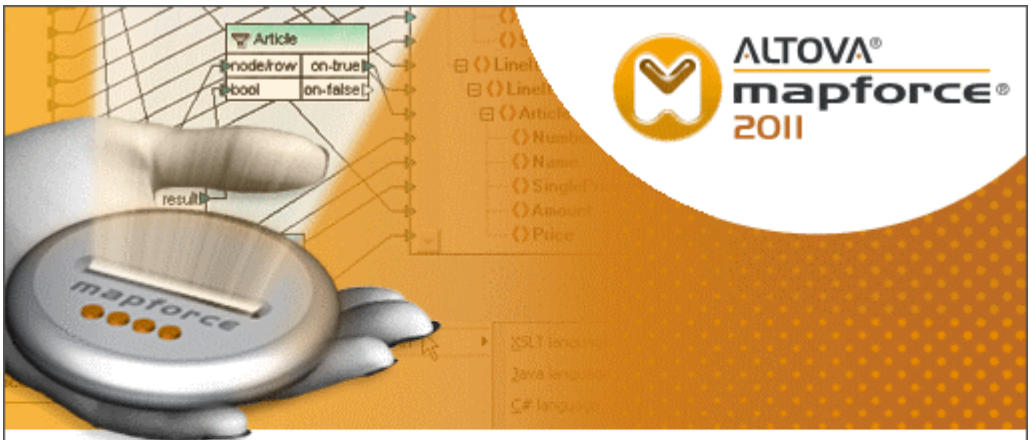
User and Reference Manual



Altova MapForce 2011 User & Reference Manual

Published: 2011

Table of Contents

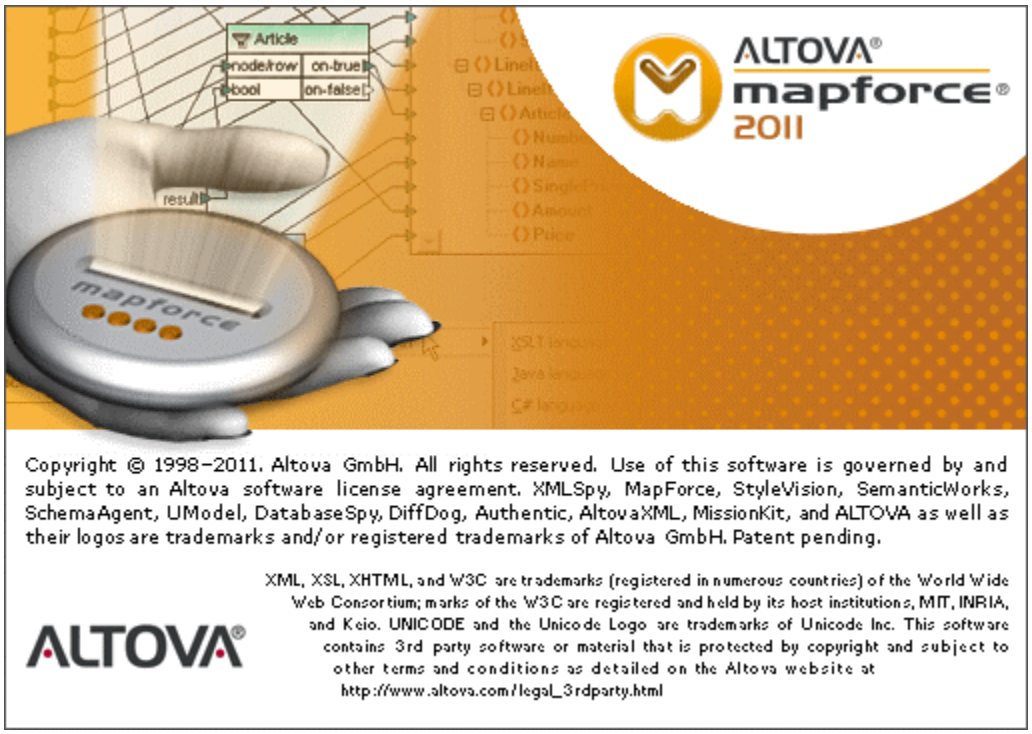
Index

Chapter 1

MapForce 2011

1 MapForce 2011

MapForce® 2011 Professional Edition is a visual data mapping tool for advanced data integration projects. MapForce® is a 32/64-bit Windows application that runs on Windows 7, Windows Vista, Windows Server 2003/2008, and Windows XP. 64-bit support is available for the Enterprise and Professional editions.



Last updated: 07/08/2011

Chapter 2

What's new...

2 What's new...

New features in MapForce Version **2011R3** include:

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

New features in MapForce Version 2011R2 include:

- Built-in Execution Engine now supports streaming output
- Find function capability in Library window
- Reverse mapping
- Extendable IF-ELSE function
- Node Name and parsing functions in Core Library
- 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 target components using StyleVision Power Stylesheets
- 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

New features in MapForce Version 2010 Release 3 include:

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

New features in MapForce Version 2010 Release 2 include:

- 64-bit MapForce Enterprise / Professional editions on 64-bit operating systems: Windows Server 2003/2008, Windows XP, Windows Vista and Windows 7Automatic connection of identical child connectors when moving a parent connector
- Support for fields in the SQL Where component
- Ability to add compiled Java .class and .NET assembly files
- Ability to tokenize input strings for further processing

New features in MapForce Version 2010 include:

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

New features in MapForce Version 2009 SP1 include:

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

New features in MapForce Version 2009 include:

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

New features in MapForce Version 2008 Release 2 include:

- Support for Streams as input/output in generated Java and C# code
- Generation of Visual Studio 2008 project files for C++ and C#
- Ability to automatically generate XML Schemas for XML files
- 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 databases - IBM DB2 and others
- Direct querying of databases
- SQL-WHERE filter and SQL statement wizard
- Code generator optimization and improved documentation

Chapter 3

MapForce overview

3 MapForce overview

Altova web site: Introduction to MapForce

What is mapping?
Basically the contents of a component are mapped, or transformed, to another component. An XML, or text document, a database, can be mapped to a different target XML document, CSV text document, or database. The transformation is accomplished by an automatically generated XSLT 1.0 or 2.0 Stylesheet, the Built-in execution engine, or generated program code.

MapForce also has the ability to have a single component process multiple input files of a directory and output multiple files to a single component as well.

When creating an XSLT transformation, a **source schema** is mapped to a **target schema**. Thus elements/attributes in the source schema are "connected" to other elements/attributes in the target schema. As an XML document instance is associated to, and defined by, a schema file, you actually end up mapping two XML documents to each other.

MapForce® supports:

- Graphical mapping from and to any combination and any number of:
  - XML Schemas as source and target

  Professional Edition, additionally:
  - Flat files: delimited (CSV) and fixed-length formats as source and target
  - Relational databases as source and target

  Enterprise Edition, additionally:
  - EDI files: UN/EDIFACT, ANSI X12 including HIPAA, HL7 2.x, IATA PADIS, and SAP IDocs as source and target
  - FlexText™ files as source and target
  - Office Open XML Excel 2007 and higher files, as source and target
  - XBRL instance files and taxonomies
- Automatic code generation
  - XSLT 1.0 and 2.0

  Professional Edition and **Enterprise** Edition, additionally:
  - XQuery
  - Java, C# and C++
  - 64-bit version support
- On-the-fly transformation and preview of all mappings, without code generation or compilation
- Ability to preview intermediate components in a mapping chain of two or more components connected to a target component (pass-through preview).
- Ability to preview output of target components using StyleVision Power Stylesheets
- Powerful visual function builder for creating user-defined functions
- Accessing MapForce user interface and functions through MapForce API (ActiveX control)
- Definition of custom XSLT 1.0 and 2.0 libraries
- Support for XPath 2.0 functions in XSLT 2.0 and XQuery
- Definition of user-defined functions/components, having complex in/outputs
- Support for source-driven / mixed content mapping and copy-all connections
- Automatic retention of mapping connectors of missing nodes/items
- Support for HL7 version 3.x. as it is XML Schema based

Professional Edition, additionally:

- XML data mapping to/from databases - IBM DB2 and others
- Direct querying of databases
- SQL-WHERE filter and SQL statement wizard
- SQL SELECT statements as mapping data sources
- Integration of custom C++, Java and C# functions
- Project management functions to group mappings
- MapForce plug-in for Eclipse 3.4 / 3.5 / 3.6
- MapForce for Microsoft Visual Studio
- Documentation of the mapping design

Enterprise Edition, additionally:

- Creation of SOAP 1.1 and SOAP 1.2 Web service projects and mapping of Web service operations from WSDL 1.1 and 2.0 files
- Direct calling of Web service functions
- FlexText™: advanced legacy file processing

All transformations are available in one workspace where multiple sources and multiple targets can be mixed, and a rich and extensible function library provides support for any kind of data manipulation.

3.1 Terminology

The terms used in this documentation are defined below.

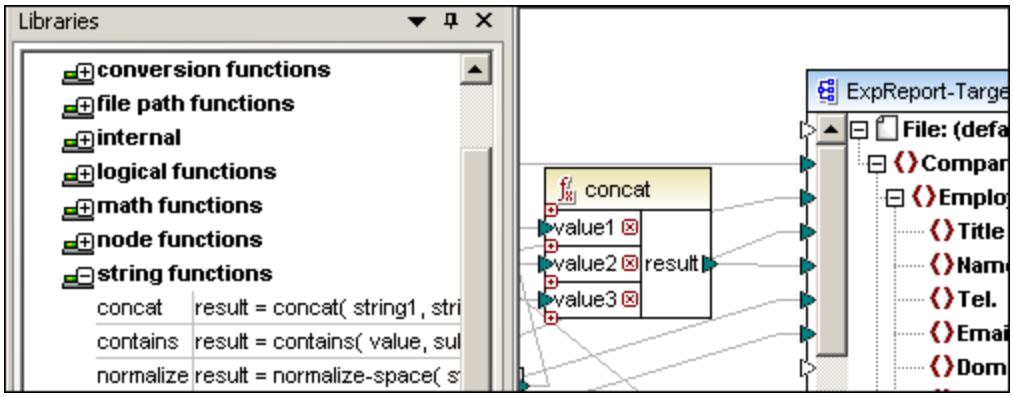
Library
A Library is a collection of functions visible in the Libraries window. There are several types of functions, core and language specific, as well as user-defined and custom functions. Please see the section on functions for more details.

Component
In MapForce many of graphical elements you can insert/import or place in the Mapping tab, become components. Components have small **triangles** which allow you to map data between source and target components by creating connections between them.

The following files become components when placed in the mapping area:

- Schemas and DTDs: Source and target schemas
- Databases: Source and target databases
- Flat files: CSV and other text files
-
- Function types: XSLT/XSLT2, XQuery, Java, C#, and C++ functions, as well as Constants, Filters and Conditions

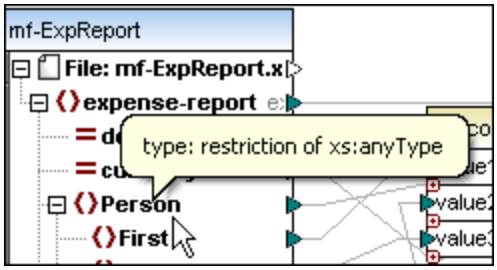
Function
A function is predefined component that operates on data e.g. **Concat**. Functions have input and/or output **parameters,** where each parameter has its own input/output icon. Functions are available in the Libraries window and are logically grouped; hitting CTRL+F allows you to search for a function. Dragging a function into the Mapping window creates a function component. Please see the section Functions and Libraries for more details.



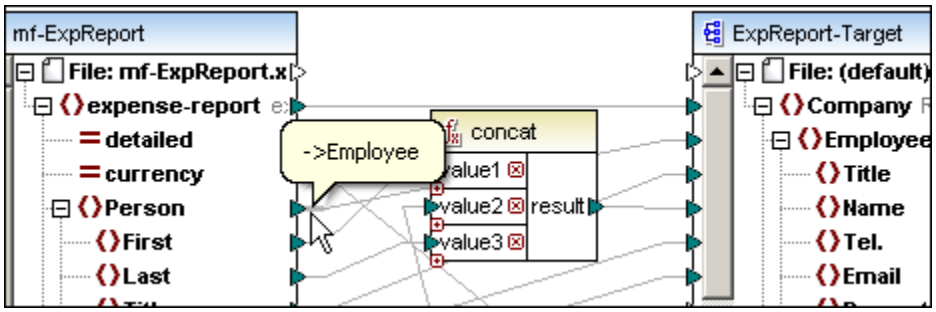
Java selected

Item
An item represents the data that can be mapped from component to component. An item can be either an **element**, an **attribute**, a database field.

Each **item** has an **input** and **output** icon. It is not mandatory that items be of the same type (element or attribute) when you create a mapping between them.



Input, Output icon
The small triangles visible on components are **input** and **output** icons. Clicking an icon and dragging, creates a **connector** which connects to another icon when you "drop" it there. The connector **represents a mapping** between the two sets of data the icons represent. Please see the section "Mapping between components" for more information.

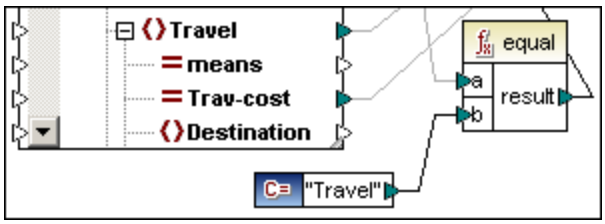


Connector
The connector is the **line** that joins two icons. It represents the **mapping** between the two sets of data the icons represent. Please see the section "Mapping between components" for more information.

Several types of connector can be defined:

- Target Driven (Standard) connectors, see: "source-driven / mixed content vs. standard mapping"
- Copy-all connectors, please see "Copy-all connections"
- Source Driven (mixed content) connectors, see "source driven and mixed content mapping"

Constant
A constant is a component that supplies fixed data to an input icon of a function or component. E.g. the string "Travel" is connected to the "b" parameter of the equal function . The data is entered into a dialog box when creating, or double clicking, the component. There is only one output icon on a constant function. You can select from the following types of data: String, Number, and All other (String).

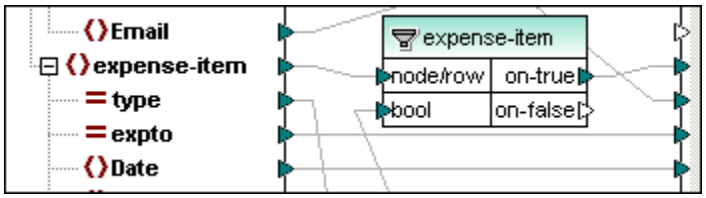


Variable
Inserts an Intermediate Variable which is equivalent to a regular (non-inline) user-defined

function. Variables are structural components, without instance files, and are used to simplify the mapping process.

Filter: Node/Row

A filter is a component that filters data using two input and output parameters: **node/row** and **bool**, and **on-true**, **on-false**. If the Boolean is true, then the value/content of the node/row parameter is forwarded to the **on-true** parameter.

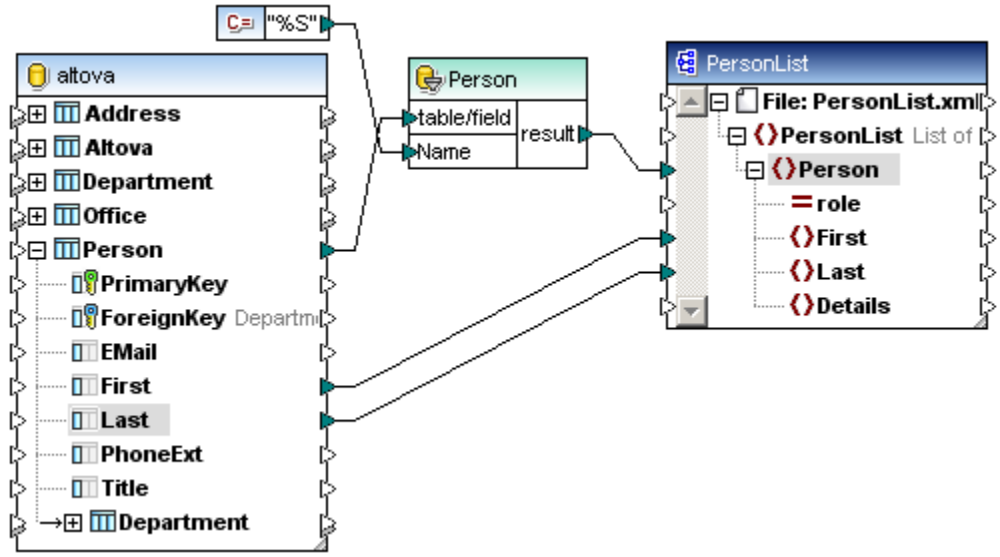


The **on-false** output parameter, outputs the complement node set defined by the mapping, please see Multiple target schemas / documents for more information.

SQL-WHERE Condition

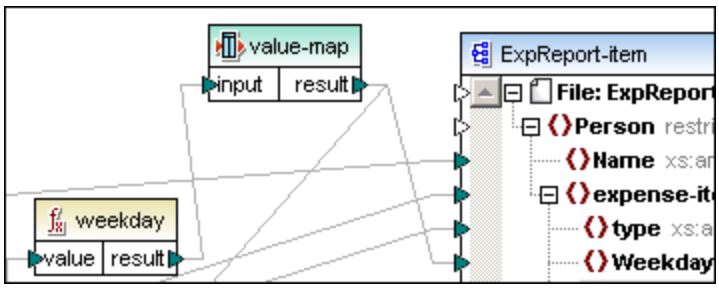
The SQL-WHERE component allows you to filter database data conditionally. Double clicking the component allows you to enter the SQL-WHERE statement. The SQL WHERE component is comprised of two parts:

- The **Select** statement that is automatically generated when you connect to a database table
- The **WHERE** clause that you manually enter in the SQL WHERE Select text box. Note that the foreign keys are automatically included in the select statement.



Value-Map

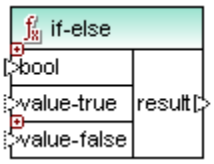
The Value-Map component allows you to transform a set of input data, into a different set of output data, using a type of lookup table.



Double clicking the component, opens the value map table. The left column of the table defines the input, while the right column defines the transformed data you want to output.

IF-Else Condition

A condition is a component 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**. Please see Condition, in the Reference section for an example.



- 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 now **extendable**. This means that you can check for multiple IF values and use the **otherwise** parameter to output the Else condition/value. Please see Insert | If-Else for more information.

Chapter 4

MapForce tutorial

4 MapForce tutorial

This tutorial takes you through several tasks which provide an overview of how to use MapForce 2011 to its fullest.

The goal of this tutorial is to map a simple employee travel expense report to a more complex company report. In our tutorial example, each employee fills in the fields of the personal report. This report is mapped to the company report and routed to the Administration department. Extra data now has to be entered in conjunction with the employee, the result being a standardized company expense report.

In this tutorial, you will learn how to:

- Set up the mapping environment
- Map the source XML file (the personal expense report) to the output target (the company expense travel report)
- Apply filters to the source data
- Generate an XSLT transformation file
- Transform the source data to the output target using the generated XSLT file

Installation and configuration
This tutorial assumes that you have successfully installed MapForce on your computer and received a free evaluation key-code, or are a registered user of the product. The evaluation version of MapForce is fully functional but limited to a 30-day period. You can request a regular license from our secure web server or through any one of our resellers.

Tutorial example files
The tutorial makes use of the following components:

- Source and (multiple) target schemas
- An MS Access database as the data source
- Several functions including: concat, filter, equal and constants

All the files used in this tutorial are initially available in the C:\Documents and Settings\All Users\Application Data\Altova folder. When any single user starts the application for the first time, the example files for that user are copied to the ...\MapForceExamples\Tutorial\ folder. Therefore do not move, edit, or delete the example files in the initial ...\All Users\.... directory.

The XSLT and transformed XML files are also supplied. The following files are used in the tutorial:

Personal expense report:

- Tut-ExpReport.mfd — The expense report mapping (single target)
- Tut-ExpReport-multi.mfd — The multi-schema target expense report mapping
- PersonDB.mfd — The employee mapping, using an MS Access DB as the data source
- mf-ExpReport.xml — Personal expense report XML instance document
- mf-ExpReport.xsd — Associated schema file

Company expense report:

- ExpReport-Target.xml — Company expense report XML instance document
- ExpReport-Target.xsd — Associated schema file

File paths in Windows XP, Windows Vista, and Windows 7

File paths given in this documentation will not be the same for all operating systems. You should note the following correspondences:

- (My) Documents folder: The My Documents folder of Windows XP is the Documents folder of Windows Vista and Windows 7. It is located by default at the following respective locations. Example files are usually located in a sub-folder of the (My) Documents folder.

Windows XP	`C:\Documents and Settings\<username>\My Documents`
Windows Vista, Windows 7	`C:\Users\<username>\Documents`

- Application folder: The Application folder is the folder where your Altova application is located. The path to the Application folder is, by default, the following.

Windows XP	`C:\Program Files\Altova`
Windows Vista, Windows 7	`C:\Program Files\Altova`
32 bit Version on 64-bit OS	`C:\Program Files (x86)\Altova`

4.1 Setting up the mapping environment

This section deals with defining the source and target schemas we want to use for the mapping.

Objective
In this section of the tutorial, you will learn how to set up the mapping environment in MapForce. Specifically, you will learn how to:

- Create the source and target schema components
- Define the source XML file
- Select the root element of the target schema

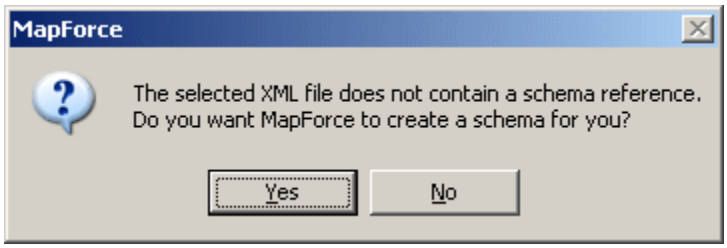
Commands used in this section

Insert XML Schema/File: Click this icon to open the standard Windows **Open** dialog box and select the file from your file system.

4.1.1 Adding components to the Mapping pane

After you have started MapForce, you must add the source and target files to a Mapping pane; this can also be done by dragging files from Windows Explorer and dropping them into a Mapping pane.

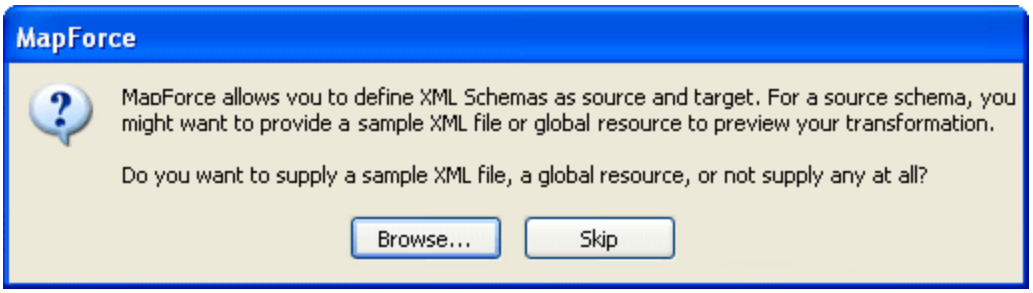
MapForce can automatically generate an XML schema based on an existing XML file if the XML Schema is not available. A dialog box automatically appears, prompting you if an accompanying XML Schema file cannot be found when inserting an XML file using the Insert XML Schema / File menu item.



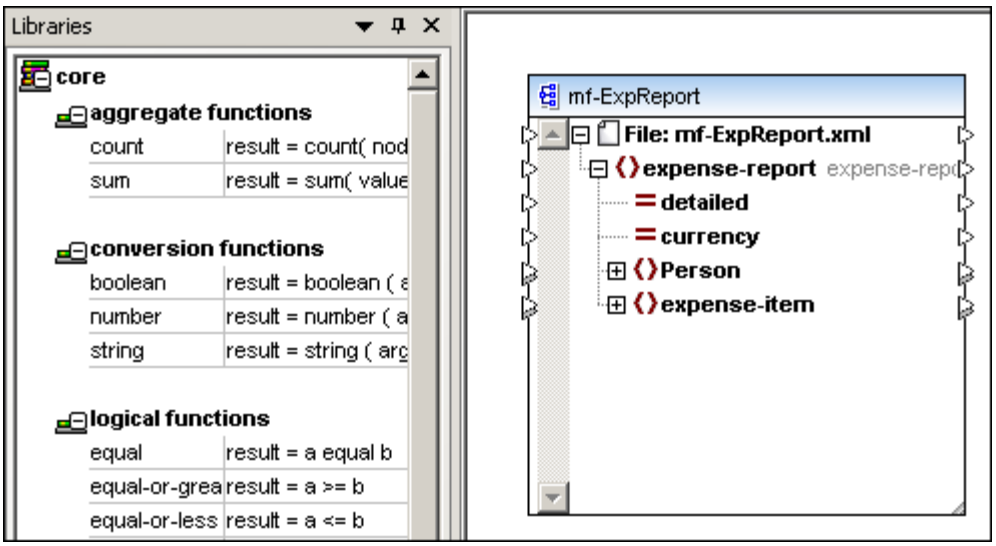
When generating a schema from an XML file, data types for elements/attributes must be inferred from the XML instance document and may not be exactly what you expect. Please check whether the generated schema is an accurate representation of the instance data.

To create the source schema component:

1. Click the **Insert XML Schema/File** icon or select the menu option **Insert | XML Schema/File...**.
2. In the **Open** dialog box, browse to the Tutorial subfolder of the ...\MapForce2011 \MapForceExamples folder and select the **mf-ExpReport.xsd** file.
   You are now prompted for a sample XML file to provide the data for the preview tab.

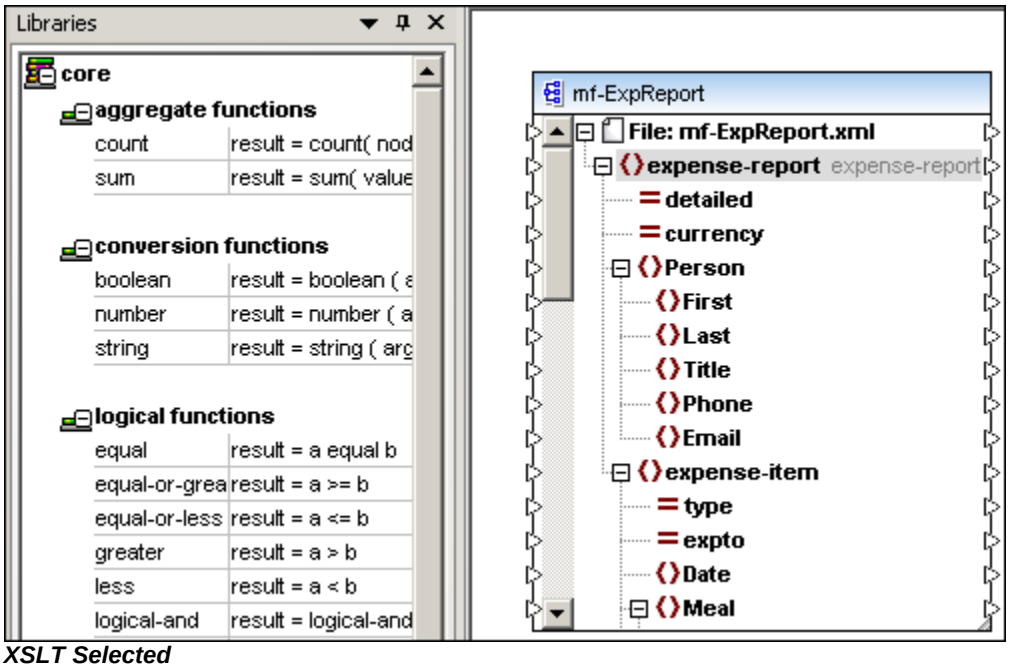


3. Click the **Browse...** button, and select the **mf-ExpReport.xml** file.
   The source schema component now appears in the Mapping pane.



XSLT selected

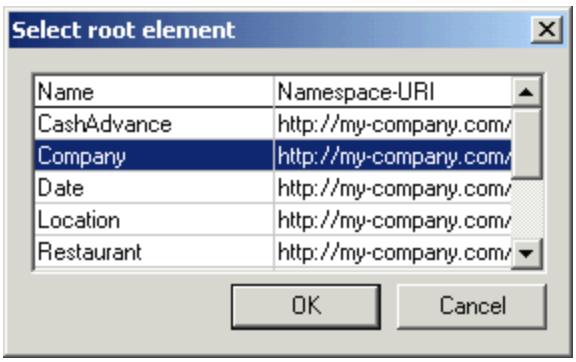
4. Click the **expense-report** item/element (of the component) and hit the ***** key, on the numeric keypad, to view all the items.
5. Click the resize corner at the lower right of the component, and drag to resize it. Note: double clicking the resize corner, resizes the component to a "**best fit**", encompassing all items.



XSLT Selected

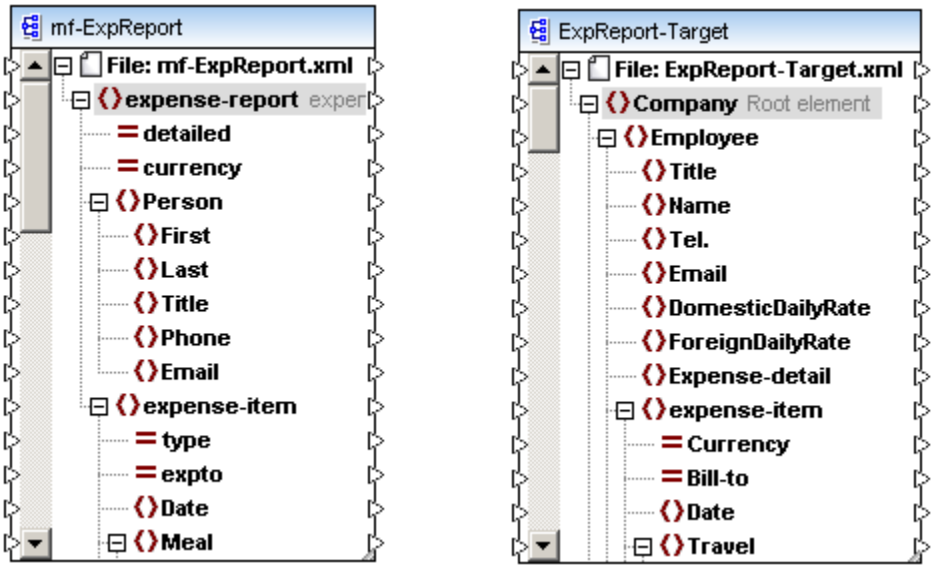
To create the target schema component:

1. Click the **Insert XML Schema/File** icon or select the menu option **Insert | XML Schema/File...**.
2. Select the **ExpReport-Target.xsd** file from the **Open** dialog box.
   You are now prompted for a sample XML file for this schema.
3. Click the **Skip** button, and select **Company** as the root element of the target document.



The target schema component now appears in the mapping tab.

4. Click the **Company** entry and hit the ***** key on the numeric keypad to view all the items.
5. Double click the resize corner icon to resize the component.



We are now ready to start mapping schema items from the source to the target schema.

4.2 Creating a mapping

In the previous section, you defined the source and target schema components of your mapping. We will now start mapping the actual data.

Objective
To learn how to map the source and target components and fine-tune your mapping result using functions and filters.

- Using connectors to map schema items
- Use a concat function to combine elements of the source data
- Filter source data to pass on only specific expenses to the target report

Commands used in this section

Auto Connect Matching Children: Click this icon to toggle the automatic connection of matching child nodes, on and off.

Insert Constant: Click this icon to add a constant component to the currently active Mapping pane.

Filter: Nodes/Rows: Click this icon to add a filter component to the currently active Mapping pane.

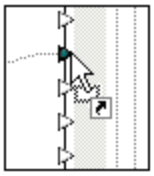
Altova web site: Mapping data - data integration and XML mapping

4.2.1 Mapping schema items

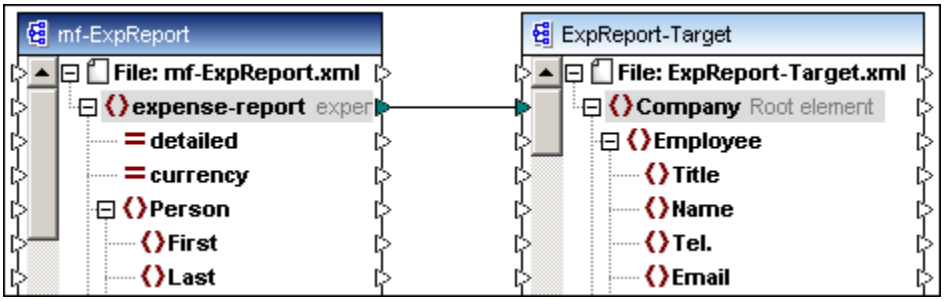
This section deals with defining the mappings between the source and target schema items.

To map the mf-ExpReport and ExpReport-Target schemas:

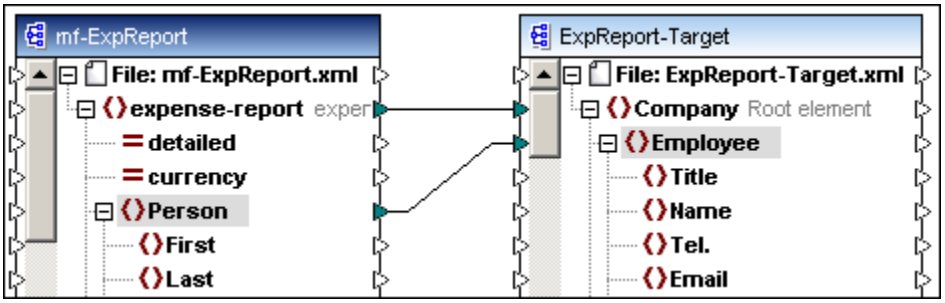
1. Click the **expense-report** item in the mf-ExpReport source schema and drag.
   A connector line is automatically created from the output icon and is linked to the mouse pointer which has now changed shape.
2. Move the mouse pointer near to the **Company** item in the ExpReport-Target schema, and "drop" the connector the moment the mouse pointer changes back to the arrow shape. A small link icon appears below the mouse pointer, and the input icon and item name in the target component, are highlighted when the drop action is possible.



A connector has now been placed between the source and target schemas. A mapping has now been created from the schema source to the target document.

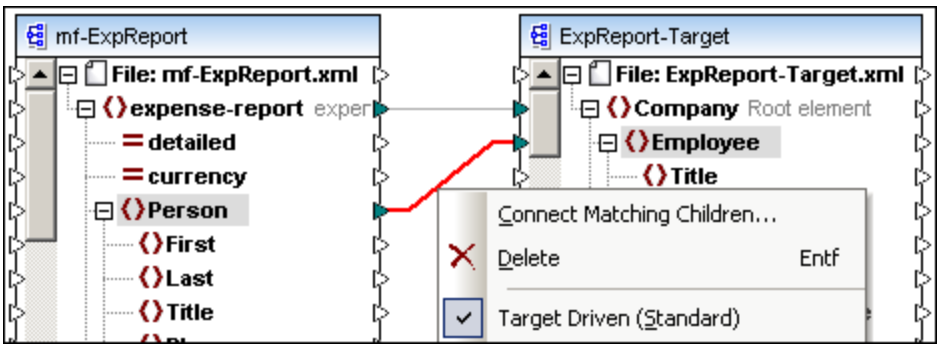


3. Use the above method to create a mapping between the Person and Employee items.

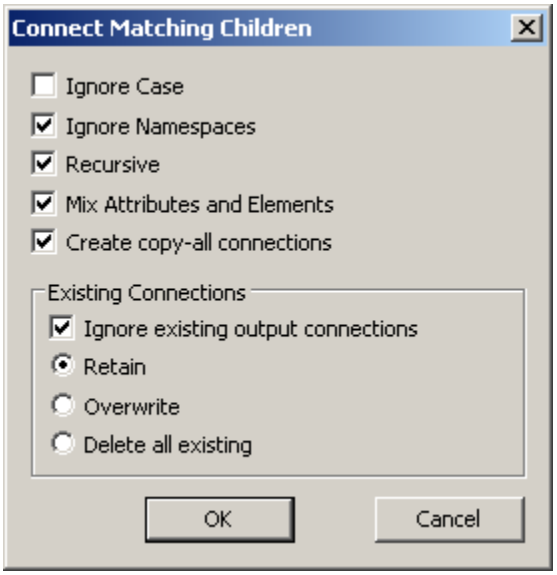


If the **Auto Connect Matching Children**  icon is active, then the Title and Email items will also be connected automatically, if not:

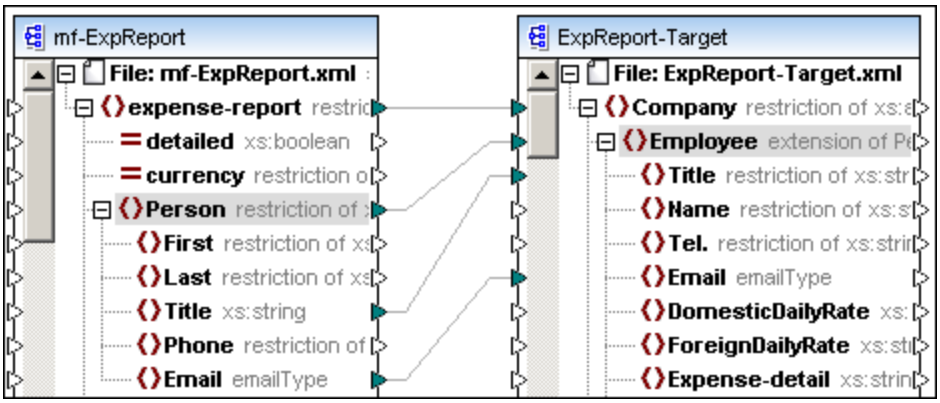
4. Right-click the "Person" connector and select **Connect Matching Children...** from the pop-up menu.



This opens the **Connect Matching Children** dialog box.

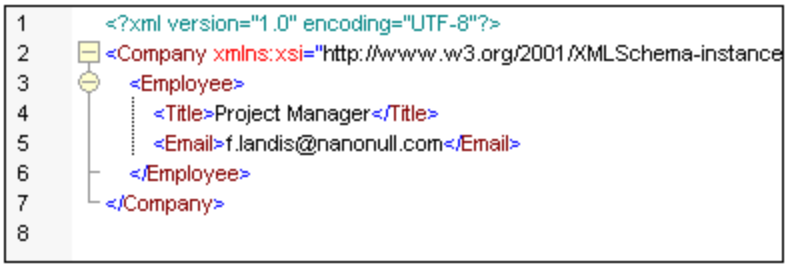


5. Activate the check boxes as shown above and click **OK** to confirm. For more information please see the section on Connector properties.



Mappings have been automatically created for the **Title** and **Email** items of both schemas.

6. Click the Output button to see the result in the Output pane.



You will notice that the Title and Email fields contain data originating from the XML Instance document.

7. Click the Mapping button to return to the Mapping pane and continue mapping.

Please note: The settings you select in the **Connect Matching Children** dialog box, are retained until you change them. These settings can be applied to a connection by either: using the context menu, or by clicking the Auto connect child items icon to activate, or deactivate this option.

4.2.2 Using functions to map data

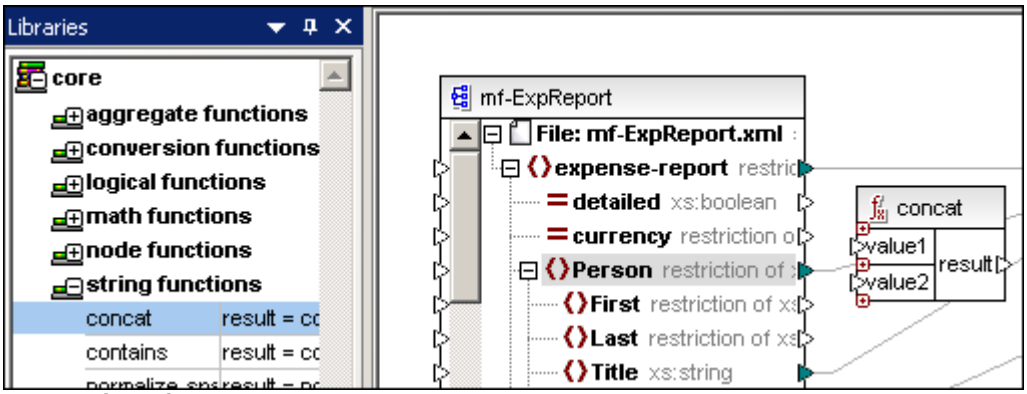
The aim of this section is to combine two sets of data from the source schema, and place the result in a single item in the target document. This will be done by:

- Using the **Concat** string function to combine the **First** and **Last** elements of the source schema
- Using a **Constant** function to supply the space character needed to separate both items
- Placing the result of this process into the **Name** item of the target schema.

Please note that some of the previously defined mappings are not shown in the following screen shots for the sake of clarity.

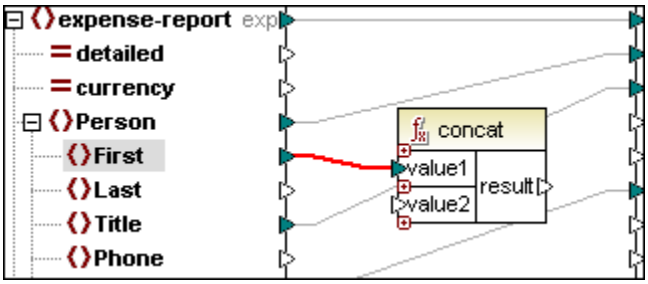
To combine items by using functions:

1. In the Libraries tab, expand the **string functions** group in the **core** library, click the **concat** entry, and drag it into the Mapping pane.

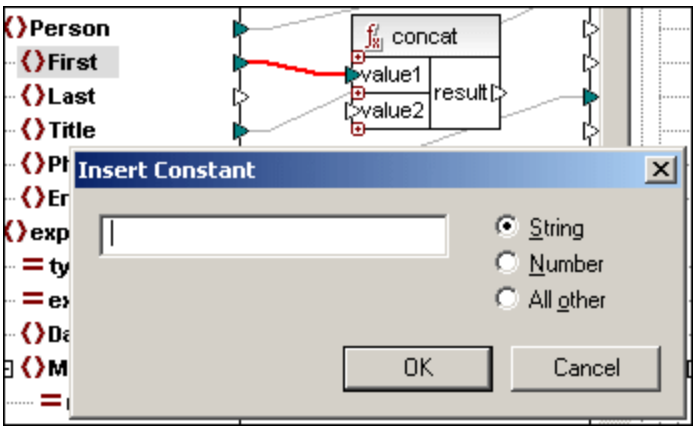


XSLT Selected

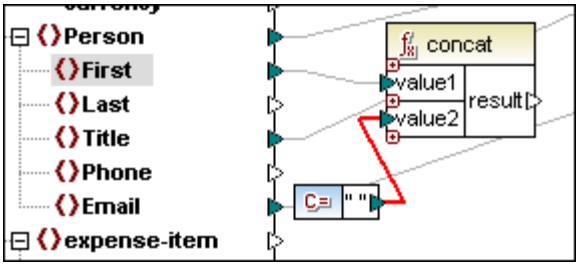
2. In the mf-ExpReport component, select item **First** and, keeping the mouse button pressed, create a connection by dragging the mouse cursor to the **value1** input of the concat component.



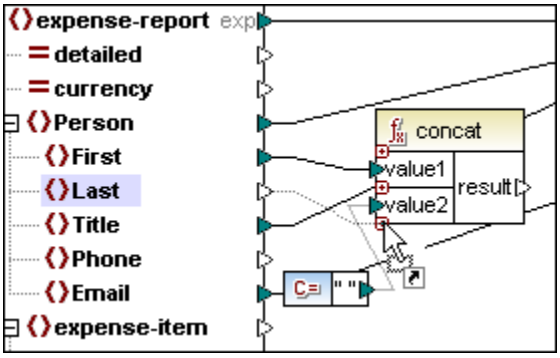
3. Right-click on the background near **value2** and select **Insert Constant** from the context menu, to insert a constant component.



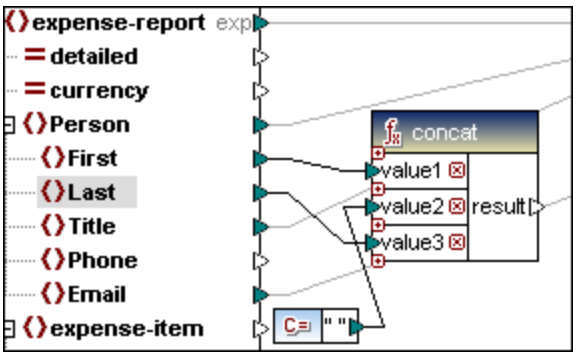
4. Enter a space character in the text box and click **OK**.
   The constant component is now in the working area. Its contents are displayed next to the output icon.
5. Create a connection between the **constant** component and **value2** of the concat component.



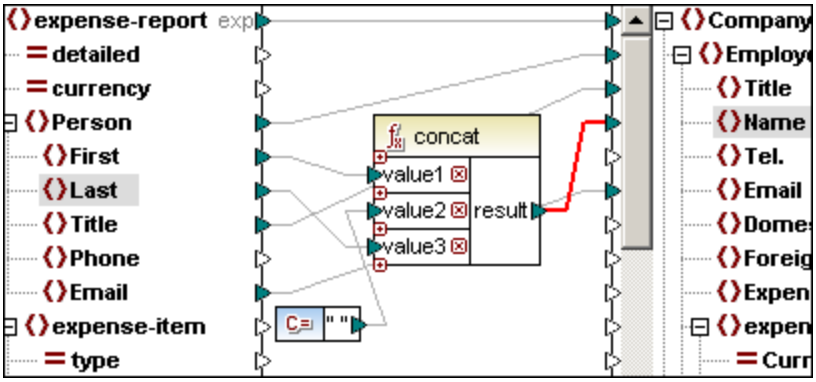
6. In the mf-ExpReport component, click the item **Last** and drop the connector on the "**+**" icon of the concat function, just below **value2**. The mouse cursor changes to show when you can drop the connector.



This automatically enlarges the concat function by one more item (value), to which the Last item is connected.



7. Connect the **result** icon of the concat component, to the **Name** item in the target schema.



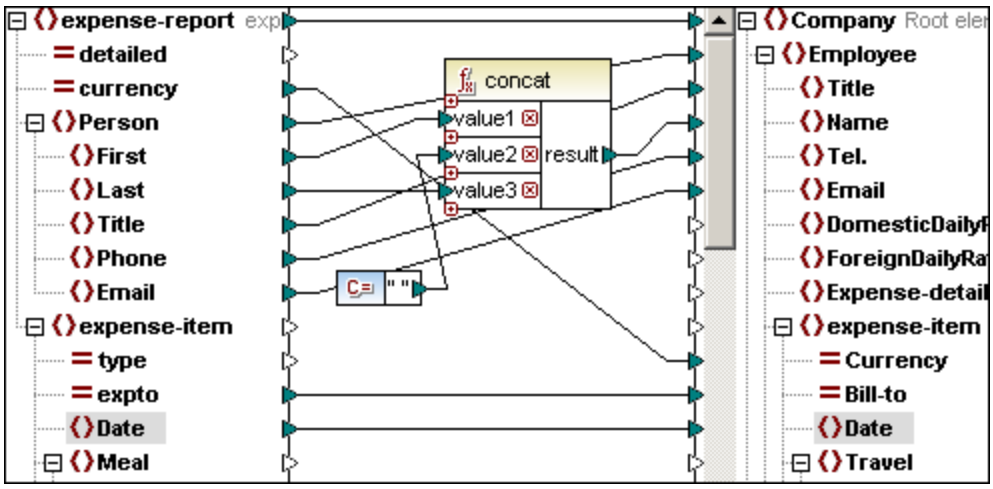
8. Click the **Output** button to see the result of the current mapping in the Output pane.

You will see that the Person name "Fred Landis" is now contained between the **Name** tags. The first and last name have been separated by a space character as well.

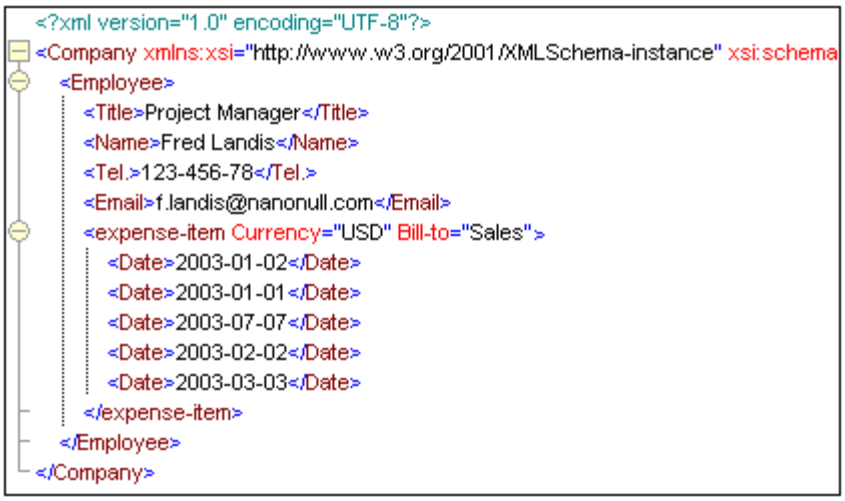
Mapping the rest of the personal data

Create mappings between the following items:

- currency to Currency
- Phone to Tel.
- expto to Bill-to
- Date to Date



Click the Output button to see the result.



There are currently five items originating from the assigned XML instance file.

Please note: Functions can be grouped into user-defined functions/components to optimize screen usage. Please see the section on User-defined functions/components for an example on how to combine the concat and constant functions into a single user-defined function/component.

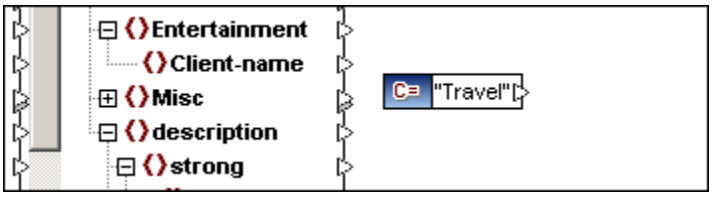
4.2.3 Filtering data

The aim of this section is to filter out the Lodging and Meal expenses, and only pass on the Travel expenses to the target schema/document. This will be done by:

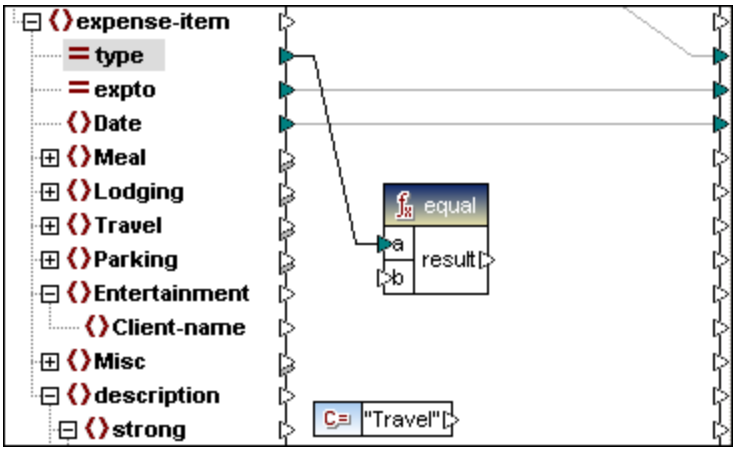
- Using the **Equal** function to test the value of a source item
- Using a **Constant** function to supply the comparison string that is to be tested
- Using the **Filter** function which passes on the Travel data, if the bool input value is true
- Placing the on-true result of this process, into the **expense-item** element of the target schema/document.

To filter data:

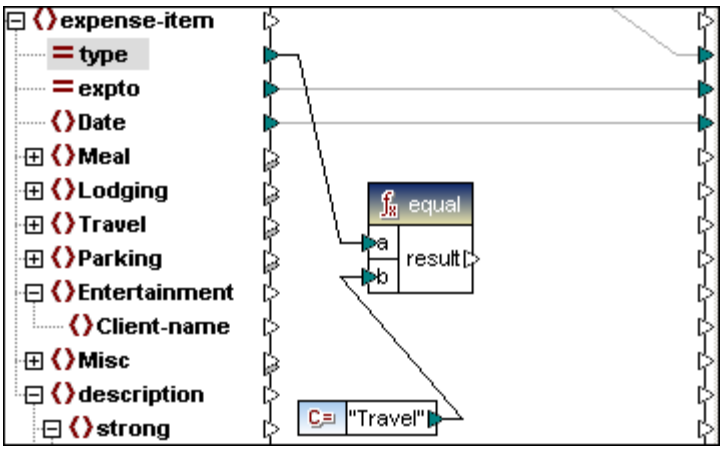
1. Click the **Insert Constant** button to insert a constant component and enter the string **"Travel"** into the input field.



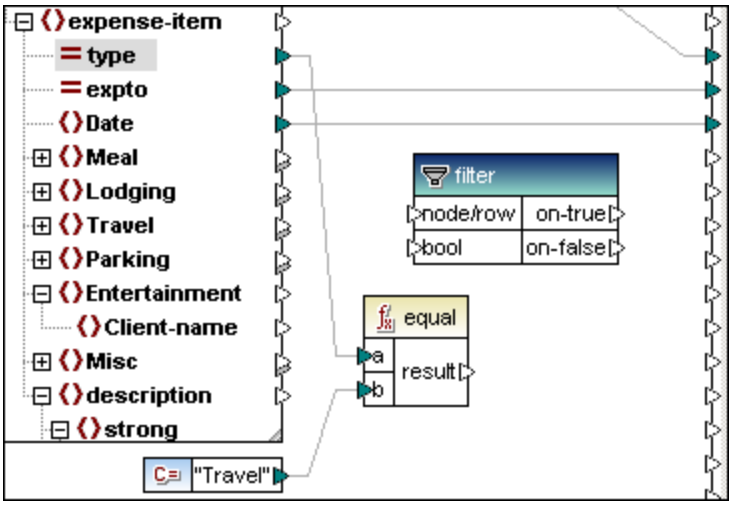
2. In the Libraries tab, expand the **logical functions** group in the **core** library and drag the logical function **equal** into the Mapping pane.
3. Connect the (expense-item) **type** item in the source schema to the **a** parameter of the equal function.



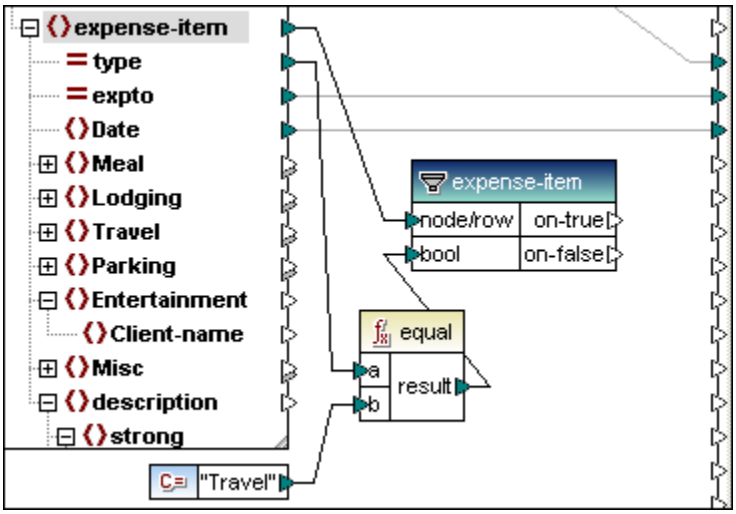
4. Connect the **result** icon of the "Travel" **constant** component, to the **b** parameter of the equal function.



5. Select the menu option **Insert | Filter: Nodes/Rows**.

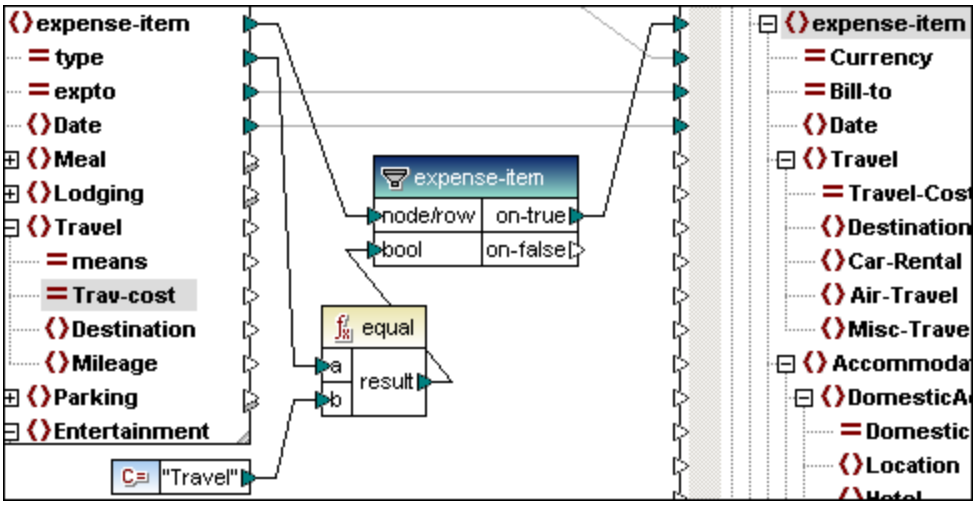


6. Connect the **result** icon of the **equal** component, to the **bool** parameter of the **filter** component.
7. Connect the **expense-item** icon of the source schema with the **node/row** parameter of the filter component.

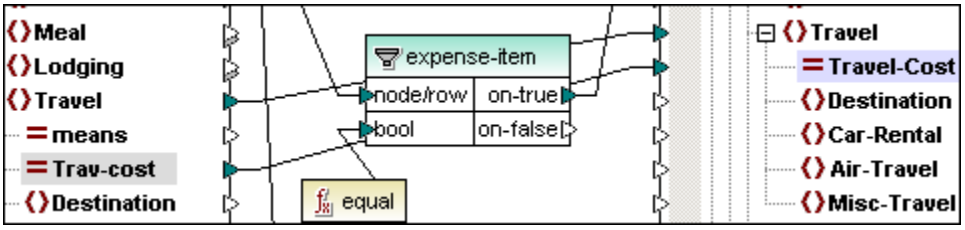


Note that the filter component name, now changes to "expense-item".

8. Connect the **on-true** icon of the **filter** component with the **expense-item** element of the target document.



9. Connect the **Travel** item in the source schema, with the **Travel** item in the target schema/document.
10. Connect the **Trav-cost** item with the **Travel-Cost** item in the target schema/document.



11. Click the **Output** button to see the result in the Output pane.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schema
3    <Employee>
4      <Title>Project Manager</Title>
5      <Name>Fred Landis</Name>
6      <Tel.>123-456-78</Tel.>
7      <Email>f.landis@nanonull.com</Email>
8      <expense-item Currency="USD" Bill-to="Development">
9        <Date>2003-01-02</Date>
10       <Travel Travel-Cost="337.88"/>
11     </expense-item>
12     <expense-item Currency="USD" Bill-to="Accounting">
13       <Date>2003-07-07</Date>
14       <Travel Travel-Cost="1014.22"/>
15     </expense-item>
16     <expense-item Currency="USD" Bill-to="Marketing">
17       <Date>2003-02-02</Date>
18       <Travel Travel-Cost="2000"/>
19     </expense-item>
20   </Employee>
21 </Company>
22
```

Please note: The **on-false** parameter of the filter component, outputs the **complement** node set that is mapped by the on-true parameter. In this example it would mean all **non-travel** expense items.

The number of expense-items have been reduced to three. Checking against the supplied **mf-ExpReport.xml** file, reveals that only the Travel records remain, the Lodging and Meal records have been filtered out.

4.3 Generating XSLT 1.0 and 2.0 code

Now that you have created the mapping, you can do the actual transformation of the source data. MapForce can generate two flavors of XSLT code: XSLT 1.0 and XSLT 2.0.

Objective
In this section of the tutorial, you will learn how to preview, generate and save the XSLT code, and how to execute the generated XSLT. Specifically, you will learn how to do the following:

- Generate and save XSLT code in the desired flavor
- Download the Altova XML engine and execute the transformation batch file

Commands used in this section

File | Generate code in: Select this option to choose the output language (i.e. **XSLT 1.0** or **XSLT 2.0**). The command opens the **Browse For Folder** dialog box where you define folder where the generated XSLT should be saved.

To generate XSLT code:

1. Select the menu item **File | Generate code in | XSLT 1.0 (or XSLT 2.0)**.
2. Select the folder you want to place the generated XSLT file in, and click **OK**.
   A message appears showing that the generation was successful.
3. Navigate to the designated folder and you will find the XSLT with the file name **MappingMapToExpReport-Target.xslt** (i.e. in the form: MappingMapTo< TargetSchemaName>).

Transforming the XML file
The folder in which the XSLT file is placed also contains a batch file called **DoTransform.bat** which uses AltovaXML to transform the XML file. AltovaXML can be downloaded from the Altova website free of charge.

To transform the personal expense report to the company expense report:

1. Download and install the free AltovaXML engine from the AltovaXML download page. AltovaXML is installed to C:\Program Files\Altova\AltovaXML2011\ per default.
2. Start the **DoTransform.bat** batch file located in the previously designated output folder. This generates the output file **ExpReport-Target.xml** in the current folder.

Note that you might need to add the AltovaXML installation location to the path variable of the Environment Variables.

4.4 Handling multiple target schemas / documents

This section deals with creating a second target schema / document, into which **non-travel** expense records will be placed, and follows on from the current tutorial example **Tut-ExpReport.mfd**.

Objective
In this section of the tutorial, you will learn how to add a second target and how to generate multiple target schema output. Specifically, you will learn how to:

- Create a second target schema component
- Filter out all non-travel output in your example report
- Define multiple target schemas of the same name
- View specific output
- Generate XSLT for multiple target schemas
- Generate program code for multiple target schemas

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.

Properties: Located in the **Components** menu. In source and target schemas, this command is used to define, among other properties, the schema file, input XML file, and output XML file.

Preview: Appears in the title bar of components when multiple target files have been defined. Click this icon to select a specific component for the output preview.

Save generated output: Located in the **Output** menu/pane. Click this icon to open the Standard Windows **Save As** dialog box and select the location where you want to save the generated output data.

Validate Output: Located in the **Output** menu/pane. Click this icon to check whether the generated output is valid. The result of the validation appears in the Messages window.

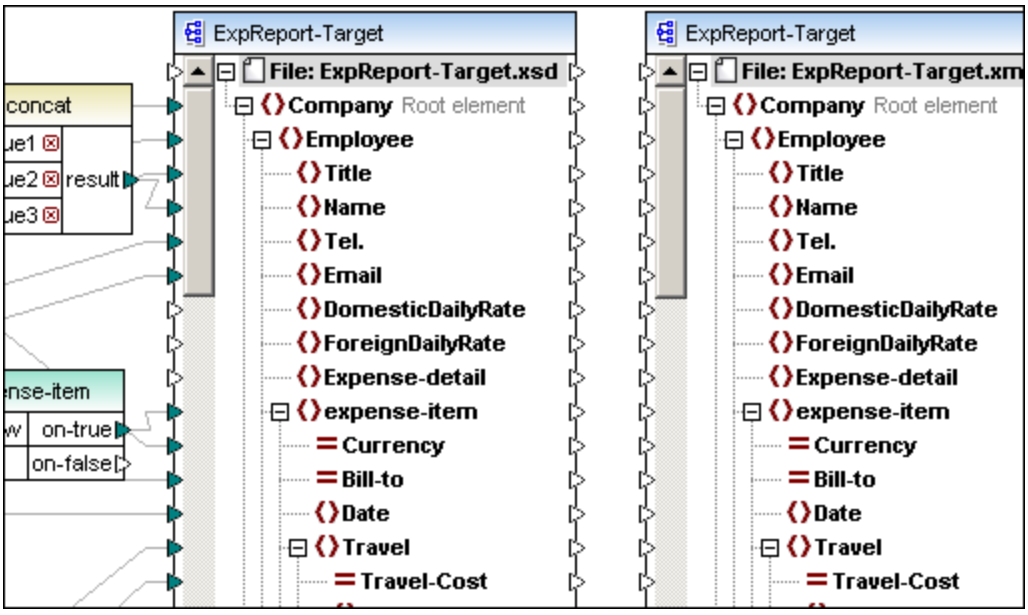
File | Generate code in: Select this option to choose the output language (i.e. **XSLT 1.0** or **XSLT 2.0**). The command opens the **Browse For Folder** dialog box where you define folder where the generated XSLT should be saved.

4.4.1 Creating a second target component

In this section of the tutorial you will learn how to create a second target schema component which filters out all the non-travel data.

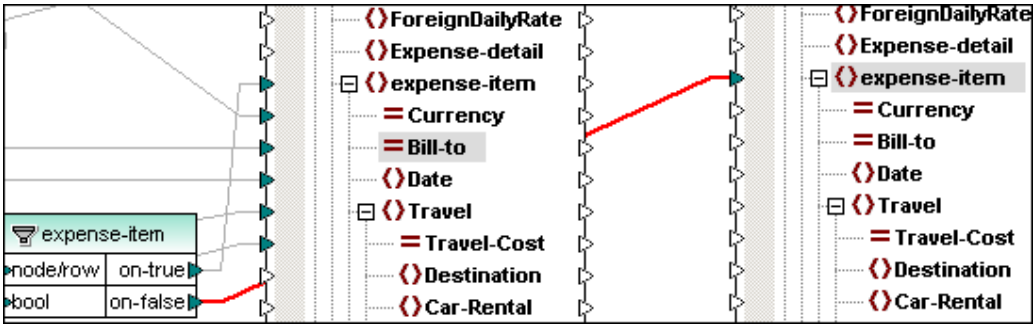
To create the second target schema component:

1. Click the **Insert XML Schema/File** icon.
2. Select the **ExpReport-Target.xsd** file from the **Open** dialog box.
   You are now prompted for a sample XML file for this schema.
3. Click **Skip**, and select **Company** as the root element of the target document.
   The target schema component now appears in the Mapping pane.
4. Click the **Company** entry and hit the ***** key on the numeric keypad to view all the items.
5. Click the expand window icon and resize the component. Place the schema components so that you can view and work on them easily.
   There is now one source schema, **mf-expReport**, and two target schemas, both **ExpReport-Target,** visible in the Mapping pane.

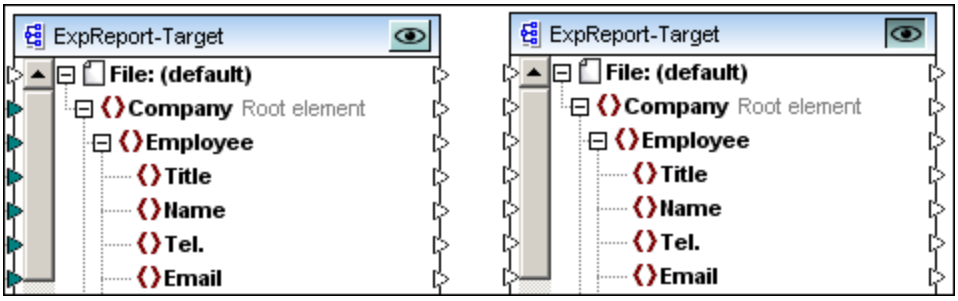


To filter out the non-travel data:

1. Connect the **on-false** icon of the **filter** component with the **expense-item** element of the **second** target schema / document.



   A message appears stating that you are now working with multiple target schemas / documents.
2. Click **OK** to confirm.



A **Preview** icon is now visible in the title bar of each target schema component.

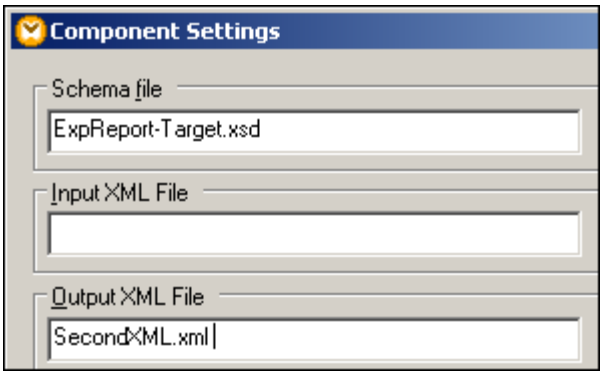
Clicking the Preview icon defines which of the target schema data is to be displayed, when you subsequently click the XSLT, XSLT2, XQuery, or Output buttons.

Defining multiple target schemas of the same name for code generation

Both target schemas have the same name in this example, so we have to make sure the generated code does not overwrite the first result file with the second. When generating XSLT there is no need to do this.

To define the Output XML file:

1. Right click the **second** target schema/document (ExpReport-Target), and select the **Properties** option.
2. Enter a file name in the **Output XML instance field**, ...\MapForceExamples\Tutorial\\ **SecondXML.xml** for example.



3. Click **OK** to close the **Properties** dialog box.
   It is recommended to insert the **absolute path** if you generate code. The example above, uses the default installation path of MapForce. Note that the output file name has now changed to **SecondXML.xml** in the component.

Creating mappings for the rest of the expense report data

Create the following mappings **between the source** schema **and second target** schema. You created the same connectors for the first target schema, so there is nothing new here:

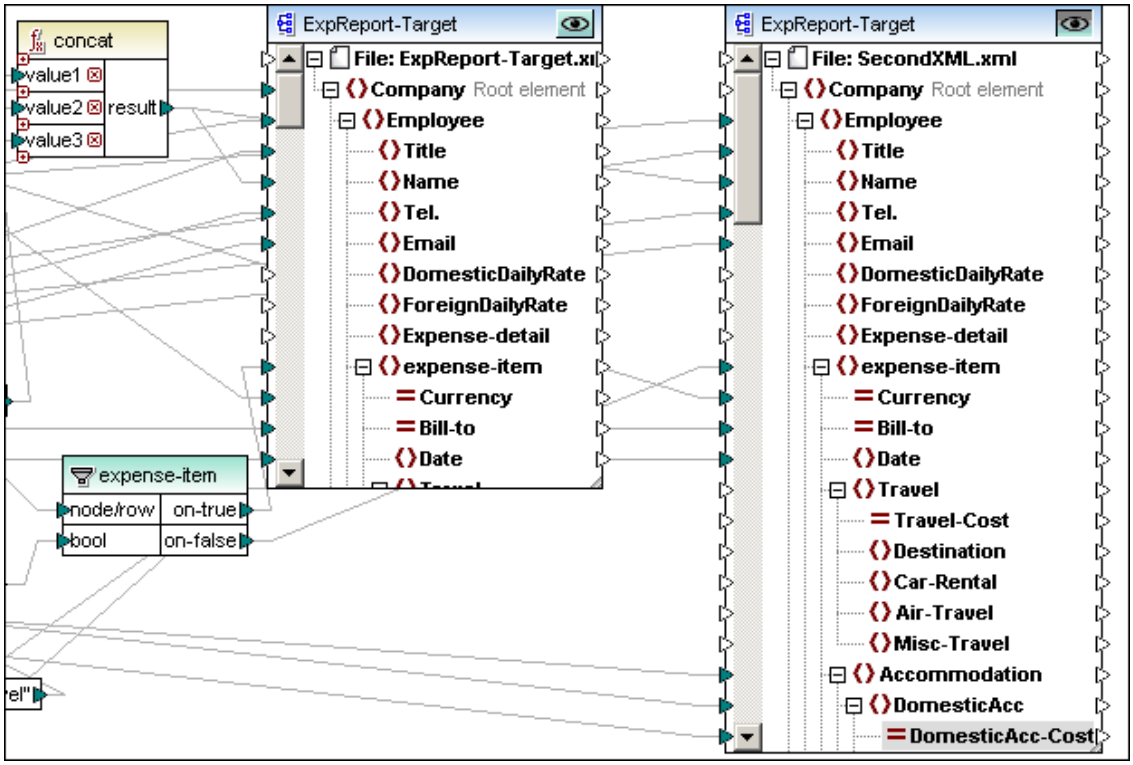
- `Person to Employee`
- `Tittle to Title`
- `Phone to Tel.`
- `Email to Email`
- `currency to Currency`
- `expto to Bill-to`
- `Date to Date`

Create the following mapping **between the existing concat function** and **second target** schema:

- `result to Name`

To create the remaining non-travel mappings:

1. Connect the **Lodging** item in the source schema to **Accommodation** in the second target schema.
2. Connect the **Lodging** item to **DomesticAcc**
3. Connect the **Lodge-Cost** item to **DomesicAcc-Cost**

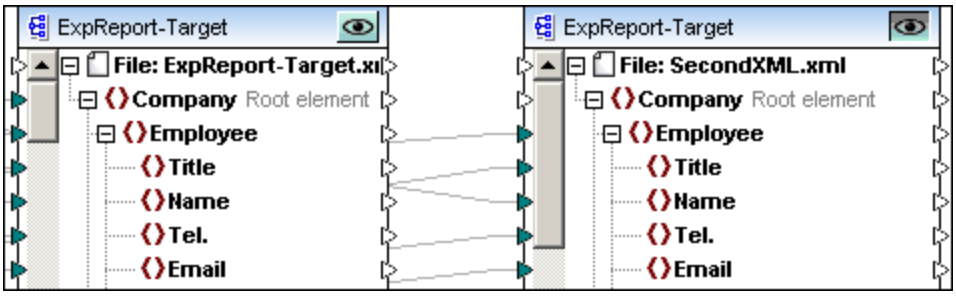


4.4.2 Viewing and generating multiple target schema output

Clicking the Preview icon lets you select which of the schema targets you want to preview.

To view specific XSLT output:

1. Click the **Preview icon** in the title bar of the **second** schema component, to make it active (if not already active).



2. Click the **Output** button of the Mapping tab group.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schema
3    <Employee>
4      <Title>Project Manager</Title>
5      <Name>Fred Landis</Name>
6      <Tel.>123-456-78</Tel.>
7      <Email>f.landis@nanonull.com</Email>
8      <expense-item Currency="USD" Bill-to="Sales">
9        <Date>2003-01-01</Date>
10       <Accommodation>
11         <DomesticAcc DomesticAcc-Cost="121.2"/>
12       </Accommodation>
13     </expense-item>
14     <expense-item Currency="USD" Bill-to="Sales">
15       <Date>2003-03-03</Date>
16     </expense-item>
17   </Employee>
18 </Company>
19
```

The XML output contains two records both billed to Sales: the Domestic Accommodation cost of $121.2 and an Expense-item record which only contains a date. This record originates from the expense-item Meal. There is currently no mapping between meal costs and domestic accommodation costs, and even if there were, no cost would appear as the XML instance does not supply one.

Please note: You can save this XML data by clicking the **Save generated output** icon, while viewing the XML output in the preview window .

The resulting XML instance file can also be validated against the target schema, by clicking the validate button .

To generate XSLT 1.0 / XSLT 2.0 code for multiple target schemas:

1. Select the menu item **File | Generate code in | XSLT 1.0 (or XSLT 2.0)**.
2. Select the folder you want to place the generated XSLT files, and click **OK**.
   A message appears showing that the generation was successful.

3. Navigate to the designated folder and you will find two XSLT files with the file names: **MappingExpReport-Target.xslt** and **MappingExpReport-Target2.xslt**

To transform the personal expense report to the company expense report:

1. Download and install the free AltovaXML engine from the AltovaXML download page. AltovaXML is installed to c:\Program Files\Altova\AltovaXML2011\ per default.
2. Start the **DoTransform.bat** batch file located in the previously designated output folder.
   This generates the output file **ExpReport-Target.xml** in the current folder.

Note: you might need to add the AltovaXML installation location to the path variable of the Environment Variables.

To generate program code for multiple target schemas:

1. Select the menu item **File | Generate code in | XQuery, Java, C#, or C++**.
2. Select the folder you want to place the generated files in, and click OK.
   A message appears showing that the generation was successful.
3. Navigate to the designated folder and compile your project.
4. Compile and execute the program code using your specific compiler.
   Two XML files are generated by the application.

Please note: A **JBuilder** project file and **Ant** build scripts are generated by MapForce to aid in compiling the Java code, see the section on JDBC driver setup for more information.

4.5 Mapping multiple source items, to single target items

In this section two simple employee travel expense reports will be mapped to a single company report. This example is a simplified version of the mapping you have already worked through in the Multiple target schemas / documents section of this tutorial.

Please note: There is an alternative method to doing this using the dynamic input/output functionality of components, please see "Dynamic file names - input / output" for a specific example.

Objective

In this section of the tutorial, you will learn how to merge two **personal travel expense reports** into a company expense travel report. Specifically, you will learn how to:

- Map schema components (recapitulation)
- Duplicate input items
- Remove duplicated items

Commands used in this section

New...: Click this icon to access the **New File** dialog box where you can create a new Mapping.

Insert XML Schema/File: Click this icon to open the standard Windows **Open** dialog box and select the file from your file system.

Auto Connect Matching Children: Click this icon to toggle the automatic connection of matching child nodes, on and off.

Duplicate Input: Located in the context menu that appears when you right-click an item in a component. Click this command to duplicate the selected item.

Remove Duplicate: Located in the context menu that appears when you right-click a duplicated item in a component. Click this command to remove the selected duplicate from the component.

Example files used in this section

Please note that the files used in this example, have been optimized to show how to map data from two input XML files into a single item in the target schema, this is not meant to be a real-life example.

• mf-ExpReport.xml	Input XML file used in previous section
• mf-ExpReport2.xml	The second input XML file
• mf-ExpReport-combined.xml	The resulting file when the mapping has been successful
• ExpReport-combined.xsd	The target schema file into which the two XML source data will be merged.
• Tut-ExpReport-msource.mfd	The mapping file for this example

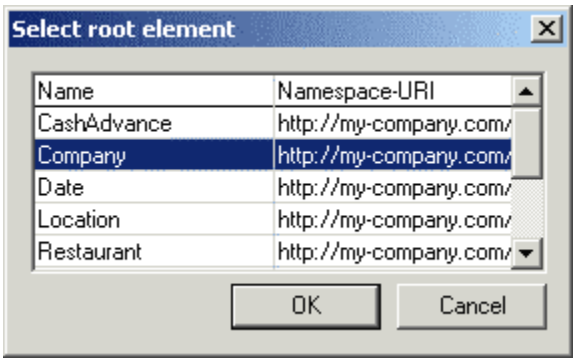
Please note: The files used in this section are also available in the ...\MapForceExamples\Tutorial\ folder.

4.5.1 Creating the mappings

The method described below, is a recapitulation of how to set up the mapping environment. This mapping is available as **Tut-ExpReport-msource.mfd** in the ...\MapForceExamples\Tutorial\.

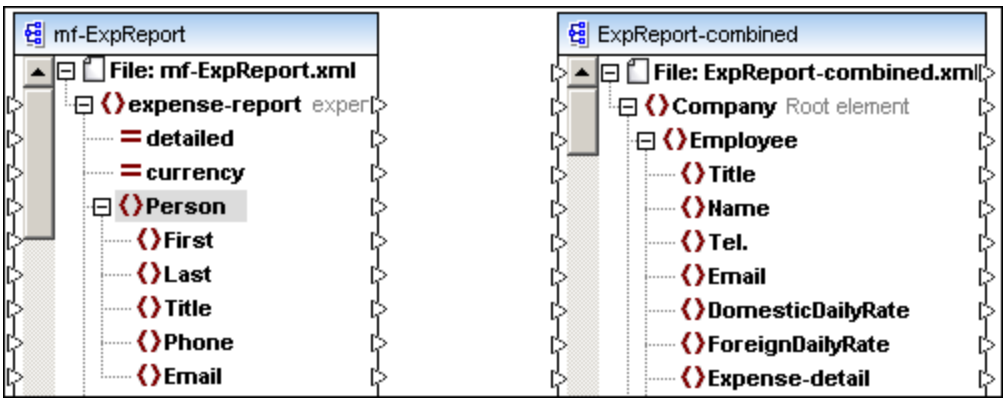
To create the mapping environment:

1. Click the **New** icon in the Standard toolbar to open the **New File** dialog box.
2. Click the **Mapping** icon and click **OK** to create a new Mapping tab.
3. Click the **Insert XML Schema/File** icon.
4. From the Tutorial sub-folder of the MapForceExamples directory, select the **mf-ExpReport.xsd** file from the **Open** dialog box, click **Browse...** in the message box that pops up, and select the **mf-ExpReport.xml** file as the XML instance file.
5. Click the **expense-report** entry, hit the ***** key on the numeric keypad to view all the items; resize the component if necessary.
6. Click the **Insert XML Schema/File** icon.
7. Select the **ExpReport-combined.xsd** file from the **Open** dialog box. You are now prompted for a sample XML file for this schema.
8. Click **Skip**, and select **Company** as the root element of the target document.



The target schema component now appears in the mapping pane.

9. Click the **Company** entry, hit the ***** key on the numeric keypad to view all the items, and resize the window if necessary.

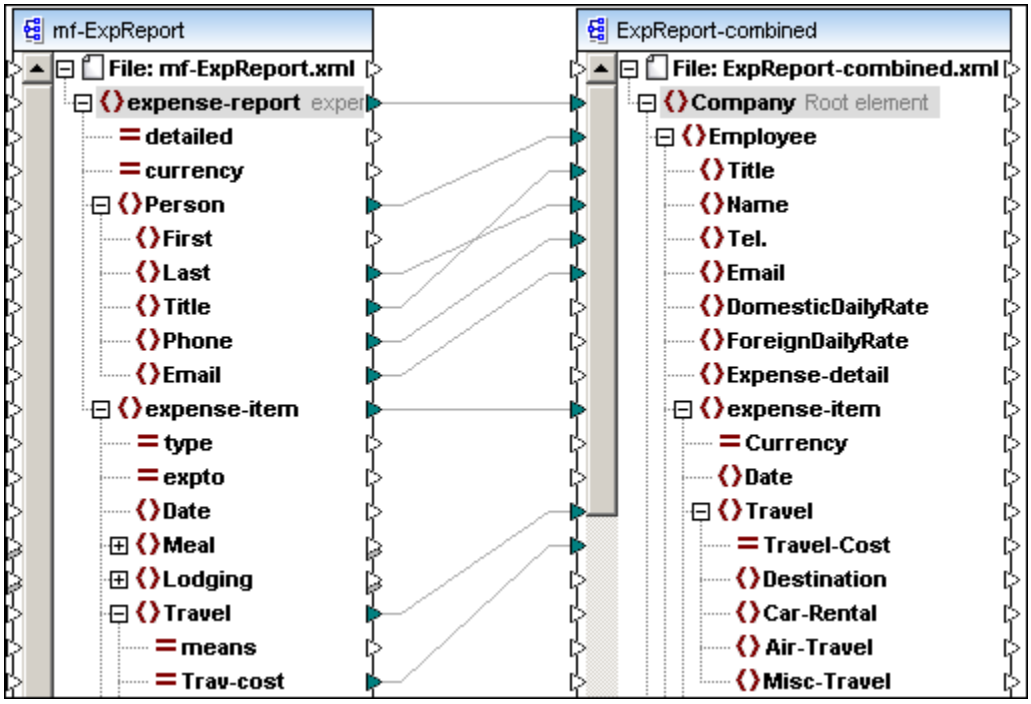


Mapping the components

Make sure that the **Auto connect child items** icon is **de**activated, before you create the following mappings between the two components:

- Expense-report to Company
- Person to Employee
- Last to Name
- Title to Title
- Phone to Tel.
- Email to Email
- expense-item to expense-item
- Travel to Travel
- Trav-cost to Travel-Cost

The mapping is shown below.



Click the Output button to see the result of the current mapping.

Please note: **Empty** <expense-item/> elements/tags are generated when child items of a **mapped parent item**, exist in the source file, which have not been mapped to the target schema. In this case, only the travel items of the expense-item parent have been mapped. There are however, two other expense items in the list: one lodging and one meal expense item. Each one of these items generates an empty parent expense-item tag.

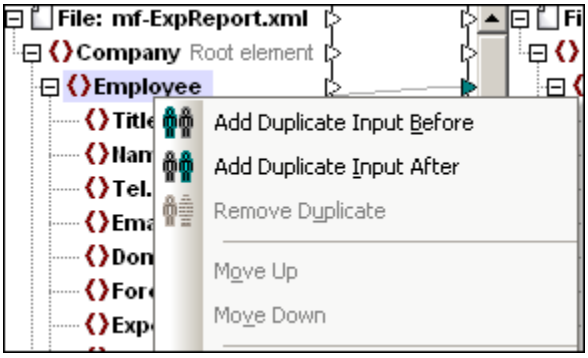
To avoid generating empty tags, create a filter such as the one described previously in the tutorial, under Filtering data, or connect the **Travel** item to the **expense-item**.

4.5.2 Duplicating input items

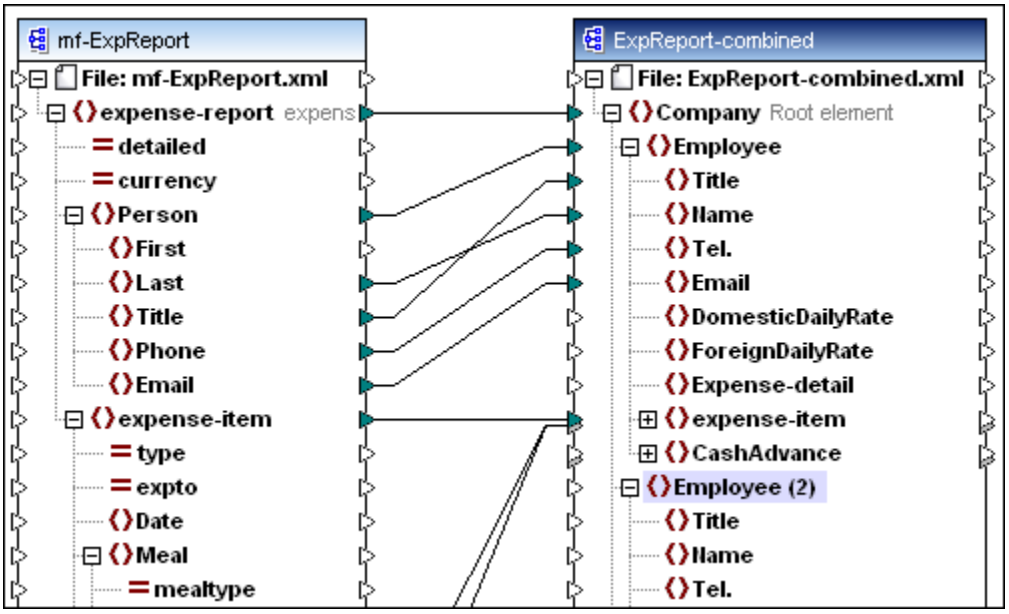
In order to map multiple source items to one and the same target item, we need to duplicate the **input items** of the target component to be able to create mappings from a different source XML file. To achieve this we will add the **second** XML source file, and create mappings from it, to the "same" inputs of the duplicated element/item in the target XML file.

Duplicating input items:

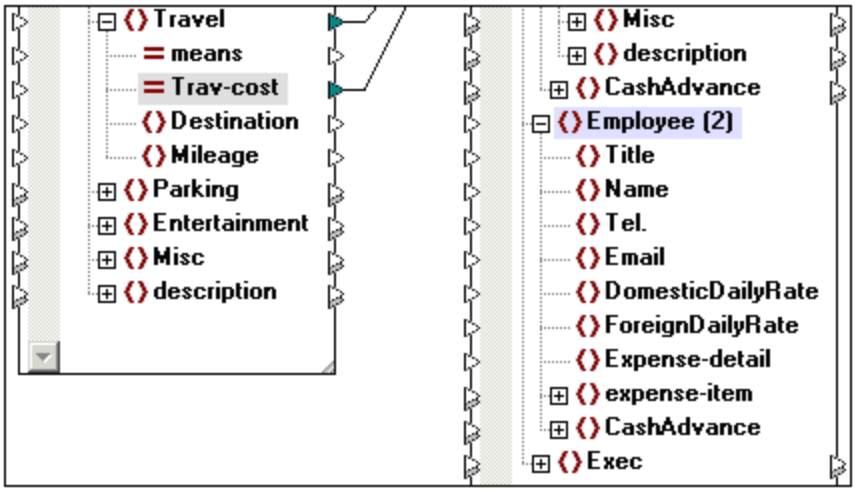
1. Right-click the Employee item in the target XML file.
2. Select the option **Add Duplicate Input After** from the context menu.



A second Employee item has now been added to the component, as **Employee(2)**.



3. Click the expand icon to see the items below it.
   The **structure** of the new Employee item, is an exact copy of the original, except for the fact that there are no output icons for the duplicated items.



You can now use these new duplicate items as the **target** for the **second** source XML data file.

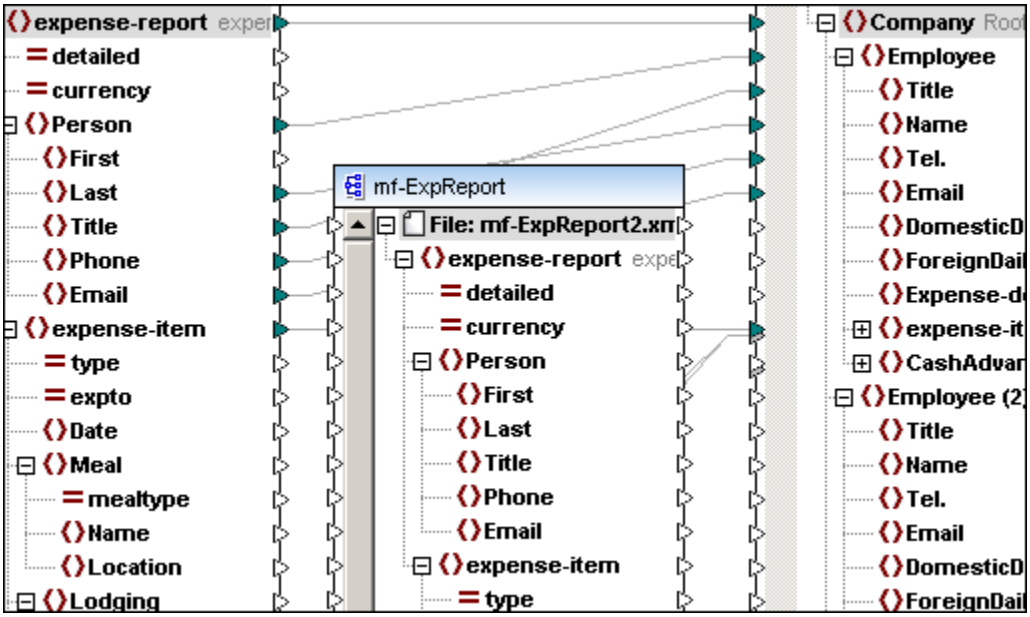
Inserting the second XML instance file

To insert the second XML instance file, the same method as well as the same XML Schema file is used as before.

To insert a second source component:

1. Click the **Insert XML Schema/File** icon.
2. Select the **mf-ExpReport.xsd** file from the **Open** dialog box, click **Browse...**, and select the **mf-ExpReport2.xml** file as the XML instance file.
3. Click the **expense-report** entry, hit the ***** key on the numeric keypad to view all items, and resize the component if necessary.

   For the sake of clarity, the new component has been placed between the two existing ones in the following graphics.

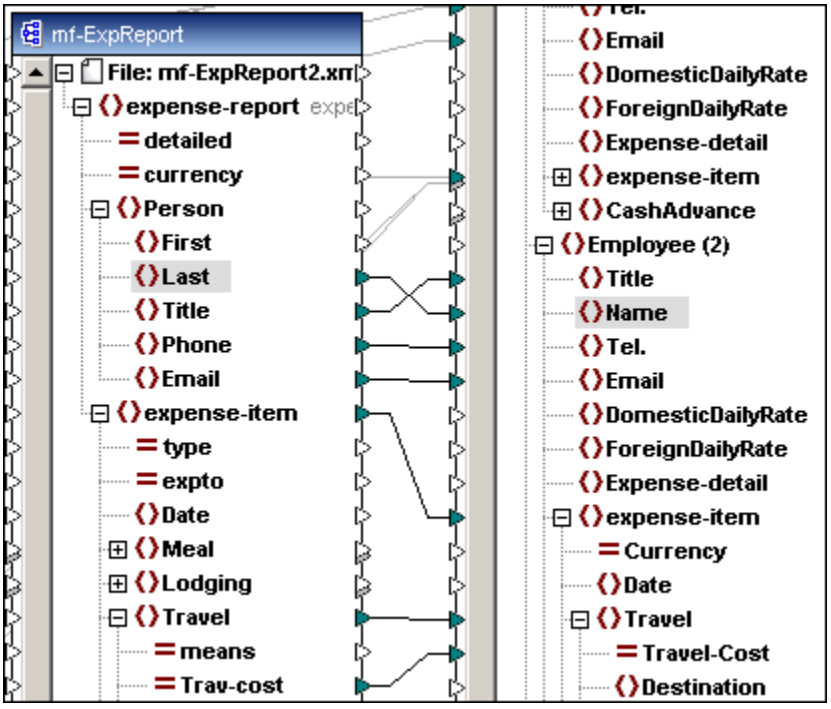


4. Create the same mappings that were defined for the first XML source file:
   - Person to Employee(2)

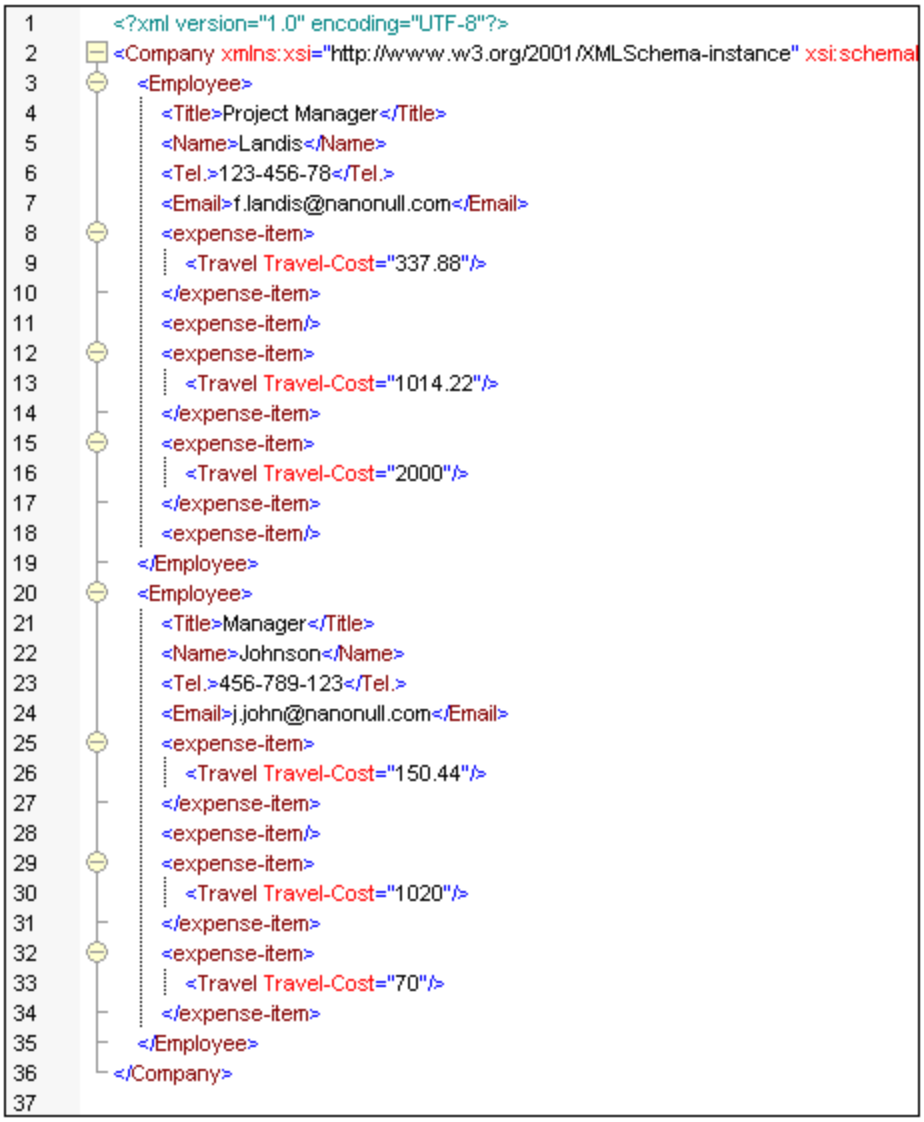
- Last to Name
- Title to Title
- Phone to Tel.
- Email to Email
- expense-item to expense-item

Scroll down, and map

- Travel to Travel, and
- Trav-cost to Travel-Cost.



5. Click the Output button to see the result of the mapping in the Output pane.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemal
  <Employee>
    <Title>Project Manager</Title>
    <Name>Landis</Name>
    <Tel.>123-456-78</Tel.>
    <Email>f.landis@nanonull.com</Email>
    <expense-item>
      <Travel Travel-Cost="337.88"/>
    </expense-item>
    <expense-item/>
    <expense-item>
      <Travel Travel-Cost="1014.22"/>
    </expense-item>
    <expense-item>
      <Travel Travel-Cost="2000"/>
    </expense-item>
    <expense-item/>
  </Employee>
  <Employee>
    <Title>Manager</Title>
    <Name>Johnson</Name>
    <Tel.>456-789-123</Tel.>
    <Email>j.john@nanonull.com</Email>
    <expense-item>
      <Travel Travel-Cost="150.44"/>
    </expense-item>
    <expense-item/>
    <expense-item>
      <Travel Travel-Cost="1020"/>
    </expense-item>
    <expense-item>
      <Travel Travel-Cost="70"/>
    </expense-item>
  </Employee>
</Company>
```

The data of the second expense report has been added to the output file. Johnson and his travel costs have been added to the expense items of Fred Landis in the company expense report.

To save the generated output to a file:

- Click the **Save** icon which appears in the title bar when the Output pane is active.

  The file, mf-ExpReport-combined.xml, is available in the ...\MapForceExamples\Tutorial\ folder.

To remove duplicated items:

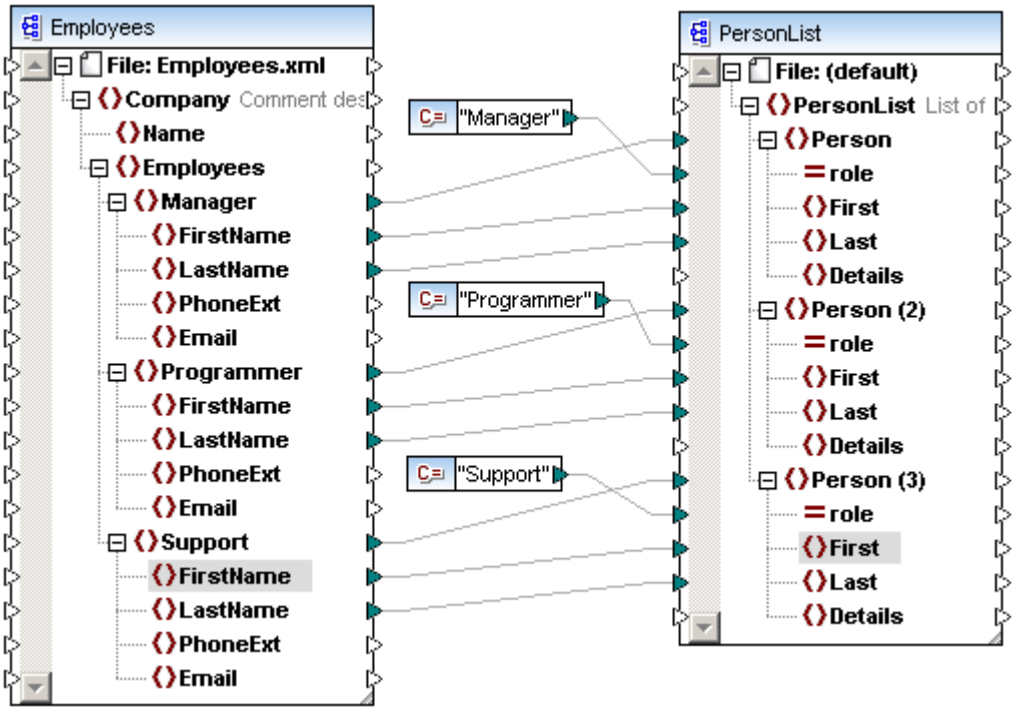
- Right click the duplicate item and select the **Remove Duplicate** entry from the menu.

Example

To see a further example involving duplicate items, please see the **PersonList.mfd** sample file available in the **...\MapForceExamples** folder.

In the **PersonList.mfd** example different elements of the source document are mapped to the

"same" element in the target Schema/XML document, ans specific elements (Manager etc.) are mapped to a generic one using a "role" attribute.




```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <PersonList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="
   C:/.../Altova/MapForce2011/MapForceExamples/PersonList.xsd">
3    <Person role="Manager">
4      <First>Vernon</First>
5      <Last>Callaby</Last>
6    </Person>
7    <Person role="Programmer">
8      <First>Frank</First>
9      <Last>Further</Last>
10   </Person>
11   <Person role="Support">
12     <First>Loby</First>
13     <Last>Matise</Last>
14   </Person>
15   <Person role="Support">
16     <First>Susi</First>
17     <Last>Sanna</Last>
18   </Person>
19 </PersonList>
```

4.6 Multi-file input / output

In this section the new multi-file input/output capabilities of MapForce will be demonstrated. Please note that this functionality is not available for XSLT 1.0 and XQuery.

A single input component will process two source documents, while a single output component will generate two output files. The example used here has been set up in Filtering data, and also been used as the basis in the Mapping multiple source items, to single target items section.

Objective

In this section of the tutorial, you will learn how to create a mapping where the source component processes two XML input files and the target component outputs two XML target files.

Commands used in this section

Save All Output Files...: Located in the **Output** menu. Click this command to save all the mapped files from the Preview pane.

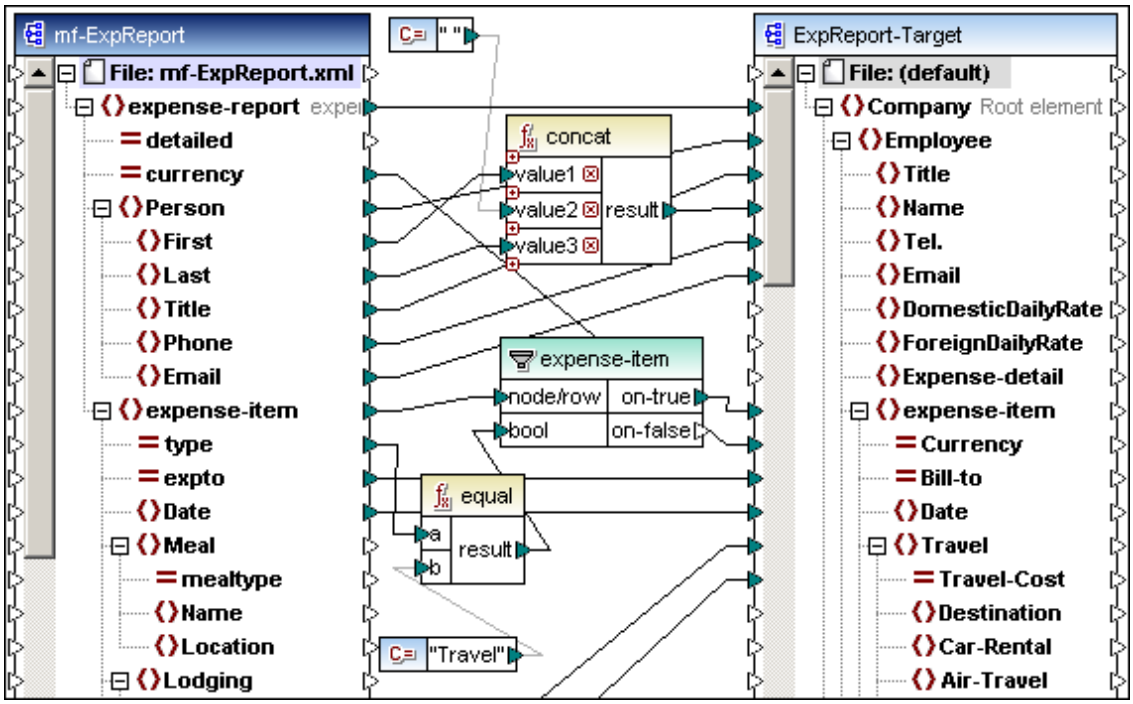
Example files used in this section

- mf-ExpReport.xml — Input XML file used in previous section
- mf-ExpReport2.xml — The second input XML file
- Tut-ExpReport-multi.mfd — The mapping file for this example

Please note: The files used in this section are also available in the ...\MapForceExamples\Tutorial\ folder.

4.6.1 Processing multiple files per input/output component

The **Tut-ExpReport.mfd** file available in the ...\MapForceExamples folder will be modified and saved under a different name in this example.



Please take note of the following items at the top of each component:

- The **File:mf-ExpReport.xml** item of mf-ExpReport, displays the Input/Output-XML file entry. One entry is shown if Input and Output files are the same; if not then **Input file name;Output file name** is displayed.

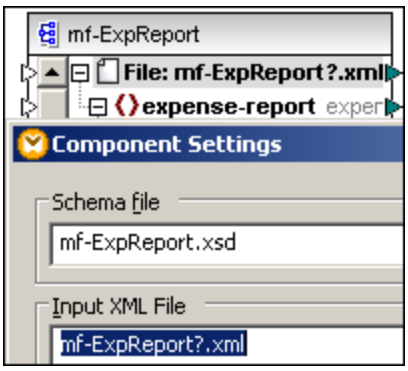
  This is automatically filled when you assign an XML instance file to an XML schema file.

- The **File: (default)** item of ExpReport-Target shows that an instance file was not assigned to the XML schema component when it was inserted, i.e. the Output-XML file field is empty. A default value will therefore be used when the mapping executes.

Processing multiple files
To be able to process multiple files, MapForce uses the wildcard character "?" in the filename of the input XML file. The "?" can be replaced by none, or one character.

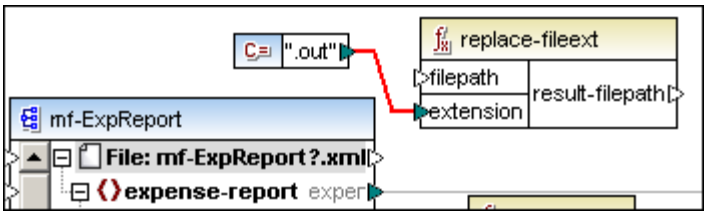
To process multiple files:
Having opened the Tut-ExpReport file available in the ...\Tutorial folder and having selected XSLT 2.0:

1. Double click the **mf-ExpReport** component on the left.
2. Enter mf-expReport**?**.xml in the Input XML File field.

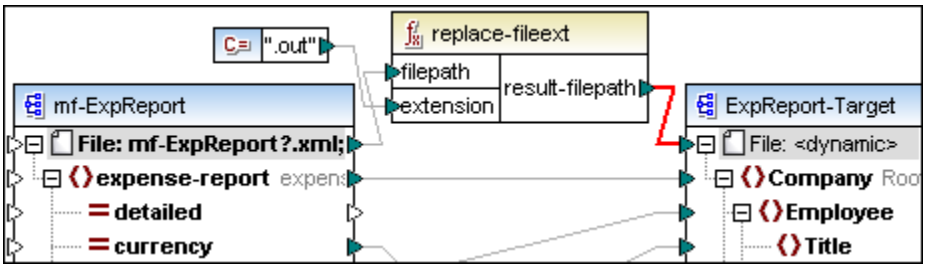


The wildcard characters **?** and ***** are supported in file names. Note that a relative path was entered here, as the Tut-ExpReport.mfd file is available in the ...\Tutorial folder (you can enter an absolute path if you want).

3. Insert the **replace-fileext** function from the **file path functions** library, then insert a constant component.
4. Enter ".out" into the constant component, and connect it to the **extension** parameter of the function.

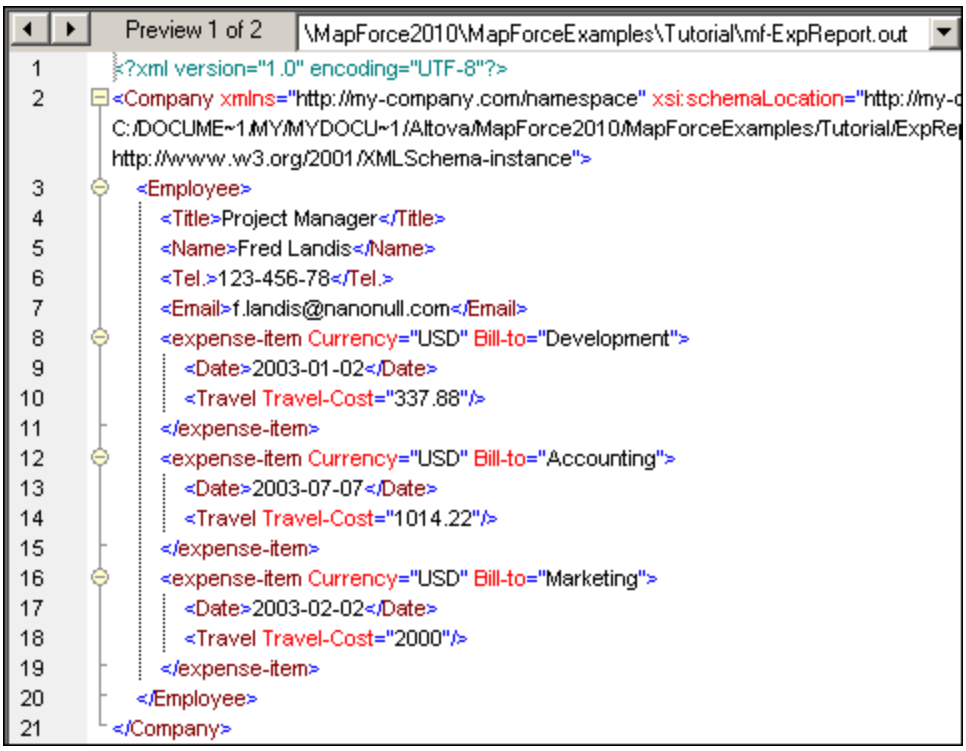


5. Connect the **File**:mf-ExpReport?.xml item of the component to the **filepath** parameter of the function.
6. Connect the **result-filepath** parameter of the function, to the File <dynamic> item of the target component.

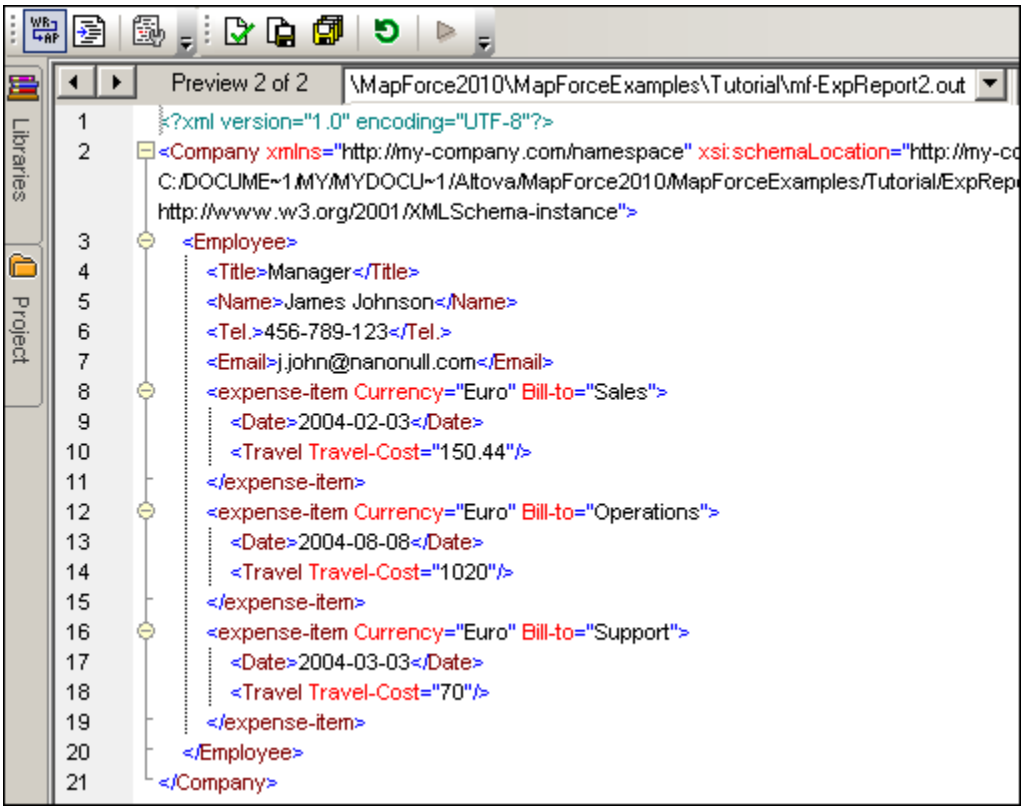


The File: item of the target component has also changed to **File: <dynamic>**.

7. Click the Output button to see the results.
   The Output window now shows the results for each input XML file in the preview window, e.g. Preview 1 of 2 as shown below.



8. Click the scroll arrow to show the result of the second input XML file.
Note that the combo box shows the name of each of the source XML files; with the *.**xml** extension replaced by the *.**out** extension.



Clicking the **Save All**  icon lets you save all the mapped files from the Preview window without having to generate code. A prompt appears if output files at the same location will be overwritten.

9. Save the mapping file under a new name.

Note: please see Dynamic input/output for more information on multiple input / output files.

4.7 Database to schema mapping

This section will show you how to use a simple Microsoft Access database as a data source, to map database data to a schema.

Objective

In this section, you will learn how to insert a database into a MapForce mapping and how you can map the tables of this database to the target schema component. Specifically, you will learn how to:

- Connect to a Microsoft Access database
- Map database tables to XML Schema nodes

Commands used in this section

New...: Click this icon to access the **New File** dialog box where you can create a new Mapping.

Insert Database: Click this command to open the **Select a Database** dialog box where you can create a connection to the database.

Insert XML Schema/File: Click this icon to open the standard Windows **Open** dialog box and select the file from your file system.

Auto Connect Matching Children: Click this icon to toggle the automatic connection of matching child nodes, on and off.

Generate code in: Located in the **File** menu. Clicking this command opens a submenu from where you can choose the desired type of code (i.e. **Java, C++** or **C#**). The command pops up the **Browse For Folder** dialog box where you define where the generated code should be saved.

Currently supported databases (and connection types) are:

- Microsoft Access versions 2003 / 2007
- Microsoft SQL Server versions 2000, 2005 and 2008
- Oracle version version 9i, 10g, and 11g
- MySQL 4.x and 5.x
- PostgreSQL 8.0, 8.1, 8.2, 8.3
- Sybase 12
- IBM DB2 version 8.x and 9
- IBM DB2 for i 5.4
- IBM DB2 for i 6.1

Please note:

MapForce fully supports the databases listed above. While Altova endeavors to support other ODBC/ADO databases, successful connection and data processing have only been tested with the listed databases.

When installing the **64-bit** version of **MapForce**, please make sure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

Note:
For an ADO connection to MS SQL Server via the driver SQL Server Native Client 10.0 the

following property values must be set in the *All* tab of the Data Link Properties dialog: (i) Set the property value of Integrated Security to a space character; (ii) Set the property value of Persist Security Info to `true`.

The table below shows the type of database created, the restrictions, and the connecting methods, when inserting databases.

	Insert Database connection methods	
Supported database	**ODBC restrictions** (unique keys are not supported by ODBC)	**ADO restrictions**
Microsoft Access (ADO)	OK Primary and Foreign keys are not supported.	OK *
MS SQL Server (ADO)	OK	OK *
Oracle (ODBC)	OK, restrictions: table containing columns of type CLOB, BLOB, BFILE; XML tables	OK, restrictions: table containing columns of type CLOB, BLOB, BFILE; XML tables; owner information, Identity constraints are not read from the database
MySQL (ODBC)	OK *	OK ⌖
Sybase (ODBC)	OK *	OK
IBM DB2 (ODBC)	OK *	OK

* **Recommended connection method for each database.**

⌖ **MySQL: When creating the ADO connection based on ODBC, it is recommended to use either the User or System DSN**.

\- **Not available**

Please note:

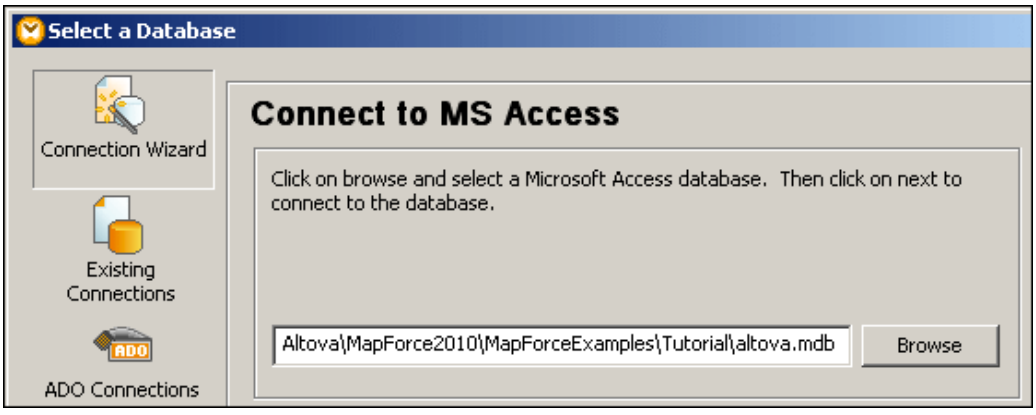
Databases can be also be inserted as "Global Resources", please see Global Resources for more information.

4.7.1 Connecting to an Access database

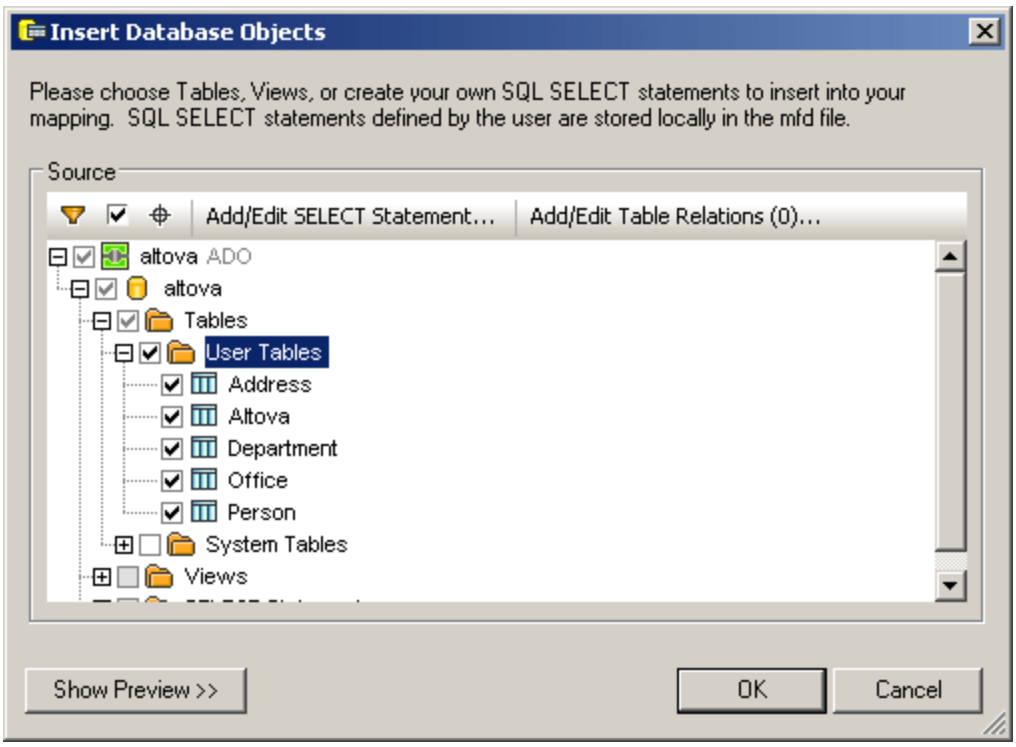
Creating the database component in MapForce:

1. Select **File | New** in MapForce to create a new mapping.
2. Click one of the icons in the toolbar: Java, C#, C++, or BUILTIN.
3. Click the **Insert Database** icon in the icon bar.
4. In the **Select a Database** dialog box, click the **Connection Wizard** button

5. Click the **Microsoft Access** radio button.
6. Click the **Next** button to continue.
7. Click the **Browse** button to select the database you want as the data source, **altova.mdb** in the ...\MapForceExamples\Tutorial\ folder in this case.
   The connection string appears in the text box.



8. Click the **Connect** button.



9. In the **Insert Database Objects** dialog box, click the check box to the **left** of "User Tables", then click the **OK** button to insert the database (schema) component. This selects all the tables of the folder.

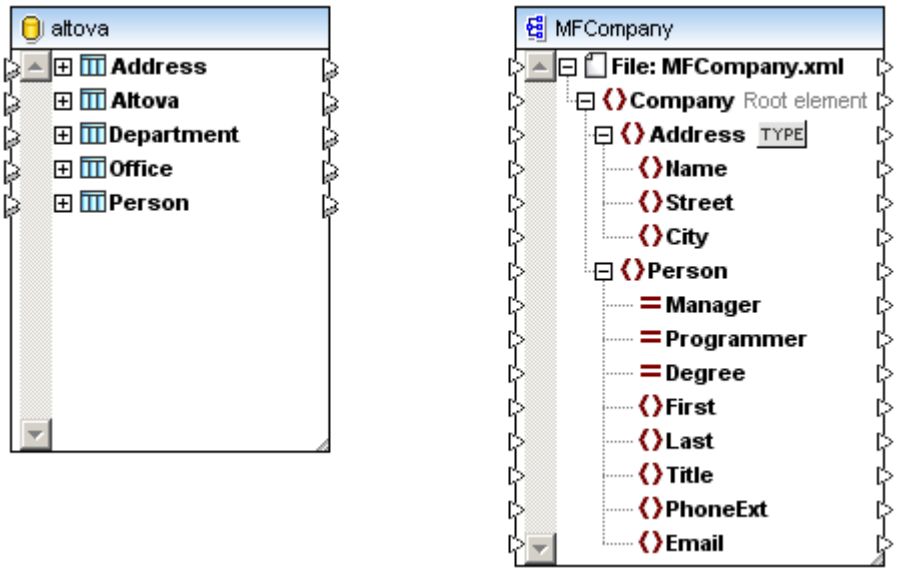
The database component appears in the mapping window. You can now create mappings to a target schema / XML document.

4.7.2 Mapping database data

Now that you have inserted the database component into the mapping window, you can add the target schema component and create the mappings between them.

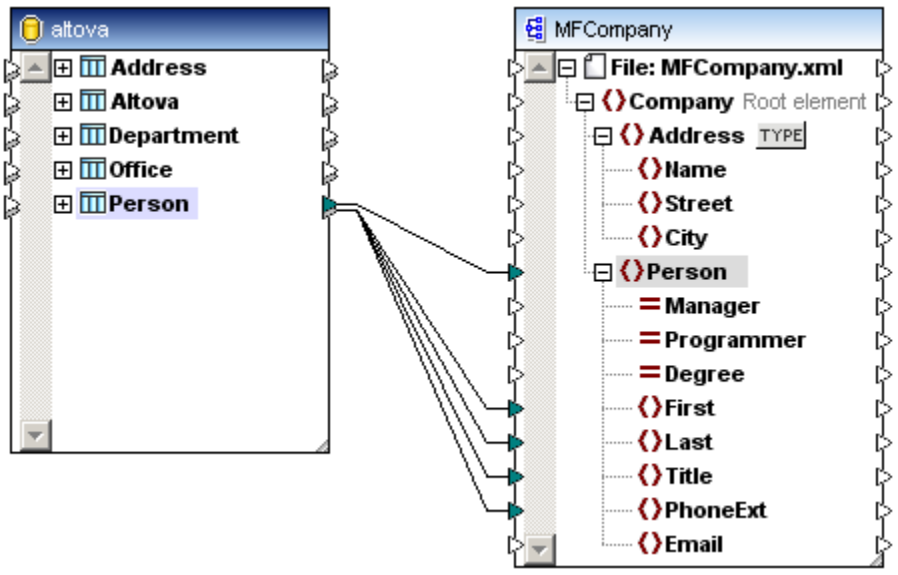
Inserting the target schema /document:

1. Click the **Insert XML Schema/File** icon, and select the **MFCompany.xsd** schema.
2. Click **Skip** when the prompt for a sample XML file appears.
3. Select **Company** as the root element and expand all items.
   You are now ready to map the database data to a schema / XML document.



Mapping database data to a schema/document in MapForce:

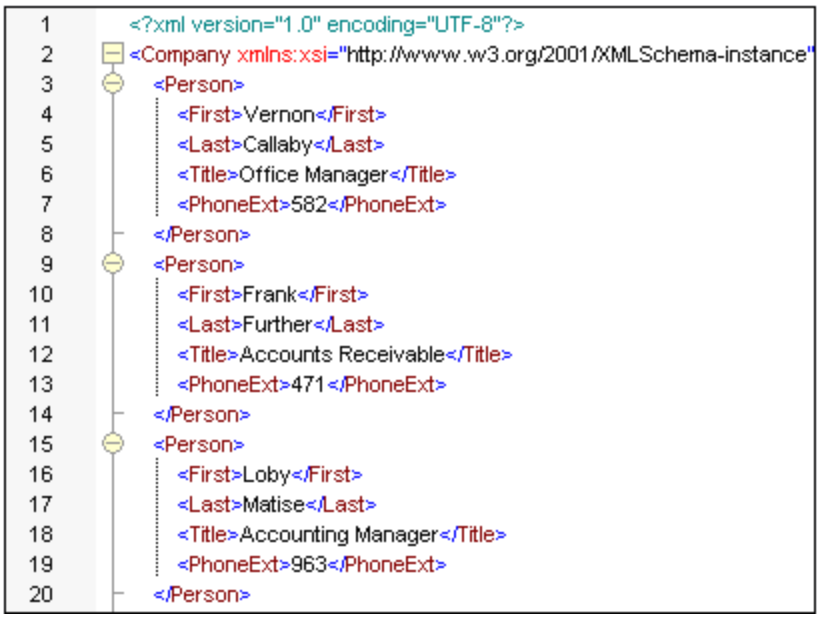
1. Activate the **Auto connect child items** icon, if not already active.
2. Click the **Person** "table" item in the database component, and connect it to the Person item in MFCompany.
   This creates connectors for all items of the same name in both components.



4. Save the MapForce file, **PersonDB** for example.
5. Click the Output button to see the result/preview of this mapping. The Built-in execution engine generates results on-the-fly without you having to generate or compile code.

Generating Java code and the resulting XML file:

1. Select the menu option **File | Generate code in | Java**.
2. Select the directory you want to place the Java files in, and click OK.
   The "Java Code generation completed" message appears when successful.
3. Compile the generated code and execute it.
   The following MFCompany.xml file is created.



```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    <Person>
4      <First>Vernon</First>
5      <Last>Callaby</Last>
6      <Title>Office Manager</Title>
7      <PhoneExt>582</PhoneExt>
8    </Person>
9    <Person>
10     <First>Frank</First>
11     <Last>Further</Last>
12     <Title>Accounts Receivable</Title>
13     <PhoneExt>471</PhoneExt>
14   </Person>
15   <Person>
16     <First>Loby</First>
17     <Last>Matise</Last>
18     <Title>Accounting Manager</Title>
19     <PhoneExt>963</PhoneExt>
20   </Person>
```

For more complex examples of database to schema mapping using:

- multiple source files
- flat and hierarchical databases

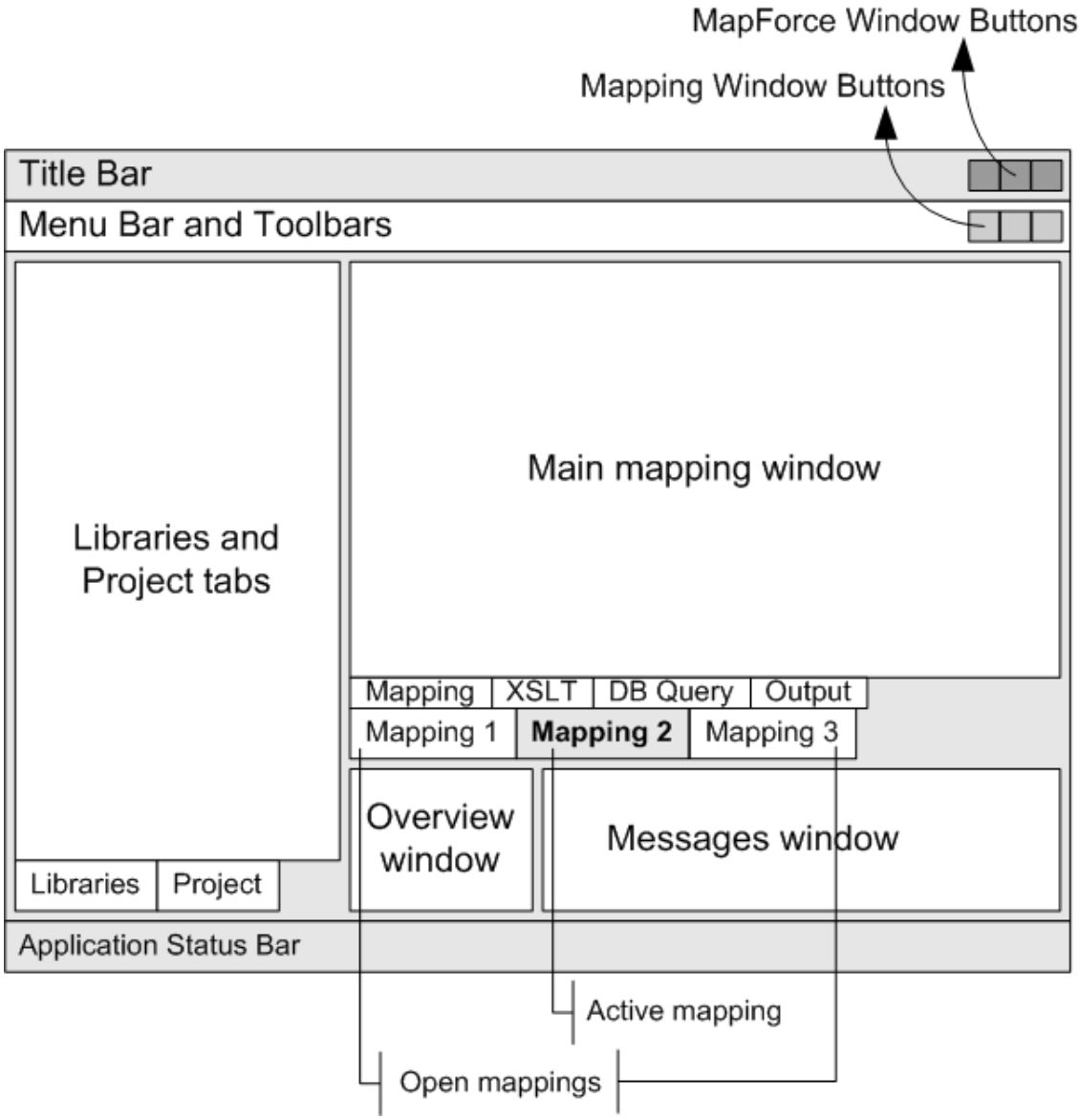
Please see the **DB_Altova_SQLXML.mfd** and **DB_Altova_Hierarchical.mfd** files in the **...\MapForceExamples** folder of MapForce.

Chapter 5

MapForce user interface

5 MapForce user interface

MapForce has four main areas: the Libraries and Project pane at left, the Mapping window (with Mapping, XSLT, XSLT2, XQuery, DBQuery, and Output panes) at right, as well as the Overview and Messages windows below.



Title Bar

The Title Bar displays the application name (i.e., MapForce) followed by the name of the active Mapping Design window. Buttons to control the MapForce application window are at right.

Menu Bar and Toolbars

The Menu Bar displays the menu items. Each toolbar displays a group of icons representing MapForce commands. You can reposition the menu bar and toolbars by dragging their handles

to the desired locations.

Libraries and Project Tabs

The **Libraries** tab provides functions that vary according to the selected output language. You can drag a function into a mapping window, and add user-defined functions.

The **Project** tab shows all files and folders that are relevant to the particular Project, or Web Service, respectively. You can add folders and files to the project and define default project settings via the **Project** menu.

Mapping Window

The **Mapping** window displays the graphical elements used to create the mapping (transformation) between the various components. The source schema displays the source schema tree and the target schema displays the target schema tree. **Connectors** connect the input and output **icons** of each schema item. Schema **items** can be either elements or attributes.

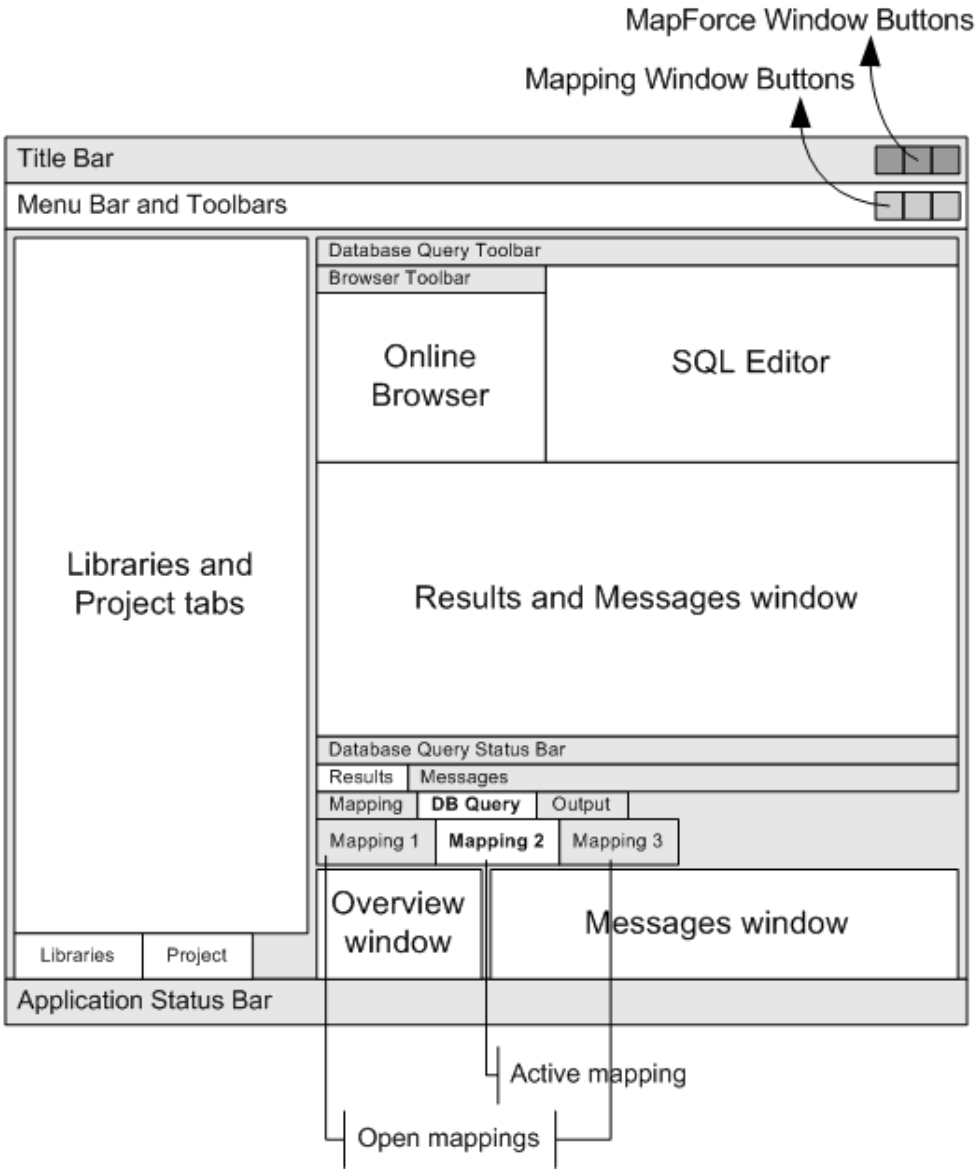
The following panes can be viewed by clicking the corresponding button at the bottom of the Mapping window:

- The **XSLT** , **XSLT2**, and **XQuery** panes display a preview of the transformation depending on the specific language selected.
- The **DB Query** pane allows you to directly query any major database. The queries/ actions defined here are independent of any of the other MapForce tabs, and are not saved as part of the *.MFD file.

  Each Database Query pane is associated with the currently active mapping, allowing **multiple database queries** per session/mapping. Note that you can also have multiple active connections, to different databases, for each Database Query pane.
- The **Output** pane displays a preview of the transformed, or mapped data, in a text view.
- The **HTML, RTF, PDF, and Word 2007+** panes display the target component data as HTML, RTF, PDF, or Word 2007+ documents if a StyleVision Power Stylesheet (SPS) is associated with the target component.

Database Query Window

If you click the **DB Query** button, additional windows, toolbars, and a status bar for connecting to, and working with, databases are displayed (*illustration below*).



The **DB Query** window allows you to directly query any major database. The queries/actions defined here are independent of any of the other MapForce tabs, and are not saved as part of the *.MFD file.

The Database Query window is divided into the following parts:

- The **Online Browser** displays connection info and database tables and gives a full overview of the objects in each database, including database constraint information, e. g. column as a primary or foreign key. In IBM DB2 version 9 databases, the Browser additionally shows registered XML schemas in a separate folder.
- The **SQL Editor** is used to write and execute SQL statements and provides features such as autogeneration and autocompletion of SQL statements, definition of regions, or insertion of line and block comments.
- The Results and Messages window provides the **Results** tab which displays the query results in tabular form, and the **Messages** tab which displays warnings or error messages.

Each Database Query window is associated with the currently active mapping, allowing **multiple database queries** per session/mapping. Note that you can also have multiple active connections, to different databases, for each Database Query tab.

Database Query Status Bar

The Database Query status bar at the bottom of the Results window displays information on the progress of the query: whether the retrieval executed successfully, was aborted or has been stopped by the user. In addition, the number of rows and columns retrieved as well as the amount of time necessary for retrieval, and the time when the query was executed.

Overview and Messages Windows

The **Overview** pane displays the mapping area as a red rectangle, which you can drag to navigate your Mapping.

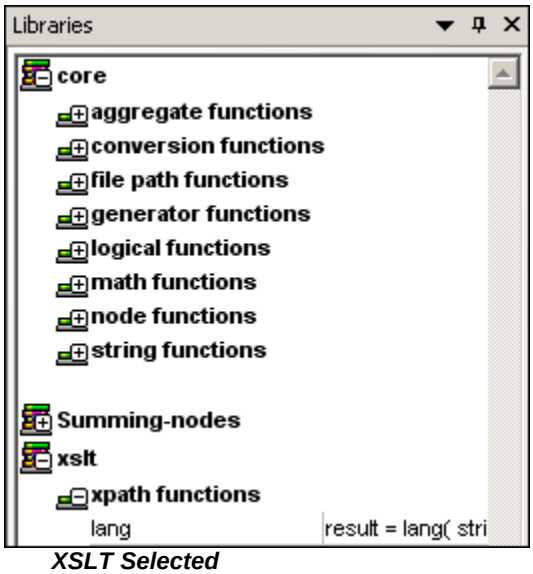
The **Messages** pane displays any validation warnings or error messages that might occur during the mapping process. Clicking a message in this pane, highlights it in the Mapping tab for you to correct.

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 mouseover a toolbar icon. The application status bar should not be confused with the Database Query status bar. If you are using the 64-bit version of MapForce, this is indicated in the status bar with the suffix (x64) after the application name. There is no suffix for the 32-bit version.

5.1 Libraries tab

The **Libraries** tab displays the available libraries for the currently selected programming language, as well as the individual **functions** of each library. A brief description of the function is also provided. Functions can be directly dragged into the **Mapping** tab. Once you do this, they become function components.



XSLT Selected

The standard **core, lang, xpath2** and **xslt** libraries are always loaded when you start MapForce, and do not need to be added by the user. The **Core** library is a collection of functions that can be used to produce all types of output: XSLT, XQuery, Java, C#, C++,. The other libraries (XSLT, XSLT2, XPath2, Lang etc.) contain functions associated with each separate type of output.

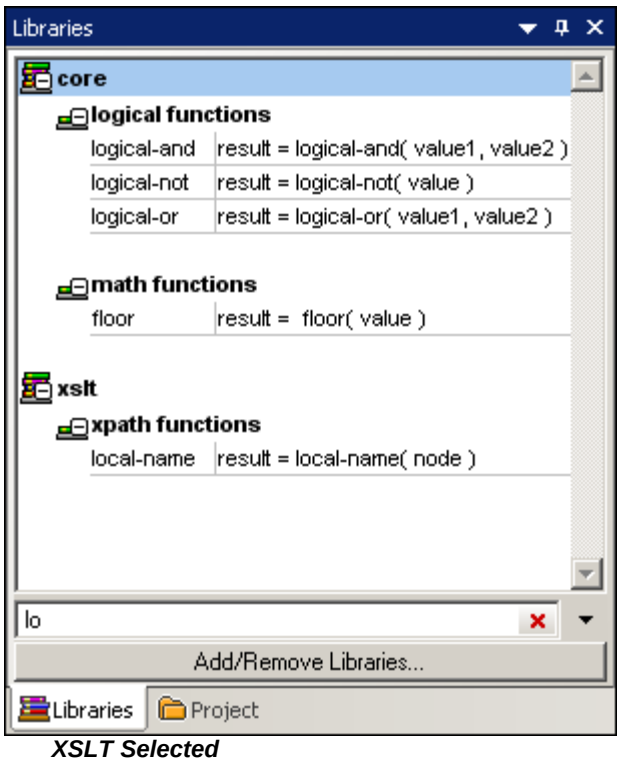
Please note: The XPath 2.0 library and its functions, are common to both XSLT 2.0 and XQuery languages. If you clicked the **Built-in Execution Engine** icon, the same functions will be available as for Java, C#, and C++.

Selecting	enables
XSLT	core and XSLT functions (XPath 1.0 and XSLT 1.0 functions)
XSLT2	core, XPath 2.0, and XSLT 2.0 functions
XQ(uery)	core and XPath 2.0 functions

XPath 2.0 restrictions: Several XPath 2.0 functions dealing with sequences are currently not available.

Finding functions in the Library window

A Find field is located at the bottom of the Libraries tab which allows you to search for function names.

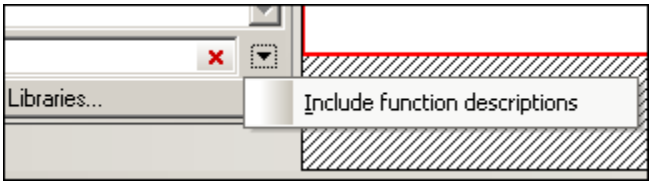


XSLT Selected

Pressing the **Esc** key cancels the filtering function in the window. Clicking the "**x**" icon has the same effect.

To find a function in the Library window:

1. Click into the Libraries window to make it active and enter the characters you are looking for, e.g. "lo".
   All functions containing these characters are now shown in the Library window, each within its respective group.
2. Click the down-arrow and select "Include function descriptions", if you want to include the text of the function descriptions in the function search.



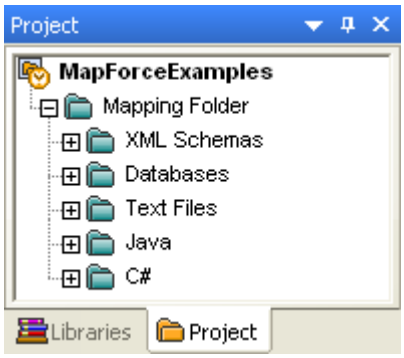
Adding new function libraries
MapForce allows you to create and integrate your own function libraries, please see the sections: Adding custom function libraries, Adding custom XSLT 1.0 functions, Adding custom XSLT 2.0 functions and User-defined functions for more information.

Please note: Custom functions/libraries can be defined for Java, C#, and C++, as well as for XSLT and XSLT 2.

5.2 Project tab

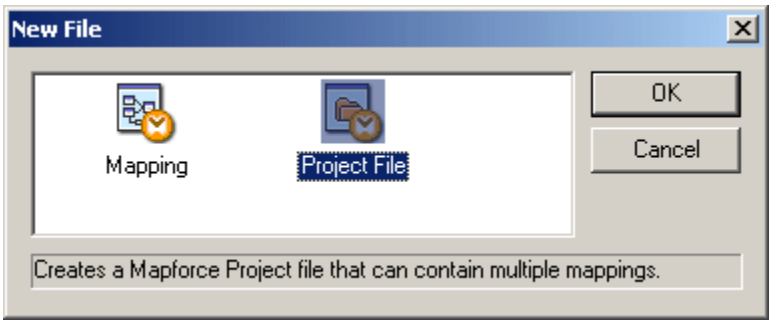
MapForce supports the Multiple Document Interface, and allows you to group your mappings into mapping projects. Project files have a *.mfp extension.

The Project tab shows all files and folders that have been added to the project directory. The project directory is the folder in which the project file is located.

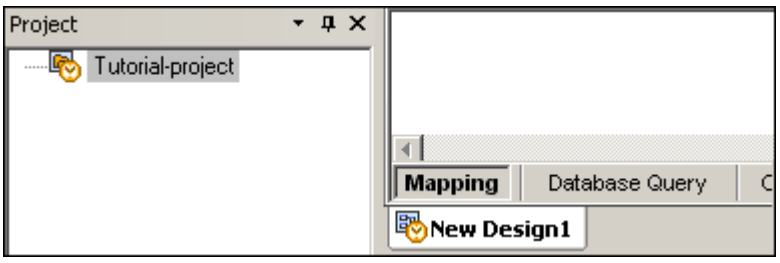


To create a project:

1. Select **File | New** and double-click the **Project File** icon.



2. Enter the project name in the **Save Project As** dialog box, and click **Save** to continue. A project folder is added to the Project tab.



3. Select **File | New** and double click the **Mapping** icon. This opens a new mapping file, "New Design1", in the Mapping pane.

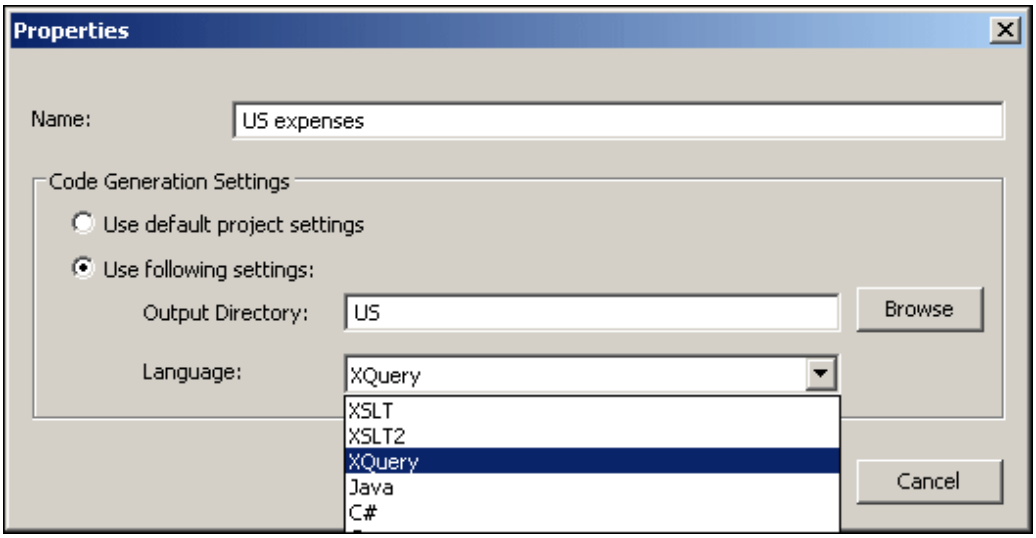
Project options

You can add files to the project after it has been created, and you can define a folder structure within the project. The respective commands are available in the **Project** menu or in the context menu that appears when you right-click the project name in the Project tab.

The **Add Files to Project...** option opens the standard Windows **Open** dialog box where you can browse for MapForce Mappings (*.mfd files) that you want to add to the project.

The **Add Active File to Project** command adds the mapping that is currently active in the Mapping window to the project. A warning message is displayed if the file has already been added to the project.

The **Create Folder** option opens the **Properties** dialog box where you can specify the folder name as well as decide whether you want to use the default project settings, or define custom settings for output directory and language. When you click **OK**, a sub-folder, of the currently selected folder, is created.



Selecting the option **Project | Add files to project**, allows you to add files that are not currently opened in MapForce.

To add mappings to a project:

- Select **Project | Add active file to project**.
  This adds the currently active file to the project. The mapping name now appears below the project name in the project tab.

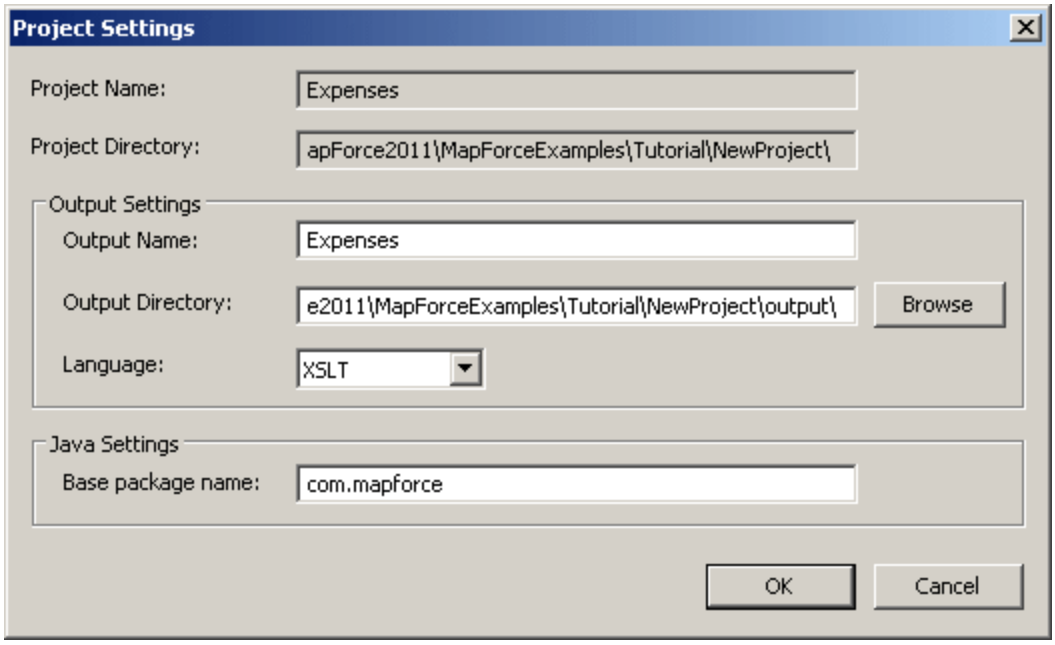
To remove a mapping from a project:

Do one of the following:

- Right-click the mapping icon below the project folder and select **Delete** from the pop-up menu.
- Select a mapping in the Project folder and hit the **Del** key.

Project settings

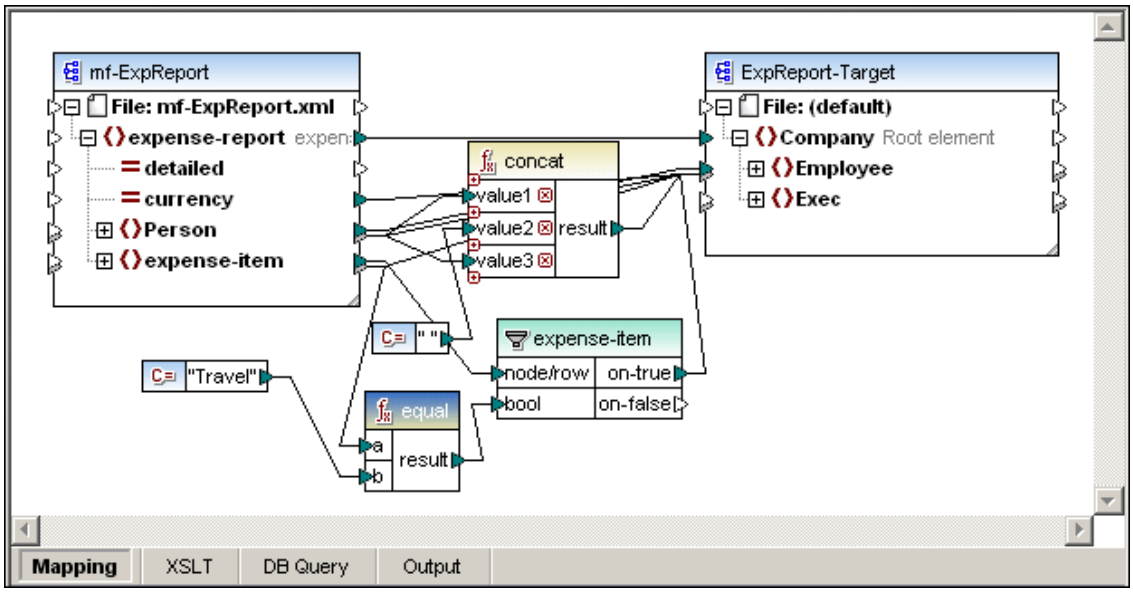
The **Project Settings** dialog box can be accessed by right-clicking the project name in the Project tab and choosing **Properties** from the context menu, or by selecting the menu option **Project | Properties**.



You can change output name, output directory, and default language in the Output Settings group box, and define the base package name in the Java settings group box. Please note that project name and project directory cannot be changed after the project has been created.

5.3 Mapping pane

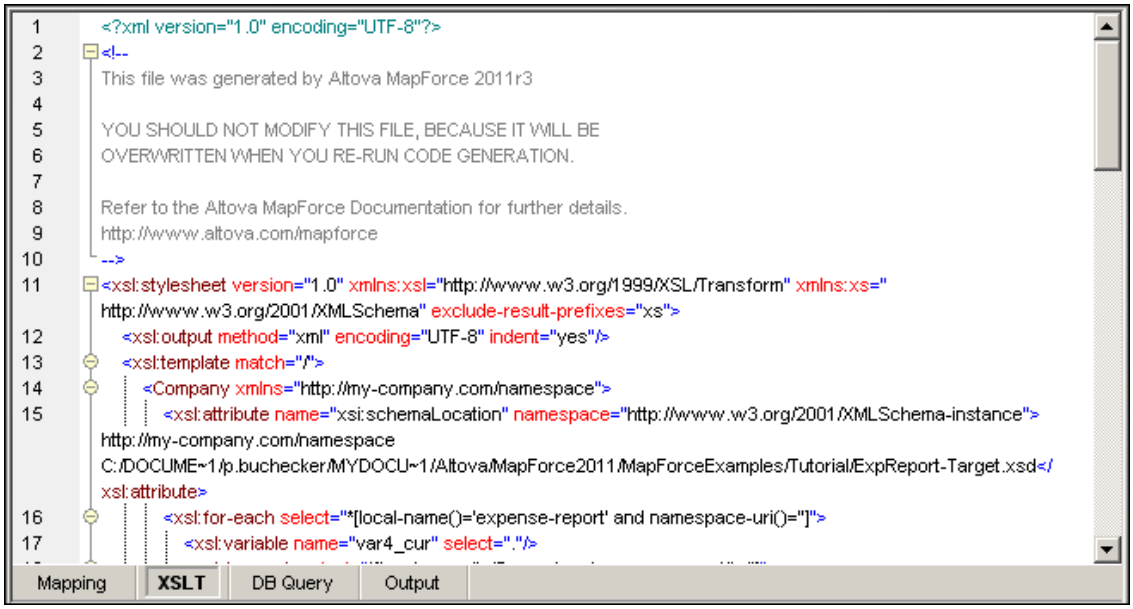
The Mapping pane is the working area in MapForce where you create your mappings.



The Mapping pane displays the graphical elements used to create the mapping (transformation) between the two components. Connectors connect the input and output icons of each schema item. Schema items can be either elements or attributes.

5.4 XSLT/XSLT2/XQuery pane

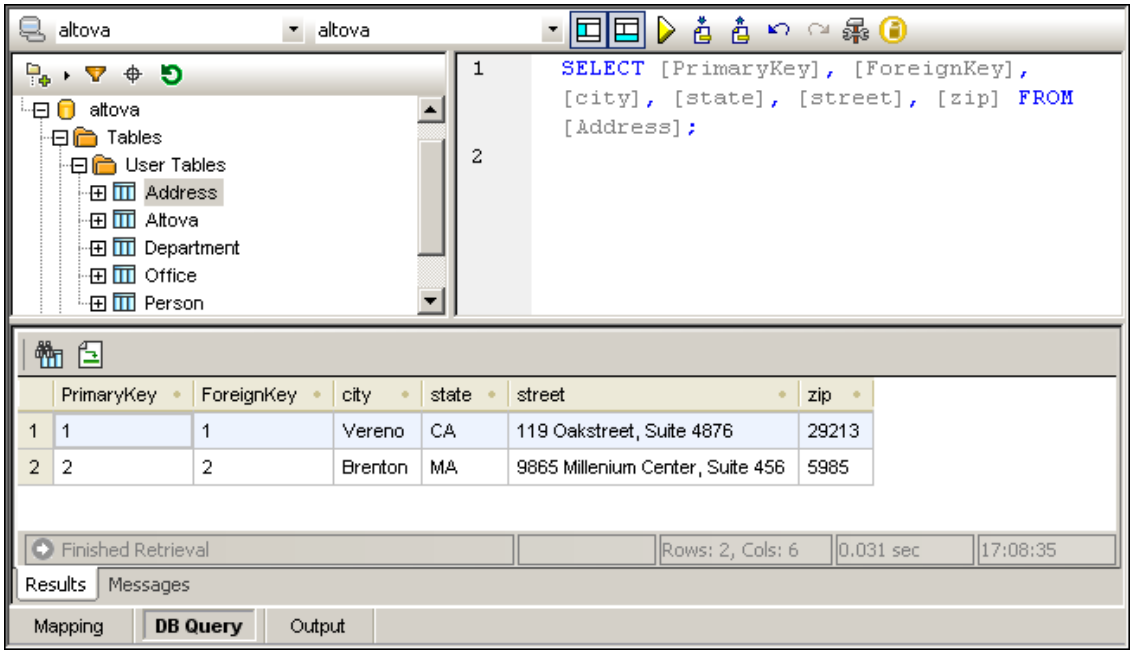
The **XSLT**, **XSLT2**, and **XQuery** panes display a preview of the transformation depending on the specific language selected.



Note: If you want to change the output language, you have to change back to the Mapping pane to do so. When a certain language tab is active, you cannot change the output language in the **Output** menu or the **Language Selection** toolbar, respectively.

5.5 DB Query pane

The **DB Query** pane allows you to directly query any major database. The queries/actions defined here are independent of any of the other MapForce tabs, and are not saved as part of the *.MFD file.



Each Database Query window is associated with the currently active mapping, allowing **multiple database queries** per session/mapping. Note that you can also have multiple active connections, to different databases, for each Database Query window.

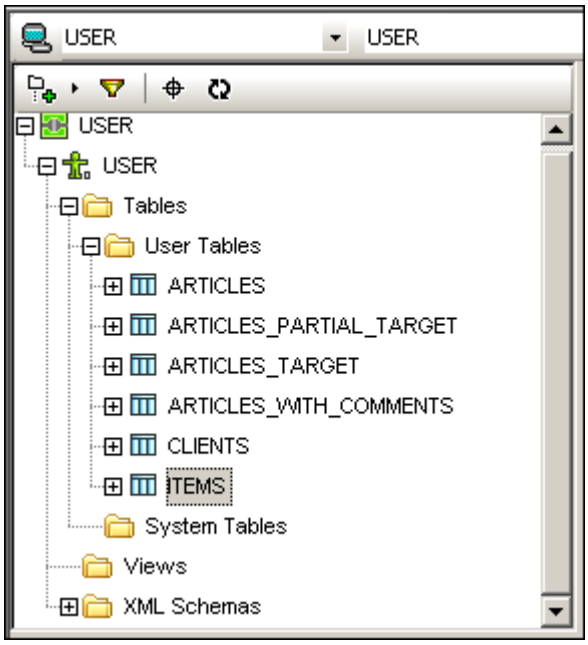
The Database Query tab is divided into the following windows:

- The **Browser** window at left, which displays connection info and database tables
- The **SQL Editor** window, to the right of the Browser window, in which you write your SQL queries
- The **Result** window which displays the query results in tabular form
- The **Messages** window which displays warnings or error messages

The top row of the Database Query window contains the Connection controls allowing you to define the working databases, as well as the connection and database schemas.

5.5.1 Browser window

For each of the (multiple) connected data sources, the **Browser** pane gives a full overview of the objects in each database, including database constraint information, e.g. column as a primary or foreign key. In IBM DB2 version 9 databases, the Browser additionally shows registered XML schemas in a separate folder.



The Browser view can be customized to:

- show specific folder **layouts** when displaying database objects,
- **find** specific objects in the database using the Object Locator,
- **filter** the number of displayed item,
- **refresh** the root object of the active data source.

Folder layouts

The Browser pane contains several predefined layouts used to display various database objects:

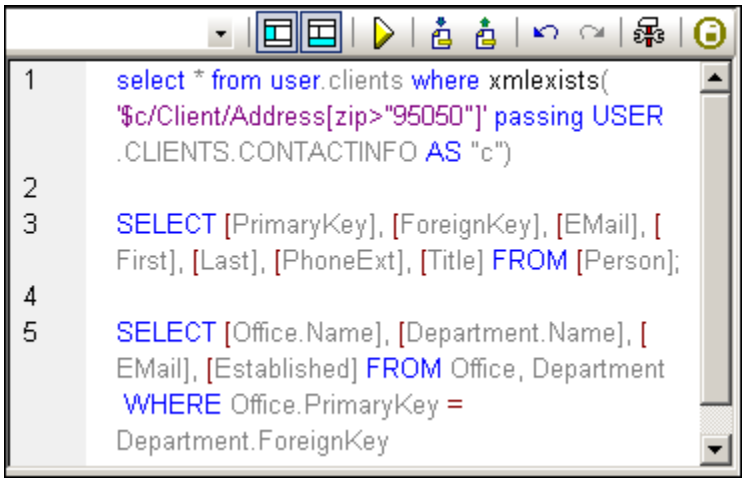
- The **Folders** layout organizes database objects into folders based on object type in a hierarchical tree, this is the default setting.
- The **No Schemas** layout is similar to the Folders layout, except that there are no database schema folders; tables are therefore not categorized by database schema.
- 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.

The default "Folders" layout displays database objects in an hierarchical manner. Depending on the selected object, different context menu options are available when you right-click an item.

5.5.2 SQL Editor

The **SQL Editor** is used to write and execute SQL statements and provides the following features:

- **Autogeneration** of SQL statements using **drag & drop** from the Browser pane
- Autocompletion of SQL statements when creating select statements
- Definition of **regions**
- Insertion of line or block **comments**



The following icons are provided in the SQL toolbar:

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 in the active window of the SQL Editor. 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.

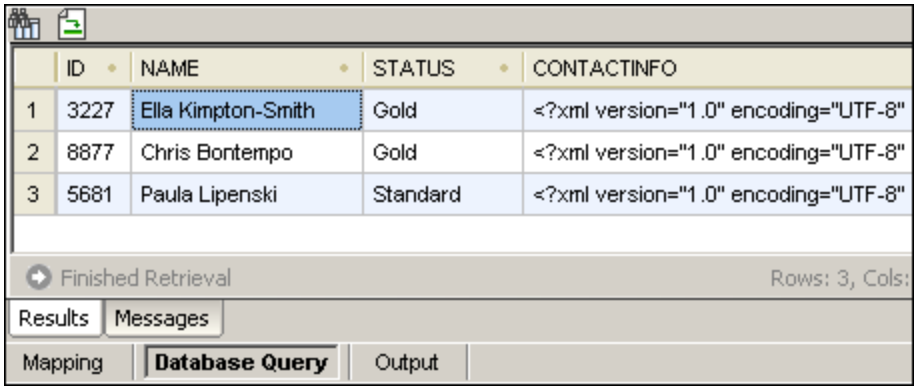
Export SQL file: Saves SQL queries for later use.

Open SQL script in DatabaseSpy: Starts DatabaseSpy and opens the script in the SQL Editor window.

Options: Opens the **Options** dialog box allowing you to define General as well as SQL Editor settings.

5.5.3 Result tab

The **Result** tab of the SQL Editor shows the record set that is retrieved as a result of a database query.



Toolbar options - Retrieval mode

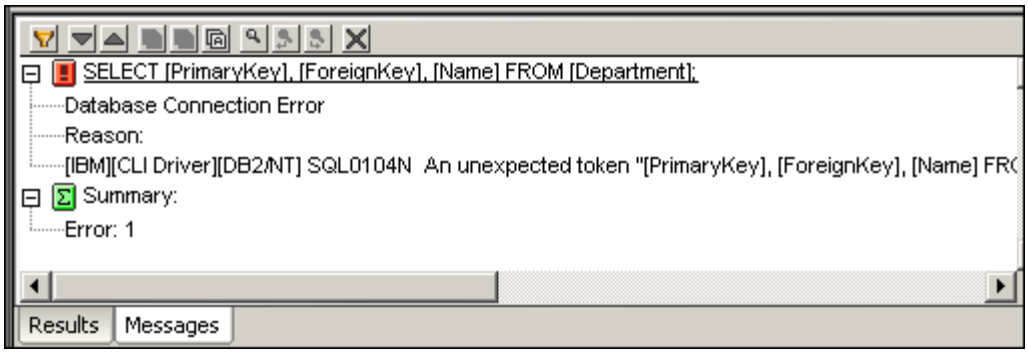
The Result window provides a toolbar that allows for the navigation between results and SQL statements and facilitates the easy retrieval of parts of database data.

Find: Searches for the input string in the Result window. The F3 function key allows you to continue the search.

Go to statement: Jumps to the SQL Editor window and highlights the group of SQL statements that produced the current result.

5.5.4 Messages tab

The **Messages** tab of the SQL Editor provides specific information on the previously executed SQL statement and reports errors or warning messages.



You can use different filters to customize the view of the Message tab or use the **Next** and **Previous** buttons to browse the window row by row. The Message tab also provides a **Find** dialog box and several options to copy text to the clipboard.

Toolbar options

The Message window provides a toolbar that allows for the navigation inside the messages and includes filters for hiding certain parts of the message.

Filter: Clicking this icon opens a popup 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 popup 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.

Please note: The same options are available in the context menu of the result window.

5.6 Output pane / BUILTIN execution engine

The result of a mapping is immediately presented in the Output tab, using the Altova XSLT or XQuery engine, depending on the selected language.

If any of the compiled programming languages (Java, C++ or C#) or BUILTIN is selected, the built-in execution engine generates the output from source components: XML/Schema files, Text files, databases, etc. The result that appears in the Output tab is the same as if the Java, C++, or C# code had been generated, compiled and executed.

Please note: The result generated by the Built-in execution engine, is an on-the-fly transformation of Database or Text, without you having to generate, or compile program code! We would recommend that you use this option until you are satisfied with the results, and then generate program code once you are done.

Depending on the **target** component of your mapping, the Output pane may show different things:

- **XML Schema/document as target**
  The screenshot below shows the output of the **DB_CompletePO.mfd** mapping available in the **...\MapForceExamples** folder. An XML Schema/document, as well as a database are used as source components in this mapping.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    <Customer>
4      <Number>3</Number>
5      <FirstName>Ted</FirstName>
6      <LastName>Little</LastName>
7      <Address>
13   </Customer>
14   <LineItems>
15     <LineItem>
16       <Article>
17         <Number>3</Number>
18         <Name>Pants</Name>
19         <SinglePrice>34</SinglePrice>
20         <Amount>5</Amount>
21         <Price>170</Price>
22       </Article>
23     </LineItem>
24     <LineItem>
25       <Article>
32     </LineItem>
33   </LineItems>
34 </CompletePO>
```

The resultant XML file can be saved by clicking the **Save generated output** icon, and validated against the referenced schema by clicking the **Validate Output** icon in the icon bar.

- **Database as target**
  Executes the mapping to the target database, taking the defined table actions into account.

```
/*
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%')
```

Clicking the **Run SQL-script** icon executes the SQL select statement and presents you with a report on the database actions, as described below:

- Actual SQL statements that were executed on the target database
  ```
  SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey]
  FROM [Altova]
  -->>> OK. One or more rows.
  ```
- Multiple table actions if any occurred
  ```
  INSERT INTO [Altova] ([Name], [PrimaryKey]) VALUES ('Organization Chart', 1)
  -->>> OK. 1 row(s).
  ```
- Results of every SQL statement

-->>> OK. n row(s). if successful, or Execution failed, and a detailed error message.

```
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).
```

- **Text file as target**
  The Built-in execution engine includes support for displaying the results of the following text files as targets:
  – CSV files (comma-separated values) – also for other delimiters than comma
  – FLF files (fixed-length fields)

```
1  "Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Albert,Aldrich,582,A.Ald
2  "Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Bert,Bander,471,b.bander
3  "Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Clive,Clovis,963,c.clovi
4  "Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,5588339,Dave,Durne
5  "Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,5588339,Eve,Ellas,
6  "Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,Fred,Fortunas,95
7  "Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,Gerry,Gundall,65
```

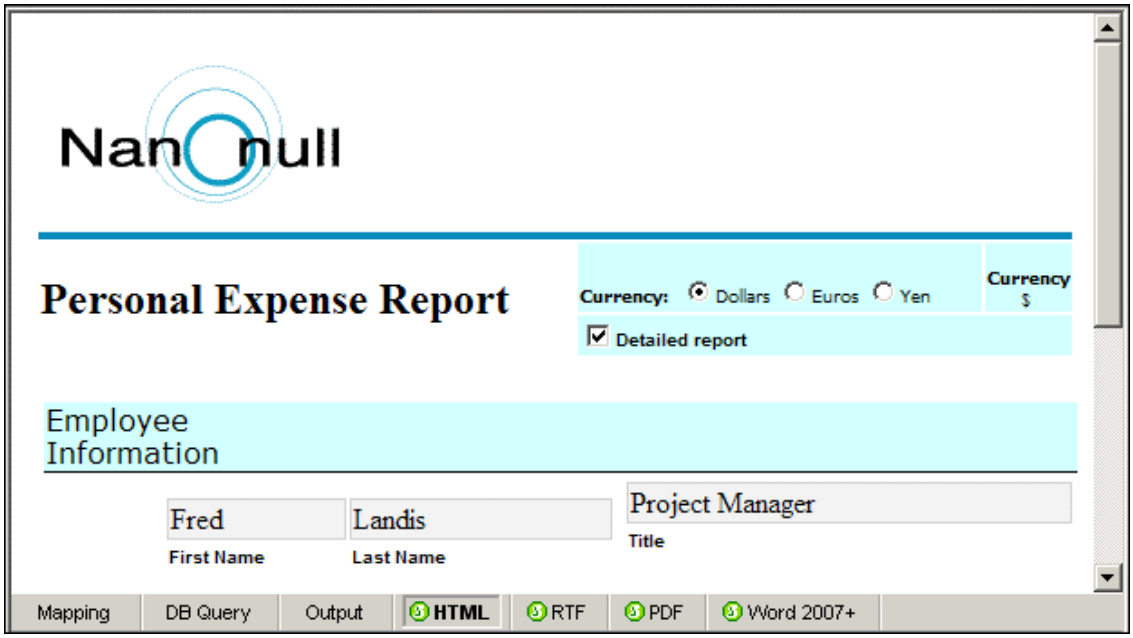
Hotkeys for the Output window (keyboard and numeric key pad)

CTRL and "+" zoom in on the text
CTRL and "-" zoom out of the text
CTRL and "0" resets the zoom factor to standard

CTRL and mouse wheel forward / backward achieve the same zoom in/out effect.

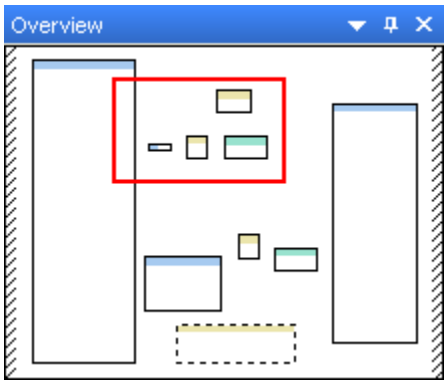
5.7 StyleVision-related panes

The **HTML**, **RTF**, **PDF**, and **Word 2007+** tabs display the target component data as HTML, RTF, PDF, or Word 2007+ documents if a StyleVision Power Stylesheet (SPS) is associated with the target component.



5.8 Overview window

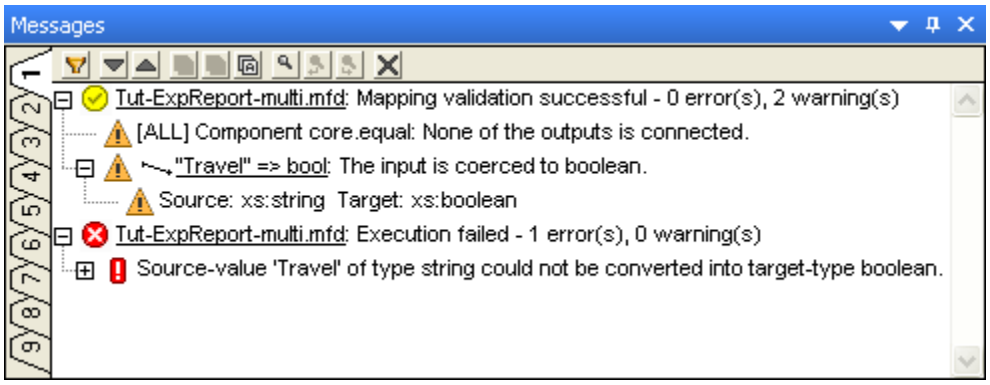
The Overview window serves as a navigator pane for large mappings. A red rectangle shows the currently visible area in the Mapping pane. You can drag the rectangle in the Overview window with your mouse to adjust the visible part of the mapping in the Mapping window.



Clicking into the Overview window will define the center of the display in the Mapping pane.

5.9 Messages window

The Messages tab shows messages, errors, and warnings when you click the Output button or perform a mapping validation.

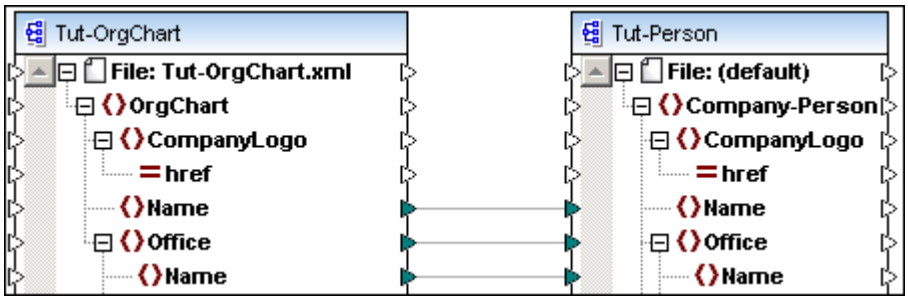


Chapter 6

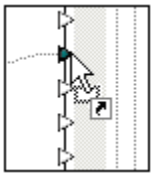
Mapping between components

6 Mapping between components

A **connector** visualizes the **mapping** between the two sets of data and allows the **source** data (value) to appear, or be transformed, into the target component e.g. schema/document or database etc.



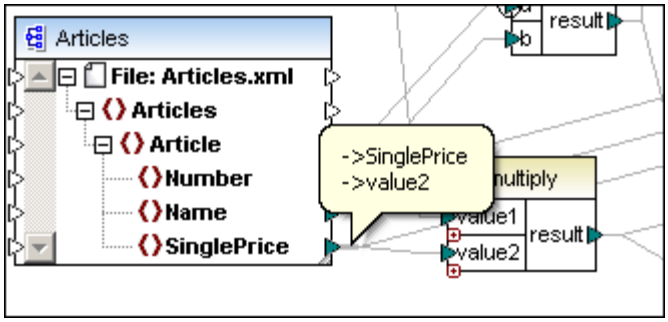
Components and **functions** have small "connection" triangles called: **input** or **output** icons. These **icons** are positioned to the left and/or right of all "mappable" **items**. Clicking an icon and dragging, creates the **mapping connector**. You can now drop it on another icon. A link icon appears next to the text cursor when the drop action is allowed.



Clicking an **item name** (element/attribute) automatically selects the correct **icon** for the dragging action.

An **input icon** can only have one connector. If you try and connect a second connector to it, a prompt appears asking if you want to replace or **duplicate** the input icon. An **output icon** can have several connectors, each to a different input icon.

Positioning the mouse pointer over the straight section of a connector (close to the input/output icon) highlights it, and causes a popup to appear. The popup displays the name(s) of the item(s) at the other end of the connector. If multiple connnectors have been defined from the same output icon, then a maximum of ten item names will be displayed. The screenshot above shows that the two target items are **SinglePrice** and **value2** of the multiply function.



