the Output window, but are available in the generated code.

Note: Java SE 6 Runtime Environment or later is needed to complete this example.

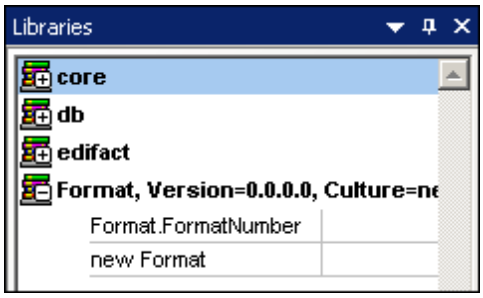
To add custom Java or .NET functions, you need the compiled Java classes (.class) or the .NET assembly files (.dll).

MapForce example files that show how these functions are used, are available in the ... \MapForceExamples**Java** and ...\MapForceExamples**C#** folders. Both example files are called **FormatNumber.mfd**.



To add the .NET assembly file:

1. Select C# as transformation language.
2. Click the Add/Remove Libraries... button (of the Libraries pane) and select the **Format.dll** file from the ...\MapForceExamples**C#\Format\bin\Debug** directory. A message appears telling you that a new function has been added. The function is now visible under "Format" in the library pane.



3. Open the **FormatNumber.mfd** file available in the ...\MapForceExamples**C#** folder (screenshot shown above).
4. Click the Output button to see the result of the mapping.

```
 1   Start date,End date,Region,Amount
 2   2008-01-01,2008-01-31,CA,"110.400,00"
 3   2008-01-01,2008-01-31,MA,"75.300,00"
 4   2008-02-01,2008-02-29,CA,"114.300,00"
 5   2008-02-01,2008-02-29,MA,"65.200,00"
 6   2008-03-01,2008-03-31,CA,"134.200,00"
 7   2008-03-01,2008-03-31,MA,"86.100,00"
 8   2008-04-01,2008-04-30,CA,"107.300,00"
 9   2008-04-01,2008-04-30,MA,"112.100,00"
10   2008-05-01,2008-05-31,CA,"114.400,00"
11   2008-05-01,2008-05-31,MA,"93.800,00"
```

Please see: Java and .NET functions - specifics for more detail on the implementation.

8.6 Adding custom Java, C# and C++ function libraries

MapForce allows you to create and add your own function libraries for Java, C# and C++. These functions can then be used in the graphical mapping similar to built-in functions.

Libraries can be added by clicking the Add/Remove Libraries button under the Libraries pane, or by selecting the menu option **Tools | Options | Add** of the Libraries tab. Libraries can be Java .**class**, Visual Studio C# .**DLL**, as well as files with an **.mff** extension.

Please note: .**MFF functions**
Many mappings using these types of custom functions (.mff) **can be previewed** by clicking the **Output** tab, i.e. using the Built-in execution engine. This applies to C# and Java functions, but only those that use the native language types, not those that use the generated Altova classes. All these functions are available when generating code.

User-defined functions created graphically in a **mapping** cannot and need not be saved/assigned to an *.mff file, as they are saved as part of the mapping file. Please see User-defined functions for more information on how to import and otherwise manage user-defined functions.

The example shown below requires you to have Java SE Runtime Environment installed.

To be able to add custom *.mff functions, you need:

- the mff file which tells MapForce what the interfaces to the functions are, and
- where the implementation can be found for the generated code. This implementation is a class in the respective programming language that contains the static methods defined in the mff file.

A basic mff file for C# would for example look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://
www.w3.org/2001/XMLSchema" xsi:noNamespaceSchemaLocation="mff.xsd" version="8"
library="helloworld">
	<implementations>
		<implementation language="cs">
			<setting name="namespace" value="HelloWorldLibrary"/>
			<setting name="class" value="Greetings"/>
			<setting name="reference" value="C:\HelloWorldLibrary
\HelloWorldLibrary.dll"/>
		</implementation>
	</implementations>
	<group name="string functions">
		<component name="hello">
			<sources>
				<datapoint name="greeting_type" type="xs:boolean"/>
			</sources>
			<targets>
				<datapoint name="result" type="xs:string"/>
			</targets>
			<implementations>
				<implementation language="cs">
					<function name="HelloFunction"/>
				</implementation>
```

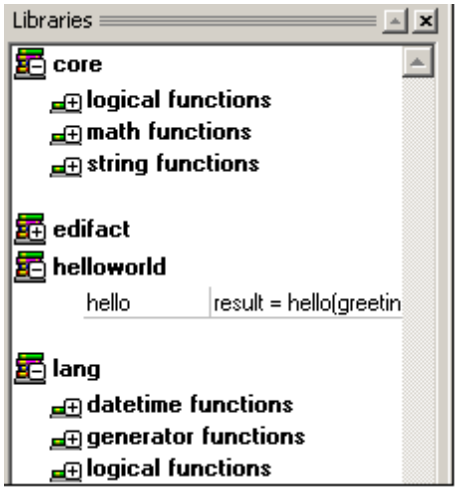


```
                    </implementations>
                    <description>
                          <short>result = hello(greeting_type)</short>
                          <long>Returns a greeting sentence according to the
given greeting_type.</long>
                    </description>
               </component>
       </group>
</mapping>
```

Please note:

The ***.mff** library files must be valid against the **mff.xsd** schema file available in the ... \MapForceLibraries directory. That schema defines the custom library configuration and is for internal use only. Altova GmbH retains the right to change this file format with new releases.

The image below shows the appearance of the mff file in MapForce. The new library "helloworld" appears as a library entry (sorted alphabetically), containing the "hello" string function.



Java Selected

Mff files can be written for more than one programming language. Every additional language must therefore contain an additional <implementation> element. The specifics on the implementation element are discussed later in this document.

Please note that the exact order in which functions are called by the generated mapping code is undefined. MapForce may choose to cache calculated results for reuse, or evaluate expressions in any order. It is therefore strongly recommended to use only functions that have no side effects.

8.6.1 Configuring the mff file

The steps needed to adapt the mff file to suit your needs, are described below.

The Library Name:
The library name is found in the mff file line shown below. By convention, the **library name** is written in **lowercase** letters.

```
<mapping          xmlns:xsi="http://www.w3.org/2001/XMLSchema-
```

```
instance"
                        xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xsi:noNamespaceSchemaLocation="mff.xsd"
                        version="8" library="helloworld">
```

The entry that will appear in the libraries window will be called "helloworld". Note that the library may **not** appear **immediately** after you have clicked the Add button in the Settings dialog box. Libraries are only displayed if at least one component exists containing an implementation for the currently selected programming language.

Libraries and their functions can be toggled on or off, by deleting or adding the respective library file (*.mff).

To add the new mff file to the libraries pane:

1. Click the "Add/Remove libraries" button.
2. Click the "Add" button in the libraries dialog box.
3. Select the *.MFF library you want to include, and click Open to load the file in the Options dialog box.

Please note:

If you save the *.**mff** file in the ...**MapForceLibraries** folder, then the library will be automatically loaded when you start MapForce and will be available immediately for all mappings.

Implementations Element for the helloworld library:

```
...
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://
www.w3.org/2001/XMLSchema"xsi:noNamespaceSchemaLocation="mff.xsd" version="8"
library="helloworld">
       <implementations>
              <implementation language="cs">
                     <setting name="namespace" value="HelloWorldLibrary"/>
                     <setting name="class" value="Greetings"/>
                     <setting name="reference" value="C:\HelloWorldLibrary
\HelloWorldLibrary.dll"/>
              </implementation>
       </implementations>
...
```

For each language that the helloworld library should support, an implementations element has to be added. The settings within each implementation allow the generated code to call the specific functions defined in Java, C++ or C#.

The specific settings for each programming language will be discussed below.

Java:

```
...
<implementation language="java">
       <setting name="package" value="com.hello.functions"/>
       <setting name="class" value="Hello"/>
</implementation>
...
```


It is important for the generated code to be able to find your **Hello.class** file. This can be achieved by making sure that it is entered in the Java CLASSPATH. The default classpath is found in the system environment variables.

Note that it is only possible to have one class per *.mff file when working with custom Java libraries.

C#:

```
...
	<implementation language="cs">
		<setting name="namespace" value="HelloWorldLibrary"/>
		<setting name="class" value="Hello"/>
		<setting name="reference" value=" C:\HelloWorldLibrary
\HelloWorldLibrary.dll "/>
	</implementation>
...
```

Note for C# : it is very important that the code uses the namespace which is defined here. C# also needs to know the location of the **dll** that is to be linked to the generated code.

C++:

```
...
	<implementation language="cpp">
			<setting name="namespace" value="helloworld"/>
			<setting name="class" value="Greetings"/>
			<setting name="path" value="C:\HelloWorldLibrary"/>
			<setting name="include" value="Greetings.h"/>
			<setting name="source" value="Greetings.cpp"/>
	</implementation>
...
```

- **namespace** is the namespace in which your **Greetings** class will be defined. It must be equal to the library name.
- **path** is the path in which the include and the source files are to be found.
- When code for a mapping is generated, the include and source files will be copied to the directory **targetdir/libraryname** (defined when selecting the menu option **File | Generate xxx code**, and selecting the directory) and included in the project file.

All the include files you supply will be included in the generated algorithm.

Adding a component:
Each component you will define, will be located within a function group. Staying with the helloworld example:

```
...
<group name="string functions">
	<component name="hello">
		…
	</component>
</group>
...
```

8.6.2 Defining the component user interface

The code shown below, defines how the component will appear when dragged into the mapping area.

```
...
<component name="hello">
	<sources>
		<datapoint name="greeting_type" type="xs:boolean"/>
	</sources>
	<targets>
		<datapoint name="result" type="xs:string"/>
	</targets>
	<implementations>
	…
	</implementations>
	<description>
		<short>result = hello(greeting_type)</short>
		<long>Returns a greeting sentence according to the given
greeting_type.</long>
	</description>
</component>
...
```

The new MapForce component:

Datapoints
Datapoints can be loosely defined as the input or output parameters of a function. The datapoints' type parameter specifies the parameters/return value type.

Please note:
Only one **target** datapoint, but multiple **source** datapoints are allowed for each function.

The datatype of each datapoint must be one of the following:

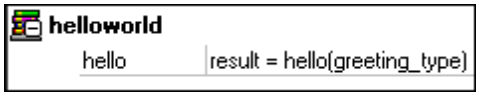
- one of the XML Schema types (eg. xs:string, xs:integer, etc.)

These datatypes have to correspond to the datatypes of the function's parameters you defined in your Java, C++ or C# library. For the mapping of XML Schema datatypes to language types, please see the tables in the Writing your libraries chapter under the particular programming language.

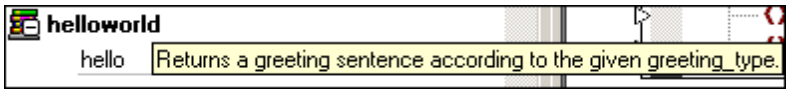
Altova has provided support for Schema simpleTypes (date, time, duration, dateTime) as classes, for each of the supported programming languages. The integration of these Schema simpleTypes in your library will be explained later in this document.

Function Descriptions:
Functions are accompanied by short and long descriptions in the library window. The short description is always shown to the right of the function name, while the long description is displayed as a ToolTip when you place the mouse cursor over the short description.

Short description:



Long description:



8.6.3 Function implementation details

We are now at the point where we need to make a connection between the function in the library window, and the function in the Java, C# or C++ classes. This is achieved with the <implementation> element.

As previously stated, one function may have multiple implementation elements – one for each supported programming language.

```
...
<component name="hello">
...
	<implementations>
		<implementation language="cs">
			<function name="HelloFunction"/>
		</implementation>
	</implementations>
...
</component>
...
```

A function may be called "helloFunction" in Java, or "HelloFunctionResponse" in C++. This is why you need to specify a separate function name for each programming language.

A function for each of the three programming languages might look like the following:

```
...
<component name="hello">
...
	<implementations>
		<implementation language="cs">
			<function name="HelloFunction"/>
		</implementation>
		<implementation language="java">
			<function name="helloFunction"/>
		</implementation>
		<implementation language="cpp">
			<function name="HelloFunctionResponse"/>
		</implementation>
	</implementations>
...
</component>
...
```

The value you supply as function name must of course exactly match the name of the method in the Java, C# or C++ class.

8.6.4 Writing your libraries

The implementation of a custom function library is a class in the respective programming language that contains static methods for each function defined in the mff file.

Please note:

If you are upgrading from a MapForce version before 2010, you may have to update the data types used in your custom functions.
The current mapping of XML Schema types to native datatypes can be found in the following sections.

The following sections describe how to create the libraries for a specific programming language:

Create a Java library
Create a C# library
Create a C++ library

8.6.4.1 Create a Java library

How to write a Java library:

1. Create a new Java class, using the previous example, name it "Hello".
2. Add the package name you provided under

```
...
<implementation language="java">
	<setting name="package" value="com.hello.functions"/>
	<setting name="class" value="Hello"/>
</implementation>
...
```

3. If you need special XML Schema types (e.g. date, duration, ...), add the line

```
import com.altova.types.*;
```

   The exact mapping of XML Schema datatypes to Java datatypes can be found in the table below.

   If you encounter problems finding the **com.altova.types** on your computer, please generate and compile Java code without custom functions; you will then find the classes in the directory you specified.

4. Add the functions you specified in the mff file as **public static**.

```
	package com.hello.functions;

	public class Hello {
		public static String HelloFunction ( boolean greetingType )
{
			if( greetingType )
				return "Hello World!";
			return "Hello User!";
		}
```



```
}
```

5. Compile the Java file to a class file, and add this to your Classpath. You have now finished creating your custom library.

Datatype Mapping

Schema Type	Java Type
anySimpleType	String
anyAtomicType	String
boolean	boolean
string	String
normalizedString	String
token	String
language	String
NMTOKEN	String
Name	String
NCName	String
ID	String
IDREF	String
ENTITY	String
untypedAtomic	String
dateTime	com.altova.types.DateTime
date	com.altova.types.DateTime
time	com.altova.types.DateTime
gYear	com.altova.types.DateTime
gYearMonth	com.altova.types.DateTime
gMonth	com.altova.types.DateTime
gMonthDay	com.altova.types.DateTime
gDay	com.altova.types.DateTime
duration	com.altova.types.Duration
base64Binary	byte[]
hexBinary	byte[]

Schema Type	Java Type
anyURI	String
QName	javax.xml.namespace.QName
NOTATION	String
double	double
float	double
decimal	java.math.BigDecimal
integer	java.math.BigInteger
nonPositiveInteger	java.math.BigInteger
negativeInteger	java.math.BigInteger
long	long
int	int
short	int
byte	int
nonNegativeInteger	java.math.BigInteger
positiveInteger	java.math.BigInteger
unsignedLong	java.math.BigInteger
unsignedInt	long
unsignedShort	long
unsignedByte	long
dayTimeDuration	com.altova.types.Duration
yearMonthDuration	com.altova.types.Duration
NMTOKENS	String
IDREFS	String
ENTITIES	String

8.6.4.2 Create a C# library

How to write a C# library:

1. Open a new Project in Visual Studio and create a class library.
2. Go to add reference, and add the **Altova.dll.**

   If you encounter problems finding **Altova.dll** on your computer, please generate and compile the C# code without custom functions; you will then find the DLL in the directory

you specified.

3. If you need special XML Schema types (e.g. date, duration, ...), add the line
   `using Altova.Types;`

   The exact mapping of XML Schema datatypes to C# datatypes can be found in the table below.

4. The class name should be the same as you specified (here "Greetings")

```
<implementation language="cs">
    <setting name="namespace" value="HelloWorldLibrary"/>
    <setting name="class" value="Greetings"/>
    <setting name="reference" value="C:\HelloWorldLibrary
\HelloWorldLibrary.dll"/>
</implementation>
```

5. Add the namespace using the same value as you specified in the mff implementation settings shown above.
6. Add your functions as **public static.**

The sample code should look like this:

```
using System;
using Altova.Types;

namespace HelloWorldLibrary
{
    public class Greetings
    {
        public static string HelloFunction(bool
GreetingType)
        {
            if( GreetingType )
                return "Hello
World!";

            return "Hello User!";
        }
    }
}
```

7. The last step is to compile the code.
   The path where the compiled dll is located must match the "reference" setting in the implementation element.

Datatype Mapping

Schema Type	C# Type
anySimpleType	string
anyAtomicType	string
boolean	bool
string	string
normalizedString	string

Schema Type	C# Type
token	string
language	string
NMTOKEN	string
Name	string
NCName	string
ID	string
IDREF	string
ENTITY	string
untypedAtomic	string
dateTime	Altova.Types.DateTime
date	Altova.Types.DateTime
time	Altova.Types.DateTime
gYear	Altova.Types.DateTime
gYearMonth	Altova.Types.DateTime
gMonth	Altova.Types.DateTime
gMonthDay	Altova.Types.DateTime
gDay	Altova.Types.DateTime
duration	Altova.Types.Duration
base64Binary	byte[]
hexBinary	byte[]
anyURI	string
QName	Altova.Types.QName
NOTATION	string
double	double
float	double
decimal	decimal
integer	decimal
nonPositiveInteger	decimal
negativeInteger	decimal

Schema Type	C# Type
long	long
int	int
short	int
byte	int
nonNegativeInteger	decimal
positiveInteger	decimal
unsignedLong	ulong
unsignedInt	ulong
unsignedShort	ulong
unsignedByte	ulong
dayTimeDuration	Altova.Types.Duration
yearMonthDuration	Altova.Types.Duration
NMTOKENS	string
IDREFS	string
ENTITIES	string

8.6.4.3 Create a C++ library

How to write a C++ library:

Create the **h** and **cpp** files using the exact name, at the same location you defined in the implementation element, for the whole library.

Header file:

1. Write "using namespace altova;"
2. Add the namespace you specified in the implementation element.
3. Add the class you specified in the implementation element of the mff, with the static functions you specified in the mff file.
4. Please remember to write "ALTOVA_DECLSPECIFIER" in front of the class name, this ensures that your classes will compile correctly - whether you use dynamic or static linkage in subsequent generated code.
5. The exact mapping of XML Schema datatypes to C++ datatypes can be found in the table below.

The resulting **header file** should look like this:

```
#ifndef HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED
#define HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED

#if _MSC_VER > 1000
```

```
	#pragma once
#endif // _MSC_VER > 1000

using namespace altova;

namespace helloworld {

class ALTOVA_DECLSPECIFIER Greetings
{
public:
	static string_type   HelloFunctionResponse(bool greetingType);
};

} // namespace HelloWorldLibrary

#endif // HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED
```

In the **cpp** file:

1. The first lines need to be the includes for **StdAfx.h** and the definitions from the **Altova** base library, please copy these lines from the sample code supplied below.
2. The **../Altova** path is correct for your source files, because they will be copied to a separate project in the resulting code that will be found at targetdir/libraryname.
3. The next line is the include for your header file you created above.
4. Add the implementations for your functions.
5. Please remember that the implementations need to be in the correct namespace you specified in the header file and in the implementations element of the mff.

The sample cpp file would look like this:

```
#include "StdAfx.h"
#include "../Altova/Altova.h"
#include "../Altova/AltovaException.h"
#include "../Altova/SchemaTypes.h"

#include "Greetings.h"

namespace helloworld {

	string_type   Greetings::HelloFunctionResponse(bool greetingType)
	{
		if( greetingType )
			return _T("Hello World!");
		return _T("Hello User!");
	}

}
```

In contrast to Java or C#, you do not need to compile your source files. They will be copied to the generated code, and are compiled with the rest of the generated mapping code.

C++ compile errors:

If you get a compiler error at the line shown below, add the path to the msado15.DLL

```
	#import "msado15.dll" rename("EOF", "EndOfFile")
```


You have to add the path where the msado15.dll is stored into the directories section of your Visual Studio environment:

1. In VS select from the menu: Tools / Options...
2. Select the "Directories" tab.
3. Select "Include files" in the pull-down "Show directories for"
4. Add a new line with the path to the file;
   for English systems usually "C:\Program Files\Common Files\System\ADO"
5. Rebuild, then everything should be fine.

Datatype Mapping

Schema Type	C++ Type
anySimpleType	string_type
anyAtomicType	string_type
boolean	bool
string	string_type
normalizedString	string_type
token	string_type
language	string_type
NMTOKEN	string_type
Name	string_type
NCName	string_type
ID	string_type
IDREF	string_type
ENTITY	string_type
untypedAtomic	string_type
dateTime	altova::DateTime
date	altova::DateTime
time	altova::DateTime
gYear	altova::DateTime
gYearMonth	altova::DateTime
gMonth	altova::DateTime
gMonthDay	altova::DateTime
gDay	altova::DateTime

Schema Type	C++ Type
duration	altova::Duration
base64Binary	altova::mapforce::blob
hexBinary	altova::mapforce::blob
anyURI	string_type
QName	altova::QName
NOTATION	string_type
double	double
float	double
decimal	double
integer	__int64
nonPositiveInteger	__int64
negativeInteger	__int64
long	__int64
int	int
short	int
byte	int
nonNegativeInteger	unsigned __int64
positiveInteger	unsigned __int64
unsignedLong	unsigned __int64
unsignedInt	unsigned __int64
unsignedShort	unsigned __int64
unsignedByte	unsigned __int64
dayTimeDuration	altova::Duration
yearMonthDuration	altova::Duration
NMTOKENS	string_type
IDREFS	string_type
ENTITIES	string_type

8.7 Java and .NET functions - specifics

You can add the following library file types to MapForce:

- Java .class files (.jar files are not supported)
- .NET .dll assembly files

Note: All functions called from a MapForce mapping should be "**idempotent**" (this means that they should return the same value each time the function is called with the same input parameters). The exact order and the number of times a function is called by MapForce is undefined and may change any time.

Java function dependencies

The imported class files and their packages do not need to be added to the CLASSPATH variable, since the Built-in execution engine, as well as generated Java code, will automatically add imported packages to the Java engine's classpath or to Ant, respectively. However, any dependencies of the imported class files and packages will not be handled automatically. Therefore, if imported Java class files or packages depend on other class files, be sure to add the parent directories of all dependent packages to the CLASSPATH environment variable before starting MapForce.

Java function support

Top-level classes, static member classes and non-static member classes are supported:

- `new <classname>(<arg1>, <arg2>, ...)`
- `<object>.new <member-class>(<arg1>, <arg2>, ...)`

Member functions and static functions are supported:

- `<function>(<arg1>, <arg2>, ...)`
- `<object>.<method>(<arg1>, ...)`

Supported connections between XML Schema and Java types:

Schema type	Java type
`xs:string`	`String`
`xs:byte`	`byte`
`xs:short`	`short`
`xs:int`	`int`
`xs:long`	`long`
`xs:boolean`	`boolean`
`xs:float`	`float`

Schema type	Java type
xs:double	double
xs:decimal	java.math.BigDecimal
xs:integer	java.math.BigInteger

Connections in both directions are possible. Other Java types are not supported.

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
xs:float	System.Single	float
xs:double	System.Double	double
xs:decimal	System.Decimal	decimal

Connections in both directions are possible. Other .NET/C# types (including array types) are not

supported. Methods using such parameters or return values will be ignored.

Object types are supported by calling their constructor, or as a return value of a method. They can be mapped to other .NET methods. Manipulating the object using MapForce means is not possible.

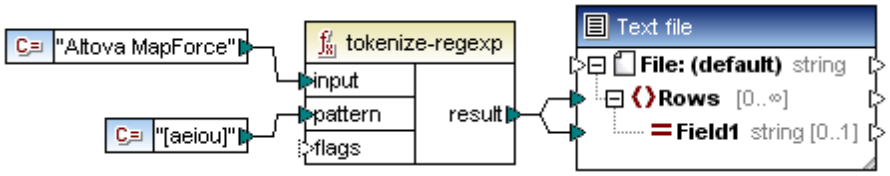
8.8 Regular Expressions

MapForce can use regular expressions in the **pattern** parameter of the **match-pattern** and **tokenize-regexp** functions, to find specific strings.

The regular expression syntax and semantics for XSLT and XQuery are identical to those defined in http://www.w3.org/TR/xmlschema-2/. Please note that there are slight differences in regular expression syntax between the various programming languages.

Terminology

input	the string that the regex works on
pattern	the regular expression
flags	optional parameter to define how the regular expression is to be interpreted
result	the result of the function



Tokenize-regexp returns a sequence of strings. The connection to the Rows item creates one row per item in the sequence.

regex syntax

Literals e.g. a single character:
e.g. The letter "a" is the most basic regex. It matches the first occurrence of the character "a" in the string.

Character classes []

This is a set of characters enclosed in square brackets.

One, and only one, of the characters in the square brackets are matched.

pattern **[aeiou]**
Matches a lowercase vowel.

pattern **[mj]ust**
Matches must or just

Please note that "pattern" is case sensitive, a lower case **a** does not match the uppercase **A.**

Character ranges **[a-z]**
Creates a range between the two characters. Only one of the characters will be matched at one time.

pattern **[a-z]**
Matches any lowercase characters between a and z.

negated classes **[^]**
using the caret as the first character after the opening bracket, negates the character class.

pattern **[^a-z]**
Matches any character not in the character class, including newlines.

Meta characters "."

Dot meta character
matches **any single** character (except for newline)

pattern **.**
Matches any single character.

Quantifiers ? + * {}

Quantifiers define how often a regex component must repeat within the input string, for a match to occur.

?	
zero or one	preceding string/chunk is optional
+	
one or more	preceding string/chunks may match one or more times
*****	
zero or more	preceding string/chunks may match zero or more times
{}	
min / max repetitions	no. of repetitions a string/chunks has to match e.g. mo{1,3} matches mo, moo, mooo.

()
subpatterns
parentheses are used to group parts of a regex together.

|
Alternation/or allows the testing of subexpressions form left to right.
(horse|make) sense - will match "horse sense" or "make sense"

flags

These are optional parameters that define how the regular expression is to be interpreted. Individual letters are used to set the options, i.e. the character is present. Letters may be in any order and can be repeated.

s

If present, the matching process will operate in the "dot-all" mode.

The meta character "**.**" matches any character whatsoever. If the input string contains "hello" and "world" on two *different* lines, the regular expression "hello*world" will only match if the **s** flag/ character is set.

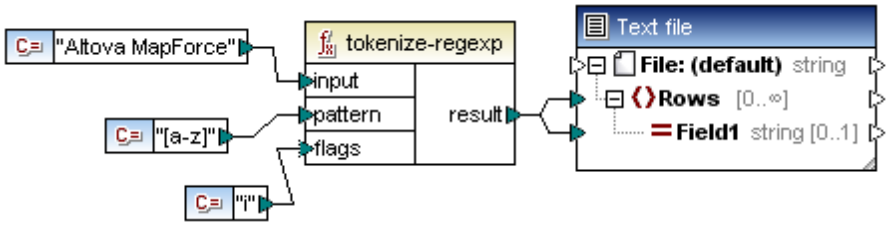
m
If present, the matching process operates in multi-line mode.

In multi-line mode the caret **^** matches the start of **any** line, i.e. the start of the entire string **and** the first character after a newline character.

The dollar character **$** matches the end of **any** line, i.e. the end of the entire string and the character immediately before a newline character.

Newline is the character #x0A.

i
If present, the matching process operates in case-insensitve mode.
The regular expression [a-z] plus the **i** flag would then match all letters a-z and A-Z.



x
If present, whitespace characters are removed from the regular expression prior to the matching process. Whitespace chars. are #x09, #x0A, #x0D and #x20.

Exception:
Whitespace characters within character class expressions are not removed e.g. [#x20].

Please note:
When generating code, the advanced features of the regex syntax might differ slightly between the various languages, please see the specific regex documentation for your language.

8.9 Function Library Reference

This reference chapter describes the MapForce built-in functions available in the Libraries pane, organized by library.

The availability of function libraries in the Libraries pane depends on the transformation language you have selected (see Selecting a transformation language). The **core** library is a collection of functions available in C++, C#, Java languages and in at least one of the following: XQuery, XPath, or XSLT. The **lang** library is dedicated to functions available in C++, C#, and Java languages. Other libraries contain functions associated with each separate type of output.

XPath 2.0 restrictions: Several XPath 2.0 functions dealing with sequences are currently not available.

8.9.1 core | aggregate functions

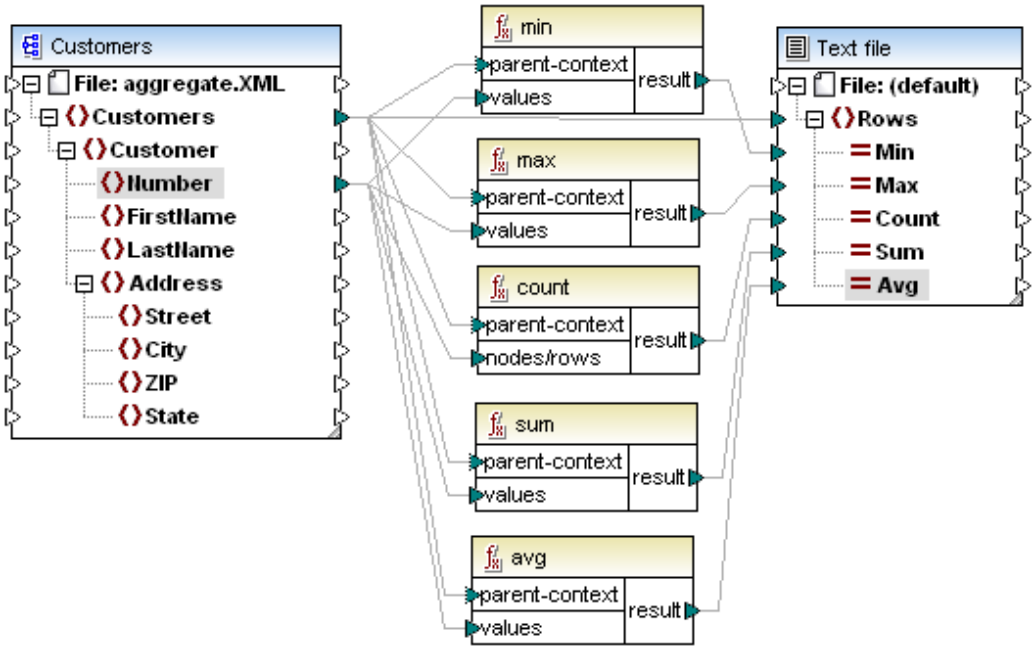
Aggregate functions perform operations on a set, or sequence, of input values. The input data for min, max, sum and avg is converted to the **decimal** data type for processing.

- The input values must be connected to the **values** parameter of the function.
- A context node (item) can be connected to the **parent-context** parameter to override the default context from which the input sequence is taken. The parent-context parameter is optional.
- The **result** of the function is connected to the specific target item.

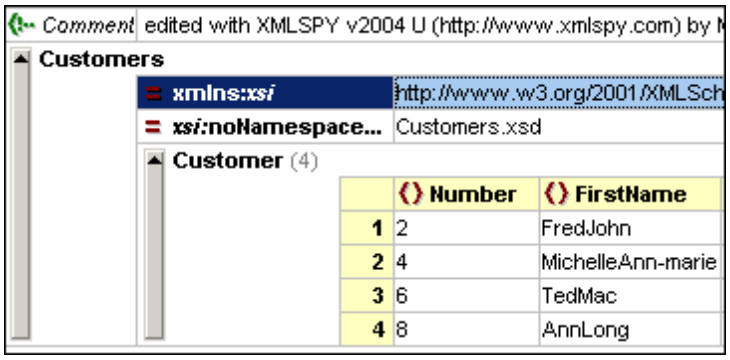
The mapping shown below is available as **Aggregates.mfd** in the ...\Tutorial folder and shows how these functions are used.

Aggregate functions have two input items.

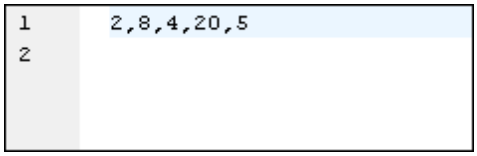
- **values** (nodes/rows) is connected to the source item that provides the data, in this case Number.
- **parent-context** is connected to the item you want to iterate over, i.e. the context, in this case over all Customers. The parameter is, however, optional.



The input instance in this case is an XML file containing the following data:



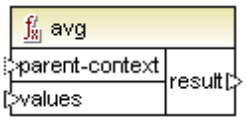
- The source data supplied to the values item is the number sequence 2,4,6,8.
- The output component in this case is a simple text file.
  Clicking the Output tab for the above mapping delivers the following result:



min=2, max=8, count=4, sum=20 and avg=5.

8.9.1.1 avg

Returns the average value of all values within the input sequence. The average of an empty set is an empty set. Not available in XSLT1.

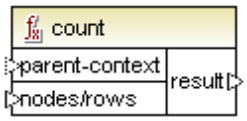


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2016\MapForceExamples** directory.

8.9.1.2 count

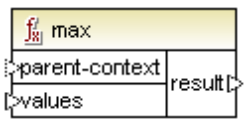
Returns the number of individual items making up the input sequence. The count of an empty set is zero. Limited functionality in XSLT1.



Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
nodes/rows	This argument must be connected to the source item to be counted.

8.9.1.3 max

Returns the maximum value of all numeric values in the input sequence. Note that this function returns an empty set if the **strings** argument is an empty set. Not available in XSLT1.



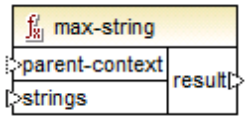
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.

Argument	Description
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the maximum from a sequence of strings, use the **max-string** function.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2016\MapForceExamples** directory.

8.9.1.4 max-string

Returns the maximum value of all string values in the input sequence. For example, `max-string("a", "b", "c")` returns "c". This function is not available in XSLT1.

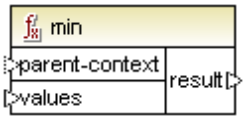


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

Note that the function returns an empty set if the **strings** argument is an empty set.

8.9.1.5 min

Returns the minumum value of all numeric values in the input sequence. The minimum of an empty set is an empty set. Not available in XSLT1.

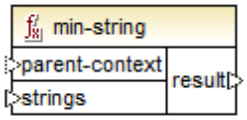


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the minimum from a sequence of strings, use the **min-string** function.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2016\MapForceExamples** directory.

8.9.1.6 min-string

Returns the minimum value of all string values in the input sequence. For example, `min-string("a", "b", "c")` returns "a". This function is not available in XSLT1.

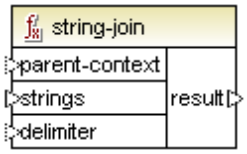


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

Note that the function returns an empty set if the **strings** argument is an empty set.

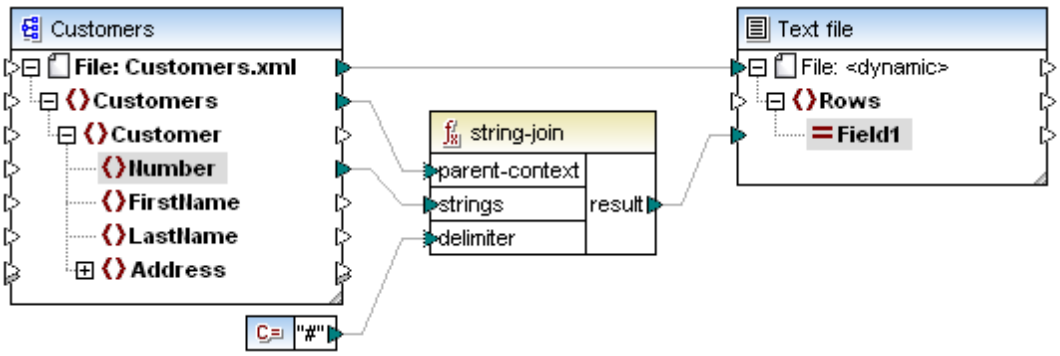
8.9.1.7 string-join

Concatenates all the values of the input sequence into one string delimited by whatever string you choose to use as the delimiter. The string-join of an empty set is the empty string. Not available in XSLT1.



The example below contains four separate customer numbers 2 4 6 and 8. The constant character supplies a hash character "#" as the delimiter.

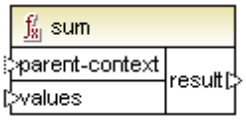
Result = 2#4#6#8



If you do not supply a delimiter, then the default is an empty string, i.e. no delimiter of any sort. Result = 2468.

8.9.1.8 sum

Returns the arithmetic sum of all values in the input sequence. The sum of an empty set is zero. Not available in XSLT1.

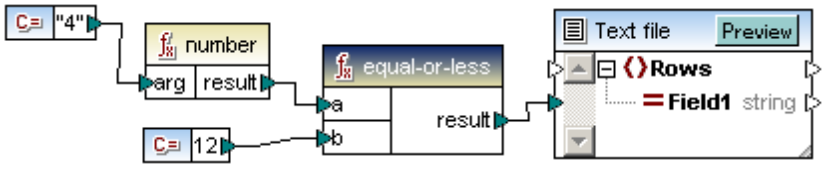


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

8.9.2 core | conversion functions

To support explicit data type conversion, several type conversion functions are available in the **conversion** library. Note that, in most cases, MapForce creates necessary conversions automatically and these functions need to be used only in special cases.

If the input nodes are of differing types, e. g. integer and string, you can use the conversion functions to force a string or numeric comparison.



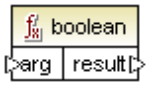
In the example above the first constant is of type string and contains the string "4".
The second constant contains the numeric constant 12. To be able to compare the two values explicitly the types must agree.

Adding a **number** function to the first constant converts the string constant to the numeric value of 4. The result of the comparisons is then "true".

Note that if the number function were not be used, i.e 4 would be connected directly to the **a** parameter, a string compare would occur, with the result being false.

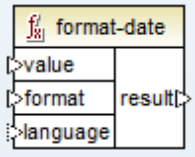
8.9.2.1 boolean

Converts an input numeric value into a boolean (as well as a string to numeric - true to 1). E.g. 0 to "false", or 1 to "true", for further use with logical functions (equal, greater etc.) filters, or if-else functions.



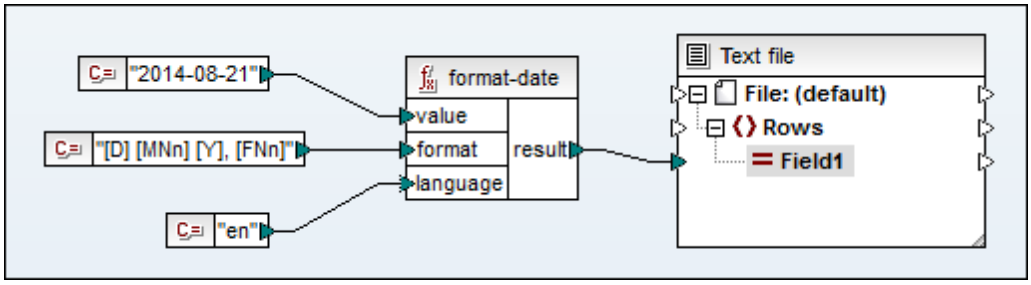
8.9.2.2 format-date

Converts an `xs:date` input value into a string and formats it according to specified options.



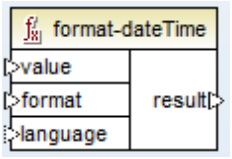
Argument	Description
value	The date to be formatted.
format	A format string identifying the way in which the date is to be formatted. This argument is used in the same way as the `format` argument in the **`format-dateTime`** function.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values: **en (default)** English **es** Spanish **de** German **ja** Japanese

In the following example, the output result is: "21 August 2014, Thursday". To translate this value to Spanish, set the value of the language argument to `es`.



8.9.2.3 format-dateTime

Converts a dateTime into a string.



Argument	Description
value	The `xs:dateTime` value to be formatted.
format	A format string identifying the way in which **value** is to be formatted.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values: **en (default)** English **es** Spanish **de** German **ja** Japanese

Note:
If the function's output (i.e. result) is connected to a node of type other than string, the formatting may be lost as the value is cast to the target type. This automatic cast can be disabled by unchecking the **Cast target values to target types** check box in the Component Settings of the target component.

The format argument consists of a string containing so-called variable markers enclosed in square brackets. Characters outside the square brackets are literal characters to be copied into the result. If square brackets are needed as literal characters in the result, then they should be doubled.

Each variable marker consists of a component specifier identifying which component of the date or time is to be displayed, an optional formatting modifier, another optional presentation modifier and an optional width modifier, preceded by a comma if it is present.


```
format := (literal | argument)*
argument := [component(format)?(presentation)?(width)?]
width := , min-width ("-" max-width)?
```

The components are:

Specifier	Description	Default Presentation
Y	year (absolute value)	four digits (2010)
M	month of the year	1-12
D	day of month	1-31
d	day of year	1-366
F	day of week	name of the day (language dependent)
W	week of the year	1-53
w	week of month	1-5
H	hour (24 hours)	0-23
h	hour (12 hour)	1-12
P	A.M. or P.M.	alphabetic (language dependent)
m	minutes in hour	00-59
s	seconds in minute	00-59
f	fractional seconds	numeric, one decimal place
Z	timezone as a time offset from UTC	+08:00
z	timezone as a time offset using GMT	GMT+n

The formatting modifier:

Character	Description	Example
1	decimal numeric format with no leading zeros: 1, 2, 3, ...	1, 2, 3
01	decimal format, two digits: 01, 02, 03, ...	01, 02, 03
N	name of component, upper case	MONDAY, TUESDAY [1)]
n	name of component, lower case	monday, tuesday [1)]
Nn	name of component, title case	Monday, Tuesday [1)]

Note: N, n, and Nn modifiers only support the following components: M, d, D.

The width modifier, if present, is introduced by a comma. It takes the form:

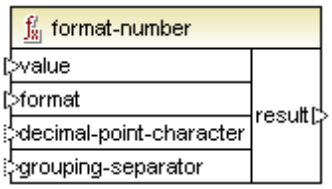
```
, min-width ("-" max-width)?
```

Supported examples

DateTime	format String	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]`	2003-11-03
2003-11-03T00:00:00	`[Y]-[M,2]-[D,2] [H,2]:[m]:[s]`	2003-11-03 00:00:00
2010-06-02T08:02	`[Y] [MNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f]`	2010 June 02 Wed 153 8:02:12.054
2010-06-02T08:02	`[Y] [MNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f] [z]`	2010 June 02 Wed 153 8:02:12.054 GMT+02:00
2010-06-02T08:02	`[Y] [MNn] [D1] [F] [H]:[m]:[s].[f] [Z]`	2010 June 2 Wednesday 8:02:12.054 +02:00
2010-06-02T08:02	`[Y] [MNn] [D] [F,3-3] [H01]:[m]:[s]`	2010 June 2 Wed 08:02:12

8.9.2.4 format-number

Converts a number into a string. The function is available for XSLT 1.0, XSLT 2.0, Java, C#, C++ and Built-in execution engine.



Argument	Description
`value`	Mandatory argument. Supplies the number to be formatted.
`format`	Mandatory argument. Supplies a format string that identifies the way in which the number is to be formatted. This argument is used in the same way as the `format` argument in the **`format-dateTime`** function.
`decimal-point-format`	Optional argument. Supplies the character to be used as the

Argument	Description
	decimal point character. The default value is the full stop (.) character.
grouping-separator	Optional argument. Supplies the character used to separate groups of numbers. The default value is the comma (,) character.

Note: If the function's output (i.e. result) is connected to a node of type other than string, the formatting may be lost as the value is cast to the target type. This automatic cast can be disabled by unchecking the **Cast target values to target types** check box in the component settings of the target component.

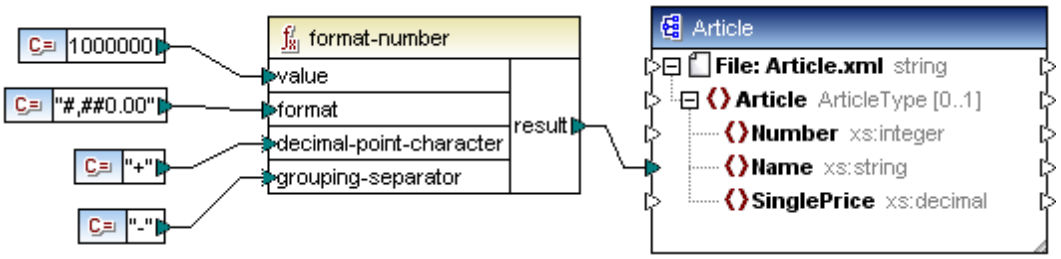
Format:

```
format := subformat (;subformat)?
  subformat := (prefix)? integer (.fraction)? (suffix)?
  prefix := any characters except special characters
  suffix := any characters except special characters
  integer := (#)* (0)* ( allowing ',' to appear)
  fraction := (0)* (#)* (allowing ',' to appear)
```

The first *subformat* is used for formatting positive numbers, and the second subformat for negative numbers. If only one *subformat* is specified, then the same subformat will be used for negative numbers, but with a minus sign added before the *prefix*.

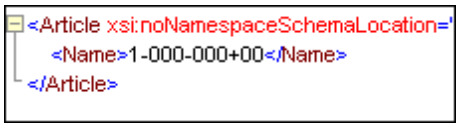
Special Character	default	Description
zero-digit	0	A digit will always appear at this point in the result
digit	#	A digit will appear at this point in the result string unless it is a redundant leading or trailing zero
decimal-point	.	Separates the integer and the fraction part of the number.
grouping-seperator	,	Seperates groups of digits.
percent-sign	%	Multiplies the number by 100 and shows it as a percentage.
per-mille	‰	Multiplies the number by 1000 and shows it as per-mille.

The characters used for decimal-point-character and grouping-separator are always "." and "," respectively. They can, however, be changed in the formatted output, by mapping constants to these nodes.



The result of the format number function shown above.

- The decimal-point character was changed to a "+".
- The grouping separator was changed to a "-"



Rounding

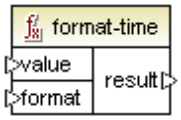
The rounding method used for this function is "half up", e.g. the value gets rounded up if the fraction is greater than or equal to 0.5. The value gets rounded down if the fraction is less than 0.5. This method of rounding only applies to generated code and the built-in execution engine.

In XSLT 1.0, the rounding mode is undefined. In XSLT 2.0, the rounding mode is "round-half-to-even".

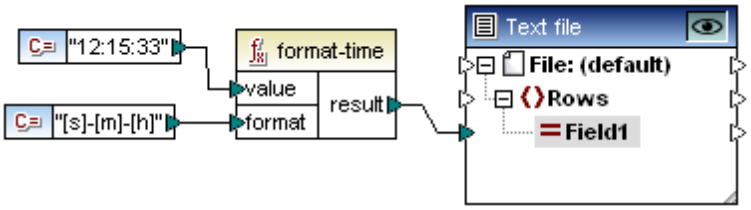
Number	Format String	Result
1234.5	#,##0.00	1,234.50
123.456	#,##0.00	123.46
1000000	#,##0.00	1,000,000.00
-59	#,##0.00	-59.00
1234	###0.0###	1234.0
1234.5	###0.0###	1234.5
.00025	###0.0###	0.0003
.00035	###0.0###	0.0004
0.25	#00%	25%
0.736	#00%	74%
1	#00%	100%
-42	#00%	-4200%
-3.12	#.00;(#.00)	(3.12)
-3.12	#.00;#.00CR	3.12CR

8.9.2.5 format-time

Converts an xs:time input value into a string. The `format` argument is used in the same way as the `format` argument in the **`format-dateTime`** function.



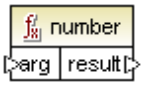
E.g



Result: 33-15-12

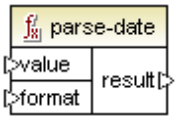
8.9.2.6 number

Converts an input string into a number. Also converts a boolean input to a number.



8.9.2.7 parse-date

Available for Java, C#, C++, and the Built-in execution engine.

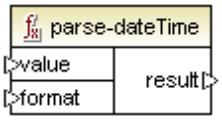


Converts a string into a date, while ignoring the time component. This function uses the parse-dateTime function as a basis, while ignoring the time component. The result is of type xs:date.

For an example of usage, see Example: Mapping Data from an RSS Feed.

8.9.2.8 parse-dateTime

Available for Java, C#, C++, and the Built-in execution engine.



Converts a string into an dateTime.

Argument	Description
value	The string containing the date/time
format	The picture string which defines how value is currently formatted

The components are:

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
H	hour (24 hours)	0-23
h	hour (12 hour)	1-12
P	A.M. or P.M.	alphabetic (language dependent)
m	minutes in hour	00-59
s	seconds in minute	00-59
f	fractional seconds	numeric, one decimal place
Z	timezone as a time offset from UTC	+08:00
z	timezone as a time offset using GMT	GMT+n

The formatting modifier:

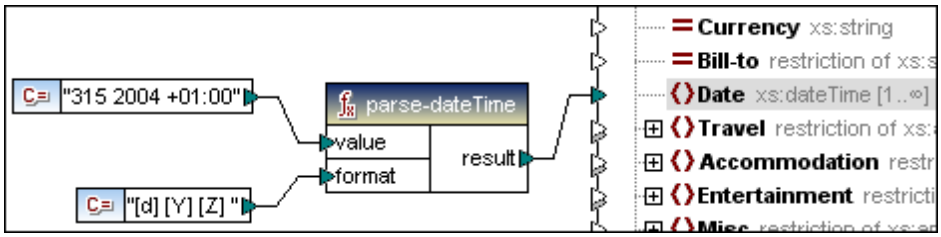
Character	Description	Example
1	decimal numeric format with no leading zeros: 1, 2, 3, ...	1, 2, 3
01	decimal format, two digits: 01, 02, 03, ...	01, 02, 03
N	name of component, upper case	MONDAY, TUESDAY
n	name of component, lower case	monday, tuesday
Nn	name of component, title case	Monday, Tuesday

1) **N, n, and Nn modifiers** only support the component M (month).

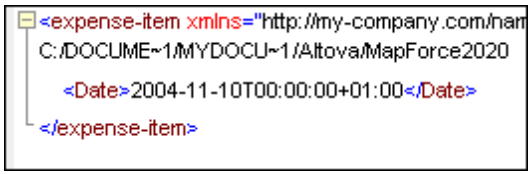
Examples:

Date String	Picture String	Result
21-03-2002 16:21:12.492 GMT +02:00	[D]-[M]-[Y] [H]:[m]:[s].[f] [z]	2002-03-21T16:21:12.492+02:00
315 2004 +01:00	[d] [Y] [Z]	2004-11-10T00:00:00+01:00
1.December.10 03:2:39 p.m. +01:00	[D].[MNn].[Y,2-2] [h]:[m]:[s] [P] [Z]	2010-12-01T15:02:39+01:00
20110620	[Y,4-4][M,2-2][D,2-2]	2011-06-20T00:00:00

Example in MapForce:

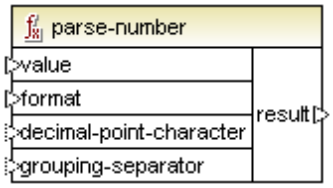


Result:



8.9.2.9 *parse-number*

Available for Java, C#, C++, and the Built-in execution engine.



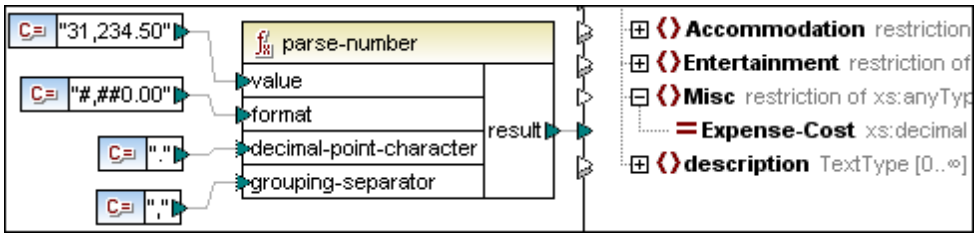
Converts an input string into a decimal number.

Argument	Description
value	The string to be parsed/converted to a number
format	A format string that identifies the way in which the number is currently formatted (optional). Default is "#,##0.#"

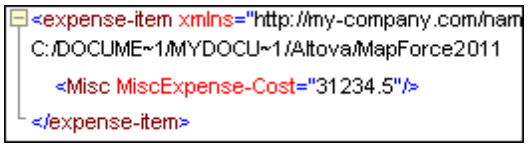
decimal-point-character	The character to be used as the decimal point character. Default is the '**.**' character (optional)
grouping-separator	The separator/delimiter used to separate groups of numbers. Default is the "**,**" character (optional)

The ***format*** string used in parse-number is the same as that used in format-number.

Example in MapForce:

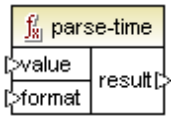


Result:



8.9.2.10 parse-time

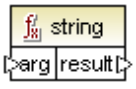
Available for Java, C#, C++, and the Built-in execution engine.



Converts a string into a time, while ignoring the date component. This function uses the parse-dateTime function as a basis, while ignoring the date component. The result is of type xs:time.

8.9.2.11 string

Converts an input value into a string. The function can also be used to retrieve the text content of a node.



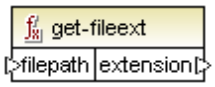
If the input node is a XML complex type, then all descendents are also output as a single string.

8.9.3 core | file path functions

The **file path** functions allow you to directly access and manipulate file path data, i.e. folders, file names, and extensions for further processing in your mappings. They can be used in all languages supported by MapForce.

8.9.3.1 get-fileext

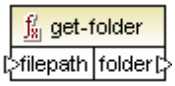
Returns the extension of the file path including the dot "." character.



E.g. 'c:\data\Sample.mfd' returns '.mfd'

8.9.3.2 get-folder

Returns the folder name of the file path including the trailing slash, or backslash character.



E.g. 'c:/data/Sample.mfd' returns 'c:/data/'

8.9.3.3 main-mfd-filepath

Returns the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.

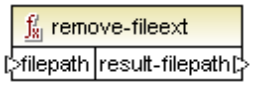
8.9.3.4 mfd-filepath

If the function is called in the main mapping, it returns the same as main-mfd-filepath function, i.e. the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.

If called within an **user-defined function** which is **imported** by a mfd-file, it returns the full path of the imported mfd file which contains the **definition** of the user-defined function.

8.9.3.5 remove-fileext

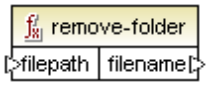
Removes the extension of the file path including the dot-character.



E.g. 'c:/data/Sample.mfd' returns 'c:/data/Sample'.

8.9.3.6 remove-folder

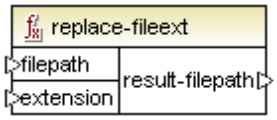
Removes the directory of the file path including the trailing slash, or backslash character.



E.g. 'c:/data/Sample.mfd' returns 'Sample.mfd'.

8.9.3.7 replace-fileext

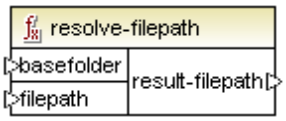
Replaces the extension of the file path supplied by the filepath parameter, with the one supplied by the connection to the extension parameter.



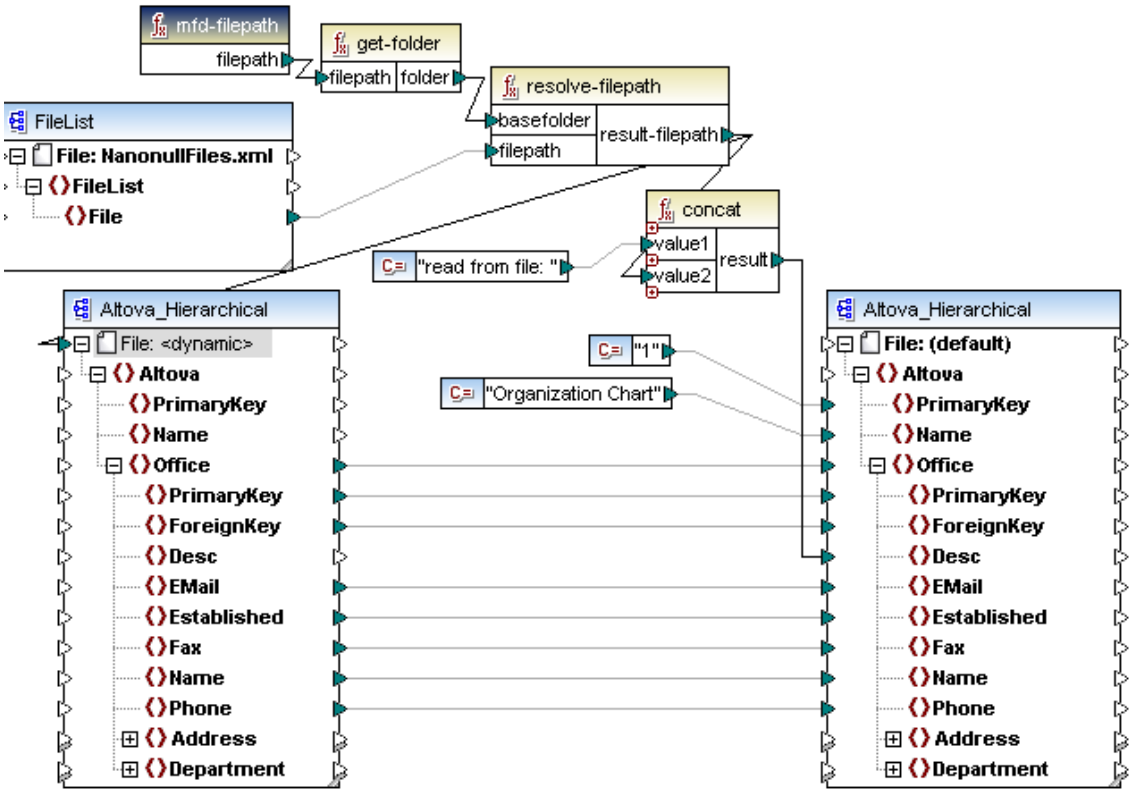
E.g. c:/data/Sample.**mfd'** as the input filepath, and '.**mfp'** as the extension, returns 'c:/data/Sample.mfp'

8.9.3.8 resolve-filepath

Resolves a relative file path to a relative, or absolute, base folder. The function supports '.' (current directory) and '..' (parent directory).



For an example, see the mapping **MergeMultipleFiles_List.mfd** available in the ... \MapForceExamples folder.

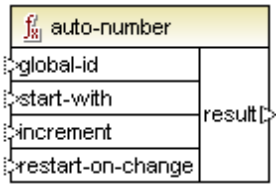


8.9.4 core | generator functions

The **core / generator** functions library includes functions which generate values.

8.9.4.1 auto-number

The **auto-number** function generates integers in target nodes of a component, depending on the various parameters you define. The function result is a value starting at **start_with** and increased by **increment**. Default values are: start-with=1 and increase=1. Both parameters can be negative.



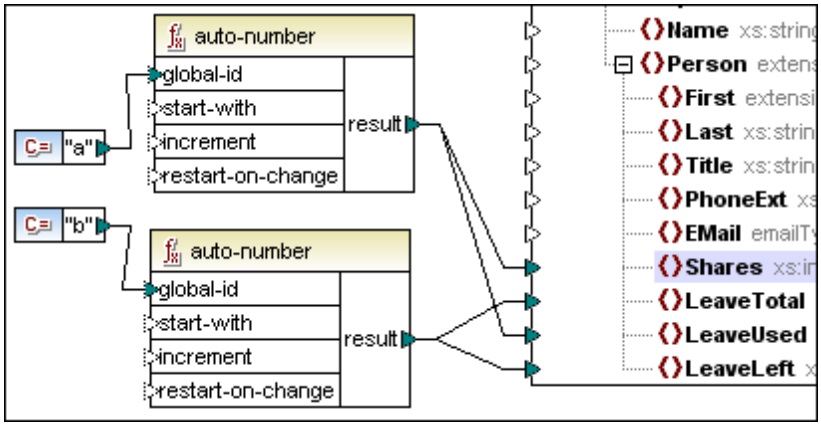
Make sure that the result connector (of the auto-number function) is **directly** connnected to a target node. The exact order in which functions are called by the generated mapping code is undefined. MapForce may choose to cache calculated results for reuse, or evaluate expressions in any order. It is therefore strongly recommended to take care when using the auto-number function.

global-id

This parameter allows you to synchronize the number sequence output of two separate auto-number functions connected to a single target component.

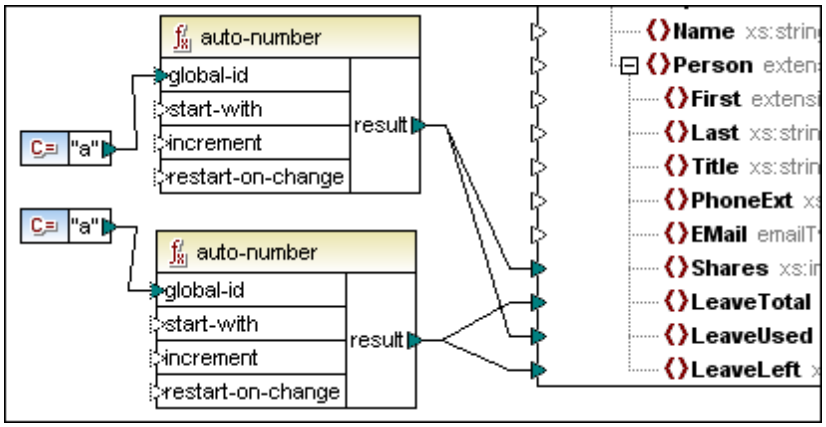
If the two auto-number functions do **not** have the same global-id, then each increments the target items separately. In the example below, each function has a different global-id i.e. a and b.

The output of the mapping is 1,1,2,2. The top function supplies the first 1 and the lower one the second 1.



If both functions have **identical** global-ids, **a** in this case, then each function "knows" about the current auto-number state (or actual value) of the other, and both numbers are then synchronised/ in sequence.

The output of the mapping is therefore 1, 2, 3, 4.The top function supplies the first 1 and the lower one now supplies a 2.

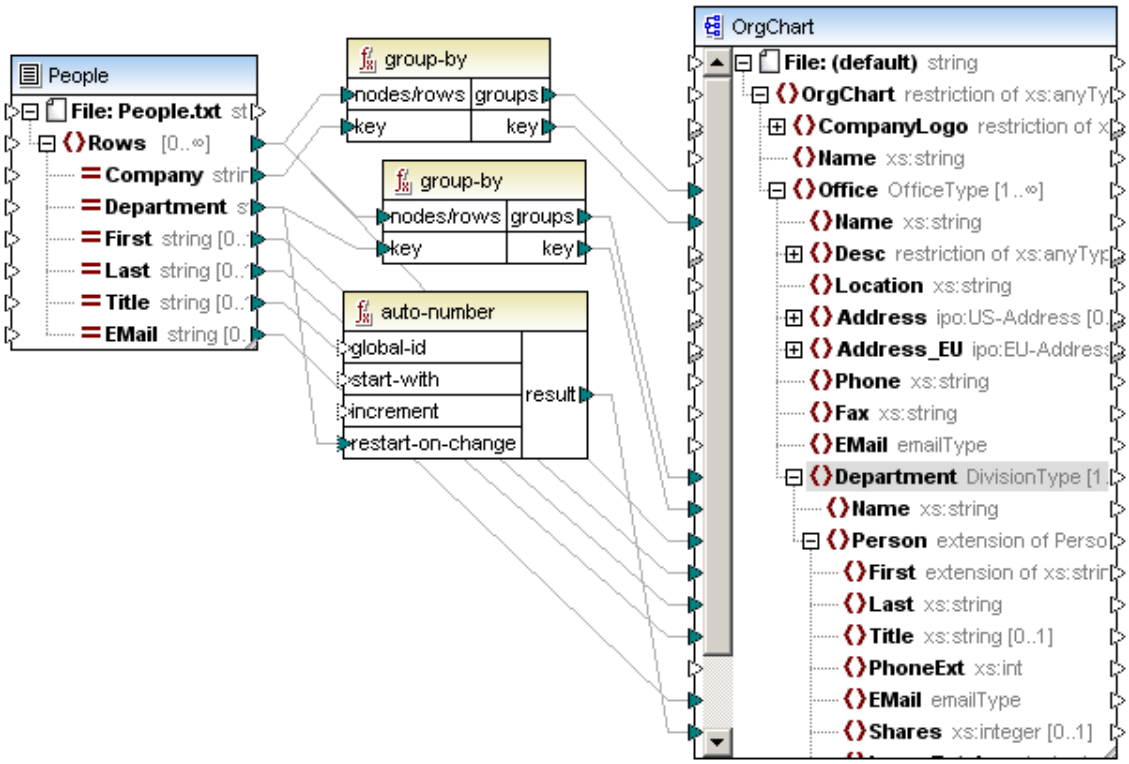


start-with
The inital value used to start the auto numbering sequence. Default is 1.

increment
The increment you want auto-number sequence to increase by. Default is 1.

restart on change
Resets the auto-number counter to "start-with", when the **content** of the connected item changes.

In the example below, start-with and increment are both using the default 1. As soon as the **content** of Department changes, i.e. the department name changes, the counter is reset and starts at 1 for each new department.



8.9.5 core | logical functions

Logical functions are (generally) used to compare input data with the result being a boolean "true" or " false". They are generally used to test data before passing on a subset to the target component using a filter.

input parameters = **a** | **b**, or **value1** | **value2**
output parameter = result

The evaluation result of two input nodes depends on the input values as well as the data types used for the comparison.

For example, the 'less than' comparison of the integer values 4 and 12 yields the boolean value "true", since 4 is less than 12. If the two input strings contain '4' and '12', the lexical analysis results in the output value "false", since '4' is alphabetically greater than the first character '1' of the second operand (12).

If all input data types are of the same type, e.g. all input nodes are numerical types, or strings, then the comparison is done for the common type.

If the input nodes are of differing types (for example, integer and string, or string and date), then the data type used for the comparison **is the most general (least restrictive) input data type** of the two input types.

Before comparing two values, all input values are converted to a common datatype. Using the previous example; the datatype "string" is less restrictive than "integer". Comparing integer value 4 with the string '12', converts integer value 4 to the string '4', which is then compared with the string '12'.

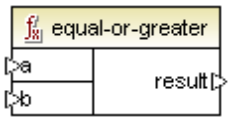
Note: Logical functions cannot be used to test the existence of null values. If you supply a null value as argument to a logical function, it returns a null value. For more information about handling null values, see Nil Values / Nillable.

8.9.5.1 equal

Result is true if a=b, else false.

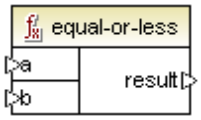
8.9.5.2 equal-or-greater

Result is true if a is equal/greater than b, else false.



8.9.5.3 equal-or-less

Result is true if a is equal/less than b, else false.



8.9.5.4 greater

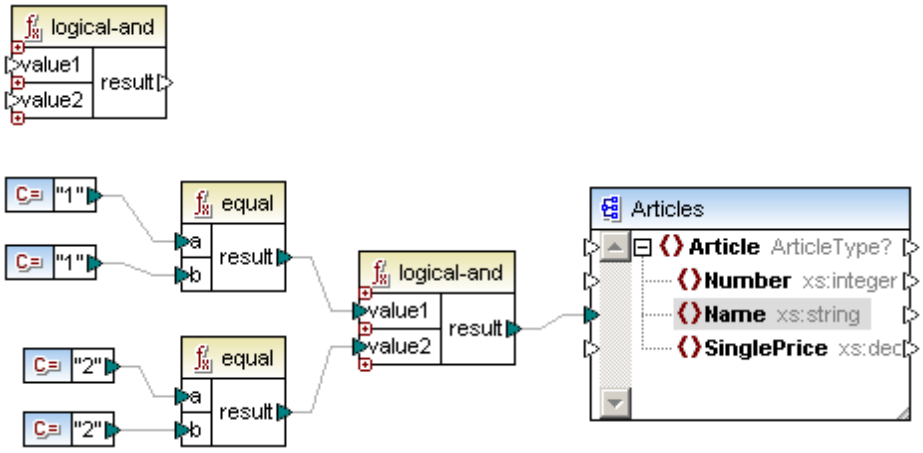
Result is true if a is greater than b, else false.

8.9.5.5 less

Result is true if a is less than b, else false.

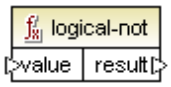
8.9.5.6 logical-and

If **both** value1 and value2 of the logical-and function are **true**, then result is true; if different then false.

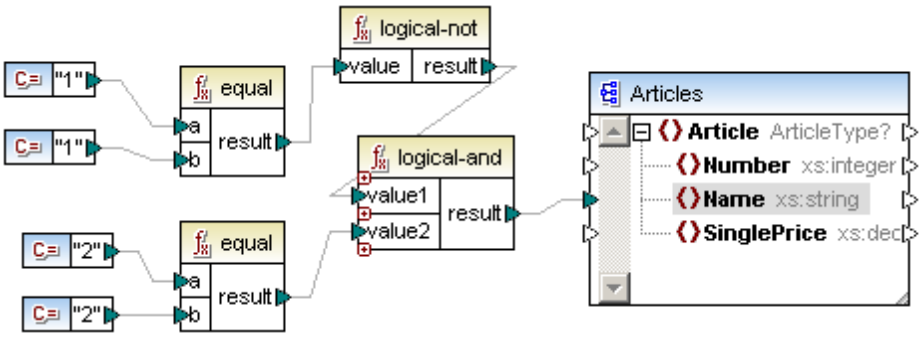


8.9.5.7 logical-not

Inverts or flips the logical state/result; if input is true, result of logical-not function is false. If input is false then result is true.

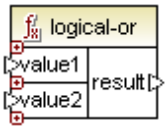


The logical-not function shown below, inverts the result of the equal function. The logical-and function now only returns true if boolean values of value1 and value2 are different, i.e. true-false, or false-true.



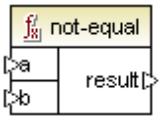
8.9.5.8 *logical-or*

Requires both input values to be boolean. If **either** value1 or value2 of the logical-or function are **true**, then the result is true. If both values are false, then result is false.



8.9.5.9 *not-equal*

Result is true if a is not equal to b.

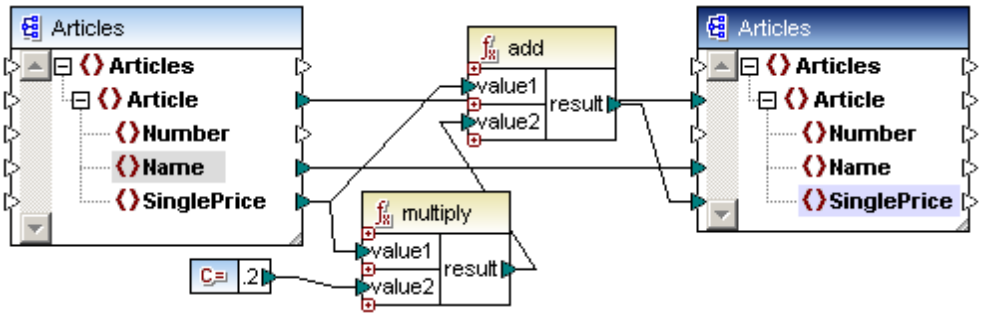


8.9.6 core | math functions

Math functions are used to perform basic mathematical functions on data. Note that they cannot be used to perform computations on durations, or datetimes.

input parameters = **value1** | **value2**
output parameter = **result**

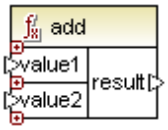
input values are automatically converted to decimal for further processing.



The example shown above, adds 20% sales tax to each of the articles mapped to the target component.

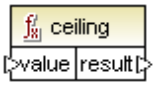
8.9.6.1 add

Result is the decimal value of adding **value1** to **value2**.



8.9.6.2 ceiling

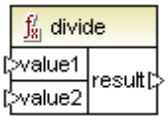
Result is the smallest integer that is greater than or equal to **value**, i.e. the next highest integer value of the decimal input **value**.



E.g. if the result of a division function is 11.2, then applying the ceiling function to it makes the result 12, i.e. the next highest whole number.

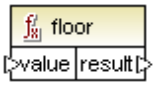
8.9.6.3 divide

Result is the decimal value of dividing **value1** by **value2**. The result precision depends on the target language. Use the round-precision function to define the precision of result.



8.9.6.4 floor

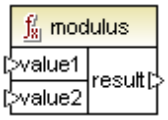
Result is the largest integer that is less than or equal to **value**, i.e. the next lowest integer value of the decimal input **value**.



E.g. if the result of a division function is 11.2, then applying the floor function to it makes the result 11, i.e. the next lowest whole number.

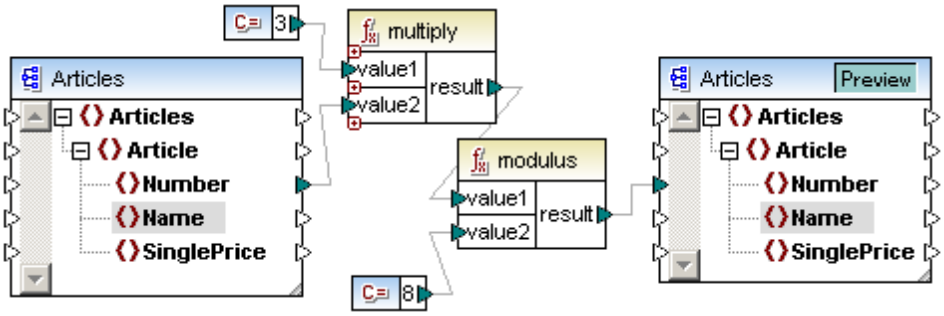
8.9.6.5 *modulus*

Result is the remainder of dividing **value1** by **value2**.



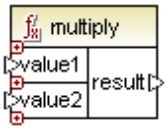
In the mapping below, the numbers have been multiplied by 3 and passed on to value1 of the modulus function. Input values are now 3, 6, 9, and 12.

When applying/using modulus 8 as value2, the remainders are 3, 6, 1, and 4.



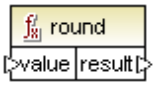
8.9.6.6 *multiply*

Result is the decimal value of multiplying **value1** by **value2**.



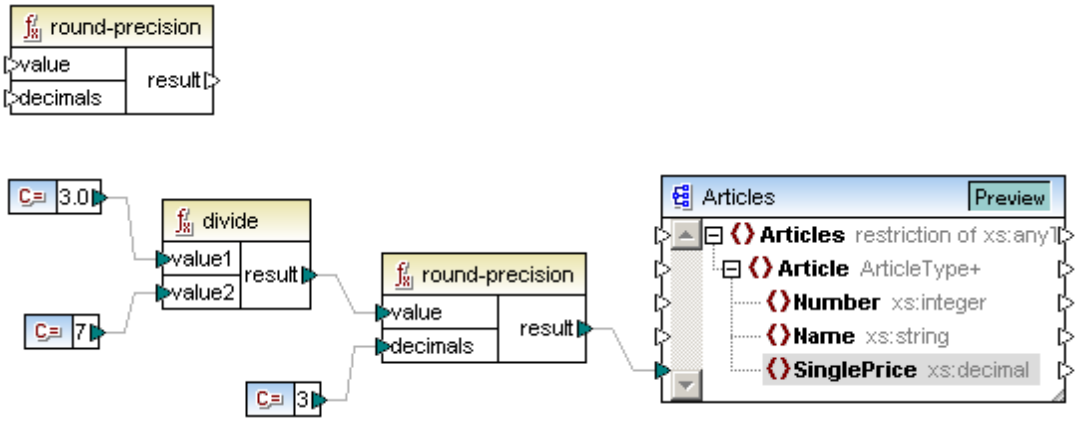
8.9.6.7 *round*

Returns the value rounded to the nearest integer. When the value is exactly in between two integers, the "Round Half Towards Positive Infinity" algorithm is used. For example, the value "10.5" gets rounded to "11", and the value "-10.5" gets rounded to "-10".



8.9.6.8 round-precision

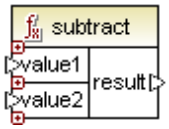
Result is the decimal value of the number rounded to the decimal places defined by "decimals".



In the mapping above, the result is 0.429. For the result to appear correctly in an XML file, make sure to map it to an element of xs:decimal type.

8.9.6.9 subtract

Result is the decimal value of subtracting **value2** from **value1**.

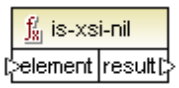


8.9.7 core | node functions

The node functions allow you to access nodes, or process nodes in a particular way.

8.9.7.1 is-xsi-nil

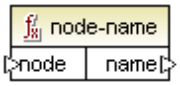
Returns true (<OrderID>true</OrderID>) if the element node, of the source componenent, has the xsi:nil attribute set to "true".



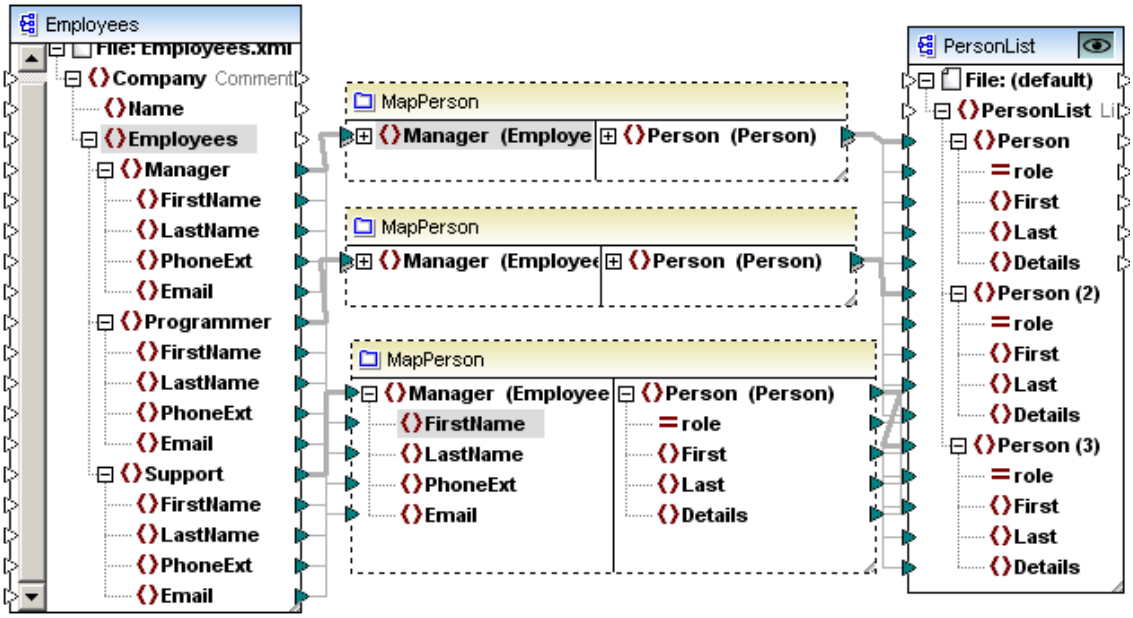
8.9.7.2 node-name

Returns the QName of the connected node unless it is an XML text() node; if this is the case, an empty QName is returned. This function only works on those nodes that have a name. If XSLT is

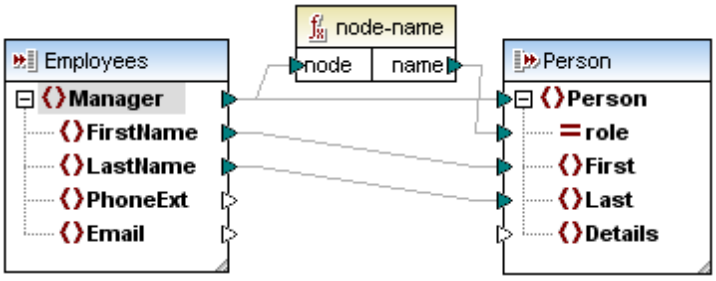
the target language (which calls fn:node-name), it returns an empty sequence for nodes which have no names.



- Getting a name from database tables/fields is not supported.
- XBRL and Excel are not supported.
- Getting a name of File input node is not supported.
- WebService nodes behave like XML nodes except that:
  - node-name from "part" is not supported.
  - node-name from root node ("Output" or "Input") is not supported.



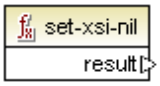
The MapPerson user-defined function uses **node-name** to return the name of the input **node**, and place it in the role attribute. The root node of the Employees.xsd, in the user-defined function, has been defined as "Manager".



Manager gets its data from **outside** the user-defined function, where it can be either: Manager, Programmer, or Support. This is the data that is then passed on to the role attribute in PersonList.

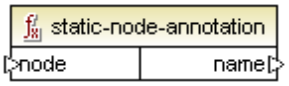
8.9.7.3 set-xsi-nil

Sets the target node to xsi:nil.



8.9.7.4 static-node-annotation

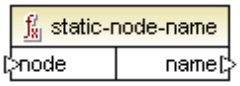
Returns the string with annotation of the connected node. The input must be: (i) a source component node, or (ii) an inline function that is directly connected to a parameter, which in turn is directly connected to a node in the calling mapping.



The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.

8.9.7.5 static-node-name

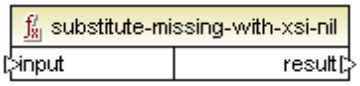
Returns the string with the name of the connected node. The input must be: (i) a source component node, or (ii) an inline function that is directly connected to a parameter, which in turn is directly connected to a node in the calling mapping.



The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.

8.9.7.6 substitute-missing-with-xsi-nil

For nodes with simple content, this function substitutes any missing (or null values) of the source component, with the `xsi:nil` attribute in the target node.



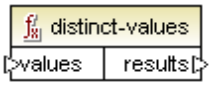
8.9.8 core | sequence functions

Sequence functions allow processing of input sequences and grouping of their content. The value/ content of the **key** input parameter, mapped to nodes/rows, is used to group the sequence.

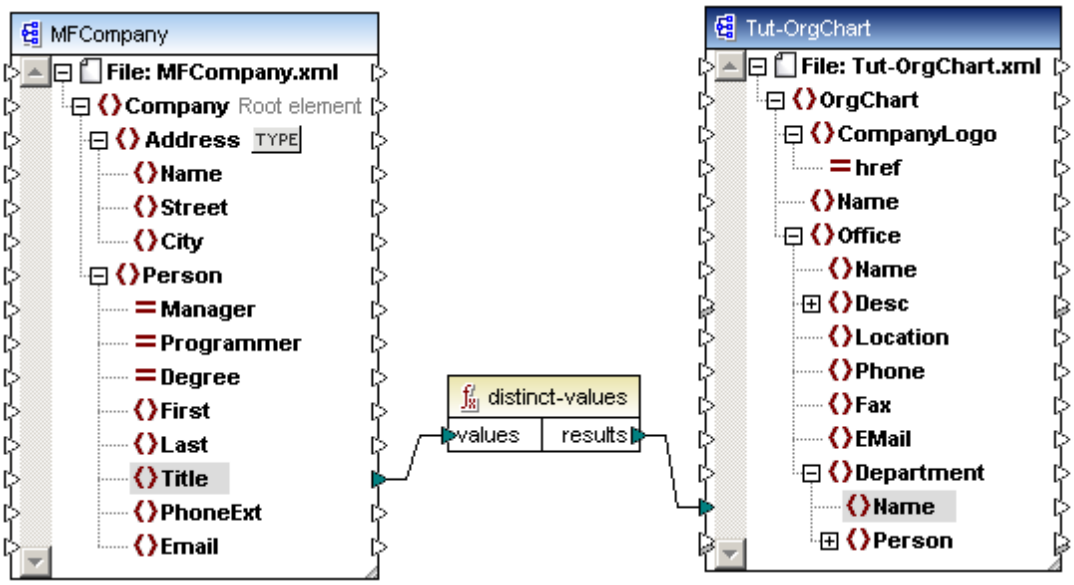
- Input parameter **key** is of an arbitrary data type that can be converted to string for **group-adjacent** and **group-by**
- Input parameter **bool** is of type Boolean for **group-starting-with** and **group-ending-with**
- The output **key** is the key of the current group.

8.9.8.1 distinct-values

Allows you to remove duplicate values from a sequence and map the unique items to the target component.



In the example below, the content of the source component "Title" items, are scanned and each unique title is mapped to the Department / Name item in the target component.

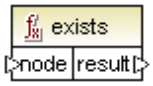


Note that the sequence of the individual Title items in the source component are retained when mapped to the target component.


```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <OrgChart xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart C:/DOCU
3    <Office>
4      <Department>
5        <Name>Office Manager</Name>
6        <Name>Accounts Receivable</Name>
7        <Name>Accounting Manager</Name>
8        <Name>Marketing Manager Europe</Name>
9        <Name>Art Director</Name>
10       <Name>Program Manager</Name>
11       <Name>Software Engineer</Name>
12       <Name>Technical Writer</Name>
13       <Name>IT Manager</Name>
14       <Name>Web Developer</Name>
15       <Name>Support Engineer</Name>
16       <Name>PR &amp;amp; Marketing Manager US</Name>
17     </Department>
18   </Office>
19 </OrgChart>
```

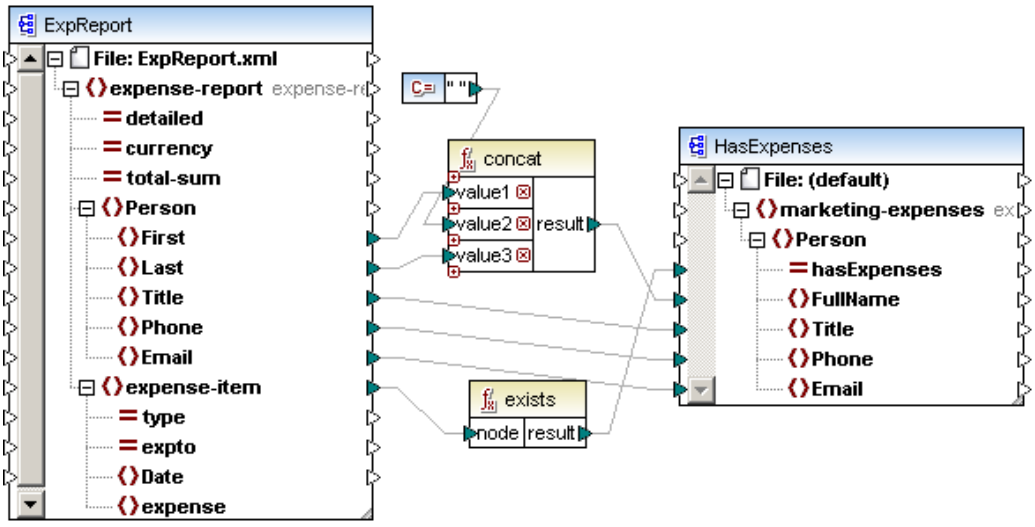
8.9.8.2 *exists*

Returns true if the node exists, else returns false.



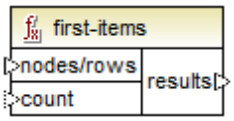
The "**HasMarketingExpenses.mfd**" file in the ...\MapForceExamples folder contains the small example shown below.

If an expense-item exists in the source XML, then the "hasExpenses" attribute is set to "true" in the target XML/Schema file.



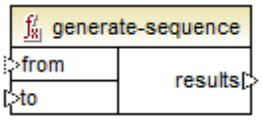
8.9.8.3 first-items

Returns the first "X" items of the input sequence, where X is the number supplied by the "count" parameter. E.g. if the value 3 is mapped to the count parameter and a parent node to the nodes/row parameter, then the first three items will be listed in the output.



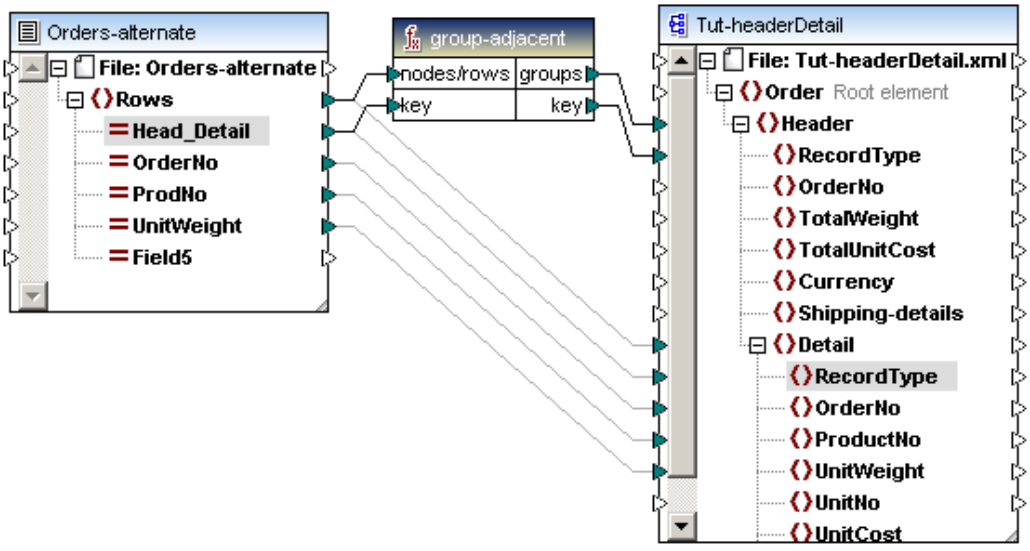
8.9.8.4 generate-sequence

Creates a sequence of integers using the "from" and "to" parameters as the boundaries.



8.9.8.5 group-adjacent

Groups the input sequence **nodes/rows** into groups of adjacent items sharing the same **key**. Note that group-adjacent uses the **content** of the node/item as the grouping key.



Given the CSV file shown below, what we want to happen is to have all the Header and Detail records in their own groups.

Head/Detail	OrderNo	ProdNo	Unitweight	Field5	Field6	Field7	Field8	Fi
string	string	string	string	string	string	string	string	st
H	111	G-889-11	9		Container ship			
D	111	B-192-409	1	2	232	Barley		
D	111	F-152-427	3	1	456	Corn		
D	111	Q-821-207	7	5	52	Coconut		
H	222	A-978-4	22		Air freight			
D	222	M-623-111	8	8	78	Oil		
D	222	L-524-201	2	3	669	Miscellaneous		

- A new group is started with the first element, in this case H.
- As the next element (or key) in the sequence is different, i.e. D, this starts a second group called D.
- The next two D elements are now added to the same group D, as they are of the same type.
- A new H group is started with a single H element.
- Followed by a new D group containing two D elements.

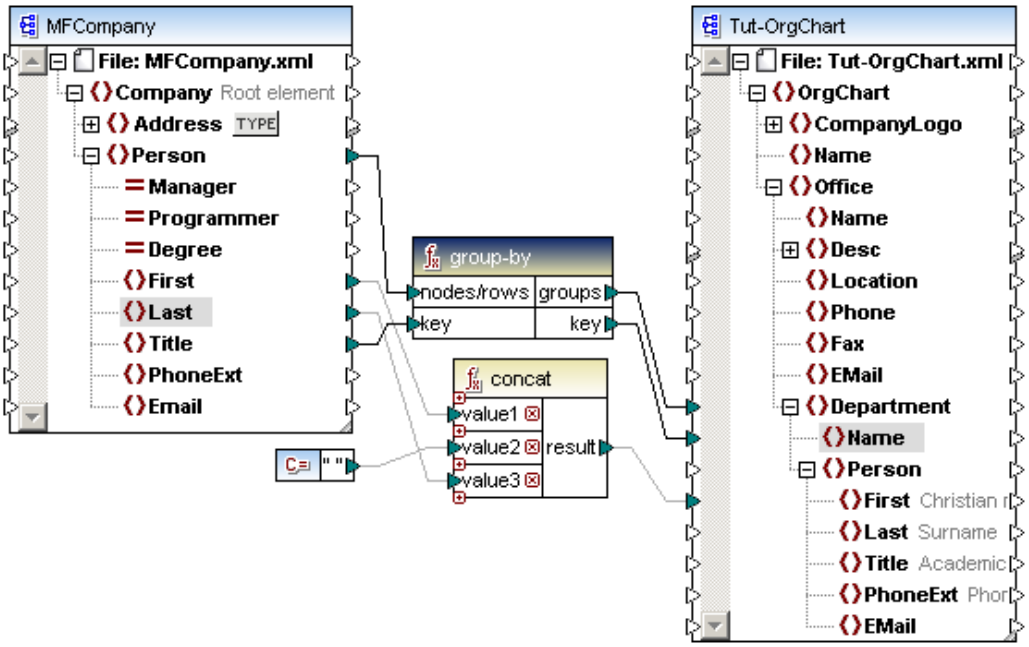
```
3   <Header>
4     <RecordType>H</RecordType>
5     <Detail>
6       <RecordType>H</RecordType>
7       <OrderNo>111</OrderNo>
8       <ProductNo>G-889-11</ProductNo>
9       <UnitWeight>9</UnitWeight>
10    </Detail>
11  </Header>
12  <Header>
13    <RecordType>D</RecordType>
14    <Detail>
15      <RecordType>D</RecordType>
16      <OrderNo>111</OrderNo>
17      <ProductNo>B-192-409</ProductNo>
18      <UnitWeight>1</UnitWeight>
19    </Detail>
20    <Detail>
21      <RecordType>D</RecordType>
22      <OrderNo>111</OrderNo>
23      <ProductNo>F-152-427</ProductNo>
24      <UnitWeight>3</UnitWeight>
25    </Detail>
26    <Detail>
27      <RecordType>D</RecordType>
28      <OrderNo>111</OrderNo>
29      <ProductNo>Q-821-207</ProductNo>
30      <UnitWeight>7</UnitWeight>
31    </Detail>
32  </Header>
```

8.9.8.6 group-by

Groups the input sequence **nodes/rows** into groups of not necessarily adjacent items sharing the same **key**. Groups are output in the order the key occurs in the input sequence. The example

below shows how this works:

- The key that defines the specific groups of the source component is the Title item. This is used to group the persons of the company.
- The group name is placed in the Department/Name item of the target component, while the concatenated person's first and last names are placed in the Person/First child item.



Note that group-by uses the **content** of the node/item as the grouping key. The content of the Title field is used to group the persons and is mapped to the Department/Name item in the target.

Note also: there is an **implied filter** of the rows from the source document to the target document, which can be seen in the included example. In the target document, each Department item only has those Person items that **match** the grouping **key**, as the group-by component creates the necessary hierarchy on the fly.

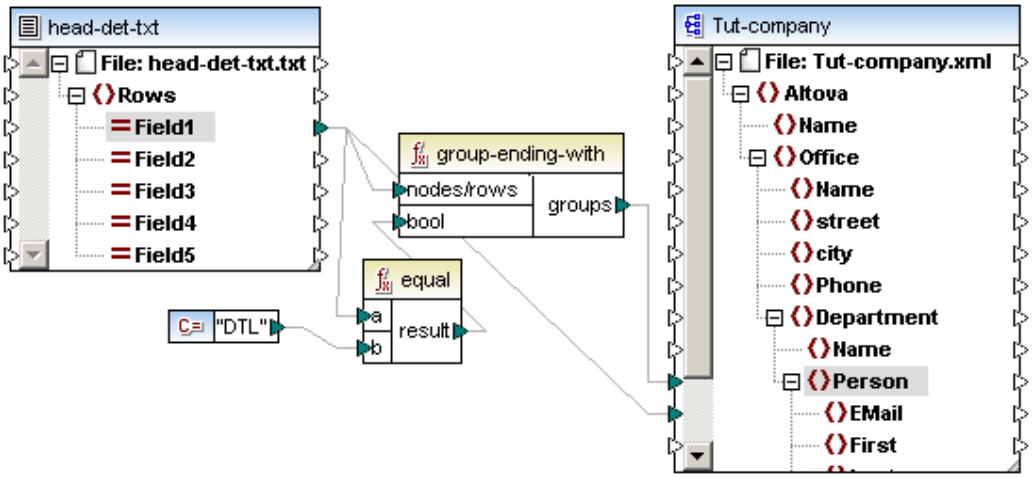
If you have a flat hierarchy (CSV, FLF, etc) with a dynamic output file name, built in part from the key value, the implied filter still exists. This means that you may not need to connect the 'groups' output to any item in the target component.

Clicking the Output button shows the result of the grouping process.

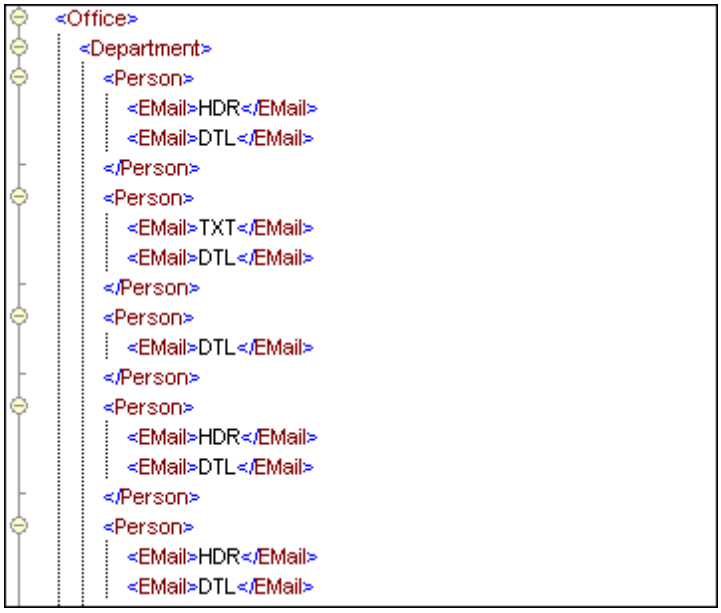

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <OrgChart xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart C:/DOCU
3    <Office>
4      <Department>
5        <Name>Office Manager</Name>
6        <Person>
7          <First>Vernon Callaby</First>
8          <First>Steve Meier</First>
9        </Person>
10     </Department>
11     <Department>
12       <Name>Accounts Receivable</Name>
13       <Person>
14         <First>Frank Further</First>
15         <First>Theo Bone</First>
16       </Person>
17     </Department>
```

8.9.8.7 group-ending-with

This function groups the input sequence **nodes/rows** into groups, ending a new group whenever **bool** is true. This example shows the result when using "DTL" as the group-ending-with item.



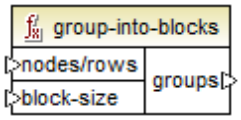
In this case the **value** of the item/nodes do not need to be identical or even exist. The node "**pattern**" i.e. the node/item names need to be identical for the grouping to occur.



The result above shows that a new group was started wherever "DTL" can be the last element.

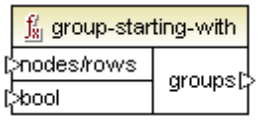
8.9.8.8 group-into-blocks

Groups the input sequence **nodes/rows** into blocks of the same size defined by the number supplied by the **block-size** parameter.

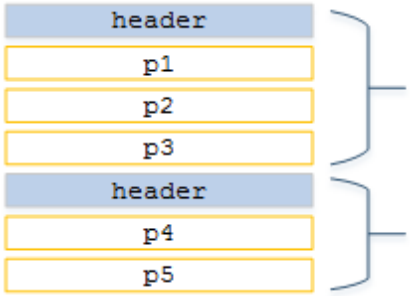


8.9.8.9 group-starting-with

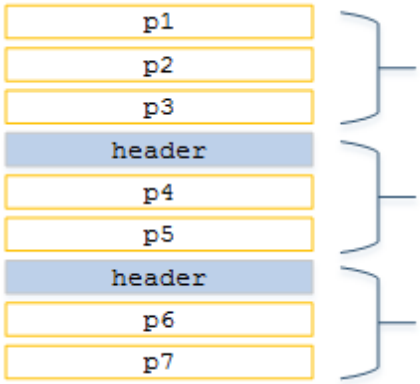
This function groups the input sequence **nodes/rows** into groups, starting a new group when **bool** is `true`.



The following example illustrates a sequence of nodes where **bool** returns `true` whenever the node "header" is encountered. Applying the **`group-starting-with`** function on this sequence of nodes results in two groups, as shown below.

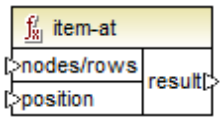


Note that the first node in the sequence starts a new group regardless of the value of **bool**. In other words, a sequence such as the one below would create three groups.



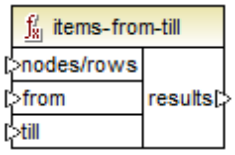
8.9.8.10 item-at

Returns the **nodes/rows** at the position supplied by the **position** parameter. The first item is at position "1".



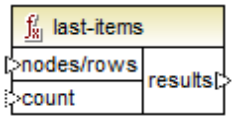
8.9.8.11 items-from-till

Returns a sequence of **nodes/rows** using the "from" and "till" parameters as the boundaries. The first item is at position "1".



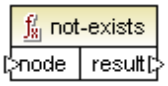
8.9.8.12 last-items

Returns the last "X" **nodes/rows** of the sequence where X is the number supplied by the "count" parameter. The first item is at position "1".



8.9.8.13 not-exists

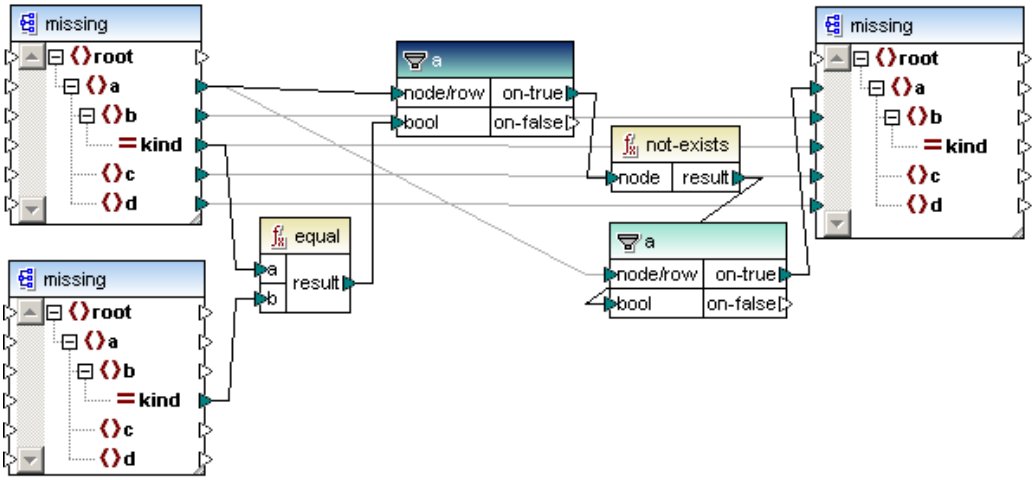
Returns false if the node exists, else returns true.



The example below shows how you can use the not-exists function to map nodes that do not exist in one of a pair of source files.

What this mapping does:

- Compare the nodes of two source XML files
- Filter out the nodes of the first source XML file, that do not exist in the second source XML file
- Map only the missing nodes, and their content, to the target file.

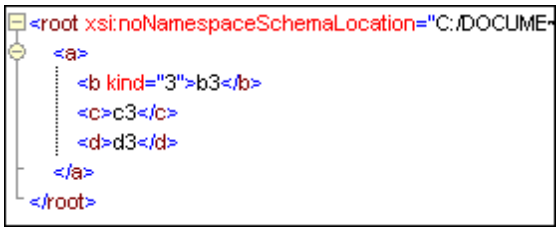


The two XML instance files are shown below, the differences between them are:

- **a.xml** (left) contains the node <b **kind="3"**>, which is missing from b.xml.
- **b.xml** (right) contains the node <b **kind="4">** which is missing from a.xml.

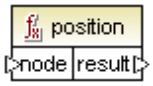
- The equal function compares the kind attribute of both XML files and passes the result to the filter.
- A not-exists function is placed after the initial filter, to select the missing nodes of each of the source files.
- The second filter is used to pass on the missing node and other data only from the a.xml file to the target.

The mapping result is that the node missing from b.xml, <b kind="3">, is passed on to the target component.



8.9.8.14 position

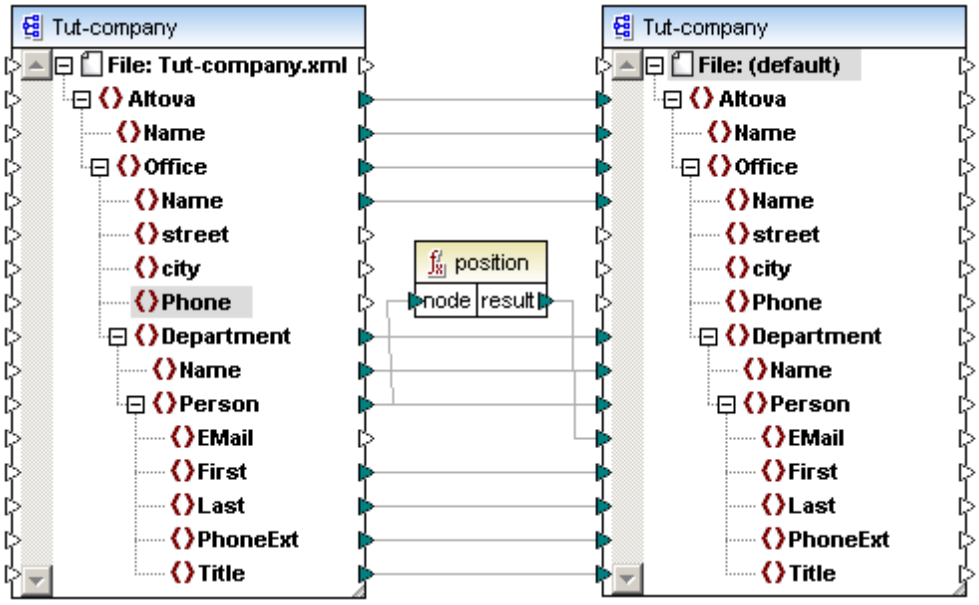
Returns the position of a node inside its containing sequence.



The position function allows you to determine the position of a specific node in a sequence, or use a specific position to filter out items based on that position.

The context item is defined by the item connected to the "node" parameter of the position function, Person, in the example below.

The simple mapping below adds a position number to each Person of each Department.



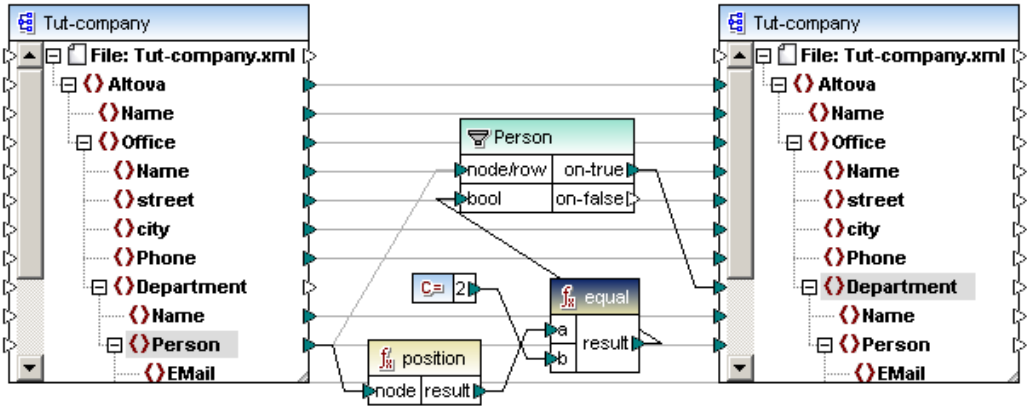
The position number is reset for each Department in the Office.

```
<Office>
  <Name>Microtech, Inc.</Name>
  <Department>
    <Name>Admin</Name>
    <Person>
      <EMail>1</EMail>
      <First>Albert</First>
      <Last>Aldrich</Last>
      <PhoneExt>582</PhoneExt>
      <Title>Manager</Title>
    </Person>
    <Person>
      <EMail>2</EMail>
      <First>Bert</First>
      <Last>Bander</Last>
      <PhoneExt>471</PhoneExt>
      <Title>Accounts Receivable</Title>
    </Person>
```

Using the position function to filter out specific nodes

Using the position function in conjunction with a filter allows you to map only those specific nodes that have a certain position in the source component.

The filter "node/row" parameter and the position "node" must be connected to the same item of the source component, to filter out a specific position of that sequence.



What this mapping does is to output:

- The second Person in each Department
- of each Office in Altova.

```
<Office>
  <Name>Microtech, Inc.</Name>
  <Department>
    <Name>Admin</Name>
    <Person>
      <EMail>b.bander@microtech.com</EMail>
      <First>Bert</First>
      <Last>Bander</Last>
      <Title>Accounts Receivable</Title>
    </Person>
  </Department>
  <Department>
    <Name>Sales and Marketing</Name>
    <Person>
      <EMail>e.ellas@microtech.com</EMail>
      <First>Eve</First>
      <Last>Ellas</Last>
      <Title>Art Director</Title>
    </Person>
  </Department>
  <Department>
    <Name>Manufacturing</Name>
    <Person>
      <EMail>g.gundall@microtech.com</EMail>
```

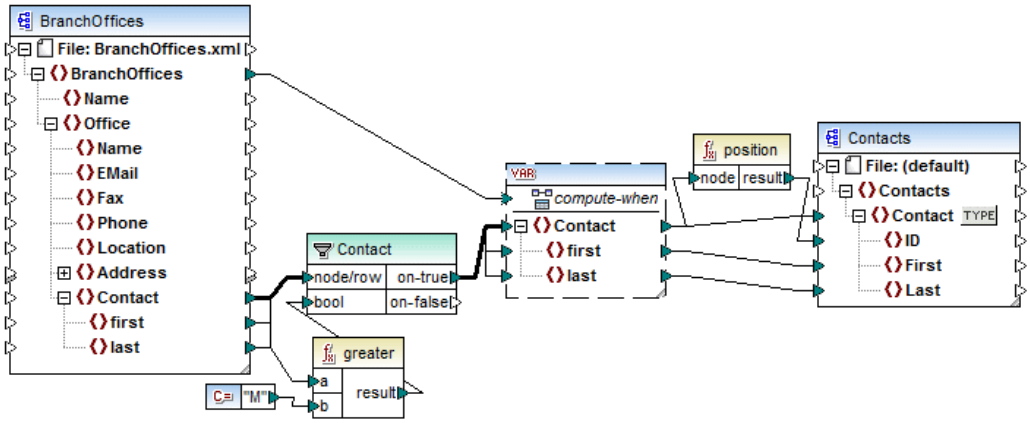
Finding the position of items in a filtered sequence:

As the filter component is not a sequence function, it cannot be used directly in conjunction with the position function to find the position of filtered items. To do this you have to use the "Variable" component.

The results of a Variable component are always sequences, i.e. a delimited list of values, which can also be used to create sequences.

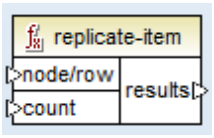
- The variable component is used to collect the filtered contacts where the last name starts with a letter higher than "M".
- The contacts are then passed on (from the variable) to the target component

- The position function then numbers these contacts sequentially



8.9.8.15 *replicate-item*

Repeats every item in the input sequence the number of times specified in the `count` argument. If you connect a single item to the `node/row` argument, the function returns N items, where N is the value of the `count` argument. If you connect a sequence of items to the `node/row` argument, the function repeats each individual item in the sequence `count` times, processing one item at a time. For example, if `count` is 2, then the sequence `(1,2,3)` produces `(1,1,2,2,3,3)`.

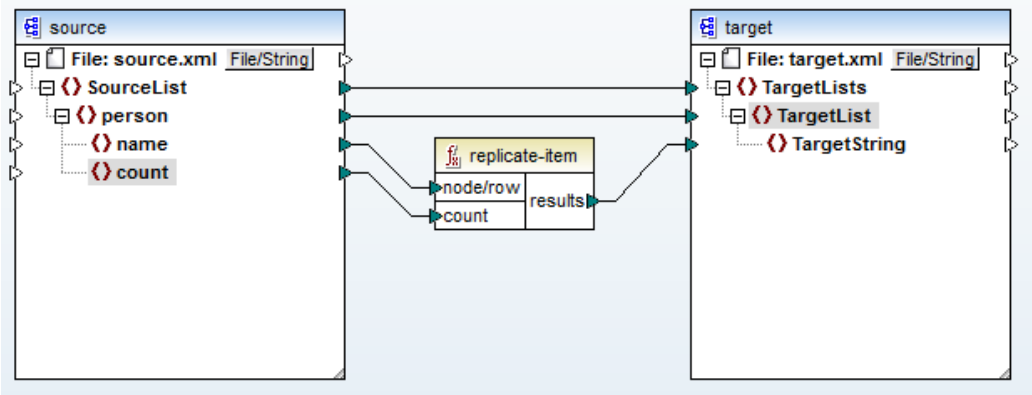


Note that you can supply a different `count` value for each item. For example, let's assume that you have a source XML file with the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<SourceList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="source.xsd">
   <person>
      <name>Michelle</name>
      <count>2</count>
   </person>
   <person>
      <name>Ted</name>
      <count>4</count>
   </person>
   <person>
      <name>Ann</name>
      <count>3</count>
   </person>
</SourceList>
```

With the help of the **`replicate-item`** function, you can repeat each person name a different number of times in a target component. To achieve this, connect the `<count>` node of each

person to the `count` input of the **`replicate-item`** function:

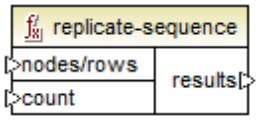


The output is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<TargetLists xsi:noNamespaceSchemaLocation="target.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <TargetList>
      <TargetString>Michelle</TargetString>
      <TargetString>Michelle</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
   </TargetList>
</TargetLists>
```

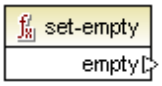
8.9.8.16 *replicate-sequence*

Repeats all items in the input sequence the number of times specified in the `count` argument. For example, if `count` is 2, then the sequence `(1,2,3)` produces `(1,2,3,1,2,3)`.



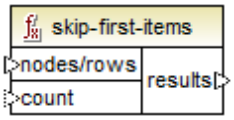
8.9.8.17 set-empty

Returns an empty sequence. For example, you can use this function to cancel default values of an XBRL document that were defined higher up the XBRL component/taxonomy.



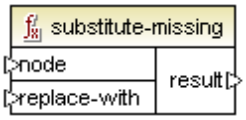
8.9.8.18 skip-first-items

Skips the first "X" items/nodes of the input sequence, where X is the number supplied by the "count" parameter, and returns the rest of the sequence.



8.9.8.19 substitute-missing

This function is a convenient combination of exists and a suitable if-else condition. Used to map the current field content if the node exists in the XML source file, otherwise use the item mapped to the "replace-with" parameter.

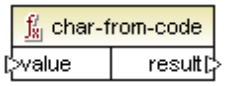


8.9.9 core | string functions

The string functions allow you to use the most common string functions to manipulate many types of source data to: extract portions, test for substrings, or retrieve information on strings.

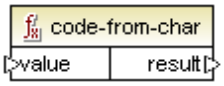
8.9.9.1 char-from-code

Result is the character representation of the decimal Unicode value of **value**.



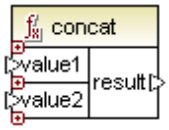
8.9.9.2 code-from-char

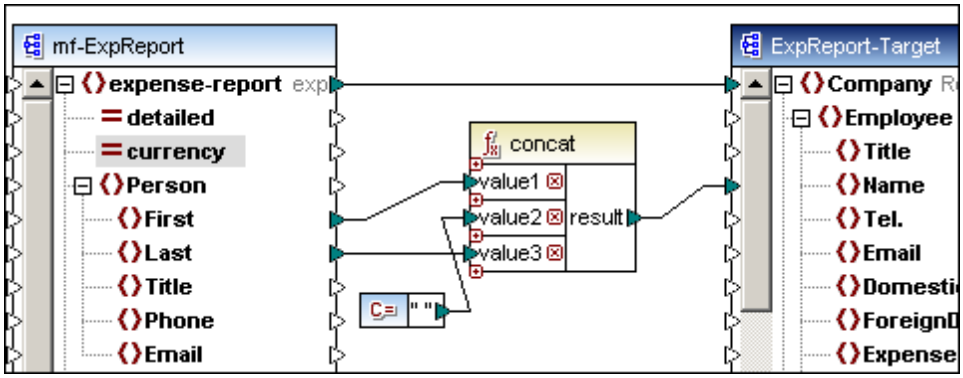
Result is the decimal Unicode value of the first character of **value**.



8.9.9.3 concat

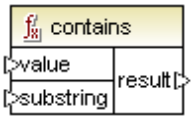
Concatenates (appends) two or more values into a single result string. All input values are automatically converted to type string.





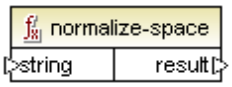
8.9.9.4 contains

Result is true if data supplied to the value parameter contains the string supplied by the substring parameter.



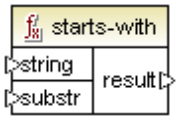
8.9.9.5 normalize-space

Result is the normalized input string, i.e. leading and trailing spaces are removed, then each sequence of multiple consecutive whitespace characters are replaced by a single whitespace character. The Unicode character for "space" is (U+0020).



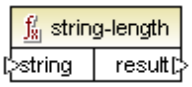
8.9.9.6 starts-with

Result is true if the input string "string" starts with **substr**, else false.



8.9.9.7 string-length

Result is the number of characters supplied by the **string** parameter.



8.9.9.8 substring

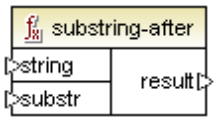
Result is the substring (string fragment) of the "string" parameter where "start" defines the position of the start character, and "length" the length of the substring.

If the length parameter is not specified, the result is a fragment starting at the start position and ending at the end position of the string. Indices start counting at 1.

E.g. substring("56789",2,3) results in 678.

8.9.9.9 substring-after

Result is the remainder of the "**string**" parameter, where the first occurrence of the **substr** parameter defines the start characters; the remainder of the string is the result of the function. An empty string is the result, if **substr** does not occur in **string**.

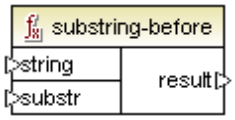


E.g. substring-after("2009/01/04","/") results in the substring 01/04. substr in this case is the first "/"character.

For an example of usage, see Example: Mapping Data from an RSS Feed.

8.9.9.10 substring-before

Result is the string fragment of the "**string**" parameter, up to the first occurrence of the **substr** characters. An empty string is the result, if **substr** does not occur in **string**.



E.g. substring-before ("2009/01/04","/") results in the substring 2009. substr in this case is the first "**/**" character.

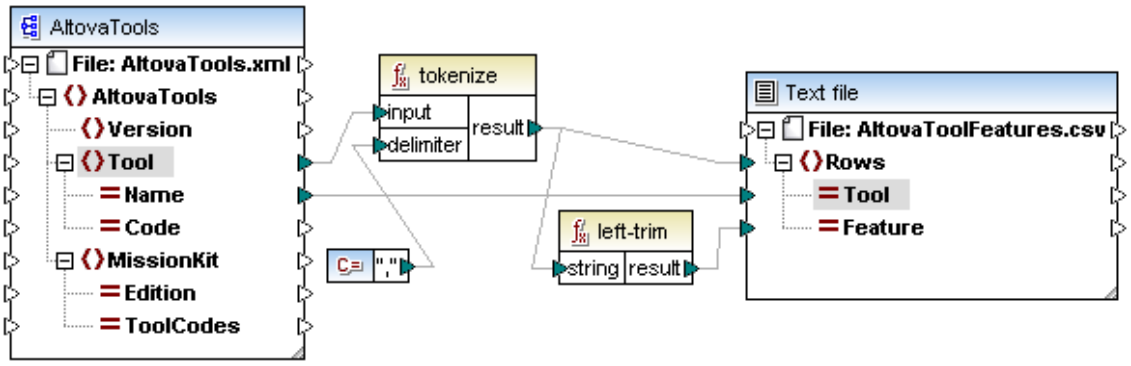
8.9.9.11 tokenize

Result is the input string split into a sequence of chunks/sections defined by the delimiter parameter. The result can then be passed on for further processing.

E.g. Input string is A,B,C and delimiter is "," - then result is A B C.

Example

The **tokenizeString1.mfd** file available in the ...\MapForceExamples folder shows how the tokenize function is used.



The XML source file is shown below. The **Tool** element has two attributes: Name and Code, with the Tool element data consisting of comma delimited text.

AltovaTools

xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:noName...	AltovaTools.xsd
Version	2010

Tool (9)

	Name	Code	Text
1	XMLSpy	XS	XML editor, XSLT editor, XSLT debugger, XQuery editor, XQuery debugger,
2	MapForce	MF	Data integration, XML mapping, database mapping, text conversion, EDI tran
3	StyleVision	SV	Stylesheet designer, electronic forms, XSLT design, XSL:FO design, databa
4	UModel	UM	UML modeling tool, code generation, reverse engineering, UML, BPMN, Sysl
5	DatabaseSpy	DS	Multi-database tool, SQL auto-completion, graphical database design, table b
6	DiffDog	DD	Diff / merge tool, compare files, sync directories, compare XML, compare O
7	SchemaAgent	SA	XML Schema management tool, IIR management, XSLT management, WSDL
8	SemanticWorks	SW	Semantic Web tool, RDF editor, OWL editor, RDF/XML and N-Triples generati
9	Authentic	AU	XML authoring tool, database editor, XML publishing tool, e-Forms editor

MissionKit (4)

What the mapping does:

- The tokenize function receives data from the **Tool** element/item and uses the comma "**,**" delimiter to split that data into separate chunks. I.e. the first chunk "XML editor".
- As the result parameter is mapped to the **Rows** item in the target component, one **row** is generated for each chunk.
- The result parameter is also mapped to the **left-trim** function which removes the leading white space of each chunk.
- The result of the left-trim parameter (each chunk) is mapped to the **Feature** item of the target component.
- The target component output file has been defined as a CSV file (AltovaToolFeatures.csv) with the field delimiter being a semicolon (double click component to see settings).

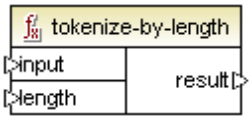
Result of the mapping:

- For each Tool element of the source file
- The (Tool) Name is mapped to the Tool item in the target component
- Each chunk of the tokenized Tool content is appended to the (Tool Name) Feature item
- E.g. The first tool, XMLSpy, gets the first Feature chunk "XML editor"
- This is repeated for all chunks of the current Tool and then for all Tools.
- Clicking the Output tab delivers the result shown below.

```
1   Tool;Feature
2   XMLSpy;XML editor
3   XMLSpy;XSLT editor
4   XMLSpy;XSLT debugger
5   XMLSpy;XQuery editor
6   XMLSpy;XQuery debugger
7   XMLSpy;XML Schema / DTD editor
8   XMLSpy;WSDL editor
9   XMLSpy;SOAP debugger
10  MapForce;Data integration
11  MapForce;XML mapping
12  MapForce;database mapping
```

8.9.9.12 tokenize-by-length

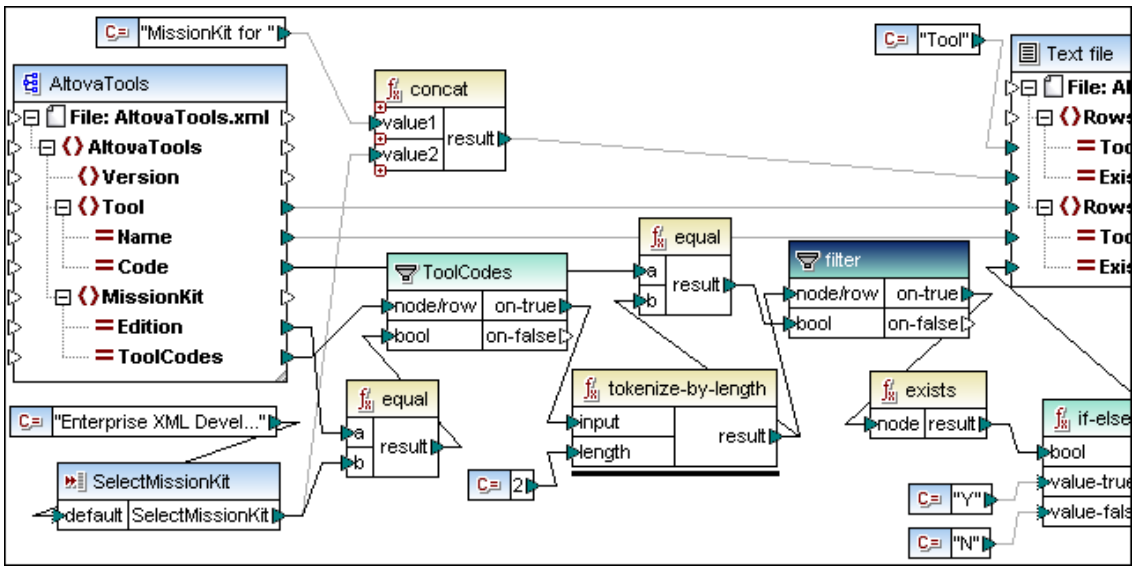
Result is the input string split into a sequence of chunks/sections defined by the length parameter. The result can then be passed on for further processing.



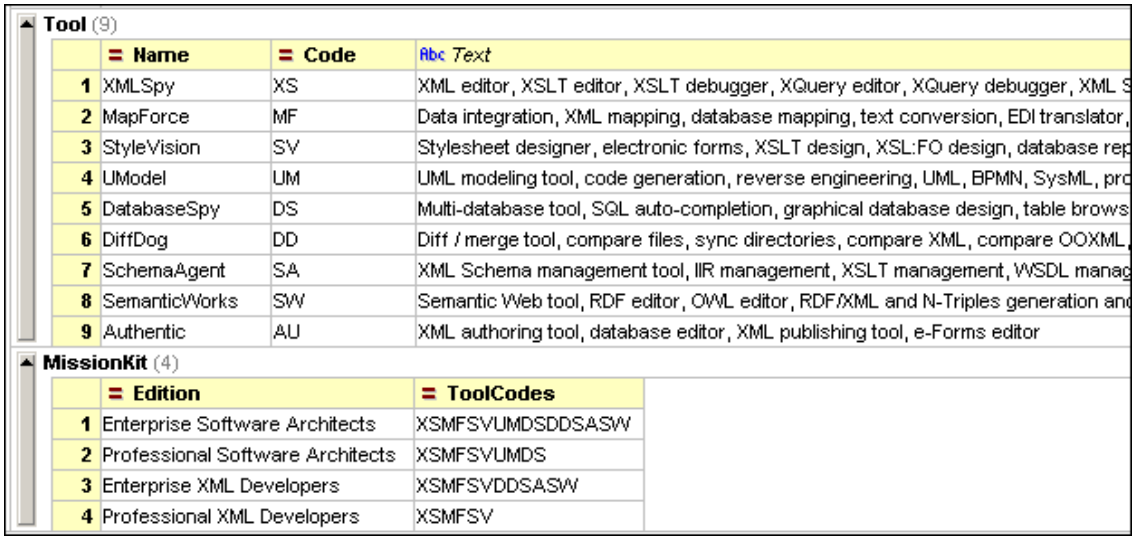
E.g. Input string is ABCDEF and length is "2" - then result is AB CD EF.

Example

The **tokenizeString2.mfd** file available in the ...\MapForceExamples folder shows how the **tokenize-by-length** function is used.



The XML source file is shown below, and is the same as the one used in the previous example. The **MissionKit** element also has two attributes: **Edition** and **ToolCodes**, but no MissionKit element content.



Tool (9)

	= Name	= Code	Abc Text
1	XMLSpy	XS	XML editor, XSLT editor, XSLT debugger, XQuery editor, XQuery debugger, XML S
2	MapForce	MF	Data integration, XML mapping, database mapping, text conversion, EDI translator,
3	StyleVision	SV	Stylesheet designer, electronic forms, XSLT design, XSL:FO design, database rep
4	UModel	UM	UML modeling tool, code generation, reverse engineering, UML, BPMN, SysML, pro
5	DatabaseSpy	DS	Multi-database tool, SQL auto-completion, graphical database design, table brows
6	DiffDog	DD	Diff / merge tool, compare files, sync directories, compare XML, compare OOXML,
7	SchemaAgent	SA	XML Schema management tool, IIR management, XSLT management, WSDL manag
8	SemanticWorks	SW	Semantic Web tool, RDF editor, OWL editor, RDF/XML and N-Triples generation and
9	Authentic	AU	XML authoring tool, database editor, XML publishing tool, e-Forms editor

MissionKit (4)

	= Edition	= ToolCodes
1	Enterprise Software Architects	XSMFSVUMDSDDSASW
2	Professional Software Architects	XSMFSVUMDS
3	Enterprise XML Developers	XSMFSVDDSASW
4	Professional XML Developers	XSMFSV

Aim of the mapping:

To generate a list showing which Altova tools are part of the respective MissionKit editions.

How the mapping works:

- The SelectMissionKit **Input** component receives its default input from a constant component, in this case "Enterprise XML Developers".
- The **equal** function compares the input value with the "**Edition**" value and passes on the result to the **bool** parameter of the ToolCodes filter.
- The **node/row** input of the ToolCodes filter is supplied by the **ToolCodes** item of the source file. The value for the Enterprise XML Developers edition is: XSMFSVDDSASW.
- The XSMFSVDDSASW value is passed to the **on-true** parameter, and futher to the **input** parameter of the **tokenize-by-length** function.

What the tokenize-by-length function does:

- The ToolCodes **input** value XSMFSVDDSASW, is split into multiple chunks of two characters each, defined by **length** parameter, which is 2, thus giving 6 chunks.
- Each chunk (placed in the b parameter) of the equal function, is compared to the 2 character **Code** value of the source file (of which there are 9 entries/items in total).
- The result of the comparison (true/false) is passed on to the **bool** parameter of the filter.
- Note that **all** chunks, of the tokenize-by-length function, are passed on to the **node/row** parameter of the filter.

- The **exists** functions now checks for existing/non-existing nodes passed on to it by the **on-true** parameter of the filter component.

  Existing nodes are those where there **is** a match between the **ToolCodes** chunk and the Code value.

  Non-existing nodes are where there was **no ToolCodes** chunk to match a Code value.

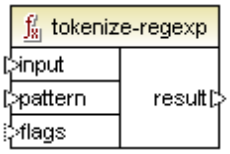
- The bool results of the **exists** function are passed on to the **if-else** function which passes on a Y to the target if the node exists, or a N, if the node does not exist.

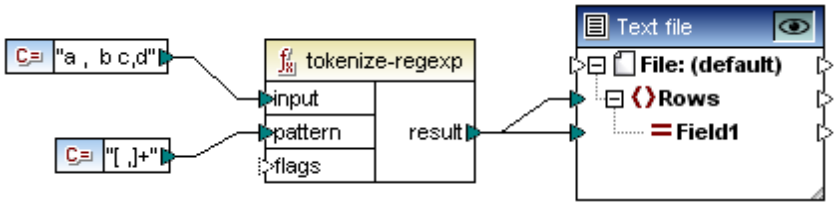
Result of the mapping:


```
1   Tool;MissionKit for Enterprise XML Developers
2   XMLSpy;Y
3   MapForce;Y
4   StyleVision;Y
5   UModel;N
6   DatabaseSpy;N
7   DiffDog;Y
8   SchemaAgent;Y
9   SemanticWorks;Y
10  Authentic;N
11
```

8.9.9.13 tokenize-regexp

Result is the input string split into a sequence of strings, where the supplied regular expression **pattern** match defines the separator. The separator strings are not output by the result parameter. Optional flags may also be used.





In the example shown above:
input string is a succession of characters separated by spaces and/or commas, i.e. a , b c,d

The regex **pattern** defines a character class **[**"space""comma"**]** - of which one and only one character will be matched in a character class, i.e. either space or comma.

The **+** quantifier specifies "one or more" occurrences of the character class/string.

result string is:

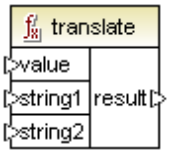
```
1   a
2   b
3   c
4   d
5
```

Please note that there are slight differences in regular expression syntax between the various languages. Tokenize-regexp in C++ is only available in Visual Studio 2008 SP1 and later.

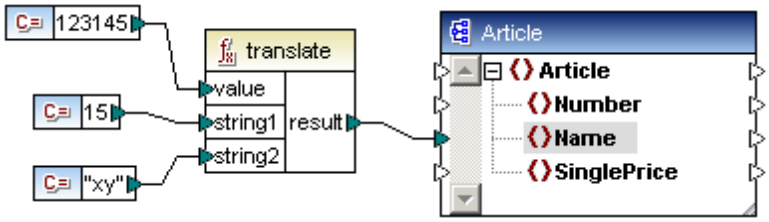
For more information on regular expressions, see Regular expressions.

8.9.9.14 translate

The characters of **string1** (search string) are replaced by the characters at the same position in **string2 (replace string**), in the input string "**value"**.



When there are no corresponding characters in string2, the character is removed.



E.g.
input string is 123145
(search) string1 is 15
(replace) string2 is xy

So:
each 1 is replaced by **x** in the input string value
each 5 is replaced by **y** in the input sting value

Result string is **x23x4y**

If string2 is empty (fewer characters than string1) then the character is removed.

E.g.2
input string aabaacbca
string1 is "**a**"
string2 is **""** (empty string)

result string is "bcbc"

E.g.3
input string aabaacbca
string1 is "**ac**"
string2 is "**ca**"

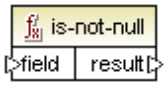
result string is "ccbccabac"

8.9.10 db

The **db** library contains functions that allow you to define the mapping results when encountering null fields in databases.

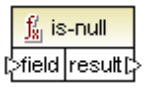
8.9.10.1 is-not-null

Returns false if the field is null, otherwise returns true.



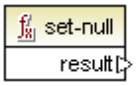
8.9.10.2 is-null

Returns true if the field is null, otherwise returns false.



8.9.10.3 set-null

Used to set a database column to null. This function will also overwrite a default value with null. If connected to something else i.e. not a database field, it will behave like an empty sequence.

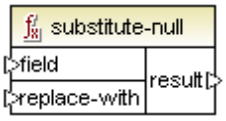


Please note:

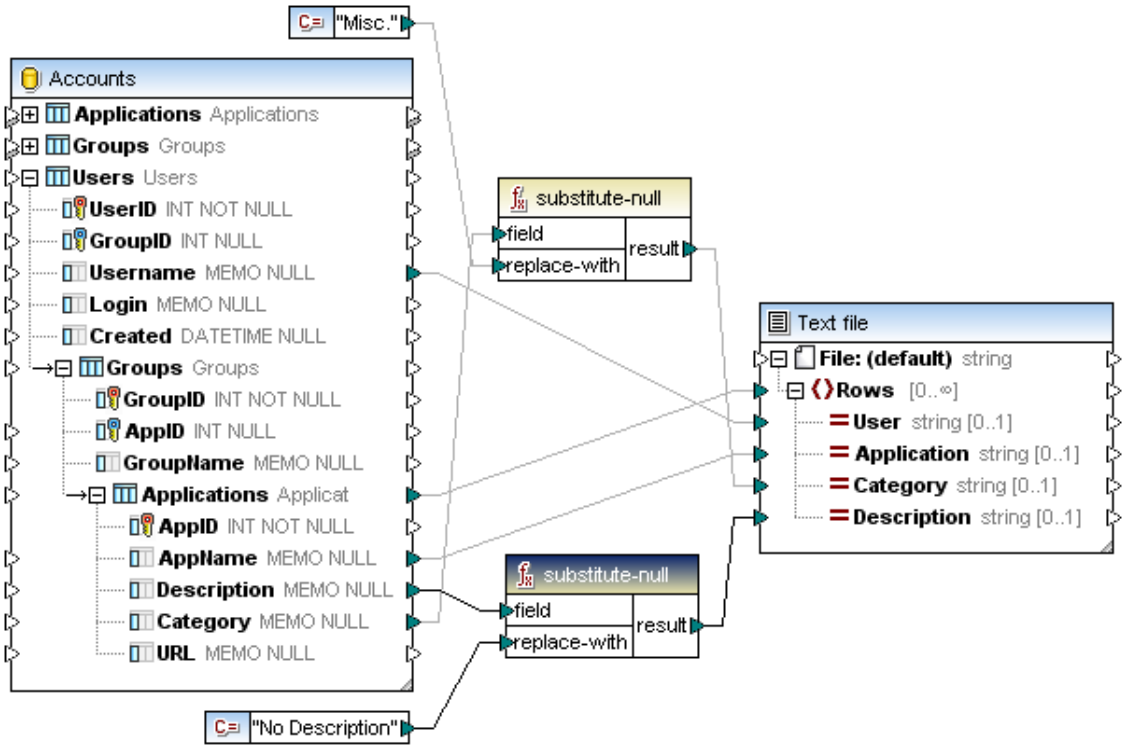
- Connecting set-null to a different function will usually result in the other function not being called at all. Connecting set-null to a sequence function, e.g. count, will call the function with an empty sequence.
- Connecting to special functions, Filters and IF-Else conditions works as expected, fields are set to null. For filters this means the "node/row" input.
- Using set-null as an input for a simpleType element will not create that element in the target component.

8.9.10.4 substitute-null

Used to map the current field content if it exists, otherwise use the item mapped to the replace-with parameter.



The image below shows an example of the substitute-null function in use, and is available as "**DB-ApplicationList**" in the **...\MapForceExamples** folder.



The first function checks if a Category entry exists in the Applications table. As one does not exist for the Notepad application, "Misc" is mapped to the Category item of the Text file.

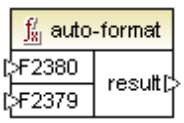
The second function checks if a Description entry exist, and maps the string "No description" if one does not exist, which is also the case with the Notepad application.

8.9.11 edifact

The **edifact** library contains functions that convert EDI date, time, periods into xs:date formats.

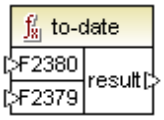
8.9.11.1 auto-format

Result is the Date / Time / Datetime value extracted from the coded source (F2380) given format code (F2379).



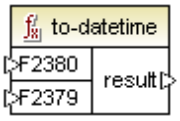
8.9.11.2 to-date

Result is the Date value extracted from the coded source (F2380) given format code (F2379).



8.9.11.3 to-datetime

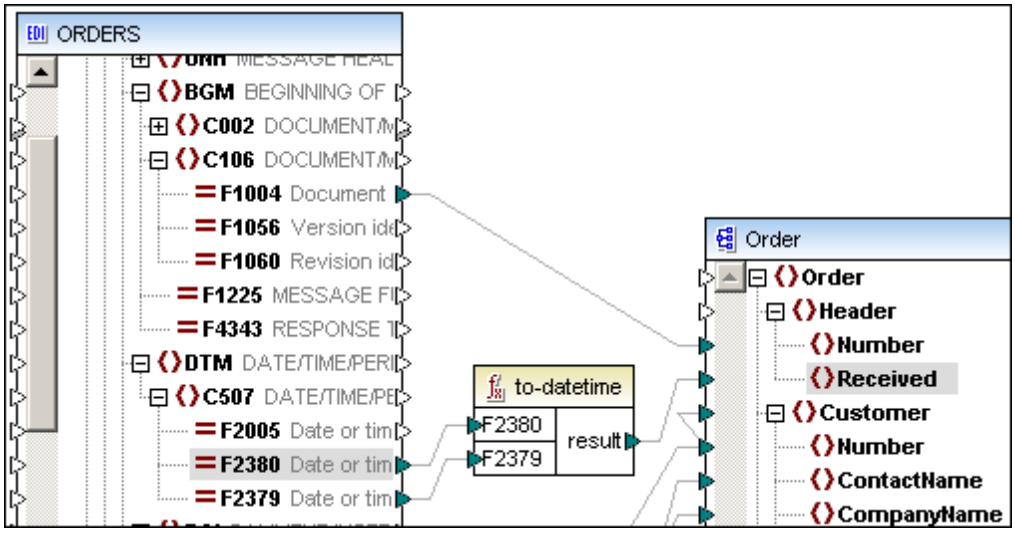
Result is the Datetime value extracted from the coded source (F2380) given format code (F2379).



E.g.
Given the EDI file shown in the screenshot below, the Date/Time/Period value is shown in the DTM field.

```
UNB+UNOB:1+003897733:01:MFGB+PARTNER ID:ZZ:ROUTING
ADDR+970101:1230+00000000000001++ORDERS++++1'
UNH+0001+ORDERS:D:08A:UN'
BGM+221+ABC123456XYZ+9'
DTM+4:20040430174 2PDT:303'
FTX+PUR+3++Pizza purchase order'
RFF+CT:123-456'
RFF+CR:1122'
NAD+SE+999::92++24h Pizza+Long Way+San-Francisco+C
CTA+SR+:Ted Little'
```

The value is converted to an xs:date format using the to-datetime function.

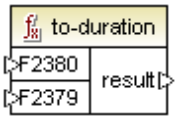


The result of the conversion is shown below. This sample is available in the ...\MapForceExamples as **EDI-Order.mfd**.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xsi:noNamespaceSchemaLocation="C:/DOCUME~1
http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://w
  <Header>
    <Number>ABC123456XYZ</Number>
    <Received>2004-04-30T17:42:00-09:00</Received>
  </Header>
  <Customer>
    <Number>123</Number>
```

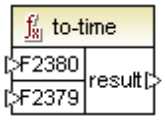
8.9.11.4 to-duration

Result is the Duration value extracted from the coded source (F2380) given format code (F2379).



8.9.11.5 to-time

Result is the Time value extracted from the coded source (F2380) given format code (F2379).

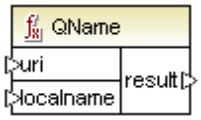


8.9.12 lang | QName functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

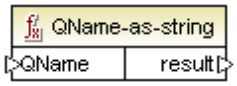
8.9.12.1 QName

Constructs a QName from a namespace URI and a local part. Use this function to create a QName in a target component. The uri and localname parameters can be supplied by a constant function.



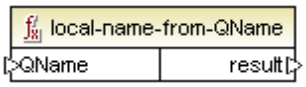
8.9.12.2 QName-as-string

Result is the unique string representation of the QName.

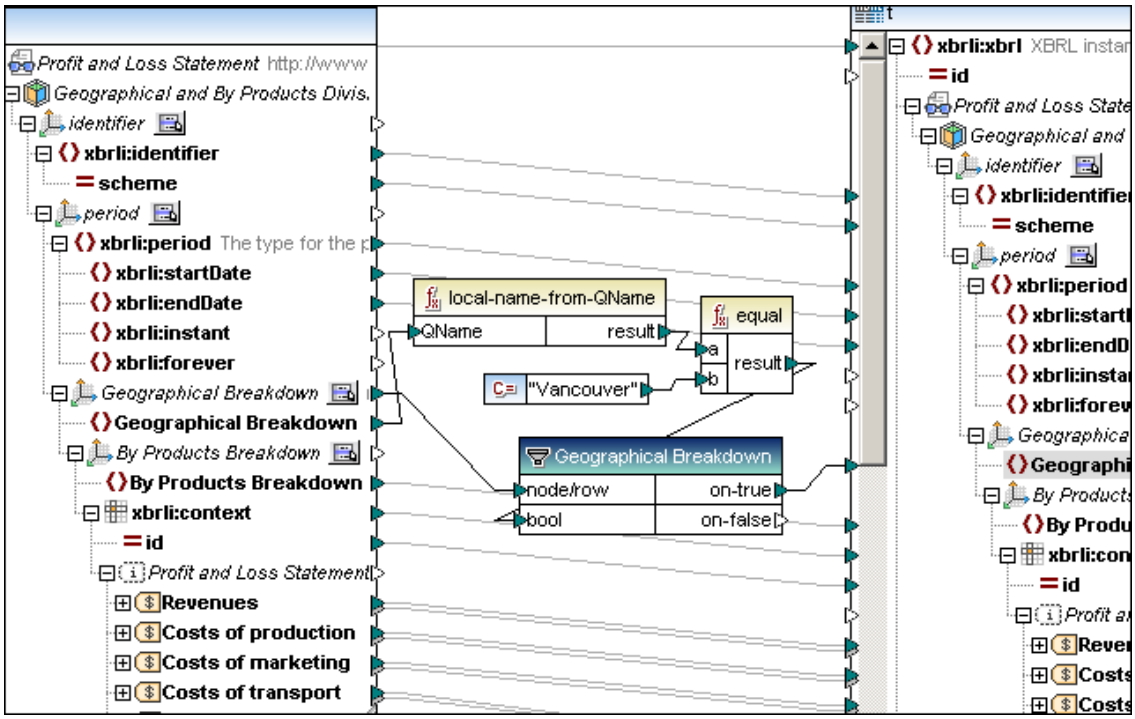


8.9.12.3 local-name-from-QName

Returns the local name part of the QName.



This function is useful when mapping XBRL instance documents containing hypercubes.



What the mapping does is filter those facts where the **local name** of the **content** of the explicit member (d-g:Vancouver) is equal to "Vancouver". Note that the content of the member is itself a QName.

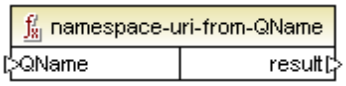
```
<xbrli:context id="Year2007_Vancouver_BeveragesTotal">
  <xbrli:entity>
    <xbrli:identifier scheme="http://www.stockexchange/ticker">ACME</xbrli:identifier>
    <xbrli:segment>
      <xbrldi:explicitMember dimension="d-g:GeographicalBreakdown">d-g:Vancouver</xbrl
      <xbrldi:explicitMember dimension="d-b:ByProductsBreakdown">d-b:BeveragesTotal</x
    </xbrli:segment>
```

All the facts that belong to the dimension GeographicalBreakdown are filtered and passed to the target component.

