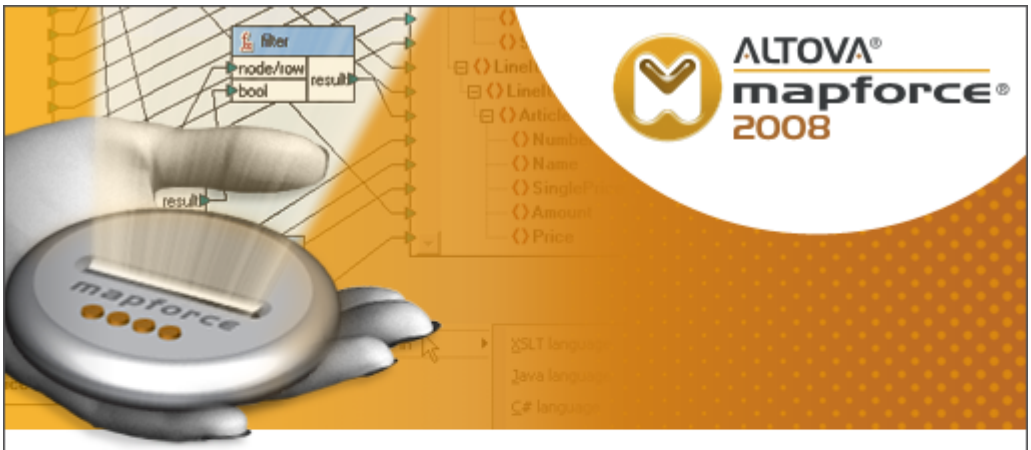
User and Reference Manual



Altova MapForce 2008 User & Reference Manual

Published: 2008

Table of Contents

Index

Chapter 1

MapForce 2008

1 MapForce 2008

MapForce® 2008 **Enterprise Edition** is a visual data mapping tool for advanced data integration projects.

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

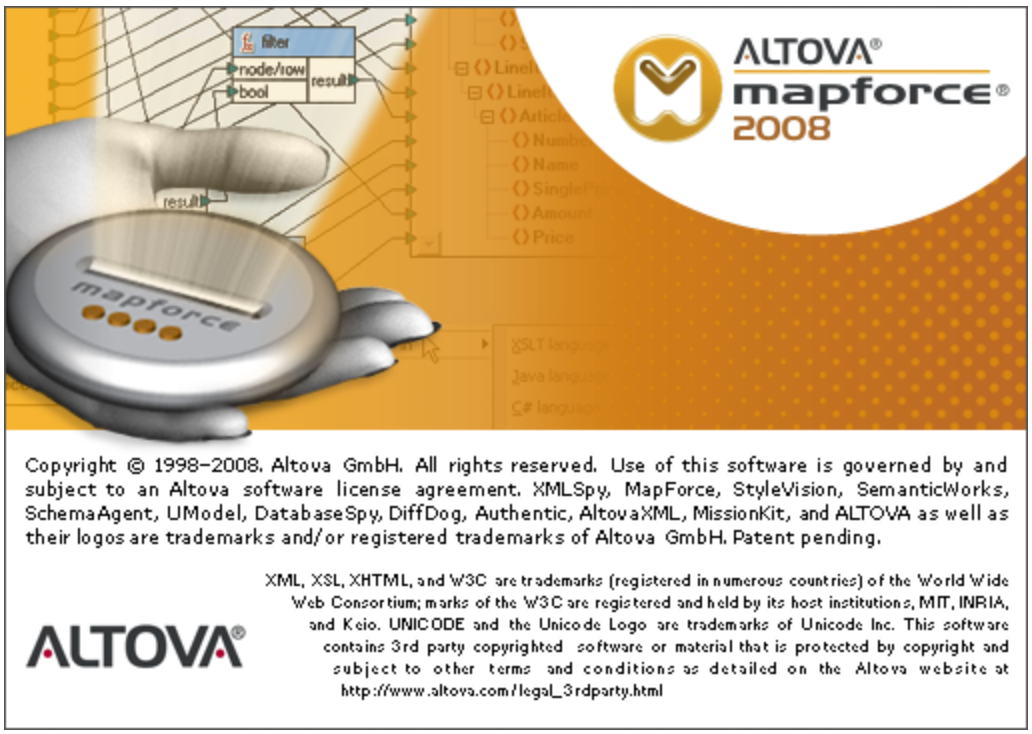
  Enterprise Edition, additionally:
  - EDI files: UN/EDIFACT and ANSI X12 as source and target
  - FlexText™ files as source and target
  - Office Open XML Excel 2007 files as source and target
- Automatic code generation
  - XSLT 1.0 and 2.0

  Professional Edition and Enterprise Edition, additionally:
  - XQuery
  - Java, C# and C++
- On-the-fly transformation and preview of all mappings, without code generation or compilation
- 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 and integration of custom C++, Java and C# functions
- 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

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
- Project management functions to group mappings
- MapForce plug-in for Eclipse
- MapForce for Microsoft Visual Studio

Enterprise Edition, additionally:
- Creation of SOAP 1.1 and SOAP 1.2 Web service projects and mapping of Web service operations from WSDL 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.



What is mapping?
Basically the contents of one component are mapped, or transformed, to another component. An XML, or text document, a database, Excel 2007 file or EDI file, can be mapped to a different target XML document, CSV text document, EDI file, Excel 2007 file or database. The transformation is accomplished by an automatically generated XSLT 1.0 or 2.0 Stylesheet, the built-in MapForce engine, or generated program code.

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.

Databases as well as EDI documents, can also be used as data sources, and map data to multiple XML Schemas, EDI documents, or other databases.

Chapter 2

What's new in MapForce Version 2008 R2

2 What's new in MapForce Version 2008 R2

New features in MapForce Version 2008 Release 2 include:

- Office Open XML Excel 2007 (*.xslx) support as source and target components
- 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
- Support for SOAP version 1.2 in Web services
- New Repeated split option "Starts with..." in FlexText
- Ability to strip database schema names from generated code
- SQL SELECT Statements as virtual tables in database components
- Local Relations - on-the-fly creation of primary/foreign key relationships
- Support for Altova Global Resources
- Performance optimizations

2.1 What's new in MapForce Version 2008

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

2.2 What's new in MapForce Version 2007 R3

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
- Full support for all EDI X12 releases from 3040 to 5030
- Full support for UN/EDIFACT messages of directories 93A to 06B
- Code generator optimization and improved documentation

Chapter 3

MapForce overview

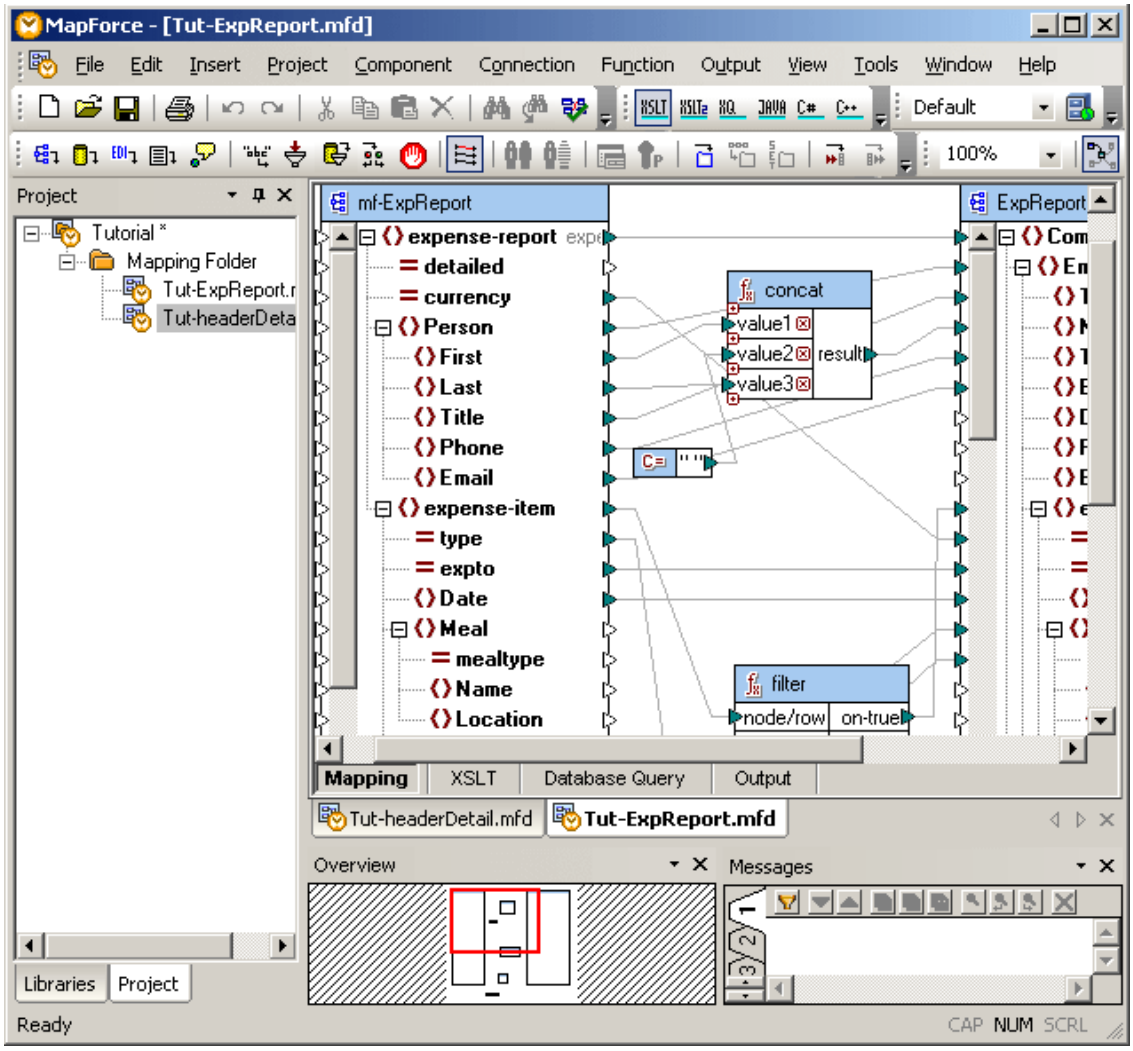
3 MapForce overview

MapForce has four main areas: the Library pane at left, the Mapping tab group at right, as well as the Overview and Messages panes below. The actual mapping process is achieved by manipulating the on-screen graphical elements in the mapping window.

- The **Libraries** pane displays language specific and user defined libraries, as well as the individual library functions. Functions can be directly dragged into the Mapping tab. The **Add/Remove Libraries...** button allows you to import external libraries.
- The **Mapping** tab displays the graphical elements used to create the mapping (transformation) between the two schemas. The source schema is the "**mf-ExpReport**" component window displaying the source schema tree. The target schema is the " **ExpReport-Target**" window displaying the target schema tree. **Connectors** connect the input and output **icons** of each schema item. Schema **items** can be either elements or attributes.

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

  The **Output** tab displays a preview of the transformed, or mapped data, in a text view.
- 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.


MapForce - [Tut-ExpReport.mfd]
File Edit Insert Project Component Connection Function Output View Tools Window Help
Default
100%
Project
Tutorial *
Mapping Folder
Tut-ExpReport.r
Tut-headerDeta
mf-ExpReport
expense-report
detailed
currency
Person
First
Last
Title
Phone
Email
expense-item
type
expto
Date
Meal
mealtype
Name
Location
concat
value1
value2
value3
result
filter
node/row
on-true
ExpReport
Mapping
XSLT
Database Query
Output
Tut-headerDetail.mfd
Tut-ExpReport.mfd
Overview
Messages
Libraries
Project
Ready
CAP NUM SCRL

3.1 Terminology

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.

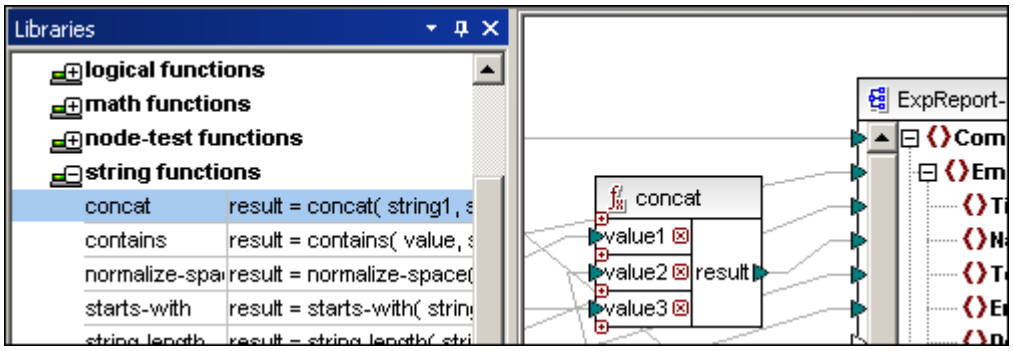
Component
In MapForce a component is a very generic "object". Almost all graphical elements you can insert/import or place in the Mapping tab, become components.

Components are recognizable by the small **triangles** they possess. These triangles (**input** and **output icons**) allow you to map data by creating a connection between them.

The following graphical elements are all components:

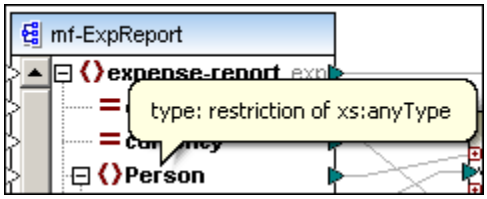
- All schema types: Source and target schemas
- All database types: Source and target databases
- All flat files: CSV and other text files
- All EDI documents UN/EDIFACT and ANSI X12: Source and target documents
- All Excel 2007 source and target files
- All function types: XSLT/XSLT2, XQuery, Java, C#, and C++ functions, as well as Constants, Filters and Conditions

Function
A function is basically an operation 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. Dragging a function into the Mapping window creates a function component. Please see the section Functions and Libraries for more details.



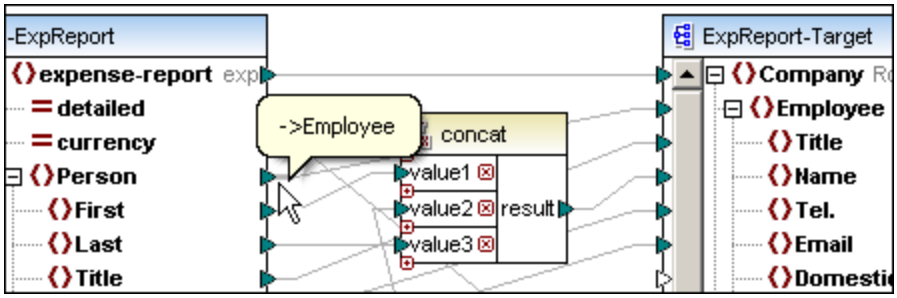
Item
An item is the unit of data that can be mapped from component to component. An item can be either an **element**, an **attribute**, a database field, or an EDI segment.

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



"Missing" item:
If items have been deleted from one of the components, e.g. a schema element/attribute is deleted from an XML schema, MapForce automatically creates placeholder items and retains the previous connectors.

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. 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: Number, and All other (String).

Filter: Node/Row
A filter is 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/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.

3.2 MapForce components

When creating a mapping, single, or multiple **data sources**, can be mapped to multiple **target components.**

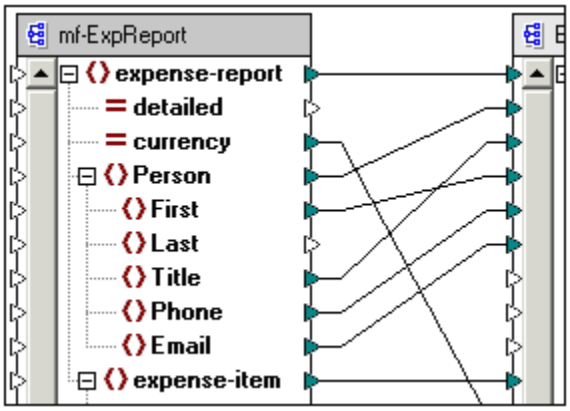
- Data sources can be: XML-Schemas/documents, CSV or text files, databases, as well as EDI messages.
- Target components can be: XML-Schemas/documents, CSV or text files, EDI messages, as well as databases.
- Target components have their own individual encoding settings. Double click the target component to see or change these settings. Please see Component for more information.

The mapping process allows the source data to be selectively transformed (or manipulated using functions) before it is output, or made available in the Output preview window.

A **data source component** can have an XML-Schemas/document database, CSV or text file, and/or an **UN/EDIFACT message** as its source. Once a data source has been imported/converted, it is used in exactly the same way as any of the other schema components in the Mapping tab.

To create a schema component (source):

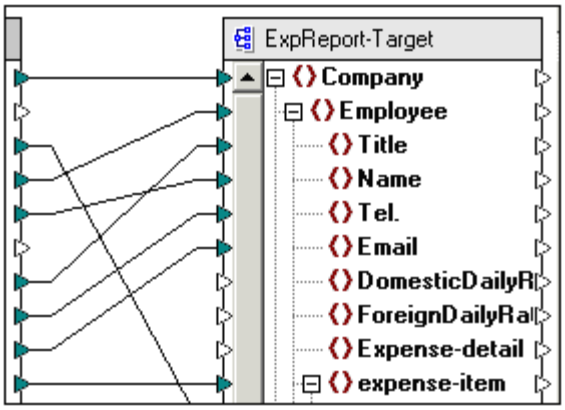
1. Click the **Insert | XML Schema/File** icon .
2. Select the schema file you want to use, from the "Open", dialog box.
   A further dialog box appears prompting you to select an **XML sample** file, if you intend to use this schema as a **data source** in this mapping.
3. Click **Browse...** if this is the case, and select the **XML instance** file.
   The schema component now appears in the Mapping tab.



You can now connect the schema source output icons, with the target (or function) input icons, to create your mappings.

To create a schema component (target)

1. Click the **Insert | XML Schema/File** icon .
2. Select the schema file you want to use from the "Open", dialog box.
   Select **Skip** when you are prompted to supply an **XML sample** file.
3. Select the **Root element** of the schema you want to use (Company) and click OK.



The schema (snippet) with the root element appears as a schema component.

The **target schema** is the basis of the XML document you want to have **generated** by the transformation.

The target schema/document can, of course, differ dramatically from the source schema. This is where the mapping process comes in, you can map any item in the source schema/database to any other item (element/attribute), in the target schema. The source data then appears at the position defined by your mapping, in the target document.

You can also define multiple output schemas. MapForce then generates XSLT, XQuery, or program code for each target schema. You can then selectively preview the different output schemas in the Output preview window, please see the section "XSLT and Output previews" for more information.

Please note:

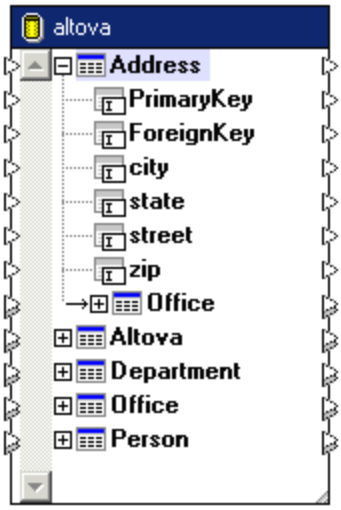
It is not necessary to associate an XML Instance document to a **target schema**. If you do so, then the XML instance document is ignored and does not affect the transformation in any way.

Clicking the root element of a schema and hitting the ***** key on the numeric keypad, expands all the schema items!

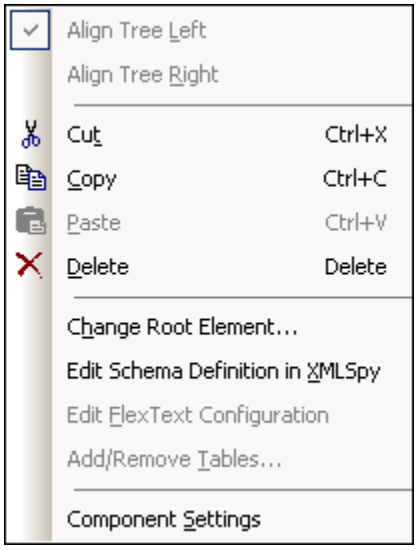
To create a Database component (source/target):

The database structure is the basis of the component and is displayed it in a tree view.

1. Click the **Insert Database** icon .
2. Select the source **database type** by clicking on one of the radio buttons (e.g. Microsoft Access), and click Next.
3. Click **Browse** to select navigate and select an Access database, (e.g. **Tutorial\altova.mdb**) and then click Next.
4. Select the database tables you want to import, or have access to (Select All).
5. Click the **Insert Now** button at the bottom of the dialog box.
   The database component now appears in the Mapping window.



Schema component **context menu**
Right clicking a schema component in the Mapping window opens the context menu.



Align tree left
Aligns all the items along the left hand window border. This display is useful when creating mappings from the source schema.

Align tree right
Aligns all the items along the right hand window border. This display is useful when creating mappings to the target schema.

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.

Change Root element
Allows you to change the root element of the XML instance document. Useful in the target schema window, as this limits or preselects the schema data.

Edit Schema definition in XMLSpy.

Starts XMLSpy and opens the schema file, ready for you to edit.

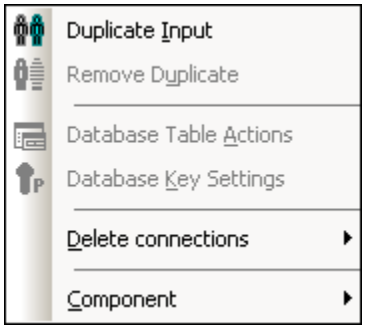
Edit FlexText Configuration
Starts FlexText and allows you to edit the FlexText component.

Add/Remove tables
Allows you to change the number of tables in the database component by selecting/deselecting them in the Database Tables group.

Component Settings
Opens the Component Settings dialog box.

Allows you to select the input and/or output XML Instance, as well as define database specific settings for code generation. Please see **Component Settings** for more information on these settings.

Item Context menu
Right clicking an **item** in a component, opens a context menu with options enabled for the specific type of component.



Duplicate input
Inserts a copy/clone of the selected item, allowing you to map multiple input data to this 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.

Please note:
Complex parameters of inlined user-defined functions/structures cannot be duplicated. Please see "Complex user-defined function - XML node as input" for more information.

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 table actions to be performed on the specific target database table. Table actions are: Insert, Update, and Delete, please see Mapping data to databases for more information.

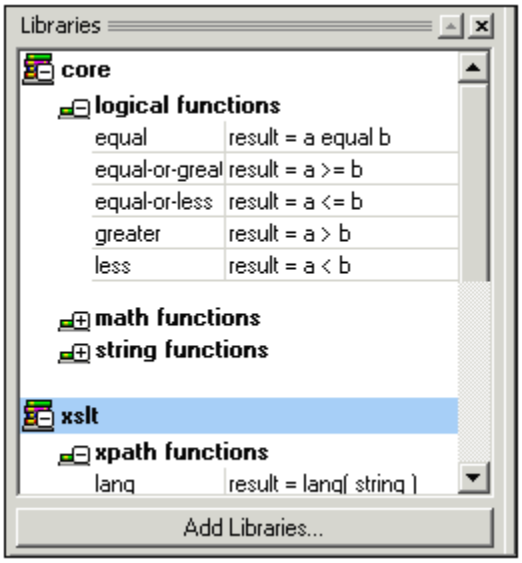
Database Key settings
Allows you to define the Key settings of database fields, please see Database Key settings for more information.

Delete connections
Allows you to delete connections related to that specific item: all direct connections, all incoming, or outgoing, child connections.

Component
Opens the Component Settings dialog box for the specific component.

3.3 Functions and libraries

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



The standard **core, lang, xpath2, edifact** 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

Selecting:
XSLT, enables the core and XSLT functions (XPath 1.0 and XSLT 1.0 functions).

XSLT2, enables the core, XPath 2.0, and XSLT 2.0 functions.

XQ(uery), enables the core and XPath 2.0 functions.

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

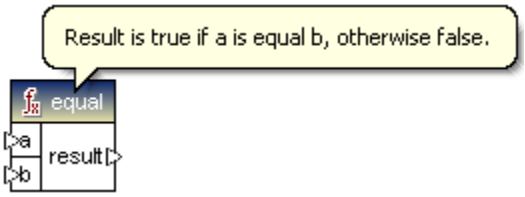
To use a function in Mapping window:

1. First select the **programming language** you intend to generate code for, by clicking one of the output icons in the title bar: XSLT/XSLT2 XQ, Java, C#, or C++.
   The functions associated with that language are now visible in the Libraries window.
   The expand and contract icons show, or hide the functions of that library.
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 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 add new function libraries:

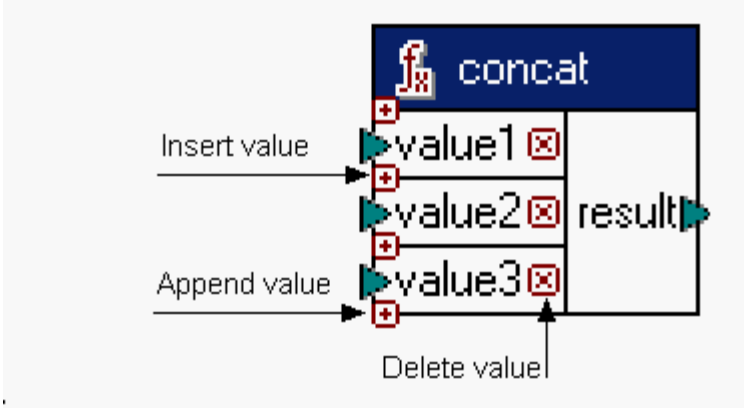
MapForce allows you to create and integrate your own function libraries, please see the sections: Adding custom function libraries, Adding custom XSLT 1.0 functions, Adding custom XSLT 2.0 functions and User-defined functions for more information.

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

Extendable functions

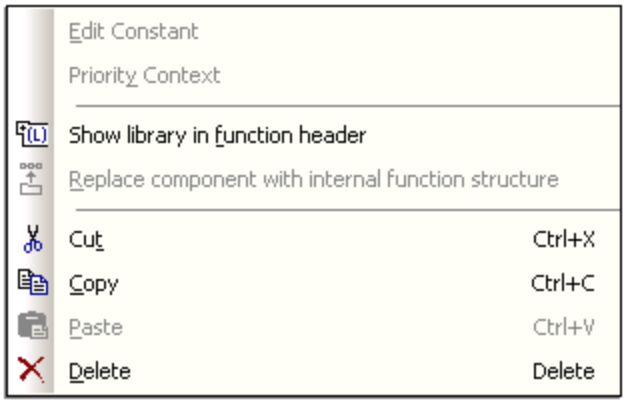
Several functions available in the function libraries are extendable: for e.g. the concat, "logical-and", "logical-or" functions. The parameters of these types of function can be inserted/appended and deleted at will. Clicking the "plus" icon inserts or appends the same type of parameter, while clicking the check mark deletes the parameter.

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



3.3.1 Function context menu

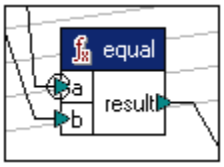
Function context menu:
Right clicking a function in the Mapping window, opens the context window.



Edit Constant
Allows you to change the entry currently defined in the Constant component. A Constant is added by clicking the **Insert Constant** icon  .

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.

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.

3.4 Projects

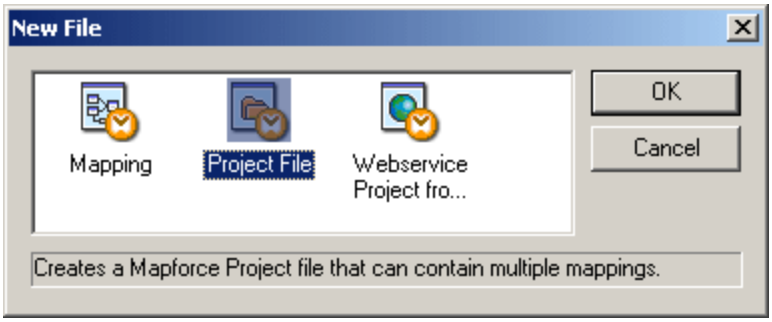
MapForce supports the Multiple Document Interface, and allows you to group your mappings into mapping projects. Project files have a ***.mfp** extension.

Two types of projects can be defined:

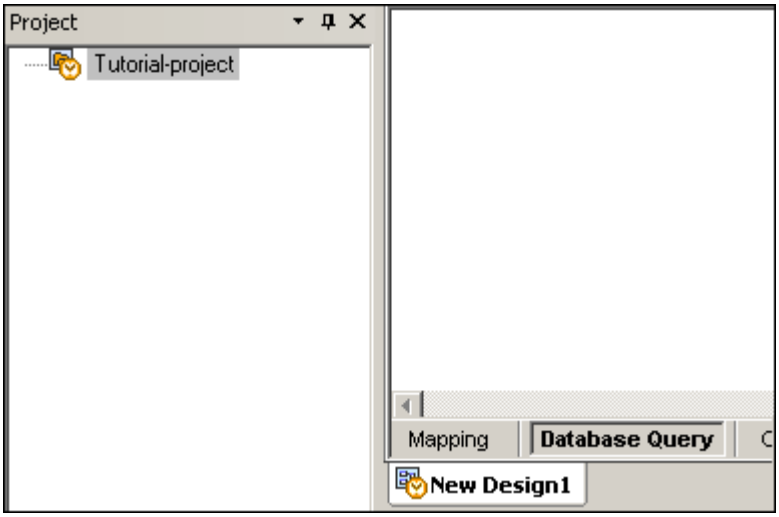
- A collection of individual mappings, i.e. a standard project
- A related set of mappings, which make up a WSDL mapping project
  Both project types support code generation for the entire project

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 Design pane.

To add mappings to a project:

1. 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.

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

To remove a mapping from a project:

1. Right click the mapping icon below the project folder,

2. Select Remove mapping from the pop-up menu.

To create a WSDL project:

A Web service project differs from a standard project in that a WSDL file is needed for its creation, and the result of the code generation process is a complete Web service. All that remains, is to compile the generated code, and deploy the Web service to your specific webserver.

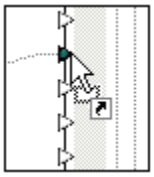
Each operation defined in the WSDL file, is presented as an individual mapping.

1. Select **File | New,** and double click the Web service Project from... icon.
2. Fill in the New Project dialog box, fields marked with an asterisk are mandatory. Please see the section MapForce and Web services for more information.

3.5 Mapping between components

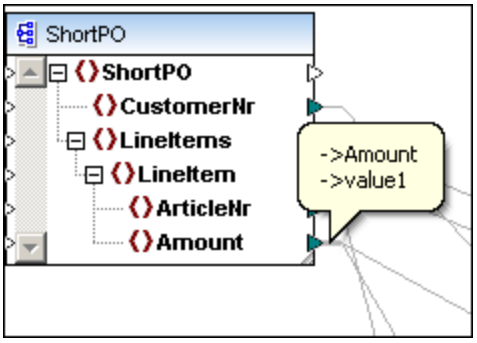
A **connector** visualizes the **mapping** between the two sets of data and allows the **source** data (value) to appear, or be transformed, into the target schema/document or database.

- **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.

  Clicking the straight section of the connector highlights it and allows you to reposition it by dragging it elsewhere.

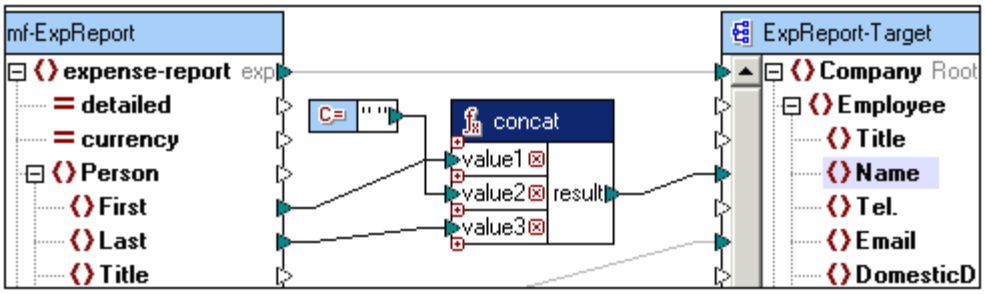


Number of connectors
Input and output icons appear on most components, there is not, however, a one to one relationship between their numbers.

- 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.

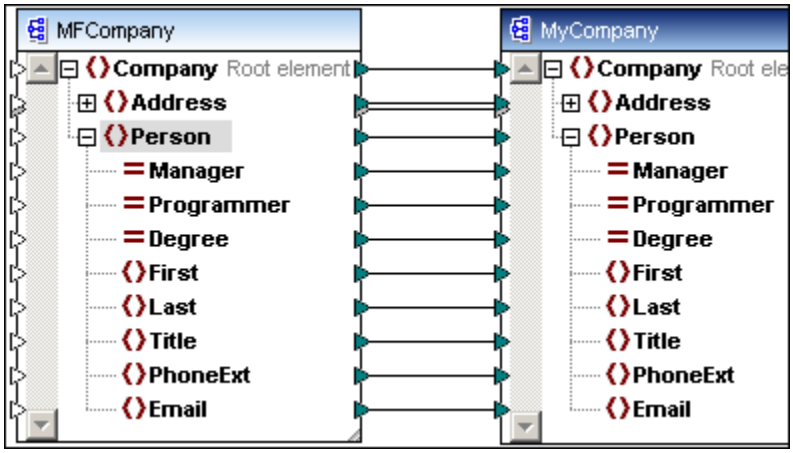
This example shows how you can use the **concat** function to combine the First and Last names and place the result in the Title element. The constant component supplies the space character between the two names.



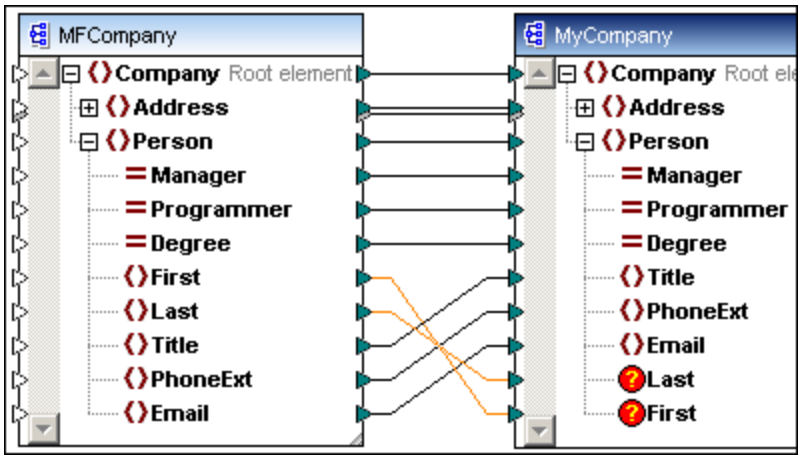
3.5.1 Missing items

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.



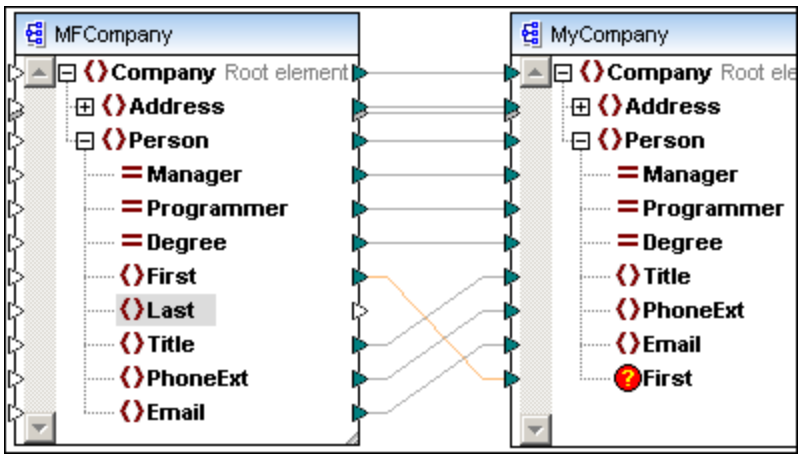
While editing MyCompany.xsd, in XMLSpy, the First and Last items in the schema are deleted. Returning to MapForce opens a Changed Files notification dialog box, prompting you to reload the schema. Clicking Reload updates the components in MapForce.



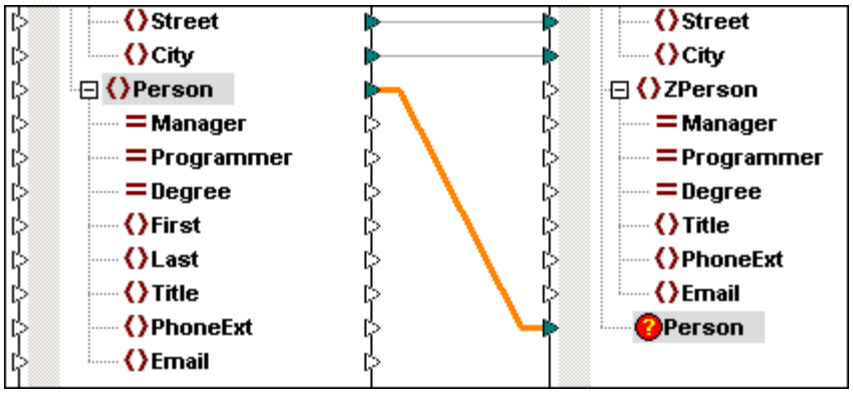
The deleted **items** and their **connectors** are now marked in the MyCompany component. You could now reconnect the connectors to other items if necessary, or delete the connectors.

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.

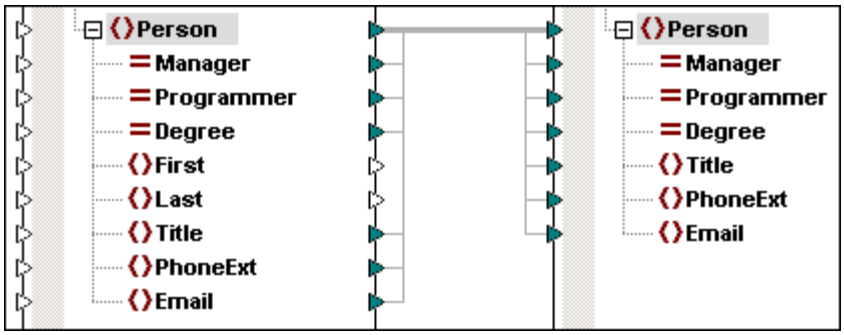
Clicking one of the highlighted connectors and deleting it, removes the "missing" item from the component, e.g. Last, in MyCompany.



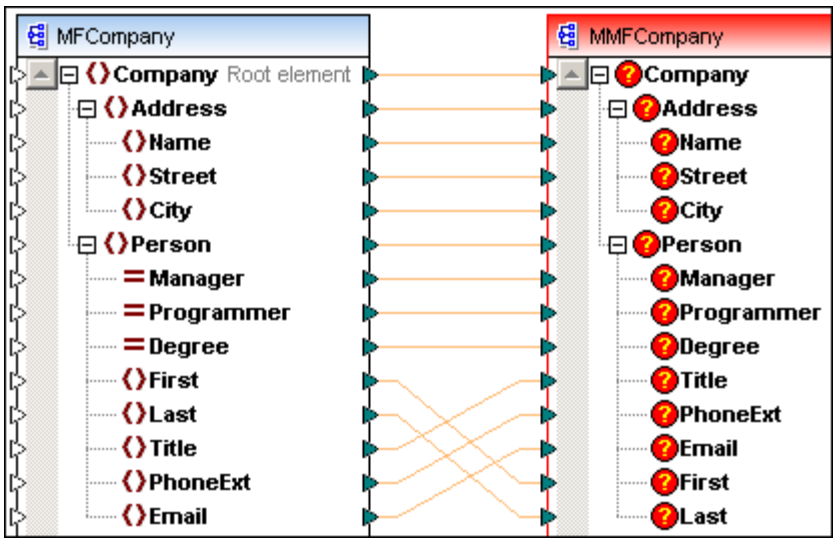
Renamed items:
If a parent item is renamed e.g. Person to ZPerson, then the original parent item connector is retained and the child items and their connectors are deleted.



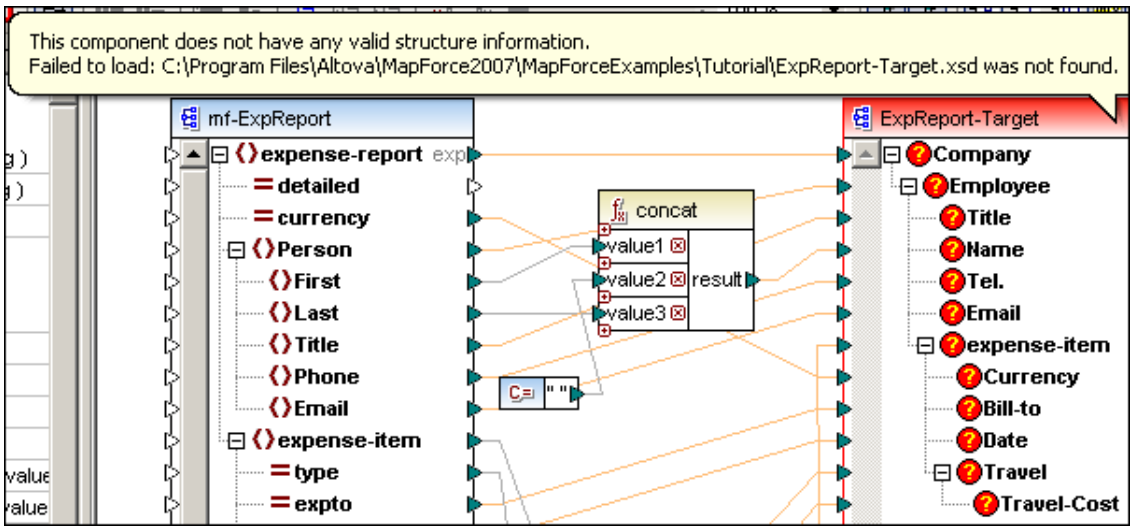
"Copy all" connectors and missing items:
Copy all connections are treated in the same way as normal connections, with the only difference being that the connectors to the missing child items are not retained or displayed.



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.

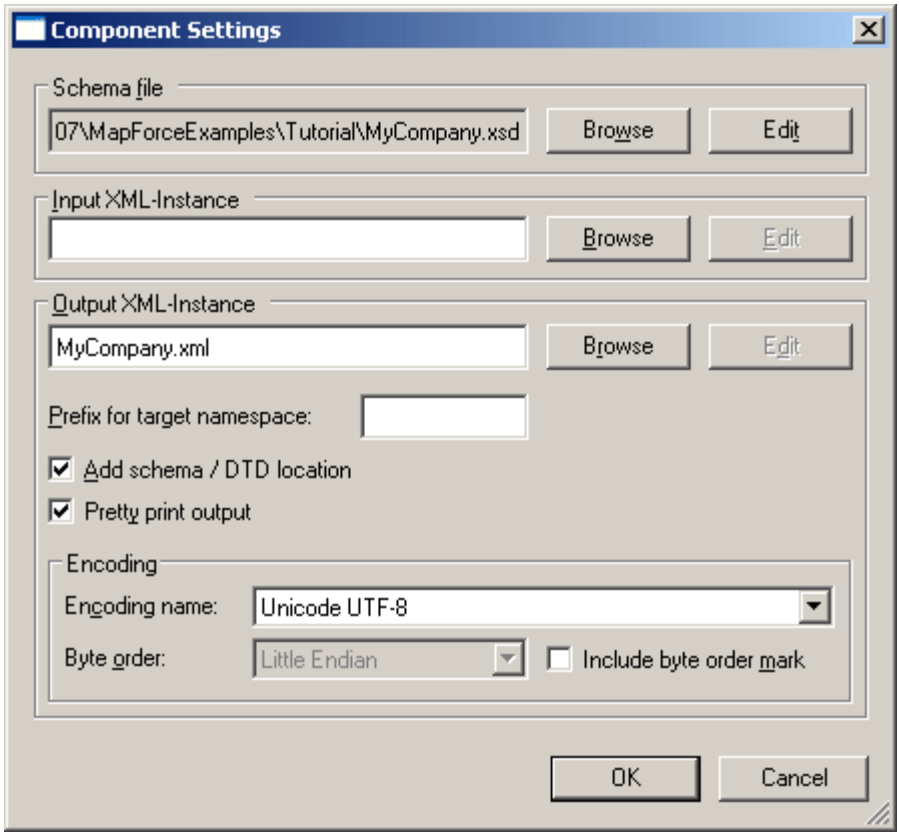


Placing the mouse cursor over the highlighted component, opens a popup containing pertinent information.



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.

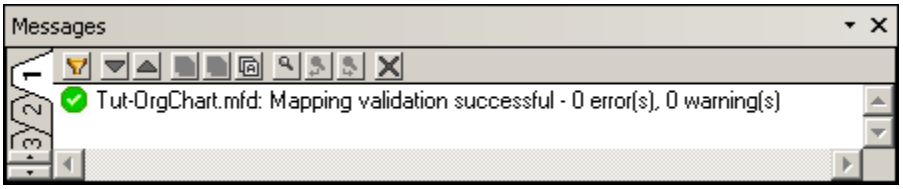
3.6 Validating mappings and mapping output

Connectors and validation
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.

To validate your mapping:

- Click the Validate Mapping icon  , or select the menu item **File | Validate Mapping.**
- Click one of the preview tabs, (XSLT, XSLT 2.0, or Output), or
- Select the menu option **File | Generate XSLT/XSLT2, Generate XQuery, Java, C#, or C++ code**

A validation message appears in the Messages window.



Note that you can use multiple message tabs if you 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.

Validation messages:

- Validation successful - X Error(s), Y 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 Y warnings.

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.

Component connections and validation results:

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.

Validating mapped 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. If the target component is an EDI file, then the output is validated against the EDI specification, please see: UN/EDIFACT and ANSI X12 as target components for more information.

- Click the Validate button  to validate the document against the schema. A "Output XML document is valid" message, or a message detailing any errors appears.

3.7 XSLT, Output tab - generating XSLT or program code

The XSLT, XSLT2, XQuery, and Output tabs of the Mapping tab group supply a **preview** of:

- the generated XSLT or XQuery code and
- the resulting transformation produced by the MapForce engine.

Please note:
The result generated by the MapForce engine, is an on-the-fly transformation of database, Text, or EDI data, 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. The generated program code will have a much faster execution speed.

To save the generated 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.
3. A message appears when the generation was successful.
4. Navigate to the previously defined folder, where you will find the generated XSLT file.

To save the XML, or output data from the Output tab:

1. Click the Output tab to preview the mapping result.
2. Click the "**Save generated output**" icon  , and specify where you want the result to be saved.

If the target is an **XML/Schema** file:

- The Save generated output icon is active. Click it to save the output.

If the target is a **Database**:

- The Run SQL-script icon is active. Click it to update, insert, or delete the database data.

To transform an XML/Schema file using the generated XSLT:

1. Open the XML file in the editor of your choice (**XMLSpy** for example).
2. Assign the XSLT file to the XML file (**XSL/XQuery | Assign XSL**).
3. Start the transformation process (**XSL/XQuery | XSL Transformation**).
   The transformed XML document appears in your editor.

To generate program code:

1. Select the specific menu option: **File | Generate code in | XSLT/XSLT2, XQuery, Java, C#, C++**)
2. Browse for the folder where you want to save the program files.
3. A message appears when the code generation was successful.
4. 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.

To search for specific data in the Output tab:

- Select the menu option **Edit | Find**, or hit the **CTRL+F** keyboard keys.
  The Find dialog box allows you to specify the search options in great detail, and also supports regular expressions.

3.8 Generating XQuery 1.0 code

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

Please note that execution speed of generated XQuery 1.0 code is significantly faster that of generated XSLT 1.0 / 2.0 code. Generated program code such as Java, C#, or C++, is of course even quicker, because it is compiled before execution.

To download the AltovaXML package (which contains the Altova XQuery engine):

- Point your Browser to http://www.altova.com/download_components.html then select and install the AltovaXQuery engine.

This example uses the **Tut-ExpReport.mfd** file supplied in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. Make sure that you have selected the XQuery  icon before you preview the results.

To preview an XQuery result:
Before generating program code it is a good idea to preview the result of the XQuery using the MapForce Engine.

Having opened the Tut-ExpReport.mfd file in MapForce:

1. Click the **XQuery** tab to preview the generated XQuery code.
2. Click the **Output** tab to preview the result of the mapping.

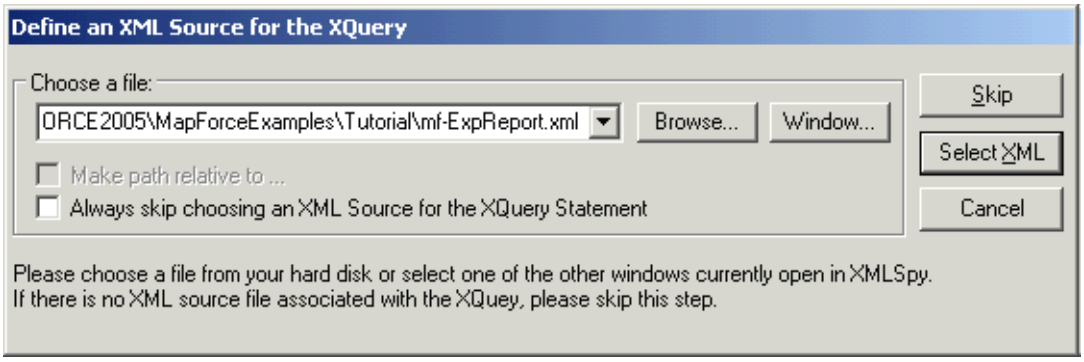
```
 1 (:
10
11   xquery version "1.0";
12
13   declare namespace a = "http://my-company.com/namespace";
14
15
16
17   for $expense-report in /expense-report
18     let $V1 := $expense-report
19   return
20     <a:Company>
21     {
22       attribute xsi:schemaLocation
23       {
26       ,
27       for $Person in $expense-report/Person
28         let $V2 := $Person
29       return
```

To generate XQuery code:

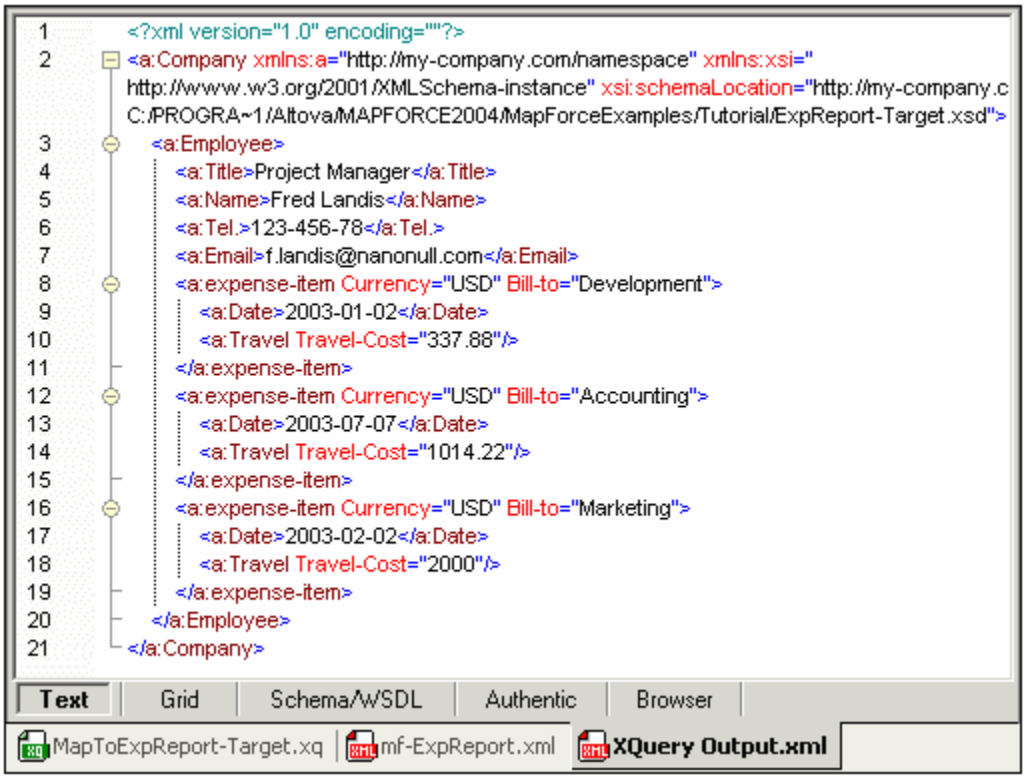
1. Open the **Tut-ExpReport.mfd file** in MapForce.
2. Select the menu item **File | Generate code in | XQuery**.
3. Select the folder you want to place the generated XQuery file in, (e.g. **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples**) and click OK.
   A message appears showing that the generation was successful.
4. Navigate to the designated folder and you will find the XQuery file with the file name **MapToExpReport-Target.xq**.

To execute the XQuery using XMLSpy:

1. Start XMLSpy and open the previously generated MapToExpReport-Target.xq file.
2. Select the menu option **XSL/XQuery | XQuery execution**.



3. Click the Browse button and select the XML file that is to act as the data source, e.g. mf-ExpReport.xml.
4. Click the "Select XML" button to execute the XQuery.



An "XQuery Output.xml" file is created which contains the mapped data. Please see the **Altova XQuery Engine** documentation for more information on the command line parameters.

3.9 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 documenation.

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

Chapter 4

MapForce tutorial

4 MapForce tutorial

Tutorial example:
In the tutorial, a simple employee travel expense report will be mapped to a more complex company report.

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.

Further formatting, cost summation, and conditional viewing options of the expense report, are made possible by having the target XML document associated with a StyleVision Power Stylesheet designed in StyleVision.

Aim of the tutorial:

- To transform the **personal expense report** to a company expense travel report
- **Selectively filter** the source data and only let the travel expense records through
- Generate an XSLT transformation file
- Transform the personal expense report to the company expense report using the generated XSLT file
- Assign a StyleVision Power Stylesheet to the resulting XML file, enabling you to view and edit the resulting file in the Authentic View

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

Files used in the tutorial:
All the files used in this tutorial are available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. The XSLT and transformed XML files are also supplied.

If multiple users use the same PC, and a different user logs on, a message box opens and prompts the new user if the installer should add the necessary files for that user. If yes, then the **example** files for that user are placed in the **...\User folder\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial.**

Tutorial files:	**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
mf-ExpReport.sps	StyleVision Power Stylesheet used to view the personal expense report in Authentic View of XMLSpy, or Authentic Desktop.
	Company expense report
ExpReport-Target.xml	Company expense report XML instance document
ExpReport-Target.xsd	Associated schema file
ExpReport-Target.sps	StyleVision Power Stylesheet used to view the Company expense report in Authentic View of XMLSpy, or Authentic Desktop.

Personal Expense Report

Currency: (•) Dollars ( ) Euros ( ) Yen Currency $

[x] Detailed report

Employee Information

First Name	Last Name	Title
Fred	Landis	Project Manager

E-Mail	Phone
f.landis@nanonull.com	123-456-78

Expense List

Type	Expense To	Date (yyyy-mm-dd)	Expenses $		Description
Travel	Development	2003-01-02	Travel: 337.88	Lodging: add Lodging	Biz jet
Lodging	Sales	2003-01-01	Travel: add Travel	Lodging: 121.2	Motel mania
Travel	Accounting	2003-07-07	Travel: 1014.22	Lodging: add Lodging	Ambassador class
Travel	Marketing	2003-02-02	Travel: 2000	Lodging: add Lodging	Hong Kong
Meal	Sales	2003-03-03	Travel: add Travel	Lodging: add Lodging	For Free

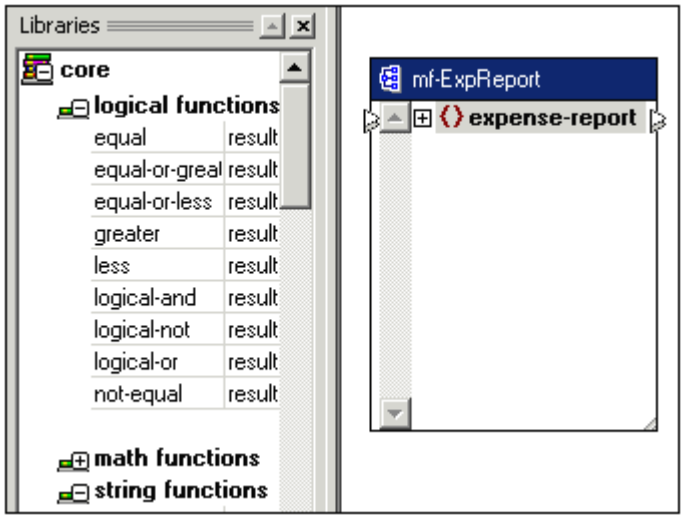
4.1 Setting up the mapping environment

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

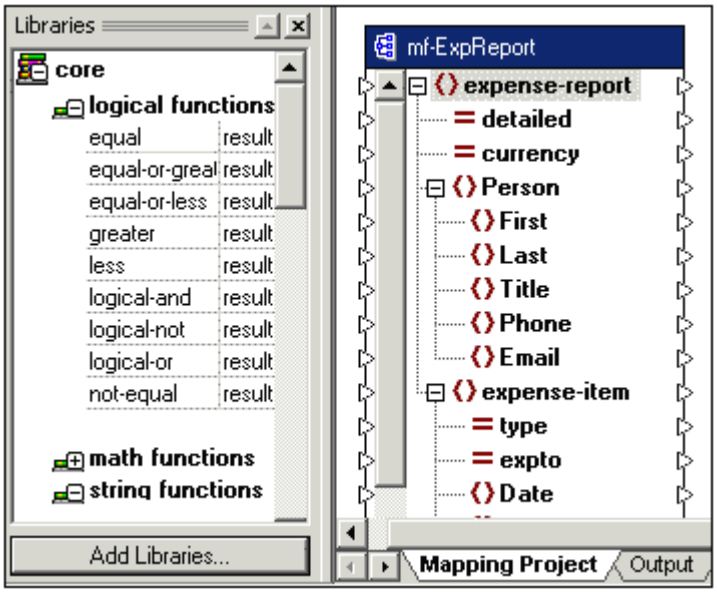
- Start MapForce.

Creating the source schema component:

1. Click the **Insert XML Schema/File** icon.
2. Select the **mf-ExpReport.xsd** file from the Open dialog box.
   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 tab.

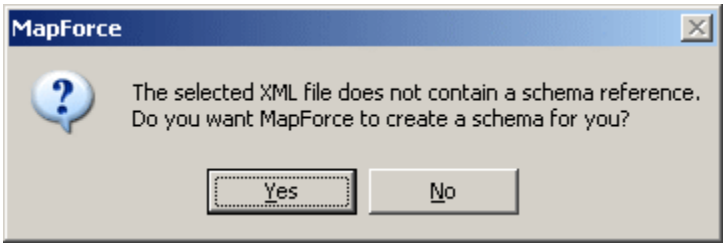


4. Click the **expense-report** entry and hit the * key, on the numeric keypad, to view all the items.
5. Click the expand icon at the lower right of the component window, and resize the window.



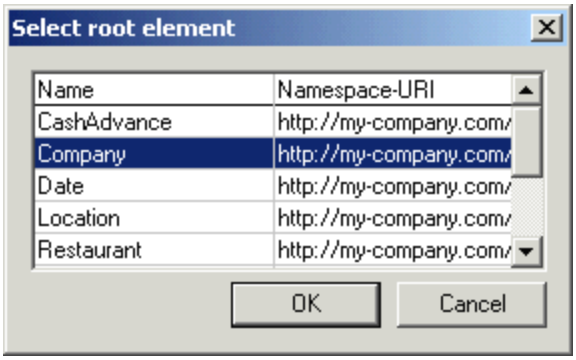
Please note:

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.



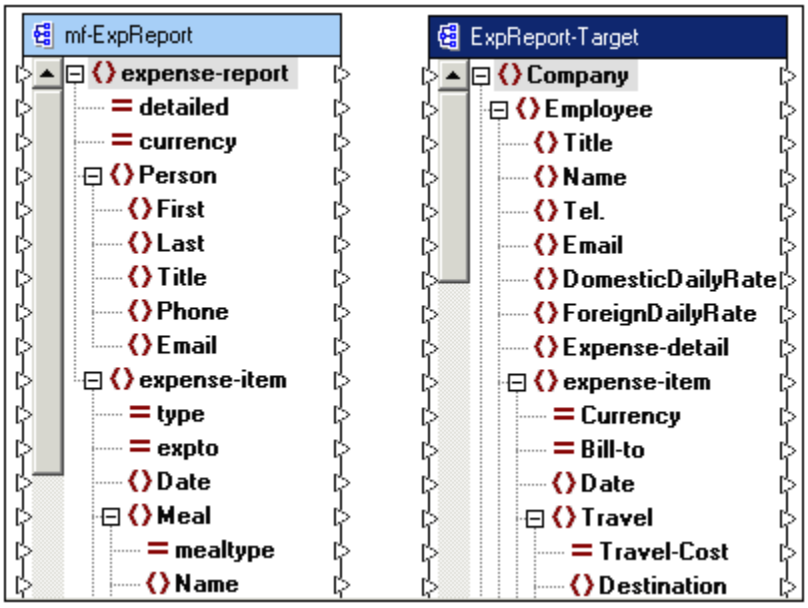
Creating the target schema component:

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 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. Click the expand window icon and resize the window.



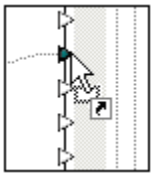
We are now ready to start mapping schema items from the source to the target

schema.

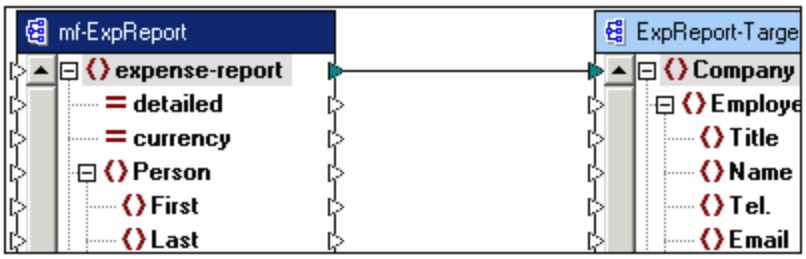
4.2 Mapping schema items

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

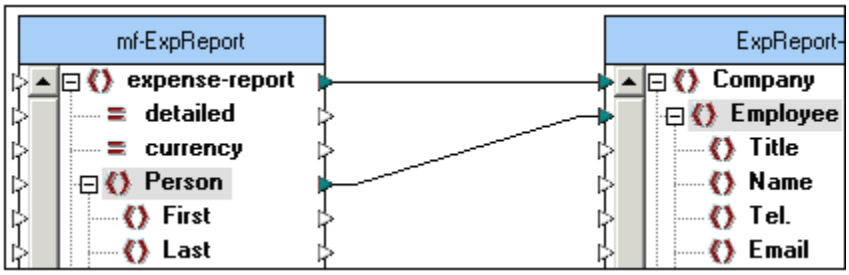
1. Click the **expense-report** item in the source schema and drag.
   A connector line is automatically created from the output icon and is linked to the mouse pointer which has now changed shape.
2. Move the mouse pointer near to the **Company** item in the target schema, and "drop" the connector the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



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.

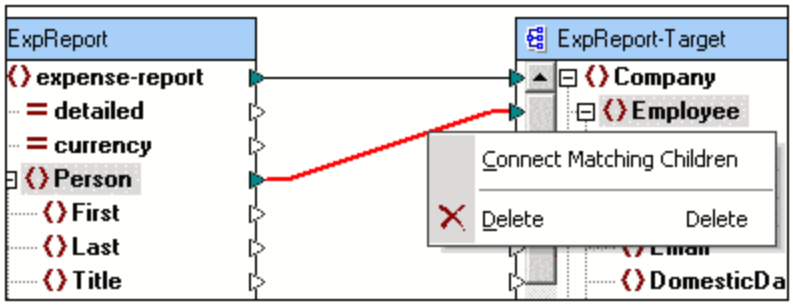


3. Use the above method to create a mapping between the Person and Employee items.

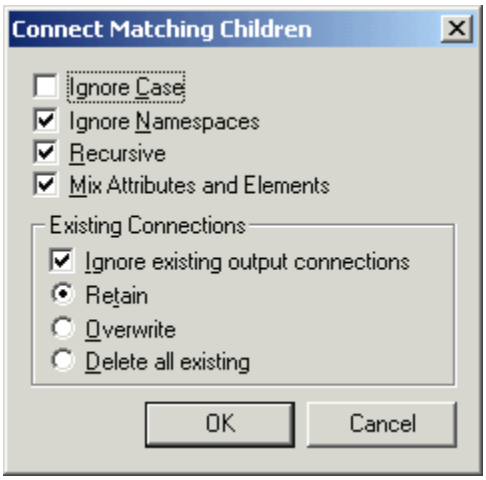


Auto-mapping
MapForce allows you to automatically connect child elements of the same name in both schemas. For more information please see the section on Connector properties.

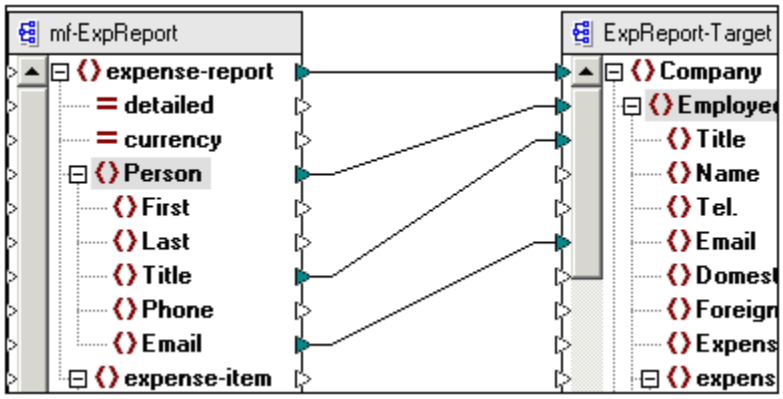
1. Right click the "Person" connector and select "Connect matching children" from the pop-up menu.
   If the child items are automatically connected, auto connect child items is active.



This opens the Connect Matching Children dialog box.

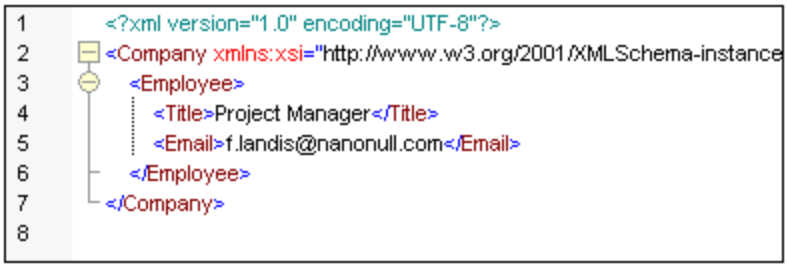


2. Activate all four check boxes, and click OK.



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

3. Click the Output tab to see if there is a result.



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

4. Click the Mapping tab to 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.

4.3 Using functions to map data

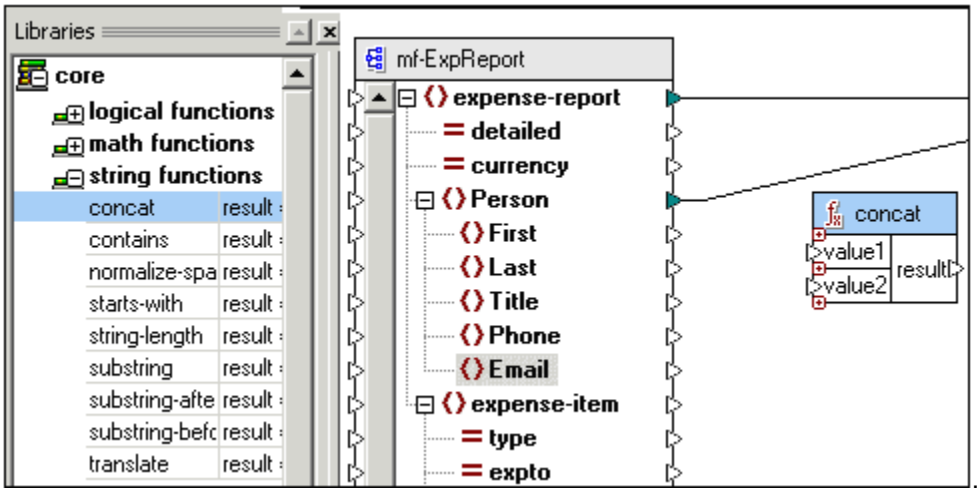
The aim of this section is to combine two sets of data from the source schema, and place the result in a single item in the target document. Please note, that some of the previously defined mappings are not shown in the following screen shots for the sake of clarity.

This will be done by:

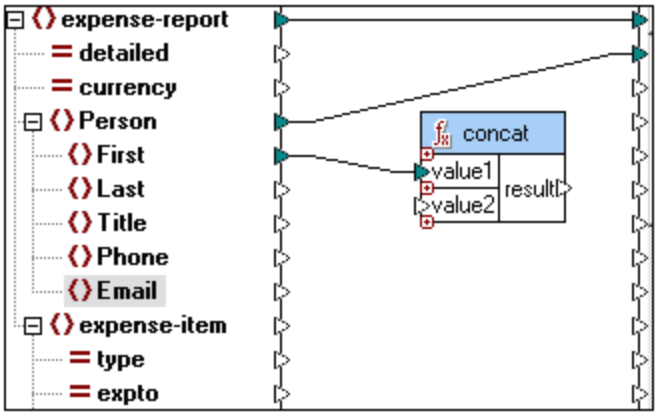
- Using the **Concat** string function to combine the **First** and **Last** elements of the source schema
- Using a **Constant** function to supply the space character needed to separate both items
- Placing the result of this process into the **Name** item of the target schema.

Using functions to combine items:

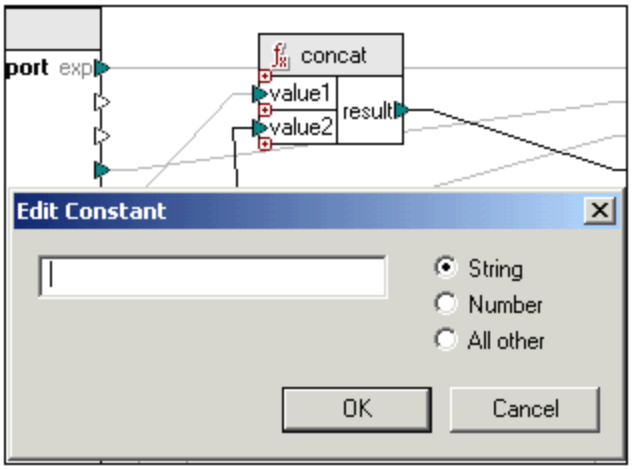
1. Click the **concat** entry of the string functions group, in the Core library, and drag it into the Mapping tab



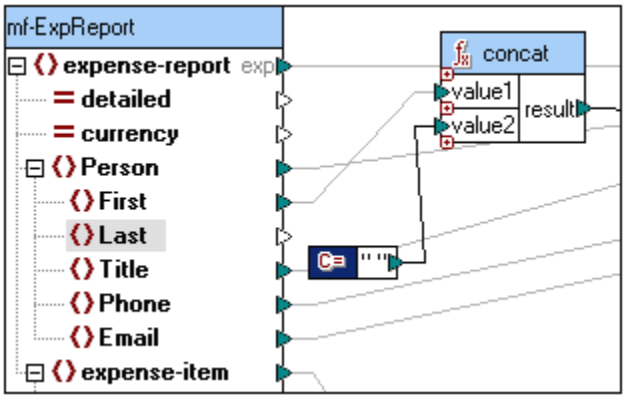
2. Create a connection between item **First** and **value1** of the concat component.



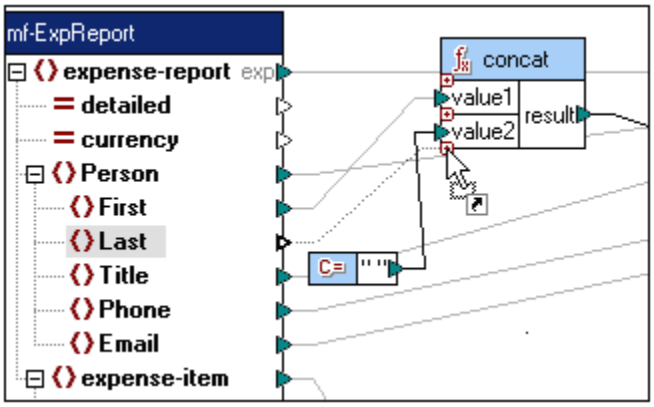
3. Click the **Insert Constant** icon  in the icon bar, to insert a constant component.



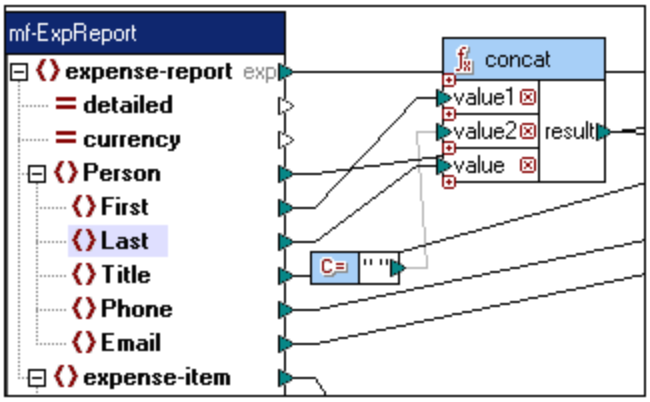
4. Enter a space character in the text box and click OK.
   The constant component is now in the working area. Its contents are displayed next to the output icon.
5. Create a connection between the **constant** component and **value2** of the concat component.



6. 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** tab to see the result of the current mapping.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    <Employee>
4      <Title>Project Manager</Title>
5      <Name>Fred Landis</Name>
6      <Email>f.landis@nanonull.com</Email>
7    </Employee>
8  </Company>
9
```

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:

1. Create mappings between the following items:
   - currency to Currency
   - Phone to Tel.
   - expto to Bill-to
   - Date to Date
2. Click the Output tab to see the result.

```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schema
  <Employee>
    <Title>Project Manager</Title>
    <Name>Fred Landis</Name>
    <Tel.>123-456-78</Tel.>
    <Email>f.landis@nanonull.com</Email>
    <expense-item Currency="USD" Bill-to="Sales">
      <Date>2003-01-02</Date>
      <Date>2003-01-01</Date>
      <Date>2003-07-07</Date>
      <Date>2003-02-02</Date>
      <Date>2003-03-03</Date>
    </expense-item>
  </Employee>
</Company>
```

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

Please note:
Functions can be grouped into user-defined functions/components to maximize screen space. 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.

4.4 Filtering data

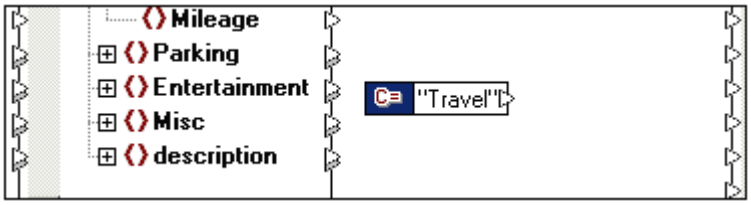
The aim of this section is to filter out the Lodging and Meal expenses, and only pass on the Travel expenses to the target schema/document.

This will be done by:

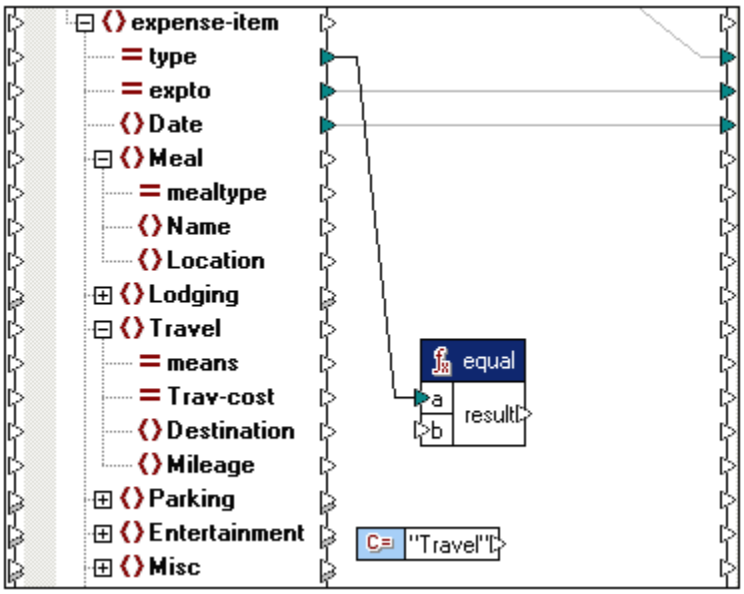
- Using the **Equal** function to test the value of a source item
- Using a **Constant** function to supply the comparison string that is to be tested
- Using the **Filter** function which passes on the Travel data, if the bool input value is true
- Placing the on-true result of this process, into the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

Filtering data:

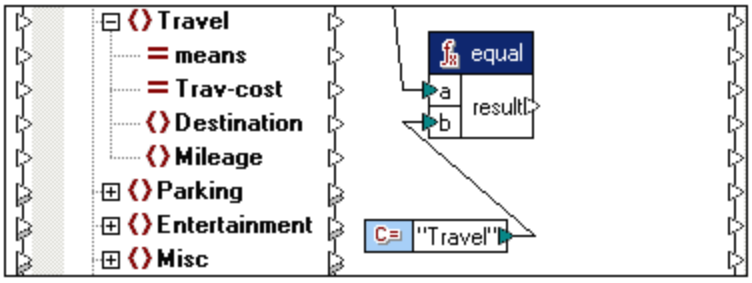
1. Insert a constant component and enter the string **Travel** in the input field.



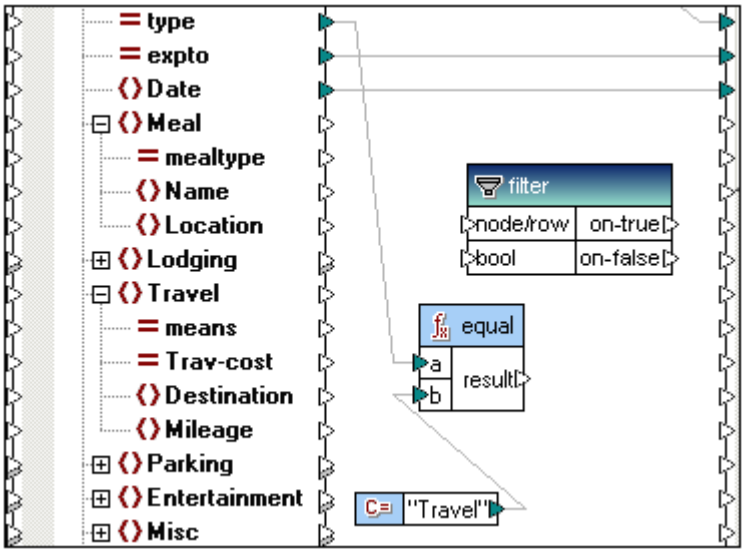
2. Insert the logical function **equal** from the core library (logical functions group).
3. Connect the (expense-item) **type** item in the source schema, to the **a** parameter of the equal function.



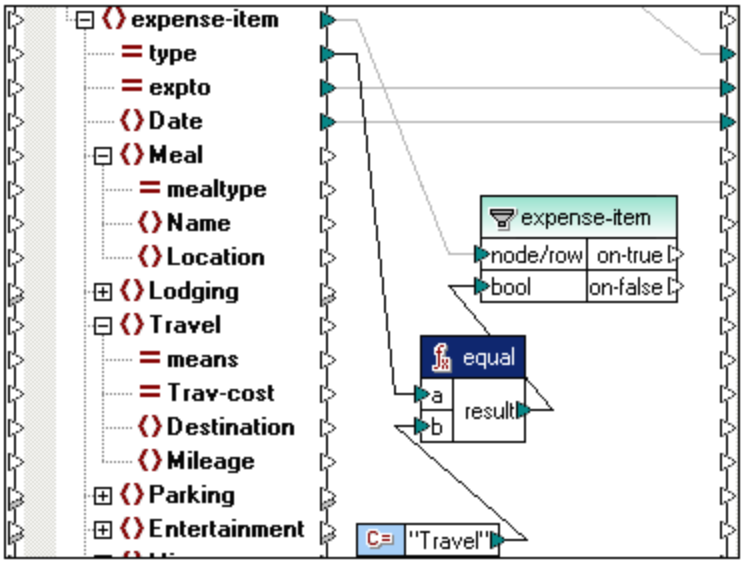
4. Connect the **result** icon of the constant component, to the **b** parameter of the equal function.



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



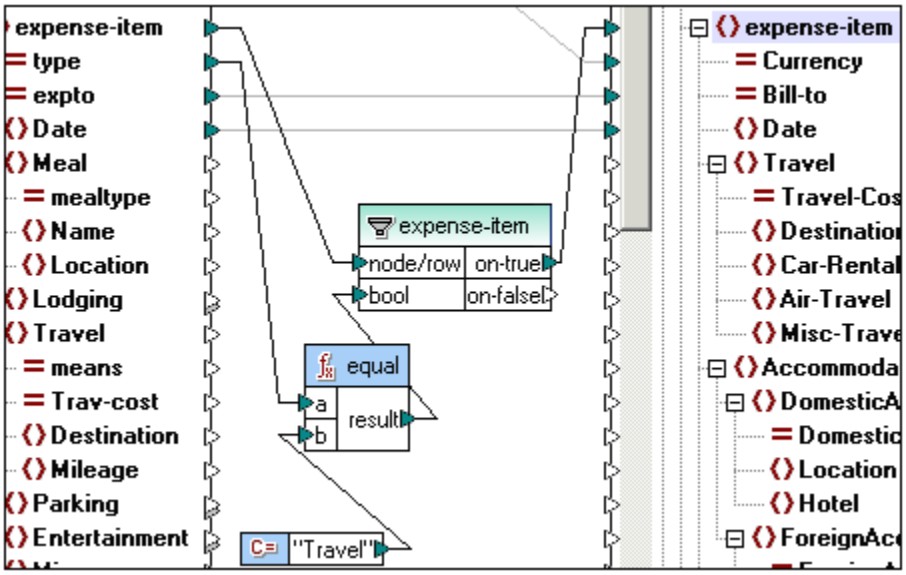
6. Connect the **result** icon of the **equal** component, to the **bool** parameter of the **filter** component.
7. Connect the **expense-item** icon of the source schema with the **node/row** parameter of the filter component.



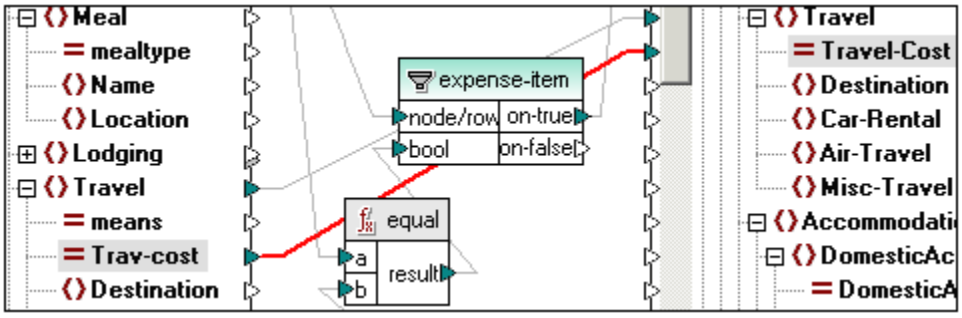
Note that the filter component name, now changes to "expense-item".

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

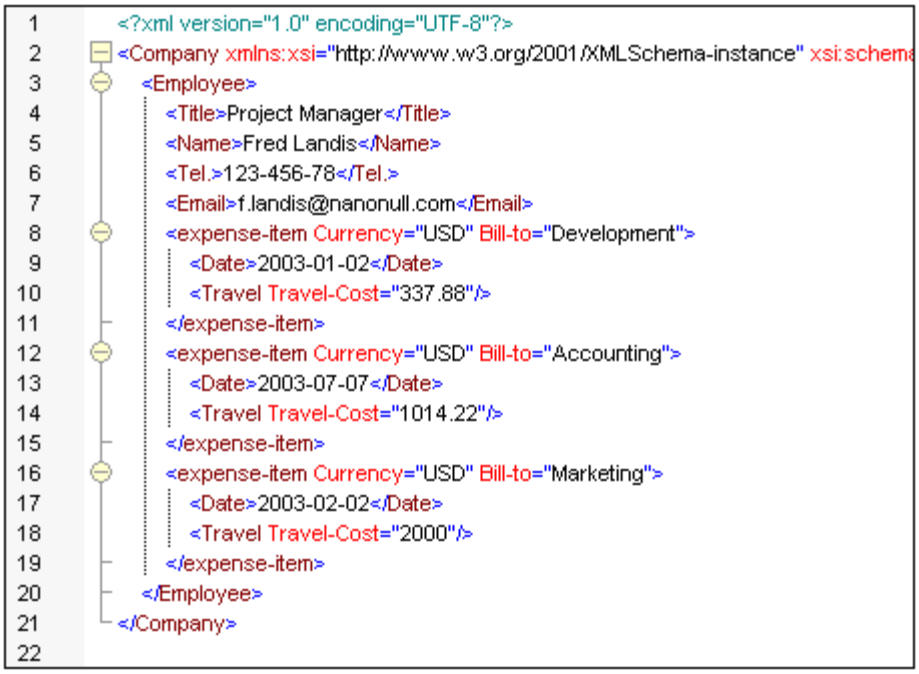
target document.



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 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: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.

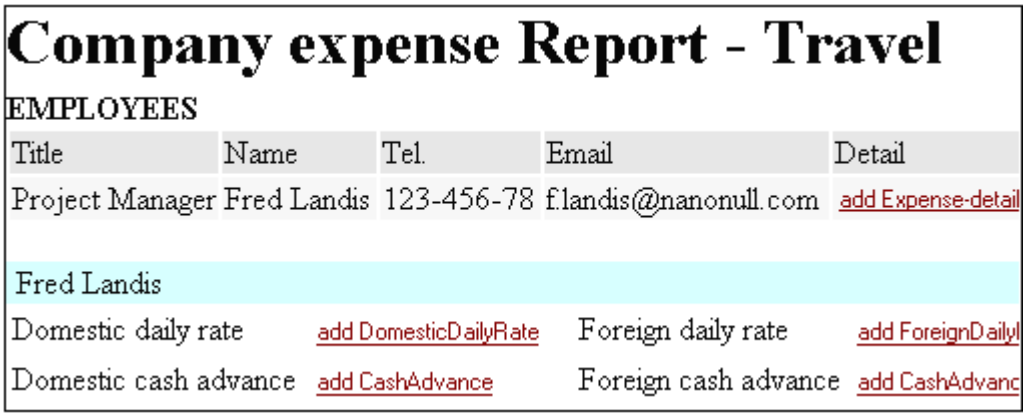
4.5 Generating XSLT 1.0 and 2.0 code

MapForce can generate two flavors of XSLT code:

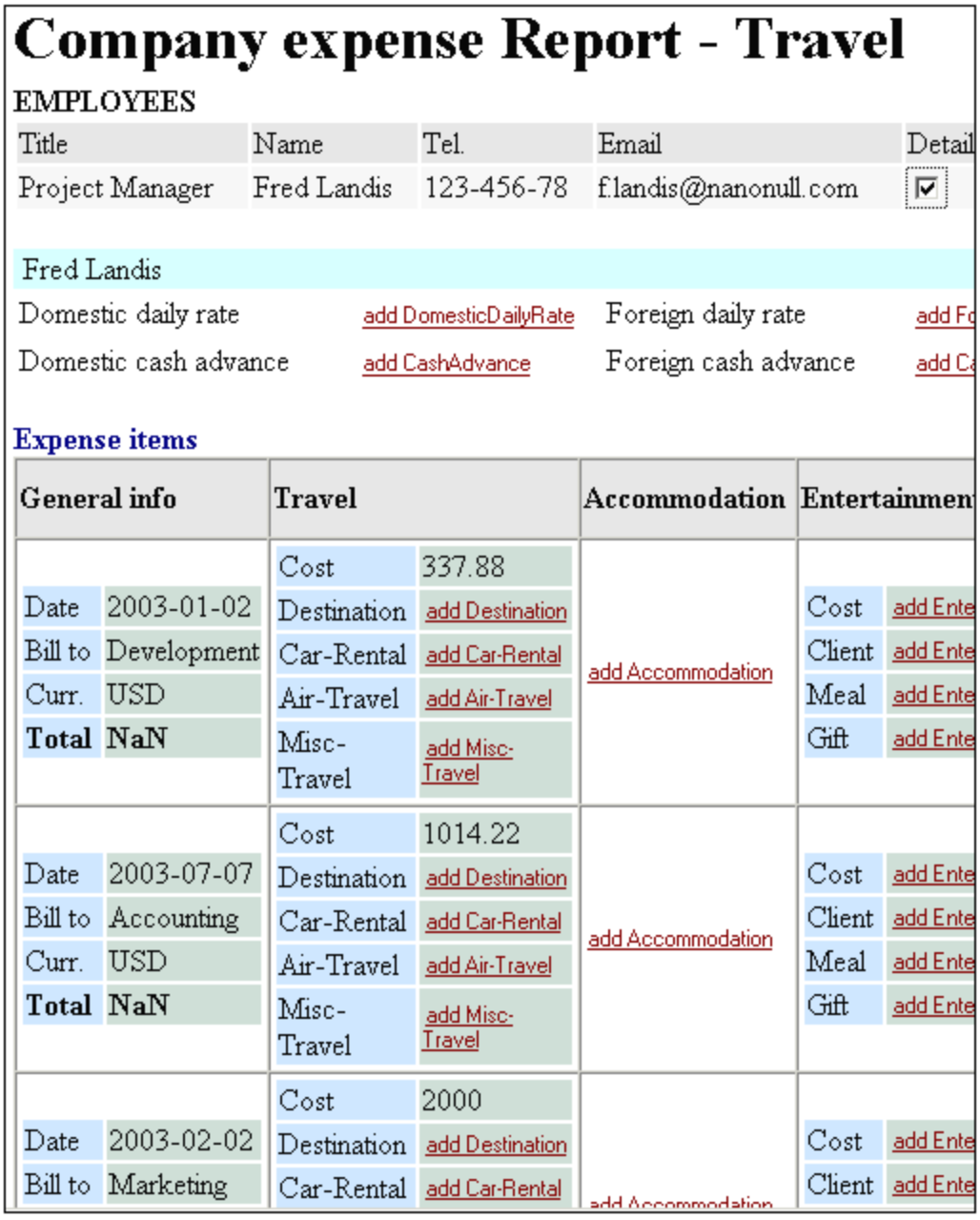
1. Select the menu item **File | Generate code in | XSLT 1.0 (or XSLT 2.0)**.
2. Select the folder you want to place the generated XSLT file in, and click OK.
   A message appears showing that the generation was successful.
3. Navigate to the designated folder and you will find the XSLT with the file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

To transform the personal expense report to the company expense report:
Having installed either XMLSpy, or Authentic Desktop you can easily transform the source to the target document.

1. Start XMLSpy, or Authentic Desktop and open the supplied **mf-ExpReport.xml** document.
2. Select the menu option **Tools | Options** and click the **XSL** tab.
3. Enter **.xml** in the *Default file extension of output file field*, and click OK.
4. Select the menu option **XSL/XQuery | XSL Transformation**.
5. Select the previously generated **MapToExpReport-Target.xslt** file, and click OK.
   An XSL Output.xml file is created.
   XMLSpy automatically selects the correct XSLT engine for the transformation.
6. Select the menu option **Authentic | Assign a StyleVision Power Stylesheet**.
7. Select the supplied stylesheet **ExpReport-Target.sps** and click OK.
8. Click the **Authentic** tab to switch to the Authentic view.



9. Click the **add Expense-detail** text in the Detail column.
   The field changes to a check box.
10. Click the check box to see the detailed expenses.



Company expense Report - Travel

EMPLOYEES

Title	Name	Tel.	Email	Detail
Project Manager	Fred Landis	123-456-78	f.landis@nanonull.com	☑

Fred Landis

Domestic daily rate	add DomesticDailyRate	Foreign daily rate	add Fo
Domestic cash advance	add CashAdvance	Foreign cash advance	add Ca

Expense items

General info		Travel		Accommodation	Entertainmen	
		Cost	337.88			
Date	2003-01-02	Destination	add Destination		Cost	add Ente
Bill to	Development	Car-Rental	add Car-Rental	add Accommodation	Client	add Ente
Curr.	USD	Air-Travel	add Air-Travel		Meal	add Ente
Total	**NaN**	Misc-Travel	add Misc-Travel		Gift	add Ente
		Cost	1014.22			
Date	2003-07-07	Destination	add Destination		Cost	add Ente
Bill to	Accounting	Car-Rental	add Car-Rental	add Accommodation	Client	add Ente
Curr.	USD	Air-Travel	add Air-Travel		Meal	add Ente
Total	**NaN**	Misc-Travel	add Misc-Travel		Gift	add Ente
		Cost	2000			
Date	2003-02-02	Destination	add Destination		Cost	add Ente
Bill to	Marketing	Car-Rental	add Car-Rental	add Accommodation	Client	add Ente

The expense report can now be completed with extra information relating to Accommodation, Entertainment and Misc. costs if necessary.

Please note:

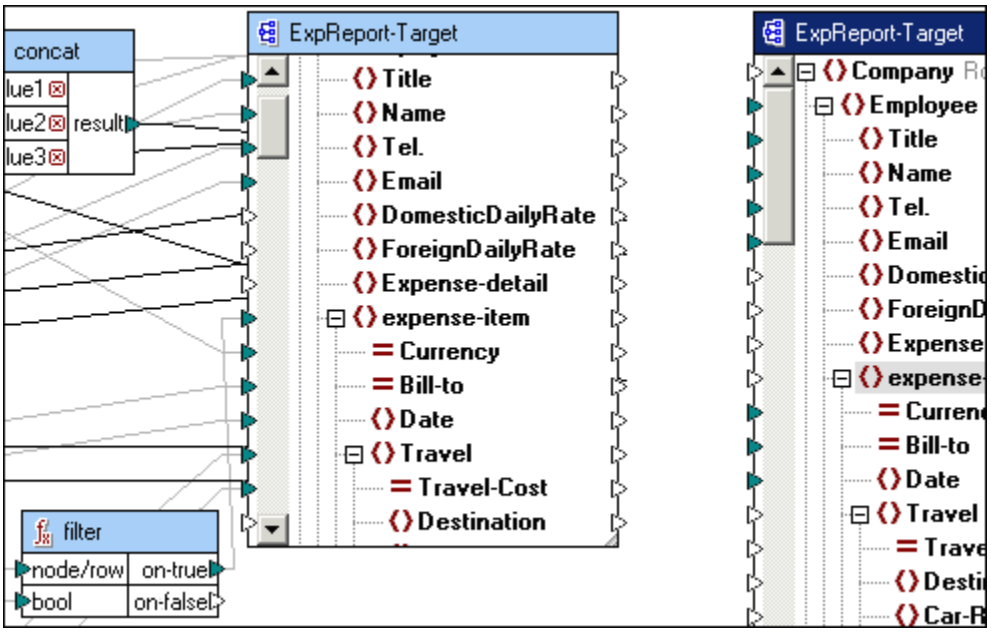
The Total field automatically sums up all Cost fields of each record. Once a number exists in all these fields, the Total field becomes live and the NaN (Not a Number) entry disappears. Subsequent changing of any of the Cost fields, automatically adjusts the Total field.

4.6 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**.

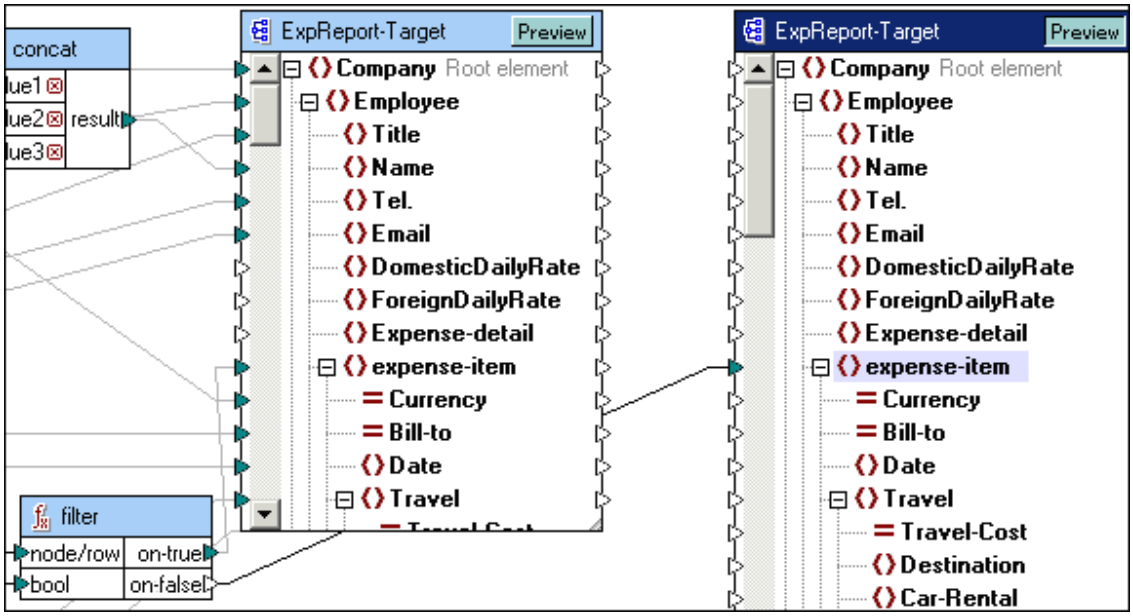
Creating the second target schema component:

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 tab.
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 tab.



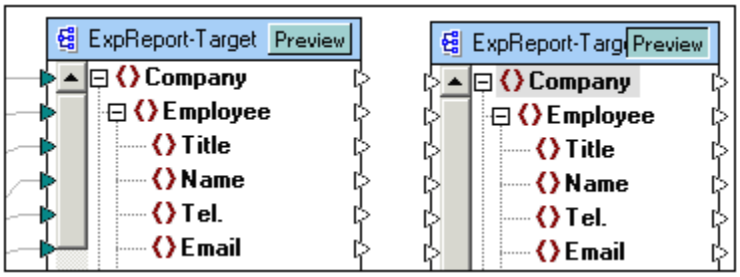
Filtering out the non-travel data:

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.



An **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 tabs.

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.

1. Right click the **second** target schema/document, and select the **Properties** option.
2. Enter a file name in the **Output XML instance field**, **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial\\SecondXML.xml** for example.

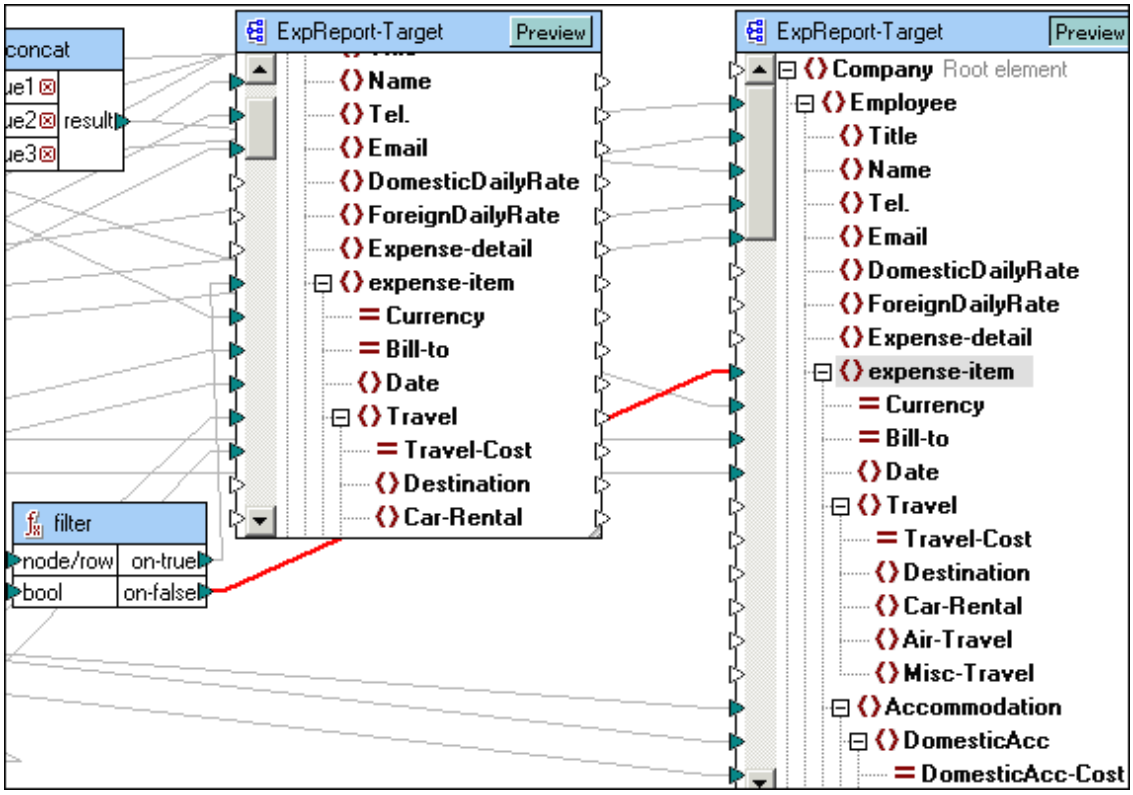
   It is recommended to insert the **absolute path** when generating code. The example above, uses the default installation path of MapForce.

Creating mappings for the rest of the expense report data:

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. 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:

Source schema - connect:	to... second Target schema
Person	Employee
Result of **existing** First and Last concatenation	Name
Title	Title
Phone	Tel.
Email	Email
currency	Currency
expto	Bill-to
Date	Date

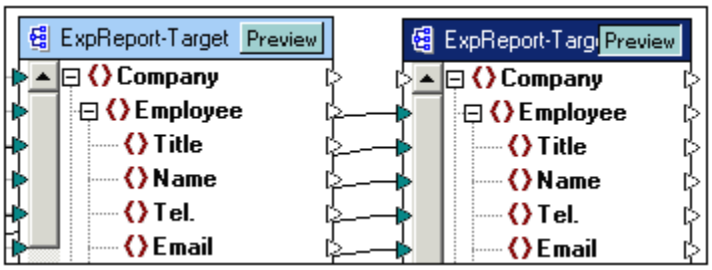


4.6.1 Viewing and generating multiple target schema output

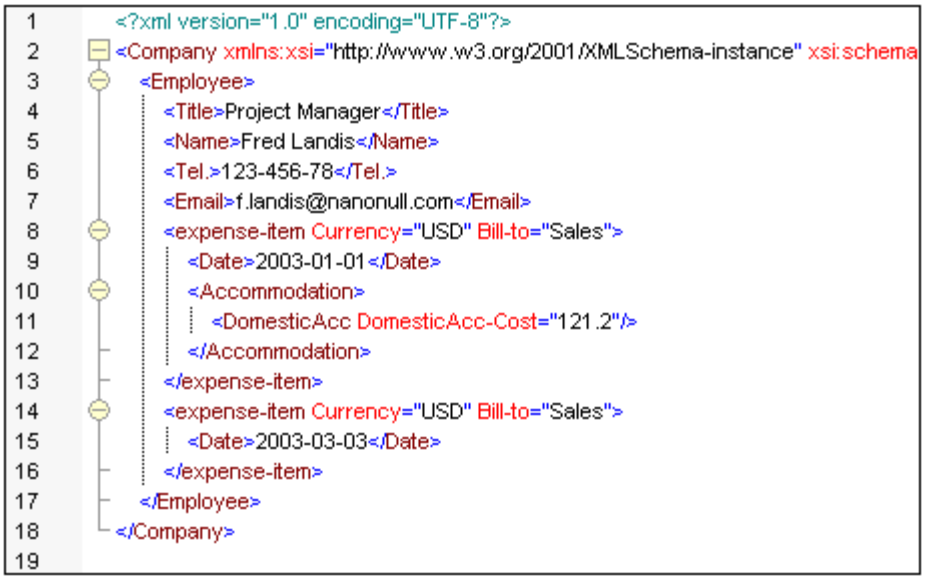
Clicking the Preview icon lets you select which of the schema targets you want to preview.

To view specific XSLT output:

1. Click the **Preview icon** in the title bar of the **second** schema component, to make it active.



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



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**

4. Having installed either XMLSpy, or Authentic Desktop, assign either of these two XSLT files to the **mf-ExpReport.xml** file, and start the transformation process.
5. Assign the supplied stylesheet, **ExpReport-Target.sps** to the file, and click the Authentic tab.

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.

4.7 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.

Aim of this section:
To merge two **personal travel expense reports** into a company expense travel report.

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.

Files used in this section:

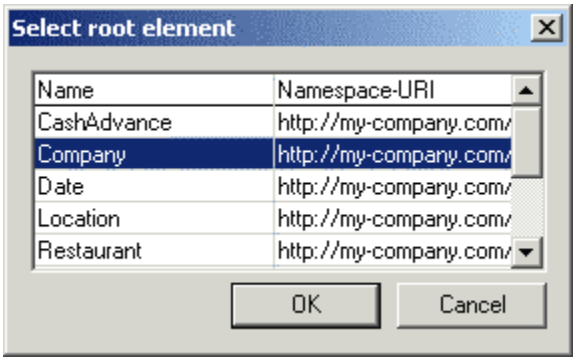
mf-ExpReport.xml	Input XML file used in previous section
mf-ExpReport2.xml	The second input XML file
mf-ExpReport-combined.xml	The resulting file when the mapping has been successful
ExpReport-combined.xsd	The target schema file into which the two XML source data will be merged.
ExpReport-combined.sps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

Please note:
The files used in this section are also available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.

4.7.1 Creating the mappings

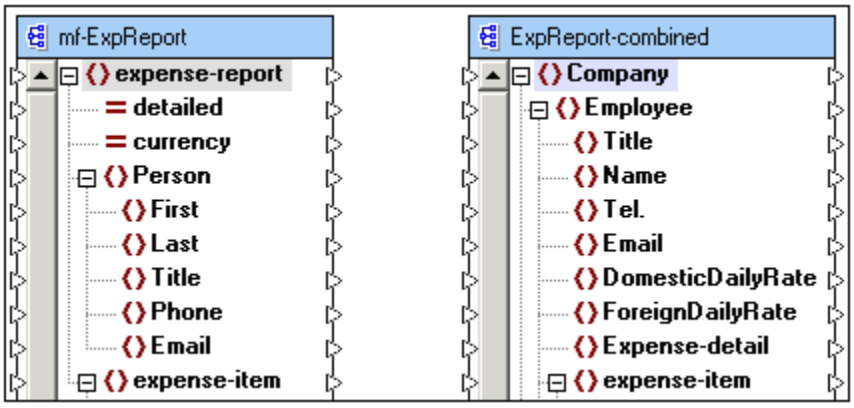
The method described below, is a recapitulation of how to set up the mapping environment.

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-ExpReport.xml** file as the XML instance file.
3. Click the **expense-report** entry, hit the * key on the numeric keypad to view all the items; resize the component if necessary.
4. Click the **Insert XML Schema/File** icon.
5. Select the **ExpReport-combined.xsd** file from the Open dialog box.
   You are now prompted for a sample XML file for this schema.
6. Click Skip, and select **Company** as the root element of the target document.



The target schema component now appears in the mapping tab.

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



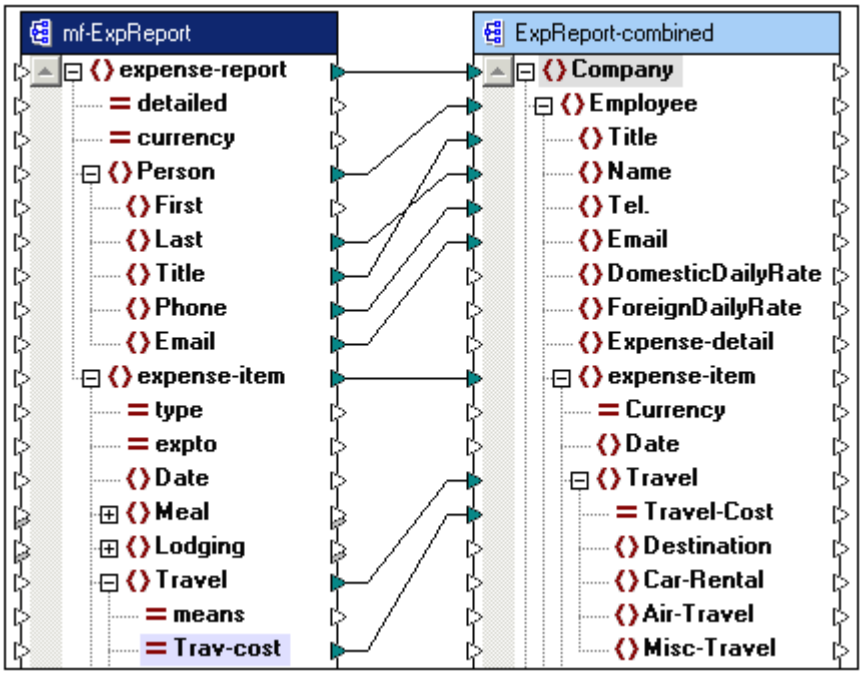
Make sure that the "Auto connect child items" icon  is deactivated, before you create the following mappings.

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 and

- Trav-cost to Travel-Cost.

  The mapping is shown below.



8. Click the Output tab to see the result of the current mapping.

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

Please note:

Empty <expense-item/> 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**.

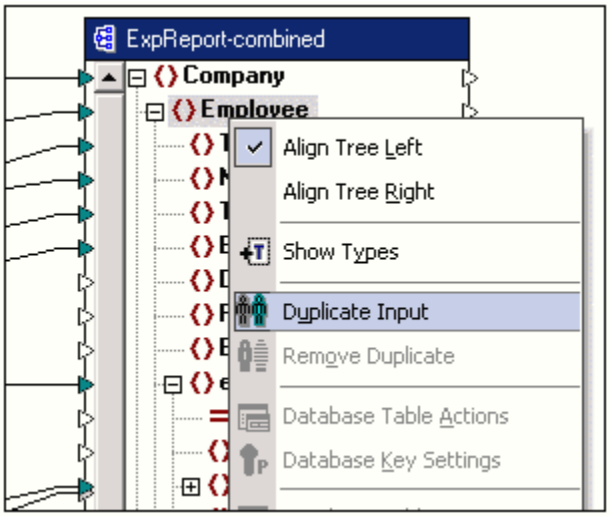
4.7.2 Duplicating input items

We now 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:

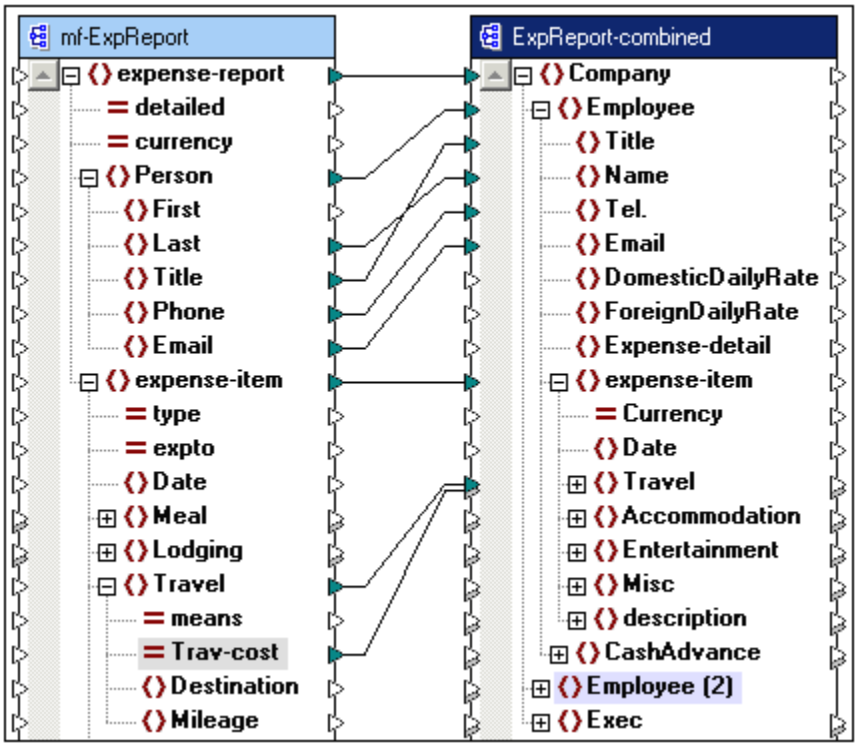
- add the **second** XML source file, and
- create mappings from it, to the "same" inputs in the target XML file.

Duplicating input items:

1. Right click the Employee entry in the target XML file.
2. Select the menu option **Duplicate input.**

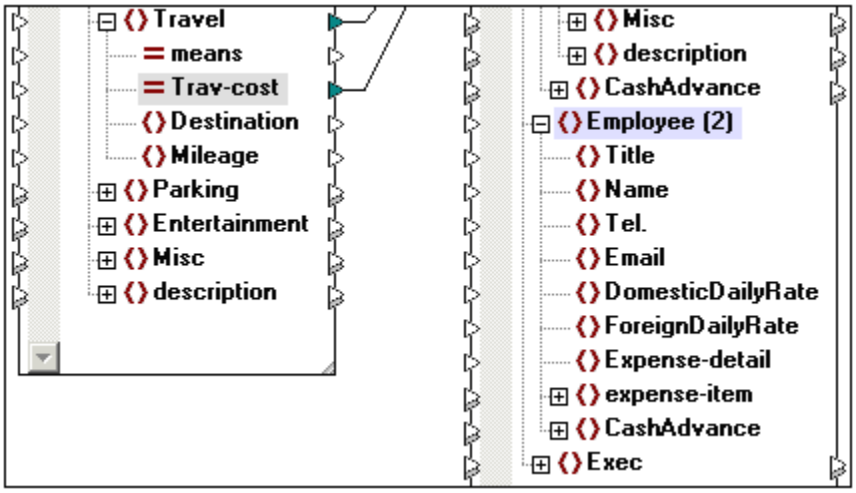


A second Employee item has now been added to the component, as **Employee(2)**.



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.

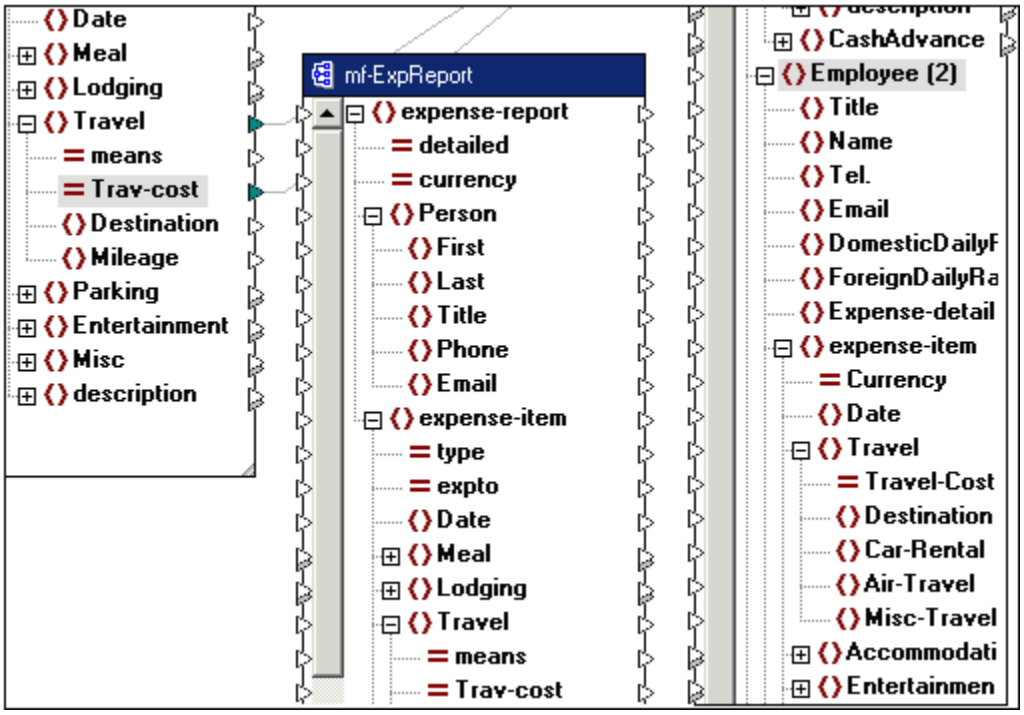


You can now use these new duplicate items as the **target** for the second source XML data file.

Use the same method as before, to insert the second XML instance file:

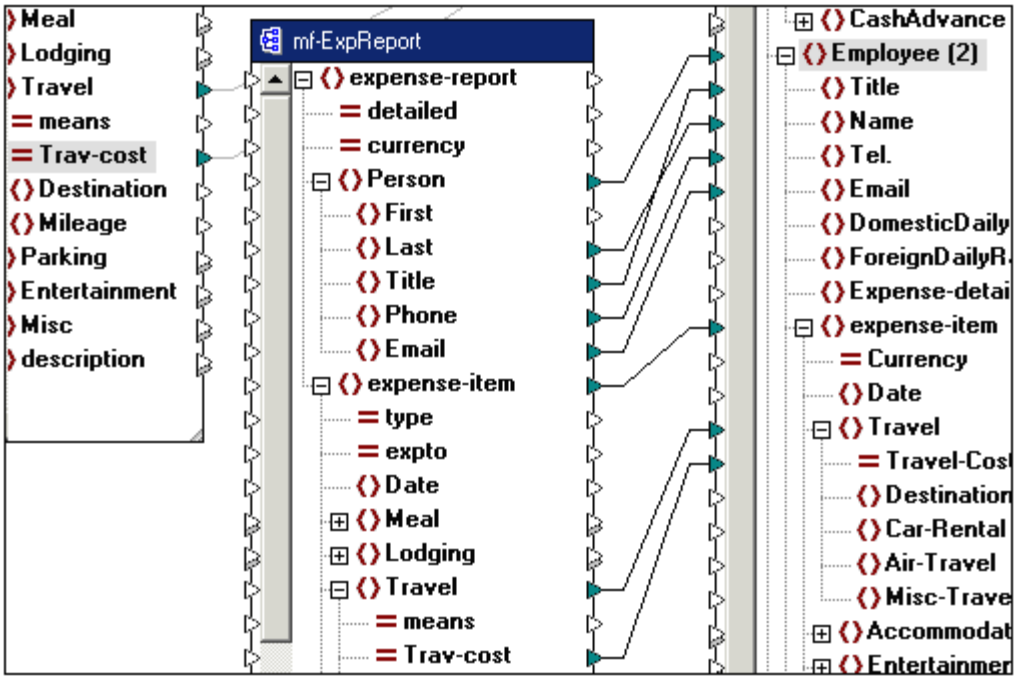
1. Click the **Insert Schema | XML instance** icon.
2. Select the **mf-ExpReport.xsd** file from the Open dialog box, click Browse..., and select the **mf-ExpReport2.xml** file as the XML instance file.
3. Click the **expense-report** entry, hit the * key on the numeric keypad to view all items, and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

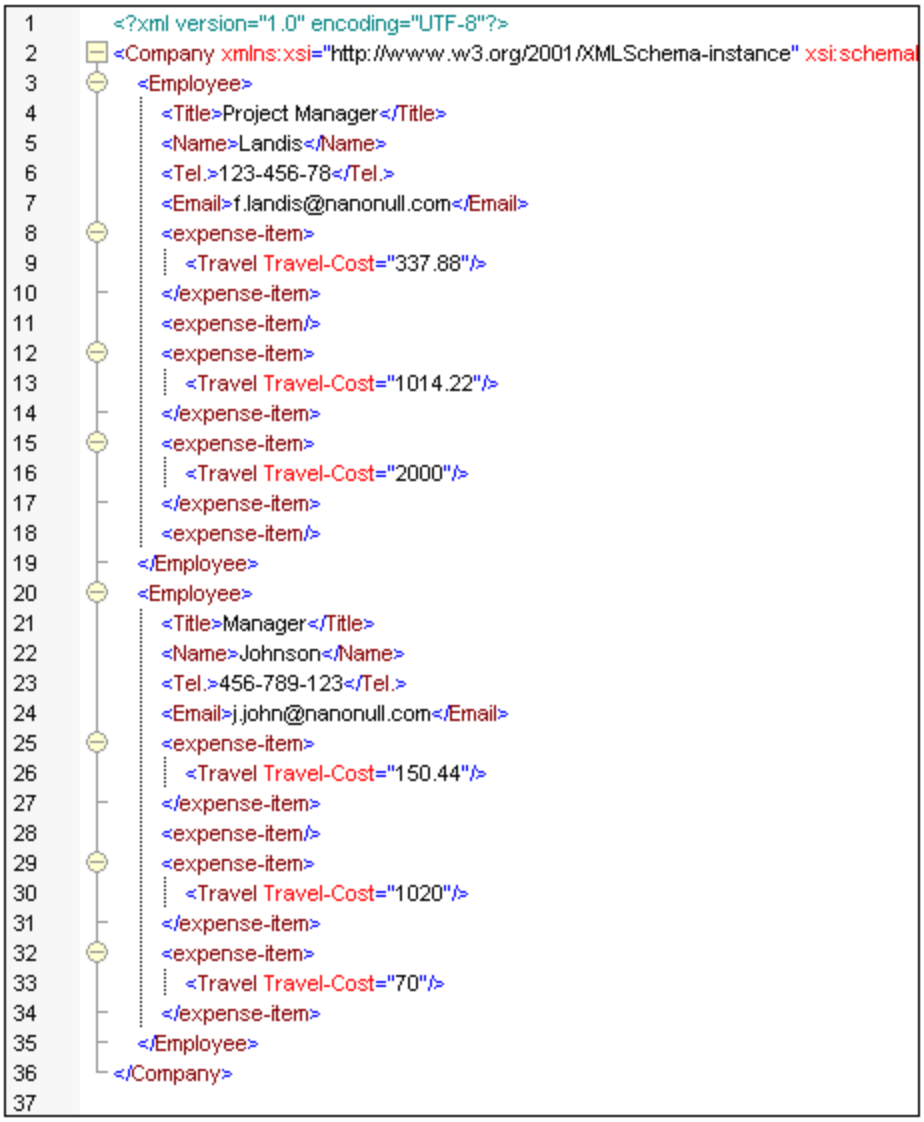


4. Create the same mappings that were defined for the first XML source file:

- 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 tab to see the result of the mapping.