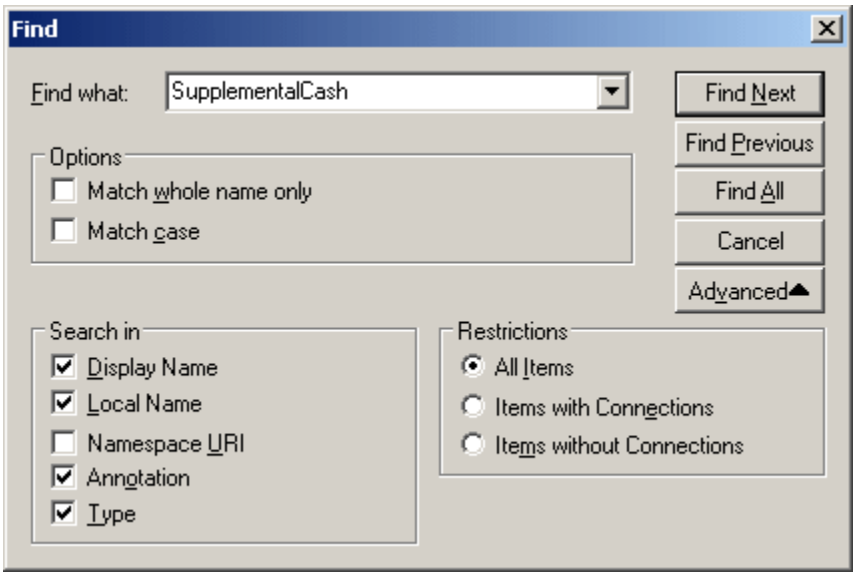
To **move** a connector to a different item, point to the straight section of the connector and drag it elsewhere.

To create a **duplicate connector** from the same source to another target, point to the straight section of the connector near the original target, and drag it to another target while holding down

the CTRL key.

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 OK.



The Advanced options visible in the screenshot above, allow you to define which items/nodes are to be searched, as well as restrict the search options based on the specific connections.

Autoconnecting items

Activating the autoconnect child items icon  , and creating a connector between two items, automatically connects all child items of the same name under the parent item.

Number of connectors

Input and output icons appear on most components, there is not, however, a one to one relationship between their numbers.

- Each **schema item** (element/attribute) has an input and output icon.
- **Database** items have input and output icons.
- Schema, database and other components within user-defined functions only have output icons.
- Duplicated items only have input icons. This allows you to map multiple inputs to them. Please see Duplicating Input items for more information.
- **Functions** can have any number of input and output icons, **one** for each **parameter**. E.g. the Add Function has two (or more) input icons, and one output icon.
- **Special** components, can have any number of icons, e.g. the Constant component only has an output icon.

6.1 Methods of mapping data (Standard / Mixed Content / Copy Child Items)

MapForce supports various methods of mapping data: Target-driven (Standard), Source-driven (Mixed Content), and Copy All (Copy Child Items).

Connectors and their properties
The following actions can be performed on a connector and produce the results described below:

- Clicking a connector highlights it in red.
- Hitting the Del key, while a connector is highlighted, deletes it immediately.
- Right-clicking a connector, opens the connector context menu.
- Double-clicking a connector, opens the Connection Settings dialog box.

Viewing connectors
MapForce allows you to selectively view the connectors in the mapping window.

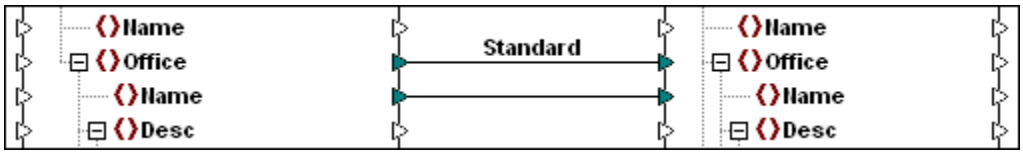
Show selected component connectors switches between showing:
- all mapping connectors in black, or
- those connectors relating to the currently selected component in black. Other connectors appear dimmed.

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.

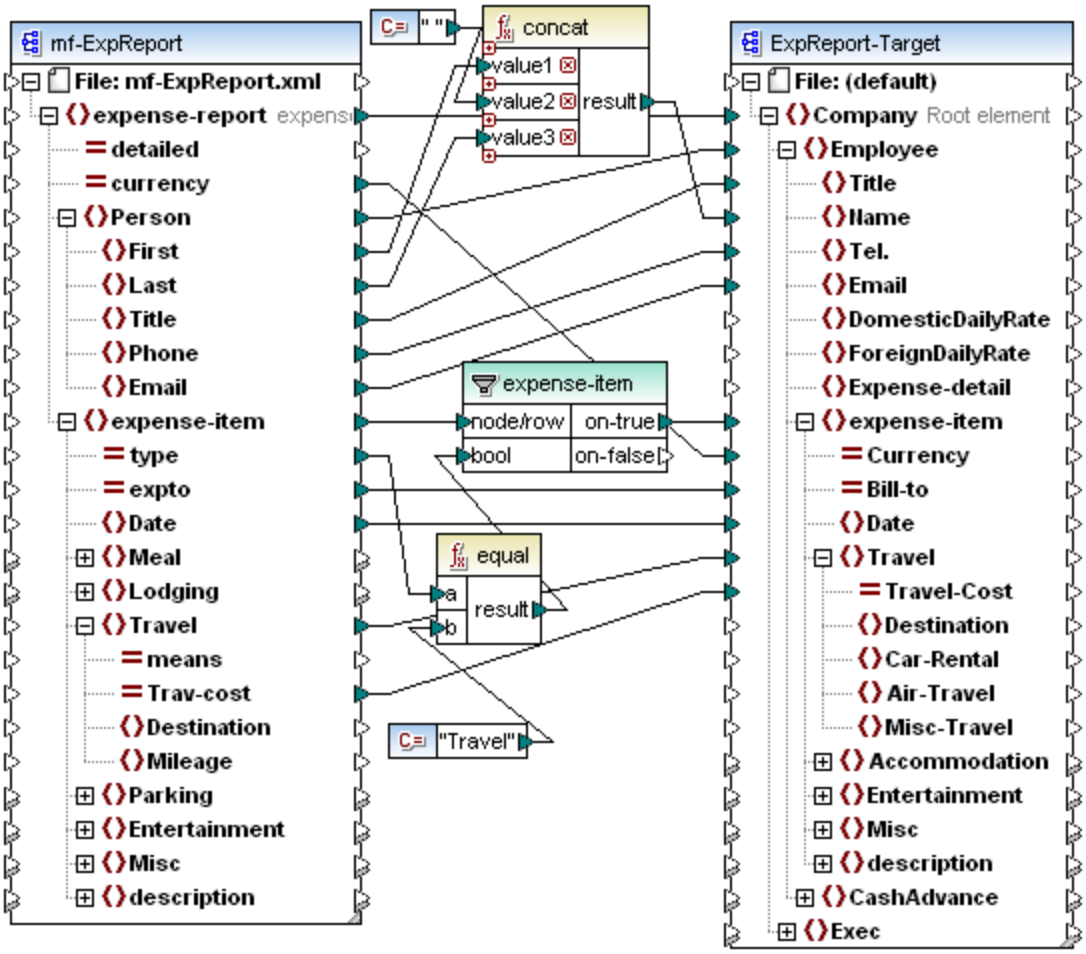
6.1.1 Target-driven / Standard mapping

Target-driven (Standard) mapping means the normal method of mapping used in MapForce, i.e. the output depends on the sequence of the target nodes.

- Mixed content text node content is not supported/mapped.
- The sequence of child nodes is dependent on the target schema file.



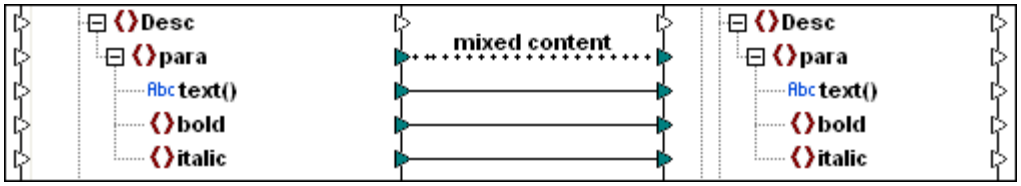
Standard mappings are shown with a solid line.



6.1.2 Source-driven / mixed content mapping

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:

- As **source** components:
  - XML schema complexTypes,
  - XML schema complexTypes of type mixed content, i.e. mixed=true.
- As **target** components:
  - XML schema complexTypes (including mixed content),
  - database tables,
  - CSV and fixed-length files.

    Note: CDATA sections are treated as text.

Mapping mixed content

The files used in the following example (**Tut-Orgchart.mfd**) 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 it's child nodes "bold" and "italic".

Please note that the para element also contains a Processing Instruction (sort alpha-ascending) as well as Comment text (Company details...) which can also be mapped.

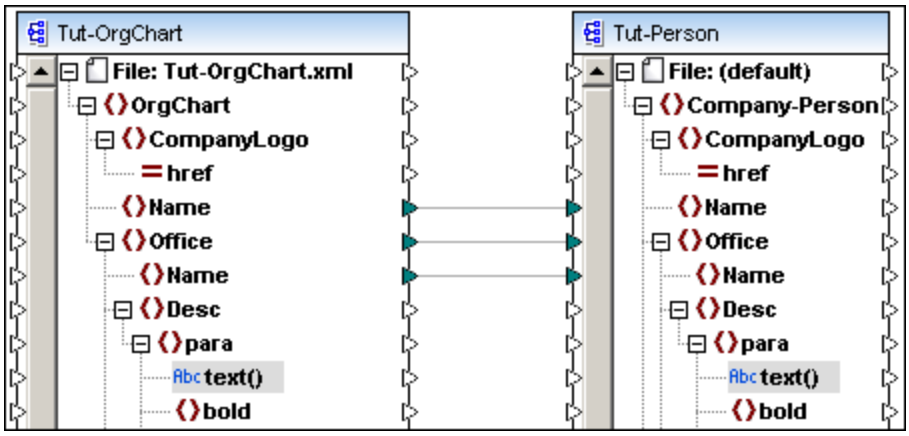
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp2 U (http://www.altova.com) by Mr. Nobody (Altova GmbH) -->
<OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Tut-OrgChart.xsd">
  <CompanyLogo href="nanonull.gif"/>
  <Name>Organization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

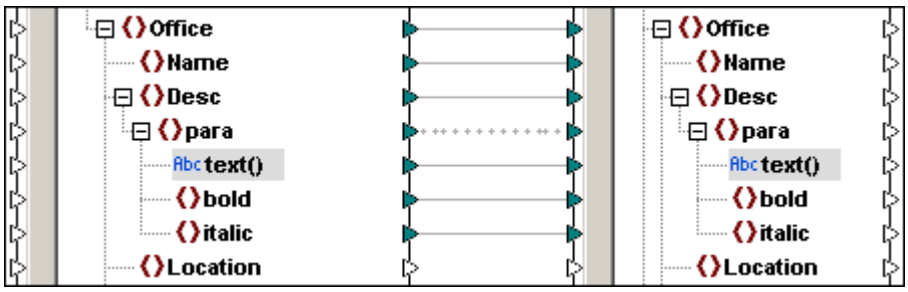


Output of above mapping
The result of the initial mapping is shown below: Organization Chart as well as the individual office names have been output.

```
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
```

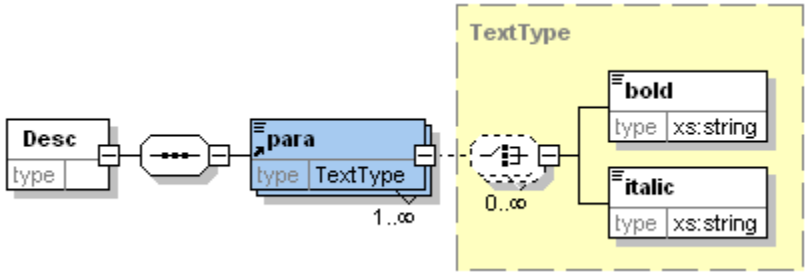
Mapping the para element

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.



Right clicking a connector and selecting Properties, allows you to annotate, or label the connector. Please see section "Connection" in the Reference section for more information.

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.



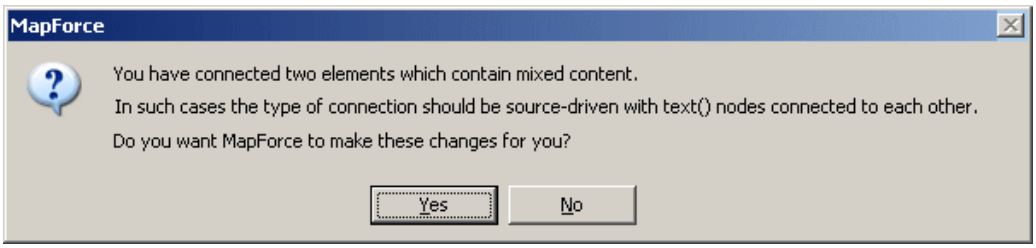
Note the following properties of the **para** element in the Content model:

- **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.



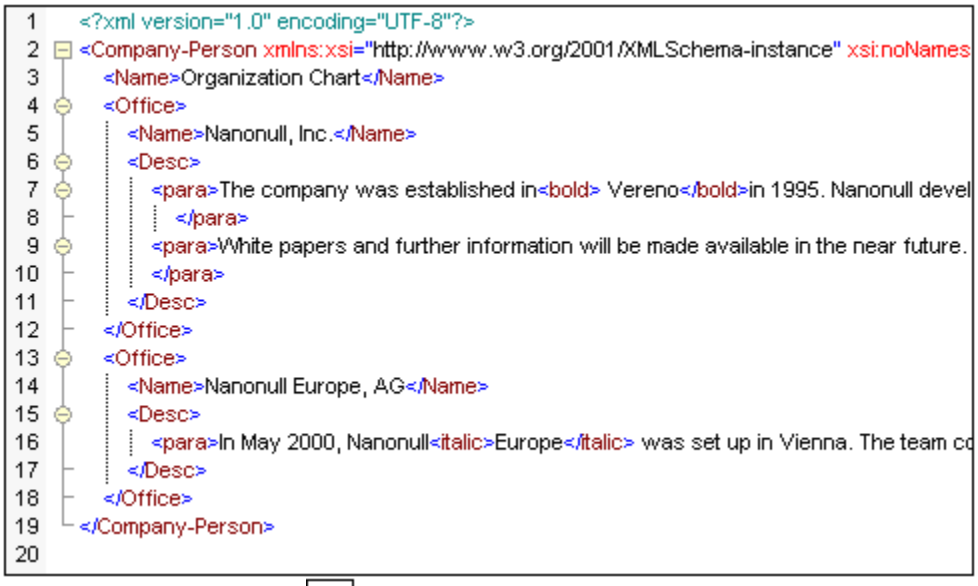
3. Click Yes to create a mixed content connection.

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.



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

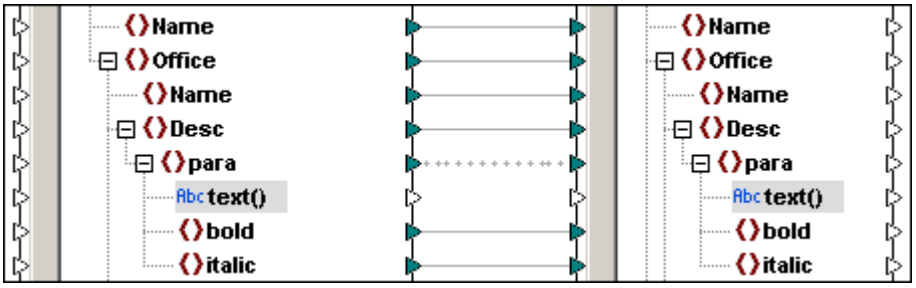
```
3     <Name>Organization Chart</Name>
4     <Office>
5       <Name>Nanonull, Inc.</Name>
6       <Desc>
7         <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull develops
      nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999 saw the
      unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand its
      operations <italic>offshore</italic>to drive down operational costs.
8           </para>
9         <para>White papers and further information will be made available in the near future.
10        </para>
11      </Desc>
12    </Office>
13    <Office>
14      <Name>Nanonull Europe, AG</Name>
15      <Desc>
16        <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>
17      </Desc>
18    </Office>
19  </Company-Person>
20
```

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

6. Switch back to the Mapping view.

To remove text nodes from mixed content items:

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



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

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

Result:

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

Mixed content example

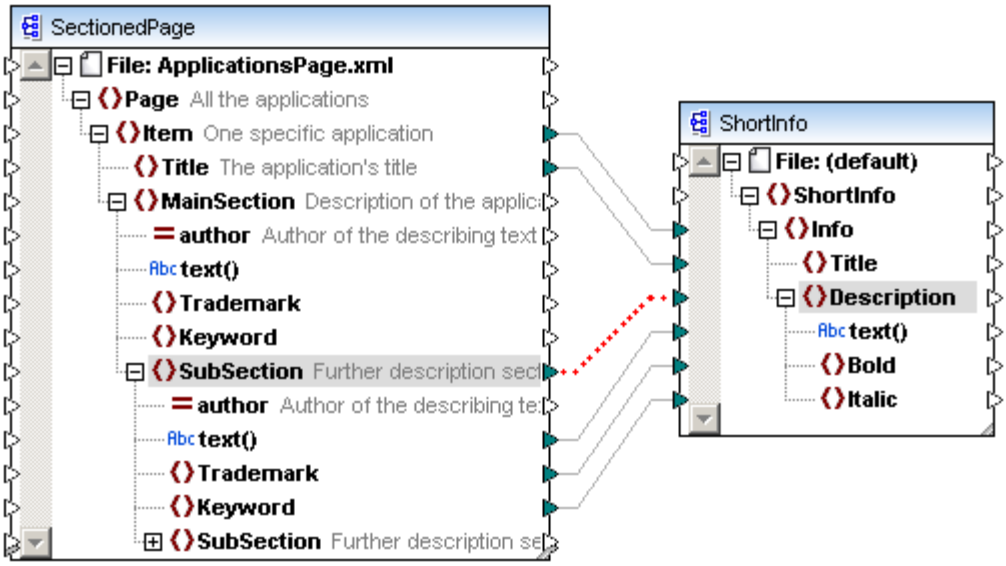
The following example is available as "**ShortApplicationInfo.mfd**" in the **...\MapForceExamples** folder.

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.

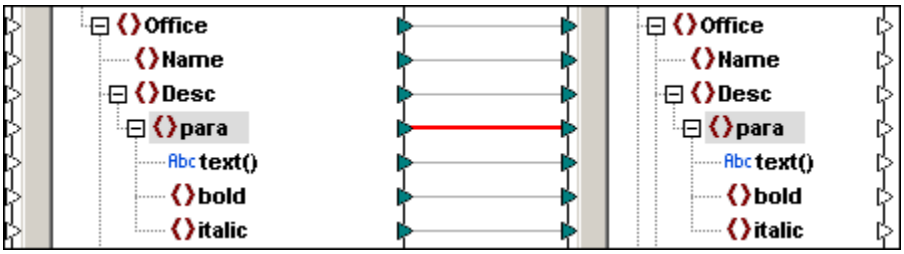
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ShortInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="
   C:/PROGRA~1/Altova/MapForce2005/MapForceExamples/ShortInfo.xsd">
3    <Info>
4      <Title>XMLSpy</Title>
5      <Description>Altova <Bold>XMLSpy</Bold> 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>
```

Using standard mapping on mixed content items

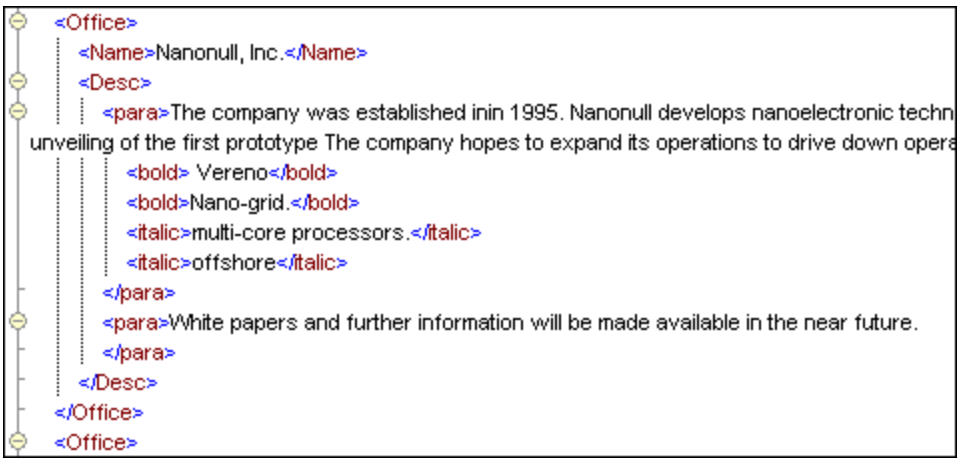
This section describes the results when defining **standard** mappings (or using standard connectors) on **mixed content** nodes. The files used in the following example (**Tut-Orgchart.mfd**) are available in the ...\MapForceExamples\Tutorial\ folder.

To create standard connections between mixed content items:

1. Create a connector element 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.



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



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

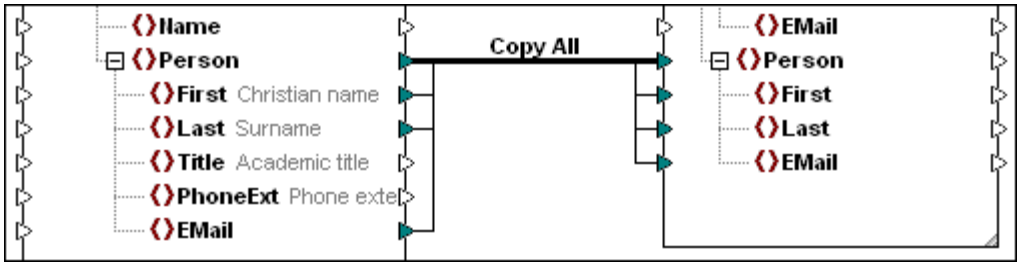
- Text() **content** is supported/mapped.
- The start/end tags of the child nodes, bold and italic, are removed from the text node.
- The child nodes appear after the mixed content node text.
- The **sequence** of child nodes depends on the sequence in the **target** XML/schema file.

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.

6.1.3 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 target item is not created.



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

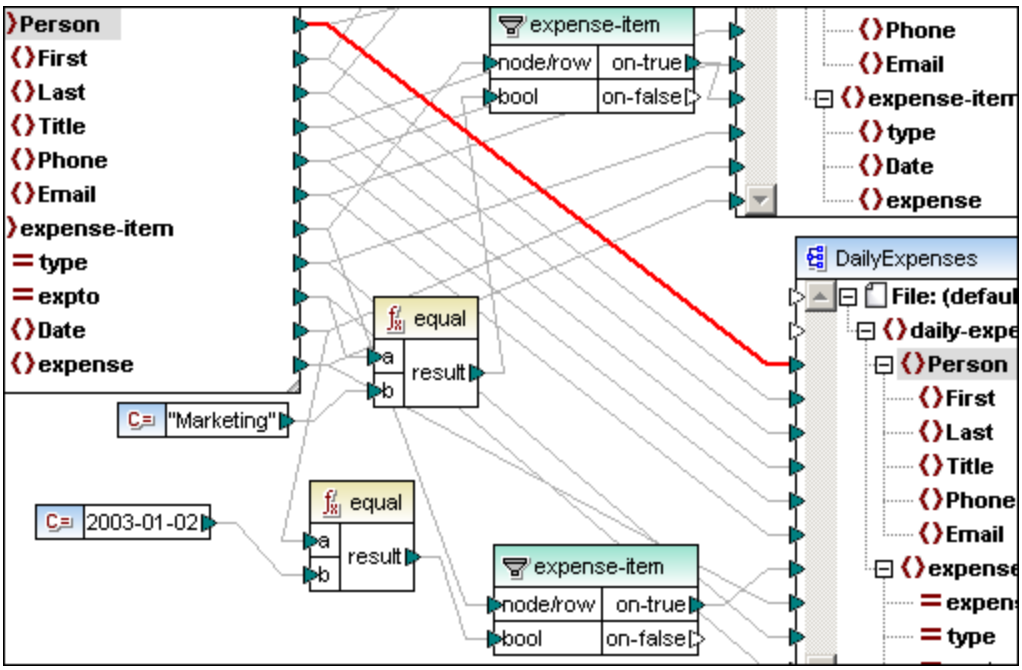
Note that only the names of the child items, but not their individual types, are compared/matched.

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

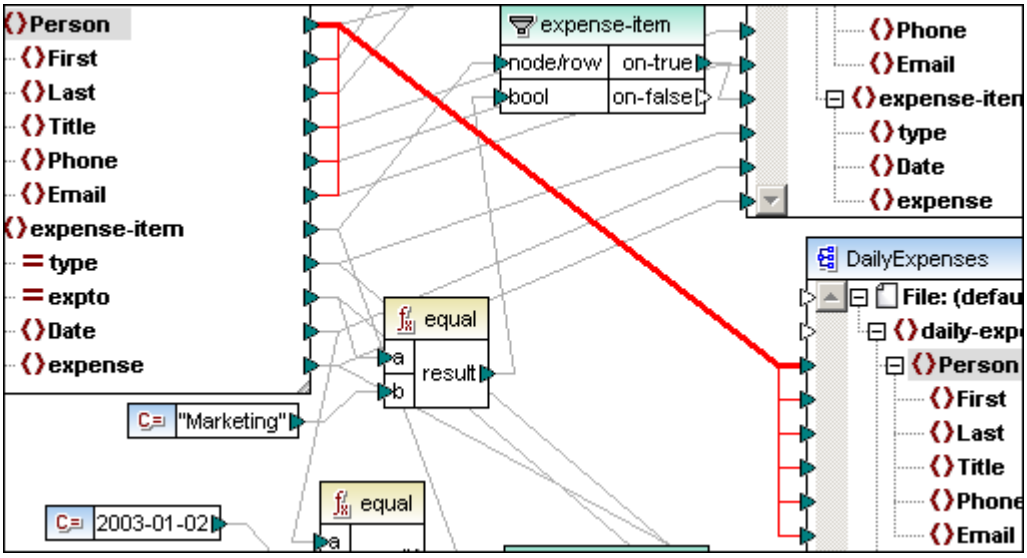
The example below shows these connectors using the **MarketingAndDailyexpenses.mfd** file in the **...\MapForceExamples** folder.

To define a Copy-all connection:

1. Right-click an existing connector, e.g. the Person connector, and select "Copy-all" from the context menu.
   A prompt appears reminding you that all connections to the target child items will be replaced by the copy-all connection.



2. Click OK to create Copy-all connectors.



All connectors to the target component, and all source and target items with identical names are created.

Please note:

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

To resolve/delete copy-all connectors:

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

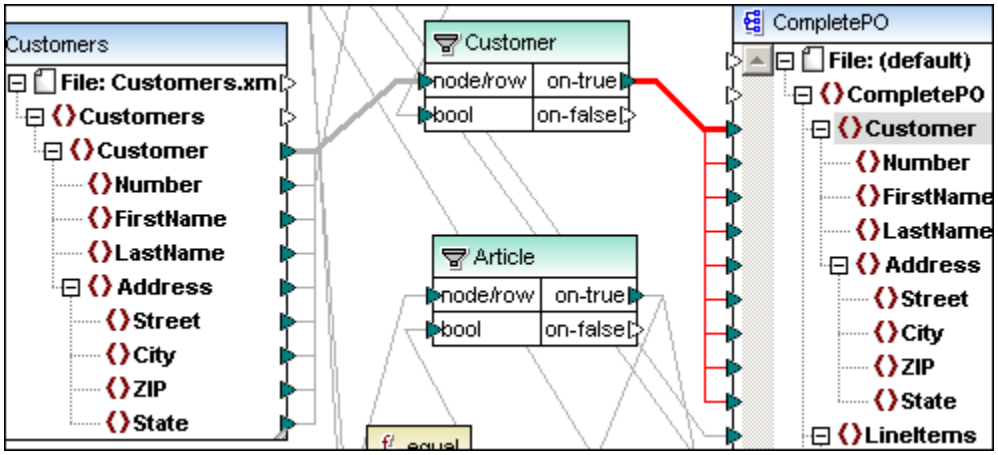
When creating Copy-all connections between a schema and a user-defined function parameter, the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components - defining" for an example.

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 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 define a copy-all connection through a filter component:

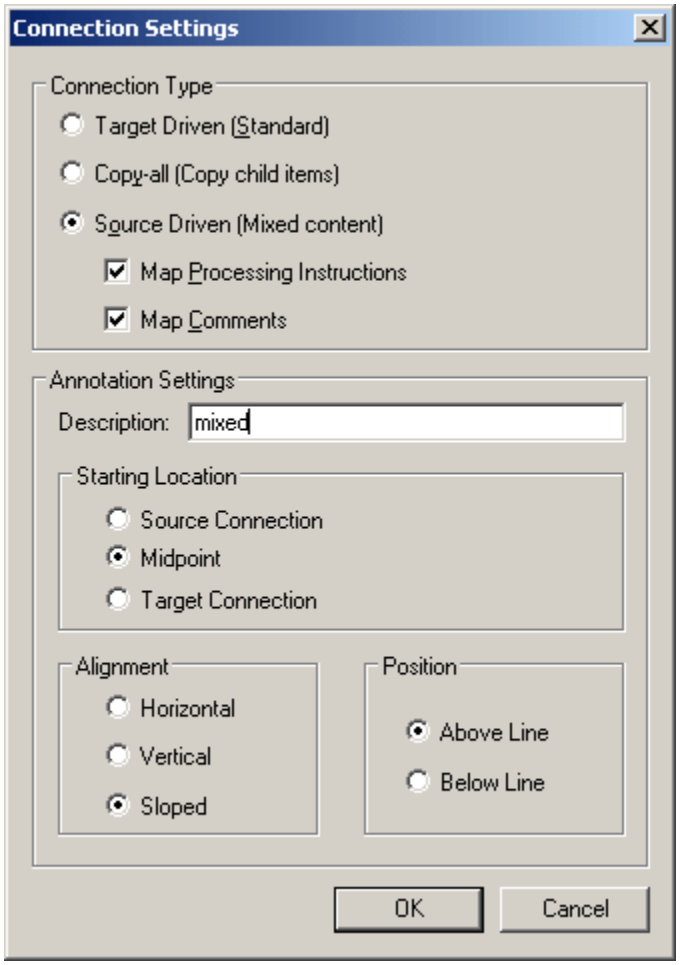
1. Right-click the filter **output** connector i.e. the right hand **on-true**/on-false connector.
2. Select Copy-all (Copy child items) from the context menu. The copy-all connector between items of the same name are created.

Please note:

To change the connector back to a different type, make sure you right click the connector on the output side of the filter. The left hand connector cannot be used to change the connection type because the on-true and on-false connectors might have different settings.

6.2 Connection settings

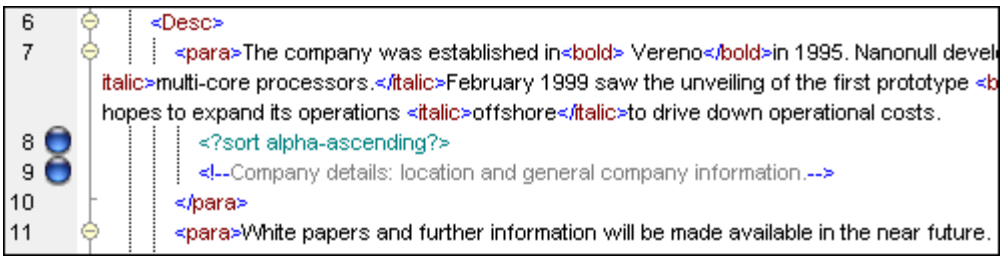
Right-clicking a connector and selecting **Properties** from the context menu, or double-clicking a connector, opens the Connection Settings dialog box in which you can define the specific (mixed content) settings of the current connector. Note that unavailable options are greyed out.



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

- **Target Driven (Standard)**: Changes the connector type to Standard mapping, please see Target-driven / Standard mapping for more information.
- **Copy-all (Copy child items)**: Changes the connector type to Copy-all and automatically connects all identical items in the source and target components. Please see Copy-all connections for more information.
- **Source Driven (mixed content)**: Changes the connector type to source driven / mixed content, and enables the selection of additional elements to be mapped. The additional elements have to be **child items** of the mapped item in the XML source file, to be able to be mapped.

  Activating the **Map Processing Instructions** and/or **Map Comments** check boxes enables you to include those data in the output file.



Please note: CDATA sections are treated as text.

Annotation Settings
Individual connectors can be labeled allowing you to comment your mapping in great detail. When you enter a character in the Description field, the Starting Location, Alignment, and Position group boxes are activated and can be edited. This option is available for **all connection types**.

To add an annotation to a connector:

1. Enter the name of the currently selected connector 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 connector. The annotation text will appear in a popup if the **Show tips** icon is active in the View Options toolbar.

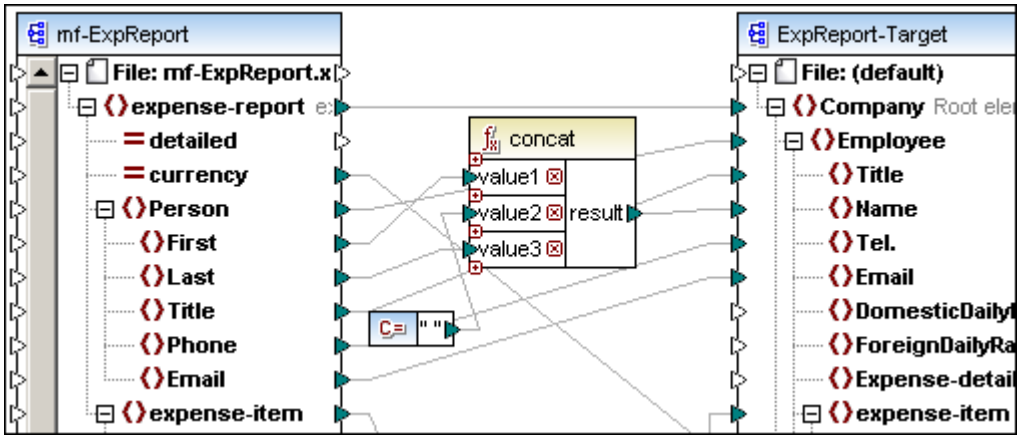
6.3 Moving connectors

Moving connectors and the effect on child connectors

When moving a parent connector to a different parent connector item, MapForce automatically matches identical child connections under the new location of the connector. This is not the same as the auto-connect child option, as it uses different rules to achieve this.

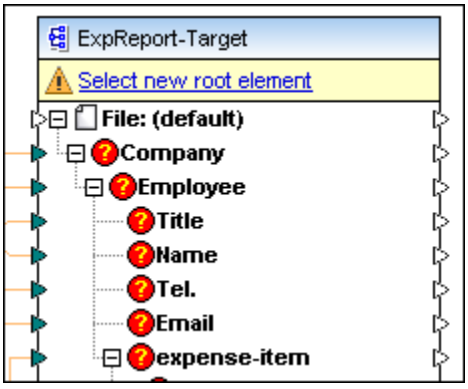
A common use of this feature is if you have an existing mapping and then change the root element of the target schema. This would normally force you to remap all descending connectors manually.

This example uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples\Tutorial folder.

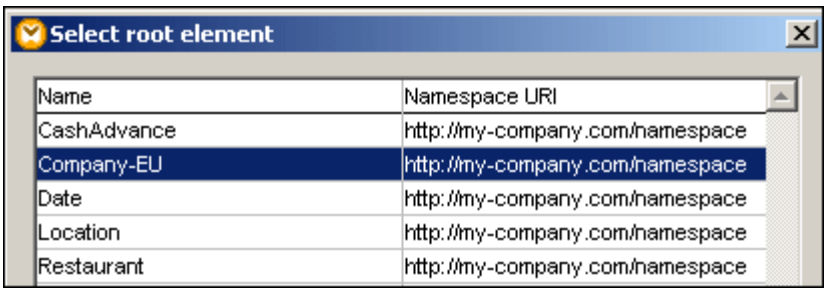


If the Company root element of the target schema, is changed to Company-EU then a "Changed files" prompt appears in MapForce.

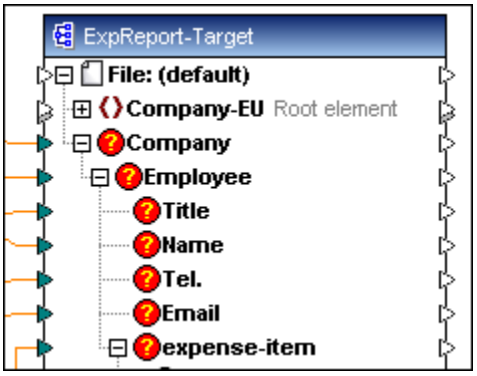
1. Click the **Reload** button to reload the updated Schema.
   You are now presented with multiple missing nodes as the root element has changed.



2. Click on the "Select new root element" link at the top of the component.

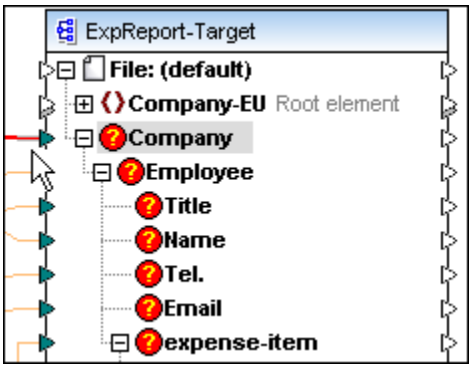


3. Select the updated root element, Company-EU and click OK to confirm.

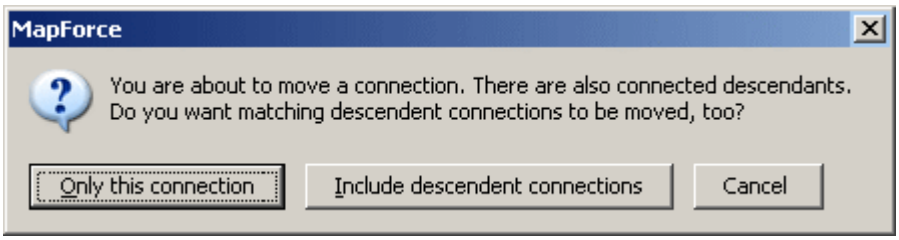


The Company-EU root element is now visible at the top of the component.

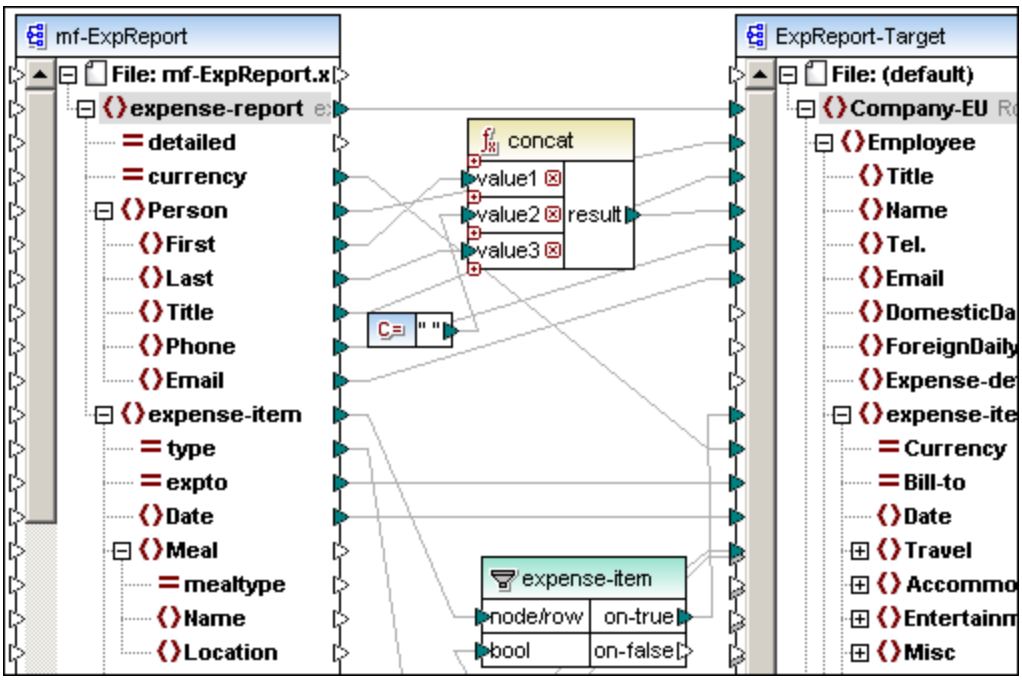
4. Click the connector on the Company item and use drag-and-drop to drop it on the new Company-EU root element.



A prompt appears asking which connectors you want to move.



5. Click the "Include descendent connections" button if you want to map the child connectors.
The "missing" item nodes have been removed and all connectors have been mapped to the correct child items under the new root element.



Please note:

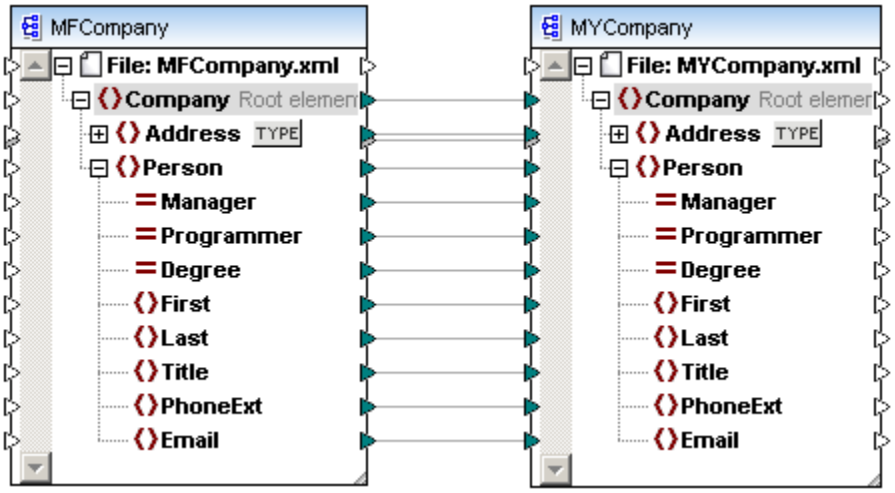
If the item/node you are mapping **to** has the same name (as the source node) but is in a different namespace, then the prompt will contain an additional button "Include descendants and map namespace".

Clicking this button moves the child connectors of the same namespace as the source parent node, to the same child nodes under the different namespace node. I.e. If the parent nodes only differ in their namespace, then the child nodes may only differ in the same way, if they are to be mapped automatically.

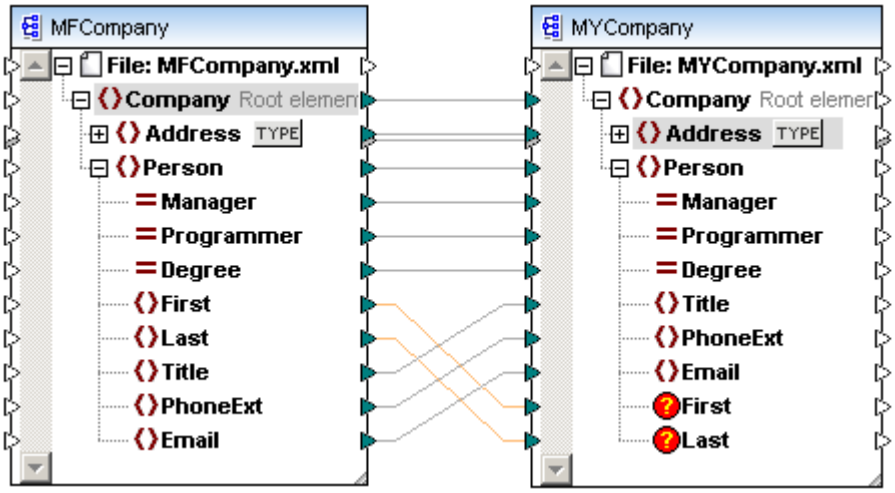
6.4 Missing items

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.



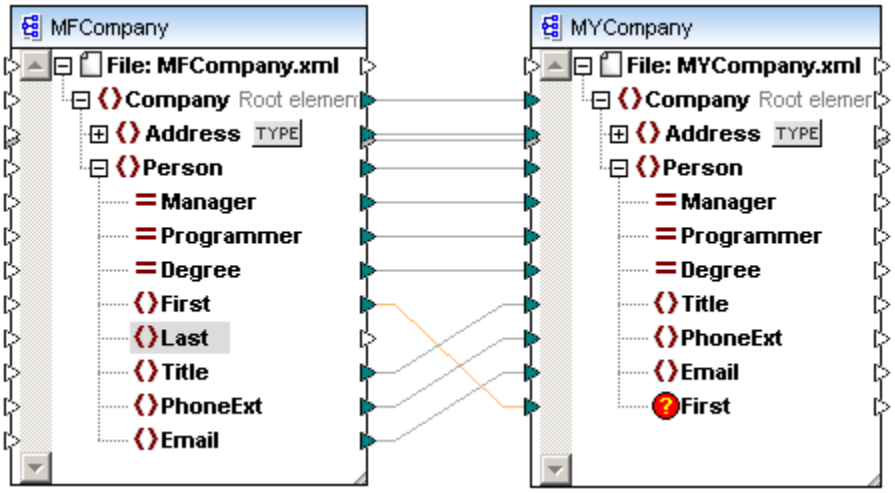
While editing MyCompany.xsd, in XMLSpy, the First and Last items in the schema are deleted. Returning to MapForce opens a Changed Files notification dialog box, prompting you to reload the schema. Clicking **Reload** updates the components in MapForce.



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.

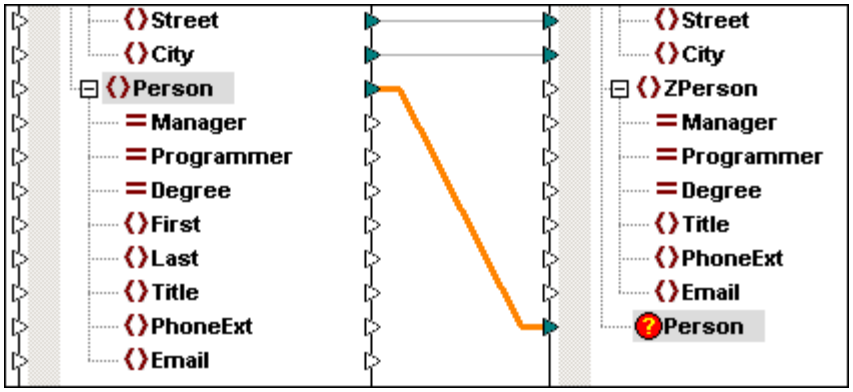
Note that you can still preview the mapping (or generate code), but warnings will appear in the Messages window if you do so at this point. All connections to, and from, missing items are ignored during preview or code-generation.

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



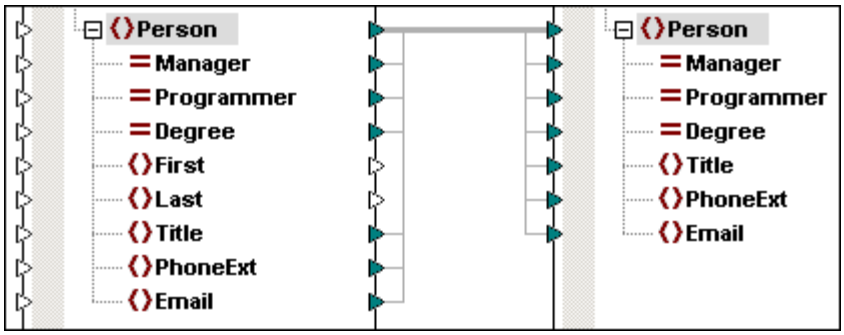
Renamed items

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

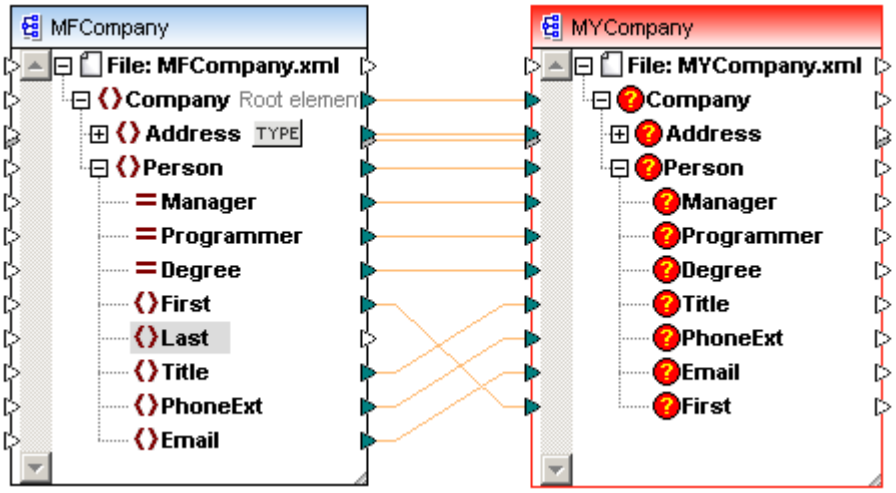
Copy all connections are treated in the same way as normal connections, with the only difference being that the connectors to the missing child items are not retained or displayed.



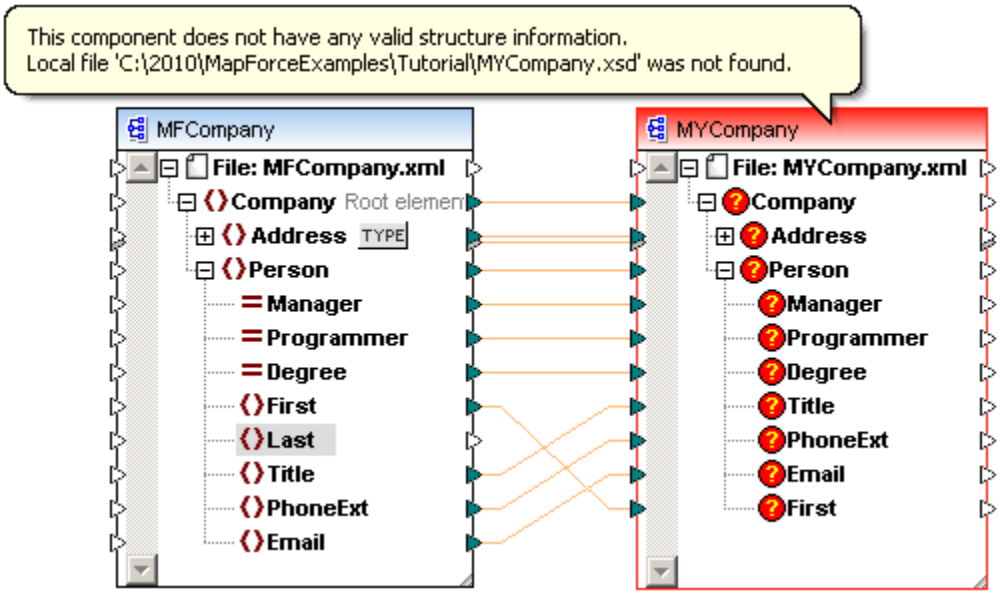
Renamed or deleted component sources

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.

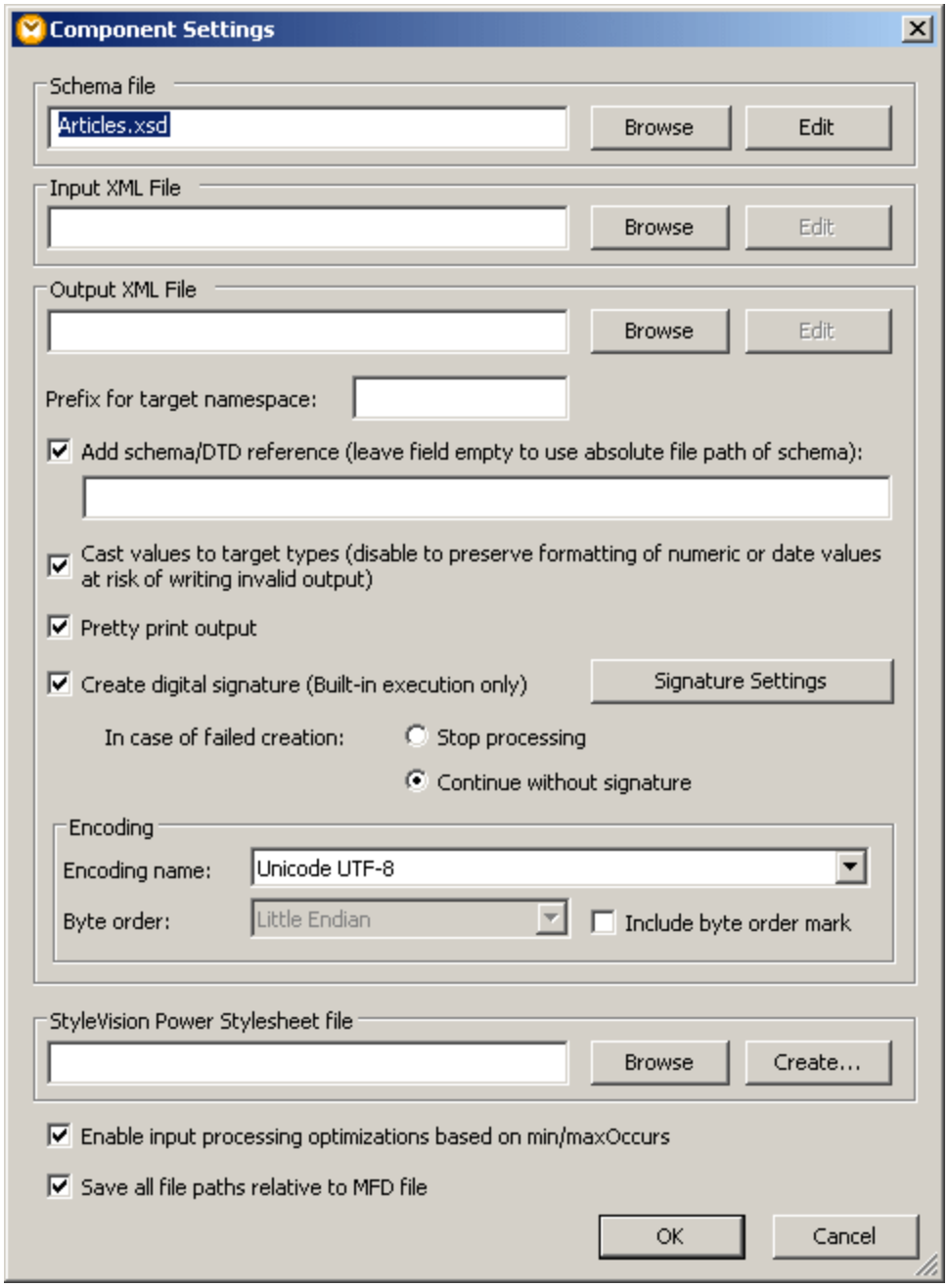


Placing the mouse cursor over the highlighted component, opens a popup containing pertinent information.



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.

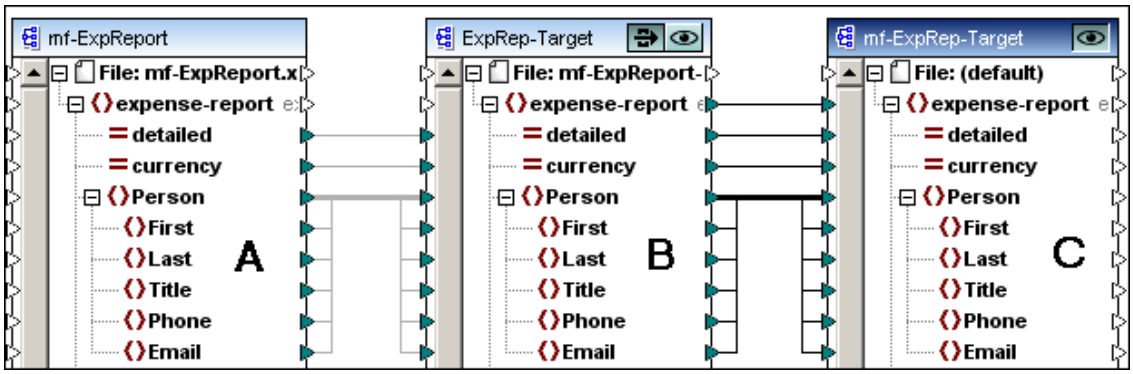
Clicking the **Change** button in the dialog box that opens if the component is a database, allows you to select a different database, or change the tables that appear in the database component. Connectors to tables of the same name will be retained.



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.

6.5 Chained mappings / pass-through components

MapForce supports mappings that consist of multiple components in a mapping chain. Chained mappings are mappings where two, or more, components are directly connected to another target component. The screenshot below, for example, shows three components A, B, and C, where C is the target component.



Component B (ExpRep-Target) is in this case the "intermediate" component as it has both input and output connectors. Intermediate component results, as well as the final mapping result, can be individually previewed and generated as code.

Preview button
Component B, as well as C, both 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. Click the preview button of the respective component, then click Output to see the mapping result.

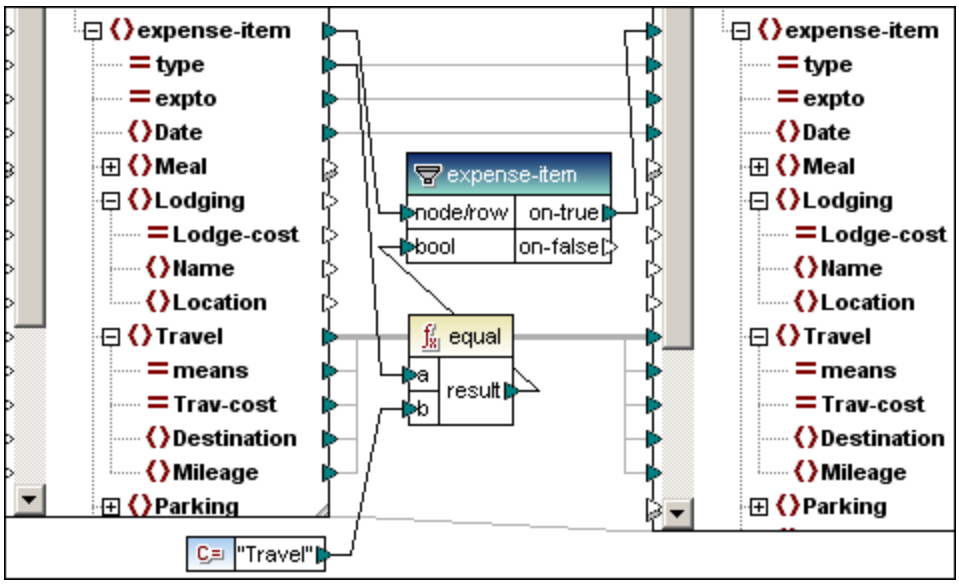
Pass-through button
The intermediate component B, has an extra button in the component title bar called "pass-through". This button allows you to define that the set of input data is passed on to the following component, e.g. component C.

- If the pass-through button (on component B) is **active**, two separate sets of data are passed through to component C, via the intermediate component B: (i) the subset of mapping data from intermediate component B to target component C, and (ii) The result set of the mapping from component A to the intermediate component B.

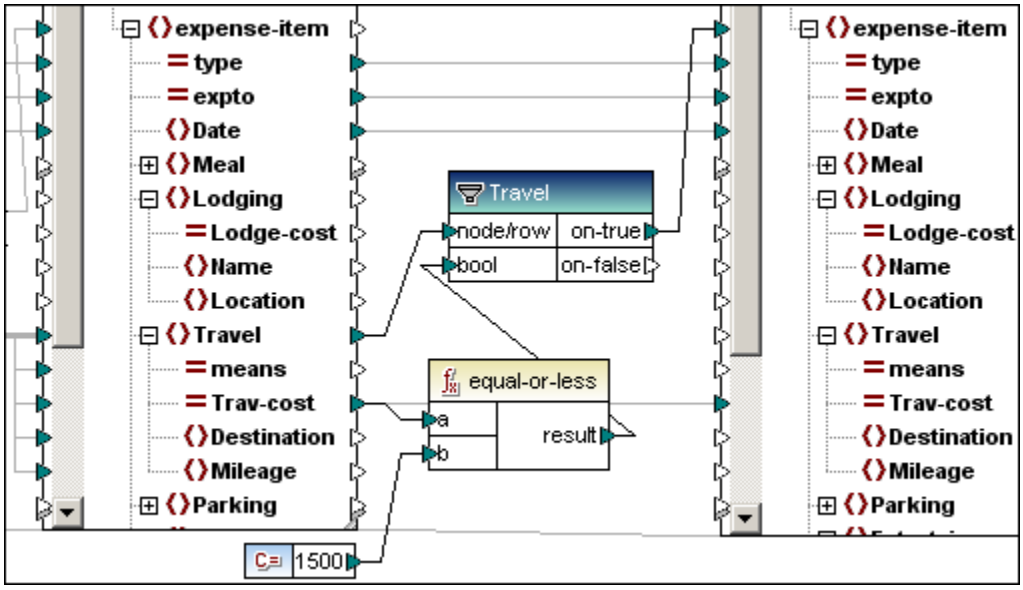
  Please note: Activating the pass-through button, automatically disables the *Input XML File* field, of the intermediate component, in the Component Settings dialog box. **File: (default)** is then seen at the top of the component.

- If the pass-through button (on component B) is **inactive**, an error message appears stating that the input file of the intermediate component could not be accessed when you check the output of component C. To make the input file of the intermediate component accessible, the result of the mapping of component A to B must be saved, and the resulting file name be placed in the *Input XML data* field of component B. Only then can data be displayed in the final component C.

Example: Tut-ExpReport-chain.mfd
This example file contains three components, as shown in the screenshot above. Component A and B filter out the Travel expense items from all the expense items.



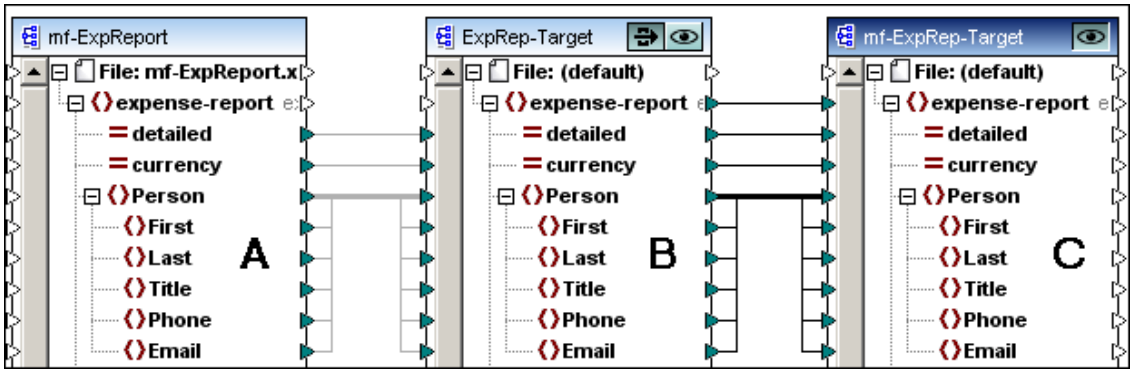
Components B and C work on the result set of A and B, and further filter out the expense items where the Travel cost is less than 1500.



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.

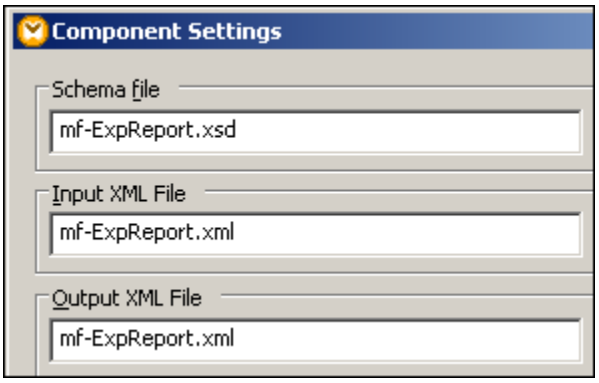
6.5.1 Chained mappings - Pass-through active

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:

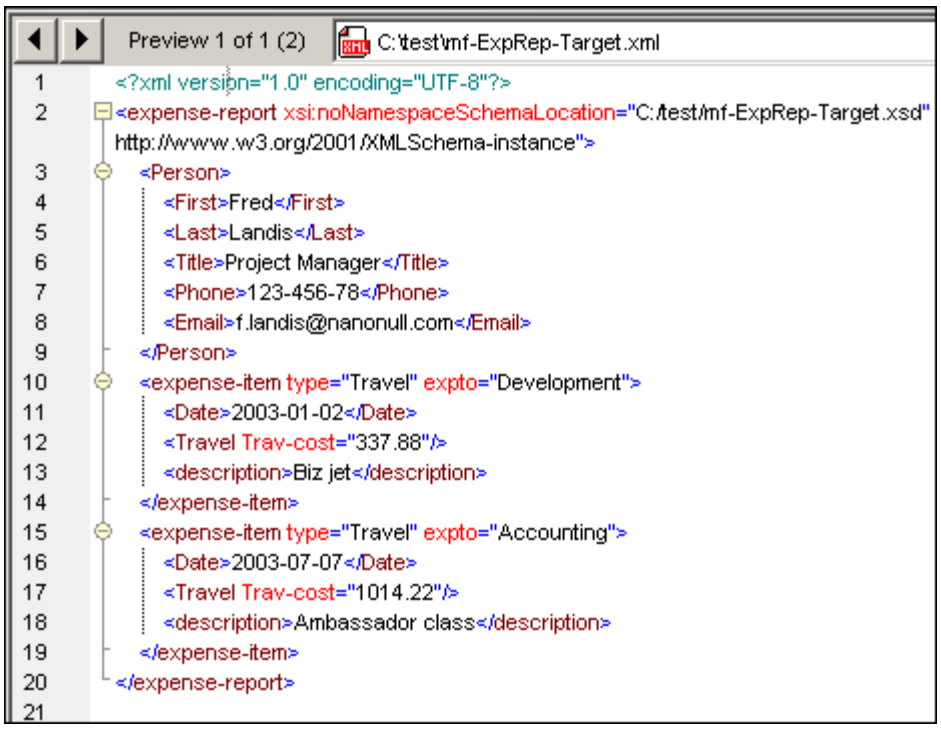
- **Component A** supplies all the mapping data, using a sample XML file. The selected working XML file (mf-ExpReport.xml) appears in the *Input XML File* field of the **Component Settings** dialog box. The Output XML File name is automatically inserted when you define an Input XML file.



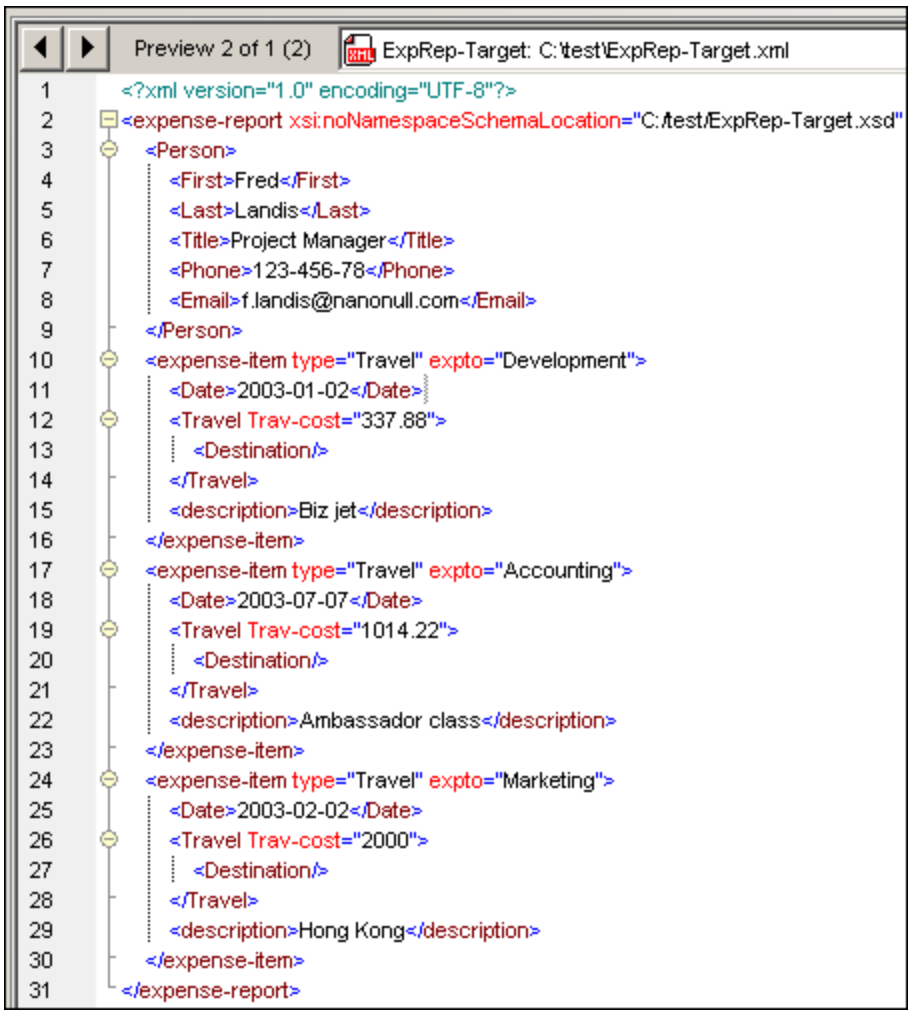
- Intermediate **component B** does **not** have an Input/Output XML File assigned to it **File: (default)**.
- Final **component C** does **not** have an Output XML File assigned to it **File: (default)**. and the preview button of component C is active.

Two separate sets of data are passed through to the Output Preview window via the intermediate component B:

I: The subset of mapping data from intermediate component B to target component C. I.e. the Travel expenses below 1500. Activate the preview button of component C to view this data.

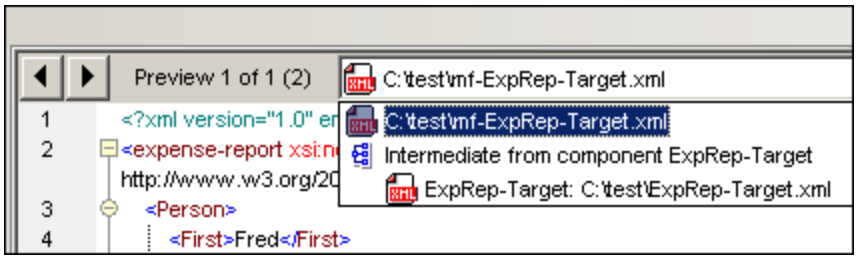


II: The result set of the mapping from component A to the intermediate component B. I.e. all Travel expense items. Activate the preview button of component B to view this data.



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

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

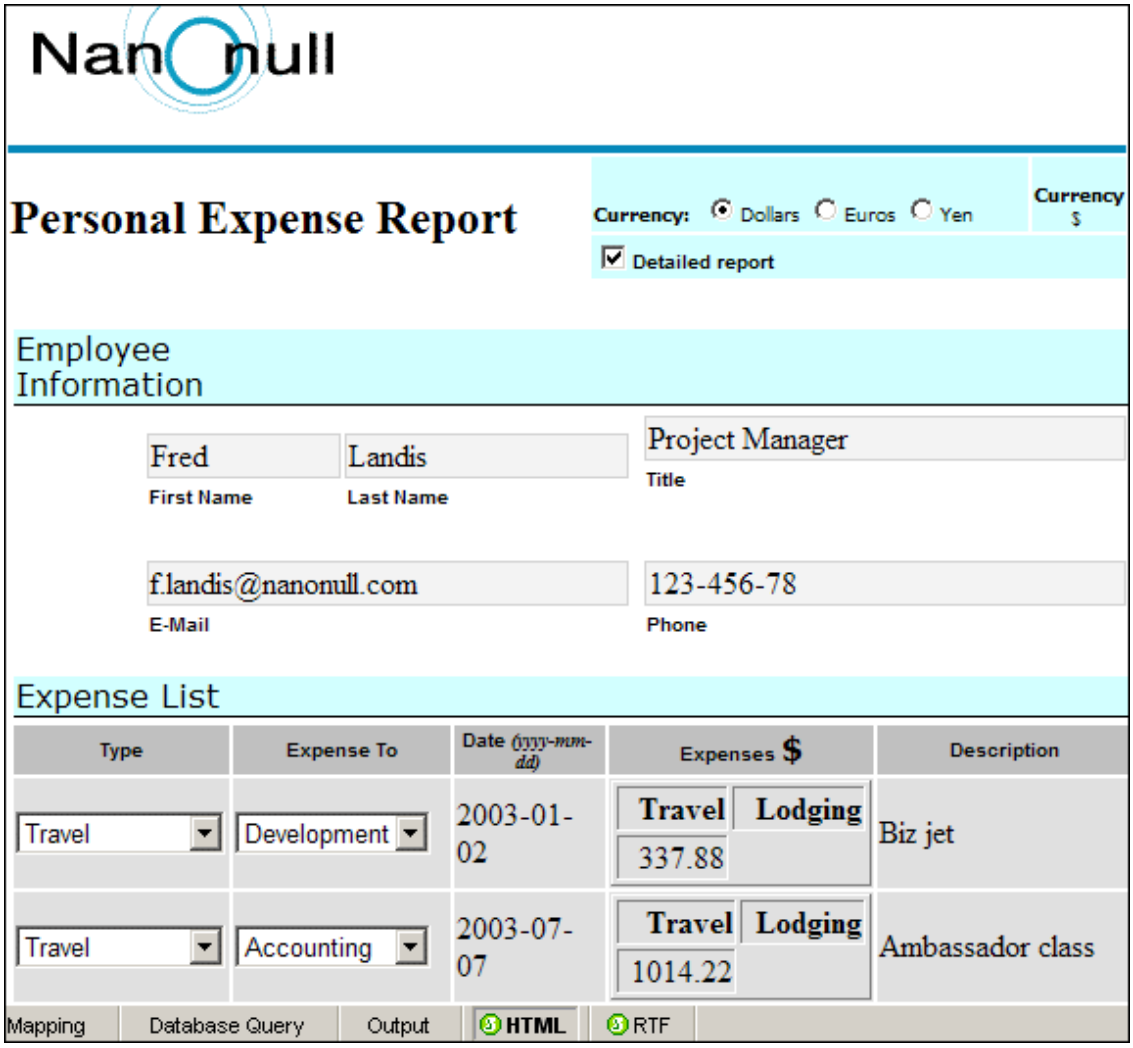


If pass-through is set as **active**, the files created by an intermediate component are either **automatically** saved as temp files or written directly to disk, and used for further processing of that components output. Please see Tools for more information.

The Preview XX **of 1** means the number of final targets from the selected target component,

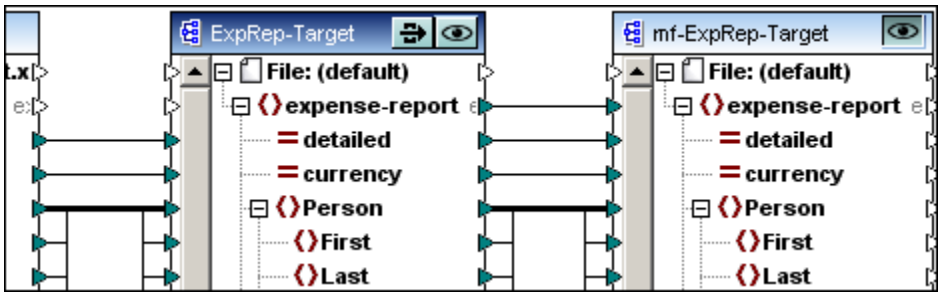
one in this case. The Preview ... **(2**) refers to the total number of results including all intermediate components.

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



6.5.2 Chained mappings - Pass-through inactive

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



To deactivate the pass-through button:

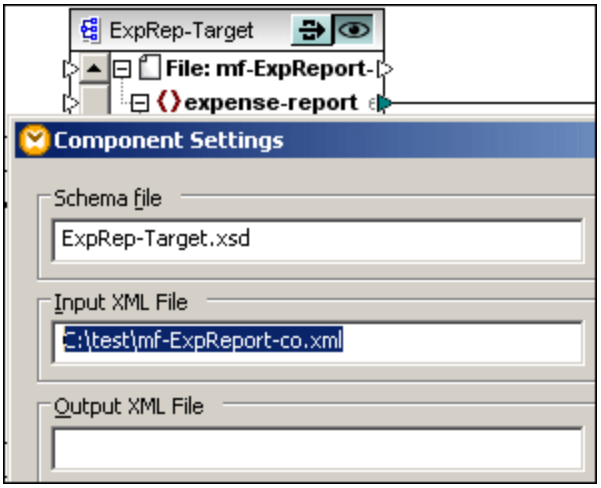
1. Deactivate the pass-through button of component B.
2. Check that the Preview button of component C is active.
3. Click the Output button to see the result of the mapping.
   An error message appears in the Message pane stating that the input file of the intermediate component could not be accessed.

Saving the intermediate mapping result

To make the input file of the intermediate component accessible, the result of the mapping of component A to B must be saved, and the resulting file name be placed in the *Input XML data field* of component B. Only then can data be displayed in the final component C.

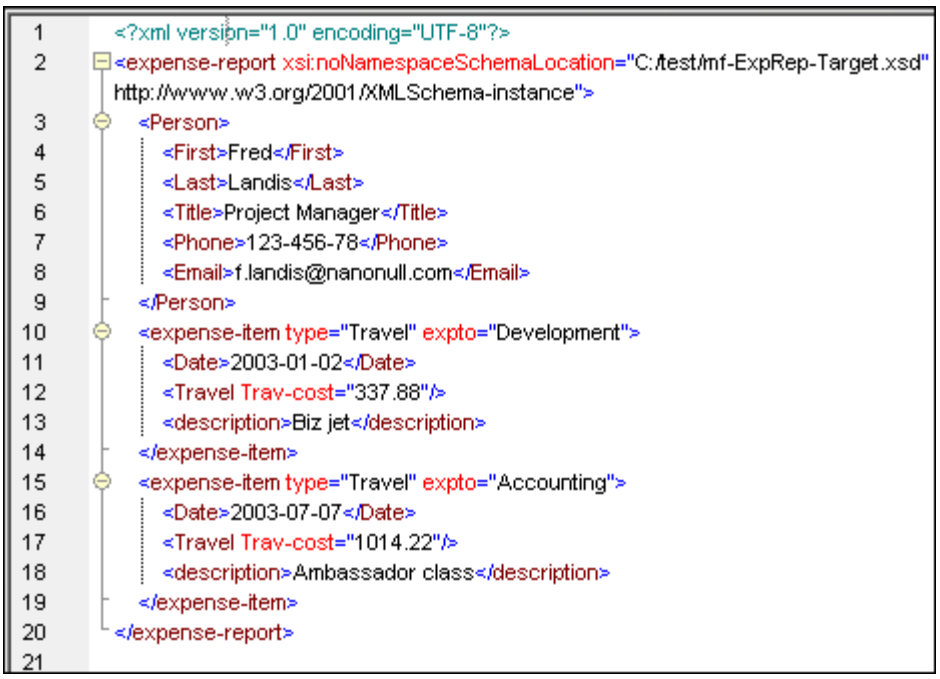
To save the intermediate mapping result to a file:

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 title 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** to occur.

4. Click the Preview button of the final target component C, then click the Output button to see the result.



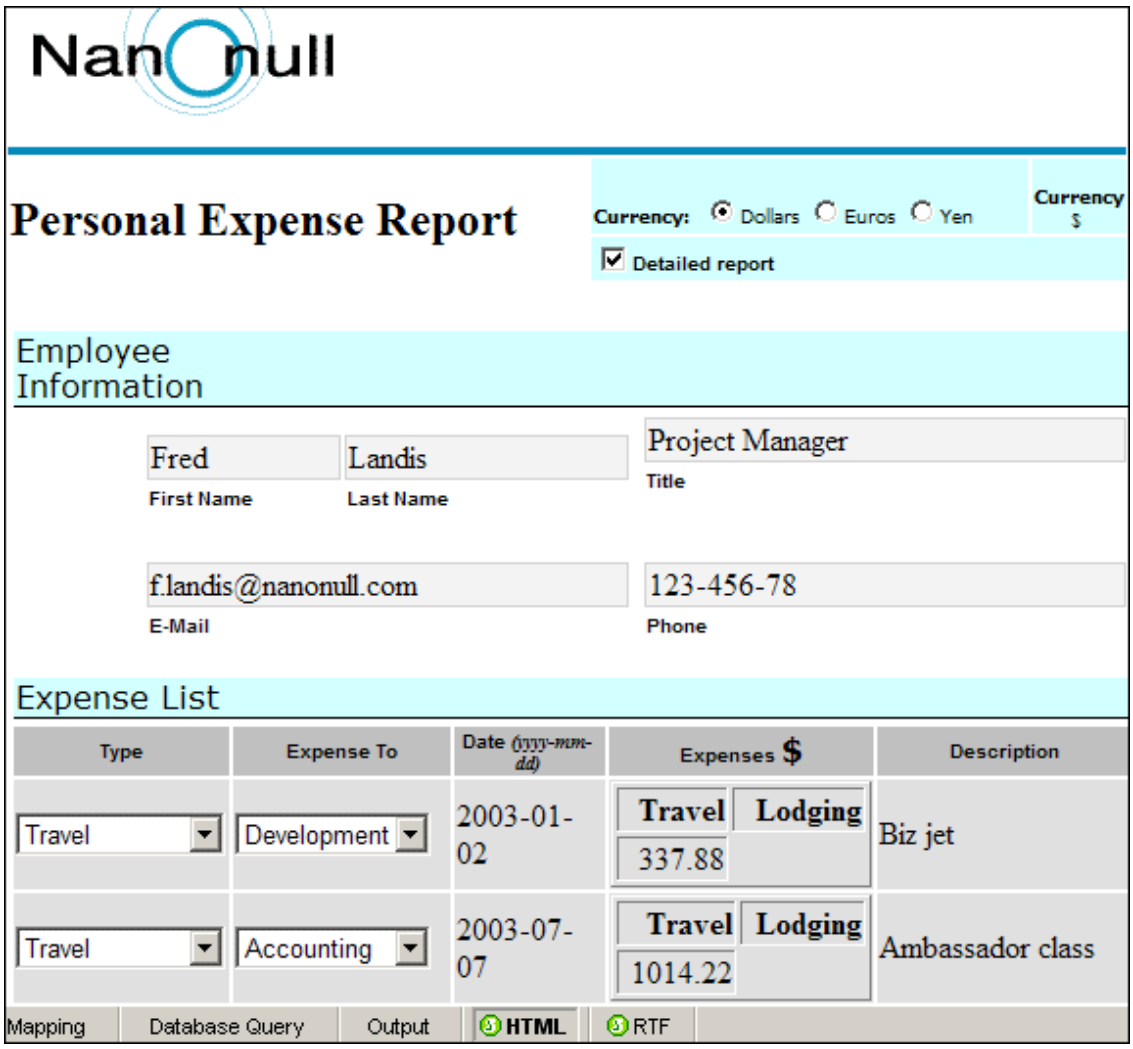
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <expense-report xsi:noNamespaceSchemaLocation="C:/test/mf-ExpRep-Target.xsd"
   http://www.w3.org/2001/XMLSchema-instance">
3    <Person>
4      <First>Fred</First>
5      <Last>Landis</Last>
6      <Title>Project Manager</Title>
7      <Phone>123-456-78</Phone>
8      <Email>f.landis@nanonull.com</Email>
9    </Person>
10   <expense-item type="Travel" expto="Development">
11     <Date>2003-01-02</Date>
12     <Travel Trav-cost="337.88"/>
13     <description>Biz jet</description>
14   </expense-item>
15   <expense-item type="Travel" expto="Accounting">
16     <Date>2003-07-07</Date>
17     <Travel Trav-cost="1014.22"/>
18     <description>Ambassador class</description>
19   </expense-item>
20 </expense-report>
21
```

The mapping result from component B to C is displayed. I.e. the Travel expenses below 1500.

Please note: Activating the pass-through button, automatically **disables** the *Input XML File* field, of the intermediate component, in the **Component Settings** dialog box. **File: (default)** is then seen at the top of the component.

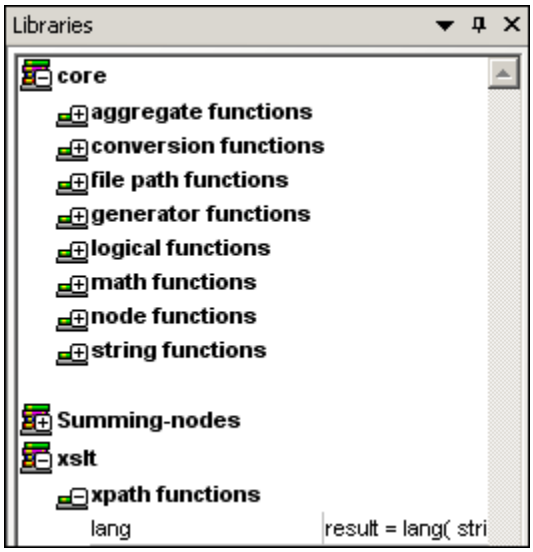
Data in the Preview tabs can be validated and saved.

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



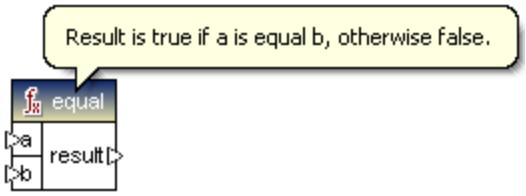
6.6 Using Functions

The functions available in the selected language (XSLT/XSLT2, BUILTIN XQ, Java, C#, or C++) are displayed in the Libraries window. The expand and contract icons show, or hide the functions of that library.



Function tooltips

Explanatory text (visible in the libraries pane) on individual functions can be toggled on/off by clicking the **Show tips** icon in the title bar. Placing the mouse pointer over a function header, displays the information on that function.



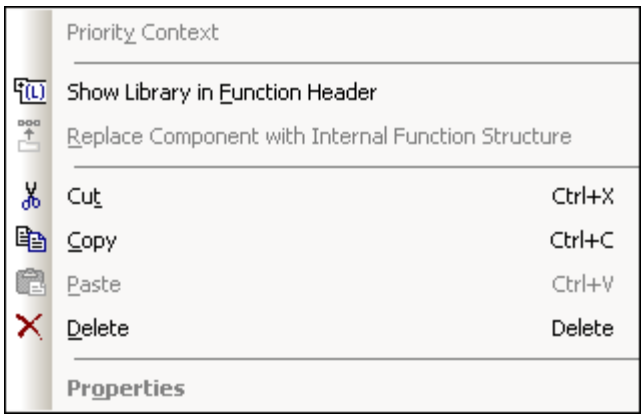
To use a function in Mapping window:

1. Select the **programming language** you intend to generate code for, by clicking one of the output icons in the title bar.
2. Click the **function name** and drag it into the Mapping window.
3. Use drag and drop to connect the input and output parameters to the various icons.

Note that placing the mouse pointer over the "result = xxx" expression in the library pane, displays a ToolTip describing the function in greater detail.

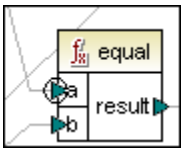
Function context menu

Right clicking a function in the Mapping window, opens the context window.



Priority Context

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. A circle appears around the icon so designated. Please see Priority Context in the Reference section, for an example.



Show Library in Function Header

Displays the library name in the function component.

Replace Component with Internal Function Structure

Replaces the user-defined component/function with its constituent parts, not available for functions.

Cut/Copy/Paste/Delete

The standard MS Windows Edit commands, allow you to cut, copy etc., any components or functions visible in the mapping window. All connectors will be retained except for those which would have to be replaced.

Properties

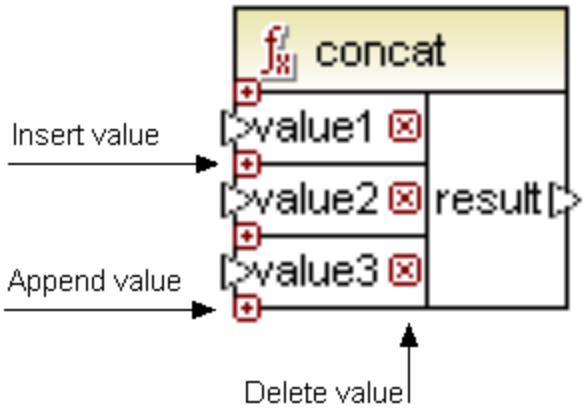
Not available for functions.

Extendable functions

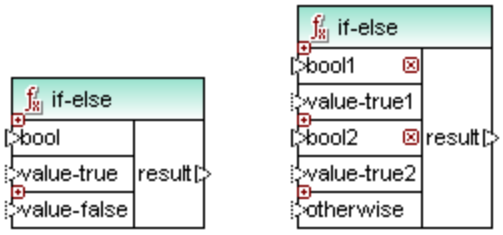
Several functions available in the function libraries are extendable: for e.g. the concat, "logical-and", "logical-or", and IF-ELSE functions. The parameters of these types of function can be inserted/appended and deleted at will.

Clicking the "plus" icon inserts or appends the same type of parameter, while clicking the check mark deletes the parameter.

Please note: "dropping" a connector on the "plus" symbol, automatically inserts/appends the parameter and connects it.



The IF test parameters, of the IF-Else function can be extended in the same way.



6.7 Previewing the transformation result

The Built-in execution engine allows you to immediately preview and save the result of a transformation, without having to go the path of generating program code, compiling it, and viewing the results.

This is achieved by simply clicking the **Output** button irrespective of the output language that you select. The Output button also supports the **Find** command, enabling you to locate any XML data, or SQL statement that you might need to find.

The **BUILTIN** icon allows you to explicitly select the Built-in execution engine as the Output target. The Built-in execution engine is also used to preview the result if one of the programming languages (Java, C#, C++) is selected. Having clicked the BUILTIN icon and clicked the Output button, all the standard options of the Output window are available, i.e. **Validate Output File**, **Save Output File...**, **Save All Output Files...**, etc.

MapForce can also be started from the command line and produce a result using the /BUILTIN parameter, without having to generate any intermediate code.

Streaming
The Built-in execution engine is the only target that supports XML, CSV, and FLF **streaming**. What this means is that the **output** files can be of an unlimited size. If the output file is very large, a **Load more...** button appears at the bottom of the Output pane. Clicking the button, appends a large chunk to the currently visible data. The **Pretty-print** button becomes active when the complete file has been loaded into the output pane.

Digital Signatures:
MapForce supports creating XML digital signatures for XML and XBRL output files. Digital signatures can only be generated when the output target is BUILTIN. A signature is created for the generated result file, when the output button is pressed, and the result file is saved. Please see: Digital signatures for more information.

Previewing the transformation output
In the **Output** menu, or by selecting one of the icons in the Language Selection toolbar (*screenshot below*), you can define which language MapForce should use to transform the data of the input file.

The following options are available:

- XSLT
- XSLT2

XSLT code can be previewed in the XSLT/XSLT2 tab, and is generated and saved using the **File | Generate code in | XSLT 1.0** or **XSLT 2.0** menu option, respectively. .

- XQuery

Please note that execution speed of generated XQuery 1.0 code is significantly faster that of generated XSLT 1.0 / 2.0 code.

- Java
- C#
- C++

Generated program code such as Java, C#, or C++, is of course even quicker than XQuery,

because it is compiled before execution.

Previewing the program code

Before generating program code it is a good idea to preview the result of the XSLT/XQuery using the Built-in execution engine.

To preview an XSLT or XQuery result:
Having opened the *.mfd file in MapForce:

1. Do one of the following:
   - Click the **XSLT** button to preview the generated XSLT 1.0 code.
   - Click the **XSLT2** button to preview the generated XSLT 2.0 code.
   - Click the **XQuery** button to preview the generated XQuery code.
2. Click the **Output** button to preview the result of the mapping

Generating XSLT and XQuery code

XSLT 1.0 and XSLT 2.0 program code generated by MapForce can be executed using Altova's XML engine which is included in the AltovaXML package, or opened directly in XMLSpy and executed using the menu option **XSL/XQuery | XSL Transformation**.

MapForce generates XQuery 1.0 program code which can be executed using Altova's XQuery engine which is included in the AltovaXML package, or opened directly in XMLSpy and executed using the menu option **XSL/XQuery | XQuery Execution**. To generate and save XQuery code in MapForce, use the **File | Generate code in | XQuery** menu option.

To download the AltovaXML package:
1. Point your Browser to http://www.altova.com/download_components.html.
2. Select and install one of the following:
   - Altova XSLT 1.0 or XSLT 2.0 engine.
   - Altova XQuery engine.

To generate XSLT code:
1. Select the menu option **File | Generate code in | XSLT 1.0 (or XSLT 2.0)**.
2. Browse for the folder where you want to save the XSLT file.
   A message appears when the generation was successful.
3. Navigate to the previously defined folder, where you will find the generated XSLT file.

To generate XQuery code:
1. Select the menu item **File | Generate code in | XQuery**.
2. Select the folder you want to place the generated XQuery file in, and click **OK**.
   A message appears when the generation was successful.
3. Navigate to the previously defined folder where you will find the generated XQuery file.

To generate program code:
1. Select the specific menu option: **File | Generate code in | Java, C# (Sharp), C++**
2. Browse for the folder where you want to save the program files.
   A message appears when the code generation was successful.
3. Compile and execute the code using your specific compiler.

Please note: Project files for various IDEs and build systems are generated automatically by MapForce. For details, see the section on JDBC driver setup as well as the code generator section for more information.

Transforming the XML source file using the generated XSLT or XQuery

When you generate XSLT or XQuery code, a batch file named DoTransform.bat, which uses AltovaXML to transform the XML file, is also generated and saved to the folder you have specified for the XSLT or XQuery file, respectively.

To transform the XML file using the generated XSLT/XQuery:

1. Download and install the free AltovaXML engine from the AltovaXML download page. AltovaXML is installed to C:\Program Files\Altova\AltovaXML2011\ per default.
2. Start the **DoTransform.bat** batch file located in the previously designated output folder. This generates the output XML file in the current folder.

Note: You might need to add the AltovaXML installation location to the path variable of the Environment Variables.

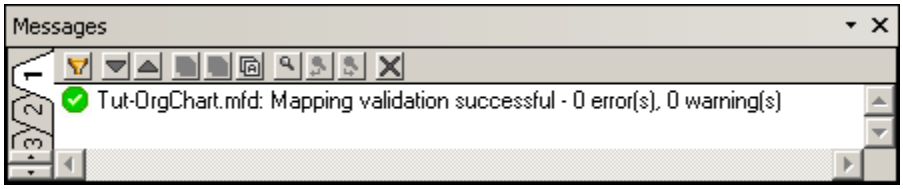
6.8 Validating mappings and mapping output

It is not mandatory for functions or components to be mapped. The Mapping tab is a work area where you can place any available components. XSLT 1.0, XSLT 2, XQuery, Java, C#, or C++ code is only generated for those components for which valid connections exist.

Free standing components do not generate any type of error or warning message.

Partially connected components can generate two types of warning:

- If a function component **input icon** is unconnected, an error message is generated and the transformation is halted.
- If the function **output icon** 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.



You can use multiple message tabs if your project contains many separate mapping files. Click one of the numbered tabs in the Messages window, and click the preview tab for a different mapping in your project. The validation message now appears in the tab that you selected. The original message in tab 1, is retained however.

Use the different icons of the Messages tab to:

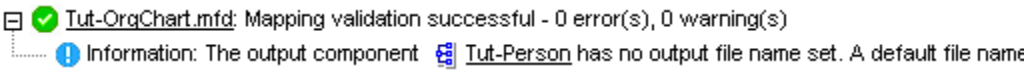
- Filter the message types, errors or warnings
- Scroll through the entries
- Copy message text to the clipboard
- Find a specific string in a message
- Clear the message window.

To validate a mapping:

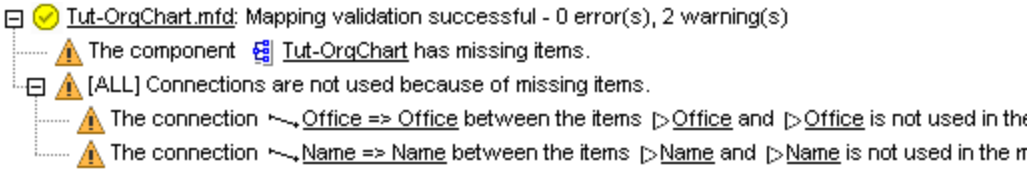
- Click the **Validate Mapping** icon in the application toolbar, or select the menu item **File | Validate Mapping**.
  A validation message appears in the Messages window.

Validation messages
Validation messages are displayed in the Messages window and indicate whether or not the validation was successful. In addition, error messages, warnings, and information on the mapping is displayed. Two types of validation messages can appear:

- Validation successful - 0 error(s), n warning(s)

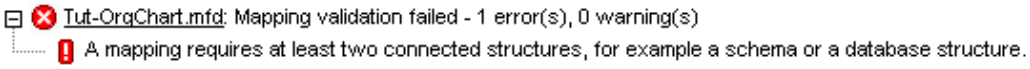


  Warnings alert you to something, while still enabling the mapping process and preview of the transformation result to continue. It is therefore possible for a mapping to have 0 errors and n warnings.



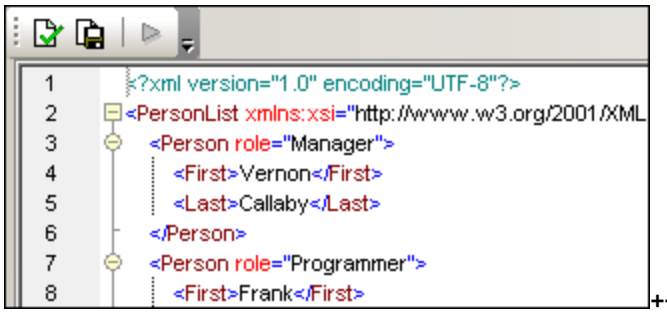
- Validation failed - x error(s), y warning(s)

  Errors, halt the transformation process and deliver an error message. An XSLT, XQuery, or Output preview is not possible when an error of this type exists. Clicking a validation message in the Messages window, highlights the offending component icon in the Mapping window.



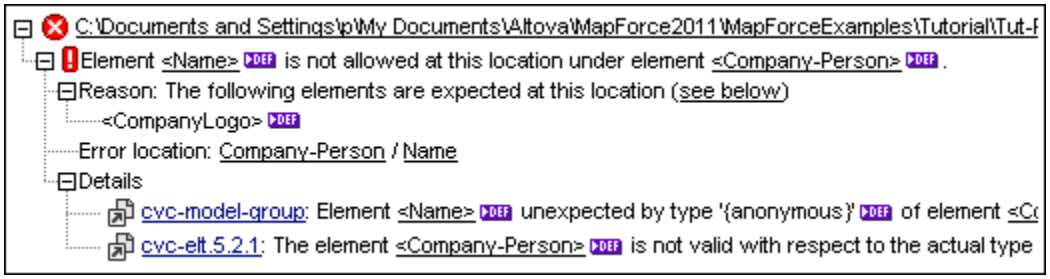
Validating the mapping output
Clicking the Output tab uses the MapForce, XSLT 1.0/2.0 or XQuery engine, to transform the data and produce a result in a Text view.

If the data is mapped to an XML Schema, then the resulting XML document can be validated against the underlying schema.



+++

The result of the validation is displayed in the Messages window. If the validation was not successful, the message contains detailed information on the errors that occurred (*see screenshot below*).

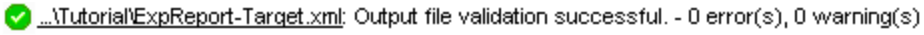


The validation message contains an number of hyperlinks you can click for more detailed information:

- 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.

To validate the mapping OUTPUT:

- Click the **Validate** button to validate the document against the schema. An "Output file validation successful. 0 error(s), 0 warning(s)" message, or a message detailing any errors appears.



Please note:
The entry in the **Add Schema/DTD reference** field of the component settings dialog box allows you to add the path of the referenced XML Schema file to the root element of the XML output.

The path 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.

6.9 Loops, groups and hierarchies

There are several ways of looping through source and target hierarchies which allow you to define how you want to loop, or group, sets of data.

The following links show you how this can be achieved. Please note that these examples are not sequential, they appear in various locations within the documentation.

Mapping XML to CSV, or fixed length text files

Mapping CSV files to XML

Creating hierarchies from CSV and fixed length text files

Value-Map - lookup table

Mapping multiple tables to one XML file

Using MapForce to create database relationships

Priority Context

Sequences, cardinality and priority context
MapForce generally maps data in an intuitive way, but there are some scenarios where the resulting output seems to have too many, or too few items. This is what this section will cover.

General rule
Generally, every connection between a source and target item means: for each source item, create one target item.

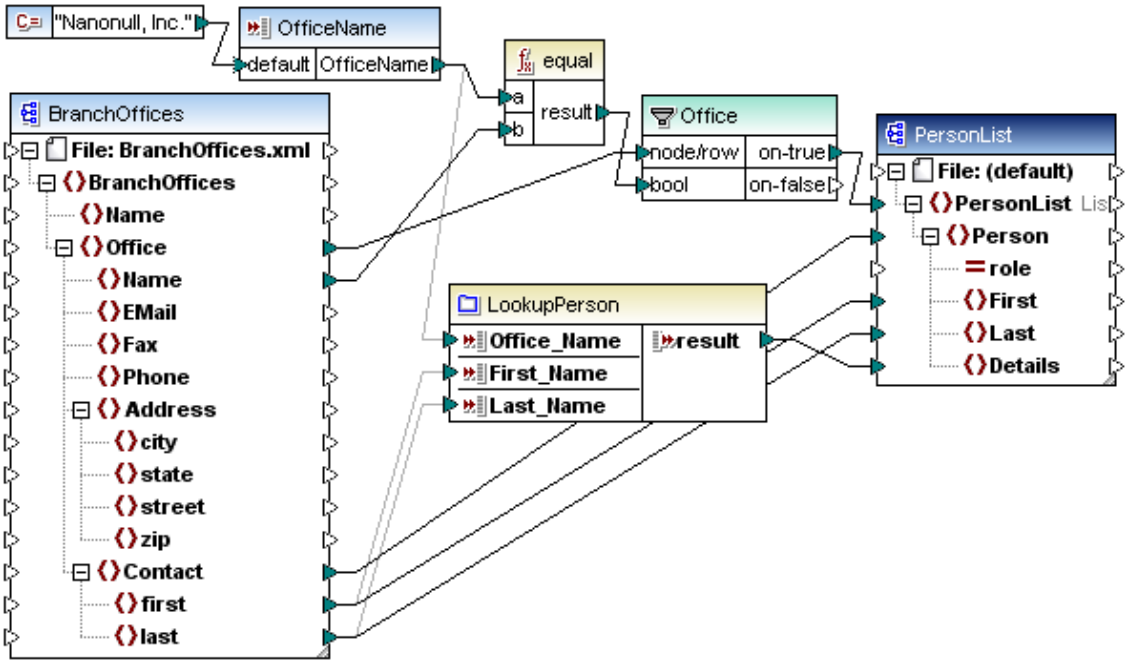
If the source node contains simple content (e.g. string, integer etc.) and the target node accepts simple content, then the content is copied, and the data type is converted if necessary.

There are some exceptions to this rule, but it generally holds for all connections.

The Context and "current" items
MapForce displays the structure of a schema, database file as a hierarchy of mappable items in the component. Each of these nodes may have many instances (or none) in the instance file or database.

Example: If you look at the source component in PersonListByBranchOffice.mfd, there is only a single node **first** (under **Contact**). In the **BranchOffices.xml** instance file, there are multiple **first** nodes and **Contact** nodes having different content, under different **Office** parent nodes.

It depends on the current **context** (of the **target** node) which source nodes are actually selected and have their data copied, via the connector, to the target component/item.



This context is defined by the **current target node** and the connections to its ancestors:

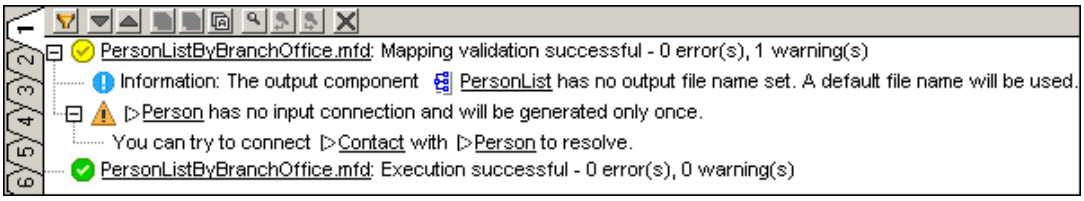
- Initially the context contains only the source components, but no specific nodes. When evaluating the mapping, MapForce processes the **target root** node first (PersonList), then works down the hierarchy.
- The connector to the **target** node is traced back to all source items directly or indirectly 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 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. This also applies to SQL-Where components, which integrate the filtering commands directly into the database query. Filter components (acting on a database component) with simple conditions are usually optimized into a WHERE clause automatically.

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 lo ops 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.

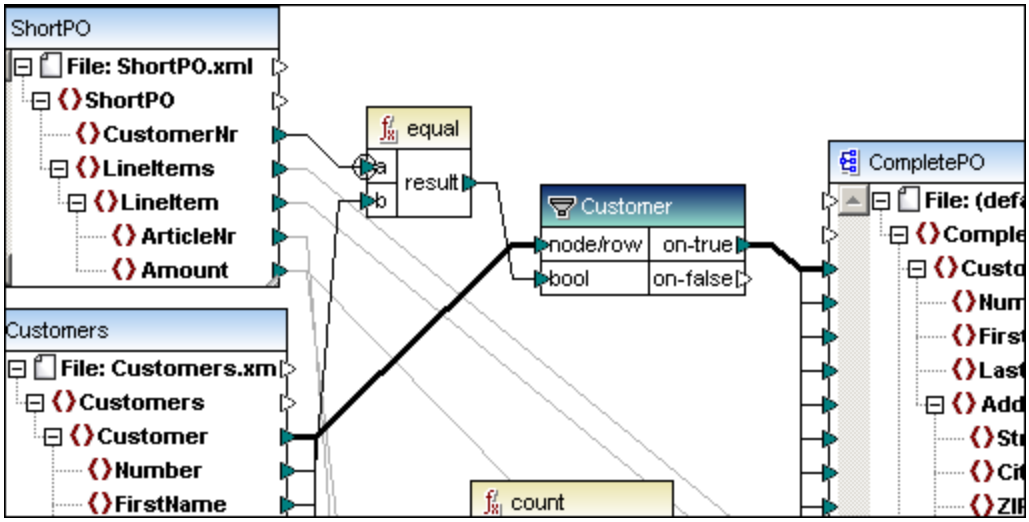
Functions with **sequence inputs** are the only functions that can produce a result if the input sequence is **empty**:

- "**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.

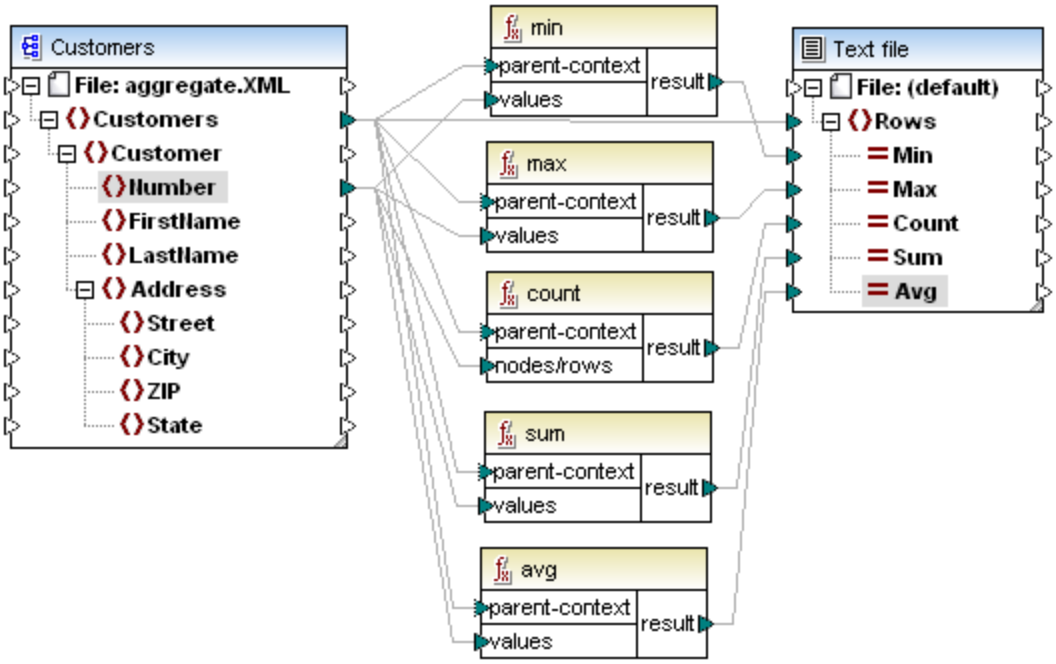
In functions connected to the bool input of **filter** conditions, the priority context affects not only the comparison function itself but also the evaluation of the filter, so it is possible to join together two source sequences (see CompletePO.mfd, CustomerNo and Number).



In this example, the priority context forces ShortPO/CustomerNr to be evaluated before iterating and filtering the Customer nodes from the Customers component. 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.

Exceptions to the general rule (for each source item, create one target item)

- A target XML root element is always created once and only once. If a sequence is connected to it, only the contents of the element will be repeated, but not the root element itself. The result may, however, 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 the root element. To create multiple output files, connect the sequence to the "File" node instead, via some function that generates file names.

- Some other nodes, like XML attributes, database fields or the output component inside a user-defined function, accept only a single value.

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.

Chapter 7

Built-in execution engine

7 Built-in execution engine

The Built-in execution engine allows you to immediately preview and save the result of a transformation, without having to go the path of generating program code, compiling it, and viewing the results.

The **BUILTIN** icon  allows you to explicitly select the Built-in execution engine as the Output target. The Built-in execution engine is also used to preview the result if one of the programming languages (Java, C#, C++) is selected.

MapForce can also be started from the command line and produce a result using the /BUILTIN parameter, without having to generate any intermediate code.

Built-in execution engine - Streaming:
The Built-in execution engine is the only target that supports XML, CSV, and FLF **streaming**. What this means is that the **output** files can be of an unlimited size. If the output file is very large, a "Load more..." button appears at the bottom of the output tab. Clicking the button, appends are large chunk to the currently visible data.

The pretty-print button becomes active when the complete file has been loaded into the output tab.

Digital Signatures:
MapForce supports creating XML digital signatures for XML and XBRL output files. Digital signatures can only be generated when the output target is BUILTIN. A signature is created for the generated result file, when the output button is pressed, and the result file is saved.

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.

Please see: Digital signatures for more information.

Chapter 8

Global Resources

8 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 Mission kits 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).

8.1 Global Resources - Files

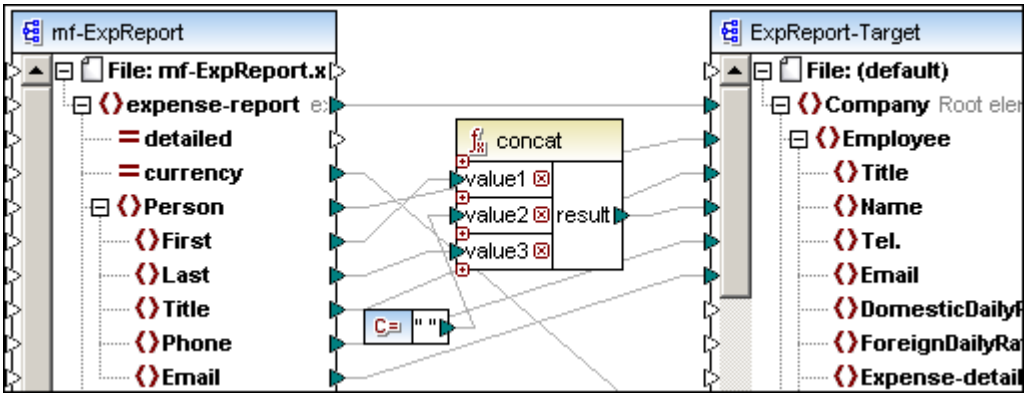
Global Resources in MapForce:

- Any input/output components **files** can be defined as global resources, e.g. XML, XML Schema, Text/CSV, database, 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.

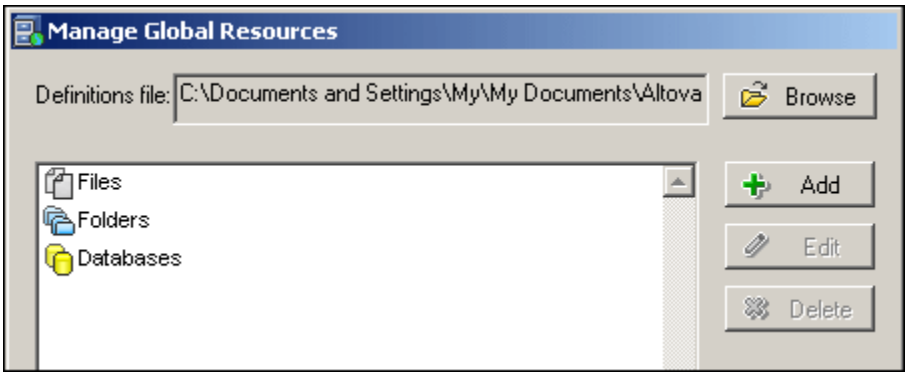
  This section uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples\Tutorial\ folder.



8.1.1 Defining / Adding global resources

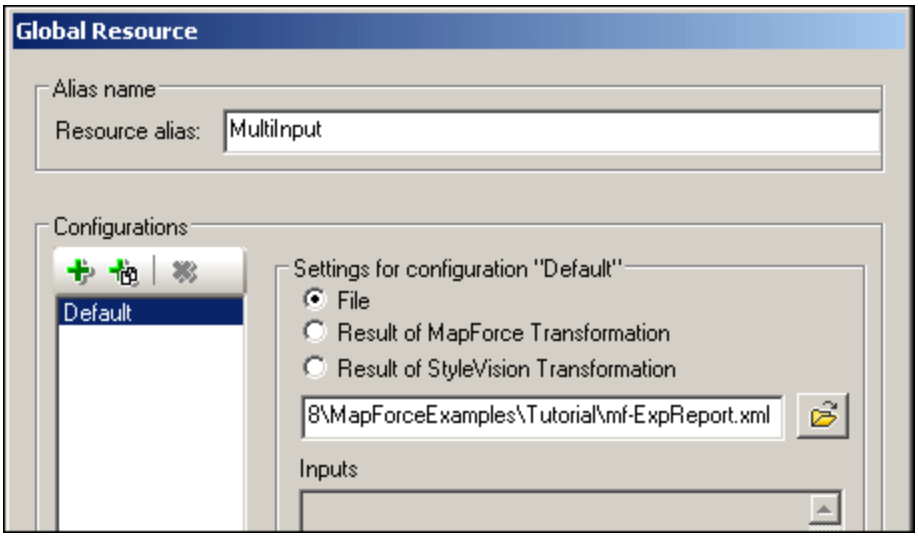
Defining / Adding a global resource file

1. Click the Global Resource icon to open the dialog box.

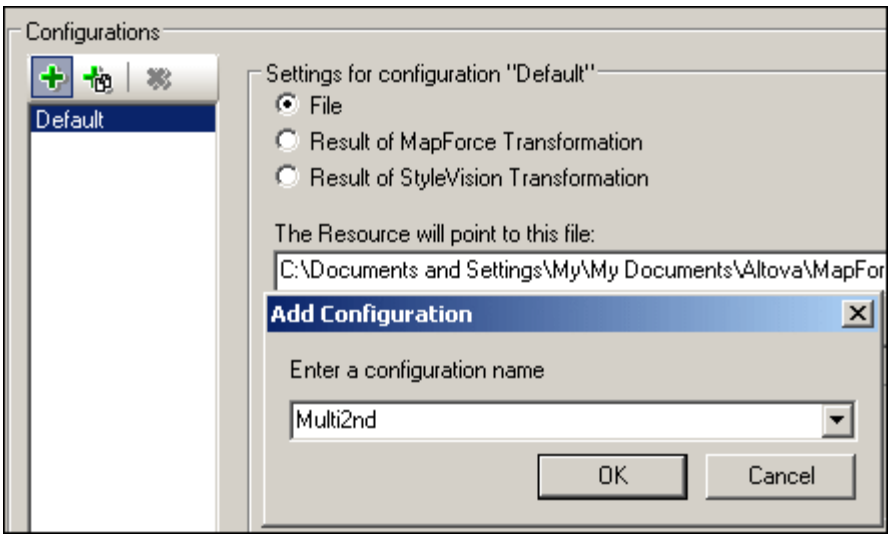


This is the state of an empty global resources file.

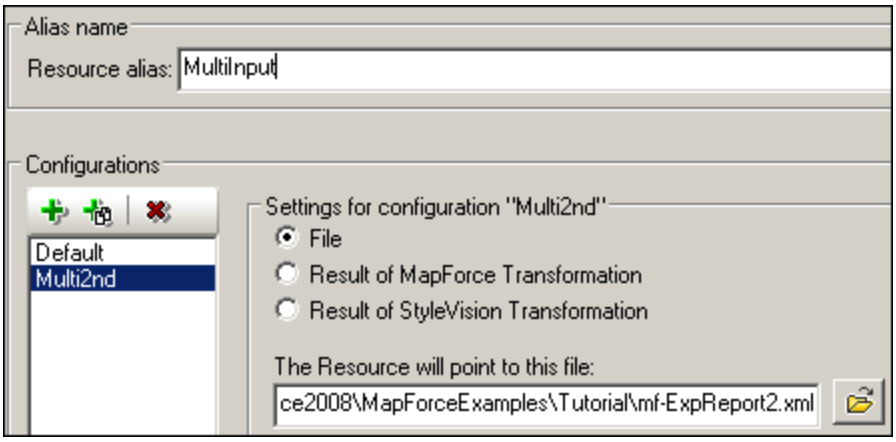
2. Click the **Add** button and select **File** from the popup.
3. Enter the name of the Resource alias e.g. **MultiInput**.
4. Click the Open folder icon and select the XML file that is to act as the "Default" input file e.g. **mf-ExpReport.xml**.



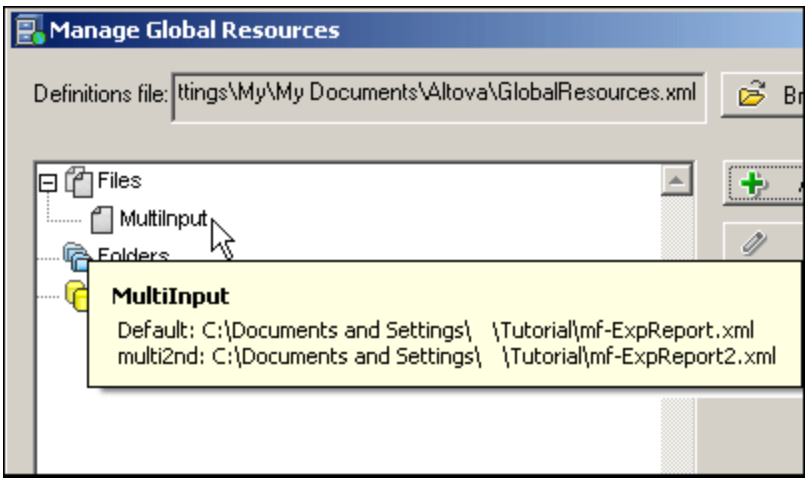
5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias. Note that the Copy configuration icon, allows you to copy a selected configuration and save it under a new name.



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.

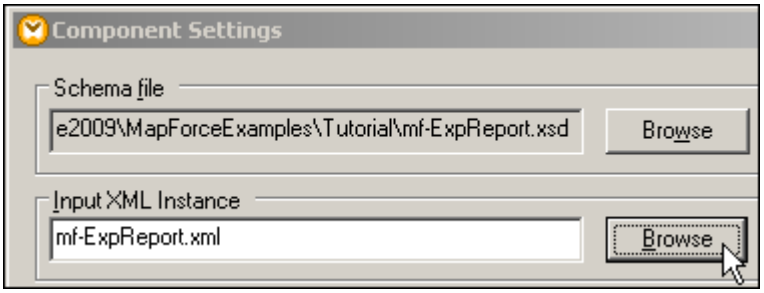


9. Click OK to confirm.
   This concludes the definition part of defining global resources. The next step is Assigning a global resource to a component.

8.1.2 Assigning a global resource

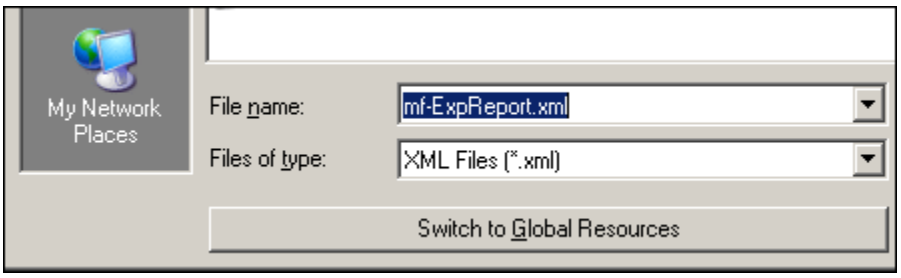
Assigning global resources to a component
We now have to assign the global resource to the component that is to make use of it.

1. Double click the **mf-ExpReport** component and click the Browse button next to the Input XML-Instance field.

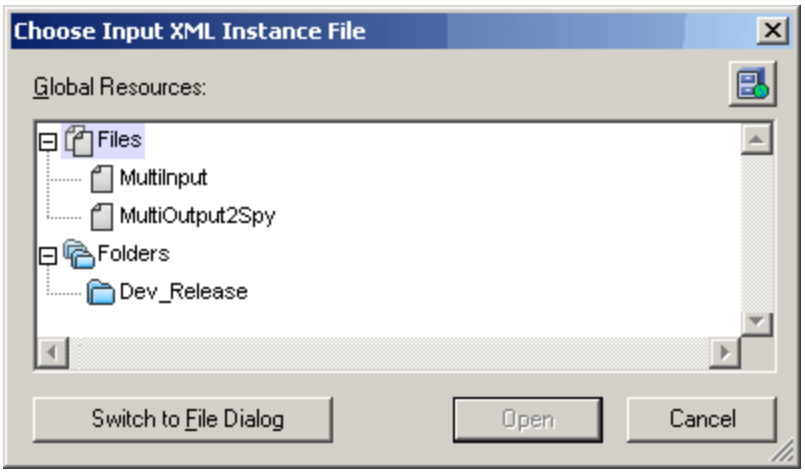


This opens the "Choose XML Instance file" dialog box.

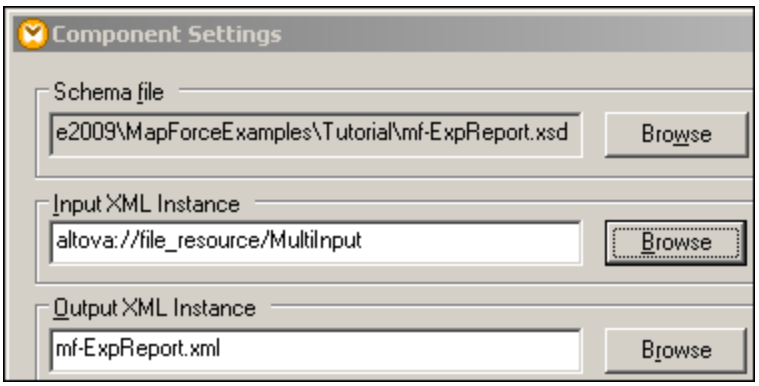
2. Click the **Switch to Global Resources** button at the base of the dialog box.



3. Click the resource you want to assign, **MultiInput** in this case, and click OK.



The **Input XML-Instance** field now contains a reference to a resource i.e. **altova://file_resource/MultiInput**.



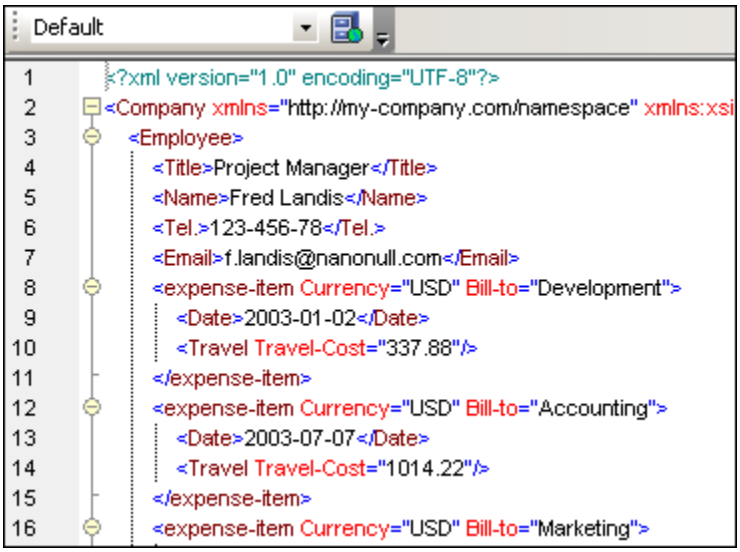
4. Click OK to complete the assignment of a resource to a component.
   The next step is Using / activating a global resource.

8.1.3 Using / activating a global resource

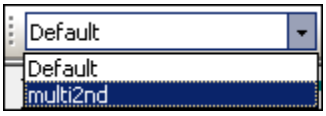
Using / activating a global resource

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".

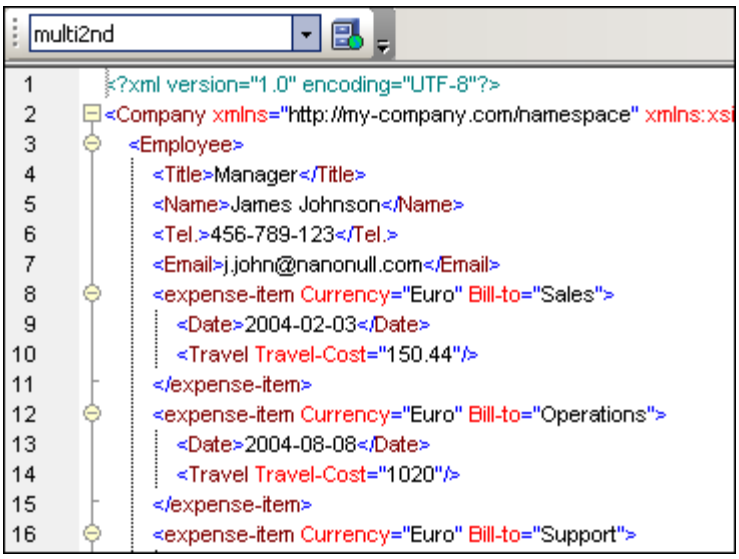
1. Click the Output tab to see the result of the mapping.



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.

8.2 Global Resources - Folders

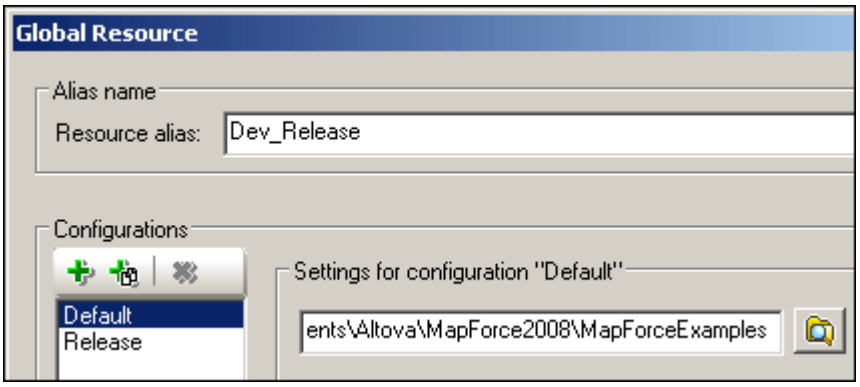
Folders can also be defined as a global resource, which means that input components can contain files that refer to different folders, for development and release cycles for example.

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.

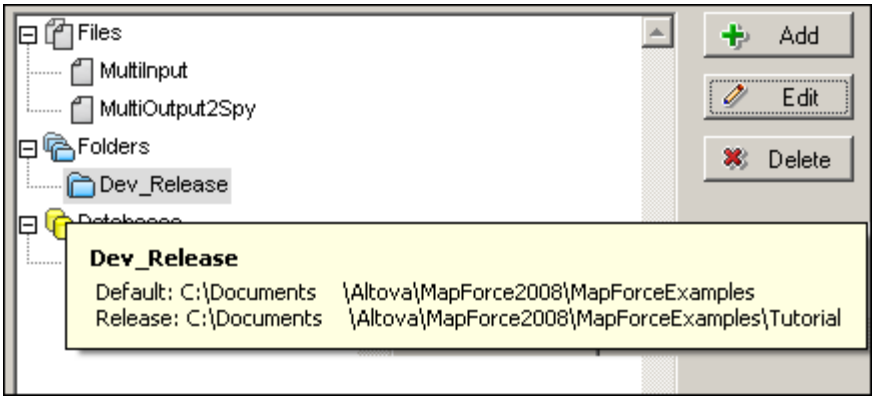
The mapping file used in this section is available as "**global-folder.mfd**" in the ...\MapForceExamples\Tutorial\ folder.

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.



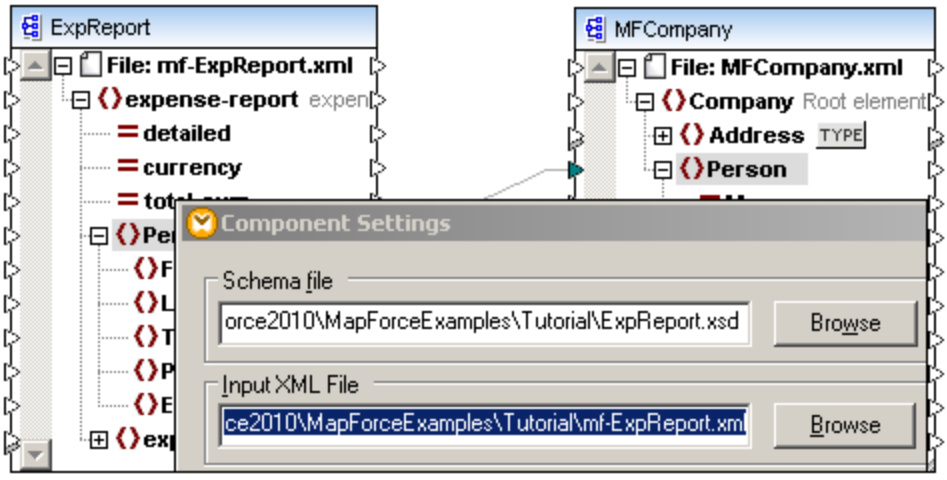
5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias, and enter a name for it e.g. Release. Note that the Copy configuration icon, allows you to copy a selected configuration and save it under a new name.
6. Click the Open folder icon and select the Release input folder, ...\MapForceExamples\Tutorial.



7. Click OK to finish the global folder definition.

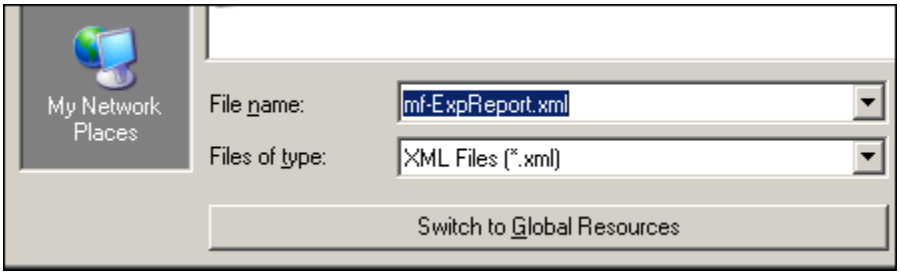
Assigning the global resource folders:

1. Double click the **ExpReport** component and click the **Browse** button next to the **Input XML File** field.

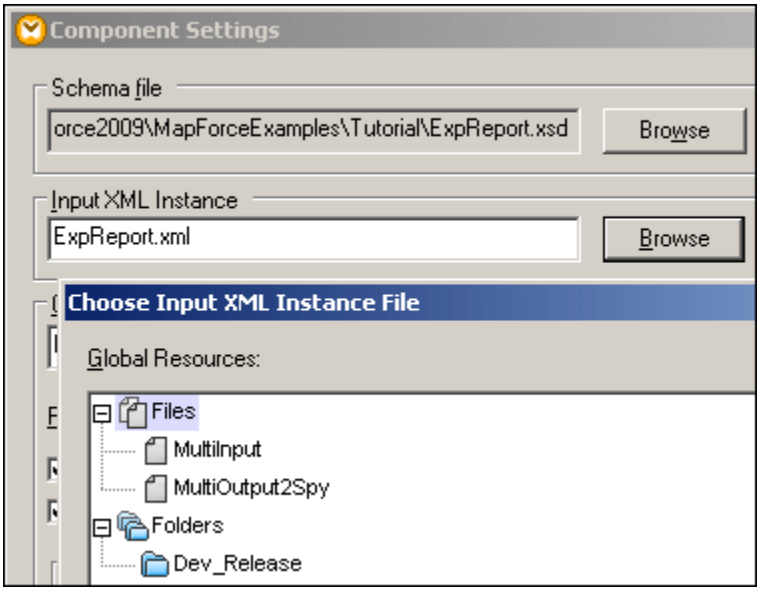


This opens the "Choose XML Instance file" dialog box.

2. Click the **Switch to Global Resources** button at the base of the dialog box.

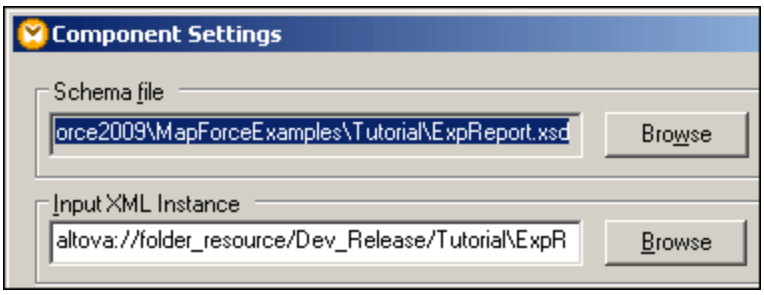


3. Click the resource you want to assign, **Dev_Release** in this case, and click OK.



The "Open..." dialog appears.

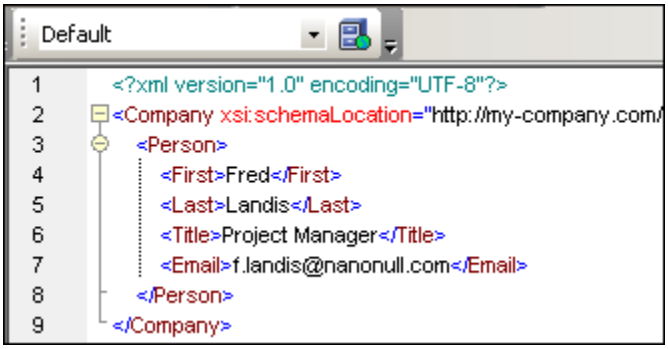
4. Select the **file** name that is to act as both the Development and Release **resource** file in each of the folders, e.g. **ExpReport.xml** and click OK to finish assigning the resource folder.



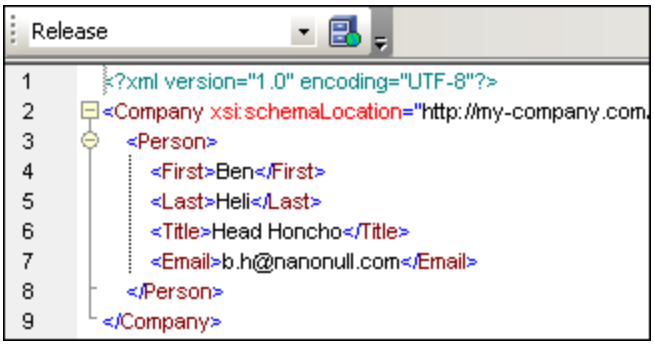
Note that this file is available in both directories but has different content.

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.

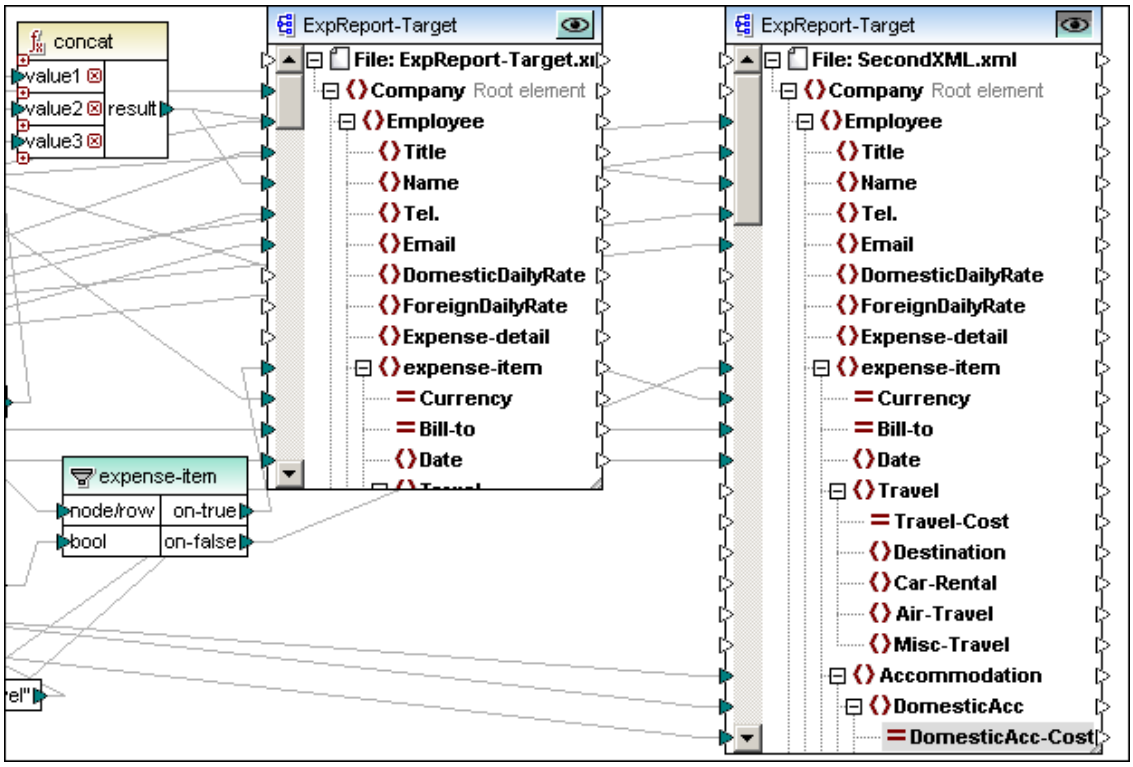


The output from the "Release" folder .../MapforceExamples/Tutorial is now displayed.

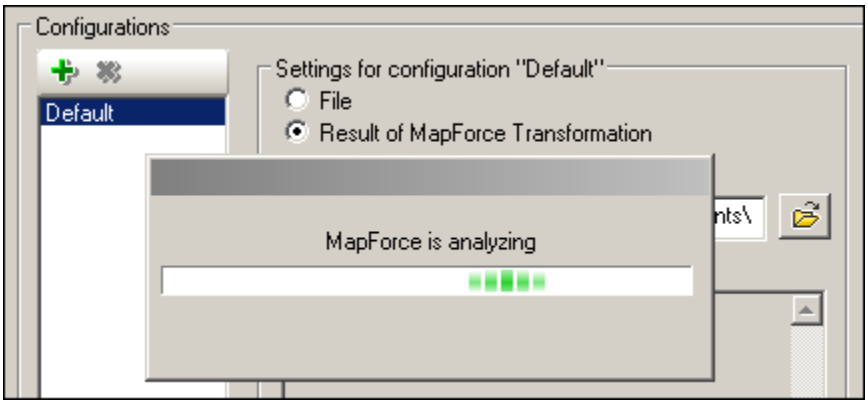
8.3 Global Resources - Application workflow

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.

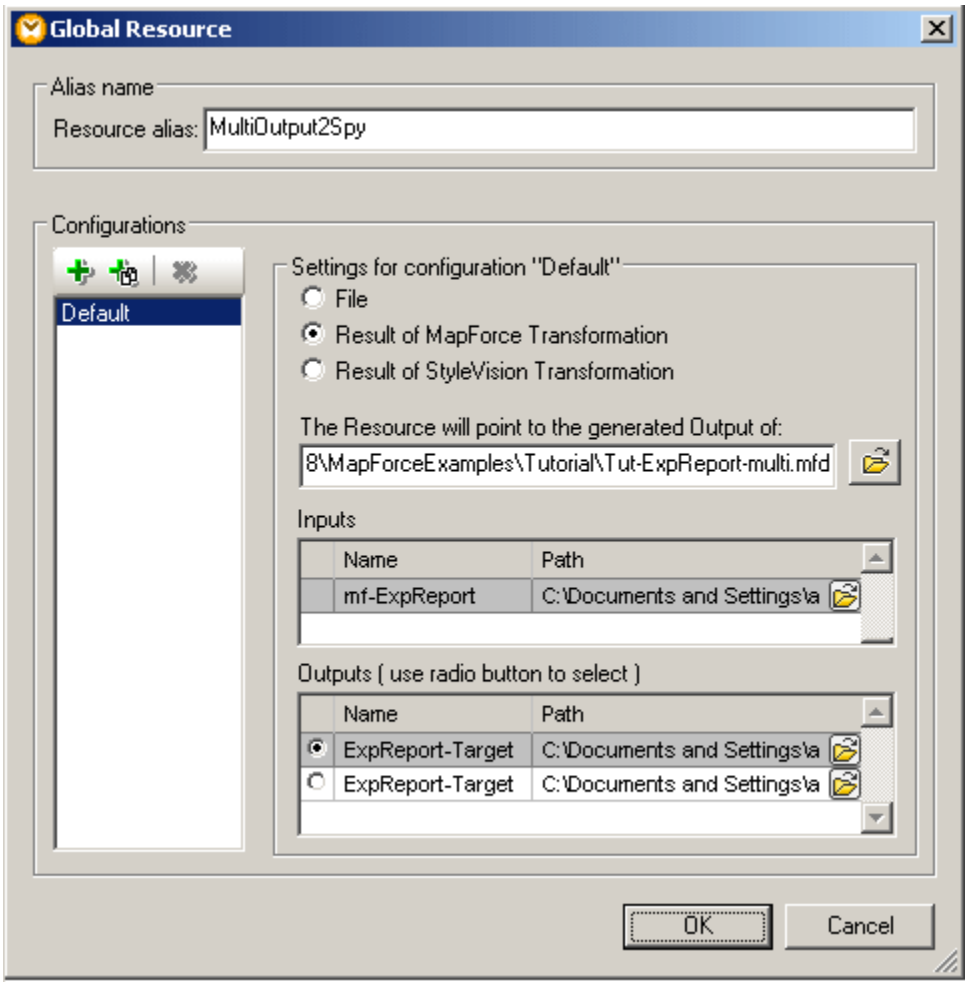
This mapping uses two output components to produce two types of filtered output; Travel and Non-travel expenses of the expense report input file. This section uses the **Tut-ExpReport-multi.mfd** mapping file available in the ...\MapForceExamples\Tutorial\ folder.



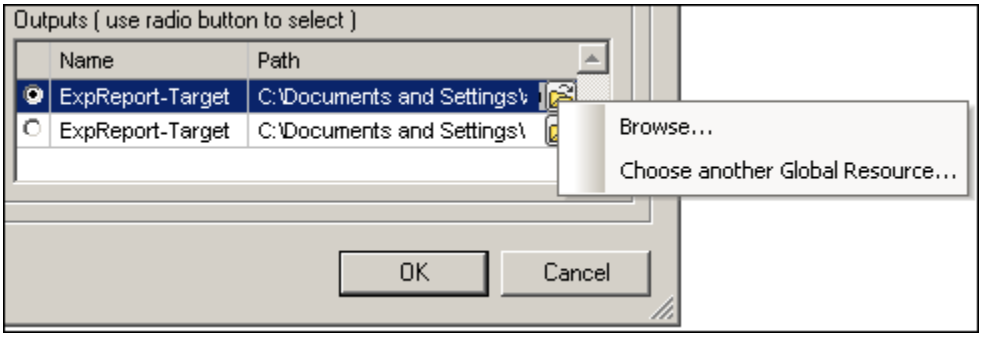
1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **File** from the popup.
3. Enter the name of the Resource alias e.g. **MultiOutput2Spy**
4. Click the "**Result of MapForce Transformation**" radio button and select the **Tut-ExpReport-multi.mfd** mapping.



MapForce analyzes the mapping and displays the input and output files in separate list boxes. Placing the mouse cursor over a path, displays the full path (and file name) in a tooltip.



5. 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.
6. Click the icon to define the **new** location of the output file, and select **Browse...** from the popup menu.

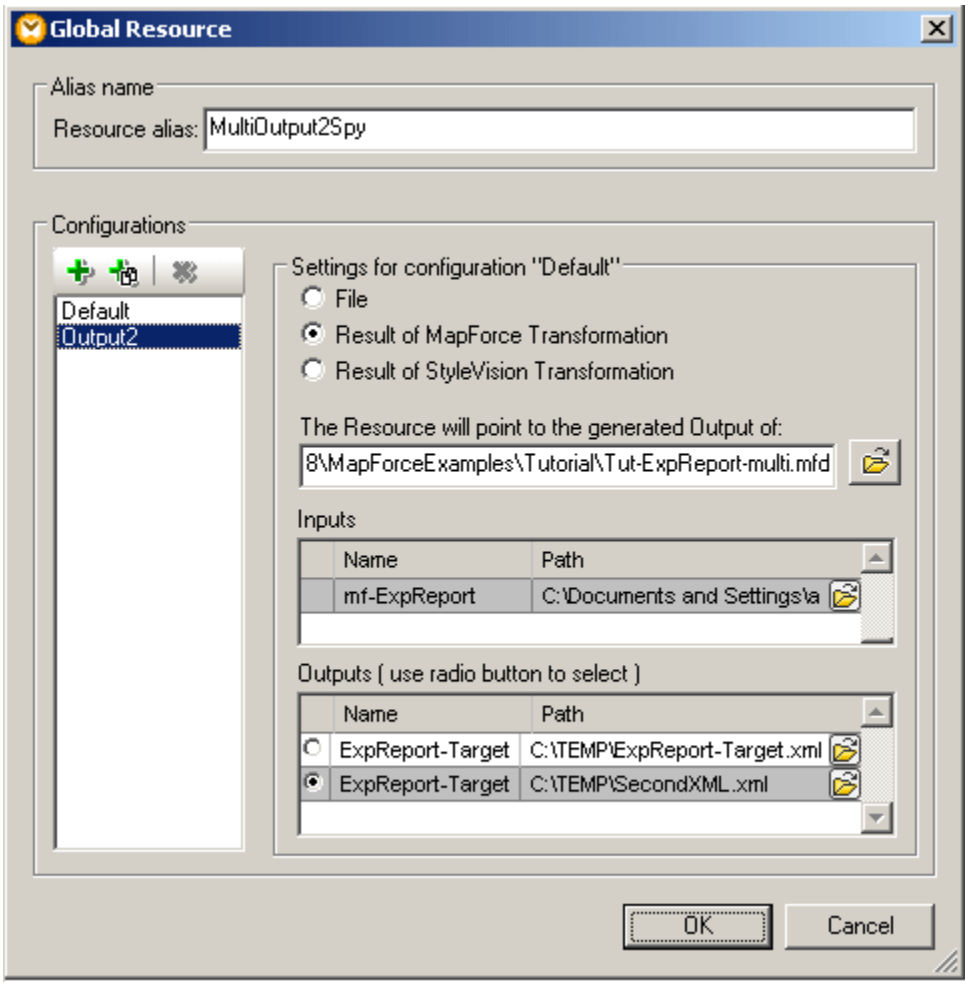


7. Enter the new output location e.g. C:\Temp and click the Save button. This location can differ from the location defined in the component settings.

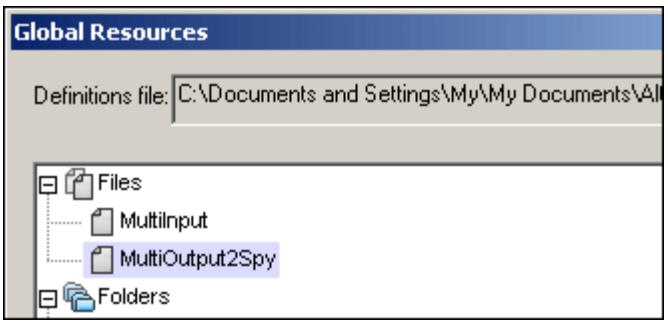
8. Click the Add button of the Configurations group (of this dialog box), to add a new configuration to the resource alias.
9. Enter the name of the configuration, e.g. **Output2**, and click the lower radio button of the Outputs listbox. Note that the output file name is **SecondXML.xml**.

10. Click the "Save as" icon 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.



11. Click OK to save the new global resources.
    The new resource alias **MultiOutput2Spy** has been added to the Global Resources definition file.

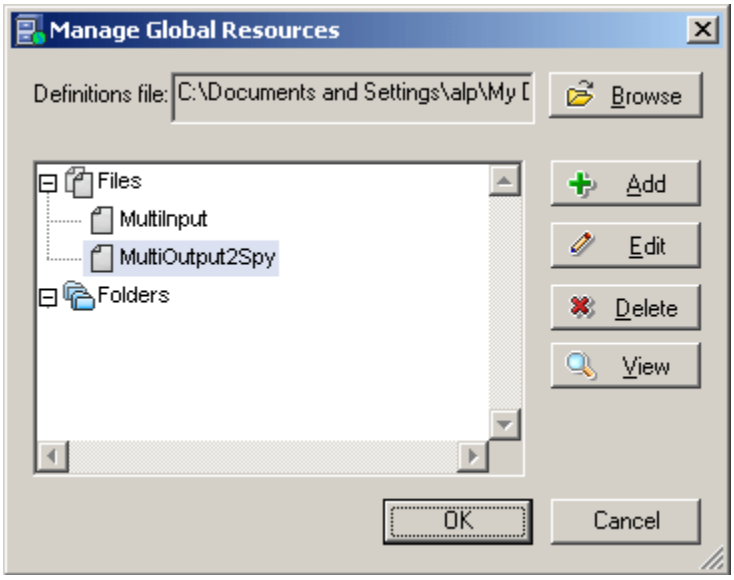


12. Click OK to complete the definition phase.

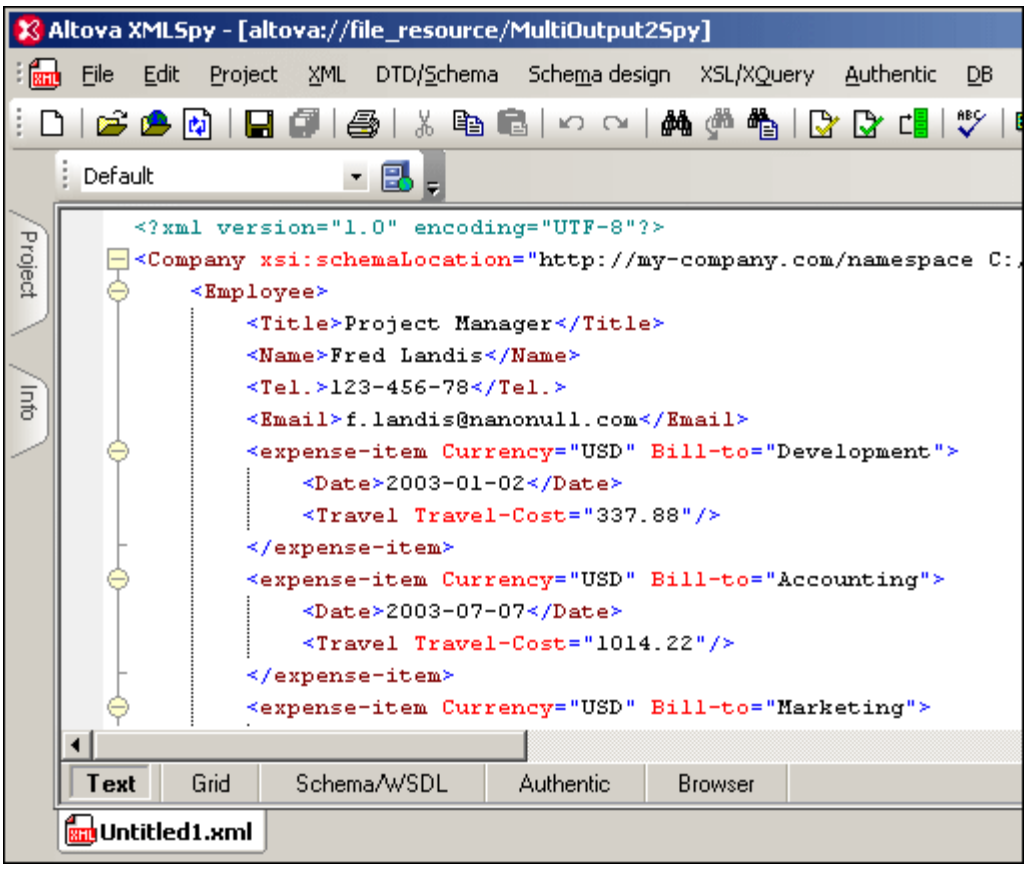
8.3.1 Start application workflow

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.

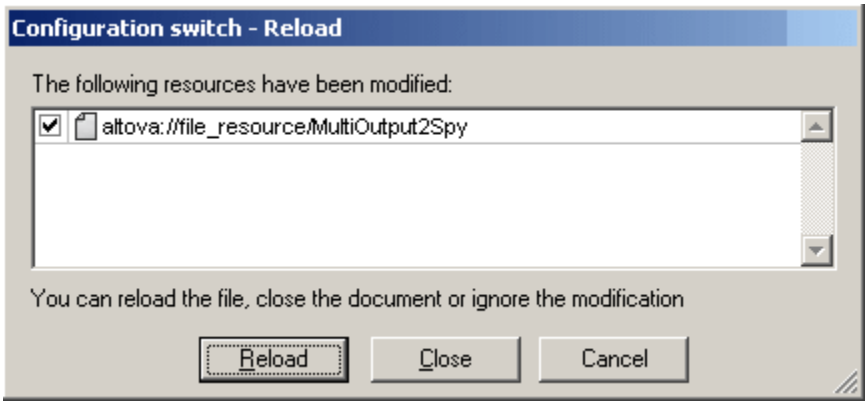


Note:

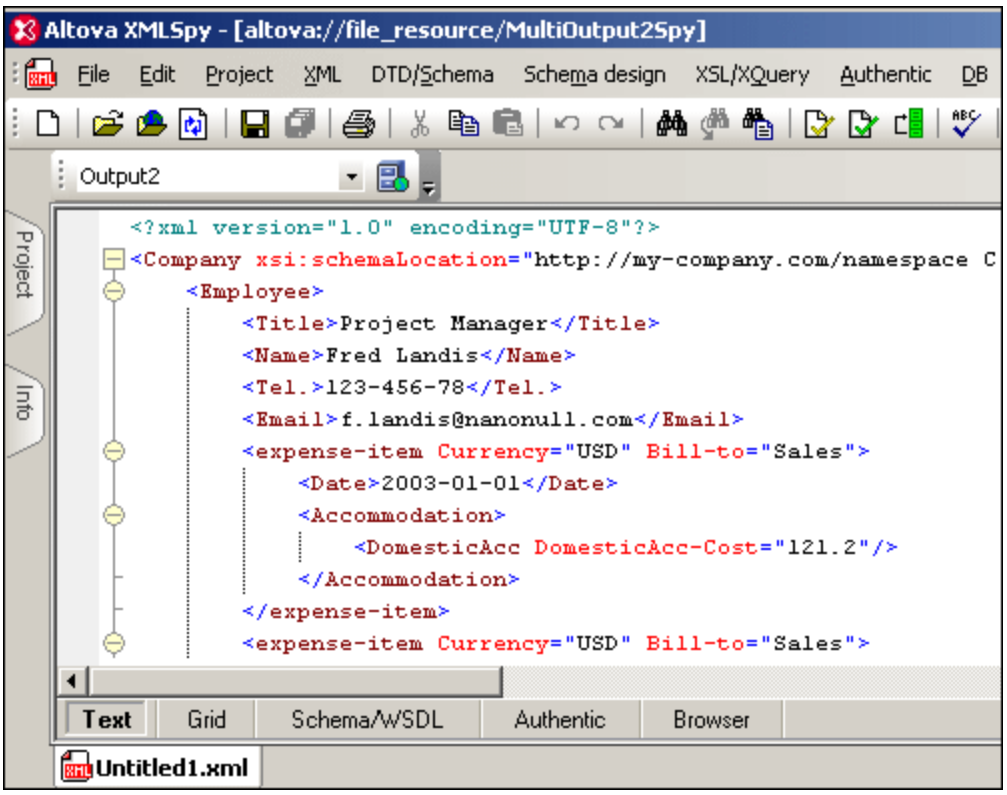
- The currently selected configuration is "**Default**".
- The name of the resource alias is in the application title bar **altova://file_resource/MultiOutput2Spy**.
- The output file has been opened as "Untitled1.xml" for further processing.
- The **ExpReport-Target.xml** file has been copied to the 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 Untitled1.xml file.

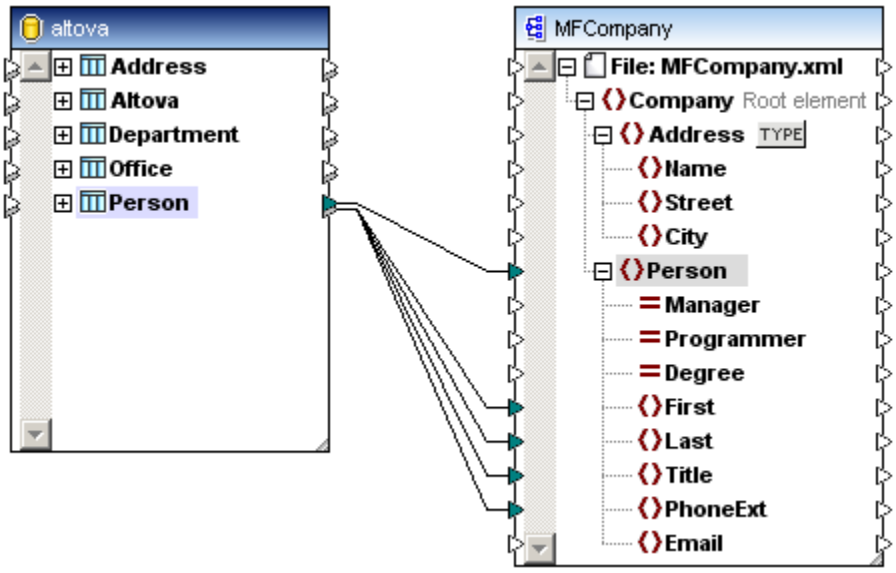
Note:

- 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.

8.4 Global Resources - Databases

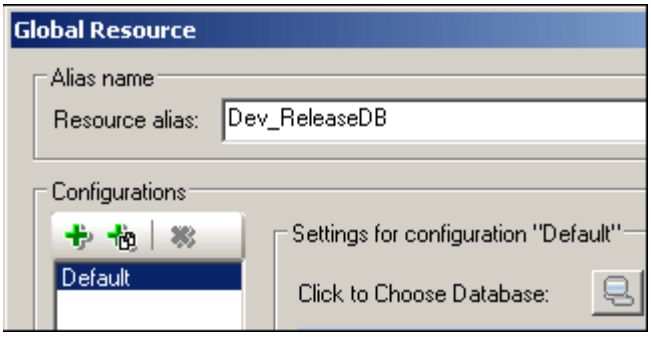
Databases components can also be defined as a global resource allowing you to refer to different databases, for development or release cycles for example. Database resources can be both source and target components.

The mapping file used in this section is available as "**PersonDB.mfd**" in the ...\MapForceExamples\Tutorial\ folder.

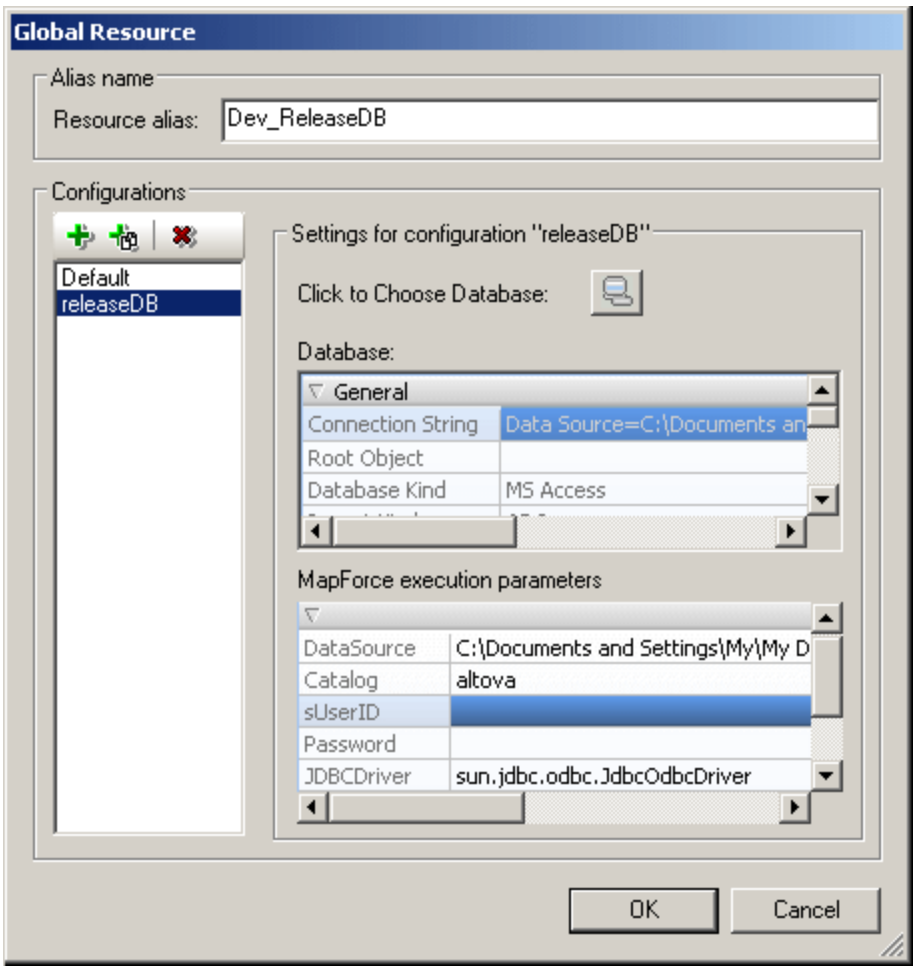


Defining / Adding a database global resource

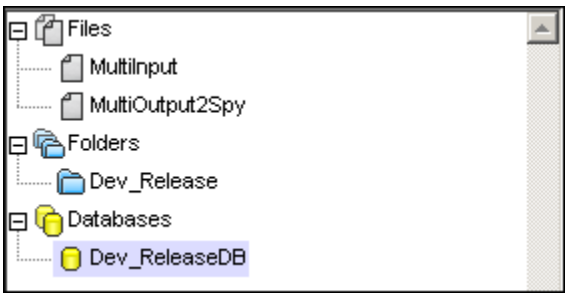
1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **Database** from the popup.
3. Enter the name of the Resource alias e.g. **Dev_ReleaseDB**.



4. Click the Connection wizard icon and select the **Altova.mdb** file 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 Connection wizard icon 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.

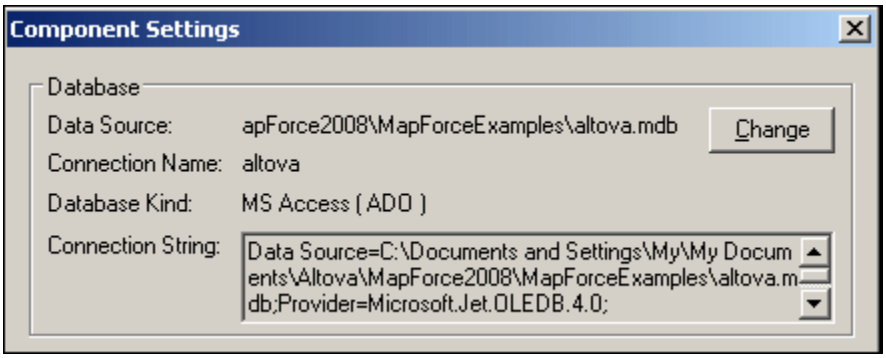
The MapForce execution parameters list box is filled with default values from the database as soon as the database has been selected. These parameters are used when generating **code** and generating the **output preview** in the Built-in execution engine.

Depending on the database you are connecting to e.g. IBM DB2, a "Choose Root

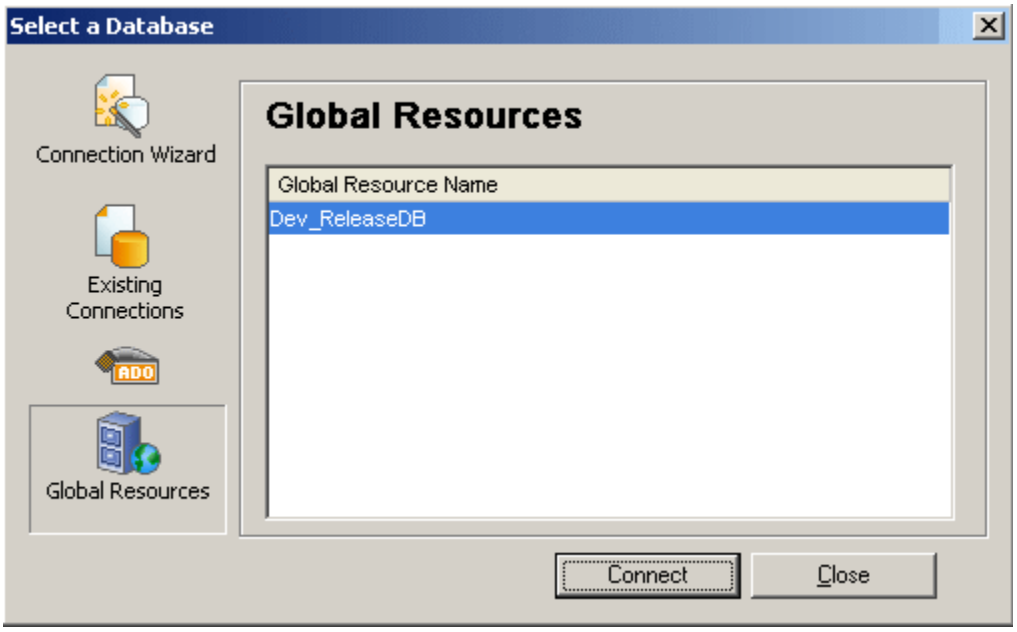
Object" dialog box may appear. This allows you to select the root object immediately through the Set Root Object button, or defer the selection until later when clicking the Skip button.

Assigning the database resource:

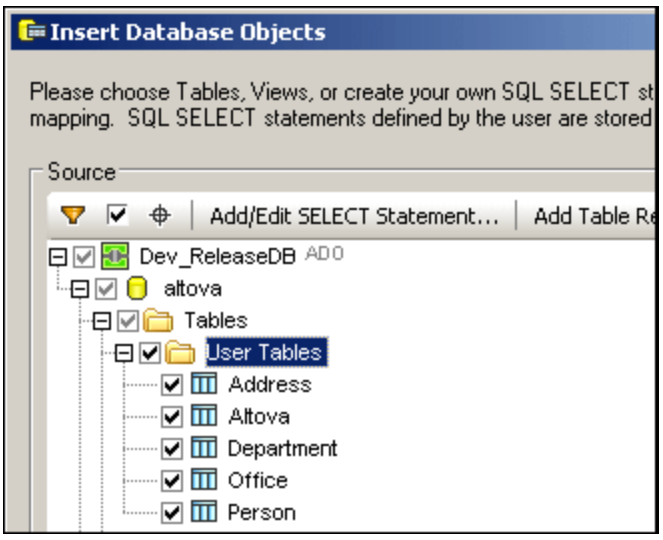
1. Double click the **Altova** database component, click the "Change" button (and then the Global Resources button if necessary).



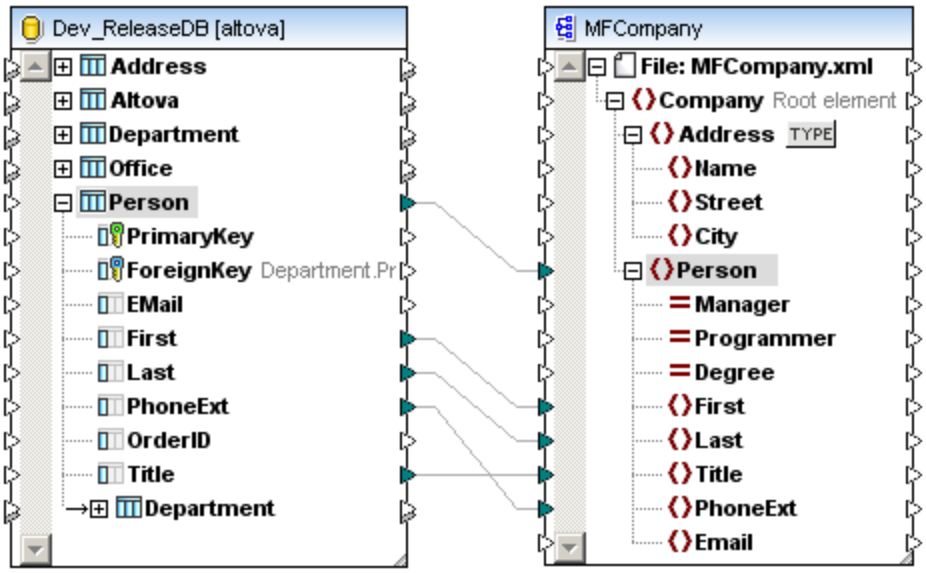
This opens the Select a Database dialog box.