```
<context id="Year2007_Vancouver_BeveragesTotal">
  <entity>
    <identifier scheme="http://www.stockexchange/ticker">ACME</identifier>
    <segment>
      <explicitMember dimension="d-b:ByProductsBreakdown" xmlns="http://xbrl.org/2006/xbrldi">d-b:Beverages
      <explicitMember dimension="d-g:GeographicalBreakdown" xmlns="http://xbrl.org/2006/xbrldi" />
    </segment>
  </entity>
  <period>
    <startDate>2007-01-01</startDate>
    <endDate>2007-12-31</endDate>
  </period>
</context>
<p:Revenues contextRef="Year2007_Vancouver_BeveragesTotal" unitRef="USD" decimals="0">4</p:Revenues>
<p:CostsOfProduction contextRef="Year2007_Vancouver_BeveragesTotal" unitRef="USD" decimals="0">1</p:Cos
<p:CostsOfMarketing contextRef="Year2007_Vancouver_BeveragesTotal" unitRef="USD" decimals="0">1</p:Cost
<p:ProfitLoss contextRef="Year2007_Vancouver_BeveragesTotal" unitRef="USD" decimals="0">2</p:ProfitLoss>
```

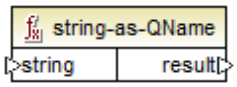
8.9.12.4 namespace-uri-from-QName

Result is the namespace URI part of the QName



8.9.12.5 string-as-QName

Converts the string representation of a QName back to a QName.

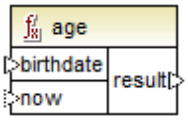


8.9.13 lang | datetime functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

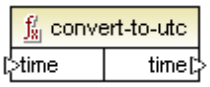
8.9.13.1 age

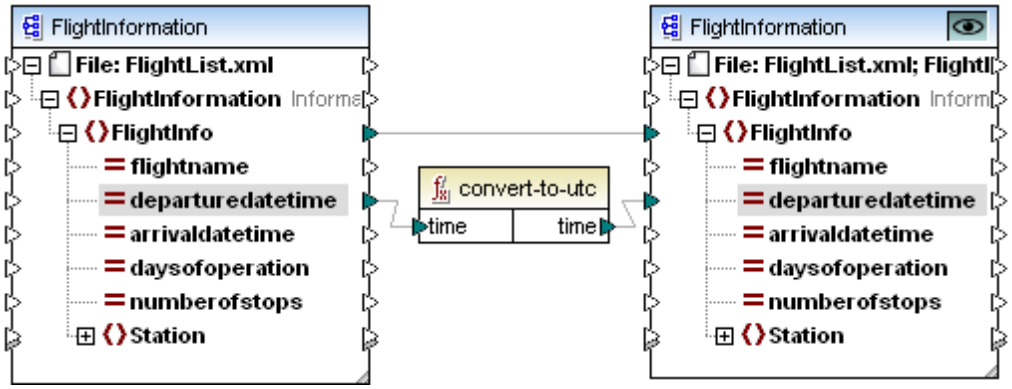
age
Result is the age of the person in full years. The *now* argument is optional and the default is the current system date. The result is then the full amount of years between the birthdate and now. If a value is mapped to the *now* argument, the result is the difference between the two dates in full years.



8.9.13.2 convert-to-utc

Converts the local "time" input parameter into Coordinated Universal Time, or GMT/Zulu time. (The function takes the timezone component, e.g. +5:00, into account).





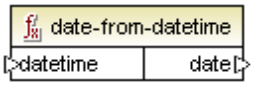
E.g. Instance document datetime:
`departuredatetime="2001-12-17T09:30:02+05:00"`

Result:
`departuredatetime="2001-12-17T04:30:02"`

Please note:
If the source dateTime is in the form `departuredatetime="2001-12-17T09:30:02Z"` then no conversion will take place because the trailing "**Z**" defines this time to be Zulu time, i.e. UTC. The result will be `departuredatetime="2001-12-17T09:30:02".`

8.9.13.3 date-from-datetime

Result is the date part of a datetime input argument. The time part of the dateTime, starting with T in the instance document, is set to zero. Note that the timezone increment is not changed.



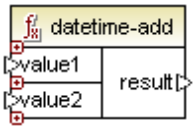
E.g. Instance document datetime:
`departuredatetime="2001-12-17T09:30:02+05:00"`

Result:

departuredatetime="2001-12-17T**00:00:00**+05:00"

8.9.13.4 datetime-add

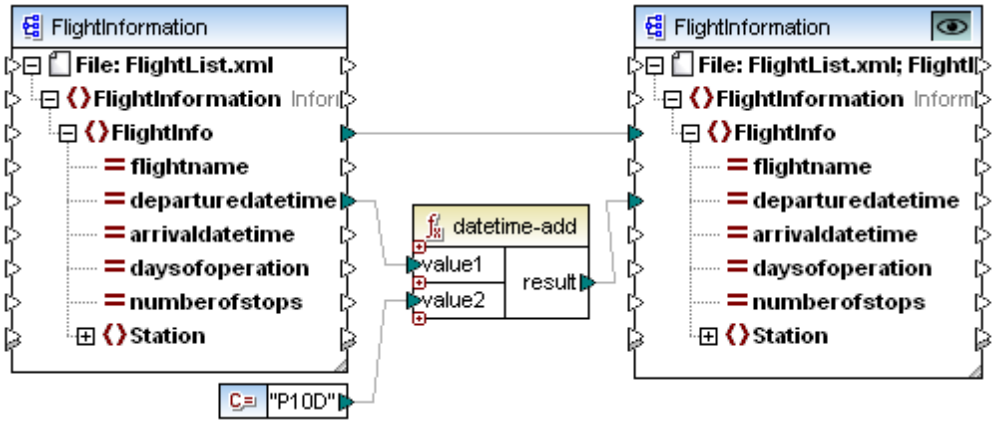
Result is the datetime obtained by adding a duration (second argument) to a datetime (first argument).



Durations must be entered in the form: P1Y2M3DT04H05M. Periods can be made negative by adding the minus character before the P designator, e.g. -P1D.

P is the period designator, and is mandatory; the rest of above period is therefore:
1 Year, 2 Months, 3 Days T(ime designator), 04 Hours, 05 Minutes.

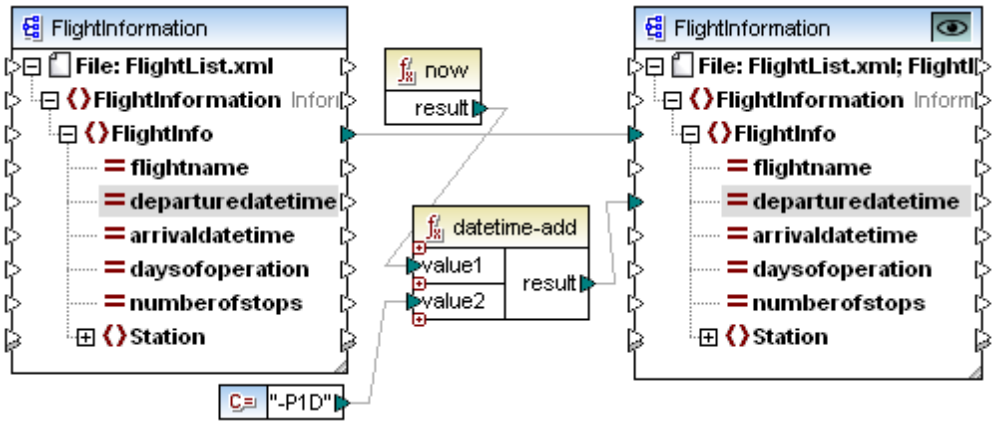
The example shown below, adds 10 days to the departuredatetime, i.e. P10D.



E.g. Instance document datetime:
departuredatetime="2001-12-**17**T09:30:02+05:00"

Result:
departuredatetime="2001-12-**27**T09:30:02+05:00"

To extract yesterdays date from dateTime input:
Use the "now" function to input the current date/time including timezone. A period can be made negative by using the minus character before the P designator, e.g. -P1D (minus 1 day).

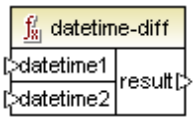


E.g. datetime now is 28th Feb 2012, 17:19:54.748(millisec)+01timezone.
now="2012-02-**28**T17:19:54.748+01:00"

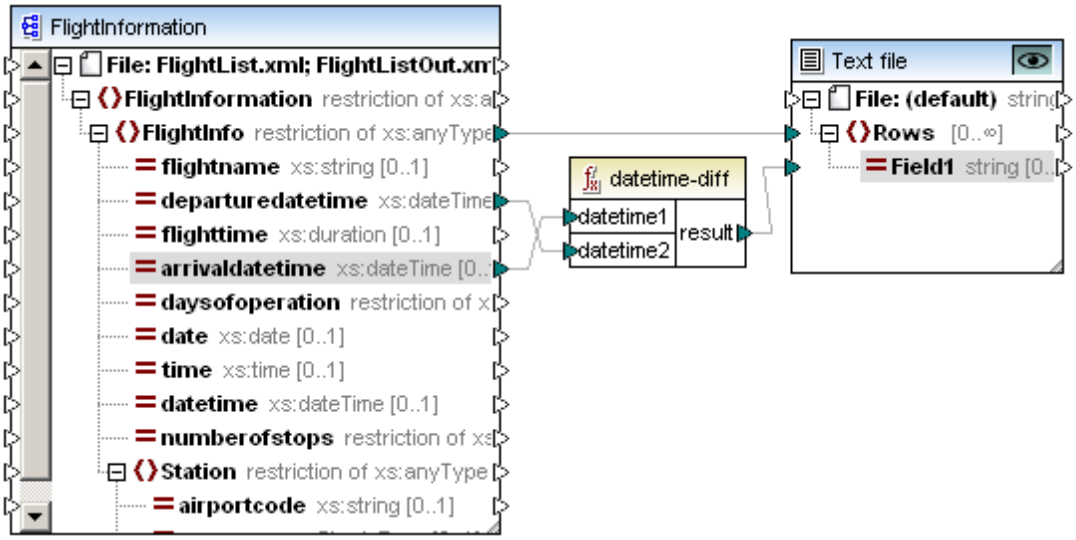
Result:
departuredatetime="2012-02-**27**T17:19:54.748+01:00"
i.e. 27th Feb 2012, 17:19:54.748(millisec)+01timezone

8.9.13.5 datetime-diff

Result is the duration obtained by subtracting datetime2 (second argument) from datetime1 (first argument). The result can be mapped to a string, or duration, datatype.



Note that the arrivaldatetime has been connected to datetime1 and departuredatetime to datetime2.



E.g. We want to find the difference, as a duration, between the departure and arrival times.

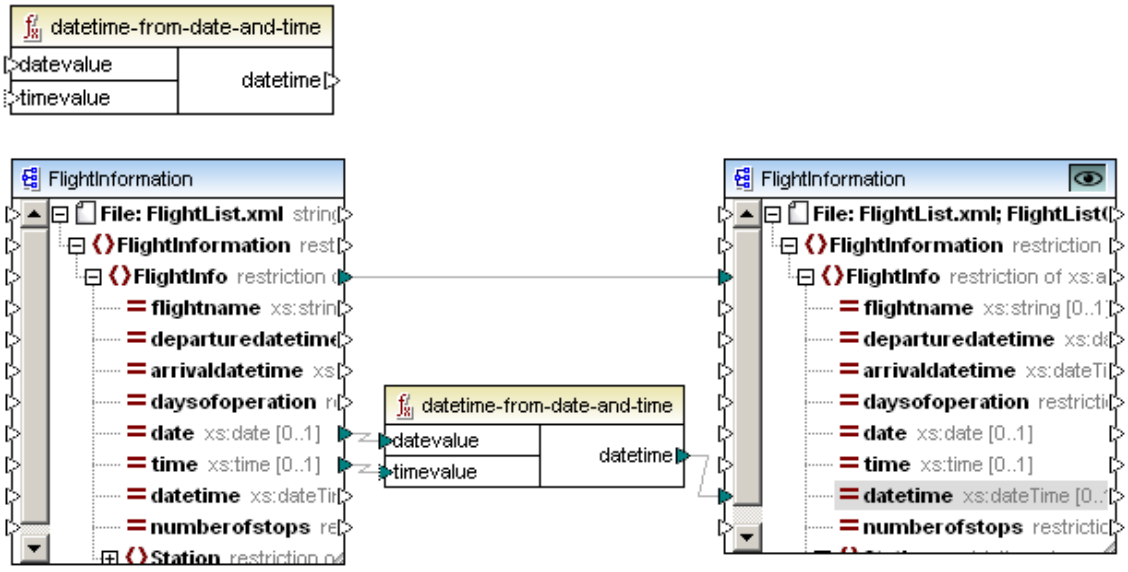
```
datetime1 arrivaldatetime="2001-12-17T19:30:02+05:00"
datetime2 departuredatetime="2001-12-17T09:30:02+05:00"
```

Result: the difference between the two is 10 hours:

```
result= PT10H
```

8.9.13.6 datetime-from-date-and-time

Result is a datetime built from a datevalue (first argument) and a timevalue (second argument). The first argument must be of type xs:date and the second xs:time. The result can be mapped to a sting or dateTime datatype.



E.g.

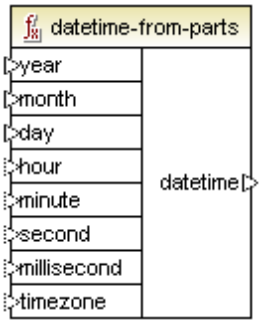
```
date="2012-06-29"
time="11:59:55"
```

Result:

```
dateTime="2012-06-29T11:59:55"
```

8.9.13.7 datetime-from-parts

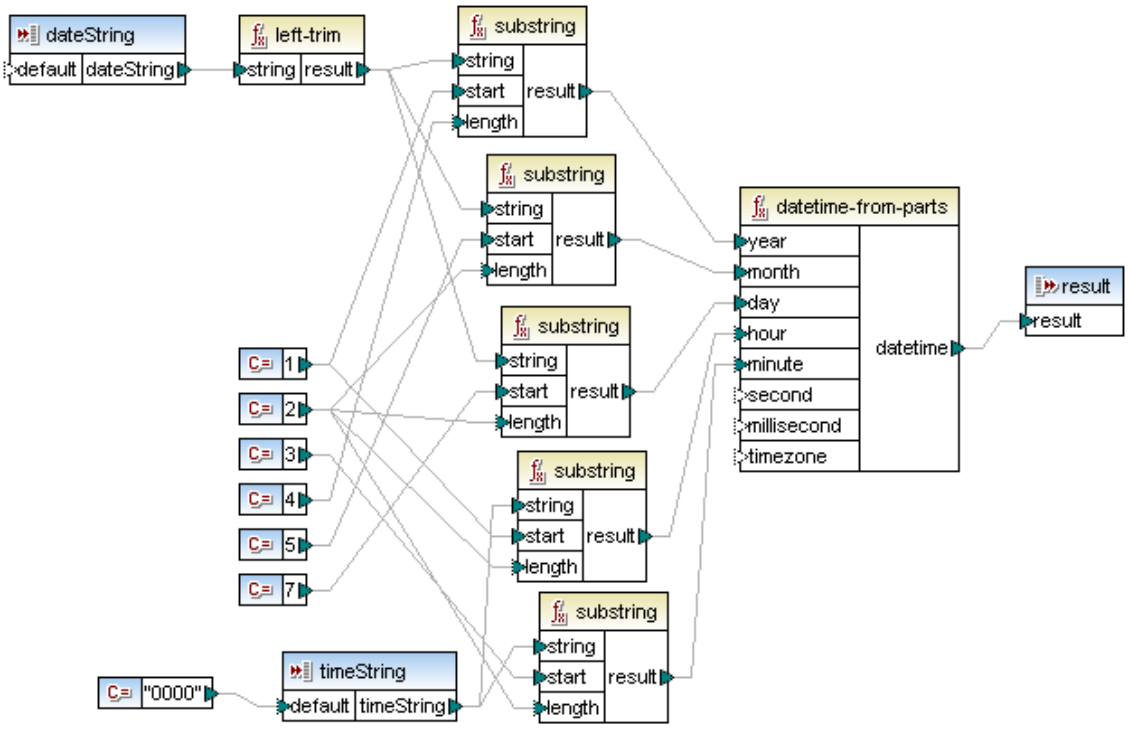
Result is a datetime built from any combination of the following parts as arguments: year, month, day, hour, minute, second, millisecond, and timezone. This function automatically normalizes the supplied parameters e.g. 32nd of January will automatically be changed to 1st February.



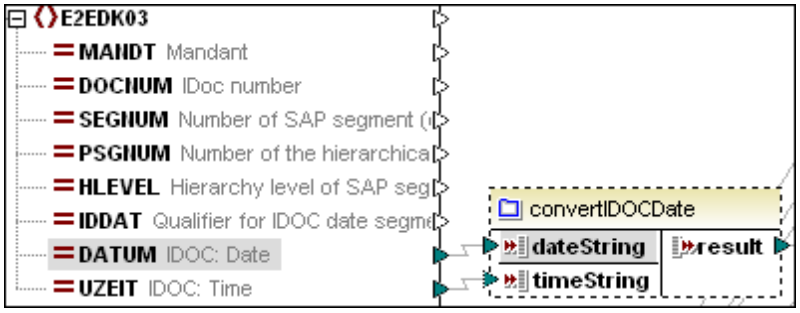
All of the arguments are of type xs:int except for millisecond, which is of type xs:decimal. The datetime result parameter is of type xs:dateTime.

The screenshot shown below is of the user-defined function "convertIDOCDate" available in the IDoc_Order.mfd mapping in the ...\MapForceExamples folder. It assembles the datetime from an input string using left-trim and substring functions.

Note that year, month, and day are mandatory parameters, the rest are optional.



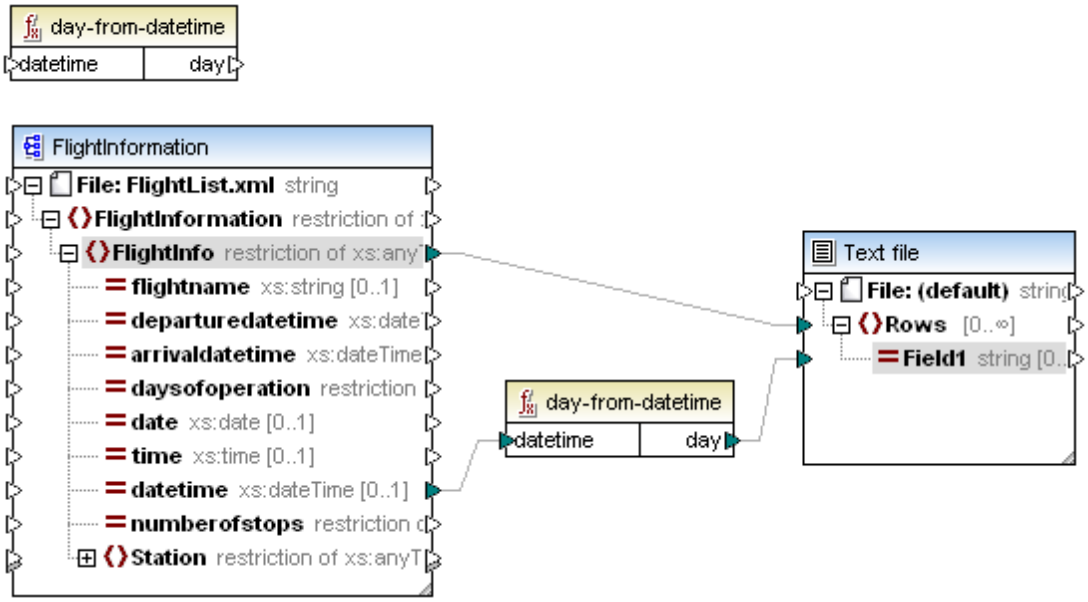
The date and time fields are supplied by the IDOC instance file:




```
IDOC:Date    19990621
ICOC:Time    0930
Result       1999-06-21T09:30:00
```

8.9.13.8 day-from-datetime

Result is the day from the datetime argument.

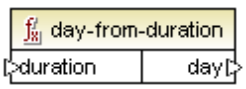


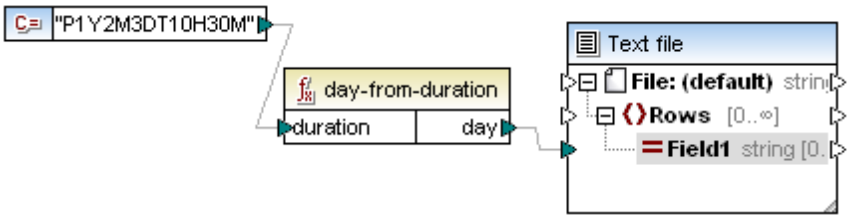
E.g.
`datetime="2001-12-17T10:30:03+01:00"`

Result: `17`

8.9.13.9 day-from-duration

Result is the day from the duration argument.

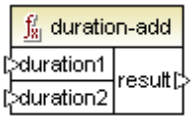




E.g.
`duration="P1Y2M3DT10H30M"`
Result: `3`

8.9.13.10 duration-add

Result is the duration obtained by adding two durations.



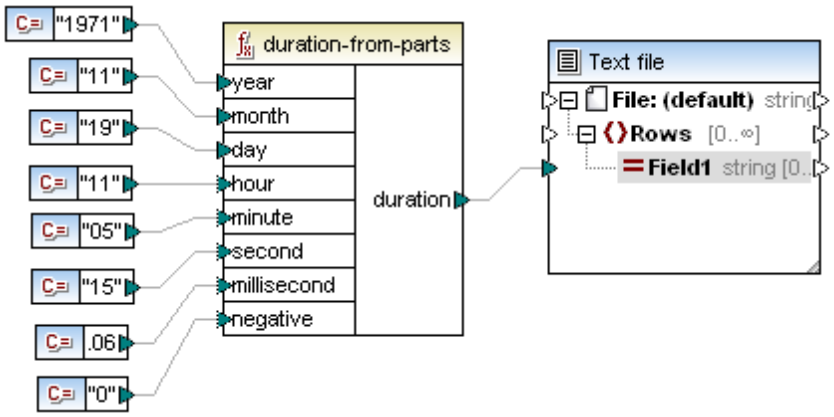
E.g.
`duration1="P0Y0M3DT03H0M" (3days 3 hours)`
`duration2="P0Y0M3DT01H0M" (3days 1 hour)`
`Result: P6DT4H (6days 4 hours)`

8.9.13.11 duration-from-parts

Result is a duration calculated by combining the following parts supplied as arguments: year, month, day, hour, minute, second, millisecond, negative.

Durations are in the form P1Y2M3DT04H05M06.07S i.e. P(eriod) 1 Year, 2 Months, 3 Days, T(ime designator), 04 Hours, 05 Minutes, 06.07 seconds.milliseconds.

All of the arguments are of type xs:int except for millisecond, which is of type xs:decimal, and negative, which is of type xs:boolean (i.e. 1 for true, 0 for false). The duration parameter is of type xs:duration.

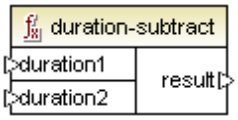


Parts: 1971 year, 11 month, 19 day, 11 hour, 05 minutes, 15.06 seconds, negative period "false".

Result:
`duration="P1971Y11M19DT11H5M15.00006S"`

8.9.13.12 duration-subtract

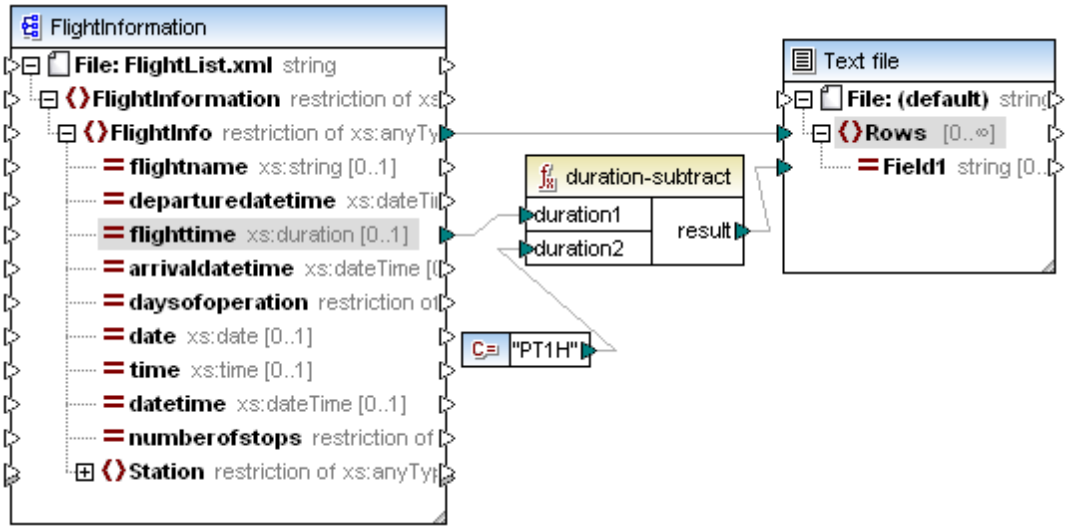
Result is the duration obtained by subtracting duration2 from duration1.



Durations must be entered in the form: P1Y2M3DT04H05M. Periods can be made negative by using the minus character before the P designator, e.g. -P1D.

P is the period designator, and is mandatory; the rest of period is therefore:
1 Year, 2 Months, 3 Days T(ime designator), 04 Hours, 05 Minutes.

The example shown below, subtracts 1 hour from flighttime, i.e. PT1H.

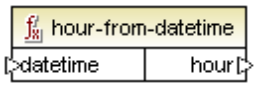


E.g.
```
duration1="P0Y0M0DT05H07M"
duration2="PT1H"
```

Result: `PT4H7M`

8.9.13.13 hour-from-datetime

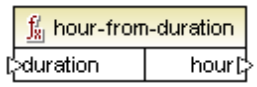
Result is the hour part of the datetime argument.



E.g.
```
datetime="2001-12-17T09:30:02+05:00"
hour= 9
```

8.9.13.14 hour-from-duration

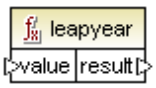
Result is the hour component of the duration argument.

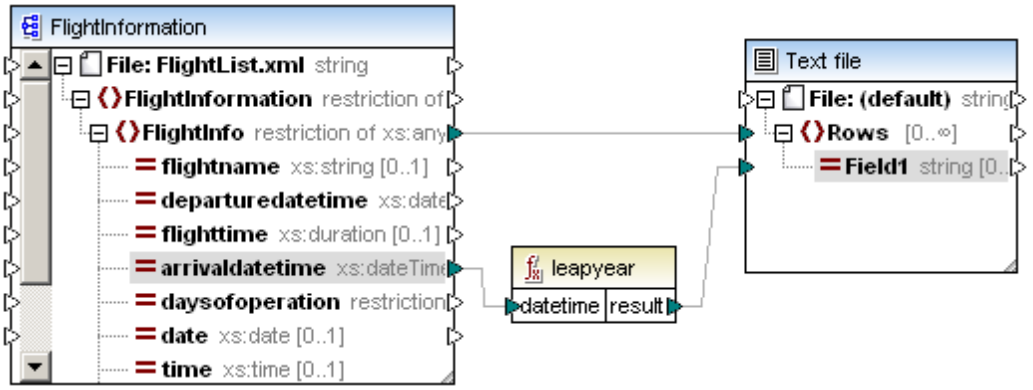


E.g.
```
duration="P0Y0M0DT05H07M"
hour= 5
```

8.9.13.15 leapyear

Result is true or false depending on whether the year of the supplied dateTime is in a leap year.

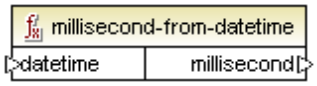




E.g.
```
arrivaldatetime="2001-12-17T19:30:02+05:00"
result="false"
```

8.9.13.16 millisecond-from-datetime

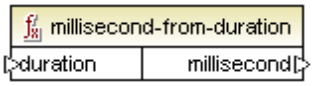
Result is the millisecond part of the datetime argument.



E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
millisecond= 544
```

8.9.13.17 millisecond-from-duration

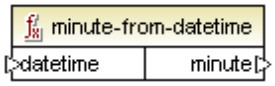
Result is the millisecond component of the duration argument.



E.g.
```
duration="P0Y0M0DT05H07M02.227S"
millisecond= 227
```

8.9.13.18 minute-from-datetime

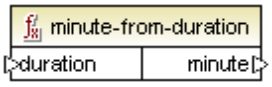
Result is the minute part of the datetime argument.



E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
minute= 30
```

8.9.13.19 minute-from-duration

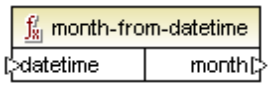
Result is the minute component of the duration argument.



E.g.
```
duration="P0Y0M0DT05H07M02.227S"
minute= 7
```

8.9.13.20 month-from-datetime

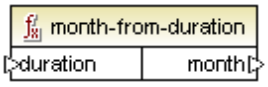
Result is the month part of the dateTime argument.



E.g.
```
datetime="2001-12-17T09:30:02.544+05:00"
month= 12
```

8.9.13.21 month-from-duration

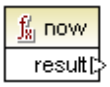
Result is the month component of the duration argument.



E.g.
```
duration="P0Y04M0DT05H07M02.227S"
month= 4
```

8.9.13.22 now

Result is the current dateTime (including timezone).



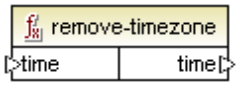
E.g.

```
result= 2012-03-06T14:44:57.567+01:00
```

For an example on how to extract yesterday's date, see the core | lang | datetime-add function.

8.9.13.23 remove-timezone

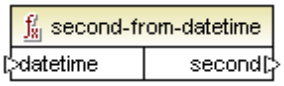
Removes the timezone component, e.g. +5:00, from the time input parameter.



E.g.
departuredatetime="2001-12-17T09:30:02**+05:00**"
time: 2001-12-17T09:30:02

8.9.13.24 second-from-datetime

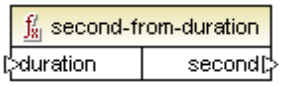
Result is the seconds part of the dateTime argument.



E.g.
datetime="2001-12-17T09:30:**02**.544+05:00"
second= 2

8.9.13.25 second-from-duration

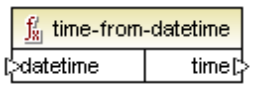
Result is the seconds component of the duration argument.



E.g.
duration="P0Y04M0DT05H07M**02**.227S"
second= 2

8.9.13.26 time-from-datetime

Result is the time part of the dateTime argument.

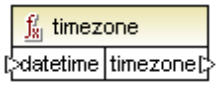


E.g.
datetime="2001-12-17T09:30:02.544+05:00"


```
time= 09:31:02+05:00
```

8.9.13.27 timezone

Returns the timezone (i.e. +05:00 here) relative to UTC of the dateTime value. NB timezone unit is minutes.



E.g.

```
datetime="2001-12-17T09:30:02.544+05:00"
timezone= 300
```

8.9.13.28 weekday

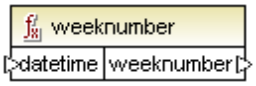
Returns the weekday of the dateTime value, starting with `Monday=1` to `Sunday=7`.

E.g.

```
datetime="2001-12-17T09:30:02.544+05:00"
weekday= 1
```

8.9.13.29 weeknumber

Returns the week number within the year specified by the dateTime value.

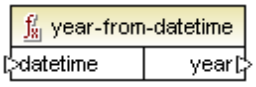


E.g.

```
datetime="2001-12-17T09:30:02.544+05:00"
weeknumber= 51
```

8.9.13.30 year-from-datetime

Result is the year part of the dateTime argument.



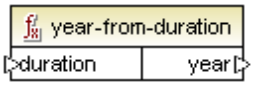
E.g.

```
datetime="2001-12-17T09:30:02.544+05:00"
```

```
year= 2001
```

8.9.13.31 year-from-duration

Result is the year component of the duration argument.



E.g.

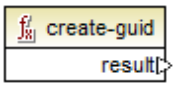
```
duration="P01Y04M0DT05H07M02.227S"
year= 1
```

8.9.14 lang | generator functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages. The generator functions generate values for database fields, which do not have any input data from the Schema, database or EDI source component.

8.9.14.1 create-guid

Result is a globally-unique identifier (as a hex-encoded string) for the specific field.

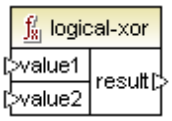


8.9.15 lang | logical functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

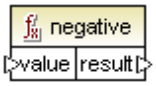
8.9.15.1 logical-xor

Result is true if value1 is different than value2, otherwise false.



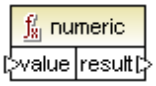
8.9.15.2 negative

Result is true if value is negative, i.e. less than zero, otherwise false.



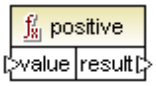
8.9.15.3 numeric

Result is true if value is a number, otherwise false. The input will usually be a string.



8.9.15.4 positive

Result is true if value is positive, i.e. equal to or greater than zero, otherwise false.

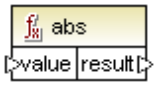


8.9.16 lang | math functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

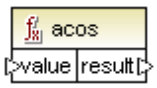
8.9.16.1 abs

Result is the absolute value of the input **value**.



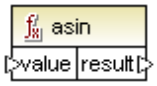
8.9.16.2 acos

Result is the arc cosine of **value**.



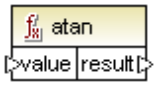
8.9.16.3 asin

Result is the arc sine of **value**.



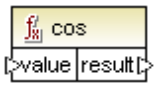
8.9.16.4 atan

Result is the arc tangent of **value**.



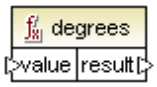
8.9.16.5 cos

Result is the cosine of **value**.



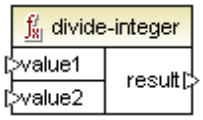
8.9.16.6 degrees

Result is the conversion of **value** in radians into degrees.



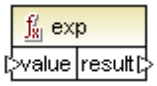
8.9.16.7 divide-integer

Result is the integer result of dividing **value1** by **value2**. E.g. 15 divide-integer 2, integer result is 7.



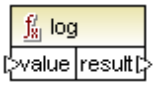
8.9.16.8 exp

Result is **e** (base natural logarithm) raised to the **value**th power.



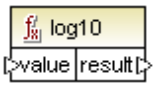
8.9.16.9 log

Result is the natural logarithm of **value**.



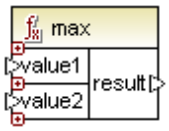
8.9.16.10 log10

Result is logarithm (base 10) of **value**.



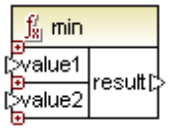
8.9.16.11 max

Result is the numerically larger value of **value1** compared to **value2**.



8.9.16.12 min

Result is the numerically smaller value of **value1** compared to **value2**.



8.9.16.13 pi

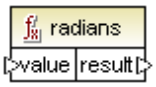
Result is the value of pi.

8.9.16.14 pow

Result is the value of **a** raised to the power **bth power**.

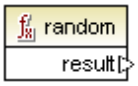
8.9.16.15 radians

Result is the conversion of **value** in degrees to radians.



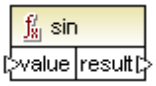
8.9.16.16 random

Result is a pseudorandom value between 0.0 and 1.0



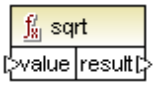
8.9.16.17 sin

Result is the sine of **value**.



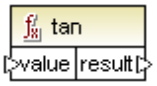
8.9.16.18 sqrt

Result is the square root of **value**.



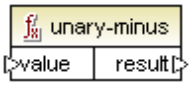
8.9.16.19 tan

Result is the tangent of **value**.



8.9.16.20 unary-minus

Result is the negation of the signed input **value**. E.g. +3 result is -3, while -3 result is 3.

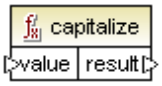


8.9.17 lang | string functions

The lang library contains functions that are available when selecting either Java, C#, or C++ languages.

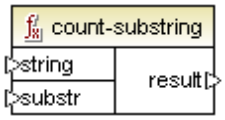
8.9.17.1 capitalize

Result is the input string **value,** where the first letter of each word is capitalized (initial caps).



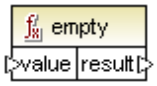
8.9.17.2 count-substring

Result is the number of times that **substr** occurs in **string**.



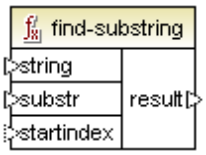
8.9.17.3 empty

Result is true if the input string **value** is empty, otherwise false.



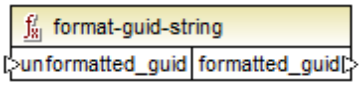
8.9.17.4 find-substring

Returns the position of the first occurence of substr. within string, starting at position startindex. The first character has position 1. If the substring could not be found, then the result is 0.



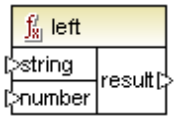
8.9.17.5 format-guid-string

Result is a correctly formatted GUID string **formatted_guid,** using **unformatted_guid** as the input string, for use in database fields. See also the create-guid function in the **lang | generator functions** library.



8.9.17.6 left

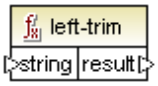
Result is a string containing the first **number** characters of **string**.



E.g. string="This is a sentence" and number=4, result is "This".

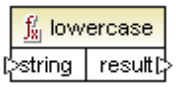
8.9.17.7 left-trim

Result is the input string with all leading whitespace characters removed.



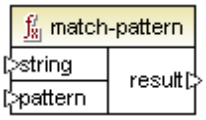
8.9.17.8 lowercase

Result is the lowercase version of the input **string**. For Unicode characters the corresponding lower-case characters (defined by the Unicode consortium) are used.



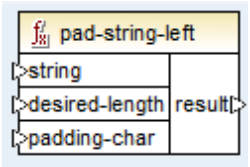
8.9.17.9 match-pattern

Result is true if the input **string** matches the regular expression defined by **pattern**, else false. The specific regular expression syntax depends on the target language (see also Regular expressions).



8.9.17.10 pad-string-left

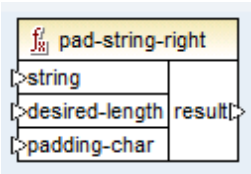
Returns a string which is padded to the left by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.



string	Specifies the input string.
desired-length	Defines the desired length of the string after padding.
padding-char	Defines the character to use as padding character.

8.9.17.11 pad-string-right

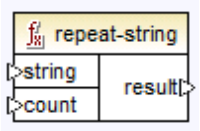
Returns a string which is padded to the right by a single specific character, up to a required length. The desired string length and the padding character are supplied as arguments.



string	Specifies the input string.
desired-length	Defines the desired length of the string after padding.
padding-char	Defines the character to use as padding character.

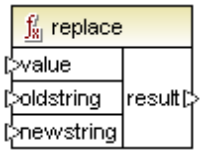
8.9.17.12 repeat-string

Repeats the string supplied as argument *n* times. The **count** argument specifies the number of times to repeat the string.



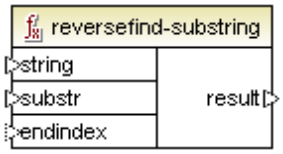
8.9.17.13 replace

Result is a new string where each instance of **oldstring**, in the input string **value**, is replaced by **newstring**.



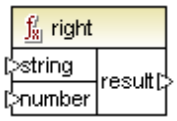
8.9.17.14 reversefind-substring

Returns the position of the first occurence of **substr**. within **string**, starting at position **endindex**, i.e. from right to left. The first character has position 1. If the substring could not be found, then the result is 0.



8.9.17.15 right

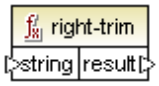
Result is a string containing the last **number** characters of **string.**



E.g. string="This is a sentence" and number=5, result is "tence".

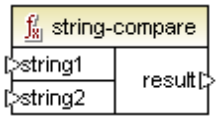
8.9.17.16 right-trim

Result is the input string with all trailing whitespace characters removed.



8.9.17.17 string-compare

Returns the result of a string comparison of **string1** with **string2** taking case into account. If string1=string2 then result is 0.

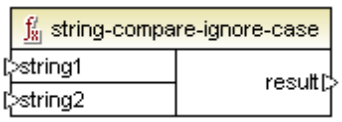


If string1 is smaller than string2 then result is < 0.

If string1 is larger than string2 then result is > 0

8.9.17.18 string-compare-ignore-case

Returns the result of a string comparison of string1 with string2 ignoring case. If string1=string2 then result is 0.

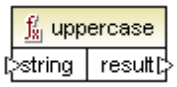


If string1 is smaller than string2 then result is < 0.

If string1 is larger than string2 then result is > 0.

8.9.17.19 uppercase

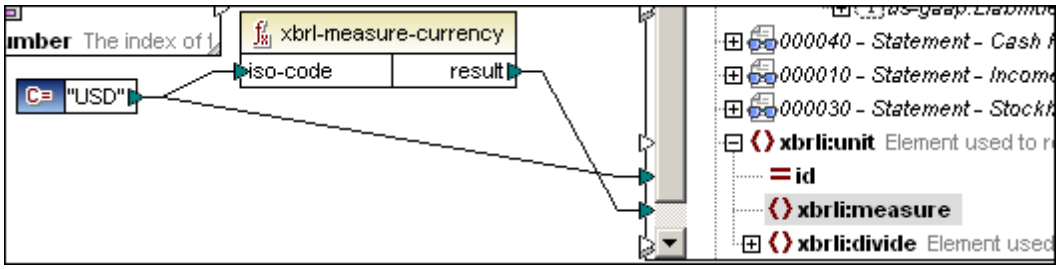
Result is the string input converted into uppercase. For Unicode characters the corresponding upper-case characters (defined by the Unicode consortium) are used.



8.9.18 xbrl

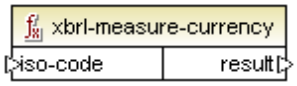
The XBRL library contains functions that convert data into the QName format necessary for XBRL instance/taxonomy files.

The xbrl:ID and xbrl:measure child elements of the xbrl:unit element are mandatory in an XBRL instance file, and must be mapped for the mapping to be valid.



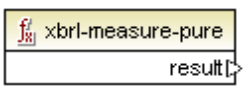
8.9.18.1 xbrl-measure-currency

Result is the QName for items of the monetaryItemType from the ISO 4217 currency code.



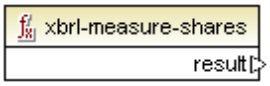
8.9.18.2 xbrl-measure-pure

Result is the QName for items of rates, percentages or ratios.



8.9.18.3 xbrl-measure-shares

Result is the QName for items of sharesItemType.



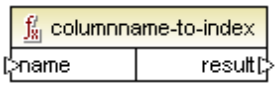
8.9.19 xlsx

The XLSX library contains functions that convert data to/from Excel date/time formats.

When generating code for C#, note that these functions depend on support for Excel components which requires .NET 3.0 or later, so Visual Studio 2005 / 2008 or 2010 must be selected in the **Tools | Options | Generation | C# settings** group for these functions to be available.

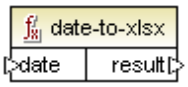
8.9.19.1 columnname-to-index

Returns the index of the column with the given name. The name of column 1 is "A".



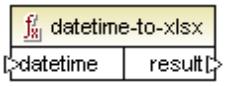
8.9.19.2 date-to-xlsx

Returns the Excel representation of *date* / *time* / *datetime* value extracted from the source.



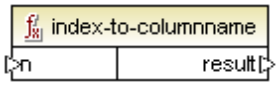
8.9.19.3 datetime-to-xslx

Returns the Excel representation of the *datetime* value extracted from the source.



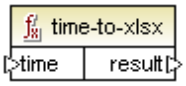
8.9.19.4 index-to-columnname

Returns the name of column *n*. The name of column 1 is "A".



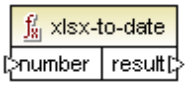
8.9.19.5 time-to-xlsx

Returns the Excel representation of the *time* value extracted from the source.



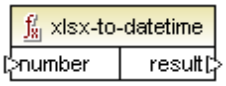
8.9.19.6 xlsx-to-date

Returns the *date* value extracted from the Excel representation.



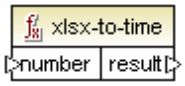
8.9.19.7 xlsx-to-datetime

Returns the *datetime* value extracted from the Excel representation.



8.9.19.8 xlsx-to-time

Returns the *time* value extracted from the Excel representation.



8.9.20 xpath2 | accessors

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

8.9.20.1 base-uri

The `base-uri` function takes a node argument as input, and returns the URI of the XML resource containing the node. The output is of type `xs:string`. MapForce returns an error if no input node is supplied.

8.9.20.2 node-name

The `node-name` function takes a node as its input argument and returns its QName. When the QName is represented as a string, it takes the form of `prefix:localname` if the node has a prefix, or `localname` if the node has no prefix. To obtain the namespace URI of a node, use the `namespace-URI-from-QName` function (in the library of QName-related functions).

8.9.20.3 string

The `string` function works like the `xs:string` constructor: it converts its argument to `xs:string`.

When the input argument is a value of an atomic type (for example `xs:decimal`), this atomic value is converted to a value of `xs:string` type. If the input argument is a node, the string value of the node is extracted. (The string value of a node is a concatenation of the values of the node's descendant nodes.)

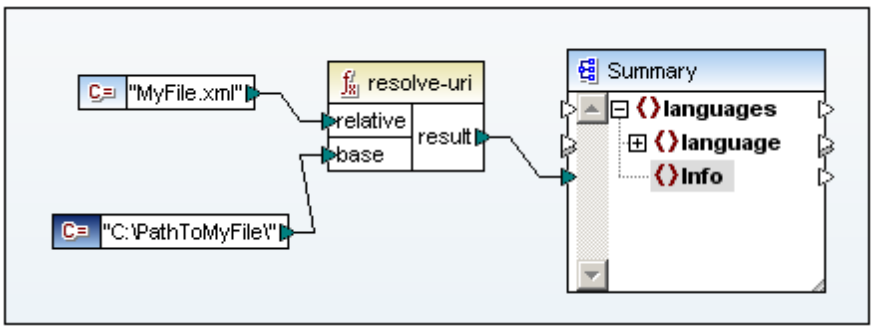
8.9.21 xpath2 | anyURI functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

8.9.21.1 resolve-uri

The `resolve-uri` function takes a URI as its first argument (datatype `xs:string`) and resolves it against the URI in the second argument (datatype `xs:string`).

The result (datatype `xs:string`) is a combined URI. In this way a relative URI (the first argument) can be converted to an absolute URI by resolving it against a base URI.



In the screenshot above, the first argument provides the relative URI, the second argument the

base URI. The resolved URI will be a concatenation of base URI and relative URI, so `C:\PathtoMyFile\MyFile.xml`.

Note: Both arguments are of datatype `xs:string` and the process of combining is done by treating both inputs as strings. So there is no way of checking whether the resources identified by these URIs actually exist. MapForce returns an error if the second argument is not supplied.

8.9.22 xpath2 | boolean functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.
The Boolean functions `true` and `false` take no argument and return the boolean constant values, `true` and `false`, respectively. They can be used where a constant boolean value is required.

8.9.22.1 false

Returns the Boolean value "false".

8.9.22.2 true

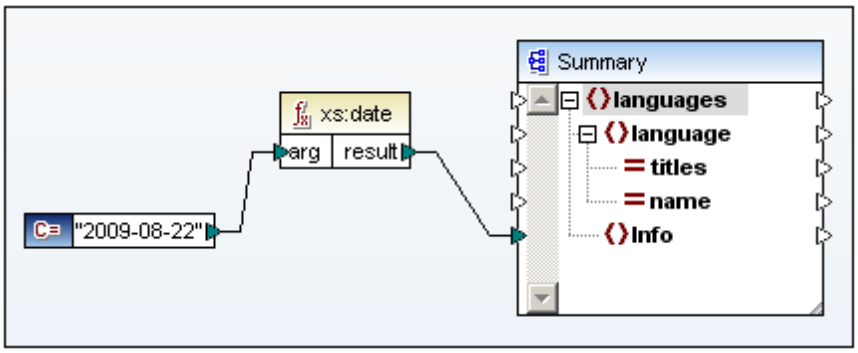
Returns the Boolean value "true".

8.9.23 xpath2 | constructors

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The functions in the Constructors part of the XPath 2.0 functions library construct specific datatypes from the input text. Typically, the lexical format of the input text must be that expected of the datatype to be constructed. Otherwise, the transformation will not be successful.

For example, if you wish to construct an `xs:date` datatype, use the `xs:date` constructor function. The input text must have the lexical format of the `xs:date` datatype, which is: `YYYY-MM-DD` (*screenshot below*).



In the screenshot above, a string constant (`2009-08-22`) has been used to provide the input argument of the function. The input could also have been obtained from a node in the source

document.

The xs:date function returns the input text (2009-08-22), which is of xs:string datatype (specified in the *Constant* component), as output of xs:date datatype.

When you mouseover the input argument in a function box, the expected datatype of the argument is displayed in a popup.

8.9.24 xpath2 | context functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The Context functions library contains functions that provide the current date and time, the default collation used by the processor, and the size of the current sequence and the position of the current node.

8.9.24.1 current-date

Returns the current date (xs:date) from the system clock.

8.9.24.2 current-dateTime

Returns the current date and time (xs:dateTime) from the system clock.

8.9.24.3 current-time

Returns the current time (xs:time) from the system clock.

8.9.24.4 default-collation

The default-collation function takes no argument and returns the default collation, that is, the collation that is used when no collation is specified for a function where one can be specified.

The Altova XSLT 2.0 Engine supports the Unicode codepoint collation only. Comparisons, including for the fn:max and fn:min functions, are based on this collation.

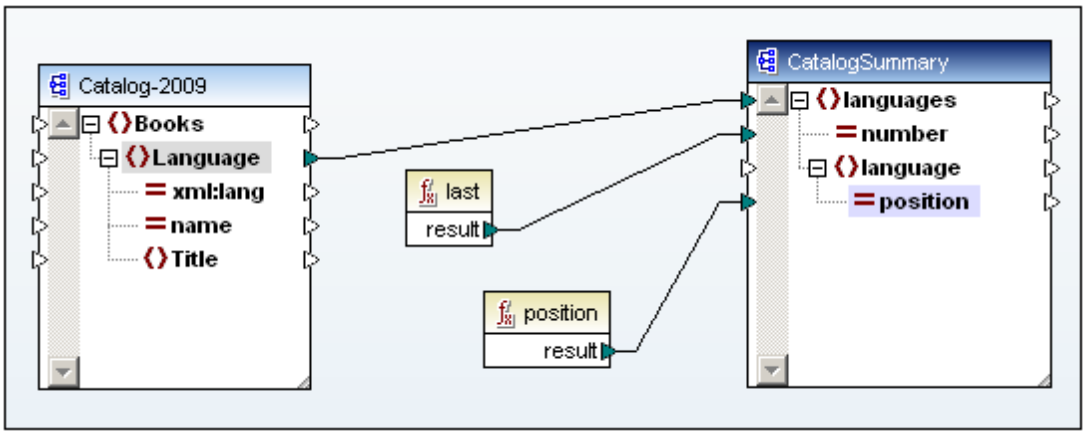
8.9.24.5 implicit-timezone

Returns the value of the "implicit timezone" property from the evaluation context.

8.9.24.6 last

The last and position functions take no argument. The last function returns the position of the last node in the context nodeset. The position function returns the position of the current node in the nodeset being processed.

The context nodeset at the nodes where the functions are directed, is the nodeset to which the functions will apply. In the screenshot below, the nodeset of `Language` elements is the context nodeset for the `last` and `position` functions.



In the example above, the `last` function returns the position of the last node of the context nodeset (the nodeset of `Language` elements) as the value of the `number` attribute. This value is also the size of the nodeset since it indicates the number of nodes in the nodeset.

The `position` function returns the position of the `Language` node being currently processed. For each `Language` element node, its position within the nodeset of `Langauge` elements is output to the `language/@position` attribute node.

We would advise you to use the **position** and **count** functions from the **core** library.

8.9.25 xpath2 | durations, date and time functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The XPath 2 duration and date and time functions enable you to adjust dates and times for the timezone, extract particular components from date-time data, and subtract one date-time unit from another.

The 'Adjust-to-Timezone' functions

Each of these related functions takes a date, time, or dateTime as the first argument and adjusts the input by adding, removing, or modifying the timezone component depending on the value of the second argument.

The following situations are possible when the first argument contains no timezone (for example, the date `2009-01` or the time `14:00:00`).

- Timezone argument (the second argument of the function) is present: The result will contain the timezone specified in the second argument. The timezone in the second argument is added.
- Timezone argument (the second argument of the function) is absent: The result will contain the implicit timezone, which is the system's timezone. The system's timezone is added.
- Timezone argument (the second argument of the function) is empty: The result will

contain no timezone.

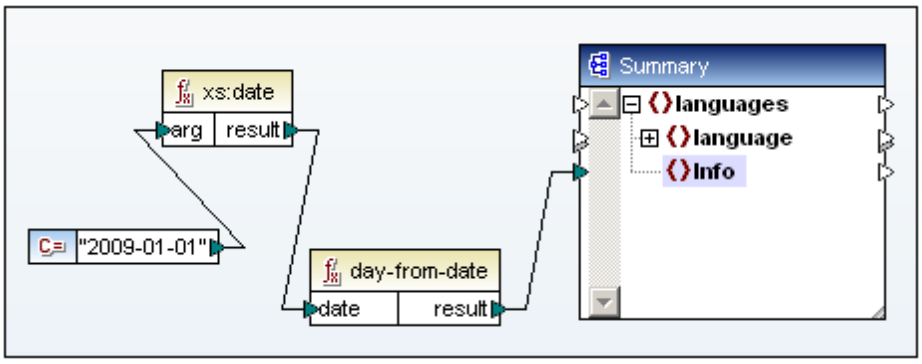
The following situations are possible when the first argument contains a timezone (for example, the date `2009-01-01+01:00` or the time `14:00:00+01:00`).

- Timezone argument (the second argument of the function) is present: The result will contain the timezone specified in the second argument. The original timezone is replaced by the timezone in the second argument.
- Timezone argument (the second argument of the function) is absent: The result will contain the implicit timezone, which is the system's timezone. The original timezone is replaced by the system's timezone.
- Timezone argument (the second argument of the function) is empty: The result will contain no timezone.

The 'From' functions

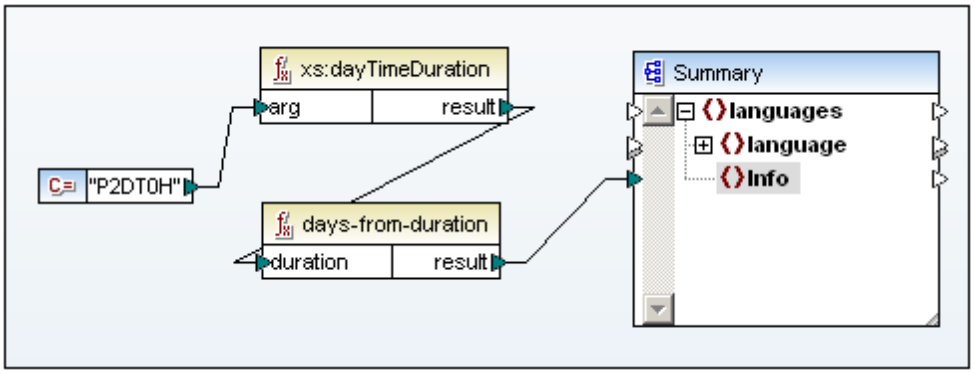
Each of the 'From' functions extracts a particular component from: (i) date or time data, and (ii) duration data. The results are of the xs:decimal datatype.

As an example of extracting a component from date or time data, consider the `day-from-date` function (*screenshot below*).



The input argument is a date (`2009-01-01`) of type `xs:date`. The `day-from-date` function extracts the day component of the date (`1`) as an xs:decimal datatype.

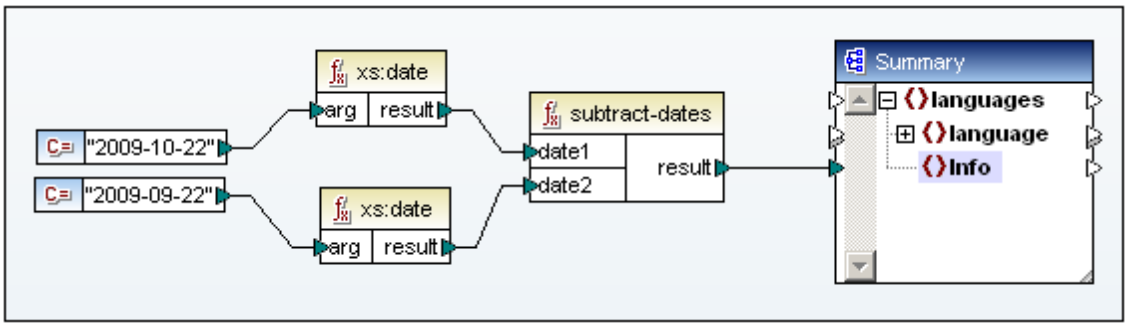
Extraction of time components from durations requires that the duration be specified either as `xs:yearMonthDuration` (for extracting years and months) or `xs:dayTimeDuration` (for extracting days, hours, minutes, and seconds). The result will be of type `xs:decimal`. The screenshot below shows a `dayTimeDuration` of `P2DT0H` being input to the `days-from-duration` function. The result is the xs:decimal `2`.



The 'Subtract' functions

Each of the three subtraction functions enables you to subtract one time value from another and return a duration value. The three subtraction functions are: `subtract-dates`, `subtract-times`, `subtract-dateTimes`.

The screenshot below shows how the subtract-dates function is used to subtract two dates (`2009-10-22` minus `2009-09-22`). The result is the dayTimeDuration `P30D`.

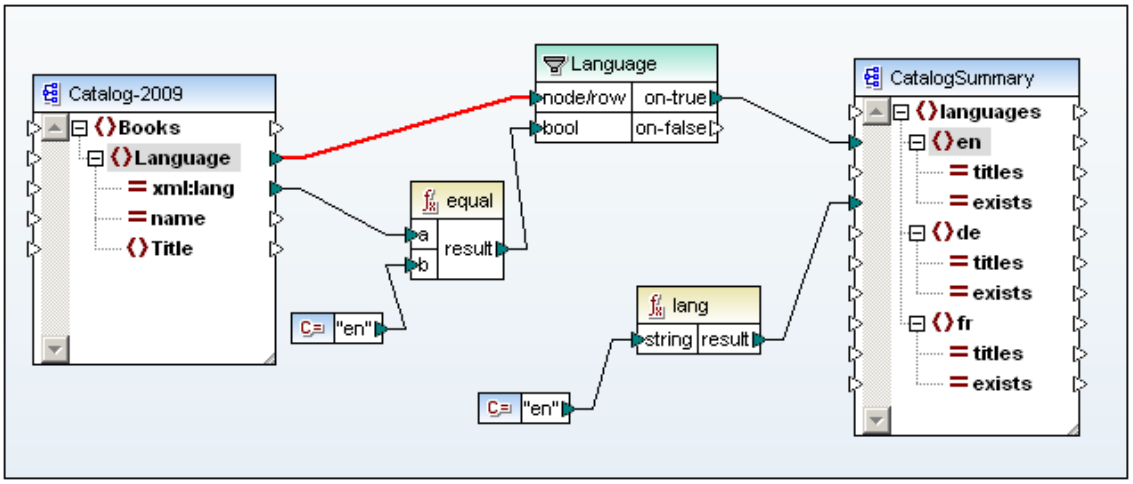


8.9.26 xpath2 | node functions

The following XPath 2 node functions are available:

lang

The `lang` function takes a string argument that identifies a language code (such as `en`). The function returns `true` or `false` depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function.



In the screenshot above notice the following:

1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. `Language` nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the point where the `en` element is created in the output document.
4. The output of the `lang` function (`true` or `false`) is sent to the `en/@exists` attribute node of the output. The argument of the function is provided by the string constant `en`. The `lang` function then checks whether the context node at this point (the `Language` element) has an `xml:lang` attribute with a value of `en` (the argument of the function). If yes, then `true` is returned, otherwise `false`.

local-name, name, namespace-uri

The `local-name`, `name`, and `namespace-uri` functions, return, respectively, the local-name, name, and namespace URI of the input node. For example, for the node `altova:Products`, the local-name is `Products`, the name is `altova:Products`, and the namespace URI is the URI of the namespace to which the `altova:` prefix is bound (say, `http://www.altova.com/examples`).

Each of these three functions has two variants:

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the `lang` function above).
- An argument that must be a node: the function is applied to the submitted node.

The output of each of these six variants is a string.

number

Converts an input string into a number. Also converts a boolean input to a number.

The number function takes a node as input, atomizes the node (that is, extracts its contents), and converts the value to a decimal and returns the converted value. The only types that can be converted to numbers are booleans, strings, and other numeric types. Non-numeric input values (such as a non-numeric string) result in `NaN` (Not a Number).

There are two variants of the number function:

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the `lang` function above).
- An argument that must be a node: the function is applied to the submitted node.

8.9.27 xpath2 | numeric functions

The following XPath 2 numeric functions are available:

abs

The `abs` function takes a numeric value as input and returns its absolute value as a decimal. For example, if the input argument is `-2` or `+2`, the function returns `2`.

round-half-to-even

The `round-half-to-even` function rounds the supplied number (first argument) to the degree of precision (number of decimal places) supplied in the optional second argument. For example, if the first argument is `2.141567` and the second argument is `3`, then the first argument (the number) is rounded to three decimal places, so the result will be `2.141`. If no precision (second argument) is supplied, the number is rounded to zero decimal places, that is, to an integer.

The 'even' in the name of the function refers to the rounding to an even number when a digit in the supplied number is midway between two values. For example, `round-half-to-even(3.475, 2)` would return `3.48`.

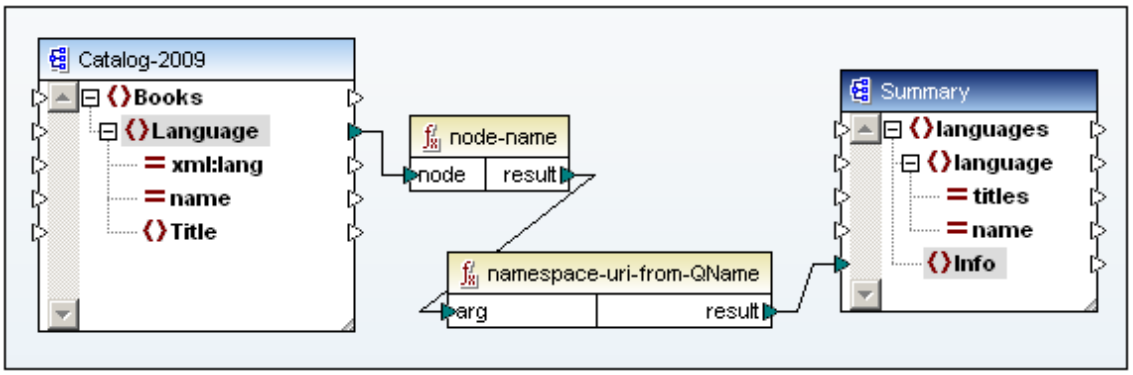
8.9.28 xpath2 | qname-related functions

There are two QName-related functions that work similarly: `local-name-from-QName` and `namespace-uri-from-QName`.

Both functions take an expanded QName (in the form of a string) as their input arguments and output, respectively, the local-name and namespace-uri part of the expanded QName.

The important point to note is that since the input of both functions are strings, a node cannot be connected directly to the input argument boxes of these functions.

The node should first be supplied to the `node-name` function, which outputs the expanded QName. This expanded QName can then be provided as the input to the two functions (*see screenshot below*).



The output of both functions is a string.

8.9.29 xpath2 | string functions

The following XPath 2 string functions are available:

compare

The `compare` function takes two strings as arguments and compares them for equality and alphabetically. If *String-1* is alphabetically less than *String-2* (for example the two string are: `A` and `B`), then the function returns `-1`. If the two strings are equal (for example, `A` and `A`), the function returns `0`. If *String-1* is greater than *String-2* (for example, `B` and `A`), then the function returns +1.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

ends-with

The `ends-with` function tests whether *String-1* ends with *String-2*. If yes, the function returns `true`, otherwise `false`.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

escape-uri

The `escape-uri` function takes a URI as input for the first string argument and applies the URI escaping conventions of RFC 2396 to the string. The second boolean argument (`escape-reserved`) should be set to `true()` if characters with a reserved meaning in URIs are to be escaped (for example "+" or "/").

For example:

`escape-uri("My A+B.doc", true())` would give `My%20A%2B.doc`
`escape-uri("My A+B.doc", false())` would give `My%20A+B.doc`

lower-case

The `lower-case` function takes a string as its argument and converts every upper-case character in the string to its corresponding lower-case character.

matches

The `matches` function tests whether a supplied string (the first argument) matches a regular expression (the second argument). The syntax of regular expressions must be that defined for the `pattern` facet of XML Schema. The function returns `true` if the string matches the regular expression, `false` otherwise.

The function takes an optional `flags` argument. Four flags are defined (`i`, `m`, `s`, `x`). Multiple flags

can be used: for example, `imx`. If no flag is used, the default values of all four flags are used.

The meaning of the four flags are as follows:

`i`	Use case-insensitive mode. The default is case-sensitive.
`m`	Use multiline mode, in which the input string is considered to have multiple lines, each separated by a newline character (`x0a`). The meta characters `^` and `$` indicate the beginning and end of each line. The default is string mode, in which the string starts and ends with the meta characters `^` and `$`.
`s`	Use dot-all mode. The default is not-dot-all mode, in which the meta character `"."` matches all characters except the newline character (`x0a`). In dot-all mode, the dot also matches the newline character.
`x`	Ignore whitespace. By default whitespace characters are not ignored.

normalize-unicode

The `normalize-unicode` function normalizes the input string (the first argument) according to the rules of the normalization form specified (the second argument). The normalization forms NFC, NFD, NFKC, and NFKD are supported.

replace

The `replace` function takes the string supplied in the first argument as input, looks for matches as specified in a regular expression (the second argument), and replaces the matches with the string in the third argument.

The rules for matching are as specified for the matches attribute above. The function also takes an optional `flags` argument. The flags are as described in the `matches` function above.

starts-with

The `starts-with` function tests whether *String-1* starts with *String-2*. If yes, the function returns `true`, otherwise `false`.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

substring-after

The substring-after function returns that part of *String-1* (the first argument) that occurs after the test string, *String-2* (the second argument). An optional third argument specifies the collation to use for the string comparison. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

substring-before

The substring-before function returns that part of *String-1* (the first argument) that occurs before the test string, *String-2* (the second argument). An optional third argument specifies the collation

to use for the string comparison. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

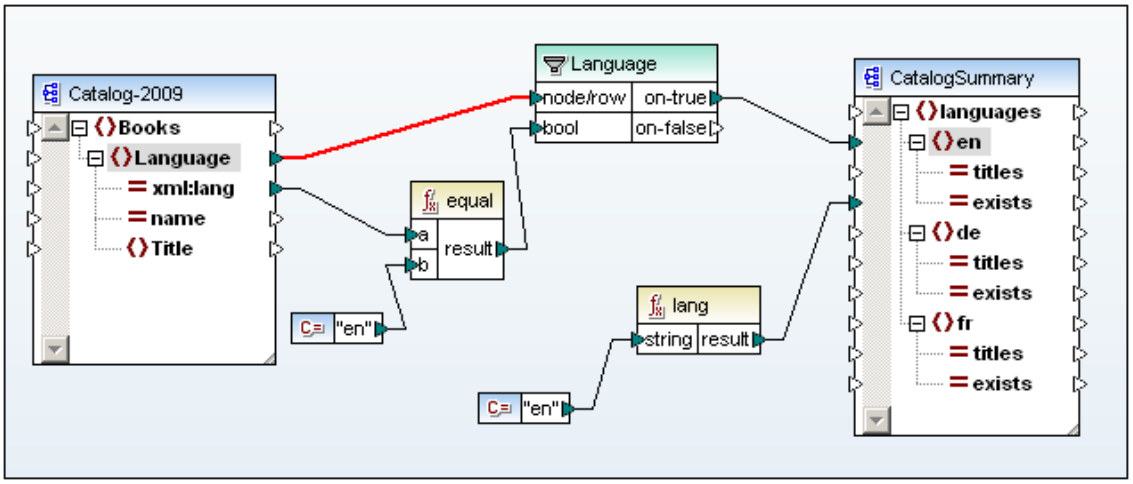
upper-case

The `upper-case` function takes a string as its argument and converts every lower-case character in the string to its corresponding upper-case character.

8.9.30 xslt | xpath functions

The functions in the XPath Functions library are XPath 1.0 nodeset functions. Each of these functions takes a node or nodeset as its context and returns information about that node or nodeset. These function typically have:

- a context node (in the screenshot below, the context node for the `lang` function is the Language element of the source schema).
- an input argument (in the screenshot below, the input argument for the `lang` function is the string constant `en`). The `last` and `position` functions take no argument.



lang

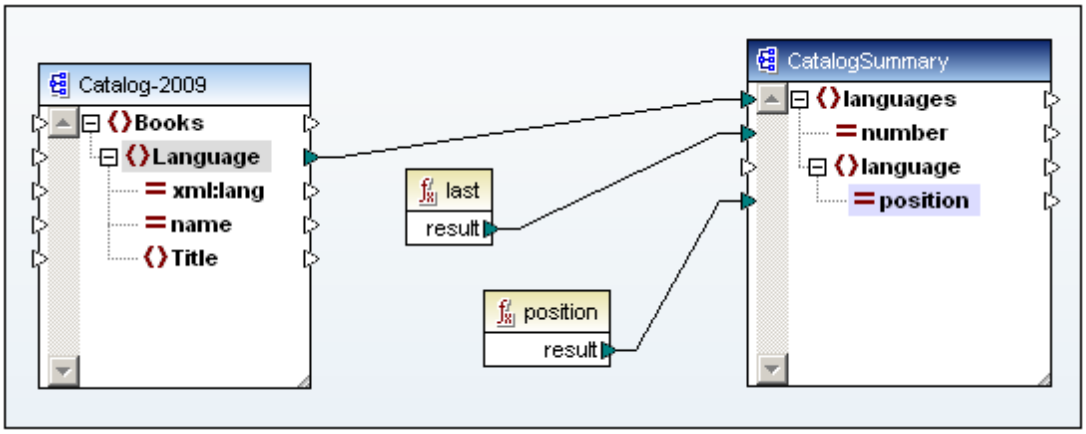
The `lang` function takes a string argument that identifies a language code (such as `en`). The function returns `true` or `false` depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function. In the screenshot above notice the following:

1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. `Language` nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the point where the `en` element is created in the output document.
4. The output of the `lang` function (`true` or `false`) is sent to the `en/@exists` attribute node of the output. The argument of the function is provided by the string constant `en`. The `lang` function then checks whether the context node at this point (the `Language` element) has an `xml:lang` attribute with a value of `en` (the argument of the function). If yes, then `true` is returned, otherwise `false`.

last, position

The `last` and `position` functions take no argument. The `last` function returns the position of the last node in the context nodeset. The `position` function returns the position of the current node in the nodeset being processed.

The context nodeset at the nodes where the functions are directed is the nodeset to which the functions will apply. In the screenshot below, the nodeset of `Language` elements is the context nodeset for the `last` and `position` functions.



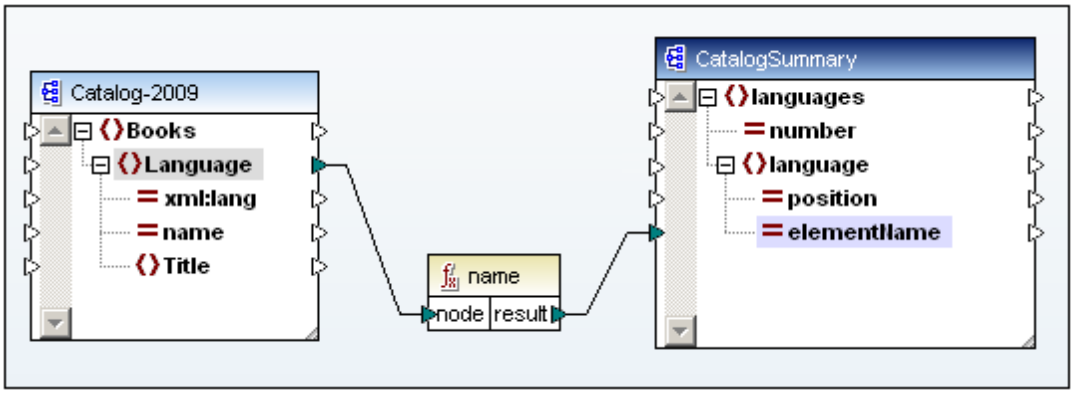
In the example above, the `last` function returns the position of the last node of the context nodeset (the nodeset of `Language` elements) as the value of the `number` attribute. This value is also the size of the nodeset since it indicates the number of nodes in the nodeset.

The `position` function returns the position of the `Language` node being currently processed. For each `Language` element node, its position within the nodeset of `Language` elements is output to the `language/@position` attribute node.

name, local-name, namespace-uri

These functions are all used the same way and return, respectively, the name, local-name, and namespace URI of the input node. The screenshot below shows how these functions are used. Notice that no context node is specified.

The name function returns the name of the `Language` node and outputs it to the `language/@elementname` attribute. If the argument of any of these functions is a nodeset instead of a single node, the name (or local-name or namespace URI) of the first node in the nodeset is returned.



The `name` function returns the QName of the node; the `local-name` function returns the local-name part of the node's QName. For example, if a node's QName is `altova:MyNode`, then `MyNode` is the local name.

The namespace URI is the URI of the namespace to which the node belongs. For example, the `altova:` prefix can be declared to map to a namespace URI in this way: `xmlns:altova="http://www.altova.com/namespaces"`.

Note: Additional XPath 1.0 functions can be found in the Core function library.

8.9.31 xslt | xslt functions

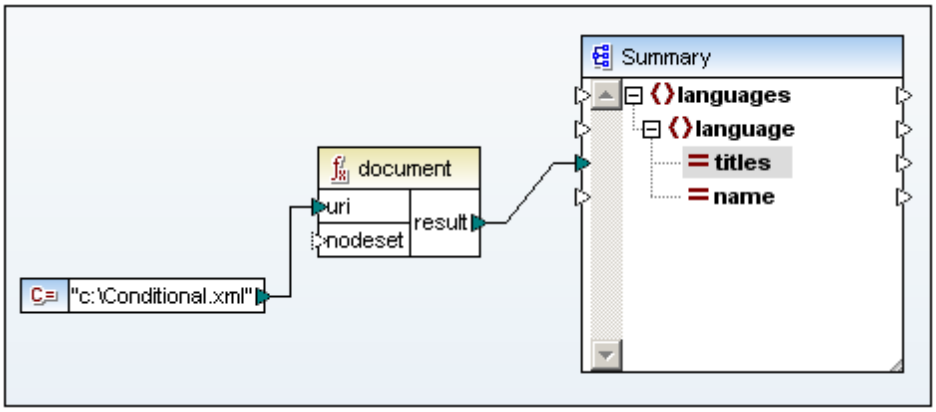
The functions in the XSLT Functions library are XSLT 1.0 functions.

8.9.31.1 currrent

The **`current`** function takes no argument and returns the current node.

8.9.31.2 document

The **`document`** function addresses an external XML document (with the `uri` argument; *see screenshot below*). The optional nodeset argument specifies a node, the base URI of which is used to resolve the URI supplied as the first argument if this URI is relative. The result is output to a node in the output document.

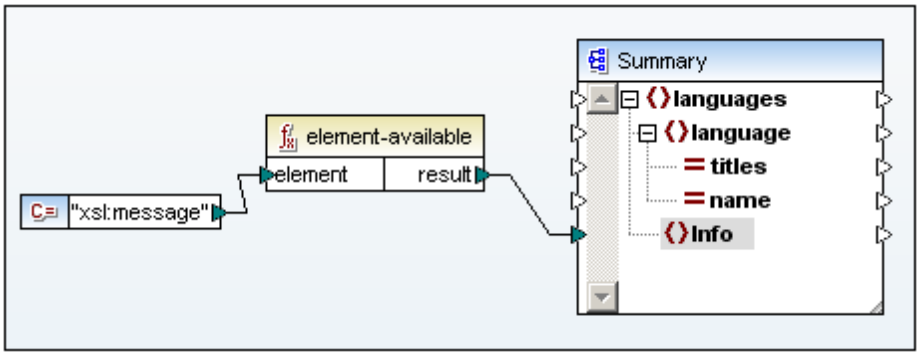


Note that the `uri` argument is a string that must be an absolute file path.

8.9.31.3 element-available

The **`element-available`** function tests whether an element, entered as the only string argument of the function, is supported by the XSLT processor.

The argument string is evaluated as a QName. Therefore, XSLT elements must have an `xsl:` prefix and XML Schema elements must have an `xs:` prefix—since these are the prefixes declared for these namespaces in the underlying XSLT that will be generated for the mapping.



The function returns a boolean.

8.9.31.4 function-available

The **`function-available`** function is similar to the **`element-available`** function and tests whether the function name supplied as the function's argument is supported by the XSLT processor.

The input string is evaluated as a QName. The function returns a boolean.

8.9.31.5 generate-id

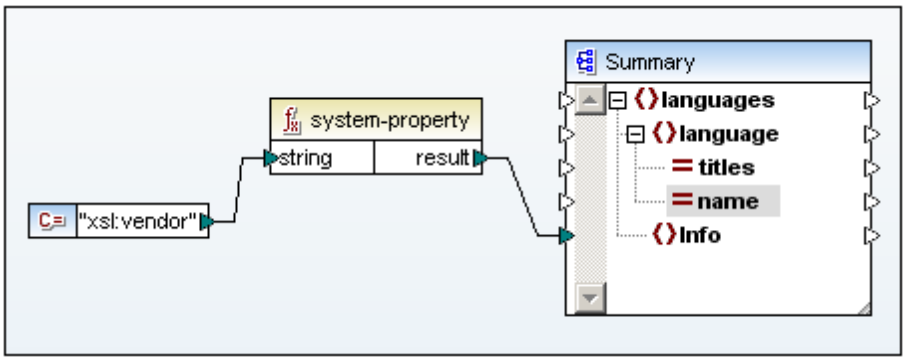
The **`generate-id`** function generates a unique string that identifies the first node in the nodeset identified by the optional input argument.

If no argument is supplied, the ID is generated on the context node. The result can be directed to any node in the output document.

8.9.31.6 system-property

The **system-property** function returns properties of the XSLT processor (the system). Three sytsem properties, all in the XSLT namespace, are mandatory for XSLT processors. These are `xsl:version`, `xsl:vendor`, and `xsl:vendor-url`.

The input string is evaluated as a QName and so must have the `xsl:prefix`, since this is the prefix associated with the XSLT namespace in the underlying XSLT stylesheet.



8.9.31.7 unparsed-enity-uri

If you are using a DTD, you can declare an unparsed entity in it. This unparsed entity (for example an image) will have a URI that locates the unparsed entity.

The input string of the function must match the name of the unparsed entity that has been declared in the DTD. The function then returns the URI of the unparsed entity, which can then be directed to a node in the output document, for example, to an `href` node.

Chapter 9

Implementing Web Services

9 Implementing Web Services

MapForce Enterprise edition enables you to generate Java or C# program code that implements SOAP Web services, based on existing Web Services Description Language (WSDL) files. With MapForce, you can map data to WSDL operations as follows:

- From the input of the WSDL operation to any data sources supported by MapForce, including flat files, XML, XBRL, EDI, Microsoft Excel, and databases.
- From data sources supported by MapForce to the output of the WSDL operation.

MapForce supports WSDL 1.1 and WSDL 2.0 (note that there are specific support details relevant for each language, see also WSDL Support Information).

Prerequisites

To create a Web service with MapForce, the WSDL file of the Web service is required. Note that you can design WSDL files and test SOAP requests with XMLSpy, for example. Additionally, you need platform-specific software required to run a Web service server, for example:

Java

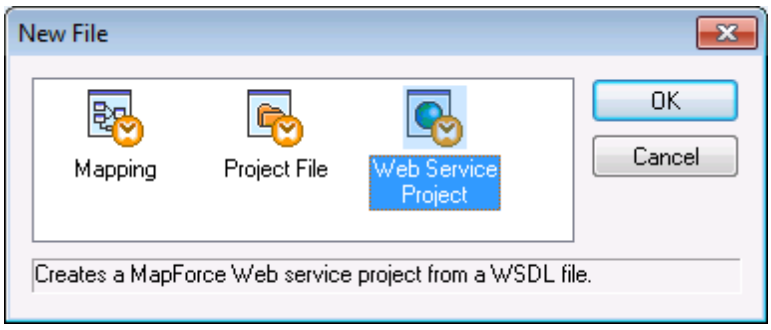
- Java Software Development Kit 1.6 or later
- Apache Tomcat: http://tomcat.apache.org
- Apache Axis2: http://ws.apache.org/axis2/, a SOAP framework running within Tomcat
- Apache Ant: http://ant.apache.org/

C#

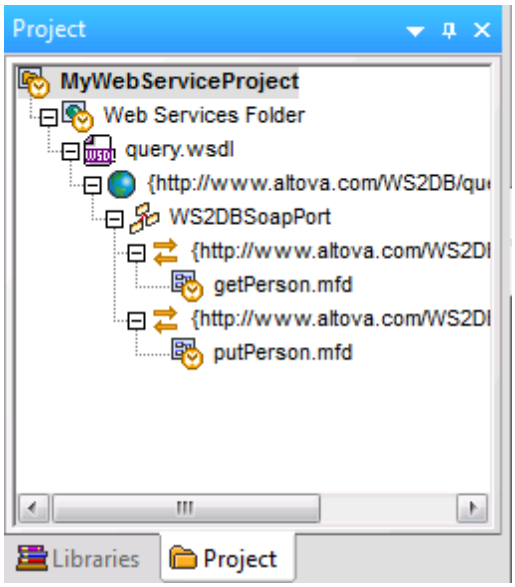
- Microsoft Visual Studio 2005 or later
- Microsoft Internet Information Services (IIS) version 5.0 or later.

How it works

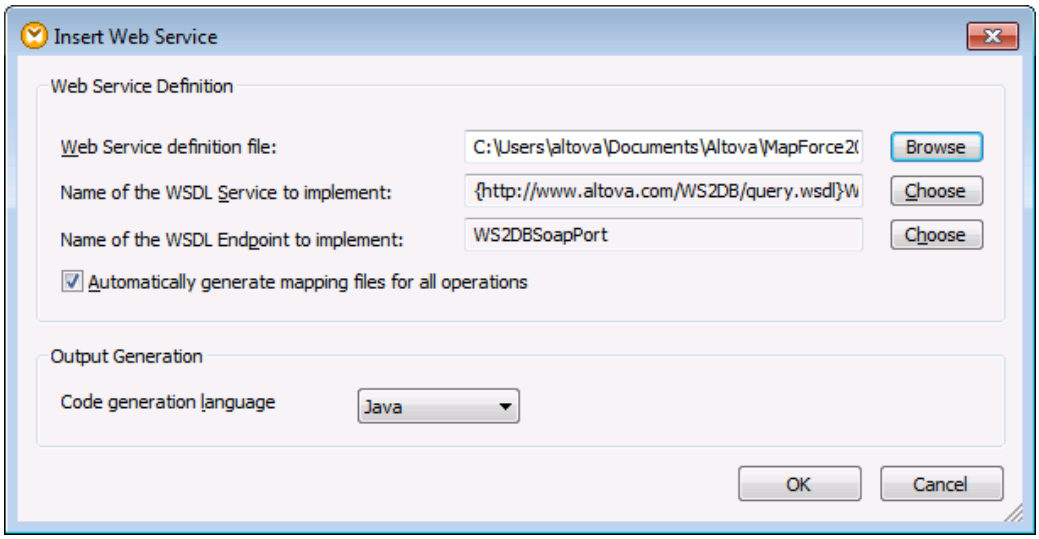
Once you have the WSDL file, you can start a new MapForce Web Service Project.



A MapForce Web Service Project has a predefined structure that enables you to quickly access a particular WSDL operation or Web service in it.



When you start a MapForce Web Service Project, you can optionally instruct MapForce to generate mapping files automatically for all WSDL operations found in the WSDL file.



As an alternative, you can select individually the WSDL Services and WSDL Endpoints to be included in the MapForce Web Service Project.

In the graphical user interface of MapForce, WSDL operations correspond to mappings, and the operation's input and output parameters appear as components in the mapping.

While working with mappings that include input or output of WSDL operations, you can preview the execution result as you would do for any other mapping. To do this, set an example input SOAP instance in the WSDL component. Then select BUILT-IN as transformation language, and click on the **Output** tab.

When your Web Service Project is ready in MapForce, you can generate the C# or Java code either for individual mappings (for testing purposes), or for the entire project. In the latter case, MapForce creates a complete Web service server in the language of choice (C# or Java).

Finally, you compile the generated C# or Java code (outside MapForce) and deploy it to your custom Web server.

Deployment of generated C# or Java code to a Web service server depends on the specifics of the third-party tools and environments you use and thus are covered in this documentation only as examples. For the installation and configuration details of third-party software, as well as those of your Web server, consult the relevant documentation.

9.1 WSDL Support Information

The following table summarizes the WSDL support details in MapForce.

WSDL support	Version 1.1, W3C Note from http://www.w3.org/TR/wsdl Version 2.0, W3C Recommendation from http://www.w3.org/TR/wsdl20/
WSDL type system	XML Schema 2001
SOAP support	Version 1.1: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ Version 1.2: http://www.w3.org/TR/soap12-part0/
Protocols	SOAP over HTTP (HTTP POST, HTTP GET protocols are not supported).
C#	The SOAPAction must be different for each operation in C#.
Bindings	Multiple operations with same name are currently not supported (Section 2.5 of the WSDL 1.1 specification).
style/use	• Document/literal: supported. • RPC/literal: supported in C# • RPC/encoded: limited support • One style/use per Web service (Java), or operation (C#) is currently supported.
SOAP headers	Depends on underlying platform.
SOAP encodingStyle	If use="encoded", encoding style **"http://schemas.xmlsoap.org/soap/encoding/"** for complete `soap:Body` is assumed. There is no support for other encoding styles. The `encodingStyle` attribute is ignored in messages (Section 4.1.1 of the SOAP 1.1 specification).
References	• References to external resources are currently not supported (Section 5.4.1 of the SOAP 1.1 specification). • References to independent elements are supported.
SOAP-ENC:Array	Linear access is supported. Partial arrays and sparse arrays are currently not supported.
Custom SOAP enhancements	Not supported.
Default or fixed values in schemas	Not supported.
Non SOAP message validation	Not validated; passed on to underlying framework.
Namespaces	Non namespace entries are invalid WSDL, and are therefore not supported (WSDL and XML 1.0).

WSDL 1.1

portType

A <portType> element defines a Web service interface, namely:

- the **operations** that can be performed.
- the **messages** involved in each operation as inputs and outputs.

types

The <types> element define the datatypes that are used by the Web service. MapForce supports XML Schemas in WSDL files, as this is the most common type system for WSDL files. MapForce displays these elements (datatypes) as items in a (message) component, allowing you to map them to other item/constructs directly.

message

The <message> element defines the **parts** of each message and the **data elements** of an operation's input and output parameters. These are the messages exchanged by the client and server. There are three types of messages: Input, Output and Fault. In MapForce, each **message** is a **component** from or to which you can map other items. Messages can consist of one or more message parts.

When using the document / literal combination in MapForce, it is necessary that the **message / part element** refer to a global element as opposed to a **type**. For example, in the following code, the `element` attribute refers to a global element defined in a schema (`ns2:Vendor`):

```
<message name="processRequest">
      <part name="inputData" element="ns2:Vendor"/>
</message>
```

Whereas the following code references a type in the schema:

```
<message name="processRequest">
      <part name="inputData" type="ns2:VendorType"/>
</message>
```

operation

Operations use messages as input and output parameters. An operation can have:

- one Input message
- zero or more Output messages
- zero or more Fault messages

Input messages can only be used as source components. Output and Fault messages can only be used as target components.

WSDL 2.0

WSDL 2.0 is substantially different from WSDL 1.1, the main differences being:

- PortTypes have been renamed to interfaces.

- Messages and parts are now defined using the XML Schema type system in the types element.
- Ports have been renamed to endpoints.
- WSDL 2.0 operation inputs and outputs are defined by the XML schema.

In MapForce, the Component Settings dialog box of a WSDL component displays "Endpoint" for both WSDL 1.1 Ports and WSDL 2.0 endpoints.

9.2 Creating Web Service Projects from WSDL Files

This topic shows you how to create a MapForce project of type "Web Service Project" that will be used to generate a Web service. The Web service in this example has the following goals:

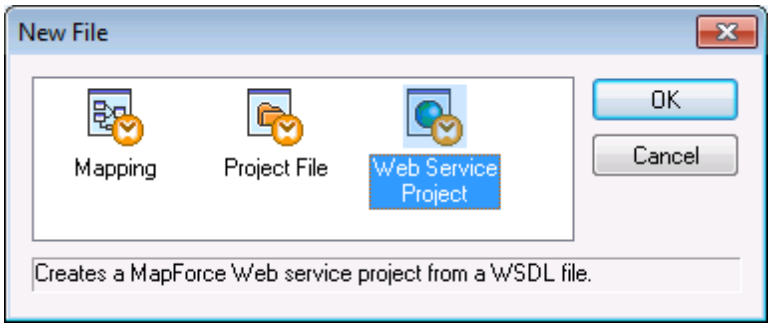
- Search for a specific set of persons in a Microsoft Access database on a server, through a SOAP Request. (The query is entered at run-time on the client, and is then sent to the server.)
- Retrieve the database records as a SOAP Response, taking the form of an XML document sent back to the client, containing all persons conforming to the query.

Please note:

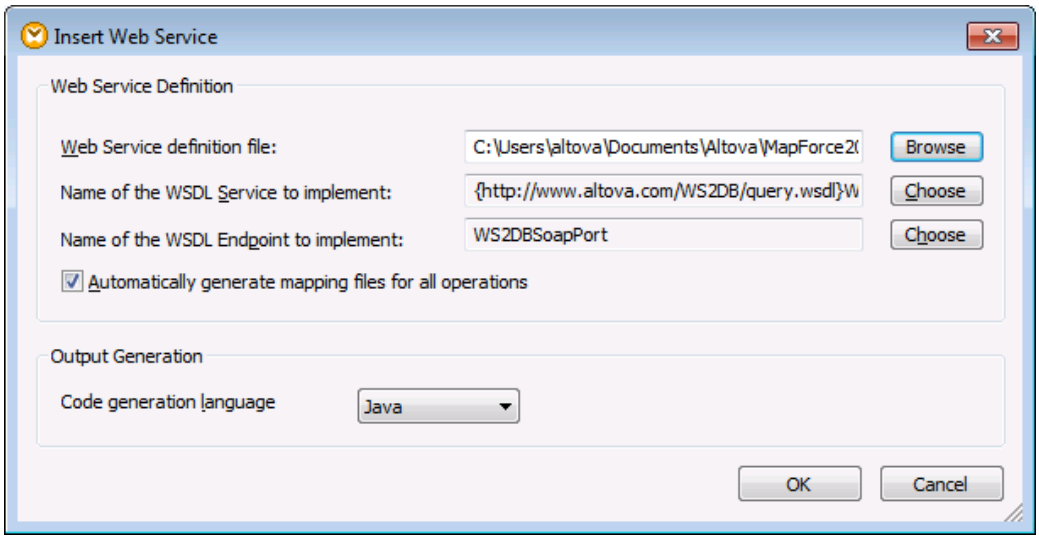
The mapping process used to generate the Web service does not depend on the target programming language, it is identical when generating Java, or C# Web services. The differences only arise when you compile and/or deploy the Web service on the web server.

Creating a Web service project

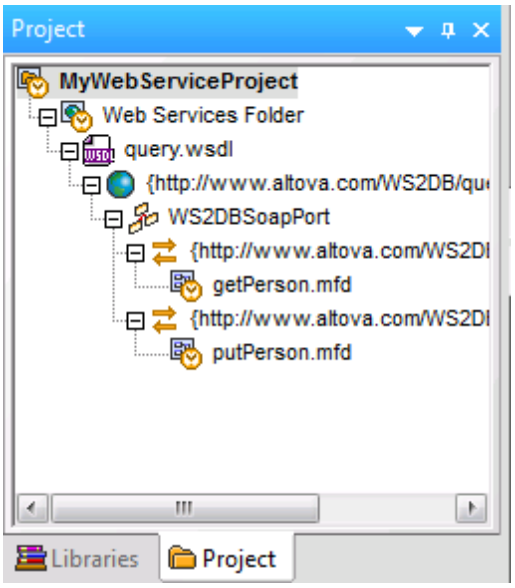
1. On the **File** menu, click **New**, and select "Web Service Project".



2. Browse for the **query.wsdl** file available in the folder **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial** . After you select the WSDL file, MapForce automatically fills in the remaining fields.



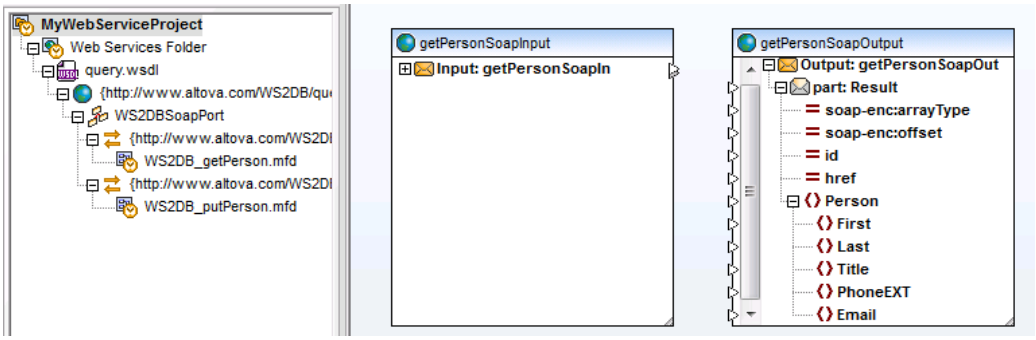
3. Click OK.
4. When prompted, enter the name of the new WSDL project, and click **Save**.



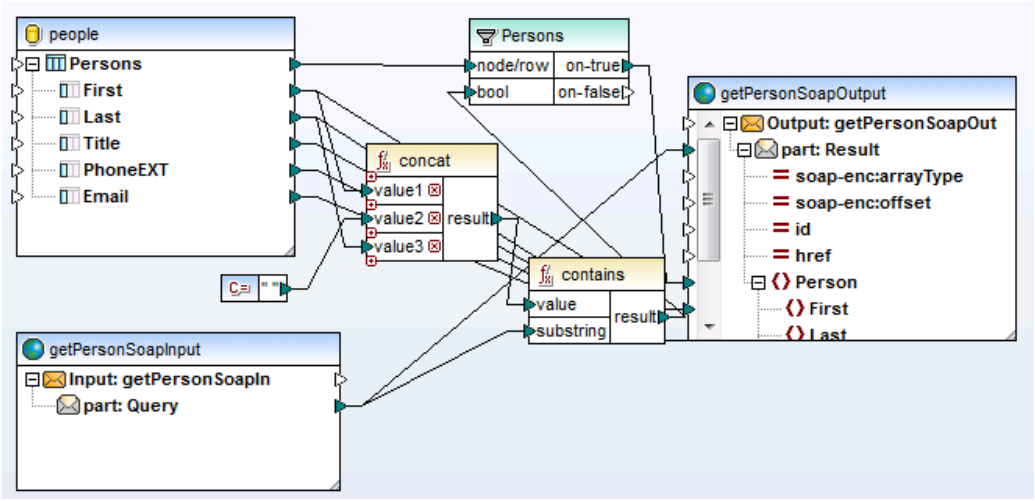
The Project tab shows the project and WSDL name, as well as each of the operations defined in the WSDL file. The two operations are **getPerson** and **putPerson**.

Defining the database query mapping

1. Double-click the **getPerson.mfd** file in the Project tab to load its contents in the main pane. The **getPersonSoapIn** component contains the query (item) which will be used to query the database through the Web service. The **getPersonSoapOut** component contains the Person items defined in the WSDL file.



2. On the **Insert** menu, click **Database**, and add to the mapping area the **people.mdb** database available in the folder **<Documents>\Altova\MapForce2016 \MapForceExamples\Tutorial**. The items in the database match those in the `getPersonSoapOut` component.
3. Add the **`concat`** function, by dragging it from the Libraries window. This function will be used to concatenate the `First` and `Last` names of each person.
4. On the **Insert** menu, click **Constant**, and add a constant which contains one space character. The constant will supply the space character between the first and the last name.
5. Add the **`contains`** function, by dragging it from the Libraries window.
6. Add a filter component (on the **Insert** menu, click **Filter: Nodes/Rows**).
7. Make connections between components as shown below.



The **part:Query** item of the `getPersonSoapIn` component is the query placeholder (it is where the query string is entered in the SOAP client, once the code has been generated, compiled, and deployed on the Web server). The **`contains`** function returns **true** when the query string matches the full or partial name of any person in the database (where "name" is provided by the **`concat`** function, and it consists of the first name, followed by a space, followed by the last name). When there is a match, the details the matching person are included in the response message. Therefore, in this particular example, the search value "Ma" would return the details of both "Martin Rope" and "Mary Morford".

You are now ready to generate code to create a Web service. Web services can be generated for Java or C#. The following sections describe the code generation, compilation and deployment of

the relevant Web services for both Java and C#.

9.3 Generating Java Web Services with MapForce

MapForce generates all necessary code and scripts needed to create a Web service through the normal code generation process (see Code Generator). However, for the Web service to be available to consumers, the generated code must be built and deployed to the Axis2 (Tomcat) server.

Note that when the Web server is running on a remote computer:

- The user must have remote administrative rights
- The Axis2 framework has to be installed on the local computer

Generating and building Java code

1. Open the **Query Person database.mfp** project from the **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial** folder. See the previous section, Creating Web service projects from WSDL files, for an example of how to create such a project.
2. On the **Project** menu, click **Generate code in | Java**, and select the target directory. When code generation completes, several folders and files are created in the target directory, including a **com** directory which contains Altova generic classes, as well as the actual classes of the Web service project.
3. Build the generated Java code (by supplying to Apache Ant the **build.xml** file generated by MapForce). As a result, an Axis Archive File (*.aar) file is created, which you can then deploy to Axis2.

When you open the generated project with Eclipse, you may see an error like "The import org.apache cannot be resolved". In this case, make sure that the Axis2 libraries are added to the Java build path. To add the Axis2 libraries to the Java build path in Eclipse 4.4.2, do the following:

1. Right-click the project in the Package Explorer, and select **Properties**.
2. Click **Java Build Path**.
3. On the **Libraries** tab, click **Add External JARs**, and add the Axis2 libraries from the `<AXIS2_HOME>\lib` folder.

Deploying the Web service

To deploy the Web service, do one of the following:

- Open the "Upload Services" Web administration page of Axis2 and upload the .aar file created in the previous step
- Do a manual upload. For example, if your Tomcat server was installed to the folder `<TOMCAT_HOME>`, you can manually copy the .aar file to `<TOMCAT_HOME>\webapps \axis2\WEB-INF\services`.

Undeployment

Delete the *.aar file from the `<TOMCAT_HOME>\webapps\axis2\WEB-INF\services` folder.

Axis2 limitations

Axis2 support for **RPC/encoded** is limited. MapForce can, however, generate RPC/encoded Web services (both SOAP 1.1 and SOAP 1.2). The limitation is that the original WSDL is not retrieved from the Web server.

This means that, for example, http://127.0.0.1/axis2/services/WS2DB?wsdl would **not** return a usable .wsdl file.

For **document/literal** Web services, the URL above will provide a usable and correct .wsdl file. It will differ from the original, however: comments will be stripped out, and namespaces will be changed. It will, however, still have the same semantics as the original .wsdl file with which the service was generated.

Although Axis2 does not support **RPC/encoded**, it is able to generate WSDL from deployed Java code (compiled code), and thus MapForce-generated code can process **RPC/encoded** messages; Axis2 is just used for transport.

Known issue: namespaces in the SOAP response message

The code generated by MapForce instantiates the **`javax.xml.transform.TransformerFactory`** class. When the class implementation is loaded, it might be read from the **`javax.xml.transform.TransformerFactory`** system property (for details, refer to the Java documentation of this class).

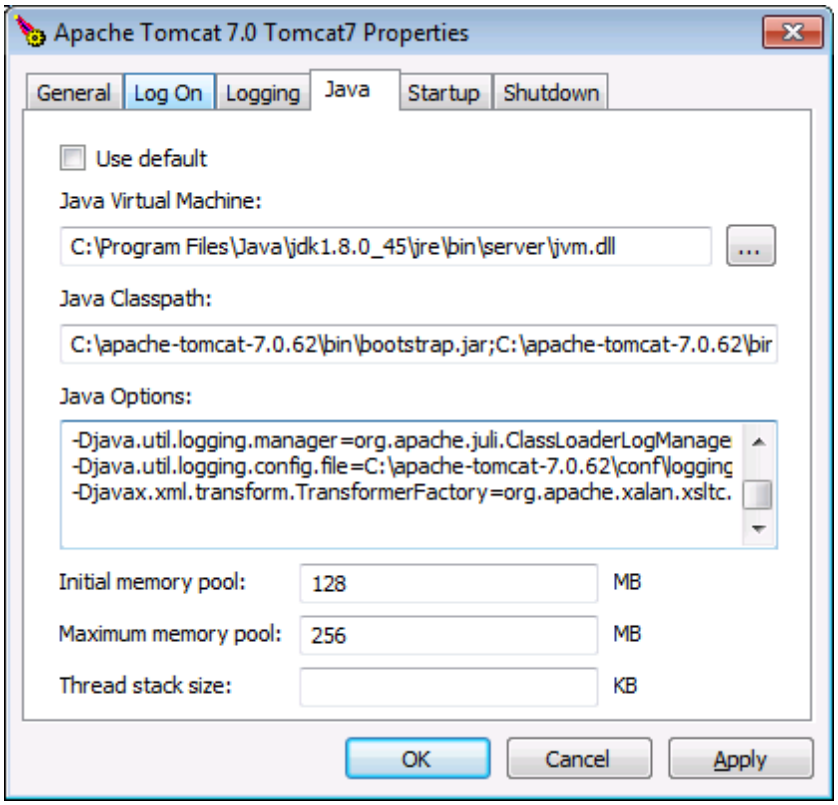
If the system property **`javax.xml.transform.TransformerFactory`** in your Java environment defines the implementation **`org.apache.xalan.transformer.TransformerIdentityImpl`**, the SOAP response message may be generated with incorrect namespaces.

To prevent this issue, it is recommended to use the **`org.apache.xalan.xsltc.trax.TransformerFactoryImpl`** implementation. You can do this by adding the following parameter to the Java Virtual Machine (JVM):

```
-
Djavax.xml.transform.TransformerFactory=org.apache.xalan.xsltc.trax.Transforme
rFactoryImpl
```

The instructions for adding the parameter to the JVM depend on the operating system and server you are using. The following instructions are applicable to Apache Tomcat 7.0 configured to run as a service application on Windows 7:

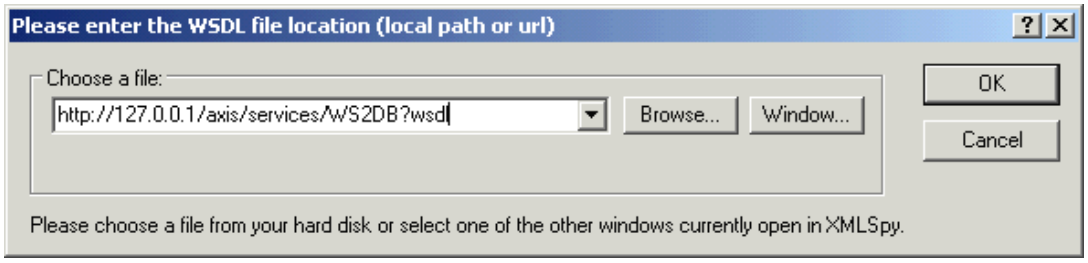
1. Run **Tomcat7w.exe** (this file is located in the **\bin** subfolder of the Tomcat distribution root folder, `CATALINA_HOME`).
2. Click the **Java** tab.
3. At the end of "Java Options" box, enter: `-Djavax.xml.transform.TransformerFactory=org.apache.xalan.xsltc.trax.TransformerFactoryImpl`



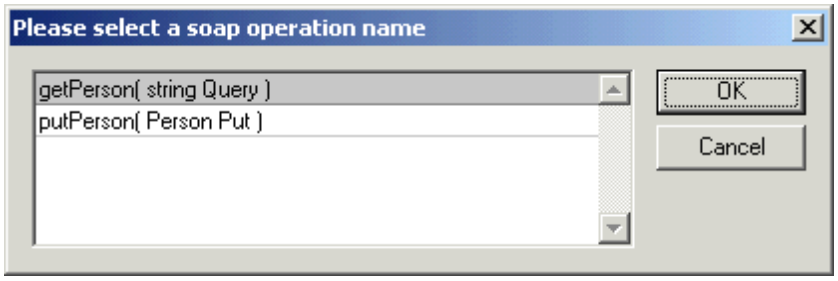
9.3.1 Using the Web Service - getPerson Operation

Using the Web service:
Using a WSDL client, e.g. XMLSpy, create and send a SOAP request.

1. Select **SOAP | Create a new SOAP request**.
2. Enter the WSDL file location on the server e.g. **http://127.0.0.1/axis2/services/WS2DB?wsdl** and hit OK.



3. Select the SOAP operation name that you want to use "**getPerson(string Query)**" in this case, and hit OK to create the SOAP Request document.



The line containing <Query xsi:type="xsd:string">**String**</Query> is the query placeholder, or string you want to search for in the database.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/en
    <SOAP-ENV:Body>
        <m:getPerson xmlns:m="http://www.altova.com/WS2DB/query.wsdl"
            <Query xsi:type="xsd:string">String</Query>
        </m:getPerson>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

We want the Web service to deliver all person records that have the "Ro" string in either the First or Last name.

4. Edit the String text and enter e.g. **Ro**.

```
<m:getPerson xmlns:m="http://www.altova.com/
    <Query xsi:type="xsd:string">Ro</Query>
```

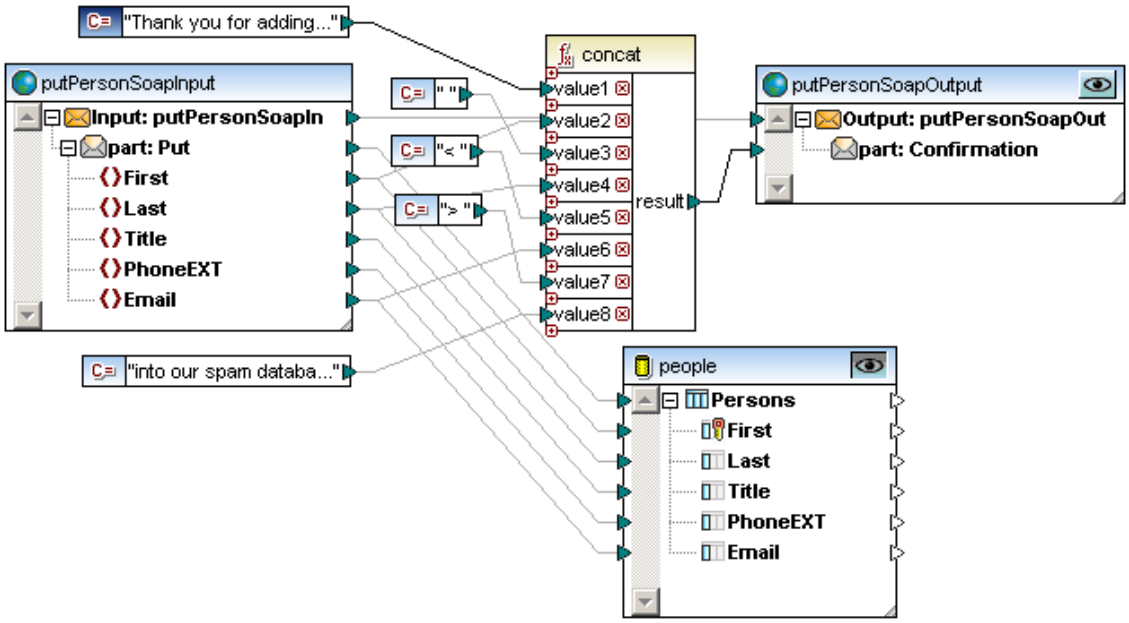
5. Select **SOAP | Send request to server** to send the request to the server.

The SOAP Response document is returned by the Web service, and contains four persons from the Access database.

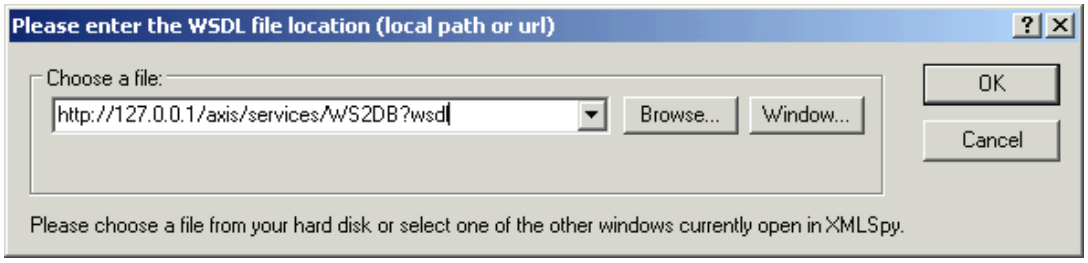
9.3.2 Using the Web Service - putPerson Operation

- Double click the **putPerson** mapping icon (below the Mappings folder) in the Project window to view the mapping.

  This opens the **putPerson.mfd** file in an additional tab.



1. Using a WSDL client, e.g. XMLSpy, create and send a SOAP request.
2. Select **SOAP | Create a new SOAP request**.
3. Enter the WSDL file location on the server e.g. **http://127.0.0.1/axis2/services/WS2DB?wsdl** and hit OK.



4. Select the SOAP operation name that you want to use "**putPerson(Person Put)**" in this case, and hit OK to create the SOAP Request document.
   The SOAP Request document supplies Person placeholder elements, for the user to fill in.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/en
    <SOAP-ENV:Body>
        <m:putPerson xmlns:m="http://www.altova.com/WS2DB/query.wsdl"
            <Put xsi:type="m0:Person">
                <First xsi:type="xsd:string">String</First>
                <Last xsi:type="xsd:string">String</Last>
                <Title xsi:type="xsd:string">String</Title>
                <PhoneEXT xsi:type="xsd:string">String</PhoneEXT>
                <Email xsi:type="xsd:string">String</Email>
            </Put>
        </m:putPerson>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5. Replace the placeholder text (String) with actual person data.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/e
    <SOAP-ENV:Body>
        <m:putPerson xmlns:m="http://www.altova.com/WS2DB/query.wsdl
            <Put xsi:type="m0:Person">
                <First xsi:type="xsd:string">Fred</First>
                <Last xsi:type="xsd:string">Flagellator</Last>
                <Title xsi:type="xsd:string">Sir</Title>
                <PhoneEXT xsi:type="xsd:string">777</PhoneEXT>
                <Email xsi:type="xsd:string">ff@home.com</Email>
            </Put>
        </m:putPerson>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

6. Select **SOAP | Send request to server** to send the request to the server.
   The "Send SOAP Request" command sends the edited SOAP request document to the server.

   The new person data is then added to the database, and a SOAP Response document is sent back as a confirmation. The mapping defines the contents of this confirmation message, i.e. Thank you for adding XYZ to our spam database.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd
http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
    <soapenv:Body>
        <m:putPersonResponse xmlns:n0="http://www.altova.com/WS2DB.xsd" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xmlns:m="http://www.altova.com/WS2DB/query
            <Confirmation xsi:type="xsd:string">Thank you for adding Fred Flagellator
&lt;ff@home.com&gt; into our spam database</Confirmation>
        </m:putPersonResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

9.4 Generating C# Web Services with MapForce

The query.wsdl file supplied in the Tutorial folder has a section that is commented out. This is the <Service name="WS2DB"> section, at the end of the file. Uncomment this section, and comment out the previous <Service...> section before starting this example. This example assumes that the web server is located on the local computer.

Please note:
The URI schemes for Axis2 (Java) and IIS (C#) are different, and changes are required in the WSDL file. The differences occur in the <service> element, which is usually at the end of the WSDL file.

Axis2-style WSDL:

```
<service name="WS2DB">
    <port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
        <soap:address location="http://localhost:8080/axis/services/WS2DB"/>
    </port>
</service>
```

IIS-style WSDL:

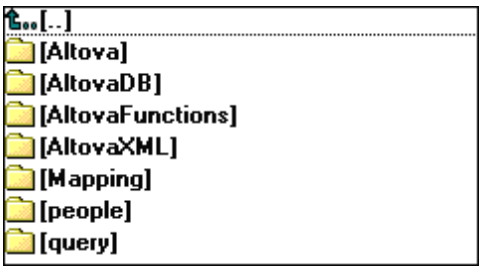
```
<service name="WS2DB">
    <port name="WS2DBSoapPort" binding="tns:WS2DBSoapBinding">
        <soap:address location="http://localhost/services/WS2DB.asmx"/>
    </port>
</service>
```

The example WSDL files contain both variants, one of which is commented out. Also, the WSDL example files include the "soapAction" parameter which is needed by IIS, but ignored by Axis2.

Generating C# code:
Having created (or opened) the Query Person database.mfp project file used in the previous section:

1. Select the menu option **Project | Generate code in | C#**.
2. Select the output directory, and click OK to generate.
   A "C# Code generation completed" message appears when the process was successful.
   If not, an error message appears detailing the specific error(s).

Several folders are automatically generated in the target directory and contain the C# code, project files, and solution files.



You can compile the generated project in Visual Studio 2005/2008/2010.

Compiling the Web service:
Note: If you are using Visual Studio 2005 or 2008, and IIS 7.x, you may first need to install the Windows feature "IIS Metabase and IIS 6 configuration compatibility". You also need to run Visual Studio as Administrator.

The following steps assume the web server is on the local machine. To compile the Web service for a remote computer, please consult the relevant Microsoft documentation (e.g. http://learn.iis.net/page.aspx/387/using-visual-studio-2008-with-IIS/).

1. Open the generated solution file ...**\output\Query_Person_database\Query_Person_database_webservice.sln**.
2. Select the menu option **Build | Build Solution** to compile the Web service project.
3. Select the menu option **Debug | Run** to start the application.

If you are using a 64-bit operating system and the sample does not work because of a missing ADO provider (the ADO provider for Access works only with 32 bit applications), you can do the following:

1. In Visual Studio, select **Build | Configuration Manager**, create a new solution platform for x86, and build.
2. In Internet Information Services (IIS) Manager, find the "application pool" your app is in, right-click "Advanced Settings" and set the property "Enable 32-Bit Applications" to **True**.

Using the Web service:

1. Using a WSDL client, e.g. XMLSpy, create and send a SOAP request.
2. Select **SOAP | Create a new SOAP request**.
3. Enter the WSDL file location on the IIS server and hit OK.
4. Select the SOAP operation name that you want to use "**getPerson(string Query)**" in this case, and hit OK to create the SOAP Request document.
   The line containing <Query xsi:type="xsd:string">**String**</Query> is the query placeholder, or string you want to search for in the database.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/en
    <SOAP-ENV:Body>
        <m:getPerson xmlns:m="http://www.altova.com/WS2DB/query.wsdl"
            <Query xsi:type="xsd:string">String</Query>
        </m:getPerson>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

   We want the Web service to deliver all person records that have the "Ro" string in either the First or Last name.
4. Edit the String text and enter e.g. **Ro**.