```
<?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 tab is active.

The file, mf-ExpReport-combined.xml, is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. Please note that it has been assigned an SPS file, which allows you to view the XML file in Authentic View of Authentic Desktop, or XMLSpy.

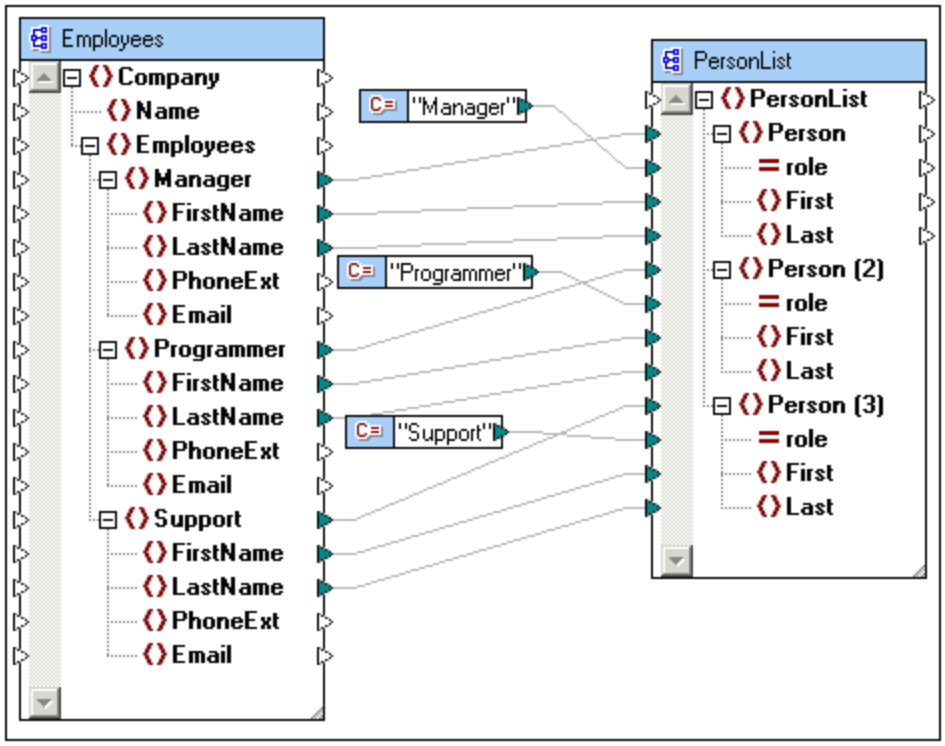
To remove duplicated items:

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

To see a further example involving duplicate items, please see the **PersonList.mfd** sample file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder.

In the example:

- Different elements of the source document are mapped to the "same" element in the target Schema/XML document.
- Specific elements (Manager etc.) are mapped to a generic one using a "role" attribute.



4.8 Database to schema mapping

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

Currently supported databases (and connection types) are:

- Microsoft Access versions 2000 / 2003 /2007
- Microsoft SQL Server versions 2000 and 2005
- Oracle version version 9i and 10g
- MySQL 4.x and 5.x
- Sybase 11, 12.0 and 12.5
- IBM DB2 version 8.x und 9

- Any ADO (compliant database)
- Any ODBC (compliant database)

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 (not recommended) 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**.

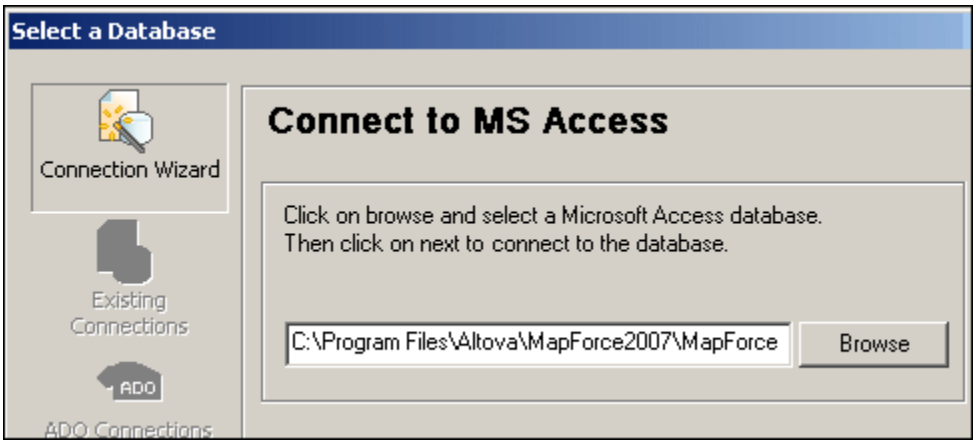
\- **Not available**

Creating the database component in MapForce:

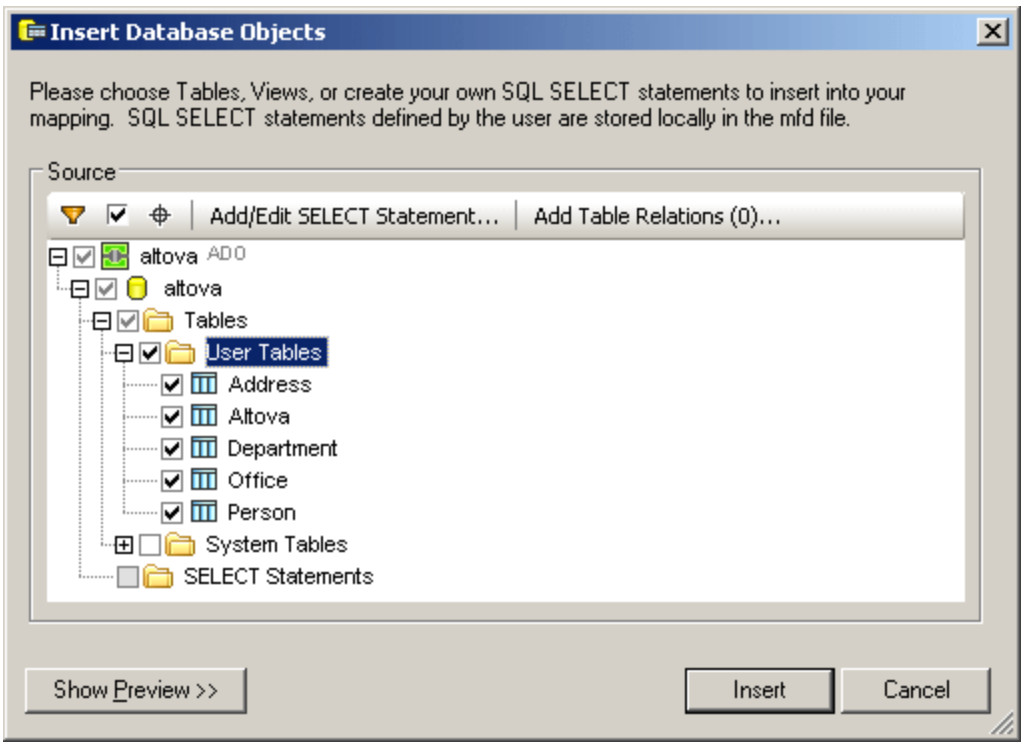
1. Select **File | New** in MapForce to create a new mapping.
2. Click one of the programming language icons in the toolbar: Java, C#, or C++.

3. Click the **Insert Database** icon  in the icon bar.

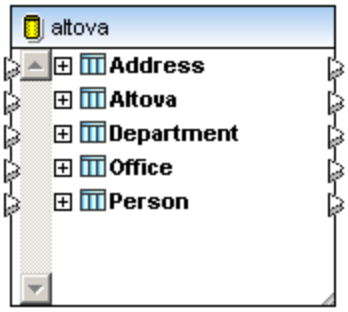
4. Click the **Microsoft Access** radio button.
5. Click the Next button to continue.
6. Click the Browse button to select the database you want as the data source, **altova.mdb** in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder in this case. The connection string appears in the text box.



7. Click the **Next** button.



8. 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.



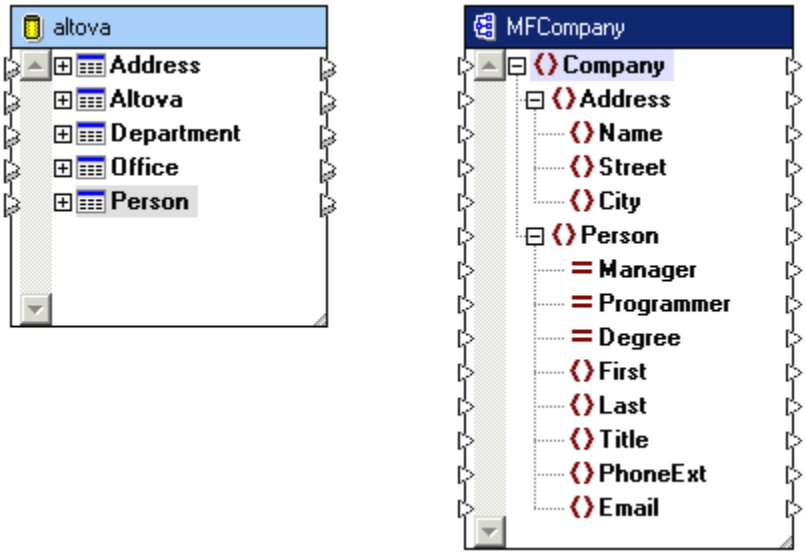
The database component appears in the mapping window. You can now create mappings to a target schema / XML document.

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

4.8.1 Mapping database data

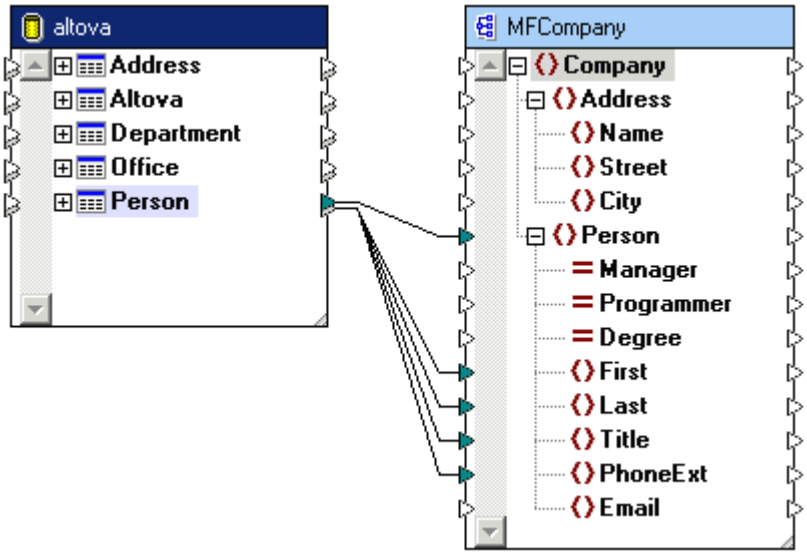
Inserting the target schema /document:

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:

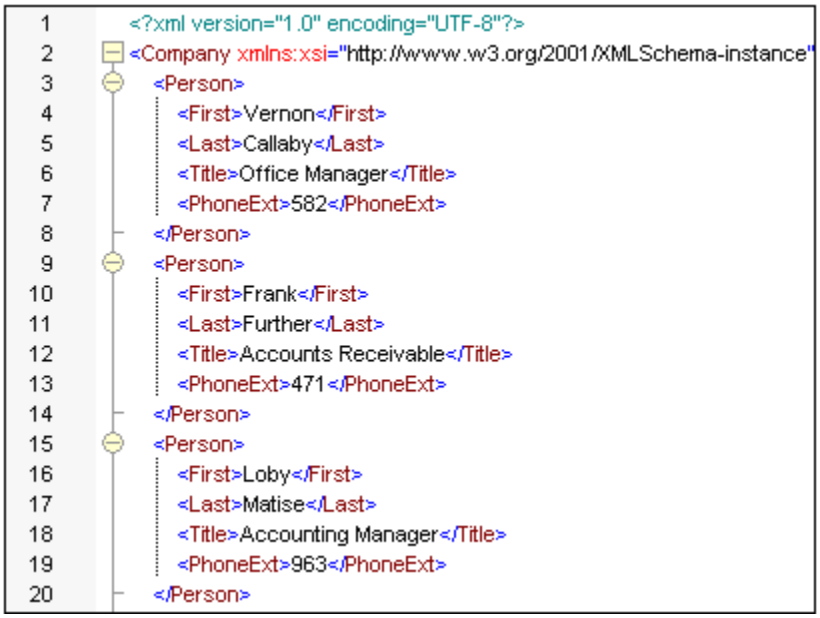
1. Activate the Auto connect child items icon  , if not already active.
2. Click the **Person** "table" item in the database component, and connect it to the Person item in MFCompany.
   This creates connectors for all items of the same name in both components.



4. Save the MapForce file, **PersonDB** for example.
5. Click the Output tab to see the result/preview of this mapping. The MapForce 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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder of MapForce.

Chapter 5

Methods of mapping data (Standard / Mixed / Copy All)

5 Methods of mapping data (Standard / Mixed / Copy All)

MapForce supports various methods of mapping data:

Target Driven (Standard)
Standard mapping means the normal method of mapping used in MapForce, i.e. the output depends on the sequence of the target nodes. Please see Source-driven / mixed content vs. standard mapping for more information.

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

Source driven / mixed content mapping
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.

Copy All mapping
This type of connection allow you to organize 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.

5.1 Source driven / mixed content mapping

MapForce supports 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.

Source-driven 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.

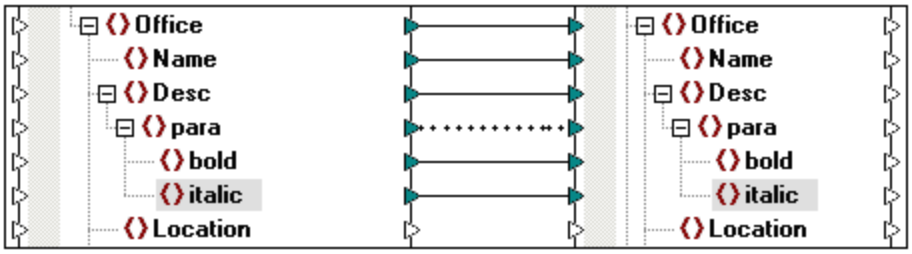
Source driven / mixed content mapping supports:

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

Please note:

Mixed content **text nodes** can only be mapped in their entirety; you cannot limit, or transform the data they contain. Filters, or any other type of function, cannot be used to access text node data.

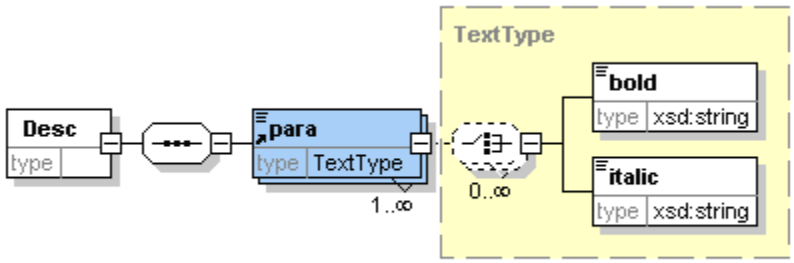
The image below shows an example of mixed content mapping. The para element is of mixed content, and the connector is shown as a **dotted** line to highlight this.



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

The files used in the following example (**Tut-Orgchart.mfd**) are available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.

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.



Content model of **para** element:

- para is a complexType with **mixed** = true, of type TextType.
- bold and italic elements are both of type **xsd: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.

Source XML instance:
A portion of the 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, see "mixed content settings".

```
<?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>
```

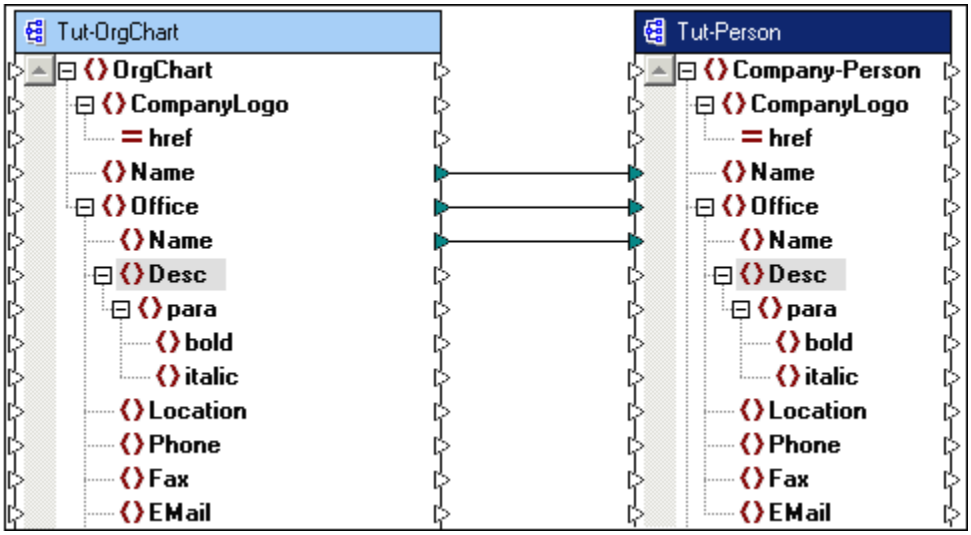
Please note the **sequence** of the text and bold/italic nodes of Nanonull., Inc in the XML instance file, they are:

<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>

Mapping
The initial state of the mapping 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
```

5.2 Default settings: mapping mixed content

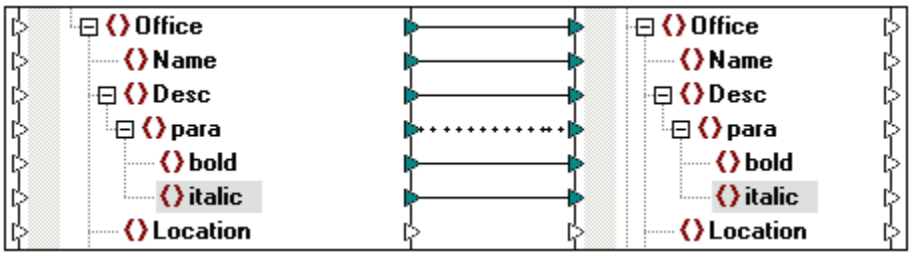
Creating mixed content connections between items:

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

   A message appears, asking if you would like to create a mixed content connection. You are also notified that the text and child items will be transferred in the same order they appear in the XML source file.
3. Click Yes to create a mixed content connector.

Please note:
Although the Desc element is not of mixed content, a message appears because the auto-connect option has been activated, and para exists in both source and target components. 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 Desc have been connected. The connector joining the para items is displayed as a dotted line, to show that it is mixed content.

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

```
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      <Desc>
7        <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull devel
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 cc
17     </Desc>
18   </Office>
19 </Company-Person>
20
```

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

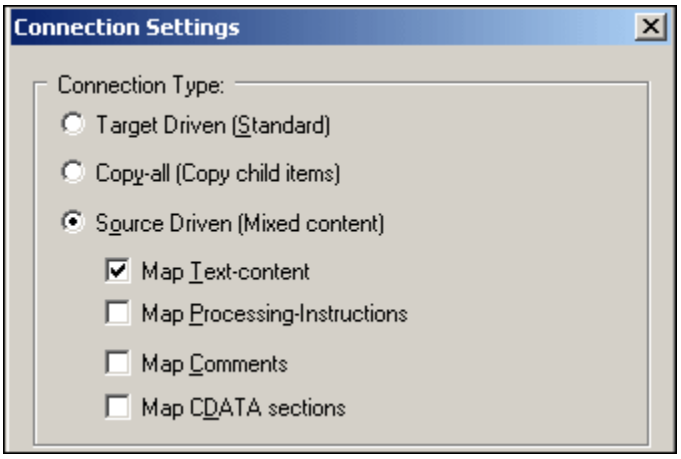

```
3     <Name>Organization Chart</Name>
4     <Office>
5       <Name>Nanonull, Inc.</Name>
6       <Desc>
7         <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull develops
      nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999 saw the
      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.

Removing text nodes from mixed content items:

1. Right click the para connector and select **Connection Settings**.



The image shows the default settings when you first create mixed content mapping. The "Map Text content" check box is active per default.

2. **Deactivate** the Map Text content check box and click OK to confirm.
3. 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
- bold and italic item **sequence** still follow that of the source XML file!

Text nodes and mixed content mapping:

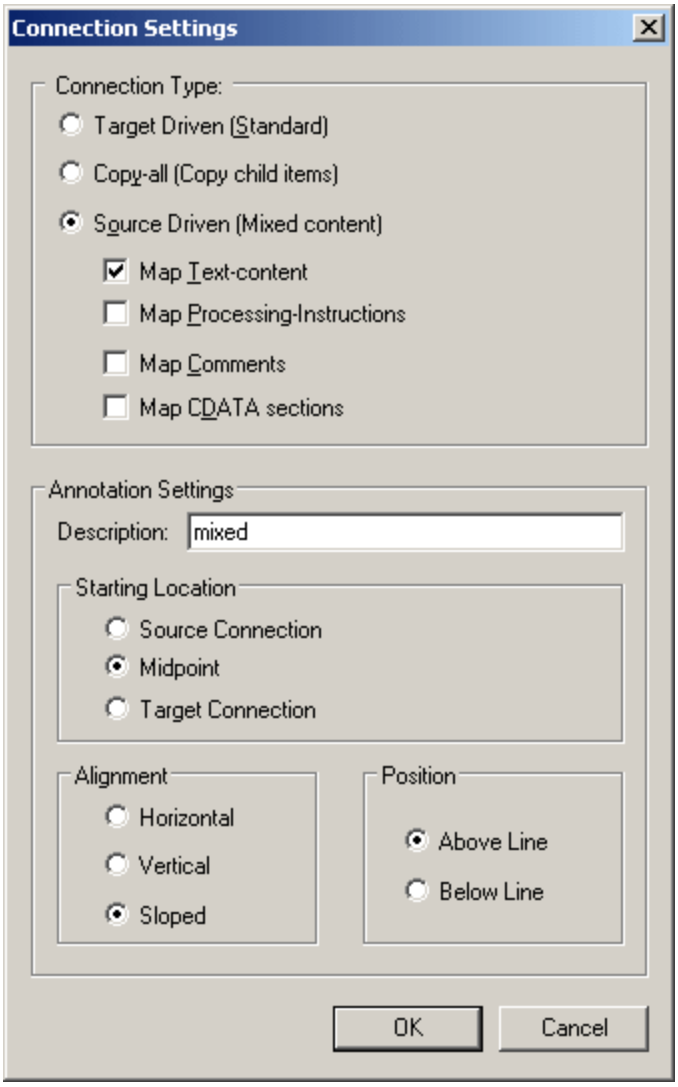
- Text nodes can only be mapped in their entirety; you cannot limit, or transform the data they contain. All text nodes of the para element are either mapped, or excluded, as in the example above.
- Filters, or any other type of function, cannot be used to access text node data.
- Mixed content child node data, i.e. data enclosed in bold/italic tags in this example, can of course be mapped individually. If a connector exists, then the child data will be mapped.
- There is currently no way of accessing the text node(s) of a mixed content element, for further processing, or filtering.

Mixed content settings:

- Right click the para connector and select Connection Settings.

  This 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.

  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: "Source-driven / mixed content vs. standard mapping" 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.

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.

5.3 Mixed content example

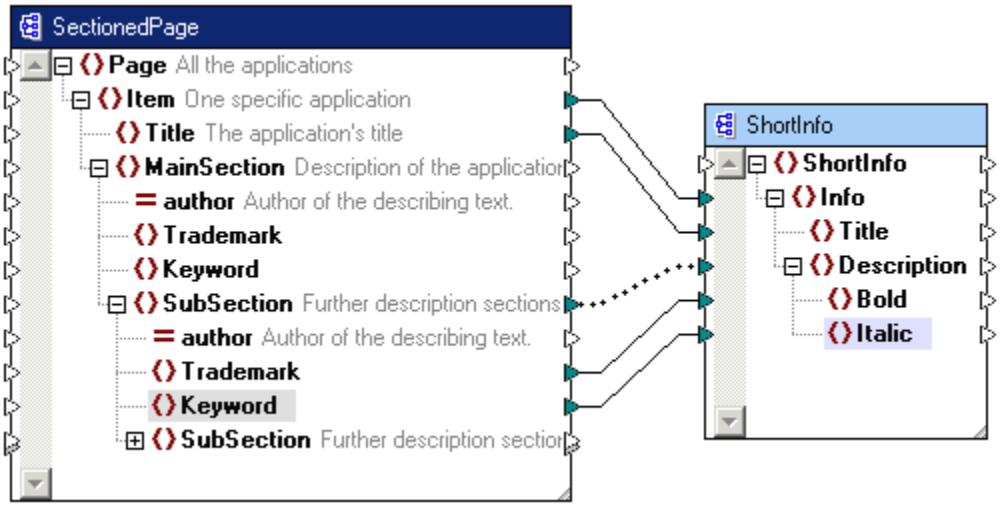
The following example is available as "**ShortApplicationInfo.mfd**" in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder.

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.
- Trademark text is mapped to the Bold item in the target
- Keyword text is mapped to the Italic item in the target



Mapping result:

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

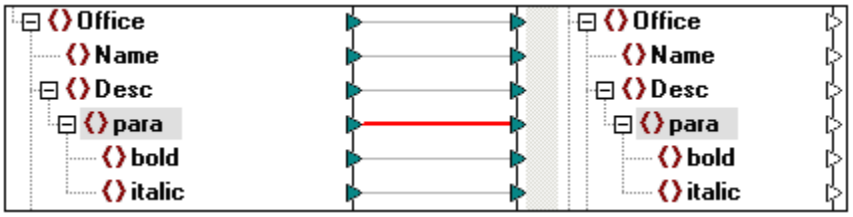
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ShortInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="
   C:/PROGRA~1/Altova/MapForce2005/MapForceExamples/ShortInfo.xsd">
3    <Info>
4      <Title>XMLSpy</Title>
5      <Description>Altova <Bold>XMLSpy</Bold>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

5.4 Source-driven / mixed content vs. standard mapping

This section describes the results when defining standard mappings (or using standard connectors) on mixed content items. The files used in the following example (**Tut-Orgchart.mfd**) are available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.

Creating standard connections between mixed content items:

1. Create a connector element between the two para items.
   A message appears, asking if you would like to create a mixed content connection. You are also notified that the text and child items will be transferred in the same order they appear in the XML source file.
2. Click No to create a standard mapping.



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

```
<Office>
  <Name>Nanonull, Inc.</Name>
  <Desc>
    <para>The company was established in Verenoin 1995. Nanonull develops
nanoelectronic technologies formulti-core processors.February 1999 saw the unveiling of the
first prototype Nano-grid.The company hopes to expand its operations offshoreto drive
down operational costs.
      <bold> Vereno</bold>
      <bold>Nano-grid.</bold>
      <italic>multi-core processors.</italic>
      <italic>offshore</italic>
    </para>
    <para>White papers and further information will be made available in the near future.
    </para>
  </Desc>
</Office>
<Office>
  <Name>Nanonull Europe, AG</Name>
```

Result:

- all **text** nodes of the para element have been retained
- mapped bold and italic text content remain
- However, bold and italic item **sequence** follow that of the **target** XML/schema file!

Target Driven (Standard) - properties
Standard mapping means the normal method of mapping used in MapForce, i.e.:

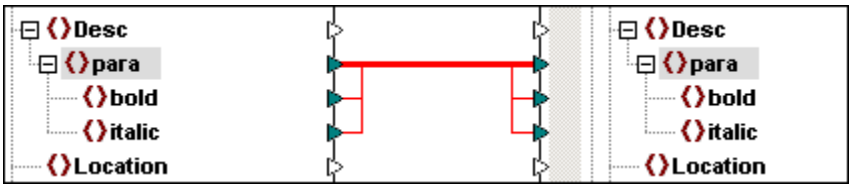
- Mixed content text node **content** is supported/mapped, but the start/end tags of the child nodes are stripped out.
- The sequence of child nodes is dependent on the target XML/schema file.
- The child nodes appear after the mixed content node text.

  In this example:
  For each **para** element, first **map all bold** items, then map **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.

Copy-all mapping:

1. Right click the para connector and select **Copy-all** from the popup window.
   The connector now appears as a solid line with the child items branching out of, and below it. Please see "Copy-all connections" for more information.



5.5 Copy-all connections

This type of connection allow you to organize your workspace and automatically connect **all** identical items in source and target components, meaning that, depending on the source and target **type**:

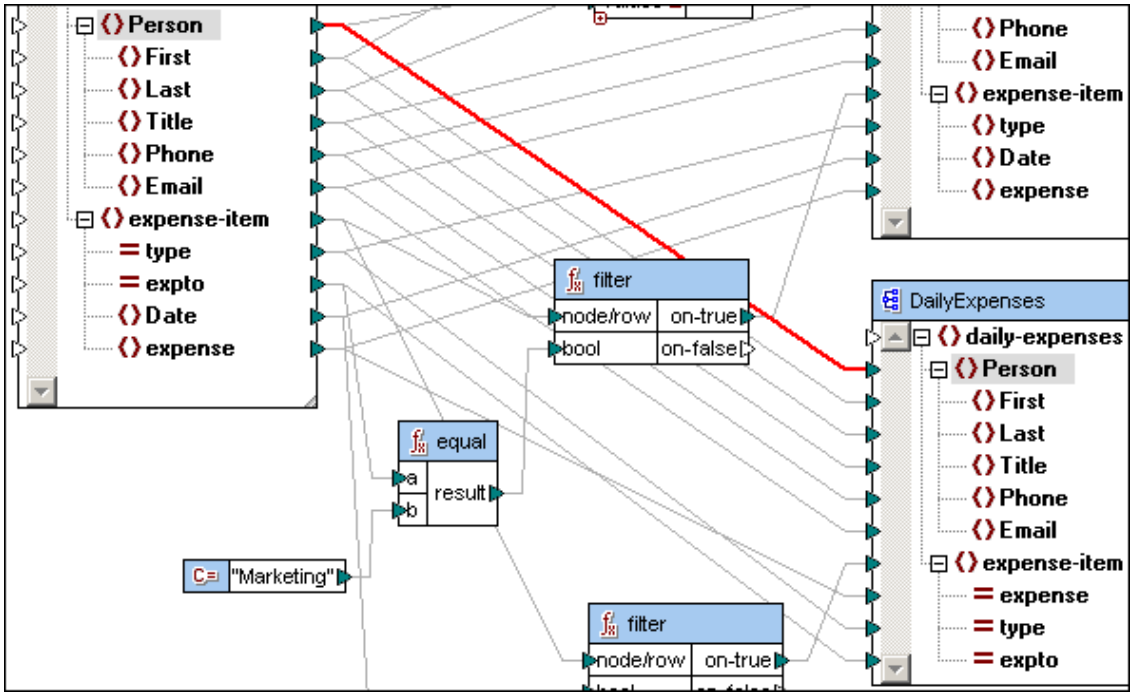
- all source child items are **copied** to the target component, if either the source and target **types** are **identical**, or if the target type is xs:anyType
- if the source and target **types** are **not identical,** and if the target type is not xs:anyType, the source data is transferred/mapped 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.
- Note that only the names of the child items, but not their individual types, are compared/matched.

Currently Copy-all connections are supported:

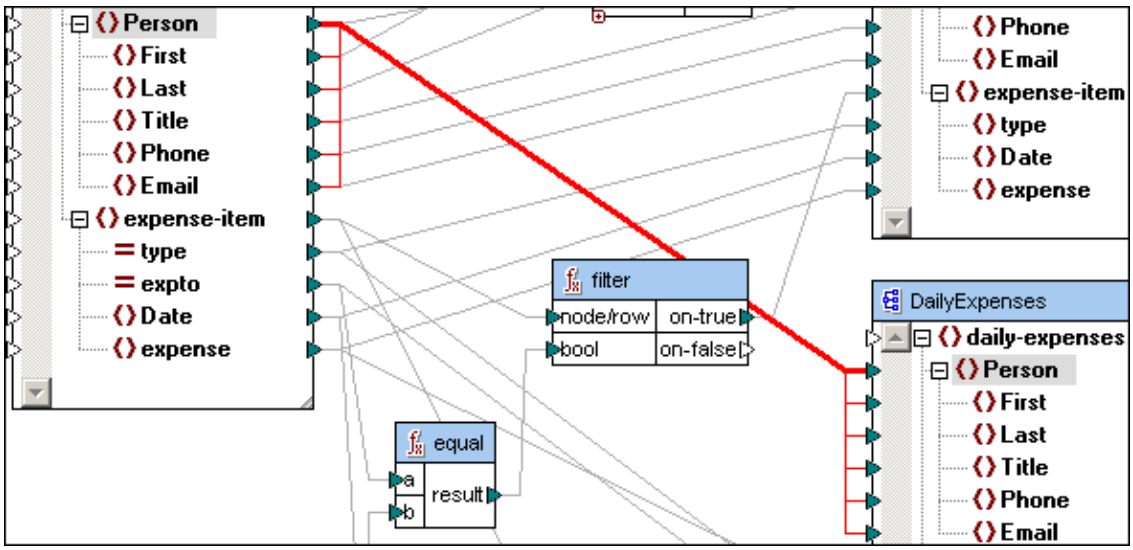
- between XML schema complex types, and
- between complex components (XML schema, database, EDI) and complex user-defined functions/components containing the same corresponding complex parameters, please see "Complex output components - defining" for an example.

The example below shows these connectors using the **MarketingAndDailyexpenses.mfd** file in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder.

1. Right click the Person component and select "Copy-all" from the context menu. A prompt appears reminding you that the target connectors will be deleted.



2. Click OK if you want 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.

Copy-all connections and user-defined functions

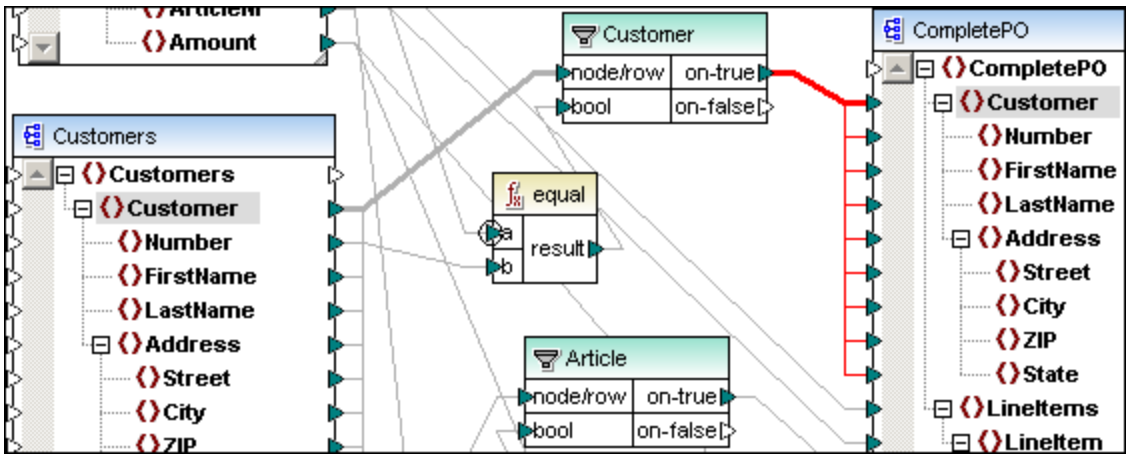
When creating Copy-all connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components - defining" for an example.

Copy-all connections and filters

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

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

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



Copy-all connections through filters cannot be created if the source component:

- is an "exists" type of component
- is a "function" component e.g. "equal"

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 connector on the output side of the filter, the left hand connector cannot be used to change the connection type.

5.6 Connector properties

Connectors and their properties:

- Clicking a connector highlights it in red.
- Hitting the **Del** key, while 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.

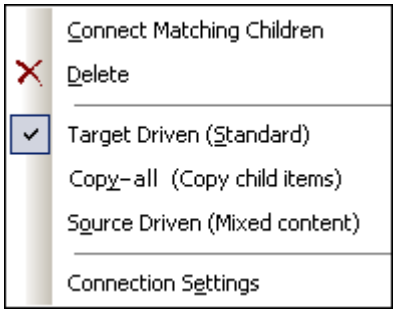
Show selected component connectors 
Switches between showing:

- 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.

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
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.

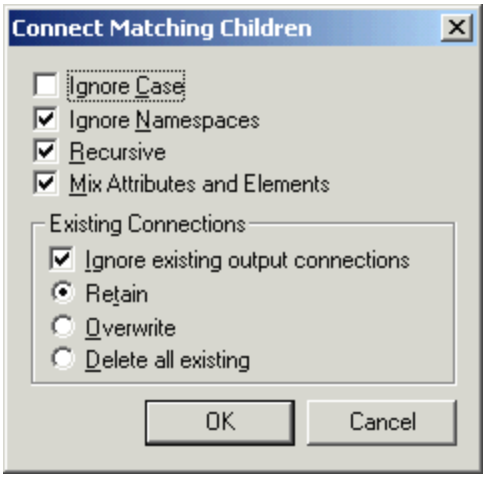
Source Driven (mixed content)

Changes the connector type to source-driven / mixed content, please see: "Source driven and mixed content mapping" for more information.

Connection settings:
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.

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 deactive 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.

Existing connections:

Ignore existing output connections:
Creates **additional** connectors to other components, even if the currently existing output icons already have connectors.

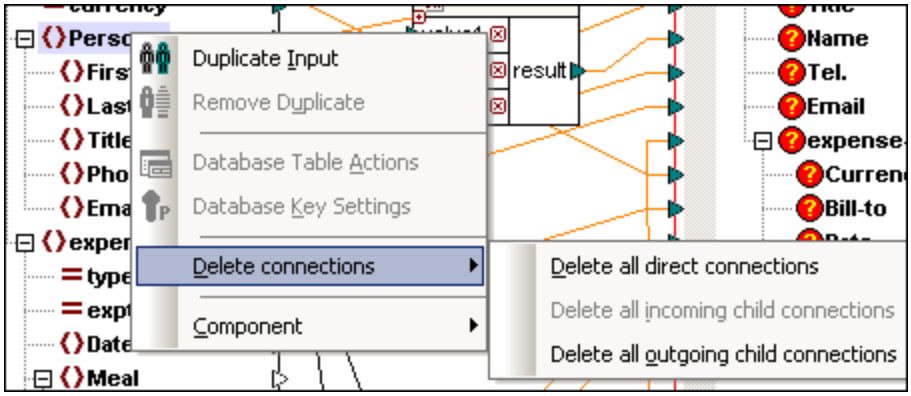
Retain
Retains existing connectors.

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.

Chapter 6

Global Resources

6 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 for example.
- 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 C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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).

6.1 Global Resources - Files

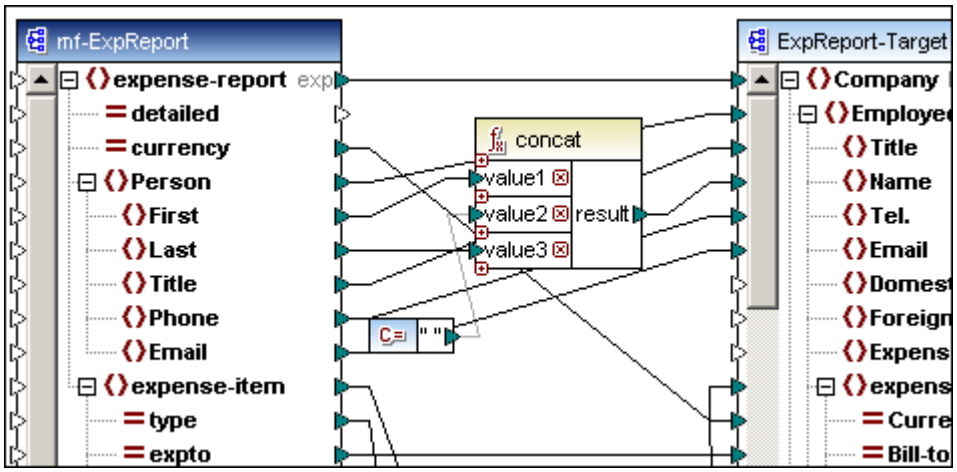
Global Resources in MapForce:

- Any input/output components **files** can be defined as global resources, e.g. XML, XML Schema, Text/CSV, database, EDI, and Excel 2007 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.

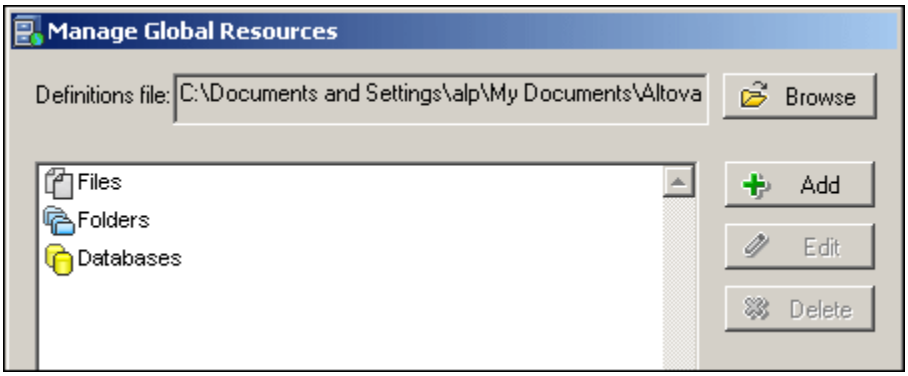
  This section uses the **Tut-ExpReport.mfd** file available in the C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples \Tutorial\ folder.



6.1.1 Defining / Adding global resources

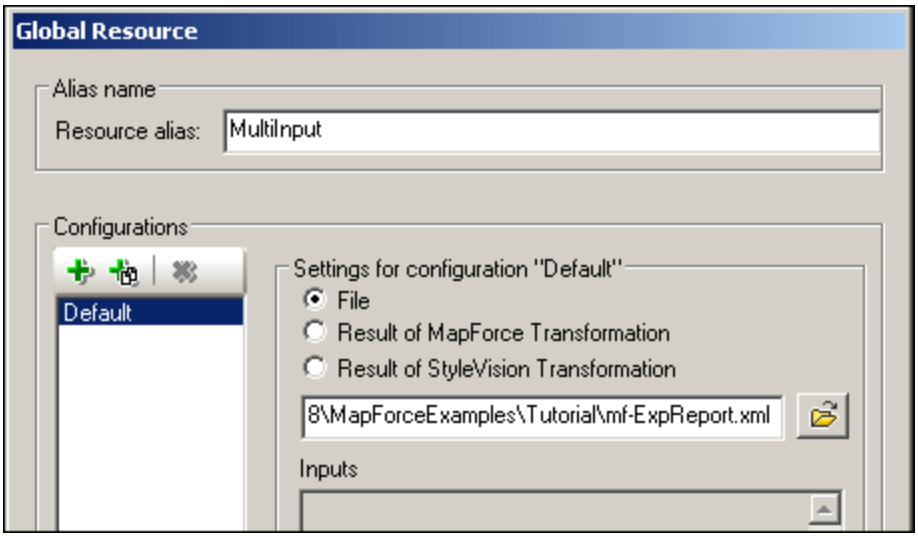
Defining / Adding a global resource file

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

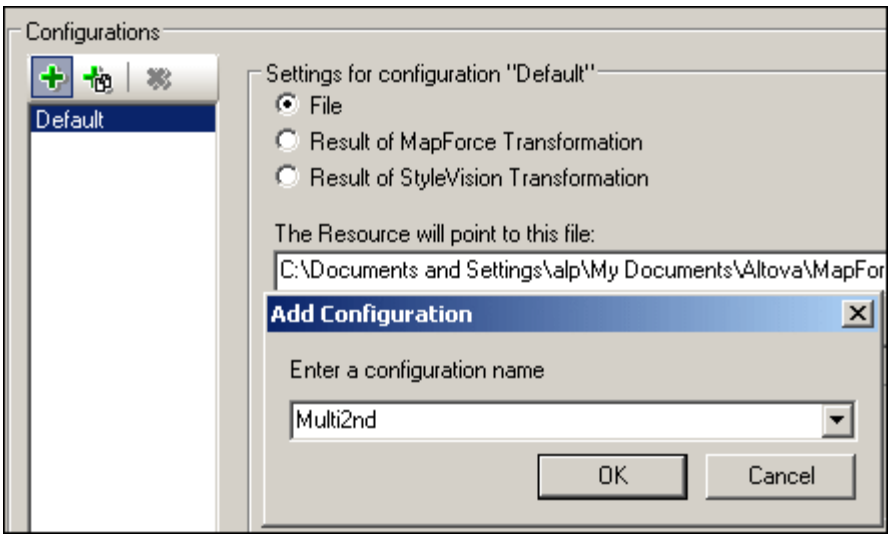


This is the state of an empty global resources file.

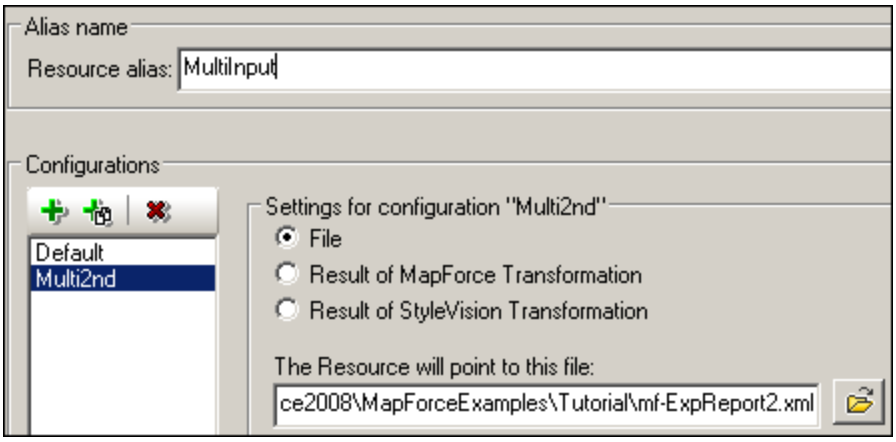
2. Click the **Add** button and select **File** from the popup.
3. Enter the name of the Resource alias e.g. **MultiInput**.
4. Click the Open folder icon and select the XML file that is to act as the "Default" input file e.g. **mf-ExpReport.xml**.



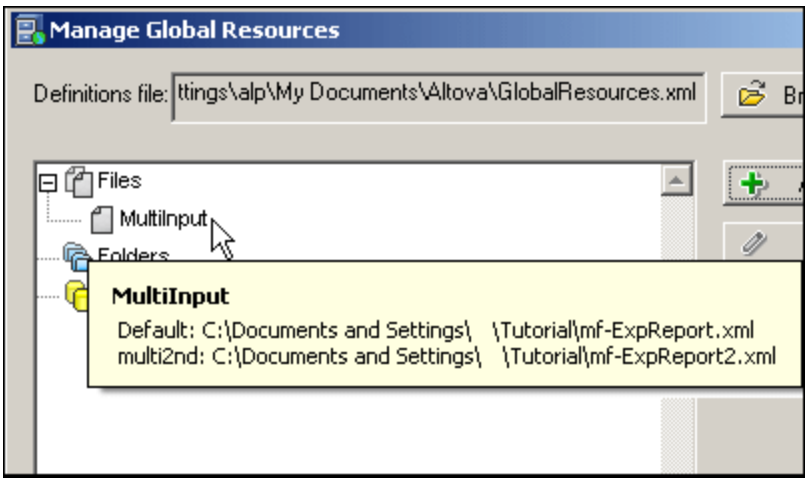
5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias. Note that the Copy configuration icon, allows you to copy a selected configuration and save it under a new name.



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.

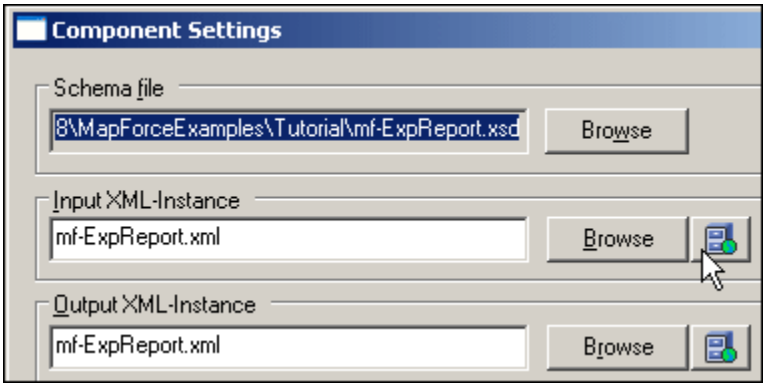


9. Click OK to confirm.
   This concludes the definition part of defining global resources. The next step is Assigning a global resource to a component.

6.1.2 Assigning a global resource

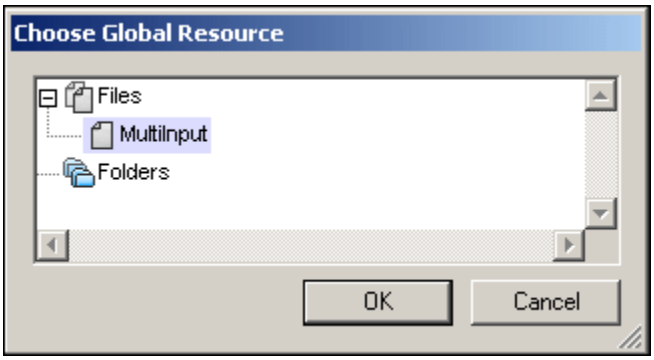
Assigning global resources to a component
We now have to assign the global resource to the component that is to make use of it.

1. Double click the **mf-ExpReport** component and click the global resource icon next to the Input XML-Instance field.

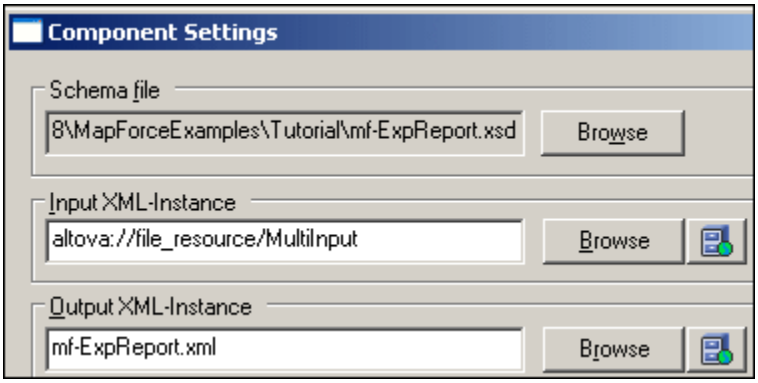


This opens the "Choose Global Resource" dialog box.

2. 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**.

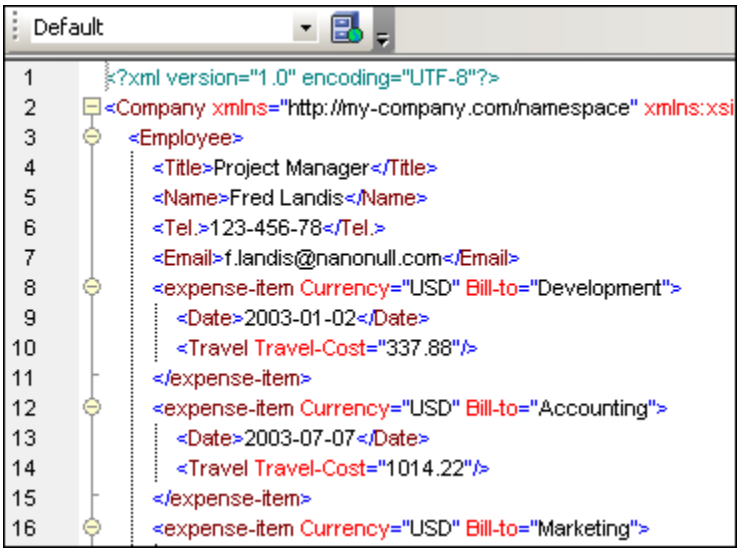


3. Click OK to complete the assignment of a resource to a component.
The next step is Using / activating a global resource.

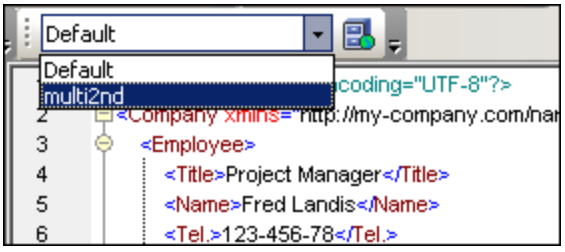
6.1.3 Using / activating a global resource

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.

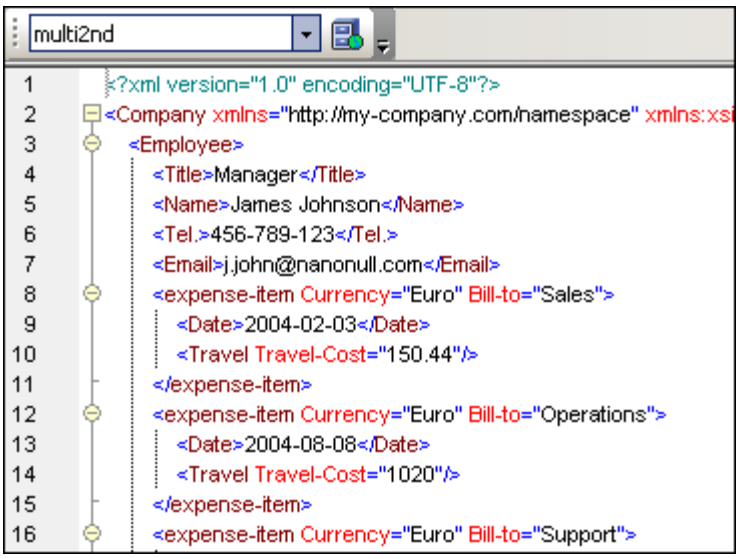


2. Click the global resources combo box and select **Multi2nd** from the combo box.



A message box opens stating that an **input** file has changed and prompts if you want to **regenerate** the output.

3. Click **Yes** to use the XML file defined by the multi2nd configuration.



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.

You can change the currently active resource while in the Output tab, there is no need to return to the mapping tab and click Output again.

6.2 Global Resources - Folders

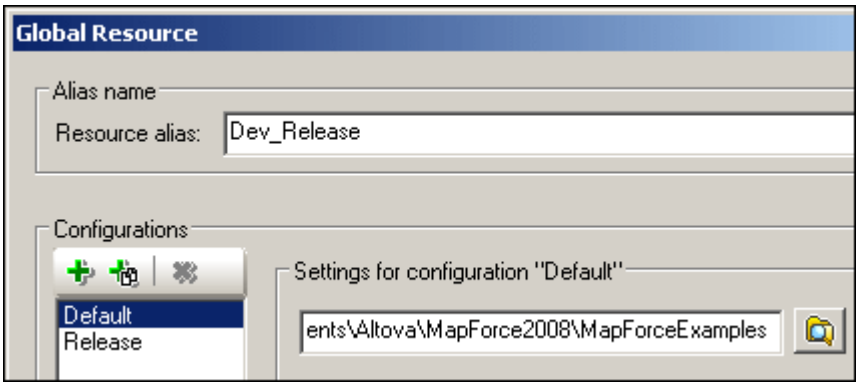
Folders can also be defined as a global resource, which means that input components can contain files that refer to different folders, for development and release cycles for example.

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.

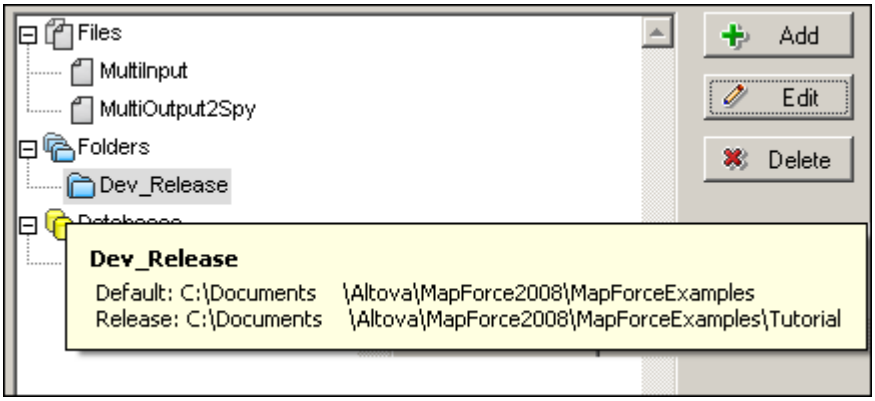
The mapping file used in this section is available as "**global-folder.mfd**" in the C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial\ folder.

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.



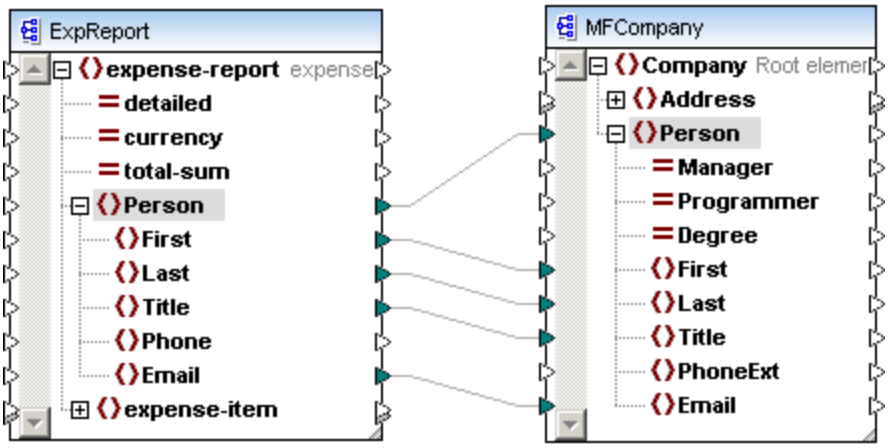
5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias, and enter a name for it e.g. Release. Note that the Copy configuration icon, allows you to copy a selected configuration and save it under a new name.
6. Click the Open folder icon and select the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



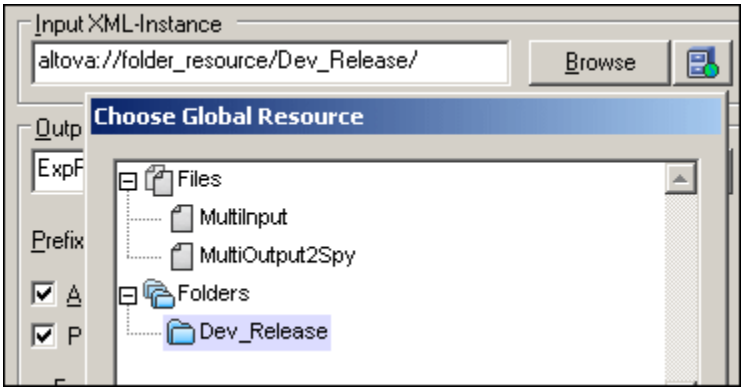
7. Click OK to finish the global folder definition.

Assigning the global resource folders:

1. Double click the **ExpReport** component and click the global resource icon next to the **Input XML-Instance** field.

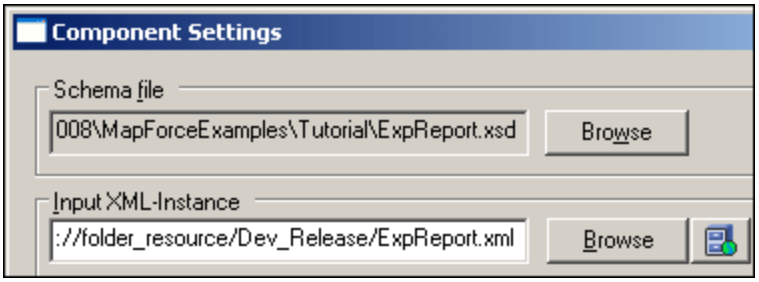


2. Click the resource you want to assign, **Dev_Release** in this case, and click OK.



The "Open..." dialog appears.

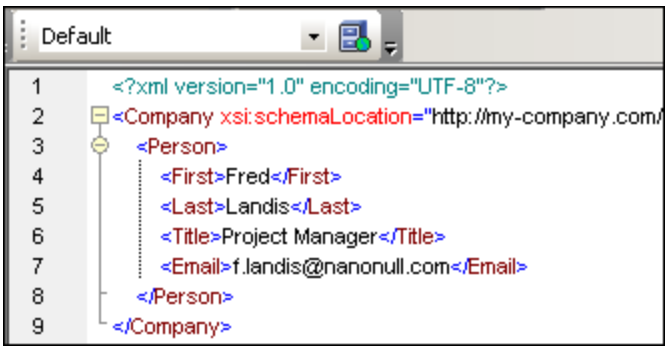
3. Select the **file** name that is to act as both the Development and Release **resource** file in each of the folders, e.g. **ExpReport.xml** and click OK to finish assigning the resource folder.



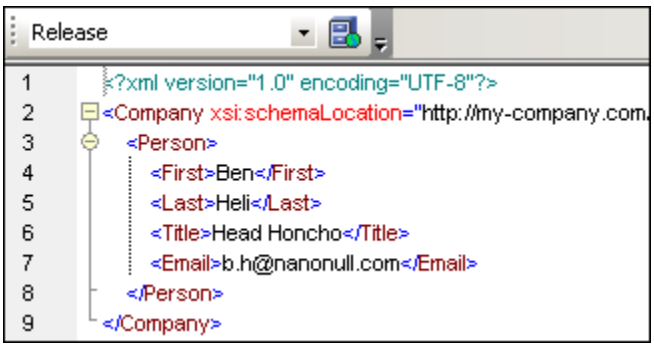
Note that this file is available in both directories but has different content.

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 Global Resource combo box and select the "**Release**" entry.
3. Click **Yes** when the prompt asks if you want to regenerate the output.

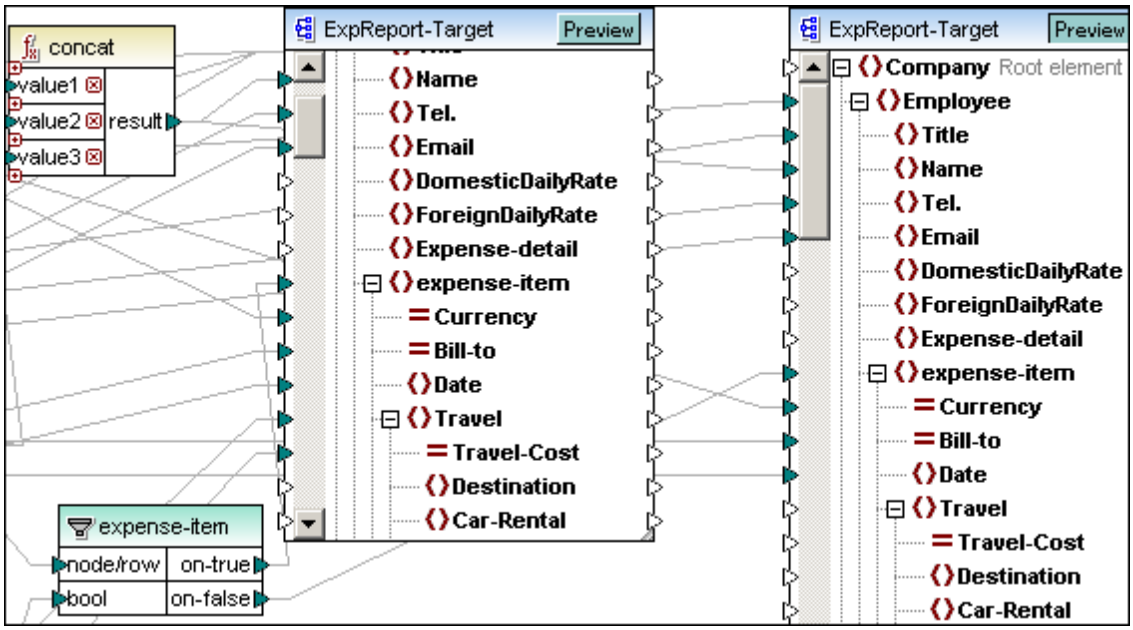


The output from the "Release" folder .../MapforceExamples/Tutorial is now displayed.

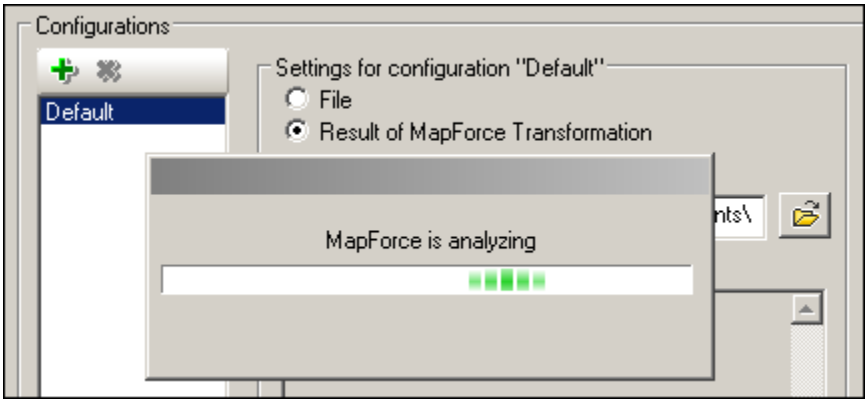
6.3 Global Resources - Application workflow

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.

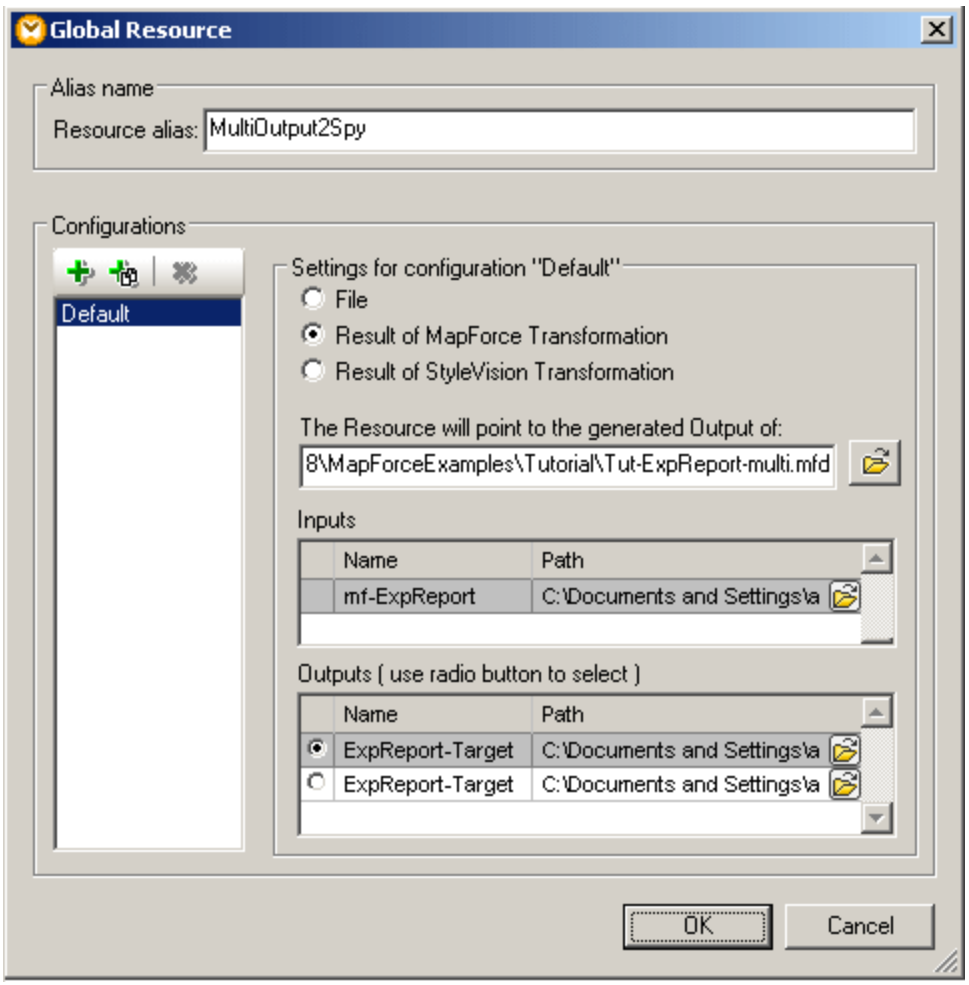
This mapping uses two output components to produce two types of filtered output; Travel and Non-travel expenses of the expense report input file. This section uses the **Tut-ExpReport-multi.mfd** mapping file available in the C:\Documents and Settings\<username>\My Documents \Altova\MapForce2008\MapForceExamples\Tutorial\ folder.



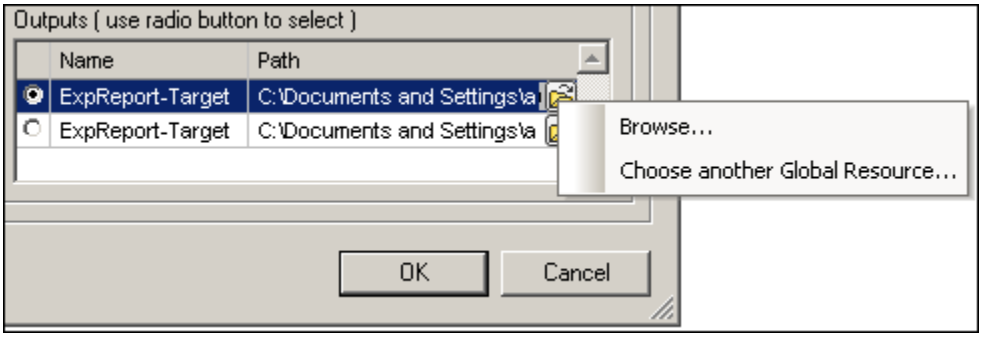
1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **File** from the popup.
3. Enter the name of the Resource alias e.g. **MultiOutput2Spy**
4. Click the "**Result of MapForce Transformation**" radio button and select the **Tut-ExpReport-multi.mfd** mapping.



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.



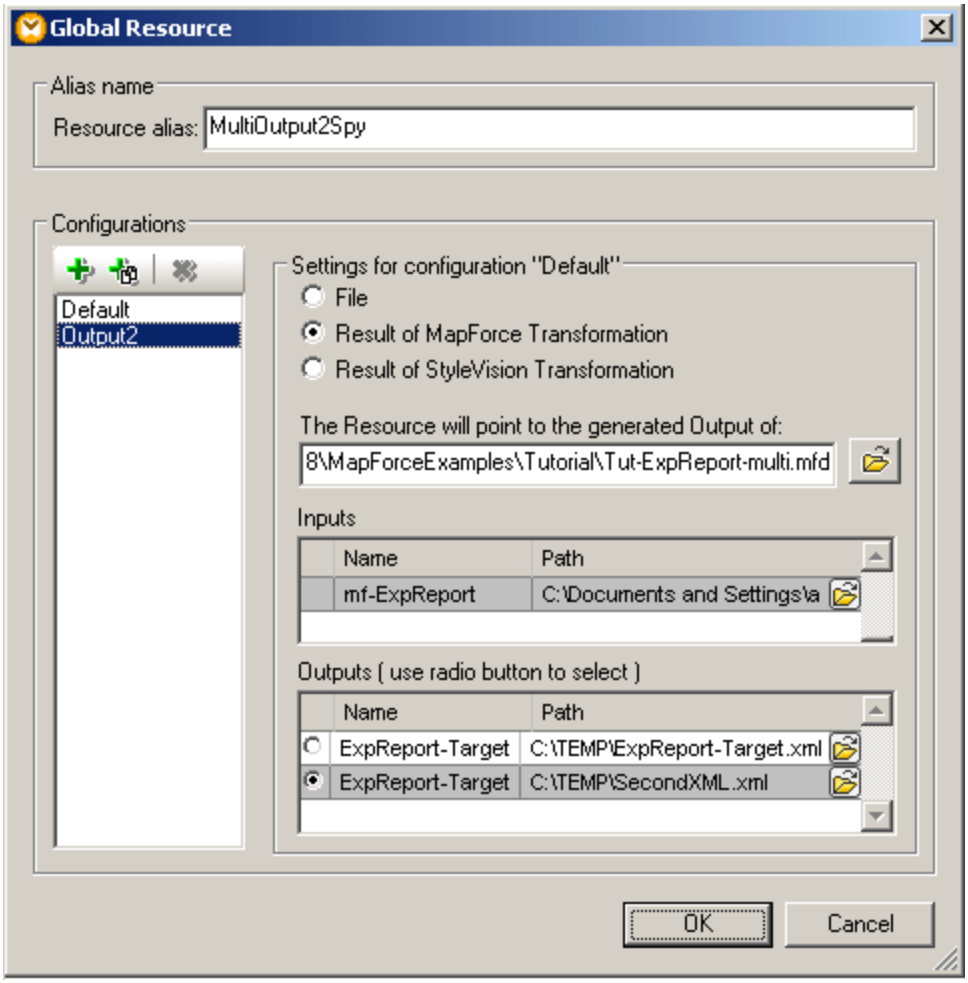
5. Click the top radio button entry in the **Outputs** section, if not already selected. Note that the output file name is **ExpReport-Target.xml** and that we are currently defining the **Default** configuration.
6. Click the "Save as" icon to define the **new** location of the output file, and select **Browse...** from the popup menu.



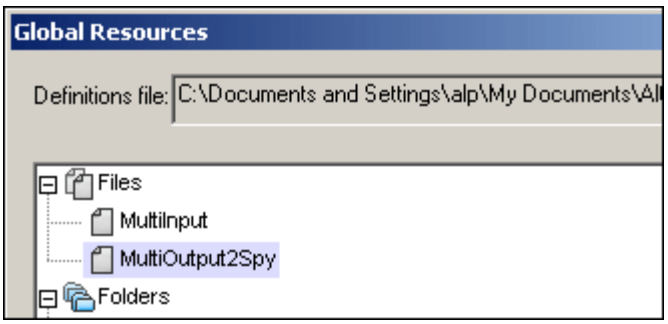
7. Enter the new output location e.g. C:\Temp. This location can differ from the location defined in the component settings.
8. Click the Add button of the Configurations group, to add a new configuration to the resource alias.
9. Enter the name of the configuration, e.g. **Output2**, and click the lower 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.

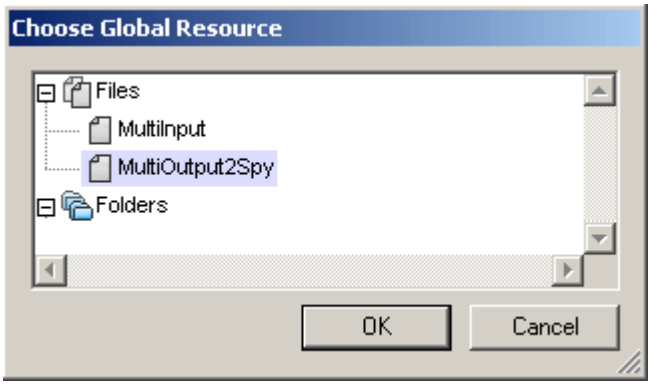


12. Click OK to complete the definition phase.

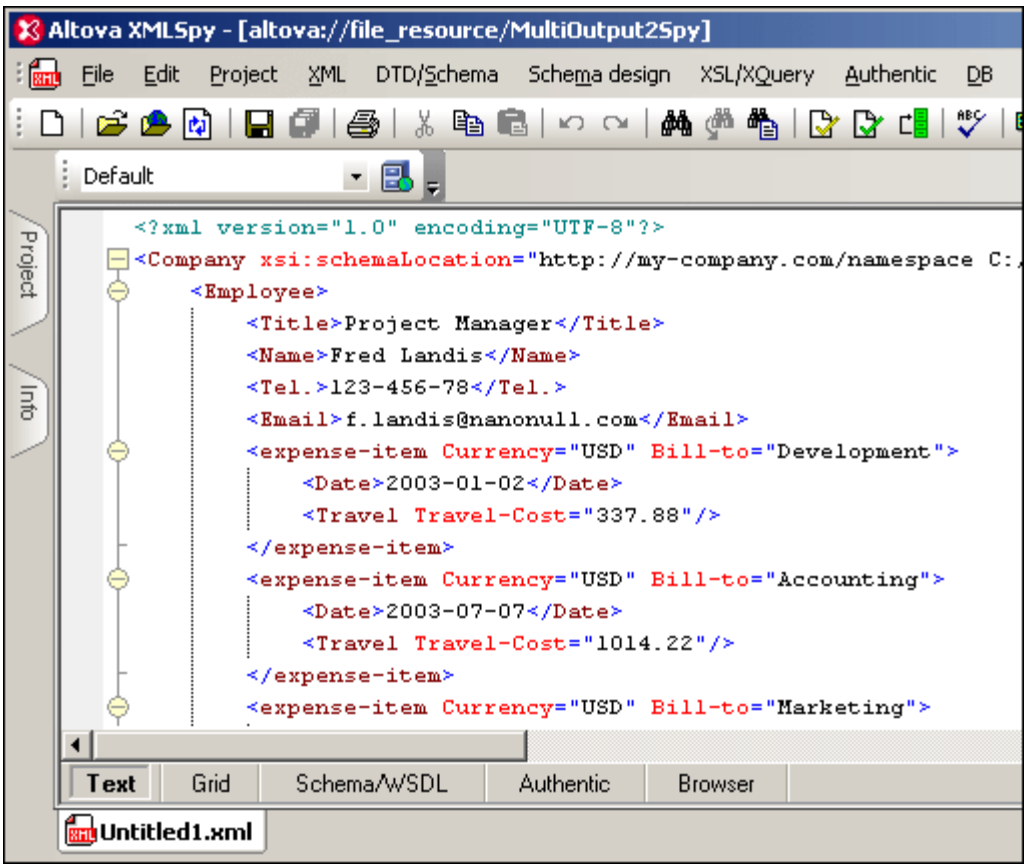
6.3.1 Start application workflow

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 **File | Open Global Resource** in XMLSpy.
3. Select **MultiOutput2Spy** and click OK.



A message box stating that MapForce is transforming appears, with the result of the transformation appearing in the Text view window.



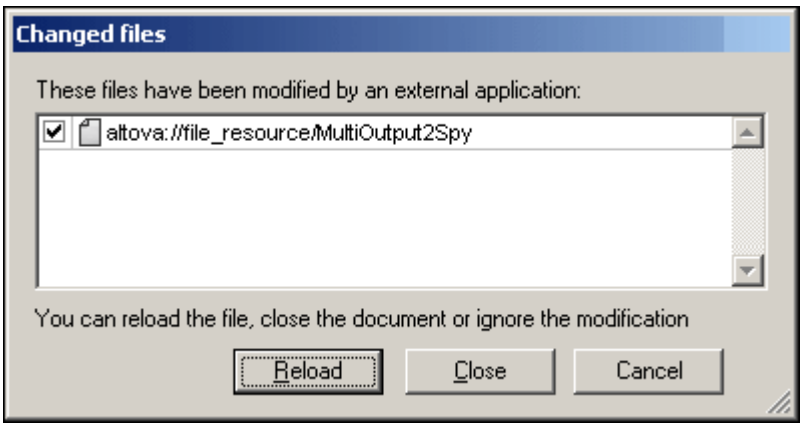
Note:

- The currently selected configuration is "**Default**".

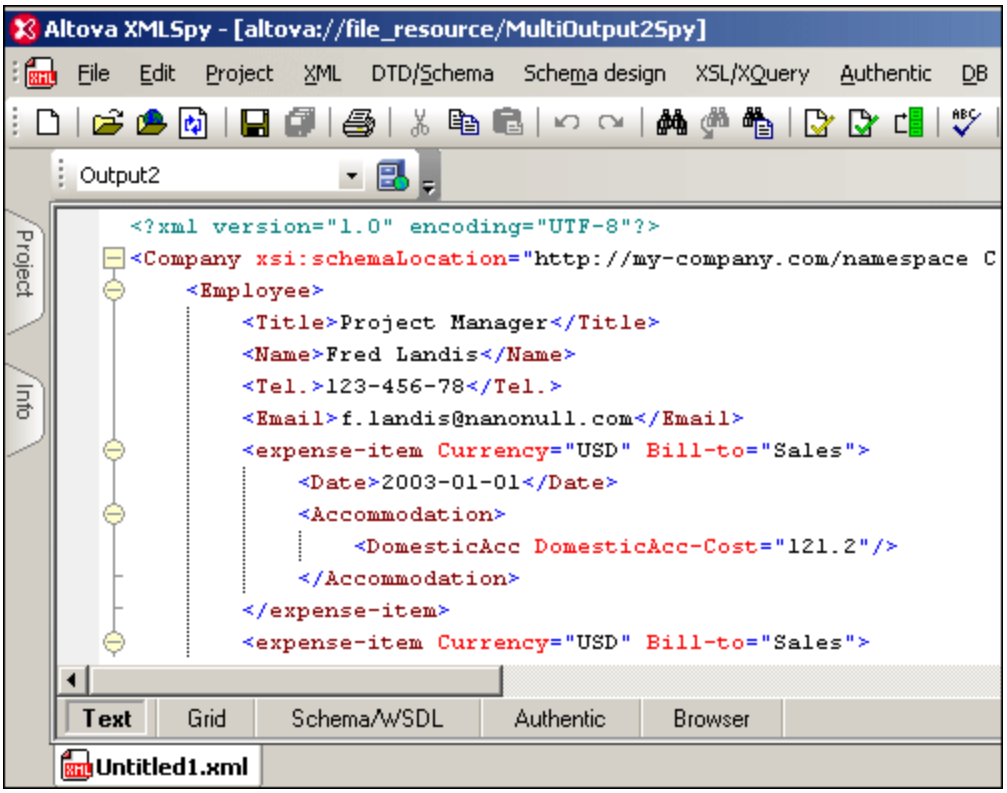
- The name of the resource alias is in the application title bar **altova://file_resource/MultiOutput2Spy**.
- The output file has been opened as "Untitled1.xml" for further processing.
- The **ExpReport-Target.xml** file has been copied to the C:\Temp folder.

To retrieve the non-travel expenses output:

1. Click the Global Resources combo box and select "**Output2**".
   A **Changed file** 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:

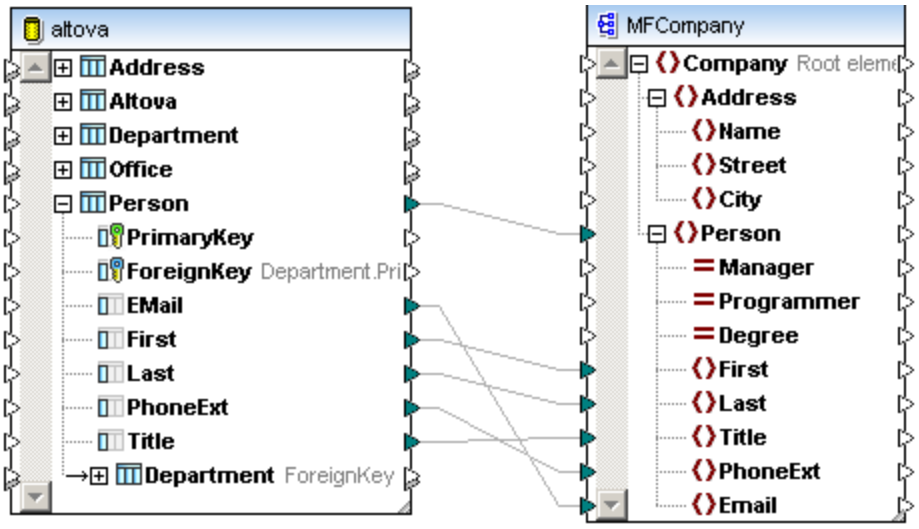
- The currently selected configuration is "**Output2**"

- The output file has been opened as "Untitled1.xml" for further processing.
- The **SecondXML.xml** file has been copied to the C:\Temp folder.

6.4 Global Resources - Databases

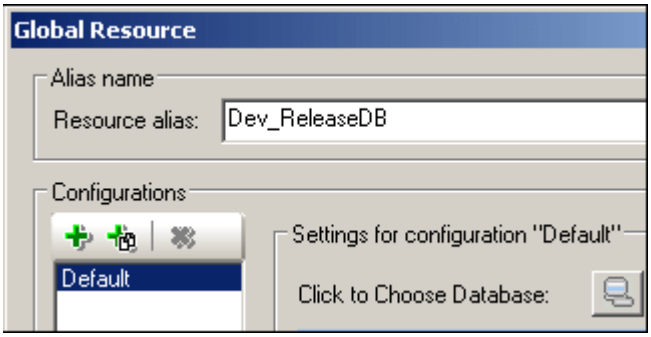
Databases components can also be defined as a global resource allowing you to refer to different databases, for development or release cycles for example. Database resources can be both source and target components.

The mapping file used in this section is available as "**PersonDB.mfd**" in the C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial\ folder.

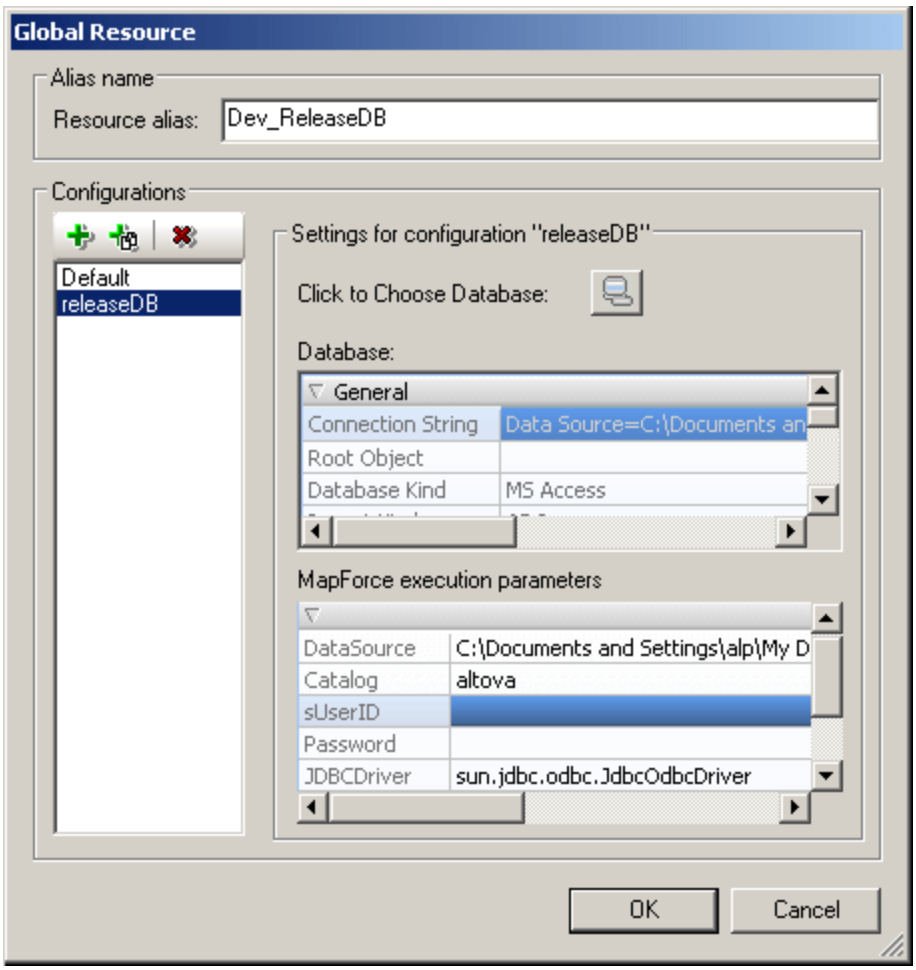


Defining / Adding a database global resource

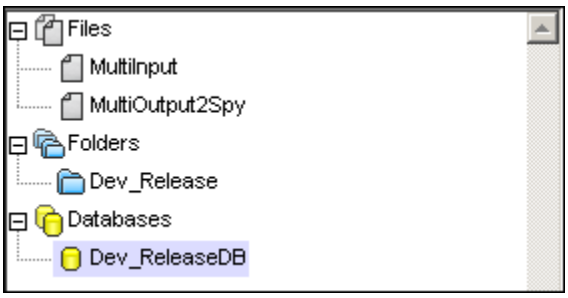
1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **Database** from the popup.
3. Enter the name of the Resource alias e.g. **Dev_ReleaseDB**.



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.

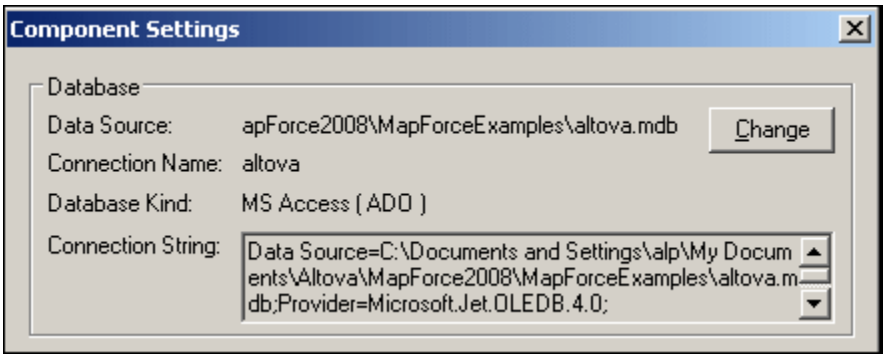
The MapForce execution parameters list box is filled with default values from the database as soon as the database has been selected. These parameters are used when generating **code** and generating the **output preview** in the MapForce engine.

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

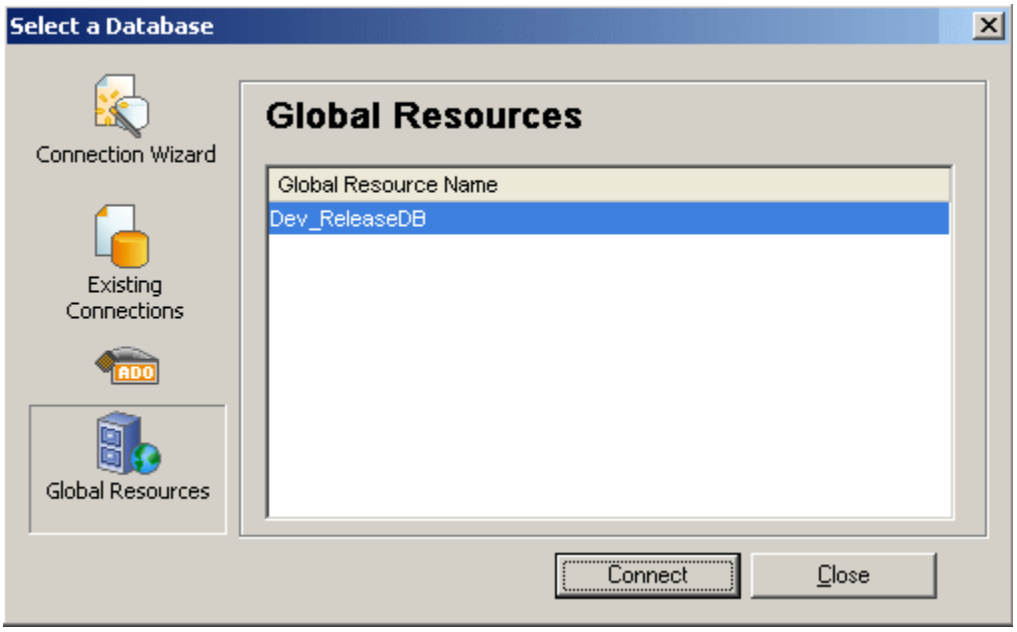
through the Set Root Object button, or defer the selection until later when clicking the Skip button.

Assigning the database resource:

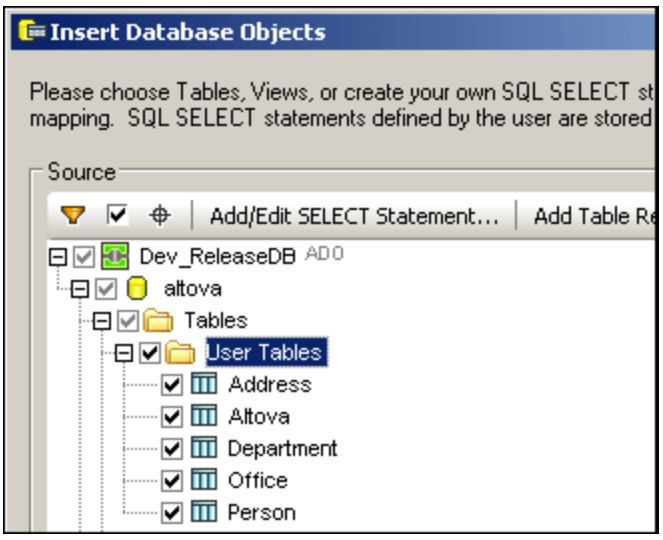
1. Double click the **Altova** database component and click the "Change" button.



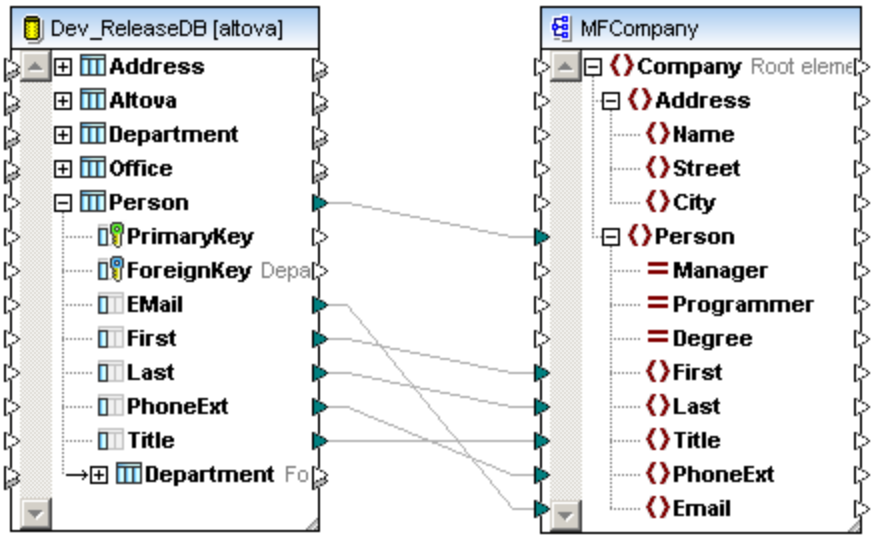
This opens the Select a Database dialog box.



2. Click the Global Resources button/icon, select the **Dev_ReleaseDB** entry and click Connect.

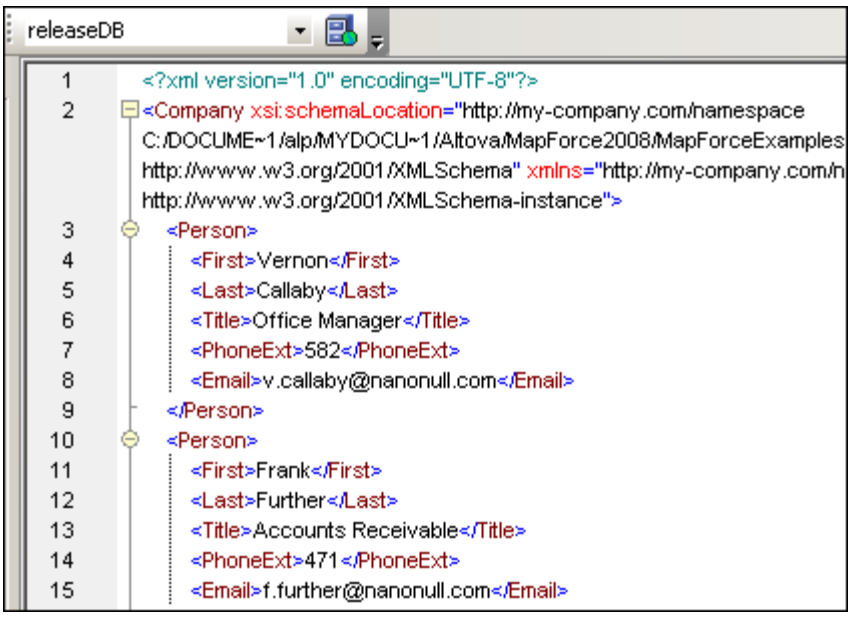


3. Select the tables that you want to be available in the component, click the Insert button, then click OK.
   The resource alias is now visible in the component header **Dev_ReleaseDB [Altova]**.



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.



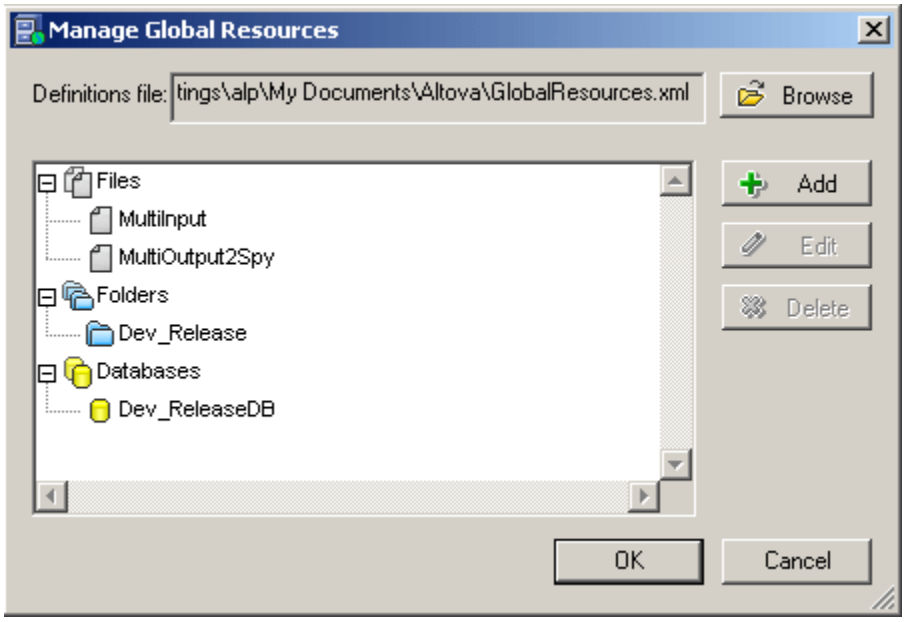
The data from the **release** database is now displayed. As both databases are identical, there is no visible difference in this example.

6.5 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** from the "Definitions file" and select the one you want to use from the "Open..." dialog box.



Whenever changes are made to the Global Resources XML file in one application, a message box is automatically opened in the other, (running application) stating that the definition file was modified. You are prompted if you want to reload the file; options are Yes or No.

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.

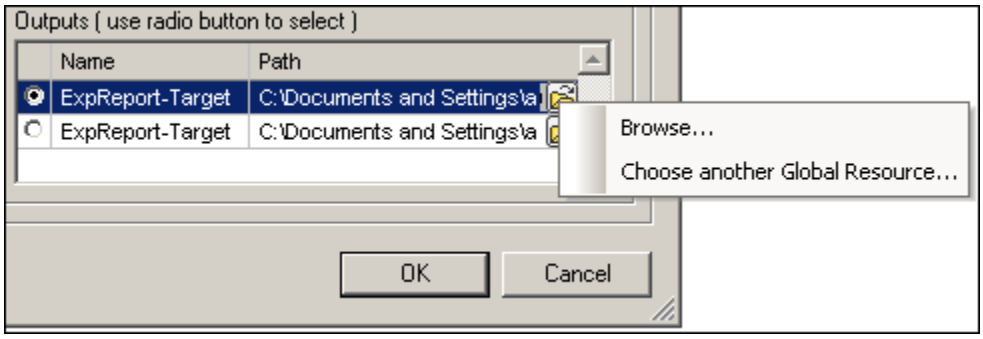
To delete a global resource, select it and click **Delete**.

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.

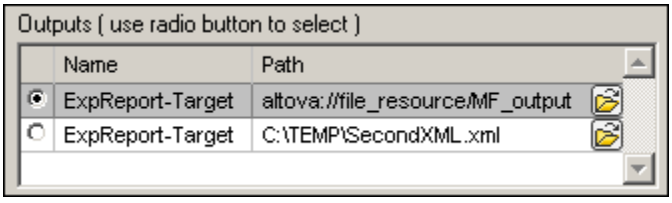
Note: Alias resource names must be unique **within** each of the Files, Folders or Databases sections. You can however define an identical alias name in two different sections, e.g. a multiInput alias can exist in the Files section as well as in the Folders section.

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.

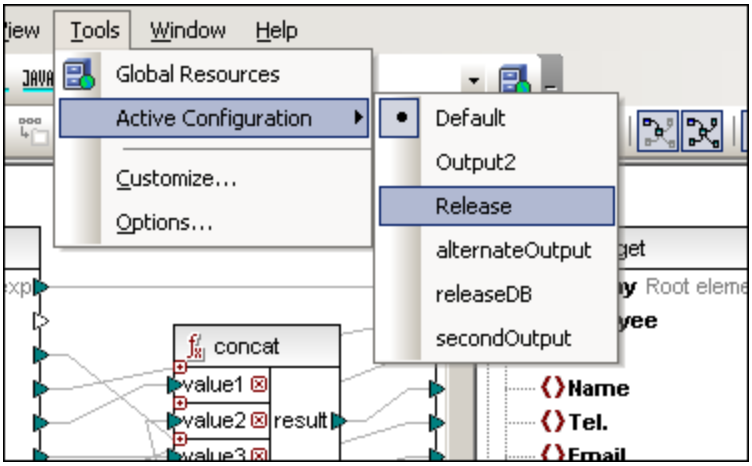
  Selecting a configuration makes that configuration active application-wide. This means

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 7

How To... Filter, Transform, Aggregate

7 How To... Filter, Transform, Aggregate

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

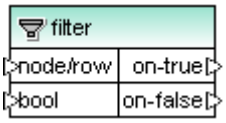
The tasks covered are:

- Mapping multiple tables to one XML file
- How to map data to the root element of target components
- Using the Value-Map function to transform data
- Using the Filter component to lookup / retrieve specific database data
- Using aggregate functions, min, max, sum
- Using boolean values in XSLT 1.0
- Boolean comparison of input nodes
- Defining the Priority context
- MapForce command-line parameters
- Using input functions to override values, and act as parameters in command line execution
- Filter components - Tips
- Checking for existing / not existing nodes
- Type conversion checking
- How to define and use exceptions
- Previewing mappings, MapForce Engine

7.1 Filter - retrieving dynamic data, lookup table

If you want to retrieve/filter data, based on specific criteria, please use the **Filter** component. The mapping shown below is available in the ...\Tutorial folder as **lookuptable.mfd**.

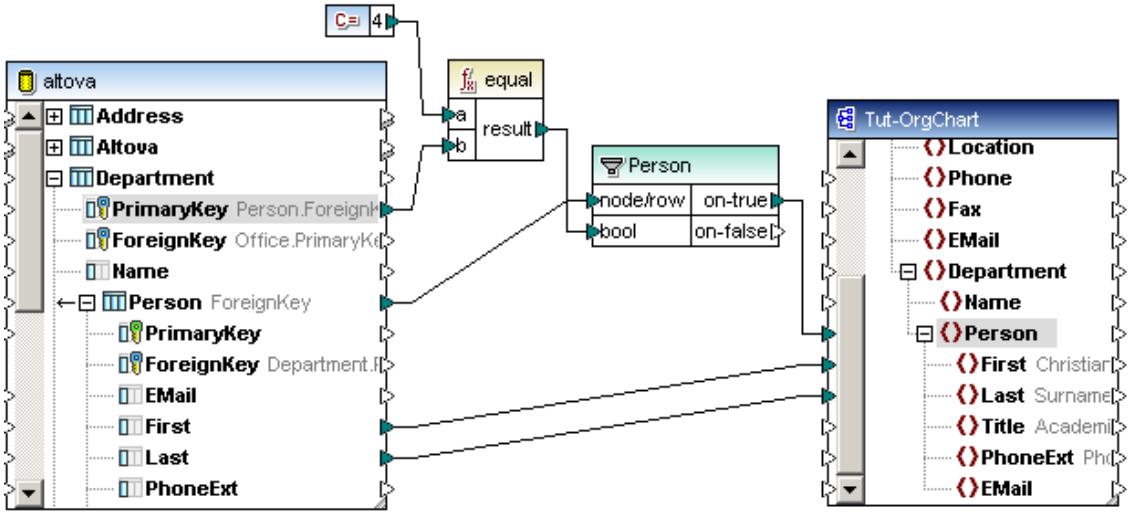
1. Click the filter icon  in the icon bar to insert the component.
   This inserts the filter component in its "unconnected" form where "filter" is visible in the title bar of the component.

   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.

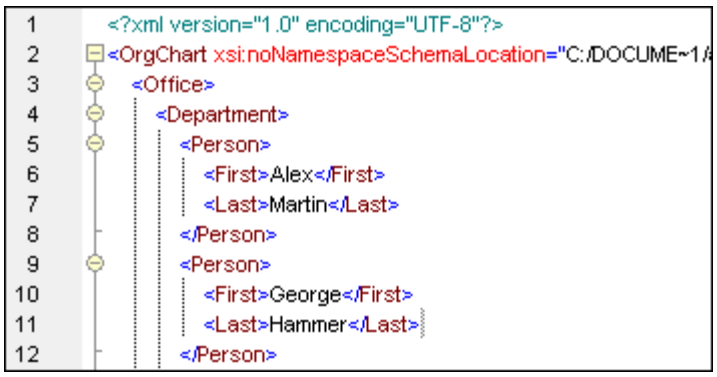
Goal: to map all First/Last names from the **Person** table, where the Department **primary** key is equal to 4.

- 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

7.1.1 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 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.

7.2 Value-Map - transforming input data

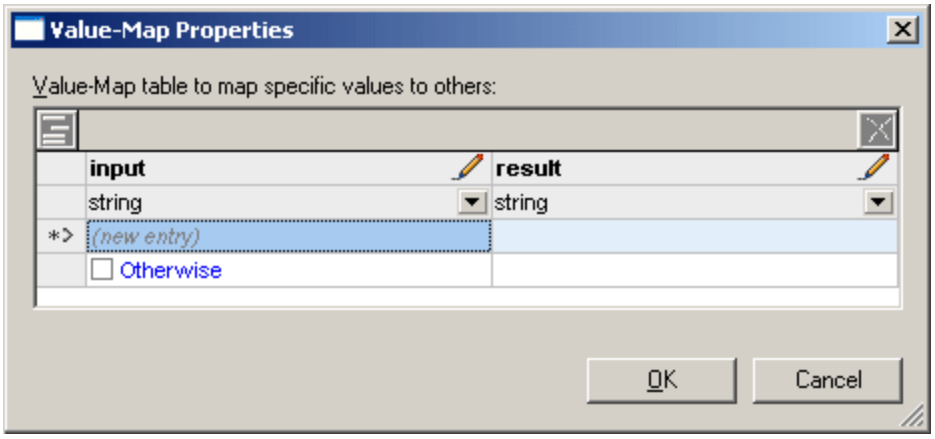
The Value-Map component allows you to transform an input value to a different output value using a lookup table. This is useful for converting different enumeration types. The component only has one input and output item.

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.

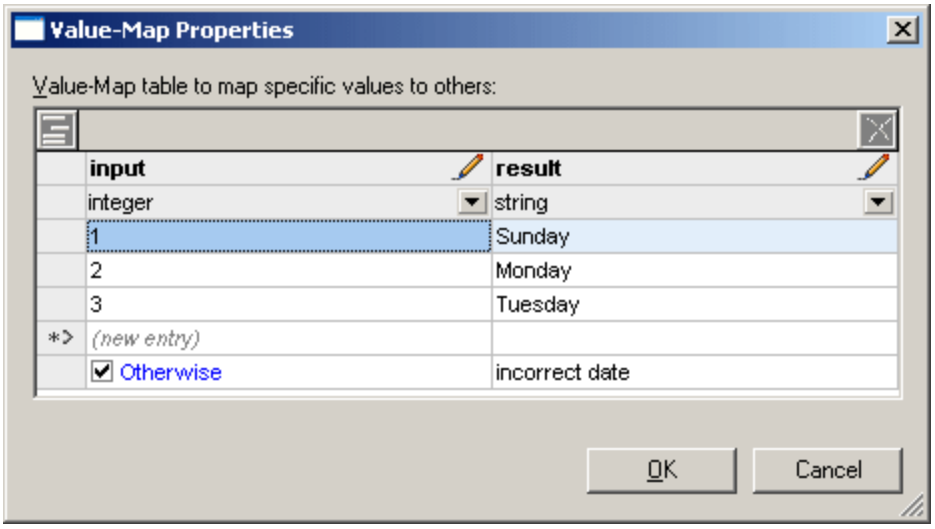
To use a Value-Map component:

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.

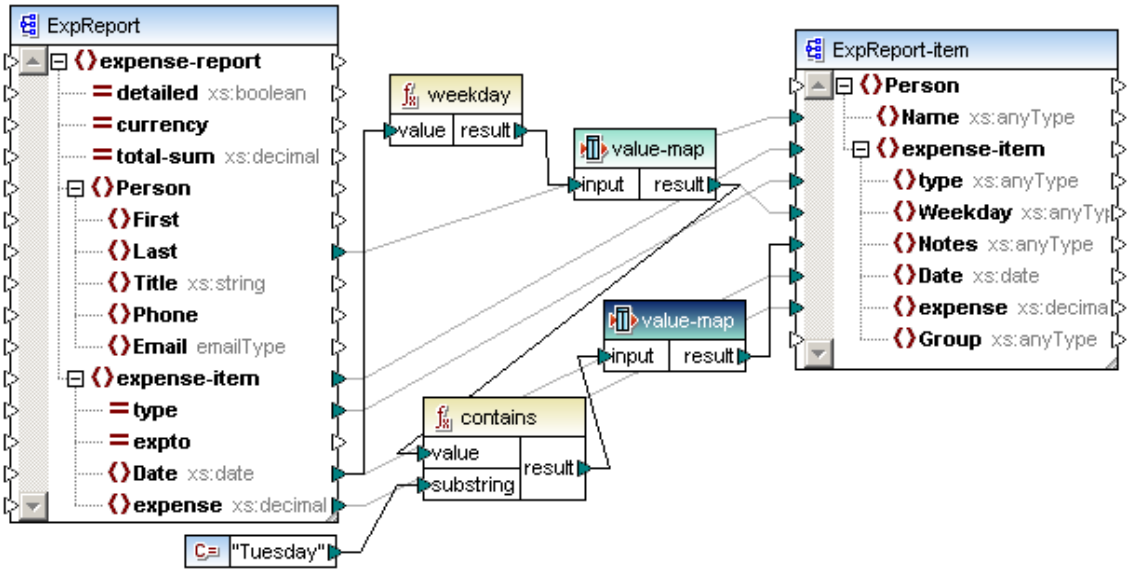


3. Enter the input value that you want to transform, in the **input** column.
4. Enter the output value you want to transform that value to, in the **result** column.
5. Simply type in the ***(new entry)*** input field to enter a new value pair.
6. Click the **datatype** combo box to select the input and result datatypes, e.g. integer and string.



7. Activate the **Otherwise** check box to define an alternative output value if the supplied values are not available on input.
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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder is a sample mapping that shows how the Value-Map can be used.

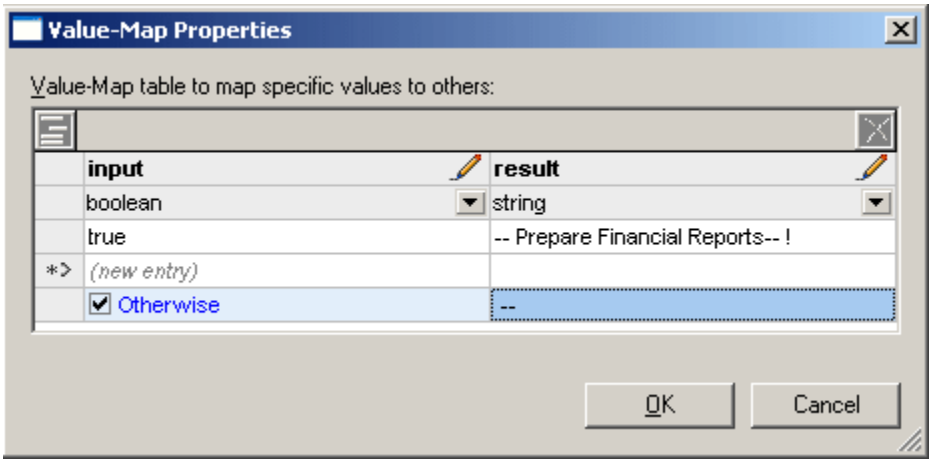


What this mapping does:
Extracts the day of the week from the Date item in the data source, converts the numerical value into text, and places it in the Weekday item of the target component i.e. Sunday, Monday etc.

- 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 top Value-Map component transforms the integers into weekdays, i.e. Sunday,

Monday, etc. as shown in the graphic at the top of this section.

- The output of the Value-Map component is mapped to the **Weekday** item in the target component, as well as to the **value** item of the **contains** function.
- If the output contains "Tuesday", as supplied by the Constant component connected to the **substring parameter,** then the corresponding output "Prepare Financial Reports" is mapped to the **Notes** item in the target component.