2. Select the **Dev_ReleaseDB** entry and click Connect.



3. Select the tables that you want to be available in the component, click the Insert button, 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.

releaseDB

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xsi:schemaLocation="http://my-company.com/namespace
   C:/DOCUME~1/MY/MYDOCU~1/Altova/MapForce2008/MapForceExamples
   http://www.w3.org/2001/XMLSchema" xmlns="http://my-company.com/n
   http://www.w3.org/2001/XMLSchema-instance">
3    <Person>
4      <First>Vernon</First>
5      <Last>Callaby</Last>
6      <Title>Office Manager</Title>
7      <PhoneExt>582</PhoneExt>
8      <Email>v.callaby@nanonull.com</Email>
9    </Person>
10   <Person>
11     <First>Frank</First>
12     <Last>Further</Last>
13     <Title>Accounts Receivable</Title>
14     <PhoneExt>471</PhoneExt>
15     <Email>f.further@nanonull.com</Email>
```

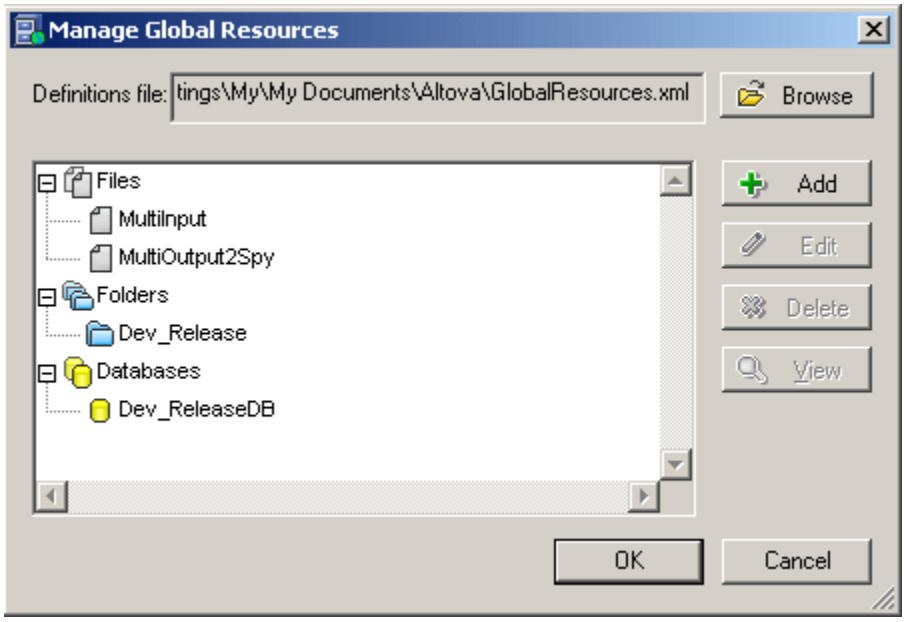
The data from the **release** database is now displayed. As both databases are identical, there is no visible difference in this example.

8.5 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, **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

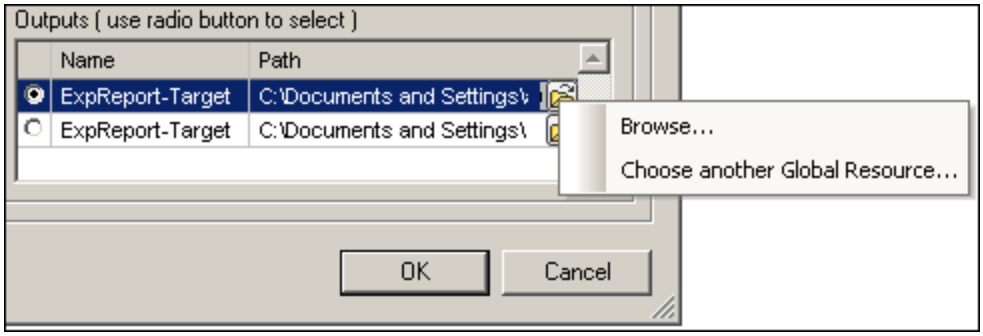
button is available in the Manage Global Resources dialog box.

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.

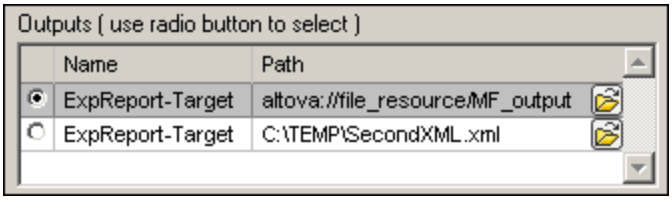
Note: Alias resource names must be unique **within** each of the Files, Folders or Databases sections. You can however define an identical alias name in two different sections, e.g. a multiInput alias can exist in the Files section as well as in the Folders section.

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).

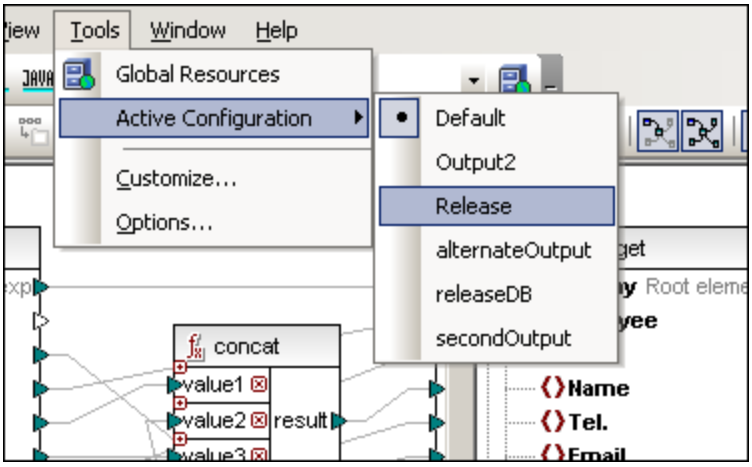
- *The active configuration* is selected via the menu item **Tools | Active Configuration** or via the Global Resources toolbar. Clicking this command (or dropdown list in the toolbar) pops up a list of configurations across all aliases.

  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.

Chapter 9

Dynamic input/output files per component

9 Dynamic input/output files per component

MapForce is able to process multiple input / output files per component, and can thus process all the files in a directory, or a subset of them, by using wildcard characters in the input component.

The example in the tutorial shows how a source component processes two XML input files, and how the target component outputs two XML documents.

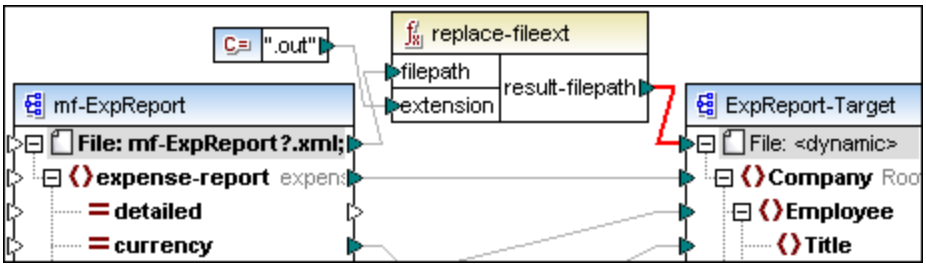
Multiple input / output files can be defined for the following components:

- XML files
- Text files (CSV, fixed length files and FlexText files)
-

Please take note of the **File:** item at the top of the abovementioned components:

- The **File:mf-ExpReport.xml** item of mf-ExpReport, displays the Input XML file entry. This is automatically filled when you assign an XML instance file to an XML schema file. (If an output file has been defined then the output file name will be also be displayed.)
- The **File: (default)** item of ExpReport-Target shows that an output instance file was not assigned to the XML schema component when it was inserted. I.e. the Output XML file field is empty. A default value will therefore be used when the mapping executes.

  Dynamic file name support is activated by mapping a string containing a file name to the File item. If the component is used as an input component, the file name may contain wildcards. See also: relative path handling.
- The **File: <dynamic>** item is shown when there is a connection to the File item, i.e. multiple files are now supported.
- The **replace-fileext** function converts the .xml extention to .out for the dynamic target files.



Dynamic/multi-file and wildcard support for MapForce supported 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	*	*	*

* 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.

Wildcards * and ? are resolved when entered in the Component Settings dialog box and also when mapping a string to the File: name node.

To transform XSLT 1.0/2.0 and XQuery code using AltovaXML, please see and Generating XQuery 1.0 code.

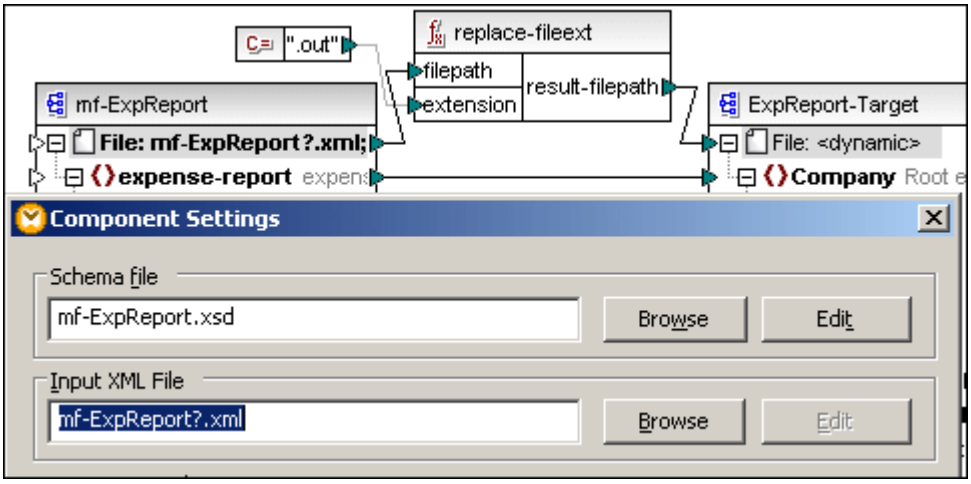
9.1 Dynamic file names - input / output

By mapping file names dynamically inside the mapping, you can:

- generate a mapping application where the input and output file names can be defined at runtime
- convert a set of files to another format (many-to-many)
- split a large file (or database) into smaller sections/parts
- merge multiple files into one large file (or load them into a database)

To process multiple input files, you can 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 map all files starting with mf-ExpReport with one following character, of which there are two in the ...\Tutorial folder.

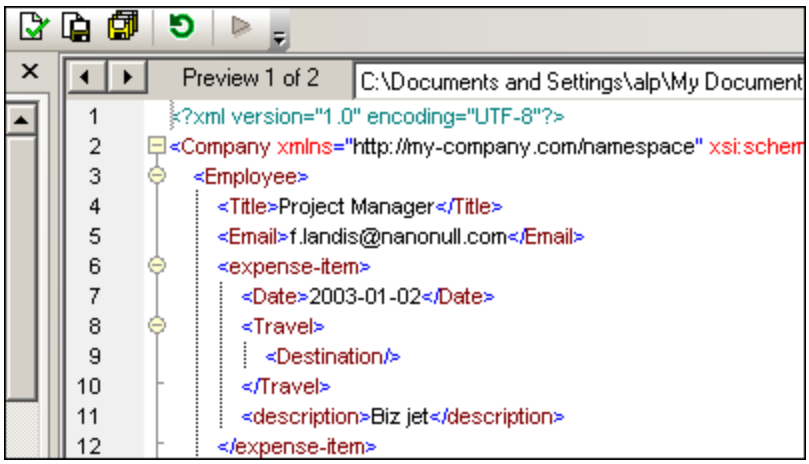


- Map a **sequence** of strings to the *File* node of the source component. Each string in the sequence represents one file name. The strings may also contain wildcards, which are automatically resolved.

  A sequence of file names can be supplied by:

  - An XML file
  - A text file (CSV or fixed length) etc.
  - Database text fields
  -

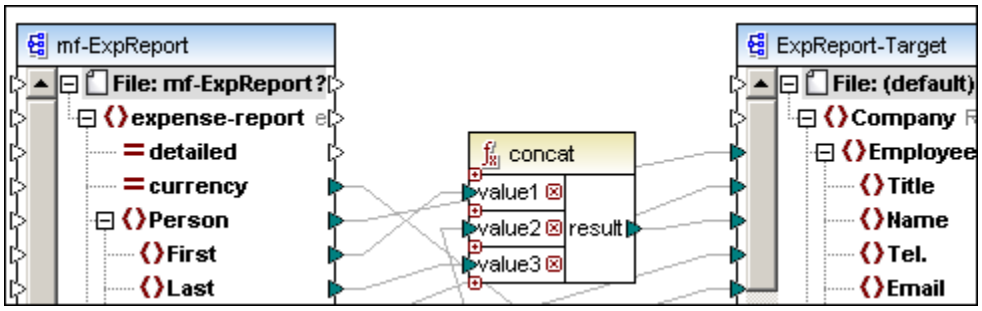
Preview of dynamic input / output mappings
Clicking the Output tab displays the mapping result in a preview window. If the mapping produces multiple output files, as shown below, Preview 1 of 2, each file has its own numbered pane in the Output tab. Click the arrow buttons to see the individual output files.



Click the Save all generated outputs icon , to save the generated output files you see here.

Multi input / single output - merging input files

Multiple input files can be merged into a **single** output file if the connector **between** the two **File:** items is removed, while the source component still accesses multiple files e.g. per wildcard "?". While the source component can take multiple files, the output component cannot. The multiple source files are thus appended in the target document.



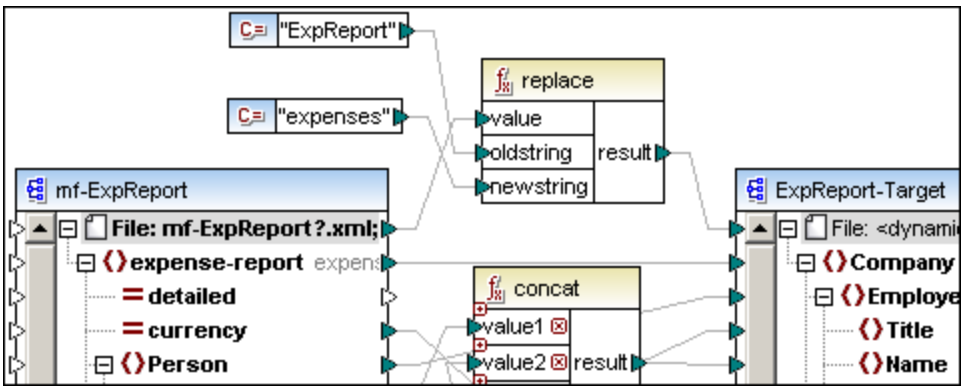
Multi input / multi output

To map multiple files n:n 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.

The full path name of the currently processed file is available by connecting the output icon of the *File* node, e.g. to concat for adding a new file extension.

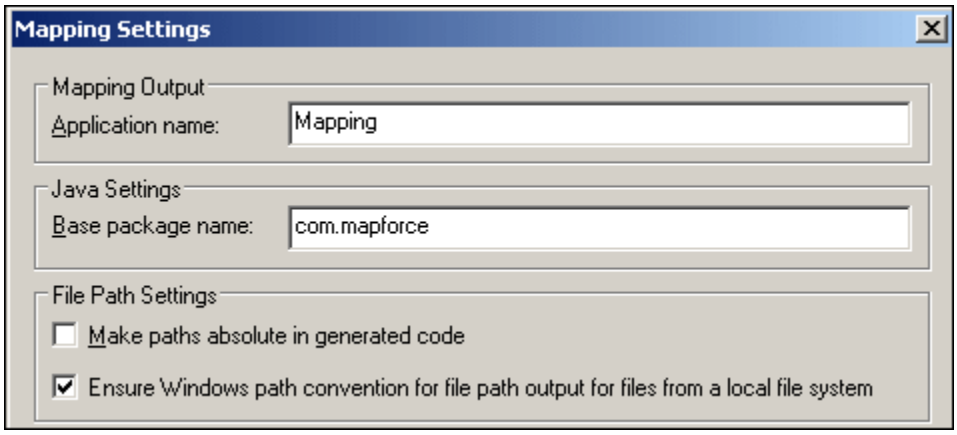
Please note:

Avoid simply connecting the File: nodes **directly** without using any processing functions, as this will overwrite your input files when you run the mapping. You can change the output file names using various functions e.g. the replace function as shown below.



The output file names in the above case will be **mf-expenses1.xml** and **mf-expenses2.xml**.

The menu option **File | Mapping Settings** allows you to globally define the file path settings used for the mapping project.



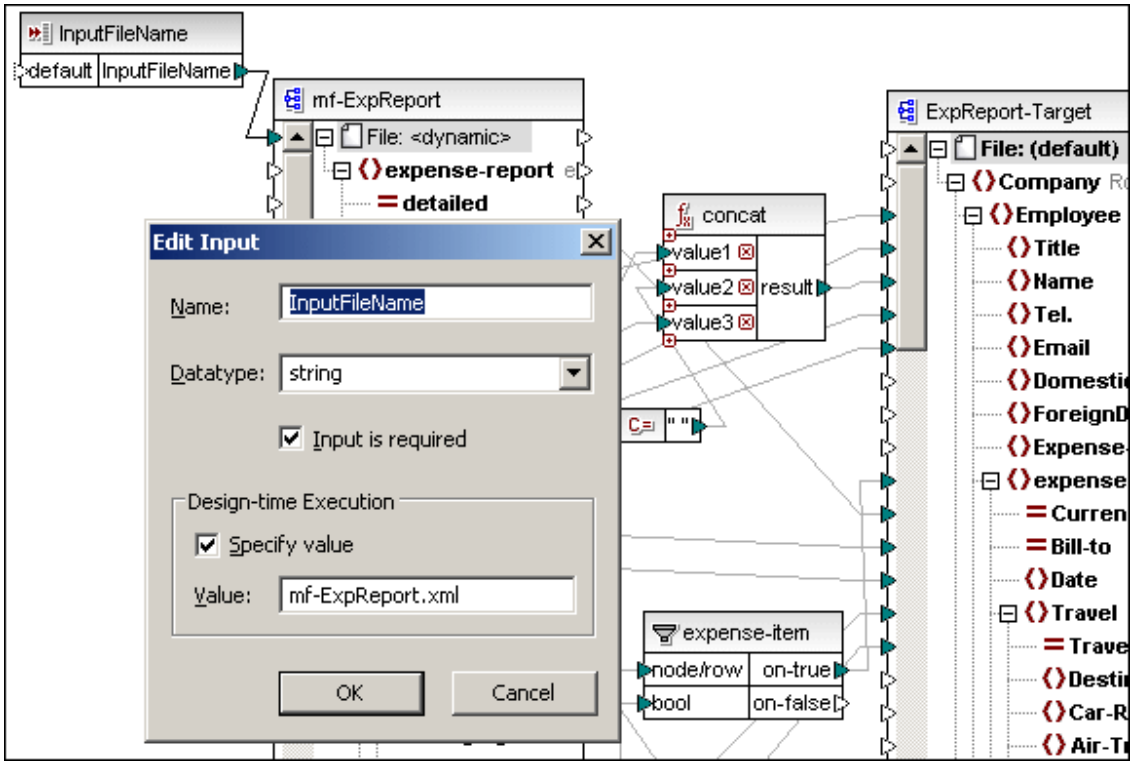
The "Ensure Windows path convention...." check box makes sure that Windows path conventions are followed. When outputting XSLT2 (and XQuery), the currently processed file name is internally retrieved using the **document-uri** function, which returns a path in the form **file:// URI** for local files.

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.

9.2 Dynamic file names as Input parameters

MapForce allows you to create special input components that can act as a **parameter** in the command line execution of the compiled mapping. This specific type of input component cannot be used **inside** a user-defined function, it is only available in the main mapping window.

To process a file using an Input parameter at runtime:
To define the path and file name at runtime, connect an input parameter component to the input icon of the *File* node in the source component. Depending on the connections to other items, this will define the input and/or the output file name.



The mf-ExpReport.xml entry in the Value field is only used for preview purposes in the Output window. It has no effect on the parameter values used when running the code from the command line. Please see Input parameters, overrides and command line parameters for more information.

When you have generated and compiled your code you can supply the file name for the mapping using the command line:
mapping.exe /InputFileName Filename.xml.

Where:

- **/InputFileName** is the name of the first parameter
- **Filename.xml** is the second parameter i.e. the dynamic file name you want to be used when running the application from the command line.

9.3 Multiple XML files from single XML source file

The content of the XML source file **mf-ExpReport.xml**, available in the ...\MapForceTutorial folder, is shown below. It consists of the expense report for Fred Landis and contains five expense items of different types. This example is available as **Tut-ExpReport-dyn.mfd** in the ...\Tutorial folder.

Aim:
To generate a separate XML file for each of the expense items listed below.

Person

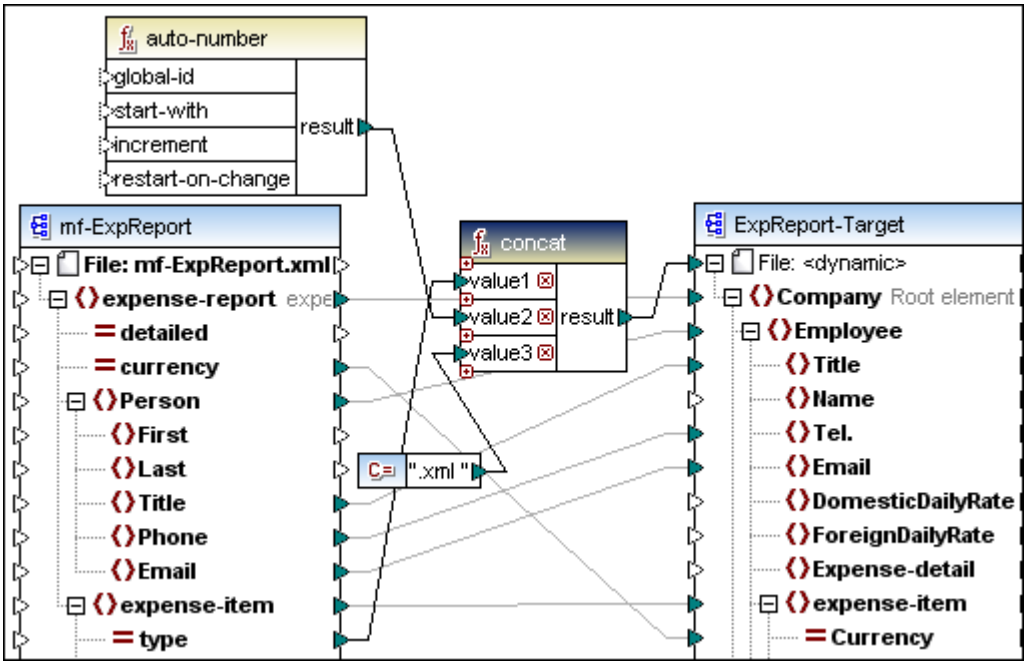
First	Fred
Last	Landis
Title	Project Manager
Phone	123-456-78
Email	f.landis@nanonull.com

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		

As the "type" element defines the specific expense item type, this is the item we will use to split up the source file.

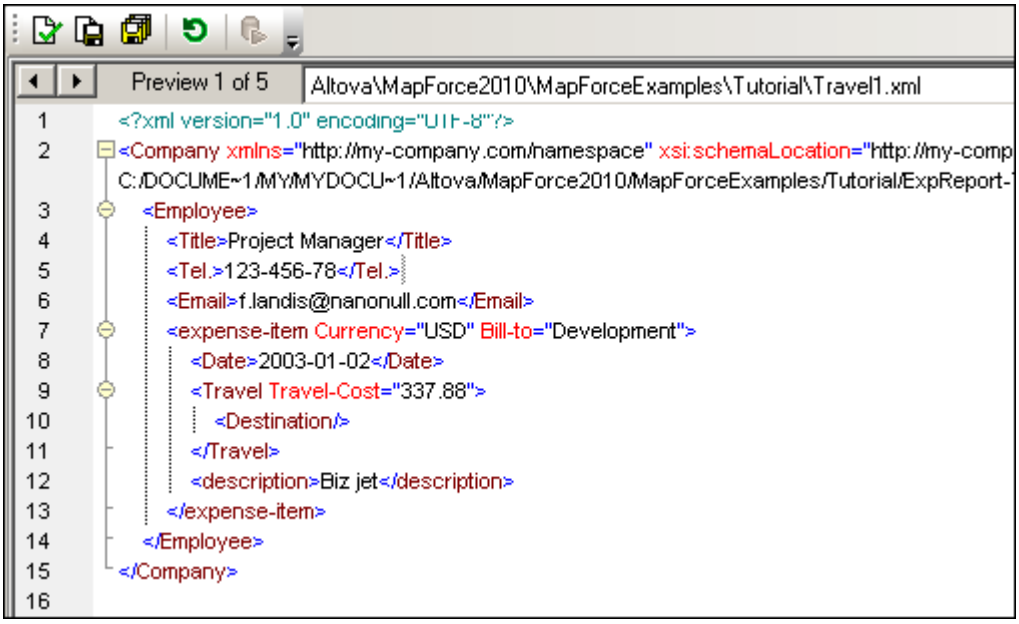
1. Insert a concat and constant function from the libraries pane.
2. Insert the auto-number function from the **core | generator functions** library of the libraries pane.



3. Create the connections as shown above: type to value1, auto-number to value2 and the

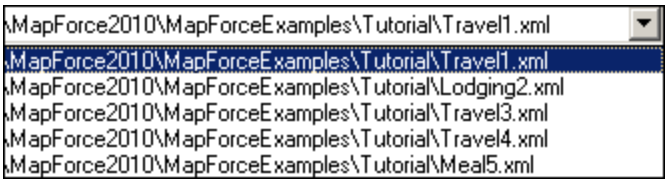
constant to value3.

4. Connect the **result** parameter of the concat function to the **File:** item of the target component. Note that File: <dynamic> is now displayed.
5. Define the remaining connections as needed.
6. Click the Output tab to see the result of the mapping.



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

7. Click the drop-down list arrow arrow to see all the files that have been generated.



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

Note:

- The **type** element supplies the first part of the file name e.g. Travel.
- The **auto-number** function supplies the file number increments (default settings are start at=1 and increase=1) thus Travel1.
- The **constant** component supplies the file extension i.e. .xml, thus Travel1.xml is the file name of the first file.
- Clicking the Save All icon  allows you to save the individual files directly from the Output tab, without having to generate code.

9.4 Multiple XML files per table

The content of the **altova.mdb** database file, available in the ...\MapForceTutorial folder, is shown below. It consists of four tables where the Person table contains 21 separate person records. This example is available as **PersonDB-dyn.mfd** in the ...\Tutorial folder.

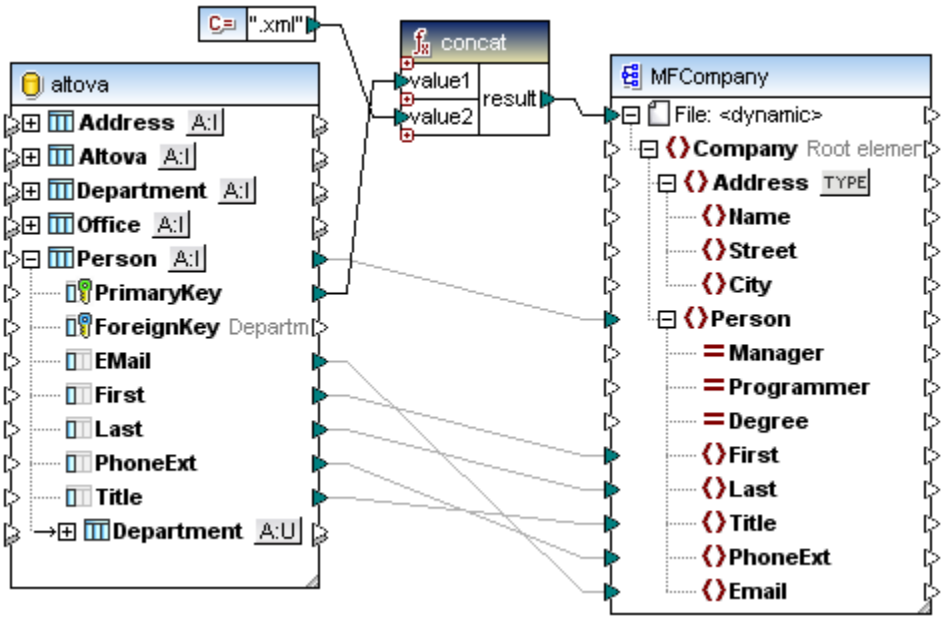
Aim:
To generate a separate XML file for each of the Person records shown below.

Person

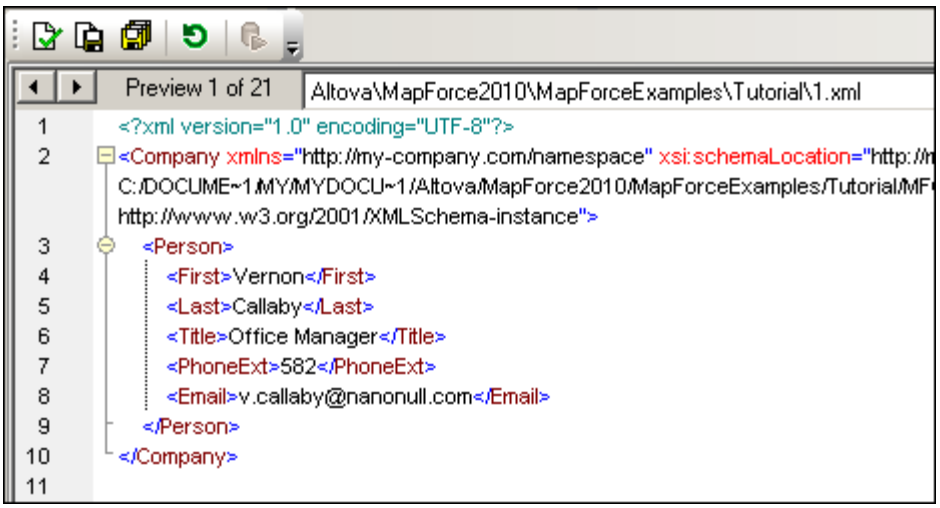
PrimaryKey	ForeignKey	EMail	First	Last
1	1	v.callaby@nanonu	Vernon	Callaby
2	1	f.further@nanonu	Frank	Further
3	1	l.matise@nanonu	Loby	Matise
4	2	j.firstbread@nanc	Joe	Firstbread
5	2	s.sanna@nanonul	Susi	Sanna
6	3	f.landis@nanonul	Fred	Landis
7	3	m.landis@nanonu	Michelle	Butler
8	3	t.little@nanonull.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

As the "PrimaryKey" field determines the specific persons in the table, this is the item we will use to split up the source database into separate files.

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



2. Create the connections as shown above: PrimaryKey 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 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 **PrimaryKey** field supplies the first part of the file name e.g. 1.
- The **constant** component supplies the file extension i.e. **.xml**, thus 1.xml is the file name of the first file.
- Clicking the Save All icon allows you to save the individual files directly from the Output tab, without having to generate code.

9.5 Relative and absolute file paths

A relative path is a path that does not start with a drive letter, i.e. it can be a file name without path. The specific context in which the relative file name is used, defines the base path. The handling of relative file names has changed in MapForce version 2010 due to the support for mapping file names as data inside a mapping.

Previous versions of MapForce (prior to 2010) saved file paths relative to the *.MFD file for files in the same, or a descendent folder, and changed them to absolute paths when they were opened/loaded.

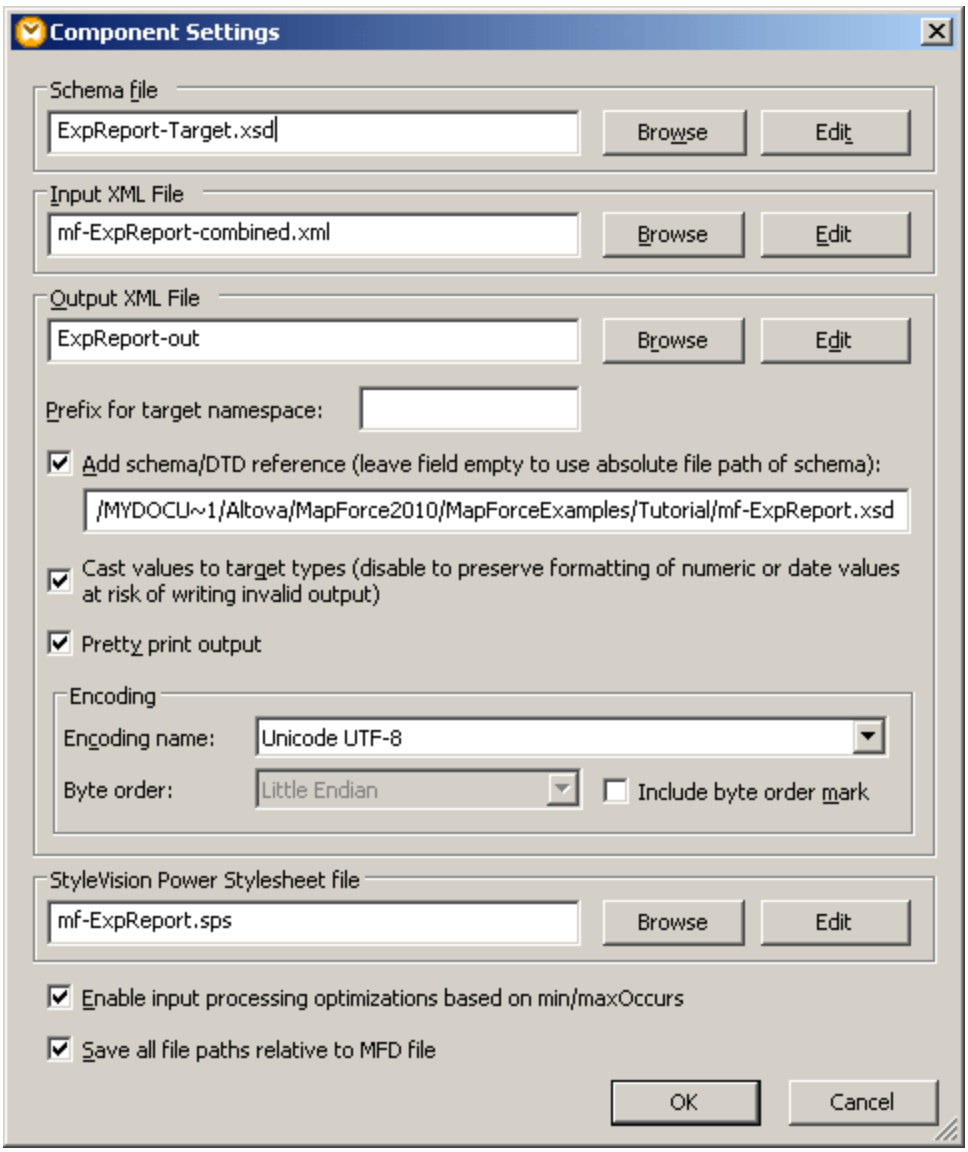
Since MapForce 2010, all references to external files, such as schemas or XML instance files, are stored the way they are entered in the dialog box – in this way, relative paths can be used where required.

Save all paths relative to MFD file
This new option, common to all component settings dialog boxes, saves all file paths (of the component) relative to the location of the current MFD file. This allows you to move a mapping together with all related files to a different location, while keeping all file references intact.

This means that:

- Absolute file paths will be changed to relative paths
- The parent directory "**..**" will be also be inserted/written
- Using Save as... will adjust the file paths (relative to the MFD file) to the new location you are saving the MFD file to

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



There are two separate types of files that are referenced from an MFD file:

- Schema-type files (XML Schemas, WSDL, FlexText configuration files, …) entered in the **schema file** field.
- Instance files entered in the **Input xxx File**, or **Output xxx File** fields.

Schema-type files

Schema-type files are used when designing a mapping. They define the **structure** of the mapped input and output instance files. This information is used to display the item trees/hierarchy in the various components. MapForce supports entering and storing a relative path to schema-type files.

- Relative paths to **schema-type** files will be always resolved **relative to the MFD file**.
- Selecting a schema via the "Open" dialog, e.g. after inserting a new component, or clicking the “Browse” button, always inserts the **absolute** path into the field.

- To set a relative path, which will also be stored in the MFD file, delete the path from the text box, or type a relative path or file name. This will happen automatically on saving the MFD file if the "Save all paths relative to MFD file" checkbox is activated. You may also use "**..**" to specify the parent folder of the MFD file.
- Saving a MFD file that references files from the same directory, then moving the complete directory to another location, **does not** update any **absolute** paths stored in the mfd file. Users who use source control systems and different working directories should therefore use relative paths, in this case.

Instance files and the execution environment
The processing of instance files is done in the generated XSLT, XQuery or in the generated application, as well as MapForce preview.

In most cases it does not make sense to interpret relative paths to instance files as being relative to the MFD file, because that path may not exist at execution time - the generated code may be deployed to a different machine. **Relative** file names for instance files are therefore resolved relative to the **execution environment**:

Target language	Base path for relative instance file name
XSLT/XSLT2	Path of the XSLT file
XQuery	Path of the XQuery file
C++, C#, Java	Working directory when running the generated code
MapForce Preview/BUILTIN (or execution from API)	Path of the MFD file
BUILTIN execution from command line	Specified on command line (Target directory)

A new check box that ensures compatibility of generated code with mapping files (*.mfd) from versions prior to Version 2010, has been added in the **File | Mapping Settings** dialog box, i.e. "**Make paths absolute in generated code**".

The state of the check box is automatically set depending on what is opened, the check box is:

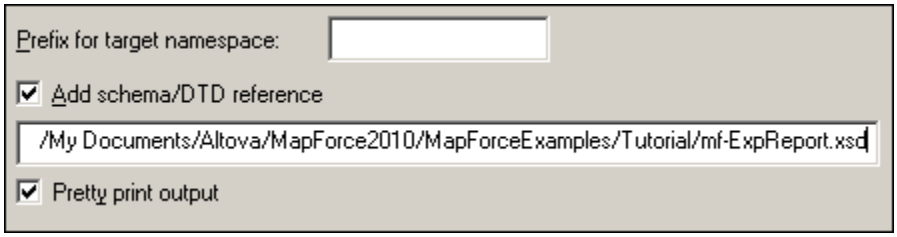
- **inactive** if a **new** mapping file, i.e. version 2010, is created or opened
  Relative paths for input and output instance files are written as is, into the generated code.

  This allows deployment of the generated code to a different directory or even machine. You must ensure that files addressed using relative paths are available in the runtime environment at the correct location.
- **active** if an older mapping file from version 2009, 2008 etc. is opened
  Relative paths for input and output instance files are made absolute (relative to the *.MFD file) before generating code. This has the same effect as generating code with an older version of MapForce.

Note that the **source** instance file name is also used for the following purposes:
- Detection of the XML root element and the referenced schema
- Validation against the selected schema
- Reading Excel worksheet names and columns
- To read column names and preview contents of text files (CSV or FLF)

New "schemaLocation" field for **target XML files**
Since schema references may be stored relative to the MFD file, and the **generated** XML file from a target component is often in a different directory, there is a way to enter a separate **schemaLocation** path for the target XML **instance**, so that the XML file can be validated there.

This is the field below the “Add schema/DTD reference” check box, of the Component Settings dialog box (Double click a component to open it). A similar field exists for the taxonomy reference in XBRL components.



The path of the referenced/associated schema, entered in this field, is written into the generated **target instance** files in the **xsi:schemaLocation** attribute, or into the DOCTYPE declaration if a DTD is used.
Note: A URL e.g. **http://mylocation.com/mf-expreport.xsd** can also be entered here.

Chapter 10

Intermediate variables

10 Intermediate 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.

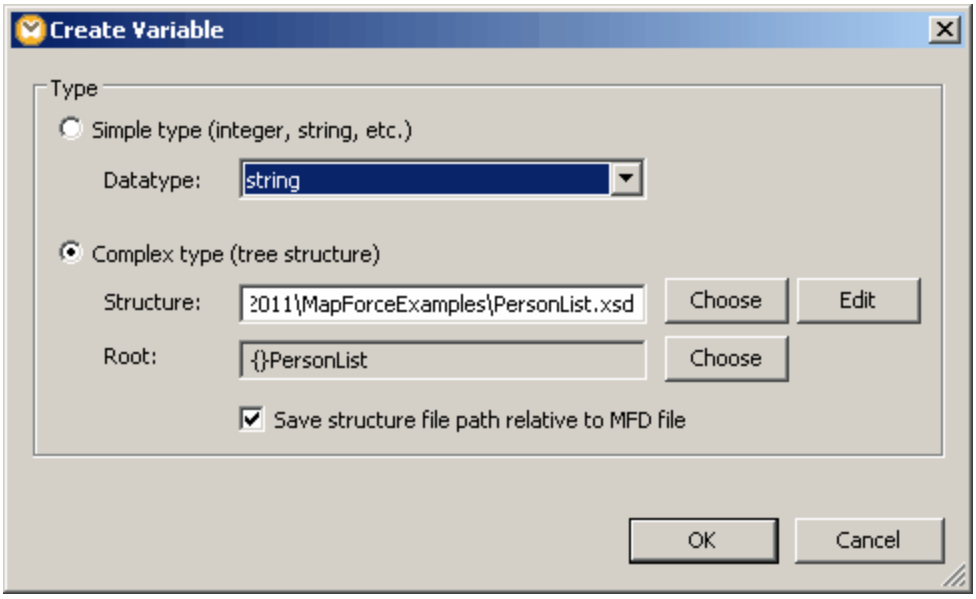
Difference between variables and chained mappings

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.

To insert 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.

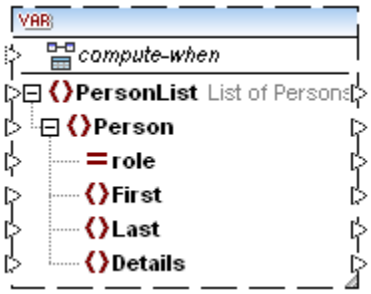


2. Click the radio button for the type of variable you want to insert, i.e. Simple type, or Complex type.

   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:

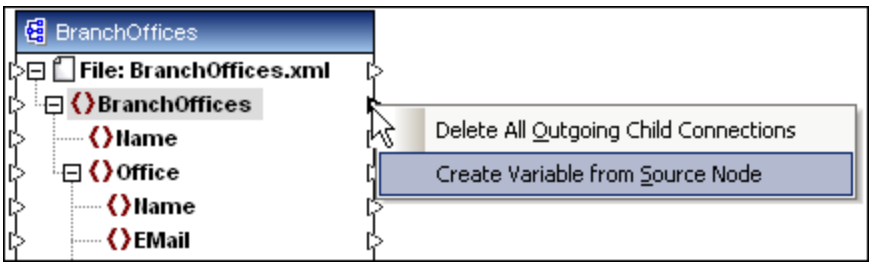


Simple variable:

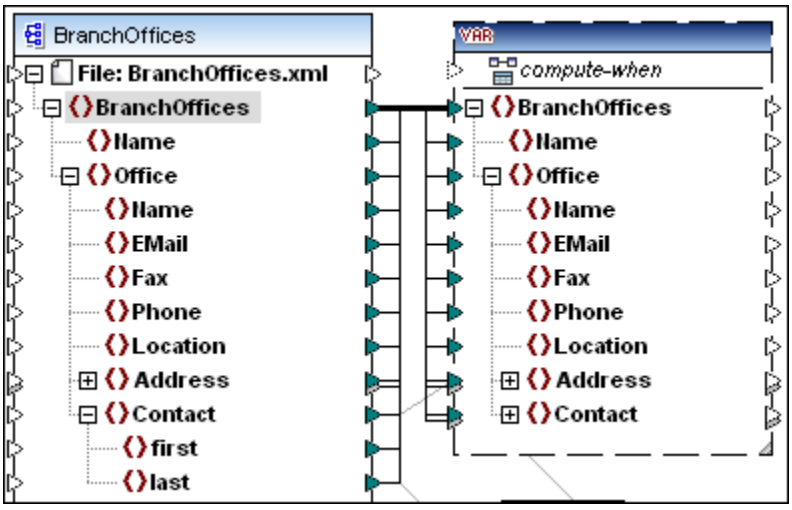
Have a single mappable item/value e.g. string, integer. Note that the "value" item can be duplicated.

Alternate methods of inserting variables:

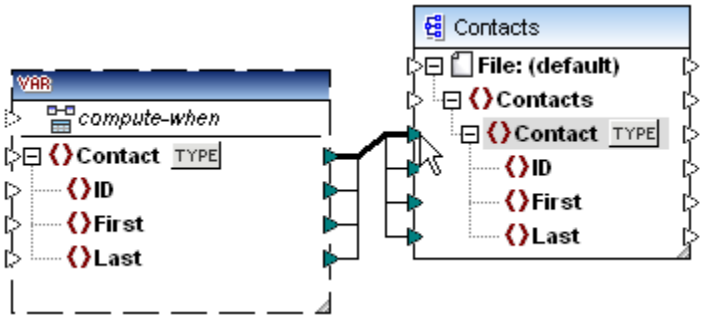
- Right click an **output** icon of a component (e.g. BranchOffices) and select "Create Variable from Source node".



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

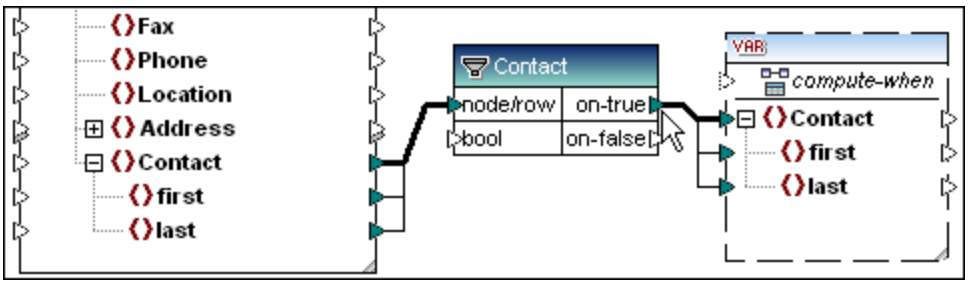


- Right click an **input** icon of a target component (e.g. Contact) and select "Create Variable for Target Node".



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 function (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 of course also contain only 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 (see "Loops, groups and hierarchies - The context").

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.

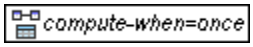
Compute-when - connected
By connecting compute-when to an **output node** of a source component, the variable is

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.

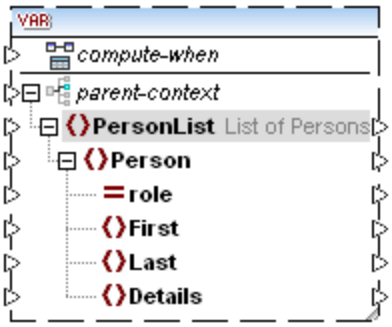
This setting causes the variable value to be computed once **before** any of the target components, making the variable essentially a global constant for the rest of the mapping.

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:

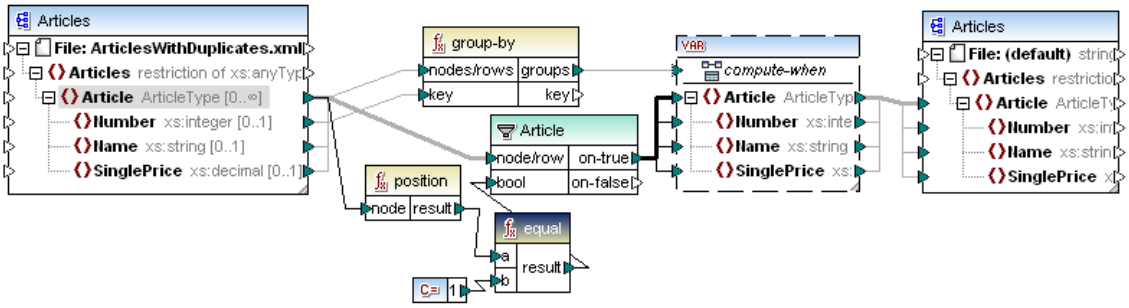
1. Right click the root node, e.g. PersonList and select "Add Parent Context" from the context menu.
   This adds a new node, "parent-context", to the existing hierarchy.



   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.

10.1 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

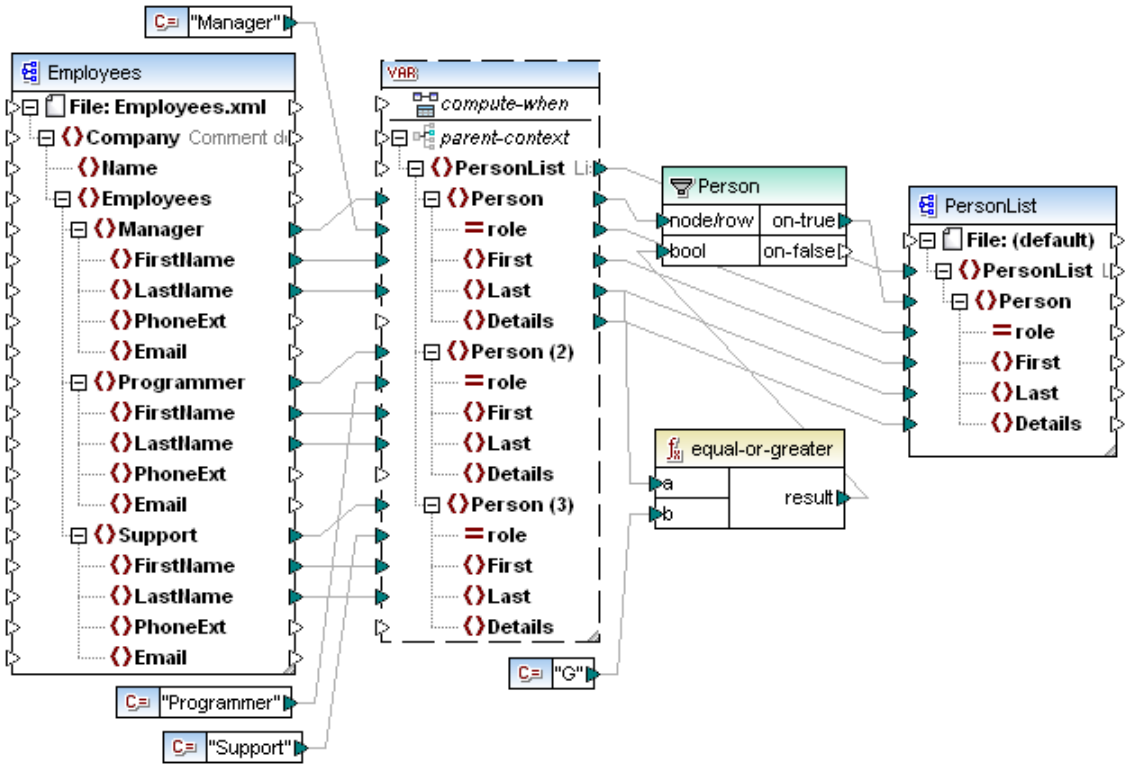
The article Number is used as the key in the group-by function, so it creates one group per unique article number. Each group thus contains one article and all the unwanted duplicates of that article. The next step is to extract the first item of each group and discard the rest.

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.

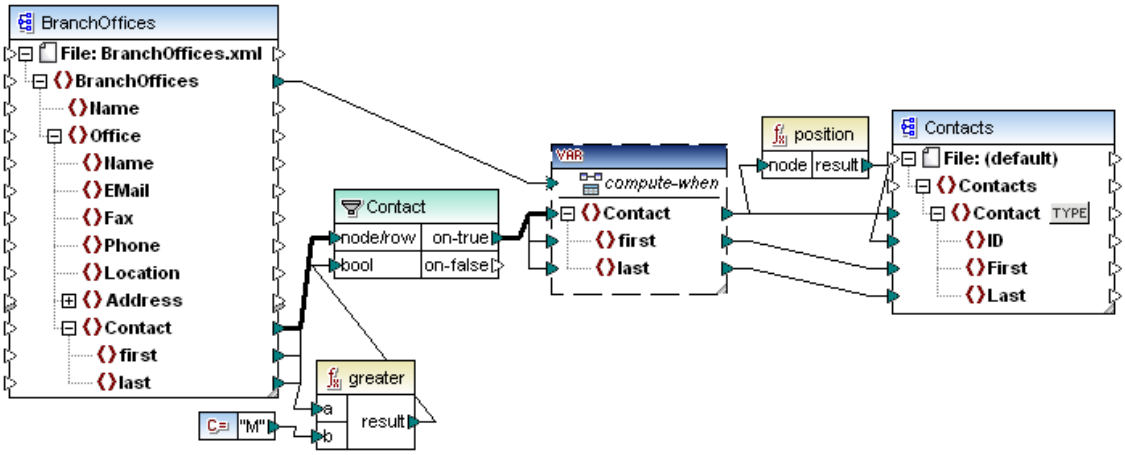


The Person node of the **variable** has been duplicated twice, and a filter has been added to filter out those persons whose Last name starts with a letter higher than "G".

Numbering nodes of a filtered sequence

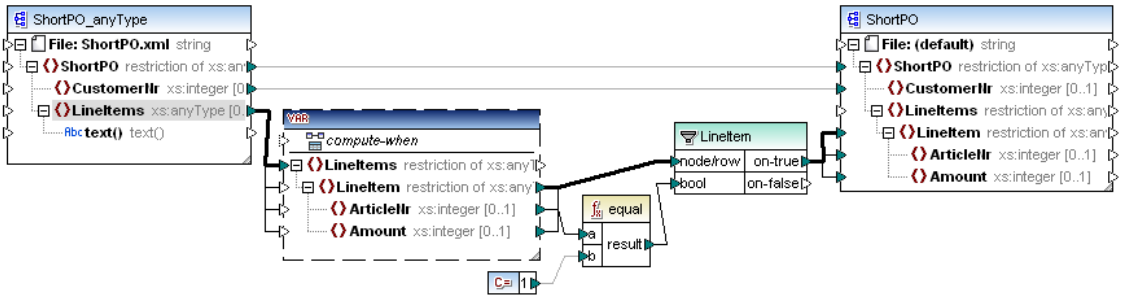
This example is available as **PositionInFilteredSequence.mfd** in ...\MapForceExamples folder and uses the variable to collect the filtered contacts where the last name starts with a letter higher than M.

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.

Chapter 11

How To... Filter, Transform, Aggregate

11 How To... Filter, Transform, Aggregate

This section deals with common tasks that will be encountered when creating your own mappings.

General:

I want to:	Read this section
Filter data based on specific criteria	Filter - retrieving dynamic data
Use **dynamic/multiple input** and output files when mapping	Dynamic and multiple input/output files per component
Map/use a **derived complex type** (xsi:type)	Derived XML Schema types - mapping to
Create a **recursive** user-defined mapping	Recursive user-defined mapping
Preview mappings (before generating code)	Previewing mappings - Built-in execution engine
Use min, max, sum, avg, and count aggregate functions	Aggregate functions
Use an **input component** as a **parameter** during command line execution	Input values /overrides
Specify an **alternative** value for an input component	Input values / overrides
Transform an input value to an output value using a **lookup table**	Value-Map - Transforming input data
Know about type conversion when mapping	Type conversion checking
Create my own **catalog** files	Catalog files in MapForce
Merge multiple source files into a single target file	Merging multiple files into one target

Nodes

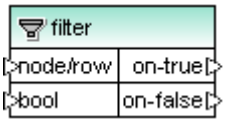
I want to:	Read this section
Test nodes; existing / not existing nodes	Node testing, exists / not exist
Group nodes by their content	Grouping nodes / node content
Map data based on the **position** of a node in a sequence	Position of context items in a sequence
Comment a mapping / specific connectors or nodes	Annotations / Commenting

I want to:	Read this section:
Create a **valid** XML target document with one root element	Mappings and root elements of target documents
Know what happens to **child items** when I use a **filters** on a parent item	Filter components - Tips
Replace **special (hex)** characters in a database with others in the target file.	Replacing special characters in database data
Define the node which is to act as the **context node** in a source file.	Priority context node/item
Map multiple **hierarchical tables** to a single XML target file	Mapping multiple tables to one XML file
Run MapForce from the **command line (Ent/Pro)**	command line parameters
Define **exceptions** in mappings	MapForce exceptions

11.1 Filter - retrieving dynamic data, lookup table

If you want to retrieve/filter data, based on specific criteria, please use the **Filter** component.

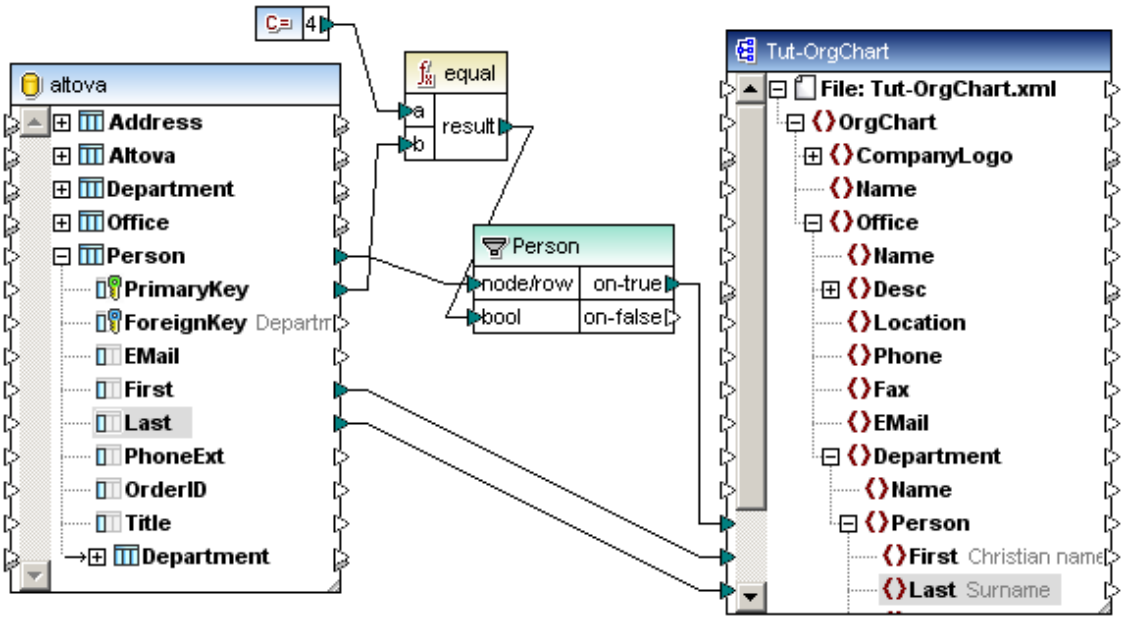
1. Click the filter icon  in the icon bar to insert the component.
   This inserts the filter component in its "unconnected" form where "filter" is visible in the title bar of the component.



   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.

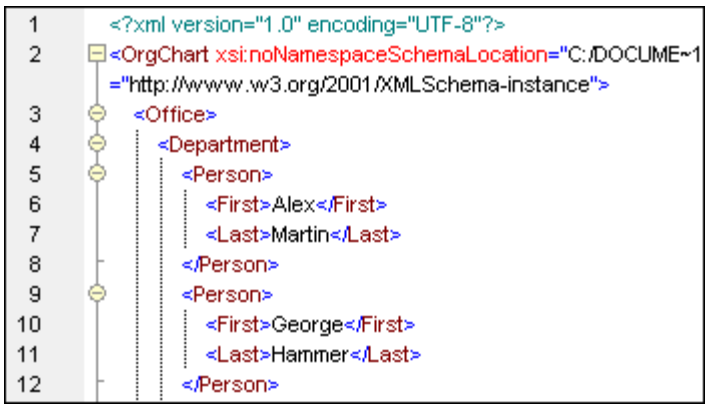
Goal: to map all First/Last names from the **Person** table, where the Department **primary** key is equal to 4.

- 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 the 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 please see:

- Filtering XML data: Filtering data:
- 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
- Database filters and queries: Database filters and queries
- Filter component tips: Filter components - Tips

11.1.1 Filter components - Tips

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, its 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 the other filter output are executed.

11.2 Value-Map - transforming input data

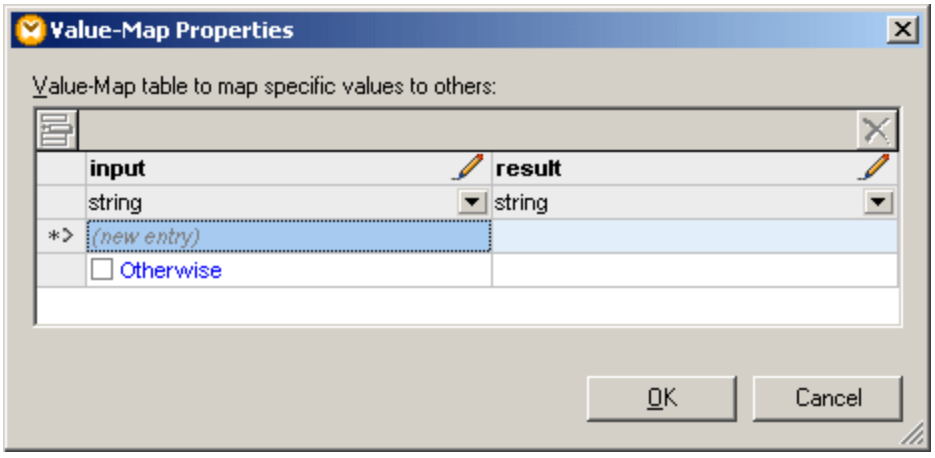
The Value-Map component allows you to transform an input value to a different output value using a lookup table. This is useful for converting different enumeration types. The component only has one input and output item.

Note: if you want to retrieve/filter data based on specific criteria, please use the **Filter** component see Lookup table - filtering dynamic data in the following section, Filtering XML data: Filtering data, Filtering CSV files: Creating hierarchies from CSV and fixed length text files.

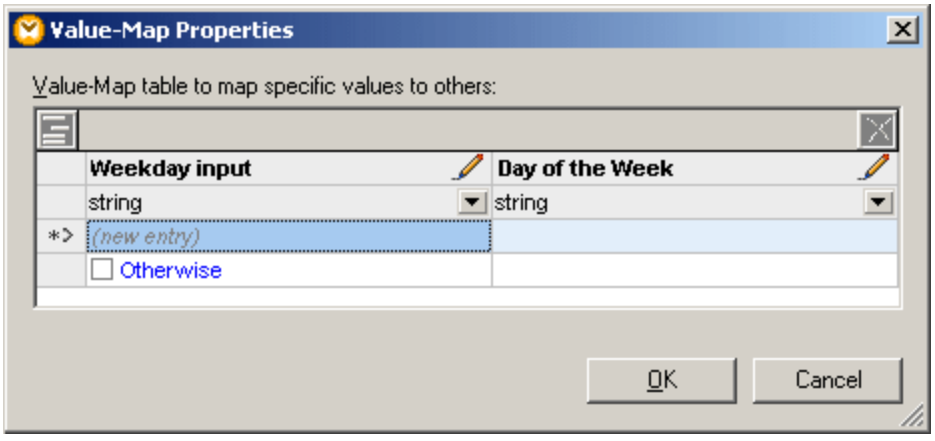
To use a Value-Map component:

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.

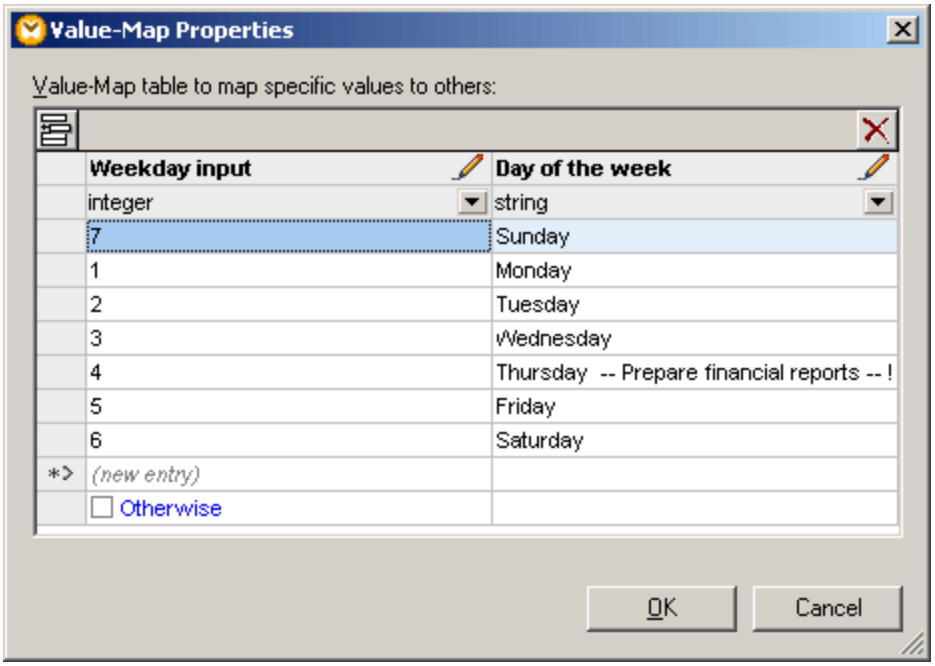


3. Click into the column headers and enter **Weekday input** in the first, and **Day of the Week** in the second.



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.

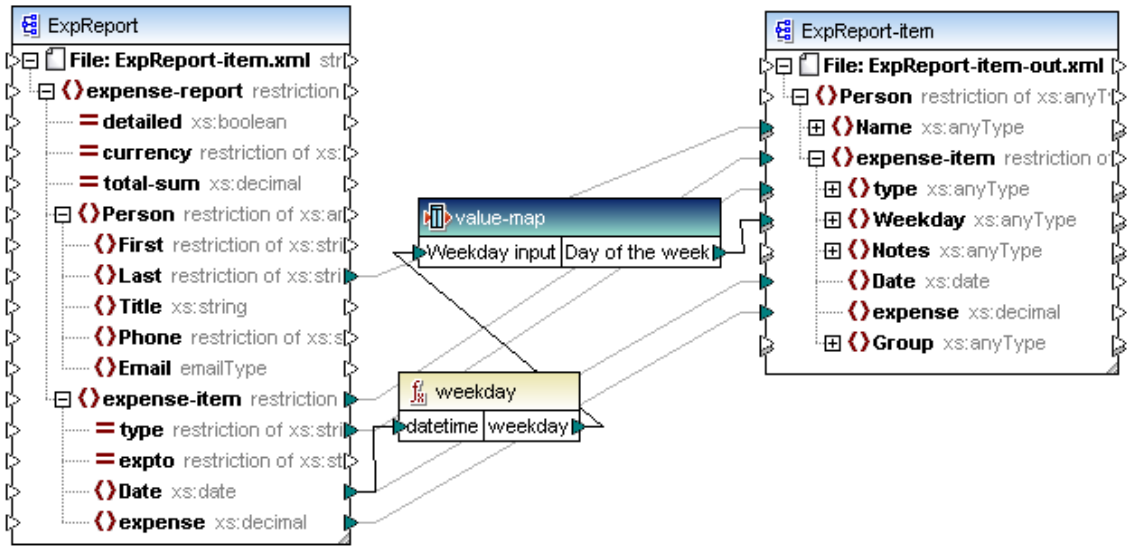
7. Click the **datatype** combo box, below the column header to select the input and output datatypes, e.g. integer and string.



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 "Thursday", then the corresponding output "Prepare Financial Reports" is mapped to the Weekday item in the target component.
- Clicking the Output tab displays the target XML file with the transformed data.

Note:

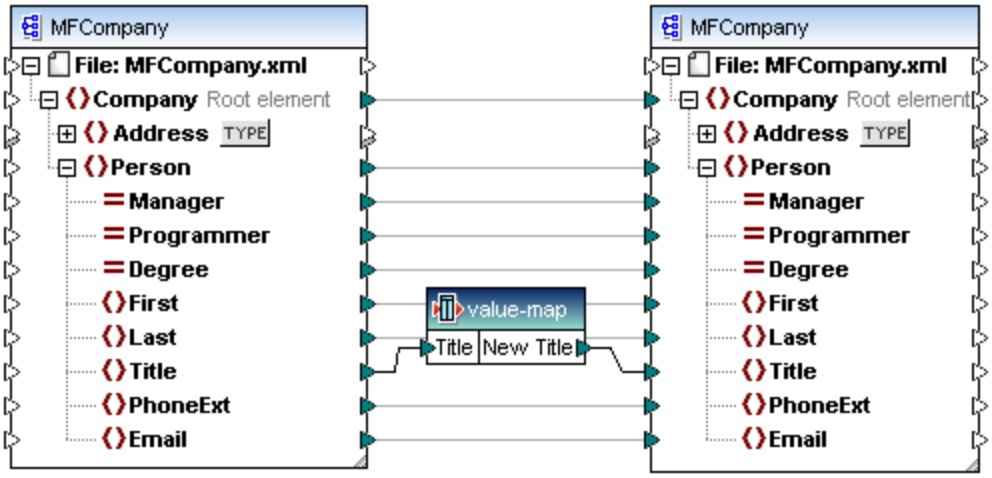
Placing the mouse cursor over the value map component opens a popup containing the currently defined values.

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".

11.2.1 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.

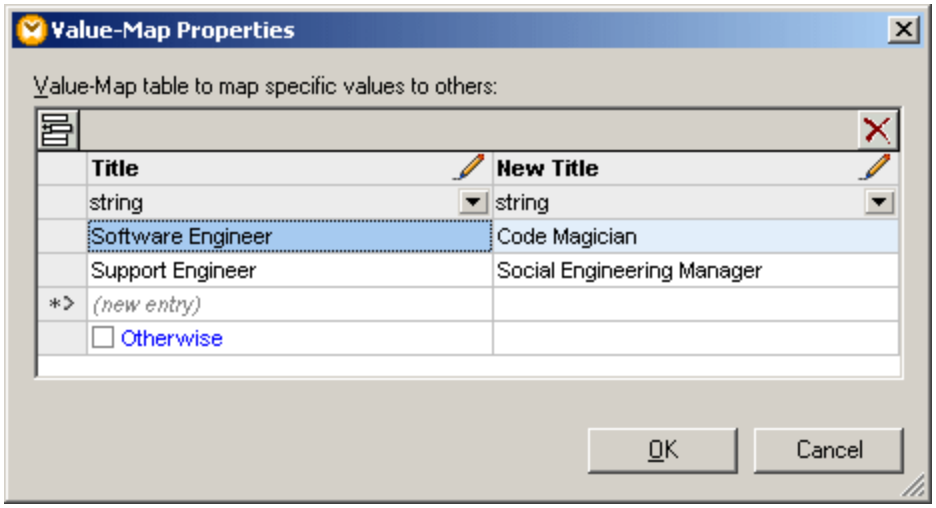
An example of this would be a company that changes some of the titles in a subsidiary. In this case it might change two title designations and want to keep the rest as they currently are.



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>
```

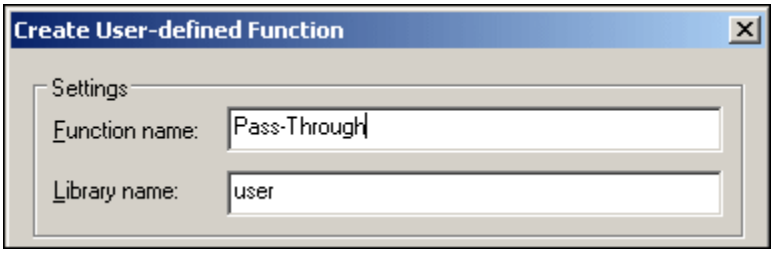
For those persons who are neither of the two types shown in the value-map component, the Title element is deleted in the output file.



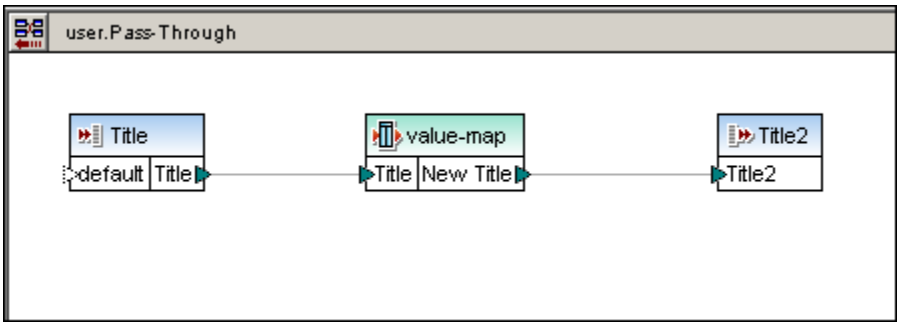
Possible alternative:
Clicking the **Otherwise** check box and entering a substitute term, does make the Title node reappear in the output file, but it now contains the same **New Title** for all other persons of the company.

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

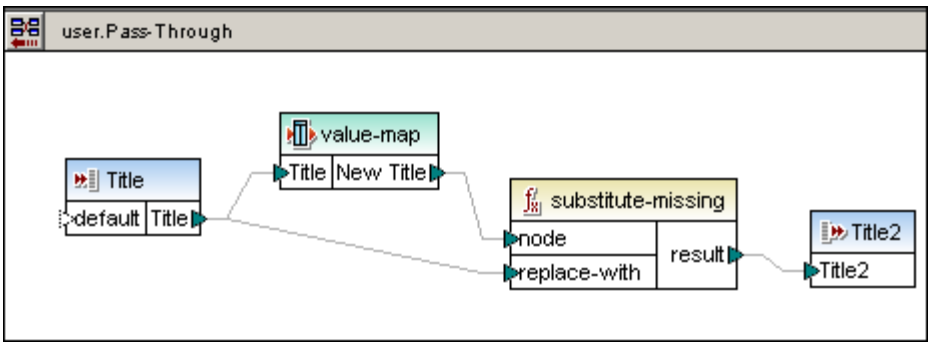
1. Click the value-map component and select **Function | Create user-defined function from Selection**.



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.

11.2.2 Value-Map component properties

Actions:

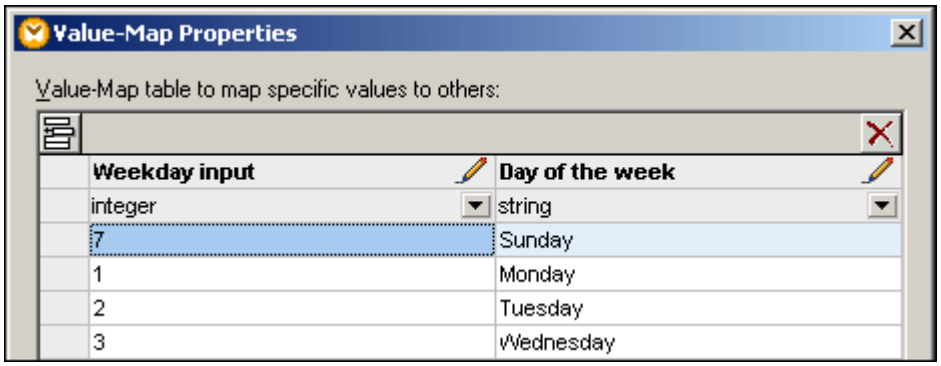
Click the insert icon to **insert** a new row before the currently active one.

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

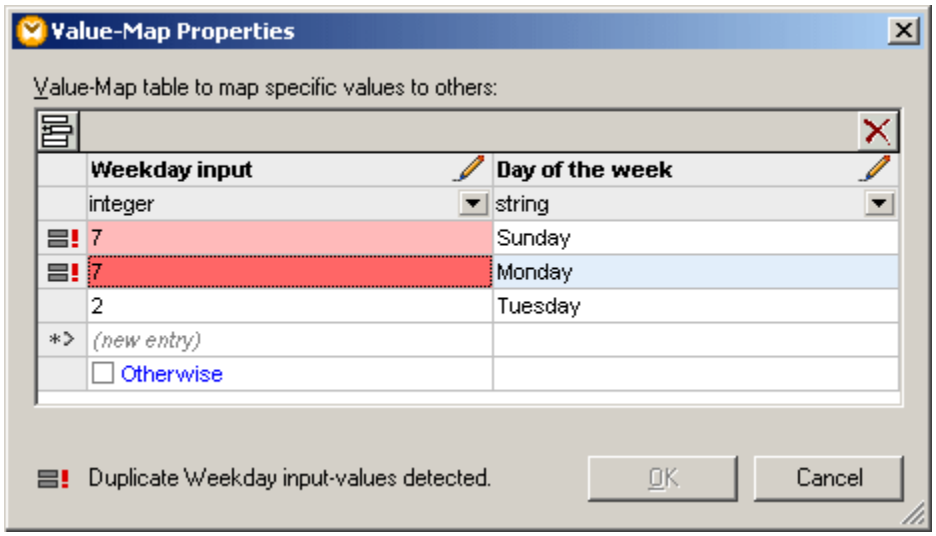
Click the edit icon to **edit** the column header.

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.

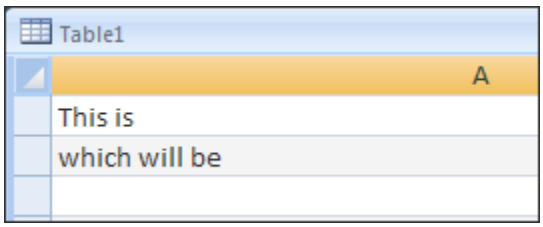


Having corrected one of the values, the OK button is again enabled.

11.3 Replacing special characters in database data

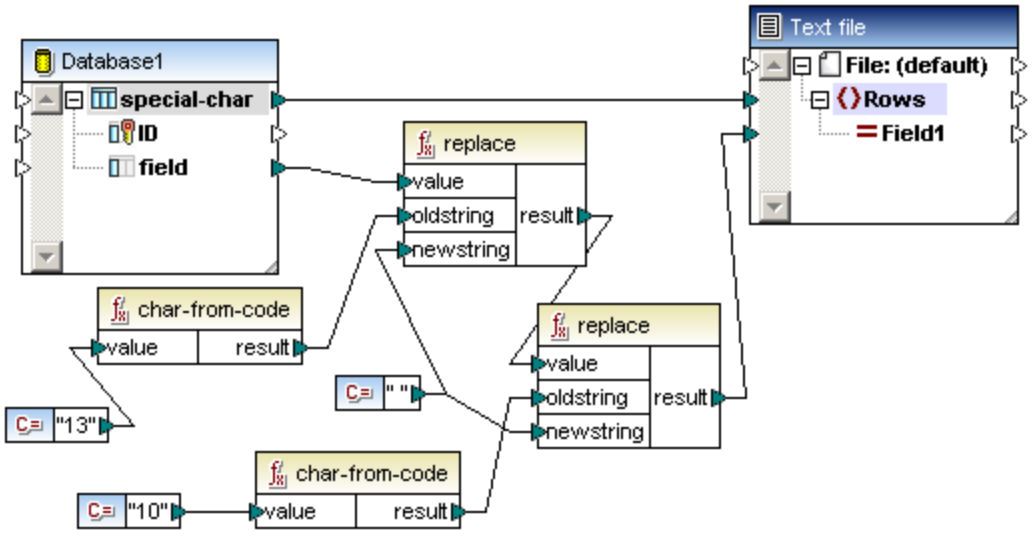
When transforming database data you might need to remove specific "special" characters e.g. newline (line break), CR/LF (hex 0D/0A) from the data source. This can be achieved by using the char-from-code function of the string function library.

The mapping shown below is available as **special-char.mfd** in the ...\MapForceExamples\Tutorial\ folder. 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.

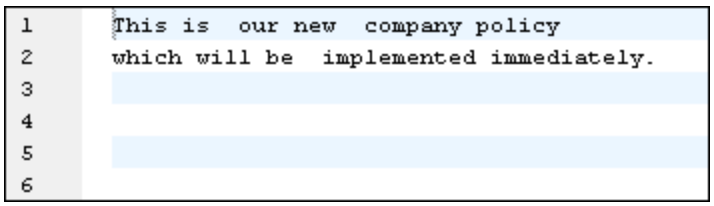


What the mapping does is:

- 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.



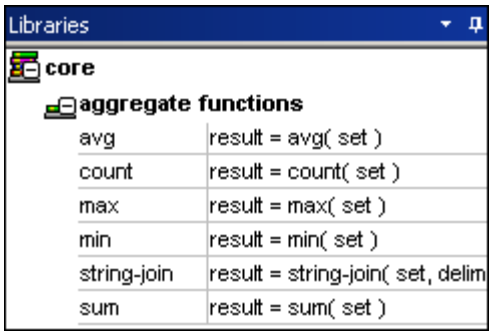
Clicking the Output tab shows the result in the preview window. The (CR) LF characters within each database field have been replaced by a space.



11.4 Aggregate functions: min, max, sum, count, avg

Aggregate functions perform operations on a set of input values (or a sequence of values) as opposed to a single value. The aggregate functions can all be found in the core library. The mapping shown below is available as **Aggregates.mfd** in the ...\Tutorial folder.

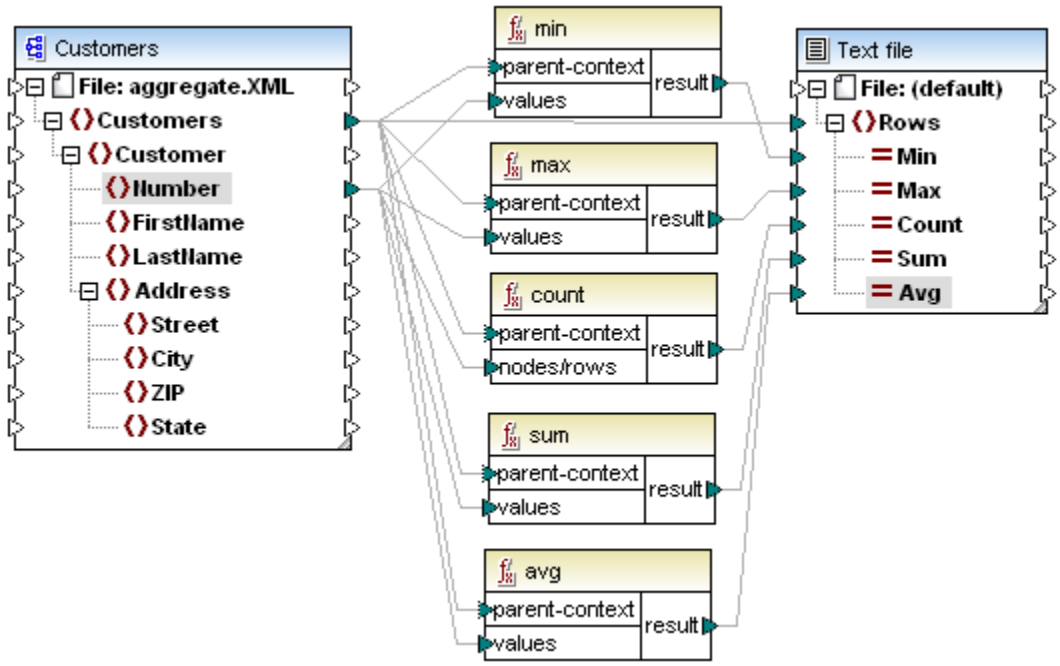
Please also see Aggregate functions - summing nodes in XSLT1 and 2 on how to create aggregate functions using Named Templates.



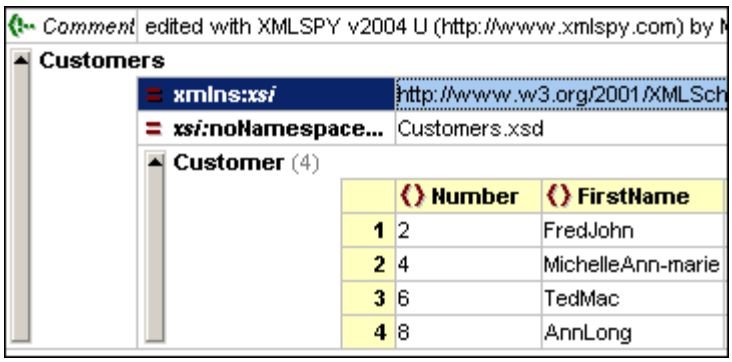
Java Selected

Aggregate functions have two input items.

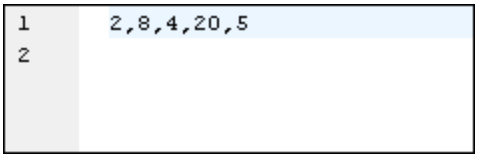
- **values** is connected to the source item that provides the data, in this case Number.
- **parent-context** is connected to the item you want to iterate over, i.e. the context, in this case over all Customers. The parameter is, howerver, optional.



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.

11.5 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.

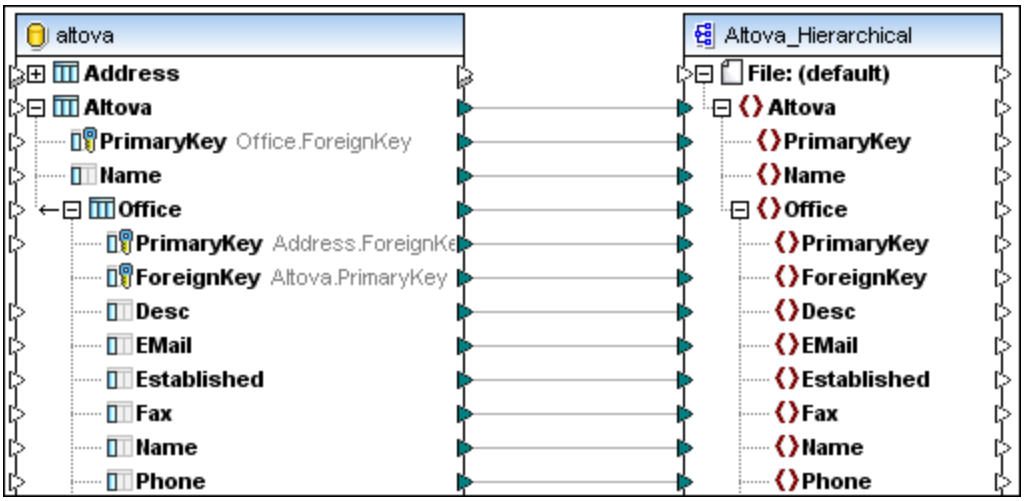
The "**DB_Altova_Hierachical.mfd**" sample file in the **...\MapForceExamples** folder shows how this can be achieved when mapping from an hierarchical database. The Altova_Hierarchical.xsd schema is also supplied in the same folder. The schema structure is practically identical to the Access database 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.

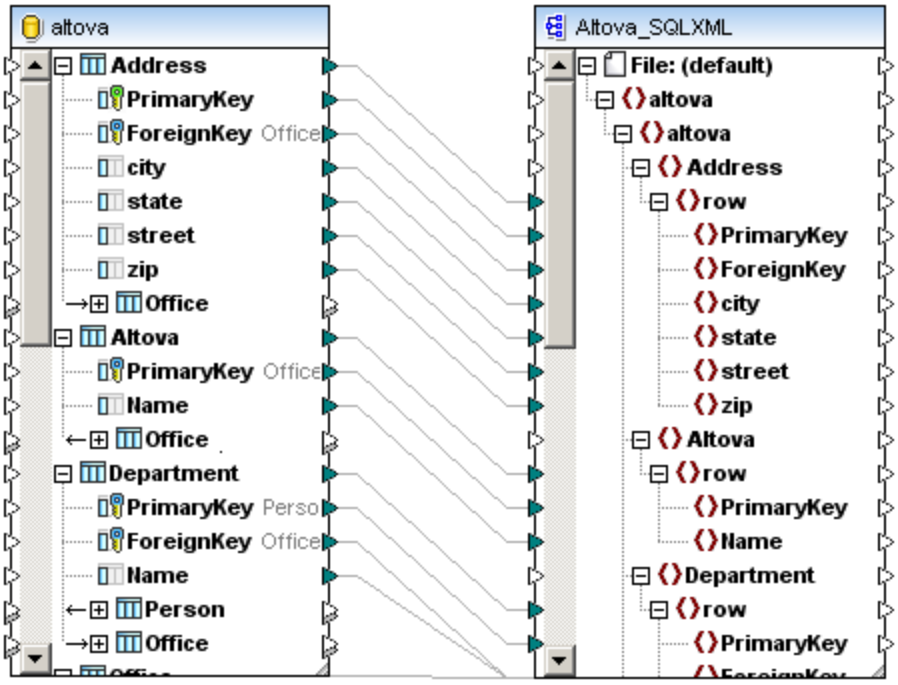
  The diagram below shows the mapping of the hierarchical Access database to Altova_Hierarchical.xsd.



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



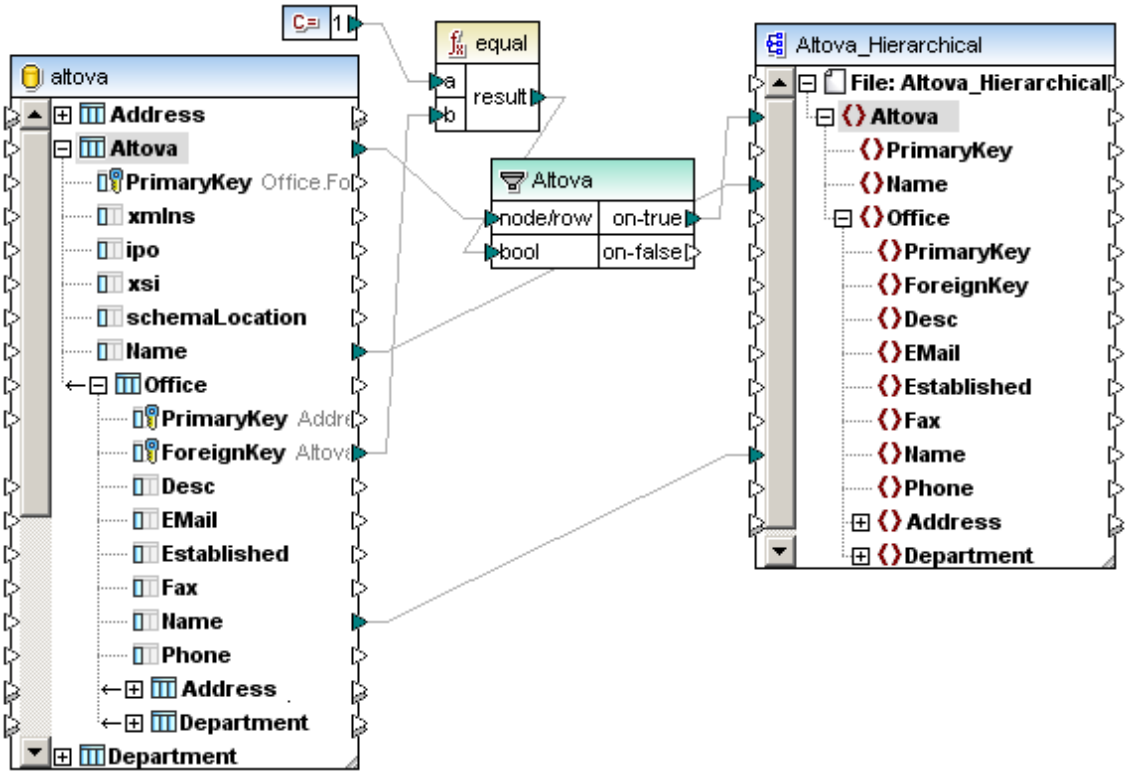
- Please note that the above example **DB_Altova_SQLXML.mfd**, does not preserve the table relationships, as mappings are created from several different "root" tables.

11.6 Mappings and root element of target documents

Root element of target XML files
When creating a mapping to the **root** element of the target Schema/XML file, please make sure that only one element, or record, is passed on to the target XML, as an XML document may only have one root element.

Use the filter component 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.



Root element not limited:
If you do not limit the target schema root element, then all source elements/records are inserted between the first root element. This will still create well-formed, but not valid, XML files.

11.7 Boolean comparison of input nodes

Data type handling in boolean functions (first introduced with MapForce 2006 SP2)
During the evaluation of the core functions, less-than, greater-than, equal, not-equal, less equal, and greater equal, the evaluation result of two input nodes depends on the input values as well as the data types used for the comparison.

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.

Differing input node types
If the input nodes are of differing types, e. g. integer and string, or string and date, then the following rule is applied:

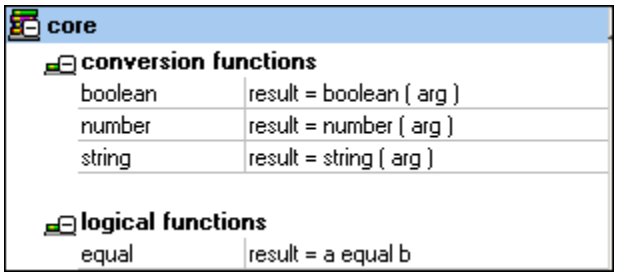
The data type used for the comparison **is always the most general, i. e. least restrictive, input data type** of the two input types.

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'.

The type handling for comparing mixed types follows the XSLT2 guidelines and prevents any content-sensitive type conversion strategies. The advantage is that the logic is fixed by the mapping and does not change dynamically.

Additional checks:
Version 2006SP2 additionally checks mappings for incompatible combinations and raises validation errors and warnings if necessary. Examples are the comparison of dates with booleans, or "datetimes" with numerical values.

In order to support explicit data type conversion, the following **type conversion** functions are available in the core library: "boolean", "number" and "string". In the previously mentioned context, these three functions are suitable to govern the interpretation of comparisons.



Adding these conversion functions to input nodes of related functions might change the common data type and the result of the evaluation in the desired manner. E. g. if string nodes store only numeric values, a numerical comparison is achieved by adding the "number" conversion function (in the **conversion** section of the **core** library) to each input node.

11.8 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.

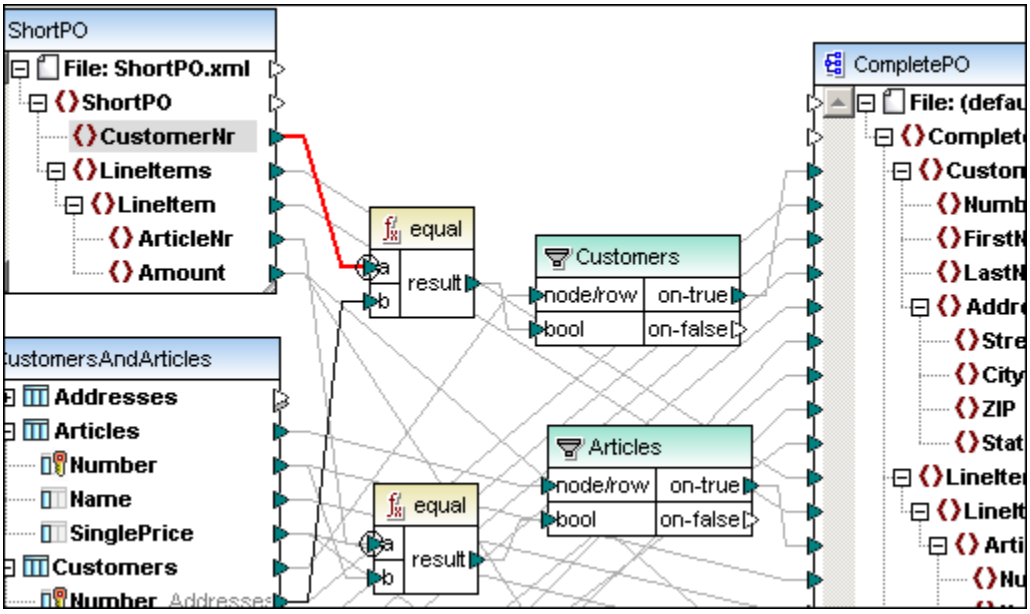
A simplified version of the complete **DB-CompletePO.mfd** file available in the **...\MapForceExamples** folder, is shown below.

Please 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 schema. 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.

- 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** function (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.

Designating the **b** parameter of the equal function (i.e. item Number), as the **priority context** would cause:

- MapForce to load the first Number into the **b** parameter
- Check against the **CustomerNr** in **a,** if not equal
- 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.



Priority context and user-defined functions:

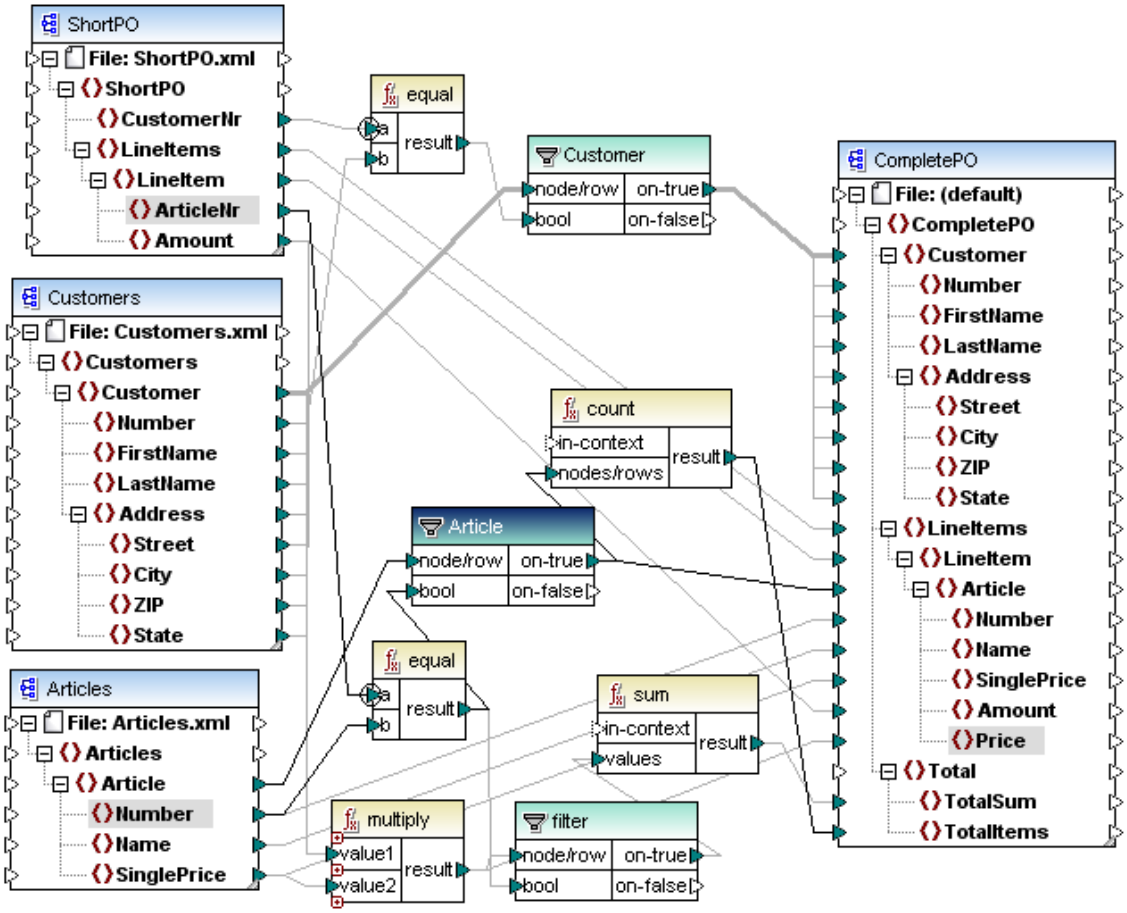
If a user-defined function has been defined of type "inline", the default setting, then a priority context cannot be defined on one of the parameters of the user-defined function. The user-defined function can, of course, contain other regular (non-inlined) user-defined functions which have priority contexts set on their parameters.

11.9 Merging multiple files into one target

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 "Dynamic file names - input / output".

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 out 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.

11.10 Command line parameters

The command line parameter syntax for MapForce is shown below.

General syntax:
MapForce.exe *filename* **[** ***/****target outputdir* **[STYLEVISION:***SVOptions***] [/GLOBALRESOURCEFILE** *globalresourcefilename* **] [/GLOBALRESOURCECONFIG** *configurationname* **] [/LOG** *logfilename* **]] [/-** *runtimeparameters* **]**

target	**{ BUILTIN | GENERATE | XSLT | XSLT2 | XQuery | JAVA | CS[:***CSOptions***] | CPP[:***CPPOptions***] }**
CSOptions	**{ VS2010 | VS2008 | VS2005 }**
CPPOptions	**{ VC10 | VC9 | VC8 },{ MSXML | XERCES | XERCES3},{ LIB | DLL },{ MFC | NoMFC }**
runtimeparameters	***/****name1 value1* **[** ***/****name2 value2* **]...**

- The square brackets **[...]** denote optional parameters.
- The 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.

Description of general parameters:

filename	path of YourMAPFORCEfile.MFD, or YourMAPFORCEproject.MFP **If the path, or file name contains a space, please use quotes around the path/file name i.e. "c:\Program Files\...\Filename"**
outputdir	For code generation, the directory the generated mapping code is to be placed in. *outputdir* is now optional if you use the */target* parameter. If an output path is not supplied, the working/current directory will be used. *outputdir* is also used as the base directory for relative input or output file names when executing the mapping using the /BUILTIN parameter. Relative means only the file name is supplied, not a complete path starting with a drive letter.
/LOG *logfilename*	generates a log file called *logfilename*. *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 current directory.

Description of target parameters:

/BUILTIN	generates all outputs using the Built-in execution engine (without you having to generate code)

/STYLEVISION:*SVOptions*	Must be used together with the BUILTIN parameter, has no effect when generating code.
/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.* If a target language is specified, e.g. Java, then that language is used for all project folders, overriding the current folder settings.
/XSLT	generates XSLT1 code
/XSLT2	generates XSLT2 code
/XQuery	generates XQuery code
/JAVA	generates the Java application
/CS	generates the C# application using the configuration of the mapping settings
/CS:*CSOptions*	generates the C# application using special configuration given in option-field of the command-line parameters
/CPP	generates the C++ application using the configuration of the mapping-settings
/CPP:*CPPOptions*	generates the C++ application using special configuration given in options-field of the command-line parameters
/- /*runtimeparameters*	Can only be used when using the BUILTIN parameter, has no effect when generating code. All following parameters only affect "input parameters"/components. ... BUILTIN /- /X 10 /Y Fred, sets the value of the input component X to 10 and the value of the input component Y to Fred.

SVOptions:

/STYLEVISION: HTML,RTF,PDF,DOCX	Generates any combination of the output formats. Must be used with the /BUILTIN parameter. If the /STYLEVISION parameter is used, at least one of the output formats e.g. HTML must be supplied.

Description of C# options:

VS2010	generates Microsoft Visual Studio 2010 solution files
VS2008	generates Microsoft Visual Studio 2008 solution files
VS2005	generates Microsoft Visual Studio 2005 solution files

Description of C++ options:

VC10	generates Microsoft Visual Studio 2010 solution files
VC9	generates Microsoft Visual Studio 2008 solution files
VC8	generates Microsoft Visual Studio 2005 solution files
MSXML	generates code using MSXML 6.0
XERCES	generates code using Apache Xerces 2.x (2.6 and later)
XERCES3	generates code using Apache Xerces 3.x
LIB	generates code for static libraries
DLL	generates code for dynamic-linked-libraries

MFC	generates code supporting MFC
NoMFC	generates code without MFC support

Please note:
VC6 workspace files are always generated for C++.

Description of global resource parameters:

/GLOBALRESOURCEFILE *globalresourcefilename*	uses the global resources defined in the specified global resource file
/GLOBALRESOURCECONFIG *configurationname*	uses the specified global resource configuration

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 (target XML document, and databases) using the Built-in transformation engine.
MapForce.exe *filename.mfd* **/BUILTIN** *outputdir*

VII)
generate all output files using the Built-in transformation engine and define the input and output **file names** when executing the command, also generate a log file.

Mapforce.exe *filename.mfd* **/BUILTIN /LOG** *logfilename* **/- /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.

When using the Built-in transformation engine, the command line values supersede the values defined in the mapping. Please see Command line - defining input/output files on how to define the input and output parameters/files.

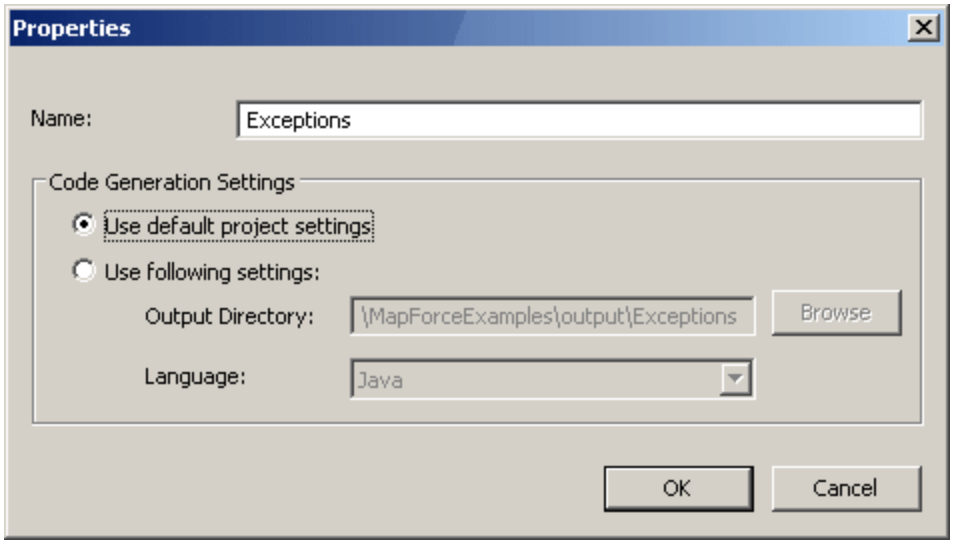
VIII)
generate all output files using the Built-in transformation engine using global resources of the global resource file for the specified configuration
Mapforce.exe *filename.mfd* **/BUILTIN** *outputdir* **/GLOBALRESOURCEFILE** *globalresourcefilename* **/GLOBALRESOURCECONFIG** *configurationname*

IX)
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 folder.



X)
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.

XI)
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.

Please see: Command line - defining input/output files on how to define input / output parameters (to supply file names) when executing the command line.

XII)
Produce formatted reports of a mapping using StyleVision and output an HTML and PDF document of the mapping.
MapForce.exe *filename.mfd* **/BUILTIN** *outputdir* **/STYLEVISION:***SVOptions* **/LOG** *logfilename*

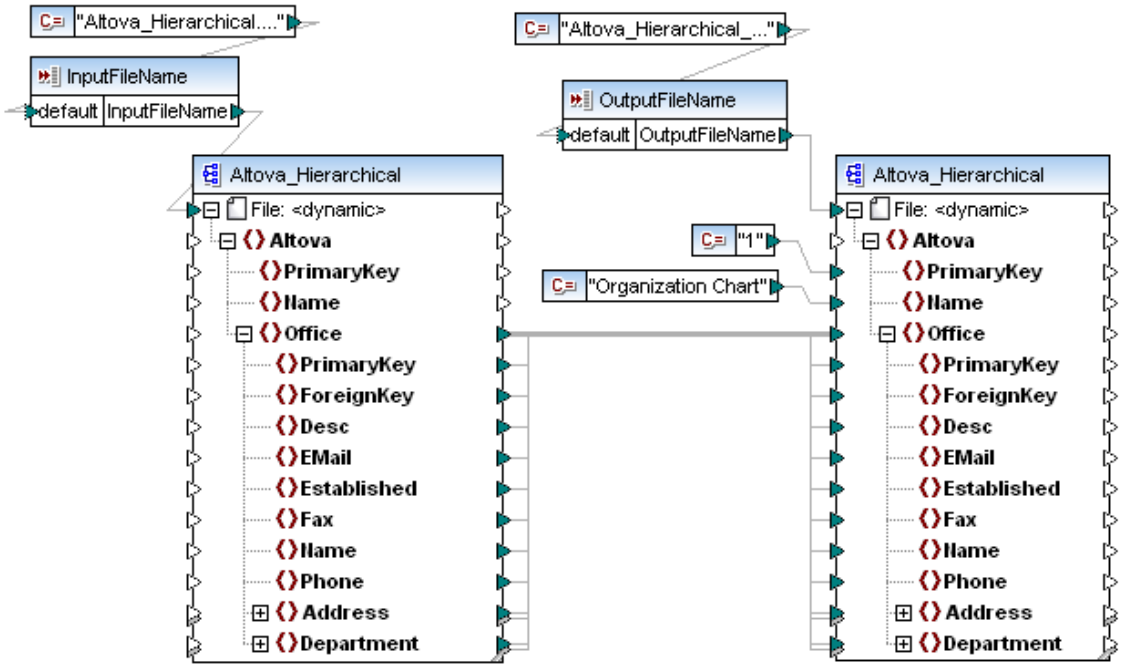
MapForce.exe myMapping.mfd /**BUILTIN** /**STYLEVISION**:HTML,PDF /**LOG** mylogfile.txt

11.11 Command line - defining input/output files

MapForce allows you to create special "**input"** components that can act as **parameters** in the command line execution of a mapping. These input components allow you to define the content of the parameter when executing the command line, as well as the values when previewing the output in MapForce. The content of a parameter can be a file name, or a specific value.

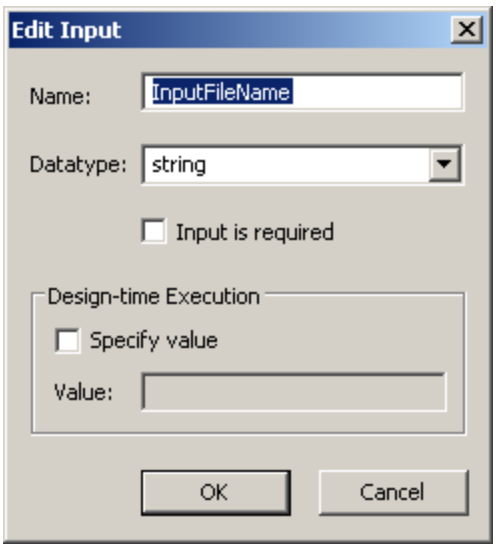
This specific type of "input" component cannot be used **inside** a user-defined function, it is only available in the main mapping window.

The mapping below (**FileNamesAsParameters.mfd** in ...\MapForceExamples) uses two such components, one to supply the input file name and the other, the output file name, to the File: <dynamic> item of each component. See Dynamic input/output files per component for more information on dynamic file names.



To define an input component / command line parameter:

1. Use the menu option **Function | Insert Input** to insert the component.
   This opens the Create Input dialog box.
2. Enter a name for the function e.g. **InputFileName** and select the datatype you want to use, e.g. string.



3. Click OK to complete the definition.
4. Use the same method to create the **OutputFileName** component/parameter.

The **/InputFileName** and /**OutputFileName** components are the special input components in the mapping that allow you to use them as file name parameters in the command line execution.

The other fields in this dialog box are used to specify the preview output (or files) while running MapForce. They are not needed when executing from the command line. The two constant components (shown in the mapping above) supply default file names to the input components for the preview output. See Input parameters - default and preview settings.

Supplying the Input and Output file names for command line execution:

Using the FileNamesAsParameters.mfd as an example, enter the command line in the form shown below.
Mapforce.exe *filename.mfd* **/BUILTIN /LOG** *logfilename* **/**runtimeparameters **/InputFileName** *inputfilename* **/OutputFileName** *Outputfilename*.

Using our example the command line could be something like this:
Mapforce.exe FileNameAsParameters.mfd /**BUILTIN /LOG** log.txt **/- /InputFileName** "altova-cmpy.xml" /**OutputFileName** cmdout.xml.

What it does:

- Uses the mapping file FileNameAsParameters.mfd
- Uses the Built-in execution engine to generate the mapping output (in the cmdout.xml file)
- Creates a log.txt file which containing any execution messages
- Supplies **altova-cmpy.xml** as the input file name
- Supplies **cmdout.xml** as the output file name, and writes the mapping result into it

The ***/-*** switch (runtime parameters) can only be used when using the BUILTIN, it has no effect when generating code. It signals that all following parameters only affect the special "input" parameters/components

Wildcards in command line execution (BUILTIN):

You can use wildcards ? or *. in the command line, e.g. /InputFileName **altova-*.xml**.

Place the wildcards in quotes if spaces exist in the path or file names.

Calling compiled programs from the command line:

To supply input / output file names to the compiled FileNameAsParameters.mfd mapping.

1. Generate Java code for the mapping in MapForce, **File | Generate code in Java**.
2. Make your generated Java code into an executable JAR file using your preferred IDE, e.g. myMapping.jar.
3. Enter the command line:

   java -jar myMapping.jar /**InputFileName** altova-cmpy.xml /**OutputFileName** altova-out.xml

Wildcards in command line execution (Java):

When calling a generated and compiled Java program from the command line, be sure to place the wildcard parameters in quotes.

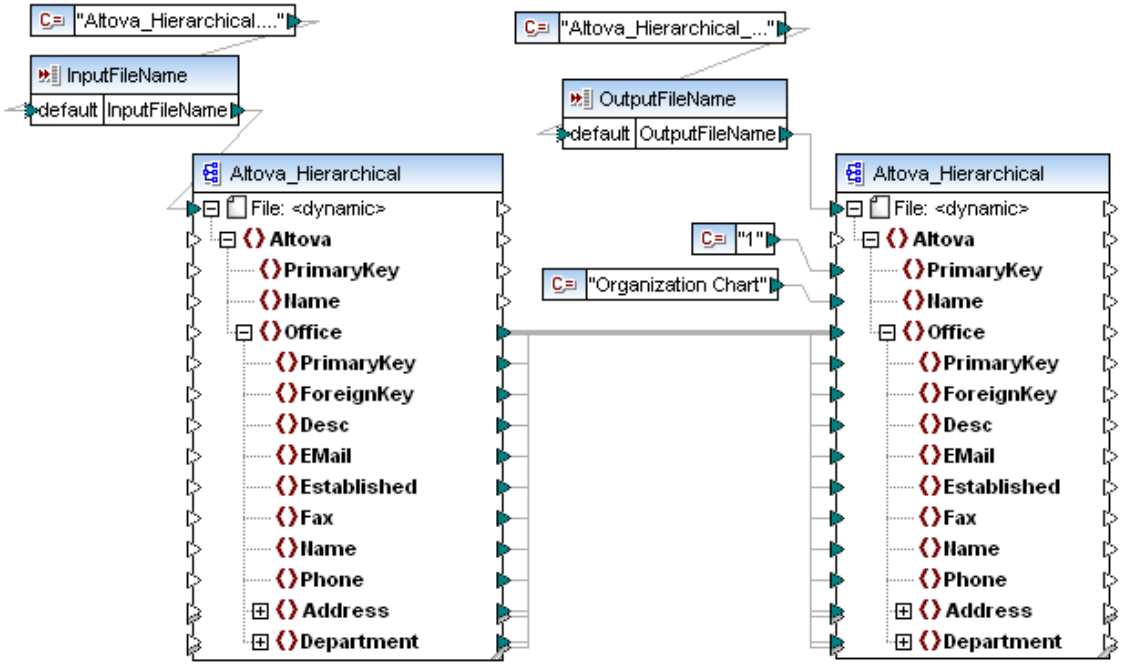
Correct usage:

java -jar myMapping.jar /InputFileName **"altova-*.xml"** /OutputFileName altova-out.xml

11.12 Input parameters - default and preview settings

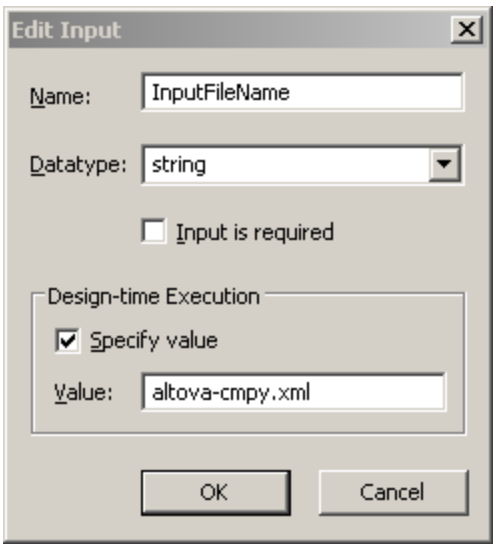
The mapping below (available as **FileNamesAsParameters.mfd** in ...\MapForceExamples folder) uses two input components, one to supply the input file name and the other, the output file name, to the File: <dynamic> item of each component.

The **/InputFileName** and **/OutputFileName** components are the special input components in the mapping, that allow you to use them as file name parameters in the command line execution. If you want to define the input data from the command line, please see Command line - defining input/output files.



To define an input component:

1. Use the menu option **Function | Insert Input** to insert the component.
   This opens the Create Input dialog box.
2. Enter a name for the function e.g. **InputFileName** and select the datatype you want to use, e.g. string.



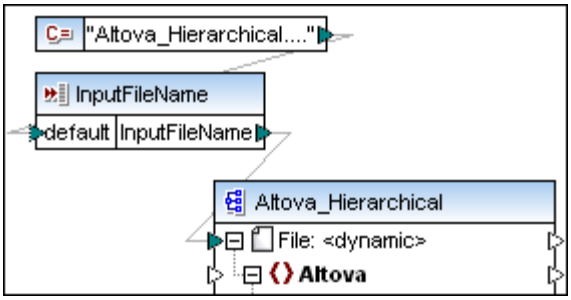
3. Click the "Specify value" check box, and enter a value/string in the textbox, that you want to use when previewing the output, e.g. altova-cmpy.xml.
4. Click the OK button then the Output button to see the result.
   The altova-cmpy.xml is used as the source file for the mapping and the mapped data appears in the output window.

   The value entered in the Design-time Execution **value** field is **only** applicable when **previewing** results in the **Output** tab. It is not used for code generation, or command line execution of mappings.

To define a default value for the output preview:

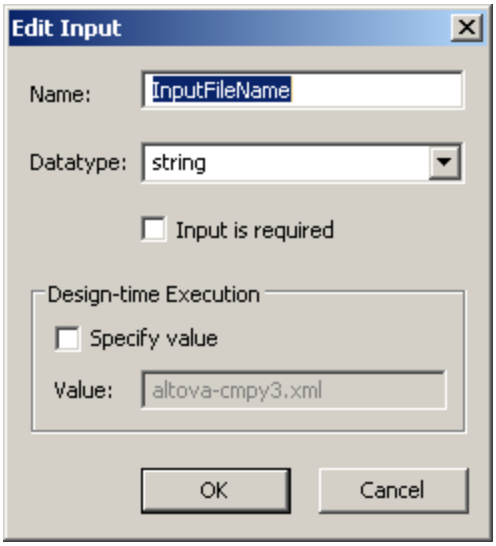
Having defined the input component and clicked the OK button, the component appears in the mapping area as shown below. Note that a default item appears on the left, while the component name is the name of the other mappable item.

1. You can use a Constant component to supply the default value as shown in the example above, e.g. Altova_Hierarchical.xml.
2. Create the constant component and connect it to the default item of the input component.



To use the default value of an Input component:

1. Double click the input component and **deselect** the "Input is required" and "Specify value" check boxes.



2. Click the Output tab to see the result of the mapping. The data from the Altova_Hierarchical.xml file is now used for the preview.

Generating XSLT 1.0 or XSLT 2.0 code:
The value in the "Value" text box is written into the **DoTransform.bat** batch file. This batch file is automatically generated for execution in the AltovaXML engine. If you want to use a different input (or output) file you can change the name in the batch file.

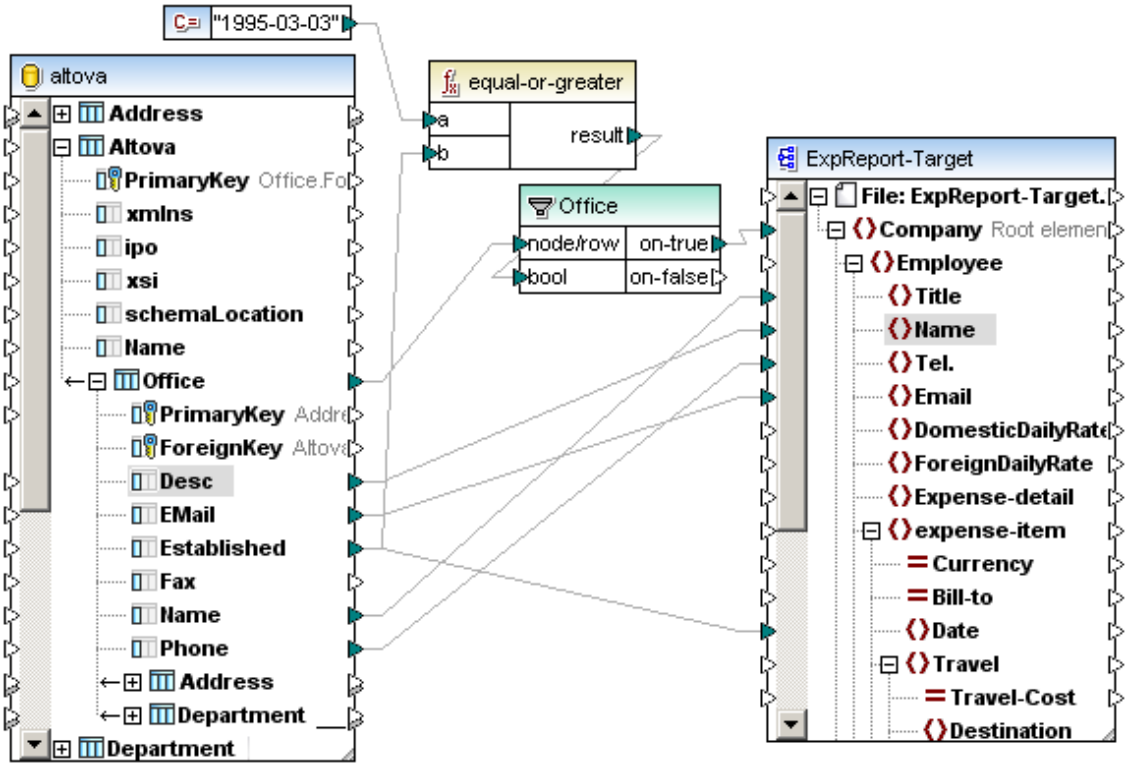
If no value has been entered in the "Value" text box, then the default value, present in the generated XSLT code, will be used.

Using default parameters/values in command line execution: ?? true ??
The "Input is required" check box must be set to **inactive** (before compiling the code) to use the default parameter from the command line. Enter **mapping.exe** to have the generated code use the default parameter from the command line.

11.13 Filtering database data by date

The example below shows how you can use the filter component to filter out 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, and is of type string.
- 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.

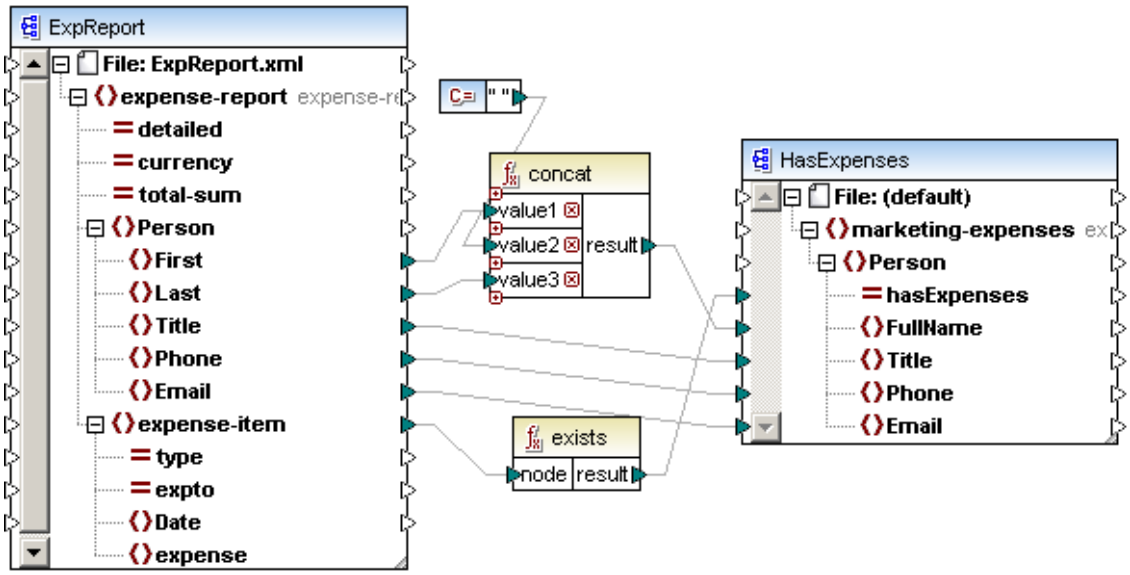


11.14 Node testing, position and grouping

The node testing functions allow you to test for the existence of nodes in the **XML instance** files. Elements or attributes defined as optional in the XML Schema, may, or may not, appear in the XML instance file. Use these functions to perform the specific node test and base further processing on the result.

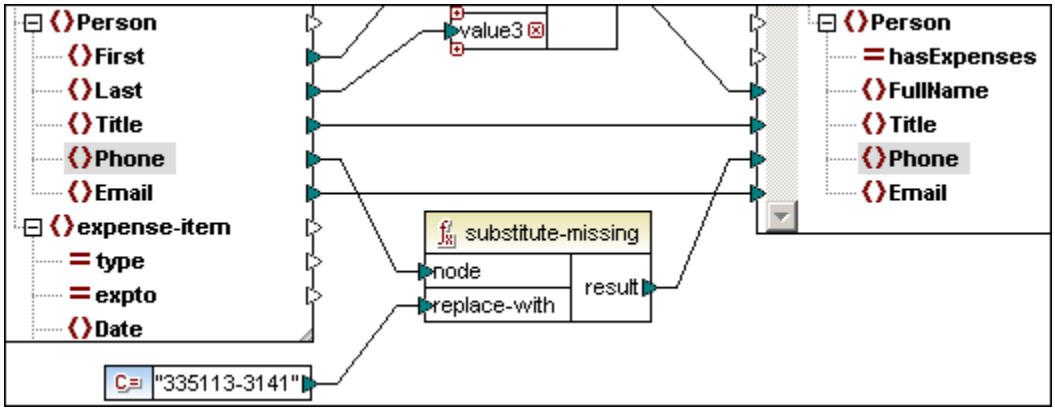
Exists
Returns true if the node exists, else returns false.
The "**HasMarketingExpenses.mfd**" file in the ...\MapForceExamples folder contains the small example shown below.

If an expense-item exists in the source XML, then the "hasExpenses" attribute is set to "true" in the target XML/Schema file.



Not-exists
Returns false if the node exists, else returns true. Please see Mapping missing nodes - using Not-exists for an example on how to map missing nodes.

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.



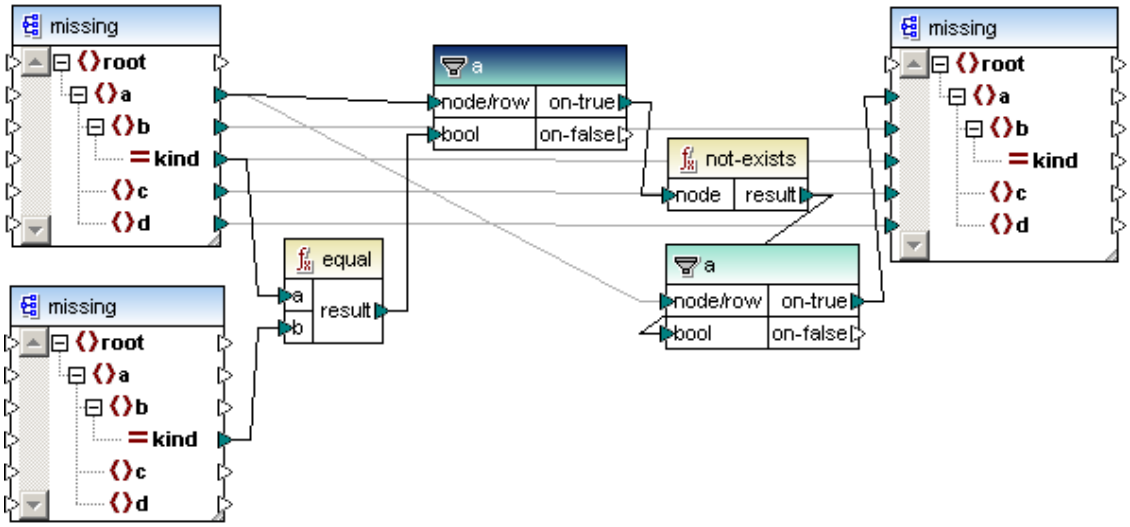
In the image above, the existence of the node "Phone" is checked in the XML instance file. If the node is not present, then the value supplied by the constant is mapped.

11.14.1 Mapping missing nodes - using Not-exists

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 is to:

- 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** at left, contains the node <b **kind="3"**>, which is missing from b.xml.
- **b.xml** at right, contains the node <b **kind="4">** which is missing from a.xml.

a.xml

```
<root xmlns:xsi="http://www.w3.
    <a>
        <b kind="1">b1</b>
        <c>c1</c>
        <d>d1</d>
    </a>
    <a>
        <b kind="2">b2</b>
        <c>c2</c>
        <d>d2</d>
    </a>
    <a>
        <b kind="3">b3</b>
        <c>c3</c>
        <d>d3</d>
    </a>
</root>
```

b.xml

```
<root xmlns:xsi="http://www.w3.
    <a>
        <b kind="1">foo</b>
        <c>foo</c>
        <d>foo</d>
    </a>
    <a>
        <b kind="2">foo</b>
        <c>foo</c>
        <d>foo</d>
    </a>
    <a>
        <b kind="4">foo</b>
        <c>foo</c>
        <d>foo</d>
    </a>
</root>
```

- The **equal** function compares the **kind** attribute of both XML files and passes the result to the filter.
- A **not-exists** function is placed after the initial filter, to select the missing nodes of each of the source files.
- The second filter is used to pass on the missing node and other data **only** from the a. xml file to the target.

The mapping result is that the node missing from b.xml, <b kind="3">, is passed on to the target component.

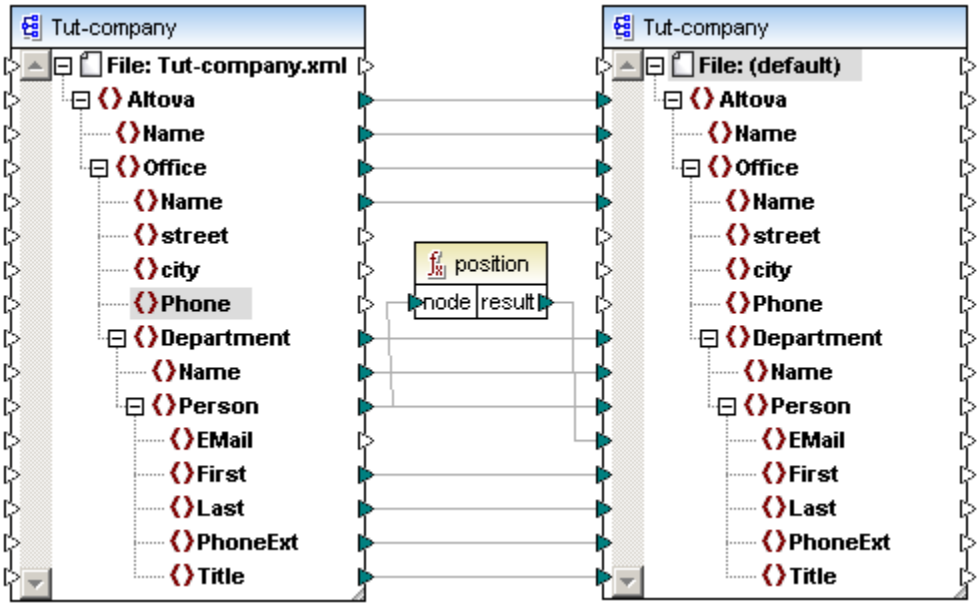
```
<root xsi:noNamespaceSchemaLocation="C:/DOCUME
    <a>
        <b kind="3">b3</b>
        <c>c3</c>
        <d>d3</d>
    </a>
</root>
```


11.14.2 Position of context items in a 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.

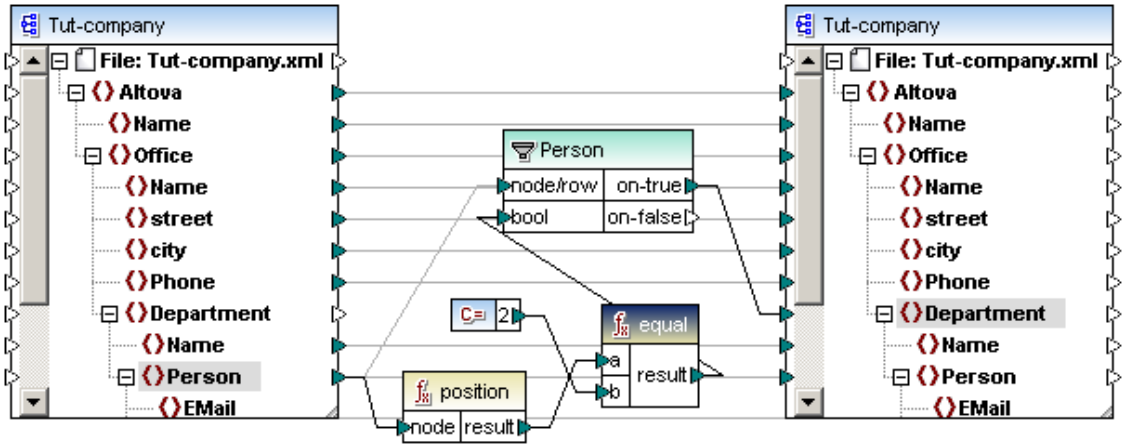


The position number is reset for each Department in the Office.

```
<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.

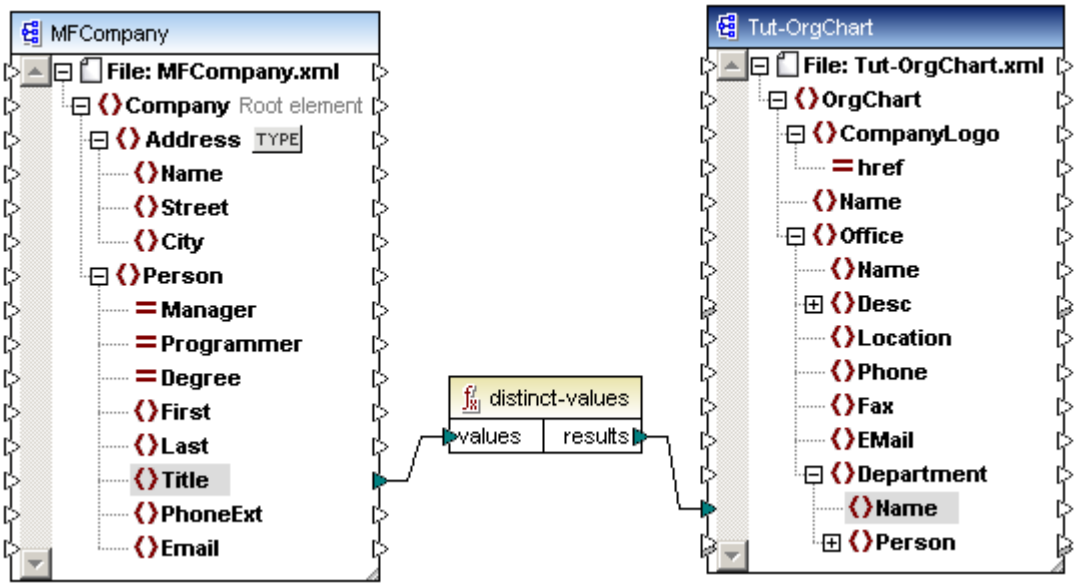
```
<Office>
  <Name>Microtech, Inc.</Name>
  <Department>
    <Name>Admin</Name>
    <Person>
      <EMail>b.bander@microtech.com</EMail>
      <First>Bert</First>
      <Last>Bander</Last>
      <Title>Accounts Receivable</Title>
    </Person>
  </Department>
  <Department>
    <Name>Sales and Marketing</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>
```


11.14.3 Grouping nodes / node content

MapForce now supports the grouping of nodes and their content. These functions can be found in the "Sequence functions" section in the Libraries window.

distinct-values
Allows you to remove duplicate values from a result set 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.

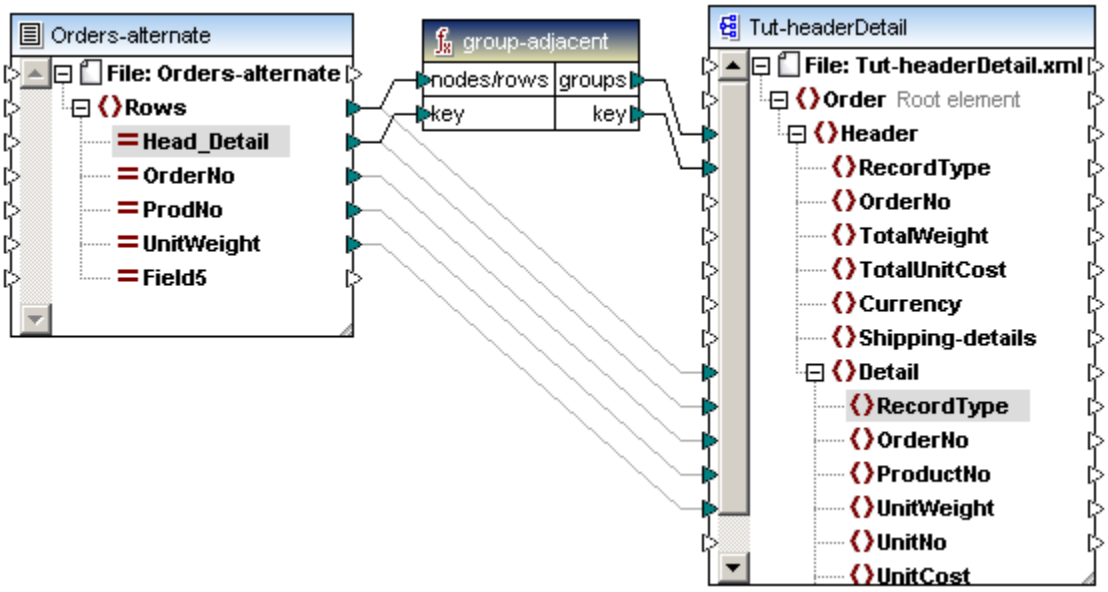


Note that the sequence of the individual Title items in the source component are retained when mapped to the target component.

```
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>
```

group-adjacent
Groups the input sequence into a series of groups where each set of identically adjacent items/

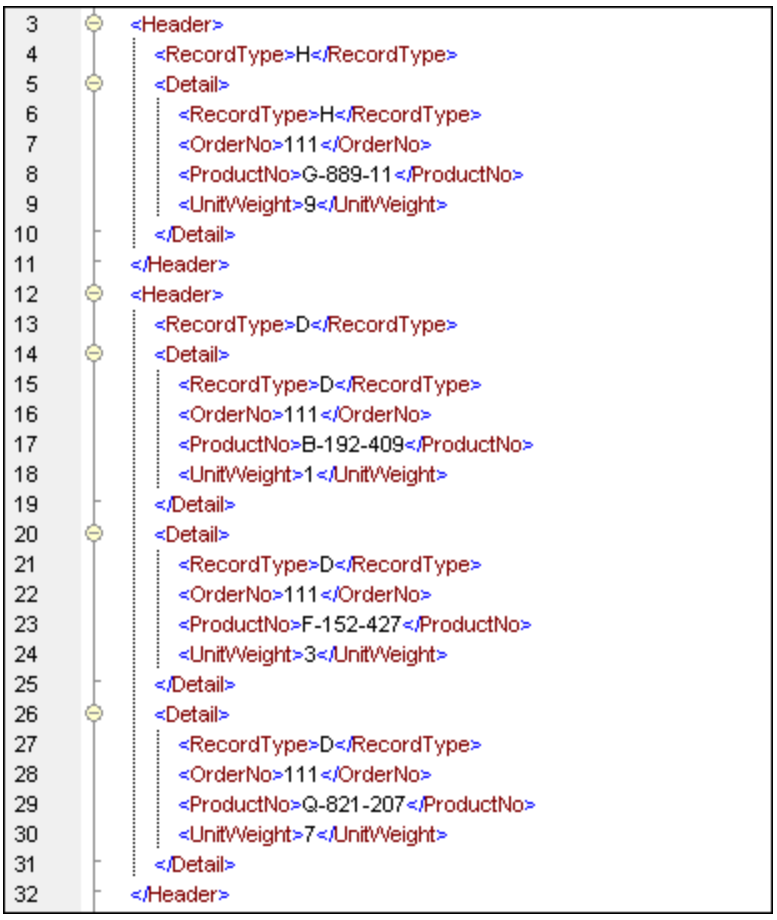
nodes are placed into a new separate group.



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.

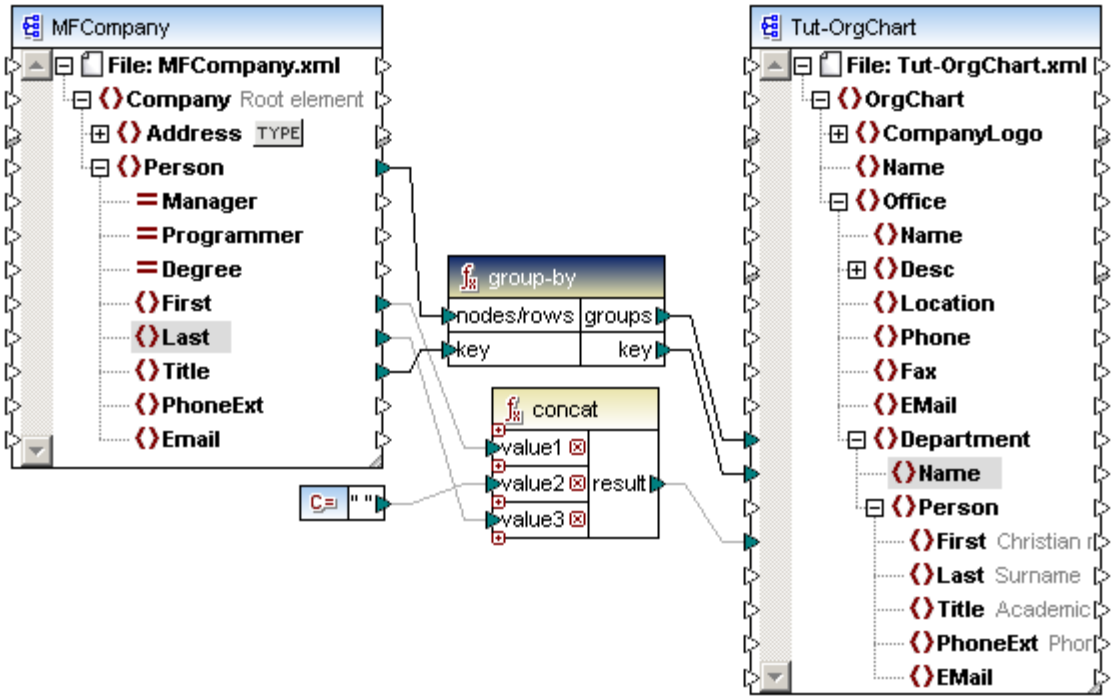


Note that group-adjacent uses the **content** of the node/item as the grouping key! The content of the Head_Detail field is used to group the records by record type in the target.

group-by

Groups the input sequence by distinct keys and outputs the series of groups along with their keys. 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 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 Ouput button shows the result of the grouping process.

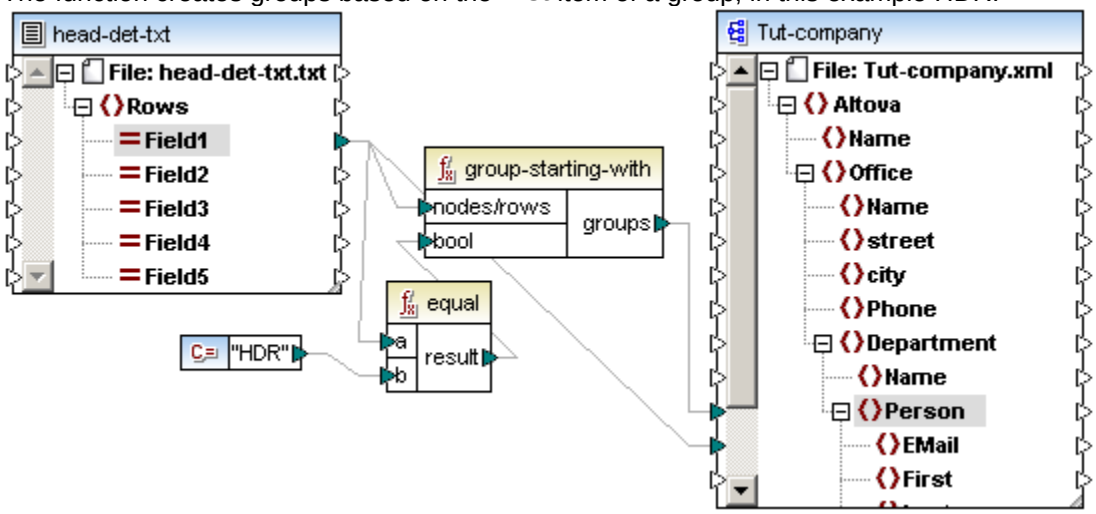
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <OrgChart xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart C:/DOCU
3    <Office>
4      <Department>
5        <Name>Office Manager</Name>
6        <Person>
7          <First>Vernon Callaby</First>
8          <First>Steve Meier</First>
9        </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>
```


group-starting-with

This function groups the input sequence by the supplied item, if it exists in the source data. A boolean function is needed to test the input data.

Field1	Field2	Field3	Field4	Field5
string	string	string	string	string
HDR	custid0001	ordid	0001	EUR
DTL	itemABC	qty0100	price	0001.100
TXT	please	deliver	ASAP	
DTL	itemxx2	qty0001	price	0010.000
DTL	itemDDD	qty0010	price	0010.500
HDR	custid0002	ordid	0001	EUR
DTL	itemABC	qty0100	price0001.100	
HDR	custid0003	ordid	0002	USD
DTL	itemDEF	qty0003	price	200.000

The function creates groups based on the **first** item of a group, in this example HDR.



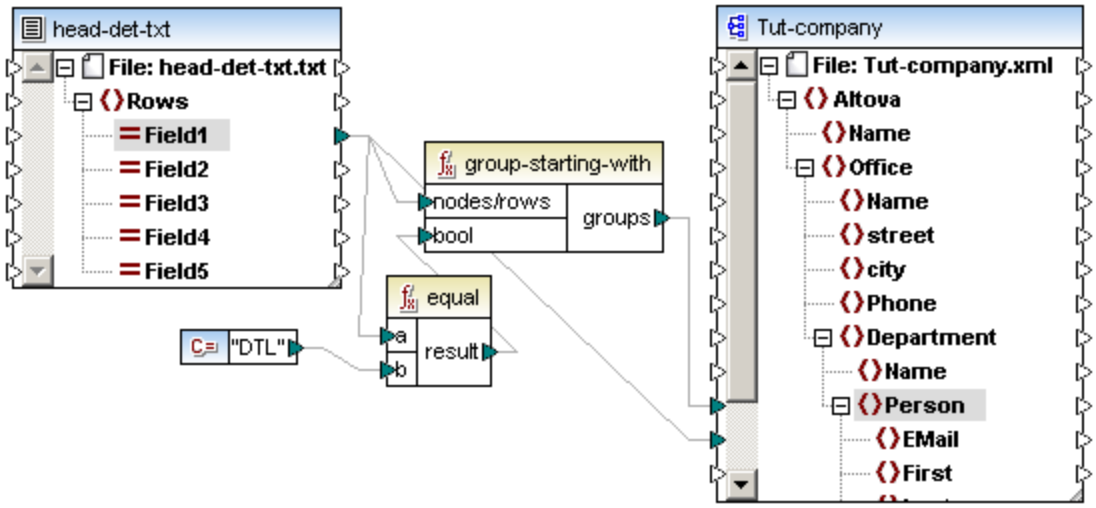
The **value** of the item/nodes do not need to be identical or even exist. The node "**pattern**" i.e. the node/item names need to be identical for the grouping to occur.

```
3   <Office>
4     <Department>
5       <Person>
6         <EMail>HDR</EMail>
7         <EMail>DTL</EMail>
8         <EMail>TXT</EMail>
9         <EMail>DTL</EMail>
10        <EMail>DTL</EMail>
11      </Person>
12      <Person>
13        <EMail>HDR</EMail>
14        <EMail>DTL</EMail>
15      </Person>
16      <Person>
17        <EMail>HDR</EMail>
18        <EMail>DTL</EMail>
19      </Person>
```

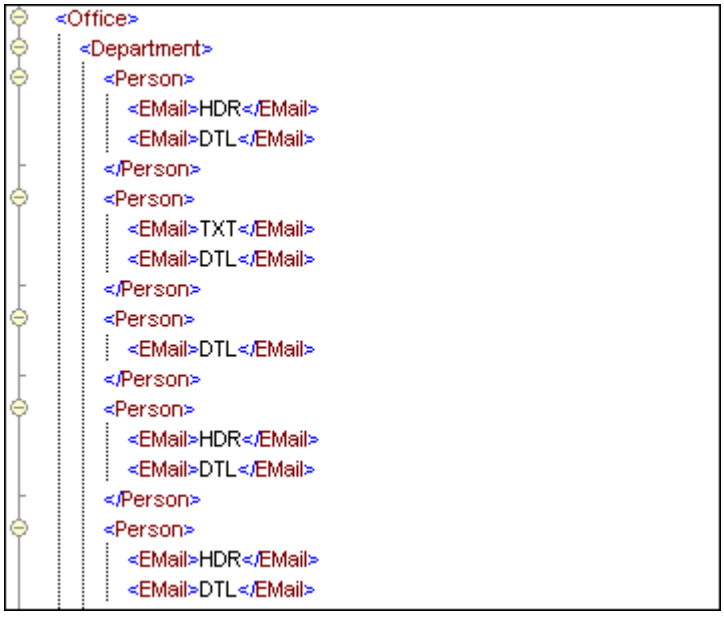
The result above shows that a new group was started for every HDR element.

group-ending-with
This complements the group-starting-with function, and ends each group of the input sequence by the supplied item, if it exists in the source data. A boolean function is needed to test the input data.

Using the same source component as the group-starting-with example above, 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.

set-empty
Allows you to cancel of an XBRL document that were defined higher up the XBRL component/ taxonomy.

11.15 Using DTDs as "schema" components

MapForce 2006 SP2 and above, support namespace-aware DTDs for source and target components. The namespace-URIs are extracted from the DTD "xmlns"-attribute declarations, to make mappings possible.

Adding DTD namespace URIs
There are however some DTDs, e.g. DTDs used by StyleVision, which contain xmlns*-attribute declarations, without namespace-URIs. These DTDs have to be extended to make them useable in MapForce.

- The DTD has to be altered 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'
        ...
>
```

11.16 Type conversion checking

From MapForce **2006 SP2** on, the generated applications and preview (with the Built-in execution engine) check for type-conversion errors in more detail, inline with XSLT2 and XQUERY.

Converting values from one type to another may now result in a runtime-error, where in prior versions of MapForce would have produced some type of result.

Example: conversion of a xs:string 'Hello', to xs:decimal

MapForce **2006** Versions up to, and including **SP1**:

XSLT:	'Hello' (or 'NaN' when passed to a function dealing with number)
XSLT2:	error: "invalid lexical value"
Xquery:	error: "invalid lexical value"
Preview with Built-in execution engine:	0
C++ app:	0
C# app:	error: "values not convertable"
Java app:	error: "values not convertable"

MapForce **2006 SP2**:

XSLT:	'Hello' (or 'NaN' when passed to a function dealing with number)
XSLT2:	error: "invalid lexical value"
Xquery:	error: "invalid lexical value"
Preview with Built-in execution engine:	**error: "string-value 'Hello' could not be converted to decimal"**
C++ app:	**error: "values not convertable"**
C# app:	error: "values not convertable"
Java app:	error: "values not convertable"

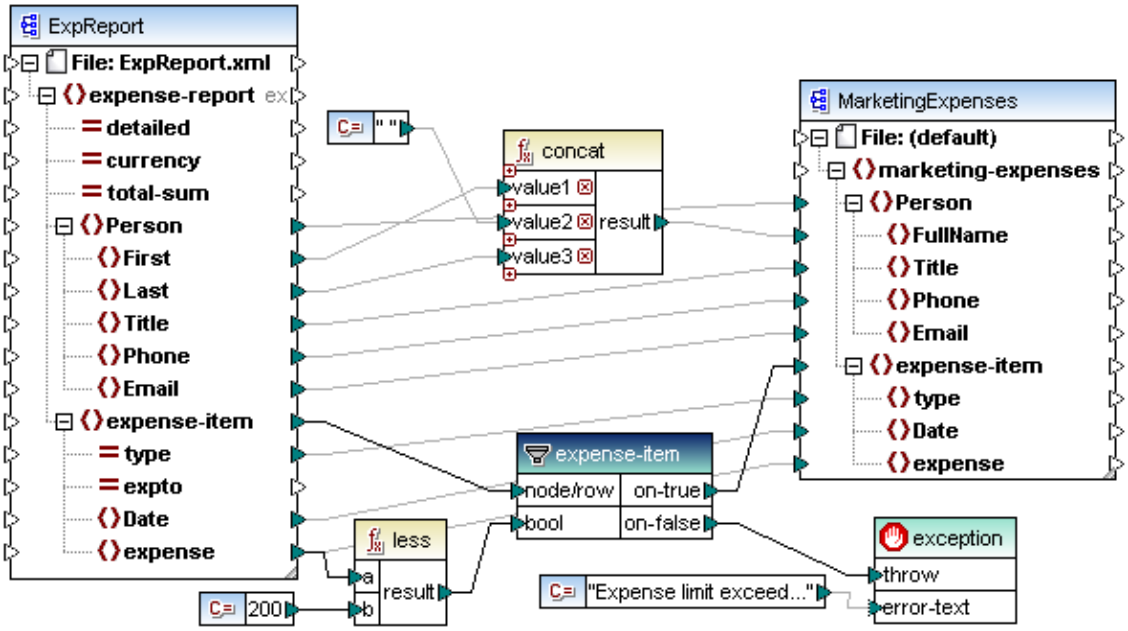
If type-conversion-errors occur, check that the types have been handled correctly. E.g. use the lang:numeric() function, to check if the source-value may be converted into a number, and then an if-else component to pass a different value in case it fails (e.g. a constant containing -1, on the value-false parameter).

11.17 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.

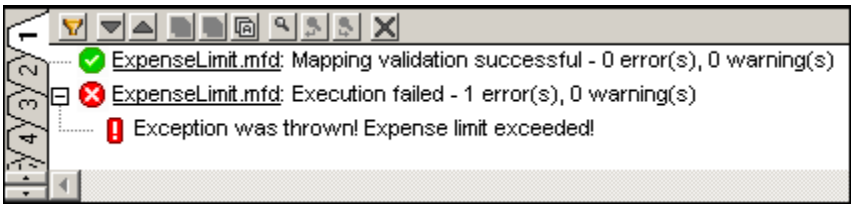
To insert an exception component:

- 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 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 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, the exception appears in the Messages window, and a Preview failed message box appears. Clicking the OK button in the message box switches to the respective XSLT2 or XQuery tab, and the line that triggered the exception is automatically highlighted.

11.18 Catalog files in MapForce

MapForce supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables MapForce to retrieve commonly used schemas (as well as 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.

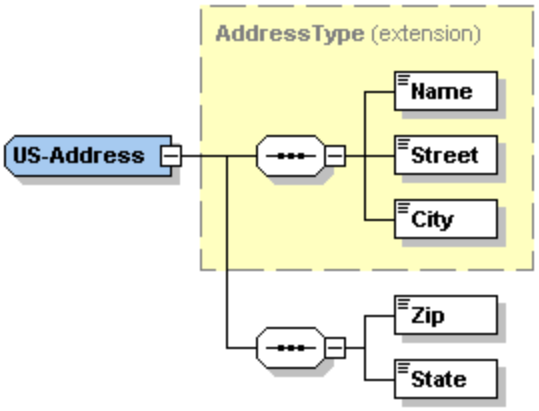
More information

For more information on catalogs, see the XML Catalogs specification.

11.19 Derived XML Schema types - mapping to

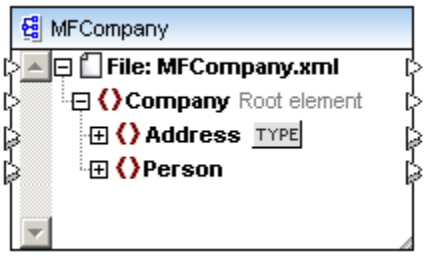
MapForce supports the mapping to/from derived types of a complex type. Derived types are complex types of an XML Schema that use the **xsi:type** attribute to identify the specific derived types.

The screenshot below shows the definition of the derived type "US-Address", in XMLSpy. The base type (or originating complex type) is, in this case, **AddressType**. Two extra elements were added to create the derived type US-Address.

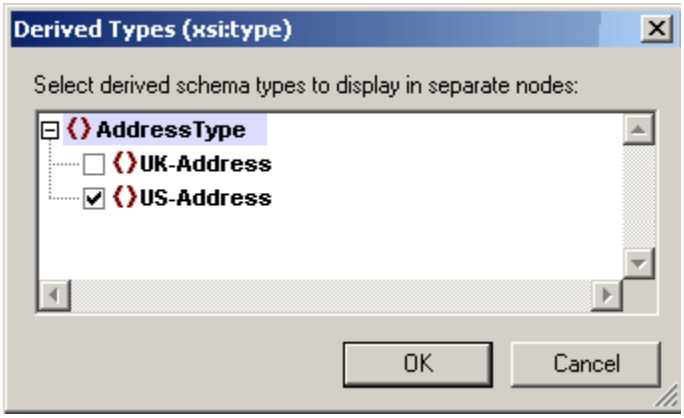


Mapping to derived types in MapForce:

1. Insert the XML schema MFCompany.xsd that is available in the ...\Tutorial folder, click Skip, then select Company as the root element.



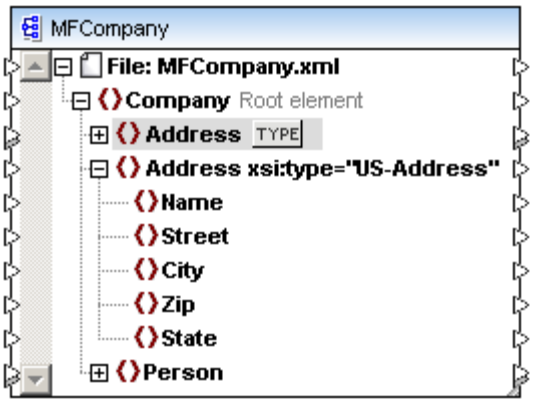
2. Click the TYPE button to the right of the Address element which shows that derived types exist in the Schema component.



3. Click the check box next to the derived type you want to use, e.g. US-Address, and confirm with OK.

A new element **Address xsi:type="US-Address"** has been added to the component.

4. Click the expand button to see the mappable items of the element.



5. You can now map directly to/from these items.

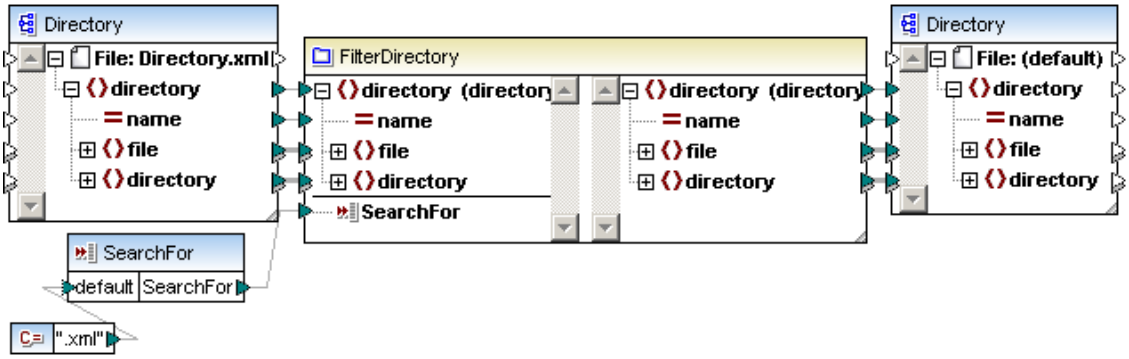
Please note:

You can include/insert multiple derived types by selecting them in the Derived Types dialog box, each will have its own xsi:type element in the component.

11.20 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.

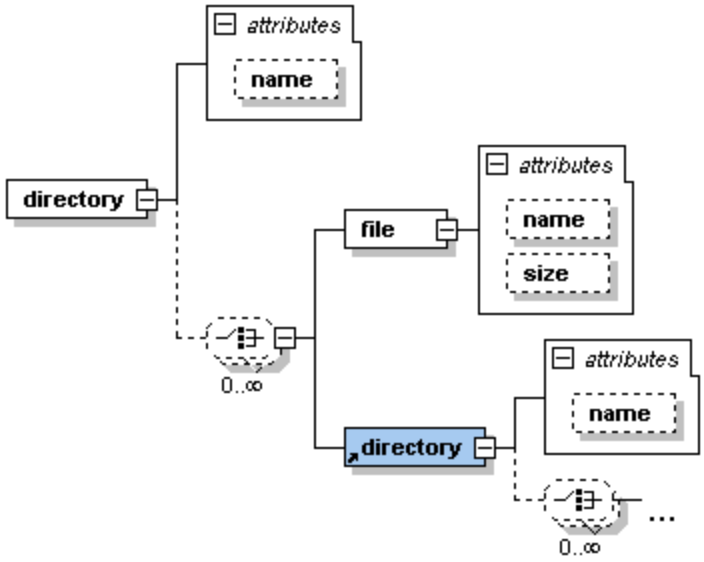
The screenshot below shows the finished mapping containing the recursive user-defined function FilterDirectory, the aim being to filter out a list of the .xml files in the source file.



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>
```

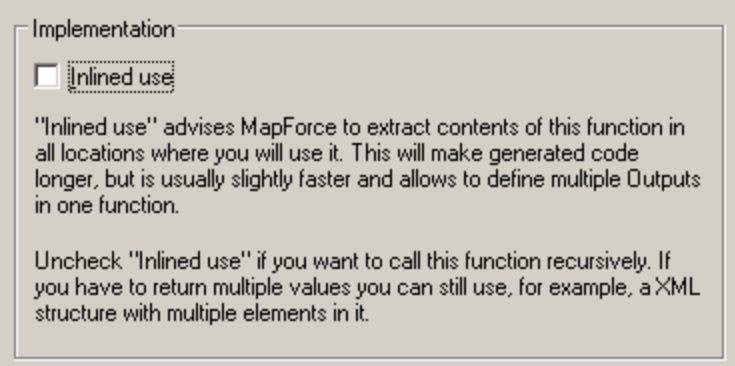
The XML schema file referenced by Directory.xml has a **recursive** element called "directory" which allows for any number of subdirectories and files below the directory element.



11.20.1 Defining a recursive user-defined function

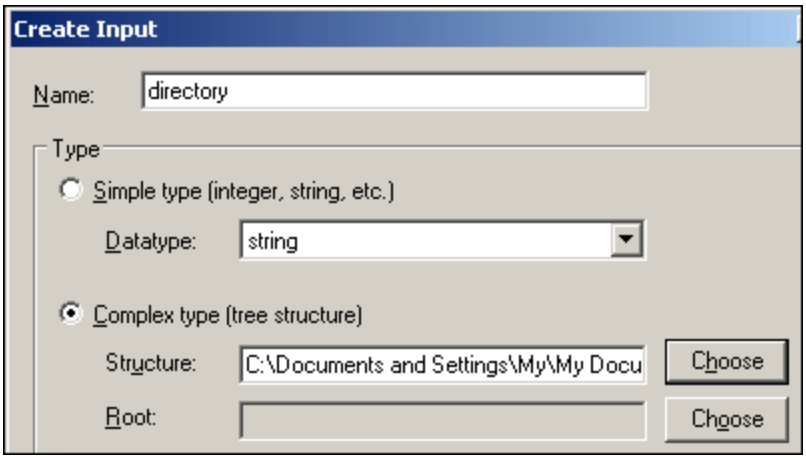
From the **main** mapping window:

1. Select **Function | 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.

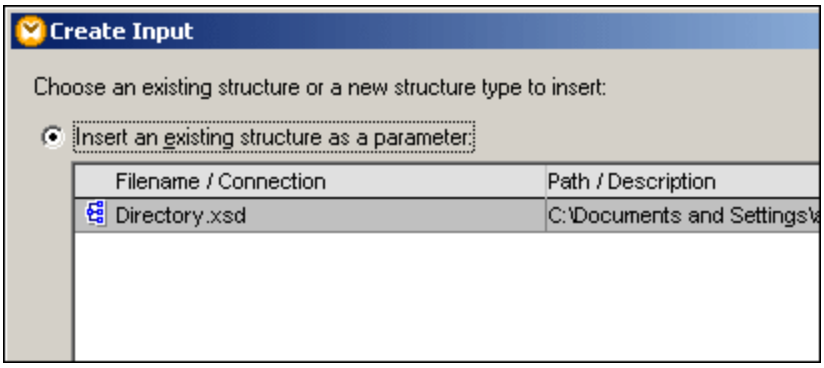


You are now in the **FilterDirectory** window where you create the user-defined function.

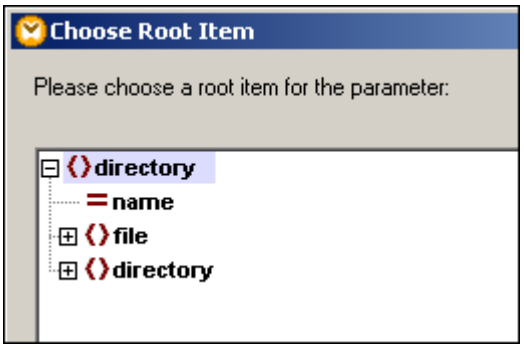
3. Select **Function | Insert Input** to insert an input component.
4. Give the component a name e.g. directory and click on the **Complex Type** (tree structure) radio button.
5. Click the **Choose** button and select the directory.xsd file in the ...\MapForceExamples directory.



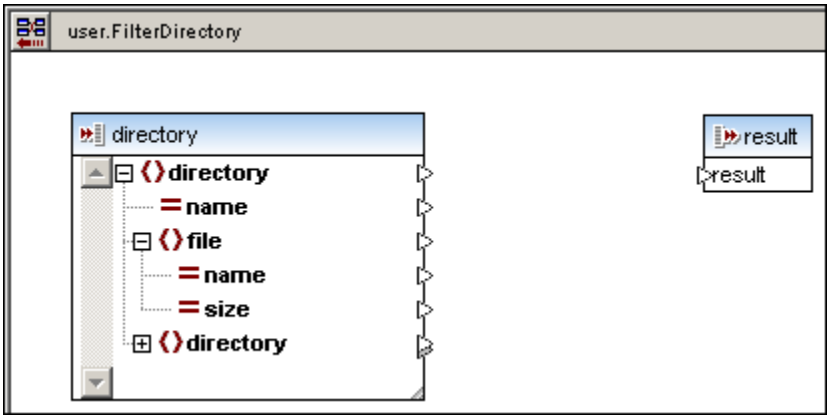
6. Select Directory.xsd from the top pane and click OK to continue.



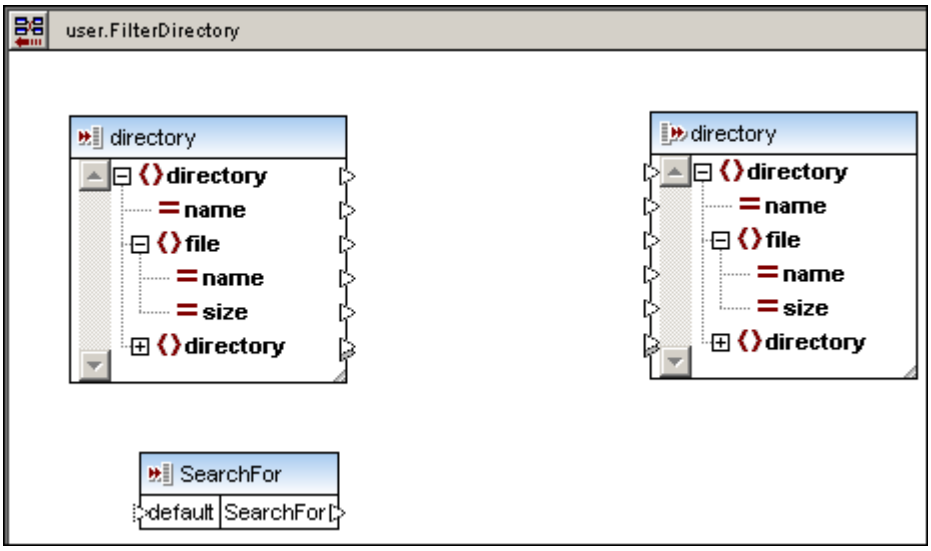
7. Click OK again when asked to select the root item, which should be "directory".



8. Click OK again to insert the complex input parameter.
The user-defined function is shown below.

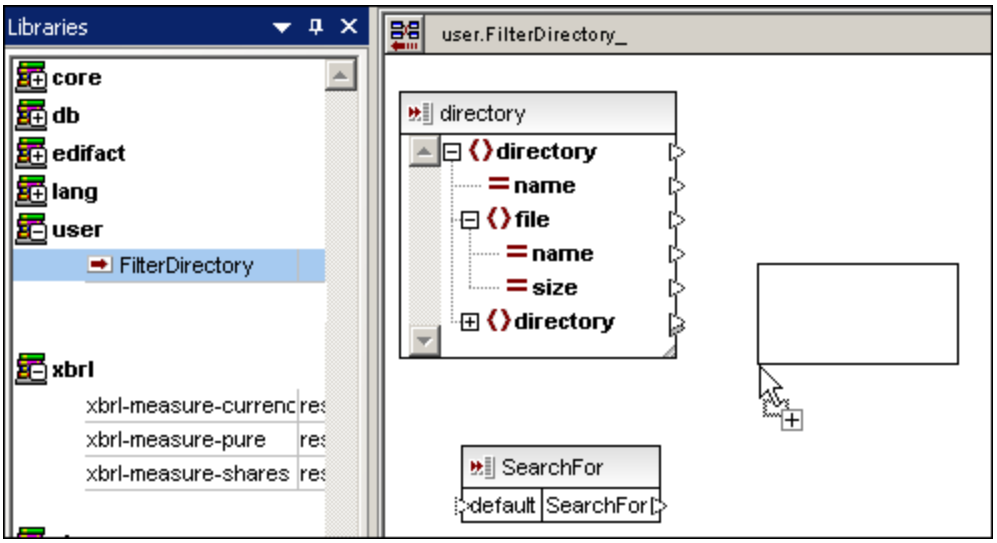


9. Delete the simple result output component, as we need to insert a complex output component here.
10. Select **Function | Insert Output...** to insert an **output** component and use the same method as outlined above, to make the output component, "directory", a complex type. You now have two complex components, one input and the other output.
11. Select **Component | Insert Input...** and insert a component of type Simple type, and enter a name e.g. **SearchFor**.



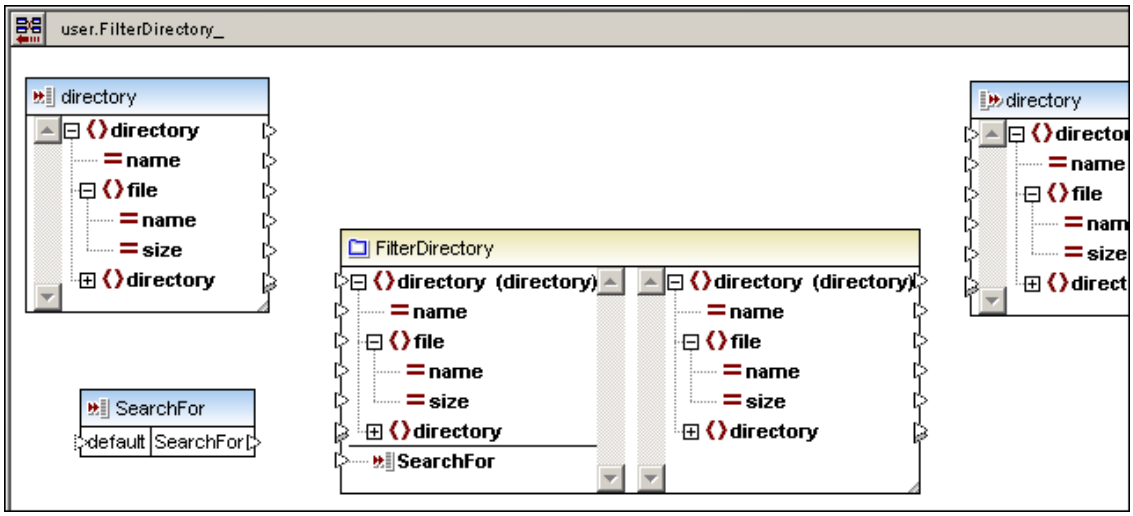
Inserting the recursive user-defined function
At this point, all the necessary input and output components have been defined for the user-defined function. What we need to do now is insert the "unfinished" function into the current user-defined function window. (You could do this at almost any point 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.

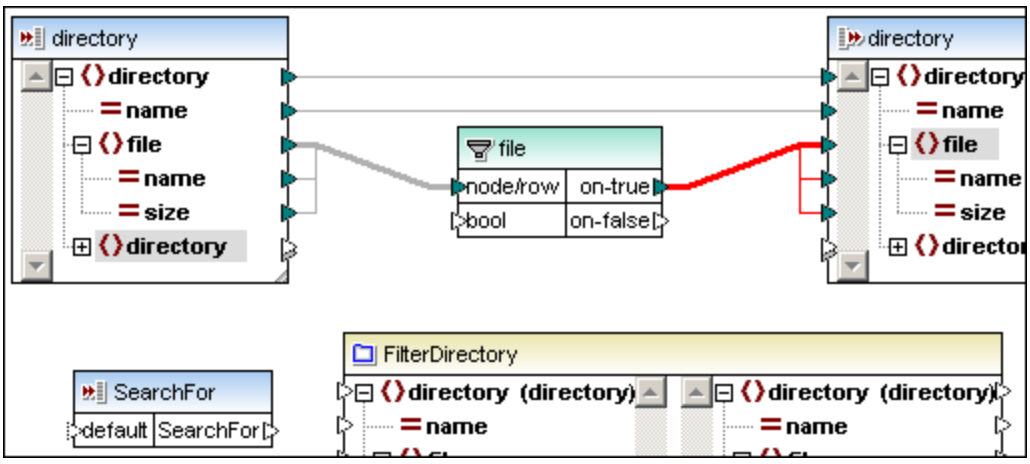


Java Selected

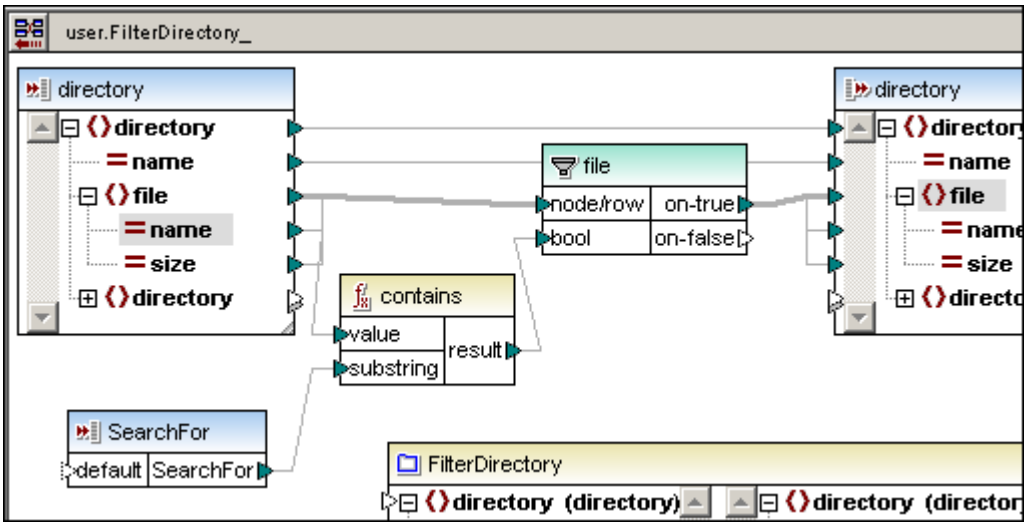
The user-defined function now appears in the user-defined function window as a recursive component.



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.



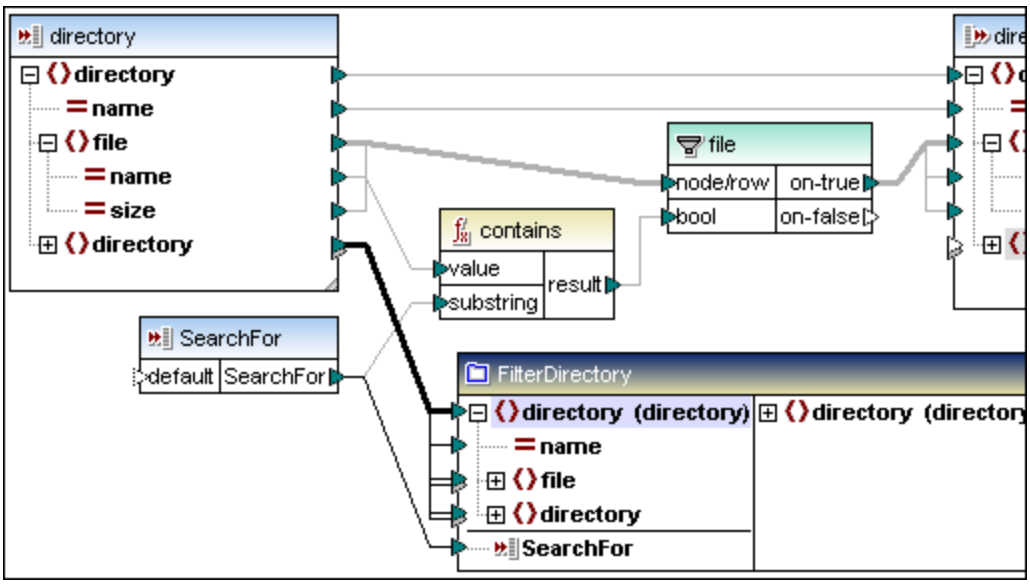
8. Connect the SearchFor item of the input component to the SearchFor item of the user-defined function.

Defining the recursion

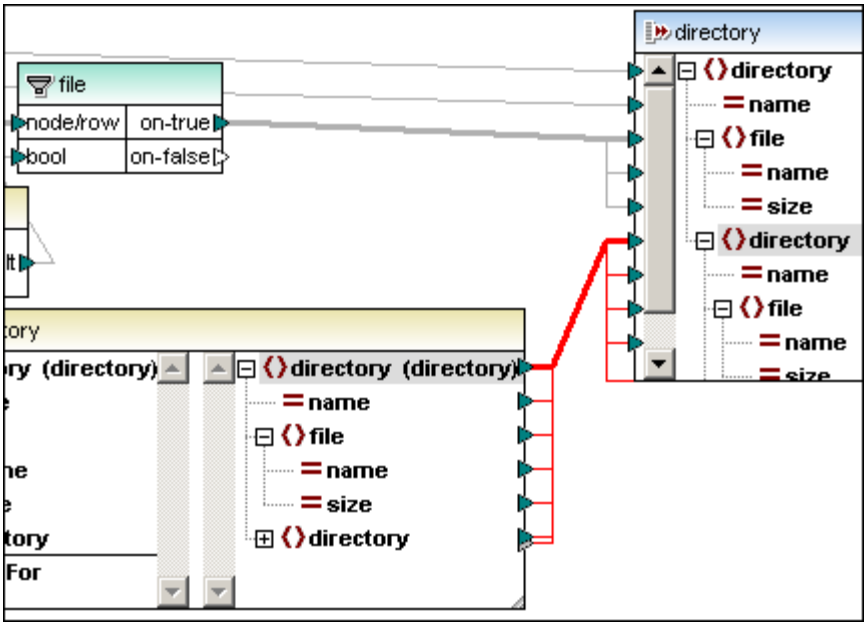
At this point, the mapping of a single directory recursion level is complete. Now we just need to define how to process a subdirectory.

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.



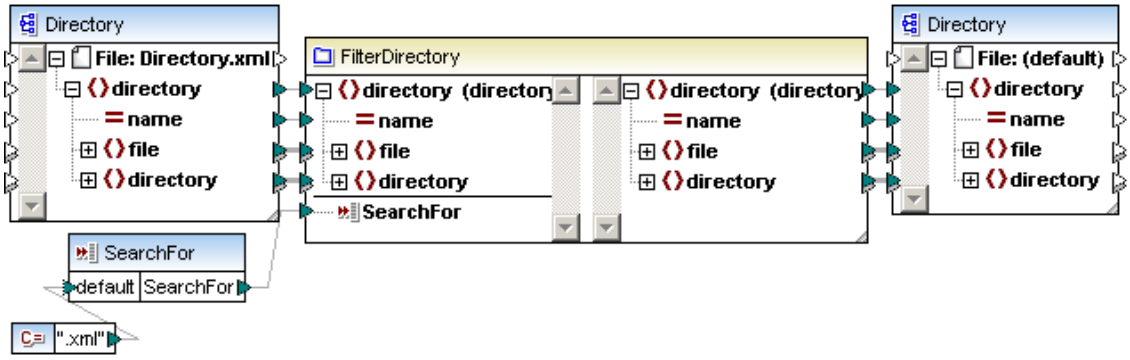
This completes the definition of the user-defined function in this window.

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:
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.

Chapter 12

StyleVision Power Stylesheets in Preview

12 StyleVision Power Stylesheets in Preview

MapForce is now able to preview/render XML Schema 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 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.

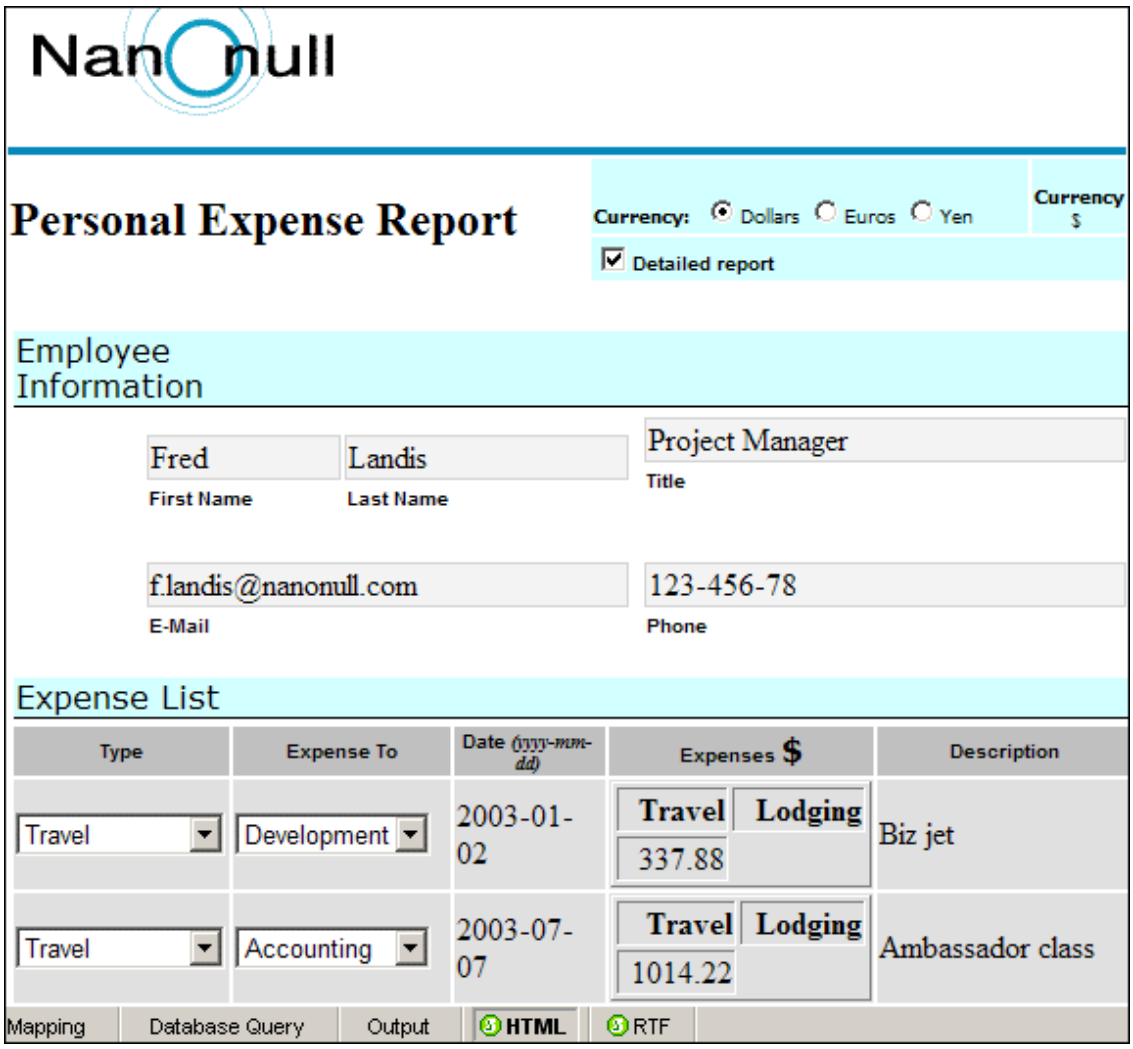
- HTML
- RTF - previews RTF
- PDF - requires FOP version 0.93 or 1.0
- Word 2007+ - previews MS Word documents from version 2007 and later

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.