```
<m:getPerson xmlns:m="http://www.altova.com/
    <Query xsi:type="xsd:string">Ro</Query>
```

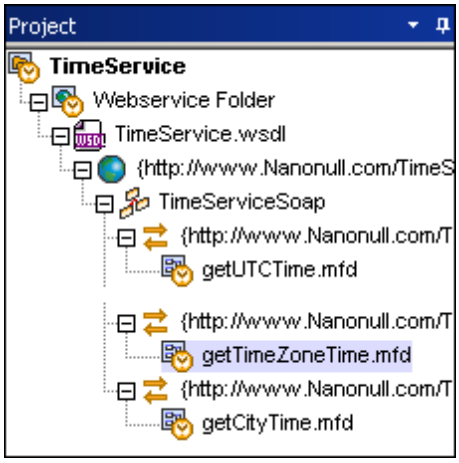
5. Select **SOAP | Send request to server** to send the request to the server.
   The SOAP Response document is returned by the Web service, and contains four persons from the Access database.

9.5 Web Service Faults

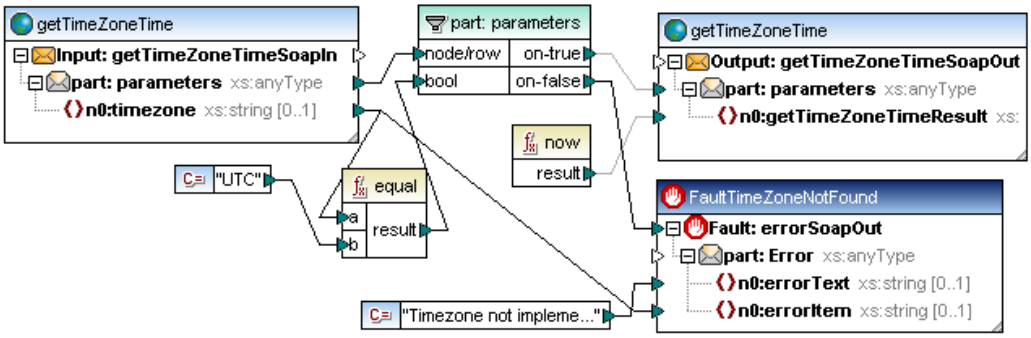
MapForce provides support for the definition of WSDL Faults. A WSDL file can contain a "fault" element for an operation and a message attribute that contains the fault message. A fault **element** has to be present in the WSDL file for you to be able to insert a Fault component in a mapping.

MapForce lets you define the condition that will throw an error and when the condition is satisfied, a user-defined message appears in the Messages window and the mapping process is stopped.

- Double-click the **getTimeZoneTime.mfd file** entry in the Project window of the **Timeservice.mfp** project available in the ...\MapForceExamples\TimeService folder to see an example containing a fault component.



This opens the mapping showing how web service faults are used (screenshot shows WSDL 1.1)



The example above shows how exceptions are defined in mappings. The exception should be triggered when **n0:timezone** is not equal to UTC.

- The **equal** component checks to see if timezone equals UTC, with the bool result being passed on to the filter component.
- If the condition is false, i.e. something other than UTC, the **on-false** parameter of the filter component activates the **Fault:errorSoapOut** exception and the mapping process is halted. (Note that you can also connect the exception to the on-true parameter, if that is

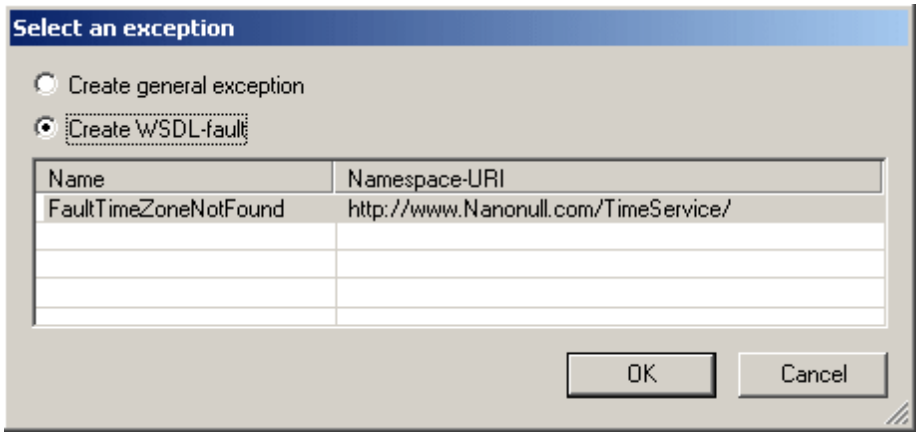
what you need.)
- Two sets of error text are supplied by the SoapFault message.

It is very important to note the filter placement in the example:

- **Both parameters** of the **filter** component, on-true and on-false, must be mapped! One of them needs to be mapped to the fault component, and the other, to the target component that receives the filtered source data. If this is not the case, the fault component will never be triggered.
- The **exception** and **target** components must be **directly connected** to the **filter** component. Functions, or other components, may not be placed between the filter and either the exception, or target components.

To insert a web service fault:

1. Select the menu option **Insert | Exception**, or click the Exception icon  in the icon bar.
2. Click the "Create WSDL-fault" radio button then the fault name in the list, and OK to insert the fault component.



Chapter 10

Calling Web Services

10 Calling Web Services

MapForce supports calling Web services directly from within a mapping. This means that you can insert a Web service call (or, in MapForce terminology, a Web service function) into a mapping, connect input and output components to it, and consume the result as required (for example, preview it in the MapForce output window, pass it to another component, or write it to a file). This effectively turns MapForce into a powerful Web service client which is easily configurable from a graphical user interface.

MapForce supports calling both WSDL-style and REST-style Web services. Therefore, when you add a Web service to the mapping, you can use one of the following approaches:

- For WSDL-style Web services, provide to MapForce the WSDL (Web Service Definition Language) file of the Web service to be called. MapForce uses the .wsdl file to communicate with the Web server. If the .wsdl file implements multiple services, endpoints, and operations, you can select or update them directly in MapForce;
- For non-WSDL Web services, manually enter into MapForce the Web service details. This includes the URL, the request method (for example, GET, POST, PUT), request and response structure (as XML or JSON schema), and parameters. Optionally, if you have the WADL (Web Application Definition Language) file of the Web service to be called, you can import the definition from the WADL file. Also, if you have a sample URL of the Web service, you can instruct MapForce to parse the URL and automatically extract any query, template, or matrix parameters from it, so that you don't have to define them manually.

To call a Web service, the language (execution engine) of the mapping must be set to BUILT-IN (see Selecting a Transformation Language). A Web service call created with MapForce may be executed either by MapForce itself, or on a different machine or even platform, by MapForce Server, through the command-line interface or an API call (http://www.altova.com/mapforce/mapforce-server.html).

For WSDL-style Web services, you can additionally set the transformation language to C# and Java. This way, WSDL-style Web service calls can also be executed from the C# or Java code generated by MapForce.

Note: In C# and Java, the default HTTPS authentication option is supported (that is, MapForce checks the server certificate). Setting additional HTTPS settings (that is, client certificates) is not supported in these languages.

When you select BUILT-IN as transformation language, you can call not only unsecured Web services, but also Web services which require basic HTTP authentication or HTTPS (TLS) authentication with digital certificates. MapForce also provides options to check if the digital certificate of the Web server you are calling is valid and trusted.

Some Web services require preemptive HTTP authentication—you can configure this option also directly in MapForce. Additionally, for WSDL-style services, basic support for WS-Security is available. Namely, you can configure in MapForce the contents of the **UsernameToken** and **Timestamp** SOAP security tokens.

If the Web server is not responding, you can configure in MapForce the interval after which the call should timeout. Other server error responses, such as bad URL, are also handled by MapForce.

In the case of WSDL-style Web services, you can use the following protocols in MapForce:

SOAP 1.1, SOAP 1.2	Both the *RPC/encoded* and *document/literal* styles are supported. If the Web server returns a WSDL fault, the mapping execution stops. For such cases, you can optionally insert an exception component on the mapping area to handle the error. If the Web server returns a non-WSDL error, the mapping execution stops, and an error message is returned (or displayed on the screen, if you are previewing the mapping in MapForce).
HTTP GET	The *url-encoded* style is supported.
HTTP POST	The *url-encoded* and *text/xml* styles are supported.

In the case of generic (REST-style) Web services, calls through the HTTP protocol using HTTP verbs (for example, GET, POST, PUT) are supported. Note that MapForce supports either XML or JSON content in the request or response body of the Web services. Other content types are currently not supported.

10.1 Adding a Web Service Call (REST-Style)

This topic describes how to call a generic (not WSDL-style) Web service from a mapping. This includes a large category of Web services that follow or partially follow an architectural style referred to as "REST" (and are typically called "RESTful", or REST-style Web services).

Generic HTTP Web services typically carry custom request or response structures in the message body part. MapForce supports both JSON or XML data in the request or response body. Therefore, from a MapForce perspective, you can call virtually any HTTP Web service which requires or returns XML or JSON structures as long as you can provide that structure to MapForce as a JSON, XML, or DTD schema. MapForce will also accept as structure an XML file with a valid schema reference.

The structure of the Web service could be published by the service provider as XML or JSON schema, or through a formal language such as WADL, or even be a human-readable specification. If you already have the XML or JSON schema of the request/response, you will need the least amount of effort to create the Web service call. If you have the Web service definition as a WADL file, you can import it from the WADL file*, and make any potential adjustments manually. Finally, if you have a sample XML or JSON instance file but don't have a schema file, you can either create or generate the schema with XMLSpy (http://www.altova.com/xmlspy.html). If necessary, XMLSpy can also convert your instance file from XML to JSON, or vice versa.

* Note that WADL provides no standard way to define JSON structures, only XML structures.

Adding a call to a generic Web service

1. On the **Insert** menu, click **Web Service Function**. (Alternatively, click the **Insert Web service function**  toolbar button).
2. Under **Service definition**, click **Manual**.
3. Optionally, if you have the WADL file describing the service, click **Import from WADL file** and select the file (see Importing Web service information from a WADL file).
4. Select the HTTP request method that MapForce should use to call the Web service. You can either select a value from the existing list, or type the name of the request method. The HTTP method names are case-sensitive.
5. Enter the URL of the Web service. If the URL to the Web service uses parameters, note the following:
   a. If you are calling a Web service with "template" or "matrix" style parameters, enclose the parameters within curly braces, for example: `http://example.org/api/products/{id}`. Then define the actual settings of each parameter in the "Parameters" table. At runtime, MapForce processes the parameter names in curly braces and produces the final URL which includes actual values.
   b. If you are calling a Web service with "query" URL parameters (for example, `http://example.org/api/products?sort=asc&category=1&page=1`), do not enter the query part in the URL text box. Instead, define the parameters only in the "Parameters" table.

   For examples, see Defining the Web service parameters.
6. Optionally, under **Timeout**, enter a period in seconds after which the connection should time out if the server is not responding.
7. If the HTTP method requires or returns a body part (either as XML or JSON structure), click the **Edit** button under **Structure** and browse for the JSON or XML schema of the

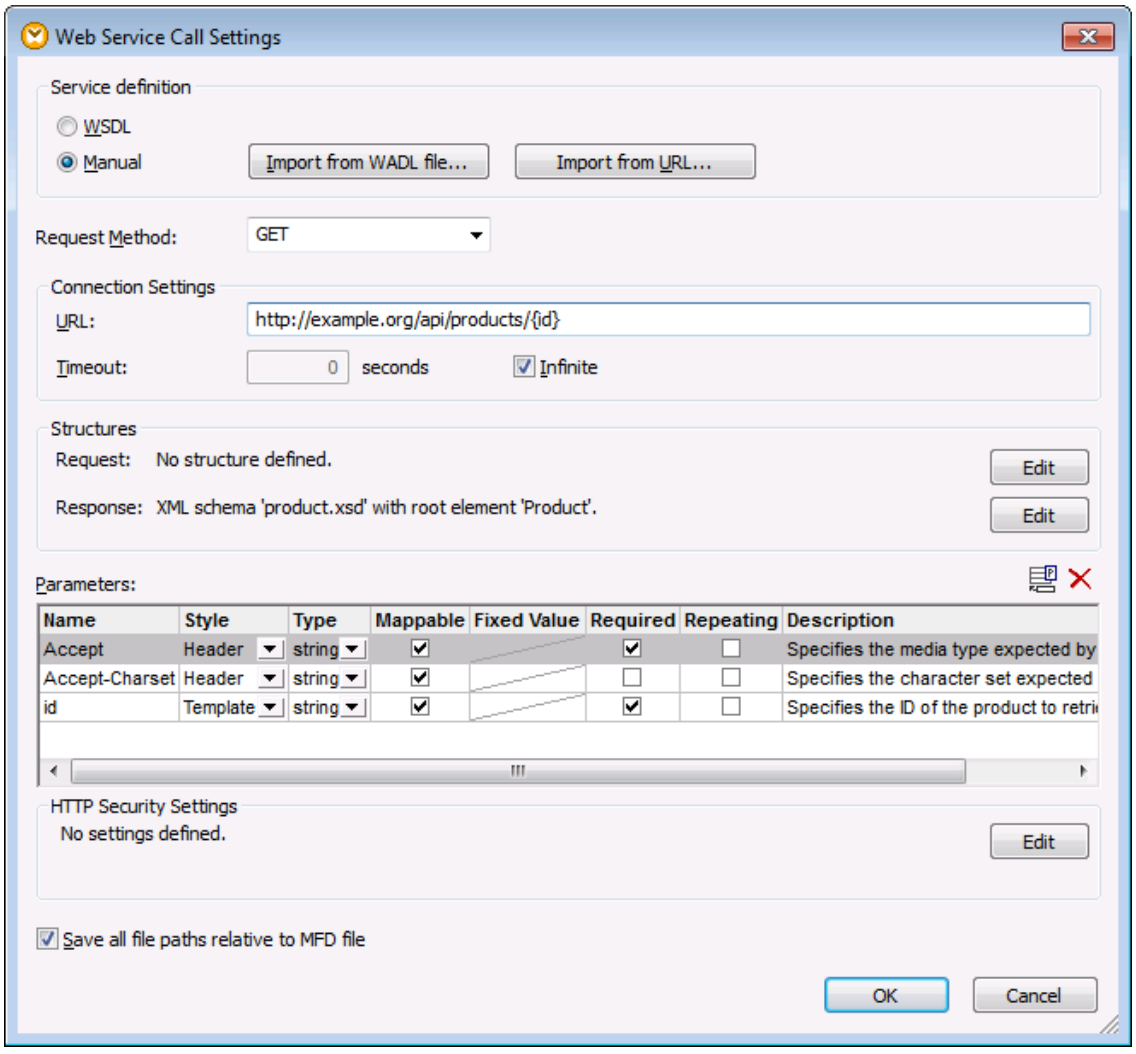
body part (see Defining the response and request structure of the Web Service).

8. Under **Parameters**, define the parameters of the Web service. Optionally, click **Import from URL** to import the parameters from a sample URL of the Web service and populate the "Parameters" table automatically (see Importing Web service parameters from a URL). After importing parameters from a URL, you can modify the contents of the "Parameters" table if necessary. For further information, see Defining the Web service parameters.

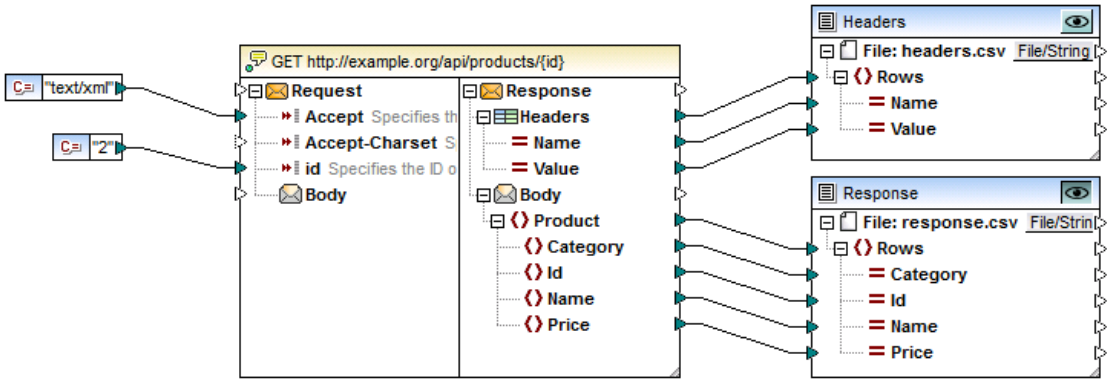
Note: To specify custom request headers, add a parameter with style "Header", where the parameter name corresponds to the header name, and the parameter value corresponds to the header value. Furthermore, if you need to provide the value of the request header from the mapping itself, set the parameter type to "Mappable".

9. If the Web service uses HTTP authentication or certificate-based security, click the **Edit** button under **HTTP Security Settings** and fill in the required fields (see Setting HTTP Security).

After you click **OK**, a new Web service component is added to the mapping area.



The mapping below calls a Web service in order to retrieve a product by its ID using a GET request. In this particular example, the ID supplied in the HTTP request has the constant value "2". However, it can also be a parameter to the mapping (see Supplying Parameters to the Mapping), or be supplied by any component supported by MapForce. In addition to the `id` parameter, the request contains the header `Accept: text/xml`. You can flexibly configure the headers to be used in the request and their values (see Defining the Web service parameters).



This particular mapping has been configured to have two outputs:

1) The headers as comma-separated values
2) The response body as comma-separated values.

To select a particular output for preview, press the button at the top-right corner of the component.

Notice that the Web service component consists of two parts: **Request** and **Response**. The **Request** part enables you to supply data from the mapping to the Web service, while the **Response** part enables you to access the data returned by the Web service and map it to other formats. The structure of the request and response depends on the parameters, as well as the XML or JSON request (or response) structure you have defined from the Web Service Call Settings dialog box (see Defining the response and request structure of the Web Service).

To call the Web service with specific request parameters (if applicable), draw mapping connections between any component supported by MapForce (for example, an XML or JSON file) and the **Request** part. Likewise, to map data returned by the Web service to another format, draw mapping connections between the **Response** part and any other component type supported by MapForce. If you are new to MapForce and need instructions about drawing mapping connections, see Working with Connections.

The response headers returned by the Web service are also mappable, if they are additional headers (the ones which do not begin with "Content"). The header values are available on the Web service component through an item called "Headers" ( ) which contains two child items: "Name" and "Value". This structure acts as sequence, and enables you to map data from any number of headers returned by the Web service. To map the data from the response headers to any other format supported by MapForce, connect the "Headers" structure node to a target sequence in the mapping. For example, if you connect the **Headers** sequence (and its children) to a **Rows** sequence (and its children) of a CSV component, one header would correspond to one row in the CSV file.

Mappings containing generic HTTP Web service calls can be executed like most other mappings, namely:

- Manually, with MapForce, by clicking the **Output** button. In this case, the result of the mapping call is immediately available in the Output pane.
- Through command line or API calls, with MapForce Server (http://www.altova.com/mapforce/mapforce-server.html). This requires compiling the mapping to a mapping execution file first (see Compiling Mappings to MapForce Server Execution Files).
- As a recurring job, with MapForce Server running under FlowForce Server control (http://www.altova.com/flowforce.html). See also Deploying Mappings to FlowForce Server.

The mapping execution fails with an appropriate error message in the following cases:

- The Web service call returns an HTTP status code greater than 299
- The response XML or JSON cannot be parsed
- The Web service cannot be called due to connection failure or DNS resolution problems.

For step-by-step examples of how to create a generic Web service call, see:

- Example: Calling a REST-Style Web Service
- Example: Mapping Data from an RSS Feed

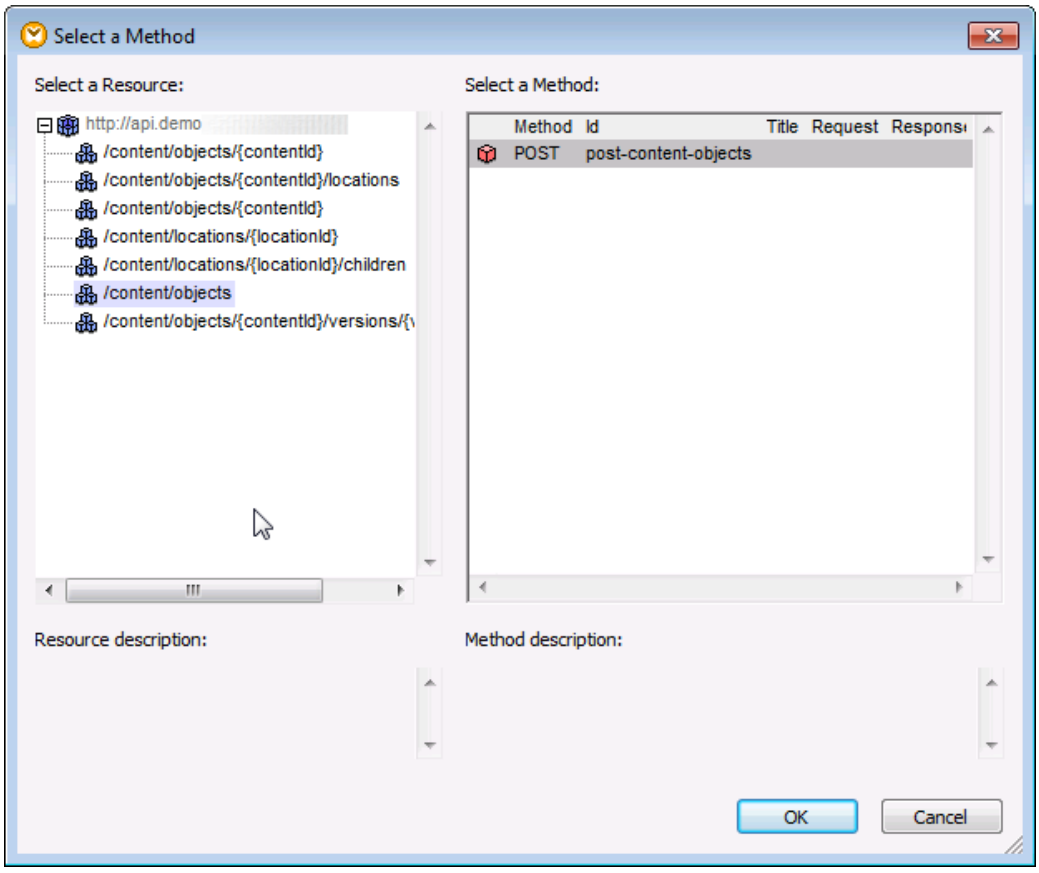
Importing Web service information from a WADL file

The Web Application Description Language (WADL) is one of the ways to represent the contents of the responses and requests used in a web application, including RESTful Web services. MapForce supports importing Web service information from WADL version 2009 (http://www.w3.org/Submission/2009/SUBM-wadl-20090831/).

If the Web service has a WADL file, you can import the Web service definition from the WADL file, rather than entering it manually. When you import information from a WADL file, the parameters and the XML message body structure are populated automatically on the Web Service Call Settings dialog box. After the import, you can modify the parameters and the response/request structure manually, if necessary. This will not affect the underlying WADL file.

To import Web service information from a WADL file:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
2. Under Service Definition, select **Manual**.
3. Click **Import from WADL file** and browse for the WADL file.



4. On the dialog box, select the resource and method, and click **OK**.

Importing Web service parameters from a URL

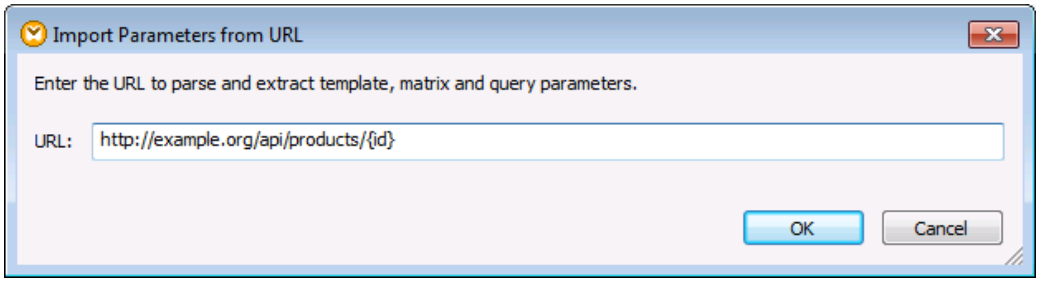
If the URL to the Web service uses parameters, you can instruct MapForce to parse the URL and import any parameters automatically. When you use this option, any template, matrix, or query parameters extracted from the URL become available in the **Parameters** table of the Web Service Call Settings dialog box, where you can further manipulate them as necessary.

Example of a URL with template parameters: `http://example.org/api/products/{id}`
Example of a URL with matrix parameters: `http://example.org/api/products;sort=asc;category=1;page=1`
Example of a URL with query parameters: `http://example.org/api/products?sort=asc&category=1&page=1`

Note: A URL must begin with either "http://" or "https://" to be parseable.

To import Web service parameters from a URL:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
2. Under **Service Definition**, select **Manual**.
3. Click **Import from URL**.



4. Enter or paste the URL to the Web service in the text box, and click OK.

Defining the Web service parameters

When you need to call a Web service with URL parameters, the parameters must be explicitly defined in the "Parameters" table of the Web Service Call Settings dialog box. The "Parameters" table is also used to define any custom headers in the Web service request, as shown in the examples below.

You can enter the parameters manually, or, optionally, you can import them from an existing WADL file or from a URL (see Importing Web service information from a WADL file and Importing Web service parameters from a URL, respectively). The imported parameters become available in the **Parameters** table, where you can further modify them if required.

To add or remove Web service parameters manually:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
2. Under **Service Definition**, select **Manual**.
3. Use the **Add Parameter** ( ) and **Delete Parameter** ( ) buttons, respectively.

The columns in the **Parameters** table have the following meaning:

Name	Specifies the name of the URL parameter. The parameter name must be unique and may consist of letters, digits, periods (.), hyphens (-) and underscores (_). No spaces are allowed in the parameter name.
Style	Specifies the syntax (style) of the URL parameter. Use the "Header" style to add a parameter to the HTTP header when calling the Web service. For example, adding a parameter `Accept` with value `text/xml` is equivalent to specifying the `Accept: text/xml` header, which informs the Web server that MapForce expects the response to be in XML format. For more information about HTTP headers, see http://www.iana.org/assignments/message-headers/message-headers.xhtml. Use the "Query" style for URL parameters that define key-value pairs using the format: **`?key=value&key=value`** For example: `http://example.org/api/products?sort=asc&category=1&page=1`

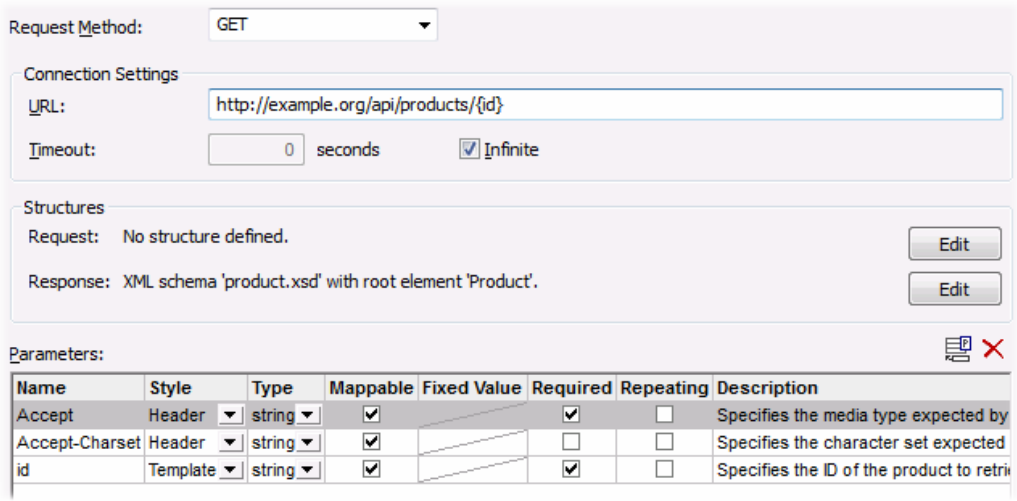
	Use the "Template" style for URL parameters enclosed within curly brackets, for example: `http://example.org/api/products/{id}` . For such parameters, MapForce escapes the values according to the RFC 6570 rules (http://tools.ietf.org/html/rfc6570). Use the "Matrix" style for URL parameters that define key-value pairs in the format: **`;key=value;key=value;`** For example: `http://example.org/api/products;sort=asc;category=1;page=1;` To use Boolean matrix parameters, set the style to "Matrix" and the type to "boolean" (see also next option).
Type	Specifies the data type of the parameter (string, integer, date, etc). This can be any XML schema type. Note that any value that is not a string is converted to a string when the Web service call takes place. Nevertheless, setting a type is meaningful if you want MapForce to show conversion error messages when you attempt to call a Web service with wrong values.
Mappable	Select this check box if you want to pass values to this parameter from the mapping. This option is mutually exclusive with the "Fixed Value" option.
Fixed Value	Specifies the value of the parameter. Applicable only if the parameter has a constant value. Not applicable if the parameter is mappable (see previous option).
Required	Select this check box if the parameter is required by the Web service. For parameters that are required and also mappable, MapForce enforces validation checks (that is, an error message is displayed if the parameter does not have a value).
Repeating	Specifies whether the parameter is single-valued or may have multiple values. This setting is applicable only for mappable parameters. It enables you to pass multiple values in the same Web service call, by means of a single parameter. When you select the check box, you can connect a sequence of values to the parameter structure node on the mapping, instead of a single value. MapForce will then handle the sequence of values depending on the style of the parameter, as follows: • For "Template" parameters, the values will be supplied to the Web service as comma-separated, for example: `http://example.org/api/products/1,2,3` • For "Query" parameters, the parameter name will be repeated for each value, for example: `http://example.org/api/products?color=red&color=green&color=blue` • If the style is "Matrix", multiple values will be separated by

	comma, for example: `http://example.org/api/products;color=red;color=green;color=blue;size=big;size=small;` • If the style is "Header", the HTTP header will be repeated for each value.
Description	Specifies the optional description of the parameter. If the parameter is mappable, the description entered here appears on the mapping component as an annotation next to the mapping item.

Example 1

The Web service illustrated below retrieves a product by its identifier (`id`) using the HTTP GET method. The URL of the Web service specifies the `id` parameter in the curly brackets. Notice that the `id` parameter also exists in the Parameters table and has the style "Template". It is also **mappable**: this causes the parameter to appear on the mapping as a structure node to which you can connect the actual value of `id` (which could be, for example, taken from a database, a file, or a constant). At mapping execution runtime, this parameter would be replaced with the actual value; so, if the value is "1", the URL becomes `http://example.org/api/products/1` .

Note that, if you want to supply a constant `id` value, you can also uncheck the option "Mappable" and enter the value in the "Fixed value" column instead of supplying it from the mapping.

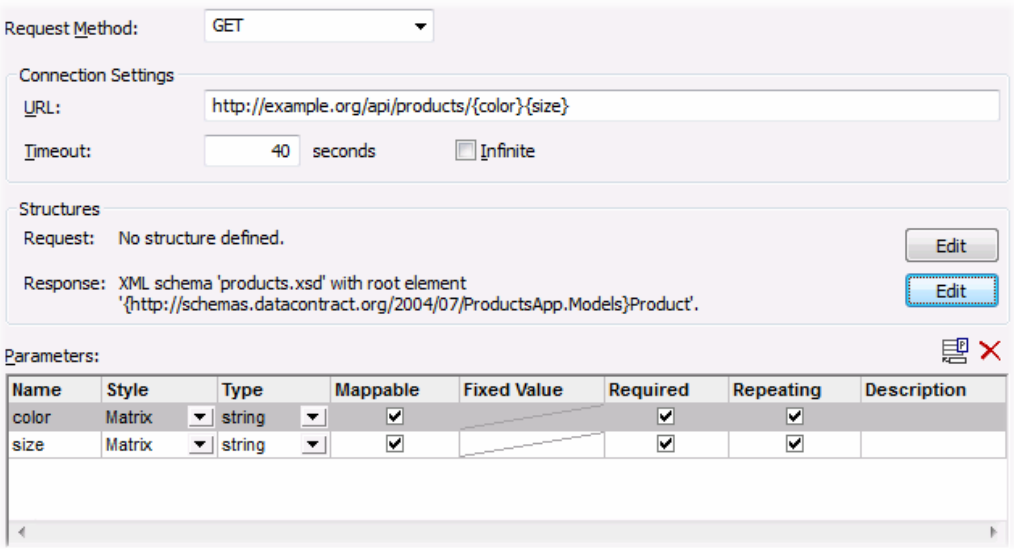


The parameters `Accept` and `Accept-Charset` have the "Header" style. These parameters are used to call the Web service with custom request headers. There are two ways to supply the header value:

- Leave the option **Mappable** checked and supply the custom header value from the mapping, or
- Uncheck the option **Mappable** and enter the value directly in the "Fixed value" column.

Example 2

The Web service illustrated below retrieves a list of products that match the color and size supplied as arguments. The style of the parameters is "Matrix", so they are defined both as placeholders inside the URL and in the mapping table. Notice that the parameters are mappable and the "Repeating" option is checked. This means that their value will be read from some sequence of values on the mapping (for example, a list of rows inside a text file, an XML node, or a database column) and supplied to the Web service at runtime. Thus, a URL such as the one below would become `http://example.org/api/products/;color=red;color=blue;size=big;size=small`, provided that the mapping supplies `red` and `blue` as colors, and `big` and `small` as size.



If you need the URL to be `http://example.org/api/products;color=red,blue;size=big,small`, do the following:

1. Enter the URL as `http://example.org/api/products;color={color};size={size}`
2. In the "Parameters" table, change the parameter style to "Template".

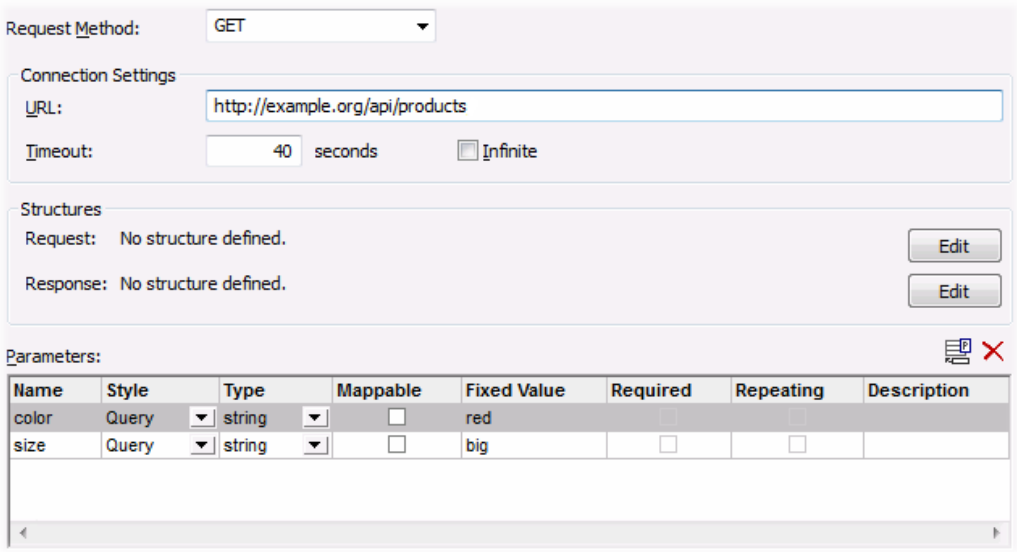
If you need the URL to be `http://example.org/api/products?color=red,blue&size=big,small`, do the following:

1. Enter the URL as `http://example.org/api/products?color={color}&size={size}`
2. In the "Parameters" table, change the parameter style to "Template".

Example 3

The Web service illustrated below also retrieves a list of products that match the color and size supplied as arguments, this time using the style "Query". For this style, it is not necessary to

define the parameters as placeholders in the URL, so they are defined only in the "Parameters" table. The parameter values are entered directly in the "Parameters" table, under "Fixed Values", so the "Mappable" option is unchecked. Thus, at mapping runtime, the URL below would become `http://example.org/api/products?color=red&size=big` .



Defining the response and request structure of the Web Service

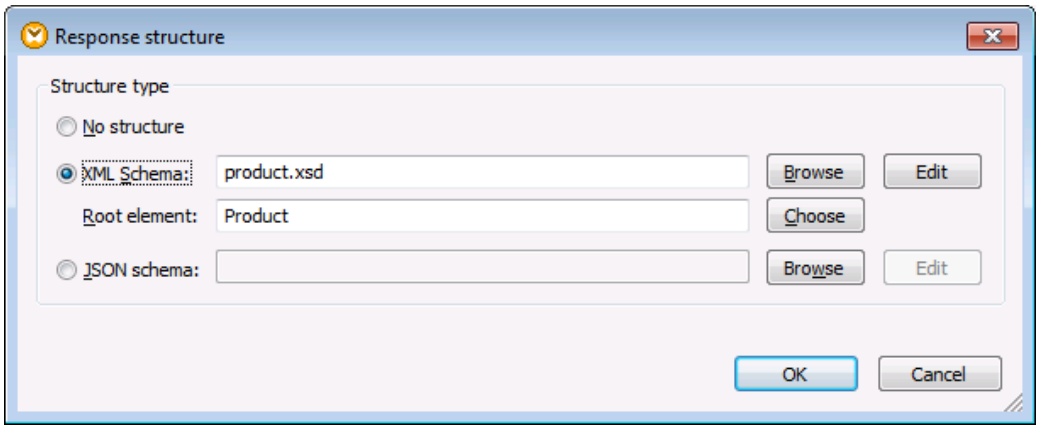
For Web services that use HTTP requests such as POST or PUT, you can include XML or JSON content in the body part of the request message. Likewise, the Web server may return XML or JSON content in the response to the API call. In such cases, you must instruct MapForce what is the schema of the body part of the HTTP message. This way, the elements defined by the structure will appear as mappable items after you add the Web service to the mapping.

To define the XML or JSON schema of a request or response structure:

1. Obtain the XML, JSON, or DTD schema of the request or response structure from the provider of the Web service. MapForce will also accept as structure an XML file with a valid schema reference.

Tip: If you have a sample request or response file but don't have a schema file, you can use XMLSpy to generate the schema file. If necessary, XMLSpy can also convert your instance file from XML to JSON, or vice versa.

2. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
3. Under Service Definition, select **Manual**.
4. Under **Structures**, click **Edit** next to **Request** (or **Response**, depending on the case).
5. On the dialog box, click **Browse** and enter the path to the schema file. If you select an XML instance file, it must have a valid schema reference.



Some XML schemas define elements with global declaration (that is, elements whose parent is the `schema` element). For such schemas, you can choose what element in the schema should be the root element of the mapping structure in MapForce. To do this, click **Choose**, and then, in the dialog box that appears, select the desired root element.

10.2 Adding a Web Service Call (WSDL-Style)

This topic describes how to add to a mapping a call to a WSDL-style Web service. For instructions on how to add a generic Web service to the mapping, see Adding a Web Service Call (REST-Style).

Before adding a WSDL-style Web service call to the mapping area, make sure that you have the Web Service Definition Language (.wsdl) that describes the Web service to which you are attempting to connect.

To add a call to a WSDL-style Web service:

1. On the **Insert** menu, click **Web Service Function**. (Alternatively, click the **Insert Web service function**  toolbar button).
2. Under Service Definition, click **WSDL** if this option is not already selected.
3. Under WSDL Settings, click **Browse** and select the Web Service Definition Language (.wsdl) file. Two sample .wsdl files are available in the MapForce Example folder, one for each version of WSDL (1.0 and 2.0). The corresponding paths are as follows: **<Documents>\Altova\MapForce2016\MapForceExamples\TimeService\TimeService.wsdl** and **<Documents>\Altova\MapForce2016\MapForceExamples\TimeServiceWsdl2\TimeService20.wsdl**.
4. If the .wsdl file defines multiple services, operations, and endpoints, click **Choose** and select the required options.
5. If necessary, define other settings as required (see Web Service Call Settings).
6. Click **OK**.

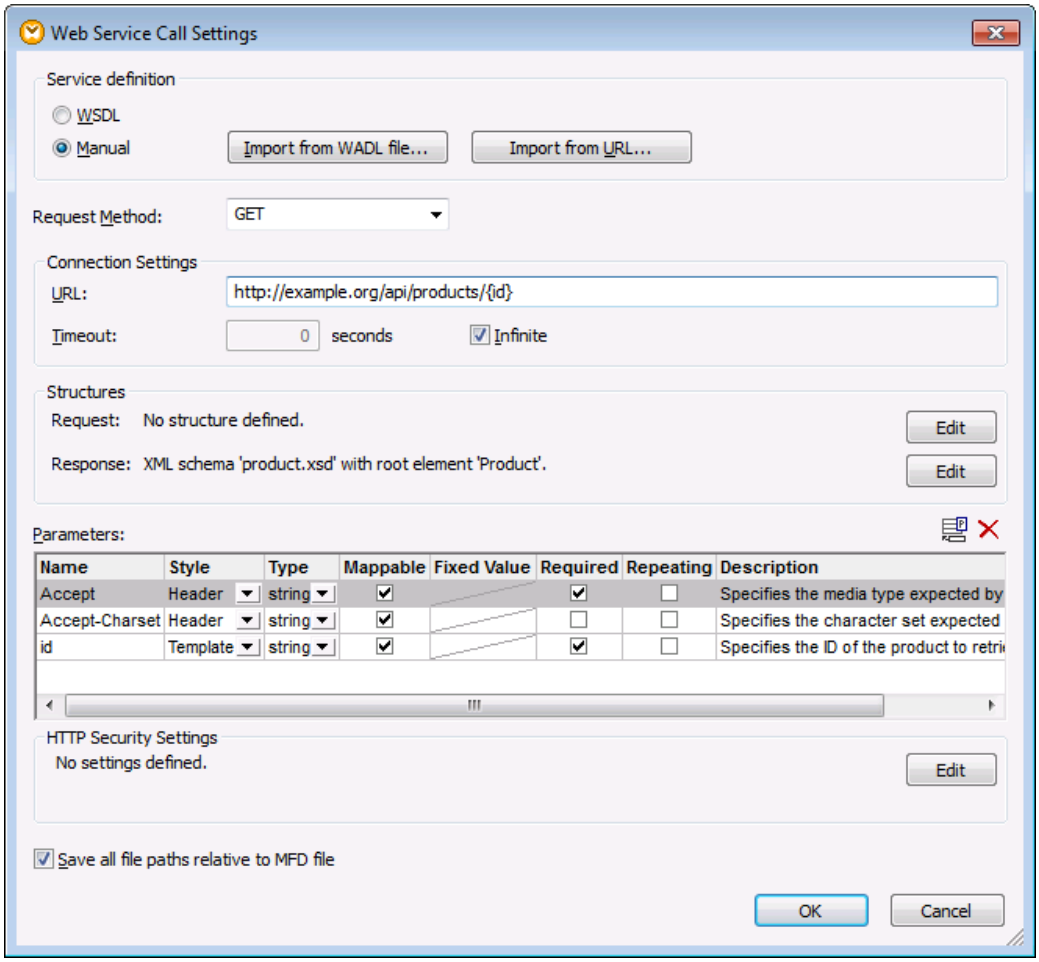
See also Example: Calling a WSDL-Style Web Service.

10.3 Web Service Call Settings

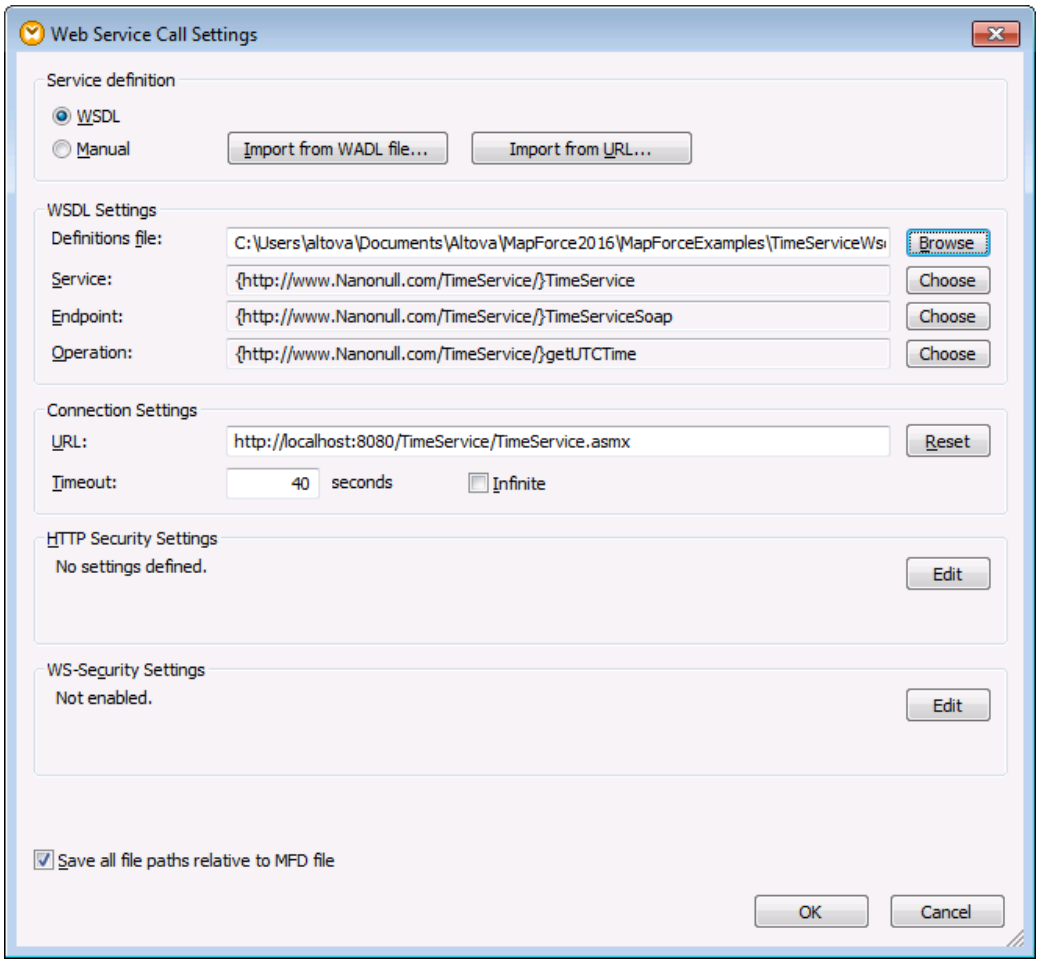
You can change the settings applicable to Web services from the Web Service Call Settings dialog box. To open this dialog box, do one of the following:

- Click the Web service component on the mapping, and, then, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

The list of available settings depends on whether the **Service definition** option at the top of the dialog box is set to **WSDL** or **Manual**. The following image illustrates the settings applicable for a generic HTTP Web service:



The following image illustrates the settings applicable for a WSDL-style Web service:



The available settings are described below.

Service Definition

If the settings you define apply to a WSDL-style Web service, select **WSDL**. Otherwise, select **Manual**.

When **Manual** is selected, you must enter the Web service settings manually into the dialog box. Optionally, if you have the WADL file of the Web service, you can import the Web service settings by clicking **Import from WADL** (see Importing Web service information from WADL). Also, if you want to extract the Web service parameters from a URL, click the **Import from URL** button (see Importing Web service parameters from a URL).

WSDL Settings

This group of settings is applicable only for WSDL-style Web services.

WSDL Definitions	Specifies the Web Service Definition Language (WSDL) file of the web service to be called.
Service	Specifies the name of the Web service to be called. If the WSDL file defines multiple web services, click **Choose** to select the required one.
Endpoint	Specifies the endpoint (or port) of the Web service to be called. If the selected Web service defines multiple web service endpoints, click **Choose** to select the required one.
Operation	Specifies the operation of the Web service to be called. If the selected endpoint defines multiple web service operations, click **Choose** to select the required one.

Request Method

This setting is applicable only for generic (not WSDL) Web services. It defines the HTTP method (verb) used by the service (for example, GET, POST, PUT, and so on).

Connection Settings

URL specifies the address (URL) of the Web service. **Timeout** defines the time interval after which the Web service call will time out if there is no response from the server. Select **Infinite** if the call should wait for a response for an indefinite amount of time.

Parameters

This group of settings is applicable only for generic (not WSDL) Web services. The **Parameters** table specifies the URL parameters with which MapForce will call the Web service (see Defining the Web service parameters).

Structures

This group of settings is applicable only for generic (not WSDL) Web services (see Defining the response and request structure of the Web Service).

HTTP Security Settings

If the Web service requires authentication (either HTTP or HTTPS), click **Edit** to specify the required authentication settings (see Setting HTTP Security).

WS-Security Settings

These settings are applicable only for WSDL-style Web services. If the Web service uses

SOAP security, click **Edit** to specify further settings (see Setting WS-Security).

Save all file paths relative to MFD file

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

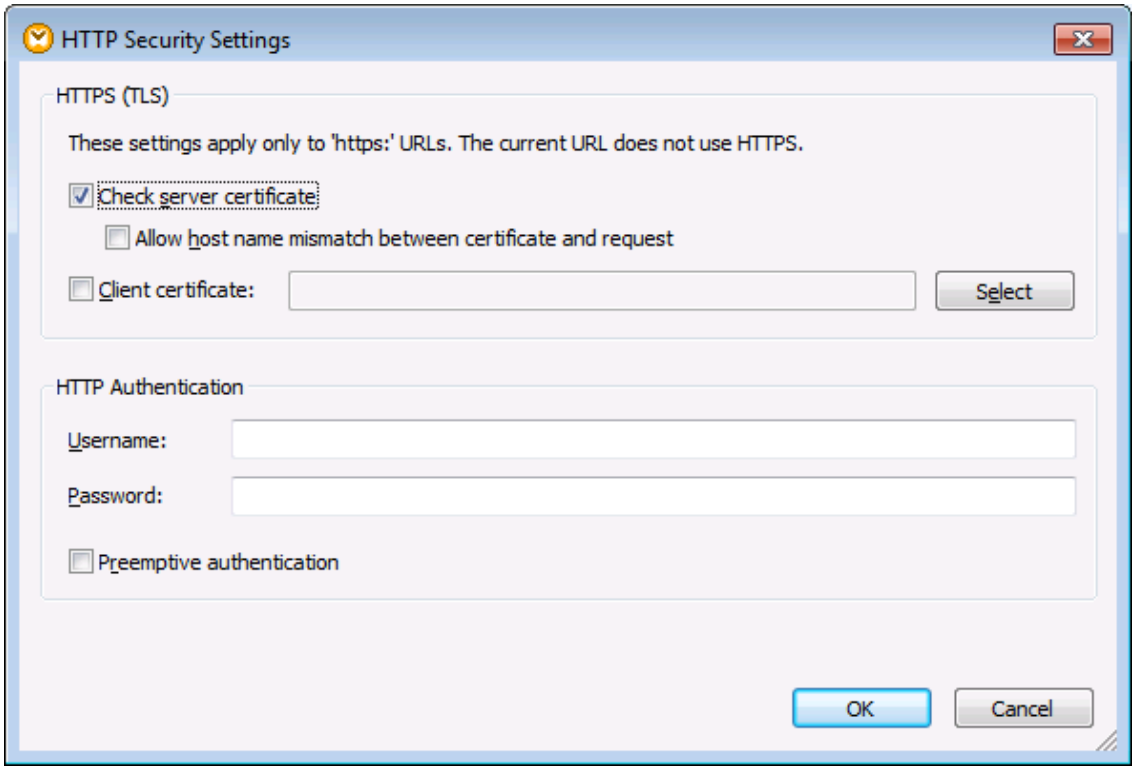
10.4 Setting HTTP Security

The HTTP security settings must be configured in the following cases:

- You are calling a Web service through HTTPS and the Web service requires a client certificate
- The service uses an incorrect server certificate
- The Web service requires HTTP authentication.

To set HTTP security:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
2. Click the **Edit** button next to HTTP Security Settings.



HTTP Security Settings dialog box

The "HTTPS (TLS)" group of options applies for Web services called through HTTPS.

Check Server Certificate	This check box is selected by default, meaning the MapForce is configured to check the certificate of the server before proceeding with the request. When this option is enabled, the Web service request (and the mapping) will fail if the server is not trusted, or if your operating system is not configured to trust the Web server.

	It is not recommended to switch this option off unless you have a good reason to do so. See also Cross-Platform Certificate Management.
Allow host name mismatch between certificate and request	Sometimes a server certificate issued for a particular host name (for example, *www.example.com*) is installed on a different host name (for example, *example.com*). Select this check box to proceed with authentication even if the host name of the certificate does not match the host name called by the Web service.
Client certificate	Click **Select** to choose a client certificate from the **Current User\Personal** certificate store. This assumes the client certificate already exists in the **Current User\Personal** certificate store; otherwise, you can import it using the Certificates snap-in (see Accessing the Certificate Stores on Windows). If the mapping will be deployed for execution to another operating system, the same certificate must be installed on the target operating system as well. For further information, see Cross-Platform Certificate Management .

The "HTTP Authentication" group of options applies if the Web service requires HTTP authentication.

Username	Specifies the username required to access the Web service.
Password	Specifies the password required to access the Web service.
Preemptive authentication	Select this check box if the Web service is configured to expect authentication data in the first call. Otherwise, MapForce attempts access without username and password and will use them if the server requires authorization (HTTP status 401).

10.5 Setting WS-Security

The WS-Security settings must be configured if the Web service is protected by WS-Security and requires that you provide the **UsernameToken** security token.

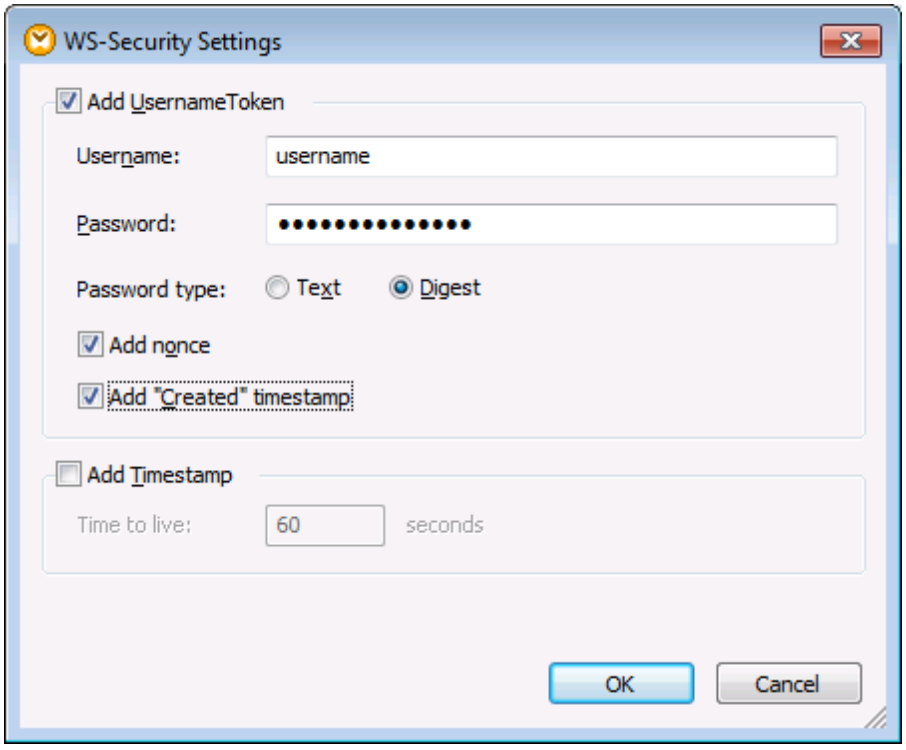
Conventions

The following abbreviations for the namespaces applicable to Web services are used in this topic:

Prefix	Namespace
wsse	`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd`
wsu	`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`

To set WS-Security:

1. Open the Web Service Call Settings dialog box (see Web Service Call Settings).
2. Click the **Edit** button next to WS-Security Settings.



WS-Security Settings dialog box

Add UsernameToken	A UsernameToken is an optional WS-security element

	present in the header of the SOAP message. The UsernameToken is used by the Web server to authenticate the caller of the Web service. In the **wsse** specification (see Conventions), this field corresponds to: `/wsse:UsernameToken`
Username	Enter the username included in the UsernameToken. In the **wsse** specification (see Conventions), this field corresponds to: `/wsse:UsernameToken/wsse:Username`
Password	Enter the text of the password included in the UsernameToken. In the **wsse** specification (see Conventions), this field corresponds to: `/wsse:UsernameToken/wsse:Password`
Password type	Select the type of password included in the UsernameToken. Select **Digest** if the Web server expects the password in this mode; otherwise select **Text**. In the **wsse** specification (see Conventions), this field corresponds to: `/wsse:UsernameToken/wsse:Password/@Type`
Add nonce	Select this check box if you want to add a nonce to the Username token. A nonce is a random value which uniquely identifies each UsenameToken to provide additional security. If you enable this option, it is recommended to enable the **Add "Created" timestamp** option as well. In the **wsse** specification (see Conventions), this field corresponds to: `/wsse:UsernameToken/wsse:Nonce`
Add "Created" timestamp	Select this option to add a timestamp to each nonce. In the **wsse** specification (see Conventions), this field corresponds to: `/wsse:UsernameToken/wsu:Created`
Add Timestamp	Select this check box if you want to enable the time-to-live (TTL) value for the SOAP message (see the next option).

	In the **wsu** specification (see Conventions), this field corresponds to: `/wsu:Timestamp`
Time to live	Enter the time-to-live (TTL) for the SOAP message to diminish the chance of someone intercepting the message and replaying it.

10.6 Example: Calling a REST-Style Web Service

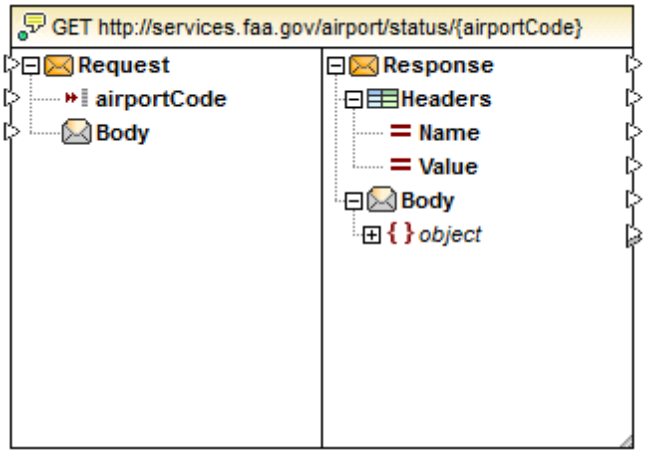
This example shows you how to call a generic HTTP Web service from MapForce. The description of the Web service called in this example can be found at http://services.faa.gov/docs/services/airport/#airportStatus . This Web service returns the current status of any major US airport, as an XML or JSON structure, accepting the three-letter airport code as argument (for example, "SFO", "IAD", "ABE", "DFW", etc). The example is accompanied by a mapping design file, which is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\AirportStatus.mfd** .

In this example, the response of the Web service is in JSON format, and is mapped to a JSON file. Therefore, the JSON schema of the Web service response structure will be required. For convenience, the JSON schema was generated with XMLSpy from the a sample response supplied by the Web service provider. It is available at the following path relative to "(My) Documents" folder: **<Documents>\Altova\MapForce2016\MapForceExamples\AirportStatus.schema.json**.

For the request part, we are going to call the Web service with a parameter which will supply the value "SFO" in the request. For the response part, we will map the response data to a JSON file. Therefore, the mapping will consist of three main components: the Web service call, the input parameter, and the JSON output file.

Step 1: Add the Web service component

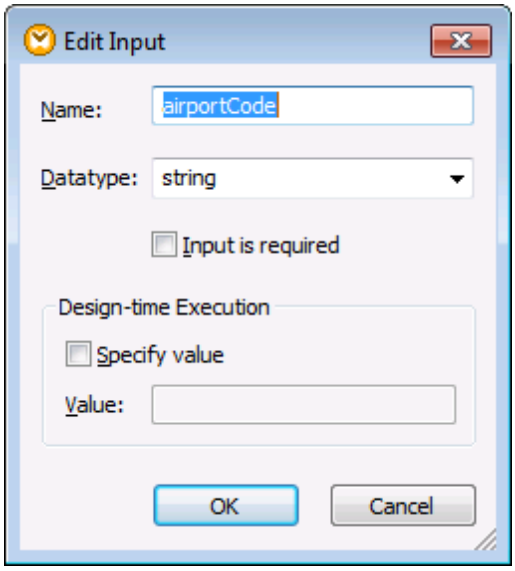
1. Make sure the transformation language of the mapping is BUILT-IN (see Selecting a Transformation Language).
2. On the **Insert** menu, click **Web Service Function**.
3. Under **Service Definition**, click **Manual**.
4. Set the request method to GET and the URL to `http://services.faa.gov/airport/status/{airportCode}`. The value within curly braces is a template parameter which will be replaced with the actual airport code at runtime (see Adding a Web Service Call (REST-Style).
5. Click the **Add Parameter** ( ) button and add a new parameter to the "Parameters" table. Notice the name of the parameter must be the same as that of the parameter specified within curly braces in the URL. Set the style to "Template", type to "String", check the "Mappable" and the "Required" option. The "Description" field is optional.
6. Add a header to tell the Web server that the client (that is, MapForce) expects JSON in the response. To do this, click again the **Add Parameter** ( ) button, name the parameter "Accept", set the style to "Header" and enter **application/json** as fixed value.
7. Under **Response**, click the **Edit** button and browse for the schema of the Web service response. The schema can be found at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\AirportStatus.schema.json**.
8. Click **OK**.



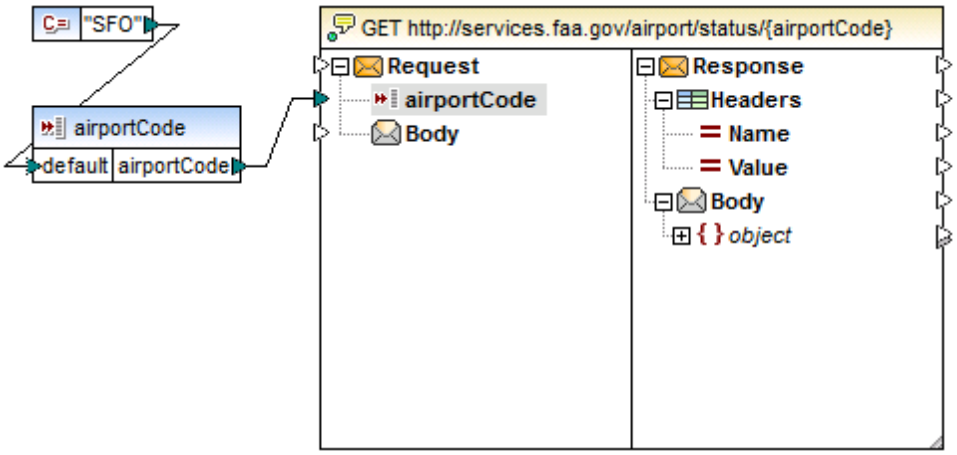
At this stage, the Web service is available on the mapping area and you are ready to map data to the request and from the response part.

Step 2: Add the input parameter

1. On the **Insert** menu, click **Insert Input**.
2. Enter "airportCode" as parameter name and click to clear the **Input is required** option (since the input will be supplied by a constant, as shown in the next step).



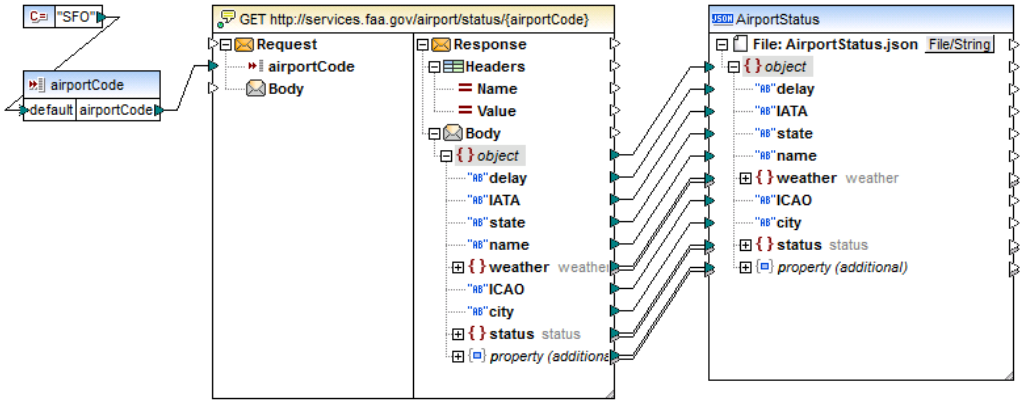
3. Add a constant (using the menu command **Insert | Constant**) with value "SFO" and connect it to the parameter input.
4. Connect the output of the aiportCode parameter to the input of the request, as shown below.



At this stage, the request part of the Web service is ready. At mapping runtime, the Web service will be called with the value provided by the constant (in this case, "SFO").

Step 3: Add the JSON output

1. On the **Insert** menu, click **JSON Schema/File**.
2. Browse for the **<Documents>\Altova\MapForce2016\MapForceExamples \AirportStatus.schema.json**, and click **Open**.
3. When prompted to supply a sample JSON file, click **Skip** (there is no need for a sample JSON file since it will be generated).
4. On the **Connection** menu, make sure that the **Auto Connect Matching Children** menu item is selected. Enabling this option saves you time with the next step.
5. Connect the `object` node of the response body of the Web service to the `object` node of the JSON component, as shown below. Since the **Auto Connect Matching Children** was enabled in the previous step, all descendent nodes are connected automatically, so you don't need to draw individual connections for each.



Step 4: Executing the mapping

You are now ready to call the Web service. Click the **Output** button to execute the mapping and preview the generated output. If the Web service call is executed successfully, the **Output** tab

displays the returned JSON structure. If the call was not successful, MapForce returns the error accordingly. As stated in the description of the Web service, an error 502 (Bad Gateway) may be returned by the Web service if the server is not available. You may also get an HTTP "Not found" (404) error if you use a custom airport code not supported by the Web service.

If you have MapForce Server (http://www.altova.com/mapforce/mapforce-server.html), you can also compile the mapping to a mapping execution file (*.mfx) and execute it from the command-line or from the MapForce Server API on the server machine where MapForce Server runs (see Compiling Mappings to MapForce Server Execution Files).

You have now finished creating a generic HTTP Web service call that uses a GET method to retrieve airport status data in real time. For more information about working with generic Web services, see also Adding a Web Service Call (REST-Style).

10.7 Example: Mapping Data from an RSS Feed

This example shows you how to call a generic HTTP Web service from MapForce in order to map data from an RSS (Rich Site Summary) feed to Microsoft Excel. The example is accompanied by a mapping design file, which is available at the following path: **<Documents>\Altova \MapForce2016\MapForceExamples\RssReader.mfd** .

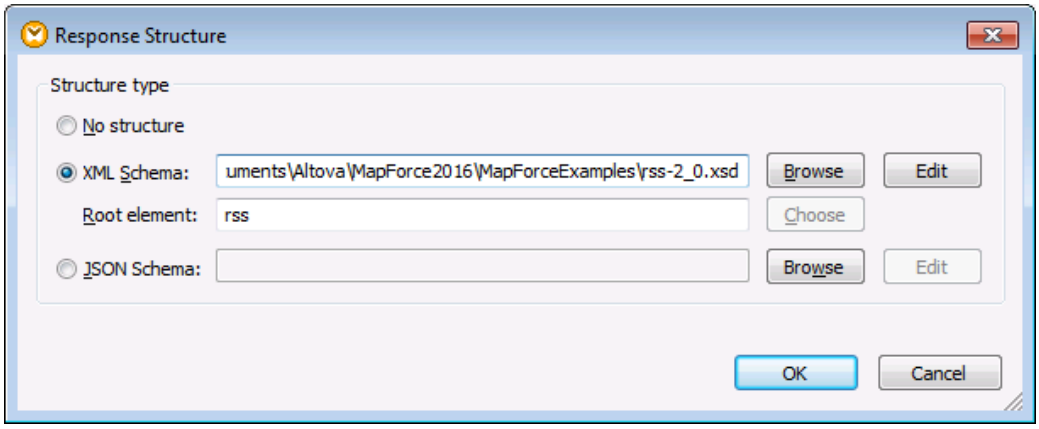
In this example, the response of the Web service is mapped from the RSS feed of the Altova blog (http://blog.altova.com/feed). The schema of the Web service response structure will be required, so that MapForce can create the structure of the data returned by the feed. For convenience, the required schema is available at the following path relative to your "(My) Documents" folder:**<Documents>\Altova\MapForce2016\MapForceExamples\rss-2-0.xsd**.

Calling the RSS feed does not require any request parameter, so the request part of the Web service call will be empty. As for the response, it will be mapped to a Microsoft Excel file. Thirdly, in order to make the publication date of each RSS entry easily readable, it will be formatted as YYYY-MM-DD in the Excel file. To achieve this goal, date processing functions will be used.

Therefore, the mapping will consist of three main parts: the Web service call, the target Excel component, and the date processing functions.

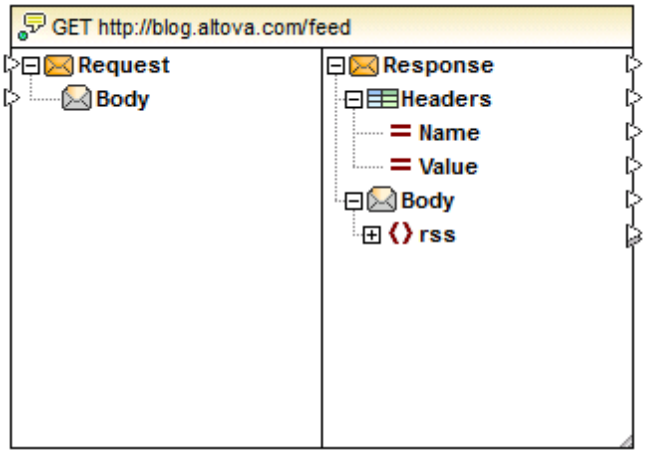
Step 1: Add the Web service component

1. Make sure the transformation language of the mapping is BUILT-IN (see Selecting a Transformation Language).
2. On the **Insert** menu, click **Web Service Function**.
3. Under **Service Definition**, click **Manual**.
4. Set the request method to GET and the URL to `http://blog.altova.com/feed`.
5. Under **Response**, click the **Edit** button and browse for the schema of the Web service response. The schema can be found at the following path: **<Documents>\Altova \MapForce2016\MapForceExamples\rss-2-0.xsd**.



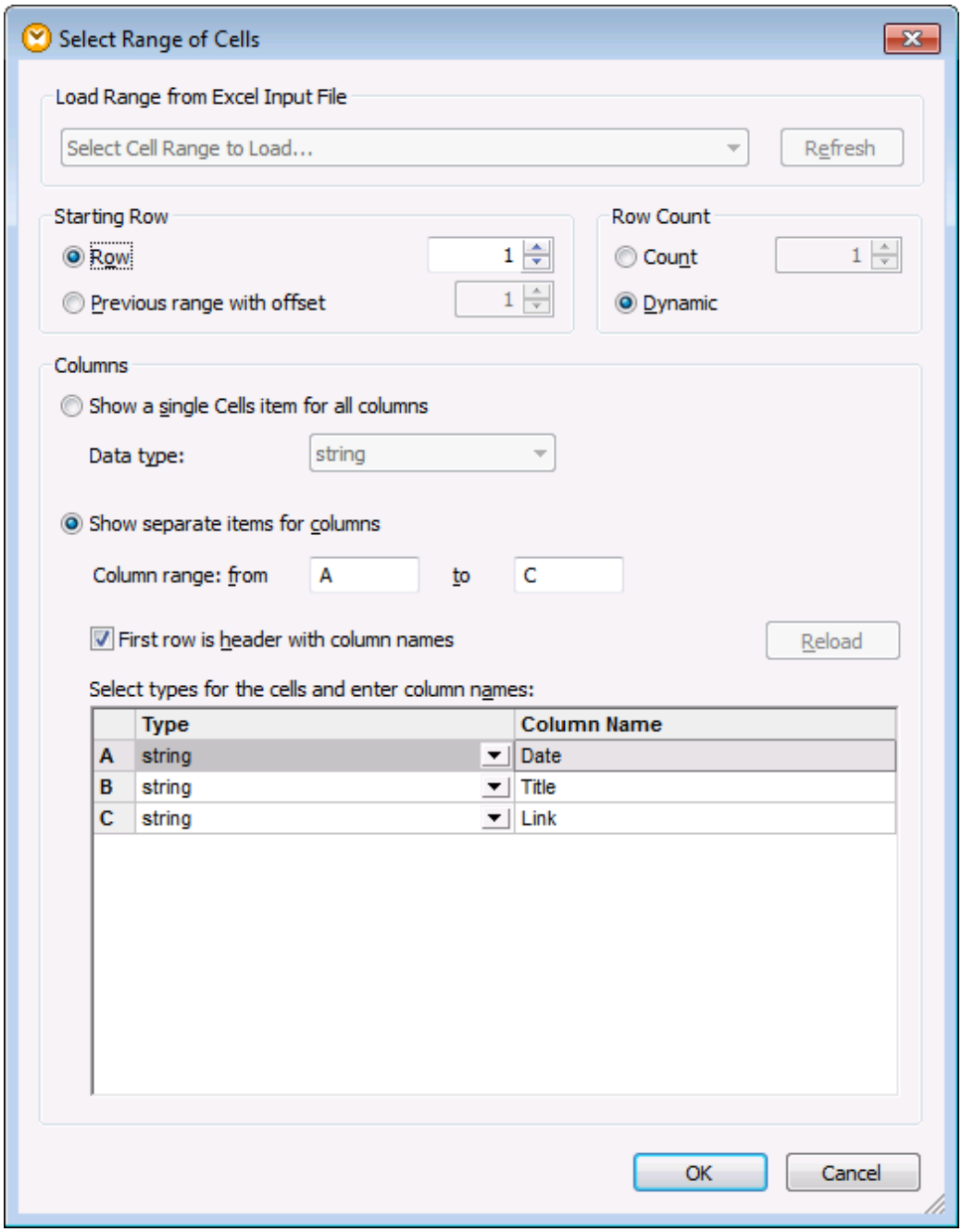
6. Click **OK**.

At this stage, the Web service is available on the mapping area and you are ready to map data from the response part.



Step 2: Add the target Excel component

1. On the **Insert** menu, click **Excel 2007+ File**.
2. When prompted to supply a sample file, click **Skip**. The Excel component is now available on the mapping area.
3. Click the button next to **Sheet 1**, and click to clear the **Show Worksheets by name** option.
4. Click the button next to **Row 1**, and change the options as shown below.



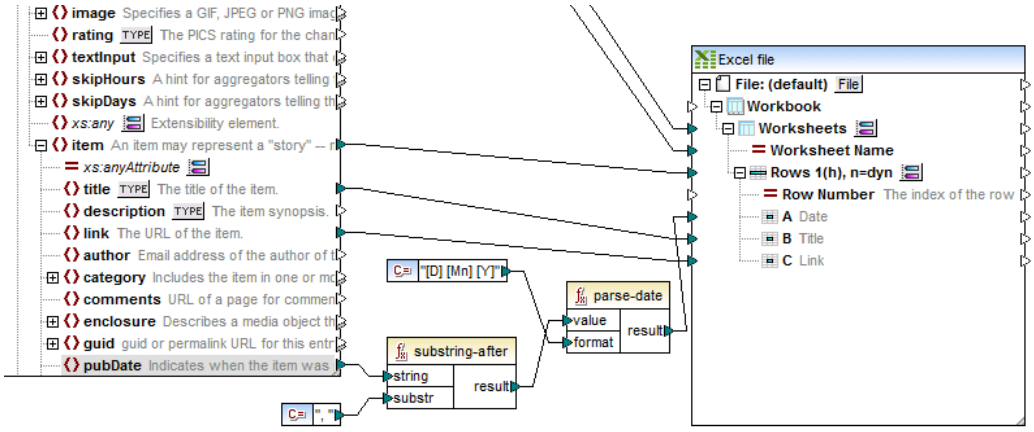
5. Connect the following items of the response body to the following items of the Excel component:

Response body	Excel file
channel	Worksheets
title	Worksheet Name
item	Rows
item/title	Row B (Title)
item/link	Row C (Link)

At this stage, the **Row A (Date)** is still missing an input value. This value will be added in the next step.

Step 3: Add the date processing functions

The date in the response of the RSS feed appears formatted in RFC-822 format, for example: `Wed, 20 Jan 2016 14:49:35 +0000`. The goal is, however, to see the date formatted in the Excel file as `YYYY-MM-DD`. To achieve this, add the **substring-after** and **parse-date** functions of the MapForce core library to the mapping (for instructions about adding functions to the mapping, see Working with Functions).



As shown above, the **substring-after** function takes the value from the **pubDate** node and returns only the text after `,` . The resulting value is then passed to another MapForce core function, **parse-date**, which parses it using the mask **`[D] [Mn] [Y]`** , and returns the value as `xs:date`. The format mask means "numeric day of the month, followed by a space, followed by the month name in title case, followed by a space, followed by the four-digit year" (for more information, see the **format-dateTime** function).

Finally, the parsed value is supplied to the **Date** row of the Excel component.

Step 4: Executing the mapping

You are now ready to call the Web service. Click the **Output** button to execute the mapping and preview the generated output. If the Web service call is executed successfully, the **Output** tab displays the returned data structure. If the call was not successful, MapForce returns the error

accordingly.

If you have MapForce Server (http://www.altova.com/mapforce/mapforce-server.html), you can also compile the mapping to a mapping execution file (*.mfx) and execute it from the command-line or from the MapForce Server API on the server machine where MapForce Server runs (see Compiling Mappings to MapForce Server Execution Files).

You have now finished creating a generic HTTP Web service call that uses a GET method to read data from an RSS feed. For general information about working with generic Web services, see also Adding a Web Service Call (REST-Style).

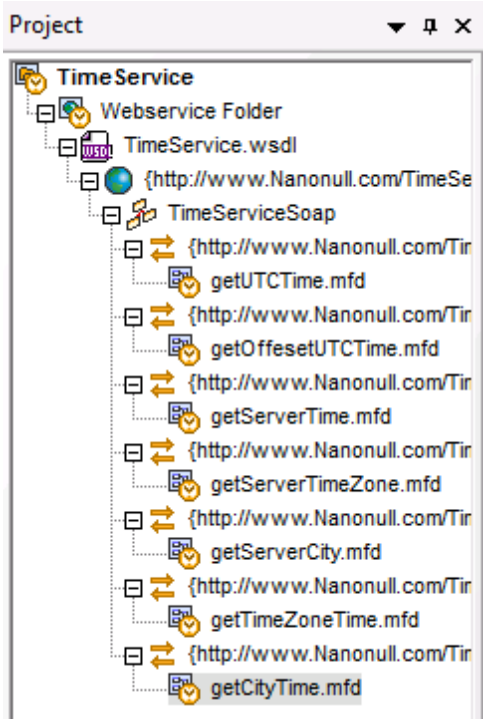
10.8 Example: Calling a WSDL-Style Web Service

This example shows you how to query a Web time service using a constant as an input. The Web service itself was implemented using MapForce. This is for demonstration purposes—you can implement the Web service with any other technology that supports a compatible protocol.

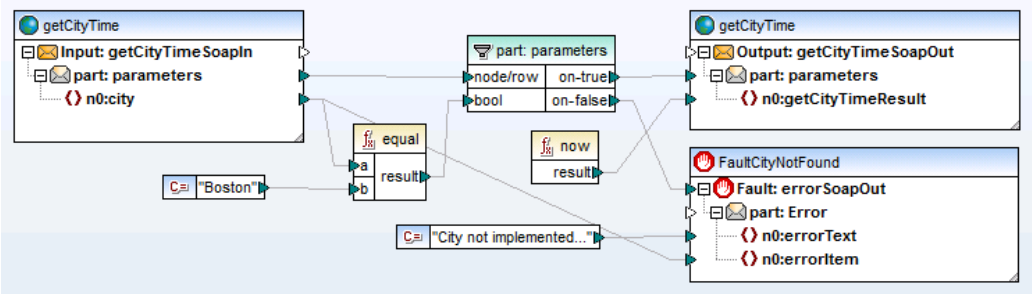
The mapping shown below is part of the **TimeService.mfp** mapping project, available in the **<Documents>\Altova\MapForce2016\MapForceExamples\TimeService** folder. The **TimeService2.mfp** project file available in the **<Documents>\Altova\MapForce2016\MapForceExamples\TimeserviceWsdl2** folder supports WSDL 2.0.

To view how the Web service is implemented:

1. Select **File | Open** and select the **TimeService.mfp** file in the **<Documents>\Altova\MapForce2016\MapForceExamples\Timeservice** folder. The files associated with the project are loaded in the Project window.



2. Double-click the **getCityTime.mfd** entry in the project window.

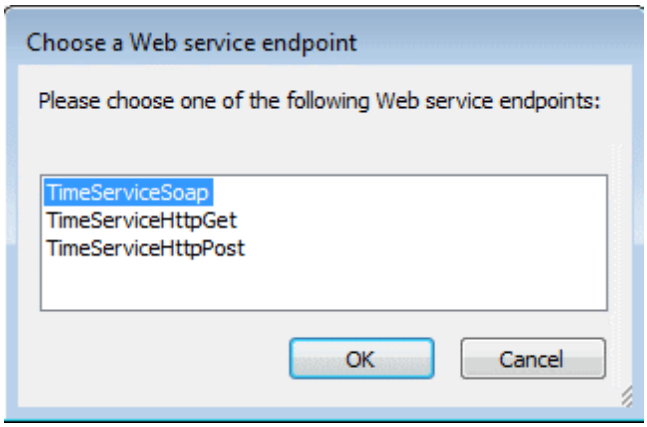


The **getCityTime.mfd** mapping accepts a city name as input and returns the current time in the output. If the city is not "Boston", a WSDL fault is returned. The mapping takes the input data from the **getCityTimeRequest.xml** file available in the **<Documents>\Altova\MapForce2016\MapForceExamples\TimeService** directory. If you double-click the **getCityTimeSoapIn** component, you can see that this file is set as data source.

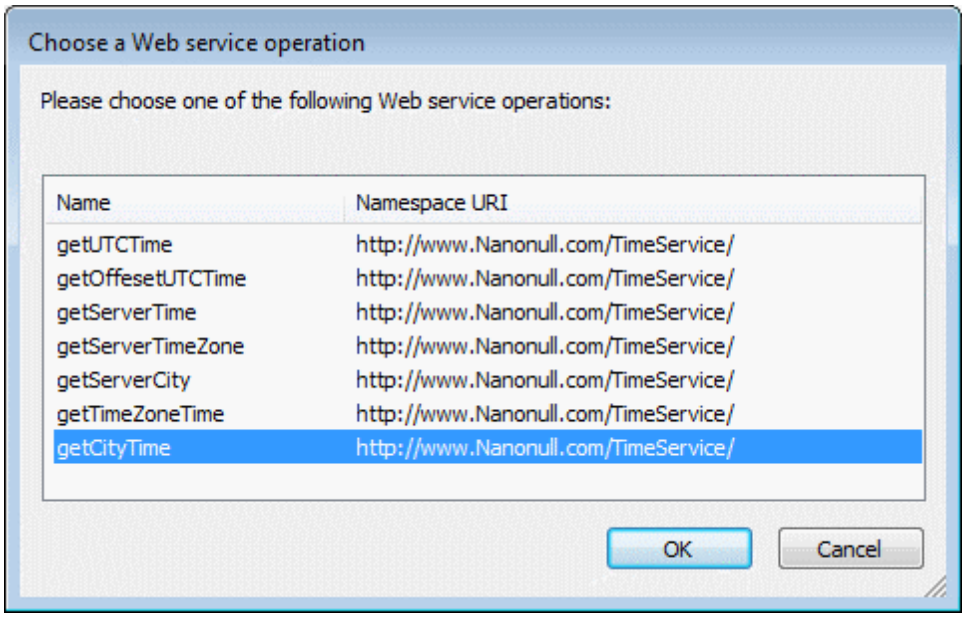
For the scope of this example, we will now assume that this particular Web service project has already been compiled and deployed to an actual Web server which you are going to call in the following steps of this example.

To call the "getCityTime" Web service function from a mapping:

1. Select **File | New**, click the **Mapping** icon and confirm with OK.
2. Select the menu option **Insert | Web service function...** or click the  toolbar button.
3. Click **Browse** to select the WSDL definition file; select **TimeService.wsdl** from the TimeService directory, then click the Open button.
4. When prompted to choose a Web service endpoint, click **TimeServiceSoap**.

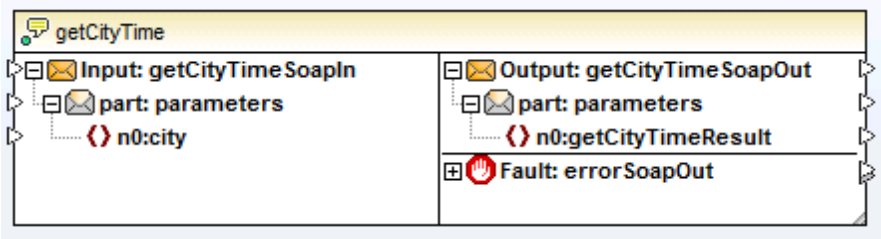


5. When prompted to choose a Web service operation, click **getCityTime**.



6. When prompted to specify the WSDL Call Settings, leave the settings unchanged, and click OK. For more information, see Web Service Call Settings.

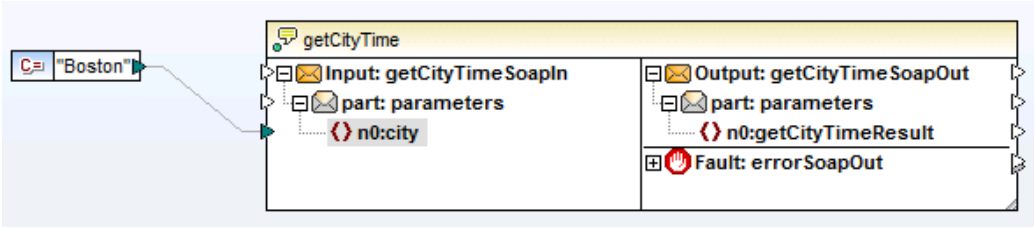
   The **getCityTime** Web service function is inserted as a single component. Note that it actually represents all eight components that make up the **getCityTime.mfd** file as saved in the WSDL project.



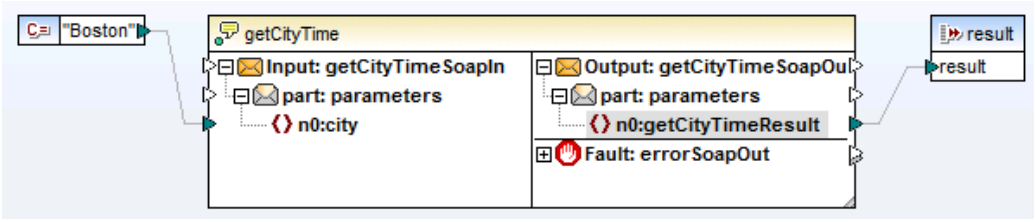
   The left section of the component defines the data input (SoapIn), while the right side defines the data output (SoapOut), which may also include a fault section, if one has been defined in the .wsdl file.

To define the input and output components for the Web service function:

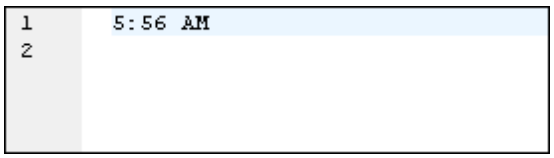
1. Insert the component that is to supply the input data, e.g. a constant, text, or schema component. For the scope of this example, insert a constant component, and enter "Boston" as the input string.



2. Connect the constant to the **n1:city** item.
3. Insert a simple output component (on the **Function** menu, click **Insert Output**).
4. Connect the n1:getCityTimeResult to the output component.



5. Click the Output tab to see the mapping output.

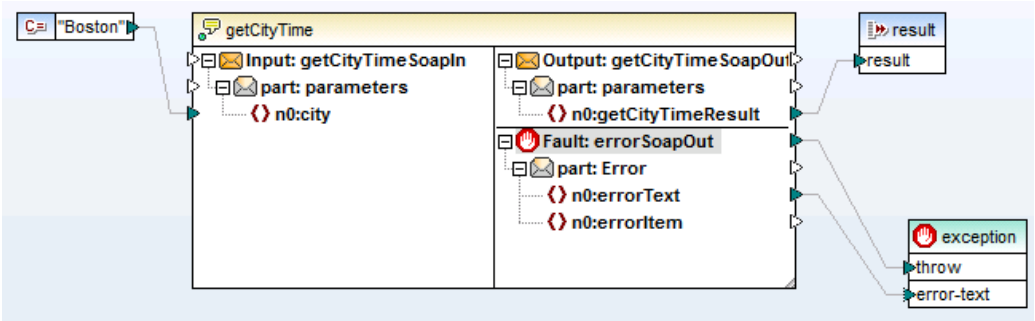


The current time in Boston is displayed in the Output window.

Note: The input value of the Web service function takes precedence over the data source of the original mapping. For example, the Constant component value "Boston" takes precedence over the **getCityTimeRequest.xml** data source file in the original mapping.

To map Web service faults:

1. Select **Insert | Exception**, or click the Exception toolbar button  .
2. Map the **Fault:** item to the **throw** item of the exception component.
3. Map the **n1:errorText** item to the **error-text** item of the exception component.



10.9 Digital Certificate Management

Digital certificate management is an integral part of secure data exchange between a client computer and a Web server. Since mappings can be executed not only on Windows by MapForce, but also on a Windows, Linux or Mac server by MapForce Server (either standalone or in FlowForce Server execution), this section deals with managing HTTPS certificates on various platforms.

In the context of secure HyperText Transport Protocol (HTTPS), it is important to distinguish between server and client certificates.

Server certificates

A server certificate is what identifies a server as a trusted entity to a client application such as MapForce. The server certificate may be digitally signed by a commercial Certificate Authority, or it may be self-signed by your organization. In either case, while designing the mapping in MapForce, you can specify the following settings:

- Whether the server certificate must be checked.
- Whether the request must proceed if a mismatch has been detected between the name certificate and the name of the host.

These settings are available on the HTTP Security Settings dialog box of MapForce (see Setting HTTP Security). When you enable server certificate checks, consider the following:

- If you are calling a Web server whose certificate is signed by a trusted Certificate Authority, your operating system will likely be already configured to trust the server certificate, and no additional configuration is necessary.
- If you are calling a Web server which provides a self-signed certificate (for example, a local network server within your organization), you will need to configure your operating system as well to trust that certificate.

In most cases, you can check the level of trust between your operating system and the Web server by typing the URL of the Web service in the browser's address bar. If the server is not trusted, or if your operating system is not configured to trust the server, your browser will display a message such as "This connection is untrusted", or "There is a problem with this website's certificate". Note that you cannot use the browser to check the level of trust with a Web server if the browser uses a certificate database other than that of the operating system (for example, Firefox 35.0.1 on Ubuntu 14.04).

On Windows, you can establish trust with the server by following the browser's instructions and importing or installing the required certificates into your system's Trusted Root Authorities store (see Trusting Server Certificates on Windows). On Mac, you can do the equivalent operation in Safari (see Trusting Server Certificates on Mac). For instructions applicable to Linux, see Trusting Server Certificates on Linux.

Client certificates

While server certificates are used to identify a server as a trusted entity, client certificates are primarily used to authenticate the caller against the Web server. If you intend to call a Web server which requires client certificates, you may need to contact the administrator of the Web server for

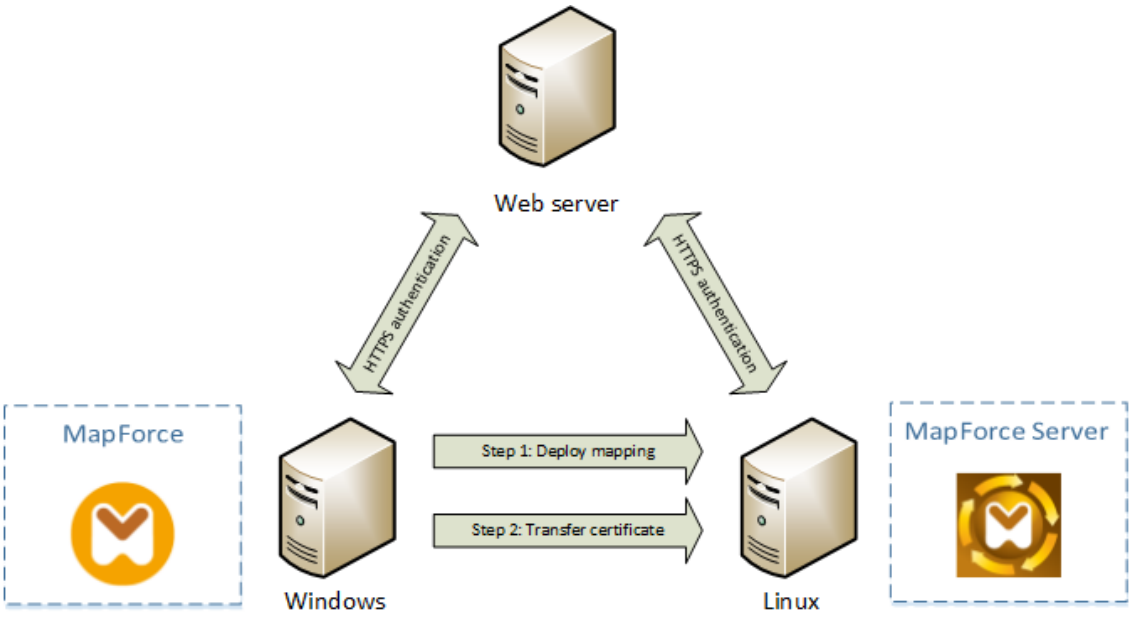
the client configuration instructions. Taking IIS (Internet Information Services) as an example, the Web server may be configured to handle HTTPS and client certificates in one of the following ways:

- Require HTTPS and ignore client certificate
- Require HTTPS and accept client certificate
- Require HTTPS and require client certificate

The success or failure of the Web service request depends both on the configuration of the Web server and the client application. For example, if the Web server is configured to require a client certificate, then, for the call to be successful, the calling application must present a valid client certificate.

From a MapForce perspective, the same is true for mappings which include Web service calls through HTTPS. In particular, to run such mappings successfully, it is assumed that the Web server has been configured to accept or require the client certificate, and that the operating system where the mapping runs provides the correct client certificate to the Web server.

The diagram below illustrates a scenario where a client certificate used in MapForce is transferred to a Linux server running MapForce Server. Once the certificate has been transferred to the target operating system, MapForce Server can use it to authenticate itself against the Web server and execute the mapping successfully.



Deploying mappings with client certificates to another computer

For HTTPS authentication in Web service calls, MapForce is capable of using Transport Layer Security (TLS) on top of HTTP, which is the successor of Secure Sockets Layer (SSL) protocol. Note that fallback to SSL may occur if either the client implementation or the server does not support TLS.

To support Web calls with client certificate authentication on multiple platforms, MapForce (and MapForce Server) relies on the certificate management implementation of each platform, thus ensuring that certificate management is always in the scope of the underlying operating system.

Each operating system provides different support for certificate management, as shown in the table below.

Platform	Certificate management and implementation
Windows	On Windows, you can manage certificates using the Certificate snap-in (see Accessing the Certificate Stores on Windows). TLS support is available through the *Secure Channel* (also known as *SChannel*) library.
Linux	On Linux, you can manage certificates using the OpenSSL (`openssl`) command line tool and library. If OpenSSL support is not already available on the Linux machine where MapForce Server is installed, you will need to download and install it before you can manage certificates. TLS support is available through the OpenSSL library (https://www.openssl.org/).
Mac	On Mac, you can manage certificates using the *Keychain Access Manager*, located under **Finder > Applications > Utilities**. TLS support is provided by the *Secure Transport* library native to the operating system.

If you execute the mapping on a Windows operating system where you can already successfully consume the same Web service that you intend to call from MapForce, no additional certificate configuration is normally required (for the conditions to run the mapping successfully on Windows, see Client Certificates on Windows). However, if you design mappings with MapForce on a Windows computer, and then deploy them to another computer (which may run a different operating system), the client certificate is not stored or copied together with the deployed package. For the Web service call (and the mapping) to execute successfully, the client certificate must exist on the target operating system as well.

To transfer a certificate from a Windows system to another Windows-based computer, export the required certificate (with private key) from the source system (see Exporting Certificates from Windows). Then import the same certificate to the **Current User\Personal** store on the target operation system (see Client Certificates on Windows).

For instructions on how to transfer client certificates to the Linux and Mac platforms, see Client Certificates on Linux and Client Certificates on Mac, respectively.

10.9.1 Trusting Server Certificates on Linux

To establish trust with a Web server on Linux, obtain the certificate file of the Web server, copy it to the system's certificate store, and then update the latter (see instructions below). One of the ways to obtain the server certificate is by using the Firefox browser, as shown in the example below.

Perform the following steps only if you are sure of the authenticity of the Web server

certificate.

Debian, Ubuntu

1. Copy the certificate file of the Web server to the following directory.

```
sudo cp /home/downloads/server_cert.crt /usr/local/share/ca-certificates/
```

2. Update the certificate store as follows:

```
sudo update-ca-certificates
```

Cent OS

1. Install the `ca-certificates` package:

```
yum install ca-certificates
```

2. Enable the dynamic certificate authority configuration feature:

```
update-ca-trust enable
```

3. Copy the server certificate to the following directory:

```
cp server_cert.crt /etc/pki/ca-trust/source/anchors/
```

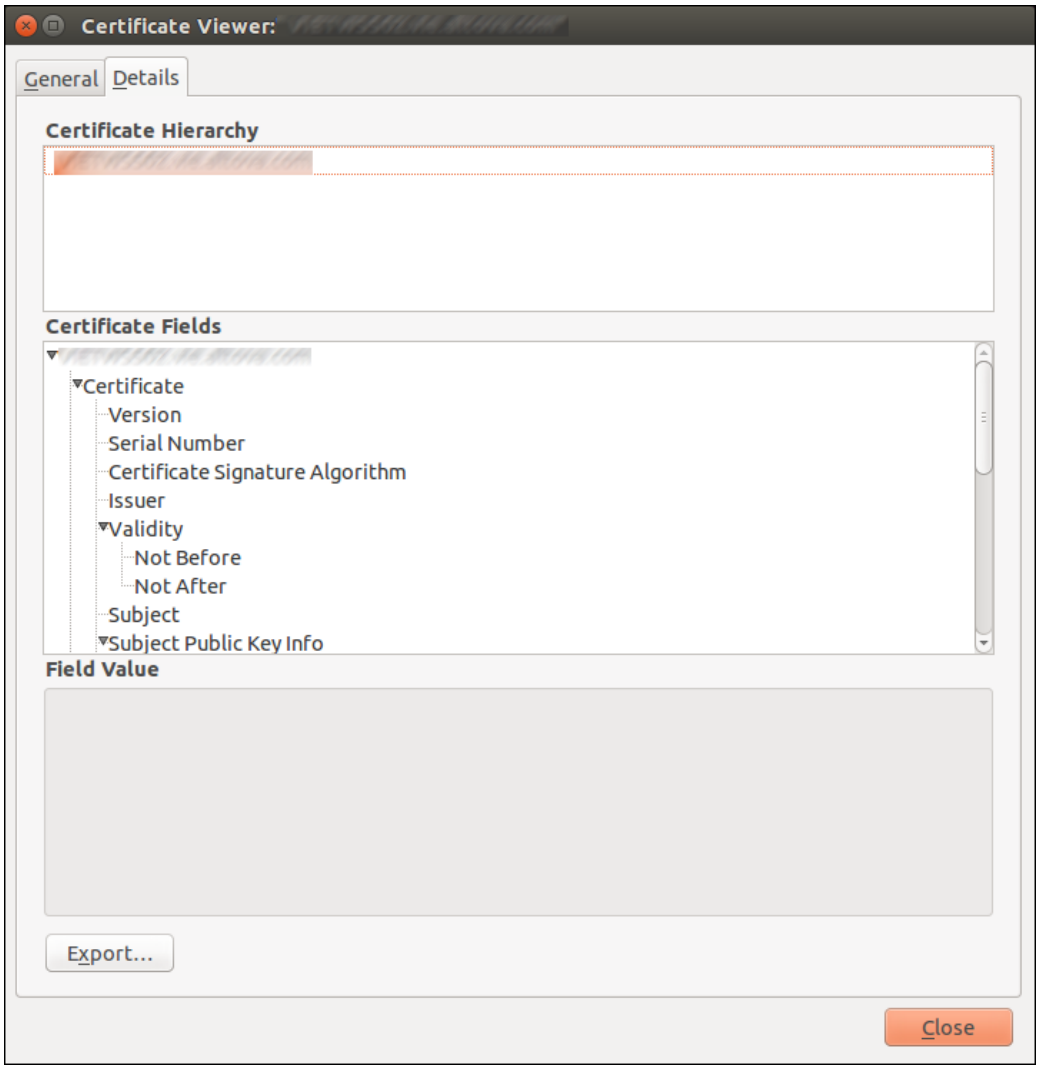
4. Use the command:

```
update-ca-trust extract
```

Example: Exporting the certificate of the Web server with Firefox on Ubuntu

1. Run Firefox and access the URL of the Web server.
2. When prompted with the message "This connection is untrusted", click **Add Security Exception**.

3. Click **View**.
4. Click the **Details** tab.



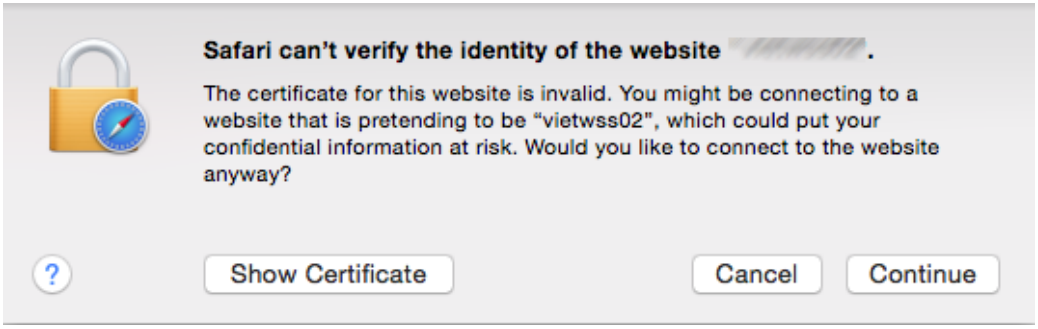
5. Click **Export** and save the certificate file to a local directory.

10.9.2 Trusting Server Certificates on Mac

To establish trust with a Web server using Safari:

> Perform the following steps only if you are sure of the authenticity of the Web server certificate.

1. In the browser address bar, enter the HTTPS address of the Web server.
2. When prompted to connect to the website, click **Show Certificate**.



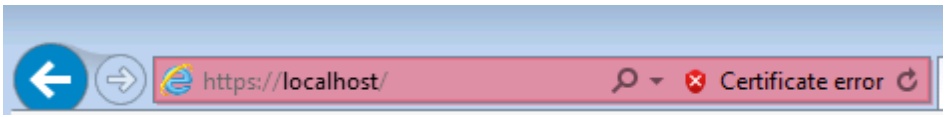
3. Select the option **Always trust {certificate} when connecting to {website}**.
4. Click **Continue** and enter your password when prompted.

10.9.3 Trusting Server Certificates on Windows

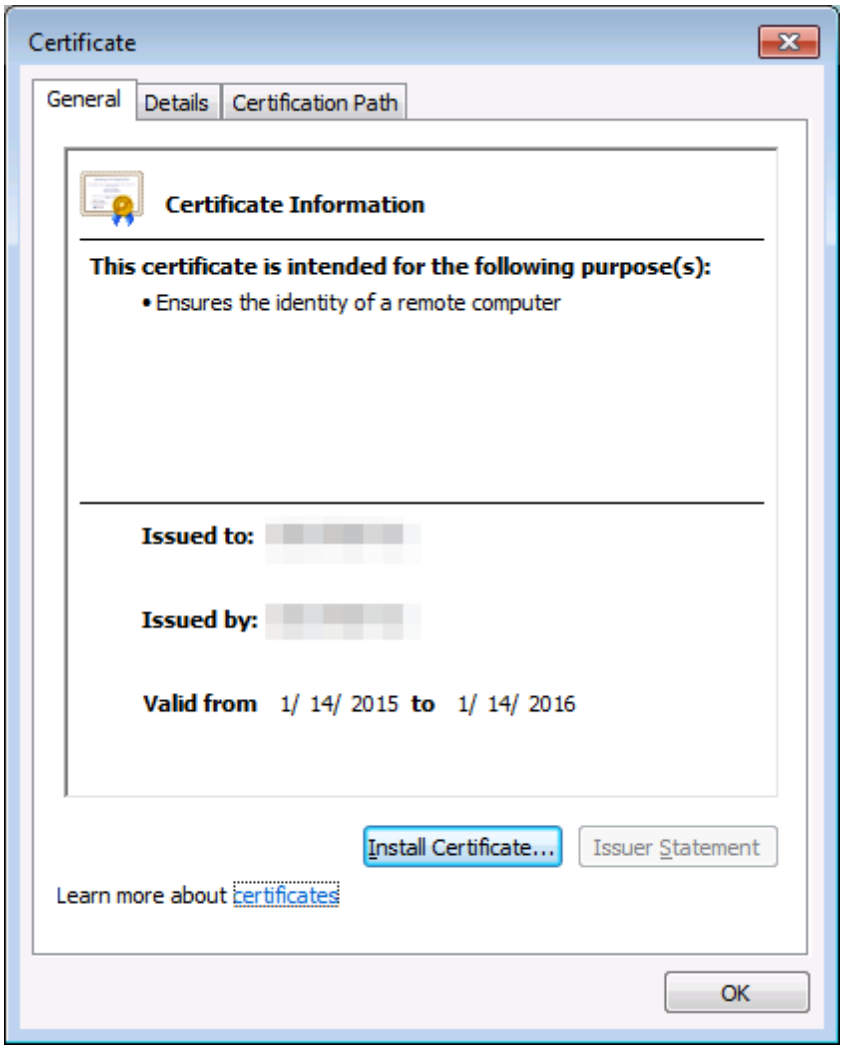
To establish trust with a Web server using Internet Explorer 11:

> Perform the following steps only if you are sure of the authenticity of the Web server certificate.

1. In the browser address bar, enter the HTTPS address of the Web server.
2. Click **Continue to this website (not recommended)**.
3. Click the Certificate error area, and then click **View certificates**.

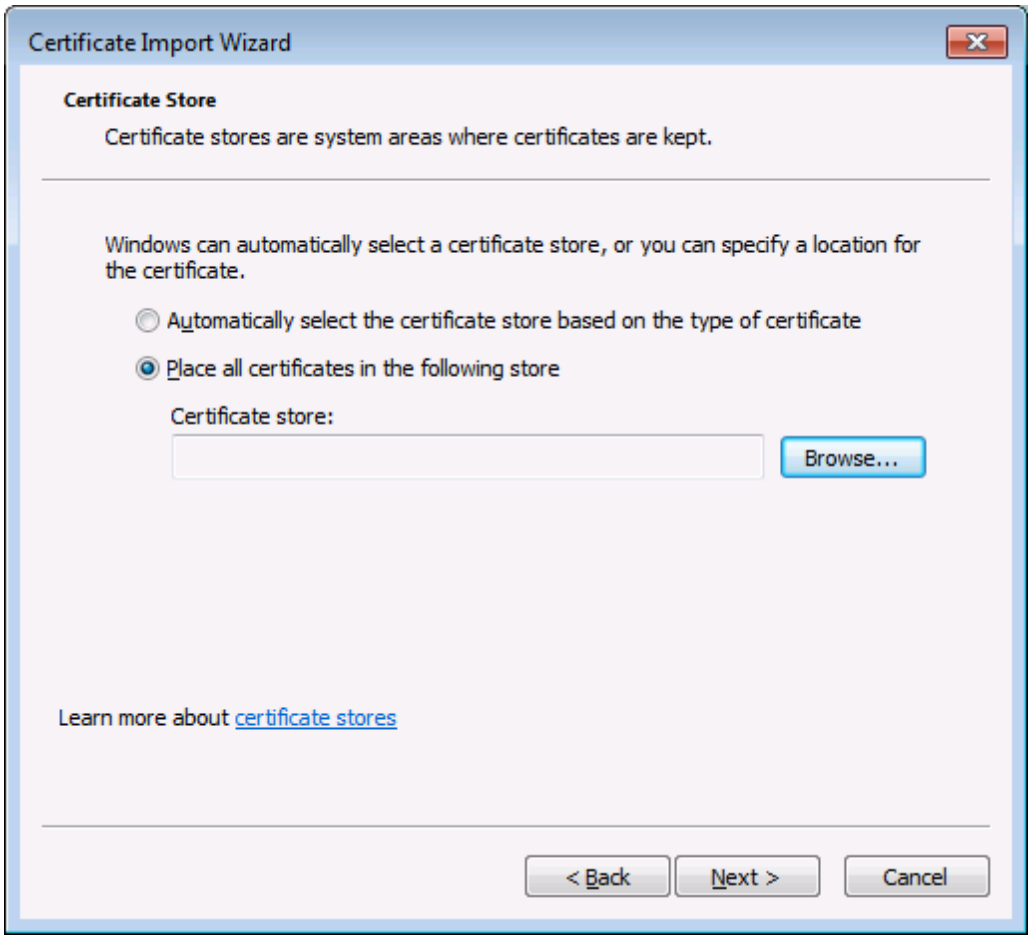


4. Click **Install Certificate**.

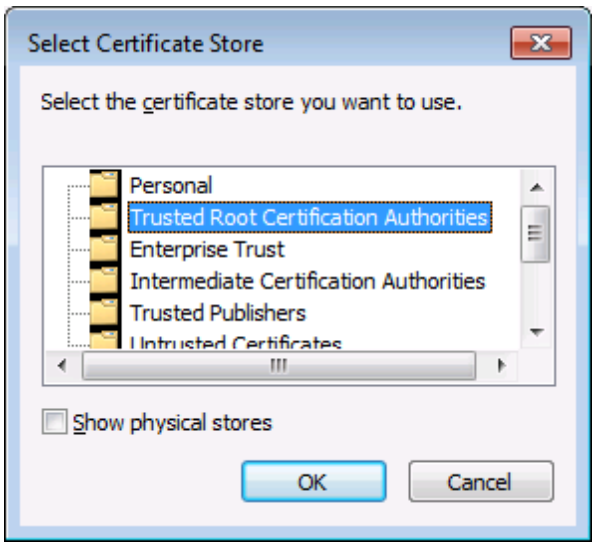

Certificate
General
Details
Certification Path
Certificate Information
This certificate is intended for the following purpose(s):
• Ensures the identity of a remote computer
Issued to:
Issued by:
Valid from 1/ 14/ 2015 to 1/ 14/ 2016
Install Certificate...
Issuer Statement
Learn more about certificates
OK

5. Click **Next**.

6. Choose to select a store manually.



7. Browse for the Trusted Root Certification Authorities, and then click OK.



8. When prompted to confirm your action, click OK.

10.9.4 Accessing the Certificate Stores on Windows

On Windows, you can manage certificates either from the *Certificates* Microsoft Management Console (MMC) snap-in, or from Internet Explorer.

To open the Certificates snap-in (for the current Windows user):

- Run `certmgr.msc` at the command line.

To open the certificate management dialog box in Internet Explorer:

1. On the **Tools** menu, click **Internet Options**.
2. Click the **Content** tab, and then click **Certificates**.

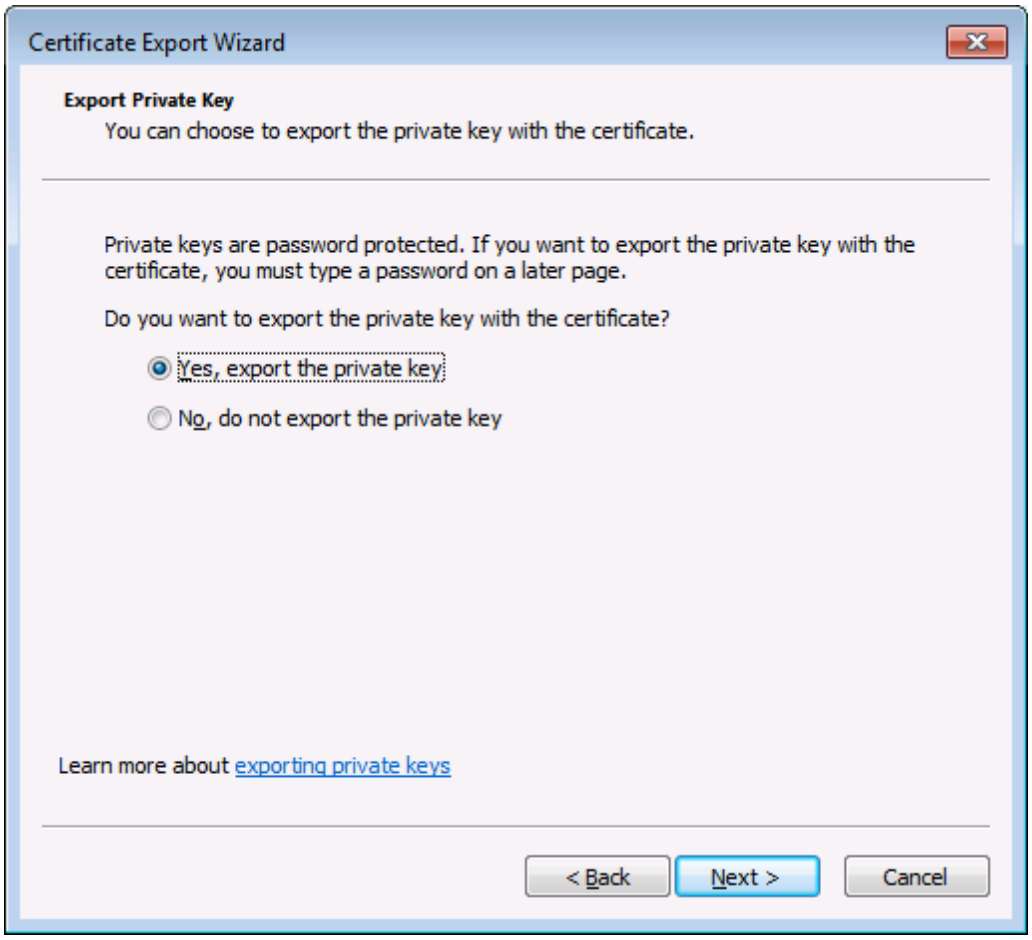
10.9.5 Exporting Certificates from Windows

For mappings that call Web services through HTTPS and are deployed to a Mac or Linux server running MapForce Server or FlowForce Server, the same client certificate must be available on the non-Windows operating system as the one used on Windows to design and test the mapping. To execute such mappings on a non-Windows operating system with MapForce Server, export the required certificate with private key from Windows and then import it into the target operating system.

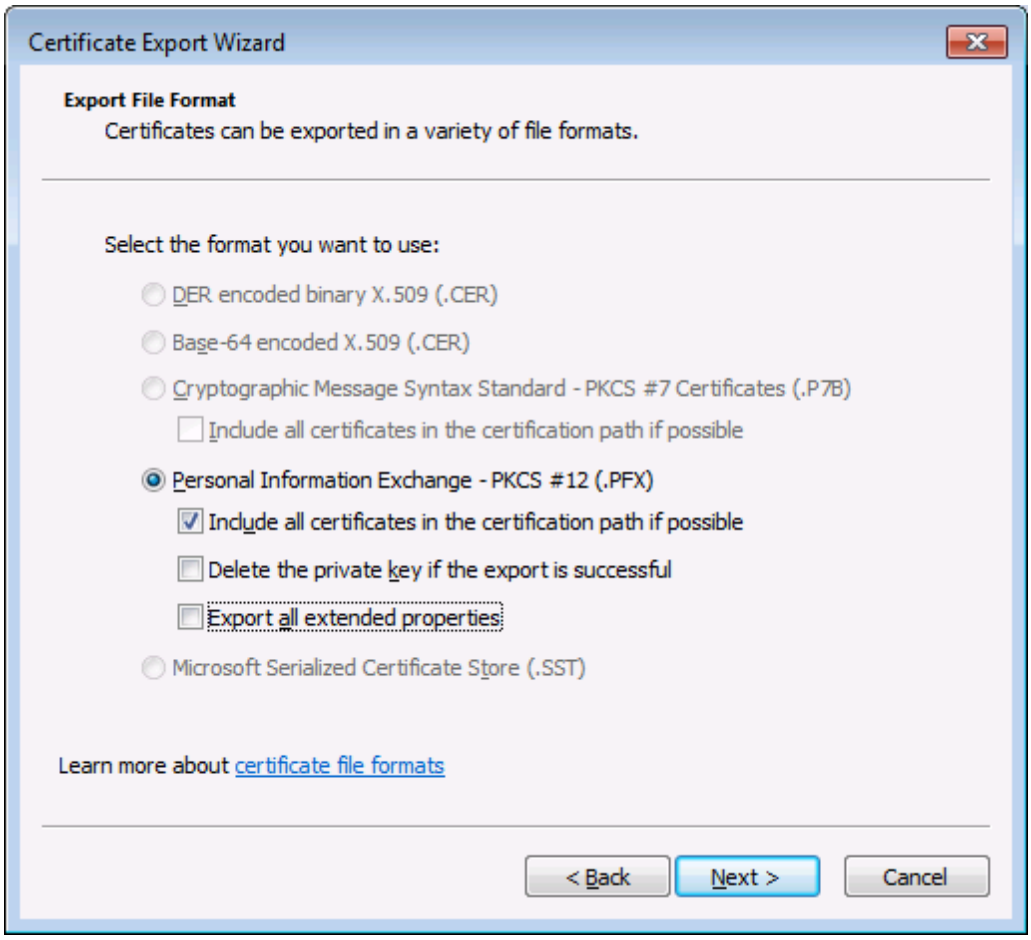
To export a certificate with private key from Windows:

1. On Windows, open the Certificates snap-in (see Accessing the Certificate Stores on Windows).
2. Right-click the certificate that you want to export, point to **All Tasks**, and then click **Export**.
3. Click **Next**.

4. Choose to export from Windows the certificate together with its private key, and then click **Next**.

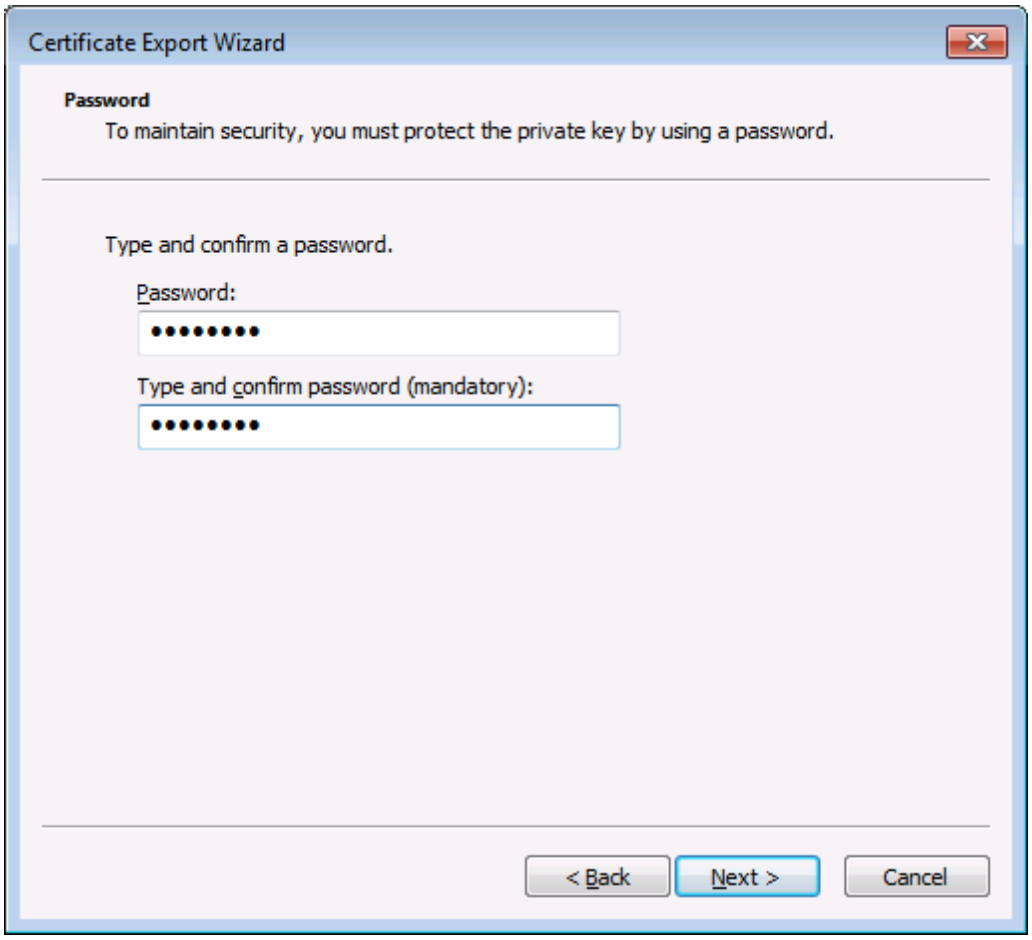


5. Choose the *Personal Information Exchange - PKCS #12 (.pfx)* file format, and then click **Next**.

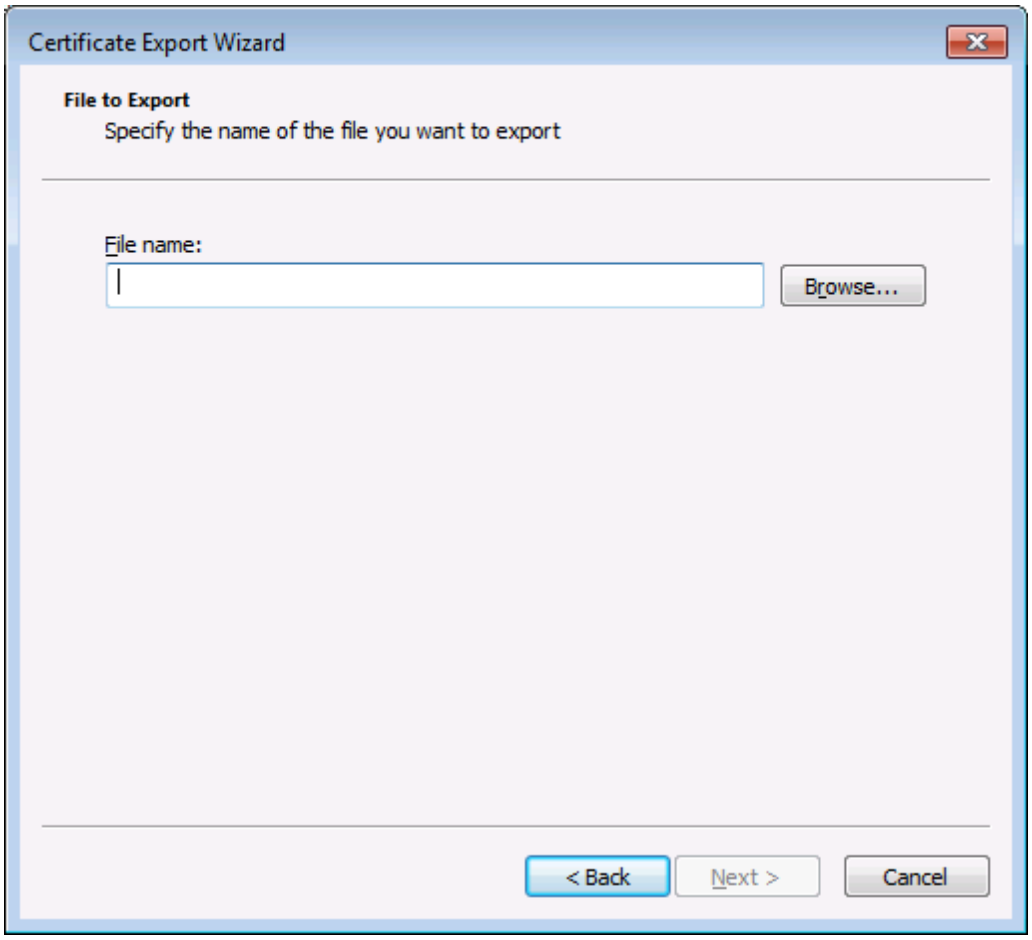


Note: Make sure not to select the option **Delete the private key if the export is successful**, otherwise you will not be able to make use of the certificate after it is exported.

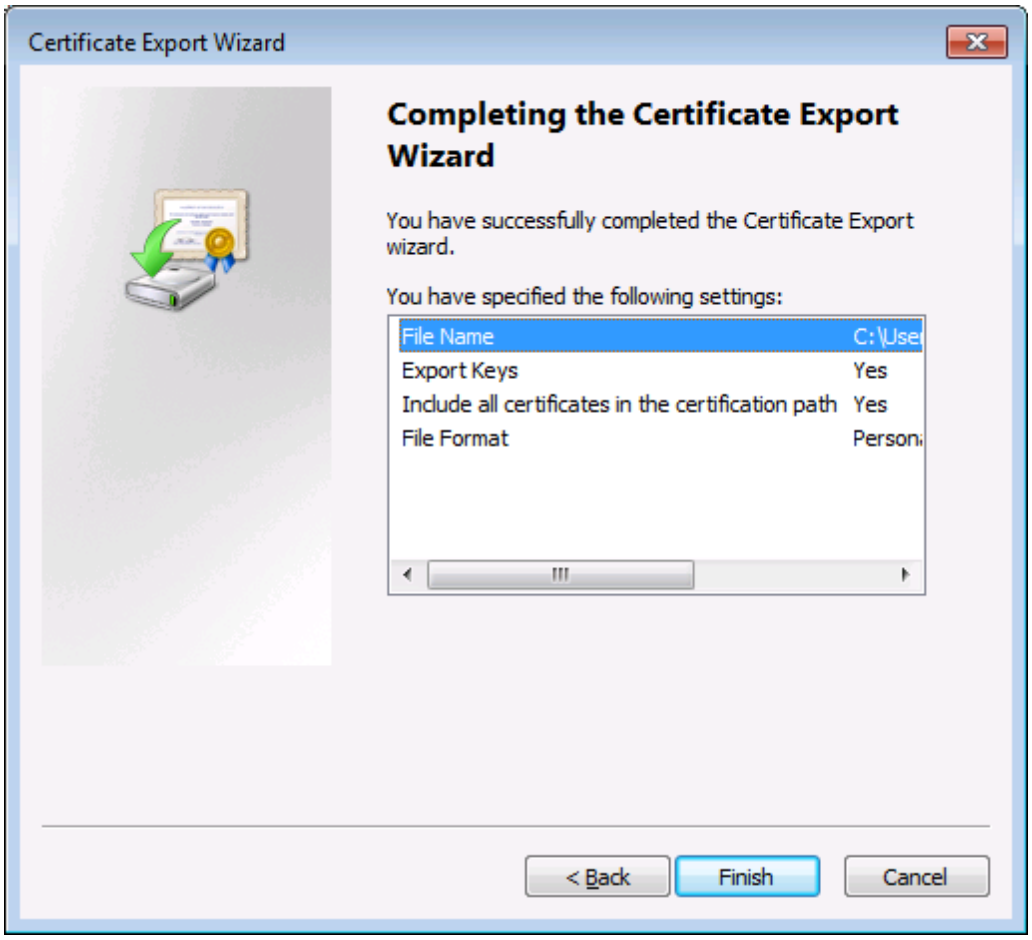
6. Enter a password, and then click **Next**. You will need this password after you copy the certificate to the target operating system.



7. Browse for the location of the file to export, and then click **Next**.



8. Click **Finish**.



10.9.6 Client Certificates on Linux

If your mappings include Web service authentication through HTTPS by means of client certificates, follow these steps to deploy such mappings to a Linux machine running MapForce Server:

1. In the HTTP Security Settings dialog box, click **Client Certificate**, and then select the required certificate from the **Current User\Personal** store on Windows (see Setting HTTP Security).
2. Save and deploy the mapping to the target operating system (see Deploying a MapForce mapping).
3. Transfer the client certificate required by the Web service call to the target operating system. Make sure that the certificate has a private key, and that the **Enhanced Key Usage** property of the certificate includes "Client authentication" as purpose.

To transfer the client certificate to Linux:

1. Export the client certificate with private key from Windows, in the *Personal Information Exchange - PKCS #12 (.pfx)* file format (see Exporting Certificates from Windows).

2. Copy the certificate file to the Linux machine.
3. Convert the .pfx file to .pem format using the command:

```
openssl pkcs12 -in cert.pfx -out "John Doe.pem" -nodes
```

This command parses the .pfx file and outputs a .pem file, without encrypting the private key. Certificates with an encrypted private key prompt for password and are not supported in server execution.

Executing the mapping

To instruct MapForce Server to use the .pem file as client certificate, set the `--certificatespath` parameter when running the mapping. The `--certificatespath` parameter defines the path of the directory where all certificates required by the current mapping are stored. For example, if the certificate file path is `/home/John/John Doe.pem`, then `--certificatespath` must be set to `/home/John`.

By default, if the `--certificatespath` parameter is not provided, MapForce Server looks for certificates in the directory `$HOME/.config/altova/certificates` of the current user.

> For the mapping to execute successfully, the certificate file is expected to have the .pem extension and the file name must match the Common Name (CN) of the certificate, including spaces (for example, **John Doe.pem**). If the CN contains a forward slash (/), it must be replaced with an underscore (_) character.

If you intend to execute the mapping as a FlowForce Server job, copy the certificate file to the `$HOME/.config/altova/certificates` directory. When running the job, FlowForce Server will use this directory to look for any certificate files required by the mapping.

> For security considerations, make sure that certificate files are not readable by other users, since they contain sensitive information.

10.9.7 Client Certificates on Mac

If your mappings include Web service authentication through HTTPS client certificates, follow these steps to deploy such mappings to a OS X running MapForce Server:

1. In the HTTP Security Settings dialog box of MapForce, click **Client Certificate**, and then select the required certificate (see Setting HTTP Security).
2. If the certificate name does not match exactly the host name of the server, select **Allow name mismatch between certificate and request**.
3. Save and deploy the mapping to the target operating system (see Deploying a MapForce mapping).
4. Transfer the client certificate required by the Web service call to the target operating system. Make sure that the certificate has a private key, and that the **Enhanced Key Usage** property of the certificate includes "Client authentication" as purpose.

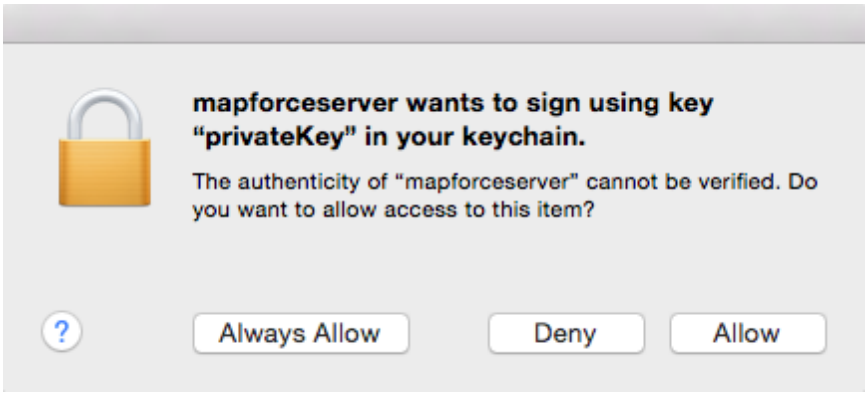
To transfer the client certificate to Mac:

1. Export the client certificate with private key from Windows, in the *Personal Information Exchange - PKCS #12 (.pfx)* file format (see Exporting Certificates from Windows) and copy the .pfx file to the Mac.
2. If this hasn't been done already, make sure that the operating system trusts the server certificate (see Trusting Server Certificates on Mac OS).
3. Run Keychain Access from **Finder > Applications > Utilities**.
4. On the **File** menu, click **Import Items**.
5. Browse for the the client certificate exported from Windows in step 1 and select a destination keychain.
6. Click **Open** and enter the password with which the certificate was encrypted.

Executing the mapping

You are now ready to run the mapping using the MapForce Server `run` command. Note the following:

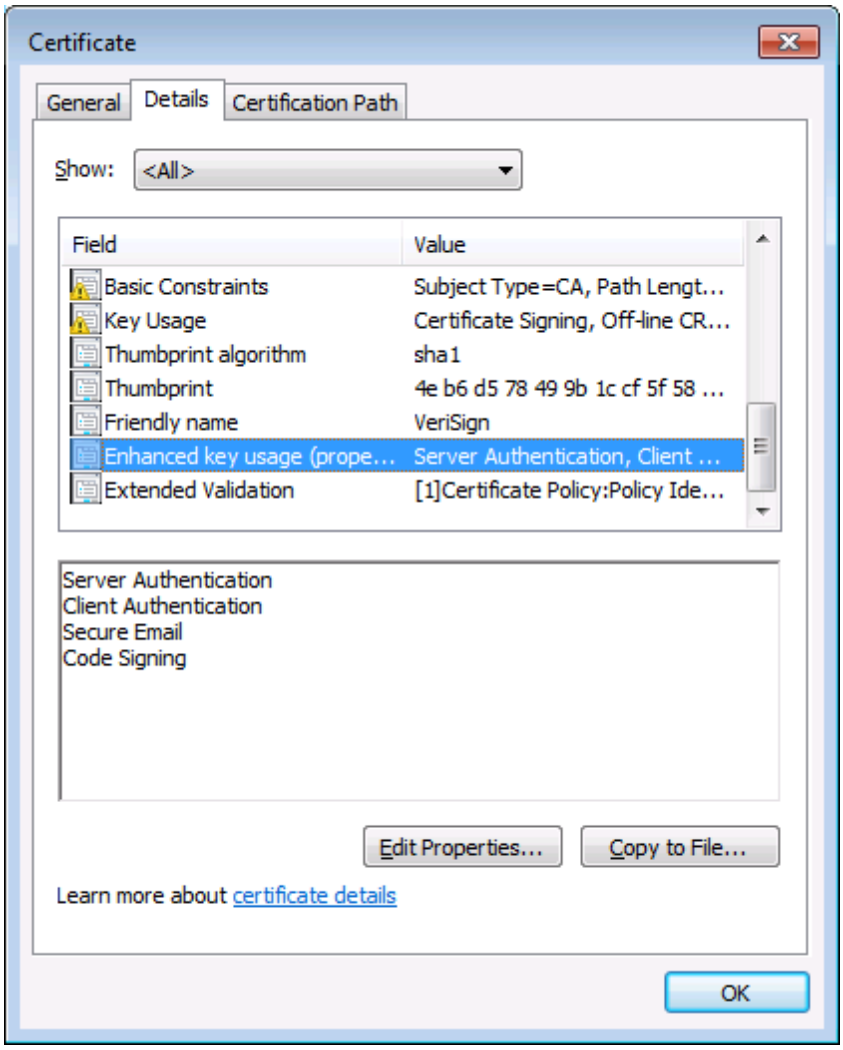
- If you execute the mapping remotely through SSH, first unlock the keychain with the `security unlock-keychain` command.
- If you execute the mapping through the Mac OS graphical user interface, when prompted to allow MapForce Server access to the keychain, click **Allow**.



10.9.8 Client Certificates on Windows

When you run on Windows a mapping which requires client certificates, the conditions to run the mapping successfully are as follows:

- The client certificate must exist in the **Current User\Personal** certificate store (also referred to as the **My** store). For the certificate to exist in this store, it must be imported through the Certificate Import Wizard. For instructions, see http://windows.microsoft.com/en-us/windows/import-export-certificates-private-keys#1TC=windows-7.
- The certificate must have a private key.
- The **Enhanced Key Usage** property of the certificate must include "Client authentication" as purpose.



In the current version of MapForce, due to a limitation of the library used by MapForce, Windows will select the required certificate automatically from the certificate store when you run the mapping. The mapping will execute successfully if, after filtering the **Current User\Personal** certificate store, the server finds a suitable certificate. Note that the HTTPS authentication (and the certificate selection operation) is managed by Windows and is not controlled by MapForce or MapForce Server. In some cases, if multiple certificates exist in the **Current User\Personal** store, an unsuitable certificate may be selected automatically by the operating system, which causes the mapping execution to fail. This situation can be avoided by limiting the number of certificates available in the **Current User\Personal** store.

Chapter 11

Automating Mappings and MapForce

11 Automating Mappings and MapForce

This section describes the command line interface of MapForce and ways to automate mapping execution with Altova server tools: RaptorXML Server, MapForce Server, and FlowForce Server.

11.1 About MapForce Server

MapForce Server (http://www.altova.com/mapforce/mapforce-server.html) is an enterprise server product that runs on high-speed servers running MS Windows, Linux and Mac OS X operating systems. It operates either as a module of FlowForce Server (http://www.altova.com/flowforce.html) or in standalone mode.

MapForce Server allows you to automate mappings using FlowForce Server. This entails deploying a MapForce mapping, defining a FlowForce job, and executing that job to produce the mapping output (see Deploying Mappings to FlowForce Server).

If MapForce Server is used as a standalone product then the MapForce mapping has to be compiled (see Compiling Mappings to MapForce Server Execution Files). 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.

Limitations:

- XML digital signatures are not supported
- Altova Global Resources (http://www.altova.com/global-resources.html) are not supported by the API; this limitation does not apply to the command line interface
- ODBC and ADO database connections are supported only on Windows (for other operating systems, see Database Connections on Linux and Mac).

For more information, refer to the MapForce Server documentation (http://manual.altova.com/MapForceServer).

11.2 Compiling Mappings to MapForce Server Execution Files

To compile a mapping via the MapForce command line:

- Execute MapForce and specify the mapping file and the `/COMPILE` command line option. The MapForce Server Execution file will be created in the same directory as the mapping file.

To compile a mapping using the MapForce GUI:

1. Open a mapping in MapForce e.g. **myMapping.mfd**.
2. Select the menu option **File | 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 at that location.

For information on how to run the MapForce Server Execution file, see the Run command of the MapForce Server documentation.

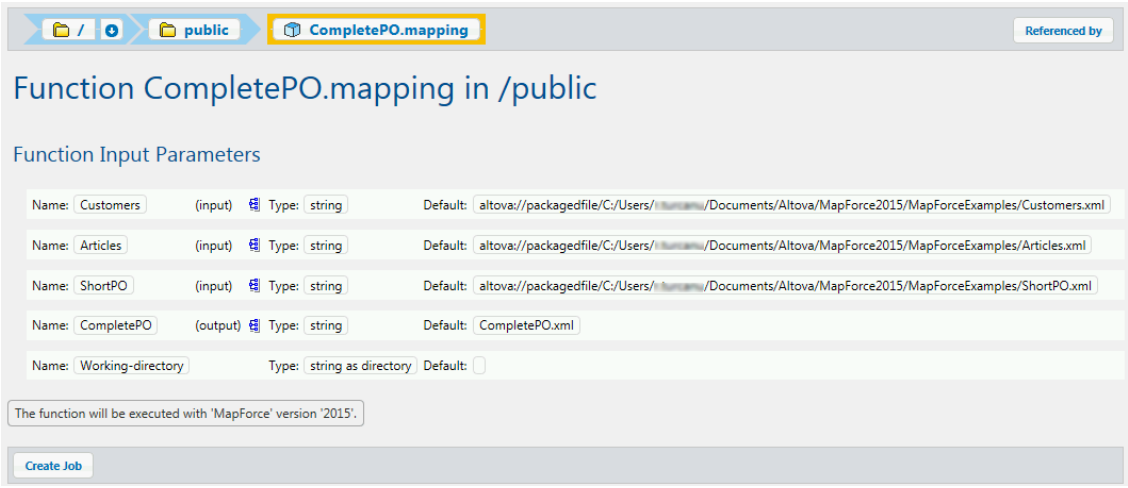
11.3 Deploying Mappings to FlowForce Server

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 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.

Since FlowForce Server does not necessarily run on the same operating system as the one where the MapForce mapping was designed, note the following when deploying mappings:

- The configuration of the operating system where FlowForce Server runs must allow for the mapping to be executed. For example, if the mapping includes database components which require specific database drivers, such drivers must exist on the target server.
- When you deploy a mapping to non-Windows platforms, ADO and ODBC database connections are changed to JDBC. For further information about database connectivity and drivers, see Connecting to a Database.
- If the mapping contains custom function calls (for example, to .dll or .class files), such dependencies are not deployed together with the mapping, since they are not known before runtime. In this case, you can copy them manually to the target server.

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.



Sample mapping function in FlowForce Server

From this point, you can create a full-featured FlowForce Server job from the deployed mapping, 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).

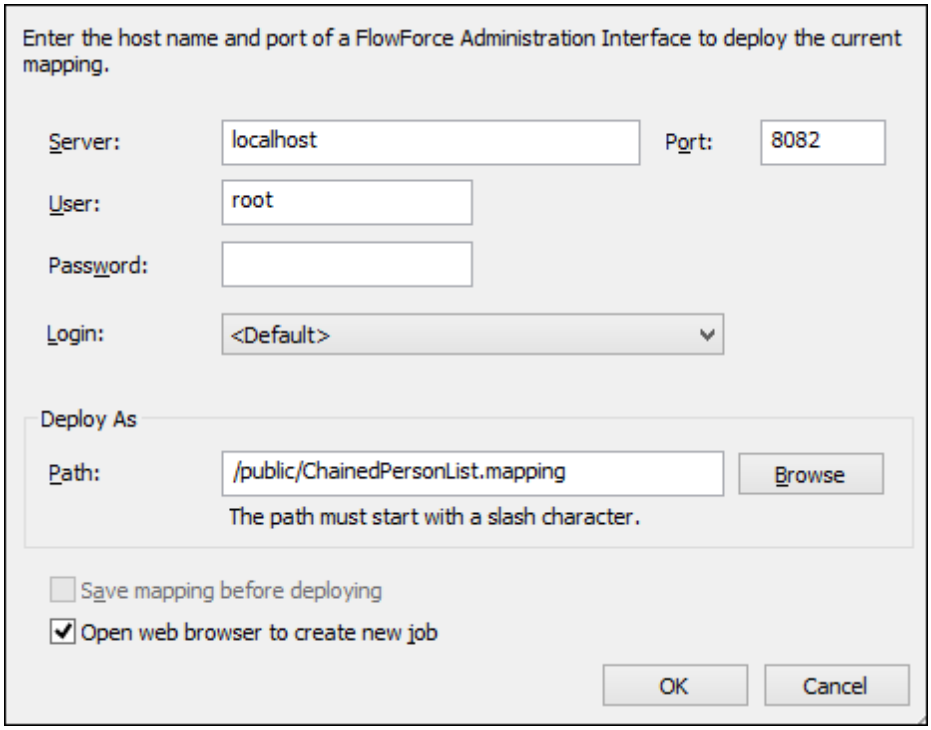
The prerequisites for deploying and running MapForce mappings as FlowForce Server jobs are as follows:

- Required licenses: MapForce Enterprise or Professional edition, MapForce Server, FlowForce Server

- FlowForce Server is running at the configured network 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).

To deploy a MapForce mapping to FlowForce Server:

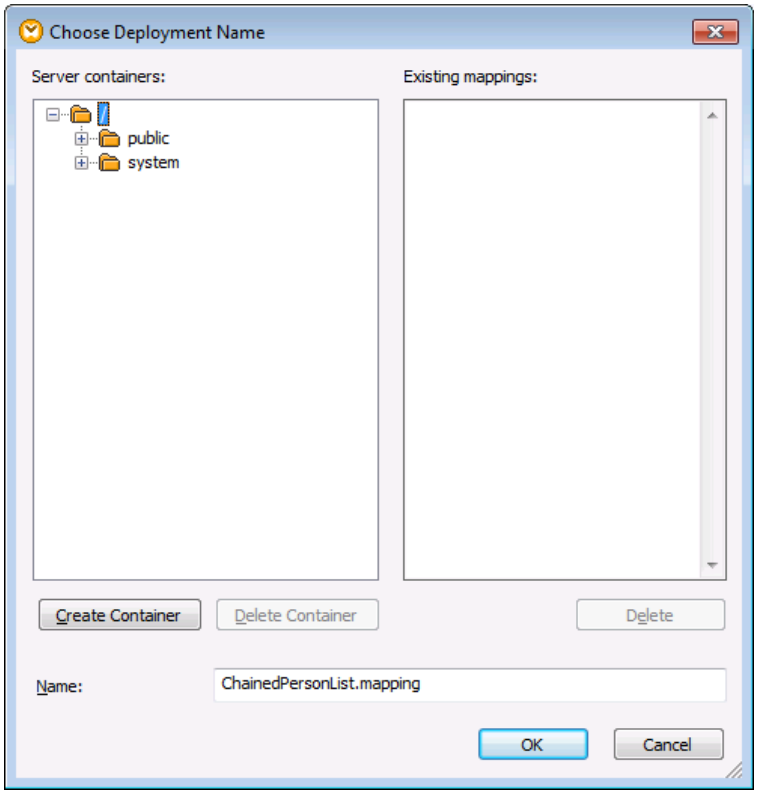
1. Ensure that the transformation language is set to BUILT-IN (see Selecting a transformation language).
2. On the **File** menu, click Deploy to **FlowForce Server**. The Deploy Mapping dialog box opens.



3. Enter your deployment settings (as described below), and click OK. If you selected the **Open web browser to create new job** check box, the FlowForce Server administration interface opens in the browser, and you can start creating a FlowForce Server job immediately.

The following table lists the mapping deployment settings available on the Deploy Mapping dialog box.

Setting	Description
Server and Port	Enter the server host name (or I.P. 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.
User and Password	The user name and password to be entered depends on the value of the Login drop-down list (see next option). If the Login drop-down list is set to **<Default>** or **Directly**, enter your FlowForce Server user name and password. Otherwise, enter your Windows user name and password, and select the Windows domain name from the Login drop-down list.
Login	If Windows Active Directory integration is enabled in FlowForce Server, select the Windows domain name from this drop-down list, and enter your Windows credentials in the User and Password fields (see previous option).
Path	Click **Browse**, and select the path where the mapping function should be saved in FlowForce Server container hierarchy. By default, the path is set to the **/public** container of FlowForce Server. From the Choose Deployment Name dialog box, you can also create new containers or delete existing containers and mappings, provided that you have the required FlowForce Server permissions and privileges.  *Choose Deployment Name dialog box*

Save mapping before deploying	This option is available if you are deploying an unsaved mapping. Select this check box to save the mapping before deployment.
Open browser to create new job	If you select this check box, the FlowForce Server Web administration interface opens in the browser after deployment, and you can start creating a FlowForce Server job immediately.

11.4 Automation with RaptorXML Server

RaptorXML Server (hereafter also called RaptorXML for short) is Altova's third-generation, super-fast XML and XBRL processor. It has been built to be optimized for the latest standards and parallel computing environments. Designed to be highly cross-platform capable, the engine takes advantage of today's ubiquitous multi-core computers to deliver lightning fast processing of XML and XBRL data.

RaptorXML is available in several editions which can be downloaded and installed from the Altova download page:

- RaptorXML Server is a very fast XML processing engine with support for XML, XML Schema, XSLT, XPath, XQuery, and more. This edition is part of the FlowForce Server installation package.
- RaptorXML+XBRL Server supports all the features of RaptorXML Server with the additional capability of processing and validating the XBRL family of standards.

Limitations:

- XML Signatures are not supported
- Global resources are not supported via the COM interface
- ODBC and ADO database connections are only supported on Windows. On other operating systems, JDBC must be used.

If you generate code in XSLT 1.0 or 2.0, or in XQuery, MapForce creates a batch file called `DoTransform.bat` which is placed in the output folder that you choose upon generation. Executing the batch file calls RaptorXML Server and executes the XSLT/XQuery transformation on the server.

If you intend to execute or automate MapForce mappings for other outputs on a server, refer to Altova MapForce Server and FlowForce Server.

Note: You can also preview the XSLT and XQuery code using the built-in engine.

11.5 MapForce Command Line Interface

General command line syntax:

MapForce.exe *filename* [**/***target* [*outputdir*] *options*

- Square brackets **[...]** denote optional parameters.
- Curly brackets **{...}** denote a parameter group containing several choices.
- The **pipe** symbol **|** denotes OR, e.g. /XSLT or /JAVA

MapForce.exe returns an exit code of 0, if the command line execution was successful. Any other value indicates a failure. You can check for this code using the IF ERRORLEVEL command in batch files.

filename
The MFD or MFP file to load. **If the path, or file name contains a space, please use quotes around the path/file name i.e. "c:\Program Files\...\Filename"**

target
The code generation target:

/COMPILE:*compileoptions*	compiles a mapping to a **MapForce Server Execution file** (.mfx) *compileoptions* **JDBC**, **NOXMLSIGNATURES** **JDBC** transforms all database connections to JDBC using the JDBC driver and the database URL defined in the database Component Settings dialog box. **NOXMLSIGNATURES** suppresses the generation of digital signatures in the MapForce Server Execution file (not supported by MapForce Server)
/GENERATE	generates project code for all mappings in the project file using the current folder settings. The project file *.MFP must be used as filename
/XSLT	generates XSLT 1.0 code
/XSLT2	generates XSLT 2.0 code
/XQuery	generates XQuery 1.0 code
/JAVA	generates a Java application
/CS	generates a C# application using the configuration of the mapping settings
/CS:*csoptions*	generates a C# application using the specified options *csoptions* { **VS2010** | **VS2008** | **VS2005** }

VS2010 generates a C# application with solution file for Visual Studio 2010

VS2008 generates a C# application with solution file for Visual Studio 2008

VS2005 generates a C# application with solution file for Visual Studio 2005

/CPP generates a C++ application using the configuration of the mapping settings

/CPP:*cppoptions* generates a C++ application using the specified options

cppoptions { **VC10** | **VC9** | **VC8** },{ **MSXML** | **XERCES** | **XERCES3** }, { **LIB** | **DLL** },{ **MFC** | **NoMFC** }

VC10 generates a C++ application with solution file for Visual Studio 2010

VC9 generates a C++ application with solution file for Visual Studio 2008

VC8 generates a C++ application with solution file for Visual Studio 2005

MSXML generates C++ code using MSXML 6.0

XERCES generates C++ code using Apache Xerces 2.x (2.6 and later)

XERCES3 generates C++ code using Apache Xerces 3.x

LIB generates C++ code for static libraries

DLL generates C++ code for dynamic-linked-libraries

MFC generates C++ code with MFC (Microsoft Foundation Classes) support

NoMFC generates C++ code without MFC support

outputdir
Specifies the directory where the generated mapping code is to be placed. This is optional. 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.

options
Specifies various options:

/**GLOBALRESOURCEFILE** *grfilename* uses the global resources defined in the specified global resource file

/**GLOBALRESOURCECONFIG** *grconfig* uses the specified global resource configuration

/**LIBRARY** *libname* Use together with a code generation target

	language to specify additional function libraries. This option can be specified more than once to load multiple libraries. These libraries are temporarily (for this one run) added to the libraries from Tools->Options->Libraries.
/**LOG** *logfilename*	Generates a log file. *logfilename* can be a full path name, i.e. directory and file name of the log file, but the directory must exist for the logfile to be generated if a full path is supplied. If you only specify the file name, then the file will be placed in the *outputdir* directory.

Notes:
Entering a relative path in one of the command line parameters is taken as being relative to the working directory, i.e. the current directory of the application calling MapForce. This applies to the path of the MFD file filename, outputdir, logfilename, and globalresourcefilename.

Defining an absolute path causes the working directory to be ignored. The absolute path is used as supplied.

Avoid using the end backslash and closing quote on the command line \" e.g. "C:\My directory\". These two characters are interpreted by the command line parser as a literal double quotation mark. Use the double backslash \\ if spaces occur in the command line and you need the quotes ("c:\My Directory\\"), or try to avoid using spaces and therefore quotes at all e.g. c:\MyDirectory.

Examples:

MapForce.exe *filename* starts MapForce and opens the file defined by *filename*.

I)
generate all XSLT files and output a log file.
MapForce.exe *filename.mfd* **/XSLT** *outputdir* **/LOG** *logfilename*

II)
generate a Java application and output a log file.
MapForce.exe *filename.mfd* **/JAVA** *outputdir* **/LOG** *logfilename*

III)
generate a C# application and output a log file.
MapForce.exe *filename.mfd* **/CS** *outputdir* **/LOG** *logfilename*

IV)
generate a C++ application using the code generation settings defined in the application options, and output a log file.
MapForce.exe *filename.mfd* **/CPP** *outputdir* **/LOG** *logfilename*

V)
generate a C++ application using the /CPP switch, overriding your C++ compiler options.
MapForce.exe *filename.mfd* **/CPP:(MSXML|XERCES),(LIB|DLL),(MFC|NoMFC)** *outputdir* **[/LOG** *logfilename* **]**

MapForce.exe *filename.mfd* **/CPP:MSXML,LIB,MFC**
Generates the C++ application using all of the first choices, in this example:

- compile for C++
- use MSXML
- generate code for static libraries
- have generated code support MFC

MapForce.exe *filename.mfd* **/CPP:XERCES,DLL,NoMFC** *outputdir* **/LOG** *logfilename*
Generates the C++ application using all of the second choices, in this example:

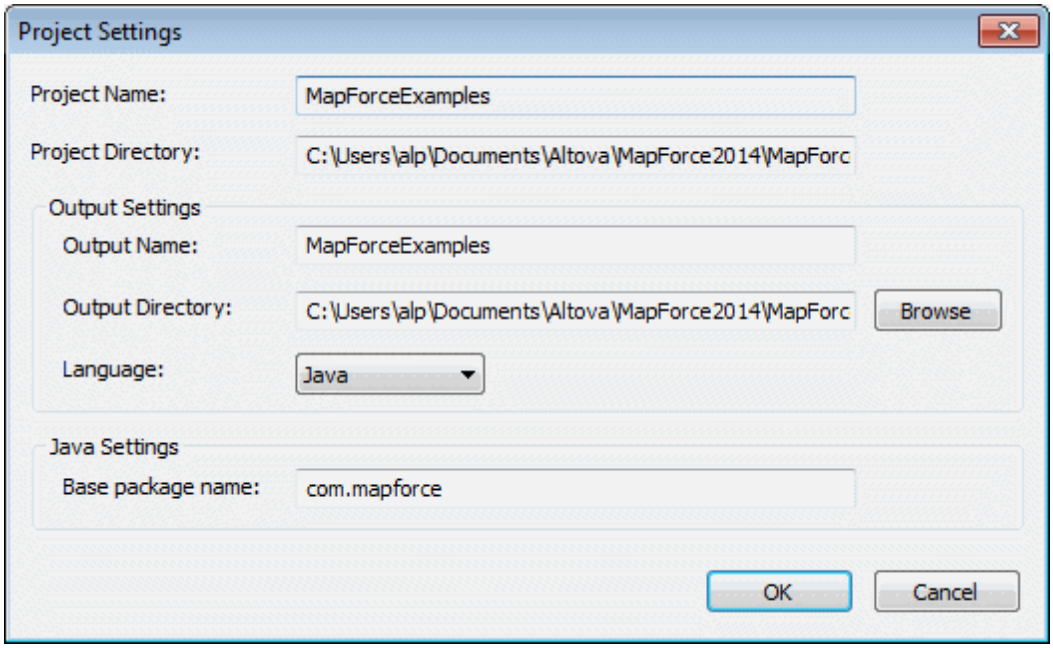
- compile for C++
- use XERCES
- generate code for dynamic libraries
- generated code not to support MFC
- create a log file in the outputdir with the name logfilename

VI)
generate all output files using global resources of the global resource file for the specified configuration
Mapforce.exe *filename.mfd outputdir* **/GLOBALRESOURCEFILE** *globalresourcefilename* **/GLOBALRESOURCECONFIG** *configurationname*

VII)
generate project code for all mappings in the **project file** using the current project folder settings
MapForce.exe *filename.mfp* **/GENERATE /LOG** *logfilename*

When generating code for the whole project, the project file name e.g. **MapForceExamples.mfp,** must be used as *filename.*
The code generation language as well as the output path for the mappings in each folder, are supplied by the folder Properties dialog box of **each** folder, when an *outputdir* is **not** specified.

If the *outputdir* parameter is used when generating project code, using GENERATE, then the path of *Outputdir* directory of the **project root** folder is overwritten by the entry you supply in the command line. This is the directory that is used when you select the "Use default project settings" radio button in the Properties dialog box of a new folder, when adding it to the project.



VIII)
generate project code in Java for all mappings in the project file
MapForce.exe *filename.mfp* **/JAVA /LOG** *logfilename*

The code generation language defined by the folder property settings are ignored, and Java is used for all mappings.

IX)
generate output files of a previously compiled (Java) mapping project, using the executable JAR file; and define the input and output files in the command line.
java -jar *mappingfile.jar* /**InputFileName** *inputfilename* /**OutputFileName** *outputfilename*

Note: the /InputFileName and /OutputFileName **parameters** are the names of special input components in the MapForce mapping that allow you to use them as parameters in the command line execution (see Input Components).

Chapter 12

Customizing MapForce

12 Customizing MapForce

This section provides information about working with Altova Global Resources, customizing the mapping output, generating and customizing mapping documentation, and working with catalog files.

12.1 Altova Global Resources

Global resources are a major enhancement in the interoperability between products of the Altova product line, and are currently available in the following Altova products: XMLSpy, MapForce, StyleVision and DatabaseSpy. Users of the Altova MissionKits have access to the same functionality in the respective products.

General uses:

- Workflows can be defined that use various Altova applications to process data.
- An application can invoke a target application, initiate data processing there, and route the data back to the originating application.
- Defining input and output data, file locations as well as databases, as global resources.
- Switching between global resources during runtime to switch between development or deployment resources.
- What-if scenarios for QA purposes.

The default location of the global resource definitions file, **GlobalResources.xml**, is c:\Documents and Settings\UserName\My Documents\Altova\. This is the default location for all Altova applications that can use global resources. Changes made to global resources are thus automatically available in all applications. The file name and location can be changed. Please see Global Resources - Properties for more information.

General mechanism:

- Global resources are **defined** in an application and automatically saved.
- Global resources are **assigned** to components whose data you intend to be variable.
- The global resource is invoked / **activated** in an application, allowing you to switch resources at runtime.

This section will describe how to define and use, global resources using existing mappings available in the ...\MapForceExamples\Tutorial\ folder.

To activate the Global Resources toolbar:
Select the menu option **Tools | Customize |** click the **Toolbar tab** and activate the Global Resources check box. This displays the global resources tool bar.

The combo box allows you to switch between the various resources, a "Default" entry is always available.
Clicking the Global Resources icon opens the Global Resources dialog box (alternatively Tools | Global Resources).

12.1.1 Global Resources - Files

Global Resources in MapForce:

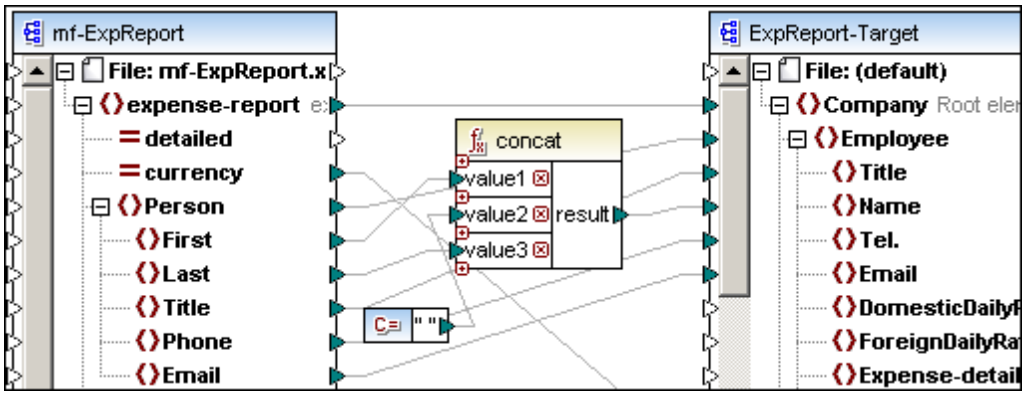
- Any input/output components **files** can be defined as global resources, e.g. XML, XML Schema, Text/CSV, database, EDI, and Excel 2007 and higher files.

Aim of this section:

- To make the source component input file, **mf-ExpReport**, a global resource.

- To **switch** between the two XML files that supply its **input data** at runtime, and check the resulting XML output in the Output preview tab.

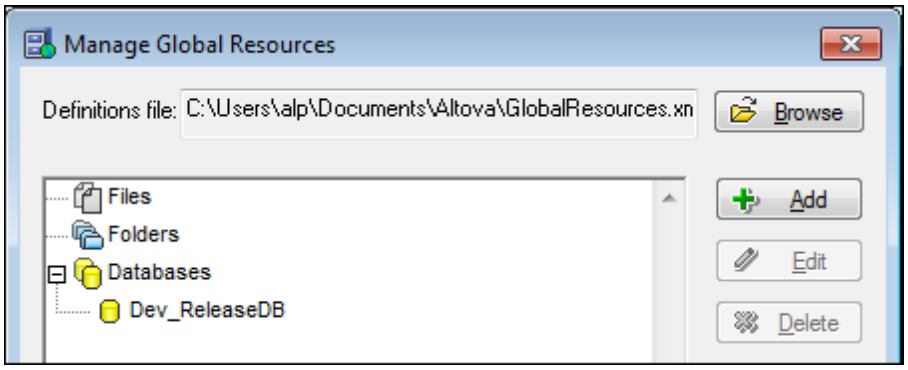
  This section uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples\Tutorial\ folder.



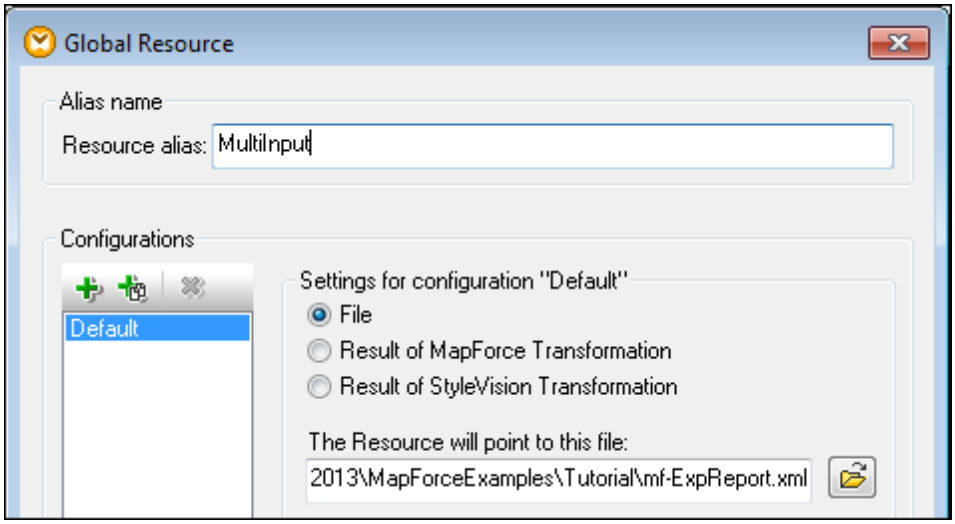
12.1.1.1 Defining / Adding global resources

Defining / Adding a global resource file:

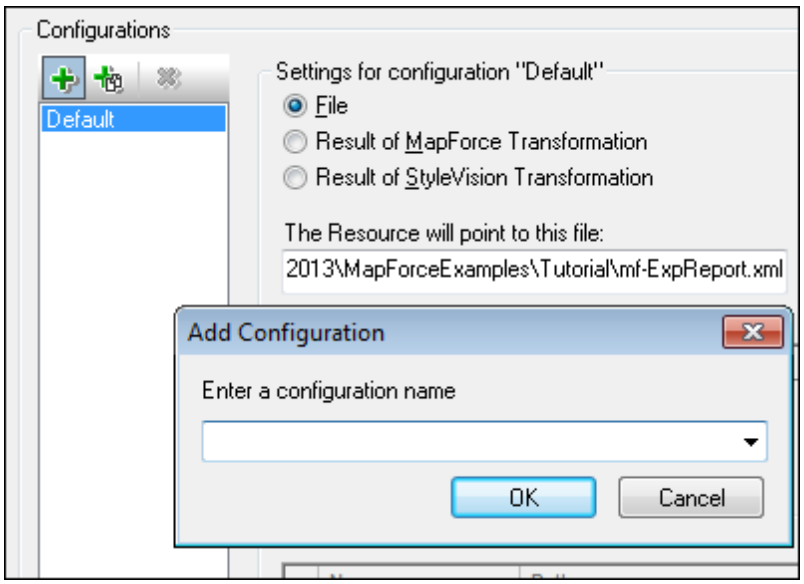
1. Click the Global Resource icon to open the dialog box.



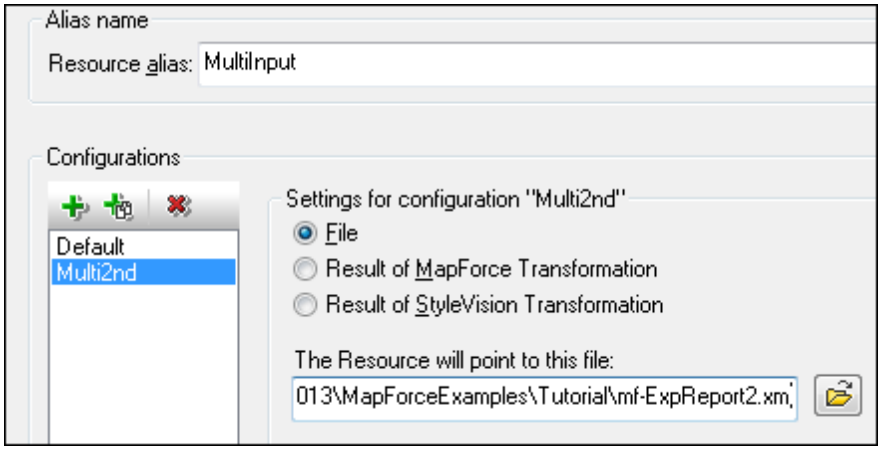
   This is the state of an empty global resources file.
2. Click the **Add** button and select **File** from the popup.
3. Enter the name of the Resource alias e.g. **MultiInput**.
4. Click the Open folder icon and select the XML file that is to act as the "Default" input file e.g. **mf-ExpReport.xml**.



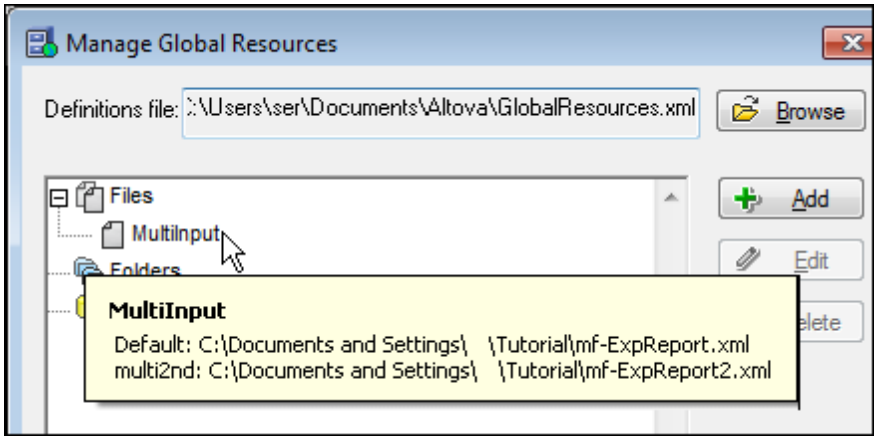
5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias. Note that the Copy configuration icon, allows you to copy a selected configuration and save it under a new name.



6. Enter a name for the configuration, **Multi2nd**, and click OK to confirm.
   Multi2nd has now been added to the Configurations list.
7. Click the Open folder icon again and select the XML file that is to act as the input file for the multi2nd configuration e.g. **mf-ExpReport2.xml**.



8. Click OK to complete the definition of the resource.
   The **MultiInput** alias has now been added to the Files section of the global resources. Placing the mouse cursor over an alias entry, opens a tooltip showing its defintion.



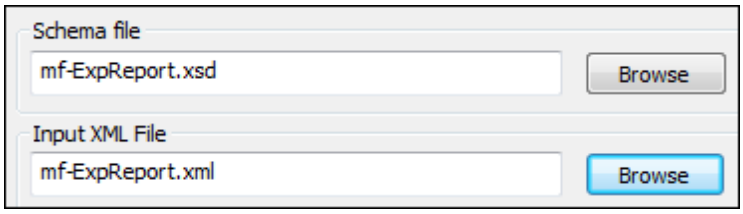
9. Click OK to confirm.
   This concludes the definition part of defining global resources. The next step is Assigning a global resource to a component.

12.1.1.2 Assigning a global resource

Assigning global resources to a component

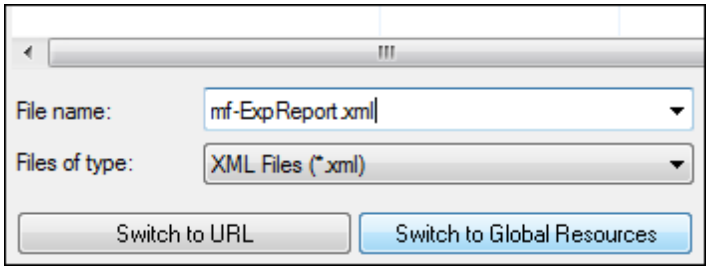
We now have to assign the global resource to the component that is to make use of it, i.e. the mf-ExpReport.xml file that is being used as a source file for the mapping.

1. Double click the **mf-ExpReport** component and click the Browse button next to the Input XML File field.

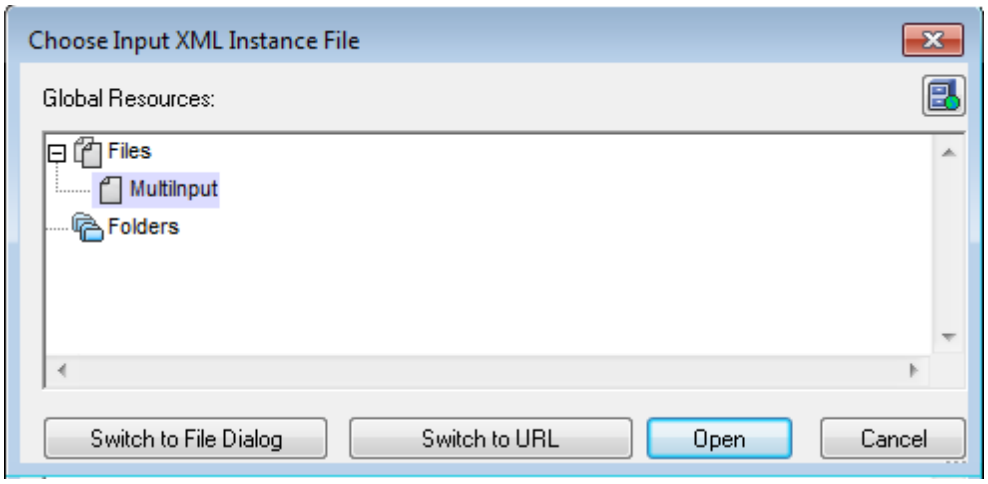


This opens the "Choose XML Instance file" dialog box.

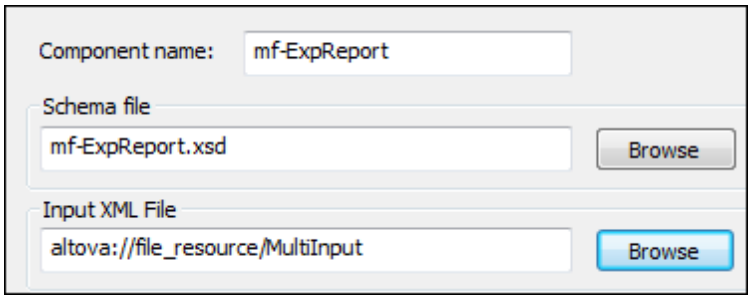
2. Click the **Switch to Global Resources** button at the base of the dialog box.



3. Click the resource you want to assign, **MultiInput** in this case, and click Open.



Note: the **Input XML File** field of the component, now contains a reference to a resource i.e. **altova://file_resource/MultiInput**.



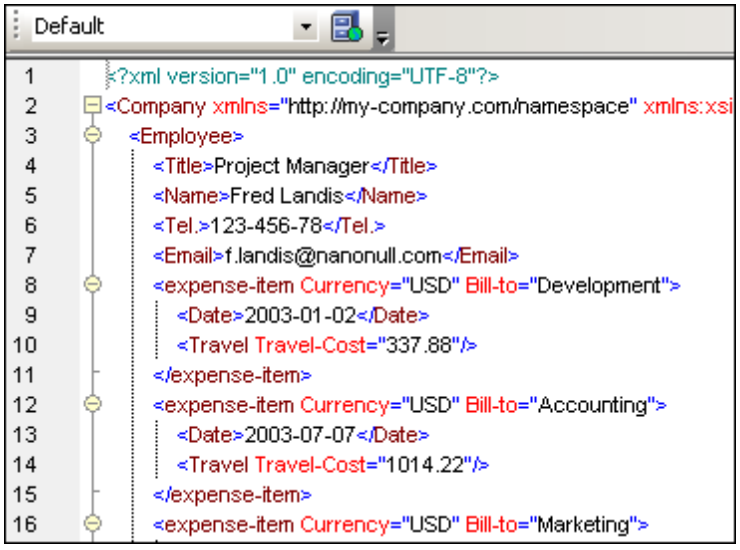
4. Click OK to complete the assignment of a resource to a component.
The next step is Using / activating a global resource.

12.1.1.3 Using / activating a global resource

Using / activating a global resource

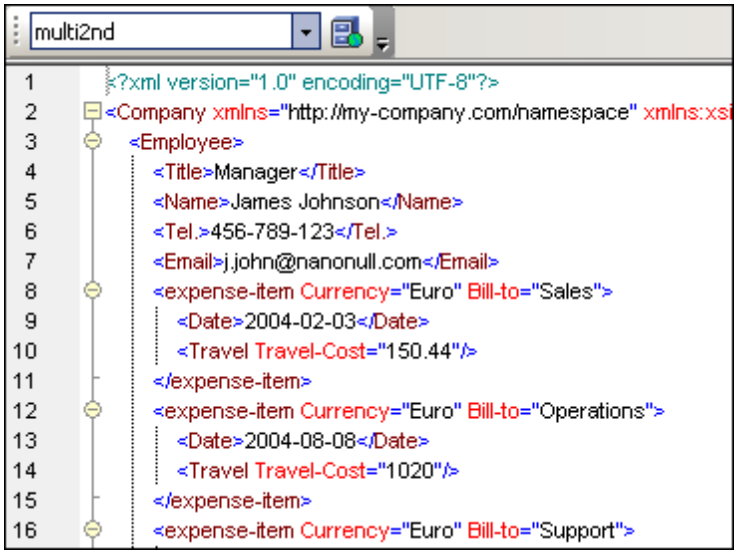
At this point the previously defined **Default** configuration for the **MultiInput** Alias is active. You can check this by noting that the entry in the Global Resources icon bar is "Default".

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



2. Click the **Mapping** tab to return to the mapping view.
3. Click the global resources combo box select **multi2nd** from the combo box.

4. Click the Output tab to see the new result.
   The mf-ExpReport2.xml file is now used as the source component for the mapping, and produces different output.



Note:

The **currently active** global resource (**multi2nd** in the global resources toolbar) determines the result of the mapping. This is also the case when you generate code.

12.1.2 Global Resources - Folders

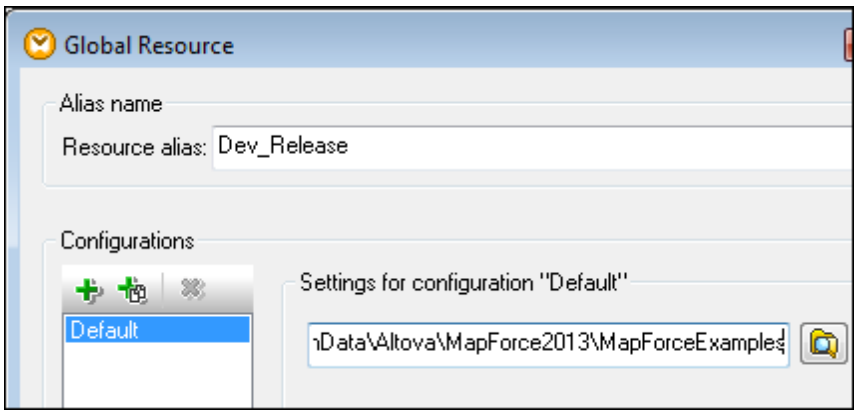
Folders can also be defined as a global resource, which means that input components can contain files that refer to different folders, for development and release cycles for example.

Defining folders for output components is not really useful in MapForce, as you are always prompted for the target folders when generating XSLT or code for other programming languages.

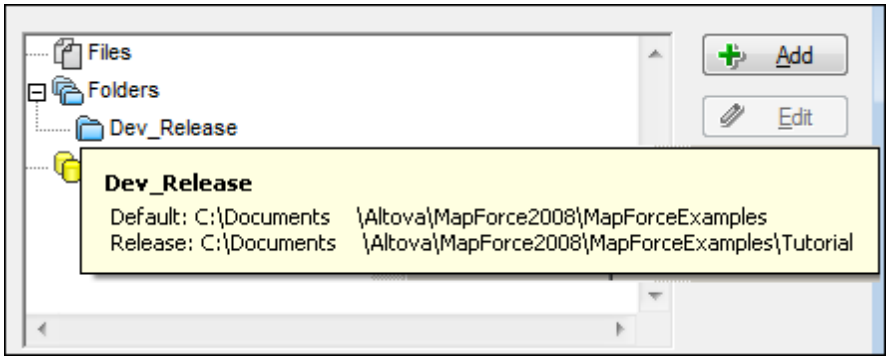
The mapping file used in this section is available as "**global-folder.mfd**" in the ...\MapForceExamples\Tutorial\ folder.

Defining / Adding global resource folders

1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **Folder** from the popup.
3. Enter the name of the Resource alias e.g. **Dev_Release**.
4. Click the Open folder icon and select the "Default" input folder, ...\MapForceExamples.



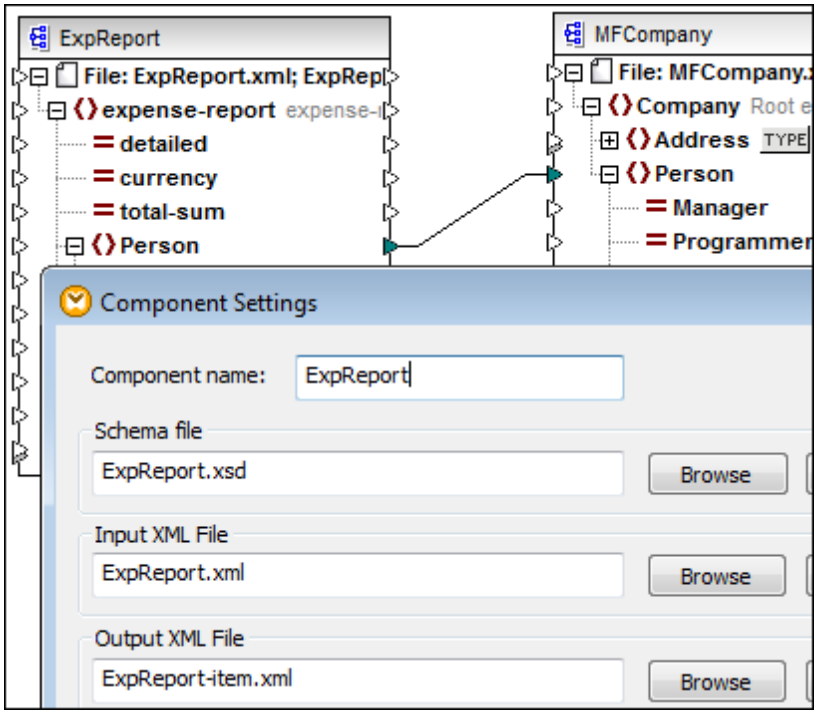
5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias, and enter a name for it e.g. Release. Note that the Copy configuration icon , allows you to copy a selected configuration and save it under a new name.
6. Click the Open folder icon and select the Release input folder, ...\MapForceExamples\Tutorial.



7. Click OK to finish the global folder definition.

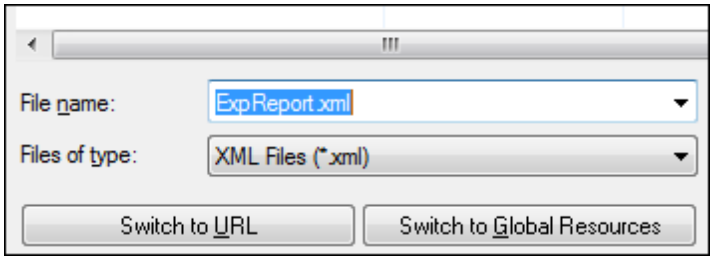
Assigning the global resource folders:

1. Double click the **ExpReport** component and click the **Browse** button next to the **Input XML File** field.

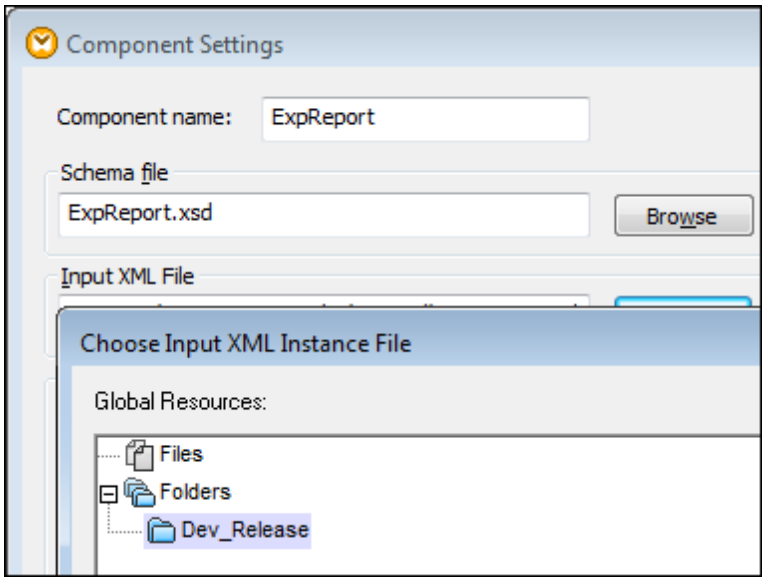


This opens the "Choose XML Instance file" dialog box.

2. Click the **Switch to Global Resources** button at the base of the dialog box.

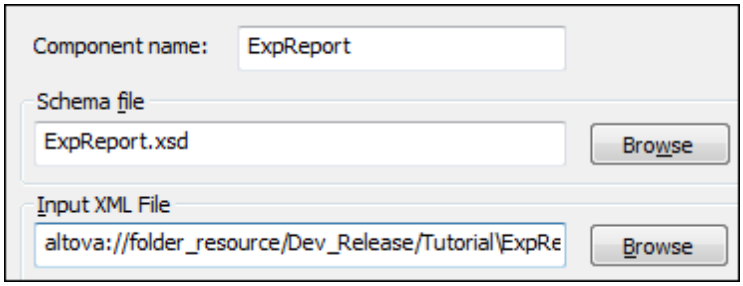


3. Click the resource you want to assign, **Dev_Release** in this case, and click OK.



The "Open..." dialog appears.

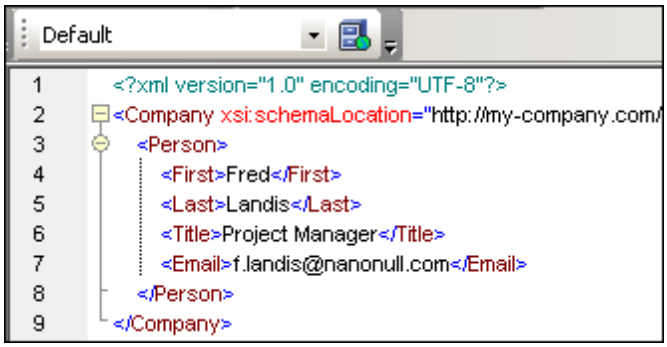
4. Select the **file** name that is to act as both the Development and Release **resource** file in each of the folders, e.g. **ExpReport.xml** and click OK to finish assigning the resource folder.



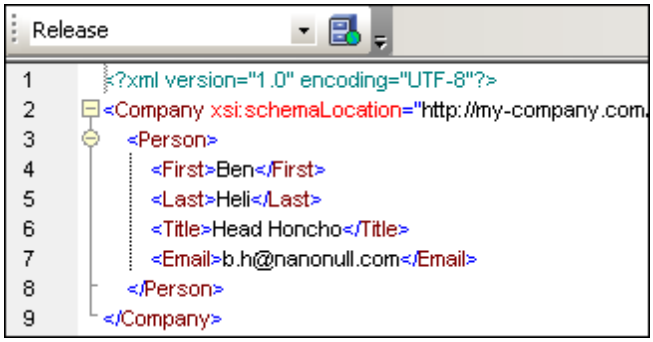
Note that this file is available in both folders but has different content.

Changing the resource folder at runtime:

1. Click the Output tab to see the result of the transformation.
   Note that this is the **Default** configuration/folder .../MapforceExamples.



2. Click the Mapping tab to return to the mapping window.
3. Click the Global Resource combo box and select the "**Release**" entry.
3. Click the Output button to see the result using the Release global resource.

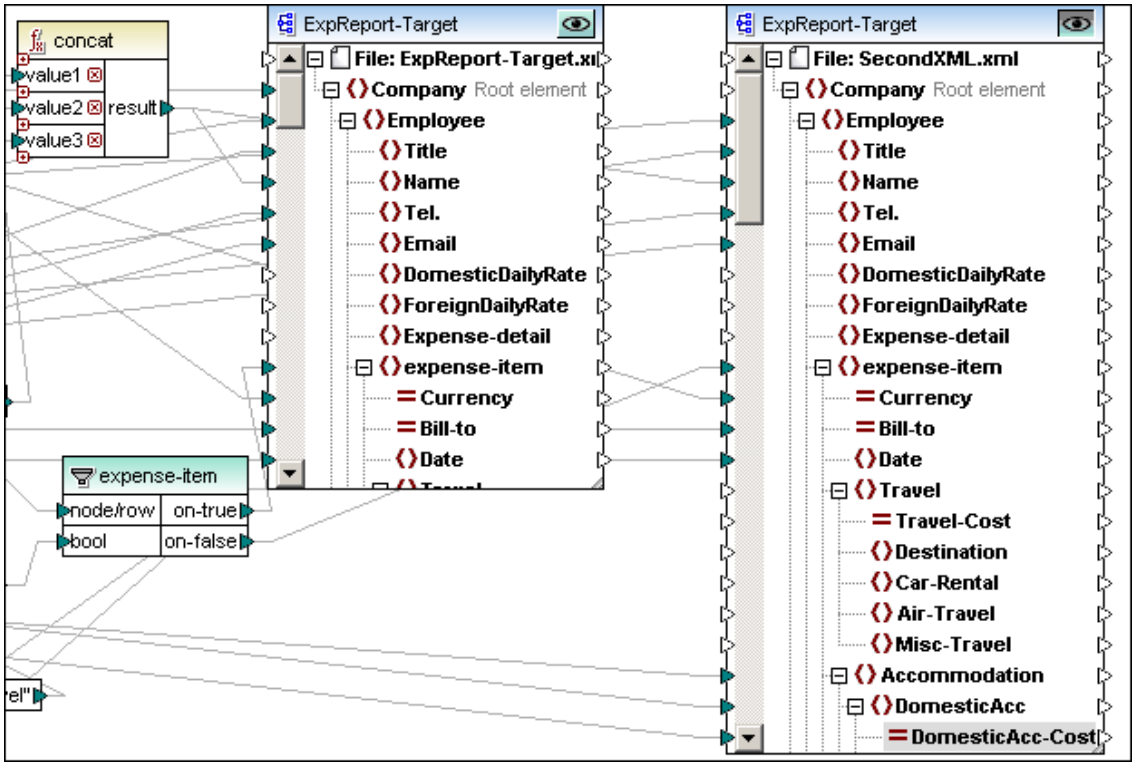


The output from the "Release" folder .../MapforceExamples/Tutorial is now displayed.

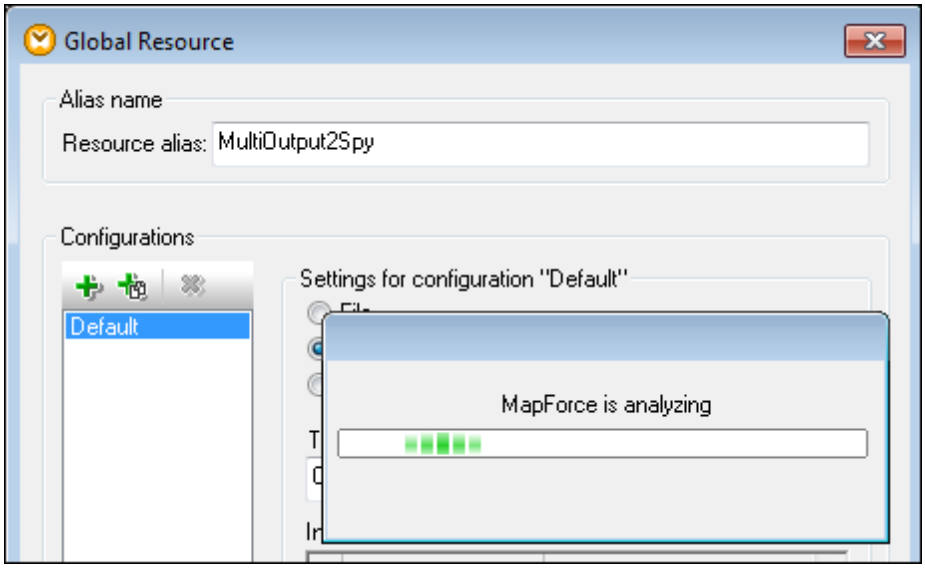
12.1.3 Global Resources - Application workflow

The aim of this section is to create a workflow situation between two Altova applications. Workflow is initiated in XMLSpy which starts MapForce and passes the generated XML file output back to XMLSpy for further processing.

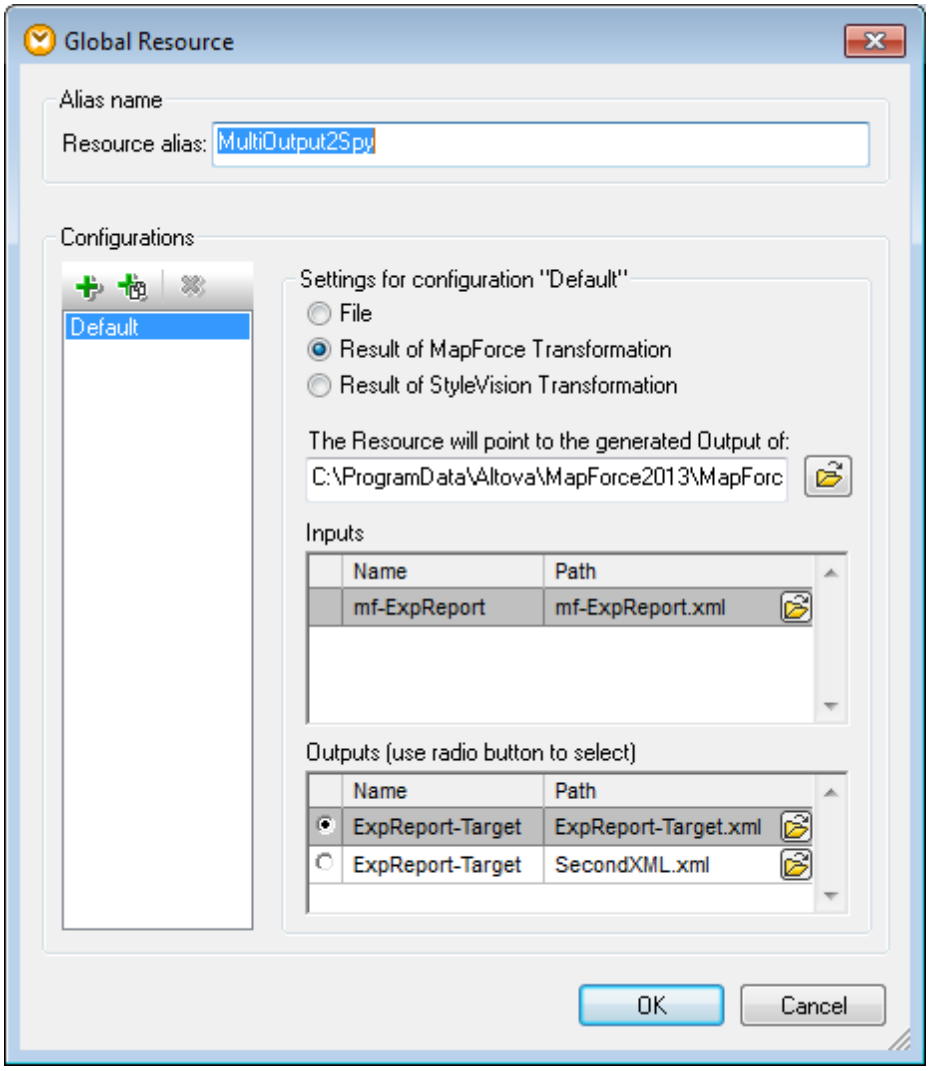
This mapping uses two output components to produce two types of filtered output; Travel and Non-travel expenses of the expense report input file. This section uses the **Tut-ExpReport-multi.mfd** mapping file available in the ...\MapForceExamples\Tutorial\ folder.



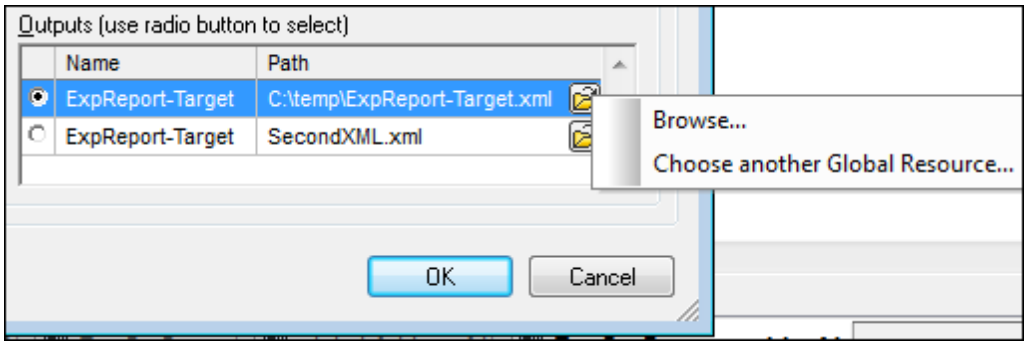
1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **File** from the popup.
3. Enter the name of the Resource alias e.g. **MultiOutput2Spy**
4. Click the "**Result of MapForce Transformation**" radio button, then click the Open file icon.
5. Select the **Tut-ExpReport-multi.mfd** mapping.



MapForce analyzes the mapping and displays the input and output files in separate list boxes.

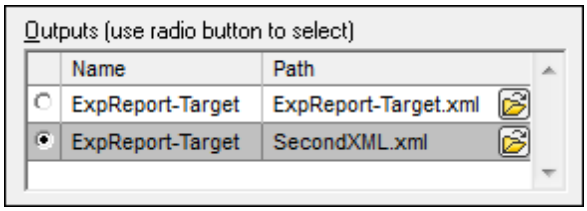


6. Click the top radio button entry in the **Outputs** section, if not already selected. Note that the output file name is **ExpReport-Target.xml** and that we are currently defining the **Default** configuration.
7. Click the icon and select Browse from the popup menu, to define the new location of the output file e.g. C:\Temp.



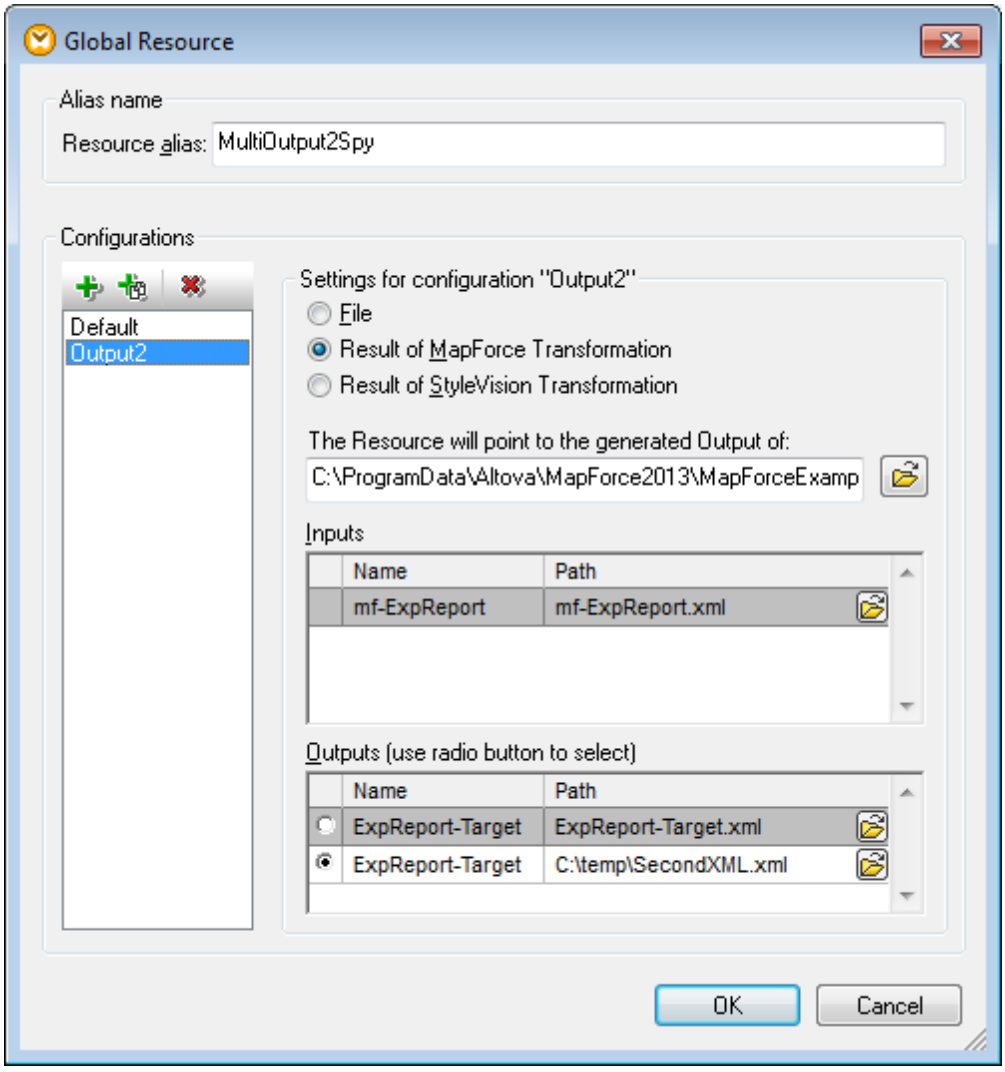
8. Enter the new output location e.g. C:\Temp and click the Save button. This location can differ from the location defined in the component settings.

9. Click the Add button of the Configurations group (of this dialog box), to add a new configuration to the resource alias.
10. Enter the name of the configuration, e.g. **Output2**, click the Open file icon, select the Tut-ExpReport-multi.mfd.
11. Click the lower radio button of the Outputs listbox. Note that the output file name is **SecondXML.xml**.

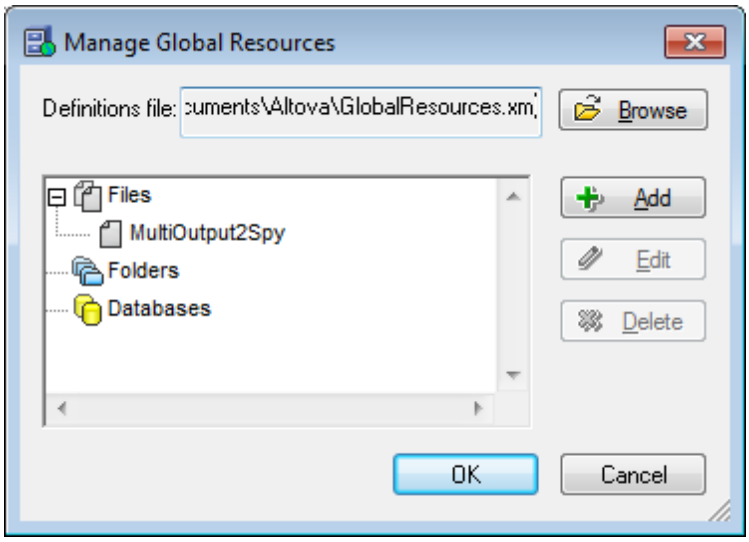


12. Click the icon and select Browse from the popup menu, to define the new location of the output file e.g. C:\Temp.

    Note: clicking the "**Choose another Global Resource...** in the popup, allows you to save the MapForce output as a global resource. I.e. the output is stored to a file that the global resource physically points to/references.



13. Click OK to save the new global resources.
    The new resource alias **MultiOutput2Spy** has been added to the Global Resources definition file.

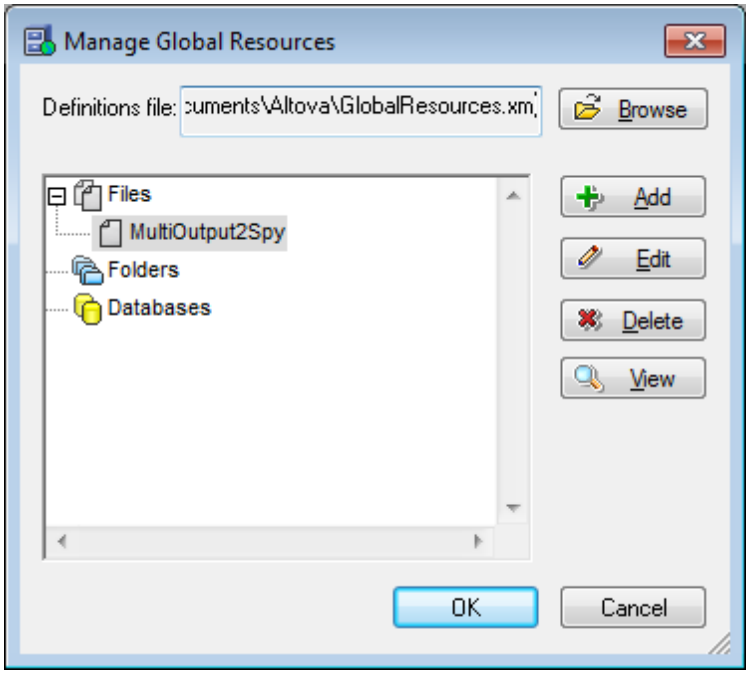


14. Click OK to complete the definition phase.

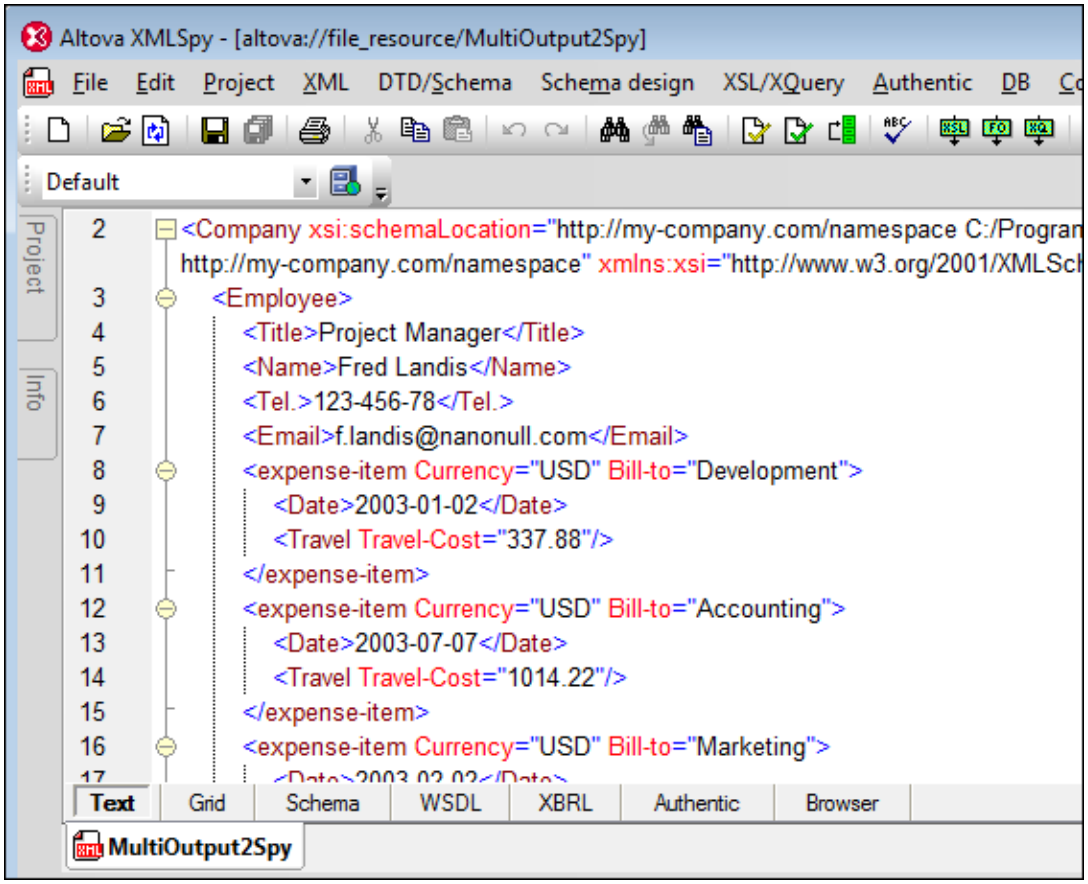
12.1.3.1 Start application workflow

This section shows how the Global Resource is activated in XMLSpy and how the resulting MapForce transformation is routed back to it.

1. Start XMLSpy and shut down MapForce, if open, to get a better view of how the two applications interact.
2. Select the menu option **Tools | Global Resources** in XMLSpy.
3. Select the **MultiOutput2Spy** entry, and click the **View** button.



A message box stating that MapForce is transforming appears, with the result of the transformation appearing in the Text view window.

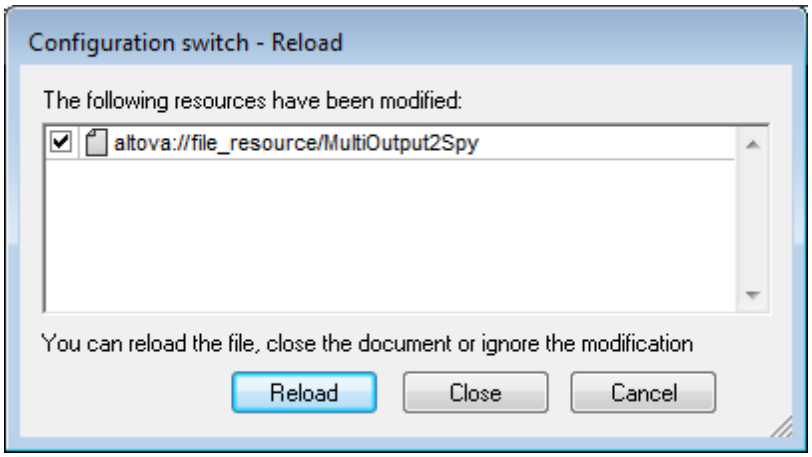


Note:

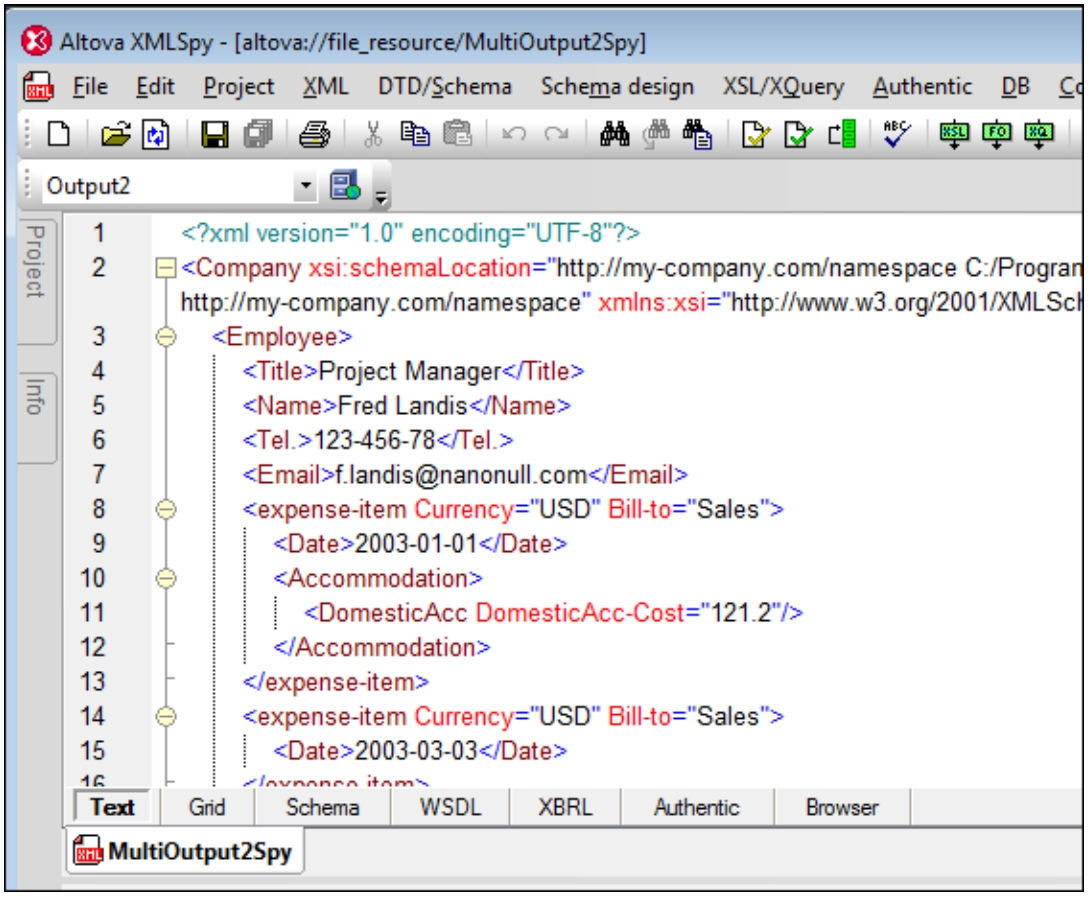
- The currently selected configuration is "**Default**".
- The name of the resource alias is in the application title bar **altova://file_resource/MultiOutput2Spy**.
- The output file has been opened as "MultiOutput2Spy.xml" for further processing.
- The **ExpReport-Target.xml** file has been copied to the C:\Temp folder.

To retrieve the non-travel expenses output:

1. Click the Global Resources combo box and select "**Output2**".
   A notification message box opens.



2. Click **Reload** to retrieve the second output file defined by the resource.



The result of the transformation appears in the Text view window and overwrites the previous MultiOutput2Spy.xml file.

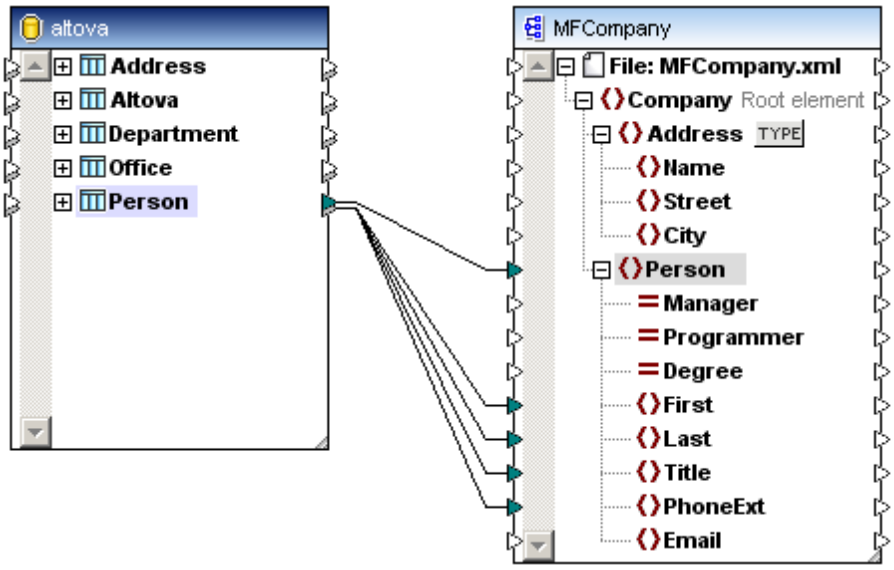
Note:

- The currently selected configuration is "**Output2**"
- The output file has been opened as "Untitled1.xml" for further processing.
- The **SecondXML.xml** file has been copied to the C:\Temp folder.

12.1.4 Global Resources - Databases

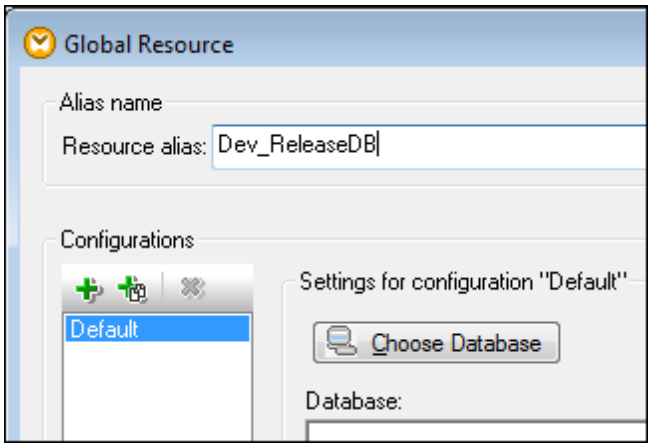
Databases components can also be defined as a global resource allowing you to refer to different databases, for development or release cycles for example. Database resources can be both source and target components.

The mapping file used in this section is available as "**PersonDB.mfd**" in the ...\MapForceExamples\Tutorial\ folder.

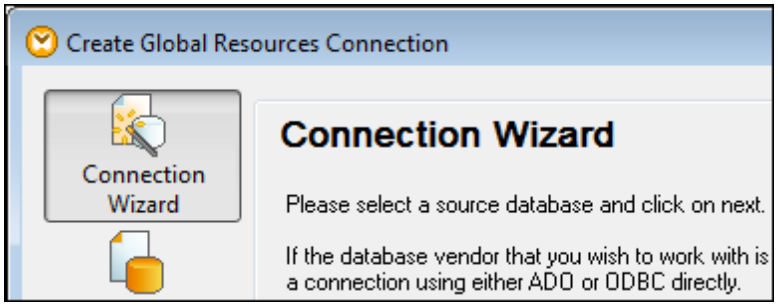


Defining / Adding a database global resource

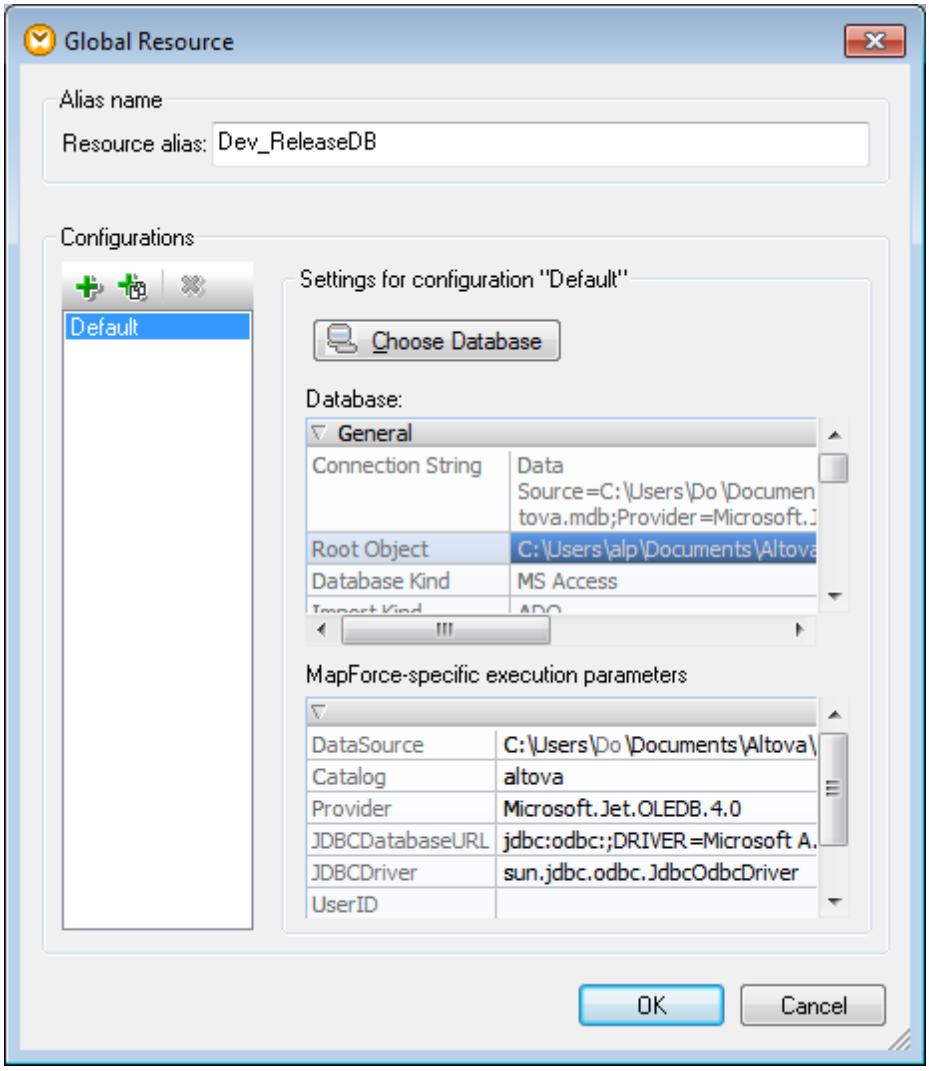
1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **Database** from the popup.
3. Enter the name of the Resource alias e.g. **Dev_ReleaseDB**.



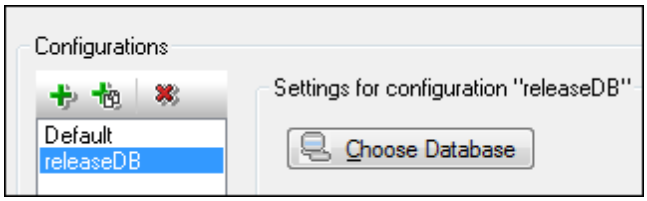
4. Click the Choose Database button, then the "Connection Wizard" button (in the Connection Wizard dialog box) and select **Altova.mdb** in the **...\MapForceExamples** folder.



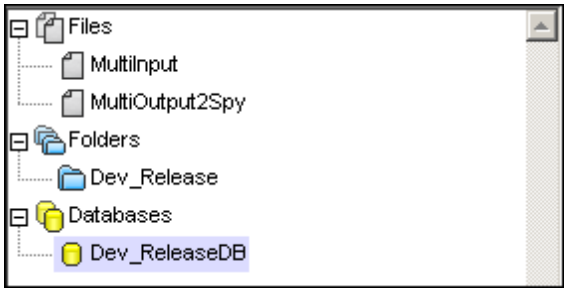
This is the **development** database.



5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias, and enter a name for it e.g. **releaseDB**.



6. Click the Choose Database button again, and select the **release** database, e.g. altova.mdb in the ...\MapForceExamples**\Tutorial** folder. Click OK to finish the database resource definition.



Please note:

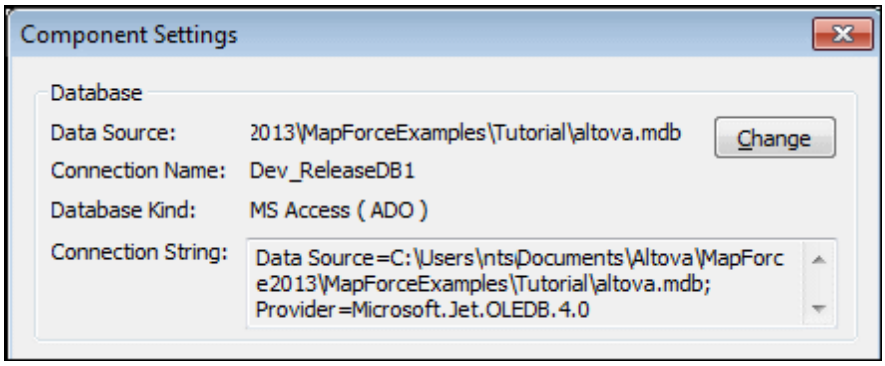
Any of the database settings in the "**Database**" or "**MapForce execution parameters**" list boxes can be edited/changed once you have selected a database. E.g. double clicking in the Password field allows you to update the current database password.

The MapForce execution parameters list box is filled with default values from the database as soon as the database has been selected. These parameters are used when generating **code** and generating the **output preview** in the Built-in execution engine.

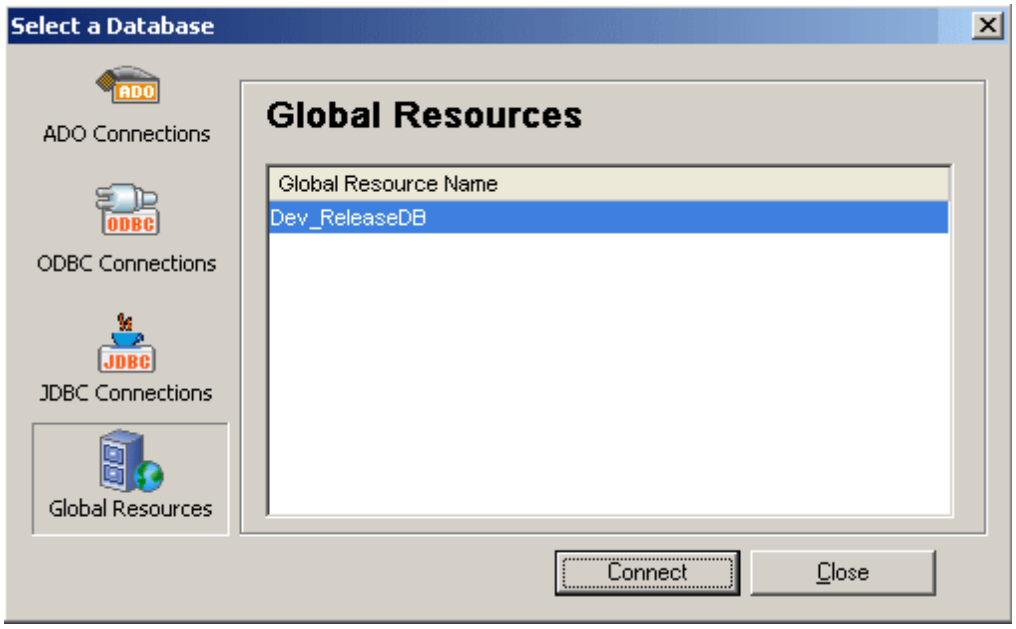
Depending on the database you are connecting to e.g. IBM DB2, a "Choose Root Object" dialog box may appear. This allows you to select the root object immediately through the Set Root Object button, or defer the selection until later when clicking the Skip button.

Assigning the database resource:

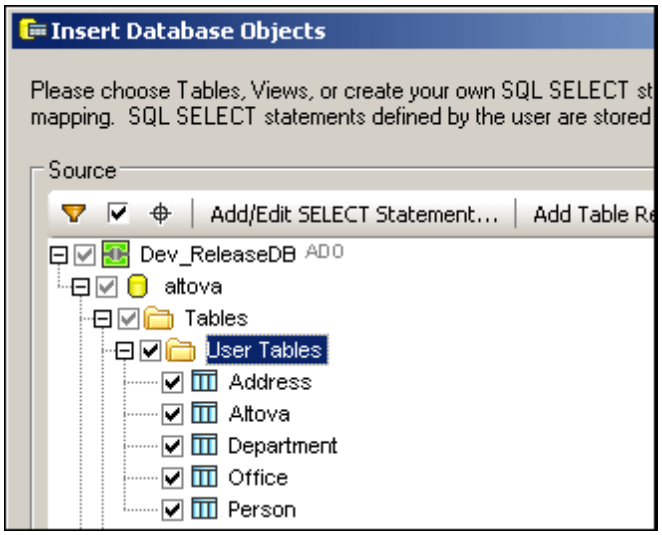
1. Double click the **Altova** database component, click the "Change" button (and then the Global Resources button if necessary).



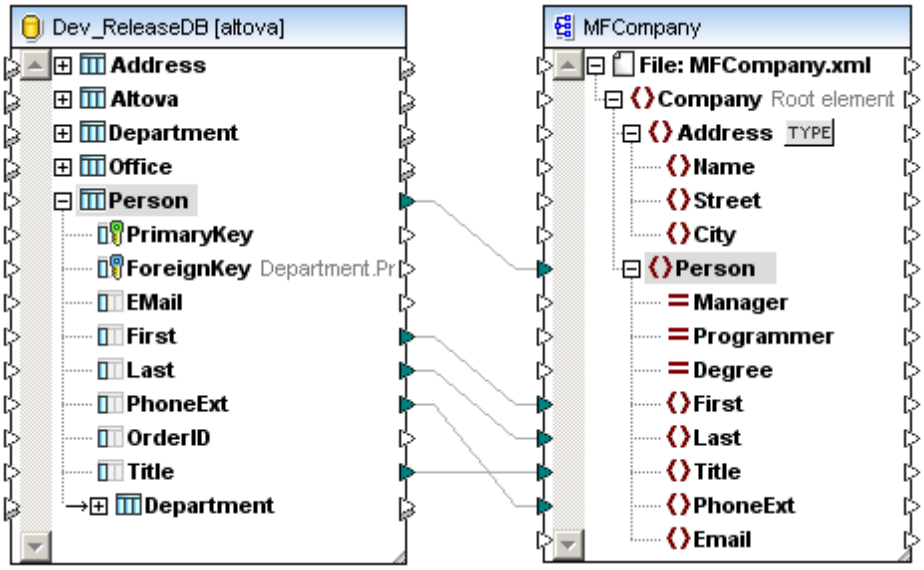
This opens the Select a Database dialog box.



2. Select the **Dev_ReleaseDB** entry and click Connect.

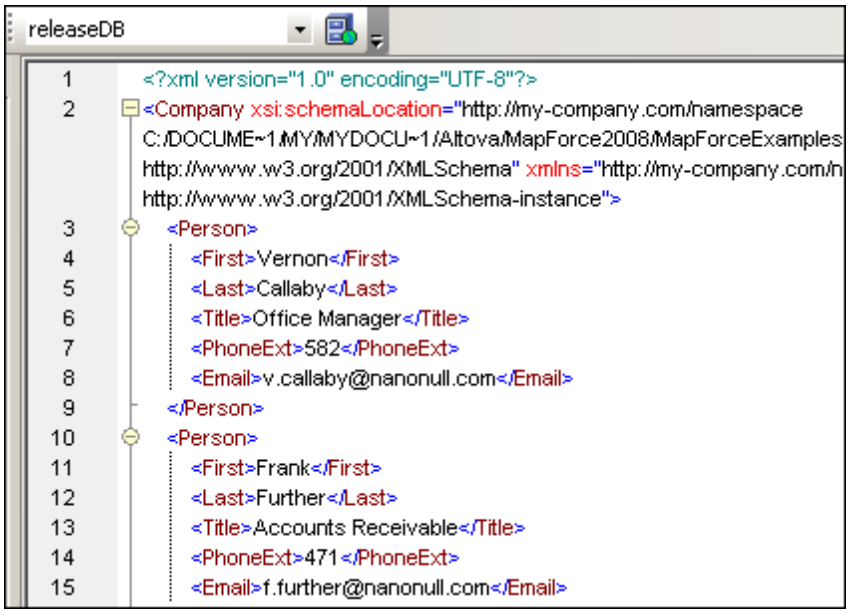


3. Select the tables that you want to be available in the component, then click OK.
The resource alias is now visible in the component header **Dev_ReleaseDB [Altova]**.



Changing the database resource at runtime:

1. Click the Output tab to see the result of the mapping.
   Note that this is the **Default** default database in the .../MapforceExamples folder.
2. Click the Global Resource combo box and select the "**releaseDB**" entry.
3. Click **Reload** when the message box appears.
4. Click the Output button to see the result of the mapping.



The data from the **release** database is now displayed. As both databases are identical, there is no visible difference in this simple example.

12.1.5 Global Resources - Properties

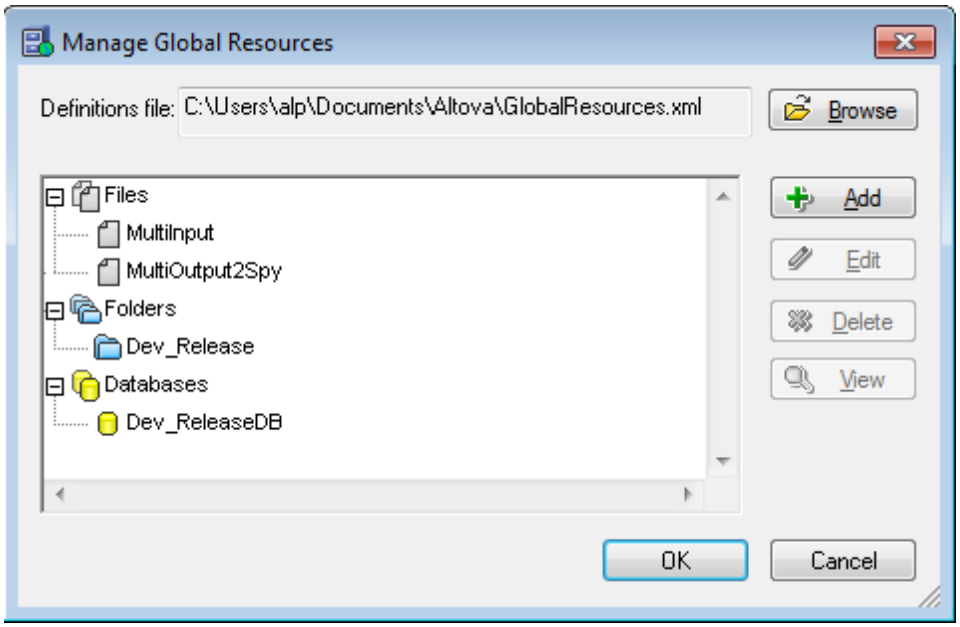
The Global Resources XML File

Global resources definitions are stored in an XML file. By default, this XML file is called

GlobalResources.xml, and it is stored in the folder C:\Documents and Settings\<username>\My Documents\Altova\. This file is set as the default Global Resources XML File for all Altova applications. As a result, a global resource defined in any application will be available to all Altova applications—assuming that all applications use this file.

You can also re-name the file and save it to any location. You can therefore have multiple Global Resources XML files. However, only one of these Global Resources XML File can be active, per application, at one time, and only the definitions contained in this file will be available to the application.

To make the Global Resources XML file active, click the **Browse** button of the "Definitions file" field and select the one you want to use from the "Open..." dialog box.



Managing global resources: adding, editing, deleting

In the Global Resources dialog, you can add a global resource to the selected Global Resources XML File, or edit or delete a selected global resource. The Global Resources XML File organizes the aliases you add into the following sections: files, folders, and databases.

To add a global resource:

Click the **Add** button and define the global resource in the Global Resource dialog that pops up. After you define a global resource and save it, the global resource (or alias) is added to the list of global definitions in the selected Global Resources XML File.

To edit a global resource:

Select it and click **Edit**. This pops up the Global Resource dialog, in which you can make the necessary changes.

To delete a global resource:

Select it and click **Delete**.

To view the result of an application workflow:

If the calling application e.g. XMLSpy, calls another application e.g. MapForce, then a View button is available in the Manage Global Resources dialog box.

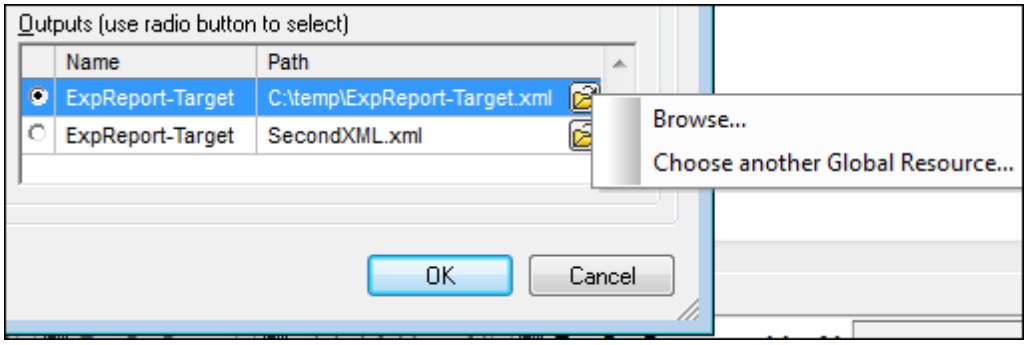
Clicking the View button shows the affect of the currently selected global resource in the callling application. Please see Global Resources - Application workflow for an example.

To save modifications made in the Managing Global Resources dialog box:
Having finished adding, editing, or deleting, make sure to click **OK** in the Global Resources dialog to save your modifications to the Global Resources XML File.

Note: Alias resource names must be unique **within** each of the Files, Folders or Databases sections. You can however define an identical alias name in two different sections, e.g. a multiInput alias can exist in the Files section as well as in the Folders section.

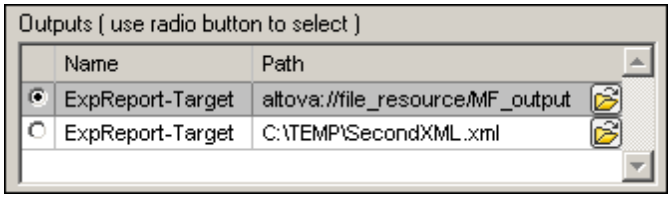
Selecting Results of MapForce transformations as a global resource

In a MapForce transformation that has multiple outputs, you can select which one of the output files should be used for the global resource by clicking its radio button.



The **output** file that is generated by the mapping can be saved as:

- a global resource via the **Choose another Global Resource** entry in the popup, visible as **altova://file_resource/MF_output**. The output is stored to a file that the global resource physically points to/references.



- a file via the icon, shown as C:\TEMP\Second.xml.

  If neither of these options is selected, a **temporary** XML file is created when the global resource is used.

Determining which resource is used at runtime

There are two application-wide selections that determine what global resources can be used and which global resources are actually used at any given time:

- *The active Global Resources XML File* is selected in the Global Resource dialog. The active Global Resources XML File can be changed at any time, and the global-resource definitions in the new active file will immediately replace those of the previously active file.

  The active Global Resources XML File therefore determines: (i) what global resources can be assigned, and (ii) what global resources are available for look-up (for example, if a global resource in one Global Resource XML File is assigned but there is no global

resource of that name in the currently active Global Resources XML File, then the assigned global resource (alias) cannot be looked up).

- *The active configuration* is selected via the menu item **Tools | Active Configuration** or via the Global Resources toolbar. Clicking this command (or dropdown list in the toolbar) pops up a list of configurations across all aliases.

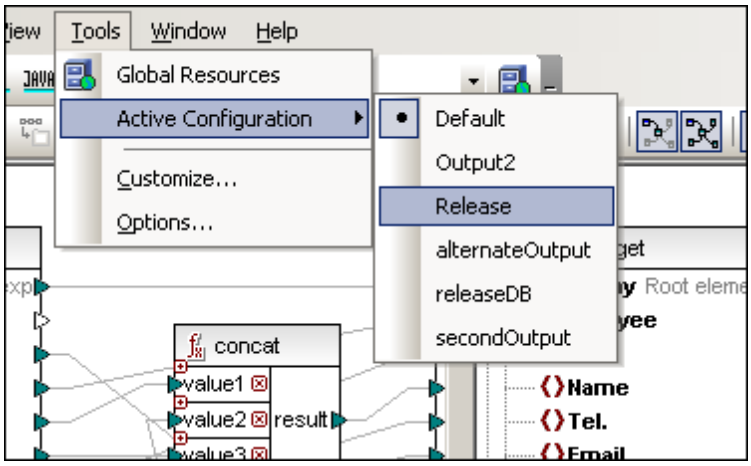
  Selecting a configuration makes that configuration active application-wide. This means that wherever a global resource (or alias) is used, the resource corresponding to the active configuration of each used alias will be loaded.

  The active configuration is applied to all used aliases. If an alias does not have a configuration with the name of the active configuration, then the **default configuration** of that alias will be used. The active configuration is not relevant when assigning resources; it is significant only when the resources are actually used.

Changing resources / configurations

Resources can be switched by selecting a different configuration name. This can be done in two ways:

- When you hover over the menu command **Tools | Active Configuration**, a submenu with a list of all configurations in the Global Resources XML File appears. Select the required configuration from the submenu.



- In the combo box of the Global Resources toolbar, select the required configuration. The Global Resources toolbar can be toggled on and off with the menu command **Tools | Customize,** then click the **Toolbar tab** and enable/disable the Global resources check box.

12.2 Styling Mapping Output with StyleVision

In MapForce, you can preview/render XML Schema and XBRL components using an associated StyleVision Power Stylesheet (SPS). This means the target component data are previewed as HTML, RTF, PDF, or Word 2007+ documents. If you are using the Enterprise edition of StyleVision then charts will also be rendered in these previews.

To be able to preview components in this way, Altova StyleVision must be installed on your computer; either as a standalone installation, or as part of Altova MissionKit.

StyleVision Power Stylesheets are created in StyleVision and are assigned to a component in MapForce. You cannot edit or change the stylesheet in MapForce directly, but you can open them via MapForce in StyleVision. In the 64-bit edition of MapForce, the Word 2007+ and RTF previews are opened as a non-embedded application.

Additional tabs are added to the MapForce tab bar if StyleVision has been installed, and an SPS file has been assigned to the target component.

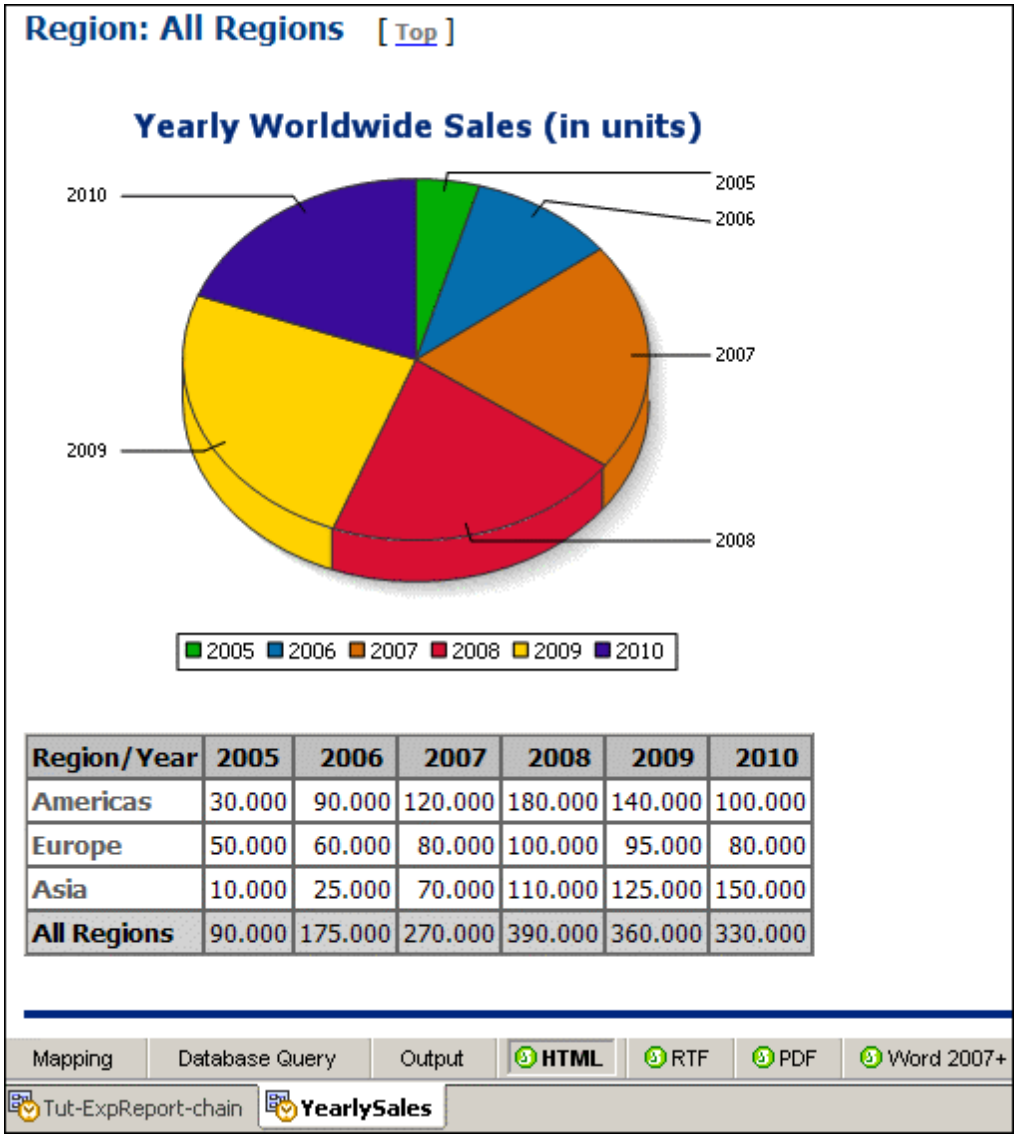
- HTML
- RTF - previews RTF
- PDF - requires Java, Acrobat Reader, and FOP (Formatting Objects Processor) version 0.93 or 1.0. FOP is installed together with StyleVision, unless you opted not to install it when installing StyleVision.
- Word 2007+ - previews MS Word documents from version 2007 and later

Please note:

The tabs available in MapForce depend on the installed version of StyleVision. All tabs mentioned above are available in the Enterprise editions. Whenever you see these extra tabs you know that an SPS file has been assigned to the component. The ...\MapForceExamples folder contains many examples where this is the case.

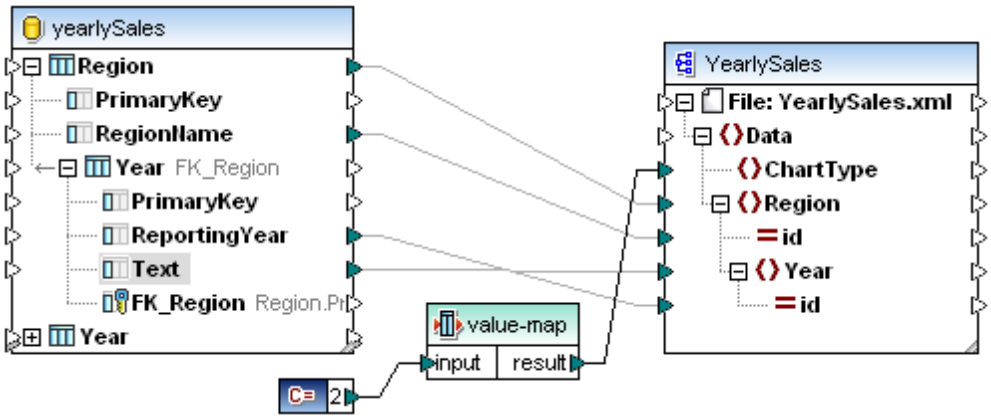
The Professional edition of StyleVision restricts the tabs to HTML and RTF in MapForce.

Note: if your mapping contains multiple linked components, i.e. a chained mapping, the SPS preview will only be visible for those components containing an SPS entry, and where the Preview button  of the component has been set as active.



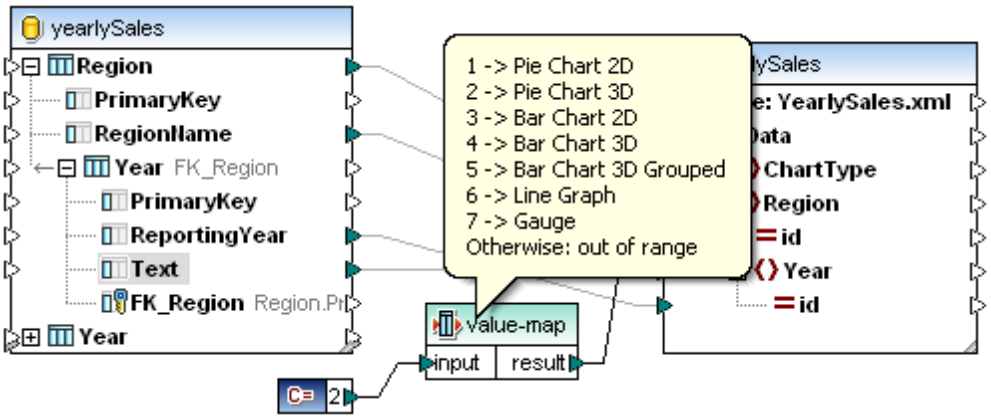
12.2.1 Styling Components with StyleVision

The **YearlySales.mfd** example shown below, is available in the ...\MapForceExamples\Tutorial folder.



Double clicking the constant component and entering a number, defines the type of chart that will appear when you click the respective StyleVision tab.

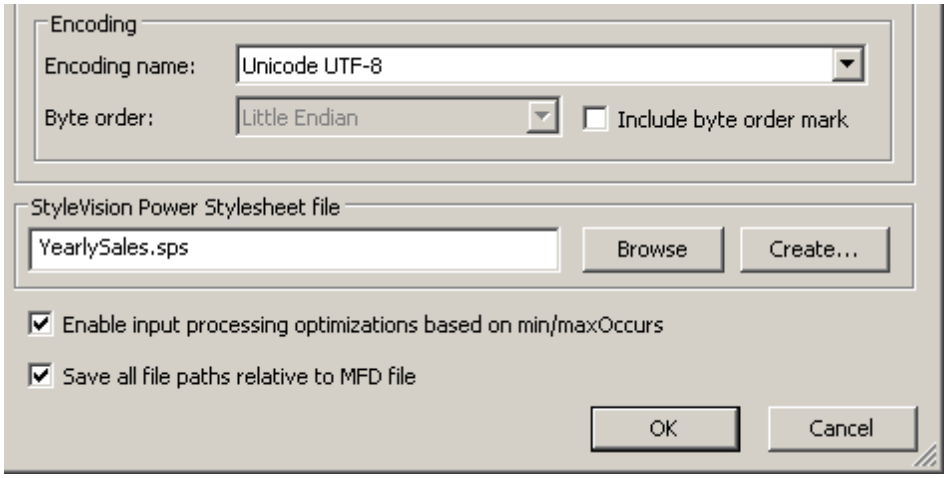
Placing the mouse cursor over the value-map function, shows the various possibilities in the popup, Pie Chart 3D in this case.



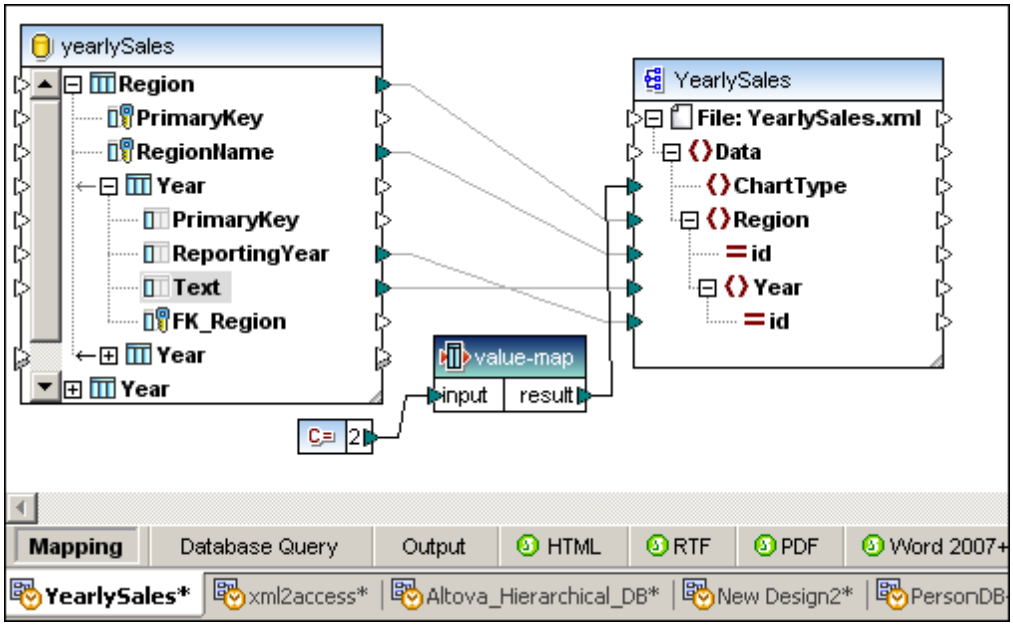
To assign an SPS template to a MapForce component:

Having designed the SPS template in StyleVision, or obtained the template from a third party,

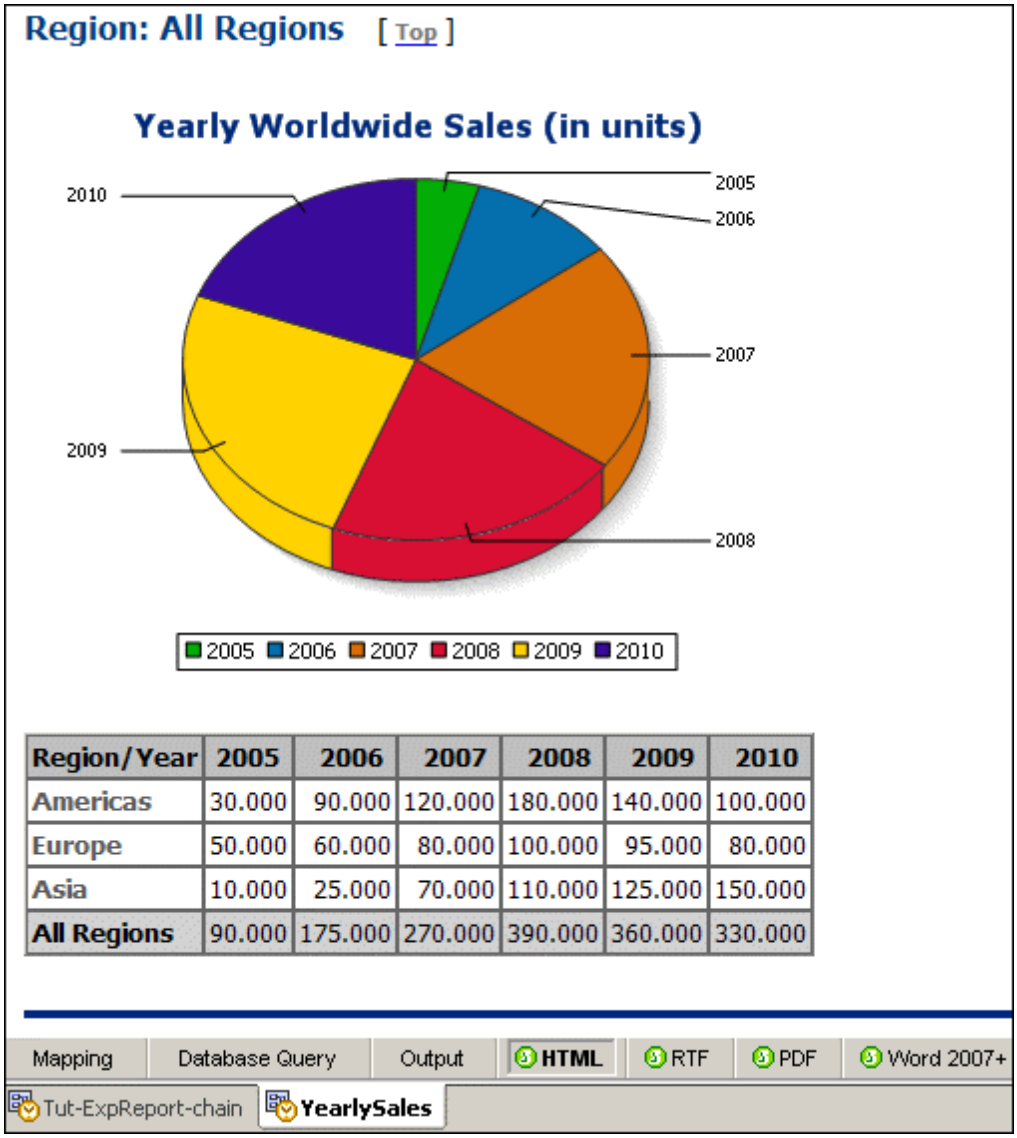
1. Double click the component title bar of the component you want to assign the SPS file to, e.g. YearlySales.



2. Click the Browse button to select the StyleVision SPS file, then click OK.



The HTML, RTF, PDF, and Word 2007+ tabs appear to the right of the Output tab. Clicking a tab renders the component data in the specific format.



Please note:

Clicking the Create... button in the Component Settings dialog box, prompts for an SPS file name and opens StyleVision allowing you to create a new SPS file. Double clicking the same component once an SPS file has been assigned, shows the Create button as Edit.

12.3 Generating and Customizing Mapping Documentation

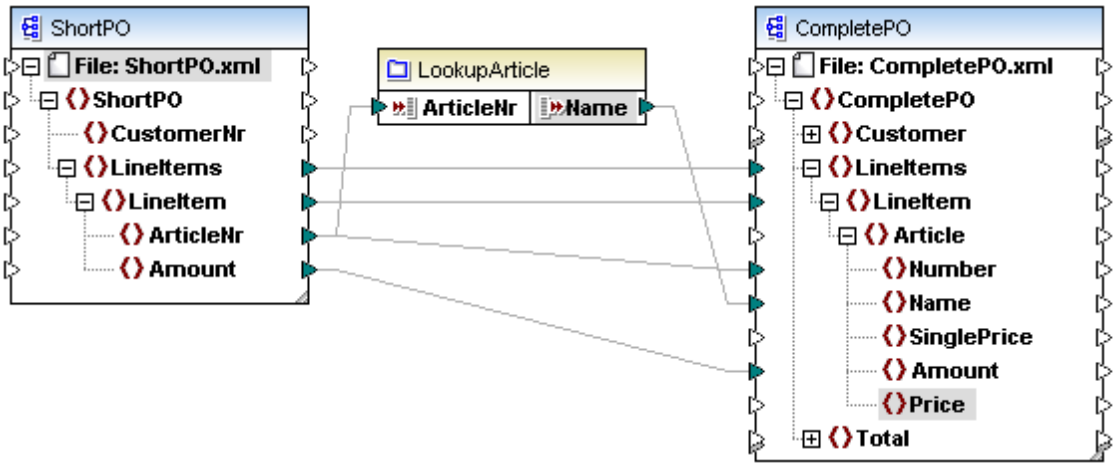
The **Generate Documentation** command generates detailed documentation about your mapping in HTML, MS Word, RTF or PDF. The documentation generated by this command can be freely altered and used; permission from Altova to do so is not required.

Documentation is generated for components you select in the Generate Documentation dialog box. You can either use the fixed design, or use a StyleVision Power Stylesheet (SPS) for the design. Using a StyleVision SPS enables you to customize the design of the generated documentation (see User-Defined Design).

Note: To use an SPS to generate mapping documentation, you must have StyleVision installed on your machine. Related elements are typically hyperlinked in the onscreen output, enabling you to navigate from component to component.

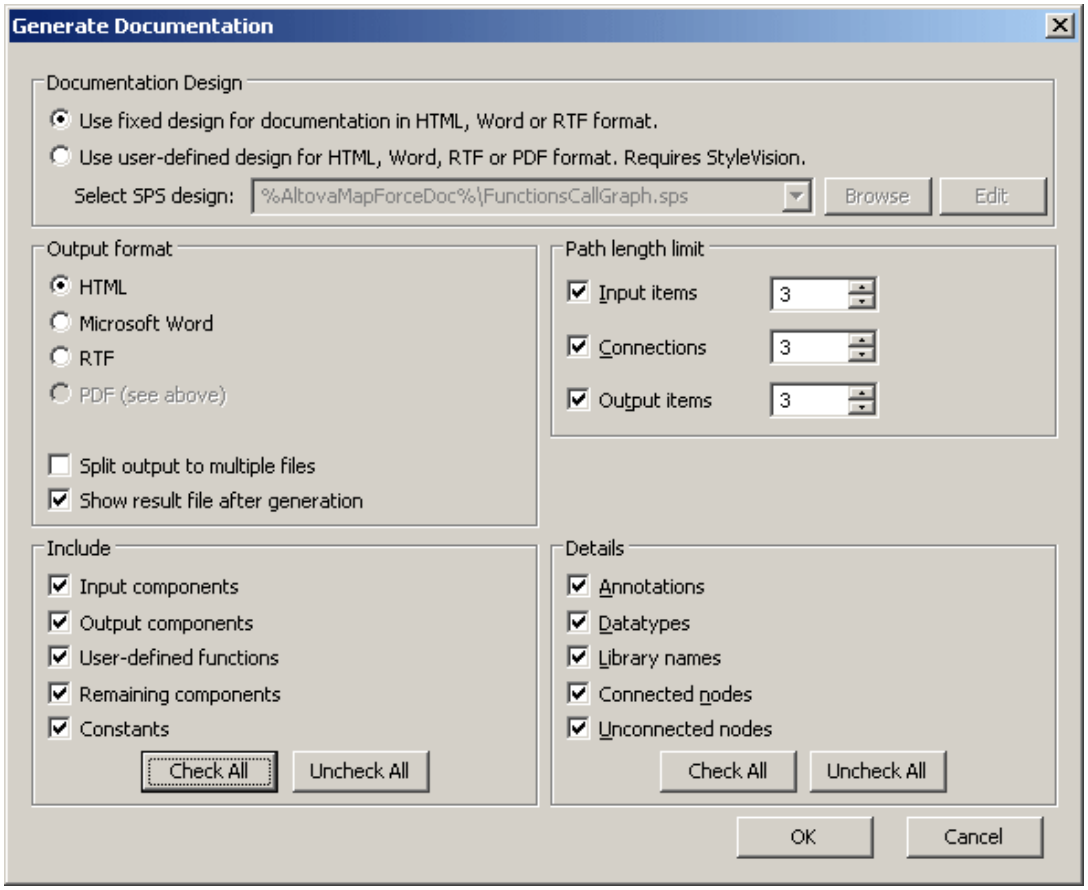
To generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

The screenshot below shows a portion of the **Lookup-standard.mfd** file available in the **...\MapForceExamples** folder.



Having opened a mapping file e.g. Lookup-standard.mfd:

- Select the menu option **File | Generate Documentation**.
  This opens the "Generate documentation" dialog box. The screenshot below shows the default dialog box settings.



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 **...\My Documents\Altova\MapForce2016\Documentation\MapForce** folder.
- Click **Browse** to browse for a predefined SPS file.
- Click **Edit** to launch StyleVision and open the selected SPS in a StyleVision window.

The following predefined SPS stylesheets are available in the ...MapForce2016\Documentation\MapForce folder:

- **FunctionCallGraph.sps** - shows the call graph of the main mapping and any user-defined functions.
- **FunctionsUsedBy.sps** - shows which functions are used directly, or indirectly, in the mapping.
- **ImpactAnalysis.sps** - lists every source and target node, and the route taken via various functions, to the target node.
- **OverallDocumentation.sps** - shows all nodes, connections, functions, and target nodes. The output using this option outputs the maximum detail and is identical to the built-in "fixed design..." output.

Output Format

- The output format is specified here: either HTML, Microsoft Word, RTF, or PDF.

  Microsoft Word documents are created with the **.doc** file extension when generated using a fixed design, and with a **.docx** file extension when generated using a StyleVision SPS.

  The PDF output format is only available if you use a StyleVision SPS to generate the documentation.

- Select "**Split output to multiple files**" if you would like separate input, output, constant components, user-defined functions from the Library component documentation. In fixed designs, links between multiple documents are created automatically.

- The "**Show Result File...**" option is enabled for all output options. When checked, the result files are displayed in default browser (HTML output), MS Word (MS Word output), and the default application for .rtf files (RTF output).

Path length limit

Allows you to define the maximum "path" length to be shown for items.

- E.g. .../ShortPO/LineItems/LineItem, which would be the maximum length for the default setting 3.

Include

- Allows you to define the specific components to appear in the documentation.

Details

Allows you to set the specific details to appear in the documentation.

- selecting "Library Names" would insert the "core" prefix for functions.
- You can document both connected, as well as unconnected nodes.

Note:
The **Check**/**Uncheck All** buttons allow you to check/uncheck all check boxes of that group.

Having used the default settings shown above, clicking **OK,** prompts you for the name of the output file and the location to which it should be saved. A portion of the fixed design generated documentation is shown below. Note that this shows a single output file.

This table shows the connections **from** the source component to the target component(s).

Mapping **lookup-standard**
(C:\Documents and Settings\My\My Documents\Altova\MapForce2011\MapForceExamples\lookup-standard.mfd)
Input **ShortPO** (ShortPO.xsd)

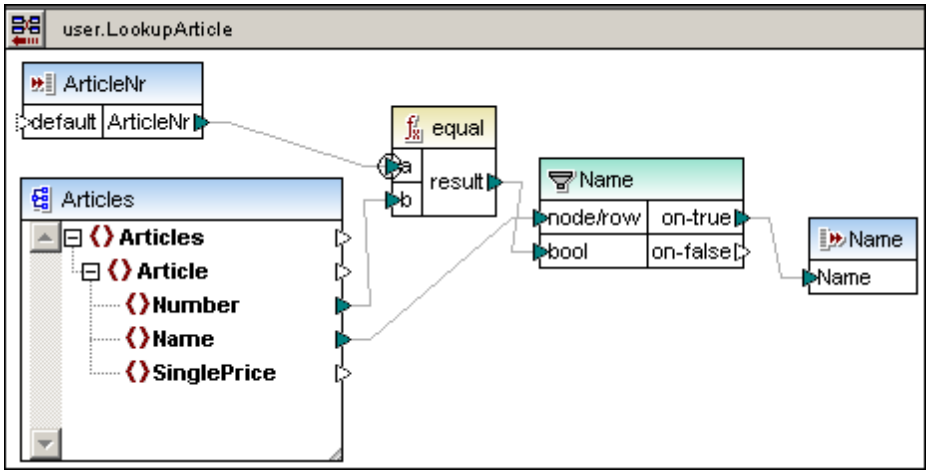
Nodes	Connections	
File: ShortPO.xml Type: string		
ShortPO Type: restriction of xs:anyType [0..1]		
ShortPO/CustomerNr Type: xs:integer		
ShortPO/LineItems Type: restriction of xs:anyType	*direct*	CompletePO/LineItems Type: restriction of xs:anyType
ShortPO/LineItems/LineItem Type: restriction of xs:anyType [1..∞]	*direct*	CompletePO/LineItems/LineItem Type: restriction of xs:anyType [1..∞]
.../LineItems/LineItem/ArticleNr Type: xs:integer	*direct*	.../LineItem/Article/Number Type: xs:integer
	user.LookupArticle => ArticleNr | Name =>	.../LineItem/Article/Name Type: xs:string
.../LineItems/LineItem/Amount Type: xs:integer	*direct*	.../LineItem/Article/Amount Type: xs:integer

The sequence in which the components are documented is: Input, Output, Constant, User-defined functions, then Library functions.

E.g. **Input component** ShortPO:

- The first two items ShortPO and ShortPO/CustomerNr are not connected to any item in the target, thus the Connections column is empty.
- **ShortPO/LineItems** is directly connected to CompletePO/**LineItems** in the target.
- /LineItems/LineItem/**ArticleNr** has two connections:
  - directly to LineItem/Article/**Number** in the target
  - to the User-defined function **LookupArticle,** with ArticleNr as the input parameter, and Name as the output parameter of the user-defined function.

  The contents of the user-defined function are shown below.



Output component CompletePO: This table shows the connections **to** the target component from the source component(s).

Output **CompletePO** (CompletePO.xsd)

Connections		Nodes
		File: CompletePO.xml Type: string
		CompletePO Type: restriction of xs:anyType [0..1]
		CompletePO/Customer Type: restriction of xs:anyType
ShortPO/LineItems Type: restriction of xs:anyType	*direct*	CompletePO/LineItems Type: restriction of xs:anyType
ShortPO/LineItems/LineItem Type: restriction of xs:anyType [1..∞]	*direct*	CompletePO/LineItems/LineItem Type: restriction of xs:anyType [1..∞]
		.../LineItems/LineItem/Article Type: extension of ArticleType
.../LineItems/LineItem/ArticleNr Type: xs:integer	*direct*	.../LineItem/Article/Number Type: xs:integer
.../LineItems/LineItem/ArticleNr Type: xs:integer	**user.LookupArticle** => ArticleNr | Name =>	.../LineItem/Article/Name Type: xs:string

- The first two items CompletePO and CompletePO/Customer are not connected to any item in the source component, thus the Connections column is empty.
- **CompletePO**/LineItems is directly connected to **ShortPO**/LineItems in the source component.
- LineItem/Article/Name is connected to the User-defined function **LookupArticle,** with LineItems/LineItem/ArticleNr as the source item.

User-defined function defines

user.LookupArticle

Input **Articles** (Articles.xsd)

Nodes	Connections	
File: Articles.xml Type: string		
Articles Type: restriction of xs:anyType [0..1]		
Articles/Article Type: ArticleType [1..∞]		
Articles/Article/Number Type: xs:integer	**core.equal** => b | result => **core.filter** => bool | on-true =>	**core.output** => Name Type: string
Articles/Article/Name Type: xs:string	**core.filter** => node/row | on-true =>	**core.output** => Name Type: string
Articles/Article/SinglePrice Type: xs:decimal		

Input **(required) core.ArticleNr**

Nodes	Connections	
ArticleNr Type: string	**core.equal** => a | result => **core.filter** => bool | on-true =>	**core.output** => Name Type: string

12.3.1 Predefined StyleVision Power Stylesheets

Function Call Graphs - PersonListByBranchOffice.mfd

This report shows call graphs of the main mapping and all user-defined functions.

Show all functions Collapse all functions

Main mapping

|---core.equal
|---core.filter
|---user.LookupPerson
| |---core.filter
| |---user.EqualAnd
| | |---core.equal
| | |---core.logical-and
| |---user.Person2Details
| | |---core.concat

user.LookupPerson

|---core.filter
|---user.EqualAnd
| |---core.equal
| |---core.logical-and
|---user.Person2Details
| |---core.concat

user.EqualAnd

|---core.equal
|---core.logical-and

user.Person2Details

|---core.concat

Functions Used By - PersonListByBranchOffice.mfd

Library **core**

Function	Directly used by	Indirectly used by
core.equal	Main mapping user.EqualAnd	user.LookupPerson
core.filter	Main mapping user.LookupPerson	
core.logical-and	user.EqualAnd	Main mapping user.LookupPerson
core.concat	user.Person2Details	Main mapping user.LookupPerson

Library **user**

Function	Directly used by	Indirectly used by
user.LookupPerson	Main mapping	
user.EqualAnd	user.LookupPerson	Main mapping
user.Person2Details	user.LookupPerson	Main mapping

Impact Analysis - PersonListByBranchOffice.mfd

This report lists every input and output node connection independently and is perfect for further impact analysis with modelling tools.

Input Node	Functions	Output Node
OfficeName	core.equal, core.filter	PersonList
OfficeName	user.LookupPerson	PersonList/Person/Details
BranchOffices/Office	core.filter	PersonList
BranchOffices/Office/Name	core.equal, core.filter	PersonList
BranchOffices/Office/Contact		PersonList/Person
.../Office/Contact/first		PersonList/Person/First
.../Office/Contact/first	user.LookupPerson	PersonList/Person/Details
.../Office/Contact/last		PersonList/Person/Last
.../Office/Contact/last	user.LookupPerson	PersonList/Person/Details

Overall Documentation - PersonListByBranchOffice.mfd

Mapping **PersonListByBranchOffice.mfd**

Input **core.OfficeName**

Nodes	Connections	
OfficeName Type: string	**core.equal => a | result =>** **core.filter => bool | on-true =>**	PersonList Type: restriction of xs:anyType [0..1] Annotation: List of Persons
	user.LookupPerson => **Office_Name | result =>**	PersonList/Person/Details Type: xs:string [0..1]

Input **BranchOffices** (BranchOffices.xsd)

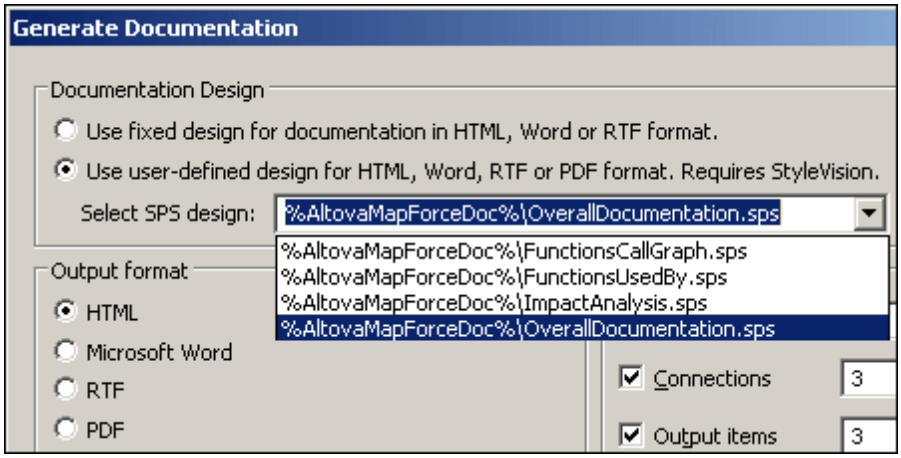
Nodes	Connections	
File: BranchOffices.xml Type: string		
BranchOffices Type: restriction of xs:anyType [0..1]		
BranchOffices/Name Type: restriction of xs:string		
BranchOffices/Office Type: restriction of xs:anyType [0..∞]	**core.filter => node/row | on-true =>**	PersonList Type: restriction of xs:anyType [0..1] Annotation: List of Persons
BranchOffices/Office/Name Type: restriction of xs:string	**core.equal => b | result =>** **core.filter => bool | on-true =>**	PersonList Type: restriction of xs:anyType [0..1] Annotation: List of Persons

12.3.2 Custom Design

Instead of the fixed design, you can create a customized design for the MapForce documentation. The customized design is created in a StyleVision SPS. Note that there are 4 predefined SPS Stylesheets supplied with MapForce, please see Documenting mapping projects.

Specifying the SPS to use for MapForce documentation

The SPS you wish to use for generating the documentation is specified in the Generate Documentation dialog (accessed via **File | Generate Documentation**). Select the "Use User-Defined Design..." radio button then click the dropdown arrow of the combo box and select the file you want. The default selection is the **OverallDocumentation.sps** entry.



These predefined SPS files are located in the ...MapForce2016\Documentation\MapForce folder.

Please note:

To use an SPS to generate documentation, you must have StyleVision installed on your machine.

Creating the SPS

A StyleVision Power Stylesheet (or SPS) is created using Altova's StyleVision product. An SPS for generating MapForce documentation must be based on the XML Schema that specifies the structure of the XML document that contains the MapForce documentation.

This schema is called **MapForceDocumentation.xsd** and is delivered with your MapForce installation package. It is stored in the ...\My Documents\Altova\MapForce2016\Documentation folder.

When creating the SPS design in StyleVision, nodes from the MapForceDocumentation.xsd schema are placed in the design and assigned styles and properties. Note that the MapForceDocumentation.xsd **includes** the Documentation.xsd file located in the folder above it.

Additional components, such as links and images, can also be added to the SPS design. How to create an SPS design in StyleVision is described in detail in the StyleVision user manual.

The advantage of using an SPS for generating mapping documentation is that you have complete control over the design of the documentation. Note also that PDF output of the documentation is available only if an SPS is used; PDF output is not available if the fixed design is used.

12.4 Catalog Files

MapForce supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables MapForce to retrieve commonly used schemas (as well as stylesheets and other files) from local user folders. This increases the overall processing speed, enables users to work offline (that is, not connected to a network), and improves the portability of documents (because URIs would then need to be changed only in the catalog files.)

The catalog mechanism in MapForce works as outlined below.

RootCatalog.xml

When MapForce starts, it loads a file called `RootCatalog.xml` (*structure shown in listing below*), which contains a list of catalog files that will be looked up. You can modify this file and enter as many catalog files to look up as you like, each in a `nextCatalog` element. Each of these catalog files is looked up and the URIs in them are resolved according to the mappings specified in them.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
     xmlns:spy="http://www.altova.com/catalog_ext"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog
Catalog.xsd">
  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/
CustomCatalog.xml"/>
  <nextCatalog catalog="CoreCatalog.xml"/>
  <!-- Include all catalogs under common schemas folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/Schemas"
catalog="catalog.xml" spy:depth="1"/>
  <!-- Include all catalogs under common XBRL folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/XBRL"
catalog="catalog.xml" spy:depth="1"/>
</catalog>
```

In the listing above, notice that in the `Schemas` and `XBRL` folders of the folder identified by the variable `%AltovaCommonFolder%` are catalog files named `catalog.xml`. (The value of the `%AltovaCommonFolder%` variable is given in the table below.)

The catalog files in the Altova Common Folder map the pre-defined public and system identifiers of commonly used schemas (such as SVG and WSDL) and XBRL taxonomies to URIs that point to locally saved copies of the respective schemas. These schemas are installed in the Altova Common Folder when MapForce is installed.You should take care not to duplicate mappings in these files, as this could lead to errors.

CoreCatalog.xml, CustomCatalog.xml, and Catalog.xml

In the `RootCatalog.xml` listing above, notice that `CoreCatalog.xml` and `CustomCatalog.xml` are listed for lookup:

- `CoreCatalog.xml` contains certain Altova-specific mappings for locating schemas in the

Altova Common Folder.

- `CustomCatalog.xml` is a skeleton file in which you can create your own mappings. You can add mappings to `CustomCatalog.xml` for any schema you require but that is not addressed by the catalog files in the Altova Common Folder. Do this using the supported elements of the OASIS catalog mechanism (*see below*).
- There are a number of `Catalog.xml` files in the Altova Common Folder. Each is inside the folder of a specific schema or XBRL taxonomy in the Altova Common Folder, and each maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.

Location of catalog files and schemas

The files `RootCatalog.xml` and `CoreCatalog.xml` are installed in the MapForce application folder. The file `CustomCatalog.xml` is located in your `MyDocuments/Altova/MapForce` folder. The `catalog.xml` files are each in a specific schema folder, these schema folders being inside the folders: `%AltovaCommonFolder%\Schemas` and `%AltovaCommonFolder%\XBRL`.

Shell environment variables and Altova variables

Shell environment variables can be used in the `nextCatalog` element to specify the path to various system locations (*see RootCatalog.xml listing above*). The following shell environment variables are supported:

`%AltovaCommonFolder%`	C:\Program Files\Altova\CommonMapForce
`%DesktopFolder%`	Full path to the Desktop folder for the current user.
`%ProgramMenuFolder%`	Full path to the Program Menu folder for the current user.
`%StartMenuFolder%`	Full path to Start Menu folder for the current user.
`%StartUpFolder%`	Full path to Start Up folder for the current user.
`%TemplateFolder%`	Full path to the Template folder for the current user.
`%AdminToolsFolder%`	Full path to the file system directory that stores administrative tools for the current user.
`%AppDataFolder%`	Full path to the Application Data folder for the current user.
`%CommonAppDataFolder%`	Full path to the file directory containing application data for all users.
`%FavoritesFolder%`	Full path of the Favorites folder for the current user.
`%PersonalFolder%`	Full path to the Personal folder for the current user.
`%SendToFolder%`	Full path to the SendTo folder for the current user.

%FontsFolder%	Full path to the System Fonts folder.
%ProgramFilesFolder%	Full path to the Program Files folder for the current user.
%CommonFilesFolder%	Full path to the Common Files folder for the current user.
%WindowsFolder%	Full path to the Windows folder for the current user.
%SystemFolder%	Full path to the System folder for the current user.
%CommonAppDataFolder%	Full path to the file directory containing application data for all users.
%LocalAppDataFolder%	Full path to the file system directory that serves as the data repository for local (nonroaming) applications.
%MyPicturesFolder%	Full path to the MyPictures folder.

How catalogs work

Catalogs are commonly used to redirect a call to a DTD to a local URI. This is achieved by mapping, in the catalog file, public or system identifiers to the required local URI. So when the DOCTYPE declaration in an XML file is read, the public or system identifier locates the required local resource via the catalog file mapping.

For popular schemas, the PUBLIC identifier is usually pre-defined, thus requiring only that the URI in the catalog file point to the correct local copy. When the XML document is parsed, the PUBLIC identifier in it is read. If this identifier is found in a catalog file, the corresponding URL in the catalog file will be looked up and the schema will be read from this location. So, for example, if the following SVG file is opened in MapForce:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="20" height="20" xml:space="preserve">
  <g style="fill:red; stroke:#000000">
     <rect x="0" y="0" width="15" height="15"/>
     <rect x="5" y="5" width="15" height="15"/>
  </g>
</svg>
```

This document is read and the catalog is searched for the PUBLIC identifier. Let's say the catalog file contains the following entry:

```
<catalog>
  ...
    <public publicId="-//W3C//DTD SVG 1.1//EN" uri="schemas/svg/svg11.dtd"/>
  ...
</catalog>
```

In this case, there is a match for the PUBLIC identifier, so the lookup for the SVG DTD is redirected to the URI schemas/svg/svg11.dtd (this path is relative to the catalog file), and this local file will be used as the DTD. If there is no mapping for the Public ID in the catalog, then the URL in the XML document will be used (in the example above: http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd).

The catalog subset supported by MapForce

When creating entries in CustomCatalog.xml (or any other catalog file that is to be read by MapForce), use only the following elements of the OASIS catalog specification. Each of the elements below is listed with an explanation of their attribute values. For a more detailed explanation, see the XML Catalogs specification. Note that each element can take the xml:base attribute, which is used to specify the base URI of that element.

- <public publicId="PublicID of Resource" uri="URL of local file"/>
- <system systemId="SystemID of Resource" uri="URL of local file"/>
- <uri name="filename" uri="URL of file identified by filename"/>
- <rewriteURI uriStartString="StartString of URI to rewrite" rewritePrefix="String to replace StartString"/>
- <rewriteSystem systemIdStartString="StartString of SystemID" rewritePrefix="Replacement string to locate resource locally"/>

In cases where there is no public identifier, as with most stylesheets, the system identifier can be directly mapped to a URL via the system element. Also, a URI can be mapped to another URI using the uri element. The rewriteURI and rewritsSystem elements enable the rewriting of the starting part of a URI or system identifier, respectively. This allows the start of a filepath to be replaced and consequently enables the targeting of another directory. For more information on these elements, see the XML Catalogs specification.

File extensions and intelligent editing according to a schema

Via catalog files you can also specify that documents with a particular file extension should have MapForce's intelligent editing features applied in conformance with the rules in a schema you specify. For example, if you create a custom file extension .myhtml for (HTML) files that are to be valid according to the HTML DTD, then you can enable intelligent editing for files with these extensions by adding the following element of text to CustomCatalog.xml as a child of the <catalog> element.

```
<spy:fileExtHelper ext="myhtml" uri="schemas/xhtml/xhtml1-transitional.dtd"/>
```

This would enable intelligent editing (auto-completion, entry helpers, etc) of .myhtml files in MapForce according to the XHTML 1.0 Transitional DTD. Refer to the catalog.xml file in the %AltovaCommonFolder%\Schemas\xhtml folder, which contains similar entries.

XML Schema and catalogs

XML Schema information is built into MapForce and the validity of XML Schema documents is

checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema for XML Schema.

The `catalog.xml` file in the `%AltovaCommonFolder%\Schemas\schema` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against either of these schemas. The referenced files are included solely to provide MapForce with entry helper info for editing purposes should you wish to create documents according to these older recommendations.

More information

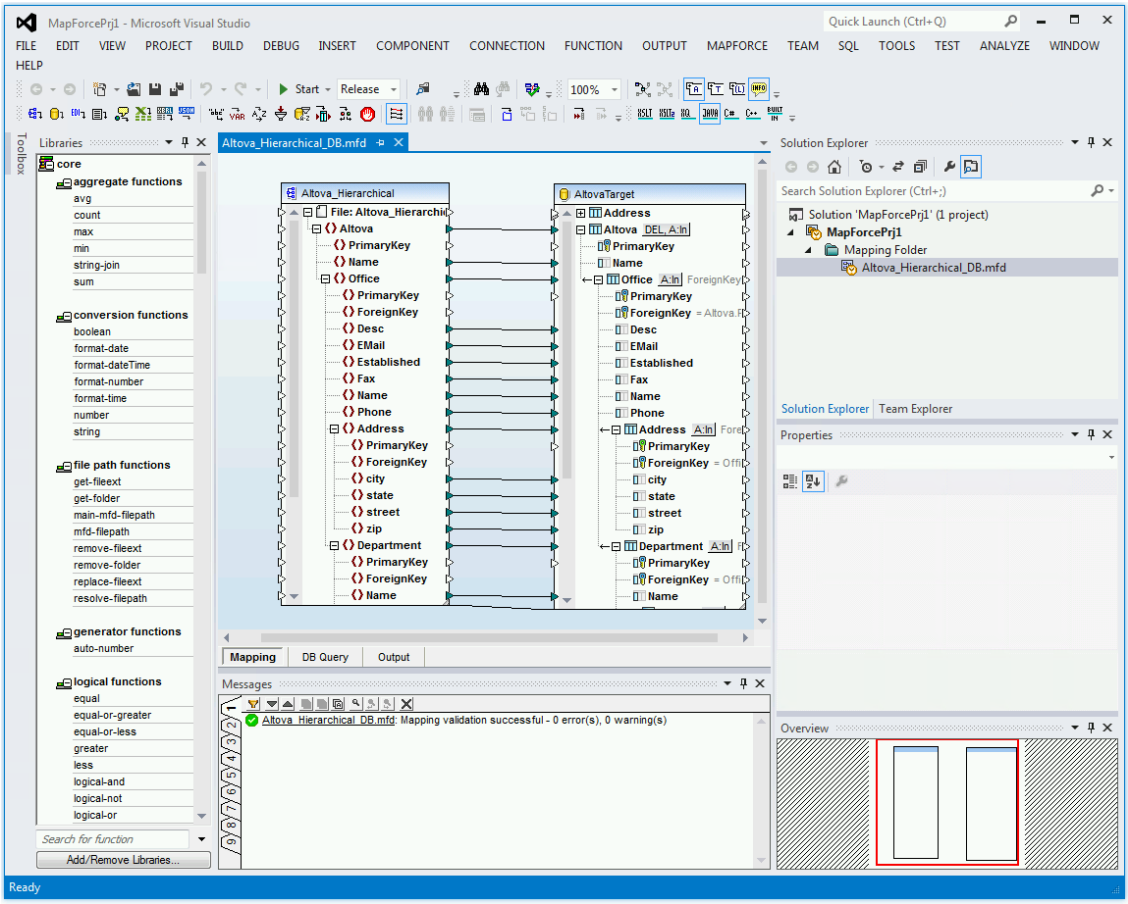
For more information on catalogs, see the XML Catalogs specification.

Chapter 13

MapForce Plug-in for Visual Studio

13 MapForce Plug-in for Visual Studio

You can integrate MapForce 2016 into the Microsoft Visual Studio versions 2005/2008/2010/2012/2013/2015. This unifies the best of both worlds, combining the mapping capabilities of MapForce with the development environment of Visual Studio. When the MapForce plug-in is enabled, you can create mapping projects and files directly from Visual Studio. You can also customize the MapForce options, including menus and toolbars, as you would do in the standalone version of MapForce.



MapForce Enterprise Edition plug-in (Visual Studio 2012)

This section contains the following topics:

- Enabling the Plug-in
- Working with Mappings and Projects
- Accessing Common Menus and Functions

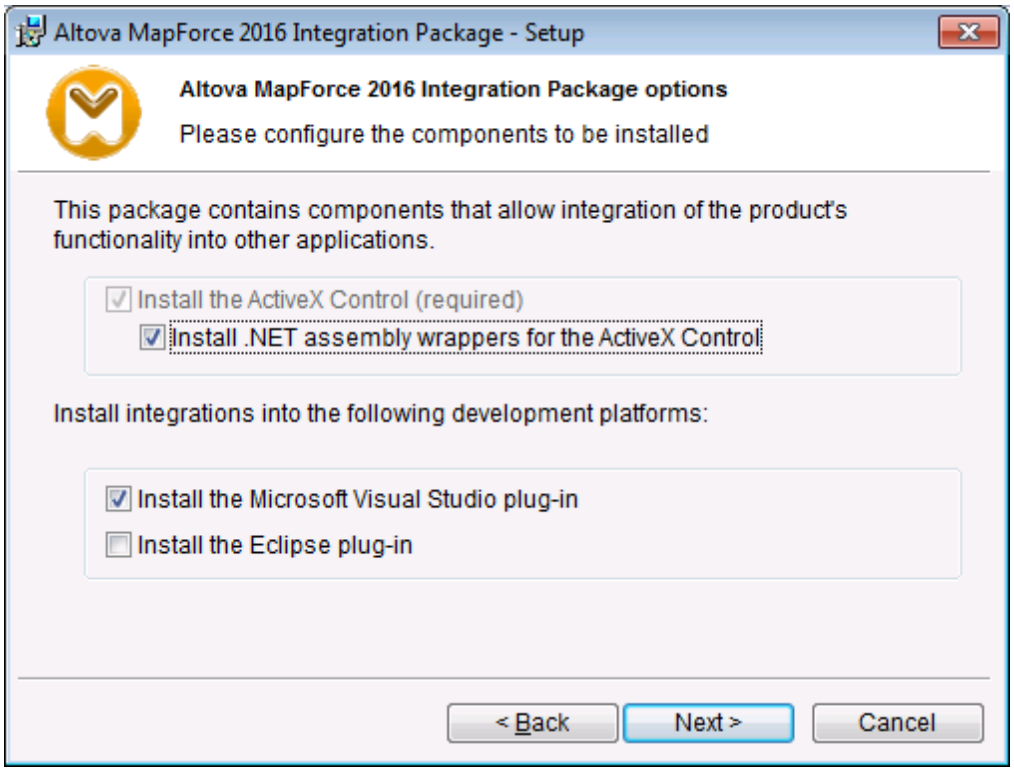
13.1 Enabling the Plug-in

Prerequisites:

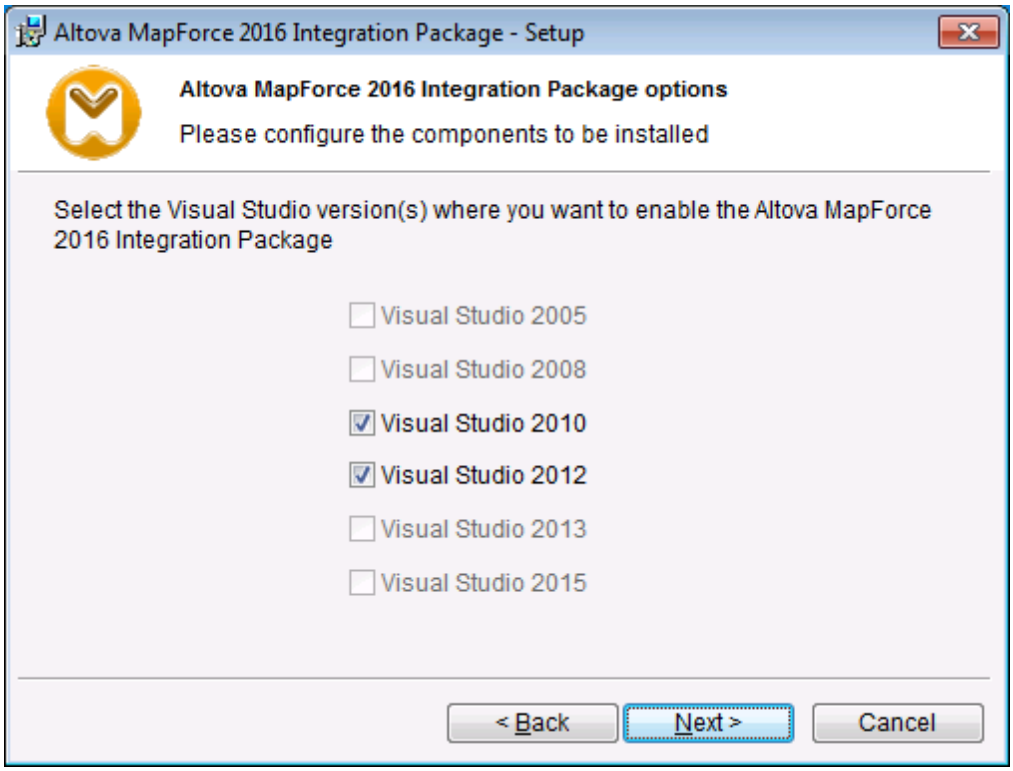
- Microsoft Visual Studio 2005/2008/2010/2012/2013/2015
- MapForce (Enterprise or Professional Edition)
- MapForce Integration Package, available for download at http://www.altova.com/download/mapforce.html.

Note: To use MapForce as a Visual Studio plug-in, install the 32-bit version of both MapForce and MapForce integration package, since there is currently no support for 64-bit plug-ins in Visual Studio.

To enable the MapForce plug-in for Visual Studio, download and run the MapForce Integration Package and follow the on-screen installation instructions. During installation, ensure that the **Install the Microsoft Visual studio plug-in** option is selected:



When prompted, select the Visual Studio version(s) where the plug-in should be enabled, for example:



Note: Only the Visual Studio versions installed on your operating system are available for selection.

Once the integration package has been installed, MapForce functionality becomes available in the Visual Studio environment.

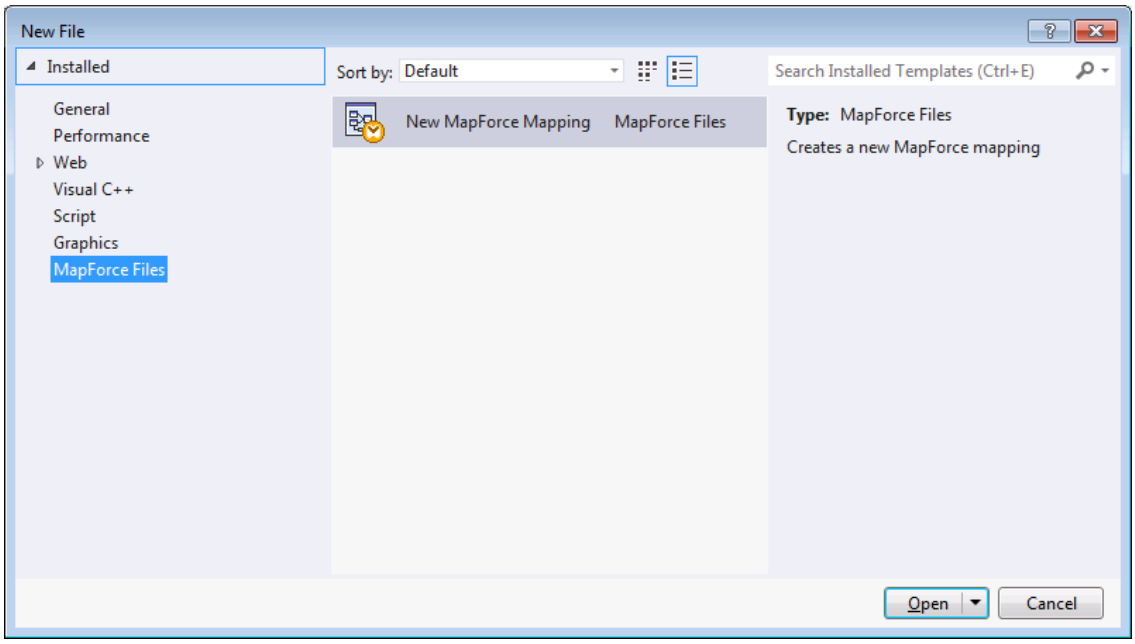
Enabling the MapForce plug-in manually

It is possible that the plug-in was not automatically enabled during the installation process. To enable it, do the following:

1. Navigate to the directory where the Visual Studio IDE executable is installed (for example, c:\Program Files\Microsoft Visual Studio 8\Common7\IDE).
2. Run the command prompt as administrator and enter `devenv.exe /setup`.
3. Wait for the process to terminate normally before starting to use the application within Visual Studio.

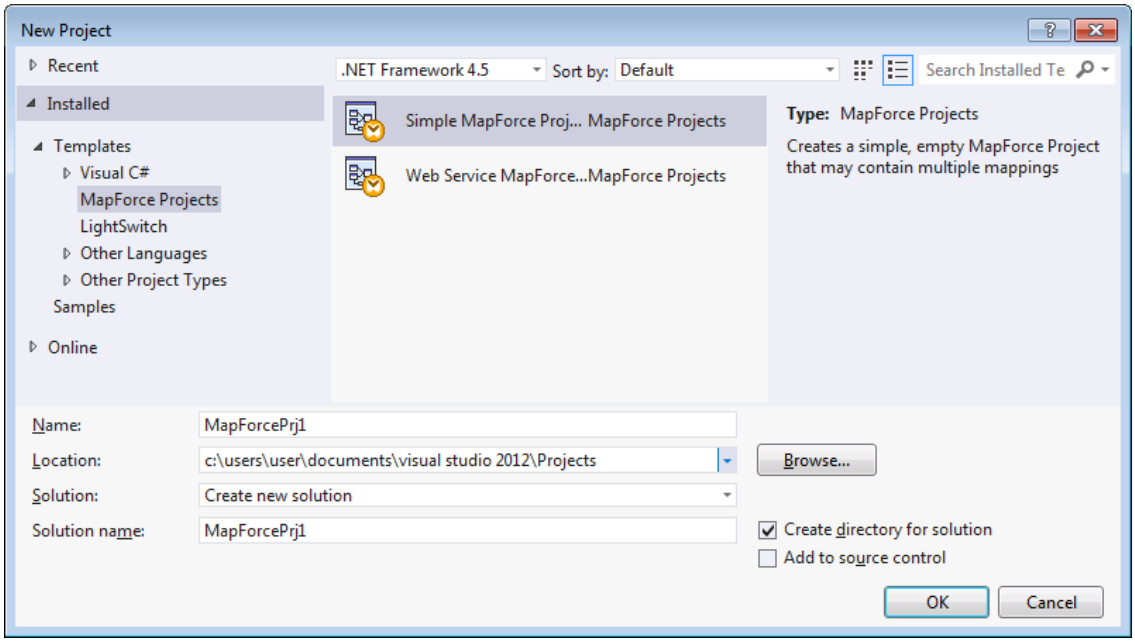
13.2 Working with Mappings and Projects

When MapForce plug-in for Visual Studio is enabled, you create and open mappings and mapping projects in a way that is applicable to the Visual Studio environment, as opposed to the standalone MapForce graphical user interface. For example, when you create a new file in Visual Studio (using the **File | New** menu command), or when you add a new item to a project (using the **Project | Add New Item** menu command), you can select **MapForce Files** as file type.



New File dialog box (Visual Studio 2012 with MapForce Enterprise edition plug-in)

In a similar way, when you create a new Visual Studio project, you can select **MapForce Projects** as project template. The following screen shot illustrates a sample New Project dialog box in Visual Studio 2012 with the MapForce Enterprise Edition plug-in enabled.



New Project dialog box (Visual Studio 2012 with MapForce Enterprise edition plug-in)

Opening existing MapForce files and projects is also done through the Visual Studio native functionality. When you need to open existing mapping files or projects, use the applicable Visual Studio menus (for example, **File | Open | Files**, or **File | Open | Project/Solutions**), and look for the MapForce-related file types.

13.3 Accessing Common Menus and Functions

When MapForce plug-in for Visual Studio is enabled, you can access common menus and functions as shown below. This is the default setup; however, you can change, if desired, the location of menus and toolbars from the **Tools | Options** menu of Visual Studio.

Global Resources

MapForce Global Resources are available in the **MapForce | Global Resources** menu of Visual Studio.

MapForce Options

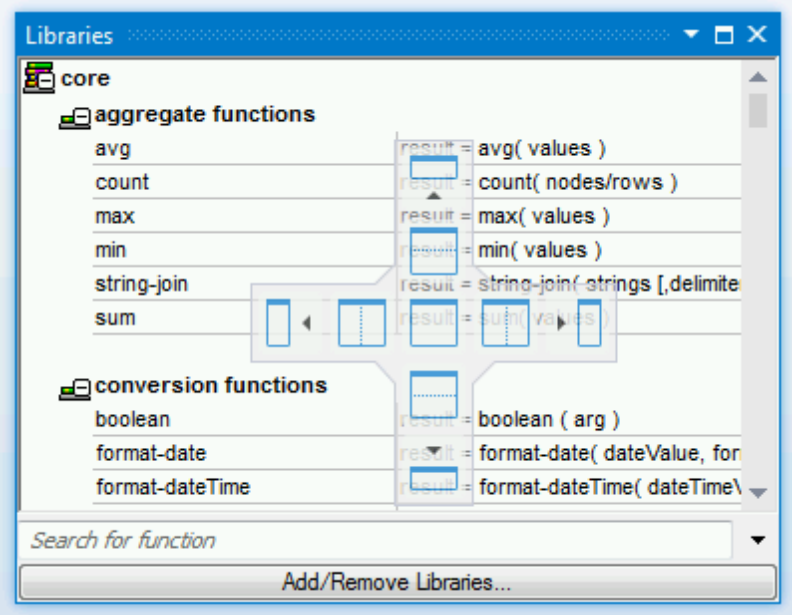
MapForce Options are available in the **Tools | MapForce Options** menu of Visual Studio.

Mapping Pane Customization

When there is a MapForce mapping opened in the main pane of Visual Studio, the **View | MapForce** menu becomes available. It includes the same options as the standalone version of MapForce.

Libraries window

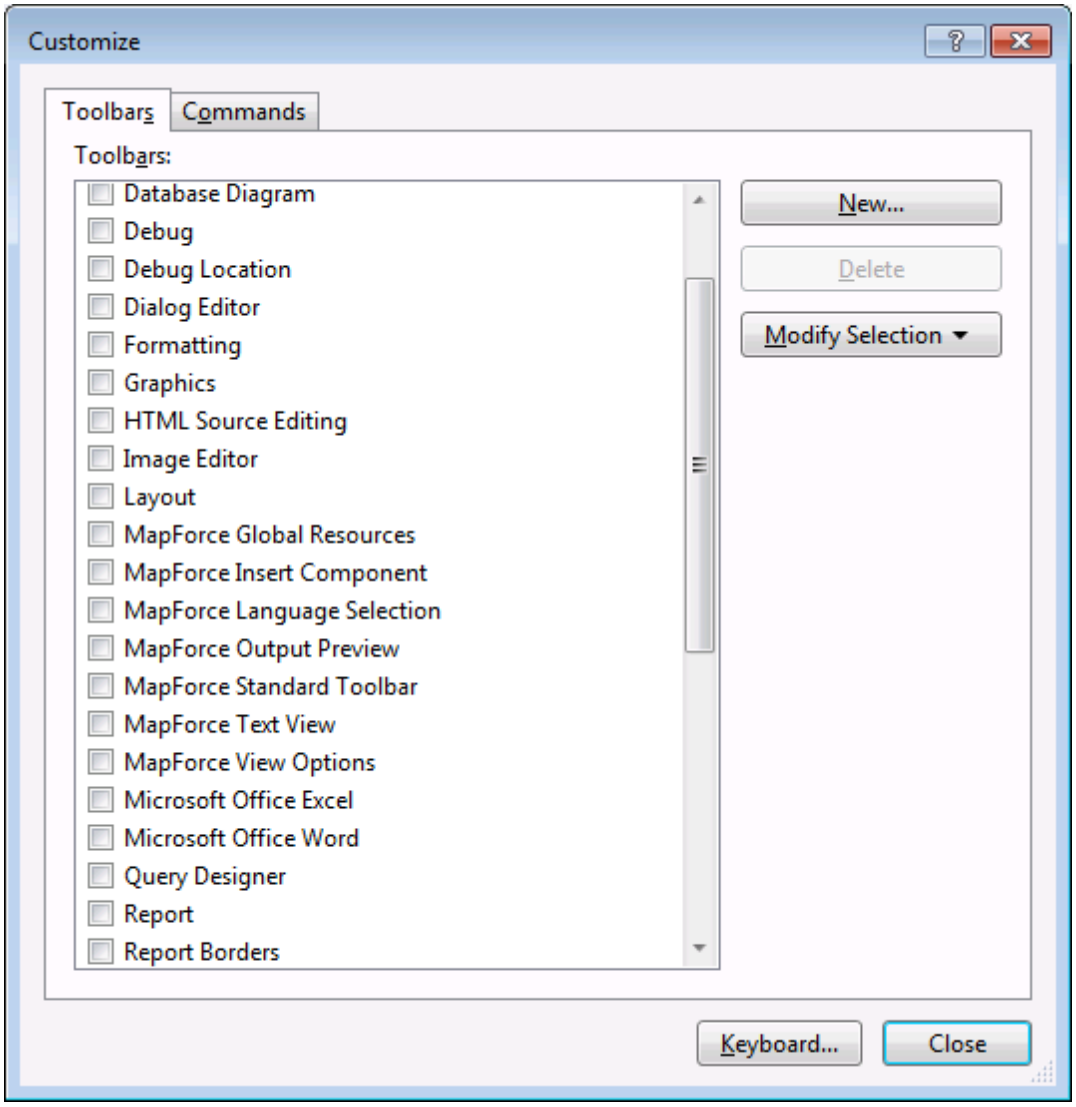
The MapForce Libraries window is not enabled by default in Visual Studio after you install the plug-in. If you work frequently with this window, you can enable it from the **View | MapForce | Library Window** menu (this menu becomes available in Visual Studio when there is a mapping file opened in the main window). Once the Libraries window is enabled, you can doc it to a particular position in the interface, like any other dockable component of Visual Studio.



The Libraries Window (Visual Studio 2012 with MapForce Enterprise edition plug-in)

Toolbar and Commands Customization

You can customize MapForce menus and toolbars from the **Tools | Options** menu of Visual Studio.



Customize dialog box (Visual Studio 2012 with MapForce Enterprise Edition plug-in)

Help and Support

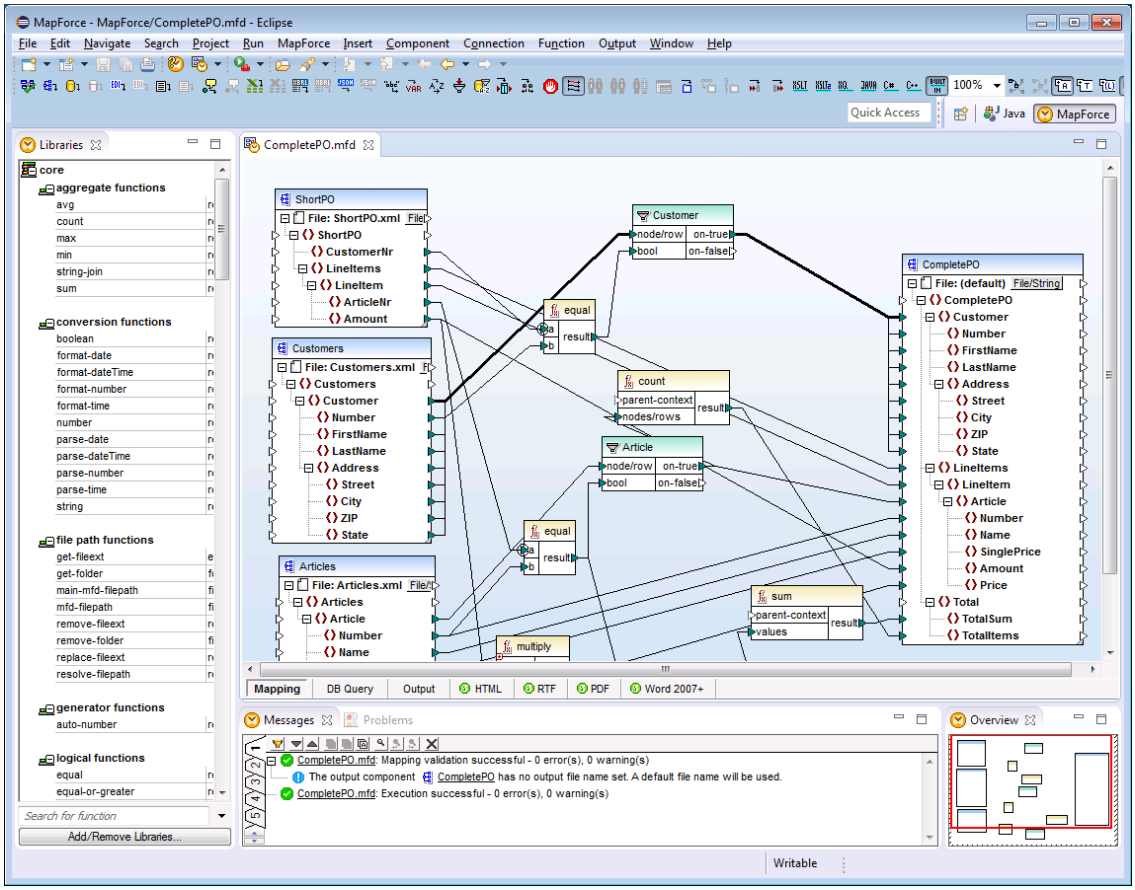
MapForce Help, Support Center, Check for Updates and About menus are available in the **Help | MapForce Help** menu of Visual Studio.

Chapter 14

MapForce Plug-in for Eclipse

14 MapForce Plug-in for Eclipse

Eclipse is an open source framework that integrates different types of applications delivered in form of plug-ins. You can integrate MapForce Enterprise and Professional Edition into Eclipse versions 4.3 / 4.4 / 4.5 and access MapForce functionality directly from Eclipse.



MapForce Enterprise Edition plug-in for Eclipse

The following topics provide help on installing and using the MapForce plug-in for Eclipse.

- Installing the MapForce Plug-in for Eclipse
- The MapForce Perspective
- Accessing Common Menus and Functions
- Working with Mapping and Projects
- Extending MapForce Plug-in for Eclipse

14.1 Installing the MapForce Plug-in for Eclipse

Prerequisites:

- Java Runtime Environment (JRE) 6.0 or later (see http://www.oracle.com/technetwork/java/javase/downloads/index.html). Install a 32-bit or 64-bit JRE to match your version of MapForce (32-bit or 64-bit).
- Eclipse Platform 4.3 / 4.4 / 4.5 (see http://www.eclipse.org). Install a 32-bit or 64-bit Eclipse to match your version of MapForce (32-bit or 64-bit).
- MapForce Enterprise or Professional Edition.

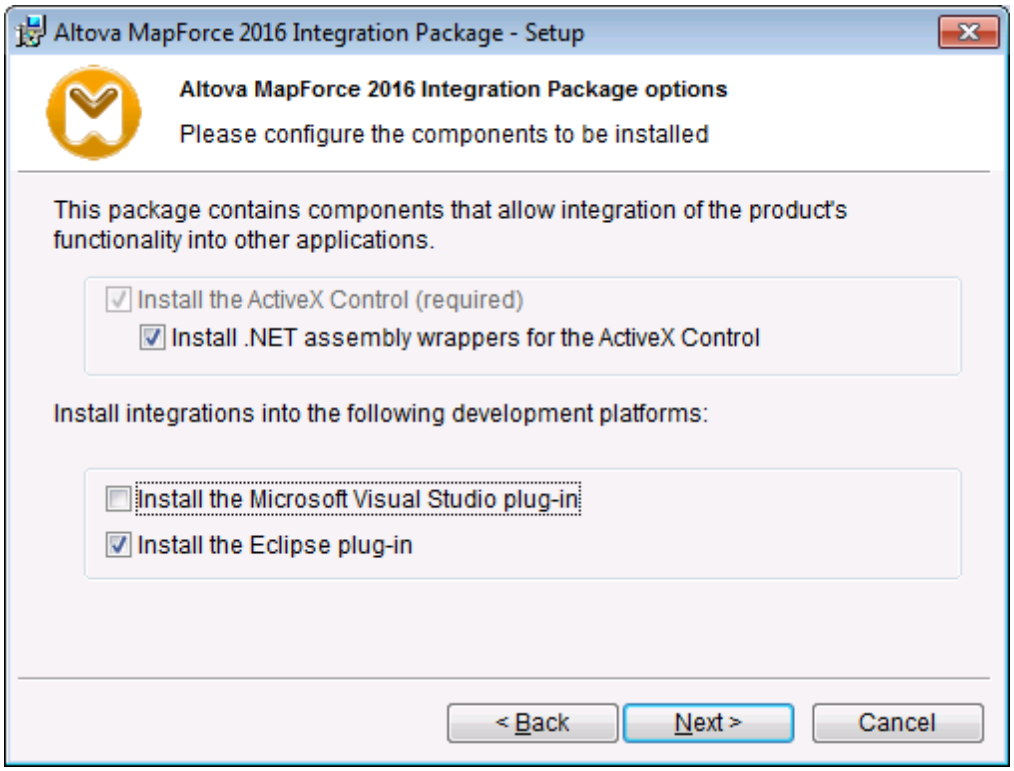
> If you installed Eclipse 4.5 using the Eclipse installer, it is not possible to run on the same machine both the 32-bit and 64-bit versions of the MapForce plug-in. This limitation originates in the Eclipse installer and does not apply if you install manually both versions of Eclipse (32-bit and 64-bit).

Installing the MapForce Plug-in for Eclipse

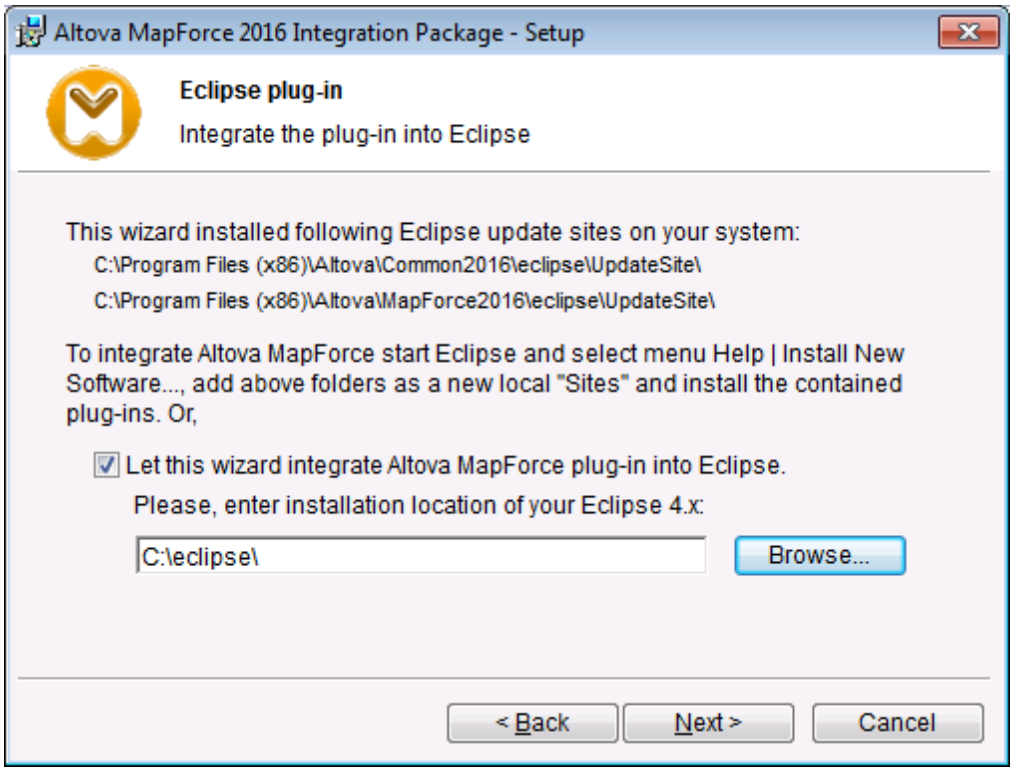
1. Download the MapForce Integration package from the Altova download page (http://www.altova.com/download_components.html).
2. Ensure that Eclipse is not running, and run the downloaded package.

> Eclipse must be closed while you install or uninstall the MapForce Integration Package.

3. When prompted, select the **Install the Eclipse plug-in** option, and then click **Next**.



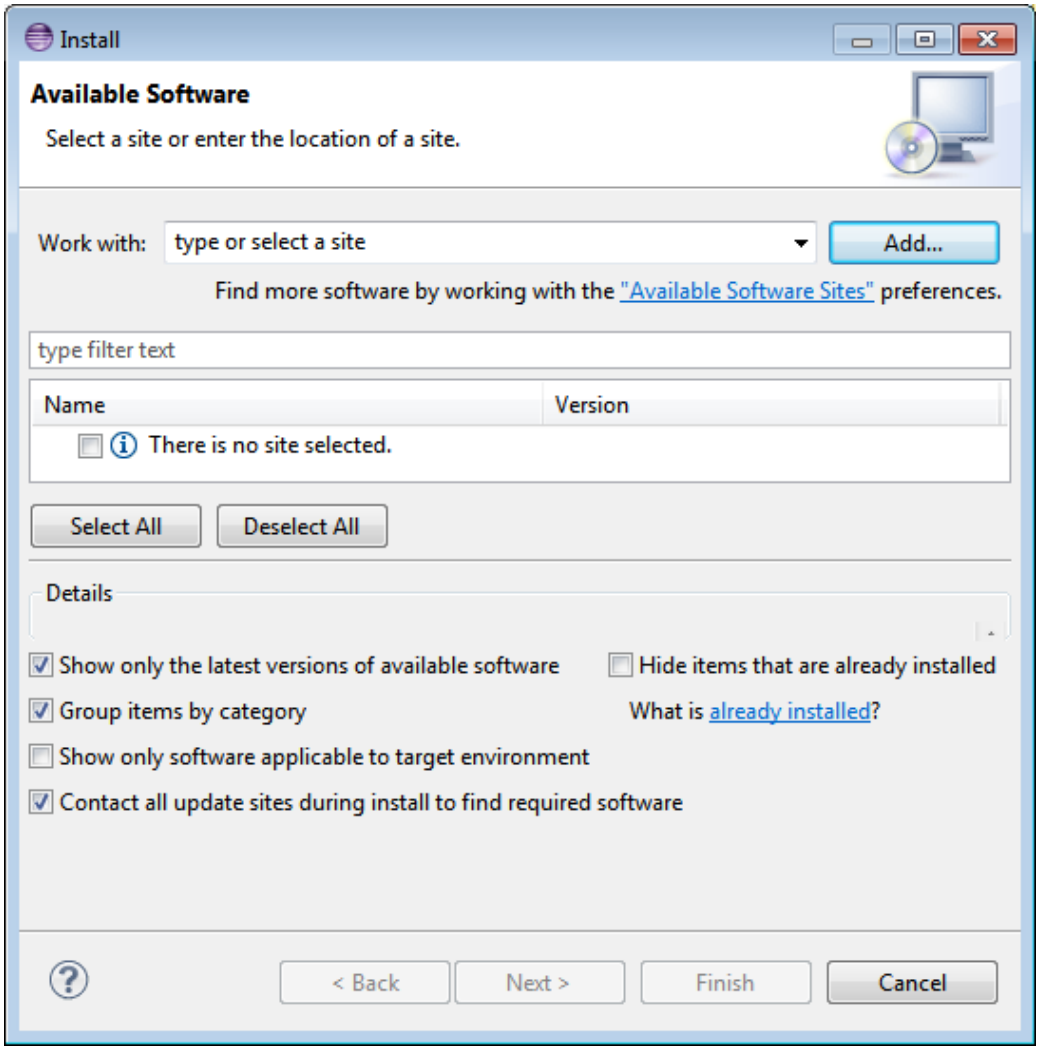
4. When prompted to choose how the MapForce plug-in should be integrated into Eclipse, do one of the following:
   a. To complete the plug-in installation automatically (this is the recommended option), select **Let this wizard integrate Altova MapForce plug-in into Eclipse**, and browse for the directory where the Eclipse executable (`eclipse.exe`) is locatmfed.
   b. To complete the plug-in installation separately in Eclipse, click to clear the **Let this wizard...** check box (see instructions below).



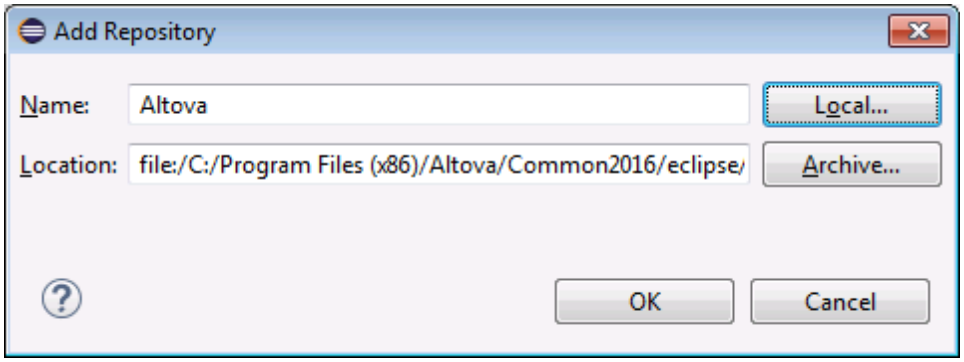
2. Click **Next**, and complete the installation. If you chose the automatic integration, the MapForce perspective and menus become available in Eclipse next time when you start Eclipse.

Integrating the MapForce plug-in for Eclipse manually

1. In Eclipse, click the menu command **Help | Install New Software**.
2. In the Install dialog that pops up (*screen shot below*), click the **Add** button.

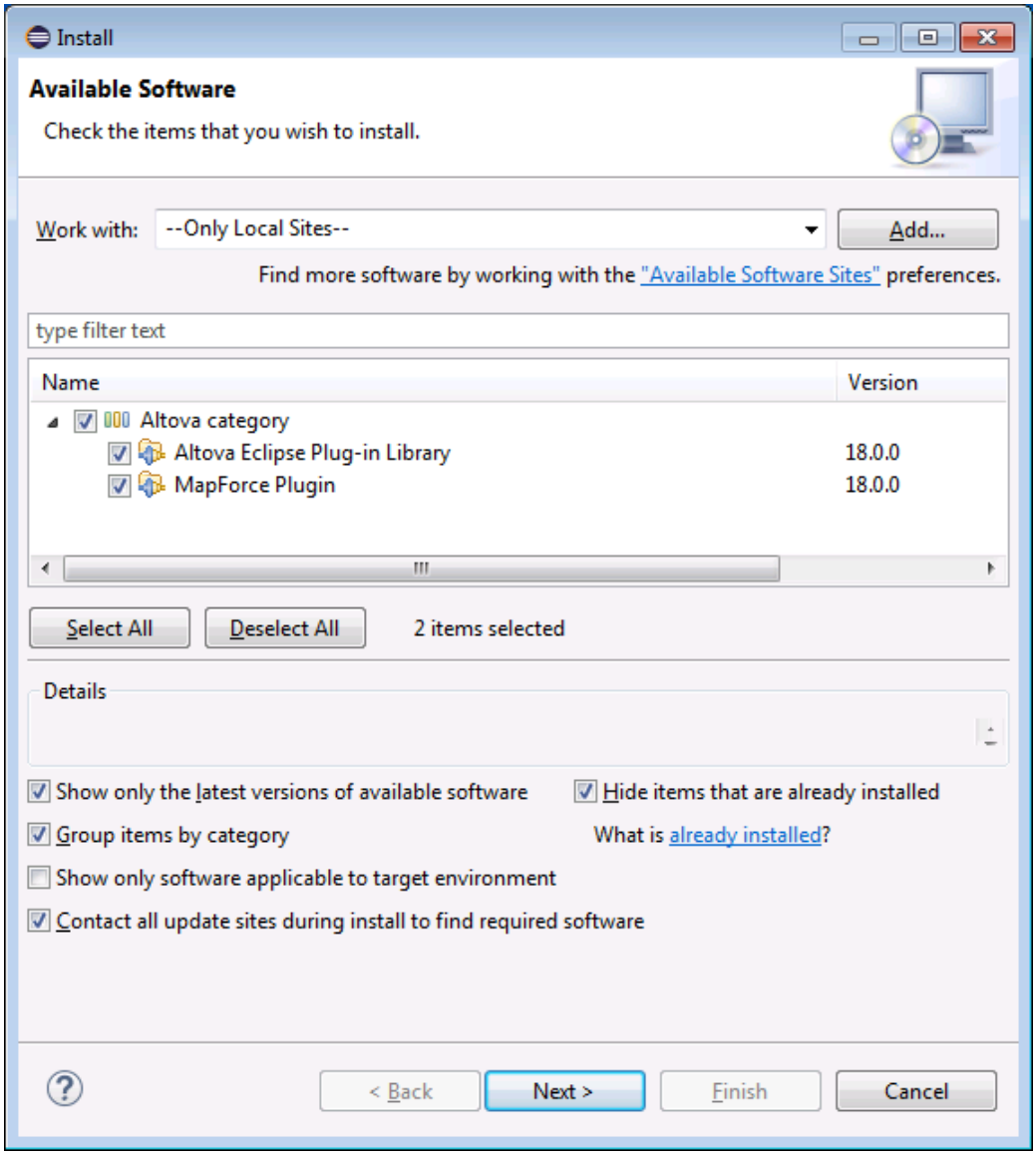


3. In the Add Repository dialog that pops up (*screen shot below*), click the **Local** button.
4. Browse for the folder `c:\Program Files\Altova\Common2016\eclipse\UpdateSite`, and select it. Provide a name for the site (such as 'Altova'), and click **OK**.



5. Repeat Steps 2 to 4, this time selecting the folder `c:\Program Files\Altova\MapForce2016\eclipse\UpdateSite`, and providing a name such as 'Altova MapForce'.
6. In the *Work With* combo box of the Install dialog, select the option *-- Only Local Sites --* (*see screen shot below*). This causes all available plug-ins to be displayed in the pane

below. Check the top-level check box of the *Altova category* folder (*see screen shot below*). Then click the **Next** button.

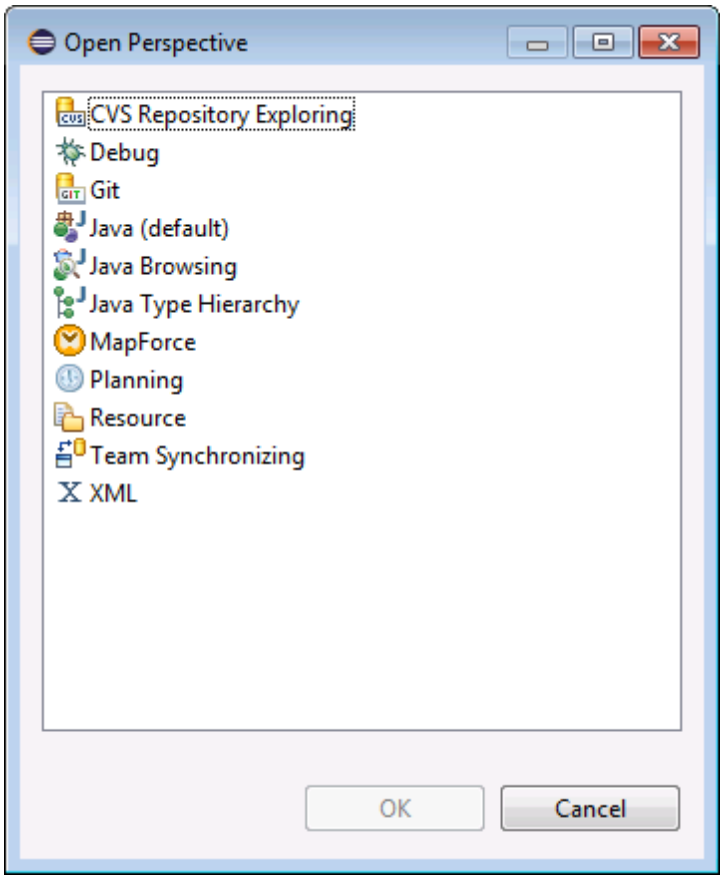


7. An *Install Details* screen allows you to review the items to be installed. Click **Next** to proceed.
8. In the *Review Licenses* screen that appears, select *I accept the terms of the license agreement.* (No license agreement additional to your MapForce Enterprise or Professional Edition license is required for the MapForce plug-in.) Then click **Finish** to complete the installation.

Note: If there are problems with the plug-in (missing icons, for example), start Eclipse from the command line with the `-clean` flag.

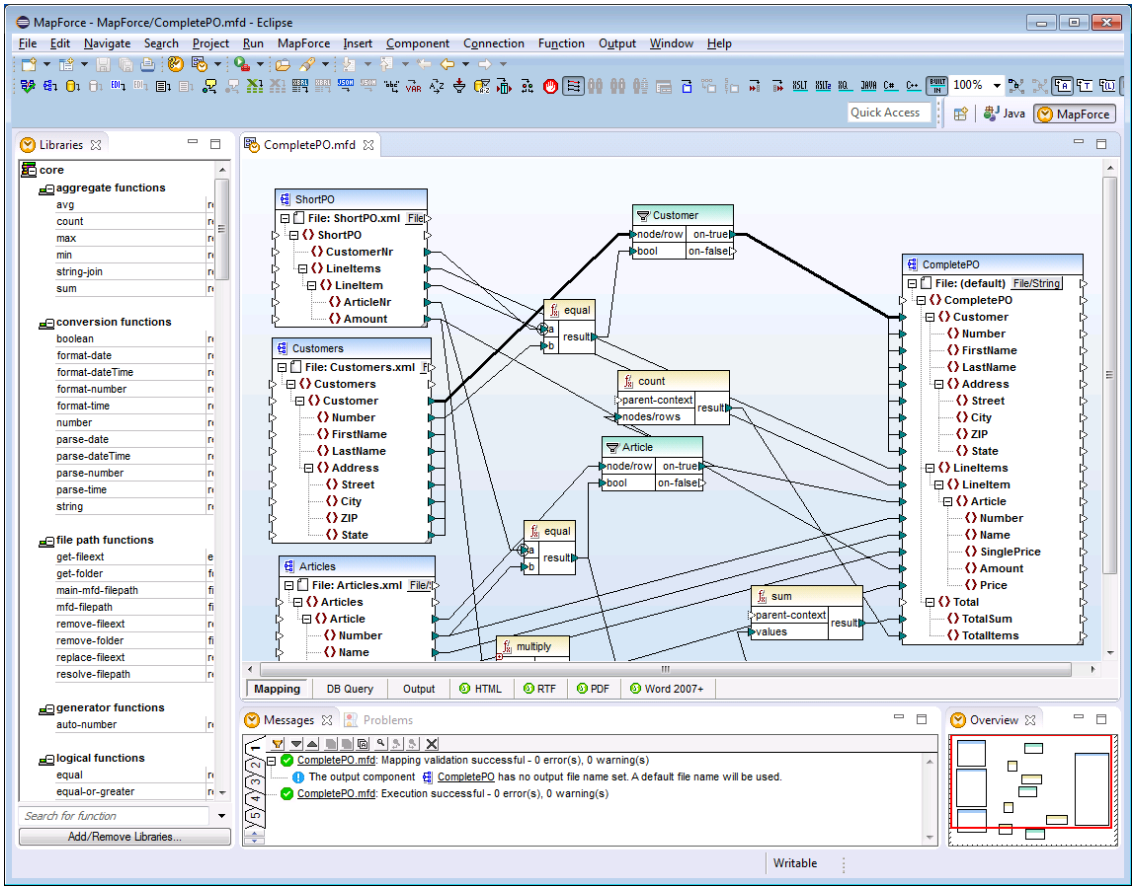
14.2 The MapForce Perspective

After you install the MapForce plug-in for Eclipse, a new perspective ("MapForce") becomes available in Eclipse. The layout of this perspective closely resembles the interface of the standalone edition of MapForce. To switch to the MapForce perspective, click **Window | Open Perspective | Other**, and choose MapForce from the list.



Selecting the MapForce perspective in Eclipse

The MapForce perspective is just like any other Eclipse perspective—you can switch to it whenever required in Eclipse (**Window | Navigation | Next Perspective**). You can also customize the views it contains, and various other options, from Eclipse preferences. (To customize the MapForce perspective in Eclipse 4.4, switch to the MapForce perspective, and then select the menu command **Window | Customize Perspective**). For more information about Eclipse perspectives, refer to Eclipse documentation. The following screen shot illustrates the Eclipse environment with the MapForce perspective switched on.

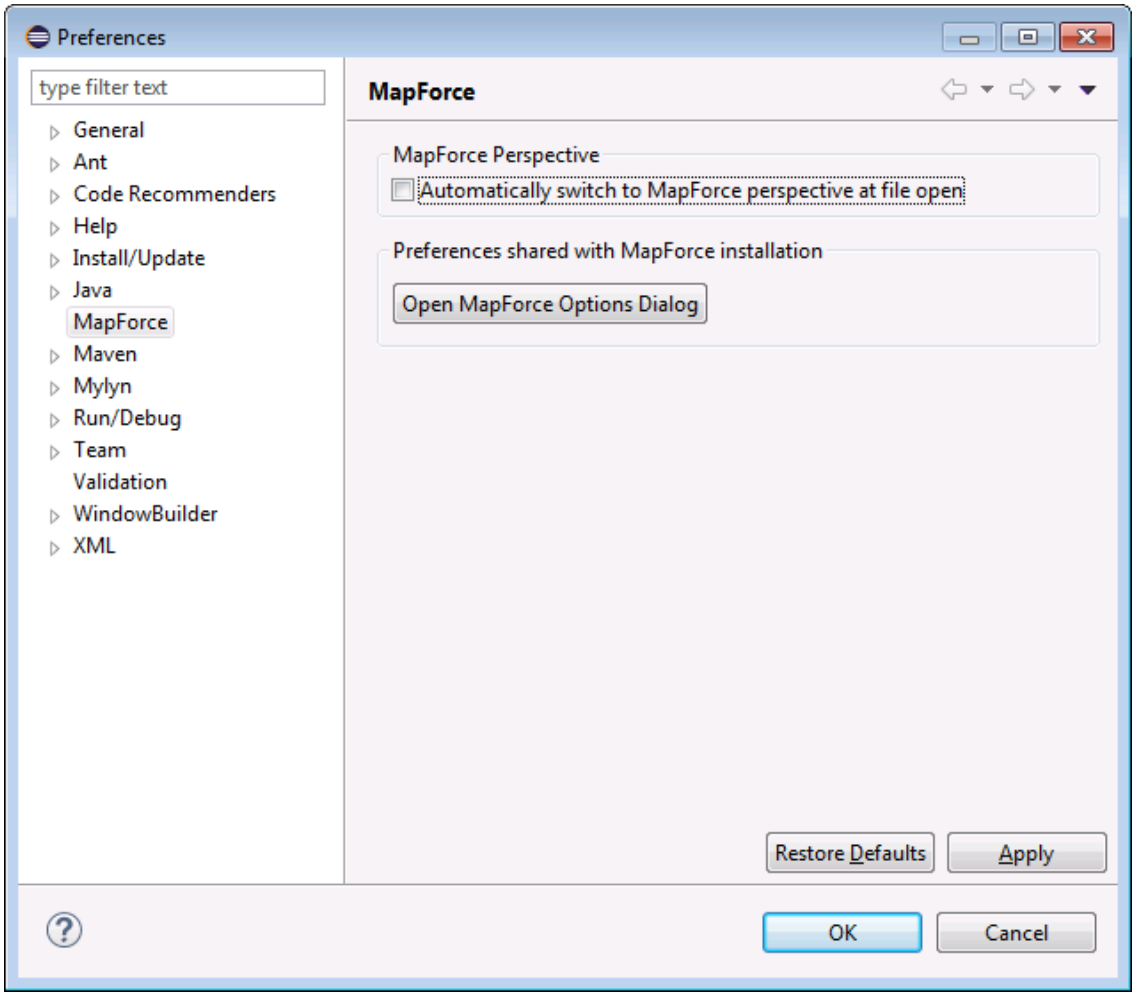


MapForce perspective (MapForce Enterprise Edition plug-in for Eclipse)

By default, the MapForce perspective in Eclipse is organized as follows:

- The mapping design window is available as an Eclipse editor. It has the same tabs and functionality as in the standalone edition of MapForce.
- The Libraries window is available as an Eclipse view, to the left of the main mapping editor. If this view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Libraries**. The Libraries view enables you to work with predefined or custom-defined functions and function libraries.
- The Messages pane is available as an Eclipse view, under the main mapping editor. If the Message view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Messages**. The messages view displays validation messages, errors, and warnings.
- The Overview pane is available as an Eclipse view. If the Overview view is not visible, switch to the MapForce perspective, and then select the menu command **Window | Show View | Overview**. This view enables you to quickly navigate to a particular region on the mapping design area when it is very big.

You can also configure Eclipse to switch to the MapForce perspective automatically when you open a MapForce mapping. To do this, select the menu command **Window | Preferences**. Select MapForce, and then select the **Automatically switch to MapForce perspective at file open** check box.



Preferences dialog box

14.3 Accessing Common Menus and Functions

In Eclipse, you can access most MapForce functionality from the same menus as in the standalone version, except for some Eclipse-specific variations which are listed below. This is the default setup; however, you can further customize the interface preferences from Eclipse, if desired (see The MapForce Perspective).

Note: In Eclipse, some MapForce menu groups or commands are disabled (or not available) if the context is not relevant. For example, the **Insert** menu becomes available only when a mapping design file (.mfd) is active in Eclipse.

For information about the MapForce standard menus, see Menu Reference.

General MapForce commands

In the standalone edition of MapForce, the commands applicable to mapping design files (such as **Validate**, **Deploy to FlowForce Server**, **Generate Code**, and others) are available in the **File** menu. In Eclipse, these commands are available in the **MapForce** menu, or in the MapForce toolbar. Note that the commands for opening or saving files (including MapForce project files) are available in the **File** menu of Eclipse.

The MapForce toolbar in Eclipse

The  toolbar button opens the MapForce help file.

The  toolbar button displays commands specific to MapForce files. When you expand this button, the available commands depend on the kind of file currently active in the Eclipse editor. For example, the commands specific to mapping design (.mfd) files are available when such a file is active (in focus) in the Eclipse editor.

Global Resources

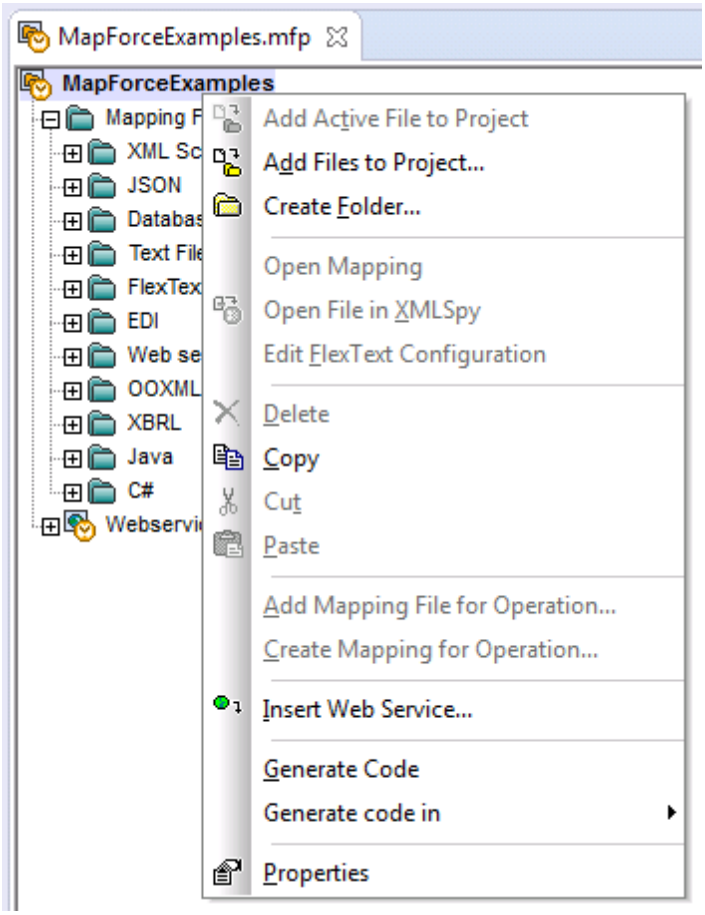
To access or manage Global Resources, do one of the following:

- Click to expand the MapForce  toolbar button, and then click **Global Resources**.
- On the MapForce menu, click **Global Resources**.

MapForce Projects

In the standard edition of MapForce, the **Project** menu contains various commands applicable to mapping project (.mfp) files. In Eclipse, these commands exist as follows:

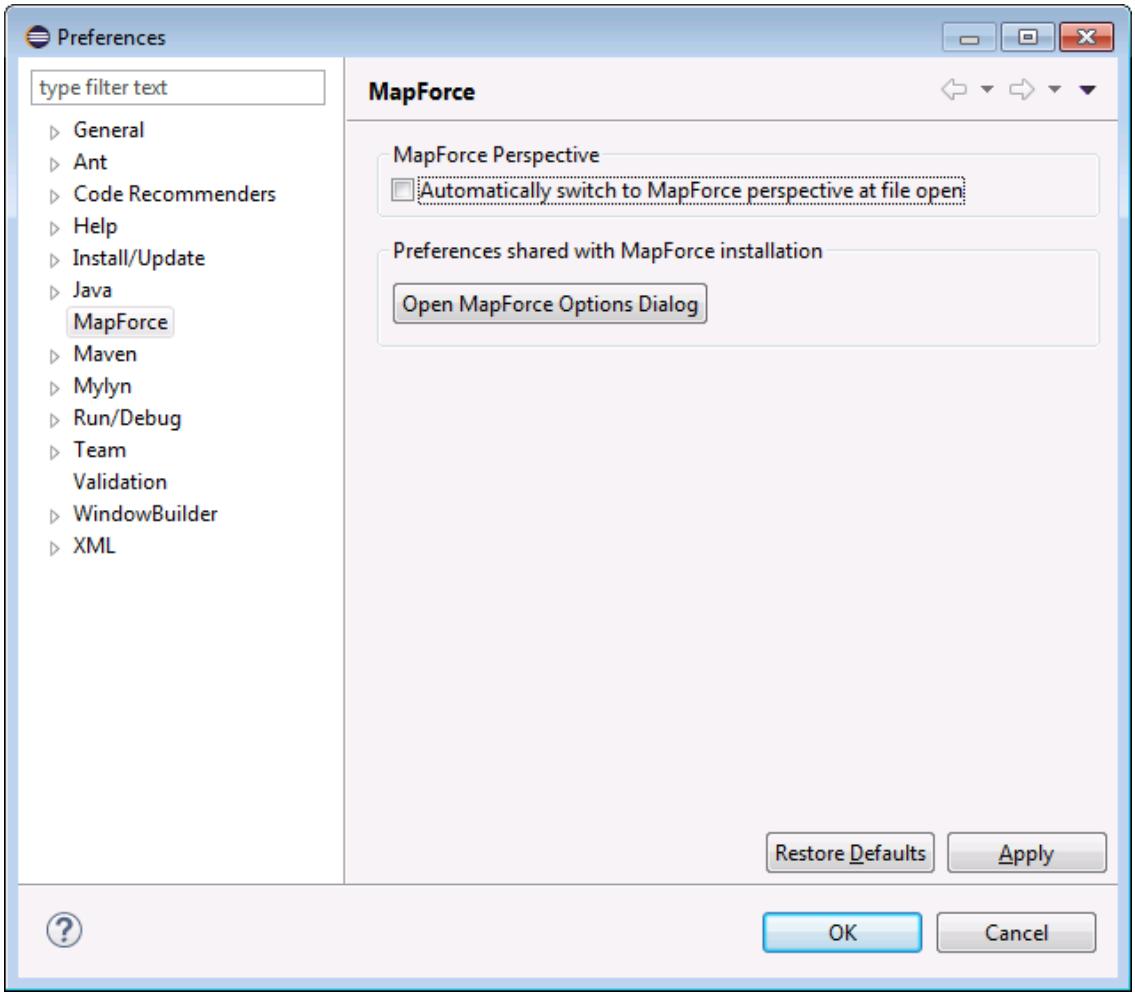
- The commands to open or save a project are available from the Eclipse **File** menu.
- Other project-specific commands are available as context commands. To display the context commands, create or open a MapForce project (.mfp) file in Eclipse, and then right-click the project.



Note that, in addition to standard MapForce projects (.mfp), in Eclipse you can also create projects of type "MapForce/Eclipse". Such projects have a dual nature, and can be configured for automatic build and generation of MapForce code. See Working with Mappings and Projects.

MapForce Options

MapForce options are available from the **Window | Preferences** menu. On the Preferences dialog box, select **MapForce**, and then click **Open MapForce Options Dialog**.



Preferences dialog box

Libraries window

In Eclipse, the MapForce Libraries window is available as a view. This view is by default located to the left of the main editor window. (All MapForce-related views become visible in Eclipse interface when the MapForce perspective is switched on, see also The MapForce Perspective).

MapForce plug-in version

To see the currently installed version of the MapForce Plug-in for Eclipse, select the Eclipse menu option **Help | About Eclipse**. Then select the MapForce icon.

Help and Support

MapForce Help, Support Center, Check for Updates and About menus are available in the **Help | MapForce Help** menu of Eclipse.

14.4 Working with Mappings and Projects

When MapForce plug-in for Eclipse is installed, you can create from Eclipse the same mappings and mapping project types as in the standalone edition of MapForce, from within an Eclipse project. To design, test, compile, and deploy mappings, and to generate mapping code, you can either create a new Eclipse project or use an existing Eclipse project (for example, a Java project to which you want to add MapForce mappings).

In addition to this, you can work with all your mappings within a special project type that becomes available in Eclipse after you install the MapForce plug-in—the **MapForce/Eclipse Project**. Unless you choose to customize it, a MapForce/Eclipse project is by default assigned both a Java Builder and a MapForce Code Generation builder. Additionally, it has two Eclipse natures: MapForce nature and the JDT (Java Development tools) nature. As a result, a MapForce/Eclipse project behaves as follows when you save or change any of its resources (such as a mapping design file):

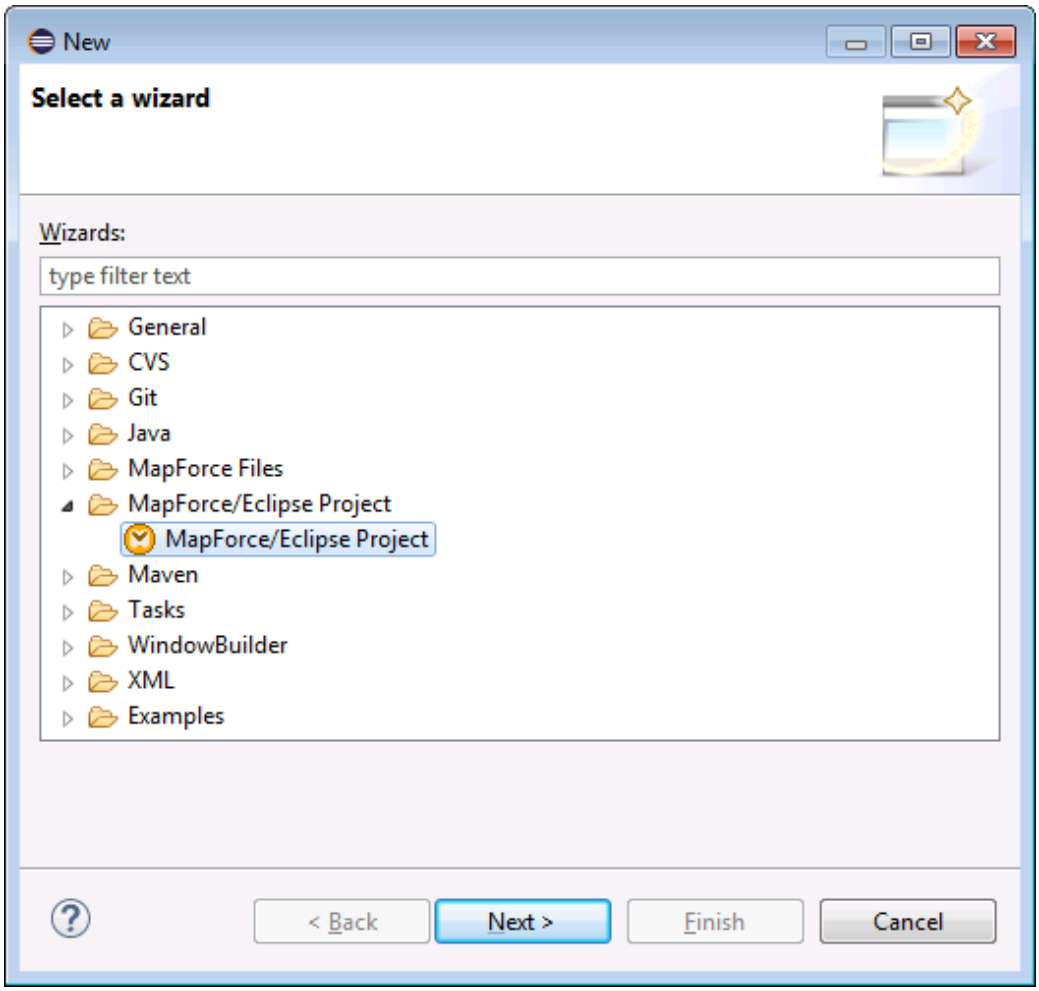
- If the **Project > Build automatically** menu option is enabled, the mapping code is generated automatically. When one or more MapForce project files exist in the MapForce/Eclipse project, the code generation language and output target folders are determined by the settings in each project file. Otherwise, Eclipse prompts you to choose a location.
- Any errors and output messages are shown in the Messages and Problems views.

This section contains the following topics:

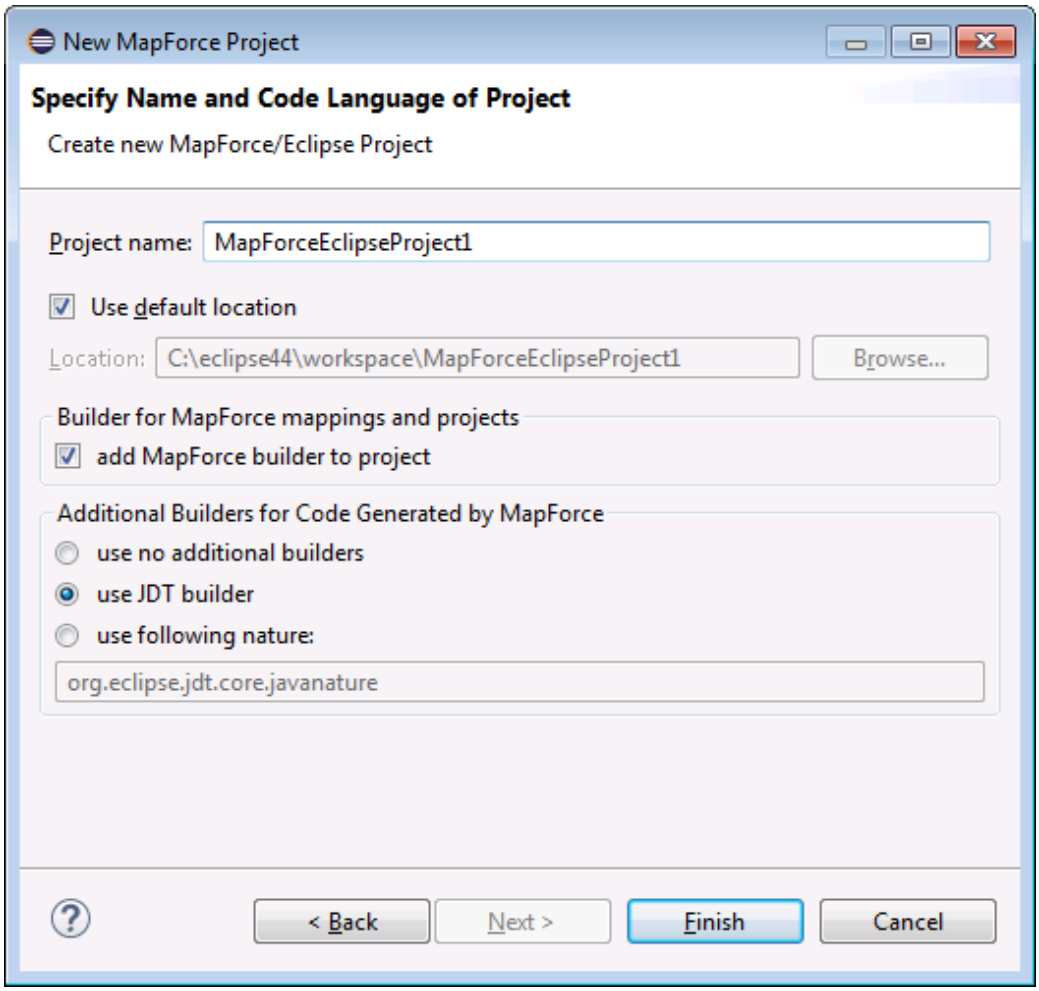
14.4.1 Creating a MapForce/Eclipse Project

To create a MapForce/Eclipse project:

1. On the **File** menu, click **New | Other**.
2. Select the **MapForce/Eclipse Project** category.


New
Select a wizard
Wizards:
type filter text
General
CVS
Git
Java
MapForce Files
MapForce/Eclipse Project
MapForce/Eclipse Project
Maven
Tasks
WindowBuilder
XML
Examples
< Back
Next >
Finish
Cancel

3. Chick **Next**.



4. Enter a project name and choose a location where to save the project. Leave the **add MapForce builder to project** and **use JDT builder** options as is.
5. Click **Finish**.

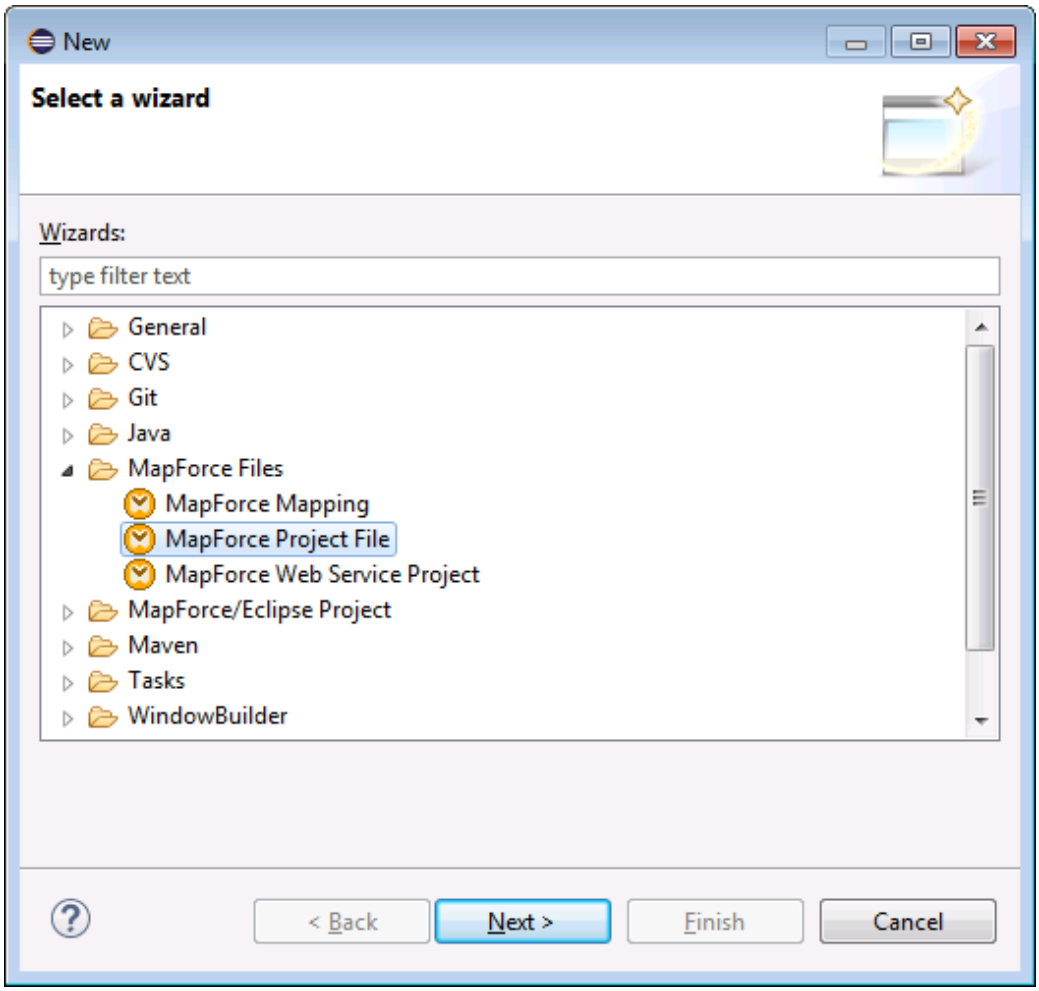
14.4.2 Creating New Mappings

You can create the following MapForce file types within an Eclipse project:

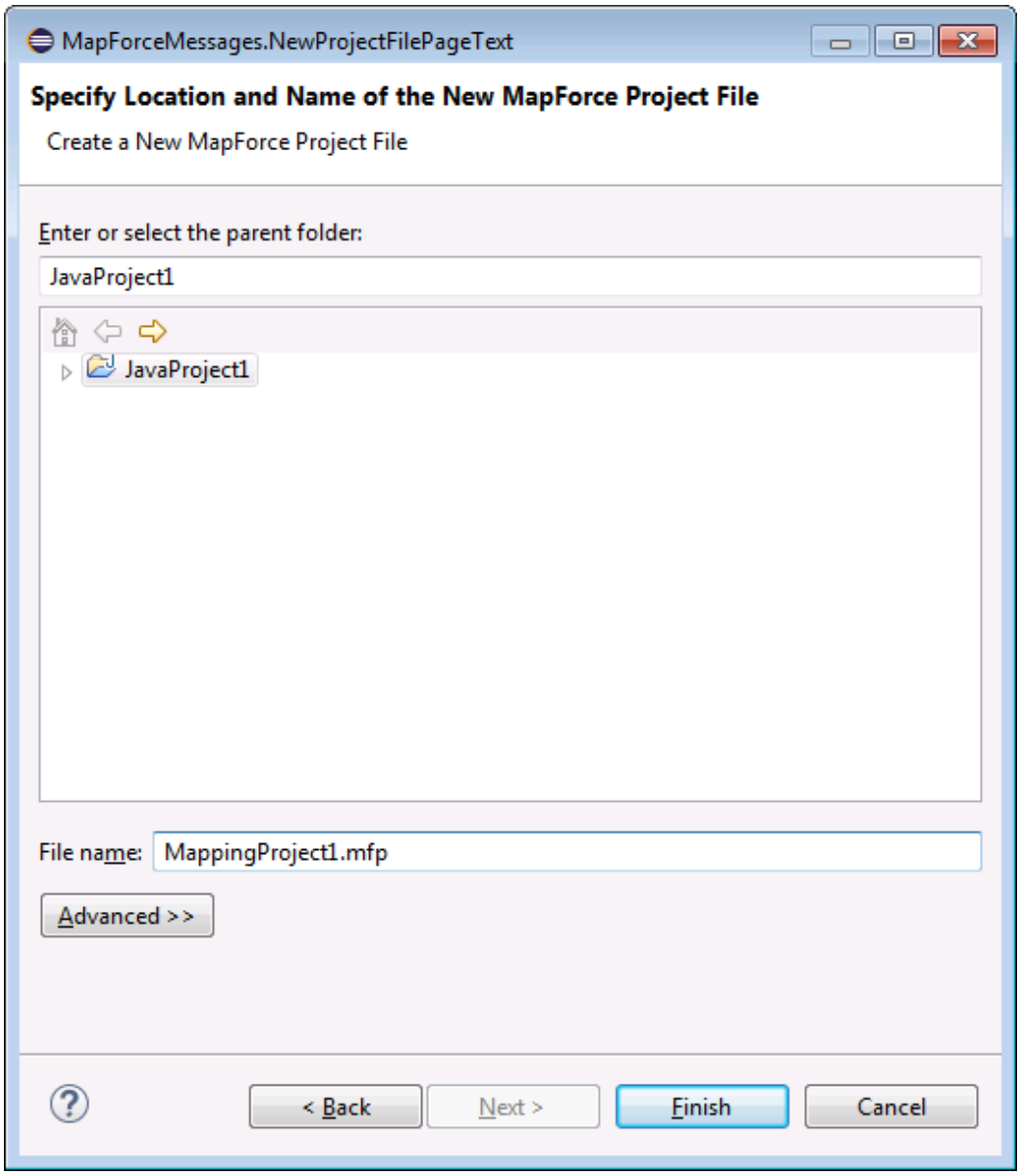
- MapForce mappings
- MapForce project files
- MapForce Web Service projects (available in MapForce Enterprise Edition)

To create any of these file types within an Eclipse project:

1. Create a new Eclipse project or open an existing one.
2. On the **File** menu, click **New**, and then click **Other**.



3. Select the required file type from the wizard dialog box, and then click **Next**.

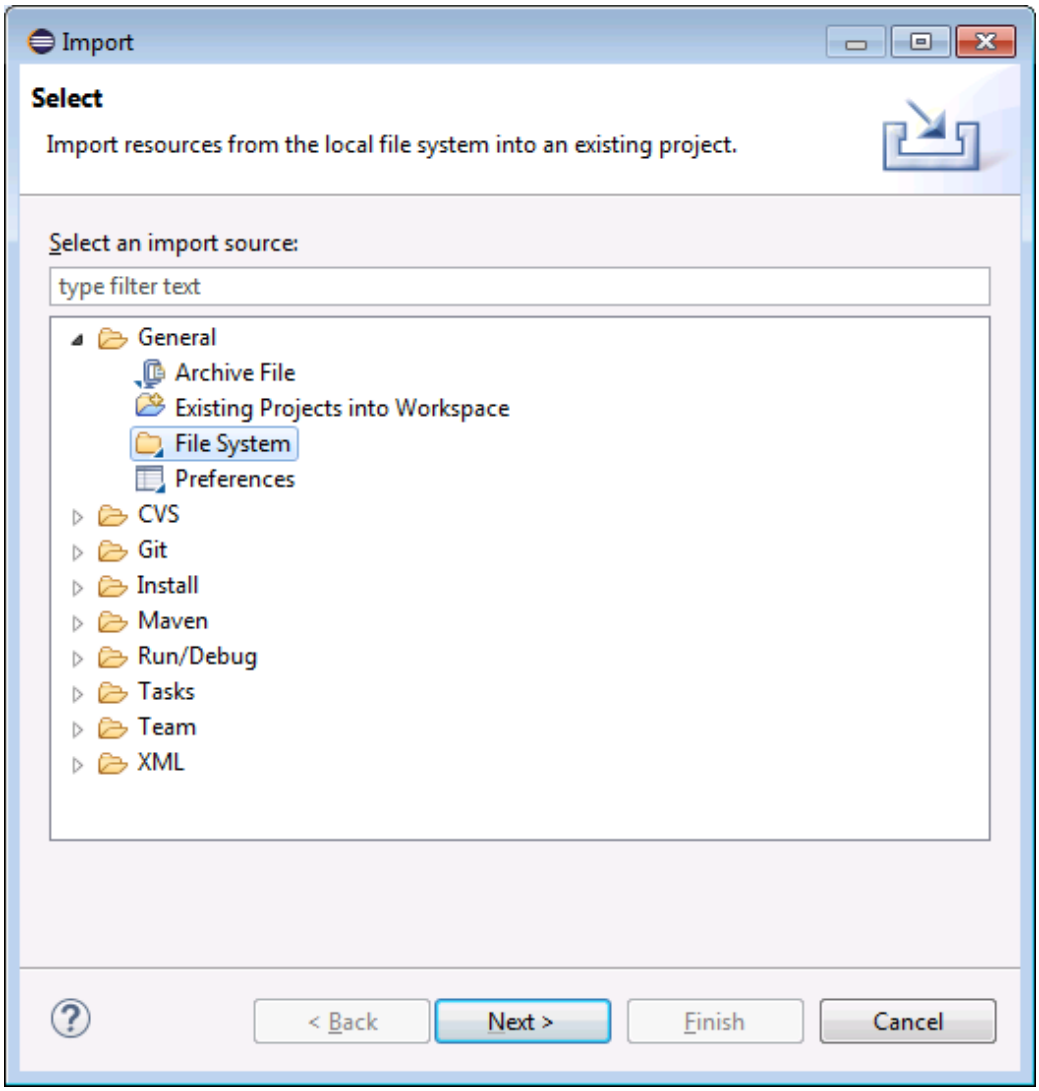


4. Select a parent folder in your existing project, and then click **Finish**.

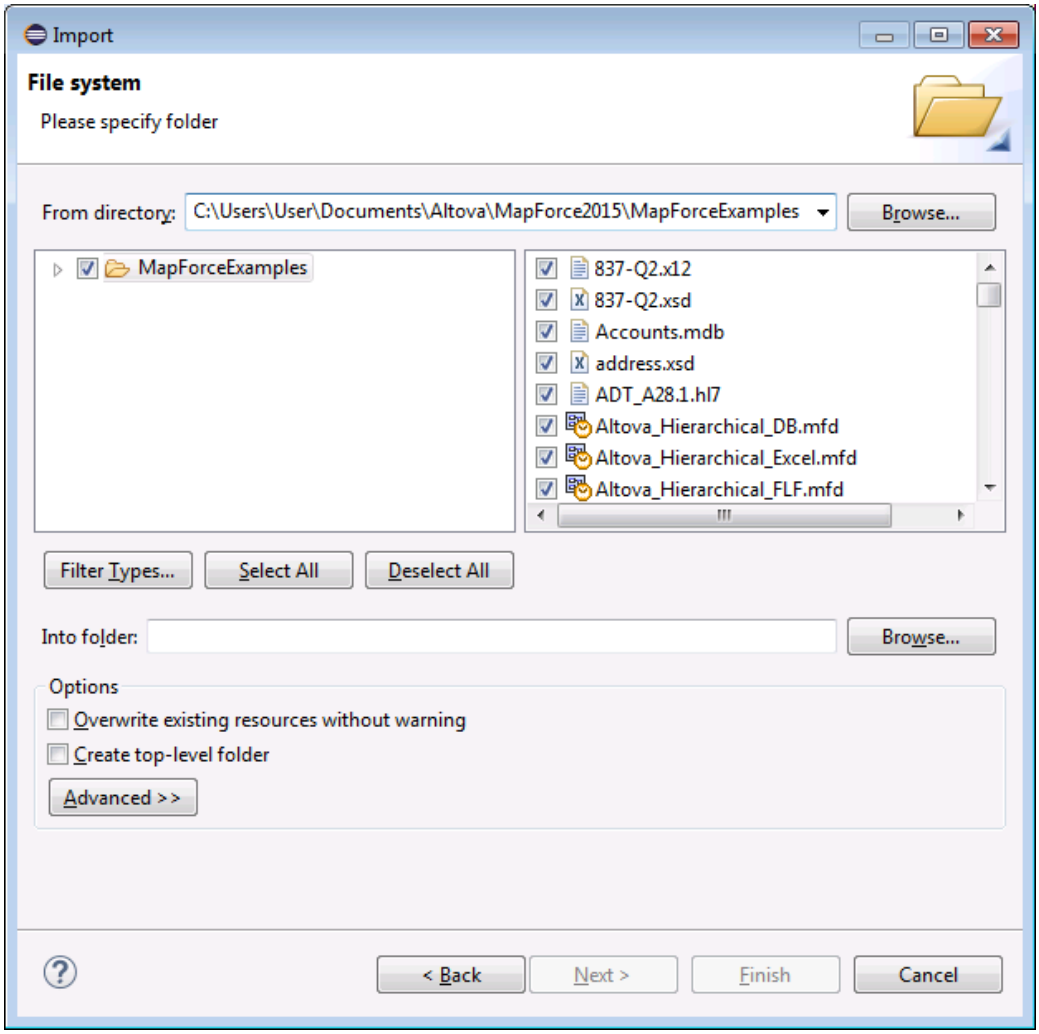
14.4.3 Importing Existing Mappings into an Eclipse Project

To import MapForce mappings and their dependent files into an existing Eclipse project:

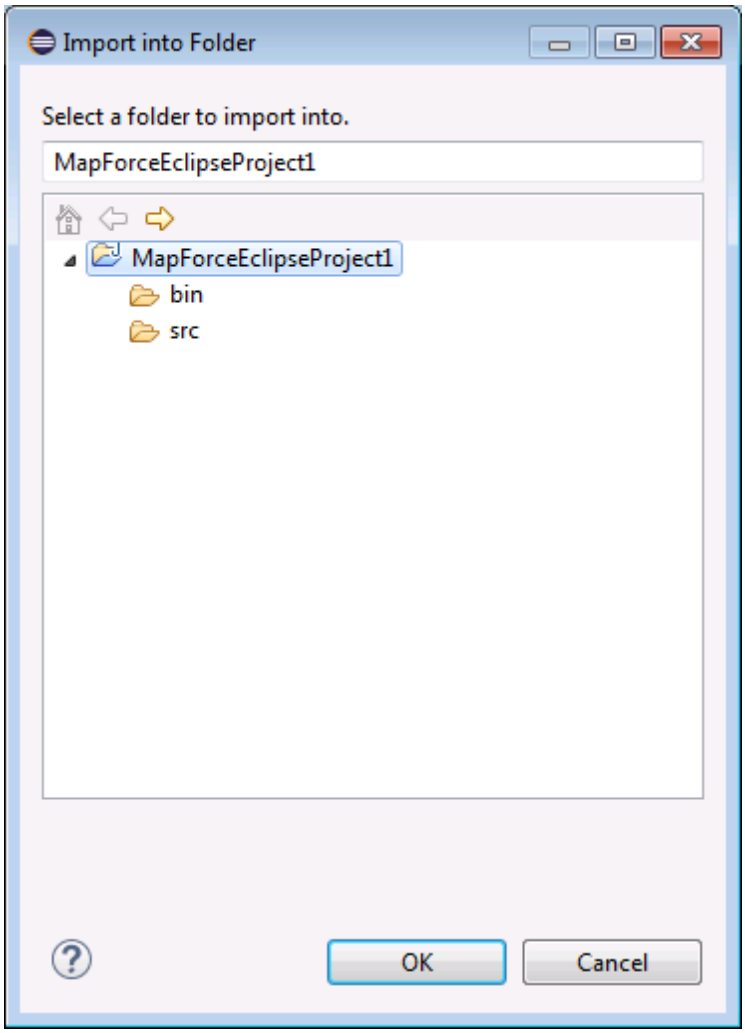
1. Open the project into which you want to import the files.
2. On the **File** menu, click **Import**.



3. Select **File System**, and then click **Next**.



4. Next to **From directory**, browse for the location of the files you want to import, and then select the required files.
5. Next to **Into folder**, click **Browse**, and select the project into which you are adding the files (in this example, *MapForceEclipseProject1*).



6. Click **OK**, and then click **Finish**.

14.4.4 Configuring Automatic Build and Generation of MapForce Code

Automatic MapForce code building and generation is enabled by default in any MapForce/Eclipse project (see Creating a MapForce/Eclipse Project). If you want to enable automatic build and generation of MapForce code in an existing project which is not of type *MapForce/Eclipse*, you can do this by manually adding to it the *MapForce Code Generation* builder and the *MapForce* nature.

To add the MapForce Code Generation builder to a project:

- Add to the Eclipse **.project** file the lines highlighted below:

```
<buildSpec>
    <buildCommand>
       <name>org.eclipse.jdt.core.javabuilder</name>
```

```
        <arguments>
        </arguments>
      </buildCommand>
      <buildCommand>
        <name>com.altova.mapforceeclipseplugin.MapForceBuilder</name>
        <arguments>
        </arguments>
      </buildCommand>
   </buildSpec>
```

To add the MapForce nature to a project:

- Add to the Eclipse **.project** file the lines highlighted below:

```
   <natures>
      <nature>org.eclipse.jdt.core.javanature</nature>
      <nature>com.altova.mapforceeclipseplugin.MapForceNature</nature>

   </natures>
```

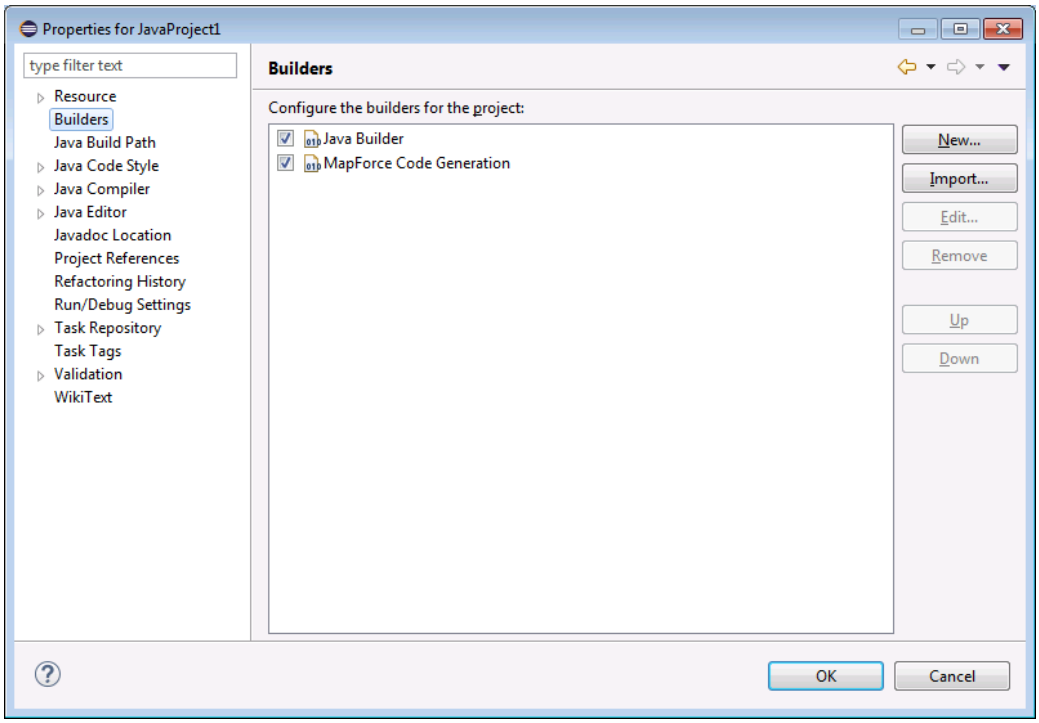
Tip: You can quickly open the **.project** file from the Navigator view of Eclipse (To enable this view, select the menu command **Window | Show View | Navigator**).

To switch automatic MapForce code generation on/off:

- On the **Project** menu, click **Build automatically**.

To disable the MapForce Code Generation builder:

1. On the **Project** menu, click **Properties**.
2. Click **Builders**.



3. Click to clear the **MapForce Code Generation** check box.

14.5 Extending MapForce Plug-in for Eclipse

The MapForce plug-in for Eclipse provides an Eclipse extension point with the ID **com.altova.mapforceeclipseplugin.MapForceAPI**. You can use this extension point to adapt, or extend the functionality of the MapForce plug-in. The extension point gives you access to the COM-Interface of the MapForce control and the MapForce API.

The MapForce Eclipse installation package contains a simple example of a plug-in that uses this extension point. It checks for any file open events of any new MapForce mappings, and sets the zoom level of the mapping view to 70%.

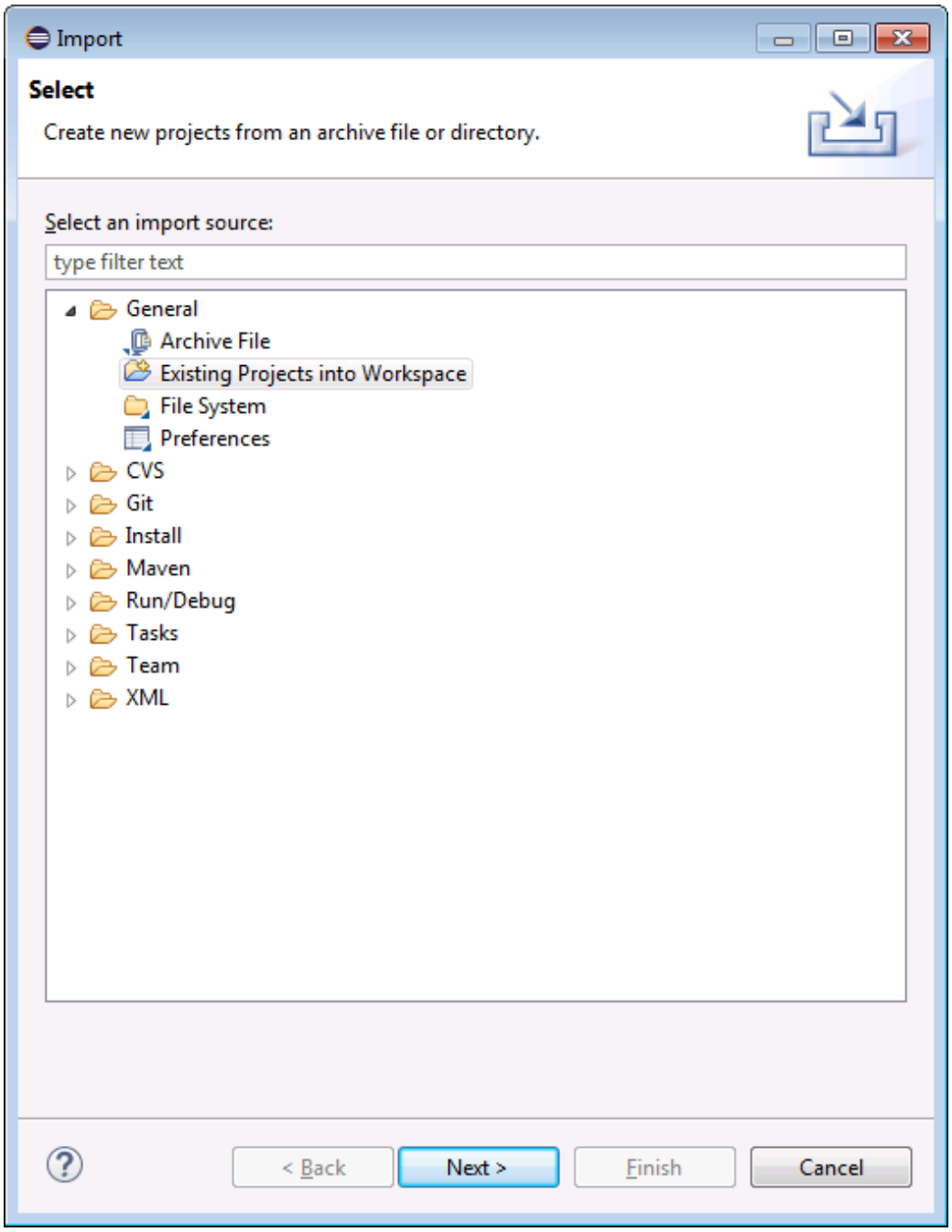
The JavaDoc documentation of the extension point is available in the MapForce plug-in installation directory (typically, **C:\Program Files\Altova\MapForce2016\eclipse\docs**).

Before you install and run the sample MapForce plug-in, ensure that the following prerequisites are met:

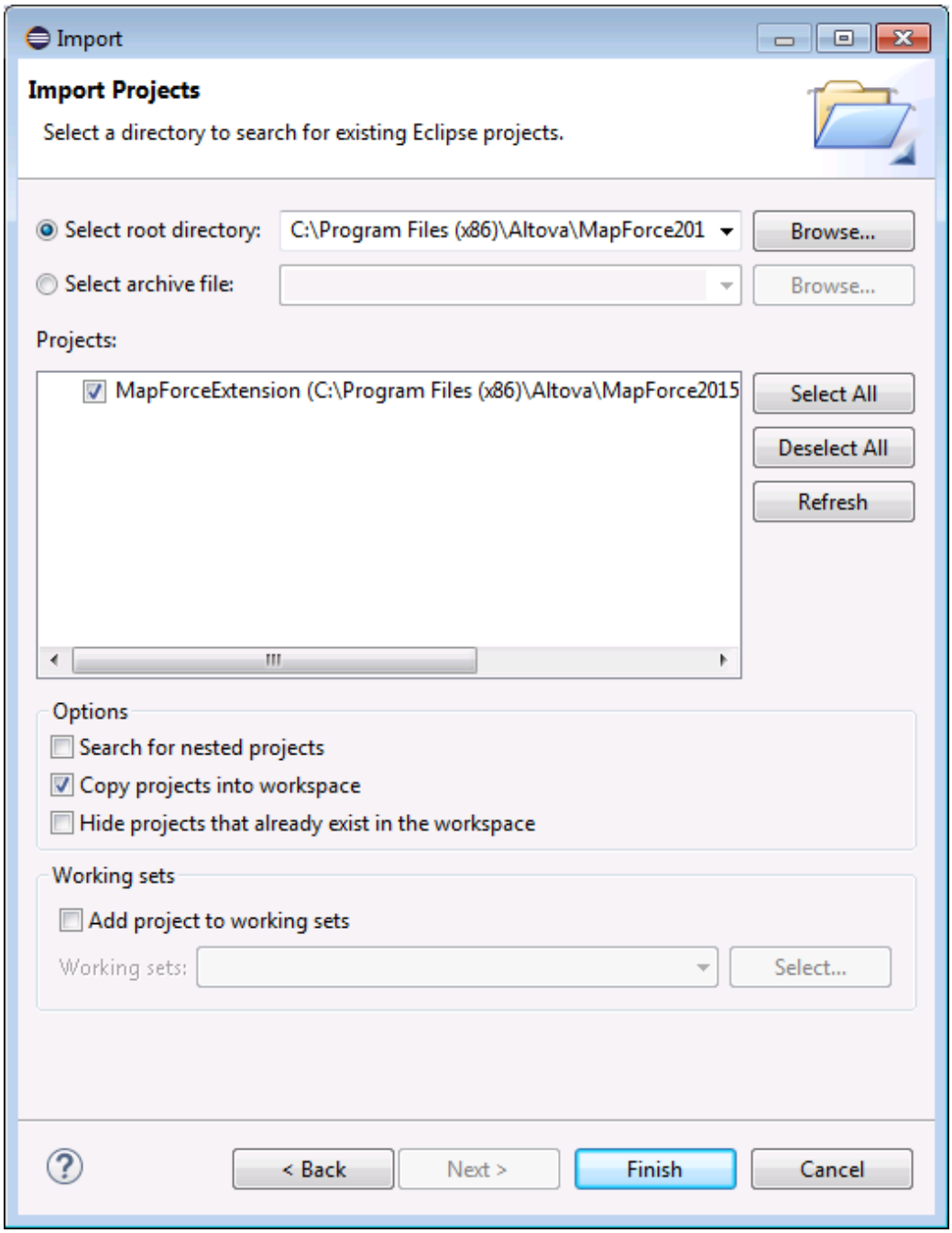
- You are using 32-bit Java, 32-bit Eclipse, 32-bit MapForce and 32-bit MapForce Integration Package.
- The JDT (Java Development Tools) plug-in is installed.
- The Eclipse PDE (plug-in development environment) is installed.

To import the sample MapForce plug-in project into your workspace:

1. Start Eclipse.
2. On the **File** menu, click **Import**.
3. Select **General | Existing projects into Workspace**, and click **Next**.



4. Click the **Browse**... button next to the "'Select root directory" field and choose the sample project directory e.g. **C:\Program Files\Altova\MapForce2016\eclipse\workspace\MapForceExtension.**

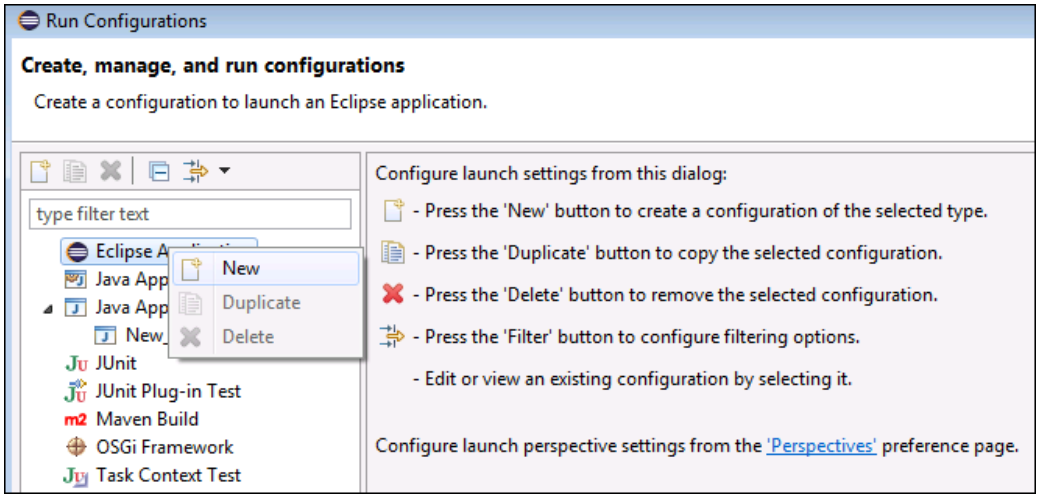


5. Select the **Copy projects into workspace** option, and then click **Finish**. A new project named "MapForceExtension" has been created in your workspace.

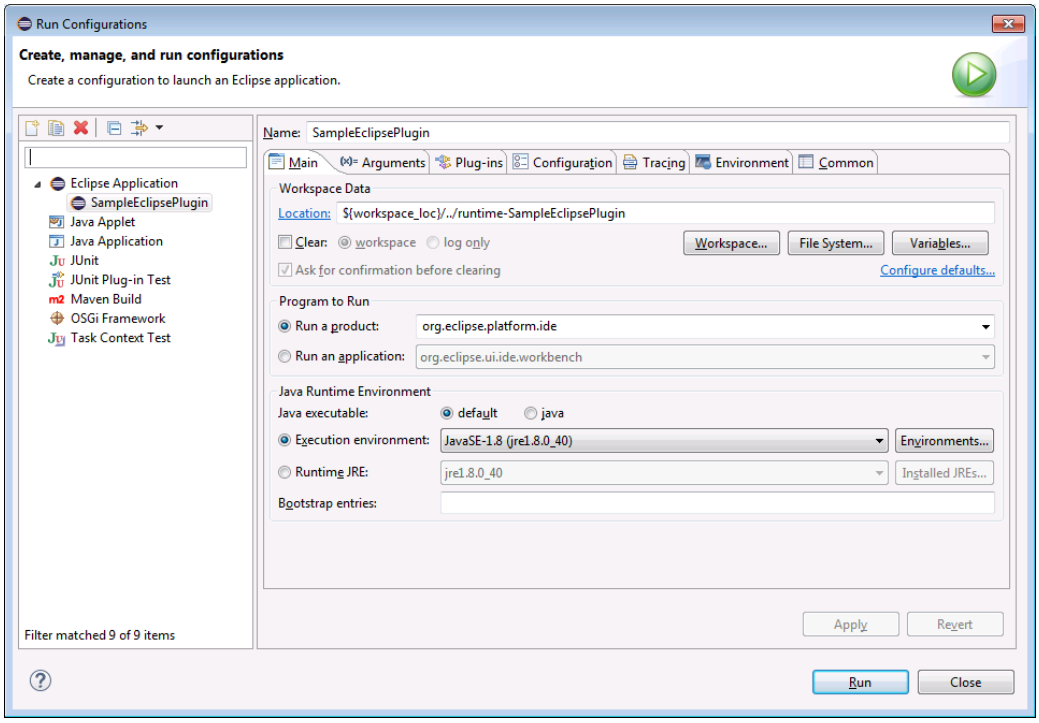
To run the sample extension plug-in:

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

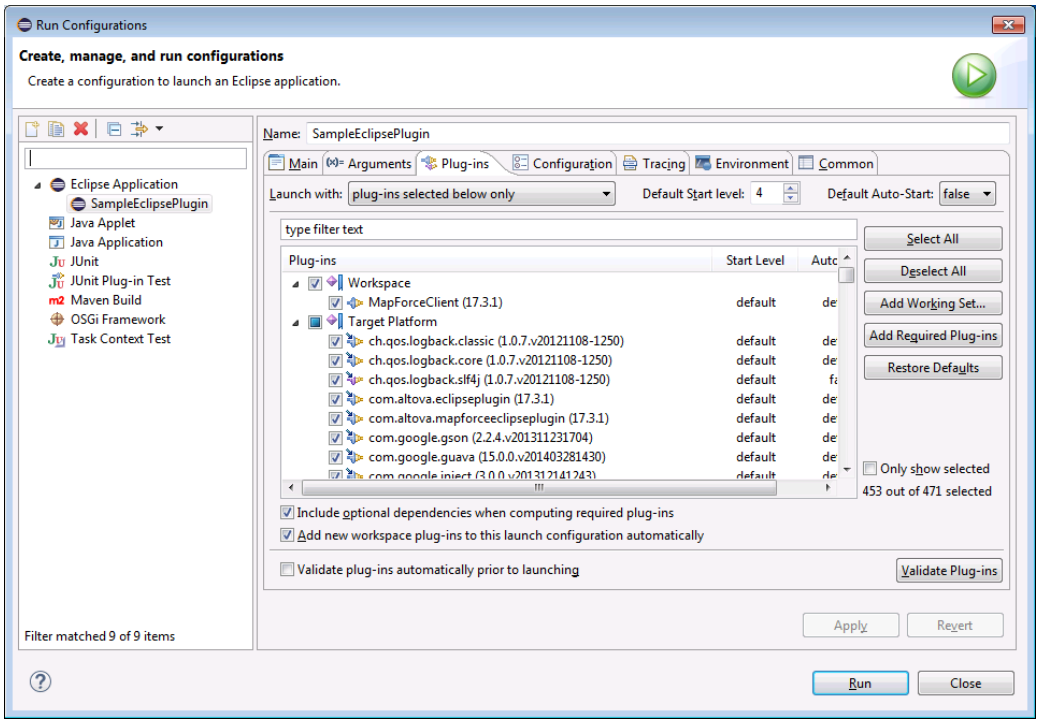
environment. To install Eclipse Plug-in Development Tools, click **Install New Software** in the **Help** menu. and install "Eclipse Plugin Development Tools" from "The Eclipse Project Updates" download site.)



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%.

Chapter 15

Menu Reference

15 Menu Reference

This reference section contains a description of the MapForce menu commands.

15.1 File

New
Creates a new mapping document, or mapping project (.mfp) .

Open
Opens previously saved mapping design (.mfd) , or mapping project (.mfp) files. Note that it is not possible to open mapping files which contain features not available in your MapForce edition.

Save
Saves the currently active mapping using the currently active file name.

Save As
Saves the currently active mapping with a different name, or allows you to supply a new name if this is the first time you save it.

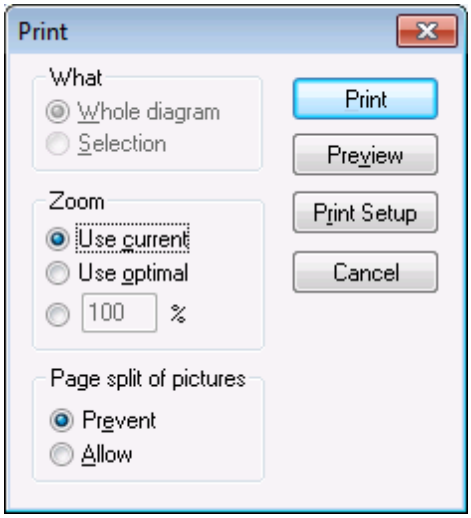
Save All
Saves all currently open mapping files.

Reload
Reloads the currently active mapping file. You are asked if you want to lose your last changes.

Close
Closes the currently active mapping file. You are asked if you want to save the file before it closes.

Close All
Closes all currently open mapping files. You are asked if you want to save any of the unsaved mapping files.

Print
Opens the Print dialog box, from where you can print out your mapping as hard copy.



Print dialog box

Use current retains the currently defined zoom factor of the mapping. **Use optimal** scales the mapping to fit the page size. You can also specify the zoom factor numerically. Component scrollbars are not printed. You can also specify if you want to allow the graphics to be split over several pages or not.

Print Preview
Opens the same Print dialog box with the same settings as described above.

Print Setup
Opens the Print Setup dialog box in which you can define the printer you want to use and the paper settings.

Validate Mapping
Validates that all mappings (connectors) are valid and displays any warnings or errors (see Validating mappings).

Mapping settings
Opens the Mapping Settings dialog box where you can define the document-specific settings (see Changing the mapping settings).

Generate code in selected language
Generates code in the currently selected language of your mapping. The currently selected language is visible as a highlighted programming language icon in the toolbar: XSLT, XSLT 2, XQuery, Java, C#, or C++.

Generate code in | XSLT (XSLT2)
This command generates the XSLT file(s) needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box where you select the location of the XSLT file. The name of the generated XSLT file(s) is defined in the Mapping Settings dialog box (see Changing the mapping settings).

Generate code in | XQuery
This command generates the XQuery file(s) needed for the transformation from the source file(s).

Selecting this option opens the Browse for Folder dialog box where you select the location of the XQuery file. The name of the generated XQuery file(s) is defined in the Mapping Settings dialog box (see Changing the mapping settings).

Generate code in | Java | C# | C++

These commands generate source code for a complete application program needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box, where you select the location of the generated files. The names of the generated application files (as well as the project files: *.csproj C# project file, *.sln solution file, *.vcproj visual C++ project file) are defined in the Mapping Settings dialog box (see Changing the mapping settings).

The file name created by the executed code appears in the **Output XML File** box of the Component settings dialog box if the target is an XML/Schema document.

Compile to MapForce Server Execution File

Generates a file that can be executed by MapForce Server to run the mapping transformation (see Compiling a MapForce mapping).

Deploy to FlowForce Server

Deploys the currently active mapping to the FlowForce Server (see Deploying a MapForce mapping).

Generate documentation

Generates documention of your mapping projects in great detail in various output formats (see Generating and Customizing Mapping Documentation).

Recent files

Displays a list of the most recently opened files.

Exit

Exits the application. You are asked if you want to save any unsaved files.

15.2 Edit

Most of the commands in this menu become active when you view the result of a mapping in the **Output** tab, or preview XSLT code in the XSLT tab.

Undo 

MapForce has an unlimited number of "Undo" steps that you can use to retrace you mapping steps.

Redo 

The redo command allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.

Find 

Allows you to search for specific text in either the XSLT, XSLT2, XQuery or Output tab.

Find Next F3 

Searches for the next occurrence of the same search string.

Find Previous Shift F3 

Searches for the previous occurrence of the same search string.

Cut/Copy/Paste/Delete

The standard windows Edit commands, allow you to cut, copy etc., any components or functions visible in the mapping window.

Select all

Selects all components in the Mapping tab, or the text/code in the XSLT, XSLT2, XQuery or Output tab.

15.3 Insert

XML Schema / File 

Adds to the mapping an XML schema or instance file. If you select an XML file which references a schema, no additional information is required for the mapping. If you select an XML file without a schema reference, you are prompted to generate a matching XML schema automatically (see Generating an XML Schema). If you select an XML schema file, you are prompted to include optionally an XML instance file which supplies the data for preview.

Database 

Adds to the mapping a database as source or target component (see Databases and MapForce).

EDI 

Adds to the mapping an EDI document which can be used as source or target component (see EDI).

Text file 

Adds to the mapping a flat file document, such as CSV or a fixed-length text file. Both types of file can be used as source and target components. Additionally, if you want to process text files with a structure other than CSV or fixed-length, you can use FlexText (see MapForce FlexText).

Web Service Function 

Adds to the mapping a call to a Web service (see Calling Web services).

Excel 2007+ File 

Adds to the mapping a Microsoft Excel 2007+ (.xlsx) file (see Microsoft OOXML Excel 2007+). If you don't have Excel 2007 or later, you can still map to or from Excel 2007+ files. In this case, you cannot preview the result in the Output tab, but you can still save it, by clicking **Save Output File** on the **Output** menu.

XBRL Document 

Adds to the mapping an XBRL instance or taxonomy document (see Adding XBRL Files as Mapping Components).

JSON Schema/File 

Adds to the mapping a JSON schema or file (see Adding JSON Files as Mapping Components).

Insert Input 

When the mapping window displays a mapping, this command adds an input component to the mapping (see Supplying Parameters to the Mapping). When the mapping window displays a user-defined function, this command adds an input component to the user-defined function (see Defining Complex Input Components).

Insert Output 

When the mapping window displays a mapping, this command adds an output component to the

mapping (see Returning String Values from a Mapping). When the mapping window displays a user-defined function, this command adds an output component to the user-defined function (see Defining Complex Output Components).

Constant 

Inserts a constant which supplies fixed data to an input connector. The data is entered into a dialog box when creating the component. You can select the following types of data: String, Number and All other.

Variable 

Inserts an Intermediate Variable which is equivalent to a regular (non-inline) user-defined function. Variables are structural components, without instance files, and are used to simplify the mapping process (see Intermediate variables).

Sort: Nodes/Rows

Inserts a component which allows you to sort nodes (see Sort Nodes/Rows).

Filter: Nodes/Rows 

Inserts a component that uses two input and output parameters: **node/row** and **bool**, and **on-true**, **on-false**. If the Boolean is true, then the value of the node/row parameter is forwarded to the on-true parameter. If the Boolean is false, then the complement value is passed on to the on-false parameter. For more information, see Filtering Data.

SQL-WHERE/ORDER 

Inserts a component which allows you to filter database data conditionally (see SQL WHERE / ORDER Component).

Value-Map 

Inserts a component that transforms an input value to an output value using a lookup table. This is useful when you need to map a set of values to another set of values (for example, month numbers to month names). For more information, see Using Value-Maps.

IF-Else Condition 

Inserts a component of type "If-Else Condition" (see Using If-Else Conditions).

Exception 

The exception component allows you to interrupt a mapping process when a specific condition is met, or define Fault messages when using WSDL mapping projects. Please see MapForce Exceptions and Web service faultsfor more information.

15.4 Project

MapForce supports the Multiple Document Interface and allows you to group your mappings into mapping projects (see Working with Mapping Projects).

Reload Project
Reloads the currently active project and switches to the Project tab.

Close Project
Closes the currently active project.

Save Project
Saves the currently active project.

Add Files to Project
Allows you to add mappings to the current project through the Open dialog box.

Add Active File to Project
Adds the currently active file to the currently open project.

Create Folder
This option adds a new folder to the current project structure, and only becomes active when this is possible. See Managing Project Folders.

Open Mapping
Opens the currently highlighted/selected mapping in the Project tab.

Create Mapping for Operation
Creates a mapping file for the currently selected operation of the WSDL project. The operation name defined in the WSDL file is supplied in the "Save as" dialog box, which is opened automatically.

Add Mapping file for Operation
Allows you to add a previously saved mapping file to the currently active WSDL operation. Select the mapping file from the "Open" dialog box.

Insert Web Service...
Allows you to insert a Web Service based on an existing WSDL file.

Open file in XMLSpy
Opens the selected WSDL file, highlighted in the Project window, in XMLSpy.

Generate code for entire project
Generates project code for the entire project currently visible in the Project window. Code is generated in the currently selected default language for all of the mapping files *.mfd in each of the folders.

Generate code in...
Generates project code in the language you select from the context menu.

Properties
Opens a dialog box where you can define project-wide settings. See Setting the Code Generation Settings.

Recent projects - 1. 2. etc.
Displays a list of the most recently opened projects.

15.5 Component

Change Root Element
Allows you to change the root element of the XML instance document.

Edit Schema Definition in XMLSpy
Selecting this option, having previously clicked an XML-Schema/document, opens the XML Schema file in the Schema view of XMLSpy where you can edit it.

Edit FlexText Configuration
Opens FlextText and enables you to edit a previously created FlexText file.

Add/Remove/Edit Database Objects
Allows you to add, remove, or change the database objects within the database component.

Create mapping to EDI X12 997
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 in the main mapping area for you to send on to the recipient. See X12 997 Acknowledgment for specific details on the specific error segments and what they mean.

Create mapping to EDI X12 999
The X12 999 Implementation Acknowledgment Transaction Set reports HIPAA implementation guide non-compliance, or application errors. Each EDI transaction sent to an organization must be responded to by sending a 999 transaction. See X12 999 - Implementation Guide non-compliance for more information.

Refresh
Reloads the structure of the currently active database component from the database.

Add Duplicate Input Before
Inserts a copy/clone of the selected item before the currently selected item. Duplicate items do not have output icons, you cannot use them as data sources. For an example, see Map Multiple Sources to One Target section in the tutorial. Right clicking a duplicate item also allows you to reposition it using the menu items Move Up/Move Down, depending on where the item is.

Add Duplicate Input After
Inserts a copy/clone of the selected item after the currently selected item. Duplicate items do not have output icons, you cannot use them as data sources. For an example, see the Map Multiple Sources to One Target section in the tutorial. Right clicking a duplicate item also allows you to reposition it using the menu items Move Up/Move Down, depending on where the item is.

Remove Duplicate
Removes a previously defined duplicate item. For an example, see the Map Multiple Sources to One Target section in the tutorial.

Database Table Actions
Allows you to define the actions to be performed with the mapped data on the specific target database table. See Table actions, key settings for more information.

Query Database
Creates a Select statement based on the table/field you clicked in the database component. Clicking a table/field once makes this command active, and the select statement is automatically placed into the Select window.

Align Tree Left
Aligns all the items along the left hand window border.

Align Tree Right
Aligns all the items along the right hand window border. This display is useful when creating mappings to the target schema.

Properties
Opens a dialog box which displays the settings of the currently selected component. See Changing the Component Settings .

15.6 Connection

Auto Connect Matching Children
Activates or deactivates the "Auto Connect Matching Children" option, as well as the icon in the icon bar.

Settings for Connect Matching Children
Opens the Connect Matching Children dialog box in which you define the connection settings (see Connecting matching children).

Connect Matching Children
This command allows you to create multiple connectors for items of the **same name,** in both the source and target schemas. The settings you define in this dialog box are retained, and are applied when connecting two items, if the "**Auto connect child items**" icon  in the title bar is active. Clicking the icon switches between an active and inactive state. For further information, see Connecting matching children.

Target Driven (Standard)
Changes the connector type to Standard mapping. For further information, see Target Driven (Standard) mapping.

Copy-all (Copy Child Items)
Creates connectors for all matching child items, where each of the child connectors are displayed as a subtree of the parent connector (see Copy-all connections).

Source Driven (Mixed Content)
Changes the connector type to *Source Driven (Mixed Content)*. For further information, see Source Driven (Mixed Content) mapping.

Properties
Opens a dialog box in which you can define the specific (mixed content) settings of the current connector. Unavailable options are greyed out. These settings also apply to **complexType** items which do not have any text nodes. For further information, see Connection settings.

15.7 Function

Create User-Defined Function
Creates a new user-defined function (see User-defined functions).

Create User-Defined Function from Selection
Creates a new user-defined function based on the currently selected elements in the mapping window.

Function Settings
Opens the settings dialog box of the currently active user-defined function allowing you to change its settings.

Remove Function
Deletes the currently active user-defined function if you are working in a context which allows this.

Insert Input
When the mapping window displays a mapping, this command adds an input component to the mapping (see Simple Input). When the mapping window displays a user-defined function, this command adds an input component to the user-defined function (see Defining Complex Input Components).

Insert Output
When the mapping window displays a mapping, this command adds an output component to the mapping (see Simple Output). When the mapping window displays a user-defined function, this command adds an output component to the user-defined function (see Defining Complex Output Components).

15.8 Output

The first group of options (**XSLT 1.0, XSLT 2.0, etc.)** allow you to define the target language you want your code to be in.

Validate Output
Validates the output XML file against the referenced schema.

Save generated Output
Saves the currently visible data in the Output tab.

Save all generated outputs
Saves all the generated output files of dynamic mappings. See Processing Multiple Input or Output Files Dynamically for more information.

Regenerate output
Regenerates the current mapping from the Output window.

Run SQL-script
If an SQL script is currently visible in the Output window, the script executes the mapping to the target database, taking the defined table actions into account.

Insert/Remove Bookmark
Inserts a bookmark at the cursor position in the Output window.

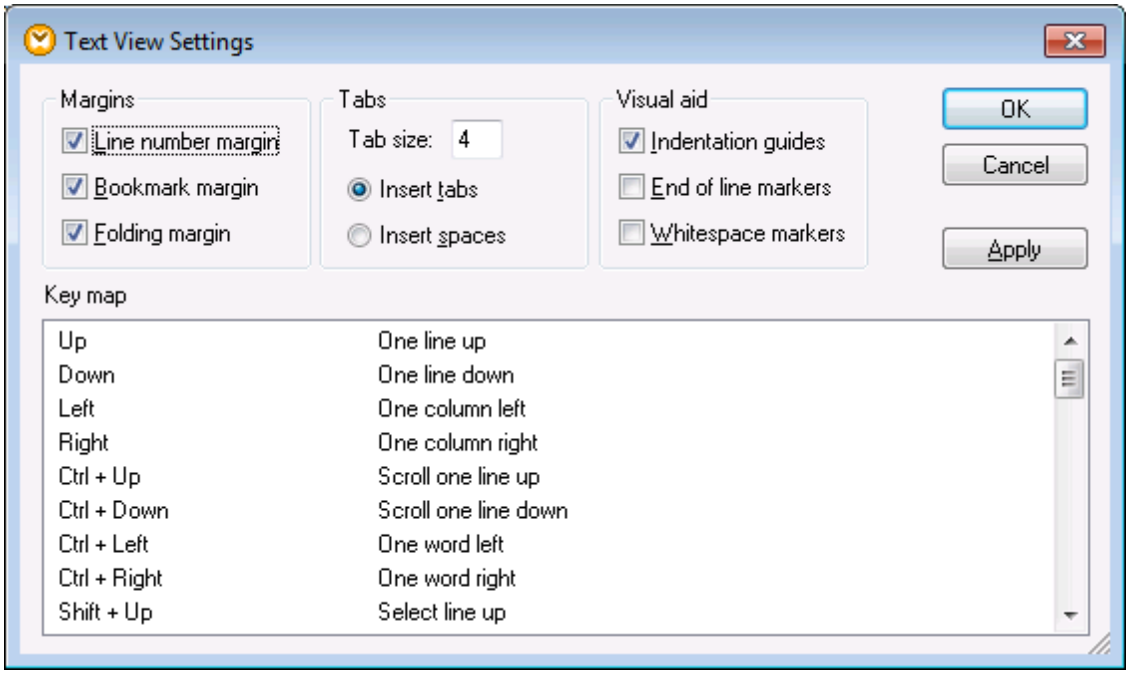
Next Bookmark
Navigates to the next bookmark in the Output window.

Previous Bookmark
Navigates to the previous bookmark in the Output window.

Remove All Bookmarks
Removes all currently defined bookmarks in the Output window.

Pretty-Print XML Text
Reformats your XML document in the Output pane to give a structured display of the document. Each child node is offset from its parent by a single tab character. This is where the Tab size settings (i.e. inserting as tabs or spaces) defined in the Tabs group, take effect.

Text View Settings
Allows you to customize the text settings in the Output window and also shows the currently defined hotkeys that apply in the window.



Note: The **Insert tabs** and **Insert spaces** options do not affect the currently visible text in the Output window; they only take effect when you use the **Output | Pretty-Print XML text** option.

15.9 Debug

Start Debugging (F11)

Starts or continues debugging until a breakpoint is hit or the mapping finishes.

Stop Debugging (Shift + F5)

Stops debugging. This command exits the debug mode and switches MapForce back to standard mode.

Step into (F11)

Executes the mapping until a single step is finished anywhere in the mapping. In the mapping debugger, a step is a logical group of dependent computations which normally produce a single item of a sequence.

Depending on the mapping context, this command roughly translates into "go to the left/go to target child/go to source parent".

Step over (F10)

Continues execution until the current step finishes (or finishes again for another item of the sequence), or an unrelated step finishes. This command steps over computations that are inputs of the current step.

Step out (Shift + F11)

Continues execution until the result of the current step is consumed or a step is executed that is not an input or child of the consumption. This command steps out of the current computation.

Depending on the mapping context, this command roughly translates into "go to the right/go to target parent/go to source child".

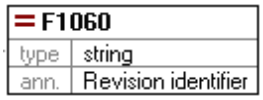
Minimal step (Ctrl + F11)

Continues execution until a value is produced or consumed. This command subdivides a step and will typically stop twice for each connection: once when its source produces a value and once when its target consumes it. MapForce does not necessarily compute values in the order the mapping would suggest, so production and consumption events do not always follow each other.

15.10 View

Show Annotations 

Displays XML schema annotations (as well as EDI info) in the component window.
If the Show Types icon is also active, then both sets of info are show in grid form.



Show Types 

Displays the schema datatypes for each element or attribute.
If the Show Annotations icon is also active, then both sets of info are show in grid form.

Show library in Function Header 

Displays the library name in parenthesis in the function title.

Show Tips 

Displays a tooltip containing explanatory text when the mouse pointer is placed over a function.

XBRL Display Options

See Configuring the XBRL Label Display Options.

Show Selected Component Connectors 

Switches between showing all mapping connectors, or those connectors relating to the currently selected components.

Show Connectors from Source to Target 

Switches between showing:

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

Zoom

Opens the Zoom dialog box. You can enter the zoom factor numerically, or drag the slider to change the zoom factor interactively.

Back

Steps back through the currently open mappings of the mapping tab.

Forward

Steps forward through the currently open mappings of the mapping tab.

Status Bar

Switches on/off the Status Bar visible below the Messages window.

Library Window

Switches on/off the Library window.

Messages
Switches on/off the Validation output window. When generating code the Messages output window is automatically activated to show the validation result.

Overview
Switches on/off the Overview window. Drag the rectangle to navigate your Mapping view.

Project window
Switches on/off the Project window.

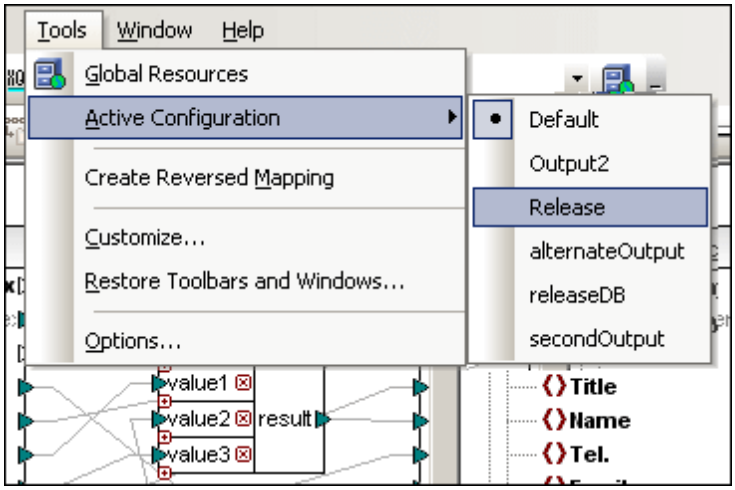
15.11 Tools

Global Resources

Opens the Manage Global Resources dialog box allowing you to Add, Edit and Delete global resources to the Global Resources XML file, please see Global Resources - Properties for more information.

Active Configuration

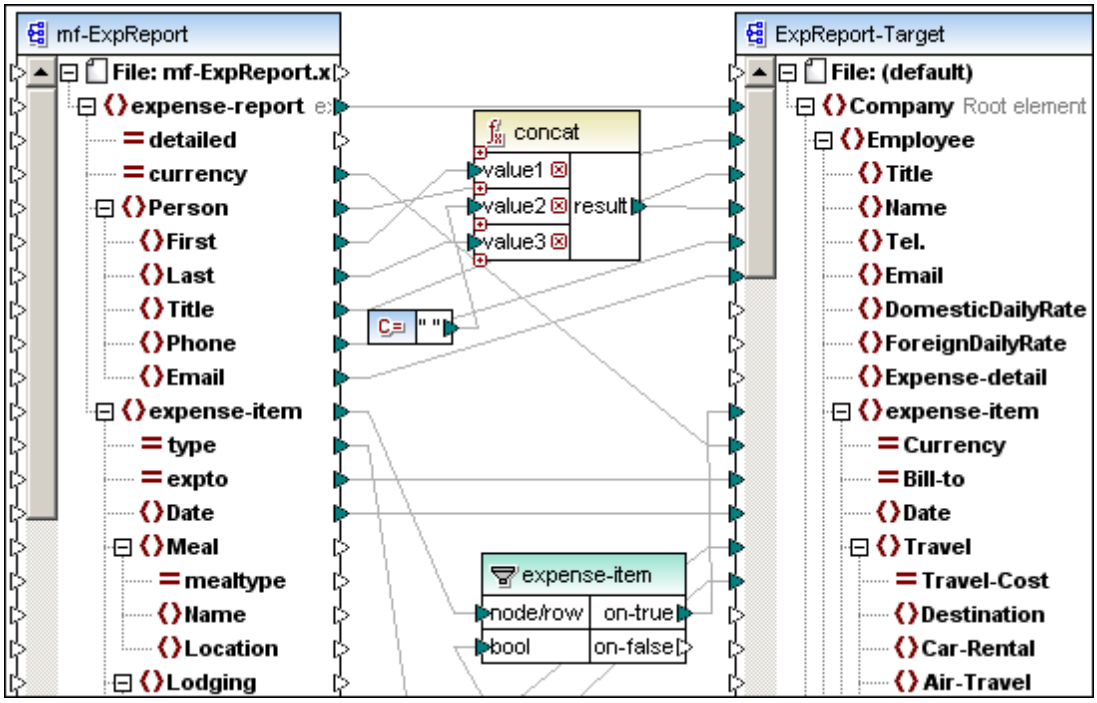
Allows you to select/change the currently active global resource from a list of all resources/ configurations in the Global Resources. Select the required configuration from the submenu.



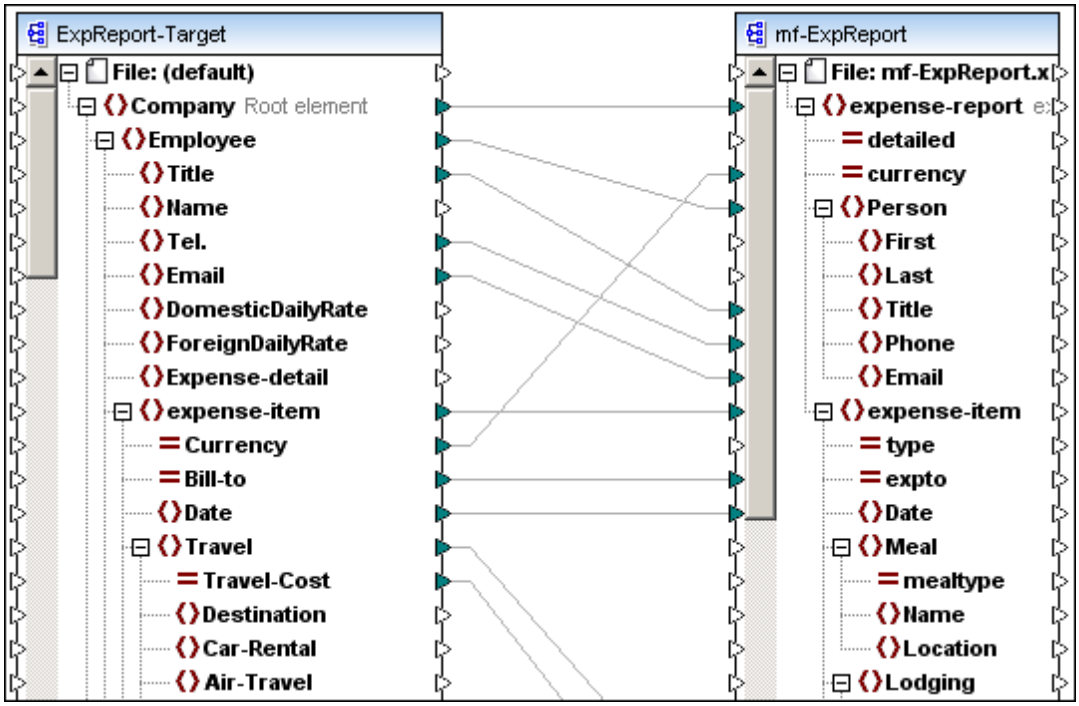
Create Reversed Mapping

Creates a "reversed" mapping from the currently active mapping in MapForce, which is to be the basis of a new mapping. Note that the result is not intended to be a complete mapping, only the direct connections between components are retained in the reversed mapping. It is very likely that the resulting mapping will not be valid, or be able to be executed when clicking the Output tab, without manual editing.

E.g. **Tut-ExpReport.mfd** in the ...\MapForceExamples\Tutorial folder:



Result of a reversed mapping:



General:

- The source component becomes the target component, and target component becomes the source.
- If an Input, and Output XML, instance file have been assigned to a component, then they will both be swapped.

Retained connections

- Direct connections between components
- Direct connections between components in a chained mapping
- The type of connection: Standard, Mixed content, Copy-All

- Pass-through component settings
- Database components are unchanged.

Deleted connections

- Connections via functions, filters etc. are deleted, along with the functions etc.
- User-defined functions
- Webservice components

Restore Toolbars and Windows

Resets the toolbars, entry helper windows, docked windows etc. to their defaults. MapForce needs to be restarted for the changes to take effect.

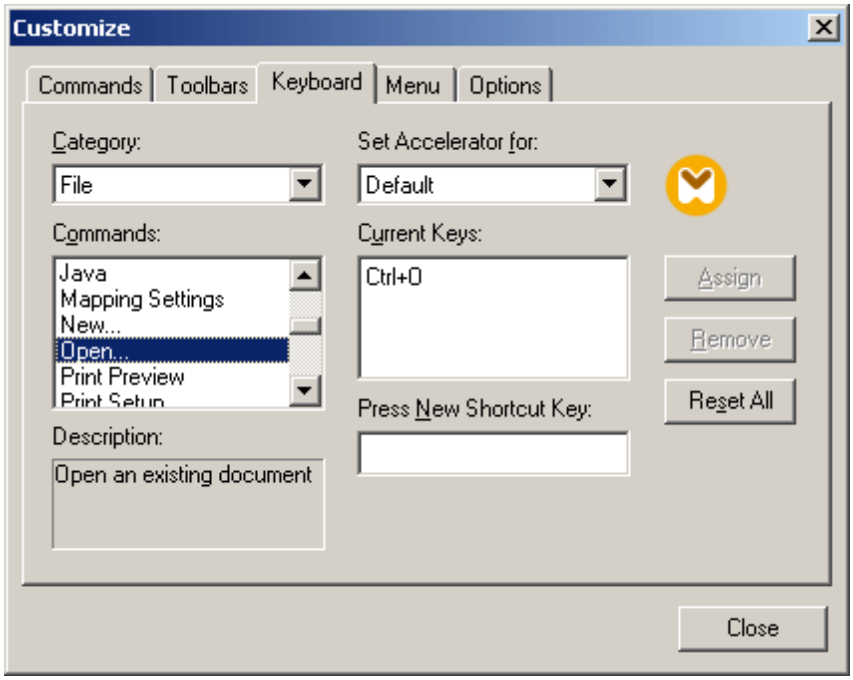
Customize...

The customize command lets you customize MapForce to suit your personal needs.

The **Keyboard** tab allows you to define (or change) keyboard shortcuts for any MapForce command.

To assign a new Shortcut to a command:

1. Select the **Tools | Customize** command and click the Keyboard tab.
2. Click the **Category** combo box to select the menu name.
3. Select the **command** you want to assign a new shortcut to, in the Commands list box
4. Click in the **Press New Shortcut Key**: text box, and press the shortcut keys that are to activate the command.



The shortcuts appear immediately in the text box. If the shortcut was assigned previously, then that function is displayed below the text box.

5. Click the **Assign** button to assign the shortcut.
   The shortcut now appears in the Current Keys list box.
   (To **clear** the entry in the Press New Shrotcut Key text box, press any of the control keys, **CTRL**, **ALT** or **SHIFT**).

To de-assign or delete a shortcut:

1. Click the shortcut you want to delete in the Current Keys list box.
2. Click the **Remove** button.
3. Click the **Close** button to confirm.

Set accelerator for:

Currently no function.

Currently assigned keyboard shortcuts:

Hotkeys by key

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

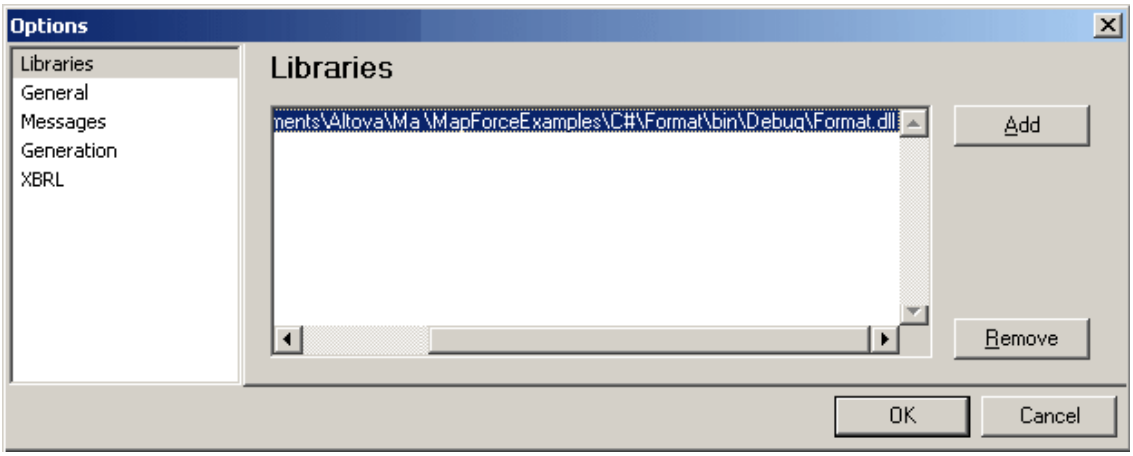
Options

Opens the Options dialog box through which you can:

- Add or delete user defined XSLT functions, or custom libraries.
- Define **general** settings, such as the default character encoding for new components, in the General tab.
- Define which message notifications you want to re-enable
- Define your specific compiler and IDE settings.
- Define the specific display settings of XBRL components

Libraries tab:

- Add or delete user-defined XSLT, or programming language Libraries/functions to MapForce.

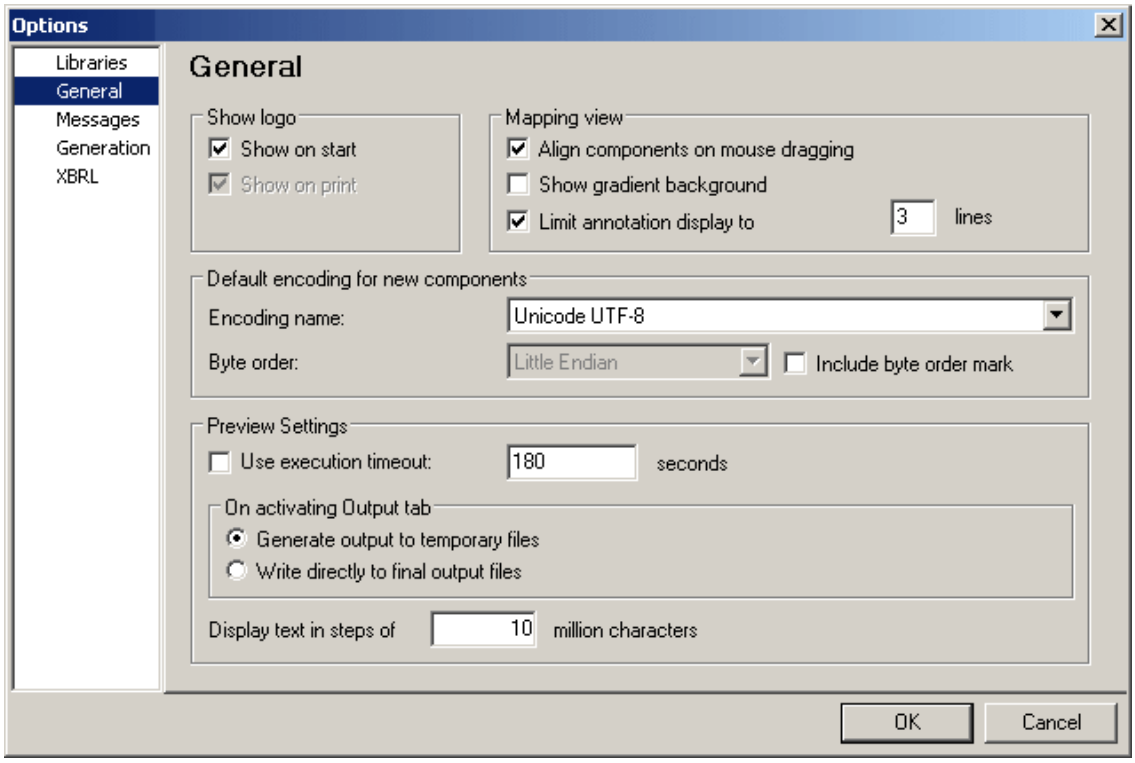


General tab:

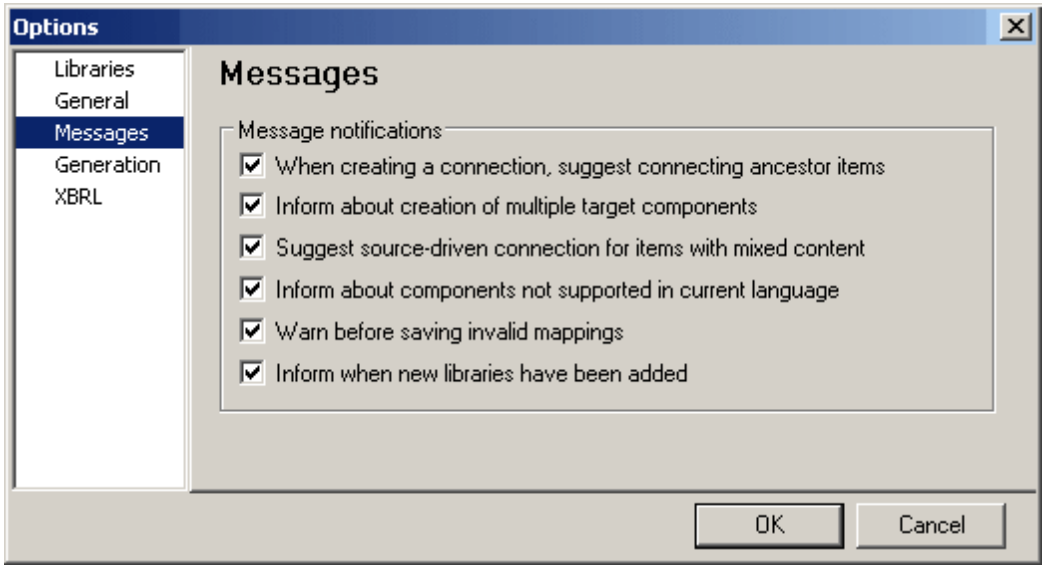
- Specify if you want to show the logo, copyright etc., on start and/or when printing.
- Align components or functions with other components, while dragging them with the mouse.
- Enable/disable the MapForce gradient background.
- Limit the annotation text in components to X lines. Also applies to SELECT statements visible in a component.
- Define the default character encoding for new components.
- Define an execution timeout for the Output tab when previewing the mapping result.
- Specify if you want to output to **temporary** files (default), or write output files **directly** to disk when clicking the Output button/tab.

  Warning: Enabling "Write directly to final output files" will overwrite output files without requesting further confirmation.

- Limit the output to X million characters, when outputting to the built-in execution engine. The Built-in execution engine is the only target that supports XML, CSV, and FLF streaming



Messages tab:
Allows you to re-enable message boxes that you previously disabled using the "Don't ask me again" check box.



Generation tab:
Allows you to define the IDE and compiler settings applicable when you generate program code (see Code generator options).

XBRL tab:
Allows you to define specific display settings of XBRL components (see General XBRL Options)

15.12 Window

Cascade

This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

Tile Horizontal

This command rearranges all open document windows as **horizontal tiles**, making them all visible at the same time.

Tile Vertical

This command rearranges all open document windows as **vertical tiles**, making them all visible at the same time.

1
2
This list shows all currently open windows, and lets you quickly switch between them. You can also use the Ctrl-TAB or CTRL F6 keyboard shortcuts to cycle through the open windows.

15.13 Help Menu

The **Help** menu contains commands to access the onscreen help manual for MapForce, commands to provide information about MapForce, and links to support pages on the Altova web site. The Help menu also contains the Registration dialog, which lets you enter your license key-code once you have purchased the product.

The description of the Help menu commands is organized into the following sub-sections:

- Table of Contents, Index, Search
- Registration, Order Form
- Other Commands

15.13.1 Table of Contents, Index, Search

The **Table of Contents** command opens the onscreen help manual for MapForce with the Table of Contents displayed in the left-hand-side pane of the Help window. The Table of Contents provides a good overview of the entire Help document. Clicking an entry in the Table of Contents takes you to that topic.

The **Index** command opens the onscreen help manual for MapForce with the Keyword Index displayed in the left-hand-side pane of the Help window. The index lists keywords and lets you navigate to a topic by double-clicking the keyword. If a keyword is linked to more than one topic, you are presented with a list of the topics to choose from.

The **Search** command opens the onscreen help manual for MapForce with the Search dialog displayed in the left-hand-side pane of the Help window. To search for a term, enter the term in the input field, and press Return. The Help system performs a full-text search on the entire Help documentation and returns a list of hits. Double-click any item to display that item.

15.13.2 Activation, Order Form, Registration, Updates

Software Activation

After you download your Altova product software, you can activate it using either a free evaluation key or a purchased permanent license key.

- ***Free evaluation key.*** When you first start the software after downloading and installing it, the Software Activation dialog will pop up. In it is a button to request a free evaluation key-code. Enter your name, company, and e-mail address in the dialog that appears, and click Request Now! The evaluation key is sent to the e-mail address you entered and should reach you in a few minutes. Now enter the key in the key-code field of the Software Activation dialog box and click **OK** to start working with your Altova product. The software will be unlocked for a period of 30 days.
- ***Permanent license key.*** The Software Activation dialog contains a button to purchase a permanent license key. Clicking this button takes you to Altova's online shop, where you can purchase a permanent license key for your product. There are two types of permanent license: single-user and multi-user. Both will be sent to you by e-mail. A *single-user license* contains your license-data and includes your name, company, e-mail, and key-code.A *multi-user license* contains your license-data and includes your company name and key-code. Note that your license agreement does not allow you to install more

than the licensed number of copies of your Altova software on the computers in your organization (per-seat license). Please make sure that you enter the data required in the registration dialog exactly as given in your license e-mail.

Note: When you enter your license information in the Software Activation dialog, ensure that you enter the data exactly as given in your license e-mail. For multi-user licenses, each user should enter his or her own name in the Name field.

The Software Activation dialog can be accessed at any time by clicking the **Help | Software Activation** command.

Order Form

When you are ready to order a licensed version of MapForce, you can use either the **Order license key** button in the Software Activation dialog (*see previous section*) or the **Help | Order Form** command to proceed to the secure Altova Online Shop.

Registration

The first time you start your Altova software after having activated it, a dialog appears asking whether you would like to register your product. There are three buttons in this dialog:

- **OK:** Takes you to the Registration Form
- **Remind Me Later:** Pops up a dialog in which you can select when you wish to be next reminded.
- **Cancel:** Closes the dialog and suppresses it in future. If you wish to register at a later time, you can use the **Help | Registration** command.

Check for Updates

Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly

15.13.3 Other Commands

The **Support Center** command is 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.

The **FAQ on the Web** command is a link to Altova's FAQ database on the Internet. The FAQ database is constantly updated as Altova support staff encounter new issues raised by customers.

The **Components Download** command is a link to Altova's Component Download Center on the Internet. From here you will be able to 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.

The **MapForce on the Internet** command is a link to the Altova website on the Internet. You can learn more about MapForce and related technologies and products at the Altova website.

The **MapForce Training** command is a link to the Online Training page at the Altova website.

Here you can select from online courses conducted by Altova's expert trainers.

The **About MapForce** command displays the splash window and version number of your product.

Chapter 16

Code Generator

16 Code Generator

Code Generator is a MapForce built-in feature which enables you to generate Java, C++ or C# code from mapping files designed with MapForce. You can generate code not only from simple mappings with a single data source and target, but also from mappings with multiple sources and multiple targets. The result is a fully-featured and complete application which performs the mapping operation for you. Once you generate the code, you can execute the mapping by running the application directly as generated. You can also import the generated code into your own application, or extend it with your own functionality.

16.1 Introduction to code generator

In the case of XML Schemas, the MapForce code generator's default templates automatically generate class definitions 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, as well as all necessary classes which perform the mapping. 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.

The generated code also includes functions which support the following operations:

- Read XML files into a Document Object Model (DOM) in-memory representation
- Write XML files from a DOM representation back to a system file
- Convert strings to XML DOM trees and vice versa.

The output program code is expressed in C++, Java or C# programming languages.

Target Language	**C++**	**C#**	**Java**
Development environments	Microsoft Visual Studio 2010 Microsoft Visual Studio 2008 Microsoft Visual Studio 2005	Microsoft Visual Studio 2010 Microsoft Visual Studio 2008 Microsoft Visual Studio 2005	Eclipse 3.4 or later Apache Ant (build.xml file)
XML DOM implementations	MSXML 6.0 or Apache Xerces 2.6 or later	System.Xml	JAXP
Database API	ADO	ADO.NET	JDBC

C++

You can configure whether the C++ generated output should use MSXML 6.0 or Apache Xerces 2.6 or later. MapForce generates complete project (.vcproj) and solution (.sln) files for Visual Studio 2005/2008/2010. The generated code optionally supports MFC.

Note: When building C++ code for Visual Studio 2005/2008/2010 and using a Xerces library precompiled for Visual C++, a compiler setting has to be changed in all projects of the solution:

1. Select all projects in the Solution Explorer.
2. On the **Project** menu, click **Properties**.
3. Click **Configuration Properties | C/C++ | Language**.
4. In the list of configurations, select *All Configurations*.
5. Change **Treat wchar_t as Built-in Type** to **No (/Zc:wchar_t-)**

C#

The generated C# code uses the .NET XML classes (System.Xml) and can be used from any .NET capable programming language, such as VB.NET, Managed C++, or J#. Project files can be generated for Visual Studio 2005, 2008, and 2010.

Java

The generated Java output is written against the industry-standard Java API for XML Parsing (JAXP) and includes an Ant build file and project files for Eclipse. Java 7 or higher is supported.

Generated output

The designated destination folder will include all the libraries and files required to execute the mapping, namely:

- A variable number of Altova libraries required by the mapping (for example, Altova function libraries, database libraries, EDI libraries)
- A complete mapping application. When compiled and run, the application performs the mapping transformation.

Code generator templates

Output code is completely customizable via a simple yet powerful template language (SPL, from Spy Programming Language) which gives full control in mapping XML Schema built-in data-types to the primitive datatypes of a particular programming language. SPL allows you to easily replace the underlying parsing and validating engine, customize code according to your company's writing conventions, or use different base libraries such as the Microsoft Foundation Classes (MFC) and the Standard Template Library (STL).

16.2 What's new ...

Version 2014

- Removal of compatibility mode option for code generation

Version 2011

- Contains bug fixes and enhancements.

Version 2010 R3

- Support for Microsoft Visual Studio 2010
- Support for MSXML 6.0 in generated C++ code
- Support for 64-bit targets for C++ and C# projects

Version 2010

- Enumeration facets from XML schemas are now available as symbolic constants in the generated classes (using 2007r3 templates).

Version 2009 sp1

- Apache Xerces version 3.x support added. (older versions starting from Xerces 2.6.x are still supported.)

Version 2009

- The generated mapping implementation was redesigned to support sequences and grouping. The API has not changed.

Version 2008 R2

- Support for generation of Visual Studio 2008 project files for C# and C++ has been added.
- Generated MapForce mapping code in C# and Java can use readers/writers, streams, strings or DOM documents as sources and targets.

Version 2008

The new 2007 R3-style SPL templates have been further enhanced:

- It is now possible to remove single elements
- Access to schema metadata (e.g. element names, facets, enumerations, occurrence, etc.) is provided

- Complex types derived by extension are now generated as derived classes

Version 2007 R3

Code Generator has been redesigned for version 2007 release 3 to simplify usage of the generated code, reduce code volume and increase performance.

- Handling of XML documents and nodes with explicit ownership, to avoid memory leaks and to enable multi-threading
- New syntax to avoid name collisions
- New data types for simpler usage and higher performance (native types where possible, new null handling, ...)
- Attributes are no longer generated as collections
- Simple element content is now also treated like a special attribute, for consistency
- New internal object model (important for customized SPL templates)
- Compatibility mode to generate code in the style of older releases
- Type wrapper classes are now only generated on demand for smaller code

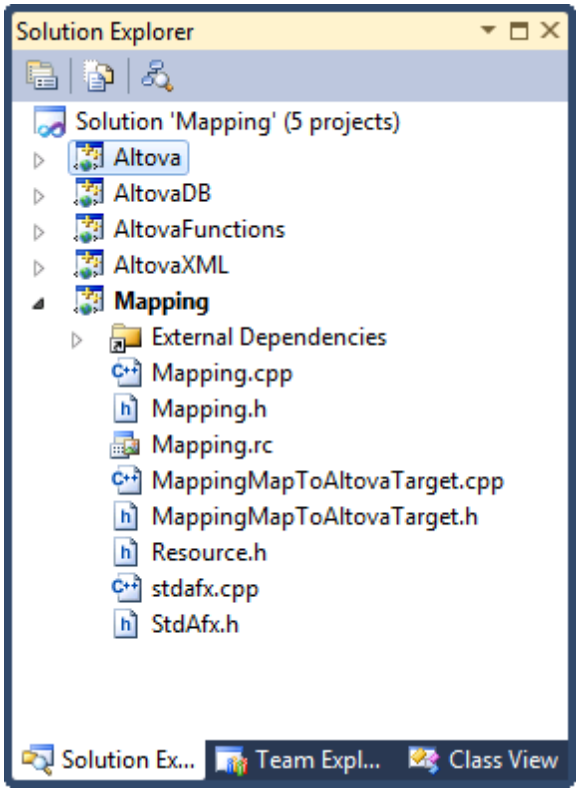
16.3 Generating C++ code

You can generate C++ code for Visual Studio 2005, 2008 and 2010. The generated code includes **.sln** and .**vcproj** files for Visual Studio. Note the following when generating code:

- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the **Tools | Options** menu, **Generation** tab.
- You can change the name of the generated mapping application and other settings from the **File | Mapping settings** menu. The default application name is **Mapping**.
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option **Component | Properties**.

A typical C++ solution generated by MapForce includes the following:

- Several Altova-signed libraries required by the mapping (all prefixed with **Altova**).
- The main mapping project (in this example, **Mapping**), which includes the mapping application and dependent files.



Sample C++ solution generated with MapForce

This section includes the following topics:

- Generating code from a mapping
- Generating code from a mapping project
- Building the project
- Running the application

16.3.1 Generating code from a mapping

To generate C++ code from a mapping design file (.mfd):

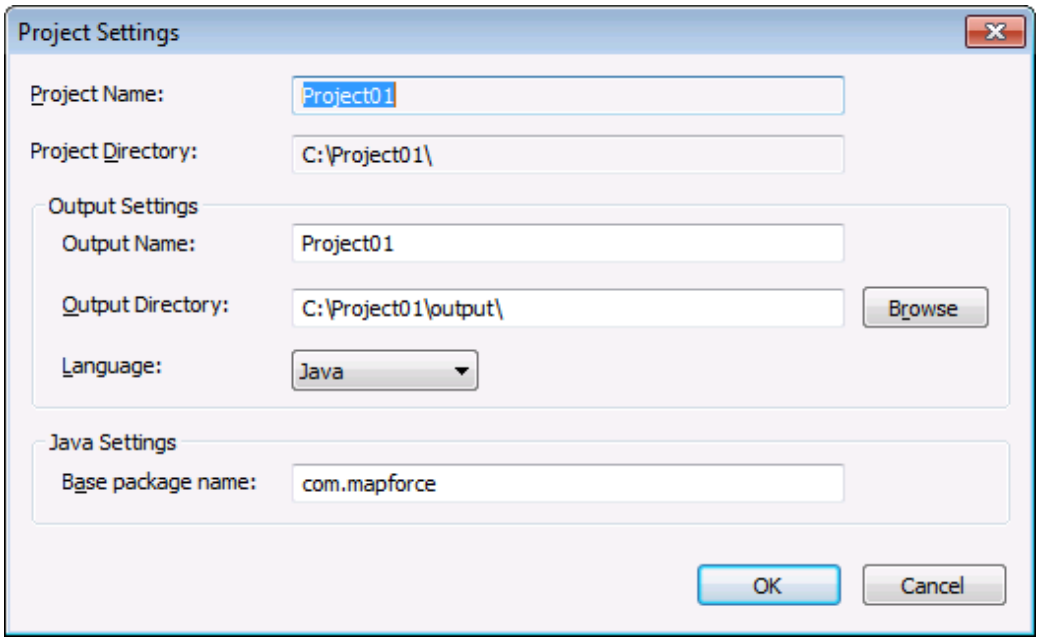
1. Review and select the desired code generation options (see Code generator options).
2. On the **File** menu, click **Generate code in | C++**.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed in the Messages window.

The default name of the generated application is **Mapping**. If required, you can change this, and other settings, from the Mapping Settings dialog box (see Changing the mapping settings).

16.3.2 Generating code from a mapping project

To generate code from a mapping project (.mfp):

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click **Properties**.



3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click **OK**.
4. On the **Project** menu, click **Generate code for the Entire Project**.

The progress and result of the code generation process (error or success message) is displayed in the Messages window.

By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the **output** sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select **Project | Properties** from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select **Properties**). Otherwise, all project folders inherit the settings from the MapForce project.

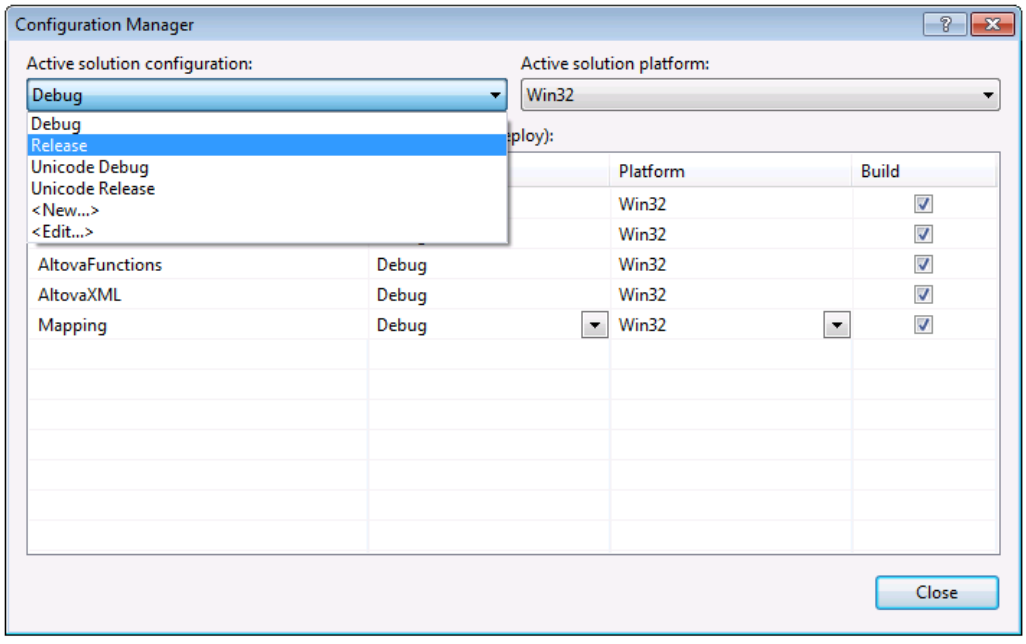
16.3.3 Building the project

Once you generated the C++ code, building it in Visual Studio is the next step. To build the generated code:

1. Open the generated solution (.sln) file in Visual Studio.

By default, the name of the solution file is **Mapping.sln**, and it is located in the **Mapping** subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to **MyApplication**, then the solution file is called **MyApplication.sln**, and it is located in the **MyApplication** subdrectory.

2. On the **Build** menu, click **Configuration Manager**.



3. Select the required build configuration (Debug, Release, Unicode Debug, Unicode Release). Note that only Unicode builds support the full Unicode character set in XML and

other files. The non-Unicode builds work with the local codepage of your Windows installation.

4. On the **Build** menu, click **Build Solution**.

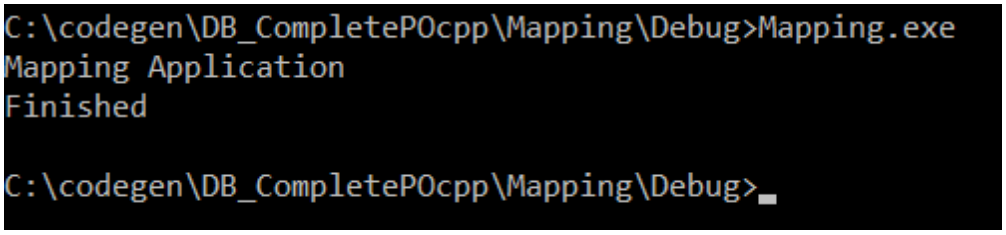
16.3.4 Running the application

Once you compile the Visual Studio project, a command-line application is produced, called **Mapping.exe**. (Note that if you changed the application name from the mapping settings, then the executable name is changed accordingly.)

You can locate the mapping application in one of the following subdirectories relative to the .sln file, depending on the build option you chose:

- Debug
- Release
- Unicode Debug
- Unicode Release

To run the application, open a command prompt, change the current directory to the path of the executable, and run it, for example:



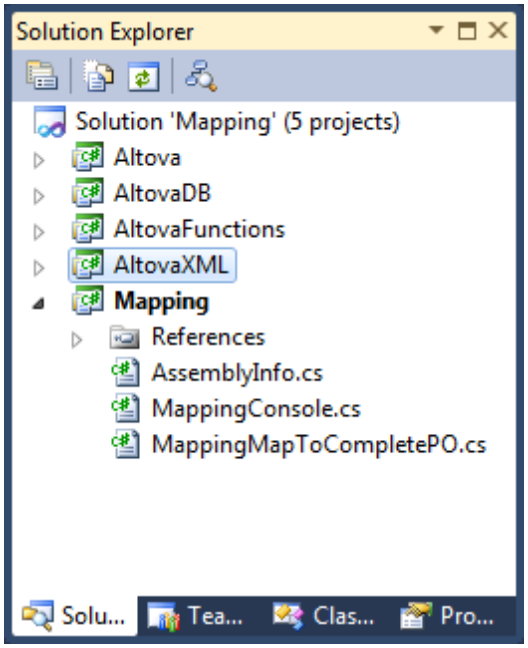
16.4 Generating C# code

You can generate C# code for Visual Studio 2005, 2008 and 2010. The generated code includes **.sln** and .**csproj** files for Visual Studio. Mono users can use Visual Studio solution files with Mono's xbuild. Note the following when generating code:

- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the **Tools | Options** menu, **Generation** tab.
- You can change the name of the generated mapping application and other settings from the **File | Mapping settings** menu. The default application name is **Mapping**.
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option **Component | Properties**.

A typical C# solution generated by MapForce includes the following:

- Several Altova-signed libraries required by the mapping (all prefixed with **Altova**).
- The main mapping project (in this example, **Mapping**), which includes the mapping application and dependent files.



Sample C# solution generated with MapForce

This section includes the following topics:

- Generating code from a mapping

- Generating code from a mapping project
- Building the project
- Running the application

16.4.1 Generating code from a mapping

To generate C# code from a mapping design file (.mfd):

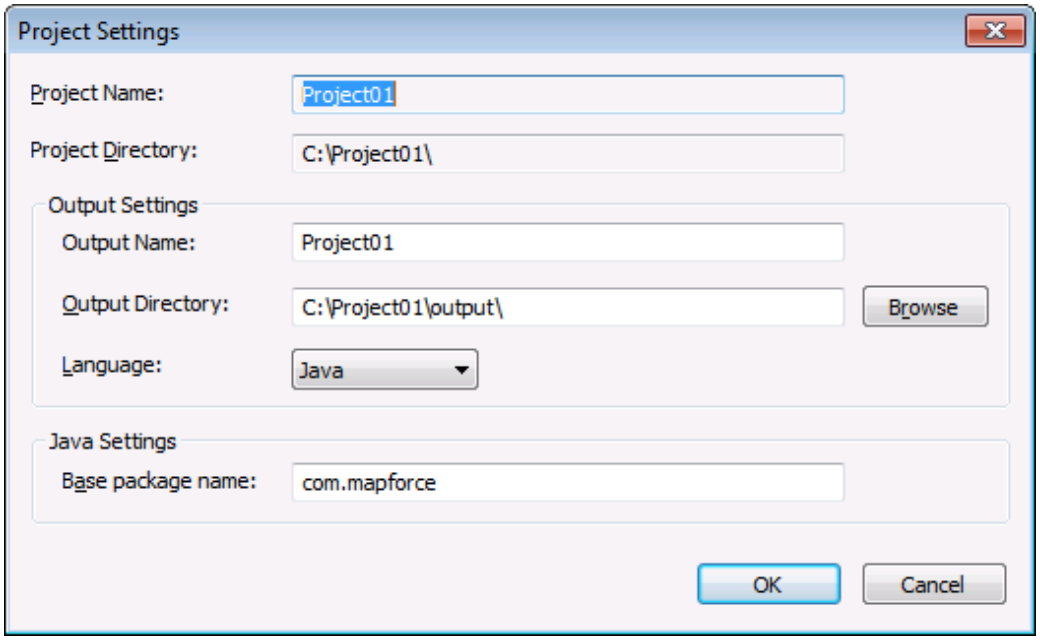
1. Review and select the desired code generation options (see Code generator options).
2. On the **File** menu, click **Generate code in | C#**.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed in the Messages window.

The default name of the generated application is **Mapping**. If required, you can change this, and other settings, from the Mapping Settings dialog box (see Changing the mapping settings).

16.4.2 Generating code from a mapping project

To generate code from a mapping project (.mfp):

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click **Properties**.



3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click **OK**.
4. On the **Project** menu, click **Generate code for the Entire Project**.

The progress and result of the code generation process (error or success message) is displayed in the Messages window.

By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the **output** sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select **Project | Properties** from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select **Properties**). Otherwise, all project folders inherit the settings from the MapForce project.

16.4.3 Building the project

Once you generated the C# code, building it in Visual Studio is the next step. To build the generated code:

1. Open the generated solution (.sln) file in Visual Studio.

By default, the name of the solution file is **Mapping.sln**, and it is located in the **Mapping** subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to **MyApplication**, then the solution file is called **MyApplication.sln**, and it is located in the **MyApplication** subdrectory.

2. On the **Build** menu, click **Configuration Manager**.
3. Select the required build configuration (Debug, Release).
4. On the **Build** menu, click **Build Solution**.

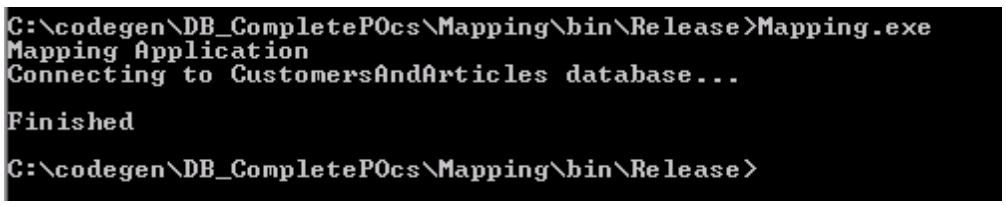
16.4.4 Running the application

Once you compile the Visual Studio project, a command-line application is produced, called **Mapping.exe**. (Note that if you changed the application name from the mapping settings, then the executable name is changed accordingly.)

You can locate the mapping application in one of the following subdirectories relative to the .sln file, depending on the build option you chose:

- bin\Debug
- bin\Release

To run the application, open a command prompt, change the current directory to the path of the executable, and run it, for example:



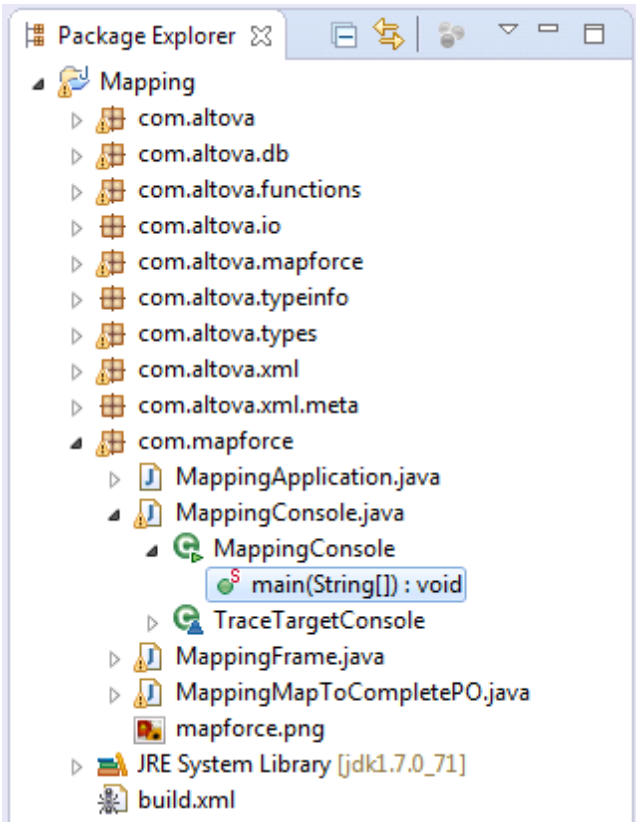
16.5 Generating Java code

You can generate program code for Java version 7 or later. Note the following when generating code:

- You can generate code either from a single mapping design (.mfd), or from a mapping project (.mfp). If you generate code from a single mapping, the resulting application executes the respective mapping transformation. If you generate code from a MapForce project (.mfp) which includes multiple mappings, the resulting application executes in bulk all mappings included in the project.
- You can change the general code generation options from the **Tools | Options** menu, **Generation** tab.
- You can change the name of the generated mapping application and other settings from the **File | Mapping settings** menu. The default application name is **Mapping**.
- If your mapping contains database components, you can view database specific settings by clicking a database component, and then selecting the menu option **Component | Properties**.

A typical Java project generated by MapForce includes the following:

- Several Altova-signed Java packages required by the mapping (all prefixed with **com.altova**).
- The **com.mapforce** package, which includes the mapping application and dependent files (as shown below, it is possible to change the name of this package). The two most important files in this package are as follows:
  - The Java mapping application as a dialog application (**MappingApplication.java**).
  - The Java mapping application as a console application (**MappingConsole.java**).
- A **build.xml** file which you can execute with Apache Ant to compile the project and generate JAR files.



Sample MapForce-generated Java application (Eclipse IDE)

This section includes the following topics:

- Generating code from a mapping
- Generating code from a mapping project
- Handling JDBC references
- Building the project with Ant
- Example: Run and compile Java code with Eclipse and Ant

16.5.1 Generating code from a mapping

To generate Java code from a mapping design file (.mfd):

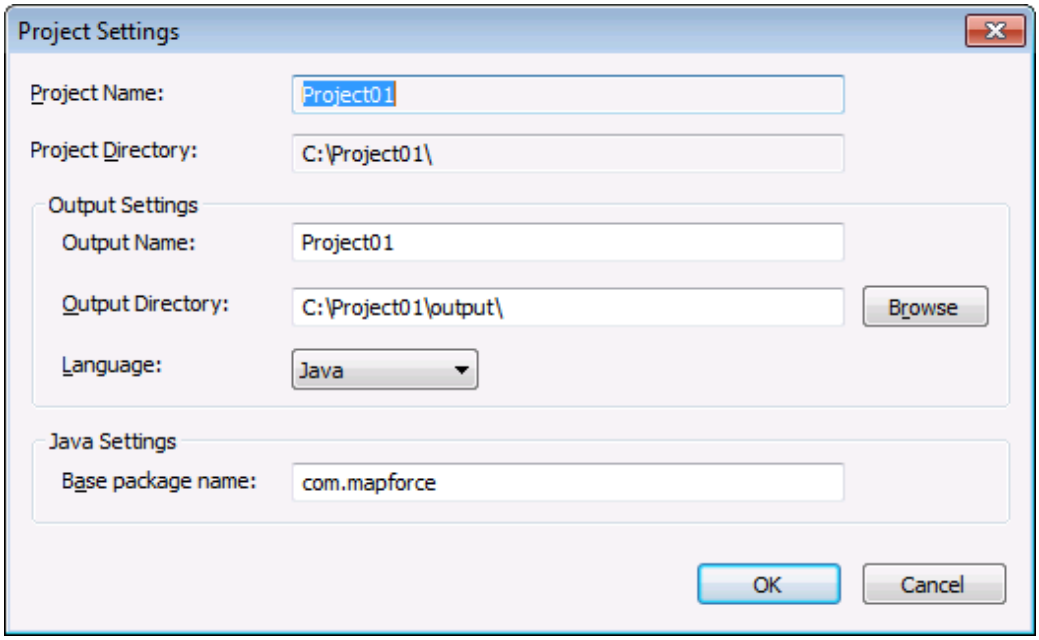
1. Review and select the desired code generation options (see Code generator options).
2. On the **File** menu, click **Generate code in | Java**.
3. Select a destination directory for the generated files, and then click OK to confirm. The result of code generation (error or success message) is displayed in the Messages window.

The default name of the generated application is **Mapping**, and the default name of the base package is **com.mapforce**. If required, you can change these from the Mapping Settings dialog box (see Changing the mapping settings).

16.5.2 Generating code from a mapping project

To generate code from a mapping project (.mfp):

1. If you haven't done so already, open the mapping project in MapForce.
2. Right-click the project in the Project window, and then click **Properties**.



3. Review and change the project settings if required (in particular, ensure that the target language and the output directory are set correctly), and then click **OK**.
4. On the **Project** menu, click **Generate code for the Entire Project**.

The progress and result of the code generation process (error or success message) is displayed in the Messages window.

By default, the name of the generated application is the same as the project name. If the project name contains spaces, these are converted to underscores in the generated code. By default, code is generated in the same directory as the MapForce project, in the **output** sub-directory.

To change the output directory and the name of the project, click the Project in the Project window, and then select **Project | Properties** from the menu. If your MapForce project contains folders, you can change the code generation settings for each individual folder (right click on the folder, and then select **Properties**). Otherwise, all project folders inherit the settings from the MapForce project.

16.5.3 Handling JDBC references

If the mapping connects to a database through JDBC, ensure that the JDBC drivers used by the mapping are installed on your system (see Creating a JDBC connection). To view the current JDBC settings of any database component, click it, and then select **Component | Properties** from the menu.

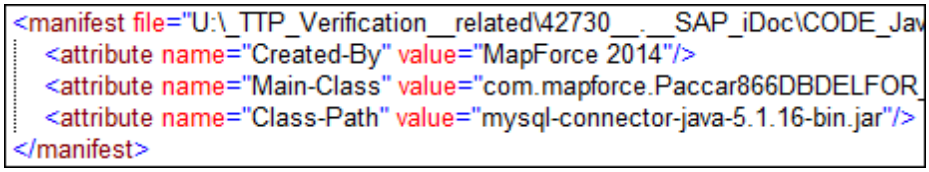
Additionally, if you build JAR files from generated Java code, add the "Class-Path" attribute for your database driver to the **build.xml** file. This ensures that the reference to the database driver is available in the manifest (**MANIFEST.MF**) file after you build the project.

To add the "Class-Path" attribute:

1. Add to the **build.xml** file a reference to the JAR file of the database driver, as a new "Class-Path" attribute. For example, for MySQL 5.1.16, the value of the "Class-Path" attribute looks as follows:

```
<attribute name="Class-Path" value="mysql-connector-java-5.1.16-
bin.jar"/>
```

   The **manifest** element of the **build.xml** file now looks similar to the screen shot below.



2. Copy the JAR file of the JDBC driver to the folder that contains the JAR file of the generated application.

16.5.4 Building the project with Ant

Apache Ant is a widely used Java library (and command-line tool) which automates building and compilation of Java projects (see http://ant.apache.org/manual/). Ant works with build files (such files define the sources and targets from which code must be compiled, as well as any specific build options). Since any MapForce-generated project includes a **build.xml** file recognized by Ant, you can easily build MapForce-generated projects with Ant.

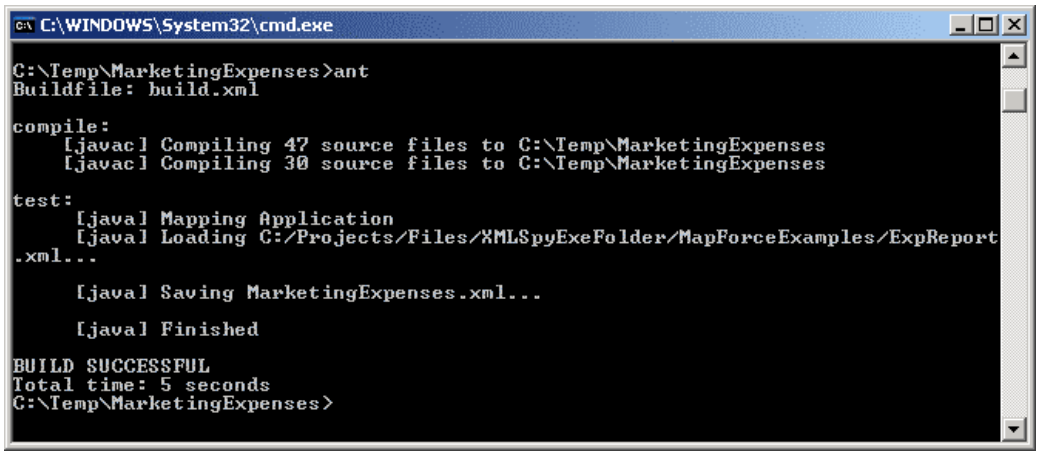
Ant may be available on your system either as a standalone installation, or bundled within Eclipse (or other Java IDEs). For instructions on how to install Ant on your system, see http://ant.apache.org/manual/. For instructions on how to use Ant in Eclipse, refer to the Eclipse tutorial, as well as the Eclipse documentation.

You can quickly check whether the standalone version of Ant (not the one bundled with Eclipse) is available on your system by opening a command prompt and typing `ant` at the command line. When Ant is already available, the resulting message will be similar to: `Buildfile: buildxml does not exist!` This message indicates that Ant is installed and it is attempting to build a **build.xml** file, but the latter does not exist in the current directory. If you run Ant from a directory which includes a **build.xml** file, Ant executes the **build.xml** file instead, with whatever build options are defined in it.

To build a MapForce-generated Java project with Ant:

1. Open a command prompt and navigate to the directory where the Java project was generated (note that the directory must contain the **build.xml** file).
2. At the command prompt, enter `ant`. This will compile and execute the Java code

according to the options defined in the **build.xml** file.



To generate a JAR file with Ant:

- At the command prompt, enter `ant jar`.

If you need help with Ant command syntax and options, enter `ant -help` at the command line.

16.5.5 Example: Build a Java application with Eclipse and Ant

This example walks you through the steps required to generate a Java application with MapForce, and compile it outside of MapForce using the Eclipse Integrated Development Environment (IDE) and Apache Ant. After completing this example, you will have created and compiled a complete Java application which executes one of the mapping samples available by default in MapForce.

If you can already compile successfully other Java applications with Eclipse and Ant, there are no special requirements to run this example. Otherwise, note the following prerequisites:

- The Java Development Kit (JDK), Eclipse (https://www.eclipse.org/), and Ant (http://ant.apache.org/) must be installed on your system. Eclipse typically includes support for building projects with Ant (see also Building the project with Ant).
- The Windows PATH environment variable must include the path to the JDK's **bin** directory (for example, **C:\Java\jdk1.7.0\bin**). This is a basic requirement for developing applications for the Java platform. For instructions, see http://docs.oracle.com/javase/tutorial/essential/environment/paths.html.

This example uses the following configuration:

- JDK 1.7
- Eclipse IDE for Java Developers (Luna Service Release 4.4.1)
- Ant 1.9.2, which is already integrated into the above-mentioned edition of Eclipse; therefore, it was not installed and configured separately.

The example is organized into the following sub-tasks:

- Step 1: Generate Java Code
- Step 2: Import the Project into Eclipse

16.5.5.1 Step 1: Generate Java code

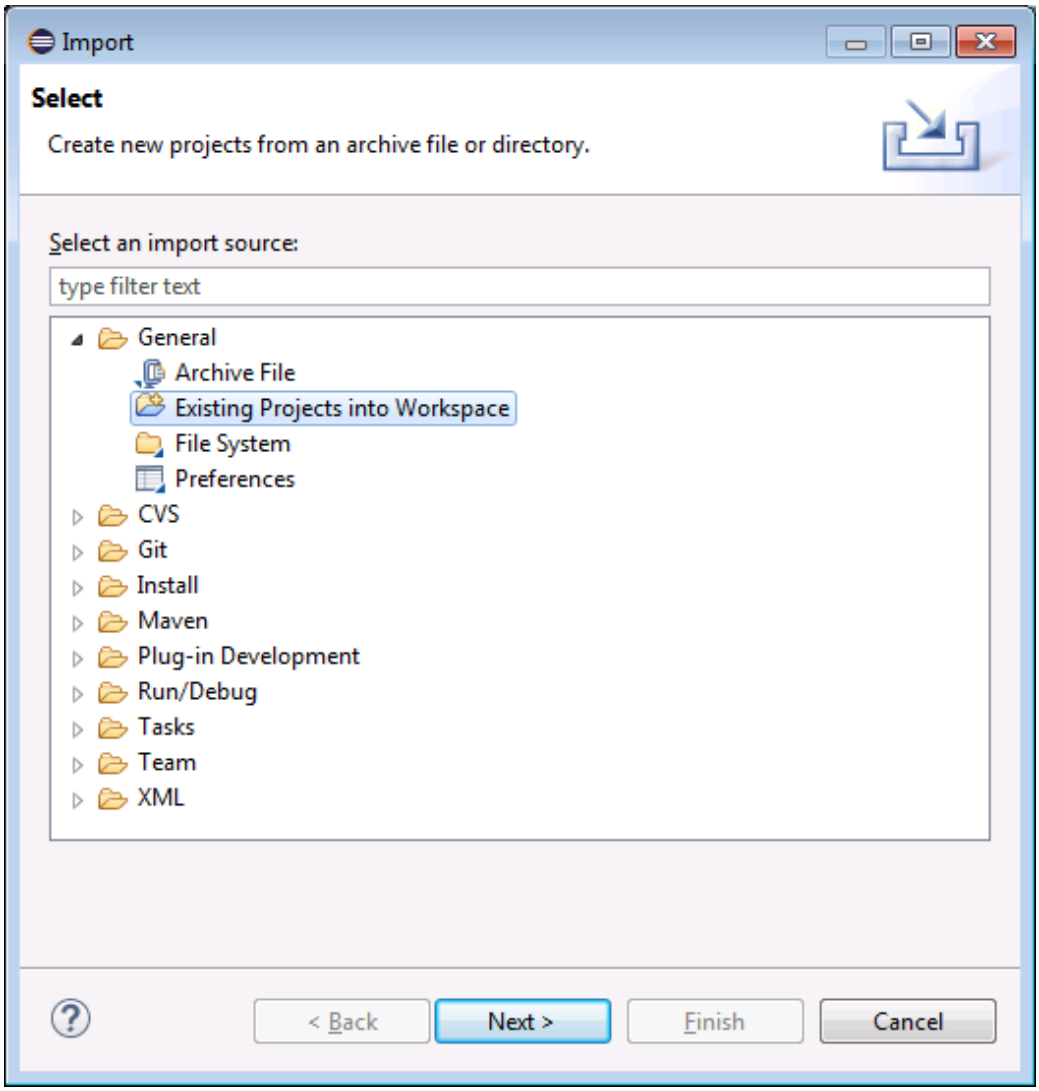
To generate the Java code in MapForce:

1. On the **File** menu, click **Open**, and browse for the **CompletePO.mfd** mapping available in the <Documents>\Altova\MapForce2016\MapForceExamples\ directory.
2. On the **Output** menu, click **Java**. This changes the transformation language to Java.
3. On the **File** menu, click **Generate code in | Java**. When prompted, browse for the directory to which the Java project must be saved. For convenience, you may choose to save the project to **C:\workspace\CompletePO** (where **C:\workspace** is your default Eclipse workspace).

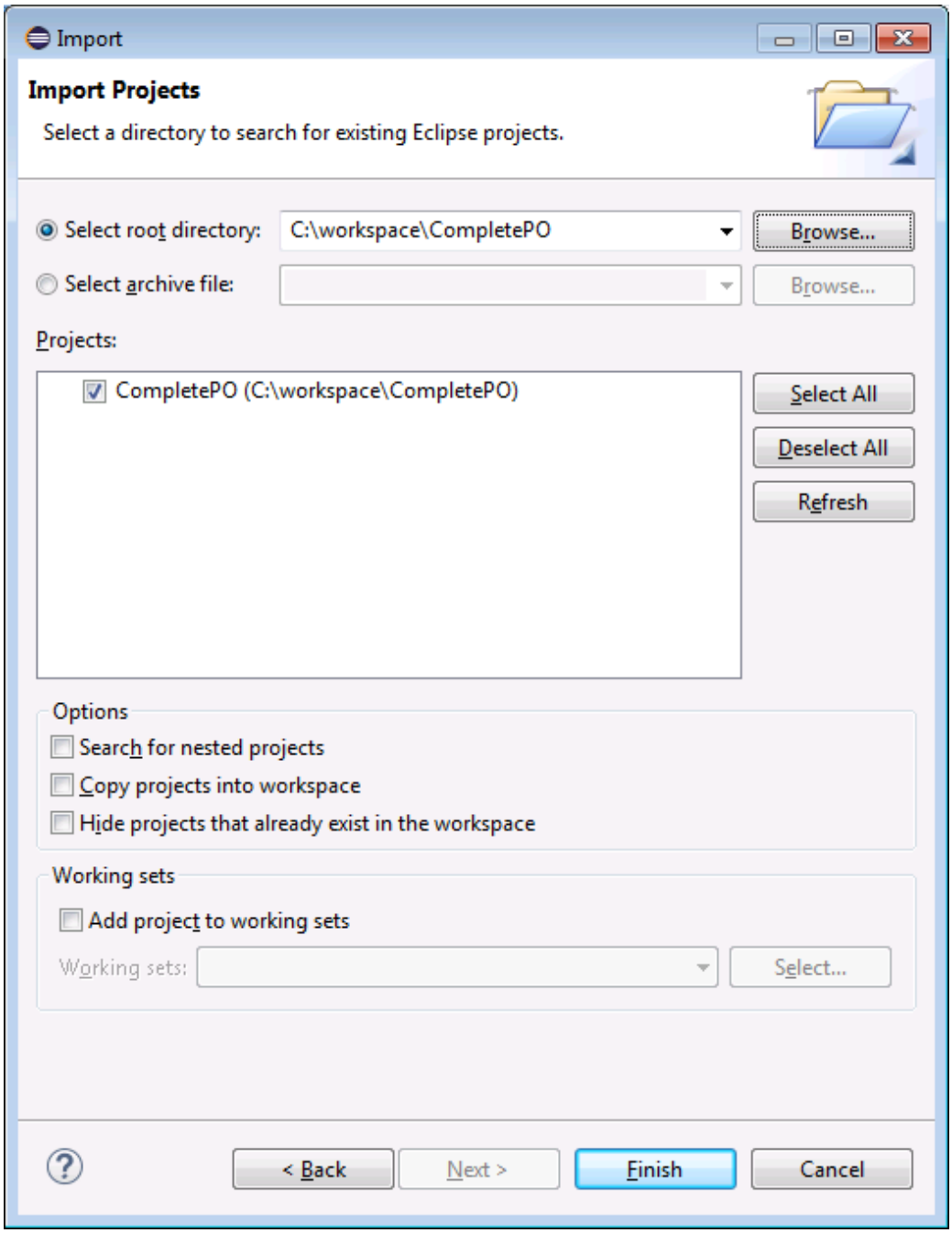
16.5.5.2 Step 2: Import the project into Eclipse

To import the project into Eclipse:

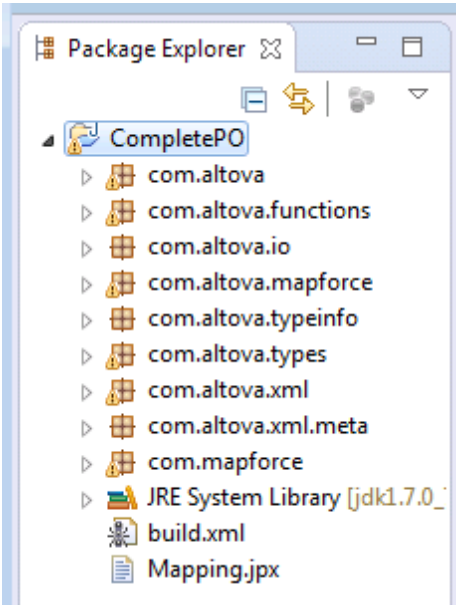
1. If you haven't done so already, run Eclipse and switch to the default Java perspective using the menu command **Window | Open Perspective**.
2. On the **File** menu, click **Import**, and then select **Existing Projects into Workspace**.



3. Click **Next**.



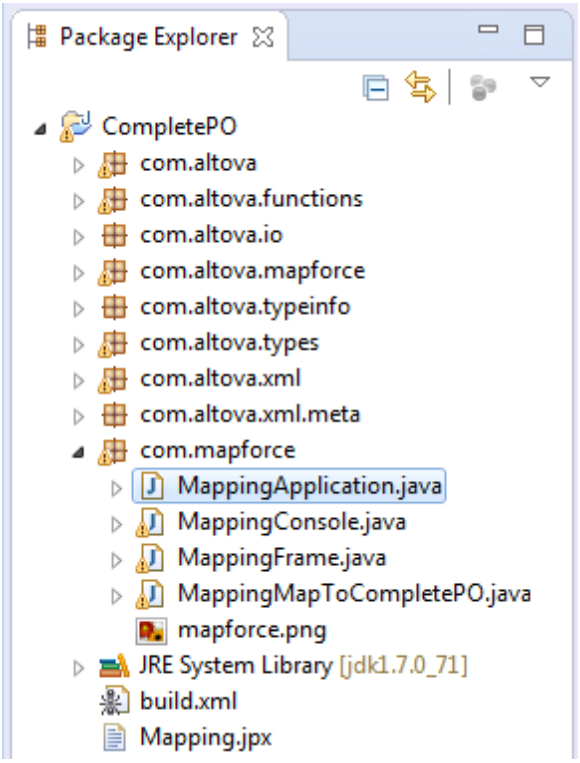
4. Browse for the folder where you have previously saved the generated code, and then click **Finish**. The Java project created by MapForce is now available in the Package Explorer view. If you cannot see the Package Explorer view, display it using the menu command **Window | Show View | Package Explorer**.



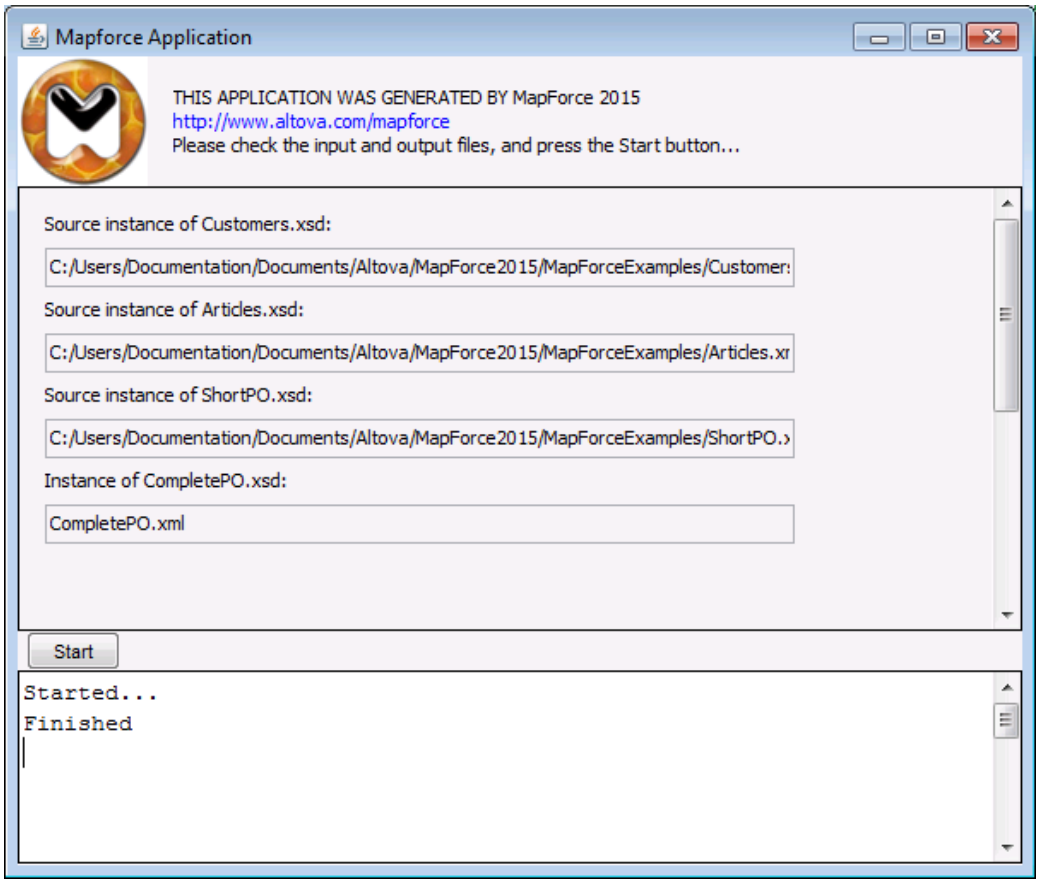
16.5.5.3 Step 3: Run the project as dialog application

To run the Java project as a GUI application:

1. In the Package Explorer view of Eclipse, click the **MappingApplication.java** file available in the **com.mapforce** package.



2. On the **Run** menu, click **Run As | Java application**.
3. On the MapForce application window, click **Start** to execute the mapping.

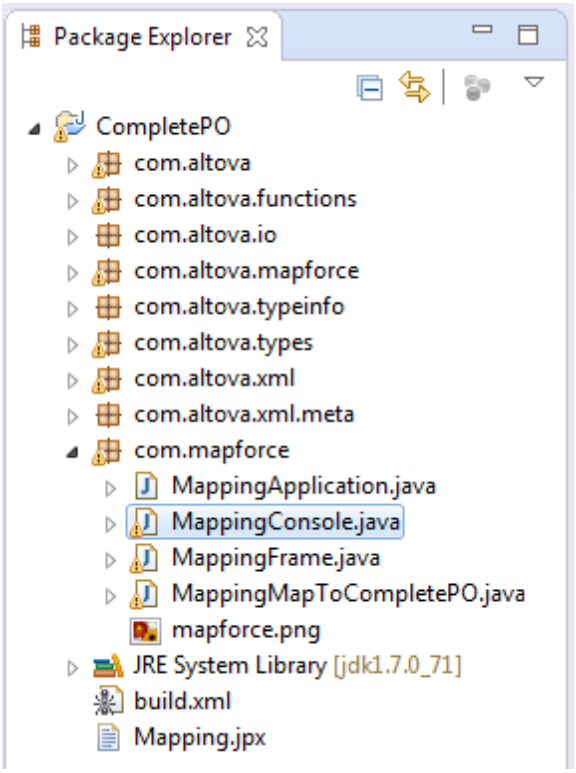


If Eclipse encounters system configuration or run-time errors, you will be prompted. Otherwise, the Java application executes the mapping transformation and generates the **CompletePO.xml** at the output path (in this example: **C:\workspace\CompletePO\CompletePO.xml**).

16.5.5.4 Step 4: Run the project as console application

To run the Java project as a console application:

1. In the Package Explorer view of Eclipse, click the **MappingConsole.java** file available in the **com.mapforce** package.



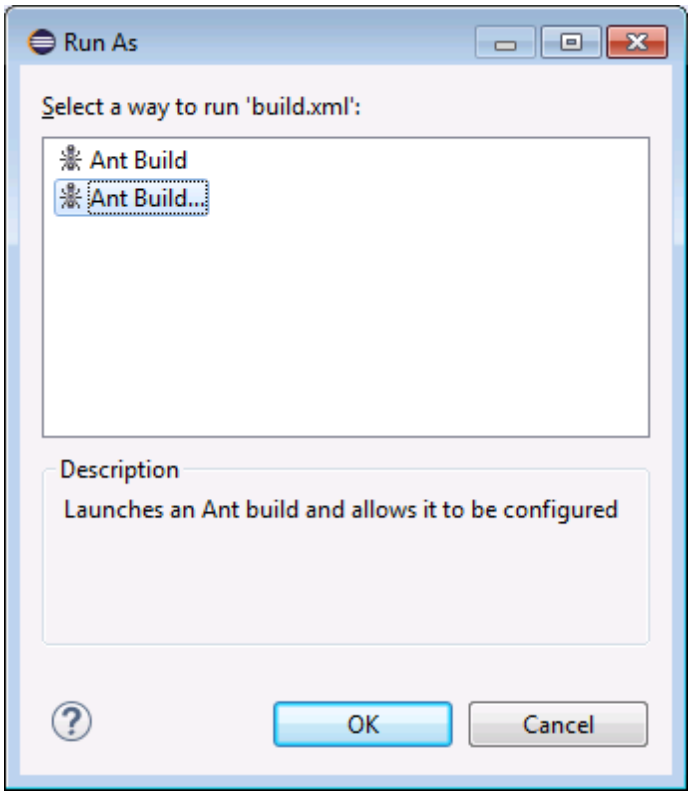
2. On the **Run** menu, click **Run As | Java application**.

If Eclipse detects system configuration or run-time errors, you will be prompted. Otherwise, the Java application executes the mapping transformation and generates the **CompletePO.xml** at the output path (in this example: **C:\workspace\CompletePO\CompletePO.xml**).

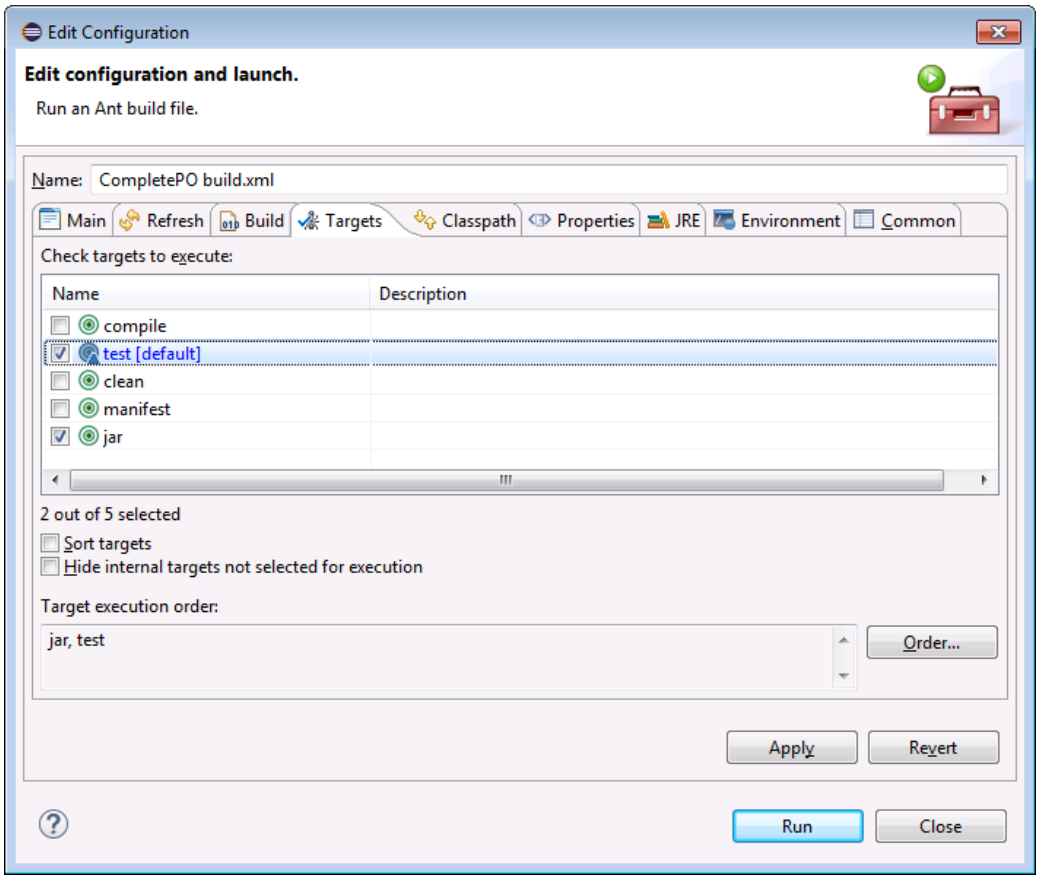
16.5.5.5 Step 5: Build the JAR file with Ant

To build the .jar file with Ant:

1. In the Package Explorer view of Eclipse, click the **build.xml** file available directly in the project root.
2. On the Run menu, click Run.



3. In the Run As dialog box, two possible options to run the Ant build file are displayed. If you choose the first option, Eclipse launches the Ant build with the default settings. If you choose the second option, you can change the settings of the Ant build before launching it. Select the second option.
4. Click to enable the targets that you wish to include in the Ant build. In this particular example, the goal is to generate a .jar file from the mapping project. Therefore, select **jar** as target, in addition to the default **test** target.



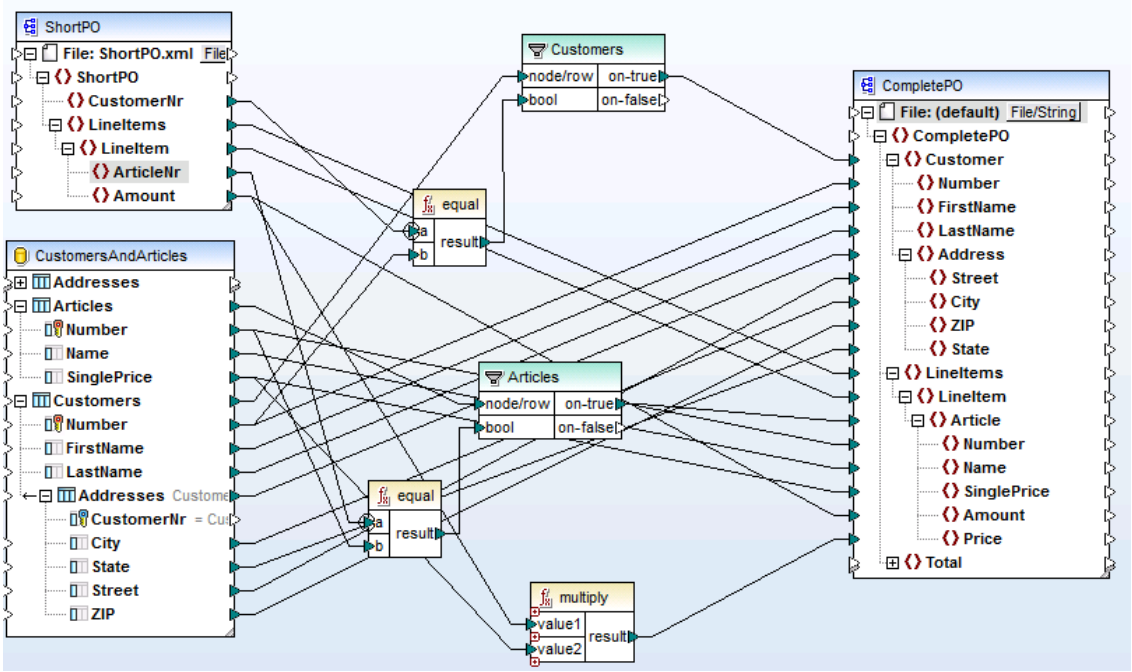
5. Click **Run**. Eclipse executes the Ant build file and displays the result in the Console view.

16.6 Integrating MapForce code in your application

MapForce-generated code can be integrated, or adapted to your specific application, even though the result of code generation is a complete and fully-functioning application. Some typical scenarios where you might want to change the generated code are as follows:

- Define custom source or target files for the mapping application
- Add custom error handling code
- In C# or Java generated code, you can also change the data type of the mapping input programmatically (for example, from string to stream).

This section provides instructions on how to achieve these goals, based on the **DB_CompletePO.mfd** sample mapping available in the <Documents>\Altova\MapForce2016 \MapForceExamples\ directory.



DB_CompletePO.mfd mapping sample in MapForce

As illustrated above, the sample mapping consists of two sources and one target:

- **ShortPO.xml** is a source XML file
- **CustomersAndArticles.mdb** is a source database
- **CompletePO.xml** is the target XML file.

In the generated code, these sources and targets will translate to two input and one output parameters supplied to the **run** method which executes the mapping (as described in the subsequent topics). For now, note the following basic points about code generation:

- The number of source and targets in the mapping design corresponds to the number of mapping parameters to the **run** method in the generated code.

- If you change the number of sources or targets of the mapping, then you will need to re-generate the code accordingly.
- If you make changes to the generated code, and then re-generate the code at the same location, all changes will be overwritten.

If a mapping includes database components, the generated **run** method includes the database connection object at the appropriate location. For example, if the mapping uses three sources (text content, XML content and a database) to map to a single output file, MapForce generates the following **run** method:

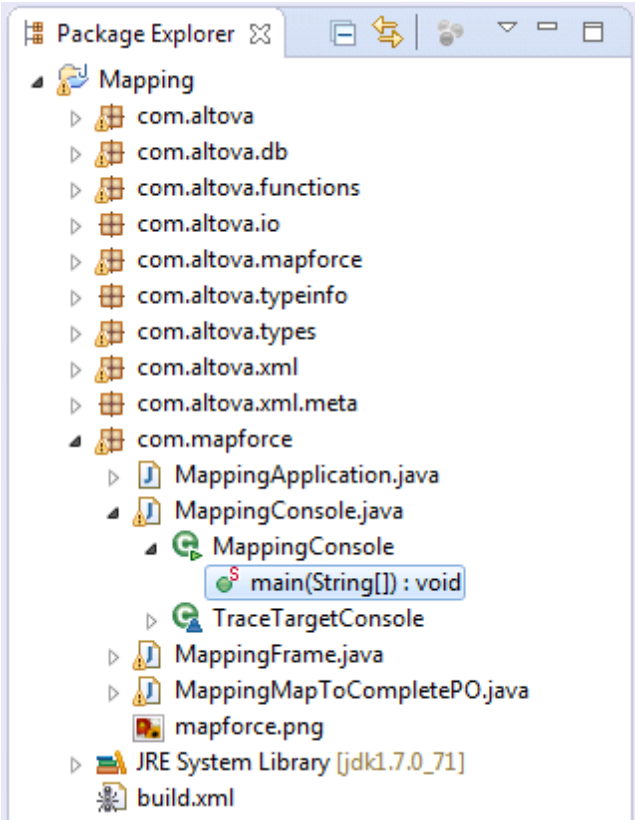
Java

```
void run(Input in1, Input in2, java.sql.Connection dbConn, Output out1);
```

The argument order is important. As you will see in the subsequent examples, you can modify **dbConn** parameters, or use the default parameters generated by MapForce when integrating your code.

16.6.1 Java example

This example uses Eclipse as Java IDE. To begin, generate Java code from the **DB_CompletePO** sample mapping available in the <Documents>\Altova\MapForce2016\MapForceExamples\ directory, and then import the project into Eclipse.



Sample MapForce-generated Java application (Eclipse IDE)

To edit the generated Java console application, locate the `main(String[] args)` method of your generated application (see the screen shot above). If you did not change the default base package name before generating code, this method is in the `MappingConsole` class of the `com.mapforce` package. Otherwise, it is in the `MappingConsole` class of your custom defined package.

To edit the generated Java dialog application, locate the place in the code where the `run` method is invoked from your generated application. If you did not change the default base package name before generating code, the **`run`** method is invoked from the class called `MappingFrame.java` of the `com.mapforce` package.

The following code sample illustrates an extract from the `main` method in the generated Java console application. The mapping sources and targets are highlighted in yellow and are defined as parameters to the run method. Since this mapping uses a database connection, the corresponding parameter has a special structure. Namely, the connection consists of the connection string (in this case, `jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);DBQ=CustomersAndArticles.mdb`), as well as two empty arguments intended for the **Username** and **Password** (in clear text) for those databases where this data is necessary.

Note that the file paths in the code below have been changed from absolute to relative.