- Note that 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.
- Clicking the Output tab displays the target XML file with the transformed data.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Person xsi:noNamespaceSchemaLocation="C:/PROGRA~1/A
3    <Name>Landis</Name>
4    <expense-item>
5      <type>Meal</type>
6      <Weekday>Tuesday</Weekday>
7      <Notes>-- Prepare Financial Reports-- !</Notes>
8      <Date>2003-01-01</Date>
9      <expense>122.11</expense>
10   </expense-item>
11   <expense-item>
12     <type>Lodging</type>
13     <Weekday>Monday</Weekday>
14     <Notes>---</Notes>
15     <Date>2003-01-14</Date>
16     <expense>122.12</expense>
```

7.2.1 Value-Map component properties

Actions:

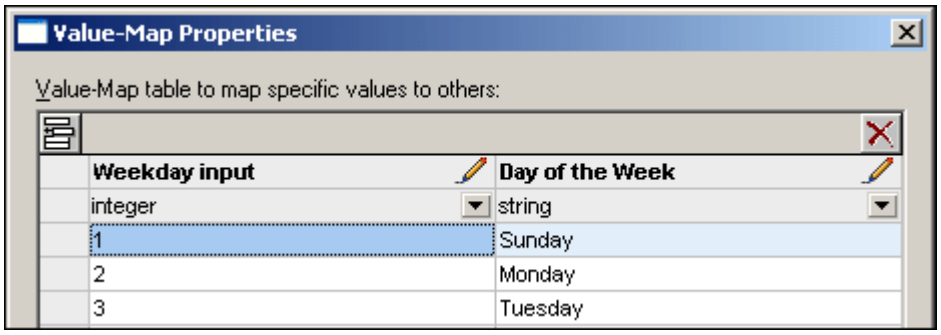
Click the insert icon to **insert** a new row before the currently active one.

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

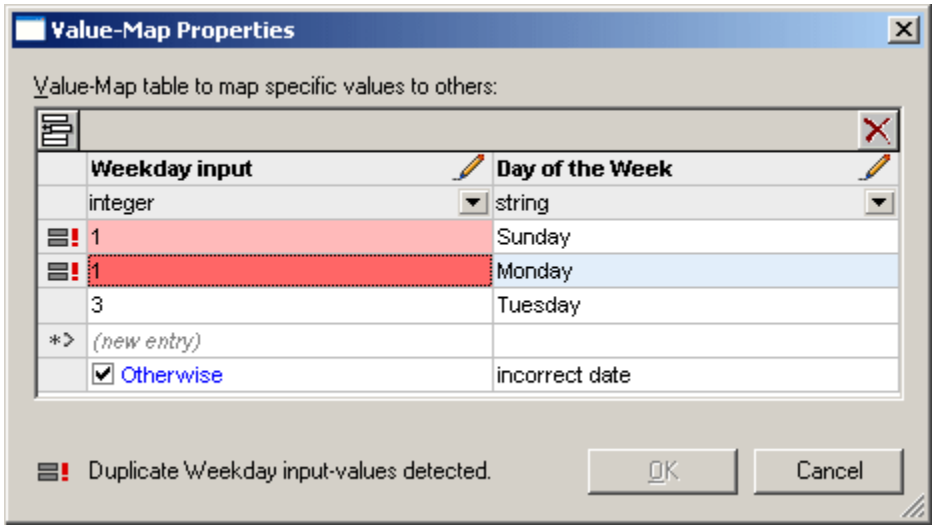
Click the edit icon to **edit** the column header.

You can also reorder lines by dragging them.

Changing the column header:
Double clicking the column header, or clicking the pencil icon, allows you to edit the column name and change it to something more meaningful. This will make it easier to identify the purpose of the component, as the column names are also displayed in the mapping.



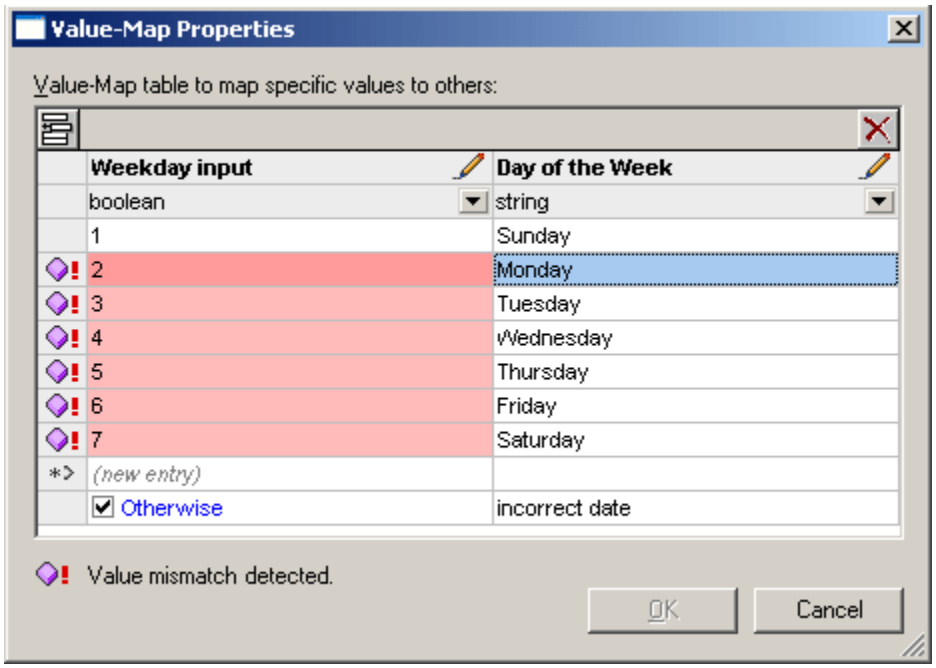
Using unique Input values:
The values entered into the input column must be unique. If you enter two identical values, both are automatically highlighted for you to enable you to correct one of them.



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

Input and output datatypes
The input and result datatypes are automatically checked when a selection is made using the combo box. If a mismatch occurs, then the respective fields are highlighted and the OK button is disabled. Change the datatype to one that is supported.

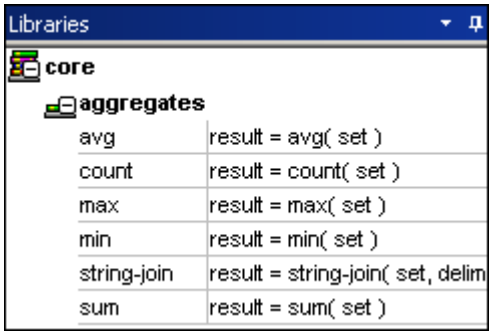
In the screenshot below a boolean and string have been selected.



7.3 Aggregate functions: min, max, sum

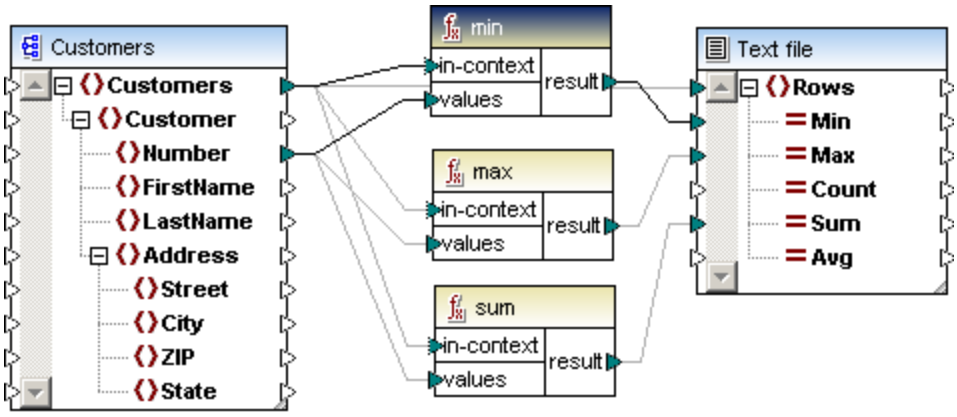
Aggregate functions perform operations on a set of input values (or a sequence of values) as opposed to a single value. The most obvious ones are min, max, count, avg (average), and sum. The aggregate functions can all be found in the core library. The mapping shown below is available as **Aggregates.mfd** in the ...\Tutorial folder.

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

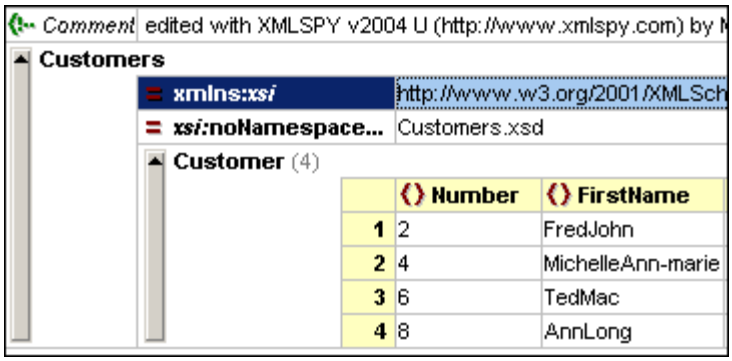


Aggregate functions have two input items.

- **values** is connected to the source item that provides the data, in this case Number.
- **in-context** is connected to the item you want to iterate over, in this case over all Customers.

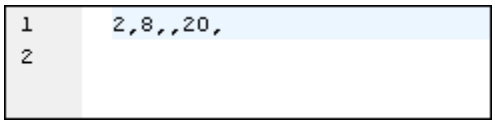


The input instance in this case is an XML file containing the following data:



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:



7.4 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.

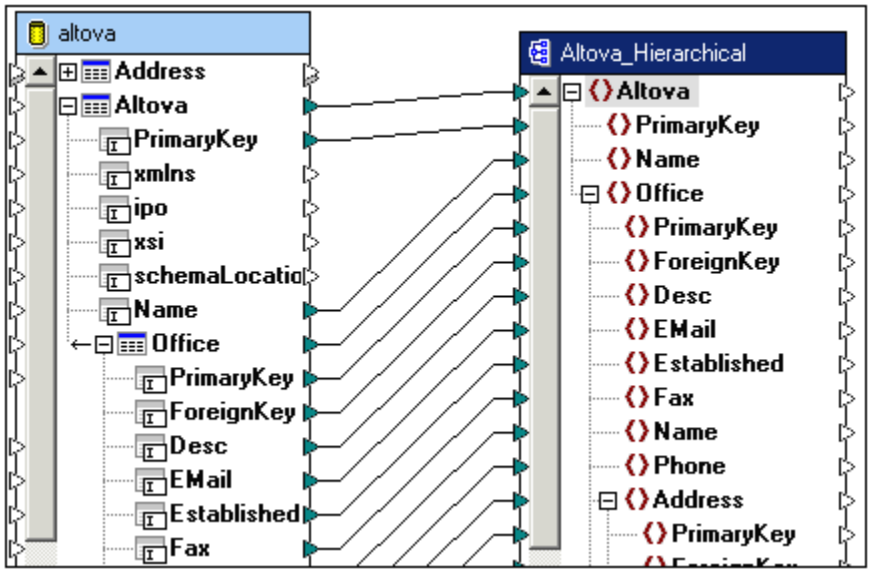
The "**DB_Altova_Hierachical.mfd**" sample file in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder shows how this can be achieved when mapping from an hierarchical database. The Altova_Hierarchical.xsd schema is also supplied in the same folder. The 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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.

Schema prerequisites:

- All tables related to Altova, appear as child items of the target root element.
- To preserve the table relationships all mappings have been created under the Altova table in the database component.

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



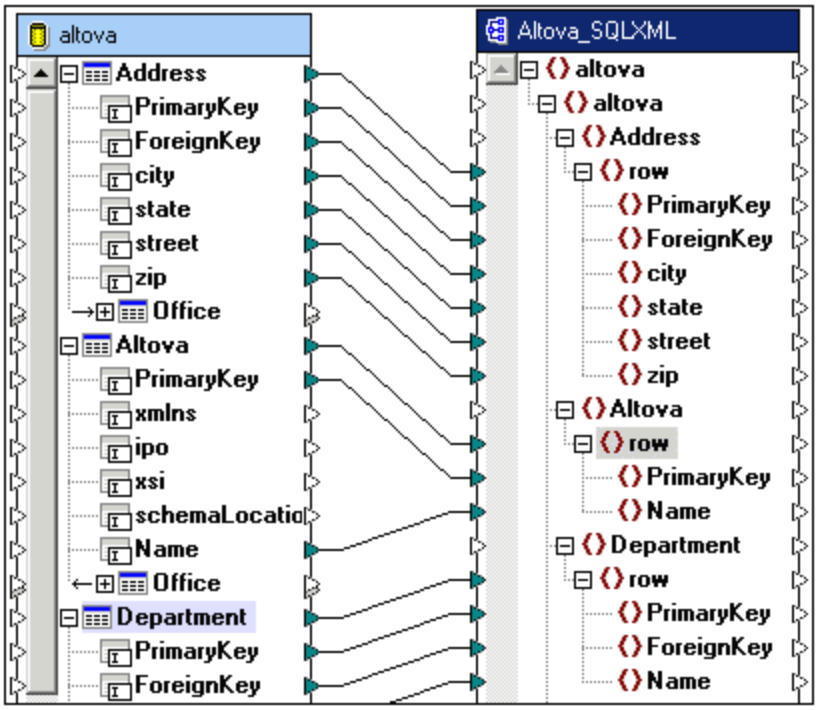
Mapping multiple flat file tables to one XML output file
The following diagram shows the same type of mapping to a flat file SQL/XML database schema.

Schema prerequisites:

- The **schema** structure has to follow the SQL/XML specifications.
- XMLSpy has the ability to create such an SQL/XML conformant file from an SQL database, by using the menu option **Convert | Create Database Schema**. You can

then use the **schema** as the target in MapForce.

- In this case each table name is mapped to the row child element, of the same element name in the schema, i.e. Address is mapped to the **row** child element of the **Address** element



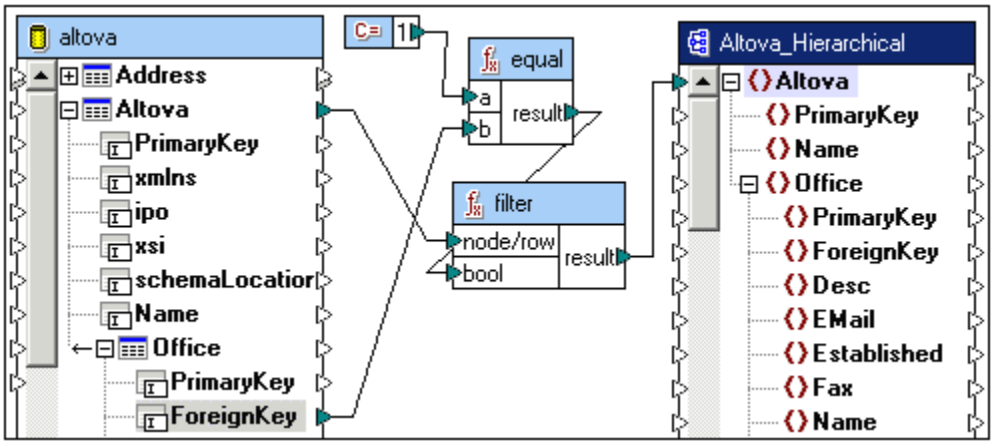
- Please note that the above example **DB_Altova_SQLXML.mfd**, does not preserve the table relationships, as mappings are created from several different "root" tables.

7.5 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.

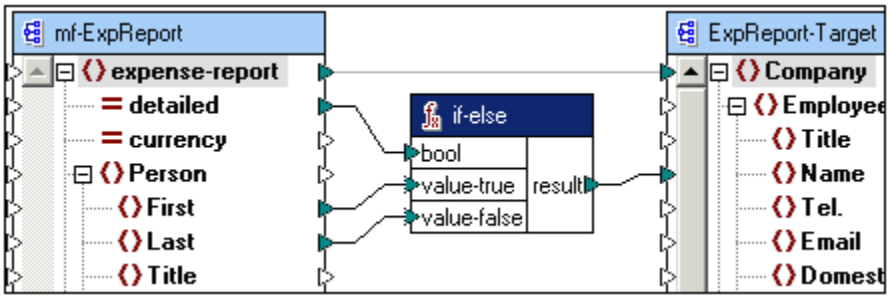
7.6 Boolean values and XSLT 1.0

XSLT 1.0 processors can only process values as strings or numbers. The values supplied by the "detailed" element in this example, can only be "true" or "false" (as defined in the schema file).

The example below tries to create an **IF-ELSE construct**, using the bool value of "detailed". Depending on the content, you should either see the First, or Last name of the Person element in the Target schema.

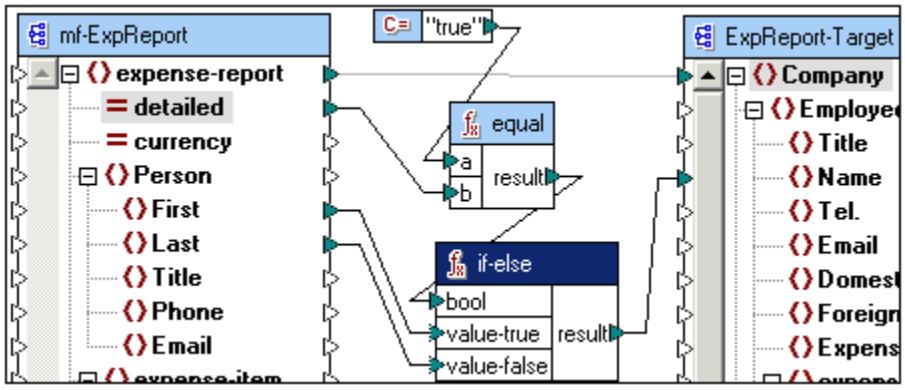
Trying out this mapping however, shows that whatever the bool value of detailed is, true or false, you will always have the contents of First in the target schema. XSLT currently takes **all string input** as True, so this method cannot be used to directly check a boolean value.

Clicking the "Insert Condition" icon  inserts the IF-Else condition function.



To use boolean values as comparison values in XSLT:

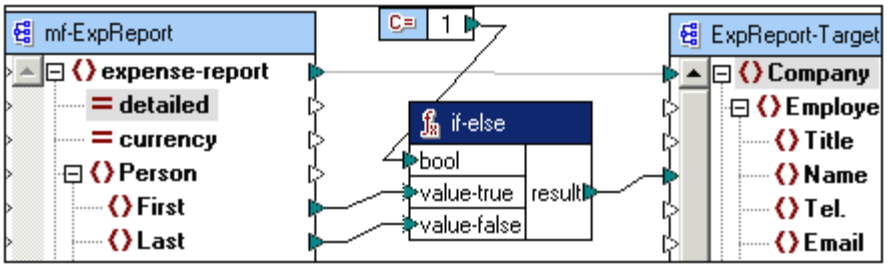
1. Supply a boolean value using the **constant** component, e.g. true.
2. Use the **equal** component to check if the value of the constant, is equal to the content of the boolean node, detailed.
3. Pass the **result** of the comparison on to the **bool** parameter of the **if-else** condition. If the **detailed** element supplies **true**, then the equal result parameter is also true.



- If the bool value (of if-else) is **true**, then the value of **First** is passed on to the target schema.
- If **false**, then the value of **Last** is passed on to the target schema.

Forcing boolean values:

There might be instances where you want to predefine, or force the result of a condition.



1. Connect the constant component directly to the **bool** parameter of an if-else/filter component.
2. Select the **Number** radio button in the "Insert constant" dialog box, and
3. Enter **1** for True, and **0** for false - depending on the condition you want satisfied.

7.7 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:

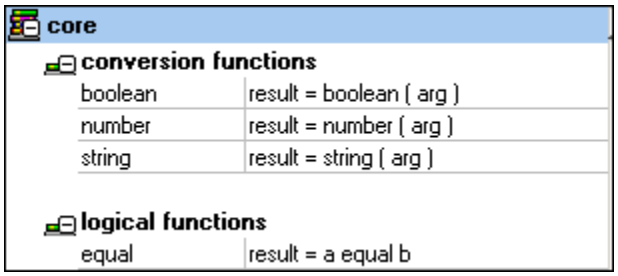
The data type used for the comparison **is always the most general, i. e. least restrictive, input data type** of the two input types.

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 SP2 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.

7.8 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.

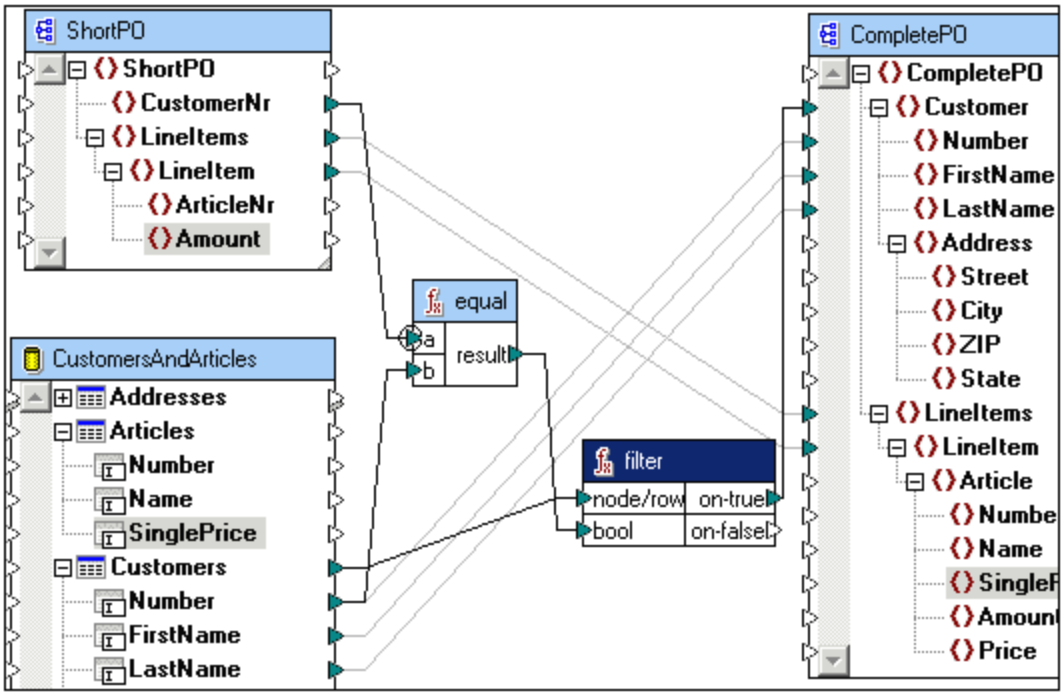
A simplified version of the complete **DB-CompletePO.mfd** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder, is shown below.

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.
- 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 "Standard" user-defined functions which have priority contexts set on their parameters.

If the user-defined function was originally of type "standard" with a priority context, and was subsequently changed to one of type "inline", then the priority context is hidden and deactivated. Changing the same function back to "standard", shows the priority context and enables it once again.

Please see

7.9 Command line parameters

The command line parameter syntax for MapForce is shown below.

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

target	**{ BUILTIN | XSLT | XSLT2 | XQuery | JAVA | CS[:***CSOptions***] | CPP[:***CPPOptions***] }**
CSOptions	**{ VS2008 | VS2005 | VS2003 | VS2002 | BORLAND | MONO }**
CPPOptions	**{ VC9 | VC8 | VC71 },{ MSXML | XERCES },{ 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

Description of general parameters:

filename	path and YourMAPFORCEfile.MFD **If the path, or file name contains a space, please use quotes around the path/file name i.e. "c:\Program Files\...\Filename"**
outputdir	directory the log file is to be placed in
/LOG *logfilename*	generates a log file

Description of target parameters:

/BUILTIN	generates all outputs using the built-in transformation engine
/- */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.
/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

Description of C# options:

VS2008	generates Microsoft VisualStudio 2008 solution files
VS2005	generates Microsoft VisualStudio 2005 solution files
VS2003	generates Microsoft VisualStudio.Net 2003 solution files
VS2002	generates Microsoft VisualStudio.Net (2002) solution files

BORLAND	generates Borland C#Builder 1.0 project-group-files
MONO	generates makefile for MONO environment

Description of C++ options:

VC9	generates Microsoft VisualStudio 2008 solution files
VC8	generates Microsoft VisualStudio 2005 solution files
VC71	generates Microsoft VisualStudio.Net 2003 solution file
MSXML	generates code using MSXML 4.0
XERCES	generates code using XERCES
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

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

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* /**XSLT** *outputdir* /**LOG** *logfilename*

II) generate a Java application and output a log file.

MapForce.exe *filename* /**JAVA** *outputdir* /**LOG** *logfilename*

III) generate a C# application and output a log file.

MapForce.exe *filename* /**CS** *outputdir* /**LOG** *logfilename*

IV) generate a C++ application using the configuration of the mapping settings, and output a log file.

MapForce.exe *filename* /**CPP** *outputdir* /**LOG** *logfilename*

V) generate a C++ application using the /CPP switch, restricting your C++ compiler options.

MapForce.exe *filename* /**CPP:(MSXML|XERCES),(LIB|DLL),(MFC|NoMFC|Builtin)** *outputdir* **[/LOG** *logfilename* **]**

MapForce.exe *filename* /**CPP:MSXML,LIB,MFC**
Generates the C++ application using all of the first choices, in this example:

- compile for C++
- use MSXML4.0
- generate code for static libraries
- have generated code support MFC

MapForce.exe *filename* **/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* **/BUILTIN** *outputdir*

VII) generate all output files using the built-in transformation engine; set the value of the input component X to 10, and the value of the input component Y, to Fred.

Mapforce.exe *filename* **/BUILTIN** *outputdir* **/- /X 10 /Y Fred**

Please note:
When using the built-in transformation engine, the command line values defined here supersede the "Preview Settings | Value", of the input components. Please see Input values, overrides and command line parameters for more information.

VIII) generate all output files using the built-in transformation engine and use all global resources of the global resource file for the specified configuration

Mapforce.exe *filename* **/BUILTIN** *outputdir* **/GLOBALRESOURCEFILE** *globalresourcefilename* **/GLOBALRESOURCECONFIG** *configurationname*

7.10 Input values, overrides and command line parameters

MapForce allows you to create special input components that can:

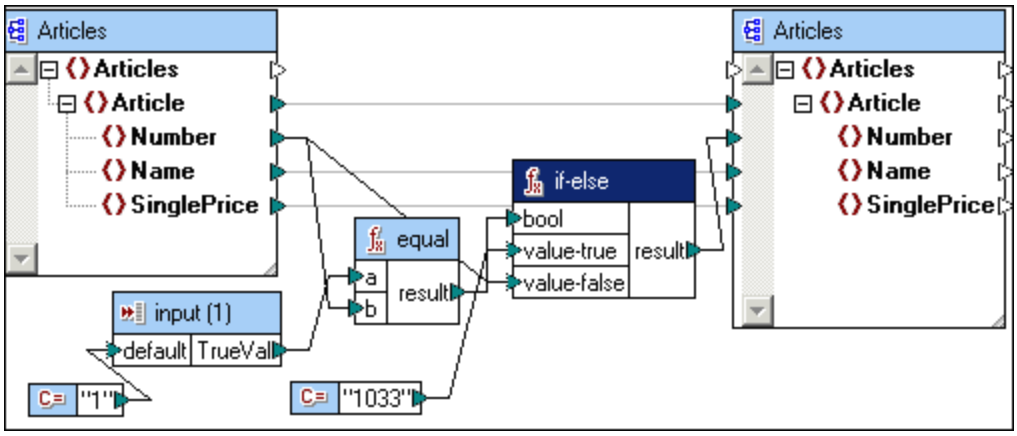
- act as a **parameter** in the command line execution of the compiled mapping, or
- define an **override,** or alternative, value for data being input by the current mapping.

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

The mapping below, uses such an input component. The aim of this mapping is to search for a specific article number, and replace it with a value 1033, if found. If the search is not successful, retain the current number.

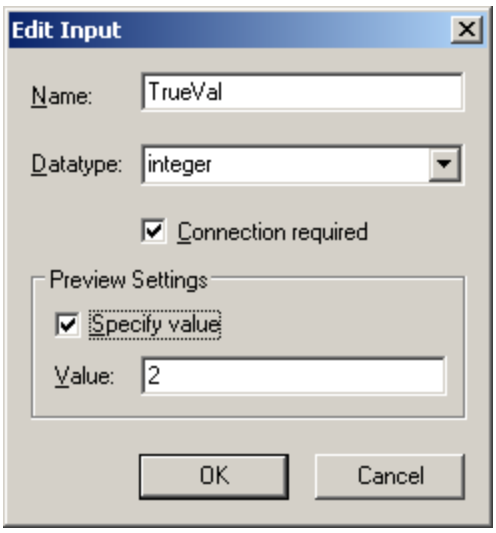
What the input component allows you to do, is override the current input which is 1, and replace it with whatever you define in the input component. Please note that the input in this example is a constant, i.e. 1, but that this will generally not be the case in a complex mappings, where the input can be any type of data from any input source.

The input component further doubles as an **input parameter** for the command line execution of the generated mapping code!



The above example uses the Articles.xsd schema and Articles.xml files, available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder. The article numbers in the source XML file are 1, 2, 3, and 4.

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 and select the datatype you want to use.
3. Click in the **Value** field, of the Preview Settings group, and enter a value. In this case, enter a value **different** from the one supplied by the constant, e.g. 2.



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

The original article number 2, has been changed to 1033. The value supplied by the input component i.e. 2, has taken precedence over the value supplied by the constant, which was 1.

Input values and Code generation:
The preview settings entered in the "Create/Edit input" dialog box are **only** applicable when **previewing** results in the Output tab. They are not used for code generation or command line execution of mappings.

The value supplied by the "**default**" input item/icon is used for the **preview,** if "Specify value" is **inactive**. **Code generation and command line execution** uses this default value if the parameter is not supplied on the command line.

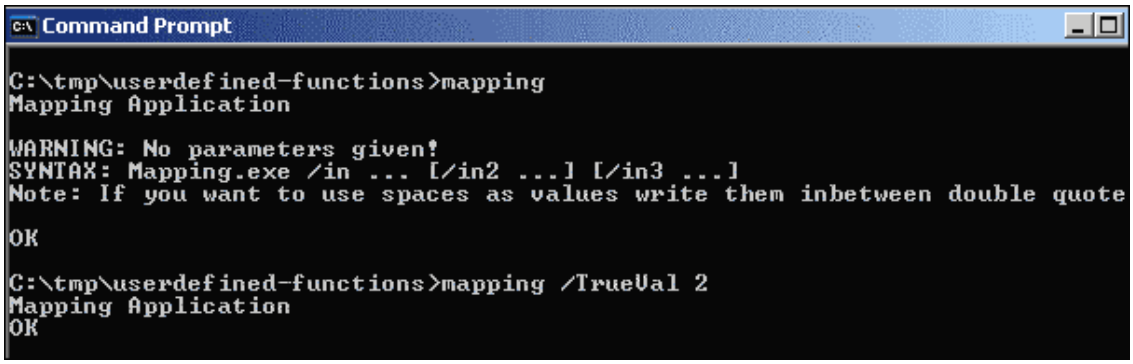
Using input values as parameters in command line execution of mappings:
Input values can be used as parameters when calling the generated mapping. In this example,

- the generated application name is **Mapping.exe**
- the input value name "**TrueVal**" is the first parameter, and
- the input value "**2**" is the second parameter.

The command line thus becomes:

mapping.exe /TrueVal 2

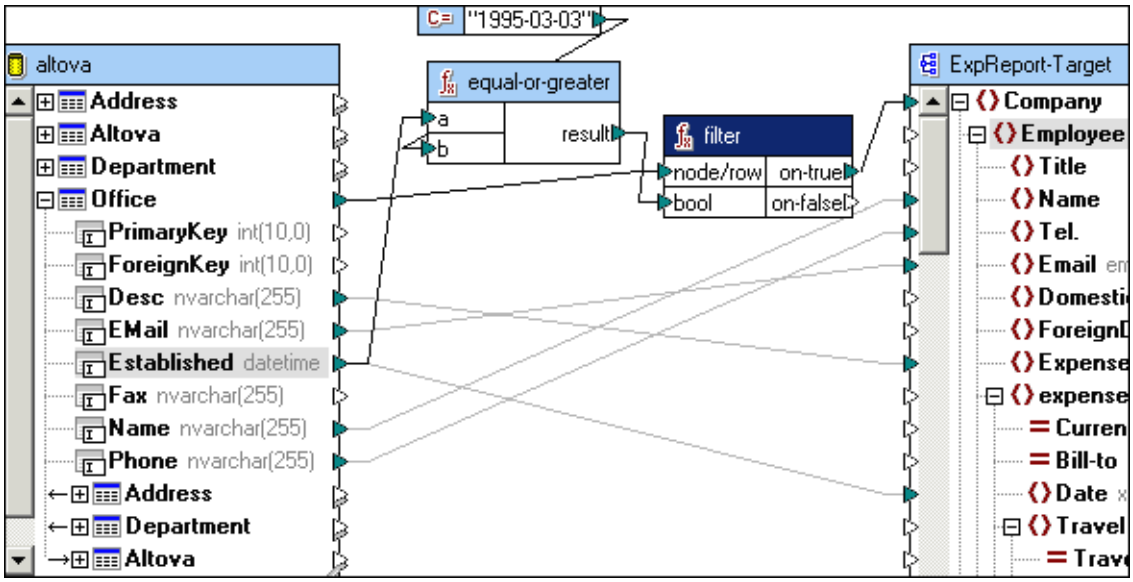
Please note:
Running mapping.exe without parameters, displays a warning message, and help on the command line syntax needed.



7.11 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.

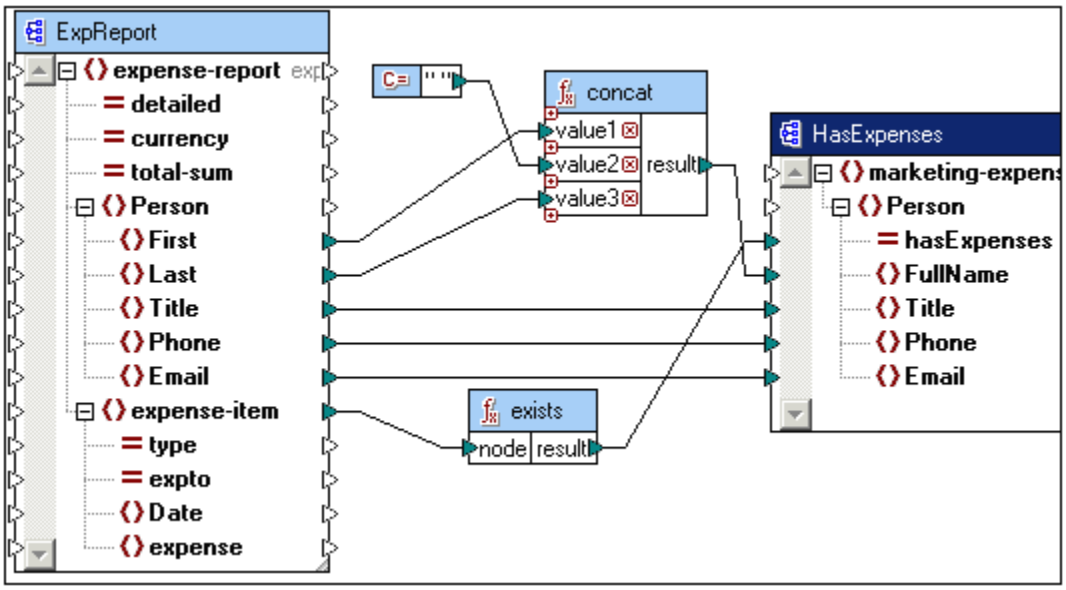


7.12 Node testing, exists / not-exist

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.

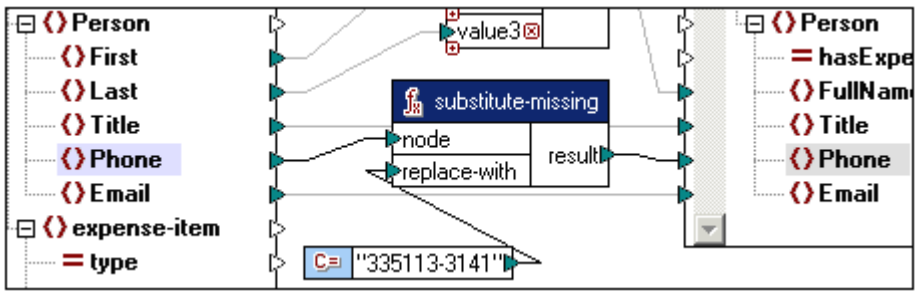
Exists
Returns true if the node exists, else returns false.
The "HasMarketingExpenses.mfd" file in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder contains the small example shown below.

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.



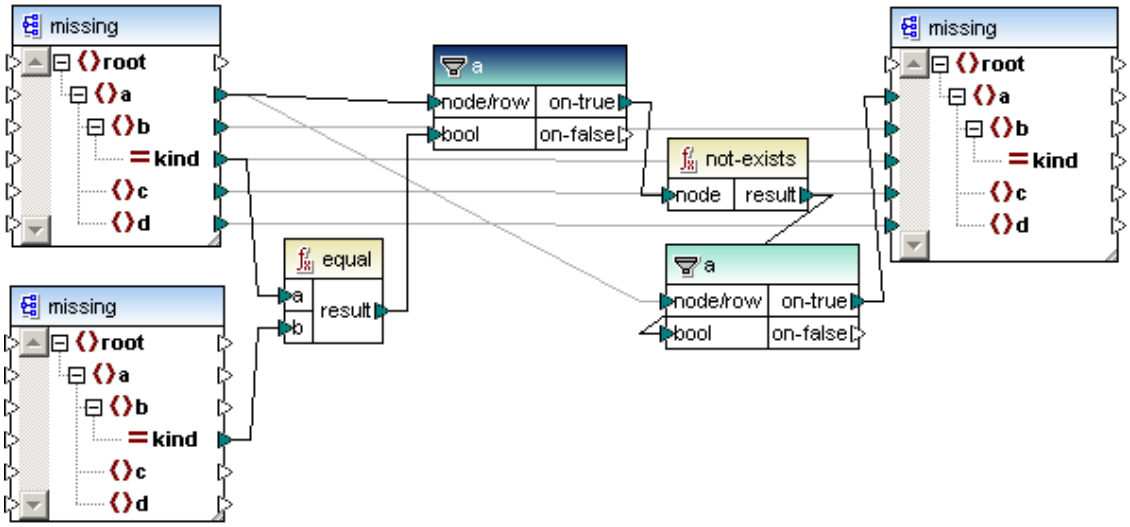
In the image above, the existence of the node "Phone" is checked in the XML instance file. If the node is not present, then the value supplied by the constant is mapped.

7.12.1 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>
```


7.13 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'
        ...
>
```

7.14 Type conversion checking

From MapForce **2006 SP2** on, the generated applications and preview (with the builtin 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 BUILTIN-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 BUILTIN-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"

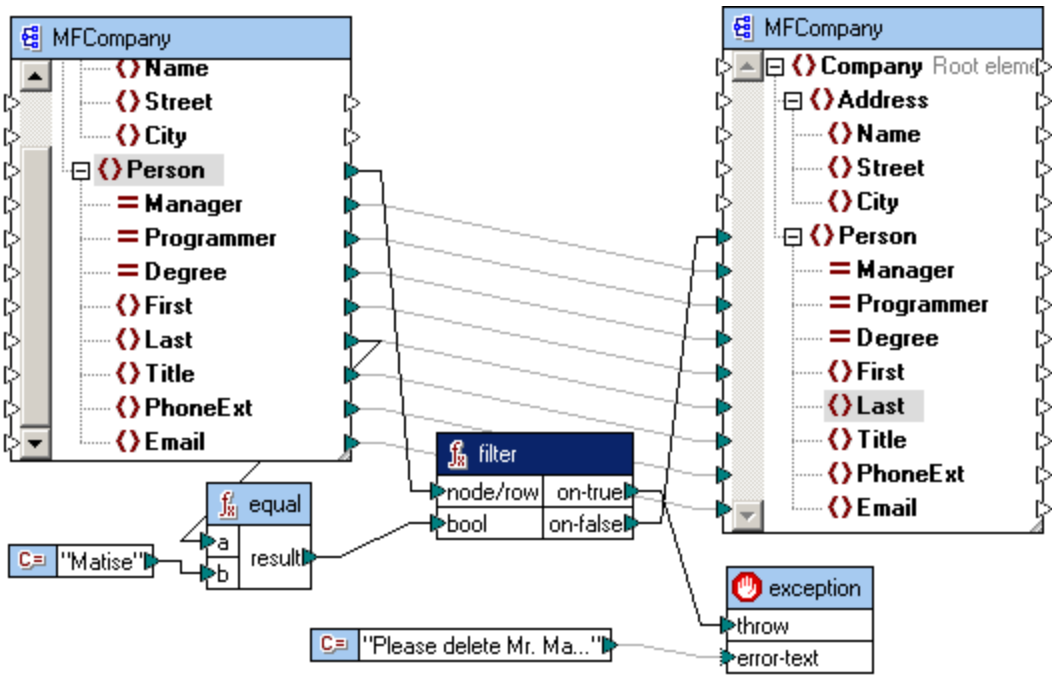
If type-conversion-errors occur, check that the types have been handled correctly. E.g. use the lang:numeric() function, to check if the source-value may be converted into a number, and then an if-else component to pass a different value in case it fails (e.g. a constant containing -1, on the value-false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

7.15 MapForce 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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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 Last name of a person equals "Matise".

- The **equal** component checks to see if Last equals Matise, and the bool result is passed on to the filter component.
- When the condition is satisfied, i.e. Matise is **True**, the **on-true** 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-false parameter, if that is what you need.)
- The error text supplied by the **constant** component is output.
- 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.

7.16 Preview mappings - MapForce Engine

The MapForce 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** tab. The Output tab also supports the find command, enabling you to find any XML data, or SQL statement that you might need to find.

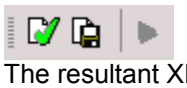
MapForce can also be started from the **command line** and produce a result, without having to generate any intermediate code. We would still, however, recommend that code be generated in one of the respective programming languages (after the development phase) due to the better execution speed of generated code.

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

XML Schema/document as target:
The result of the 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#) is selected, the MapForce engine generates the output and any data source components can be used: XML/Schema files, Text files, databases, etc. The result you would have achieved if the Java, C++, or C# code had been generated, compiled and executed, appears in the Output tab.

The screen shot below shows the output of the **DB_CompletePO.mfd** mapping available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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 icon, and validated against the referenced schema by clicking the validate icon in the icon bar.

Database as target:

SQL pseudo-code is displayed when a database component is the target. The complete list of statements (all Select, Insert, Update or Delete statements), is displayed for you to preview before executing. Please note that these statements are not suitable for executing in any SQL query tool because of possible syntax or value formatting differences. Also, the result of the effective execution will not in all cases be identical to the preview statements, as e.g. table actions are executed conditionally depending on the actual contents of the database.

```
/*
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:\Program Files\Altova\MapForce2007\MapForceExam
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FRON

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
```

Clicking the Run SQL-script icon, executes the SQL select statement and presents you with a report on the database actions, as shown in the screen shot below.

- Actual SQL statement that were executed on the target database
- Multiple table actions if any occurred i.e. "UPDATE ….. -->>> OK. 0 rows affected." and the "INSERT …. -->> OK. 1 rows affected".
- Results of every SQL statement:
  e.g. **OK** and xx rows affected if successful, or **FAILED**, and a detailed error message.

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Program Files\Altova\MapForce2007\MapForceExamp
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM
-->>> OK. One or more rows.

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
-->>> OK. 1 row(s).
```

Text file as target:

The MapForce engine includes support for displaying the results of 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
```

- UN/EDIFACT and ANSI X12 targetsd
- A FlexText template which defines file structure and content used as a target.

MS OOXML Excel 2007 files as target

Microsoft Excel 2007 only needs to be installed if you intend to **preview** Excel 2007 sheets in the **Output** tab. If you use a previous version of Microsoft Excel, or you don't have Excel installed at all, you will still be able to map to/from Excel 2007 files. You cannot preview the result in the Output tab, but you can still save it, by clicking the Save output icon.

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.

Chapter 8

Databases and MapForce

8 Databases and MapForce

MapForce allows you to not only map database data to XML documents, but you can do the reverse as well, map XML data to databases and even create mappings between databases! MapForce takes primary and foreign key constraints into account and also generates transaction data which ensures data integrity.

Please note:

Database access currently requires that you use one of the programming languages: Java, C#, or C++. **XQuery** code can only be generated for XML data sources and targets.

Currently supported databases (and connection types) are:

- Microsoft Access versions 2000 / 2003 /2007
- Microsoft SQL Server versions 2000 and 2005
- Oracle version version 9i and 10g
- MySQL 4.x and 5.x
- Sybase 11, 12.0 and 12.5
- IBM DB2 version 8.x und 9

- Any ADO (compliant database)
- Any ODBC (compliant database)

8.1 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 MapForce 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
- 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;

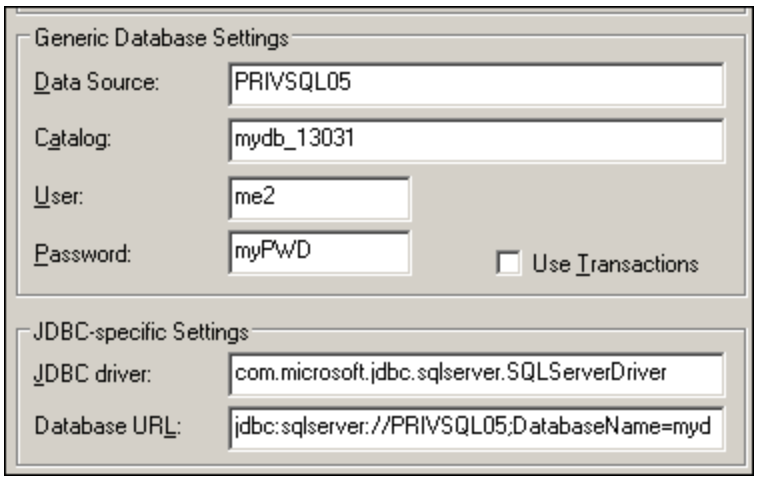
If you are using the integrated Windows Authentication to connect to the MS SQL 2005 database it will also be necessary to use **sqljdbc_auth.dll** depending on the type of machine, 32 bit or 64 bit, and then place this DLL into your Windows System path i.e. Windows/System32.

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

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

8.2 Development environments for code generation

Below is a list of the requirements for each of the development environments, as well as other tools, that are needed when generating code using MapForce.

Java Minimum requirements:
Java SE (Standard Edition) JDK 1.5
Apache ANT 1.5.3

Other vendor supported IDEs:
Eclipse 3.x
Borland JBuilder 8

Optional:
Sun 1 Studio - import of ANT build-file into IDE

C# Minimum requirements:
Microsoft .Net Framework SDK 1.0 or later - for compilation and build process

Additionally:
Microsoft Visual Studio.NET 2002 / 2003
Microsoft Visual Studio 2005
Microsoft Visual Studio 2008
Borland C#Builder 1.0
Mono 1.x

C++ Minimum requirements:

Supported IDEs:
Microsoft Visual Studio 6.0 - for compilation, build process and as IDE.
Microsoft Visual C++ 2003
Microsoft Visual C++ 2005
Microsoft Visual C++ 2008

8.3 Mapping XML data to databases

MapForce allows you to not only map database data to XML documents, but you can do the reverse as well, map XML data to databases and even create mappings between databases! MapForce takes primary and foreign key constraints into account and also generates transaction data which ensure data integrity.

Database functions (table actions) supported by MapForce:

- Insert
- Update
- Delete
- Database key field handling

Examples for each of these table actions follow, and are of a simple nature to get you acquainted with how to achieve the specific goals.

Currently supported databases (and connection types) are:

- Microsoft Access versions 2000 / 2003 /2007
- Microsoft SQL Server versions 2000 and 2005
- Oracle version version 9i and 10g
- MySQL 4.x and 5.x
- Sybase 11, 12.0 and 12.5
- IBM DB2 version 8.x und 9

- Any ADO (compliant database)
- Any ODBC (compliant database)

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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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.

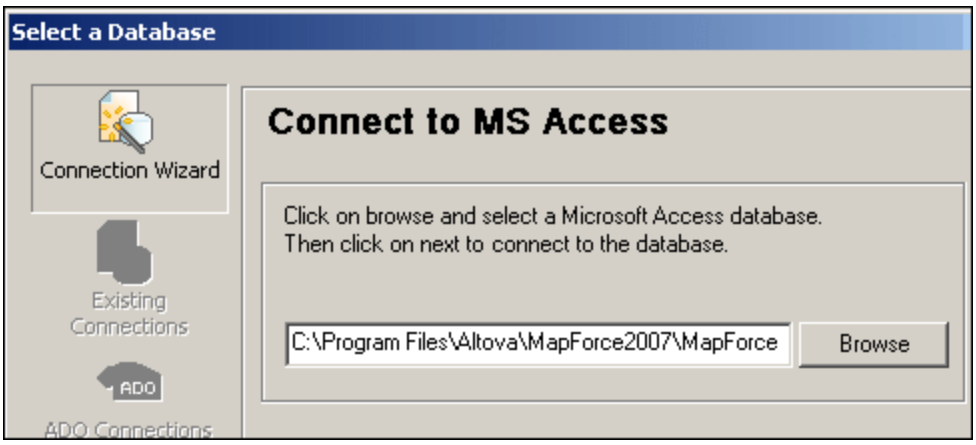
MapForce is able to map from password protected Access files, if they are added via the **Any ODBC** option in the "Select a source database" dialog box. The user and password settings can then be entered in the wizard.

8.3.1 Setup of XML to database mapping

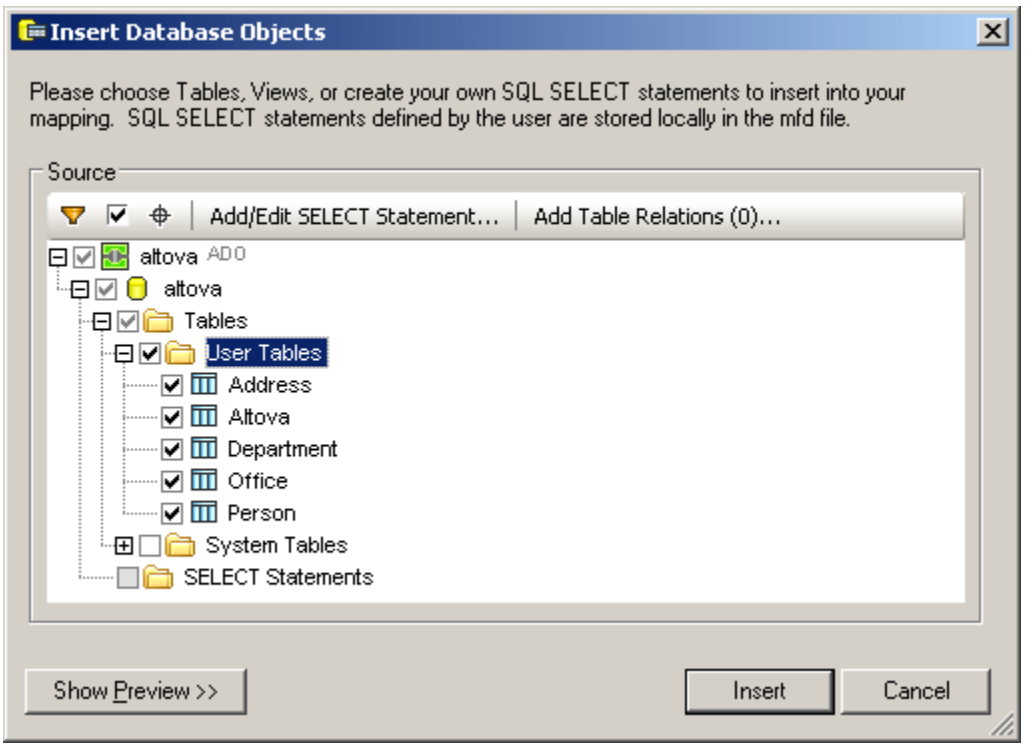
Setting up an XML to database mapping, is in no way different from the methods previously described.

1. Click the **Insert Schema | XML instance** icon, and select the **Altova_Hierarchical.xsd** .
2. 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.
3. Click the **Insert Database** icon, select the Microsoft Access (ADO) entry and click Next.

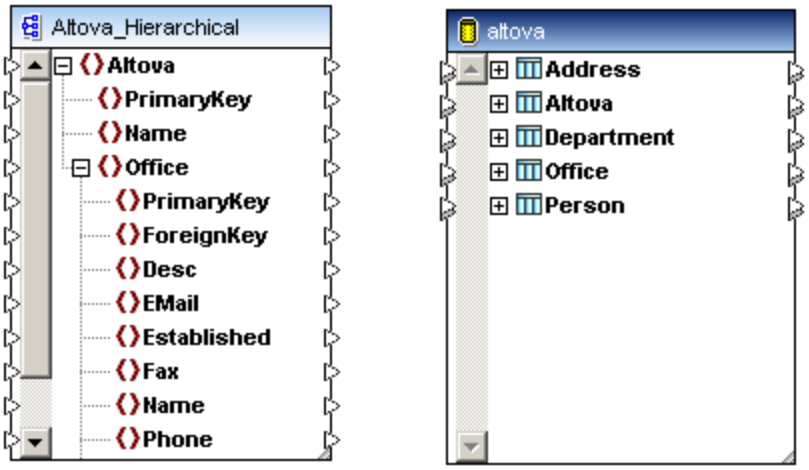
4. Click the **C++** icon in the title bar to specify the language the generated code should support. This setting also loads the language related library into the Libraries window.
5. Click the Browse button to select the **altova.mdb** database available from the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder, and click Next.



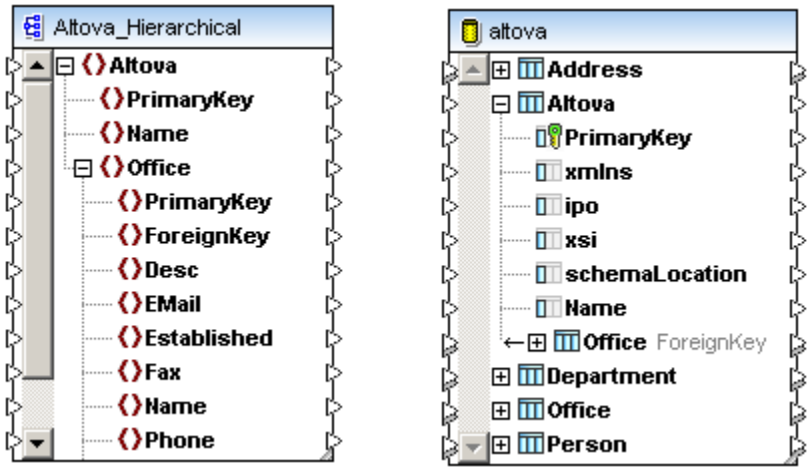
This dialog box allows you to define the specific Tables, Views or System tables that you want to appear in the Database component.



6. Click the check box to the left of User Tables to select them all, and click OK 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
- **Component settings**, allows you to change the component database by clicking the Change button, and using the wizard to select a different database.

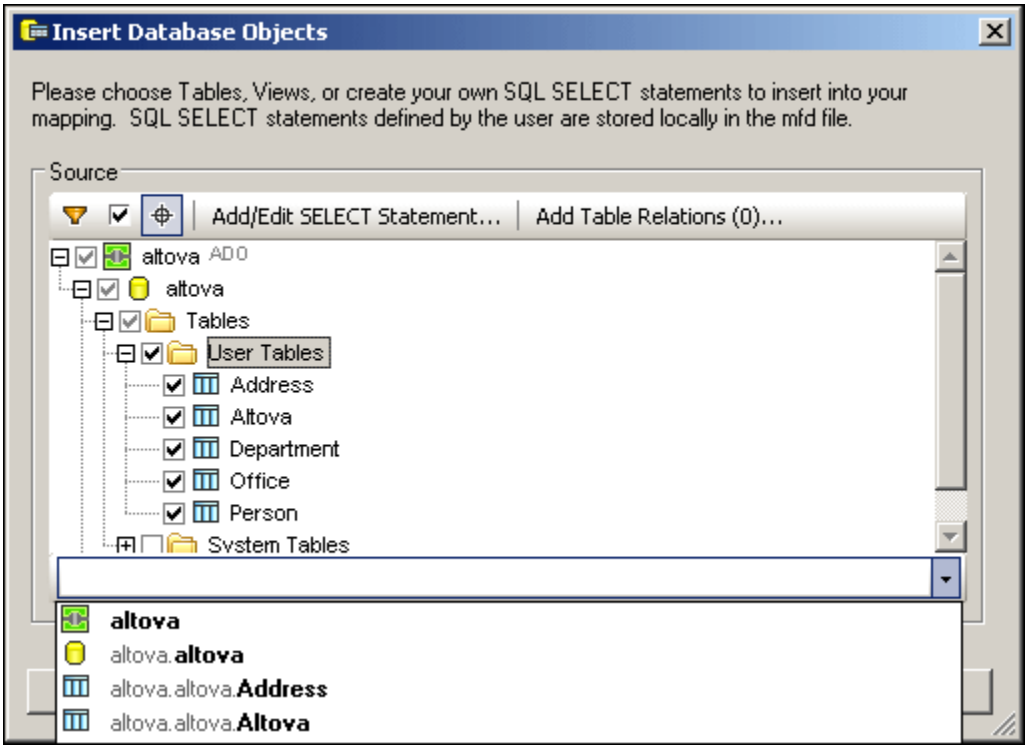
8.3.2 Inserting databases - table preview customization

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:

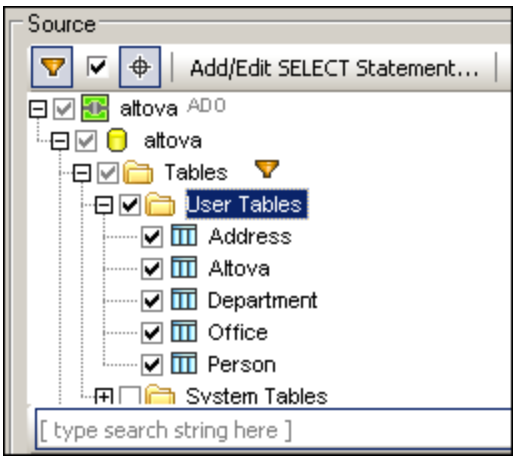
1. Click the **Object Locator** icon or press **Ctrl+L** to search for specific database items.
   A drop-down list containing all selectable items appears at the bottom of the window.



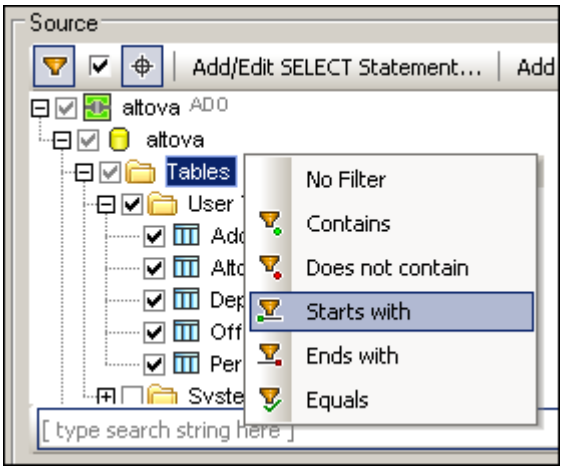
2. Enter the string you want to search for, or select the item from the drop-down list.

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.

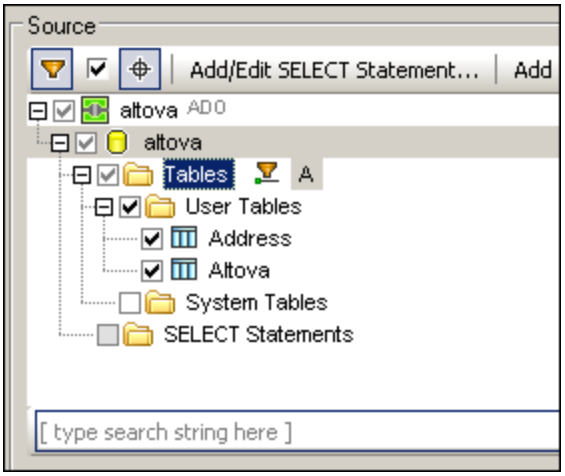


2. Click the filter icon next to the folder, and database objects, you want to filter. Select the filtering option from the popup menu that appears.



An empty field appears next to the filter icon.

3. Enter the characters you want to act as the filter e.g. A.

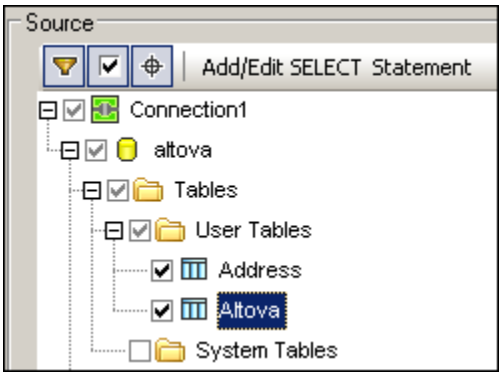


The results are automatically updated as you type.

Showing checked objects only

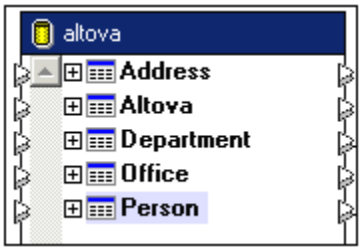
1. Click the **Show checked objects only** icon in the toolbar to have only those tables

displayed, where the check mark is present.



8.3.3 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.



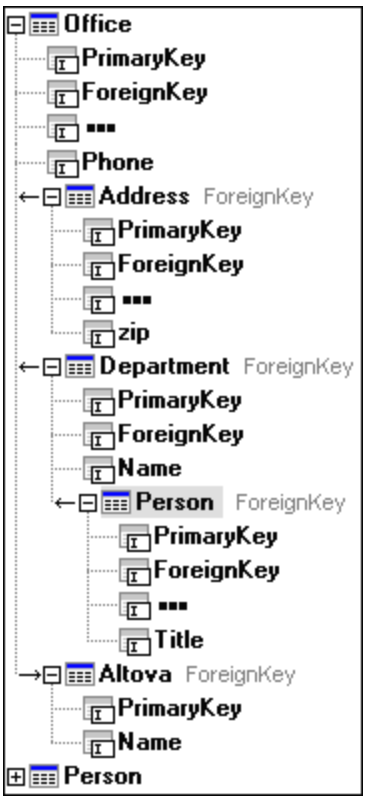
Let us call the table names visible in the above diagram "**root**" tables, i.e. they are the top level, or **root** of the tree view. Expanding a table name displays all the tables related to it. The "root" tables are usually displayed in alphabetical sort order; this has no bearing on the actual table 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.

8.3.4 Database action: Insert

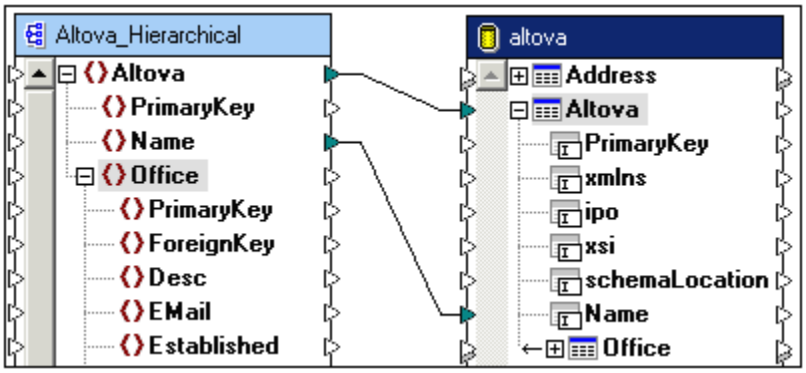
The first example in this section, deals with the simple task of **adding** a new office orgchart to the Altova table. The only fields available in the Altova table are: PrimaryKey and Name.

The second example **inserts** related office tables to the new orgchart record.

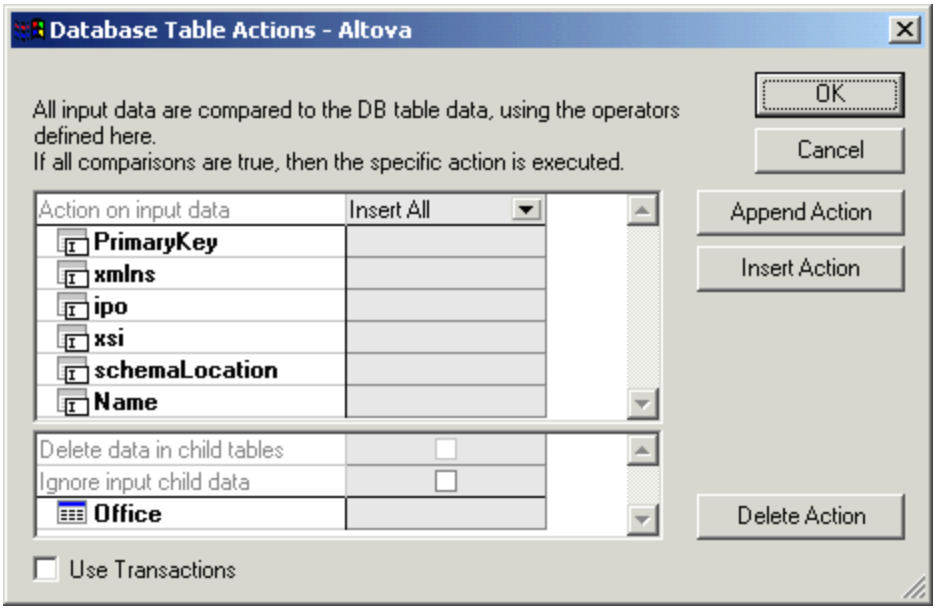
1. Insert the Altova_Hierarchical.xsd schema (and assign **altova-cmpy.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping.
3. Create the following mappings:

   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.

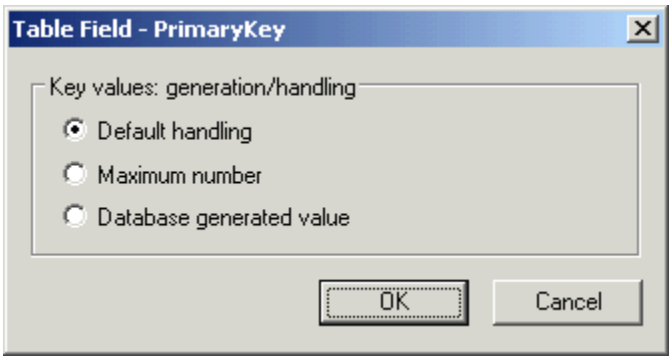


4. Right click the **Altova** entry and select the menu item "**Database Table Actions**". There is currently only one table action column defined in this dialog box, **Insert All**. (Update if... and Delete if... table actions are selected by clicking the column header combo box, whereas **additional** table actions, can be defined by clicking the Append, or Insert Action buttons.)

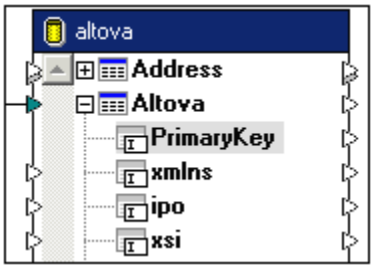


The table action, Insert All, inserts all **mapped fields** of the current table into the database. We now have to define the status of the new PrimaryKey field for this action.

5. Click OK to confirm the current settings.
6. Right click the **PrimaryKey** item, then select the **Database Key Settings** entry.



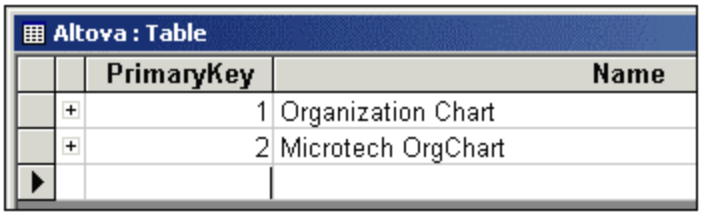
7. Select the **Maximum number** entry, and click OK to confirm.
   You will notice that the input icon for the PrimaryKey field is now unavailable.



8. Click the Output tab at the bottom of the mapping window to see the pseudo-SQL code 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:\Program Files\Altova\MapForce2007\MapForceExam
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FRON

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
```

9. 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.
10. Open the Altova database in Access to see the effect.



A new Microtech OrgChart record has been added to the Altova table with the new PrimaryKey 2. The data for this record originated in the input XML instance.

11. Switch back to MapForce.

    You will now see a record of what happened when the SQL script was processed.

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Program Files\Altova\MapForce2007\MapForceExamp
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM
-->>> OK. One or more rows.

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
-->>> OK. 1 row(s).
```

Please note:

You can only run SQL scripts once from the Output window, you have to switch back to the Mapping window, and to the Output window again, to re-run the script.

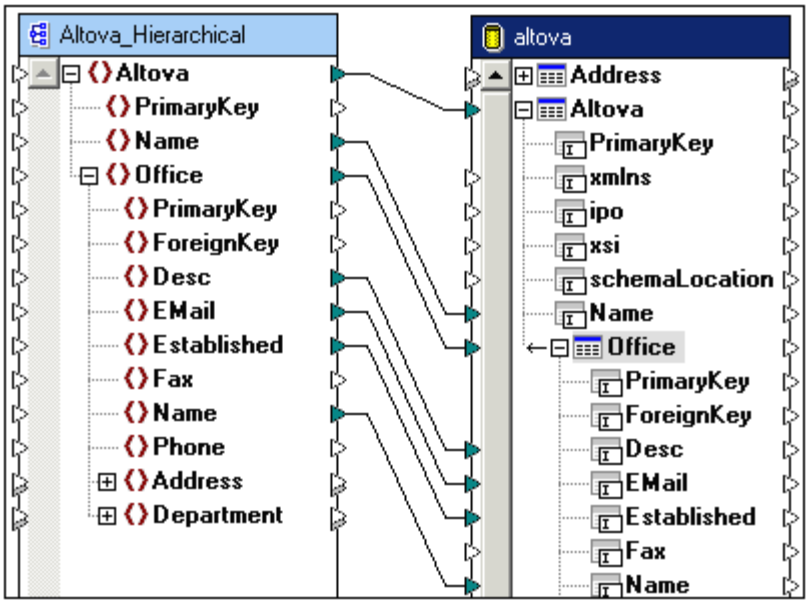
As stated in the comment section shown above, the SQL script is **pseudo-SQL** and cannot be executed in an SQL tool. You cannot infer anything about date formats, specific database syntax etc. from the pseudo code displayed here! It is meant to be an aid into what has, or will be executed.

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. Right click the **Office** entry and select the menu item "**Database Table Actions**". The Insert All... table action is selected by default, you do not have to make any changes here, click OK to confirm.
2. Right click the **Office | PrimaryKey** field and select the **Database Key Settings** entry.
3. Select the **Maximum number** entry and click OK to confirm.
4. Create the following mappings between the two components:
   Office to Office
   Desc to Desc, and
   Email to Email
   Established to Established, and
   Name to Name.



5. Click the Output tab to see the pseudo-SQL code.

```
 9  SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey]
10
11  INSERT INTO [Altova] ([Name],[PrimaryKey])
12      VALUES ('Microtech OrgChart',%PrimaryKey%)
13
14  SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey]
15
16  INSERT INTO [Office] ([ForeignKey],[Desc],[EMail],[Established],[Name],[PrimaryKey])
17      VALUES ('%PrimaryKey%','Microtechnology products are currently the bleeding edge of com
18
19  SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey]
20
21  INSERT INTO [Office] ([ForeignKey],[Desc],[EMail],[Established],[Name],[PrimaryKey])
22      VALUES ('%PrimaryKey%','Microtech established its new office on Feb 30','nextoffice@micr
```

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.

Altova : Table

	PrimaryKey	Name
	1	Organization Chart

	PrimaryKey	Desc	EMail	Established
	1	The company was es	office@nanonul	1992-04-01
	2	On March 1st, 2000,	nextoffice@nan	2001-03-01

	PrimaryKey	Name
	2	Microtech OrgChart

	PrimaryKey	Desc	EMail	Established
	3	Microtechnology proc	office@microte	1992-04-01
	4	Microtech establishe	nextoffice@mic	2001-03-01

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.

```
10  SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM
11  -->>> OK. One or more rows.
12
13  INSERT INTO [Altova] ([Name],[PrimaryKey])
14      VALUES ('Microtech OrgChart',%PrimaryKey%)
15  -->>> OK. 1 row(s).
16
17  SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM [
18  -->>> OK. One or more rows.
19
20  INSERT INTO [Office] ([ForeignKey],[Desc],[EMail],[Established],[Name],[PrimaryKey])
21      VALUES (2,'Microtechnology products are currently the bleeding edge of computer technology.','offi
22  -->>> OK. 1 row(s).
23
24  SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM [
25  -->>> OK. One or more rows.
26
27  INSERT INTO [Office] ([ForeignKey],[Desc],[EMail],[Established],[Name],[PrimaryKey])
28      VALUES (2,'Microtech established its new office on Feb 30','nextoffice@microtech.com','2001-03-0
29  -->>> OK. 1 row(s).
```

8.3.5 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.

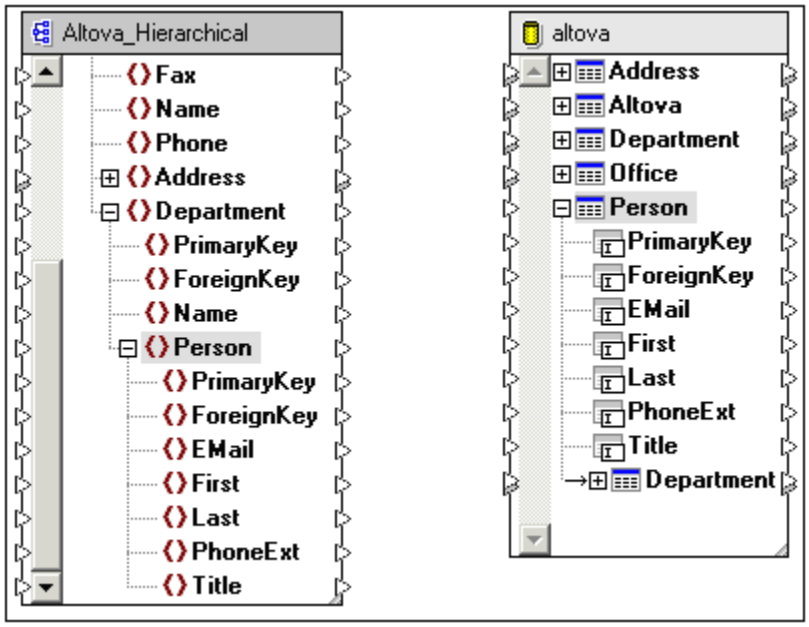
Files used in this example:

- Altova_Hierarchical.xsd
- altova-cmpy.xml
- altova.mdb

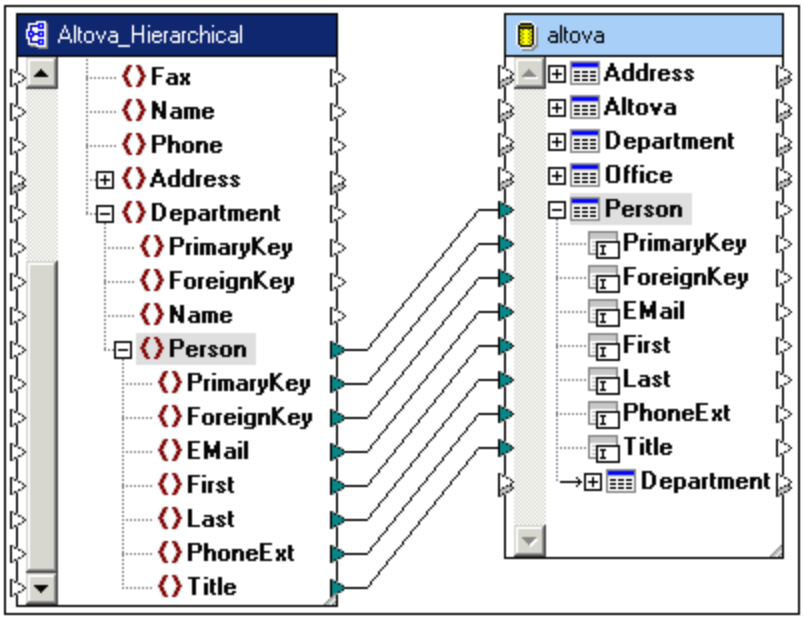
Aim:

To **update** the person fields of the Person table.

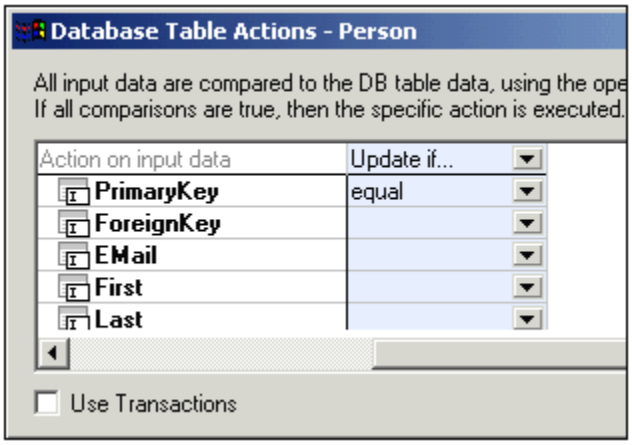
1. Insert the Altova_Hierarchical schema (and assign **altova-cmpy.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping.



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. Right click the **Person** entry and select the menu item "**Database Table Actions**".
6. Click the Table action combo box, 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.

8. Click the Output tab at the bottom of the mapping window to see the pseudo-SQL code that this mapping produces.
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@microt	Ingrid	Idilko	951
10	3	j.judy@microte	June	Judy	753
11	3	k.krove@microt	Karl	Krove	334

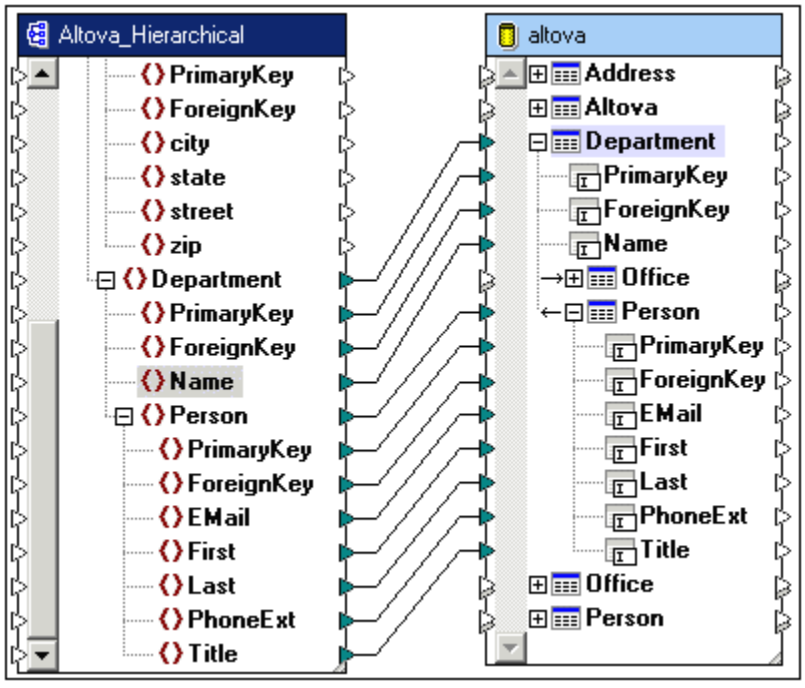
Second Example:
This slightly more complex example, attempts to update records in both the Department and Person tables, as well as add any new Person records which might exist in the XML input file. The "**root**" table used in this example is thus the **Department** table.

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.

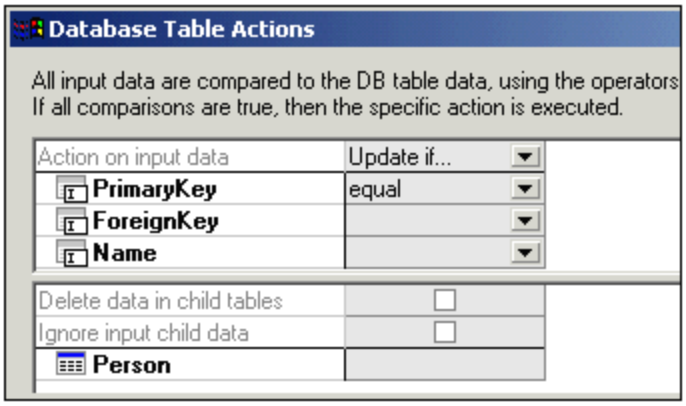
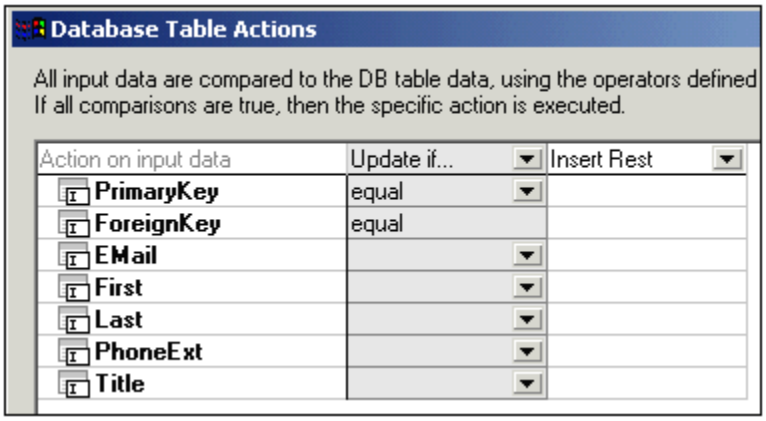


Table action: **Person** table

- Table action **Update if... "equal"** defined for PrimaryKey.



- Table action **Insert Rest** defined as the second table action should the first comparison, Update if..., fail.
  Click the **Append Action** button to append a new Table action column.


```
 9    UPDATE [Department]
10      SET [ForeignKey]=1,[Name]='Admin' WHERE ([PrimaryKey]=1)
11
12    SELECT [ForeignKey],[PrimaryKey] FROM [Department] WHERE ([PrimaryKey]=1)
13
14    UPDATE [Person]
15      SET [EMail]='A.Aldrich@microtech.com',[First]='Albert',[Last]='Aldrich',[PhoneExt]=582,[
16
17    UPDATE [Person]
18      SET [EMail]='b.bander@microtech.com',[First]='Bert',[Last]='Bander',[PhoneExt]=471,[Tit
19
20    UPDATE [Person]
21      SET [EMail]='c.clovis@microtech.com',[First]='Clive',[Last]='Clovis',[PhoneExt]=963,[Title]
```

Please note:
As stated in the comment section, the SQL script is **pseudo-SQL** and cannot be executed in an SQL tool. You cannot infer anything about date formats, specific database syntax etc. from the pseudo code displayed here! It is meant to be an aid into what has, or will be executed.

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.

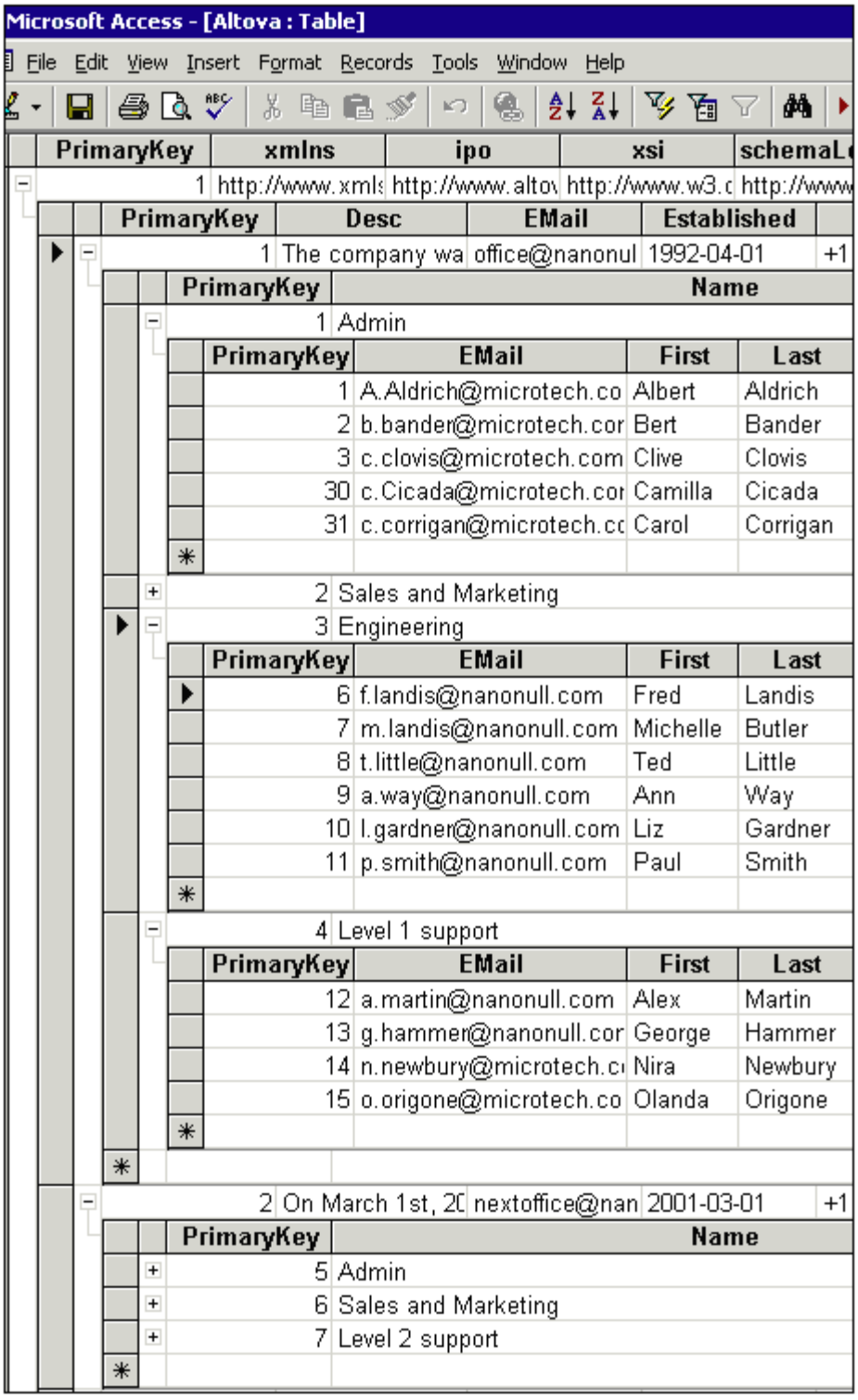
Processing sequence Person table:

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.

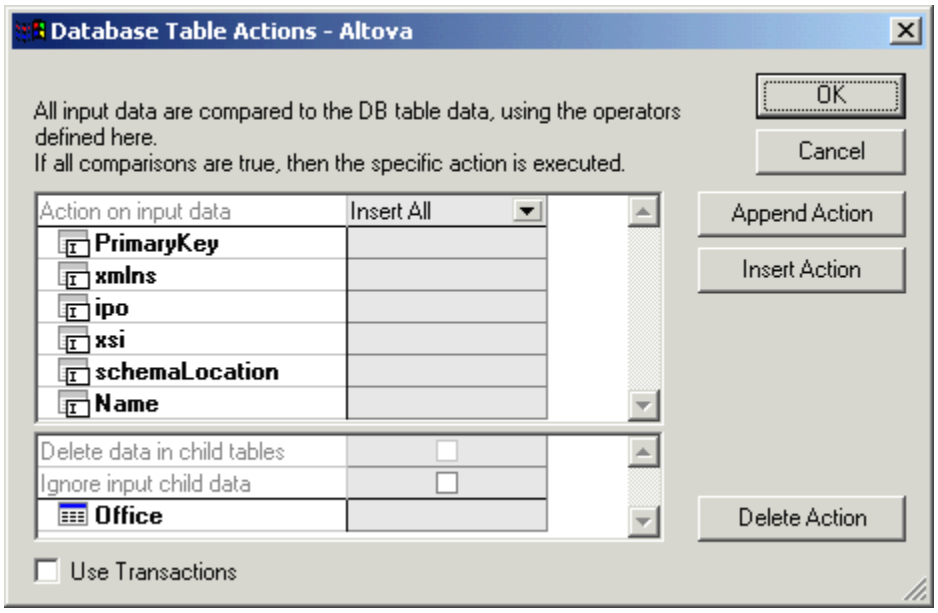
Person table: **Update if**... condition **false**:
source and target keys are not identical, i.e. source keys exist which have no match in the target database, the update if... condition fails, therefore:

- 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:

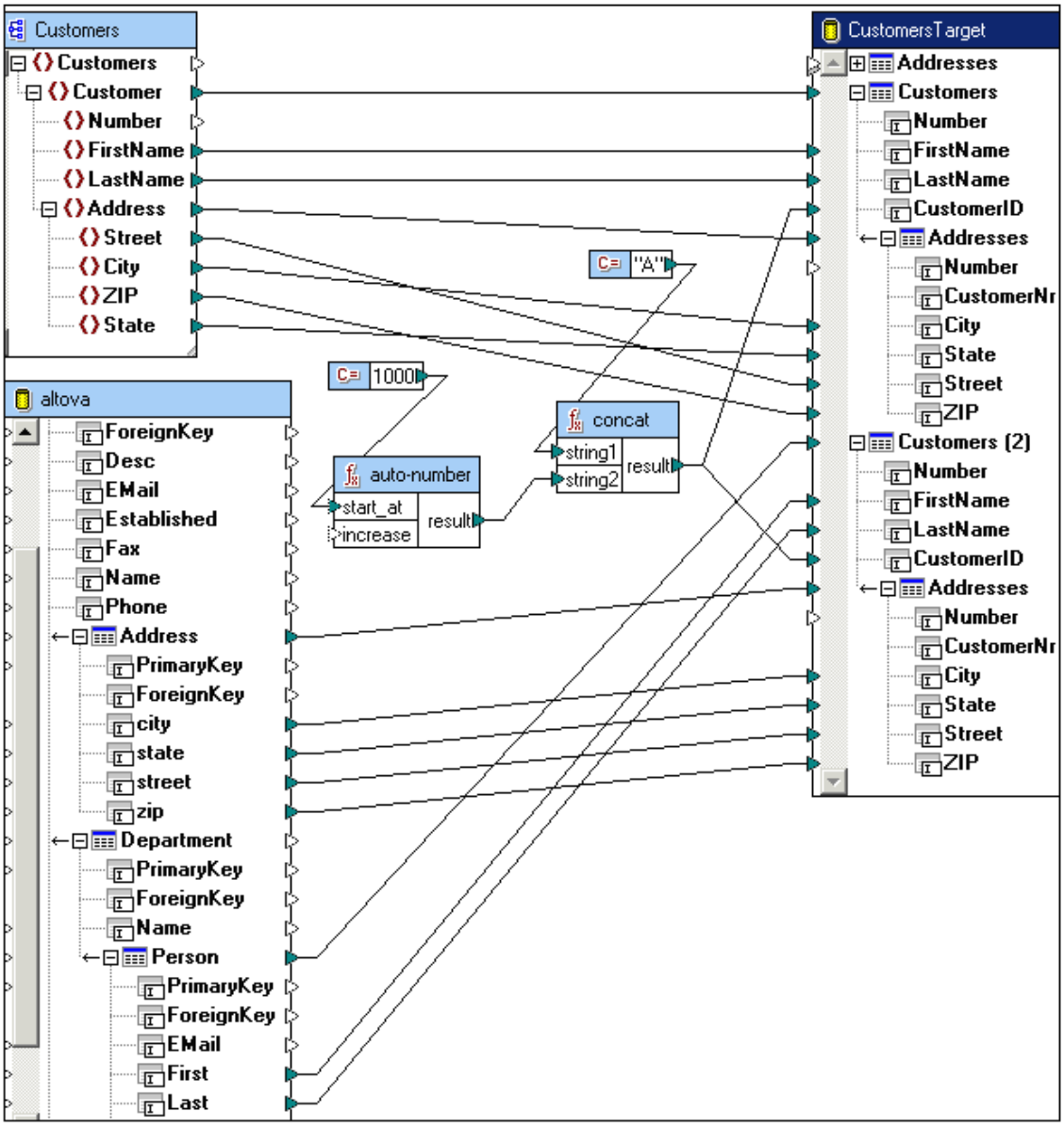
- Updates parent table data (Department records).
  Retains existing database tables (Engineering in this case) which do not have a counterpart in the input XML file (no source keys for the Update if... comparison).
- Delete if... only deletes those Person records for which a corresponding **Person** 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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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 an 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.



8.3.6 Database action: Delete

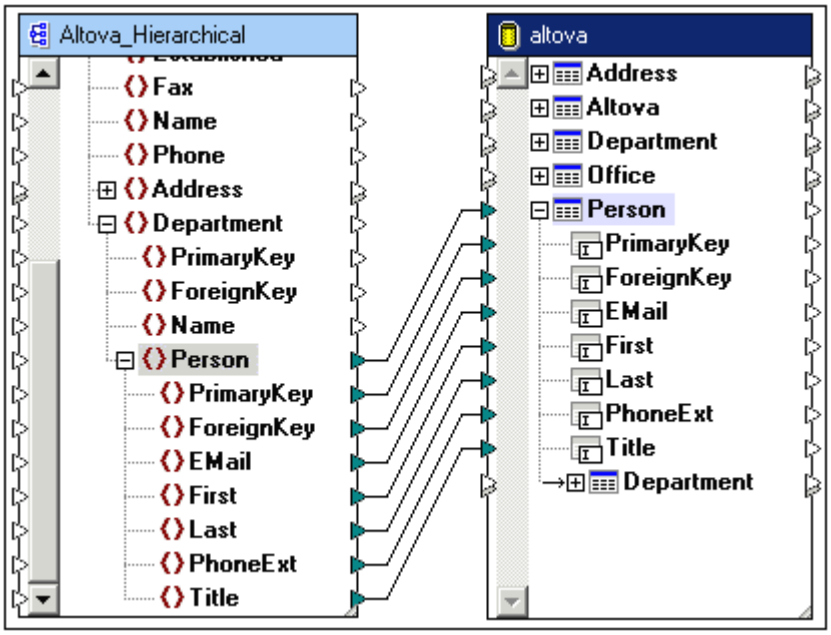
The table action Delete if... is used to selectively delete data from tables. This is achieved by selecting specific items/fields of the source and target components which are to be compared. The specific table action is then executed depending on the outcome of this comparison.

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.

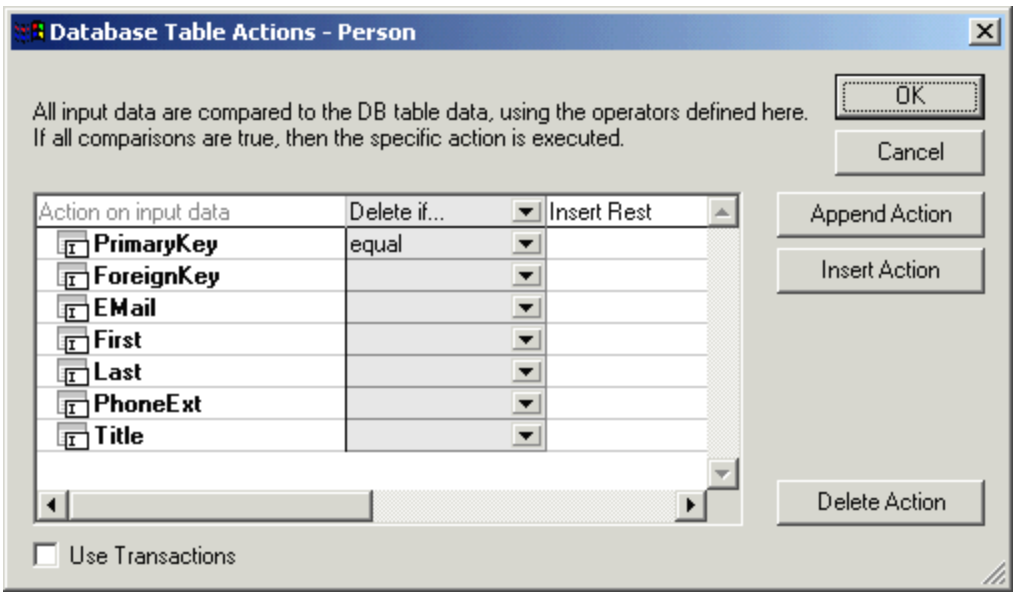
Aim:

- 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. Right click the **Person** entry and select the menu item "**Database Table Actions**".
6. Click the Table action 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.

8. Click the Output tab at the bottom of the mapping window to see the pseudo-SQL code that this mapping produces.
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.

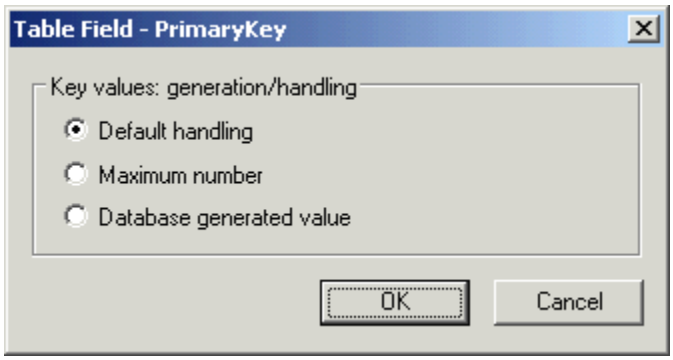
Two additional examples of the Delete if... table action can be viewed in the Update if... combinations section.

8.3.7 Database Key settings

When mapping to databases, MapForce lets you specify how the primary key will be handled. The three options below, are only available if you right click a key field, and select the menu option Database Key settings.

The primary key setting should take the table action defined for that table into account. E.g. when inserting records, the primary key setting should generally be "Maximum number", so that new records are automatically appended to existing ones.

An input icon is only available when "Default handling" is selected. This allows source data to be mapped to the database field directly.



Default handling
This is the standard setting for all database fields.

- an input icon exists when this option is selected, allowing you to map data directly
- the value supplied by the source item, is used as the key value in the database

Maximum number
Use this setting when you want to **Insert** records into the database.

- an input icon is not available, when you select this option.
- the **select** statement **queries** the database for the maximum value of the primary key. This value is then incremented by one and **inserted** into the new field.

Database generated value
Use this setting when the database generates/uses the **Identity function** to generate key values, and you want to **Insert** records.

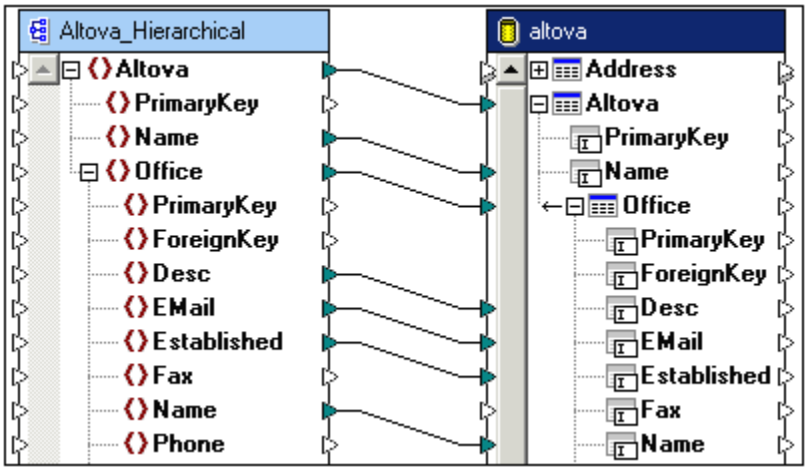
- an input icon is not available, when you select this option.
- the select statement **inserts** the mapped data into the database, **queries** the database for the key value generated by the identity function, and writes it into the key field.

8.3.8 Database Table Actions and transaction processing

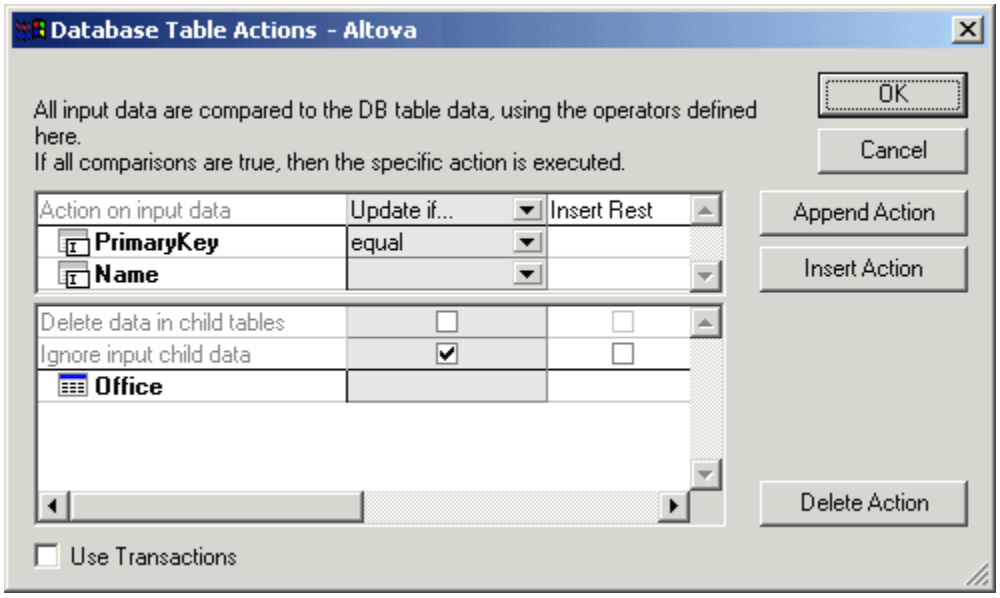
Table actions allow you to define how specific table data are to be manipulated. MapForce supports the table actions: insert, update and delete. One or more fields are used to compare source and target data to determine if the table action is to be executed.

The Table Action dialog box allows you to define the:

- fields that will be compared (e.g. PrimaryKey)
- operators used for the comparison (equal, equal ignore case), and
- action taken, when all conditions of each column are fulfilled.



Data may originate from any data source: XML file, EDI message, database, text, Constant component etc. The mappings that define which data are to be manipulated, are created using connectors in the Mapping window.



- Table Actions are processed from left to right. In the example above, the **Update if...** column is processed and then the **Insert Rest...** column.
- **All** the 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.

- If a condition is not satisfied, then the table action for that column is ignored, and the next column is processed.
- If none of the conditions are "true", no table action takes place.

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.

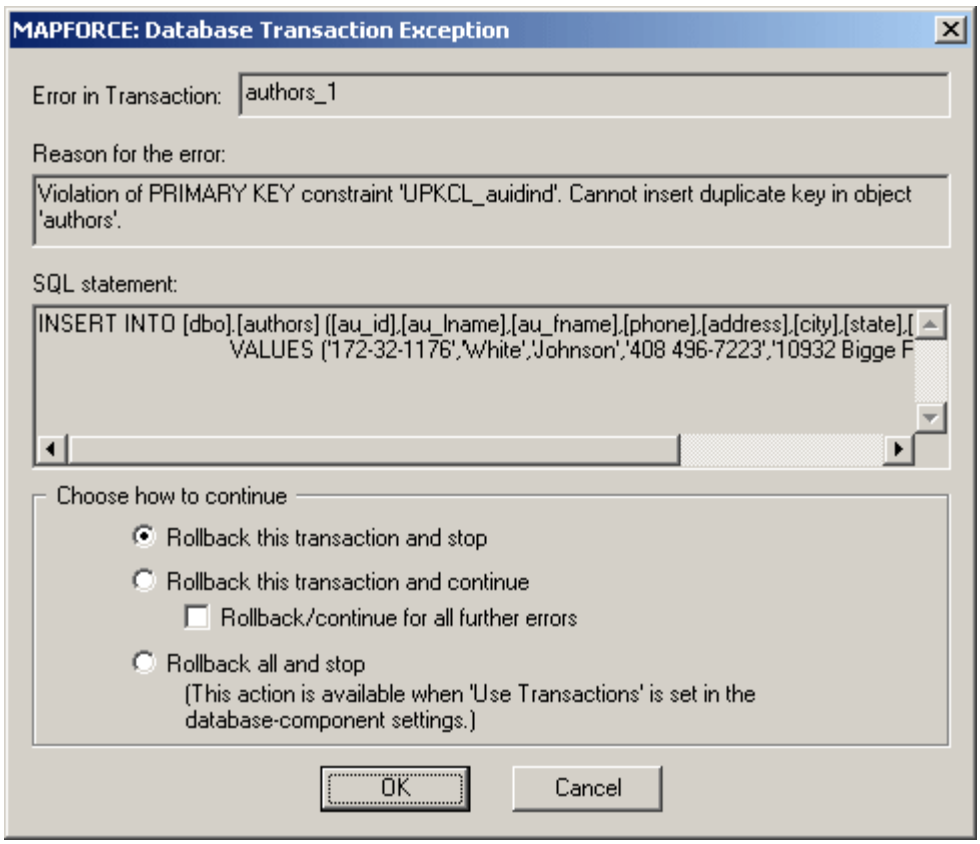
Ignore input child data:
Use this option when you want to update specific table data, without affecting any of the child tables/records of that table.

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.

8.3.9 Generating output values

The Java, C++, and C# libraries have been extended by two functions which can generate values for database fields, which do not have any input data from the Schema, database or EDI source component.

Auto-number and create-guid can both generate values for fields. Both functions are located in the **generator functions** subset of the **lang** library.

auto-number
is generally used to generate primary key values for a numeric field.

create-guid
Creates a globally-unique identifier (as a hex-encoded string) for the specific field.

8.4 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

8.4.1 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.

8.4.2 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	

8.4.3 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	

8.4.4 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	

8.4.5 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.

8.4.6 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()	

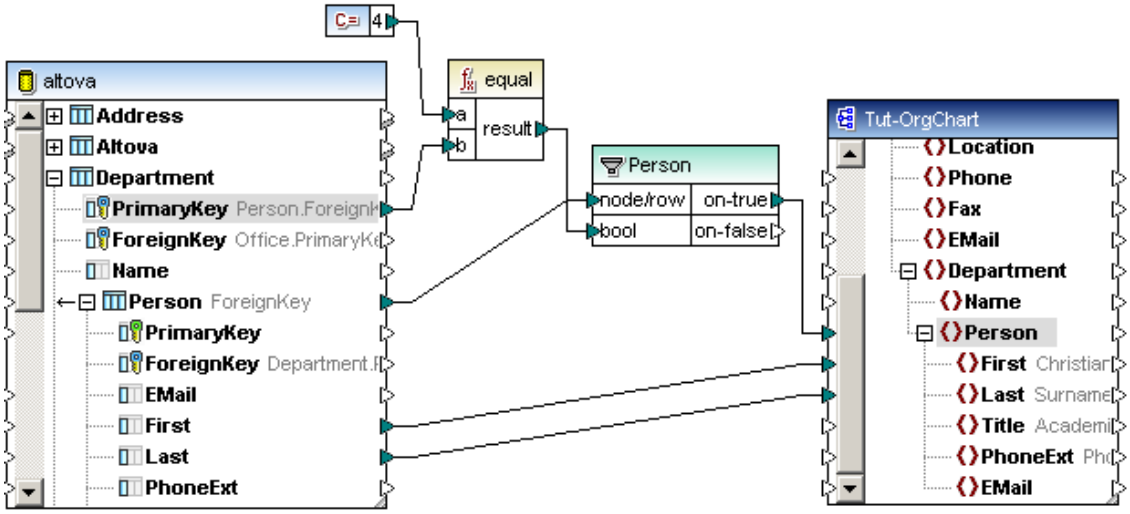
SQL-Execution:			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	y	
	retrieve parameter types	n	
	special issues when using ?	n	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	n	not 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	

8.5 Database relationships and how to preserve or discard them

Maintaining database relationships

To map all First/Last names from the Person table, where the Department primary key is equal to 4, create mappings under the **same** "root" table, Department:

- **PrimaryKey** is mapped from the Department "root" table.
- **First/Last** are mapped from the Person table, **which is a child** of the Department "root" table.



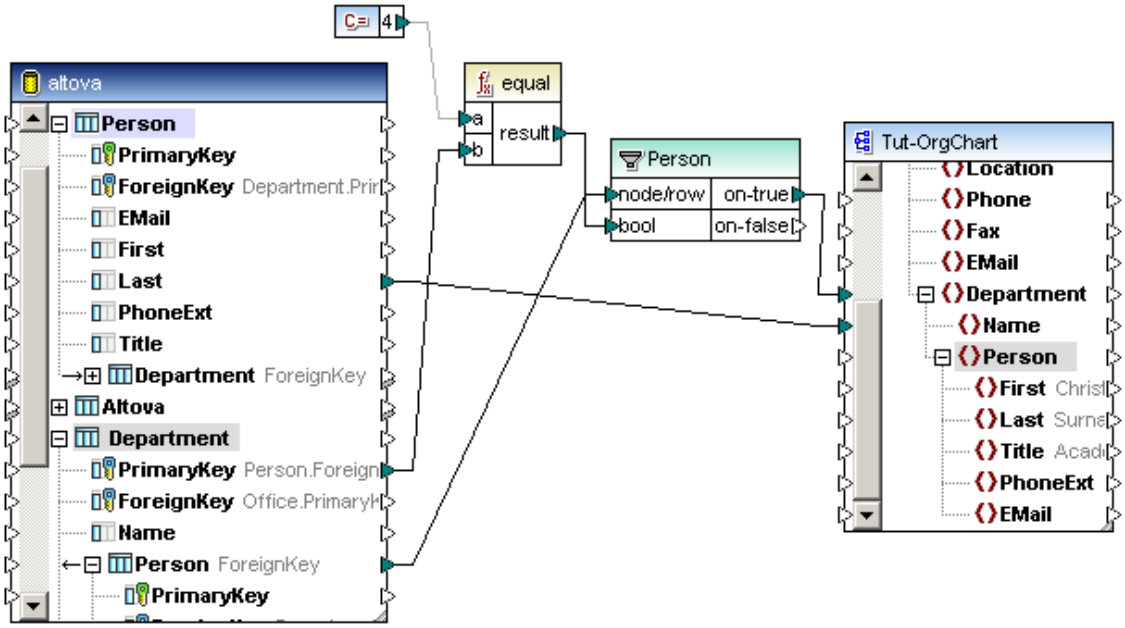
Result of the above mapping:
The last names of the persons in the Department with the primary key of 4, (IT & Technical support), are displayed in the Output tab.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <OrgChart xsi:noNamespaceSchemaLocation="C:/DOCUME~1/
3    <Office>
4      <Department>
5        <Person>
6          <First>Alex</First>
7          <Last>Martin</Last>
8        </Person>
9        <Person>
10         <First>George</First>
11         <Last>Hammer</Last>
12       </Person>
```

Discarding database relationships

Create mappings of the same fields from **different** root tables (e.g. Department and Office)

- **PrimaryKey** is mapped from the Department "root" table.
- **Last** is mapped from the Person "root" table.



Result of the above mapping:
This mapping method does not deliver the same result, as the table dependencies between the Department and Person tables are now not taken into account.

The result contains the last names of all 21 persons in the database, the **filtering** by the Department primary key has clearly not succeeded.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     <Office>
4       <Name>Callaby</Name>
5       <Name>Further</Name>
6       <Name>Matise</Name>
7       <Name>Firstbread</Name>
          ...
24      <Name>Redgreen</Name>
25    </Office>
26  </OrgChart>
27
```

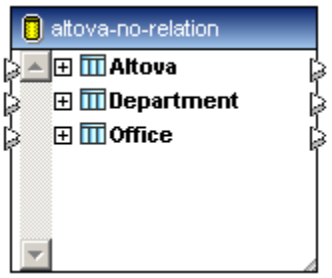
8.6 Local Relations - creating database relationships

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.

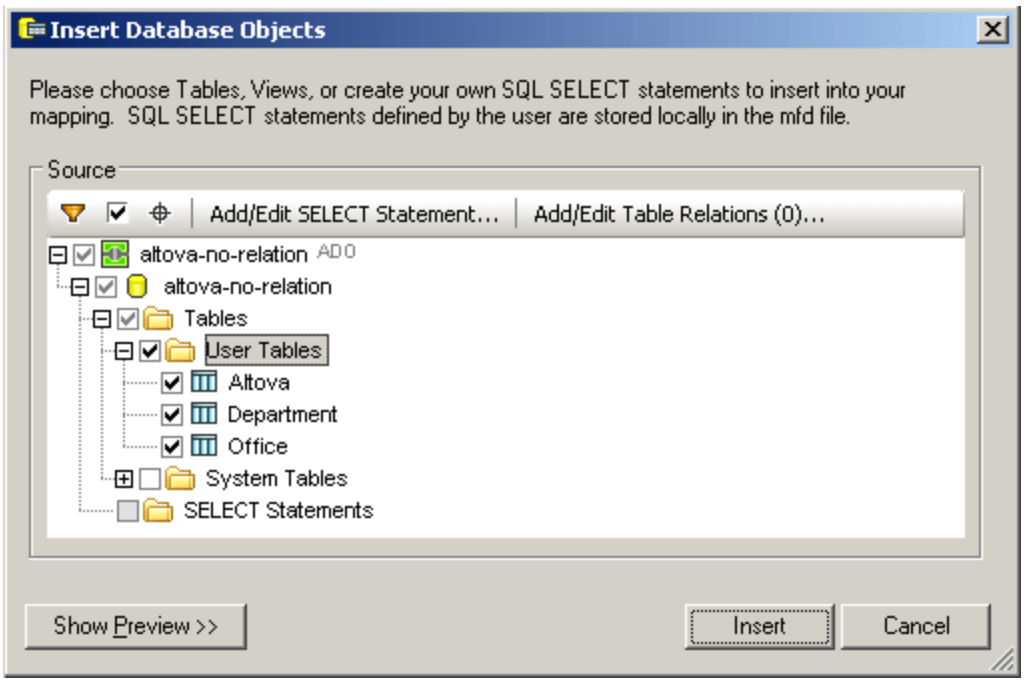


The aim of this example is to display the offices of Altova and show the departments in each.

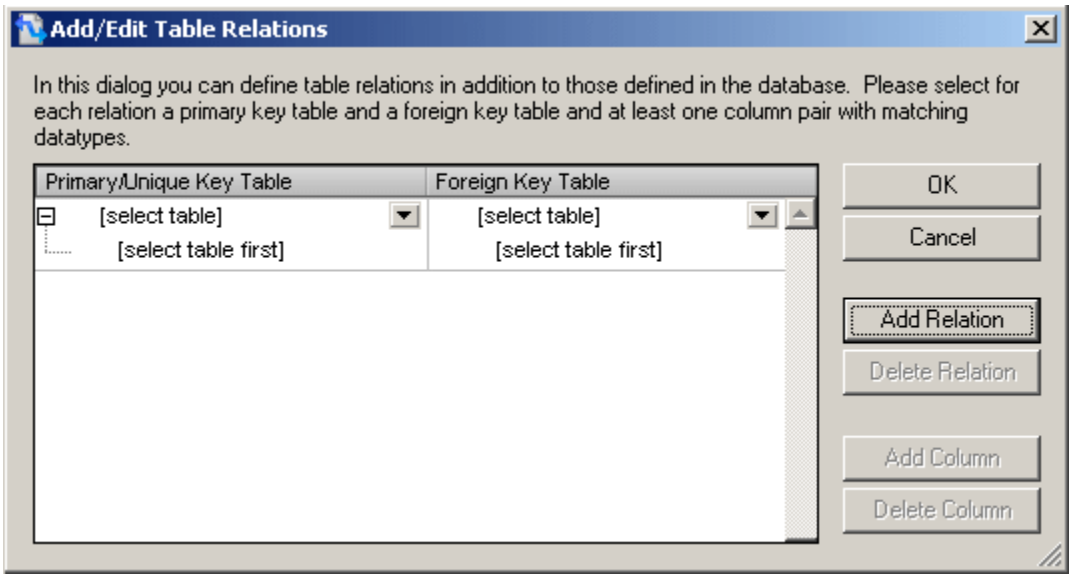
None of the tables visible in the **altova-no-relation** tree have any child tables, all tables are on the same "**root**" level. The content of each table is limited to the fields it contains. We can however, use MapForce to extract related database data, even though relationships have not been explicitly 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.

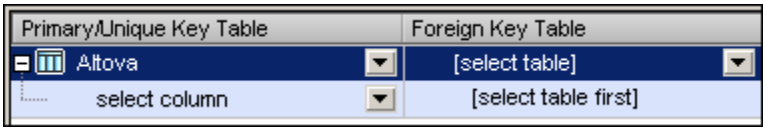


4. Click the **Add/Edit Table Relations** button in the icon bar.
   This opens the Add/Edit Table Relations dialog box.
5. Click the **Add Relation** button.

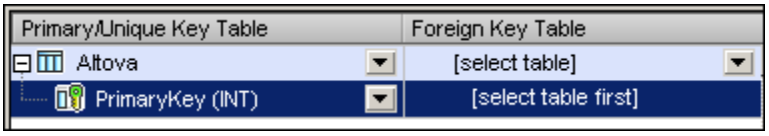


The two combo boxes allow you to select the tables you want to create relations for. The left combo box is the Primary/Unique Key Table, the right one, the Foreign Key Table.

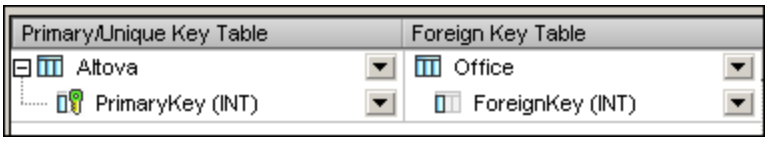
6. Click the left combo box and select the **Altova** table.



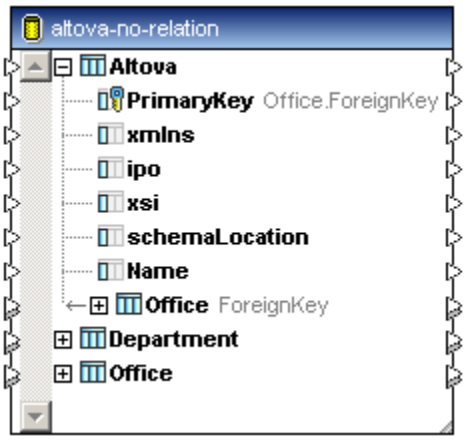
7. Click the "select column" combo box below it, and select the **Primary Key** entry.



8. Select the **Office** table and **ForeignKey** column for the Foreign Key table.



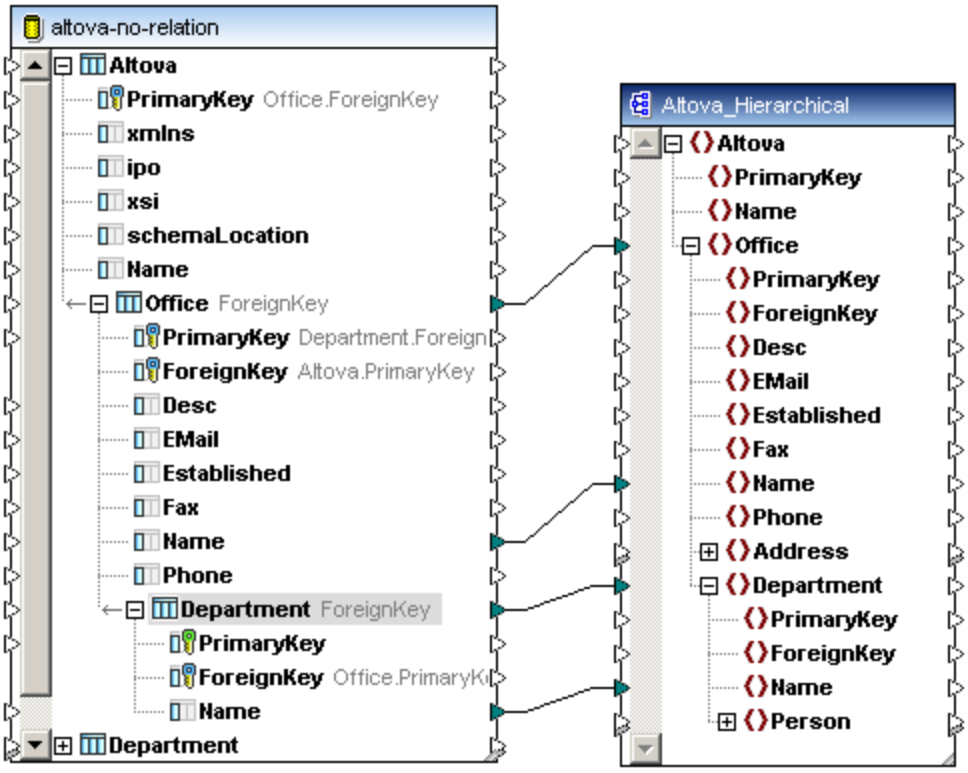
9. Click the OK button to complete the local relation defintion, then click the **Insert** button to insert the database into the mapping area.



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/alp/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>
```

8.7 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.

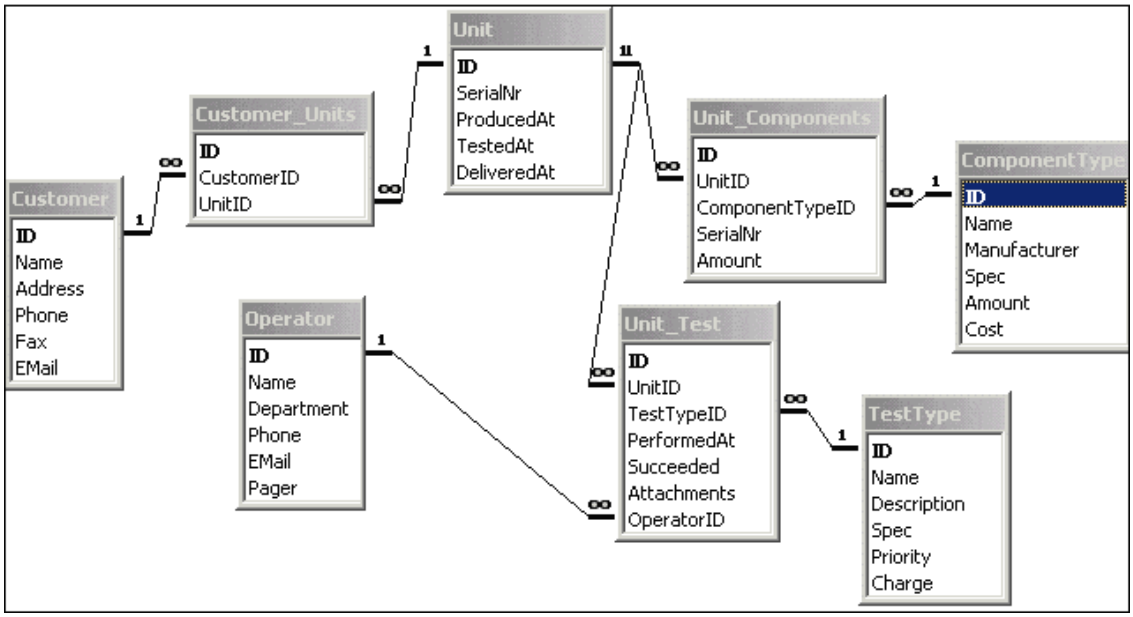
This may lead to inconveniently large tree structures if all tables are selected in a single database component. It is however possible to create multiple database components, of the same database, which only use/import those tables that are needed for the mapping process. This method also makes for 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:

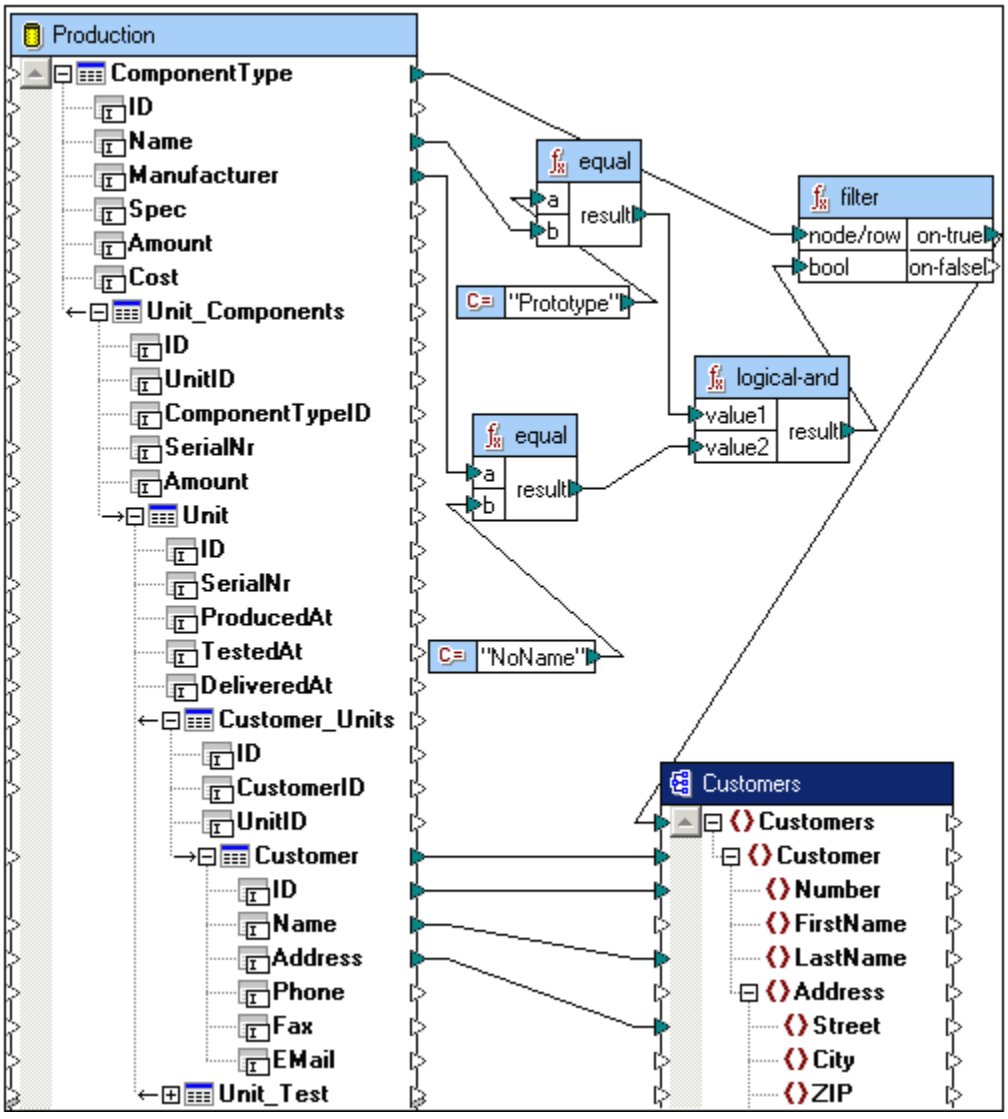


8.7.1 Complete database import

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



8.7.2 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:

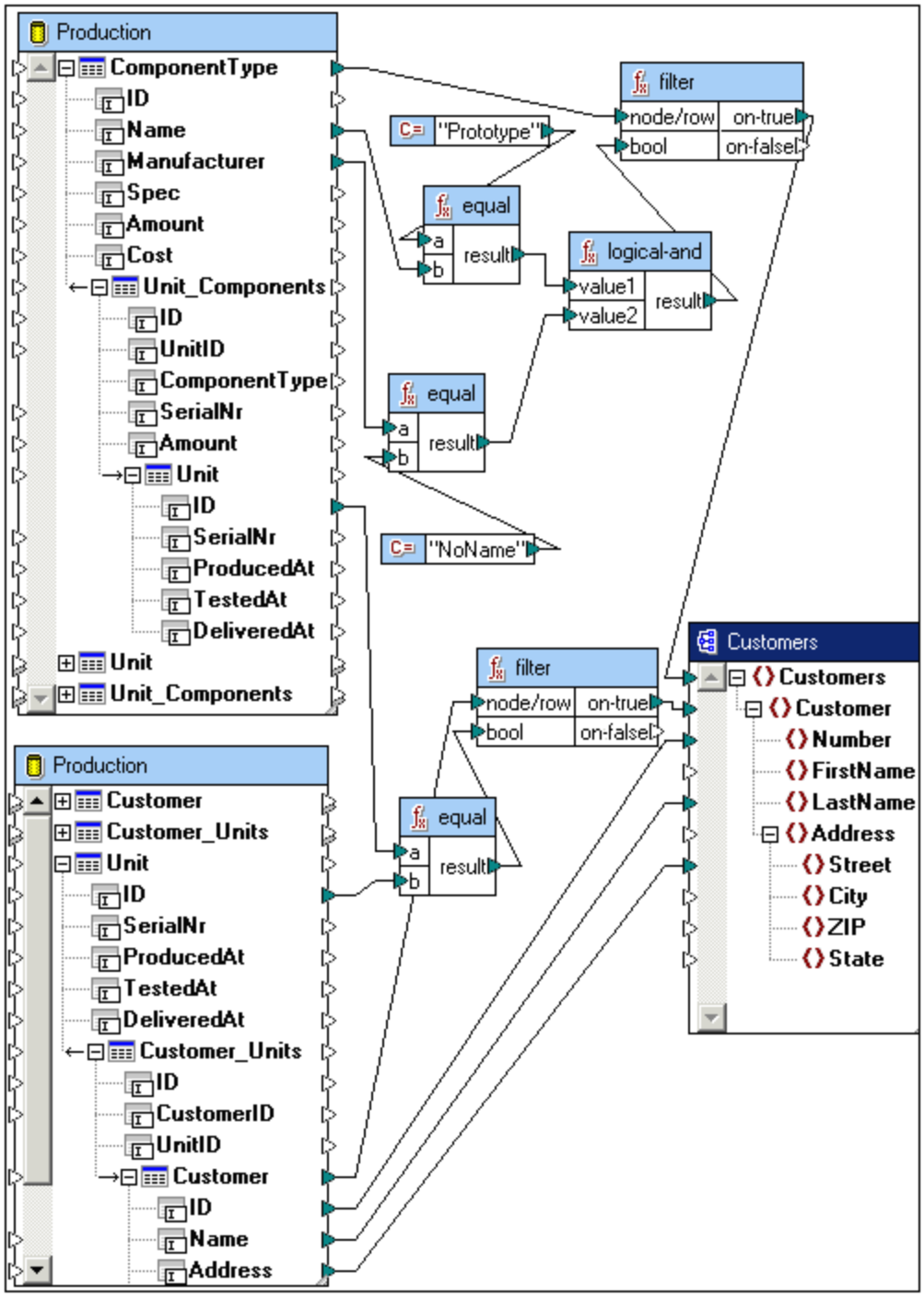
- ComponentType
- Unit_Components
- Unit

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

8.8 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:

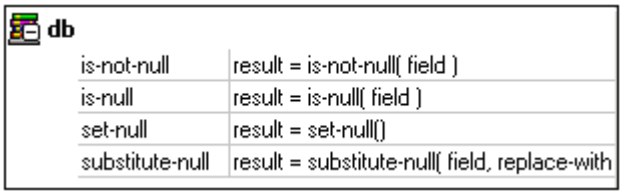
MapForce function	Database function
"floor"	"FLOOR()"
"ceiling"	"CEILING()"
"round"	"ROUND()"
"concat"	"+"
"substring"	"SUBSTRING()"
"contains"	"CHARINDEX()"
"string-length"	"LEN()"
"uppercase"	"UPPER()"
"lowercase"	"LOWER()"
"find-substring"	"CHARINDEX()"
"empty"	"IsEmpty()"

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()"

8.9 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.