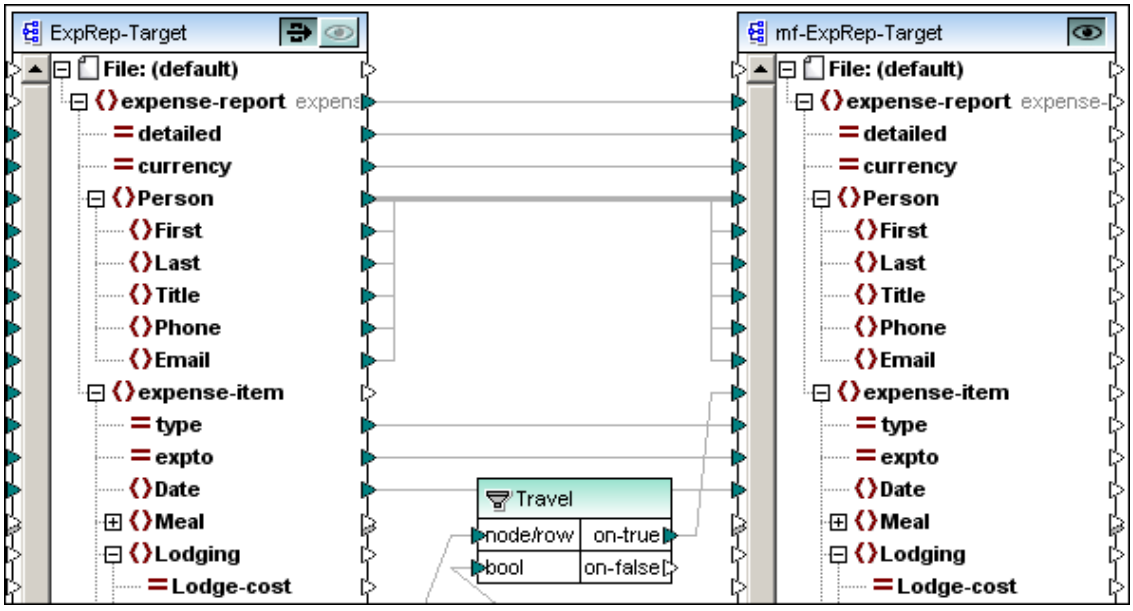
The Professional edition of StyleVision restricts the tabs to HTML and RTF in MapForce.

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.


Nanonull
Personal Expense Report
Currency: Dollars Euros Yen
Currency $
Detailed report
Employee Information
Fred
Landis
First Name
Last Name
Project Manager
Title
f.landis@nanonull.com
E-Mail
123-456-78
Phone
Expense List
Type
Expense To
Date (yyyy-mm-dd)
Expenses $
Description
Travel
Development
2003-01-02
Travel
Lodging
337.88
Biz jet
Travel
Accounting
2003-07-07
Travel
Lodging
1014.22
Ambassador class
Mapping
Database Query
Output
HTML
RTF

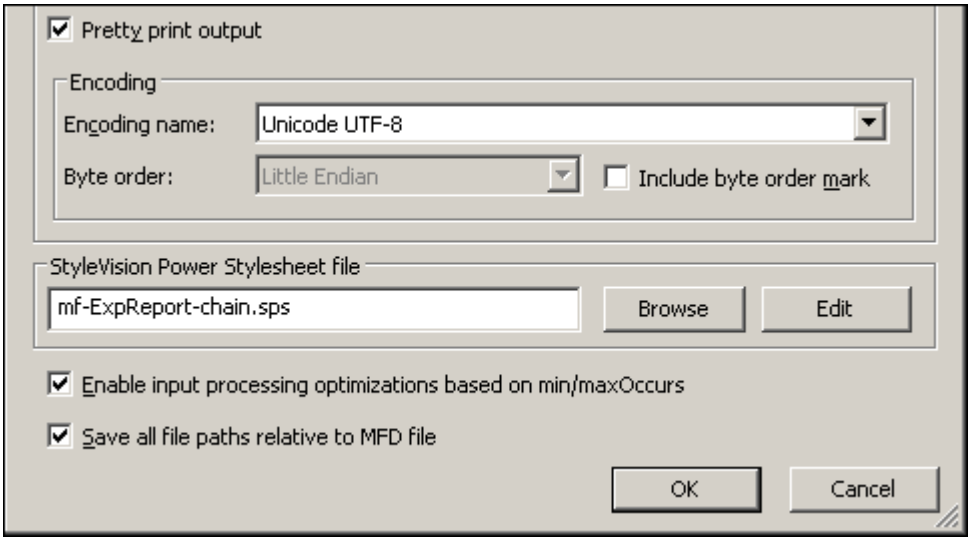
12.1 Assigning an SPS file to a component

The **Tut-ExpReport-chain.mfd** example shown below, is available in the ... \MapForceExamples\Tutorial folder. For more information on chained mapping, as shown in the example below, please see Chained mappings - Pass-through components.

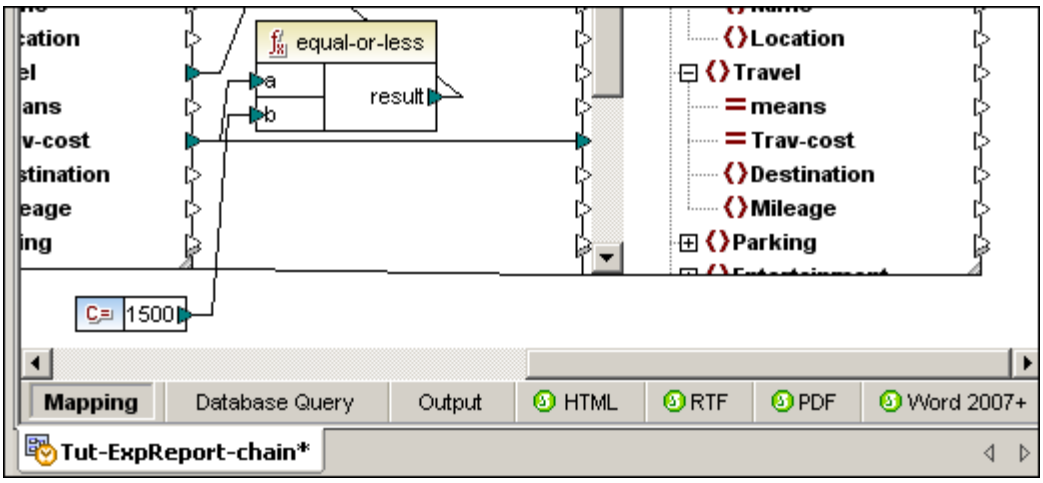


To assign an SPS template to a MapForce component:
Having designed the SPS template in StyleVision, or obtained the template from a third party,

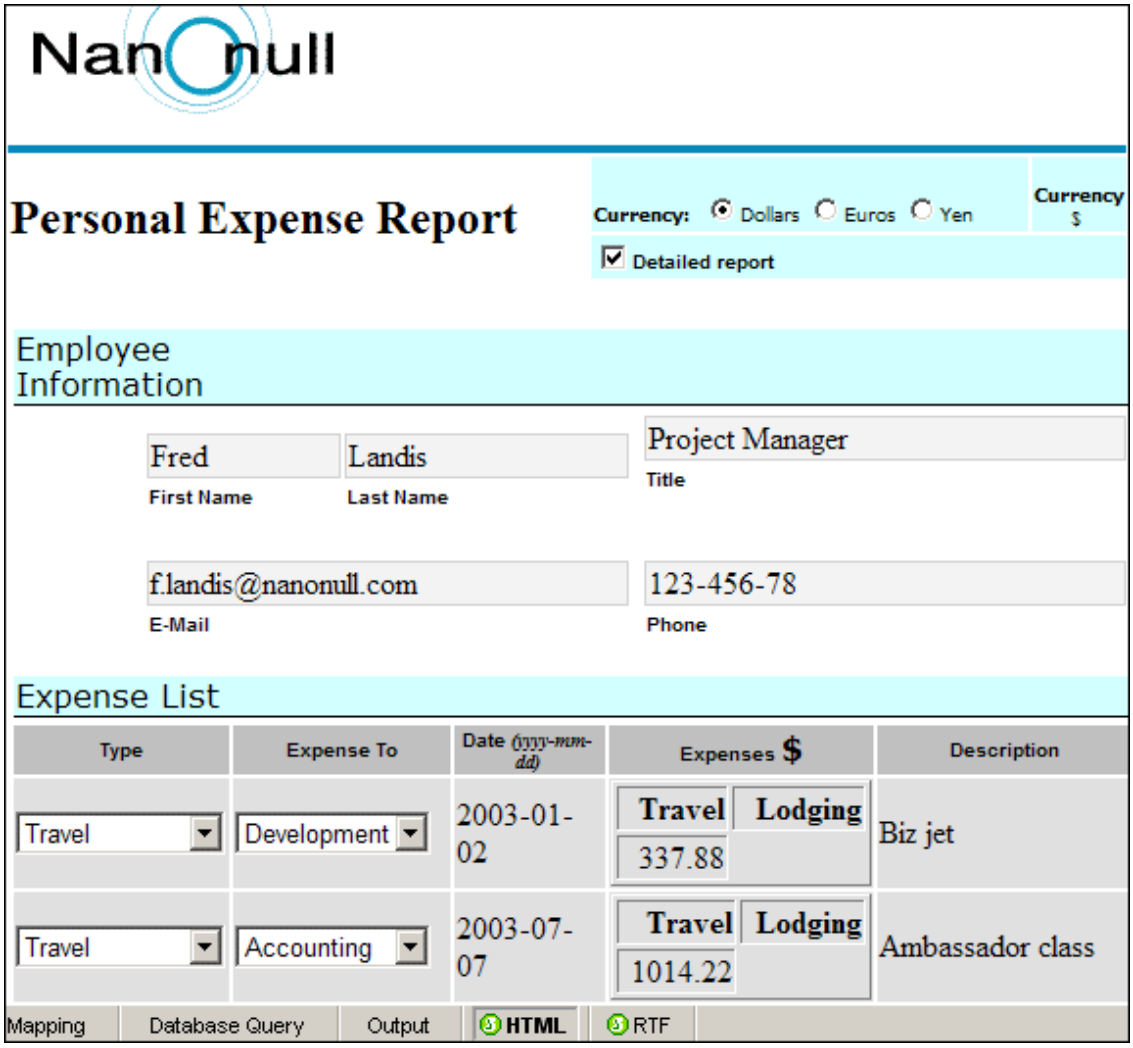
1. Double click the component title bar of the component you want to assign the SPS file to, e.g. mf-ExpRep-Target.



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.

Chapter 13

Documenting mapping projects

13 Documenting mapping projects

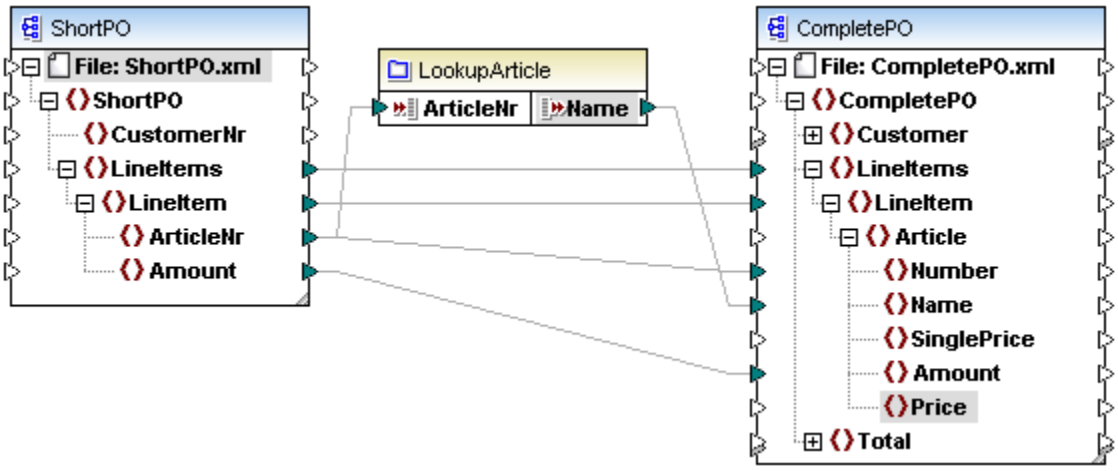
The **Generate Documentation** command generates detailed documentation about your mapping in HTML, MS Word, RTF or PDF. The documentation generated by this command can be freely altered and used; permission from Altova to do so is not required.

Documentation is generated for components you select in the Generate Documentation dialog box. You can either use the fixed design, or use a StyleVision SPS for the design. Using a StyleVision SPS enables you to customize the design of the generated documentation. How to do this is explained in the section, User-Defined Design.

Note: To use an SPS to generate schema 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.

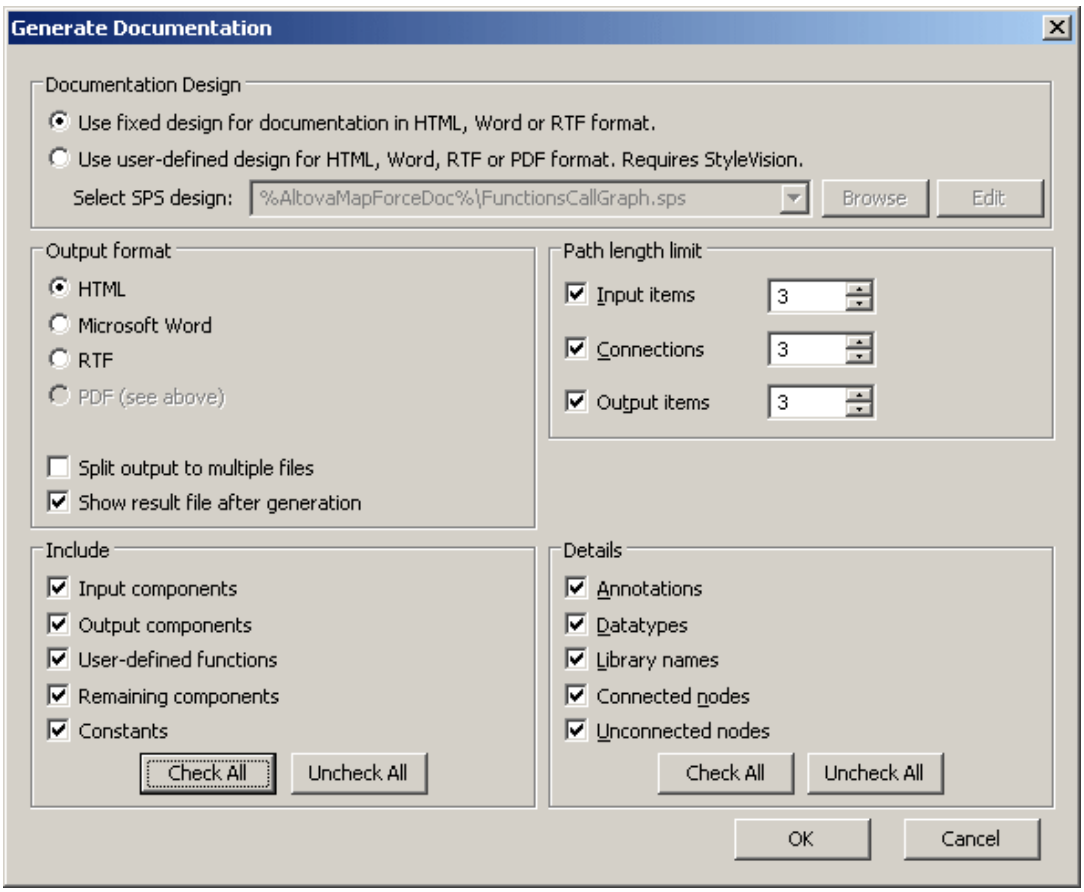
To generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

The screenshot below shows a portion of the **Lookup-standard.mfd** file available in the **...\MapForceExamples** folder.



Having opened a mapping project e.g. Lookup-standard.mfd:

1. 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 \MapForce2011\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 ...\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 Browser View (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)

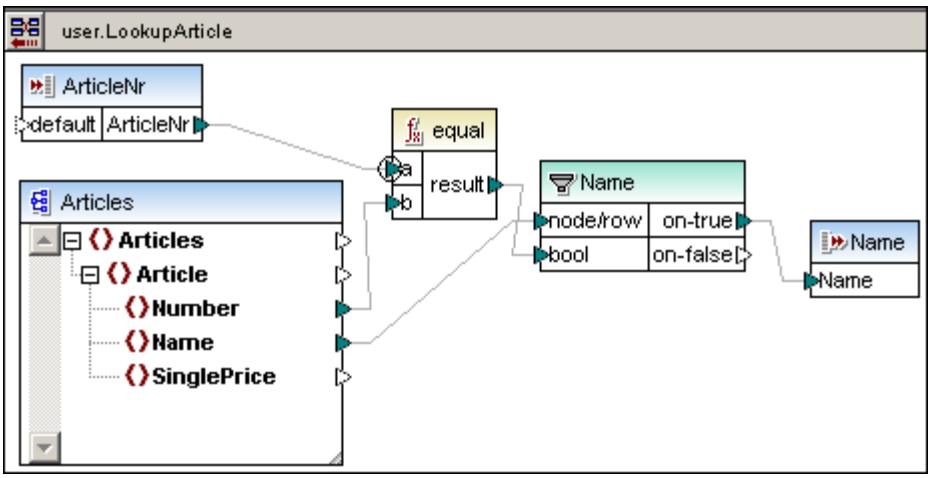
Nodes	Connections	
File: ShortPO.xml Type: string		
ShortPO Type: restriction of xs:anyType [0..1]		
ShortPO/CustomerNr Type: xs:integer		
ShortPO/LineItems Type: restriction of xs:anyType	*direct*	CompletePO/LineItems Type: restriction of xs:anyType
ShortPO/LineItems/LineItem Type: restriction of xs:anyType [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 connectors:
  - 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

13.1 Supplied SPS 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]

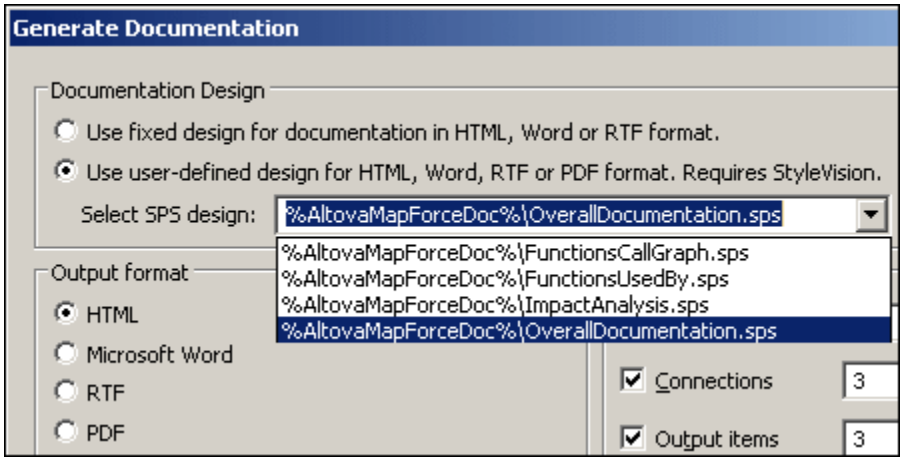
Input **BranchOffices** (BranchOffices.xsd)

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

13.2 User-Defined 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 ...\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\MapForce2011\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 schema 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.

Chapter 14

Nil Values / Nillable

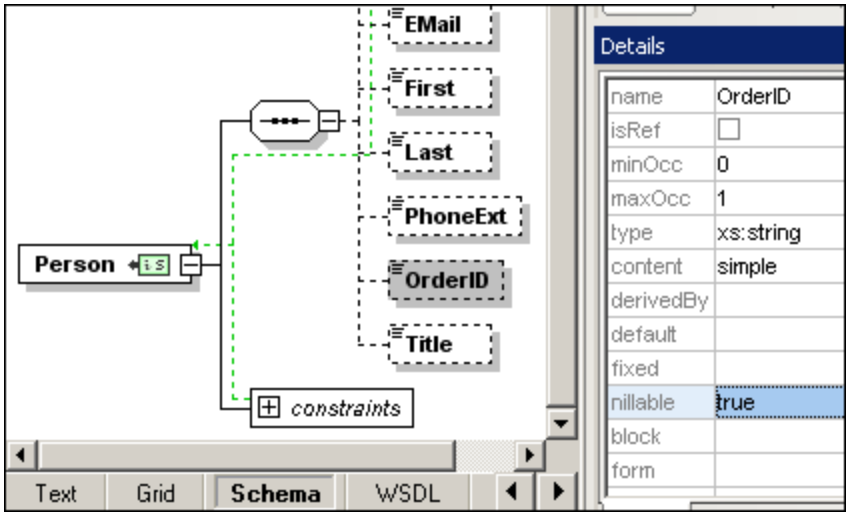
14 Nil Values / Nillable

The XML Schema specification allows for elements to be valid without content (despite a content type which does not allow empty content), if the nillable="true" attribute has been defined for the specific element in the schema.

An **attribute**, xsi:nil="true" in the XML **instance** document, is used to indicate that the element exists but that it has no content.

Note that this only applies to **element** values, and not attribute values. The element with xsi:nil="true" may not have element or textual content, but may still have other attributes.

The **nillable="true"**
attribute is defined in the XML **Schema** and can be present in both the source and target components.



The **xsi:nil="true"**
attribute is defined in the XML **Instance** file and is only present in the instance file.

```
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 attribute is not displayed explicitly in the MapForce graphical mapping, because it is handled automatically in most cases: A "nilled" node (that has the xsi:nil="true" attribute set) exists, but its content does not exist.

Nillable elements as mapping source

Whenever a mapping is reading data **from** nilled XML element, the xsi:nil attribute is automatically checked. If the value of xsi:nil is true, the content will be treated as non-existant.

When creating 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, the xsi:nil attribute is inserted into the target element e.g. <OrderID xsi:nil="true"/>.

When creating a **Copy-All** mapping from a nillable source element to a nillable target element, where xsi:nil is set on a source element, the xsi:nil attribute is inserted into the target element e. g. <OrderID xsi:nil="true"/>.

Using functions

Connecting the "exists" function to a nilled source element will return "true" for all elements, due to the fact that the **element** node actually exists, even if it has no content.

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.

Using functions that expect simple values (e.g. multiply, concat) on elements where xsi:nil has been set, do 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.

To substitute a "nilled" (non-existing) source element value with something specific, use the **substitute-missing** function.

Mapping xsi:nil to a database field

When mapping a "nilled" XML element to a database column, a NULL value is written. You can also use the **set-null** function if you unconditionally want to set a database field to null.

Nillable elements as mapping target

When creating 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, the xsi:nil attribute is inserted into the target element e.g. <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. Define a **Copy-All connection** to copy the xsi:nil attribute from the source element.

When mapping an **empty sequence** or a database NULL value to a target element, the element will not be created at all - independent of its nillable designation.

Using functions

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.

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 (e.g. multiply) involved a nilled source node and therefore yielded no result.

This function can also be used to create elements with xsi:nil="true" from database NULL values. This function works only for nodes with simple content.

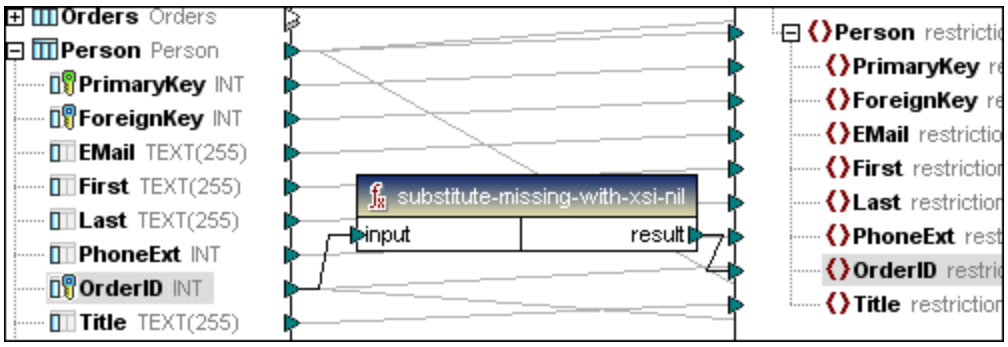
To create xsi:nil on an element of **complex type** depending on some condition, do the following:

- Create the mapping as usual for the case where the target node contains content.
- Insert a filter component into the connection to the target node, and define the condition when there is any content.
- Use the "duplicate input" function to duplicate your target node.
- Connect the **set-xsi-nil** function via another filter component, to the duplicated target node.
- Insert a logical-not function to negate the condition, and connect it to the filter between set-xsi-nil and the duplicated target node.

Mapping Null fields from database components
If a null database field is mapped to an nillable element of an XML schema, then only those target elements are generated, 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.

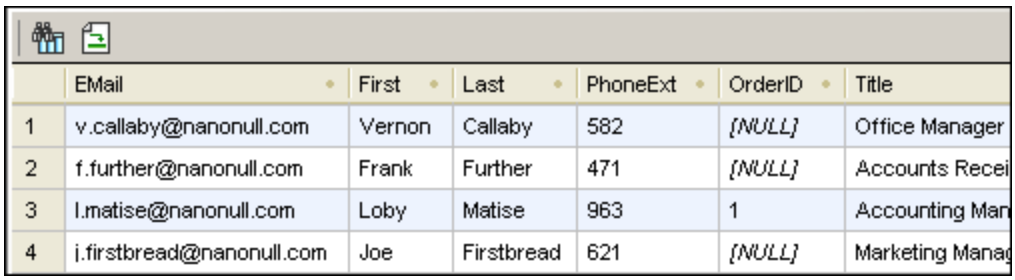
To force the creation of all elements in the target component:
Use the **substitute-missing-with-xsi-nil** function from the node functions of the core library.



Target elements are now created for all database fields.

- All (missing)/Null database fields now 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 find/see 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

Chapter 15

Databases and MapForce

15 Databases and MapForce

Altova web site: Mapping Database data

MapForce 2011 provides powerful support for database mapping, including mapping between database data and XML, as well as flat files, and even other database formats.

MapForce takes primary and foreign key constraints into account and also generates transactions which ensures data integrity.

Please note:
Database access requires that you use one of the following: Java, C#, C++, or BUILTIN (the Built-in execution engine). **XQuery** code can only be generated for XML data sources and targets.

Currently supported databases (and connection types) are:

- Microsoft Access versions 2003 / 2007
- Microsoft SQL Server versions 2000, 2005 and 2008
- Oracle version version 9i, 10g, and 11g
- MySQL 4.x and 5.x
- PostgreSQL 8.0, 8.1, 8.2, 8.3
- Sybase 12
- IBM DB2 version 8.x and 9
- IBM DB2 for i 5.4
- IBM DB2 for i 6.1

Please note:
MapForce fully supports the databases listed above. While Altova endeavors to support other ODBC/ADO databases, successful connection and data processing have only been tested with the listed databases.

When installing the **64-bit** version of **MapForce**, please make sure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

Note:
For an ADO connection to MS SQL Server via the driver SQL Server Native Client 10.0 the following property values must be set in the *All* tab of the Data Link Properties dialog: (i) Set the property value of Integrated Security to a space character; (ii) Set the property value of Persist Security Info to `true`.

15.1 JDBC driver setup

JDBC drivers have to be installed for you to compile Java code when mapping database data. They are **not** needed for using the Built-in execution engine, or if you generate code for C++ or C#.

Overview

This section describes how to download and install JDBC drivers and how to use them with **Ant** and **JBuilder**. A **JBuilder** project file and **Ant** build scripts are generated by MapForce when generating Java code.

JDBC drivers are used by MapForce generated Java applications to connect to, and exchange data with several different databases. These JDBC drivers need to be installed first, to successfully run the generated Java application(s).

In general JDBC drivers can be found at http://industry.java.sun.com/products/jdbc/drivers

MapForce generated Java applications were tested with the following JDBC-drivers:

- MS Access
- MSSQL2000
- Oracle 9i
- MySQL
- PostgreSQL
- Sybase
- IBM DB2

This section assumes the following:

- the reader is familiar with setting Java CLASSPATHs
- Java SDK and Ant, or JBuilder is already installed and is working correctly
- at least one of the databases described below is running and the minimum privilege - read-only, is granted

---

MS Access

The JDBC-ODBC-bridge is already installed with Java SDK.

Java internal usage

Driver	sun.jdbc.odbc.JdbcOdbcDriver
URL	jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);DBQ=Sourcename...

Microsoft SQL Server 2000

Download from http://www.microsoft.com/sql/

Ant Settings

Please make sure that the following jar file entries are in the CLASSPATH:

```
C:\Program Files\Microsoft SQL Server 2000 Driver for JDBC\lib\
msbase.jar;C:\Program Files\Microsoft SQL Server 2000 Driver for
JDBC\lib\mssqlserver.jar;C:\Program Files\Microsoft SQL Server 2000
Driver for JDBC\lib\msutil.jar
```

assuming that "`C:\Program Files\Microsoft SQL Server 2000 Driver for JDBC`" was your installation folder.

JBuilder Settings

Use the menu option **Tools | Configure JDKs...** then click **Add** to add all the jar files listed above.

Java internal usage

Driver	com.microsoft.jdbc.sqlserver.SQLServerDriver
URL	jdbc:microsoft:sqlserver://localhost

Microsoft SQL Server 2005 JDBC driver 1.1

Download the Microsoft SQL Server 2005 JDBC Driver from the Microsoft website.
http://www.microsoft.com/downloads/details.aspx?FamilyID=6d483869-816a-44cb-9787-a866235efc7c&DisplayLang=en

Assuming you extract the downloaded zip archive to **C:**\, please make sure that the following jar file entry is in the CLASSPATH:

C:\Microsoft SQL Server 2005 JDBC Driver\sqljdbc_1.1\enu\sqljdbc.jar

JDBC specific settings in Mapforce:
NOTE: Mapforce will automatically add the JDBC Driver and Databse URL settings that would be used for the MS SQL 2000 JDBC connection even if you connect to a MS SQL 2005 server, so these must be changed before generating Java code!

Using 2000 JDBC:
JDBC Driver: com.microsoft.jdbc.**sqlserver**.SQLServerDriver
Database URL: jdbc:microsoft:**sqlserver**://localhost

Using 2005 JDBC:
JDBC Driver: com.microsoft.**sqlserver**.jdbc.SQLServerDriver
Database URL: jdbc:**sqlserver**
://PRIVSQL05;DatabaseName=mydb_13031;SelectMethod=Cursor;

If you use the integrated windows authentication method to connect to your SQL 2005 server, you have to add **integratedSecurity=true** to your Database URL.

e.g.
jdbc:sqlserver://PRIVSQL05;DatabaseName=mydb_13031; integratedSecurity=true; SelectMethod=Cursor;

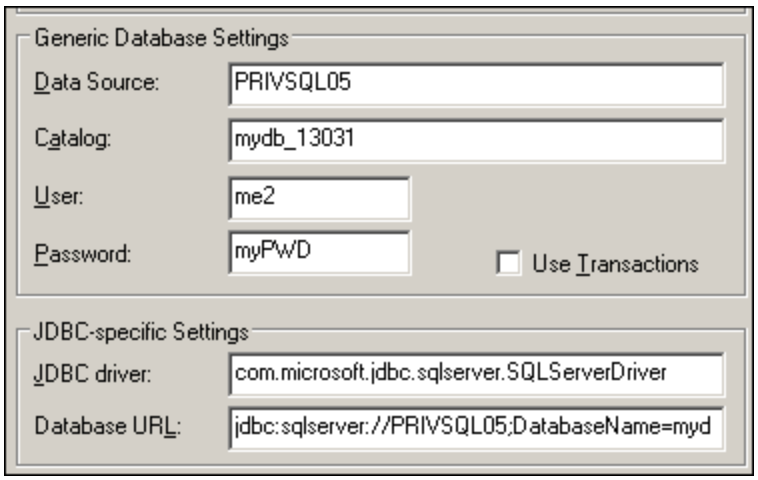
If you are using the integrated Windows Authentication to connect to the MS SQL 2005 database it will also be necessary to use **sqljdbc_auth.dll** depending on the type of machine, 32 bit or 64 bit, and then place this DLL into your Windows System path i.e. Windows/System32.

sqljdbc_auth.dll can be found at the locations shown below if you extracted the ZIP file to C:\.
C:\Microsoft SQL Server 2005 JDBC Driver\sqljdbc_1.1\enu\auth\x86 x86 version.

or,

C:\Microsoft SQL Server 2005 JDBC Driver\sqljdbc_1.1\enu\auth\x64 x 64 bit version.

If you do not use the integrated Windows Authentication, i.e. you are connecting to the database using a database username and password, then you can enter the user name and password into the Generic Database Settings dialog (in this case you would leave integratedSecurity=true out of the Database URL). Double click the database component to open this dialog box.



Oracle 9i, 10, 11

Download the Oracle9i Release 2 (9.2.0.3) driver for JDK 1.4: **ojdbc14.jar**
from http://otn.oracle.com/software/tech/java/sqlj_jdbc
You will need to have an account, or sign up to the Oracle Technology Network to access these drivers.

Ant Settings
Add the full path to ojdbc14.jar to the CLASSPATH.

JBuilder Settings
Use the menu option **Tools | Configure JDKs...** then click **Add** to add the jar file above.

Java internal usage

Driver	oracle.jdbc.OracleDriver
URL	jdbc:oracle:oci:@localhost

MySQL 4, 5

Download the Connector/J JDBC driver from http://www.mysql.fr/downloads/connector/j/

MySQL 4 and 5 use the Connector/J (C:\Program Files\MySQL\Connector JDBC 5.1\mysql-connector-java-5.1.13-bin.jar)

PostgreSQL

Download the JDBC driver from http://jdbc.postgresql.org/download.html

The PostgreSQL JDBC driver allows Java programs to connect to a PostgreSQL database using standard, database independent Java code. It is a pure Java (Type IV) implementation, so all you need to do is download a jar file and you're on your way.

The driver provides are reasonably complete implementation of the JDBC 3 specification in addition to some PostgreSQL specific extensions. Licence Open Source.

Download the ODBC driver from http://www.postgresql.org/ftp/odbc/versions/

Sybase 12

Download the Sybase jConnect drive for JDBC from
http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect

---

IBM DB2

Download the JDBC driver from:
http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.swg.im.dbclient.install.doc/doc/t0007315.html

The IBM DB2 native Jdbc driver is gnerally located in C:\Program Files\IBM\SQLLIB_01\java.

15.2 Installing Database clients / drivers

This document is designed to aid end users when installing the various database clients and is not intended as a definitive resource, but as a recommendation, and is supplied without warranty.

We would however, advise you to have a database administrator install any client software and avoid a manual installation by an end user.

You can find instructions for the installation of each specific of database client/driver in the following sections.

Precondition:
The specific database is already installed in your environment, and you have access to the database server via LAN.

15.2.1 SQL Server

Microsoft SQL Server databases:

SQL Server 2000

- Ask your database administrator for the Microsoft SQL Server DB client and install it.
- Launch the Enterprise Manager, right click on "SQL SERVER Group", choose "New SQL Server Registration" and add a server.
- Ask your database administrator for login details i.e. user name and password, to perform a connection.
- Choose "SQL Server authentication" enter the user name and password, then add the SQL Server to an existing SQL Server Group.
- Expand the left tree and choose a database: Microsoft SQL Server->SQL Server Group->{Server name}->Databases->{Database name}

SQL Server 2005

- Ask your database administrator for the Microsoft SQL Server DB client and install it
- Select Client Components and Document Components in the Dialog "Components to install"
- Do not select any component and click "Advanced"
- Select "Client Components" (entire contents will be installed)
- Select "Documentation, Samples, and..." (entire contents will be installed)
- Click "Next" and "Install".

SQL Server 2008

- Ask your database administrator for the Microsoft SQL Server DB client and install it.

15.2.2 Oracle

Oracle databases:

Oracle 9i

- Ask your database administrator for the Oracle client administrator package and install it.
- Copy the two files **sqlnet.ora** and **tnsnames.ora** from the package to ...\ORACLE \ora92\network\ADMIN.
- Ask your database administrator for login details i.e. user name and password, needed to connect to the database.

You can find downloads at http://www.oracle.com/technology/software/products/oracle9i/index.html.

Oracle 10g

- Ask your database administrator for the Oracle client administrator package and install it.
- During installation the Oracle Net Configuration Assistant will start.
- Click “Next” in each of the dialog boxes, without changing any of the default settings.
- One dialog box will prompt you for the **service name,** and another will prompt you for the **server name**. Type in the correct service and server names.
- Use standard port number 1521, or ask your administrator for the correct port number.
- Ask your database administrator for the login details i.e. user name and password, needed to connect to the database.

You can find downloads at http://www.oracle.com/technology/software/products/database/index.html.

Oracle 11g

- Ask your database administrator for the Oracle client administrator package and install it.
- Follow the installation instructions.
- Ask your database administrator for the login details i.e. user name and password, needed to connect to the database.

You can find downloads at http://www.oracle.com/technology/software/products/database/index.html.

15.2.3 IBM DB2

IBM DB2 databases:

DB2 Version 8 / 9

Precondition: JRE (Java Runtime Environment) 1.4.2 or later must be installed.

- Ask your database administrator for the IBM DB2 client installation package and install it. Follow the installation instructions.
- When the installation is finished, start the "Configuration Assistant" to configure your database.
  Start IBMDB2->setup tools->Configuration Assistant.
  A message is displayed: "... Would you like to add a database now?".
  Click "Yes", the "Add Database Wizard" will start.
  Choose "Search the network", "Next", "Add system", "Discover" and select the system name.
- Add a database:
  Start IBMDB2->General administration Tools->Control Center.
  Expand the tree: Control center->All Cataloged Systems->{Servername}->Instances->{instance_name}->Databases.
  Right click on Databases and choose the menu item "Add".
  Click on "Discover", select a database and click "OK".
- Register the database as a data source:
  Start the "Configuration Assistant" and select a database.
  Choose "data Source" in the left list.
  Choose "Register this database for ODBC" and click check box "As user data source".
  Click "Finish".
- Test the connection to the database:
  Start the "Configuration Assistant" and select a database.
  Choose the menu item "Selected->Test connection".
  Select as connection type CLI, ODBC.
  Ask your database administrator for user name and password and click "Test connection"
- Please make sure you check (activate) the "Remember Password" check box, or the connection settings to the database will not be retained.

Please see http://www-01.ibm.com/software/de/data/db2ims/db2ud.html. for more information.

DB2 for i 5.4

Ask your database administrator for instructions on how to access the database server. Note that iSeries is the new name for the AS400 range of computers.

15.2.4 MySQL

MySQL databases

MySQL 4 / 5

Compared to other databases, MySQL offers a separate driver installation. You do not need to install a client.

- Installing the ODBC driver
  - Install the latest ODBC driver (version 5.1.xx at the time of writing)
  - Start the Control Panel and select "Administrative tools"->"Data Source (ODBC)"
  - Select "User DNS" and click "Add"
  - Select the latest "MySQL ODBC Driver" (version 5.1.xx at the time of writing) and click "Finish"
  - Fill in the "Data Source Name", "Server", "User", "Password" and "Database" fields
  - Click "Test" and - and if successful - "OK"

You can find downloads at http://dev.mysql.com/downloads/connector/odbc/

15.2.5 PostgreSQL

PostgreSQL databases

PostgreSQL 8.x databases

PostgreSQL 8.x

Compared to other databases, PostgreSQL offers a separate driver installation. You do not need to install a client.

- Install ODBC driver: Run **psqlodbc.msi** and follow instructions

You can find downloads at http://www.postgresql.org/ftp/odbc/versions/msi

15.2.6 Sybase

Sybase databases:

Sybase 12

Sybase 12

Ask your database administrator for the Sybase client package and install it.

- Install the Sybase client and follow the installation instructions
- You might need to edit the file c:\sybase\ini**sql.ini**
- Start the Control Panel and select "Administrative tools"->"Data Source (ODBC)"
  - Select "User DNS" and click "Add"
  - Choose the driver "Sybase ASE ODBC Driver" and click "Finish"
  - Enter the name of the DSN file into the "Data Source Name" field
  - Add "Winsock" to "Network Library Name"
  - Add "<server name>,2048" to "Network Address"
  - Add "<database name> to the "Database Name" field
  - Click "Test" and - if successful - "OK"

Please see
http://www.sybase.com/products/databasemanagement/adaptiveserverenterprise for more information.

The sql.ini file entries are generally in the form,

```
[<servername>]
master=TCP, <servername>,<portnumber>
query=TCP, <servername>,< portnumber>
```

15.2.7 Microsoft Access

Microsoft Access 2004 and 2007

There is no need to install any clients or drivers to use these databases, all you need is direct access over a LAN.

15.2.8 Firebird

Firebird databases:

Firebird 2.0.5 / 2.1.2

Firebird 2.0.5 / 2.1.2

The Firebird ODBC driver requires the installation of a client DLL, therefore install the Firebird client and then the driver.

- Run Windows executable installer for Full Classic.
  You can select only the client installation.
- Run Windows **Full Install.exe** to install the driver.
- Start the Control Panel and select "Administrative tools"->"Data Source (ODBC)"
- Select "User DNS" and click "Add"
- Select "Firebird Driver" and click "Finish"
- Fill in the "Data Source Name", "Database", "Database Account" and "Password" fields
- Select **fbClient.dll** from the client installation into "Client"
- Click "OK"

Client download for version 2.0.5
http://www.firebirdsql.org/index.php?op=files&id=engine_205

Client download for version 2.1.2
http://www.firebirdsql.org/index.php?op=files&id=engine_212

ODBC driver download for both versions
http://www.firebirdsql.org/index.php?op=files&id=odbc

15.3 Mapping XML data to databases

Files used in this section:

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** 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.

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.

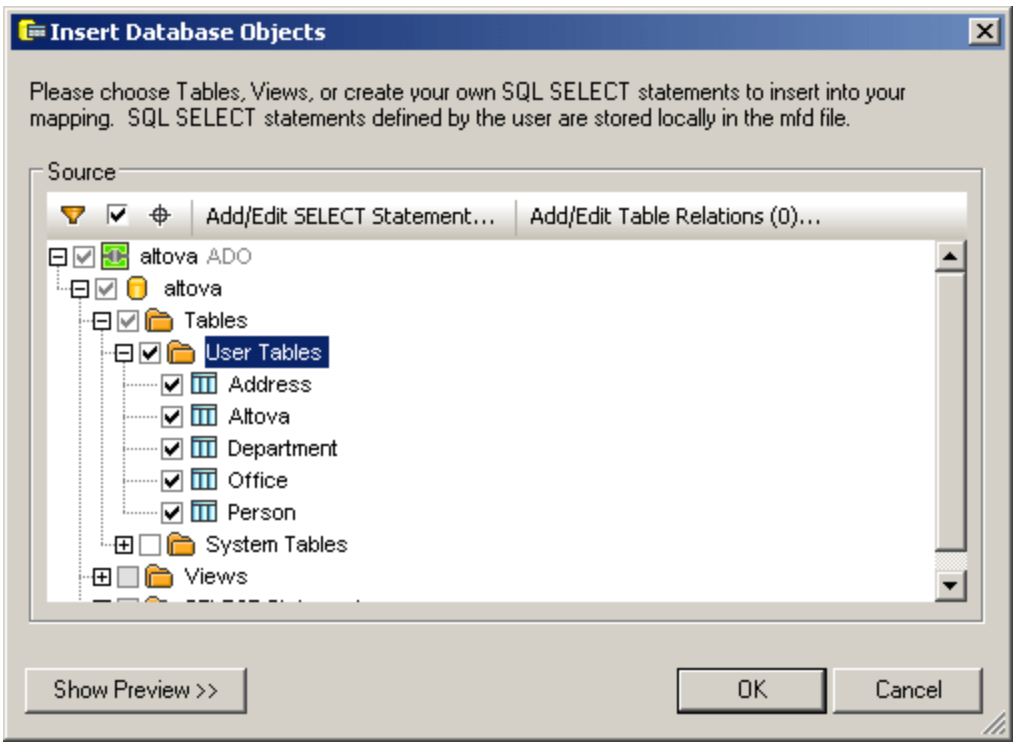
15.3.1 Setup of XML to database mapping

Setting up an XML to database mapping, is in no way different from the methods previously described.

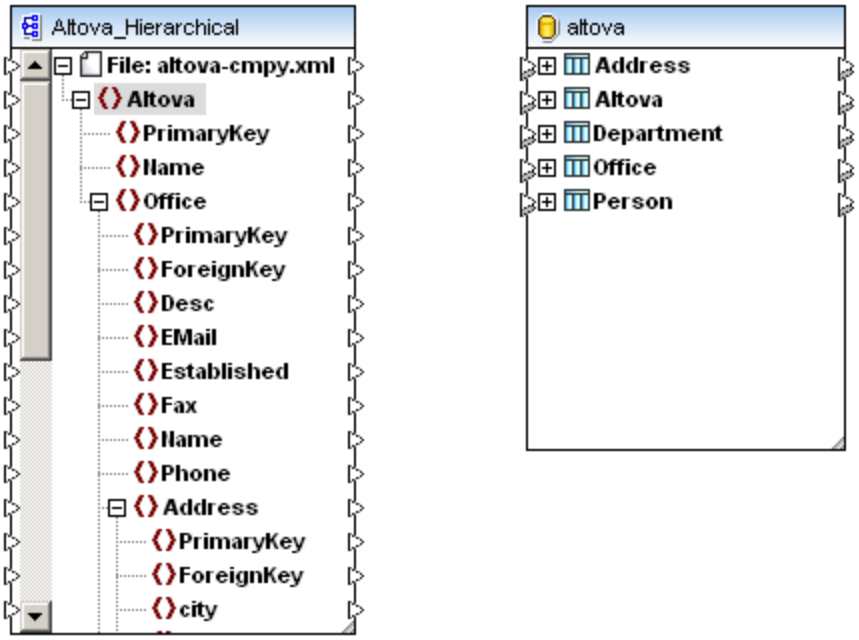
1. Click a programming language icon in the title bar to specify the language the generated code should support. This setting also loads the language related library into the Libraries window.
2. Click the **Insert Schema | XML instance** icon, and select the **Altova_Hierarchical.xsd** .
3. Select the **Altova-cmpy.xml** file as the XML instance file. Click the **Altova** entry, and hit the ***** key on the numeric keypad to view the items; resize the component if necessary.
4. Click the **Insert Database** icon, select the Microsoft Access (ADO) entry and click Next.

5. Click the Browse button to select the **altova.mdb** database available from the ...\MapForceExamples\Tutorial\ folder, and click Connect.

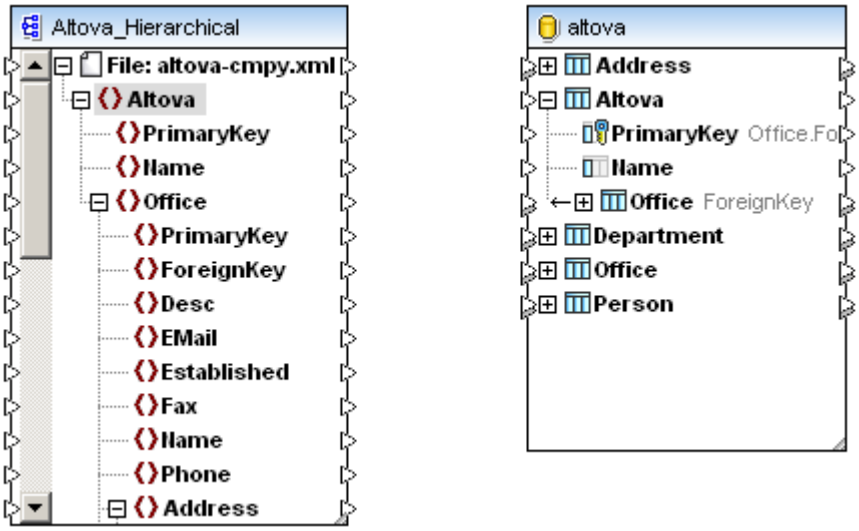
This dialog box allows you to define the specific Tables, Views or System tables that you want to appear in the Database component.



6. Click the check box to the left of User Tables to select them all, and click Insert to insert the database.



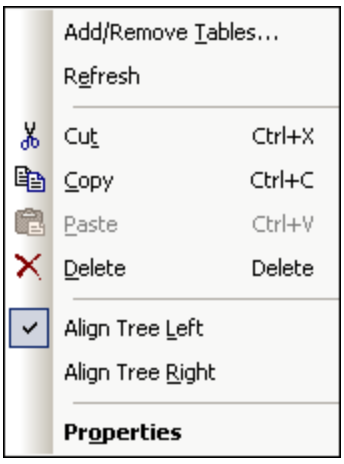
7. Click the + expand icon of the **Altova** item, to display the Altova table fields.



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.

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.

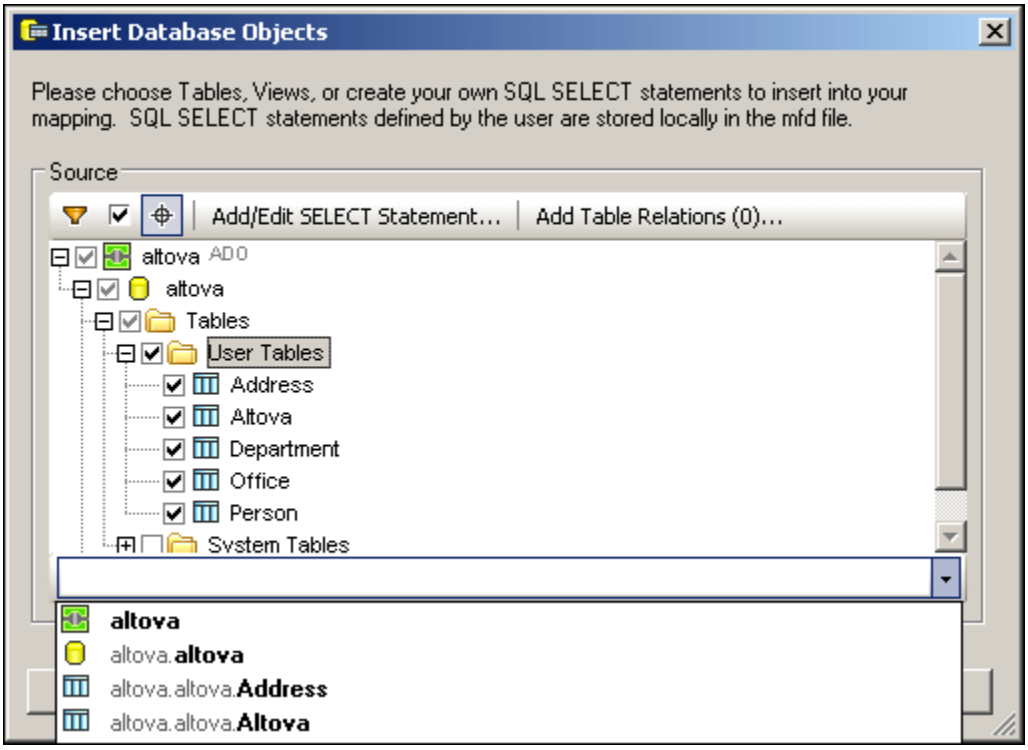
15.3.2 Inserting databases - table preview customization

The Select Tables, Views dialog box contains an icon bar which allows you customize, or find specific items in the table preview window.

- **Object Locator** allows you to find specific database items
- **Filter** allows you to restrict tables by existing characters
- **Show checked objects only** 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.

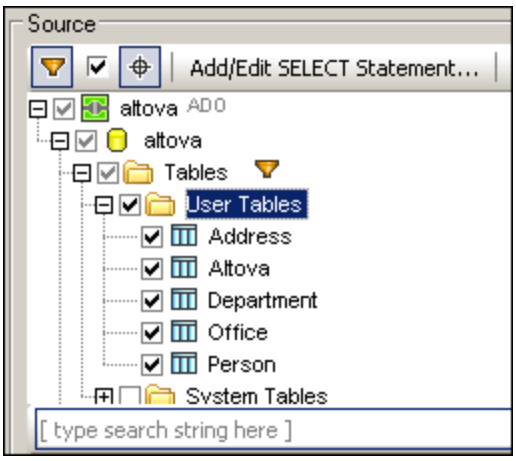


2. Enter the string you want to search for, or select the item from the drop-down list.

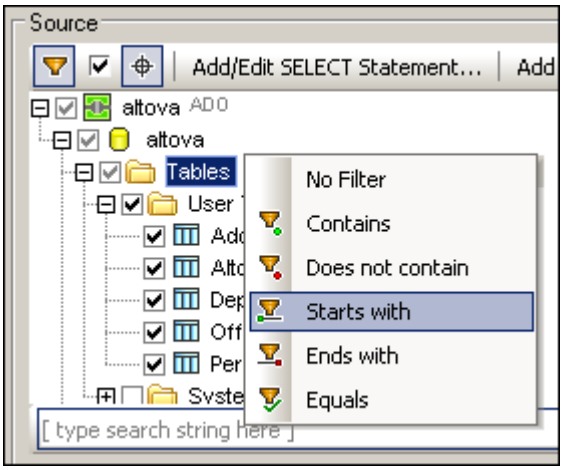
Filtering objects in the preview window:

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.

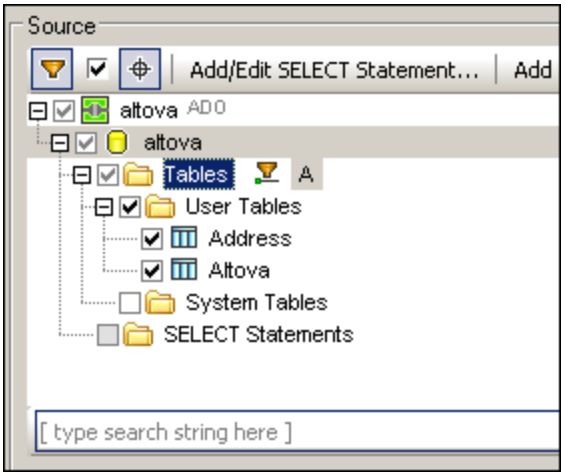


2. Click the filter icon next to the folder, and database objects, you want to filter. Select the filtering option from the popup menu that appears.



An empty field appears next to the filter icon.

3. Enter the characters you want to act as the filter e.g. A.

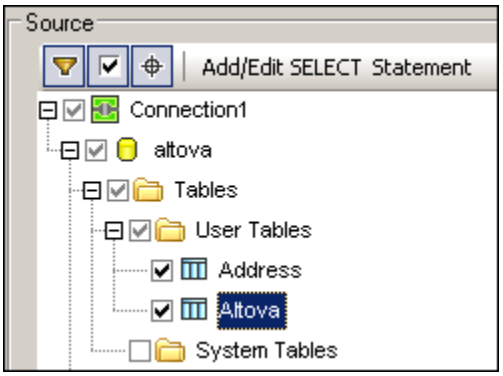


The results are automatically updated as you type.

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.



15.3.3 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.

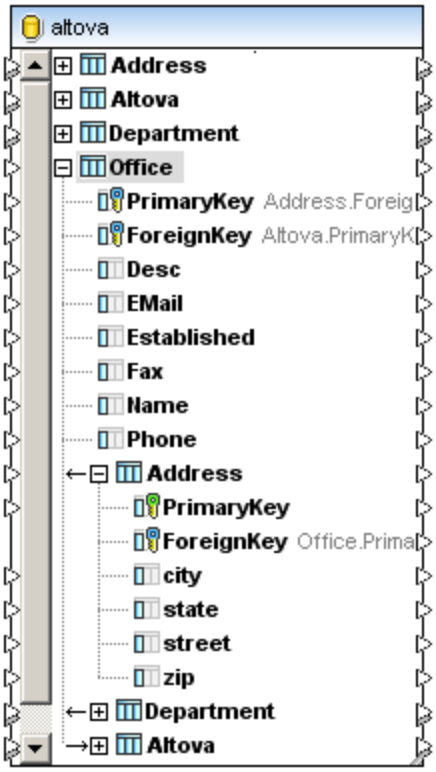
Let us call the table names visible in the above diagram "**root**" tables, i.e. they are the top level, or **root** of the tree view. Expanding a table name displays all the tables related to it. The "root" tables are usually displayed in alphabetical sort order; this has no bearing on the actual table 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.

15.3.4 Database action: Insert

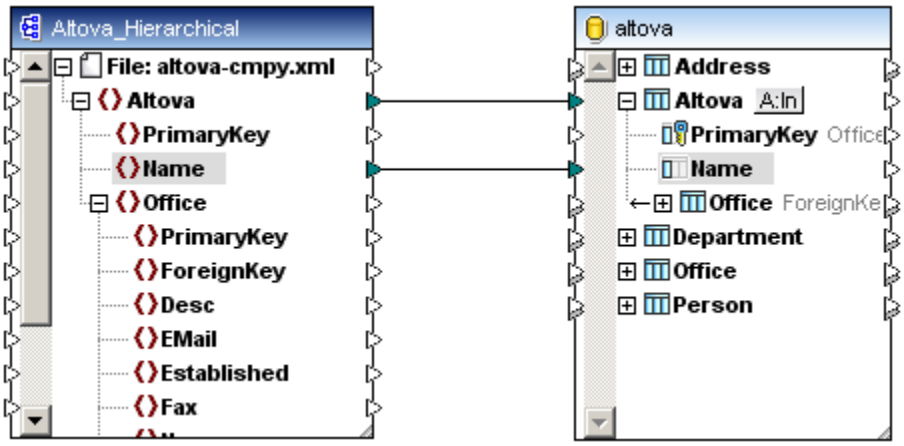
The first example in this section, deals with the simple task of **adding** a new office orgchart to the Altova table.

The second example **inserts** related office tables to the new orgchart record.

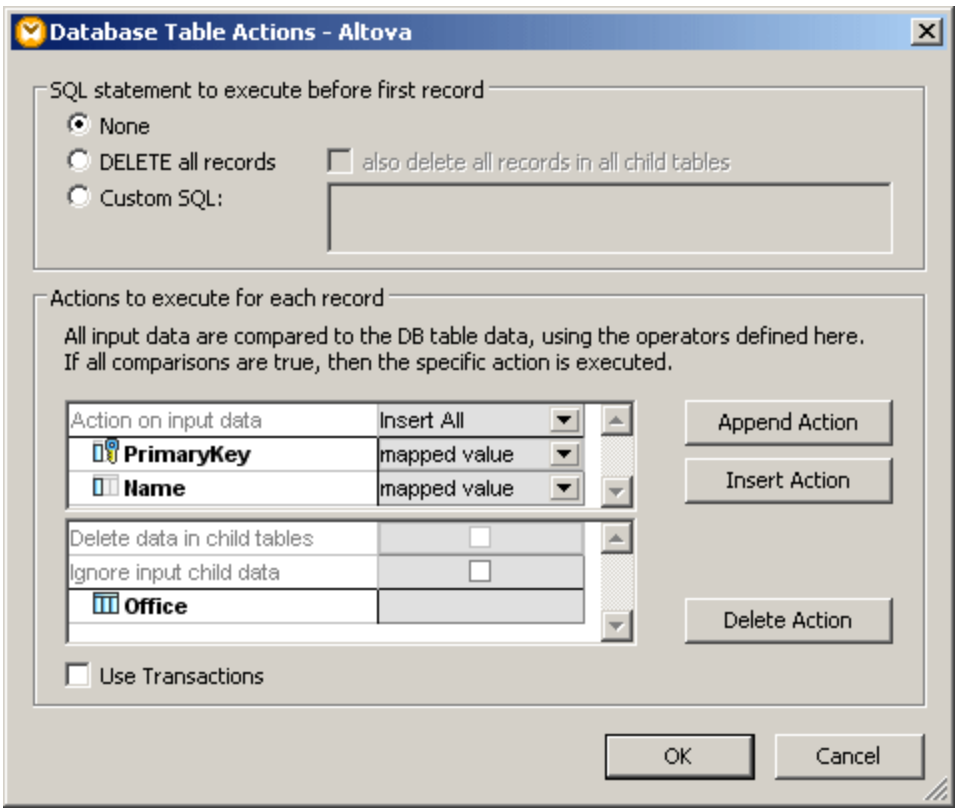
1. Insert the **Altova_Hierarchical.xsd** schema (and assign **altova-cmpy.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping.
3. Create the following mappings:

   Altova to Altova, and
   Name to Name

   Please note: If all Altova, Office etc. items are automatically mapped, the option "Auto-connect children" is active. Select undo, and then the menu option **Connection | Auto-connect matching children,** to disable this option.

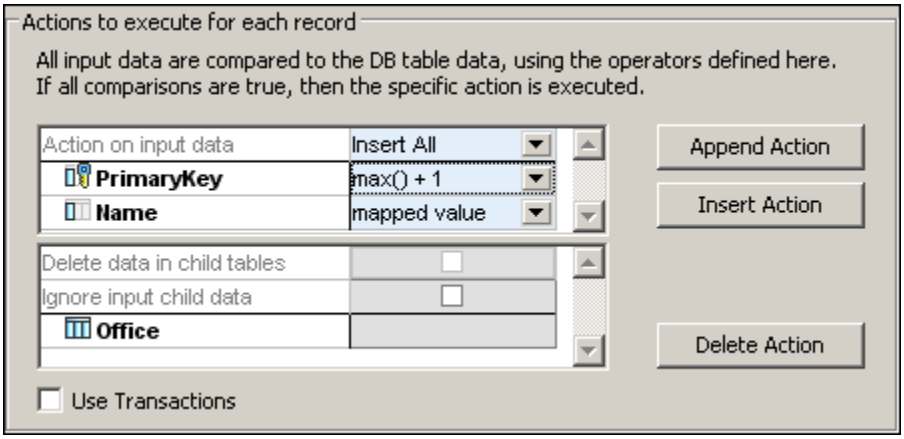


4. The icon to the right of the table (**A:In** in the screen shot above) displays the currently defined **table action** defined for that table, i.e. Action:Insert. Clicking the icon opens the Database Table Actions dialog box where the various table actions can be defined.
   Click the Table Actions A:In icon next to the **Altova** table entry.
   There is currently only one table action column defined in this dialog box, **Insert All**.



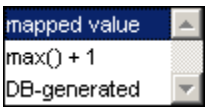
The table action, Insert All, will insert records with all **mapped fields** of the current table into the database. We now have to define how to generate the new PrimaryKey field for this action.

5. Click the combo box to the right of the **PrimaryKey** field and select **max() + 1**, then click OK to confirm.



Database key settings

The database key settings for the Insert action, are set using the combo boxes to the right of each field.



- The **mapped value** option, allows source data to be mapped to the database field

directly, and is the standard setting for all database fields.

- To generate the key values based on the existing keys in the database, select "**Max() + 1**", so that new records are automatically appended to existing ones.
- Use the **DB-generated** setting when the database generates/uses the **Identity function** to generate key values.

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%')
```

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. Click OK to confirm.
8. Open the Altova database in Access to see the effect.

Altova : Table

	PrimaryKey	Name
+	1	Organization Chart
+	2	Microtech OrgChart

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.

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).
```

Please note:

You can rerun SQL scripts from the Output window by clicking the Regenerate Output icon  .

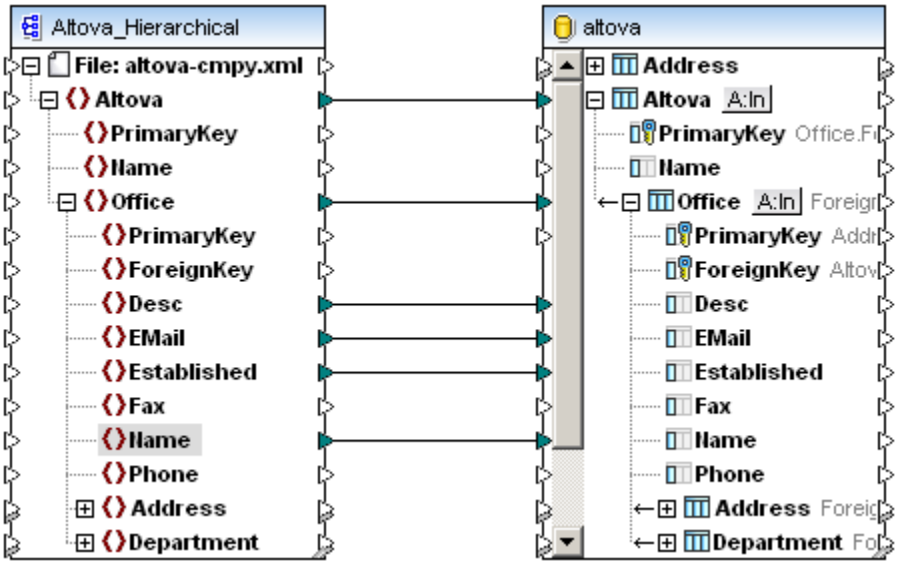
The script executes the mapping to the target database, taking the defined table actions into account.

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.

1. Click the **Office** table actions icon.
   The **Insert All..**. table action is selected by default, you do not have to make any changes here.
2. Click the **PrimaryKey** combo box and select the **max()+1** entry.
3. Create the following mappings between the two components:
   Office to Office
   Desc to Desc
   Email to Email
   Established to Established, and
   Name to Name.



4. Click the Output tab 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%')

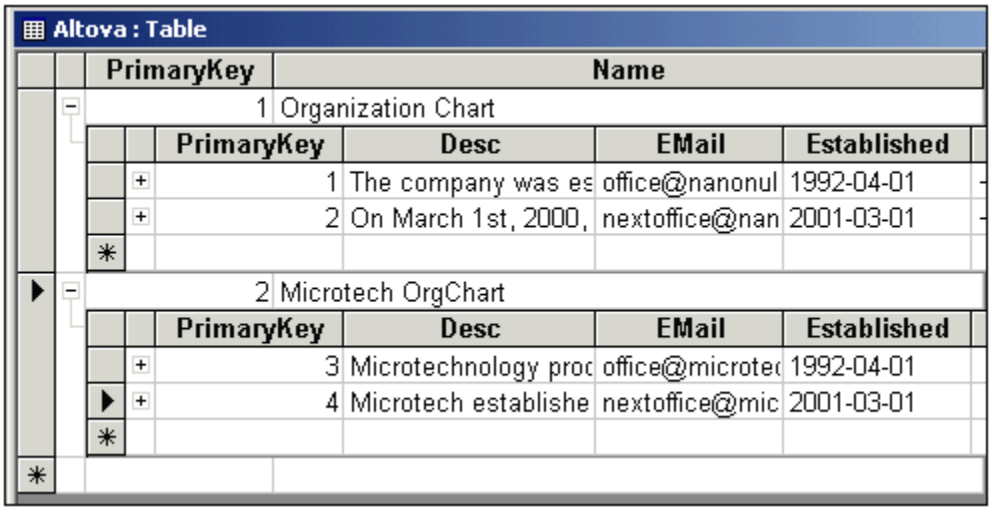
SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM
[Office]
-->>> %PrimaryKey2%

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%')
```

6. Click the Run SQL script icon  to run the script and insert the new tables.
7. Double click the **Altova** table to see the effect in MS-Access.



Two new offices have been added to the Microtech OrgChart.

8. 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).
```

15.3.5 Database action: Update

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.

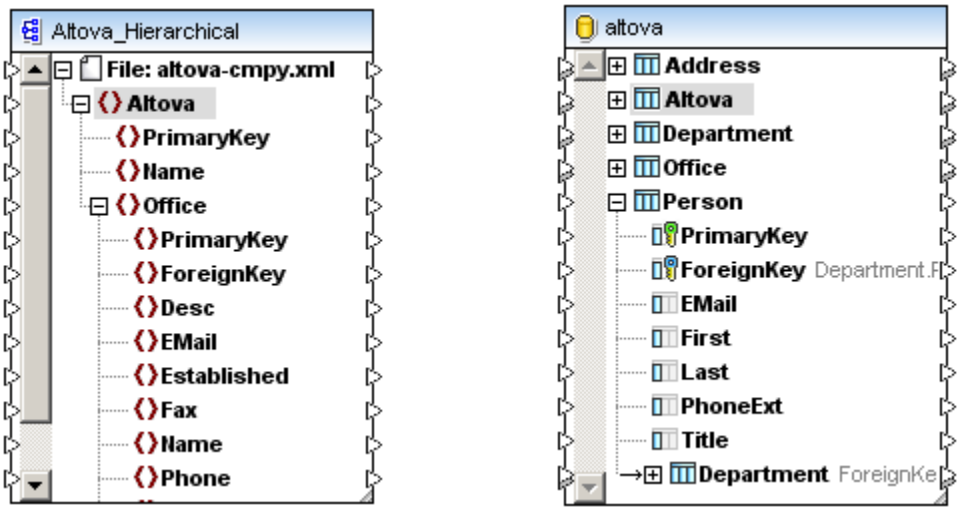
Files used in this example:

- Altova_Hierarchical.xsd
- altova-cmpy.xml
- altova.mdb

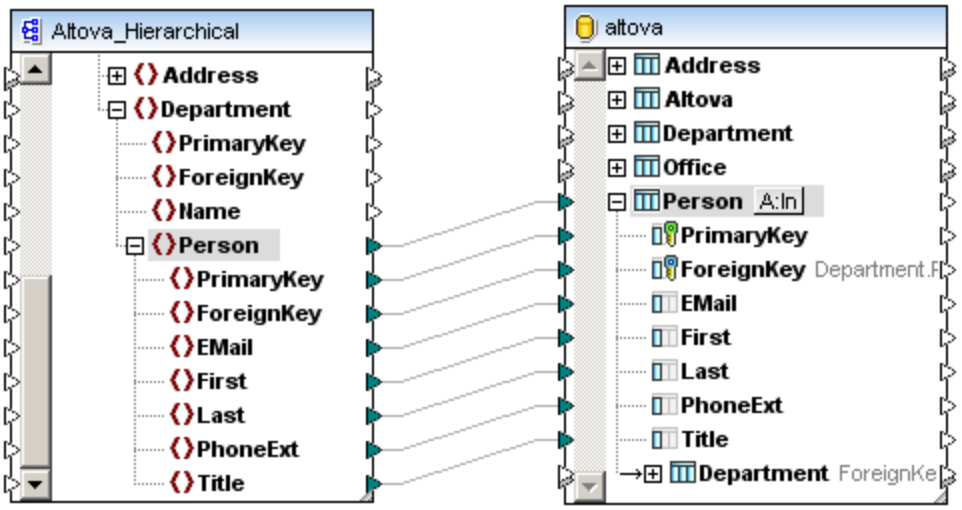
Aim:

To **update** the person fields of the Person table.

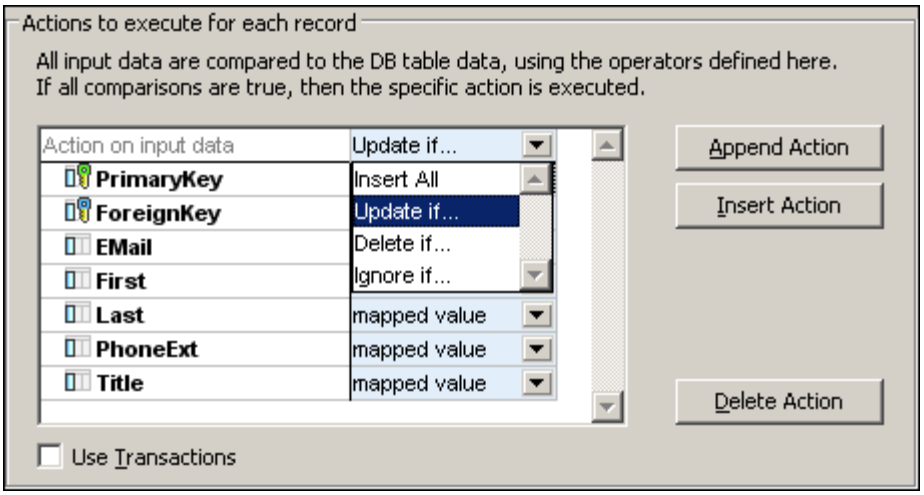
1. Insert the Altova_Hierarchical schema (and assign **altova-cmpy.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping.



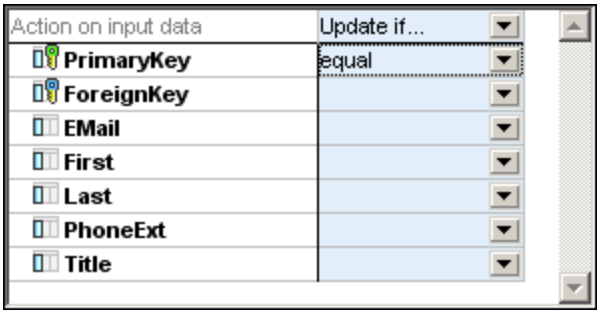
3. Activate the "Auto connect matching children" icon 
4. Click the **Person** item in the XML source file and drag the connector to the Person item of the database. Make sure that you connect to the "**root**" table, Person.
   All matching child items are mapped automatically.



5. Click the **Person** Table Actions icon A:In to open the dialog box.
6. Click the Action on input data combo box, the topmost entry, and select **Update if...**

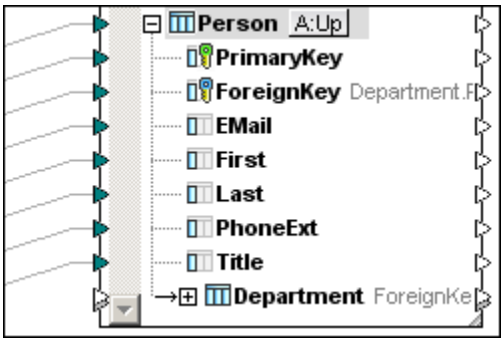


7. Click the combo box in-line with the PrimaryKey entry, and select the **equal** entry, click OK to confirm.



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 tab 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. Click OK to confirm.

10. Open the **Altova** database, 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

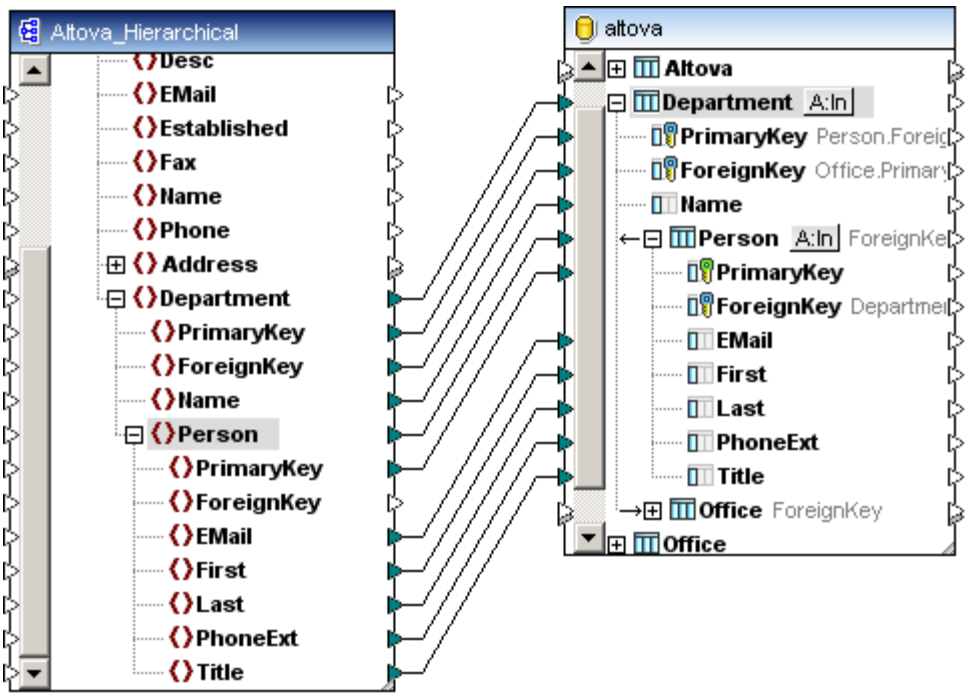
Second Example:
This slightly more complex example, attempts to update records in both the Department and Person tables, as well as add any new Person records which might exist in the XML input file. The "**root**" table used in this example is thus the **Department** table.

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



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.

Table action: **Department** table

- Table actions **Update if... "equal"** defined for PrimaryKey, i.e. update the Department name if it has changed.

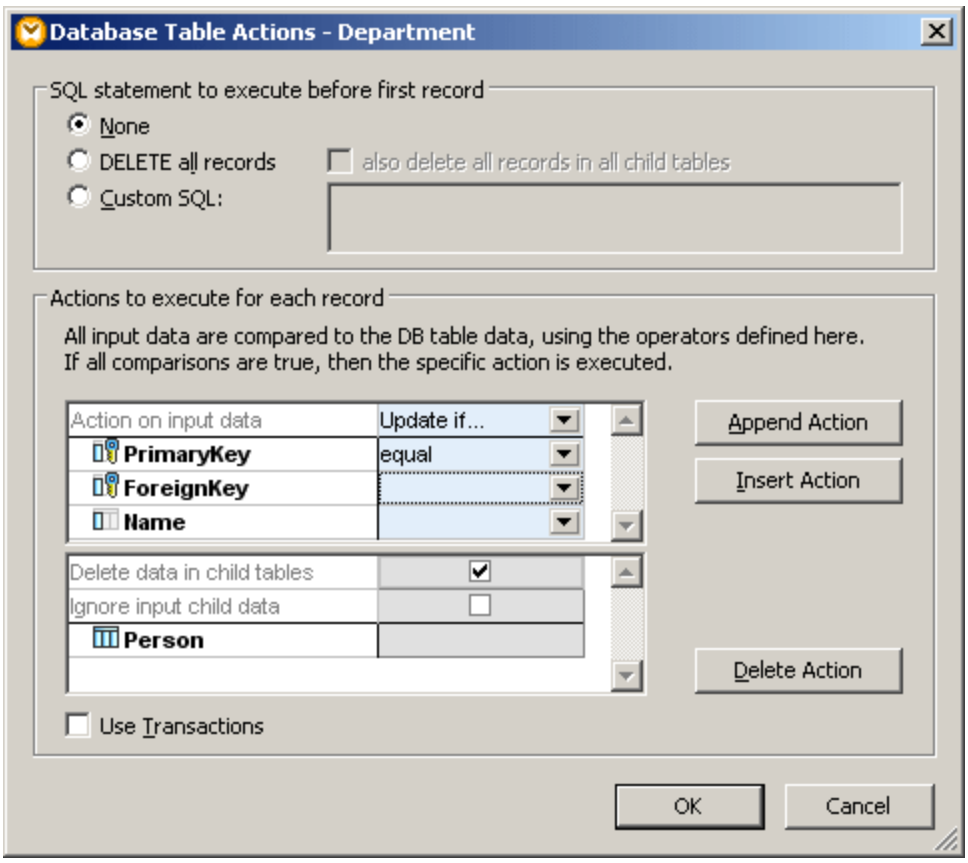
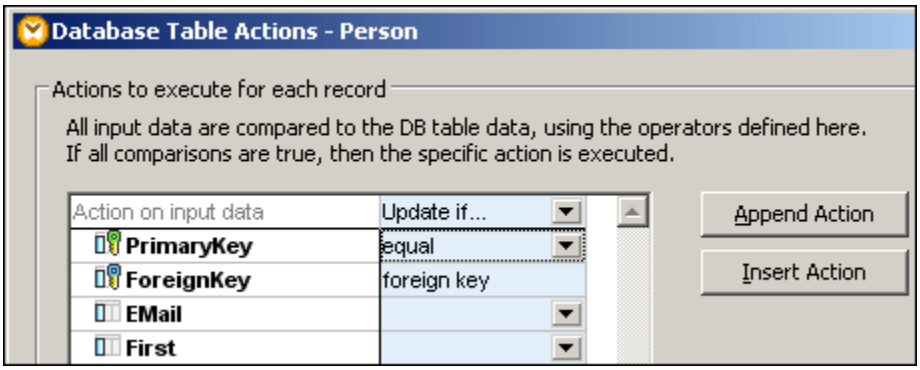
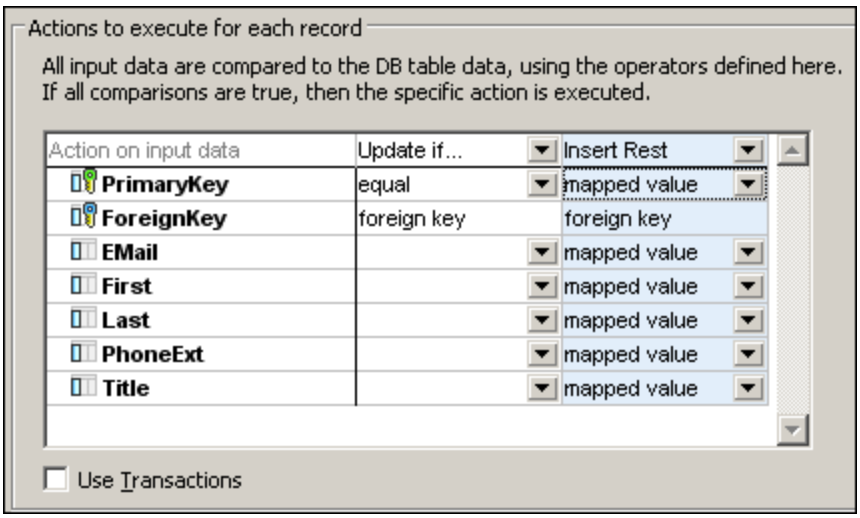


Table action: **Person** table

- Table action **Update if... "equal"** defined for PrimaryKey.



1. 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.



2. Click OK to complete the table actions definition. Note that the table actions icon has now been updated to **A:Up,In**.

4. Click the Output tab 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 Department table:

Department table: **Update if**... condition **true**:
source and target keys are identical, therefore:

- update each Department record where the keys are identical.
- 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).

Department table: **Update if**... condition **false**:
source and target keys are not identical, i.e. source keys exist which have no match in the target database,

the update if... condition fails, therefore:

- none of the Department records are updated.

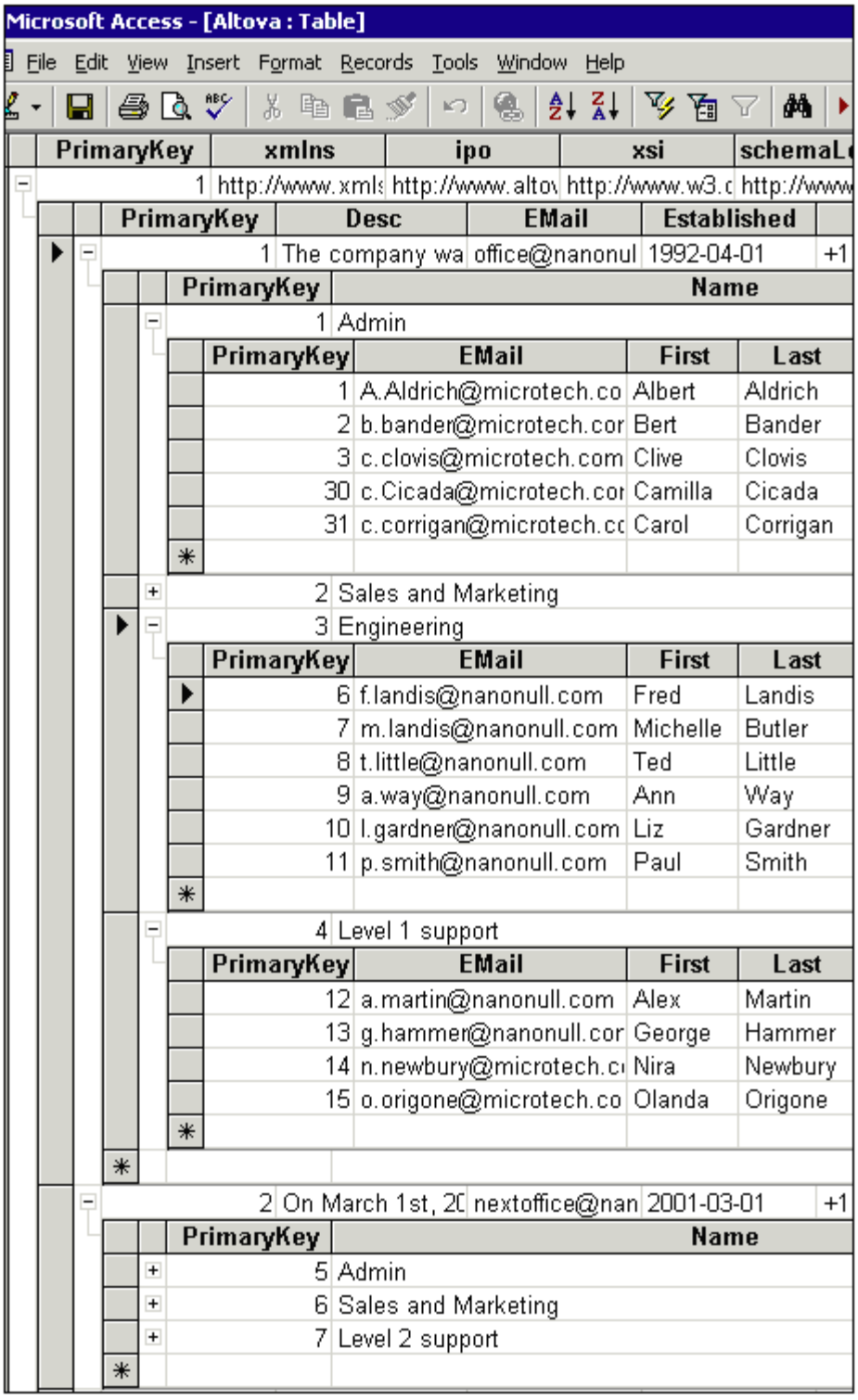
Processing sequence Person table:

Person table: **Update if**... condition **true**:
source and target keys are identical, therefore:

- update each Person record where the keys are identical.
- if records exist in the database with no counterpart in the **source** file, then these records are retained and remain unchanged.

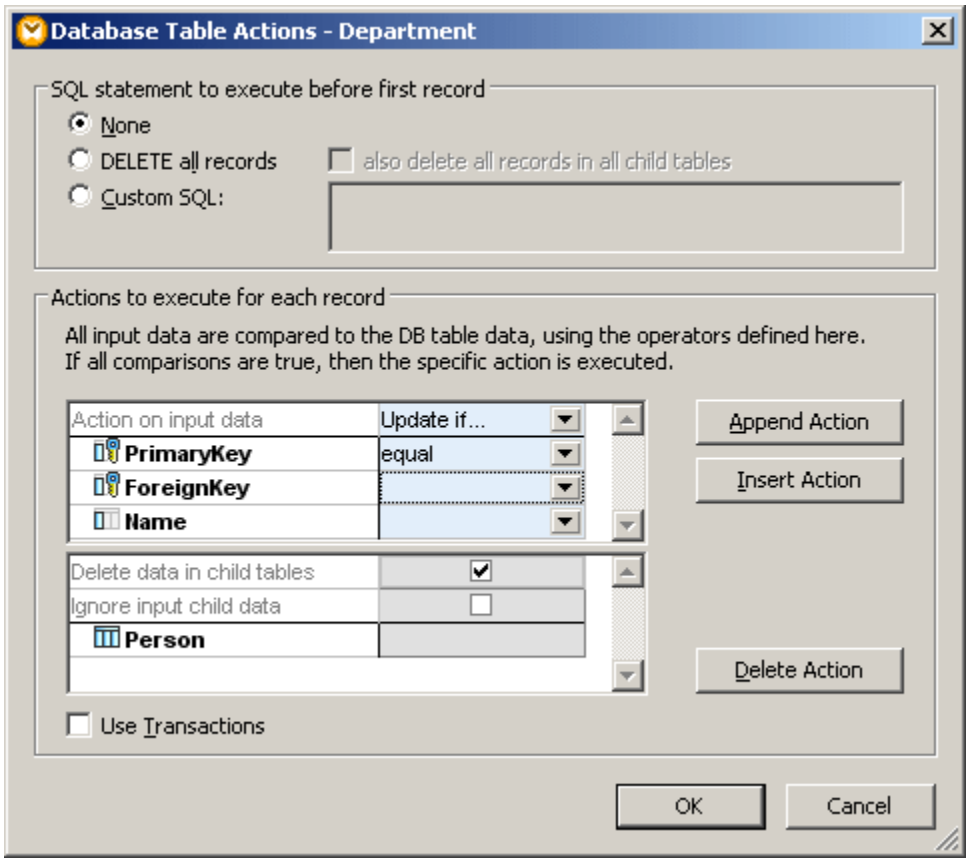
Person table: **Update if**... condition **false**:
source and target keys are not identical, i.e. source keys exist which have no match in the target database, the update if... condition fails, therefore:

- move on to the next Table Action column: **Insert Rest**...
- insert the new Person records into the Person table if any exist.
  In this case, two new person records are added to the Admin department, with the person primary keys of 30, and 31, respectively.



Update if... combinations - with delete child data

This section describes the effect of the **Update if...** condition on a parent table combined with each of the possible table actions 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

Update if.. on parent table, **Insert all...** on child table

Parent table - **Department**			
Table action		Update if...	compare PrimaryKey
Delete data in child tables	☑		
Ignore input child data	☐		
child table - **Person**			
Table action		**Insert all...**	compare PrimaryKey
Delete data in child tables			
Ignore input child data			

Result:

- Updates parent table data (Department records)
- Deletes child data of those tables which satisfy the Update if... condition (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... on parent table, **Update if...** on child table

Parent table - **Department**			
Table action		Update if...	compare PrimaryKey
Delete data in child tables	☑		
Ignore input child data	☐		
child table - **Person**			
Table action		**Update if...**	compare PrimaryKey
Delete data in child tables			
Ignore input child data			

Result:

- Updates parent table data (Department records).
- Deletes child data of those tables which satisfy the Update if... condition (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).
- 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.

Update if... on parent table, **Delete if...** on child table **(Delete data in child tables - active)**

Parent table - **Department**			
Table action		Update if...	compare PrimaryKey
Delete data in child tables	☑		
Ignore input child data	☐		
child table - **Person**			
Table action		**Delete if...**	compare PrimaryKey
Delete data in child tables			
Ignore input child data			

Result:

- Updates parent table data (Department records).
- Deletes child data (Person records) from all Departments because the "**Delete data in child tables**" option is active. All Person records are 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.
  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).
- The child table data (Person records) are deleted before the Table action, Delete if..., is executed, no records are deleted.

Update if... on parent table, **Delete if...** on child table **(Delete data in child tables - deactivated)**

Parent table - **Department**			
Table action		Update if...	compare PrimaryKey
Delete data in child tables	☐		Delete data... **not active** !
Ignore input child data	☐		
child table - **Person**			
Table action		**Delete if...**	compare PrimaryKey
Delete data in child tables			
Ignore input child data			

Result:

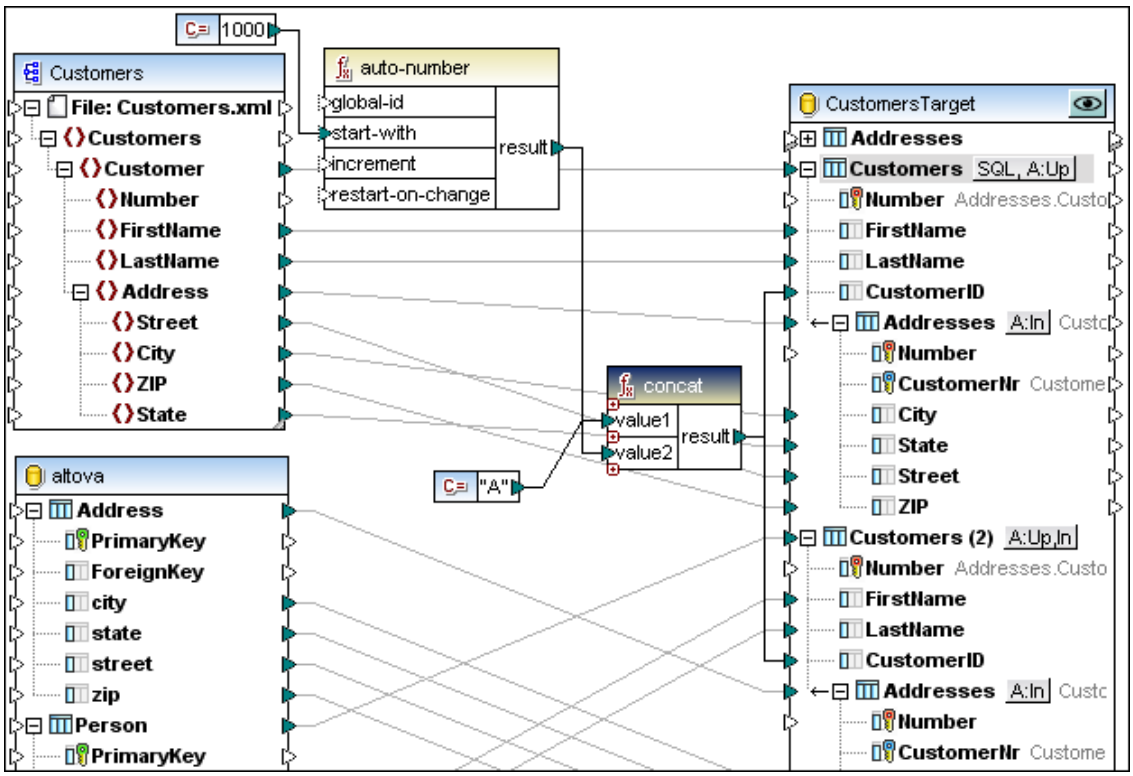
- Updates parent table data (Department records).
  Retains existing database tables (Engineering in this case) which do not have a counterpart in the input XML file (no source keys for the Update if... comparison).
- Delete if... only deletes those Person records for which a corresponding **Person** PrimaryKey exists in the source XML file.
- 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 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 database. The target table (Customers) is duplicated.

  CustomerID for each record are created anew, with the initial value being A1000.


1000
Customers
File: Customers.xml
Customers
Customer
Number
FirstName
LastName
Address
Street
City
ZIP
State
auto-number
global-id
start-with
increment
restart-on-change
result
concat
value1
value2
result
"A"
altova
Address
PrimaryKey
ForeignKey
city
state
street
zip
Person
PrimaryKey
CustomersTarget
Addresses
Customers SQL, A:Up
Number Addresses.Custo
FirstName
LastName
CustomerID
Addresses A:In Custc
Number
CustomerNr Custome
City
State
Street
ZIP
Customers (2) A:Up,In
Number Addresses.Custo
FirstName
LastName
CustomerID
Addresses A:In Custc
Number
CustomerNr Custome

15.3.6 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.

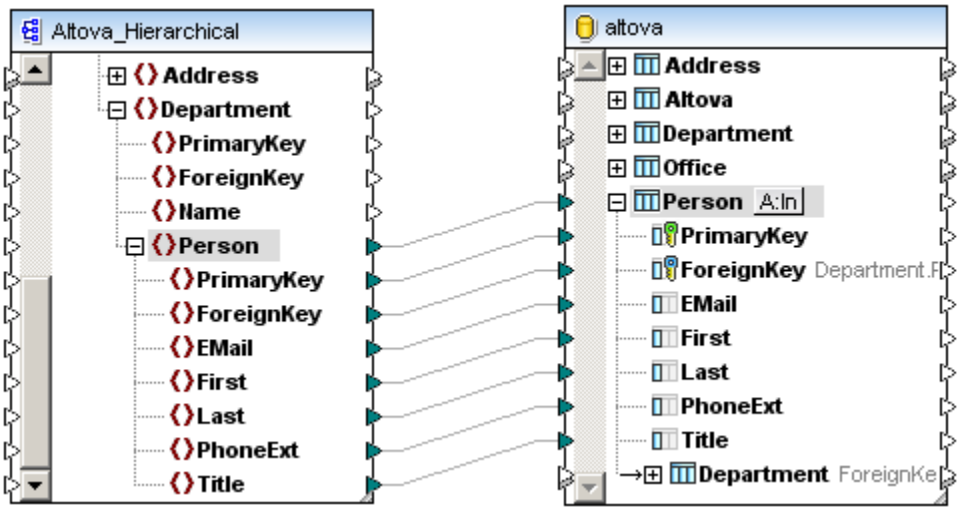
Please note:

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.

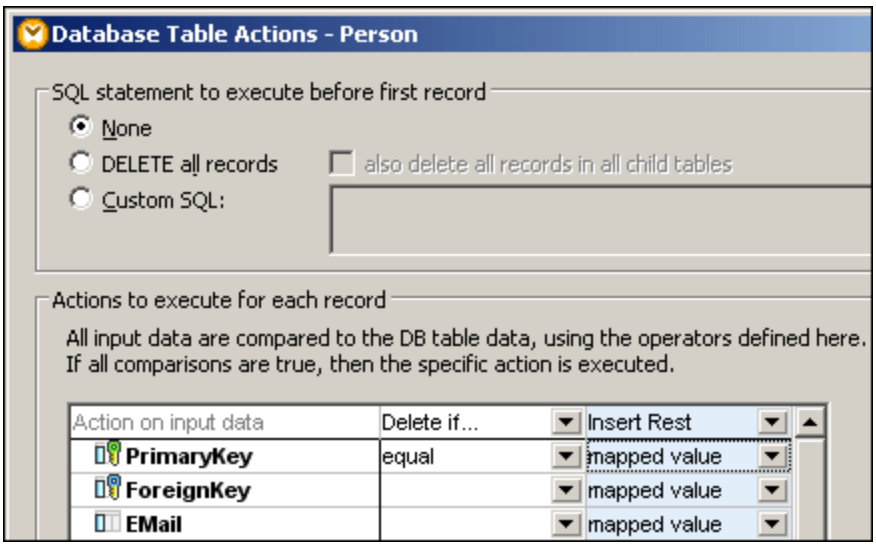
Aim:

- To delete the existing Person records in the database, and
- Insert new Person records from the input XML file.

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.



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 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".



The source and target primary keys are compared using the "equal" operator. If the two keys are identical, then the mapped fields of the Person tables are deleted. Once this has been achieved, the next table action is started, in this case Insert Rest.

Insert Rest inserts all those records, from the source XML file, which do not have a counterpart key/field in the database.

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.

8. Click the Output tab 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. Click OK to confirm.
10. Open the Altova database 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.(	Camilla	Cicada
	31	1	c.corrigan@microtech	Carol	Corrigan
*					

Person table: **Delete if**... condition **true**:
source and target keys are identical, therefore:

- delete each Person record where the keys are identical
- if records exist in the database with no counterpart key/field in the source file, then these records are not deleted and remain unchanged.

Person table: **Delete if**... condition **false**:
source and target keys are not identical, i.e. source keys exist which have no match in the target database, the delete if... condition fails, therefore:

- move on to the next Table Action column: **Insert Rest**...
- insert the new Person records into the Person table if any exist.

  In this case, two new person records are added to the Administration department, each with the person primary key of 30, and 31, respectively.

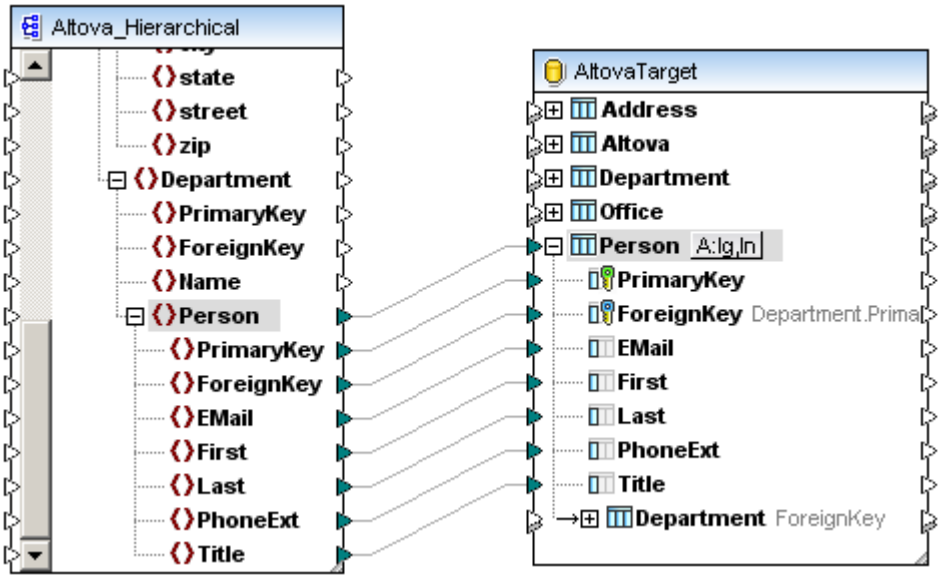
Two additional examples of the Delete if... table action can be viewed in the Update if... combinations section.

15.3.7 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.

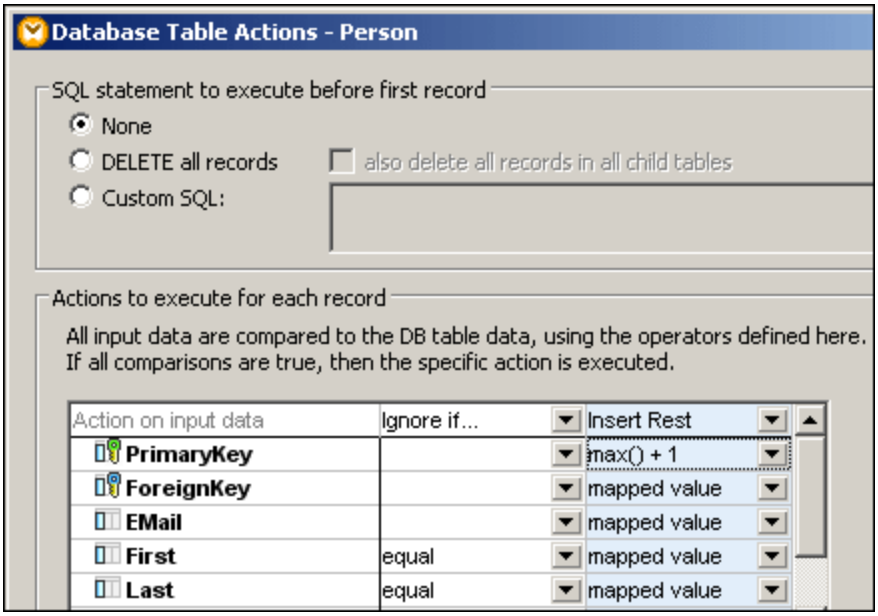
This action can be used together with a following "Insert Rest" action to ignore all input records that already exist in the database, and to insert any new ones.

Example:



Aim:

- To ignore duplicate Person records in the database, and
- Insert new Person records from the input XML file.



- The **Ignore if..**. table action compares the First and Last fields of the source XML with

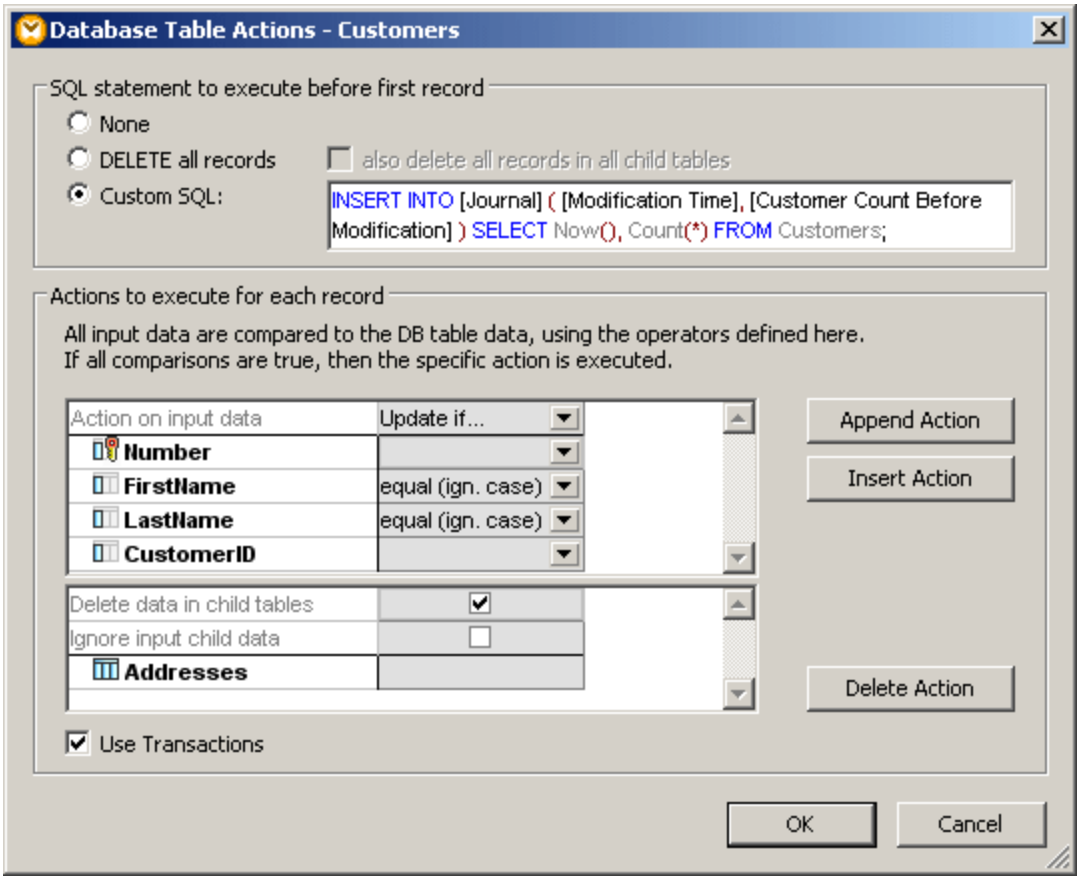
the same fields in the database.

- 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.

15.3.8 Table Actions, key settings, transaction processing

The Database Table Actions dialog box lets you specify:

- An optional SQL statement to be executed prior to any other table action
- The table actions to be performed for each record delivered to the target table by the mapping
- The database key settings for inserting new records



The **SQL statement to execute...** section allows you to define SQL Statements that affects the table as a whole.

- You can DELETE all records of the table **prior** to processing the **table actions**.
- Write a custom SQL statement to affect the complete table.

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.

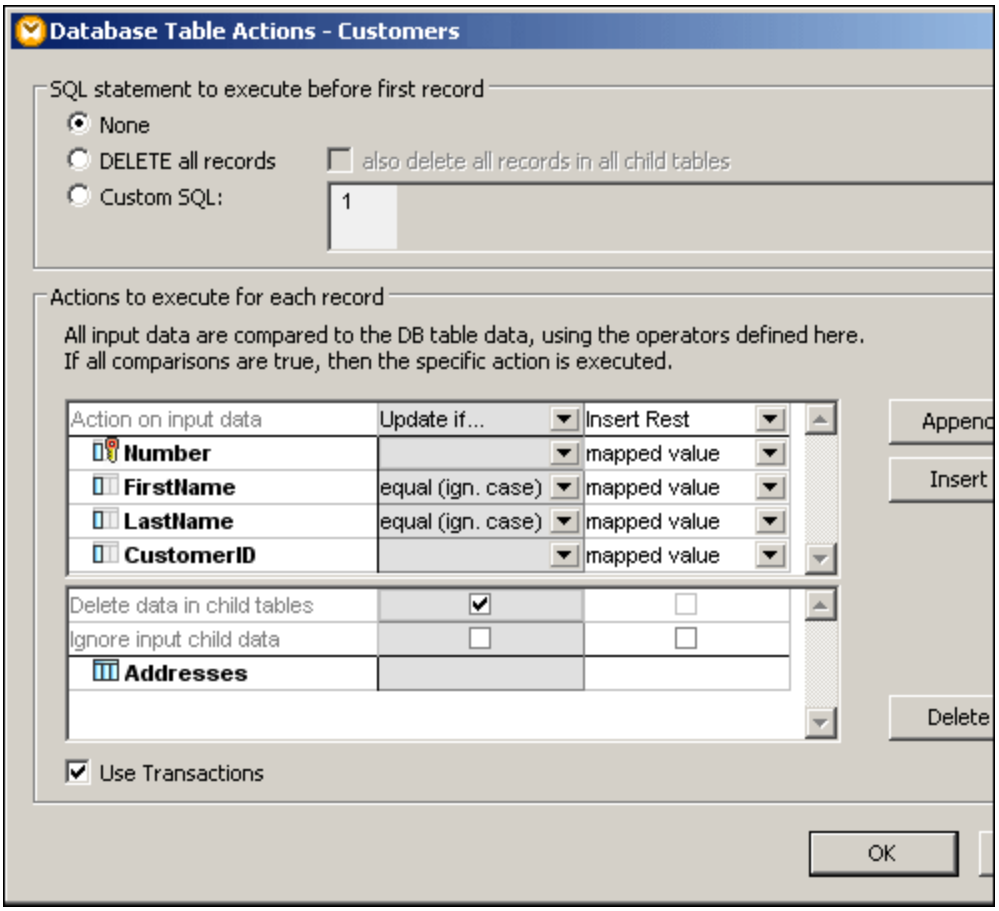
Table actions

The Table Actions define how the mapped records are processed. MapForce supports the table actions: Insert, Update, Delete and Ignore. One or more fields are used to compare source and target data to determine if the table action is to be executed.

The Update, Delete and Ignore table actions require a **condition** that defines for which records they should apply.
The Insert action is unconditional but it has **key settings** to define how to generate the primary key.

For each 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.

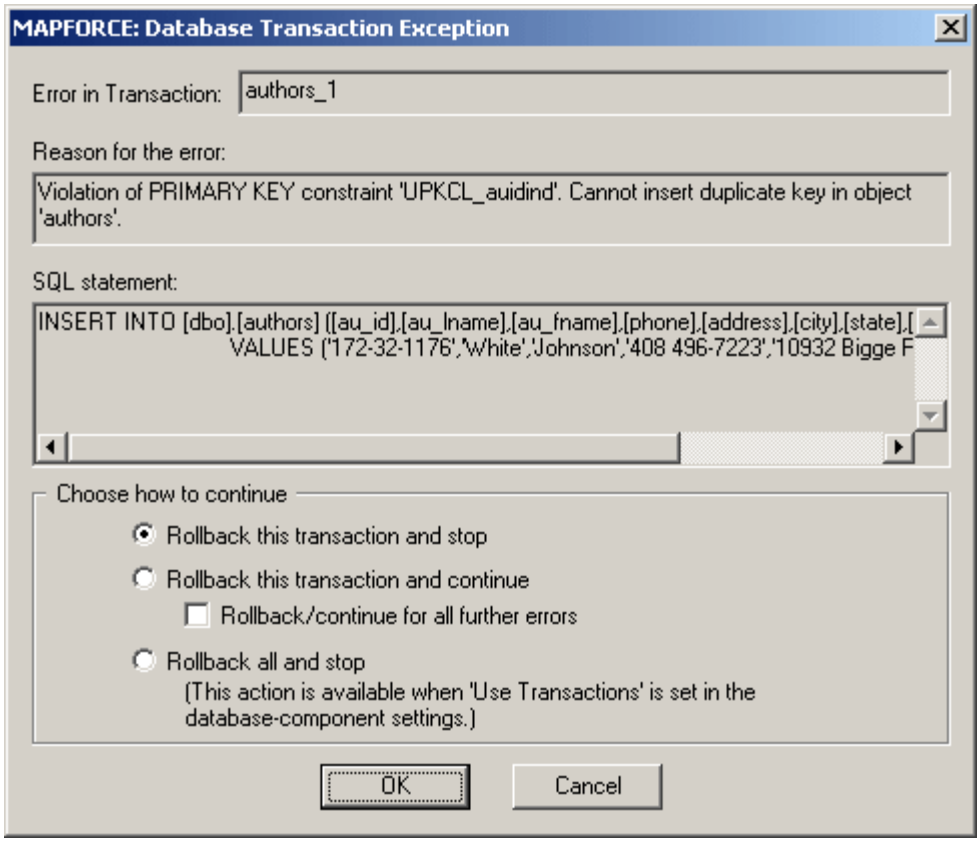
Ignore input child data:
Use this option when you want to update specific table data, without affecting any of the child tables/records of that table.

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 setting can also activated for the database component, by right clicking it, in the Component Settings dialog box of the respective database component. 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.

15.3.9 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, or database.database.

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.

15.4 Database feature matrix

The following tables supply information on the mapping capabilities of MapForce vis-a-vis the major database types.

The following information is supplied:

- General info relating to Database as service, and authentication issues
- Supported connection types
- SQL support for: schemas, join statements etc.
- Transaction methods supported

15.4.1 Database info - MS Access

	MS Access	supported	Notes
General:			
	DB engine as service	n	implemented in OLEDB-provider or ODBC-driver
	own authentication	y	authentication is possible
	Trusted authentication	n	
Connection:			
	OLE DB	y	
	OLE DB connection-string issues	none	
	ODBC	y	
	ODBC connection-string issues	DBQ	must be applied
	ODBC connection-string issues	DATABASE	must not be applied
	JDBC	y	via ODBC
	JDBC URL issues	none	
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	[] or ""	
	support for DB-schemas	n	not supported by Access
	identity support	y	
	MF: read back identity value	@@IDENTITY	
	sub select support	y	
	JOIN support	n	limited support
	MF: upper function	Ucase(..)	

SQL-Execution:			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	n	
	retrieve parameter types	?	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	n	not supported by Access
	set transaction isolation	n	not supported by Access
	MF: begin transaction	API-call	

	MF: commit transaction	API-call	
	MF: rollback transaction	API-call	
	MF: set save point	--	
	MF: rollback to save point	--	
	MF used init. Statements	none	

The "datetime" data type has different semantics for XML Schema and Access. In XML Schema, the date is mandatory, in Access it is optional. Since MapForce uses XML Schema datatypes internally, this prevents the user from directly mapping xs:time fields to a datetime field in Access.

The workaround is to:
Use the "datetime-from-date-and-time" function and use the date "1899-12-30". This is the date that Access uses internally for "only-time" values.

15.4.2 Database info - MS SQL Server

	MS SQLServer	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	
	Trusted authentication	y	
Connection:			
	OLE DB	y	
	ODBC	y	
	ODBC connection-string issues	Select Method=Cursor	must be applied
	JDBC	y	
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	[] or ""	
	support for DB-schemas	y	
	identity support	y	
	MF: read back identity value	@@IDENTITY	
	sub select support	y	
	JOIN support	y	
	MF: upper function	UPPER()	
SQL-Execution:			
	exec. multiple-stat. in one	y	
	command separator	';' or 'GO'	
	special error handling	n	
	retrieve parameter types	y	with limits
	special issues when using ?	y	DATETIME datatype not supported when using ODBC
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	y	a MUST when using nested transactions. Mixing API-transaction handling and SQL-transaction-commands is not possible
	Nested transactions supported	y	via SAVEPOINTS
	set transaction isolation	y	
	MF: begin transaction	BEGIN TRANSACTION	
	MF: commit transaction	COMMIT TRANSACTION	
	MF: rollback transaction	ROLLBACK TRANSACTION	

	MF: set save point	SAVEPOINT	
	MF: rollback to save point	ROLLBACK TO	
	MF used init. Statements	none	

15.4.3 Database info - Oracle

	Oracle	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	
	Trusted authentication	n	
Connection:			
	OLE DB	n	not supported by MapForce
	ODBC	y	
	ODBC connection-string issues	DATABASE	must not be applied
	JDBC	y	
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	""	
	support for DB-schemas	y	
	identity support	n	must use triggers
	MF: read back identity value	not supported	
	sub select support	y	
	JOIN support	y	
	MF: upper function	UPPER()	

SQL-Execution:			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	n	
	retrieve parameter types		
	special issues when using ?	n	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	y	via SAVEPOINTS
	set transaction isolation	y	
	MF: begin transaction	API-call	
	MF: commit transaction	API-call	
	MF: rollback transaction	API-call	
	MF: set save point	SAVEPOINT	
	MF: rollback to save point	ROLLBACK TO SAVEPOINT	

	MF used init. Statements	none	

15.4.4 Database info - MySQL

	MySQL	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	
	Trusted authentication	n	
	special issues	TYPE=INNODB	for tables when relations, transactions, are used
Connection:			
	OLE DB	n	not supported by MapForce
	ODBC	y	
	JDBC	y	via ODBC
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	``	
	support for DB-schemas	y	
	identity support	y	
	MF: read back identity value	@@IDENTITY	
	sub select support	n	special implementation for DELETE necessary
	JOIN support	y	
	MF: upper function	UPPER()	

SQL-Execution:			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	n	
	retrieve parameter types	y	with limits
	special issues when using ?	n	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	n	MySQL does not produce an error, and continues if no nested transactions exist
	set transaction isolation	y	
	MF: begin transaction	API-call	
	MF: commit transaction	API-call	
	MF: rollback transaction	API-call	
	MF: set save point	SAVEPOINT	

	MF: rollback to save point	ROLLBACK TO	
	MF used init. Statements	SET AUTOCOMMIT=0	

15.4.5 Database info - Sybase

	Sybase	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	
	Trusted authentication	n	
Connection:			
	OLE DB	n	not supported by MapForce
	ODBC	y	
	ODBC connection-string issues	Select Method=Cursor	must be applied
	JDBC	y	via ODBC
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification		
	support for DB-schemas	y	
	identity support	y	
	MF: read back identity value	@@IDENTITY	
	sub select support	y	
	JOIN support	y	
	MF: upper function	UPPER()	

SQL-Execution:			
	exec. multiple-stat. in one	y	
	command separator	none	
	special error handling	n	
	retrieve parameter types		
	special issues when using ?	y	only ASCII-127 characters are allowed in string constants when using ODBC
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	not supported by MAPFORCE	
	Nested transactions supported	n	Sybase does not produce an error, continues if no nested transactions exist
	set transaction isolation	y	
	MF: begin transaction	API-call	
	MF: commit transaction	API-call	

	MF: rollback transaction	API-call	
	MF: set save point	SAVE TRANSACTION	
	MF: rollback to save point	ROLLBACK	
	MF used init. Statements	none	

Having defined relationships between tables using the Sybase 'sp_primarykey' and 'sp_foreignkey' procedures, it is additionally necessary to use ALTER TABLE to add a constraint to the table describing the foreign key relationship to have the primary/foreign relationships appear in MapForce.

15.4.6 Database info - IBM DB2

	IBM DB2	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	uses local windows user-accounts
	Trusted authentication	y	see 'own authentication'
Connection:			
	OLE DB	n	not supported by MapForce
	ODBC	y	
	JDBC	y	via ODBC
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	""	
	support for DB-schemas	y	
	identity support	y	
	MF: read back identity value	identity_val_local()	
	sub select support	y	
	JOIN support	y	
	MF: upper function	UPPER()	
SQL-Execution:			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	y	
	retrieve parameter types	n	
	special issues when using ?	n	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	n	not supported by DB2
	set transaction isolation	y	
	MF: begin transaction	API-call	
	MF: commit transaction	API-call	
	MF: rollback transaction	API-call	
	MF: set save point	--	not supported by DB2
	MF: rollback to save point	--	not supported by DB2
	MF used init. Statements	none	

15.5 Database relationships - preserve / discard

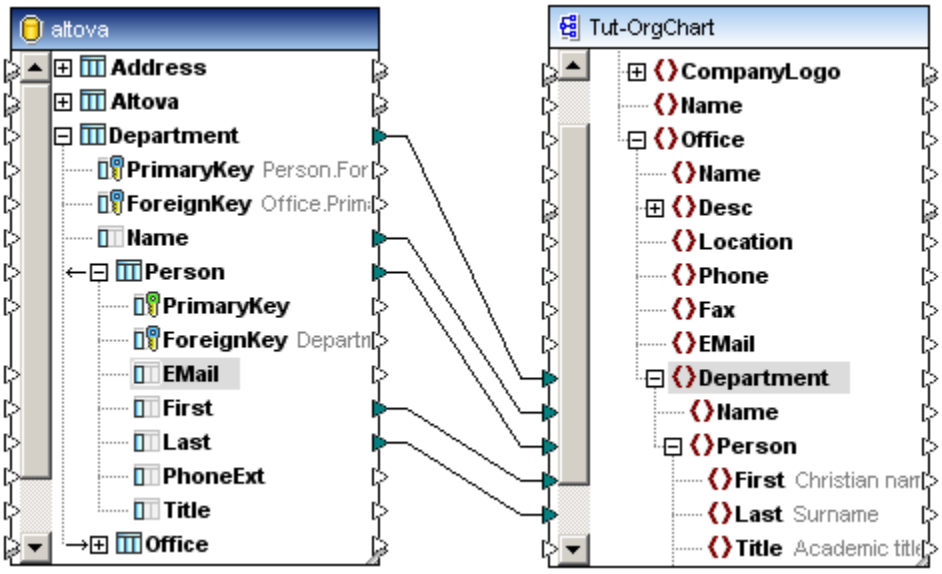
The following section describes how to keep database relationships between tables intact, or how to discard those table relationships.

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.



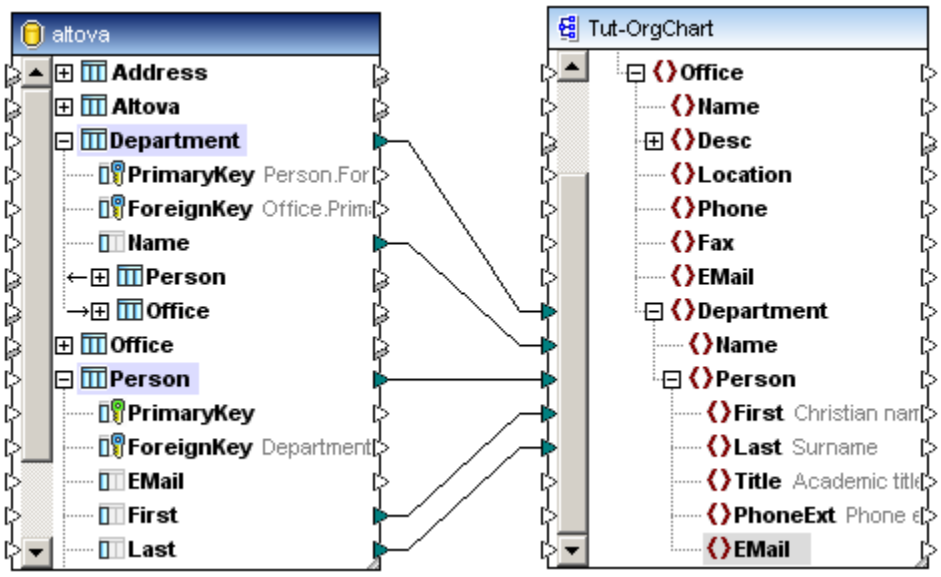
Result of the above mapping:
The names of the persons in each each of the departments is shown in the output component.

```
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>
```

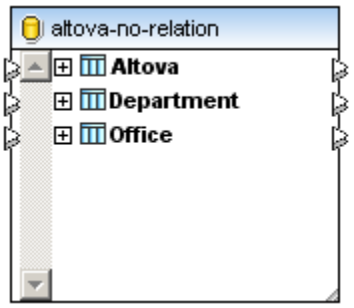
15.6 Local Relations - creating database relationships

MapForce allows you to extract related database data, even if no such relationships explicitly exist in the source database. You can define the primary/foreign key relations between tables, views and SELECT statements 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

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.

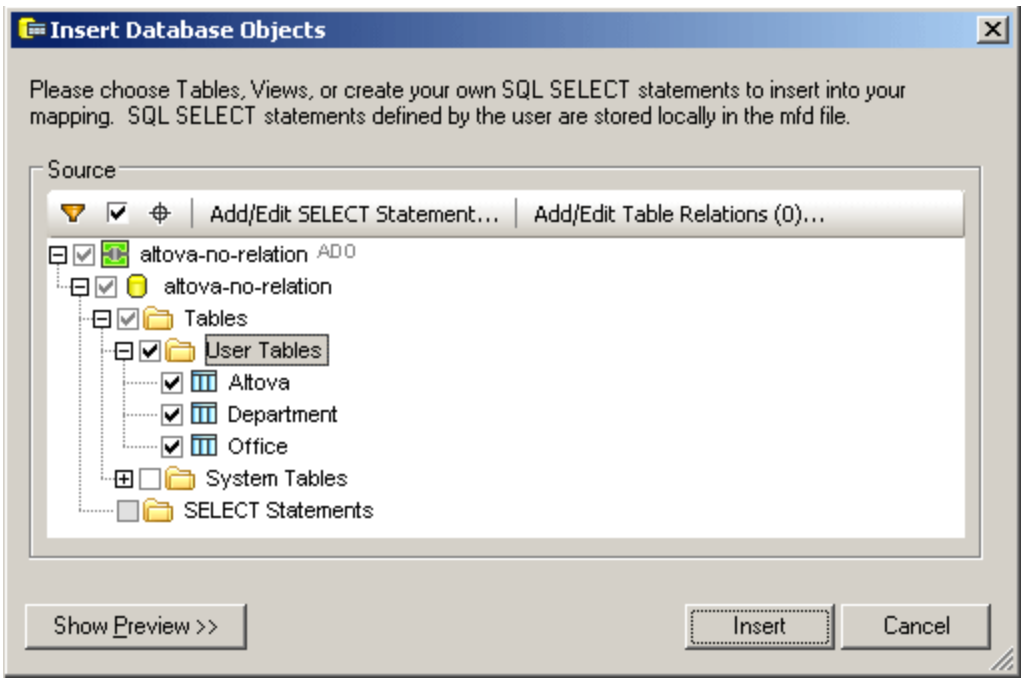


The aim of this example is to display the offices of Altova and show the departments in each.

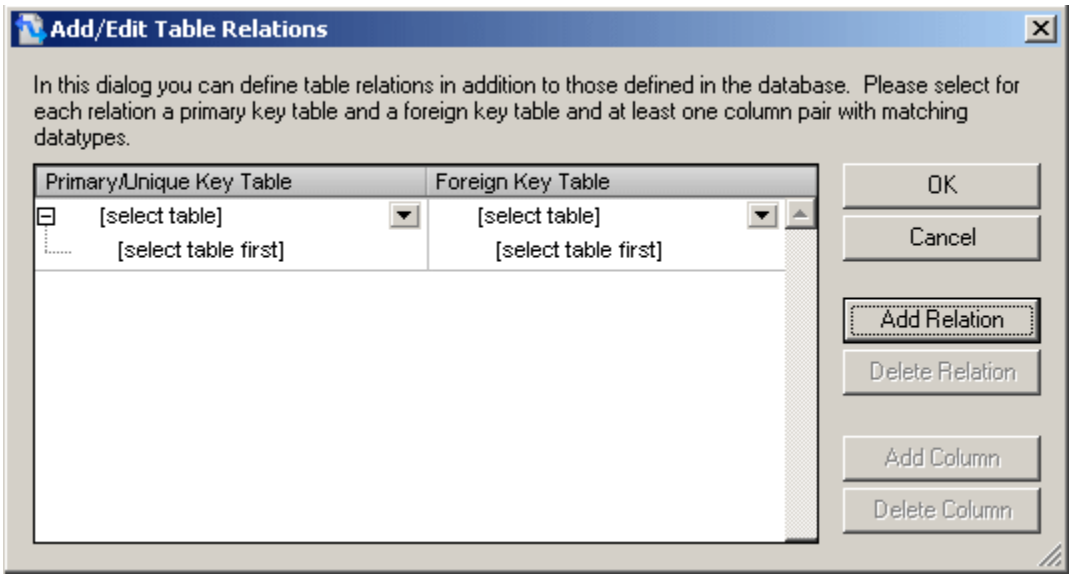
None of the tables visible in the **altova-no-relation** tree have any child tables, all tables are on the same "**root**" level. The content of each table is limited to the fields it contains. We can however, use MapForce to extract related database data, even though relationships have not been explicitly 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.

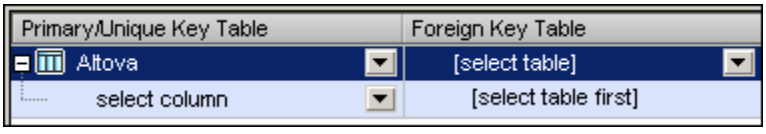


4. Click the **Add/Edit Table Relations** button in the icon bar.
   This opens the Add/Edit Table Relations dialog box.
5. Click the **Add Relation** button.

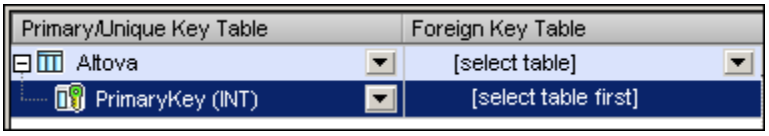


The two combo boxes allow you to select the tables you want to create relations for. The left combo box is the Primary/Unique Key Table, the right one, the Foreign Key Table.

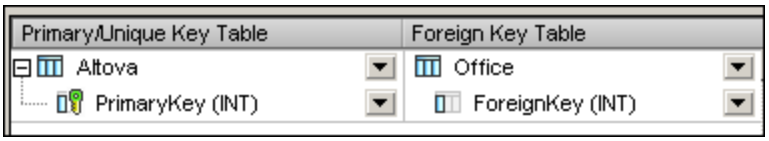
6. Click the left combo box and select the **Altova** table.



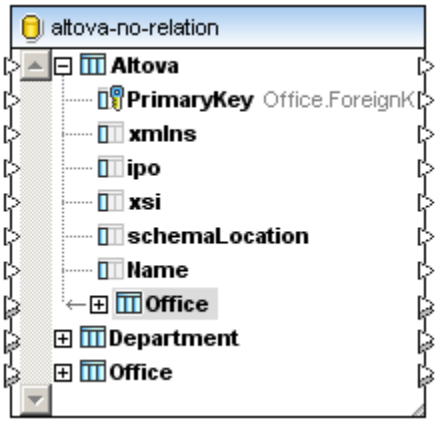
7. Click the "select column" combo box below it, and select the **Primary Key** entry.



8. Select the **Office** table and **ForeignKey** column for the Foreign Key table.



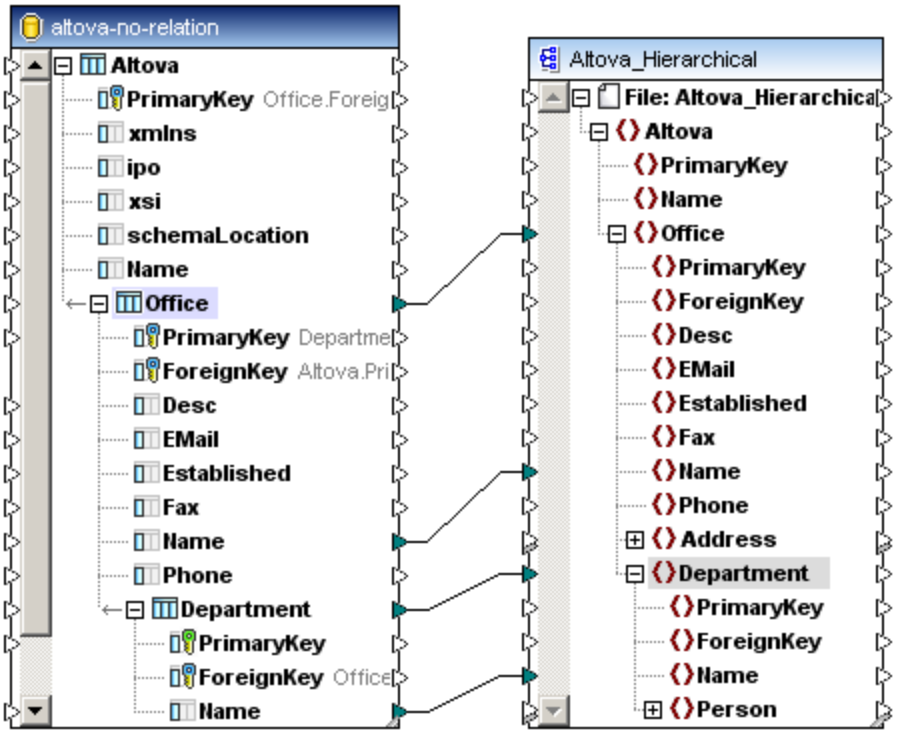
9. Click the OK button to complete the local relation defintion, then click the **Insert** button to insert the database into the mapping area.



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, 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"

```
<Altova xsi:noNamespaceSchemaLocation="C:/DOCUME~1/MYDOCU~1/Altov
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Department>
      <Name>Administration</Name>
    </Department>
    <Department>
      <Name>Marketing</Name>
    </Department>
    <Department>
      <Name>Engineering</Name>
    </Department>
    <Department>
      <Name>IT &amp;amp; Technical Support</Name>
    </Department>
  </Office>
  <Office>
    <Name>Nanonull Partners, Inc.</Name>
    <Department>
      <Name>Administration</Name>
```

15.7 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.

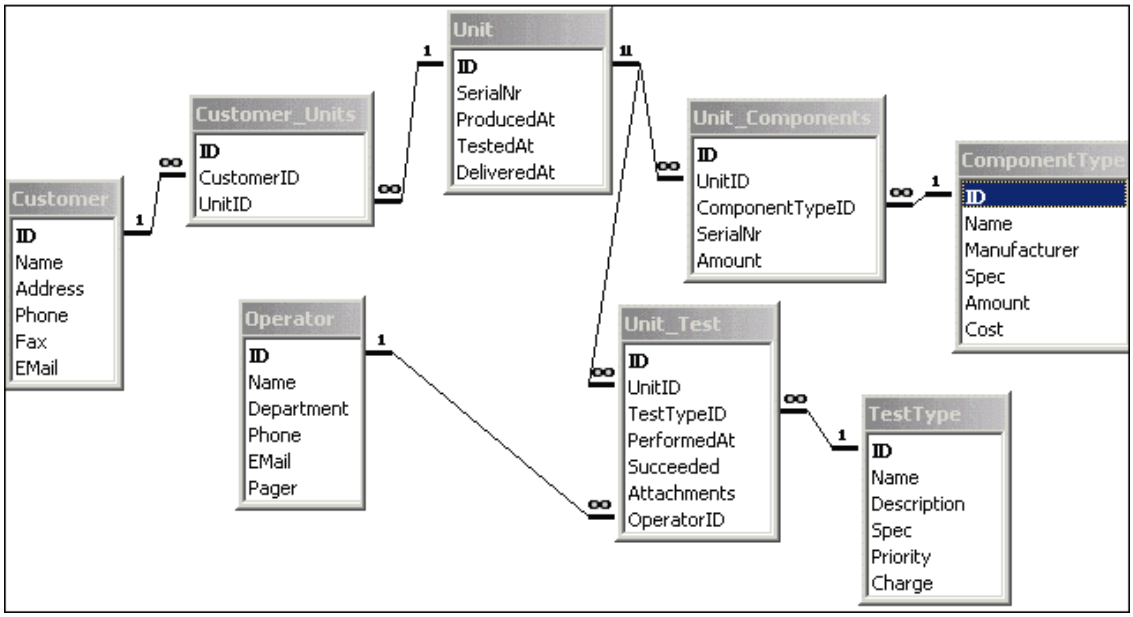
This may lead to inconveniently large tree structures if all tables are selected in a single database component. It is however possible to create multiple database components, of the same database, which only use/import those tables that are needed for the mapping process. This method also makes for 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:

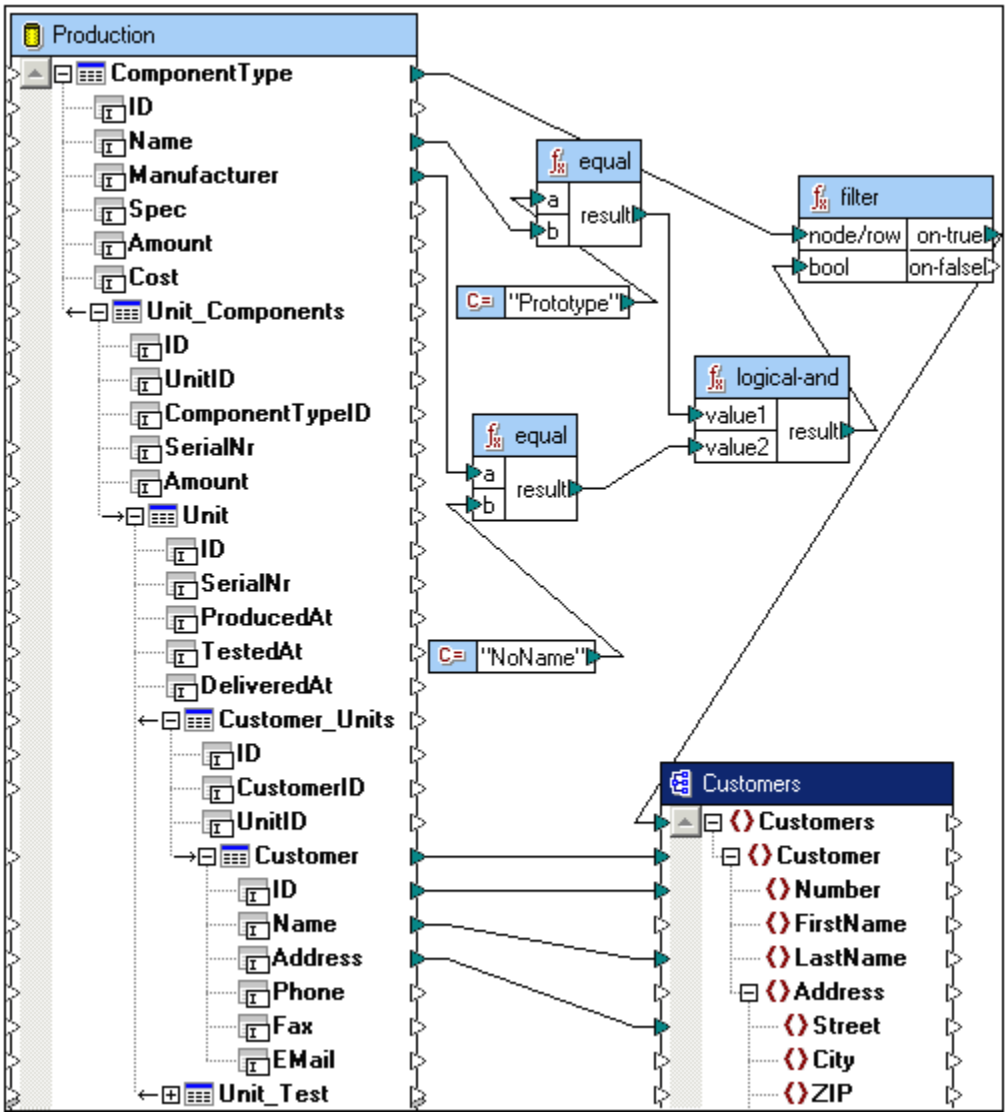


15.7.1 Complete database import

Option 1:
Import the complete database i.e. with **all** the tables it contains. The Product 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 out:

- the Component Name "Prototype" AND
- the Manufacturer "NoName", along with
- the related Customer ID and address data



15.7.2 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:

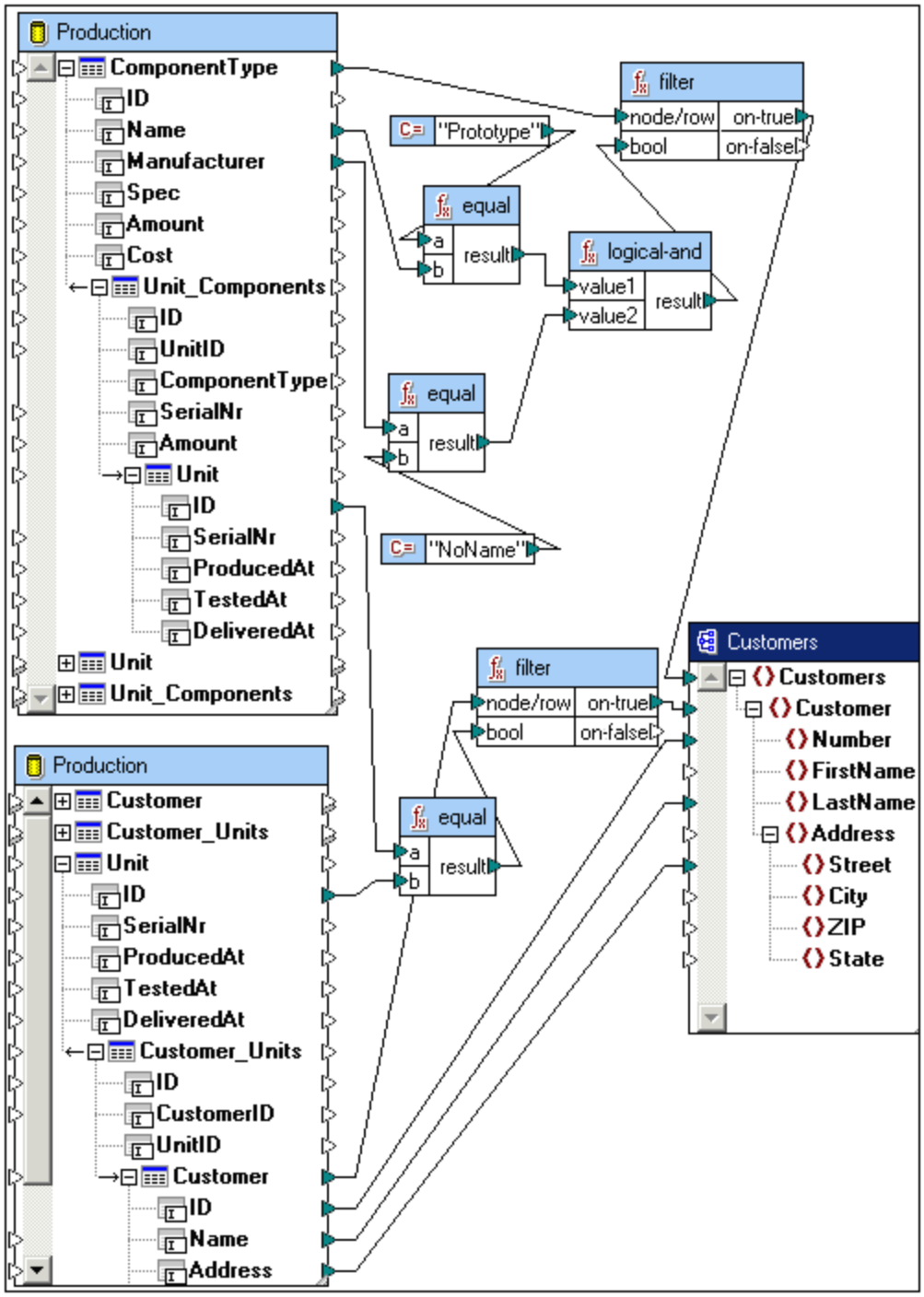
- ComponentType
- Unit_Components
- Unit

Component 2, insert:

- Unit
- Customer_Units
- Customer

Mapping process:

- filter out 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.


Production
ComponentType
ID
Name
Manufacturer
Spec
Amount
Cost
Unit_Components
ID
UnitID
ComponentType
SerialNr
Amount
Unit
ID
SerialNr
ProducedAt
TestedAt
DeliveredAt
Unit
Unit_Components
filter
node/row
on-true
bool
on-false
"Prototype"
equal
a
b
result
logical-and
value1
value2
result
equal
a
b
result
"NoName"
filter
node/row
on-true
bool
on-false
Customers
Customers
Customer
Number
FirstName
LastName
Address
Street
City
ZIP
State
Production
Customer
Customer_Units
Unit
ID
SerialNr
ProducedAt
TestedAt
DeliveredAt
Customer_Units
ID
CustomerID
UnitID
Customer
ID
Name
Address
equal
a
b
result

15.8 Database Filters and queries

When generating program code, MapForce optimizes database access by generating direct database queries where possible. The MAPFORCE **filter** component in conjunction with specific functions, is what makes this possible.

The filter component **generally** retrieves every record of a specific table and checks each to see if the filter condition is satisfied. If it is, the record is forwarded to the on-true/on-false parameters. This generates a select statement something like: **select "type" from "expense-item"**. This method is time consuming when using large databases, and an alternative method is used which transfers the workload to the database.

MapForce analyzes the mapping and checks for specific functions that support direct queries. Select statements are then generated for these functions, e.g. **select * from "expense-item" where type = "Travel".** Most of the work is now done by the database and the resulting dataset is then passed on for further processing.

The MapForce functions that support direct queries are show below.

Operators available for all database types:

MapForce function	Database function
"equal"	"="
"not-equal"	"<>"
"equal-or-greater"	">="
"equal-or-less"	"<="
"less"	"<"
"greater"	">"
"logical-or"	"or"
"logical-and"	"and"
"add"	"+"
"subtract"	"-"
"multiply"	"*"
"divide"	"/"
"modulus"	"%"

Functions for all database types:

"logical-not"	"not"

MS SQLServer specific functions:

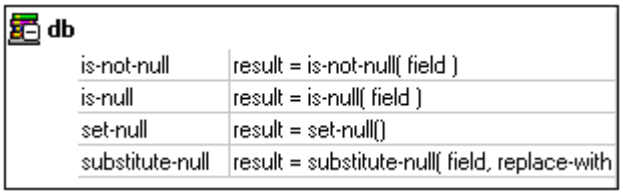
MapForce function	Database function
"floor"	"FLOOR()"
"ceiling"	"CEILING()"
"round"	"ROUND()"
"concat"	"+"
"substring"	"SUBSTRING()"
"contains"	"CHARINDEX()"
"string-length"	"LEN()"
"uppercase"	"UPPER()"
"lowercase"	"LOWER()"
"find-substring"	"CHARINDEX()"
"empty"	"IsEmpty()"

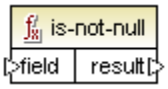
MS Access specific functions:

MapForce function	Database function
"round"	"Round()"
"concat"	"+"
"substring"	"Mid()"
"contains"	"InStr(1,..)"
"string-length"	"Len()"
"uppercase"	"UCase()"
"lowercase	"LCase()"
"find-substring"	"InStr(1,..)"
"empty"	"IsEmpty()"

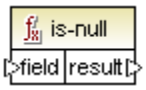
15.9 Database, Null processing functions

New null processing functions have been added to the DB language library.

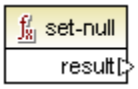




is-not-null
Returns false if the field is null, otherwise returns true.



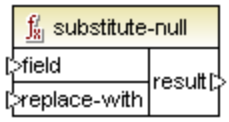
is-null
Returns true if the field is null, otherwise returns false.



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.

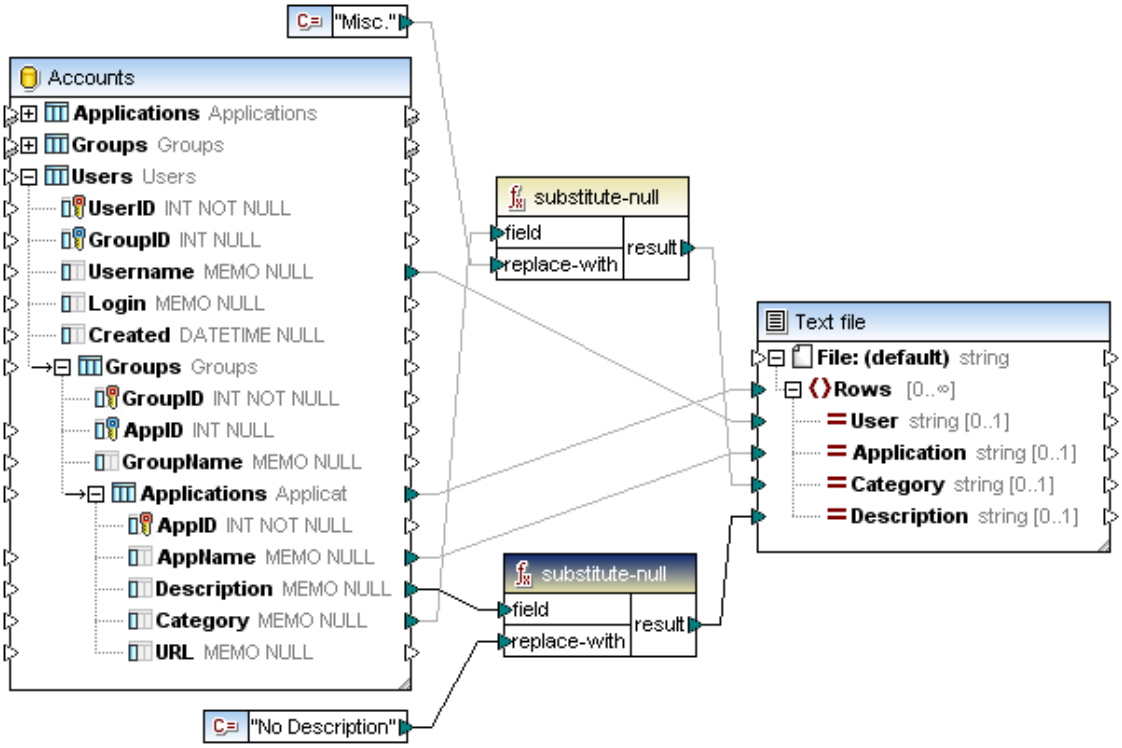
Please note:

- 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.
- Connecting set-null to a simpleType element will not create that element in the target component.
- Connecting set-null to a complexType element, as well as a table or row, is not allowed. A validaton error occurs when this is done.



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.

15.10 SQL WHERE Component / condition

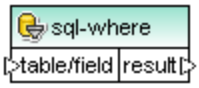
MapForce allows you to filter out database data conditionally using the SQL WHERE component.

The SQL WHERE component is comprised of two parts:

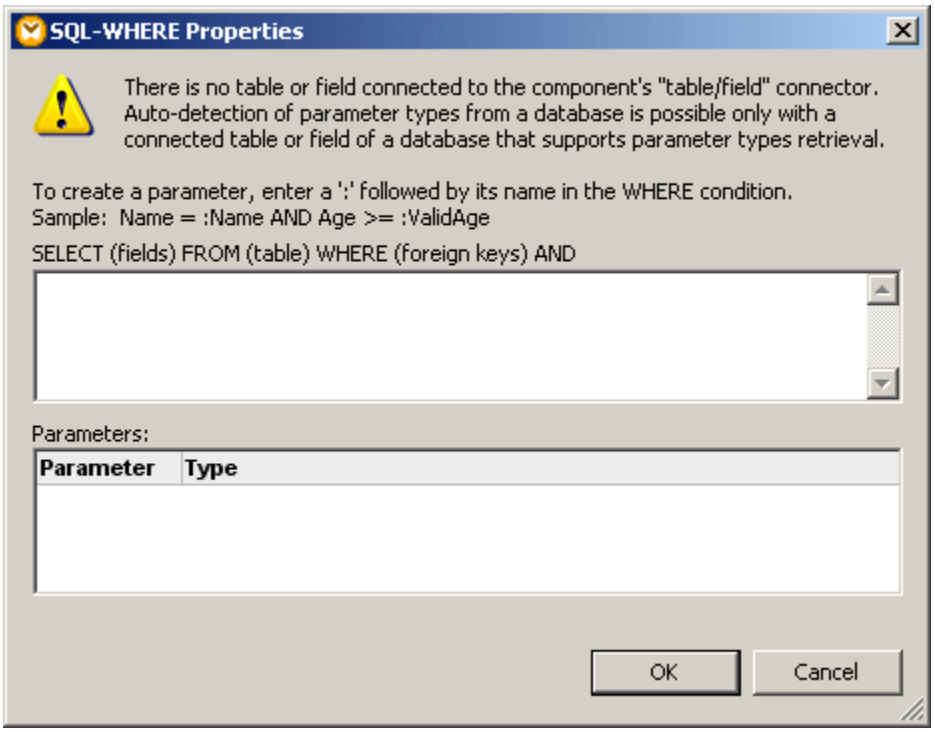
- The **SELECT** statement that is automatically generated when you connect to a database table or field.
- The **WHERE** clause that you manually enter in the SELECT text box. Note that the foreign keys are automatically included in the select statement.

To insert an SQL WHERE component:

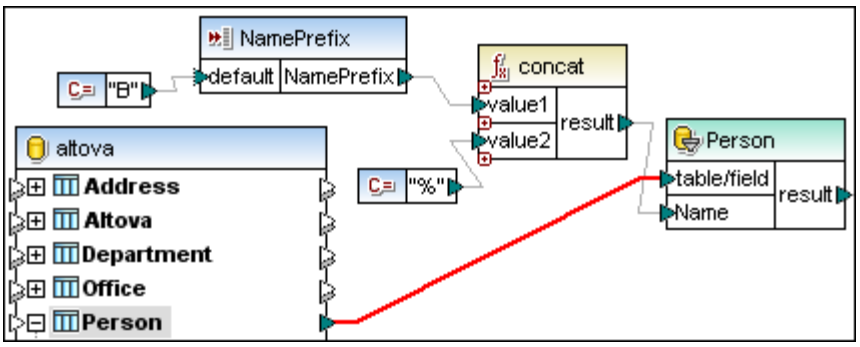
1. Click the SQL WHERE icon  in the icon bar to insert it.



2. Double click the SQL WHERE component to see its contents, shown below, then click Cancel to close it.



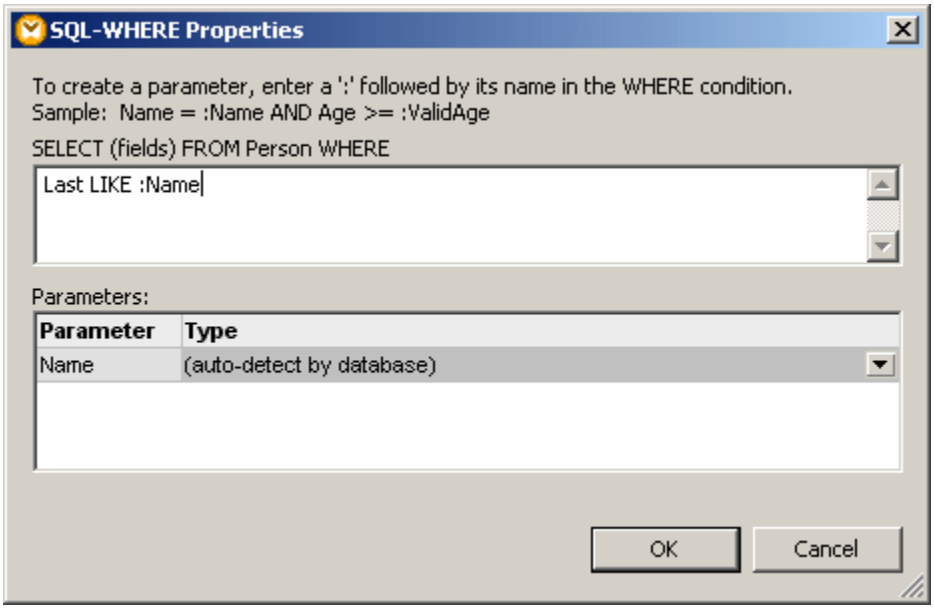
3. Make a connection between the source table/field that you want to query and the **table/field** input icon of the SQL WHERE component (e.g. Person table to table/field).



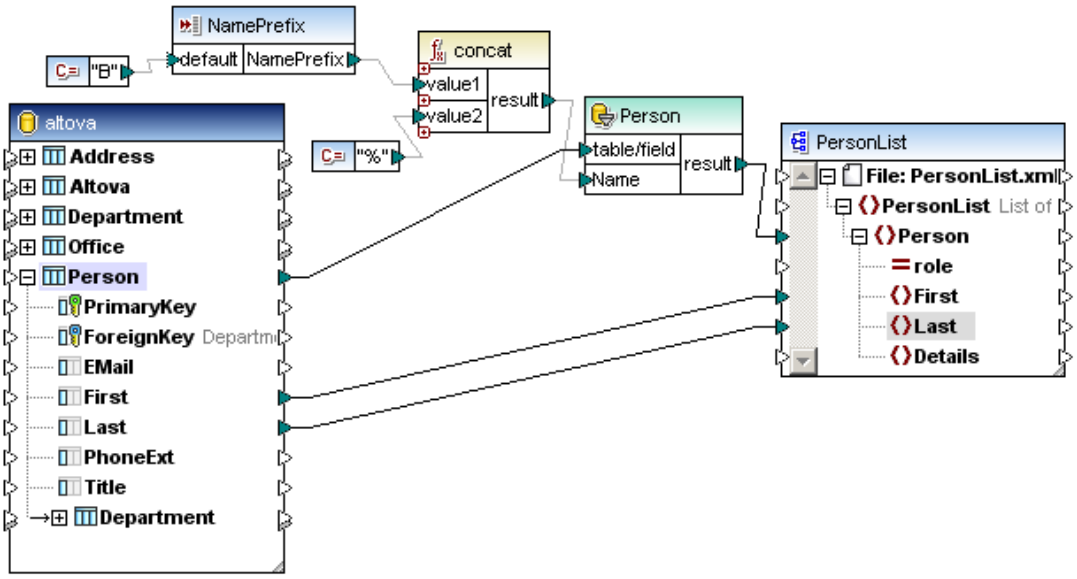
4. Double click the SQL WHERE component to write the WHERE query in the top text box.
   The SELECT statement, just above the text box, is automatically generated for you when a connection is made from a table, or field, to the **table/field** input icon of the SQL WHERE component.

   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 "%" 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).



15.10.1 SQL WHERE 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 e.g. IBM DB2.

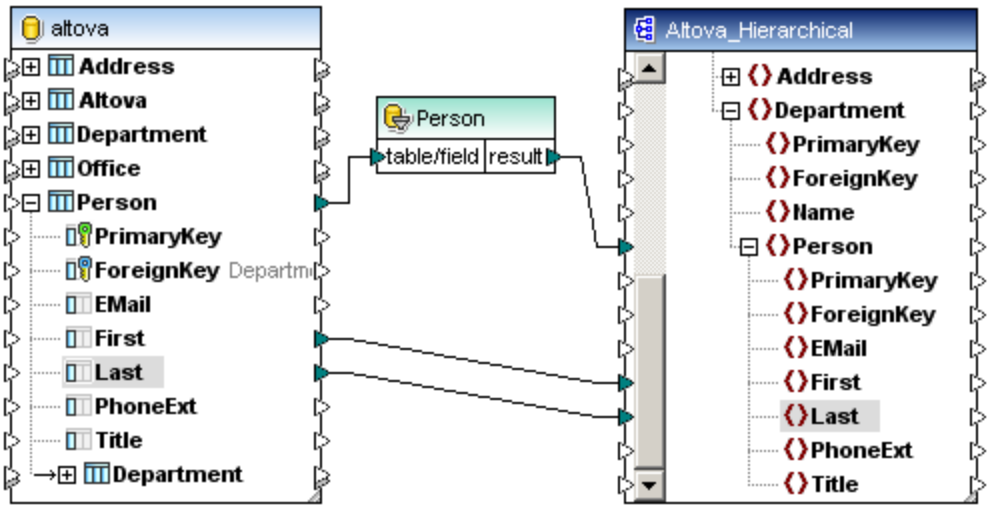
E.g. **xmlexists**('$c/Client/Address[zip>"55116"]' passing USER.CLIENTS. CONTACTINFO AS "c")

Select from Person WHERE
First > "C" AND Last > "C"

Retrieves those records where the contents of First and Last are greater than the letter C. Retrieves all names from Callaby onwards.

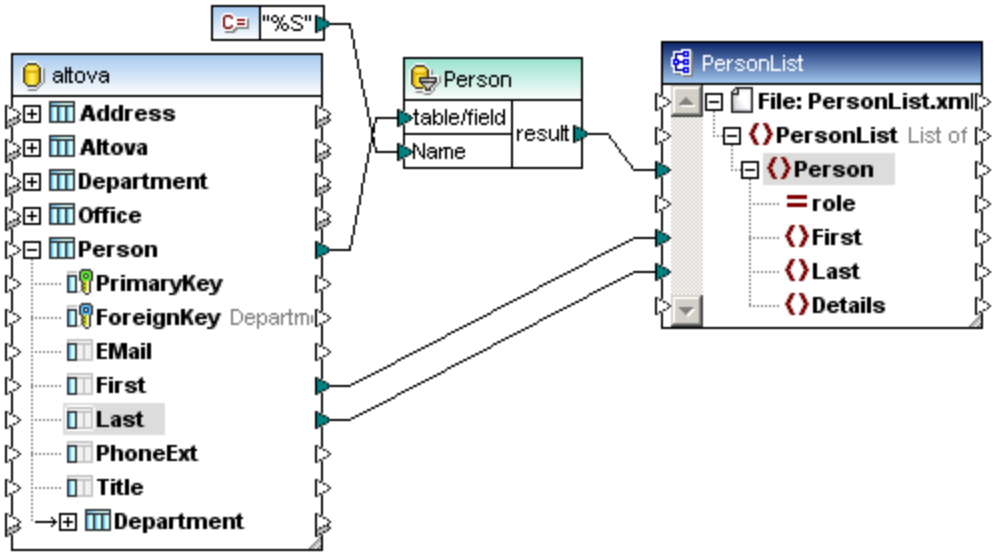
Note how the connectors are placed:

- The connector to the **table** 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 data of the first and last fields are connected to sub-items in the target component for them to appear in the result.



Select from Person WHERE
Last LIKE :Name

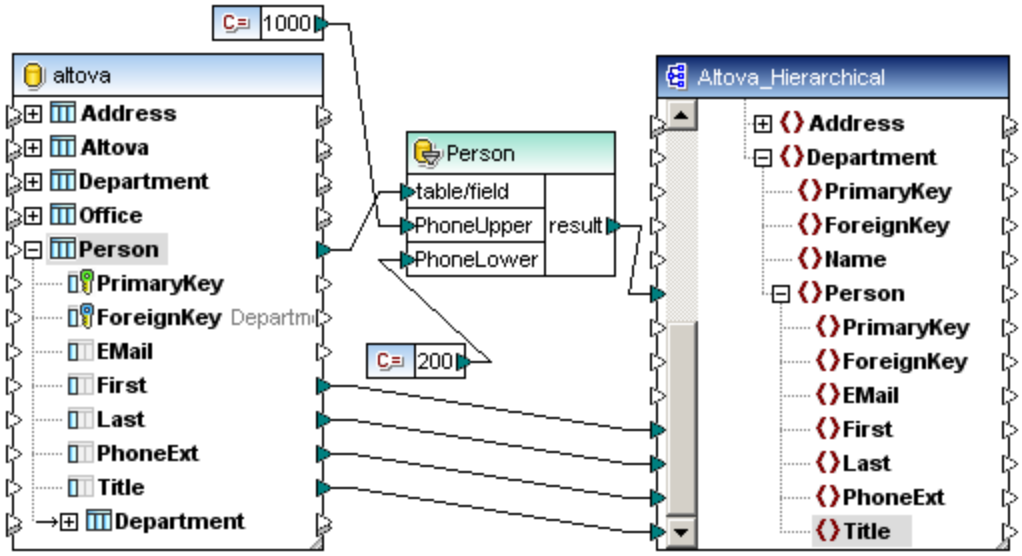
This WHERE statement creates a parameter **Name** which appears as a parameter in the SQL WHERE component. In this case it is used to search for a pattern in the column "Last". The wildcard % denotes any number of characters.



The Constant component supplies the values needed to search for all Last names ending in "S", i.e. %S.

Select from Person WHERE
PhoneExt < :PhoneUpper and PhoneExt > :PhoneLower

This WHERE statement 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.



Please note:
The WHERE clause in this example could also be rephrased using the BETWEEN operator:

Select from Person WHERE
PhoneExt BETWEEN :PhoneUpper and :PhoneLower

15.11 Mapping XML data to / from databases generically

MapForce supports the mapping of XML data to / from several databases, including MS Access and IBM DB2 version 9. This section describes the mapping **xml2access.mfd** file available in the ...**Tutorial** folder.

XML documents can be mapped to/from all **string**, **varchar** and **memo** fields of sufficient length. Please note that the character encoding of such documents is always that of the underlying string field in the particular database. If the field does not store text as Unicode, some characters cannot be represented. Full support for all XML encodings is only possible in native IBM DB2 version 9 XML fields.

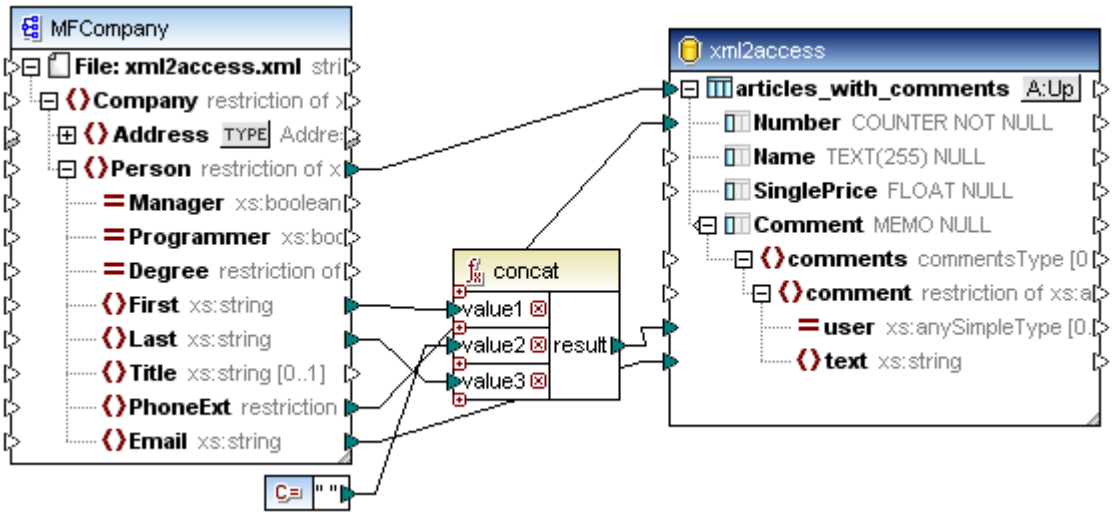
For the XML nodes to appear in the database component, which allows mapping to and from the XML items, a schema must be **assigned**. Once this is done the mapping process proceeds as usual.

MapForce currently supports the assigning of **one** XML Schema per database column, and the selection of a **single** "root" element of that schema.

Open the mapping file **xml2access.mfd** file in the ...**Tutorial** folder.

What this mapping example does:

- Updates the XML data in the **target** database in the **Comment** column, if:
- The Number content of the PhoneExt and Number fields are identical.



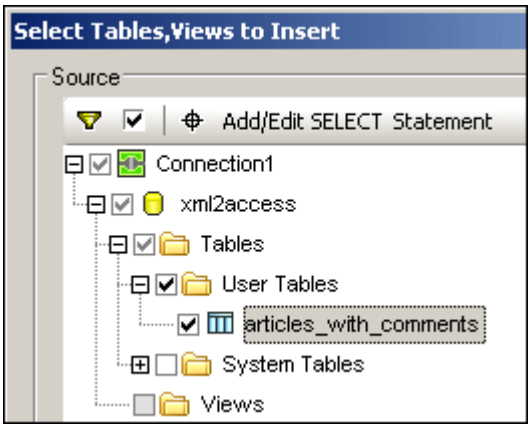
To insert the data source component:

1. Click the **Insert XML Schema/File** icon, and select the **MFCompany.xsd** schema.
2. Click **Browse..** when the prompt for a **sample XML** file appears, and select **xml2access.xml**

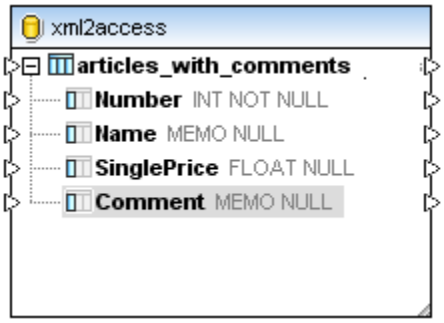
To insert the database target and map data to it:

1. Click the **Insert Database** icon in the icon bar, click MS Access (ADO), then click Next.

2. Click the Browse button and select the **xml2access.mdb** file in the ...**Tutorial** folder, then click Next.

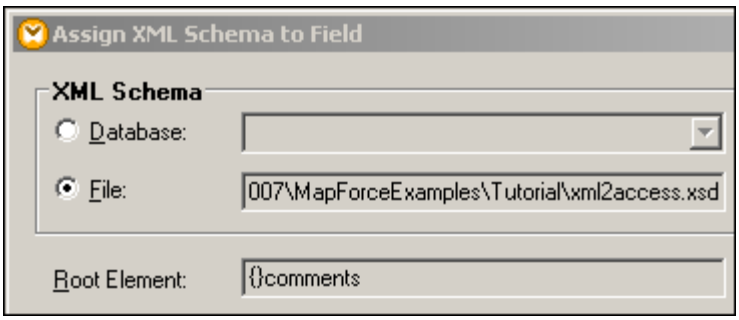


3. Click the **articles_with_comments** table check box then click OK.

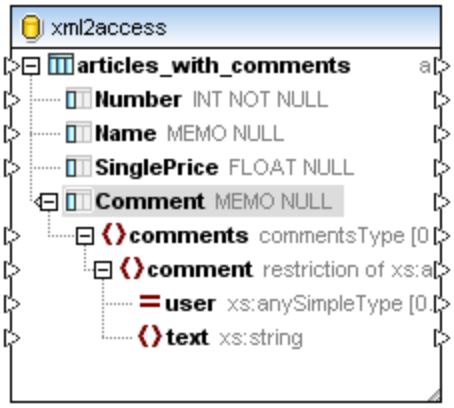


To assign an XML schema to a database:

1. Right click the Comment item/column in the database target and select **Assign XML Schema to Field...**
3. Click the **File** radio button if necessary, select **xml2access.xsd** and click OK.

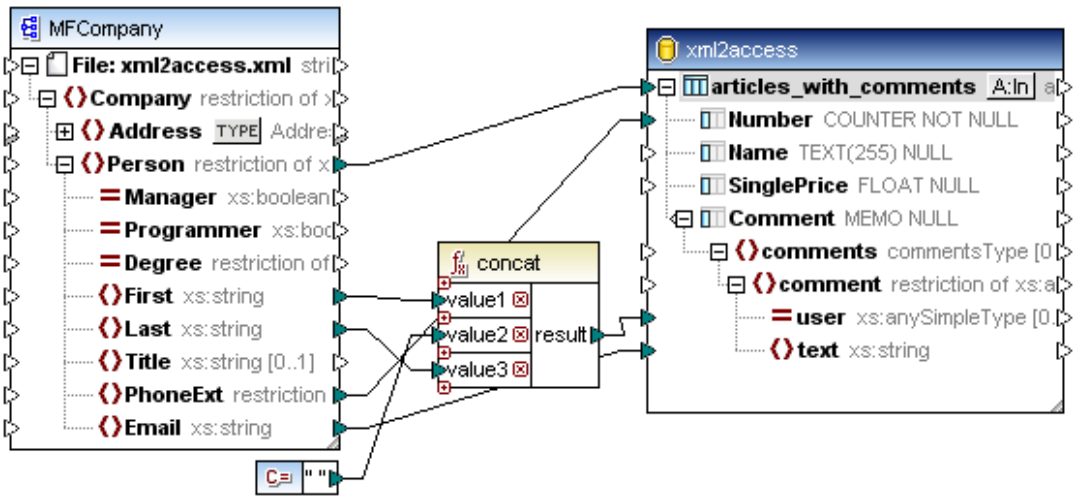


4. Expand the **Comment** item to be able to create connectors to the respective items.



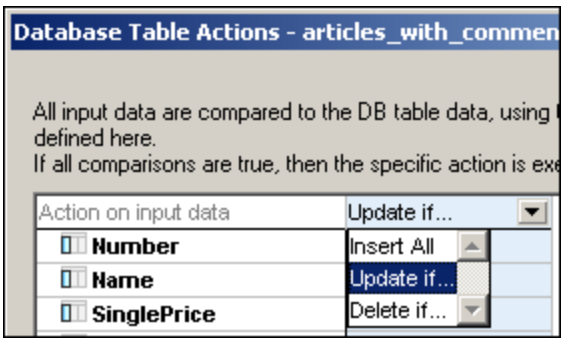
The XML schema node tree containing all mappable items appears below the Comment item/column. You are now ready to map XML data to the database.

5. Create the mapping connectors as seen in the top screenshot.

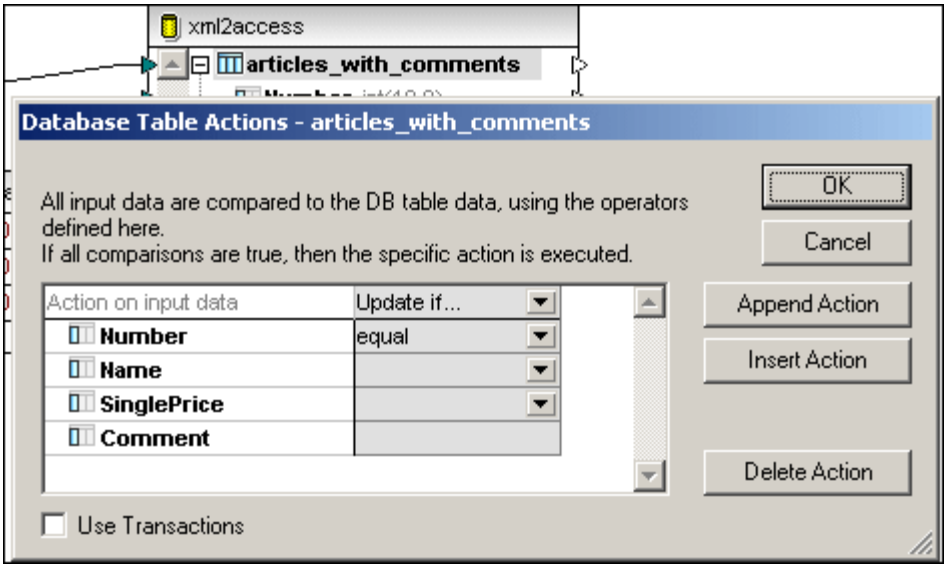


To define the table actions and map the data:

1. Right click the articles_with_comments table item and select **Database Table actions**.
2. Click the **Insert All** combo box and change it to **Update if...**



3. Click the combo box next to **Number** and select **equal,** then click OK.



4. Click the **Output** tab, then the Word Wrap icon  to see more of the SQL script.
5. Click the "Run SQL-script" icon  to insert the XML data into the database.

```
UPDATE [articles_with_comments]
   SET [Comment]='<?xml version="1.0"?><comments xsi:noNamespaceSchemaLocation="C:\Program
Files\Altova\MapForce2007\MapForceExamples\Tutorial\xml2access.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><comment user="Vernon
Callaby"><text>v.callaby@nanonull.com</text></comment></comments>' WHERE (
[articles_with_comments].[Number]=1)
-->>> OK. 1 row(s).

UPDATE [articles_with_comments]
   SET [Comment]='<?xml version="1.0"?><comments xsi:noNamespaceSchemaLocation="C:\Program
Files\Altova\MapForce2007\MapForceExamples\Tutorial\xml2access.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><comment user="Frank
Further"><text>f.further@nanonull.com</text></comment></comments>' WHERE (
[articles_with_comments].[Number]=2)
-->>> OK. 1 row(s).