```
com.altova.io.Input ShortPO2Source =
com.altova.io.StreamInput.createInput("ShortPO.xml");
```



```
com.altova.io.Output CompletePO2Target = new
com.altova.io.FileOutput("CompletePO.xml");

MappingMapToCompletePOObject.run(
                  com.altova.db.Dbs.newConnection(
                     "jdbc:odbc:;DRIVER=Microsoft Access Driver
(*.mdb);DBQ=CustomersAndArticles.mdb",
                     "",
                     ""),
                  ShortPO2Source,
                  CompletePO2Target);
```

To define custom mapping source or target files:

- Locate the parameters passed to the `run` method and edit them as required. In the sample above, `com.altova.db.Dbs.newConnection` and `ShortPO2Source` is the mapping input and `CompletePO2Target` is the mapping output.

To add extra error handling code:

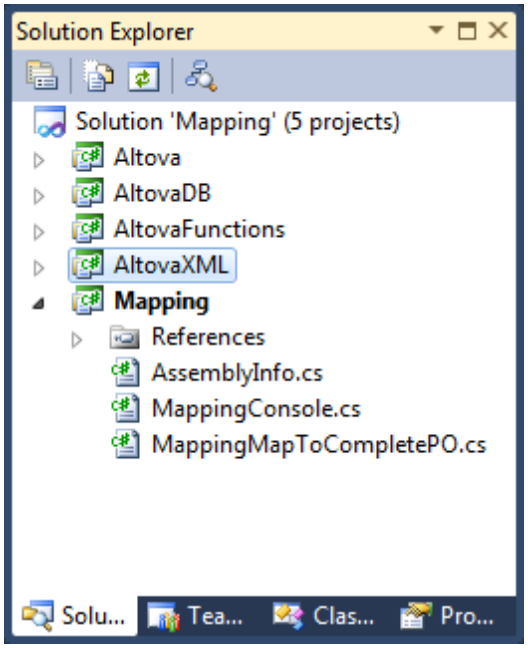
- Edit the code below the **`catch`** `(Exception e)` code (in case of a Java console application)
- Edit the code below the **`catch`** `(Exception ex) code` (in case of a Java dialog application)

For instructions on how to change the data type of parameters supplied as mapping input/output, see Changing the data type of the mapping input/output (C#, Java).

16.6.2 C# example

This example uses the Visual Studio 2010 IDE. To begin, generate C# code from the **DB_CompletePO** sample mapping available in the <Documents>\Altova\MapForce2016 \MapForceExamples\ directory, and then open the solution in Visual Studio.

By default, the name of the solution file is **Mapping.sln**, and it is located in the **Mapping** subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to **MyApplication**, then the solution file is called **MyApplication.sln**, and it is located in the **MyApplication** subdrectory.



Sample C# solution generated with MapForce

Open the MappingConsole.cs file, and locate the `main(String[] args)` method. The following code sample illustrates an extract from the `main` method. The mapping sources and targets are highlighted in yellow and are defined as parameters to the **Run** method. Since this mapping reads data from a database, there is also an input parameter which is a database connection string. If necessary, you can modify the connection string of the database.

Note that the file paths in the code below have been changed from absolute to relative.

```
Altova.IO.Input ShortPO2Source =
Altova.IO.StreamInput.createInput("ShortPO.xml");
Altova.IO.Output CompletePO2Target = new
Altova.IO.FileOutput("CompletePO.xml");

MappingMapToCompletePOObject.Run(
                                  "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=CustomersAndArticles.mdb; ",
                                  ShortPO2Source,
                                  CompletePO2Target);
```

To define custom mapping source or target files:

- Locate the parameters passed to the **Run** method and edit them as required. In the sample above, the mapping input is a connection string to the CustomersAndArticles.mdb and **ShortPO2Source** . The mapping output is **CompletePO2Target** .

To add extra error handling code:

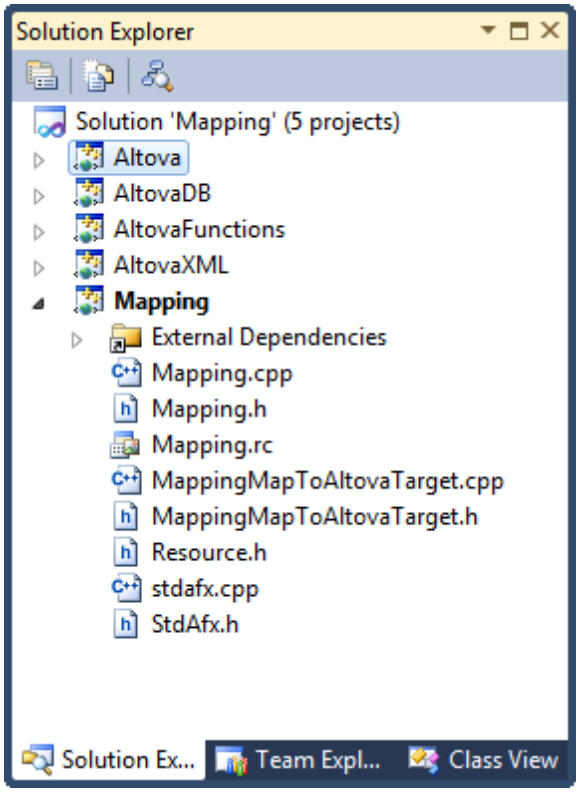
- Edit the code below the `catch (Exception e)` code

For instructions on how to change the data type of parameters supplied as mapping input/output, see Changing the data type of the mapping input/output (C#, Java).

16.6.3 C++ example

This example uses the Visual Studio 2010 IDE. To begin, generate C++ code from the **DB_CompletePO** sample mapping available in the <Documents>\Altova\MapForce2016 \MapForceExamples\ directory, and then open the solution in Visual Studio.

By default, the name of the solution file is **Mapping.sln**, and it is located in the **Mapping** subdirectory relative to the directory where you saved the generated code. If you changed the application name from the mapping settings, then the name of the .sln file is changed accordingly. For example, if you changed the application name to **MyApplication**, then the solution file is called **MyApplication.sln**, and it is located in the **MyApplication** subdrectory.



Sample C++ solution generated with MapForce

Open the Mapping.cpp file, and locate the `_tmain` method. The following code sample illustrates an extract from this method. The mapping sources and targets are defined as parameters to the **Run** method. Since this mapping reads data from a database, there is also an input parameter

which is a database connection string. If necessary, you can modify the connection string of the database.

Note that the file paths in the code below have been changed from absolute to relative.

```
MappingMapToCompletePO MappingMapToCompletePOObject;
                    MappingMapToCompletePOObject.Run(
                            _T("Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=CustomersAndArticles.mdb; "),
                            _T("ShortPO.xml"),
                            _T("CompletePO.xml"));
```

To define custom mapping source or target files:

- Locate the parameters passed to the **Run** method and edit them as required. In the code sample above, the mapping input is a connection string to the **CustomersAndArticles.mdb** database and **_T("ShortPO.xml")** . The mapping output is **_T("CompletePO.xml")** .

To add extra error handling code:

- Edit the code below the **catch** (CAltovaException& e) code.

16.6.4 Changing the data type of the mapping input/output (C#, Java)

This topic provides details on the object types you can use programmatically, if you intend to run MapForce mappings from a custom Java or C# application.

You can use several input and output objects (such as files, strings, DOM documents, and others) as parameters the **run** method. The **run** method is the most important function of generated mapping classes. It has one parameter for each static source or input component in the mapping, and a final parameter for the output component. Components that process multiple files do not appear as parameters to the **run** method, because in this case the file names are processed dynamically inside the mapping.

The objects that you can provide as parameters to the **run** method are available in the **com.altova.io** package (Java) and Altova.IO namespace (C#). The base classes of the generated input and output objects are as follows:

C#

```
Altova.IO.Input
Altova.IO.Output
```

Java

```
com.altova.io.Input
com.altova.io.Output
```

The object types supported as input/output parameters to the **run** method, including their applicable input/output file formats, are listed in the following table.

Object Type	XML	Microsoft Excel*	EDI* **	FlexText*	CSV	Fixed-length files
Files	Y	Y	Y	Y	Y	Y
Binary stream objects	Y	Y	Y	Y	Y	Y
Strings	Y	–	Y	Y	Y	Y
I/O Reader/Writer (character stream objects)	Y	–	Y	Y	Y	Y
DOM documents	Y	–	–	–	–	–

***** Formats supported only in MapForce Enterprise Edition
****** Includes X12 and HL7

Files

File objects (identified in the code file names) have the following definition:

C#

```
Altova.IO.FileInput(string filename)
Altova.IO.FileOutput(string filename)
```

Java

```
com.altova.io.FileInput(String filename)
com.altova.io.FileOutput(String filename)
```

Binary stream objects

Binary stream objects in the generated code represent an alternative way to working with physical files; there are no advantages as far as memory use is concerned. Binary stream objects have the following definition:

C#

```
Altova.IO.StreamInput(System.IO.Stream stream)
Altova.IO.StreamOutput(System.IO.Stream stream)
```

Java

```
com.altova.io.StreamInput(java.io.InputStream stream)
com.altova.io.StreamOutput(java.io.OutputStream stream)
```

Notes:

- Binary stream objects are expected to be opened and ready-to-use before calling the **run**

method.

- By default, the **run** method closes the stream when finished. To prevent this behaviour, insert the following code before calling the **run** method:

Java

```
MappingMapToSomething.setCloseObjectsAfterRun(false);  // Java
```

C#

```
MappingMapToSomething.CloseObjectsAfterRun = false;   // C#
```

Strings

String objects have the following definition:

C#

```
Altova.IO.StringInput(string content)
Altova.IO.StringOutput(StringBuilder content)
```

Java

```
com.altova.io.StringInput(String xmlcontent)
com.altova.io.StringOutput()
```

In Java, **StringOutput** does not take an argument. Content can be accessed with:

```
// mapping from String to (another) String
String MyText = "<here>is some XML text</here>";

Input input = new StringInput(MyText);
Output output = new StringOutput();

MappingMapToMyText.run(input, output);

String myTargetData = output.getString().toString();
```

The **getString()** method returns a **StringBuffer**, hence the need for **toString()**.

In C#, **StringOutput** takes an argument (**StringBuilder**) which you need to provide beforehand. The **StringBuilder** may already contain data, so the mapping output is appended to it.

Excel sources/targets cannot map to or from strings.

I/O Reader/Writer (character stream objects)

Character stream objects have the following definition:

C#

```
Altova.IO.ReaderInput(System.IO.TextReader reader)
Altova.IO.WriterOutput(System.IO.TextWriter writer)
```

Java

```
com.altova.io.ReaderInput(java.io.Reader reader)
com.altova.io.WriterOutput(java.io.Writer writer)
```

Notes:

- Character stream objects are expected to be opened and ready-to-use before calling the **run** method.
- Excel sources/targets cannot be read from, or written to, character streams.
- By default, the **run** method closes the stream when finished. To prevent this behaviour, insert the following code before calling the **run** method:

Java

```
MappingMapToSomething.setCloseObjectsAfterRun(false);  // Java
```

C#

```
MappingMapToSomething.CloseObjectsAfterRun = false;   // C#
```

DOM documents

DOM documents have the following definition:

C#

```
Altova.IO.DocumentInput(System.Xml.XmlDocument document)
Altova.IO.DocumentOutput(System.Xml.XmlDocument document)
```

Java

```
com.altova.io.DocumentInput(org.w3c.dom.Document document)
com.altova.io.DocumentOutput(org.w3c.dom.Document document)
```

Notes:

- The document passed to the **DocumentOutput** constructor as target must be empty.
- After calling **run**, the DOM Document generated by the constructor of **DocumentOutput** already contains mapped data so "save to document" is not necessary. After mapping, you can manipulate the document as necessary.
- Only XML content can be mapped to DOM documents.

Example

Let's assume you want to integrate the code generated by MapForce into your Java application.

Your MapForce mapping consists of two source XML files and a target text file. When you generate the MapForce code, the **run** function looks as follows:

```
void run(Input in1, Input in2, Output out1);
```

Let's also assume that your application requires that you map data from a local file and binary stream into a character stream. Since data is supplied from other sources, your application must declare the sources and targets as:

```
String filename; // Declare the source of the first input
Java.io.InputStream stream; // Declare the source of the second input
Java.io.Writer writer; // Declare the output as character stream
```

The following wrappers must be constructed for the MapForce-generated **run** function:

```
// com.altova.io is considered imported here:
Input input1 = new FileInput(filename);
Input input2 = new StreamInput(stream);
Output output1 = new WriterOutput(writer);
```

Now you can call the MapForce generated run function:

```
MappingMapToSomething.run(input1, input2, output1);
```

The **C#** behavior is almost identical, except that **run** is called **Run**, and the .NET stream and reader/writer classes are named differently.

Using the same technique, you can also use other input and output types, such as strings or DOM documents.

16.7 Using the generated code libraries

When you generate code from a mapping, MapForce generates a complete application that executes all steps of the mapping automatically. Optionally, you can generate libraries for all the XML schemas used in the mapping. These allow your code to easily create or read XML instances that are used or created by the mapping code.

To generate libraries for all the XML schemas used in the mapping, select the **Generate Wrapper Classes** check box in the Options dialog (see Code generator options). When you do this, MapForce creates not only the mapping application, but also wrapper classes for all schemas used in the mapping. These can be used in your own code in the same way as libraries generated by XMLSpy's code generator.

Name	Purpose
Altova	Base library containing common runtime support, identical for every schema.
AltovaXML	Base library containing runtime support for XML, identical for every schema.
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.

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

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 default settings in the SPL 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 much more elaborate data type model than Java, C# or C++. Code Generator converts the 44 XML Schema types to a couple of language-specific primitive types, or to classes delivered with the Altova library. The mapping of simple types can be configured in the SPL template.

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.

Generated Classes

MapForce generally generates one class per type found in the XML Schema, or per element in the DTD. One additional class per library is generated to represent the document itself, which can be used for loading and saving the document.

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.

16.7.1 Example schema

To keep things simple, we will work with a very small xsd file, the source code is shown below. Save this as **Library.xsd** if you want to get the same results as our example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.nanonull.com/LibrarySample" xmlns:xs="http://
www.w3.org/2001/XMLSchema" targetNamespace="http://www.nanonull.com/
LibrarySample" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:element name="Library">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Book" type="BookType" minOccurs="0"
maxOccurs="unbounded"/>
         </xs:sequence>
      </xs: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="ISBN" type="xs:string" use="required"/>
      <xs:attribute name="Variant" type="BookVariant"/>
   </xs:complexType>
   <xs:simpleType name="BookVariant">
      <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>
```

We will only use this schema file in this example, it will therefore be easy to figure out the differences between the different programming languages in the following code examples.

Please note that the generated classes will be named differently from xsd to xsd file. When working with your own schemas or other samples, make sure you adapt the names of the functions and variables.

16.7.2 Using the generated Java library

Generated Packages

Name	Purpose
com.altova	Base package containing common runtime support, identical for every schema
com.altova.xml	Base package containing runtime support for XML, identical for every schema
com.*YourSchema*	Package containing declarations generated from the input schema, named as the schema file or DTD
com.*YourSchema*Test	Test application skeleton (XMLSpy only)

DOM Implementation

The generated Java code uses JAXP (Java API for XML Processing) as the underlying DOM interface.

Data Types

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.

XML Schema	Java	Remarks
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	See DateTime and Duration Classes
xs:duration	com.altova.types.Duration	See DateTime and Duration Classes
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 Document Class

In addition to the classes for the types declared in the XML Schema, the document class is generated. It contains all possible root elements as members, and in addition the following methods ("Doc" stands for the name of the generated document class itself):

Method	Purpose
static Doc loadFromFile(String fileName)	Loads an XML document from a file
static Doc loadFromString(String xml)	Loads an XML document from a string
static Doc loadFromBinary(byte[] xml)	Loads an XML document from a byte array
void saveToFile(String fileName, boolean prettyPrint, String encoding = "UTF-8")	Saves an XML document to a file with the specified encoding. If no encoding is specified, UTF-8 is assumed.
void saveToFile(String fileName, boolean prettyPrint, String encoding, boolean bigEndian, boolean writeBOM)	Saves an XML document to a file 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
byte[] saveToBinary(boolean prettyPrint, String encoding = "UTF-	Saves an XML document to a byte array with the specified encoding. If no encoding is

Method	Purpose
8")	specified, UTF-8 is assumed.
byte[] saveToBinary(boolean prettyPrint, String encoding, boolean bigEndian, boolean writeBOM)	Saves an XML document to a byte array with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
static Doc createDocument()	Creates a new, empty XML document.
void setSchemaLocation(String schemaLocation)	Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.

Generated Type Class

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 and mixed types, **getValue**() and **setValue**(string) methods are generated. For simple types with enumeration facets, **getEnumerationValue**() and **setEnumerationValue**(int) can be used together with generated constants for each enumeration value. In addition, the method **getStaticInfo**() allows accessing schema information as **com.altova.xml.meta.SimpleType** or **com.altova.xml.meta.ComplexType** (see below).

XML Schema derivation of complexTypes by extension results in derived classes in the generated library.

Generated Member Attribute Class

For each member attribute of a type, a class with the following methods is created.

Method	Purpose
boolean exists()	Returns true if the attribute exists
void remove()	Removes the attribute from its parent element
AttributeType getValue()	Gets the attribute value
void setValue(AttributeType value)	Sets the attribute value
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.
void setEnumerationValue(int)	Generated for enumeration types only. Pass one of the constants generated for the

Method	Purpose
	possible values to this method to set the value.
`com.altova.xml.meta.Attribute getInfo()`	Returns an object for querying schema information (see below)

Generated Member Element Class

For each member element of a type, a class with the following methods is created.

Method	Purpose
`MemberType first()`	Returns the first instance of the member element
`MemberType at(int index)`	Returns the instance of the member element
`MemberType last()`	Returns the last instance of the member element
`MemberType append()`	Creates a new element and appends it to its parent
`boolean exists()`	Returns true if at least one element exists
`int count()`	Returns the count of elements
`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
`java.util.Iterator iterator()`	Returns an object for iterating instances of the member element.
`MemberType getValue()`	Gets the element content (only generated if element can have simple or mixed content)
`void setValue(MemberType value)`	Sets the element content (only generated if element can have simple or mixed content)
`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.
`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.

Method	Purpose
`com.altova.xml.meta.Element getInfo()`	Returns an object for querying schema information (see below)

Creating XML Documents

The generated code library makes it very easy to create XML files. Let's start with an example XML file for the example schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<Library xsi:schemaLocation="http://www.nanonull.com/LibrarySample
Library.xsd" xmlns="http://www.nanonull.com/LibrarySample" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <Book ISBN="0764549642" Variant="Paperback">
      <Title>The XML Spy Handbook</Title>
      <Author>Altova</Author>
   </Book>
</Library>
```

The following Java code was used to create this file. The classes Library2, LibraryType and BookType are generated out of the schema and represent the complete document, the type of the <Library> element, and the complex type BookType, which is the type of the <Book> element. Note that the automatic name de-collision renamed the Library class to Library2 to avoid a possible conflict with the library namespace name.

```
   protected static void example() throws Exception {
      // create a new, empty XML document
      Library2 libDoc = Library2.createDocument();

      // create the root element <Library> and add it to the document
      LibraryType lib = libDoc.Library3.append();

      // create a new <Book> and add it to the library
      BookType book = lib.Book.append();

      // set the ISBN attribute of the book
      book.ISBN.setValue("0764549642");

      // set the Variant attribute of the book using an enumeration constant
      book.Variant.setEnumerationValue( BookVariant.EPAPERBACK );

      // add the <Title> and <Author> elements, and set values
      book.Title.append().setValue("The XML Spy Handbook");
      book.Author.append().setValue("Altova");

      // 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);
   }
```

Note that all elements are created by calling append(), and that attributes (like ISBN in the example) can simply be accessed like structure members.

```
   protected static void example() throws Exception {
      // load XML document
      Library2 libDoc = Library2.loadFromFile("Library1.xml");

      // get the first (and only) root element <Library>
      LibraryType lib = libDoc.Library3.first();

      // it is possible to check whether an element exists:
      if (!lib.Book.exists())
      {
         System.out.println("This library is empty.");
         return;
      }

      // iteration: for each <Book>...
      for (java.util.Iterator itBook = lib.Book.iterator();
itBook.hasNext(); )
      {
         BookType book = (BookType) itBook.next();

         // output values of ISBN attribute and (first and only) title element
         System.out.println("ISBN: " + book.ISBN.getValue());
         System.out.println("Title: " + book.Title.first().getValue());

         // read and compare an enumeration value
         if (book.Variant.getEnumerationValue() == BookVariant.EPAPERBACK)
            System.out.println("This is a paperback book.");

         // for each <Author>...
         for (java.util.Iterator itAuthor = book.Author.iterator();
itAuthor.hasNext(); )
            System.out.println("Author: " + ((com.Library.xs.stringType)
itAuthor.next()).getValue());

         // alternative: use count and index
         for (int j = 0; j < book.Author.count(); ++j)
            System.out.println("Author: " + book.Author.at(j).getValue());
      }
   }
```

Reading XML Documents

The following example reads the file we just created and demonstrates the various methods for

reading XML files:

Note the type of the book.Author member: xs.stringType is a generated class for any element with simple string content. Similar classes are generated for all other XML Schema types.

Of course you can also load a document, modify it by adding or deleting elements and attributes, and then save the changed file back to disk.

Error Handling

Errors are reported by exceptions. The following exception classes are defined in the namespace com.altova:

Class	Base Class	Description
`SourceInstanceUnvailableException`	Exception	A problem occurred while loading an XML instance.
`TargetInstanceUnavailableException`	Exception	A problem occurred while saving an XML instance.

In addition, the following Java exceptions are commonly used:

Class	Description
`java.lang.Error`	Internal program logic error (independent of input data)
`java.lang.Exception`	Base class for runtime errors
`java.lang.IllegalArgumentsException`	A method was called with invalid argument values, or a type conversion failed.
`java.lang.ArithmeticException`	Exception thrown when a numeric type conversion fails.

Accessing Metadata

The generated library allows accessing static schema information via the following classes. The methods that return one of the metadata classes return null if the respective property does not exist.

com.altova.xml.meta.SimpleType

Method	Purpose
`SimpleType getBaseType()`	Returns the base type of this type
`String getLocalName()`	Returns the local name of the type

Method	Purpose
`String getNamespaceURI()`	Returns the namespace URI of the type
`int getMinLength()`	Returns the value of this facet
`int getMaxLength()`	Returns the value of this facet
`int getLength()`	Returns the value of this facet
`int getTotalDigits()`	Returns the value of this facet
`int getFractionDigits()`	Returns the value of this facet
`String getMinInclusive()`	Returns the value of this facet
`String getMinExclusive()`	Returns the value of this facet
`String getMaxInclusive()`	Returns the value of this facet
`String getMaxExclusive()`	Returns the value of this facet
`String[] getEnumerations()`	Returns an array of all enumeration facets
`String[] getPatterns()`	Returns an array of all pattern facets

`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`

com.altova.xml.meta.ComplexType

Method	Purpose
`Attribute[] GetAttributes()`	Returns a list of all attributes
`Element[] GetElements()`	Returns a list of all elements
`ComplexType getBaseType()`	Returns the base type of this type
`String getLocalName()`	Returns the local name of the type
`String getNamespaceURI()`	Returns the namespace URI of the type
`SimpleType getContentType()`	Returns the simple type of the content
`Element findElement(String localName, String namespaceURI)`	Finds the element with the specified local name and namespace URI

Method	Purpose
`Attribute findAttribute(String localName, String namespaceURI)`	Finds the attribute with the specified local name and namespace URI

com.altova.xml.meta.Element

Method	Purpose
`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.
`int getMinOccurs()`	Returns the minOccurs value defined in the schema
`int getMaxOccurs()`	Returns the maxOccurs value defined in the schema
`String getLocalName()`	Returns the local name of the element
`String getNamespaceURI()`	Returns the namespace URI of the element

com.altova.xml.meta.Attribute

Method	Purpose
`SimpleType getDataType()`	Returns the type of the attribute content
`boolean isRequired()`	Returns true if the attribute is required
`String getLocalName()`	Returns the local name of the attribute
`String getNamespaceURI()`	Returns the namespace URI of the attribute

16.7.3 Using the generated C++ library

Generated Libraries

Name	Purpose
`Altova`	Base library containing common runtime support, identical for every schema
`AltovaXML`	Base library containing runtime support for XML (MSXML or Xerces), identical for every schema

Name	Purpose
YourSchema	Library containing declarations generated from the input schema, named as the schema file or DTD
*YourSchema*Test	Test application skeleton (XMLSpy only)

DOM Implementations

The generated C++ code supports either Microsoft MSXML or Apache Xerces 2.6 or higher, depending on code generation settings. The syntax for using the generated code is identical for both DOM implementations.

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 to **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. Please take care with the **_T()** macros. This macro ensures that string constants are stored correctly, whether you're compiling for Unicode or non-Unicode programs.

Data Types

The default mapping of XML Schema types to C++ data types is:

XML Schema	C++	Remarks
xs:string	string_type	string_type is defined as std::string or std:wstring
xs:boolean	bool	
xs:decimal	double	C++ does not have a decimal type, so double is used.
xs:float, xs:double	double	
xs:integer	__int64	xs:integer has unlimited range, mapped to __int64 for efficiency reasons.
xs:nonNegativeInteger	unsigned __int64	see above
xs:int	int	
xs:unsignedInt	unsigned int	
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	altova::DateTime	See DateTime and Duration Classes

XML Schema	C++	Remarks
xs:duration	altova::Duration	See DateTime and Duration Classes
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 Document Class

In addition to the classes for the types declared in the XML Schema, the document class is generated. It contains all possible root elements as members, and in addition the following methods ("*CDoc*" stands for the name of the generated document class itself):

Method	Purpose
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
static CDoc LoadFromBinary(const std:vector<unsigned char>& xml)	Loads an XML document from a byte array
void SaveToFile(const string_type& fileName, bool prettyPrint, const string_type& encoding = "UTF-8")	Saves an XML document to a file with the specified encoding. If no encoding is specified, UTF-8 is assumed.
void SaveToFile(const string_type& fileName, bool prettyPrint, const string_type& encoding, bool bigEndian, bool writeBOM)	Saves an XML document to a file 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, const string_type& lineend) Note: Only available for Xerces3.	Saves an XML document to a file with the specified encoding and the specified line end.
void SaveToFile(const string_type& fileName, bool prettyPrint, const string_type& encoding, bool bigEndian, bool writeBOM, const string_type& lineend) Note: Only available for Xerces3.	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.
string_type SaveToString(bool	Saves an XML document to a string

Method	Purpose
`prettyPrint)`	
`std::vector<unsigned char> SaveToBinary(bool prettyPrint, const string_type& encoding = "UTF-8")`	Saves an XML document to a byte array with the specified encoding. If no encoding is specified, UTF-8 is assumed.
`std::vector<unsigned char> SaveToBinary(bool prettyPrint, const string_type& encoding, bool bigEndian, bool writeBOM)`	Saves an XML document to a byte array with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
`static CDoc CreateDocument()`	Creates a new, empty XML document. Must be released using DestroyDocument().
`void DestroyDocument()`	Destroys a document. All references to the document and its nodes are invalidated. This must be called when finished working with a document.
`void SetSchemaLocation(const string_type& schemaLocation)`	Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.
`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.

Generated Type Class

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 of schema information as **altova::meta::SimpleType** or **altova::meta::ComplexType** (see below).

XML Schema derivation of complexTypes by extension results in derived classes in the generated library.

Generated Member Attribute Class

For each member attribute of a type, a class with the following methods is created.

Method	Purpose
`bool exists()`	Returns true if the attribute exists
`void remove()`	Removes the attribute from its parent 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.
`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.
`altova::meta::Attribute info()`	Returns an object for querying schema information (see below)

In addition, assignment and conversion operators from/to the attribute's native type are created, so it can be used directly in assignments.

Generated Member Element Class

For each member element of a type, a class with the following methods is created.

Method	Purpose
`MemberType first()`	Returns the first instance of the member element
`MemberType operator[](unsigned int index)`	Returns the member element specified by the index
`MemberType last()`	Returns the last instance of the member element
`MemberType append()`	Creates a new element and appends it to its parent
`bool exists()`	Returns true if at least one element exists
`unsigned int count()`	Returns the count of elements
`void remove()`	Deletes all occurrences of the element from its parent
`void remove(unsigned int index)`	Deletes the occurrence of the element specified by the index
`Iterator<MemberType> all()`	Returns an object for iterating instances of the member element

Method	Purpose
`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.
`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.
`altova::meta::Element info()`	Returns an object for querying schema information (see below)

For simple and mixed content elements, assignment and conversion operators from/to the element's native type are created, so it can be used directly in assignments.

Creating XML Documents

The generated code library makes it very easy to create XML files. Let's start with an example XML file for the example schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<Library xsi:schemaLocation="http://www.nanonull.com/LibrarySample
Library.xsd" xmlns="http://www.nanonull.com/LibrarySample" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <Book ISBN="0764549642" Variant="Paperback">
      <Title>The XML Spy Handbook</Title>
      <Author>Altova</Author>
   </Book>
</Library>
```

The following C++ code was used to create this file. The classes CLibrary, CLibraryType and CBookType are generated out of the schema and represent the complete document, the type of the <Library> element, and the complex type BookType, which is the type of the <Book> element.

```
using namespace Library;
void Example()
{
    // create a new, empty XML document
    CLibrary libDoc = CLibrary::CreateDocument();

    // create the root element <Library> and add it to the document
    CLibraryType lib = libDoc.Library2.append();

    // create a new <Book> and add it to the library
    CBookType book = lib.Book.append();

    // set the ISBN attribute of the book
    book.ISBN = _T("0764549642");

    // set the Variant attribute of the book using an enumeration constant
    book.Variant.SetEnumerationValue( CBookVariant::k_Paperback );

    // add the <Title> and <Author> elements, and set values
    book.Title.append() = _T("The XML Spy Handbook");
    book.Author.append() = _T("Altova");

    // set the schema location (this is optional)
    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("Library1.xml"), true);

    // destroy the document - all references to the document and any of its
nodes become invalid
    libDoc.DestroyDocument();
}
```

Note that all elements are created by calling append(), and that attributes (like ISBN in the example) can simply be accessed like structure members. Remember to destroy the document when you have finished using it.

Reading XML Documents

The following example reads the file we just created and demonstrates the various methods for reading XML files:

```
using namespace Library;
void Example()
{
    // load XML document
    CLibrary libDoc = CLibrary::LoadFromFile(_T("Library1.xml"));

    // get the first (and only) root element <Library>
```

```
      CLibraryType lib = libDoc.Library2.first();

      // it is possible to 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 << "ISBN: " << tstring(itBook->ISBN) << std::endl;
            tcout << "Title: " << tstring(itBook->Title.first()) <<
   std::endl;

            // read and compare an enumeration value
            if (itBook->Variant.GetEnumerationValue() ==
CBookVariant::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 - all references to the document and any of its
   nodes become invalid
      libDoc.DestroyDocument();
   }
```

To access the contents of attributes and elements as strings for the `cout <<` operator, an explicit cast to `tstring` is required.
Of course you can also load a document, modify it by adding or deleting elements and attributes, and then save the changed file back to disk.

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 Metadata

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::SimpleType

Method	Purpose
`operator bool()`	Returns true if this is not the NULL SimpleType
`operator !()`	Returns true if this is the NULL SimpleType
`SimpleType GetBaseType()`	Returns the base type of this type
`string_type GetLocalName()`	Returns the local name of the type
`string_type GetNamespaceURI()`	Returns the namespace URI of the type
`unsigned int GetMinLength()`	Returns the value of this facet
`unsigned int GetMaxLength()`	Returns the value of this facet
`unsigned int GetLength()`	Returns the value of this facet
`unsigned int GetTotalDigits()`	Returns the value of this facet
`unsigned int GetFractionDigits()`	Returns the value of this facet
`string_type GetMinInclusive()`	Returns the value of this facet
`string_type GetMinExclusive()`	Returns the value of this facet
`string_type GetMaxInclusive()`	Returns the value of this facet
`string_type GetMaxExclusive()`	Returns the value of this facet
`std::vector<string_type> GetEnumerations()`	Returns a list of all enumeration facets
`std::vector<string_type> GetPatterns()`	Returns a list of all pattern facets
`WhitespaceType GetWhitespace()`	Returns the value of the whitespace facet, which is one of: Whitespace_Unknown Whitespace_Preserve Whitespace_Replace Whitespace_Collapse

altova::meta::ComplexType

Method	Purpose
`operator bool()`	Returns true if this is not the NULL ComplexType
`operator !()`	Returns true if this is the NULL ComplexType

Method	Purpose
`std::vector<Attribute> GetAttributes()`	Returns a list of all attributes
`std::vector<Element> GetElements()`	Returns a list of all elements
`ComplexType GetBaseType()`	Returns the base type of this type
`string_type GetLocalName()`	Returns the local name of the type
`string_type GetNamespaceURI()`	Returns the namespace URI of the type
`SimpleType GetContentType()`	Returns the simple type of the content
`Element FindElement(const char_type* localName, const char_type* namespaceURI)`	Finds the element with the specified local name and namespace URI
`Attribute FindAttribute(const char_type* localName, const char_type* namespaceURI)`	Finds the attribute with the specified local name and namespace URI

altova::meta::Element

Method	Purpose
`operator bool()`	Returns true if this is not the NULL Element
`operator !()`	Returns true if this is the NULL Element
`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.
`unsigned int GetMinOccurs()`	Returns the minOccurs value defined in the schema
`unsigned int GetMaxOccurs()`	Returns the maxOccurs value defined in the schema
`string_type GetLocalName()`	Returns the local name of the element
`string_type GetNamespaceURI()`	Returns the namespace URI of the element

altova::meta::Attribute

Method	Purpose
`operator bool()`	Returns true if this is not the NULL Attribute

Method	Purpose
`operator !()`	Returns true if this is the NULL Attribute
`SimpleType GetDataType()`	Returns the type of the attribute content
`bool IsRequired()`	Returns true if the attribute is required
`string_type GetLocalName()`	Returns the local name of the attribute
`string_type GetNamespaceURI()`	Returns the namespace URI of the attribute

16.7.4 Using the generated C# library

Generated Libraries

Name	Purpose
`Altova`	Base library containing common runtime support, identical for every schema
`AltovaXML`	Base library containing runtime support for XML, identical for every schema
YourSchema	Library containing declarations generated from the input schema, named as the schema file or DTD
YourSchema`Test`	Test application skeleton (XMLSpy only)

DOM Implementation

The generated C# code uses the .NET standard System.XML library as the underlying DOM implementation.

Data Types

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`	

XML Schema	C#	Remarks
xs:unsignedLong	ulong	
xs:int	int	
xs:unsignedInt	uint	
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	Altova.Types.DateTime	See DateTime and Duration Classes
xs:duration	Altova.Types.Duration	See DateTime and Duration Classes
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 Document Class

In addition to the classes for the types declared in the XML Schema, the document class is generated. It contains all possible root elements as members, and in addition the following methods ("Doc" stands for the name of the generated document class itself):

Method	Purpose
static Doc LoadFromFile(string fileName)	Loads an XML document from a file
static Doc LoadFromString(string xml)	Loads an XML document from a string
static Doc LoadFromBinary(byte[] xml)	Loads an XML document from a byte array
void SaveToFile(string fileName, bool prettyPrint, string encoding = "UTF-8")	Saves an XML document to a file with the specified encoding. If no encoding is specified, UTF-8 is assumed.
void SaveToFile(string fileName, bool prettyPrint, string encoding, string lineend)	Saves an XML document to a file with the specified encoding and the specified lineend.
string SaveToString(bool prettyPrint)	Saves an XML document to a string
byte[] SaveToBinary(bool prettyPrint, string encoding = "UTF-8")	Saves an XML document to a byte array with the specified encoding. If no encoding is specified, UTF-8 is assumed.
static Doc CreateDocument()	Creates a new, empty XML document
static Doc CreateDocument(string	Creates a new, empty XML document, with

Method	Purpose
`encoding)`	encoding of type "encoding".
`void SetSchemaLocation(string schemaLocation)`	Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.
`void SetDTDLocation(string dtdLocation)`	Adds a DOCTYPE declaration with the specified system ID. A root element must already exist.

Generated Type Class

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 and mixed types, a **Value** property is generated to access the text content. For simple types with enumeration facets, the **EnumerationValue** property can be used together with generated constants for each enumeration value. In addition, the property **StaticInfo**() allows accessing of schema information as **Altova.Xml.Meta.SimpleType** or **Altova.Xml.Meta.ComplexType** (see below).

XML Schema derivation of complexTypes by extension results in derived classes in the generated library.

Generated Member Attribute Class

For each member attribute of a type, a class with the following methods or properties is created.

Method/Property	Purpose
`bool Exists()`	Returns true if the attribute exists
`void Remove()`	Removes the attribute from its parent element
`AttributeType Value`	Sets or gets the attribute value
`int EnumerationValue`	Generated for enumeration types only. Sets or gets the attribute value using one of the constants generated for the possible values, reading 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 below)

Generated Member Element Class

For each member element of a type, a class with the following methods or properties is created. The class implements the standard System.Collections.IEnumerable interface, so it can be used with the *foreach* statement.

Method/Property	Purpose
`MemberType First`	Returns the first instance of the member element
`MemberType this[int index]`	Returns the member element specified by the index
`MemberType At(int index)`	Returns the member element specified by the index
`MemberType Last`	Returns the last instance of the member element
`MemberType Append()`	Creates a new element and appends it to its parent
`bool Exists`	Returns true if at least one element exists
`int Count`	Returns the count of elements
`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
`System.Collections.IEnumerator GetEnumerator()`	Returns an object for iterating instances of the member element.
`MemberType Value`	Sets or gets the element content (only generated if element can have mixed or simple content)
`int EnumerationValue`	Generated for enumeration types only. Sets or gets the element value using one of the constants generated for the possible values, reading returns **Invalid** if the value does not match any of the enumerated values in the schema.
`Altova.Xml.Meta.Element Info`	Returns an object for querying schema information (see below)

Creating XML Documents

The generated code library makes it very easy to create XML files. Let's start with an example XML file for the example schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<Library xsi:schemaLocation="http://www.nanonull.com/LibrarySample
Library.xsd" xmlns="http://www.nanonull.com/LibrarySample" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <Book ISBN="0764549642" Variant="Paperback">
      <Title>The XML Spy Handbook</Title>
      <Author>Altova</Author>
   </Book>
</Library>
```

The following C# code was used to create this file. The classes Library2, LibraryType and BookType are generated out of the schema and represent the complete document, the type of the <Library> element, and the complex type BookType, which is the type of the <Book> element. Note that the automatic name de-collision renamed the Library class to Library2 to avoid a possible conflict with the library namespace name.

```
   using Library;
   ...
      protected static void Example()
      {
            // create a new, empty XML document
            Library2 libDoc = Library2.CreateDocument();

            // create the root element <Library> and add it to the document
            LibraryType lib = libDoc.Library3.Append();

            // create a new <Book> and add it to the library
            BookType book = lib.Book.Append();

            // set the ISBN attribute of the book
            book.ISBN.Value = "0764549642";

            // set the Variant attribute of the book using an enumeration
constant
            book.Variant.EnumerationValue =
BookVariant.EnumValues.ePaperback;

            // add the <Title> and <Author> elements, and set values
            book.Title.Append().Value = "The XML Spy Handbook";
            book.Author.Append().Value = "Altova";

            // 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);
      }
```

Note that all elements are created by calling Append(), and that attributes (like ISBN in the example) can simply be accessed like structure members.

Reading XML Documents

The following example reads the file we just created and demonstrates the various methods for reading XML files:

```
   using Library;
   ...
      protected static void Example()
      {
            // load XML document
            Library2 libDoc = Library2.LoadFromFile("Library1.xml");

            // get the first (and only) root element <Library>
            LibraryType lib = libDoc.Library3.First;

            // it is possible to check whether an element exists:
            if (!lib.Book.Exists)
            {
                  Console.WriteLine("This library is empty.");
                  return;
            }

            // iteration: for each <Book>...
            foreach (BookType book in lib.Book)
            {
                  // output values of ISBN attribute and (first and only)
title element
                  Console.WriteLine("ISBN: " + book.ISBN.Value);
                  Console.WriteLine("Title: " + book.Title.First.Value);

                  // read and compare an enumeration value
                  if (book.Variant.EnumerationValue ==
BookVariant.EnumValues.ePaperback)
                        Console.WriteLine("This is a paperback book.");

                  // for each <Author>...
                  foreach (xs.stringType author in book.Author)
                        Console.WriteLine("Author: " + author.Value);

                  // alternative: use count and index
                  for (int j = 0; j < book.Author.Count; ++j)
                        Console.WriteLine("Author: " +
book.Author[j].Value);
            }
      }
```

Note the type of the book.Author member: xs.stringType is a generated class for any element

with simple string content. Similar classes are generated for all other XML Schema types.

Of course you can also load a document, modify it by adding or deleting elements and attributes, and then save the changed file back to disk.

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 Metadata

The generated library allows accessing static schema information via the following classes. The properties that return one of the metadata classes return null if the respective property does not exist.

Altova.Xml.Meta.SimpleType

Method/Property	Purpose
`SimpleType BaseType`	Returns the base type of this type
`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
`int MinLength`	Returns the value of this facet
`int MaxLength`	Returns the value of this facet
`int Length`	Returns the value of this facet
`int TotalDigits`	Returns the value of this facet
`int FractionDigits`	Returns the value of this facet
`string MinInclusive`	Returns the value of this facet
`string MinExclusive`	Returns the value of this facet
`string MaxInclusive`	Returns the value of this facet
`string MaxExclusive`	Returns the value of this facet
`string[] Enumerations`	Returns a list of all enumeration facets
`string[] Patterns`	Returns a list of all pattern facets
`WhitespaceType Whitespace`	Returns the value of the whitespace facet, which is one of: Unknown Preserve Replace Collapse

Altova.Xml.Meta.ComplexType

Method/Property	Purpose
`Attribute[] Attributes`	Returns a list of all attributes
`Element[] Elements`	Returns a list of all elements
`ComplexType BaseType`	Returns the base type of this type
`string LocalName`	Returns the local name of the type
`string NamespaceURI`	Returns the namespace URI of the type

Method/Property	Purpose
`XmlQualifiedName QualifiedName`	Returns the qualified name of this type
`SimpleType ContentType`	Returns the simple type of the content
`Element FindElement(string localName, string namespaceURI)`	Finds the element with the specified local name and namespace URI
`Attribute FindAttribute(string localName, string namespaceURI)`	Finds the attribute with the specified local name and namespace URI

Altova.Xml.Meta.Element

Method/Property	Purpose
`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.
`int MinOccurs`	Returns the minOccurs value defined in the schema
`int MaxOccurs`	Returns the maxOccurs value defined in the schema
`string LocalName`	Returns the local name of the element
`string NamespaceURI`	Returns the namespace URI of the element
`XmlQualifiedName QualifiedName`	Returns the qualified name of the element

Altova.Xml.Meta.Attribute

Method/Property	Purpose
`SimpleType DataType`	Returns the type of the attribute content
`bool Required()`	Returns true if the attribute is required
`string LocalName`	Returns the local name of the attribute
`string NamespaceURI`	Returns the namespace URI of the attribute
`XmlQualifiedName QualifiedName`	Returns the qualified name of the attribute

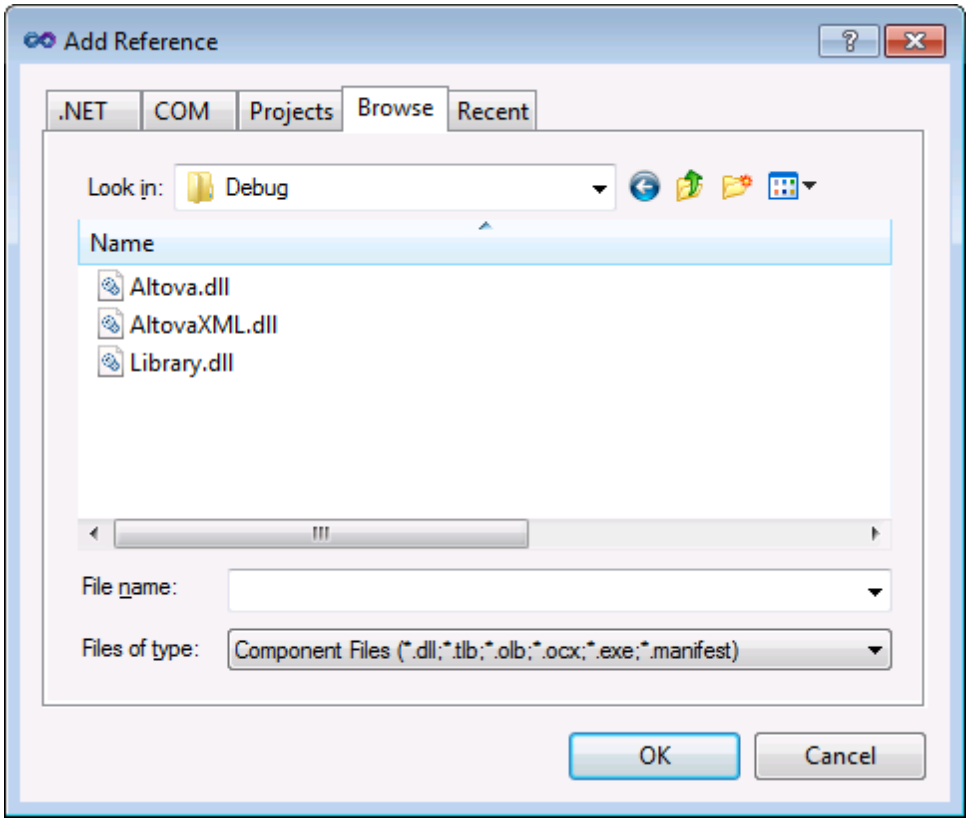
16.7.5 Integrating Altova libraries into existing projects

To use the Altova libraries in your custom project, refer to the libraries from your project (or include them into your project), as shown below for each language.

C#

To integrate the Altova libraries into an existing C# project:

1. After MapForce generates code from a schema (for example, **YourSchema.xsd**), build the generated **YourSchema.sln** solution in Visual Studio. This solution is in a project folder with the same name as the schema.
2. Right-click your existing project in Visual Studio, and select **Add Reference**.
3. On the Browse tab, browse for the following libraries: **Altova.dll**, **AltovaXML.dll**, and **YourSchema.dll** located in the **bin\Debug** directory of the project generated from your schema.



C++

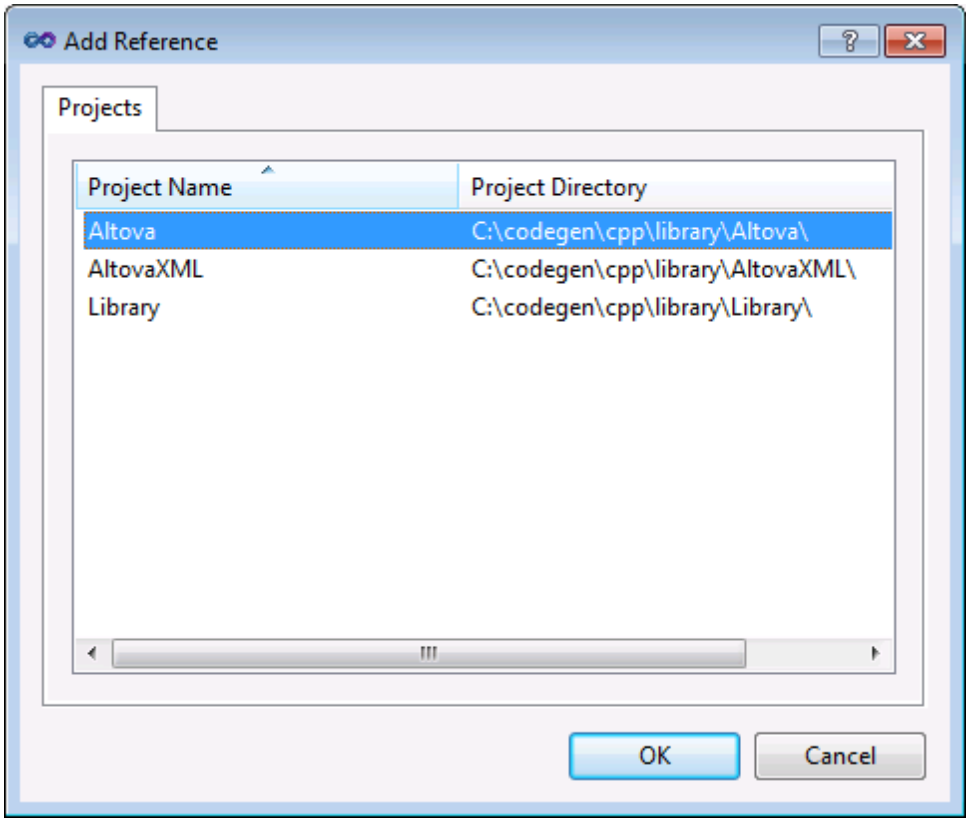
The easiest way to integrate the libraries into an existing C++ project is to add the generated project files to your solution. For example, let's assume that you generated code from a schema called **Library.xsd** and selected **c:\codegen\cpp\library** as target directory. The generated

libraries in this case are available at:

- c:\codegen\cpp\library\Altova\vcxproj
- c:\codegen\cpp\library\AltovaXML\AltovaXML.vcxproj
- c:\codegen\cpp\library\Library.vcxproj

First, open the generated **c:\codegen\cpp\library\Library.sln** solution and build it in Visual Studio.

Next, open your existing Visual Studio solution (in Visual Studio 2010, in this example), right-click it, select **Add | Existing Project**, and add the project files listed above, one by one. Be patient while Visual Studio parses the files. Next, right-click your project and select **Properties**. In the Property Pages dialog box, select **Common Properties | Framework and References**, and then click **Add New Reference**. Next, select and add each of the following projects: *Altova*, *AltovaXML*, and *Library*.



See also the MSDN documentation for using functionality from a custom library, as applicable to your version of Visual Studio, for example:

- If you chose to generate static libraries, see https://msdn.microsoft.com/en-us/library/ms235627(v=vs.100).aspx
- If you chose to generate dynamic libraries: https://msdn.microsoft.com/en-us/library/ms235636(v=vs.100).aspx

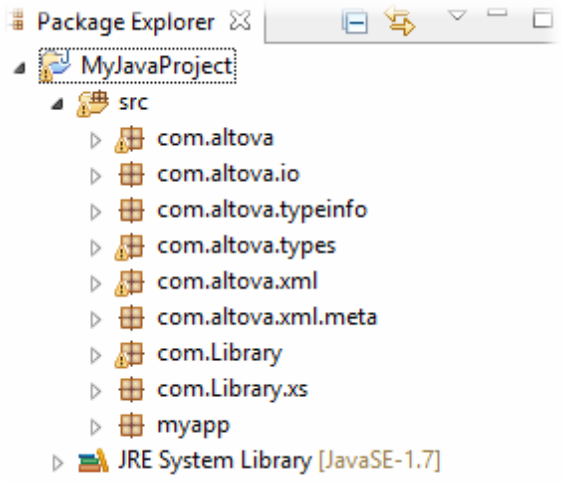
The option to generate static or dynamic libraries is available in code generation options (see Code generator options).

Java

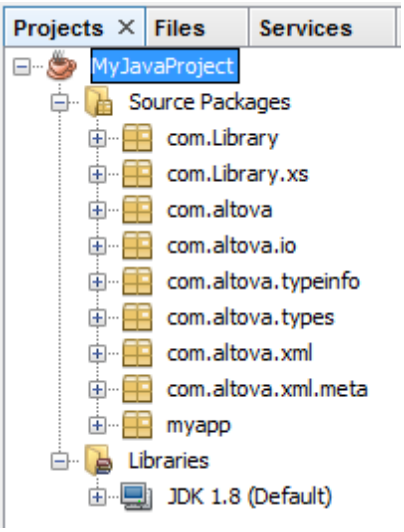
One of the ways to integrate the Altova packages into your Java project is to copy the **com** directory of the generated code to the directory which stores the source packages of your Java project (for example, **C:\Workspace\MyJavaProject\src**). For example, let's assume that you generated code in **c:\codegen\java\library**. The generated Altova classes in this case are available at **c:\codegen\java\library\com**.

After copying the libraries, refresh the project. To refresh the project in Eclipse, select it in the Package Explorer, and press **F5**. To refresh the project in NetBeans IDE 8.0, select the menu command **Source | Scan for External Changes**.

Once you perform the copy operation, the Altova packages are available in the Package Explorer (in case of Eclipse), or under "Source Packages" in the Projects pane (in case of NetBeans IDE).



Altova packages in Eclipse 4.4



Altova packages in NetBeans IDE 8.0.2

16.8 DateTime and Duration Classes

If your XML instance files use schema types related to time and duration, these are converted to Altova `DateTime` and `Duration` types in the generated code. Specifically, during code generation, the following XML types are converted to the `Altova.Types.DateTime` type:

- `xs:dateTime`
- `date`
- `time`
- `gYearMonth`
- `gYear`
- `gMonthDay`
- `gDay`
- `gMonth`

The `xs:duration` schema type is converted to Altova `Duration` type.

This documentation section provides reference to Altova `DateTime` and `Duration` classes, organized by language (C#, C++, Java). The implementation is slightly different for each language. In general, in any project that references the Altova libraries, the `DateTime` and `Duration` classes enable you to perform type casts, change the timezone, format time values, or work with time durations.

16.8.1 C#

This section includes description of Altova `DateTime` and `Duration` classes applicable to C#.

16.8.1.1 Altova.Types.DateTime

The `Altova.Types.DateTime` class exposes the following public members.

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`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, and second supplied as arguments.

Name	Description
second)	
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.

Name	Description

`static DateTime Parse( string s )`	Creates a `DateTime` object from the string supplied as argument. For example, the following sample string values would be converted successfully to a `DateTime` object:
	
	`2015-01-01T23:23:23`
	`2015-01-01`
	`2015-11`
	`23:23:23`
	
	An exception is raised if the string cannot be converted to a `DateTime` object.
	
	Note that this method is static and can only be called on the `Altova.Types.DateTime` class itself, not on an instance of the class.

`static DateTime Parse(string s, DateTimeFormat format)`	Creates a `DateTime` object from a string, using the format supplied as argument. For the list of possible formats, see Altova.Types.DateTimeFormat. An exception is raised if the string cannot be converted to a `DateTime` object. Note that this method is static and can only be called on the `Altova.Types.DateTime` class itself, not on an instance of the class.
`override string ToString()`	Converts the `DateTime` object to a string.
`string ToString(DateTimeFormat format)`	Converts the `DateTime` object to a string, using the format supplied as argument. For the list of possible formats, see Altova.Types.DateTimeFormat.

Operators

Name	Description
`!=`	Determines if `DateTime` a is not equal to `DateTime` b.
`<`	Determines if `DateTime` a is less than `DateTime` b.
`<=`	Determines if `DateTime` a is less than or equal to `DateTime` b.
`==`	Determines if `DateTime` a is equal to `DateTime` b.
`>`	Determines if `DateTime` a is greater than `DateTime` b.
`>=`	Determines if `DateTime` a is greater than or equal to

Name	Description
	`DateTime` b.

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
using Altova.Types;
```

The following code listing illustrates various ways to create `DateTime` objects:

```
protected static void DateTimeExample1()
{
	// Create a DateTime object from the current system time
	Altova.Types.DateTime dt = new
Altova.Types.DateTime(System.DateTime.Now);
	Console.WriteLine("The current time is: " + dt.ToString());

	// Create an Altova DateTime object from parts (no timezone)
	Altova.Types.DateTime dt1 = new Altova.Types.DateTime(2015, 10,
12, 10, 50, 33);
	Console.WriteLine("My custom time is : " + dt1.ToString());

	// Create an Altova DateTime object from parts (with UTC+60
minutes timezone)
	Altova.Types.DateTime dt2 = new Altova.Types.DateTime(2015, 10,
12, 10, 50, 33, 60);
	Console.WriteLine("My custom time with timezone is : " +
dt2.ToString());

	// Create an Altova DateTime object by parsing a string
	Altova.Types.DateTime dt3 = Altova.Types.DateTime.Parse("2015-01-
01T23:23:23");
	Console.WriteLine("Time created from string: " + dt3.ToString());

	// Create an Altova DateTime object by parsing a string formatted
as schema date
	Altova.Types.DateTime dt4 = Altova.Types.DateTime.Parse("2015-01-
01", DateTimeFormat.W3_date);
	Console.WriteLine("Time created from string formatted as schema
date: " + dt4.ToString());
}
```

The following code listing illustrates various ways to format `DateTime` objects:

```
protected static void DateTimeExample2()
{
	// Create a DateTime object from the current system time
	Altova.Types.DateTime dt = new
Altova.Types.DateTime(System.DateTime.Now);
```

```
        // Output the unformatted DateTime
        Console.WriteLine("Unformatted time: " + dt.ToString());

        // Output this DateTime formatted using various formats
        Console.WriteLine("S_DateTime:       " +
dt.ToString(DateTimeFormat.S_DateTime));
        Console.WriteLine("S_Days:           " +
dt.ToString(DateTimeFormat.S_Days));
        Console.WriteLine("S_Seconds:        " +
dt.ToString(DateTimeFormat.S_Seconds));
        Console.WriteLine("W3_date:          " +
dt.ToString(DateTimeFormat.W3_date));
        Console.WriteLine("W3_dateTime:      " +
dt.ToString(DateTimeFormat.W3_dateTime));
        Console.WriteLine("W3_gDay:          " +
dt.ToString(DateTimeFormat.W3_gDay));
        Console.WriteLine("W3_gMonth:        " +
dt.ToString(DateTimeFormat.W3_gMonth));
        Console.WriteLine("W3_gMonthDay:     " +
dt.ToString(DateTimeFormat.W3_gMonthDay));
        Console.WriteLine("W3_gYear:         " +
dt.ToString(DateTimeFormat.W3_gYear));
        Console.WriteLine("W3_gYearMonth:    " +
dt.ToString(DateTimeFormat.W3_gYearMonth));
        Console.WriteLine("W3_time:          " +
dt.ToString(DateTimeFormat.W3_time));
}
```

16.8.1.2 Altova.Types.Duration

The `Altova.Types.Duration` class exposes the following public members.

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`	Initializes a new instance of the `Duration` class to a duration built from parts supplied as arguments.

Name	Description
partseconds, bool bnegative)	

Properties

Name	Description
int Months	Gets or sets the number of months of the current instance of Duration.
System.TimeSpan Value	Gets or sets the value (as System.TimeSpan) of the current instance of Duration.
int Years	Gets or sets the number of years of the current instance of Duration.

Methods

Name	Description
override bool Equals(object other)	Returns **true** if the specified object is equal to the current object; **false** otherwise.
override int GetHashCode()	Returns the hash code of the current instance.
bool IsNegative()	Returns **true** if the current instance of Duration represents a negative duration.
static Duration Parse(string s, ParseType pt)	Returns an Altova.Types.Duration object parsed from the string supplied as argument, using the parse type supplied as argument. Valid parse type values: **DURATION** Parse duration assuming that year, month, day, as well as time duration parts exist. **YEARMONTH** Parse duration assuming that only year and month parts exist. **DAYTIME** Parse duration assuming that only the day and time parts exist. Note that this method is static and can only be called on the class itself, not on an instance of the class.
override string ToString()	Converts the current Duration instance to string. For example, a time span of 3 hours, 4 minutes, and 5 seconds would be converted to "PT3H4M5S".

Name	Description
`string ToYearMonthString()`	Converts the current `Duration` instance to string, using the "Year and Month" parse type.

Operators

Name	Description
`!=`	Determines if `Duration` a is not equal to `Duration` b.
`==`	Determines if `Duration` a is equal to `Duration` b.

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
using Altova.Types;
```

The following code listing illustrates various ways to create `Duration` objects:

```
protected static void DurationExample1()
{
	// Create a new time span of 3 hours, 4 minutes, and 5 seconds
	System.TimeSpan ts = new TimeSpan(3, 4, 5);
	// Create a Duration from the time span
	Duration dr = new Duration(ts);
	// The output is: PT3H4M5S
	Console.WriteLine("Duration created from TimeSpan: " +
dr.ToString());

	// Create a negative Altova.Types.Duration from 6 years, 5
months, 4 days, 3 hours,
	// 2 minutes, 1 second, and .33 of a second
	Duration dr1 = new Duration(6, 5, 4, 3, 2, 1, .33, true);
	// The output is: -P6Y5M4DT3H2M1.33S
	Console.WriteLine("Duration created from parts: " +
dr1.ToString());

	// Create a Duration from a string using the DAYTIME parse type
	Duration dr2 = Altova.Types.Duration.Parse("-P4DT3H2M1S",
Duration.ParseType.DAYTIME);
	// The output is -P4DT3H2M1S
	Console.WriteLine("Duration created from string: " +
dr2.ToString());

	// Create a duration from ticks
	Duration dr3 = new Duration(System.DateTime.UtcNow.Ticks);
	// Output the result
	Console.WriteLine("Duration created from ticks: " +
```

```
dr3.ToString());
}
```

The following code listing illustrates getting values from `Duration` objects:

```
protected static void DurationExample2()
{
	// Create a negative Altova.Types.Duration from 6 years, 5
months, 4 days, 3 hours,
	// 2 minutes, 1 second, and .33 of a second
	Duration dr = new Duration(6, 5, 4, 3, 2, 1, .33, true);
	// The output is: -P6Y5M4DT3H2M1.33S
	Console.WriteLine("The complete duration is: " + dr.ToString());

	// Get only the year and month part as string
	string dr1 = dr.ToYearMonthString();
	Console.WriteLine("The YEARMONTH part is: " + dr1);

	// Get the number of years in duration
	Console.WriteLine("Years: " + dr.Years);

	// Get the number of months in duration
	Console.WriteLine("Months: " + dr.Months);
}
```

16.8.1.3 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.	`735913.631897345108796296293 63`
S_Seconds	Formats the value as number of seconds elapsed since the UNIX epoch, with a precision of a ten-millionth of a second.	`63582937678.0769062`
W3_date	Formats the value as schema date.	`2015-11-12`
W3_dateTime	Formats the value as schema	`2015-11-12T15:12:14.5194251`

Value	Description	Example
	dateTime.	
W3_gDay	Formats the value as schema gDay.	`---12` (assuming that the date is 12th of the month)
W3_gMonth	Formats the value as schema gMonth.	`--11` (assuming that the month is November)
W3_gMonthDay	Formats the value as schema gMonthDay.	`--11-12` (assuming that the date is 12th of November)
W3_gYear	Formats the value as schema gYear.	`2015` (assuming that the year is 2015)
W3_gYearMonth	Formats the value as schema gYearMonth.	`2015-11` (assuming that the year is 2015 and the month is November)
W3_time	Formats the value as schema time, with a precision of a ten-millionth of a second.	`15:19:07.5582719`

16.8.2 C++

This section includes description of Altova `DateTime` and `Duration` classes applicable to C++.

16.8.2.1 altova::DateTime

The `Altova::DateTime` class exposes the following public members.

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,`	Initializes a new instance of the `DateTime` class to the year, month, day, hour, minute, and second supplied as argument.

Name	Description
`unsigned char minute, double second)`	
`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.

Name	Description
`__int64 Value() const`	Returns the value of the `DateTime` object, expressed in the number of ticks (100-nanosecond intervals) that have elapsed since 12:00:00 midnight, January 1, 0001.
`int Weekday() const`	Returns the day of week of the current `DateTime` object, as an integer. Values range from 0 through 6, where 0 is Monday (ISO-8601).
`int Weeknumber() const`	Returns the number of week in the year of the current `DateTime` object. The return values are according to ISO-8601.
`int WeekOfMonth() const`	Returns the number of week in the month of the current `DateTime` object. The return values are according to ISO-8601.
`int Year() const`	Returns the year of the current `DateTime` object.

Example

```
void Example()
{
	// initialize a new DateTime instance to 12:00:00 midnight,
January 1st, 0001
	altova::DateTime dt1 = altova::DateTime();

	// initialize a new DateTime instance using the year, month, day,
hour, minute, and second
	altova::DateTime dt2 = altova::DateTime(2015, 11, 10, 9, 8, 7);

	// initialize a new DateTime instance using the year, month, day,
hour, minute, second, and UTC +01:00 timezone
	 altova::DateTime dt = altova::DateTime(2015, 11, 22, 13, 53, 7,
60);

	// Get the value of this DateTime object
	std::cout << "The number of ticks of the DateTime object is: " <<
dt.Value() << std::endl;

	// Get the year
	cout << "The year is: " << dt.Year() << endl;
	// Get the month
	cout << "The month is: " << (int)dt.Month() << endl;
	// Get the day of the month
	cout << "The day of the month is: " << (int) dt.Day() << endl;
	// Get the day of the year
	cout << "The day of the year is: " << dt.DayOfYear() << endl;
	// Get the hour
	cout << "The hour is: " << (int) dt.Hour() << endl;
```

```
	// Get the minute
	cout << "The minute is: " << (int) dt.Minute() << endl;
	// Get the second
	cout << "The second is: " << dt.Second() << endl;
	// Get the weekday
	cout << "The weekday is: " << dt.Weekday() << endl;
	// Get the week number
	cout << "The week of year is: " << dt.Weeknumber() << endl;
	// Get the week in month
	cout << "The week of month is: " << dt.WeekOfMonth() << endl;

	// Check whether a DateTime instance has a timezone
	if (dt.HasTimezone() == TRUE)
	{
		// output the value of the Timezone
		cout <<  "The timezone is: " << dt.Timezone() << endl;
	}
	else
	{
		cout <<  "No timezone has been defined." << endl;
	}

	// Construct a DateTime object with a timezone UTC+01:00 (Vienna)
	altova::DateTime vienna_dt = DateTime(2015, 11, 23, 14, 30, 59,
+60);
	// Output the result in readable format
	cout << "The Vienna time: "
		<< (int) vienna_dt.Month()
		<< "-" << (int) vienna_dt.Day()
		<< " " << (int) vienna_dt.Hour()
		<< ":" << (int) vienna_dt.Minute()
		<< ":" << (int) vienna_dt.Second()
		<< endl;

	// Convert the value to UTC time
	DateTime utc_dt = DateTime(vienna_dt.NormalizedValue());
	// Output the result in readable format
	cout << "The UTC time:    "
		<< (int) utc_dt.Month()
		<< "-" << (int) utc_dt.Day()
		<< " " << (int) utc_dt.Hour()
		<< ":" << (int) utc_dt.Minute()
		<< ":" << (int) utc_dt.Second()
		<< endl;

	// Check if a year is a leap year
	int year = 2016;
	if( altova::DateTime::IsLeapYear(year) )
	{ cout << year << " is a leap year" << endl; }
	else
	{ cout << year << " is not a leap year" << endl; }
}
```

16.8.2.2 altova::Duration

The `altova::Duration` class exposes the following public members.

Constructors

Name	Description
`Duration()`	Initializes a new instance of the `Duration` class to an empty value.
`Duration(const DayTimeDuration& dt)`	Initializes a new instance of the `Duration` class to a duration defined by the `dt` argument (see altova::DayTimeDuration).
`Duration(const YearMonthDuration& ym)`	Initializes a new instance of the `Duration` class to the duration defined by the `ym` argument (see altova::YearMonthDuration).
`Duration(const YearMonthDuration& ym, const DayTimeDuration& dt)`	Initializes a new instance of the `Duration` class to the duration defined by both the `dt` and the `ym` arguments (see altova::YearMonthDuration and altova::DayTimeDuration).

Methods

Name	Description
`int Days() const`	Returns the number of days in the current `Duration` instance.
`DayTimeDuration DayTime() const`	Returns the day and time duration in the current `Duration` instance, expressed as a `DayTimeDuration` object (see altova::DayTimeDuration).
`int Hours() const`	Returns the number of hours in the current `Duration` instance.
`bool IsNegative() const`	Returns Boolean **true** if the current `Duration` instance is negative.
`bool IsPositive() const`	Returns Boolean **true** if the current `Duration` instance is positive.
`int Minutes() const`	Returns the number of minutes in the current `Duration` instance.
`int Months() const`	Returns the number of months in the current `Duration` instance.

Name	Description
`double Seconds() const`	Returns the number of seconds in the current `Duration` instance.
`YearMonthDuration YearMonth() const`	Returns the year and month duration in the current `Duration` instance, expressed as a `YearMonthDuration` object (see altova::YearMonthDuration).
`int Years() const`	Returns the number of years in the current `Duration` instance.

Example

The following code listing illustrates creating a new `Duration` object, as well as reading values from it.

```
void ExampleDuration()
{
	// Create an empty Duration object
	altova::Duration empty_duration = altova::Duration();

	// Create a Duration object using an existing duration value
	altova::Duration duration1 = altova::Duration(empty_duration);

	// Create a YearMonth duration of six years and five months
	altova::YearMonthDuration yrduration =
altova::YearMonthDuration(6, 5);

	// Create a DayTime duration of four days, three hours, two
minutes, and one second
	altova::DayTimeDuration dtduration = altova::DayTimeDuration(4,
3, 2, 1);

	// Create a Duration object by combining the two previously
created durations
	altova::Duration duration = altova::Duration(yrduration,
dtduration);

	// Get the number of years in this Duration instance
	cout << "Years:  " << duration.Years() << endl;

	// Get the number of months in this Duration instance
	cout << "Months: " << duration.Months() << endl;

	// Get the number of days in this Duration instance
	cout << "Days:   " << duration.Days() << endl;

	// Get the number of hours in this Duration instance
	cout << "Hours:  " << duration.Hours() << endl;

	// Get the number of hours in this Duration instance
```



```
        cout << "Minutes: " << duration.Minutes() << endl;

        // Get the number of seconds in this Duration instance
        cout << "Seconds: " << duration.Seconds() << endl;
}
```

16.8.2.3 altova::DayTimeDuration

The `altova::DayTimeDuration` class exposes the following public members.

Constructors

Name	Description
`DayTimeDuration()`	Initializes a new instance of the `DayTimeDuration` class to an empty value.
`DayTimeDuration(int days, int hours, int minutes, double seconds)`	Initializes a new instance of the `DayTimeDuration` class to the number of days, hours, minutes, and seconds supplied as arguments.
`explicit DayTimeDuration(__int64 value)`	Initializes a new instance of the `DayTimeDuration` class to a duration that consists of as many ticks (100-nanosecond intervals) as supplied in the **value** argument.

Methods

Name	Description
`int Days() const`	Returns the number of days in the current `DayTimeDuration` instance.
`int Hours() const`	Returns the number of hours in the current `DayTimeDuration` instance.
`bool IsNegative() const`	Returns Boolean **true** if the current `DayTimeDuration` instance is negative.
`bool IsPositive() const`	Returns Boolean **true** if the current `DayTimeDuration` instance is positive.
`int Minutes() const`	Returns the number of minutes in the current `DayTimeDuration` instance.
`double Seconds() const`	Returns the number of seconds in the current `DayTimeDuration` instance.

`__int64 Value() const`	Returns the value (in ticks) of the current `DayTimeDuration` instance.

16.8.2.4 altova::YearMonthDuration

The `altova::YearMonthDuration` class exposes the following public members.

Constructors

Name	Description
`YearMonthDuration()`	Initializes a new instance of the `YearMonthDuration` class to an empty value.
`YearMonthDuration(int years, int months)`	Initializes a new instance of the `YearMonthDuration` class to the number of years and months supplied in the **years** and **months** arguments.
`explicit YearMonthDuration(int value)`	Initializes a new instance of the `YearMonthDuration` class to a duration that consists of as many ticks (100-nanosecond intervals) as supplied in the **value** argument.

Methods

Name	Description
`bool IsNegative() const`	Returns Boolean **true** if the current `YearMonthDuration` instance is negative.
`bool IsPositive() const`	Returns Boolean **true** if the current `YearMonthDuration` instance is positive.
`int Months() const`	Returns the number of months in the current `YearMonthDuration` instance.
`int Value() const`	Returns the value (in ticks) of the current `YearMonthDuration` instance.
`int Years()`	Returns the number of years in the current `YearMonthDuration` instance.

16.8.3 Java

This section includes description of Altova `DateTime` and `Duration` classes applicable to Java. These classes exist in the `com.altova.types` package.

16.8.3.1 com.altova.types.DateTime

The `DateTime` class exposes the following public members.

Constructors

Name	Description
`public DateTime()`	Initializes a new instance of the `DateTime` class to an empty value.
`public DateTime(DateTime newvalue)`	Initializes a new instance of the `DateTime` class to the `DateTime` value supplied as argument.
`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.
`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.
`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.
`public DateTime(Calendar newvalue)`	Initializes a new instance of the `DateTime` class to the `java.util.Calendar` value supplied as argument.

Methods

Name	Description
`static DateTime now()`	Returns the current time as a `DateTime` object.

`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`

Name	Description
`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: **0** A timezone offset is not defined. **1** The timezone is UTC. **2** A timezone offset has been defined.
`void setDay( int nDay )`	Sets the day of the current `DateTime` instance to the value supplied as argument.
`void setHasTimezone( int nHasTZ )`	Sets the timezone information of the current `DateTime` instance to the value supplied as argument. This method can be used to strip the timezone information or set the timezone to UTC (Coordinated Universal Time). Valid values for the `nHasTZ` argument: **0** Set the timezone offset to undefined.

Name	Description
	1 Set the timezone to UTC. 2 If the current object has a timezone offset, leave it unchanged.
`void setHour( int nHour )`	Sets the hour of the current `DateTime` instance to the value supplied as argument.
`void setMinute( int nMinute )`	Sets the minute of the current `DateTime` instance to the value supplied as argument.
`void setMonth( int nMonth )`	Sets the month of the current `DateTime` instance to the value supplied as argument.
`void setPartSecond( double nPartSecond )`	Sets the fractional part of the second of the current `DateTime` instance to the value supplied as argument.
`void setSecond( int nSecond )`	Sets the second of the current `DateTime` instance to the value supplied as argument.
`void setTimezoneOffset( int nOffsetTZ )`	Sets the timezone offset of the current `DateTime` instance to the value supplied as argument. The value `nOffsetTZ` must be an integer (positive or negative) and must be expressed in minutes.
`void setYear( int nYear )`	Sets the year of the current `DateTime` instance to the value supplied as argument.
`String toString()`	Returns the string representation of the current `DateTime` instance, for example: `2015-11-24T15:50:56.968+01:00`

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
import com.altova.types.*;
```

The following code listing illustrates various ways to create `DateTime` objects:

```
protected static void DateTimeExample1()
{
	// Initialize a new instance of the DateTime class to the current
time
	DateTime dt = new DateTime(DateTime.now());
	System.out.println("DateTime created from current date and time:
" + dt.toString());
```

```
	// Initialize a new instance of the DateTime class by supplying
the parts
	DateTime dt1 = new DateTime(2015, 11, 23, 14, 30, 24, .459);
	System.out.println("DateTime from parts (no timezone): " +
dt1.toString());

	// Initialize a new instance of the DateTime class by supplying
the parts
	DateTime dt2 = new DateTime(2015, 11, 24, 14, 30, 24, .459, -60);
	System.out.println("DateTime from parts (with negative timezone):
" + dt2.toString());

	// Initialize a new instance of the DateTime class by parsing a
string value		DateTime dt3 = DateTime.parse("2015-11-
24T12:54:47.969+01:00");
	System.out.println("DateTime parsed from string: " +
dt3.toString());
}
```

The following code listing illustrates getting values from `DateTime` objects:

```
protected static void DateTimeExample2()
{
	// Initialize a new instance of the DateTime class to the current
time
	DateTime dt = new DateTime(DateTime.now());

	// Output the formatted year, month, and day of this DateTime
instance
	String str1 = String.format("Year: %d; Month: %d; Day: %d;",
dt.getYear(), dt.getMonth(), dt.getDay());
	System.out.println(str1);

	// Output the formatted hour, minute, and second of this DateTime
instance
	String str2 = String.format("Hour: %d; Minute: %d; Second: %d;",
dt.getHour(), dt.getMinute(), dt.getSecond());
	System.out.println(str2);

	// Return the timezone (in minutes) of this DateTime instance
	System.out.println("Timezone: " + dt.getTimezoneOffset());

	// Get the DateTime as a java.util.Calendar value
	java.util.Calendar dt_java = dt.getValue();
	System.out.println("" + dt_java.toString());

	// Return the day of week of this DateTime instance
	System.out.println("Weekday: " + dt.getWeekday());

	// Check whether the DateTime instance has a timezone defined
	switch(dt.hasTimezone())
	{
```

```
                case 0:
                        System.out.println("No timezone.");
                        break;
                case 1:
                        System.out.println("The timezone is UTC.");
                        break;
                case 2:
                        System.out.println("This object has a timezone.");
                        break;
                default:
                        System.out.println("Unable to determine whether a
timezone is defined.");
                        break;
        }
}
```

The following code listing illustrates changing the timezone offset of a `DateTime` object:

```
protected static void DateTimeExample3()
{
        // Create a new DateTime object with timezone -0100 UTC
        DateTime dt = new DateTime(2015, 11, 24, 14, 30, 24, .459, -60);
        // Output the value before the change
        System.out.println("Before: " + dt.toString());
        // Change the offset to +0100 UTC
        dt.setTimezoneOffset(60);
        // Output the value after the change
        System.out.println("After:  " + dt.toString());
}
```

16.8.3.2 com.altova.types.Duration

The `Duration` class exposes the following public members.

Constructors

Name	Description
`Duration(Duration newvalue)`	Initializes a new instance of the `Duration` class to the `Duration` object supplied as argument.
`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
`static Duration getFromDayTime( int newday, int newhour, int newminute, int newsecond, double newpartsecond )`	Returns a `Duration` object created from the number of days, hours, minutes, seconds, and fractional second parts supplied as argument.
`static Duration getFromYearMonth( int newyear, int newmonth )`	Returns a `Duration` object created from the number of years and months supplied as argument.
`static Duration parse( String s )`	Returns a `Duration` object created from the string supplied as argument. For example, the string `-P1Y1M1DT1H1M1.333S` can be used to create a negative duration of one year, one month, one day, one hour, one minute, one second, and 0.333 fractional parts of a second. To create a negative duration, append the minus sign (-) to the string.
`static Duration parse( String s, ParseType pt )`	Returns a `Duration` object created from the string supplied as argument, using a specific parse format. The parse format can be any of the following: **ParseType.DAYTIME** — Must 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** — Must be used when the string **s** consists of any of the following: years, months, days, hours, minutes, seconds, fractional second parts, for example `P1Y1M1DT1H1M1.333S`. **ParseType.YEARMONTH** — Must be used when the string **s** consists of any of the following: years, months. For example: `P3Y2M`.
`int getDay()`	Returns the number of days in the current `Duration` instance.
`long getDayTimeValue()`	Returns the day and time value (in milliseconds) of the current `Duration` instance. Years and months are ignored.

Name	Description
`int getHour()`	Returns the number of hours in the current `Duration` instance.
`int getMillisecond()`	Returns the number of milliseconds in the current `Duration` instance.
`int getMinute()`	Returns the number of minutes in the current `Duration` instance.
`int getMonth()`	Returns the number of months in the current `Duration` instance.
`double getPartSecond()`	Returns the number of fractional second parts in the current `Duration` instance.
`int getSecond()`	Returns the number of seconds in the current `Duration` instance.
`int getYear()`	Returns the number of years in the current `Duration` instance.
`int getYearMonthValue()`	Returns the year and month value (in months) of the current `Duration` instance. Days, hours, seconds, and milliseconds are ignored.
`boolean isNegative()`	Returns Boolean **true** if the current `Duration` instance is positive.
`void setDayTimeValue(long l)`	Sets the duration to the number of milliseconds supplied as argument, affecting only the day and time part of the duration.
`void setNegative( boolean isnegative )`	Converts the current `Duration` instance to a negative duration.
`void setYearMonthValue(int l)`	Sets the duration to the number of months supplied as argument. Only the years and months part of the duration is affected.
`String toString()`	Returns the string representation of the current `Duration` instance, for example: `-P4DT4H4M4.774S`
`String toYearMonthString()`	Returns the string representation of the YearMonth part of the current `Duration` instance, for example: `P1Y2M`

Examples

Before using the following code listings in your program, ensure the Altova types are imported:

```
import com.altova.types.*;
import com.altova.types.Duration.ParseType;
```

The following code listing illustrates various ways to create `Duration` objects:

```
protected static void ExampleDuration()
{
	// Create a negative duration of 1 year, 1 month, 1 day, 1 hour,
1 minute, 1 second,
	// and 0.333 fractional second parts
	Duration dr = new Duration(1, 1, 1, 1, 1, 1, .333, true);

	// Create a duration from an existing Duration object
	Duration dr1 = new Duration(dr);

	// Create a duration of 4 days, 4 hours, 4 minutes, 4
seconds, .774 fractional second parts
	Duration dr2 = Duration.getFromDayTime(4, 4, 4, 4, .774);

	// Create a duration of 3 years and 2 months
	Duration dr3 = Duration.getFromYearMonth(3, 2);

	// Create a duration from a string
	Duration dr4 = Duration.parse("-P4DT4H4M4.774S");

	// Create a duration from a string, using specific parse formats
	Duration dr5 = Duration.parse("-P1Y1M1DT1H1M1.333S",
ParseType.DURATION);
	Duration dr6 = Duration.parse("P3Y2M", ParseType.YEARMONTH);
	Duration dr7 = Duration.parse("-P4DT4H4M4.774S",
ParseType.DAYTIME);
}
```

The following code listing illustrates getting and setting the value of `Duration` objects:

```
protected static void DurationExample2()
{
	// Create a duration of 1 year, 2 month, 3 days, 4 hours, 5
minutes, 6 seconds,
	// and 333 milliseconds
	Duration dr = new Duration(1, 2, 3, 4, 5, 6, .333, false);
	// Output the number of days in this duration
	System.out.println(dr.getDay());

	// Create a positive duration of one year and 333 milliseconds
	Duration dr1 = new Duration(1, 0, 0, 0, 0, 0, .333, false);
	// Output the day and time value in milliseconds
	System.out.println(dr1.getDayTimeValue());

	// Create a positive duration of 1 year, 1 month, 1 day, 1 hour,
1 minute, 1 second,
	// and 333 milliseconds
```

```
        Duration dr2 = new Duration(1, 1, 1, 1, 1, 1, .333, false);
        // Output the year and month value in months
        System.out.println(dr2.getYearMonthValue());

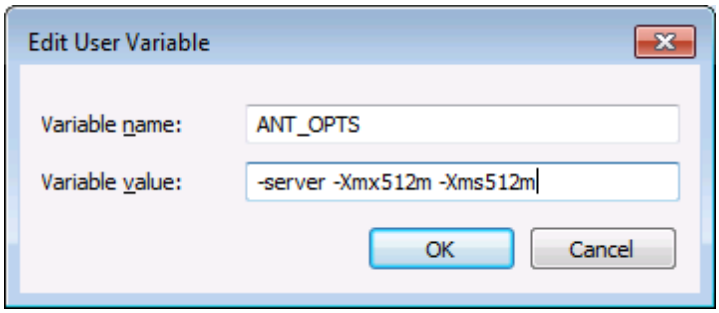
        // Create a positive duration of 1 year and 1 month
        Duration dr3 = new Duration(1, 1, 0, 0, 0, 0, 0, false);
        // Output the value
        System.out.println("The duration is now: " + dr3.toString());
        // Set the DayTime part of duration to 1000 milliseconds
        dr3.setDayTimeValue(1000);
        // Output the value
        System.out.println("The duration is now: " + dr3.toString());
        // Set the YearMonth part of duration to 1 month
        dr3.setYearMonthValue(1);
        // Output the value
        System.out.println("The duration is now: " + dr3.toString());
        // Output the year and month part of the duration
        System.out.println("The YearMonth part of the duration is: " +
dr3.toYearMonthString());
}
```

16.9 Code generation tips

Resolving "Out of memory" exceptions during Java compilation

Complex mappings with large schemas can produce a large amount of code, which might cause a java.lang.OutofMemory exception during compilation using Ant. To rectify this:

- Add the environment variable **ANT_OPTS,** which sets specific Ant options such as the memory to be allocated to the compiler, and add values as shown below.



- To make sure that the compiler and the generated code run in the same process as Ant, change the `fork` attribute, in **build.xml**, to `false`.

You may need to customize the values depending on the amount of memory in your machine and the size of the project you are working with. For more details, see your Java VM documentation.

When running the `ant jar` command, you may get an error message similar to "`[...] archive contains more than 65535 entities`". To prevent this, it is recommended that you use Ant 1.9 or later, and, in the **build.xml** file, add `zip64mode="as-needed"` to the `<jar>` element.

Reserving method names

When customizing code generation using the supplied SPL files, it might be necessary to reserve names to avoid collisions with other symbols. To do this:

1. Navigate to subdirectory corresponding to the programming language of the **spl** subdirectory of the program installation directory e.g. `C:\Program Files\Altova \MapForce2016\spl\java\`.
2. Open the **settings.spl** file and insert a new line into the reserve section e.g. reserve "myReservedWord".
3. Regenerate the program code.

XML Schema support

The following XML Schema constructs are translated into code:

- XML Namespaces: Mapped to namespaces in the target programming language

Simple types:

- Built-in XML Schema types: Mapped to native primitive types or classes in the Altova types library
- Derived by extension
- Derived by restriction
- Facets
- Enumerations
- Patterns

Complex types:

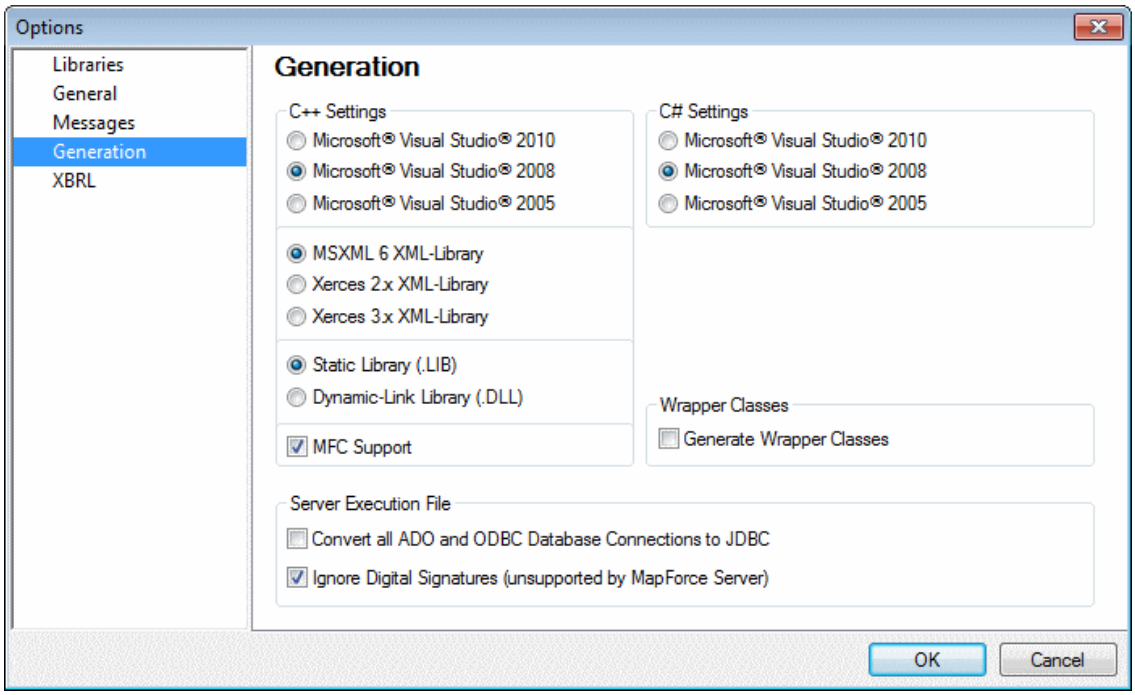
- Built-in anyType node
- User-defined complex types
- Derived by extension: Mapped to derived classes
- Derived by restriction
- Complex content
- Simple content
- Mixed content

The following advanced XML Schema features **are not**, or not fully supported in MapForce, when generating optional **wrapper** classes:

- Wildcards: xs:any and xs:anyAttribute
- Content models (sequence, choice, all): Top-level compositor available in SPL, but not enforced by generated classes
- default and fixed values for attributes: Available in SPL, but not set or enforced by generated classes
- attributes xsi:type, abstract types: SetXsiType methods are generated and need to be called by the user
- Union types: Not all combinations are supported
- Substitution groups: Partial support (resolved like "choice")
- attribute nillable="true" and xsi:nil
- uniqueness constraints
- key and keyref

16.10 Code generator options

The menu option **Tools | Options** lets you specify general as well as specific MapForce settings.



The available settings are as follows.

C++ Settings	Defines the specific compiler settings for the C++ environment, namely: • The Visual Studio edition (2005, 2008, 2010) • The XML library (MSXML, Xerces 2.x, Xerces 3.x) • Whether static or dynamic libraries must be generated • Whether code must be generated with or without MFC support
C# Settings	Defines the specific compiler settings for the C# environment, namely, the Visual Studio edition (2005, 2008, 2010).
Wrapper Classes	Allows you to generate wrapper classes for XML schemas. These wrapper classes can be used by custom code that includes the code generated by MapForce.
Convert all ADO and ODBC Database Connections to JDBC	This option is applicable when you compile mappings to MapForce Server execution files. If the option is enabled, ADO and ODBC database connections are transformed to JDBC using the JDBC driver and the database URL defined in the Database Component Settings dialog box.
Ignore Digital Signatures (unsupported by	This check box is activated by default to skip any digital signature information since currently there is no support for digital signatures

MapForce Server)	in MapForce Server. Deactivating the switch adds digital signature information to the MapForce Server execution file.

16.11 The way to SPL (Spy Programming Language)

This section gives an overview of Spy Programming Language, the code generator's template language.

It is assumed that you have prior programming experience, and are familiar with operators, functions, variables and classes, as well as the basics of object-oriented programming - which is used heavily in SPL.

The templates used by MapForce are supplied in the ...\MapForce\spl folder. You can use these files as an aid to help you in developing your own templates.

How code generator works

Inputs to the code generator are the template files (.spl) and the object model provided by MapForce. The template files contain SPL instructions for creating files, reading information from the object model and performing calculations, interspersed with literal code fragments in the target programming language.

The template file is interpreted by the code generator and outputs **.cpp, .java, .cs** source code files, project files, or any other type of file depending on the template. The source code can then be compiled into an executable file that accesses XML data described by the schema file.

SPL files have access to a wide variety of information that is collated from the source schemas. Please note that an SPL file is not tied to a specific schema, but allows access to all schemas! Make sure you write your SPL files generically, avoid structures etc. which apply to specific schemas!

Example: Creating a new file in SPL:

```
[create "test.cpp"]
#include "stdafx.h"
[close]
```

This is a very basic SPL file. It creates a file named **test.cpp,** and places the include statement within it. The close command completes the template.

16.11.1 Basic SPL structure

An SPL file contains literal text to output, interspersed with code generator instructions.

Code generator instructions are enclosed in square brackets '**[**' and '**]**'.
Multiple statements can be included in a bracket pair. Additional statements have to be separated by a new line or a colon '**:**'.

Valid examples are:

```
[$x = 42
$x = $x + 1]
```

or

```
[$x = 42: $x = $x + 1]
```


Adding text to files

Text not enclosed by **[** and **]**, is written directly to the current output file. If there is no current output file, the text is ignored (see Using files how to create an output file).
To output literal square brackets, escape them with a backslash: **\[** and **\]**; to output a backslash use ****.

Comments

Comments inside an instruction block always begin with a **'** character, and terminate on the next line, or at a block close character **]**.

16.11.2 Declarations

The following statements are evaluated while parsing the SPL template file. They are **not** affected by flow control statements like conditions, loops or subroutines, and are always evaluated exactly once.

These keywords, like all keywords in SPL, are not case sensitive.

Remember that all of these declarations must be inside a block delimited by square brackets.

map *mapname key* **to** *value* [, *key* **to** *value*]...

This statement adds information to a map. See below for specific uses.

map schemanativetype *schematype* **to** *typespec*

The specified built-in XML Schema type will be mapped to the specified native type or class, using the specified formatter. This setting applies only to code generation for version 2007r3 and higher. Typespec is a native type or class name, followed by a comma, followed by the formatter class instance.

Example:
```
map schemanativetype "double" to "double,Altova::DoubleFormatter"
```

map type *schematype* **to** *classname*

The specified built-in XML Schema type will be mapped to the specified class. This setting applies only to code generation for version 2007 or lower.

Example:
```
map type "float" to "CSchemaFloat"
```

default *setting* **is** *value*

This statement allows you to affect how class and member names are derived from the XML Schema.
Note that the setting names are case sensitive.

Example:
```
default "InvalidCharReplacement" is "_"
```

Setting name	Explanation
ValidFirstCharSet	Allowed characters for starting an identifier
ValidCharSet	Allowed characters for other characters in an identifier
InvalidCharReplacement	The character that will replace all characters in names that are not in the ValidCharSet
AnonTypePrefix	Prefix for names of anonymous types*
AnonTypeSuffix	Suffix for names of anonymous types*
ClassNamePrefix	Prefix for generated class names
ClassNameSuffix	Suffix for generated class names
EnumerationPrefix	Prefix for symbolic constants declared for enumeration values
EnumerationUpperCase	"on" to convert the enumeration constant names to upper case
FallbackName	If a name consists only of characters that are not in ValidCharSet, use this one

* Names of anonymous types are built from AnonTypePrefix + element name + AnonTypeSuffix

reserve *word*

Adds the specified word to the list of reserved words. This ensures that it will never be generated as a class or member name.

Example:
```
reserve "while"
```

include *filename*

Example:
```
include "Module.cpp"
```

includes the specified file as SPL source. This allows you to split your template into multiple files for easier editing and handling.

16.11.3 Variables

Any non-trivial SPL file will require variables. Some variables are predefined by the code generator, and new variables may be created simply by assigning values to them.

The **$** character is used when **declaring** or **using** a variable, a variable name is always prefixed by **$**.
Variable names are **case sensitive**.

Variables types:

- integer - also used as boolean, where 0 is false and everything else is true
- string
- object - provided by MapForce
- iterator - see foreach statement

Variable types are declared by first assignment:

```
[$x = 0]
```
x is now an integer.

```
[$x = "teststring"]
```
x is now treated as a string.

Strings

String constants are always enclosed in double quotes, like in the example above. **\n** and **\t** inside double quotes are interpreted as newline and tab, **\"** is a literal double quote, and **** is a backslash. String constants can also span multiple lines.

String concatenation uses the **&** character:

```
[$BasePath = $outputpath & "/" & $JavaPackageDir]
```

Objects

Objects represent the information contained in the XML schemas, database structures, text files and mappings. Objects have **properties**, which can be accessed using the **.** operator. It is not possible to create new objects in SPL (they are predefined by the code generator, derived from the input mapping), but it is possible to assign objects to variables.

Example:

```
class [=$class.Name]
```

This example outputs the word "class", followed by a space and the value of the **Name** property of the **$class** object.

16.11.4 Predefined variables

After a Schema file is analyzed by the code generator, the objects in the table below exist in the Template Engine.

Name	Type	Description
$schematype	integer	1 for DTD, 2 for XML Schema
$TheLibrary	Library	The library derived from the XML Schema or DTD
$module	string	Name of the source Schema without extension
$outputpath	string	The output path specified by the user, or the default output path

For **C++** generation only:

Name	Type	Description
$domtype	integer	1 for MSXML, 2 for Xerces
$XercesVersion	integer	2 for Xerces 2.x, 3 for Xerces 3.x
$libtype	integer	1 for static LIB, 2 for DLL
$mfc	boolean	True if MFC support is enabled
$vc6project	boolean	True if Visual C++ project files are to be generated
$VS2005Project	boolean	True if Visual C++ 2005 project files are to be generated
$VS2008Project	boolean	True if Visual C++ 2008 project files are to be generated
$VS2010Project	boolean	True if Visual C++ 2010 project files are to be generated

For **C#** generation only:

Name	Type	Description
$CreateVS2005Project	boolean	True if Visual Studio 2005 project files are to be generated
$CreateVS2008Project	boolean	True if Visual Studio 2008 project files are to be generated
$CreateVS2010Project	boolean	True if Visual Studio 2010 project files are to be generated

16.11.5 Creating output files

These statements are used to create output files from the code generation.

Remember that all of these statements must be inside a block delimited by square brackets.

create *filename*

creates a new file. The file has to be closed with the **close** statement. All following output is written to the specified file.

Example:

```
[create $outputpath & "/" & $JavaPackageDir & "/" & $application.Name & ".java"]
package [=$JavaPackageName];

public class [=$application.Name]Application {
...
}
```



```
[close]
```

close

closes the current output file.

=$*variable*

writes the value of the specified variable to the current output file.

Example:
```
[$x = 20+3]
The result of your calculation is [=$x] - so have a nice day!
```
- The file output will be:

```
The result of your calculation is 23 - so have a nice day!
```

write *string*

writes the string to the current output file.

Example:
```
[write "C" & $name]
```

This can also be written as:
```
C[=$name]
```

filecopy *source* to *target*

copies the source file to the target file, without any interpretation.

Example:
```
filecopy "java/mapforce/mapforce.png" to $outputpath & "/" & $JavaPackageDir &
"/mapforce.png"
```

16.11.6 Operators

Operators in SPL work like in most other programming languages.

List of SPL operators in descending precedence order:

.	Access object property
( )	Expression grouping
true	boolean constant "true"
false	boolean constant "false"
&	String concatenation
-	Sign for negative number
not	Logical negation
*	Multiply

/	Divide
%	Modulo
+	Add
-	Subtract
<=	Less than or equal
<	Less than
>=	Greater than or equal
>	Greater than
=	Equal
<>	Not equal
and	Logical conjunction (with short circuit evaluation)
or	Logical disjunction (with short circuit evaluation)
=	Assignment

16.11.7 Conditions

SPL allows you to use standard "if" statements. The syntax is as follows:

```
if condition
	statements
else
	statements
endif
```

or, without else:

```
if condition
	statements
endif
```

Please note that there are no round brackets enclosing the condition!
As in any other programming language, conditions are constructed with logical and comparison operators.

Example:

```
[if $namespace.ContainsPublicClasses and $namespace.Prefix <> ""]
	whatever you want ['inserts whatever you want, in the resulting file]
[endif]
```

Switch

SPL also contains a multiple choice statement.

Syntax:

```
switch $variable
	case X:
		statements
	case Y:
	case Z:
```



```
            statements
      default:
            statements
endswitch
```

The case labels must be constants or variables.

The switch statement in SPL does not fall through the cases (as in C), so there is no need for a "break" statement.

16.11.8 Collections and foreach

Collections and iterators

A collection contains multiple objects - like a ordinary array. Iterators solve the problem of storing and incrementing array indexes when accessing objects.

Syntax:

```
foreach iterator in collection
      statements
next
```

Example:

```
[foreach $class in $classes
      if not $class.IsInternal
            ]     class [=$class.Name];
[     endif
next]
```

Example 2:

```
[foreach $i in 1 To 3
     Write "// Step " & $i & "\n"
     ‘ Do some work
next]
```

The first line:
$classes is the **global object** of all generated types. It is a collection of single class objects.

Foreach steps through all the items in $classes, and executes the code following the instruction, up to the **next** statement, for each of them.

In each iteration, **$class** is assigned to the next class object. You simply work with the class object instead of using, classes[i]->Name(), as you would in C++.

All collection iterators have the following additional properties:

Index	The current index, starting with 0
IsFirst	true if the current object is the first of the collection (index is 0)
IsLast	true if the current object is the last of the collection
Current	The current object (this is implicit if not specified and can be left out)

Example:

```
[foreach $enum in $facet.Enumeration
       if not $enum.IsFirst
              ], [
       endif
       ]"[=$enum.Value]"[
next]
```

16.11.9 Subroutines

Code generator supports subroutines in the form of procedures or functions.

Features:

- By-value and by-reference passing of values
- Local/global parameters (local within subroutines)
- Local variables
- Recursive invocation (subroutines may call themselves)

16.11.9.1 Subroutine declaration

Subroutines

Syntax example:

```
Sub SimpleSub()
... lines of code
EndSub
```

- **Sub** is the keyword that denotes the procedure.
- **SimpleSub** is the name assigned to the subroutine.
- Round **parenthesis** can contain a parameter list.
- The code block of a subroutine starts immediately after the closing parameter parenthesis.
- **EndSub** denotes the end of the code block.

Please note:
Recursive or cascaded subroutine **declaration** is not permitted, i.e. a subroutine may not contain another subroutine.

Parameters
Parameters can also be passed by procedures using the following syntax:

- All parameters must be variables
- Variables must be prefixed by the **$** character
- Local variables are defined in a subroutine
- Global variables are declared explicitly, outside of subroutines
- Multiple parameters are separated by the comma character "**,**" within round parentheses
- Parameters can pass values

Parameters - passing values
Parameters can be passed in two ways, by value and by reference, using the keywords **ByVal** and **ByRef** respectively.

Syntax:
```
' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
] ...
```

- **ByVal** specifies that the parameter is passed by value. Note that most objects can only be passed by reference.
- **ByRef** specifies that the parameter is passed by reference. This is the default if neither ByVal nor ByRef is specified.

Function return values
To return a value from a subroutine, use the **return** statement. Such a function can be called from within an expression.

Example:
```
' define a function
[Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )
if $namespacePrefix = ""
  return $localName
else
  return $namespacePrefix & ":" & $localName
endif
EndSub
]
```

16.11.9.2 Subroutine invocation

Use **call** to invoke a subroutine, followed by the procedure name and parameters, if any.

```
Call SimpleSub()
```

or,

```
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
```

Function invocation
To invoke a function (any subroutine that contains a **return** statement), simply use its name inside an expression. Do not use the **call** statement to call functions.
Example:

```
$QName = MakeQualifiedName($namespace, "entry")
```

16.11.9.3 Subroutine example

Highlighted example showing subroutine declaration and invocation.

A sample SPL file:

```
[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
]
```

The same sample code:

```
[create $outputpath & $module & "output.txt"

' define sub SimpleSub()
Sub SimpleSub()
]SimpleSub() called
[endsub

' execute sub SimpleSub()
Call SimpleSub()

$ParamByValue       = "Original Value"
]ParamByValue       = [=$ParamByValue]
```



```
[$ParamByRef  = "Original Value"
]ParamByRef   = [=$ParamByRef]

' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
]CompleteSub called.
       param = [=$param]
       paramByValue = [=$paramByValue]
       paramByRef = [=$paramByRef]
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]      Set values inside Sub
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]CompleteSub finished.
[endsub

' run sub CompleteSub()
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
]
ParamByValue=[=$ParamByValue]
ParamByRef=[=$ParamByRef]
[
Close
]
```

16.11.10 Built in Types

The section describes the properties of the built-in types used in the predefined variables which describe the parsed schema.

16.11.10.1 Library

This object represents the whole library generated from the XML Schema or DTD.

Property	Type	Description
SchemaNamespaces	Namespace collection	Namespaces in this library
SchemaFilename	string	Name of the XSD or DTD file this library is derived from
SchemaType	integer	1 for DTD, 2 for XML Schema
Guid	string	A globally unique ID
CodeName	string	Generated library name (derived from schema file name)

16.11.10.2 Namespace

One namespace object per XML Schema namespace is generated. Schema components that are not in any namespace are contained in a special namespace object with an empty NamespaceURI.
Note that for DTD, namespaces are also derived from attributes whose names begin with "xmlns".

Property	Type	Description
CodeName	string	Name for generated code (derived from prefix)
LocalName	string	Namespace prefix
NamespaceURI	string	Namespace URI
Types	Type collection	All types contained in this namespace
Library	Library	Library containing this namespace

16.11.10.3 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	Namespace containing this type
Attributes	Member collection	Attributes contained in this type*
Elements	Member collection	Child elements contained in this type
IsSimpleType	boolean	True for simple types, false for complex types
IsDerived	boolean	True if this type is derived from another type, which is also represented by a Type object
IsDerivedByExtension	boolean	True if this type is derived by extension
IsDerivedByRestriction	boolean	True if this type is derived by restriction
IsDerivedByUnion	boolean	True if this type is derived by union

Property	Type	Description
IsDerivedByList	boolean	True if this type is derived by list
BaseType	Type	The base type of this type (if IsDerived is true)
IsDocumentRootType	boolean	True if this type represents the document itself
Library	Library	Library containing this type
IsFinal	boolean	True if declared as final in the schema
IsMixed	boolean	True if this type can have mixed content
IsAbstract	boolean	True if this type is declared as abstract
IsGlobal	boolean	True if this type is declared globally in the schema
IsAnonymous	boolean	True if this type is declared locally in an element

For simple types only:

Property	Type	Description
IsNativeBound	boolean	True if native type binding exists
NativeBinding	NativeBinding	Native binding for this type
Facets	Facets	Facets of this type
Whitespace	string	Shortcut to the Whitespace facet

* Complex types with text content (these are types with mixed content and complexType with simpleContent) have an additional unnamed attribute member that represents the text content.

16.11.10.4 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.

Property	Type	Description
NamespaceURI	string	The namespace URI of this Element/Attribute within XML instance documents/streams.
DeclaringType	Type	Type originally declaring the member (equal to ContainingType for non-inherited members)
ContainingType	Type	Type where this is a member of
DataType	Type	Data type of this member's content
Library	Library	Library containing this member's DataType
IsAttribute	boolean	True for attributes, false for elements
IsOptional	boolean	True if minOccurs = 0 or optional attribute
IsRequired	boolean	True if minOccurs > 0 or required attribute
IsFixed	boolean	True for fixed attributes, value is in Default property
IsDefault	boolean	True for attributes with default value, value is in Default property
IsNillable	boolean	True for nillable elements
IsUseQualified	boolean	True if NamespaceURI is not empty
MinOccurs	integer	minOccurs, as in schema. 1 for required attributes
MaxOccurs	integer	maxOccurs, as in schema. 0 for prohibited attributes, -1 for unbounded
Default	string	Default value

16.11.10.5 NativeBinding

This object represents the binding of a simple type to a native type in the target programming language, as specified by the "schemanativetype" map.

Property	Type	Description
ValueType	string	Native type
ValueHandler	string	Formatter class instance

16.11.10.6 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	Type this facet is declared on
FacetCheckerName	string	Name of facet checker (from schemafacet map)
FacetValue	string or integer	Actual value of this facet

Chapter 17

The MapForce API

17 The MapForce API

The COM-based API of MapForce enables clients to easily access the functionality of MapForce. As a result, it is now possible to automate a wide range of tasks.

MapForce follows the common specifications for automation servers set out by Microsoft. It is possible to access the methods and properties of the MapForce API from common development environments, such as those using .NET, C++ and VisualBasic, and with scripting languages like JavaScript and VBScript.

The following guidelines should be considered in your client code:

- Do not hold references to objects in memory longer than you need them. If a user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Be aware that if your client code crashes, instances of MapForce may still remain in the system.
- See Error handling for details of how to avoid annoying error messages.
- Free references explicitly, if using languages such as C++.

To integrate your version of MapForce2016 into the Microsoft Visual Studio versions 2005, 2008, 2010, or 2012 you need to do the following:

- Install Microsoft Visual Studio
- Install MapForce (Enterprise or Professional Edition)
- Download and run the MapForce Visual Studio .NET Edition integration for Microsoft Visual Studio .NET package. This package is available on the MapForce (Enterprise and Professional Editions) download page at www.altova.com.

- Example files for the integration package are installed into the respective language folders below the **c:\Program Files\Altova\MapForce2016\Examples\ActiveX** installation folder.

Please note:
The manager control (e.g. MapForceControl) of ActiveX Controls can be instantiated only once in a process (even if it is destroyed, you cannot re-create it. To avoid this create the MapForce Control and its parent window only once and then show/hide the window as needed.

Mapforce integration and deployment on client computers:

If you create a .NET application and intend to distribute it to other clients, you will have to install the following on the client computer(s):

- The MapForce application
- The MapForce integration package.
- The custom integration code.

17.1 Overview

This overview of the MapForce API provides you with the object model for the API and a description of the most important API concepts. The following topics are covered:

- The object model
- Example: Code-Generation
- Example: Mapping Execution
- Example: Project Support
- Error handling

17.1.1 Object model

The starting point for every application which uses the MapForce API is the `Application` object.

To create an instance of the `Application` object, call `CreateObject("MapForce.Application")` from VisualBasic, or a similar function from your preferred development environment, to create a COM object. There is no need to create any other objects to use the complete MapForce API. All other interfaces are accessed through other objects, with the `Application` object as the starting point.

The application object consists of the following parts (each indentation level indicates a child–parent relationship with the level directly above):

```
Application
   Options
   Project
      ProjectItem
   Documents
      Document
         MapForceView
         Mapping
            Component
               Datapoint
            Components
            Connection
         Mappings
   ErrorMarkers
      ErrorMarker
AppOutputLines
   AppOutputLine
      AppOutputLines
         ...
      AppOutputLineSymbol
```

Once you have created an `Application` object, you can start using the functionality of MapForce. You will generally either open an existing `Document`, create a new one, or generate code for, or from, this document.

17.1.2 Example: Code-Generation

See also
Code Generation

The following JScript example shows how to load an existing document and generate different kinds of mapping code for it.

```
// ------------------- begin JScript example ---------------------
// Generate Code for existing mapping.
// works with Windows scripting host.

// ----------------- helper function ------------------
function Exit(strErrorText)
{
	WScript.Echo(strErrorText);
	WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
	if (objErr != null)
		Exit ("ERROR: (" + (objErr.number & 0xffff) + ")" +
objErr.description + " - " + strText);
	else
		Exit ("ERROR: " + strText);
}
// ------------------------------------------------------

// ----------------- MAIN ------------------

// ----- create the Shell and FileSystemObject of the windows scripting
try
{
	objWshShell = WScript.CreateObject("WScript.Shell");
	objFSO = WScript.CreateObject("Scripting.FileSystemObject");
}
catch(err)
	{ Exit("Can't create WScript.Shell object"); }

// ----- open MapForce or access running instance and make it visible
try
{
	objMapForce = WScript.GetObject ("", "MapForce.Application");
	objMapForce.Visible = true;	// remove this line to perform
background processing
}
catch(err) { WScript.Echo ("Can't access or create MapForce.Application"); }

// ----- open an existing mapping. adapt this to your needs!
objMapForce.OpenDocument(objFSO.GetAbsolutePathName ("Test.mfd"));
```

```
// ----- access the mapping to have access to the code generation methods
var objDoc = objMapForce.ActiveDocument;

// ----- set the code generation output properties and call the code generation
methods.
// ----- adapt the output directories to your needs
try
{
	// ----- code generation uses some of these options
	var objOptions = objMapForce.Options;

	// ----- generate XSLT -----
	objOptions.XSLTDefaultOutputDirectory = "C:\\test\\TestCOMServer\\XSLT";
	objDoc.GenerateXSLT();

	// ----- generate Java Code -----
	objOptions.CodeDefaultOutputDirectory = "C:\\test\\TestCOMServer\\Java";
	objDoc.GenerateJavaCode();

	// ----- generate CPP Code, use same cpp code options as the last time
-----
	objOptions.CodeDefaultOutputDirectory = "C:\\test\\TestCOMServer\\CPP";
	objDoc.GenerateCppCode();

	// ----- generate C# Code, use options C# code options as the last time
-----
	objOptions.CodeDefaultOutputDirectory = "C:\\test\\TestCOMServer\\CHash";
	objDoc.GenerateCHashCode();
}
catch (err)
	{ ERROR ("while generating XSL or program code", err); }


// hide MapForce to allow it to shut down
objMapForce.Visible = false;

// -------------------- end example ---------------------
```

17.1.3 Example: Mapping Execution

The following JScript example shows how to load an existing document with a simple mapping, access its components, set input- and output-instance file names and execute the mapping.

```
/*
   This sample file performs the following operations:

   Load existing MapForce mapping document.
   Find source and target component.
   Set input and output instance filenames.
   Execute the transformation.
```

```
   Works with Windows scripting host.
*/

// ---- general helpers ------------------------------

function Exit( message )
{
   WScript.Echo( message );
   WScript.Quit(-1);
}

function ERROR( message, err )
{
   if( err != null )
      Exit( "ERROR: (" + (err.number & 0xffff) + ") " + err.description + " - "
+ message );
   else
      Exit( "ERROR: " + message );
}



// ---- MapForce constants -----------------------

var eComponentUsageKind_Unknown     = 0;
var eComponentUsageKind_Instance    = 1;
var eComponentUsageKind_Input       = 2;
var eComponentUsageKind_Output      = 3;



// ---- MapForce helpers -----------------------

// Searches in the specified mapping for a component by name and returns it.
// If not found, throws an error.
function FindComponent( mapping, component_name )
{
   var components = mapping.Components;
   for( var i = 0 ; i < components.Count ; ++i )
   {
      var component = components.Item( i + 1 );
      if( component.Name == component_name )
         return component;
   }
   throw new Error( "Cannot find component with name " + component_name );
}



// Browses components in a mapping and returns the first one found acting as
// source component (i.e. having connections on its right side).
function GetFirstSourceComponent( mapping )
{
   var components = mapping.Components;
   for( var i = 0 ; i < components.Count ; ++i )
```



```
   {
      var component = components.Item( i + 1 );
      if( component.UsageKind == eComponentUsageKind_Instance &&
         component.HasOutgoingConnections )
      {
         return component;
      }
   }
   throw new Error( "Cannot find a source component" );
}

// Browses components in a mapping and returns the first one found acting as
// target component (i.e. having connections on its left side).
function GetFirstTargetComponent( mapping )
{
   var components = mapping.Components;
   for( var i = 0 ; i < components.Count ; ++i )
   {
      var component = components.Item( i + 1 );
      if( component.UsageKind == eComponentUsageKind_Instance &&
         component.HasIncomingConnections )
      {
         return component;
      }
   }
   throw new Error( "Cannot find a target component" );
}



function IndentTextLines( s )
{
   return "\t" + s.replace( /\n/g, "\n\t" );
}

function GetAppoutputLineFullText( oAppoutputLine )
{
   var s = oAppoutputLine.GetLineText();
   var oAppoutputChildLines = oAppoutputLine.ChildLines;
   var i;

   for( i = 0 ; i < oAppoutputChildLines.Count ; ++i )
   {
      oAppoutputChildLine = oAppoutputChildLines.Item( i + 1 );
      sChilds = GetAppoutputLineFullText( oAppoutputChildLine );
      s += "\n" + IndentTextLines( sChilds );
   }

   return s;
}

// Create a nicely formatted string from AppOutputLines
function GetResultMessagesString( oAppoutputLines )
{
   var s1 = "Transformation result messages:\n";
```

```
   var oAppoutputLine;
   var i;

   for( i = 0 ; i < oAppoutputLines.Count ; ++i )
   {
      oAppoutputLine = oAppoutputLines.Item( i + 1 );
      s1 += GetAppoutputLineFullText( oAppoutputLine );
      s1 += "\n";
   }

   return s1;
}



// ---- MAIN -----------------------------------

var wshShell;
var fso;
var mapforce;

// create the Shell and FileSystemObject of the windows scripting system
try
{
   wshShell = WScript.CreateObject( "WScript.Shell" );
   fso = WScript.CreateObject( "Scripting.FileSystemObject" );
}
catch( err )
   { ERROR( "Can't create windows scripting objects", err ); }

// open MapForce or access currently running instance
try
{
   mapforce = WScript.GetObject( "", "MapForce.Application" );
}
catch( err )
   { ERROR( "Can't access or create MapForce.Application", err ); }

try
{
       // Make MapForce UI visible. This is an API requirement for output
generation.
  mapforce.Visible = true;

       // open an existing mapping.
   // **** adjust the examples path to your needs ! **************
   var sMapForceExamplesPath = fso.BuildPath(
            wshShell.SpecialFolders( "MyDocuments" ),
            "Altova\\MapForce2016\\MapForceExamples" );
   var sDocFilename = fso.BuildPath( sMapForceExamplesPath, "PersonList.mfd" );
   var doc = mapforce.OpenDocument( sDocFilename );

   // Find existing components by name in the main mapping.
   // Note, the names of components may not be unique as a schema component's
name
```

```
   // is derived from its schema file name.
   var source_component = FindComponent( doc.MainMapping, "Employees" );
   var target_component = FindComponent( doc.MainMapping, "PersonList" );
   // If you do not know the names of the components for some reason, you could
   // use the following functions instead of FindComponent.
   //var source_component = GetFirstSourceComponent( doc.MainMapping );
   //var target_component = GetFirstTargetComponent( doc.MainMapping );

   // specify the desired input and output files.
   source_component.InputInstanceFile = fso.BuildPath( sMapForceExamplesPath,
"Employees.xml" );
   target_component.OutputInstanceFile = fso.BuildPath( sMapForceExamplesPath,
"test_transformation_results.xml" );

   // Perform the transformation.
   // You can use doc.GenerateOutput() if you do not need result messages.
   // If you have a mapping with more than one target component and you want
   // to execute the transformation only for one specific target component,
   // call target_component.GenerateOutput() instead.
   var result_messages = doc.GenerateOutputEx();

   var summary_info =
         "Transformation performed from " + source_component.InputInstanceFile
+ "\n" +
         "to " + target_component.OutputInstanceFile + "\n\n" +
         GetResultMessagesString( result_messages );
   WScript.Echo( summary_info );
}
catch( err )
{
   ERROR( "Failure", err );
}
```

17.1.4 Example: Project Support

See also
Code Generation

The following JScript example shows how you can use the MapForce project and project-item objects of the MapForce API to automate complex tasks. Depending on your installation you might need to change the value of the variable `strSamplePath` to the example folder of your MapForce installation.

To successfully run all operations in this example below, you will need the Enterprise version of MapForce. If you have the Professional version running, you should comment out the lines that insert the WebService project. Users of the Basic edition will not have access to project-related functions at all.

```
// ////////////// global variables /////////////////
var objMapForce = null;
var objWshShell = null;
var objFSO = null;
```

```
// !!! adapt the following path to your needs. !!!
var strSamplePath = "C:\\Documents and Settings\\<username>\\My Documents\
\Altova\\MapForce2016\\MapForceExamples\\Tutorial\\";

// ////////////////////// Helpers ////////////////////////////

function Exit(strErrorText)
{
	WScript.Echo(strErrorText);
	WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
	if (objErr != null)
		Exit ("ERROR: (" + (objErr.number & 0xffff) + ")" +
objErr.description + " - " + strText);
	else
		Exit ("ERROR: " + strText);
}

function CreateGlobalObjects ()
{
	// the Shell and FileSystemObject of the windows scripting host often
always useful
	try
	{
		objWshShell = WScript.CreateObject("WScript.Shell");
		objFSO = WScript.CreateObject("Scripting.FileSystemObject");
	}
	catch(err)
		{ Exit("Can't create WScript.Shell object"); }

	// create the MapForce connection
	// if there is a running instance of MapForce (that never had a
connection) - use it
	// otherwise, we automatically create a new instance
	try
	{
		objMapForce = WScript.GetObject("", "MapForce.Application");
	}
	catch(err)
	{
		{ Exit("Can't access or create MapForce.Application"); }
	}
}

// ------------------------------------------------------------
// print project tree items and their properties recursively.
// ------------------------------------------------------------
function PrintProjectTree( objProjectItemIter, strTab )
{
```

```
	while ( ! objProjectItemIter.atEnd() )
	{
		// get current project item
		objItem = objProjectItemIter.item();

		try
		{
			// ----- print common properties
			strGlobalText += strTab + "[" + objItem.Kind + "]" +
objItem.Name + "\n";

			// ----- print code generation properties, if available
			try
			{
				if ( objItem.CodeGenSettings_UseDefault )
					strGlobalText += strTab + "  Use default code
generation settings\n";
				else
					strGlobalText += strTab + "  code generation
language is " +
							objItem.CodeGenSettings_Langu
age +
							" output folder is " +
objItem.CodeGenSettings_OutputFolder + "\n";
			}
			catch( err ) {}

			// ----- print WSDL settings, if available
			try
			{
				strGlobalText += strTab + "  WSDL File is " +
objItem.WSDLFile +
						" Qualified Name is " +
objItem.QualifiedName + "\n";
			}
			catch( err ) {}
		}
		catch( ex )
			{ strGlobalText += strTab + "[" + objItem.Kind + "]\n" }

		// ---- recurse
		PrintProjectTree( new Enumerator( objItem ), strTab + '  ' );

		objProjectItemIter.moveNext();
	}
}

// ------------------------------------------------------------
// Load example project installed with MapForce.
// ------------------------------------------------------------
function LoadSampleProject()
{
	// close open project
```

```
	objProject = objMapForce.ActiveProject;
	if ( objProject != null )
		objProject.Close();

	// open sample project and iterate through it.
	// sump properties of all project items

	objProject = objMapForce.OpenProject(strSamplePath +
"MapForceExamples.mfp");
	strGlobalText = '';
	PrintProjectTree( new Enumerator (objProject), ' ' )
	WScript.Echo( strGlobalText );

	objProject.Close();
}

// ------------------------------------------------------------
// Create a new project with some folders, mappings and a
// Web service project.
// ------------------------------------------------------------
function CreateNewProject()
{
	try
	{
		// create new project and specify file to store it.
		objProject = objMapForce.NewProject(strSamplePath + "Sample.mfp");

		// create a simple folder structure
		objProject.CreateFolder( "New Folder 1");
		objFolder1 = objProject.Item(0);
		objFolder1.CreateFolder( "New Folder 2");
		objFolder2 = ( new Enumerator( objFolder1 ) ).item();  // an
alternative to Item(0)

		// add two different mappings to folder structure
		objFolder1.AddFile( strSamplePath + "DB_Altova_SQLXML.mfd");
		objMapForce.Documents.OpenDocument(strSamplePath +
"InspectionReport.mfd");
		objFolder2.AddActiveFile();

		// override code generation settings for this folder
		objFolder2.CodeGenSettings_UseDefault = false;
		objFolder2.CodeGenSettings_OutputFolder = strSamplePath +
"SampleOutput"
		objFolder2.CodeGenSettings_Language = 1;	//C++

		// insert Web service project based on a wsdl file from the
installed examples
		objProject.InsertWebService( strSamplePath + "TimeService/
TimeService.wsdl",
					"{http://www.Nanonull.com/TimeService/}
TimeService",
							"TimeServiceSoap",
```

```
                                                                    true );
                objProject.Save();
                if ( ! objProject.Saved )
                        WScript.Echo("problem occurred when saving project");

                // dump project tree
                strGlobalText = '';
                PrintProjectTree( new Enumerator (objProject), ' ' )
                WScript.Echo( strGlobalText );
        }
        catch (err)
        { ERROR("while creating new project", err ); }
}

// ------------------------------------------------------------
// Generate code for a project's sub-tree. Mix default code
// generation parameters and overloaded parameters.
// ------------------------------------------------------------
function GenerateCodeForNewProject()
{
        // since the Web service project contains only initial mappings,
        // we generate code only for our custom folder.
        // code generation parameters from project are used for Folder1,
        // whereas Folder2 provides overwritten values.
        objFolder = objProject.Item(0);
        objFolder1.GenerateCode();
}

// ////////////////////// MAIN ////////////////////////////

CreateGlobalObjects();
objMapForce.Visible = true;

LoadSampleProject();
CreateNewProject();
GenerateCodeForNewProject();

// uncomment to shut down application when script ends
// objMapForce.Visible = false;
```

17.1.5 Error handling

The MapForce API returns errors in two different ways. Every API method returns an `HRESULT`. This return value informs the caller about any malfunctions during the execution of the method. If the call was successful, the return value is equal to `S_OK`. C/C++ programmers generally use `HRESULT` to detect errors.

VisualBasic, scripting languages, and other high-level development environments do not give the programmer access to the returning `HRESULT` of a COM call. They use the second error-raising mechanism supported by the MapForce API, the `IErrorInfo` interface. If an error occurs, the

API creates a new object that implements the `IErrorInfo` interface. The development environment takes this interface and fills its own error-handling mechanism with the provided information.

The following text describes how to deal with errors raised from the MapForce API in different development environments.

VisualBasic
A common way to handle errors in VisualBasic is to define an error handler. This error handler can be set with the `On Error` statement. Usually the handler displays an error message and does some cleanup to avoid spare references and any kind of resource leaks. VisualBasic fills its own `Err` object with the information from the `IErrorInfo` interface.

Example:

```
Sub Validate()
  'place variable declarations here

  'set error handler
  On Error GoTo ErrorHandler

  'if generation fails, program execution continues at ErrorHandler:
  objMapForce.ActiveDocument.GenerateXSLT()

  'additional code comes here

  'exit
  Exit Sub

  ErrorHandler:
  MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &
         "Description: " & Err.Description)
End Sub
```

JavaScript
The Microsoft implementation of JavaScript (JScript) provides a try-catch mechanism to deal with errors raised from COM calls. It is very similar to the VisualBasic approach, in that you also declare an error object containing the necessary information.

Example:

```
function Generate()
{
 // please insert variable declarations here

 try
 {
  objMapForce.ActiveDocument.GenerateXSLT();
 }
 catch(Error)
 {
  sError = Error.description;
  nErrorCode = Error.number & 0xffff;
```



```
   return false;
  }

  return true;
 }
```

C/C++
C/C++ gives you easy access to the `HRESULT` of the COM call and to the `IErrorInterface`.

```
HRESULT hr;

// Call GenerateXSLT() from the MapForce API
if(FAILED(hr = ipDocument->GenerateXSLT()))
{
 IErrorInfo *ipErrorInfo = Null;

 if(SUCCEEDED(::GetErrorInfo(0, &ipErrorInfo)))
 {
  BSTR bstrDescr;
  ipErrorInfo->GetDescription(&bstrDescr);

  // handle Error information
  wprintf(L"Error message:\t%s\n",bstrDescr);
  ::SysFreeString(bstrDescr);

  // release Error info
  ipErrorInfo->Release();
 }
}
```

17.1.6 Programming Languages

Programming languages differ in the way they support COM access. A few examples for the most frequently used languages (*links below*) will help you get started. The code listings in this section are available in files in the API Examples folder and can be tested straight away. The path to the API Examples folder is given below:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/MapForce2016/MapForceExamples/API/`
Windows Vista, Windows 7/8	`C:/Users/<username>/Documents/ Altova/MapForce2016/MapForceExamples/API/`

C#
C# can be used to access the Application API functionality. The code listings show how to access the API for certain basic functionality.

Java

The MapForce API can be accessed from Java code. This section explains how some basic MapForce functionality can be accessed from Java code. It is organized into the following sub-sections.

JScript

The JScript listings demonstrate the following basic functionality.

17.1.6.1 C#

The C# programming language can be used to access the Application API functionality. You could use Visual Studio 2008, 2010, or 2012 to create the C# code, saving it in a Visual Studio project. Create the project as follows:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the MapForce Type Library by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed COM components. Select the MapForce Type Library component from the list to add it.
3. Enter the code you want.
4. Compile the code and run it.

Example C# project

Your MapForce package contains an example C# project, which is located in the C# folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/MapForce2016/MapForceExamples/API/`
Windows Vista, Windows 7/8	`C:/Users/<username>/Documents/ Altova/MapForce2016/MapForceExamples/API/`

You can compile and run the project from within Visual Studio 2008, 2010, or 2012.

Platform configuration

If you have a 64-bit operating system and are using a 32-bit installation of MapForce, you must add the x86 platform in the solution's Configuration Manager and build the sample using this configuration.

A new x86 platform (for the active solution in Visual Studio) can be created in the New Solution Platform dialog (**Build | Configuration Manager | Active solution platform | <New...>**)

File List:

Readme.txt	This file
AutomateMapForce.csproj	Visual Studio 2008, 2010, and 2012 project file
AutomateMapForce_VS2008.sln	Visual Studio 2008 solution file
AutomateMapForce_VS2010.sln	Visual Studio 2010 solution file

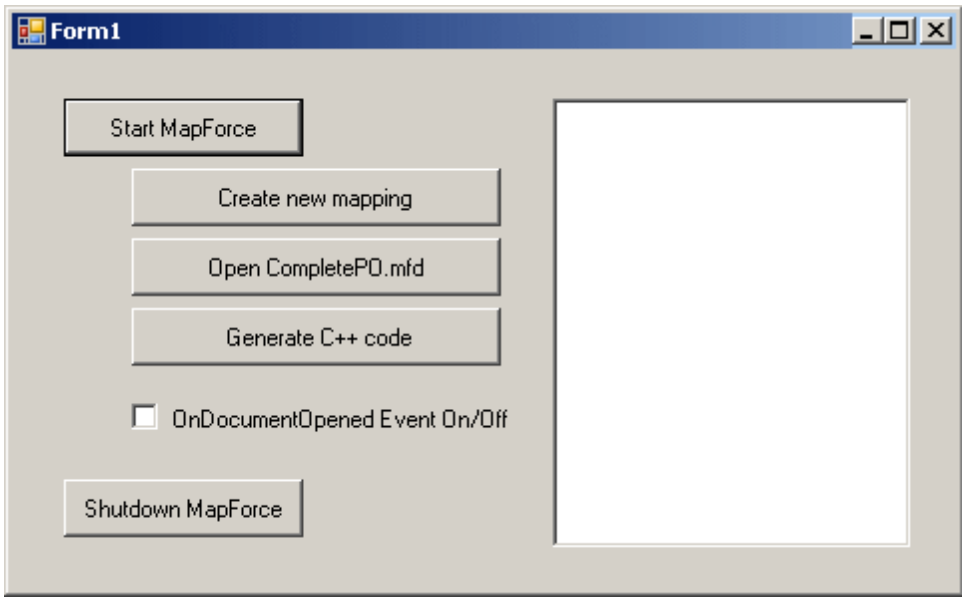
Program.cs	C# example source code

Properties:
Form1.cs
Form1.Designer.cs
Form1.resx

What the example does

The example shows a simple user interface (*screenshot below*) with buttons that invoke basic MapForce operations:

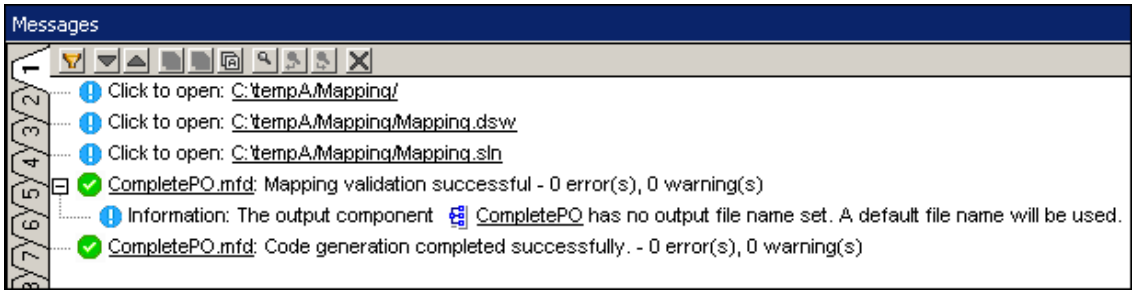
- Starts MapForce
- Creates a new mapping design
- Opens the CompletePO.mfd file from the ...\MapForceExamples folder
- Generates C++ code in a temp directory
- Shuts down MapForce



Compiling and running the example

Double-click the file `AutomateMapForce_VS2008.sln` (for Visual Studio 2008) or the file `AutomateMapForce_VS2010.sln` (for Visual Studio 2010). Alternatively the file can be opened from within Visual Studio (with **File | Open | Project/Solution**). To compile and run the example, select **Debug | Start Debugging** or **Debug | Start Without Debugging**.

Clicking the GenerateC++ code button generates code for the previously opened CompletePO.mfd mapping.



Code listing of Form1.cs

The listing is commented for ease of understanding. Parts of the code are also presented separately in the sub-sections of this section, according to the Application API functionality they access.