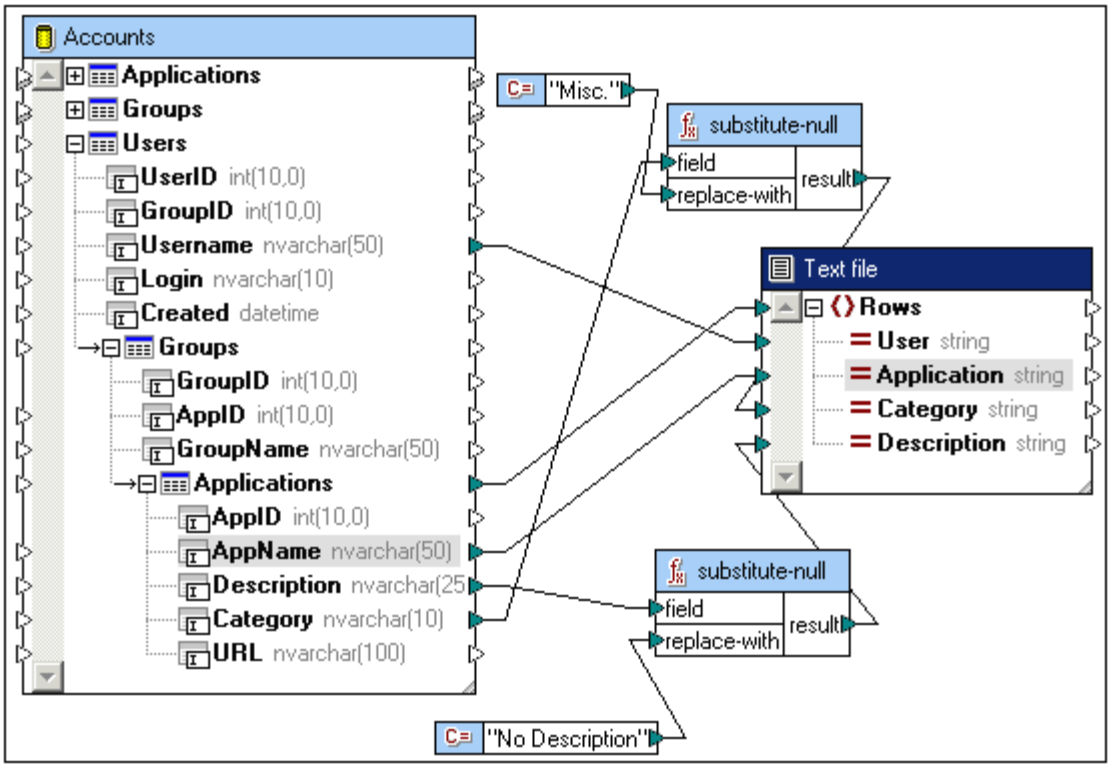
set-null
Used to set a database column, or text field to null. This function will also overwrite a default value with null.

Please note:

- Connecting this function to another function will generally not lead to a null result! (The null input will be cast to "", 0, or "false".)
- Connecting to special functions, Filters and IF-Else conditions works as expected, fields are set to null.
- Using set-null as an input for a simpleType element will not create that element in the target component.
- Connecting this function to a complexType element, as well as a table, or row is not allowed. A validation 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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder.



The first function checks if a Category entry exists in the Applications table. As one does not exist for the Notepad application, "Misc" is mapped to the Category item of the Text file.

The second function checks if a Description entry exist, and maps the string "No description" if one does not exist, which is also the case with the Notepad application.

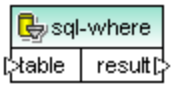
8.10 SQL WHERE Component / condition

MapForce allows you to filter database data conditionally using the SQL WHERE component. The SQL WHERE component is comprised of two parts:

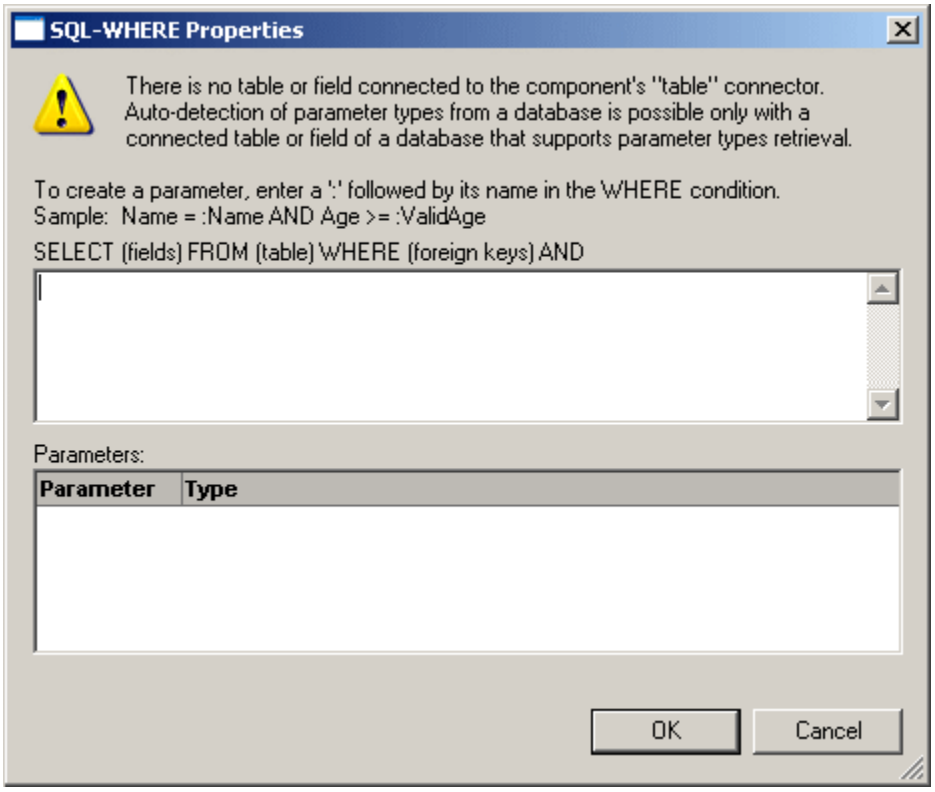
- The **Select** statement that is automatically generated when you connect to a database table
- The **WHERE** clause that you manually enter in the SQL WHERE Select text box. Note that the foreign keys are automatically included in the select statement.

To insert an SQL WHERE component:

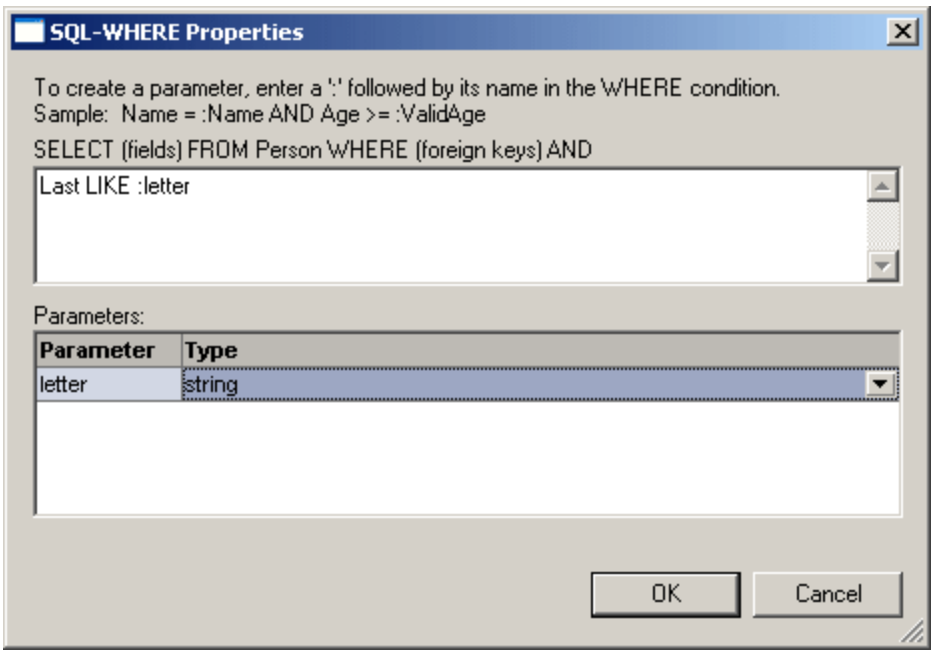
1. Select the SQL WHERE icon  in the icon bar to insert it.



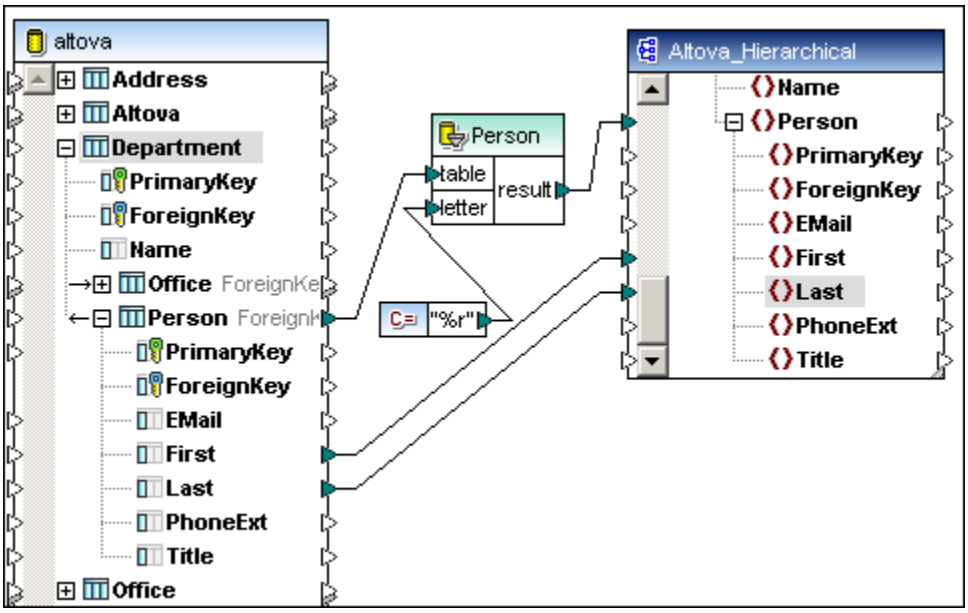
2. Make a connection between the source table that you want to query and the **table** item of the SQL WHERE component.
3. Double click the component to write the WHERE query in the top text box. The lower pane allows you to define the datatype of a parameter defined in the query (a warning message is displayed in the dialog box while the component is not connected to a database).



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** input icon of the SQL WHERE component.



The WHERE statement shown above, defines a parameter called "letter" of type string, 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, **%r**, which results in all persons being found whose last name ends in a "r".



8.10.1 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.

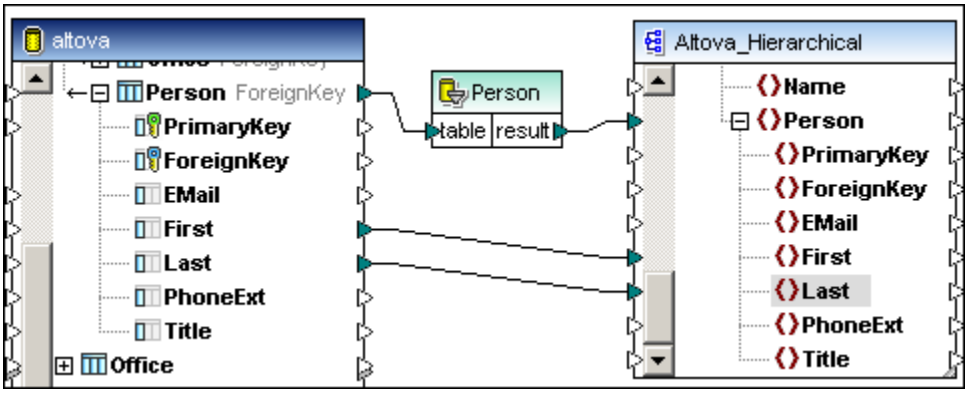
E.g. **xmlexists**('$c/Client/Address[zip>"55116"]' passing USER.CLIENTS. CONTACTINFO AS "c")

Select from Person WHERE
First > "C" AND Last > "C"

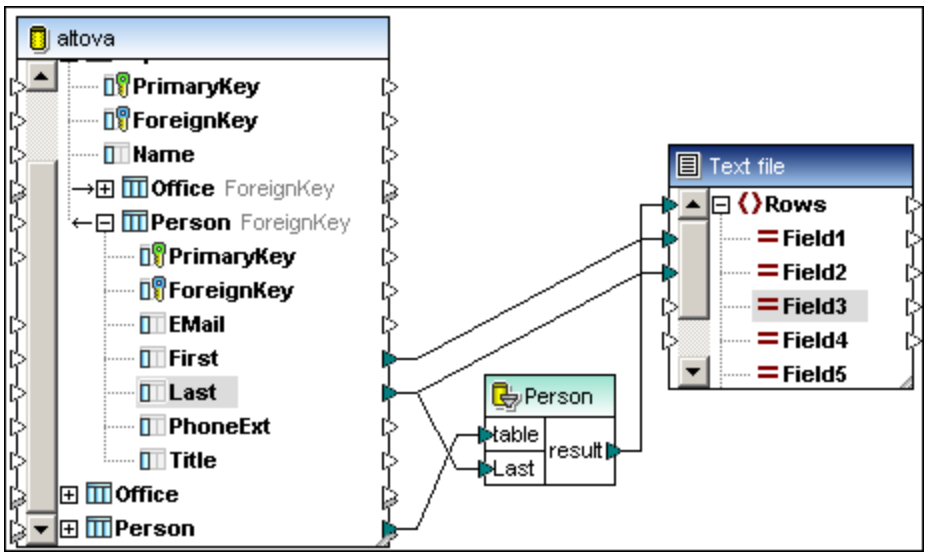
Retrieves those records where the contents of First and Last are greater than the letter C. Retrieves all names from Callaby onwards.

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.



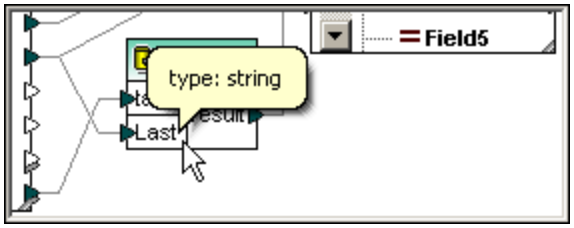
Select from Person WHERE
Last =:Last and Last > "L"
This WHERE statement creates a parameter **Last** which then appears as a parameter/item in the SQL WHERE component.



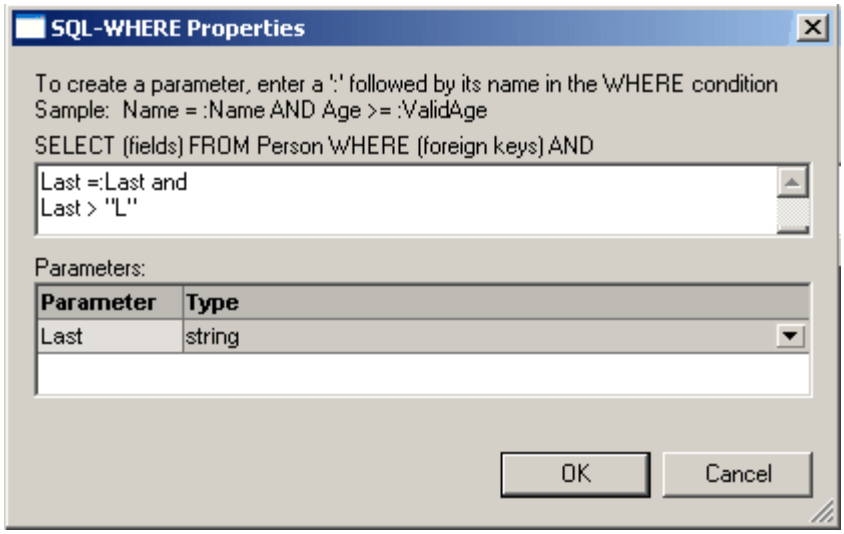
As soon as **:Last** is entered in the dialog box, the Parameter name appears in the lower area and the combo box can be used to select the datatype e.g. string. The default setting, **auto-detect by database**, is available if the database supports parameter type auto-detection.

Parameter	Type
Last	(auto-detect by database)

The actual parameter type can be seen, once the dialog box has been closed, if you place the mouse cursor over the parameter item in the SQL WHERE component.

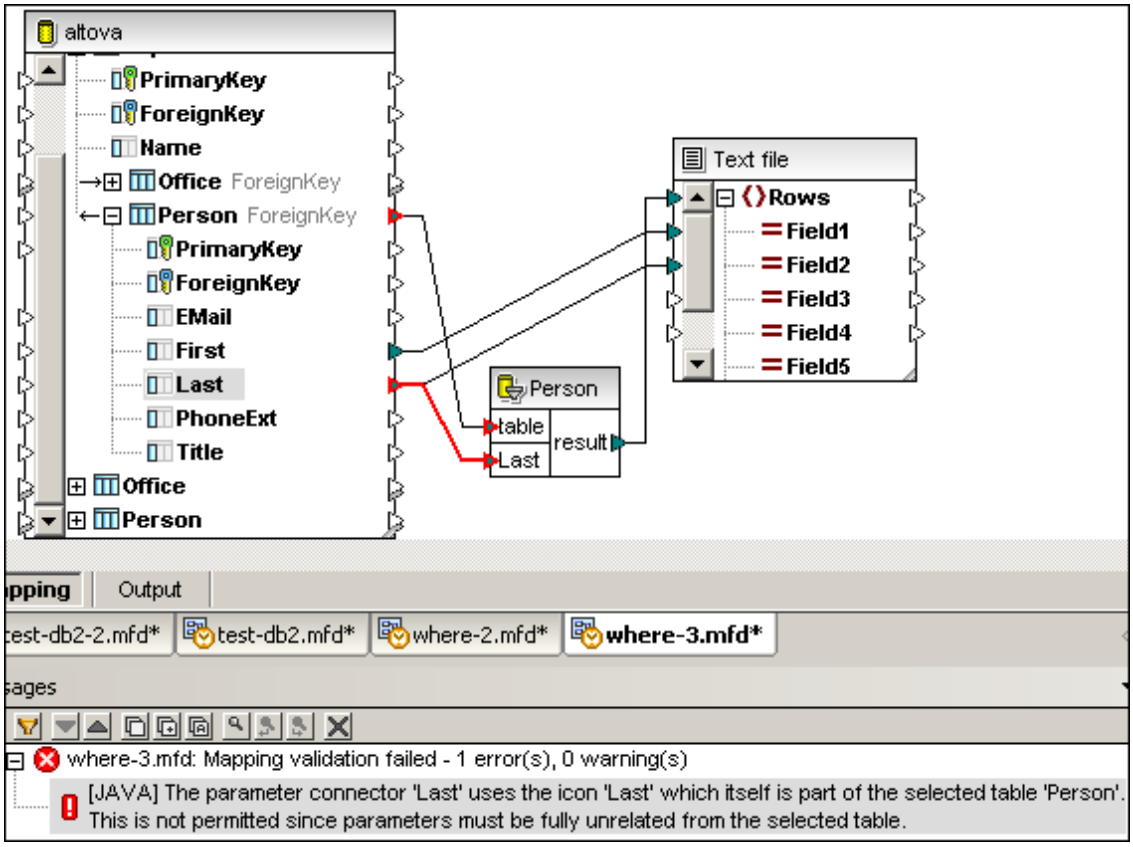


Note that you can create a **line-break** in the SELECT field using the CTRL+M shortcut.

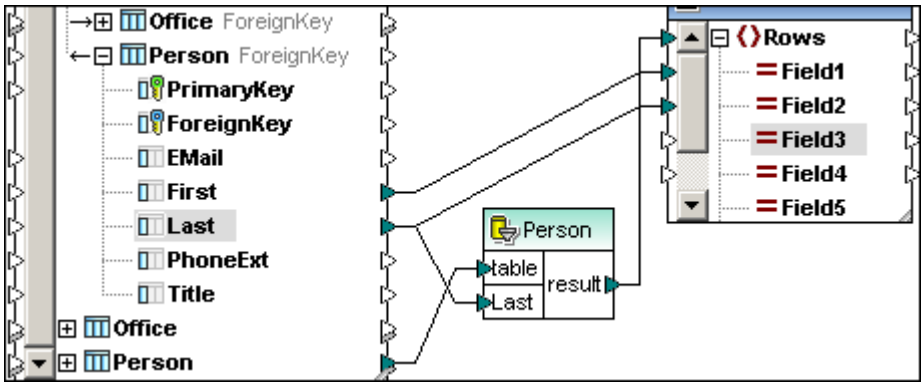


Parameters defined in the SQL WHERE window, that are **directly** connected to a database

field, must be mapped independently of the table they query, i.e. the **table** item and the **Last** item cannot both be mapped from the same table. If this is done, then the error message shown below appears when you preview or validate the mapping. Clicking the error message highlights the relevant connectors.

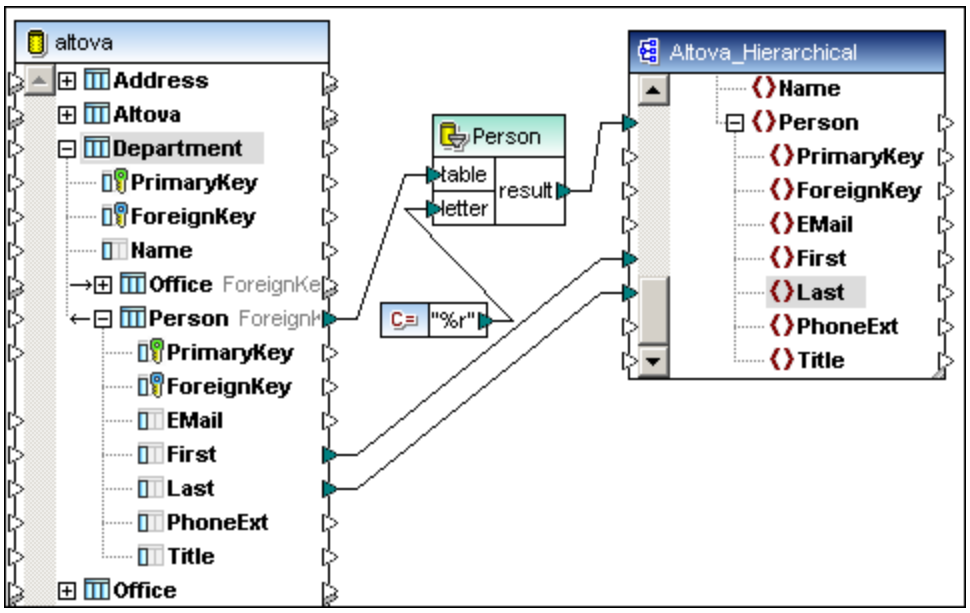


To make the parameter independent, move the **table** connector to a different **Person** table in the database hierarchy as shown in the screenshot below.



Select from Person WHERE
Last LIKE :letter

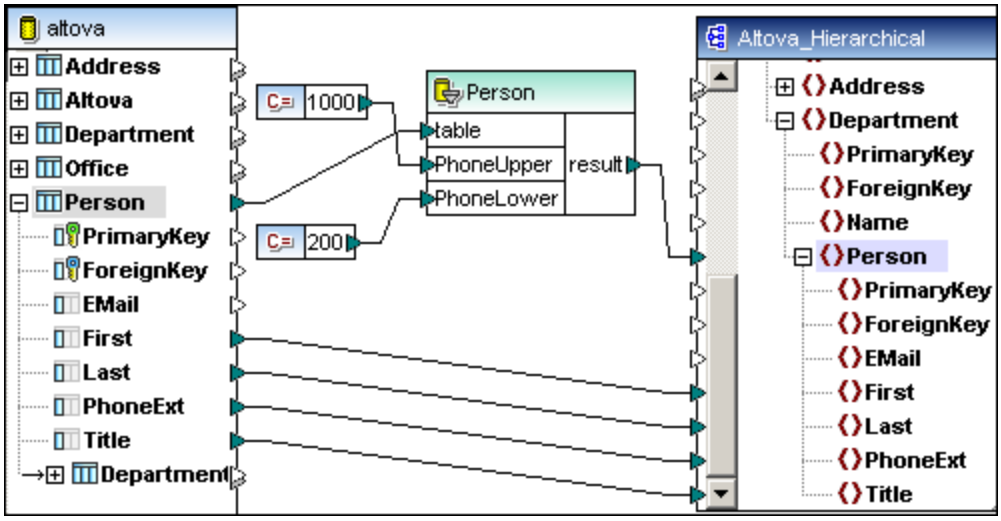
This WHERE statement creates a parameter **Letter** which appears as a parameter in the SQL WHERE component. In this case it is used to search for a pattern in the column "Last". The wildcard % denotes any number of characters.



The Constant component supplies the two values needed to search for all Last names ending in "r", i.e. **%r**.

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

8.11 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.

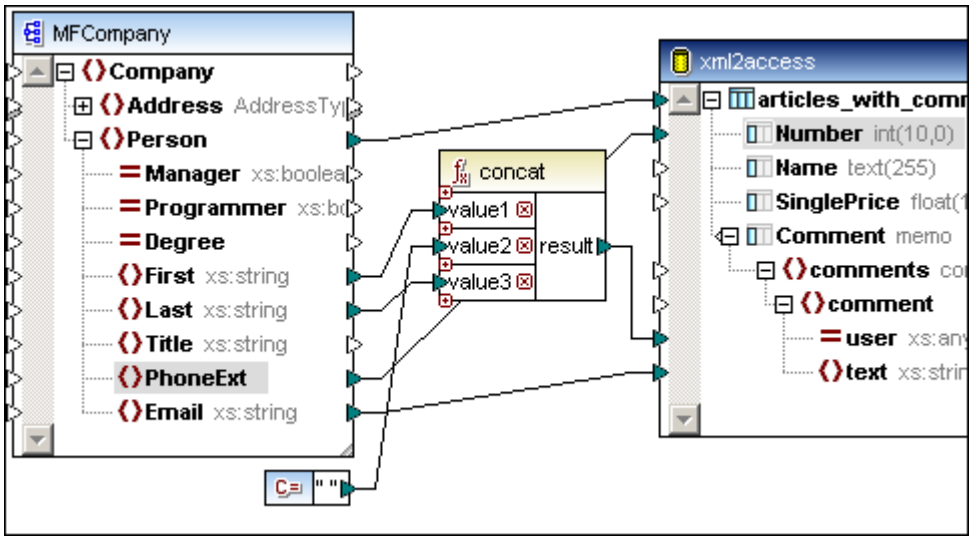
For the XML nodes to appear in the database component, which allows mapping to and from the XML items, a schema must be **assigned**. Once this is done the mapping process proceeds as usual.

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.



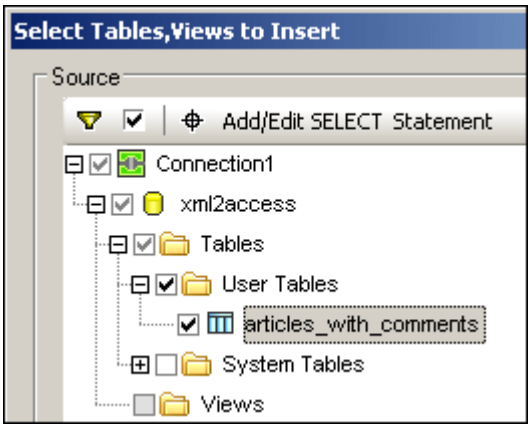
To insert the data source component:

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.

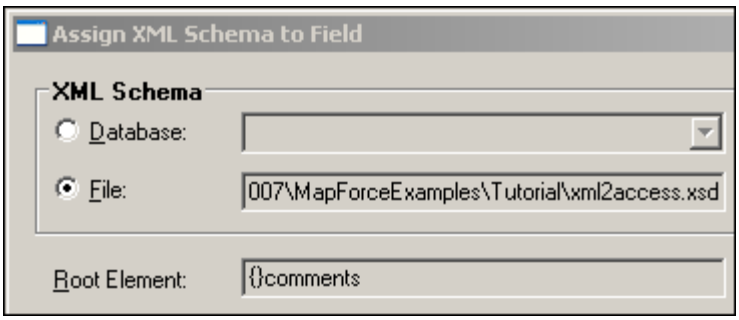
2. Click the Browse button and select the **xml2access.mdb** file in the ...**Tutorial** folder, then click Next.



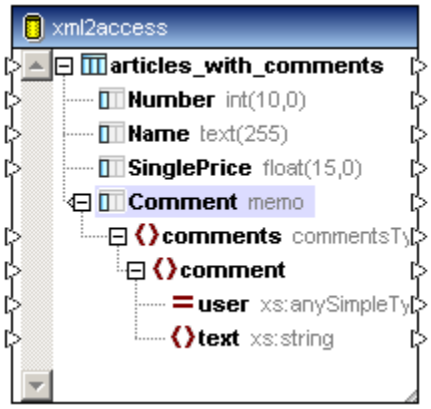
3. Click the **articles_with_comments** table check box then click OK.

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.

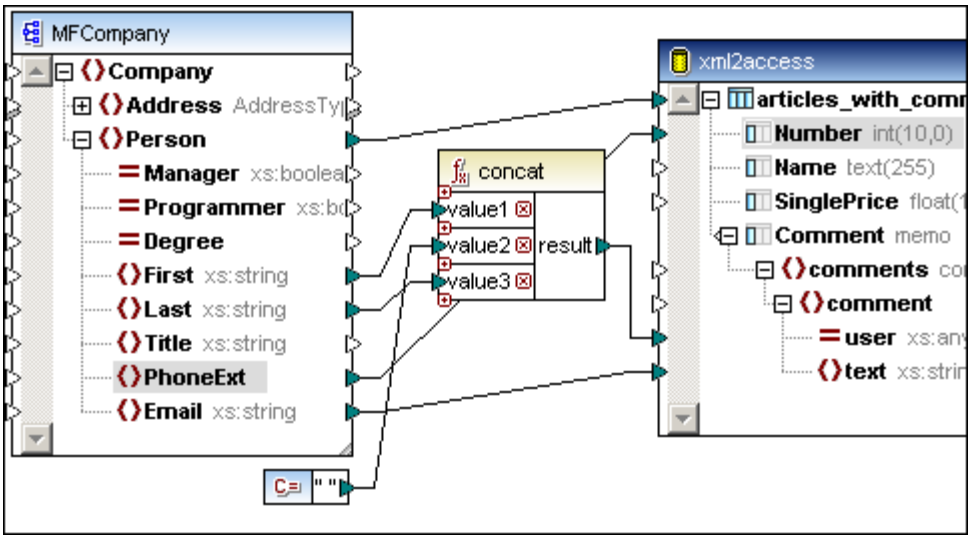


4. Expand the **Comment** item to be able to create connectors to the respective items.



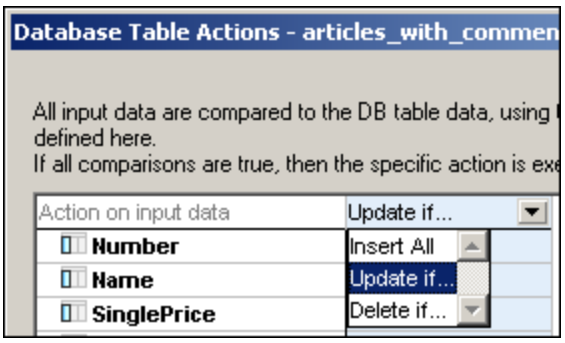
The XML schema node tree containing all mappable items appears below the Comment item/column. You are now ready to map XML data to the database.

5. Create the mapping connectors as seen in the top screenshot.

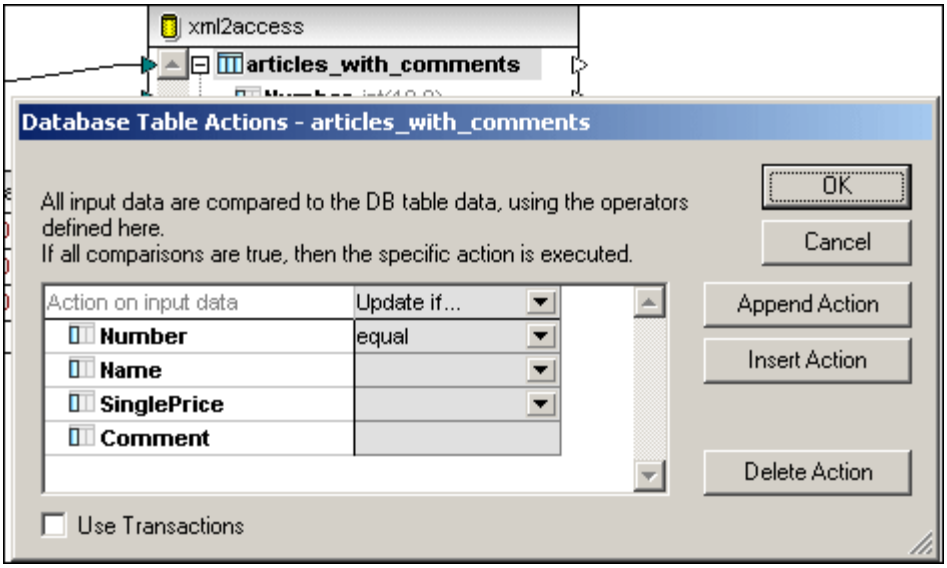


To define the table actions and map the data:

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).
```

8.12 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.

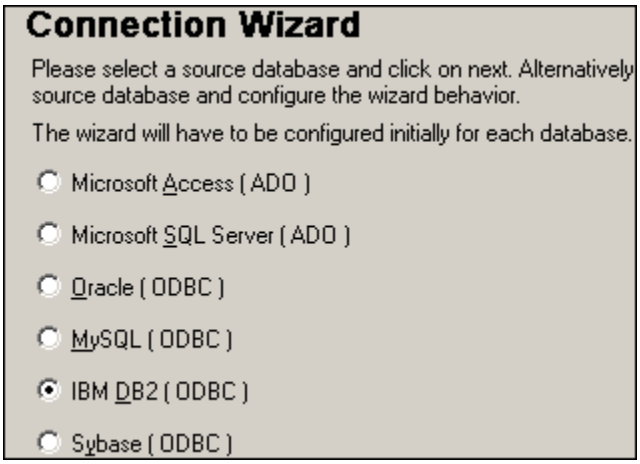
MapForce currently supports the assigning of **one** XML Schema per database column, and the selection of a **single** "root" element of that schema.

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.

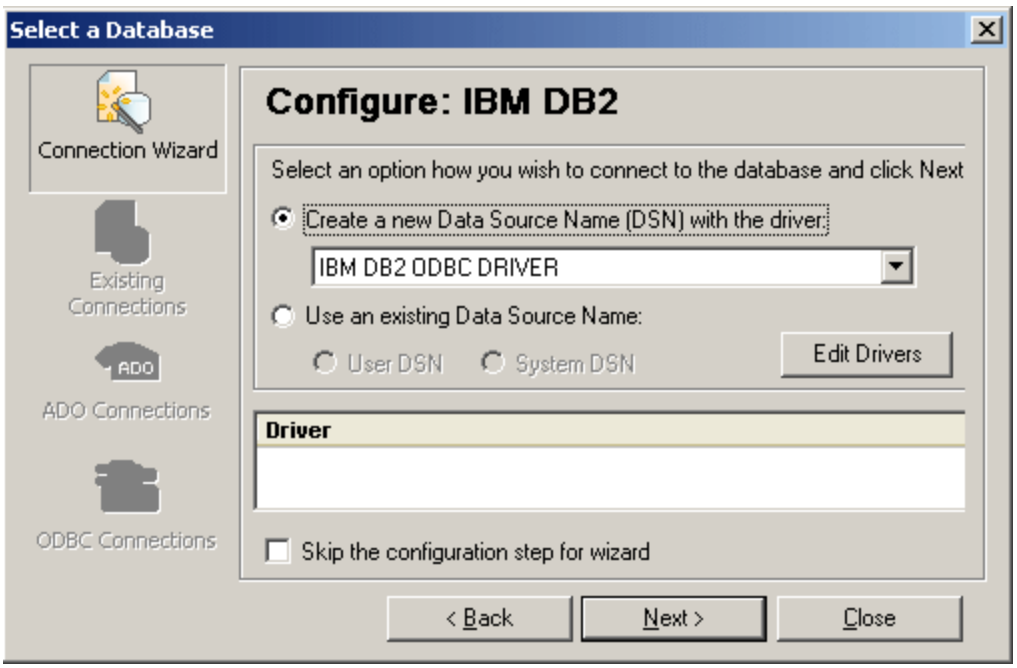
When adding an ODBC data source for iSeries Access ODBC driver, the "iSeries Access for Windows ODBC Setup" dialog box is opened. Select the "Performance" tab, click the "Advanced" button and **uncheck** the "Allow query timeout" check box option.

Configuring and inserting an IBM DB2 database:

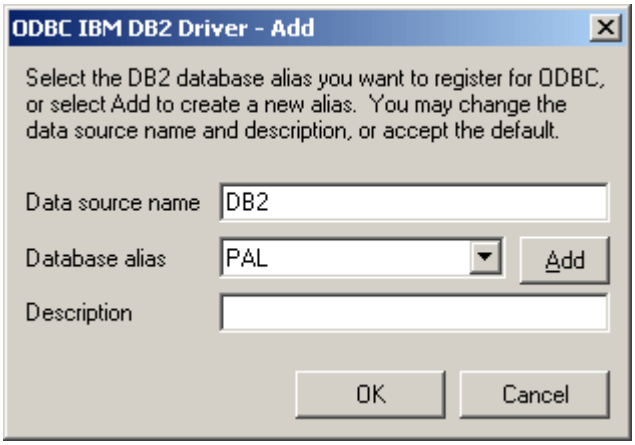
1. Click one of the programming language icons in the title bar: Java, C#, or C++.
2. Click the **Insert Database** icon  in the icon bar.



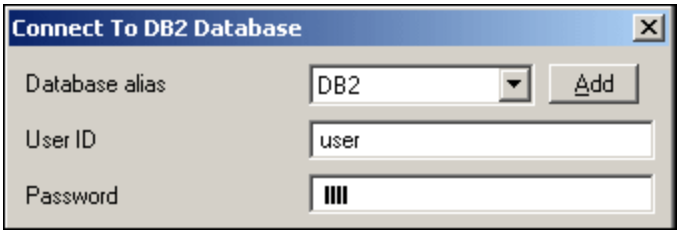
3. Click the IBM DB2 (ODBC) radio button and click Next.
   You can now define if you want to create a new Data Source Name or use an existing one.



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.

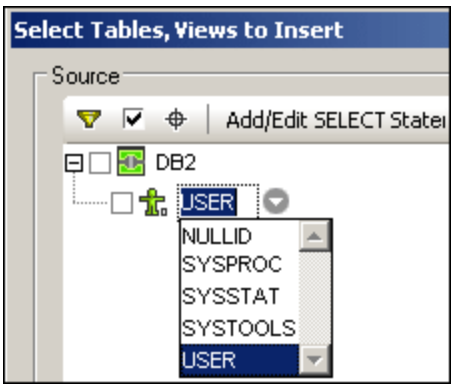


6. Enter the login data and click OK to connect to the database.

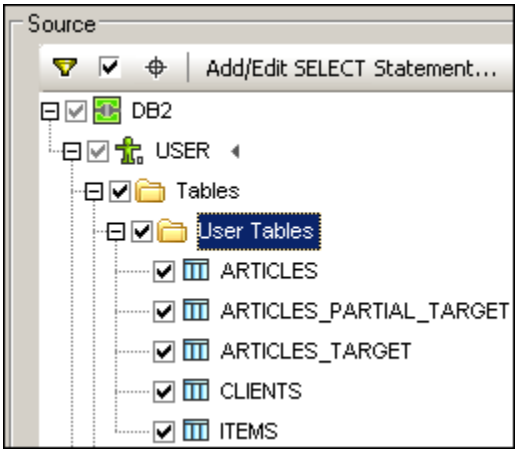


This opens the Select Tables dialog box.

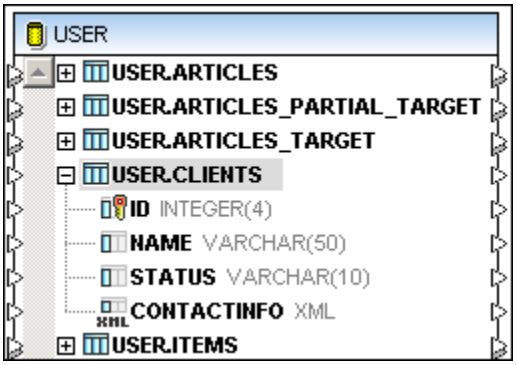
7. Click the database schema icon  and select the correct database schema e.g. USER.



All USER tables are now visible.

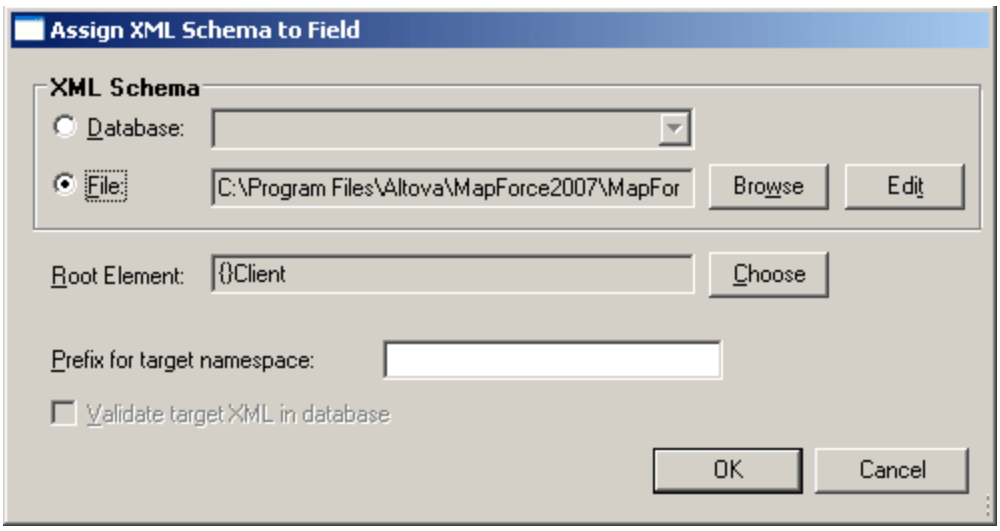


8. Click the check box next to "User Tables" to select all child tables, then click OK to insert the database component.
9. Click the expand icon next to the USER.CLIENTS table to see its contents. Note that the CONTACTINFO column is of type **XML**.

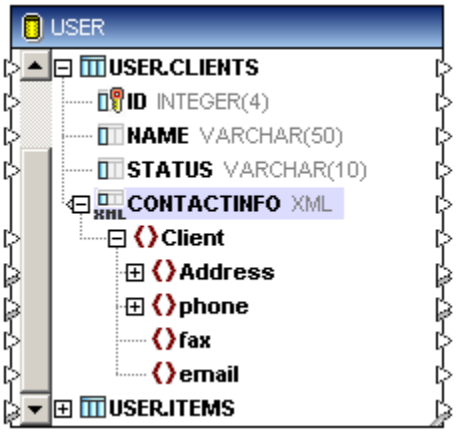


Assigning an XML Schema to an XML file:

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.

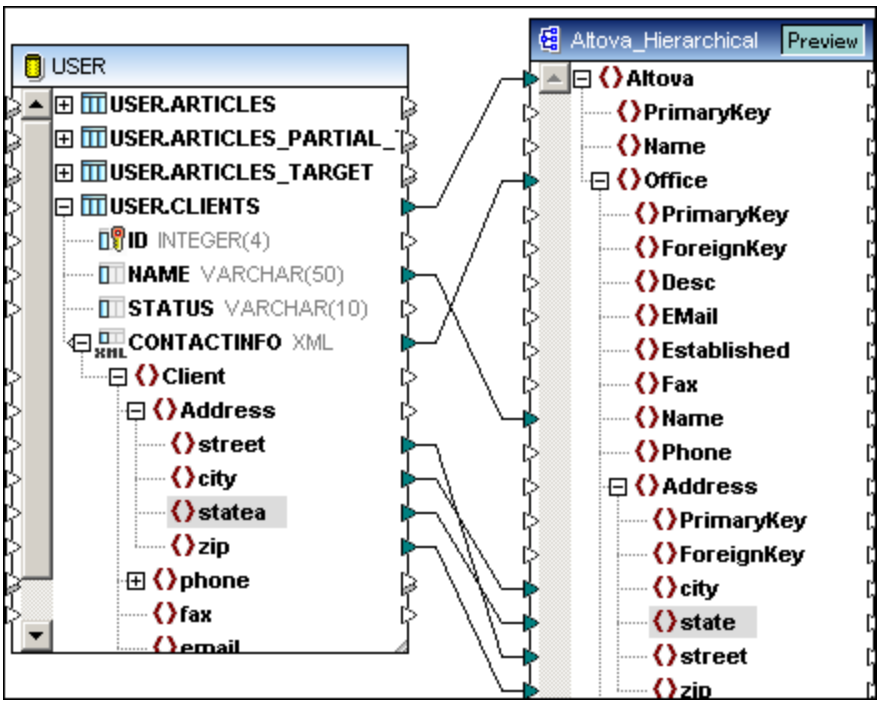


3. Choose the Root element of the schema that is to appear in the component e.g. Client, and click OK to confirm.

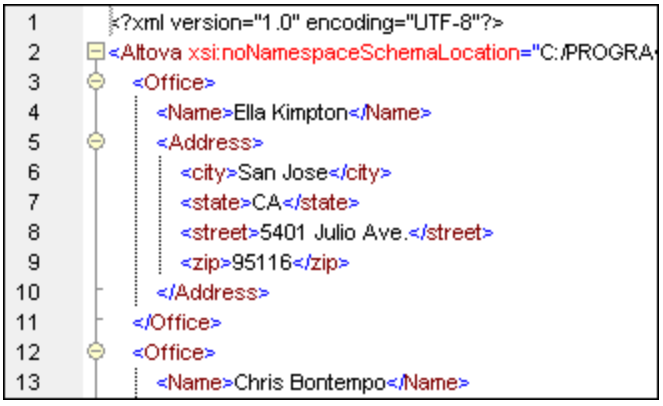


The "Client" item appears below CONTACTINFO; click the expand icons to see the schema structure.

4. Map connectors from the schema items to the target component, which is an XML document in this case.

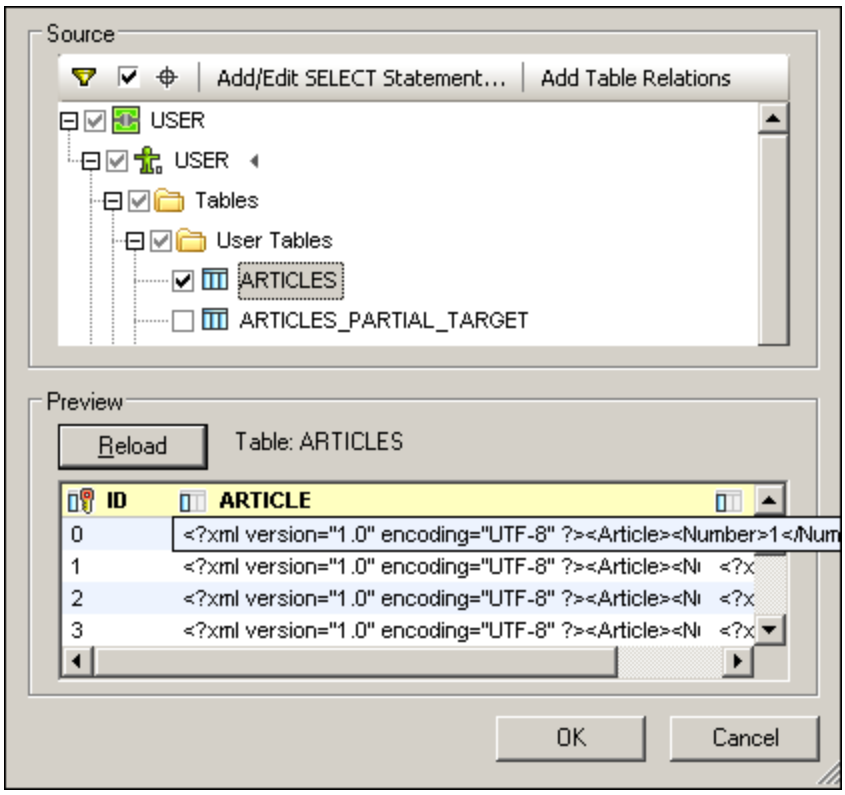


5. Click the Output button to see the result of the mapping.



Previewing table content

Clicking the Preview button, while the **Select Tables / Views to insert** dialog box is open in the Quick connect 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.



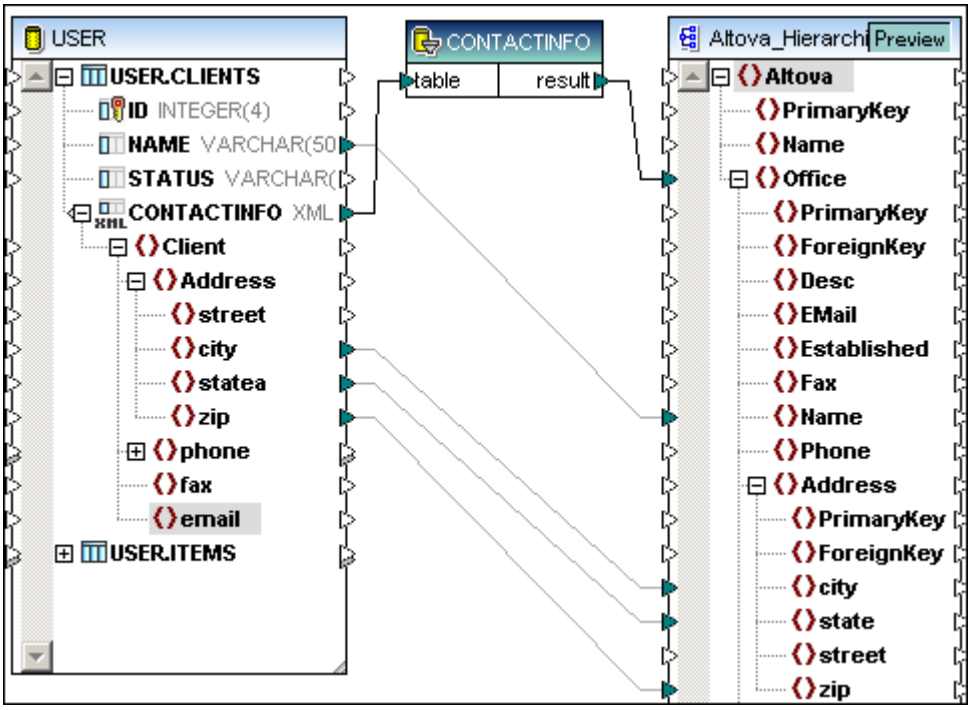
8.12.1 Querying and mapping XML data in IBM DB2

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.

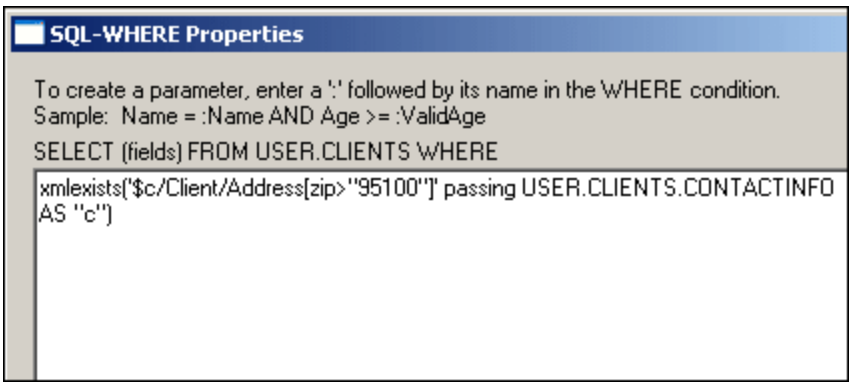
Having inserted the DB2 database and assigned the XML schema to the CONTACTINFO item:

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.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Altova xsi:noNamespaceSchemaLocation="C:/PROGRA~1/Altova
3    <Office>
4      <Name>Ella Kimpton</Name>
5      <Address>
6        <city>San Jose</city>
7        <state>CA</state>
8        <zip>95116</zip>
9      </Address>
10   </Office>
11   <Office>
12     <Name>Chris Bontempo</Name>
13     <Address>
14       <city>San Jose</city>
15       <zip>95124</zip>
16     </Address>
17   </Office>
```

8.12.2 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.

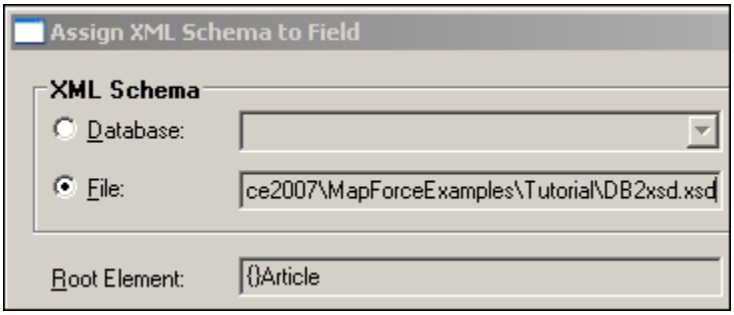
Please note that you can also query databases directly using the Database Query tab, please see Querying databases directly - Database Query tab for more information.

To insert the data source component:

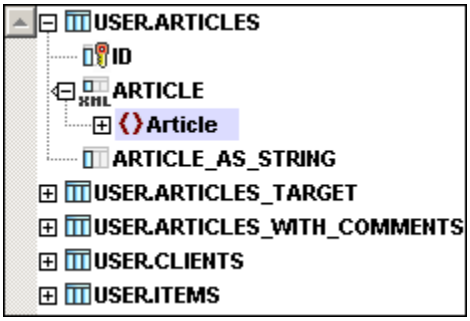
1. Click the **Insert XML Schema/File** icon , and select the **DB2asTarget.xsd** schema.
2. Click Browse when the prompt for a sample XML file appears, and select **DB2asTarget.xml**
3. Select **Articles** as the root element and expand all items.

To insert the database target and map data to it:

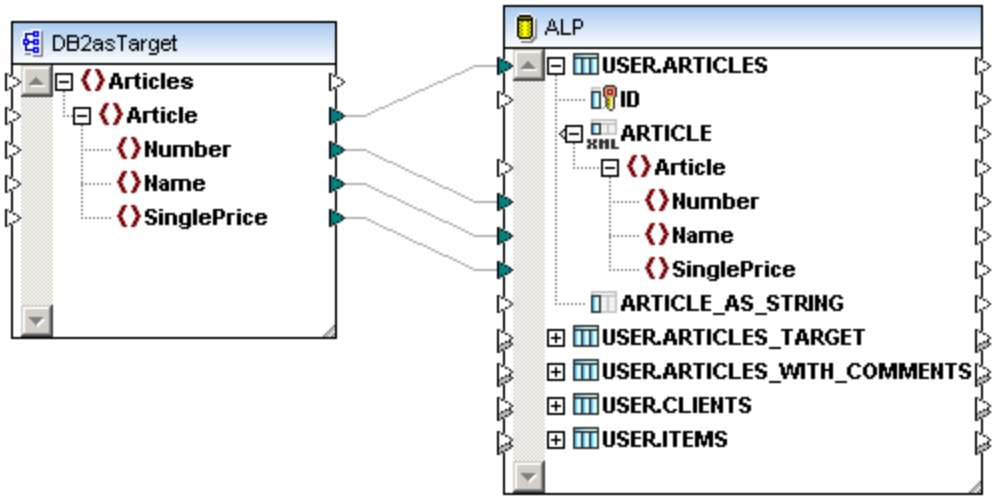
1. Using the method discussed in insert the IBM DB2 database to insert the database.
2. Right click the ARTICLE item/column in the database target and select **Assign XML Schema to Field...**
3. Click the **File** radio button, select **DB2xsd.xsd** and click OK.



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.

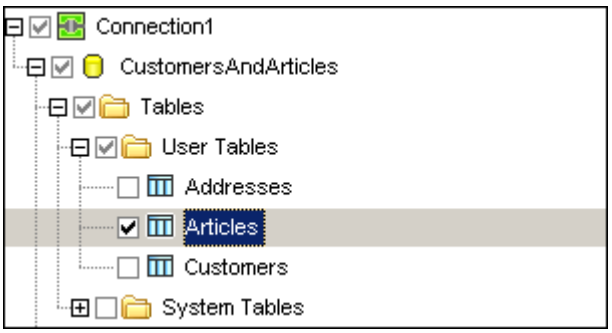
8.12.3 Mapping data - database to database

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:

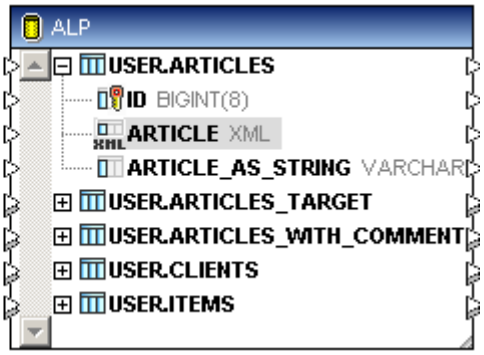
1. Click the **Insert Database** icon , and select the **CustomersAndArticles.mdb** database from the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder.
2. Select the **Articles** table and click OK to insert.



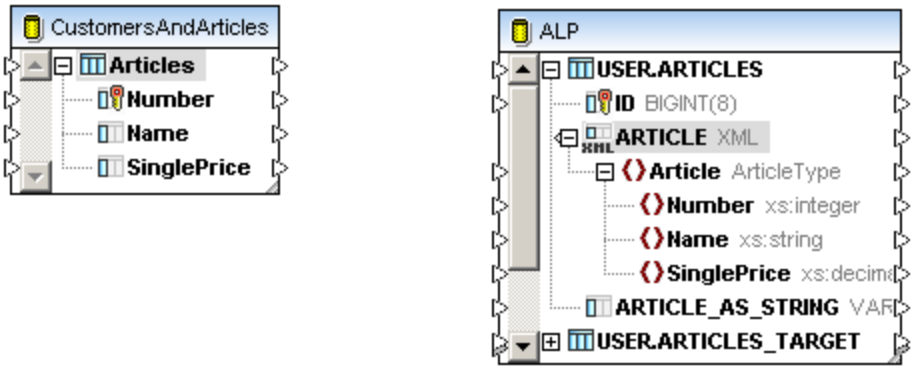
The database table is now visible as a database component.

To insert the IBM DB2 target database and map data to it:

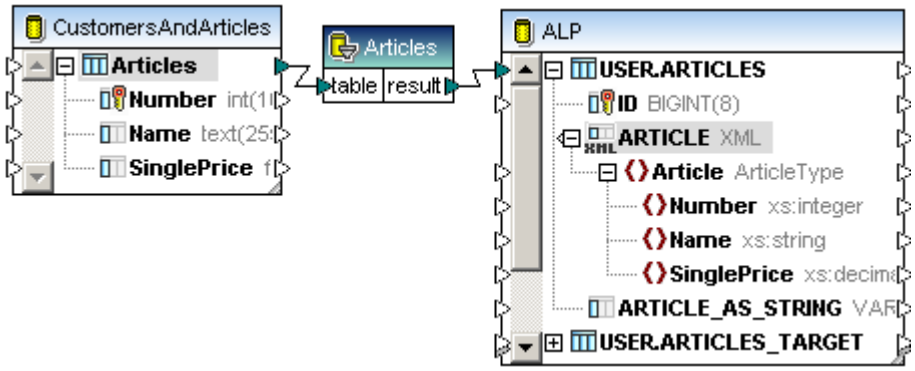
1. Using the method discussed in insert the IBM DB2 database to insert the database.



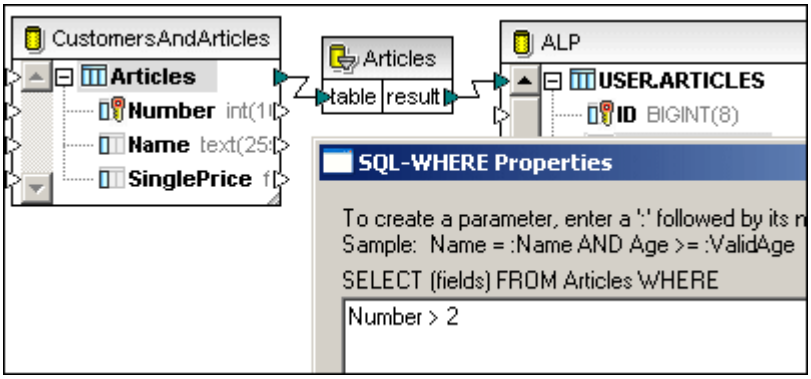
2. Right click the ARTICLE item/column in the database target and select **Assign XML Schema to Field...**
3. Click the **File** radio button, select the **DB2xsd.xsd** schema file, then click OK.



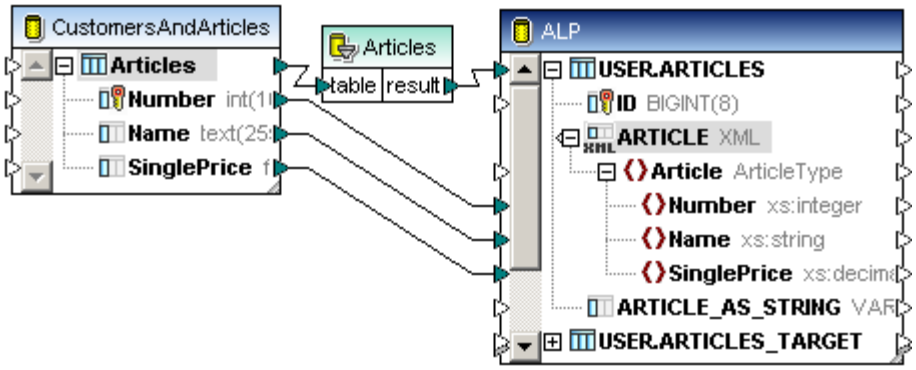
4. Click the SQL WHERE icon  in the icon bar to insert it.
5. Connect the **Articles** item, of the database source, to the **table** item component.
6. Connect the **result** item to the USER.ARTICLES item in the target component.



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>')
```

8.13 Querying databases directly - Database Query tab

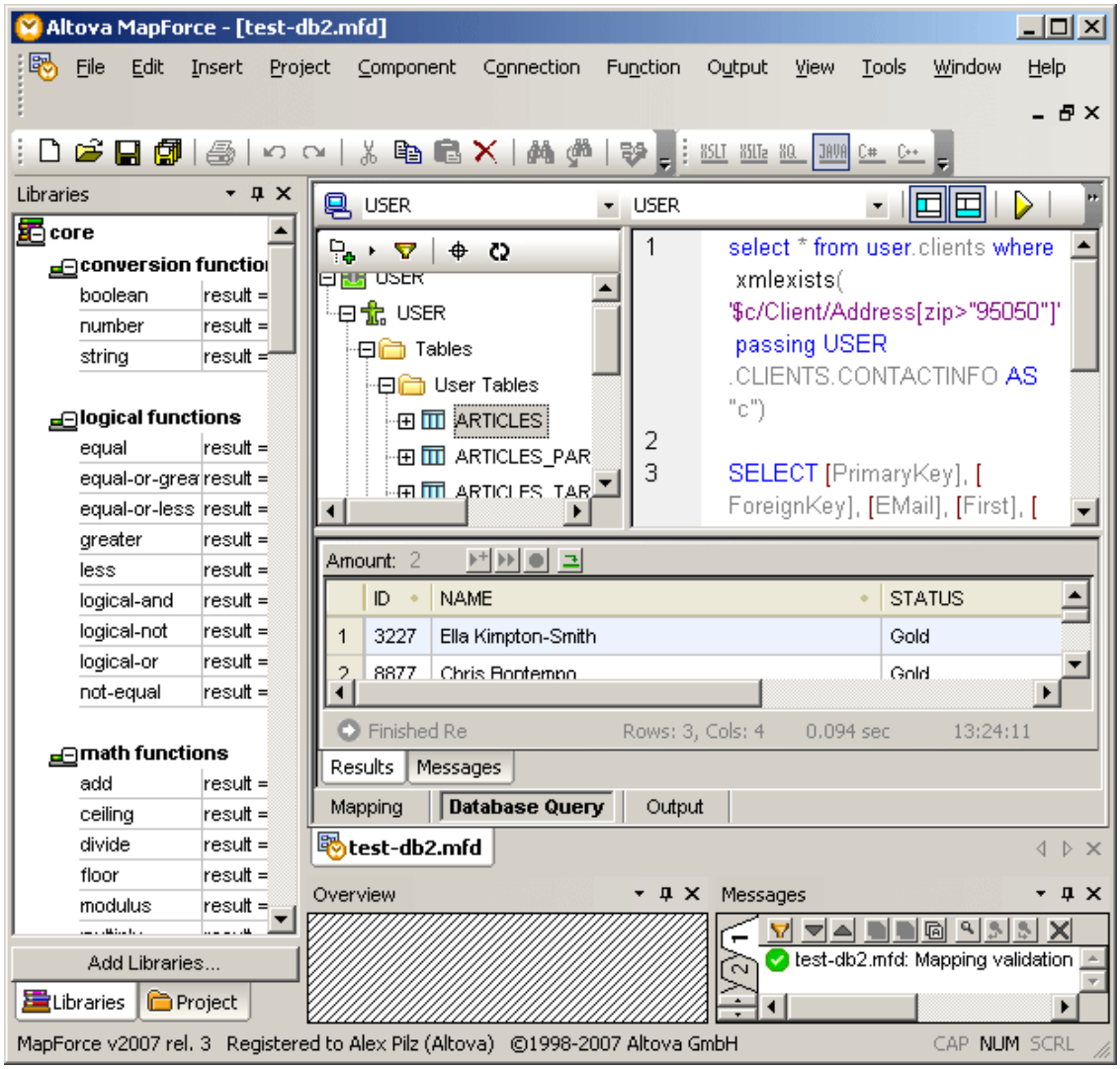
MapForce has a dedicated database query tab that allows you to directly query any major database. The queries/actions defined here are independent of any of the other MapForce tabs, and are not saved as part of the *.MFD file.

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.



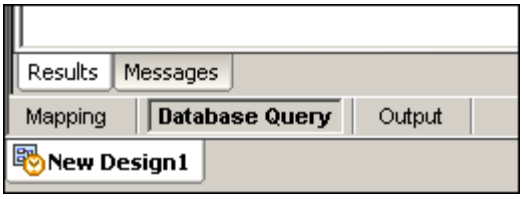
8.13.1 Selecting / connecting to a database

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. You can also

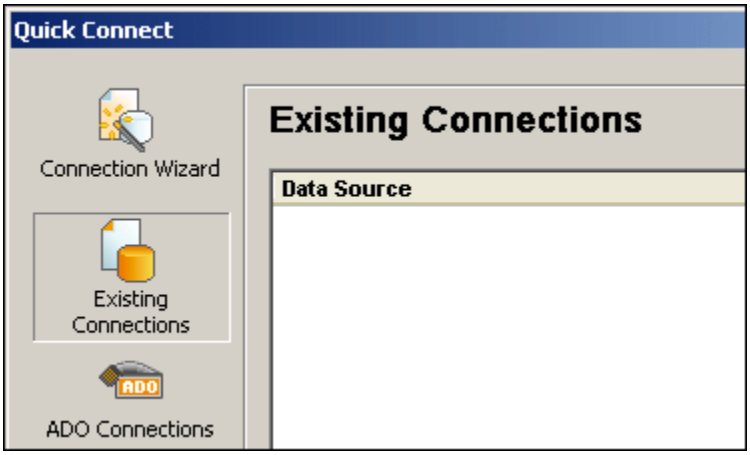
The examples shown in this section use a sample database supplied with IBM DB2 version 9.

To connect to a database:

1. Click the **Database Query** tab in the main window.



2. Click the Quick Connect icon  in the Connection icon bar.

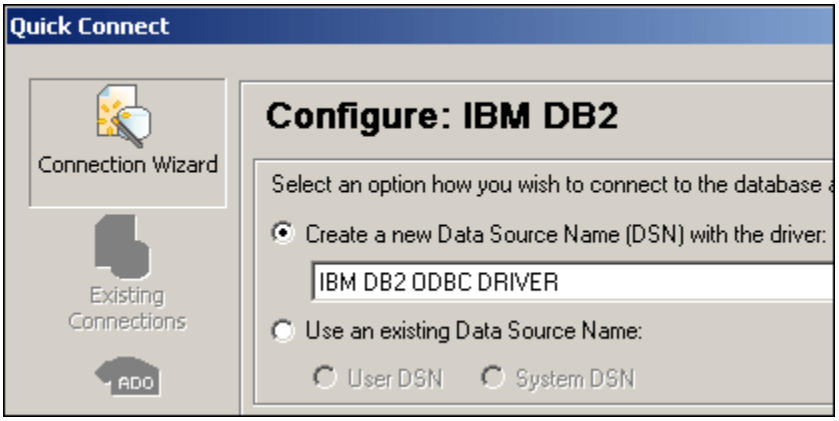


This opens the Quick Connect wizard through which you can connect to any type of database. If you have mappings open which contain database connections, then they will appear in the Existing Connection page shown here. At this point the assumption is that no connections are currently active.

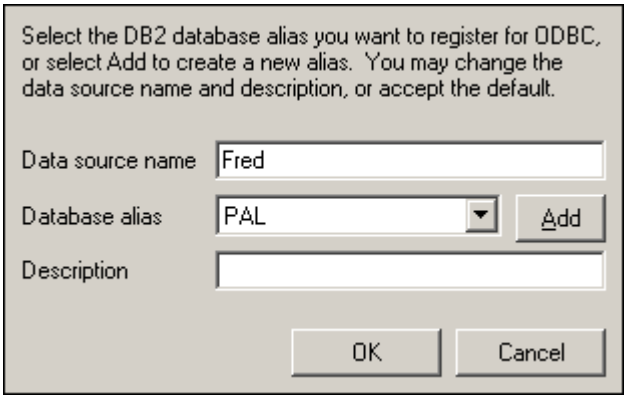
Clicking the Global Resources icon  in the left pane, allows you to select from databases that have been defined as global resources, please see Global Resources - Databases for more information.

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.

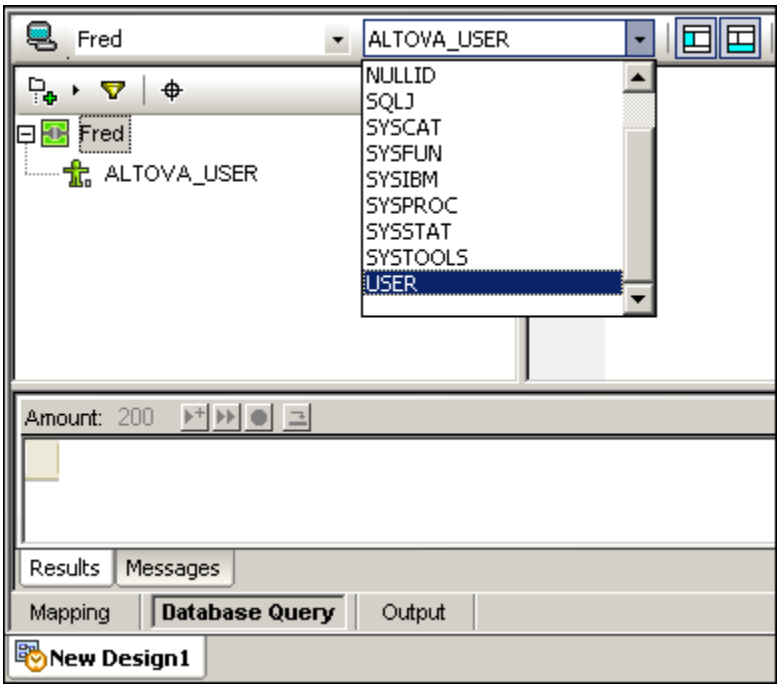


5. Click Next if you want to create a new Data Source Name (DSN), or click the "Use an existing Data Source Name" radio button if you previously defined a DSN.

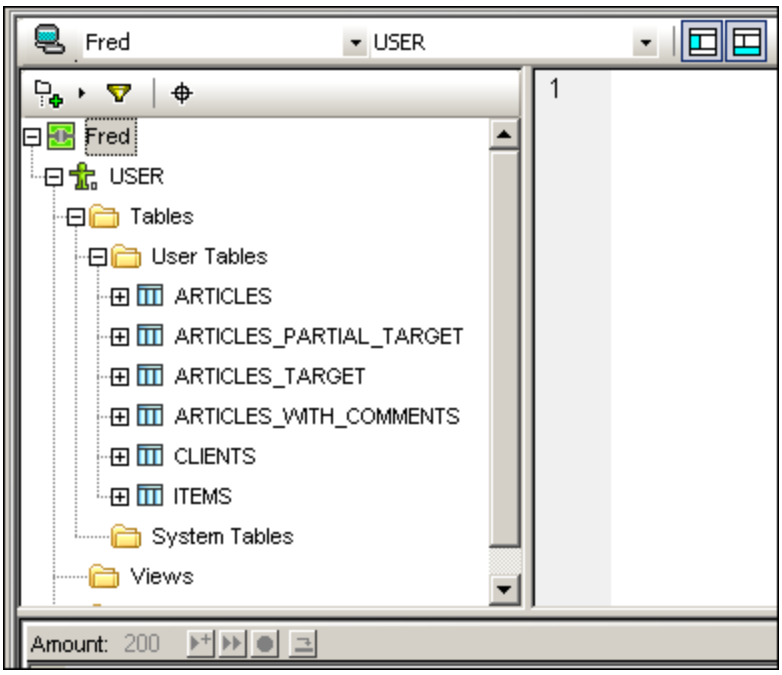


6. Enter the DSN name and the database alias, then click OK to continue.

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.



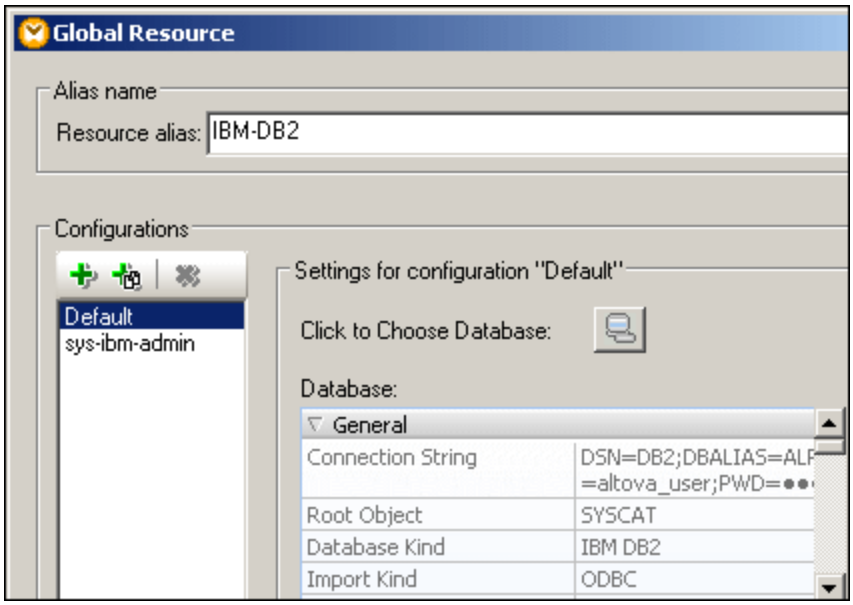
The database tables are now visible under the Tables folder. The "root" objects for the various databases are shown in the table below:

MS SQL Server	database
Oracle	schema
MS Access	database
MySQL	database
DB2	schema
Sybase	database

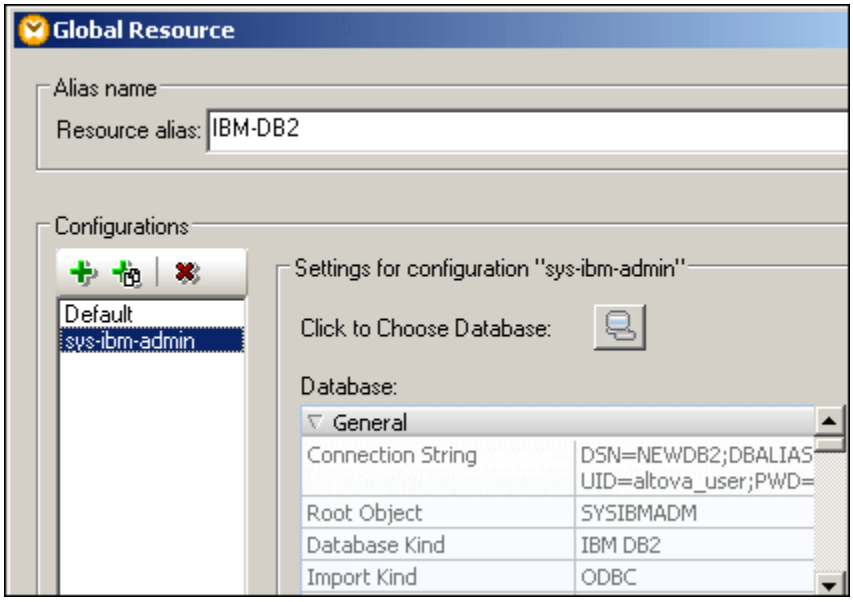
8.13.2 Selecting a database Global Resource

Clicking the Global Resources icon in the Quick Connect 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.



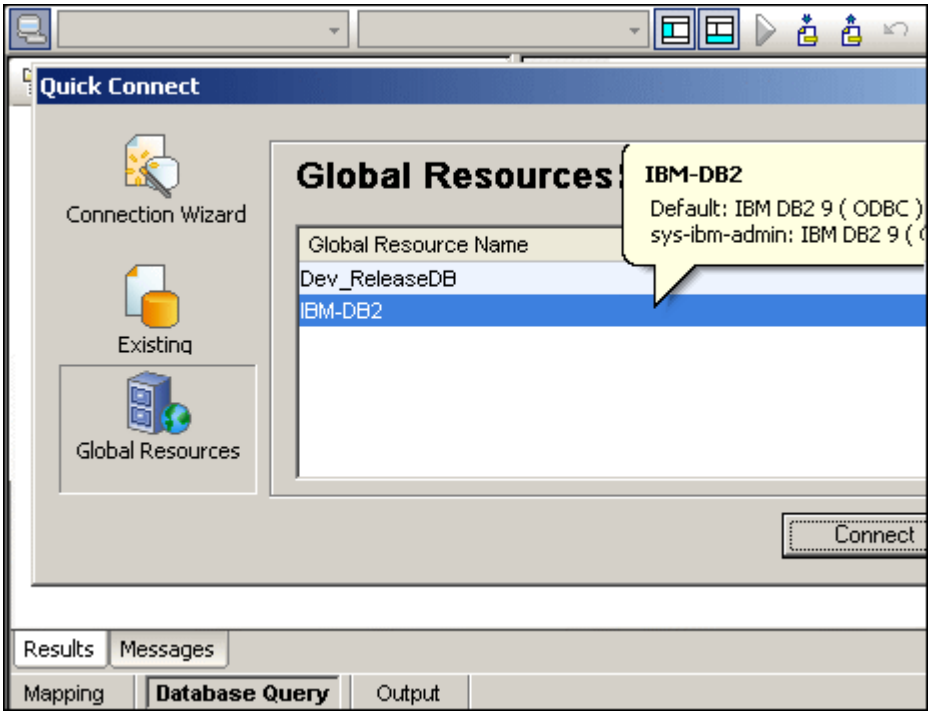
Whereas the **sys-ibm-admin** configuration accesses the same database but only the SYSIBMADM tables/Root object.



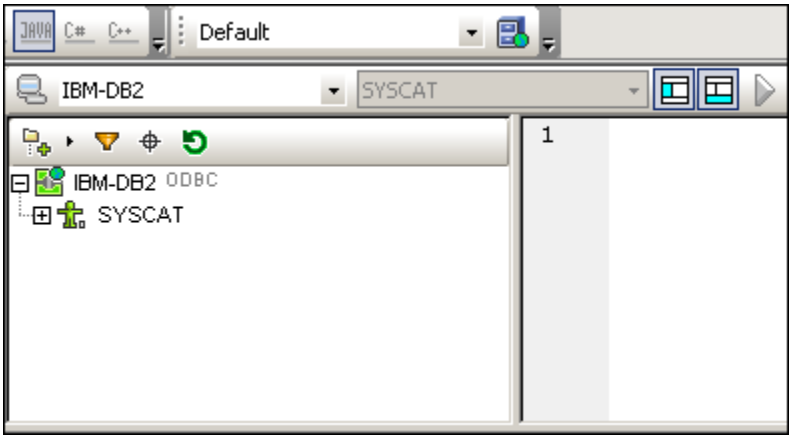
To switch the configuration in the DB-Query window:

1. Click the Database Query tab to open the query window.

2. Click the Quick Connect icon  in the Connection icon bar, then click the Global Resources icon.

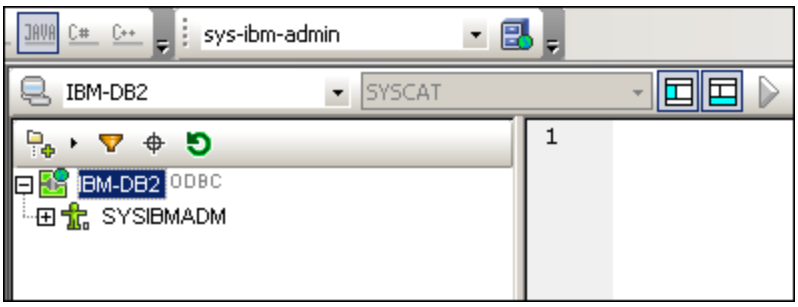


3. Click the Global Resource name e.g. IBM-DB2 and click the Connect button.



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.

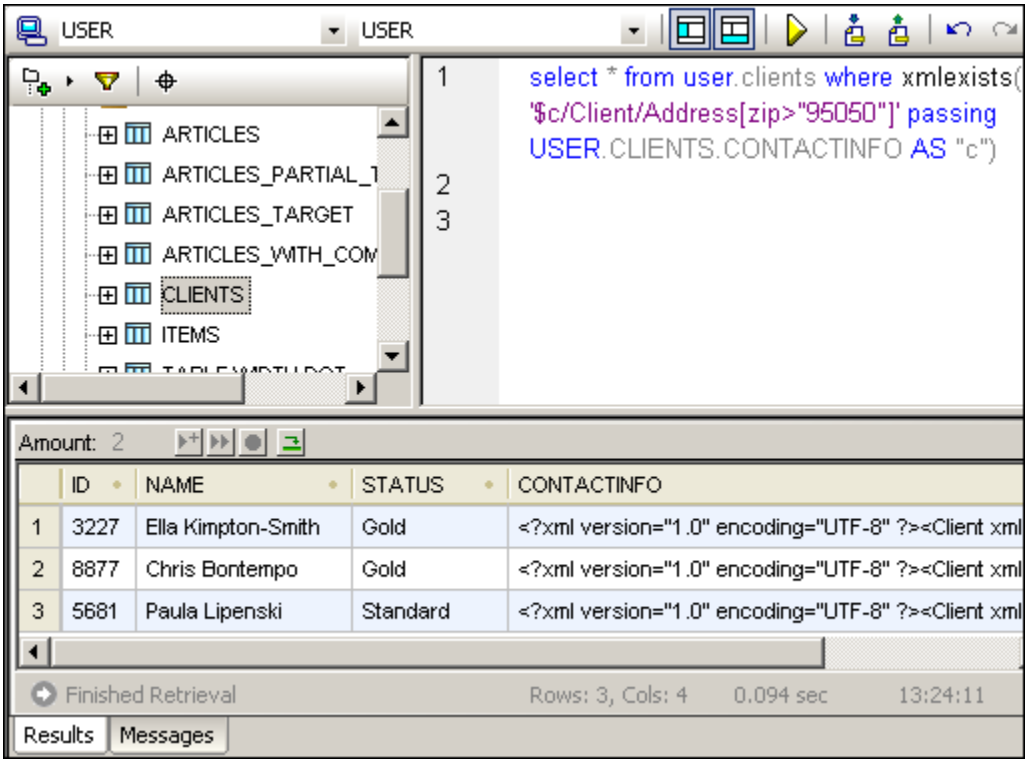


The new root object / table is now visible and you can query the specific tables.

8.13.3 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.

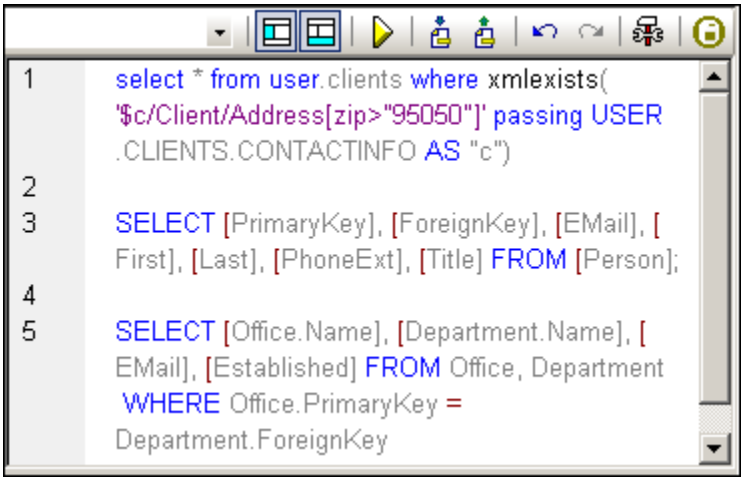
The Retrieval mode allows you to copy, select, or sort the data in the Results tab, by right clicking and selecting the respective option from the context menu. Please see Database Query - Results & Messages tab for more information.

8.13.4 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 <%SQLSPY%>: Starts <%SQLSPY%> 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.

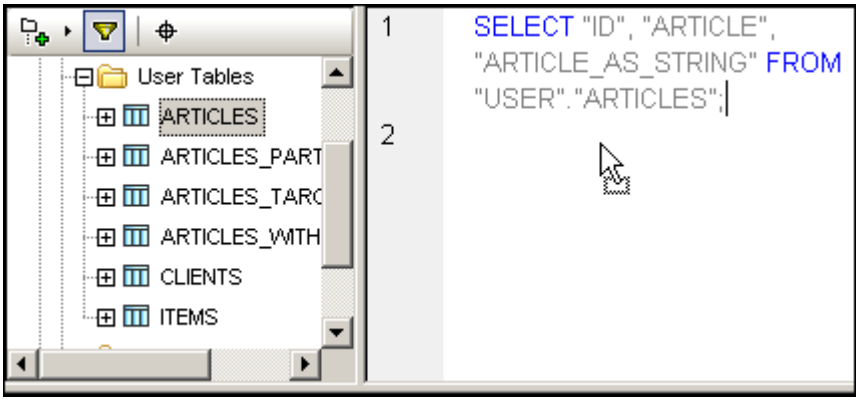
Generating SQL statements

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.



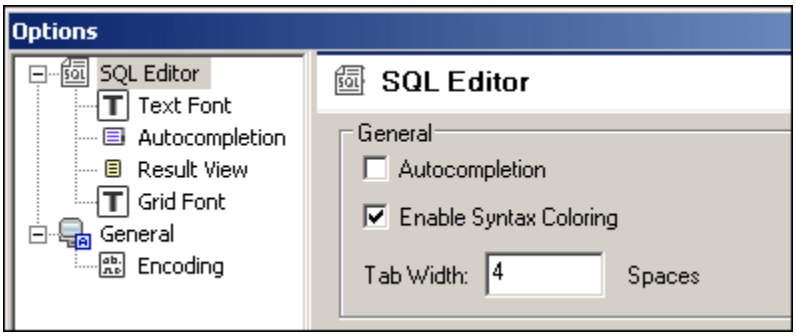
An SQL statement appears in the SQL Editor.

To generate SQL statements using the context menu:

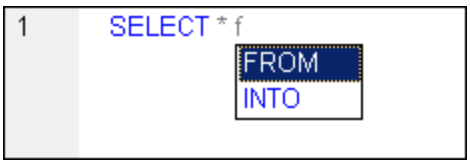
1. Right-click a database object in the Browser and select **Show in SQL Editor | Select**.

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.



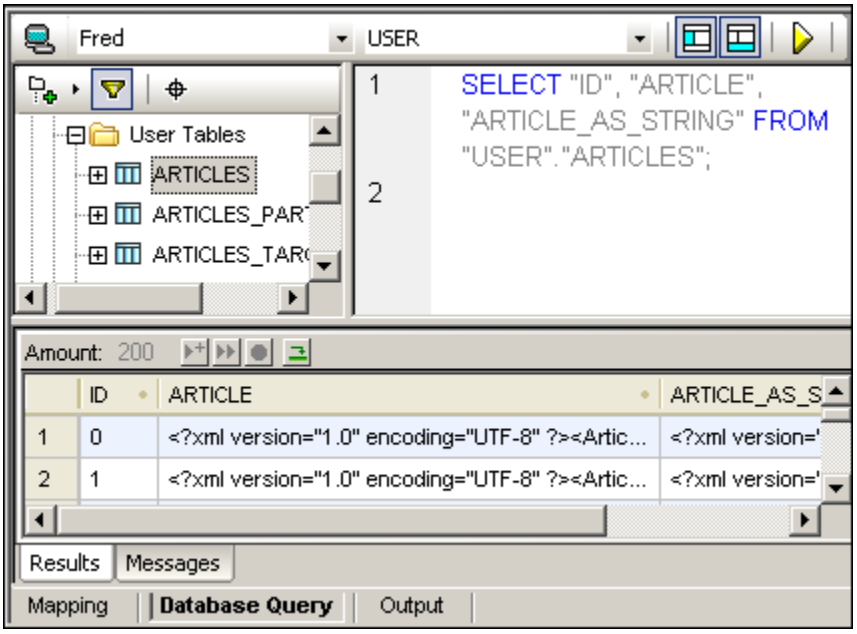
Autocomplete popups appear while entering the select statement. Press Enter when you want to insert/use a highlighted option.

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

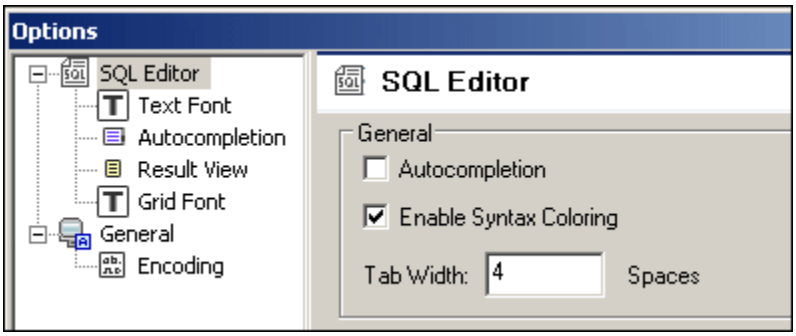
Autocompletion

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:

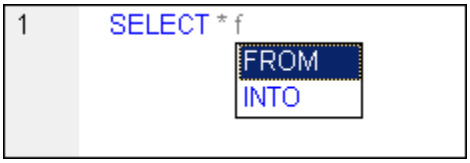
- MS SQL Server 2000
- MS SQL Server 2005
- MS Access 2003
- IBM DB2 9

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.



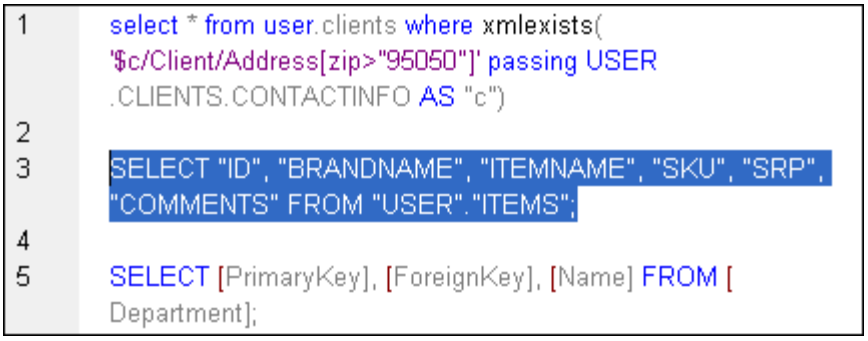
Autocomplete popups appear while you enter the statement. Press Enter when you want to insert a highlighted option.

Commenting out text

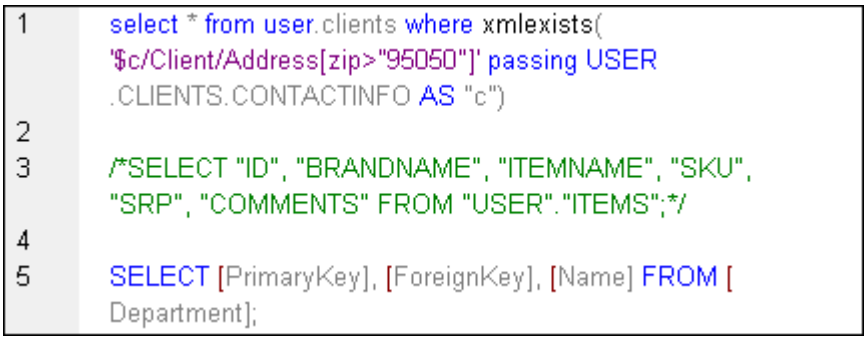
The SQL Editor allows yout to comment out statements, parts of statements, or groups of statements. These statements, or the respective parts of them, are skipped when the SQL script is being executed.

To comment out a section of text:

1. Select a statement or part of a statement.



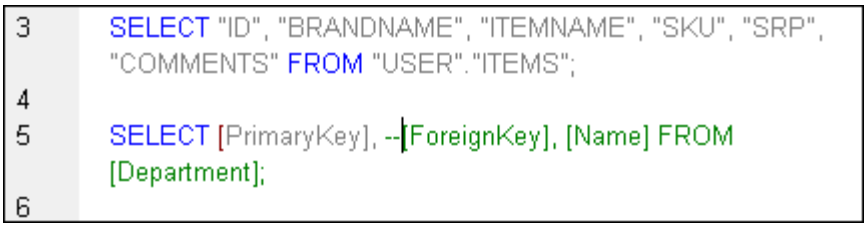
2. Right click the selected statement and select **Insert / Remove Block Comment**.



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**.



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:

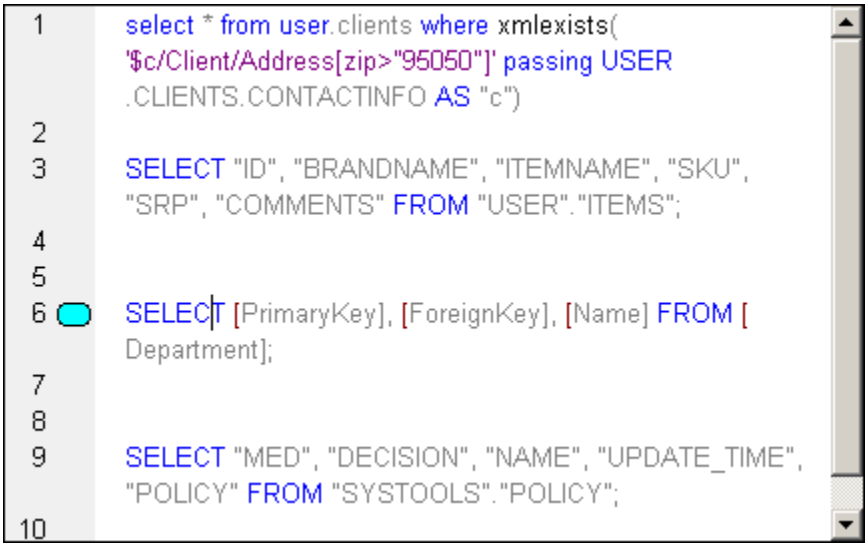
1. Select the part of the statement that is commented out.
   If you want to remove a line comment, it is sufficient to select only the comment marks -- before the comment.
2. Right click and select **Insert / Remove Block (or Line) Comment**.

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

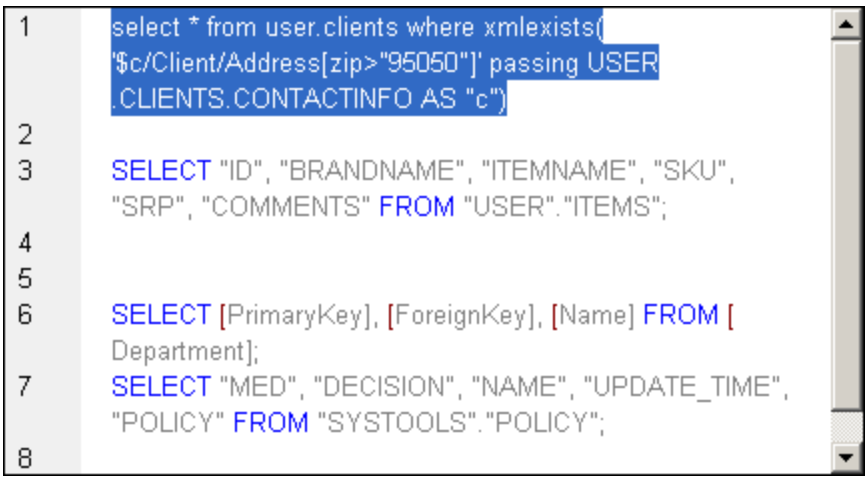
Regions are sections of text that you mark and declare as a unit to structure your SQL scripts. Regions can be collapsed and expanded to display or hide parts of SQL scripts. It is also possible to nest regions within other regions.

When you insert a region, an expand/collapse icon and a `--region` comment are inserted above the selected text.

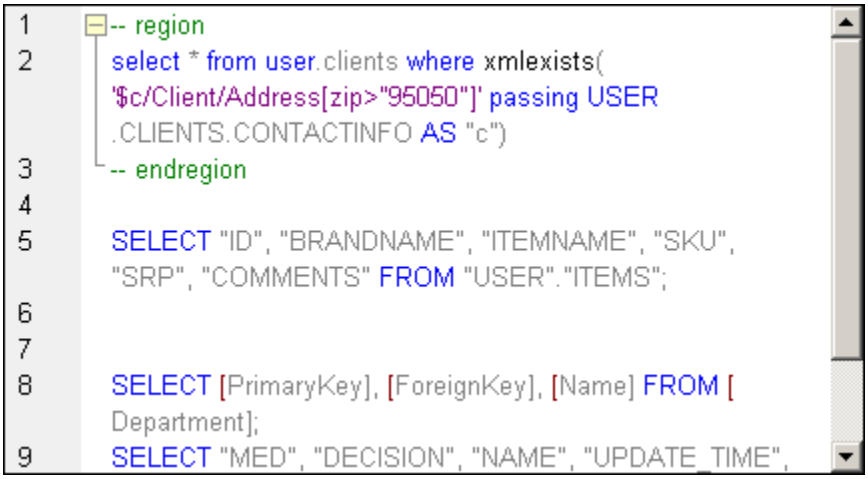
Please note: You can change the name of a region by appending descriptive text to the --region comment. The word "region" must not be deleted, e.g. --region DB2query.

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.

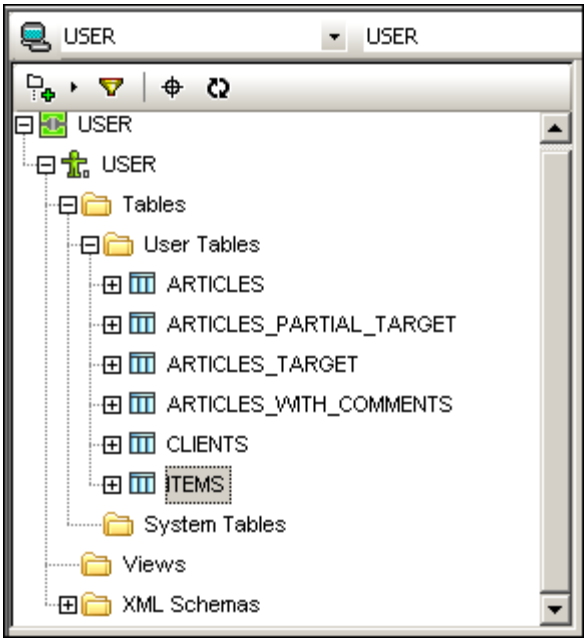
8.13.5 Database Query - Browser window

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:

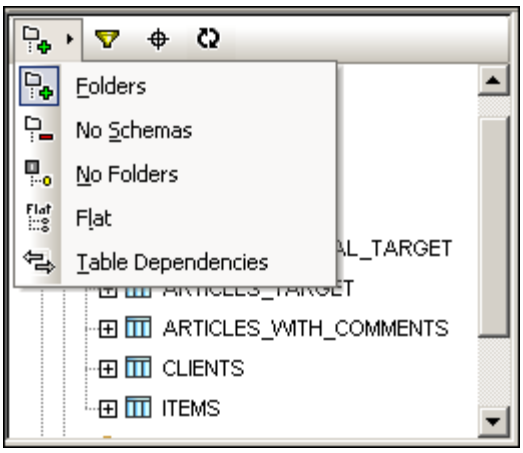
- show specific folder **layouts** when displaying database objects
- **find** specific objects in the database using the Object Locator
- **filter** the number of displayed items
- **refresh** the root object of the active data source.

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.

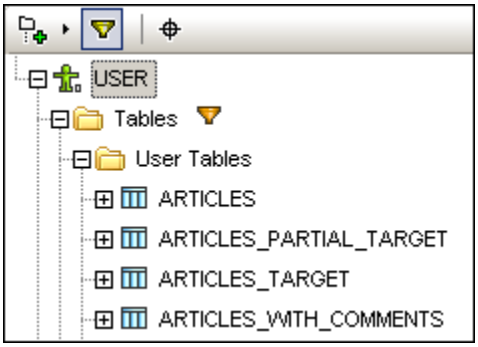
Filtering and finding database objects

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.

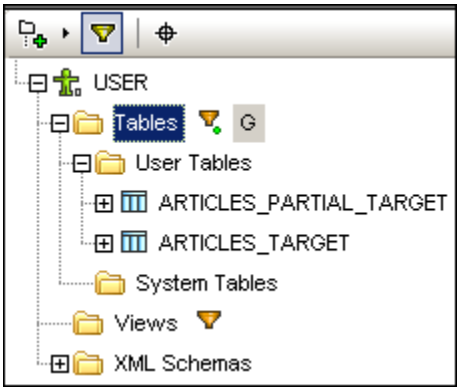
To filter objects in the Browser:

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.

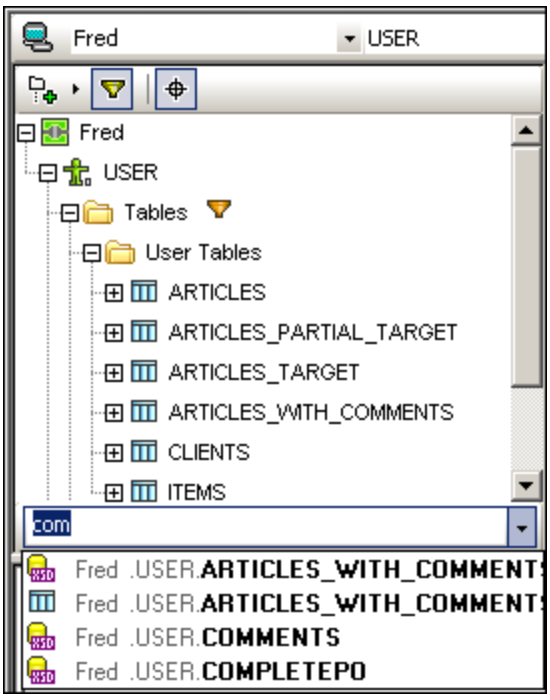


Finding database objects

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.

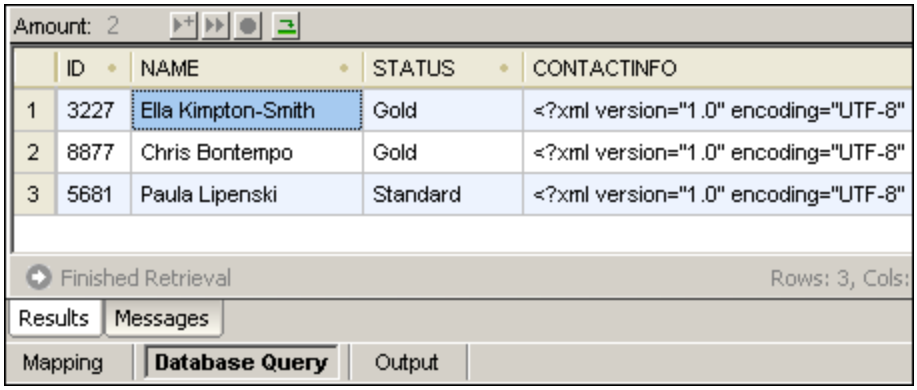
  E.g. "USER"."ARTICLES_PARTIAL_TARGET"."CC1175533269606"

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.

8.13.6 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

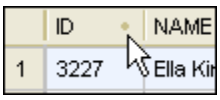
  Holding down the CTRL key while clicking allows you to make multiple selections. If a column, or cell, contains XML data then this data can also be copied.

- 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.

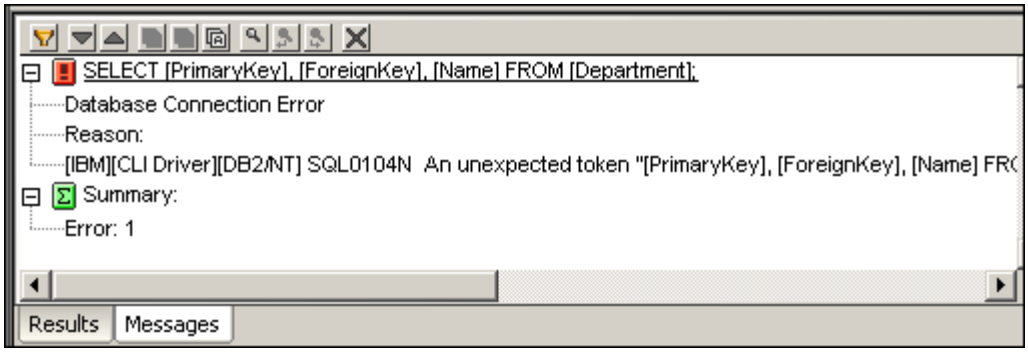
Retrieve next n rows: Retrieves the next n rows from the query in the active SQL Editor window. You can define the number of rows in the SQL Editor options.

Retrieve outstanding rows: Retrieves all the remaining rows from the query in the active SQL Editor window.

Stop retrieval: Stops the retrieval of database data. The data that has been retrieved so far is displayed in the Result tab. Use the **Retrieve next n rows** button or the **Retrieve outstanding rows** button, respectively, or the context menu to resume retrieval.

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.

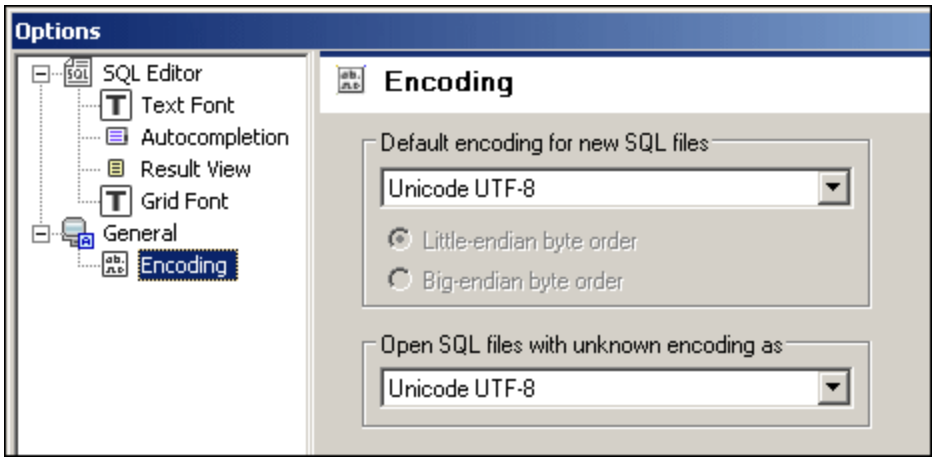
Find next: Jumps to the next occurrence of the string specified in the **Find** dialog box.

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.

8.13.7 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.

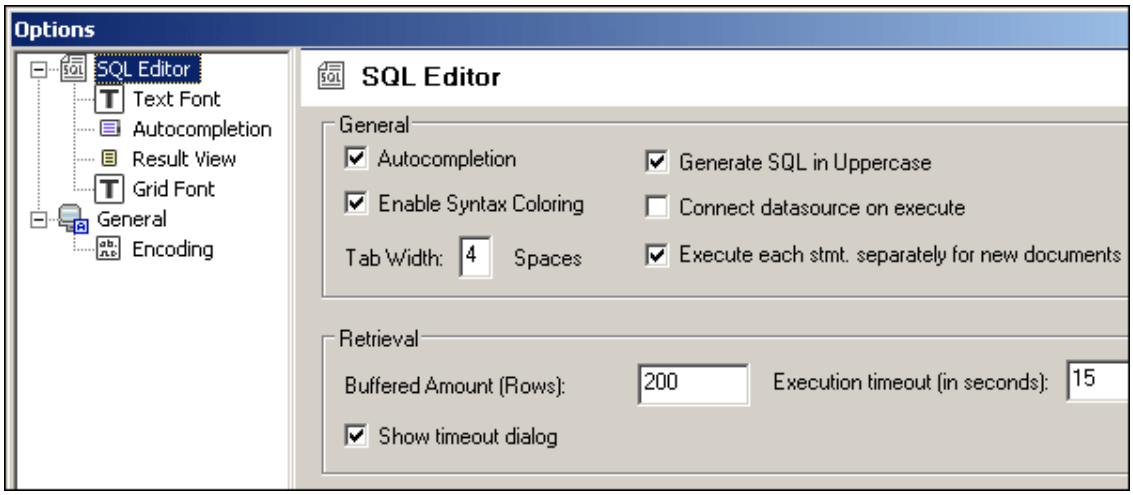
The encoding for existing files will, of course, always be retained.

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 Text Font, Autocompletion, Result View, and Grid Font 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.

You can choose to **generate SQL in Uppercase** for a better overview.

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.

To execute the individual SQL statements individually, activate the **Execute each stmt. separately for new documents** check box.

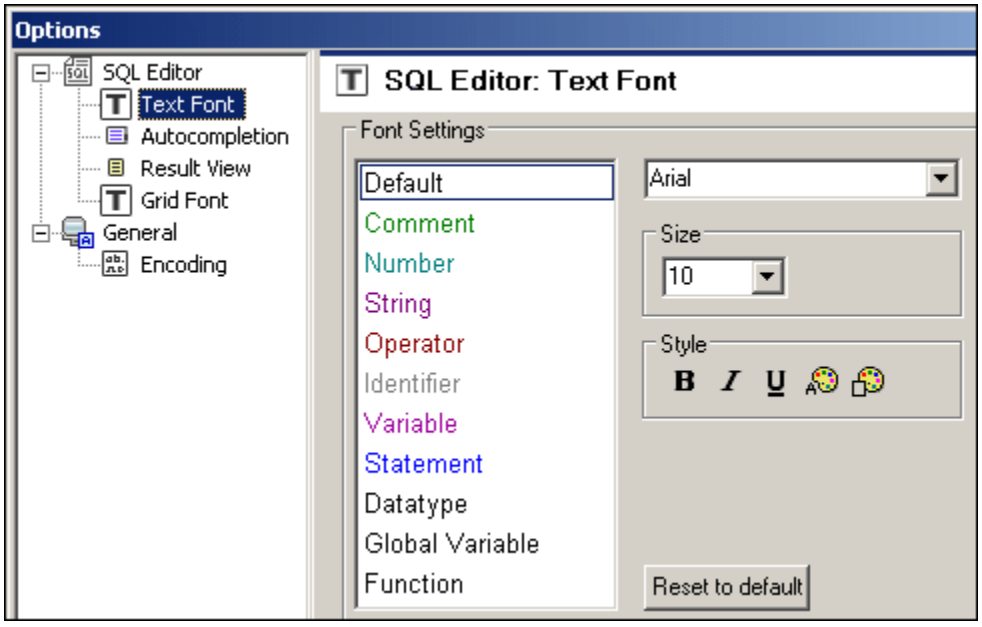
Retrieval
The number entered into the **Buffered Amount(Rows)** field defined the number of rows that initally appear in the Result window.

Specify the maximum amount of time permissible for SQL execution, in seconds.

Activating the **Show timeout dialog** check box, allows you to change the time-out settings when the permissible execution period is exceeded.

Text font

The Text Font section of the **Options** dialog box lets you configure color and font settings of different parts of SQL statements.

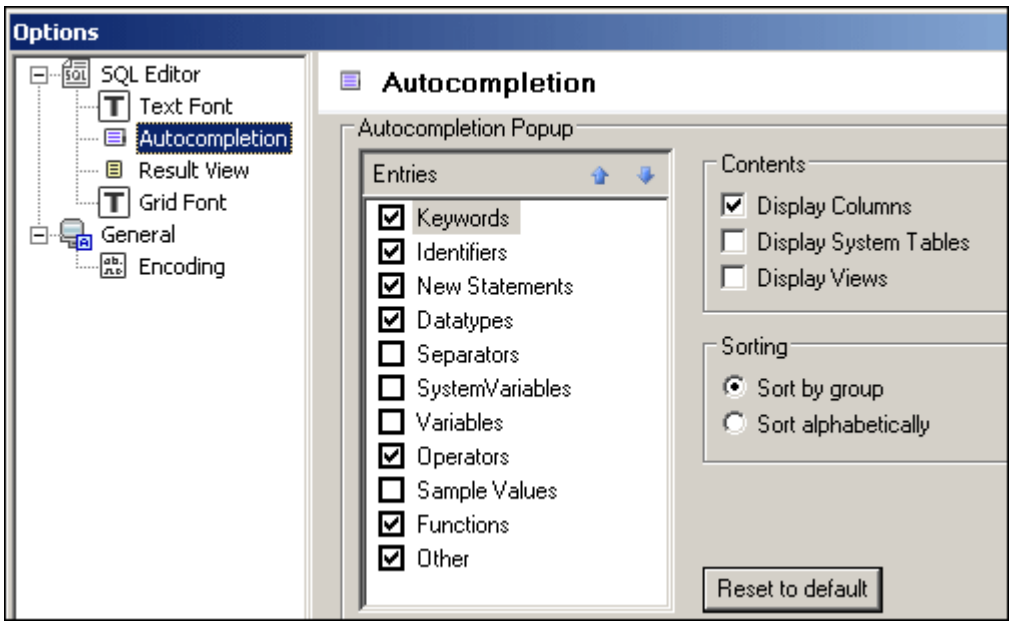


The font settings listed in the Font Settings list box are elements of SQL statements. You can choose the common font face, style, and size of all text that appears in SQL Editor. Note that the same font and size is used for all text types.

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.

Autocompletion

The Autocompletion section of the **Options** dialog box lets you configure the content and appearance of the autocompletion popup. The activation of the Autocompletion function is made in the SQL Editor tab.



Autocompletion Popup
Use the check boxes to specify which categories of entries will appear in the autocompletion drop-down list in the SQL Editor window.

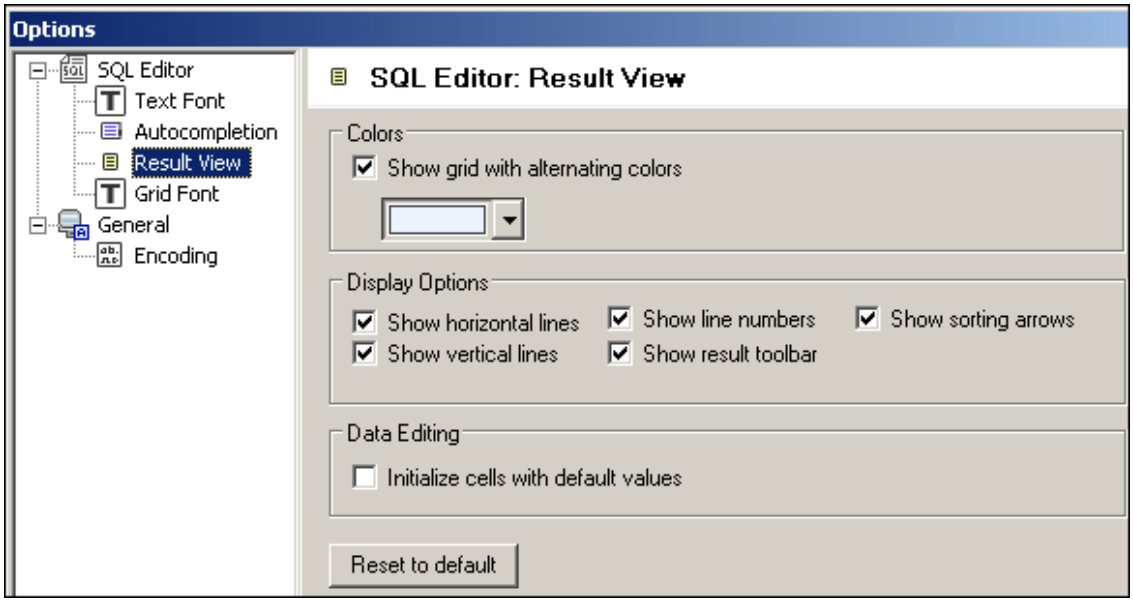
The **Contents** group lets you define whether columns, system tables, or views should be displayed in the autocompletion popup.

The **Sorting** group allows yot to choose whether you want the entries to appear sorted by **group** or **alphabetically**.

Click the **Reset to default** button to restore the original settings.

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.



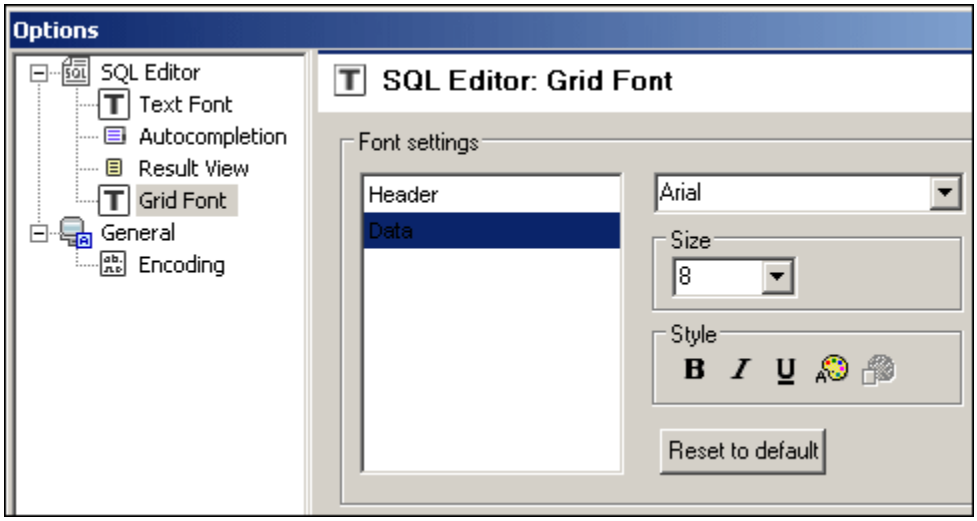
Colors
You can display rows in Result tabs as simple grid or with alternating white and colored rows.

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.

Click the **Reset to default** button to restore the original settings.

Grid font

In the Grid Font section of the **Options** dialog box you can define the appearance of the Result tab of SQL editor.

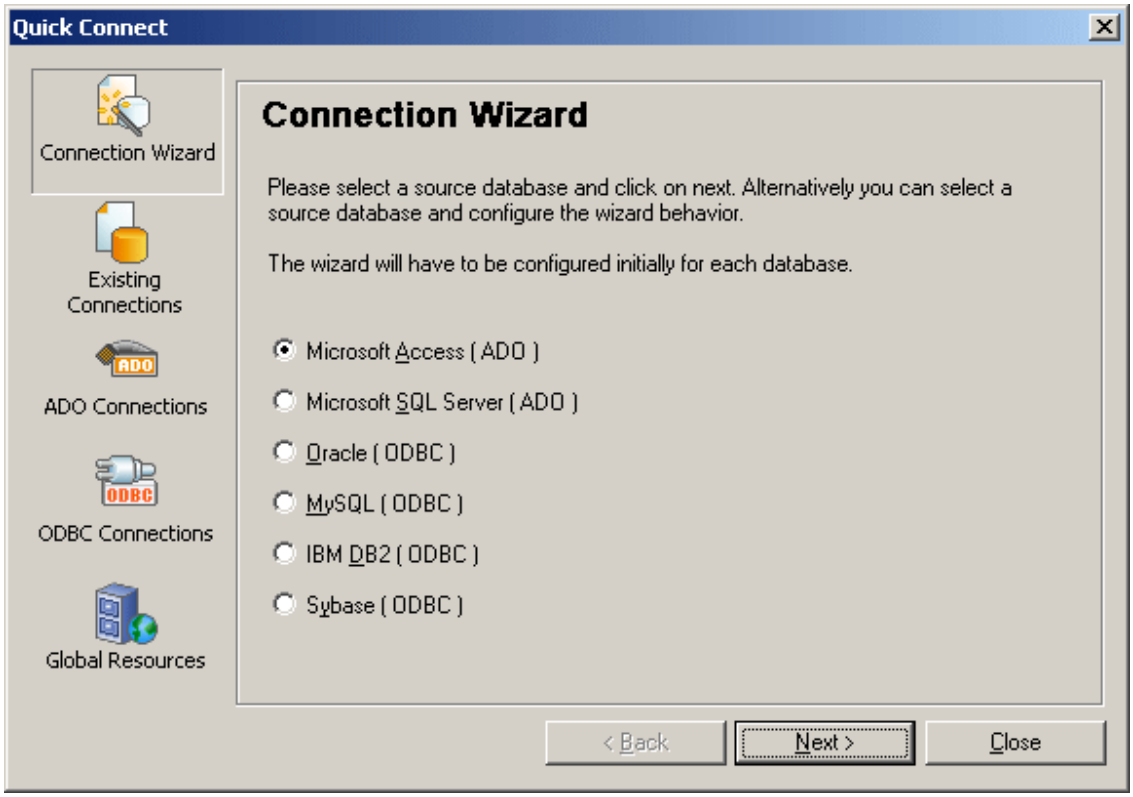


You can choose the font face, size, and style for the header row and the data rows in the Result tabs.

Note that, unlike in the Text Font section, it is possible to specify different font faces for header and data rows. Click the **Reset to default** button to restore the original settings.

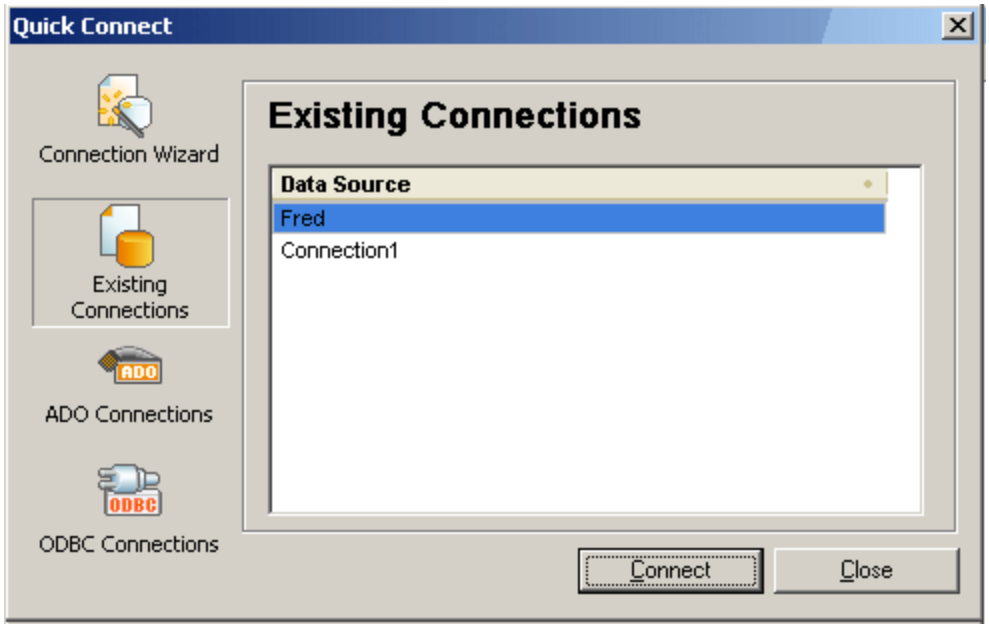
8.13.8 Using the Connection Wizard

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.

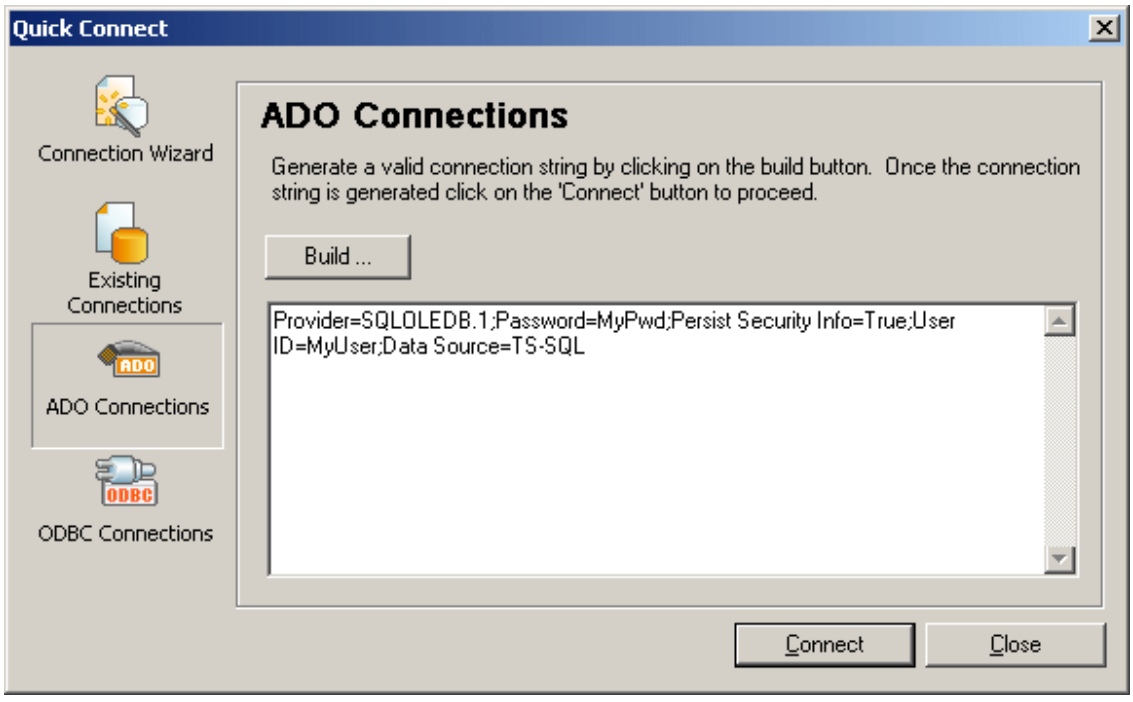
- Select a connection in the list and click **Connect** to connect to that database.



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.

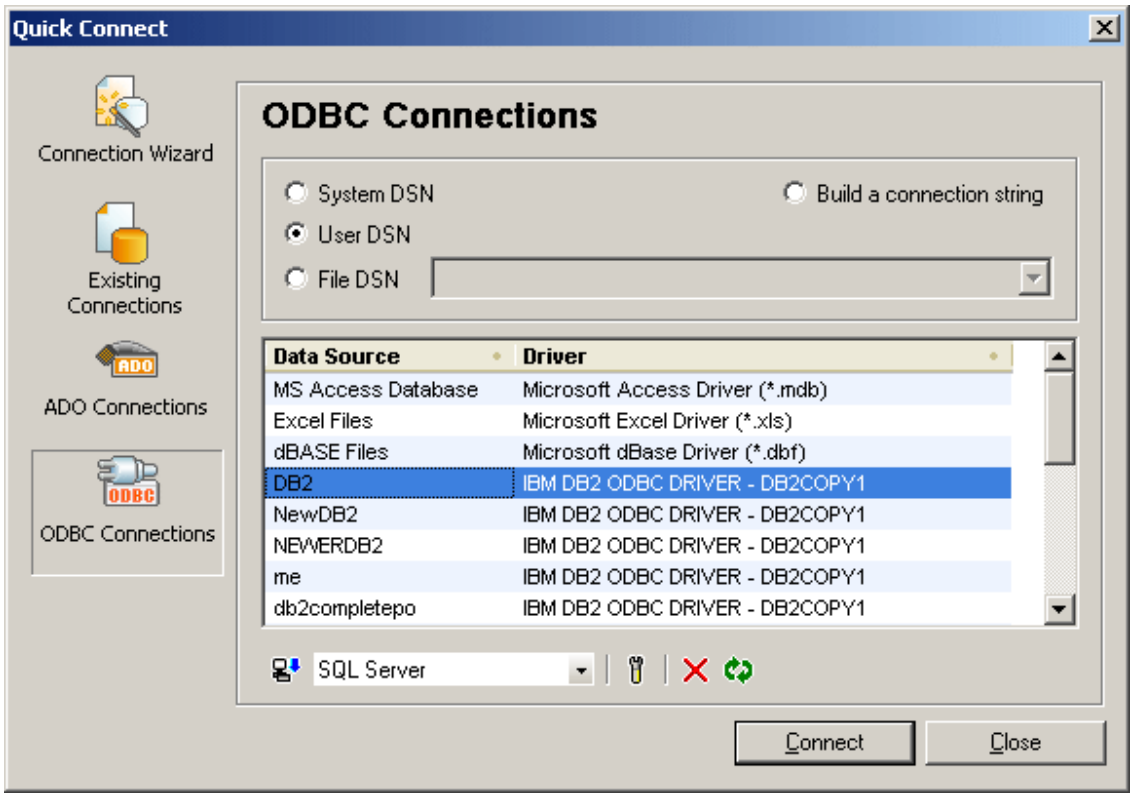
  An ADO (ActiveX Data Objects) connection can be created without carrying out any preliminary steps, such as creating a DSN. Always use an ADO connection for MS Access databases, as ODBC does not support relationships.



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.

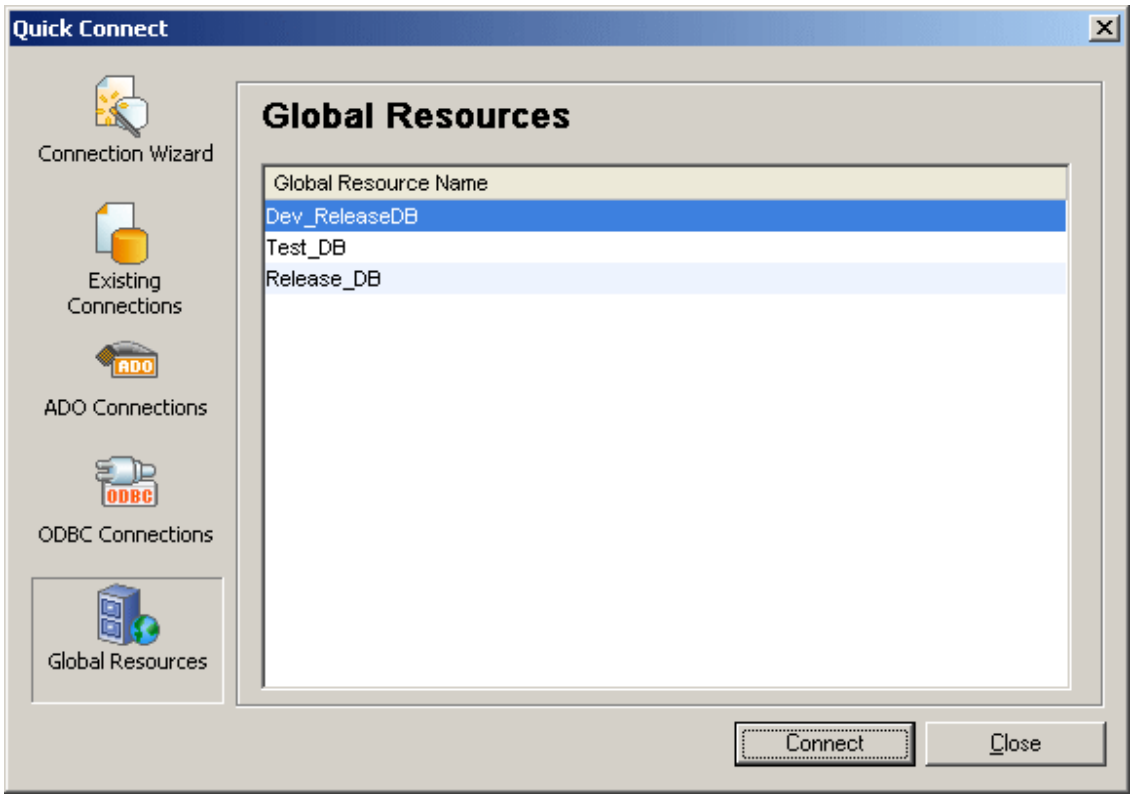
Driver list used to define a new DSN when clicking the **Create New DSN** icon to the left of the combo box.

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.

The **Global resources** pane allows you to select from previously defined global resource database aliases. Please see: Global Resources - Databases for more information.



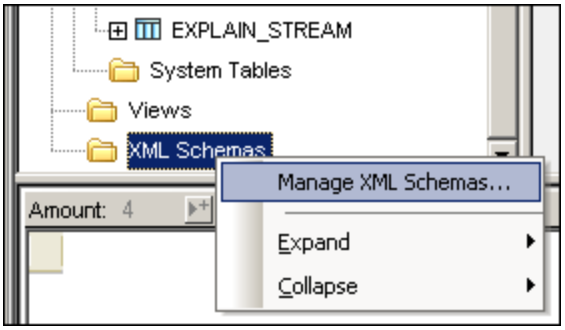
8.13.9 Managing XML Schemas

XML Schemas can be added to and dropped from individual database schemas in an IBM DB2 database. To manage schemas, you need to be connected to the IBM DB2 database, then select the database schema/s for which XML Schemas need to be added or dropped.

To start the connection process, click the Quick Connect icon  in the dialog. This pops up the Quick Connect dialog, through which you can make the connection to the database.

Connecting to the IBM DB2 database

1. Right click the XML Schemas folder in the Browser view and select **Manage XML Schemas**.

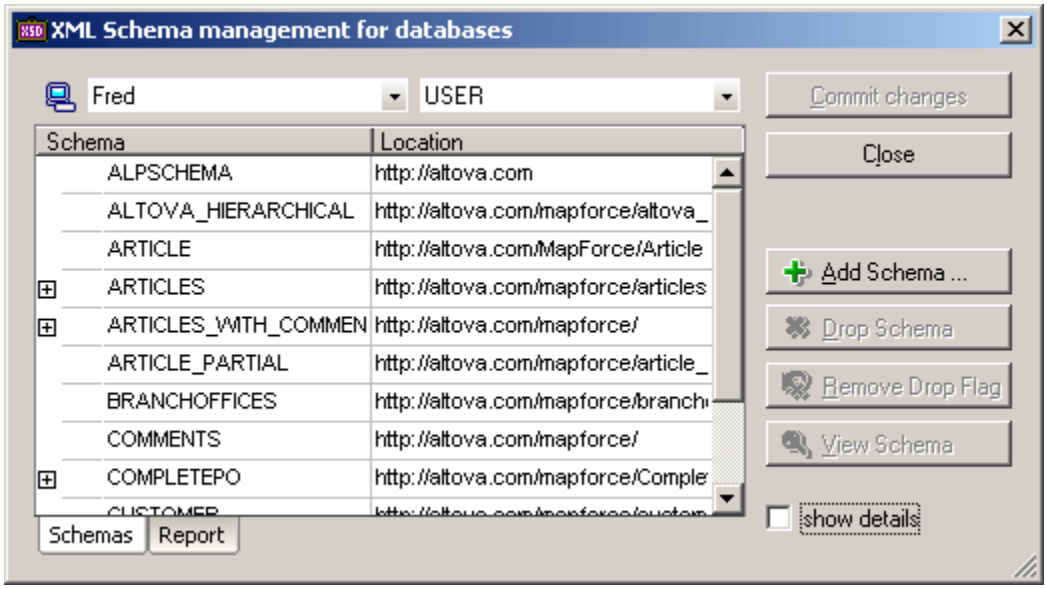


Please note:

Right clicking a schema in the XML Schemas folder allows you to view its contents by selecting **View in XMLSpy**.

Displaying list of XML Schemas

After the connection to the IBM DB2 database has been established, the database is listed in the combo box at left. If more than one connection is currently active, you can select the required database in this combo box.



The combo box at right lists all the database schemas of the currently selected IBM DB2 database. When a database schema is selected in this combo box, all the XML Schemas registered for the selected database schema are displayed in the main pane. In the screenshot

above, all the XML Schemas registered with the `Altova_User` database schema are listed, together with their locations. Checking the **Show Details** check box displays additional information.

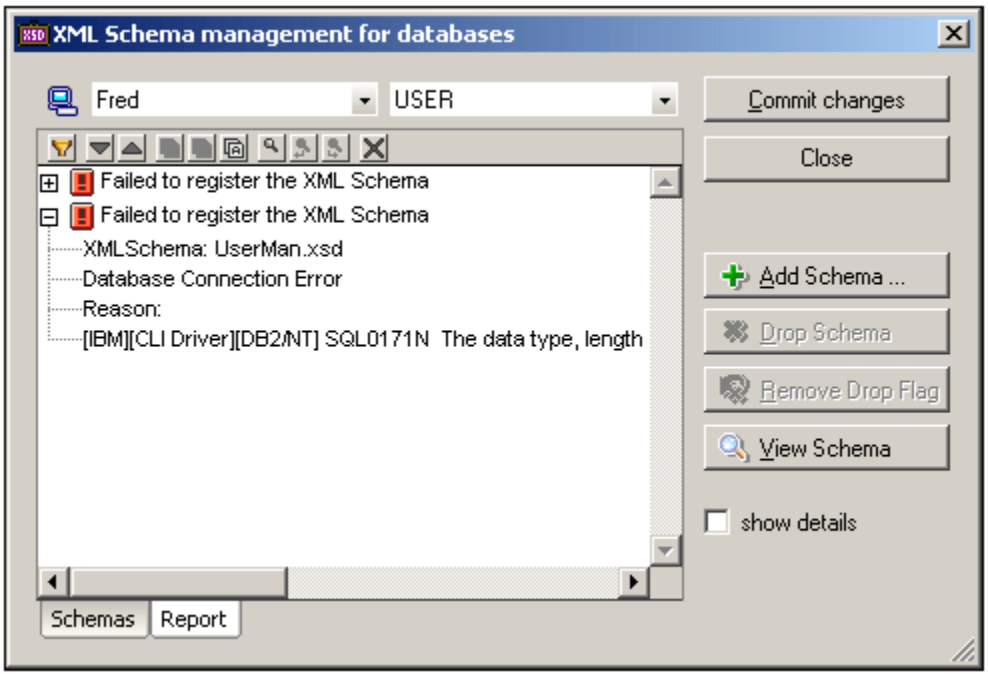
Managing XML Schemas
The list of schemas in the main pane represents the schemas registered for the selected database schema. After the list of XML Schemas is displayed, you can add schemas to the list or drop (delete) schemas from the list.

To **add** a schema, click the **Add** button, browse for the required schema file, and select it. The selected schema file is added to the list in the main pane. Clicking the **Commit Changes** button registers the newly added schema with the database schema.

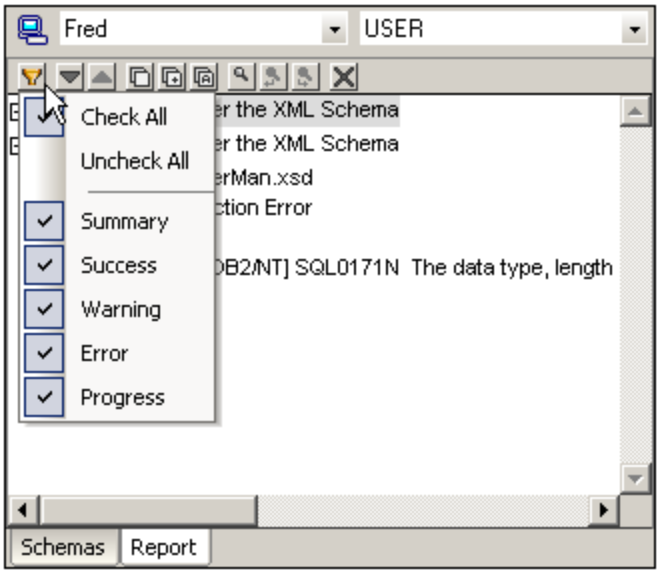
To **drop** a schema, select the schema in the main pane and click the **Drop Schema** button. A Drop Flag is assigned to the schema, indicating that it is scheduled to be dropped when changes are next committed. The Drop Flag can be removed by selecting the flagged schema and clicking the **Remove Drop Flag** button. When the **Commit Changes** button is clicked, all schemas that have been flagged for dropping will be unregistered from the database schema.

Clicking the View Schema button opens the schema in XMLSpy. To close the XML Schema Management dialog, click the **Close** button.

Reports
When the **Commit Changes** button is clicked, the database is modified according to the changes you have made. A report of the Commit action is displayed in the Report pane, enabling you to evaluate the success of the action and to debug possible errors. Each subsequent report is displayed below the previous one.



The report pane has a toolbar containing icons that enable you to customize the display of the report listing, navigate the listing, copy report messages, search for text, and clear the pane.

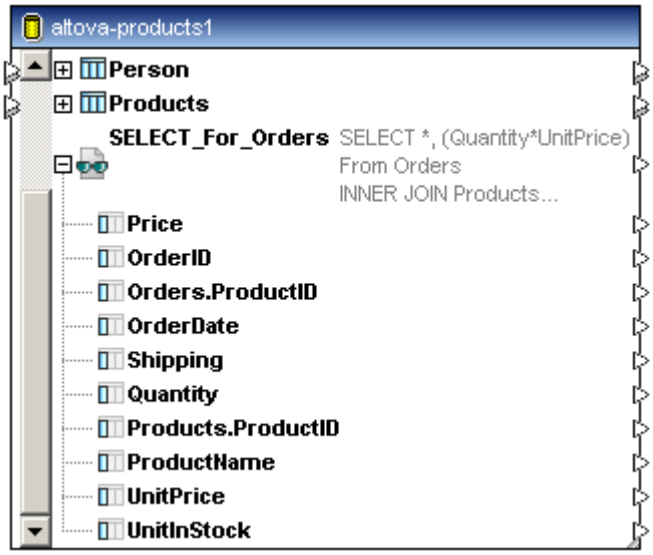


The **Filter** icon enables the display of particular types of messages to be toggled on or off. The **Next** and **Previous** icons move the selection down and up the list, respectively. Messages can also be copied with or without their child components to the clipboard, enabling them to be pasted in documents. The **Find** function enables you to specify a search term and then search up or down the listing for this term. Finally, the **Clear** icon clears the contents of the Report pane.

8.14 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.



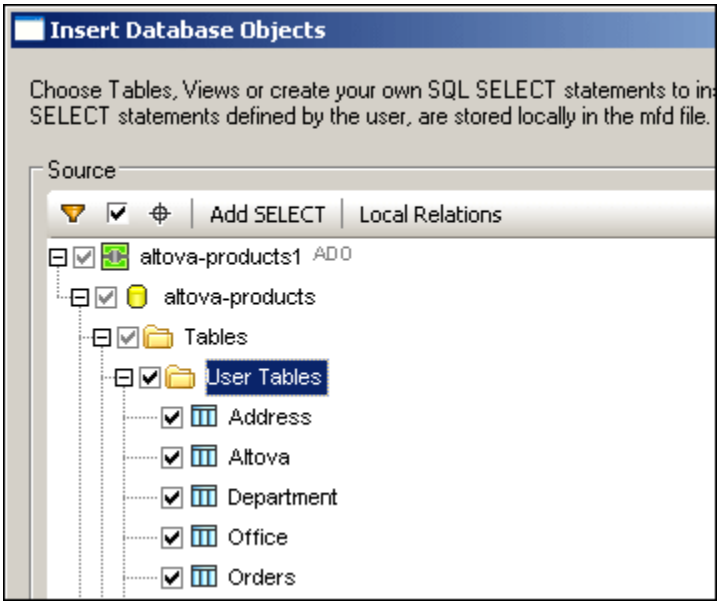
This section uses the **altova-products.mdb** file available in the **C:\Documents and Settings \<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. The **select-component.mfd** mapping, which shows an example is also available in the same folder.

8.14.1 Creating SELECT statements

This section uses the **altova-products.mdb** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.

To create a SELECT statement in a database component:

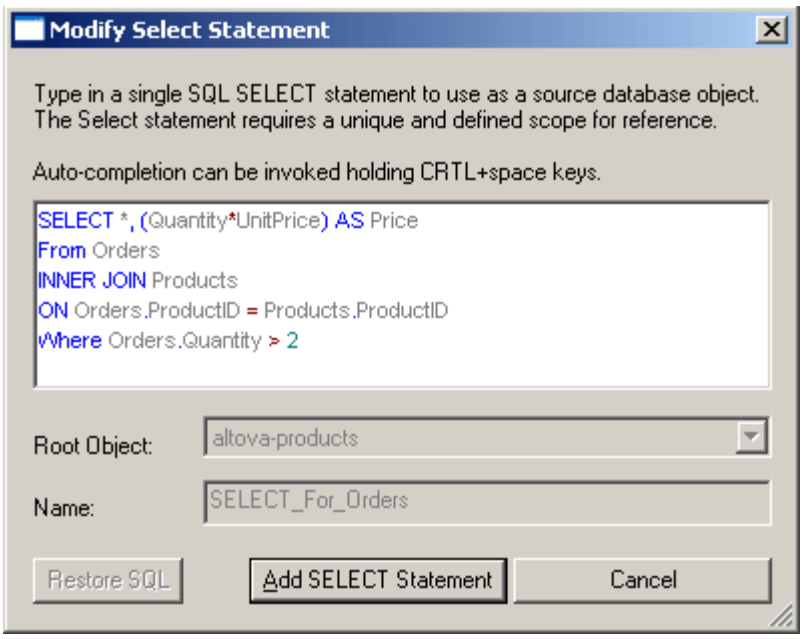
1. Click the **Insert Database** icon in the icon bar.
2. Click the **Microsoft Access** radio button, then click Next.
3. Click the Browse button to select the database you want as the data source, **altova-products.mdb** in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder in this case, continue with Next.
4. Click the User tables check box to select all tables of the database.



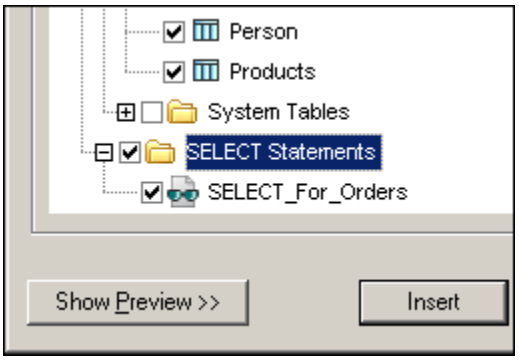
5. Right click the **Orders** table and select "Generate and add an SQL statement" from the popup menu, (or click the Add SELECT button in the icon bar).
   A default SELECT statement is already available in the dialog box.
6. Delete or edit the statement to suit you needs. The example uses the following 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.



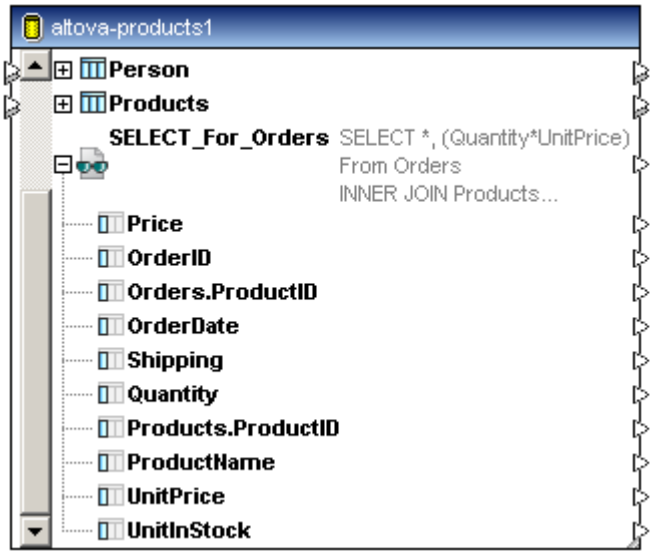
The "virtual table" has now been addded to the SELECT Statements folder in the dialog box.

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 next to the **Select_For_Orders** item.



The column items of both tables defined by the inner join are available to be mapped.

Please note:

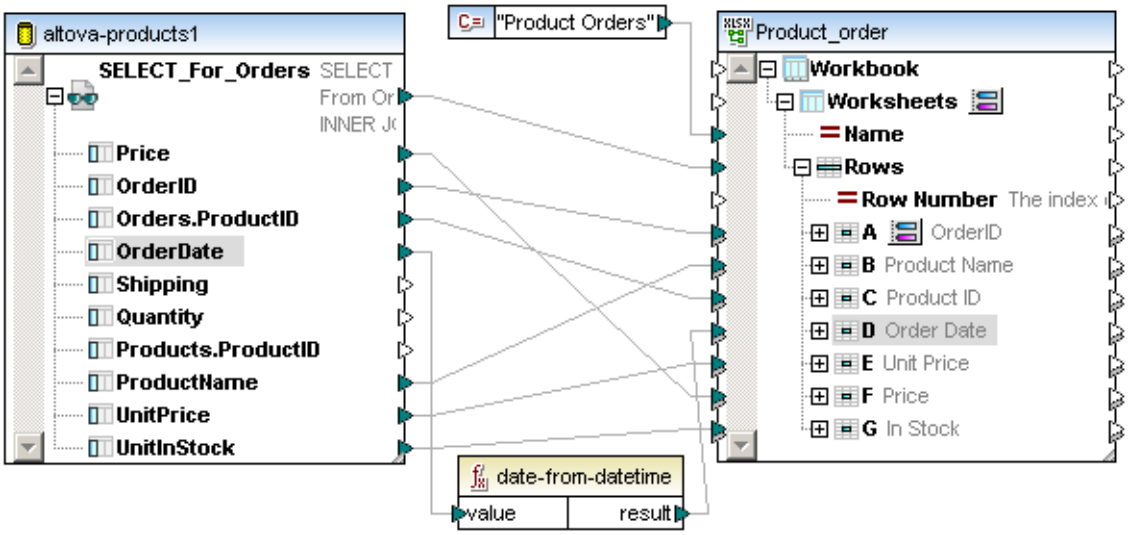
The **Price** field is the product of the two fields, Quantity and UnitPrice, and is actually a correlation name: SELECT *, (Quantity*UnitPrice) AS Price.

8.14.2 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.**

The aim of this mapping is to output the specific order data from the database and map it to an OOXML Excel 2007 spreadsheet.

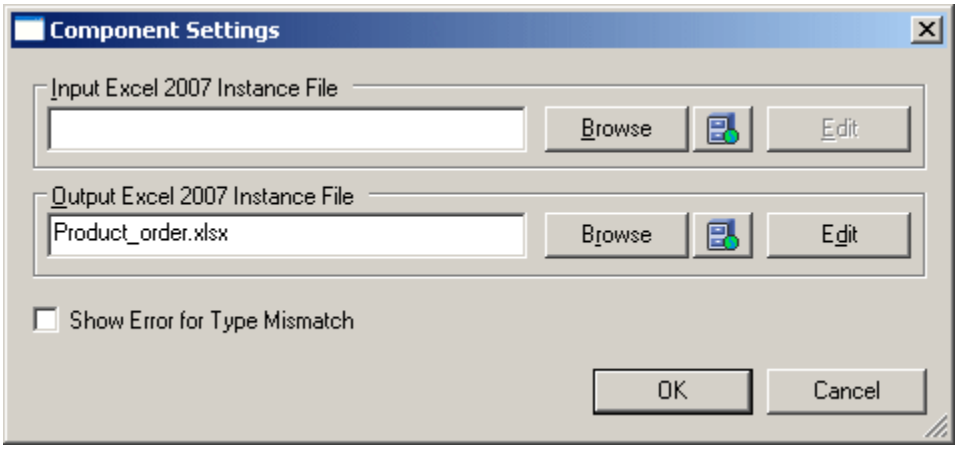
Open the **select-component.mfd** mapping, available in the **C:\Documents and Settings \<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.



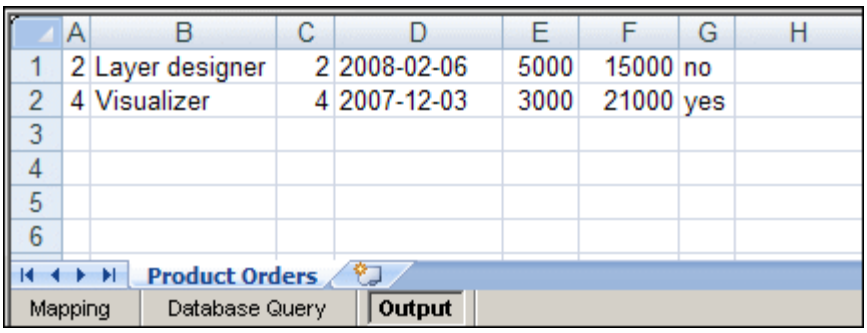
What this mapping does:

- Creates a Select statement component "altova-products" using an Inner join statement.
- Maps the database data to an empty Excel 2007 template file. For more information on how to map to/from Excel 2007 spreadsheets, please see Mapping MS OOXML Excel 2007 files.

  Note that double clicking the Excel component title bar allows you to define the input, or output file names of the Excel component.



- The **date-from-datetime** functions converts the database date format to the simple date format.
- Clicking the Output button shows you a preview of the contents of the Excel file.



	A	B	C	D	E	F	G	H
1	2	Layer designer	2	2008-02-06	5000	15000	no	
2	4	Visualizer	4	2007-12-03	3000	21000	yes	
3								
4								
5								
6								

Product Orders

Mapping | Database Query | Output

Chapter 9

Mapping CSV and Text files

9 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++) 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
- UN/EDIFACT and ANSI X12 to/from flat file formats or databases
- A FlexText template which defines file structure and content, defined in the FlexText module, and is inserted as a component into a mapping. Please see MapForce FlexText for more information.

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 MapForce 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:

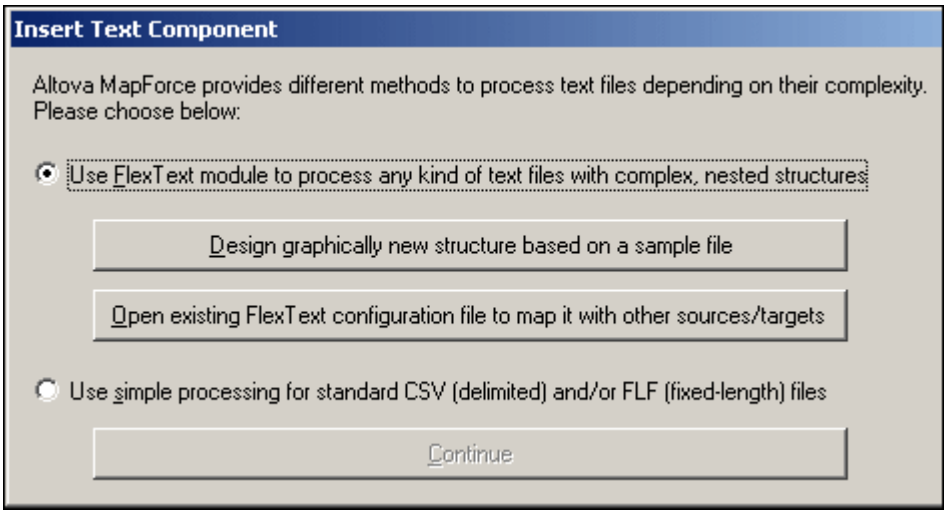
All the following examples using CSV files as source or target components, can also be accomplished with Fixed length text files. The only difference is that the field lengths have to be defined manually, please see "Mapping Fixed Length Text files" on how define field lengths.

9.1 Mapping CSV files to XML

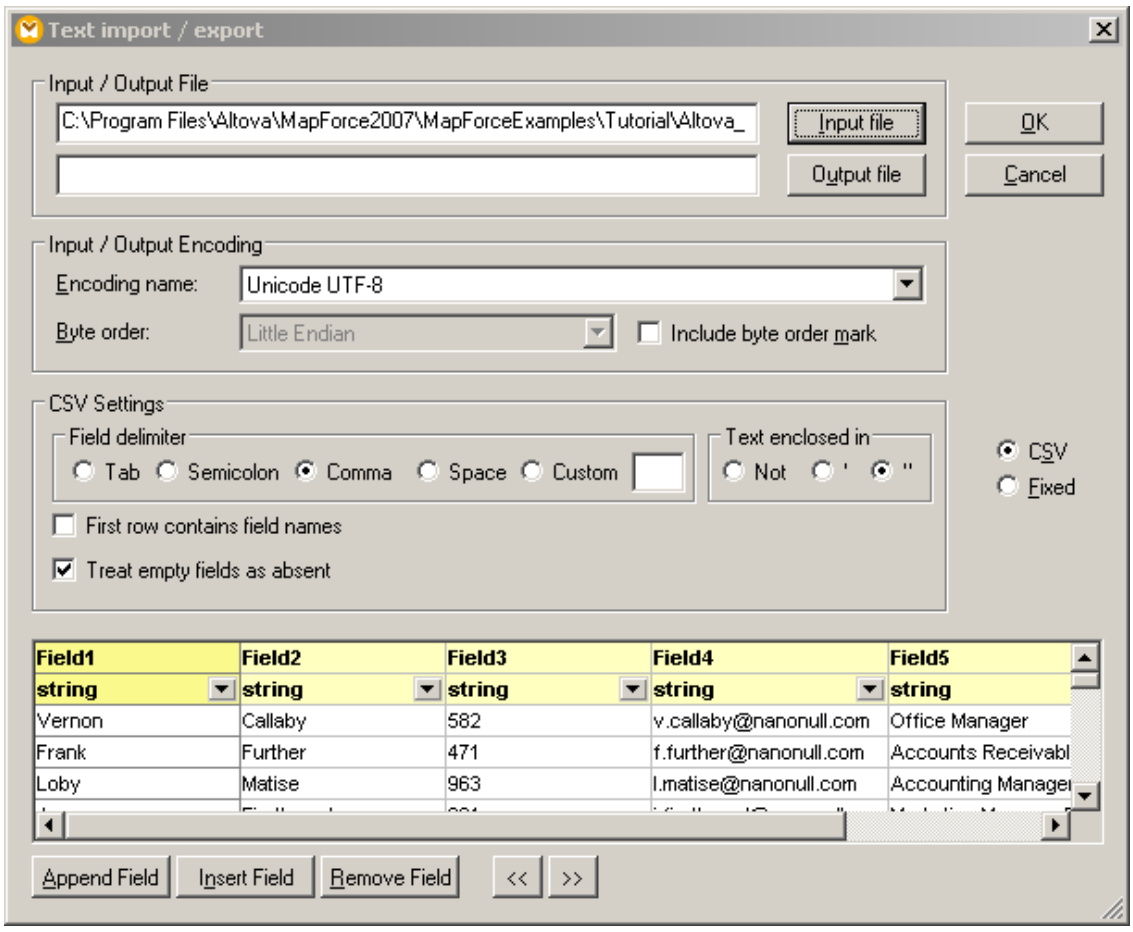
This example maps a simple CSV file to an XML file, based on the MFCompany.xsd schema file. All the files used in the following examples are available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.

- Having made sure you selected one of the programming languages, **Java**, **C#**, or **C++**, by clicking the respective toolbar icon.

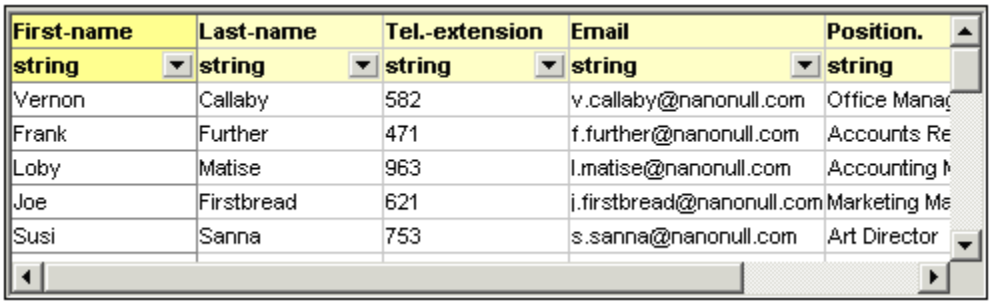
1. Select the menu option **Insert | Text file**, or click the "Insert Text file" icon  This opens the "Insert Text Component" dialog box.



Click the **Use simple processing** ... radio button and click the **Continue** button. 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.

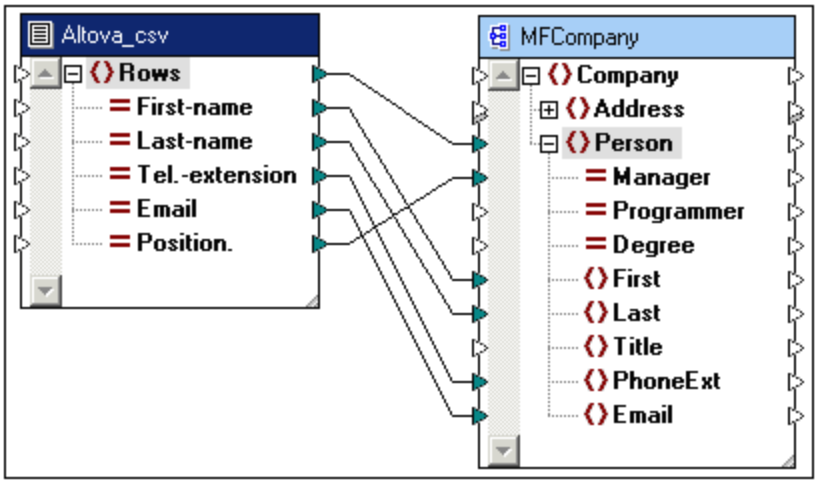


2. Click the **Input file** button and select the CSV file, e.g. Altova_csv.csv. The file contents are now visible in the Preview window. Please note that the Preview window only displays the first 20 rows of the text file.
3. Click into the Field1 header and change the text, e.g. First-name. Do the same for all the other fields, e.g. Last-name, Tel.-extension, Email, and Position.



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.

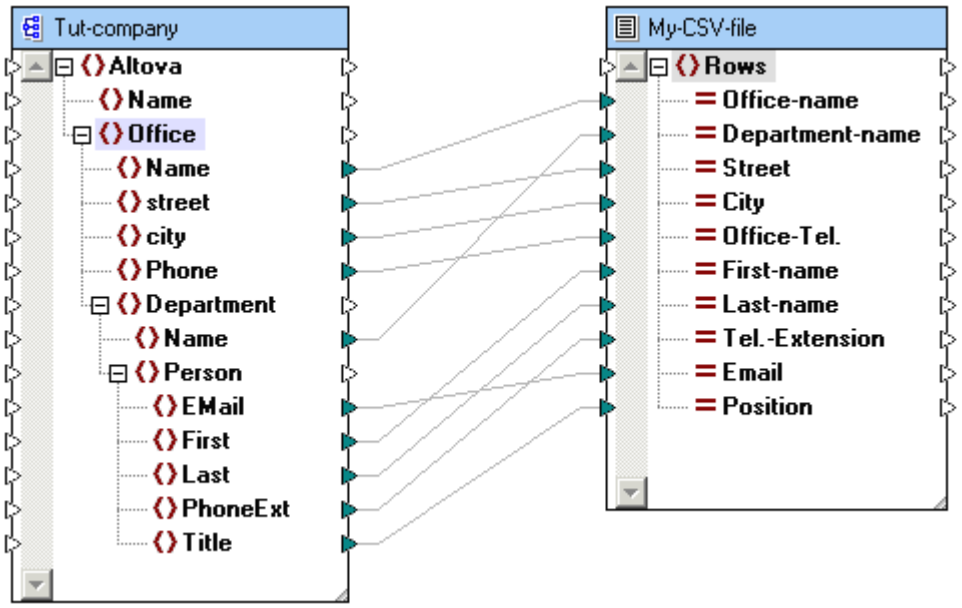
9.2 XML to CSV, iterating through items

This example is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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.

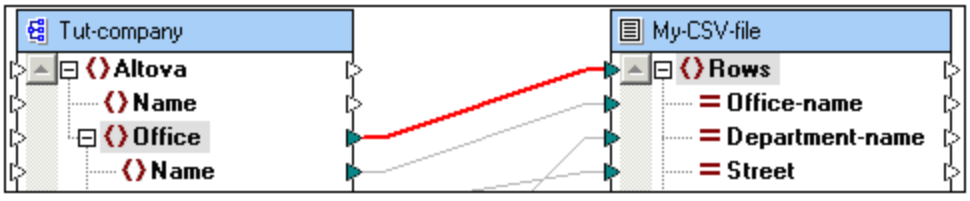


Clicking the Output tab produces the result you see below, which may not be what you expect, we only see output for one office.