UPDATE [articles_with_comments]
   SET [Comment]='<?xml version="1.0"?><comments xsi:noNamespaceSchemaLocation="C:\Program
Files\Altova\MapForce2007\MapForceExamples\Tutorial\xml2access.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><comment user="Loby
Matise"><text>l.matise@nanonull.com</text></comment></comments>' WHERE ([articles_with_comme
[Number]=3)
-->>> OK. 1 row(s).
```

15.12 IBM DB2 - Mapping XML data to / from databases

MapForce supports the mapping of XML data to / from IBM DB2 version 9 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. An analogous mapping example of mapping XML data to a MS Access database, **xml2access.mfd**, is available in its entirety.

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.

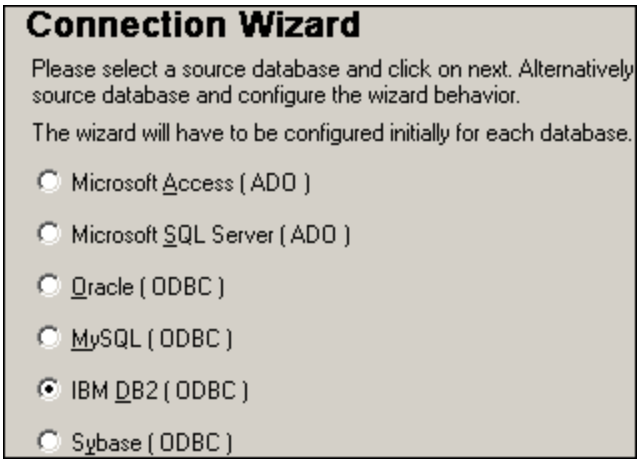
MapForce currently supports the assigning of **one** XML Schema per database column, and the selection of a **single** "root" element of that schema.

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.

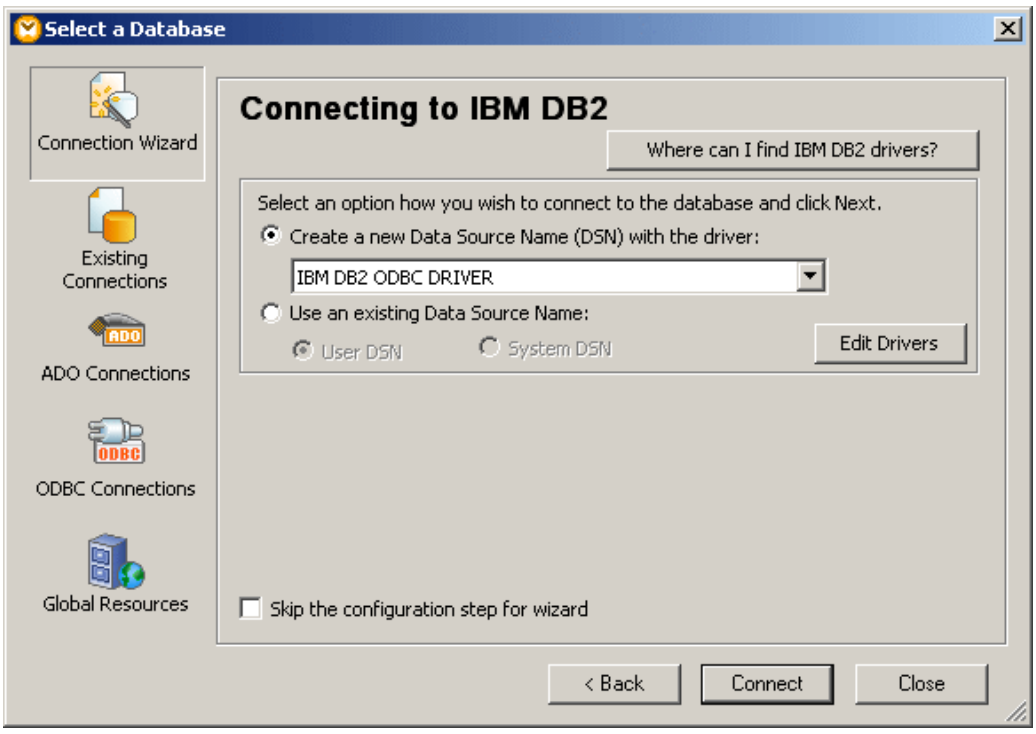
When adding an ODBC data source for iSeries Access ODBC driver, the "iSeries Access for Windows ODBC Setup" dialog box is opened. Select the "Performance" tab, click the "Advanced" button and **uncheck** the "Allow query timeout" check box option.

Configuring and inserting an IBM DB2 database:

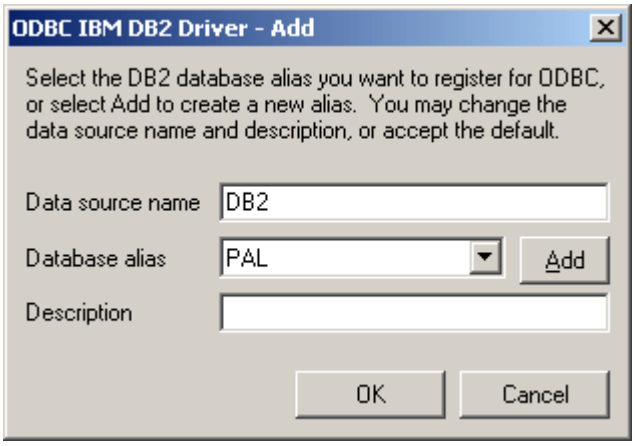
1. Click one of the icons in the title bar: Java, C#, C++, or BUILTIN.
2. Click the **Insert Database** icon  in the icon bar.



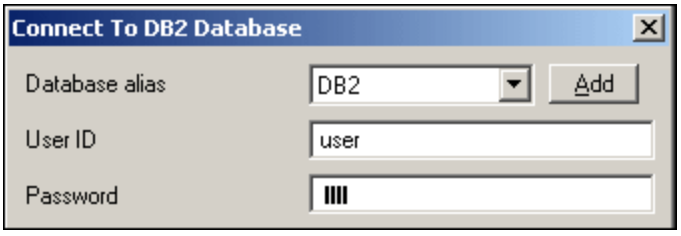
3. Click the IBM DB2 (ODBC) radio button and click Next.
4. Select a driver from the Driver list, by clicking the appropriate check box.



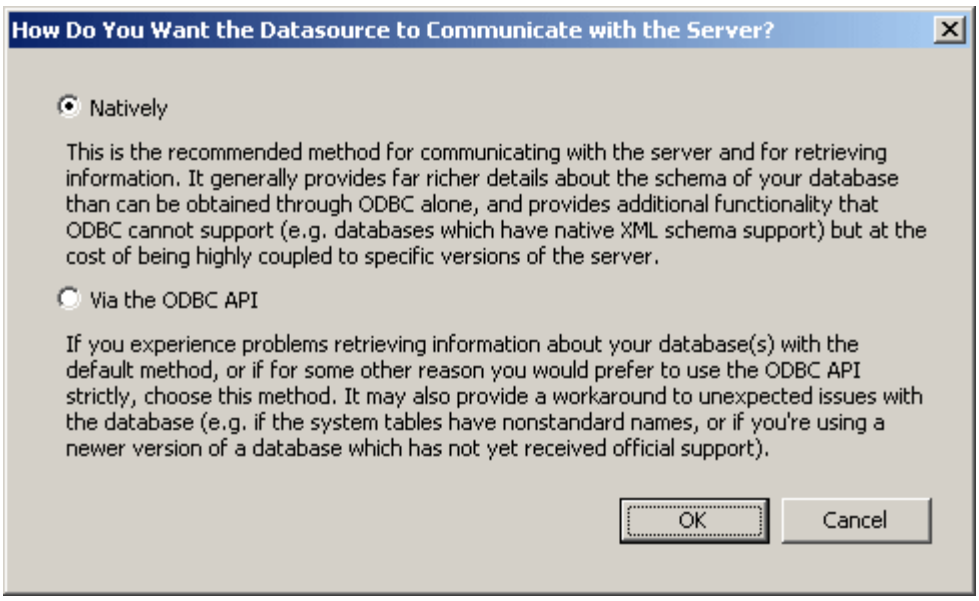
4. Click **Next** to define a new Data Source Name (DSN).
5. Enter the Data source name, select (or Add) the Database alias, then click OK.



6. Enter the login data and click OK to connect to the database.

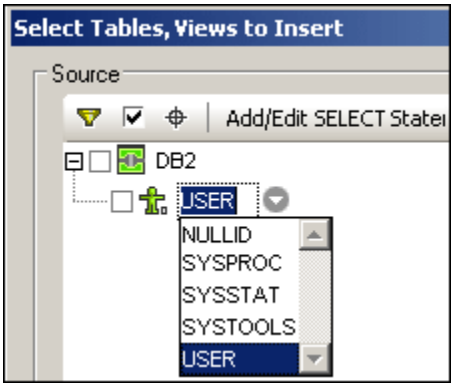


Entering the required login info and clicking OK opens a further dialog box allowing you to choose how the datasource will communicate with the server; natively or via the ODBC API.

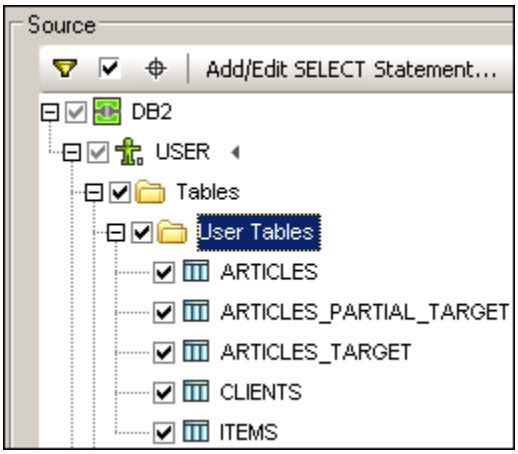


The recommended method is "Natively", then click the OK button.
This opens the Select Tables dialog box.

7. Click the database schema icon and select the correct database schema e.g. USER.



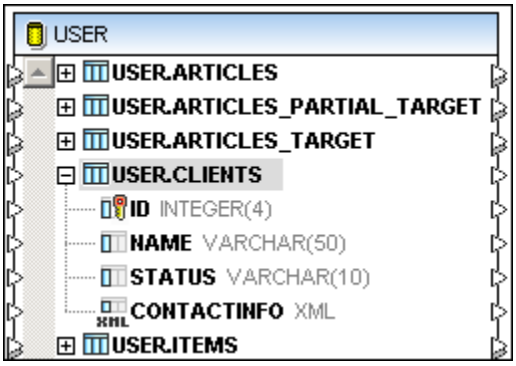
All USER tables are now visible.



8. Click the check box next to "User Tables" to select all child tables, then click OK to

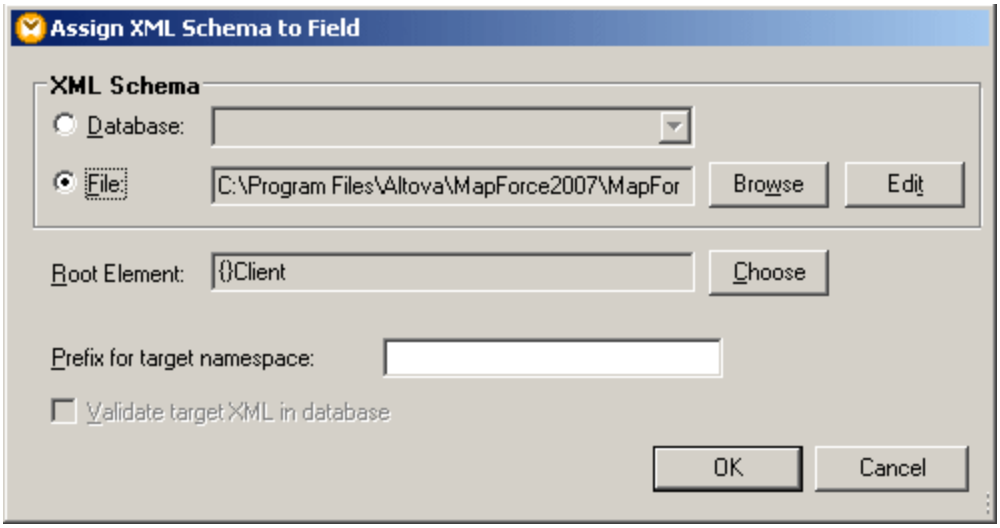
insert the database component.

9. Click the expand icon next to the USER.CLIENTS table to see its contents. Note that the CONTACTINFO column is of type **XML**.

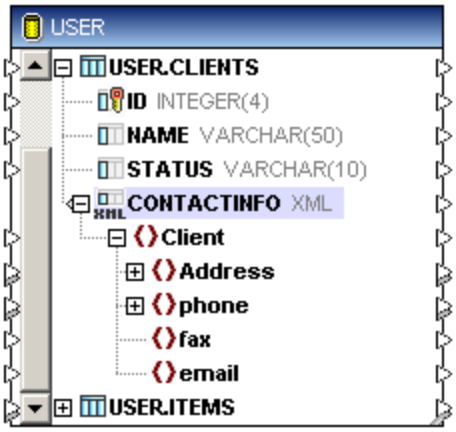


Assigning an XML Schema to an XML file:

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.

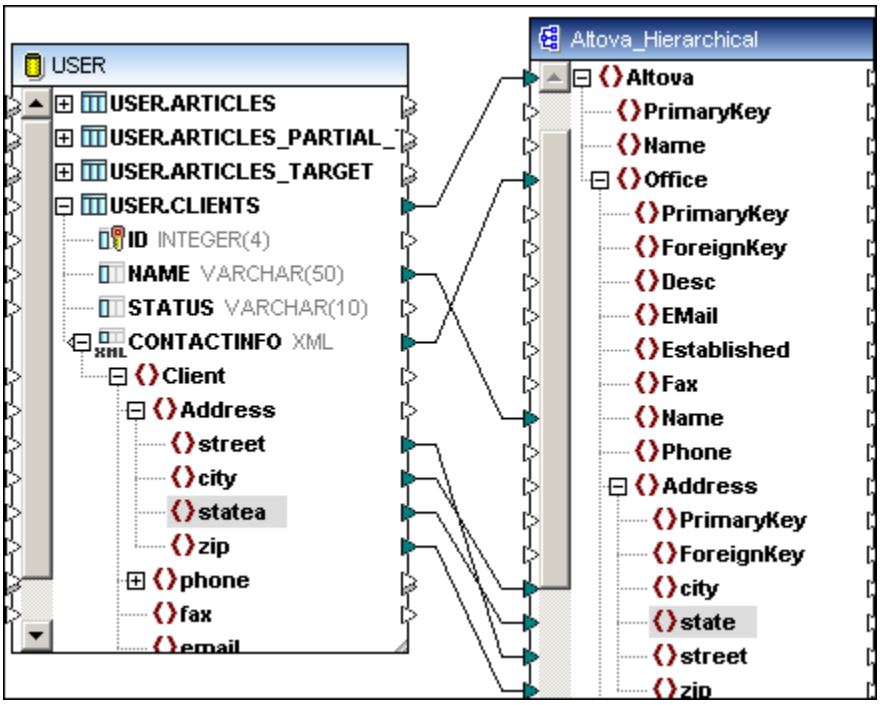


3. Choose the Root element of the schema that is to appear in the component e.g. Client, and click OK to confirm.

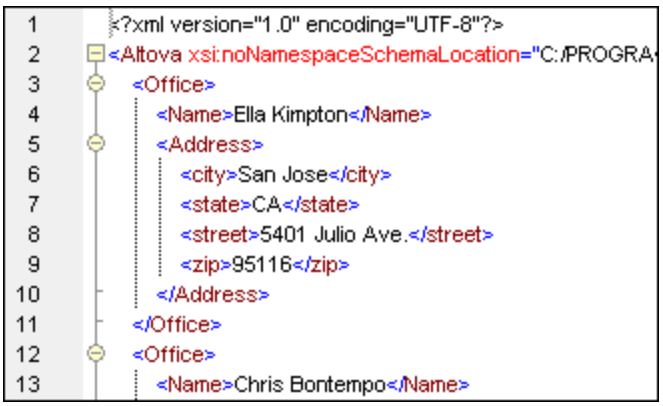


The "Client" item appears below CONTACTINFO; click the expand icons to see the schema structure.

4. Map connectors from the schema items to the target component, which is an XML document in this case.

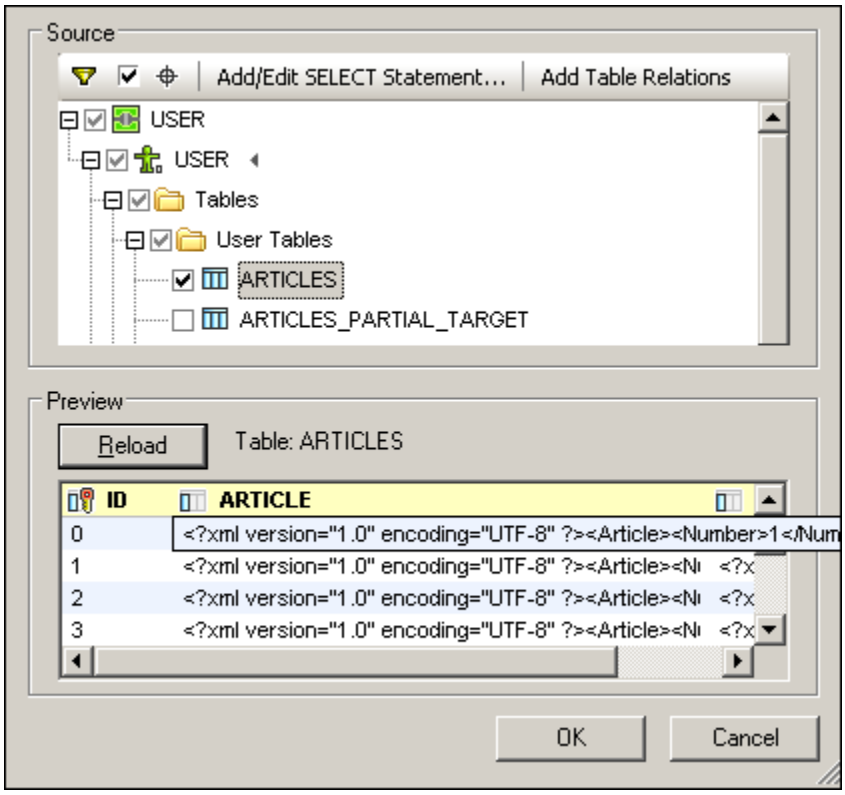


5. Click the Output button to see the result of the mapping.



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.



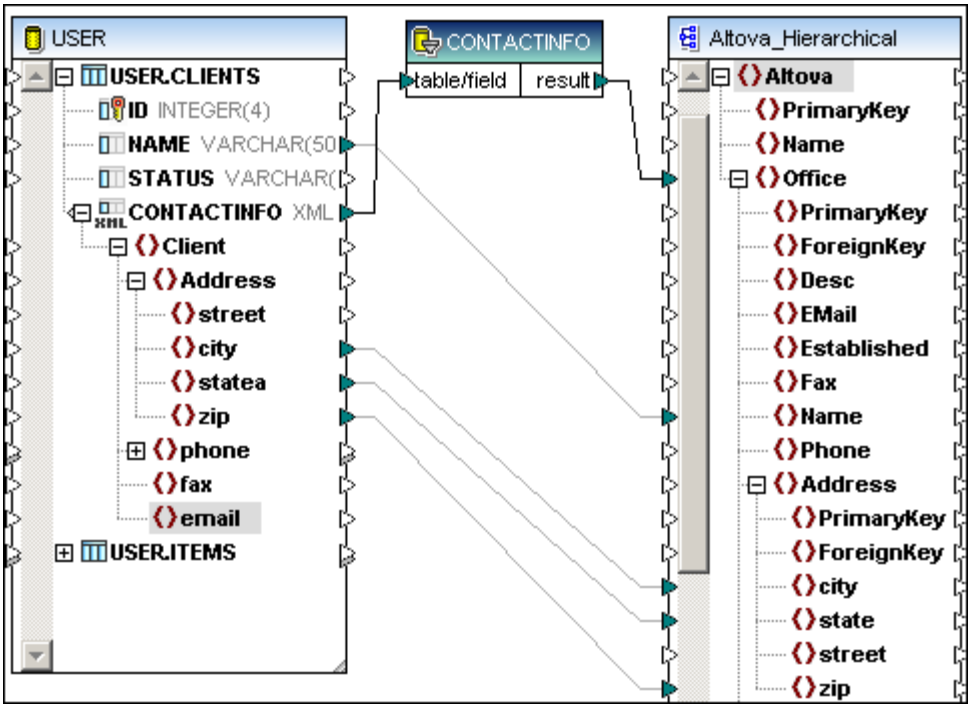
15.12.1 Querying and mapping XML data in IBM DB2

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.

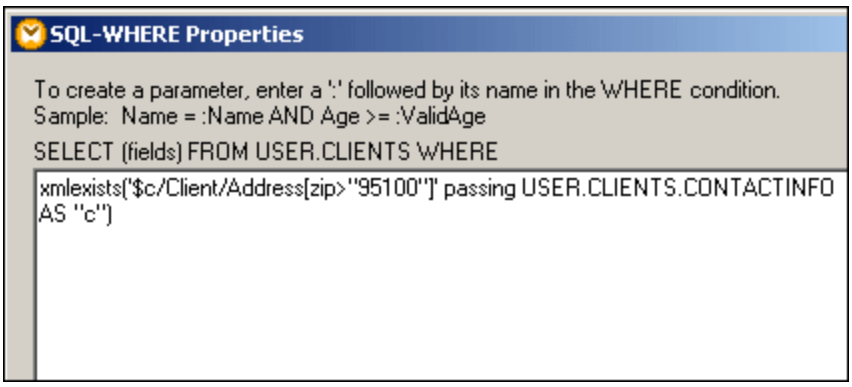
Having inserted the DB2 database and assigned the XML schema to the CONTACTINFO item:

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.

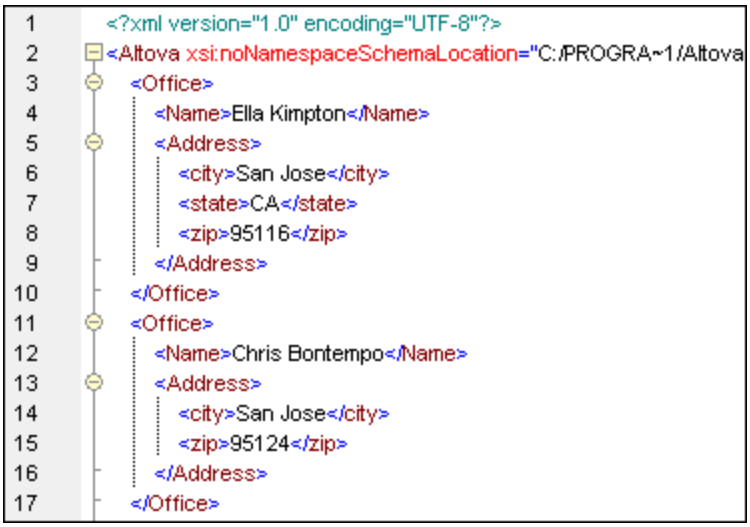
4. Double click the CONTACTINFO component to create the query.
5. Enter the SQL/XML WHERE query e.g. **xmlexists('$c/Client/Address[zip>"95100"]' passing USER.CLIENTS.CONTACTINFO AS "c")**



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.



15.12.2 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.

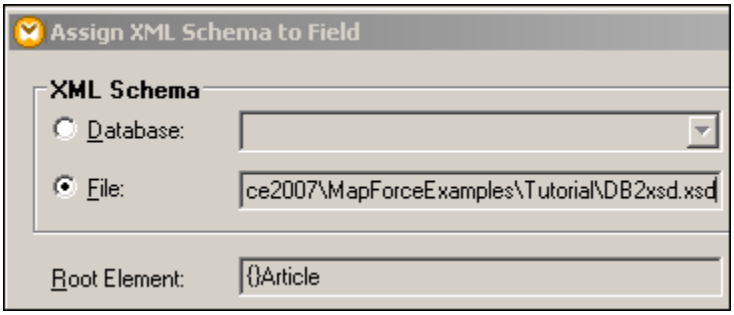
Please note that you can also query databases directly using the Database Query tab, please see Querying databases directly - Database Query tab for more information.

To insert the data source component:

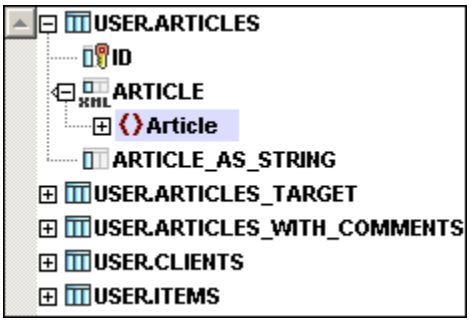
1. Click the **Insert XML Schema/File** icon  , and select the **DB2asTarget.xsd** schema.
2. Click Browse when the prompt for a sample XML file appears, and select **DB2asTarget.xml**
3. Select **Articles** as the root element and expand all items.

To insert the database target and map data to it:

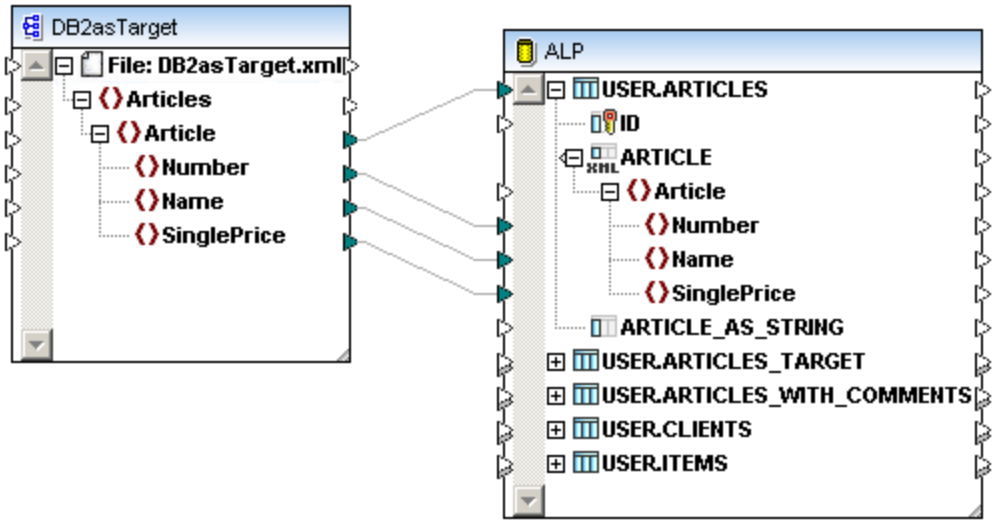
1. Using the method discussed in insert the IBM DB2 database to insert the database.
2. Right click the ARTICLE item/column in the database target and select **Assign XML Schema to Field...**
3. Click the **File** radio button, select **DB2xsd.xsd** and click OK.



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.

```
1   /*
2   The following SQL statements were executed during "Generate output" function.
3   Every single result is written right to the "-->>>" string.
4   These statements are only for preview and may not be executed in another SQL query tool!
5   */
6
7   INSERT INTO "USER"."ARTICLES" ("ARTICLE")
8       VALUES ('<?xml version="1.0" encoding="UTF-8"?><Article
9   xsi:noNamespaceSchemaLocation="C:\Program
10  Files\Altova\MapForce2007\MapForceExamples\Tutorial\DB2xsd.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Number>11</Number><Name>
    ers</Name><SinglePrice>99</SinglePrice></Article>')
    -->>> OK. 1 row(s).
```

The output window now shows if the commands were executed successfully.

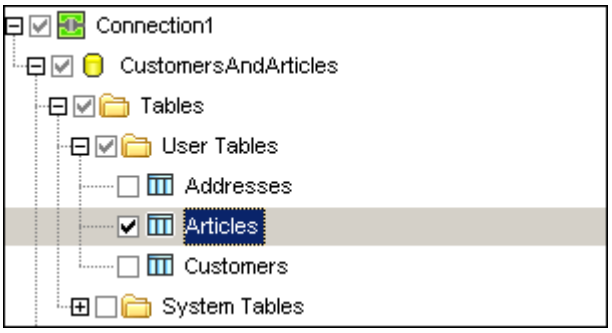
15.12.3 Mapping data - database to database

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:

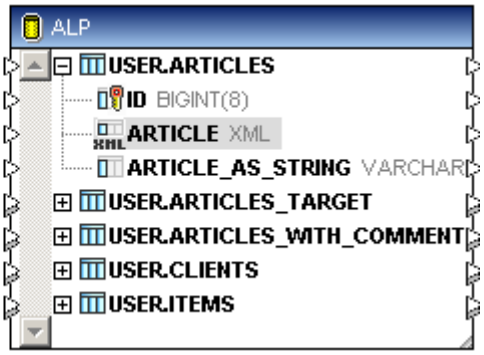
1. Click the **Insert Database** icon  , and select the **CustomersAndArticles.mdb** database from the ...\MapForceExamples folder.
2. Select the **Articles** table and click OK to insert.



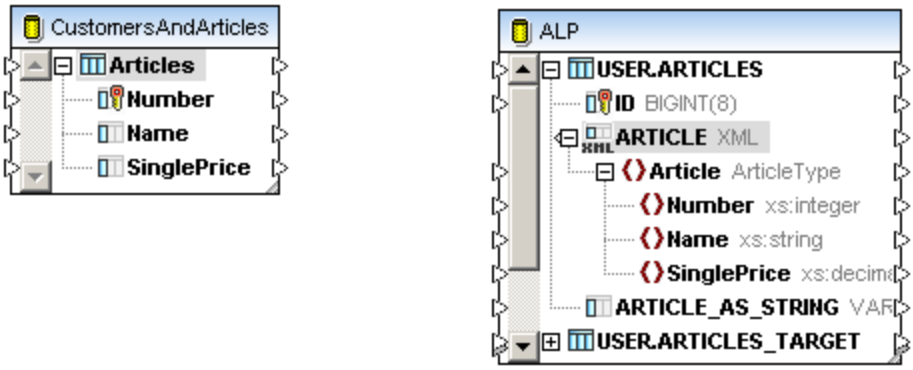
The database table is now visible as a database component.

To insert the IBM DB2 target database and map data to it:

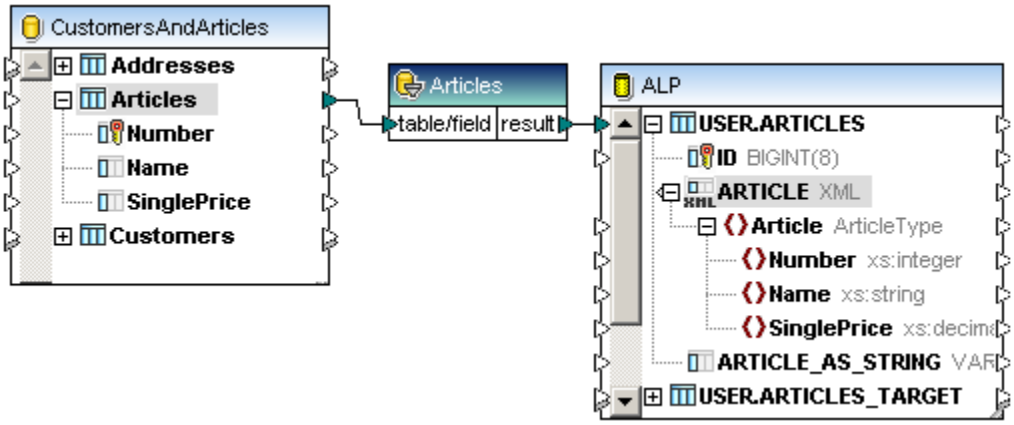
1. Using the method discussed in insert the IBM DB2 database to insert the database.



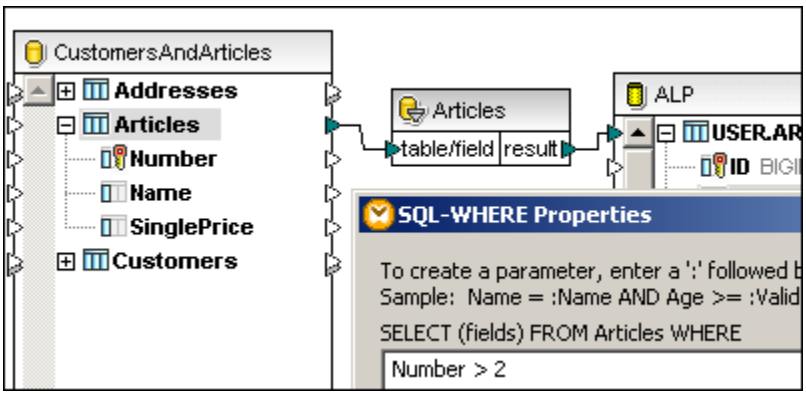
2. Right click the ARTICLE item/column in the database target and select **Assign XML Schema to Field...**
3. Click the **File** radio button, select the **DB2xsd.xsd** schema file, then click OK.



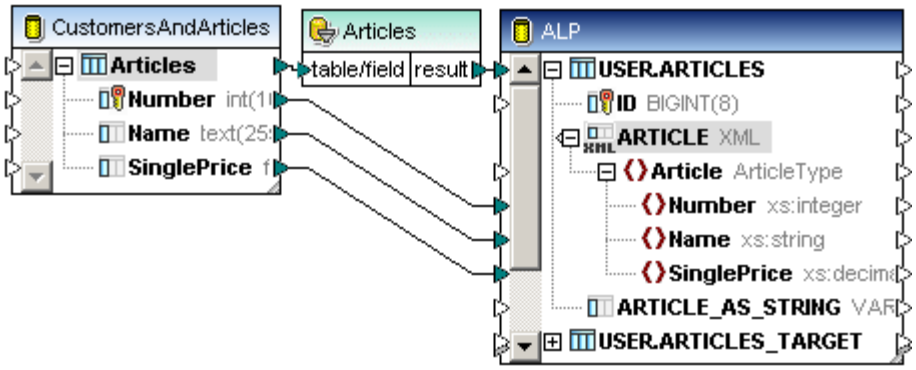
4. Click the SQL WHERE icon  in the icon bar to insert it.
5. Connect the **Articles** item, of the database source, to the **table** item component.
6. Connect the **result** item to the USER.ARTICLES item in the target component.



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>')
```

15.13 SQL Server 2005 - Mapping XML data

MapForce supports the mapping of XML data to/from SQL Server 2005 version (and higher) databases. The examples in this section assume that you have access to an SQL Server 2005 database.

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.

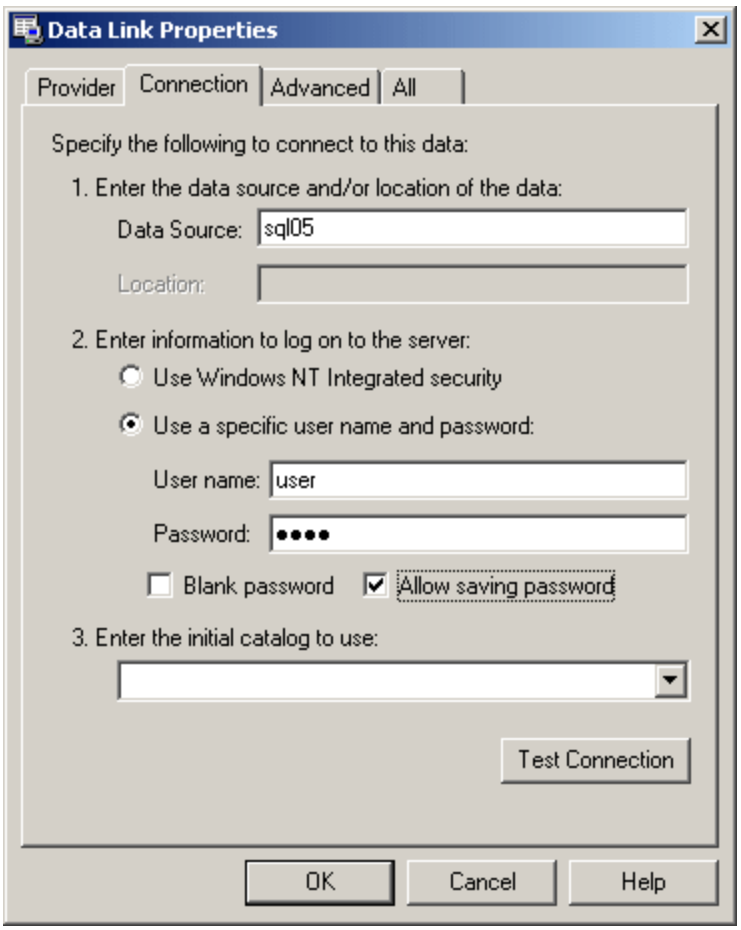
MapForce currently supports the assigning of **one** XML Schema per database column, and the selection of a **single** "root" element of that schema.

Configuring and inserting an SQL Server 2005 database:

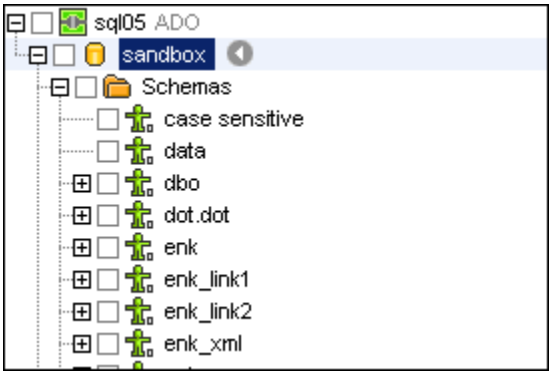
1. Click one of the icons in the title bar: Java, C#, C++, or BUILTIN.
2. Click the **Insert Database** icon  in the icon bar.

3. Select the Microsoft SQL Server (ADO) radio button and click Next.
4. Select a driver from the driver list by clicking the drop-down combo box arrow.

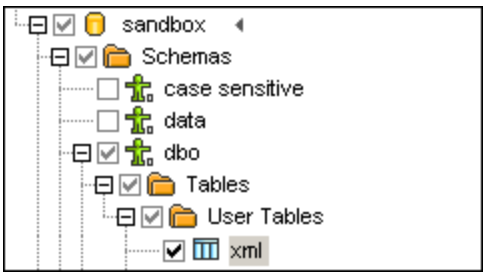
5. Click Next to define the Data Link Properties.
6. Fill in the Data Source, User name, and Password fields. Make sure to check/activate the Allow saving password check box.



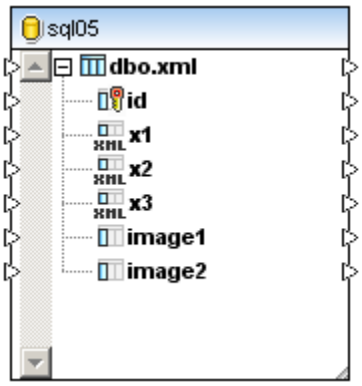
7. Click the "Test Connection" button, then the OK button to insert the component.
8. Click the database schema icon and select the correct database schema e.g. sandbox from the list box.



9. Click a check box to select the tables you want to insert, e.g. xml, then click OK to insert the database component.
   Note: you can preview the table content by clicking the **Preview** button, please see below.

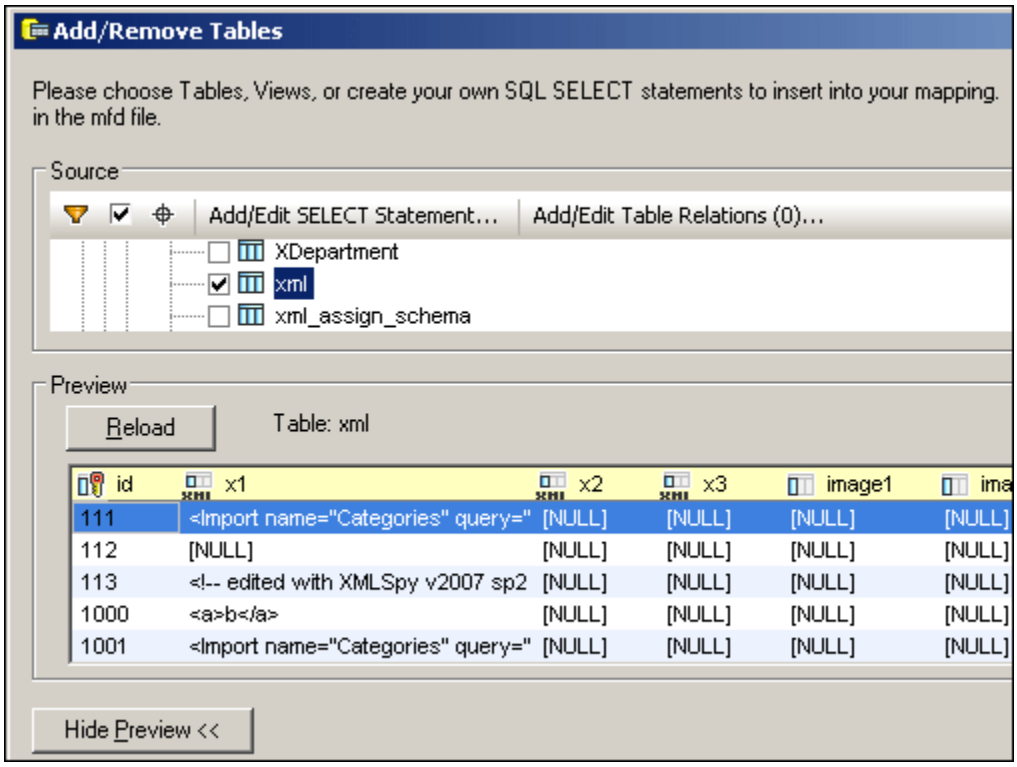


The xml table has now been inserted as a database component into MapForce.



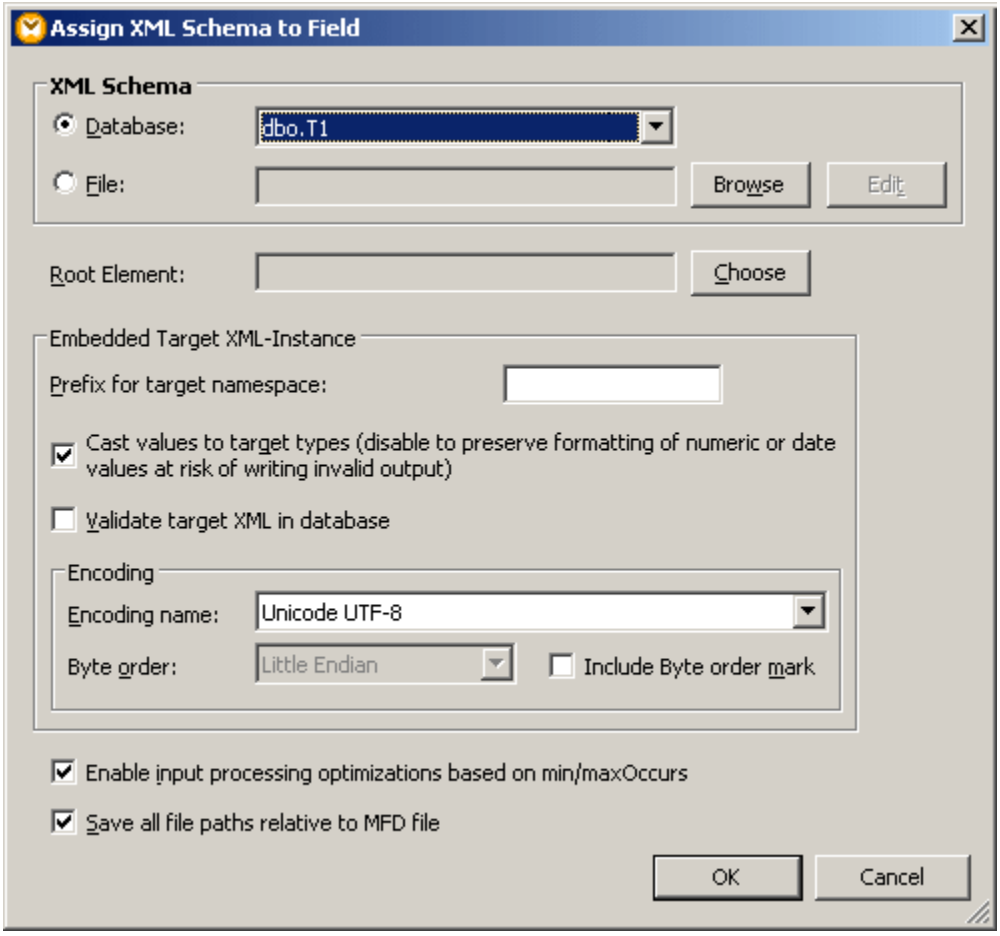
Note that the **x1**, **x2**, and **x3** columns are all of type **XML**.

Preview of the xml table:



Assigning an XML Schema to an XML file:

1. Right click the column you want to assign the schema to, e.g. **x1**, 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.



3. Click OK to assign the new schema file.

15.14 Querying databases directly - Database Query tab

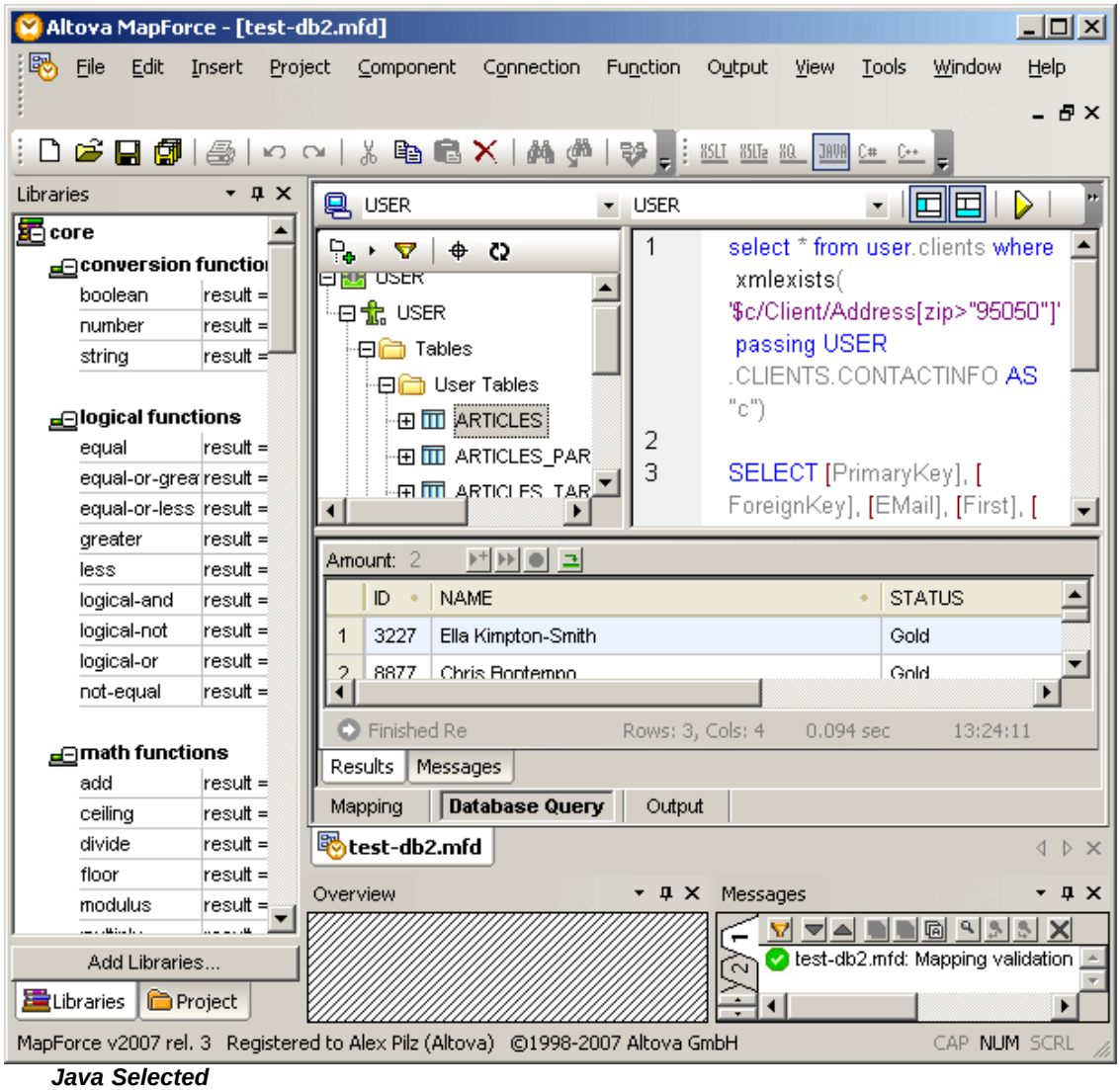
MapForce has a dedicated database query tab that allows you to directly query any major database. The queries/actions defined here are independent of any of the other MapForce tabs, and are not saved as part of the *.MFD file.

Each **Database Query** tab is associated with the currently active mapping, allowing multiple database queries per session/mapping. Note that you can also have **multiple active connections**, to different databases, for each Database Query tab.

The windows of the tab are the:

- **Browser** window at left, which displays connection info and database tables
- **SQL Editor** window, to the right of the Browser window, in which you write your SQL queries
- **Result** window which displays the query results in tabular form
- **Messages** window which displays warnings or error messages

The top row of the Database Query window contains the Connection controls allowing you to define the working databases, as well as the connection and database schemas.



Java Selected

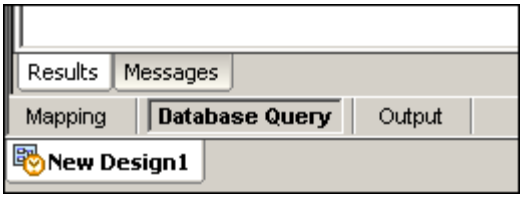
15.14.1 Selecting / connecting to a database

To be able to query a database you first have to make a connection to it. Note that multiple connections can exist in each Database Query window and clicking the Data source combo box allows you to switch between databases and query them from the SQL Editor.

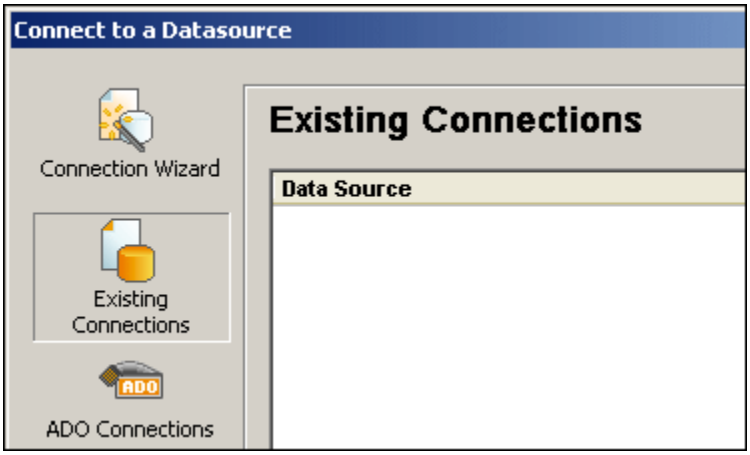
The examples shown in this section use a sample database supplied with IBM DB2 version 9.

To connect to a database:

1. Click the **Database Query** tab in the main window.



2. Click the  icon in the Connection icon bar.

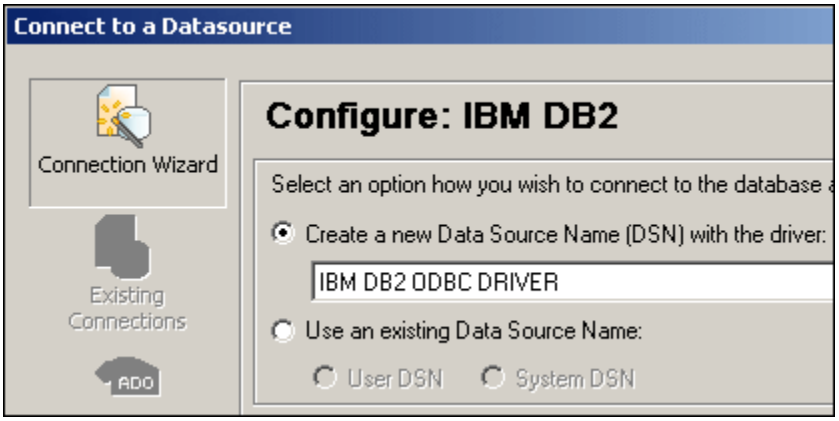


This opens a connection wizard through which you can connect to any type of database. If you have mappings open which contain database connections, then they will appear in the Existing Connection page shown here. At this point the assumption is that no connections are currently active.

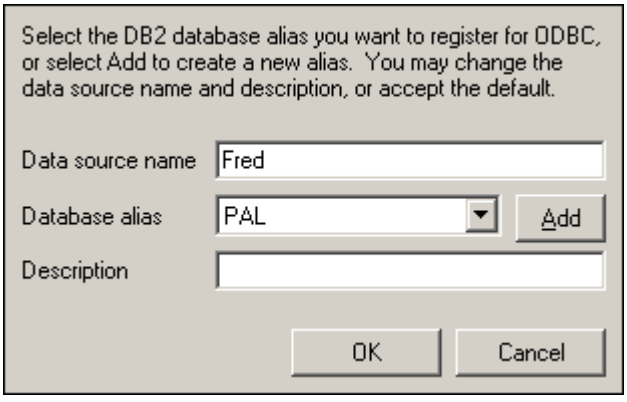
Clicking the Global Resources icon  in the left pane, allows you to select from databases that have been defined as global resources, please see Global Resources - Databases for more information.

3. Click the Connection Wizard icon to create a new connection.

4. Select the database you want to connect to e.g. IBM DB2, and click Next.

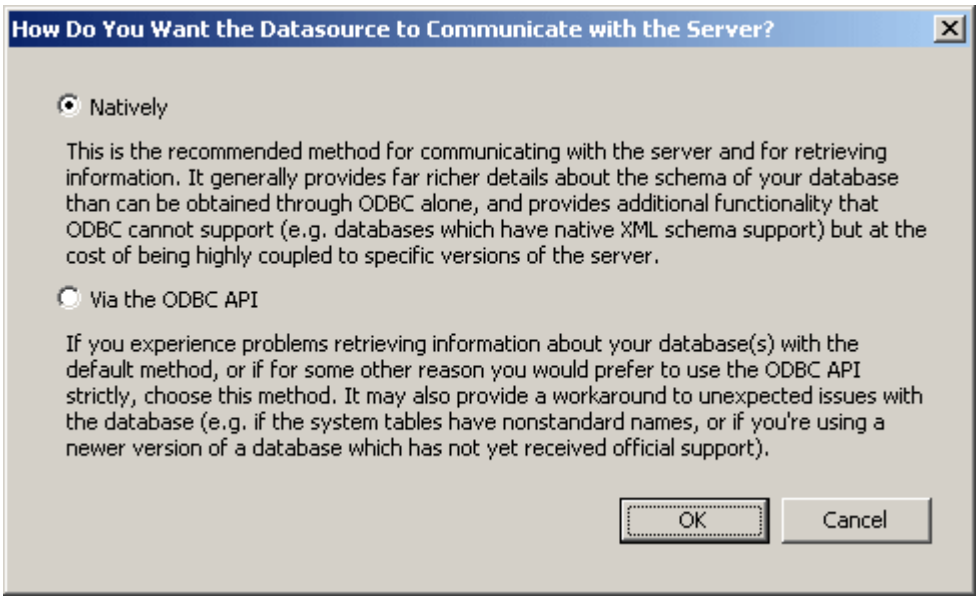


5. Click Next if you want to create a new Data Source Name (DSN), or click the "Use an existing Data Source Name" radio button if you previously defined a DSN.



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

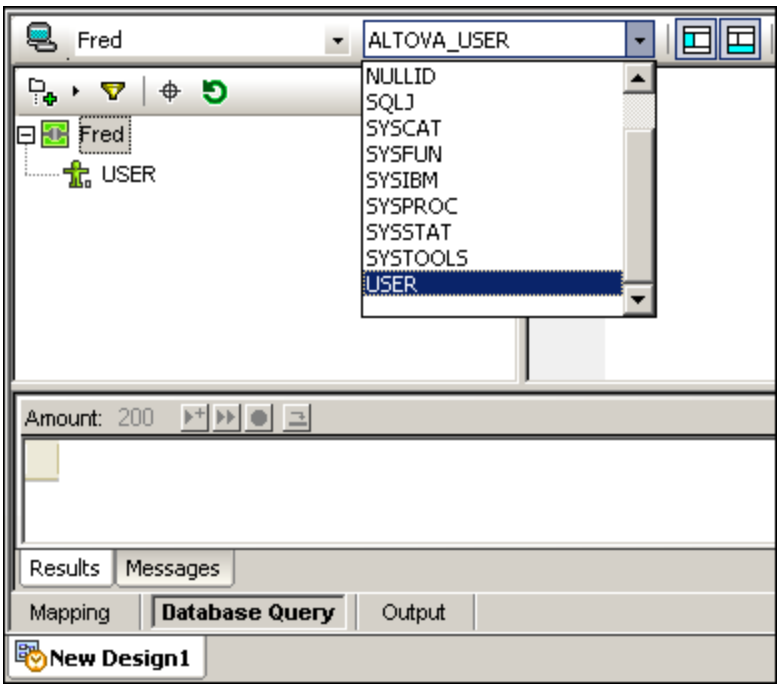
ODBC API.



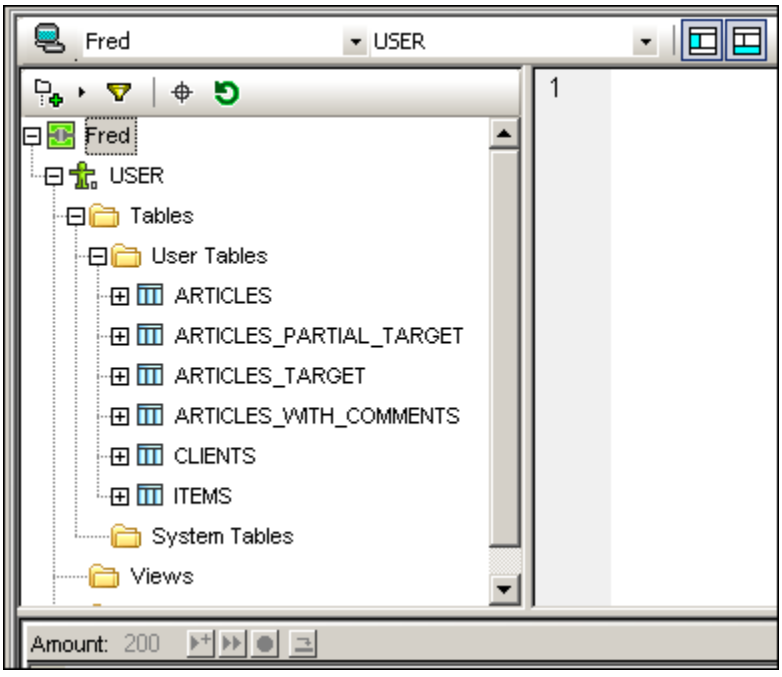
The recommended method is "Natively", then click the OK button.

6. Enter the DSN name and the database alias, then click OK to continue.

7. Enter the database login data and click OK to continue.
   A connection to the DB2 database has now been established.



8. Click the second combo box and select the "Root object" e.g. the USER database schema.



The database tables are now visible under the Tables folder. The "root" objects for the various databases are shown in the table below:

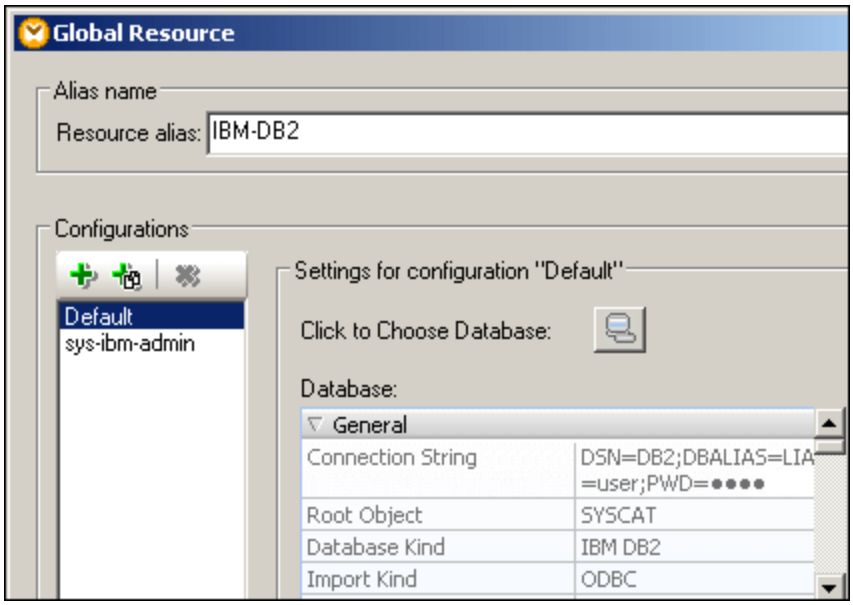
MS SQL Server	database
Oracle	schema
MS Access	database
MySQL	database

DB2	schema
Sybase	database

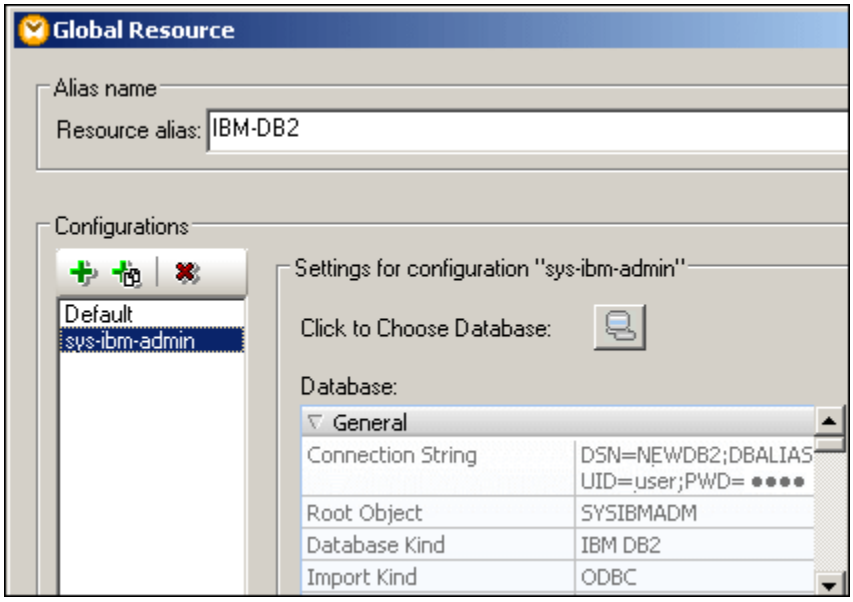
15.14.2 Selecting a database Global Resource

Clicking the Global Resources icon in the connection dialog box allows you to select from databases that have been defined as global resources. Please see Global Resources - Databases on how to define a database global resource.

Global resources can be defined as using the same database but having different tables for each configuration.
The **Default** configuration accesses the DB2 database SYSCAT tables/Root Object as shown in the screenshot below.



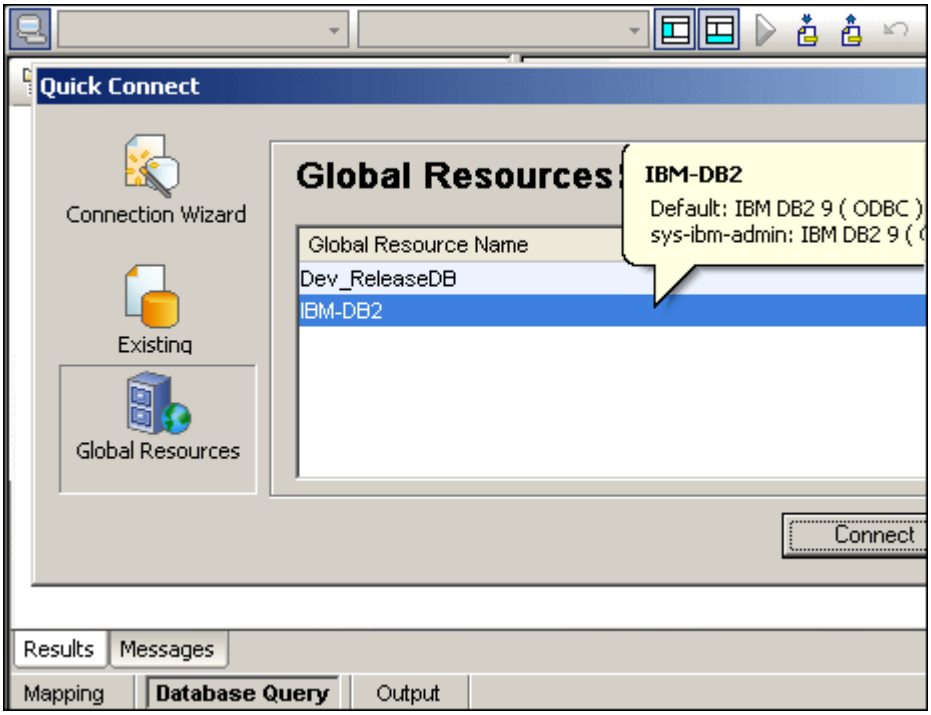
Whereas the **sys-ibm-admin** configuration accesses the same database but only the SYSIBMADM tables/Root object.



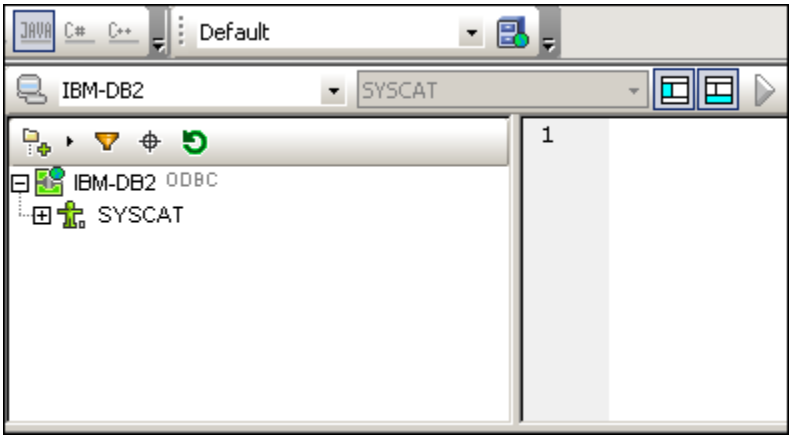
To switch the configuration in the DB-Query window:

1. Click the Database Query tab to open the query window.

2. Click the connection icon  in the Connection icon bar, then click the Global Resources icon.

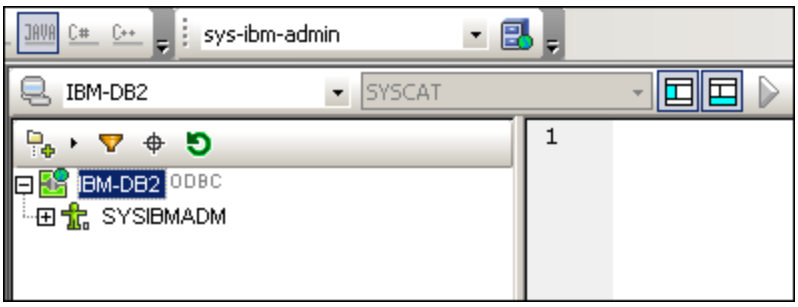


3. Click the Global Resource name e.g. IBM-DB2 and click the Connect button.



The global resource name is now visible in the left combo box, with the Root object SYSCAT greyed out in the right one.

4. Click the Global resource combo box and select sys-ibm-admin.
The "Configuration Switch - Reload" dialog box opens at this point and asks if you want to reload the resource.
5. Click the Reload button to switch the configuration.

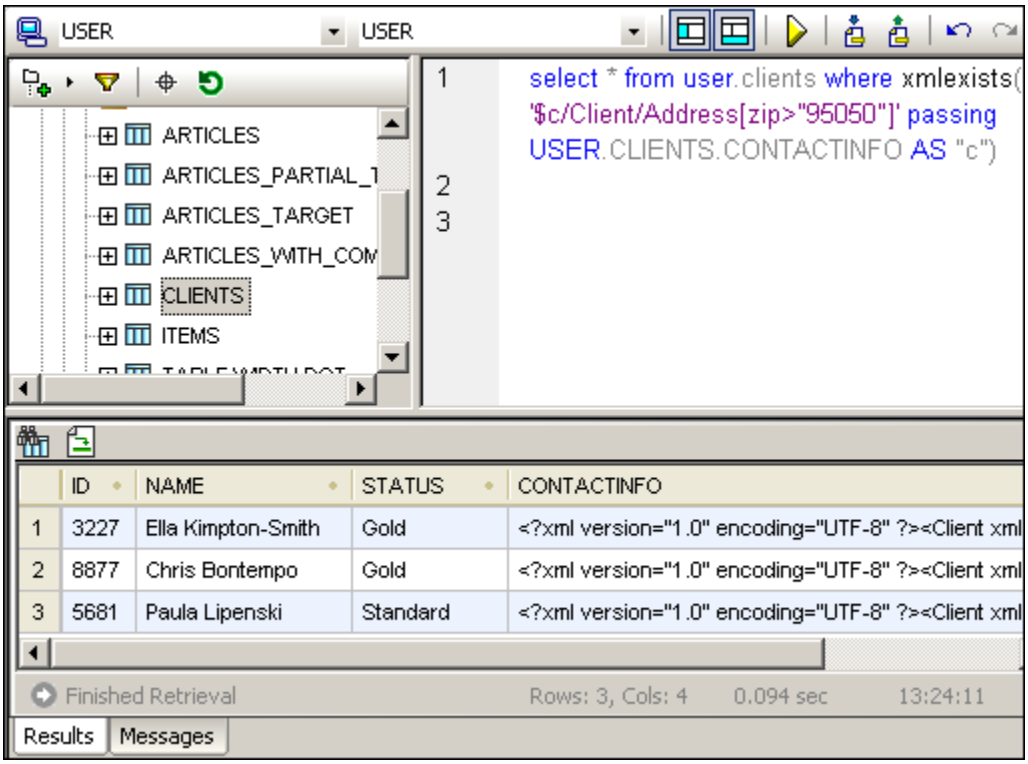


The new root object / table is now visible and you can query the specific tables.

15.14.3 Querying data

To query database data:
Having connected to the database as discussed in the previous section, Selecting / connecting to a database:

1. Click the Import SQL file icon and select the **db2-query.sql** file available in the ...\ **Tutorial** folder. You can also enter you own SQL statement in the SQL window if you do not want to use the supplied file.
2. Click the **Execute** button.



The database data is retrieved and displayed in the Results tab in tabular form. Note that the status bar Finished Retrieval displays the current mode **Retrieval,** and other pertinent result set information.

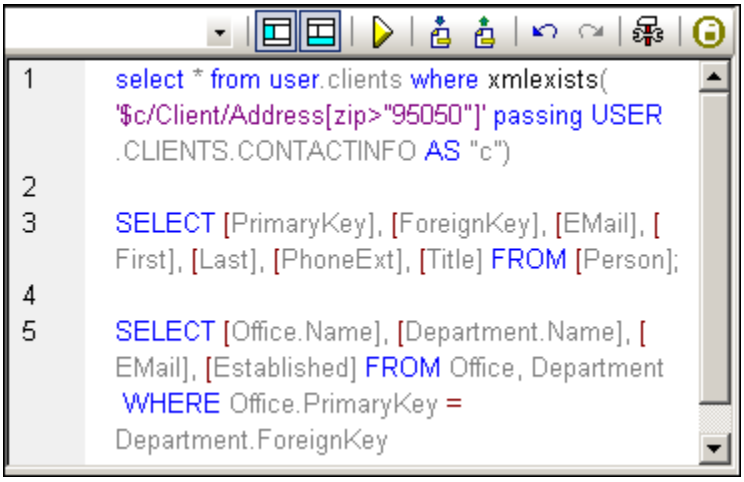
The Retrieval mode allows you to copy, select, or sort the data in the Results tab, by right clicking and selecting the respective option from the context menu. Please see Database Query - Results & Messages tab for more information.

15.14.4 Database Query - SQL window

The **SQL Editor** is used to write and execute SQL statements.

SQL Editor features:

- **autogeneration** of SQL statements using **drag&drop** from the Browser pane
- autocompletion of SQL statements when creating select statements
- definition of **regions**
- insertion of line or block **comments**



The following icons are provided in the SQL toolbar:

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 in the active window of the SQL Editor. 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 window.

Options: Opens the Options dialog box allowing you to define General as well as SQL Editor settings.

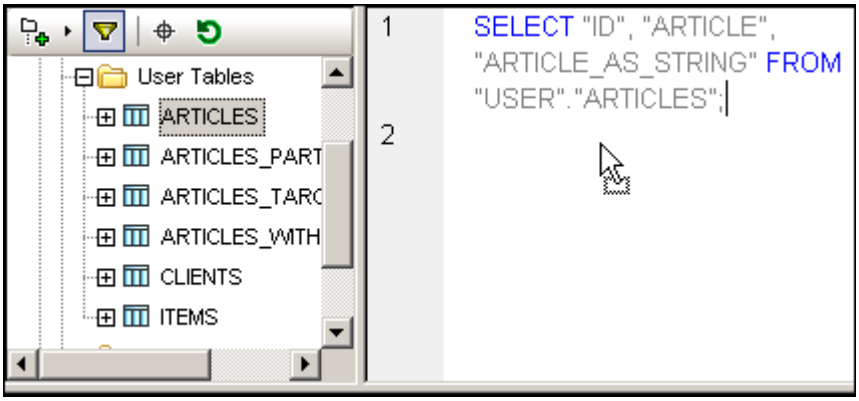
Generating SQL statements

SQL statements based on existing tables and columns in the Browser can be generated automatically using several methods.

- Dragging a database object from the Browser pane into the SQL Editor
- Right-clicking a database object in the Browser and selecting the specific option from the context menu.

To generate SQL statements using drag and drop:

1. Click the ARTICLES table in the Browser and drag it into the SQL Editor window.



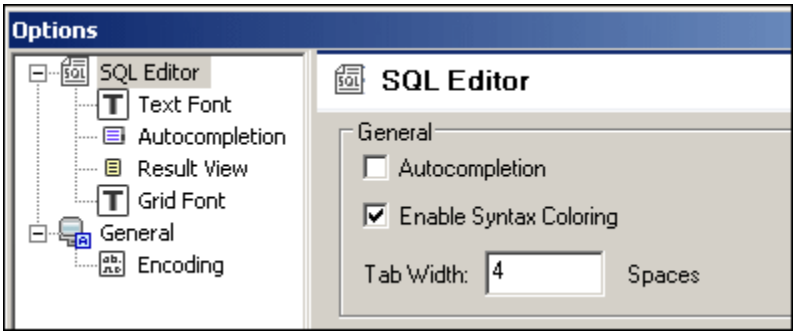
An SQL statement appears in the SQL Editor.

To generate SQL statements using the context menu:

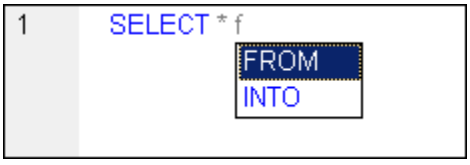
1. Right-click a database object in the Browser and select **Show in SQL Editor | Select**.

To create SQL statements manually using autocompletion:

1. Click the Options icon in the toolbar, then click the **SQL Editor** entry in the tree at left.
2. Activate the **Autocompletion** check box in the General section and click OK to confirm.



3. Start entering the SQL statement in the SQL Editor.



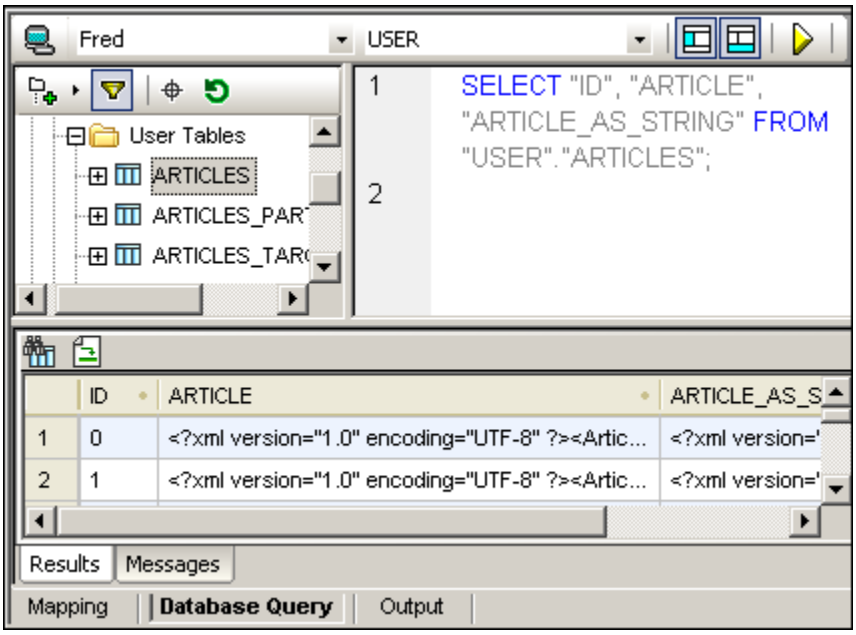
Autocomplete popups appear while entering the select statement. Press Enter when you want to insert/use a highlighted option.

Executing SQL statements

SQL statements that have been created, or opened, can be executed directly from the SQL Editor.

To execute SQL in an SQL Editor window:

1. Enter or select an SQL statement in the SQL Editor.
2. Click the **Execute** button.



If a data source is not connected, a popup message is displayed asking whether you would like to connect to the database.

3. Click **Yes** in the message box to connect to the data source.

To select individual SQL statements:

- Use the mouse to mark the specific statement.
- Move the mouse cursor into the **line number** columm of the SQL Editor and click to select a complete statement.
- Click three times in an existing select statement.

Saving and opening SQL scripts

You can save any SQL that appears in an SQL Editor window and re-use the script later on.

To save the content of an SQL Editor window to a file:

1. Click the **Export SQL file** icon **, and enter a name for the SQL script.**

To open a previously saved SQL file:

1. Click the **Import SQL file** icon **,** and select the SQL file you want to open in the SQL window.

SQL Editor features

You can edit SQL statements in the SQL Editor as in any other text editor. The SQL Editor provides additional features such as:

- autocompletion
- commenting out text
- bookmarks
- regions

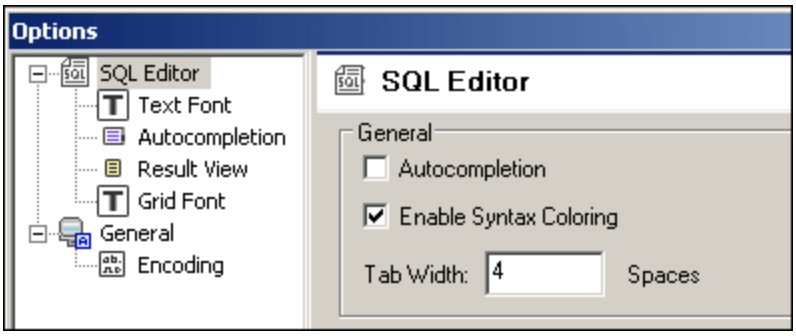
Autocompletion

When entering an SQL statement in the SQL Editor, autocompletion helps you by offering lists of appropriate keywords, data types, identifiers, separators, and operators depending on the type of statement you are entering. Autocompletion is currently available for the following databases:

- MS SQL Server 2000
- MS SQL Server 2005
- MS Access 2003
- IBM DB2 9

To activate autocompletion:

1. Click the Options icon  in the toolbar, then click the **SQL Editor** entry in the tree at left.
2. Activate the **Autocompletion** check box in the General section and click OK to confirm.



3. Start entering the SQL statement in the SQL Editor.

Autocomplete popups appear while you enter the statement. Press Enter when you want to insert a highlighted option.

Commenting out text

The SQL Editor allows yout 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.

```
1   select * from user.clients where xmlexists(
    '$c/Client/Address[zip>"95050"]' passing USER
    .CLIENTS.CONTACTINFO AS "c")
2
3   SELECT "ID", "BRANDNAME", "ITEMNAME", "SKU", "SRP",
    "COMMENTS" FROM "USER"."ITEMS";
4
5   SELECT [PrimaryKey], [ForeignKey], [Name] FROM [
    Department];
```

2. Right click the selected statement and select **Insert / Remove Block Comment**.

```
1   select * from user.clients where xmlexists(
    '$c/Client/Address[zip>"95050"]' passing USER
    .CLIENTS.CONTACTINFO AS "c")
2
3   /*SELECT "ID", "BRANDNAME", "ITEMNAME", "SKU",
    "SRP", "COMMENTS" FROM "USER"."ITEMS";*/
4
5   SELECT [PrimaryKey], [ForeignKey], [Name] FROM [
    Department];
```

The statement is commented out.

To comment out text line by line:

1. Right click at the position you want to comment out the text and select **Insert / Remove Line Comment**.

```
3   SELECT "ID", "BRANDNAME", "ITEMNAME", "SKU", "SRP",
    "COMMENTS" FROM "USER"."ITEMS";
4
5   SELECT [PrimaryKey], --[ForeignKey], [Name] FROM
    [Department];
6
```

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:

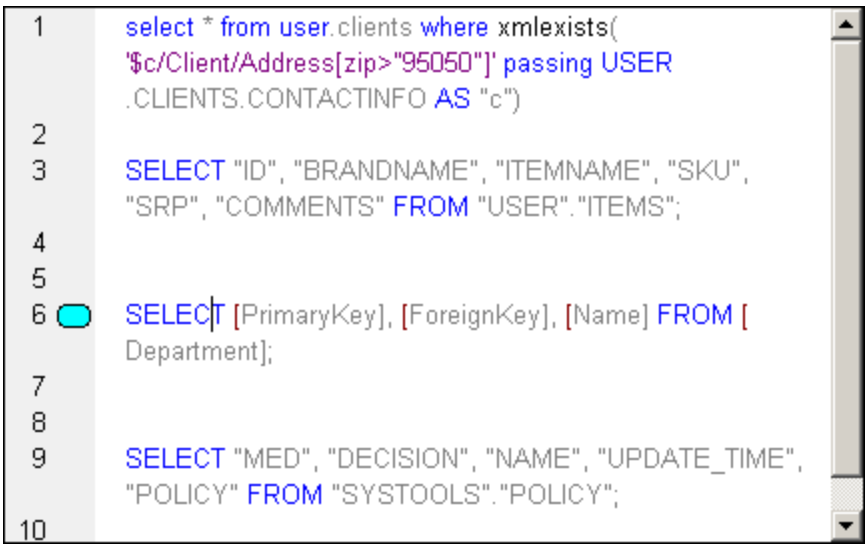
1. Select the part of the statement that is commented out.
   If you want to remove a line comment, it is sufficient to select only the comment marks - - before the comment.
2. Right click and select **Insert / Remove Block (or Line) Comment**.

Using bookmarks

Bookmarks are used to mark items of interest in long scripts.

To insert a bookmark:

1. Right click in the line you want to have bookmarked and select **Insert/Remove Bookmark** from the context menu.



A cyan bookmark icon is displayed in the margin at the beginning of the bookmarked line.

To remove a bookmark:

1. Right click in the line you want to remove the bookmark from 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

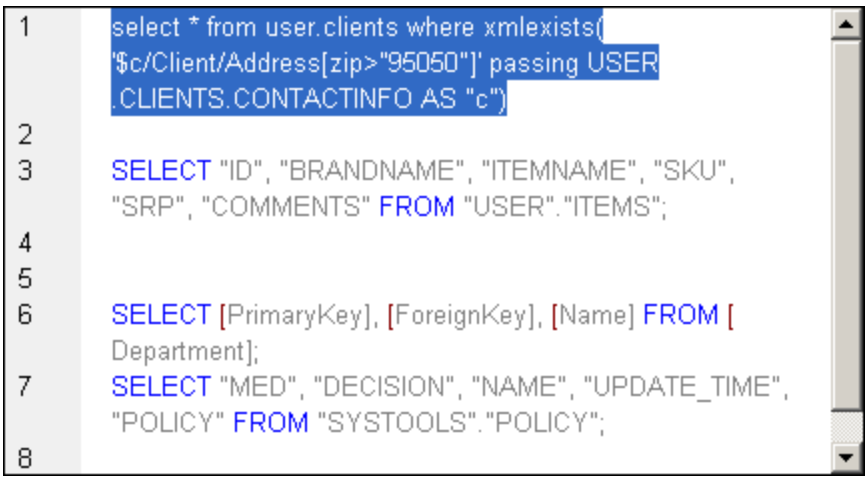
Regions are sections of text that you mark and declare as a unit to structure your SQL scripts. Regions can be collapsed and expanded to display or hide parts of SQL scripts. It is also possible to nest regions within other regions.

When you insert a region, an expand/collapse icon and a `--region` comment are inserted above the selected text.

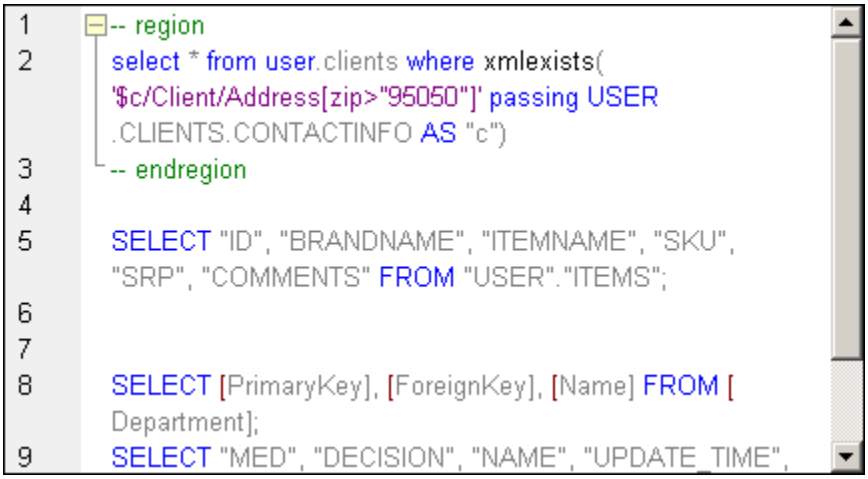
Please note: You can change the name of a region by appending descriptive text to the --region comment. The word "region" must not be deleted, e.g. --region DB2query.

To create a region:

1. In the SQL Editor, select the statements you want to make into a region.



2. Right click and selet **Add Region** from the context menu.
   The area that you marked becomes a region and can be expanded or collapsed.



3. Click the + or - box to expand or collapse the region.

To remove a region:

- Delete the `-- region` and `-- endregion` comments.

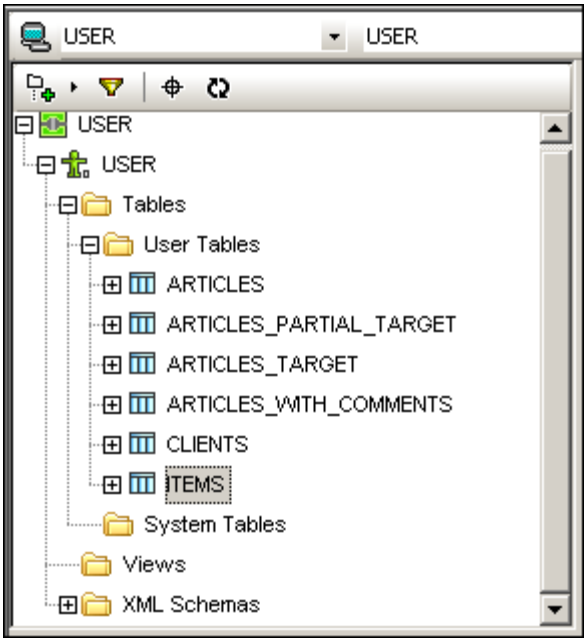
15.14.5 Database Query - Browser window

For each of the (multiple) connected data sources, the **Browser** pane gives a full overview of the objects in each database, including database constraint information, e.g. column as a primary or foreign key. In IBM DB2 version 9 databases, the Browser additionally shows registered XML schemas in a separate folder.

The Browser view can be customized to:

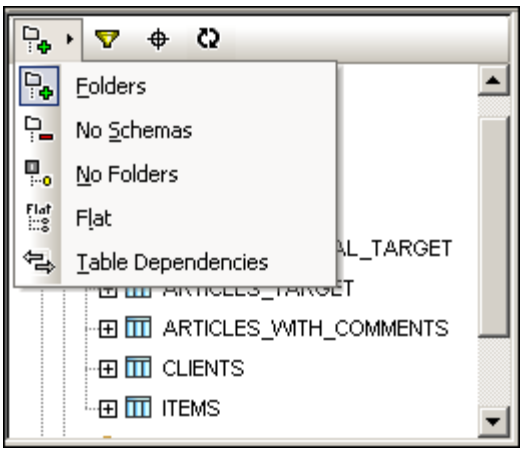
- show specific folder **layouts** when displaying database objects
- **find** specific objects in the database using the Object Locator
- **filter** the number of displayed items
- **refresh** the root object of the active data source.

The default "Folders" layout displays database objects in an hierarchical manner. Depending on the selected object, different context menu options are available when you right-click an item.



To select a layout for the Browser:

- In the Browser, click the layout  icon and select the layout from the drop-down list. Note that the icon changes with the selected layout.



Browser window - Layouts
The Browser pane contains several predefined layouts used to display various database objects:

- 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.

To sort tables into User and System tables:

1. In the Browser, right-click the Tables folder.
   A context-sensitive menu appears.
2. Select **Sort into User and System Tables**.
   The tables are sorted alphabetically in the User Tables and System Tables folders.

   Please note: You must be in the Folders, No Schemas or Flat layout in order to access this function.

To refresh the root object of the active data source:

- Click the Refresh icon in the icon bar.

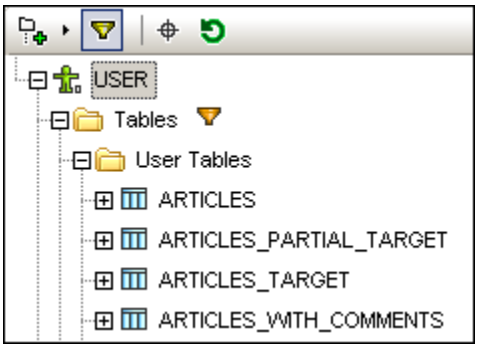
Filtering and finding database objects

The Browser allows you to filter schemas, tables, and views by name or part of a name. Objects are filtered as you type in the characters. Filtering is case-insensitive by default.

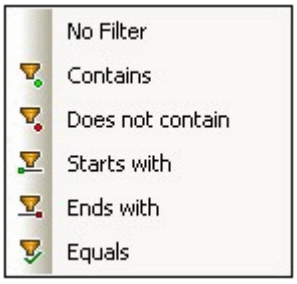
Please note: The filter function does not work if you are using No Folders layout.

To filter objects in the Browser:

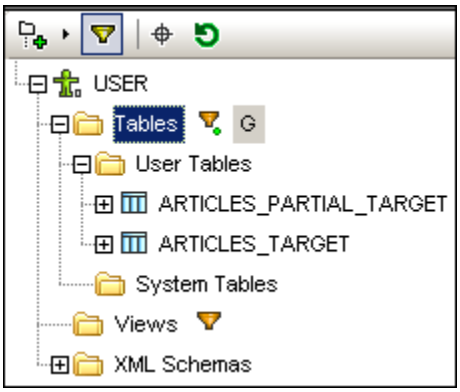
1. Click the **Filter Folder contents** icon in the toolbar.
   Filter icons appear next to all folders in the currently selected layout.



2. Click the filter icon next to the folder you want to filter, and select the filtering option from the popup menu e.g. Contains.



   An empty field appears next to the filter icon.
3. Enter the string you want to search for e.g. G. The results are adjusted as you type.

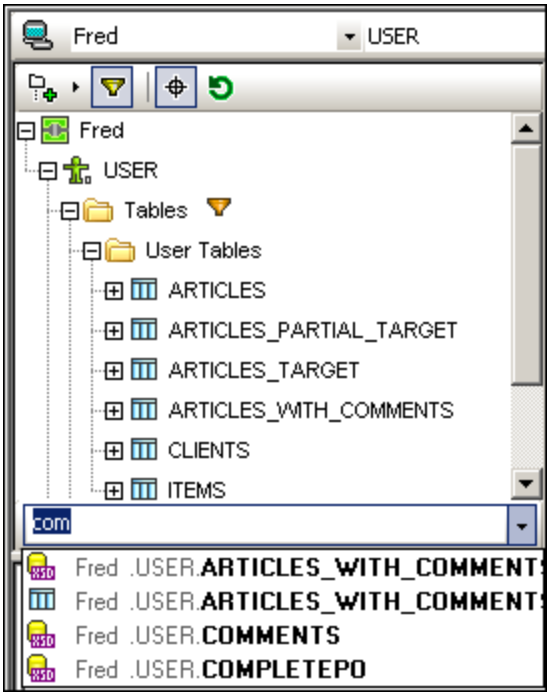


Finding database objects

To find a specific database item by its name, you can either use filtering functions or the **Object Locator** in the Browser.

To find database elements using the Object Locator:

1. In the Browser, click the **Object Locator** icon.
   A drop-down list appears at the bottom of the Browser.
2. Enter the string you want to look for, e.g., "com".
   Clicking the drop-down arrow displays all elements that contain that string.



3. Click the object in the list to see it in the Browser.

Context options in Browser view

Types of context menu are available in the Browser view:

- Right clicking the "root" object
- Right clicking a **folder** e.g. User tables
- Right clicking any type of database **object** e.g. table ARTICLES

Right clicking the "root" object allows you to **Refresh** the database.

Right clicking a **folder** always presents the same choices:
Expand | Siblings | Children
Collapse | Siblings | Children

Right clicking a **database object** presents:
Show in SQL Editor and the submenu items discussed below.

Please note: The syntax of the statements may vary depending on the database you are using. The descriptions below use Microsoft SQL Server 2000 as an example. Use SHIFT + CLICK and CTRL + CLICK to select multiple database objects.

The following options are available in the context menu for **tables**:

- **Select**: Creates a SELECT statement that retrieves data from all columns of the source table.
  E.g. SELECT "ID", "ARTICLE", "ARTICLE_AS_STRING" FROM "USER"."ARTICLES";
- **Name**: Returns the name of the table. The names are printed on separate lines, separated by commas.
- **Path**: Returns the full path of the tables, i.e., DataSourceName.DatabaseName. SchemaName.TableName. You can also select several tables. The names are printed

on separate lines, separated by commas.

The following options are available in the context menu for **columns**:

- **Select**: Creates a SELECT statement that retrieves data from the selected column(s) of the parent table.
  E.g. SELECT "ARTICLE" FROM "USER"."ARTICLES";
- **Name**: Returns the name of the selected column. The names are printed on separate lines, separated by commas.
- **Path**: Returns the full path of the column, i.e., DataSourceName.DatabaseName. SchemaName.TableName.ColumnName. You can also select several columns. The names are printed on separate lines, separated by commas.

The following options are available in the context menu for **constraints**:

- **Name**: Returns the name of the selected constraint. The names are printed on separate lines, separated by commas.
- **Path**: Returns the full path of the constraint, i.e., DataSourceName.DatabaseName. SchemaName.TableName.ConstraintName. The names are printed on separate lines, separated by commas.

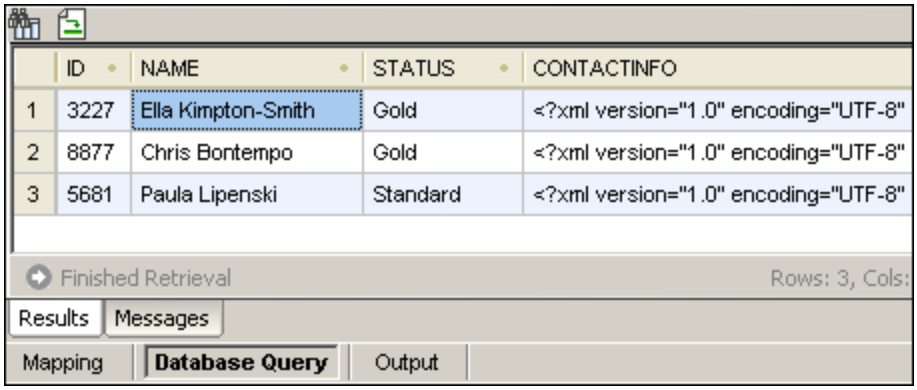
  E.g. "USER"."ARTICLES_PARTIAL_TARGET"."CC1175533269606"

The following options are available in the context menu for **indexes**:

- **Name**: Returns the name of the selected index. The names are printed on separate lines, separated by commas.
- **Path**: Returns the full path of the index, i.e., DataSourceName.DatabaseName. SchemaName.TableName.IndexName. The names are printed on separate lines, separated by commas.

15.14.6 Database Query - Results & Messages tab

The **Result tab** of the SQL Editor shows the record set that is retrieved as a result of a database query.



Selecting & copying data in the Result window:

There are various methods of selecting data in this window, which you can then copy to other applications.

- Click a column header to select the column data
- Click a row number to mark row data
- Click individual cells

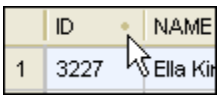
  Holding down the CTRL key while clicking allows you to make multiple selections. If a column, or cell, contains XML data then this data can also be copied.

- Right click and select **Copy selected cells** from the context menu.

  Note: The context menu can also be used to select data, **Selection | Row | Column | All.**

To sort data in Result windows:

- 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.

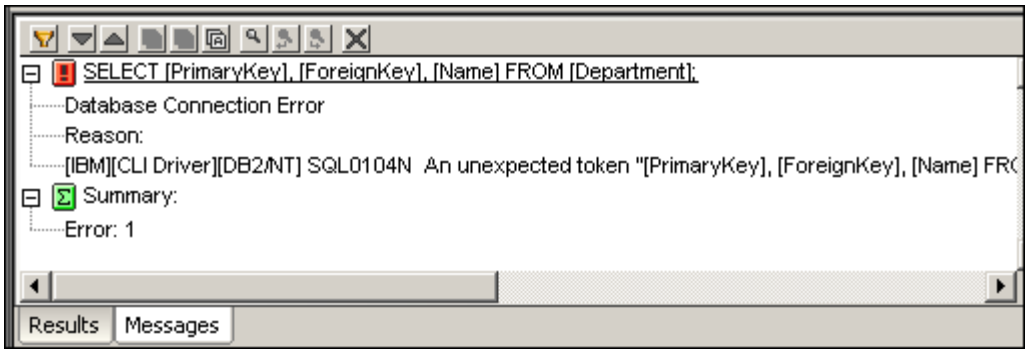
Toolbar options - Retrieval mode

The Result window provides a toolbar that allows for the navigation between results and SQL statements and facilitates the easy retrieval of parts of database data.

Find: Searches for the input string in the Result window. The F3 function key allows you to continue the search.

Go to statement: Jumps to the SQL Editor window and highlights the group of SQL statements that produced the current result.

The **Messages tab** of the SQL Editor provides specific information on the previously executed SQL statement and reports errors or warning messages.



You can use different filters to customize the view of the Message tab or use the **Next** and **Previous** buttons to browse the window row by row. The Message tab also provides a **Find** dialog box and several options to copy text to the clipboard.

Toolbar options
The Message window provides a toolbar that allows for the navigation inside the messages and includes filters for hiding certain parts of the message.

Filter: Clicking this icon opens a popup 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 popup 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.

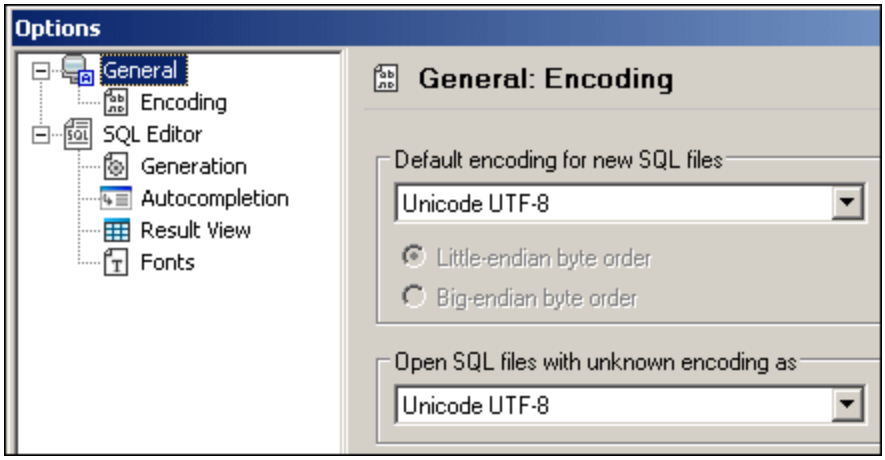
Find next: Jumps to the next occurrence of the string specified in the **Find** dialog box.

Clear: Removes all messages from the Message tab of the SQL Editor window.

Please note:
The same options are available in the context menu of the result window.

15.14.7 Database Query - Settings

The Encoding section of the **Options** dialog box, allows you to specify several file encoding options.



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 (i.e., UTF-16, UCS-2, or UCS-4), you can also choose between little-endian and big-endian byte ordering for the SQL files.

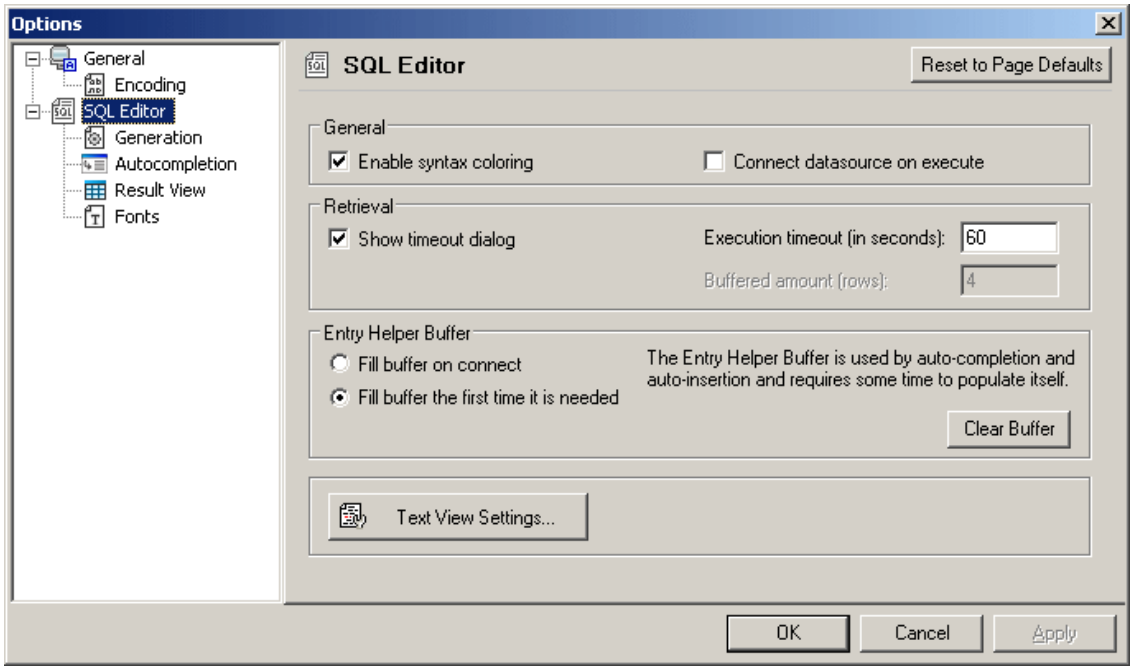
The encoding for existing files will, of course, always be retained.

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.

Please note: SQL files which have no encoding specification are correctly saved with a UTF-8 encoding.

SQL Editor options

The main page of the SQL Editor options defines the visual appearance of the editor and sets the autocompletion options. Additional SQL Editor-related settings are defined in the Generation , Autocompletion, Result view, and Fonts options.



General
The SQL Editor provides an autocompletion feature which lets you select from a drop-down list of SQL key words and database object names as you type. This check box activates the autocompletion settings defined in the Autocompletion page.

Syntax coloring emphasizes different elements of SQL syntax using different colors.

Adjust the **tab width** to define the number of spaces that are to be inserted when the Tab key is pressed in an SQL Editor window.

Activating the **Connect datasource on execute** check box connects to the corresponding data source automatically whenever an SQL file 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 you want the entry helper buffer to be filled, on connection or only the first time it is needed.

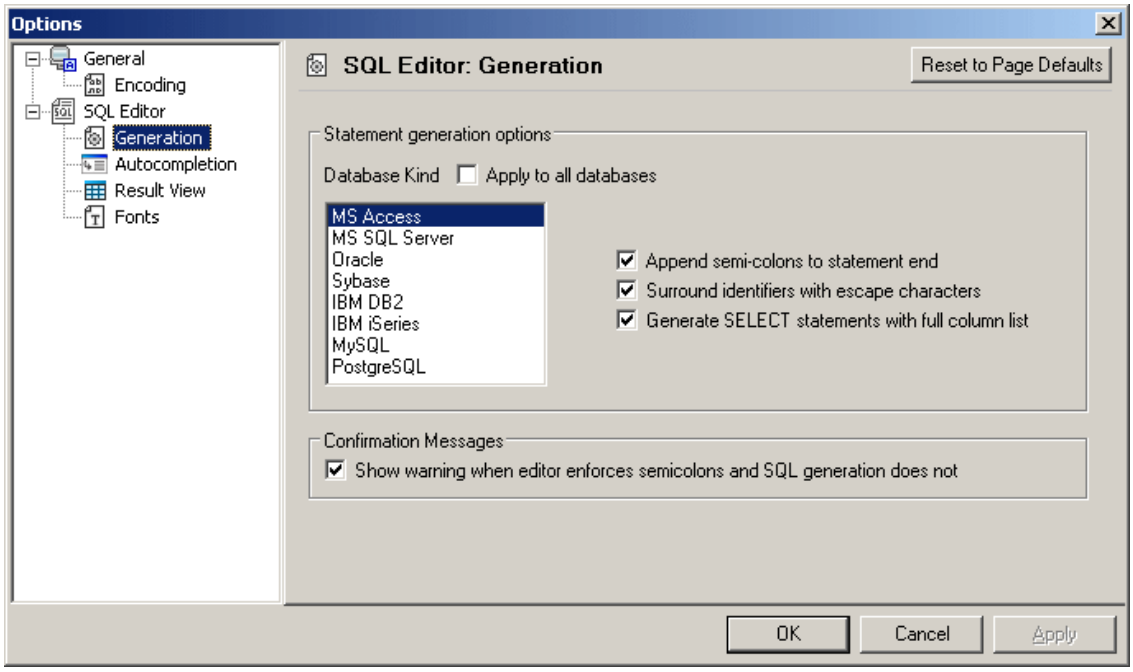
Text View Setting button
Allows you to define the specific Text view settings: Margins, Tabs, Visual aids, as well as showing you the Text view navigation hotkeys.

Generation

The Generation section allows you define the statement generation syntax for the various databases.

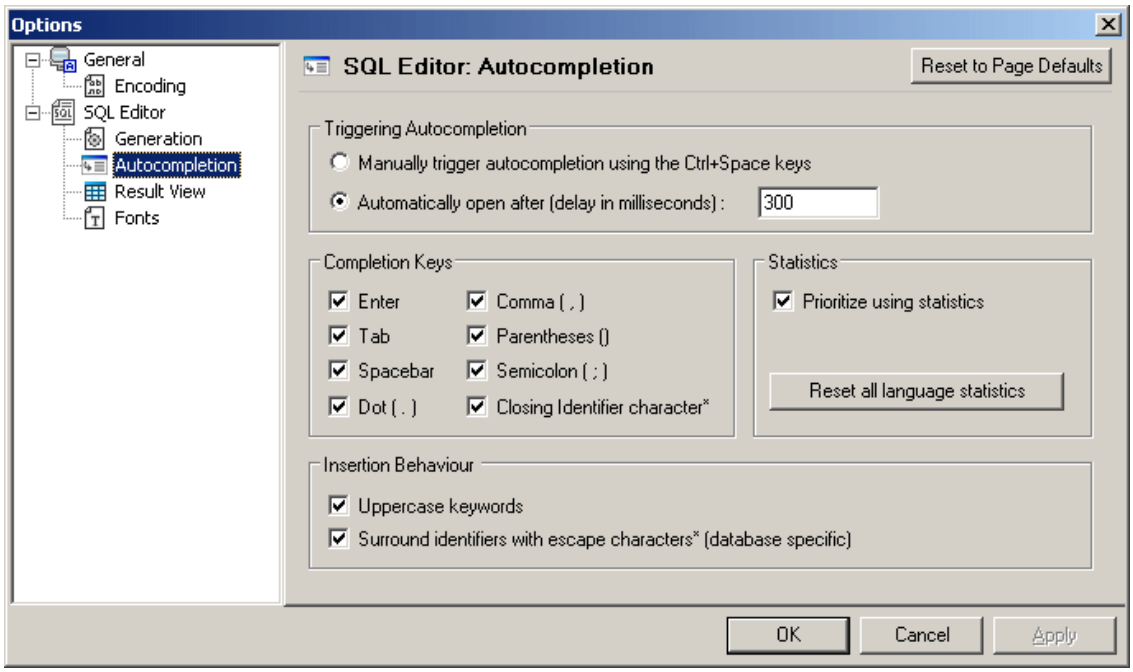
Clicking a database in the list box allows you to define the specific syntax using the three check

boxes at right.
Activating the "Apply to all databases" check box greys out the database list and applies the check box settings to all databases in the list.



Autocompletion

The Autocompletion section of the **Options** dialog box lets you configure the specific autocompletion settings.



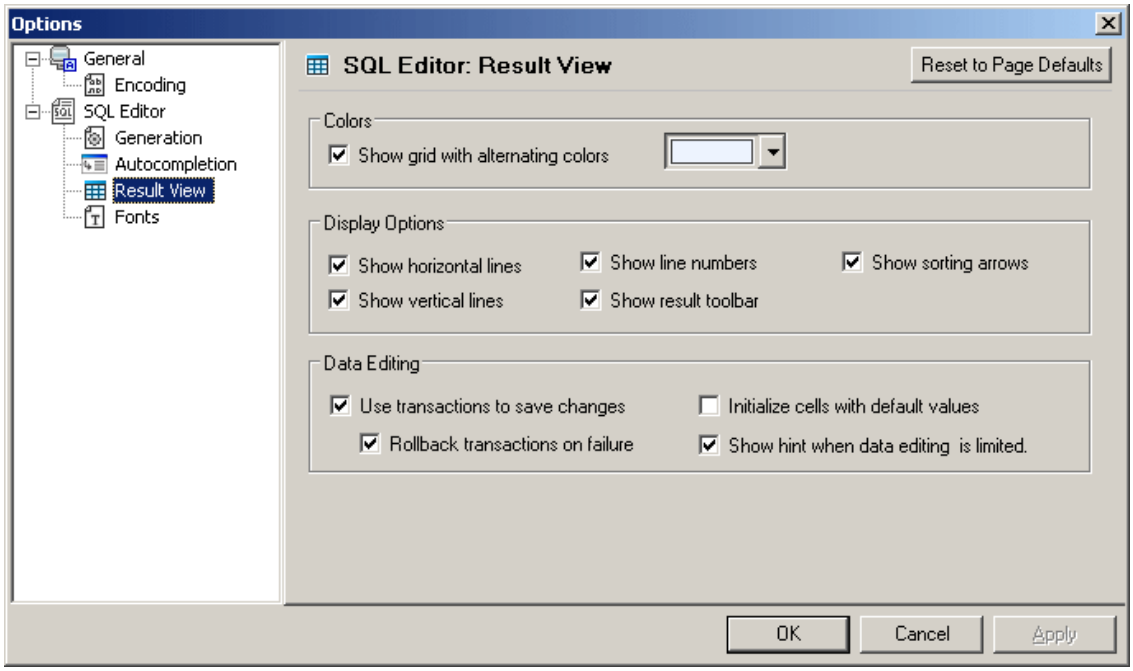
Autocompletion can be triggered manually or automatically.

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.

Result view

The Result View section of the **Options** dialog box lets you configure aspects of the appearance of the Result window in the SQL Editor.



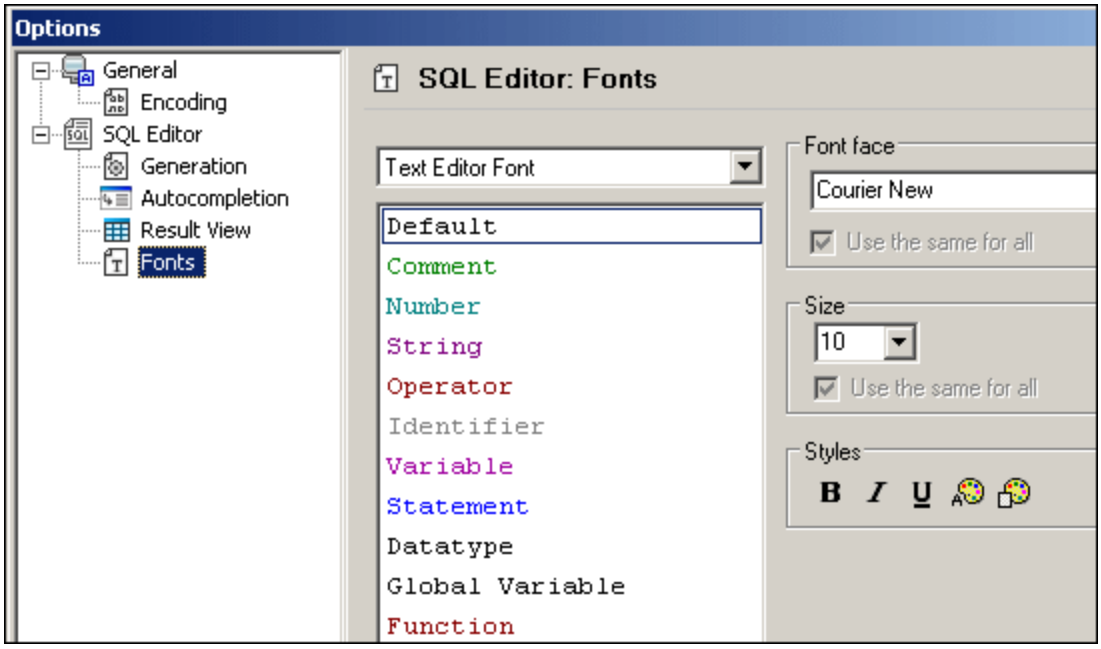
Colors
You can display rows in Result tabs as simple grid or with alternating white and colored rows.

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.

Fonts

The Text Font section of the **Options** dialog box lets you configure color and font settings of different parts of SQL statements.

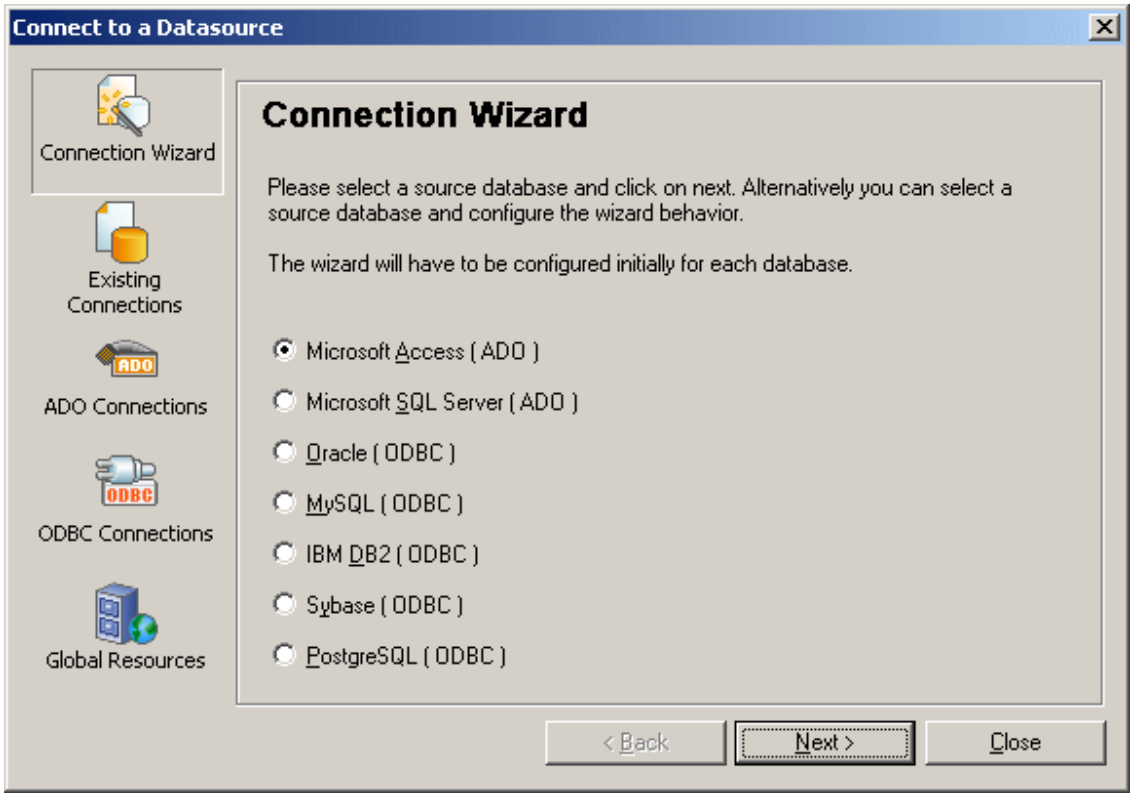


The font settings listed in the Font Settings list box are elements of SQL statements. You can choose the common font face, style, and size of all text that appears in SQL Editor. Note that the same font and size is used for all text types.

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.

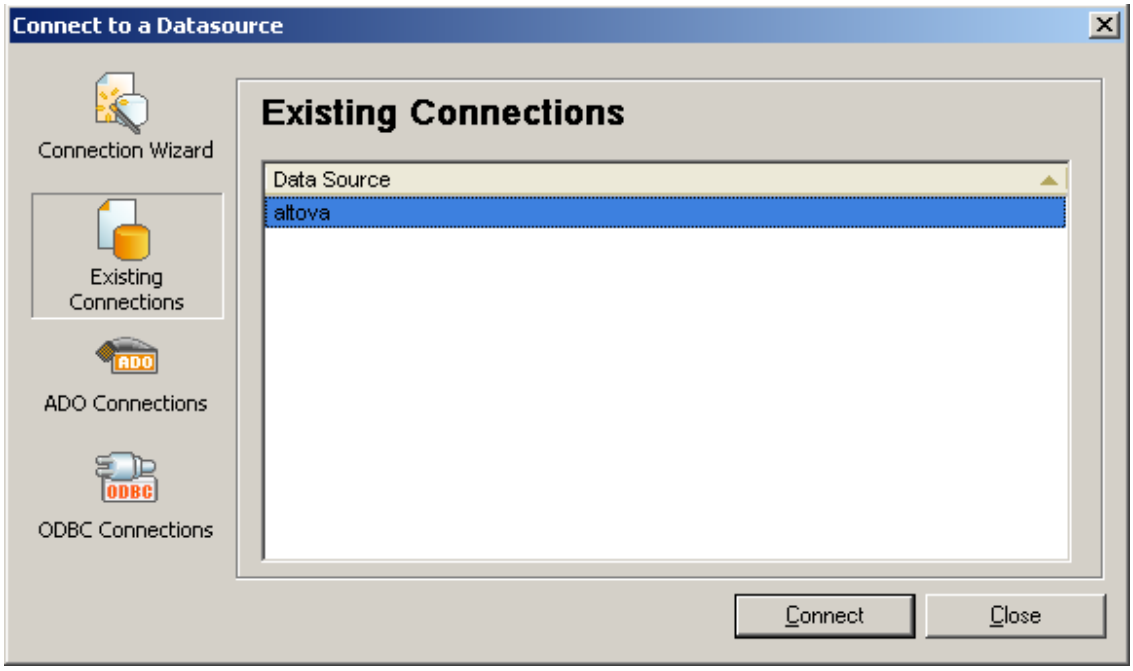
15.14.8 Using the Connection Wizard

The Connection Wizard simplifies the choices needed to connect to a database and presents a number of connection types. It allows you to quickly create connections to any of the database types in the list.



The **Existing Connections** pane lists all the currently **active** database connections.

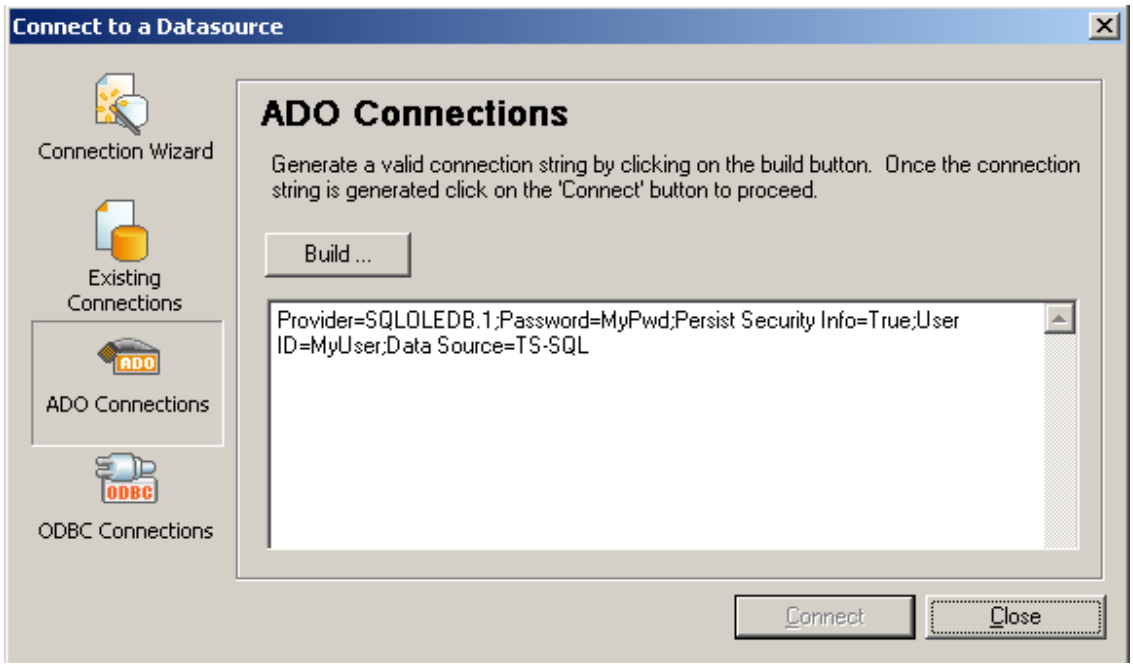
- Select a connection in the list and click **Connect** to connect to that database.



The **ADO Connections** pane allows you to create an ADO connection to a database using the Build button.

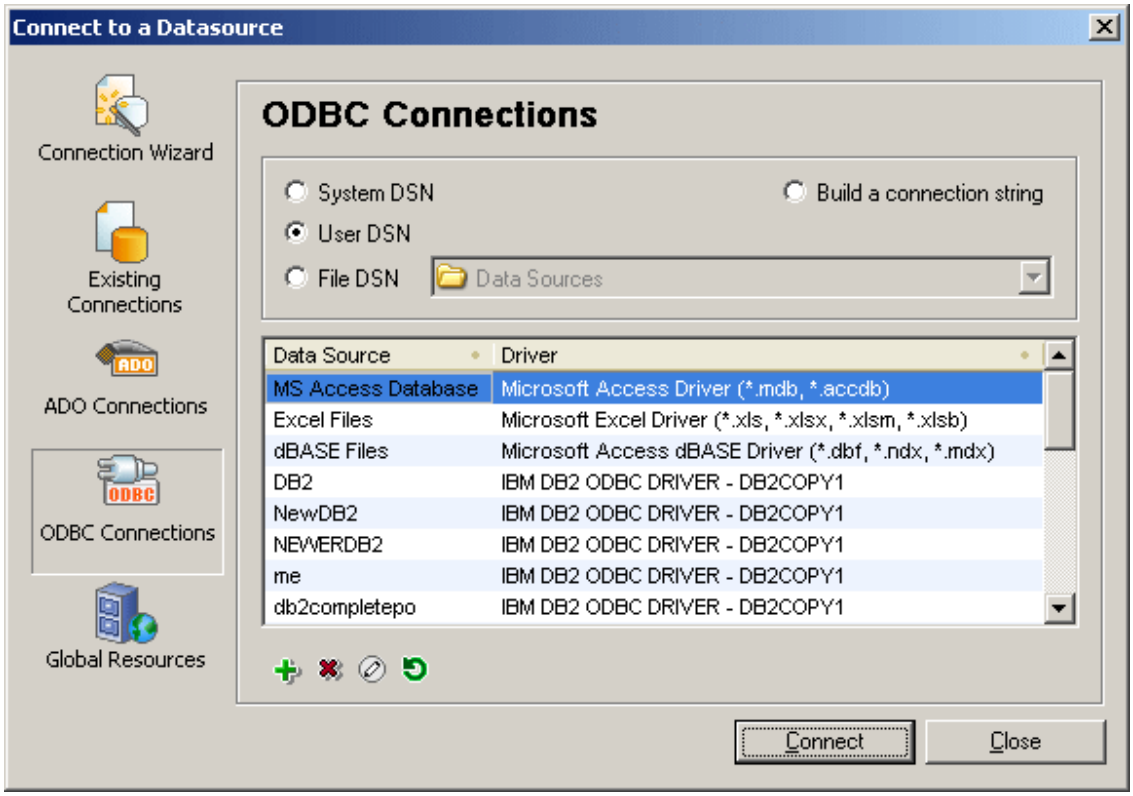
- Click the **Build** button to create the connection string, which appears in the window once it has been defined, then click the **Connect** button to connect to the database.

  An ADO (ActiveX Data Objects) connection can be created without carrying out any preliminary steps, such as creating a DSN. Always use an ADO connection for MS Access databases, as ODBC does not support relationships.



The **ODBC Connections** pane allows you to create an ODBC (Open Database Connectivity) connection to a database. You can choose from among the following types:

- System DSN (data source name): This type of DSN can be used by anyone who has access to the computer. DSN information is stored in the registry.
- User DSN: This type of DSN is created for a specific user and is also stored in the registry.
- File DSN: For this type of DSN, DSN information is stored in a text file with DSN extension.



Please note: to create an ODBC connection you must first create a DSN. The data source list box contains all the previously created DSNs.

Clicking the `System DSN` or the `UserDSN` (Data Source Name) radio button presents several icons at the bottom of the pane which are used to create new, or maintain, existing DSNs:

Create new DSN: Creates a new DSN for the **driver** currently selected in the driver drop-down list located to the right of this icon.

Edit Data Source Name: Allows you to change the settings of the DSN that is selected in the data source list above.

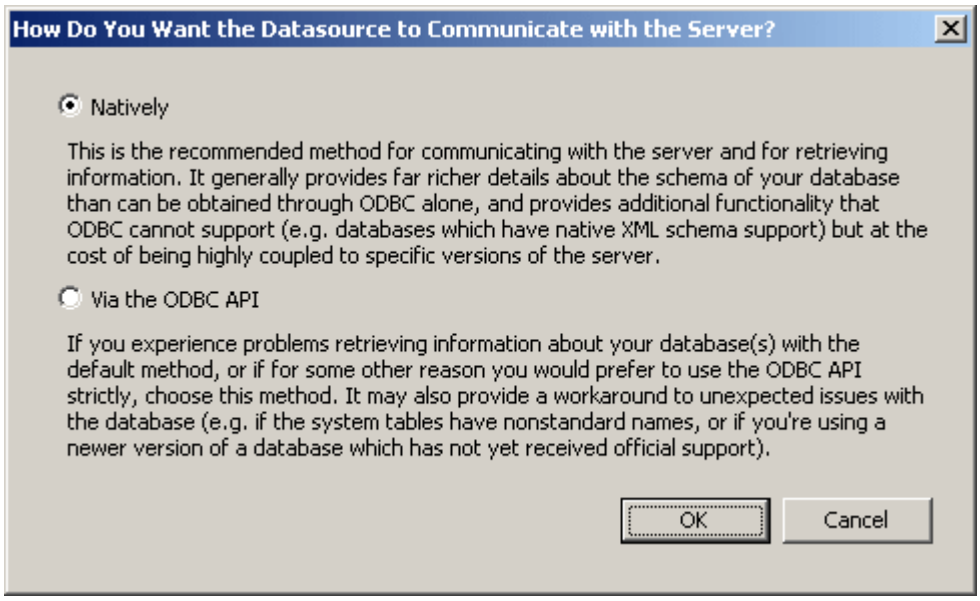
Remove Data Source Name: Removes the currently selected DSN from the data source list.

Refresh Data Source Name: Updates the data source list.

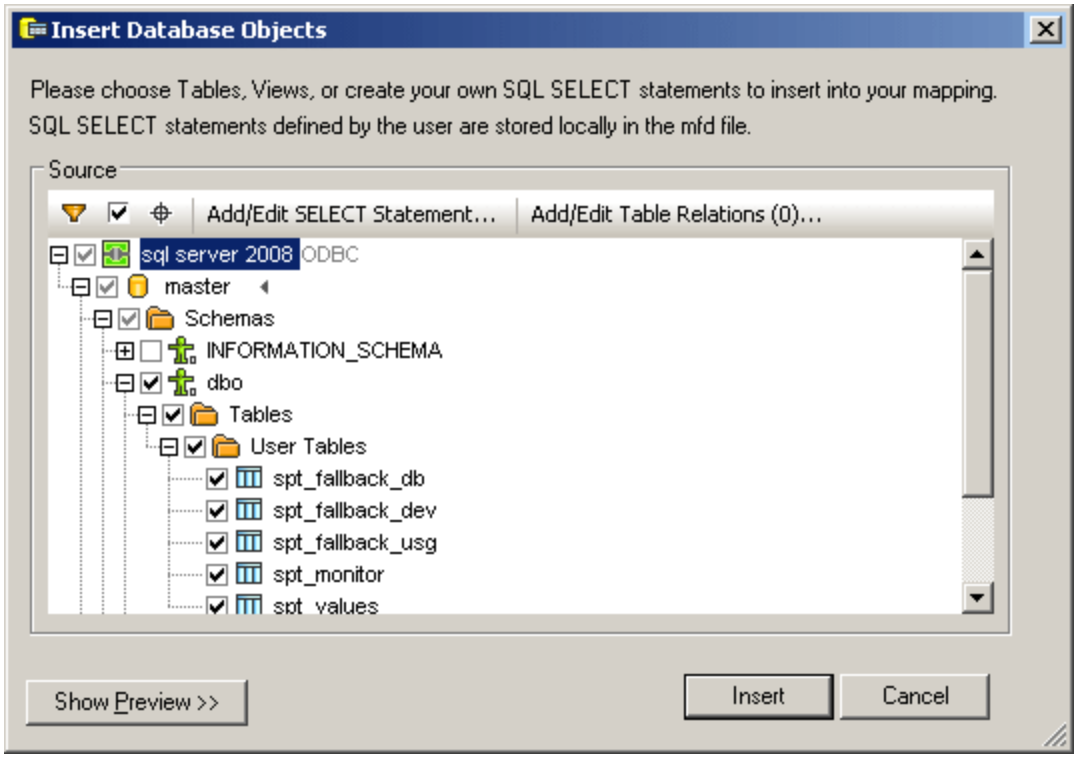
Double clicking one of the data source names in the list box, opens the specific database login

dialog box.

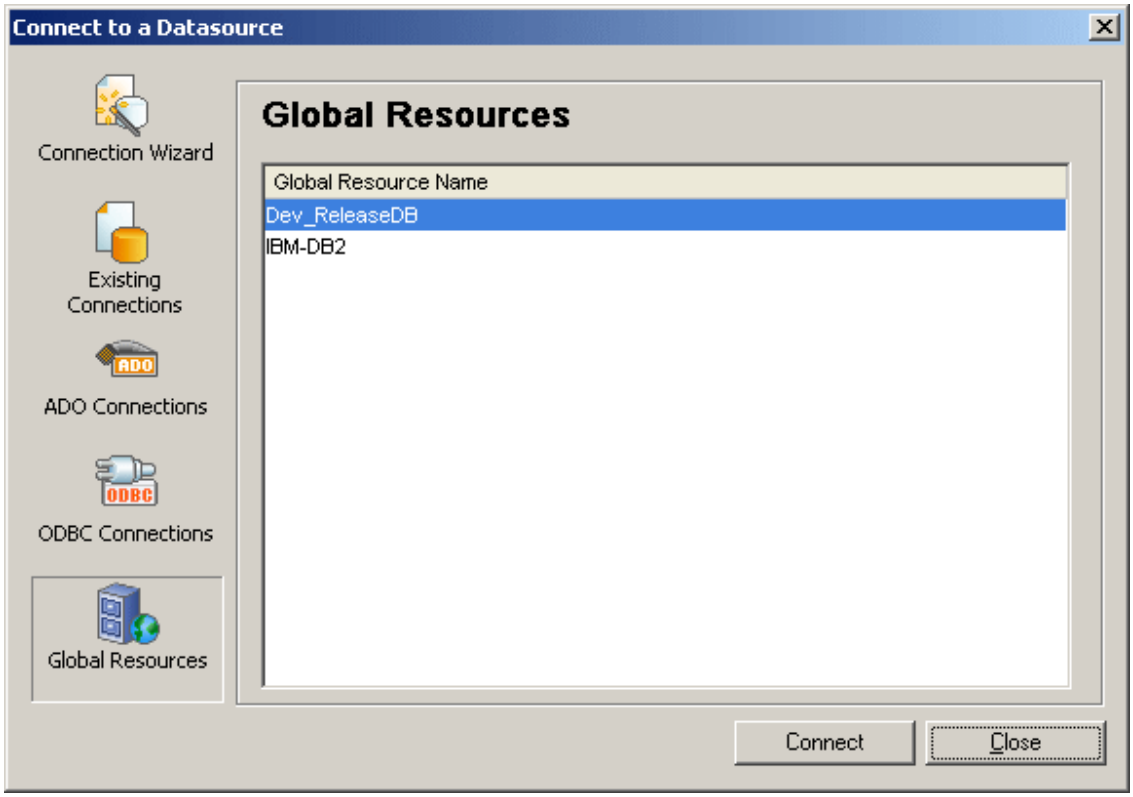
Entering the required login info and clicking OK opens a further dialog box allowing you to choose how the datasource will communicate with the server; natively or via the ODBC API.



Select the specific method and click OK to insert the database objects.



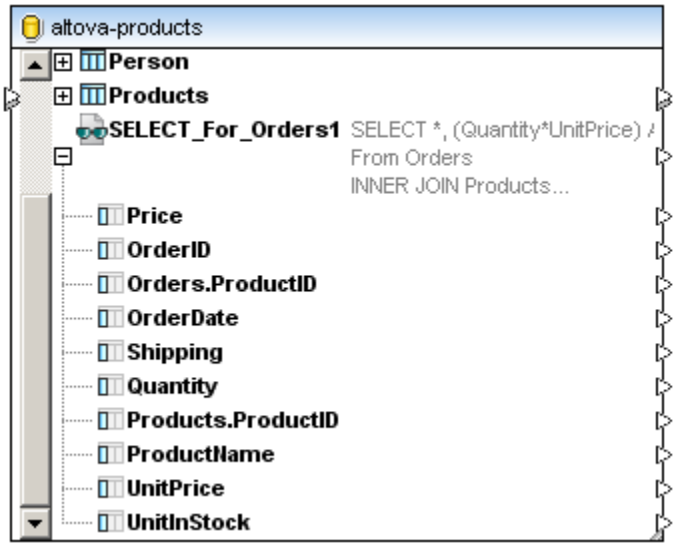
The **Global resources** pane allows you to select from previously defined global resource database aliases. Please see: Global Resources - Databases for more information.



15.15 SQL SELECT Statements as virtual tables

MapForce supports the creation of SQL SELECT statements 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 are created during the process of inserting existing database tables into MapForce.



Note:

The SELECT statement is visible within the component. To define the number of lines you want to see of the statement, select the menu option **Tools | Options**, click the General tab and enter the number of lines in the Mapping View group.

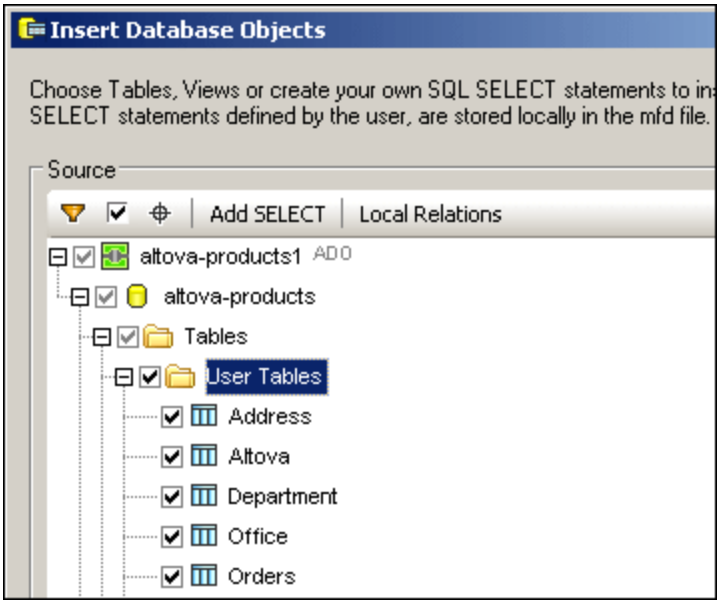
This section uses the **altova-products.mdb** file available in the ...\MapForceExamples\Tutorial\ folder. The **select-component.mfd** mapping, which shows an example is also available in the same folder.

15.15.1 Creating SELECT statements

This section uses the **altova-products.mdb** file available in the ...\MapForceExamples\Tutorial\ folder.

To create a SELECT statement in a database component:

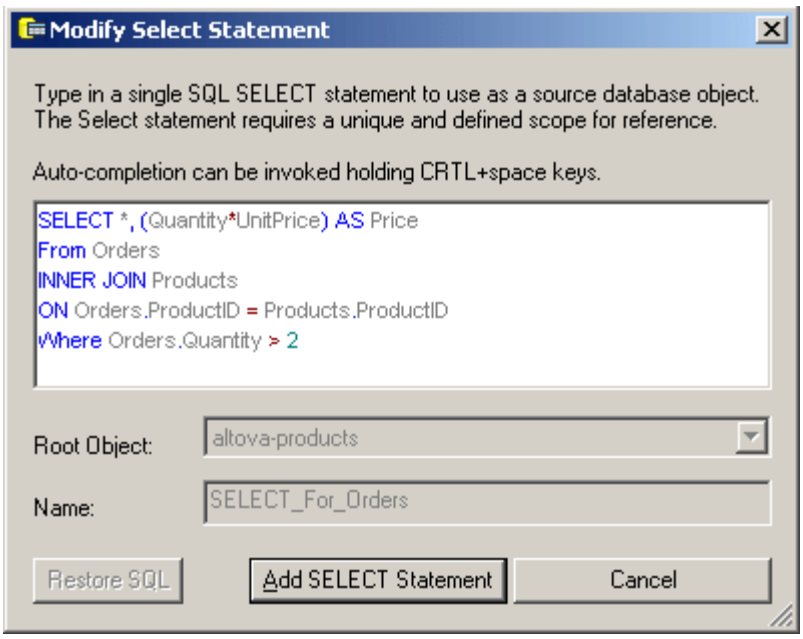
1. Click the **Insert Database** icon in the icon bar.
2. Click the **Microsoft Access** radio button, then click Next.
3. Click the Browse button to select the database you want as the data source, **altova-products.mdb** in the ...\MapForceExamples\Tutorial\ folder in this case, continue with Next.
4. Click the User tables check box to select all tables of the database.



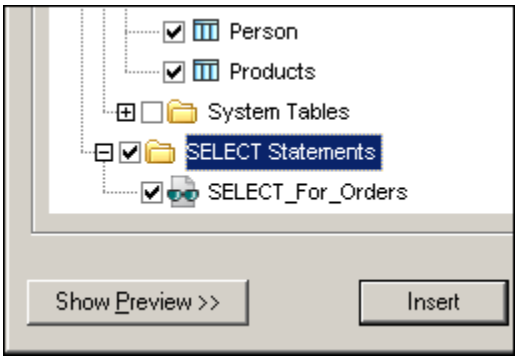
5. Right click the **Orders** table and select "Generate and add an SQL statement" from the popup menu, (or click the Add SELECT button in the icon bar).
   A default SELECT statement is already available in the dialog box.
6. Delete or edit the statement to suit you needs. The example uses the following statement:

```
SELECT *, (Quantity*UnitPrice) AS Price
From Orders
INNER JOIN Products
ON Orders.ProductID = Products.ProductID
Where Orders.Quantity > 2
```

   Please note that all calculated expressions in the SELECT clause need to have a unique correlation name (like "AS Price" in this example) to be available as a mappable item.

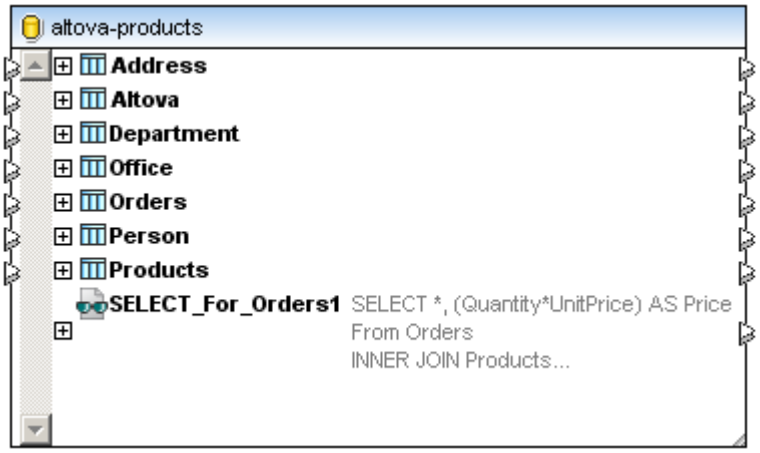


7. Click the **ADD SELECT statement** button to insert the component.



The "virtual table" has now been addded to the SELECT Statements folder in the dialog box.

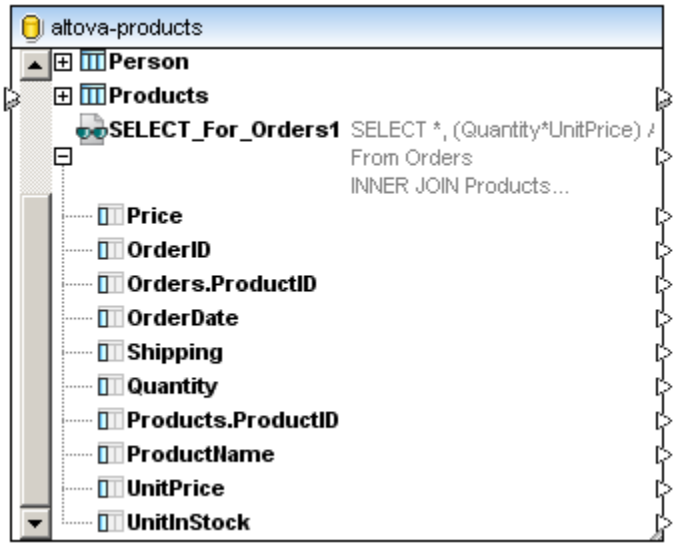
8. Click the Insert button to insert the component into the mapping area.



What was inserted:

- All tables of the Altova database, Address to Products.
- The **Select_For_orders** virtual table (result-set table), containing all selected columns or expressions defined by the SELECT statement.

1. Click the expand icon below the **Select_For_Orders** item.



The column items of both tables defined by the inner join are available to be mapped.

Please note:

The **Price** field is the product of the two fields, Quantity and UnitPrice, and is actually a correlation name: SELECT *, (Quantity*UnitPrice) AS Price.

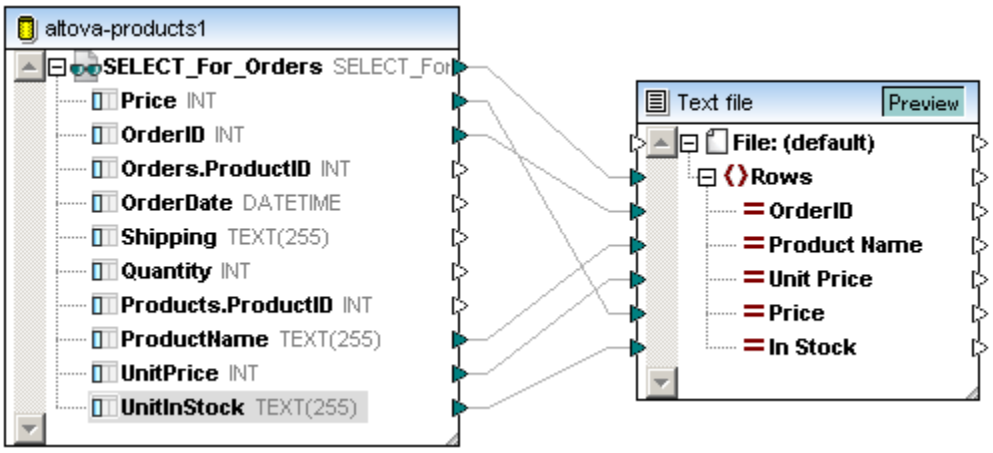
15.15.2 SELECT statement example

The previous section showed how to insert a SELECT statement in a database component, and what fields the component contains when it is inserted into the mapping area. This example uses the same database used previously i.e. **altova-products.mdb** available in the ...\MapForceExamples\Tutorial\folder.

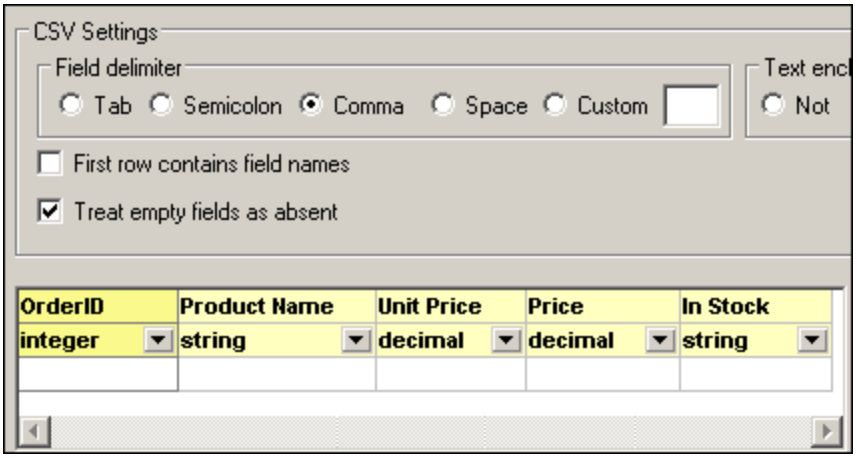
What this mapping does:

- Creates a Select statement component "altova-products" using an Inner join statement.
- Maps the database data to an empty Text/CSV file.

1. Insert the SELECT statement as discussed previously.



2. Insert a Text component, name the items and select the appropriate types for the various fields, i.e. integer, string etc.



3. Connect the two components as shown above and click the Output button to see the result.

```
1   2,Layer designer,5000,15000,no
2   4,Visualizer,3000,21000,yes
3
```


Chapter 16

Mapping CSV and Text files

16 Mapping CSV and Text files

MapForce includes support for the mapping of flat file formats, i.e. CSV files and Text files as both source and target components. Please note that you need to select one of the programming languages (Java, C#, or C++), or BUILTIN as the mapping output, to be able to work with text files.

Supported text file formats are:

- CSV files (comma-separated values) - also for other delimiters than comma
- FLF files (fixed-length fields)
- Other text files whose structure is defined in a FlexText template (Enterprise Edition only)

This bi-directional mapping support includes:

- XML schema to/from flat file formats
- Database to/from flat file formats
- There are two ways that mapped flat file data can be generated/saved:
- By clicking the Output tab which generates a preview using the Built-in execution engine, selecting the menu option **Output | Save output file**, or clicking the  icon, to save the result
- By selecting **File | Generate code in | Java, C#, or C++** then compiling and executing the generated code.

Please note:

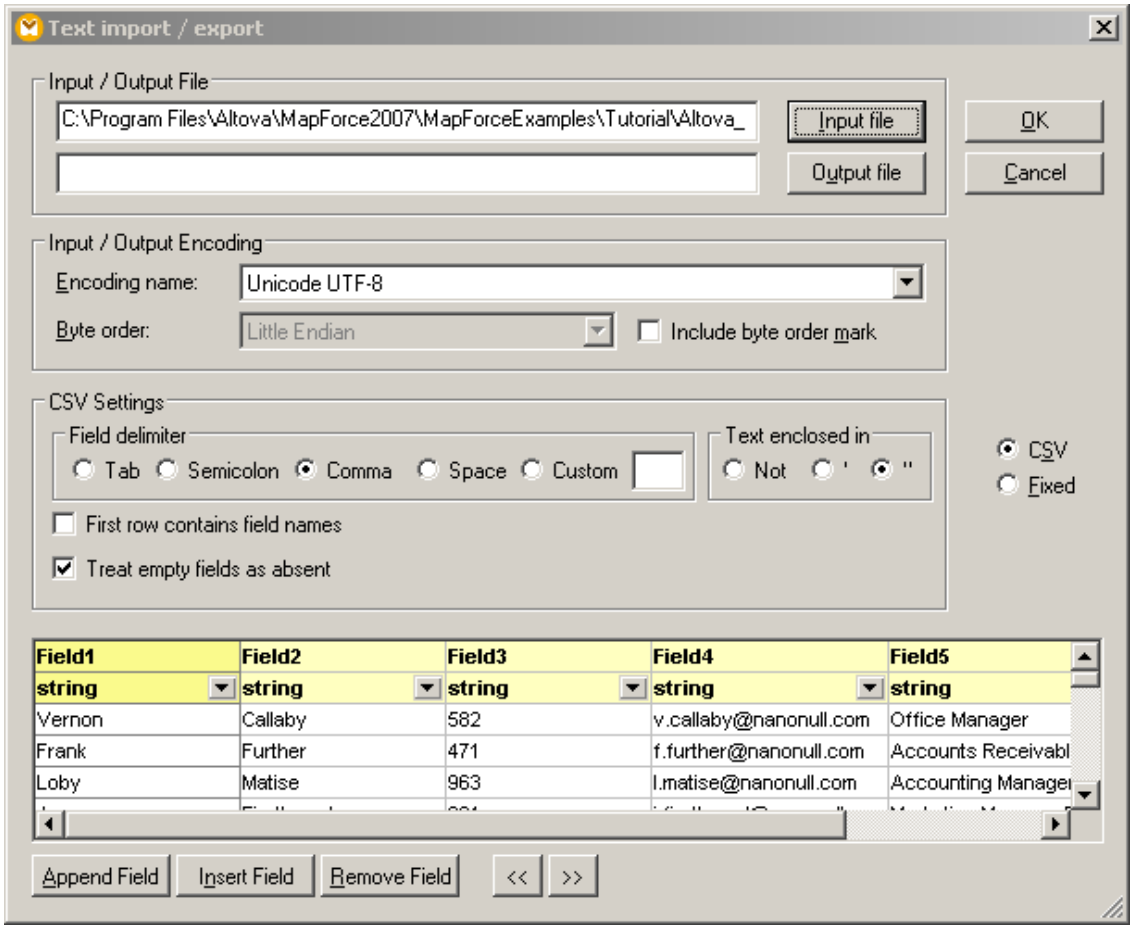
All the following examples using CSV files as source or target components, can also be accomplished with Fixed length text files. The only difference is that the field lengths have to be defined manually, please see "Mapping Fixed Length Text files" on how define field lengths.

16.1 Mapping CSV files to XML

This example maps a simple CSV file to an XML file, based on the MFCompany.xsd schema file. All the files used in the following examples are available in the **...\MapForceExamples\Tutorial** folder.

- Having made sure you selected **Java**, **C#**, **C++**, or **BUILTIN** by clicking the respective toolbar icon.

1. Select the menu option **Insert | Text file**, or click the "Insert Text file" icon  This opens the Text import / export dialog box, in which you can select the type of file you want to work with CSV, or Fixed length files. The CSV radio button is active by default.



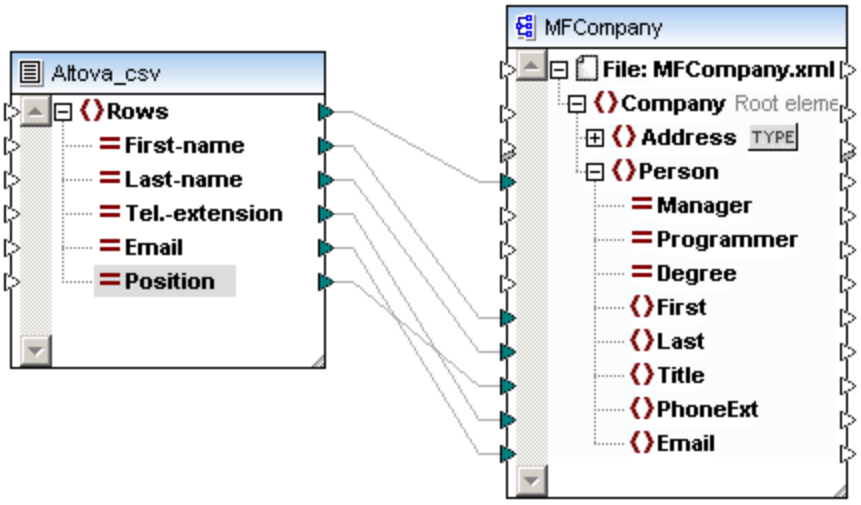
2. Click the **Input file** button and select the CSV file, e.g. Altova_csv.csv. The file contents are now visible in the Preview window. Please note that the Preview window only displays the first 20 rows of the text file.
3. Click into the **Field1** header and change the text, e.g. First-name. Do the same for all the other fields, e.g. Last-name, Tel.-extension, Email, and Position.

First-name	Last-name	Tel.-extension	Email	Position.
string	string	string	string	string
Vernon	Callaby	582	v.callaby@nanonull.com	Office Manag
Frank	Further	471	f.further@nanonull.com	Accounts Re
Loby	Matise	963	l.matise@nanonull.com	Accounting M
Joe	Firstbread	621	j.firstbread@nanonull.com	Marketing Ma
Susi	Sanna	753	s.sanna@nanonull.com	Art Director

Please note:

Hitting the **Tab** keyboard key, allows you to cycle through all the fields: header1, field type1, header2 etc.

3. Click the OK button when you are satisfied with the settings.
   The CSV component is now visible in the Mapping.
4. Select the menu option **Insert | XML/Schema file** and select **MFCompany.xsd**.
5. Click **Skip**, when asked if you want to supply a sample XML file, and select Company as the root element.



6. Map the corresponding items of both components, making sure to map the **Rows** item to the **Person** item in the schema target, then 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 have been successfully mapped to an XML file.

Please note:

The connector from the **Rows** item in the CSV file, to the Person item in the schema is essential, as it defines which elements will be iterated through; i.e. for each Row in the CSV file a new Person element will be created in the XML output file.

Please see the examples that follow, on how the **Rows** item influences the output if you are mapping **to** a CSV, or fixed length text file.

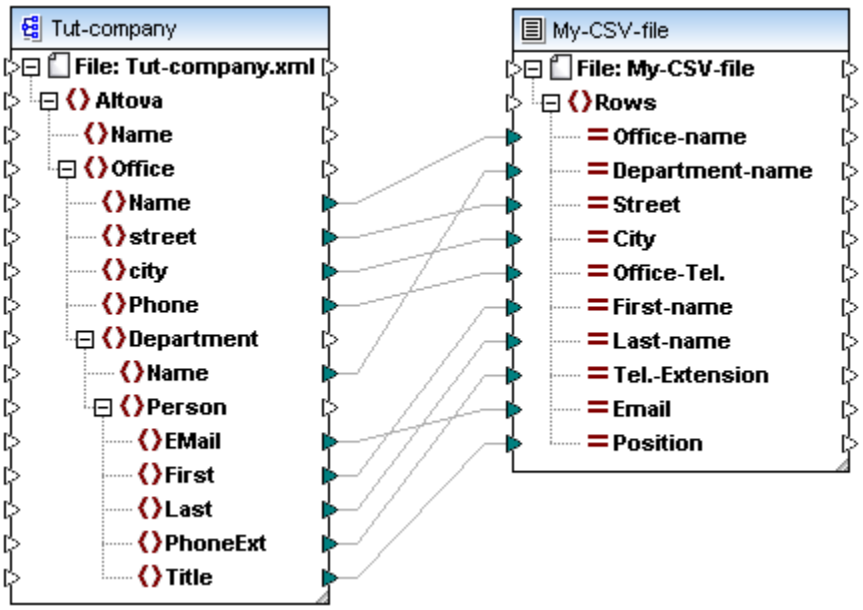
16.2 XML to CSV, iterating through items

This example is available in the ...\MapForceExamples\Tutorial\ folder as **Tut-xml2csv.mfd**.

- **Tut-company.xsd** and **Tut-company.xml** are the source schema and XML data source respectively.
- "My-CSV-file" is the text file component. The name is entered in the "Input file" field of the Text import /export dialog box.

  The mapping example is for illustration purposes only, it is not supposed to be a real-life example.

The diagram below shows how you would generally expect to map an XML file to a CSV file.

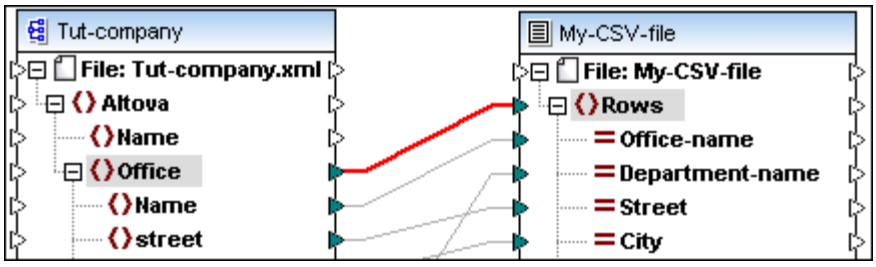


Clicking the Output tab produces the result you see below, which may not be what you expect, we only see output for one office.

```
1   "Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Ottow
2
```

In order to be able to iterate through all offices and have the output appear in the CSV file, it is necessary to connect Office to Rows. What this means is: for each Office item of the source XML, create a Row in the target CSV file. MapForce allows you to specify the field, or item which is to act as the "root"/iterator for the output using the **Rows** item.

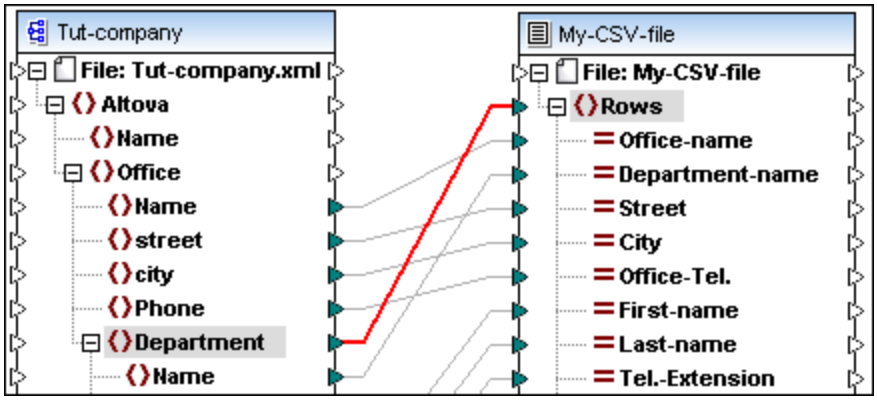
Mapping the **Office** item to the **Rows** item, results in all individual Offices (and mapped items) being output.



The Office items are output in the source file sequence.


```
1   "Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,558833
2   "Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Otto
3
```

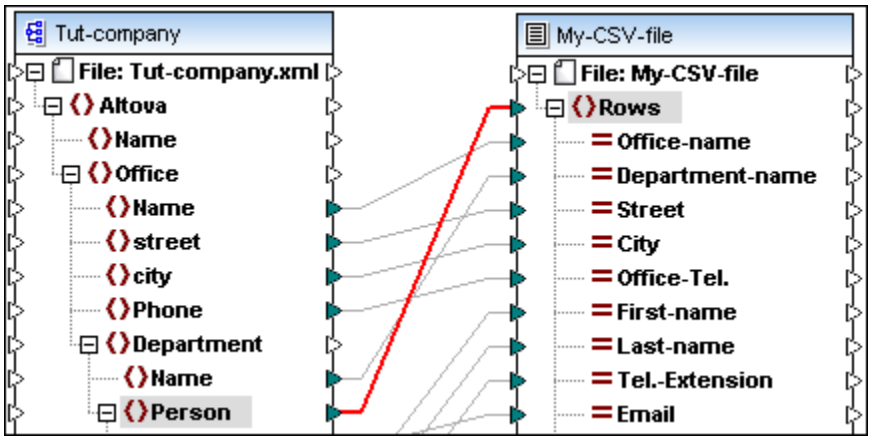
Mapping **Department** to the **Rows** item results in all of the Departments being output.



The Departments are output in the source file sequence, for each Office.

```
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
```

Mapping **Person** to the **Rows** item results in all the Persons being output.



The Persons are output in the source file sequence, i.e. each Person within each Department, for each Office.

```
1   "Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Albert,Aldrich,582,A.Ald
2   "Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Bert,Bander,471,b.bander
3   "Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Clive,Clovis,963,c.clovi
4   "Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,5588339,Dave,Durne
5   "Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,5588339,Eve,Ellas,
6   "Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,Fred,Fortunas,95
7   "Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,Gerry,Gundall,65
```

16.3 Creating hierarchies from CSV and fixed length text files

This example is available in the ...\MapForceExamples\Tutorial\ folder as **Tut-headerDetail.mfd**.

The example uses a CSV file with fields that define the specific record types, and has the following format:

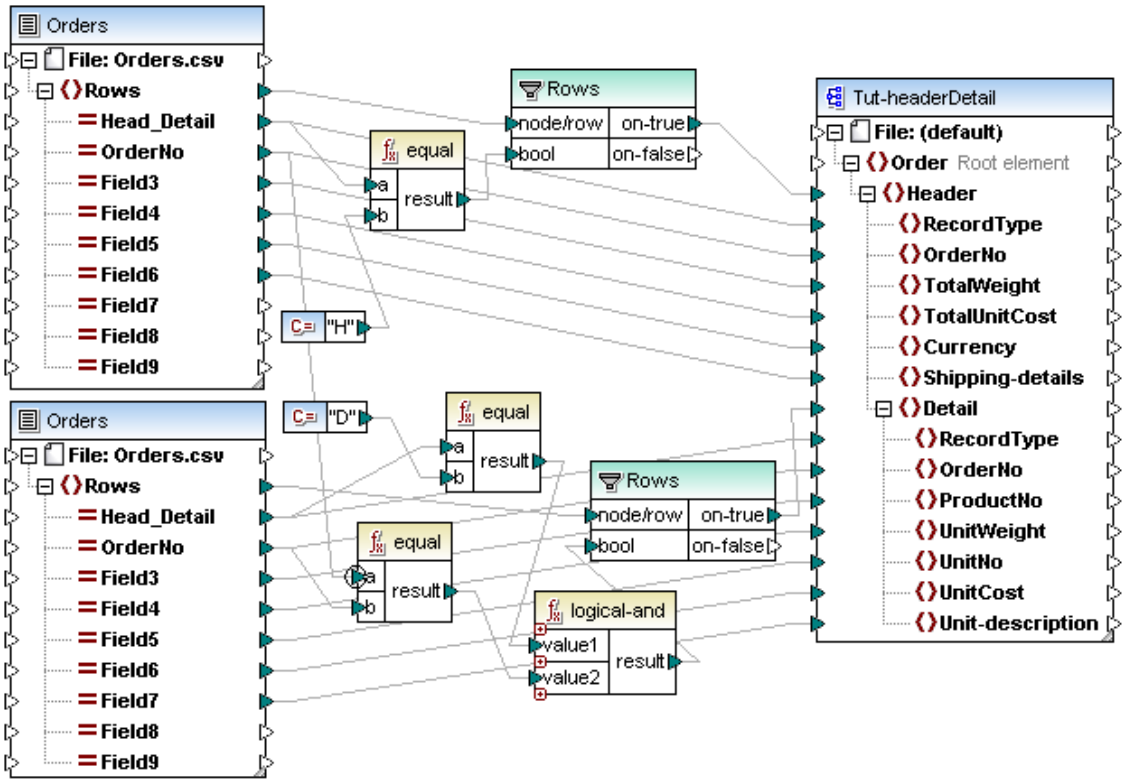
- Field 1: H defines a header record and D a detail record.
- Field 2: A common/key for both header and detail records.
- Each header/detail record is on a separate line.