The code essentially consists of a series of handlers for the buttons in the user interface shown in the screenshot above.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // An instance of MapForce accessed via its automation interface.
        MapForceLib.Application MapForce;

        // Location of examples installed with MapForce
        String strExamplesFolder;

        private void Form1_Load(object sender, EventArgs e)
        {
            // locate examples installed with MapForce.
            // REMARK: You might need to adapt this if you have a different
major version of the product.
            strExamplesFolder =
Environment.GetEnvironmentVariable("USERPROFILE") + "\\My Documents\\Altova\
\MapForce2012\\MapForceExamples\\";
        }

        // handler for the "Start MapForce" button
        private void StartMapForce_Click(object sender, EventArgs e)
```



```
        {
            if (MapForce == null)
            {
                Cursor.Current = Cursors.WaitCursor;

                // if we have no MapForce instance, we create one a nd make it
visible.
                MapForce = new MapForceLib.Application();
                MapForce.Visible = true;

                Cursor.Current = Cursors.Default;
            }
            else
            {
                // if we have already an MapForce instance running we toggle its
visibility flag.
                MapForce.Visible = !MapForce.Visible;
            }
        }

        // handler for the "Open CompletePO.mfd" button
        private void openCompletePO_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
                StartMapForce_Click(null, null);

            // Open one of the sample files installed with the product.
            MapForce.OpenDocument(strExamplesFolder + "CompletePO.mfd");
        }

        // handler for the "Create new mapping" button
        private void newMapping_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
                StartMapForce_Click(null, null);

            // Create a new mapping
            MapForce.NewMapping();
        }

        // handler for the "Shutdown MapForce" button
        // shut-down application instance by explicitly releasing the COM
object.
        private void shutdownMapForce_Click(object sender, EventArgs e)
        {
            if (MapForce != null)
            {
                // allow shut-down of MapForce by releasing UI
                MapForce.Visible = false;

                // explicitly release COM object
                try
                {
                    while
(System.Runtime.InteropServices.Marshal.ReleaseComObject(MapForce) > 0) ;
                }
```

```
                finally
                {
                    // avoid later access to this object.
                    MapForce = null;
                }
            }
        }

        // handler for button "Generate C++ Code"
        private void generateCppCode_Click(object sender, EventArgs e)
        {
            if (MapForce == null)
                listBoxMessages.Items.Add("start MapForce first.");
            // COM errors get returned to C# as exceptions. We use a try/catch
block to handle them.
            try
            {
                String strError = "";
                MapForceLib.Document doc = MapForce.ActiveDocument;

                listBoxMessages.Items.Add("Active document " + doc.Name);
                MapForceLib.ErrorMarkers errMarkers =
doc.GenerateCodeEx(MapForceLib.ENUMProgrammingLanguage.eCpp);

                System.Collections.IEnumerator errMarkersCollection =
errMarkers.GetEnumerator();

                bool bEmpty = true;
                while (errMarkersCollection.MoveNext())
                {
                    bEmpty = false;
                    Object obj = errMarkersCollection.Current;

                    if (obj is MapForceLib.ErrorMarker)
                        strError = ((MapForceLib.ErrorMarker)obj).Text;
                    listBoxMessages.Items.Add("Error text: " + strError);
                }

                if (bEmpty)
                    listBoxMessages.Items.Add("Code generation completed
successfully.");
            }
            catch (Exception ex)
            {
                // The COM call was not successful.
                // Probably no application instance has been started or no
document is open.
                MessageBox.Show("COM error: " + ex.Message);
            }
        }

        delegate void addListBoxItem_delegate(string sText);
        // called from the UI thread
        private void addListBoxItem(string sText)
        {
            listBoxMessages.Items.Add(sText);
```



```
        }
        // wrapper method to allow to call UI controls methods from a worker
thread
        void syncWithUIthread(Control ctrl, addListBoxItem_delegate
methodToInvoke, String sText)
        {
            // Control.Invoke: Executes on the UI thread, but calling thread
waits for completion before continuing.
            // Control.BeginInvoke: Executes on the UI thread, and calling
thread doesn't wait for completion.
            if (ctrl.InvokeRequired)
                ctrl.BeginInvoke(methodToInvoke, new Object[] { sText });
        }

        // event handler for OnDocumentOpened event
        private void handleOnDocumentOpened(MapForceLib.Document i_ipDocument)
        {
            String sText = "";

            if (i_ipDocument.Name.Length > 0)
                sText = "Document " + i_ipDocument.Name + " was opened!";
            else
                sText = "A new mapping was created.";

            // we need to synchronize the calling thread with the UI thread
because
            // the COM events are triggered from a working thread
            addListBoxItem_delegate methodToInvoke = new
addListBoxItem_delegate(addListBoxItem);
            // call syncWithUIthread with the following arguments:
            // 1 - listBoxMessages - list box control to display messages from
COM events
            // 2 - methodToInvoke  - a C# delegate which points to the method
which will be called from the UI thread
            // 3 - sText           - the text to be displayed in the list box
            syncWithUIthread(listBoxMessages, methodToInvoke, sText);
        }

        private void checkBoxEventOnOff_CheckedChanged(object sender, EventArgs
e)
        {
            if (MapForce != null)
            {
                if (checkBoxEventOnOff.Checked)
                    MapForce.OnDocumentOpened += new
MapForceLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpe
ned);
                else
                    MapForce.OnDocumentOpened -= new
MapForceLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpe
ned);
            }
        }
    }
}
```

17.1.6.2 Java

The Application API can be accessed from Java code. To allow accessing the MapForce automation server directly from Java code, the libraries listed below must reside in the `classpath`. They are installed in the folder: `JavaAPI` in the MapForce application folder.

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers
- `AltovaAutomation.jar`: Java classes to access Altova automation servers
- `MapForceAPI.jar`: Java classes that wrap the MapForce automation interface
- `MapForceAPI_JavaDoc.zip`: a `Javadoc` file containing help documentation for the Java API

Note: In order to use the Java API, the DLL and jar files must be in the Java Classpath.

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 of the example Java project, see the section, Example Java Project.

Rules for mapping the Application API names to Java

The rules for mapping between the Application API and the Java wrapper are as follows:

- ***Classes and class names***
  For every interface of the MapForce automation interface a Java class exists with the name of the interface.

- ***Method names***
  Method names on the Java interface are the same as used on the COM interfaces but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with `get` and `set` can be used. If a property does not support write-access, no setter method is available. Example: For the `Name` property of the `Document` interface, the Java methods `getName` and `setName` are available.

- ***Enumerations***
  For every enumeration defined in the automation interface, a Java enumeration is defined with the same name and values.

- ***Events and event handlers***
  For every interface in the automation interface that supports events, a Java interface with the same name plus '`Event`' is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name of this Java class is the name of the event interface plus '`DefaultHandler`'. For example:
  `Application`: Java class to access the application
  `ApplicationEvents`: Events interface for the Application
  `ApplicationEventsDefaultHandler`: Default handler for `ApplicationEvents`

The following section explains how some basic MapForce functionality can be accessed from Java code.

- Example Java Project

Example Java Project

The MapForce installation package contains an example Java project, located in the Java folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/MapForce2016/MapForceExamples/API/`
Windows Vista, Windows 7/8	`C:/Users/<username>/Documents/ Altova/MapForce2016/MapForceExamples/API/`

This folder contains Java examples for the MapForce API. 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:

`AltovaAutomation.dll`	Java-COM bridge: DLL part
`AltovaAutomation.jar`	Java-COM bridge: Java library part
`MapForceAPI.jar`	Java classes of the MapForce API
`RunMapForce.java`	Java example source code
`BuildAndRun.bat`	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
`.classpath`	Eclipse project helper file
`.project`	Eclipse project file
`MapForceAPI_JavaDoc.zip`	`Javadoc` file containing help documentation for the Java API

What the example does

The example starts up MapForce and performs a few operations, including opening and closing documents. When done, MapForce stays open. You must close it manually.

Running the example from the command line

To run the example from the command line, open a command prompt window, go to the Java folder of the API Examples folder (*see above for location*), and then type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a Java Development Kit (JDK) 7 or later installation on your computer.

Press the **Return** key. The Java source in `RunMapForce.java` will be compiled and then executed.

Loading the example in Eclipse

Open Eclipse and use the **File | Import... | General | Existing Projects into Workspace** command to add the Eclipse project file (`.project`) located in the Java folder of the API Examples folder (*see above for location*). The project `RunMapForce` will then appear in your Package Explorer or Navigator.

Select the project and then the command **Run as | Java Application** to execute the example.

Note: You can select a class name or method of the Java API and press F1to get help for that class or method.

Java source code listing

The Java source code in the example file `RunMapForce.java` is listed below with comments.

```
// access general JAVA-COM bridge classes
import java.util.Iterator;

import com.altova.automation.libs.*;

// access MapForce Java-COM bridge
import com.altova.automation.MapForce.*;
import com.altova.automation.MapForce.Enums.ENUMProgrammingLanguage;

/**
 * A simple example that starts MapForce COM server and performs a few
operations on it.
 * Feel free to extend.
 */
public class RunMapForce
{
   public static void main(String[] args)
   {
      // an instance of the application.
      Application mapforce = null;

      // instead of COM error handling use Java exception mechanism.
      try
      {
         // Start MapForce as COM server.
         mapforce = new Application();
         // COM servers start up invisible so we make it visible
         mapforce.setVisible(true);

         // The following lines attach to the application events using a default
implementation
         // for the events and override one of its methods.
         // If you want to override all document events it is better to derive
your listener class
         // from DocumentEvents and implement all methods of this interface.
         mapforce.addListener(new ApplicationEventsDefaultHandler()
```



```
            {
               @Override
               public void onDocumentOpened(Document i_ipDoc) throws
AutomationException
               {
                  String name = i_ipDoc.getName();

                  if (name.length() > 0)
                     System.out.println("Document " + name + " was opened.");
                  else
                     System.out.println("A new mapping was created.");
               }
            });

            // Locate samples installed with the product.
            String strExamplesFolder = System.getenv("USERPROFILE") + "\\My
Documents\\Altova\\MapForce2012\\MapForceExamples\\";
            // create a new MapForce mapping and generate c++ code
            Document newDoc = mapforce.newMapping();
            ErrorMarkers err1 =
newDoc.generateCodeEx(ENUMProgrammingLanguage.eCpp);
            display(err1);
            // open CompletePO.mfd and generate c++ code
            Document doc = mapforce.openDocument(strExamplesFolder +
"CompletePO.mfd");
            ErrorMarkers err2 = doc.generateCodeEx(ENUMProgrammingLanguage.eCpp);
            display(err2);

            doc.close();
            doc = null;

            System.out.println("Watch MapForce!");
         }
         catch (AutomationException e)
         {
            // e.printStackTrace();
         }
         finally
         {
            // Make sure that MapForce can shut down properly.
            if (mapforce != null)
               mapforce.dispose();

            // Since the COM server was made visible and still is visible, it will
keep running
            // and needs to be closed manually.
            System.out.println("Now close MapForce!");
         }
      }

      public static void display(ErrorMarkers err) throws AutomationException
      {
         Iterator<ErrorMarker> itr = err.iterator();

         if (err.getCount() == 0)
             System.out.print("Code generation completed successfully.\n");
```

```
      while (itr.hasNext())
      {
         String sError = "";
          Object element = itr.next();
          if (element instanceof ErrorMarker)
             sError = ((ErrorMarker)element).getText();
          System.out.print("Error text: " + sError + "\n");
      }
   }
}
```

17.1.6.3 JScript

This section contains listings of JScript code that demonstrate the following basic functionality:

Example files

The code is available in example files that you can test as is or modify to suit your needs. The JScript example files are located in the `JScript` folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/MapForce2016/MapForceExamples/API/`
Windows Vista, Windows 7/8	`C:/Users/<username>/Documents/ Altova/MapForce2016/MapForceExamples/API/`

The example files can be run in one of two ways:

- *From the command line:*
  Open a command prompt window and type the name of one of the example scripts (for example, `Start.js`). The Windows Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7, 8) will execute the script.
- *From Windows Explorer:*
  In Windows Explorer, browse for the JScript file and double-click it. The Windows Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script. After the script is executed, the command console gets closed automatically.

`DocumentAccess.js`	Shows how to open, iterate and close documents
`GenerateCode.js`	Shows how to invoke code generation using JScript
`Start.js`	Start Mapforce registered as an automation server or connect to a running instance.
`Readme.txt`	This file.

Start application

The JScript below starts the application and shuts it down. If an instance of the application is already running, the running instance will be returned.

Script listing

The JScript listing below is explained with comments in the code.

```
// Initialize application's COM object. This will start a new instance of the
application and
// return its main COM object. Depending on COM settings, a the main COM object
of an already
// running application might be returned.
try
{
   objMapForce = WScript.GetObject("", "MapForce.Application");
}
catch(err)
{
   Exit("Can't access or create MapForce.Application");
}

// if newly started, the application will start without its UI visible. Set it
to visible.
objMapForce.Visible = true;

WScript.Echo(objMapForce.Edition + " has successfully started. ");

objMapForce.Visible = false; // will shutdown application if it has no more COM
connections
//objMapForce.Visible = true;   // will keep application running with UI visible
```

Running the script

The JScript code listed above is available in the file `Start.js` located in the `JScript` folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/MapForce2016/MapForceExamples/API/`
Windows Vista, Windows 7/8	`C:/Users/<username>/Documents/ Altova/MapForce2016/MapForceExamples/API/`

To run the script, start it from a command prompt window or from Windows Explorer.

Simple document access

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and close documents.

Script listing

The JScript listing below is explained with comments in the code.

```
// Initialize application's COM object. This will start a new instance of the
application and
// return its main COM object. Depending on COM settings, a the main COM object
of an already
// running application might be returned.
try
{
   objMapForce = WScript.GetObject("", "MapForce.Application");
}
catch(err)
{
   Exit("Can't access or create MapForce.Application");
}

// if newly started, the application will start without its UI visible. Set it
to visible.
objMapForce.Visible = true;

// *************************** code snippet for "Simple Document Access"
***************************************

// Locate examples via USERPROFILE shell variable. The path needs to be adapted
to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\MapForce2012\\MapForceExamples\\";

objMapForce.Documents.OpenDocument(strExampleFolder + "CompletePO.mfd");
objMapForce.Documents.OpenDocument(strExampleFolder +
"Altova_Hierarchical_DB.mfd");

// *************************** code snippet for "Simple Document Access"
***************************************

// *************************** code snippet for "Iteration"
***************************************

// go through all open documents using a JScript Enumerator
for (var iterDocs = new Enumerator(objMapForce.Documents); !iterDocs.atEnd();
iterDocs.moveNext())
{
   objName = iterDocs.item().Name;
   WScript.Echo("Document name: " + objName);
}
```

```
// go through all open documents using index-based access to the document
collection
for (i = objMapForce.Documents.Count; i > 0; i--)
   objMapForce.Documents.Item(i).Close();


// *************************** code snippet for "Iteration"
**************************************

//objMapForce.Visible = false;      // will shutdown application if it has no
more COM connections
objMapForce.Visible = true;   // will keep application running with UI visible
```

Running the script

The JScript code listed below is available in the file `DocumentAccess.js` located in the `JScript` folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/MapForce2016/MapForceExamples/API/`
Windows Vista, Windows 7/8	`C:/Users/<username>/Documents/ Altova/MapForce2016/MapForceExamples/API/`

To run the script, start it from a command prompt window or from Windows Explorer.

Generate code

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and generate C++ code.

Script listing

The JScript listing below is explained with comments in the code.

```
// Initialize application's COM object. This will start a new instance of the
application and
// return its main COM object. Depending on COM settings, a the main COM object
of an already
// running application might be returned.
try
{
   objMapForce = WScript.GetObject("", "MapForce.Application");
}
catch(err)
{
   Exit("Can't access or create MapForce.Application");
}

// if newly started, the application will start without its UI visible. Set it
```

```
to visible.
objMapForce.Visible = true;

// ************************** code snippet for "Simple Document Access"
**************************************

// Locate examples via USERPROFILE shell variable. The path needs to be adapted
to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\MapForce2012\\MapForceExamples\\";

objMapForce.Documents.OpenDocument(strExampleFolder + "CompletePO.mfd");
//objMapForce.Documents.OpenDocument(strExampleFolder +
"Altova_Hierarchical_DB.mfd");
objMapForce.Documents.NewDocument();

// ************************** code snippet for "Simple Document Access"
**************************************

// ************************** code snippet for "Iteration"
**************************************

objText = "";
// go through all open documents using a JScript Enumerator and generate c++
code
for (var iterDocs = new Enumerator(objMapForce.Documents); !iterDocs.atEnd();
iterDocs.moveNext())
{
   objText += "Generated c++ code result for document " + iterDocs.item().Name
+ " :\n";
   objErrorMarkers = iterDocs.item().generateCodeEx(1); //
ENUMProgrammingLanguage.eCpp = 1

   bSuccess = true;
   for (var iterErrorMarkers = new
Enumerator(objErrorMarkers); !iterErrorMarkers.atEnd();
iterErrorMarkers.moveNext())
   {
      bSuccess = false;
      objText += "\t" + iterErrorMarkers.item().Text + "\n";
   }

   if (bSuccess)
       objText += "\tCode generation completed successfully.\n";

   objText += "\n";
}

WScript.Echo(objText);

// go through all open documents using index-based access to the document
collection
for (i = objMapForce.Documents.Count; i > 0; i--)
   objMapForce.Documents.Item(i).Close();
```

```
// ************************** code snippet for "Iteration"
**************************************

//objMapForce.Visible = false;      // will shutdown application if it has no
more COM connections
objMapForce.Visible = true;   // will keep application running with UI visible
```

Running the script

The JScript code listed below is available in the file `GenerateCode.js` located in the `JScript` folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/MapForce2016/MapForceExamples/API/`
Windows Vista, Windows 7/8	`C:/Users/<username>/Documents/ Altova/MapForce2016/MapForceExamples/API/`

To run the script, start it from a command prompt window or from Windows Explorer.

17.2 Object Reference

Object Hierarchy

```
Application
   Options
   Project
      ProjectItem
   Documents
      Document
         MapForceView
         Mapping
            Component
               Datapoint
            Components
            Connection
         Mappings
   ErrorMarkers
      ErrorMarker
AppOutputLines
   AppOutputLine
      AppOutputLines
         ...
      AppOutputLineSymbol

Enumerations
```

Description
This section contains the reference of the MapForce API 3.0 Type Library.

17.2.1 Application

The Application interface is the interface to a MapForce application object. It represents the main access point for the MapForce application itself.
This interface is the starting point to do any further operations with MapForce or to retrieve or create other MapForce related automation objects.

Events
Events

Properties and Methods
Properties to navigate the object model:
Application
Parent
Options
Project
Documents

Application status:

Visible
Name
Quit
Status
WindowHandle

MapForce designs:
NewDocument
OpenDocument
OpenURL
ActiveDocument

MapForce projects:
NewProject (Enterprise or Professional edition is required)
OpenProject (Enterprise or Professional edition is required)
ActiveProject (Enterprise or Professional edition is required)

MapForce code generation:
HighlightSerializedMarker

Global resources:
GlobalResourceConfig
GlobalResourceFile

Version information:
Edition
IsAPISupported
MajorVersion
MinorVersion

Examples
The following examples show how the automation interface of MapForce can be accessed from different programming environments in different languages.

```
' ------------------- begin VBA example ---------------------
' create a new instance of <SPY-MAP>.
Dim objMapForce As Application
Set objMapForce = CreateObject("MapForce.Application")
' -------------------- end example ---------------------

' ------------------- begin VBScript example ---------------------
' access a running, or create a new instance of MapForce.
' works with scripts running in the Windows scripting host.
Set objMapForce = GetObject("MapForce.Application");
' -------------------- end example ---------------------

// ------------------- begin JScript example ---------------------
// Access a running, or create a new instance of <MapForce
// works with scripts executed in the Windows scripting host
try
{
	objMapForce = WScript.GetObject ("", "MapForce.Application");
	// unhide application if it is a new instance
```

```
        objMapForce.Visible = true;
}
catch(err) { WScript.Echo ("Can't access or create MapForce.Application"); }
// -------------------- end example ---------------------
```

17.2.1.1 Events

This object supports the following events:

OnDocumentOpened
OnProjectOpened
OnShutdown

OnDocumentOpened

Event: OnDocumentOpened (*i_objDocument* as Document)

Description
This event is triggered when an existing or new document is opened. The corresponding close event is Document.OnDocumentClosed.

OnProjectOpened

Event: OnProjectOpened (*i_objProject* as Project)

Description
This event is triggered when an existing or new project is loaded into the application. The corresponding close event is Project.OnProjectClosed.

OnShutdown

Event: OnShutdown ()

Description
This event is triggered when the application is shutting down.

17.2.1.2 ActiveDocument

Property: ActiveDocument as Document (read-only)

Description
Returns the automation object of the currently active document. This property returns the same

as `Documents.ActiveDocument`.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.3 ActiveProject

Property: `ActiveProject` as `Project` (read-only)

Description
Returns the automation object of the currently active project.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.4 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.5 Documents

Property: `Documents` as `Documents` (read-only)

Description
Returns a collection of all currently open documents.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.6 Edition

Property: `Edition` as `String` (read-only)

Description
Returns the edition of the application, e.g. "Altova MapForce Enterprise Edition" for the enterprise

edition.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.7 GlobalResourceConfig

Property: `GlobalResourceConfig` as `String`

Description
Gets or sets the name of the active global resource configuration file. Per default, the file is called GlobalResources.xml.

The configuration file can be renamed and saved to any location. You can therefore have multiple Global Resources XML files. However, only one of these Global Resources XML File can be active, per application, at one time, and only the definitions contained in this file will be available to the application.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.8 GlobalResourceFile

Property: `GlobalResourceFile` as `String`

Description
Gets or sets the global resource definition file. Per default the file is called GlobalResources.xml.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.9 HighlightSerializedMarker

Method: `HighlightSerializedMarker` (*`i_strSerializedMarker`* as `String`)

Description
Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document has not already been loaded, it will be loaded first. See Document.GenerateCodeEx for a method to retrieve a serialized marker.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

1007 The string passed in *i_strSerializedMarker* is not recognized as a serialized MapForce marker.
1008 The marker points to a location that is no longer valid.

17.2.1.10 IsAPISupported

Property: `IsAPISupported` as `Boolean` (read-only)

Description
Returns whether the API is supported in this version of MapForce.

Errors
1001 Invalid address for the return parameter was specified.

17.2.1.11 MajorVersion

Property: `MajorVersion` as `Long` (read-only)

Description
The major version number of the product, e.g. 2006 for 2006 R2 SP1.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.12 MinorVersion

Property: `MinorVersion` as `Long` (read-only)

Description
The minor version number of the product, e.g. 2 for 2006 R2 SP1.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.13 Name

Property: `Name` as `String` (read-only)

Description
The name of the application.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.14 NewDocument

Method: `NewDocument` () as `Document`

Description
Creates a new empty document. The newly opened document becomes the `ActiveDocument`. This method is a shortened form of `Documents.NewDocument`.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.15 NewProject

Method: `NewProject` () as `Project`

Description
Creates a new empty project. The current project is closed. The new project is accessible under `ActiveProject`.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.16 OpenDocument

Method: `OpenDocument` (*`i_strFileName`* as `String`) as `Document`

Description
Loads a previously saved document file and continues working on it. The newly opened document becomes the `ActiveDocument`. This method is a shorter form of `Documents.OpenDocument`.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.17 OpenProject

Method: `NewProject` () as `Project`

Description
Opens an existing Mapforce project (`*.mfp`). The current project is closed. The newly opened project is accessible under `ActiveProject`.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.18 OpenURL

Method: `OpenURL` (*`i_strURL`* as `String`, *`i_strUser`* as `String`, *`i_strPassword`* as `String`)

Description
Loads a previously saved document file from an URL location. Allows user name and password to be supplied.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.19 Options

Property: `Options` as `Options` (read-only)

Description
This property gives access to options that configure the generation of code.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.20 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors
1000 The object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.21 Quit

Method: `Quit` ()

Description
Disconnects from MapForce to allow the application to shutdown. Calling this method is optional since MapForce keeps track of all external COM connections and automatically recognizes a disconnection. For more information on automatic shutdown see the `Visible` property.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.22 ServicePackVersion

Property: `ServicePackVersion` as `Long` (read-only)

Description
The service pack version number of the product, e.g. 1 for 2010 R2 SP1.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

17.2.1.23 Status

Property: `Status` as `Long` (read-only)

Description
The status of the application. It is one of the values of the `ENUMApplicationStatus` enumeration.

Errors
1001 Invalid address for the return parameter was specified.

17.2.1.24 Visible

Property: `Visible` as `Boolean`

Description
`True` if MapForce is displayed on the screen (though it might be covered by other applications or be iconized).

`False` if MapForce is hidden. The default value for MapForce when automatically started due to a request from the automation server `MapForce.Application` is `false`. In all other cases, the

property is initialized to `true`.

An application instance that is visible is said to be controlled by the user (and possibly by clients connected via the automation interface). It will only shut down due to an explicit user request. To shut down an application instance, set its visibility to false and clear all references to this instance within your program. The application instance will shut down automatically when no further COM clients are holding references to it.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.2.1.25 WindowHandle

Property: `WindowHandle` () as `long` (read-only)

Description
Retrieve the application's Window Handle.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

17.2.2 AppOutputLine

Represents a message line. In contrast to ErrorMarker, its structure is more detailed and can contain a collection of child lines, therefore forming a tree of message lines.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Line access:
`GetLineSeverity`
`GetLineSymbol`
`GetLineText`
`GetLineTextEx`
`GetLineTextWithChildren`
`GetLineTextWithChildrenEx`

A single AppOutputLine consists of one or more sub-lines.
Sub-line access:
`GetLineCount`

A sub-line consists of one or more cells.
Cell access:
`GetCellCountInLine`
`GetCellIcon`

GetCellSymbol
GetCellText
GetCellTextDecoration
GetIsCellText

Below an AppOutputLine there can be zero, one, or more child lines which themselves are of type AppOutputLine, which thus form a tree structure.

Child lines access:
ChildLines

17.2.2.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.2 ChildLines

Property: ChildLines as AppOutputLines (read-only)

Description
Returns a collection of the current line's direct child lines.

Errors
4100 The application object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.3 GetCellCountInLine

Method: GetCellCountInLine (*nLine* as Long) as Long

Description
Gets the number of cells in the sub-line indicated by *nLine* in the current AppOutputLine.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.4 GetCellIcon

Method: `GetCellIcon` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `Long`

Description
Gets the icon of the cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine*.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.5 GetCellSymbol

Method: `GetCellSymbol` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `AppOutputLineSymbol`

Description
Gets the symbol of the cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine*.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.6 GetCellText

Method: `GetCellText` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `String`

Description
Gets the text of the cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine*.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.7 GetCellTextDecoration

Method: `GetCellTextDecoration` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `Long`

Description
Gets the decoration of the text cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine*.
It can be one of the `ENUMAppOutputLine_TextDecoration` values.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.8 GetIsCellText

Method: `GetIsCellText` (*`nLine`* as `Long`, *`nCell`* as `Long`) as `Boolean`

Description

Returns true, if the cell indicated by *nCell* in the current AppOutputLine's sub-line indicated by *nLine* is a text cell.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.9 GetLineCount

Method: `GetLineCount` () as `Long`

Description

Gets the number of sub-lines the current line consists of.

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.10 GetLineSeverity

Method: `GetLineSeverity` () as `Long`

Description

Gets the severity of the line. It can be one of the `ENUMAppOutputLine_Severity` values:

Errors

4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.11 GetLineSymbol

Method: `GetLineSymbol` () as `AppOutputLineSymbol`

Description

Gets the symbol assigned to the whole line.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.12 GetLineText

Method: `GetLineText` () as `String`

Description
Gets the contents of the line as text.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.13 GetLineTextEx

Method: `GetLineTextEx` (*`psTextPartSeperator`* as `String`, *`psLineSeperator`* as `String`) as `String`

Description
Gets the contents of the line as text using the specified part and line separators.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.14 GetLineTextWithChildren

Method: `GetLineTextWithChildren` () as `String`

Description
Gets the contents of the line including all child and descendant lines as text.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.15 GetLineTextWithChildrenEx

Method: `GetLineTextWithChildrenEx` (*`psPartSep`* as `String`, *`psLineSep`* as `String`, *`psTabSep`* as `String`, *`psItemSep`* as `String`) as `String`

Description
Gets the contents of the line including all child and descendant lines as text using the specified part, line, tab and item separators.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.2.16 Parent

Property: `Parent` as `AppOutputLines` (read-only)

Description
The parent object according to the object model.

Errors
4100 The object is no longer valid.
4101 Invalid address for the return parameter was specified.

17.2.3 AppOutputLines

Represents a collection of AppOutputLine message lines.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Iterating through the collection:
`Count`
`Item`

17.2.3.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
4000 The object is no longer valid.
4001 Invalid address for the return parameter was specified.

17.2.3.2 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of lines in the collection.

Errors
4000 The object is no longer valid.
4001 Invalid address for the return parameter was specified.

17.2.3.3 Item

Property: `Item` (*`nIndex`* as `Integer`) as `AppOutputLine` (read-only)

Description
Retrieves the line at `nIndex` from the collection. Indices start with 1.

Errors
4000 The object is no longer valid.
4001 Invalid address for the return parameter was specified.

17.2.3.4 Parent

Property: `Parent` as `AppOutputLine` (read-only)

Description
The parent object according to the object model.

Errors
4000 The object is no longer valid.
4001 Invalid address for the return parameter was specified.

17.2.4 AppOutputLineSymbol

An AppOutputLineSymbol represents a link in an AppOutputLine message line which can be clicked in the MapForce Messages window.
It is applied to a cell of an AppOutputLine or to the whole line itself.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Access to AppOutputLineSymbol methods:
GetSymbolHREF
GetSymbolID
IsSymbolHREF

17.2.4.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
4200 The object is no longer valid.
4201 Invalid address for the return parameter was specified.

17.2.4.2 GetSymbolHREF

Method: GetSymbolHREF () as String

Description
If the symbol is of type URL, returns the URL as a string.

Errors
4200 The object is no longer valid.
4201 Invalid address for the return parameter was specified.

17.2.4.3 GetSymbolID

Method: GetSymbolHREF () as Long

Description
Gets the ID of the symbol.

Errors
4200 The object is no longer valid.
4201 Invalid address for the return parameter was specified.

17.2.4.4 IsSymbolHREF

Method: IsSymbolHREF () as Boolean

Description
Indicates if the symbol is of kind URL.

Errors
4200 The object is no longer valid.
4201 Invalid address for the return parameter was specified.

17.2.4.5 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors
4200 The object is no longer valid.
4201 Invalid address for the return parameter was specified.

17.2.5 Component

A Component represents a MapForce component.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Component properties:
`HasIncomingConnections`
`HasOutgoingConnections`
`CanChangeInputInstanceFile`
`CanChangeOutputInstanceFile`
`ComponentName`

`ID`
`IsParameterInputRequired`
`IsParameterSequence`
`Name`
`Preview`
`Schema`
`SubType`
`Type`

Instance related properties:
`InputInstanceFile`
`OutputInstanceFile`

Datapoints:
`GetRootDatapoint`

Execution:

GenerateOutput

17.2.5.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.2 CanChangeInputInstanceFile

Property: CanChangeInputInstanceFile as Boolean (read-only)

Description
Indicates if the input instance file name can be changed.

Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true.
If the component does not have a filename node, false is returned.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.3 CanChangeOutputInstanceFile

Property: CanChangeOutputInstanceFile as Boolean (read-only)

Description
Indicates if the output instance file name can be changed.

Returns false if the component has a filename node and this node has a connection on its left (input) side, otherwise returns true.
If the component does not have a filename node, false is returned.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.4 ComponentName

Property: ComponentName as `String`

Description
Gets or sets the component's name.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1246	The component does not support setting its name.
1247	Invalid component name.

17.2.5.5 GenerateOutput

Method: `GenerateOutput` (*[out]* *`pbError`* as `Boolean`) as `AppOutputLines`

Description
Generates the output file(s) defined in the mapping for the current component only, using a MapForce internal mapping language. The name(s) of the output file(s) are defined as property of the current component which is the output item in the mapping for this generation process.

Remarks
pbError is an output-only parameter. You will receive a value only if the calling language supports output parameters. If not, the value you pass here will remain unchanged when the function has finished.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1248	Generating output is only supported when the graphical user interface is visible.

17.2.5.6 GetRootDatapoint

Method: GetRootDatapoint(side as ENUMComponentDatapointSide, strNamespace as String, strLocalName as String, strParameterName as String) as Datapoint

Description
Gets a root datapoint on the left (input) or right (output) side of a component. To access children and descendants, the Datapoint object provides further methods.

The *side* parameter indicates if an input, or output, datapoint of a component is to be retrieved.

The specified namespace and local name, indicate the specific name of the node whose datapoint is to be retrieved. For components with structural information such as schema components, you will have to provide the namespace together with the local name, or you can just pass an empty string for the namespace.

File-based components like the schema component contain a special node on their root, the filename node. There, GetRootDatapoint can only find the filename node. You will have to pass namespace "`http://www.altova.com/mapforce`" and local name "`FileInstance`" to retrieve a datapoint of this node.

The specified parameter name should be an empty string unless the component in question is a function call component . Since a user-defined function might contain input or output parameters of the same structure, the function call component calling this user-defined function can have more than one root node with an identical namespace and local name.

They will then differ only by their parameter names, which are in fact the names of the according parameter components in the user-defined function mapping itself.

It is not mandatory to specify the parameter name, though. In that case, the method will return the first root datapoint matching the specified namespace and local name.

Errors

1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.
1231 Datapoint not found.

17.2.5.7 HasIncomingConnections

Property: `HasIncomingConnections` as `Boolean` (read-only)

Description

Indicates if the component has any incoming connections (on its left side) not including the filename node. An incoming connection on the filename node does not have any effect on the returned value.

Errors

1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.8 HasOutgoingConnections

Property: `HasOutgoingConnections` as `Boolean` (read-only)

Description

Indicates if the component has any outgoing connections (on its right side).

Errors

1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.9 ID

Property: `ID` as `Unsigned Long` (read-only)

Description
Retrieves the component ID.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.10 InputInstanceFile

Property: `InputInstanceFile` as `String`

Description
Gets or sets the component's input instance file.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.11 IsParameterInputRequired

Property: IsParameterInputRequired as `Boolean`

Description
Gets or sets, if the input parameter component requires an ingoing connection on the function call component of the user-defined function this input parameter component is in.
This property works only for components, which are input parameter components.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.
1232 This operation works only for an input parameter component.
1240 Changing the document not allowed. It is read-only.

17.2.5.12 IsParameterSequence

Property: IsParameterSequence as `Boolean`

Description
Gets or sets, if the input or output parameter component supports sequences.
This property works only for components, which are input or output parameter components.

Errors

1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.
1233 This operation works only for an input or output parameter component.
1240 Changing the document not allowed. It is read-only.

17.2.5.13 Name

Property: `Name` as `String (read only)`

Description
Gets the component's name.

Errors

1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.14 OutputInstanceFile

Property: `OutputInstanceFile` as `String`

Description
Gets or sets the component's output instance file.

Trying to access the OutputInstanceFile of a component via the API does not return any data if the "File" connector of the component has been connected to another item in the mapping.

Errors

1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.15 Parent

Property: `Parent` as `Mapping` (read-only)

Description
The parent object according to the object model.

Errors

1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.16 Preview

Property: Preview as `Boolean`

Description
Gets or sets, if the component is the current preview component.

This property works only for components, which are target components in the document's main mapping. Only one target component in the main mapping can be the preview component at any time.

When setting this property, it is only possible to set it to true. This then will also implicitly set the Preview property of all other components to false.

If there is just a single target component in the main mapping, it is also the preview component.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.
1234	Only a target component in the main mapping can be set as preview component.
1235	A component cannot be set as non-preview component. Set another component as preview component instead.

17.2.5.17 Schema

Property: `Schema` as `String` (read-only)

Description
Retrieves the component's schema file name.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.2.5.18 SubType

Property: `SubType` as `ENUMComponentSubType` (read-only)

Description
Retrieves the component's sub type.

Errors

1200	The object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.2.5.19 Type

Property: `Type` as `ENUMComponentType` (read-only)

Description
Retrieves the component's type.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.5.20 UsageKind

Property: `UsageKind` as `ENUMUsageKind` (read-only)

Description
Retrieves the component's usage kind.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.6 Components

Represents a collection of Component objects.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Iterating through the collection:
`Count`
`Item`

17.2.6.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.6.2 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of components in the collection.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.6.3 Item

Property: `Item` (*`nIndex`* as `Integer`) as `Component` (read-only)

Description
Retrieves the component at `nIndex` from the collection. Indices start with 1.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.6.4 Parent

Property: `Parent` as `Mapping` (read-only)

Description
The parent object according to the object model.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.7 Connection

A Connection object represents a connector between two components.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Properties

ConnectionType

17.2.7.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
2100 The object is no longer valid.
2101 Invalid address for the return parameter was specified.

17.2.7.2 ConnectionType

Property: ConnectionType as ENUMConnectionType

Description
Gets or sets the connection's type.

Errors
2100 The application object is no longer valid.
2101 Invalid address for the return parameter was specified.
2102 Changing the document not allowed. It is read-only.
2103 Failed changing connection type.

17.2.7.3 Parent

Property: Parent as Mapping (read-only)

Description
The parent object according to the object model.

Errors
2100 The object is no longer valid.
2101 Invalid address for the return parameter was specified.

17.2.8 Datapoint

A Datapoint object represents an input or output icon of a component.

Properties and Methods
Properties to navigate the object model:
Application

Parent

Methods
GetChild

17.2.8.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
2000 The object is no longer valid.
2001 Invalid address for the return parameter was specified.

17.2.8.2 GetChild

Method: GetChild(strNamespace as String, strLocalName as String, searchFlags as ENUMSearchDatapointFlags) as Datapoint

Description
Scans for a direct child datapoint of the current datapoint, by namespace and local name.

Search flags can be passed as combination of values (combined using binary OR) of the ENUMSearchDatapointFlags enumeration.

A schema component with elements that contain mixed content, each display an additional child node, the so-called text() node. To retrieve a datapoint of a text() node, you will have to pass an empty string in strNamespace as well as `"#text"` in strLocalName and eSearchDatapointElement in searchFlags.

Errors
2000 The application object is no longer valid.
2001 Invalid address for the return parameter was specified.
2002 Datapoint not found.

17.2.8.3 Parent

Property: Parent as Component (read-only)

Description
The parent object according to the object model.

Errors
2000 The object is no longer valid.

2001 Invalid address for the return parameter was specified.

17.2.9 Document

A Document object represents a MapForce document (a loaded MFD file).
A document contains a main mapping and zero or more local user-defined-function mappings.

Events
Events

Properties and Methods
Properties to navigate the object model:
Application
Parent

File handling:
Activate
Close
FullName
Name
Path
Saved
Save
SaveAs

Mapping handling:
MainMapping
Mappings
CreateUserDefinedFunction

Component handling:
FindComponentByID

Code generation:
OutputSettings_ApplicationName
JavaSettings_BasePackageName

GenerateCHashCode
GenerateCodeEx
GenerateCppCode
GenerateJavaCode
GenerateXQuery
GenerateXSLT
GenerateXSLT2
HighlightSerializedMarker

Mapping execution:
GenerateOutput
GenerateOutputEx

View access:

MapForceView

Obsolete:
OutputSettings_Encoding

17.2.9.1 Events

This object supports the following events:

OnDocumentClosed
OnModifiedFlagChanged

OnDocumentClosed

Event: OnDocumentClosed (*i_objDocument* as Document)

Description
This event is triggered when a document is closed. The document object passed into the event handler should not be accessed. The corresponding open event is Application.OnDocumentOpened.

OnModifiedFlagChanged

Event: OnModifiedFlagChanged (*i_bIsModified* as Boolean)

Description
This event is triggered when a document's modification status changes.

17.2.9.2 Activate

Method: Activate ()

Description
Makes this document the active document.

Errors
1200 The application object is no longer valid.

17.2.9.3 Application

Property: Application as Application (read-only)

Description

Retrieves the application's top-level object.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.4 Close

Method: `Close` ()

Description
Closes the document without saving.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.5 CreateUserDefinedFunction

Method: CreateUserDefinedFunction(strFunctionName as String, strLibraryName as String, strSyntax as String, strDetails as String, blnlinedUse as Boolean) as Mapping

Description
Creates a user defined function in the current document.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1208 Failed creating user-defined function.
1209 Changing the document not allowed. It is read-only.

17.2.9.6 FindComponentByID

Method: `FindComponentByID` (*nID* as `Unsigned Long`) as `Component`

Description
Searches in the whole document, so all its mappings, for the component with the specified id.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.7 FullName

Property: `FullName` as St r i ng

Description
Path and name of the document file.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.8 GenerateCHashCode

Method: `GenerateCHashCode` ()

Description
Generate C# code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

See also
Code Generation

17.2.9.9 GenerateCodeEx

Method: `GenerateCodeEx` (*i_nLanguage* as `ENUMProgrammingLanguage`) as `ErrorMarkers`

Description
Generates code that will perform the mapping. The parameter *i_nLanguage* specifies the target language. The method returns an object that can be used to enumerate all messages created by the code generator. These are the same messages that get displayed in the Messages window of MapForce.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

See also
Code Generation

17.2.9.10 GenerateCppCode

Method: `GenerateCppCode` ()

Description
Generates C++ code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

See also
`Code Generation`

17.2.9.11 GenerateJavaCode

Method: `GenerateJavaCode` ()

Description
Generates Java code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

See also
`Code Generation`

17.2.9.12 GenerateOutput

Method: `GenerateOutput` ()

Description
Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1206 Error during execution of mapping algorithm.
1210 Generating output is only supported when the graphical user interface is visible.

This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.

See also
Code Generation

17.2.9.13 GenerateOutputEx

Method: `GenerateOutputEx` () as `AppOutputLines`

Description
Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping. This method is identical to `GenerateOutput` except for its return value containing the resulting messages, warnings and errors arranged as trees of `AppOutputLine`s.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1206	Error during execution of mapping algorithm.
1210	Generating output is only supported when the graphical user interface is visible.

This method can only be used when the MapForce (running as a COM server) main window is visible, or is embedded with a graphical user interface. If the method is called while MapForce is not visible, then an error will occur.

See also
Code Generation

17.2.9.14 GenerateXQuery

Method: `GenerateXQuery` ()

Description
Generates mapping code as XQuery. Uses the properties defined in `Application.Options` to configure code generation.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1204	Error during XSLT/XSLT2/XQuery code generation.

See also
Code Generation

17.2.9.15 GenerateXSLT

Method: `GenerateXSLT` ()

Description
Generates mapping code as XSLT. Uses the properties defined in `Application.Options` to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1204 Error during XSLT/XSLT2/XQuery code generation.

See also
`Code Generation`

17.2.9.16 GenerateXSLT2

Method: `GenerateXSLT2` ()

Description
Generates mapping code as XSLT2. Uses the properties defined in `Application.Options` to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1204 Error during XSLT/XSLT2/XQuery code generation.

See also
`Code Generation`

17.2.9.17 HighlightSerializedMarker

Method: `HighlightSerializedMarker` (*`i_strSerializedMarker`* as `String`)

Description
Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document is not already loaded, it will be loaded first. See `GenerateCodeEx` for a method to retrieve a serialized marker.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.
1007 The string passed in *i_strSerializedMarker* is not recognized a serialized MapForce marker.
1008 The marker points to a location that is no longer valid.

17.2.9.18 JavaSettings_BasePackageName

Property: `JavaSettings_BasePackageName` as `String`

Description
Sets or retrieves the base package name used when generating Java code. This property is available in UI-dialog for the Document Settings.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

See also
`Code Generation`

17.2.9.19 MainMapping

Property: `MainMapping` as `Mapping` (read-only)

Description
Retrieves the main mapping of the document.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.20 MapForceView

Property: `MapForceView` as `MapForceView` (read-only)

Description
This property gives access to functionality specific to the MapForce view.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.21 Mappings

Property: `Mappings` as `Mappings` (read-only)

Description
Returns a collection of the mappings contained in the document.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.22 Name

Property: `Name` as `String`

Description
Name of the document file without file path.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.23 OutputSettings_ApplicationName

Property: `OutputSettings_ApplicationName` as `String`

Description
Sets or retrieves the application name available in the Document Settings dialog.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

See also
`Code Generation`

17.2.9.24 OutputSettings_Encoding (obsolete)

Property: `OutputSettings_Encoding` as `String`

Description
obsolete
This property is not supported anymore. Mapping output encoding settings do not exist anymore. Components have individual output encoding settings.

See also
`Code Generation`

17.2.9.25 Parent

Property: `Parent` as `Documents` (read-only)

Description
The parent object according to the object model.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.26 Path

Property: `Path` as `String`

Description
Path of the document file without name.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.27 Save

Method: `Save` ()

Description
Save the document to the file defined by `Document.FullName`.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.28 SaveAs

Method: `SaveAs` (*`i_strFileName`* as `String`)

Description
Save document to specified file name, and set `Document.FullName` to this value if save operation was successful.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.9.29 Saved

Property: `Saved` as `Boolean` (read-only)

Description
Tr ue if the document was not modified since the last save operation, f al se otherwise.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.10 Documents

Represents a collection of Document objects.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Open and create mappings:
`OpenDocument`
`NewDocument`

Iterating through the collection:
`Count`
`Item`
`ActiveDocument`

17.2.10.1 ActiveDocument

Property: `ActiveDocument` as `Document` (read-only)

Description
Retrieves the active document. If no document is open, `null` is returned.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

17.2.10.2 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

17.2.10.3 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of documents in the collection.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

17.2.10.4 Item

Property: `Item` (*`nIndex`* as `Integer`) as `Document` (read-only)

Description
Retrieves the document at `nIndex` from the collection. Indices start with 1.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

17.2.10.5 NewDocument

Method: `NewDocument` () as `Document`

Description
Creates a new document, adds it to the end of the collection, and makes it the active document.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

17.2.10.6 OpenDocument

Method: `OpenDocument` (*`strFilePath`* as `String`) as `Document`

Description
Opens an existing mapping document (`*.mfd`). Adds the newly opened document to the end of the collection and makes it the active document.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

17.2.10.7 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors
1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

17.2.11 ErrorMarker

Represents a simple message line. In difference to AppOutputLine, error markers do not have a hierarchical structure.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Access to message information:
`DocumentFileName`
`ErrorLevel`
`Highlight`
`Serialization`
`Text`

17.2.11.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.

17.2.11.2 DocumentFileName

Property: `DocumentFileName` as `String` (read-only)

Description
Retrieves the name of the mapping file that the error marker is associated with.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.

17.2.11.3 ErrorLevel

Property: `ErrorLevel` as `ENUMCodeGenErrorLevel` (read-only)

Description
Retrieves the severity of the error.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.

17.2.11.4 Highlight

Method: `Highlight` ()

Description
Highlights the item that the error marker is associated with. If the corresponding document is not open, it will be opened.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.
1008 The marker points to a location that is no longer valid.

17.2.11.5 Serialization

Property: `Serialization` as `String` (read-only)

Description
Serialize error marker into a string. Use this string in calls to Application.HighlightSerializedMarker or Document.HighlightSerializedMarker to highlight the marked item in the mapping. The string can be persisted and used in other instantiations of MapForce or its Control.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.

17.2.11.6 Text

Property: `Text` as `String` (read-only)

Description
Retrieves the message text.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.

17.2.11.7 Parent

Property: `Parent` as `ErrorMarkers` (read-only)

Description
The parent object according to the object model.

Errors
1900 The object is no longer valid.
1901 Invalid address for the return parameter was specified.

17.2.12 ErrorMarkers

Represents a collection of ErrorMarker objects.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Iterating through the collection:
`Count`

Item

17.2.12.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
1800 The object is no longer valid.
1801 Invalid address for the return parameter was specified.

17.2.12.2 Count

Property: Count as Integer (read-only)

Description
Retrieves the number of error markers in the collection.

Errors
1800 The object is no longer valid.
1801 Invalid address for the return parameter was specified.

17.2.12.3 Item

Property: Item (*nIndex* as Integer) as ErrorMarker (read-only)

Description
Retrieves the error marker at nIndex from the collection. Indices start with 1.

Errors
1800 The object is no longer valid.
1801 Invalid address for the return parameter was specified.

17.2.12.4 Parent

Property: Parent as Application (read-only)

Description
The parent object according to the object model.

Errors
1800 The object is no longer valid.

1801 Invalid address for the return parameter was specified.

17.2.13 MapForceView

Represents the current view in the MapForce Mapping tab for a document.
A document has exactly one MapForceView which displays the currently active mapping.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

View activation and view properties:
`Active`
`ShowItemTypes`
`ShowLibraryInFunctionHeader`
`HighlightMyConnections`
`HighlightMyConnectionsRecursivly`

Mapping related properties:
`ActiveMapping`
`ActiveMappingName`

Adding items:
`InsertWSDLCall`
`InsertXMLFile`
`InsertXMLSchema`
`InsertXMLSchemaWithSample`

17.2.13.1 Active

Property: `Active` as `Boolean`

Description
Use this property to query if the mapping view is the active view, or set this view to be the active one.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.2 ActiveMapping

Property: ActiveMapping as Mapping

Description

Gets or sets the currently active mapping in the document this MapForceView belongs to.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.3 ActiveMappingName

Property: ActiveMappingName as String

Description
Gets or sets the currently active mapping by name in the document this MapForceView belongs to.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.4 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.5 HighlightMyConnections

Property: `HighlightMyConnections` as `Boolean`

Description
This property defines whether connections from the selected item only should be highlighed.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.6 HighlightMyConnectionsRecursivey

Property: `HighlightMyConnectionsRecursively` as `Boolean`

Description
This property defines if only the connections coming directly or indirectly from the selected item should be highlighed.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.7 InsertWSDLCall

Method: `InsertWSDLCall (i_strWSDLFileName as String)`

Description
Adds a new WSDL call component to the mapping.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.8 InsertXMLFile (obsolete)

Method: `InsertXMLFile (i_strXMLFileName as String, i_strRootElement as String)`

Description
obsolete
MapForceView.InsertXMLFile is obsolete. Use Mapping.InsertXMLFile instead.

Adds a new component to the mapping.

The component's internal structure is determined by the schema referenced in the specified XML file.

The second parameter defines the root element of this schema, if there is more than one candidate.
When passing an empty string as root element, the root element of the xml file will be used. Otherwise if more candidates are available, a Select Root Element dialog will pop up regardless if MapForce is visible or not.

The specified XML file is used as the input sample to evaluate the mapping.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.9 InsertXMLSchema (obsolete)

Method: `InsertXMLSchema (i_strSchemaFileName as String, i_strRootElement as String)`

Description
obsolete
MapForceView.InsertXMLSchema is obsolete. Use Mapping.InsertXMLSchema instead.

Adds a new component to the mapping.

The component's internal structure is determined by the specified schema file.

The second parameter defines the root element of this schema if there is more than one candidate.
If the passed root element is an empty string and more candidates are available, a Select Root Element dialog will pop up regardless if MapForce is visible or not.

No XML input sample is assigned to this component.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.2.13.10 InsertXMLSchemaWithSample (obsolete)

Method: `InsertXMLSchemaWithSample (i_strSchemaFileName as String, i_strXMLSampleName as String, i_strRootElement as String)`

Description
obsolete
MapForceView.InsertXMLSchemaWithSample is obsolete. Use Mapping.InsertXMLFile instead. Notice, Mapping.InsertXMLFile does not require a parameter for passing the root element. The root element is automatically set as the xml file's root element name.

Adds a new component to the mapping.

The component's internal structure is determined by the specified schema file.

The second parameter is stored as the XML input sample for mapping evaluation.

The third parameter defines the root element of this schema if there is more than one candidate. When passing an empty string as root element, the root element of the xml sample file will be used.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

17.2.13.11 Parent

Property: `Parent` as `Document` (read-only)

Description
The parent object according to the object model.

Errors
1300 The object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.12 ShowItemTypes

Property: `ShowItemTypes` as `Boolean`

Description
This property defines if types of items should be shown in the mapping diagram.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.13.13 ShowLibraryInFunctionHeader

Property: `ShowLibraryInFunctionHeader` as `Boolean`

Description
This property defines whether the name of the function library should be part of function names.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

17.2.14 Mapping

A Mapping object represents a mapping in a document, so the main mapping, or a local user-defined-function mapping.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Mapping properties:
`IsMainMapping`

Name

Components in the mapping:
Components

Adding items:
CreateConnection
InsertFunctionCall
InsertXMLFile
InsertXMLSchema
InsertXMLSchemaInputParameter
InsertXMLSchemaOutputParameter

17.2.14.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.14.2 Components

Property: Components as Components (read-only)

Description
Returns a collection of all components in the current mapping.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.14.3 CreateConnection

Method: CreateConnection(DatapointFrom as Datapoint, DatapointTo as Datapoint) as Connection

Description
Creates a connection between the two supplied datapoints (DatapointFrom & DatapointTo).

It will fail to do so if the DatapointFrom is not an output-side datapoint, the DatapointTo is not an input-side datapoint, or a connection between these two datapoints already exists.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1241	Failed creating the connection.

17.2.14.4 InsertFunctionCall

Method: InsertFunctionCall(strFunctionName as String, strLibraryName as String) as Component

Description

Inserts a function call component into the current mapping.

The specified library and function names indicate the function or user-defined function to be called.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1242	Failed creating function call component.

17.2.14.5 InsertXMLFile

Method: InsertXMLFile(i_strFileName as String, i_strSchemaFileName as String) as Component

Description

Adds a new XML schema component to the mapping.

The component's internal structure is determined by the schema referenced in the specified XML file (i_strFileName) or, if the XML file does not reference a schema file, by the separately specified schema file (i_strSchemaFileName).

If the XML file has a schema file reference, then the parameter i_strSchemaFileName is ignored.

The root element of the XML file will be used in the component.

The specified XML file is used as the input sample to evaluate the mapping.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1244	Failed creating component.

17.2.14.6 InsertXMLSchema

Method: InsertXMLSchema(i_strSchemaFileName as String, i_strXMLRootName as String) as Component

Description
Adds a new XML schema component to the mapping.

The component's internal structure is determined the specified schema file.

The second parameter defines the root element of this schema if there is more than one candidate.

If the passed root element is an empty string and more candidates are available, a Select Root Element dialog will pop up if MapForce is visible. If MapForce is invisible, no dialog will pop up and only an error is returned.

No XML input sample is assigned to this component.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1244	Failed creating component.

17.2.14.7 InsertXMLSchemaInputParameter

Method: InsertXMLSchemaInputParameter(strParamName as String, strSchemaFileName as String, strXMLRootElementName as String) as Component

Description
Inserts an XML schema input parameter component into the current mapping.

The current mapping has to be a **user-defined** function. Trying to insert it (the schema input parameter) into the **main mapping** will fail.

strParamName is the name of the input parameter component to create and **strSchemaFileName** and **strXMLRootElementName** indicate the respective schema file and the root element of the schema file to be used.

If the passed root element is an empty string and more candidates are available, a Select Root Element dialog will pop up if MapForce is visible. If MapForce is invisible, no dialog will pop up and only an error is returned.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1243	Failed creating parameter component.
1245	This operation is not supported for the main mapping.

17.2.14.8 InsertXMLSchemaOutputParameter

Method: InsertXMLSchemaOutputParameter(strParamName as String, strSchemaFileName as String, strXMLRootElementName as String) as Component

Description
Inserts an XML schema output parameter component into the current mapping.

The current mapping has to be a **user-defined** function. Trying to insert it (the schema output paramter) into the **main** mapping will fail.

strParamName is the name of the output parameter component to create and **strSchemaFileName** and **strXMLRootElementName** indicate the schema file and the root element of the schema file to be used respectively.

If the passed root element is an empty string and more candidates are available, a Select Root Element dialog will pop up if MapForce is visible. If MapForce is invisible, no dialog will pop up and only an error is returned.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1240	Changing the document not allowed. It is read-only.
1243	Failed creating parameter component.
1245	This operation is not supported for the main mapping.

17.2.14.9 IsMainMapping

Property: `IsMainMapping` as `Boolean` (read-only)

Description
Indicates if the current mapping is the main mapping of the document the mapping is in.

True means it is the main mapping.
False means it is a user defined function (UDF).

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

17.2.14.10 Name

Property: `Name` as `String` (read-only)

Description
The name of the mapping / user defined function (UDF).

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.14.11 Parent

Property: `Parent` as `Document` (read-only)

Description
The parent object according to the object model.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.15 Mappings

Represents a collection of Mapping objects.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Iterating through the collection:
`Count`
`Item`

17.2.15.1 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.15.2 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of mappings in the collection.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.15.3 Item

Property: `Item` (*`nIndex`* as `Integer`) as `Mapping` (read-only)

Description
Retrieves the mapping at `nIndex` from the collection. Indices start with 1.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.15.4 Parent

Property: `Parent` as `Document` (read-only)

Description
The parent object according to the object model.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

17.2.16 Options

This object gives access to all MapForce options available in the **Tools | Options** dialog.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

General options:
`ShowLogoOnPrint`
`ShowLogoOnStartup`
`UseGradientBackground`

Options for code generation:
`DefaultOutputEncoding`
`DefaultOutputByteOrder`
`DefaultOutputByteOrderMark`
`XSLTDefaultOutputDirectory`

CodeDefaultOutputDirectory
CPPSettings_DOMType
CPPSettings_GenerateVC6ProjectFile
CppSettings_GenerateVSProjectFile
CPPSettings_LibraryType
CPPSettings_UseMFC
CSharpSettings_ProjectType

17.2.16.1 Application

Property: Application as Application (read-only)

Description
Retrieves the application's top-level object.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

17.2.16.2 CodeDefaultOutputDirectory

Property: CodeDefaultOutputDirectory as String

Description
Specifies the target directory where files generated by Document.GenerateCppCode, Document.GenerateJavaCode and Document.GenerateCHashCode, are placed.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

17.2.16.3 CPPSettings_DOMType

Property: CPPSettings_DOMType as ENUMDOMType

Description
Specifies the DOM type used by Document.GenerateCppCode.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.
1402 The parameter value is out of range
1403 The parameter value is not available anymore

See also
Code Generation

17.2.16.4 CPPSettings_GenerateVC6ProjectFile

Property: CPPSettings_GenerateVC6ProjectFile as Boolean

Description
Specifies if VisualC++ 6.0 project files should be generated by Document.GenerateCppCode.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

17.2.16.5 CppSettings_GenerateVSProjectFile

Property: CppSettings_GenerateVSProjectFile as ENUMProjectType

Description
Specifies which version of VisualStudio (2005 / 2008 / 2010) project files should be generated by Document.GenerateCppCode.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.
1402 The parameter value is out of range
1403 The paramater value is not available anymore

See also
Code Generation

17.2.16.6 CPPSettings_LibraryType

Property: CPPSettings_LibraryType as ENUMLibType

Description
Specifies the library type used by Document.GenerateCppCode.

Errors
1400 The application object is no longer valid.

1401 Invalid address for the return parameter was specified.

See also
`Code Generation`

17.2.16.7 CPPSettings_UseMFC

Property: `CPPSettings_UseMFC` as `Boolean`

Description
Specifies if MFC support should be used by C++ code generated by `Document.GenerateCppCode`.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
`Code Generation`

17.2.16.8 CSharpSettings_ProjectType

Property: `CSharpSettings_ProjectType` as `ENUMProjectType`

Description
Specifies the type of C# project used by `Document.GenerateCHashCode`.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.
1402 The parameter value is out of range
1403 The paramater value is not available anymore

See also
`Code Generation`

17.2.16.9 DefaultOutputByteOrder

Property: `DefaultOutputByteOrder` as `String`

Description
Byte order for the file encoding used for output files.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

17.2.16.10 DefaultOutputByteOrderMark

Property: DefaultOutputByteOrderMark as Boolean

Description
Indicates if a byte order mark (BOM), is to be included in the file encoding of output files.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

17.2.16.11 DefaultOutputEncoding

Property: DefaultOutputEncoding as String

Description
File encoding used for output files.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

17.2.16.12 GenerateWrapperClasses

Property: GenerateWrapperClasses as Boolean

Description
Indicates if wrapper classes are also to be generated when generating code.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

17.2.16.13 JavaSettings_ApacheAxisVersion (obsolete)

Property: `JavaSettings_ApacheAxisVersion` as `ENUMApacheAxisVersion`

Description
Specifies the Apache Axis version to use when generating Java code for web service implementations with SOAP 1.1.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
`Code Generation`

17.2.16.14 Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors
1400 The object is no longer valid.
1401 Invalid address for the return parameter was specified.

17.2.16.15 ShowLogoOnPrint

Property: `ShowLogoOnPrint` as `Boolean`

Description
Show or hide the MapForce logo on printed outputs.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

17.2.16.16 ShowLogoOnStartup

Property: `ShowLogoOnStartup` as `Boolean`

Description
Show or hide the MapForce logo on application startup.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

17.2.16.17 UseGradientBackground

Property: `UseGradientBackground` as `Boolean`

Description
Set or retrieve the background color mode for a mapping window.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

17.2.16.18 XSLTDefaultOutputDirectory

Property: `XSLTDefaultOutputDirectory` as `String`

Description
Specifies the target directory where files generated by `Document.GenerateXSLT` are placed.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
`Code Generation`

17.2.17 Project (Enterprise or Professional Edition)

A Project object represents a project and its tree of project items in MapForce.

Events
Events

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

File handling:
`FullName`
`Name`
`Path`
`Saved`

Save
Close

Project tree navigation:
Count
Item
_NewEnum

Project tree manipulation:
AddActiveFile
AddFile
InsertWebService (Enterprise edition only)
CreateFolder

Code-generation:
Output_Folder
Output_Language
Output_TextEncoding
Java_BasePackageName
GenerateCode
GenerateCodeEx
GenerateCodeIn
GenerateCodeInEx

For examples of how to use the properties and methods listed above, see Example: Project Support. Note: in order to use these properties and methods, you will need to have the Enterprise or Professional edition of MapForce installed on your computer.

For operations with Web services, the Enterprise edition is required.

17.2.17.1 Events

This object supports the following events:

OnProjectClosed

OnProjectClosed

Event: OnProjectClosed (*i_objProject* as Project)

Description
This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is Application.OnProjectOpened.

17.2.17.2 _NewEnum

Property: _NewEnum () as IUnknown (read-only)

Description
This property supports language-specific standard enumeration.

Errors
1500 The object is no longer valid.

Examples

```
// ------------------------------------------------------------
// JScript sample - enumeration of a project's project items.
function AllChildrenOfProjectRoot()
{
    objProject = objMapForce.ActiveProject;
    if ( objProject != null )
    {
        for ( objProjectIter = new Enumerator(objProject); !
objProjectIter.atEnd(); objProjectIter.moveNext() )
        {
            objProjectItem = objProjectIter.item();

            // do something with project item here
        }
    }
}

// ------------------------------------------------------------
// JScript sample - iterate all project items, depth first.
function IterateProjectItemsRec(objProjectItemIter)
{
    while ( ! objProjectItemIter.atEnd() )
    {
        objProjectItem = objProjectItemIter.item();
        // do something with project item here

        IterateProjectItemsRec( new Enumerator(objProjectItem) );

        objProjectItemIter.moveNext();
    }
}
function IterateAllProjectItems()
{
    objProject = objMapForce.ActiveProject;
    if ( objProject != null )
    {
        IterateProjectItemsRec( new Enumerator(objProject) );
    }
}
```

17.2.17.3 AddActiveFile

Method: `AddActiveFile` () as `ProjectItem`

Description
Adds the currently open document to the mapping folder of the project's root.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.
1503	No active document is available.
1504	Active documents needs to be given a path name before it can be added to the project.
1705	Mapping could not be assigned to project. Maybe it is already contained in the target folder.

17.2.17.4 AddFile

Method: `AddFile` (*`i_strFileName`* as `String`) as `ProjectItem`

Description
Adds the specified document to the mapping folder of the project's root.

Errors

1500	The object is no longer valid.
1501	The file name is empty. Invalid address for the return parameter was specified.
1705	Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.

17.2.17.5 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the top-level application object.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

17.2.17.6 Close

Method: `Close` ()

Description
Closes the project without saving.

Errors
1500 The object is no longer valid.

17.2.17.7 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves number of children of the project's root item.

Errors
1500 The object is no longer valid.

Examples
See `Item` or `_NewEnum`.

17.2.17.8 CreateFolder

Method: `CreateFolder` (*`i_strFolderName`* as `String`) as `ProjectItem`

Description
Creates a new folder as a child of the project's root item.

Errors
1500 The object is no longer valid.
1501 Invalid folder name or invalid address for the return parameter was specified.

17.2.17.9 FullName

Property: `FullName` as `String` (read-only)

Description
Path and name of the project file.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

17.2.17.10 GenerateCode

Method: `GenerateCode` ()

Description
Generates code for all project items of the project. The code language and output location is determined by properties of the project and project items.

Errors
1500 The object is no longer valid.
1706 Error during code generation

17.2.17.11 GenerateCodeEx

Method: `GenerateCode` () `as` `ErrorMarkers`

Description
Generates code for all project items of the project. The code language and output location are determined by properties of the project and project items. An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the *Messages* window of MapForce.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.
1706 Error during code generation

17.2.17.12 GenerateCodeIn

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`)

Description
Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items.

Errors
1500 The object is no longer valid.
1706 Error during code generation

17.2.17.13 GenerateCodeInEx

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`) `as` `ErrorMarkers`

Description
Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items. An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the *Messages* window of MapForce.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.
1706 Error during code generation

17.2.17.14 InsertWebService

Method: `InsertWebService(i_strWSDLFile as String, i_strService as String, i_strPort as String, i_bGenerateMappings as Boolean) as ProjectItem`

Description
Inserts a new Web service project into the project's Web service folder. If `i_bGenerateMappings` is true, initial mapping documents for all ports get generated automatically.

Errors
1500 The object is no longer valid.
1501 WSDL file can not be found or is invalid.
Service or port names are invalid.
Invalid address for the return parameter was specified.
1503 Operation not supported by current edition.

17.2.17.15 Item

Property: `Item(i_nItemIndex as Integer) as ProjectItem` (read-only)

Description
Returns the child at `i_nItemIndex` position of the project's root. The index is zero-based. The largest valid index is `Count-1`. For an alternative to visit all children see `_NewEnum`.

Errors
1500 The object is no longer valid.

Examples

```
// ------------------------------------------------------------
// JScript code snippet - enumerate children using Count and Item.
for( nItemIndex = 0; nItemIndex < objProject.Count; nItemIndex++ )
{
    objProjectItem = objProject.Item(nItemIndex);
    // do something with project item here
```

```
}
```

17.2.17.16 Java_BasePackageName

Property: `Java_BasePackageName` as `String`

Description
Sets or gets the base package name of the Java packages that will be generated. This property is used only when generating Java code.

Errors

1500	The object is no longer valid.
1501	Invalid package name specified.
	Invalid address for the return parameter was specified.

17.2.17.17 Name

Property: `Name` as `String` (read-only)

Description
Name of the project file without file path.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

17.2.17.18 Output_Folder

Property: `Output_Folder` as `String`

Description
Sets or gets the default output folder used with `GenerateCode` and `GenerateCodeIn`. Project items can overwrite this value in their `CodeGenSettings_OutputFolder` property, when `CodeGenSettings_UseDefault` is set to false.

Errors

1500	The object is no longer valid.
1501	Invalid folder name specified.
	Invalid address for the return parameter was specified.

17.2.17.19 Output_Language

Property: `Output_Language` as `ENUMProgrammingLanguage`

Description
Sets or gets the default language for code generation when using GenerateCode. Project items can overwrite this value in their CodeGenSettings_OutputLanguage property, when CodeGenSettings_UseDefault is set to false.

Errors
1500 The object is no longer valid.
1501 Invalid language specified.
Invalid address for the return parameter was specified.

17.2.17.20 Output_TextEncoding

Property: Output_TextEncoding as String

Description
Sets or gets the text encoding used when generating XML-based code.

Errors
1500 The object is no longer valid.
1501 Invalid text encoding specified.
Invalid address for the return parameter was specified.

17.2.17.21 Parent

Property: Parent as Application (read-only)

Description
The parent object according to the object model.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

17.2.17.22 Path

Property: Path as String (read-only)

Description
Path of the project file without name.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

17.2.17.23 Save

Method: Save ()

Description
Saves the project to the file defined by FullName.

Errors
1500 The object is no longer valid.
1502 Can't save to file.

17.2.17.24 Saved

Property: Saved as Boolean (read-only)

Description
True if the project was not modified since the last Save operation, false otherwise.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

17.2.18 ProjectItem (Enterprise or Professional Edition)

A ProjectItem object represents one item in a project tree.

Properties and Methods
Properties to navigate the object model:
Application
Parent

Project tree navigation:
Count
Item
_NewEnum

Project item properties:
Kind
Name
WSDLFile (only available to Web service project items)
QualifiedName (only available to Web service project items)

Project tree manipulation:
AddActiveFile (only available to folder items)
AddFile (only available to folder items)
CreateFolder (only available to folder items)

CreateMappingForProject (only available to Web service operations)
Remove

Document access:
Open (only available to mapping items and Web service operations)

Code-generation:
CodeGenSettings_UseDefault
CodeGenSettings_OutputFolder
CodeGenSettings_Language
GenerateCode
GenerateCodeEx
GenerateCodeIn
GenerateCodeInEx

For examples of how to use the properties and methods listed above, see Example: Project Support. Note that, in order to use these properties and methods, you will need to have the Enterprise or Professional edition of MapForce installed on your computer. For operations with Web services, the Enterprise edition is required.

17.2.18.1 _NewEnum

Property: _NewEnum () as IUnknown (read-only)

Description
This property supports language specific standard enumeration.

Errors
1700 The object is no longer valid.

Examples
See Project.Item or Project._NewEnum.

17.2.18.2 AddActiveFile

Method: AddActiveFile () as ProjectItem

Description
Adds the currently active document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.

Errors
1700 The object is no longer valid.
1701 The file name is empty.
Invalid address for the return parameter was specified.
1703 No active document is available.
1704 Active documents needs to be given a path name before it can be added to the project.
1705 Mapping could not be assigned to project.
The file does not exist or is not a MapForce mapping.

Maybe the file is already assigned to the target folder.

17.2.18.3 AddFile

Method: `AddFile` (*`i_strFileName`* as `String`) as `ProjectItem`

Description
Adds the specified document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.

Errors

1700	The object is no longer valid.
1701	The file name is empty.
	Invalid address for the return parameter was specified.
1705	Mapping could not be assigned to project.
	The file does not exist or is not a MapForce mapping.
	Maybe the file is already assigned to the target folder.

17.2.18.4 Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the top-level application object.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

17.2.18.5 CodeGenSettings_Language

Property: `CodeGenSettings_Language` as `ENUMProgrammingLanguage`

Description
Gets or sets the language to be used with `GenerateCode` or `Project.GenerateCode`. This property is consulted only if `CodeGenSettings_UseDefault` is set to false.

Errors

1700	The object is no longer valid.
1701	Invalid language or invalid address for the return parameter was specified.

17.2.18.6 CodeGenSettings_OutputFolder

Property: `CodeGenSettings_OutputFolder` as `String`

Description
Gets or sets the output directory to be used with `GenerateCode`, `GenerateCodeIn`, `Project.GenerateCode` or `Project.GenerateCodeIn`. This property is consulted only if `CodeGenSettings_UseDefault` is set to false.

Errors

1700	The object is no longer valid.
1701	An invalid output folder or an invalid address for the return parameter was specified.

17.2.18.7 CodeGenSettings_UseDefault

Property: `CodeGenSettings_UseDefault` as `Boolean`

Description
Gets or sets whether output directory and code language are used as defined by either (a) the parent folders, or (b) the project root. This property is used with calls to `GenerateCode`, `GenerateCodeIn`, `Project.GenerateCode` and Project.GenerateCodeIn. If this property is set to false, the values of `CodeGenSettings_OutputFolder` and `CodeGenSettings_Language` are used to generate code for this project item..

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

17.2.18.8 Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves number of children of this project item. Also see `Item`.

Errors

1700	The object is no longer valid.

Examples
See `Project.Item` or `Project._NewEnum`.

17.2.18.9 CreateFolder

Method: `CreateFolder` (*`i_strFolderName`* as `String`) as `ProjectItem`

Description
Creates a new folder as a child of this project item.

Errors

1700	The object is no longer valid.
1701	Invalid folder name or invalid address for the return parameter was specified.
1702	The project item does not support children.

17.2.18.10 CreateMappingForProject

Method: `CreateMappingForProject` (*`i_strFileName`* as `String`) as `ProjectItem`

Description
Creates an initial mapping document for a Web service operation and saves it to `i_strFileName`. When using `Project.InsertWebService` you can use the `i_bGenerateMappings` flag to let MapForce automatically generate initial mappings for all ports.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1707	Cannot create new mapping. The project item does not support auto-creation of initial mappings or a mapping already exists.
1708	Operation not supported in current edition.

17.2.18.11 GenerateCode

Method: `GenerateCode` ()

Description
Generates code for this project item and its children. The code language and output location is determined by `CodeGenSettings_UseDefault`, `CodeGenSettings_Language` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

Errors

1700	The object is no longer valid.
1706	Error during code generation.

17.2.18.12 GenerateCodeEx

Method: `GenerateCode` () `as` `ErrorMarkers`

Description
Generates code for this project item and its children. The code language and output location are determined by `CodeGenSettings_UseDefault`, `CodeGenSettings_Language` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the *Messages* window of MapForce.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1706	Error during code generation.

17.2.18.13 GenerateCodeIn

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`)

Description
Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

Errors

1700	The object is no longer valid.
1701	Invalid language specified.
1706	Error during code generation.

17.2.18.14 GenerateCodeInEx

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`) `as` `ErrorMarkers`

Description
Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the *Messages* window of MapForce.

Errors

1700 The object is no longer valid.
1701 Invalid language specified or invalid address for the return parameter was specified.
1706 Error during code generation.

17.2.18.15 Item

Property: `Item(i_nItemIndex as Integer) as ProjectItem` (read-only)

Description

Returns the child at `i_nItemIndex` position of this project item. The index is zero-based. The largest valid index is `Count` - 1.
For an alternative to visit all children see `_NewEnum`.

Errors

1700 The object is no longer valid.

Examples

See `Project.Item` or `Project._NewEnum`.

17.2.18.16 Kind

Property: `Kind` as `ENUMProjectItemType` (read-only)

Description

Retrieves the kind of the project item. Availability of some properties and the applicability of certain methods is restricted to specific kinds of project items. The description of all methods and properties contains information about these restrictions.

Errors

1700 The object is no longer valid.
1701 Invalid address for the return parameter was specified.

17.2.18.17 Name

Property: `Name` as `String`

Description

Retrieves or sets the name of a project item. The name of most items is read-only. Exceptions are user-created folders, the names of which can be altered after creation.

Errors

1700 The object is no longer valid.
1701 Invalid address for the return parameter was specified.
1702 Project item does not allow to alter its name.

17.2.18.18 Open

Method: `Open` () as `Document`

Description
Opens the project item as a document or makes the corresponding document the active one, if it is already open. The project item must be a MapForce mapping or, for Enterprise edition only, Web service operation.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	The project item does not refer to a MapForce mapping file.
1708	Operation not supported in current edition.

17.2.18.19 Parent

Property: `Parent` as `Project` (read-only)

Description
Retrieves the project that this item is a child of. Has the same effect as `Application.ActiveProject`.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

17.2.18.20 QualifiedName

Property: `QualifiedName` as `String` (read-only)

Description
Retrieves the qualified name of a Web service item.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	The project item is not a part of a Web service.

17.2.18.21 Remove

Method: `Remove` ()

Description

Remove this project item and all its children from the project tree.

Errors

1700 The object is no longer valid.

17.2.18.22 WSDLFile

Property: `WSDLFile` as `String` (read-only)

Description

Retrieves the file name of the WSDL file defining the Web service that hosts the current project item.

Errors

1700 The object is no longer valid.
1701 Invalid address for the return parameter was specified.
1702 The project item is not a part of a Web service.

17.3 Enumerations

This is a list of all enumerations used by the MapForce API. If your scripting environment does not support enumerations, use the number-values instead.

17.3.1 ENUMApacheAxisVersion (obsolete)

Description
Enumeration values to select the Apache Axis version.

Possible values:

eApacheAxisVersion_Axis	= 1
eApacheAxisVersion_Axis2	= 2

See also
`Code Generation`

17.3.2 ENUMApplicationStatus

Description
Enumeration values to indicate the status of the application.

Possible values:

eApplicationRunning	= 0
eApplicationAfterLicenseCheck	= 1
eApplicationBeforeLicenseCheck	= 2
eApplicationConcurrentLicenseCheckFailed	= 3

17.3.3 ENUMAppOutputLine_Severity

Description
Enumeration values to identify the severity of an AppOutputLine.

Possible values:

eSeverity_Undefined	= -1
eSeverity_Info	= 0
eSeverity_Warning	= 1
eSeverity_Error	= 2
eSeverity_CriticalError	= 3
eSeverity_Success	= 4
eSeverity_Summary	= 5
eSeverity_Progress	= 6
eSeverity_DataEdit	= 7
eSeverity_ParserInfo	= 8
eSeverity_PossibleInconsistencyWarning	= 9

eSeverity_Message	= 10
eSeverity_Document	= 11
eSeverity_Rest	= 12
eSeverity_NoSelect	= 13
eSeverity_Select	= 14
eSeverity_Autoinsertion	= 15
eSeverity_GlobalResources_DefaultWarning	= 16

17.3.4 ENUMAppOutputLine_TextDecoration

Description
Enumeration values for the different kinds of text decoration of an AppOutputLine.

Possible values:

eTextDecorationDefault	= 0
eTextDecorationBold	= 1
eTextDecorationDebugValues	= 2
eTextDecorationDB_ObjectName	= 3
eTextDecorationDB_ObjectLink	= 4
eTextDecorationDB_ObjectKind	= 5
eTextDecorationDB_TimeoutValue	= 6
eTextDecorationFind_MatchingString	= 7
eTextDecorationValidation_Speclink	= 8
eTextDecorationValidation_ErrorPosition	= 9
eTextDecorationValidation_UnkownParam	= 10

17.3.5 ENUMCodeGenErrorLevel

Description
Enumeration values to identify severity of code generation messages.

Possible values:

eCodeGenErrorLevel_Information	= 0
eCodeGenErrorLevel_Warning	= 1
eCodeGenErrorLevel_Error	= 2
eCodeGenErrorLevel_Undefined	= 3

17.3.6 ENUMComponentDatapointSide

Description
Enumeration values to indicate the side of a datapoint on its component.

Possible values:

eDatapointSideInput	= 0
eDatapointSideOutput	= 1

See also

GetRootDatapoint

17.3.7 ENUMComponentSubType

Description
Enumeration values to indicate component sub types.

Possible values:

eComponentSubType_None	= 0
eComponentSubType_Text_EDI	= 1
eComponentSubType_Text_Flex	= 2
eComponentSubType_Text_CSVFLF	= 3

17.3.8 ENUMComponentType

Description
Enumeration values to indicate component types.

Possible values:

eComponentType_Unknown	= 0
eComponentType_XML	= 1
eComponentType_DB	= 2
eComponentType_Text	= 3
eComponentType_Excel	= 4
eComponentType_WSDL	= 5
eComponentType_XBRL	= 6

17.3.9 ENUMComponentUsageKind

Description
Enumeration values to indicate component usage kind.

Possible values:

eComponentUsageKind_Unknown	= 0
eComponentUsageKind_Instance	= 1
eComponentUsageKind_Input	= 2
eComponentUsageKind_Output	= 3
eComponentUsageKind_Variable	= 4

17.3.10 ENUMConnectionType

Description
Enumeration values to indicate the type of a connection.

Possible values:

eConnectionTypeTargetDriven	= 0
eConnectionTypeSourceDriven	= 1
eConnectionTypeCopyAll	= 2

See also
ConnectionType

17.3.11 ENUMDOMType

Description
Enumeration values to specify the DOM type used by generated C++ mapping code.

Possible values:

eDOMType_xerces	= 1
eDOMType_xerces3	= 2
eDOMType_msxml6	= 3

Obsolete values

eDOMType_msxml4	= 0

Obsolete in this context means that this value is not supported and should not be used.

`eDOMType_xerces` indicates Xerces 2.x usage; `eDOMType_xerces3` indicates Xerces 3.x usage.

See also
`Code Generation`

17.3.12 ENUMLibType

Description
Enumeration values to specify the library type used by the generated C++ mapping code.

Possible values:

eLibType_static	= 0
eLibType_dll	= 1

See also
`Code Generation`

17.3.13 ENUMProgrammingLanguage

Description
Enumeration values to select a programming language.

Possible values:

eUndefinedLanguage	= -1
eJava	= 0
eCpp	= 1
eCSharp	= 2
eXSLT	= 3
eXSLT2	= 4
eXQuery	= 5

See also
`Code Generation`

17.3.14 ENUMProjectItemType

WDescription
Enumeration to identify the different kinds of project items that can be children of Project or folder-like ProjectItems.

Possible values:

eProjectItemType_Invalid	= -1
eProjectItemType_MappingFolder	= 0
eProjectItemType_Mapping	= 1
eProjectItemType_WebServiceFolder	= 2
eProjectItemType_WebServiceRoot	= 3
eProjectItemType_WebServiceService	= 4
eProjectItemType_WebServicePort	= 5
eProjectItemType_WebServiceOperation	= 6
eProjectItemType_ExternalFolder	= 7
eProjectItemType_LibrarzFolder	= 8
eProjectItemType_ResourceFolder	= 9
eProjectItemType_VirtualFolder	= 10

See also
`ProjectItem.Kind`

17.3.15 ENUMProjectType

Description
Enumeration values to select a project type for generated C# mapping code.

Possible values:

	eVisualStudio2005Project	= 4	Also for C++ code
	eVisualStudio2008Project	= 5	Also for C++ code
	eVisualStudio2010Project	= 6	Also for C++ code
Obsolete values	eVisualStudioProject	= 0	
	eVisualStudio2003Project	= 1	
	eBorlandProject	= 2	

Obsolete in this context means that this value is not supported and should not be used.

See also
`Code Generation`

17.3.16 ENUMSearchDatapointFlags

Description
Enumeration values used as bit-flags; to be used as combination of flags when searching for a datapoint.

Possible values:

eSearchDatapointElement	= 1
eSearchDatapointAttribute	= 2

See also
GetChild

17.3.17 ENUMViewMode

Description
Enumeration values to select a MapForce view.

Possible values:

eMapForceView	= 0
eXSLView	= 1
eOutputView	= 2

Chapter 18

ActiveX Integration

18 ActiveX Integration

MapForceControl is a control that provides a means of integration of the MapForce user interface and the functionality described in this section into most kinds of applications. ActiveX technology was chosen so as to allow integration using any of a wide variety of languages; this enables C++, C#, Visual Basic, or HTML to be used for integration (ActiveX components integrated in HTML officially only work with Microsoft Internet Explorer). The attached Java wrapper library allows integration into Java. All components are full OLE Controls, which makes integration as simple as possible. Two different levels of integration are provided, thus enabling the integration to be adapted to a wide range of needs.

To integrate MapForce you must install the MapForce Integration Package. Ensure that you install MapForce first, and then the MapForce Integration Package. For the integration pack to work correctly, JRE 6 (or later) must be installed.

For a successful integration you have to consider the following main design factors:

- What technology or programming language can the hosting application use to integrate the MapForceControl?
- Should the integrated UI look exactly like MapForce with all its menus, toolbars, and windows, or will a subset of these elements—like allowing only one document and a restricted set of commands—be more effective?
- How deep will the integration be? Should the MapForce user interface be used as is? Are user interface extensions and/or restrictions required? Can some frequently used tasks be automated?

The sections, Integration at the Application Level and Integration at Document Level describe the key steps at these respective levels. The Programming Languages section provides examples in C#, HTML, Visual Basic, and Java. Looking through these examples will help you to make the right decisions quickly. The section, Object Reference, describes all COM objects that can be used for integration, together with their properties and methods.

For automation tasks, the MapForce Automation Interface is accessible from the MapForceControl as well.

For information about how to integrate MapForce into Microsoft Visual Studio see the section, MapForce in Visual Studio.

MapForce integration and deployment on client computers:

If you create a .NET application and intend to distribute it to other clients, you will have to install the following on the client computer(s):

- The MapForce application
- The MapForce integration package.
- The custom integration code.

18.1 Integration at Application Level

Integration at application level is simple and straightforward. It allows you to embed the complete interface of MapForce into a window of your application. Since you get the whole user interface of MapForce, you get 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`. Its property `IntegrationLevel` defaults to application-level. You may use `Appearance` and `BorderStyle` to configure the appearance of the control's wrapper window. Do not instantiate or access `MapForceControlDocument` or `MapForceControlPlaceHolder` ActiveX controls when integrating at application-level.

If you have any initialization to do or if you want to automate some behaviour of MapForce, use the properties, methods, and events described for `MapForceControl`. Consider using `MapForceControl.Application` for more complex access to MapForce functionality.

In the Programming Languages | HTML section is an example (Integration at Application Level) that shows how the MapForce application can be embedded in an HTML page. For usage with other programming languages, or more sophisticated access, see the Examples of integration at document-level (in C#, HTML, and Java).

18.2 Integration at Document Level

Integration at document level gives you freedom over instantiation and placement of the following parts of the MapForce user interface:

- Editing windows for MapForce mappings
- MapForce overview window
- MapForce library window
- MapForce validation window
- MapForce project window

If necessary, a replacement for the menus and toolbars of MapForce must be provided by your application.

You will need to instantiate and access multiple ActiveX controls, depending on which user interface parts you want to re-use. All these controls are contained in the MapForceControl OCX.

- Use MapForceControl to set the integration level and access application wide functionality.
- Use MapForceControlDocument to create any number of editor windows. It may be sufficient to create only one window and re-use it, depending on your needs.
- Optionally Use MapForceControlPlaceholder to embed MapForce overview, library, validation and project windows.
- Access run-time information about commands, menus, and toolbars available in MapForceControl to seamlessly integrate these commands into your application's menus and toolbars. See Query MapForce Commands for more information.

If you want to automate some behaviour of MapForce use the properties, methods, and events described for the MapForceControl, MapForceControlDocument and MapForceControlPlaceHolder. Consider using MapForceControl.Application, MapForceControlDocument.Document and MapForceControlPlaceHolder.Project for more complex access to MapForce functionality. However, to open a document always use MapForceControlDocument.Open or MapForceControlDocument.New on the appropriate document control. To open a project always use MapForceControlPlaceHolder.OpenProject on a placeholder control embedding a MapForce project window.

See Examples on how to instantiate and access the necessary controls in different programming environments.

MapForce integration and deployment on client computers:
If you create a .NET application and intend to distribute it to other clients, you will have to install the following on the client computer(s):

- The MapForce application
- The MapForce integration package.
- The custom integration code.

18.2.1 Use MapForceControl

To integrate at document level, instantiate a `MapForceControl` first. Set the property `IntegrationLevel` to `ICActiveXIntegrationOnDocumentLevel (= 1)`. Set the

window size of the embedding window to `0x0` to hide any user interface behind the control. You may use `Appearance` and `BorderStyle` to configure the appearance of the control's wrapper window.

Avoid using the method `Open` since this might lead to unexpected results. Use the corresponding open methods of `MapForceControlDocument` and `MapForceControlPlaceHolder`, instead.

See Query MapForce Commands for a description of how to integrate MapForce commands into your application. Send commands to MapForce via the method `Exec`. Query if a command is currently enabled or disabled using the method `QueryStatus`.

18.2.2 Use MapForceControlDocument

An instance of the `MapForceControlDocument` ActiveX control allows you to embed one MapForce document editing window into your application. You can use any number of instances you need.

Use the method `Open` to load any other existing file.

The control does not support a read-only mode. The value of the property `ReadOnly` is ignored.

Use `Path` and `Save` or methods and properties accessible via the property `Document` to access document functionality.

18.2.3 Use MapForceControlPlaceHolder

Instances of MapForceControlPlaceHolder ActiveX controls allow you to selectively embed the additional helper windows of MapForce into your application. The property `PlaceholderWindowID` selects the MapForce helper window to be embedded. Use only one MapForceControlPlaceHolder for each window identifier. See for valid window identifiers.

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.

18.2.4 Query MapForce Commands

When integrating at document-level, no menu or toolbar from MapForce is available to your application. Instead, you can query all the commands and the structure of the application menu at runtime. Professional applications will need to integrate this menu in a sophisticated manner into their own menu structure. Your installation of MapForce even provides you with command label images used within MapForce. See the folder `<Documents>\Altova\MapForce2016\MapForceExamples\\ActiveX\Images` of your MapForce installation for icons in GIF format. The file names correspond to the `labels` of commands.

18.3 Programming Languages

This section contains examples of MapForce document-level integration using different container environments and programming languages. (The HTML section additionally contains examples of integration at application level.) Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX` of your MapForce installation.

18.3.1 C#

The C# example shows how to integrate the MapForceControl in a common desktop application created with C# using Visual Studio. The following topics are covered:

- Building a dynamic menu bar based on information the MapForceControl API provides.
- Usage of MapForce Placeholder controls in a standard frame window.
- Usage of a MapForce Placeholder control in a sizeable Tool Window.
- How to handle an event raised by the MapForceControl API.

The source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX\C#` of your MapForce installation. The example application is complete and, under normal circumstances, you should be able to build and run it without additional configuration.

Note the following:

1. Before building and running the sample C# solution, first copy it to a directory where you have write permissions. Otherwise, you will need to run Visual Studio as administrator.
2. You might be prompted to upgrade the solution to the version of Visual Studio that you are using. In that case, follow the Visual Studio wizard steps to complete the process.
3. To build the sample C# solution, .NET platform 4.0 or later is required.
4. The example C# solution has two build configurations in Visual Studio: x32 and x64. If you want to build using the x64 configuration, ensure that you have installed the 64-bit version of MapForce and MapForce Integration Package.

18.3.1.1 Introduction

Adding the MapForce components to the Toolbox
Before you take a look at the sample project, please add the ActiveX controls to the Visual Studio Toolbox. The MapForce Installer will have already installed the assemblies in the .NET Global Assembly Cache (GAC). If you open the Toolbox dialog under **Tools | Choose Toolbox Items** the controls will appear as `AxMapForceControl`, `AxMapForceControlDocument` and `AxMapForceControlPlaceholder` on the .NET Framework Components tab. Check all to make them available to the IDE.

Now you can open the `MapForceApplication.sln` file to load the project.

18.3.1.2 Placing the MapForceControl

It is necessary to have one MapForceControl instance to set the integration level and to manage the Document and Placeholder controls of the MapForce library. The control is accessible via the General section of the Toolbox helper window in the IDE. To add it you need to select the component in the Toolbox window and drag a rectangle wherever you want to have it in the destination window. If you have an application which does not open a window on startup you can use a simple invisible Form with the control on it which is created manually in the code.

The example project adds this instance to the main MdiContainer MDIMain. If you open MDIMain in the Design View from the Solution Explorer you will see a light blue rectangle at the top-left side in the client area of the Frame window. Selecting this rectangle will show you the properties of the MapForceControl. It is important to set the `IntegrationLevel` property to **`ICActiveXIntegrationOnDocumentLevel`** in order to turn on the Document and Placeholder support of the MapForce library. Properties of the <%MAPCTRL%> component placed in the MDIFrame Window of the example application are shown below:

⊞ (DataBindings)	
⊞ (DynamicProperties)	
(Name)	**axMapForceControl**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
CausesValidation	True
Dock	None
ImeMode	NoControl
IntegrationLevel	**ICActiveXIntegrationOnDocumentLevel**
⊞ Location	**280; 8**
Locked	False
Modifiers	Private
ReadOnly	**True**
⊞ Size	**224; 112**
TabIndex	**1**
TabStop	**False**
Tag	
Visible	**False**

Set the Visible flag to False to avoid any confusion about the control for the user.

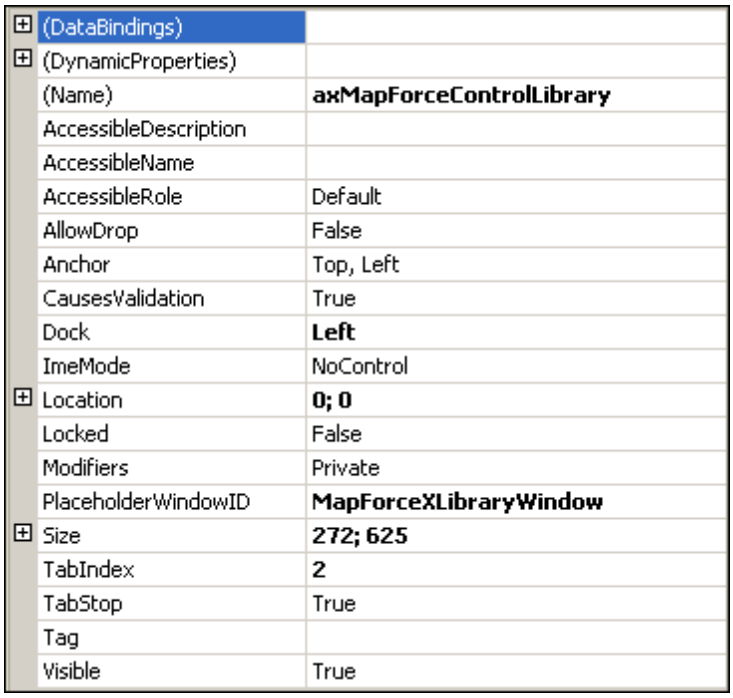
18.3.1.3 Adding the Placeholder Control

Placeholders on the MDI Frame

The example project has to place Placeholder controls on the main MDI Frame. They are also added via the Toolbox window by dragging a rectangle on the destination Form. To set the type of the Placeholder which should be displayed one has to set the `PlaceholderWindowID` property. This property can also be changed during runtime in the code of the application. The

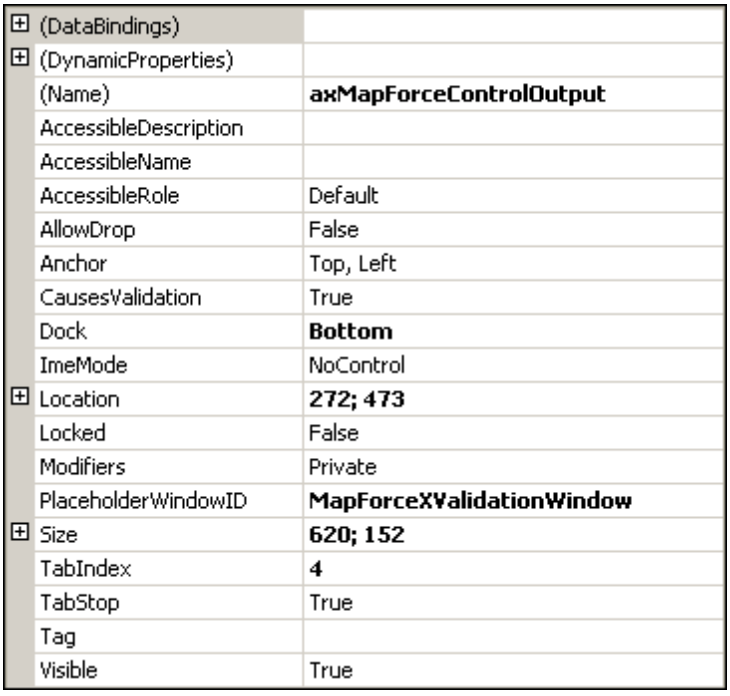
Placeholder control would change its content immediately.

Properties of the Library window on the left side of the MDIMain Frame window are shown below:



(DataBindings)	
(DynamicProperties)	
(Name)	**axMapForceControlLibrary**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
CausesValidation	True
Dock	**Left**
ImeMode	NoControl
Location	**0; 0**
Locked	False
Modifiers	Private
PlaceholderWindowID	**MapForceXLibraryWindow**
Size	**272; 625**
TabIndex	**2**
TabStop	True
Tag	
Visible	True

Properties of the Output window at the bottom:



(DataBindings)	
(DynamicProperties)	
(Name)	**axMapForceControlOutput**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
CausesValidation	True
Dock	**Bottom**
ImeMode	NoControl
Location	**272; 473**
Locked	False
Modifiers	Private
PlaceholderWindowID	**MapForceXValidationWindow**
Size	**620; 152**
TabIndex	**4**
TabStop	True
Tag	
Visible	True

The Placeholders also have the Anchor and Dock properties set in order to react on resizing of the Frame window.

Placeholder on a separate Toolwindow
It is also possible to place a Placeholder control on a separate floating Toolwindow. To do this, create a new Form as a Toolwindow and add the control as shown above. The MapForceOverviewWnd in the sample project contains the Overview window of MapForce.

Properties of the Overview Toolwindow:

⊞ (DataBindings)	
⊞ (DynamicProperties)	
(Name)	**axMapForceControlOverview**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	**Top, Bottom, Left, Right**
CausesValidation	True
Dock	None
ImeMode	NoControl
⊞ Location	**0; 0**
Locked	False
Modifiers	**Public**
PlaceholderWindowID	**MapForceXOverviewWindow**
⊞ Size	**292; 266**
TabIndex	**0**
TabStop	True
Tag	
Visible	True

However, all Placeholder controls need a connection to the main MapForceControl. Normally this connection can be established automatically and there is nothing more to do. The two placeholders on the MDI Frame work like this. In the case of the Placeholder control in the Toolwindow, we need to add some code to the `public MDIMain()` method in `MDIMain.cs`:

```
m_MapForceOverview = new MapForceOverviewWnd();

MapForceControlLib.MapForceControlPlaceHolderClass type =
(MapForceControlLib.MapForceControlPlaceHolderClass)
m_MapForceOverview.axMapForceControlOverview.GetOcx();
type.AssignMultiDocCtrl((MapForceControlLib.MapForceControlClass)
axMapForceControl.GetOcx());

m_MapForceOverview.Show();
```

The MapForceOverviewWnd is created and shown here. In addition, a special method of the Placeholder control is called in order to connect the MapForcecontrol to it. `AssignMultiDocCtrl()` takes the MapForceControl as parameter and registers a reference to it in the Placeholder control.

18.3.1.4 Retrieving Command Information

The MapForceControl gives access to all commands of MapForce through its `CommandsStructure` property. The example project uses the `MapForce Commands` and `MapForce Command` interfaces to dynamically build a menu in the MDI Frame window.

The code to add the commands will be placed in the `MDIMain` method of the `MapForceApplication` class in the file `MDIMain.cs`:

```
public MDIMain()
{
    .
    .
    .
    MapForceControlLib.MapForceCommands objCommands;
    objCommands = axMapForceControl.CommandsStructure;

    long nCount = objCommands.Count;

    for(long idx = 0;idx < nCount;idx++)
    {
        MapForceControlLib.MapForceCommand     objCommand;
        objCommand = objCommands[(int)idx];

        // We are looking for the Menu with the name IDR_MAPFORCE. This menu
        contains
        // the complete main menu of MapForce.

        if(objCommand.Label == "IDR_MAPFORCE")
        {
            InsertMenuStructure(mainMenu.MenuItems, 1, objCommand, 0, 0,
            false);
        }
    }
    .
    .
    .
}
```

`mainMenu` is the name of the menu object of the MDI Frame window created in the Visual Studio IDE. `InsertMenuStructure` takes the MapForce menu from the IDR_MAPFORCE command object and adds the MapForce menu structure to the already existing menu of the sample project. No commands from the **File**, **Project**, or **Window** menu are added.

The new commands are instances of the class `CustomMenuItem`, which is defined in `CustomMenuItem.cs`. This class has an additional member to save the MapForce command ID, which is taken to execute the command using `Exec` on selecting the menu item. This code from `InsertMenuStructure` creates the new command:

```
CustomMenuItem   newMenuItem = new CustomMenuItem();

if(objCommand.IsSeparator)
    newMenuItem.Text = "-";
```

```
else
{
    newMenuItem.Text = strLabel;
    newMenuItem.m_MapForceCmdID = (int)objCommand.ID;
    newMenuItem.Click += new EventHandler(AltovaMenuItem_Click);
}
```

You can see that all 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"))
    {
        CustomMenuItemcustomItem = (CustomMenuItem)sender;

        ProcessCommand(customItem.m_MapForceCmdID);
    }
}

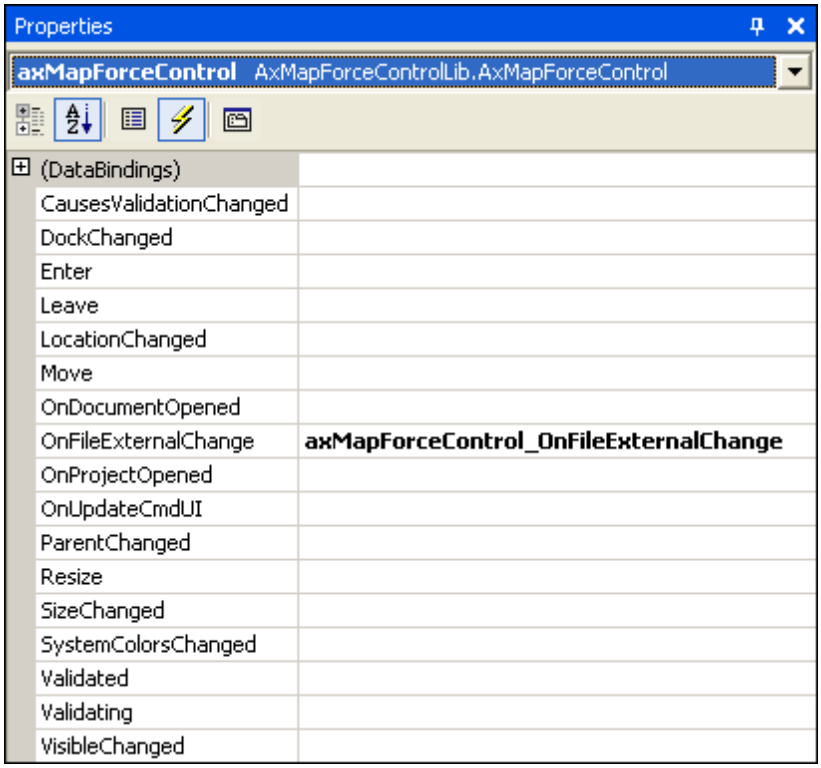
private void ProcessCommand(int nID)
{
    MapForceDoc  docMapForce = GetCurrentMapForceDoc();

    if(docMapForce != null)
        docMapForce.axMapForceControlDoc.Exec(nID);
    else
        axMapForceControl.Exec(nID);
}
```

`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.

18.3.1.5 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 picture below shows the events of the main MapForceControl:



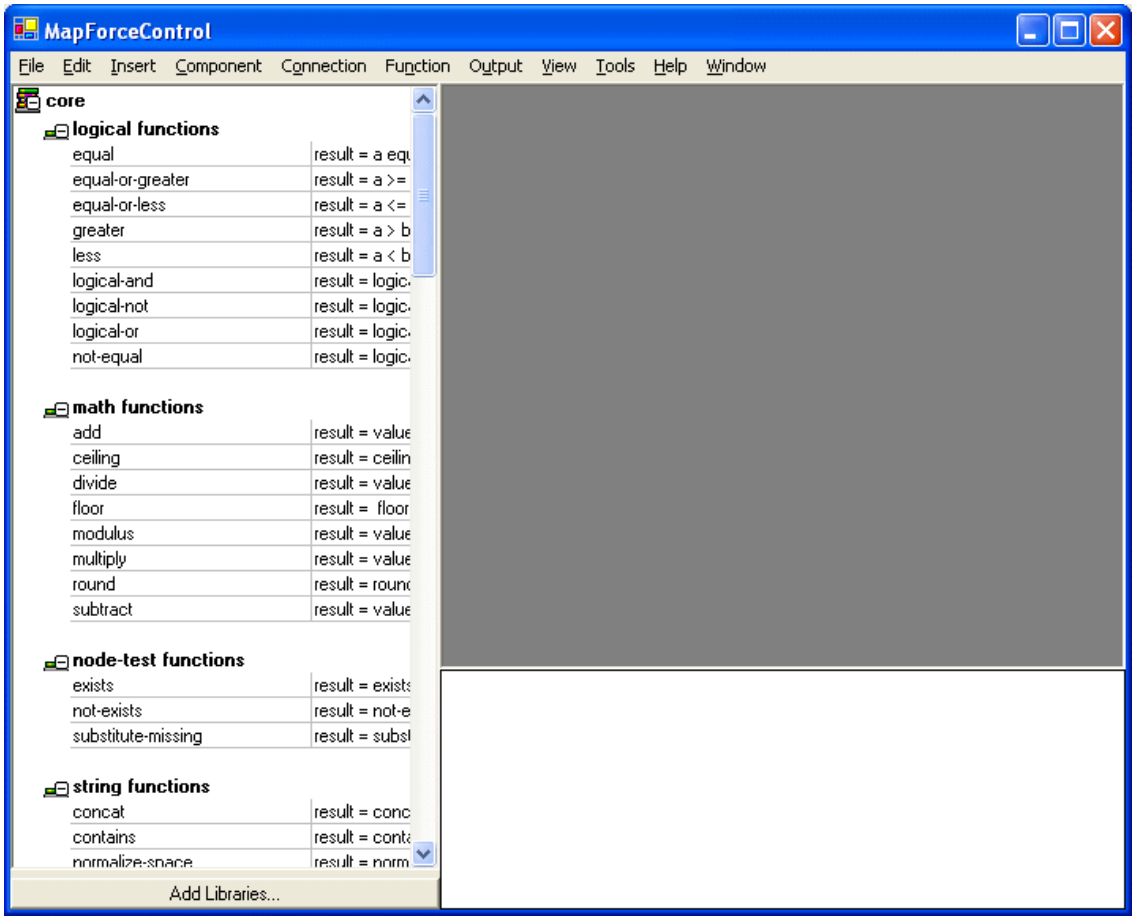
As you can see, the example project only overrides the `OnFileExternalChange` event. The creation of the C# delegate is done for you by the C# Framework. All you need to do is to fill the empty event handler. The handler implementation turns off any file reloading and displays a message box to inform the user that a file loaded by the MapForceControl has been changed from outside:

```
private void axMapForceControl_OnFileExternalChange(object sender,
AxMapForceControlLib._DMapForceControlEvents_OnFileExternalChangeEvent e)
{
    MessageBox.Show("Attention: The file " + e.strPath + " has been changed
from outside\nbut reloading is turned off in the sample application!");

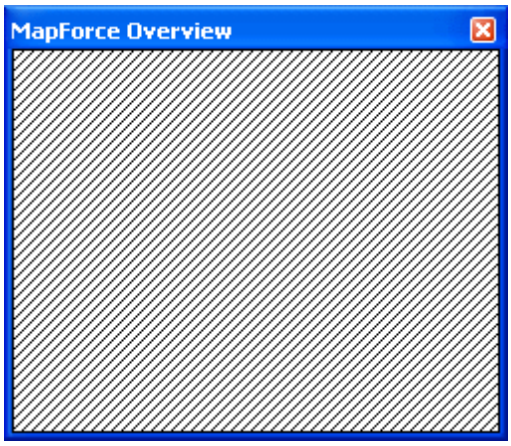
    // This turns off any file reloading:
    e.varRet = false;
}
```

18.3.1.6 Testing the Example

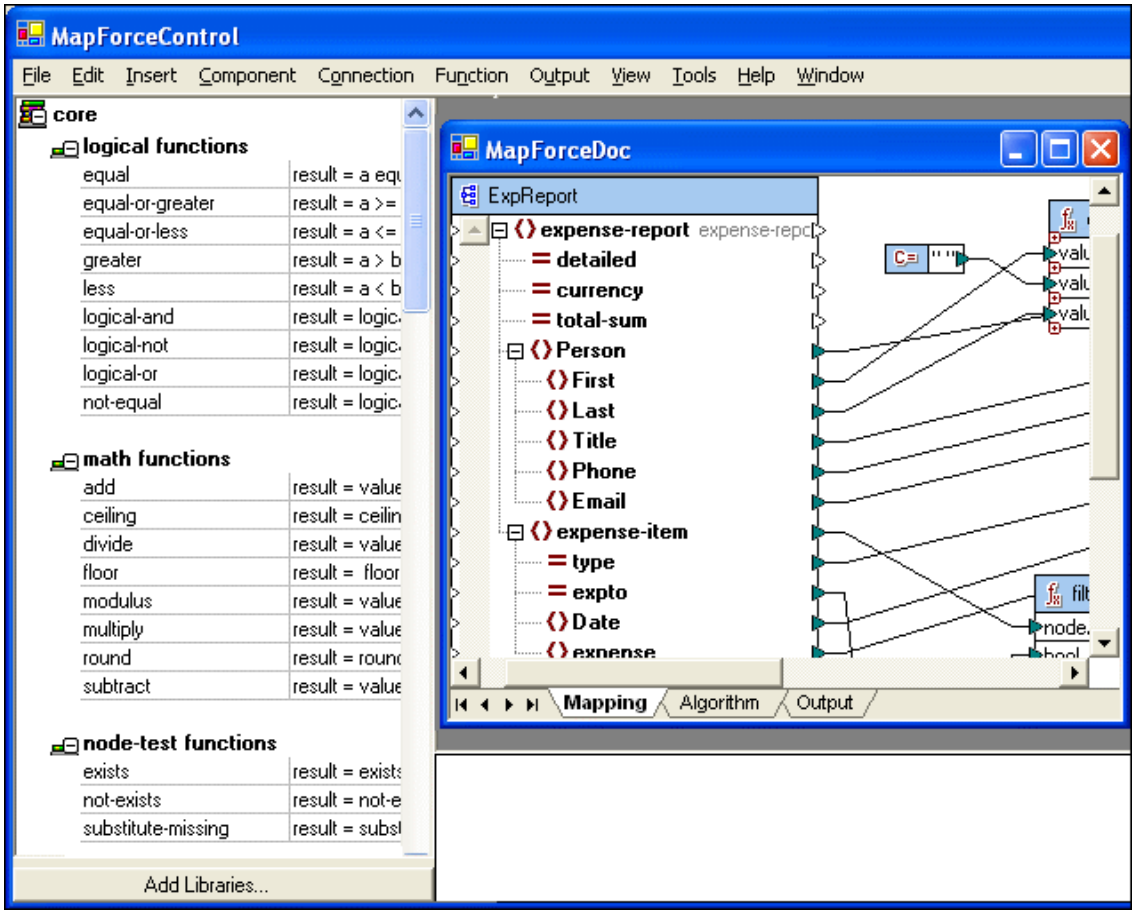
After adding the assemblies to the Toolbox (see Introduction), you can run the sample project with F5 without the need to change anything in the code. The main MDI Frame window is created together with a floating Toolwindow containing the Overview window of MapForce. The application looks something like the screenshot below:



The floating Overview Toolwindow is also created:



Use **File | Open** to open the file `MarketingExpenses.mfd`, which is in the MapForce examples folder. The file is loaded and displayed in an own document child window:



After you load the document, you can try using menu commands. Note that context menus are also available. If you like, you can also load additional documents. Save any modifications using the **File | Save** command.

18.3.2 HTML

The code listings in this section show how to integrate the MapForceControl at application level and document level. Source code for all examples is available in the folder `<ApplicationFolder>\Examples\ActiveX\HTML` of your MapForce installation. The examples work only in Internet Explorer.

Note: When it runs in 64-bit mode, Internet Explorer 10 or later does not load ActiveX controls.

18.3.2.1 Integration at Application Level

This example shows a simple integration of the MapForce control at application-level into a HTML page. The integration is described in the following sections:

- Instantiate a MapForceControl in HTML code.
- Implement buttons to load documents and automate code-generation tasks.
- Define actions for some application events.

The code for this example is available at the following location in your MapForce installation: `<Documents>\Altova\MapForce2016\MapForceExamples\\ActiveX\HTML \MapForceActiveX_ApplicationLevel.htm`.

Instantiate the Control

The HTML `Object` tag is used to create an instance of the MapForceControl. The `Classid` is that of MapForceControl. Width and height specify the window size. No additional parameters are necessary, since application-level is the default.

```
<OBJECT id="objMapForceControl"
        Classid="clsid:A38637E9-5759-4456-A167-F01160CC22C1"
        width="800"
        height="500"
        VIEWASTEXT>
</OBJECT>
```

Add Button to Open Default Document

As a simple example of how to automate some tasks, we add a button to the page:

```
<input type="button" value="Open Marketing Expenses" onclick="BtnOpenMEFile()">
```

When clicked, a predefined document will be opened in the MapForceControl. We use a method to locate the file relative to the MapForceControl so the example can run on different installations.

```
<SCRIPT ID=Javahandlers LANGUAGE=javascript>
// --------------------------------
// open a pre-defined document
function BtnOpenMEFile()
{

             objMapForceControl.Open("C:\Documents and Settings\username\My
Documents\Altova\XMLSpy2016\Examples/MarketingExpenses.mfd");

}
</SCRIPT>
```

Add Buttons for Code Generation

Although code-generation for the active document is available via menus, we want to have buttons that will generate code without asking the user for the location of the output. The method is similar to that used in the previous section.

First come the buttons:

```
<input type="button" value="Generate XSLT" onclick="BtnGenerate( 0 )">
<input type="button" value="Generate Java" onclick="BtnGenerate( 1 )">
<input type="button" value="Generate C++" onclick="BtnGenerate( 2 )">
```

```
<input type="button" value="Generate C#" onclick="BtnGenerate( 3 )">
```

Then we provide the script that will generate the code into sub-folders of the currently defined default output folders.

```
<SCRIPT ID=Javahandlers LANGUAGE=javascript>
// ---------------------------------------------------------------------
// generate code for active document into language-specific sub folders of
// the current default output directory. No user interaction necessary.
function BtnGenerate(languageID)
{
	// get top-level object of automation interface
	var objApp = objMapForceControl.Application;

	// get the active document
	var objDocument = objApp.ActiveDocument;

	// retrieve object to set the generation output path
	var objOptions = objApp.Options;

	if ( objDocument == null )
		alert( "no active document found" );
	else
	{
		if (languageID == 0)
		{
			objOptions.XSLTDefaultOutputDirectory =
objOptions.XSLTDefaultOutputDirectory + "\\XSLTGen";
			objDocument .GenerateXSLT();
		}
		else if (languageID == 1)
		{
			objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/JavaCode";
			objDocument .GenerateJavaCode();
		}
		else if (languageID == 2)
		{
			objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/CPPCode";
			objDocument .GenerateCppCode();
		}
		else if (languageID == 3)
		{
			objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/CSharpCode";
			objDocument .GenerateCHashCode();
		}
	}
}
</SCRIPT>
```

Connect to Custom Events

The example implements two event callbacks for MapForceControl custom events to show the principle:

```
<!-- ------------------------------------------------------------ -->
<!--  custom event 'OnDocumentOpened" of MapForceControl object  -->
<SCRIPT LANGUAGE="javascript">
       function objMapForceControl::OnDocumentOpened( objDocument )
       {
       // alert("Document '" + objDocument.Name + "' opened!");
       }
</SCRIPT>

<!-- ------------------------------------------------------------ -->
<!--  custom event 'OnDocumentClosed" of MapForceControl object  -->
<SCRIPT LANGUAGE="javascript">
       function objMapForceControl::OnDocumentClosed( objDocument )
       {
       // alert("Document '" + objDocument.Name + "' closed!");
       }
</SCRIPT>
```

18.3.2.2 Integration at Document Level

This example shows an integration of the MapForce control at document-level into a HTML page. The following topics are covered:

- Instantiate a MapForceControl ActiveX control object in HTML code
- Instantiate a MapForceControlDocument ActiveX control to allow editing a MapForce file
- Instantiate one MapForceControlPlaceHolder for a MapForceControl project window
- Instantiate one MapForceControlPlaceHolder to alternatively host one of the MapForce helper windows
- Create a simple customer toolbar for some heavy-used MapForce commands
- Add some more buttons that use the COM automation interface of MapForce
- Use event handlers to update command buttons

This example is available in its entirety in the file `MapForceActiveX_ApplicationLevel.htm` within the `C:\Program Files\Altova\MapForce2016\Examples\ActiveX\HTML\` folder of your MapForce installation.

Instantiate the MapForceControl

MapForceControlThe HTML `OBJECT` tag is used to create an instance of the MapForceControl. The Classid is that of MapForceControl. Width and height are set to 0 since we use this control as manager control without use for its user interface. The integration level is specified as a parameter within the `OBJECT` tag.

```
<OBJECT id="objMapForceXMapForceControl"
```

```
        Classid="clsid:A38637E9-5759-4456-A167-F01160CC22C1"
        width="0"
        height="0"
        VIEWASTEXT>
   <PARAM NAME="IntegrationLevel" VALUE="1">
</OBJECT>
```

Create Editor Window

The HTML `OBJECT` tag is used to embed an editing window. The additional custom parameter specifies that the control is to be initialized with a new empty mapping.

```
<OBJECT id="objDoc1"
        Classid="clsid:DFBB0871-DAFE-4502-BB66-08CEB7DF5255"
        width="600"
        height="500"
        VIEWASTEXT>
    <PARAM NAME="NewDocument">
</OBJECT>
```

Create Project Window

The HTML `OBJECT` tag is used to create a MapForceControlPlaceHolder window. The first additional custom parameter defines the placeholder to show the MapForce project window. The second parameter loads one of the example projects delivered with your MapForce installation (located in the `<yourusername>/MyDocuments` folder).

```
<OBJECT id="objProjectWindow"
        Classid="clsid:FDEC3B04-05F2-427d-988C-F03A85DE53C2"
        width="200"
        height="200"
        VIEWASTEXT>
    <PARAM name="PlaceholderWindowID" value="3">
          <PARAM name="FileName" value="MapForceExamples/MapForceExamples.mfp">
</OBJECT>
```

Create Placeholder for Helper Windows

The HTML `OBJECT` tag is used to instantiate a MapForceControlPlaceHolder ActiveX control that can host the different MapForce helper windows. Initially, no helper window is shown. See the example file.

```
<OBJECT id="objEHWindow"
        Classid="clsid:FDEC3B04-05F2-427d-988C-F03A85DE53C2"
        width="200"
        height="200"
        VIEWASTEXT>
```

```
	<PARAM name="PlaceholderWindowID" value="0">
</OBJECT>
```

Three buttons allow us to switch the actual window that will be shown. The JavaScript execute on-button-click sets the property `PlaceHolderWindowID` to the corresponding value defined in

```
<input type="button" value="Library Window" onclick="BtnHelperWindow(0)">
<input type="button" value="Overview Window" onclick="BtnHelperWindow(1)">
<input type="button" value="Validation Window" onclick="BtnHelperWindow(2)">

<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
//
---------------------------------------------------------------------------
// specify which of the windows shall be shown in the placeholder control.
function BtnHelperWindow(i_ePlaceholderWindowID)
{
	objEHWindow.PlaceholderWindowID = i_ePlaceholderWindowID;
}
</SCRIPT>
```

Create a Custom Toolbar

The custom toolbar consists of buttons with images of MapForce commands. The command ID numbers can be found in the button elements shown below

```
<button id="btnInsertXML" title="Insert XML Schema/File"
onclick="BtnDoCommand(13635)">
	  <img src="..\Images\ID_INSERT_XSD.gif" width="16" height="16" />
</button>
<button id="btnInsertDB" title="Insert Database" onclick="BtnDoCommand(13590)">
	  <img src="..\Images\ID_INSERT_DATABASE.gif" width="16" height="16" />
</button>
<button id="btnInsertEDI" title="Insert EDI" onclick="BtnDoCommand(13591)">
	  <img src="..\Images\ID_INSERT_EDI.gif" width="16" height="16" />
</button>
```

On clicking one of these buttons the corresponding command ID is sent to the manager control.

```
<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
// ---------------------------------------------------------------------
// perform any command specified by cmdID.
// command routing includes application, active document and view.
function BtnDoCommand(cmdID)
{
	objMapForceX.Exec( cmdID );
	msgtext.innerText = "Command " + cmdID + " performed.";
}
</SCRIPT>
```


Create More Buttons

In the example, we add some more buttons to show some automation code.

```
<p>
    <input type="button" value="New File" onclick="BtnNewFile(objDoc1)">
    <input type="button" value="Save File" onclick="BtnSaveFile(objDoc1)">
    <input type="text" title="Path" id="strPath" width="150">
    <input type="button" value="Open MarketingExpenses"
onclick="BtnOpenMEFile(objDoc1)">
</p>
<p>
    <input type="button" id="GenerateXSLT"   value="Generate XSLT"
onclick="BtnGenerate( objDoc1, 0 )">
    <input type="button" id="GenerateJava"   value="Generate Java"
onclick="BtnGenerate( objDoc1, 1 )">
    <input type="button" id="GenerateCpp"    value="Generate C++"
onclick="BtnGenerate( objDoc1, 2 )">
    <input type="button" id="GenerateCSharp" value="Generate C#"
onclick="BtnGenerate( objDoc1, 3 )">
</p>
```

The corresponding JavaScript looks like this:

```
<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
// ---------------------------------------------------------
// open a document in the specified document control window.
function BtnOpenMEFile(objDocCtrl)
{
       // do not use MapForceX.Application.OpenDocument(...) to open a document,
       // since then MapForceControl wouldn't know a control window to show
       // the document in. Instead:

       objDocCtrl.OpenDocument("C:\Documents and Settings\username\My Documents\
              Altova\XMLSpy2016\Examples/MarketingExpenses.mfd");
       objDocCtrl.setActive();

}

// ---------------------------------------------------------------------
// open a new empty document in the specified document control window.
function BtnNewFile(objDocCtrl)
{
       objDocCtrl.OpenDocument("");
       objDocCtrl.setActive();
}

// ---------------------------------------------------------------------
// Saves the current file in the specified document control window.
function BtnSaveFile(objDocCtrl)
{
       if(objDocCtrl.Path.length > 0)
              objDocCtrl.SaveDocument();
```

```
        else
        {
                if(strPath.value.length > 0)
                {
                        objDocCtrl.Path = strPath.value;
                        objDocCtrl.SaveDocument();
                }
                else
                {
                        alert("Please set path for the document first!");
                        strPath.focus();
                }
        }

        objDocCtrl.setActive();
}
</SCRIPT>
```

Create Event Handler to Update Button Status

Availability of a command may vary with every mouse click or keystroke. The custom event `OnUpdateCmdUI` of MapForceControl gives us an opportunity to update the enabled/disabled state of buttons associated with MapForce commands. The method `MapForceControl.QueryStatus` is used to query whether a command is enabled or not.

```
<SCRIPT LANGUAGE="javascript">

        function objMapForceX::OnUpdateCmdUI()
        {
        if ( document.readyState == "complete" )        // 'complete'
        {
                // update status of buttons
                GenerateXSLT.disabled =  ! (objDoc1.QueryStatus(13617) & 0x02);
        // not enabled
                GenerateJava.disabled =  ! (objDoc1.QueryStatus(13587) & 0x02);
        // not enabled
                GenerateCpp.disabled =   ! (objDoc1.QueryStatus(13589) & 0x02);
        // not enabled
                GenerateCSharp.disabled = ! (objDoc1.QueryStatus(13588) & 0x02);
        // not enabled

                btnFuncUserDef.disabled = ! (objDoc1.QueryStatus(13633) & 0x02);
                btnFuncUserDefSel.disabled = ! (objDoc1.QueryStatus(13634) & 0x02);
                btnFuncSettings.disabled = ! (objDoc1.QueryStatus(13632) & 0x02);
                btnInsertInput.disabled = ! (objDoc1.QueryStatus(13491) & 0x02);

                btnGenXSLT.disabled = ! (objDoc1.QueryStatus(13617) & 0x02);
                btnGenXSLT2.disabled = ! (objDoc1.QueryStatus(13618) & 0x02);
                btnGenXQuery.disabled = ! (objDoc1.QueryStatus(13586) & 0x02);
                btnGenCPP.disabled = ! (objDoc1.QueryStatus(13589) & 0x02);
                btnGenCSharp.disabled = ! (objDoc1.QueryStatus(13588) & 0x02);
```



```
            btnGenJava.disabled = ! (objDoc1.QueryStatus(13587) & 0x02);
        }
        }

        // set activity status of simulated toolbar
</SCRIPT>
```

18.3.3 Java

MapForce ActiveX components can be accessed from Java code. To allow this, the libraries listed below must reside in the classpath. These libraries are partly delivered with the MapForce Integration Package and are placed in the folder: `JavaAPI` in the MapForce application folder.

- `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 on the Java Classpath.

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 of the example Java project, see the section, Example Java Project.

Rules for mapping the ActiveX Control names to Java

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

The `MapForceActiveX.jar` library

The classes and interfaces contained in the jar file are part of the `com.altova.automation.MapForce` package. They are described in the `MapForceActiveX_JavaDoc.zip` file from the folder `Examples\ActiveX\Java` in the MapForce application folder.

This section

This section shows how some basic MapForce ActiveX functionality can be accessed from Java code. It is organized into the following sub-sections:

- Example Java Project
- Creating the ActiveX Controls
- Loading Data in the Controls
- Basic Event Handling
- Menus
- UI Update Event Handling
- Creating a MapForce Mapping Table

18.3.3.1 Example Java Project

The MapForce installation package contains an example Java project, located in the ActiveX Examples folder of the application folder: `<ApplicationFolder>\Examples\ActiveX\Java\`.

The Java example shows how to integrate the MapForceControl in a common desktop application created with Java. You can test it directly from the command line using the batch file `BuildAndRun.bat`, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list

The Java examples folder contains all the files required to run the example project. These files are listed below:

`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
`MapForceActiveX.jar`	Java classes of the MapForce ActiveX control
`MapForceContainer.java`	Java example source code
`MapForceContainerEventHandler.java`	Java example source code
`XMLTreeDialog.java`	Java example source code
`BuildAndRun.bat`	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
`.classpath`	Eclipse project helper file
`.project`	Eclipse project file
`MapForceActiveX_JavaDoc.zip`	`Javadoc` file containing help documentation for the Java API

What the example does

The example places one MapForce document editor window, the MapForce project window, the MapForce library window and the MapForce validation window in an AWT frame window. It reads out the main menu defined for MapForce and creates an AWT menu with the same structure. You can use this menu or the project window to open and work with files in the document editor.

You can modify the example in any way you like.

The following specific features are described in code listings:

- Creating the ActiveX Controls: Starts MapForce, which is registered as an automation server, or activates MapForce if it is already running.
- Loading Data in the Controls: Locates one of the example documents installed with MapForce and opens it.

- Basic Event Handling: Changes the view of all open documents to Text View. The code also shows how to iterate through open documents.
- Menus: Validates the active document and shows the result in a message box. The code shows how to use output parameters.
- UI Update Event Handling: Shows how to handle MapForce events.
- Creating a MapForce Mapping Table: Shows how to create a MapForce mapping table and prepare it for modal activation.

Running the example from the command line

Open a command prompt window and type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a Java Development Kit (JDK) 7 or later installation on your computer.

Press **Return**. The Java source in `MapForceContainer.java` will be compiled and then executed.

Loading the example in Eclipse

Open Eclipse and use the **Import | Existing Projects into Workspace** command to add the Eclipse project file located in the same folder as this Readme file to your workspace. Since you may not have write-access in this folder it is recommended to tell Eclipse to copy the project files into its workspace. The project `MapForceContainer` will then appear in your Package Explorer or Navigator.

If you want to use the 64-bit version of the MapForce ActiveX control you need to use a 64-bit version of Eclipse.

Select the project and then the command **Run as | Java Application** to execute the example.

Note: You can select a class name or method of the Java API and press F1 to get help for that class or method.

Java source code listing

The Java source code in the example file `MapForceContainer.java` is listed below with comments.

```
001 // access general JAVA-COM bridge classes
002 import com.altova.automation.libs.*;
003
004 //access MapForce Java-COM bridge
005 import com.altova.automation.MapForce.*;
006 import
com.altova.automation.MapForce.Enums.MapForceControlPlaceholderWindow;
007 import com.altova.automation.MapForce.Enums.ICActiveXIntegrationLevel;
008
009 //access AWT and Swing components
010 import java.awt.*;
```



```
011 import java.awt.event.*;
012
013 import javax.swing.*;
014
015 /**
016  * A simple example of a container for MapForce document-level integration
using Java AWT/Swing.
017  * The application's GUI shows a single document editing window and 3
different tool windows:
018  * The project window, the validation tool window and the library window.
019  *
020  * The application's menu gets created by reading it out from the MapForce
control.
021  *
022  * Communication between the project window and the document editing window
gets established
023  * by the event handler for the onOpenedOrFocused event. See
MapForceControlEventsDefaultHandler
024  * for further events.
025  *
026  * Feel free to modify and extend this sample.
027  *
028  * @author Altova GmbH
029  */
030 public class MapForceContainer
031 {
032   /**
033    * MapForce manager control - always needed
034    */
035   public static MapForceControl       mapForceControl = null;
036   /**
037    * MapForceDocument editing control
038    */
039   public static MapForceControlDocument   mapForceDocument = null;
040   /**
041    * Tool windows - MapForce place-holder controls
042    */
043   private static MapForceControlPlaceHolder mapForceProjectToolWindow;
044   private static MapForceControlPlaceHolder mapForceValidationToolWindow;
045   private static MapForceControlPlaceHolder mapForceLibraryToolWindow;
046
047   /**
048    * The hosting frame
049    */
050   private static Frame            frame;
051
052
053   /**
054    * Helper function that initializes the Document control
055    */
056   public static void initMapForceDocument()
057   {
058     try
059     {
060       if ( mapForceDocument != null )
061         frame.remove( mapForceDocument );
062       mapForceDocument = new MapForceControlDocument();
063       frame.add( mapForceDocument, BorderLayout.CENTER );
```

```
064       frame.validate();
065       // move the focus to the document control - used when querying for the
command status while enabling/disabling menu items
066       mapForceDocument.requestFocusInWindow();
067     }
068     catch ( Exception ex )
069     {
070       ex.printStackTrace();
071     }
072   }
073
074   /**
075    * Creation of the tree dialog
076    */
077   private static void loadTable(Mapping rootElement)
078   {
079     MapForceTable dlg = new MapForceTable( rootElement, frame );
080     dlg.pack();
081     dlg.setBounds( 0, 0, 800, 300 );
082     dlg.setLocationRelativeTo(frame);
083     dlg.setVisible( true );
084   }
085
086   /**
087    * The main entry point.
088    * Creates the application's frame and the MapForce windows.
089    * MapForceControl, although not visible, is important for the
coordination between the different ActiveX controls of MapForce.
090    */
091   public static void main( String[] args )
092   {
093     // in case of severe errors somewhere in the ActiveX controls an
AutomationException gets thrown
094     try
095     {
096       // Create the main frame of the application
097       frame = new Frame( "Java ActiveX host window" );
098       frame.setLayout( new BorderLayout() );
099
100       // Create the set of buttons and arrange them in a panel
101       Dimension btnDim = new Dimension ( 145, 25 );
102       JPanel westPanel = new JPanel();
103       westPanel.setPreferredSize( new Dimension( 155, 400 ) );
104       westPanel.setMaximumSize( new Dimension( 155, 800 ) );
105       westPanel.add( new Label( "Open documents") ) ;
106       JButton btnMarkExp = new JButton("MarketingExpenses");
westPanel.add( btnMarkExp );  btnMarkExp.setPreferredSize( btnDim );
107       JButton btnComplPO = new JButton("CompletePO");
westPanel.add( btnComplPO );  btnComplPO.setPreferredSize( btnDim );
108       westPanel.add( new Label( "Create/destroy") );
109       JButton btnProject = new JButton("Project window");
westPanel.add( btnProject );  btnProject.setPreferredSize( btnDim );
110       JButton btnValid = new JButton("Validation window");
westPanel.add( btnValid );    btnValid.setPreferredSize( btnDim );
111       JButton btnLibrary = new JButton("Library window");
westPanel.add( btnLibrary );  btnLibrary.setPreferredSize( btnDim );
112       westPanel.add( new Label( "Create menu") );
113       JButton btnMenu = new JButton("Load file menu");
westPanel.add( btnMenu );   btnMenu.setPreferredSize( btnDim );
114       westPanel.add( new Label( "Component structure") );
```



```
115       JButton btnTree = new JButton("Open table");
westPanel.add( btnTree );   btnTree.setPreferredSize( btnDim );
116
117       // Create the MapForce ActiveX control; the parameter determines that
we want to place document controls and place-holder
118       // controls individually. It gives us full control over the menu, as
well.
119       mapForceControl = new MapForceControl
( ICActiveXIntegrationLevel.ICActiveXIntegrationOnDocumentLevel.getValue(),
false );
120
121
122       // Panel that will hold the Project/Xpath/Attributes windows
123       final JPanel southPanel = new JPanel();
124
125       frame.add( southPanel,    BorderLayout.SOUTH );
126       frame.add( mapForceControl, BorderLayout.NORTH );
127       frame.add( westPanel,   BorderLayout.WEST );
128       initMapForceDocument();
129
130       // Listen in this class to communication events ( e.g.
onOpenedOrFocused is handled )
131       final MapForceContainerEventHandler handlerObject = new
MapForceContainerEventHandler();
132       mapForceControl.addListener( handlerObject );
133
134       // Prepare a shutdown mechanism
135       frame.addWindowListener( new WindowAdapter()
136       {
137         public void windowClosing( WindowEvent ev )
138         {
139           frame.dispose();
140           System.exit(0); }
141       } );
142       frame.setVisible( true );
143
144       // Locate samples installed with the product.
145       final String strExamplesFolder = System.getenv( "USERPROFILE" ) + "\
\My Documents\\Altova\\MapForce2013\\MapForceExamples\\";
146
147       // Create a project window and open the sample project in it
148       mapForceProjectToolWindow = new
MapForceControlPlaceHolder( MapForceControlPlaceholderWindow.MapForceXProjectWin
dow.getValue(), strExamplesFolder + "MapForceExamples.mfp" ) ;
149       mapForceProjectToolWindow.setPreferredSize( new Dimension( 200,
200 ) );
150       // For the beginning hide the project window
151       mapForceProjectToolWindow.setVisible( false ) ;
152       frame.add( mapForceProjectToolWindow, BorderLayout.NORTH ) ;
153
154       // Open the Marketing file when button is pressed
155       btnMarkExp.addActionListener( new ActionListener() {
156         public void actionPerformed(ActionEvent e) {
157           try {
158             // Instruct the Document control to open the file - avoid
calling the open method of MapForceControl (see help)
159             mapForceDocument.open( strExamplesFolder +
"MarketingExpenses.mfd" );
160             mapForceDocument.requestFocusInWindow();
161           } catch (AutomationException e1) {
```

```
            e1.printStackTrace();
          }
        }
      } );

      // Open the Complete file when button is pressed
      btnComplPO.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 + "CompletePO.mfd" );
            mapForceDocument.requestFocusInWindow();
          } catch (AutomationException e1) {
            e1.printStackTrace();
          }
        }
      } );

      // Show/hide the project window when button is pressed
      btnProject.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          if ( !mapForceProjectToolWindow.isVisible() ) {
            // remove the hidden window from the frame (which acted like a
temporary parent)
            frame.remove( mapForceProjectToolWindow ) ;
            southPanel.add( mapForceProjectToolWindow );
            mapForceProjectToolWindow.setVisible( true ) ;
          } else {
            mapForceProjectToolWindow.setVisible( false ) ;
            southPanel.remove( mapForceProjectToolWindow );
            // Add the hidden window to the frame (temporary parent)
            frame.add( mapForceProjectToolWindow, BorderLayout.NORTH ) ;
          }
          frame.validate();
        }
      } ) ;

      // Create/destroy the Validation window when button is pressed
      btnValid.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          if ( mapForceValidationToolWindow == null ) {
            try {
              // Create a new window and add it to the south panel
              mapForceValidationToolWindow = new MapForceControlPlaceHolder(
MapForceControlPlaceholderWindow.MapForceXValidationWindow.getValue(), "" );
              mapForceValidationToolWindow.setPreferredSize( new
Dimension( 200, 200 ) );
              southPanel.add( mapForceValidationToolWindow );
            } catch (AutomationException e1) {
              e1.printStackTrace();
            }
          } else {
            // Remove the window and the reference
            southPanel.remove( mapForceValidationToolWindow );
            mapForceValidationToolWindow = null;
          }
          frame.validate();
        }
      } ) ;
```



```
218
219       // Create/destroy the Library window when button is pressed
220       btnLibrary.addActionListener( new ActionListener() {
221         public void actionPerformed(ActionEvent e) {
222           if ( mapForceLibraryToolWindow == null ) {
223             try {
224               // Create a new window and add it to the south panel
225               mapForceLibraryToolWindow = new
MapForceControlPlaceHolder( MapForceControlPlaceholderWindow.MapForceXLibraryWin
dow.getValue(), "" );
226               mapForceLibraryToolWindow.setPreferredSize( new
Dimension( 200, 200 ) );
227               southPanel.add( mapForceLibraryToolWindow );
228             } catch (AutomationException e1) {
229               e1.printStackTrace();
230             }
231           } else {
232             southPanel.remove( mapForceLibraryToolWindow );
233             mapForceLibraryToolWindow = null;
234           }
235           frame.validate();
236         }
237       } ) ;
238
239       // Load the file menu when the button is pressed
240       btnMenu.addActionListener( new ActionListener() {
241         public void actionPerformed(ActionEvent e) {
242           try {
243             // Create the menubar that will be attached to the frame
244             MenuBar mb = new MenuBar();
245             // Load the main menu's first item - the File menu
246             MapForceCommand xmlSpyMenu =
mapForceControl.getMainMenu().getSubCommands().getItem( 0 );
247             // Create Java menu items from the Commands objects
248             Menu fileMenu = new Menu();
249             handlerObject.fillMenu( fileMenu, xmlSpyMenu.getSubCommands() );
250             fileMenu.setLabel( xmlSpyMenu.getLabel().replace( "&", "" ) );
251             mb.add( fileMenu );
252             frame.setMenuBar( mb );
253             frame.validate();
254           } catch (AutomationException e1) {
255             e1.printStackTrace();
256           }
257           // Disable the button when the action has been performed
258           ((AbstractButton) e.getSource()).setEnabled( false );
259         }
260       } ) ;
261
262       // Load a table when the button is pushed
263       btnTree.addActionListener( new ActionListener() {
264         public void actionPerformed(ActionEvent e) {
265           try {
266             loadTable( mapForceDocument.getDocument().getMainMapping() );
267           } catch (AutomationException e1) {
268             e1.printStackTrace();
269           }
270         }
271       } ) ;
272
273       // Show the main window
```

```
274       frame.setBounds( 0, 0, 800, 600 );
275       frame.validate();
276
277       // feel free to extend.
278     }
279     catch ( AutomationException e )
280     {
281       e.printStackTrace();
282     }
283   }
284
285 }
```

18.3.3.2 Creating the ActiveX Controls

The code listing below show how ActiveX controls can be created. The constructors will create the Java wrapper objects. Adding these Canvas-derived objects to a panel or to a frame will trigger the creation of the wrapped ActiveX object.

```
01   /**
02    * MapForce manager control - always needed
03    */
04   public static MapForceControl         mapForceControl = null;
05
06   /**
07    * MapForceDocument editing control
08    */
09   public static MapForceControlDocument     mapForceDocument = null;
10
11   /**
12    * Tool windows - MapForce place-holder controls
13    */
14   private static MapForceControlPlaceHolder   mapForceProjectToolWindow =
null;
15   private static MapForceControlPlaceHolder   mapForceValidationToolWindow =
null;
16   private static MapForceControlPlaceHolder   mapForceLibraryToolWindow =
null;
17
18   // Create the MapForce ActiveX control; the parameter determines that we
want
     // to place document controls and place-holder controls individually.
19   // It gives us full control over the menu, as well.
20      mapForceControl = new MapForceControl(
        ICActiveXIntegrationLevel.ICActiveXIntegrationOnDocumentLevel.getValue()
, false );
21
22      mapForceDocument = new MapForceControlDocument();
23      frame.add( mapForceDocument, BorderLayout.CENTER );
24
25
26   // Create a project window and open the sample project in it
27      mapForceProjectToolWindow = new MapForceControlPlaceHolder(
        MapForceControlPlaceholderWindow.MapForceXProjectWindow.getValue(),
        strExamplesFolder + "MapForceExamples.mfp" ) ;
```



```
28      mapForceProjectToolWindow.setPreferredSize( new Dimension( 200, 200 ) );
```

18.3.3.3 Loading Data in the Controls

The code listing below show how data can be loaded in the ActiveX controls.

```
1    // Locate samples installed with the product.
2       final String strExamplesFolder = System.getenv( "USERPROFILE" ) +
        "\\My Documents\\Altova\\MapForce2016\\MapForceExamples\\";
3       mapForceProjectToolWindow = new
MapForceControlPlaceHolder( MapForceControlPlaceholderWindow.MapForceXProjectWin
dow.getValue(), strExamplesFolder + "MapForceExamples.mfp" ) ;
```

18.3.3.4 Basic Event Handling

The code listing below shows how basic events can be handled. When calling the MapForceControl's open method, or when trying to open a file via the menu or Project tree, the onOpenedOrFocused event is sent to the attached event handler. The basic handling for this event is opening the file by calling the MapForceDocumentControl's open method.

```
01       // Open the Marketing file when button is pressed
02       btnMarkExp.addActionListener( new ActionListener() {
03         public void actionPerformed(ActionEvent e) {
04           try {
05             // Instruct the Document control to open the file - avoid calling
the open method of MapForceControl (see help)
06             mapForceDocument.open( strExamplesFolder +
"MarketingExpenses.mfd" );
07             mapForceDocument.requestFocusInWindow();
08           } catch (AutomationException e1) {
09             e1.printStackTrace();
10           }
11         }
12       } );
13       public void onOpenedOrFocused( String i_strFileName, boolean
i_bOpenWithThisControl, boolean i_bFileAlreadyOpened ) throws
AutomationException
14   {
15     // Handle the New/Open events coming from the Project tree or from the
menus
16     if ( !i_bFileAlreadyOpened )
17     {
18       // This is basically an SDI interface, so open the file in the already
existing document control
19       try {
20         MapForceContainer.mapForceDocument.open( i_strFileName );
21         MapForceContainer.mapForceDocument.requestFocusInWindow();
22       } catch (Exception e) {
23         e.printStackTrace();
24       }
25     }
```

```
26   }
```

18.3.3.5 Menus

The code listing below shows how menu items can be created. Each `MapForceCommand` object gets a corresponding `MenuItem` object, with the `ActionCommand` set to the ID of the command. The actions generated by all menu items are handled by the same function, which can perform specific handlings (like reinterpreting the closing mechanism) or can delegate the execution to the `MapForceControl` object by calling its `exec` method. The `menuMap` object that is filled during menu creation is used later (see section UI Update Event Handling).

```
01
02       // Load the file menu when the button is pressed
03       btnMenu.addActionListener( new ActionListener() {
04         public void actionPerformed(ActionEvent e) {
05           try {
06             // Create the menubar that will be attached to the frame
07             MenuBar mb = new MenuBar();
08             // Load the main menu's first item - the File menu
09             MapForceCommand xmlSpyMenu =
mapForceControl.getMainMenu().getSubCommands().getItem( 0 );
10             // Create Java menu items from the Commands objects
11             Menu fileMenu = new Menu();
12             handlerObject.fillMenu( fileMenu, xmlSpyMenu.getSubCommands() );
13             fileMenu.setLabel( xmlSpyMenu.getLabel().replace( "&", "" ) );
14             mb.add( fileMenu );
15             frame.setMenuBar( mb );
16             frame.validate();
17           } catch (AutomationException e1) {
18             e1.printStackTrace();
19           }
20           // Disable the button when the action has been performed
21           ((AbstractButton) e.getSource()).setEnabled( false );
22         }
23       } ) ;
24   /**
25    * Populates a menu with the commands and submenus contained in an
MapForceCommands object
26    */
27   public void fillMenu(Menu newMenu, MapForceCommands mapForceMenu) throws
AutomationException
28   {
29     // For each command/submenu in the mapForceMenu
30     for ( int i = 0 ; i < mapForceMenu.getCount() ; ++i  )
31     {
32       MapForceCommand mapForceCommand = mapForceMenu.getItem( i );
33       if ( mapForceCommand.getIsSeparator() )
34         newMenu.addSeparator();
35       else
36       {
37         MapForceCommands subCommands = mapForceCommand.getSubCommands();
38         // Is it a command (leaf), or a submenu?
39         if ( subCommands.isNull() || subCommands.getCount() == 0 )
40         {
41           // Command -> add it to the menu, set its ActionCommand to its ID
```



```
and store it in the menuMap
42            MenuItem mi = new
MenuItem( mapForceCommand.getLabel().replace( "&", "" ) );
43            mi.setActionCommand( "" + mapForceCommand.getID() );
44            mi.addActionListener( this );
45            newMenu.add( mi );
46            menuMap.put( mapForceCommand.getID(), mi );
47          }
48          else
49          {
50            // Submenu -> create submenu and repeat recursively
51            Menu newSubMenu = new Menu();
52            fillMenu( newSubMenu, subCommands );
53            newSubMenu.setLabel( mapForceCommand.getLabel().replace( "&",
"" ) );
54            newMenu.add( newSubMenu );
55          }
56        }
57      }
58    }
59    /**
60     * Action handler for the menu items
61     * Called when the user selects a menu item; the item's action command
corresponds to the command table for MapForce
62     */
63    public void actionPerformed( ActionEvent e )
64    {
65      try
66      {
67        int iCmd = Integer.parseInt( e.getActionCommand() );
68        // Handle explicitly the Close commands
69        switch ( iCmd )
70        {
71          case 57602:       // Close
72          case 34050:       // Close All
73            MapForceContainer.initMapForceDocument();
74            break;
75          default:
76            MapForceContainer.mapForceControl.exec( iCmd );
77            break;
78        }
79      }
80      catch ( Exception ex )
81      {
82        ex.printStackTrace();
83      }
84
85    }
```

18.3.3.6 UI Update Event Handling

The code listing below shows how a UI-Update event handler can be created.

```
01 /**
02     * Call-back from the MapForceControl.
03     * Called to enable/disable commands
04     */
```

```
05   @Override
06   public void onUpdateCmdUI() throws AutomationException
07   {
08     // A command should be enabled if the result of queryStatus contains the
Supported (1) and Enabled (2) flags
09     for ( java.util.Map.Entry<Integer, MenuItem> pair : menuMap.entrySet() )
10
      pair.getValue().setEnabled( MapForceContainer.mapForceControl.queryStatus(
pair.getKey() ) > 2 );
11   }
12  /**
13    * Call-back from the MapForceControl.
14    * Usually called while enabling/disabling commands due to UI updates
15    */
16   @Override
17   public boolean onIsActiveEditor( String i_strFilePath ) throws
AutomationException
18   {
19     try {
20       return
MapForceContainer.mapForceDocument.getDocument().getFullName().equalsIgnoreCase(
i_strFilePath );
21     } catch ( Exception e ) {
22       return false;
23     }
24   }
```

18.3.3.7 Listing the Properties of a MapForce Mapping

The listing below shows how a Mapping object in MapForce can be loaded as a table and prepared for modal activation.

```
01 //access MapForce Java-COM bridge
02 import com.altova.automation.MapForce.*;
03 import com.altova.automation.MapForce.Component;
04 import com.altova.automation.MapForce.Enums.ENUMComponentUsageKind;
05
06 //access AWT and Swing components
07 import java.awt.*;
08 import javax.swing.*;
09 import javax.swing.table.*;
10
11
12 /**
13  * A simple example of a table control loading the structure from a Mapping
object.
14  * The class receives an Mapping object, loads its components in a JTable,
and prepares
15  * for modal activation.
16  *
17  * Feel free to modify and extend this sample.
18  *
19  * @author Altova GmbH
20  */
21 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() {
```

```
75          public String getColumnName(int col) { return columnNames[col]; }
76          public int getRowCount() { return data.length; }
77          public int getColumnCount() { return columnNames.length; }
78          public Object getValueAt(int row, int col) { return data[row][col];
}
79          public boolean isCellEditable(int row, int col) { return false; }
80          public Class getColumnClass(int c) { return getValueAt(0,
c).getClass(); }
81      } );
82
83      // Set width
84      for( index = 0 ; index < columnNames.length ; ++index )
85        myTable.getColumnModel().getColumn( index ).setMinWidth( 80 );
86      myTable.getColumnModel().getColumn( 5 ).setMinWidth( 400 );
87    }
88    catch (Exception e)
89    {
90      e.printStackTrace();
91    }
92  }
93
94 }
```

18.3.4 Visual Basic

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.

18.4 Command Table for MapForce

Tables in this section list the names and identifiers of all commands that are available within MapForce. Every sub-section lists the commands from the corresponding top-level menu of MapForce. The left-most column shows the command's menu text to make it easier for you to identify the functionality behind the command. The last sub-section is a collection of those commands that are not accessible via the main menu.

Depending on the edition of MapForce you have installed, some of these commands might not be supported. See `Query MapForce Commands` on how to query the current resource structure and command availability.

Use the command identifiers with `MapForceControl.QueryStatus` or `MapForceControlDocument.QueryStatus` to check the current status of a command. Use `MapForceControl.Exec` or `MapForceControlDocument.Exec` to execute a command.