```
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.

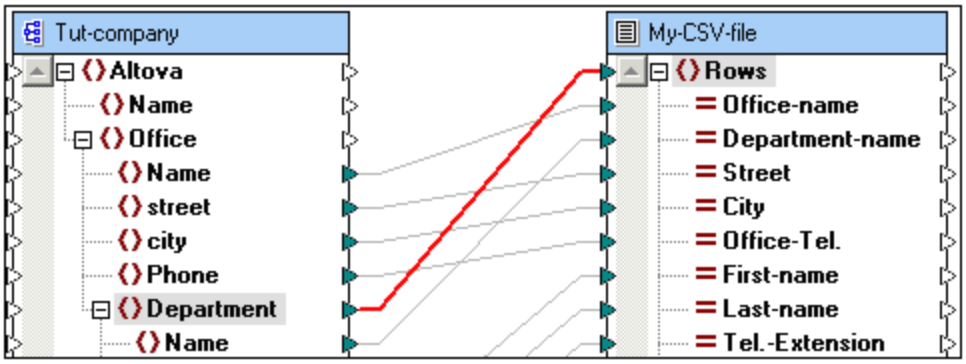
Mapping the **Office** item to the **Rows** item, results in all individual Offices (and mapped items) being output.



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
```

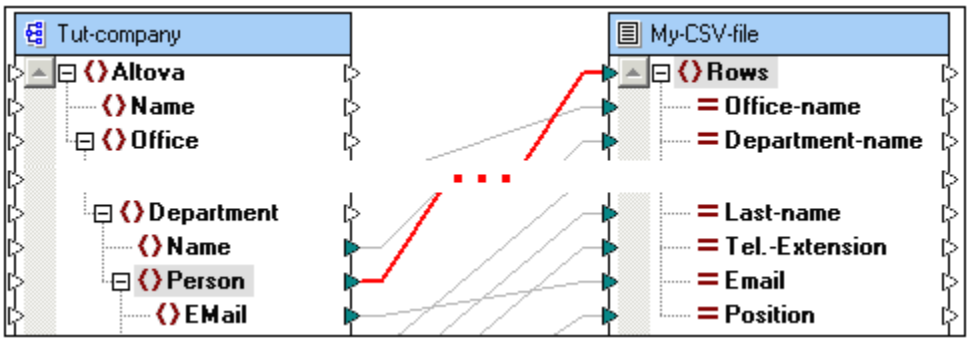
Mapping **Department** to the **Rows** item results in all of the Departments being output.



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
```

9.3 Creating hierarchies from CSV and fixed length text files

This example is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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:

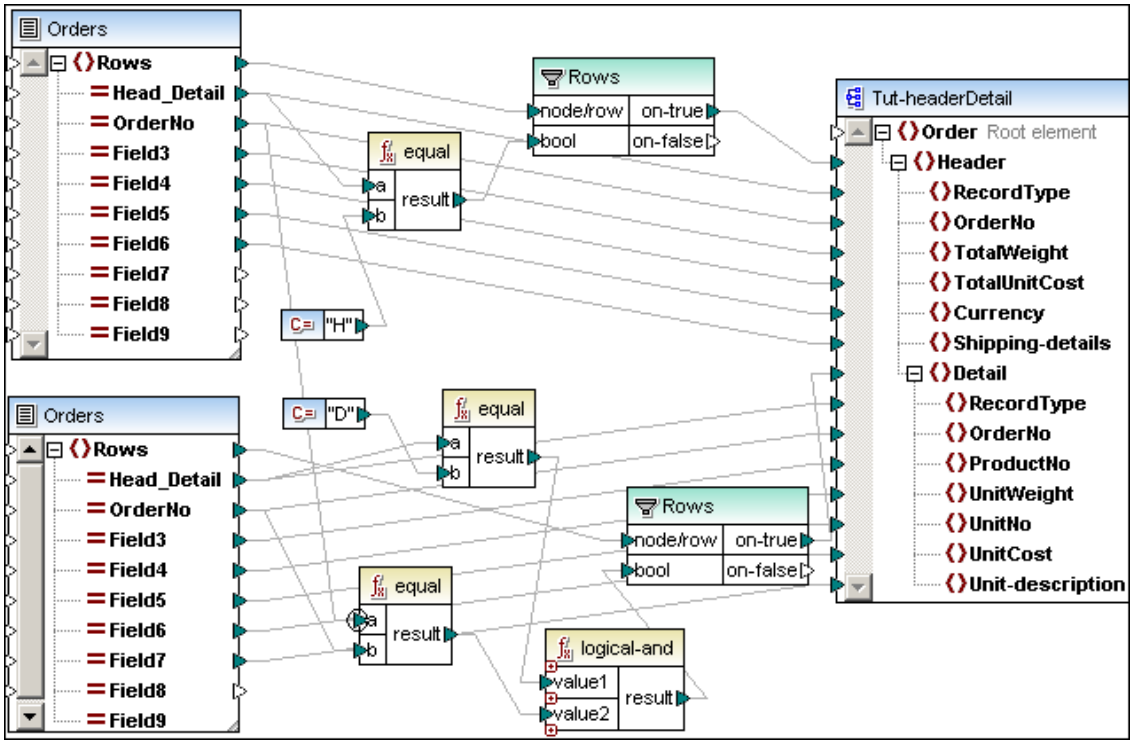
- Field 1: H defines a header record and D a detail record.
- Field 2: A common/key for both header and detail records.
- Each header/detail record is on a separate line.

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.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation
  <Header>
    <RecordType>H</RecordType>
    <OrderNo>111</OrderNo>
    <TotalWeight>332.1</TotalWeight>
    <TotalUnitCost>22537.7</TotalUnitCost>
    <Currency/>
    <Shipping-details>Container ship</Shipping-details>
    <Detail>
      <RecordType>D</RecordType>
      <OrderNo>111</OrderNo>
      <ProductNo>A-1579-227</ProductNo>
      <UnitWeight>10</UnitWeight>
      <UnitNo>3</UnitNo>
      <UnitCost>400</UnitCost>
      <Unit-description>Microtome</Unit-description>
    </Detail>
    <Detail>
      <RecordType>D</RecordType>
      <OrderNo>111</OrderNo>
      <ProductNo>B-152-427</ProductNo>
      <UnitWeight>7</UnitWeight>
      <UnitNo>6</UnitNo>
      <UnitCost>1200</UnitCost>
      <Unit-description>Miscellaneous</Unit-description>
    </Detail>
  </Header>
```

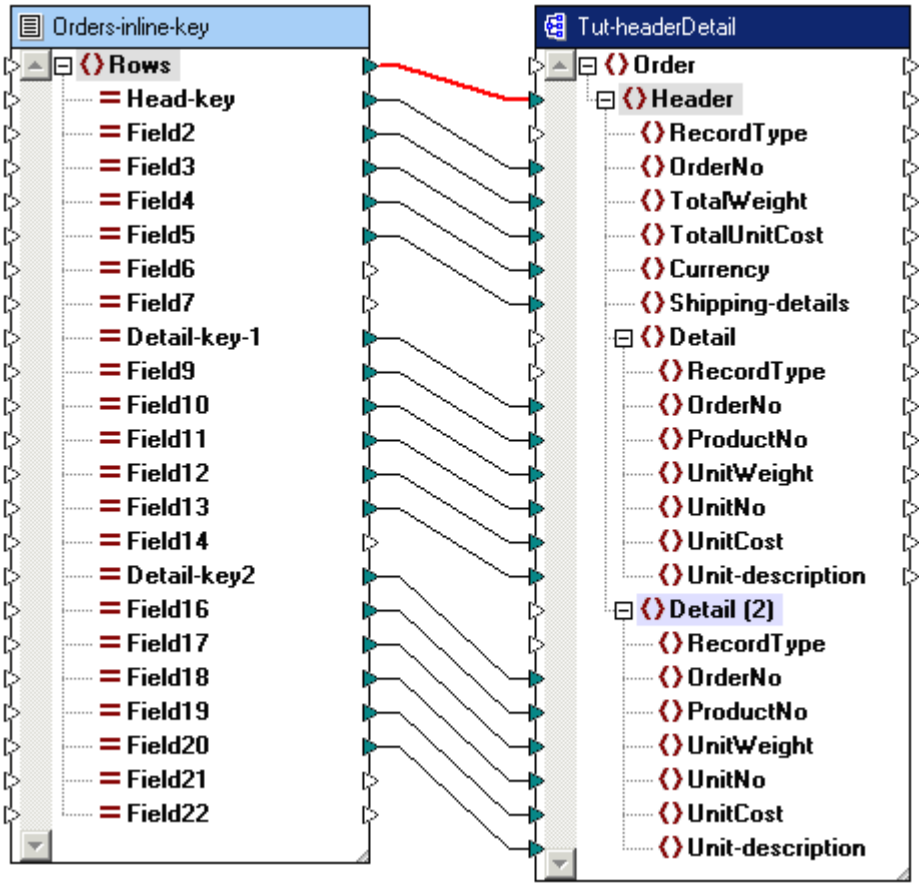

The second example uses a slightly different CSV file and is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder as **Head-detail-inline.mfd**. however:

- 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 exactly the same XML file that was produced in the above example.

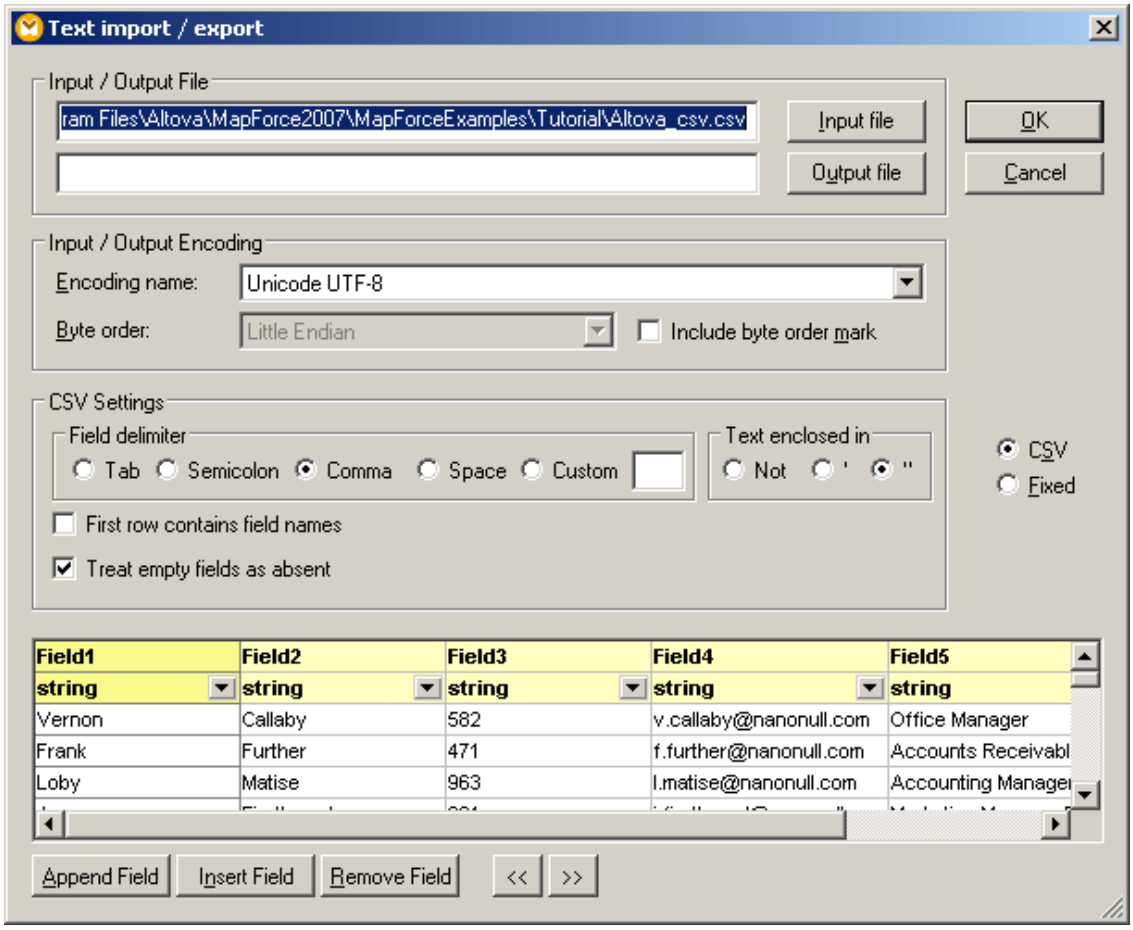


9.4 CSV file options

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 only used when generating code for Java, C++, or C#.

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.

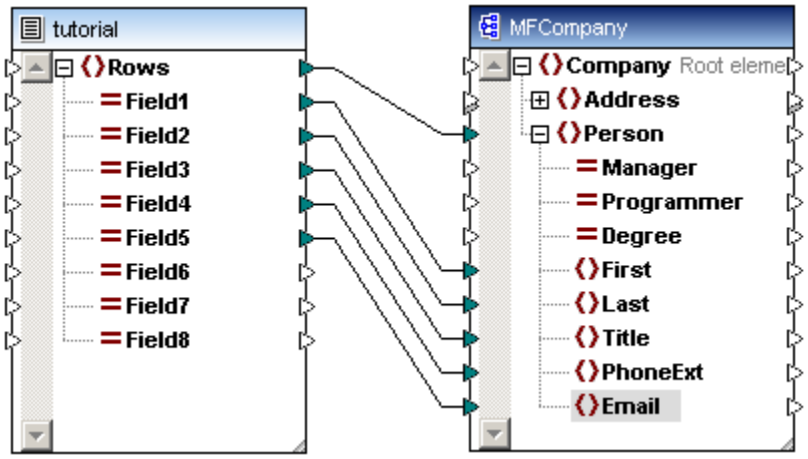
CSV Settings - Field delimiter:
Select the delimiter type for the text file (CSV files are comma delimited "," per default). You can also enter a custom delimiter in the **Custom** field.

Click into the Custom field and:

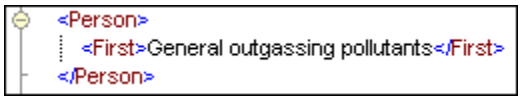
- Hit a keyboard key to enter a new value, or
- Double click in the Custom field, to mark the current value, and hit a different keyboard key to change the entry.

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.

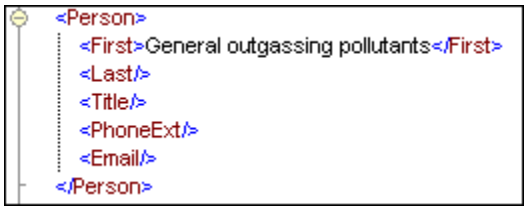
Treat empty fields as absent
Allows you to define that empty fields in the source file, will not produce a corresponding empty item (element or attribute) in the target file.



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.



Inactive:
The empty fields of the source file produce the corresponding target items in the output file, i.e. the elements, Last, Title, PhoneExt 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.

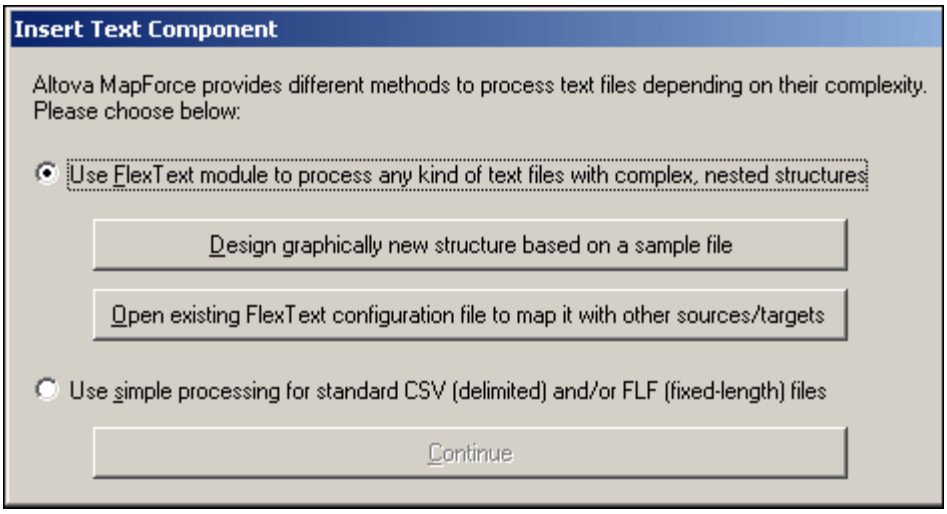
Append field, **Insert** field, **Remove** field:
Allows you to append, insert or remove fields in the preview window, which defines the structure of the CSV file.

Next / **Previous**
Clicking one of these buttons moves the currently active column left or right in the preview window.

9.5 Mapping Fixed Length Text files (to a database)

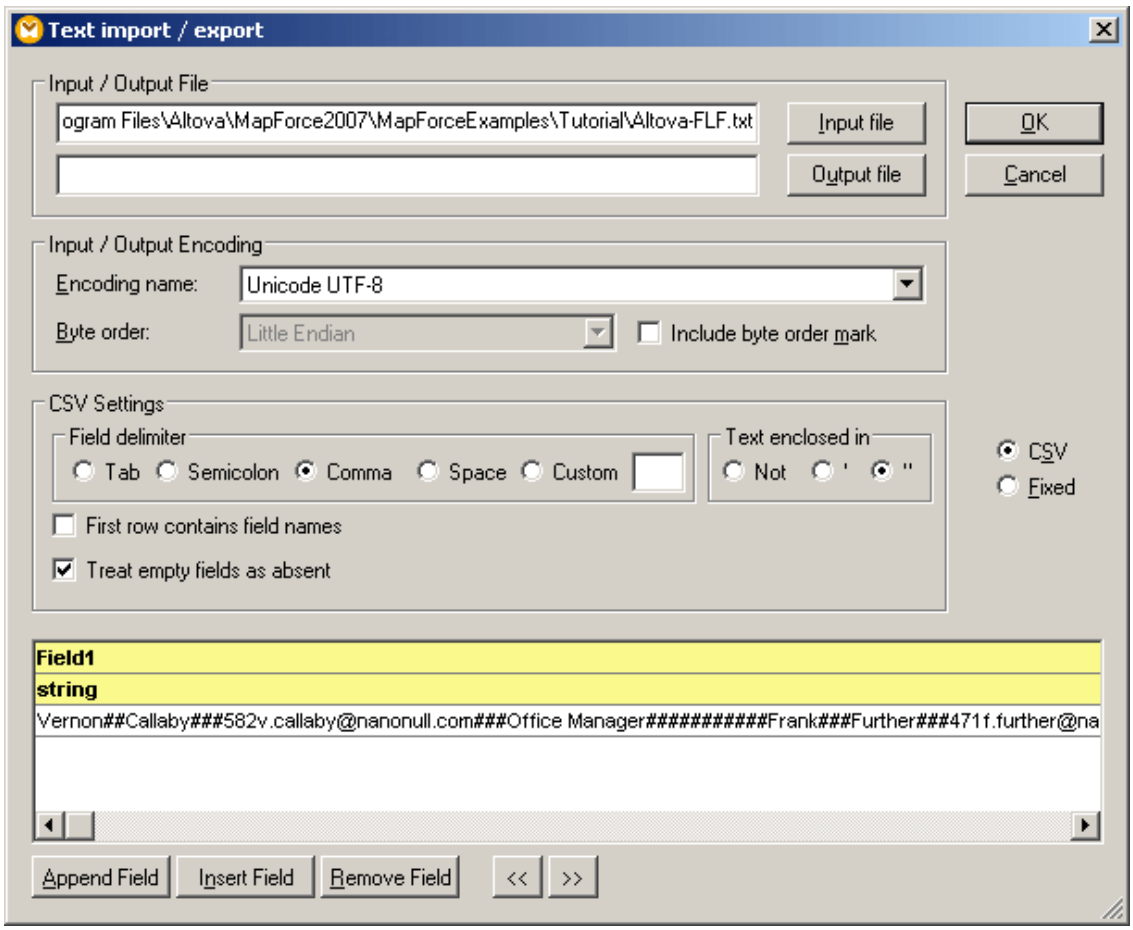
This example maps a simple text file to a MS Access database. The source text file is one continuous string with no carriage returns, or line feeds. All the files used in the following examples are available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.

1. Select the menu option **Insert | Text file**, or click the insert Text file icon  . This opens the "Insert Text Component" dialog box.

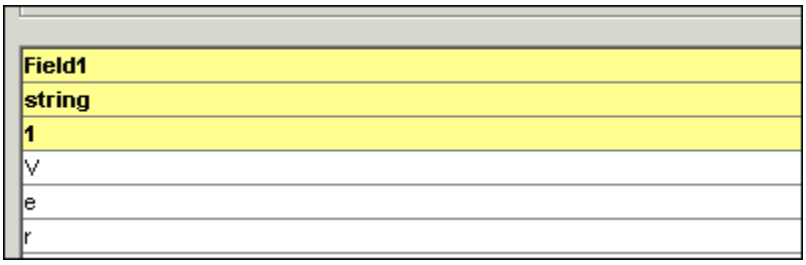


   Click the **Use simple processing** ... radio button and click the **Continue** button.

   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.

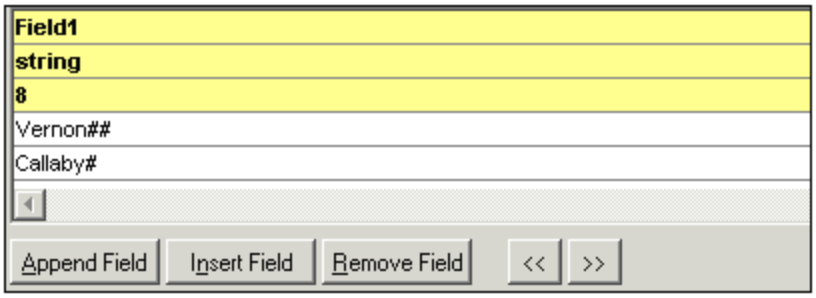


The preview changes at this point. What we now have, is a fixed format comprising of:

- a single field called "Field1"
- where the format is of type "string", and the
- field length is one character (V from person Vernon)

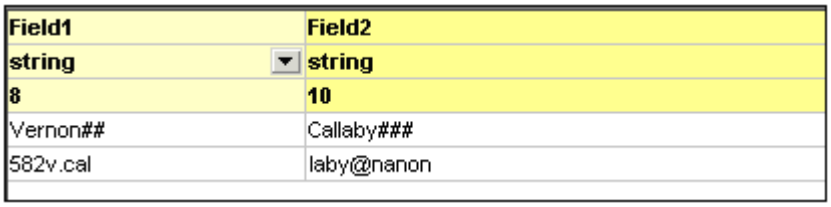
Field 1 now contains additional data.

5. Click into the row containing the **1** character, change the value to 8, and hit Return.

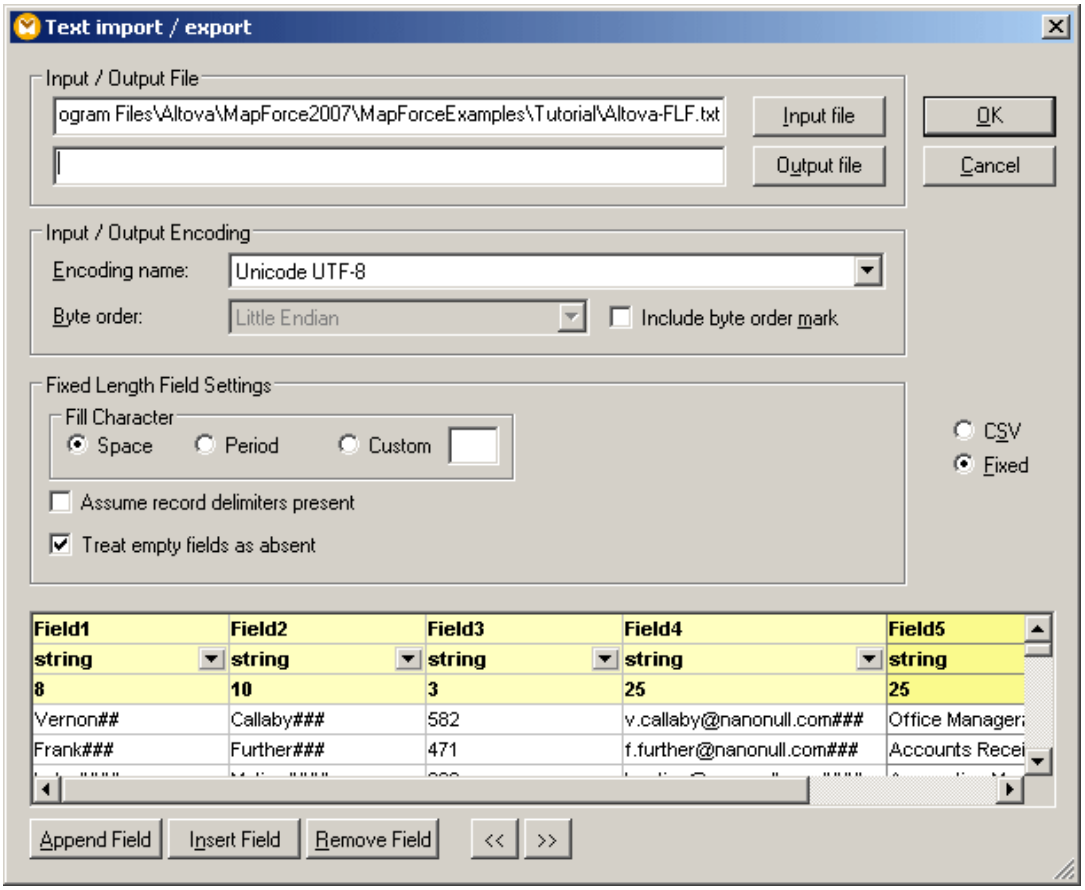


More data is now visible in the first column, which is now defined as 8 characters wide.

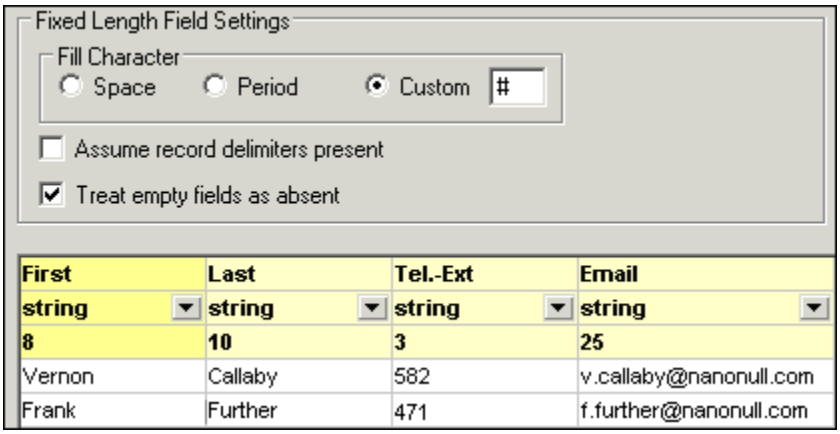
6. Click the **Append Field** button to add a new field, and make the length of the second field, 10 characters.



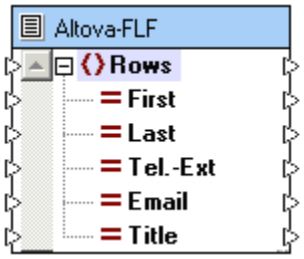
7. Use the same method to create three more fields of the following lengths: 3, 25, and 25 characters, and change the field headers to make them easier to map: First, Last, Tel.-Ext, Email, Title. The preview will then look like this:



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.

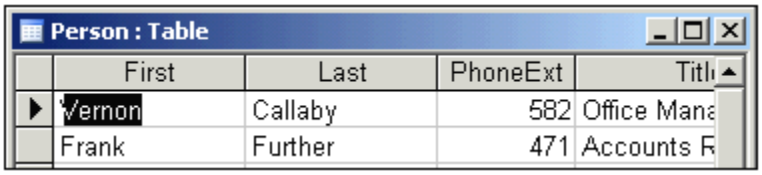


9. Click OK to complete the definition.

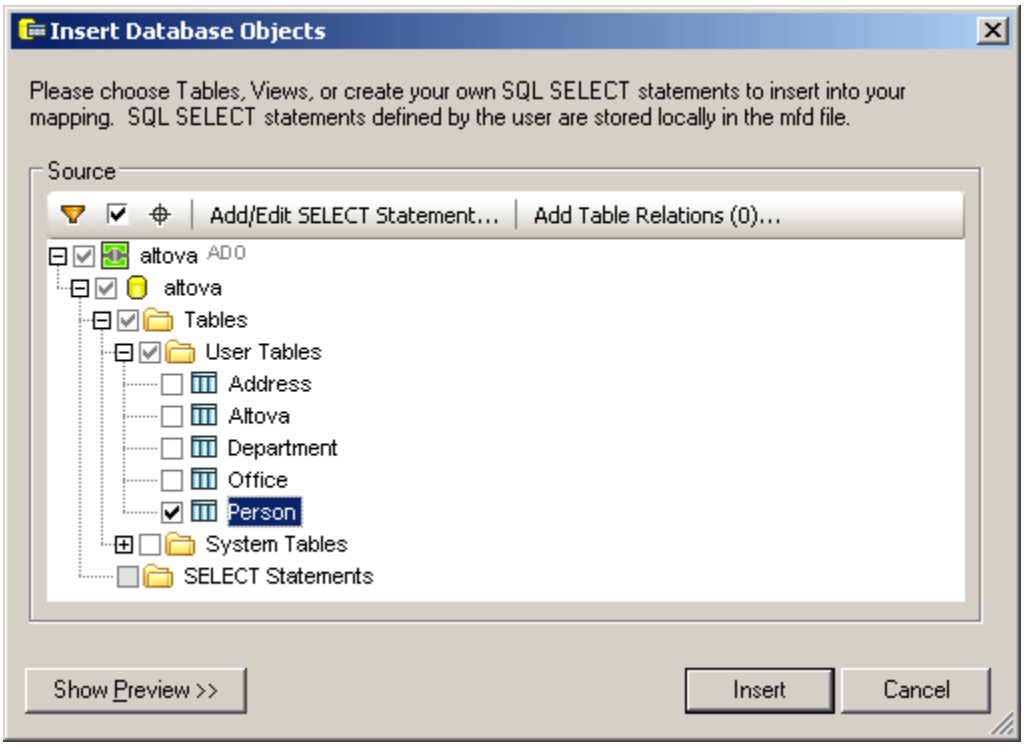


The Text file component appears in the Mapping window. Data can now be mapped to, and from, this component.

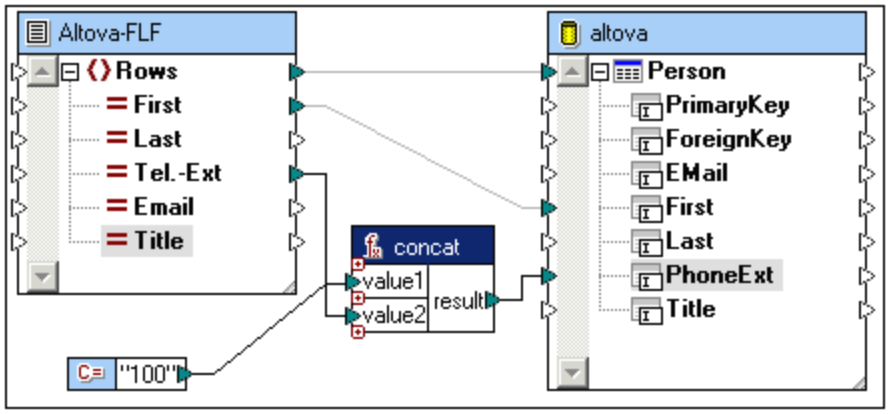
Mapping text files to a database:
This section uses the fixed length text file to update the Telephone extension entries in the **altova.mdb** database.



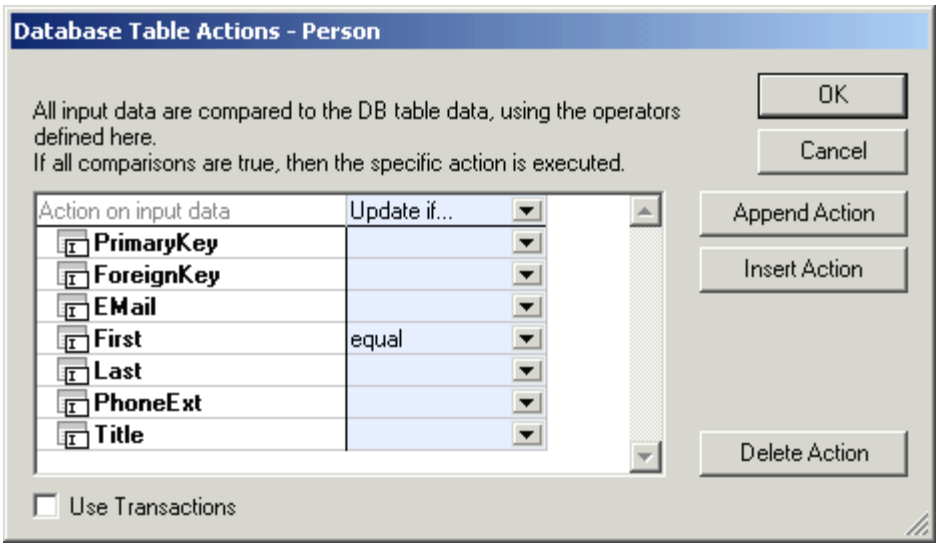
1. Select the menu option **Insert | Database,** click the Microsoft Access radio button, then click Next.
2. Select the **altova.mdb** database available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder, and click Next.
3. Select the **Person** 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.



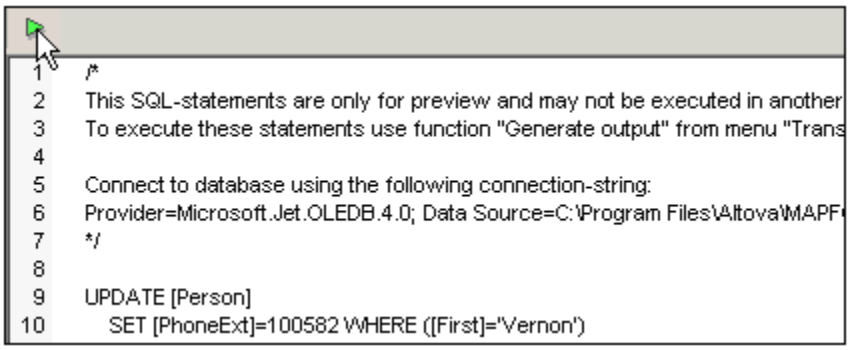
9. Right click the Person entry and select Database table actions.



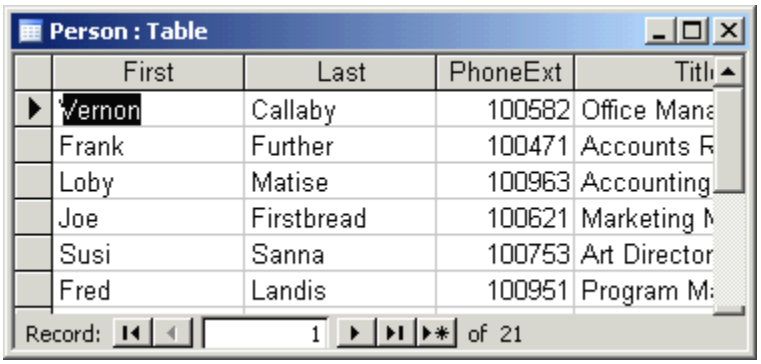
10. Click the "Action on input data" combo box and select the "Update if..." entry.
11. Click the combo box of the "First" row, select "equal", and click OK to confirm.

    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.

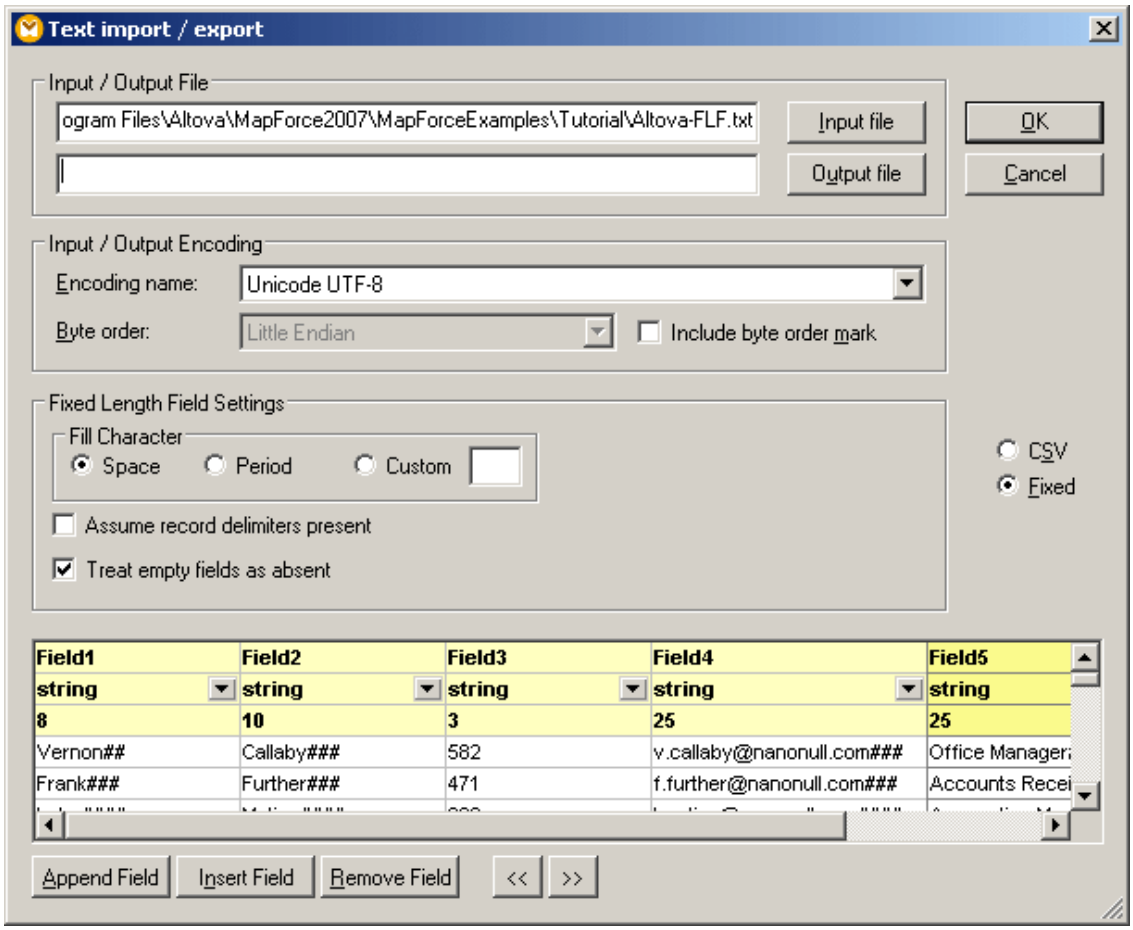


9.5.1 Fixed Length Text file options

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 only used when generating code for Java, C++, or C#.

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.

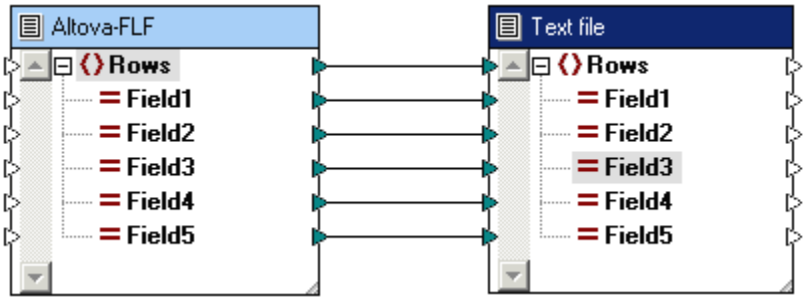
Fill Character
This option allows you to define the characters that are to be used to complete, or fill-in, the rest of the (fixed) field when the incoming data is shorter than the respective field definitions. The custom field allows you to define your own fill character in the Custom field.

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!

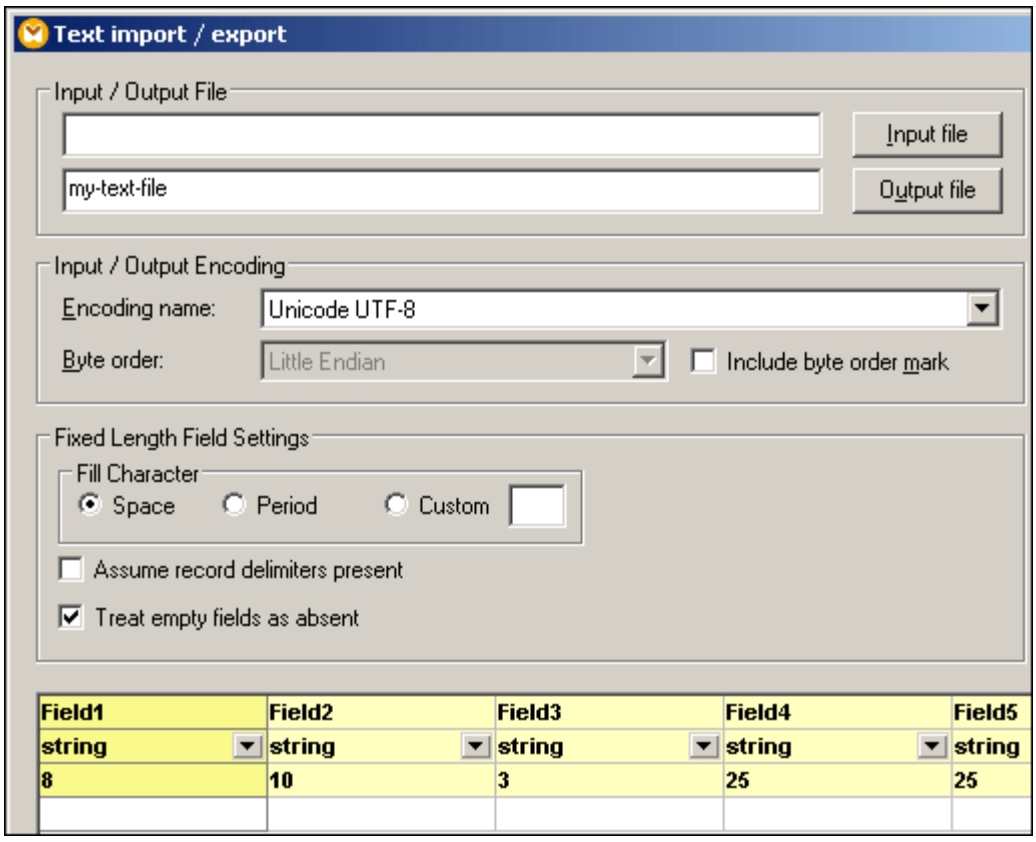
You can also enter a custom fill character in the **Custom** field. Click into the Custom field and:

- 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.

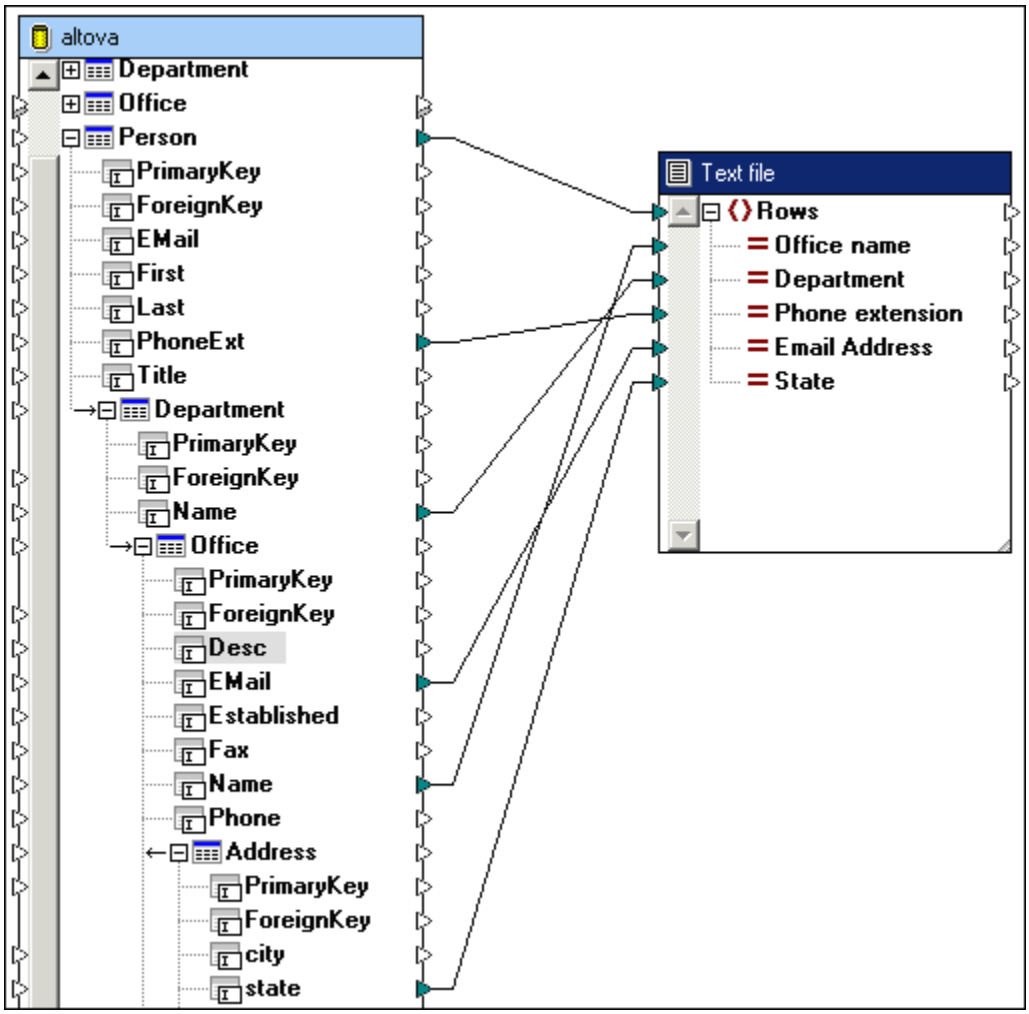
Inactive:
The empty fields of the source file produce the corresponding target items in the output file.

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.

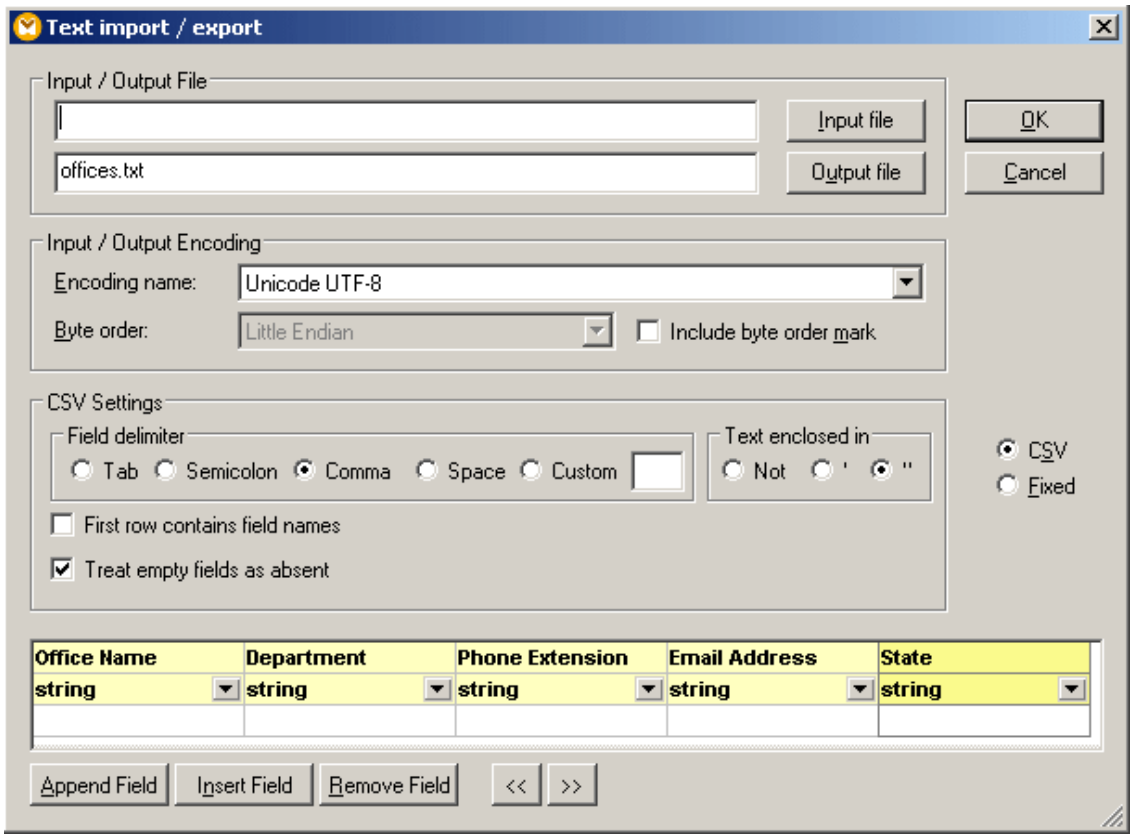
Next / **Previous**
Clicking one of these buttons moves the currently active column left or right in the preview window.

9.6 Mapping Database to CSV/Text files

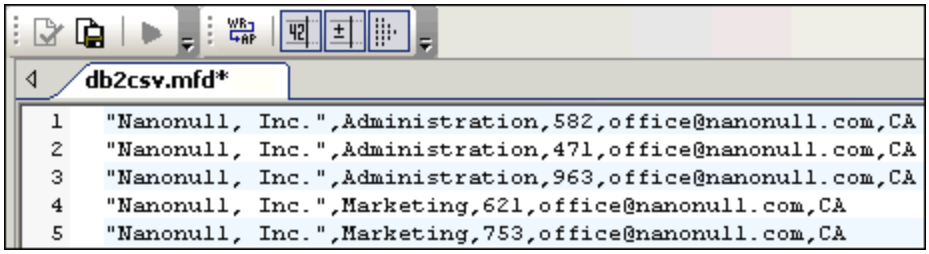
This example maps a simple MS Access database, altova.mdb, to a CSV file. The altova.mdb file is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.



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 10

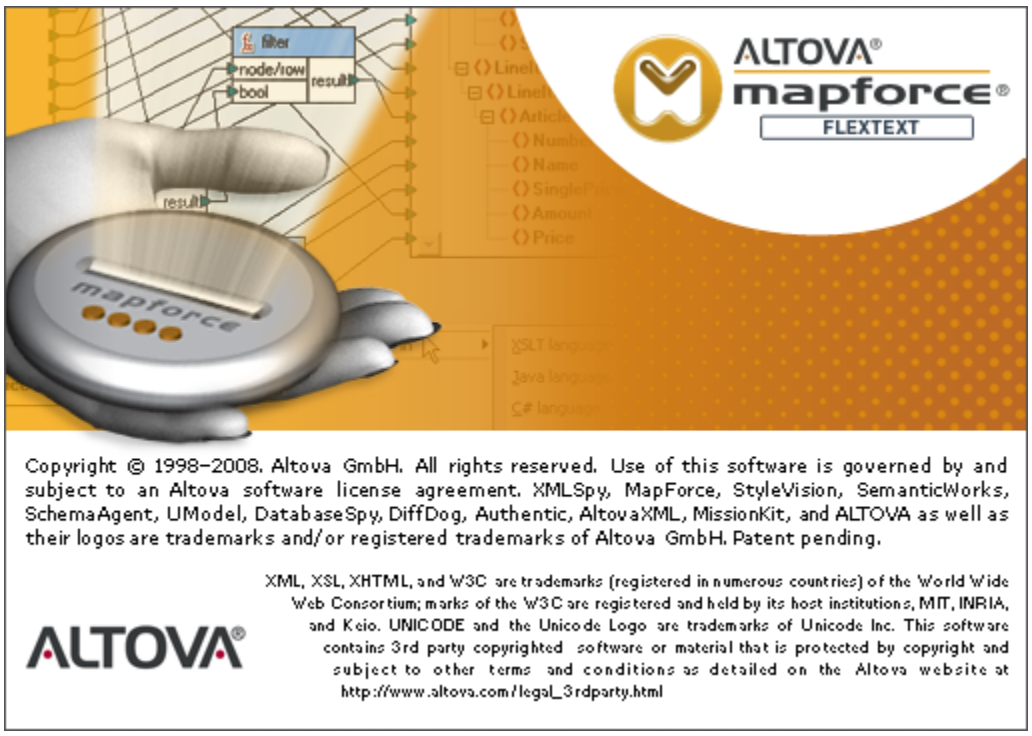
MapForce FlexText

10 MapForce FlexText

The FlexText module of MapForce 2008 allows advanced processing and mapping of legacy text files.

A FlexText template, which defines file structure and content, is defined in the FlexText module, and is then inserted as a component into a mapping, where you can further decide which items/sections you want to map to other target files.

Target files may be any of the many types that MapForce supports: text, XML, database, or EDI files.



10.1 Overview

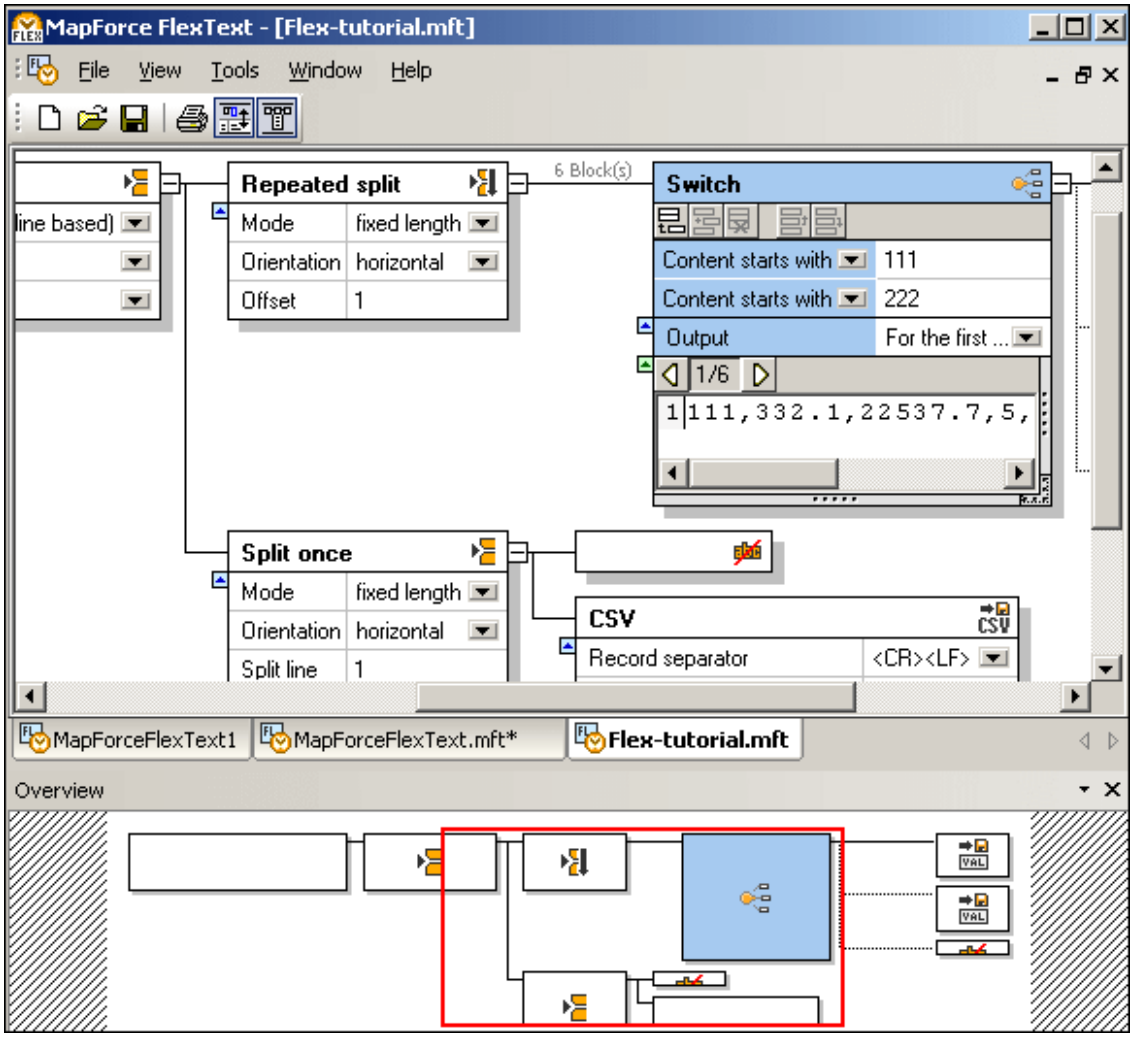
FlexText provides a graphical interface which allows you to map extremely complex flat files, containing multiple delimiters, nested in-line structures and other complexities in MapForce, ready for your code-generation needs.

FlexText produces a template which is then loaded into MapForce, where the individual items can be mapped to any type of target component. The template works on a text file that is supplied/opened in MapForce. This allows you to reuse the same template for multiple text files and in multiple mappings.

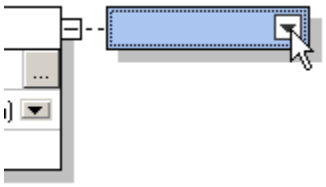
FlexText lets you define the structure of the flat file interactively, and get instant feedback in the Sample Text pane.

FlexText has three main panes: Design, Overview and Sample Text pane.

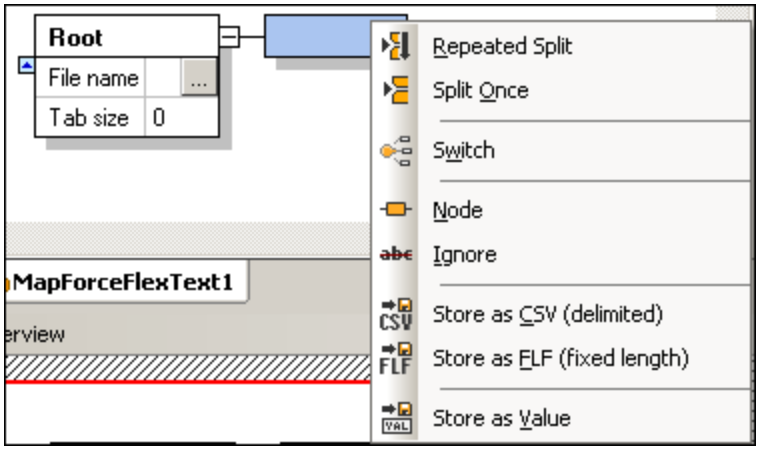
- The Design pane contains text fragment containers, with default names describing their function e.g. Repeated Split.
- The individual containers, or Sample Text pane, display the contents of the currently active container.
- The Overview pane gives a birds-eye view of all the containers in the Design pane. The red rectangle is used to navigate the Design pane.



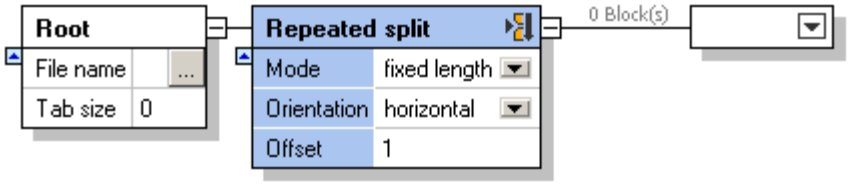
Containers have clickable container icons, which allow you to define the type and content of the container.



Clicking the container icon opens a pop-up menu from which you can select the container type.



Each of the options you select, define the function and content of the container, and present further options for you to refine the content to be provided to the target component in MapForce.



Having selected an option:

- The container changes appearance, type and icon, e.g. "Repeated split" appear in the title bar
- Default options are visible e.g.: mode=fixed length, Orientation=horizontal and Offset=1.
- A new container is automatically appended to the current one.
- Clicking the expand / collapse icon allows you to hide/show container compartments.
- Pressing the Shift key allows you to collapse containers as a group. Two chevrons appear when Shift is pressed. Clicking the handle collapses the section of the container tree to the right of the one clicked.

The "Node Text in Design view" icon  icon, displays the active container contents, in the Sample Text pane.

The "Auto-collapse unselected node text"  icon, displays the content in the active container, all other containers which contain content, are collapsed.

10.2 FlexText Tutorial

The tutorial will show you how to use the most common, and most powerful, features of FlexText to process a text file and map its output in various ways in MapForce.

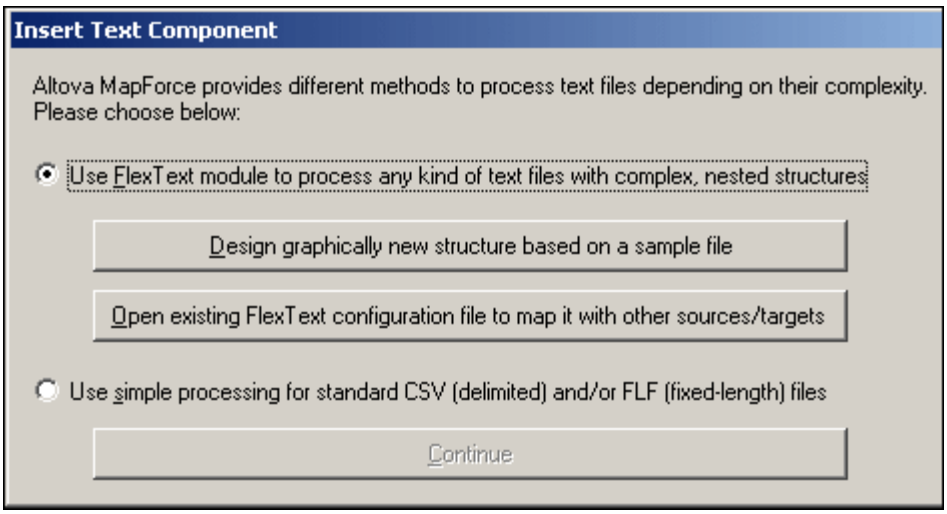
The example uses the **Flex-tutorial.txt** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder, and has the following format:

```
111,332.1,22537.7,5,Container ship,Mega,
111,A1579227,10,3,400,Microtome,
111,B152427,7,6,1200,Miscellaneous,
222,978.4,7563.1,69,Air freight,Mini,
222,ZZAW561,10,5,10000,Gas Chromatograph,,