Creating hierarchical XML structures from flat files using "Key" fields
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,,
```

Aim of the mapping is to:

- Map the flat file CSV to an hierarchical XML file, and
- Filter out the Header records, designated with an H, and
- Associate the respective detail records, designated with a D, with each of the header records



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.

Notes on the mapping:
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 the H records. 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 function.

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>
```

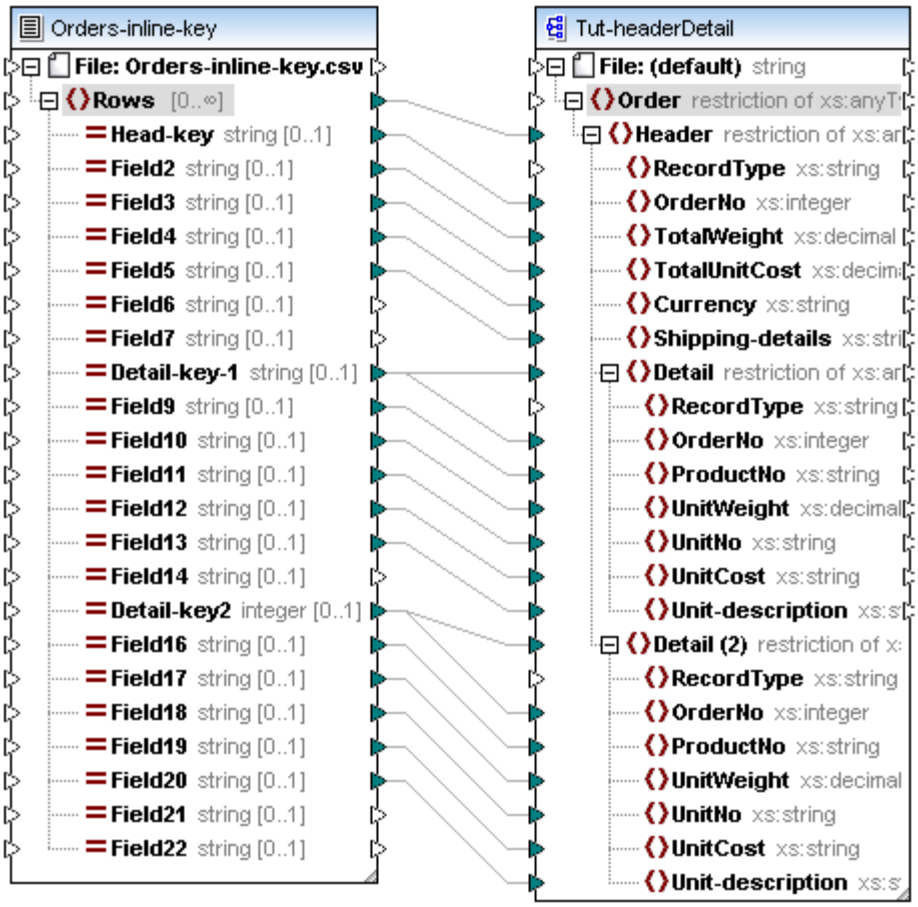
The second example uses a slightly different CSV file and is available in the ...\MapForceExamples\Tutorial\ folder as **Head-detail-inline.mfd**. however:

- 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,,
```

Please note:

- 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 above example.

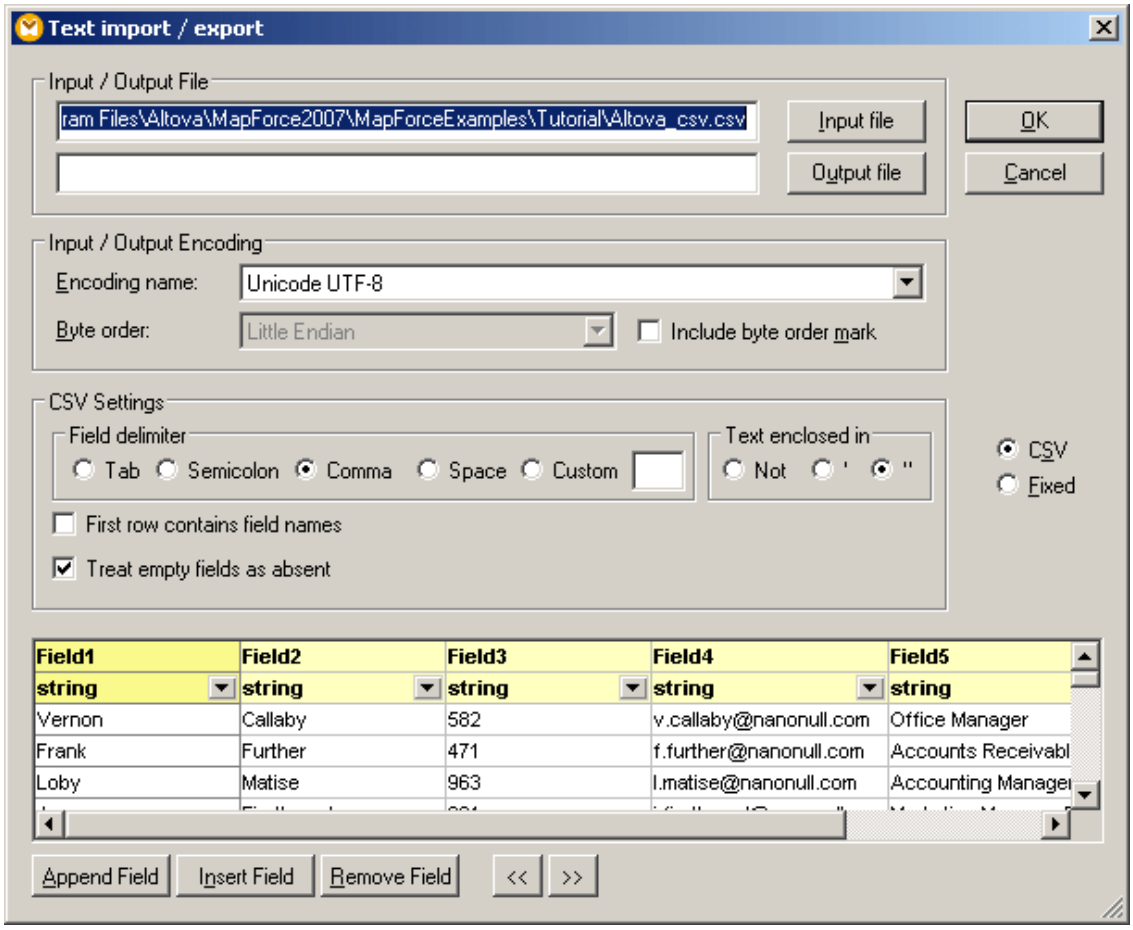


16.4 CSV file options

Right click the **Altova_csv** component and select **Component Settings** to open the dialog box.

CSV Text import / export options:
When defining field formats in this dialog box, type checking of the respective fields is automatically performed. If the input data and the field format defined here to do not agree, then the data is highlighted in red. E.g. changing field2 from string to integer would make all surnames of that column appear in red.

Please note:
The field types that one can select for a specific column, are based on the default XML schema datatypes. E.g. The Date type is in the form: YYYY-MM-DD.



Input file:
Select the CSV file you want to use as the source file for this component. This file name will be used for reading example data and field information, for the output preview and for code generation.

Please note:
This field can remain empty if you are using the Text file component as a **target** for your mapping. In this case, the file encoding automatically defaults to UTF-8. You can define the field type, field names, formatting etc. and click OK to create a text file target.

Clicking the **Output** tab then allows you to **save** this text file, by clicking the "Save generated output as..." icon  including its mapped contents.

Entering a name in this text box (without using a file extension) assigns this name to the component.

Output file:
Select the target file you want to output data to, when **generating code** with MapForce. Make sure that the input and output files are different when generating code, or the source file will be overwritten when the code is executed. This option is used when generating code for Java, C++, or C#, or when saving the output file from BUILTIN (the Built-in execution engine).

File encoding:
Allows you to define/select the character encoding of the input and output text file. If there is no entry in the Input file field, then the encoding automatically defaults to UTF-8.

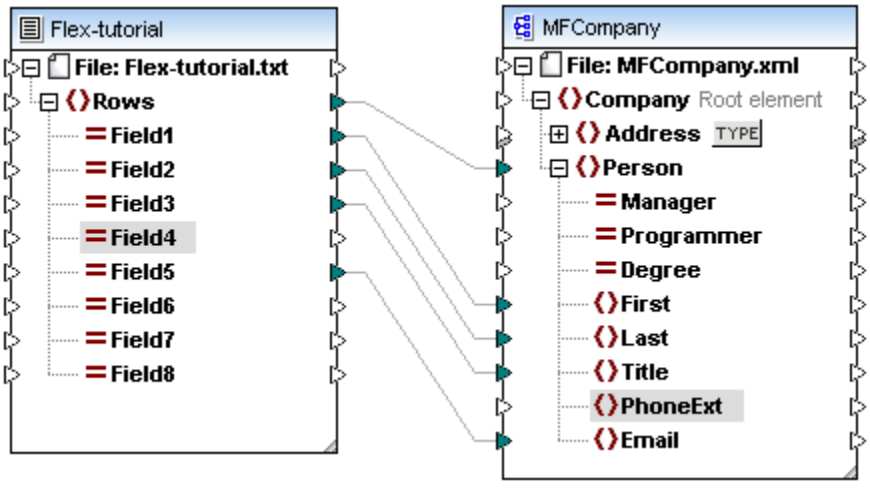
CSV Settings - Field delimiter:
Select the delimiter type for the text file (CSV files are comma delimited "," per default). You can also enter a custom delimiter in the **Custom** field.

Click into the Custom field and:

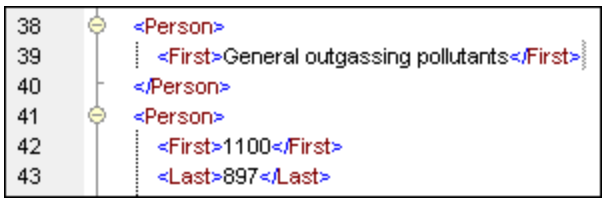
- Hit a keyboard key to enter a new value, or
- Double click in the Custom field, to mark the current value, and hit a different keyboard key to change the entry.

First row contains field names:
Sets the **values** in the first record of the text file as the column headers (visible in the preview window). The column headers then appear as the item names when the Text component is displayed in the mapping.

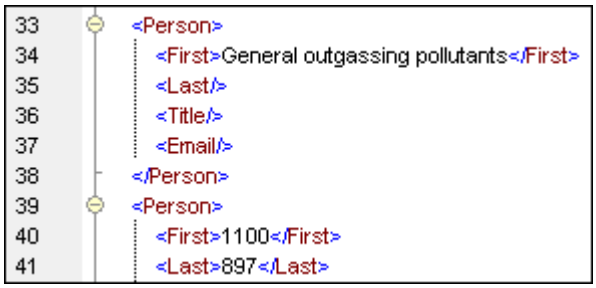
Treat empty fields as absent
Allows you to define that empty fields in the source file, will not produce a corresponding empty item (element or attribute) in the target file.



Active:
One of the fields of the source file only contains data in Field1; fields2 to 5 are empty. When active, the target items that do not receive data from the source file do not appear in the output (line 39).



Inactive:
The empty fields of the source file produce the corresponding target items in the output file, i.e. the elements, Last, Title, and Email in this example.



Please note:
The **delimiters** for the empty fields in the source file must exist however, e.g. "General outgassing pollutants**,,,,"**.

Text enclosed in:
Text files exported from legacy systems sometimes enclose text values in quotes to distinguish them from numeric values.

Select this option if the text file contains strings which include the Field delimiter that you have currently defined. The same delimiter character can then occur within a string without affecting the text file segmentation/partitioning. E.g. your fields (strings) contain a comma character "," but you are also using this character as the default CSV delimiter.

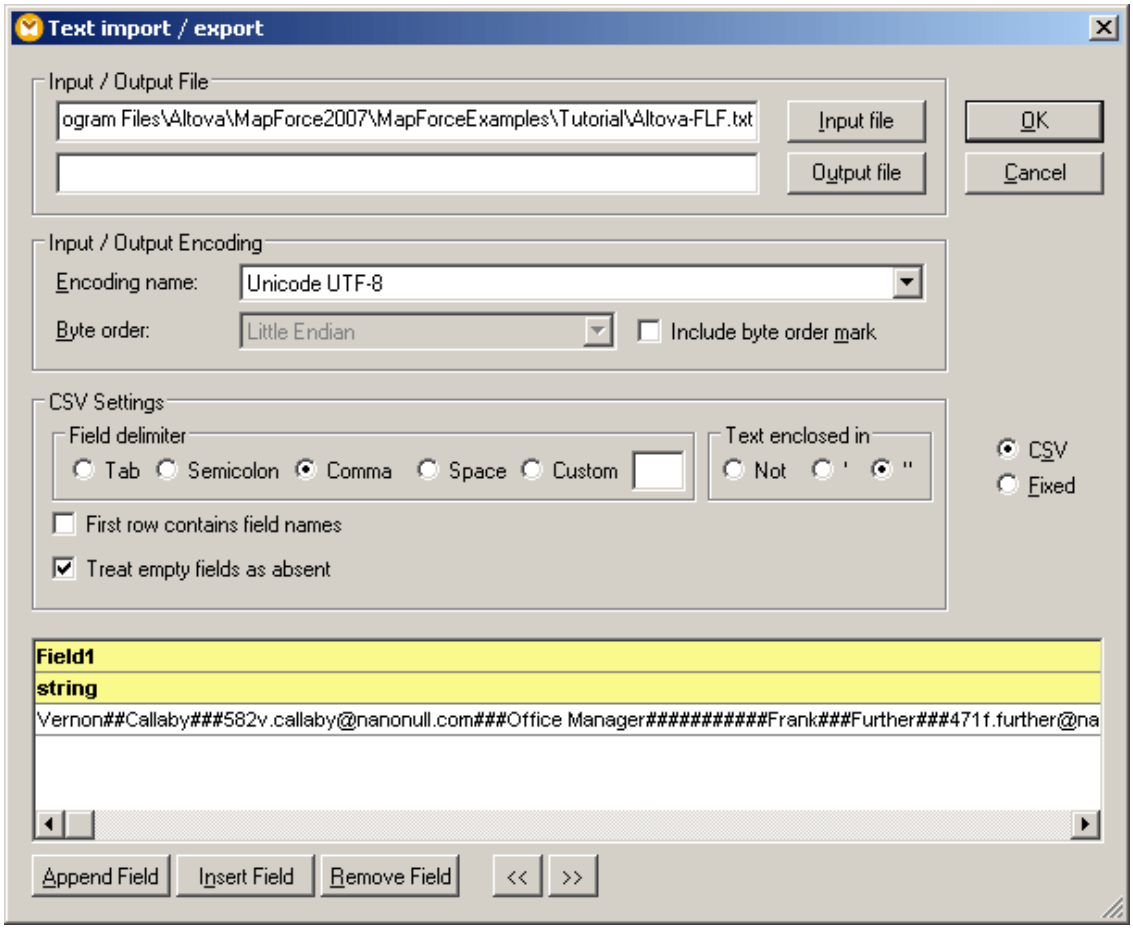
Append field, **Insert** field, **Remove** field:
Allows you to append, insert or remove fields in the preview window, which defines the structure of the CSV file.

Next / Previous
Clicking one of these buttons moves the currently active column left or right in the preview window.

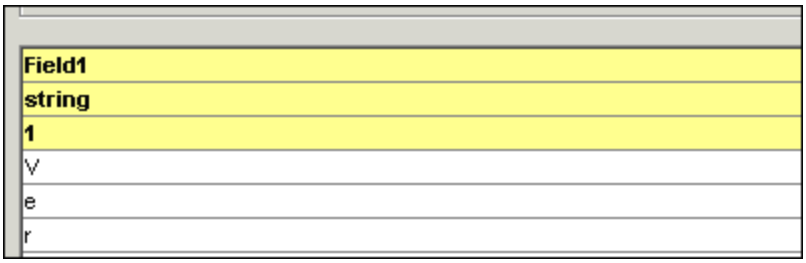
16.5 Mapping Fixed Length Text files (to a database)

This example maps a simple text file to a MS Access database. The source text file is one continuous string with no carriage returns, or line feeds. All the files used in the following examples are available in the **...\MapForceExamples\Tutorial** folder.

1. Select the menu option **Insert | Text file**, or click the insert Text file icon  .
   This opens the Text import / export dialog box, in which you can select the type of file, and specific settings, you want to work with.
2. Click the **Input file** button and select the **Altova-FLF.txt** file.
   You will notice that the file is made up of a single string, and contains fill characters of type #.



3. Click the **Fixed** radio button (below CSV).
4. Uncheck the "**Assume record delimiters present**" check box.

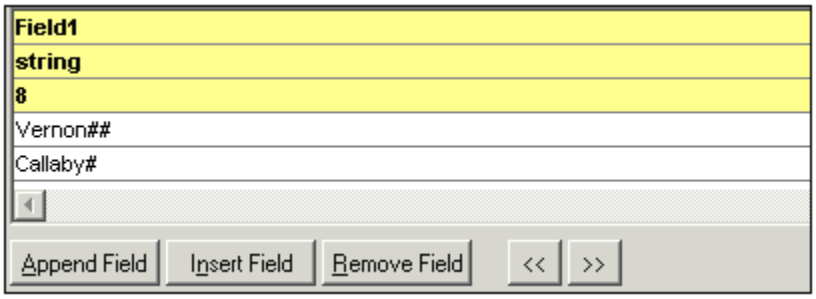


The preview changes at this point. What we now have, is a fixed format comprising of:

- a single field called "Field1"
- where the format is of type "string", and the
- field length is one character (V from person Vernon)

Field 1 now contains additional data.

5. Click into the row containing the **1** character, change the value to 8, and hit Return.

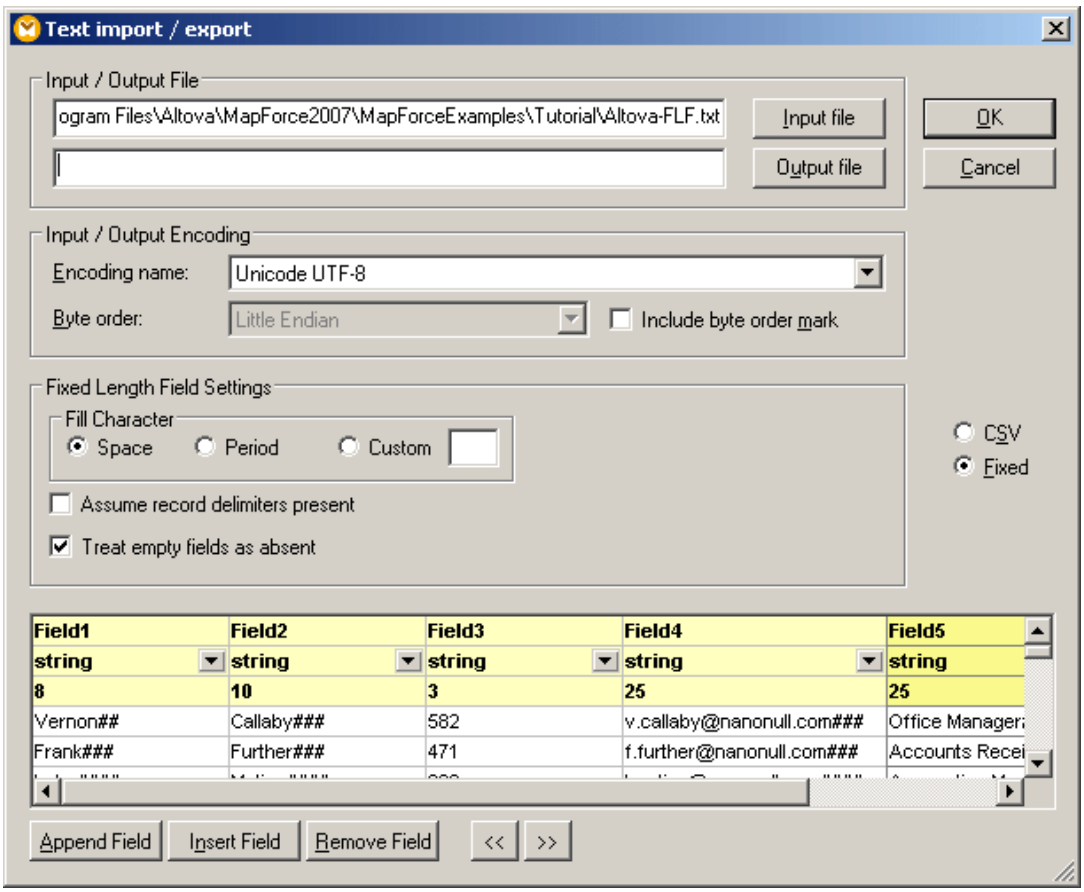


More data is now visible in the first column, which is now defined as 8 characters wide.

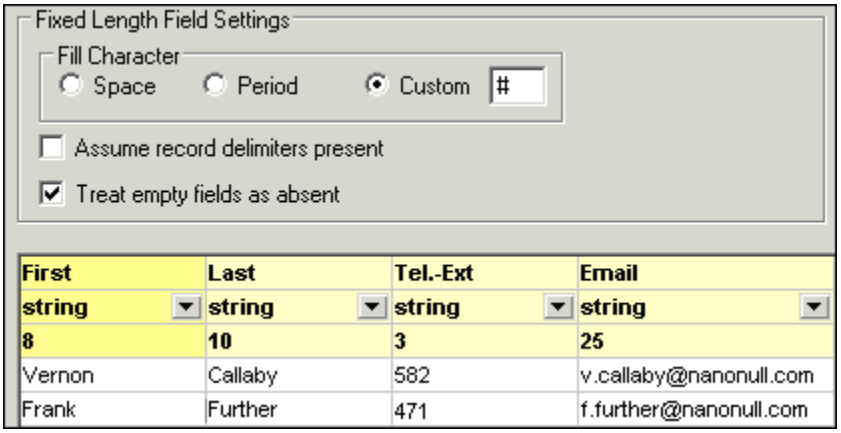
6. Click the **Append Field** button to add a new field, and make the length of the second field, 10 characters.

Field1	Field2
string	string
8	10
Vernon##	Callaby###
582v.cal	laby@nanon

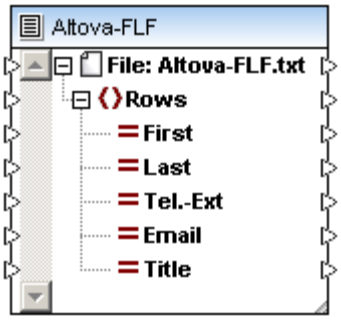
7. Use the same method to create three more fields of the following lengths: 3, 25, and 25 characters, and change the field headers to make them easier to map: First, Last, Tel.-Ext, Email, Title. The preview will then look like this:



8. Click into the Custom text box of the Fixed Length Field Settings group, and enter the hash (#) character. This has the effect of removing the identical fill character from the text file being input.

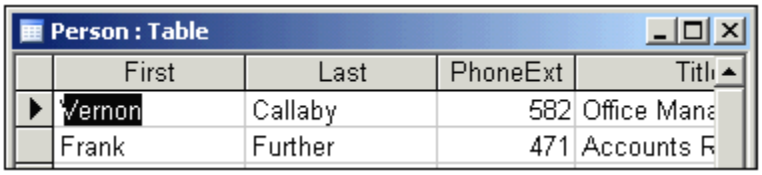


9. Click OK to complete the definition.

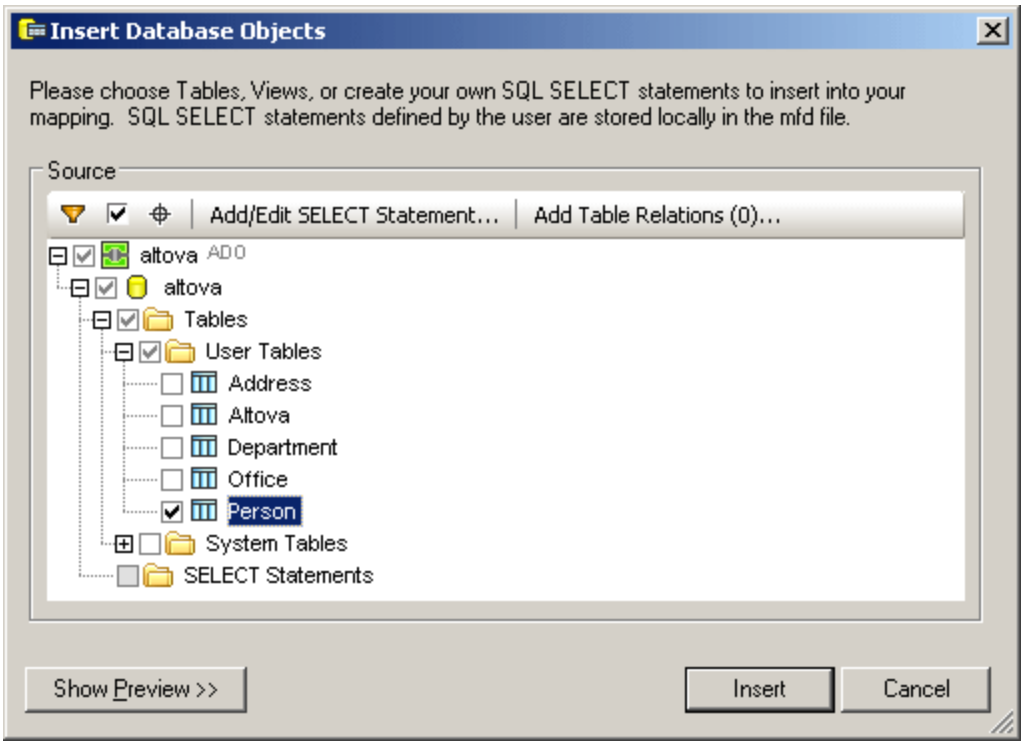


The Text file component appears in the Mapping window. Data can now be mapped to, and from, this component.

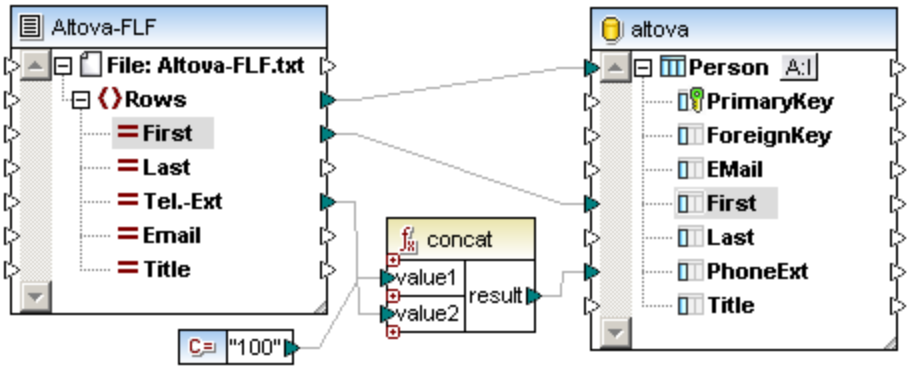
Mapping text files to a database:
This section uses the fixed length text file to update the Telephone extension entries in the **altova.mdb** database.



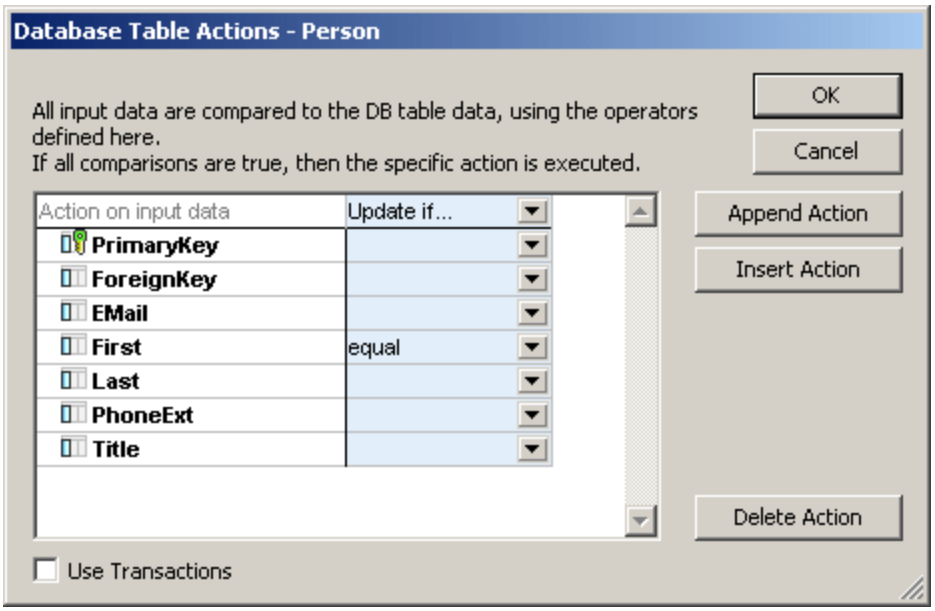
1. Select the menu option **Insert | Database,** click the Microsoft Access radio button, ten click Next.
2. Select the **altova.mdb** database available in the ...\MapForceExamples\Tutorial\ folder, and click Next.
3. Select the **Person** table by clicking the corresponding check box in the Database Tables list box.



4. Click the **Insert** button to create the database component.
5. Click the expand icon to see the table contents.
6. Drag the **concat** function from the libraries window into the Design tab.
7. Select the menu option **Insert | Constant**, click the Number radio button, and enter 100 as the new telephone extension prefix.
8. Create the mapping as shown in the graphic below.

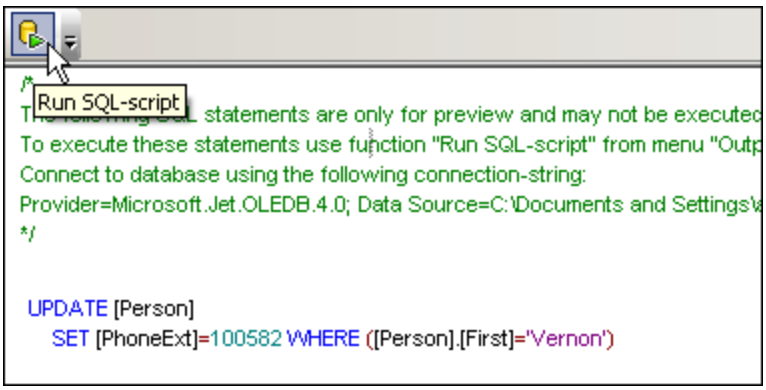


9. Right click the Person entry and select Database table actions.

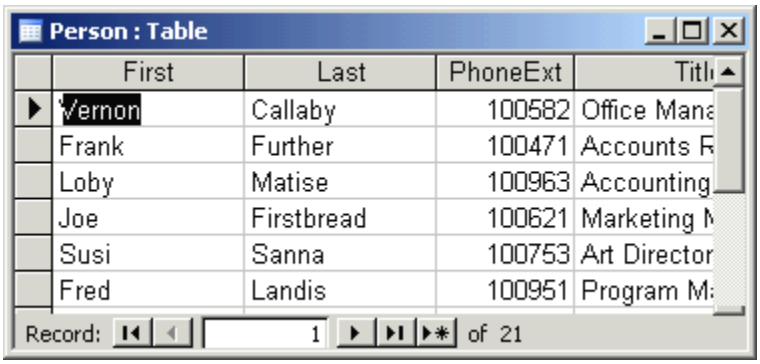


10. Click the "Action on input data" combo box and select the "**Update if...**" entry.
11. Click the combo box of the "First" row, select "equal", and click OK to confirm.

    Person table data is only updated if the **First** names of the source and database field are identical. The action taken when this is true, is actually defined by the mapping. In this case the telephone extension is prefixed by 100, and placed in the PhoneExt field of the Person table.
12. Click the Output tab to generate the SQL statement preview, then click the Run SQL-script button  to execute the SQL statements.



The telephone extension fields of all persons are updated in the database.

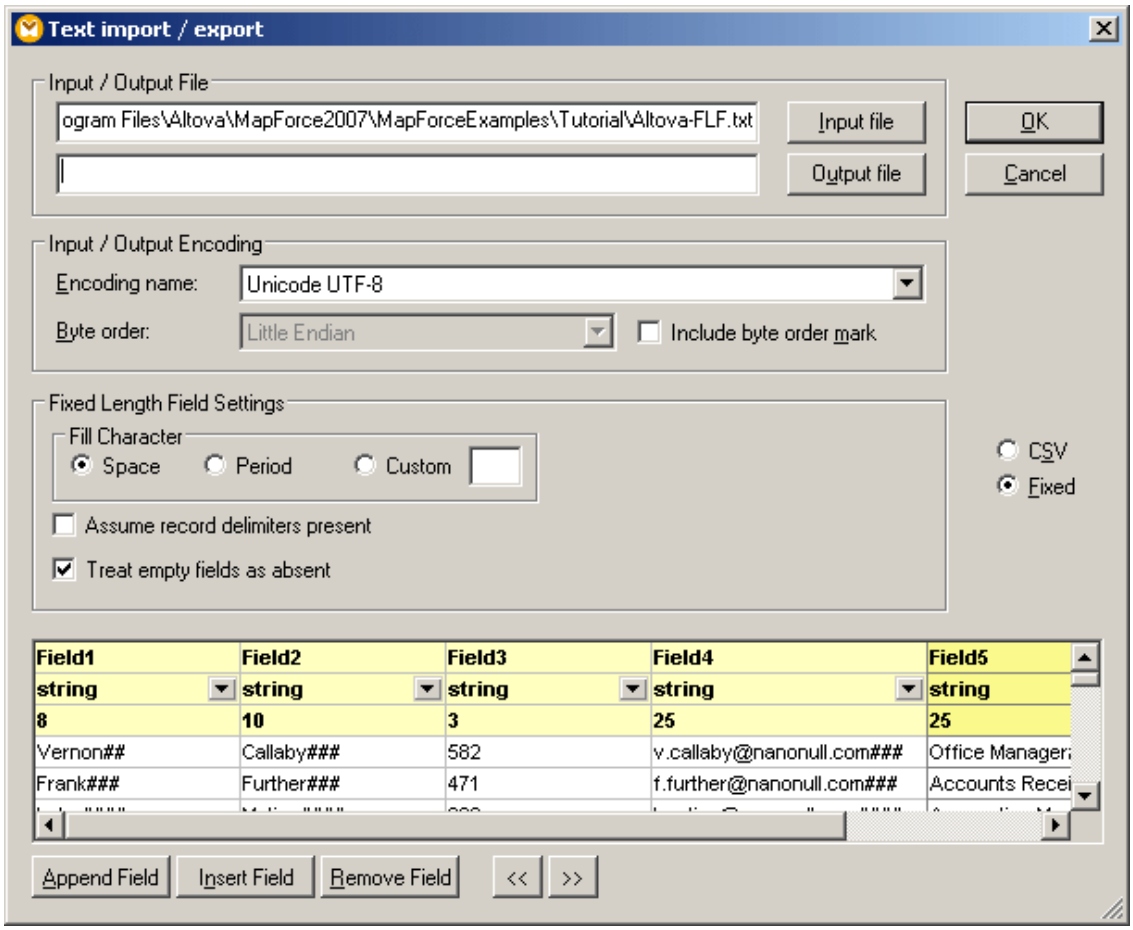


16.5.1 Fixed Length Text file options

Right click the **Altova-FLF** Text file component and select **Component Settings** to open the dialog box.

Fixed Length Text import / export options:
When defining field formats in this dialog box, type checking of the respective fields is automatically performed. If the input data and the field format defined here to do not agree, then the data is highlighted in red.

Please note:
The field types that one can select for a specific column, are based on the default XML schema datatypes. E.g. The Date type is in the form: YYYY-MM-DD.



Input file:
Select the text file you want to use as the source file for this component.

Please note:
This field can remain empty if you are using the Text file component as a **target** component for a mapping. In this case, the file encoding automatically defaults to UTF-8. You can define the field type, field names, formatting etc. and click OK to create a text file target.

Clicking the **Output** tab then allows you to **save** this text file, with its mapped output, by clicking the "Save generated output as..." icon  .

Entering a name in this text box (without using a file extension) assigns this name to the component.

Output file
Select the target file you want to output data to, when **generating code** with MapForce. Make sure that the input and output files are different when generating code, or the source file will be overwritten when the code is executed. This option is used when generating code for Java, C++, C#, or when saving the output file from BUILTIN (the Built-in execution engine).

File encoding
Allows you to define/select the character encoding of the input and output text file. If there is no entry in the Input file field, then the encoding automatically defaults to UTF-8.

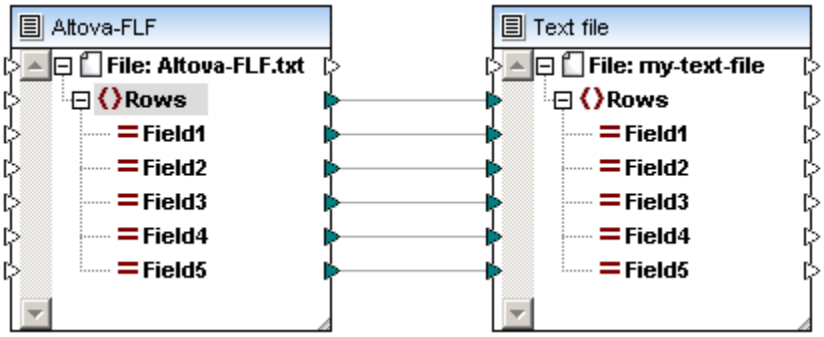
Fill Character
This option allows you to define the characters that are to be used to complete, or fill-in, the rest of the (fixed) field when the incoming data is shorter than the respective field definitions. The custom field allows you to define your own fill character in the Custom field.

Stripping fill characters:
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!

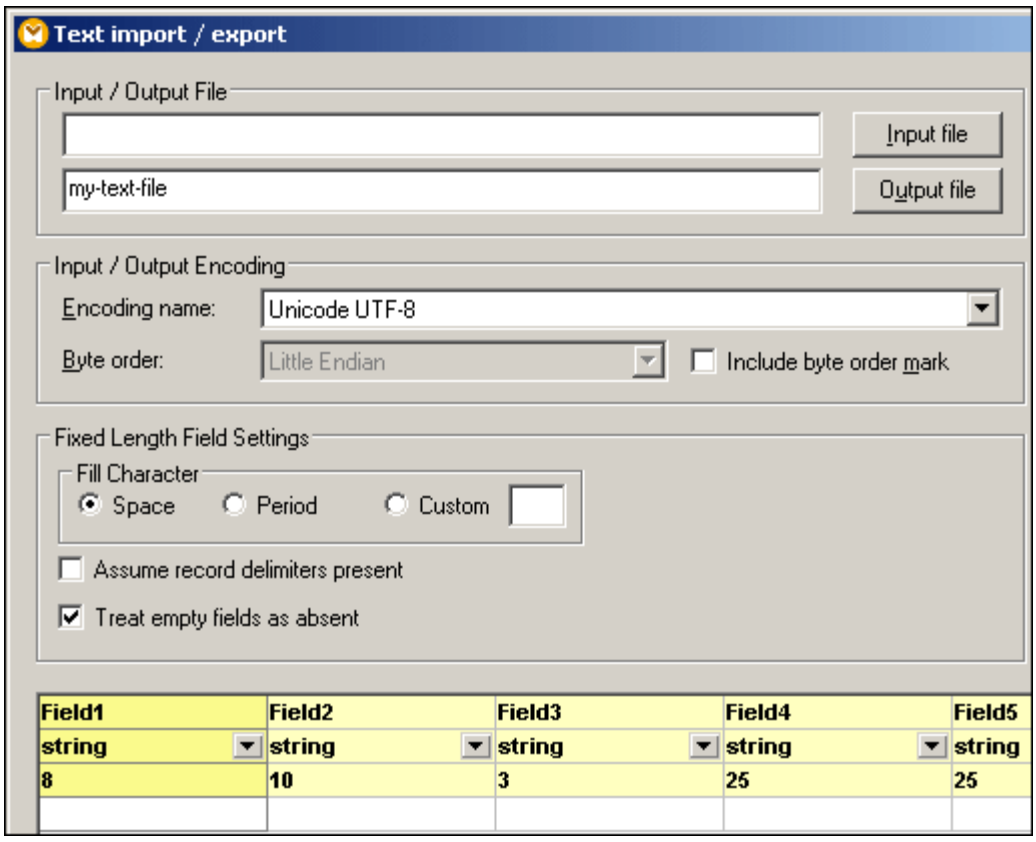
You can also enter a custom fill character in the **Custom** field. Click into the Custom field and:

- Hit a keyboard key to enter a new value, or
- Double click in the Custom field, to mark the current value, and hit a different keyboard key to change the entry.

Assume record delimiters present:
If a fixed length text file (single string) is the data source for another fixed length text file (mapping of two text files), then setting this option in the **target** file, creates new lines after the last column of the target has been filled.



In the example above the Altova-FLF text file is mapped to an empty target text file, my-text-file.



Please note:

- There is no **Input file** entry, which means that this text component only receives data from the mapped source component.
- Field lengths have been defined to correspond to the field lengths in the data source "Altova-FLF".
- No data can be seen in the preview, as the target component is not based on an existing text file.
- Clicking the Output tab, displays the mapped data.

Check box "Assume record delimiters present"

- if checked,
  a new record is created after the sum of the defined field lengths, i.e. in this case all fields add up to 71 characters, a new record will be created for character 72.

```
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#
```

- if unchecked,
  the mapped data appears as one long string, including the defined fill characters.

```
1  Vernon##Callaby###582v.callaby@nanonull.com###Office Manager##########Frankx
```

Treat empty fields as absent
Allows you to define that empty fields in the source file, will not produce a correspondingly

empty item (element or attribute) in the target file.

Active:
When active, the target items that do not receive data from the source file do not appear in the output.

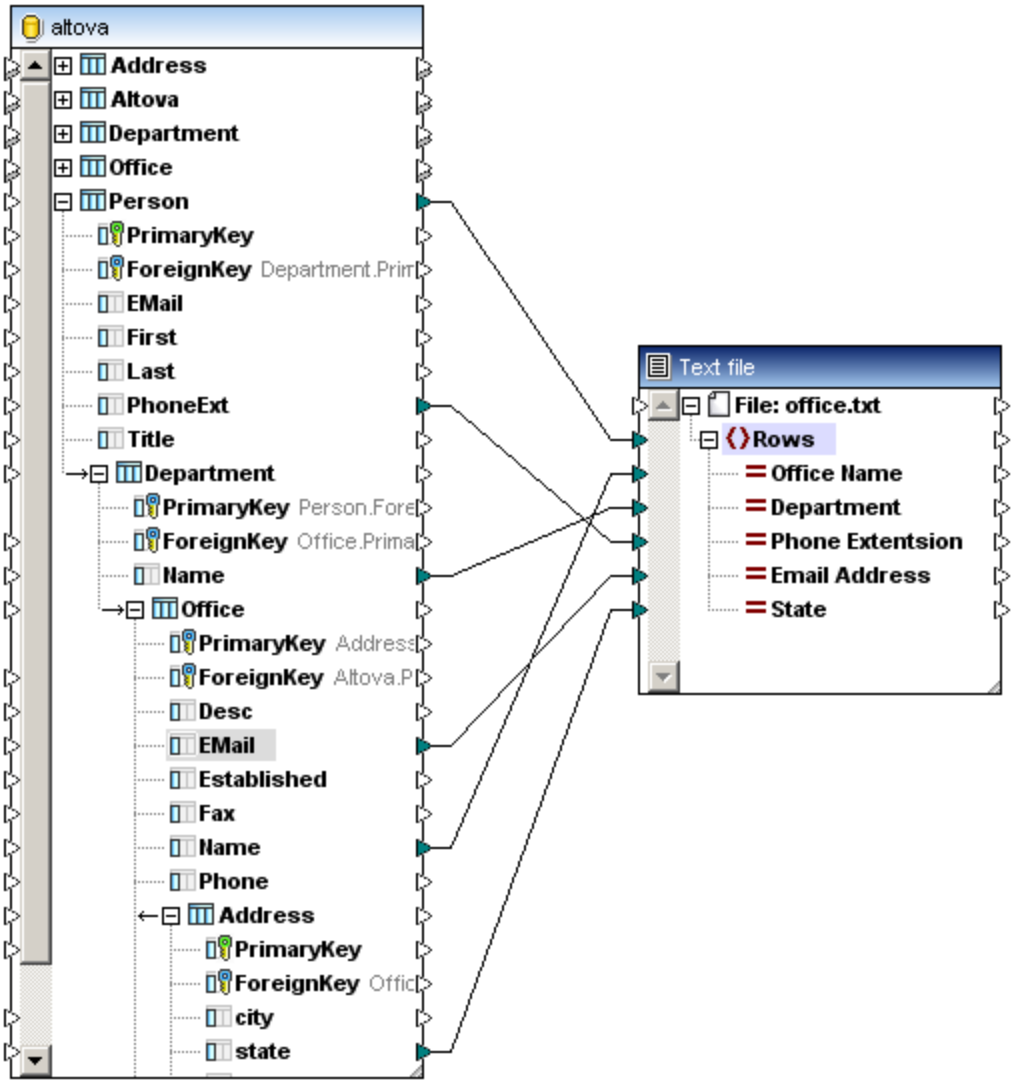
Inactive:
The empty fields of the source file produce the corresponding target items in the output file.

Append field, **Insert** field, **Remove** field:
Allows you to append, insert or remove fields in the preview window, which defines the structure of the fixed length file.

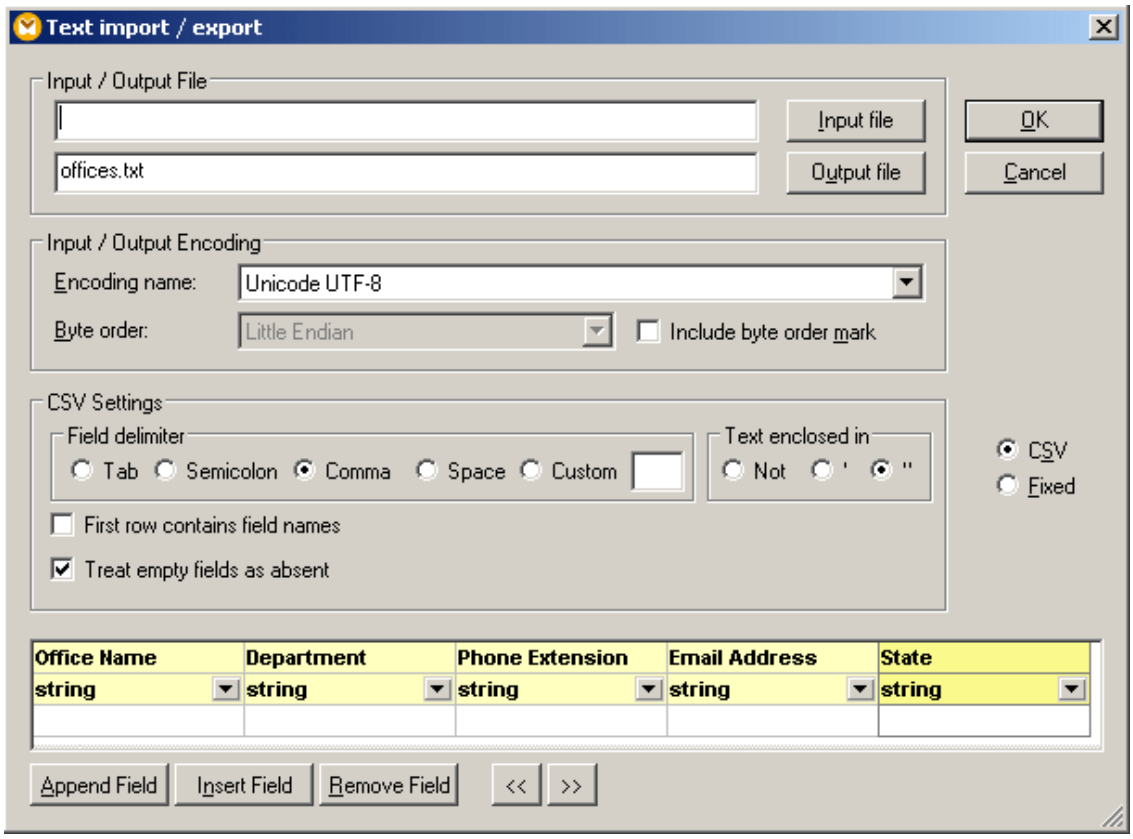
Next / **Previous**
Clicking one of these buttons moves the currently active column left or right in the preview window.

16.6 Mapping Database to CSV/Text files

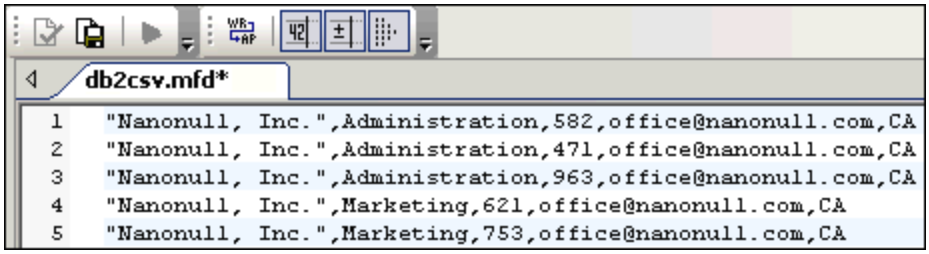
This example maps a simple MS Access database, altova.mdb, to a CSV file. The **altova.mdb** file is available in the ...\MapForceExamples\Tutorial\ folder.



The Offices.txt file entry, entered in the Output file field, is the name that is automatically supplied when you click the "Save generated output" icon from the Output tab.



Click the "Save generated output" icon to generate/output the text file.



Chapter 17

HL7 v3.x to/from XML schema mapping

17 HL7 v3.x to/from XML schema mapping

Support for HL7 version **3.x** is automatically included in MapForce 2011 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 MapForce Libraries page of the altova website. Select the Custom Setup in the installer, to only install the HL7 V3 components and XML Schemas.

Location of HL7 XML Schemas after installation:

Windows XP machine:	“C:\Program Files\Altova\Common2011\Schemas\ hl7v3“
Windows Vista machine:	“C:\Program Files\Altova\Common2011\Schemas\ hl7v3“
Windows7 machine:	“C:\Program Files\Altova\Common2011\Schemas\ hl7v3“

If a 32-bit MapForce application is used on a 64-bit operating system, then the location is “C:\Program Files(x86)\Altova\Common2011\Schemas\ hl7v3“.

HL7 documents can be used as source and target components in MapForce. This data can also be mapped to any number of XML schema, database or other components.

Chapter 18

Libraries and Functions

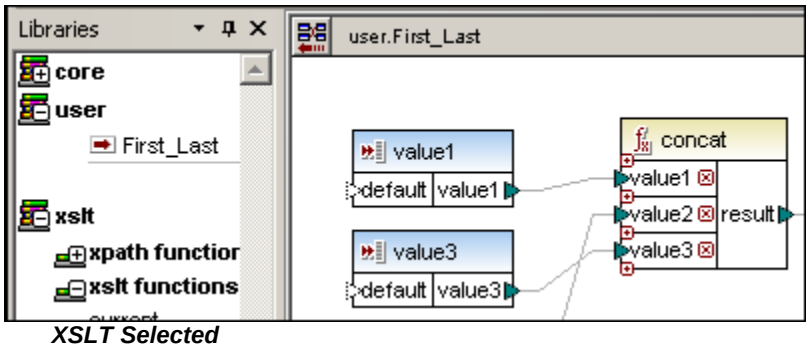
18 Libraries and Functions

The following sections describe how to define your own user-defined functions as well as libraries for the various programming languages.

18.1 Defining User-defined functions

MapForce allows you to create user-defined functions visually, in the same way as in the main main mapping window.

These functions are then available as function entries in the Libraries window (e.g First_Last), and are used in the same way as the currently existing functions. This allows you to organize your mapping into separate building blocks, and reuse them in the same, or different mappings.



XSLT Selected

User-defined functions are stored in the ***.mfd** file, along with the main mapping.

A user-defined function uses **input** and **output components** to pass information from the main mapping (or another user-defined function) to the user-defined function and back.

User-defined functions can contain "local" source components (i.e that are within the user-defined function itself) such as XML schemas or databases, which are useful when implementing lookup functions.

User-defined functions can contain any number of input and outputs where any of these can be in the form of: simple values, XML nodes, or databases.

User-defined functions are useful when:

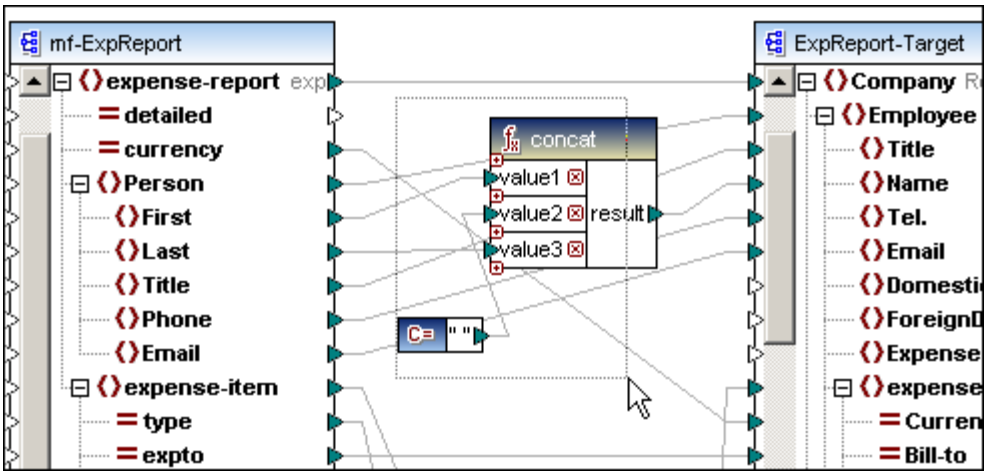
- combining multiple processing functions into a single component, e.g. for formatting a specific field or looking up a value
- reusing these components any number of times
- importing user-defined functions into other mappings (by loading the mapping file as a library)
- using inline functions to break down a complex mapping into smaller parts that can be edited individually
- mapping **recursive schemas** by creating recursive user-defined functions

User-defined functions can be either built from scratch, or from functions already available in the mapping tab.

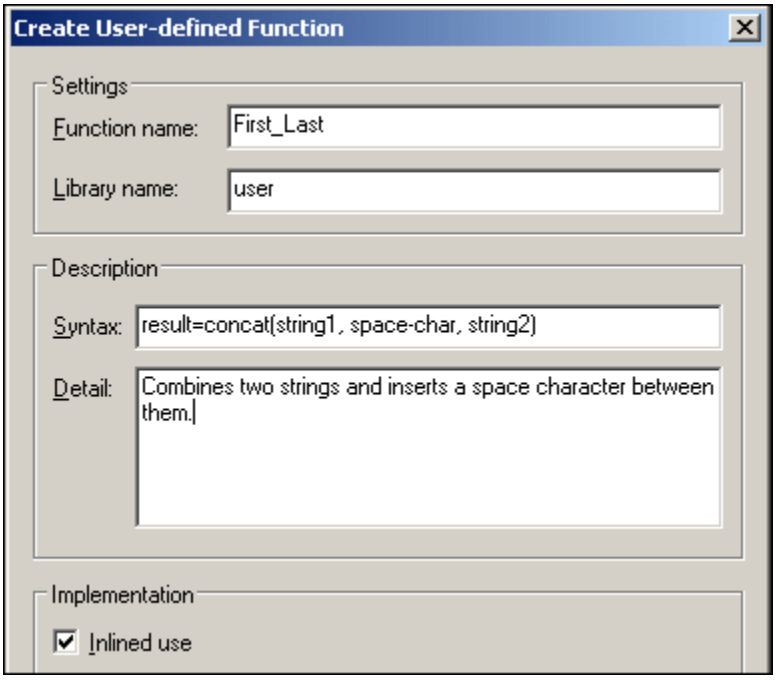
This example uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples\Tutorial\ folder.

To create a user-defined function from existing components:

1. Drag to mark both the concat and the constant components (you can also hold down the CTRL key and click the functions individually).

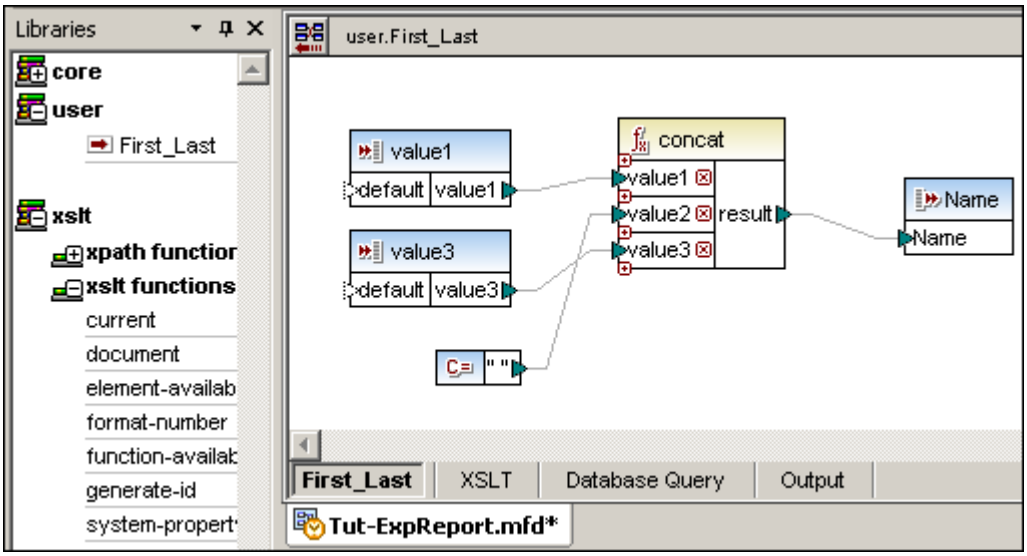


2. Select the menu option **Function | Create User-Defined Function from Selection**.
3. Enter the name of the new user-defined function (First_Last).
   Note: valid characters are: alphanumeric, a-z, A-Z, 0-9 as well as underscore "_", hyphen/dash "-" and colon ":".
4. Use the Syntax and Detail fields 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.



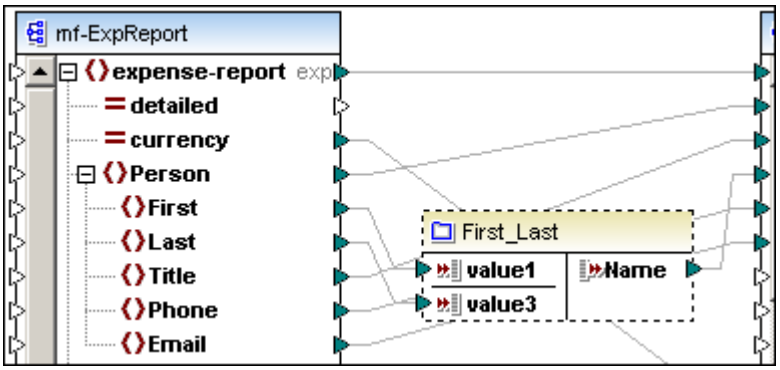
The individual elements that make up the function group appear in a tab with the

function name. The new library "user" appears in the Libraries pane with the function name "First_Last" below it.



XSLT Selected

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

To open a user-defined function:

Do one of the following:

- Double-click the title bar of a user-defined function component
  or
- 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.

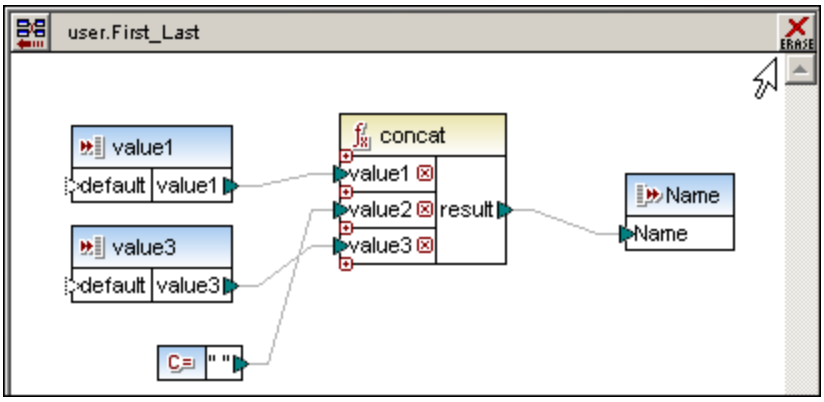
 The Home button returns you to the main mapping tab from within the user-defined function.

 The Back button takes you back through your history

 The Forward button moves you forward through your history

To delete a user-defined function from a library:

1. Double click the specific user-defined function in the Libraries window. The user-defined function is visible in its tab.
2. Click the **Erase** button in the top right of the title bar to delete the function.



Reusing - exporting and importing User-defined functions:

User-defined functions, defined in one mapping, can be imported into any other mapping:

1. Click the **Add/Remove Libraries** button, at the base of the Libraries pane, click the Add button and select a ***.mfd** file that contains the user-defined function(s) you want to import.

   The user-defined function now appear in the Libraries window (under "user" if that is the default library you selected). You can of course enter anything in the "Library name" field when defining the user-defined function.
2. Drag the imported function into the mapping to make use of it.

   Library Names

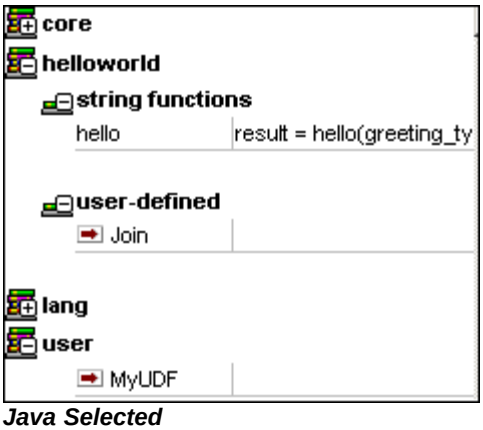
   Note: It is possible to use the same library name for user-defined functions in multiple *.mfd files and/or custom libraries (see Adding custom libraries section).

   Functions from all available sources will appear under the same library name/header in the Libraries pane. However, only the functions in the currently active document can be

edited by double-clicking.

In the following example:

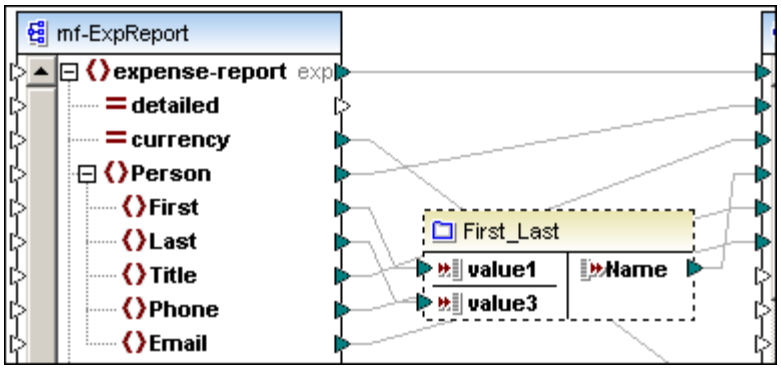
- the function "hello" in the "helloworld" library is imported from a custom *.mff library,
- the function "Join" in the "helloworld" library is a user-defined function defined in the **current** *.mfd file and
- the function "MyUDF" in the "user" library is also a user-defined function defined in the current *.mfd file



Java Selected

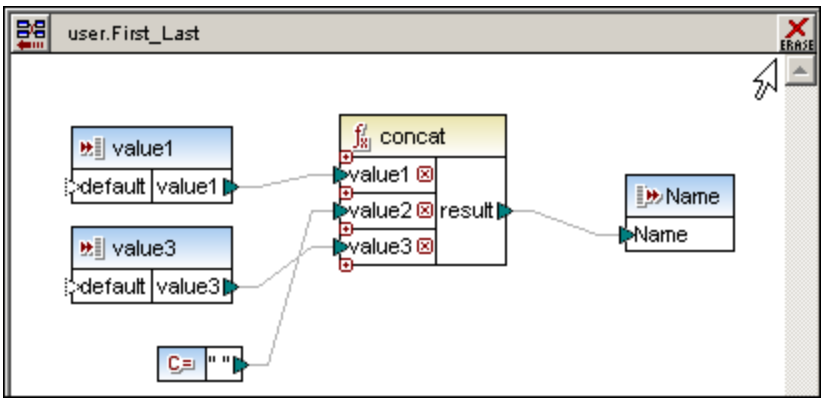
Note that possible changes in imported functions are applied to importing mappings when saving the library ***.mfd** file.

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:

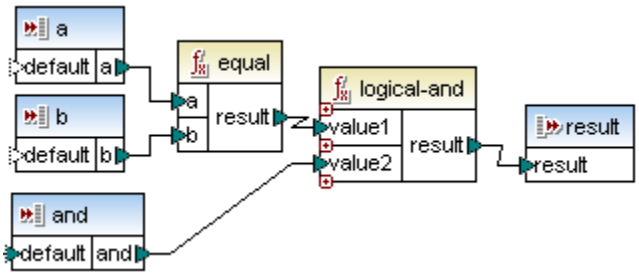
- Component positioning and resizing actions are "undoable".
- Newly added input or output components, are created below the last input or output component.
- Complex and simple parameters can be mixed. The parameter order is derived from the component positions.

18.1.1 Function parameters

Function **parameters** are represented inside a user-defined function by **input** and **output components.**

Input components/parameters: **a**, **b**, **and**

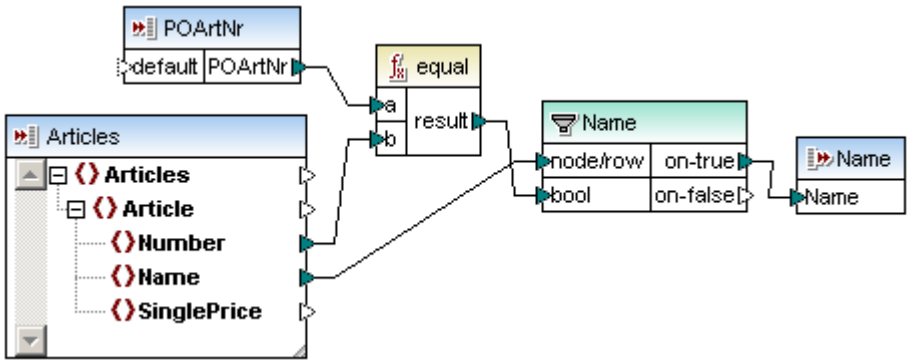
Output component/parameter: **result**



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

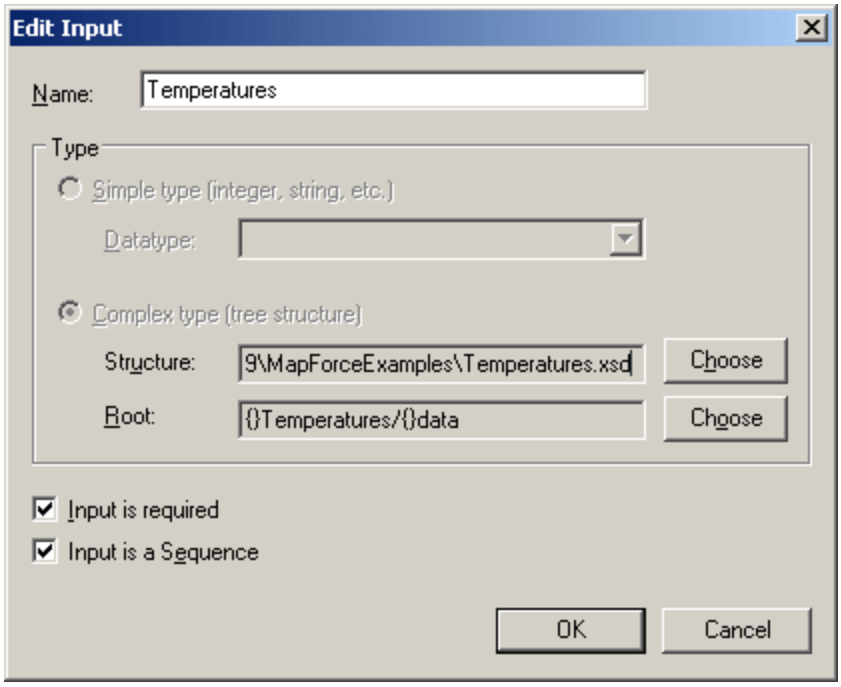
Note:
The user-defined functions shown above are all available in the **PersonListByBranchOffice.mfd** file available in the ...\MapForceExamples folder.

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

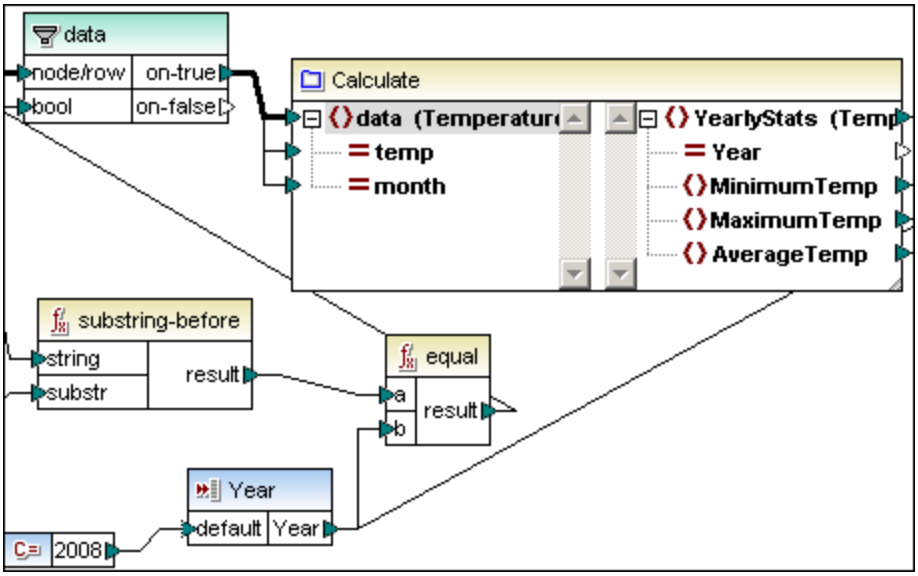
specific value from the input sequence. Please see Aggregate functions for more information.

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.

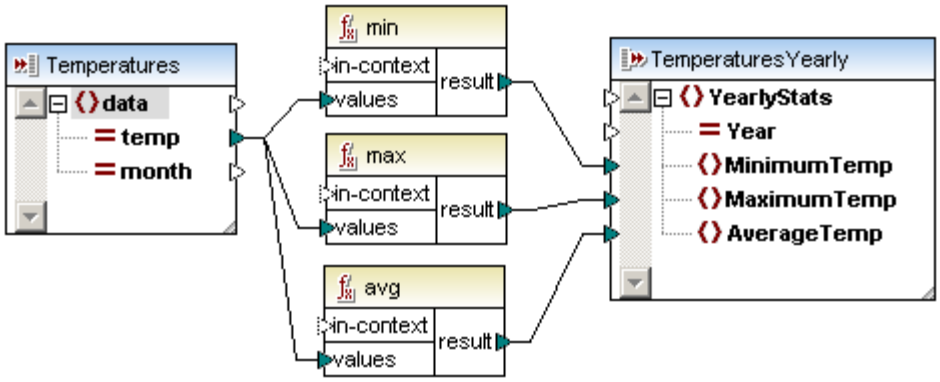


This type of input data, sequence or non-sequence, determines **how often** the function is called.

- 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 are 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.

18.1.2 Inline and regular user-defined functions

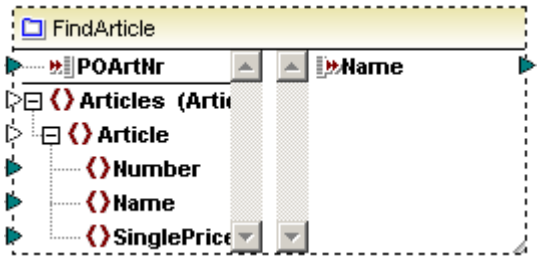
Inline functions differ fundamentally from regular functions, in the way that they are implemented when code is generated.

- 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

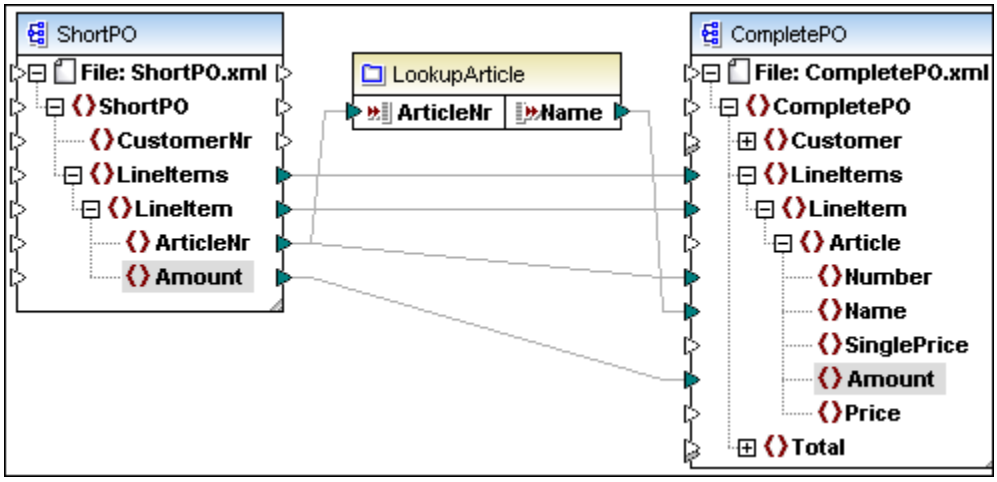
When creating Copy-all connections between a schema and a complex user-defined function parameter, the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components - defining" for an example.

18.1.3 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:

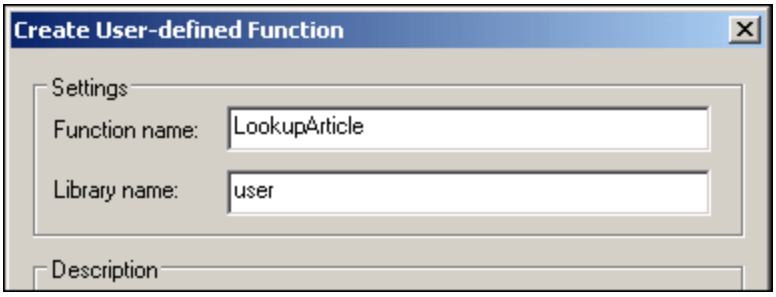
- supplies Articles/Number data from the Articles XML file, to be compared to Article numbers of a different XML file, ShortPO in this case.



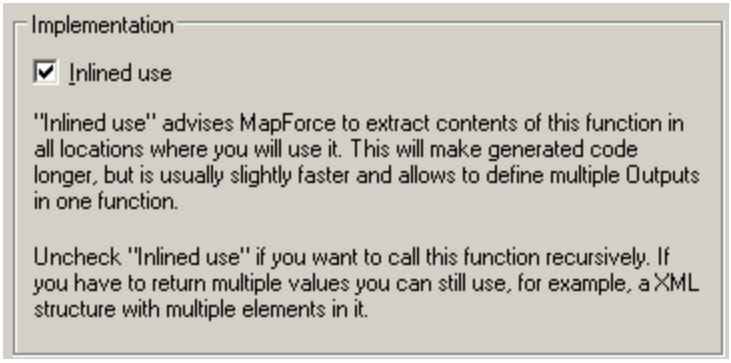
- Insert the ShortPO.xsd and assign ShortPO.xml as the source XML file.
- Insert the CompletePO.xsd schema file, and select CompletePO as the root element.
- Insert a new user-defined function using the method described below.

To create a user-defined function from scratch:

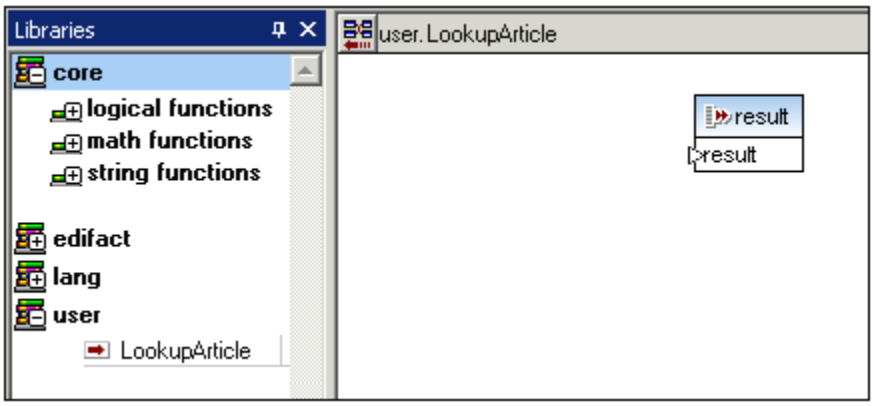
1. Select the menu option **Function | Create User-defined function**.
2. Enter the name of the function e.g. LookupArticle.



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.

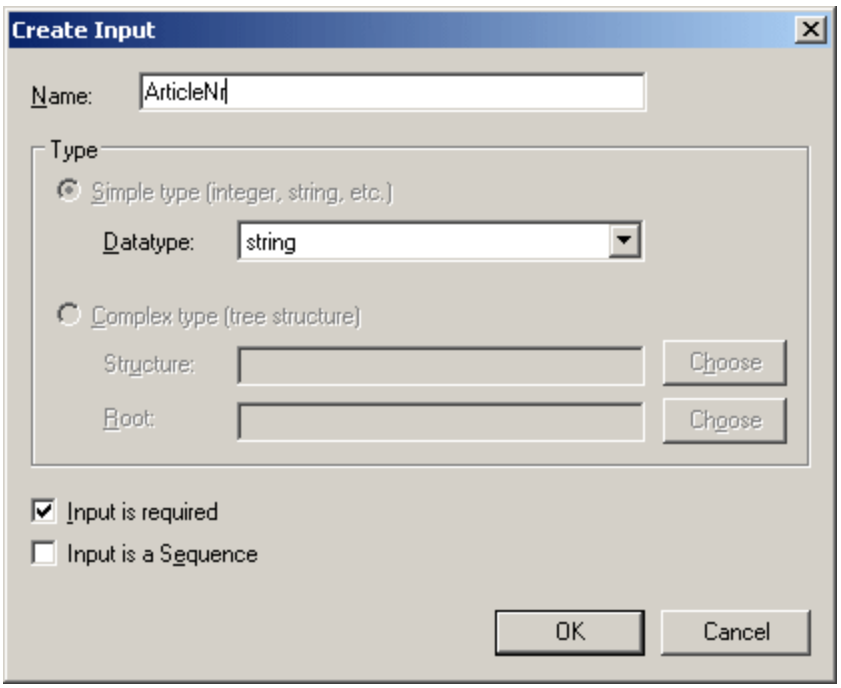


Java Selected

This is the working area used to define the user-defined function.

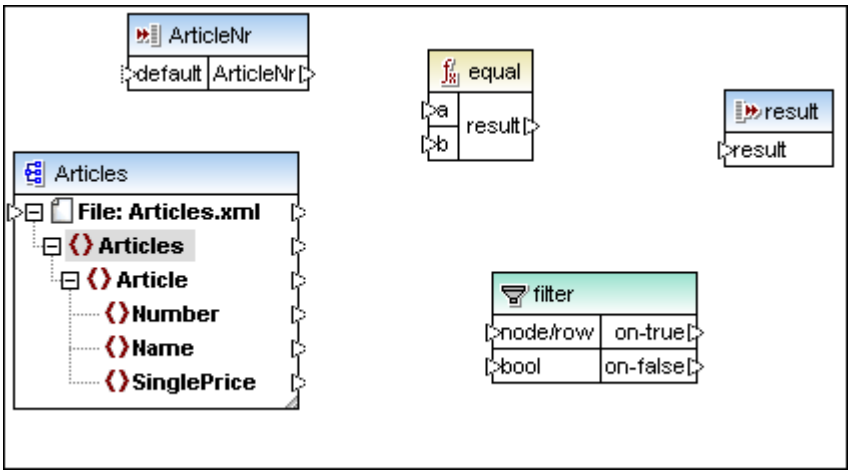
A new library has been created in the Libraries pane with the name "user" and the function name "LookupArticle".

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.



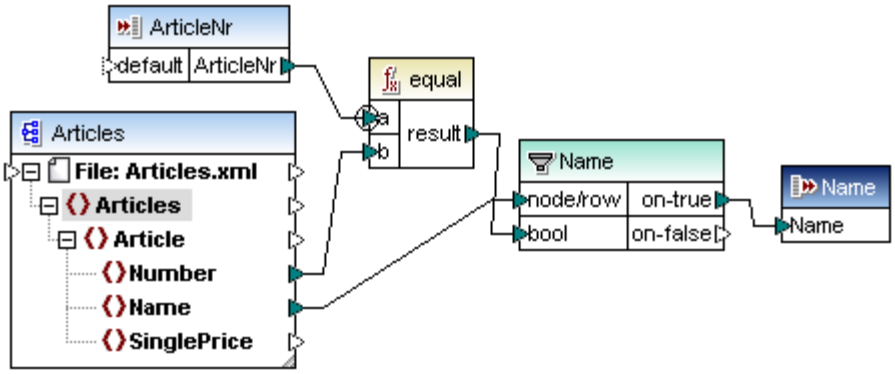
This component acts as a data input to the user-defined function and supplies the input icon of the user-defined function.

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.

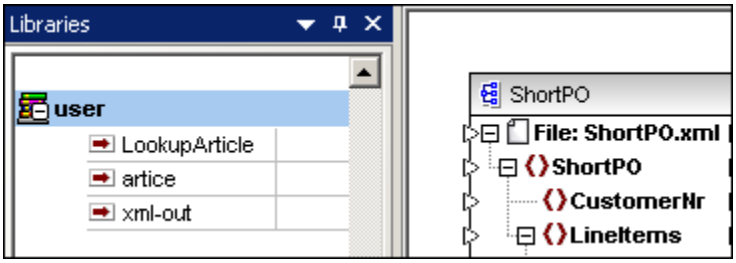
Please note:

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.

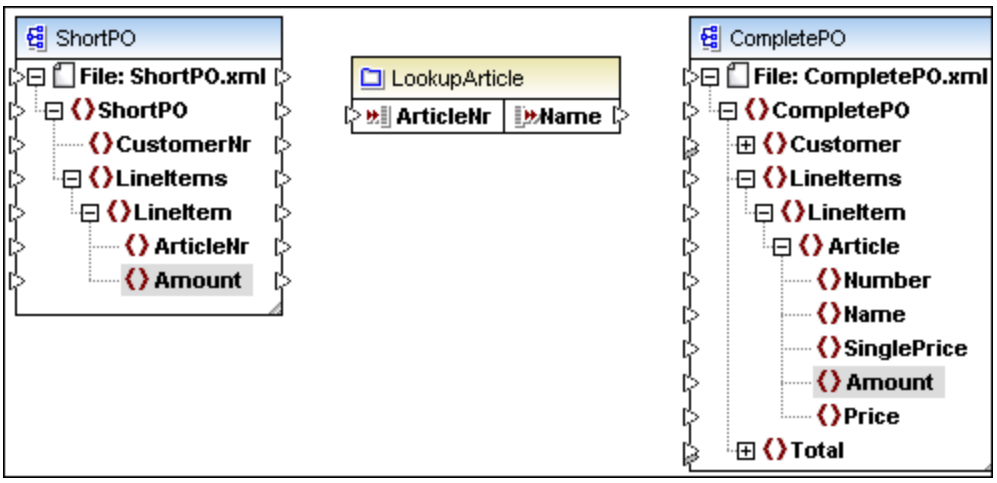


Java Selected

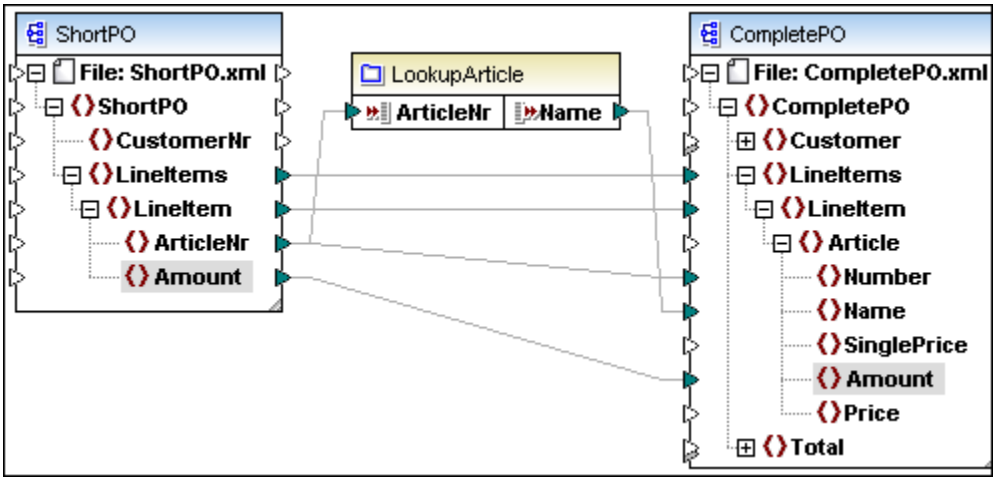
11. Drag the LookupArticle function into the Mapping window.

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.



18.1.4 Complex user-defined function - XML node as input

This example is provided as the **lookup-udf-in.mfd** file available in the ...\MapForceExamples folder. What this section will show, is how to define an inline user-defined function that contains a complex input component.

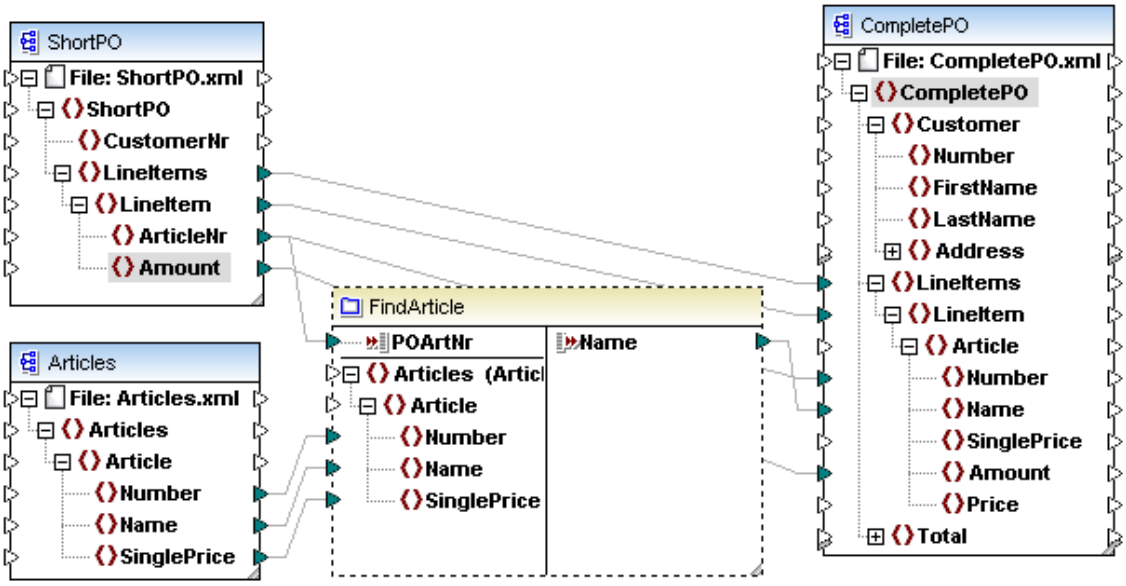
Note that the user-defined function "FindArticle" consists of two halves.

A left half which contains the input parameters:

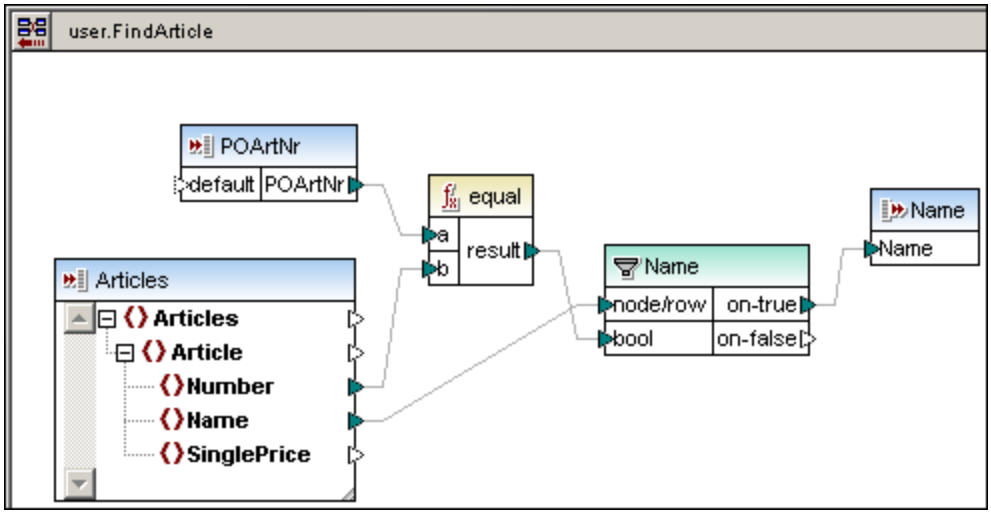
- a simple input parameter **POArtNr**
- a complex input component **Articles,** with mappings directly to its XML child nodes

A right half which contains:

- a simple output parameter called "**Name"**.



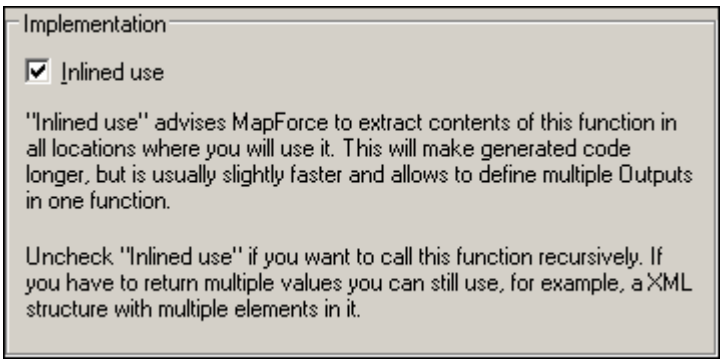
The screenshot below shows the constituent components of the user-defined function, the two input components at left and the output component at right.



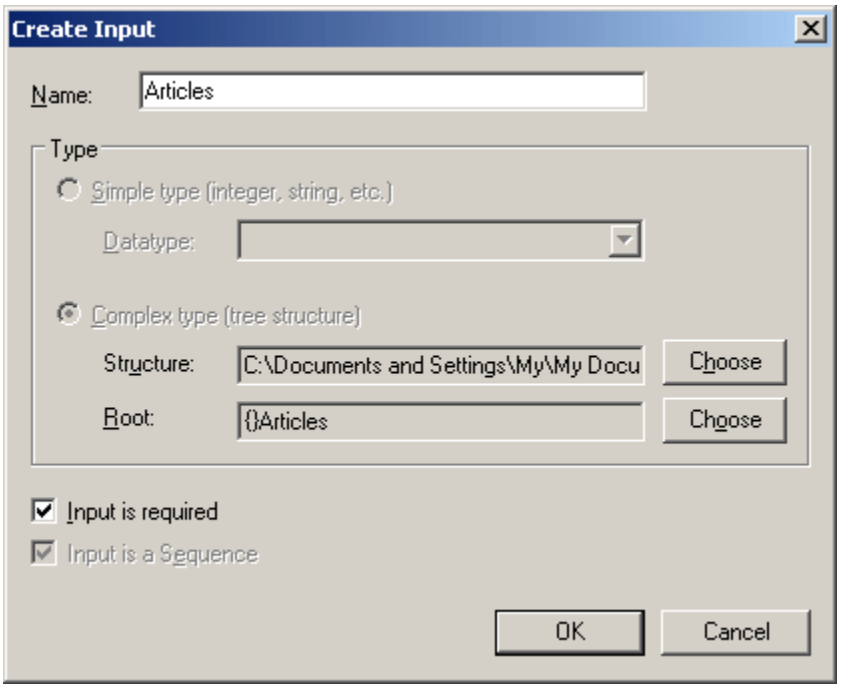
Complex input components - defining

Defining complex input components:

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.



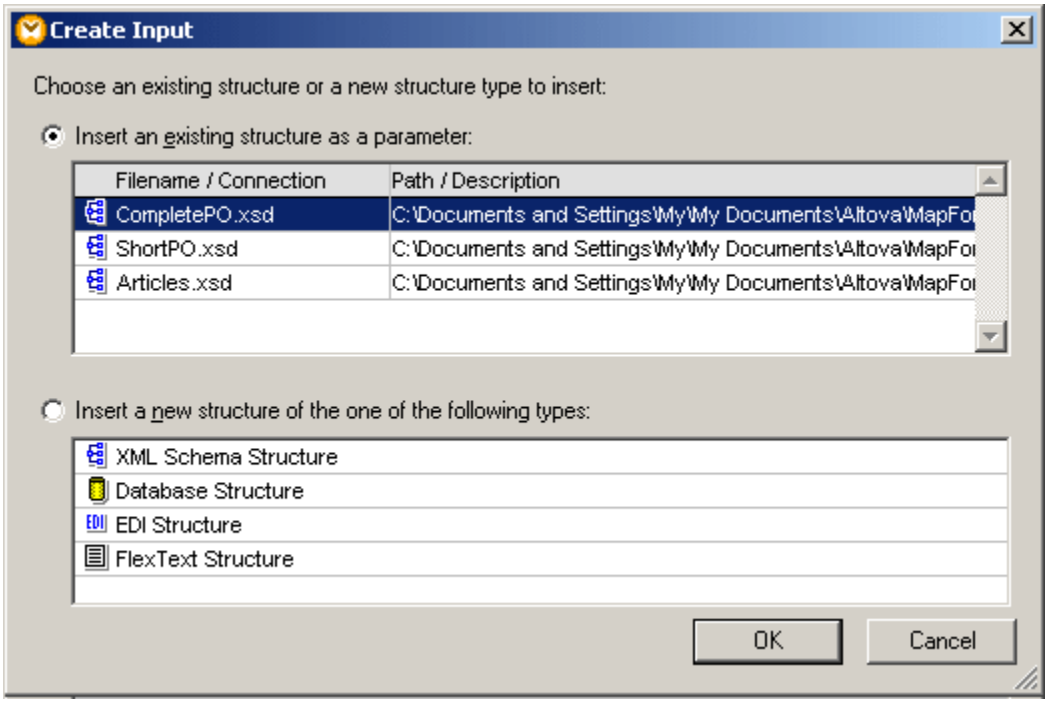
2. Click the **Insert input component** icon  in the icon bar.
3. Enter the name of the input component into the Name field.



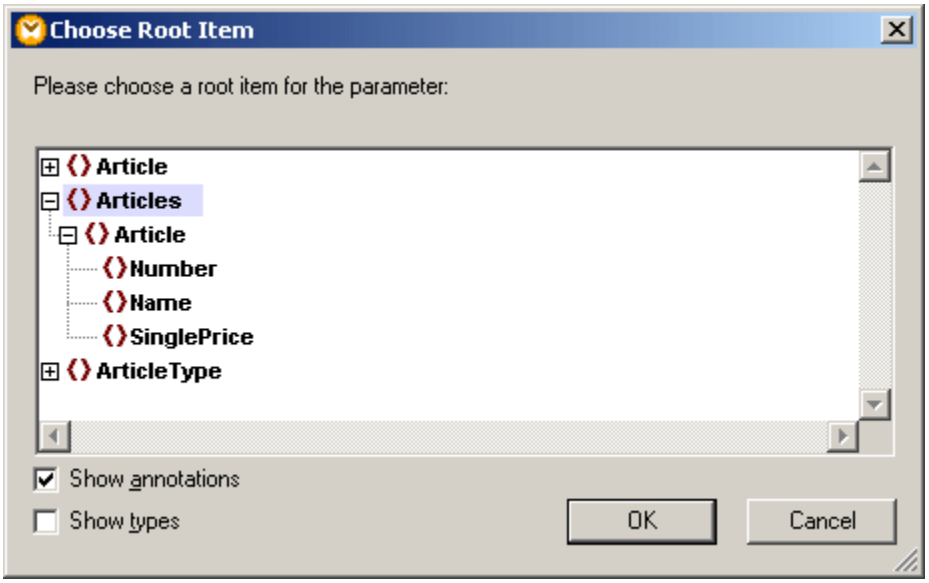
4. Click the **Complex type (tree structure)** radio button, then click the "**Choose**" button next to the Structure field.
This opens another dialog box.

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 file.

The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, file.

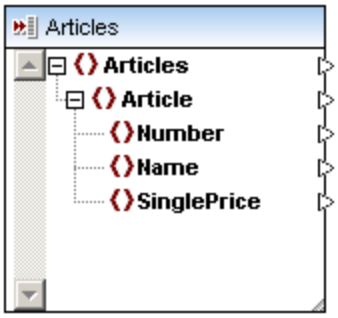


5. Click "Insert a new structure... " radio button, select the XML Schema Structure entry, and click OK to continue.
6. Select **Articles.xsd** from the "Open" dialog box.
7. Click the element that you would like to become the root element in the component, e.g. Articles, and click OK, then OK again to close both dialog boxes.

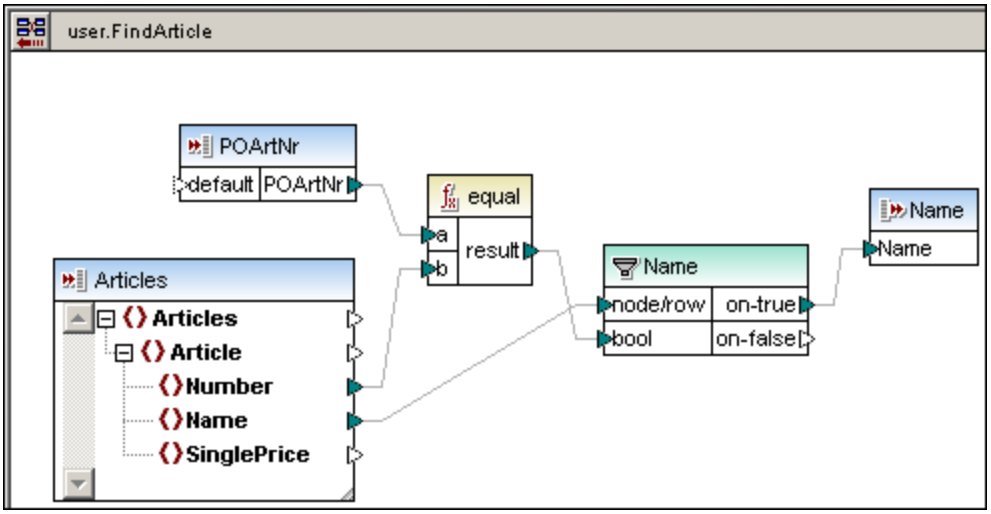


The Articles component is inserted into the user-defined function. Please note the input

icon to the left of the component name. This shows that the component is used as a complex input component.



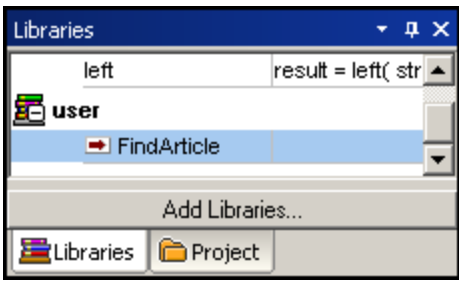
8. Insert the rest of the components as shown in the screenshot below, namely: a second "simple" input component (called POArtNr), filter, equal and output component (called Name), and connect them as shown.



Please note:

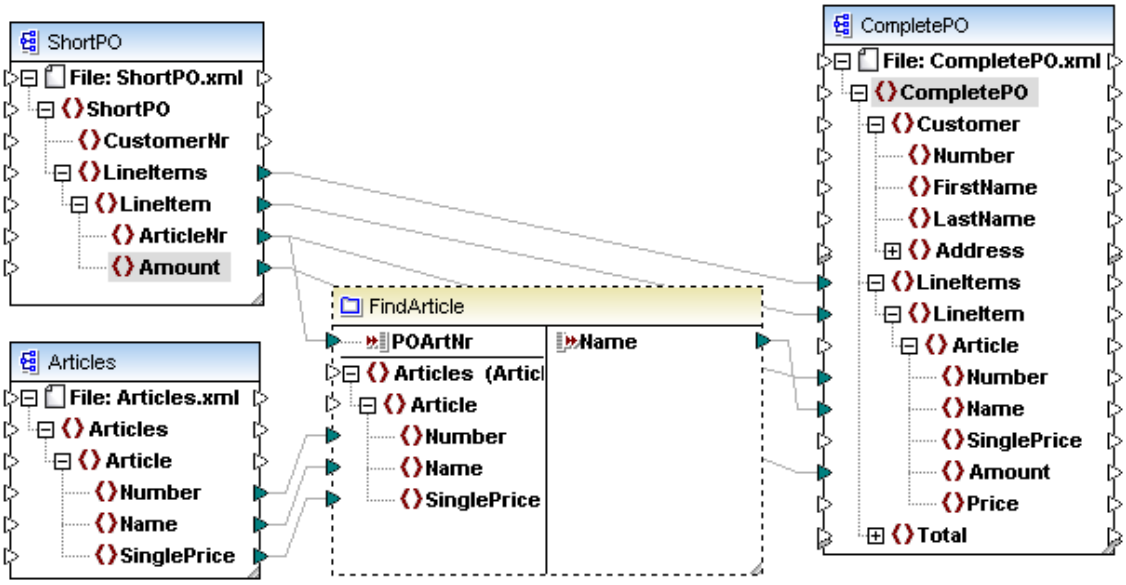
- The **Articles** input component receives its data from **outside** of the user-defined function. Input icons that allow mapping to this component, are available there.
- An XML **instance** file to provide data from within the user-defined function, cannot be assigned to a complex input component.
- The other input component POArtNr, supplies the ShortPO article number data to which the **Article | Number** is compared.
- The filter component filters out 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.



Java Selected

12. Create the connections as shown in the screenshot below.

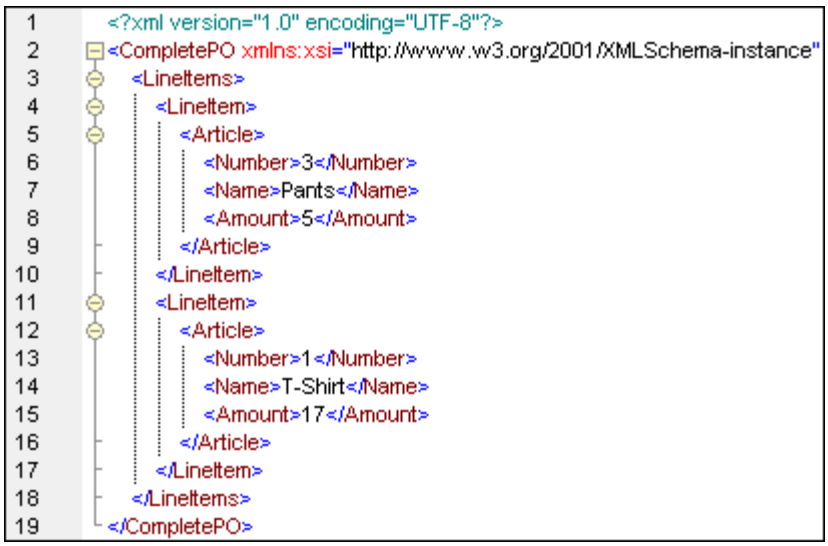


The left half contains the input parameters to which items from two schema/xml files are mapped:

- **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 on the filtered line items which have the same Article number, to the Name item of Complete PO.



```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <LineItems>
    <LineItem>
      <Article>
        <Number>3</Number>
        <Name>Pants</Name>
        <Amount>5</Amount>
      </Article>
    </LineItem>
    <LineItem>
      <Article>
        <Number>1</Number>
        <Name>T-Shirt</Name>
        <Amount>17</Amount>
      </Article>
    </LineItem>
  </LineItems>
</CompletePO>
```

Please 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.

18.1.5 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.

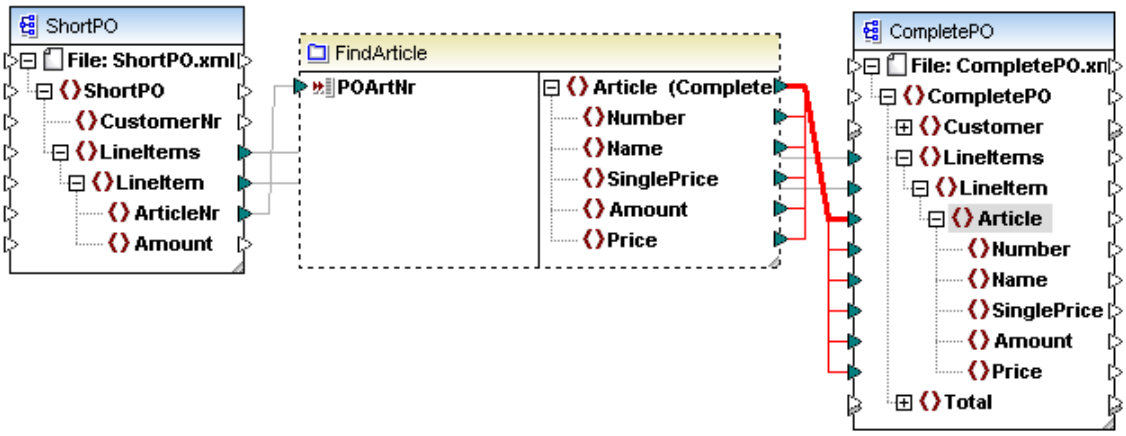
Note that the user-defined function FindArticle consists of two halves.

A left half which contains the input parameter:

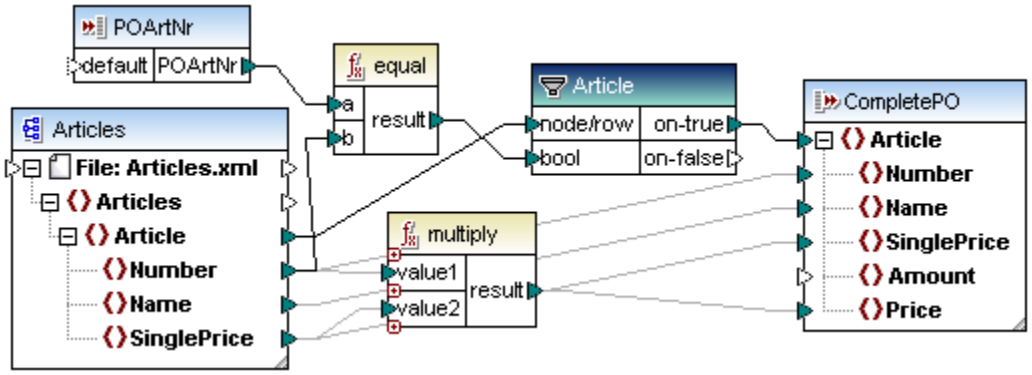
- a simple input parameter **POArtNr**

A right half which contains:

- a complex output component **Article (CompletePO)** with its XML child nodes mapped to CompletePO.



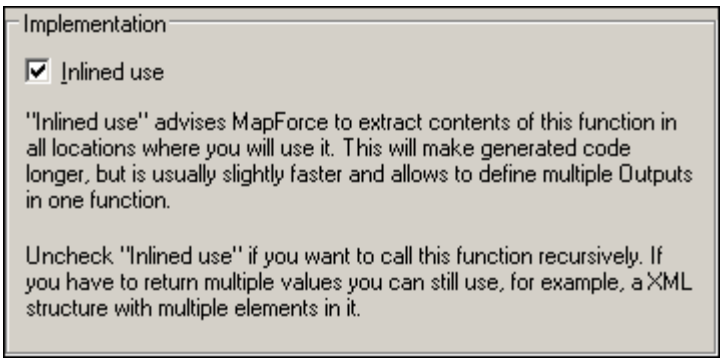
The screenshot below shows the constituent components of the user-defined function, the input components at left and the complex output component at right.



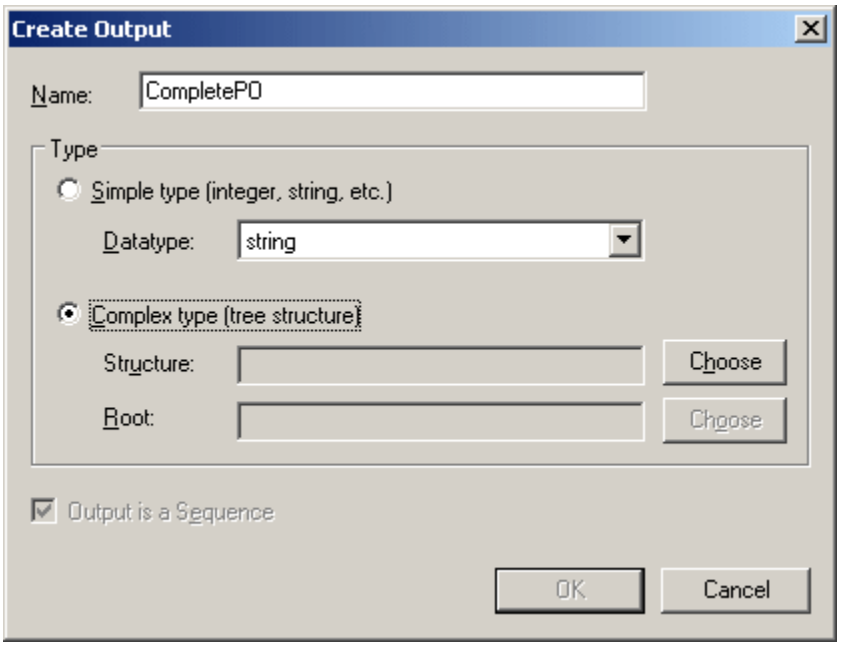
Complex output components - defining

Defining complex output components:

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.



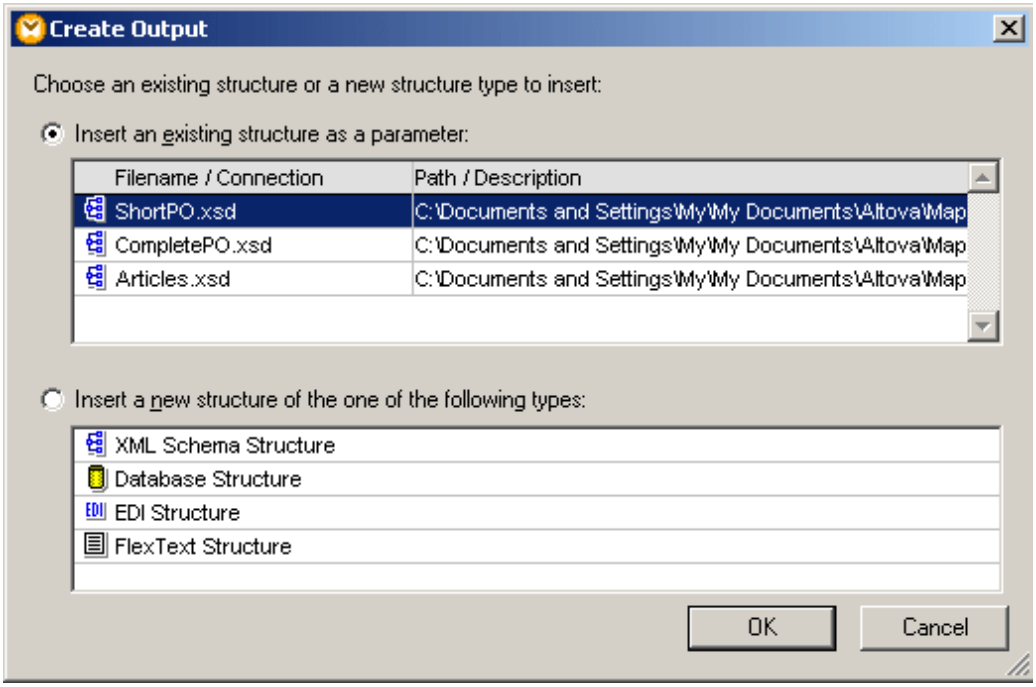
2. Click the Insert **Output** icon  in the icon bar, and enter a name e.g. CompletePO.



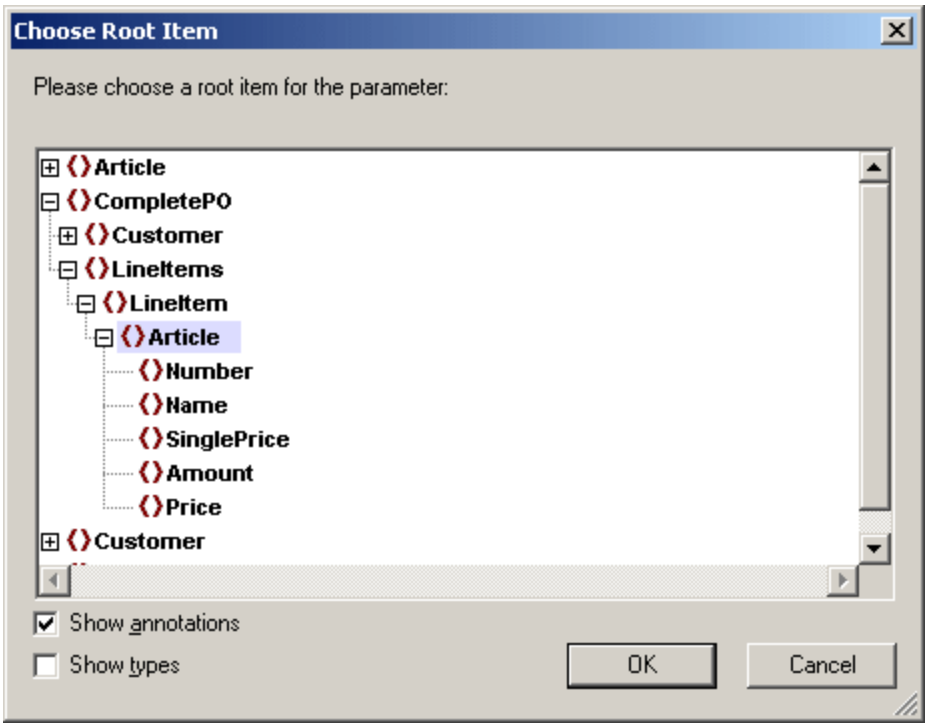
3. Click the **Complex type...** radio button, then click the "**Choose**" button.
This opens another dialog box.

   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 file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, file.

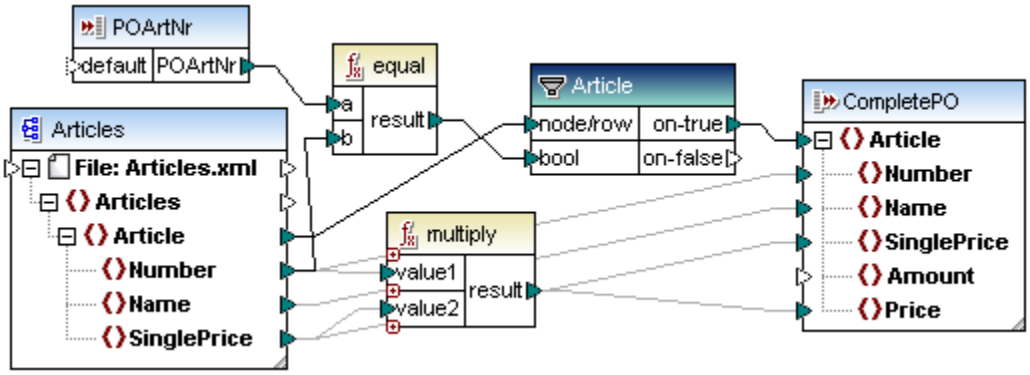


4. Click "**Insert new structure...** " radio button, select the XML Schema Structure entry, and click OK to continue.
5. Select the **CompletePO.xsd** from the "Open" dialog box.
6. Click the element that you would like to become the root element in the component, e.g. **Article**, and click OK, then 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.

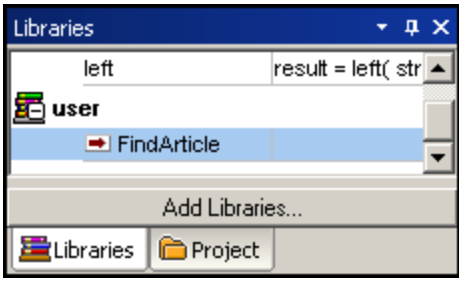
7. Insert the Articles schema/XML file into the user-defined function and assign the **Articles.xml** as the XML instance.
8. Insert the rest of the components as shown in the screenshot below, namely: the "simple" input components (POArtNr), filter, equal and multiply components, and connect them as shown.



Please note:

- 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 out the records where both numbers are identical, and passes them on to the CompletePO output component.

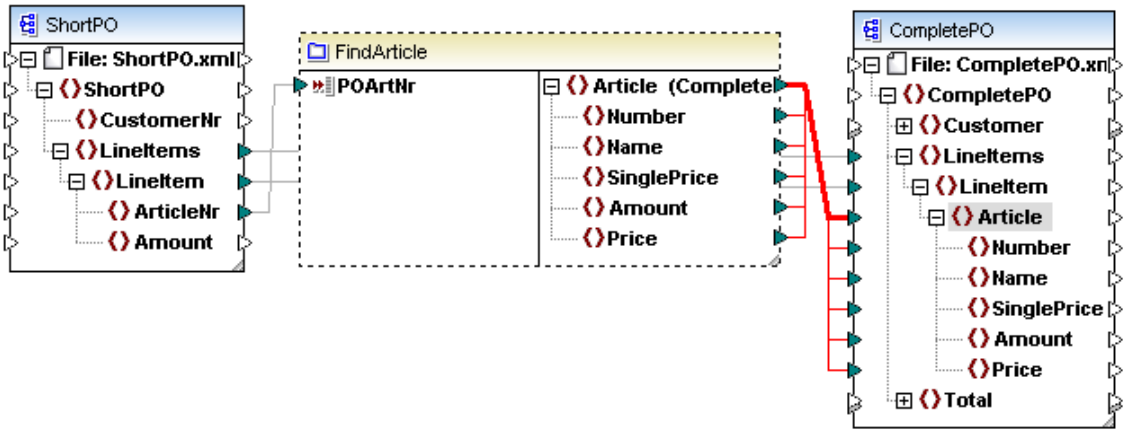
9. Click the Home icon  to return to the mapping.
10. Drag the newly created user-defined component from the Libraries pane into the mapping.



Java Selected

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:

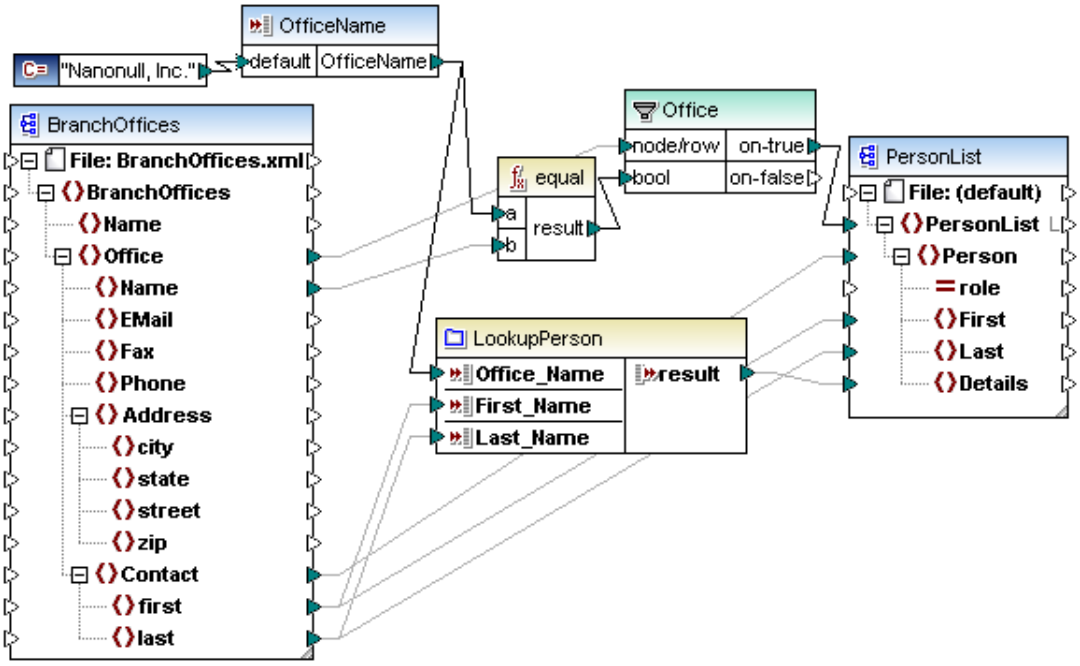
- a complex output component called "**Article (CompletePO)**" with its XML child nodes, which maps the filtered items, of the same Article number, to CompletePO.

```
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>
```

18.1.6 User-defined function - example

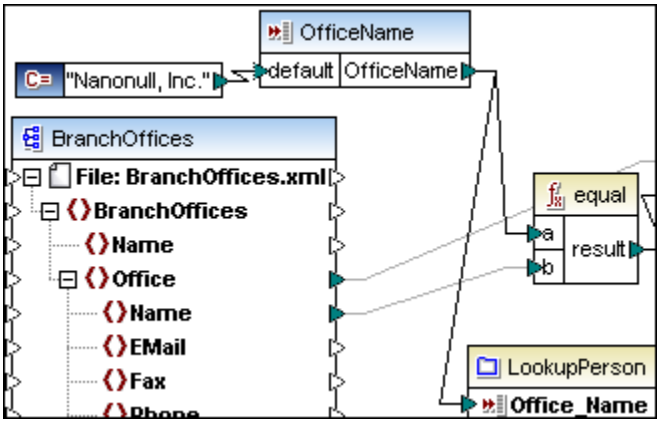
The **PersonListByBranchOffice.mfd** file available in the **...\MapForceExamples** folder, describes the following features in greater detail:

- 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.

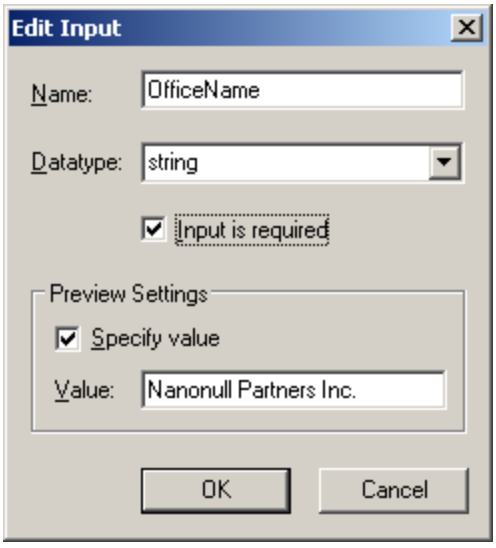


To define the Input value:

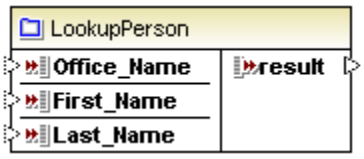
1. Double click the input component and enter a different value in the "Value" text box of the Preview Mode group e.g. "Nanonull Partners, Inc.", and click OK to confirm.
2. Click the Output tab to see the effect.
   A different set of persons are now displayed.

   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 values, overrides and command line parameters for more information.



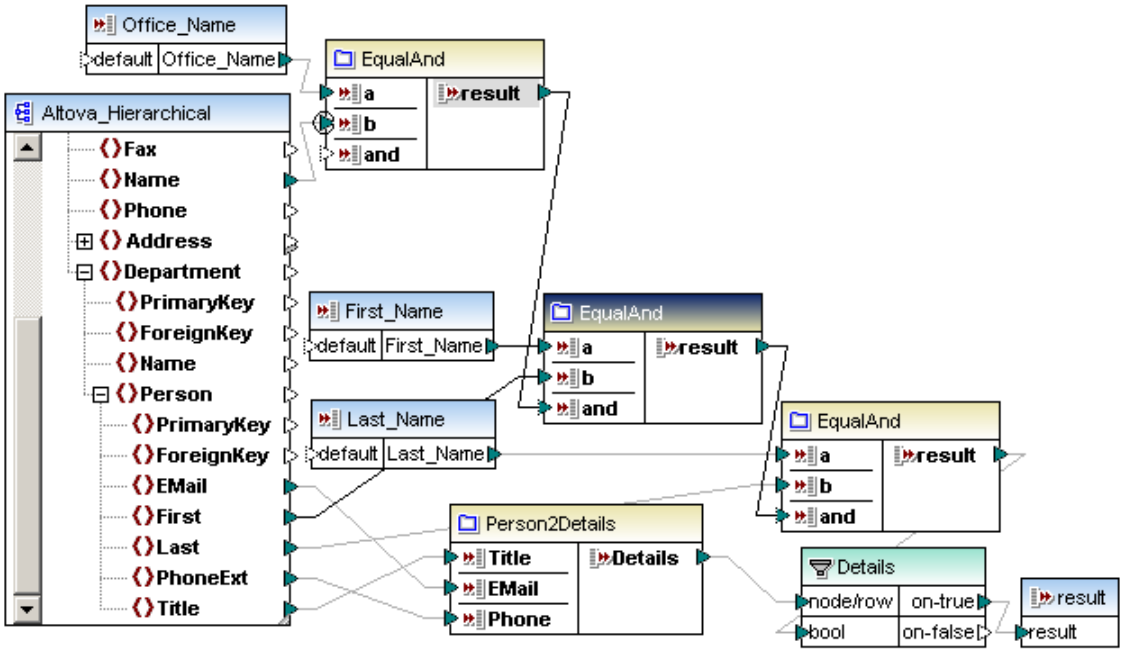
LookupPerson component



Double clicking this user-defined component displays its constituent components shown below. What this component does is:

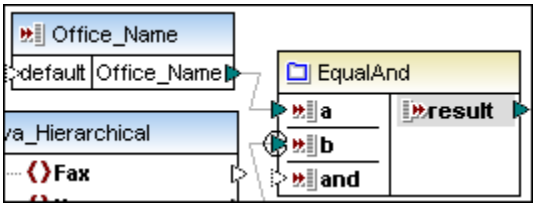
- **Compares** the Office, First, and Last names of BranchOffices.xml, with the same fields of the Altova_Hierarchical.xml file, using the **input** components and the **EqualAnd** user-defined components.
- **Combines** the Email, PhoneExt and Title items using the **Person2Details** user-defined function
- **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.



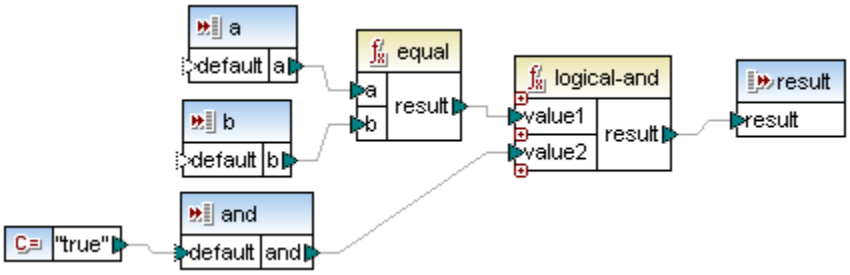
- The three **input** components, Office_Name, First_Name, Last_Name, receive their data from the BranchOffices.xml file.
- The **EqualAnd** component compares two values and provides an **optional** comparison value, as well as a default value.
- Person2Details combines three person fields and passes on the result to the filter component.

EqualAnd component



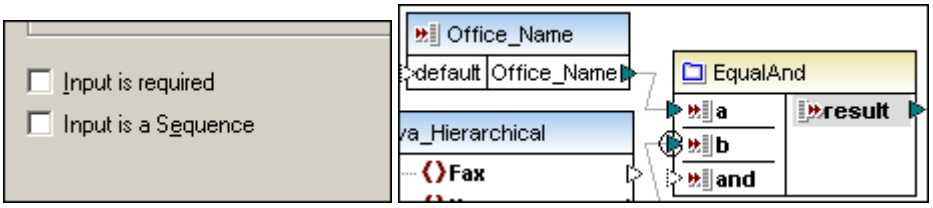
Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- 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.



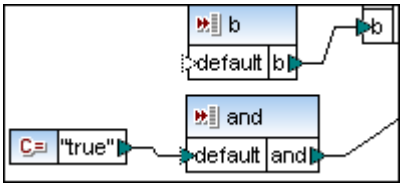
Optional parameters

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.



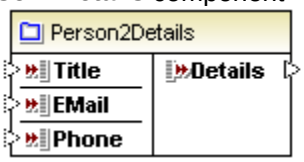
If "Input is required" is **unchecked**, then:

- 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".



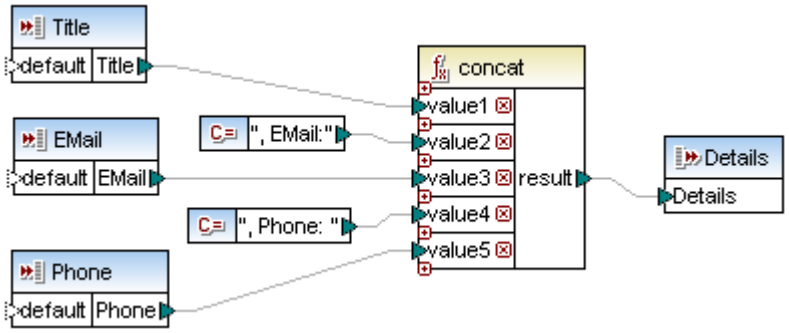
- A mapping from another item, mapped to the optional Input, takes precedence over the default value. E.g. the "and" parameter of second EqualAnd function, receives input data from the "result" parameter of the first EqualAnd user-defined function.

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).



18.2 Adding custom XSLT and XQuery 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 the output, by clicking the XSLT icon, or selecting **Output | XSLT 1.0**.

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 1.0 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

Please note:

When writing named templates please make sure that the XPath statements used in the template are bound to the correct namespace(s). The namespace bindings of the mapping can be viewed by clicking the XSLT tab. Please see: the XSLT 1.0 implementation specific document for more information.

18.2.1 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

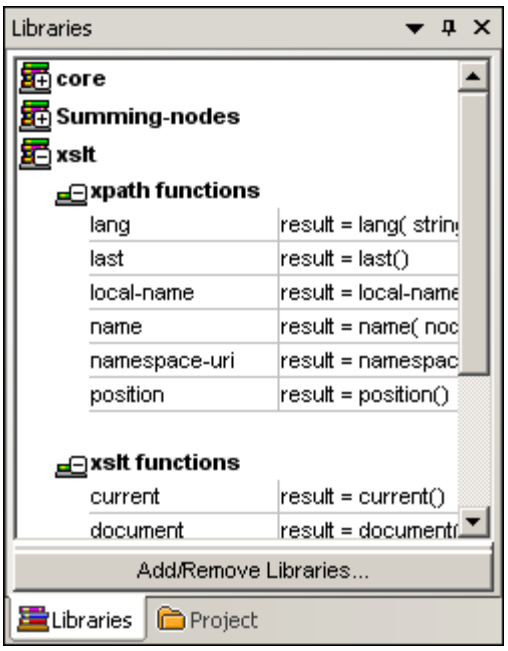
Please see: Aggregate functions for an additional example of using named templates to sum nodes.

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.

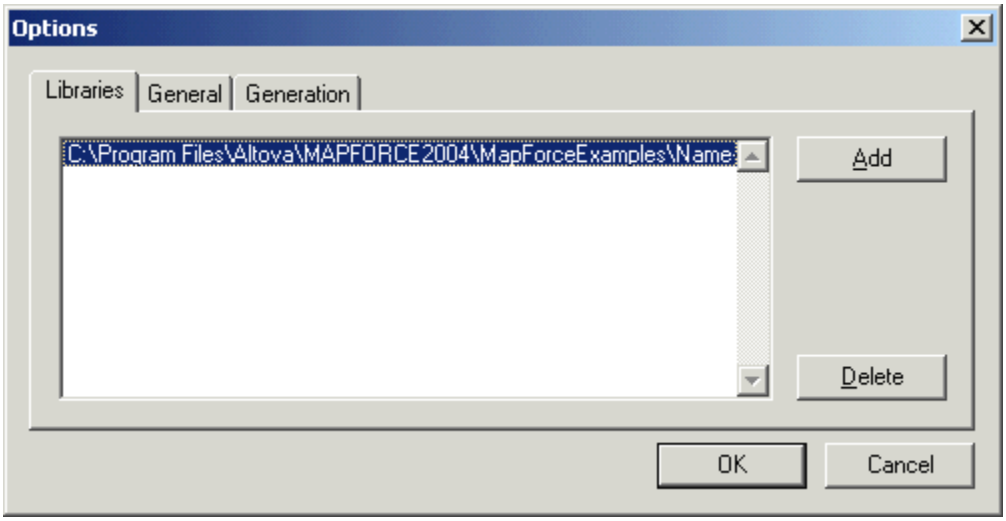
```
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
4
5    <xsl:template match="*">
6      <xsl:for-each select=".">
7      <xsl:call-template name="tokenize">
8      <xsl:with-param name="string" 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.

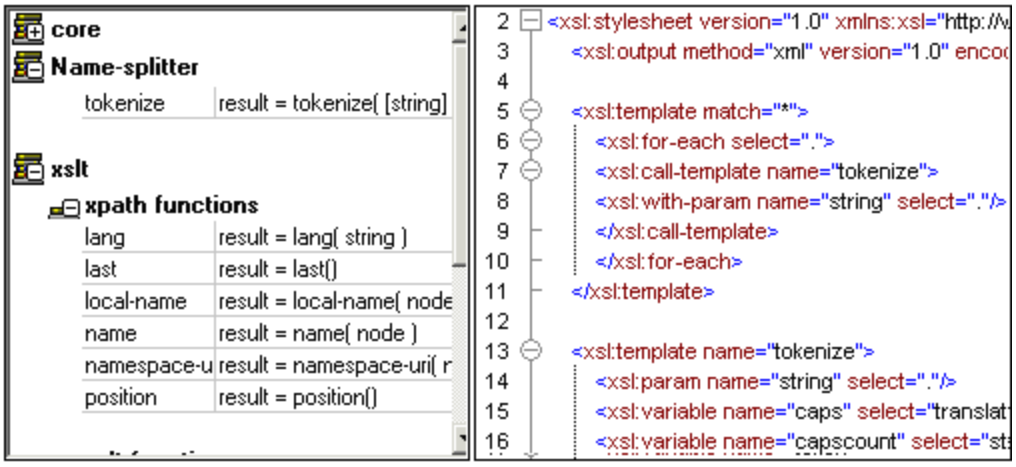


XSLT Selected

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.



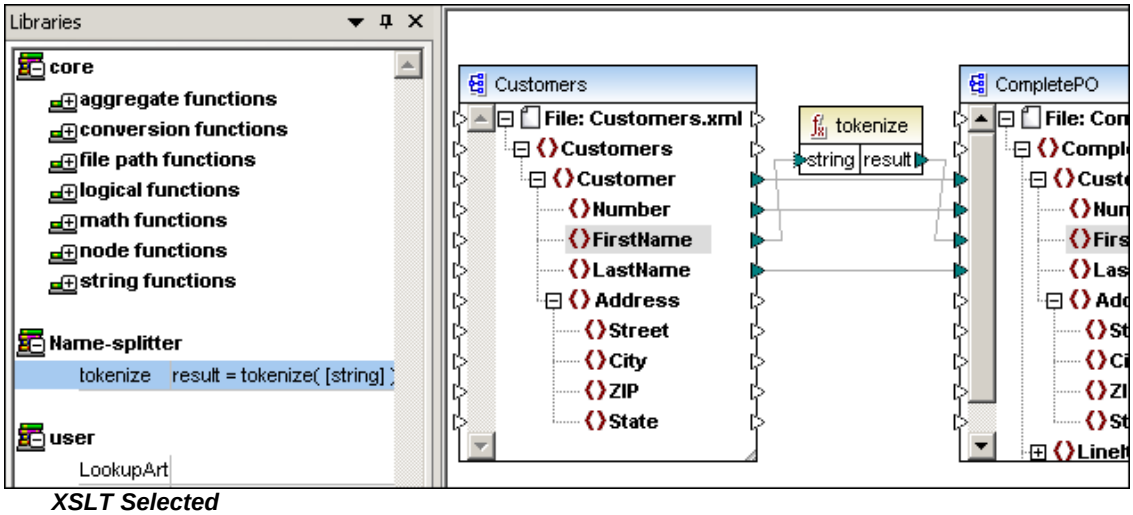
4. Click **OK** to insert the new function.



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.



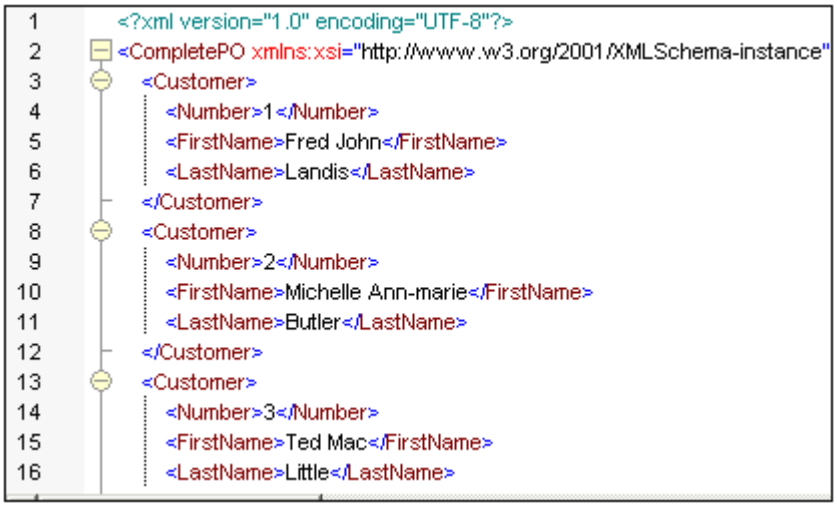
XSLT Selected

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.

18.2.2 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">

Please see: the XSLT 2.0 implementation specific document for more information, as well as Aggregate functions for an additional example of using named templates to sum nodes.

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.

18.2.3 Adding custom XQuery functions

MapForce allows you to import XQuery library modules.

Please see: the XQuery implementation specific document for more information.

18.2.4 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:

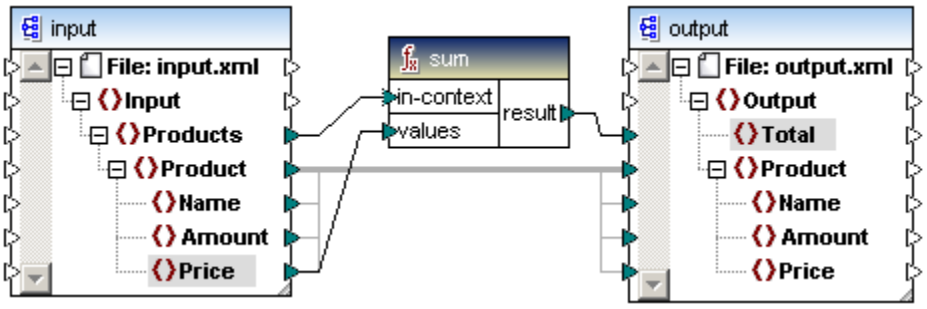
Summing-nodes.mfd	mapping file
input.xml	input XML file
input.xsd and output.xsd	source and target schema files
Summing-nodes.xslt	xslt file containing a named template to sum the individual nodes

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.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Input xmlns:xsi="http://www.w3.org/2001/XMLSchema
3      <Products>
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     </Products>
15 </Input>
```

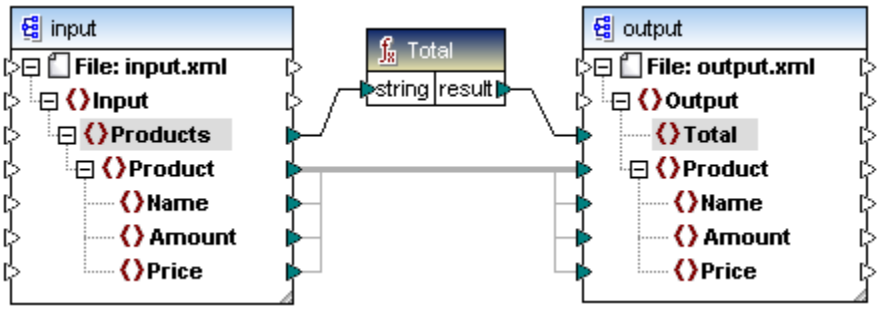
The screenshot below shows the XSLT stylesheet which uses the named template "**Total**" and a single parameter "string". What the template does, is work through the XML input file and sum all the values obtained by the XPath expression **/Product/Price**, in the document.

```
<?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 ...\MapForceExamples\Tutorial\ folder.
3. Drag in the Total function from the newly created Summing-nodes library and create the mappings as shown below.



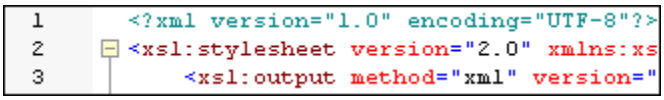
4. Click the Output tab to preview the mapping result.

```
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".



18.3 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 of course 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.

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:

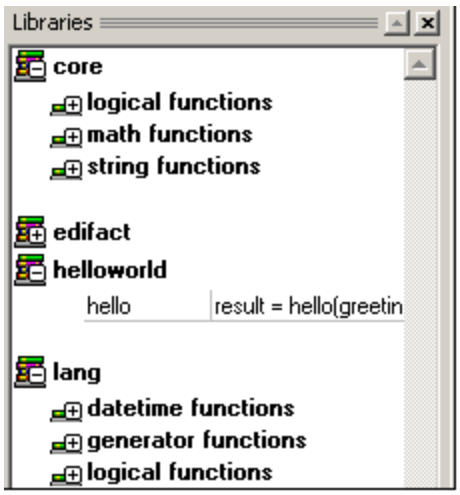
```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:noNamespaceSchemaLocation="mff.xsd" version="8" library="helloworld">
   <implementations>
      <implementation language="cs">
         <setting name="namespace" 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.

18.3.1 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>
...
```


18.3.2 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:

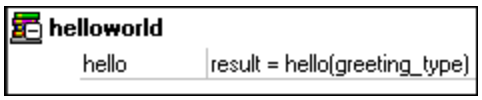
- one of the XML Schema types (eg. xs:string, xs:integer, etc.)
-

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.

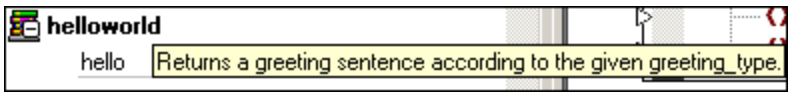
Altova has provided support for Schema simpleTypes (date, time, duration, dateTime) as classes, for each of the supported programming languages. The integration of these Schema simpleTypes in your library will be explained later in this document.

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:



18.3.3 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.

18.3.4 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

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[]
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

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
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
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

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
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

18.4 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.

Supported:

- 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:

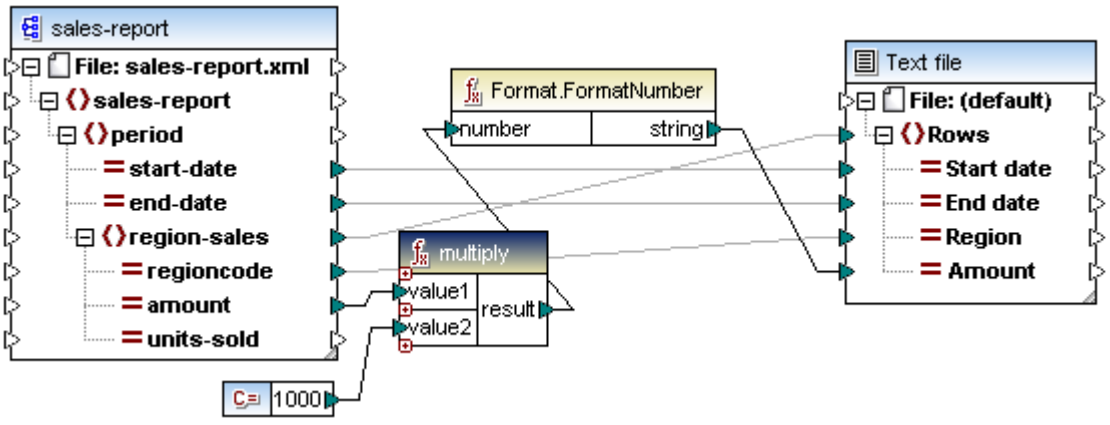
- Native C++ DLLs (or other DLLs that are not a .NET assembly)
- Using .class or DLL files/functions, while having the Output language set to C++
- Using Java or .NET functions directly in XSLT (a custom XSLT function that acts as an adapter, would have to be written)

  C++ functions cannot be previewed in the Output window, but are available in the generated code.

To add custom Java or .NET functions, you need:

- The compiled Java classes (.class) or
- The .NET assembly files (.dll).

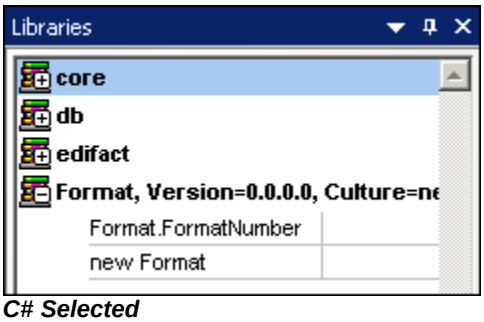
MapForce example files that show how these functions are used, are available in the ...\MapForceExamples**Java** and ...\MapForceExamples**C#** folders. Both example files are called **FormatNumber.mfd**.



To add the .NET assembly file:

1. 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.



C# Selected

2. Open the **FormatNumber.mfd** file available in the ...\MapForceExamples**C#** folder (screenshot shown above).
3. 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.

18.5 Java and .NET functions - specifics

Implementation details:

Adding Libraries:

- Java: `.class` files can be added to MapForce (.jar files are not supported)
- .NET: `.dll` assembly files can be added to MapForce

Warning: All functions called from a MapForce mapping should be "**idempotent**" (this means that they it 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 functions – setup:
If imported java class files depend on other class files, be sure to adjust the CLASSPATH environment variable first before starting MapForce. The parent directories of all dependent packages should be added to the CLASSPATH variable.

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.

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>, ...)`

Types
Supported connections between XML Schema and Java types:

Schema type	Java type
xs:string	String
xs:byte	byte
xs:short	short
xs:int	int
xs:long	long
xs:boolean	boolean
xs:float	float
xs:double	double
xs:decimal	java.math.BigDecimal

xs:integer	java.math.BigInteger

Connections in both directions are possible. Other Java types 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>, ...)`

Types:
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 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 .NET methods. Manipulating the object using MapForce means is not possible.

18.6 Functions Reference

This reference section describes all the functions that are available in the Libraries pane for each of the supported languages: XSLT1, XSLT2, XQuery, Java, C#, and C++, as well as BUILTIN (the Built-in execution engine).

The following libraries are currently available:

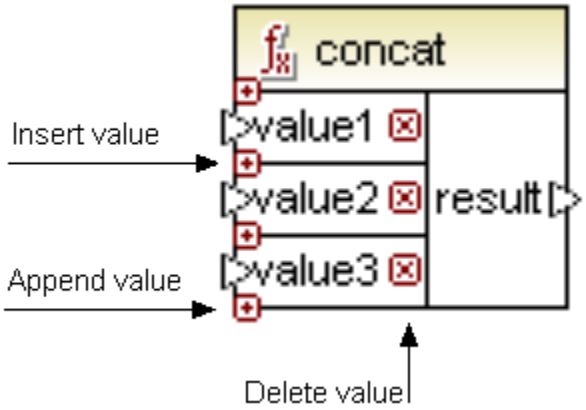
- **core**
- **db**
- **lang**
- **xpath2**
- **xslt**

Extendable functions

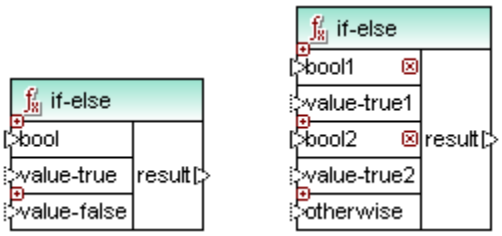
Several functions available in the function libraries are extendable: for e.g. the concat, "logical-and", "logical-or", and IF-ELSE functions. The parameters of these types of function can be inserted/appended and deleted at will.

Clicking the "plus" icon inserts or appends the same type of parameter, while clicking the check mark deletes the parameter.

Please note: "dropping" a connector on the "plus" symbol, automatically inserts/appends the parameter and connects it.



The IF test parameters, of the IF-Else function can be extended in the same way.



Placing the mouse cursor over the function title bar, pops up a tooltip describing the function.

Placing it over a parameter (any input or result parameter) displays the datatype of the argument in a tooltip.

18.6.1 core

The core library supplies the most useful functions for all languages. The sequence functions are not available if XSLT (XSLT 1.0) has been selected.

Core library

- aggregates
- conversion functions
- logical functions
- math functions
- node functions
- sequence functions
- string functions

aggregates

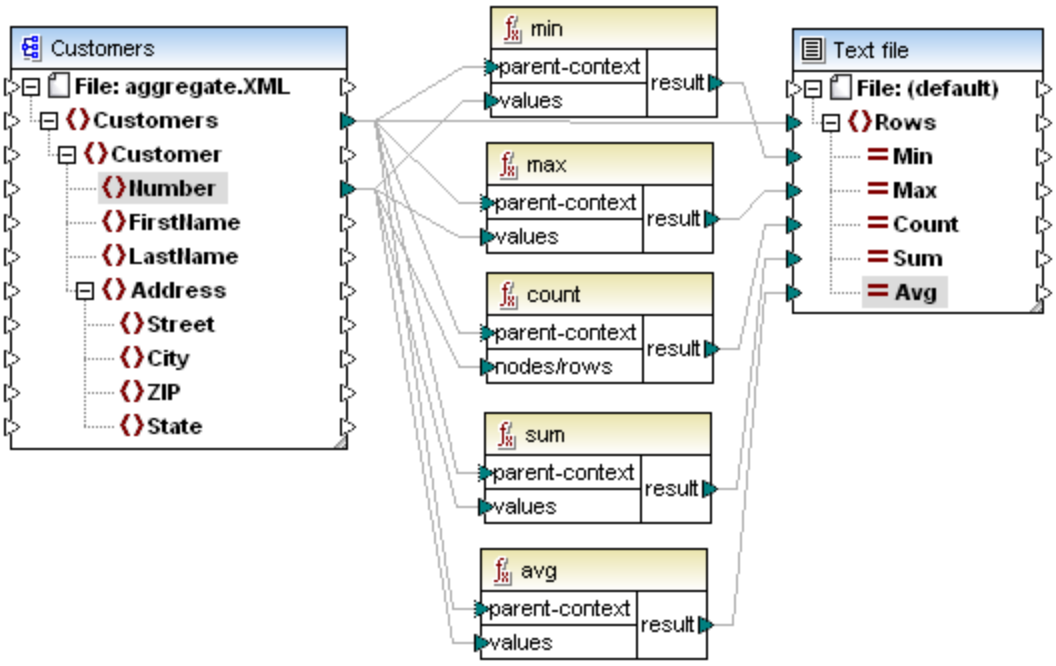
Aggregate functions perform operations on a set, or sequence, of input values. The input data for min, max, sum and avg is converted to the **decimal** datatype for processing.

- 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. This also means that the parent-context parameter is optional!
- The **result** of the function is connected to the specific target item.

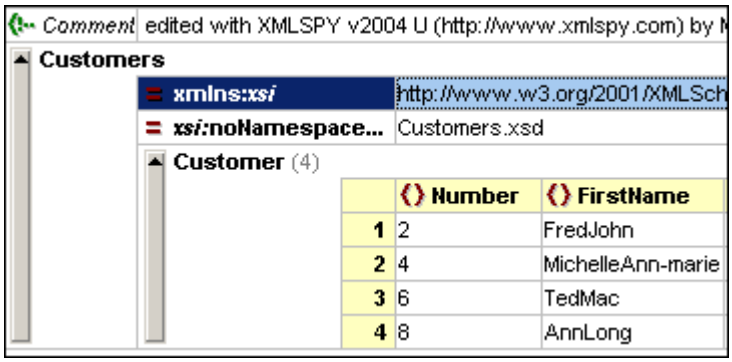
The mapping shown below is available as **Aggregates.mfd** in the ...\Tutorial folder and shows how these functions are used.

Aggregate functions have two input items.

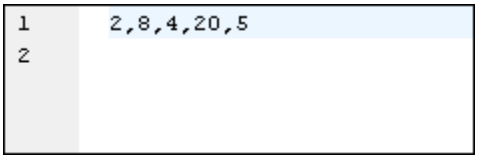
- **values** is connected to the source item that provides the data, in this case Number.
- **parent-context** is connected to the item you want to iterate over, i.e. the context, in this case over all Customers. The parameter is, howerver, optional.



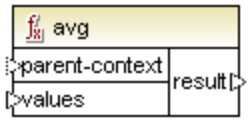
The input instance in this case is an XML file containing the following data:



- 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:

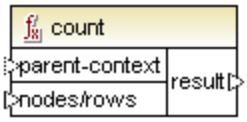


min=2, max=8, count=4, sum=20 and avg=5.



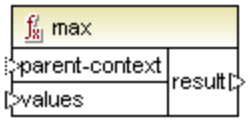
avg

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.



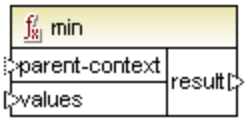
count

Returns the number of individual items making up the input sequence. The count of an empty set is zero. Limited functionality in XSLT1.



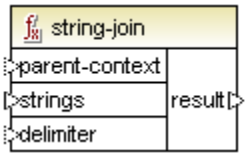
max

Returns the maximum value of all values in the input sequence. The maximum of an empty set is an empty set. Not available in XSLT1.



min

Returns the minumum value of all values in the input sequence. The minimum of an empty set is an empty set. Not available in XSLT1.

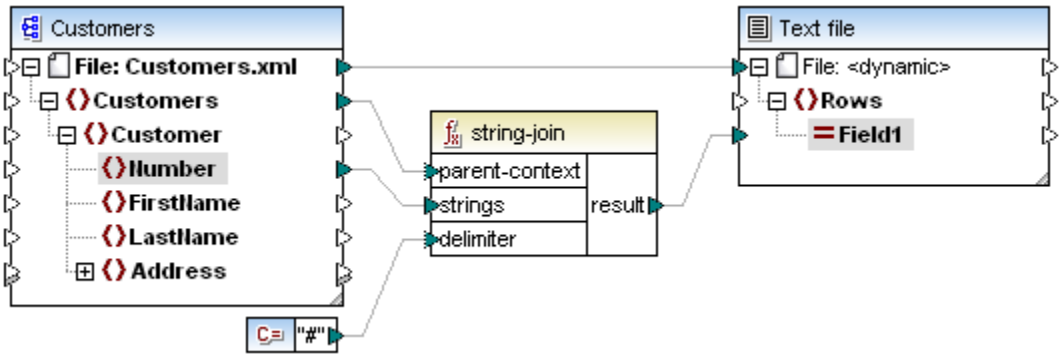


string-join

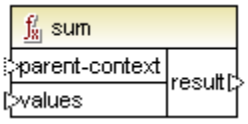
Concatenates all the values of the input sequence into one string delimited by whatever string you choose to use as the delimiter. The string-join of an empty set is the empty string. Not available in XSLT1.

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.



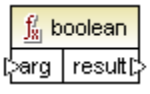
sum

Returns the arithmetic sum of all values in the input sequence. The sum of an empty set is zero. Not available in XSLT1.

conversion functions

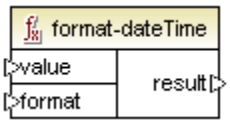
To support explicit data type conversion, the type conversion functions "boolean", "number" and "string" are available in the conversion function library. Note that in most cases, MapForce creates necessary conversions automatically and these functions need to be used only in special cases.

input parameters = **arg**
output parameter = **result**



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.



format-dateTime

Converts a dateTime into a string.

Argument	Description
value	The dateTime to be formatted
format	A format string identifying the way in which the dateTime is to be formatted

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 String

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 or PST with alphabetic modifier
z	timezone as a time offset using GMT	GMT+n

The presentation modifier:

Specifier	Description	Example
o	Indicates ordinal numbering	[1o] gives 1st, 2nd, 3rd, 4th,...

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
i	roman numerals	i, ii, iii, iv
I	roman numerals, upper case	I, II, III, IV
N	name of component, upper case	MONDAY, TUESDAY
n	name of component, lower case	monday, tuesday
Nn	name of component, title case	Monday, Tuesday
W	number expressed, upper case (not for parsing)	ONE, TWO, THREE
w	number expressed, lower case (not for parsing)	one, two, three
Ww	number expressed, title case (not for parsing)	One, Two, Three

1) N, n, and Nn modifiers only support the following components: M, d, D.

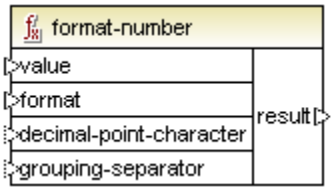
The width modifier, if present, is introduced by a comma. It takes the form:

```
, 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



format-number

Available for XSLT 1.0, XSLT 2.0, Java, C#, C++ and Built-in execution engine.

Argument	Description
value	The number to be formatted
format	A format string that identifies the way in which the number is to be formatted
decimal-point-format	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)

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
A format string identifies the way in which the number is to be formatted.

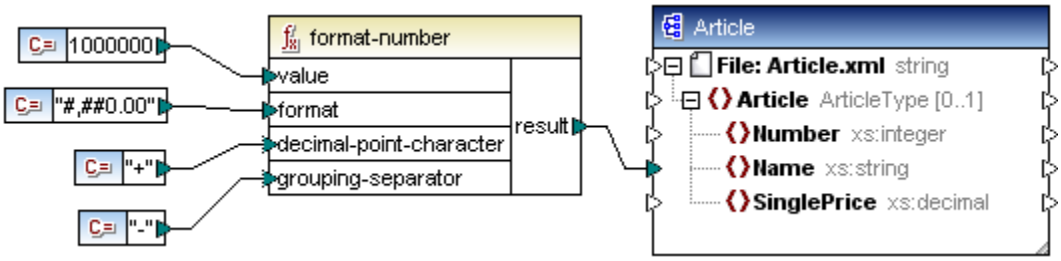
Format:


```
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.

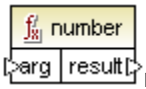
- The decimal-point character was changed to a "+".
- The grouping separator was changed to a "-"

```
<Article xsi:noNamespaceSchemaLocation='
  <Name>1-000-000+00</Name>
</Article>
```

Rounding

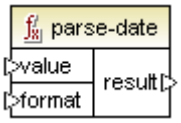
The rounding method used for this function is half up, e.g. rounds up if the fraction is > or equal to 0.5. Rounds down if fraction is <0.5.

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



number

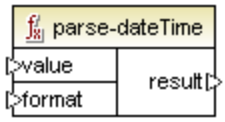
Converts an input string into a number. Also converts a boolean input to a number.



parse-date

Available for Java, C#, C++, and the Built-in execution engine.

Converts a string into an date, while ignoring the time component.



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 or PST with alphabetic modifier
z	timezone as a time offset using GMT	GMT+n

The formatting modifier:

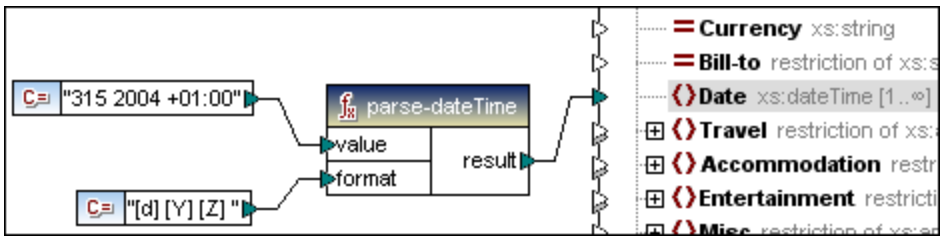
Character	Description	Example
1	decimal numeric format with no leading zeros: 1, 2, 3, ...	1, 2, 3
01	decimal format, two digits: 01, 02, 03, ...	01, 02, 03
N	name of component, upper case	MONDAY, TUESDAY
n	name of component, lower case	monday, tuesday
Nn	name of component, title case	Monday, Tuesday

1) **N, n, and Nn modifiers** only support the component M (month).

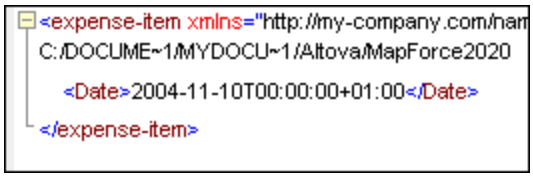
Examples:

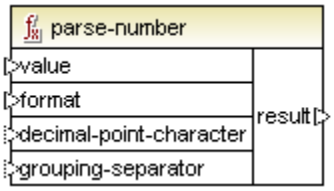
Date String	Picture String	Result
21-03-2002 16:21:12.492 GMT+02:00	[D]-[M]-[Y] [H]:[m]:[s].[f] [z]	2002-03-21T16:21:12.492+02:00
315 2004 +01:00	[d] [Y] [Z]	2004-11-10T00:00:00+01:00
1.December.10 03:2:39 p.m. +01:00	[D].[MNn].[Y,2-2] [h]:[m]:[s] [P] [Z]	2010-12-01T15:02:39+01:00
20110620	[Y,4-4][M,2-2][D,2-2]	2011-06-20T00:00:00

Example in MapForce:



Result:





parse-number

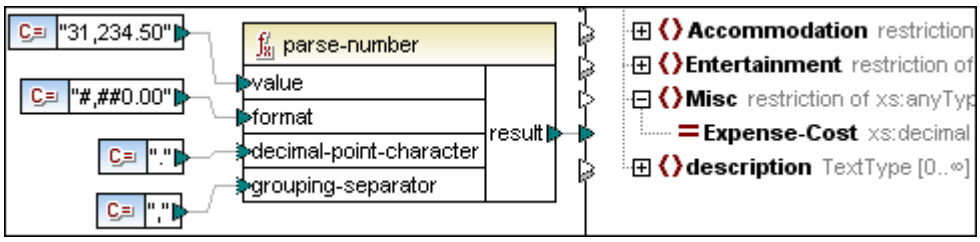
Available for Java, C#, C++, and the Built-in execution engine.

Converts an input string into a decimal number.

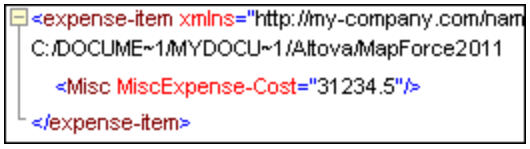
Argument	Description
value	The string to be parsed/converted to a number
format	A format string that identifies the way in which the number is currently formatted (optional). Default is "#,##0.#"
decimal-point-character	The character to be used as the decimal point character. Default is the '.' character (optional)
grouping-separator	The separator/delimiter used to separate groups of numbers. Default is the "," character (optional)

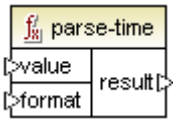
The ***format*** string used in parse-number is the same as that used in format-number.

Example in MapForce:



Result:

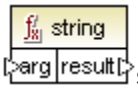




parse-time

Available for Java, C#, C++, and the Built-in execution engine.

Converts a string into an time, while ignoring the date component.



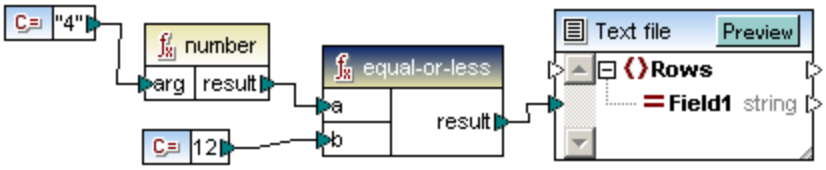
string

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.

Comparing differing input node types

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.



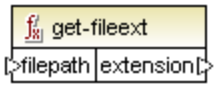
In the example above the first constant is of type string and contains the string "4".
The second constant contains the numeric constant 12. To be able to compare the two values explicitly the types must agree.

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.

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.

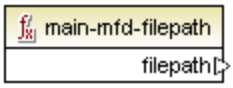


get-fileext

Returns the extension of the file path including the dot "." character.
E.g. 'c:\data\Sample.mfd' returns '.mfd'

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/'



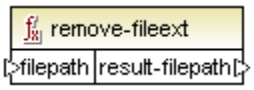
main-mfd-filepath

Returns the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.

mfd-filepath

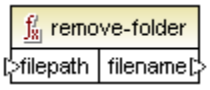
If the function is called in the main mapping, it returns the same as main-mfd-filepath function, i. e. the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.

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.



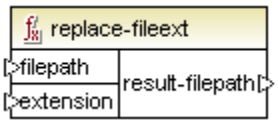
remove-fileext

Removes the extension of the file path including the dot-character.
E.g. 'c:/data/Sample.mfd' returns 'c:/data/Sample'.



remove-folder

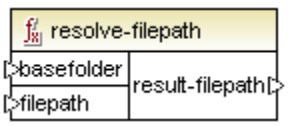
Removes the directory of the file path including the trailing slash, or backslash character.
E.g. 'c:/data/Sample.mfd' returns 'Sample.mfd'.



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.

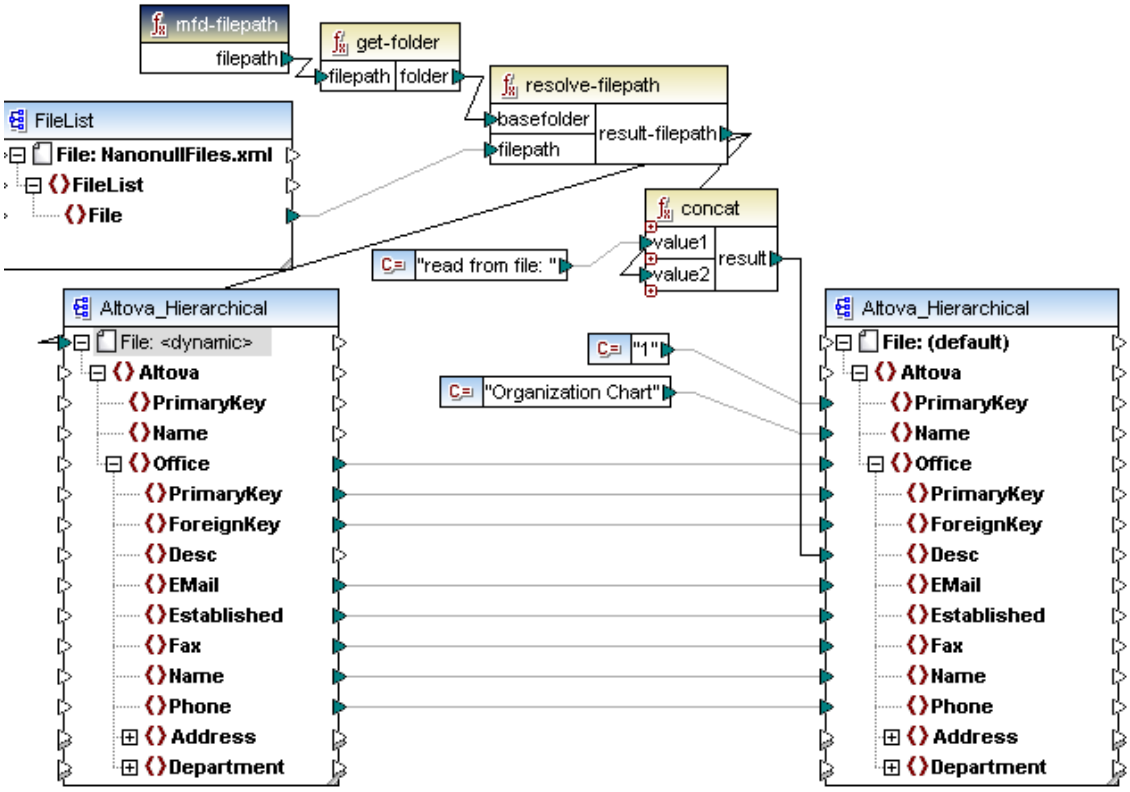
E.g. c:/data/Sample.**mfd'** as the input filepath, and '.**mfp'** as the extension, returns 'c:/data/Sample.mfp'



resolve-filepath

Resolves a relative file path to a relative, or absolute, base folder. The function supports '.' (current directory) and '..' (parent directory).

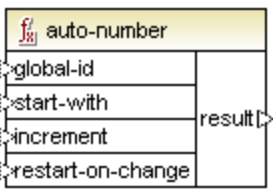
Please see the mapping **MergeMultipleFiles_List.mfd** available in the ...\MapForceExamples folder, for an example.



generator functions

The **auto-number** function generates integers in target nodes of a component, depending on the various parameters you define.

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.



auto-number

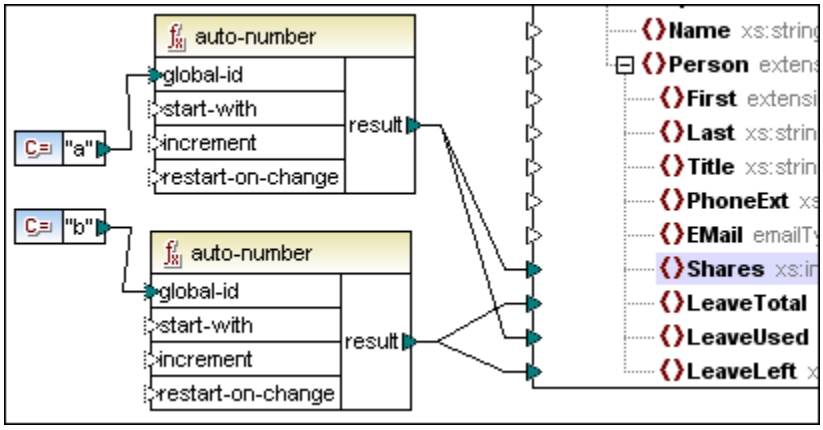
Result is a value starting at **start_with** and increased by **increment**. Default values are: start-with=1 and increase=1. Both parameters can be negative.

global-id

This parameter allows you to synchronize the number sequence output of two separate auto-number functions connected to a single target component.

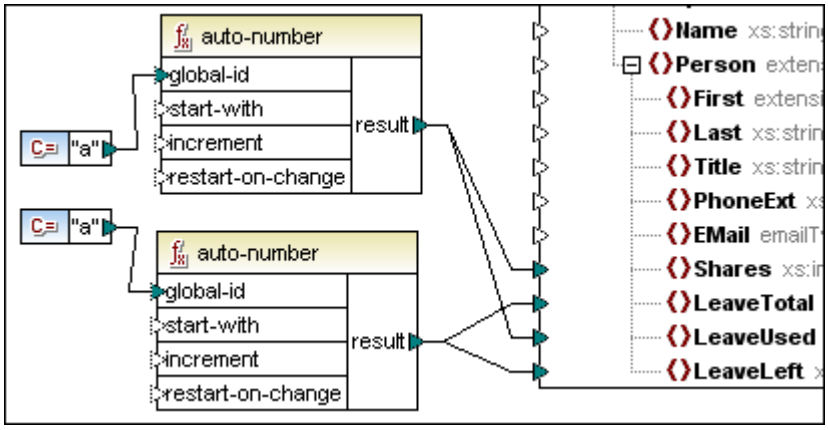
If the two auto-number functions do **not** have the same global-id, then each increments the target items separately. In the example below, each function has a different global-id i.e. a and b.

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.



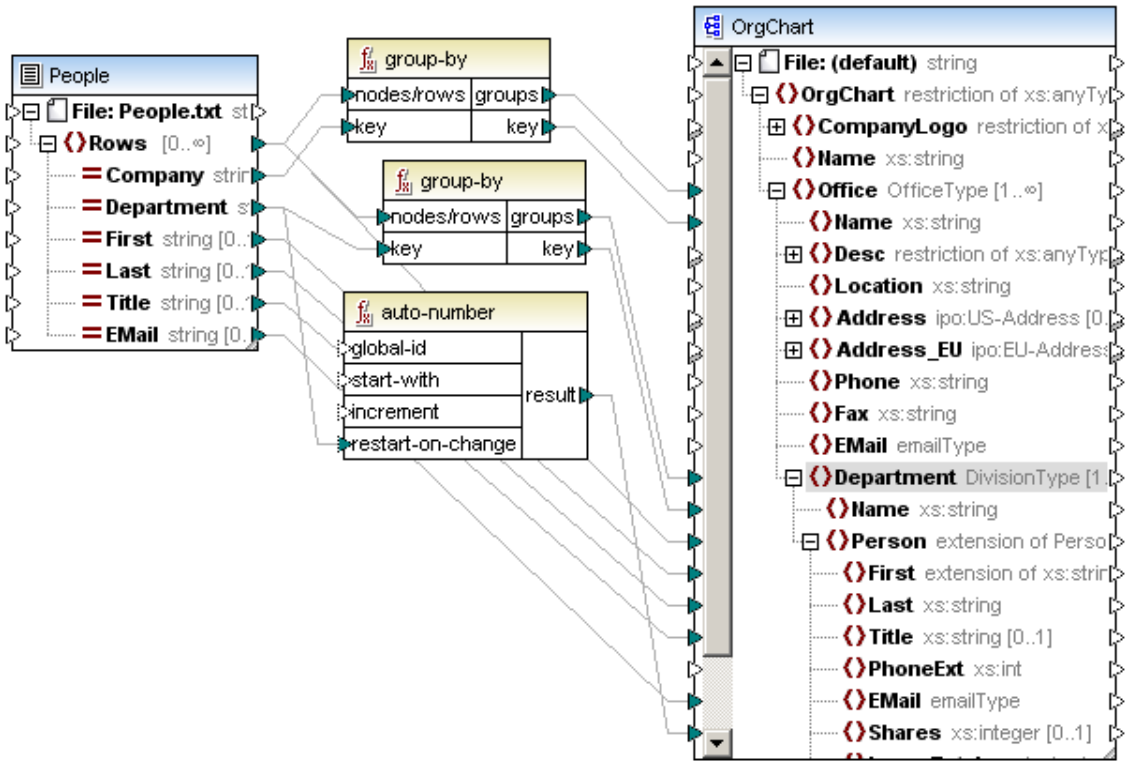
start-with
The inital value used to start the auto numbering sequence. Default is 1.

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.



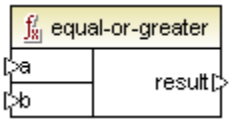
logical functions

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.

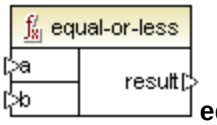
When comparing input parameters, MapForce selects the most specific common type for each parameter and then compares them. If the common type is anySimpleType, then both input parameters are compared as strings.

input parameters = **a** | **b**, or **value1** | **value2**
output parameter = result

equal
Result is true if a=b, else false.



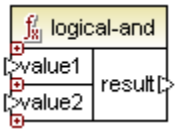
equal-or-greater
Result is true if a is equal/greater than b, else false.



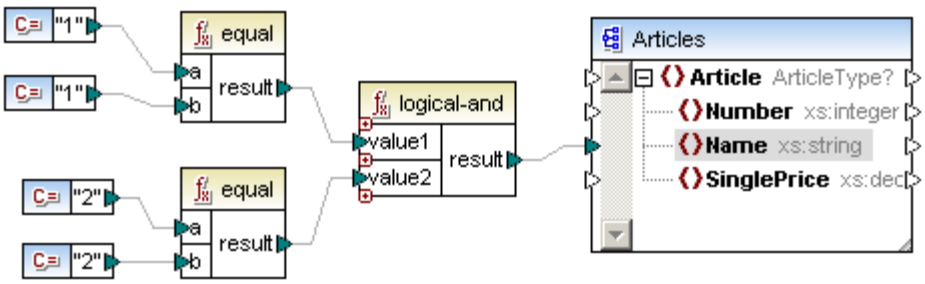
equal-or-less
Result is true if a is equal/less than b, else false.

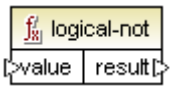
greater
Result is true if a is greater than b, else false.

less
Result is true if a is less than b, else false.



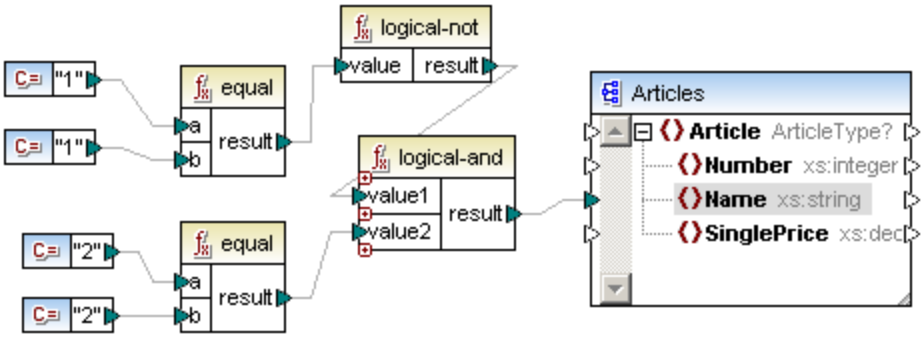
logical-and
If **both** value1 and value2 of the logical-and function are **true**, then result is true; if different then false.

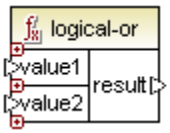




logical-not
Inverts or flips the logical state/result; if input is true, result of logical-not function is false. If input is false then result is true.

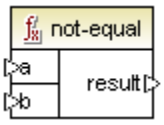
The logical-not function shown below, inverts the result of the equal function. The logical-and function now only returns true if boolean values of value1 and value2 are different, i.e. true-false, or false-true.





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.



not equal

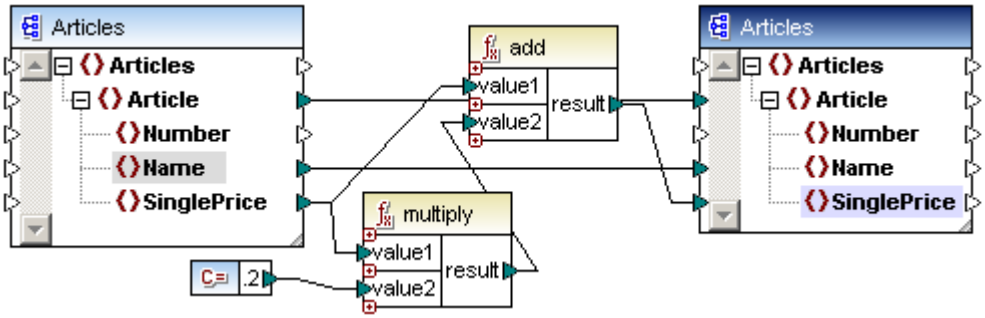
Result is true if a is not equal to b.

math functions

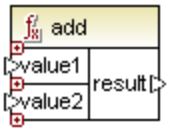
Math functions are used to perform basic mathematical functions on data. Note that they cannot be used to perform computations on durations, or datetimes.

input parameters = **value1** | **value2**
output parameter = **result**

input values are automatically converted to decimal for further processing.

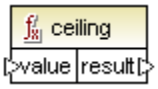


The example shown above, adds 20% sales tax to each of the articles mapped to the target component.



add

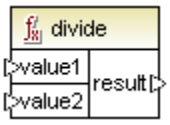
Result is the decimal value of adding **value1** to **value2**.



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.



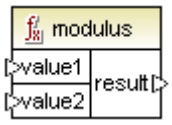
divide

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.

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.

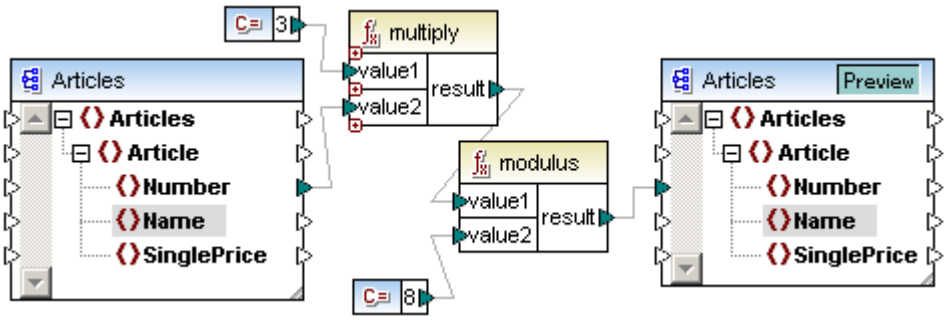


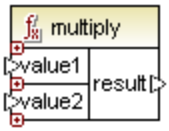
modulus

Result is the remainder of dividing **value1** by **value2**.

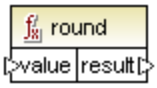
In the mapping below, the numbers have been multiplied by 3 and passed on to value1 of the modulus function. Input values are now 3, 6, 9, and 12.

When applying/using modulus 8 as value2, the remainders are 3, 6, 1, and 4.

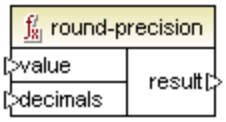




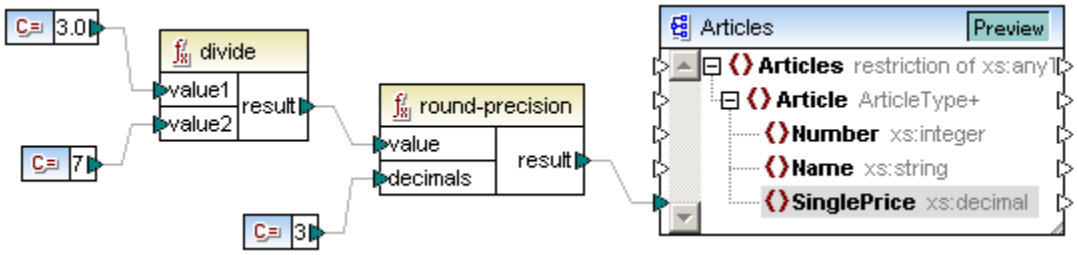
multiply
Result is the decimal value of multiplying **value1** by **value2**.



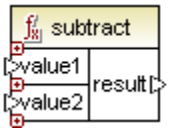
round
Result is the rounding by half, of **value** to the nearest integer.



round-precision
Result is the decimal value of the number rounded to the decimal places defined by "decimals".



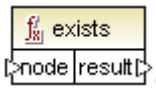
In the mapping above the result = 0.429.
If you want this result to appear correctly in an XML file, make sure to map to an element of xs: decimal type.



subtract
Result is the decimal value of subtracting **value2** from **value1**.

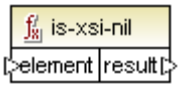
node functions

The node testing functions allow you to test for the existence/non-existence of nodes in many types of input files, XML schema, text, database, EDI and even function results. Exists actually checks for a non-empty sequence i.e. if any node exists.



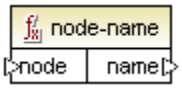
exists
Returns true if the node exists, else returns false.

Please see exists for an example.



is-xsi-nil

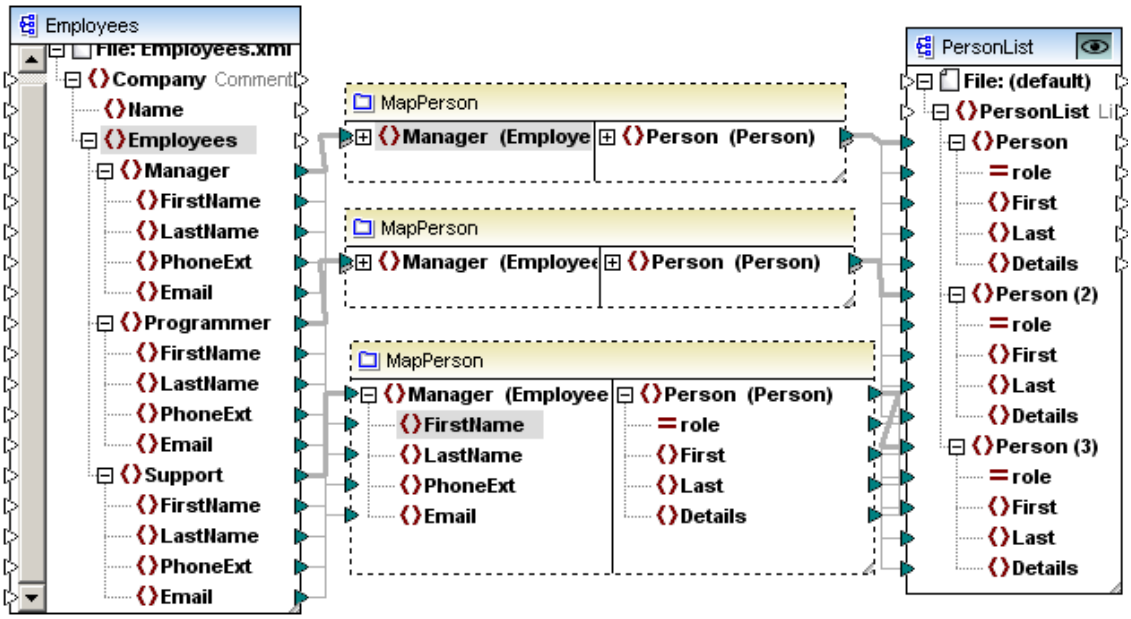
Returns true (<OrderID>true</OrderID>) if the element node, of the source componenent, has the xsi:nil attribute set to "true".



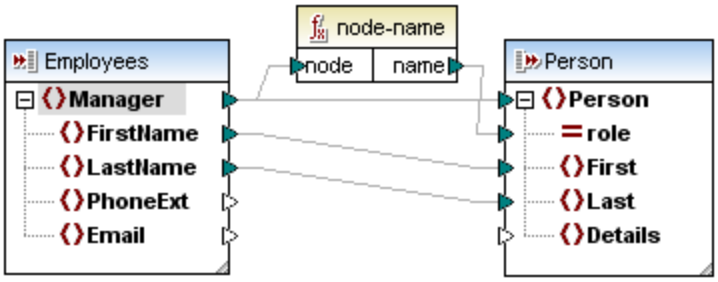
node-name

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.

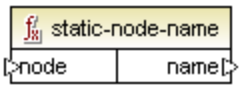
- Getting a name from database tables/fields is not supported.
- XBRL and Excel are not supported.
- Getting a name of File input node is not supported.
- WebService nodes behave like XML nodes except that:
  - node name from "part" is not supported.
  - node-name from root node ("Output" or "Input") 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".



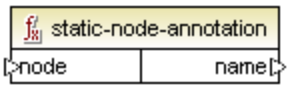
Manager gets its data from **outside** the user-defined function, where it can be either: Manager, Programmer, or Support. This is the data that is then passed on to the role attribute in PersonList.



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.

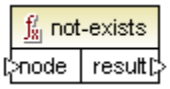
The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.



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.

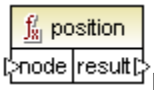
The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.



not-exists

Returns false if the node exists, else returns true.

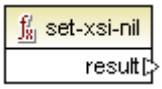
Please see not-exists for an example.



position

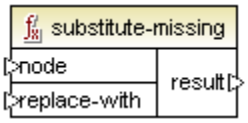
Returns the position of a node inside its containing sequence.

Please see position for an example.



set-xsi-nil

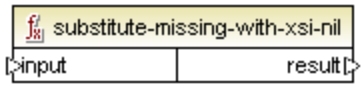
Sets the target node to xsi:nil.



subsitute-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.

Please see substitute-missing for an example.



substitute-missing-with-xsi-nil

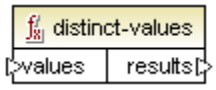
Substitutes any missing (or null values) of the source component, with the xsi:nil attribute in the target node.

sequence functions

Sequence functions are not available if XSLT (XSLT 1.0) has been selected.

MapForce supports sequence functions which allow the processing of input sequences and the grouping of their content. The value/content of the **key** input parameter, mapped to nodes/rows, is used to group the sequence.

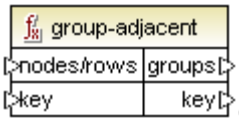
- Input parameter **key** is of an arbitrary data type that can be converted to string for **group-adjacent** and **group-by**
- Input parameter **bool** is of type Boolean for **group-starting-with** and **group-ending-with**
- The output **key** is the key of the current group.



distinct-values

Allows you to remove duplicate values from a sequence and map the unique items to the target component.

Please see distinct-values for an example.

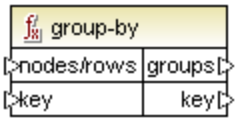


group-adjacent

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!

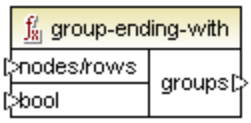
Please see group-adjacent for an example.



group-by

Groups the input sequence **nodes/rows** into groups of not neccessarily adjacent items sharing the same **key**. Goups are output in the order the key occurs in the input sequence.

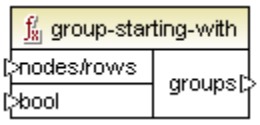
Please see group-by for an example.



group-ending-with

This function groups the input sequence **nodes/rows** into groups, ending a new group whenever **bool** is true.

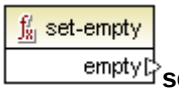
Please see group-ending-with for an example.



group-starting-with

This function groups the input sequence **nodes/rows** into groups, starting a new group whenever **bool** is true.

Please see group-starting-with for an example.

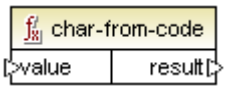


set-empty

Allows you to cancelof an XBRL document that were defined higher up the XBRL component/ taxonomy.

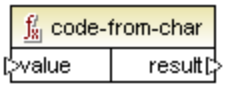
string functions

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.



char-from-code

Result is the character representation of the decimal Unicode value of **value**.

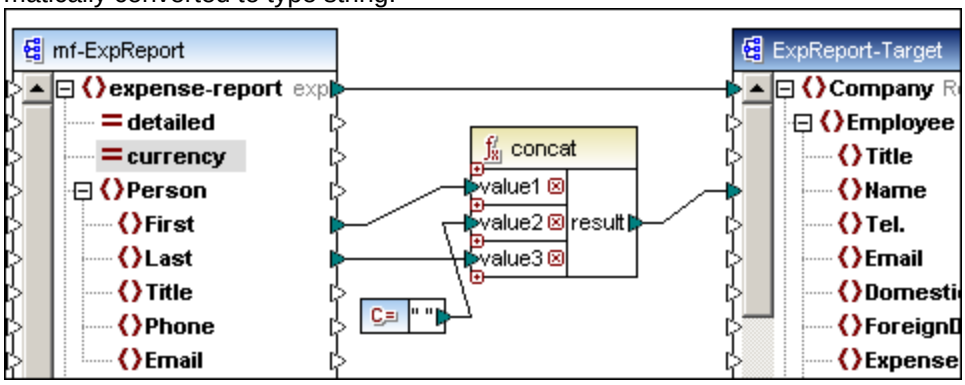


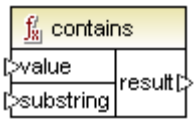
code-from-char

Result is the decimal Unicode value of the first character of **value**.

concat

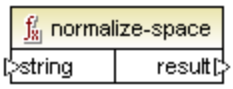
Concatenates (appends) two or more values into a single result string. All input values are auto matically converted to type string.





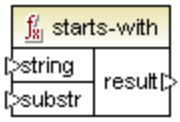
contains

Result is true if data supplied to the value parameter contains the string supplied by the substring parameter.



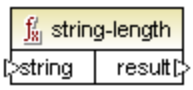
normalize-space

Result is the normalized input string, i.e. leading and trailing spaces are stripped out, and multiple consecutive whitespace characters are condensed into a single whitespace character.



starts-with

Result is true if the input string "string" starts with **substr**, else false.



string-length

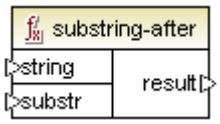
Result is the number of characters supplied by the **string** parameter.

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.

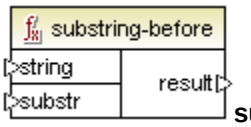
E.g. substring("56789",2,3) results in 678.



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.



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**.

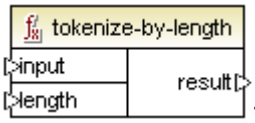
E.g. substring-before ("2009/01/04","/") results in the substring 2009. substr in this case is the first "**/"** character.

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.

Please see Tokenize examples for a specific example supplied with MapForce.

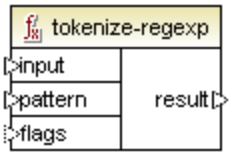


tokenize-by-length

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.

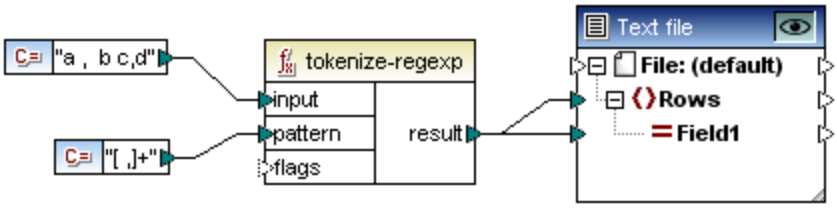
E.g. Input string is ABCDEF and length is "2" - then result is AB CD EF.

Please see Tokenize examples for a specific example supplied with MapForce.



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	

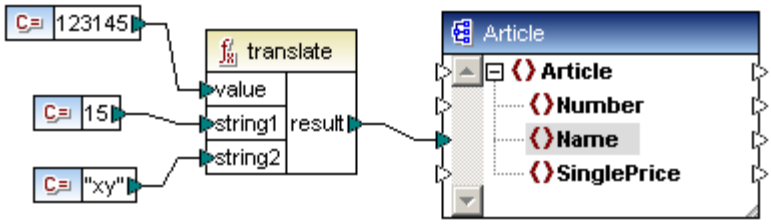
Please note that there are slight differences in regular expression syntax between the various languages. Tokenize-regexp in C++ is only available in Visual Studio 2008 SP1 and later.

For more information on regular expressions please see: Regular expressions.

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"

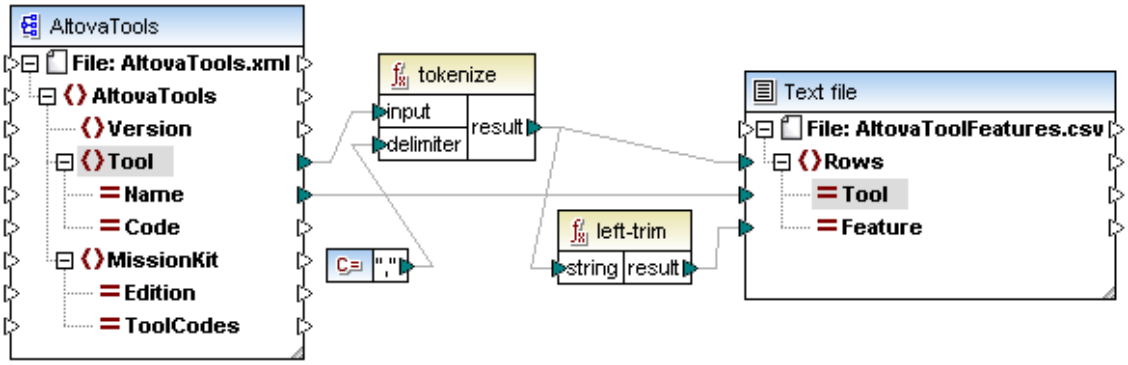
E.g.3
input string aabaacbca
string1 is "**ac**"
string2 is "**ca**"

result string is "ccbccabac"

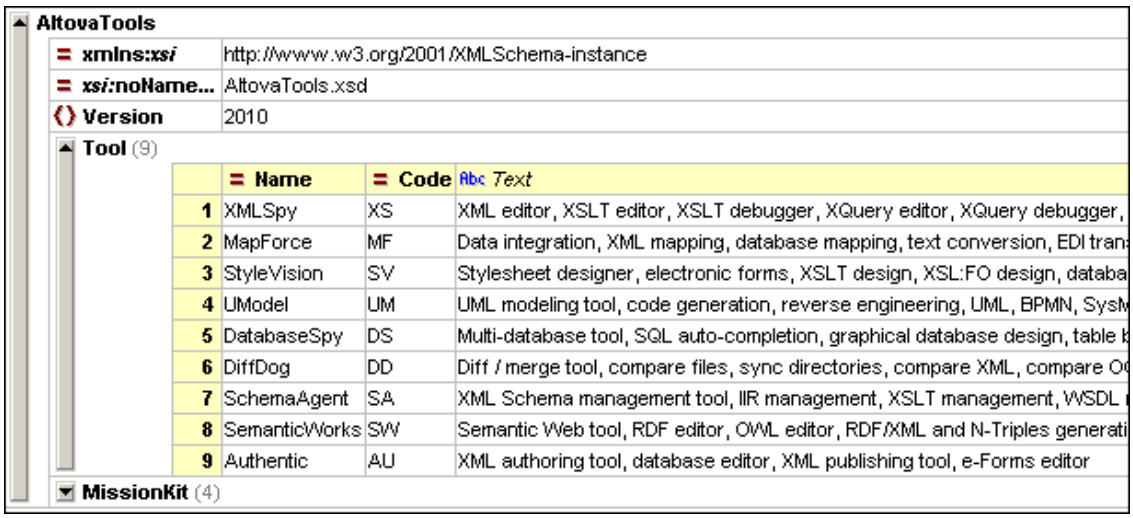
Tokenize examples

Example **tokenize**

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
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).

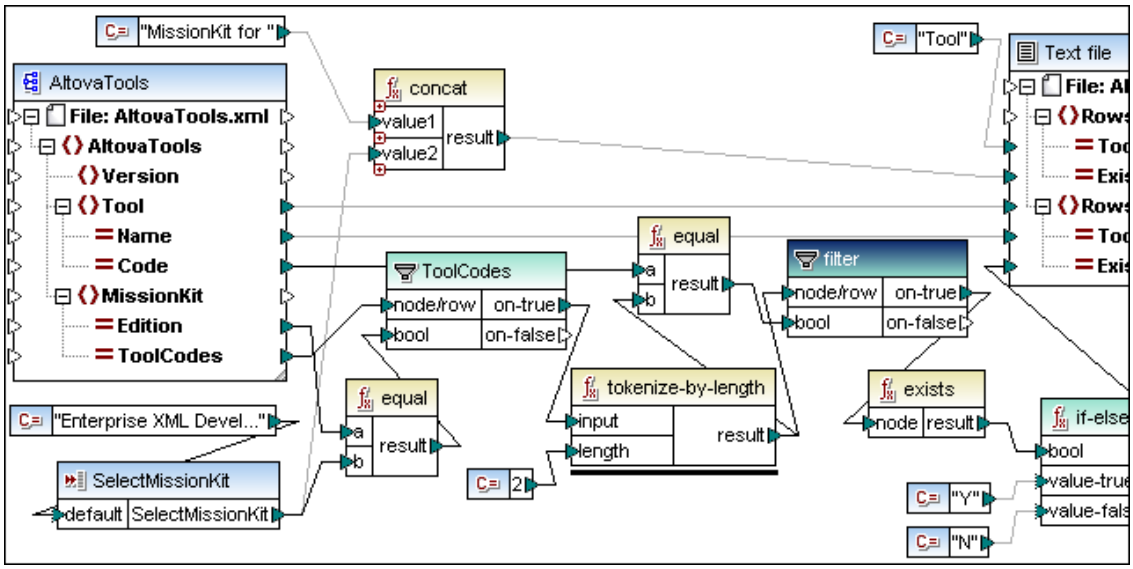
Result of the mapping:

- 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
```

Example **tokenize-by-length**

The **tokenizeString2.mfd** file available in the ...\MapForceExamples folder shows how the **tokenize-by-length** function is used.