18.4.1 File Menu

Commands from the File menu:

Menu Text	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
Save Project	ID_FILE_SAVEPROJECT	32378
Close Project	ID_FILE_CLOSEPROJECT	32355
Print...	IDC_FILE_PRINT	57607

Print Preview	IDC_FILE_PRINT_PREVIEW	57609
Print Setup...	ID_FILE_PRINT_SETUP	57606
Validate Mapping	ID_MAPPING_VALIDATE	32347
Generate code in selected language	ID_FILE_GENERATE_SELECTED_CODE	32362
Generate code in/XSLT 1.0	ID_FILE_GENERATEXSLT	32360
Generate code in/XSLT 2.0	ID_FILE_GENERATEXSLT2	32361
Generate code in/XQuery	ID_FILE_GENERATEXQUERY	32359
Generate code in/Java	ID_FILE_GENERATEJAVACODE	32358
Generate code in/C# (Sharp)	ID_FILE_GENERATECSCODE	32357
Generate code in/C++	ID_FILE_GENERATECPPCODE	32356
Generate documentation...	ID_FILE_GENERATE_DOCUMENTATION	32468
Mapping Settings...	ID_MAPPING_SETTINGS	32396
Recent Files/Recent File	ID_FILE_MRU_FILE1	57616
Recent Projects/Recent Project	ID_FILE_MRU_PROJECT1	32364
Exit	ID_APP_EXIT	57665

18.4.2 Edit Menu

Commands from the Edit menu:

Menu Text	Command Name	ID
Undo	ID_EDIT_UNDO	57643
Redo	ID_EDIT_REDO	57644
Find...	ID_EDIT_FIND	57636
Find next	ID_EDIT_FINDNEXT	32349
Find previous	ID_EDIT_FINDPREV	32350
Cut	ID_EDIT_CUT	57635
Copy	ID_EDIT_COPY	57634
Paste	ID_EDIT_PASTE	57637
Delete	ID_EDIT_CLEAR	57632
Select All	ID_EDIT_SELECT_ALL	57642

18.4.3 Insert Menu

Commands from the Insert menu:

Menu Text	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
Constant	ID_INSERT_CONSTANT	32388
Filter: Nodes/Rows	ID_INSERT_FILTER	32391
SQL-WHERE Condition	ID_INSERT_SQLWHERE_CONDITION	32351
Value-Map	ID_INSERT_VALUEMAP	32354
IF-Else Condition	ID_INSRT_CONDITION	32394
Exception	ID_INSERT_EXCEPTION	32311

18.4.4 Project Menu

Commands from the Project menu:

Menu Text	Command Name	ID
Add Files to Project...	ID_PROJECT_ADDFILESTOPROJECT	32420
Add Active File to Project...	ID_PROJECT_ADDACTIVEFILETOPROJECT	32419
Create Folder	ID_POPUP_PROJECT_CREATE_FOLDER	32310
Open Mapping for Operation	ID_POPUP_OPENOPERATIONSMAPPING	13692
Create Mapping for Operation...	ID_POPUP_CREATEMAPPINGFOROPERATION	32399
Add Mapping File for Operation...	ID_POPUP_PROJECT_ADD_MAPPING	32309
Reload Project	ID_PROJECT_RELOAD	32374
Insert Web Service...	ID_POPUP_PROJECT_INSERT_WEBSERVICE	32306
Open File In XMLSpy	ID_POPUP_PROJECT_OPENINXMLSPY	32305
Generate Code for Entire Project	ID_POPUP_PROJECT_GENERATE_PROJECT	32304
Generate code in/XSLT 1.0	ID_PROJECT_GENERATEXSLT	32425
Generate code in/XSLT 2.0	ID_PROJECT_GENERATEXSLT2	32426
Generate code in/XQuery	ID_PROJECT_GENERATEXQUERY	32424
Generate code in/Java	ID_PROJECT_GENERATEJAVACODE	32423
Generate code in/C# (Sharp)	ID_PROJECT_GENERATECSCODE	32422
Generate code in/C++	ID_PROJECT_GENERATECPPCODE	32421
Project Properties...	ID_PROJECT_PROPERTIES	32404

18.4.5 Component Menu

Commands from the Component menu:

Menu Text	Command Name	ID
Align Tree Left	ID_COMPONENT_LEFTALIGNTREE	32338
Align Tree Right	ID_COMPONENT_RIGHTALIGNTREE	32340
Change Root Element	ID_COMPONENT_CHANGEROOTELEMENT	32334
Edit Schema Definition in XMLSpy	ID_COMPONENT_EDIT_SCHEMA	32337

Edit FlexText Configuration	ID_COMPONENT_EDIT_MFT	32301
Add/Remove Tables...	ID_COMPONENT_SELECTTABLES	32346
Refresh	IDC_COMMAND_REFRESH_COMPONENT	32373
Duplicate Input	ID_COMPONENT_CREATE_DUPLICATE_ICON	32335
Remove Duplicate	ID_COMPONENT_REMOVE_DUPLICATE_ICON	32339
Database Table Actions	ID_POPUP_DATABASETABLEACTIONS	32400
Database Key Settings	ID_POPUP_VALUEKEYSETTINGS	32417
Query Database	ID_QUERY_DATABASE	32341
Properties	ID_COMPONENT_PROPERTIES	32336

18.4.6 Connection Menu

Commands from the Connection menu:

Menu Text	Menu Text	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 Mapping (mixed-content)	ID_POPUP_ORDERBYSOURCECONNECTION	32403
Properties	ID_POPUP_CONNECTION_SETTINGS	32398

18.4.7 Function Menu

Commands from the Function menu:

Menu Text	Command Name	ID
Create User-Defined Function...	ID_FUNCTION_CREATE_EMPTY	32380
Create User-Defined Function From Selection...	ID_FUNCTION_CREATE_FROM_SELECTION	32381
Function Settings...	ID_FUNCTION_SETTINGS	32387
Remove function	ID_FUNCTION_REMOVE	32385
Insert Input	ID_FUNCTION_INSERT_INPUT	32383
Insert Output...	ID_FUNCTION_INSERT_OUTPUT	32402

18.4.8 Output Menu

Commands from the Output menu:

Menu Text	Command Name	ID
XSLT 1.0	ID_SELECT_LANGUAGE_XSLT	32433
XSLT 2.0	ID_SELECT_LANGUAGE_XSLT2	32434
XQuery	ID_SELECT_LANGUAGE_XQUERY	32432
Java	ID_SELECT_LANGUAGE_JAVA	32431
C# (Sharp)	ID_SELECT_LANGUAGE_CSHARP	32430
C++	ID_SELECT_LANGUAGE_CPP	32429
Validate output file	ID_XML_VALIDATE	32458
Save Output File...	IDC_FILE_SAVEGENERATEDOUTPUT	32321
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
Save All Output Files	IDC_FILE_SAVEALLGENERATEDOUTPUT	32374
Regenerate Output	ID_REGENERATE_PREVIEW_OUTPUT	32480
Text View Settings	ID_TEXTVIEWSETTINGSDIALOG	32472

18.4.9 View Menu

Commands from the View menu:

Menu Text	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
Show selected component connectors	ID_VIEW_AUTOHIGHLIGHTCOMPONENTCONNECTIONS	32443
Show connectors from source to target	ID_VIEW_RECURSIVEAUTOHIGHLIGHT	32447
Zoom...	ID_VIEW_ZOOM	32451
Status Bar	ID_VIEW_STATUS_BAR	32449
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

XBRL Display Options	ID_VIEW_XBRL_DISPLAY_OPTIONS	32473
Back	ID_CMD_BACK	32479
Forward	ID_CMD_FORWARD	32478

18.4.10 Tools Menu

Commands from the Tools menu:

Menu Text	Command Name	ID
Global Resources	IDC_GLOBALRESOURCES	37401
Active Configuration	IDC_GLOBALRESOURCES_SUBMENUENTRY1	37408
Customize...	ID_VIEW_CUSTOMIZE	32444
Options...	ID_TOOLS_OPTIONS	32441
Restore Toolbars and Windows	ID_APP_RESET_TOOLBARS_AND_WINDOWS	32956

18.4.11 Window Menu

Commands from the Window menu:

Menu Text	Command Name	ID
Cascade	ID_WINDOW_CASCADE	57650
Tile Horizontal	ID_WINDOW_TILE_HORZ	57651
Tile Vertical	ID_WINDOW_TILE_VERT	57652

18.4.12 Help Menu

Commands from the Help menu:

Menu Text	Command Name	ID
Table of Contents...	IDC_HELP_CONTENTS	32322
Index..	IDC_HELP_INDEX	32323
Search...	IDC_HELP_SEARCH	32324
Software Activation...	IDC_ACTIVATION	32701
Order Form...	IDC_OPEN_ORDER_PAGE	32326
Registration...	IDC_REGISTRATION	32330
Check for Updates...	IDC_CHECK_FOR_UPDATES	32700
Support Center...	IDC_OPEN_SUPPORT_PAGE	32327
FAQ on the Web...	IDC_SHOW_FAQ	32331
Components Download...	IDC_OPEN_COMPONENTS_PAGE	32325
MapForce on the Internet..	IDC_OPEN_XML_SPY_HOME	32328

MapForce Training...	IDC_OPEN_MAPFORCE_TRAINING_PAGE	32300
About MapForce...	ID_APP_ABOUT	57664

18.4.13 Commands Not in Main Menu

Commands not in the main menu:

Menu Text	Command Name	ID
	IDC_QUICK_HELP	32329
Edit FlexText Configuration	ID_COMPONENT_EDIT_MFT	32301
Priority Context	ID_COMPONENT_PRIORITYCONTEXT	32318
	ID_EDIT_FINDPREV	32350
	ID_FUNCTION_GOTO_MAIN	32382
Insert Input	ID_FUNCTION_INSERT_INPUT_AT_POINT	32384
	ID_FUNCTION_REMOVE	32385
Replace component with internal function structure	ID_FUNCTION_REPLACE_WITH_COMPONENTS	32386
	ID_MAPFORCEVIEW_ZOOM	32395
	ID_NEXT_PANE	32397
Add Active File to Project	ID_POPUP_PROJECT_ADDACTIVEFILETOPROJECT	32405
Add Files to Project...	ID_POPUP_PROJECT_ADDFILESTOPROJECT	32406
C++	ID_POPUP_PROJECT_GENERATECPPCODE	32408
C# (Sharp)	ID_POPUP_PROJECT_GENERATECSCODE	32409
Java	ID_POPUP_PROJECT_GENERATEJAVACODE	32410
XQuery	ID_POPUP_PROJECT_GENERATEXQUERY	32411
XSLT 1.0	ID_POPUP_PROJECT_GENERATEXSLT	32412
XSLT 2.0	ID_POPUP_PROJECT_GENERATEXSLT2	32413
Generate All	ID_POPUP_PROJECT_GENERATE_ALL	32303
Generate code in default language	ID_POPUP_PROJECT_GENERATE_CODE	32414
Open	ID_POPUP_PROJECT_OPEN_MAPPING	32307
Properties...	ID_POPUP_PROJECT_PROJECTPROPERTIES	32428
Remove	ID_POPUP_PROJECT_REMOVE	32308
Add/Remove Selections...	ID_POPUP_STRUCTURENODE_SELECTIONS	32491
	ID_PREV_PANE	32418
	ID_TOGGLE_FOLDINGMARGIN	32438
	ID_TOGGLE_INDENTGUIDES	32439
	ID_TOGGLE_NUMLINEMARGIN	32440
	ID_WORD_WRAP	32457

18.5 Accessing MapForceAPI

The focus of this documentation is the ActiveX controls and interfaces required to integrate the MapForce user interface into your application. To allow you to automate or control the functionality of the integrated components, the following properties give you access to the MapForce automation interface (MapForceAPI):

`MapForceControl.Application`
`MapForceControlDocument.Document`
`MapForceControlPlaceHolder.Project`

Some restrictions apply to the usage of the MapForce automation interface when integrating MapForceControl at document-level. See Integration at document level for details.

18.6 Object Reference

Objects:
MapForceCommand
MapForceCommands
MapForceControl
MapForceControlDocument
MapForceControlPlaceHolder

To give access to standard MapForce functionality, objects of the **MapForce automation interface** can be accessed as well. See MapForceControl.Application, MapForceControlDocument.Document and MapForceControlPlaceHolder.Project for more information.

18.6.1 MapForceCommand

Properties:
`ID`
`Label`
`IsSeparator`
`ToolTip`
`StatusText`
`Accelerator`
`SubCommands`

Description:
Each `Command` object can be one of three possible types:

- **Command**: `ID` is set to a value greater `0` and `Label` is set to the command name. `IsSeparator` is `false` and the `SubCommands` collection is empty.
- **Separator**: `IsSeparator` is `true`. `ID` is `0` and `Label` is not set. The `SubCommands` collection is empty.
- (**Sub**) **Menu**: The `SubCommands` collection contains `Command` objects and `Label` is the name of the menu. `ID` is set to `0` and `IsSeparator` is false.

18.6.1.1 Accelerator

Property: `Label` as string

Description:
For command objects that are children of the ALL_COMMANDS collection, this is the accelerator key defined for the command. If the command has no accelerator key assigned, this property returns the empty string.

The string representation of the accelerator key has the following format:

```
[ALT+][CTRL+][SHIFT+]key
```

Where key is converted using the Windows Platform SDK function GetKeyNameText.

18.6.1.2 ID

Property: ID as long

Description:
ID is 0 for separators and menus.
For commands, this is the ID which can be used with Exec and QueryStatus.

18.6.1.3 IsSeparator

Property: IsSeparator as boolean

Description:
True if the command is a separator.

18.6.1.4 Label

Property: Label as string

Description:
Label is empty for separators.
For command objects that are children of the ALL_COMMANDS collection, this is a unique name. Command icons are stored in files with this name. See Query Commands for more information.

For command objects that are children of menus, the label property holds the command's menu text.
For sub-menus, this property holds the menu text.

18.6.1.5 StatusText

Property: Label as string

Description:
For command objects that are children of the ALL_COMMANDS collection, this is the text shown in the status bar when the command is selected.

18.6.1.6 SubCommands

Property: SubCommands as Commands

Description:
The SubCommands collection holds any sub-commands if this command is actually a menu or submenu.

18.6.1.7 ToolTip

Property: ToolTip as string

Description:
For command objects that are children of the ALL_COMMANDS collection, this is the text shown as tool-tip.

18.6.2 MapForceCommands

Properties:
Count
Item

Description:
Collection of Command objects to get access to command labels and IDs of the MapForceControl. Those commands can be executed with the Exec method and their status can be queried with QueryStatus.

18.6.2.1 Count

Property: Count as long

Description:
Number of Command objects on this level of the collection.

18.6.2.2 Item

Property: Item (n as long) as Command

Description:
Gets the command with the index n in this collection. Index is 1-based.

18.6.3 MapForceControl

Properties:
IntegrationLevel
Appearance

Application
BorderStyle
CommandsList
CommandsStructure (deprecated)
EnableUserPrompts
MainMenu
Toolbars

Methods:
Open
Exec
QueryStatus

Events:
OnUpdateCmdUI
OnOpenedOrFocused
OnCloseEditingWindow
OnFileChangedAlert
OnContextChanged
OnDocumentOpened
OnValidationWindowUpdated

This object is a complete ActiveX control and should only be visible if the MapForce library is used in the Application Level mode.

CLSID: A38637E9-5759-4456-A167-F01160CC22C1
ProgID: Altova.MapForceControl

18.6.3.1 Properties

The following properties are defined:

IntegrationLevel
EnableUserPrompts
Appearance
BorderStyle

Command related properties:
CommandsList
MainMenu
Toolbars
CommandsStructure (deprecated)

Access to MapForceAPI:
Application

Appearance

Property: Appearance as short

Dispatch Id: *-520*

Description:
A value not equal to `0` displays a client edge around the control. Default value is `0`.

Application

Property: `Application` as Application

Dispatch Id: *1*

Description:
The `Application` property gives access to the `Application` object of the complete MapForce automation server API. The property is read-only.

BorderStyle

Property: `BorderStyle` as short

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

CommandsList

Property: `CommandList` as `Commands` (read-only)

Dispatch Id: *1004*

Description:
This property returns a flat list of all commands defined available with MapForceControl.

EnableUserPrompts

Property: `EnableUserPrompts` as boolean

Dispatch Id: *1006*

Description:
Setting this property to *false*, disables user prompts in the control. The default value is *true*.

IntegrationLevel

Property: `IntegrationLevel` as `ICActiveXIntegrationLevel`

Dispatch Id: *1000*

Description:
The `IntegrationLevel` property determines the operation mode of the control. See also Integration at the application level and Integration at document level for more information.

Note: It is important to set this property immediately after the creation of the `MapForceControl` object.

MainMenu

Property: `MainMenu as Command` (read-only)

Dispatch Id: *1003*

Description:
This property gives access to the description of the MapForceControl main menu.

Toolbars

Property: `Toolbars` as `Commands` (read-only)

Dispatch Id: *1005*

Description:
This property returns a list of all toolbar descriptions that describe all toolbars available with MapForceControl.

18.6.3.2 Methods

The following methods are defined:

`Open`
`Exec`
`QueryStatus`

Exec

Method: `Exec` (`nCmdID` as long) as boolean

Dispatch Id: *6*

Description:
`Exec` calls the `MapForce` command with the ID `nCmdID`. If the command can be executed, the method returns `true`. See also `CommandsStructure` to get a list of all available commands and `QueryStatus` to retrieve the status of any command.

Open

Method: `Open` (`strFilePath` as string) as boolean

Dispatch Id: *5*

Description:
The result of the method depends on the extension passed in the argument `strFilePath`. If the file extension is `.sps`, a new document is opened. If the file extension is `.svp`, the corresponding project is opened. If a different file extension is passed into the method, the control tries to load the file as a new component into the active document.

Do not use this method to load documents or projects when using the control in document-level integration mode. Instead, use `MapForceControlDocument.Open` and `MapForceControlPlaceHolder.OpenProject`.

QueryStatus

Method: `QueryStatus` (`nCmdID` as long) as long

Dispatch Id: *7*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if `QueryStatus` returns `0` the command ID is not recognized as a valid MapForce command. If `QueryStatus` returns a value of `1` or `5`, the command is disabled.

18.6.3.3 Events

The MapForceControl ActiveX control provides the following connection point events:

OnUpdateCmdUI
OnOpenedOrFocused
OnCloseEditingWindow
OnFileChangedAlert
OnContextChanged

OnDocumentOpened
OnValidationWindowUpdated

OnCloseEditingWindow

Event: `OnCloseEditingWindow` (i_strFilePath as String) as boolean

Dispatch Id: *1002*

Description:
This event is triggered when MapForce needs to close an already open document. As an answer to this event, clients should close the editor window associated with *i_strFilePath*. Returning *true* from this event indicates that the client has closed the document. Clients can return *false* if no specific handling is required and MapForceControl should try to close the editor and destroy the associated document control.

OnContextChanged

Event: `OnContextChanged` (`i_strContextName` as String, `i_bActive` as bool) as bool

Dispatch Id: *1004*

Description:
This event is not used in MapForce

OnDocumentOpened

Event: `OnDocumentOpened` (`objDocument` as Document)

Dispatch Id: *1*

Description:
This event is triggered whenever a document is opened. The argument `objDocument` is a `Document` object from the MapForce automation interface and can be used to query for more details about the document, or perform additional operations. When integrating on document-level, it is often better to use the event `MapForceControlDocument.OnDocumentOpened` instead.

OnFileChangedAlert

Event: `OnFileChangedAlert` (`i_strFilePath` as String) as bool

Dispatch Id: *1001*

Description:
This event is triggered when a file loaded with MapForceControl is changed on the hard disk by another application. Clients should return true, if they handled the event, or false, if MapForce should handle it in its customary way, i.e. prompting the user for reload.

OnLicenseProblem

Event: `OnLicenseProblem` (`i_strLicenseProblemText` as String)

Dispatch Id: *1005*

Description:
This event is triggered when MapForceControl detects that no valid license is available for this control. In case of restricted user licenses this can happen some time after the control has been initialized. Integrators should use this event to disable access to this control's functionality. After returning from this event, the control will block access to its functionality (e.g. show empty windows in its controls and return errors on requests).

OnOpenedOrFocused

Event: `OnOpenedOrFocused` (`i_strFilePath` as String, `i_bOpenWithThisControl` as bool)

Dispatch Id: *1000*

Description:
When integrating at application level, this event informs clients that a document has been opened, or made active by MapForce.

When integrating at document level, this event instructs the client to open the file `i_strFilePath` in a document window. If the file is already open, the corresponding document window should be made the active window.

if `i_bOpenWithThisControl` is `true`, the document must be opened with MapForceControl, since internal access is required. Otherwise, the file can be opened with different editors.

OnToolWindowUpdated

Event: `OnToolWindowUpdated(pToolWnd` as long `)`

Dispatch Id: *1006*

Description:
This event is triggered when the tool window is updated.

OnUpdateCmdUI

Event: `OnUpdateCmdUI ()`

Dispatch Id: *1003*

Description:
Called frequently to give integrators a good opportunity to check status of MapForce commands using `MapForceControl.QueryStatus`. Do not perform long operations in this callback.

OnValidationWindowUpdated

Event: `OnValidationWindowUpdated ()`

Dispatch Id: *3*

Description:
This event is triggered whenever the validation output window, is updated with new information.

18.6.4 MapForceControlDocument

Properties:
`Appearance`
`BorderStyle`
`Document`
`IsModified`
`Path`
`ReadOnly`

Methods:
`Exec`
`New`
`Open`
`QueryStatus`
`Reload`
`Save`
`SaveAs`

Events:
`OnDocumentOpened`
`OnDocumentClosed`
`OnModifiedFlagChanged`
`OnContextChanged`

OnFileChangedAlert
OnActivate

If the MapForceControl is integrated in the Document Level mode each document is displayed in an own object of type `MapForceControlDocument`. The `MapForceControlDocument` contains only one document at the time but can be reused to display different files one after another.

This object is a complete ActiveX control.

```
CLSID: DFBB0871-DAFE-4502-BB66-08CEB7DF5255
ProgID: Altova.MapForceControlDocument
```

18.6.4.1 Properties

The following properties are defined:

ReadOnly
IsModified
Path
Appearance
BorderStyle

Access to MapForceAPI:
Document

Appearance

Property: `Appearance` as short

Dispatch Id: *-520*

Description:
A value not equal to `0` displays a client edge around the document control. Default value is `0`.

BorderStyle

Property: `BorderStyle` as short

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

Document

Property: `Document` as Document

Dispatch Id: *1*

Description:
The `Document` property gives access to the `Document` object of the MapForce automation server API. This interface provides additional functionality which can be used with the document loaded in the control. The property is read-only.

IsModified

Property: `IsModified` as boolean (read-only)

Dispatch Id: *1006*

Description:
`IsModified` is *true* if the document content has changed since the last open, reload or save operation. It is *false*, otherwise.

Path

Property: `Path` as string

Dispatch Id: *1005*

Description:
Sets or gets the full path name of the document loaded into the control.

ReadOnly

Property: `ReadOnly` as boolean

Dispatch Id: *1007*

Description:
Using this property you can turn on and off the read-only mode of the document. If `ReadOnly` is `true` it is not possible to do any modifications.

18.6.4.2 Methods

The following methods are defined:

Document handling:
New
Open
Reload
Save
SaveAs

Command Handling:
Exec
QueryStatus

Exec

Method: `Exec` (`nCmdID` as long) as boolean

Dispatch Id: *8*

Description:
`Exec` calls the MapForce command with the ID `nCmdID`. If the command can be executed, the method returns `true`. The client should call the `Exec` method of the document control if there is currently an active document available in the application.

See also `CommandsStructure` to get a list of all available commands and `QueryStatus` to retrieve the status of any command.

New

Method: `New` () as boolean

Dispatch Id: *1000*

Description:
This method initializes a new document inside the control..

Open

Method: `Open` (`strFileName` as string) as boolean

Dispatch Id: *1001*

Description:

Open loads the file strFileName as the new document into the control.

QueryStatus

Method: QueryStatus (nCmdID as long) as long

Dispatch Id: *9*

Description:
QueryStatus returns the enabled/disabled and checked/unchecked status of the command specified by nCmdID. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if QueryStatus returns 0 the command ID is not recognized as a valid MapForce command. If QueryStatus returns a value of 1 or 5 the command is disabled. The client should call the QueryStatus method of the document control if there is currently an active document available in the application.

Reload

Method: Reload () as boolean

Dispatch Id: *1002*

Description:
Reload updates the document content from the file system.

Save

Method: Save () as boolean

Dispatch Id: *1003*

Description:
Save saves the current document at the location Path.

SaveAs

Method: `SaveAs` (`strFileName` as string) as boolean

Dispatch Id: *1004*

Description:
`SaveAs` sets `Path` to *strFileName* and then saves the document to this location.

18.6.4.3 Events

The MapForceControlDocument ActiveX control provides following connection point events:

`OnDocumentOpened`
`OnDocumentClosed`
`OnModifiedFlagChanged`
`OnContextChanged`
`OnFileChangedAlert`
`OnActivate`
`OnSetEditorTitle`

OnActivate

Event: `OnActivate` ()

Dispatch Id: *1005*

Description:
This event is triggered when the document control is activated, has the focus, and is ready for user input.

OnContextChanged

Event: `OnContextChanged` (`i_strContextName` as String, `i_bActive` as bool) as bool

Dispatch Id: *1004*

Description: None

OnDocumentClosed

Event: `OnDocumentClosed` (`objDocument` as Document)

Dispatch Id: *1001*

Description:

This event is triggered whenever the document loaded into this control is closed. The argument `objDocument` is a `Document` object from the MapForce automation interface and should be used with care.

OnDocumentOpened

Event: `OnDocumentOpened` (`objDocument` as Document)

Dispatch Id: *1000*

Description:
This event is triggered whenever a document is opened in this control. The argument `objDocument` is a `Document` object from the MapForce automation interface, and can be used to query for more details about the document, or perform additional operations.

OnDocumentSaveAs

Event: `OnContextDocumentSaveAs` (`i_strFileName` as String)

Dispatch Id: *1007*

Description:
This event is triggered when this document gets internally saved under a new name.

OnFileChangedAlert

Event: `OnFileChangedAlert` () as bool

Dispatch Id: *1003*

Description:
This event is triggered when the file loaded into this document control is changed on the hard disk by another application. Clients should return true, if they handled the event, or false, if MapForce should handle it in its customary way, i.e. prompting the user for reload.

OnModifiedFlagChanged

Event: `OnModifiedFlagChanged` (`i_bIsModified` as boolean)

Dispatch Id: *1002*

Description:
This event gets triggered whenever the document changes between modified and unmodified state. The parameter *i_bIsModifed* is *true* if the document contents differs from the original content, and *false*, otherwise.

OnSetEditorTitle

Event: OnSetEditorTitle ()

Dispatch Id: *1006*

Description:
This event is being raised when the contained document is being internally renamed.

18.6.5 MapForceControlPlaceHolder

Properties available for all kinds of placeholder windows:
`PlaceholderWindowID`

Properties for project placeholder window:
`Project`

Methods for project placeholder window:
`OpenProject`
`CloseProject`

The `MapForceControlPlaceHolder` control is used to show the additional MapForce windows like Overview, Library or Project window. It is used like any other ActiveX control and can be placed anywhere in the client application.

```
CLSID: FDEC3B04-05F2-427d-988C-F03A85DE53C2
ProgID: Altova.MapForceControlPlaceHolder
```

18.6.5.1 Properties

The following properties are defined:

`PlaceholderWindowID`

Access to MapForceAPI:
`Project`

Label

Property: `Label` as String (read-only)

Dispatch Id: *1001*

Description:
This property gives access to the title of the placeholder. The property is read-only.

PlaceholderWindowID

Property: `PlaceholderWindowID` as

Dispatch Id: *1*

Description:
Using this property the object knows 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 enumeration. The control changes its state immediately and shows the new MapForce window.

Project

Property: `Project` as Project (read-only)

Dispatch Id: *2*

Description:
The `Project` property gives access to the `Project` object of the MapForce automation server API. This interface provides additional functionality which can be used with the project loaded into the control. The property will return a valid project interface only if the placeholder window has `PlaceholderWindowID` with a value of `MapForceXProjectWindow (=3)`. The property is read-only.

18.6.5.2 Methods

The following method is defined:

`OpenProject`
`CloseProject`

OpenProject

Method: `OpenProject` (`strFileName` as string) as boolean

Dispatch Id: *3*

Description:
`OpenProject` loads the file `strFileName` as the new project into the control. The method will fail if the placeholder window has a `PlaceholderWindowID` different to `XMLSpyXProjectWindow (=3)`.

CloseProject

Method: `CloseProject ()`

Dispatch Id: *4*

Description:
`CloseProject` closes the project loaded the control. The method will fail if the placeholder window has a `PlaceholderWindowID` different to `MapForceXProjectWindow (=3)`.

18.6.5.3 Events

The MapForceControlPlaceholder ActiveX control provides following connection point events:

`OnModifiedFlagChanged`

OnModifiedFlagChanged

Event: `OnModifiedFlagChanged (i_bIsModified as boolean)`

Dispatch Id: *1*

Description:
This event gets triggered only for placeholder controls with a `PlaceholderWindowID` of `MapForceXProjectWindow (=3)`. The event is fired whenever the project content changes between modified and unmodified state. The parameter *i_bIsModifed* is *true* if the project contents differs from the original content, and *false*, otherwise.

OnSetLabel

Event: `OnSetLabel(i_strNewLabel as string)`

Dispatch Id: *1000*

Description:
Raised when the title of the placeholder window is changed.

18.6.6 Enumerations

The following enumerations are defined:

`ICActiveXIntegrationLevel`
`MapForceControlPlaceholderWindow`

18.6.6.1 ICActiveXIntegrationLevel

Possible values for the IntegrationLevel property of the MapForceControl.

```
ICActiveXIntegrationOnApplicationLevel  = 0
ICActiveXIntegrationOnDocumentLevel     = 1
```

18.6.6.2 MapForceControlPlaceholderWindow

This enumeration contains the list of the supported additional MapForce windows.

```
MapForceXNoWindow           = -1
MapForceXLibraryWindow      =  0
MapForceXOverviewWindow     =  1
MapForceXValidationWindow   =  2
MapForceXProjectWindow      =  3
```

Chapter 19

Appendices

19 Appendices

These appendices contain technical information about MapForce and important licensing information. Each appendix contains sub-sections as given below:

Technical Data

- OS and memory requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage
- License metering

License Information

- Electronic software distribution
- Copyrights
- End User License Agreement

19.1 Engine information

This section contains information about implementation-specific features of the Altova XML Validator, Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery Engine.

19.1.1 XSLT and XQuery Engine Information

The XSLT and XQuery engines of MapForce follow the W3C specifications closely and are therefore stricter than previous Altova engines—such as those in previous versions of XMLSpy. As a result, minor errors that were ignored by previous engines are now flagged as errors by MapForce.

For example:

- It is a type error (`err:XPTY0018`) if the result of a path operator contains both nodes and non-nodes.
- It is a type error (`err:XPTY0019`) if `E1` in a path expression `E1/E2` does not evaluate to a sequence of nodes.

If you encounter this kind of error, modify either the XSLT/XQuery document or the instance document as appropriate.

This section describes implementation-specific features of the engines, organized by specification:

- XSLT 1.0
- XSLT 2.0
- XQuery 1.0

19.1.1.1 XSLT 1.0

The XSLT 1.0 Engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999. Note the following information about the implementation.

Notes about the implementation

When the `method` attribute of `xsl:output` is set to HTML, or if HTML output is selected by default, then special characters in the XML or XSLT file are inserted in the HTML document as HTML character references in the output. For instance, the character ` ` (the decimal character reference for a non-breaking space) is inserted as ` ` in the HTML code.

19.1.1.2 XSLT 2.0

This section:

- Engine conformance

Conformance

The XSLT 2.0 engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XSLT 2.0 Recommendation of 23 January 2007 and XPath 2.0 Recommendation of 14 December 2010.

Backwards Compatibility

The XSLT 2.0 engine is backwards compatible. The only time the backwards compatibility of the XSLT 2.0 engine comes into effect is when using the XSLT 2.0 engine to process an XSLT 1.0 stylesheet. Note that there could be differences in the outputs produced by the XSLT 1.0 Engine and the backwards-compatible XSLT 2.0 engine.

Namespaces

Your XSLT 2.0 stylesheet should declare the following namespaces in order for you to be able to use the type constructors and functions available in XSLT 2.0. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
XPath 2.0 functions	`fn:`	`http://www.w3.org/2005/xpath-functions`

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    ...
</xsl:stylesheet>
```

The following points should be noted:

- The XSLT 2.0 engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.
- When using type constructors and types from the XML Schema namespace, the prefix

used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).

- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string('Hello')`, the expression evaluates as `fn:string('Hello')`—not as `xs:string('Hello')`.

Schema-awareness
The XSLT 2.0 engine is schema-aware. So you can use user-defined schema types and the `xsl:validate` instruction.

Implementation-specific behavior
Given below is a description of how the XSLT 2.0 engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

`xsl:result-document`
Additionally supported encodings are (the Altova-specific): `x-base16tobinary` and `x-base64tobinary`.

`function-available`
The function tests for the availability of in-scope functions (XSLT 2.0, XPath 2.0, and extension functions).

`unparsed-text`
The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol. Additionally supported encodings are (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

`unparsed-text-available`
The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol. Additionally supported encodings are (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

Note: The following encoding values, which were implemented in earlier versions of RaptorXML's predecessor product, AltovaXML, are now deprecated: `base16tobinary`, `base64tobinary`, `binarytobase16` and `binarytobase64`.

19.1.1.3 XQuery 1.0

This section:

Conformance
The XQuery 1.0 Engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XQuery 1.0 Recommendation of 14 December 2010. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the XQuery 1.0 Engine implements these features.

Schema awareness
The XQuery 1.0 Engine is **schema-aware**.

Encoding
The UTF-8 and UTF-16 character encodings are supported.

Namespaces
The following namespace URIs and their associated bindings are pre-defined.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
Schema instance	`xsi:`	`http://www.w3.org/2001/XMLSchema-instance`
Built-in functions	`fn:`	`http://www.w3.org/2005/xpath-functions`
Local functions	`local:`	`http://www.w3.org/2005/xquery-local-functions`

The following points should be noted:

- The XQuery 1.0 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.

- Since the built-in functions namespace listed above is the default functions namespace in XQuery, the `fn:` prefix does not need to be used when built-in functions are invoked (for example, `string("Hello")` will call the `fn:string` function). However, the prefix `fn:` can be used to call a built-in function without having to declare the namespace in the query prolog (for example: `fn:string("Hello")`).
- You can change the default functions namespace by declaring the `default function namespace` expression in the query prolog.
- When using types from the XML Schema namespace, the prefix `xs:` may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: `xs:date` and `xs:yearMonthDuration`.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Note that the `untypedAtomic`, `dayTimeDuration`, and `yearMonthDuration` datatypes have been moved, with the CRs of 23 January 2007, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is reported. Note, however, that some functions have the same name as schema datatypes, e.g. `fn:string` and `fn:boolean`. (Both `xs:string` and `xs:boolean` are defined.) The namespace prefix determines whether the function or type constructor is used.

XML source document and validation

XML documents used in executing an XQuery document with the XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

Static and dynamic type checking

The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression `xs:string("1") + 1` returns an error because the addition operation cannot be carried out on an operand of type `xs:string`.

Library Modules

Library modules store functions and variables so they can be reused. The XQuery 1.0 Engine

supports modules that are stored in **a single external XQuery file**. Such a module file must contain a `module` declaration in its prolog, which associates a target namespace. Here is an example module:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the `import module` statement in the query prolog. The `import module` statement only imports functions and variables declared directly in the library module file. As follows:

```
import module namespace modlib = "urn:module-library" at "modulefilename.xq";

if        ($modlib:company = "Altova")
then       modlib:webaddress()
else       error("No match found.")
```

External functions

External functions are not supported, i.e. in those expressions using the `external` keyword, as in:

```
declare function hoo($param as xs:integer) as xs:string external;
```

Collations

The default collation is the Unicode-codepoint collation, which compares strings on the basis of their Unicode codepoint. Other supported collations are the ICU collations listed here. To use a specific collation, supply its URI as given in the list of supported collations. Any string comparisons, including for the `fn:max` and `fn:min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Precision of numeric types

- The `xs:integer` datatype is arbitrary-precision, i.e. it can represent any number of digits.
- The `xs:decimal` datatype has a limit of 20 digits after the decimal point.
- The `xs:float` and `xs:double` datatypes have limited-precision of 15 digits.

XQuery Instructions Support

The `Pragma` instruction is not supported. If encountered, it is ignored and the fallback expression is evaluated.

19.1.2 XSLT and XPath/XQuery Functions

This section lists Altova extension functions and other extension functions that can be used in XPath and/or XQuery expressions. Altova extension functions can be used with Altova's XSLT and XQuery engines, and provide functionality additional to that available in the function libraries defined in the W3C standards.

General points

The following general points should be noted:

- Functions from the core function libraries defined in the W3C specifications can be called without a prefix. That's because the XSLT and XQuery engines read non-prefixed functions as belonging to a default functions namespace which is that specified in the XPath/XQuery functions specifications`http://www.w3.org/2005/xpath-functions`. If this namespace is explicitly declared in an XSLT or XQuery document, the prefix used in the namespace declaration can also optionally be used on function names.
- In general, if a function expects a sequence of one item as an argument, and a sequence of more than one item is submitted, then an error is returned.
- All string comparisons are done using the Unicode codepoint collation.
- Results that are QNames are serialized in the form `[prefix:]localname`.

Precision of xs:decimal

The precision refers to the number of digits in the number, and a minimum of 18 digits is required by the specification. For division operations that produce a result of type `xs:decimal`, the precision is 19 digits after the decimal point with no rounding.

Implicit timezone

When two `date`, `time`, or `dateTime` values need to be compared, the timezone of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `implicit-timezone()` function.

Collations

The default collation is the Unicode codepoint collation, which compares strings on the basis of their Unicode codepoint. Other supported collations are the ICU collations listed below. To use a specific collation, supply its URI as given in the list of supported collations (*table below*). Any string comparisons, including for the `max` and `min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Language	URIs
`da`: Danish	`da_DK`

de: German	de_AT, de_BE, de_CH, de_DE, de_LI, de_LU
en: English	en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA, en_GB, en_GU, en_HK, en_IE, en_IN, en_JM, en_MH, en_MP, en_MT, en_MU, en_NA, en_NZ, en_PH, en_PK, en_SG, en_TT, en_UM, en_US, en_VI, en_ZA, en_ZW
es: Spanish	es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE
fr: French	fr_BE, fr_BF, fr_BI, fr_BJ, fr_BL, fr_CA, fr_CD, fr_CF, fr_CG, fr_CH, fr_CI, fr_CM, fr_DJ, fr_FR, fr_GA, fr_GN, fr_GP, fr_GQ, fr_KM, fr_LU, fr_MC, fr_MF, fr_MG, fr_ML, fr_MQ, fr_NE, fr_RE, fr_RW, fr_SN, fr_TD, fr_TG
it: Italian	it_CH, it_IT
ja: Japanese	ja_JP
nb: Norwegian Bokmal	nb_NO
nl: Dutch	nl_AW, nl_BE, nl_NL
nn: Nynorsk	nn_NO
pt: Portuguese	pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST
ru: Russian	ru_MD, ru_RU, ru_UA
sv: Swedish	sv_FI, sv_SE

Namespace axis
The namespace axis is deprecated in XPath 2.0. Use of the namespace axis is, however, supported. To access namespace information with XPath 2.0 mechanisms, use the `in-scope-prefixes()`, `namespace-uri()` and `namespace-uri-for-prefix()` functions.

19.1.2.1 Altova Extension Functions

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

Functions defined in the W3C's XPath/XQuery Functions specifications can be used in: (i) XPath expressions in an XSLT context, and (ii) in XQuery expressions in an XQuery document. In this documentation we indicate the functions that can be used in the former context (XPath in XSLT) with an **`XP`** symbol and call them XPath functions; those functions that can be used in the latter

(XQuery) context are indicated with an XQ symbol; they work as XQuery functions. The W3C's XSLT specifications—not XPath/XQuery Functions specifications—also define functions that can be used in XPath expressions in XSLT documents. These functions are marked with an XSLT symbol and are called XSLT functions. The XPath/XQuery and XSLT versions in which a function can be used are indicated in the description of the function (*see symbols below*). Functions from the XPath/XQuery and XSLT function libraries are listed without a prefix. Extension functions from other libraries, such as Altova extension functions, are listed with a prefix.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

XSLT functions

XSLT functions can only be used in XPath expressions in an XSLT context (similarly to XSLT 2.0's `current-group()` or `key()` functions). These functions are not intended for, and will not work in, a non-XSLT context (for instance, in an XQuery context). Note that XSLT functions for XBRL can be used only with editions of Altova products that have XBRL support.

XPath/XQuery functions

XPath/XQuery functions can be used both in XPath expressions in XSLT contexts as well as in XQuery expressions:

- Date/Time
- Geolocation
- Image-related
- Numeric
- Sequence
- String
- Miscellaneous

Chart functions (Enterprise and Server Editions only)

Altova extension functions for charts are supported only in the Enterprise and Server Editions of Altova products and enable charts to be generated from XML data.

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
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

Standard functions

distinct-nodes [altova:]

altova:distinct-nodes(*node()**)** as **node()*** XSLT1 XSLT2 XSLT3

Takes a set of one or more nodes as its input and returns the same set minus nodes with duplicate values. The comparison is done using the XPath/XQuery function `fn:deep-equal`.

Examples

- **altova:distinct-nodes**(`country`) returns all child `country` nodes less those having duplicate values.

evaluate [altova:]

altova:evaluate(XPathExpression *as xs:string***[, ValueOf$p1, ... ValueOf$pN])** XSLT1 XSLT2 XSLT3

Takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression. For example: **altova:evaluate('//Name[1]')** returns the contents of the first `Name` element in the document. Note that the expression `//Name[1]` is passed as a string by enclosing it in single quotes.

The `altova:evaluate` function can optionally take additional arguments. These arguments are the values of in-scope variables that have the names `p1`, `p2`, `p3`... `pN`. Note the following points about usage: (i) The variables must be defined with names of the form `pX`, where `X` is an integer; (ii) the `altova:evaluate` function's arguments (*see signature above*), from the second argument onwards, provide the values of the variables, with the sequence of the arguments corresponding to the numerically ordered sequence of variables: `p1` to `pN`: The second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on; (iii) The variable values must be of type `item*`.

Example

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
```

outputs `"hi 20 10"`

In the listing above, notice the following:

- The second argument of the `altova:evaluate` expression is the value assigned to the variable `$p1`, the third argument that assigned to the variable `$p2`, and so on.
- Notice that the fourth argument of the function is a string value, indicated by its being enclosed in quotes.
- The `select` attribute of the `xs:variable` element supplies the XPath expression. Since this expression must be of type `xs:string`, it is enclosed in single quotes.

Examples to further illustrate the use of variables

- ```
  <xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, //Name[1])" />
  ```
  Outputs value of the first `Name` *element.*

- ```
  <xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, '//Name[1]')" />
  ```
  Outputs "//Name[1]"

The `altova:evaluate()` extension function is useful in situations where an XPath expression in the XSLT stylesheet contains one or more parts that must be evaluated dynamically. For example, consider a situation in which a user enters his request for the sorting criterion and this criterion is stored in the attribute `UserReq/@sortkey`. In the stylesheet, you could then have the expression: `<xsl:sort select="`**`altova:evaluate(../UserReq/@sortkey)`**`" order="ascending"/>`. The `altova:evaluate()` function reads the `sortkey` attribute of the `UserReq` child element of the parent of the context node. Say the value of the `sortkey` attribute is `Price`, then `Price` is returned by the `altova:evaluate()` function and becomes the value of the `select` attribute: `<xsl:sort select="`**`Price`**`" order="ascending"/>`. If this `sort` instruction occurs within the context of an element called `Order`, then the `Order` elements will be sorted according to the values of their `Price` children. Alternatively, if the value of `@sortkey` were, say, `Date`, then the `Order` elements would be sorted according to the values of their `Date` children. So the sort criterion for `Order` is selected from the `sortkey` attribute at runtime. This could not have been achieved with an expression like: `<xsl:sort select="`**`../UserReq/@sortkey`**`" order="ascending"/>`. In the case shown above, the sort criterion would be the `sortkey` attribute itself, not `Price` or `Date` (or any other current content of `sortkey`).

Note: The static context includes namespaces, types, and functions—but not variables—from the calling environment. The base URI and default namespace are inherited.

More examples

- Static variables: `<xsl:value-of select="$i3, $i2, $i1" />`
  Outputs the values of three variables.

- Dynamic XPath expression with dynamic variables:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />
  ```
  Outputs "30 20 10"

- Dynamic XPath expression with no dynamic variable:

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath)" />
```

Outputs error: No variable defined for $p3.

▼ **encode-for-rtf [altova:]**

altova:encode-for-rtf(input *as xs:string*, **preserveallwhitespace** *as xs:boolean*, **preservenewlines** *as xs:boolean***)** as **xs:string** XSLT2 XSLT3

Converts the input string into code for RTF. Whitespace and new lines will be preserved according to the boolean value specified for their respective arguments.

[Top]

XBRL functions

Altova XBRL functions can be used only with editions of Altova products that have XBRL support.

▼ **xbrl-footnotes [altova:]**

altova:xbrl-footnotes(*node()***)** as **node()*** XSLT2 XSLT3

Takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

▼ **xbrl-labels [altova:]**

altova:xbrl-labels(*xs:QName*, *xs:string***)** as **node()*** XSLT2 XSLT3

Takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL label nodes associated with the input node.

[Top]

XPath/XQuery Functions: Date and Time

Altova's date/time extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data held as XML Schema's various date and time datatypes. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future

versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

▼ Grouped by functionality

- Add duration to xs:dateTime and return xs:dateTime
- Add a duration to xs:date and return xs:date
- Add a duration to xs:time and return xs:time
- Format and retrieve durations
- Remove timezone from functions that generate current date/time
- Return weekday as integer from date
- Return week number as integer from date
- Build date, time, or duration type from lexical components of each type
- Construct date, dateTime, or time type from string input
- Age-related functions

▼ Grouped alphabetically

altova:add-days-to-date
altova:add-days-to-dateTime
altova:add-hours-to-dateTime
altova:add-hours-to-time
altova:add-minutes-to-dateTime
altova:add-minutes-to-time
altova:add-months-to-date
altova:add-months-to-dateTime
altova:add-seconds-to-dateTime
altova:add-seconds-to-time
altova:add-years-to-date
altova:add-years-to-dateTime
altova:age
altova:age-details
altova:build-date
altova:build-duration
altova:build-time
altova:current-dateTime-no-TZ
altova:current-date-no-TZ
altova:current-time-no-TZ
altova:format-duration
altova:parse-date
altova:parse-dateTime
altova:parse-duration
altova:parse-time
altova:weekday-from-date
altova:weekday-from-dateTime
altova:weeknumber-from-date
altova:weeknumber-from-dateTime

[Top]

Add a duration to xs:dateTime XP3 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

Adds a duration in seconds to an `xs:dateTime` (*see examples below*). The second argument is the number of seconds to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), 20) returns 2014-01-15T14:00:30
- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), -5) returns 2014-01-15T14:00:05

[Top]

Add a duration to xs:date XP3 XQ3

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 XQ3

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 XQ3

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 XQ3

Adds a duration in days to a date. The second argument is the number of days to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

Examples

- **altova:add-days-to-date**(xs:date("2014-01-15"), 10) returns 2014-01-25
- **altova:add-days-to-date**(xs:date("2014-01-15"), -8) returns 2014-01-07

[Top]

Format and retrieve durations XP3 XQ3

These functions add a duration to **xs:date** and return **xs:date**. The `xs:date` type has a format of `CCYY-MM-DD`.

format-duration [altova:]

altova:format-duration(Duration *as xs:duration*, **Picture** *as xs:string*) as **xs:string** XP3 XQ3

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 XQ3

Takes a patterned string as the first argument, and a picture string as the second argument. The input string is parsed on the basis of the picture string, and an xs:duration is returned.

Examples

- **altova:parse-duration**("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "P2DT2H53M11.7S"
- **altova:parse-duration**("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "P3M2DT2H53M"

[Top]

Add a duration to xs:time XP3 XQ3

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 XQ3

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 XQ3

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 XQ3

Adds a duration in seconds to a time. The second argument is the number of seconds to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`. The Seconds component can be in the range of `0` to `59.999`.

Examples

- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20) returns 14:00:20
- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20.895) returns 14:00:20.895

[Top]

Remove the timezone part from date/time datatypes XP3 XQ3

These functions remove the timezone from the current **xs:dateTime**, **xs:date**, or **xs:time** values, respectively. Note that the difference between `xs:dateTime` and `xs:dateTimeStamp` is that in the case of the latter the timezone part is required (while it is optional in the case of the former). So the format of an `xs:dateTimeStamp` value is: `CCYY-MM-DDThh:mm:ss.sss±hh:mm`. or `CCYY-MM-DDThh:mm:ss.sssZ`. If the date and time is read from the system clock as `xs:dateTimeStamp`, the `current-dateTime-no-TZ()` function can be used to remove the timezone if so required.

current-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() as **xs:dateTime** XP3 XQ3

This function takes no argument. It removes the timezone part of `current-dateTime()` (which is the current date-and-time according to the system clock) and returns an `xs:dateTime` value.

Examples

If the current dateTime is **2014-01-15T14:00:00+01:00**:

- **altova:current-dateTime-no-TZ**() returns 2014-01-15T14:00:00

current-date-no-TZ [altova:]

altova:current-date-no-TZ() as **xs:date** XP3 XQ3

This function takes no argument. It removes the timezone part of `current-date()` (which is the current date according to the system clock) and returns an `xs:date` value.

Examples

If the current date is **2014-01-15+01:00**:

- **altova:current-date-no-TZ**() returns 2014-01-15

current-time-no-TZ [altova:]

altova:current-time-no-TZ() as **xs:time** XP3 XQ3

This function takes no argument. It removes the timezone part of `current-time()` (which is the current time according to the system clock) and returns an `xs:time` value.

Examples

If the current time is **14:00:00+01:00**:

- **altova:current-time-no-TZ**( ) returns 14:00:00

[Top]

Return the weekday from xs:dateTime or xs:date XP3 XQ3

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 XQ3

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 XQ3

Takes a date-with-time as its first argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Monday=1. If the second (integer) argument is 0, then the weekdays are numbered 1 to 7 starting with Sunday=1. If the second argument is an integer other than 0, then Monday=1. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

Examples

- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 1) returns 1, which would indicate a Monday
- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 4) returns 1, which would indicate a Monday
- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 0) returns 2, which would indicate a Monday.

weekday-from-date [altova:]

altova:weekday-from-date(Date *as xs:date***)** as **xs:integer** XP3 XQ3

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 XQ3
Takes a date as its first argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Monday=1. If the second (Format) argument is 0, then the weekdays are numbered 1 to 7 starting with Sunday=1. If the second argument is an integer other than 0, then Monday=1. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

Examples

- **altova:weekday-from-date**(xs:date("2014-02-03"), 1) returns 1, which would indicate a Monday
- **altova:weekday-from-date**(xs:date("2014-02-03"), 4) returns 1, which would indicate a Monday
- **altova:weekday-from-date**(xs:date("2014-02-03"), 0) returns 2, which would indicate a Monday.

[Top]

Return the week number from xs:dateTime or xs:date XP2 XQ1 XP3 XQ3

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 XQ3
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 XQ3

Returns the week number of the submitted **DateTime** argument as an integer. The second argument (**Calendar**) specifies the calendar system to follow.

Supported **Calendar** values are:

- **0 = US calendar** (*week starts Sunday*)
- **1 = ISO standard, European calendar** (*week starts Monday*)
- **2 = Islamic calendar** (*week starts Saturday*)

Default is **0**.

Examples

- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 0)` returns `13`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 1)` returns `12`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 2)` returns `13`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00") )` returns `13`

The day of the dateTime in the examples above (`2014-03-23T00:00:00`) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

[Top]

Build date, time, and duration datatypes from their lexical components XP3 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

Takes two arguments to build a value of type xs:yearMonthDuration. The first arguments provides the Years part of the duration value, while the second argument provides the Months part. If the second (Months) argument is greater than or equal to 12, then the integer is divided by 12; the quotient is added to the first argument to provide the Years part of the duration value while the remainder (of the division) provides the Months part. To build a duration of type xs:dayTimeDuration., see the next signature.

Examples

- **altova:build-duration**(2, 10) returns P2Y10M
- **altova:build-duration**(14, 27) returns P16Y3M
- **altova:build-duration**(2, 24) returns P4Y

altova:build-duration(Days *as xs:integer*, **Hours** *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*) as **xs:dayTimeDuration** XP3 XQ3

Takes four arguments and combines them to build a value of type xs:dayTimeDuration. The first argument provides the Days part of the duration value, the second, third, and fourth arguments provide, respectively, the Hours, Minutes, and Seconds parts of the duration value. Each of the three Time arguments is converted to an equivalent value in terms of the next higher unit and the result is used for calculation of the total duration value. For example, 72 seconds is converted to 1M+12S (1 minute and 12 seconds), and this value is used for calculation of the total duration value. To build a duration of type xs:yearMonthDuration., see the previous signature.

Examples

- **altova:build-duration**(2, 10, 3, 56) returns P2DT10H3M56S

- `altova:build-duration(1, 0, 100, 0)` returns `P1DT1H40M`
- `altova:build-duration(1, 0, 0, 3600)` returns `P1DT1H`

[Top]

Construct date, dateTime, and time datatypes from string input XP2 XQ1 XP3 XQ3

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 XQ3

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 XQ3

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 XQ3
Returns the input string **Time** as an **xs:time** value.The second argument **TimePattern** specifies the pattern (sequence of components) of the input string. **TimePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

H	Hour
m	minutes
s	seconds

The pattern in **TimePattern** must match the pattern in **Time**. Since the output is of type **xs:time**, the output will always have the lexical format **HH:mm:ss**.

Examples

- **altova:parse-time**(xs:string("13:56:24"), "[H]:[m]:[s]") returns 13:56:24
- **altova:parse-time**("13-56-24", "[H]-[m]") returns 13:56:00
- **altova:parse-time**("time=13h56m24s", "time=[H]h[m]m[s]s") returns 13:56:24
- **altova:parse-time**("time=24s56m13h", "time=[s]s[m]m[H]h") returns 13:56:24

[Top]

Age-related functions XP3 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

Returns three integers that are, respectively, the years, months, and days between the two argument dates. The sum of the returned years+months+days together gives the total time difference between the two input dates; it does not matter whether the earlier or later of the two dates is submitted as the first argument. The return values do not indicate whether the input date occurs earlier or later than the current date. Return values are always positive.

Examples

- **altova:age-details**(xs:date("2014-01-16"), xs:date("2014-01-15")) returns (0 0 1)
- **altova:age-details**(xs:date("2014-01-15"), xs:date("2014-01-16")) returns (0 0 1)

[Top]

XPath/XQuery Functions: Geolocation

The following geolocation XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

parse-geolocation [altova:]

altova:parse-geolocation(GeolocationInputString *as xs:string***)** as **xs:decimal+** XP3 XQ3

Parses the supplied `GeolocationInputString` argument and returns the geolocation's latitude and longitude (in that order) as a sequence two `xs:decimal` items. The formats in which the geolocation input string can be supplied are listed below.

Note: The `image-exif-data` function and the Exif metadata's `@Geolocation` attribute can be used to supply the geolocation input string (*see example below*).

Examples

- **altova:parse-geolocation**("33.33 -22.22") returns the sequence of two `xs:decimals` (33.33, 22.22)
- **altova:parse-geolocation**("48°51'29.6""N 24°17'40.2""") returns the sequence of two `xs:decimals` (48.8582222222222, 24.2945)
- **altova:parse-geolocation**('48°51''29.6"N 24°17''40.2"') returns the sequence of two `xs:decimals` (48.8582222222222, 24.2945)
- **altova:parse-geolocation**(**image-exif-data**(//MyImages/Image20141130.01)/**@Geolocation**) returns a sequence of two `xs:decimals`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (**"**) while unit indicators that are escaped are highlighted in blue (**""**).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `W/E`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `W/E`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-distance-km [altova:]

altova:geolocation-distance-km(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) as **xs:decimal** XP3 XQ3

Calculates the distance between two geolocations in kilometers. The formats in which the geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-distance-km**("33.33 -22.22", "48°51'29.6""N 24° 17'40.2""") returns the xs:decimal 4183.08132372392

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:
`33.33N  -22°44'55.25"`
`33.33  22°44'55.25"W`
`33.33  22.45`

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

`geolocation-distance-mi [altova:]`

`altova:geolocation-distance-mi(``GeolocationInputString-1` *as xs:string*`,` `GeolocationInputString-2` *as xs:string*`)` as `xs:decimal` `XP3` `XQ3`
Calculates the distance between two geolocations in miles. The formats in which a geolocation input string can be supplied are listed below. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data` function and the Exif metadata's `@Geolocation` attribute can be used to supply geolocation input strings.

Examples

- **`altova:geolocation-distance-mi(`**`"33.33  -22.22", "48°51'29.6""N  24° 17'40.2"""`**`)`** returns the `xs:decimal` `2599.40652340653`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated

by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (**''**) while unit indicators that are escaped are highlighted in blue (**''''**).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `W/E`)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**N/W**) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `W/E`)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**N/W**) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**N/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation

33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-within-polygon [altova:]

altova:geolocation-within-polygon(Geolocation *as xs:string*, **((PolygonPoint** *as xs:string*)+)) as **xs:boolean** XP3 XQ3

Determines whether **Geolocation** (the first argument) is within the polygonal area described by the **PolygonPoint** arguments. If the PolygonPoint arguments do not form a closed figure (formed when the first point and the last point are the same), then the first point is implicitly added as the last point in order to close the figure. All the arguments (Geolocation and PolygonPoint+) are given by geolocation input strings (*formats listed below*). If the Geolocation argument is within the polygonal area, then the function returns true(); otherwise it returns false(). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-within-polygon(**"33 -22", ("58 -32", "-78 -55", "48 24", "58 -32"))** returns true()
- **altova:geolocation-within-polygon(**"33 -22", ("58 -32", "-78 -55", "48 24"))** returns true()
- **altova:geolocation-within-polygon(**"33 -22", ("58 -32", "-78 -55", "48°51'29.6""N 24°17'40.2""")) returns true()

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for

(**N/W**) is optional
+/-D°M'S.SS" +/-D°M'S.SS"
Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

- Decimal degrees, with suffixed orientation (N/S, W/E)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-within-rectangle [altova:]

altova:geolocation-within-rectangle(Geolocation *as xs:string*, **RectCorner-1** *as xs:string*, **RectCorner-2** *as xs:string***)** as **xs:boolean** XP3 XQ3
Determines whether **Geolocation** (the first argument) is within the rectangle defined by the second and third arguments, **RectCorner-1** and **RectCorner-2**, which specify opposite corners of the rectangle. All the arguments (Geolocation, **RectCorner-1** and **RectCorner-2**) are given by geolocation input strings (*formats listed below*). If the Geolocation argument is within the rectangle, then the function returns true(); otherwise it returns false(). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to

W).

Note: The `image-exif-data` function and the Exif metadata's `@Geolocation` attribute can be used to supply geolocation input strings.

Examples

- **`altova:geolocation-within-rectangle(`**`"33 -22", "58 -32", "-48 24"`**`)`** returns `true()`
- **`altova:geolocation-within-rectangle(`**`"33 -22", "58 -32", "48 24"`**`)`** returns `false()`
- **`altova:geolocation-within-rectangle(`**`"33 -22", "58 -32", "48°51'29.6""S 24°17'40.2"""`**`)`** returns `true()`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `W/E`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `W/E`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional

+/-D.DD +/-D.DD
Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

- *Altova Exif Attribute: Geolocation*

  The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	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]

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
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

`suggested-image-file-extension [altova:]`

altova:suggested-image-file-extension(Base64String *as string***)** as **string?** XP3 XQ3
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 XQ3
Takes a Base64-encoded 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*).

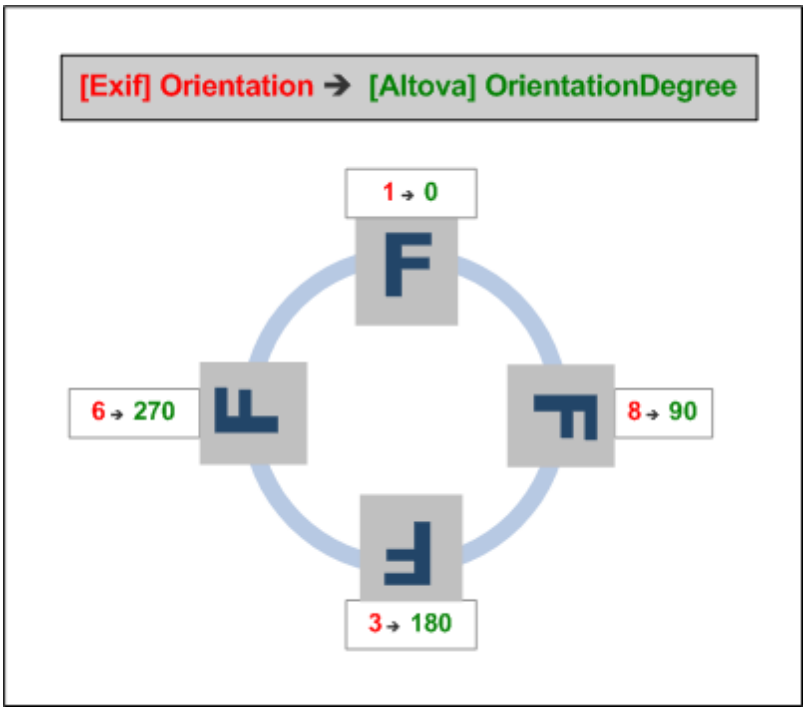
GPSLatitu	GPSLatitude	GPSLongitu	GPSLongitude	Geolocation

de	Ref	de	Ref	
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

- *Altova Exif Attribute: OrientationDegree*

The Altova XPath/XQuery Engine generates the custom attribute **OrientationDegree** from the Exif metadata tag **Orientation**.

OrientationDegree translates the standard Exif tag `Orientation` from an integer value (`1`, `8`, `3`, or `6`) to the respective degree values of each (`0`, `90`, `180`, `270`), as shown in the figure below. Note that there are no translations of the `Orientation` values of `2`, `4`, `5`, `7`. (These orientations are obtained by flipping image `1` across its vertical center axis to get the image with a value of `2`, and then rotating this image in 90-degree jumps clockwise to get the values of `7`, `4`, and `5`, respectively).



- *Listing of standard Exif meta tags*

- `ImageWidth`
- `ImageLength`
- `BitsPerSample`
- `Compression`
- `PhotometricInterpretation`
- `Orientation`
- `SamplesPerPixel`
- `PlanarConfiguration`
- `YCbCrSubSampling`

- YCbCrPositioning
- XResolution
- YResolution
- ResolutionUnit
- StripOffsets
- RowsPerStrip
- StripByteCounts
- JPEGInterchangeFormat
- JPEGInterchangeFormatLength
- TransferFunction
- WhitePoint
- PrimaryChromaticities
- YCbCrCoefficients
- ReferenceBlackWhite
- DateTime
- ImageDescription
- Make
- Model
- Software
- Artist
- Copyright

- ExifVersion
- FlashpixVersion
- ColorSpace
- ComponentsConfiguration
- CompressedBitsPerPixel
- PixelXDimension
- PixelYDimension
- MakerNote
- UserComment
- RelatedSoundFile
- DateTimeOriginal
- DateTimeDigitized
- SubSecTime
- SubSecTimeOriginal
- SubSecTimeDigitized
- ExposureTime
- FNumber
- ExposureProgram
- SpectralSensitivity
- ISOSpeedRatings
- OECF
- ShutterSpeedValue
- ApertureValue
- BrightnessValue
- ExposureBiasValue
- MaxApertureValue
- SubjectDistance
- MeteringMode
- LightSource
- Flash
- FocalLength
- SubjectArea
- FlashEnergy

- SpatialFrequencyResponse
- FocalPlaneXResolution
- FocalPlaneYResolution
- FocalPlaneResolutionUnit
- SubjectLocation
- ExposureIndex
- SensingMethod
- FileSource
- SceneType
- CFAPattern
- CustomRendered
- ExposureMode
- WhiteBalance
- DigitalZoomRatio
- FocalLengthIn35mmFilm
- SceneCaptureType
- GainControl
- Contrast
- Saturation
- Sharpness
- DeviceSettingDescription
- SubjectDistanceRange
- ImageUniqueID

- GPSVersionID
- GPSLatitudeRef
- GPSLatitude
- GPSLongitudeRef
- GPSLongitude
- GPSAltitudeRef
- GPSAltitude
- GPSTimeStamp
- GPSSatellites
- GPSStatus
- GPSMeasureMode
- GPSDOP
- GPSSpeedRef
- GPSSpeed
- GPSTrackRef
- GPSTrack
- GPSImgDirectionRef
- GPSImgDirection
- GPSMapDatum
- GPSDestLatitudeRef
- GPSDestLatitude
- GPSDestLongitudeRef
- GPSDestLongitude
- GPSDestBearingRef
- GPSDestBearing
- GPSDestDistanceRef
- GPSDestDistance
- GPSProcessingMethod
- GPSAreaInformation
- GPSDateStamp
- GPSDifferential

[Top]

XPath/XQuery Functions: Numeric

Altova's numeric extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

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 XQ3

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 XQ3

This function resets the number of the auto-numbering counter named in the `ID` argument. The number is reset to the number specified by the `StartsWith` argument of the `altova:generate-auto-number` function that created the counter named in the `ID` argument.

Examples

- **altova:reset-auto-number**("ChapterNumber") resets the number of the auto-numbering counter named `ChapterNumber` that was created by the `altova:generate-auto-number` function. The number is reset to the value of the `StartsWith` argument of the `altova:generate-auto-number` function that created `ChapterNumber`.

[Top]

Numeric functions

hex-string-to-integer [altova:]

altova:hex-string-to-integer(HexString *as xs:string***)** as **xs:integer** XP3 XQ3

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 XQ3

Takes an integer argument and returns its Base-16 equivalent as a string.

Examples

- **altova:integer-to-hex-string**(1) returns '1'
- **altova:integer-to-hex-string**(9) returns '9'
- **altova:integer-to-hex-string**(10) returns 'A'
- **altova:integer-to-hex-string**(11) returns 'B'
- **altova:integer-to-hex-string**(15) returns 'F'
- **altova:integer-to-hex-string**(16) returns '10'
- **altova:integer-to-hex-string**(32) returns '20'
- **altova:integer-to-hex-string**(33) returns '21'
- **altova:integer-to-hex-string**(90) returns '5A'

[Top]

Number-formatting functions

generate-auto-number [altova:]

altova:generate-auto-number(ID *as xs:string*, **StartsWith** *as xs:double*, **Increment** *as xs:double*, **ResetOnChange** *as xs:string***)** as xs:integer XP1 XP2 XQ1 XP3 XQ3

Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the StartsWith argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the Increment argument. In effect, the altova:generate-auto-number function creates a counter having a name specified by the ID argument, with this counter being incremented each time the function is called. If the value of the ResetOnChange argument changes from that of the previous function call, then the value of the number to be generated is reset to the StartsWith value. Auto-numbering can also be reset by using the altova:reset-auto-number function.

Examples

- **altova:generate-auto-number**("ChapterNumber", 1, 1, "SomeString") will return one number each time the function is called, starting with 1, and incrementing by 1 with each call to the function. As long as the fourth argument remains "SomeString" in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called ChapterNumber) will reset to 1. The value of ChapterNumber can also be reset by a call to the altova:reset-auto-number function, like this: altova:reset-auto-number("ChapterNumber").

[Top]

XPath/XQuery Functions: Sequence

Altova's sequence extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional

functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

attributes [altova:]

altova:attributes(AttributeName *as xs:string***)** as **attribute()*** XP3 XQ3

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 XQ3

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 XQ3

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 XQ3

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The context node must be the parent node of the element/s being searched for. The second argument is a string containing option flags. Available flags are:
r = switches to a regular-expression search; `ElementName` must then be a regular-expression search string;
f = If this option is specified, then `ElementName` provides a full match; otherwise `ElementName` need only partially match an element name to return that element. For example: if **f** is not specified, then `MyElem` will return `MyElement`;
i = switches to a case-insensitive search;
p = includes the namespace prefix in the search; `ElementName` should then contain the namespace prefix, for example: `altova:MyElement`.
The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed.

Examples

- **altova:elements**("MyElement", "rip") returns MyElement()*
- **altova:elements**("MyElement", "pri") returns MyElement()*
- **altova:elements**("MyElement", "") returns MyElement()*
- **altova:attributes**("MyElem", "rip") returns MyElement()*
- **altova:attributes**("MyElements", "rfip") returns no match
- **altova:elements**("MyElement", "Rip") returns an unrecognized-flag error.
- **altova:elements**("MyElement",) returns a missing-second-argument error.

find-first [altova:]

altova:find-first((Sequence *as item()**), **(Condition(Sequence-Item** *as xs:boolean***))** as **item()?** XP3 XQ3

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 XQ3**
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 XQ3**
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 XQ3
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 XQ3

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, `Condition`, is a reference to an XPath function that takes one argument (has an arity of **1**) and returns a boolean. Each item of **Sequence** is submitted, in turn, to the function referenced in `Condition`. (*Remember:* This function takes a single argument.) The first **Sequence** item that causes the function in **Condition** to evaluate to **true()** has its index position in **Sequence** returned as the result of **altova:find-first-pos**, and the iteration stops.

Examples

- **altova:find-first-pos**(5 to 10, function($a) {$a mod 2 = 0}) returns xs:integer 2

The **Condition** argument references the XPath 3.0 inline function, **function()**, which declares an inline function named **$a** and then defines it. Each item in the **Sequence** argument of **altova:find-first-pos** is passed, in turn, to **$a** as its input value. The input value is tested on the condition in the function definition (`$a mod 2 = 0`). The index position in the sequence of the first input value to satisfy this condition is returned as the result of **altova:find-first-pos** (in this case **2**, since **6**, the first value (in the sequence) to satisfy the condition, is at index position **2** in the sequence).

- **altova:find-first-pos**((2 to 10), (function($a) {$a+3=7})) returns xs:integer 3

Further examples

If the file **C:\Temp\Customers.xml** exists:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns 1

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** exists:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns 2

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** also does not exist:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns no result

Notes about the examples given above

- The XPath 3.0 function, `doc-available`, takes a single string argument, which is used as a URI, and returns `true` if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The `doc-available` function can be used for **Condition**, the second argument of `altova:find-first-pos`, because it takes only one argument (arity=1), because it takes an `item()` as input (a string which is used as a URI), and returns a boolean value.
- Notice that the `doc-available` function is only referenced, not called. The `#1` suffix that is attached to it indicates a function with an arity of 1. In its entirety `doc-available#1` simply means: *Use the doc-availabe() function that has arity=1,*

passing to it as its single argument, in turn, each of the items in the first sequence. As a result, each of the two strings will be passed to **doc-available()**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **doc-available()** function evaluates to `true()` and the index position of that string in the sequence is returned as the result of the **altova:find-first-pos** function. *Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

substitute-empty [altova:]

altova:substitute-empty(FirstSequence *as item()**, **SecondSequence** *as item()***)** as **item()*** **XP3** **XQ3**

If `FirstSequence` is empty, returns `SecondSequence`. If `FirstSequence` is not empty, returns `FirstSequence`.

Examples

- **altova:substitute-empty**((1,2,3), (4,5,6)) returns (1,2,3)
- **altova:substitute-empty**((), (4,5,6)) returns (4,5,6)

XPath/XQuery Functions: String

Altova's string extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	**XP1** **XP2** **XP3**
XSLT functions (used in XPath expressions in XSLT):	**XSLT1** **XSLT2** **XSLT3**
XQuery functions (used in XQuery expressions in XQuery):	**XQ1** **XQ3**

camel-case [altova:]

altova:camel-case(InputString *as xs:string***)** as **xs:string** **XP3** **XQ3**

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** **XQ3**

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** **XQ3**

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** **XQ3**

Returns a string containing the character at the position specified by the Position argument, in the string submitted as the InputString argument. The result string will be empty if no character exists at the index submitted by the Position argument.

Examples

- **altova:char**("2014-01-15", 5) returns -
- **altova:char**("USA", 1) returns U

- **altova:char**("USA", 10) returns the empty string.
- **altova:char**("USA", -2) returns the empty string.

first-chars [altova:]

altova:first-chars(X-Number *as xs:integer***)** as **xs:string** XP3 XQ3
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 XQ3
Returns a string containing the first X-Number of characters of the string submitted as the InputString argument.

Examples

- **altova:first-chars**("2014-01-15", 5) returns 2014-
- **altova:first-chars**("USA", 1) returns U

last-chars [altova:]

altova:last-chars(X-Number *as xs:integer***)** as **xs:string** XP3 XQ3
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 XQ3
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 XQ3
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 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

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 XQ3

This function takes an `xs:string` argument, removes any trailing whitespace, and returns a right-trimmed `xs:string`.

Examples

- `altova:trim-string-right("  Hello World  ")` returns `"  Hello World"`
- `altova:trim-string-right("Hello World  ")` returns `"Hello World"`
- `altova:trim-string-right("  Hello World")` returns `"  Hello World"`

- `altova:trim-string-right("Hello World")` returns `"Hello World"`
- `altova:trim-string-right("Hello  World")` returns `"Hello  World"`

XPath/XQuery Functions: Miscellaneous

The following general purpose XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

URI functions

`get-temp-folder [altova:]`

`altova:get-temp-folder()` as `xs:string` XP2 XQ1 XP3 XQ3

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`.

[Top]

Chart Functions

The chart functions listed below enable you to create, generate, and save charts as images. They are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

The chart functions are **XPath functions** (not XSLT functions), and organized into two groups:

- Functions for generating and saving charts
- Functions for creating charts

Note: Chart functions are supported only in **Altova's Server products** and the **Enterprise Editions of Altova products**.

Note: Supported image formats for charts in server editions are `jpg`, `png`, and `bmp`. The best option is png because it is lossless and compressed. In Enterprise editions, the supported formats are `jpg`. `png`, `bmp`, and `gif`.

Functions for generating and saving charts

These functions take the chart object (obtained with the chart creation functions) and either generate an image or save an image to file

`altova:generate-chart-image` `($chart, $width, $height, $encoding) as atomic`

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `binarytobase64` or `binarytobase16`

The function returns the chart image in the specified encoding.

`altova:generate-chart-image` `($chart, $width, $height, $encoding, $imagetype) as atomic`

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `base64Binary` or `hexBinary`
- `$imagetype` may be one of the following image formats: `png`, `gif`, `bmp`, `jpg`, `jpeg`. Note that `gif` is not supported on server products. *Also see note at top of page*.

The function returns the chart image in the specified encoding and image format.

---

altova:save-chart-image ($chart, $filename, $width, $height) as empty() ***(Windows only)***

where

- $chart is the chart extension item obtained with the altova:create-chart function
- $filename is the path to and name of the file to which the chart image is to be saved
- $width and $height must be specified with a length unit

The function saves the chart image to the file specified in $filename.

---

altova:save-chart-image ($chart, $filename, $width, $height, $imagetype) as empty() ***(Windows only)***

where

- $chart is the chart extension item obtained with the altova:create-chart function
- $filename is the path to and name of the file to which the chart image is to be saved
- $width and $height must be specified with a length unit
- $imagetype may be one of the following image formats: png, gif, bmp, jpg, jpeg. Note that gif is not supported on server products. *Also see note at top of page*.

The function saves the chart image to the file specified in $filename in the image format specified.

Functions for creating charts

The following functions are used to create charts.

altova:create-chart($chart-config, $chart-data-series*) as chart extension item

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function

The function returns a chart extension item, which is created from the data supplied via the arguments.

---

altova:create-chart-config($type-name, $title) as chart-config extension item

where

- $type-name specifies the type of chart to be created: Pie, Pie3d, BarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, RoundGauge, BarGauge
- $title is the name of the chart

The function returns a chart-config extension item containing the configuration information of the chart.

---

altova:create-chart-config-from-xml($xml-struct) as chart-config extension item

where

- $xml-struct is the XML structure containing the configuration information of the chart

The function returns a chart-config extension item containing the configuration information of the chart. This information is supplied in an XML data fragment.

---

altova:create-chart-data-series($series-name?, $x-values*, $y-values*) as chart-data-series extension item

where

- $series-name specifies the name of the series
- $x-values gives the list of X-Axis values
- $y-values gives the list of Y-Axis values

The function returns a chart-data-series extension item containing the data for building the chart: that is, the names of the series and the Axes data.

---

altova:create-chart-data-row(x, y1, y2, y3, ...) as chart-data-x-*N*y-row extension item

where

- x is the value of the X-Axis column of the chart data row
- yN are the values of the Y-Axis columns

The function returns a chart-data-x-Ny-row extension item, which contains the data for the X-Axis column and Y-Axis columns of a single series.

---

altova:create-chart-data-series-from-rows(`$series-names as xs:string*, $row*) as chart-data-series extension item`

where

- `$series-name` is the name of the series to be created
- `$row` is the chart-data-x-Ny-row extension item that is to be created as a series

The function returns a chart-data-series extension item, which contains the data for the X-Axis and Y-Axes of the series.

---

altova:create-chart-layer(`$chart-config, $chart-data-series*) as chart-layer extension item`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function

The function returns a chart-layer extension item, which contains chart-layer data.

---

altova:create-multi-layer-chart(`$chart-config, $chart-data-series*, $chart-layer*)`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function
- `$chart-layer` is the chart-layer extension item obtained with the `altova:create-chart-layer` function

The function returns a multi-layer-chart item.

---

altova:create-multi-layer-chart($chart-config, $chart-data-series*, $chart-layer*, xs:boolean $mergecategoryvalues)

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function
- $chart-layer is the chart-layer extension item obtained with the altova:create-chart-layer function

The function returns a multi-layer-chart item.

Chart Data XML Structure

Given below is the XML structure of chart data, how it might appear for the Altova extension functions for charts. This affects the appearance of the specific chart. Not all elements are used for all chart kinds, e.g. the <Pie> element is ignored for bar charts.

Note: Chart functions are supported only in the **Enterprise and Server Editions** of Altova products.

<chart-config>
<General
SettingsVersion="1" *must be provided*
ChartKind="BarChart" *Pie, Pie3d, BarChart, StackedBarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, AreaChart, StackedAreaChart, RoundGauge, BarGauge, CandleStick*
BKColor="#ffffff" *Color*
BKColorGradientEnd="#ffffff" *Color. In case of a gradient, BKColor and BKColorGradientEnd define the gradient's colors*
BKMode="#ffffff" *Solid, HorzGradient, VertGradient*
BKFile="Path+Filename" *String. If file exists, its content is drawn over the background.*
BKFileMode="Stretch" *Stretch, ZoomToFit, Center, Tile*
ShowBorder="1" *Bool*
PlotBorderColor="#000000" *Color*
PlotBKColor="#ffffff" *Color*
Title="" *String*
ShowLegend="1" *Bool*
OutsideMargin="3.%" *PercentOrPixel*
TitleToPlotMargin="3.%" *PercentOrPixel*
LegendToPlotMargin="3.%" *PercentOrPixel*
Orientation="vert" *Enumeration: possible values are: vert, horz*
>
<TitleFont

```
            Color="#000000"   Color
            Name="Tahoma"   String
            Bold="1"   Bool
            Italic="0"   Bool
            Underline="0"   Bool
            MinFontHeight="10.pt"   FontSize (only pt values)
            Size="8.%"   FontSize />
         <LegendFont
            Color="#000000"
            Name="Tahoma"
            Bold="0"
            Italic="0"
            Underline="0"
            MinFontHeight="10.pt"
            Size="3.5%" />
         <AxisLabelFont
            Color="#000000"
            Name="Tahoma"
            Bold="1"
            Italic="0"
            Underline="0"
            MinFontHeight="10.pt"
            Size="5.%" />
         </General>

         <Line
                ConnectionShapeSize="1.%"   PercentOrPixel
                DrawFilledConnectionShapes="1"   Bool
                DrawOutlineConnectionShapes="0"   Bool
                DrawSlashConnectionShapes="0"   Bool
                DrawBackslashConnectionShapes="0"   Bool
         />

         <Bar
                ShowShadow="1"   Bool
                ShadowColor="#a0a0a0"   Color
                OutlineColor="#000000"   Color
                ShowOutline="1"   Bool
         />

         <Area
                Transparency="0"   UINT ( 0-255 ) 255 is fully transparent, 0 is opaque
                OutlineColor="#000000"   Color
                ShowOutline="1"   Bool
         />

         <CandleStick
                FillHighClose="0"   Bool. If 0, the body is left empty. If 1, FillColorHighClose
is used for the candle body
                FillColorHighClose="#ffffff"   Color. For the candle body when close >
open
                FillHighOpenWithSeriesColor="1"   Bool. If true, the series color is used to
fill the candlebody when open > close
                FillColorHighOpen="#000000"   Color. For the candle body when open > close
```

and `FillHighOpenWithSeriesColor` *is false*
/>

<Colors *User-defined color scheme: By default this element is empty except for the style and has no Color attributes*
`UseSubsequentColors ="1"` *Boolean. If* `0`*, then color in overlay is used. If* `1`*, then subsequent colors from previous chart layer is used*
`Style="User"` *Possible values are: "Default", "Grayscale", "Colorful", "Pastel", "User"*
`Colors="#52aca0"` *Color: only added for user defined color set*
`Colors1="#d3c15d"` *Color: only added for user defined color set*
`Colors2="#8971d8"` *Color: only added for user defined color set*
`...`
`ColorsN=""` *Up to ten colors are allowed in a set: from Colors to Colors9*
</Colors>

<Pie
`ShowLabels="1"` *Bool*
`OutlineColor="#404040"` *Color*
`ShowOutline="1"` *Bool*
`StartAngle="0."` *Double*
`Clockwise="1"` *Bool*
`Draw2dHighlights="1"` *Bool*
`Transparency="0"` *Int (0 to 255: 0 is opaque, 255 is fully transparent)*
`DropShadowColor="#c0c0c0"` *Color*
`DropShadowSize="5.%"` *PercentOrPixel*
`PieHeight="10.%"` *PercentOrPixel. Pixel values might be different in the result because of 3d tilting*
`Tilt="40.0"` *Double (10 to 90: The 3d tilt in degrees of a 3d pie)*
`ShowDropShadow="1"` *Bool*
`ChartToLabelMargin="10.%"` *PercentOrPixel*
`AddValueToLabel="0"` *Bool*
`AddPercentToLabel="0"` *Bool*
`AddPercentToLabels_DecimalDigits="0"` *UINT (0 – 2)*
>
<LabelFont
Color="#000000"
Name="Arial"
Bold="0"
Italic="0"
Underline="0"
MinFontHeight="10.pt"
Size="4.%" />
</Pie>

<XY>
<XAxis *Axis*
AutoRange="1" *Bool*
AutoRangeIncludesZero="1" *Bool*
RangeFrom="0." *Double: manual range*
RangeTill="1." *Double : manual range*
LabelToAxisMargin="3.%" *PercentOrPixel*
AxisLabel="" *String*
AxisColor="#000000" *Color*

```
                        AxisGridColor="#e6e6e6"   Color
                        ShowGrid="1"   Bool
                        UseAutoTick="1"   Bool
                        ManualTickInterval="1."   Double
                        AxisToChartMargin="0.px"   PercentOrPixel
                        TickSize="3.px"   PercentOrPixel
                        ShowTicks="1"   Bool
                        ShowValues="1"   Bool
                        AxisPosition="LeftOrBottom"   Enums: "LeftOrBottom",
"RightOrTop", "AtValue"
                        AxisPositionAtValue = "0"   Double
                        >
                        <ValueFont
                               Color="#000000"
                               Name="Tahoma"
                               Bold="0"
                               Italic="0"
                               Underline="0"
                               MinFontHeight="10.pt"
                               Size="3.%"    />
                </XAxis>
                <YAxis   Axis (same as for XAxis)
                        AutoRange="1"
                        AutoRangeIncludesZero="1"
                        RangeFrom="0."
                        RangeTill="1."
                        LabelToAxisMargin="3.%"
                        AxisLabel=""
                        AxisColor="#000000"
                        AxisGridColor="#e6e6e6"
                        ShowGrid="1"
                        UseAutoTick="1"
                        ManualTickInterval="1."
                        AxisToChartMargin="0.px"
                        TickSize="3.px"
                        ShowTicks="1"   Bool
                        ShowValues="1"   Bool
                        AxisPosition="LeftOrBottom"   Enums: "LeftOrBottom",
"RightOrTop", "AtValue"
                        AxisPositionAtValue = "0"   Double
                        >
                        <ValueFont
                               Color="#000000"
                               Name="Tahoma"
                               Bold="0"
                               Italic="0"
                               Underline="0"
                               MinFontHeight="10.pt"
                               Size="3.%"/>
                </YAxis>
        </XY>

        <XY3d
                AxisAutoSize="1"   Bool: If false, XSize and YSize define the aspect ration of
x and y axis. If true, aspect ratio is equal to chart window
```

```
            XSize="100.%"   PercentOrPixel. Pixel values might be different in the result
because of 3d tilting and zooming to fit chart
            YSize="100.%"   PercentOrPixel. Pixel values might be different in the result
because of 3d tilting and zooming to fit chart
            SeriesMargin="30.%"   PercentOrPixel. Pixel values might be different in the
result because of 3d tilting and zooming to fit chart
            Tilt="20."   Double. -90 to +90 degrees
            Rot="20."   Double. -359 to +359 degrees
            FoV="50.">   Double. Field of view. 1-120 degree
            >
            <ZAxis
                  AutoRange="1"
                  AutoRangeIncludesZero="1"
                  RangeFrom="0."
                  RangeTill="1."
                  LabelToAxisMargin="3.%"
                  AxisLabel=""
                  AxisColor="#000000"
                  AxisGridColor="#e6e6e6"
                  ShowGrid="1"
                  UseAutoTick="1"
                  ManualTickInterval="1."
                  AxisToChartMargin="0.px"
                  TickSize="3.px" >
                  <ValueFont
                        Color="#000000"
                        Name="Tahoma"
                        Bold="0"
                        Italic="0"
                        Underline="0"
                        MinFontHeight="10.pt"
                        Size="3.%"/>
            </ZAxis>
      </XY3d>

      <Gauge
            MinVal="0."   Double
            MaxVal="100."   Double
            MinAngle="225"   UINT: -359-359
            SweepAngle="270"   UINT: 1-359
            BorderToTick="1.%"   PercentOrPixel
            MajorTickWidth="3.px"   PercentOrPixel
            MajorTickLength="4.%"   PercentOrPixel
            MinorTickWidth="1.px"   PercentOrPixel
            MinorTickLength="3.%"   PercentOrPixel
            BorderColor="#a0a0a0"   Color
            FillColor="#303535"   Color
            MajorTickColor="#a0c0b0"   Color
            MinorTickColor="#a0c0b0"   Color
            BorderWidth="2.%"   PercentOrPixel
            NeedleBaseWidth="1.5%"   PercentOrPixel
            NeedleBaseRadius="5.%"   PercentOrPixel
            NeedleColor="#f00000"   Color
            NeedleBaseColor="#141414"   Color
            TickToTickValueMargin="5.%"   PercentOrPixel
```

```
                MajorTickStep="10."   Double
                MinorTickStep="5."   Double
                RoundGaugeBorderToColorRange="0.%"   PercentOrPixel
                RoundGaugeColorRangeWidth ="6.%"   PercentOrPixel
                BarGaugeRadius="5.%"   PercentOrPixel
                BarGaugeMaxHeight="20.%"   PercentOrPixel
                RoundGaugeNeedleLength="45.%"   PercentOrPixel
                BarGaugeNeedleLength="3.%"   PercentOrPixel
                >
                <TicksFont
                        Color="#a0c0b0"
                        Name="Tahoma"
                        Bold="0"
                        Italic="0"
                        Underline="0"
                        MinFontHeight="10.pt"
                        Size="4.%"
                />
                <ColorRanges>   User-defined color ranges. By default empty with no child
element entries
                        <Entry
                                From="50. "   Double
                                FillWithColor="1"   Bool
                                Color="#00ff00"   Color
                        />
                        <Entry
                                From="50.0"
                                FillWithColor="1"
                                Color="#ff0000"
                        />
                        ...
                </ColorRanges>
        </Gauge>
</chart-config>
```

Example: Chart Functions

The example XSLT document below shows how Altova extension functions for charts can be used. Given further below are an XML document and a screenshot of the output image generated when the XML document is processed with the XSLT 2.0 Engine.

Note: Chart functions are supported only in the **Enterprise and Server Editions** of Altova products.

Note: For more information about how chart data tables are created, see the documentation of Altova's XMLSpy and StyleVision products.

XSLT document

This XSLT document (*listing below*) uses Altova chart extension functions to generate a pie chart. It can be used to process the XML document listed further below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:altovaext="http://www.altova.com/xslt-extensions"
    exclude-result-prefixes="#all">
    <xsl:output version="4.0" method="html" indent="yes" encoding="UTF-8"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>
                    <xsl:text>HTML Page with Embedded Chart</xsl:text>
                </title>
            </head>
            <body>
                <xsl:for-each select="/Data/Region[1]">
                    <xsl:variable name="extChartConfig" as="item()*">
                        <xsl:variable name="ext-chart-settings" as="item()*">
                            <chart-config>
                                <General
                                    SettingsVersion="1"
                                    ChartKind="Pie3d"
                                    BKColor="#ffffff"
                                    ShowBorder="1"
                                    PlotBorderColor="#000000"
                                    PlotBKColor="#ffffff"
                                    Title="{@id}"
                                    ShowLegend="1"
                                    OutsideMargin="3.2%"
                                    TitleToPlotMargin="3.%"
                                    LegendToPlotMargin="6.%"
                                    >
                                    <TitleFont
                                        Color="#023d7d"
                                        Name="Tahoma"
                                        Bold="1"
                                        Italic="0"
                                        Underline="0"
                                        MinFontHeight="10.pt"
                                        Size="8.%" />
                                </General>
                            </chart-config>
                        </xsl:variable>
                        <xsl:sequence select="altovaext:create-chart-config-from-
xml( $ext-chart-settings )"/>
                    </xsl:variable>
                    <xsl:variable name="chartDataSeries" as="item()*">
                        <xsl:variable name="chartDataRows" as="item()*">
                            <xsl:for-each select="(Year)">
                                <xsl:sequence select="altovaext:create-chart-data-
```

```
row( (@id), ( .) )"/>
                        </xsl:for-each>
                    </xsl:variable>
                    <xsl:variable name="chartDataSeriesNames" as="xs:string*"
select=" ( (&quot;Series 1&quot;), &apos;&apos; )[1]"/>
                    <xsl:sequence
                        select="altovaext:create-chart-data-series-from-
rows( $chartDataSeriesNames, $chartDataRows)"/>
                </xsl:variable>
                <xsl:variable name="ChartObj" select="altovaext:create-
chart( $extChartConfig, ( $chartDataSeries), false() )"/>
                <xsl:variable name="sChartFileName" select="'mychart1.png'"/>
                <img src="{$sChartFileName, altovaext:save-chart-
image( $ChartObj, $sChartFileName, 400, 400 ) }"/>
            </xsl:for-each>
        </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

XML document

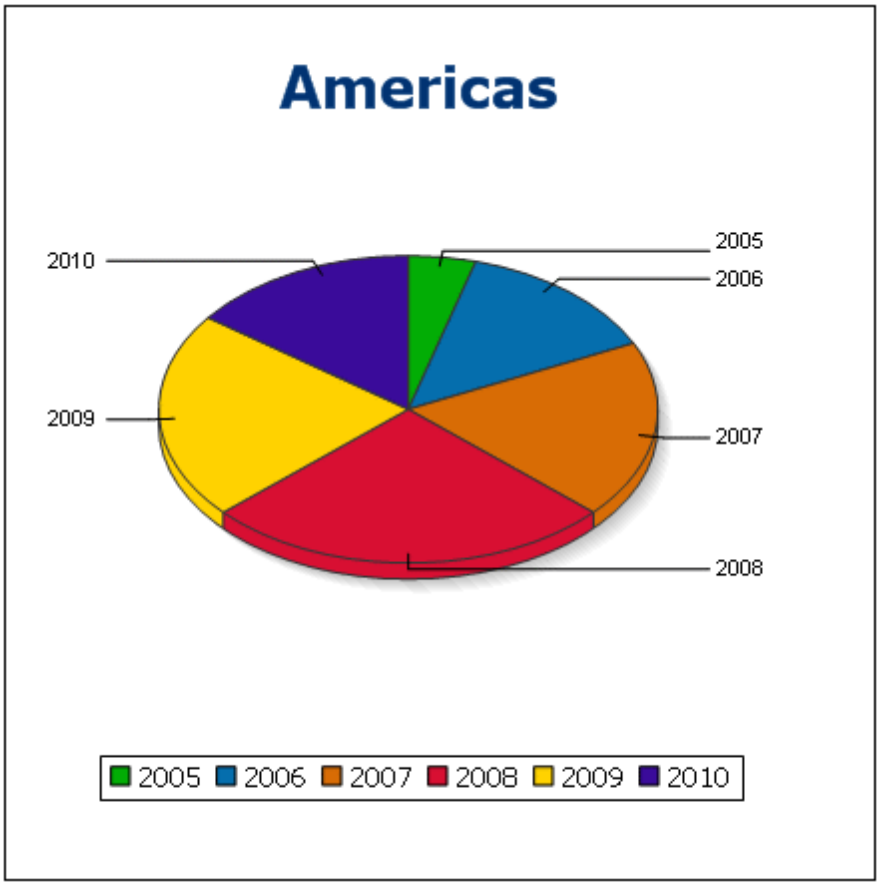
This XML document can be processed with the XSLT document above. Data in the XML document is used to generate the pie chart shown in the screenshot below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="YearlySales.xsd">
   <ChartType>Pie Chart 2D</ChartType>
   <Region id="Americas">
      <Year id="2005">30000</Year>
      <Year id="2006">90000</Year>
      <Year id="2007">120000</Year>
      <Year id="2008">180000</Year>
      <Year id="2009">140000</Year>
      <Year id="2010">100000</Year>
   </Region>
   <Region id="Europe">
      <Year id="2005">50000</Year>
      <Year id="2006">60000</Year>
      <Year id="2007">80000</Year>
      <Year id="2008">100000</Year>
      <Year id="2009">95000</Year>
      <Year id="2010">80000</Year>
   </Region>
   <Region id="Asia">
      <Year id="2005">10000</Year>
      <Year id="2006">25000</Year>
      <Year id="2007">70000</Year>
      <Year id="2008">110000</Year>
      <Year id="2009">125000</Year>
```

```
        <Year id="2010">150000</Year>
    </Region>
</Data>
```

Output image

The pie chart show below is generated when the XML document listed above is processed with the XSLT document.



19.1.2.2 Miscellaneous Extension Functions

There are several ready-made functions in programming languages such as Java and C# that are not available as XQuery/XPath functions or as XSLT functions. A good example would be the math functions available in Java, such as `sin()` and `cos()`. If these functions were available to the designers of XSLT stylesheets and XQuery queries, it would increase the application area of stylesheets and queries and greatly simplify the tasks of stylesheet creators. The XSLT and XQuery engines used in a number of Altova products support the use of extension functions in Java and .NET, as well as MSXSL scripts for XSLT. This section describes how to use extension functions and MSXSL scripts in your XSLT stylesheets and XQuery documents. The available extension functions are organized into the following sections:

- Java Extension Functions
- .NET Extension Functions
- MSXSL Scripts for XSLT

The two main issues considered in the descriptions are: (i) how functions in the respective libraries are called; and (ii) what rules are followed for converting arguments in a function call to the required input format of the function, and what rules are followed for the return conversion (function result to XSLT/XQuery data object).

Requirements

For extension functions support, a Java Runtime Environment (for access to Java functions) and .NET Framework 2.0 (minimum, for access to .NET functions) must be installed on the machine running the XSLT transformation or XQuery execution, or must be accessible for the transformations.

Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- Java: Constructors
- Java: Static Methods and Static Fields
- Java: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to Java
- Datatypes: Java to XPath/XQuery

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with `java:` (*see below for examples*). The namespace declaration should identify a Java class, for example: `xmlns:myns="java:java.lang.Math"`. However, it could also simply be: `xmlns:myns="java"` (without a colon), with the identification of the Java class being left to the `fname()` part of the extension function.
- The `fname()` part identifies the Java method being called, and supplies the arguments for the method (*see below for examples*). However, if the namespace URI identified by the `prefix:` part does not identify a Java class (*see preceding point*), then the Java class should be identified in the `fname()` part, before the class and separated from the class by

a period (*see the second XSLT example below*).

Note: The class being called must be on the classpath of the machine.

XSLT example

Here are two examples of how a static method can be called. In the first example, the class name (`java.lang.Math`) is included in the namespace URI and, therefore, must not be in the `fname()` part. In the second example, the `prefix:` part supplies the prefix `java:` while the `fname()` part identifies the class as well as the method.

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
              select="jmath:java.lang.Math.cos(3.14)" />
```

The method named in the extension function (`cos()` in the example above) must match the name of a public static method in the named Java class (`java.lang.Math` in the example above).

XQuery example

Here is an XQuery example similar to the XSLT example above:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

User-defined Java classes

If you have created your own Java classes, methods in these classes are called differently according to: (i) whether the classes are accessed via a JAR file or a class file, and (ii) whether these files (JAR or class) are located in the current directory (the same directory as the XSLT or XQuery document) or not. How to locate these files is described in the sections User-Defined Class Files and User-Defined Jar Files. Note that paths to class files not in the current directory and to all JAR files must be specified.

User-Defined Class Files

If access is via a class file, then there are two possibilities:

- The class file is in a package. The XSLT or XQuery file is in the same folder as the Java package. (*See example below.*)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class

file. (See example below.)

- The class file is in a package. The XSLT or XQuery file is at some random location. (See example below.)
- The class file is not packaged. The XSLT or XQuery file is at some random location. (See example below.)

Consider the case where the class file is not packaged and is in the same folder as the XSLT or XQuery document. In this case, since all classes in the folder are found, the file location does not need to be specified. The syntax to identify a class is:

```
java:classname
```

where

`java`: indicates that a user-defined Java function is being called; (Java classes in the current directory will be loaded by default)

`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call.

Class file packaged, XSLT/XQuery file in same folder as Java package

The example below calls the `getVehicleType()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is also in the folder `JavaProject`.

```
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/2005/xpath-functions"
      xmlns:car="java:com.altova.extfunc.Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file not packaged, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()`method of the `Car` class of the `com.altova.extfunc` package. The `Car` class file is in the following folder location: `JavaProject/com/altova/extfunc`. The XSLT file is also in the folder `JavaProject/com/altova/extfunc`.


```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

where

`java:` indicates that a user-defined Java function is being called
`uri-of-package` is the URI of the Java package
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:com.altova.extfunc.Car?path=file:///C:/
JavaProject/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

Class file not packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. The location of the class file is specified within the namespace URI as a query string. The syntax is:

```
java:classname[?path=uri-of-classfile]
```

where

`java`: indicates that a user-defined Java function is being called
`uri-of-classfile` is the URI of the folder containing the class file
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:Car?path=file:///C:/JavaProject/com/altova/
extfunc/" >

       <xsl:output exclude-result-prefixes="fn car xsl xs"/>

       <xsl:template match="/">
          <xsl:variable name="myCar" select="car:new('red')" />
          <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
       </xsl:template>

       </xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

The method is then called by using the prefix of the namespace URI that identifies the class: `classNS:method()`

In the above:

`java:` indicates that a Java function is being called
`classname` is the name of the user-defined class
`?` is the separator between the classname and the path
`path=jar:` indicates that a path to a JAR file is being given
`uri-of-jarfile` is the URI of the jar file
`!/` is the end delimiter of the path
`classNS:method()` is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```
              xmlns:ns1="java:docx.layout.pages?path=jar:file:///c:/projects/
docs/docx.jar!/"
              ns1:main()

              xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
              ns2:docx.layout.pages.main()
```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
       <xsl:variable name="myCar" select="car:Car1.new('red')" />
       <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be implicitly converted to XPath/XQuery datatypes, then the Java extension function will return a sequence that is an XPath/XQuery datatype. If the result of a Java constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped Java object with a type that is the name of the class returning that Java object. For example, if a constructor for the class `java.util.Date` is called

(`java.util.Date.new()`), then an object having a type `java.util.Date` is returned. The lexical format of the returned object may not match the lexical format of an XPath datatype and the value would therefore need to be converted to the lexical format of the required XPath datatype and then to the required XPath datatype.

There are two things that can be done with a Java object created by a constructor:

- It can be assigned to a variable:
```
<xsl:variable name="currentdate" select="date:new()"
xmlns:date="java:java.util.Date" />
```
- It can be passed to an extension function (*see Instance Method and Instance Fields*):
```
<xsl:value-of select="date:toString(date:new())"
xmlns:date="java:java.util.Date" />
```

Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples

Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`).) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery example

A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()"/>
  <xsl:template match="/">
      <enrollment institution-id="Altova School"
                date="{date:toString($CurrentDate)}"
                type="
{jlang:Object.toString(jlang:Object.getClass( date:new() ))}">
      </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the parameter `CurrentDate` takes the value returned by a constructor for the class `java.util.Date`. This value is then passed as an argument to the instance method `date:toString` in order to supply the value of `/enrollment/@date`.

Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (primitive), `java.lang.Boolean`
`xs:integer`	int, long, short, byte, float, double, and the wrapper classes of these, such as `java.lang.Integer`
`xs:float`	`float` (primitive), `java.lang.Float`, `double` (primitive)
`xs:double`	`double` (primitive), `java.lang.Double`
`xs:decimal`	`float` (primitive), `java.lang.Float`, `double`(primitive), `java.lang.Double`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

Datatypes: Java to XPath/XQuery

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.

One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

.NET Extension Functions

If you are working on the .NET platform on a Windows machine, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

- .NET: Constructors
- .NET: Static Methods and Static Fields
- .NET: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to .NET
- Datatypes: .NET to XPath/XQuery

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

Parameters

To load an assembly, the following parameters are used:

`asm`	The name of the assembly to be loaded.
`ver`	The version number (maximum of four integers separated by periods).
`sn`	The key token of the assembly's strong name (16 hex digits).
`from`	A URI that gives the location of the assembly (DLL) to be loaded. If the URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.
`partialname`	The partial name of the assembly. It is supplied to `Assembly.LoadWith.PartialName()`, which will attempt to load the assembly. If `partialname` is present, any other parameter is ignored.
`loc`	The locale, for example, `en-US`. The default is `neutral`.

If the assembly is to be loaded from a DLL, use the `from` parameter and omit the `sn` parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the `sn` parameter and omit the `from` parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (*see example below*).

Examples of namespace declarations

An example of a namespace declaration in XSLT that identifies the system class `System.Environment`:

```
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as `Trade.Forward.Scrip`:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class `MyManagedDLL.testClass`:. Two cases are distinguished:

1. When the assembly is loaded from the GAC:

```
declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
       ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. When the assembly is loaded from the DLL (complete and partial references below):

```
       declare namespace cs="clitype:MyManagedDLL.testClass?from=file:///
C:/Altova
       Projects/extFunctions/MyManagedDLL.dll;

   declare namespace cs="clitype:MyManagedDLL.testClass?
   from=MyManagedDLL.dll;
```


XSLT example

Here is a complete XSLT example that calls functions in system class `System.Math`:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes" />
   <xsl:template match="/">
      <math xmlns:math="clitype:System.Math">
         <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
         <pi><xsl:value-of select="math:PI()"/></pi>
         <e><xsl:value-of select="math:E()"/></e>
         <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
      </math>
   </xsl:template>
</xsl:stylesheet>
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`. The extension functions identify methods in the class `System.Math` and supply arguments where required.

XQuery example

Here is an XQuery example fragment similar to the XSLT example above:

```
<math xmlns:math="clitype:System.Math">
   {math:Sqrt(9)}
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this case a system class. The XQuery expression identifies the method to be called and supplies the argument.

.NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a `'No constructor found'` error is returned.

Constructors that return XPath/XQuery datatypes

If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

Constructors that return .NET objects

If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:

- It can be used within a variable:
  ```
  <xsl:variable name="currentdate" select="date:new(2008, 4, 29)"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be passed to an extension function (*see Instance Method and Instance Fields*):
  ```
  <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be converted to a string, number, or boolean:
- ```
  <xsl:value-of select="xs:integer(data:get_Month(date:new(2008, 4, 29)))"
  xmlns:date="clitype:System.DateTime" />
  ```

.NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```
```

An XSLT example showing a call to a field (considered a method with no argument) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

An XSLT example showing a call to a property (syntax is `get_PropertyName()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length('my string')"
xmlns:string="clitype:System.String"/>
```

An XQuery example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

.NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:variable name="releasedate"
         select="date:new(2008, 4, 29)"
         xmlns:date="clitype:System.DateTime"/>
      <doc>
         <date>
            <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
               xmlns:date="clitype:System.DateTime"/>
         </date>
         <date>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008, 4, 29)`) as its first

and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

`xs:string`	`StringValue, string`
`xs:boolean`	`BooleanValue, bool`
`xs:integer`	`IntegerValue, decimal, long, integer, short, byte, double, float`
`xs:float`	`FloatValue, float, double`
`xs:double`	`DoubleValue, double`
`xs:decimal`	`DecimalValue, decimal, double, float`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`.

When a .NET object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example `System.DateTime.ToString()`) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (*see example below*).

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (*see below*).

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-
prefix">

   function-1 or variable-1
   ...
   function-n or variable-n

</msxsl:script>
```

The <msxsl:script> element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the <msxsl:script> element. **The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.** Consequently, the .NET scripting languages can be used within the <msxsl:script> element.

The language attribute accepts the same values as the language attribute on the HTML <script> element. If the language attribute is not specified, then Microsoft JScript is assumed as the default.

The implements-prefix attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the <msxsl:script> element will be in the namespace identified by the prefix specified in the implements-prefix attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

Example

Here is an example of a complete XSLT stylesheet that uses a function defined within a <msxsl:script> element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
    ' Input: A currency value: the wholesale price
    ' Returns: The retail price: the input value plus 20% margin,
    ' rounded to the nearest cent
    dim a as integer  = 13
    Function AddMargin(WholesalePrice) as integer
      AddMargin = WholesalePrice * 1.2 + a
    End Function
  ]]>
  </msxsl:script>

  <xsl:template match="/">
    <html>
```



```
      <body>
        <p>
          <b>Total Retail Price =
            $<xsl:value-of select="user:AddMargin(50)"/>
          </b>
          <br/>
          <b>Total Wholesale Price =
            $<xsl:value-of select="50"/>
          </b>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies

An assembly can be imported into the script by using the **msxsl:assembly** element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl:assembly` element is to be used.

```
<msxsl:script>
       <msxsl:assembly name="myAssembly.assemblyName" />
       <msxsl:assembly href="pathToAssembly" />

       ...

</msxsl:script>
```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral
PublicKeyToken=a46b3f648229c514"
```

or a short name, such as `"myAssembly.Draw"`.

Namespaces

Namespaces can be declared with the **msxsl:using** element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl:using` element is used so as to declare namespaces.

```
<msxsl:script>
       <msxsl:using namespace="myAssemblyNS.NamespaceName" />

       ...

</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.

19.2 Technical Data

This section contains useful background information on the technical aspects of your software. It is organized into the following sections:

19.2.1 OS and Memory Requirements

Operating System
Altova software applications are available for the following platforms:

- 32-bit Windows applications for Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10, Windows Server 2003/2008
- 64-bit Windows applications for Windows Vista, Windows 7, Windows 8, Windows 10, Windows Server 2012

Memory
Since the software is written in C++ it does not require the overhead of a Java Runtime Environment and typically requires less memory than comparable Java-based applications. However, each document is loaded fully into memory so as to parse it completely and to improve viewing and editing speed. The memory requirement increases with the size of the document.

Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, available memory can rapidly be depleted.

19.2.2 Altova XML Validator

When opening any XML document, the application uses its built-in XML validator to check for well-formedness, validate the document against a schema (if specified), and build trees and infosets. The XML validator is also used to provide intelligent editing help while you edit documents and to dynamically display any validation error that may occur.

The built-in XML validator implements the Final Recommendation of the W3C's XML Schema 1.0 and 1.1 specification. New developments recommended by the W3C's XML Schema Working Group are continuously being incorporated in the XML validator, so that Altova products give you a state-of-the-art development environment.

19.2.3 Altova XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0, 2.0, and 3.0 Engines and the Altova XQuery 1.0 and 3.1 Engines. Documentation about implementation-specific behavior for each engine is in the

appendices of the documentation (Engine Information), should that engine be used in the product.

Note: Altova MapForce generates code using the XSLT 1.0, 2.0 and XQuery 1.0 engines.

19.2.4 Unicode Support

Altova's XML products provide full Unicode support. To edit an XML document, you will also need a font that supports the Unicode characters being used by that document.

Please note that most fonts only contain a very specific subset of the entire Unicode range and are therefore typically targeted at the corresponding writing system. If some text appears garbled, the reason could be that the font you have selected does not contain the required glyphs. So it is useful to have a font that covers the entire Unicode range, especially when editing XML documents in different languages or writing systems. A typical Unicode font found on Windows PCs is Arial Unicode MS.

In the `/Examples` folder of your application folder you will find an XHTML file called `UnicodeUTF-8.html` that contains the following sentence in a number of different languages and writing systems:

- *When the world wants to talk, it speaks Unicode*
- *Wenn die Welt miteinander spricht, spricht sie Unicode*
- 世界的に話すなら、Unicode です。)

Opening this XHTML file will give you a quick impression of Unicode's possibilities and also indicate what writing systems are supported by the fonts available on your PC.

19.2.5 Internet Usage

Altova applications will initiate Internet connections on your behalf in the following situations:

- If you click the "Request evaluation key-code" in the Registration dialog (**Help | Software Activation**), the three fields in the registration dialog box are transferred to our web server by means of a regular http (port 80) connection and the free evaluation key-code is sent back to the customer via regular SMTP e-mail.
- In some Altova products, you can open a file over the Internet (**File | Open | Switch to URL**). In this case, the document is retrieved using one of the following protocol methods and connections: HTTP (normally port 80), FTP (normally port 20/21), HTTPS (normally port 443). You could also run an HTTP server on port 8080. (In the URL dialog, specify the port after the server name and a colon.)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, the referenced schema document is also retrieved through a HTTP connection (port 80) or another protocol specified in the URL (see Point 2 above). A schema document will also be retrieved when an XML file is validated. Note that validation might happen automatically upon opening a document if you have instructed the application to do this (in the File tab of the Options dialog (**Tools | Options**)).
- In Altova applications using WSDL and SOAP, web service connections are defined by the WSDL documents.
- If you are using the **Send by Mail** command (**File | Send by Mail**) in XMLSpy, the

current selection or file is sent by means of any MAPI-compliant mail program installed on the user's PC.

- As part of Software Activation and LiveUpdate as further described in the Altova Software License Agreement.

19.3 License Information

This section contains:

- Information about the distribution of this software product
- Information about software activation and license metering
- Information about the intellectual property rights related to this software product
- The End-User License Agreement governing the use of this software product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

19.3.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge before making a purchasing decision.
- Once you decide to buy the software, you can place your order online at the Altova website and immediately get a fully licensed product within minutes.
- When you place an online order, you always get the latest version of our software.
- The product package includes a comprehensive integrated onscreen help system. The latest version of the user manual is available at www.altova.com (i) in HTML format for online browsing, and (ii) in PDF format for download (and to print if you prefer to have the documentation on paper).

30-day evaluation period
After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into this evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you have to purchase an Altova Software License Agreement, which is delivered in the form of a key-code that you enter into the Software Activation dialog to unlock the product. You can purchase your license at the online shop at the Altova website.

Helping Others within Your Organization to Evaluate the Software
If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may only distribute the Setup programs, provided that they are not modified in any way. Any person that accesses the software installer that you have provided, must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.

For further details, please refer to the Altova Software License Agreement at the end of this section.

19.3.2 Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license
When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi license
If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (*see* the IANA website (http://www.iana.org/) *for details*) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

You will also notice that, if you are online, your Altova product contains many useful functions; these are unrelated to the license-metering technology.

19.3.3 Intellectual Property Rights

The Altova Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

Altova software contains certain Third Party Software that is also protected by intellectual property laws, including without limitation applicable copyright laws as described in detail at http://www.altova.com/legal_3rdparty.html.

All other names or trademarks are the property of their respective owners.

19.3.4 Altova End User License Agreement

THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS

ALTOVA® END USER LICENSE AGREEMENT

Licensor:
Altova GmbH
Rudolfsplatz 13a/9
A-1010 Wien
Austria

Important - Read Carefully. Notice to User:

This End User License Agreement ("Agreement") is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software ("Software") and any accompanying documentation, including, without limitation printed materials, 'online' files, or electronic documentation ("Documentation"). By clicking the "I accept" and "Next" buttons below, or by installing, or otherwise using the Software, you agree to be bound by the terms of this Agreement as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below, whether or not you decide to purchase the Software. You agree that this agreement is enforceable like any written agreement negotiated and signed by you. If you do not agree, you are not licensed to use the Software, and you must destroy any downloaded copies of the

Software in your possession or control. You may print a copy of this Agreement as part of the installation process at the time of acceptance. Alternatively, a copy of this Agreement may be found at http://www.altova.com/eula and a copy of the Privacy Policy may be found at http://www.altova.com/privacy.

1. SOFTWARE LICENSE

(a) License Grant.

(i) Upon your acceptance of this Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license, without the right to grant sublicenses, to install and use a copy of the Software on one compatible personal computer or workstation in the same local area network (LAN) up to the Permitted Number of computers. Subject to the limitations set forth in Section 1(c), you may install and use a copy of the Software on more than one of your compatible personal computers or workstations if you have purchased a Named-User license. Subject to the limitations set forth in Sections 1(d) and 1(e), users may use the software concurrently on a network. The Permitted Number of computers and/or users and the type of license, e.g. Installed, Named-Users, and Concurrent-User, shall be determined and specified at such time as you elect to purchase the Software. Installed user licenses are intended to be fixed and not concurrent. In other words, you cannot uninstall the Software on one machine in order to reinstall that license to a different machine and then uninstall and reinstall back to the original machine. Installations should be static. Notwithstanding the foregoing, permanent uninstallations and redeployments are acceptable in limited circumstances such as if an employee leaves the company or the machine is permanently decommissioned. During the evaluation period, hereinafter defined, only a single user may install and use the software on one (1) personal computer or workstation. If you have licensed the Software as part of a suite of Altova software products (collectively, the "Suite") and have not installed each product individually, then the Agreement governs your use of all of the software included in the Suite.

(ii) If you have licensed SchemaAgent, then the terms and conditions of this Agreement apply to your use of the SchemaAgent server software ("SchemaAgent Server") included therein, as applicable, and you are licensed to use SchemaAgent Server solely in connection with your use of Altova Software and solely for the purposes described in the accompanying documentation.

(iii) If you have licensed Software that enables users to generate source code, your license to install and use a copy of the Software as provided herein permits you to generate source code based on (i) Altova Library modules that are included in the Software (such generated code hereinafter referred to as the "Restricted Source Code") and (ii) schemas or mappings that you create or provide (such code as may be generated from your schema or mapping source materials hereinafter referred to as the "Unrestricted Source Code"). In addition to the rights granted herein, Altova grants you a non-exclusive, non-transferable, limited license to compile the complete generated code (comprised of the combination of the Restricted Source Code and the Unrestricted Source Code) into executable object code form, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer the Restricted Source Code to a third-party in the un-compiled form unless said third-party already has a license to the Restricted Source Code through their separate agreement with Altova. Notwithstanding anything to the contrary herein, you may not distribute, incorporate or combine with other software, or otherwise use the Altova Library modules or Restricted Source Code, or any Altova intellectual property embodied in or associated with the Altova Library modules or Restricted Source Code, in any manner that would subject the Restricted Source Code to the terms of a copyleft, free software or open source license that would require the Restricted Source Code or Altova Library modules source code to be disclosed in source code form. Notwithstanding anything to the contrary herein, you may not use the Software to develop and distribute other

software programs that directly compete with any Altova software or service without prior written permission. Altova reserves all other rights in and to the Software. With respect to the feature(s) of UModel that permit reverse-engineering of your own source code or other source code that you have lawfully obtained, such use by you does not constitute a violation of this Agreement. Except as otherwise expressly permitted in Section 1(j) reverse engineering of the Software is strictly prohibited as further detailed therein.

(iv) In the event Restricted Source Code is incorporated into executable object code form, you will include the following statement in (1) introductory splash screens, or if none, within one or more screens readily accessible by the end-user, and (2) in the electronic and/or hard copy documentation: "Portions of this program were developed using Altova® [name of Altova Software, e.g. MapForce® 2016] and includes libraries owned by Altova GmbH, Copyright © 2007-2016 Altova GmbH (www.altova.com)."

(b) Server Use for Installation and Use of SchemaAgent. You may install one (1) copy of the Software on a computer file server within your internal network solely for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers in a commercial environment only. If you have licensed SchemaAgent, then you may install SchemaAgent Server on any server computer or workstation and use it in connection with your Software. No other network use is permitted, including without limitation using the Software either directly or through commands, data or instructions from or to a computer not part of your internal network, for Internet or Web-hosting services or by any user not licensed to use this copy of the Software through a valid license from Altova.

(c) Named-Use. If you have licensed the "Named-User" version of the software, you may install the Software on up to five (5) compatible personal computers or workstations of which you are the primary user thereby allowing you to switch from one computer to the other as necessary provided that only one (1) instance of the Software will be used by you as the Named-User at any given time. If you have purchased multiple Named-User licenses, each individual Named-User will receive a separate license key code.

(d) Concurrent Use in Same Local Area Network (LAN). If you have licensed a "Concurrent-User" version of the Software, you may install the Software on any compatible computers in a commercial environment only, up to ten (10) times the Permitted Number of users, provided that only the Permitted Number of users actually use the Software at the same time and further provided that the computers on which the Software is installed are on the same local area network (LAN). The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses. Each separate local area network (LAN) requires its own set of separate Concurrent User Licenses for those wishing to use the Concurrent User versions of the Software in more than one location or on more than one network, all subject to the above Permitted Number limitations and based on the number of users using the Software. If a computer is not on the same local area network (LAN), then a locally installed user license or a license dedicated to concurrent use in a virtual environment is required.

(e) Concurrent Use in Virtual Environment. If you have purchased Concurrent-User Licenses, you may install a copy of the Software on a single host terminal server (Microsoft Terminal Server or Citrix Metaframe), application virtualization server (Microsoft App-V, Citrix XenApp, or VMWare ThinApp) or virtual machine environment within your internal network for the sole and exclusive purpose of permitting individual users within your organization to access and use the Software through a terminal server, application virtualization session, or virtual machine environment from another computer provided that the total number of users that access or use the Software concurrently at any given point in time on such network, virtual machine or terminal server does not exceed the Permitted Number; and provided that the total number of users

authorized to use the Software through the terminal server, application virtualization session, or virtual machine environment does not exceed ten (10) times the Permitted Number of users. Key codes for concurrent users cannot be deployed to more than one host terminal server, application virtualization server or virtual machine environment. You must deploy a reliable and accurate means of preventing users from exceeding the Permitted Number of concurrent users. Altova makes no warranties or representations about the performance of Altova software in a terminal server, application virtualization session, or virtual machine environment and the foregoing are expressly excluded from the limited warranty in Section 5 hereof. Technical support is not available with respect to issues arising from use in such environments.

(f) Backup and Archival Copies. You may make one (1) backup and one (1) archival copy of the Software, provided your backup and archival copies are not installed or used on any computer and further provided that all such copies shall bear the original and unmodified copyright, patent and other intellectual property markings that appear on or in the Software. You may not transfer the rights to a backup or archival copy unless you transfer all rights in the Software as provided under Section 3.

(g) Key Codes, Upgrades and Updates. Prior to your purchase and as part of the registration for the thirty (30) day evaluation period, as applicable, you will receive an evaluation key code. You will receive a purchase key code when you elect to purchase the Software from either Altova GmbH or an authorized reseller. The purchase key code will enable you to activate the Software beyond the initial evaluation period. You may not re-license, reproduce or distribute any key code except with the express written permission of Altova. If the Software that you have licensed is an upgrade or an update, then the latest update or upgrade that you download and install replaces all or part of the Software previously licensed. The update or upgrade and the associated license keys does not constitute the granting of a second license to the Software in that you may not use the upgrade or updated copy in addition to the copy of the Software that it is replacing and whose license has terminated.

(h) Title. Title to the Software is not transferred to you. Ownership of all copies of the Software and of copies made by you is vested in Altova, subject to the rights of use granted to you in this Agreement. As between you and Altova, documents, files, stylesheets, generated program code (including the Unrestricted Source Code) and schemas that are authored or created by you via your utilization of the Software, in accordance with its Documentation and the terms of this Agreement, are your property unless they are created using Evaluation Software, as defined in Section 4 of this Agreement, in which case you have only a limited license to use any output that contains generated program code (including Unrestricted Source Code) such as Java, C++, C#, VB.NET or XSLT and associated project files and build scripts, as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures only for the thirty (30) day evaluation period.

(i) Reverse Engineering. Except and to the limited extent as may be otherwise specifically provided by applicable law in the European Union, you may not reverse engineer, decompile, disassemble or otherwise attempt to discover the source code, underlying ideas, underlying user interface techniques or algorithms of the Software by any means whatsoever, directly or indirectly, or disclose any of the foregoing, except to the extent you may be expressly permitted to decompile under applicable law in the European Union, if it is essential to do so in order to achieve operability of the Software with another software program, and you have first requested Altova to provide the information necessary to achieve such operability and Altova has not made such information available. Altova has the right to impose reasonable conditions and to request a reasonable fee before providing such information. Any information supplied by Altova or obtained by you, as permitted hereunder, may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is

substantially similar to the expression of the Software. Requests for information from users in the European Union with respect to the above should be directed to the Altova Customer Support Department.

(j) Other Restrictions. You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software to third parties except to the limited extent set forth in Section 3 or as otherwise expressly provided. You may not copy the Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this Agreement must contain the same copyright, patent and other intellectual property markings that appear on or in the Software. You may not modify, adapt or translate the Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the Software; knowingly take any action that would cause the Software to be placed in the public domain; or use the Software in any computer environment not specified in this Agreement. You may not permit any use of or access to the Software by any third party in connection with a commercial service offering, such as for a cloud-based or web-based SaaS offering.

You will comply with applicable law and Altova's instructions regarding the use of the Software. You agree to notify your employees and agents who may have access to the Software of the restrictions contained in this Agreement and to ensure their compliance with these restrictions.

(k) NO GUARANTEE. THE SOFTWARE IS NEITHER GUARANTEED NOR WARRANTED TO BE ERROR-FREE NOR SHALL ANY LIABILITY BE ASSUMED BY ALTOVA IN THIS RESPECT. NOTWITHSTANDING ANY SUPPORT FOR ANY TECHNICAL STANDARD, THE SOFTWARE IS NOT INTENDED FOR USE IN OR IN CONNECTION WITH, WITHOUT LIMITATION, THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL EQUIPMENT, MEDICAL DEVICES OR LIFE SUPPORT SYSTEMS, MEDICAL OR HEALTH CARE APPLICATIONS, OR OTHER APPLICATIONS WHERE THE FAILURE OF THE SOFTWARE OR ERRORS IN DATA PROCESSING COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE AND ANY DATA GENERATED OR PROCESSED BY THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL DEFEND, INDEMNIFY AND HOLD ALTOVA, ITS OFFICERS AND EMPLOYEES HARMLESS FROM ANY THIRD PARTY CLAIMS, DEMANDS, OR SUITS THAT ARE BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE IN YOUR USE OR ANY DATA GENERATED BY THE SOFTWARE IN YOUR USE.

2. INTELLECTUAL PROPERTY RIGHTS

You acknowledge that the Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You acknowledge that Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. You will take no actions which adversely affect Altova's intellectual property rights in the Software. Trademarks shall be used in accordance with accepted trademark practice, including identification of trademark owners' names. Trademarks may only be used to identify printed output produced by the Software, and such use of any trademark does not give you any right of ownership in that

trademark. Altova®, XMLSpy®, Authentic®, StyleVision®, MapForce®, UModel®, DatabaseSpy®, DiffDog®, SchemaAgent®, SemanticWorks®, MissionKit®, Markup Your Mind®, Nanonull™, RaptorXML™, RaptorXML Server™, RaptorXML +XBRL Server™, Powered By RaptorXML™, FlowForce Server™, StyleVision Server™, and MapForce Server™ are trademarks of Altova GmbH. (pending or registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows XP, Windows Vista, Windows 7, and Windows 8 are trademarks of Microsoft. W3C, CSS, DOM, MathML, RDF, XHTML, XML and XSL are trademarks (registered in numerous countries) of the World Wide Web Consortium (W3C); marks of the W3C are registered and held by its host institutions, MIT, INRIA and Keio. Except as expressly stated above, this Agreement does not grant you any intellectual property rights in the Software. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

3. LIMITED TRANSFER RIGHTS

Notwithstanding the foregoing, you may transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer this Agreement, the Software and all other software or hardware bundled or pre-installed with the Software, including all copies, updates and prior versions, and all copies of font software converted into other formats, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; (c) the receiving party secures a personalized key code from Altova; and (d) the receiving party accepts the terms and conditions of this Agreement and any other terms and conditions upon which you legally purchased a license to the Software. Notwithstanding the foregoing, you may not transfer education, pre-release, or not-for-resale copies of the Software.

4. PRE-RELEASE AND EVALUATION PRODUCT ADDITIONAL TERMS

If the product you have received with this license is pre-commercial release or beta Software ("Pre-release Software"), then this Section applies. In addition, this section applies to all evaluation and/ or demonstration copies of Altova software ("Evaluation Software") and continues in effect until you purchase a license. To the extent that any provision in this section is in conflict with any other term or condition in this Agreement, this section shall supersede such other term(s) and condition(s) with respect to the Pre-release and/or Evaluation Software, but only to the extent necessary to resolve the conflict. You acknowledge that the Pre-release Software is a pre-release version, does not represent final product from Altova, and may contain bugs, errors and other problems that could cause system or other failures and data loss. CONSEQUENTLY, THE PRE-RELEASE AND/OR EVALUATION SOFTWARE IS PROVIDED TO YOU **"AS-IS" WITH NO WARRANTIES FOR USE OR PERFORMANCE,** AND ALTOVA DISCLAIMS ANY WARRANTY OR LIABILITY OBLIGATIONS TO YOU OF ANY KIND, WHETHER EXPRESS OR IMPLIED. WHERE LEGALLY LIABILITY CANNOT BE EXCLUDED FOR PRE-RELEASE AND/OR EVALUATION SOFTWARE, BUT IT MAY BE LIMITED, ALTOVA'S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIFTY DOLLARS (USD $50) IN TOTAL. If the Evaluation Software has a time-out feature, then the software will cease operation after the conclusion of the designated evaluation period. Upon such expiration date, your license will expire unless otherwise extended. Your license to use any output created with the Evaluation Software that contains generated program code (including Unrestricted Source Code) such as Java, C++, C, VB.NET or XSLT and associated project files and build scripts as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures terminates automatically upon the expiration of the designated evaluation period but the license to use such output is revived upon your purchase of a license for the Software that you evaluated and used to create such output. Access to any files created with the Evaluation Software is entirely at your risk. You acknowledge that Altova has not promised or guaranteed to you that Pre-release Software will be announced or made available to anyone in the future, that Altova has no express or implied

obligation to you to announce or introduce the Pre-release Software, and that Altova may not introduce a product similar to or compatible with the Pre-release Software. Accordingly, you acknowledge that any research or development that you perform regarding the Pre-release Software or any product associated with the Pre-release Software is done entirely at your own risk. During the term of this Agreement, if requested by Altova, you will provide feedback to Altova regarding testing and use of the Pre-release Software, including error or bug reports. If you have been provided the Pre-release Software pursuant to a separate written agreement, your use of the Software is governed by such agreement. You may not sublicense, lease, loan, rent, distribute or otherwise transfer the Pre-release Software. Upon receipt of a later unreleased version of the Pre-release Software or release by Altova of a publicly released commercial version of the Software, whether as a stand-alone product or as part of a larger product, you agree to return or destroy all earlier Pre-release Software received from Altova and to abide by the terms of the license agreement for any such later versions of the Pre-release Software.

5. LIMITED WARRANTY AND LIMITATION OF LIABILITY

(a) Limited Warranty and Customer Remedies. Altova warrants to the person or entity that first purchases a license for use of the Software pursuant to the terms of this Agreement that (i) the Software will perform substantially in accordance with any accompanying Documentation for a period of ninety (90) days from the date of receipt, and (ii) any support services provided by Altova shall be substantially as described in Section 6 of this agreement. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the Software, if any, are limited to ninety (90) days. Altova's and its suppliers' entire liability and your exclusive remedy shall be, at Altova's option, either (i) return of the price paid, if any, or (ii) repair or replacement of the Software that does not meet Altova's Limited Warranty and which is returned to Altova with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, misapplication, abnormal use, Trojan horse, virus, or any other malicious external code. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. This limited warranty does not apply to Evaluation and/or Pre-release Software.

(b) No Other Warranties and Disclaimer. THE FOREGOING LIMITED WARRANTY AND REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ALTOVA OR ITS SUPPLIER'S BREACH OF WARRANTY. ALTOVA AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, AND FOR ANY WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT WHICH THE SAME CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO YOU IN YOUR JURISDICTION, ALTOVA AND ITS SUPPLIERS MAKE NO WARRANTIES, CONDITIONS, REPRESENTATIONS OR TERMS, EXPRESS OR IMPLIED, WHETHER BY STATUTE, COMMON LAW, CUSTOM, USAGE OR OTHERWISE AS TO ANY OTHER MATTERS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ALTOVA AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, INFORMATIONAL CONTENT OR ACCURACY, QUIET ENJOYMENT, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

(c) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW EVEN IF A REMEDY FAILS ITS ESSENTIAL PURPOSE, IN NO EVENT SHALL ALTOVA

OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ALTOVA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ALTOVA'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT. Because some states and jurisdictions do not allow the exclusion or limitation of liability, the above limitation may not apply to you. In such states and jurisdictions, Altova's liability shall be limited to the greatest extent permitted by law and the limitations or exclusions of warranties and liability contained herein do not prejudice applicable statutory consumer rights of person acquiring goods otherwise than in the course of business. The disclaimer and limited liability above are fundamental to this Agreement between Altova and you.

(d) Infringement Claims. Altova will indemnify and hold you harmless and will defend or settle any claim, suit or proceeding brought against you by a third party that is based upon a claim that the content contained in the Software infringes a copyright or violates an intellectual or proprietary right protected by United States or European Union law ("Claim"), but only to the extent the Claim arises directly out of the use of the Software and subject to the limitations set forth in Section 5 of this Agreement except as otherwise expressly provided. You must notify Altova in writing of any Claim within ten (10) business days after you first receive notice of the Claim, and you shall provide to Altova at no cost such assistance and cooperation as Altova may reasonably request from time to time in connection with the defense of the Claim. Altova shall have sole control over any Claim (including, without limitation, the selection of counsel and the right to settle on your behalf on any terms Altova deems desirable in the sole exercise of its discretion). You may, at your sole cost, retain separate counsel and participate in the defense or settlement negotiations. Altova shall pay actual damages, costs, and attorney fees awarded against you (or payable by you pursuant to a settlement agreement) in connection with a Claim to the extent such direct damages and costs are not reimbursed to you by insurance or a third party, to an aggregate maximum equal to the purchase price of the Software. If the Software or its use becomes the subject of a Claim or its use is enjoined, or if in the opinion of Altova's legal counsel the Software is likely to become the subject of a Claim, Altova shall attempt to resolve the Claim by using commercially reasonable efforts to modify the Software or obtain a license to continue using the Software. If in the opinion of Altova's legal counsel the Claim, the injunction or potential Claim cannot be resolved through reasonable modification or licensing, Altova, at its own election, may terminate this Agreement without penalty, and will refund to you on a pro rata basis any fees paid in advance by you to Altova. THE FOREGOING CONSTITUTES ALTOVA'S SOLE AND EXCLUSIVE LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT. This indemnity does not apply to situations where the alleged infringement, whether patent or otherwise, is the result of a combination of the Altova software and additional elements supplied by you.

6. SUPPORT AND MAINTENANCE

Altova offers multiple optional "Support & Maintenance Package(s)" ("SMP") for the version of Software product edition that you have licensed, which you may elect to purchase in addition to your Software license. The Support Period, hereinafter defined, covered by such SMP shall be delineated at such time as you elect to purchase a SMP. Your rights with respect to support and maintenance as well as your upgrade eligibility depend on your decision to purchase SMP and the level of SMP that you have purchased:

(a) If you have not purchased SMP, you will receive the Software AS IS and will not receive any maintenance releases or updates. However, Altova, at its option and in its sole discretion on

a case by case basis, may decide to offer maintenance releases to you as a courtesy, but these maintenance releases will not include any new features in excess of the feature set at the time of your purchase of the Software. In addition, Altova will provide free technical support to you for thirty (30) days after the date of your purchase (the "Support Period" for the purposes of this paragraph 6(a), and Altova, in its sole discretion on a case by case basis, may also provide free courtesy technical support during your thirty (30) day evaluation period. Technical support is provided via a Web-based support form only, and there is no guaranteed response time.

(b) If you have purchased SMP, then solely for the duration of its delineated Support Period, **you are eligible to receive the version of the Software edition** that you have licensed and all maintenance releases and updates for that edition that are released during your Support Period. For the duration of your SMP's Support Period, you will also be eligible to receive upgrades to the comparable edition of the next version of the Software that succeeds the Software edition that you have licensed for applicable upgrades released during your Support Period. The specific upgrade edition that you are eligible to receive based on your Support Period is further detailed in the SMP that you have purchased. Software that is introduced as separate product is not included in SMP. Maintenance releases, updates and upgrades may or may not include additional features. In addition, Altova will provide Priority Technical Support to you for the duration of the Support Period. Priority Technical Support is provided via a Web-based support form only and Altova will make commercially reasonable efforts to respond via e-mail to all requests within forty-eight (48) hours during Altova's business hours (MO-FR, 8am UTC – 10pm UTC, Austrian and US holidays excluded) and to make reasonable efforts to provide work-arounds to errors reported in the Software.

During the Support Period you may also report any Software problem or error to Altova. If Altova determines that a reported reproducible material error in the Software exists and significantly impairs the usability and utility of the Software, Altova agrees to use reasonable commercial efforts to correct or provide a usable work-around solution in an upcoming maintenance release or update, which is made available at certain times at Altova's sole discretion.

If Altova, in its discretion, requests written verification of an error or malfunction discovered by you or requests supporting example files that exhibit the Software problem, you shall promptly provide such verification or files, by email, telecopy, or overnight mail, setting forth in reasonable detail the respects in which the Software fails to perform. You shall use reasonable efforts to cooperate in diagnosis or study of errors. Altova may include error corrections in maintenance releases, updates, or new major releases of the Software. Altova is not obligated to fix errors that are immaterial. Immaterial errors are those that do not significantly impact use of the Software as determined by Altova in its sole discretion. Whether or not you have purchased the Support & Maintenance Package, technical support only covers issues or questions resulting directly out of the operation of the Software and Altova will not provide you with generic consultation, assistance, or advice under any circumstances.

Updating Software may require the updating of software not covered by this Agreement before installation. Updates of the operating system and application software not specifically covered by this Agreement are your responsibility and will not be provided by Altova under this Agreement. Altova's obligations under this Section 6 are contingent upon your proper use of the Software and your compliance with the terms and conditions of this Agreement at all times. Altova shall be under no obligation to provide the above technical support if, in Altova's opinion, the Software has failed due to the following conditions: (i) damage caused by the relocation of the Software to another location or CPU; (ii) alterations, modifications or attempts to change the Software without Altova's written approval; (iii) causes external to the Software, such as natural disasters, the failure or fluctuation of electrical power, or computer equipment failure; (iv) your failure to maintain the Software at Altova's specified release level; or (v) use of the Software with other software

without Altova's prior written approval. It will be your sole responsibility to: (i) comply with all Altova-specified operating and troubleshooting procedures and then notify Altova immediately of Software malfunction and provide Altova with complete information thereof; (ii) provide for the security of your confidential information; (iii) establish and maintain backup systems and procedures necessary to reconstruct lost or altered files, data or programs.

7. SOFTWARE ACTIVATION, UPDATES AND LICENSE METERING

(a) License Metering. The Software includes a built-in license metering module that is designed to assist you with monitoring license compliance in small local area networks (LAN). The metering module attempts to communicate with other machines on your local area network (LAN). You permit Altova to use your internal network for license monitoring for this purpose. This license metering module may be used to assist with your license compliance but should not be the sole method. Should your firewall settings block said communications, you must deploy an accurate means of monitoring usage by the end user and preventing users from using the Software more than the Permitted Number.

(b) License Compliance Monitoring. You are required to utilize a process or tool to ensure that the Permitted Number is not exceeded. Without prejudice or waiver of any potential violations of the Agreement, Altova may provide you with additional compliance tools should you be unable to accurately account for license usage within your organization. If provided with such a tool by Altova, you (a) are required to use it in order to comply with the terms of this Agreement and (b) permit Altova to use your internal network for license monitoring and metering and to generate compliance reports that are communicated to Altova from time to time.

(c) 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 Master 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 between your computer and the Altova Master License Server. You agree that Altova may use these measures and you agree to follow any applicable requirements. You further agree that use of license key codes that are not or were not generated by Altova and lawfully obtained from Altova, or an authorized reseller as part of an effort to activate or use the Software violates Altova's intellectual property rights as well as the terms of this Agreement. You agree that efforts to circumvent or disable Altova's copyright protection mechanisms, the license management mechanism, or the Altova Master License Server violate Altova's intellectual property rights as well as the terms of this Agreement. Altova expressly reserves the rights to seek all available legal and equitable remedies to prevent such actions and to recover lost profits, damages and costs.**

(d) LiveUpdate. Altova provides a new LiveUpdate notification service to you, which is free of charge. Altova may use your internal network and Internet connection for the purpose of transmitting license-related data to an Altova-operated LiveUpdate server to validate your license at appropriate intervals and determine if there is any update available for you.

(e) Use of Data. The terms and conditions of the Privacy Policy are set out in full at http://www.altova.com/privacy and are incorporated by reference into this Agreement. By your acceptance of the terms of this Agreement and/or use of the Software, you authorize the collection, use and disclosure of information collected by Altova for the purposes provided for in this Agreement and/or the Privacy Policy. Altova has the right in its sole discretion to amend this provision of the Agreement and/or Privacy Policy at any time. You are encouraged to review the

terms of the Privacy Policy as posted on the Altova Web site from time to time.

(f) Audit Rights. You agree that Altova may audit your use of the Software for compliance with the terms of this Agreement at any time, upon reasonable notice. In the event that such audit reveals any use of the Software by you other than in full compliance with the terms of this Agreement, you shall reimburse Altova for all reasonable expenses related to such audit in addition to any other liabilities you may incur as a result of such non-compliance.

(g) Notice to European Users. Please note that the information as described in paragraph 7(d) above may be transferred outside of the European Economic Area, for purposes of processing, analysis, and review, by Altova, Inc., a company located in Beverly, Massachusetts, U.S.A., or its subsidiaries or Altova's subsidiaries or divisions, or authorized partners, located worldwide. You are advised that the United States uses a sectoral model of privacy protection that relies on a mix of legislation, governmental regulation, and self-regulation. You are further advised that the Council of the European Union has found that this model does not provide "adequate" privacy protections as contemplated by Article 25 of the European Union's Data Directive. (Directive 95/46/EC, 1995 O.J. (L 281) 31). Article 26 of the European Union's Data Directive allows for transfer of personal data from the European Union to a third country if the individual has unambiguously given his consent to the transfer of personal information, regardless of the third country's level of protection. By agreeing to this Agreement, you consent to the transfer of all such information to the United States and the processing of that information as described in this Agreement and the Privacy Policy.

8. TERM AND TERMINATION

This Agreement may be terminated (a) by your giving Altova written notice of termination; (b) by Altova, at its option, giving you written notice of termination if you commit a breach of this Agreement and fail to cure such breach within ten (10) days after notice from Altova; or (c) at the request of an authorized Altova reseller in the event that you fail to make your license payment or other monies due and payable. In addition the Agreement governing your use of a previous version of the Software that you have upgraded or updated is terminated upon your acceptance of the terms and conditions of the Agreement accompanying such upgrade or update. Upon any termination of the Agreement, you must cease all use of the Software that this Agreement governs, destroy all copies then in your possession or control and take such other actions as Altova may reasonably request to ensure that no copies of the Software remain in your possession or control. The terms and conditions set forth in Sections 1(h), 1(i), 1(j), 1(k), 1(l), 2, 5, 7, 9, 10, 11, and 11 survive termination as applicable.

9. RESTRICTED RIGHTS NOTICE AND EXPORT RESTRICTIONS

The Software was developed entirely at private expense and is commercial computer software provided with **RESTRICTED RIGHTS**. Use, duplication or disclosure by the U.S. Government or a U.S. Government contractor or subcontractor is subject to the restrictions set forth in this Agreement and as provided in FAR 12.211 and 12.212 (48 C.F.R. §12.211 and 12.212) or DFARS 227. 7202 (48 C.F.R. §227-7202) as applicable. Consistent with the above as applicable, Commercial Computer Software and Commercial Computer Documentation licensed to U.S. government end users only as commercial items and only with those rights as are granted to all other end users under the terms and conditions set forth in this Agreement. Manufacturer is Altova GmbH, Rudolfsplatz 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the Software or Documentation except as authorized by United States law and the laws of the jurisdiction in which the Software was obtained. In particular, but without limitation, the Software or Documentation may not be exported or re-exported (i) into (or to a national or resident of) any U.S. embargoed country or (ii) to anyone on the U.S. Treasury Department's list

of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the Software, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

10. U.S. GOVERNMENT ENTITIES

Notwithstanding the foregoing, if you are an agency, instrumentality or department of the federal government of the United States, then this Agreement shall be governed in accordance with the laws of the United States of America, and in the absence of applicable federal law, the laws of the Commonwealth of Massachusetts will apply. Further, and notwithstanding anything to the contrary in this Agreement (including but not limited to Section 5 (Indemnification)), all claims, demands, complaints and disputes will be subject to the Contract Disputes Act (41 U.S.C. §§7101 *et seq.*), the Tucker Act (28 U.S.C. §1346(a) and §1491), or the Federal Tort Claims Act (28 U.S.C. §§1346(b), 2401-2402, 2671-2672, 2674-2680), FAR 1.601(a) and 43.102 (Contract Modifications); FAR 12.302(b), as applicable, or other applicable governing authority. For the avoidance of doubt, if you are an agency, instrumentality, or department of the federal, state or local government of the U.S. or a U.S. public and accredited educational institution, then your indemnification obligations are only applicable to the extent they would not cause you to violate any applicable law (e.g., the Anti-Deficiency Act), and you have any legally required authorization or authorizing statute.

11. THIRD PARTY SOFTWARE

The Software may contain third party software which requires notices and/or additional terms and conditions. Such required third party software notices and/or additional terms and conditions are located at our Website at http://www.altova.com/legal_3rdparty.html and are made a part of and incorporated by reference into this Agreement. By accepting this Agreement, you are also accepting the additional terms and conditions, if any, set forth therein.

12. JURISDICTION, CHOICE OF LAW, AND VENUE

If you are located in the European Union and are using the Software in the European Union and not in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht, Wien (Commercial Court, Vienna) in connection with any such dispute or claim.

If you are located in the United States or are using the Software in the United States then this Agreement will be governed by and construed in accordance with the laws of the Commonwealth of Massachusetts, USA (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the federal or state courts of the Commonwealth of Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts of the Commonwealth of Massachusetts in connection with any such dispute or claim.

If you are located outside of the European Union or the United States and are not using the Software in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree

that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht Wien (Commercial Court, Vienna) in connection with any such dispute or claim. This Agreement will not be governed by the conflict of law rules of any jurisdiction or the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded.

13. TRANSLATIONS

Where Altova has provided you with a foreign translation of the English language version, you agree that the translation is provided for your convenience only and that the English language version will control. If there is any contradiction between the English language version and a translation, then the English language version shall take precedence.

14. GENERAL PROVISIONS

This Agreement contains the entire agreement and understanding of the parties with respect to the subject matter hereof, and supersedes all prior written and oral understandings of the parties with respect to the subject matter hereof. Any notice or other communication given under this Agreement shall be in writing and shall have been properly given by either of us to the other if sent by certified or registered mail, return receipt requested, or by overnight courier to the address shown on Altova's Web site for Altova and the address shown in Altova's records for you, or such other address as the parties may designate by notice given in the manner set forth above. This Agreement will bind and inure to the benefit of the parties and our respective heirs, personal and legal representatives, affiliates, successors and permitted assigns. The failure of either of us at any time to require performance of any provision hereof shall in no manner affect such party's right at a later time to enforce the same or any other term of this Agreement. This Agreement may be amended only by a document in writing signed by both of us. In the event of a breach or threatened breach of this Agreement by either party, the other shall have all applicable equitable as well as legal remedies. Each party is duly authorized and empowered to enter into and perform this Agreement. If, for any reason, any provision of this Agreement is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this Agreement, and this Agreement shall continue in full force and effect to the fullest extent allowed by law. The parties knowingly and expressly consent to the foregoing terms and conditions.

Last updated: 2015/09/03

Chapter 20

Glossary

20 Glossary

The glossary section includes the list of terms pertaining to MapForce.

20.1 C

Component

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

- Constants
- Databases
- Filters
- Conditions
- Flat files (CSV, fixed-length, and other text files)
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- Variables
- XBRL documents
- XML Schemas and DTDs

Connection

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

Connector

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

20.2 F

Fixed Length Field (FLF)

A common text format where data is conventionally separated into fields which have a fixed length (for example, the first 5 characters of every row represent a transaction ID, and the next 20 characters represent a transaction description).

FlexText

FlexText is a module in MapForce Enterprise Edition which enables you to convert data from non-standard or legacy text files of high complexity to other formats supported by MapForce, and vice versa.

20.3 I

Input component

An input component is a MapForce component that enables you to pass simple values to a mapping. Input components are commonly used to pass file names or other string values to a mapping at runtime. Input components should not be confused with source components.

20.4 M

MapForce

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

Mapping

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

MFF

The file name extension of MapForce function files.

MFD

The file name extrension of MapForce design documents (mappings).

MFP

The file name extension of MapForce Project files.

MFT

The file name extension of MapForce FlextText template documents.

20.5 O

Output component

An output component (or "simple ouput") is a MapForce component which enables you to return a string value from the mapping. Output components represent just one possible type of target components, but should not be confused with the latter.

20.6 P

parent-context

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

20.7 S

Source component

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

20.8 T

Target component

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

Index

.

0

2

8

9

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z