General outgassing pollutants
1100,897,22.1,716235,LOX
1110,9832,22991.30,002,NOX
1120,1213,33.01,008,SOX
```

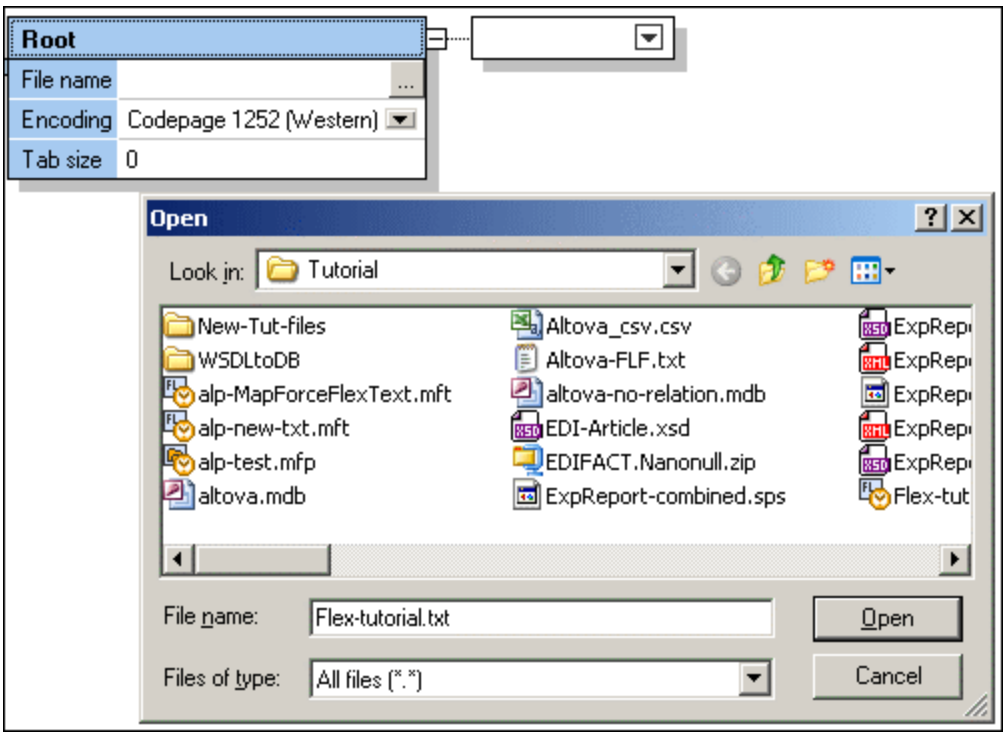
Aim of the tutorial:

- To separate out the records containing 111, and 222 keys, into separately mappable items.
- To discard the plain text record.
- To create a CSV file of the remaining records.

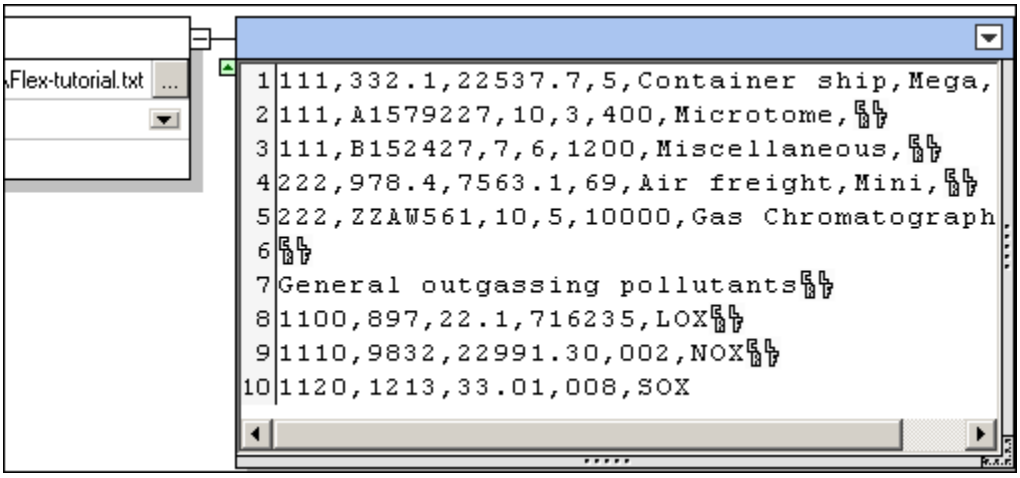
1. Start MapForce, and open a new mapping file.
2. Select **Insert | Text file**, or click the Insert Text file icon  .
3. Click the "**Design graphically new structure ...**" button.



4. Enter a name for your FlexText template, and click Save to continue (e.g. Flex-tutorial.mft).
   An empty design, along with the "Open" dialog box are displayed.



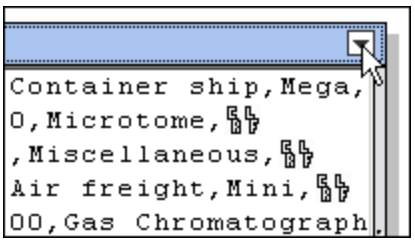
5. Select the **Flex-tutorial.txt** file in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder, and confirm by clicking Open.



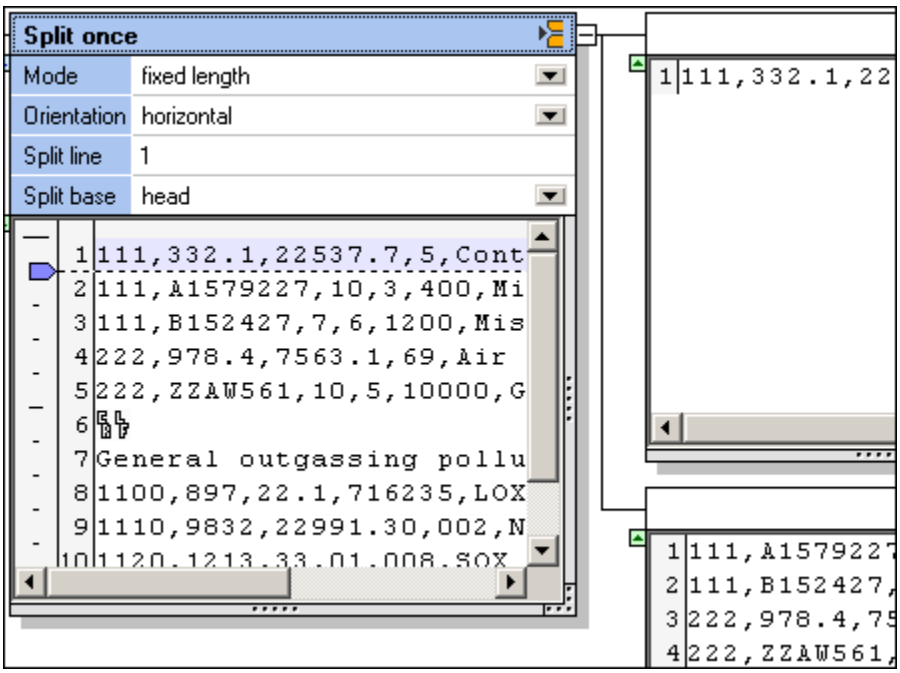
The text file contents are now visible.

Clicking the "Node Text in Design view" icon  icon, displays the active container contents, in the Sample Text pane.

Activating "Auto-collapse unselected node text"  , displays the content in the active container, all other containers which contain content, are collapsed.



6. Click the container icon at the top right, and select **Split once** from the pop-up menu. Two new containers appear next to the Split once container. For more information on the Split once condition, please see: Split once.



The default settings of the Split once container are visible: fixed length, horizontal and split line=1.

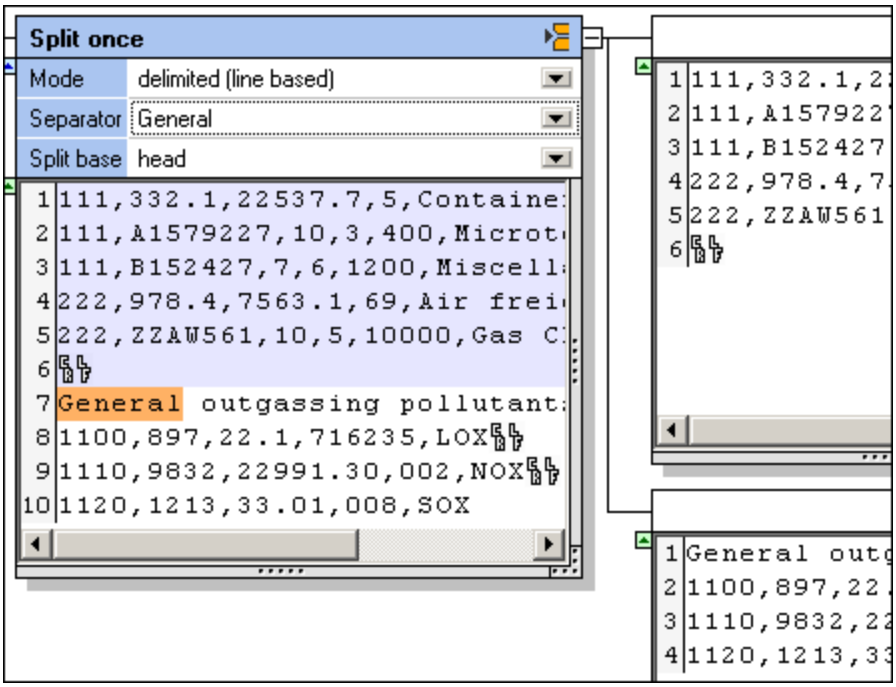
The result of these default settings are also visible:

- The top container contains the first line of the text file, highlighted in the **Split once** container.
- The lower container contains the rest of the text file.

10.3 Creating split conditions

FlexText allows you to define so-called split conditions, that allow you to segment text fragments in various ways.

1. Click the **Mode** combo box and select "delimited (line based)".
2. Double click the Separator field and enter "General".

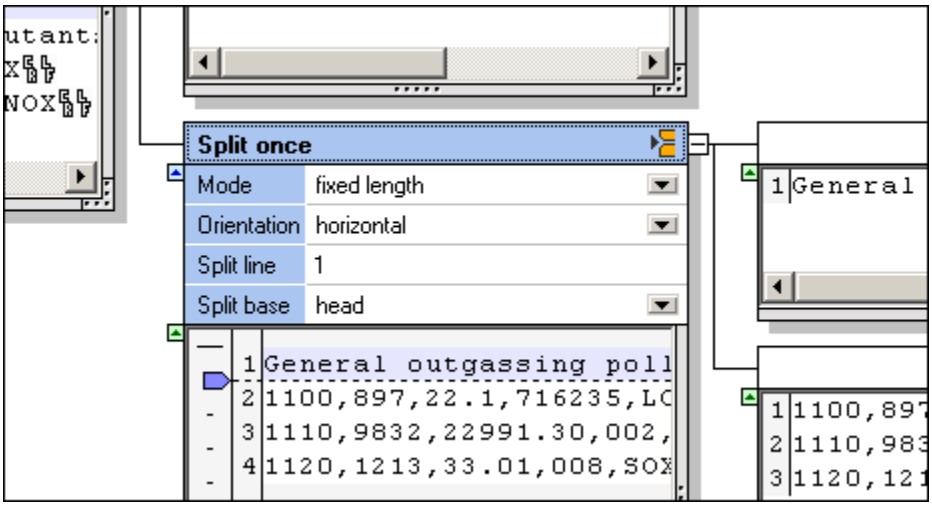


The text fragments in the respective containers have now changed.

Entering "General" and using delimited (line based), allows you to split off that section of text that contains the string "General", into the lower container. The text fragment up to the separator, is placed in the top container. For more information on Repeated split conditions, please see Repeated split.

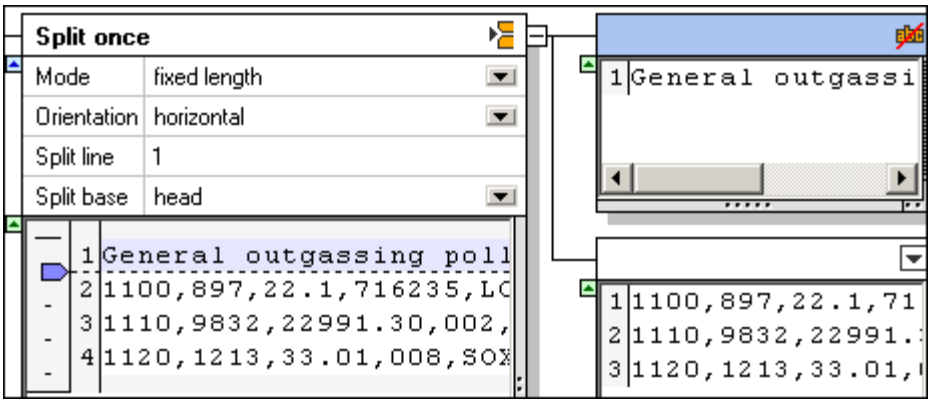
What we want to do now, is work on the lower container to produce a CSV file containing the records with 1100 and up.

3. Click the lower container and change it to **Split once**.



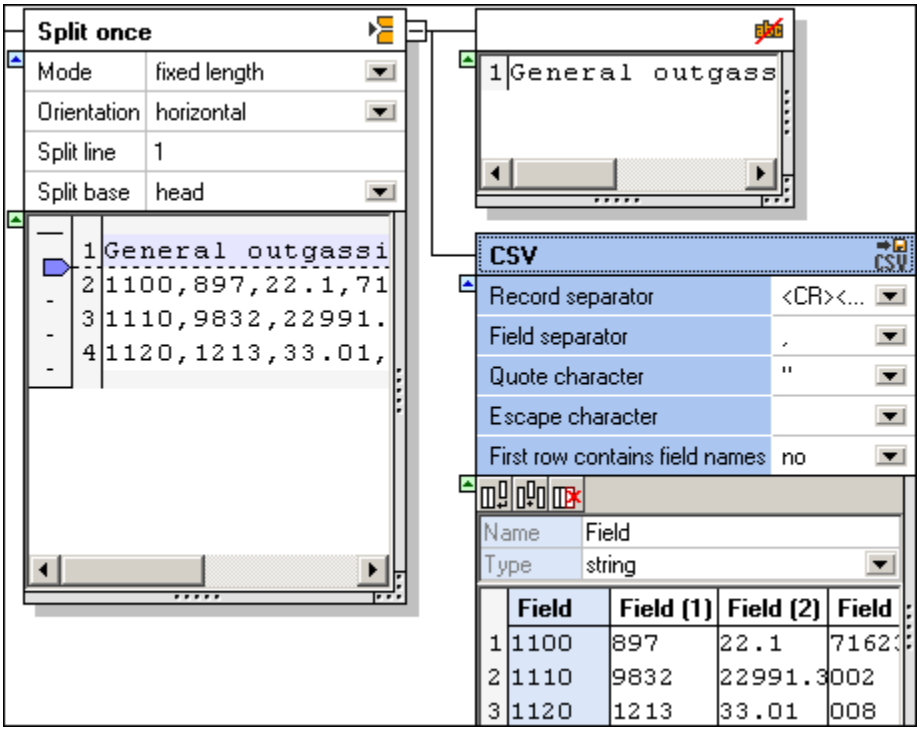
Two new containers are created. The default settings can remain as they are, because we now want to split off the first line of this text fragment, and ignore it. The remaining fragment in the lower container will be made into a CSV file.

4. Click the top container and change it to **Ignore**.



The text fragment, and thus mapping item, of this container has now been made unavailable for mapping in MapForce.

5. Click the lower container icon and change it to **Store as CSV**.



The container now shows the text fragment in a tabular form. The default settings can be retained.

Configuring the CSV file:
If you want to change the field names, click the field, in the table, and then change the entry in the **Name** field. Columns can also be appended, inserted and deleted in this container, please see "Store as CSV" for more information.

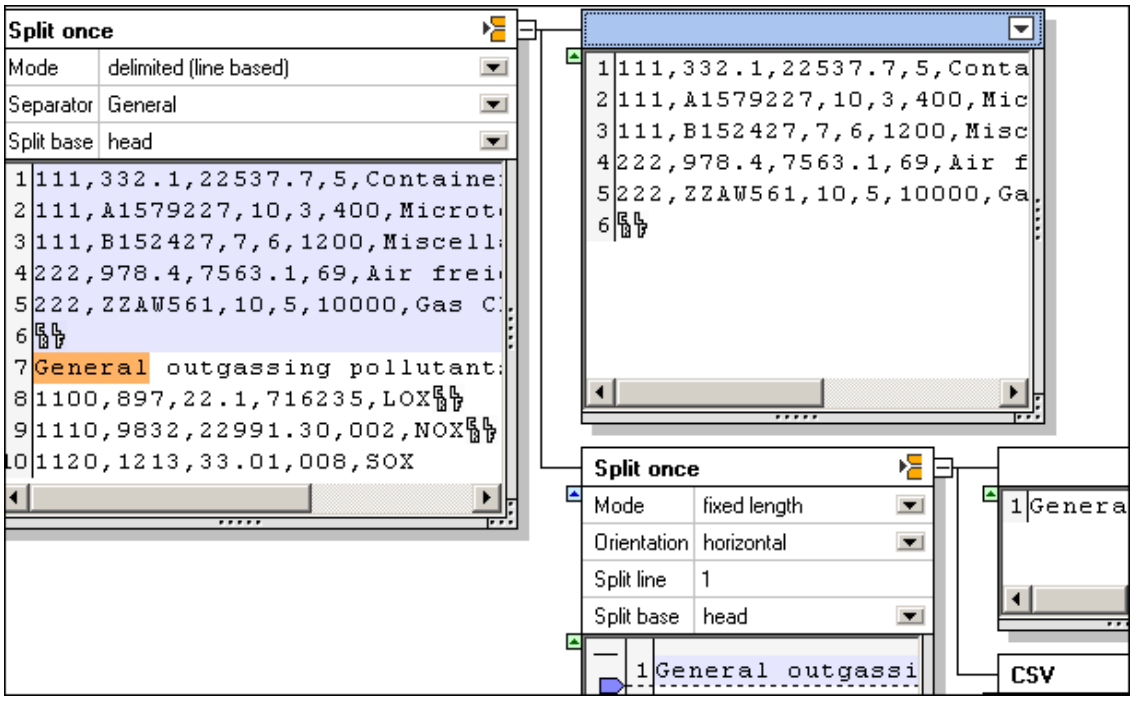
We can now continue with defining the remaining text fragment.

10.4 Defining multiple conditions per container/fragment

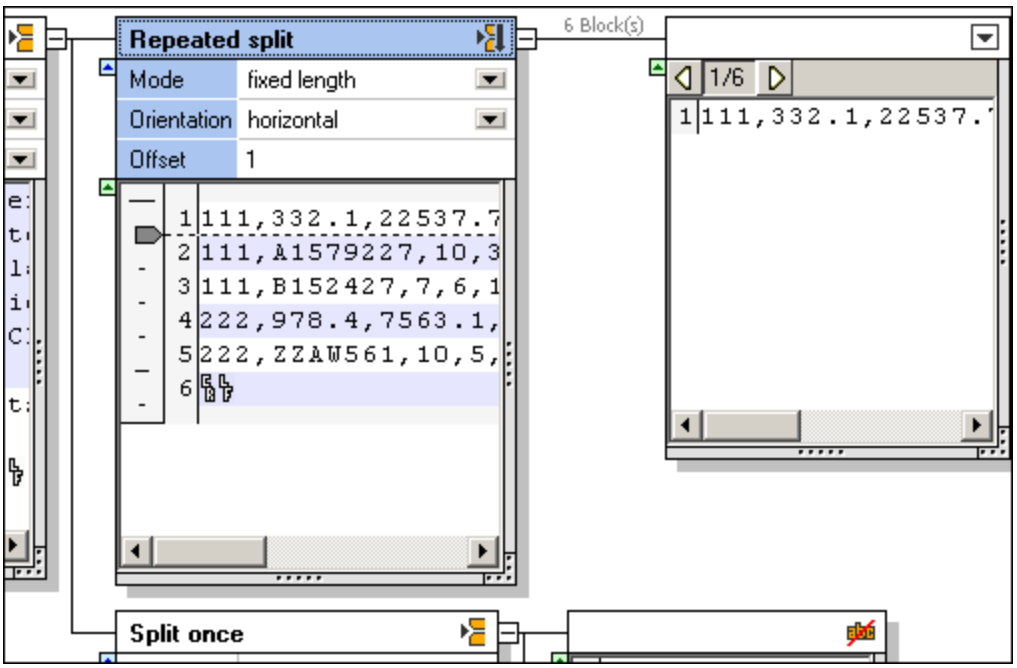
FlexText allows you to define multiple conditions per text fragment, using the Switch container. An associated container is automatically allocated to each condition that you define.

The current state of the tutorial at this point is that lower text fragment, of the first **Split once** container, has been defined:

- A Split once container splits off the first line into an Ignore container.
- The remaining segment is defined/stored as a CSV file.



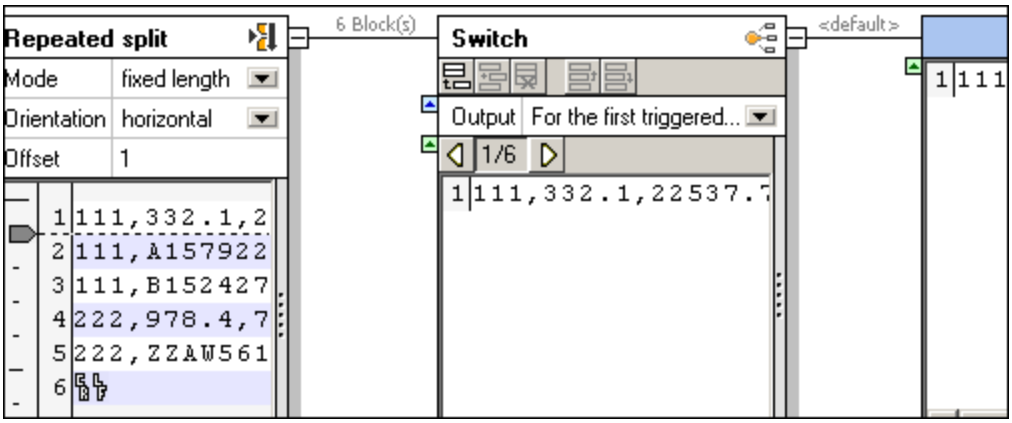
1. Click the top container icon and change it to **Repeated split**.



The default settings are what we need at this point. The text fragment is split into multiple text blocks of a single line each. The associated container shows a preview of each of the text blocks.

Clicking the Next text block icon  , allows you to cycle through all the text fragments, of which there are 6.

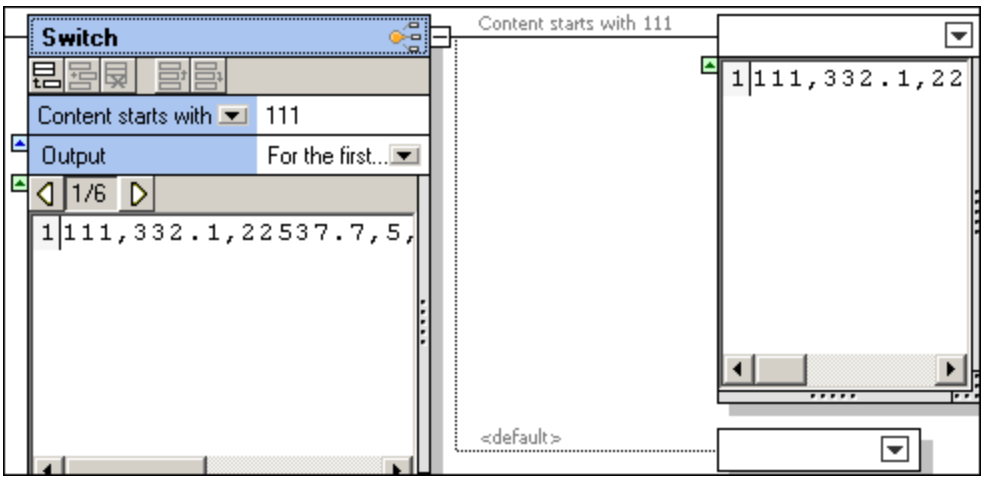
2. Click the new container and change it to **Switch**.



The initial state of the Switch container is shown above.

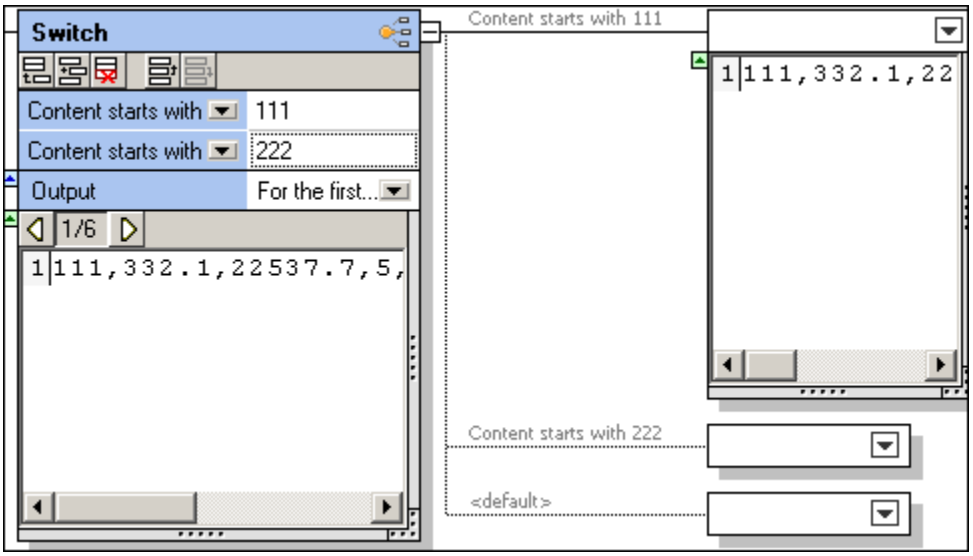
- An associated container "default", has been added.
- The content of the first record 1/6, is displayed in the default container.

3. Click the Append condition icon  in the "Switch" title bar, to add a new condition.
4. Double click in the field "Content starts with", and enter 111.



This defines the first condition. An associated container (Content starts with 111) has been added above the "default" container.

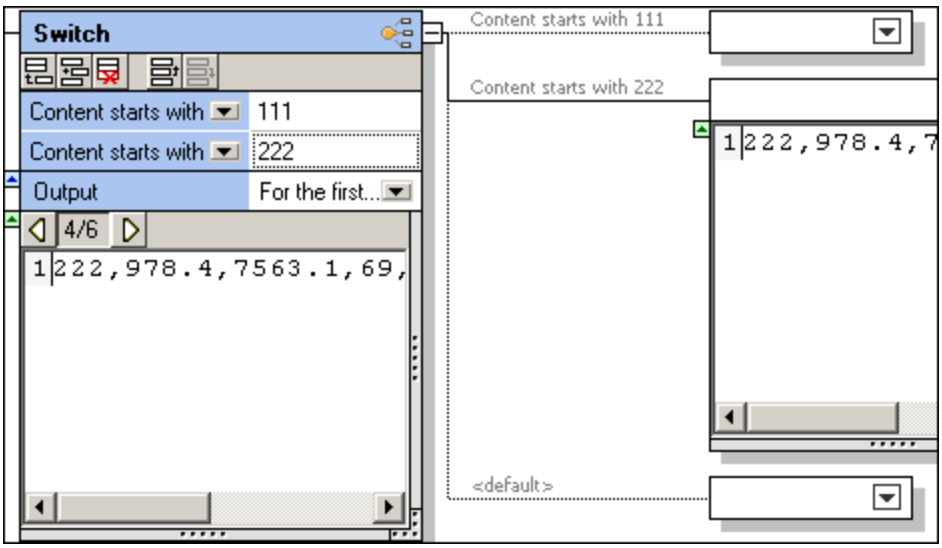
5. Click the append icon again, and enter "222" in the Content starts with field.



A third container has been added (Content starts with 222).

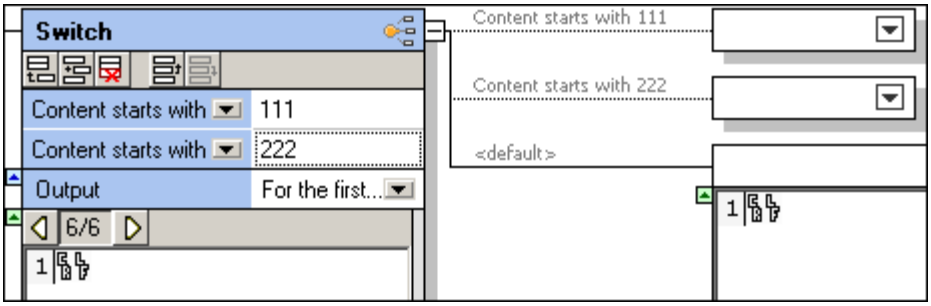
Please note:
Clicking the "Contents starts with" combo box, allows you to select the "Contains" option. This allows you to specify a "string" which can occur anywhere in the text fragment.

7. Click the Next text block icon , several times to see the effect.



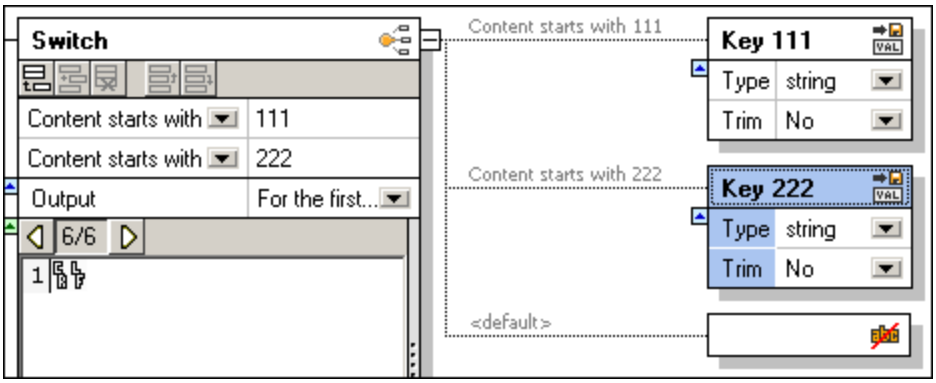
Upon reaching record 4 of 6, container 222 opens up, and displays its content.

8. Continue clicking, till you reach record 6 of 6. A single CR / LF character is displayed in the default container.



If a data fragment in the current block satisfies a condition, then the **complete data** of that block is passed on to the associated container. Data is not split up in any way, it is just routed to the associated container, or to the default container if it does not satisfy any of the defined conditions.

9. Click the first two containers and change them to **Store as value**. Click the last container and change it to **Ignore**.
10. Double click the "Store" text, and add descriptive text e.g Key 111 and Key 222.

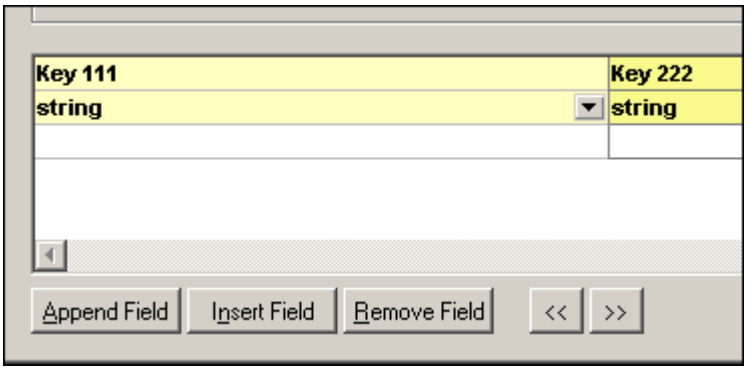


11. Save the FlexText template, e.g. Flex-Tutorial.mft.

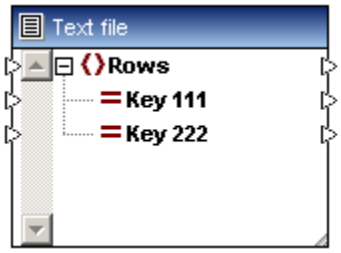
10.5 Using FlexText templates in MapForce

Creating the target components for use with the FlexText source:

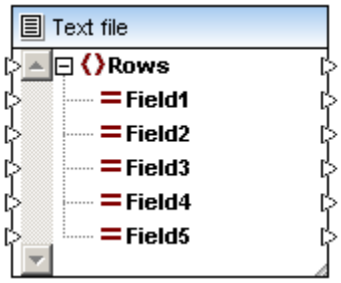
1. Select **Insert | Text file**, or click the Insert Text file icon  .
2. Click the "**Use simple processing...**" radio button and click **Continue**. This opens the Text import / export dialog box.
3. Click the **Append Field** button to add a new field.
4. Double click the Field1 field name and change it to **Key 111**.



5. Do the same for the other field, and name it **Key 222** and click OK to confirm. A text component with two fields has now been created.

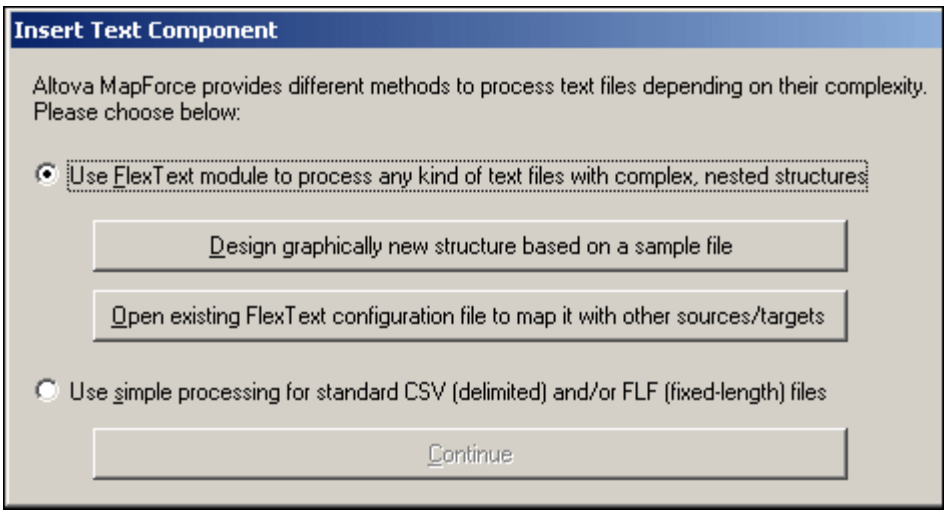


6. Use the same method to create a second text component that consists of five fields.

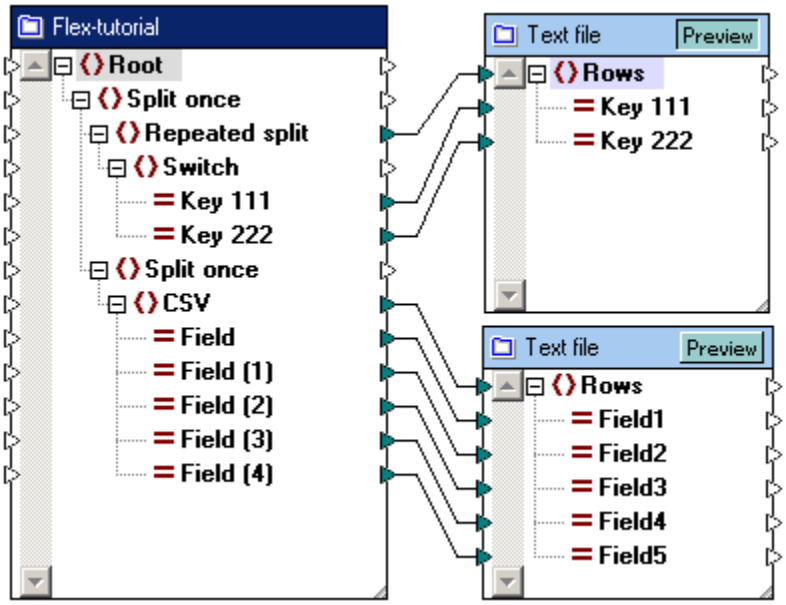


Using the FlexText template in MapForce:

1. Start, or switch back to MapForce, and select **Insert | Text file**.



2. Click the **Open existing FlexText configuration file...,** button and select the previously defined FlexText template (e.g. Flex-tutorial.mft).
   The structure of the MapForce component, mirrors that of the containers in Design view in FlexText.



3. Map the various items to the previously defined target components, and click the Output tab to preview the results.

Mapping preview of the top text component:

```
 1   "111,332.1,22537.7,5,Container ship,Mega,
 2   ",
 3   "111,A1579227,10,3,400,Microtome,
 4   ",
 5   "111,B152427,7,6,1200,Miscellaneous,
 6   ",
 7   ,"222,978.4,7563.1,69,Air freight,Mini,
 8   "
 9   ,"222,ZZAW561,10,5,10000,Gas Chromatograph,,
10   "
11
```

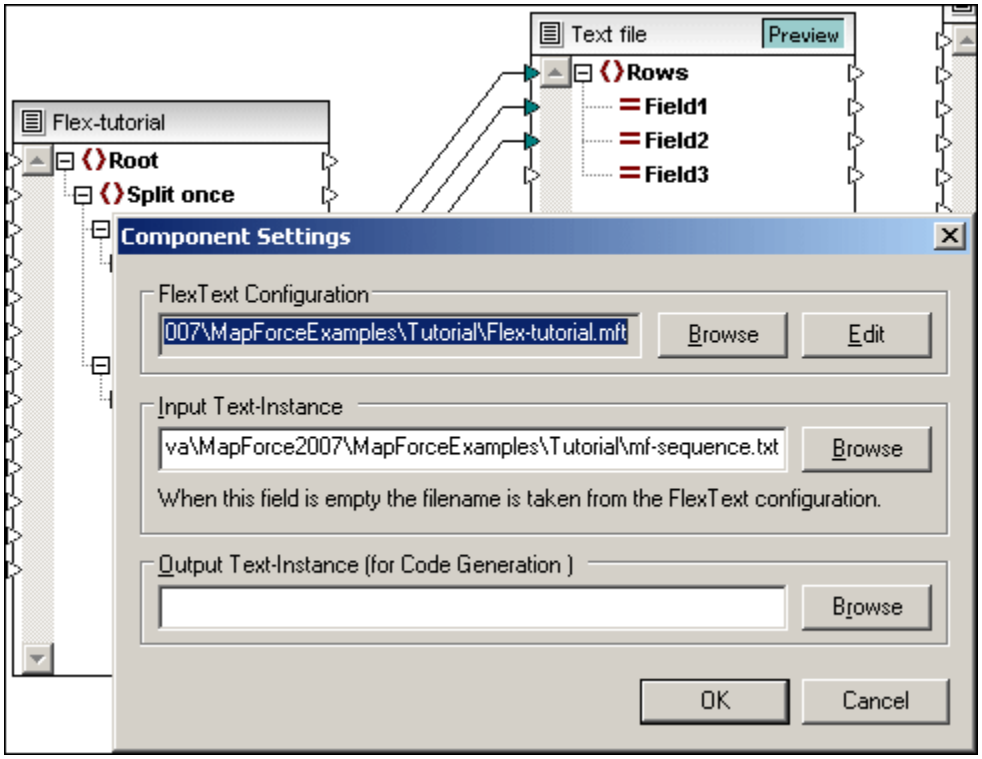
Mapping preview of the lower text component:

```
1   1100,897,22.1,716235,LOX
2   1110,9832,22991.30,002,NOX
3   1120,1213,33.01,008,SOX
4
```


10.5.1 FlexText data sources

Double clicking a FlexText component opens the **Component Settings** dialog box, in which you can define the various input and output files.

The **FlexText Configuration** field contains the data source defined when the component was created.



If an entry exists in the **Input Text-Instance** field, then this data source is used in place of the one in the FlexText Configuration field.

The entry in the **Output Text-Instance** field is only used when generating code. Enter a name, when generating code as FlexText automatically generates a file name. Entering a full path also allows you to specifically define the target directory e.g. c:\myfiles\sequence.txt.

10.6 Using FlexText as a target component

The main use of FlexText components is to reduce, or split off sections of text files and map the relevant items of the FlexText component to other target components. FlexText can, however, also be used as target component to assemble or reconstitute separate files into a single file, although this can entail a fair amount of trial and error to achieve the desired results.

Using a FlexText component as a **target** reverses the operations previously defined in it. Instead of splitting a file into various subsections you assemble/reconstitute a file.

In general the inverse of each operation defined in the FlexText template is carried out (in a bottom-up fashion) when using it as a target:

- a split becomes a merge, e.g. mapping to a repeated split delimited by "," becomes a merge items separated by ",".
- store becomes load
- and switch becomes "choose the first match".

FlexText component as target component
As soon as a connection is made between a data source component and one of the **input** items of a FlexText component, the FlexText component data source is ignored. The data provided by the newly mapped source component now takes precedence.

Notes:

- If text mapped to a "Store as..." (Store as CSV and FLF) container, then the separator is retained. Text might however, be truncated if a fixed length split occurs in a node above the "Store as..." node.

- Fixed Width Splits truncate the left/top section if Split Base=Head, or the right/bottom section if Split Base=tail, to the predefined length. The trunctated section is then as long as the defined length in characters. If the text is too short, then space characters are inserted to pad the section.

- FlexText would normally insert separators (or whitespace for fixed splits) between the items of a split operation, but this is not the case for 'delimited (line based)' splits. The 'delimited (line based)' operation is not a perfectly reversible operation. The "Delimited" text may occur anywhere in the first line and is included in the text, and therefore an automatic process cannot reliably add it.

  - Delimited (linebased), will not add a separator to the first line if it is missing.

  - Delimited (floating), will add a separator between two sections.

- The switch operation cannot be inverted in a meaningful way except for simple cases. The switch scans its branches for the first branch that contains data, and uses/inserts this data.

  Only the first connection of a switch operation is mapped. To transfer data to the remaining switch containers, filters have to be defined for the remaining connectors and the duplication of the switch parent item is necessary, so that each switch item returns a single item which is then fed to a repeated split item to merge all of them.

- Mapping to a child of a single split container discards all mapping results except for the last item. Only a single result is retained, even if multiple results were generated.

The following analogy to the XML Schema content model gives some idea of FlexText's behavior when used as a target:

- A repeated split is a repeatable element.
- A single split forms part of a 'sequence' content model group.
- A switch forms a 'choice' content model group, each case being a possible child element.
- A store creates an element of simple type.

10.7 FlexText Reference

The reference section describes the various features of FlexText, and shows how best to use them, to achieve specific results.

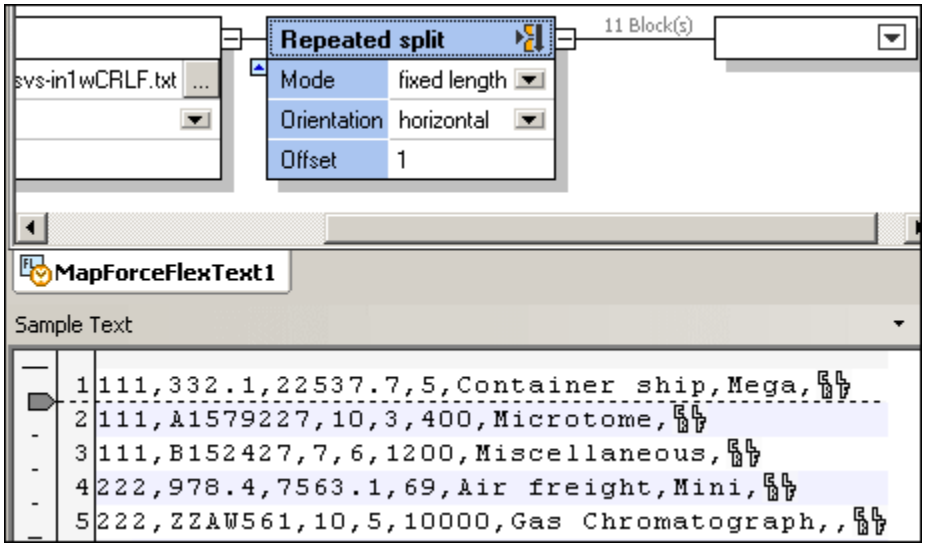
10.7.1 Repeated split

Using this option initially creates a single container. The container contains the text defined by the condition set in **Repeated Split**. There are several versions of the Repeated split option; Fixed length, Delimited Floating, Delimited Line based, and Delimited Starts with...

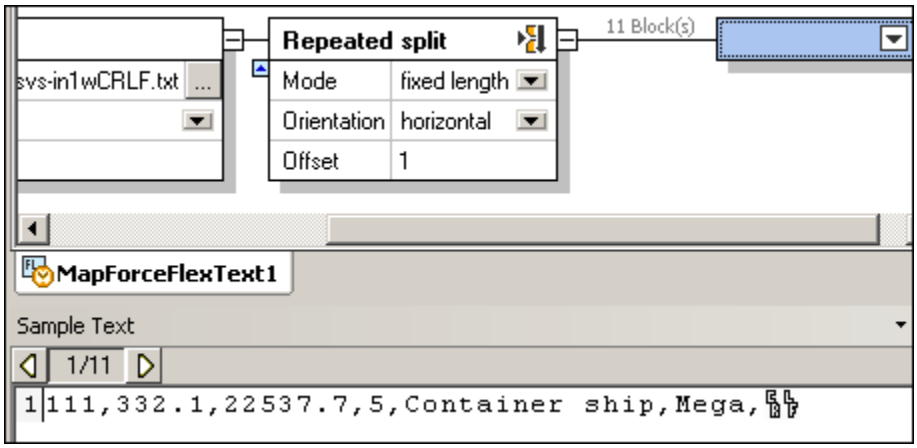
When you first select this option, default parameters are automatically set and the resultant fragments appear in the associated container. Note that the Repeated Split container is currently active, and the preview displays all current records/lines, in the Sample Text pane.

Container **default settings**:

Mode	fixed length
Orientation	horizontal
Offset	1



Default result:
Each line of text appears as a line/record in the new container, as the Offset is 1. Click the new container to preview its contents. The Sample Text **scroll arrows**, let you scroll through each of the 11 blocks/fragments produced by these settings.



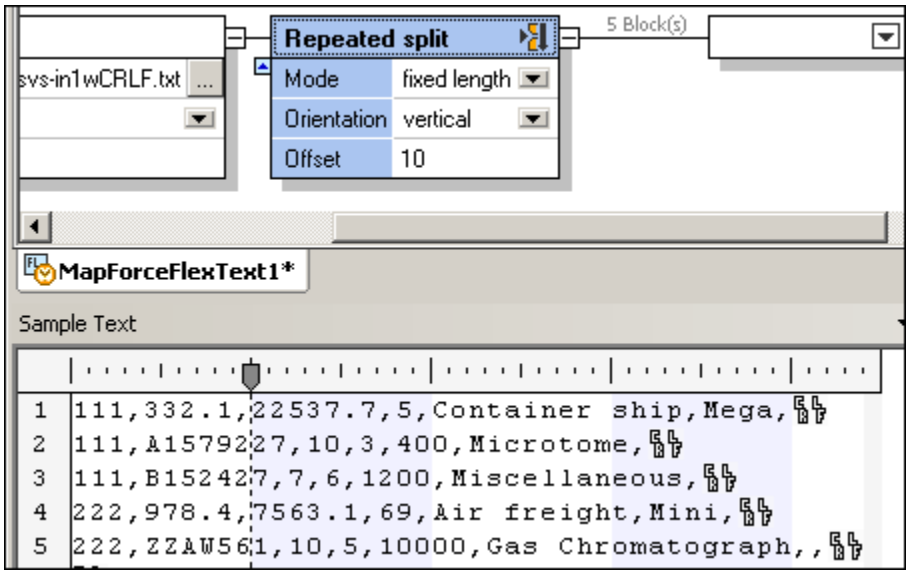
Mode - Fixed length

This is the default value.

Orientation:
Allows you to define how the text fragment is to be split, by lines/records, or columns.

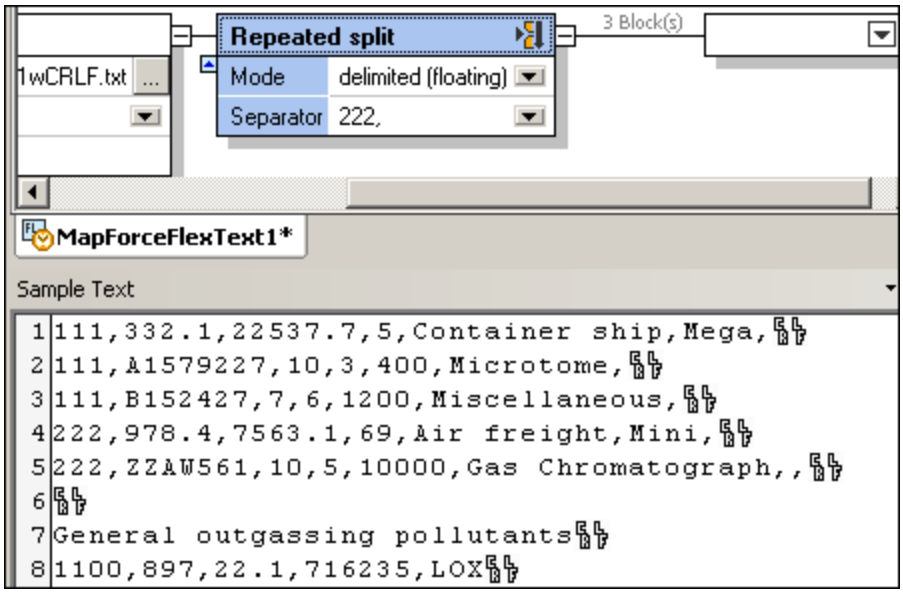
Horizontal
Splits the fragment into **multiple** horizontal sections (see above). Enter a value into the Offset field, or drag the tab on the vertical ruler.

Vertical
Splits the fragment into **multiple** vertical columns. Enter a value into the Offset field, or drag the tab on the horizontal ruler. Each fragment contains the characters of the column defined by the Offset width e.g. 10, to the end of the file/fragment.



Mode - Delimited Floating

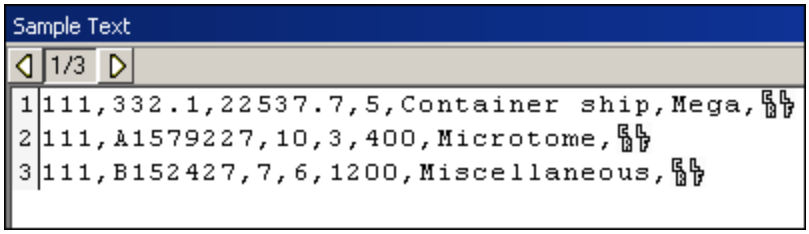
Default settings:
Separator (none)



- Creates **multiple** fragments defined by separator characters, that you enter in the Separator field.
- The separator characters are **not included** in the fragment.
- A block/fragment is defined as the text between the first character **after** the separator, up to the last character before the next instance of the same separator (Except for first and last fragments, please see below).

Using the separator "**222,**" as shown above, produces 3 separate fragments:

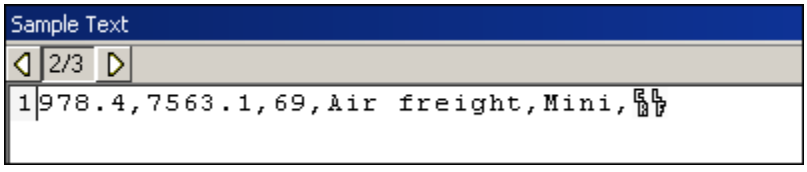
fragment 1, contains all characters from the **start** of the fragment, to the start of the **first separator** (222,), i.e. from 111... to Miscellaneous,.



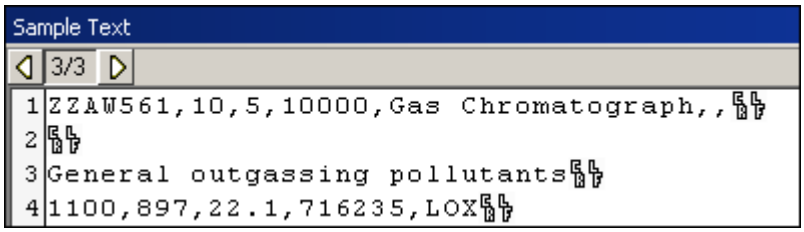
If the separator is not the first set of characters of the first line in the fragment, as in this example, then the first fragment includes all the text **up to** the first instance of the separator. Eg. 222,.

If 111, were the separator, then the first fragment would only consist of the first line of this fragment, minus the separator itself.

fragment 2, contains the **first** line containing the separator 222, without the separator.



fragment 3, contains the **next** line containing the separator 222, without the separator itself, up to the end of the text file/fragment.



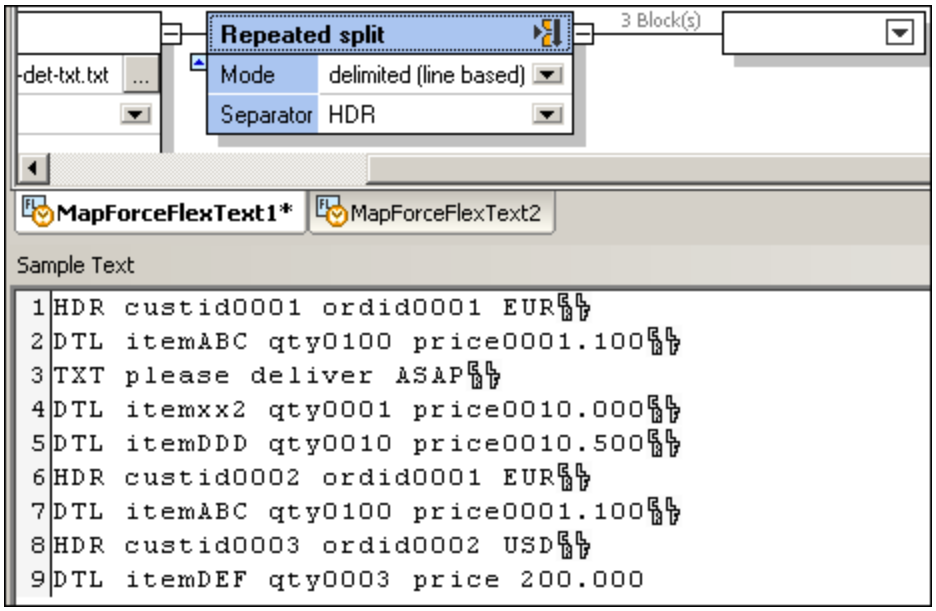
Use this option when you want to process fragments:

- Containing separators which you want to strip out
- Where the fragment might not have CR/LF characters, and thus have in-line separators.

```
ous,222,978.4,7563.1,69,Air freight,Mini,222,ZZAW561,
```


Mode - Delimited Line based

Default settings:
Separator (none)



- Creates multiple fragments defined by separator characters that you enter in the Separator field.
- The separator characters are **included** in the fragment.
- A fragment is defined as the **entire line** containing the separator, up to the **next line** containing the same separator.
- If the separator does **not** appear in the first **line**, then the first fragment contains the line(s) up to the first line containing the separator.

Using the separator "**HDR**" as shown above, produces 3 separate fragments:

fragment 1, contains all characters from the start of the file/fragment, including all lines up to the next line containing the same separator.

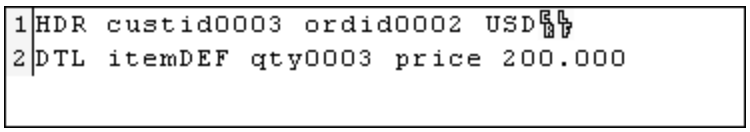
```
1 HDR custid0001 ordid0001 EUR
2 DTL itemABC qty0100 price0001.100
3 TXT please deliver ASAP
4 DTL itemxx2 qty0001 price0010.000
5 DTL itemDDD qty0010 price0010.500
```

Note that this option allows you access to any number of lines between two separators. E.g. Header / Detail / Text files, where the DTL, or TXT lines may be optional, or not in sequence.

fragment 2, contains all characters/lines from the second occurrence of HDR, till the next occurrence of HDR.

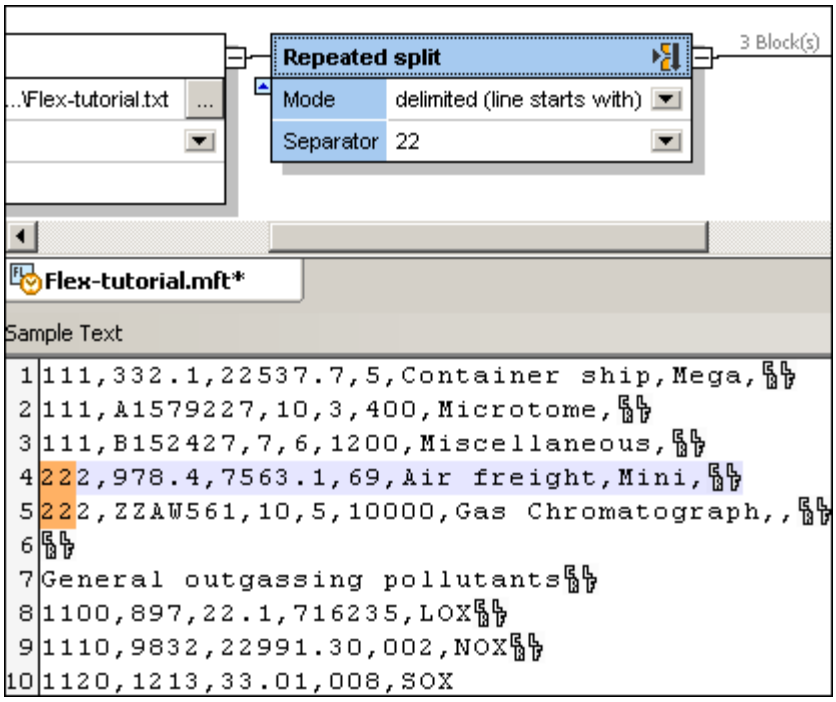
```
1 HDR custid0002 ordid0001 EUR
2 DTL itemABC qty0100 price0001.100
```

fragment 3, contains all characters/lines from the third occurrence of HDR, till the next occurrence of HDR.



Mode - Delimited Starts with

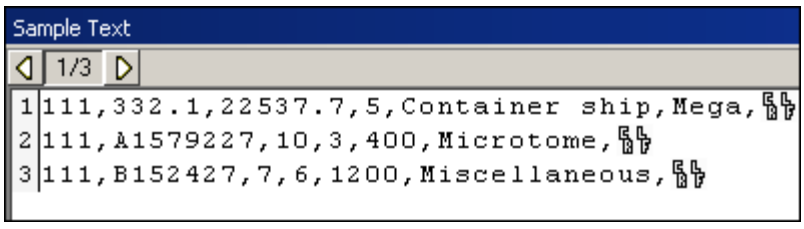
Default settings:
Separator (none)



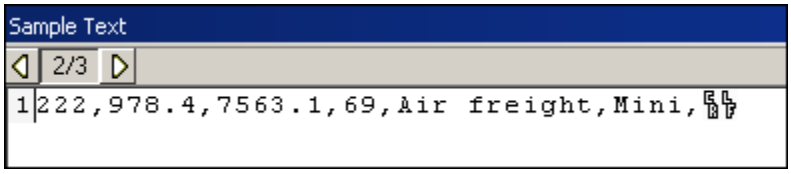
- Creates multiple fragments defined by separator characters that you enter in the Separator field.
- The separator characters are **included** in the fragment.
- A fragment is defined as the **entire line** - **starting** with the separator, up to the **next line** containing the same separator at the **start** of the line.
- If the separator does **not** appear in the first **line**, then the first fragment contains the line(s) up to the first line containing the separator.

Using the separator "**22**" as shown above, produces 3 separate fragments:

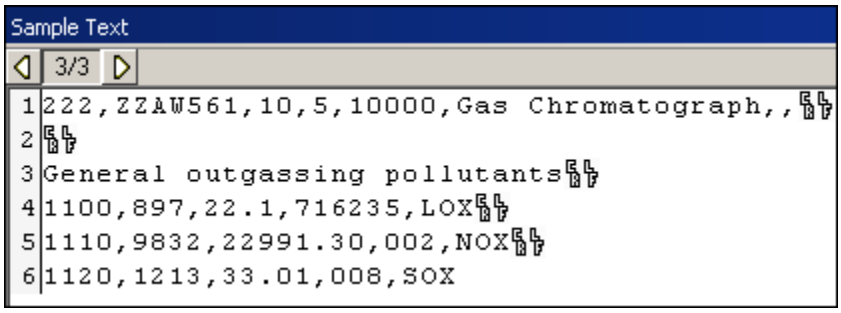
fragment 1, contains all characters from the start of the file/fragment, including all lines up to the line containing the separator 22.



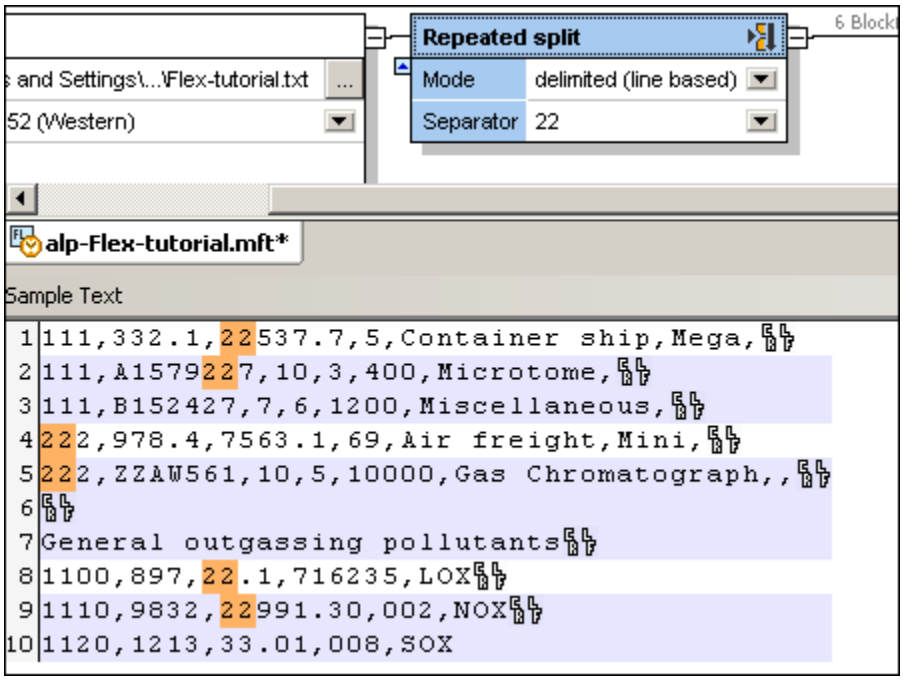
fragment 2, contains all characters/lines from the second occurrence of 22, till the next occurrence of 22, which in this case is only one line.



fragment 3, contains all characters/lines from the third occurrence of 22, till the end of the file/fragment.



Contrast the above example to what would happen if we used **delimited line based** and separator as **22**:



There would be six fragments, composed of lines that contained 22 anywhere in that line.

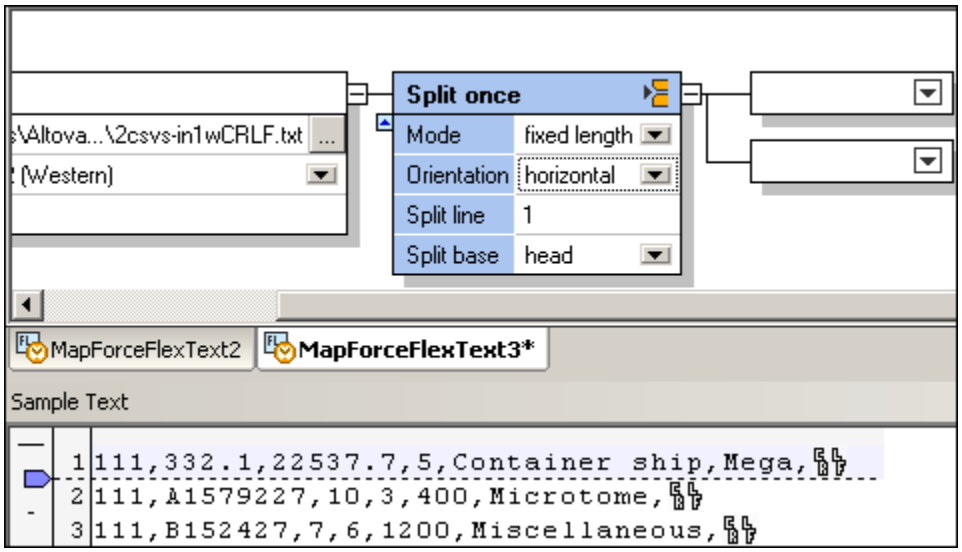
10.7.2 Split once

Using this option creates two vertically aligned containers. The top container contains the text defined by the condition set in the **Split once** container. The bottom container contains the rest of the text file/fragment. There are several versions of the Split once option: Fixed length, Delimited Floating, and Delimited Line Based.

When you first select this option, default parameters are automatically set, and the resultant fragments appears in both containers. Note that the **Split once** container is currently active, and displays a preview of all current records/lines, in the Sample Text pane.

Container **default settings** are:

Mode	fixed length
Orientation	horizontal
split line	1
Split base	head



Default result:
The first line of text appears in the top container. The bottom container contains the rest of the text file/fragment.

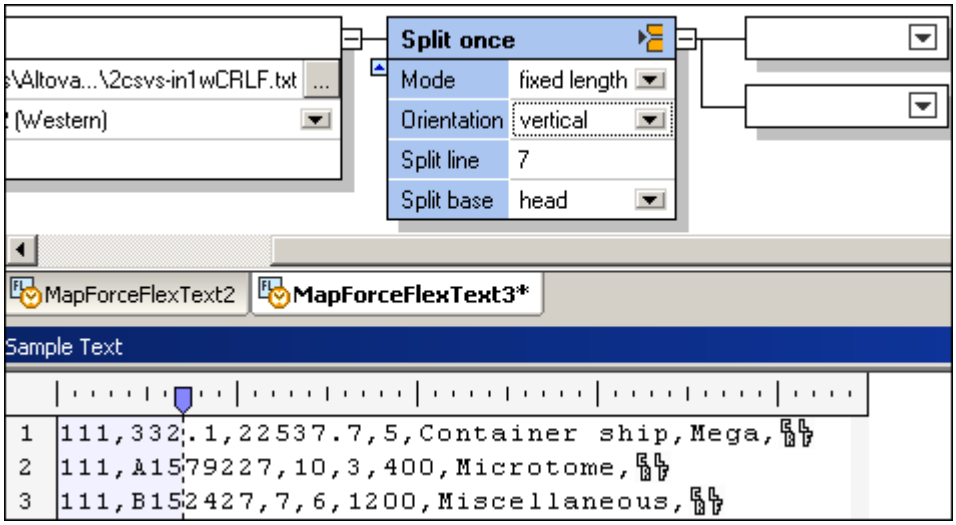
Mode - Fixed length

This is the default value.

Orientation:
Allows you to define how the text fragment is to be split, by lines/records, or columns.

Horizontal
Splits the fragment into **two** horizontal sections. Enter a value into the Split line field, or drag the tab on the vertical ruler.

Vertical
Splits the fragment into **two** vertical columns. Enter a value into the Split line field, or drag the tab on the horizontal ruler.



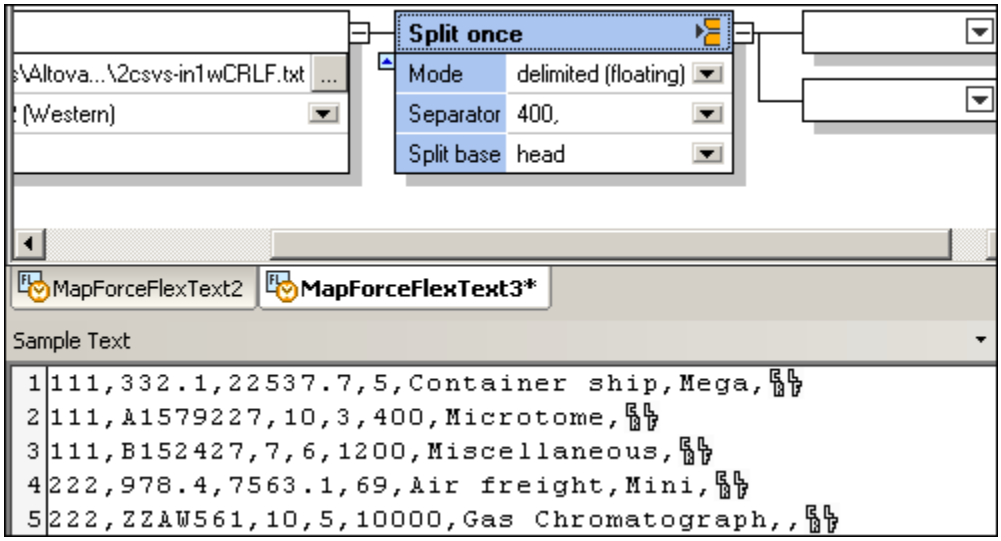
Split Line:
The number of lines after which the fragment should be divided into two.

Split base:

Head	splits by the number of split lines from the top
Tail	splits by the number of split lines from the bottom (use when no. of lines/records unknown).

Mode - Delimited Floating

Default settings:
Separator (none)
Split base head



- Creates **two** fragments defined by separator characters that you enter in the Separator field.
- The separator characters are **not included** in the fragment.
- The first fragment is defined as the text between the first character of the file/fragment, up to the last character before the separator.
- The second fragment is defined as the first character after the "separator", up to the last character in the file/fragment.
- If the separator appears in the first/last **position** of the file/fragment, then the top container remains empty.

Use this method when you want to split off **one section** of a file, or fragment, where the separator is anywhere in the file/fragment. This is generally useful in files that do not contain CR, or LF characters, and you want to split the fragment into two, at some specific in-line location.

The **top** fragment contains the text up to the separator (i.e. 400,)

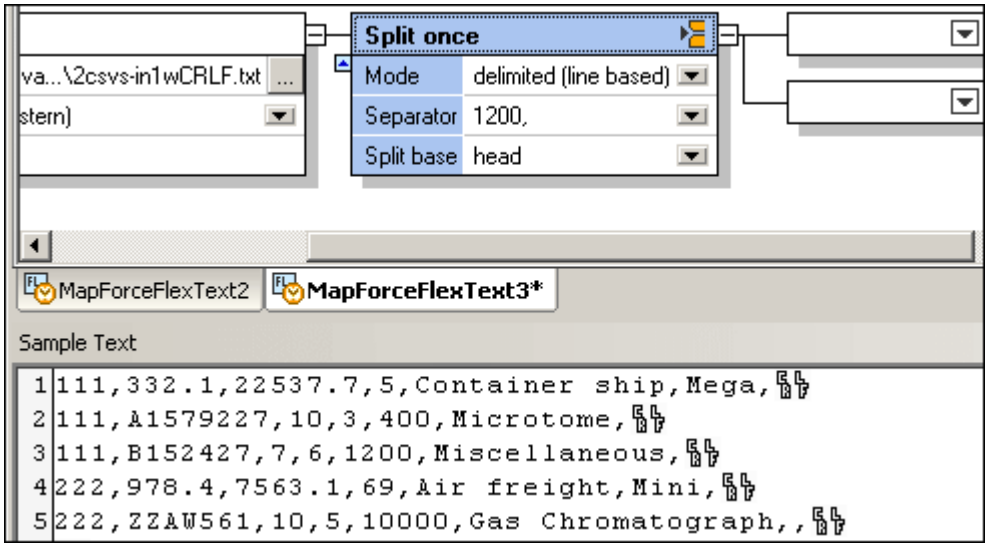
```
1 111,332.1,22537.7,5,Container ship,Mega,
2 111,A1579227,10,3,
```

The **bottom** fragment contains the remaining characters after the separator.

```
1 Microtome,
2 111,B152427,7,6,1200,Miscellaneous,
3 222,978.4,7563.1,69,Air freight,Mini,
4 222,ZZAW561,10,5,10000,Gas Chromatograph,
```

Mode - Delimited Line based

Default settings:
Separator (none)
Split base head



- Creates **two** fragments defined by separator characters that you enter in the Separator field.
- The separator characters are **included** in the fragment.
- The first fragment is defined as all the text, up to the line containing the separator.
- The second fragment is defined as the text, and line, including the separator up to the end of the file/fragment.
- If the separator appears in the first/last **line**, of the file/fragment, then the top container remains empty.

Use this method to split a file, or fragment into two, where the separator is anywhere in one of the lines. The line containing the separator is not split, but is retained whole. This is generally useful in files containing record delimiters (CR/LF), and you want to split the fragment into two separate fragments.

The **top** fragment contains the text **up to** the **line** containing the separator.

```
1 111,332.1,22537.7,5,Container ship,Mega,
2 111,A1579227,10,3,400,Microtome,
```

The **bottom** fragment contains the entire line containing the separator (1200,), and all remaining lines to the end of the file/fragment.

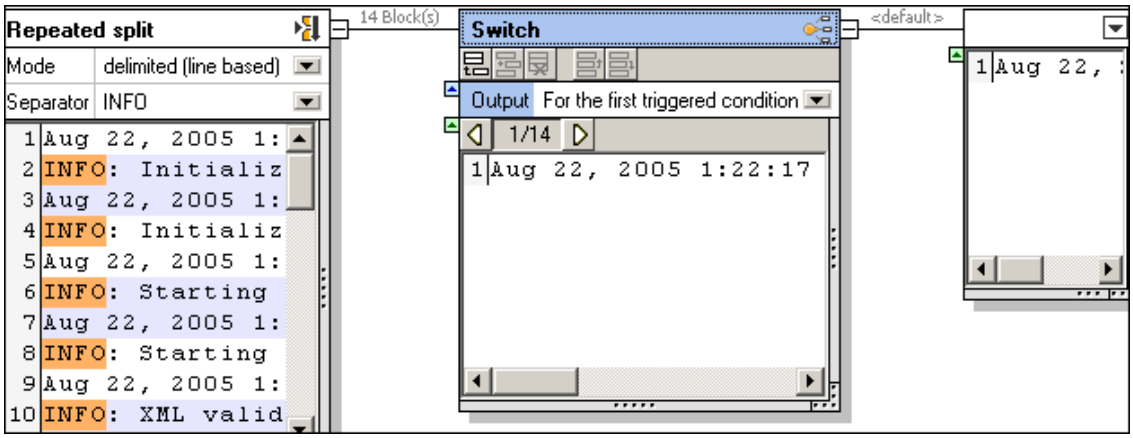
```
1 111,B152427,7,6,1200,Miscellaneous,
2 222,978.4,7563.1,69,Air freight,Mini,
3 222,ZZAW561,10,5,10000,Gas Chromatograph,,
```


10.7.3 Switch

Using the Switch option allows you to define multiple keywords, or conditions, for a single text fragment. Each keyword you define, has its own associated container which receives data only if the specific condition is satisfied, i.e. true. If none of the conditions are satisfied, then the specific fragment is mapped to a "default" container.

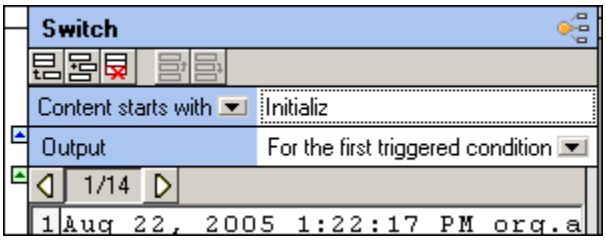
Container **default settings** are:
Output For the first triggered condition.

The example below processes a Tomcat log file, where the individual processes are to be separated out, and made mappable. When you first define a Switch container, only the **default** container appears to the right of the Switch container. All data is automatically passed on to it.



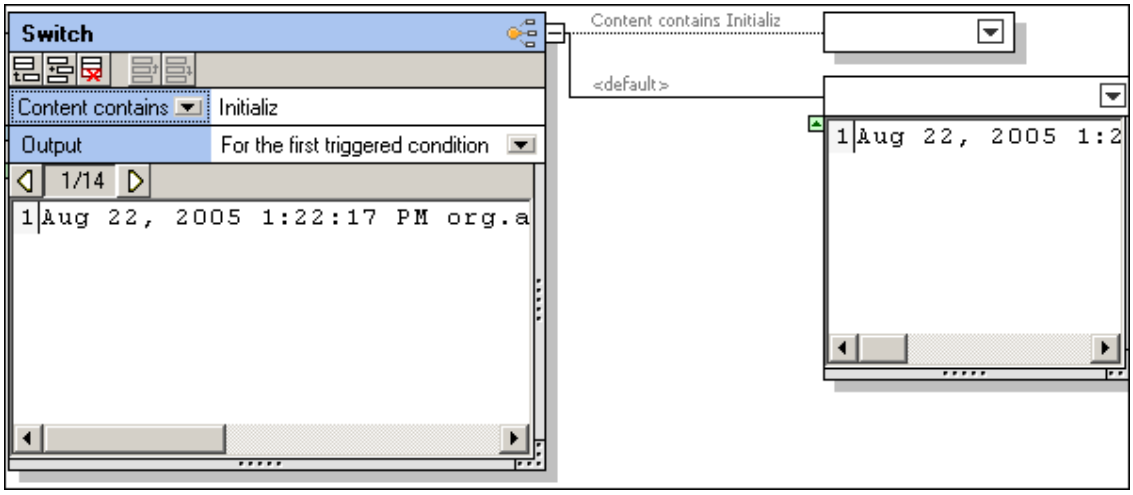
The repeated split container, using delimited (line based), separates all INFO sections out of the log file and passes them on to the Switch container.

1. Click the append icon  to add a new condition to the Switch container.
2. Double click in the "**Content starts with**" field, enter "**Initializ**" and hit Return.

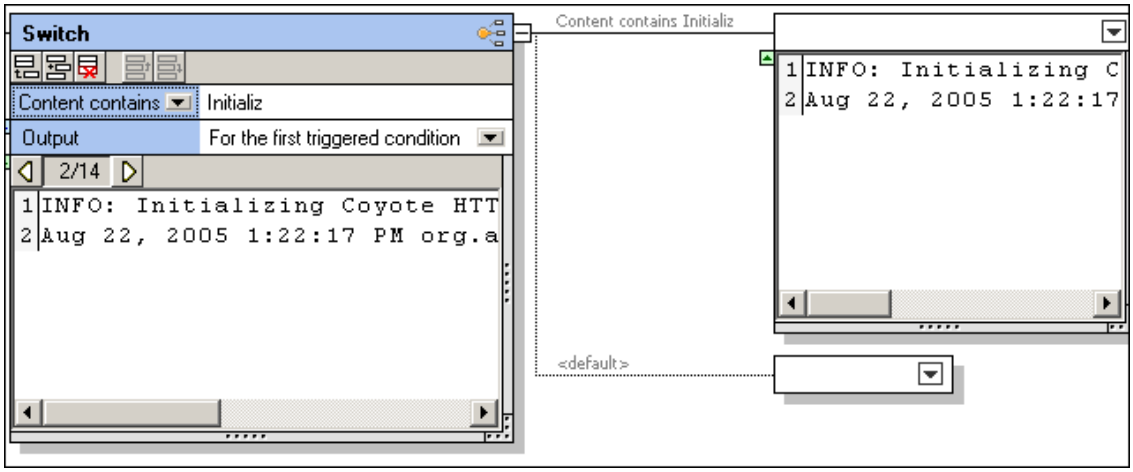


A new container is added. Data will be forwarded to this container if the condition is true. If not, the data is forwarded to the default container.

3. Click the "**Content starts with**" combo box, and change it to "Content contains".
The first condition has now been defined and you can see the result below.
The first fragment does not contain "Initializ", and its contents are therefore forwarded to the **default** container.

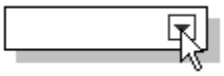


4. Click the Display next block icon , to see the next text fragment.

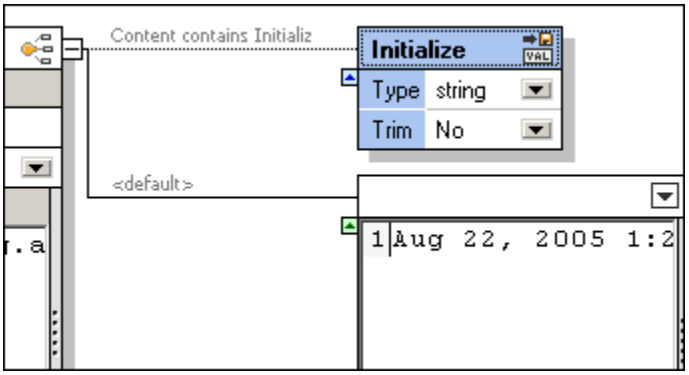


The Initializing... fragment now appears in its associated container, and the default container is empty. Stepping through the fragments gives you a preview of what the individual containers hold.

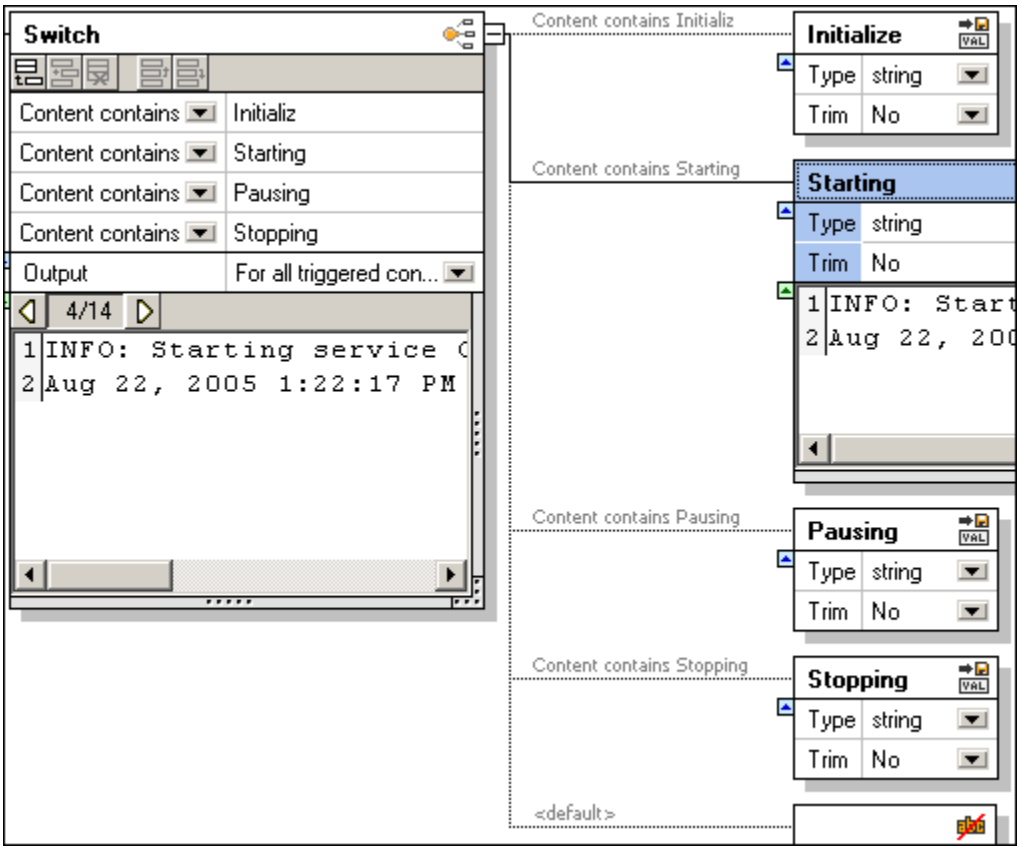
5. Click the container icon button, and select **Store as value**.



6. Double click in the "Store" title bar and change the text e.g. Initialize.

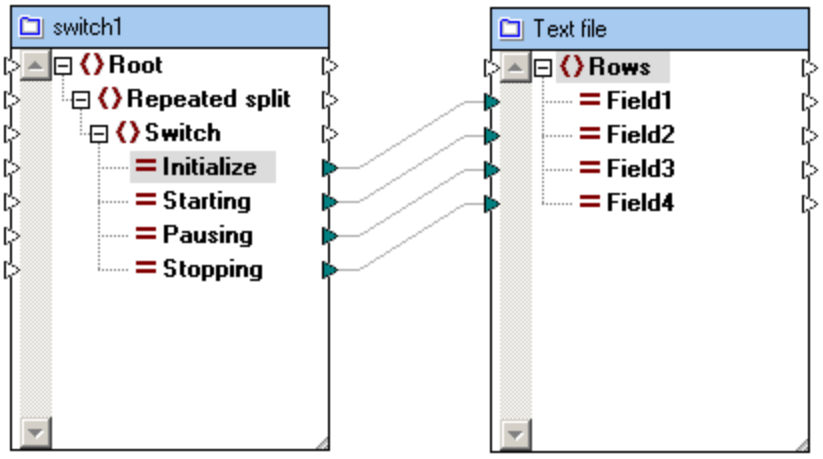


7. Click the append icon  to add a new condition to the Switch container.
8. Double click in the "**Content starts with**" field, enter "Starting" and hit Return.
   You can add as many conditions as you need e.g. Pausing, and Stopping.
   Give each of the associated containers a name, to make recognition in MapForce easier.



   The screenshot above shows all four conditions, and the contents of the "Starting" container at block/fragment no 4. The associated containers have all been renamed to make identification in the MapForce component easier.

   Note that conditions can be moved up and down in the condition list, using the respective Move Up/Down buttons  , or  .
9. Save the template and insert it in MapForce.



Please note:

If a text fragment in the current fragment satisfies a condition, then the **complete data** of that fragment is passed on to the associated container. Data is not split up in any way, it is just routed to the associated containers, or to the default container if it does not satisfy any of the defined conditions.

The associated containers produced by Switch, can be used for further processing. You can change such a container to Split once, Repeated split, or anything else if you wish.

Content starts with:
Data is only passed to the associated container, if the condition string appears at the start of the text fragment.

Content contains:
Data is passed on to the associated container, if the condition string appears anywhere in the text fragment.

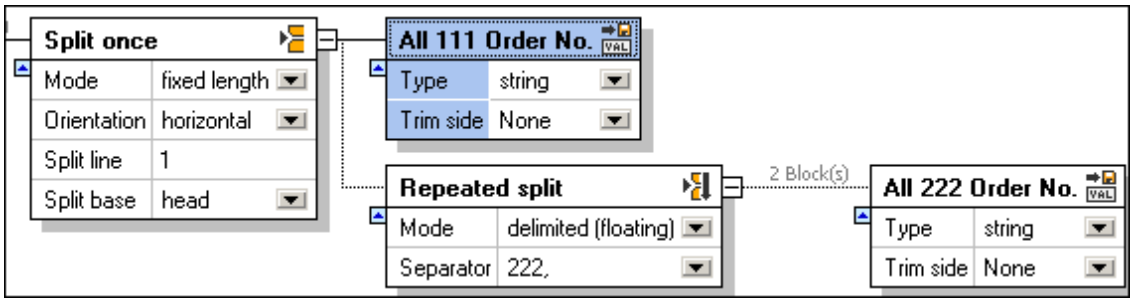
For the first triggered condition:
Data is passed on when **one** of the **conditions** in the condition list is **true**. Any other conditions that are true are ignored, and no data is passed on to any of the associated containers.

For all triggered conditions:
Outputs data for **every** condition that is true in the condition list. This makes it possible to have multiple occurrences of the same data/fragment in multiple associated containers simultaneously. This might occur if a text fragment contains text that satisfies two conditions simultaneously e.g. "initializing starting sequence" in the example above.

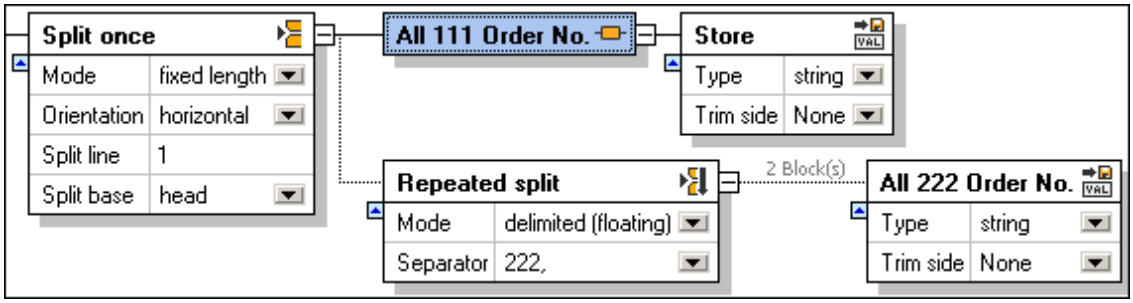
10.7.4 Node

Allows you to add a new hierarchical level to the FlexText, and MapForce tree structures. The data that the following node/container contains, is passed on as is.

In the screenshot below, the All 111 Order No. container is the last container in the top branch.



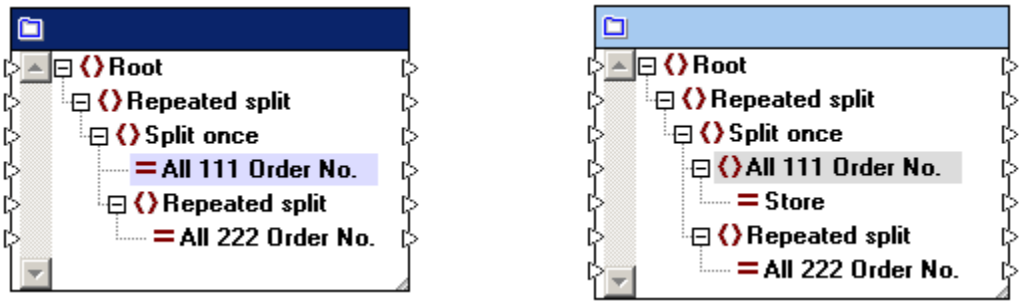
Click that containers icon, and select the Node option from the popup.



A new container has been added to the right of the current one.

Please note:
The automatically appended container was then manually defined as "**Store as value**".

The screenshot below shows both template structures as they appear when inserted into MapForce.

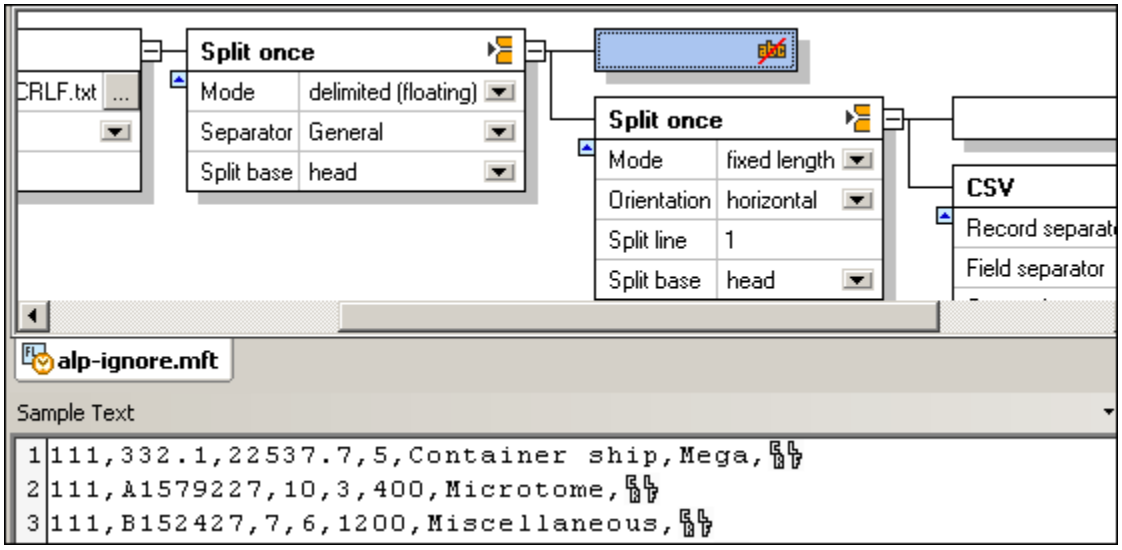


The left component shows the initial structure before adding the new Node.

The right component shows how the component structure has changed. "All 111..." is now a parent item, and a new child item "Store" has been added below it.

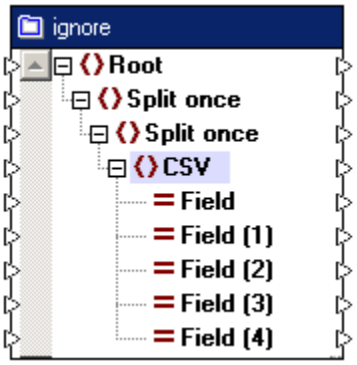
10.7.5 Ignore

Allows you to suppress the output of a specific text fragment. What this means, is that the container and any data it may contain, will not be made available as a mappable item in the FlexText component in MapForce.

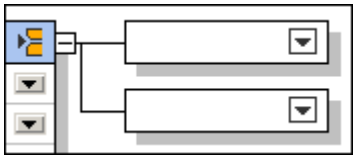


In the example shown above, the active container has been set to "Ignore". The Sample text that it contains will therefore not appear as a mappable item in MapForce.

The text template when inserted into MapForce, has the structure shown below. There is no mappable item between the two "Split once" items.



Please note:
Default "ignore" containers also exist. These are the new containers that are automatically appended when selecting "Split once" and "Repeated split" etc.



The contents of these containers are not initially mappable/available to MapForce when the template is inserted. You have to select one of the container options in FlexText: Store as value, Store as CSV etc., to be able to map them.

10.7.6 Store as CSV (separated)

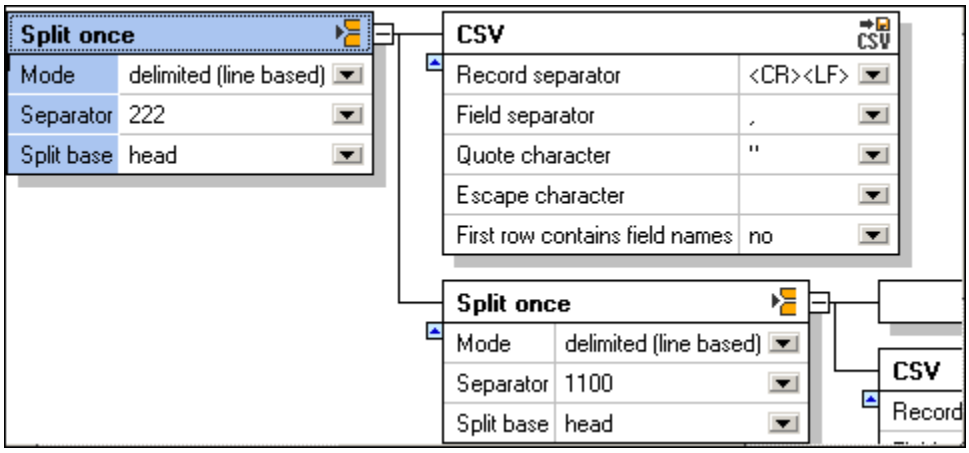
Store CSV allows you to store text fragments as CSV text, and map individual columns to MapForce. Any number of CSV containers/components can be created in FlexText, and each of the CSV containers may have different separators.

The Sample Text pane provides an overview of the current CSV fragment, and also allows you to specify individual field names, and field types. Each column appears as a mappable item in the FlexText component in MapForce.

Container **default settings** are:

Record separator	CR LF
Field separator	,
Quote character	"
Escape character	(none)
First row contains field names	no
Treat empty fields as absent	yes

The following example shows how data in a small text file is split up into two CSV files, and mapped to separate XML files in MapForce.



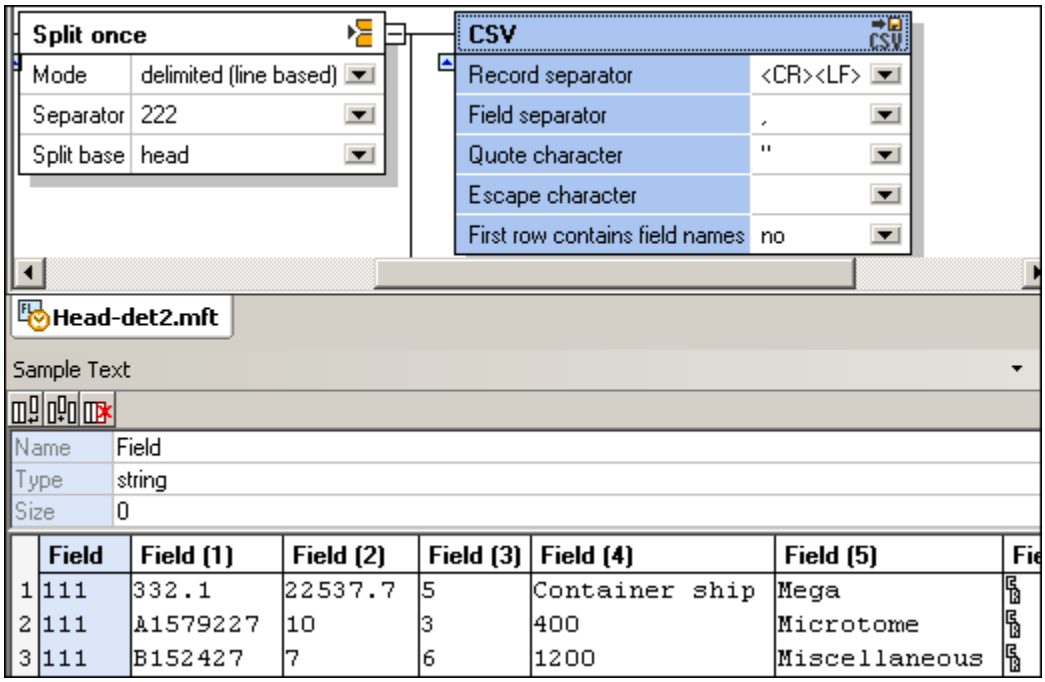
The **Split once** container shown above, is used to create two containers. The **delimited (line based)** function with the separator 222, is used to achieve this. All records up to the first occurrence of 222, are passed to the CSV container. The first, consisting of all records containing 111, is then defined as a CSV container. The Sample Text pane shows the contents of the currently active container "Split once".

Sample Text

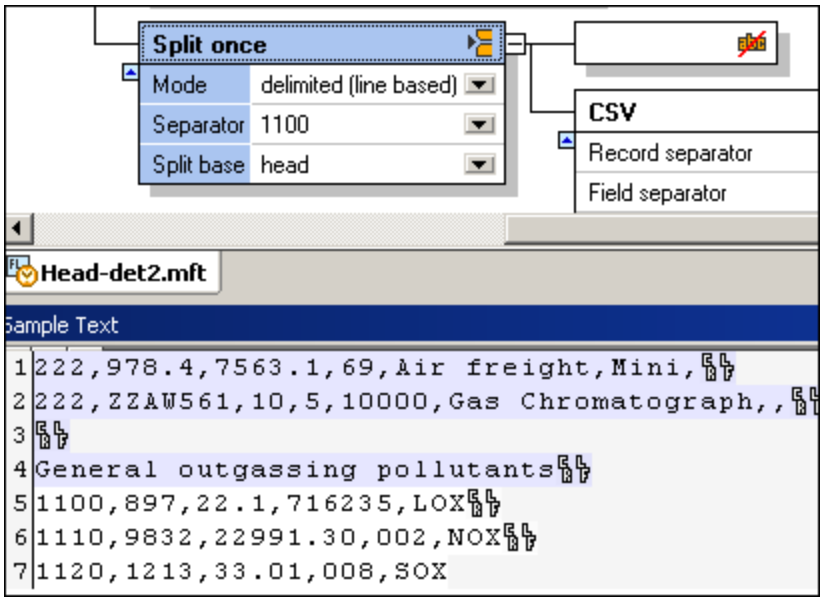
```
1 111,332.1,22537.7,5,Container ship,Mega,
2 111,A1579227,10,3,400,Microtome,
3 111,B152427,7,6,1200,Miscellaneous,
4 222,978.4,7563.1,69,Air freight,Mini,
5 222,ZZAW561,10,5,10000,Gas Chromatograph,,
6
7 General outgassing pollutants
8 1100,897,22.1,716235,LOX
9 1110,9832,22991.30,002,NOX
10 1120,1213,33.01,008,SOX
```

The default CSV settings have not been changed. Clicking the CSV container shows its

contents in tabular form.

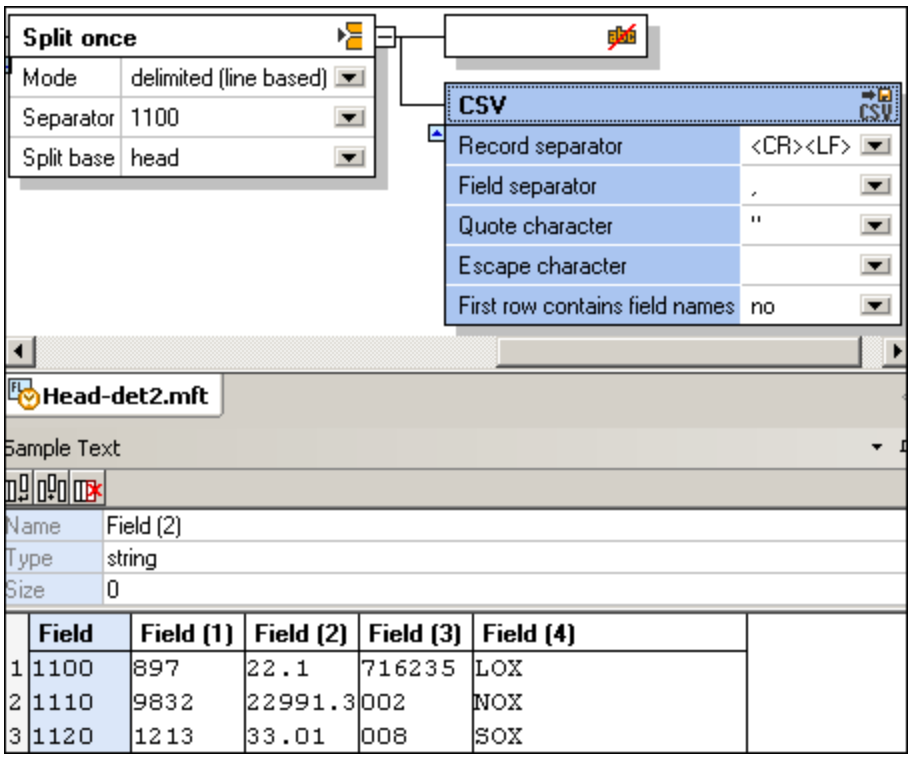


The second container holds the remaining data, and is made into another **Split once** container. This creates two more containers, one of which will be the second CSV. Clicking the Split once container, shows the current contents.

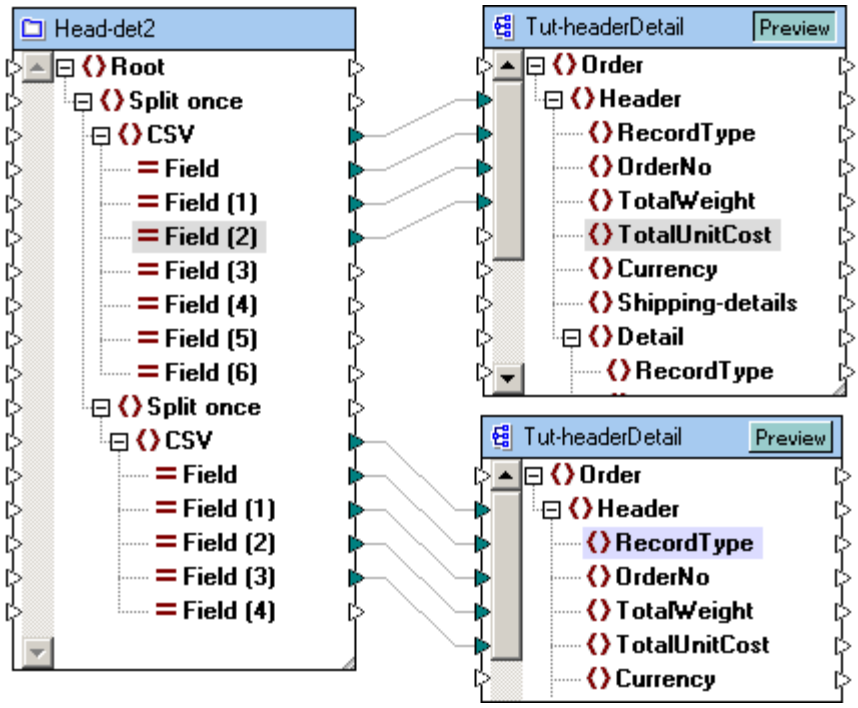


The delimited (line based) function, using 1100 as the separator, is used to split the remaining data into two sections.

- All records up to the first occurrence of 1100, are passed to the first container which is made non-mappable, by defining it as "Ignore" .
- The second container is then defined as CSV. The default settings have not been changed. Clicking the CSV container shows the contents in tabular form.



Inserting the FlexText template into MapForce allows you to map the data to any of the supported target files. In this example, each of the CSV items are mapped to two separate XML files.



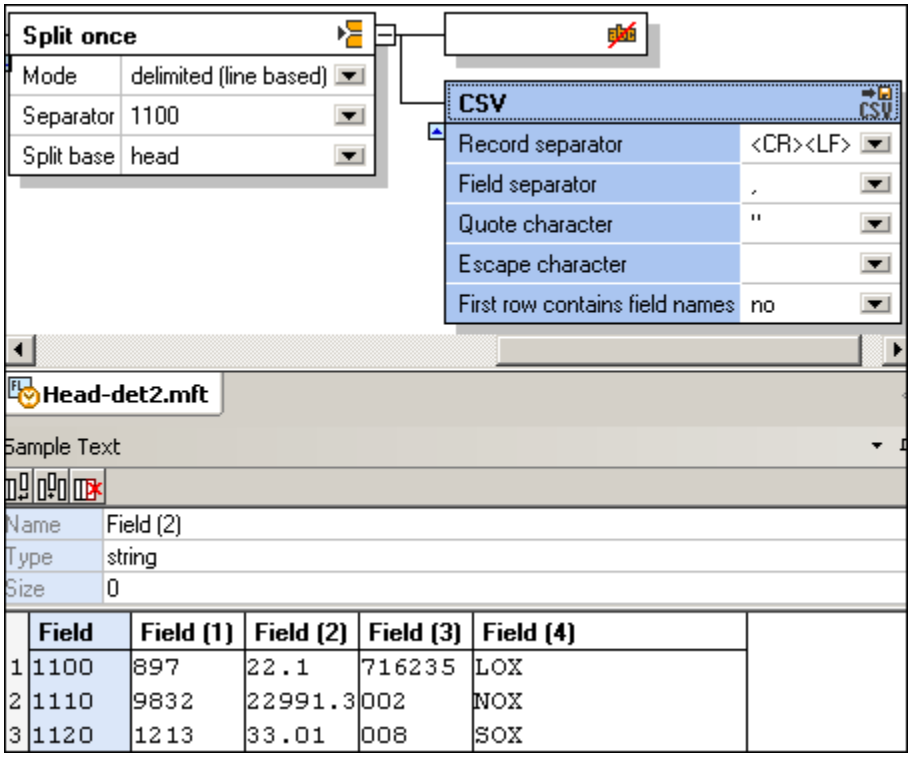
Note that not all of the items in the CSV sections are mapped to the target files. The first XML file contains all 111 record types.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>332</OrderNo>
    <TotalWeight>22537.7</TotalWeight>
  </Header>
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>0</OrderNo>
    <TotalWeight>10</TotalWeight>
  </Header>
  <Header>
    <RecordType>111</RecordType>
    <OrderNo>0</OrderNo>
    <TotalWeight>7</TotalWeight>
  </Header>
</Order>
```

The second XML file contains all records starting with 1100.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Header>
    <RecordType>1100</RecordType>
    <OrderNo>897</OrderNo>
    <TotalWeight>22.1</TotalWeight>
    <TotalUnitCost>716235</TotalUnitCost>
  </Header>
  <Header>
    <RecordType>1110</RecordType>
    <OrderNo>9832</OrderNo>
    <TotalWeight>22991.3</TotalWeight>
    <TotalUnitCost>2</TotalUnitCost>
  </Header>
  <Header>
    <RecordType>1120</RecordType>
    <OrderNo>1213</OrderNo>
    <TotalWeight>33.01</TotalWeight>
    <TotalUnitCost>8</TotalUnitCost>
  </Header>
</Order>
```

Configuring the CSV container/data:



Clicking a field in the Sample Text pane highlights it, allowing you to configure it further.

- Click in the **Name** field to edit the default text that is presented.
- Click in the **Type** field to define the field datatype: string, boolean, decimal etc.
- Click the append icon to append a new field.
- Click the insert icon to insert a field before the currently active field.
- Click the delete icon to delete the currently active field.

Please note:
The field boundaries can be dragged by the mouse to display the data.

Treat empty fields as absent
Allows you to define that empty fields in the source file, will not produce a corresponding empty item (element or attribute) in the target file.

Note that the delimiters for the empty fields in the source file must exist however, e.g. "General outgassing pollutants,,,,".

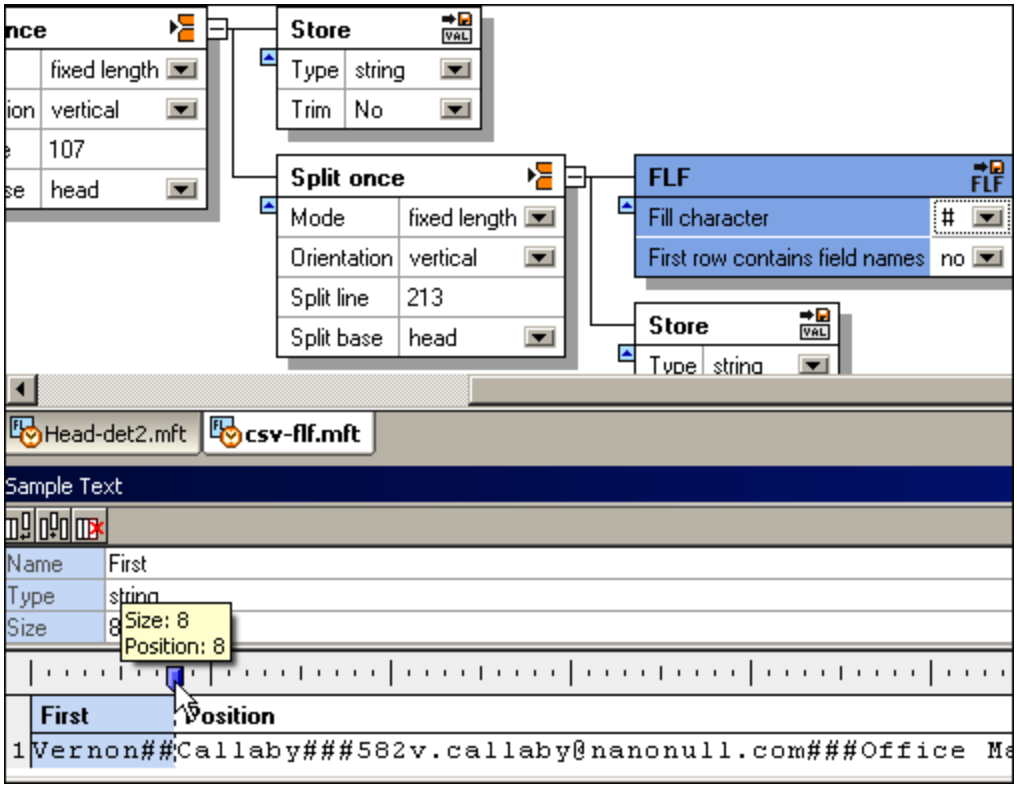
10.7.7 Store as FLF (delimited)

Store FLF allows you to store text fragments as fixed length text, and map individual columns to MapForce. Any number of FLF containers/components can be created in FlexText, and each of the FLF containers may have different fill characters.

The Sample Text pane provides an overview of the current FLF fragment, and also allows you to specify field names, lengths, and widths. Each column appears as a mappable item in the text component in MapForce.

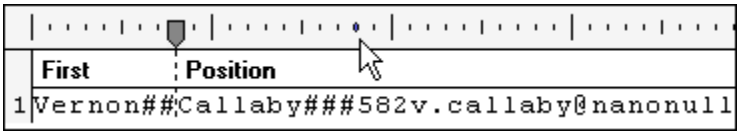
Container **default settings** are:

Fill character	(none)
First row contains field names	no
Treat empty fields as absent	yes

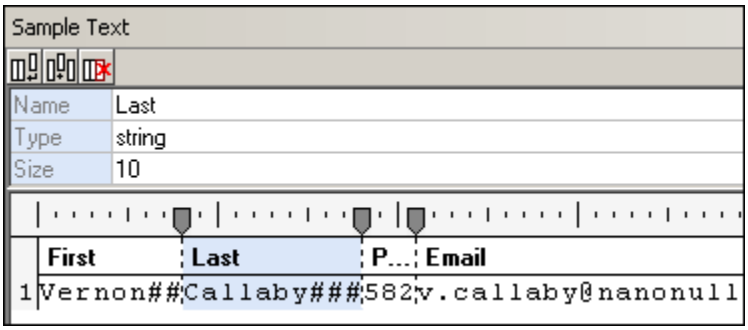


Configuring the FLF container/data:
Having defined a container as "Store FLF", the Sample Text pane appears as shown in the screenshot above. A default field of width 10 is automatically inserted.

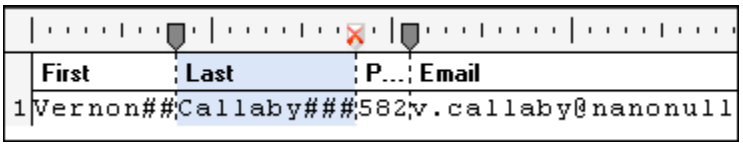
- Click the tab icon on the ruler and drag, to reposition it. A popup appears showing you the current position.
- Positioning the cursor over the ruler displays a "dot"; clicking places a new tab at the click position.



- Having defined the new position, click the field to select it, and edit the name in the Name field.



- To **remove** a field, click the tab icon and drag it off the ruler. The tab icon changes when this action can be successfully completed.

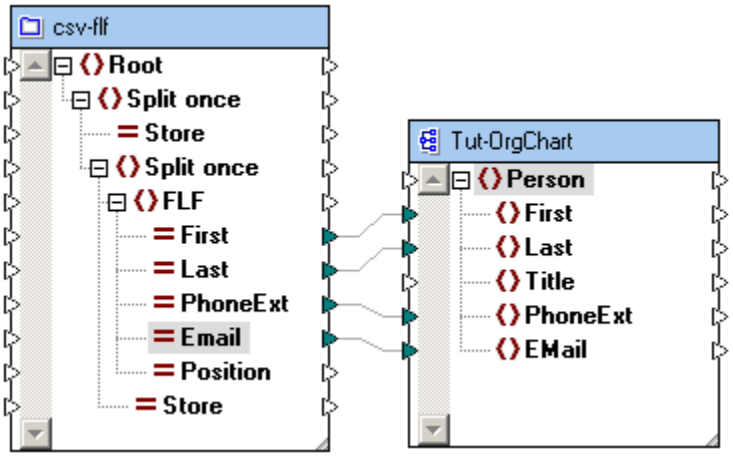


Please note:
Dragging a tab on the ruler, automatically repositions all tabs to the right of it. To retain the other tab positions, hold down SHIFT before moving the tab.

Clicking a field in the Sample Text pane highlights it, allowing you to further configure it.

- Click the append icon to append a new field, of length 10.
- Click the insert icon to insert a field before the currently active field, length 10.
- Click the delete icon to delete the currently active field.
- Click in the **Name** field to edit the default text that is presented.
- Click in the **Type** field to define the field datatype: string, boolean, decimal etc.

Inserting the FlexText template into MapForce allows you to map the data to any of the supported target files. In this example, FLF items are mapped to XML items.



Treat empty fields as absent
Allows you to define that empty fields in the source file, will not produce a corresponding empty item (element or attribute) in the target file.

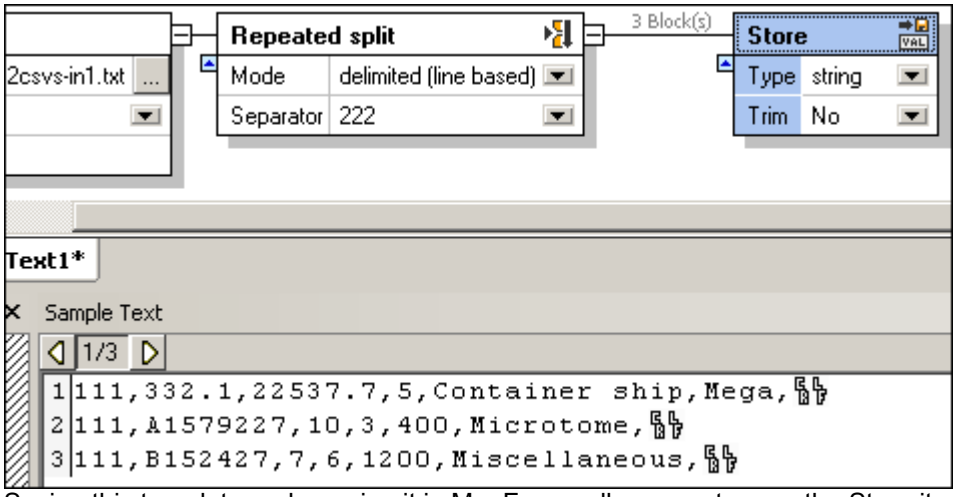
Note that the delimiters for the empty fields in the source file must exist however, e.g. "General outgassing pollutants**,,,,**".

10.7.8 Store value

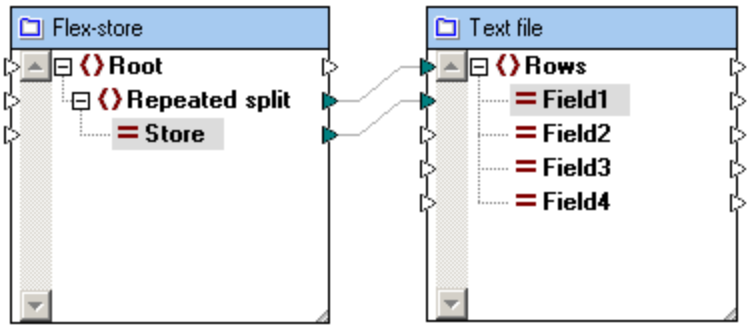
Allows you to define a container, which makes its data available as a mappable item, in MapForce. If you do not change the container name in FlexText, then the mappable item appears with the name "Store".

Container **default settings** are:
Type string
Trim no

The screenshot below shows the "Store" container with its contents visible in the Sample Text pane.



Saving this template and opening it in MapForce, allows you to map the Store item to other items in a target component.



Please note:
The field1 item in the target text file, will contain all 3 fragments supplied by the Store item, when you click the Output tab to preview the result.

Type
Allows you to define the datatype of the text fragments.

Trim side
Defines the side from which the characters will be trimmed, left, right or both. Selecting Yes, activates the "Trim character set" option.

Trim character set
Defines the characters you want to trim from this text fragment. You can enter any number of

characters here, by double clicking in the field. The characters you enter are removed from the Trim side(s) of the fragment.

Chapter 11

MapForce and EDI

11 MapForce and EDI

MapForce supports the Electronic Data Interchange formats UN/EDIFACT, and ANSI X12, and allows you map data to, and from these EDI documents. Please note that you need to select one of the programming languages (Java, C#, or C++) as the mapping output, to be able to work with EDI files.

UN/EDIFACT
is a de-facto financial industry standard for document interchange (also a UN standard). MapForce supports the messages contained in directories **93A - 07B**, with **2007B** as default of the UN/EDIFACT standard. There are approximately 200 different message types in this directory.

Please see: http://www.unece.org/trade/untdid/welcome.htm for more information on directory downloads, service codes etc.

UN/EDIFACT 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.

ANSI X12
is an industry standard for document interchange. MapForce supports versions: 3040, 3050, 3060, 3070, 4010, 4020, 4030, 4040, 4041, 4042, 4050, 4051, 4052, 4060, 5010, 5011, 5012, 5020 and 5030, with the default being **5030** of the **ANSI X12** specification. Please see http://www.x12.org/ for information on X12 publications.

ANSI X12 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.

11.1 EDI Terminology

The following short list describes the main UN/EDIFACT terms and their counterparts in ANSI X12, the US related standard.

Messages (ANSI X12 - Transactions)
A single EDI document is a message or transaction, and is defined as a group of segments in a standard sequence.

Segment
A single "record" contained in a message.

Segments are identified by a two or three character ID at the beginning of the segment. A group of related elements comprise a segment tag (or segment ID - ANSI X12). Segments of a transaction can be defined as mandatory or conditional (optional).

Element
An individual data field within a segment. An element can be thought of as a field, i.e. it contains one type of data, a name, or an address, for example. Elements can be further subdivided into composite elements, consisting of component elements or subelements.

Separators
Elements are delimited by "separator characters". In UN/EDIFACT these are either default characters, or defined in an optional UNA control segment.

Default characters

colon	**:**	component element separator
plus	**+**	data element separator
apostrophe	**'**	segment terminator

Message envelope
These are special "Service segments" (ANSI X12 - control segments) known as the envelope header and envelope trailer pair. In EDIFACT these are defined as UNH and UNT segments.

If several transactions of the same type are sent to the same recipient, they can be grouped into a **Group Envelope** defined by a UNG-UNE segment pair. These messages thus have the same Group ID.

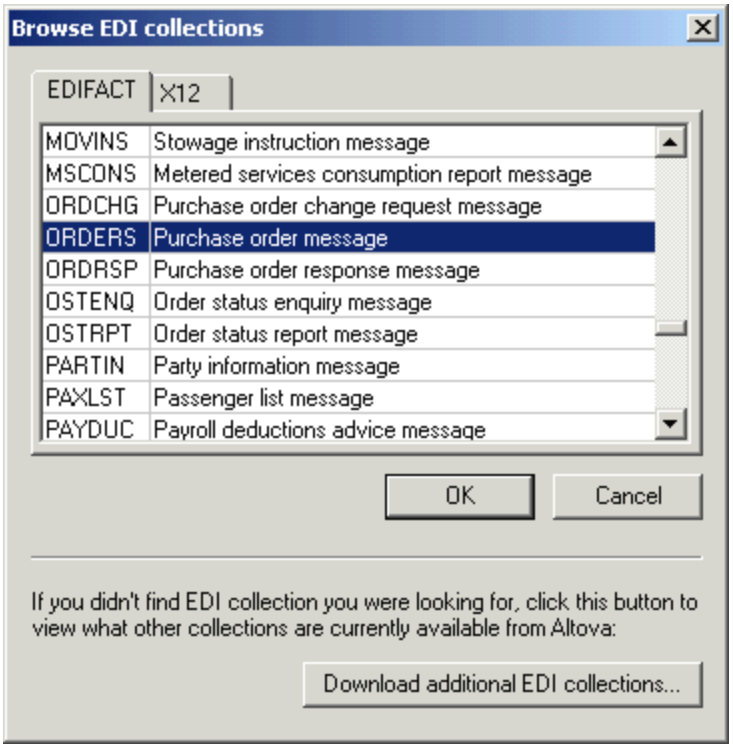
Interchange envelope
These are a collection of Group envelopes and/or messages for the same recipient.

11.2 UN/EDIFACT to XML Schema mapping

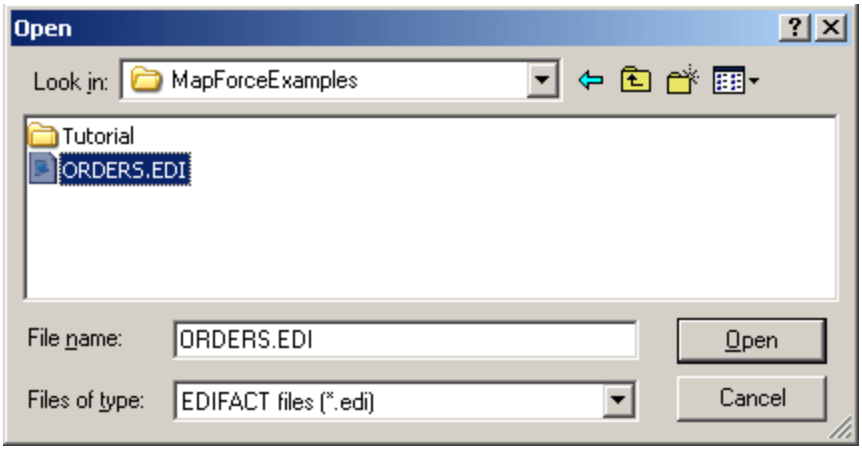
The object of this exercise is to show how map data from UN/EDIFACT messages, to an XML Schema/document to produce an XML file for further processing. The mapping discussed in this example, is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** directory as **EDI_Order.mfd**.

Creating the EDIFACT component in MapForce:

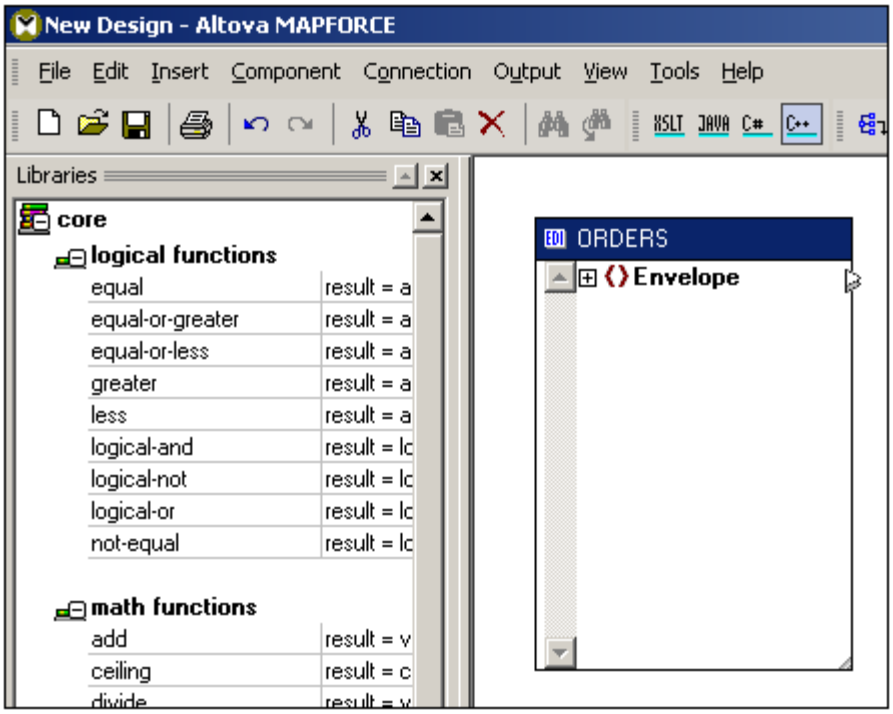
1. Create a new project and ensure that one of the programming language icons is active, i.e. Java, C#, or C++. It is not possible to generate XSLT 1.0 / 2.0 or XQuery code when mapping from EDI files.
2. Select the menu option **Insert | EDI**, or click the EDI icon, and select **ORDERS** from the list of EDI collections, then click OK.



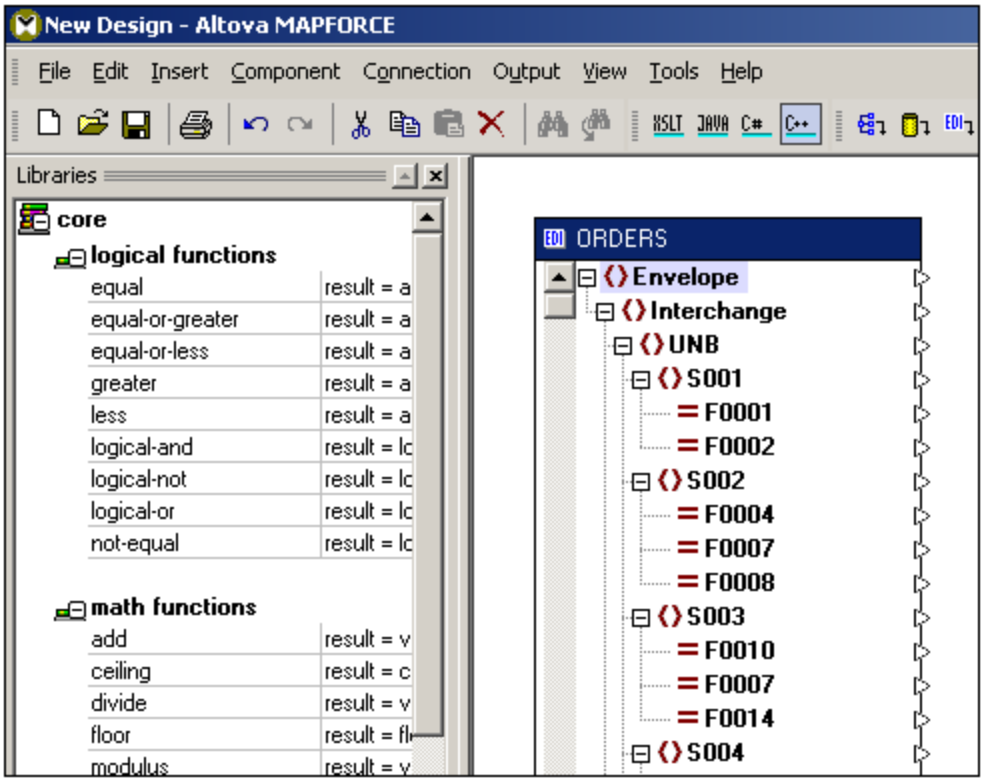
3. Select ORDERS.EDI from the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** directory and click on Open.



4. The EDI component now appears in the Mapping area.

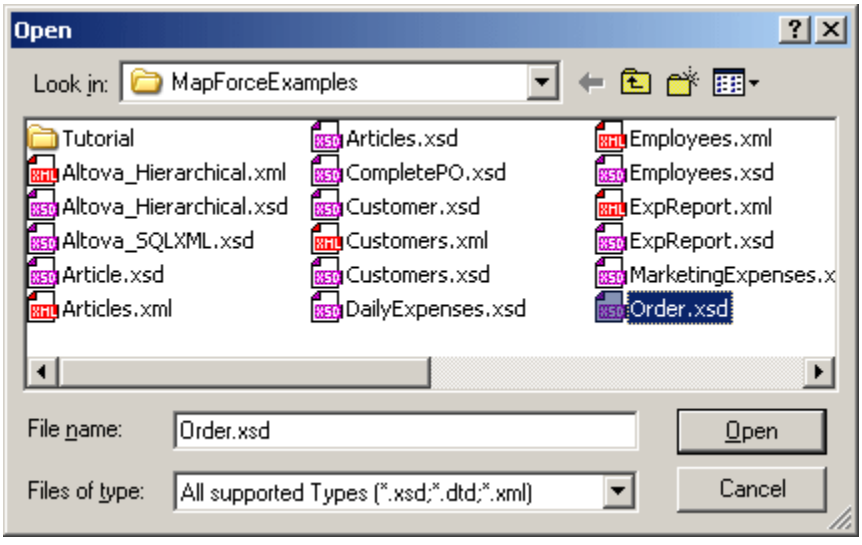


5. Click the **Envelope** entry and hit the * key on the numeric keypad to view all the items.
6. Click the **expand** icon at the lower right of the component window, and resize the window.

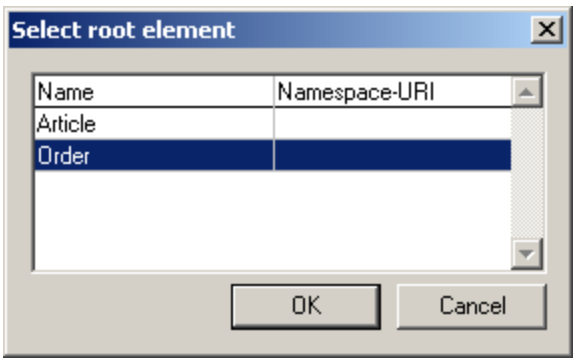


Creating the target schema component:

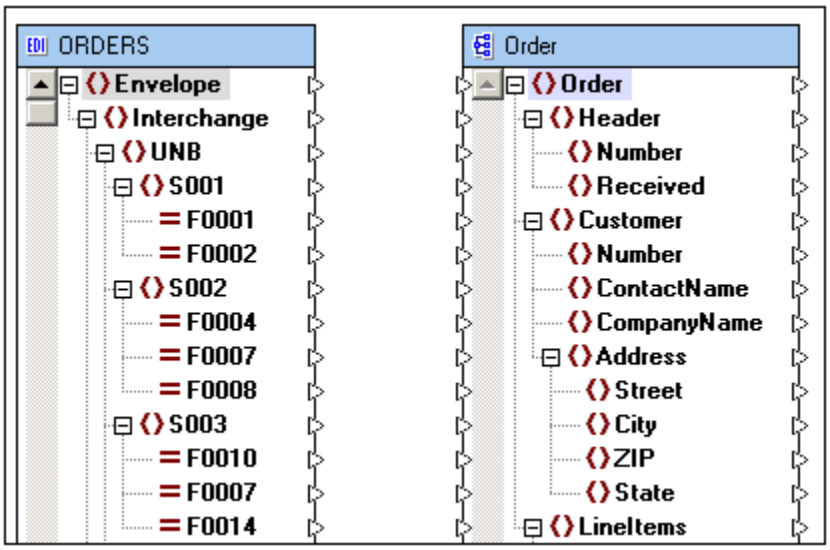
1. Click the **Insert XML Schema/File** icon, and select **Order.xsd** from the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** directory.



2. When prompted to supply an example file, click **Skip** and select **Order** as the root of the target document.



3. Click the **Order** entry and hit the * key on the numeric keypad to view all the items.
4. Click the **expand** icon at the lower right of the component window, and resize the window.



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

Mapping the EDI items:

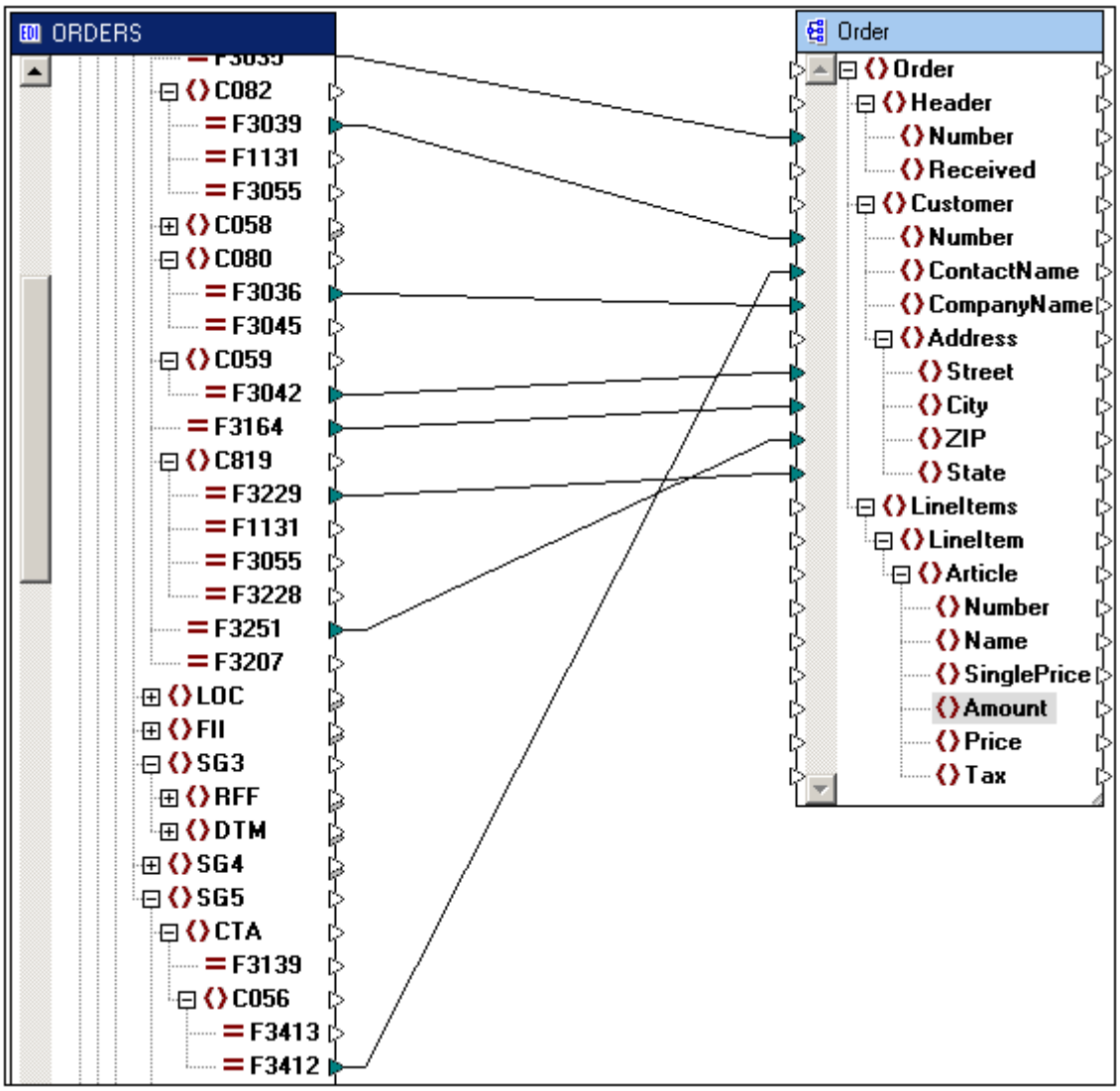
The EDI component now shows us the structure of a message, based on the collection (ORDERS) we selected.

Typically, not all of the nodes will actually contain data, so the project author must be sufficiently familiar with the EDI documents being worked on, to locate the relevant nodes. In this case, the following nodes (starting from the Group/Message node) need to be mapped directly:

BGM_C106_1004 -> Order.Header.Number

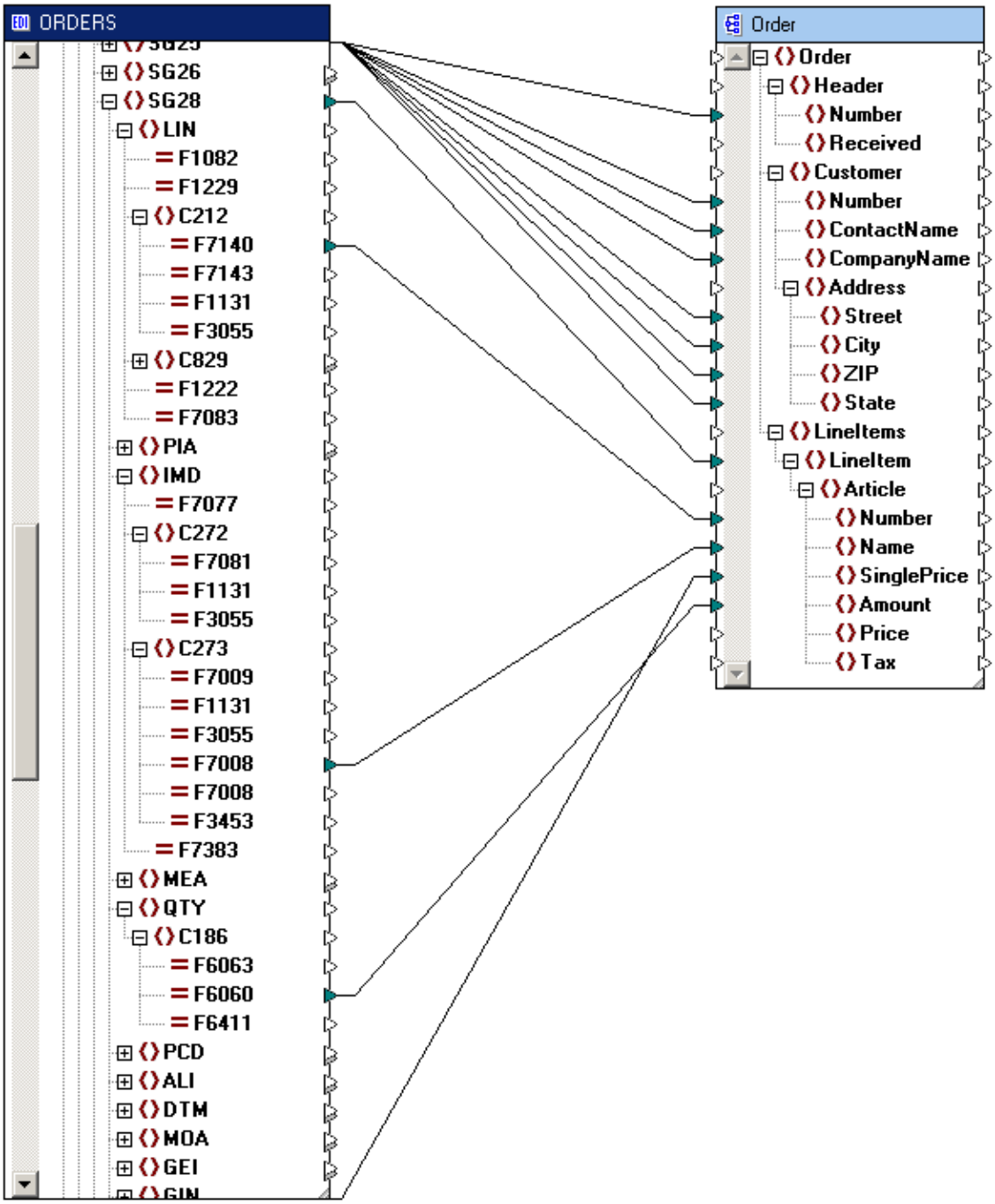
SG2_NAD_C082_3039 -> Order.Customer.Number
SG2_NAD_C080_3036 -> Order.Customer.CompanyName
SG2_NAD_C059_3042 -> Order.Customer.Address.Street
SG2_NAD_3164 -> Order.Customer.Address.City
SG2_NAD_C819_3229 -> Order.Customer.Address.State
SG2_NAD_3251 -> Order.Customer.Address.ZIP
SG2_SG5_CTA_C056_3413 -> Order.Customer.ContactName

At this stage, the mapping should look similar to the graphic below:



Continue the mapping process and map:

```
SG2U                    -> lrderLLineftems
SG2ULLfNLC2N2LF7NQ0     -> lrderLLineftemsLLineftemLArticleLNumber
SG2ULfMDLC27PLF700U     -> lrderLLineftemsLLineftemLArticleLName
SG2ULQqYLCNUSLFS0S0     -> lrderLLineftemsLLineftemLArticleLAmount
SG2ULSGP2LPRfLCR0VLFRNNU->
                        lrderLLineftemsLLineftemLArticleLSinglePrice
```

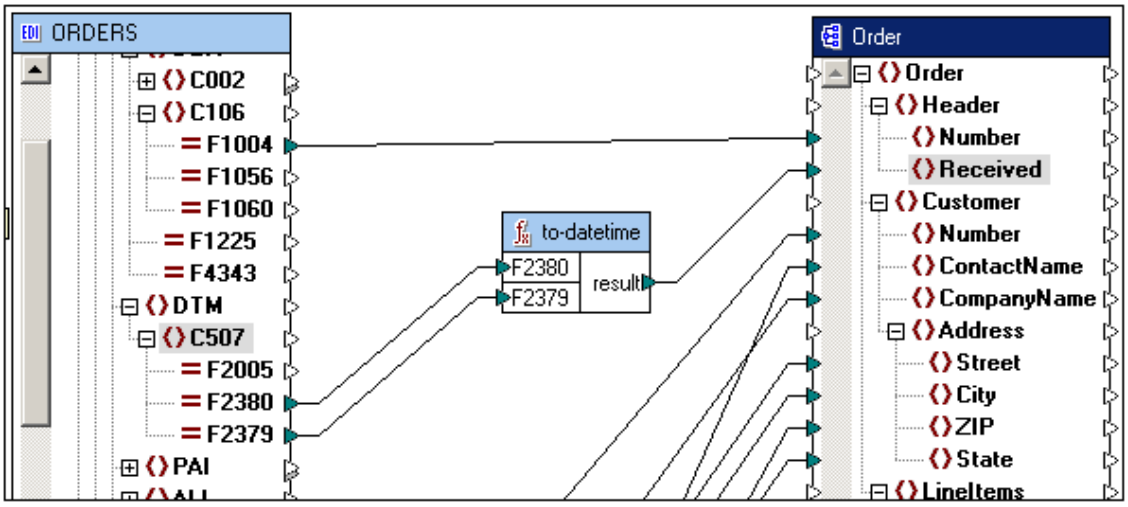



Drag a **to-datetime** function from the **edifact** library into the Mapping area.

By applying the F2380 and F2379 components of the **DTM/C507** element we can create an appropriately formatted **Received** datetime.

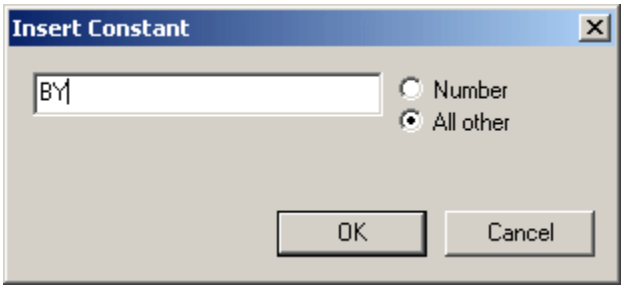
We therefore map the following fields:

DTM/C507/F2380	-> the F2380 input of the to-datetime function
DTM/C507/F2379	-> the F2379 input of the to-datetime function
The **result** of the **to-datetime** function	-> Order/Header/Received



Filtering out the Buyer purchase orders:
At this point we want to filter out the "Buyer" purchase orders. These can be identified by the party function code qualifier of the NAD (Name and address) segment. In this case, the value 'BY' indicates a "Buyer" (Party to whom merchandise and/or service is sold).

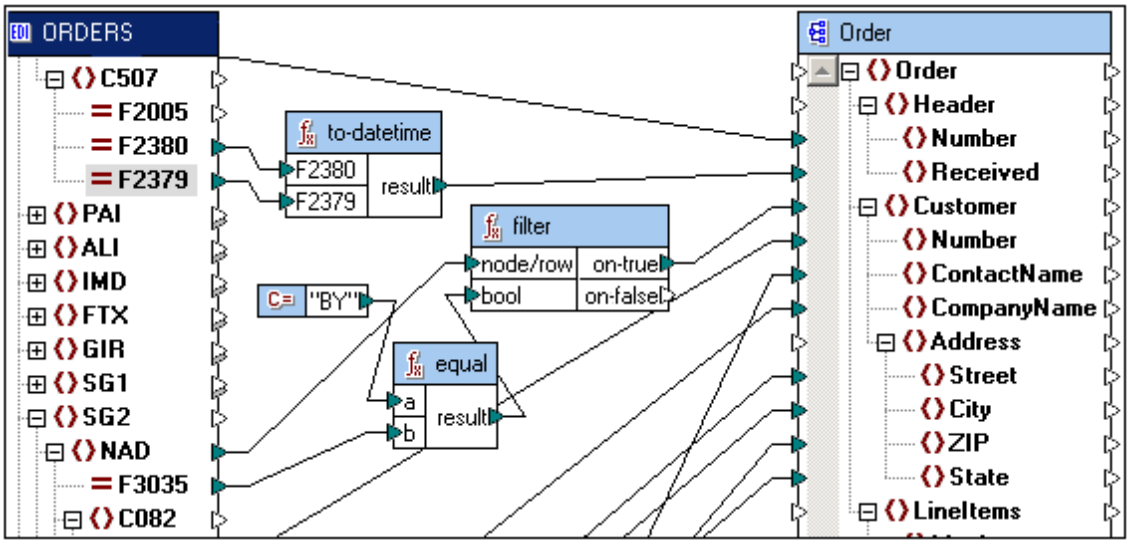
1. Drag an **equal** function from the **core** library, into the Mapping area.
2. Select the menu option **Insert | Filter: Nodes/Rows**.
3. Insert a Constant component using **Insert | Constant**, click "All other" and type 'BY' into the text field:



Map the following items:

SG2/NAD/F3035	-> the **b** input of the **equal** function
The **Constant** "BY"	-> the **a** in put of the **equal** function
The **result** of the **equal** function	-> the **bool** input of the **filter** function
SG2/NAD	-> the **node/row** input of the **filter** function
The result of the filter function	-> **Order/Customer** in the schema

The aim here is, to only map data if the NAD node refers to a 'Buyer', as identified by the party function code qualifier 'BY'.



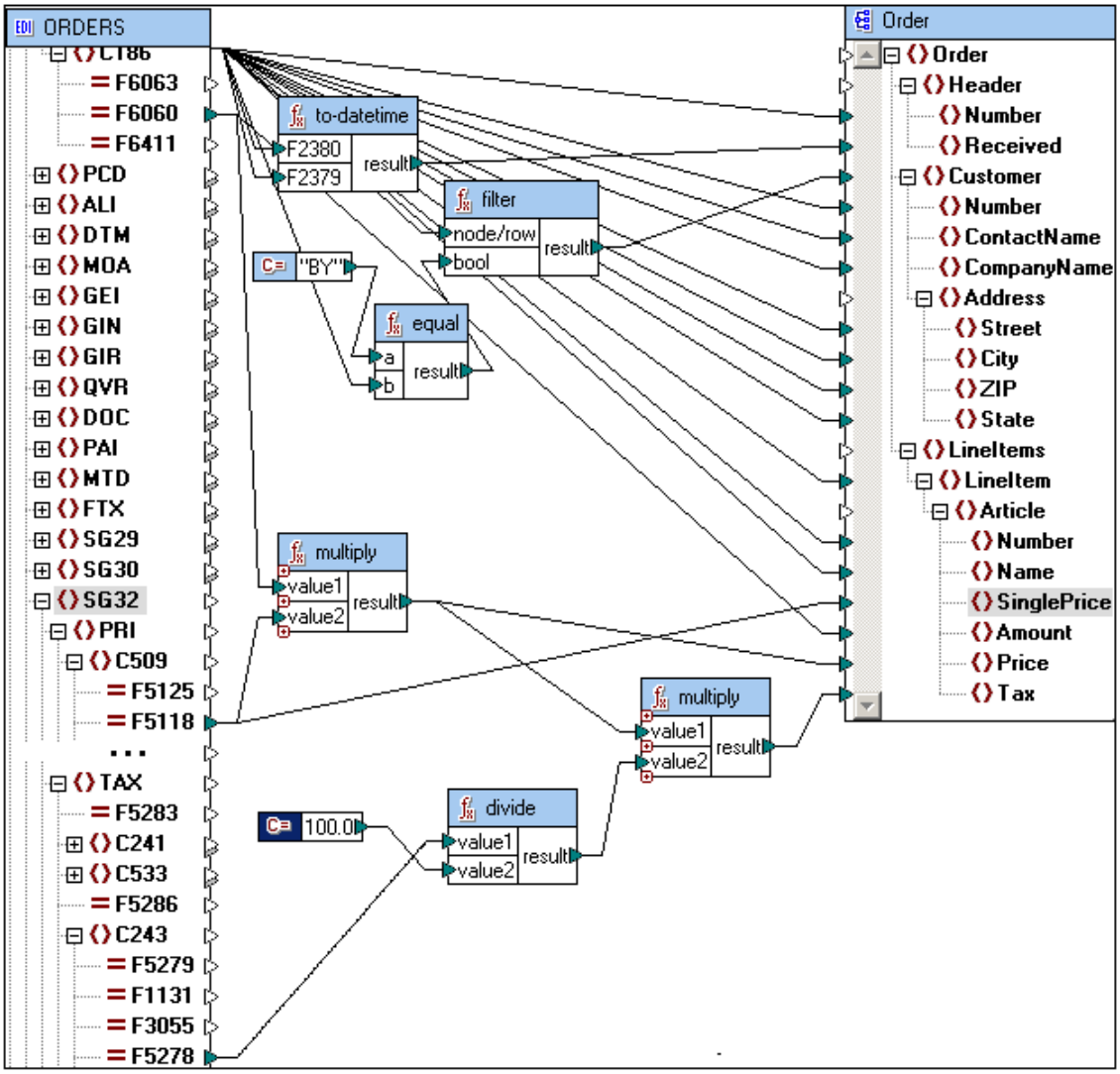
The final step in this task is to calculate the pricing and tax costs.

1. From the core library, drag two **multiply** and one **divide** function into the Mapping area.
2. Insert a Constant function (**Insert | Constant**) make sure Number is selected, and enter 100.0 into the text field.
3. Map the following items:

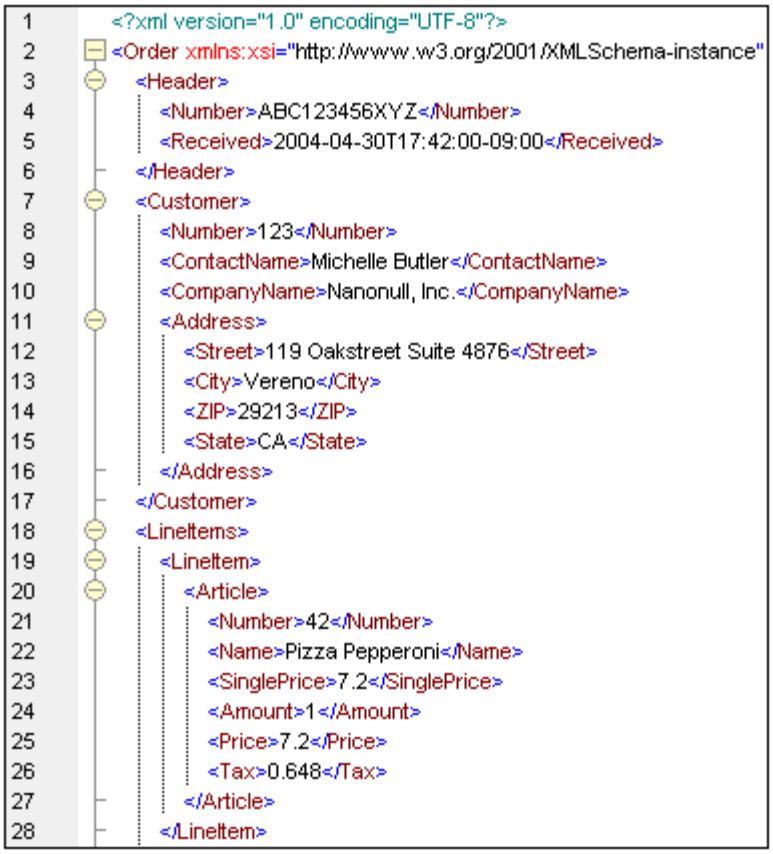
SG28/QTY/C186/F6060	-> **value1** of the first **multiply** function
SG28/SG32/PRI/C509/F5118	-> **value2** of the first **multiply** function
The **result** of the first **multiply** function	-> Order/LineItems/LineItem/Article/Price
SG28/SG38/TAX/C243/F5278	-> **value1** of the **divide** function
The **Constant** "100.0"	-> **value2** of the **divide** function

The **result** of the first **multiply** function	-> **value1** of the second **multiply** function
The **result** of **divide** function	-> **value2** of the second **multiply** function
The result of the second multiply function	->Order/LineItems/LineItem/Article/Tax

The mapping in your mapping should look like this:



Clicking the output tab performs an on-the-fly-transformation and presents you with the XML document result:



```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Header>
    <Number>ABC123456XYZ</Number>
    <Received>2004-04-30T17:42:00-09:00</Received>
  </Header>
  <Customer>
    <Number>123</Number>
    <ContactName>Michelle Butler</ContactName>
    <CompanyName>Nanonull, Inc.</CompanyName>
    <Address>
      <Street>119 Oakstreet Suite 4876</Street>
      <City>Vereno</City>
      <ZIP>29213</ZIP>
      <State>CA</State>
    </Address>
  </Customer>
  <LineItems>
    <LineItem>
      <Article>
        <Number>42</Number>
        <Name>Pizza Pepperoni</Name>
        <SinglePrice>7.2</SinglePrice>
        <Amount>1</Amount>
        <Price>7.2</Price>
        <Tax>0.648</Tax>
      </Article>
    </LineItem>
```

11.3 UN/EDIFACT and ANSI X12 as target components

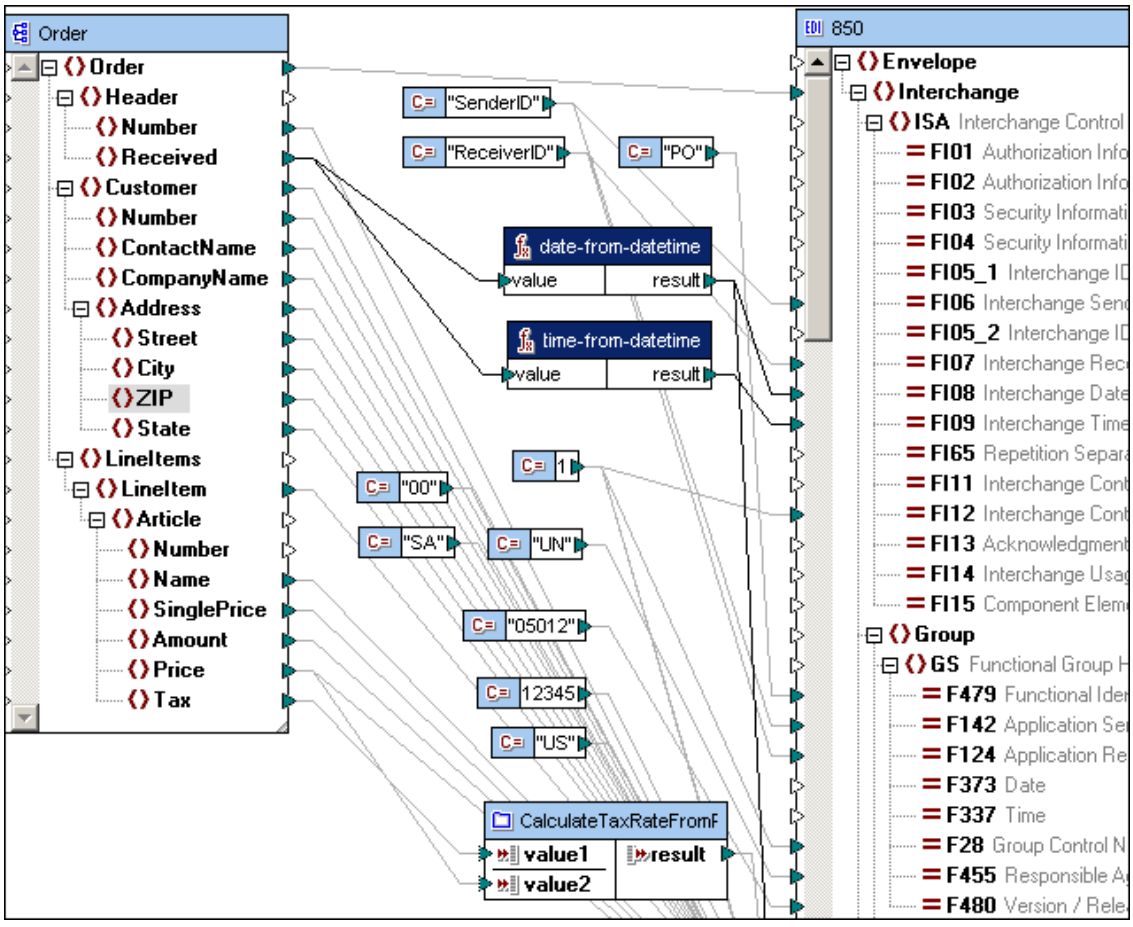
The **XML_To_X12_Order.mfd** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder maps an XML file to an ANSI X12 EDI file. The resulting EDI output file can be validated against the X12 specification by clicking the "Validate output file" icon.

The following sections describe:

- the settings common to both EDI formats
- the validation rules and automatic data completion settings used for each of the specific EDI formats.

Please note:

If it is important to retain the element/item sequence present in the source component, please make sure you create connectors using the "Source Driven (mixed content)" option, see the section on Source driven and mixed content mapping for more information.



Validation

Validation is only available in the Output tab, and not when generating code. Select the menu item "**Output | Validate output XML file**" to validate the EDI Output file.

If we regard the elements of an EDI document as nodes, we can differentiate between those having children and those without. Childless nodes, so-called leaf nodes, contain data; nodes having children structure data into logical groups.

Two properties are checked during validation of **leaf nodes**:

- whether the data string is less than, or of equal length, to that specified in the configuration file and,
- if the datatype is set to decimal, i.e. whether the string contains data which can be interpreted as a decimal according to EDIFACT conventions.

Properties checked for **parent nodes** are:

- the child nodes that are to be expected (specified by the configuration files), and
- the min. and max. occurrence of each child node.

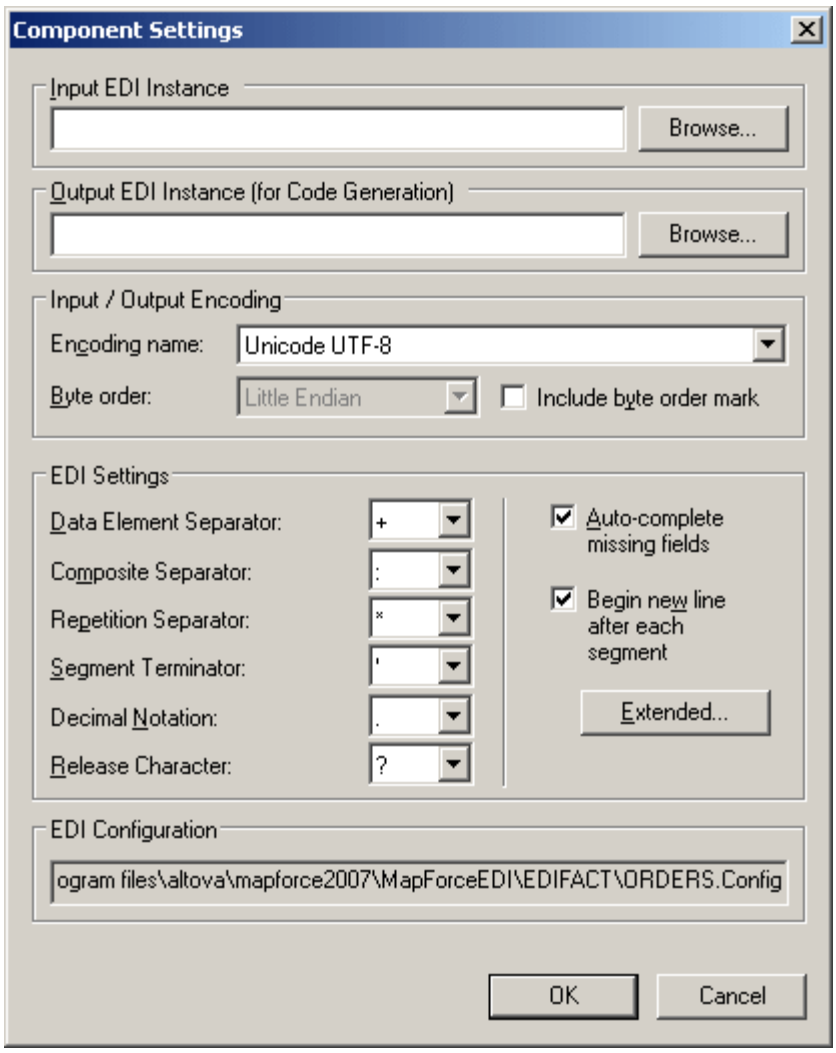
Automatic data completion
Automatic data completion does not generally change existing data. It does, however, create (specific) mandatory nodes and inserts data where necessary, after the mapping process, to produce a valid document.

Please note that fields not listed in the "Automatic data completion" sections that follow, are NOT inserted, or created. The correct values cannot be ascertained automatically.

Generic Settings:
Several settings can be defined that are applicable to all EDI documents:

- Auto-complete missing fields:
  should auto-completion be enabled or not. Default: **true**
- Begin new line after each segment:
  should a new line be appended to each segment for improved readability. The EDI standard ignores these lines if present in a message. Default: **true**
- Data element separator: see EDIFACT/X12 specification.
  Default: **+**
- Composite Separator: see EDIFACT/X12 specification.
  Default **:**
- Segment Terminator: see EDIFACT/X12 specification.
  Default **'**
- Decimal Notation: see EDIFACT/X12 specification.
  Default **.**
- Release Character: see EDIFACT/X12 specification.
  Default **?**

Right click an EDI component and select **Component Settings** to open the EDI component settings dialog box. The UN/EDIFACT settings are used as defaults for both EDI components.



Clicking the Extended... button, opens the respective extended EDI settings dialog box.

Defining your own separators (can be non-printable):
You can use any characters outside the **a-z, 0-9** range, in the first three "EDI Settings" combo box group in this dialog box.

1. Use the Character Map application to select and copy the specific character into the system clipboard.
2. Paste the character into one of the combo boxes.

Mapping date and time datatypes
Prior to MapForce 2006R3, date and time were of type "string" in EDIFACT and X12 components. From this release on, date and time can be mapped directly to/from **xsd:time**, or **xsd:date** (also from SQL date/time).

User defined functions prior to the R3 version that convert xsd:date, or xsd:time into an EDI format, generate an error message and cannot be used as they are. Please use the schema datatype **xsd:datetime**, which can be mapped using the built-in functions **date-from-datetime** or **time-from-datetime,** in the datetime library.

If you need to map dates within user-defined functions, please make sure that they adhere to the ISO 8601 date/time format i.e. YYYY-MM-DD.

11.3.1 UN/EDIFACT target - validation

The following items are checked when a UN/EDIFACT document is validated:

- Whether a UNB and a UNZ segment exist.
- Whether UNB/S004 contains a valid date/time specification.
- Whether UNB/0020 and UNZ/0020 contain the same value.
- Whether UNZ/0036 contains the correct number; which is defined as the number of functional groups, if present, or the number of messages. If there are functional groups, this should be the number of functional groups, otherwise it should be the number of messages contained in the interchange.

Each **functional group** is checked:

- Whether it contains a matching UNG and UNE pair.
- Whether UNG/S004 contains a valid date/time specification.
- Whether UNE/0060 contains the correct number of messages contained in the functional group.

Each **message** is checked:

- Whether it contains a matching UNH and UNT pair.
- Whether UNH/S009/0052 contains the same value as UNG/S008/0052 of the enclosing functional group.
- Whether UNH/0062 and UNT/0062 contain the same value.
- Whether UNH/S009/0065 contains the correct message type specifier.
- Whether UNT/0074 contains the correct number of segments contained in the message.

Automatic data completion for EDIFACT makes sure:

- a UNB and a UNZ segment exist
- That if either UNG or UNE exist, that the other ID also exists
- That a UNH and a UNT segment exist
- That UNB/S001 exists. If it does not contain data, the syntax level and syntax version number from the user-defined settings are used. See **Settings | Syntax version number**.
- That UNB/S002 and UNB/S003 exist.
- That UNB/S004 exists. If it does not contain data, the current date/time in EDI format is inserted.
- That UNZ/0036 exists. If it does not contain data, the number of functional groups or messages is calculated and inserted.
- That UNZ/0020 exists. If it does not contain data, the value from UNB/0020 is copied.

Please note:

Any fields not mentioned in this section (Automatic data completion) are NOT inserted, or created. The correct values cannot be ascertained automatically.

Given a (target) parent element A (in the target EDI component) with child items x, y, and z - where y is mandatory, parent element A will only be created in the output file if

the mandatory child element "y" in the target component has been mapped!

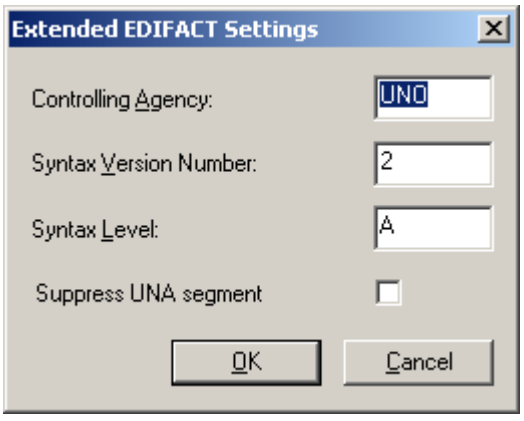
Functional group checking makes sure:

- That UNG/0038 exists. If it does not contain data, the name of the message is inserted.
- That UNG/S006 and UNG/S007 exist.
- That UNG/S004 exists. If it does not contain data, the current date/time in EDI format is inserted.
- That UNG/0051 exists. If it does not contain data, the first two characters of the controlling agency from the user-defined settings are inserted. See **Settings | Controlling agency**.
- That UNE/0060 exists. If it does not contain data, the number of messages in the group is calculated and inserted.
- That UNE/0048 exists. If it does not contain a value, the value from UNG/0048 is copied.
- That UNG/0048 exists. If it does not contain a value, the value from UNE/0048 is copied.

Message checking makes sure:

- That UNH/S009/0065 exists. If it does not contain data, the name of the message is inserted.
- That UNH/S009/0052 and UNH/S009/0054 exist.
- That UNH/S009/0051 exists. If it does not contain data, the first two characters of the controlling agency from the user-defined settings are inserted. See **Settings | Controlling agency**.
- That UNT/0074 exists. If it does not contain data, the number of segments in the message is calculated and inserted.
- That UNT/0062 exists. If it does not contain data, the value from UNH/0062 is copied.
- That UNH/0062 exists. If it does not contain data, the value from UNT/0062 is copied. (If only the trailer segment number is mapped, then the corresponding field in the header segment is supplied with the same value)

Settings
Clicking the Extended... button in the component settings dialog box opens the extended settings dialog box. Please see the UN/EDIFACT specification for more details.



- Controlling agency: Default **UNO**
- Syntax version number: Default **2**
- Syntax level: Default **A**
- Suppress UNA segment: Default **unchecked.**

11.3.2 ANSI X12 target - validation

The following items are checked when a ANSI X12 document is validated:

- Whether an ISA and an IEA segment exist
- Whether ISA/I01 contains a legal authorization information qualifier. See Legal values and qualifiers Legal authorization information qualifiers.
- Whether ISA/I03 contains a legal security information qualifier. See Legal values and qualifiers Legal security information qualifiers.
- Whether the two ISA/I05 segments contain legal interchange ID qualifiers. See Legal values and qualifiers Legal interchange ID qualifiers.
- Whether ISA/I08 contains a well-formed date value.
- Whether ISA/I09 contains a well-formed time value.
- Whether ISA/I13 contains a legal boolean value. See Legal values and qualifiers Legal boolean values.
- Whether ISA/I14 contains a legal interchange usage indicator. See Legal values and qualifiers Legal interchange usage indicators.
- Whether ISA/I12 and IEA/I12 contain the same value.
- Whether IEA/I16 contains the correct number of function groups in the interchange.

Each **function group** is checked:

- If there is a matching GS and GE pair.
- Whether GS/373 contains a well-formed date value.
- Whether GS/337 contains a well-formed time value.
- Whether GS/28 and GE/28 contain the same value.
- Whether GE/97 contains the correct number of messages in the function group.

Each **message** is checked:

- If there is a matching ST and SE pair.
- Whether ST/143 contains the correct message identifier.
- Whether ST/329 and SE/329 contain the same value.
- Whether SE/96 contains the correct number of segments in the message.

Automatic data completion for EDI/X12 makes sure:

- That an ISA and IEA pair exist on the interchange level.
- That if either GS or GE exist, the other ID also exists.
- That there is at least one ST/SE pair on the message level.
- That ISA/I01 and ISA/I03 exist. If they do not contain data, 00 is inserted.
- That ISA/I02 and ISA/I04 exist. If they do not contain data, ten blanks are inserted.
- That both ISA/I05 segments exist. If they do not contain data, ZZ is inserted.
- That ISA/I08 exists. If it does not contain data, the current date in EDI format is inserted.

- That ISA/I09 exists. If it does not contain data, the current time in EDI format is inserted.
- That ISA/I65 exists. If it does not contain data, the repetition separator is inserted.
- That ISA/I11 exists. If it does not contain data, the interchange control version number from the user-defined settings is inserted. See **Settings | Interchange control version-number**.
- That ISA/I12 exists.
- That ISA/I13 exists. If it does not contain data, the request acknowledgment setting is used. See **Settings | Request acknowledgement**.
- That ISA/I14 exists. If it does not contain data, P is inserted.
- That ISA/I15 exists. If it does not contain data, the composite separator from the user-defined settings is inserted. See **Settings | composite separator**.
- That IEA/I16 exists. If it does not contain data, the number of function groups in the interchange is calculated and inserted.
- That IEA/I12 exists. If it does not contain data, the value from ISA/I12 is copied.

Please note:

Any fields not mentioned in this section (Automatic data completion) are NOT inserted, or created. The correct values cannot be ascertained automatically.

Given a (target) parent element A (in the target EDI component) with child items x, y, and z - where y is mandatory, parent element A will only be created in the output file if the mandatory child element "y" in the target component has been mapped!

The potentially existing function group, is checked:

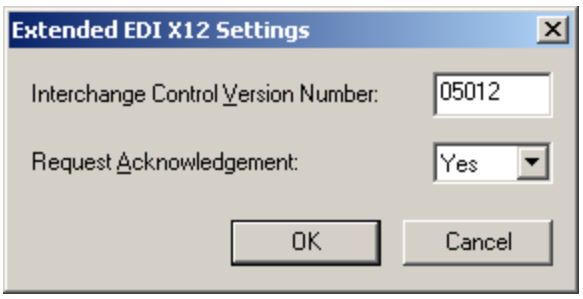
- That GS/373 exists. If it does not contain data, the current date in EDI format is inserted.
- That GS/337 exists. If it does not contain data, the current time in EDI format is inserted.
- That GE/97 exists. If it does not contain data, the number of messages in the function group are calculated and inserted.
- That GE/28 exists. If it does not contain data, the value from GS/28 is copied.

Message checking makes sure:

- That ST/143 exists. If it does not contain data, the name of the message is inserted.
- That SE/96 exists. If it does not contain data, the number of segments in the message is calculated and inserted.
- That ST/329 and SE/329 exist. If SE/329 does not contain data, the value from ST/329 is copied.

Settings

Clicking the Extended... button in the component settings dialog box opens the extended settings dialog box. Please see the EDI/X12 specification for more details:



- Interchange control version number: Default **05030**
- Request acknowledgement: Default **yes**

11.3.3 Legal values and qualifiers

Legal authorization information qualifiers:

00 01 02 03 04 05

Legal security information qualifiers:

00 01

Legal interchange ID qualifiers:

01 02 03 04 08 09 11 12 13 14 15 16

17 18 19 20 21 22 23 24 25 NR ZZ

Legal boolean values:

0 1

Legal interchange usage indicators:

P T I

11.4 Converting customized EDI configuration files

The format of the configuration files used to customize EDI messages has undergone a major change in MapForce **2006 R3**. Although several files are used in the customization process, only one configuration file needs to be converted, namely the *.Config or *.Config.xml file.

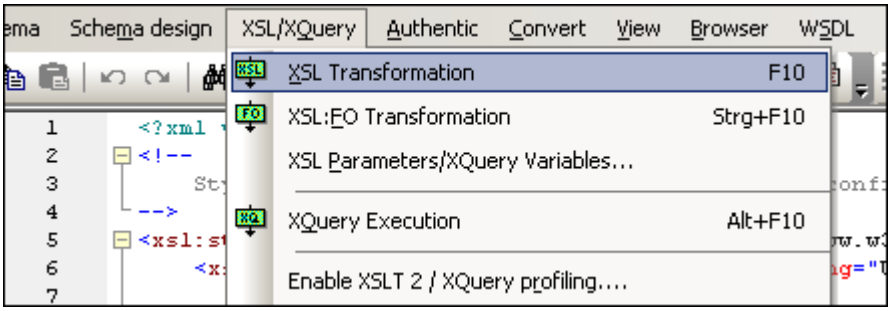
A supplied XSLT stylesheet converts the old customization files to the new customization format.

- The XSLT file, **Convert-EDI-config.xslt**, file is available in the ...**MapForceEDI** folder.
- Files that can be converted are ***.Config** or ***.Config.xml** files.
- Both UN/EDIFACT as well as ANSI X12 customization files can be transformed.

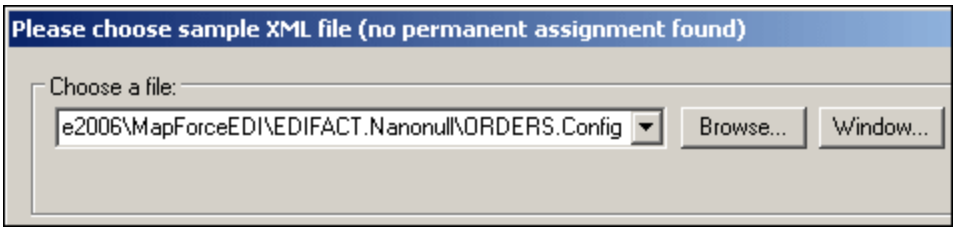
Converting the configuration files using Altova XMLSpy

To convert a pre-2006R3 EDI configuration file to the new format:

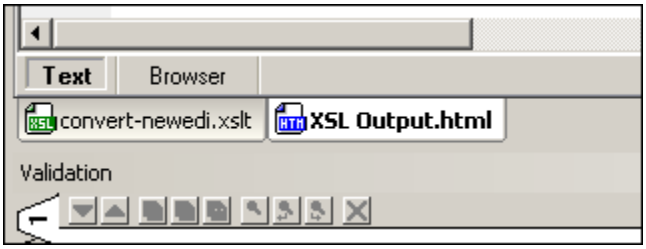
1. Open the supplied XSLT stylesheet, **Convert-EDI-config.xslt,** in XMLSpy.
2. Select **XSL/XQuery | XSL Transformation**.



3. Choose the configuration file that you want to convert (***.Config,** or ***.Config.xml**).



4. Click OK to start the transformation.
   XMLSpy opens a new window called "XSL Output.html", which contains the transformed configuration file. The html extension is the default setting; the converted file is XML however.



5. Click the "Text" tab to see the resulting configuration file.
6. Check the converted EDI configuration file (please see Checking transformed configuration files).
7. Save the transformed file as **<YourConfig>.Config** inside a subfolder of the ...\ **MapForceEDI** folder.

8. Add the new file name to the **EDI.Collection** file of the folder by adding the line <Message Type="<YourType>" File="<YourConfig>.Config" Description="<YourDescription>"/> inside the <Messages> tag.
9. Start MapForce. You can now use your customized message. Please see Customizing an EDIFACT message, or Customizing an ANSI X12 transaction for more information on customization specifics.

Converting the configuration files using a Command Line XSLT Processor
You can also convert the configuration file using a command line XSLT Processor, such as the AltovaXML Processor, which is available from the "Components and Free Tools Download Center" of the Altova.com website.

To convert a configuration file using the Altova XSLT Processor
Command line syntax for AltovaXML:

AltovaXML -xslt1 ***xsltfile*** **-in** ***xmlfile*** **[-out** ***outputfile*****] [options]**

To convert EDI configuration files this actually means:

AltovaXML -xslt1 ***xsltfile*** **-in** ***Old_Config_file*** **-out** ***New_Config_file***

1. Open a command prompt and enter, e.g.

   AltovaXML -**xslt1** Convert-EDI-config.xslt -**in** old850.Config -**out** new850.Config

   Please enclose the path parameters in double quotes (") if they contain any space characters.

Checking transformed configuration files:
The conversion process produces a version of the configuration file that adheres to the new configuration format.

Areas that may still need to be looked at:

- To parse **standalone** messages, i.e. messages not enclosed in group or interchanging wrapping, you may have to add minOccurs="0" on certain segments (UNB, UNZ for EDIFACT; ISA, GS for X12).
- ANSI X12 date and time fields were of type "string" in the previous format. To have MapForce treat these as date and time fields, you can change the type accordingly (type="date" or type="time").
- The new configuration files distinguish between **repeated elements** and sequences consisting of several **instances** of a **single** element. Repeated elements are separated by the Repetition Separator, while several instances of a single element are separated by the normal Data Element Separator (visible in the Component Settings dialog box).

  When converting old configuration files, the XSLT stylesheet assumes that groups and segments that have **Repeat** greater than one are always repeated, while in the case of composite and data elements, Repeat is an abbreviation for a sequence of individual elements. Therefore **maxOccurs** is output/generated for groups and segments, while **mergedEntries** is output/generated for composite and data elements.
- The new configuration files support ANSI X12 **implicit decimals**. To be able to use implicit decimals, add implicitDecimals="<number of implicit decimal digits>" to the appropriate data element.

  If you specify implicitDecimals="2" then a value of 1295 in the source EDI message, is treated as 12.95 in MapForce.
- Old configuration files did not contain an entry for the EDIFACT UNA special segment. The XSLT stylesheet therefore inserts the UNA segment entry inside the Interchange, if

the Interchange originally started with UNB.

- The XSLT file detects whether an ANSI X12 or UN/EDIFACT message is being transformed, by checking for the existence of the ISA segment, and sets the <Meta> tag accordingly.

11.5 Customizing an EDIFACT message

MapForce allows you to customize EDIFACT messages to take different nonstandard, or changed EDIFACT formats into account.

This example uses the **Orders-Custom-EDI.mfd** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. Please note that a ZIP file, **EDIFACT.Nanonull.zip**, is also included in the ...\Tutorial folder. The ZIP file contains the files final **result** of the customization procedure.

The EDIFACT file available in the ...\Tutorial folder, **Orders-Custom.EDI**, has been changed to include a new component element in the CTA segment:

- Line 9 contains a **Mr**. entry, and
- Line 11 contains a **Mrs** entry.

```
 1    UNB+UNOB:1+003897733:01:MFGB+PARTNER ID:ZZ:ROUTING
      ADDR+970101:1230+00000000000001++ORDERS++++1'
 2    UNH+0001+ORDERS:S:93A:UN'
 3    BGM+221+ABC123456XYZ+9'
 4    DTM+4:200404301742PDT:303'
 5    FTX+PUR+3++Pizza purchase order'
 6    RFF+CT:123-456'
 7    RFF+CR:1122'
 8    NAD+SE+999::92++24h Pizza+Long Way+San-Francisco+CA+34424+US'
 9    CTA+SR+:Ted Little:Mr'
10    NAD+BY+123::92++Nanonull, Inc.+119 Oakstreet Suite 4876+Vereno+CA+29213+US'
11    CTA+PD+:Michelle Butler:Mrs'
12    NAD+ST+123::92++Nanonull, Inc.+119 Oakstreet Suite 4876+Vereno+CA+29213+US'
13    TAX+9++++++1-12345-6789-0'
14    CUX+2:USD:9'
15    PCD+12:2'
16    TDT+20++++:::24h Pizza Fast Carrier'
17    LOC+16+Nanonull Main Entrance'
```

The CTA (Contact Information) segment of the ORDERS message must be extended for the new data to be able to be mapped.

Please note:
Customization information for EDIFACT and ANSI X12, is supplied by two structurally equivalent files that allow the use of multiple consecutive elements of the same name in each of the EDI formats. EDIFACT uses the **EDSD.segment** file while ANSI X12 uses the **X12.Segment** file. Please see "Multiple consecutive elements" for more information.

The format of the configuration files used to customize EDI messages has undergone a major change in MapForce 2006 R3, please see Converting customized EDI configuration files or more information.

11.5.1 EDIFACT: customization set up

The text in the following sections describes how to customize the configuration files to be able to map the changed EDIFACT message to the **ORDER-EDI.xsd** schema. It assumes that the supplied ZIP file is **not** used.

Setting up the customizing example:

- Create an EDIFACT.Nanonull folder.
- copy the following files from the ...**MapForceEDI\EDIFACT** folder into the EDIFACT.Nanonull folder:

  Admin.Segment
  EDI.Collection *
  EDSD.Segment *
  Orders.config *
  UNCL.Codelist

- Change the attributes of the files marked with an asterisk "*" to **read-write** to make them editable.

Configuring the EDI.Collection file:

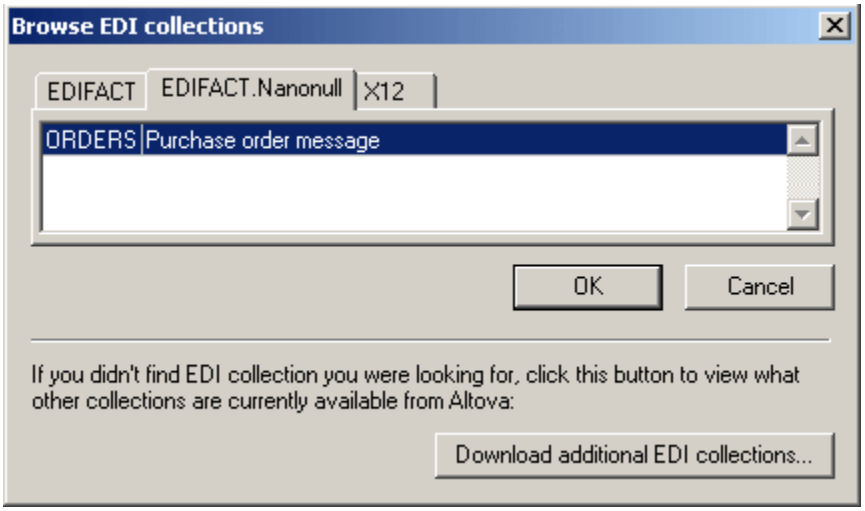
1. Open "EDI.Collection" file in XMLSpy, or in you preferred editor.
2. Remove all "Message" elements, except for the "ORDERS" message. Make sure you retain the <Messages> tags however!

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="2">
    <Meta>
        <Version>D</Version>
        <Release>04B</Release>
        <Agency>UN</Agency>
    </Meta>
    <Message Type="ORDERS" File="ORDERS.Config" Description="Purchase order message"/>
</Messages>
```

3. Save the file.

To see the content of the collection file:

1. Start MapForce, select **Insert | EDI**, or click the Insert EDI icon.
   The Browse EDI collections dialog box opens displaying a new tab named "EDIFACT.Nanonull". Only one entry is visible; the first column shows the message type "ORDERS", and the second the message description text.



Please note:

MapForce searches through all sibling subfolders under the "...**MapforceEDI**" directory and scans for a file called "EDI.Collection". The folder name containing an EDI Collection, appears as a tab in the dialog box. The listbox shows the current content of the collection file, which in our example only contains an ORDERS message entry.

The goal of this section is to redefine the CTA (Contact Information) segment by adding the X1000 field to make it available to individual messages. CTA consists of one field (F3139) and one composite (C056).

There are several ways the customization can be achieved:

- **Globally** by customizing the **EDSD.segment** file. All segments, in all messages that use composite C056, will contain/reference the new element.
- **Locally** by customizing the **ORDERS.config** file. All segments in the current message that use composite C056, will contain the new element.
- **Inline** by customizing the **ORDERS.config** file. Only the customized segment (CTA) in the current message will contain the new element.

UNCL.Codelist
This file defines the various EDIFACT codes and the values that they may contain, and is used for validation of output files in MapForce. If you organization uses a special code not in the EDIFACT code list, add it here.

EDI.Collection
The Collection file contains a list of all messages in the current directory. It is used to provide a list of the available messages when inserting an EDIFACT file into MapForce. The following example contains only one message, namely "ORDERS".

Edit this file to contain only those messages relevant to your work.

Orders.config
The configuration file for Purchase order message files. This file contains all groups and segment definitions used in Orders messages. Changes made to this file can define local or inline customizations.

Admin.Segment
"Admin.segment" describes the Interchange level administrative segments. They are all used to parse the EDIFACT file.

EDSD.Segment (Electronic Data Segment Definition)
This file defines the Segment, Composite and Field names of the EDIFACT files, and is used when parsing the EDIFACT file. Changes made to this file are global customizations, and apply to all segments and messages.

11.5.2 Global customization

- Changes only have to be made to the **EDSD.Segment** file to be able to access the new X1000 field globally.
- All **segments**, in **all messages** that use composite C056, will contain/reference the new element.

Composite redefinition in EDSD.Segment file:
Open the EDSD.Segment file in XMLSpy, or in you preferred editor, and navigate to **Config | Elements | Composite | C056**.

```
<Composite name="C056" info="DEPARTMENT OR EMPLOYEE DETAILS">
    <Data ref="F3413" minOccurs="0"/>
    <Data ref="F3412" minOccurs="0"/>
</Composite>
```

Insert the following line in the C056 segment, under F3412:

```
<Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
```

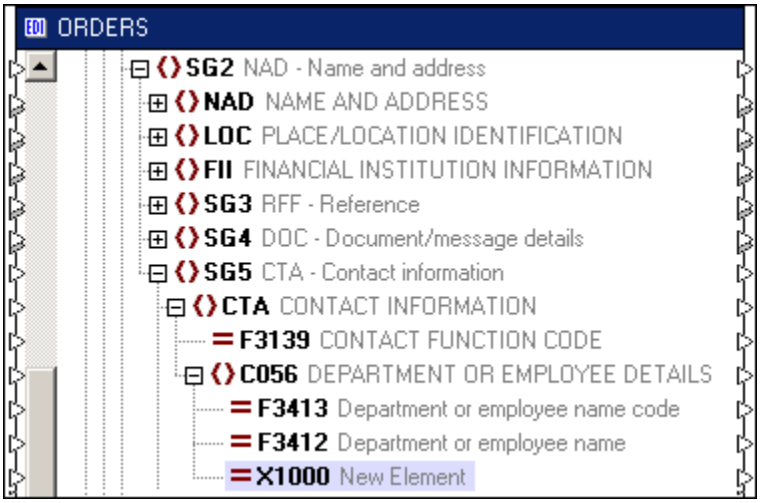
The composite definition appears as shown below:

```
<Composite name="C056" info="DEPARTMENT OR EMPLOYEE DETAILS">
    <Data ref="F3413" minOccurs="0"/>
    <Data ref="F3412" minOccurs="0"/>
    <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
</Composite>
```

Please note:
The new X1000 field is defined using the "name" attribute as opposed to other fields of the segment which use the "ref" attribute. The two other fields are defined at the beginning of the EDSD-Segment file, outside of the Composite section, (using the Data name element) and are only referenced here. The new field can now be referenced from different Segments or Composites.

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "EDIFACT.Nanonull" tab, and select the ORDERS message.
3. Click the Cancel button to skip the selection of the source EDI file for the moment. The ORDERS component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message/SG2/SG5/CTA/056**, to see the new X1000 element.



11.5.3 Local customization

- Changes only have to be made to the **ORDERS.config** file, at the specified location, to be able to access the new X1000 field locally.
- All segments in the **[Orders]** message that use composite C056, will contain/reference the new element.

Local customization - composite redefinition in ORDERS.Config file:
Open the ORDERS.Config file in XMLSpy, or in you preferred editor, and navigate to **Config | Message** element.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
3       <Meta>
4           <Version>D</Version>
5           <Release>04B</Release>
6           <Agency>UN</Agency>
7       </Meta>
8       <Include href="Admin.Segment"/>
9       <Include href="EDSD.Segment"/>
10      <Include href="UNCL.Codelist"/>
11      <Message>
12          <MessageType>ORDERS</MessageType>
13          <Description>Purchase order message</Description>
14          <Revision>14</Revision>
15          <Date>2004-11-23</Date>
16          <Group name="Envelope">
```

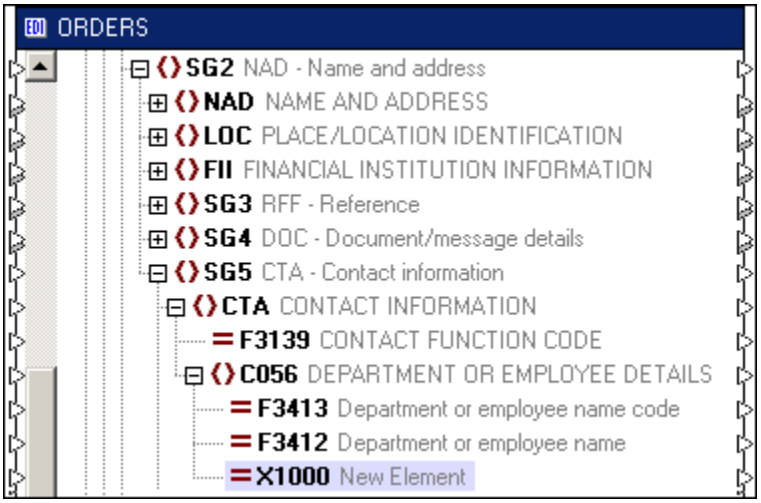
Insert the following lines before the first "**Include href**..." element insert the following lines (this can be copied from the EDSD.Segment file):

```
<Elements>
  <Composite name="C056" info="DEPARTMENT OR EMPLOYEE DETAILS">
    <Data ref="F3413" minOccurs="0"/>
    <Data ref="F3412" minOccurs="0"/>
    <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
  </Composite>
</Elements>
```

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Ver
3       <Meta>
4           <Version>D</Version>
5           <Release>04B</Release>
6           <Agency>UN</Agency>
7       </Meta>
8   <Elements>
9       <Composite name="C056" info="DEPARTMENT OR EMPLOYEE DETAILS">
10          <Data ref="F3413" minOccurs="0"/>
11          <Data ref="F3412" minOccurs="0"/>
12          <Data name="X1000" type="string" maxLength="35" minOccurs
13      </Composite>
14  </Elements>
15      <Include href="Admin.Segment"/>
16      <Include href="EDSD.Segment"/>
17      <Include href="UNCL.Codelist"/>
18      <Message>
19          <MessageType>ORDERS</MessageType>
```

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "EDIFACT.Nanonull" tab, and select the ORDERS message.
3. Click the Cancel button to skip the selection of the source EDI file for the moment. The ORDERS component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message/SG2/SG5/CTA/056**, to see the new X1000 element.



Please note:
The Elements | Composite... C056 section, must be inserted before the **Include** element due to the definition of the EDIConfig.xsd file, (in the MapForceEDI folder) which allows you to validate the result of EDI mappings in the Output window.

11.5.4 Inline customization

- Changes only have to be made to the **ORDERS.config** file, at the specified location, to be able to have inline access to the new X1000 field.
- Only the redefined CTA segment in the **current** message, will contain/reference the new element.
- In other words, the CTA segment is redefined locally to contain a redefined Composite C056, with the local definition of the new field X1000.

Open the ORDERS.Config in XMLSpy, or in you preferred editor, and navigate to **Message | Envelope | Interchange | Group | Message | SG2 | SG5 | CTA** (or search for **SG5**).

```
47    <Group name="SG5" minOccurs="0" maxOccurs="5" info="CTA - Contact information">
48        <Segment ref="CTA"/>
49        <Segment ref="COM" minOccurs="0" maxOccurs="5"/>
50    </Group>
```

Replace the line :

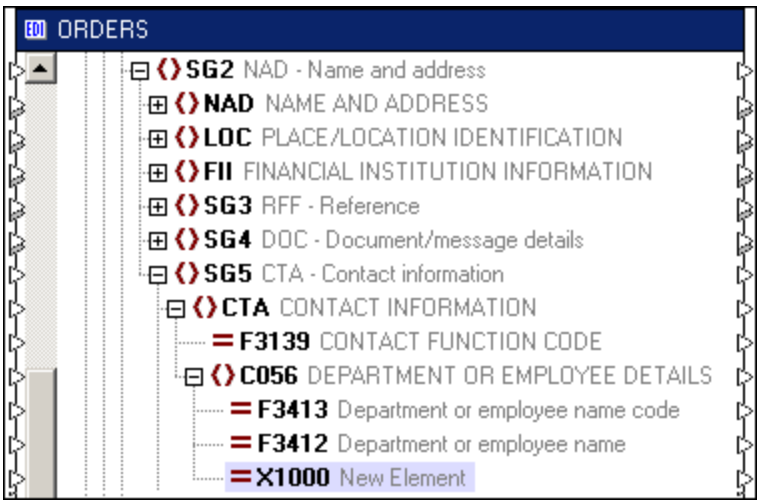
with the following lines:

```
<Segment name="CTA" info="CONTACT INFORMATION">
  <Data ref="F3139" minOccurs="0"/>
    <Composite name="C056" minOccurs="0" info="DEPARTMENT OR EMPLOYEE DETAILS">
    <Data ref="F3413" minOccurs="0"/>
    <Data ref="F3412" minOccurs="0"/>
    <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
  </Composite>
</Segment>
```

```
47    <Group name="SG5" minOccurs="0" maxOccurs="5" info="CTA - Contact information">
48        <Segment name="CTA" info="CONTACT INFORMATION">
49        <Data ref="F3139" minOccurs="0"/>
50            <Composite name="C056" minOccurs="0" info="DEPARTMENT OR EMPLOYEE DETAILS">
51            <Data ref="F3413" minOccurs="0"/>
52            <Data ref="F3412" minOccurs="0"/>
53            <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Ele
54        </Composite>
55    </Segment>
56    <Segment ref="COM" minOccurs="0" maxOccurs="5"/>
57    </Group>
```

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "EDIFACT.Nanonull" tab, and select the ORDERS message.
3. Click the Cancel button to skip the selection of the source EDI file for the moment. The ORDERS component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message/SG2/SG5/CTA/C056**, to see the new X1000 element.



11.5.5 Customized Orders mapping example

The mapping visible in the images below, Orders-Custom-EDI.mfd, is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** directory.

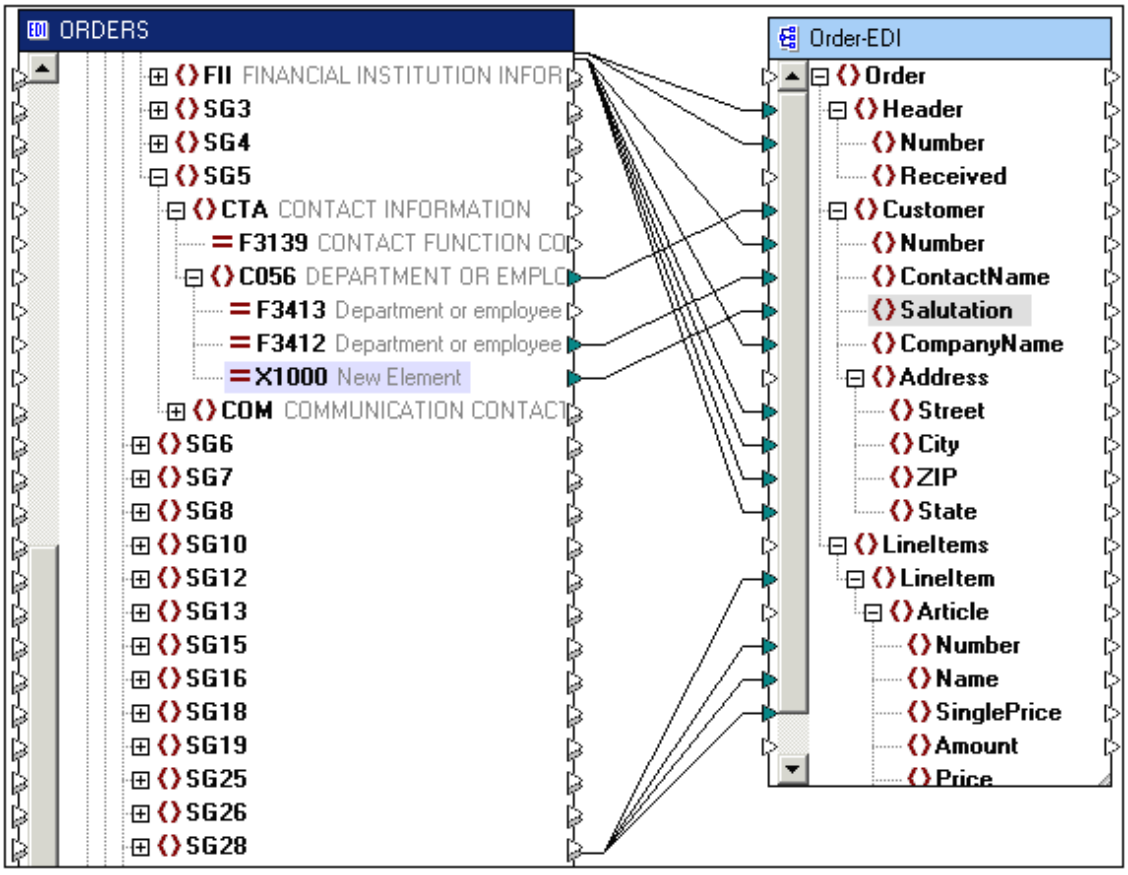
The example maps the ORDERS-Custom.EDI file to the Order-EDI schema. The field that has been added to the EDI structure, X1000, has been mapped to the Salutation item.

To see the customization result:

1. Create a new folder under the "**...\MapforceEDI**" directory and name it e.g. "**EDIFACT.Nanonull**"
2. Unzip the supplied EDIFACT.Nanonull.zip file (from the ...\Tutorial folder) into the new folder. The ZIP file contains the follow files:

   Admin.Segment
   EDI.Collection
   EDSD.Segment
   Orders.config
   UNCL.Codelist

3. Open the **Orders-Custom-EDI.mfd file** and click the Output tab to see the result of the mapping.



Clicking the Output tab displays the mapping result shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
  <Header>
    <Number>ABC123456XYZ</Number>
  </Header>
  <Customer>
    <Number>92</Number>
    <ContactName>Ted Little</ContactName>
    <Salutation>Mr</Salutation>
    <CompanyName>24h Pizza</CompanyName>
    <Address>
      <Street>Long Way</Street>
      <City>San-Francisco</City>
      <ZIP>34424</ZIP>
      <State>CA</State>
    </Address>
  </Customer>
  <Customer>
    <Number>92</Number>
    <ContactName>Michelle Butler</ContactName>
    <Salutation>Mrs</Salutation>
    <CompanyName>Nanonull, Inc.</CompanyName>
```

Code generation note:
When generating C++ code, a class named "CX1000Type" is generated which is accessible from the "CC056Type" class.

11.6 Customizing an ANSI X12 transaction

MapForce allows you to customize X12 transactions to take different nonstandard, or changed X12 formats into account.

This example uses the **Orders-Custom-X12.mfd** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. Please note that a ZIP file, **X12.Nanonull.zip**, is also included in the ...\Tutorial folder. The ZIP file contains the files final result of the customization procedure.

The X12 source file available in the ...\Tutorial folder, **Orders-Custom.X12**, has been changed to include a new field in the N2 segment:

- Line 6 contains an additional ++**Mrs** entry

```
 1    ISA+00+          +00+          +ZZ+SenderID       +ZZ+ReceiverID
 2    GS+PO+SenderID+ReceiverID+20060308+182347+1+UN+050112'
 3    ST+850+12345'
 4    BEG+00+SA+ABC123456XYZ++20040430'
 5    N1+1 +Nanonull, Inc.++123'
 6    N2+Michelle Butler++Mrs'
 7    N3+119 Oakstreet Suite 4876'
 8    N4+Vereno+CA+29213'
 9    PO1++1++7.2'
10    PO3+1 +++7.2++1+US'
11    PID+1++++Pizza Pepperoni'
12    TXI+1 ++9'
13    AMT+1+720'
14    PO1++2++13.2'
15    PO3+1 +++6.6++2+US'
```

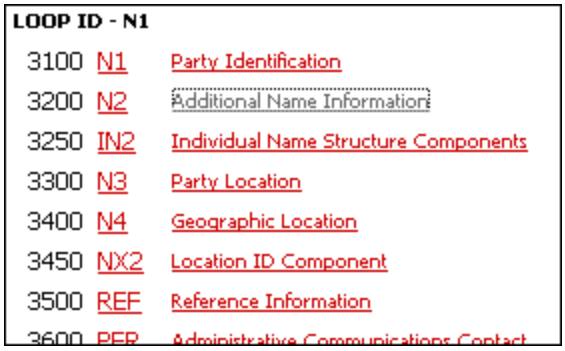
Please note:
Customization information for EDIFACT and ANSI X12, is supplied by two structurally equivalent files that allow the use of multiple consecutive elements of the same name in each of the EDI formats. EDIFACT uses the **EDSD.segment** file while ANSI X12 uses the **X12.Segment** file. Please see "Multiple consecutive elements" for more information.

The format of the configuration files used to customize EDI messages has undergone a major change in MapForce 2006 R3, please see Converting customized EDI configuration files for more information.

11.6.1 Customizing X12 source files

When customizing an X12 transaction it is important to take note that segments often allow the use of multiple **consecutive elements** of the same **name**.

Multiple consecutive elements of the same name
The goal of this section is to be able to map an ANSI X12 file that has been customized to an XML schema file. A new field has been added to the N2 segment i.e. "Mrs". This additional data field should be mapped to the Salutation field in the XML schema.



Adding a new field to an X12 file would normally entail adding a single separator character, generally the + character, followed by the data. The N2 segment specification, shown below, allows for two consecutive fields specified as 'Name'.

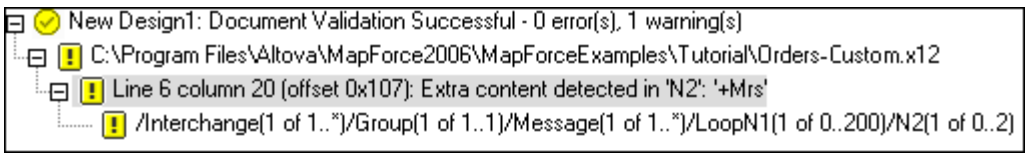
REF	ELE ID	NAME	RPT	ATTRIBUTES
01	93	Name	M	AN 1/60
02	93	Name	O	AN 1/60

The X12 source file therefore has to take this into account, by adding an "empty" field separator for the first (mandatory) occurrence, and a second one to separate the actual data. This means that **++Mrs** has to be entered for the data to adhere to the X12 specification.

You can find out the specific fields that have multiple entries by looking at the **X12.Segment file** supplied with MapForce, in the ...**MapForceEDI\X12** folder. Any segment that may contain multiple identical field entries (e.g. 93), is shown as the field number with a **mergedEntries** attribute which defines how many multiples may occur, in this case 2.

```
<Segment name="N2" info="Additional Name Information">
    <Data ref="F93" mergedEntries="2"/>
</Segment>
```

Inserting the X12 file with only one + separator, causes the following warning to appear, in the Messages window, when the EDI component is inserted and the sample EDI file has been assigned:



The Messages window supplies very specific help on the location and cause of a message. Clicking the respective message displays the specific line in the component if a connector is missing, or shows that extra content exists for one of the items.

11.6.2 X12 customization set up

The text in the following sections describes how to customize the configuration files to be able to map the changed X12 transaction to the **ORDER-x12.xsd** schema. It assumes that the supplied ZIP file is **not** used.

Setting up the customizing example:

- copy the following files from the ...**MapForceEDI\X12** folder into the X12.Nanonull folder:

  EDI.Collection
  850.config
  X12.Segment
- Change the attributes of these files to **read-write,** to make them editable.

Configuring the EDI.Collection file:

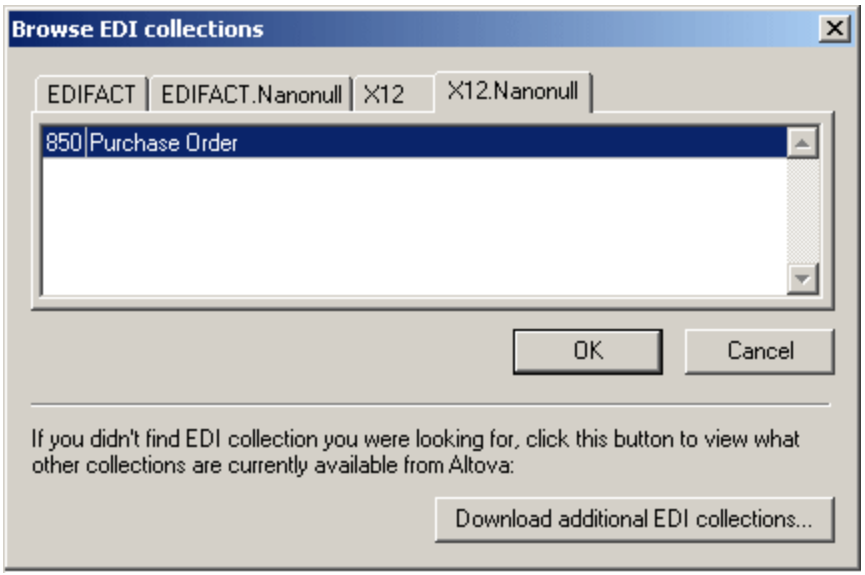
1. Open "EDI.Collection" file in XMLSpy, or in you preferred editor.
2. Remove all "Message" elements, except for the "850 Purchase Orders". Make sure you retain the <Messages> tags however!

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages Version="2">
    <Meta>
        <Release>5012</Release>
        <Agency>X12</Agency>
    </Meta>
    <Message Type="850" File="850.Config" Description="Purchase Order"/>
    </Messages>
```

3. Save the file.

To see the content of the collection file:

1. Start MapForce, select **Insert | EDI**, or click the Insert EDI icon.
   The Browse EDI collections dialog box opens displaying a new tab named "X12.Nanonull". Only one entry is visible; the first column shows the message type "850", and the second the message description text "Purchase Order".



Please note:

MapForce searches through all sibling subfolders under the "...**MapforceEDI**" directory and scans for a file called "EDI.Collection". The folder name containing an EDI Collection, appears as a tab in the dialog box. The listbox shows the current content of the collection file, which in our example only contains a Purchase Order entry.

The goal of this section is to redefine the N2 "Additional Name Information" segment by adding the X1000 field to make it available to individual transactions. N2 currently consists of one field, "F93 Name".

There are several ways the customization can be achieved:

- **Globally** by customizing the **X12.segment** file. All segments, in all transactions that use N2, will contain/reference the new element.
- **Locally** by customizing the **850.config** file. All segments in the current transaction that use N2, will contain the new element.
- **Inline** by customizing the **850.config** file. Only the customized segment (N2) in the current transaction will contain the new element.

EDI.Collection
The Collection file contains a list of all transactions in the current directory. It is used to provide a list of the available transactions when inserting an X12 file into MapForce. The following example contains only one transaction, namely "Purchase Order".

850.config
The configuration file for Purchase order transaction files. This file contains all groups and segment definitions used in purchase order transactions. Changes made to this file can define local or inline customizations.

X12.Segment
This file defines the Segment, Composite and Field names of the X12 files, and is used when parsing the file. Changes made to this file are global customizations, and apply to all segments and transactions.

11.6.3 Global customization

- Changes only have to be made to the **X12.Segment** file to be able to access the new X1000 field globally.
- All **segments**, in **all transaction** that use **N2**, will contain/reference the new element.

Redefinition in X12.Segment file:
Open the X12.Segment file and navigate to **Config | Elements | Segment name="N2" info="Additional Name Information"**.

```
<Segment name="N2" info="Additional Name Information">
    <Data ref="F93" mergedEntries="2"/>
</Segment>
```

Insert the following line under F93:

<Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>

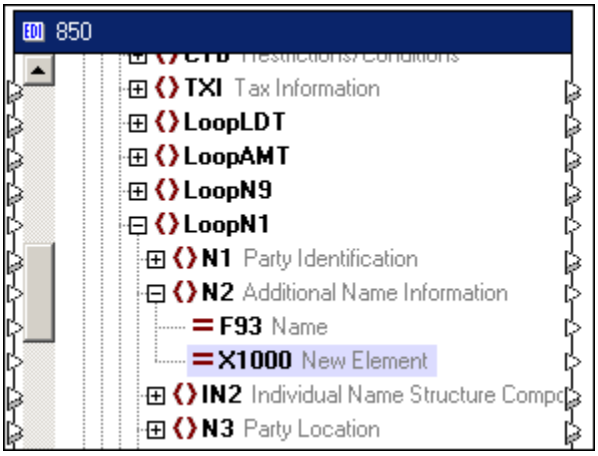
The definition appears as shown below:

```
<Segment name="N2" info="Additional Name Information">
    <Data ref="F93" mergedEntries="2"/>
    <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
</Segment>
```

Please note:
The new X1000 field is defined using the "name" attribute as opposed to other fields of the segment which use the "ref" attribute. The F93 field is defined at the beginning of the X12.Segment file using the Data name element and is only referenced here. The new field can now be referenced from different Segments or Composites.

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "X12.Nanonull" tab, and select the 850 Purchase Order transaction.
3. Click the Cancel button to skip the selection of the source EDI file for the moment. The 850 component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message/LoopN1/N2**, to see the new X1000 element.



11.6.4 Local customization

- Changes only have to be made to the **850.config** file, at the specified location, to be able to access the new X1000 field locally.
- All **segments** in the **current** transaction that use Segment N2, will contain/reference the new element.

Local customization - segment redefinition in 850.Config file:
Open the 850.Config file and navigate to **Config | Include**.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <Config Version="2">
3       <Meta>
4           <Release>5012</Release>
5           <Agency>X12</Agency>
6       </Meta>
7       <Include href="X12.Segment"/>
8       <Message>
9           <MessageType>850</MessageType>
```

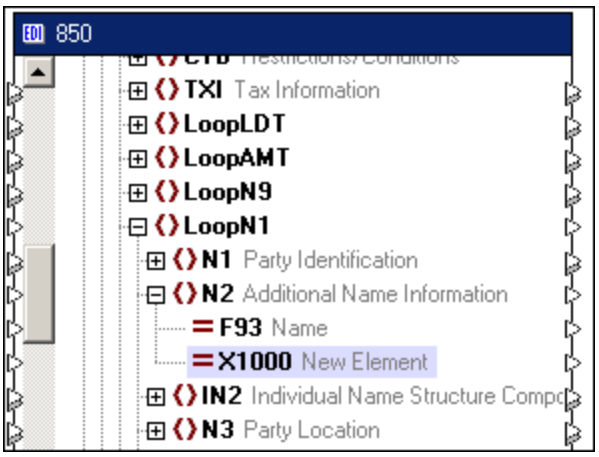
Insert the following lines before the "**Include href...**" element:

```
<Elements>
        <Segment name="N2" info="Additional Name Information">
        <Data ref="F93" mergedEntries="2"/>
        <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
        </Segment>
</Elements>
```

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <Config Version="2">
3       <Meta>
4           <Release>5012</Release>
5           <Agency>X12</Agency>
6       </Meta>
7   <Elements>
8       <Segment name="N2" info="Additional Name Information">
9       <Data ref="F93" mergedEntries="2"/>
10      <Data name="X1000" type="string" maxLength="35" minOcc
11      </Segment>
12  </Elements>
13      <Include href="X12.Segment"/>
14      <Message>
15          <MessageType>850</MessageType>
16          <Description>Purchase Order</Description>
```

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "X12.Nanonull" tab, and select the 850 Purchase Order transaction.
3. Click the Cancel button to skip the selection of the source EDI file for the moment. The 850 component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message/LoopN1/N2**, to see the new X1000 element.



Please note:
The Elements | Segments... section, should be inserted before the **Include** element due to the definition of the EDIConfig.xsd file, (in the MapForceEDI folder) which allows you to validate the result of EDI mappings in the Output window.

11.6.5 Inline customization

- Changes only have to be made to the **850.config** file, at the specified location, to be able to access the new X1000 field locally.
- Only the redefined segment N2 in the **current** transaction, will contain/reference the new X1000 field.
- In other words, the segment is redefined locally to contain the new field X1000.

Local customization - segment redefinition in 850.Config file:
Open the 850.Config file and navigate to **Config | Group | Message** | **Group name="LoopN1"** (or search for **LoopN1**).

```
73  <Group name="LoopN1" maxOccurs="200" minOccurs="0">
74      <Segment ref="N1"/>
75      <Segment ref="N2" minOccurs="0" maxOccurs="2"/>
76      <Segment ref="IN2" minOccurs="0" maxOccurs="unbounded"/>
77      <Segment ref="N3" minOccurs="0" maxOccurs="2"/>
78      <Segment ref="N4" minOccurs="0" maxOccurs="unbounded"/>
```

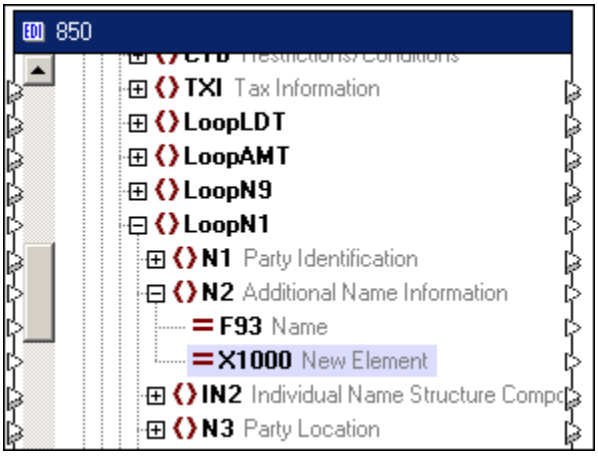
Replace the Segment ref="N2"... line with the following lines:

```
<Segment name="N2" info="Additional Name Information">
        <Data ref="F93" mergedEntries="2"/>
        <Data name="X1000" type="string" maxLength="35" minOccurs="0" info="New Element"/>
</Segment>
```

```
73  <Group name="LoopN1" maxOccurs="200" minOccurs="0">
74      <Segment ref="N1"/>
75      <Segment name="N2" info="Additional Name Information">
76          <Data ref="F93" mergedEntries="2"/>
77          <Data name="X1000" type="string" maxLength="35" minOcc
78      </Segment>
79      <Segment ref="IN2" minOccurs="0" maxOccurs="unbounded"/>
```

Previewing the new field in MapForce:

1. Select the menu option **Insert | EDI**, or click the Insert EDI  icon.
2. Click the "X12.Nanonull" tab, and select the 850 Purchase Order transaction.
3. Click the Cancel button to skip the selection of the source EDI file for the moment. The 850 component is now visible in the mapping window.
4. Navigate to **Envelope/Interchange/Group/Message/LoopN1/N2**, to see the new X1000 element.

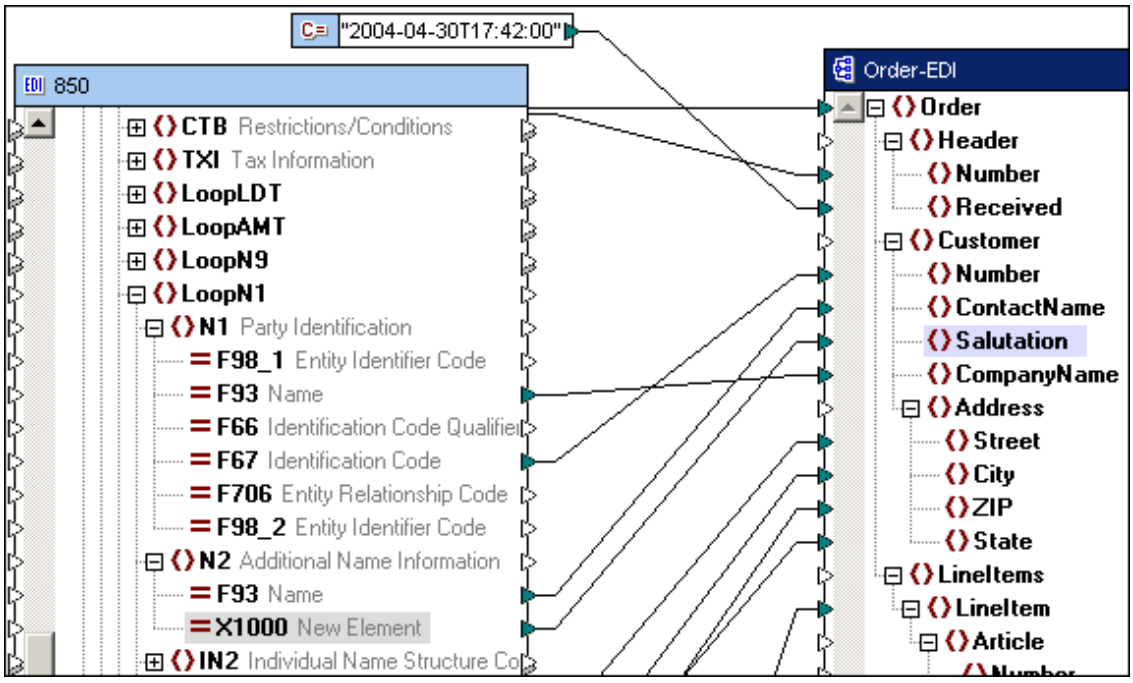


11.6.6 Customized X12 mapping example

The mapping visible in the images below, **Orders-Custom-X12.mfd**, is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** directory.

The example maps the Orders-Custom.x12 file to the Order-X12 schema. The field that has been added to the EDI structure, X1000, has been mapped to the Salutation item.

1. Create a new folder under the "**...\MapforceEDI**" directory and name it e.g. "**X12.Nanonull**"
2. Unzip the supplied X12.Nanonull.zip file (from the ...\Tutorial folder) into the new folder.
3. Open the **Orders-Custom-X12.mfd file.**



Clicking the Output tab displays the mapping result shown below.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
3    <Header>
4      <Number>ABC123456XYZ</Number>
5      <Received>2004-04-30T17:42:00</Received>
6    </Header>
7    <Customer>
8      <Number>123</Number>
9      <ContactName>Michelle Butler</ContactName>
10     <Salutation>Mrs</Salutation>
11     <CompanyName>Nanonull, Inc.</CompanyName>
12     <Address>
13       <Street>119 Oakstreet Suite 4876</Street>
14       <City>Vereno</City>
15       <ZIP>29213</ZIP>
16       <State>CA</State>
17     </Address>
```

Chapter 12

Mapping MS OOXML Excel 2007 files

12 Mapping MS OOXML Excel 2007 files

MapForce includes support for the mapping of Microsoft Excel 2007 (Office Open XML) files (*.xlsx), as both source and target components.

Microsoft Excel 2007 only needs to be installed if you intend to **preview** Excel 2007 sheets in the **Output** tab. If you use a previous version of Microsoft Excel, or you don't have Excel installed at all, you will still be able to map to/from Excel 2007 files. You cannot preview the result in the Output tab, but you can still save it, by clicking the Save output icon.

When generating code for C#, note that Visual Studio 2005 or 2008 must be selected in the **Tools | Options | Generation | C# settings** group, because only the .NET framework 3.0 or higher supports the package file format of OOXML. Code generation for Excel 2007 supports Java and C# for reading and writing, and XSLT 2 for reading only.

There are two ways that mapped data can be generated/saved:

- By clicking the Output tab which generates a preview in MS Excel 2007 using the built-in MapForce 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#** then compiling and executing the generated code (Excel files are not supported in C++ code generation).

Excel 2007 files files can be inserted as:

- An XLSX template along with a sample XLSX file which supplies the data to preview your mapping (as a source component).
- An empty XLSX template without a sample XLSX file (as a target component).

The following sections describe:

- How to define the mappable items of an Excel file
- Mapping Excel 2007 files to XML
- Mapping Database data to a Excel 2007 file

The mapping file used in the following examples is available as **Excel-mapping.mfd** in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. It contains three separate example mappings that will be described in more detail in each section.

12.1 Defining the mappable items of an Excel Workbook

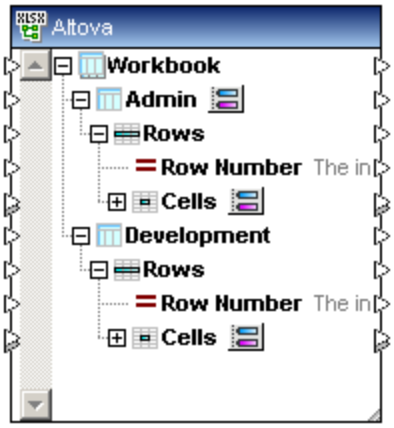
MapForce can read and write tabular data in Excel workbooks. The basic hierarchical structure of an Excel workbook is:

Workbook
    Worksheets
        Rows
            Cells (intersection of rows and columns)

Excel workbooks have no schemas like XML files, so their structure is defined directly in the MapForce component. It is possible to define the worksheet names and column names.

To insert an Excel *.xlsx file:

1. Click the "Insert Excel 2007 file" icon  in the toolbar.
   You are now prompted for a sample XLSX file to provide the data for the preview tab. The information in this file can be used to customize the tree structure of the MapForce component.
2. Click "**Browse...**" and select the file you want to use as a source component e.g. Altova.xlsx.



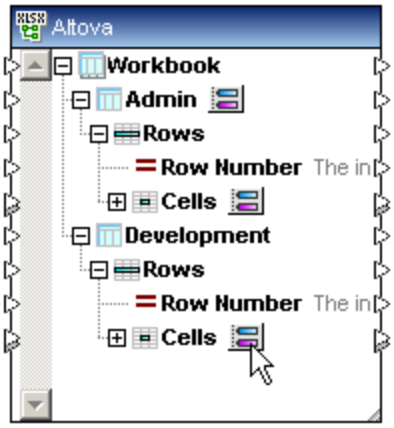
   The source component appears in the mapping tab. It contains several icons  that allow you to define the structure of your workbook, identifying the mappable data. Clicking one of these icons opens a dialog box allowing you to do so. The Worksheets are automatically imported from the XLSX file and displayed by default, you can however change this.

Defining the "mappable" Cells:

By default, all cells of the individual rows in a worksheet are represented by a single "Cells" item in MapForce. This means that all cells in each row will be processed sequentially. This mode is most useful if all columns in a worksheet have the same meaning, i.e. each row is a list of values.

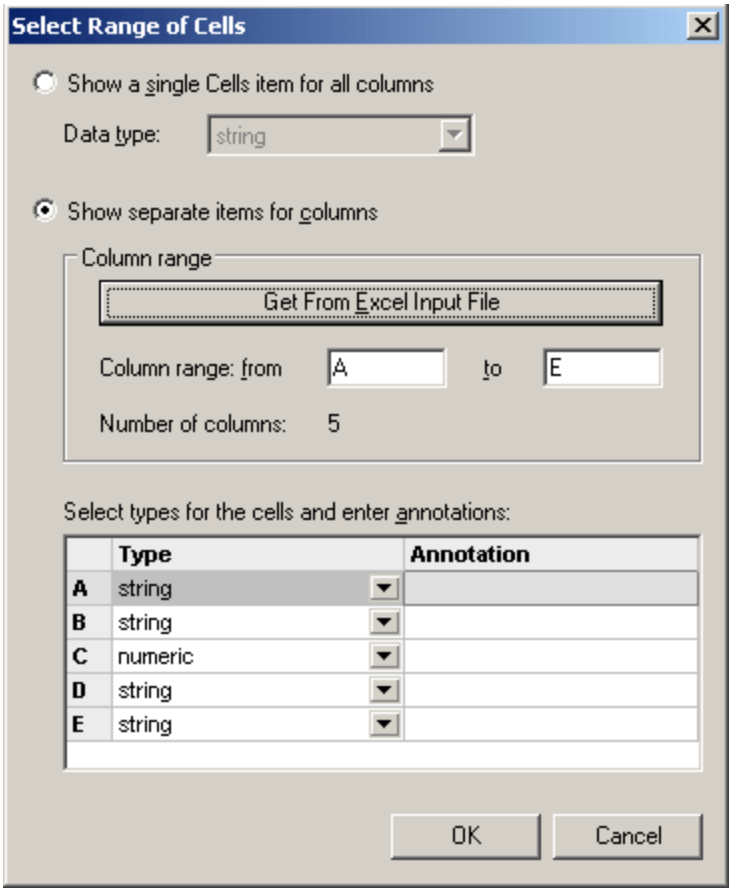
If you have a typical table structure with different columns, you can instead display separate items for each column in the MapForce tree, making it easy to access specific cells:

1. Click the **Cells**  icon in the Excel component for the specific worksheet whose columns you want to define.



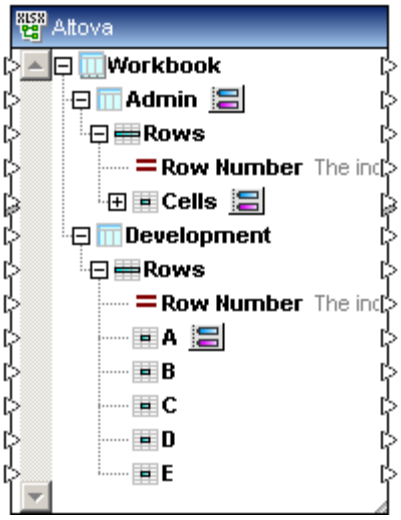
This opens the dialog box shown below.

2. Click the "Show separate items for columns" radio button, and click the "Get from Excel Input File" button. MapForce displays the cell range A to E.



You can change any of the type fields using the Type combo box, and add annotations to help you associate the column names to their contents.

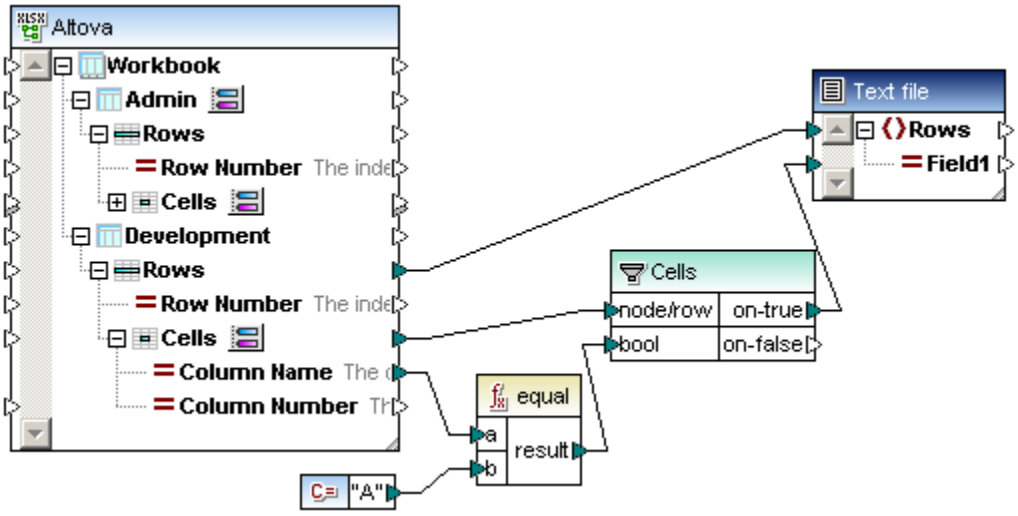
3. Click OK to confirm the selection. MapForce now displays separate items for each column:



To redefine the mappable cells (re-open the dialog box):

- Click the "Cells" icon next to the **A** item.

Dialog box options:
Show a single Cells item for all Columns: Collapses all cell items into a single mappable Cells item as shown below. The example below uses a filter to output all the cells for Column name "A".



Column Range
Get from Excel input file: Click this button to transfer the maximum cell range and cell types from Excel to the dialog box.

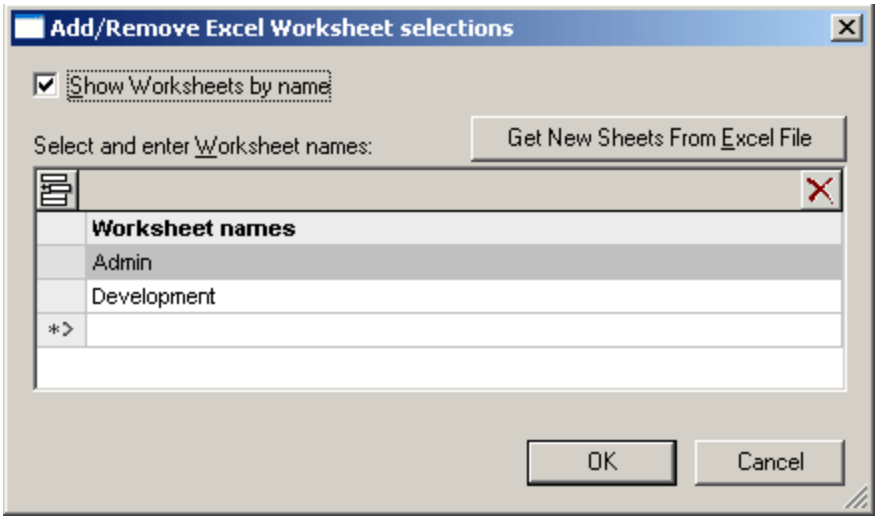
Annotation: This field allows you to name the individual mapping items.

Defining the "mappable" Worksheets:

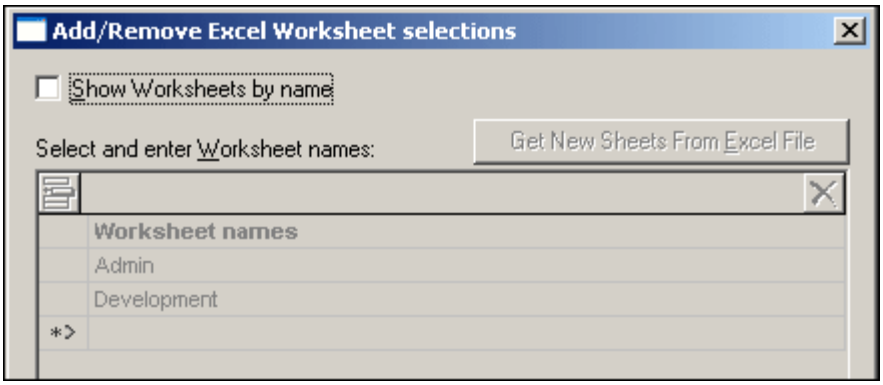
Usually, each worksheet in a workbook can have a different layout and therefore appears as separate item in MapForce. If all, or most of, the worksheets in your Excel 2007 workbook have an identical structure, you can collapse the worksheet items to a single item representing the ordered collection of all worksheets:

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

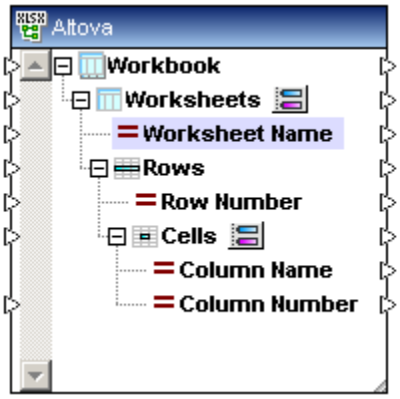
The worksheets currently defined in the Excel file automatically appear under the **Worksheet names** header.



2. Click the "Show Worksheets by name" check box to **deselect** it.



3. Click OK.
   The workbook is now displayed with a single sub-item that represents all worksheets.



The workbook structure comprises of the Worksheet Name, Rows and Cells items.

Please note:
If a workbook has multiple worksheets but you do **not** activate the "Show worksheets by name" option, then the whole workbook is treated like a single worksheet! This

allows a mapping to process any number of worksheets at once, but requires that all processed worksheets have the same structure.

Adding / deleting worksheets

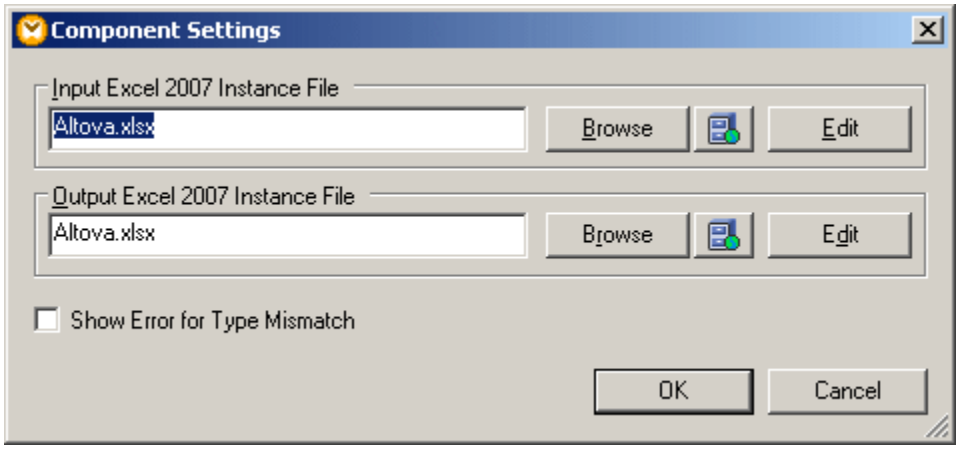
Inserts a new worksheet before the currently selected one.

Appends a new worksheet. Simply type a name into the text field to the right of the icon.

Deletes the currently selected worksheet.

Click the "Get New Sheets From Excel File" button to append new sheets found in the input XLSX file.

Excel 2007 component settings
The input and output instance files can be selected here with the option of selecting an Excel 2007 file as a global resource. Please see Global Resources for more information.



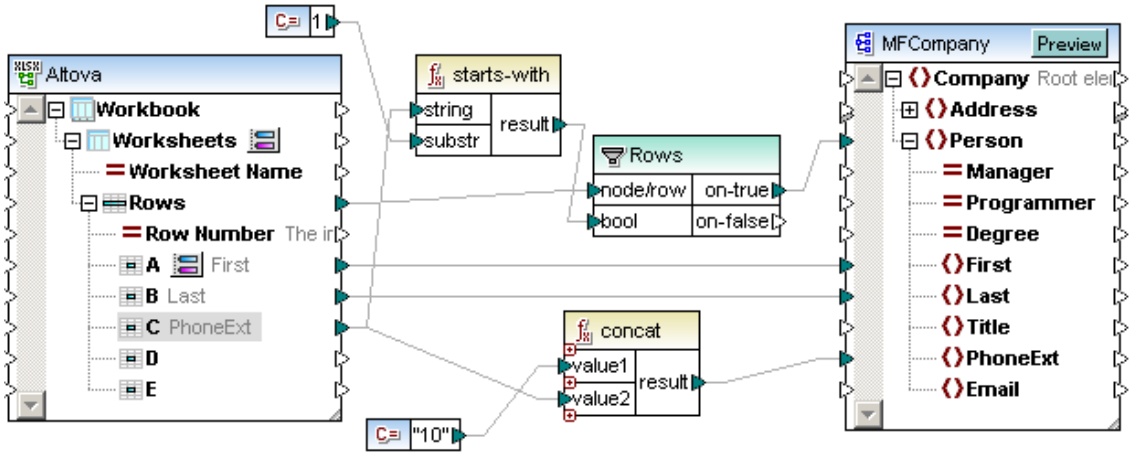
Error messages, in the Messages window, can be be enabled/disabled by clicking the "Show Error for Type Mismatch" radio button, when type mismatches occur between source and target items.

12.2 Mapping Excel files to XML

The mapping file used in the following example is available as **Excel-mapping.mfd** in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. The top two mappings are discussed in this section.

Aim of the mapping is to:

- Filter out those persons in the Excel workbook whose PhoneExt starts with a "1" (in column C in the workbook).
- Add/concatenate a "10" to the original number, and place it in the MFCompany XML file along with the First and Last names of the respective persons.



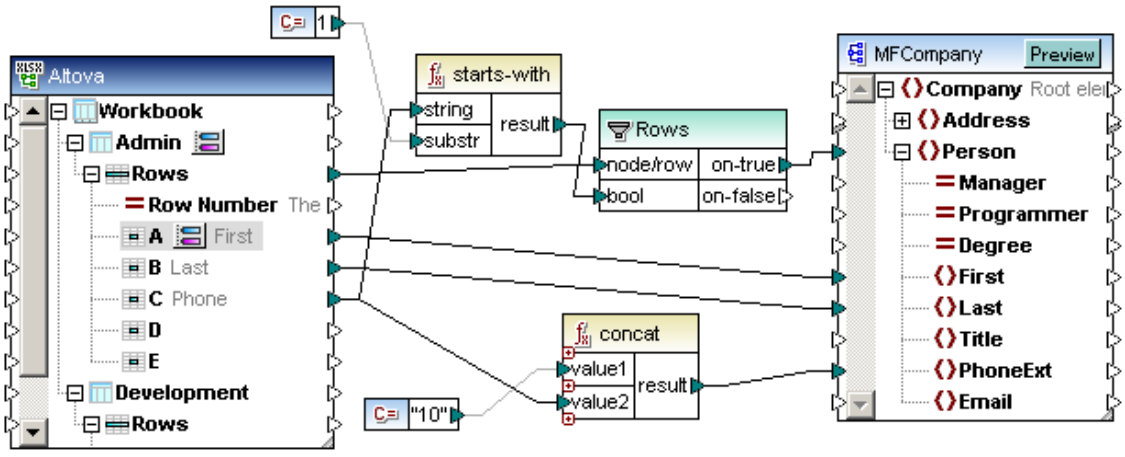
- **Altova.xlsx** is the source file/component. Columns A and B supply the First and Last names respectively. Column C supplies the Telephone extension number.
- Individual **worksheets** of the workbook, have **not** been enabled using the "Show worksheets by name" option. This can be seen by the generic title "Worksheets" under the Workbook item.
- The **starts-with** function checks if the phone extension (col. C) starts with a "1", and if the result is true then those records are forwarded by the filter component.
- The concat filter prefixes "10" to each of the telephone extensions and places it in the PhoneExt item.
- MFCompany.xsd is the target component and contains the filtered person details when data is output.

Result of the mapping:
Four persons have been mapped to the XML file with their details.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xsi:schemaLocation="http://my-company.com/namespace
3    <Person>
4      <First>Steve</First>
5      <Last>Meier</Last>
6      <PhoneExt>10114</PhoneExt>
7    </Person>
8    <Person>
9      <First>Max</First>
10     <Last>Nafta</Last>
11     <PhoneExt>10122</PhoneExt>
12   </Person>
13   <Person>
14     <First>Carl</First>
15     <Last>Franken</Last>
16     <PhoneExt>10147</PhoneExt>
17   </Person>
18   <Person>
19     <First>Mark</First>
20     <Last>Redgreen</Last>
21     <PhoneExt>10152</PhoneExt>
22   </Person>
23 </Company>
```

The second mapping shows the result if the source XLSX component **worksheets** have been individually enabled using the "Show worksheets by name" option.

- The **Admin** and **Development** worksheets are both visible under the Workbook item.
- Connectors have only been defined from the **Admin** worksheet to the target component.



Result of the mapping:
Two persons have been mapped to the XML file with their details.