The XML source file is shown below, and is the same as the one used in the previous example. The **MissionKit** element also has two attributes: **Edition** and **ToolCodes**, but no MissionKit element content.

Tool (9)

	Name	Code	Text
1	XMLSpy	XS	XML editor, XSLT editor, XSLT debugger, XQuery editor, XQuery debugger, XML S
2	MapForce	MF	Data integration, XML mapping, database mapping, text conversion, EDI translator,
3	StyleVision	SV	Stylesheet designer, electronic forms, XSLT design, XSL:FO design, database rep
4	UModel	UM	UML modeling tool, code generation, reverse engineering, UML, BPMN, SysML, pro
5	DatabaseSpy	DS	Multi-database tool, SQL auto-completion, graphical database design, table brows
6	DiffDog	DD	Diff / merge tool, compare files, sync directories, compare XML, compare OOXML,
7	SchemaAgent	SA	XML Schema management tool, IIR management, XSLT management, WSDL manag
8	SemanticWorks	SW	Semantic Web tool, RDF editor, OWL editor, RDF/XML and N-Triples generation and
9	Authentic	AU	XML authoring tool, database editor, XML publishing tool, e-Forms editor

MissionKit (4)

	Edition	ToolCodes
1	Enterprise Software Architects	XSMFSVUMDSDDSASW
2	Professional Software Architects	XSMFSVUMDS
3	Enterprise XML Developers	XSMFSVDDSASW
4	Professional XML Developers	XSMFSV

Aim of the mapping:
To generate a list showing which Altova tools are part of the respective MissionKit editions.

How the mapping works:

- The SelectMissionKit **Input** component receives its default input from a constant component, in this case "Enterprise XML Developers".
- The **equal** function compares the input value with the "**Edition**" value and passes on the result to the **bool** parameter of the ToolCodes filter.

- The **node/row** input of the ToolCodes filter is supplied by the **ToolCodes** item of the source file. The value for the Enterprise XML Developers edition is: XSMFSVDDSASW.
- The XSMFSVDDSASW value is passed to the **on-true** parameter, and 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 function.

  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
```

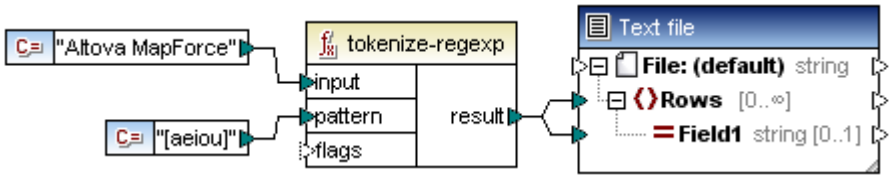
Regular expressions

MapForce can use regular expressions in the **pattern** parameter of the the **match-pattern** and **tokenize-regexp** functions, to find specific strings of the input parameter.

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.

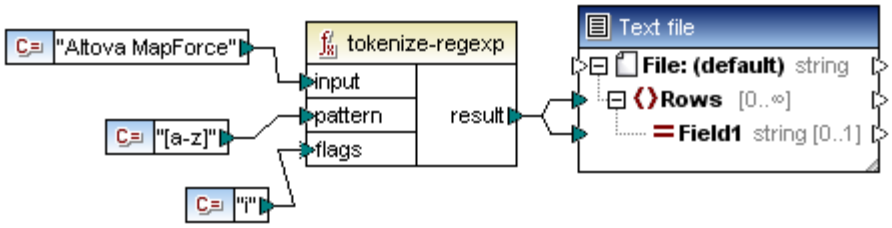
m
If present, the matching process operates in multi-line mode.

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.



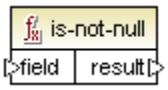
x
If present, whitespace characters are removed from the input string prior to the matching process. Whitespace chars. are #x09, #x0A, #x0D and #x20.

Exception:
Whitespace characters within character class expressions are not removed e.g. [#x20].

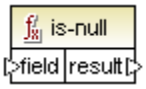
Please note:
When generating code, the advanced features of the regex syntax might differ slightly between the various languages, please see the specific regex documentation for your language.

18.6.2 db

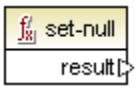
The db library contains functions that allow you to define the mapping results when encountering null fields in databases.



is-not-null
Returns false if the field is null, otherwise returns true.



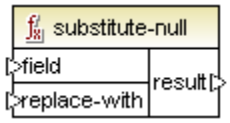
is-null
Returns true if the field is null, otherwise returns false.



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.

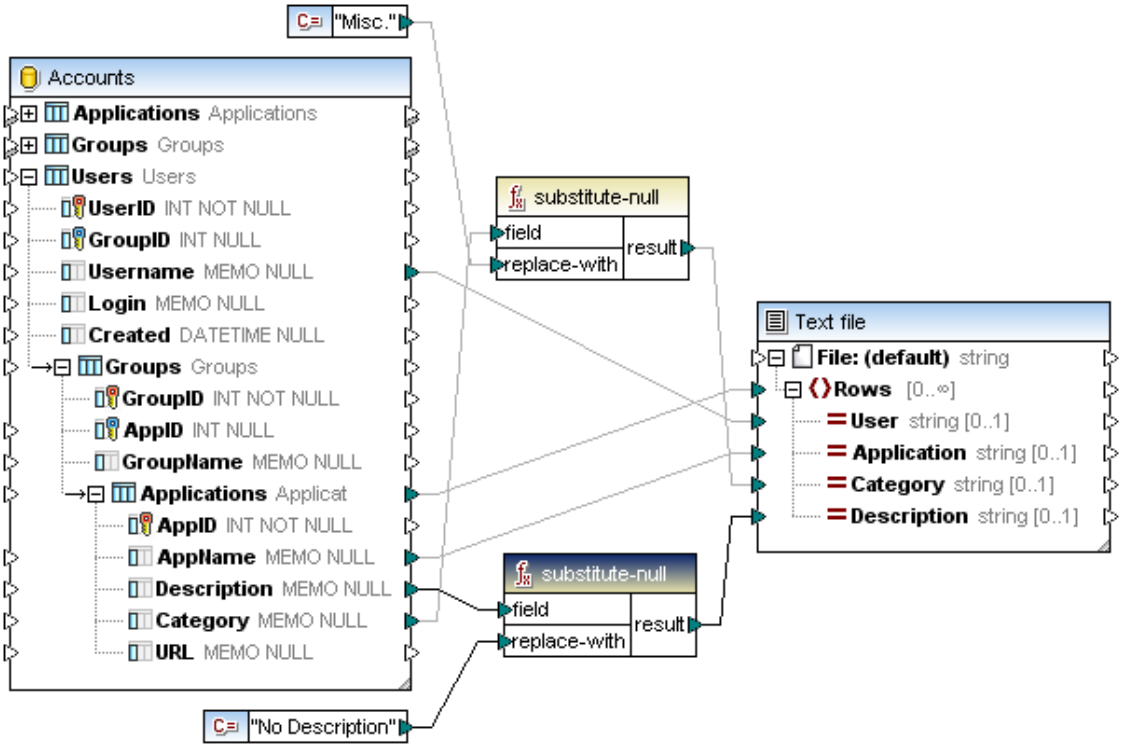
Please note:

- 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.
- Connecting set-null to a simpleType element will not create that element in the target component.
- Connecting set-null to a complexType element, as well as a table or row, is not allowed. A validaton error occurs when this is done.



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.

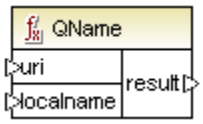
The second function checks if a Description entry exist, and maps the string "No description" if one does not exist, which is also the case with the Notepad application.

18.6.3 lang

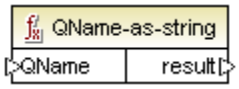
The lang library contains an assortment of functions that are available when selecting either Java, C#, or C++ languages.

- QName functions
- datetime functions
- generator functions
- logical functions
- math functions
- string functions

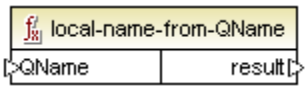
QName functions



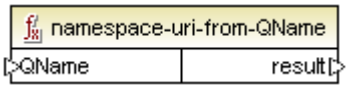
QName
Result is a QName constucted from the namespace URI and the local name.



QName-as-string
Result is the unique string representation of the QName.

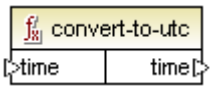


local-name-from-QName
Result is the local name part of the QName.

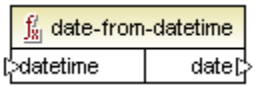


namespace-uri-from-QName
Result is the namespace URI part of the QName

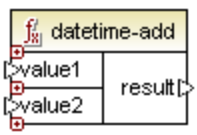
datetime functions



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. +1:00, into account).

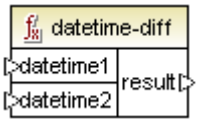


date-from-datetime
Result is the date part of a datetime input argument.



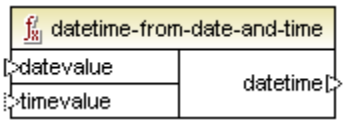
datetime-add
Result is the datetime obtained by adding a duration (second argument) to a datetime (first

argument).



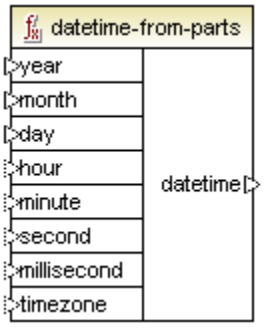
datetime-diff

Result is the duration obtained by subtracting datetime2 (second argument) from datetime1 (first argument).



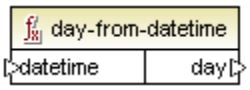
datetime-from-date-and-time

Result is a datetime built from a datevalue (first argument) and a timevalue (second argument).



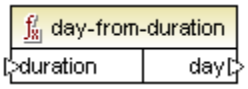
datetime-from-parts

Result is a datetime built from any combination of the following parts submitted as arguments: year, month, day, hour, minute, second, millisecond, timezone. This function automatically normalizes the supplied parameters e.g. 32nd of January will automatically be 1st February.



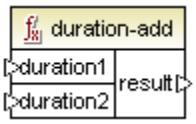
day-from-datetime

Result is the day represented by the submitted datetime argument.



day-from-duration

Result is the day represented by the submitted duration argument.

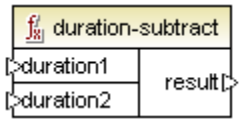


duration-add

Result is the duration obtained by adding two durations.

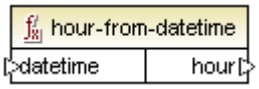
duration-from-parts

Result is a duration calculated by combining the following parts submitted as arguments: year, month, day, hour, minute, second, millisecond, negative.



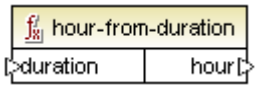
duration-subtract

Result is the duration obtained by subtracting duration2 from duration1.



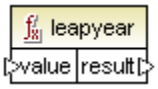
hour-from-datetime

Result is the hour part of the submitted datetime argument.



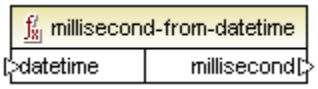
hour-from-duration

Result is the hour component of the submitted duration argument.



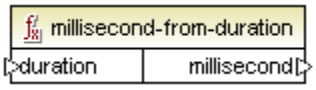
leapyear

Result is true or false according to whether the year of the submitted dateTime is in a leap year.



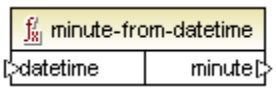
millisecond-from-datetime

Result is the millisecond part of the submitted datetime argument.



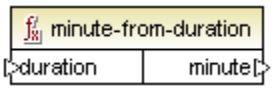
millisecond-from-duration

Result is the millisecond component of the submitted duration argument.



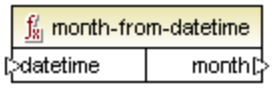
minute-from-datetime

Result is the minute part of the submitted datetime argument.

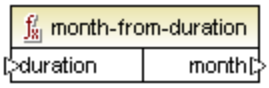


minute-from-duration

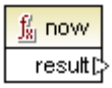
Result is the minute component of the submitted duration argument.



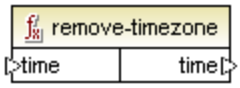
month-from-datetime
Result is the month part of the submitted dateTime argument.



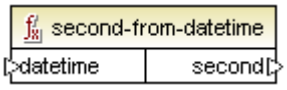
month-from-duration
Result is the month component of the submitted duration argument.



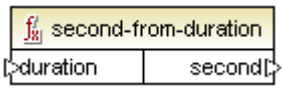
now
Result is the current dateTime (including timezone).



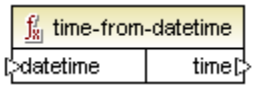
remove-timezone
Removes the timezone component, e.g. +1:00, from the time input parameter.



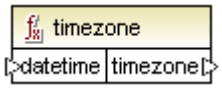
second-from-datetime
Result is the seconds part of the submitted dateTime argument.



second-from-duration
Result is the seconds component of the submitted duration argument.

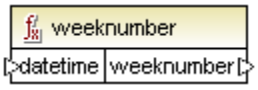


time-from-datetime
Result is the time part of the submitted dateTime argument.

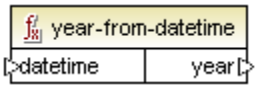


timezone
Returns the timezone relative to UTC of the submitted dateTime value. Unit of difference is minutes

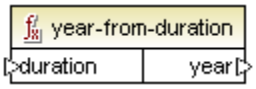
weekday
Returns the weekday of the submitted dateTime value, starting with `Monday=1` to `Sunday=7`.



weeknumber
Returns the number of the week within the year represented by the submitted dateTime value.



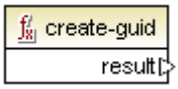
year-from-datetime
Result is the year part of the submitted dateTime argument.



year-from-duration
Result is the year component of the submitted duration argument.

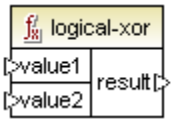
generator functions

The generator functions generate values for database fields, which do not have any input data from the Schema, database or EDI source component.

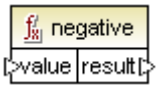


create-guid
Result is a globally-unique identifier (as a hex-encoded string) for the specific field.

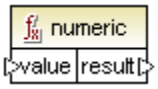
logical functions



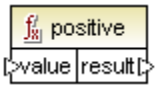
logical-xor
Result is true if value1 is different than value2, otherwise false.



negative
Result is true if value is negative, i.e. less than zero, otherwise false.

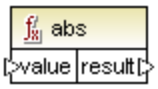


numeric
Result is true if value is a number, otherwise false. The input will usually be a string.

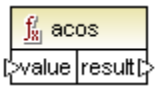


positive
Result is true if value is positive, i.e. equal to or greater than zero, otherwise false.

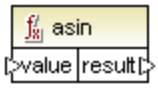
math functions



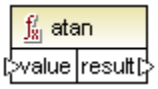
abs
Result is the absolute value of the input **value**.



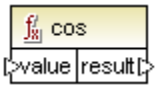
acos
Result is the arc cosine of **value**.



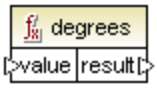
asin
Result is the arc sine of **value**.



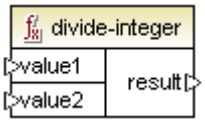
atan
Result is the arc tangent of **value**.



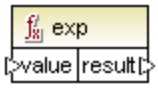
cos
Result is the cosine of **value**.



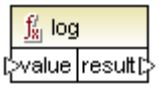
degrees
Result is the conversion of **value** in radians into degrees.



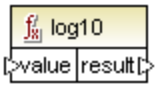
divide-integer
Result is the integer result of dividing **value1** by **value2**. E.g. 15 divide-integer 2, integer result is 7.



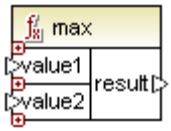
exp
Result is **e** (base natural logarithm) raised to the **value**th power.



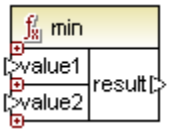
log
Result is the natural logarithm of **value**.



log10
Result is logarithm (base 10) of **value**.



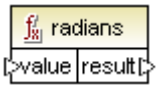
max
Result is the numerically larger value of **value1** compared to **value2**.



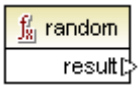
min
Result is the numerically smaller value of **value1** compared to **value2**.

pi
Result is the value of pi.

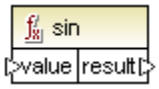
pow
Result is the value of **a** raised to the power **bth power**.



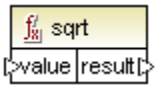
radians
Result is the conversion of **value** in degrees to radians.



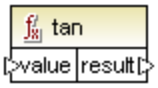
random
Result is a pseudorandom value between 0.0 and 1.0



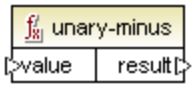
sin
Result is the sine of **value**.



sqrt
Result is the square root of **value**.

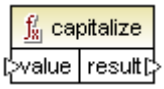


tan
Result is the tangent of **value**.

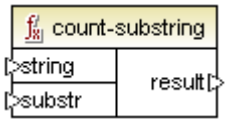


unary-minus
Result is the negation of of the signed input **value**. E.g. +3 result is -3, while -3 result is 3.

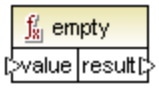
string functions



capitalize
Result is the input string **value,** where the first letter of each word is capitalized (initial caps).

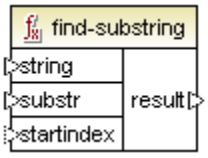


count-substring
Result is the number of times that **substr** occurs in **string**.



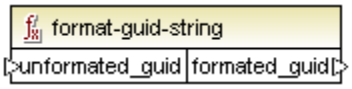
empty

Result is true if the input string **value** is empty, otherwise false.



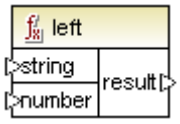
find-substring

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.



format-guid-string

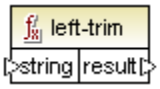
Result is a correctly formatted GUID string **formatted_guid,** using **unformatted_guid** as the input string, for use in database fields. See also the create guid function in the generator functions.



left

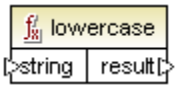
Result is a string containing the first **number** characters of **string**.

E.g. string="This is a sentence" and number=4, result is "This".



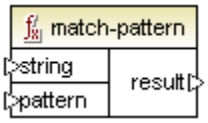
left-trim

Result is the input string with all leading whitespace characters removed.



lowercase

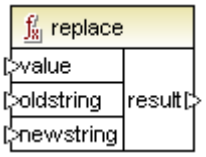
Result is the lowercase version of the input **string**. For Unicode characters the corresponding lower-case characters (defined by the Unicode consortium) are used.



match-pattern

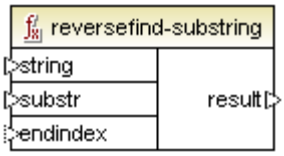
Result is true if the input **string** matches the regular expression defined by **pattern**, else false. The specific regular expression syntax depends on the target language.

Please see: Regular expressions for more information on regular expressions.



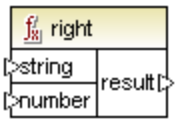
replace

Result is a new string where each instance of **oldstring**, in the input string **value**, is replaced by **newstring**.



reversefind-substring

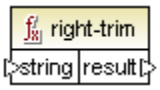
Returns the position of the first occurence of **substr**. within **string**, starting at position **endindex** , i.e. from right to left. The first character has position 1. If the substring could not be found, then the result is 0.



right

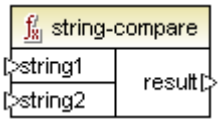
Result is a string containing the last **number** characters of **string.**

E.g. string="This is a sentence" and number=5, result is "tence".



right-trim

Result is the input string with all trailing whitespace characters removed.

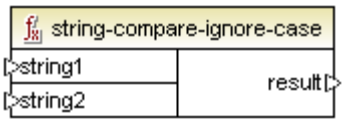


string-compare

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

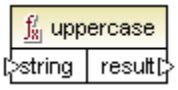


string-compare-ignore-case

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.



uppercase

Result is the string input converted into uppercase. For Unicode characters the corresponding upper-case characters (defined by the Unicode consortium) are used.

18.6.4 xpath2

XPath2 functions are available when either XSLT2 or XQuery langauges are selected.

- accessor functions
- anyURIfunctions
- boolean functions
- constructors
- context functions
- durations, date and time functions
- node functions
- numeric functions
- QName functions
- string functions

accessors

The following accessor functions are available:

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.

document-uri
Not implemented.

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).

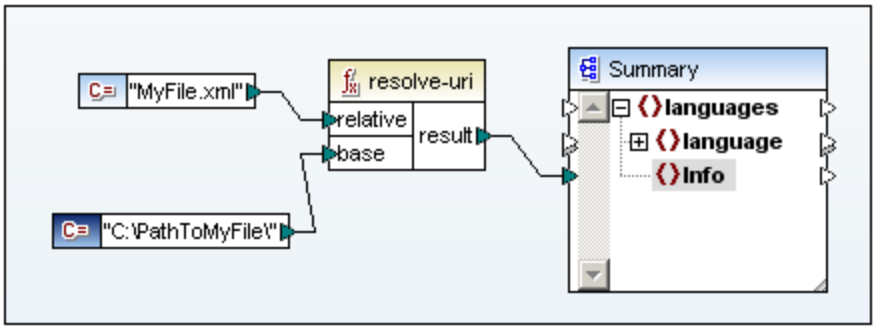
string
The `string` function works like the `xs:string` constructor: it converts its argument to `xs:string`.

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.)

anyURI functions

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 teh second argument is not supplied.

boolean functions

The Boolean functions `true` and `false` take no argument and return the boolean constant values, `true` and `false`, respectively. They can be used where a constant boolean value is required.

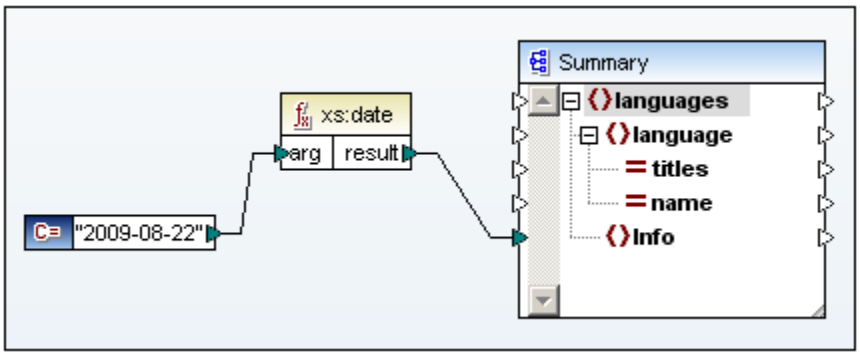
true
Inserts the boolean value "true".

false
Inserts the boolean value "false".

constructors

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.

context functions

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.

Date-time functions

The `current-date`, `current-time`, and `current-dateTime` functions take no argument and return the current date and/or time from the system clock.

The datatype of the result depends on the particular function: `current-date` returns `xs:date`, `current-time` returns `xs:time`, and `current-dateTime` returns `xs:dateTime`.

default-collation

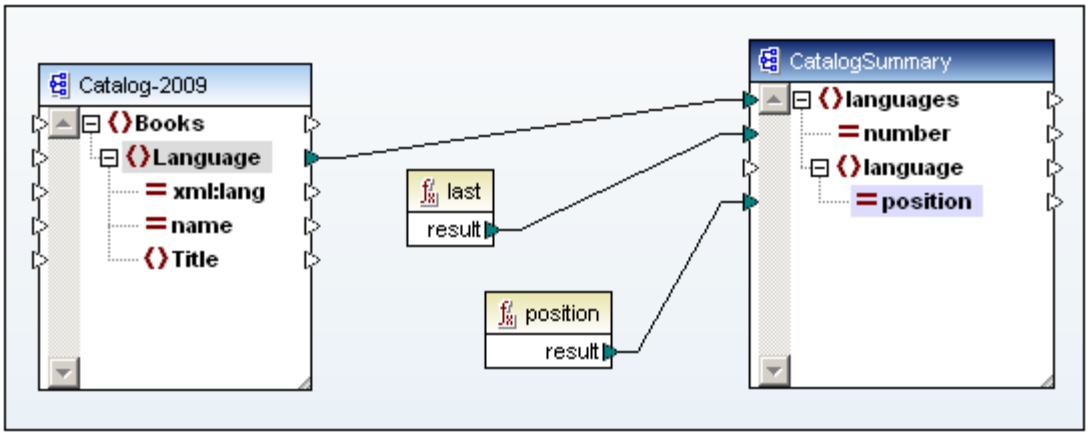
The default-collation function takes no argument and returns the default collation, that is, the collation that is used when no collation is specified for a function where one can be specified.

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.

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 `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.

durations, date and time functions

The 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.

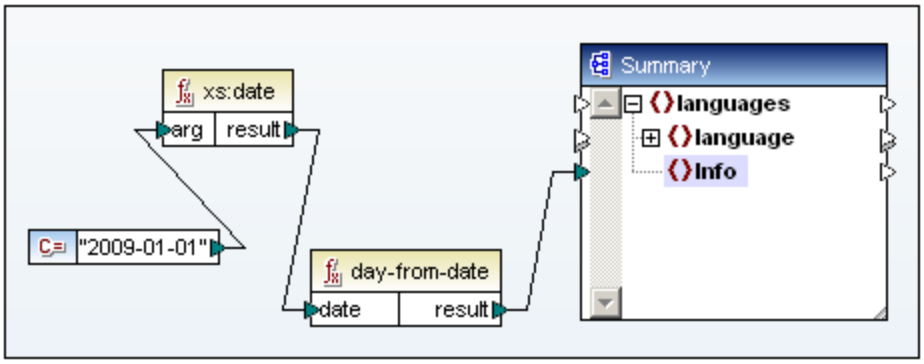
The following situations are possible when the first argument contains a timezone (for example, the date `2009-01-01+01:00` or the time `14:00:00+01:00`).

- 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

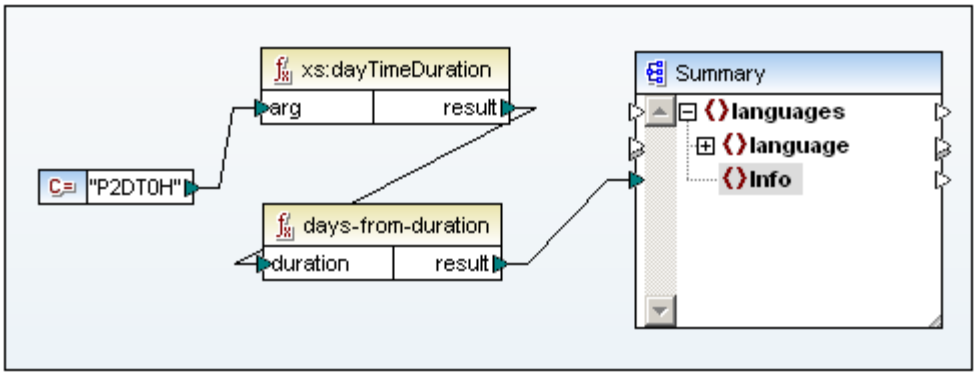
Each of the 'From' functions extracts a particular component from: (i) date or time data, and (ii) duration data. The results are of the xs:decimal datatype.

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.

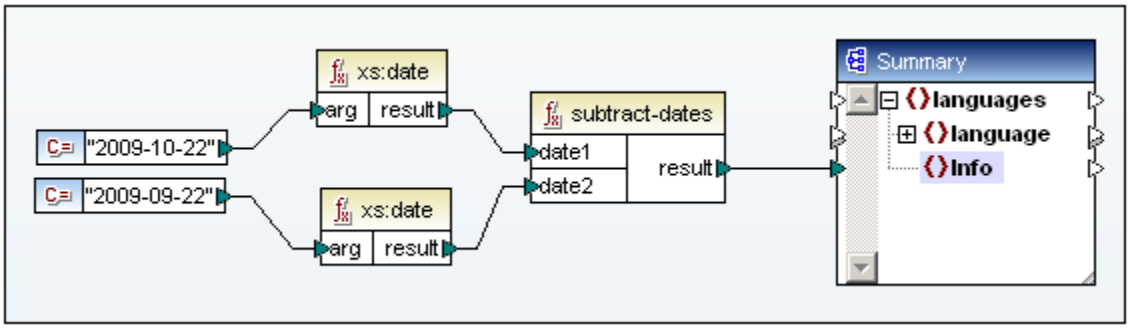
Extraction of time components from durations requires that the duration be specified either as `xs:yearMonthDuration` (for extracting years and months) or `xs:dayTimeDuration` (for extracting days, hours, minutes, and seconds). The result will be of type `xs:decimal`. The screenshot below shows a `dayTimeDuration` 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 funstions 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`.

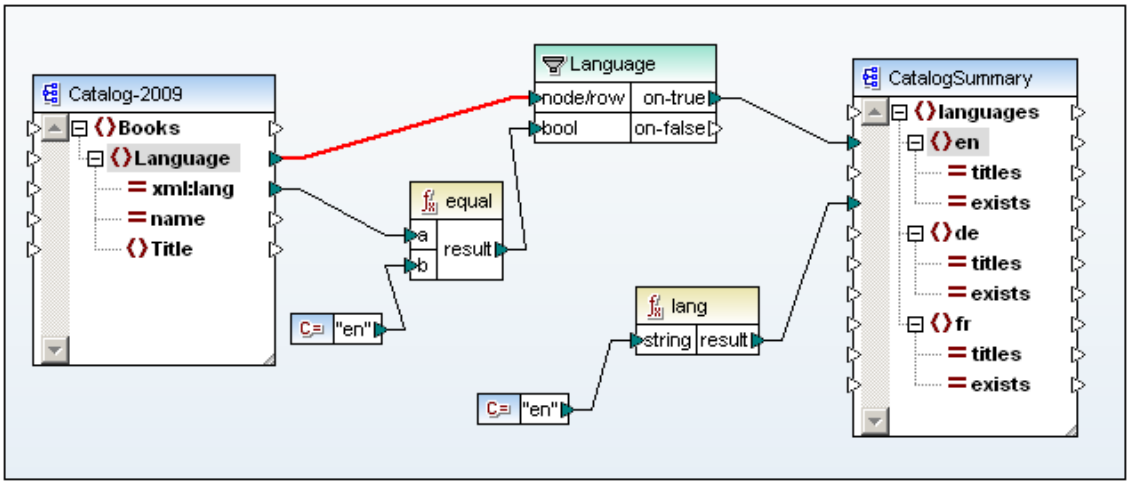


Note: When you mouseover the input argument in a function box, the expected datatype of the argument is displayed in a popup.

node functions

The following 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.

numeric functions

The following 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`.

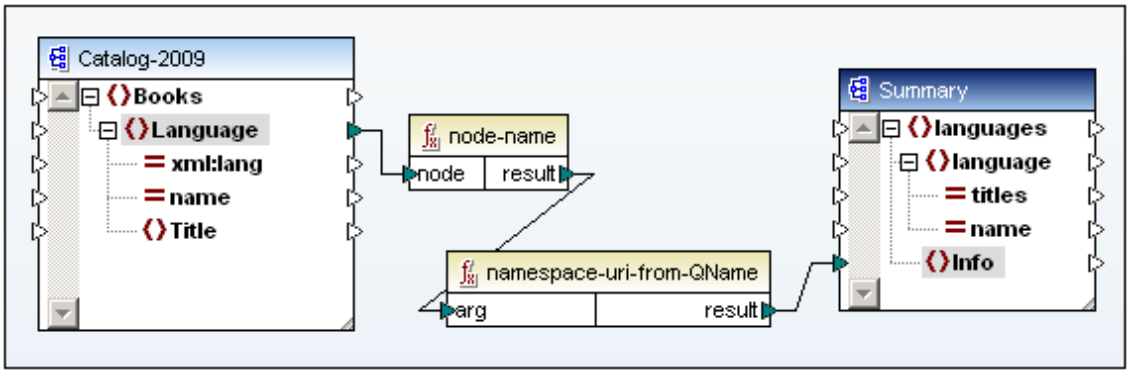
qname-related functions

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.

string functions

The following 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.

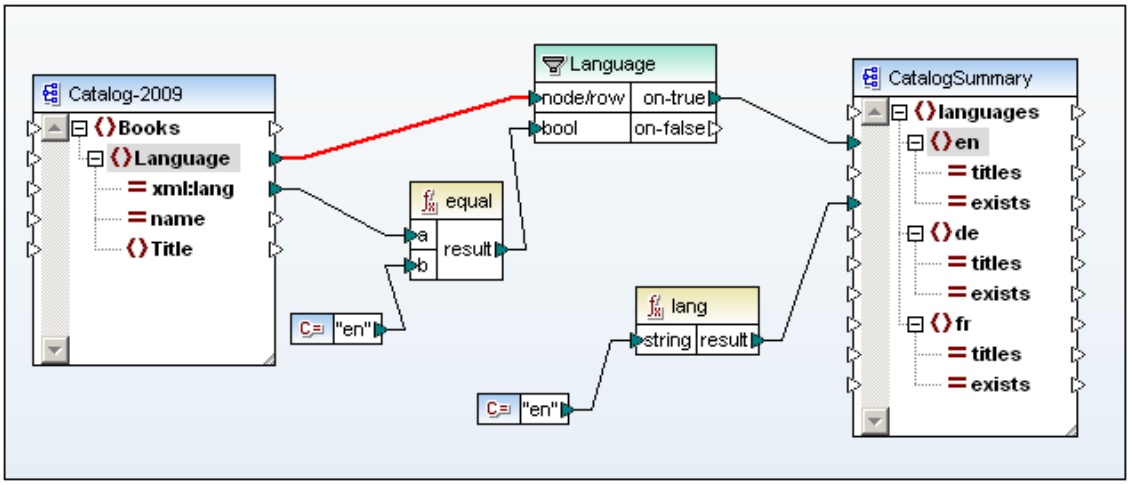
18.6.5 xslt

- xpath functions
  The functions in the XPath Functions library are XPath 1.0 nodeset functions.
- xslt functions
  The functions in the XSLT Functions library are XSLT 1.0 functions.

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:

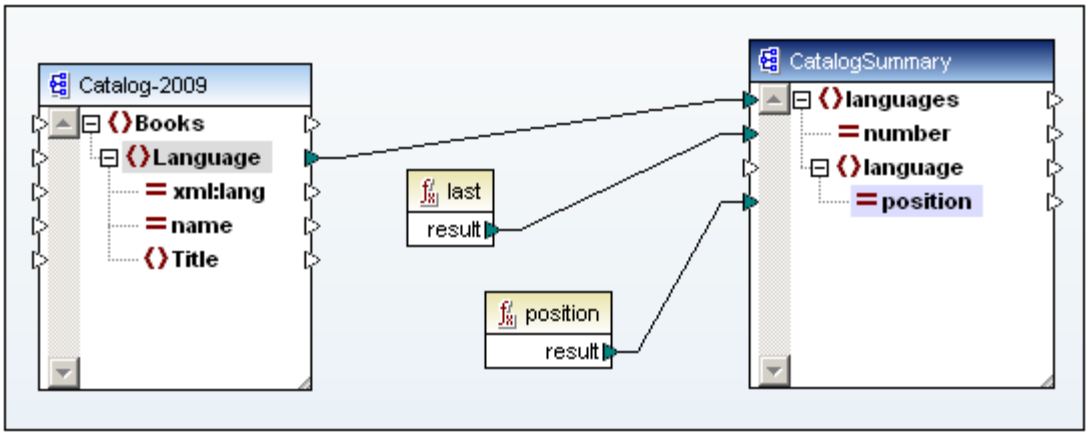
1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. `Language` nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the 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.

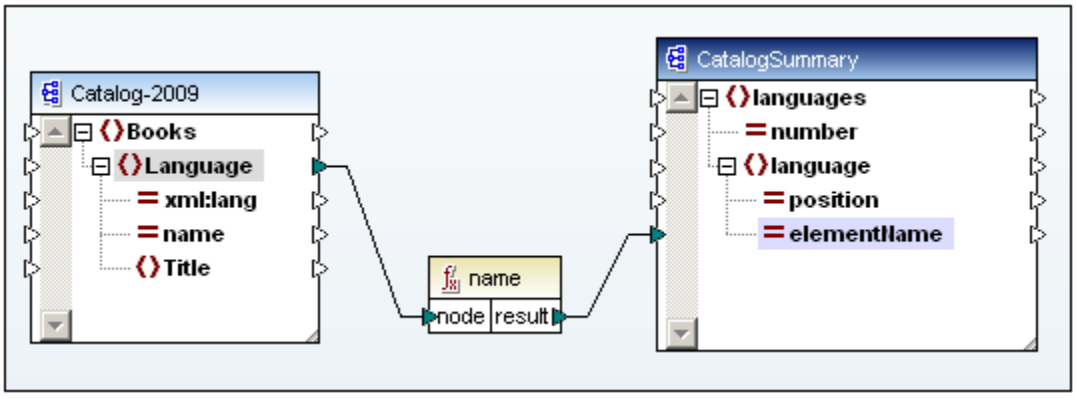


In the example above, the `last` function returns the position of the last node of the context nodeset (the nodeset of `Language` elements) as the value of the `number` attribute. This value is also the size of the nodeset since it indicates the number of nodes in the nodeset.

The `position` function returns the position of the `Language` node being currently processed. For each `Language` element node, its position within the nodeset of `Langauge` 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.

xslt functions

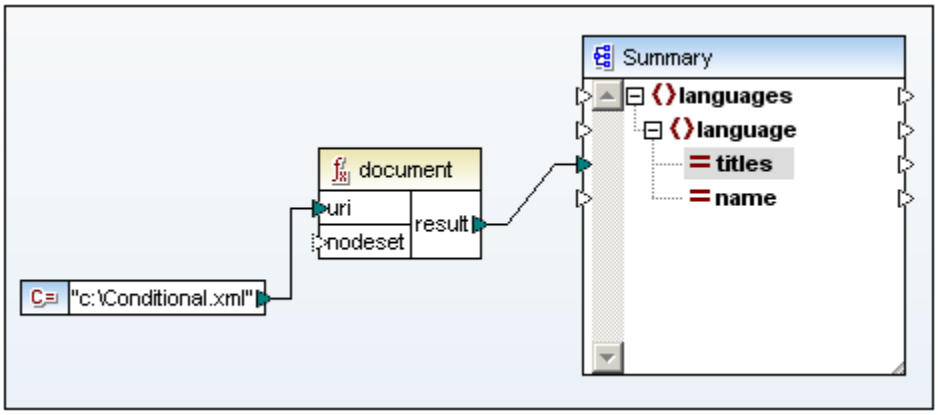
The functions in the XSLT Functions library are XSLT 1.0 functions, and are described below. Drag a function into the mapping to use it. When you mouseover the input argument part of a function box, the expected datatype of the argument is displayed in a popup.

current

The current function takes no argument and returns the current node.

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.

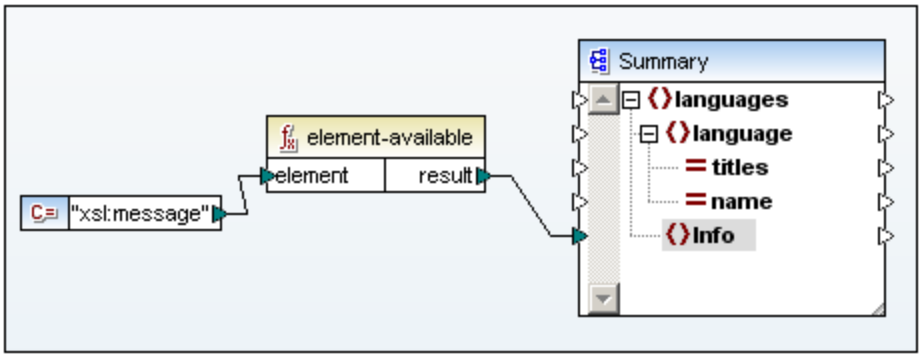


Note that the `uri` argument is a string that must be an absolute file path.

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.



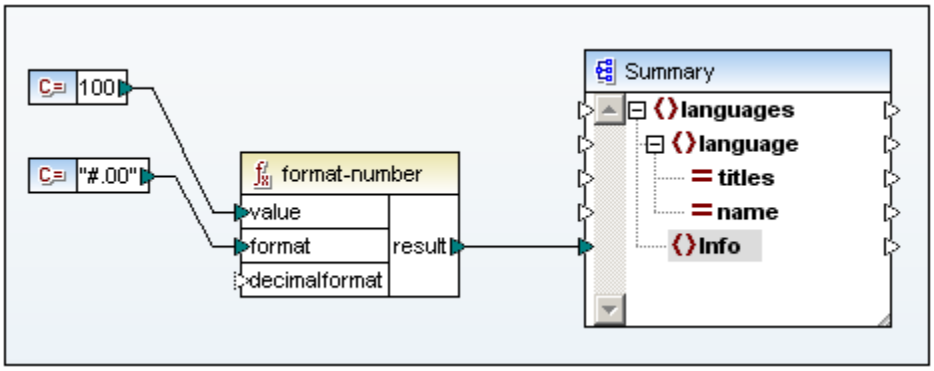
The function returns a boolean.

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.

format-number
The `format-number` takes an integer as its first argument (`value`) and a format string as its second argument (`format`). The third optional argument is a string that names the decimal format to use. If this argument is not used, then the default decimal format is used.

Decimal formats are defined by the XSLT 1.0 `decimal-format` element: each decimal format so defined can be named and the name can be used as the third argument of the `format-number` function. If a decimal format is defined without a name, it is the default decimal format for the transformation.



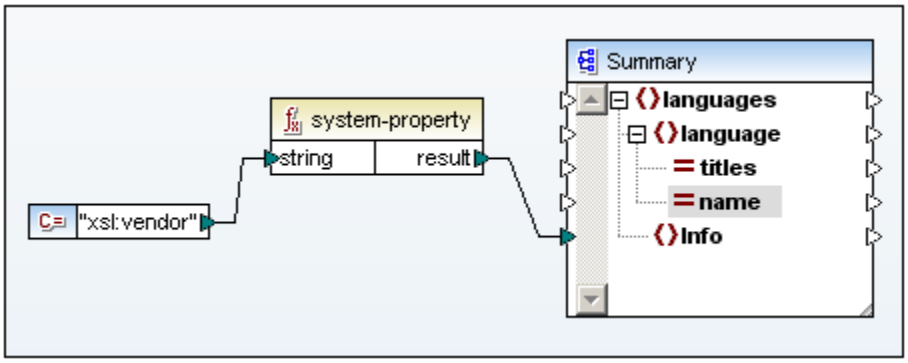
The function returns the number formatted as a string.

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.

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.



unparsed-entity-uri
If you are using a DTD, you can declare an unparsed entity in it. This unparsed entity (for example an image) will have a URI that locates the unparsed entity.

The input string of the function must match the name of the unparsed entity that has been declared in the DTD. The function then returns the URI of the unparsed entity, which can then be directed to a node in the output document, for example, to an `href` node.

Chapter 19

QName support

19 QName support

QNames (Qualified Names) allow you reference and abbreviate namespace URIs used in XML and XBRL instance documents. There are two types of QNames; Prefixed or Unprefixed QNames.

PrefixedName	Prefix	**:**	LocalPart
UnPrefixedName			LocalPart

where LocalPart is an Element or Attribute name.

```
<Doc xmlns:x="http://myCompany.com">
        <x:part>
</Doc>
```

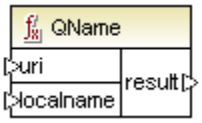
x is the namespace reference to "**http://myCompany.com**" and <x:part> is therefore a valid QName, as:

x is the namespace prefix
part is the LocalPart, i.e. the element name.

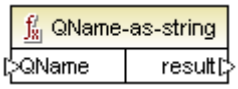
When mapping from source to target components QName prefixes will be resolved.

MapForce supports the following QName functions in the **Lang** section of the Libraries pane:

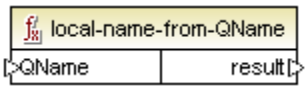
QName
Constructs a QName from a namespace URI and a local part. Use this function to create a QName in a target component. The uri and localname parameters can be supplied by a constant function.



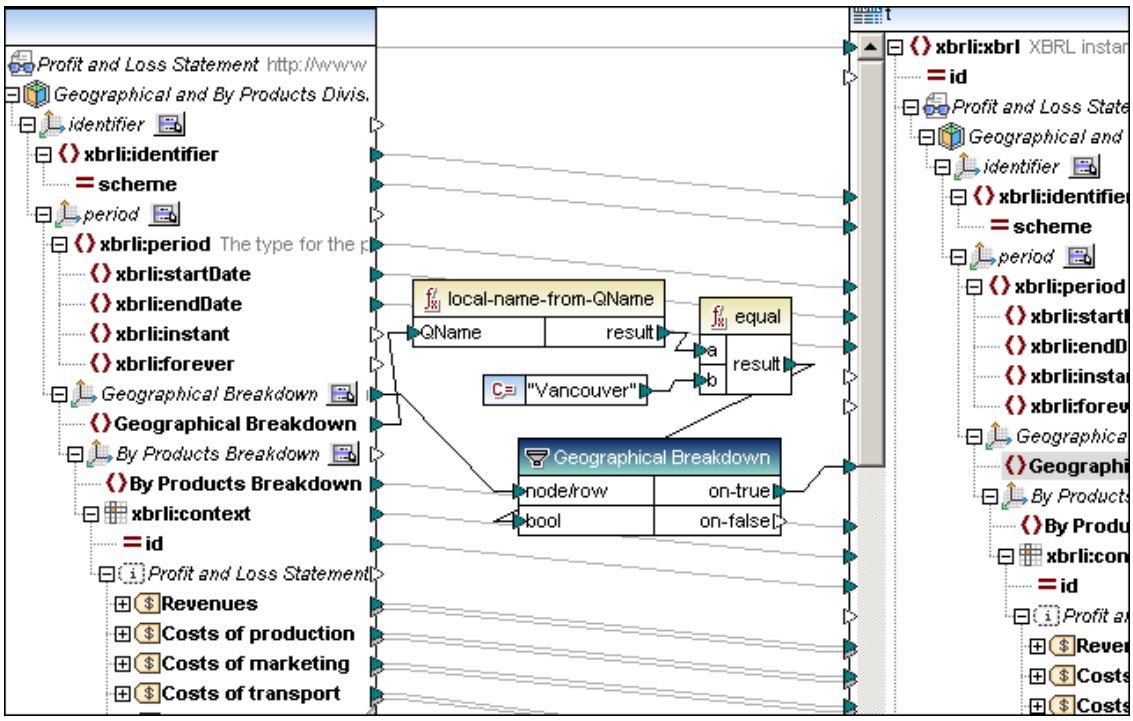
QName-as-string
Converts a QName to a string in the form **{http://myCompany.com}local**.



local-name-from-QName
Extracts the local name from a QName.



This function is extremely useful when mapping XBRL instance documents containing hypercubes.



What the mapping does is filter out 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 out 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>
```

namespace-uri-from-QName
Extracts the namespace URI from a QName.

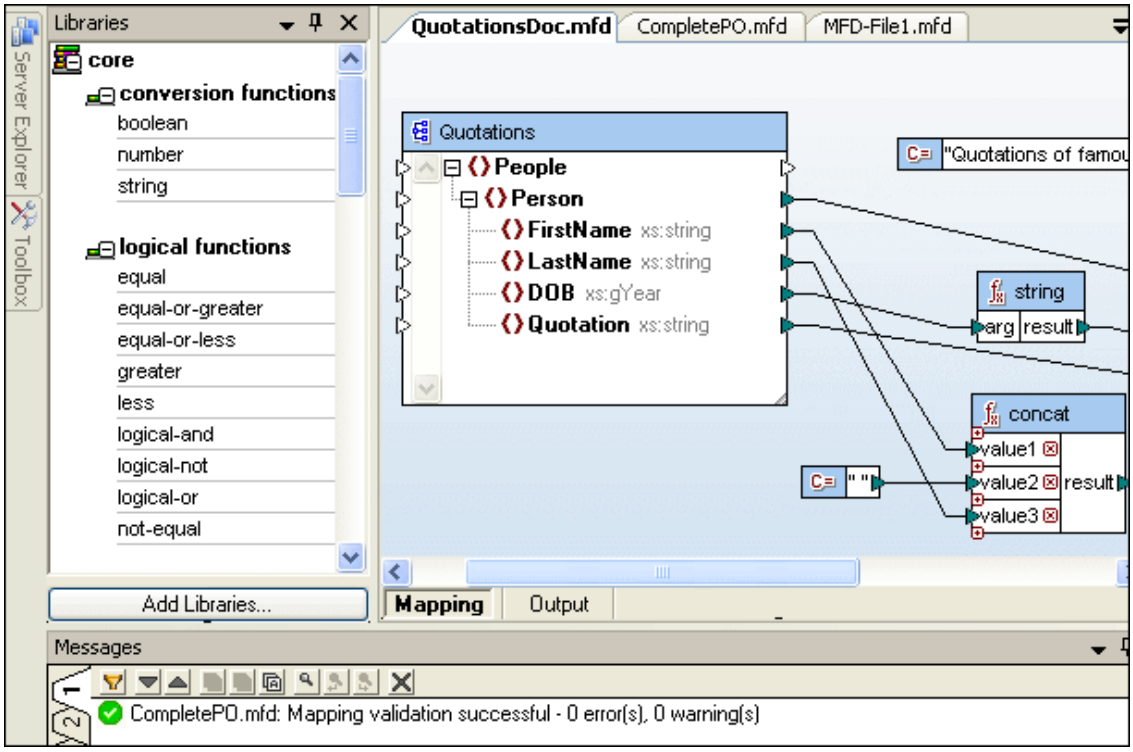
Chapter 20

MapForce plug-in for MS Visual Studio

20 MapForce plug-in for MS Visual Studio

You can integrate your version of MapForce2011 into the Microsoft Visual Studio versions 2005, 2008 and 2010. This unifies the best of both worlds, integrating advanced mapping capabilities with the advanced development environment of Visual Studio. To do this, you need to do the following:

- Install Microsoft Visual Studio
- Install MapForce (Enterprise or Professional Edition)
- Download and run the **MapForce Integration Package**. This package is available on the MapForce (Enterprise and Professional Editions) download page at www.altova.com.



Once the integration package has been installed, you will be able to use MapForce in the Visual Studio environment.

How to enable the plug-in
It is possible that the plug-in was not automatically enabled during the installation process.

To enable the plug-in:

1. Navigate to the **directory** Visual Studio IDE executable was installed in, e.g. c:\Program Files\Microsoft Visual Studio 8\Common7\IDE
2. Enter the following command on the command-line **devenv.exe /setup**.
3. Wait for the process to terminate normally before starting to use the application within Visual Studio.

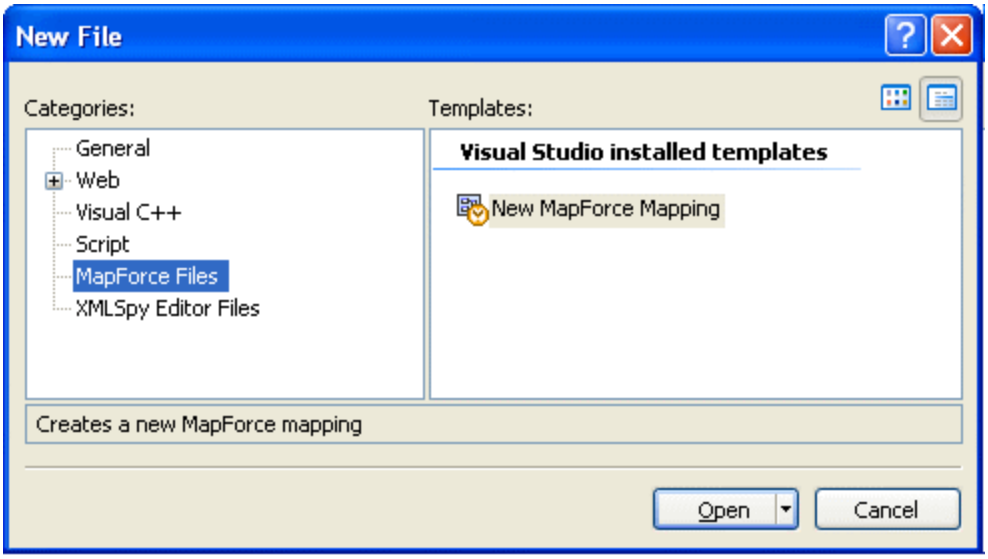
Please note:
Before you take a look at the sample project please add the **assemblies** to the .NET IDE Toolbox. The MapForce Installer will have already installed the assemblies in the .NET Global Assembly Cache (GAC).

If you open the Toolbox dialog under **Tools | Add/Remove Toolbox Items** the controls will appear as AxMapForceControl, AxMapForceControlDocument and AxMapForceControlPlaceholder on the .NET Framework Components tab. Check all of them to make them available to the IDE.

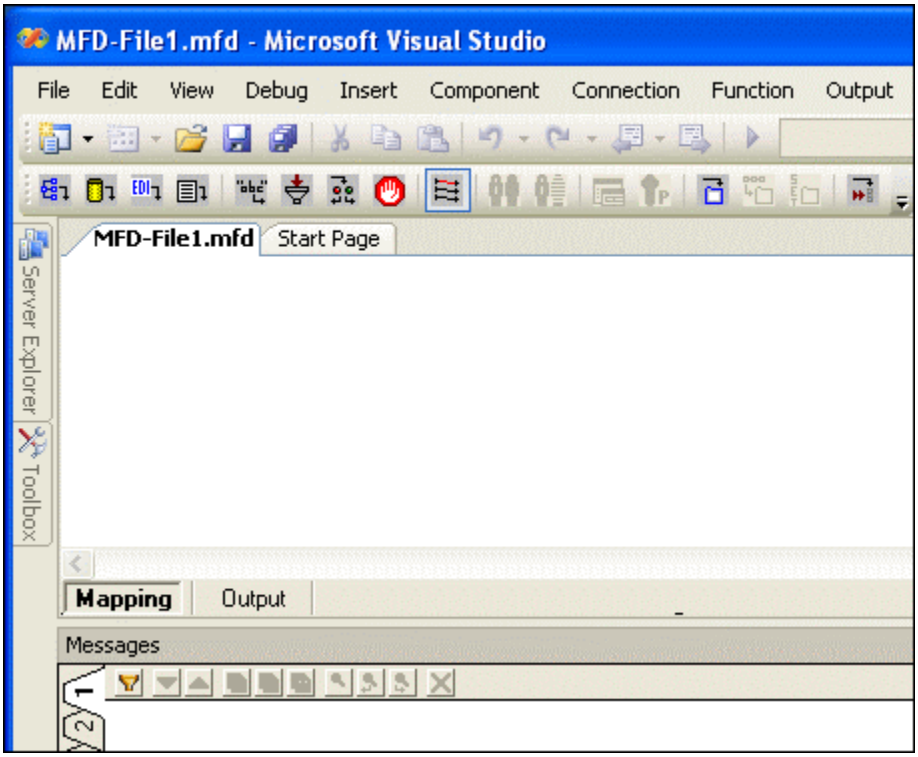
20.1 Opening MapForce files in Visual Studio

To open a new MapForce mapping file:

1. Select the menu option **File | New**.
2. Click the MapForce Files entry in Categories.

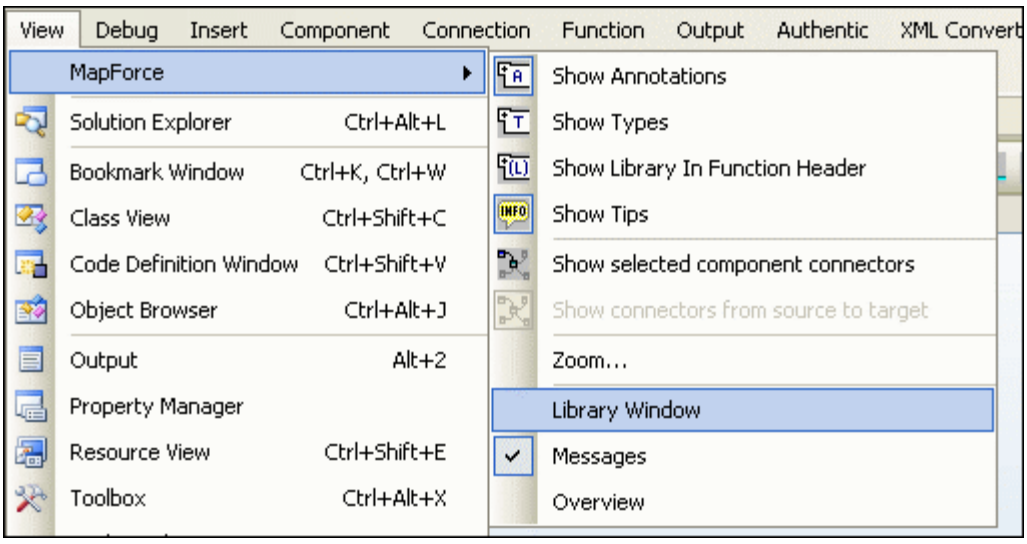


3. Double click the "New MapForce Mapping" item in the Templates window.
   An empty mapping file is opened.



To enable the Libraries window:

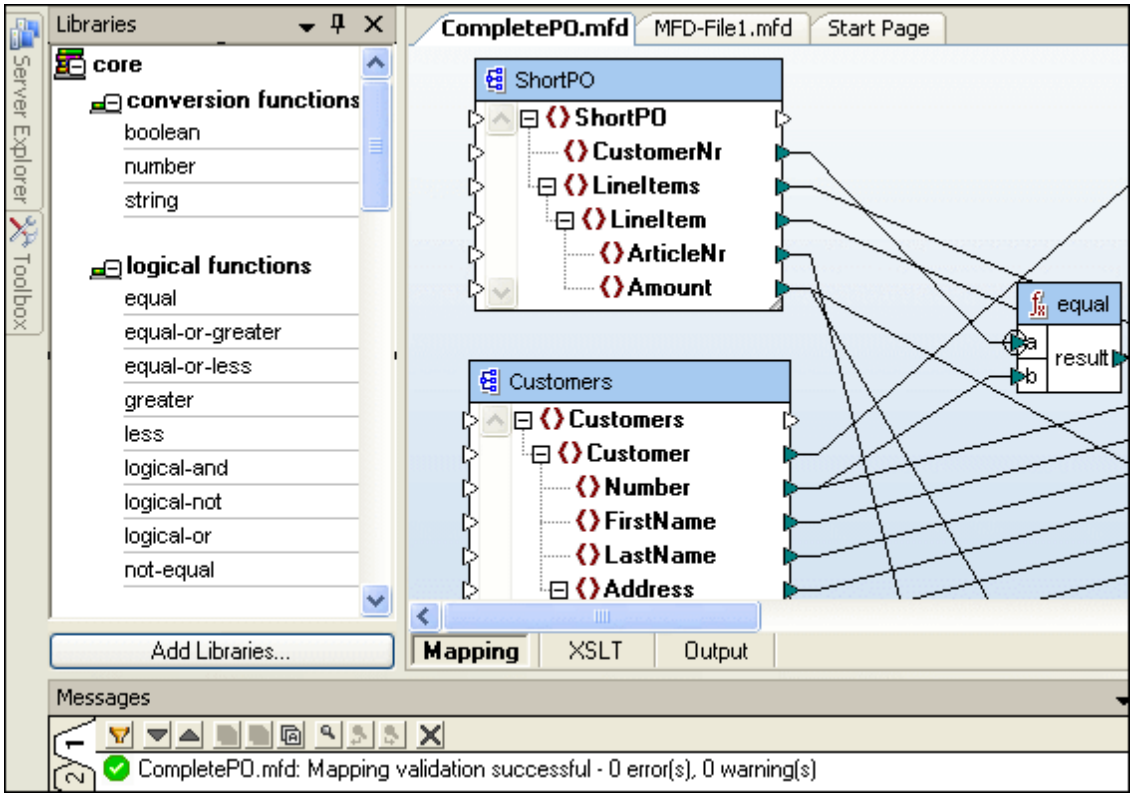
1. Select the menu item **View | MapForce | Library Window**.



2. Dock the floating window at the position you want to use it e.g. left border.

To open a supplied sample file:

1. Select the menu option **File | Open,** navigate to the **...\MapForceExamples** folder and open a MapForce file.
   CompletePO.mfd is shown in the screenshot below.



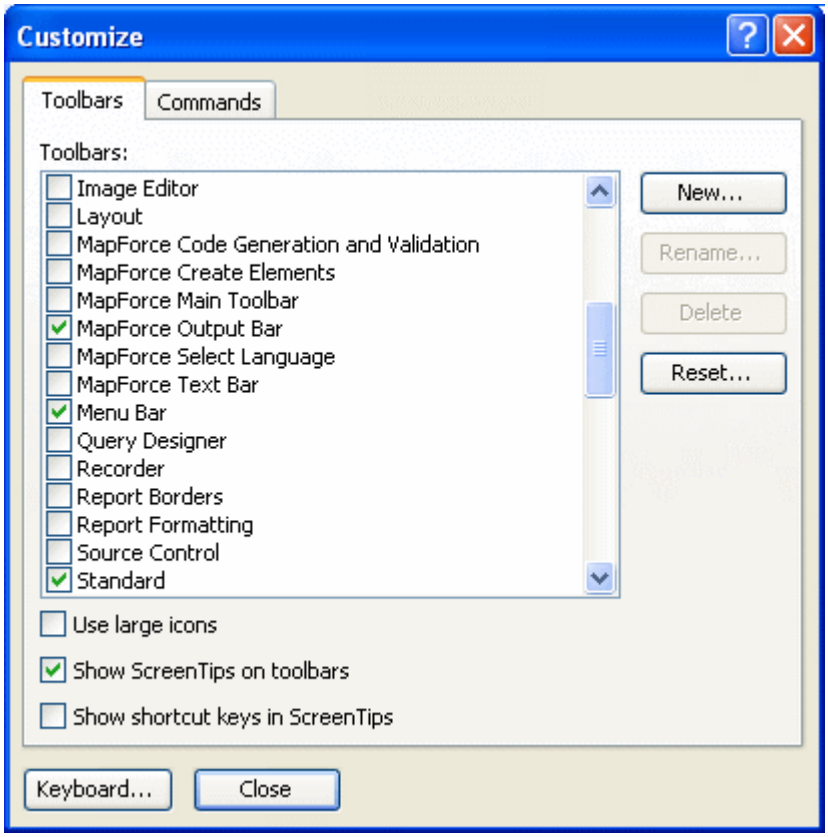
20.2 Differences between Visual Studio and standalone versions

The Enterprise and Professional MapForce plug-ins are integrated into all versions of MS Visual Studio in the same way. Please note there is no support for MapForce projects in the Visual Studio version.

Changed functionality in the Visual Studio editions:

Menu **Edit,** Undo and Redo
The Undo and Redo commands affect all actions (copy, paste, etc.) made in the development environment, including all actions in MapForce.

Menu **Tools | Customize | Toolbar**, **Commands**.



These tabs contain both Visual Studio and MapForce commands.

Menu **View**
The View menu contains the submenu MapForce, which allows you to enable or disable the MapForce tool panes. It also gives access to MapForce view settings.

Menu **Help**
The **Help** menu contains the submenu MapForce Help, which is where you can open the MapForce help. It also contains links to the Altova Support center, Component download area, etc.

Unsupported features of the Visual Studio edition of MapForce

Both the **Project** pane and **Project** menu are not available in these editions. This means that MapForce projects, as well as WSDL projects, cannot be opened in these editions.

Chapter 21

MapForce plug-in for Eclipse

21 MapForce plug-in for Eclipse

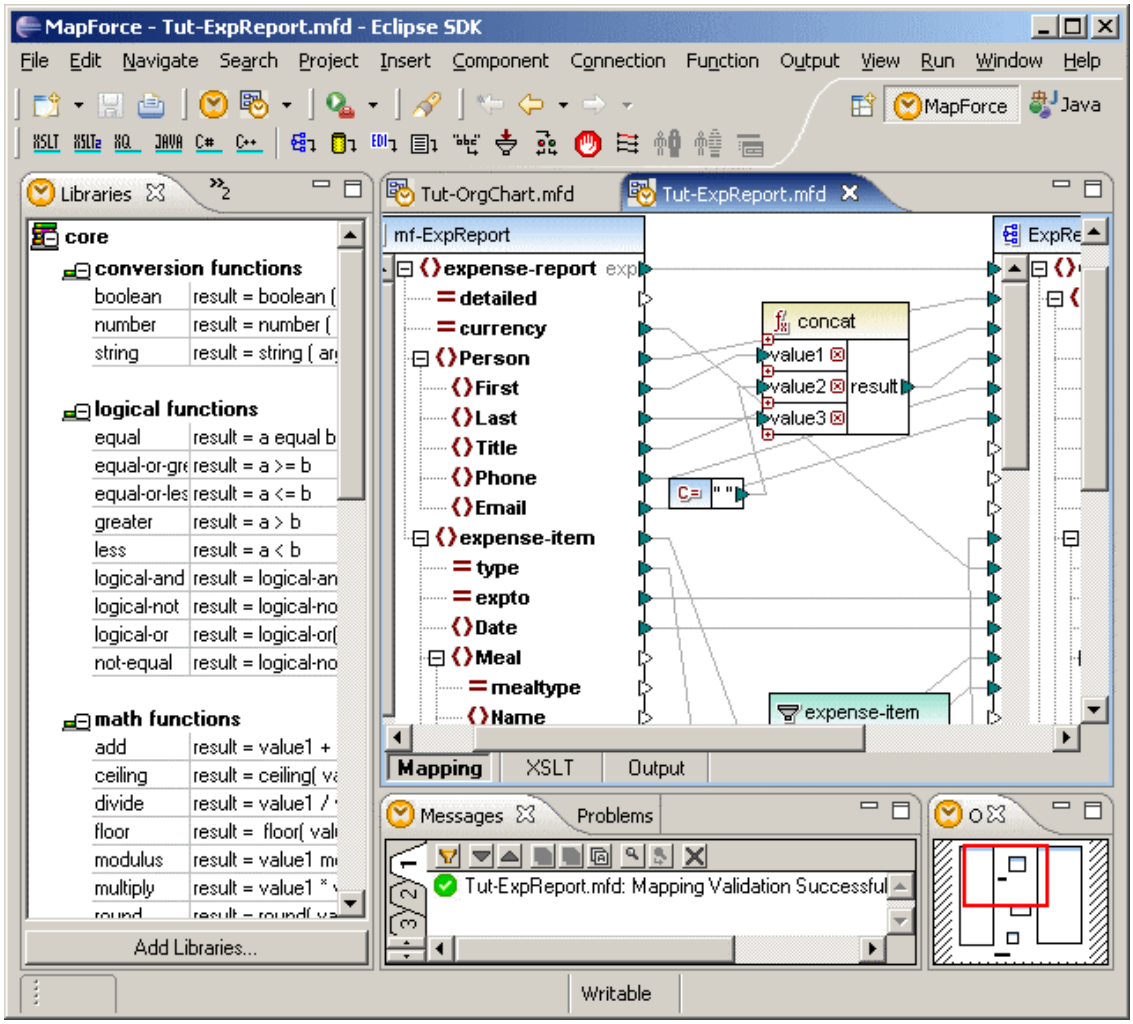
Eclipse is an open source framework that integrates different types of applications delivered in form of plugins. MapForce for the Eclipse Platform, is an Eclipse Plug-in that allows you to access the functionality of a previously installed MapForce Edition from within the Eclipse 3.4 or higher Platform.

Installation Requirements
To successfully install the MapForce Plug-in for Eclipse 3.4 or higher, you need the following:

- The specific MapForce Edition you intend to use: Enterprise, or Professional
- The Eclipse package, as well as
- The appropriate Java Runtime Edition
- Download and run the **MapForce Integration Package for Eclipse**. This package is available on the MapForce download page at www.altova.com.

The MapForce Plug-in for Eclipse supplies the following functionality:

- A fully-featured visual data mapping tool for advanced data integration projects.
- Code generation capability in the Edition specific programming languages.
- MapForce user help under the menu item **Help | MapForce| Table of Contents.**



Java run-time environment (JavaRTE) prerequisites:
The Eclipse plug-in supports Eclipse versions 3.4 and higher, which require a JavaRTE (run-time environment) of version 1.5 or higher.

If the error message shown below occurs when trying to open a document, this indicates that Eclipse is using an older JavaRTE. Eclipse uses the PATH environment variable to find a javaw.exe.

Error:
java.lang.UnsupportedClassVersionError: com/altova/.... (Unsupported major.minor version 49.0)

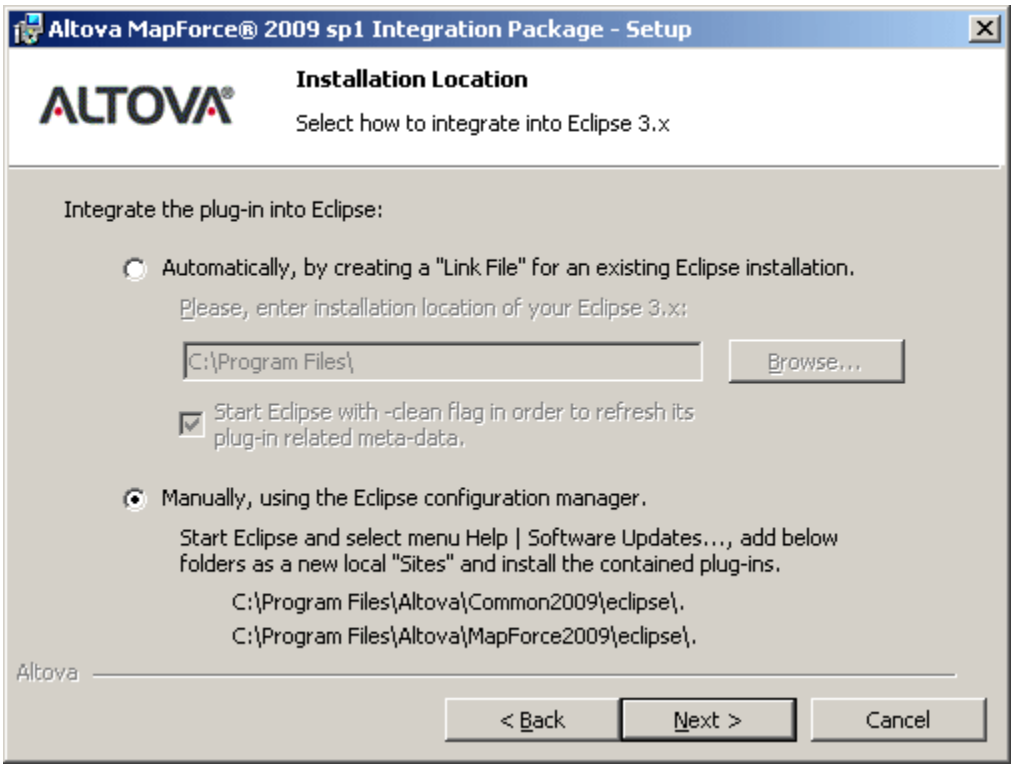
The problem can be solved by either:

- running Eclipse with the command-line parameter **-vm** and supplying the path to a **javaw.exe** of version 1.5 or higher.
- checking the PATH variable for the location of the javaw.exe that gets found first, (if multiple installations of Eclipse exist) and changing it to point to the newer version.

21.1 Manually installing MapForce plugin

Manually installing the MapForce Plug-in for Eclipse:
To install the MapForce Plug-in:

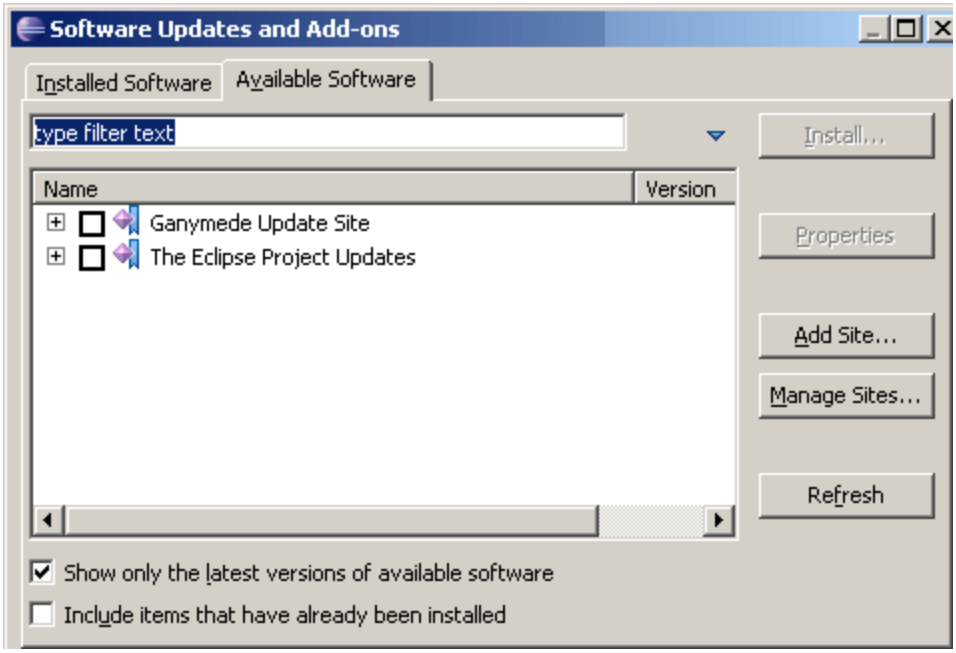
- Download and install the MapForce edition you intend to use from the Download section of the Altova.com website, i.e. Enterprise, Professional or Basic edition.
- Download and install the MapForce Plug-in for Eclipse from the Download section of the Altova.com website. You will be prompted for the installation folder of Eclipse during the installation process.



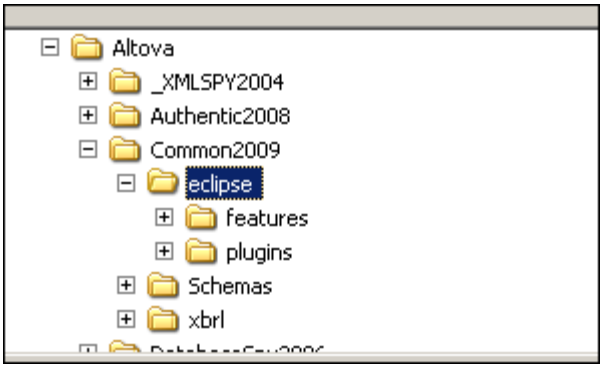
1. Click "Manually, using the Eclipse configuration manager", then click Next to continue.
2. Click Install and then Finish to complete the installation procedure.

Configuring the Eclipse installation to use a **previously** installed MapForce plug-in:

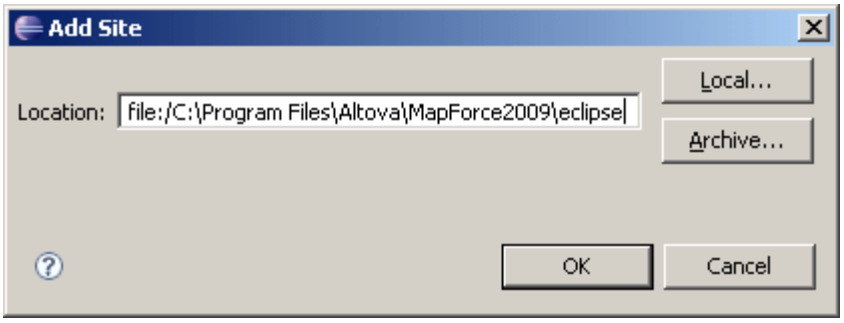
1. Start Eclipse and select the menu option **Help | Software Updates.**



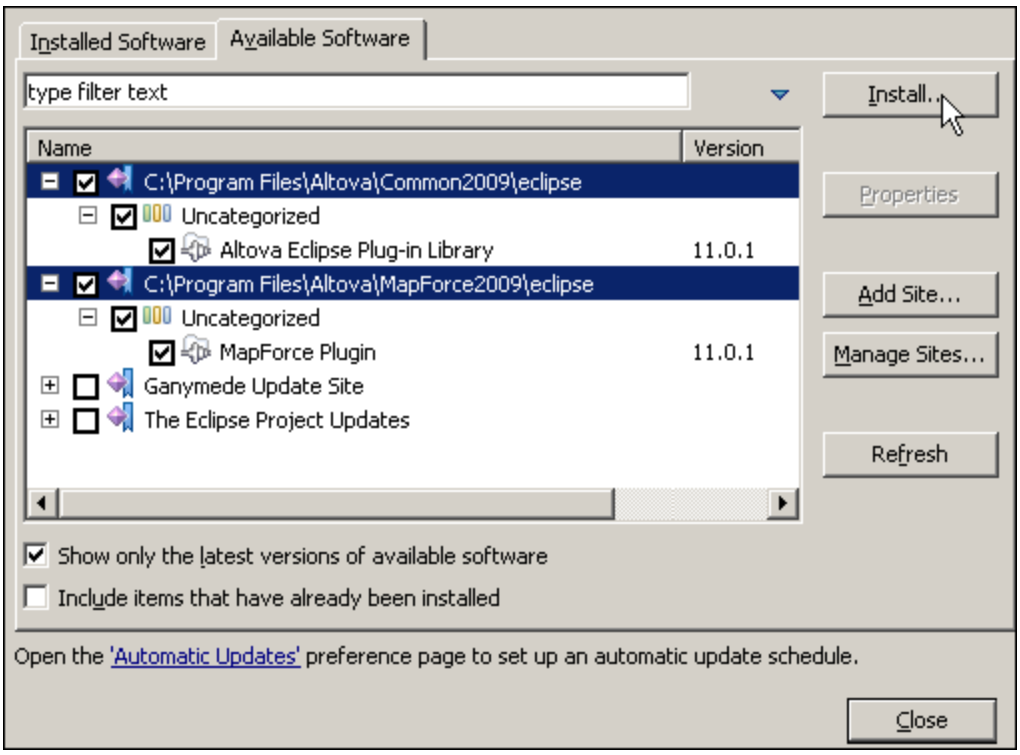
2. Click the **Add Site...** button, then the Local button.
3. Browse to the folder ...**Program Files\Altova\Common2011** and select the eclipse folder and click OK to continue.



4. Click the Add Site button again, click Local and browse to **...\...Program Files\Altova\MapForce2011\eclipse** and click OK to continue.

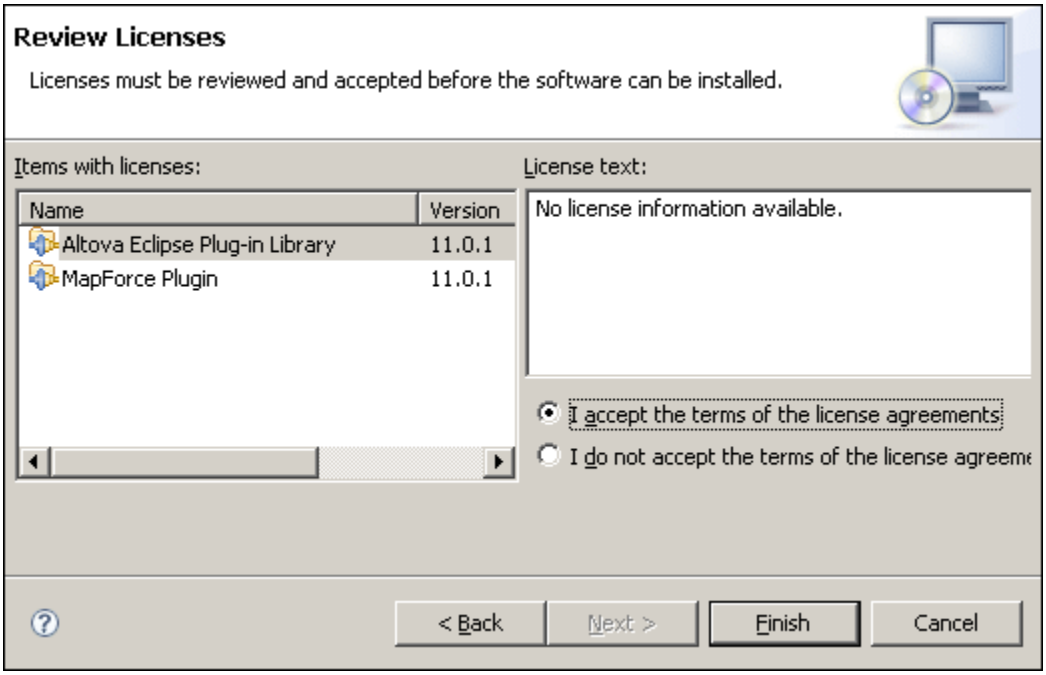


5. Click the top check boxes of each of the path entries to select the plugins then click **Install** to continue.



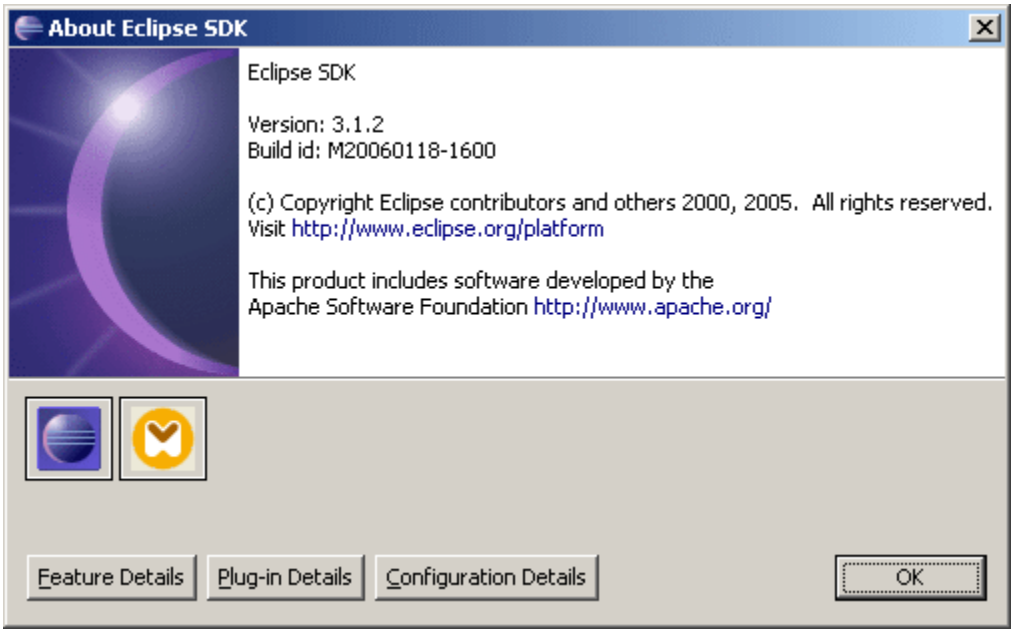
An Installation review dialog box allowing you to confirm that the checked items will be installed opens.

6. Click Next to continue.
7. The Review Licenses dialog box opens, click I accept the terms... then click Finish to complete the installation.



To check the currently installed version:

1. Select the menu option **Help | About Eclipse SDK.**



2. Click the MapForce icon, to view the version specifics.

Note:

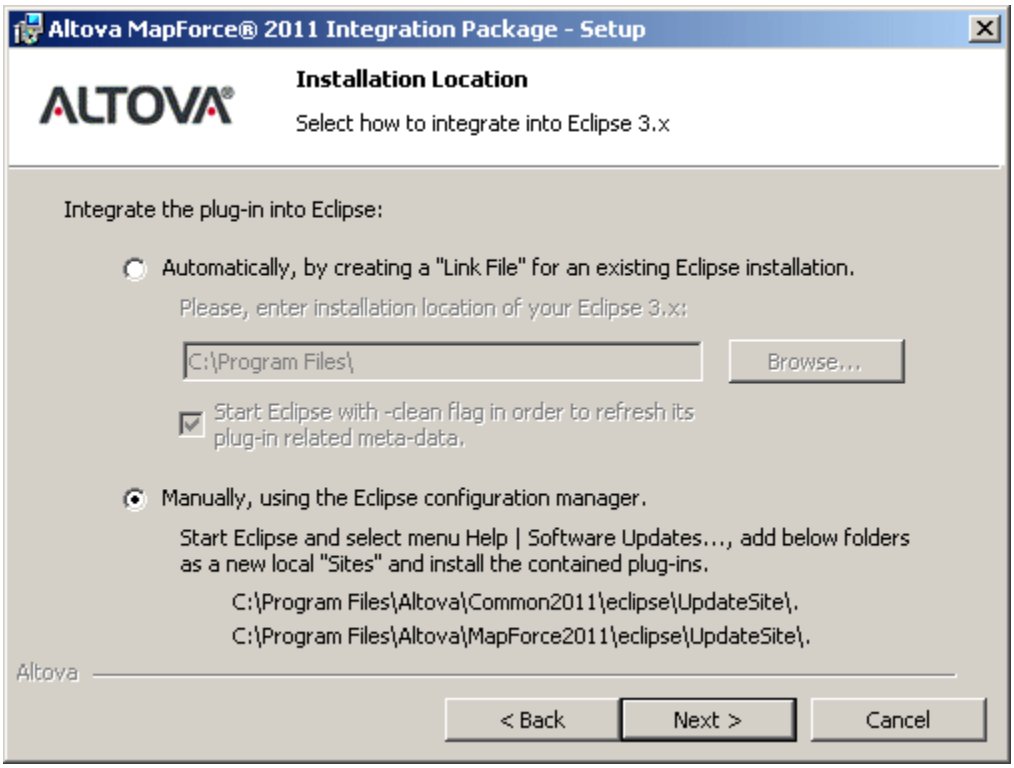
If the pluging does not seem to be installed, or toolbar buttons are missing, start Eclipse once with the "clean" parameter.

E.g. **eclipse.exe -clean**.

21.2 Manually installing MapForce plugin - eclipse 3.6

Manually installing the MapForce Plug-in for Eclipse:
To install the MapForce Plug-in:

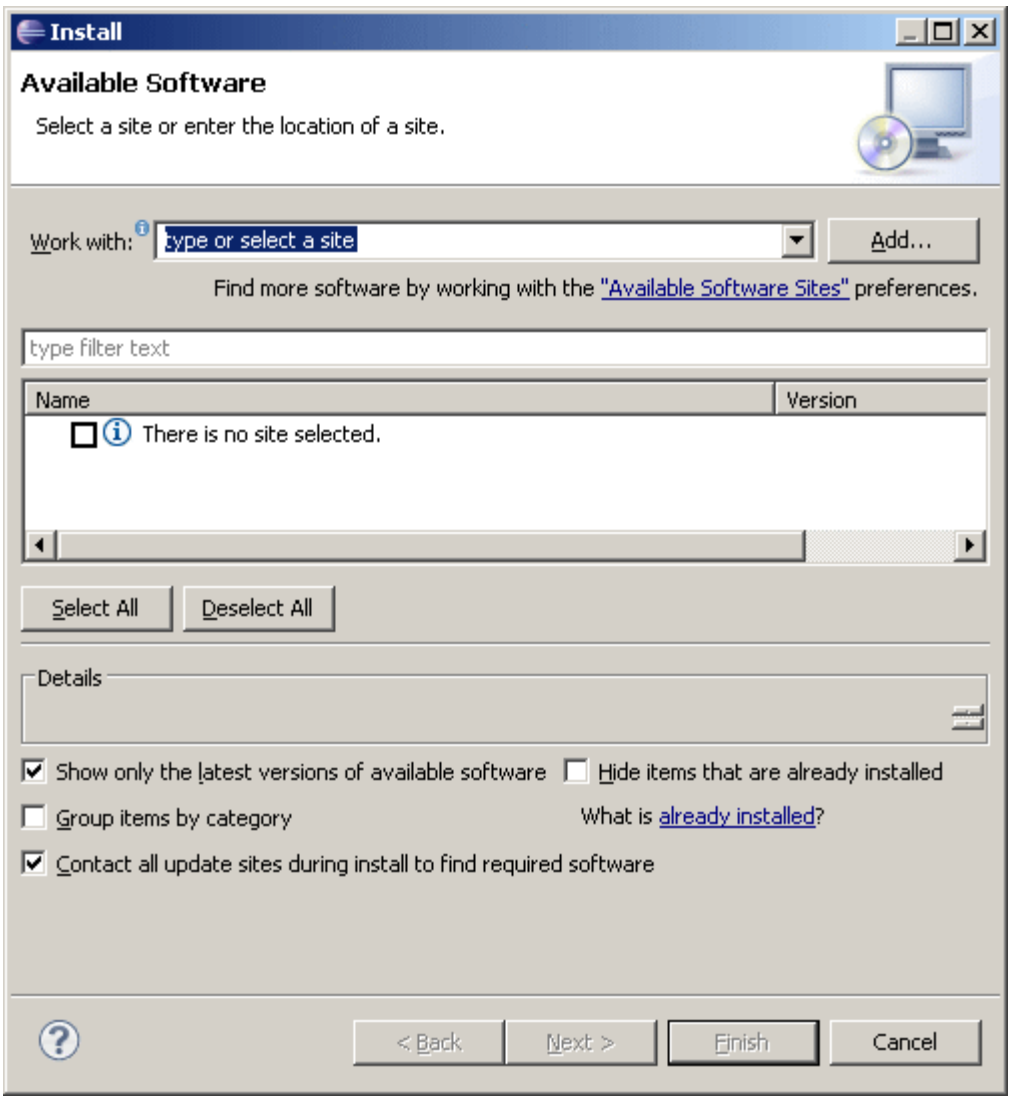
- Download and install the MapForce edition you intend to use from the Download section of the Altova.com website, i.e. Enterprise, Professional or Basic edition.
- Download and install the MapForce Plug-in for Eclipse from the Download section of the Altova.com website. You will be prompted for the installation folder of Eclipse during the installation process.



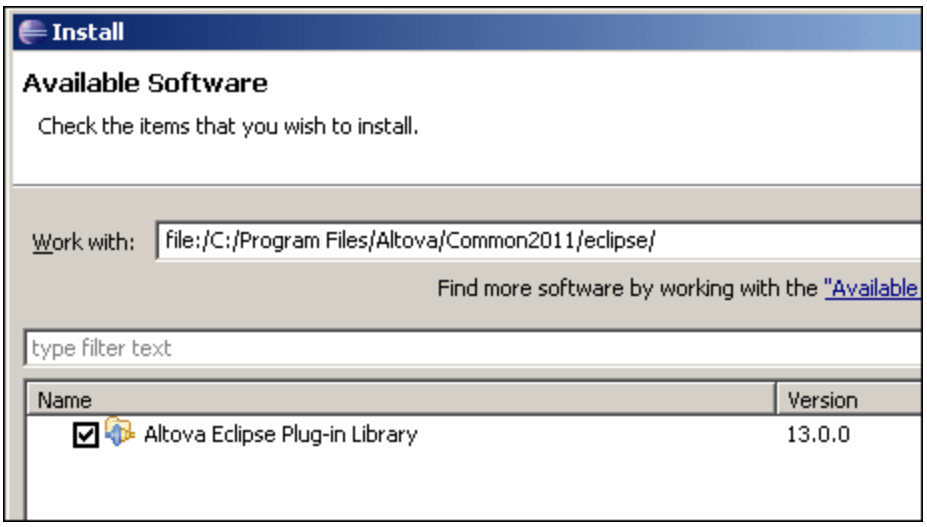
1. Click "Manually, using the Eclipse configuration manager", then click Next to continue.
2. Click Install and then Finish to complete the installation procedure.

Configuring the Eclipse installation to use a **previously** installed MapForce plug-in:

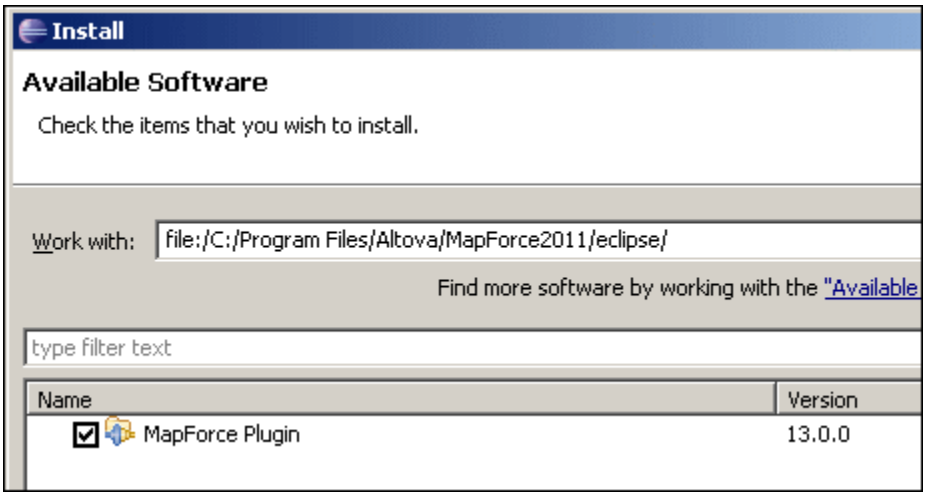
1. Start Eclipse and select the menu option **Help | Install new Software.**



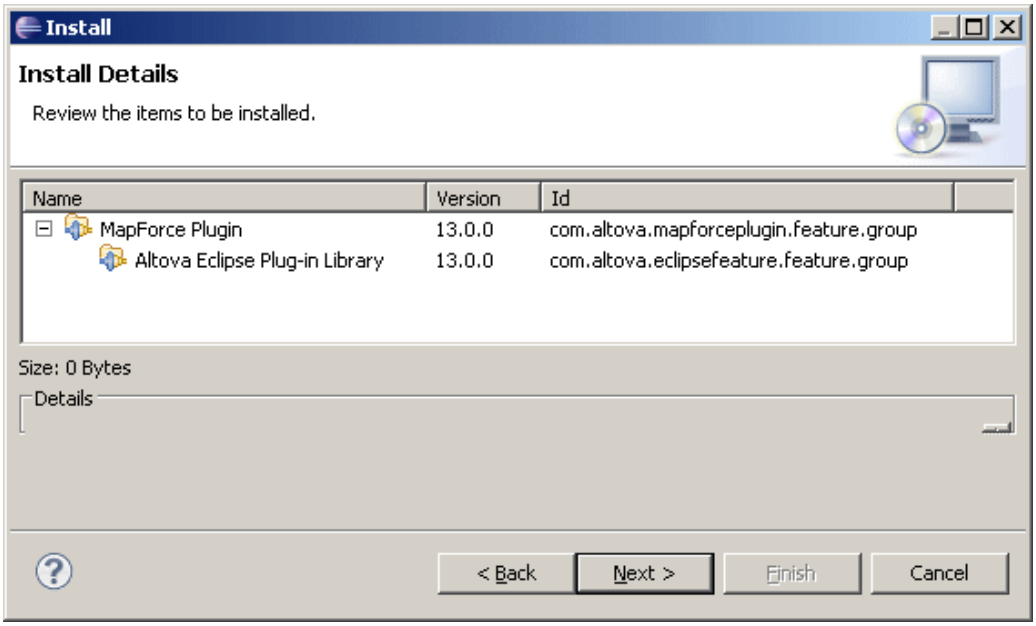
2. Click the **Add...** button, then the Local... button in the following dialog box.
3. Browse to the folder ...**Program Files\Altova\Common2011,** select the **eclipse** folder and click OK to continue.



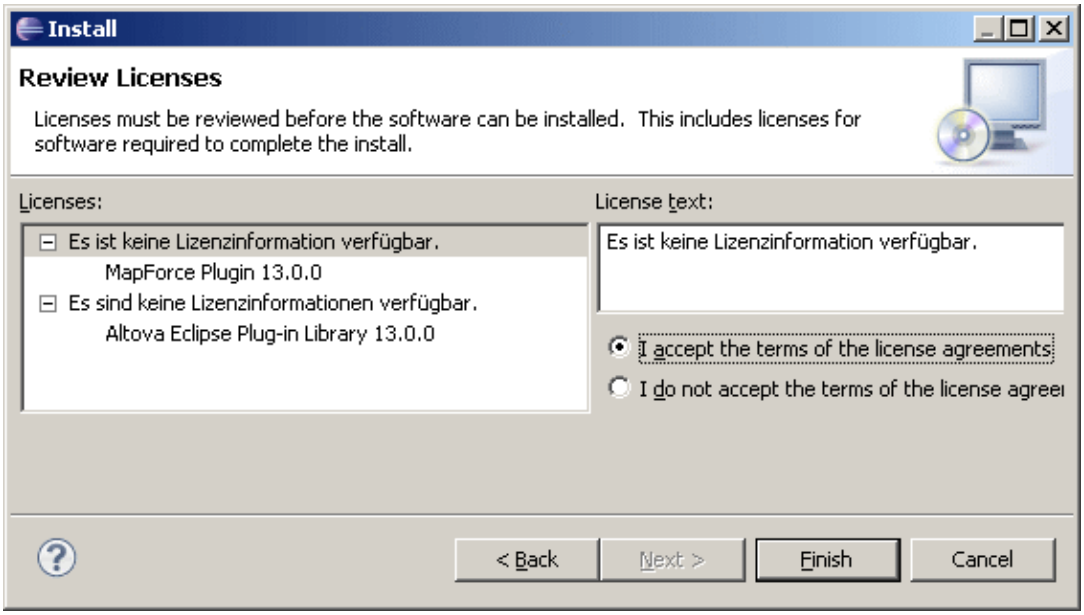
4. Click the "Altova Eclipse Plug-in Library" check box to select it.
5. Click the Add... button again, click Local and browse to **...Program Files\Altova\MapForce2011\eclipse** and click OK to continue.



5. Click the "MapForce Plugin" check box to select it.
6. Click the Next... button to continue.
   An Installation review dialog box opens and allows you to confirm the checked items to be installed.

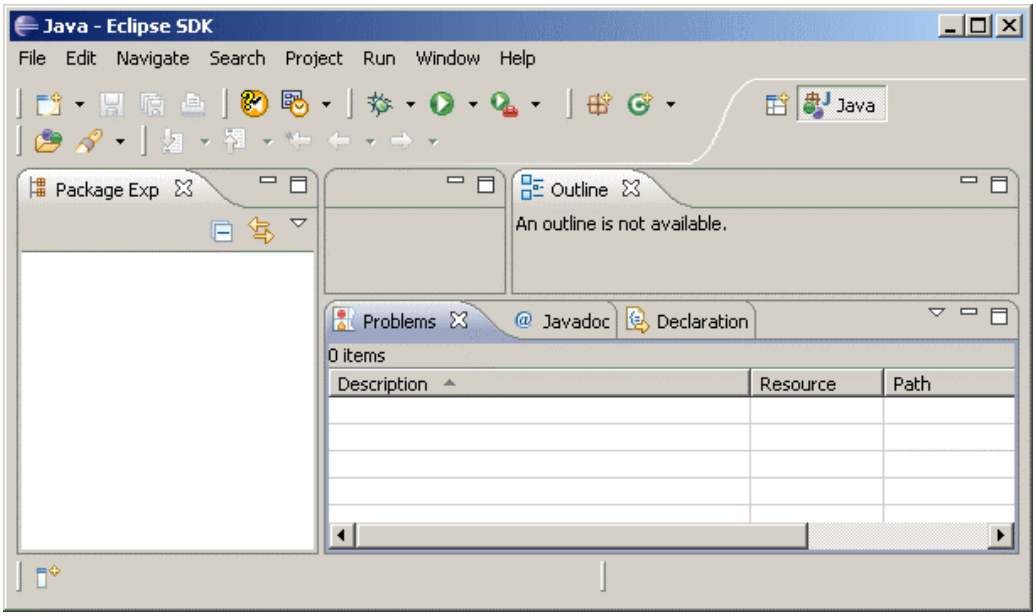


6. Click Next to continue.
7. The Review Licenses dialog box opens, click the "I accept the terms..." radio button then click Finish to complete the installation.



An "Unsigned content..." warning dialog box appears, click OK to install the plugin.

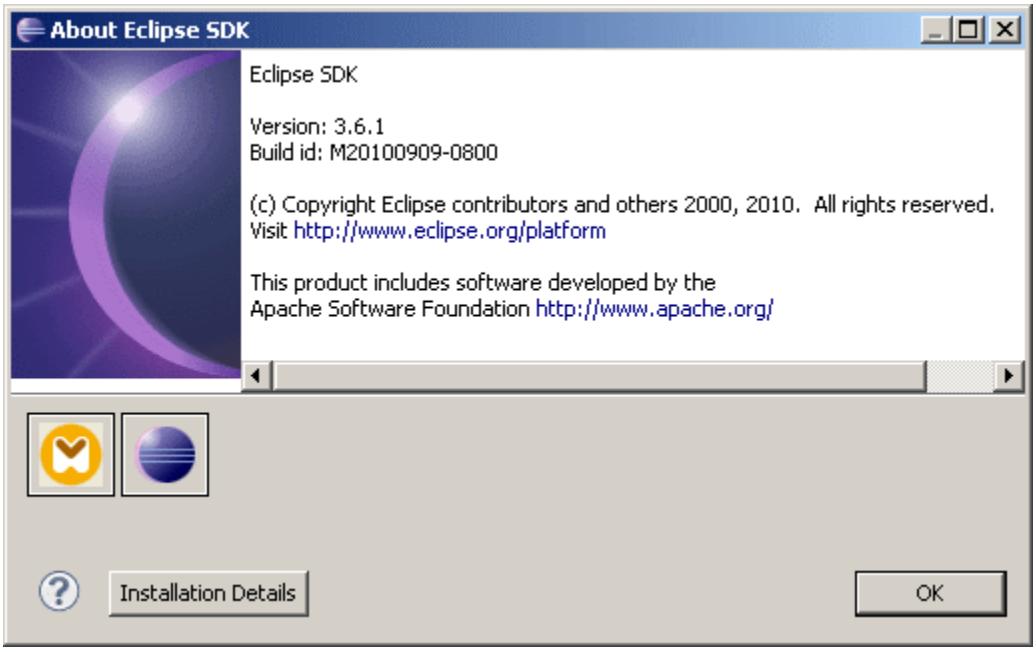
8. Click Restart eclipse to make sure that the changes take effect.



Two MapForce icons are now visible in the main toolbar.

To check the currently installed version:

1. Select the menu option **Help | About Eclipse SDK.**



2. Click the MapForce icon, to view the version specifics.

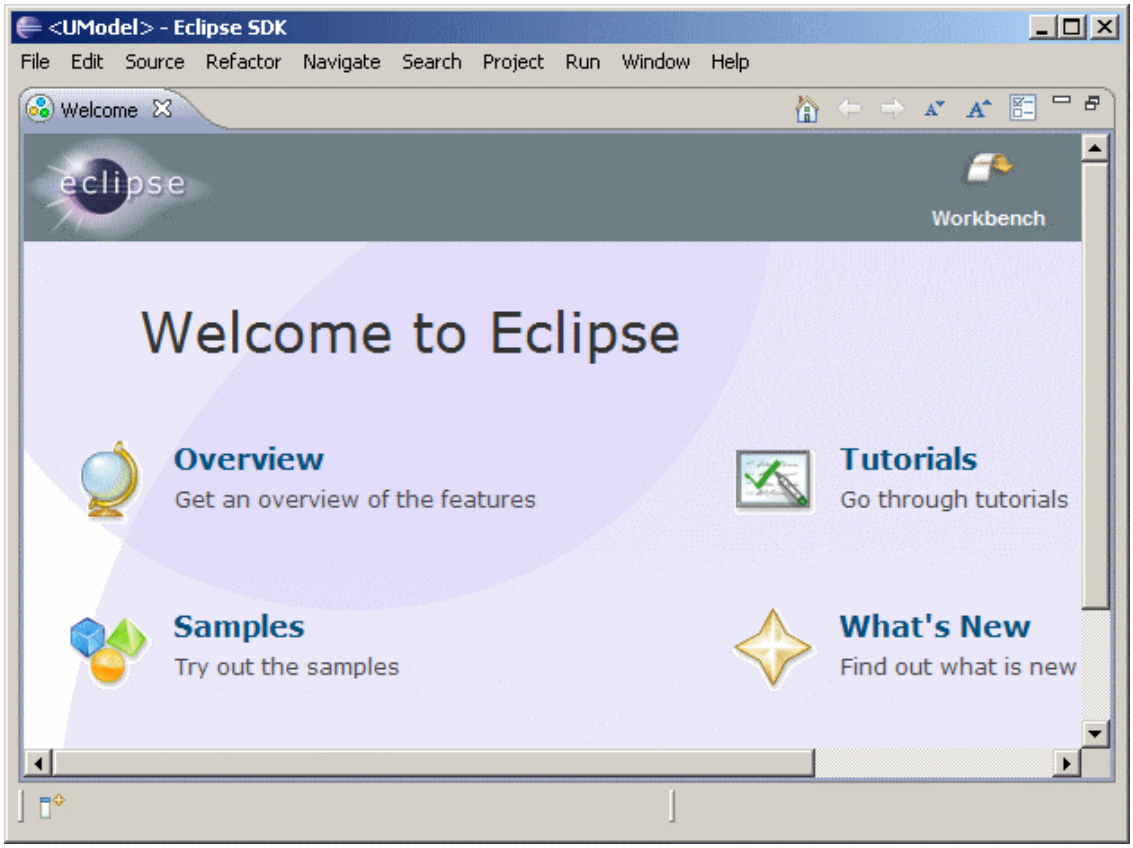
Note:

If the pluging does not seem to be installed, or toolbar buttons are missing, start Eclipse once with the "clean" parameter.

E.g. **eclipse.exe -clean**.

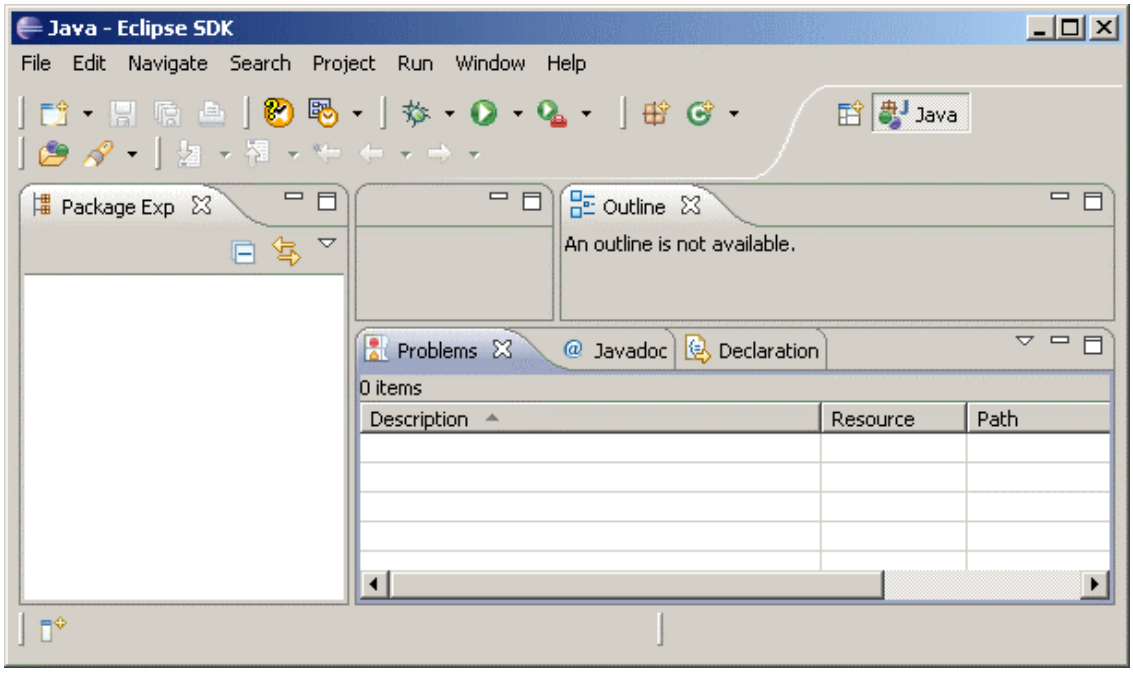
21.3 Starting Eclipse and using MapForce plugin

Place the cursor over the arrow symbol (top right), and click to open the workbench. This opens an empty MapForce window in Eclipse.



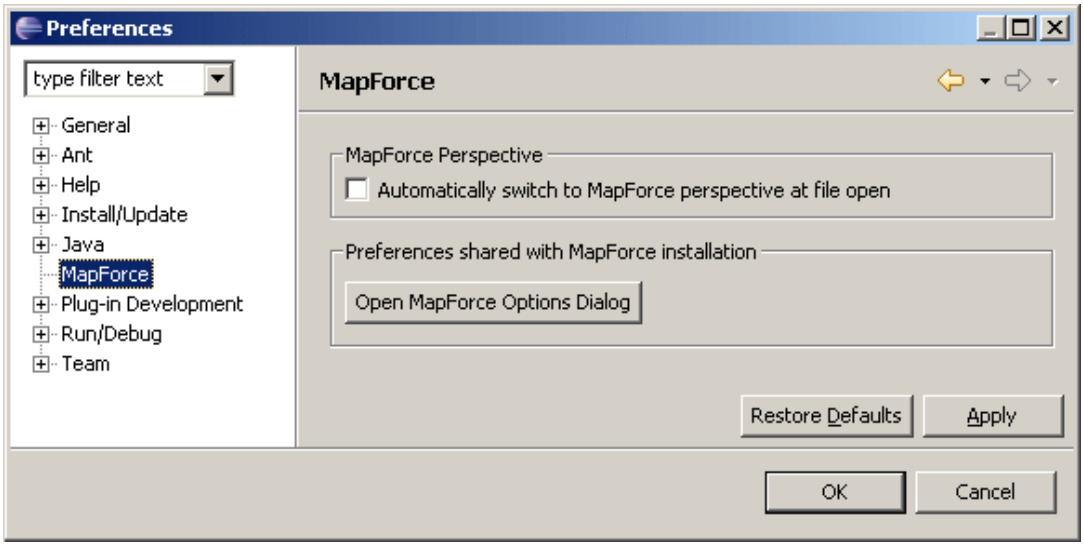
If you do not see this start screen, you can select **Help | Welcome** (in the eclipse IDE) at any time to open it.

Starting Eclipse and Using MapForce Plug-in:
Having used the MapForce for Eclipse installer, you are presented with an empty Eclipse environment.



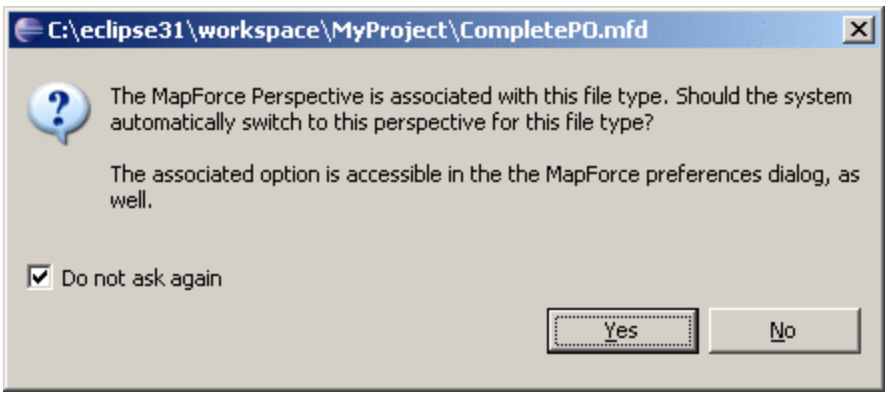
MapForce properties:

1. Select the menu option **Window | Preferences,** and click the MapForce entry.
2. Activate the available check box, to switch to the MapForce perspective when opening a file.



Clicking the "Open MapForce Options Dialog" button, opens the Options dialog which allows you to define the specific MapForce settings, i.e. Libraries, Code generation settings etc.

Double clicking a MapForce mapping file (*.mfd) initially opens a message box stating that a MapForce perspective is associated with this type of file, and prompts if you want Eclipse to automatically switch to the MapForce perspective in the future. These settings can be changed later through the **Window | Preferences | MapForce | MapForce Perspective** option.

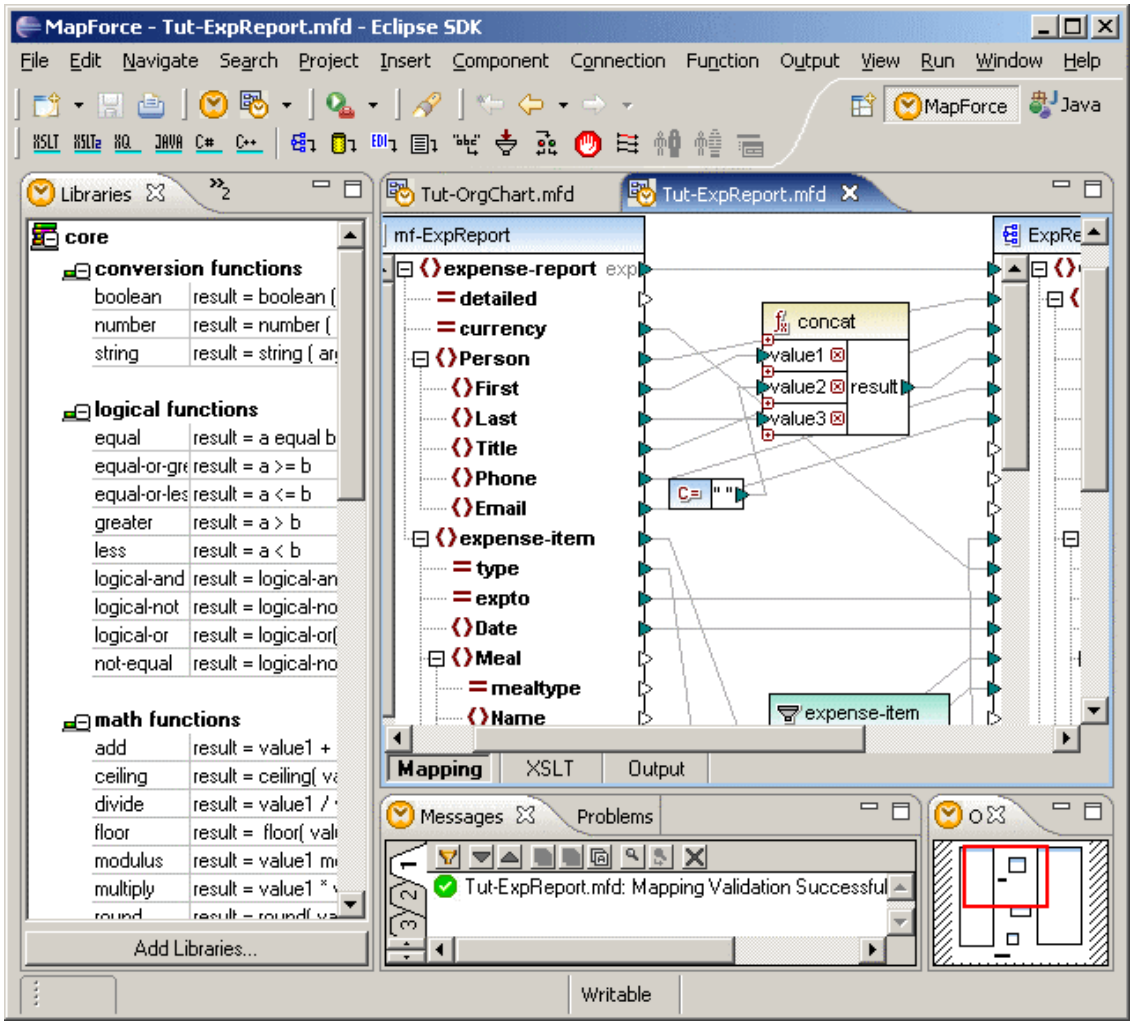


21.4 MapForce / Editor, View and Perspectives

The MapForce perspective can be automatically set if you activate the "Automatically switch to MapForce perspective at file open" in the **Window | Preferences** dialog box. You can also use the option described below to enable the perspective.

To enable the MapForce perspective in Eclipse:

- Select the menu option **Window | Open perspective | Other | MapForce.** (The screenshot below shows the Tut-ExpReport.mfd mapping.)



The individual MapForce tabs are now visible in the Eclipse Environment:

- **Libraries** tab at left, allows you to select predefined or user-defined functions.
- **Messages** tab displays validation messages, errors and warnings
- **Overview** tab displays an iconized view of the mapping file.

The editor pane is where you design your mappings and preview their output, and consists of the following tabs:

Mapping, which displays the graphical mapping design.

XSLT, which displays the generated XSLT code. The name of this tab reflects the option you have selected under Output | XSLT 1.0, XSLT2, or XQuery.

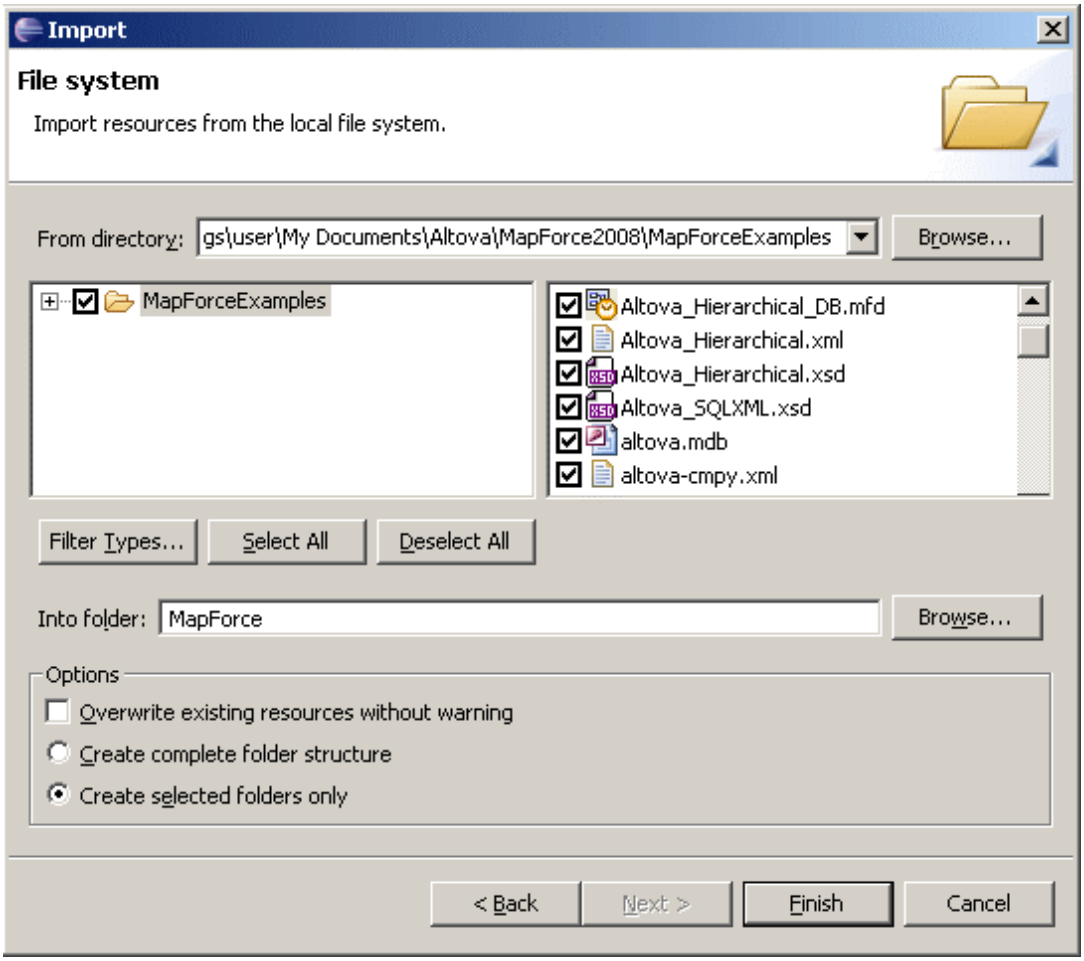
Output, which displays the Mapping output, in this case the XML data.

21.5 Importing MapForce examples folder into Navigator

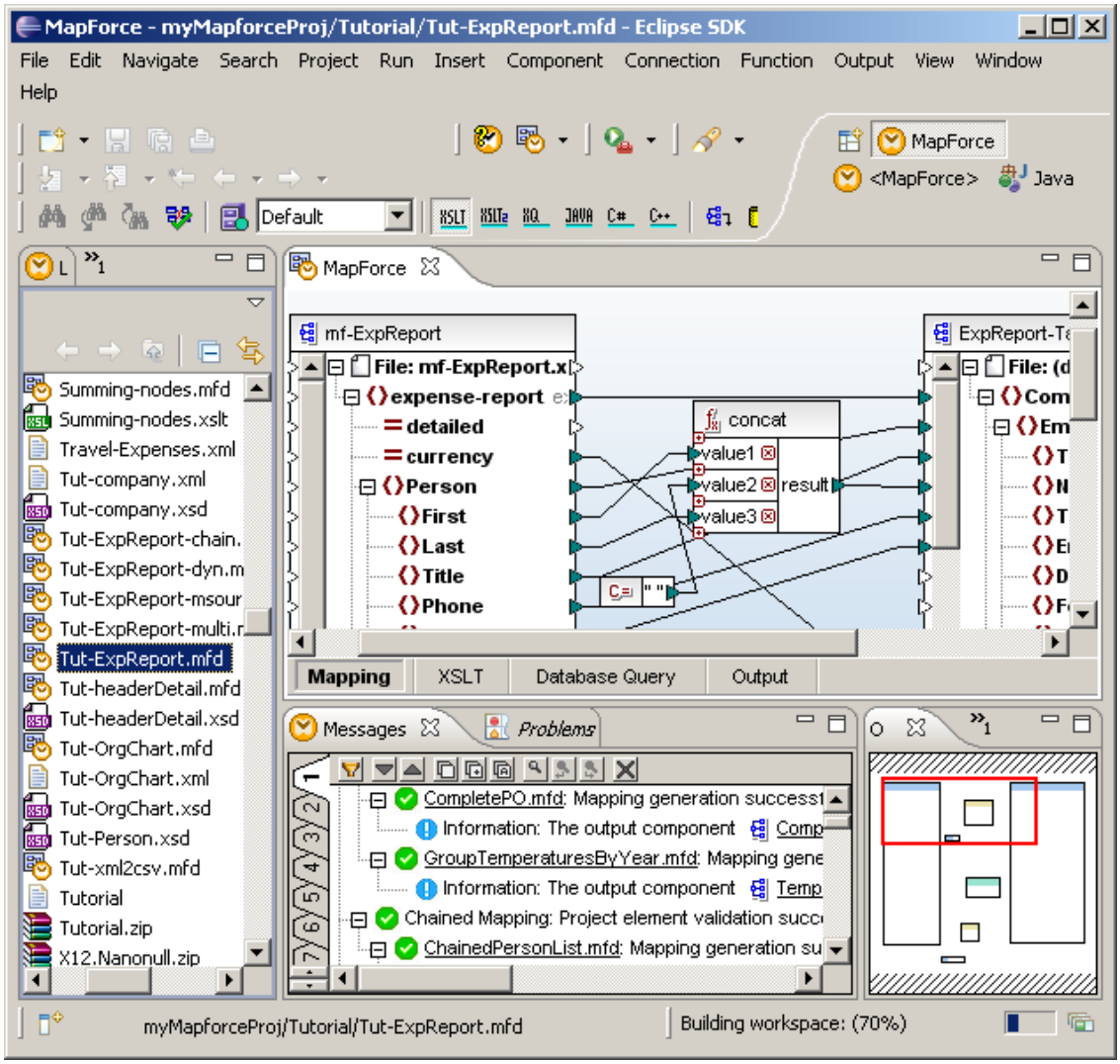
To Import the MapForce Examples folder into the Navigator:

Create an eclipse project using File | New | Project | MapForce/Eclipse project, if a project does not already exist.

1. Right-click the project name in the **Navigator** tab and click **Import**.
2. Select "General | File system", then click **Next**.
3. Click the **Browse** button to the right of the "From directory:" text box, and select the MapForceExamples directory in your MapForce folder.
4. Click the **MapForceExamples** check box..
   This selects all files in the various subdirectories in the window at right.



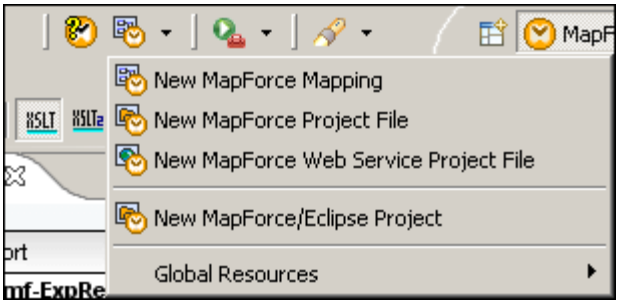
5. If not automatically supplied, click the **Browse** button, next to the "Into folder:" text box, to select the target folder, then click **Finish**.
   The selected folder structure and files will be copied into the Eclipse workspace.
6. Double-click a file in Navigator to open it.



21.6 Creating new MapForce files (mapping and project file)

To create a new MapForce mapping or project files:

- Click the New MapForce... combo box and select the required option.



- New MapForce mapping, creates a single mapping file.
- New MapForce Project File, creates a MapForce project that can combine multiple mappings into one code-generation unit.
- New MapForce Web Service Project File, creates a Web Service project.
- New MapForce Project, creates a new MapForce/Eclipse project, adding the folder to the Navigator window. New MapForce/Eclipse projects are Eclipse projects with a MapForce builder assigned to them. See Using MapForce Eclipse projects for automatic build for details.

21.7 MapForce code generation

Build Integration
MapForce mappings can be contained in any eclipse project. Generation of mapping code can be triggered manually by selecting one of the **'Generate Code...**' menu entries for the mapping or MapForce project file. Full integration into the Eclipse auto-build process is achieved by assigning the MapForce builder to an Eclipse project.

For manual code generation see Build mapping code manually

For automatic generation of mapping code please see Using MapForce Eclipse projects for automatic build and Adding MapForce nature to existing Eclipse Project.

21.7.1 Build mapping code manually

To manually build mapping code for a single mapping:

1. Open, or select the mapping, and select **File | Generate Code in**, or **File | Generate Code in Selected Language.**
   You are prompted for a target folder for the generated code.
2. Select the folder and click OK to start code generation.
   Any errors or warnings are displayed in the MapForce Messages tab.

To manually build mapping code for multiple mappings combined into a MapForce **project:**

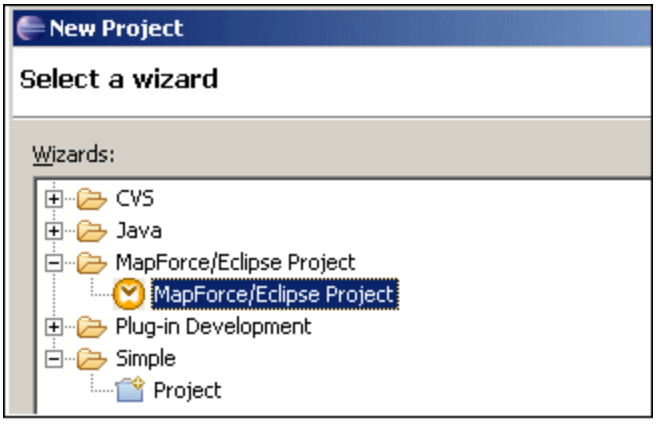
1. Open, or select the MapForce project file.
2. Select the root node or any other node in the project document.
3. Select **Generate Code**, or **Generate Code in** from the right mouse-button menu**.**
   The target folder for the generated code is determined by the properties of the selected node or properties of its parents.
4. Any errors or warnings are displayed in the MapForce Messages tab.

21.7.2 Using MapForce Eclipse projects for automatic build

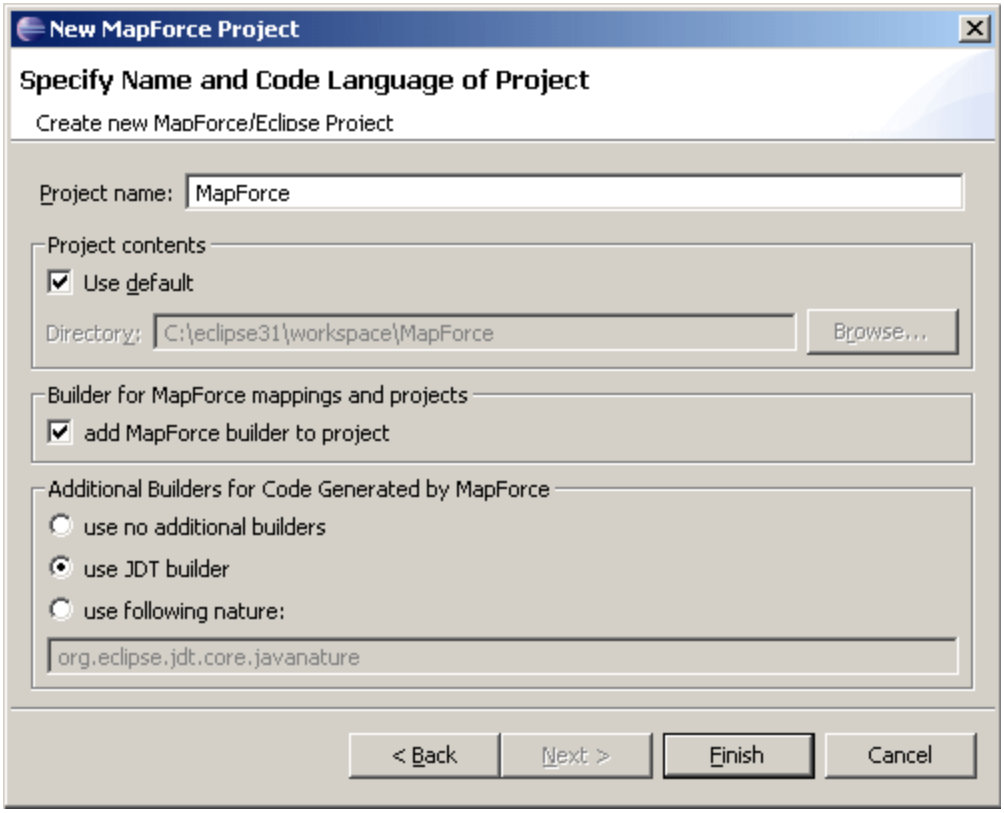
The MapForce plug-in has a built-in project builder. This builder can be identified by the project nature ID `"com.altova.mapforceeclipseplugin.MapForceNature"`. MapForce Eclipse projects have this nature automatically assigned. To use the MapForce project builder in other Eclipse projects see "Adding MapForce nature to existing Eclipse Project" for more information.

To create a new MapForce Eclipse Project:

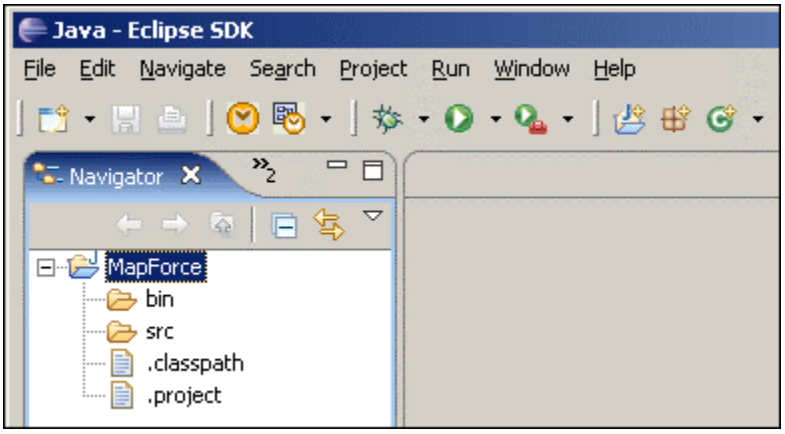
1. Click the Navigator tab to make it active.
2. Right-click in the Navigator window, and select **New | Project.**
3. Expand the MapForce/Eclipse Project entry and select MapForce/Eclipse, then click Next.



4. Enter the project name (e.g. MapForce) and change any of the other project settings to suit your environment, then click Finish. Note the default setting in the "Additional Builders..." group, use JTD builder.



An Eclipse project folder and optionally some more folders and files inside this folder have been created.



You can now create MapForce mappings and MapForce project files inside this Eclipse project, or copy existing ones into it. Whenever a mapping or MapForce project file changes, the corresponding mapping code will be generated automatically. Code generation errors and warnings will be shown in the MapForce view called **Messages** and added to the **Problems** view of Eclipse.

A MapForce Eclipse project is an Eclipse project with the MapForce nature assigned to it, and therefore uses the MapForce builder.

If one or more MapForce project files are present in the Eclipse project, the code generation language and output target folders are determined by the settings in these files.

If a MapForce project file is not present in the Eclipse project:

But the Eclipse project has been assigned the JDT nature:

- Then, the mapping code generation defaults to Java language, and the project's Java source code directory is used as the mapping code output directory.

  Saving a mapping automatically generates the mapping code in Java and compilation of the Java code. Use the Java debug or run command, to test the resulting mapping application.

But the project has **not** been assigned the JDT nature:

- Then the output target folder is the project folder, and the code generation language defaults to the current setting in the MapForce Options.

To activate the Automatic Build process:

1. Make sure that the menu option **Project | Build automatically** is checked.

To temporarily deactivate automatic building of MapForce mapping code:

This is only available to Eclipse projects that have added the MapForce nature.

1. Right click the Eclipse project**,** in the Navigator pane.
2. Select **Properties** from the context menu.
3. Click the "Builders" entry in the left pane of the project properties dialog.
4. Un-check the **MapForce builder** check box in the right pane.
   Modifications to any mapping files or MapForce project files in this Eclipse project, will now no longer trigger automatic generation of mapping code.

21.7.3 Adding MapForce nature to existing Eclipse Project

Applying the MapForce Nature to Existing Projects:
Add the following text to the **natures** section of the **.project** file in the Eclipse project (e.g. in the c:\eclipse33\workspace\MapForce\ folder):

<nature>com.altova.mapforceeclipseplugin.MapForceNature</nature>

```
<natures>
 <nature>org.eclipse.jdt.core.javanature</nature>
 <nature>com.altova.mapforceeclipseplugin.MapForceNature</nature>
</natures>
```

Any MapForce project files and mappings contained in this project will now participate in the automatic build process. For MapForce specific details see Using MapForce Eclipse projects for automatic build.

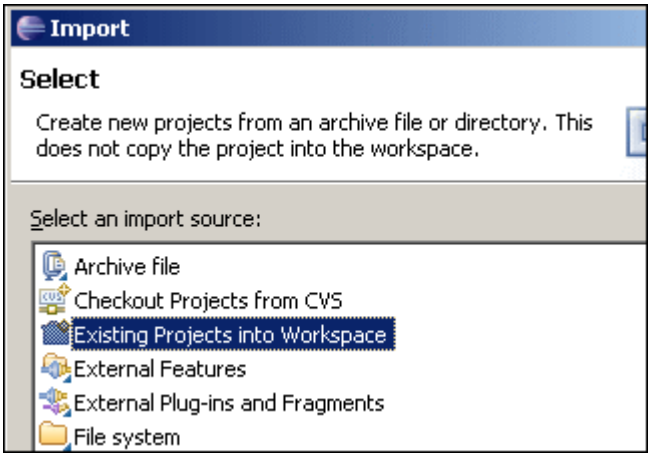
21.8 Extending MapForce plug-in

MapForce plug-in 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 MapForceAPI.

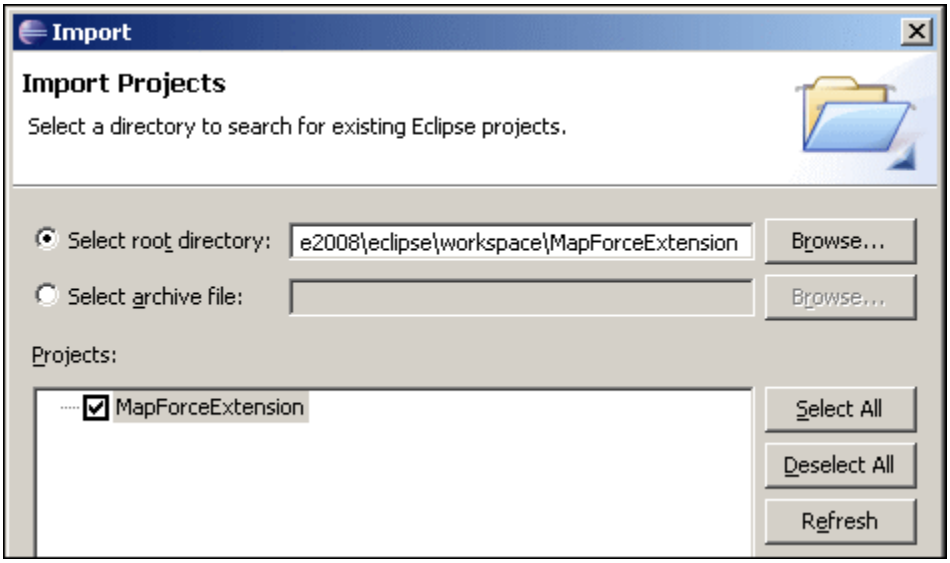
Your MapForce Eclipse installation package contains a simple example of a plug-in that uses this extension point. It checks for any file open events of any new MapForce mappings, and sets the zoom level of the mapping view to 70%.

Installing the Sample extension plug-in:
MapForce plug-in requires the JDT (Java Development Tools) plug-in to be installed.

1. Start Eclipse.
2. Right click in **Navigator** or PackageExplorer, and select the menu item **Import**.
3. Select "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\MapForce2011\eclipse\workspace\MapForceExtension.**.



5. Click Finish.

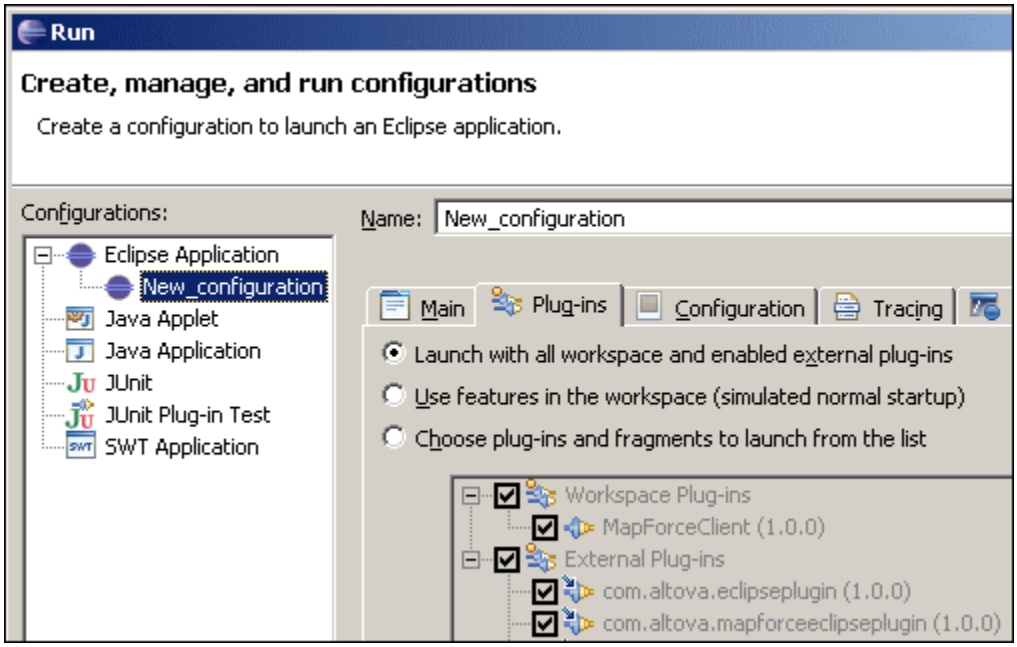
A new project named "MapForceExtension" has been created in your workspace.

Accessing javadoc for the extension point of MapForce plug-in:

1. Open the **index.html** in the docs folder of the plugin installation e.g. c:\Program Files \Altova\MapForce2011\eclipse\plugins\com.altova.mapforceeclipseplugin_11.0.0\docs\

Running the Sample extension plug-in:

1. Switch to the Java perspective.
2. Select the menu option **Run | Run...**
3. Select Eclipse Application and click **New_configuration**.



4. Check that the project MapForceClient is selected in the 'Plug-ins' tab.
5. Click the Run button.
   A new Eclipse Workbench opens.
6. Open any MapForce mapping in the new Workbench. It will now open with a zoom level of 70%.

Chapter 22

User Reference

22 User Reference

The following section lists all the menus and menu options in MapForce, and supplies a short description of each.

22.1 File

New
Creates a new mapping document, or mapping project (mfp)

Open
Opens previously saved mapping (*.mfd) , or mapping project (mfp) files.

Please note:

Opening a mapping that contains features available in a higher-level MapForce edition is not possible.

E.g. A mapping containing Web service features in the Professional version, or database mappings the Basic editions is not possible.

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.

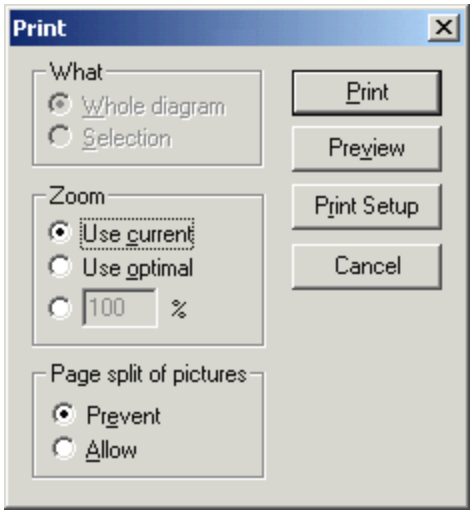
Close
Closes the currently active mapping file. You are asked if you want to save the file before it closes.

Close All
Closes all currently open mapping files. You are asked if you want to save any of the unsaved mapping files.

Save Project
Saves the currently active project.

Close Project
Closes the currently active project.

Print
Opens the Print dialog box, from where you can printout your mapping as hardcopy.



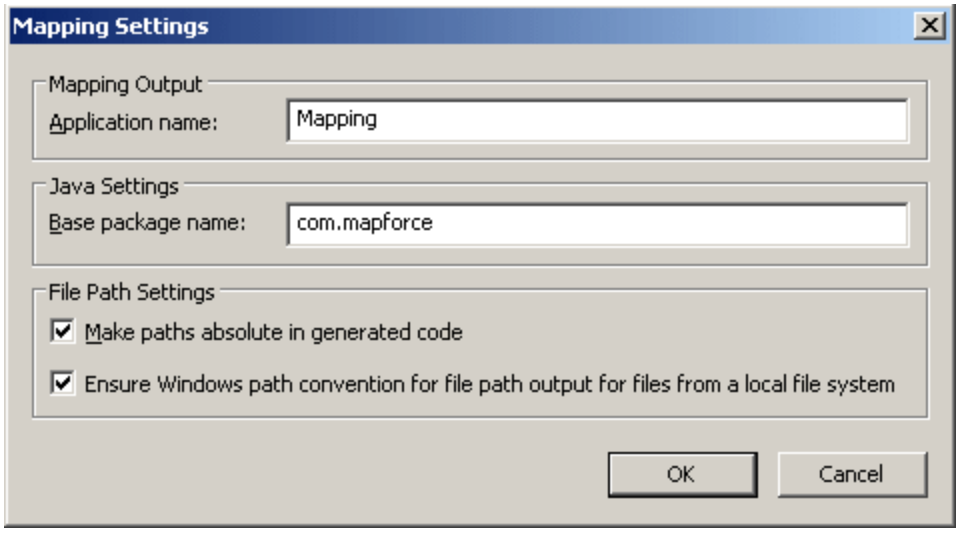
"Use current", retains the currently defined zoom factor of the mapping. "Use optimal" scales the mapping to fit the page size. You can also specify the zoom factor numerically. Component scrollbars are not printed. You can also specify if you want to allow the graphics to be split over several pages or not.

Print Preview
Opens the same Print dialog box with the same settings as described above.

Print Setup
Open the Print Setup dialog box in which you can define the printer you want to use and the paper settings.

Validate Mapping
Validating a Mapping validates that all mappings (connectors) are valid and displays any warnings or errors.
Please see "Validating mappings" for more information.

Mapping settings



The document-specific settings are defined here. They are stored in the *.mfd file.

Mapping Output

Application Name: defines the XSLT1.0/2.0 file name prefix or the Java, C# or C++ application name for the generated transformation files.

Java settings

Base Package Name: defines the base package name for the Java output.

File Path Settings

Make paths absolute in generated code
Ensures compatibility of generated code with mapping files (*.mfd) from versions prior to Version 2010, please see Relative and absolute file paths for more information.

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.

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.

Note: the name of the generated XSLT file(s) is defined in the **Application Name** field of the Mapping Output dialog box. This dialog is opened by selecting **File | Mapping Settings** menu option.

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.

Note: the name of the generated XQuery file(s) is defined in the **Application Name** field of the Mapping Output dialog box. This dialog is opened by selecting **File | Mapping Settings** menu option.

Generate code in | Java
Generate code in | C#
Generate code in | C++
These commands generates 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.

Note: 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 **Application Name** field of the Mapping Output dialog box. This dialog is opened by selecting **File | Mapping Settings** menu option.

The **file name** created by the executed code, is that which appears in the **Output XML instance (for Code Generation)** field of the Component settings dialog box if the target is an XML/Schema document.

Generate documentation
Generates documention of your mapping projects in great detail in various output formats. Please see Documenting mapping projects for more information.

Recent files
Displays a list of the most recently opened files.

Recent projects
Displays a list of the most recently opened projects.

Exit
Exits the application. You are asked if you want to save any unsaved files.

22.2 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 Prevous Shift F3 
Serches 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.

22.3 Insert

XML Schema / File 
Inserts an XML schema file into the mapping tab as data source or target component. You can select XML files with a schema reference, in this case the referenced schema is automatically inserted. If you select an XML schema file, you are prompted if you want to include an XML instance file which supplies the data for the XSLT, XSLT2, XQuery, and Output previews. If you select an XML file without a schema reference, you are prompted if you want to generate a matching XML schema automatically.

Database 
Inserts a database component as data source or target component. The database supplies the schema structure and displays it in a tree view.

Text file 
Inserts a flat file document, i.e. CSV, or a fixed length text file. Both types of file be used as source and target components.

Constant 
Inserts a constant which is a function component that supplies fixed data to an input icon. The data is entered into a dialog box when creating the component. There is only one output icon on a constant function. You can select the following types of data: String, Number and All other.

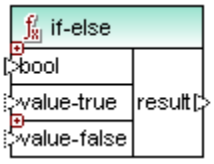
Variable 
Inserts an Intermediate Variable which is equivalent to a regular (non-inline) user-defined function. Variables are structural components, without instance files, and are used to simplify the mapping process. Please see Intermediate variables for more information.

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. Please see the tutorial example on how to use a filter.

SQL-WHERE condition 
Inserts a special filter component for database data that allows you to append any SQL WHERE clause to the queries generated by MapForce.
Please see: SQL WHERE Component / condition for more information.

Value-Map 
Inserts a component that transforms an input value to an output value using a lookup table. The component only has one input and output item. Please see Value-Map - transforming input data for more information.

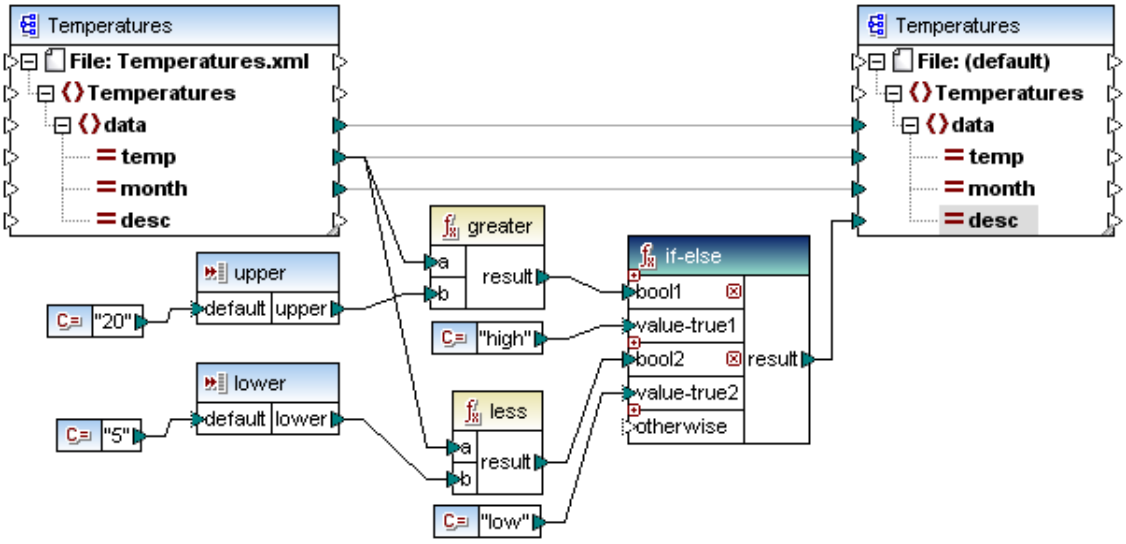
 **IF-Else Condition**
A condition is a component which allows you to pass on different sets of data depending on the outcome of a preset condition. The component header displays the text **if-else**.



- 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.

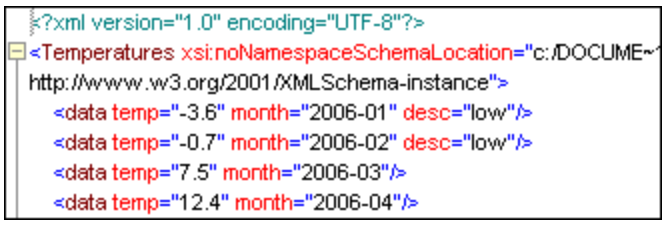
Clicking the "plus" icon inserts or appends a new if-else **pair**, i.e. boolX and value-trueX, while clicking the "x" deletes the parameter pair.



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:



Exception

The exception component allows you to interrupt a mapping process when a specific condition is met. Please see MapForce Exceptions, for more information.

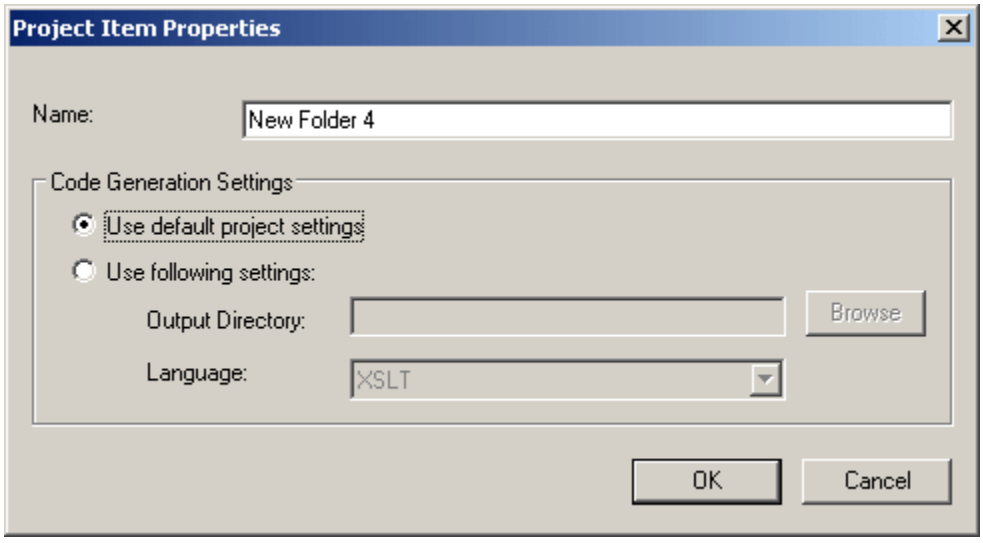
22.4 Project

MapForce supports the Multiple Document Interface and allows you to group your mappings into mapping projects. Project files have a ***.mfp** extension.

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. The default project settings can be applied, or you can define your own by clicking the "Use following settings" radio button.

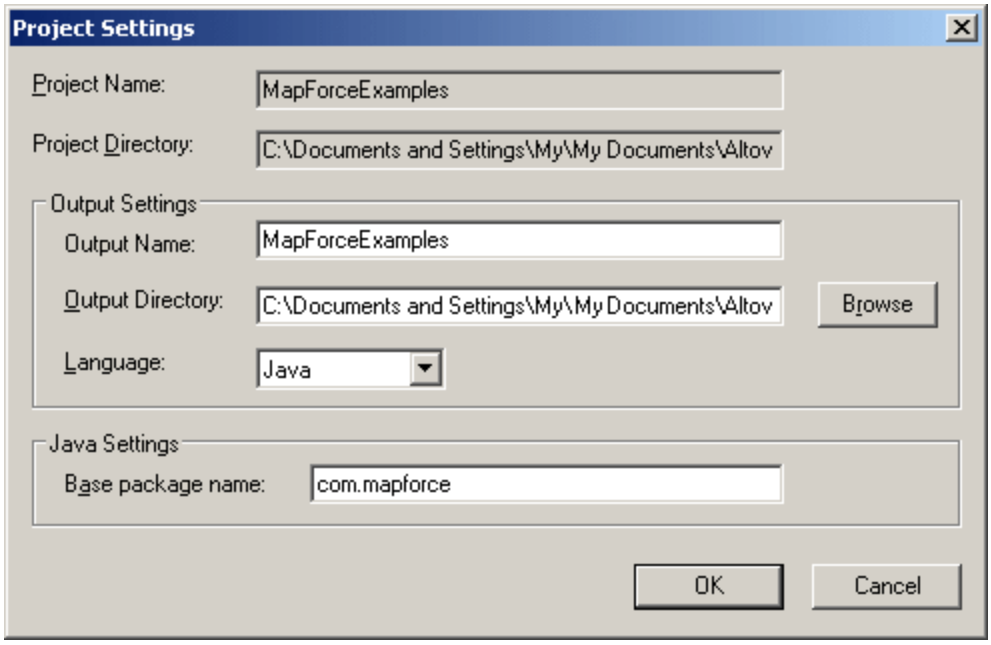


Generate code for entire project:
Generates project code for the entire project currently visible in the Project window. Code is generated in the currently selected default language for all of the mapping files *.mfd in each of the folders.

Generate code in...
Generates project code in the language you select from the flyout menu.

Reload Project
Reloads the currently active project and switches to the Project tab.

Project properties:
Click the project name (e.g. MapforceExamples) in the Project window and select Project | Properties. The dialog box lets you to define the project-wide settings. Clicking on a folder, or file name, in the project window and selecting this command, opens a dialog box for that specific item.

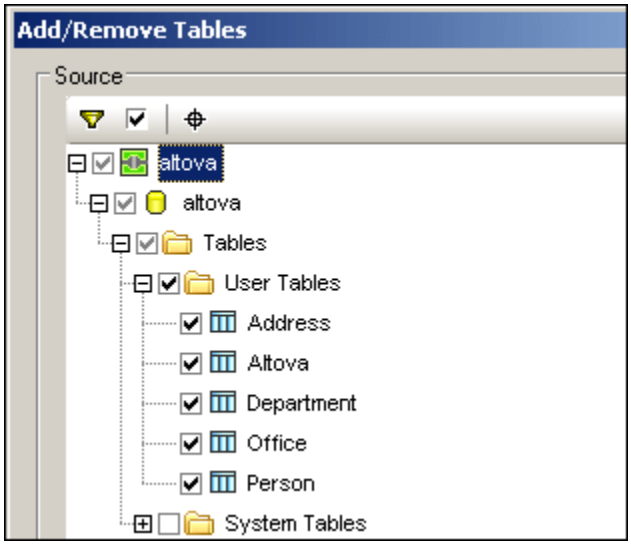


22.5 Component

Change Root Element
Allows you to change the root element of the XML instance document.

Edit Schema Definition in XMLSpy
Selecting this option, having previously clicked an XML-Schema/document, opens the XML Schema file in the Schema view of XMLSpy where you can edit it.

Add/Remove Tables
Allows you to change the number of tables in the database component by selecting/deselecting them in the Database Tables group.



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. Please 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. Please 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. Please see the Duplicating input items section in the tutorial for an example of this.

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. Please see the Duplicating input items section in the tutorial for an example of this.

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. Please see the Duplicating input items section in the tutorial for more information.

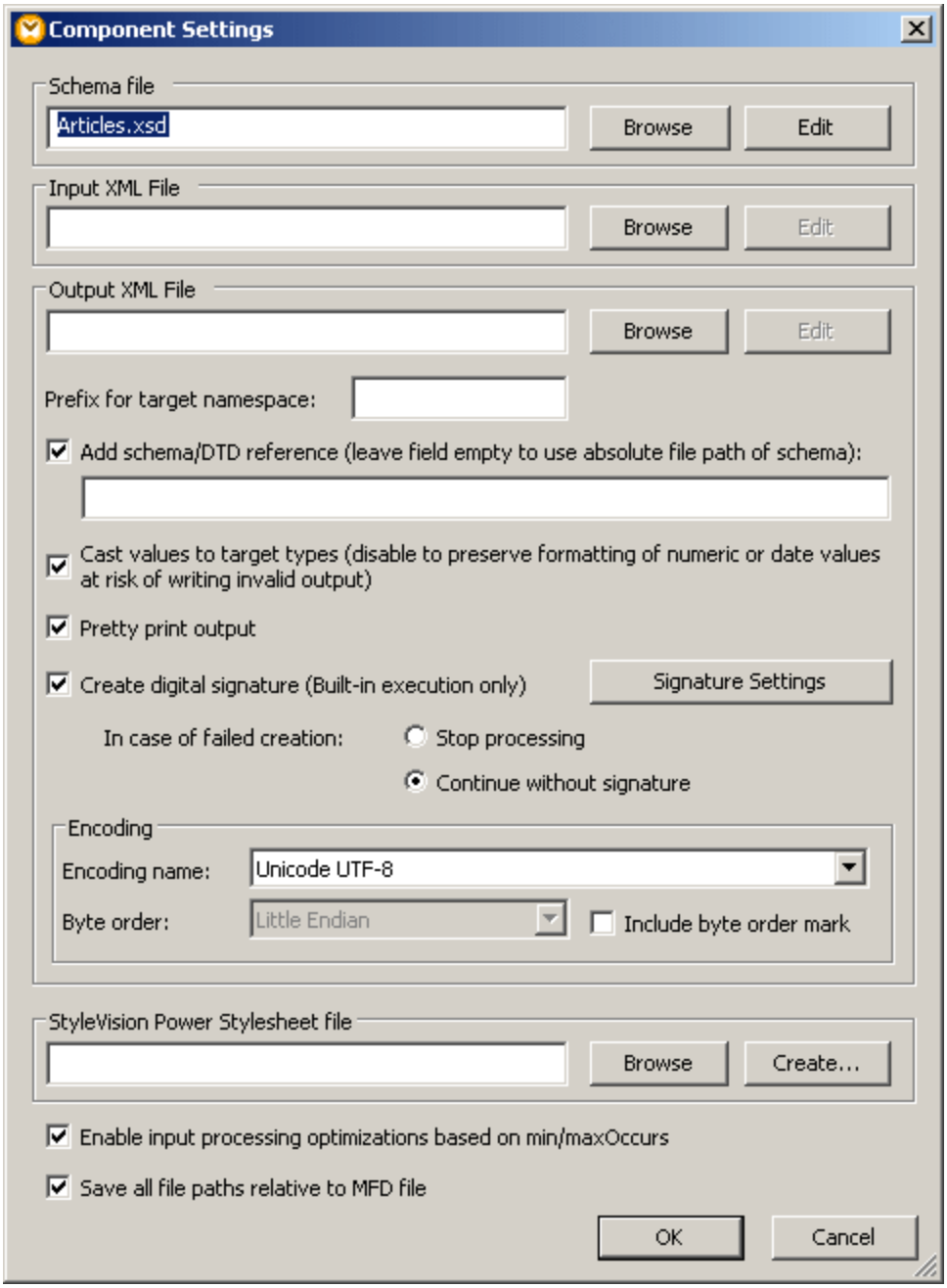
Database Table Actions
Allows you to define the actions to be performed with the mapped data on the specific target database table. Please see Table actions, key settings for more information.

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 currently selected component settings. If the component is an XML-Schema file then the Component Settings dialog box is opened. If the component is a Text file, then the "Text import / export" dialog box is opened.



Schema file: Shows the file name and path of the target schema.

Input XML-File: Allows you to select, or change the XML-Instance for the currently selected schema component. This field is filled when you first insert the schema component and assign an XML-instance file.

Output XML-File: This is file name and path where the XML target instance is placed, when generating and executing program code. The file name is also visible as the first item in the component.

The entry from the Input XML-Instance field, is automatically copied to this field when you assign the XML-instance file. If you do not assign an XML-Instance file to the component, then this field contains the entry **schemafilenameandpath.xml.**

Prefix for target namespace: Allows you to enter a prefix for the Target Namespace if this is a schema / XML document. A Target namespace has to be defined in the target schema, for the prefix to be assigned here.

Add Schema/DTD reference: Adds the path of the referenced XML Schema file to the root element of the XML output.

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. E.g. if you want to send the resulting XML output to someone who does not have access to the underlying XML Schema.

Cast target values to target types: Allows you to define if the target XML schema types should be used when mapping (default - active), or if all data mapped to the target component should be treated as **string** values.

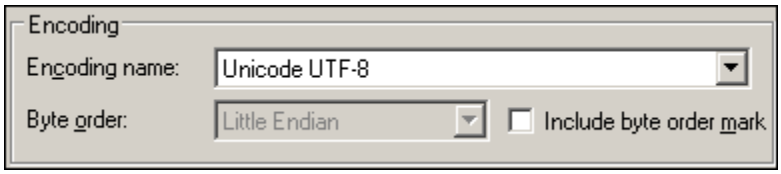
Deactivating this option allows you to retain the precise formatting of values. E.g., this is useful to "satisfy" a pattern facet in a schema, that requires a specific number of decimal digits in a numeric value.

You can use mapping functions to format the number as a string in the required format, and then map this string to the target.

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

Pretty-print output: Reformats 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.)

Encoding settings



From MapForce version 2008 each component, source, as well as target, has its own encoding settings. This means that the *.mfd mapping files do not have a default encoding, each component that makes up the mapping file has its own. Components in this general sense are all XML, Text components.

There is however a default encoding setting defined in the **Tools | Options** General tab, called "Default encoding for new components" which is applied whenever new components are created/inserted. When mappings from previous versions are opened, the default encoding setting will be used.

The encoding control group consists of 3 controls:

- Encoding name selection combobox.
- Byte order selection combobox (little endian, big endian).
- Include Byte Order Mark checkbox.

Default settings are:

- UTF-8
- little endian (disabled for UTF-8)
- no Byte Order Mark.

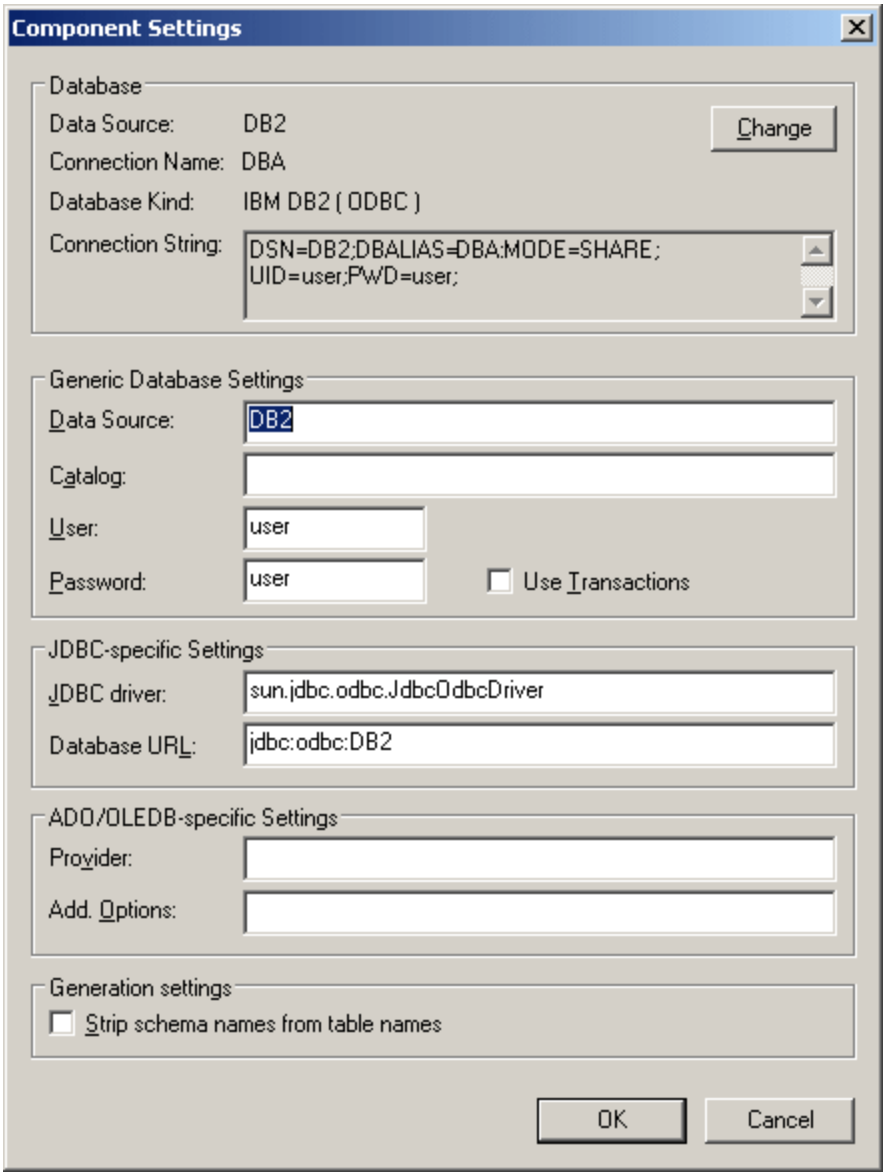
Please note:

Activating the Byte Order Mark check box in the Component Settings dialog box, does not have any effect when outputting **XSLT 1.0/2.0,** as these languages do not support BOMs (Byte Order Marks).

Enable input processing optimizations based on min/maxOccurs
MapForce version 2009 introduces special handling for sequences that are known to contain exactly one item, e.g. 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**, this optimization can be disabled in the component settings of XML and EDI components.

The **database settings** for this dialog box are only displayed if you open the component settings dialog box of a database component.



Database

Data Source: displays the name of the current database component. Clicking the **Change** button allows you to select a different database, or redefine the tables that are to be in the component if you select the same database. 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 tables**.

Connection String: Displays the current database connection string. This field cannot be edited.

Generic Database settings:

DataSrc: displays the data source name.

Catalog: displays the name of the specific database.

User: Enter the user name needed to access the database, if required.

Password: Enter the password needed to access the database, if required

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.

JDBC -specific Settings:

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. Please 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.

User: Enter the user name needed to access the database, if required.

Password: Enter the password needed to access the database, if required.

ADO/OLEDB-specific settings:

Provider: Displays the currently active provider for the database component. The provider is automatically entered when you define the database component.

add. Options: Displays additional database options.

Generation settings:

Strip schema names from table names: allows you to omit database schema names from generated code, only retaining the table names for added flexibility.

Note that this option does not work for SQL Select statements.

22.6 Connection

Auto Connect Matching Children
Activates or de-activates the "Auto connect child items" function, 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.

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. Please see the section on Connector properties for further information.

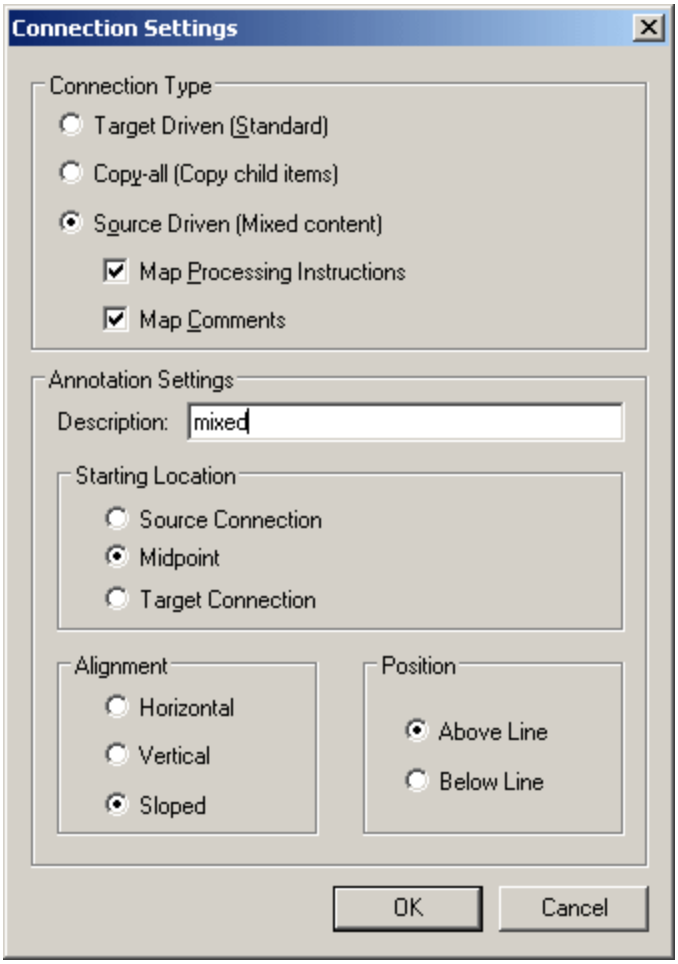
Target Driven (Standard)
Changes the connector type to Standard mapping, please see: "Source-driven / mixed content vs. standard mapping" for more information.

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, please see "Copy-all connections" for more information.

Source Driven (Mixed Content)
Changes the connector type to source driven / mixed content, and enables the selection of additional elements to be mapped. The additional elements have to be **child items** of the mapped item in the XML source file, to be able to be mapped. Please see Default settings: mapping mixed content for more information.

Properties:
Opens a dialog box in which you can define the specific (mixed content) settings of the current connector. Note that unavailable options are greyed out.

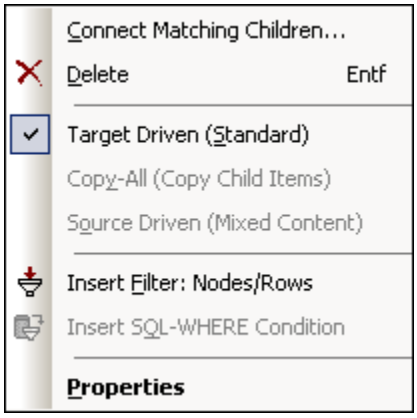
Please note that these settings also apply to **complexType** items which do not have any text nodes!



Annotation settings:
Individual connectors can be **labeled** for clarity.

1. Double click a connector and enter the name of the connector in the Description field. This enables all the options in the Annotation Settings group.
2. Use the remaining groups to define the position and alignment of the label.

Connector context menu:



Connect matching children

Opens the "Connect Matching Children" dialog box, allowing you to change the connection settings and connect the items when confirming with OK.

Delete
Deletes the selected connector.

Target Driven (Standard)
Changes the connector type to Standard mapping, please see: "Source-driven / mixed content vs. standard mapping" for more information.

Copy-all (Copy Child Items)
Changes the connector type to "Copy-all" and connects all child items of the same name in a graphically optimized fashion, please see "Copy-all connections" for more information.

Source Driven (Mixed Content)
Changes the connector type to source-driven / mixed content, please see: "Source driven and mixed content mapping" for more information.

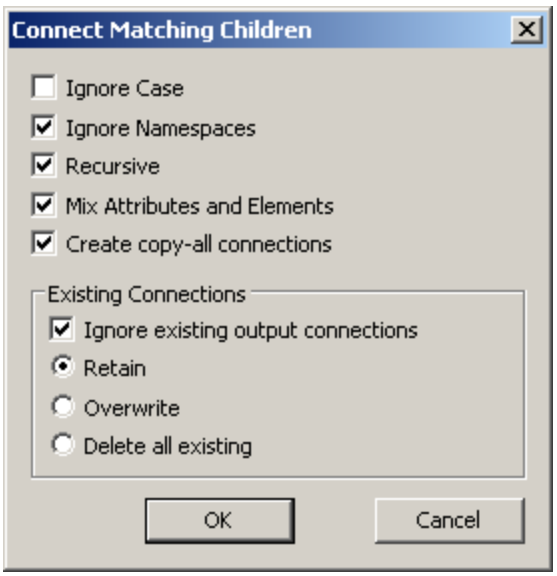
Insert Filter: Nodes/Rows
Inserts a Filter component into the connector. The source connector is connected to the nodes/row parameter, and the target connector is connected to the on-true parameter. Please see Filter - retrieving dynamic data, lookup table for more information.

Insert SQL-Where Condition
Inserts a SQL-Where component into the connector. The source connector is connected to the table parameter, and the target connector is connected to the result parameter. Please see SQL SELECT Statements as virtual tables for more information.

Properties:
Opens the Connections Settings dialog, in which you can define the specific mixed content settings as well as the connector annotation settings, please see the Connection section in the Reference section.

Connect Matching Children dialog box
This command allows you to create multiple connectors between items of the **same name** in both the source and target components. Note that a copy-all connection is created by default.

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



3. Select the required options discussed in the text below, and click OK to create the connectors.

   Connectors are created for all the child items that have identical names and adhere to the settings defined in the dialog box.

Please note:
The settings you define here 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.

Ignore Case:
Ignores the case of the child item names.

Ignore Namespaces:
Ignores the namespaces of the child items.

Recursive:
Having created the first set of connectors, the grandchild items are then checked for identical names. If some exist, then connectors are also created for them. The child elements of these items are now checked, and so on.

Mix Attributes and Elements:
Allows the creation of connectors between items of the same name, even if they are of different types e.g. two "Name" items exist, but one is an element, the other an attribute. If set active, a connector is created between these items.

Create copy-all connections:
Default setting is active. Creates copy-all connection between source and target items if possible.

Existing connections:

Ignore existing output connections:
Creates **additional** connectors to other components, even if the currently existing output icons already have connectors.

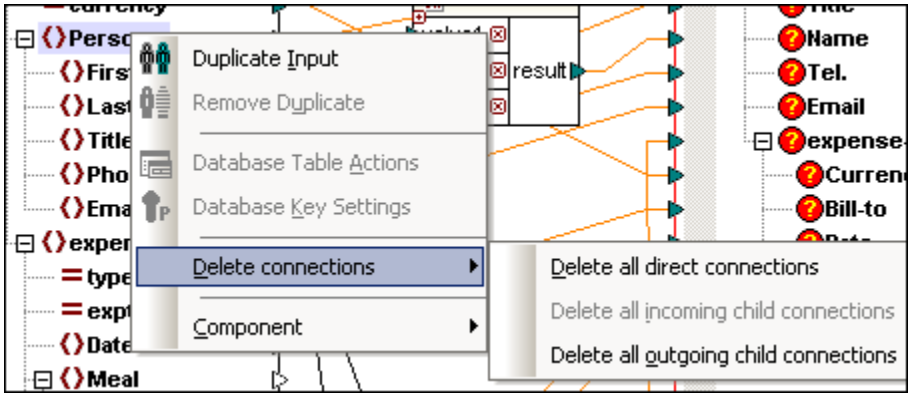
Retain
Retains existing connectors.

Overwrite:
Recreates connectors, according to the settings defined. Existing connectors are scrapped.

Delete all existing:
Deletes all existing connectors, before creating new ones.

Deleting connections

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



Right click the item name in the component, not the connector itself, Person in this example. Select **Delete Connections | Delete all ... connections**.

Delete all direct connections:
Deletes all connectors 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 connectors.

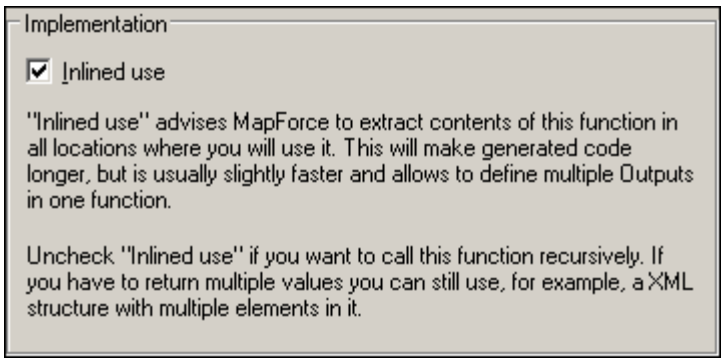
Delete all outgoing child connections:
Only active if you have right clicked an item in a source component. Deletes all outgoing child connectors.

22.7 Function

Create User-Defined Function...:
Creates a new user-defined function. Selecting this option creates and empty user-defined function, into which you insert the components you need. A single output component is automatically inserted when you define such a function, and only one output component can be present in a user-defined function unless it is defined as inlined. Please see "Creating a user-defined function from scratch" for more information.

Create User-Defined Function from Selection:
Creates a new user-defined function based on the currently selected elements in the mapping window. Please see "Adding user-defined functions" for more information.

Function Settings:
Opens the settings dialog box of the currently active user-defined function allowing you to change the current settings. Use this method to change the user-defined function type, i.e. double click the title bar of a user-defined function to see its contents, then select this menu option to change its type.

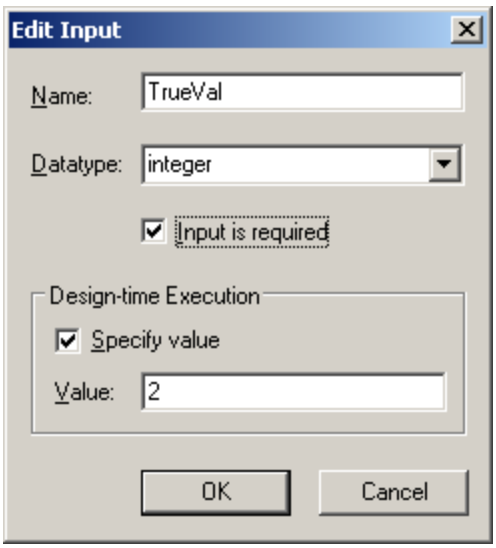


Remove Function
Deletes the currently active user-defined function while working on an existing user-defined function, in the tab of that name. I.e this only works on existing user-defined functions while viewing their contents.

A prompt appears reminding you that instances may become invalid and in what libraries the user-defined function exists.

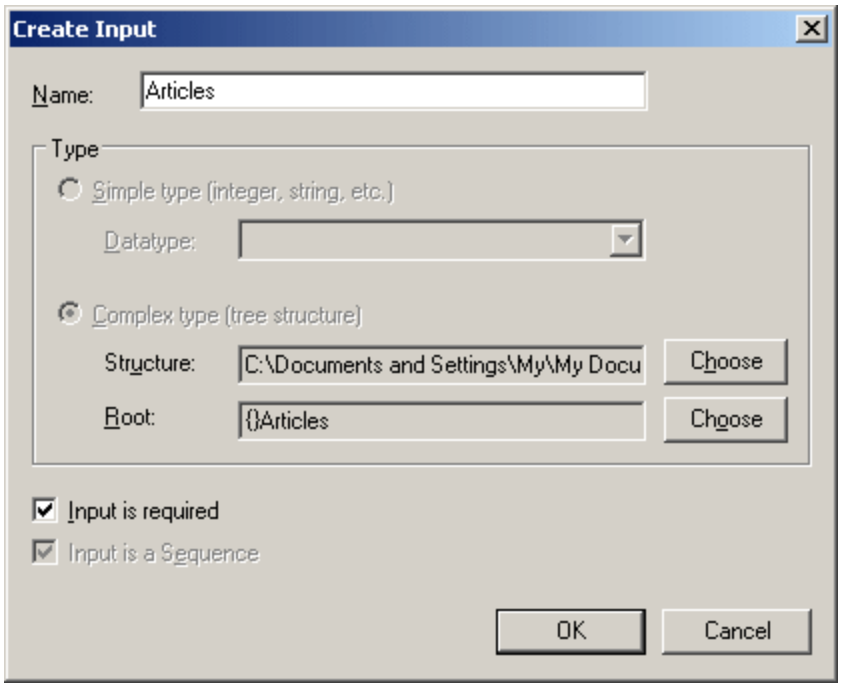
Insert Input:
Inserts an "input" component into the mapping, or into a user-defined function.

If you are working in the **main** Mapping tab, the dialog box shown below is displayed. This type of input component allows you to define a **parameter** in the command line execution of the compiled mapping.
Please see "Input values, overrides and command line parameters" for more information.



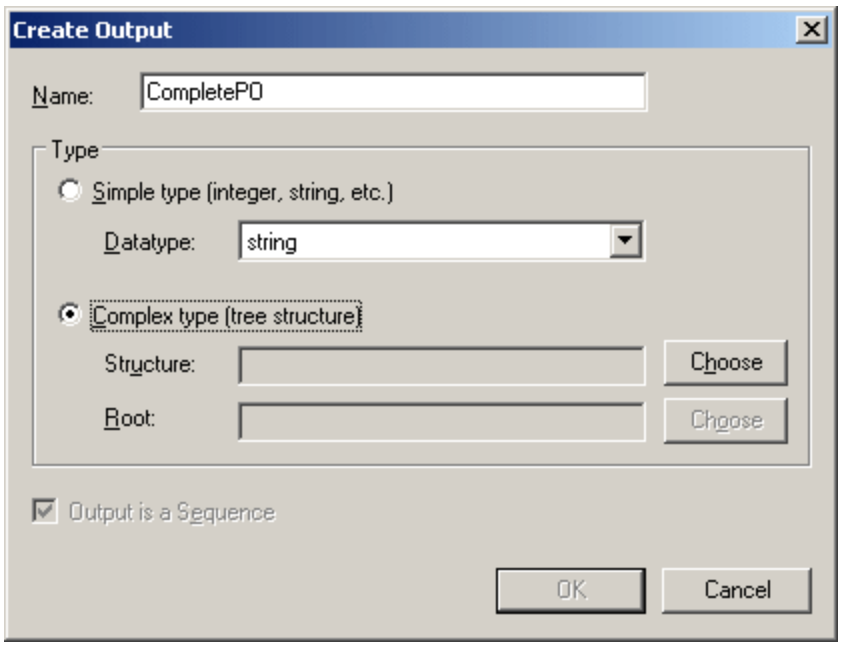
If you are working in a user-defined function tab, the dialog box shown below is displayed. This type of input component allows you to define:

- simple inputs
- complex inputs, e.g. schema structures



Insert Output
Inserts an "Output" component into a user-defined function. In a user-defined function tab, the dialog box shown below is displayed. This type of input component allows you to define:

- simple outputs
- complex outputs, e.g. schema structures


Create Output
Name:
CompletePO
Type
Simple type (integer, string, etc.)
Datatype:
string
Complex type (tree structure)
Structure:
Choose
Root:
Choose
Output is a Sequence
OK
Cancel

22.8 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.

Note that the Built-in execution engine presents a preview of the mapping result, when you click the Output tab and cannot be used to generate program code. Please see the Built-in execution engine section for more information.

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. Please see: Dynamic Input/Output file name for more information.

Regenerate output
Regenerates the current mapping from the Output windown. 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

Run SQL-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.

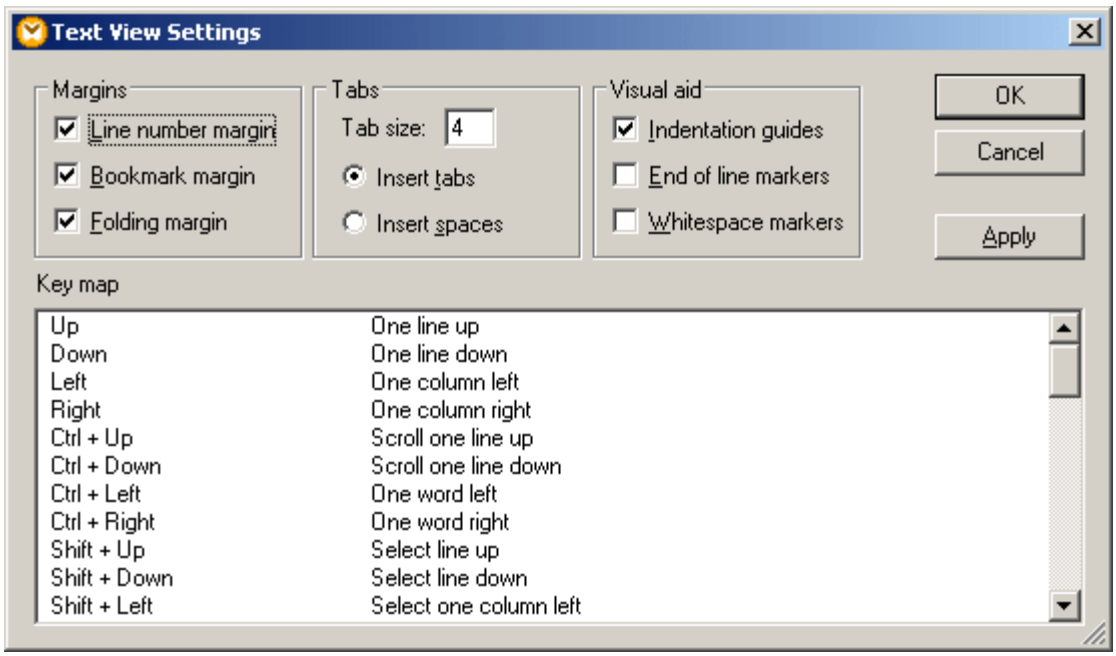
Next Bookmark
Navigates to the next bookmark in the Output window.

Previous Bookmark
Navigates to the previous bookmark in the Output window.

Remove All Bookmarks
Removes all currently defined bookmarks in the Output window.

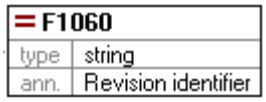
Pretty-Print XML Text
Reformats your XML document in the Output pane to give a structured display of the document. Each child node is offset from its parent by a single tab character.

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.


Text View Settings
Margins
Line number margin
Bookmark margin
Folding margin
Tabs
Tab size: 4
Insert tabs
Insert spaces
Visual aid
Indentation guides
End of line markers
Whitespace markers
OK
Cancel
Apply
Key map
Up One line up
Down One line down
Left One column left
Right One column right
Ctrl + Up Scroll one line up
Ctrl + Down Scroll one line down
Ctrl + Left One word left
Ctrl + Right One word right
Shift + Up Select line up
Shift + Down Select line down
Shift + Left Select one column left

22.9 View

Show Annotations 
Displays XML schema annotations in the component window.
If the Show Types icon is also active, then both sets of info are show in grid form.



Show Types 
Displays the schema datatypes for each element or attribute.
If the Show Annotations icon is also active, then both sets of info are show in grid form.

Show library in Function Header 
Displays the library name in parenthesis in the function title.

Show Tips 
Displays a tooltip containing explanatory text when the mouse pointer is placed over a function.

Show Selected Component Connectors 
Switches between showing:

- all mapping connectors, or
- those connectors relating to the currently selected components.

Show Connectors from Source to Target 
Switches between showing:

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

Zoom
Opens the Zoom dialog box. You can enter the zoom factor numerically, or drag the slider to change the zoom factor interactively.

Back
Steps back through the currently open mappings of the mapping tab.

Forward
Steps forward through the currently open mappings of the mapping tab.

Status Bar
Switches the Status Bar, visible below the Messages window, on or off.

Library Window
Switches the Library window, containing all library functions, on or off.

Messages
Switches the Validation output window on, or off. When generating code the Messages output window is automatically activated to show the validation result.

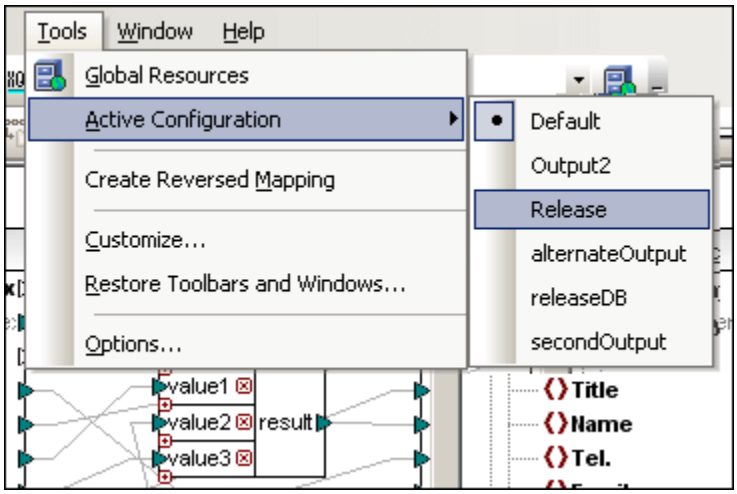
Overview
Switches the Overview window on, or off. Drag the rectangle to navigate your Mapping view.

Project window
Switches the Project window on, or off.

22.10 Tools

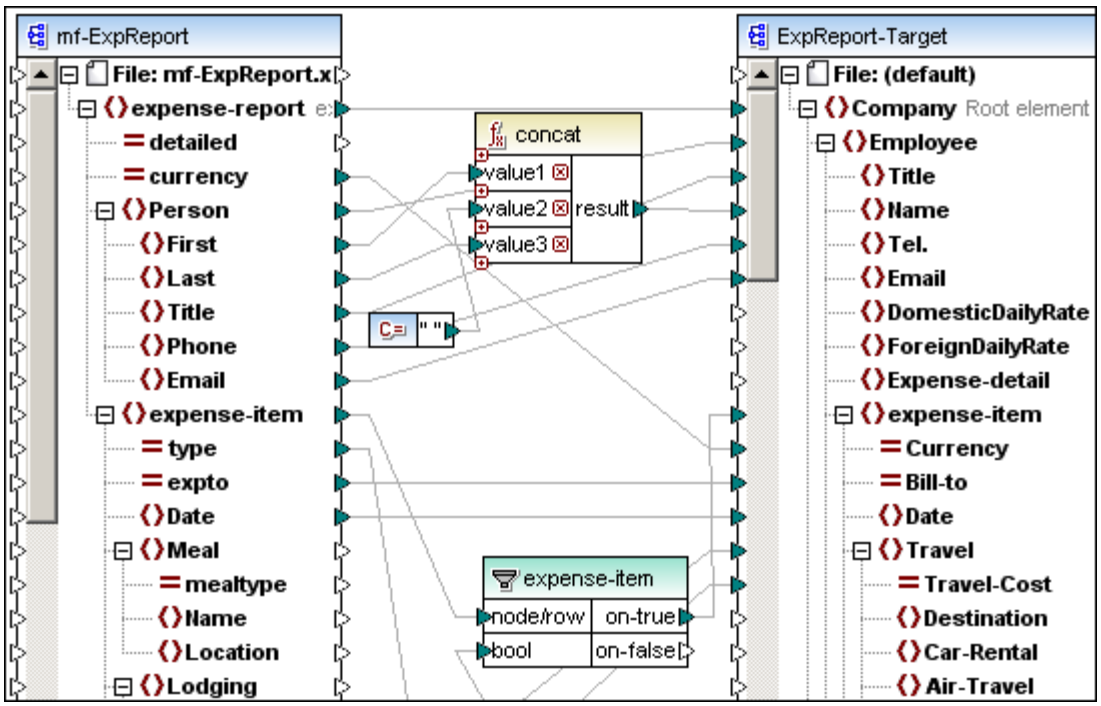
Global Resources
Opens the Manage Global Resources dialog box allowing you to Add, Edit and Delete global resources to the Global Resources XML file, please see Global Resources - Properties for more information.

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.

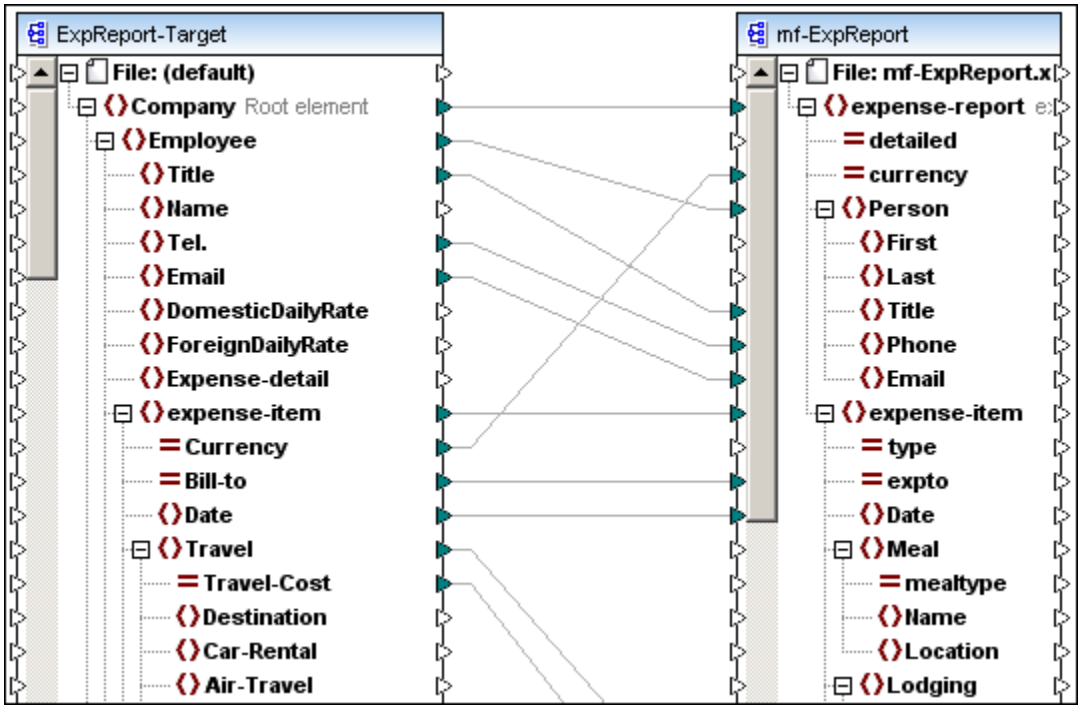


Create Reversed Mapping
Creates a "reversed" mapping from the currently active mapping in MapForce, which is to be the basis of a new mapping. Note that the result is not intended to be a complete mapping, only the direct connections between components are retained in the reversed mapping. It is very 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.

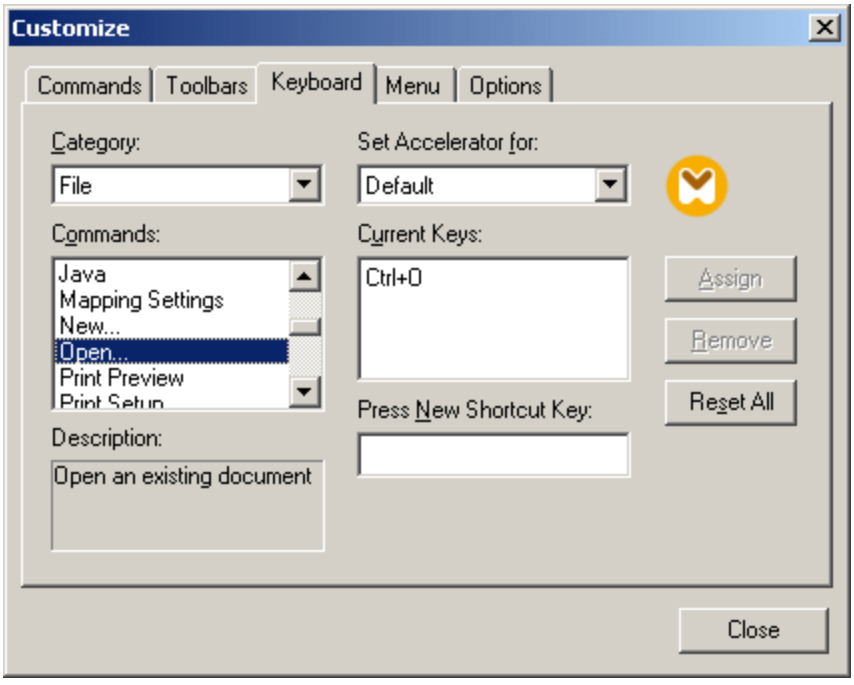
Customize...

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

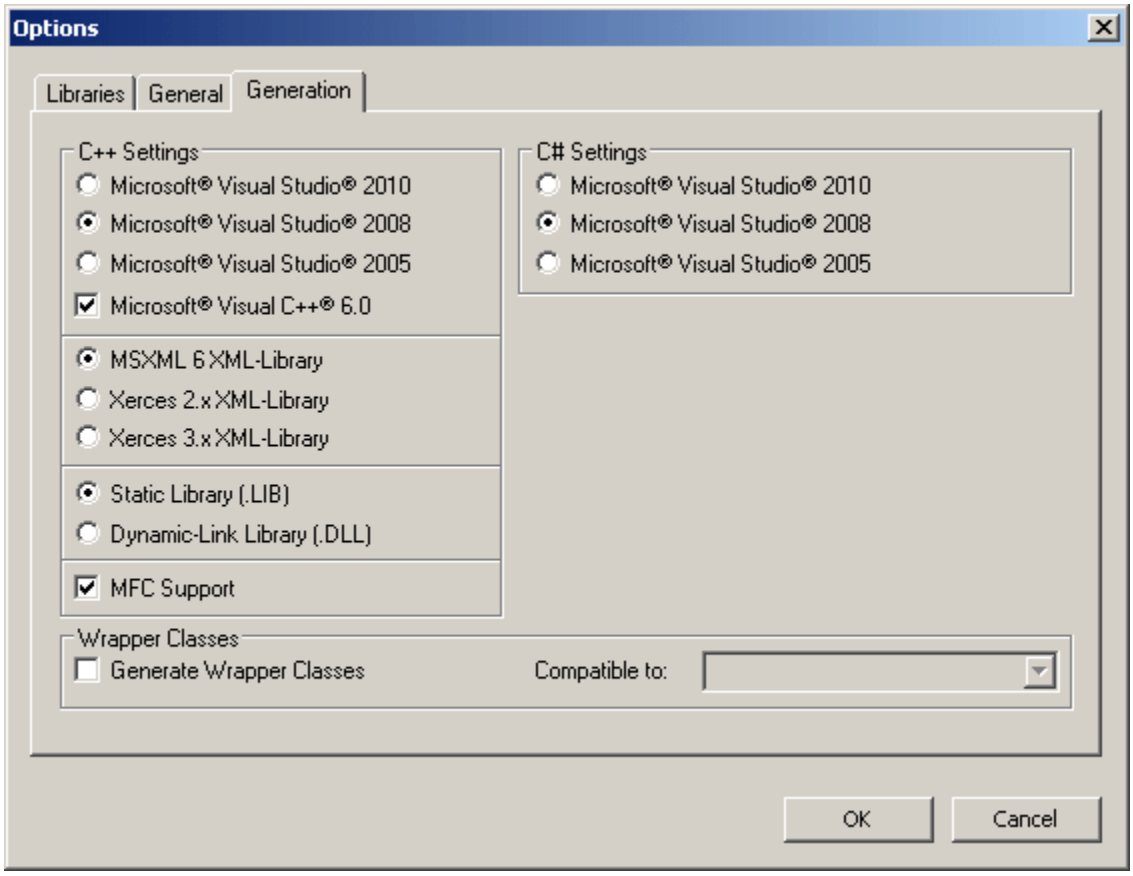
Zooming hotkeys

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 your specific compiler and IDE settings.



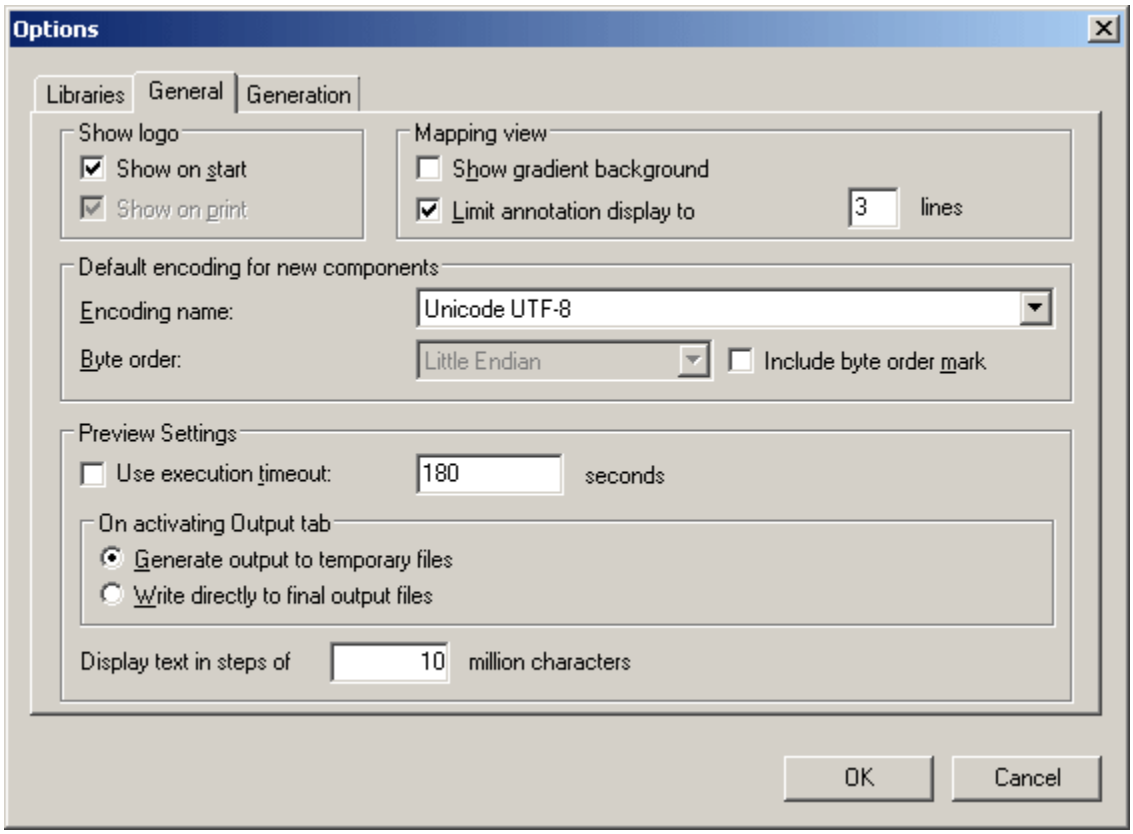
C++ Settings:

Defines the specific compiler settings for the C++ environment.

C# Settings:

Defines the specific compiler settings for the C# environment.

Wrapper Classes:
Allows you to select the version compatibility of the generated wrapper classes for XML schemas. Currently available options are: V2005r3, 2006-2007, 2007r3 and higher. These wrapper classes can be used by custom code that includes the code generated by MapForce.



General tab:

- Specify if you want to show the logo on start and/or when printing.
- 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
- 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.

- Allows you to 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

Libraries tab:

- Add or delete user-defined XSLT, or programming language Libraries/functions to MapForce.

22.11 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.

22.12 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

22.12.1 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.

22.12.2 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

22.12.3 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.

22.13 Oracle client installation

The instructions below describe the setting up of a new connection to an existing Oracle database somewhere on the local network. The Local net service name configuration wizard follows the same sequence when installing the Net Service during the initial installation of the Oracle client.

1. Select the menu option **Programs | Oracle - OraHome92 | Configuration and migration tools | Net Configuration Assistant**.
   This opens the Oracle Net Configuration Assistant.
2. Click the **Local Net Service Name configuration** radio button and click Next.
3. Click **Add** to add a new net service name and click Next.
4. Select the installed Oracle version, e.g. **Oracle 8i** or later... and click Next.
5. Enter the **Service Name** of the database you want to connect to e.g. TestDB and click Next. The database's service name is normally its global database name.
6. Select the **network protocol** used to access the database e.g. TCP, and click Next.
7. Enter the **Host name** of the computer on which the database is installed, and enter the port number if necessary. Click Next to continue.
8. Click the **Yes** radio button, to test the database connection, and click Next.
9. You can change the Login parameters if the test was not successful, by clicking the Change Login button, and trying again. Click Next to continue.
10. Enter the **Net Service Name** in the field of the same name, this can be any name you want. This is the name you will enter in the Database field, of the **Oracle login** dialog box in MapForce. Click Next to continue.
11. This completes the Net Service Name configuration. Click Next to close the dialog box.

Chapter 23

Code Generator

23 Code Generator

MapForce includes a built-in code generator which can automatically generate Java, C++ or C# class files from XML Schema definitions, text files, and databases.

Mapping is not limited to simple one-to-one relationships; MapForce allows you to mix multiple sources and multiple targets, to map any combination of different data sources in a mixed environment.

The result of the code generation is a fully-featured and complete application which performs the mapping for you. You can run the application directly as generated, or you may insert the generated code into your own application, or extend it with your own functionality.

23.1 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.

Additional code is implemented, such as functions which read XML files into a Document Object Model (DOM) in-memory representation, write XML files from a DOM representation back to a system file, or to 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 C++ 2010 Microsoft Visual C++ 2008 Microsoft Visual C++ 2005 Microsoft Visual C++ 6.0	Microsoft Visual Studio 2010 Microsoft Visual Studio 2008 Microsoft Visual Studio 2005	Apache Ant (build.xml file) Eclipse 3.4+ Borland JBuilder
XML DOM implementations	MSXML 6.0 or Apache Xerces 2.6 or later	System.Xml	JAXP
Database API (MapForce only)	ADO	ADO.NET	JDBC

C++
The C++ generated output uses either MSXML 6.0, or Apache Xerces 2.6 or later. Both MapForce and XMLSpy generate complete project and solution/workspace files for Visual C++ 6.0 or Visual Studio 2005 to 2010 directly. **.sln** and .**vcproj** files are generated in addition to the **.dsw /.dsp** files for Visual Studio 6.0. The generated code optionally supports MFC, e.g. by generated CString conversion methods.

Please note:
When building C++ code for Visual Studio 2005/2008/2010 and using a Xerces library precompiled for Visual C++ 6.0, a compiler setting has to be changed in all projects of the solution:

1. Select all projects in the Solution Explorer.
2. [Project] | [Properties] | [Configuration Properties] | [C/C++] | [Language]
3. Select *All Configurations*
4. 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, e.g. VB.NET, Managed C++, J# or any of the several languages that target the .NET platform. 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 a JBuilder project file, an Ant build file and project files for Eclipse. Java 5 (JDK 1.5) or higher is supported.

Generated output in MapForce:

Generated output	Location	MapForce
Standard libraries	"Altova" folder	☑
Schema wrapper libraries	Schema name folder	☑
Database type info libraries	Database name folder	☑
		☑
Application		
Mapping application (complete app.) Compiling and executing, performs the defined mapping.		☑
Mapping application can be:	nowextended by user, or imported into own application	☑

Code generator templates
Output code is completely customizable via a simple yet powerful template language which gives full control in mapping XML Schema built-in data-types to the primitive datatypes of a particular programming language.
It 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).

23.2 What's new ...

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 R2

-

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.

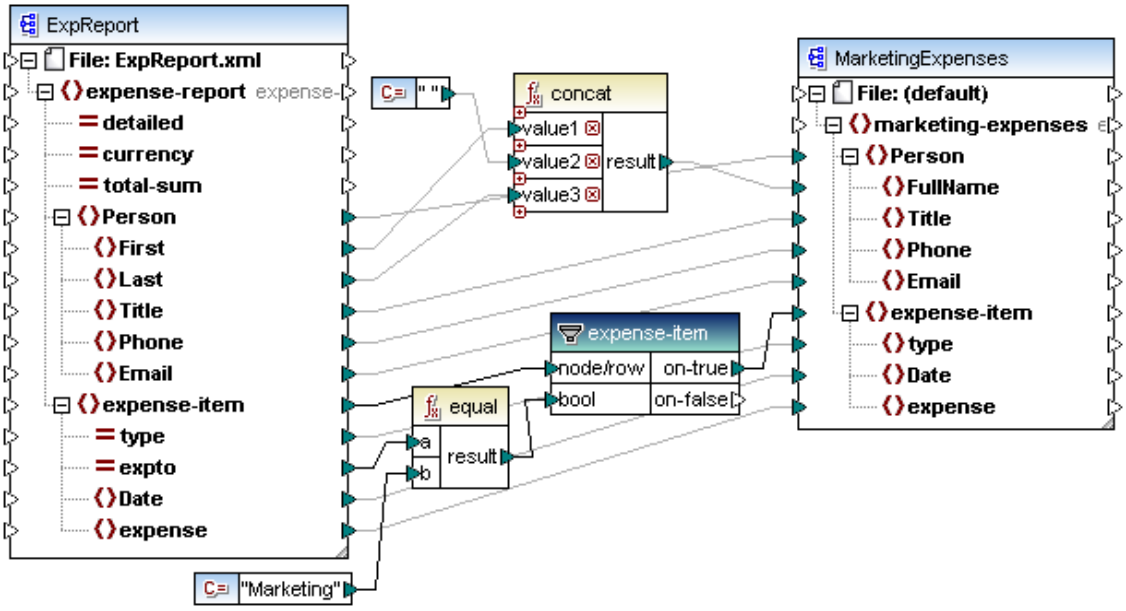
- Handling of XML documents and nodes with explicit ownership, to avoid memory leaks and to enable multi-threading
- New syntax to avoid name collisions
- New data types for simpler usage and higher performance (native types where

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

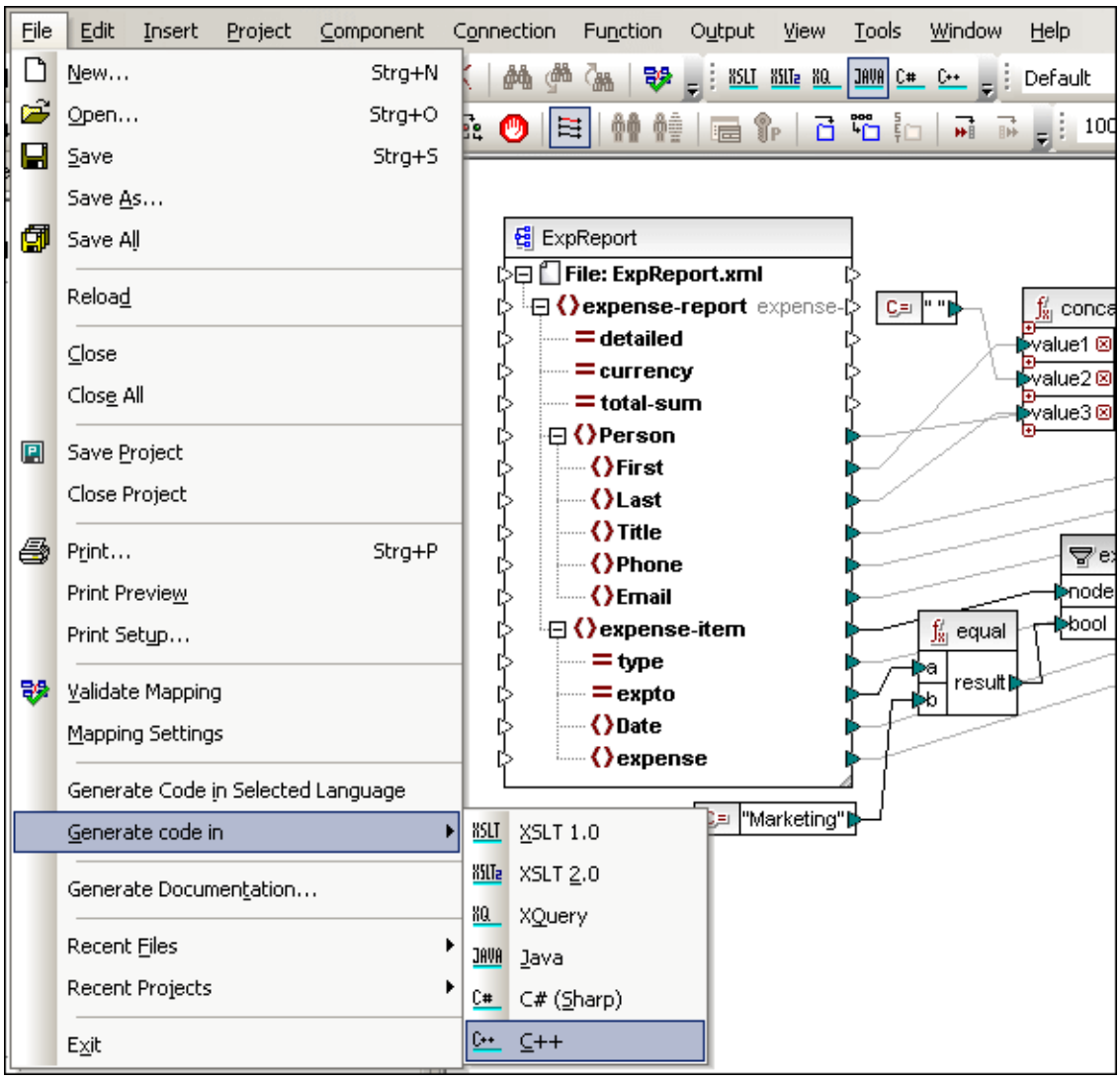
23.3 Generating program code

This example shows the general sequence that needs to be followed to generate program code from MapForce. The example uses the MarketingExpenses.mfd file available in the **...\MapForceExamples** folder. Please also see Integrating code in your application for more information.

1. Open the **MarketingExpenses.mfd** mapping file.



2. Select the menu option **File | Generate code in | C++**.



The **Browse for Folder** dialog box opens at this point.

3. Navigate to the folder that you want the code to be placed in, and click OK to confirm. A "Code Generation completed successfully" message appears.
4. The generated code is placed in subdirectories below the directory you specified, and contains all the necessary libraries etc. needed to compile and execute the mapping code.

The sequence shown here is repeated later in this document, with additional information on the build and compile process of each of the programming languages:

For more information please see the sections below:
Generating Java code
Generating C# code
Generating C++ code

23.3.1 Generating Java code

Prerequisites and default settings:

The generated MapForce application supports Java 5 or higher.

JDBC drivers have to be installed to compile Java code when mapping database data. Please see the section JDBC driver setup for more information.

The menu option **File | Mapping settings** defines the mapping project settings. The default settings are:

- Application name=Mapping
- Base Package Name=com.mapforce

JDBC-specific settings can be viewed by clicking a database component, and selecting the menu option **Component | Properties**.

The table below shows the different applications that can be compiled in each of the environments:

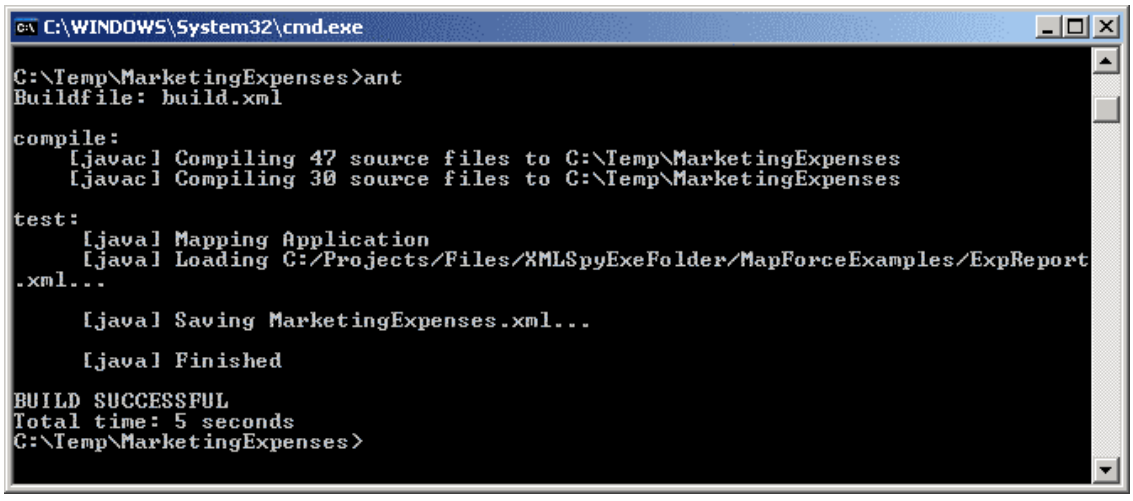
File name (with default application name)	Note
MappingConsole.java	Console application
MappingApplication.java	Dialog application

To generate Java code in MapForce:

1. Select the menu option **File | Generate code in | Java**.
   You are then prompted for the target directory of the generated files.
2. Select the directory you want to place the files in, and click OK to confirm (eg. c:\codegen\Java).
   A "Java Code generation completed" message appears when the process was successful. If not, an error message appears detailing the specific error(s).

If you are using an **Ant** build script:

- Navigate to the Java subdirectory and execute "ant" (which automatically opens the **build.xml** file)
- This will **compile** and **execute** the Java code. The XML target instance file is automatically generated at the end of this sequence.

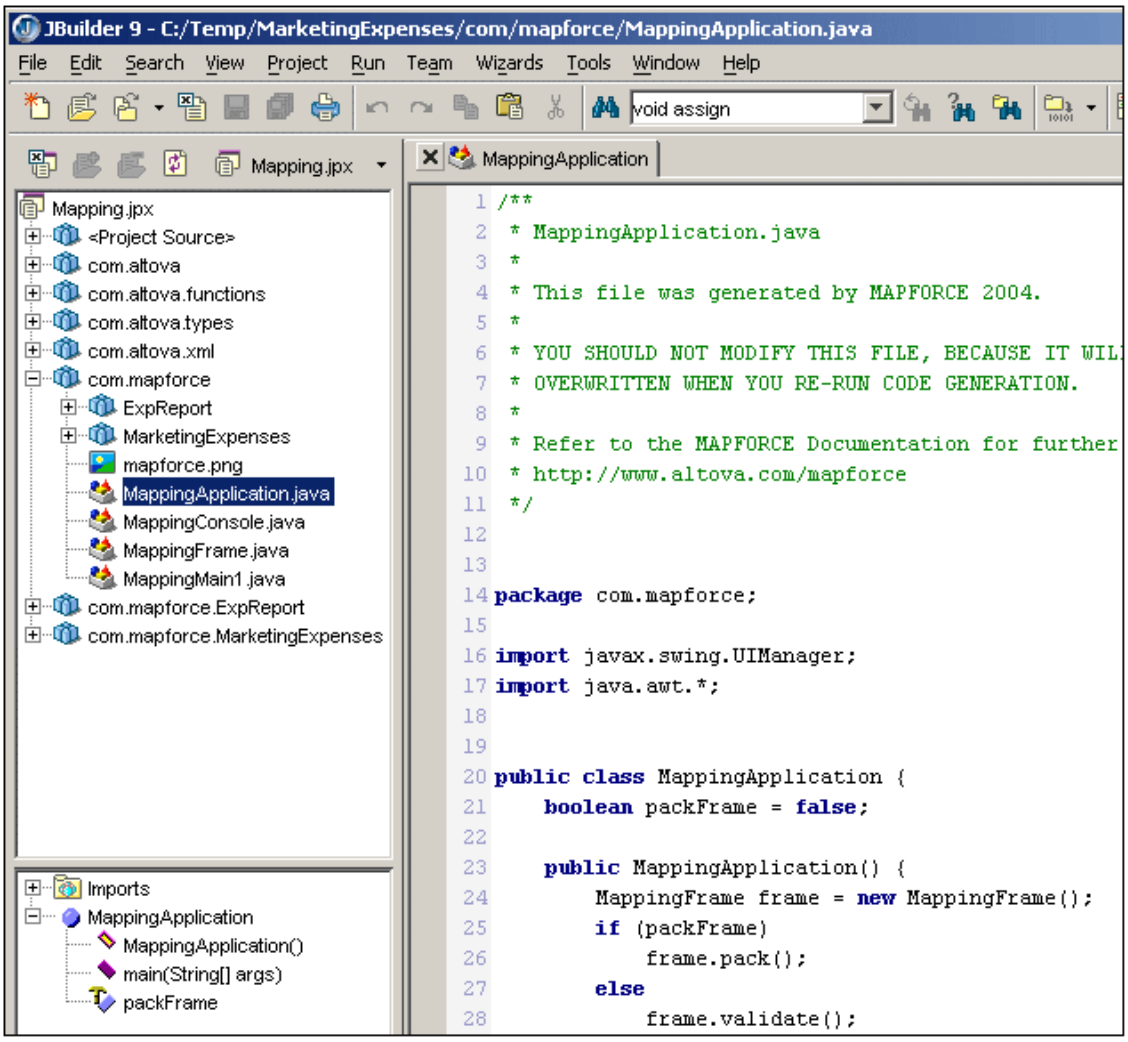


For further information please see:
Generating Java code using JBuilder

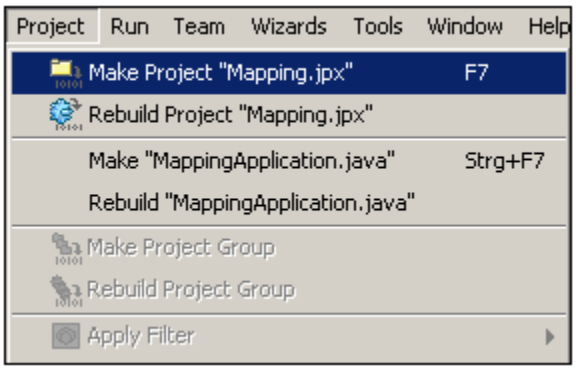
Generating Java code using JBuilder

If you are using Borland **JBuilder**,

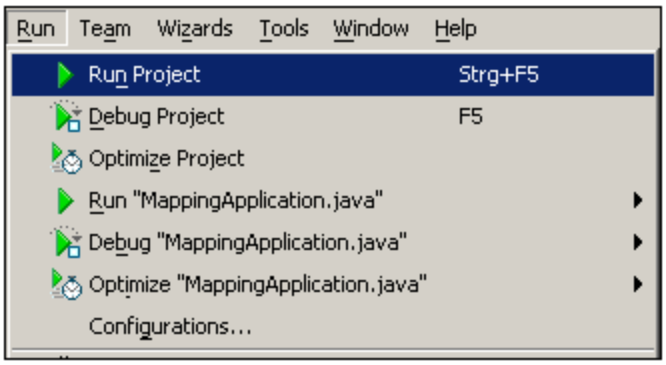
- Navigate to the Java subdirectory and open the **Mapping.jpx** file to compile the Java code,



Select **Make Project "Mapping.jpx"** to compile the Java file.



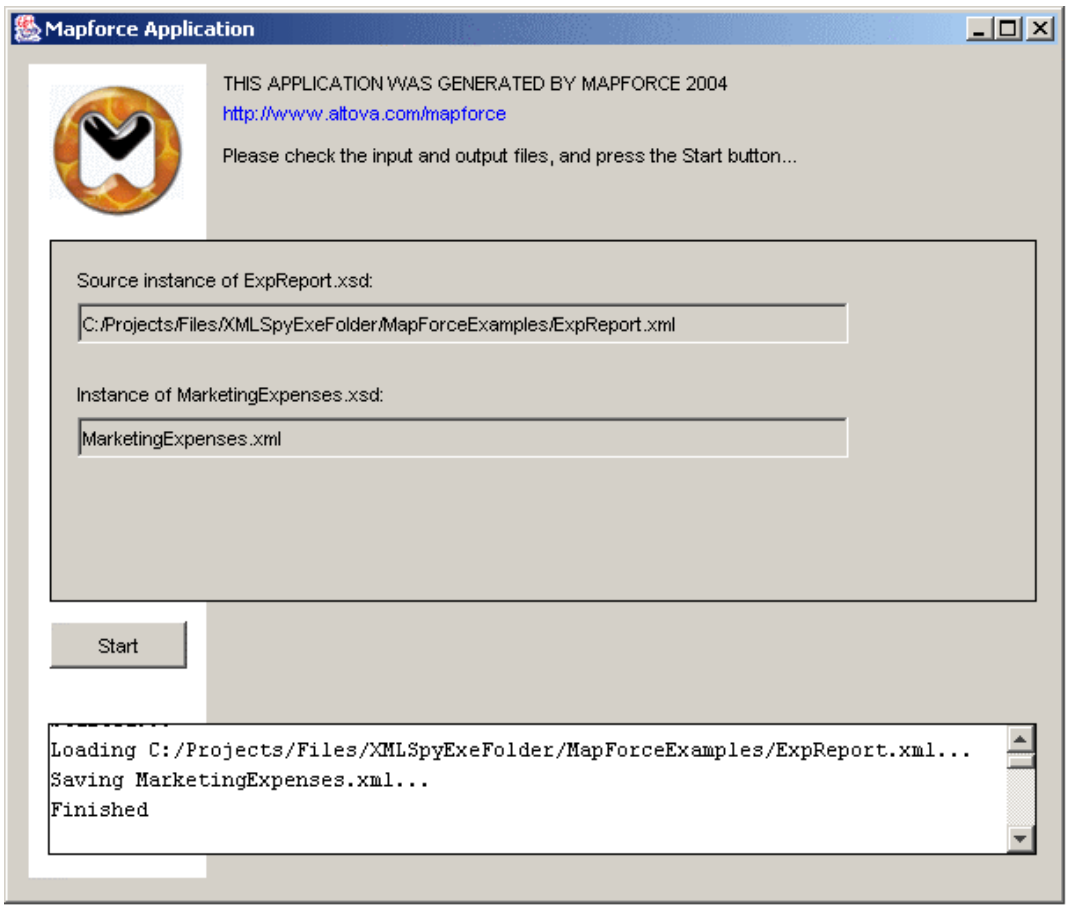
Select the menu option **Run | Run project.**



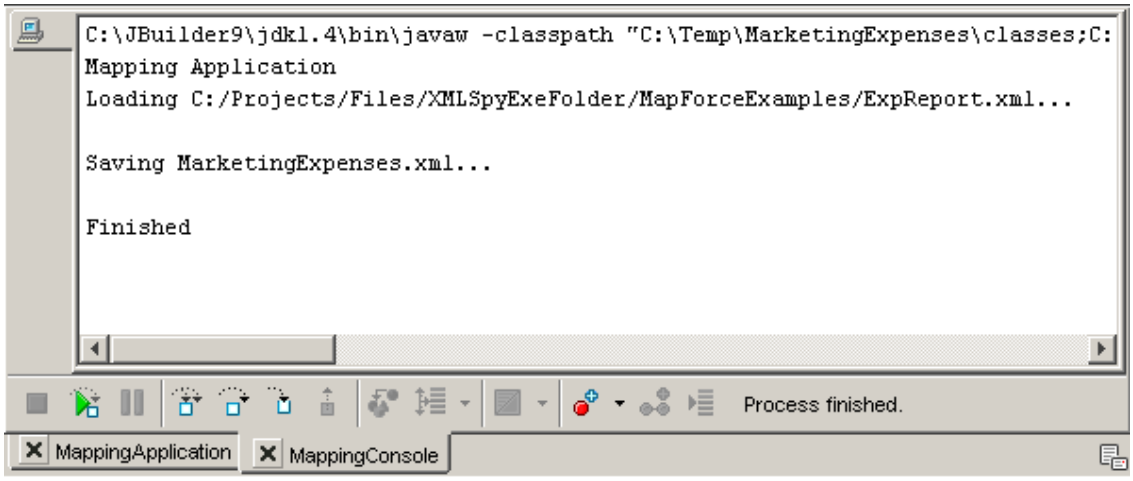
Execute either:

- MappingApplication, or
- MappingConsole
  In both cases the MarketingExpenses.xml target file is created.

MappingApplication



MappingConsole
The screenshot below shows the mapping output in JBuilder.



The **com** subdirectory contains the generated code in various subdirectories.

23.3.2 Generating C# code

Prerequisites and default settings:

The menu option **File | Mapping settings** defines the mapping project settings. The default settings are:

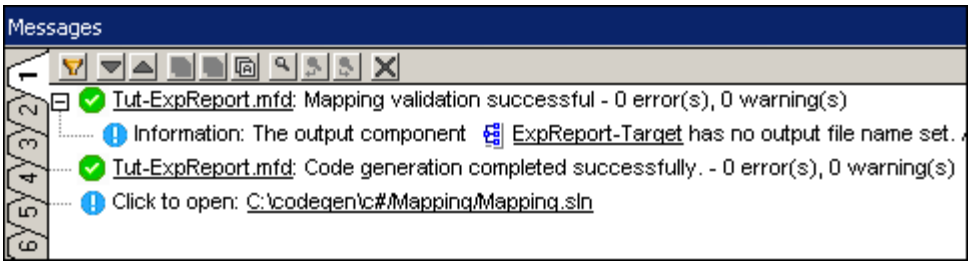
- Application name=Mapping

Database-specific settings can be viewed by clicking a database component, and selecting the menu option **Component | Properties**.

.sln and **csproj** files are generated for Microsoft Visual Studio. The **Generation** tab under menu option **Tools | Options** allows you to choose the target IDE version. Mono users can use Visual Studio solution files with Mono's xbuild.

To generate C# code in MapForce:

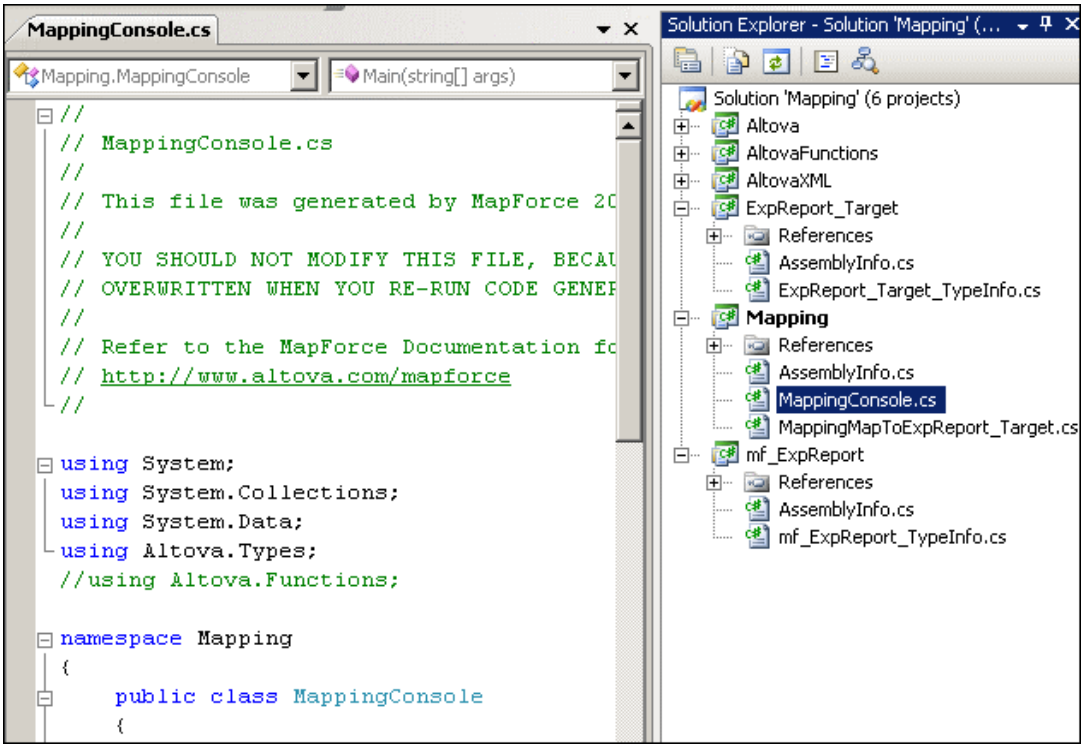
1. Select the menu option **File | Generate code in | C# (sharp)**.
   You are then prompted for the target directory of the generated files.
2. Select the directory you want to place the files in, and click OK to confirm (eg. c:\codegen\C#).
   A "C# Code generation completed" message appears when the process was successful. If not, an error message appears detailing the specific error(s).



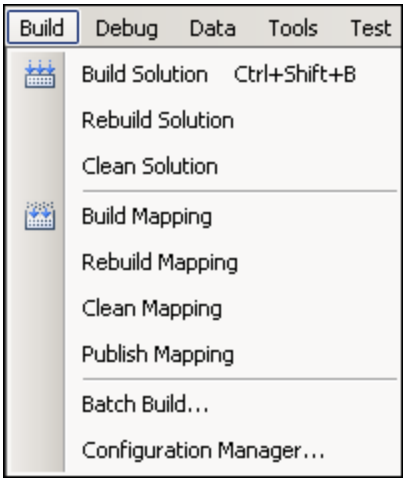
Folder **c:\codegen\C#\mapping**
You can now compile the project in Microsoft Visual Studio.

- Mapping.sln (for Visual Studio)

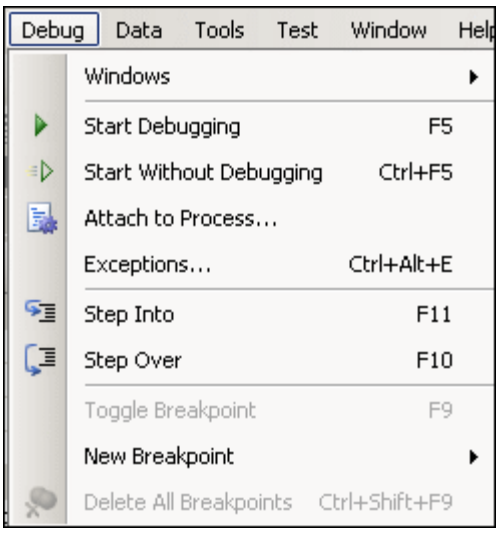
1. Click the "**Click to open: C:\codegen\c#/Mapping/Mapping.sln**" link in the Messages window of MapForce to open the solution file. The "Click to open..." link is available for C# and C++. If you compile for Java you will have to navigate to the folder manually.



2. Select the menu option **Build | Build Solution** to compile the mapping project.



3. Select the menu option **Debug | Start Debugging** to start the application.



The mapping application is started and the target XML file is created.

23.3.3 Generating C++ code

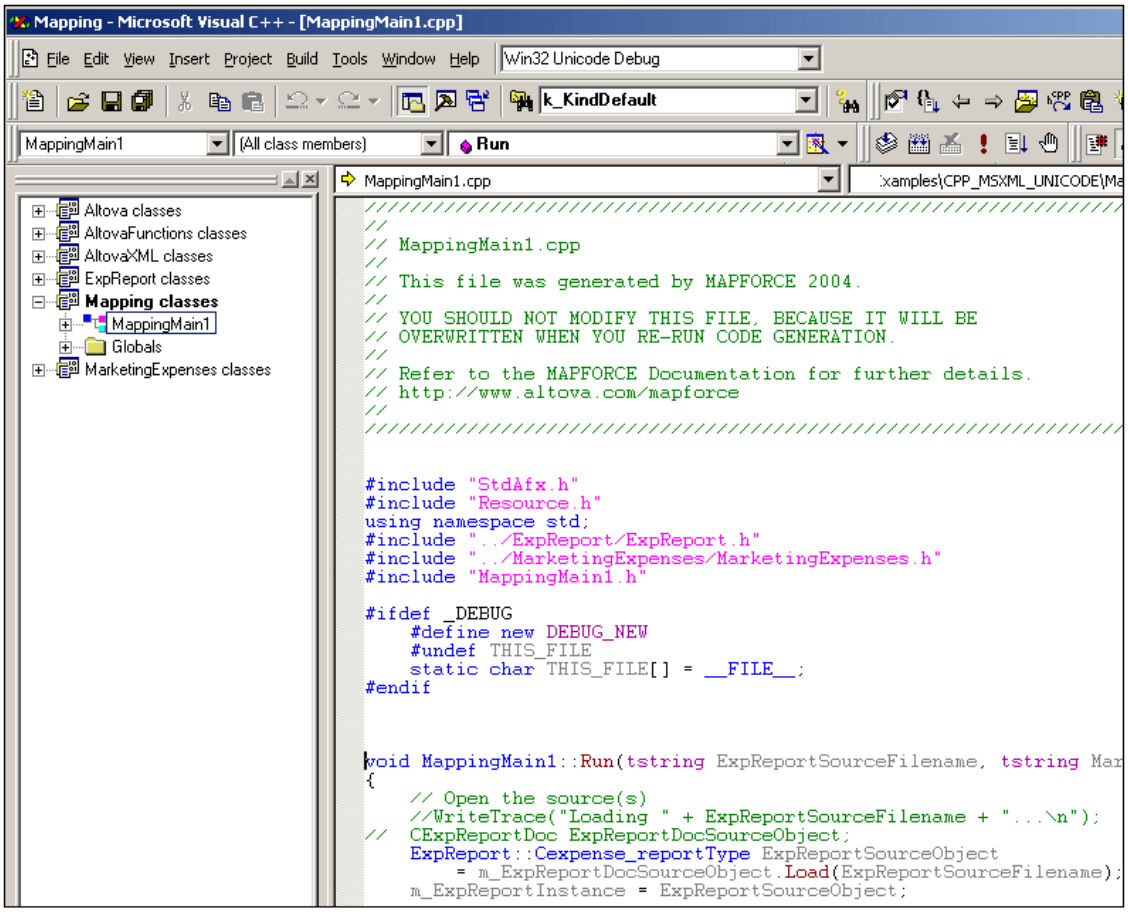
Prerequisites and default settings:

- C++ code generation for both MapForce and XMLSpy supports Visual C++ 6.0, / Visual Studio 2005, 2008 and 2010 directly.
- **.sln** and .**vcproj** files are generated in addition to the **.dsw** /**.dsp** files for Visual Studio 6.0. The **Generation** tab under menu option **Tools | Options** allows you to choose the target IDE and code generation settings.
- The menu option **File | Mapping settings** defines the mapping project settings. The default settings are: Application name=Mapping.

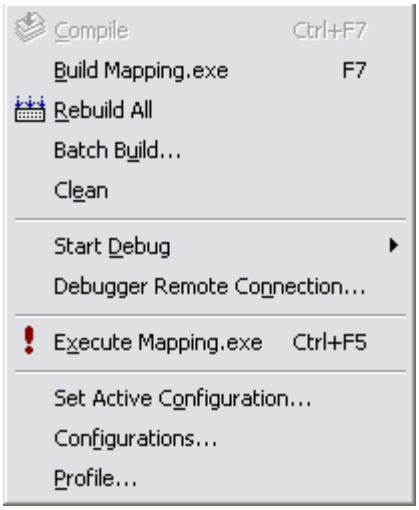
Database specfic settings can be viewed by clicking a database component, and selecting the menu option **Component | Properties**.

To generate C++ code in MapForce:

1. Select the menu option **File | Generate code in | C++**.
   You are then prompted for the target directory of the generated files.
2. Select the directory you want to place the files in, and click OK to confirm (eg. c:\codegen\C++).
   A "C++ Code generation completed" message appears when the process was successful. If not, an error message appears detailing the specific error(s).
3. Open the **mapping.dsw** or **mapping.sln** file in the **Mapping** subdirectory, i.e. c:\codegen\C++\mapping in Microsoft Visual C++.

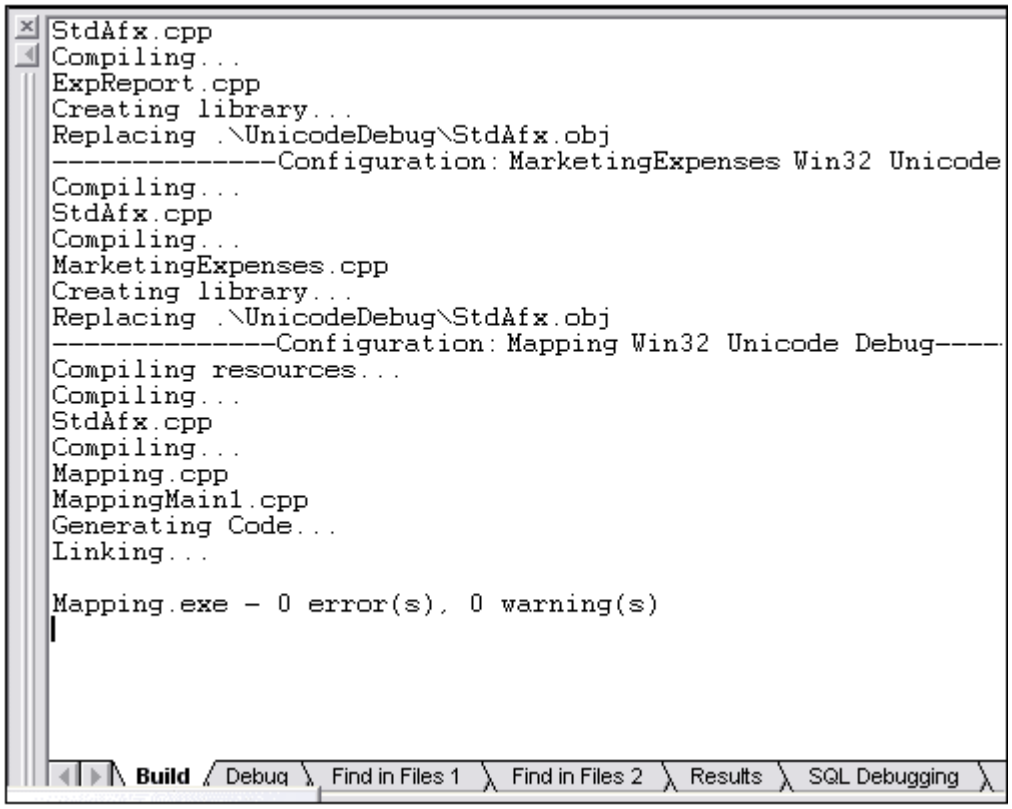


4. Select the menu option **Build | Build Mapping.exe.**

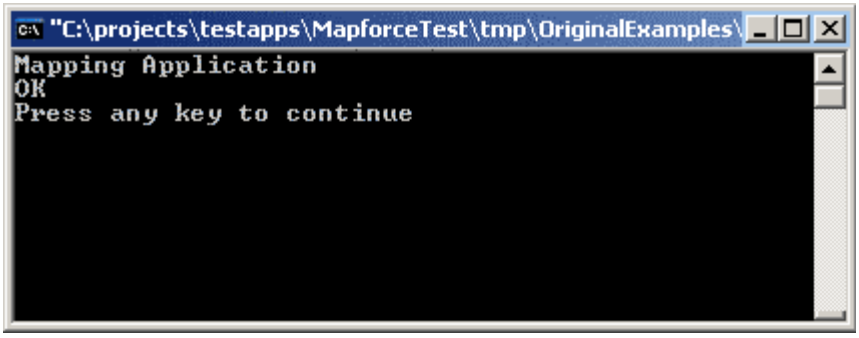


You can select from four different Build configurations. Please note that only Unicode builds will support the full Unicode character set in XML and other files. The non-Unicode builds will work with the local codepage of your Windows installation.

Debug:	Debug Unicode Debug NonUnicode
Release:	Release Unicode Release Non Unicode

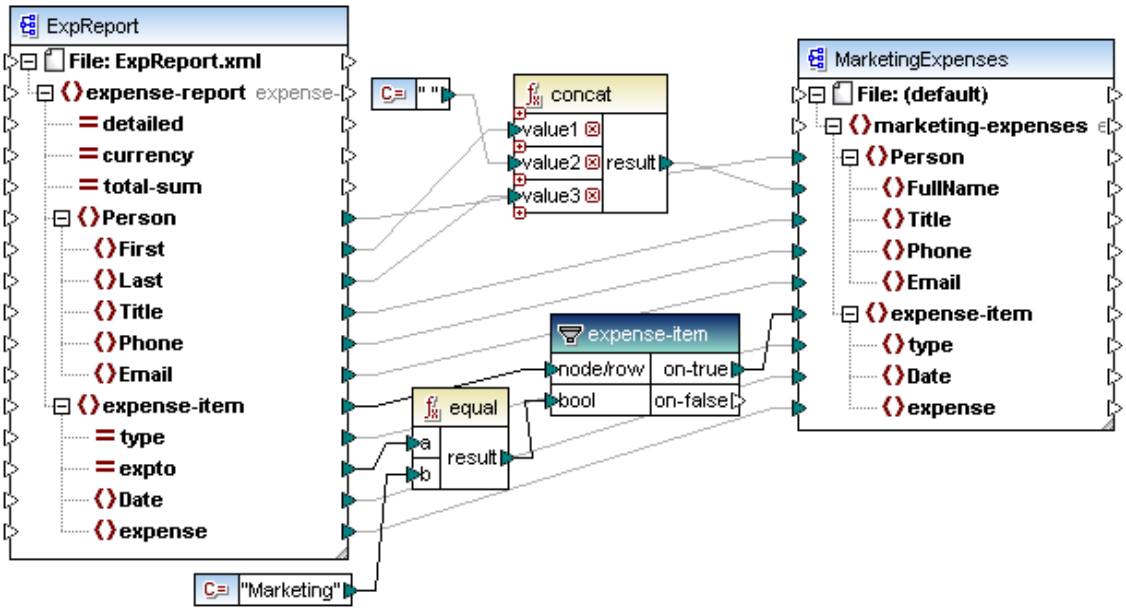


5. Once the code has been built, execute the **Mapping.exe** program to map your data.



23.4 Code generation mapping example

The mapping example shown below, uses the Marketing Expenses mapping project (MarketingExpenses.mfd) available in the **...\MapForceExamples** folder. Please also see Integrating code in your application for more information.



- The parameters to the **Run** functions in the code below shows the **source** file path, as well as the **target** XML file produced by the mapping. Multiple source files can appear, however, only one target file may be associated.

```
MappingMapToMarketingExpensesObject.run(
"C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml",
              "MarketingExpenses.xml" );
```

- If **multiple targets** exist, then the MappingMapToXXX section, shown below, is **repeated** for each target.

```
      MappingMapToMarketingExpenses MappingMapToMarketingExpensesObject = new
MappingMapToMarketingExpenses();
      MappingMapToMarketingExpensesObject.registerTraceTarget(ttc);
      MappingMapToMarketingExpensesObject.run(
          "C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml",
          "MarketingExpenses.xml"
      );
```

- Extra error handling code can be inserted under the Exception section:

```
catch (Exception e), or
catch (CAltovaException& e)
```

The code snippets below, are the mapping code generated for each specific programming language. They can be easily customized (for example, to accept file names from the command line) or integrated in other programs.

Java (com/mapforce/MappingConsole.java)

```
public static void main(String[] args) {
```

```
    System.out.println("Mapping Application");
    try { // Mapping
        TraceTargetConsole ttc = new TraceTargetConsole();
        MappingMapToMarketingExpenses MappingMapToMarketingExpensesObject = new
MappingMapToMarketingExpenses();
        MappingMapToMarketingExpensesObject.registerTraceTarget(ttc);
        MappingMapToMarketingExpensesObject.run(
            "C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml",
            "MarketingExpenses.xml"
        );
        System.out.println("Finished");
    }
    catch (com.altova.UserException ue) {
        System.out.print("USER EXCEPTION:");
        System.out.println( ue.getMessage() );
        System.exit(1);
    }
    catch (Exception e) {
        System.out.print("ERROR:");
        System.out.println( e.getMessage() );
        e.printStackTrace();
        System.exit(1);
    }
}
```

C# (Mapping/MappingConsole.cs)

```
public static void Main(string[] args)
{
    try
    {
        TraceTargetConsole ttc = new TraceTargetConsole();
        MappingMapToMarketingExpenses MappingMapToMarketingExpensesObject = new
MappingMapToMarketingExpenses();
        MappingMapToMarketingExpensesObject.RegisterTraceTarget(ttc);
        MappingMapToMarketingExpensesObject.Run(
            "C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml",
            "MarketingExpenses.xml");
        Console.Out.WriteLine("Finished");
    }
    catch (Altova.UserException ue)
    {
        Console.Out.Write("USER EXCEPTION: ");
        Console.Out.WriteLine( ue.Message );
        System.Environment.Exit(1);
    }
    catch (Exception e)
    {
        Console.Out.Write("ERROR: ");
        Console.Out.WriteLine( e.Message );
        Console.Out.WriteLine( e.StackTrace );
        System.Environment.Exit(1);
    }
}
```

C++ (Mapping/Mapping.cpp)

```
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
    tcout << _T("Mapping Application") << endl;

    try
    {
        CoInitialize(NULL);
        {
            MappingMapToMarketingExpenses MappingMapToMarketingExpensesObject;
```



```
            MappingMapToMarketingExpensesObject.Run(
                _T(
"C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml"),
                _T("MarketingExpenses.xml"));
        }
        CoUninitialize();

        tcout << _T("OK") << endl;
        return 0;
    }
    catch (CAltovaException& e)
    {
        if (e.IsUserException())
            tcerr << _T("User Exception: ");
        else
            tcerr << _T("Error: ");
        tcerr << e.GetInfo().c_str() << endl;
        return 1;
    }
    catch (_com_error& e)
    {
        tcerr << _T("COM-Error from ") << (TCHAR*)e.Source() << _T(":") <<
endl;
        tcerr << (TCHAR*)e.Description() << endl;
        return 1;
    }
    catch (std::exception& e)
    {
        cerr << "Exception: " << e.what() << endl;
        return 1;
    }
    catch (...)
    {
        tcerr << _T("Unknown error") << endl;
        return 1;
    }
}
```

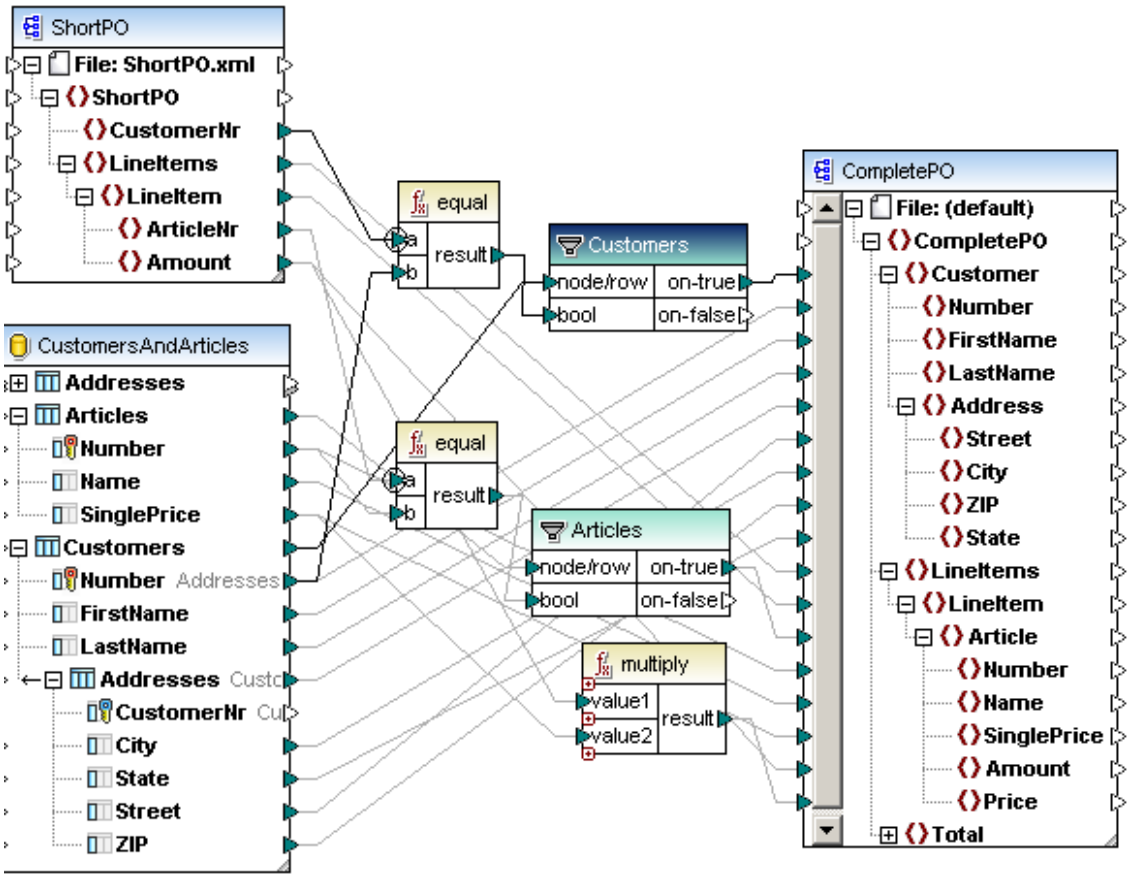
23.5 Integrating MapForce code in your application

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. The mapping project visible below, **DB_CompletePO.mfd**, is available in the **...\MapForceExamples** folder.

Please also see the section Generating program code for information on how the generated code is produced.

This section describes how to:

- **Modify** the source and target files of a mapping project
- Use an **XML input stream** as a data source, and
- Where to add your own **error handling** code



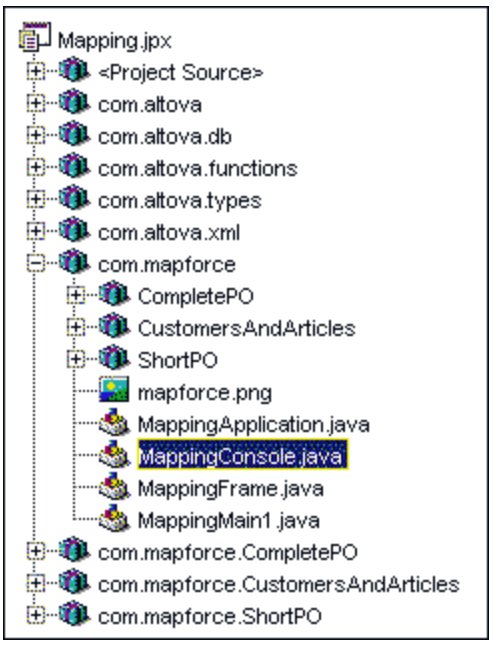
This example consists of two sources and one target:

- ShortPO.xml as a **source XML** file
- CustomersAndArticles.mdb as a **source database**, and
- CompletePO.xml as the **target XML** file.

23.5.1 MapForce code in Java applications

This example assumes that you are using **Borland JBuilder** as your Java environment. Having generated the Java code in MapForce, and defined a folder named "output" as the output folder,

1. Navigate to the output folder, and open the **Mapping.jpx** project file.
2. Double click the **MappingConsole.java** file.



A snippet of the code is shown below.

```
        {
            com.altova.io.Input ShortPO2Source = new com.altova.io.FileInput(
"ShortPO.xml");
            com.altova.io.Output CompletePO2Target = new
com.altova.io.FileOutput("CompletePO.xml");

            MappingMapToCompletePOObject.run(
                  java.sql.DriverManager.getConnection(
                     "jdbc:odbc:;DRIVER=Microsoft Access Driver
(*.mdb);DBQ=CustomersAndArticles.mdb",
                     "",
                     ""),
                  ShortPO2Source,
                  CompletePO2Target);
        }
```

Please note that the path names in the generated source code have been deleted for the sake of clarity.

Looking at **MappingMapToCompletePO.run**:
All parameters passed to the **run** method, **except for** the last one (CompletePO.xml), are source files. Please ensure that the last parameter is the target file.

In this example:

the **source** files are:

- XML file: ShortPO.xml

- Database: CustomerAndArticles.mdb including the connection string.
  The **two empty parameters** **""** following the initial database parameter, are intended for the **Username** and **Password** (in clear text) for those databases where this data is necessary.

the **target** file is:

- CompletePO.xml

To define your own source or target files:

- Edit the parameters passed to the FileInput and FileOutput constructors.

To use an XML input stream as the XML data source:

- Replace FileInput with StreamInput and pass a `java.io.InputStream` instead of a file name.

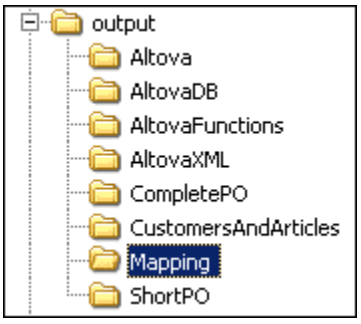
To add extra error handling code:

- Edit the code below the **`catch`** `(Exception ex)` code.

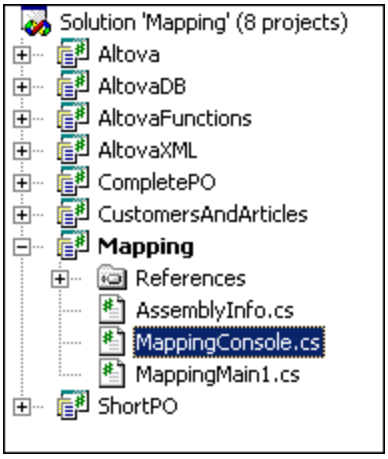
23.5.2 MapForce code in C# applications

Having generated the C# code in MapForce, and defined a folder named "output" as the output folder,

1. Navigate to the output**Mapping** folder, and open the **Mapping.sln** file in Visual Studio.



2. Double click the **MappingConsole.cs** file.



A snippet of the code is shown below.

```
            {
               Altova.IO.Input ShortPO2Source = new Altova.IO.FileInput(
"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);
            }
```

Please note that the path names in the generated source code have been deleted for the sake of clarity.

Looking at **MappingMapToCompletePO.Run**:
All parameters passed to the **Run** method, **except for** the last one (CompletePO.xml), are source files. Please ensure that the last parameter is the target file.

In this example:

the **source** files are:

- XML file: Short.PO.xml
- Database: CustomerAndArticles.mdb including the connection string

the **target** file is:

- CompletePO.xml

To define your own source or target files:

- Edit the parameters passed to the FileInput and FileOutput constructors.

To use an XML input stream as the XML data source:

- Replace FileInput with StreamInput and pass a `System.IO.Stream` instead of a file name.

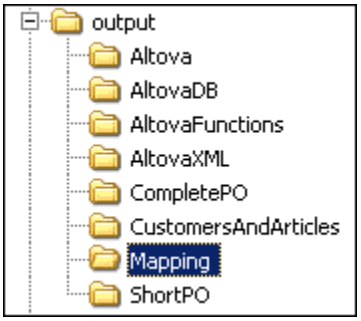
To add extra error handling code:

- Edit the code below the `catch (Exception e)` code.

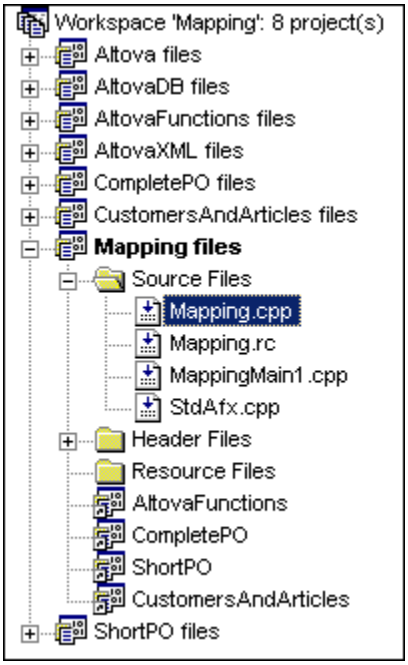
23.5.3 MapForce code in C++ applications

Having generated the C++ code in MapForce, and defined a folder named "output" as the output folder,

1. Navigate to the output**Mapping** folder, and open the workspace file **mapping.dsw** in MS Visual C++ 6.0 (or **mapping.sln** in later versions of Visual C++)



2. Double click the **Mapping.cpp** file to open the mapping project file.



A snippet of the code is shown below.

```
      CoInitialize(NULL);
      {
         MappingMapToCompletePO MappingMapToCompletePOObject;
         MappingMapToCompletePOObject.Run(
            _T("Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=CustomersAndArticles.mdb; "),
            _T("ShortPO.xml"),
            _T("CompletePO.xml"));
      }
      CoUninitialize();

      tcout << _T("Finshed") << endl;
      return 0;
```

Please note that the path names in the generated source code have been deleted for the sake

of clarity.

Looking at **MappingMapToCompletePO.Run**:
All parameters passed to the **Run** method, **except for** the last one (CompletePO.xml), are source files. Please ensure that the last parameter is the target file.

In this example:

the **source** files are:

- XML file: Short.PO.xml
- Database: CustomerAndArticles.mdb including the connection string

the **target** file is:

- CompletePO.xml

To define your own source or target files:

- Directly edit the parameters passed to the run method of MappingMapToCompletePOObject.

To use an XML input stream as the XML data source:

- Navigate to the run method declaration in the code, and configure the specific parameters there.

To add extra error handling code:

- Edit the code below the `catch (CAltovaException& e)` code.

23.5.4 Data Stream support

Starting with version 2008 release 2, generated code in C# and Java supports data streams as input/output, in addition to file-based input/output.

The most important function of generated mapping classes is the "run" method. 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.

Prior to MapForce version 2008 R2, the run method required file names and/or database connection objects as arguments. It is now also possible to provide input or output objects instead of file names.

The following sources and targets are supported in any combination:

- Files (identified by file names)
- Binary streams
- Character streams (readers/writers) - will ignore the character encoding set in the MapForce component
- Strings - will ignore the character encoding set in the MapForce component
- DOM documents (for XML only)

These objects are available in the **com.altova.io** package (Java) and **Altova.IO** namespaces (C#), and are always generated. Namespace/package prefixes have been omitted for greater clarity in the following section.

Input and output objects have base classes called **Input** and **Output** i.e., **com.altova.io.Input** and **com.altova.io.Output** for Java, and **Altova.IO.Input** and **Altova.IO.Output** for C#.

The "**text**" label used below represents any CSV or FLF file.

Note that Excel files can only be mapped to/from files, or binary streams, and are only available in MapForce Enterprise Edition.

The following wrapper objects are used as parameters of the "run" method:

Java:

files/file names (for XML, text, and Excel):

```
com.altova.io.FileInput(String filename)
com.altova.io.FileOutput(String filename)
```

streams (for XML, text, and Excel):

```
com.altova.io.StreamInput(java.io.InputStream stream)
com.altova.io.StreamOutput(java.io.OutputStream stream)
```

strings (for XML and text):

```
com.altova.io.StringInput(String xmlcontent)
com.altova.io.StringOutput()
```

Java IO reader/writer (for XML and text):

```
com.altova.io.ReaderInput(java.io.Reader reader)
com.altova.io.WriterOutput(java.io.Writer writer)
```

DOM documents (for XML only):

```
com.altova.io.DocumentInput(org.w3c.dom.Document document)
```

```
com.altova.io.DocumentOutput(org.w3c.dom.Document document)
```

C#:

files/file names (for XML, text, and Excel):

```
Altova.IO.FileInput(string filename)
Altova.IO.FileOutput(string filename)
```

streams (for XML, text, and Excel):

```
Altova.IO.StreamInput(System.IO.Stream stream)
Altova.IO.StreamOutput(System.IO.Stream stream)
```

strings (for XML and text):

```
Altova.IO.StringInput(string content)
Altova.IO.StringOutput(StringBuilder content)
```

Java IO reader/writer (for XML and text):

```
Altova.IO.ReaderInput(System.IO.TextReader reader)
Altova.IO.WriterOutput(System.IO.TextWriter writer)
```

DOM documents (for XML only):

```
Altova.IO.DocumentInput(System.Xml.XmlDocument document)
Altova.IO.DocumentOutput(System.Xml.XmlDocument document)
```

MapForce generates the run function which takes several **Inputs** and one **Output.**

Simple Example

We are using a Java (or C#) application, and want MapForce to generate mapping code which will be integrated into our application. E.g. the mapping consists of two source xml files and a target text file. MapForce generates the following run function:

```
void run(Input in1. Input in2, Output out1);
```

Let's assume that our application requires that we map data from a local file and binary stream into an character stream. The data is supplied from other sources, and we want to integrate the MapForce-generated code into our application. The application provides sources and targets as:

```
String filename;
Java.io.InputStream stream;
Java.io.Writer writer;
```

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);
```

The MapForce generated run function needs to be called at this point:

```
MappingMapToSomething.run(input1, input2, output1);
```

That's it.

The **C#** behavior is almost identical, except that "run" is called **Run**, and the .NET stream and reader/writer classes are named differently.

Other input / output types, such as strings or DOM documents, are used in the same manner;

the differences are noted below.

Streams Behavior

The following rules need to be observed for binary or character streams:

- Streams and Readers/Writers are expected to be opened and ready-to-use, before calling run.
- By default, the MapForce generated run function will close streams, readers/writers when finished. If you do not want this to happen before calling run function, insert:

  MappingMapToSomething.setCloseObjectsAfterRun(true); // Java

  or

  MappingMapToSomething.CloseObjectsAfterRun = false; // C#

- Excel sources/targets cannot be read from, or written to, character streams.

StringOutput behavior

There is a subtle difference between C# and Java StringOutput behavior.

In **Java**, StringOutput does not take an argument. Content can be accessed with:

```
// mapping from String to (another) String

String blah = "<here>is some xml text</here>";

Input input = new StringInput(blah);
Output output = new StringOutput();

MappingMapToBlah.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.

DOM Document behavior

If your application requires it, you can pass DocumentInput / DocumentOutput as arguments to "run".

Note that in target mode the document passed to the DocumentOutput constructor needs to be empty.

After calling "run" the DOM Document, provided/generated by the constructor of DocumentOutput, already contains mapped data so "save to document" is not necessary. After mapping, you may manipulate the document any way you want.

Only XML content can be mapped to DOM documents.

Databases

If a mapping includes database components, the run function will include the database connection object at the appropriate location.

E.g. If the mapping maps from Text content, XML content and data from a database to some output file, MapForce generates the following run function:

```
void run(Input in1, Input in2, java.sql.Connection dbConn, Output
out1);
```

The argument order is important here. You can modify dbConn parameters, or use the default parameters generated by MapForce when integrating your code.

23.6 Using the generated code library

Overview
Code generation in MapForce generates a complete application that executes all steps of the mapping automatically. If you intend to integrate the mapping code into your own application, you may want to 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.

If you chose to generate wrapper classes in the Options dialog, MapForce creates not only the mapping application for you, but additionally generates 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.

The library generated by MapForce 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.

Compatibility Mode
Depending on the value of the setting "Generate code compatible to", the generated classes have different names, members and methods. The following description applies to **version 2007r3 and later only**. For the other settings please refer to the chapter about using generated code compatible to old versions. There were no compatibility-breaking changes since version 2007r3.

Generated Libraries
When MapForce generates code from an XML Schema or DTD, the following libraries are generated:

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)

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.

23.6.1 Example schema

Using the generated classes
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.

23.6.2 Using the generated Java library

Please note: This topic describes code generation when using the option **Generate code compatible to V2007r3** in the code generation dialog.

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. Please note that the default DOM implementation delivered with Java 1.4 does not automatically create namespace attributes when serializing XML documents. If you encounter problems, please update to Java 5 or later.

Data Types

The default mapping of XML Schema types to Java data types is:

XML Schema	Java	Remarks
xs:string	String	
xs:boolean	boolean	
xs:decimal	java.math.BigDecimal	
xs:float, xs:double	double	
xs:integer	java.math.BigInteger	
xs:long	long	
xs:unsignedLong	java.math.BigInteger	Java does not have unsigned types.
xs:int	int	
xs:unsignedInt	long	Java does not have unsigned types.
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	com.altova.types.DateTime	
xs:duration	com.altova.types.Duration	
xs:hexBinary and xs:base64Binary	byte[]	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string	

All XML Schema types not contained in this list are derived types, and mapped to the same Java type as their respective base type.

Generated 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-8")	Saves an XML document to a byte array with the specified encoding. If no encoding is 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 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 created.

Method	Purpose
MemberType **first**()	Returns the first instance of the member element
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
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.
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.

Reading XML Documents

The following example reads the file we just created and demonstrates the various methods for reading XML files:

```
	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().get
Value());

                    // 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());
             }
      }
```

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
SourceInstanceUnavailableException	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
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 a list of all enumeration facets
String[] **getPatterns**()	Returns a list 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
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

23.6.3 Using the generated C++ library

Please note: This topic describes code generation when using the option **Generate code compatible to V2007r3** in the code generation dialog.

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
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	
xs:duration	altova::Duration	
xs:hexBinary and xs:base64Binary	std::vector<unsigned char>	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string_type	

All XML Schema types not contained in this list are derived types, and mapped to the same C++ type as their respective base type.

Generated 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.
string_type **SaveToString**(bool prettyPrint)	Saves an XML document to a string
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
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 are 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
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
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

23.6.4 Using the generated C# library

Please note: This topic describes code generation when using the option **Generate code compatible to V2007r3** in the code generation dialog.

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:

XML Schema	C#	Remarks
xs:string	string	
xs:boolean	bool	
xs:decimal	decimal	xs:decimal has unlimited range and precision, mapped to decimal for efficiency reasons.
xs:float, xs:double	double	
xs:long	long	
xs:unsignedLong	ulong	
xs:int	int	
xs:unsignedInt	uint	
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	Altova.Types.DateTime	
xs:duration	Altova.Types.Duration	
xs:hexBinary and xs:base64Binary	byte[]	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string	

All XML Schema types not contained in this list are derived types, and mapped to the same C# type as their respective base type.

Generated 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.
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 encoding)	Creates a new, empty XML document, with 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
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

23.6.5 Using generated code compatible to old versions

Compatibility Mode
The code generator has been improved multiple times in the recent releases of Altova products, sometimes in ways that require changes to the code that uses the generated libraries. To preserve compatibility with existing code using libraries generated with earlier releases of MapForce, you can set the compatibility mode to the desired release. Default is version 2007r3, which creates the latest version of the code libraries described in the previous chapters.

The following chapters describe usage of the generated classes when using one of the following compatibility options:

- Code compatible to version 2005r3
- Code compatible to version 2006 to 2007

Please note that these compatibility options will be removed in a future version.

Version 2005r3:
Code generated code for XML schemas up till V2005R3 uses a static DOM document instance as parent for all nodes. This allows creating free-standing nodes that can be added to a document later, but may rise issues with multi-threading or memory management.

```
{
  LibraryType lib=new LibraryType();
   BookType book = new BookType();
   book.addISBN( new SchemaString( "0764549642" ) );
   book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
   book.addAuthor( new SchemaString( "Altova" ) );
   lib.addBook( book );

   LibraryDoc doc = new LibraryDoc();
   doc.setRootElementName( "http://www.nanonull.com/LibrarySample", "Library" );
  doc.setSchemaLocation( "Library.xsd" ); //optional
  doc.save( "Library1.xml", lib );
}
```

Version 2006 to 2007:
In C# and Java code generated with version 2006 to 2007, the standard (non-parameter) constructor is not used, because it does not establish a reference to a DOM document. A new document is created first which can then be referenced by the root node. All new child nodes must then also be created in the correct document. This is accomplished by using the generated factory functions called newXXX, please see the example code below.

```
{
  LibraryDoc doc = new LibraryDoc();
  doc.setSchemaLocation( "Library.xsd" ); //optional

     // create root element (with no namespace prefix)
     LibraryType lib = new LibraryType(doc, "
http://www.nanonull.com/LibrarySample", "", "Library" );

     BookType book = lib.newBookType();    // factory functions are
generated for all members of a complex type
     book.addISBN( new SchemaString( "0764549642" ) );
     book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
     book.addAuthor( new SchemaString( "Altova" ) );
     lib.addBook( book );
```



```
    doc.save( "Library1.xml", lib );
}
```

Version 2007r3:
The code generated by version 2007r3 has been completely redesigned to allow for multi-threading, avoid name collisions, and simplify the usage of the libraries. For details, see Using the generated code library.

Creating XML files (XMLSpy 2006)

Please note: This topic describes code generation when using the option **Generate code compatible to V2006-2007** in the code generation dialog.

Java:

To give a general overview, here is the code to create a test file:

```
protected static void example() throws Exception {
        LibraryDoc doc = new LibraryDoc();
        doc.setSchemaLocation("Library.xsd");  // optional
        LibraryType lib = new LibraryType(doc,
"http://www.nanonull.com/LibrarySample", "", "Library" );
        BookType book = lib.newBook();
        book.addISBN( new SchemaString( "0764549642" ) );
        book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
        book.addAuthor( new SchemaString( "Altova" ) );
        lib.addBook( book );
        doc.save("Library1.xml", lib);
}
```

The idea of the generated classes is the following:
Create a document and associate it with the root element of the generated schema so that the tree is stored in the document.

First we create a new document:

```
LibraryDoc doc = new LibraryDoc();
```

Optionally we can also set the schema location.

```
doc.setSchemaLocation( "Library.xsd" );
```

We create the root element, passing to the method the document object that we have already generated, the namespace of `library.xsd`, and the name of the root element in the schema:

```
LibraryType lib = new LibraryType(doc,
"http://www.nanonull.com/LibrarySample", "", "Library" );
```

LibraryType is a class that was generated. It is similar to the <xs:element name="Library"> in the schema file. This is the root element for the xml files being generated (as in generate sample xml file).

We now have a new Library object. All we have to do is fill it with Book objects. Look at the schema again and notice that books are stored as a sequence within the Library.

The next step is to make an object of **BookType** and fill the new object.

```
BookType book = lib.newBook();
```

Definition of BookType in the schema file:

```
<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:complexType>
```

We will insert an ISBN number, a title and an author.

```
book.addISBN( new SchemaString( "0764549642" ) );
book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
book.addAuthor( new SchemaString( "Altova" ) );
```

We filled the BookType object, and we now have to add it to the library sequence:

```
lib.addBook(book);
```

Finally, save the file. The first parameter is the file name, the second is the root element itself.

```
doc.save( "Library1.xml", lib );
```

C++:

The C++ implementation is very similar to the Java one, we will therefore only discuss the differences between the two.

The function to be inserted above the main function is as follows:

```
void Example()
{
    CLibraryDoc doc;
    CLibraryType lib;
    doc.SetRootElementName(_T("http://www.nanonull.com/LibrarySample"), _T(
"Library"));
    doc.SetSchemaLocation(_T("Library.xsd")); // optional
    CBookType book;
    book.AddISBN( _T("0764549642") );
    book.AddTitle( _T("The XML Spy Handbook") );
    book.AddAuthor( _T("Altova") );
    lib.AddBook( book );
    doc.Save(_T("Library1.xml"), lib);
}
```

First off, you'll notice that all classes begin with a capital C - the rest of the class naming stays the same. All member function names start with a capital letter.

Please take care with the **_T** macros. The macro ensures that the string constant is stored correctly, whether you're compiling for Unicode or non-Unicode programs.

C#:


```
protected static void Example()
{
	LibraryDoc doc = new LibraryDoc();
	doc.SetSchemaLocation( "Library.xsd" ); // optional
	// create root element with no namespace prefix
	LibraryType lib = new LibraryType(doc, "
http://www.nanonull.com/LibrarySample", "", "Library" );
	BookType book = lib.NewBook();		// factory functions are generated for all
members of a complex type
	book.AddISBN( new SchemaString( "0764549642" ) );
	book.AddTitle( new SchemaString( "The XML Spy Handbook" ) );
	book.AddAuthor( new SchemaString( "Altova" ) );
	lib.AddBook( book );
	doc.Save( "Library1.xml", lib );
}
```

Creating XML files (XMLSpy 2005)

Please note: This topic describes code generation when using the option **Generate code compatible to V2005R3** in the code generation dialog.

Java:

To give a general overview, here is the code to create a test file:

```
protected static void example() {
	LibraryType lib = new LibraryType();
	BookType book = new BookType();
	book.addISBN( new SchemaString( "0764549642" ) );
	book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
	book.addAuthor( new SchemaString( "Altova" ) );
	lib.addBook( book );

	LibraryDoc doc = new LibraryDoc();
	doc.setRootElementName( "http://www.nanonull.com/LibrarySample", "Library"
);
	doc.setSchemaLocation( "Library.xsd" ); // optional
	doc.save( "Library1.xml", lib );
}
```

The idea of the generated classes is the following:
Generate the tree you want to save in the new file, create a document and store the tree in it.

The first line:

```
LibraryType lib = new LibraryType();
```

LibraryType is a class that was generated. It is similar to the <xs:element name="Library"> in the schema file. This is the root element for the xml files being generated (as in generate sample xml file).

We now have a new Library object. All we have to do is fill it with Book objects. Look at the schema again and notice that books are stored as a sequence within the Library.

The next step is to make an object of **BookType** and fill the new object.

```
BookType book = new BookType();
```

Definition of BookType in the schema file:

```
<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:complexType>
```

We will insert an ISBN number, a title and an author.

```
book.addISBN( new SchemaString( "0764549642" ) );
book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
book.addAuthor( new SchemaString( "Altova" ) );
```

We filled the BookType object, and we now have to add it to the sequence of **internationalPriceTypes** within Test_Empty:

```
lib.addBook(book);
```

Done, the tree is complete! All we have to do now, is create a new document and define the namespace and name of the root element:

```
LibraryDoc doc = new LibraryDoc();
doc.setRootElementName( "http://www.nanonull.com/LibrarySample", "Library"
);
```

Optionally we can also set the schema location.

```
doc.setSchemaLocation( "Library.xsd" );
```

Finally, save the file, - the first parameter is the file name, the second is the root element itself.

```
doc.save( "Library1.xml", lib );
```

C++:

The C++ implementation is very similar to the Java one, we will therefore only discuss the differences between the two.

The function to be inserted above the main function is as follows:

```
void Example()
{
    CLibraryType lib;
    CBookType book;
    book.AddISBN( _T("0764549642") );
    book.AddTitle( _T("The XML Spy Handbook") );
    book.AddAuthor( _T("Altova") );
    lib.AddBook( book );

    CLibraryDoc doc;
    doc.SetRootElementName( _T("http://www.nanonull.com/LibrarySample"), _T(
"Library") );
    doc.SetSchemaLocation( _T("Library.xsd") ); // optional
```



```
        doc.Save( _T("Library1.xml"), lib );
}
```

First off, you'll notice that all classes begin with a capital C - the rest of the class naming stays the same. All member function names start with a capital letter.

Please take care with the **_T** macros. The macro ensures that the string constant, is stored correctly, whether you're compiling for Unicode or non Unicode programs.

C#:

```
protected static void Example()
{
        LibraryType lib = new LibraryType();
        BookType book = new BookType();
        book.AddISBN( new SchemaString( "0764549642" ) );
        book.AddTitle( new SchemaString( "The XML Spy Handbook" ) );
        book.AddAuthor( new SchemaString( "Altova" ) );
        lib.AddBook( book );

        LibraryDoc doc = new LibraryDoc();
        doc.SetRootElementName( "http://www.nanonull.com/LibrarySample", "Library"
);
        doc.SetSchemaLocation( "Library.xsd" ); // optional
        doc.Save( "Library1.xml", lib );
}
```

Opening and parsing existing XML files (XMLSpy 2006)

Please note: This topic describes code generation when using the option **Generate code compatible to V2006-2007** in the code generation dialog.

We will now use the **Library1.xml** file generated with our sample program. Please make sure that this file exists, or change the path to it in the load command.

Java:

Sample source code:

```
protected static void example2() {

        LibraryDoc doc = new LibraryDoc();
        LibraryType lib = new LibraryType(doc.load("Library1.xml"));

        if (!lib.hasBook()) {
                System.out.println("This library is empty");
                return;
        }

        for (int i = 0; i < lib.getBookCount(); i++) {
                BookType book;
                try {
                        book = lib.getBookAt(i);

                        System.out.println("ISBN:   " + book.getISBN().toString());
                        System.out.println("Title:  " + book.getTitle().toString());
```

```
                        for (int j = 0; j < book.getAuthorCount(); j++) {
                                System.out.println("Author: "+
book.getAuthorAt(j).toString());
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

in the ***main()*** function add a call to:

```
        example2();
```

after

```
        example1();
```

The rest is reasonably straightforward.
First create a new variable of type Document, and one for the root element, which is immediately filled by the **doc.load** function. The only parameter it needs, is the path to the xml file being used.

```
        LibraryDoc doc = new LibraryDoc();
        LibraryType lib = new LibraryType( doc.load( "Library1.xml" ) );
```

We now have a document and know that lib contains a sequence of BookType objects.

The next step is to dump the values of all of the Book elements:

```
        if( !lib.hasBook() ) {
```

Why check for objects? The call just shows that a function exists that checks this for you.

Step through the books and output their values:

```
        for( int i = 0; i < lib.getBookCount(); i++ ) {
                BookType book = lib.getBookAt( i );

                System.out.println( "ISBN:   " + book.getISBN().asString() );
                System.out.println( "Title:  " + book.getTitle().asString() );

                for( int j = 0; j < book.getAuthorCount(); j++ ) {
                        System.out.println( "Author: " + book.getAuthorAt( j ).asString()
);
                }
        }
```

We're finished, all the values are listed in your output window.

C++:

This section only explains the differences to the java code.

```
void Example2()
{
        CLibraryDoc doc;
        CLibraryType lib = doc.Load( _T("Library1.xml") );

        if ( !lib.HasBook() )
```



```
        {
        cout << "This library is empty" << endl;
        return;
        }

        for ( int i = 0; i < lib.GetBookCount(); i++ )
        {
                CBookType book = lib.GetBookAt( i );
                cout << "ISBN:   " << book.GetISBN() << endl;
                cout << "Title:  " << book.GetTitle() << endl;

                for ( int j = 0; j < book.GetAuthorCount(); j++ )
                {
                        cout << "Author: " << book.GetAuthorAt( j ) << endl;
                }
        }
}
```

All member functions start with a capital letter, and all class names start with a C.

C#:

This section only explains the differences to the java code.

```
protected static void Example2()
{
        LibraryDoc doc = new LibraryDoc();
        LibraryType lib = new LibraryType( doc.Load( "Library1.xml" ) );

        if ( !lib.HasBook() )
        {
                Console.WriteLine( "This library is empty" );
                return;
        }

        for ( int i = 0; i < lib.GetBookCount(); i++ )
        {
                BookType book = lib.GetBookAt( i );
                Console.WriteLine( "ISBN:   {0}", book.GetISBN() );
                Console.WriteLine( "Title:  {0}", book.GetTitle() );

                for( int j = 0; j < book.GetAuthorCount(); j++ )
                {
                        Console.WriteLine( "Author: {0}", book.GetAuthorAt( j ) );
                }
        }
}
```

All member functions start with a capital letter.

Opening and parsing existing XML files (XMLSpy 2005)

Please note: This topic describes code generation when using the option **Generate code compatible to V2005R3** in the code generation dialog.

We will now use the **Library1.xml** file generated with our sample program. Please make sure that this file exists, or change the path to it in the load command.

Java:

Sample source code:

```
protected static void example2() {
        LibraryDoc doc = new LibraryDoc();
        LibraryType lib = new LibraryType( doc.load( "Library1.xml" ) );

if( !lib.hasBook() ) {
        System.out.println( "This library is empty" );
        return;
}

for( int i = 0; i < lib.getBookCount(); i++ ) {
        BookType book = lib.getBookAt( i );
        System.out.println( "ISBN: " + book.getISBN().asString() );
        System.out.println( "Title: " + book.getTitle().asString() );

                for( int j = 0; j < book.getAuthorCount(); j++ ) {
                        System.out.println( "Author: " + book.getAuthorAt( j
).asString());
                }
        }
}
```

in the ***main()*** function add a call to:

```
example2();
```

after

```
example1();
```

The rest is reasonably straightforward.
First create a new variable of type Document, and one for the root element, which is immediately filled by the **doc.load** function. The only parameter it needs, is the path to the xml file being used.

```
LibraryDoc doc = new LibraryDoc();
LibraryType lib = new LibraryType( doc.load( "Library1.xml" ) );
```

We now have a document and know that lib contains a sequence of BookType objects.

The next step is to dump the values of all of the Book elements:

```
if( !lib.hasBook() ) {
```

Why check for objects? The call just shows that a function exists, that checks this for you.

Step through the books and dump their values:

```
for( int i = 0; i < lib.getBookCount(); i++ ) {
        BookType book = lib.getBookAt( i );

        System.out.println( "ISBN:   " + book.getISBN().asString() );
        System.out.println( "Title:  " + book.getTitle().asString() );
```



```
                for( int j = 0; j < book.getAuthorCount(); j++ ) {
                        System.out.println( "Author: " + book.getAuthorAt( j ).asString()
);
                }
        }
```

We're finished, all the values are listed in your output window.

C++:

This section only explains the differences to the java code.

```
void Example2()
{
        CLibraryDoc doc;
        CLibraryType lib = doc.Load( _T("Library1.xml") );

        if( !lib.HasBook() )
        {
                cout << "This library is empty" << endl;
                return;
        }

        for( int i = 0; i < lib.GetBookCount(); i++ )
        {
                CBookType book = lib.GetBookAt( i );

                cout << "ISBN:   " << book.GetISBN() << endl;
                cout << "Title:  " << book.GetTitle() << endl;

                for( int j = 0; j < book.GetAuthorCount(); j++ )
                {
                        cout << "Author: " << book.GetAuthorAt( j ) << endl;
                }
        }
}
```

All member functions start with a capital letter, and all Classnames start with a C.

C#:

This section only explains the differences to the java code.

```
protected static void Example2()
{
        LibraryDoc doc = new LibraryDoc();
        LibraryType lib = new LibraryType( doc.Load( "Library1.xml" ) );

        if( !lib.HasBook() )
        {
                Console.WriteLine( "This library is empty" );
                return;
        }

        for( int i = 0; i < lib.GetBookCount(); i++ )
        {
```

```
                    BookType book = lib.GetBookAt( i );

                    Console.WriteLine( "ISBN:   {0}", book.GetISBN() );
                    Console.WriteLine( "Title:  {0}", book.GetTitle() );

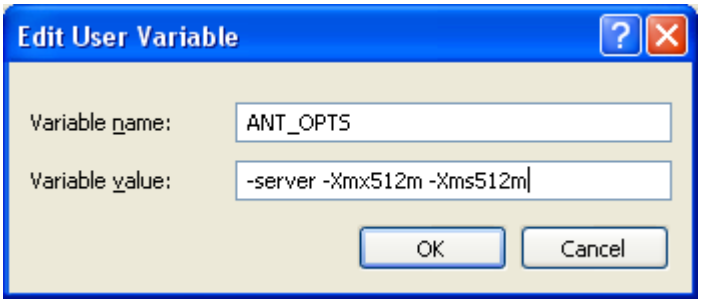
                    for( int j = 0; j < book.GetAuthorCount(); j++ )
                    {
                            Console.WriteLine( "Author: {0}", book.GetAuthorAt( j ) );
                    }
            }
    }
```

All member functions start with a capital letter.

23.7 Code generation tips

Out-of-memory exceptions during Java compilation and how to resolve them
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 please see your Java VM documentation.

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\ MapForce2011\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

23.8 Code generator options

The menu option **Tools | Options** lets you specify general as well as specific MapForce settings.

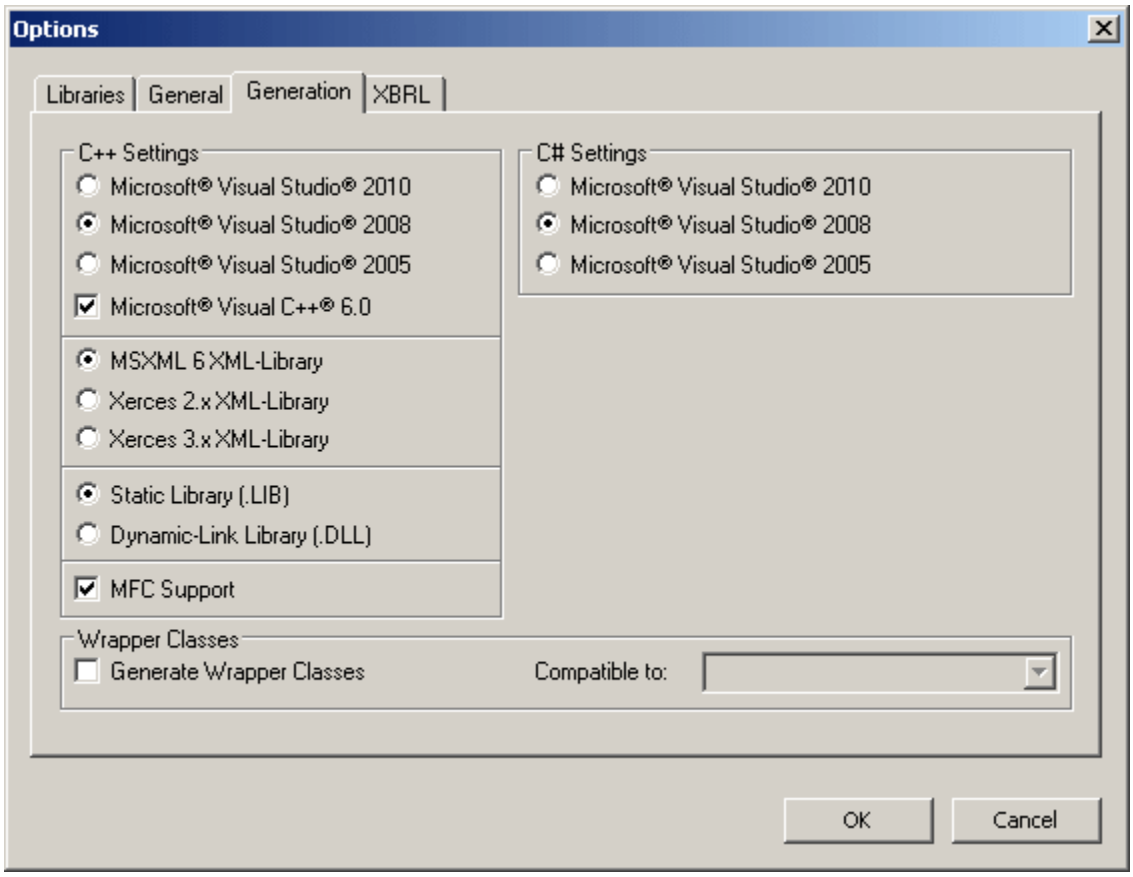
Generation tab:
This tab lets you define the settings for C++ and C# programming languages.

C++

- Microsoft Visual Studio 2010, 2008, 2005, Visual C++ 6.0 project file.
- generate code using either MSXML 6.0 or Apache Xerces XML library
- generate static libraries, or Dynamic-link libraries
- generate code with or without MFC support

C#
Select the type of project file you want to generate:

- Microsoft Visual Studio 2010, 2008, 2005, project file
- Mono Makefile support was removed in MapForce version 2011. Mono users can use Visual Studio solution files with Mono's xbuild.



Wrapper Classes:
MapForce starting with version 2007r3 uses completely new code generator templates that no longer depend on wrapper classes to be generated for the source and target schemas, database structures, EDI and text structures. This has the advantage of smaller code that is faster to compile, uses less memory, and executes faster. However, if you want to integrate the generated MapForce mapping into your own program, and you want to access the source

and/or target XML instances in your program using wrapper classes like those generated by XMLSpy, you can still enable the generation of these classes here.

Compatibility Mode:
The code generator has been improved multiple times in the recent releases of Altova products, sometimes in ways that require changes to the code that uses the generated libraries. To preserve compatibility with existing code using libraries generated with earlier releases of MapForce, you can set the compatibility mode to the desired release. Default is version 2007r3, which creates the latest version of the code libraries.

- Please make it a point to use the new code generation features i.e., set the version to the latest available.
- The availability of the Compatibility Mode is only temporary, it will be removed in a future version.

23.9 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 **...**MapForce2011**\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.

23.9.1 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 **]**.

23.9.2 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.

23.9.3 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.

23.9.4 Predefined variables

After a Schema file is analyzed by the code generator, the objects in the table below exist in the Template Engine.

Current object model (for code compatibility to version 2007r3 and later)

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

Old object model (for code compatibility up to version 2007)

Name	Type	Description
$schematype	integer	1 for DTD, 2 for XML Schema
$namespaces	Namespace collection	Collection of Namespace objects
$classes	Class collection	All the complex types, elements,… in a flat view
$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++ 6.0 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

23.9.5 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"
```

23.9.6 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

23.9.7 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.

23.9.8 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]
```

Collection manipulation routines:

collection **SortByName**(bAscending)
returns a collection whose elements are sorted by name (case sensitive) in ascending or descending order.

collection **SortByNameNoCase**(bAscending)
returns a collection whose elements are sorted by name (case insensitive) in ascending or descending order

Example:
```
$SortedNestedClassifier = $Class.nestedClassifier.SortByNameNoCase( true )
```

collection **SortByKind**(bAscending)
returns a collection whose elements are sorted by kind names (e.g. “Class”, “Interface”,…) in ascending or descending order.

collection **SortByKindAndName**(bAscendingKind, bAscendingName)
returns a collection whose elements are sorted by kind (e.g. “Class”, “Interface”,…) in ascending or descending order and if the kinds are equal by name (case sensitive in ascending or descending order)

collection **SortByKindAndNameNoCase**(bAscending)
returns a collection whose elements are sorted by kind (e.g. “Class”, “Interface”,…) in ascending or descending order and if the kinds are equal by name (case insensitive in ascending or descending order)

23.9.9 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)

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
]
```

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")
```

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
]
```

23.9.10 Built in Types

The section describes the properties of the built-in types used in the predefined variables which describe the parsed schema.

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)

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

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 colleciton	Child elements contained in this type
IsSimpleType	boolean	True for simple types, false for complex types
IsDerived	boolean	True if this type is derived from another type, which is also represented by a Type object
IsDerivedByExtension	boolean	True if this type is derived by extension

IsDerivedByRestriction	boolean	True if this type is derived by restriction
IsDerivedByUnion	boolean	True if this type is derived by union
IsDerivedByList	boolean	True if this type is derived by list
BaseType	Type	The base type of this type (if IsDerived is true)
IsDocumentRootType	boolean	True if this type represents the document itself
Library	Library	Library containing this type
IsFinal	boolean	True if declared as final in the schema
IsMixed	boolean	True if this type can have mixed content
IsAbstract	boolean	True if this type is declared as abstract
IsGlobal	boolean	True if this type is declared globally in the schema
IsAnonymous	boolean	True if this type is declared locally in an element

For simple types only:

Property	Type	Description
IsNativeBound	boolean	True if native type binding exists
NativeBinding	NativeBinding	Native binding for this type
Facets	Facets	Facets of this type
Whitespace	string	Shortcut to the Whitespace facet

* Complex types with text content (these are types with mixed content and complexType with simpleContent) have an additional unnamed attribute member that represents the text content.

Member

This object represents an attribute or element in the XML Schema. It is used to create class members of types.

Property	Type	Description
CodeName	string	Name for generated code (derived from local name or parent declaration)
LocalName	string	Original name in the schema. Empty for the special member representing text content of complex types.
NamespaceURI	string	The namespace URI of this Element/Attribute within XML instance documents/streams.
DeclaringType	Type	Type originally declaring the member (equal to ContainingType for non-inherited members)
ContainingType	Type	Type where this is a member of
DataType	Type	Data type of this member's content
Library	Library	Library containing this member's DataType
IsAttribute	boolean	True for attributes, false for elements
IsOptional	boolean	True if minOccurs = 0 or optional attribute
IsRequired	boolean	True if minOccurs > 0 or required attribute
IsFixed	boolean	True for fixed attributes, value is in Default property
IsDefault	boolean	True for attributes with default value, value is in Default property

IsNillable	boolean	True for nillable elements
IsUseQualified	boolean	True if NamespaceURI is not empty
MinOccurs	integer	minOccurs, as in schema. 1 for required attributes
MaxOccurs	integer	maxOccurs, as in schema. 0 for prohibited attributes, -1 for unbounded
Default	string	Default value

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

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

Old object model (up to v2007)

The following objects are part of the old object model for code compatibility up to version 2007. They will be removed in a future release.

Namespace

Namespace abstraction. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
URI	string	The URI of this namespace
Prefix	string	The prefix of this namespace
ContainsPublicClasses	boolean	True if the Classes collection contains at least one non-internal Class object
Classes	Class collection	Collection of the classes in this namespace - step through them using **foreach**

Class

For every user-defined type declared in the schema (xsd) a class is generated. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
HasNamespace	boolean	True, if there is an associated Namespace object
Namespace	Namespace	The Namespace object this Class object is part of
NamespacePrefix	string	Prefix of the namespace in which the class is
NamespaceURI	string	URI of the namespace in which the class is
Name	string	Name of the type in the resulting file
SchemaName	string	Name of the type as in the original schema file
HasBaseObject	boolean	True if this type is derived from another type, which is also represented by a Class object.
BaseObject	Class	The base Class object if HasBaseObject is true
BaseNamespaceURI	string	Namespace URI of the base class
Base	string	Name of the base class
SchemaBase	string	Name of the base type as in the original schema file
BuiltInBase	string	Name of the root simple type
BuiltInSchemaBase	string	Name of the root simple type as in the original schema file
IsAbstract	boolean	True if this is an abstract type
IsSimpleType	boolean	True if this is a simple type
IsComplexFromSimpleType	boolean	True if this is a complex type and is derived from a simple type
IsComplexType	boolean	True if this is a complex type
IsSequence	boolean	True if the top-level group-type is "sequence"
IsChoice	boolean	True if the top-level group-type is "choice"
IsAll	boolean	True if the top-level group-type is "all"
Description	string	Description of this type. May contain line breaks.
IsGlobal	boolean	True if this type is declared globally in the schema
IsAnonymous	boolean	True if this type is declared locally in an element

IsInternal	boolean	True if this is the internal root pseudo-class that contains all global classes as members
Members	Member collection	A class representing a complexType contains one or more Members.
Facets	Facet collection	A class representing a simpleType or a complexType derived from a simpleType may contain Facets.

Member

Abstraction of a member-variable inside a class. Members are created for all children of a complex type. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
NamespaceURI	string	The namespace URI of this Element/Attribute within XML instance documents/streams.
Name	string	Name in the resulting file
SchemaName	string	Name as in the original schema file
XmlName	string	The name as it is expected to appear in XML instance documents/streams.
Type	string	The name of the class which represents the schema type
SchemaType	string	The schema type
TypeObject	Class	Generated class for this type. See explanation below *
HasTypeObject	boolean	True if TypeObject has a valid value
Description	string	Description of the Element/Attribute. Can contain line feeds.
IsBuiltinType	boolean	True if the type is a built-in schema type, e.g. string, unsignedInt, dateTime... It is the negation of HasTypeObject.
IsSimpleType	boolean	True if the type of this member is a simpleType
IsComplexFromSimpleType	boolean	True if the type of this member is a complex type and is derived from a simple type.
IsElement	boolean	True if this is an element
IsAttribute	boolean	True if this is an attribute
NodeType	string	"Element" or "Attribute"
MinOcc	integer	minOccurs, as in schema. 1 for required attributes
MaxOcc	integer	maxOccurs, as in schema. 0 for prohibited attributes
IsQualified	boolean	True if the form of the Element/Attribute is set to "qualified".
IsMixed	boolean	True if this element can have mixed content
HasTextValue	boolean	True if this member can contain text
Default	string	The default value defined in the schema
Fixed	string	The fixed value defined in the schema

*** TypeObject:**
For every type declared in the schema (xsd) a class is generated. All elements inside a complexType are generated as members of such classes. The types of these members can either be built-in types or user-defined types:

Built-in XML Schema types have no TypeObject, because they are mapped to predefined classes using the map type instruction.

For each member with a user-defined type, the TypeObject represents the class generated for this type.

Example:

```
<xs:complexType name="TypeA">
        <xs:sequence>
                <xs:element name="B" type="TypeB"/>
        </xs:sequence>
</xs:complexType>
<xs:complexType name="TypeB"/>
```

CodeGen would create two classes:

```
class TypeA
{
   ...
   TypeB  B;
}

class TypeB
{
   ...
}
```

Explanation:
When class "TypeA" is generated the member B is also generated. To find out the type of B, use the "TypeObject" method of the element which returns the class "TypeB". So we can get the name and use it there.

Facet

This class consists of the constraint itself, and several boolean variables. This object is part of the old object model for code compatibility up to version 2007.

The boolean variables tell you the kind of constraint you're currently dealing with. The names of the Boolean variables are identical to those used in the schema specification.

Property	Type	Description
Constraint	string	Holds the value of the facet
IsLength	boolean	
IsMinLength	boolean	
IsMaxLength	boolean	
IsMinInclusive	boolean	
IsMinExclusive	boolean	
IsMaxInclusive	boolean	
IsMaxExclusive	boolean	
IsTotalDigits	boolean	
IsFractionDigits	boolean	
IsWhiteSpace	boolean	
IsPattern	boolean	
IsEnumeration	boolean	
Enumeration	Enumeration collection	Holds a collection of Enumeration objects if this facet is of type enumeration.

Pattern	Pattern collection	Holds a collection of Enumeration objects if this facet is of type pattern.

Enumeration

Abstraction of an enumeration entry inside a facet. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
Index	integer	Holds the index of this enumeration value, starting with 0
Value	string	Holds an enumeration value

Pattern

Abstraction of a pattern entry inside a facet. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
Index	integer	Holds the index of this pattern value, starting with 0
Value	string	Holds a pattern value

Chapter 24

The MapForce API

24 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 MapForce2011 into the Microsoft Visual Studio versions 2005, 2008 and 2010 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\MapForce2011\Examples\ActiveX** installation folder.

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.

24.1 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

24.1.1 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.

24.1.2 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 ---------------------
```

24.1.3 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
{
   // open an existing mapping.
   // **** adjust the examples path to your needs ! **************
   var sMapForceExamplesPath = fso.BuildPath(
            wshShell.SpecialFolders( "MyDocuments" ),
            "Altova\\MapForce2011\\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 );
}
```

24.1.4 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\\MapForce2011\\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_Language +
                                                                " output folder is " +
objItem.CodeGenSettings_OutputFolder + "\n";
                        }
                        catch( err ) {}

                        // ----- print WSDL settings, if available
                        try
                        {
                                strGlobalText += strTab + "  WSDL File is " +
objItem.WSDLFile +
                                                        " Qualified Name is " +
objItem.QualifiedName + "\n";
                        }
                        catch( err ) {}
                }
                catch( ex )
                        { strGlobalText += strTab + "[" + objItem.Kind + "]\n" }

                // ---- recurse
                PrintProjectTree( new Enumerator( objItem ), strTab + '  ' );

                objProjectItemIter.moveNext();
        }
}

// ---------------------------------------------------------
// Load example project installed with MapForce.
// ---------------------------------------------------------
function LoadSampleProject()
{
        // close open project
        objProject = objMapForce.ActiveProject;
        if ( objProject != null )
                objProject.Close();
```



```
	// open sample project and iterate through it.
	// 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;
```

24.1.5 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)))
  {
   BSTRbstrDescr;
   ipErrorInfo->GetDescription(&bstrDescr);

   // handle Error information
   wprintf(L"Error message:\t%s\n",bstrDescr);
   ::SysFreeString(bstrDescr);

   // release Error info
   ipErrorInfo->Release();
  }
 }
```

24.2 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.

24.2.1 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 ---------------------
```

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.

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.

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.

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.

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.

Edition

Property: `Edition` as String (read-only)

Description
The edition of the product, e.g. Enterprise, Professional, Basic.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

24.2.2 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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

24.2.3 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

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.

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.

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.

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.

24.2.4 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

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.

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.

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.

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.

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.

24.2.5 Component

A Component represents a MapForce component.

Properties and Methods
Properties to navigate the object model:
Application
Parent

Component properties:
HasIncomingConnections
HasOutgoingConnections
CanChangeInputInstanceFile
CanChangeOutputInstanceFile
ID
IsParameterInputRequired
IsParameterSequence
Name
Preview
Schema
SubType
Type

Instance related properties:
InputInstanceFile
OutputInstanceFile

Datapoints:
GetRootDatapoint

Execution:
GenerateOutput

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Name

Property: `Name` as String (read-only)

Description
Retrieves the name of the component.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

OutputInstanceFile

Property: `OutputInstanceFile` as String

Description
Gets or sets the component's output instance file.

Errors
1200 The object is no longer valid.
1201 Invalid address for the return parameter was specified.

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.

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.

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.

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.

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.

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.

24.2.6 Components

Represents a collection of Component objects.

Properties and Methods
Properties to navigate the object model:
Application
Parent

Iterating through the collection:
Count
Item

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.

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.

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.

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.

24.2.7 Connection

A Connection object represents a connector between two components.

Properties and Methods
Properties to navigate the object model:
Application
Parent

Properties
ConnectionType

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.

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.

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.

24.2.8 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

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.

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.

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.

24.2.9 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

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.

Activate

Method: `Activate` ()

Description
Makes this document the active document.

Errors
1200 The application object is no longer valid.

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.

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.

CreateUserDefinedFunction

Method: CreateUserDefinedFunction(strFunctionName as String, strLibraryName as String, strSyntax as String, strDetails as String, bInlinedUse as Boolean) as Mapping

Description
Creates a user defined function in the current document.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1208	Failed creating user-defined function.
1209	Changing the document not allowed. It is read-only.

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.

FullName

Property: `FullName` as String

Description
Path and name of the document file.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

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

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

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

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

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.

See also
Code Generation

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.

See also
Code Generation

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

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

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

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.

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

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.

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.

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.

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.

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`

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`

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.

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.

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.

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.

Saved

Property: `Saved` as `Boolean` (read-only)

Description
`True` if the document was not modified since the last save operation, `false` otherwise.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

24.2.10 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

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.

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.

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.

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.

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.

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.

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.

24.2.11 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

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.

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.

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.

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.

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.

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.

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.

24.2.12 ErrorMarkers

Represents a collection of ErrorMarker objects.

Properties and Methods
Properties to navigate the object model:
Application
Parent

Iterating through the collection:
Count
Item

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.

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.

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.

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.

24.2.13 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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

24.2.14 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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

24.2.15 Mappings

Represents a collection of Mapping objects.

Properties and Methods
Properties to navigate the object model:
Application
Parent

Iterating through the collection:
Count
Item

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.

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.

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.

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.

24.2.16 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:
CompatibilityMode
DefaultOutputEncoding
DefaultOutputByteOrder
DefaultOutputByteOrderMark
XSLTDefaultOutputDirectory
CodeDefaultOutputDirectory
CPPSettings_DOMType
CPPSettings_GenerateVC6ProjectFile
CppSettings_GenerateVSProjectFile
CPPSettings_LibraryType
CPPSettings_UseMFC
CSharpSettings_ProjectType

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.

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

CompatibilityMode

Property: `CompatibilityMode` as `Boolean`

Description
Set to true to generate code compatible with Version 2005R3. Set to false to use newly added code generation features in `Document.GenerateCppCode`, `Document.GenerateCHashCode`, `Document.GenerateJavaCode` and `Document.GenerateXSLT`

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

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

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

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

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

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

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

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

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

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

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

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

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.

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.

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.

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.

WrapperClassesVersion

Property: `WrapperClassesVersion` as `ENUMWrapperClassesVersion`

Description
Specifies the wrapper class version to be generated when generating code, if the generation of wrapper classes is enabled.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

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

24.2.17 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.

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`.

_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) );
    }
}
```

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.

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.

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.

Close

Method: `Close` ()

Description
Closes the project without saving.

Errors
1500 The object is no longer valid.

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`.

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.

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.

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

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

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

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

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.

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
}
```

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.

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.

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.

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.

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.

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.

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.

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.

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.

24.2.18 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.

_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.

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.

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.

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.

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.

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.

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.

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`.

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.

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.

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.

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.

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.

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.

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`.

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.

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.

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.

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.

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.

Remove

Method: `Remove` ()

Description

Remove this project item and all its children from the project tree.

Errors

1700 The object is no longer valid.

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.

24.3 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.

24.3.1 ENUMApacheAxisVersion (obsolete)

Description
Enumeration values to select the Apache Axis version.

Possible values:

eApacheAxisVersion_Axis	= 1
eApacheAxisVersion_Axis2	= 2

See also
Code Generation

24.3.2 ENUMApplicationStatus

Description
Enumeration values to indicate the status of the application.

Possible values:

eApplicationRunning	= 0
eApplicationAfterLicenseCheck	= 1
eApplicationBeforeLicenseCheck	= 2
eApplicationConcurrentLicenseCheckFailed	= 3

24.3.3 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

24.3.4 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

24.3.5 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

24.3.6 ENUMComponentDatapointSide

Description
Enumeration values to indicate the side of a datapoint on its component.

Possible values:

eDatapointSideInput	= 0
eDatapointSideOutput	= 1

See also
GetRootDatapoint

24.3.7 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

24.3.8 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

24.3.9 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

24.3.10 ENUMConnectionType

Description
Enumeration values to indicate the type of a connection.

Possible values:

eConnectionTypeTargetDriven	= 0
eConnectionTypeSourceDriven	= 1
eConnectionTypeCopyAll	= 2

See also
ConnectionType

24.3.11 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

24.3.12 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

24.3.13 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

24.3.14 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

24.3.15 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

24.3.16 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

24.3.17 ENUMViewMode

Description
Enumeration values to select a MapForce view.

Possible values:

eMapForceView	= 0
eXSLView	= 1
eOutputView	= 2

24.3.18 ENUMWrapperClassesVersion

Description
Enumeration values to select the version of generated wrapper classes.

Possible values:

eWrapperClassesVersion_2005r3	= 0
eWrapperClassesVersion_2006	= 1
eWrapperClassesVersion_2007r3	= 2

See also
Code Generation

Chapter 25

MapForce Integration

25 MapForce 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#, VisualBasic, or HTML to be used for integration. ActiveX components officially only work with Microsoft Internet Explorer. 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 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, both of which have examples in various programming languages, 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.

25.1 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 this section is an example (Example: HTML) showing 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.

25.1.1 Example: HTML

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: `MapForceExamples\ActiveX\HTML\MapForceActiveX_ApplicationLevel.htm`.

Note: This example works only in Internet Explorer.

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\XMLSpy2011\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>
```

25.2 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.

25.2.1 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`.

25.2.2 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 supports 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.

25.2.3 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.

25.2.4 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 `MapForceExamples\ActiveX\Images` of your MapForce installation for icons in GIF format. The file names correspond to the `labels` of commands.

25.2.5 Examples

This section contains examples of MapForce document-level integration using different container environments and programming languages. Source code for all examples is available in the folder `MapForceExamples\ActiveX` of your MapForce installation.

C#

The C# example shows how to integrate the MapForceControl in a common desktop application created with C# using Visual Studio 2008.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.

Please note that the example application is already complete. There is no need to change anything if you want to run and see it working. The following steps describe what general actions and considerations must be taken in order to create a project such as this.

Introduction

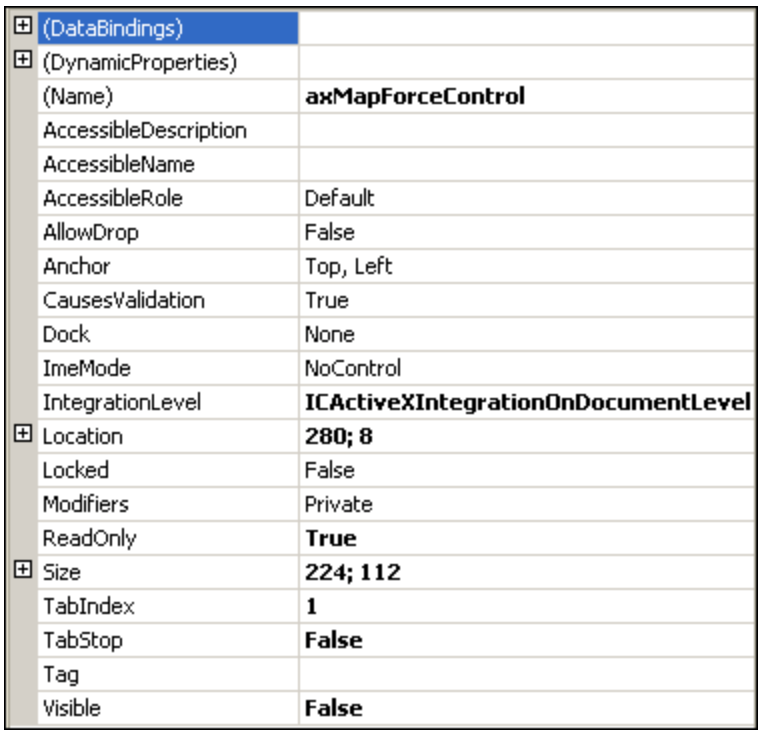
Adding the MapForce components to the Toolbox
Before you take a look at the sample project please add the assemblies to the .NET IDE Toolbox. The MapForce Installer will have already installed the assemblies in the .NET Global Assembly Cache (GAC). If you open the Toolbox dialog under **Tools | Add/Remove 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 in the `ActiveX\C#\MapForceApplication` folder to load the project.

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.

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

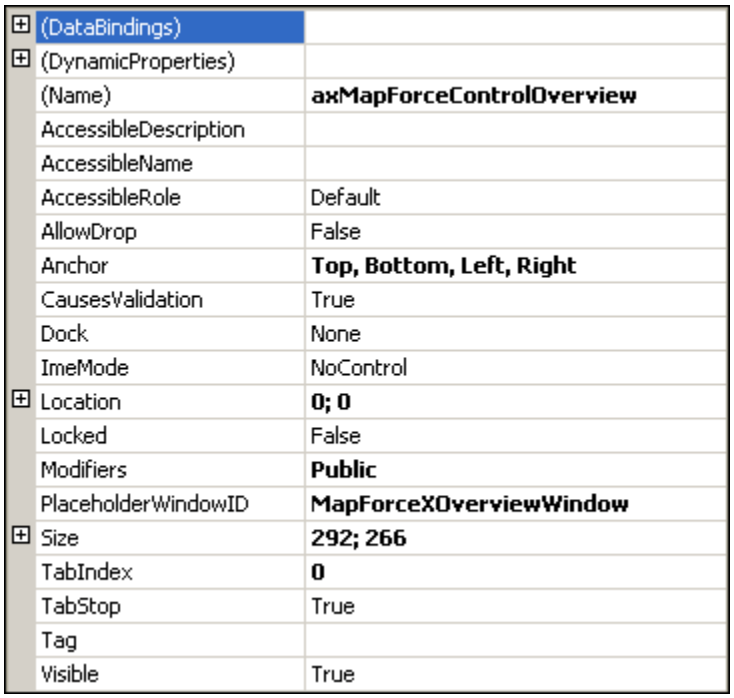
The Placeholders also have the Anchor and Dock properties set in order to react on resizing of the Frame window.

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 MapForce

OverviewWnd 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.axM
apForceControlOverview.GetOcx();
type.AssignMultiDocCtrl((MapForceControlLib.MapForceControlClass)axMapForc
eControl.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.

Retrieving Command Information

The MapForceControl gives access to all commands of MapForce through its `CommandsStructure` property. The example project uses the `MapForceCommands` and `MapForceCommand` 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("MapForce
    Application.CustomMenuItem"))
    {
        CustomMenuItemcustomItem = (CustomMenuItem)sender;

        ProcessCommand(customItem.m_MapForceCmdID);
    }
}
```



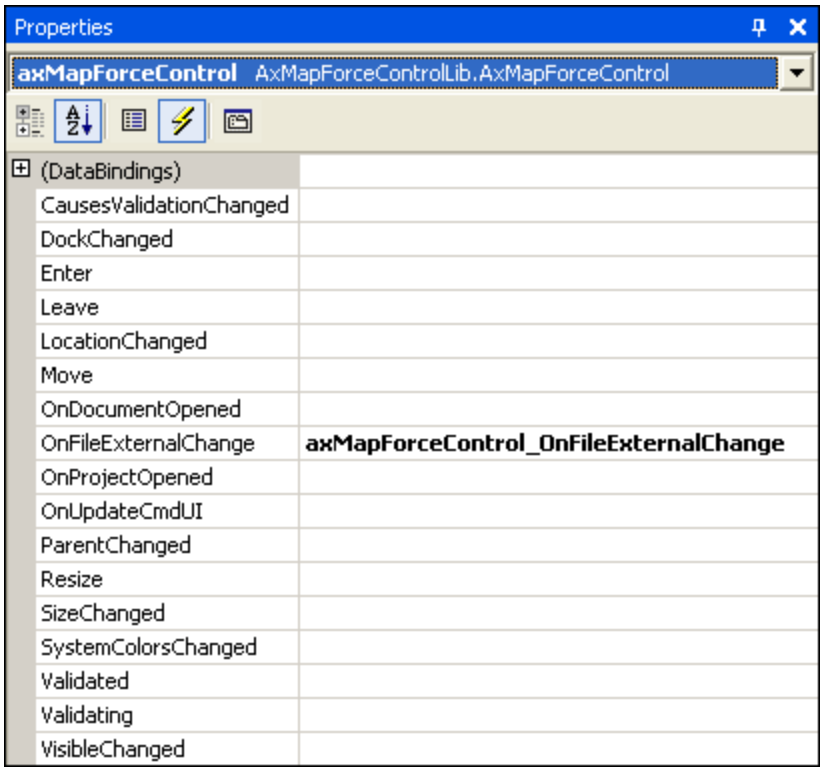
```
private void ProcessCommand(int nID)
{
    MapForceDoc  docMapForce = GetCurrentMapForceDoc();

    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.

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:

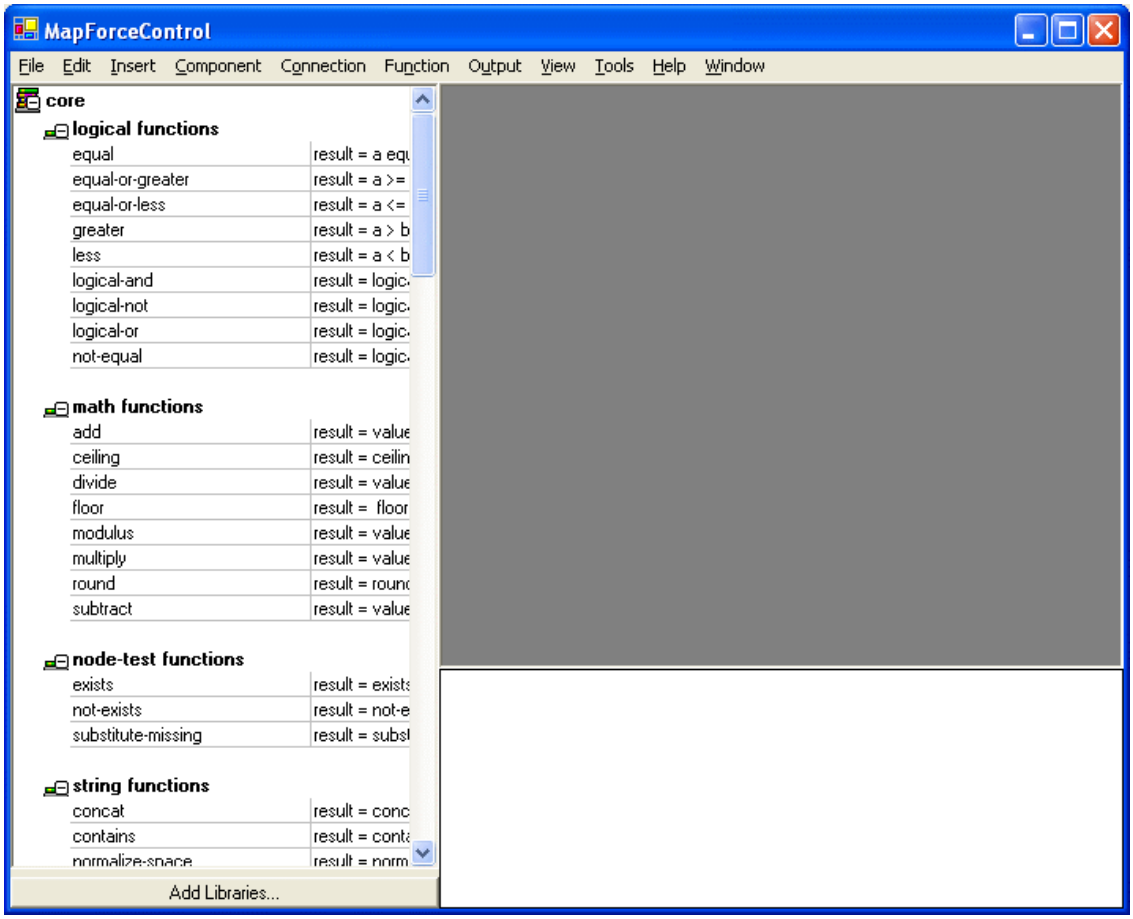
```
private void axMapForceControl_OnFileExternalChange(object sender,
AxMapForceControlLib._DMapForceControlEvents_OnFileExternalChangeEvent e)
```

```
{
    MessageBox.Show("Attention: The file " + e.strPath + " has been changed
from outside\nbut reloading is turned off in the sample application!");

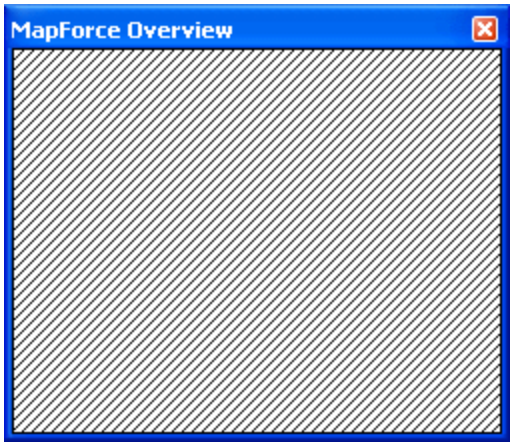
    // This turns off any file reloading:
    e.varRet = false;
}
```

Testing the Example

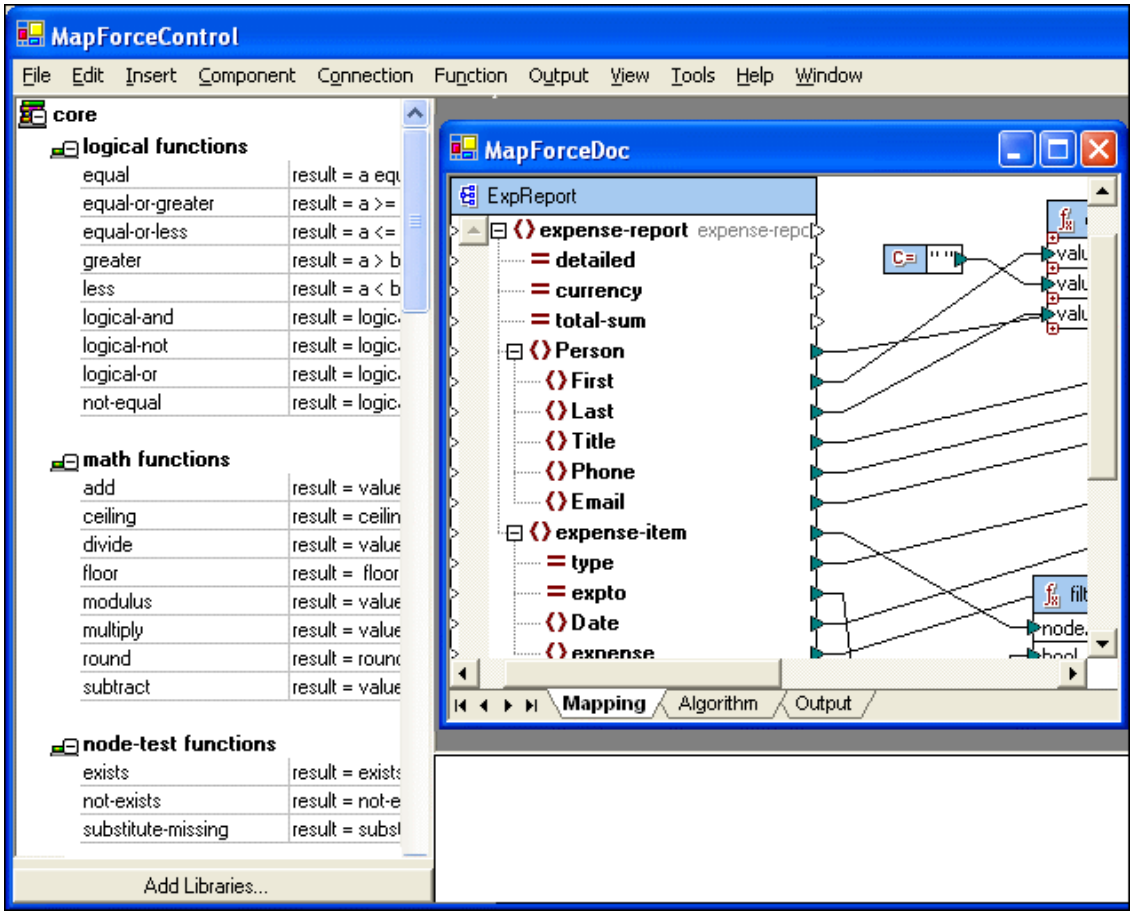
After adding the assemblies to the Toolbox (see Introduction), you can run the sample project with F5 without the need to change anything in the code. The main MDI Frame window is created together with a floating Toolwindow containing the Overview window of MapForce. The application looks something like 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.

HTML

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 `MapForce ActiveX_ApplicationLevel.htm` within the `C:\Program Files\Altova\MapForce2011\Examples\ActiveX\HTML\` folder of your MapForce installation.

Note: This example works only in Internet Explorer.

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

```
<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\XMLSpy2011\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 mouseclick 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>
```

Visual Basic

Source code for an integration of MapForceControl into a VisualBasic program can be found in the folder `MapForceExamples\ActiveX\VisualBasic6` relative to your MapForce installation.

25.3 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.

25.3.1 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

25.3.2 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

25.3.3 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

25.3.4 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

25.3.5 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

25.3.6 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

25.3.7 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

25.3.8 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

25.3.9 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

25.3.10 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

25.3.11 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

25.3.12 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

25.3.13 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

25.4 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.

25.5 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.

25.5.1 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.

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.

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.

IsSeparator

Property: IsSeparator as boolean

Description:
True if the command is a separator.

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.

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.

SubCommands

Property: `SubCommands` as `Commands`

Description:
The `SubCommands` collection holds any sub-commands if this command is actually a menu or submenu.

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.

25.5.2 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.

Count

Property: Count as long

Description:
Number of Command objects on this level of the collection.

Item

Property: Item (n as long) as Command

Description:
Gets the command with the index n in this collection. Index is 1-based.

25.5.3 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
```

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.

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.

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 harddisk 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.

25.5.4 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
```

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 functionalities 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.

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: `OpenDocument` (`strFileName` as string) as boolean

Dispatch Id: *1004*

Description:
`SaveAs` sets `Path` to *strFileName* and then saves the document to this location.

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 harddisk 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.

25.5.5 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
```

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 functionalities 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.

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)`.

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.

25.5.6 Enumerations

The following enumerations are defined:

ICActiveXIntegrationLevel
MapForceControlPlaceholderWindow

ICActiveXIntegrationLevel

Possible values for the IntegrationLevel property of the MapForceControl.

```
ICActiveXIntegrationOnApplicationLevel    = 0
ICActiveXIntegrationOnDocumentLevel       = 1
```

MapForceControlPlaceholderWindow

This enumeration contains the list of the supported additional MapForce windows.

```
MapForceXNoWindow             = -1
MapForceXLibraryWindow        =  0
MapForceXOverviewWindow       =  1
MapForceXValidationWindow     =  2
MapForceXProjectWindow        =  3
```

Chapter 26

Appendices

26 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

26.1 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.

26.1.1 XSLT 1.0 Engine: Implementation Information

The Altova XSLT 1.0 Engine is built into Altova's XMLSpy, StyleVision, Authentic, and MapForce XML products. It is also available in the free AltovaXML package. The Altova XSLT 1.0 Engine implements and conforms to the World Wide Web Consortium's XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999 . Limitations and implementation-specific behavior are listed below.

Limitations

- The **`xsl:preserve-space`** and **`xsl:strip-space`** elements are not supported.
- 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 directly as special characters; they are not inserted as HTML character references in the output. For instance, the character ` ` (the decimal character reference for a non-breaking space) is not inserted as ` ` in the HTML code, but directly as a non-breaking space.

Implementation's handling of whitespace-only nodes in source XML document

The XML data (and, consequently, the XML Infoset) that is passed to the Altova XSLT 1.0 Engine is stripped of boundary-whitespace-only text nodes. (A boundary-whitespace-only text node is a whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping may have an effect on the value returned by the `fn:position()`, `fn:last()`, and `fn:count()` functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The `fn:position()`, `fn:last()`, and `fn:count()` functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when `xsl:apply-templates` is used to apply templates. When the `fn:position()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `para[3]`, which is short for `para[position()=3]`), boundary-whitespace-only nodes are irrelevant since only the named elements (`para` in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, `*[10]`.)

Note: If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
   <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either

the <b> or <i> elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</i>.</para> or
<para>This is <b>bold&#x20;</b> <i>italic</i>.</para> or
<para>This is <b>bold</b><i>&#x20;italic</i>.</para>
```

When any of the para elements above is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

26.1.2 XSLT 2.0 Engine: Implementation Information

The Altova XSLT 2.0 Engine is built into Altova's XMLSpy, StyleVision, Authentic, and MapForce XML products. It is also available in the free AltovaXML package. This section describes the engine's implementation-specific aspects of behavior. It starts with a section giving general information about the engine, and then goes on to list the implementation-specific behavior of XSLT 2.0 functions.

For information about implementation-specific behavior of XPath 2.0 functions, see the section, XPath 2.0 and XQuery 1.0 Functions.

General Information

The Altova XSLT 2.0 Engine conforms to the World Wide Web Consortium's (W3C's) XSLT 2.0 Recommendation of 23 January 2007. Note the following general information about the engine.

Backwards Compatibility

The Altova XSLT 2.0 Engine is backwards compatible. The only time the backwards compatibility of the XSLT 2.0 Engine comes into play is when using the XSLT 2.0 Engine of Altova XML 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.

In all other Altova products, the backwards-compatibility issue never arises. This is because these products automatically select the appropriate engine for the transformation. For example, consider that in XMLSpy you specify that a certain XML document be processed with an XSLT 1.0 stylesheet. When the transformation command is invoked, XMLSpy automatically selects the XSLT 1.0 Engine of XMLSpy to carry out the transformation.

Note: The stylesheet version is specified in the `version` attribute of the `stylesheet` or `transform` element of the stylesheet.

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 Altova 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`).
- With the CRs of 23 January 2007, the `untypedAtomic` and duration datatypes (`dayTimeDuration` and `yearMonthDuration`), which were formerly in the XPath Datatypes namespace (typically prefixed `xdt:`) have been moved to the XML Schema namespace.
- 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 Altova XSLT 2.0 Engine is schema-aware.

Whitespace in XML document
By default, the Altova XSLT 2.0 Engine strips all boundary whitespace from boundary-whitespace-only nodes in the source XML document. The removal of this whitespace affects the values that the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions return. For more details, see Whitespace-only Nodes in XML Document in the XPath 2.0 and XQuery 1.0 Functions section.

Note: If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
   <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</>.</para> or
<para>This is <b>bold&#x20;</b> <i>italic</>.</para> or
<para>This is <b>bold</b><i>&#x20;italic</>.</para>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

XSLT 2.0 elements and functions
Limitations and implementation-specific behavior of XSLT 2.0 elements and functions are listed

in the section XSLT 2.0 Elements and Functions.

XPath 2.0 functions
Implementation-specific behavior of XPath 2.0 functions is listed in the section XPath 2.0 and XQuery 1.0 Functions.

XSLT 2.0 Elements and Functions

Limitations
The **`xsl:preserve-space`** and **`xsl:strip-space`** elements are not supported.

Implementation-specific behavior
Given below is a description of how the Altova XSLT 2.0 Engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

`xsl:result-document`
Additionally supported encodings are: `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: `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: `x-binarytobase16` and `x-binarytobase64`.

Note: The following encoding values, which were implemented in earlier versions of AltovaXML are now deprecated: `base16tobinary`, `base64tobinary`, `binarytobase16` and `binarytobase64`.

26.1.3 XQuery 1.0 Engine: Implementation Information

The Altova XQuery 1.0 Engine is built into Altova's XMLSpy and MapForce XML products. It is also available in the free AltovaXML package. This section provides information about implementation-defined aspects of behavior.

Standards conformance

The Altova XQuery 1.0 Engine conforms to the World Wide Web Consortium's (W3C's) XQuery 1.0 Recommendation of 23 January 2007. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the Altova XQuery 1.0 Engine implements these features.

Schema awareness

The Altova 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 Altova 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 Altova 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. No type checking is done in the static analysis phase. 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 Altova 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. No other collation is currently supported. Comparisons, including the `fn:max` function, are based on this collation.

Character normalization
No character normalization form is supported.

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.

XQuery Functions Support
For information about implementation-specific behavior of XQuery 1.0 functions, see the section, XPath 2.0 and XQuery 1.0 Functions.

26.1.4 XPath 2.0 and XQuery 1.0 Functions

XPath 2.0 and XQuery 1.0 functions are evaluated by:

- the **Altova XPath 2.0 Engine**, which (i) is a component of the Altova XSLT 2.0 Engine, and (ii) is used in the XPath Evaluator of Altova's XMLSpy product to evaluate XPath expressions with respect to the XML document that is active in the XMLSpy interface.
- the **Altova XQuery 1.0 Engine**.

This section describes how XPath 2.0 and XQuery 1.0 functions are handled by the Altova XPath 2.0 Engine and Altova XQuery 1.0 Engine. Only those functions are listed, for which the behavior is implementation-specific, or where the behavior of an individual function is different in any of the three environments in which these functions are used (that is, in XSLT 2.0, in XQuery 1.0, and in the XPath Evaluator of XMLSpy). Note that this section does not describe how to use these functions. For more information about the usage of functions, see the World Wide Web Consortium's (W3C's) XQuery 1.0 and XPath 2.0 Functions and Operators Recommendation of 23 January 2007.

General Information

Standards conformance

- The Altova XPath 2.0 Engine implements the World Wide Web Consortium's (W3C's) XPath 2.0 Recommendation of 23 January 2007. The Altova XQuery 1.0 Engine implements the World Wide Web Consortium's (W3C's) XQuery 1.0 Recommendation of 23 January 2007. The XPath 2.0 and XQuery 1.0 functions support in these two engines is compliant with the XQuery 1.0 and XPath 2.0 Functions and Operators Recommendation of 23 January 2007.
- The Altova XPath 2.0 Engine conforms to the rules of XML 1.0 (Fourth Edition) and XML Namespaces (1.0).

Default functions namespace

The default functions namespace has been set to comply with that specified in the standard. Functions can therefore be called without a prefix.

Boundary-whitespace-only nodes in source XML document

The XML data (and, consequently, the XML Infoset) that is passed to the Altova XPath 2.0 Engine and Altova XQuery 1.0 Engine is stripped of boundary-whitespace-only text nodes. (A boundary-whitespace-only text node is a child whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping has an effect on the value returned by the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when `xsl:apply-templates` is used to apply templates. When the `fn:position()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `para[3],` which is short for `para[position()=3]`),

boundary-whitespace-only nodes are irrelevant since only the named elements (`para` in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, `*[10]`.)

Numeric notation
On output, when an `xs:double` is converted to a string, scientific notation (for example, `1.0E12`) is used when the absolute value is less than 0.000001 or greater than 1,000,000. Otherwise decimal or integer notation is used.

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 `fn:implicit-timezone()` function.

Collations
Only the Unicode codepoint collation is supported. No other collations can be used. String comparisons, including for the `fn:max` and `fn:min` functions, are based on this collation.

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 `fn:in-scope-prefixes()`, `fn:namespace-uri()` and `fn:namespace-uri-for-prefix()` functions.

Static typing extensions
The optional static type checking feature is not supported.

Functions Support

The table below lists (in alphabetical order) the implementation-specific behavior of certain functions. The following general points should be noted:

- In general, when 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`.

Function Name	Notes

`base-uri`	• If external entities are used in the source XML document and if a node in the external entity is specified as the input node argument of the `base-uri()` function, it is still the base URI of the including XML document that is used—not the base URI of the external entity. • The base URI of a node in the XML document can be modified using the `xml:base` attribute.

`collection`	• The argument is a relative URI that is resolved against the current base URI.
	• If the resolved URI identifies an XML file, then this XML file is treated as a catalog which references a collection of files. This file must have the form:
	`<collection>`
	`  <doc href="uri-1" />`
	`  <doc href="uri-2" />`
	`  <doc href="uri-3" />`
	`</collection>`
	The files referenced by the `href` attributes are loaded, and their document nodes are returned as a sequence.
	• If the resolved URI does not identify an XML file with the catalog structure described above, then the argument string (in which wildcards such as `?` and `*` are allowed) is used as a search string. XML files with names that match the search expression are loaded, and their document nodes are returned as a sequence. See examples below.
	• XSLT example: The expression `collection("c:\MyDocs\*.xml")//Title` returns a sequence of all `DocTitle` elements in the `.xml` files in the `MyDocs` folder.
	• XQuery example: The expression `{for $i in collection(c:\MyDocs\*.xml) return element doc{base-uri($i)}}` returns the base URIs of all the `.xml` files in the `MyDocs` folder, each URI being within a `doc` element.
	• The default collection is empty.

Function Name	Notes
`count`	• See note on whitespace in the General Information section.
`current-date, current-dateTime, current-time`	• The current date and time is taken from the system clock. • The timezone is taken from the implicit timezone provided by the evaluation context; the implicit timezone is taken from the system clock. • The timezone is always specified in the result.
`deep-equal`	• See note on whitespace in the General Information section.

doc	• An error is raised only if no XML file is available at the specified location or if the file is not well-formed. The file is validated if a schema is available. If the file is not valid, the invalid file is loaded without schema information.
id	• In a well-formed but invalid document that contains two or more elements having the same ID value, the first element in document order is returned.
in-scope-prefixes	• Only default namespaces may be undeclared in the XML document. However, even when a default namespace is undeclared on an element node, the prefix for the default namespace, which is the zero-length string, is returned for that node.
last	• See note on whitespace in the General Information section.
lower-case	• The Unicode character set is supported.
normalize-unicode	• The normalization forms NFC, NFD, NFKC, and NFKD are supported.

Function Name	Notes
position	• See note on whitespace in the General Information section.
resolve-uri	• If the second, optional argument is omitted, the URI to be resolved (the first argument) is resolved against the base URI from the static context, which is the URI of the XSLT stylesheet or the base URI given in the prolog of the XQuery document. • The relative URI (the first argument) is appended after the last "/" in the path notation of the base URI notation. • If the value of the first argument is the zero-length string, the base URI from the static context is returned, and this URI includes the file name of the document from which the base URI of the static context is derived (e.g. the XSLT or XML file).
static-base-uri	• The base URI from the static context is the base URI of the XSLT stylesheet or the base URI specified in the prolog of the XQuery document. • When using XPath Evaluator in the XMLSpy IDE, the base URI from the static context is the URI of the active XML document.
upper-case	• The Unicode character set is supported.

26.1.5 Extensions

There are several ready-made functions in programming languages such as Java and C# that are not available as XPath 2.0 / XQuery 1.0 functions or as XSLT 2.0 functions. A good example of such functions are 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.

Altova Engines (XSLT 1.0, XSLT 2.0, and XQuery 1.0), which are used in a number of Altova products, support the use of extension functions in Java and .NET. The Altova XSLT Engines additionally support MSXSL scripts for XSLT 1.0 and 2.0 and Altova's own extension functions.

You should note that extension functions are always called from XPath expressions. This section describes how to use extension functions and MSXSL scripts in your XSLT stylesheets and XQuery queries. These descriptions are organized into the following sections:

- Java Extension Functions
- .NET Extension Functions
- MSXSL Scripts for XSLT
- Altova Extension Functions

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: XSLT/XQuery to Java
- Datatypes: Java to XSLT/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.
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file.
- The class file is in a package. The XSLT or XQuery file is at some random location.
- The class file is not packaged. The XSLT or XQuery file is at some random location.

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, 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: XSLT/XQuery to .NET
- Datatypes: .NET to XSLT/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.

Altova Extension Functions

Altova extension functions are in the namespace **http://www.altova.com/xslt-extensions** and are indicated in this section with the prefix **altova:**, which is assumed to be bound to the namespace given above.

The following extension functions 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.

General functions

- **altova:evaluate()**
- **altova:distinct-nodes()**
- **altova:encode-for-rtf()**
- **altova:xbrl-labels()**
- **altova:xbrl-footnotes()**
- **altova:generate-auto-number()**
- **altova:reset-auto-number()**
- **altova:get-temp-folder()**

General Functions

The following extension functions 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.

- **altova:evaluate()**
- **altova:distinct-nodes()**
- **altova:encode-for-rtf()**
- **altova:xbrl-labels()**
- **altova:xbrl-footnotes()**

- altova:generate-auto-number()
- altova:reset-auto-number()
- altova:get-temp-folder()

altova:evaluate()
The altova:evaluate() function takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression.

```
altova:evaluate(XPathExp as xs:string)
```

For example:

```
altova:evaluate('//Name[1]')
```

In the example above, note that the expression //Name[1] is passed as a string by enclosing it in single quotes. The altova:evaluate function returns the contents of the first Name element in the document.

The altova:evaluate function can take additional (optional) arguments. These arguments are, respectively, the values of variables with the names p1, p2, p3... pN that can be used in the XPath expression.

```
altova:evaluate(XPathExp as xs:string [, p1value ... pNvalue])
```

where

- the variable names must be of the form pX, X being an integer
- the sequence of the function's arguments, from the second argument onwards corresponds to the sequence of variables named p1 to pN. So the second argument will be the value of the variable p1, the third argument that of the variable p2, and so on.
- The variable values must be of type item*

For 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 above listing, 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.

The following examples further illustrate usage:

```
<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).

Variables can be used in the altova:evaluate() extension function as shown in the examples below:

- 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.

Note: The static context includes namespaces, types, and functions—but not variables—from the calling environment. The base URI and default namespace are inherited.

altova:distinct-nodes()

The altova:distinct-nodes() function 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.

```
altova:distinct-nodes( $arg as node()* )  as node()*
```

altova:encode-for-rtf()

The altova:encode-for-rtf() function converts the input string into code for RTF.

```
altova:encode-for-rtf( $inputstr as xs:string?,
$preserveallwhitespace as xs:boolean,
$preservenewlines as xs:boolean) as xs:string
```

Whitespace and new lines will be preserved according to the boolean value specified for their respective parameters.

altova:xbrl-labels()

The `altova:xbrl-labels()` function takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL labels associated with the input node.

```
altova:xbrl-labels( $name as xs:QName, $file as xs:string )  as node()*
```

altova:xbrl-footnotes()

The `altova:footnotes()` function takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

```
altova:footnotes( $arg as node() )  as node()*
```

altova:generate-auto-number(id as xs:string, start-with as xs:integer, increment as xs:integer, reset-on-change as xs:string)

Generates a series of numbers having the specified ID. The start integer and the increment is specified.

altova:reset-auto-number(id as xs:string)

This function resets the auto-numbering of the auto-numbering series specified with the ID argument. The series is reset to the start integer of the series (see `altova:generate-auto-number` above).

altova:get-temp-folder as xs:string

Gets the temporary folder.

26.2 Technical Data

This section contains useful background information on the technical aspects of your software. It is organized into the following sections:

- OS and Memory Requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode Support
- Internet Usage

26.2.1 OS and Memory Requirements

Operating System
Altova software applications are:

- 32-bit Windows applications for Windows XP, Windows Server 2003 and 2008, Windows Vista, and Windows 7, or
- 64-bit Windows applications for Windows Vista and Windows 7

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.

26.2.2 Altova XML Parser

When opening any XML document, the application uses its built-in validating parser (the Altova XML Parser) to check for well-formedness, validate the document against a schema (if specified), and build trees and Infosets. The Altova XML Parser 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 Altova XML Parser implements the Final Recommendation of the W3C's XML Schema specification. New developments recommended by the W3C's XML Schema Working Group are continuously being incorporated in the Altova Parser, so that Altova products give you a state-of-the-art development environment.

26.2.3 Altova XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery 1.0 Engines. Documentation about implementation-specific behavior for each engine is in the section Engine Information, in Appendix 1 of the product documentation, should that engine be used in the product.

These three engines are also available in the AltovaXML package, which can be downloaded from the Altova website free of charge. Documentation for using the engines is available with the AltovaXML package.

26.2.4 Unicode Support

Unicode is the 16-bit character-set (extendable to 32-bit) defined by the Unicode Consortium. It provides a unique number for every character,

- no matter what the platform,
- no matter what the program,
- no matter what the language.

Fundamentally, computers just deal with numbers. They store letters and other characters by assigning a number for each one. Before Unicode was invented, there were hundreds of different encoding systems for assigning these numbers. No single encoding could contain enough characters: for example, the European Union alone requires several different encodings to cover all its languages. Even for a single language like English, no single encoding was adequate for all the letters, punctuation, and technical symbols in common use.

These encoding systems used to conflict with one another. That is, two encodings used the same number for two different characters, or different numbers for the same character. Any given computer (especially servers) needs to support many different encodings; yet whenever data is passed between different encodings or platforms, that data always runs the risk of corruption.

Unicode is changing all that!
Unicode provides a unique number for every character, no matter what the platform, no matter what the program, and no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystems, Microsoft, Oracle, SAP, Sun, Base and many others.

Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646. It is supported in many operating systems, all modern browsers, and many other products. The emergence of the Unicode Standard, and the availability of tools supporting it, are among the most significant recent global software technology trends.

Incorporating Unicode into client-server or multi-tiered applications and web sites offers significant cost savings over the use of legacy character sets. Unicode enables a single software product or a single web site to be targeted across multiple platforms, languages and countries without re-engineering. It allows data to be transported through many different systems without corruption.

Windows XP

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. Consequently you may encounter XML documents that contain "unprintable" characters, because the font you have selected does not contain the required glyphs. Therefore it can sometimes be very useful to have a font that covers the entire Unicode range - especially when editing XML documents from all over the world.

The most universal font we have encountered is a typeface called Arial Unicode MS that has been created by Agfa Monotype for Microsoft. This font contains over 50,000 glyphs and covers the entire set of characters specified by the Unicode 2.1 standard. It needs 23MB and is included with Microsoft Office 2000.

We highly recommend that you install this font on your system and use it with the application if you are often editing documents in different writing systems. This font is not installed with the "Typical" setting of the Microsoft Office setup program, but you can choose the Custom Setup option to install this font.

In the `/Examples` folder in your application folder you will also find a new XHTML file called `Unicode-UTF8.html` that contains the sentence "When the world wants to talk, it speaks Unicode" in many different languages ("Wenn die Welt miteinander spricht, spricht sie Unicode") and writing-systems (世界的に話すなら、Unicode です。) - this line has been adopted from the 10th Unicode conference in 1997 and is a beautiful illustration of the importance of Unicode for the XML standard. Opening this file will give you a quick impression on what is possible with Unicode and what writing systems are supported by the fonts available on your PC installation.

Right-to-Left Writing Systems

Please note that even under Windows NT 4.0 any text from a right-to-left writing-system (such as Hebrew or Arabic) is not rendered correctly except in those countries that actually use right-to-left writing-systems. This is due to the fact that only the Hebrew and Arabic versions of Windows NT contains support for rendering and editing right-to-left text on the operating system layer.

26.2.5 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.
- If you use the URL mode of the Open dialog box to open a document directly from a URL (**File | Open | Switch to URL**), that document is retrieved through a http (port 80) connection. (*This functionality is available in XMLSpy and Authentic Desktop.*)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, it is also retrieved through a http (port 80) connection once you validate the XML document. This may also happen automatically upon opening a document if you have instructed the application to automatically validate files upon opening in the File tab of the Options dialog (**Tools | Options**). (*This functionality is available in XMLSpy and Authentic Desktop.*)
- 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 this manual and the Altova Software License Agreement.

26.3 License Information

This section contains:

- Information about the distribution of this software product
- 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.

26.3.1 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.

26.3.2 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* http://www.isi.edu/in-notes/iana/assignments/port-numbers *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.

26.3.3 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.

26.3.4 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 ("Software License 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 Software License 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 Software License Agreement as part of the installation process at the time of acceptance. Alternatively, you may go to our Web site at http://www.altova.com/eula to download and print a copy of this Software License Agreement for your files and http://www.altova.com/privacy to review the Privacy Policy.

1. SOFTWARE LICENSE

(a) License Grant.

(i) Upon your acceptance of this Software License 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 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. The Permitted Number of computers and/or users shall be determined and specified at such time as you elect to purchase the Software. 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 Software License 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 Software License 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 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. 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(i) 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® 2011] and include libraries owned by Altova GmbH, Copyright © 2007-2011 Altova GmbH (www.altova.com)."

(b) Server Use. 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.

If you have purchased Concurrent User Licenses as defined in Section 1(d), and subject to limits set forth therein, you may install a copy of the Software on a terminal server (Microsoft Terminal Server, Citrix Metaframe, etc.), application virtualization server (Microsoft App-V, Citrix XenApp, VMWare ThinApp, etc.) 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 six (6) times the Permitted Number of users. Accordingly, the limitations set forth in Section 1(d) regarding the number of installations and the requirement that the usage be on the same physical network shall not apply to terminal server, application virtualization session, or virtual machine environments. 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 and technical support is not available with respect to issues arising from use in such environments.

(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. 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 physical computer network. The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses. Each separate physical network or office location 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 physical network, then a locally installed user license is required. Home User restrictions and limitations with respect to the Concurrent User licenses used on home computers are set forth in Section 1(f).

(e) 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.

(f) Home Use (Personal and Non-Commercial). In order to further familiarize yourself with the Software and allow you to explore its features and functions, you, as the primary user of the computer on which the Software is installed for commercial purposes, may also install one copy of the Software on only one (1) home personal computer (such as your laptop or desktop) solely for your personal and non-commercial ("HPNC") use. This HPNC copy may not be used in any commercial or revenue-generating business activities, including without limitation, work-from-home, teleworking, telecommuting, or other work-related use of the Software. The HPNC copy of the Software may not be used at the same time on a home personal computer as the Software is being used on the primary computer.

(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 Software License 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 Software License 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 Software License 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 Software License Agreement.

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 Software License Agreement and to ensure their compliance with these restrictions.

(k) 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 3RD 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

Acknowledgement of Altova's 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. XMLSpy®, Authentic®, StyleVision®, MapForce®, UModel®, DatabaseSpy®, DiffDog®, SchemaAgent®, SemanticWorks®, MissionKit®, Markup Your Mind®, Nanonull™, and Altova® are trademarks of Altova GmbH. (registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows XP, Windows Vista, and Windows 7 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 Software License 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 each of this Software License 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 Software License 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 Software License 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 Software License 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 Software License 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 SOFTWARE LICENSE 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 Software License 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 Software License 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 infringements that would not be such, except for customer-supplied elements.

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 Software License Agreement before installation. Updates of the operating system and application software not specifically covered by this Software License Agreement are your responsibility and will not be provided by Altova under this Software License 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 Software License 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. Altova has a built-in license metering module that helps you to avoid any unintentional violation of this Software License Agreement. Altova may use your internal network for license metering between installed versions of the Software.

(b) Software Activation. **Altova's 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 between your computer and the Altova 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 Software License Agreement. You agree that efforts to circumvent or disable Altova's copyright protection mechanisms or license management mechanism violate Altova's intellectual property rights as well as the terms of this Software License 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.**

(c) 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.

(d) 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 Software License Agreement. By your acceptance of the terms of this Software License 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 Software License Agreement and/or the Privacy Policy. Altova has the right in its sole discretion to amend this provision of the Software License 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.

(e) 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 Software License Agreement, you consent to the transfer of all such information to the United

States and the processing of that information as described in this Software License Agreement and the Privacy Policy.

8. TERM AND TERMINATION

This Software License 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 Software License 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 Software License Agreement governing your use of a previous version that you have upgraded or updated of the Software is terminated upon your acceptance of the terms and conditions of the Software License Agreement accompanying such upgrade or update. Upon any termination of the Software License Agreement, you must cease all use of the Software that this Software License 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), 2, 5(b), 5(c), 5(d), 7(d), 7(e), 9, 10 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 Software License 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. 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.

11. GENERAL PROVISIONS

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 Software License 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 Software License 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 Software License 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 Software License 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.

This Software License 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 Software License 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 Software License 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 Software License Agreement. This Software License 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 Software License 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 Software License Agreement. If, for any reason, any provision of this Software License Agreement is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this Software License Agreement, and this Software License 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: 2011-06-08

Index

(

.

<

0

2

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