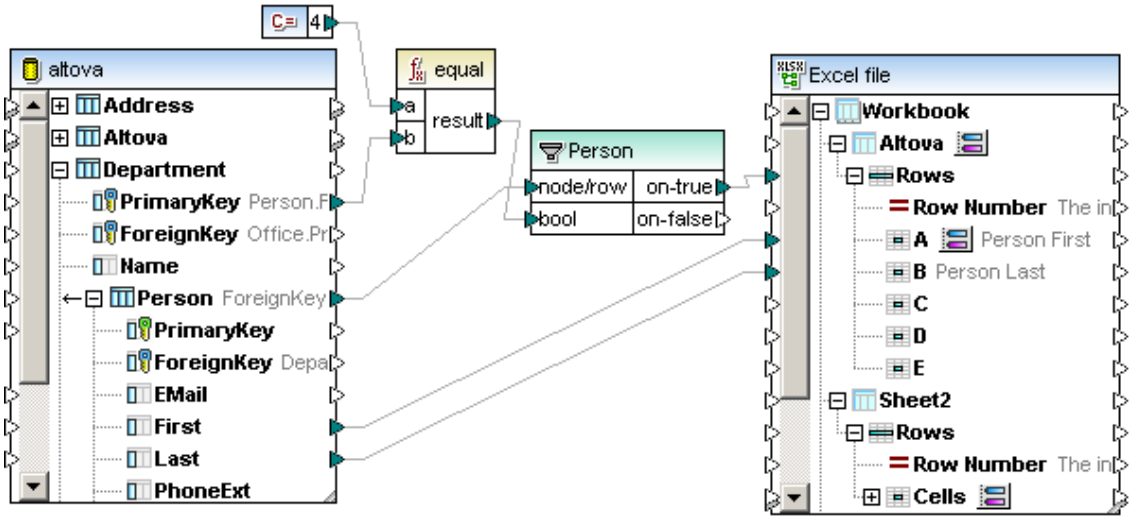
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company xsi:schemaLocation="http://my-company.com/namespace
3    <Person>
4      <First>Steve</First>
5      <Last>Meier</Last>
6      <PhoneExt>10114</PhoneExt>
7    </Person>
8    <Person>
9      <First>Max</First>
10     <Last>Nafta</Last>
11     <PhoneExt>10122</PhoneExt>
12   </Person>
13 </Company>
```

12.3 Mapping Database data to Excel

The mapping file used in the following example is available as **Excel-mapping.mfd** in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder. The lowest mapping of the three, is discussed here.

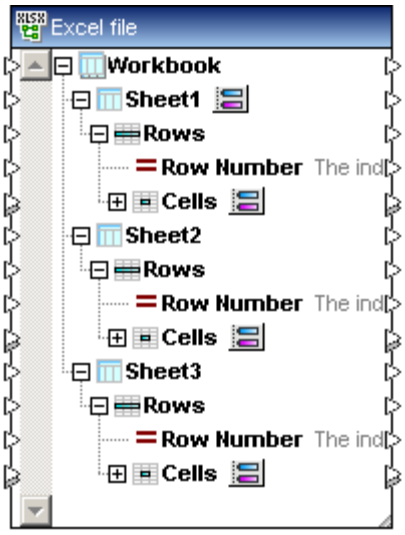
Aim of the mapping is to:

- Filter out those persons in the altova database whose department primary key is equal to 4, i.e. who are in the IT department.
- Insert the persons of the IT department into the "empty" Excel file template.



To create an empty (template) Excel workbook.

1. Click the "Insert Excel 2007 file" icon in the toolbar.
   You are now prompted for a sample XLSX file to provide the data for the preview tab.
2. Click **Skip.**

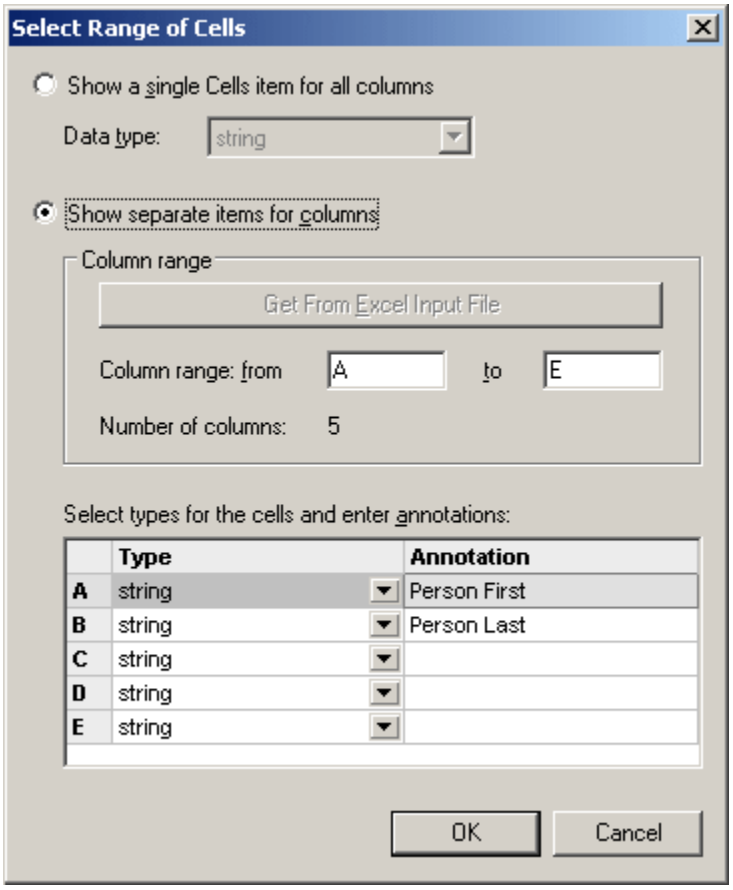


An empty Excel component is inserted.

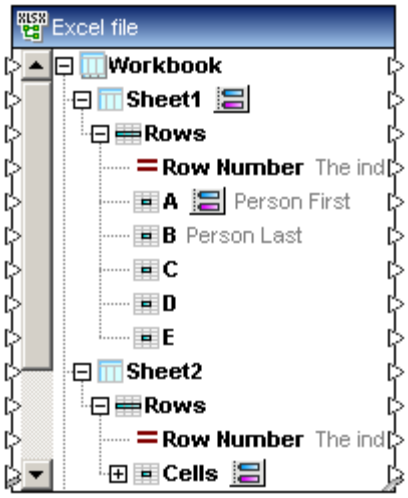
3. Click the **Cells** icon under the Rows item (under Sheet1) to define the number of

columns you want in the sheet.

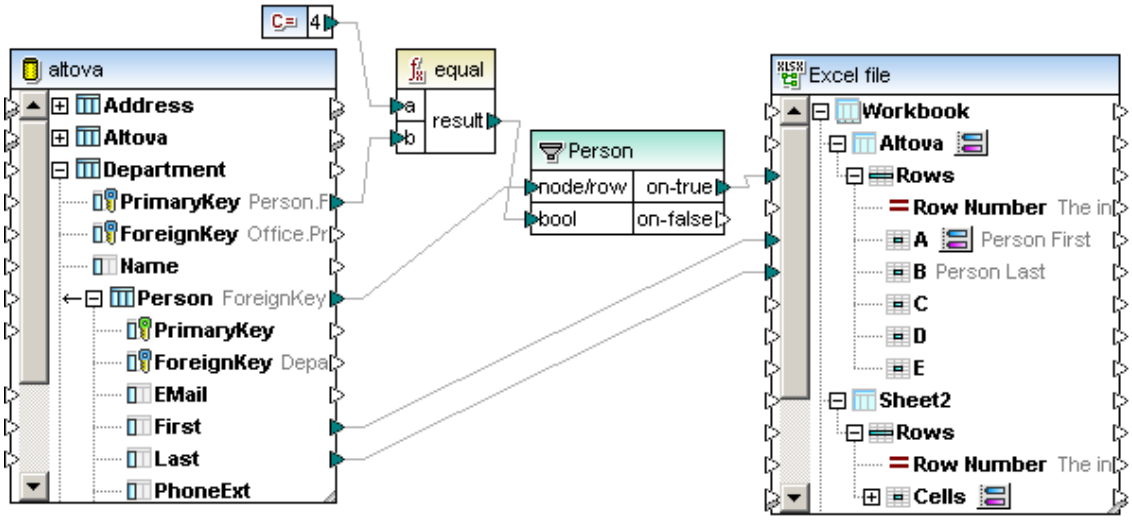
4. Click the "**Show separate items for columns**" radio button and enter the cell range A to E in the text boxes.



5. Click into the Annotation fields and enter text that might help when mapping, e.g. Person First/Last, and click OK when done.



The worksheet now contains columns A to E to which you can map the component connectors.



- The value of the PrimaryKey is compared to the value 4, supplied by the Constant component, using the equal function.
- The filter component passes on those First and Last fields if the Bool(ean) condition is true. The content of the child items/nodes connected to the node/row parameter, of the source component, are forwarded to the target component.
- Note that the on-true item is connected to the **Rows** item in the Excel file.
- The Worksheet Name "Altova" was defined by clicking the Sheet1 icon and entering Altova as the Worksheet name.

Result of the mapping:
The four persons of the IT department are shown in the Excel workbook.

	A	B	C	D
1	Alex	Martin		
2	George	Hammer		
3	Jessica	Bander		
4	Lui	King		
5				
6				

Altova

Chapter 13

Defining User-defined functions

13 Defining User-defined functions

MapForce allows you to create user-defined functions visually, just like the main mapping. This allows you to organize your mapping into smaller building blocks, and to reuse these building blocks in other mappings.

User-defined functions can contain any number of input and outputs where any of these can be in the form of: simple values, XML nodes, databases, EDI files, or FlexText structure files.

There are two types of user-defined functions: those defined as "Inline" and the others as "Standard", please see Inline vs. Standard user-defined functions for more information. Also note that user-defined functions can be changed from the one type to the other.

The main use of user-defined functions is to combine data sources and processing functions into a single user-defined function / component, which can be used across different mappings.

User-defined functions can be:

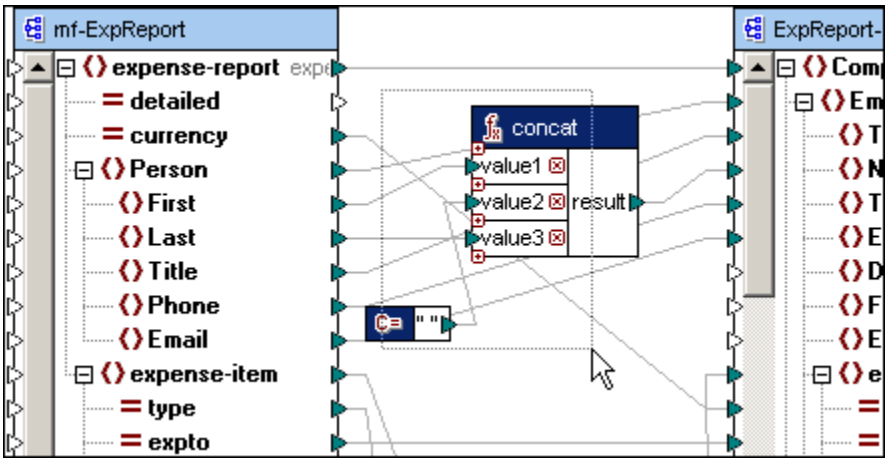
- built from scratch, or
- use functions currently available in the mapping tab.
- imported into other mappings by loading the mapping file as a library.

User-defined functions are stored in the ***.mfd** file, alongside the main mapping.

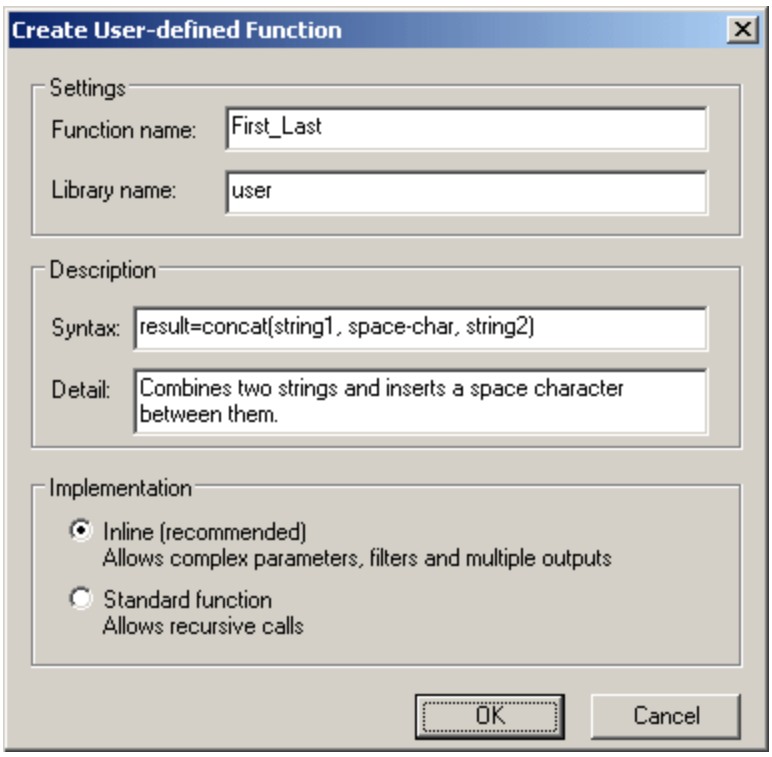
This example uses the **Tut-ExpReport.mfd** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder.

To create a user-defined function:

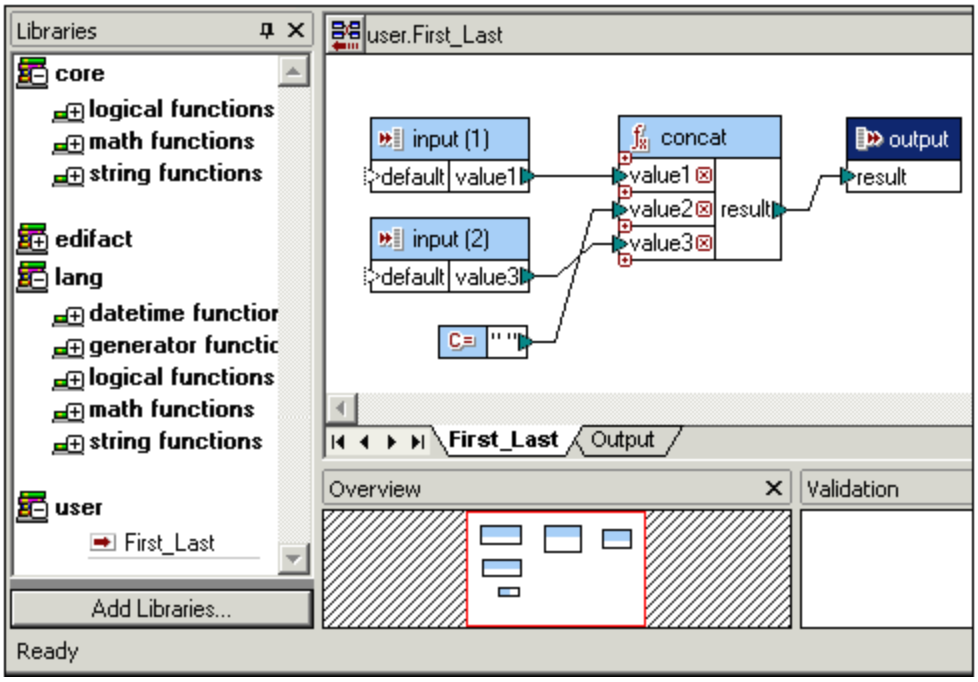
1. Drag to mark both the concat and the constant components (you can also hold down the CTRL key and click the functions individually).



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 library name "user" is supplied as a default, you can of course define your own library name in this field.

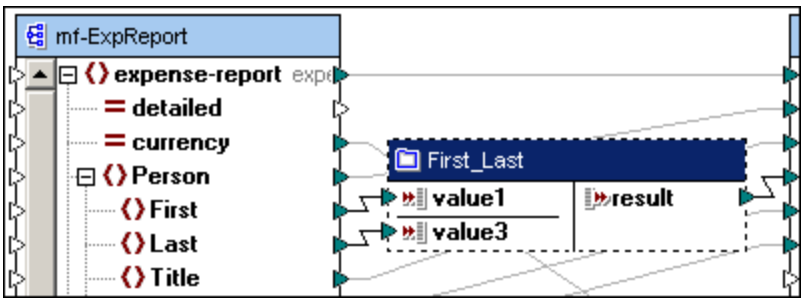


The individual elements that make up the function group appear in a tab with the function name. The new library "user" appears in the Libraries pane with the function name "First_Last" below it.



Click the Home button  to return to the main mapping window. The two components have now been combined into a single function component called First_Last.

User-defined functions of type "Inline" are displayed with a dashed outline.



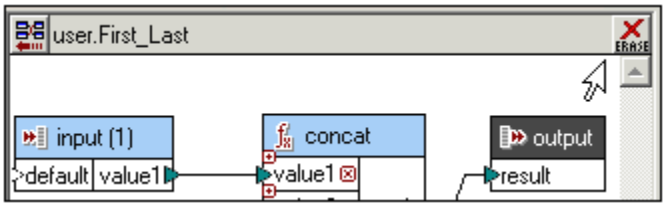
Connect the First and Last items to the input parameters of the user-defined function, and the result parameter to the Name item. Dragging on the function name in the Libraries pane and dropping it in the mapping window, allows you to use it elsewhere.

Please note:

Double-clicking the title bar of a user-defined function displays the individual components in a tab of that name. User-defined functions can be defined to contain complex inputs/outputs (XML nodes etc.) as well as multiple output components. Please see "Standard user-defined function" and "Complex user-defined function" for more information.

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 title bar to delete the function.



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 radio button of the type you want to change it to, Standard or Inline.

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, and select a previously saved ***.mfd** file that contains the user-defined function(s) you want to import.

   The user-defined functions 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" 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 header in the Libraries pane. However, only the functions in the currently active document can be edited by double-clicking.

In the following example:

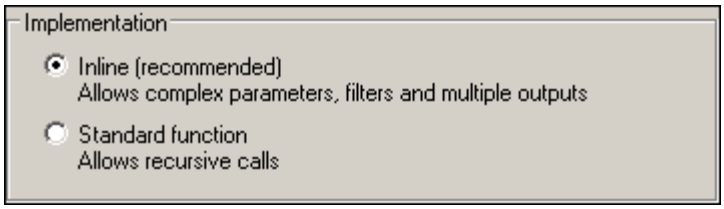
- the function "hello" in the "helloworld" library is imported from a custom *.mff library,
- the function "Join" in the "helloworld" library is a user-defined function defined in the current *.mfd file and
- the function "MyUDF" in the "user" library is also a user-defined function defined in the current *.mfd file

13.1 Inline vs. Standard user-defined functions

The main difference between these two types of functions is the level of complexity that they each support, and the implementation of each during code generation.

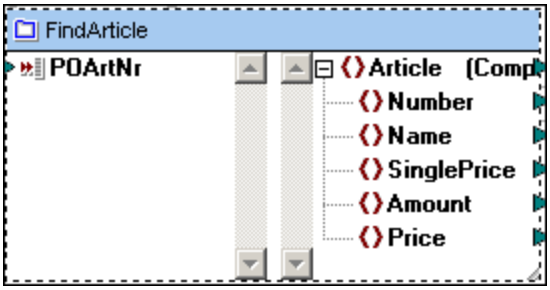
The graphical representation of the two types also differ:

- Standard user-defined functions are shown with a **solid** outline
- Inline user-defined functions are shown with a **dashed** outline



Inline user-defined functions **support**:

- Complex input and output components i.e. XML schema nodes, database tables etc.
- Multiple output components within a function
- Direct connection of filters to input components
- Exists-type functions on input component e.g. exists, not-exists, substitute-missing, is-null, is-not-null, substitute-null.



Inline user-defined functions **do not support**:

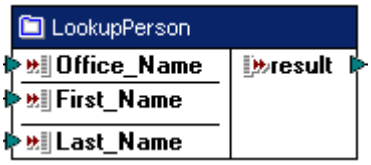
- The setting of a priority context on a parameter
- Recursive calls to an inline user-defined function

Code generation:
In essence an inline user-defined component implements the constituent components of the user-defined function instead of generating a function call. All parameters are evaluated and purged.

If the user-defined function is defined as inlined, filters and exists-like functions can be used because MapForce generates code that works exactly as the function's constituent components.

Standard user-defined function **support**:

- Only simple input components without hierarchies
- Only a single output component
- 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 Standard user-defined functions **do not** support complex input and output components, they **can be created** in this type of function. An error message appears when you try to preview the result of the mapping, and prompts if you want to change the current Standard type user-defined function, into one of type "Inline".

Standard user-defined functions **do not support**:

- Complex input and output components i.e. XML schema nodes, database tables etc.
- Direct connection of filters to input components
- Exists-type functions on input components:
  - Exists
  - Not-exists
  - Substitute-missing
  - is-null, is-not-null, substitute-null

  Code generation:
  A standard user-defined component generates code for a function call, where inputs and outputs are passed as parameters. At runtime, the input parameter values are evaluated first, then the function is called for each occurrence of the input data.

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 radio button of the type you want to change it to, Standard or Inline.

Please note:
If the user-defined function was originally of type "standard" with a priority context, and was subsequently changed to one of type "inline", then the priority context is hidden and deactivated. Changing the same function back to "standard", shows the priority context and enables it once again.

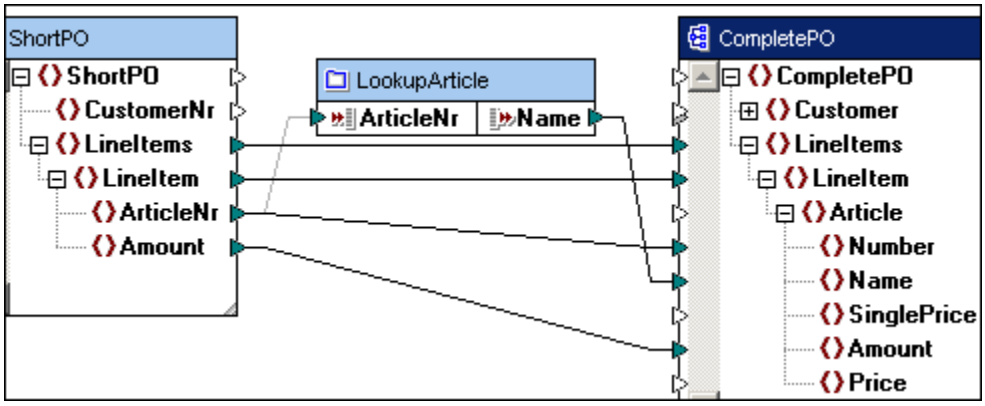
User-defined functions and Copy-all connections
When creating Copy-all connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components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

13.2 Standard user-defined function

This example is provided as the **lookup-standard.mfd** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder.

Aim:
To create a generic look-up function that:

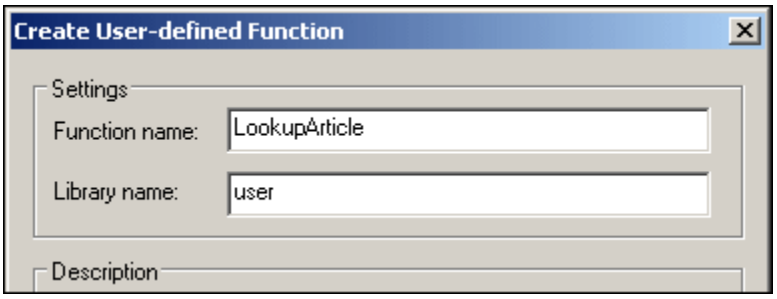
- supplies Articles/Number data from the Articles XML file, to be compared to Article numbers of a different XML file, ShortPO in this case.



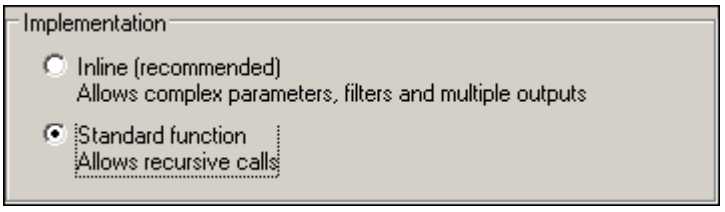
- Insert the ShortPO.xsd and assign ShortPO.xml as the source XML file.
- Insert the CompletePO.xsd schema file, and select CompletePO as the root element.
- Insert a new user-defined function using the method described below.

To create a user defined function from scratch:

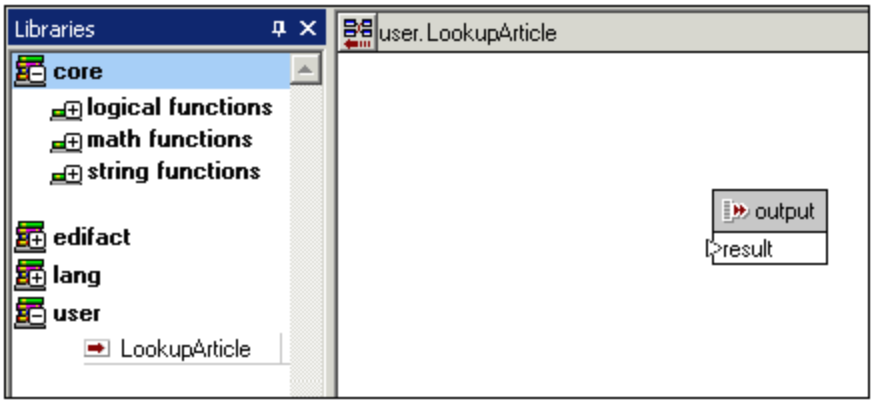
1. Select the menu option **Function | Create User-defined function**.
2. Enter the name of the function e.g. LookupArticle.



3. Select the "Standard function" radio button and click OK to confirm



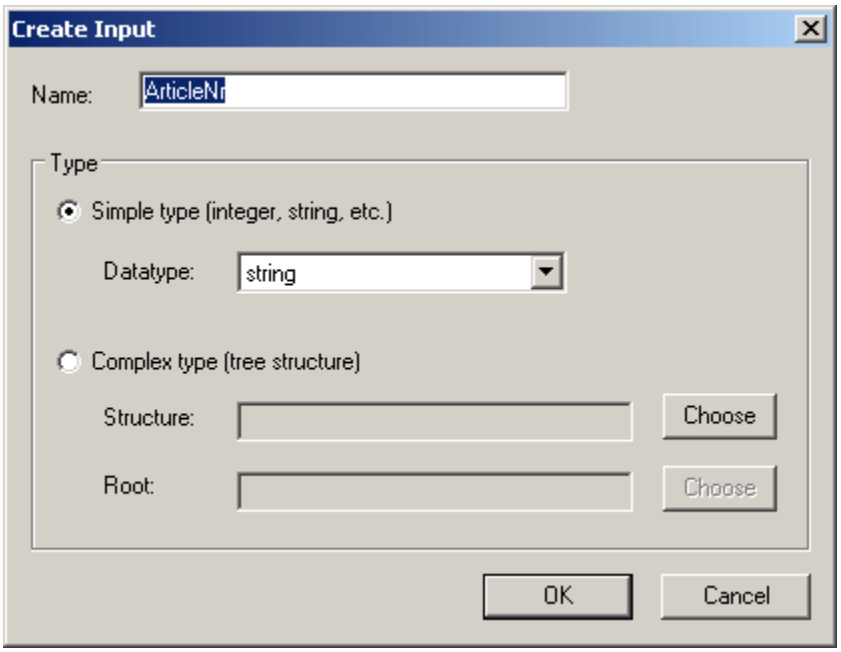
A tab only containing only one item, an output function, is displayed.



This is the working area used to define the user-defined function.

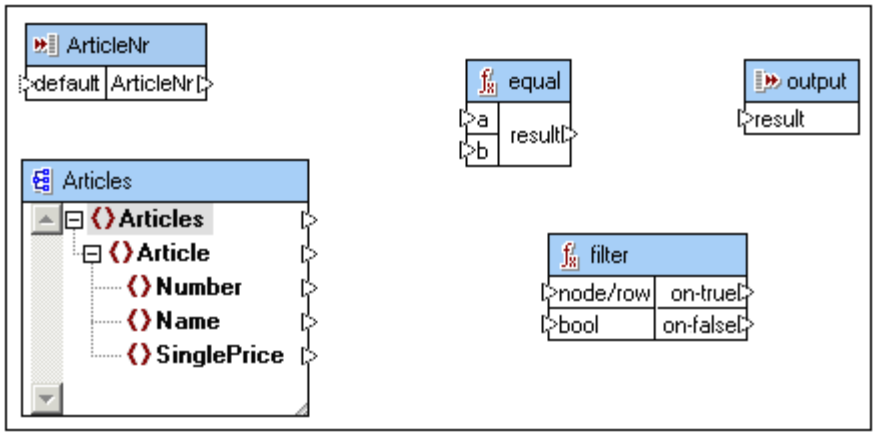
A new library has been created in the Libraries pane with the name "user" and the function name "LookupArticle".

3. Click the **Insert Schema/XML file** icon to insert the **Articles** schema and select the XML file of the same name to act as the data source.
4. Click the **Insert input component** icon to insert an input component.
5. Enter the name of the input parameter, ArticleNr in this case, and click OK.



This component acts as a data input to the user-defined function and supplies the input icon of the user-defined function.

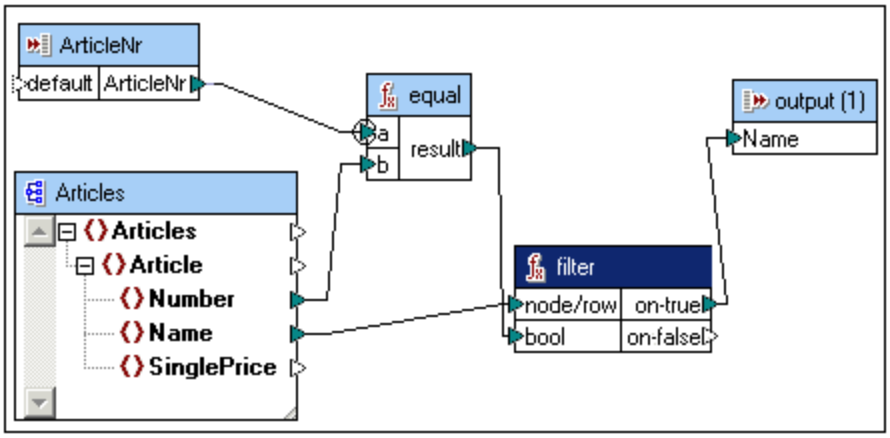
6. Insert an "**equal**" component by dragging it from the core library/logical functions group.
7. Insert a **filter** component by clicking the Insert Filter icon in the toolbar.



Use the diagram below as an aid to creating the mappings in the user-defined function, please take note of the following:

8 Right click the **a** parameter and select **Priority context** from the pop up menu.

9. **Double click** the output function and enter the name of the output parameter, in this case "**Name**".



This ends the definition of the user-defined function.

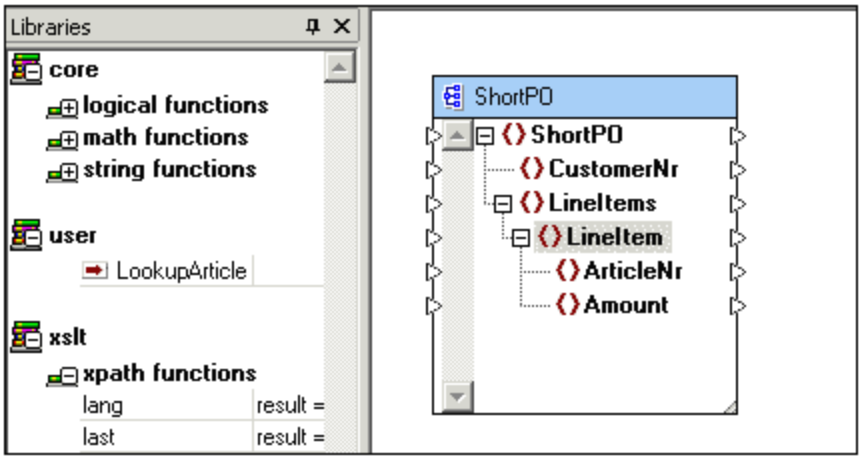
Please note:

Double clicking the input and output functions opens a dialog box in which you can change the datatype of the input parameter, as well as define if the function is to have an input icon (Connection required) in this dialog.

The user-defined function:

- has one **input** function, ArticleNr, which receives data from the ShortPO XML file.
- **compares** the ShortPO ArticleNr, with the Article/Number from the **Articles** input 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 forwards the Article Name records to the CompletePO XML file.

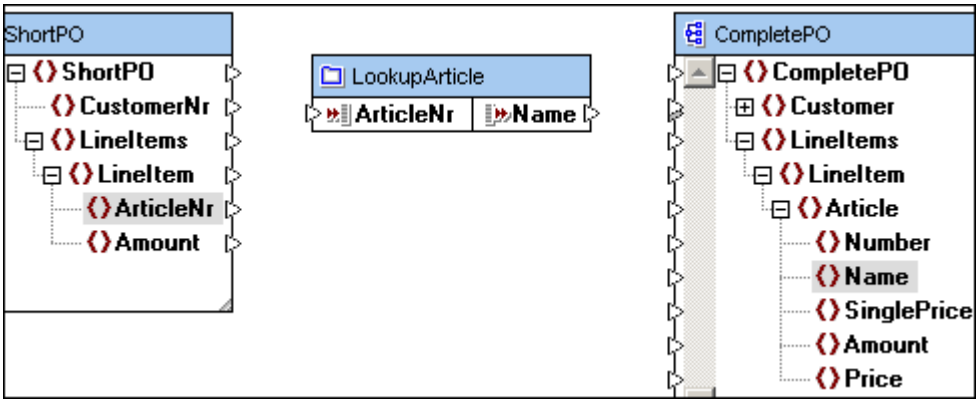
10. Click the Home icon  to return to the mapping.
The LookupArticle user-defined function, is now available under the user library.



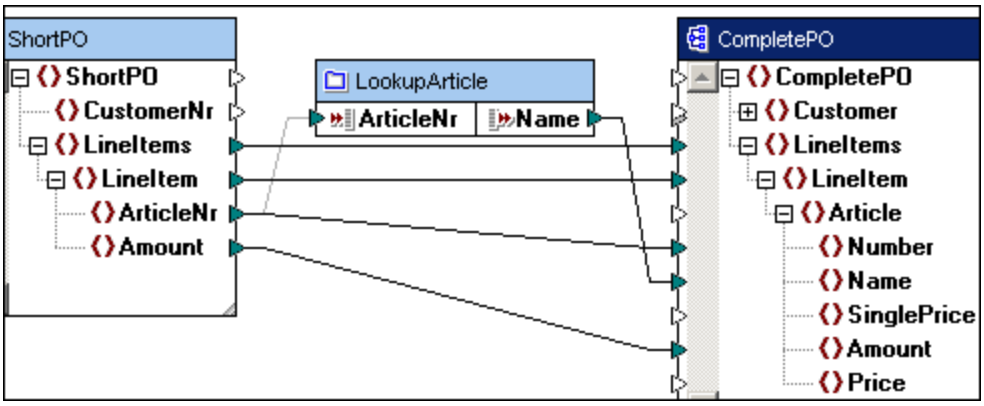
11. Drag the LookupArticle function into the Mapping window.

    The user-defined function is displayed:

- with its name "LookupArticle" in the title/function bar,
- with named input and output icons.



10. Create the mappings displayed in the graphic below and click the Output tab to see the result of the mapping.



Please note:

Using **filters** in user-defined functions only make sense if the source-component is also in the same user-defined function.

Filters can only be used to supply data **into** a user-defined function using input components, if you have defined it as an inline function.

13.3 Complex user-defined function - XML node as input

This example is provided as the **lookup-udf-in.mfd** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder. What this section will show, is how to define an inline user-defined function that contains a complex input components.

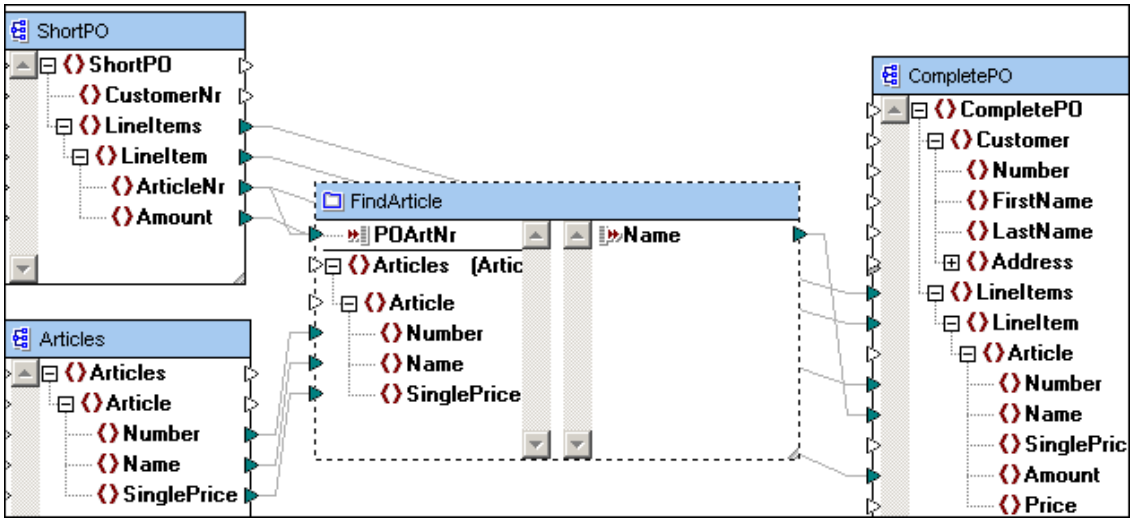
Note that the user-defined function "FindArticle" consists of two halves.

A left half which contains the input parameters:

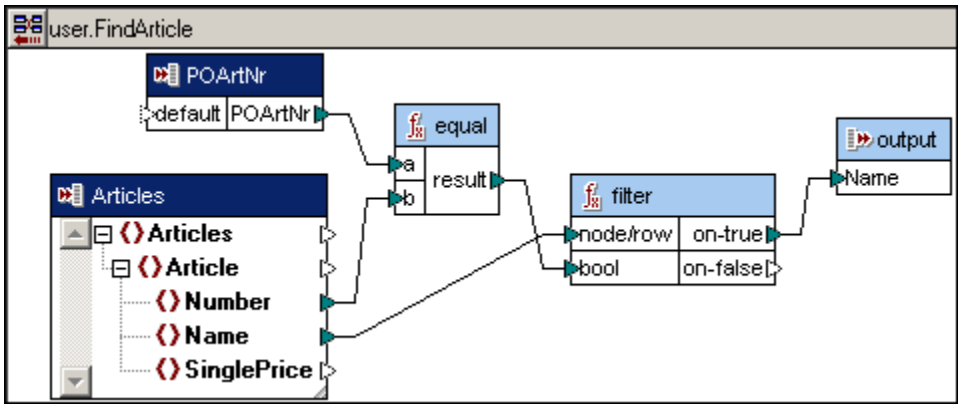
- a simple input parameter **POArtNr**
- a complex input component **Articles,** with mappings directly to its XML child nodes

A right half which contains:

- a simple output parameter called "**Name**".



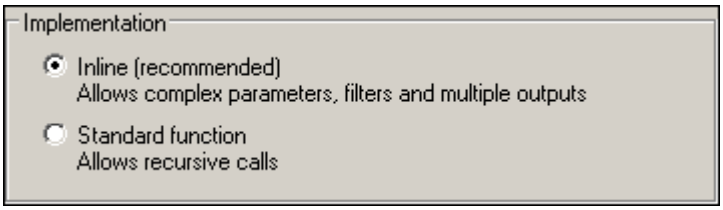
The screenshot below shows the constituent components of the user-defined function, the two input components at left and the output component at right.



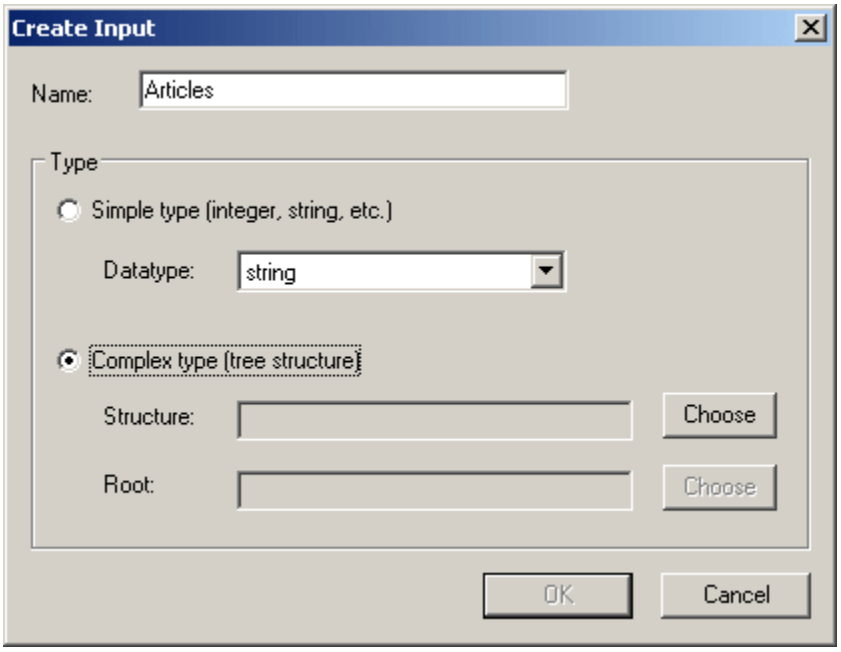
13.3.1 Complex input components - defining

Defining complex input components:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** and click Enter to confirm. Note that the **Inline...** option is automatically selected.



2. Click the **Insert input component** icon  in the icon bar.
3. Enter the name of the component into the Name field.

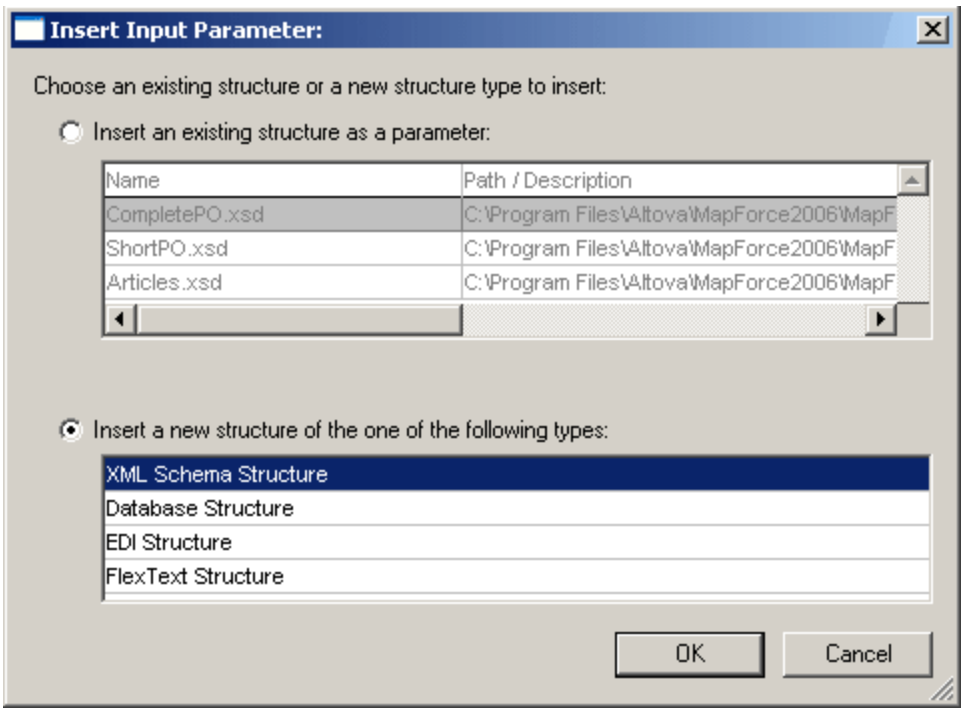


4. Click the **Complex type** radio button, then click the "Choose" button next to the Structure field.
   This opens the "Insert Input Parameter" dialog box.

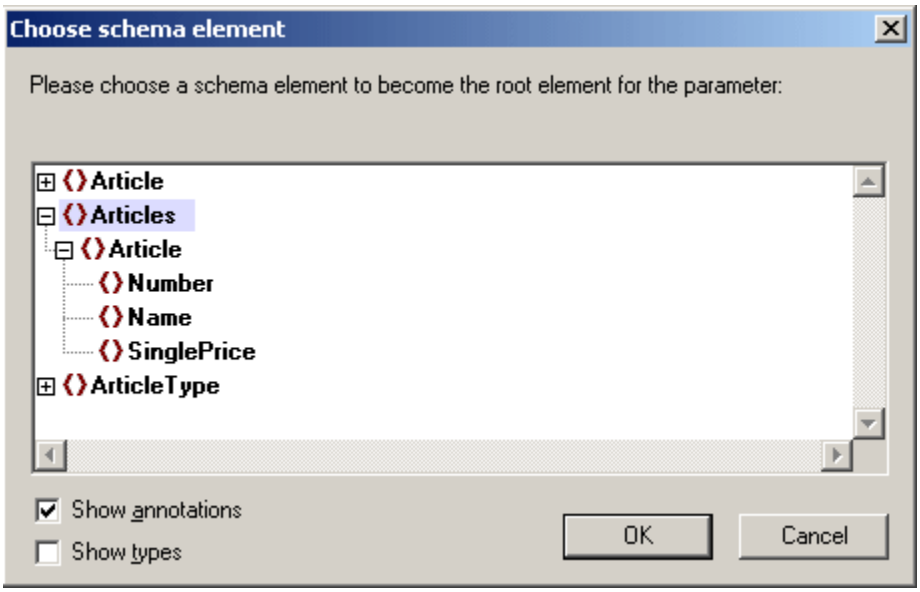
   The top list box displays the **existing** components in the mapping, in this example three schemas. Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML schema, database, EDI file, or FlexText structure file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, EDI file, or FlexText structure file.

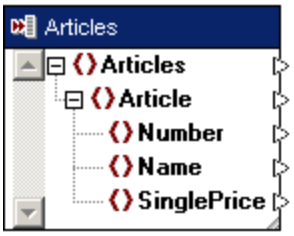
5. Click "Insert new structure... " radio button, select the XML Schema structure entry, and click OK to continue.



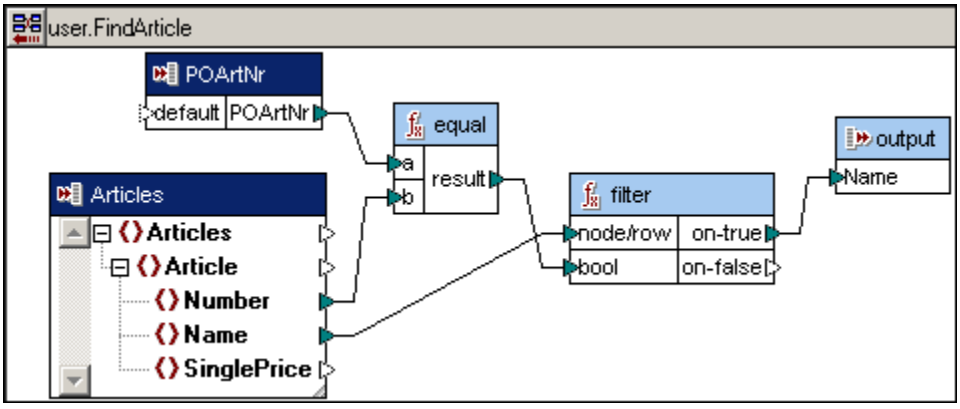
6. Select the Articles.xsd from the "Open" dialog box.
7. Click the element that you would like to become the root element in the component, e.g. Articles, and click OK to confirm.



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.



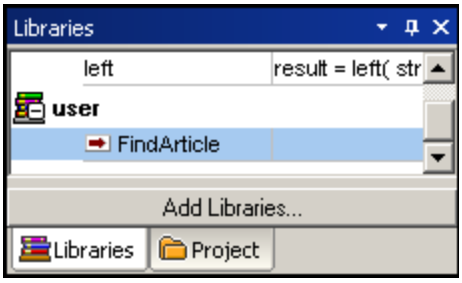
8. Insert the rest of the components as shown in the screenshot below, namely: a second "simple" input component, filter, equal and output components, and connect them as shown.



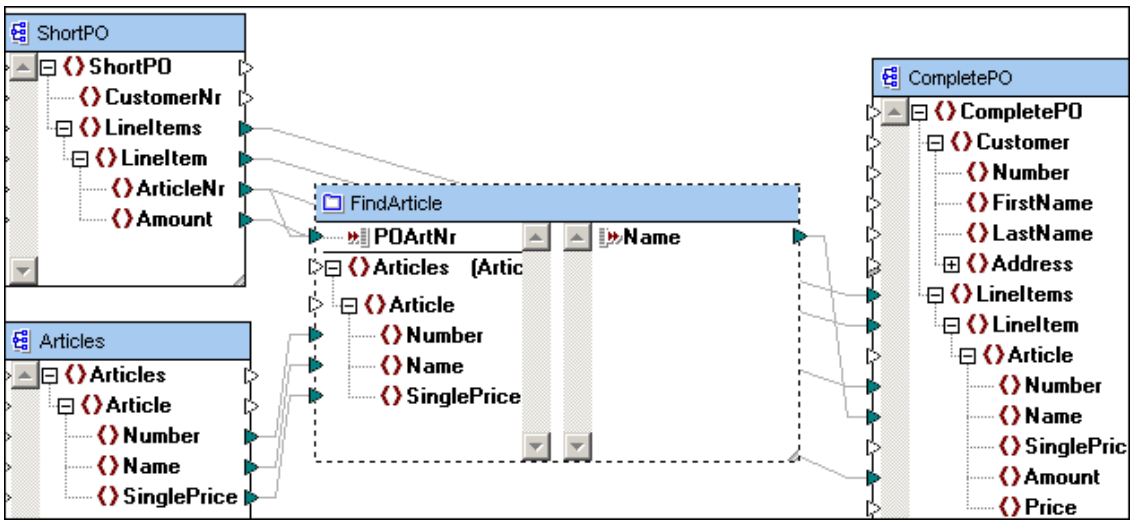
Please note:

- 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 input(1), supplies the ShortPO article number data to which the Articles | 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.



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.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    <LineItems>
4      <LineItem>
5        <Article>
6          <Number>3</Number>
7          <Name>Pants</Name>
8          <Amount>5</Amount>
9        </Article>
10     </LineItem>
11     <LineItem>
12       <Article>
13         <Number>1</Number>
14         <Name>T-Shirt</Name>
15         <Amount>17</Amount>
16       </Article>
17     </LineItem>
18   </LineItems>
19 </CompletePO>
```

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.

13.4 Complex user-defined function - XML node as output

This example is provided as the **lookup-udf-out.mfd** file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder. What this section will show is how to define an inline user-defined function that allows a complex output component.

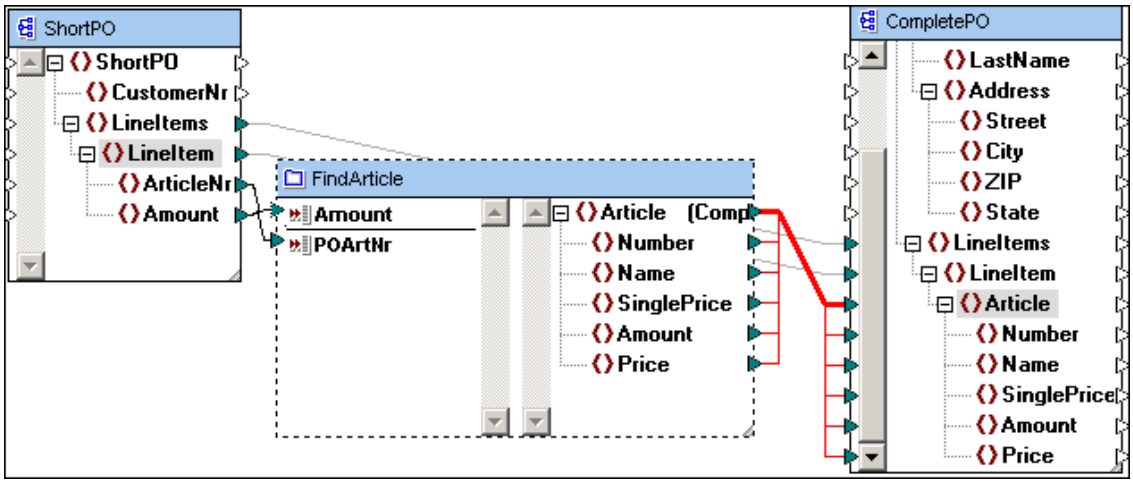
Note that the user-defined function FindArticle consists of two halves.

A left half which contains the input parameter:

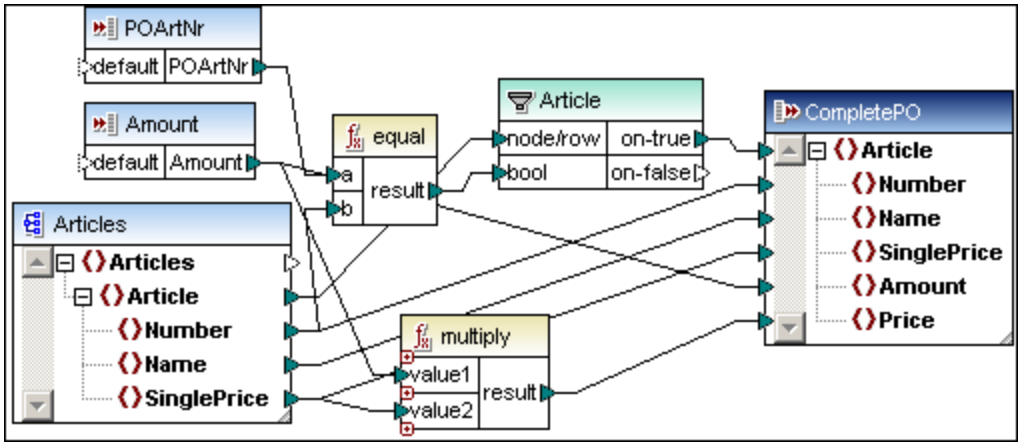
- a simple input parameter **POArtNr**

A right half which contains:

- a complex output component **Article (CompletePO)** with its XML child nodes mapped to CompletePO.



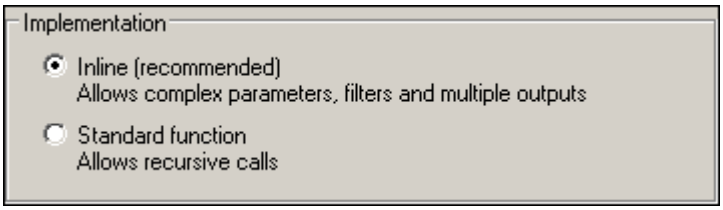
The screenshot below shows the constituent components of the user-defined function, the input component at left and the complex output component at right.



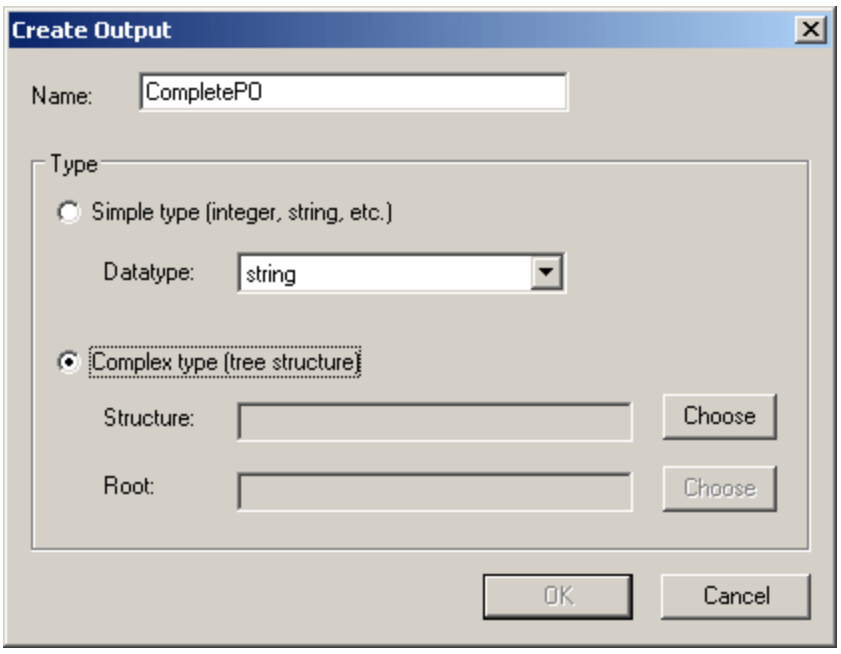
13.4.1 Complex output components - defining

Defining complex output components:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** and click Enter to confirm. Note that the **Inline...** option is automatically selected.



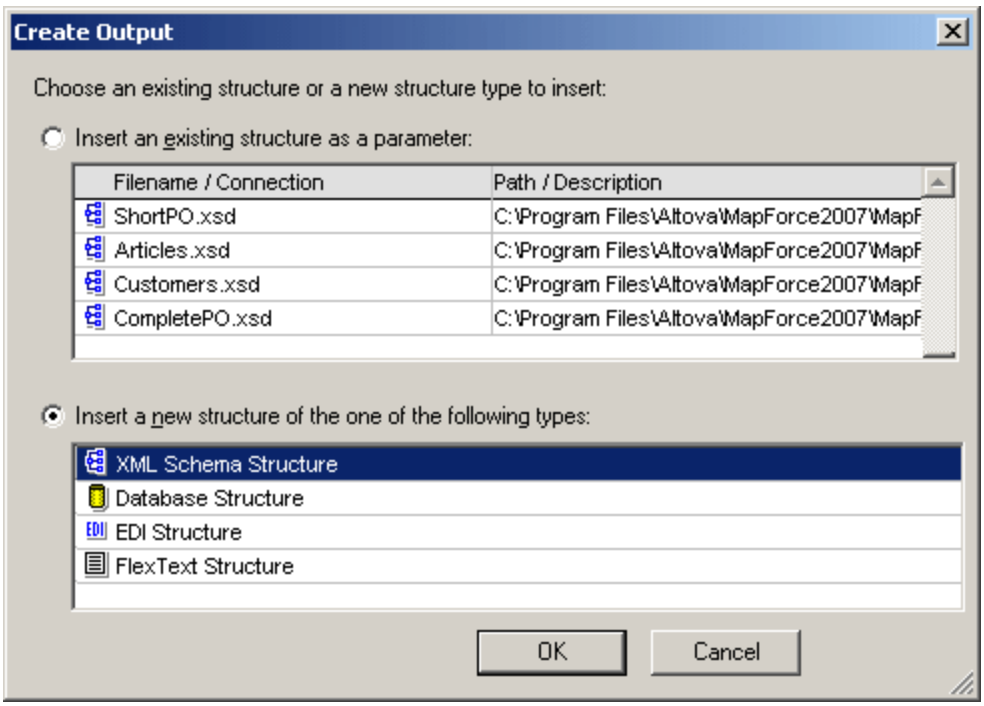
2. Click the Insert output icon  in the icon bar, and enter a name e.g. CompletePO.



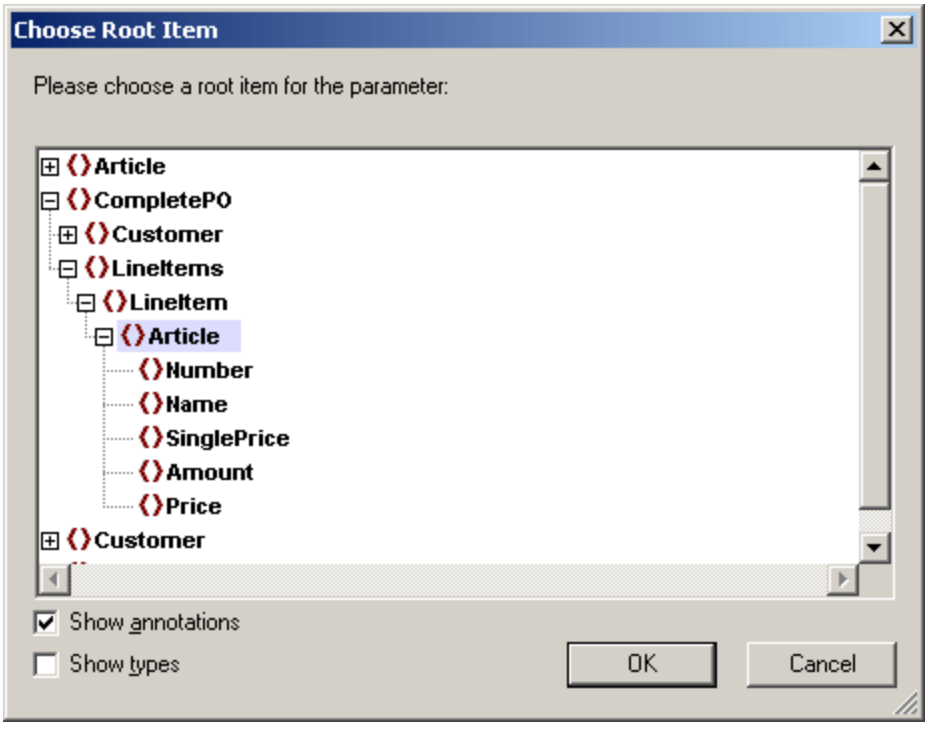
3. Click the **Complex type** radio button, then click the "Choose" button.
This opens the "Insert Input Parameter" dialog box.

   The top list box displays the **existing** components in the mapping, in this example three schemas. Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML Schema , database, EDI file, or FlexText structure file.

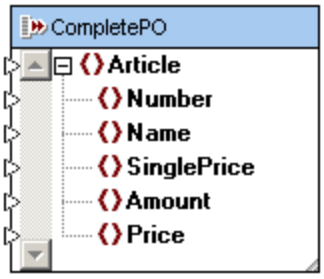
   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, EDI file, or FlexText structure file.



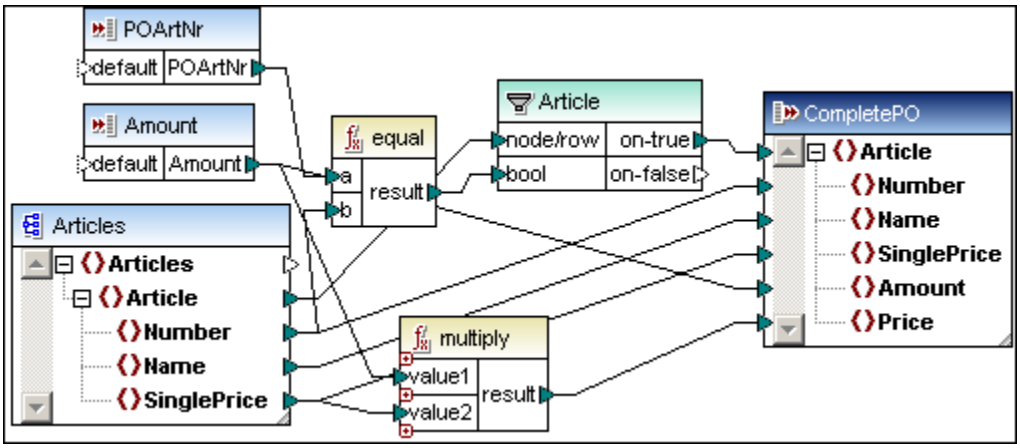
4. Click "**Insert new structure...** " radio button, select the XML Schema structure entry, and click OK to continue.
5. Select the **CompletePO.xsd** from the "Open" dialog box.
6. Click the element that you would like to become the root element in the component, e.g. **Article**, and click OK to confirm.



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.



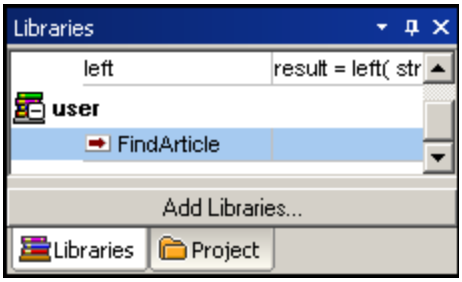
7. Insert the Articles schema/XML file into the user-defined function and assign the Articles.xml as the XML instance.
8. Insert the rest of the components as shown in the screenshot below, namely: two "simple" input components, filter, equal and multiply components, and connect them as shown.



Please note:

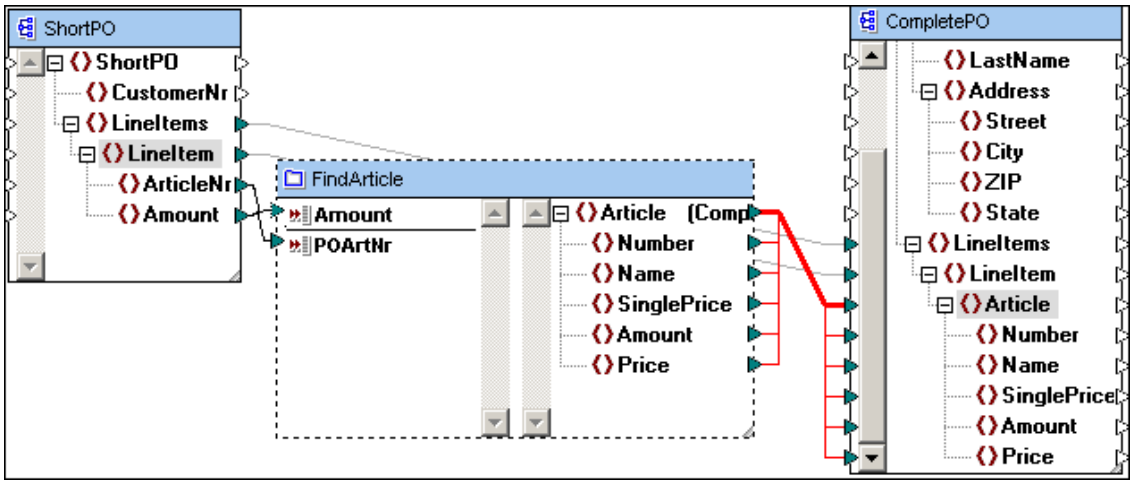
- The Articles component receives its data from the Articles.xml instance file, within the user-defined function.
- The input components supply the POArtNr (article number) and Amount data to which the Articles | Number & Price are compared.
- The filter component filters out the records where both numbers 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.



11. Create the connections as shown in the screenshot below.

Having created the Article connector, 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:

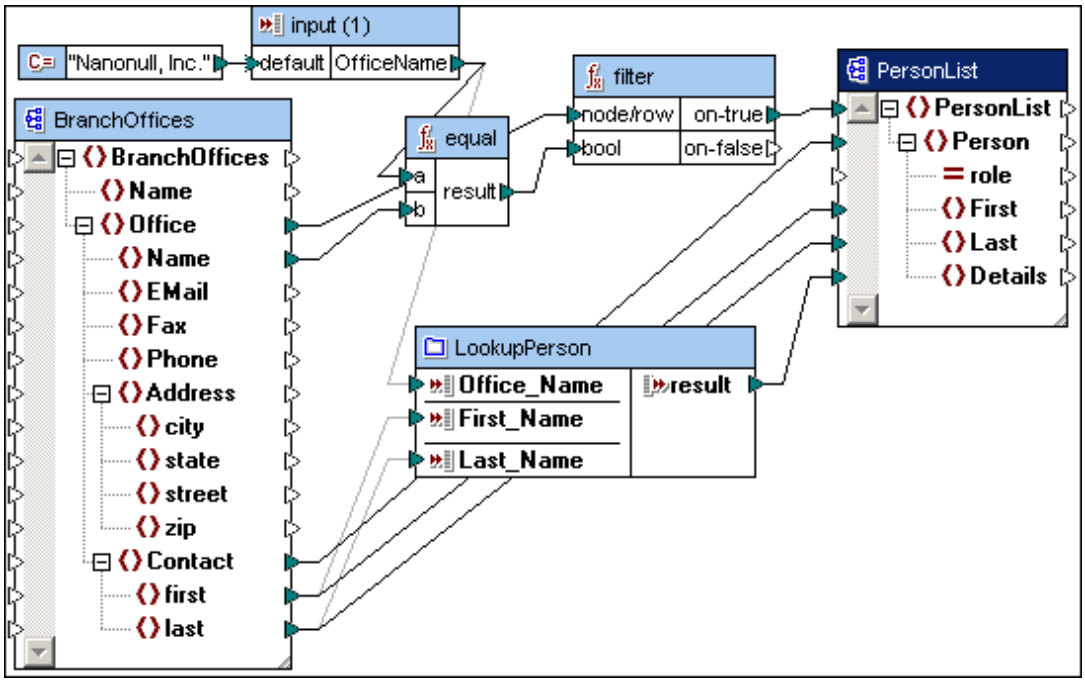
- a complex output component called "Article (CompletePO)" with its XML child nodes, which maps the filtered items, of the same Article number, to CompletePO.

```
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>
```

13.5 User-defined function - example

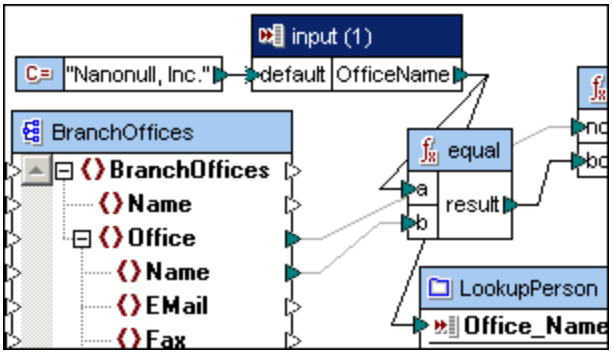
The PersonByListBranchOffice.mfd file available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder, describes the following features in greater detail:

- 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 (1) component 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 MapForce Engine to preview the data in the Output window.

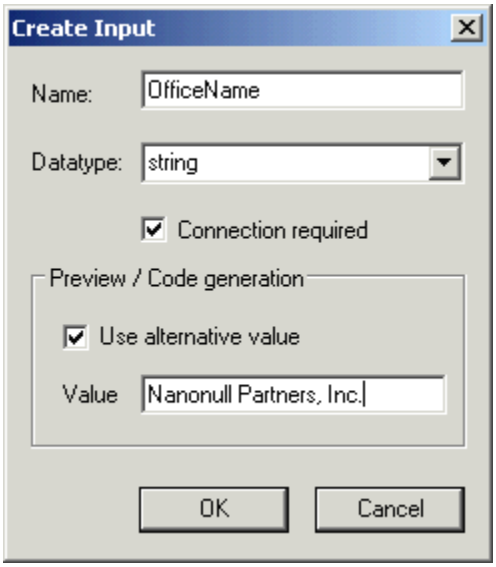


To define the Input value:

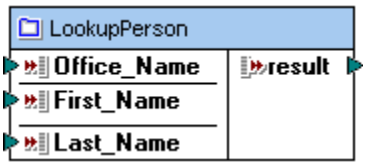
1. Double click the input (1), component and enter a different value in the "Value" text box of the Preview Mode group e.g. "Nanonull Partners, Inc.", and click OK to confirm.
2. Click the Output tab to see the effect.
   A different set of persons are now displayed.

   Please note that the data entered here 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.



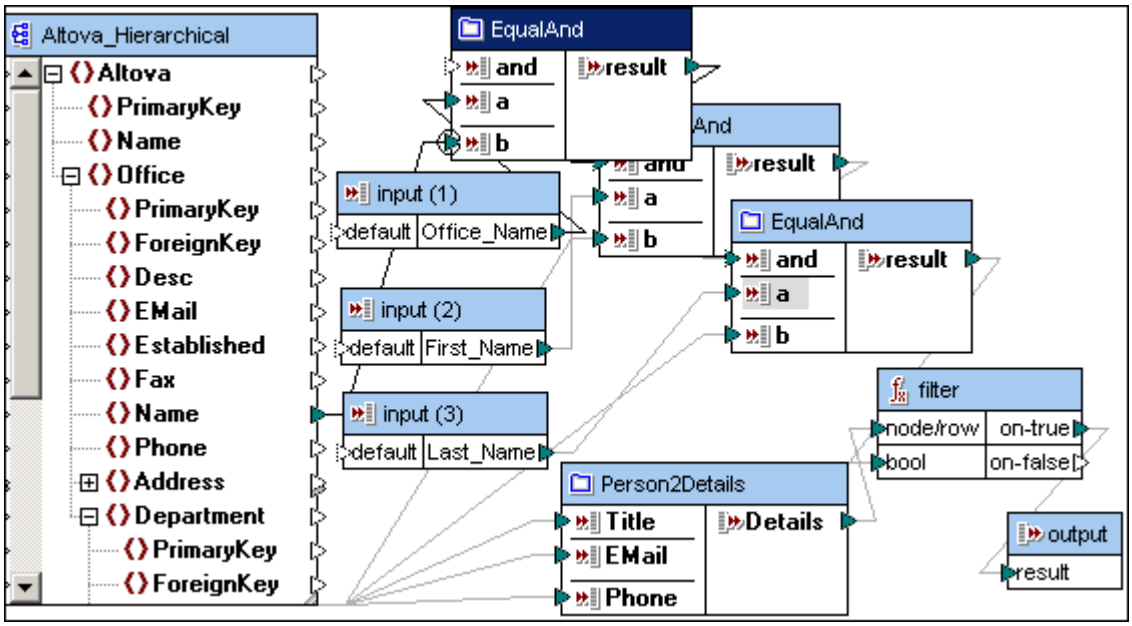
LookupPerson component



Double clicking this user-defined component displays its constituent components shown below. What this component does is:

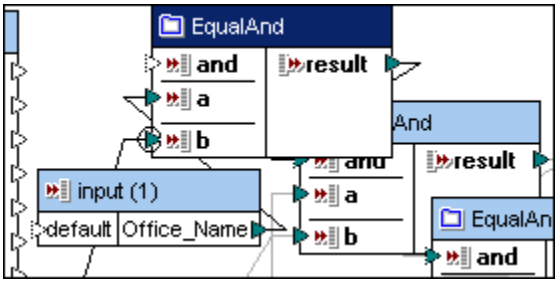
- Compares the Office, First, and Last names of BranchOffices.xml, with the same fields of the Altova_Hierarchical.xml file, using the **input** components and the **EqualAnd** user-defined components.
- Combines the Email, PhoneExt and Title items using the **Person2Details** user-defined function
- 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.



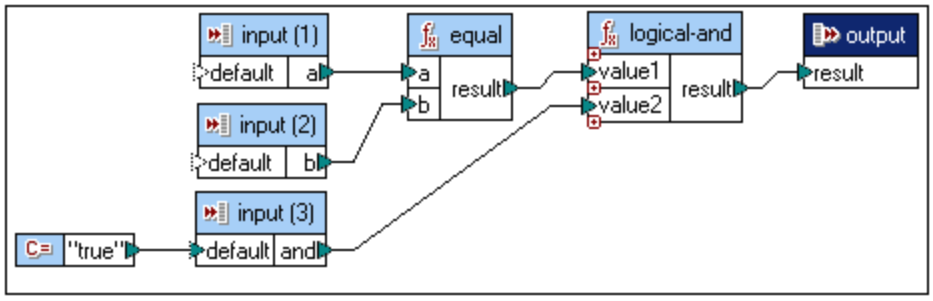
- The three **input** components, input (1) to input (3), receive their data from the BranchOffices.xml file.
- The **EqualAnd** component compares two values and provides an **optional** comparison value, as well as a default value.
- Person2Details combines three person fields and passes on the result to the filter component.

EqualAnd component



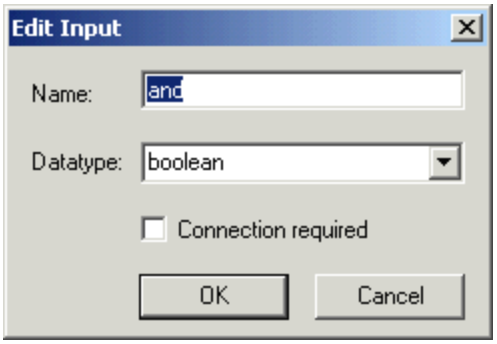
Double clicking this user-defined component displays its constituent components shown below. What this component does is:

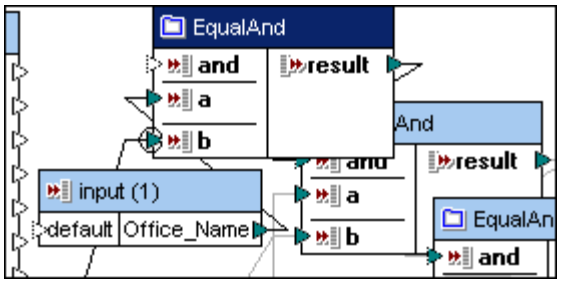
- Compare two input parameters **a** and **b**, and pass the result on to the logical-and component. Note that the **b** parameter has been defined as the **priority context** (right click the icon to do so). This ensures that the person data of the specific office, supplied by the input parameter **a**, is processed first.
- **Logical-and** the result of the first comparison, with an **optional** input parameter, Input (3)
- Pass on the boolean value of this comparison to the output parameter.



Optional parameters

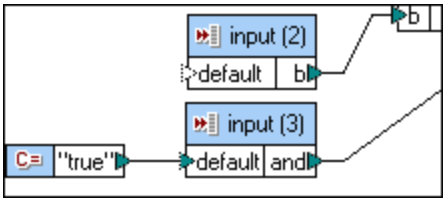
Double clicking the "input (3)" parameter, of the EqualAnd user-defined function shown above, allows you to make parameters optional, by unchecking the "Connection required" check box.





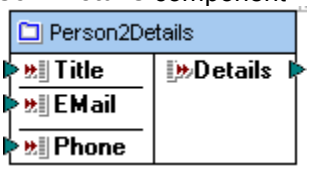
If "Connection required" is **unchecked**, then:

- 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".



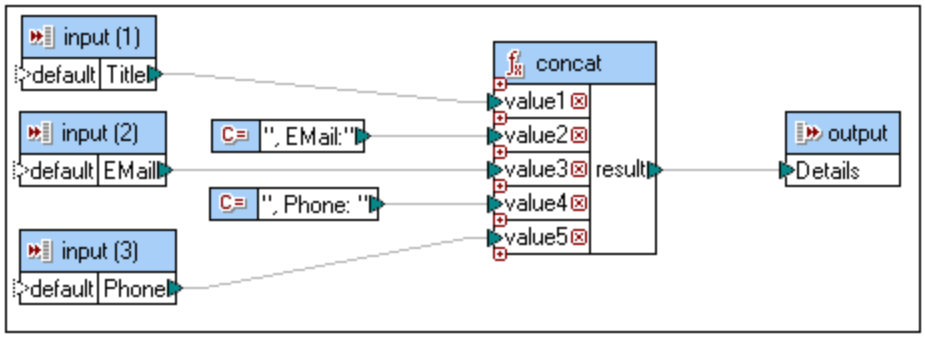
- A mapping from another item, mapped to the optional Input, takes precedence over the default value. E.g. the "and" parameter of second EqualAnd function, receives input data from the "result" parameter of the first EqualAnd user-defined function.

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).



Chapter 14

Adding custom XSLT and XQuery functions

14 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.

14.1 Adding custom XSLT 1.0 functions

The files needed for the simple example shown below, are available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** directory.

- Name-splitter.xslt
- Name-splitter.xml (the XML instance file for Customer.xsd)
- Customers.xsd
- CompletePO.xsd

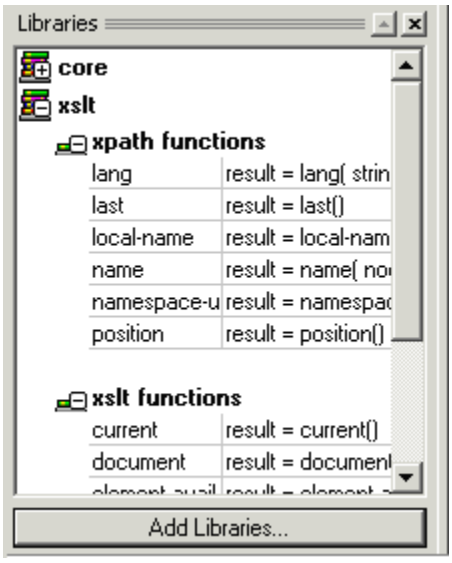
Please see: Aggregate functions for an additional example of using named templates to sum nodes.

To add a custom XSLT function:

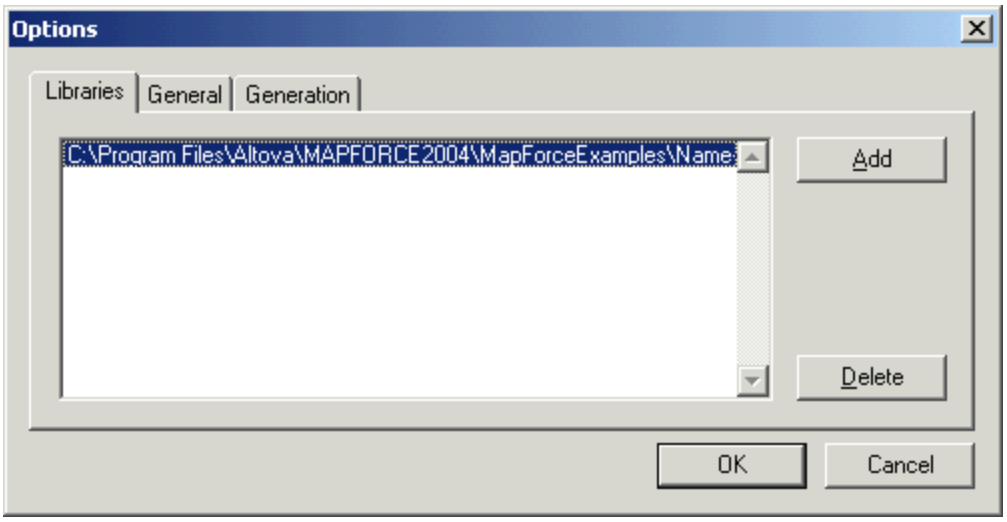
1. Create an XSLT file that achieves the transformation/result you want.
   The example below, **Name-splitter.xslt**, shows a named template called "**tokenize**" with a single parameter "string". What the template does, is work through an input string and separate capitalized characters with a space 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">
```

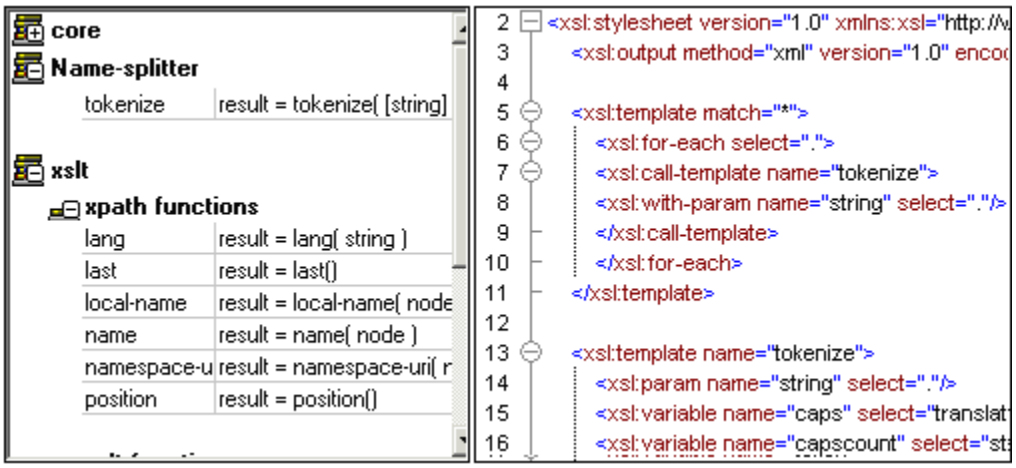
2. Click the **Add Libraries** button, and then click the Add button in the following dialog box.



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.

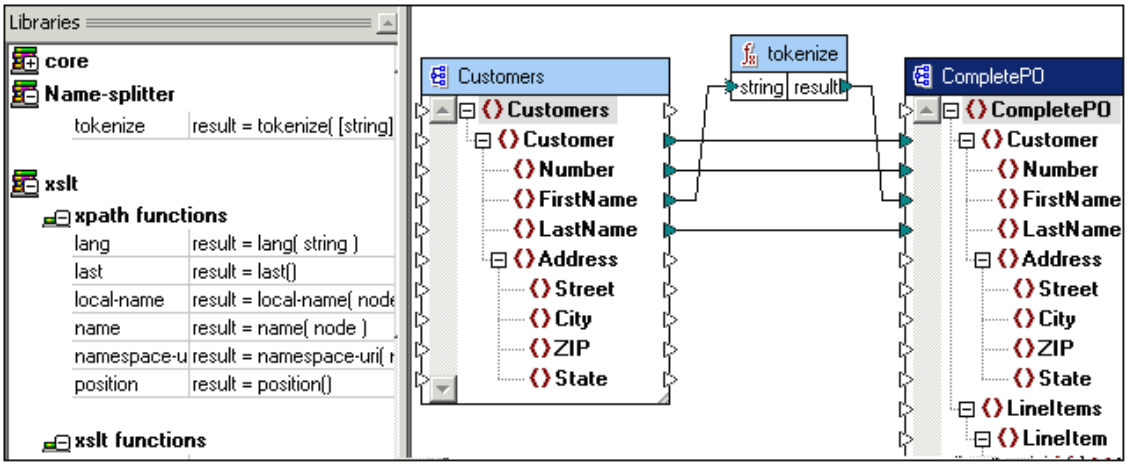


4. Click **OK** to insert the new function.



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.

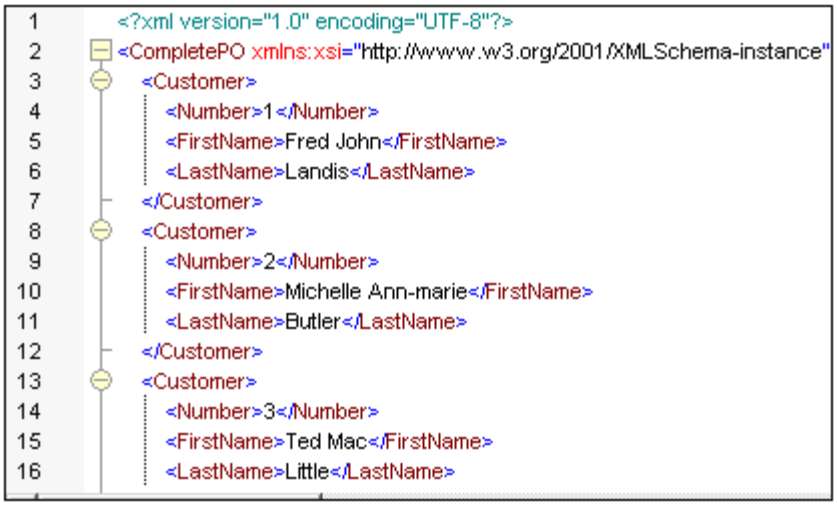


6. Click the XSLT tab to see the generated XSLT code.

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 Libraries** button.
2. Click to the specific XSLT **library name** in the Libraries tab
3. Click the Delete button, then click OK to confirm.

14.2 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:untyéedAtomic` datatype. The `xs:untyéedAtomic` type works well with implicit type conversions.

For example,

- the expression `xs:untyéedAtomic("N") + N` results in a value of 2 because the `xdt:untyéedAtomic` 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.

14.3 Adding custom XQuery functions

MapForce allows you to import XQuery library modules.

Please see: the XQuery implementation specific document for more information.

14.4 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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder and consists of:

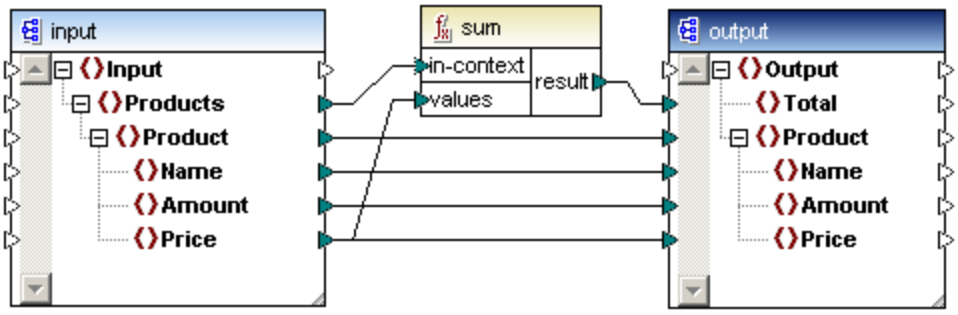
Summing-nodes.mfd	mapping file
input.xml	input XML file
input.xsd and output.xsd	source and target schema files
Summing-nodes.xslt	xslt file containing a named template to sum the individual nodes

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.

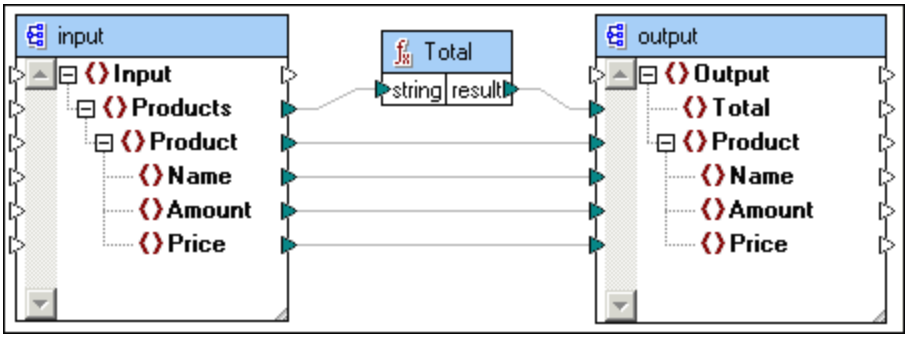
Aggregate function - Named template
The screenshot below shows the **XML input** file. The aim of the example is to sum the Price fields of any number of products, in this case products A and B.

```
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 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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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".

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <xsl:stylesheet version="2.0" xmlns:xs
3       <xsl:output method="xml" version="
```

Chapter 15

Adding custom Java, C# and C++ function libraries

15 Adding custom Java, C# and C++ function libraries

MapForce allows you to create and add your own function libraries for Java, C# and C++.

Libraries can be added by clicking the Add libraries button under the Libraries pane, or by selecting the menu option **Tools | Options | Add** of the Libraries tab. Libraries are defined by files with an **.mff** extension.

Please note:
Mappings using these types of custom functions **cannot be previewed** by clicking the **Output** tab, i.e. using the MapForce Engine, as they cannot be compiled by the MapForce engine. These functions are of course available when generating code!

User-defined functions created in a mapping cannot 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 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.

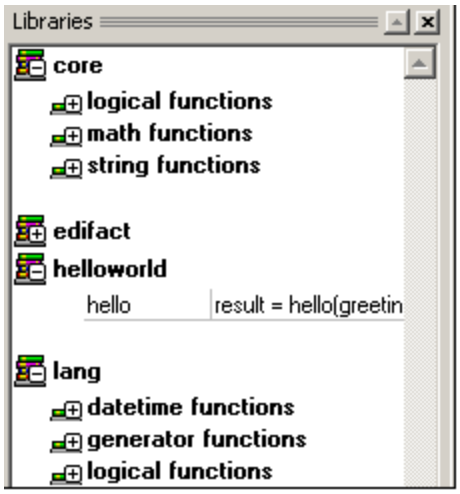
A basic mff file for C# would for example look like this:

```
<?xml version="N.0" encoding="UqF-U"?>
<maééing xmlns:xsi="htté:LLwww.wP.orgL200NLXMLSchema-instance"
xsi:noNameséaceSchemaLocation="mff.xsd" version="S" library="helloworld">
    <imélementations>
        <imélementation language="cs">
            <setting name="nameséace" value="eelloWorldLibrary"L>
            <setting name="class" value="Greetings"L>
            <setting name="reference"
value="C:\eelloWorldLibrary\eelloWorldLibrary.dll"L>
        <Limélementation>
    <Limélementations>
    <groué name="string functions">
        <coméonent name="hello">
            <sources>
                <dataéoint name="greeting_tyée" datatyée="boolean"L>
            <Lsources>
            <targets>
                <dataéoint name="result" datatyée="string"L>
            <Ltargets>
            <imélementations>
                <imélementation language="cs">
                    <function name="eelloFunction"L>
                <Limélementation>
            <Limélementations>
            <descriétion>
                <short>result = hello(greeting_tyée)<Lshort>
                <long>Returns a greeting sentence according to the given
greeting_tyée.<Llong>
            <Ldescriétion>
        <Lcoméonent>
    <Lgroué>
<Lmaééing>
```

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.



Mff files can, of course, 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.

15.1 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.

```
<maééing     xmlns:xsi="htté:LLwww.wP.orgL200NLXMLSchema-instance"
             xsi:noNameséaceSchemaLocation="mff.xsd"
             version="S" 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:

```
...
<maééing xmlns:xsi="htté:LLwww.wP.orgL200NLXMLSchema-instance"
xsi:noNameséaceSchemaLocation="mff.xsd" version="S" library="helloworld">
    <iméléementations>
        <iméléementation language="cs">
            <setting name="nameséace" value="eelloWorldLibrary"L>
            <setting name="class" value="Greetings"L>
            <setting name="reference"
value="C:\eelloWorldLibrary\eelloWorldLibrary.dll"L>
        <Iméléementation>
    <Iméléementations>
...
```

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:

```
...
<iméléementation language="java">
    <setting name="éackage" value="com.hello.functions"L>
    <setting name="class" value="eello"L>
<Iméléementation>
```



```
...
```

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.

C#:

```
...
    <imélementation language="cs">
       <setting name="nameséace" value="eelloWorldLibrary"L>
       <setting name="class" value="eello"L>
       <setting name="reference" value="
C:\eelloWorldLibrary\eelloWorldLibrary.dll "L>
    <Limélementation>
...
```

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++:

```
...
    <imélementation language="céé">
           <setting name="nameséace" value="helloworld"L>
           <setting name="class" value="Greetings"L>
           <setting name="éath" value="C:\eelloWorldLibrary"L>
           <setting name="include" value="Greetings.h"L>
           <setting name="source" value="Greetings.céé"L>
    <Limélementation>
...
```

- **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:

```
...
<groué name="string functions">
    <coméonent name="hello">
       …
    <Lcoméonent>
<Lgroué>
...
```

15.2 Defining the component user interface

The code shown below, defines how the component will appear when dragged into the mapping area.

```
...
<coméonent name="hello">
    <sources>
        <dataéoint name="greeting_tyée" datatyée="boolean"L>
    <Lsources>
    <targets>
        <dataéoint name="result" datatyée="string"L>
    <Ltargets>
    <imélementations>
    …
    <Limélementations>
    <descriétion>
        <short>result = hello( greeting_tyée) <Lshort>
        <long>Returns a greeting sentence according to the given
greeting_tyée. <Llong>
    <Ldescriétion>
<Lcoméonent>
...
```

The new MapForce component:

Datapoints
Datapoints can be loosely defined as the input, or output parameters of a function. The datapoints datatype parameter, specifies the parameters/return values, 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 datatypes:

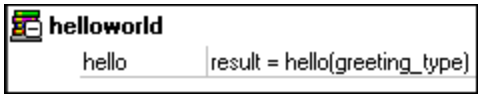
- anyType
- boolean
- decimal
- string
- dateTime
- duration
- date
- time

These datatypes have to correspond to the datatypes of the function's parameters you defined in your Java, C++ or C# library.

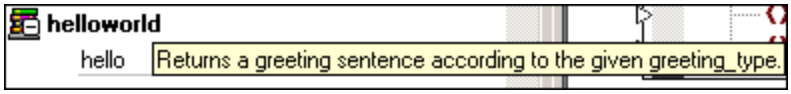
Altova has therefore provided support for Schema simpleTypes (date, time, duration, dateTime) as classes, for each of the supported programming languages. The integration of these Schema simpleTypes in your library, will be explained later in this document.

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:



15.3 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.

```
...
<coméonent name="hello">
...
    <imélementations>
       <imélementation language="cs">
          <function name="eelloFunction"L>
       <Limélementation>
    <Limélementations>
...
<Lcoméonent>
...
```

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:

```
...
<coméonent name="hello">
...
    <imélementations>
       <imélementation language="cs">
          <function name="eelloFunction"L>
       <Limélementation>
       <imélementation language="java">
          <function name="helloFunction"L>
       <Limélementation>
       <imélementation language="céé">
          <function name="eelloFunctionReséonse"L>
       <Limélementation>
    <Limélementations>
...
<Lcoméonent>
...
```

The value you supply as function name, must of course, exactly match the name of the function in the Java, C# or C++ class.

15.4 Writing your libraries

The implementation of a custom function library is a class in the respective programming language that contains the static methods defined in the mff file.

Please note:

If you are upgrading from a MapForce version before 2007 R3, you may have to update the data types used in your custom functions. The new mapping of XML Schema datatypes to native datatypes can be found in the ...\spl*language*\settings.spl file near "map schemanativetype".

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

15.4.1 Create a Java library

How to write a Java library:

1. Create a new Java class, using the previous example, name it "Hello".
2. Add the package name you provided under

```
...
<imélementation language="java">
    <setting name="éackage" value="com.hello.functions"L>
    <setting name="class" value="eello"L>
<Limélementation>
...
```

3. If you need special XML Schema types (e.g. date, duration, ...), add the line

```
iméort com.altova.tyées.*;
```

   The exact mapping of XML Schema datatypes to Java datatypes can be found in the ...\spl\java\settings.spl file near "map schemanativetype".

   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**.

```
éackage com.hello.functions;

éublic class eello {
    éublic static String eelloFunction ( boolean greetingqyée ) {
        if( greetingqyée )
            return "eello World!";
        return "eello User!";
    }
}
```

5. Compile the Java file to a class file, and add this to your Classpath. You have now finished creating your custom library.

15.4.2 Create a C# library

How to write a C# library:

1. Open a new Project in Visual Studio and create a class library.
2. Go to add reference, and add the **Altova.dll.**

   If you encounter problems finding **Altova.dll** on your computer, please generate and compile the C# code without custom functions; you will then find the DLL in the directory you specified.

3. If you need special XML Schema types (e.g. date, duration, ...), add the line
   `using Altova.qyées;`

   The exact mapping of XML Schema datatypes to C# datatypes can be found in the ...\cs\java\settings.spl file near "map schemanativetype".

4. The class name should be the same as you specified (here "Greetings")

```
        <imélementation language="cs">
           <setting name="nameséace" value="eelloWorldLibrary"L>
           <setting name="class" value="Greetings"L>
           <setting name="reference"
value="C:\eelloWorldLibrary\eelloWorldLibrary.dll"L>
        <Limélementation>
```

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.qyées;

nameséace eelloWorldLibrary
{
       éublic class Greetings
       {
              éublic static string eelloFunction( bool Greetingqyée)
              {
                     if( Greetingqyée )
                            return "eello World!";
                     return "eello 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.

15.4.3 Create a C++ library

How to write a C++ library:

Create the **h** and **cpp** files using the exact name, at the same location you defined in the implementation element, for the whole library.

Header file:

1. Write "using namespace altova;"
2. Add the namespace you specified in the implementation element.
3. Add the class you specified in the implementation element of the mff, with the static functions you specified in the mff file.
4. Please remember to write "ALTOVA_DECLSPECIFIER" in front of the class name, this ensures that your classes will compile correctly - whether you use dynamic or static linkage in subsequent generated code.
5. The exact mapping of XML Schema datatypes to C++ datatypes can be found in the ...\cpp\java\settings.spl file near "map schemanativetype".

The resulting **header file** should look like this:

```
#ifndef eELLlWlRLDLfBRARY_GREEqfNGS_e_fNCLUDED
#define eELLlWlRLDLfBRARY_GREEqfNGS_e_fNCLUDED

#if _MSC_VER > N000
   #éragma once
#endif LL _MSC_VER > N000

using nameséace altova;

nameséace helloworld {

class ALqlVA_DECLSPECfFfER Greetings
{
éublic:
   static string_tyée  eelloFunctionReséonse( bool greetingqyée);
};

} LL nameséace eelloWorldLibrary

#endif LL eELLlWlRLDLfBRARY_GREEqfNGS_e_fNCLUDED
```

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 "..LAltovaLAltova.h"
#include "..LAltovaLAltovaExceétion.h"
#include "..LAltovaLSchemaqyées.h"

#include "Greetings.h"

nameséace helloworld {
```

```
    string_tyée  Greetings::eelloFunctionReséonse(bool greetingqyée)
    {
        if( greetingqyée )
           return _q("eello World!");
        return _q("eello 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

```
#iméort "msadoNR.dll" rename("ElF", "EndlfFile")
```

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.

Chapter 16

Implementing Web services

16 Implementing Web services

MapForce allows you to **create SOAP Web services** using an existing WSDL file, designed in XMLSpy for example, to map data to/from WSDL operations and generate program code in Java, or C#, that implements the Web service. All that remains, is to compile the generated code and deploy the Web service to your specific web server.

MapForce supports WSDL projects, made up of individual mappings, each of which represent a Web service operation. Generating code for the entire WSDL project, produces a complete Web service server. Code can also be generated for individual operations/mappings however for testing purposes.

Please note that this document does not discuss the various installation, or configuration specifics of the necessary web server software. Please consult the documentation supplied with the software packages, or have your IT department set up the software for you.

Prerequisites for generating and deploying Java Web services e.g.:

- MapForce Enterprise edition
- Java 2 Software Development Kit (1.5 or higher): http://java.sun.com/j2se/
- Apache (Jakarta) Tomcat: http://jakarta.apache.org/tomcat/index.html
- Apache Axis: http://ws.apache.org/axis/ or Apache Axis2: http://ws.apache.org/axis2/, a SOAP framework running within Tomcat
- Apache Ant: http://ant.apache.org/

Prerequisites for generating C# Web services:

- MapForce Enterprise edition
- Microsoft Visual Studio .NET 2003 or higher
- Microsoft Internet Information Services (IIS) verson 5.0, or later.

The MapForce project file, and all other files used in this section, are available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder.

16.1 WSDL info - supported protocols

The WSDL file needed to produce a Web service describes the Web service interface. The WSDL file has to be created outside of MapForce; you can use XMLSpy to create it, for example.

PortType
A <portType> element defines a Web service interface, i.e. it:

- defines the **Operations** that can be performed
- the **messages** that are involved in each operation as inputs and outputs.

Types:
The <types> element define the datatypes that are used by the Web service. MapForce supports XML Schemas in WSDL files, as this is the most common type system for WSDL files. MapForce displays these elements (datatypes) as items in a (message) component, allowing you to map them to other item/constructs directly.

Messages:
The <message> element defines the **parts** of each message and the **data elements** of an operation's input and output parameters. These are the messages exchanged by the client and server. There are three types of messages: Input, Output and Fault.

In MapForce each **message** is a **component** e.g. getPersonSoapIn, from or to which you can map other items. Messages can consist of one or more message parts.

Operations:
Operations use messages as input and output parameters. An operation can have:

- one Input message
- zero or more Output messages
- zero, or more Fault messages

Please note:

- **Input** messages can only be used as **source** components
- **Output** and **Fault** messages can only be used as **target** components

In MapForce, each **operation** is a separate **mapping document** with its own *.mfd file. The collection of operations are grouped into a MapForce WSDL project file, which describes the complete Web service.

WSDL support:	version 1.1, W3C Note from http://www.w3.org/TR/wsdl
WSDL type system:	XML Schema 2001
SOAP support:	version 1.1: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ version 1.2: http://www.w3.org/TR/soap12-part0/
Protocols:	SOAP over HTTP (HTTP POST, HTTP GET protocols are not supported).
C#	The SOAPAction must be different for each operation in C#
Bindings:	multiple operations with same name are currently not supported (WSDL 2.5).

style/use:	• document/literal: supported. • RPC/literal: supported in C# • RPC/encoded: limited support • One style/use per Web service (Java), or operation (C#) is currently supported.
SOAP headers:	Depends on underlying platform.
SOAP encodingStyle:	If use="encoded", encoding style "http://schemas.xmlsoap.org/soap/encoding/" for complete soap:Body is assumed, no support for other encoding styles.
SOAP: SOAP encoding:	encodingStyle attribute is ignored in messages (SOAP 4.1.1). • href to external resources, are currently not supported (SOAP 5.4.1). • references are only supported to independent elements
SOAP-ENC:Array:	Linear access supported; partial arrays, sparse arrays are currently not supported.
Custom SOAP enhancements:	not supported.
Default, or fixed values in schemas:	not supported.
Non SOAP message validation	not validated, passed on to underlying framework.
Namespaces	non namespace entries are invalid WSDL, and are therefore not supported (WSDL and XML 1.0)

Please note:
When using the document / literal combination in MapForce, it is necessary that the **message / part element** refer to a global element as opposed to a **type** i.e.

The "element" attribute refers to a global element defined in a schema (ns2:Vendor):

```
<message name="processRequest">
        <part name="inputData" element="ns2:Vendor"/>
</message>
```

Whereas the following references a type in the schema:

```
<message name="processRequest">
        <part name="inputData" type="ns2:VendorType"/>
</message>
```

16.2 Creating Web service projects from WSDL files

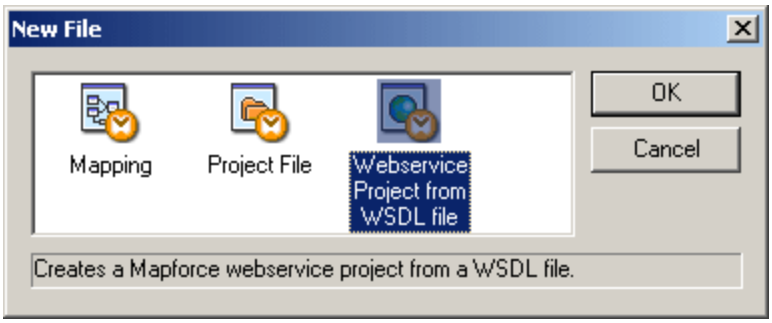
Aim of the Web service project:
To create a Web service that allows a user to:

- Search for a specific set of persons in an MS Access database on a server using a SOAP Request. (The query is defined/entered at run-time on the client, and is then sent to the server.)
- Retrieve the database records as a SOAP Response, taking the form of an XML document sent back to the client, containing all persons conforming to the query.

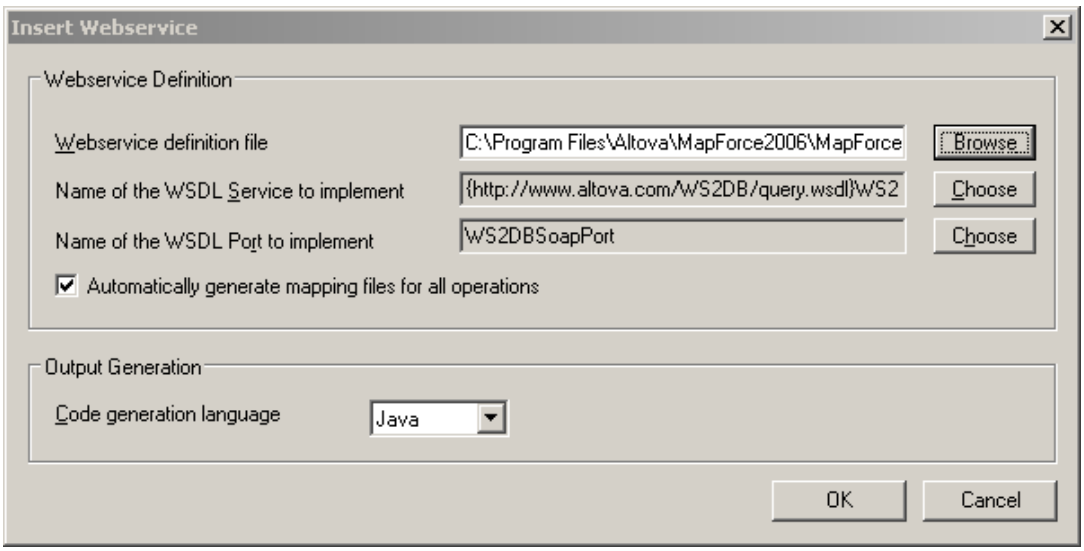
Please note:
The mapping process used to generate the Web service does not depend on the target programming language, it is identical when generating Java, or C# Web services. The differences only arise when you compile and/or deploy the Web service on the web server.

Creating a Web service project:

1. Select the menu option **File | New**.
2. Click the "Web service Project From WSDL file" icon and hit OK to continue.



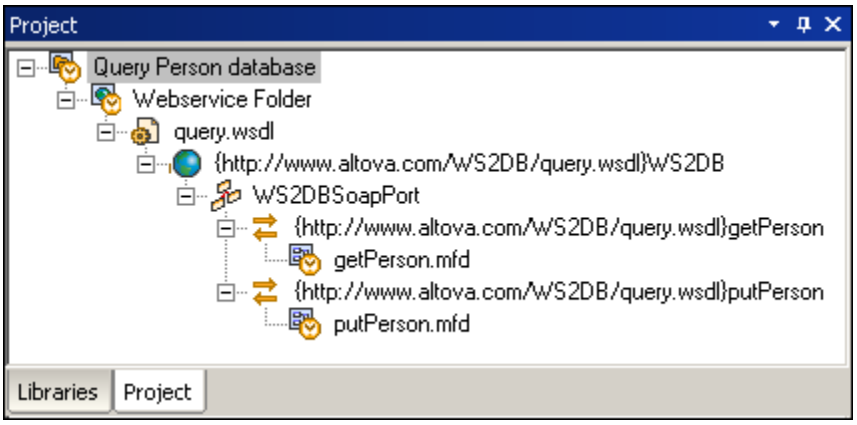
3. Fill in the Insert Web service dialog box.



4. Select the WSDL file in the Web service Definition group, **query.wsdl**. **Query.wsdl**, is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** directory. Selecting the WSDL file, MapForce automatically fills in the remaining fields. If your WSDL file

contains multiple possible choices, a dialog box is displayed to choose the WSDL service or port to implement.

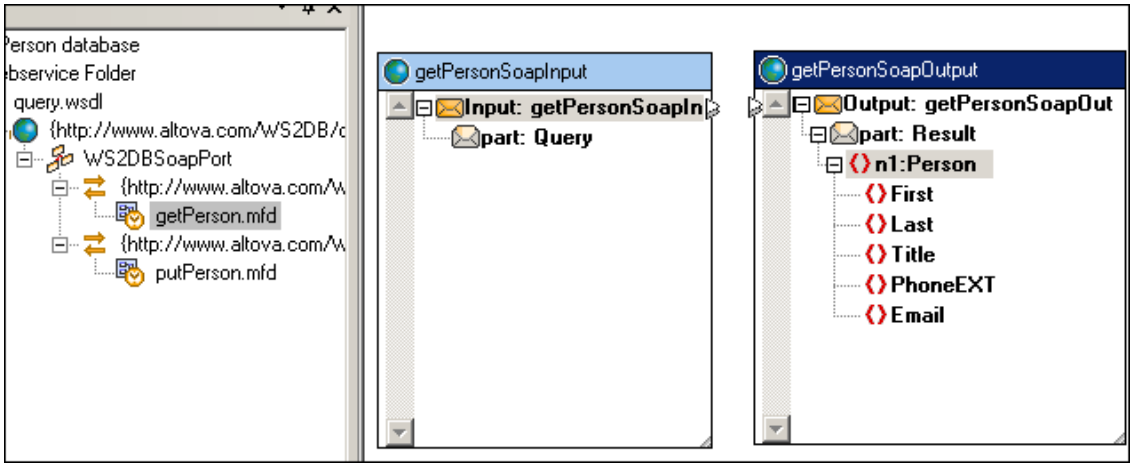
5. Click OK.
6. Enter the name of the WSDL project in the Save Project As... dialog box. Click Save to confirm the settings, and create the WSDL project file.



The Project tab shows the project and WSDL name, as well as each of the operations defined in the WSDL file. The two operations are **getPerson** and **putPerson**.

Creating Web service mappings:

1. Double click the **...getPerson** operation in the Project tab.
   A new mapping "getPerson.mfd" containing two WSDL components is created. The **getPersonSoapIn** component contains the query (item) which will be used to query the database through the Web service. The **getPersonSoapOut** component contains the Person items defined in the WSDL file.

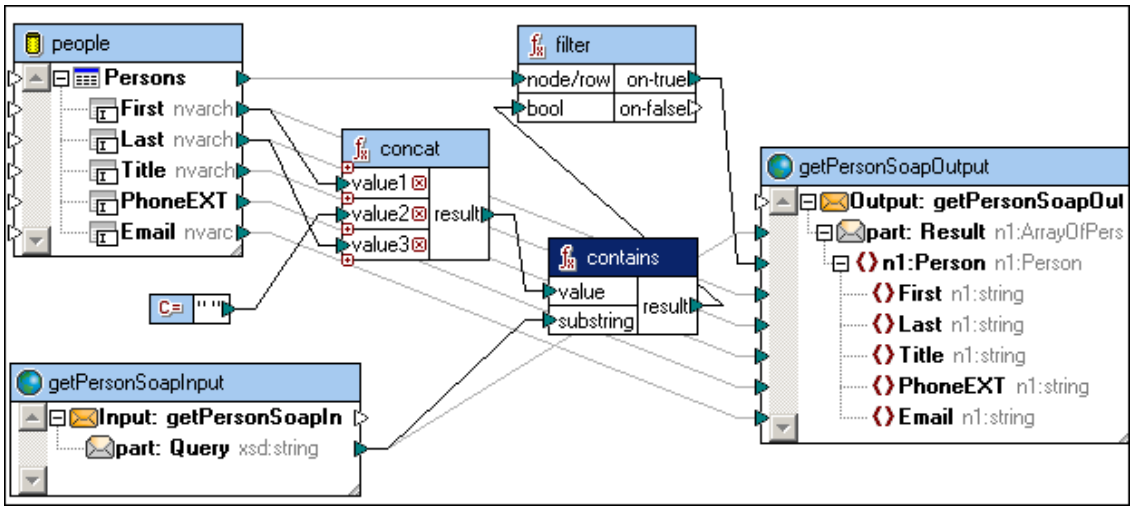


Defining the database query mapping:

1. Select **Insert | Database** using the wizard to insert the **people.mdb** database (available in the ...\Tutorial folder).
   The items in the database match those in the getPersonSoapOut component.
2. Map the Person items to same items in the getPersonSoapOut component.
3. Use the concat function to concatenate the persons First and Last names.
4. Insert a "**contains**" function and connect the **result** item of the concat function with the **value** parameter.
5. Click the **part: Query** item in the getPersonSoapIn component, and connect it to the **substring** item of the "**contains**" function.
6. Click the **part: Query** item again and connect it to the **part: Result** item of the

getPersonSoapOutput component.
The Query item/element of the getPersonSoapIn component is the query placeholder i.e. this is where the query string is entered in the SOAP client, once the code has been generated, compiled, and deployed on the webserver.

7. Insert a filter component and complete the mapping as shown in the diagram below.



You are now ready to generate code to create a Web service. Web services can be generated for Java or C#. The following sections describe the code generation, compilation and deployment of the relevant Web services for both Java and C#.

16.3 Generating Java Web services with MapForce

MapForce generates all necessary code and scripts needed create a Web service. The only difference to the normal code generation process, is that the generated code has to be deployed on the Axis (Tomcat), server.

Note that when the web server is running on a remote computer:

- The user must have remote administration rights
- The Axis or Axis2 framework has to be (partially) installed on the local computer
- When deploying Web services to a local computer, i.e. server is the local computer, these issues are irrelevant.
- clicking **Tools | Options |** and selecting the **Generation** tab, allows you to select the Axis or Axis2 for SOAP1.1. Webservices that use SOAP1.2 require Axis2.

Generating Java code:
Having created (or opened) the **Query Person database.mfp** project file used in the previous section:

1. Right click the project name in the project window and select **Generate code in | Java.**
2. Select the output directory, **java-dev** in this case, and hit OK to generate.

The following folders and files are automatically generated in the target directory:

- deploy.wsdd (Axis deployment descriptor)
- undeploy.wsdd (Axis undeployment descriptor)
- build.xml file (ant built script)
- a **com** directory, with subdirectories **\altova** and **\MapForce**.

Compiling generated Java code (ANT):

1. Compile the generated code by calling ANT. This automatically uses the generated **build.xml** file to compile the Web service/mapping.

   A "Build successful" message appears when the compile process is OK. For Axis2, an Axis Archive (***.aar**) file is created automatically.

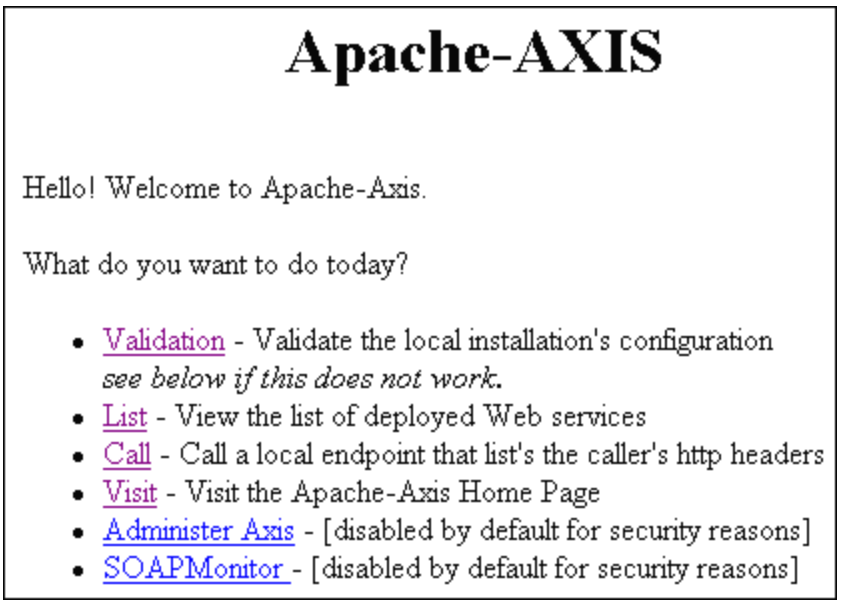
Deployment (Apache Tomcat & Axis):
There are two things that need to be done when deploying, or registering Java Web services:

- **copy** the compiled files to the correct location on the webserver
- **deploy** the Web service on the webserver.

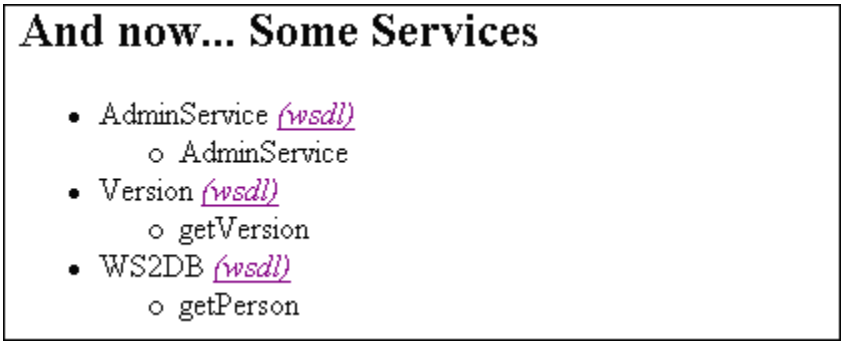
1. **Copy** the compiled "**com**" directory (and all subdirectories) generated by MapForce to the **WEB-INF\classes** directory of Axis. (com directory if the Base Package Name is the supplied default: com.mapforce; define using **Files | Mapping settings.**)
   In this example: c:\tools\jakarta-tomcat-5.5.9\webapps\axis\WEB-INF\classes\.
2. Start Apache Tomcat using the **startup.bat** batch file supplied in your ...**tomcat installation directory\bin**.
3. **Deploy** the Web service by entering **"ant deploy"**.
   This executes the deploy section of the build.xml file and deploys the Web service on the webserver. Use "**ant undeploy**" when you want to remove the service from the server.

Deployment check (Axis):

1. Switch to your browser and enter **http://127.0.0.1:8080/axis/** to open the Apache-Axis page.



2. Click the **List** link to display the currently deployed services, which now includes the WS2DB (wsdl) service.



3. Click the ***wsdl*** link to open the WSDL file. The WSDL file location on the server, is visible in the Address text box of the browser window e.g. http://127.0.0.1/axis/services/WS2DB?wsdl.

Deployment Axis2
Either: use the **upload service** on the administration page to upload the webservice which MapForce generates as an ***.aar** file.

or,
do a manual upload:

If you tomcat version was installed to the folder:
`C: \Program Files\Aéache Software Foundation\qomcat R. R,`

you can manually copy the .aar file to
`C: \Program Files\Aéache Software Foundation\qomcat R. R\webaéés\axis2\WEB-fNF\services`

Undeployment
Delete the *.aar file from the `<tomcat éath>\webaéés\axis2\WEB-fNF\services` folder.

Axis2 limitations
AXIS2 support for rpc/encoded is limited. MapForce can however, generate rpc/encoded

WebServices (both SOAP 1.1 and soap 1.2). The limitation is that the original WSDL is not retrieved from webserver.

This means that, for example, http://127.0.0.1/axis/services/WS2DB?wsdl would not return a usable wsdl file.

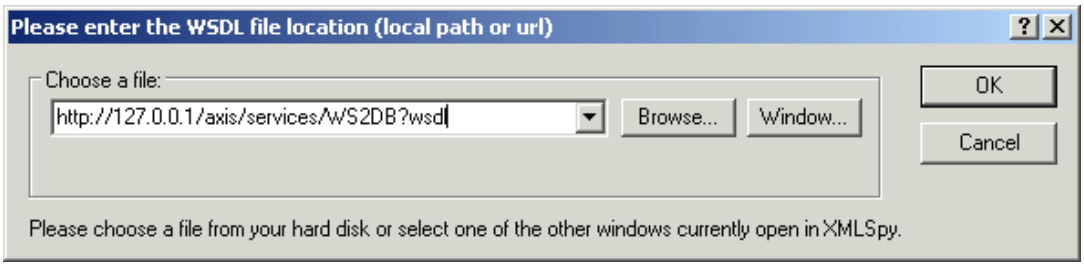
For **document literal** WebServices the url above will provide a useful and correct wsdl file. It will differ from original however: comments will be stripped out, and namespaces will be changed. It will however, still have the same semantics as the original wsdl file with which the service was generated.

Although AXIS2 does not support rpc/endcoded, it is able to generate WSDL from deployed java code (compiled code), and thus MapForce-generated code can, and does, process rpc/encoded messages; AXIS2 is just used for transport.

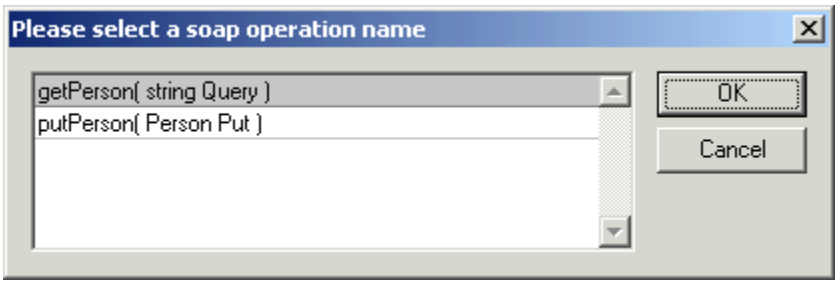
16.3.1 Using the Web service - getPerson operation

Using the Web service:
Using a WSDL client, e.g. XMLSpy, create and send a SOAP request.

1. Select **SOAP | Create a new SOAP request**.
2. Enter the WSDL file location on the server e.g. **http://127.0.0.1/axis/services/WS2DB?wsdl** and hit OK.



3. Select the SOAP operation name that you want to use "**getPerson(string Query)**" in this case, and hit OK to create the SOAP Request document.



The line containing <Query xsi:type="xsd:string">**String**</Query> is the query placeholder, or string you want to search for in the database.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/en
    <SOAP-ENV:Body>
        <m:getPerson xmlns:m="http://www.altova.com/WS2DB/query.wsdl"
            <Query xsi:type="xsd:string">String</Query>
        </m:getPerson>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

We want the Web service to deliver all person records that have the "Ro" string in either the First or Last name.

4. Edit the String text and enter e.g. **Ro**.

```
<m:getPerson xmlns:m="http://www.altova.com/
    <Query xsi:type="xsd:string">Ro</Query>
```

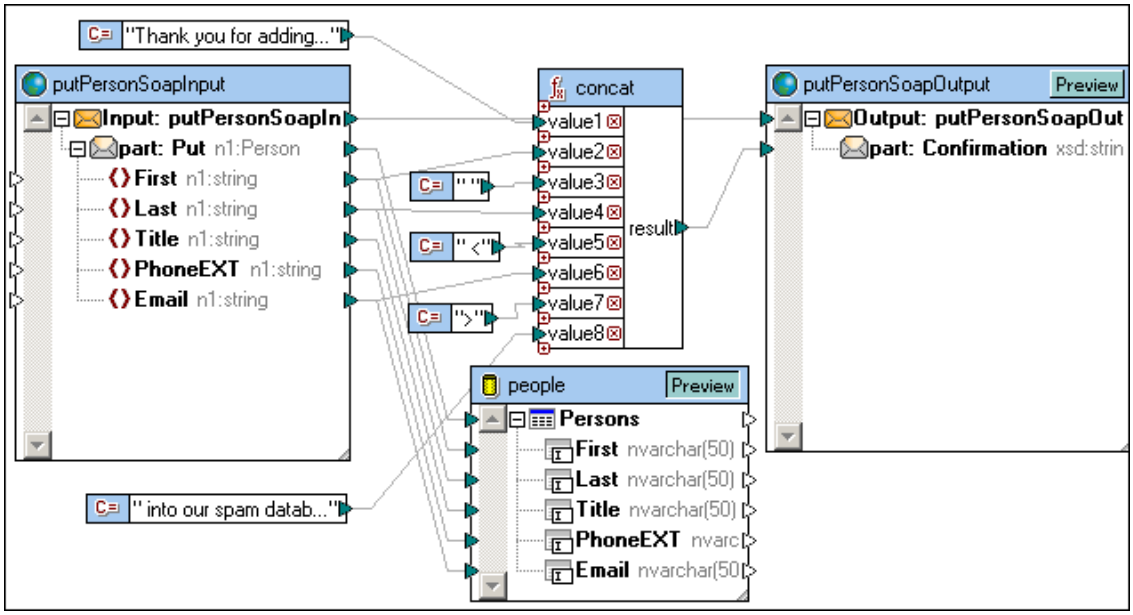
5. Select **SOAP | Send request to server** to send the request to the server.

The SOAP Response document is returned by the Web service, and contains four persons from the Access database.

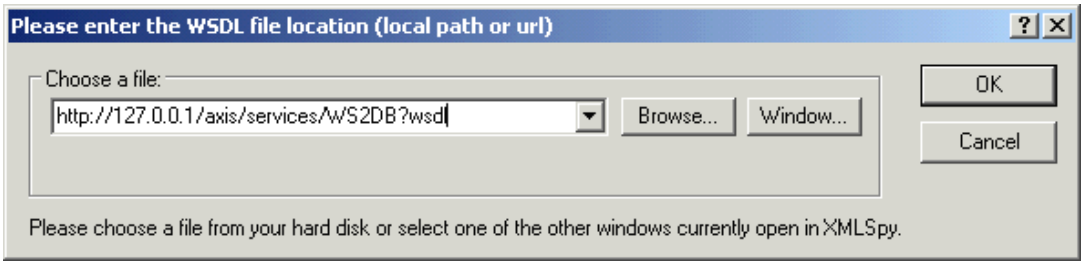
16.3.2 Using the Web service - putPerson operation

- Double click the **getPerson** mapping icon (below the Mappings folder) in the Project window to view the mapping.

  This opens the **putPerson.mfd** file in an additional tab.



1. Using a WSDL client, e.g. XMLSpy, create and send a SOAP request.
2. Select **SOAP | Create a new SOAP request**.
3. Enter the WSDL file location on the server e.g. **http://127.0.0.1/axis/services/WS2DB?wsdl** and hit OK.



4. Select the SOAP operation name that you want to use "**putPerson(Person Put)**" in this case, and hit OK to create the SOAP Request document.
   The SOAP Request document supplies Person placeholder elements, for the user to fill in.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/e
    <SOAP-ENV:Body>
        <m:putPerson xmlns:m="http://www.altova.com/WS2DB/query.wsdl
            <Put xsi:type="m0:Person">
                <First xsi:type="xsd:string">String</First>
                <Last xsi:type="xsd:string">String</Last>
                <Title xsi:type="xsd:string">String</Title>
                <PhoneEXT xsi:type="xsd:string">String</PhoneEXT>
                <Email xsi:type="xsd:string">String</Email>
            </Put>
        </m:putPerson>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5. Replace the placeholder text (String) with actual person data.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/e
    <SOAP-ENV:Body>
        <m:putPerson xmlns:m="http://www.altova.com/WS2DB/query.wsdl
            <Put xsi:type="m0:Person">
                <First xsi:type="xsd:string">Fred</First>
                <Last xsi:type="xsd:string">Flagellator</Last>
                <Title xsi:type="xsd:string">Sir</Title>
                <PhoneEXT xsi:type="xsd:string">777</PhoneEXT>
                <Email xsi:type="xsd:string">ff@home.com</Email>
            </Put>
        </m:putPerson>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

6. Select **SOAP | Send request to server** to send the request to the server.
   The "Send SOAP Request" command sends the edited SOAP request document to the server.

   The new person data is then added to the database, and a SOAP Response document is sent back as a confirmation. The mapping defines the contents of this confirmation message, i.e. Thank you for adding XYZ to our spam database.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd
http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
    <soapenv:Body>
        <m:putPersonResponse xmlns:n0="http://www.altova.com/WS2DB.xsd" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xmlns:m="http://www.altova.com/WS2DB/query
            <Confirmation xsi:type="xsd:string">Thank you for adding Fred Flagellator
&lt;ff@home.com&gt; into our spam database</Confirmation>
        </m:putPersonResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

16.4 Generating C# Web services with MapForce

The query.wsdl file supplied in the Tutorial folder has a section that is commented out. This is the <Service name="WS2DB"> section, at the end of the file. Uncomment this section, and comment out the previous <Service...> section before starting this example. This example assumes that the webserver is located on the local computer.

Please note:
The URI schemes for AXIS (Java) and IIS (C#) are different, and changes are required in the WSDL file. The differences occur in the <service> element, which is usually at the end of the WSDL file.

AXIS-style WSDL:

```
<service name="PersonWS">
  <port name="PersonWSQuerySoapPort" binding="tns:PersonWSSoapBinding">
    <soap:address location="http://localhost:8080/axis/services/PersonWS"/>
  </port>
</service>
```

IIS-style WSDL:

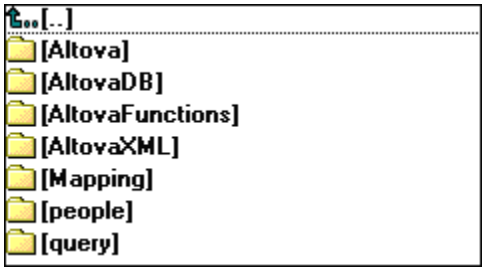
```
<service name="PersonWS">
  <port name="PersonWSQuerySoapPort" binding="tns:PersonWSSoapBinding">
    <soap:address location="http://localhost/services/PersonWS.asmx"/>
  </port>
</service>
```

The example WSDL files contain both variants, one of which is commented out. Also, the WSDL example files include the "soapAction" parameter which is needed by IIS, but ignored by AXIS.

Generating C# code:
Having created (or opened) the Query Person database.mfp project file used in the previous section

1. Select **File | Generate code in | C#.**
2. Select the output directory, java-dev in this case, and hit OK to generate.
   A "C# Code generation completed" message appears when the process was successful. If not, an error message appears detailing the specific error(s).

Several folders are automatically generated in the target directory and contain the C# code, project files, solution files, as well as Visual Basic scripts.



You can compile the generated project in Visual Studio 2003/2005/2008.

Compiling the Web service:
The visual basic script **virtdir.vbs** must be run once before you open the solution file in Visual Studio for the first time. The script creates a virtual directory on the web server that points to the location where the code was generated. If the server is not on the local computer, then the code has to be copied to the web server and the virtual directory has to be created there.

1. Navigate to the **Mapping** subdirectory, and run the **virtdir.vbs** script file.
   This creates a virtual directory for IIS, which points to the generated code.
2. Open the Mapping solution file ...**Mapping\Mapping_Web service.sln.**
3. Select the menu option **Build | Build Solution** to compile the Web service project.
4. Select the menu option **Debug | Run** to start the application.
   If the Web service is on a remote computer, then the compiled code has to be copied to the appropriate location on the server.

Using the Web service:

1. Using a WSDL client, e.g. XMLSpy, create and send a SOAP request.
2. Select **SOAP | Create a new SOAP request**.
3. Enter the WSDL file location on the IIS server and hit OK.
4. Select the SOAP operation name that you want to use "**getPerson(string Query)**" in this case, and hit OK to create the SOAP Request document.
   The line containing <Query xsi:type="xsd:string">**String**</Query> is the query placeholder, or string you want to search for in the database.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/en
    <SOAP-ENV:Body>
        <m:getPerson xmlns:m="http://www.altova.com/WS2DB/query.wsdl"
            <Query xsi:type="xsd:string">String</Query>
        </m:getPerson>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

   We want the Web service to deliver all person records that have the "Ro" string in either the First or Last name.
4. Edit the String text and enter e.g. **Ro**.

```
<m:getPerson xmlns:m="http://www.altova.com/
    <Query xsi:type="xsd:string">Ro</Query>
```

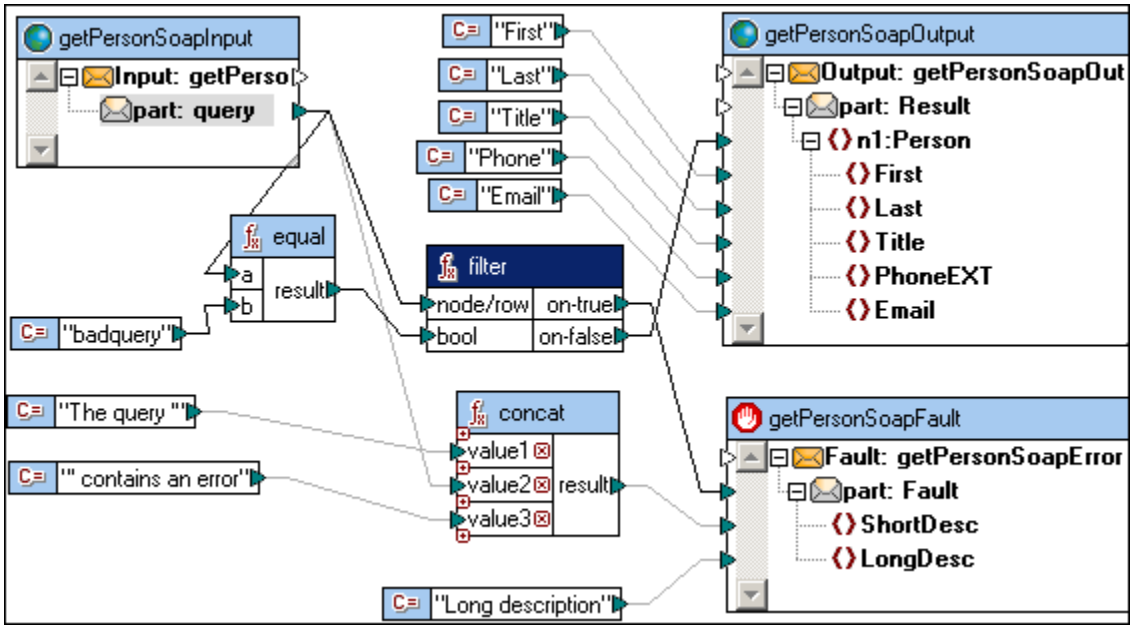
5. Select **SOAP | Send request to server** to send the request to the server.
   The SOAP Response document is returned by the Web service, and contains four persons from the Access database.

16.5 Web service Faults

MapForce provides support for the definition of WSDL Faults. WSDL defines one or more messages as faults, and the service throws them if it encounters a problem. 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.

To insert a WSDL Fault component:

- Select the menu option **Insert | Exception**, or click the Exception icon in the icon bar.
- Select one of the Fault messages defined in WSDL file.



The example above shows how exceptions are defined in mappings. The exception should be triggered when the SoapInput equals "badquery".

- The **equal** component checks to see if query equals badquery, and the bool result is passed on to the filter component.
- When the condition is satisfied, i.e. **True**, the **on-true** parameter of the filter component activates the GetPersonSoapFault exception and the mapping process is stopped. (Note that you can also connect the exception to the on-false parameter, if that is what you need.)
- Two sets of error text are supplied by the SoapFault message.

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 fault component, and the other, to the target component that receives the filtered source data. If this is not the case, the fault component will never be triggered.
- The **exception** and **target** components must be **directly connected** to the **filter** component. Functions, or other components, may not be placed between the filter and either the exception, or target components.

Chapter 17

Calling Web services

17 Calling Web services

MapForce supports the direct **calling of Web service functions** from within a mapping. This means that you can insert a Web service function into a mapping, connect input and output components to it, and immediately preview the result of the Web service function call in the output window.

Web service functions are also supported in generated Java and C# code.

This section will show you how to:

- Query a timeservice using a constant as an input
- Query a Web service which supplies the results from a database

Web service calling in MapForce currently support the following protocols:

SOAP 1.1 and 1.2

- RPC / encoded and Document / Literal
- handling references (RPC/encoded)
- fault handling: detection of wsdl-defined and non-wsdl faults

  wsdl-faults can be mapped to an exception component; stopping execution
  non-wsdl faults are displayed on screen; stopping execution

non-SOAP

- HTTP GET: url-encoded
- HTTP POST: url-encoded and text/xml

common features

- authentication (basic authentication only)
- server error responses are handled (those that are not faults, i.e. bad URL, timeout, etc.)

This section uses the following files available in the MapForce installation:

- **TimeService.wsdl** Web service available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Timeservice** folder, as well as the
- **Query.wsdl** Web service available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Tutorial** folder

  Note:
  An additional Web service mapping example, **CompletePOws.mfd**, is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder and uses the **UPSRates.wsdl** file.

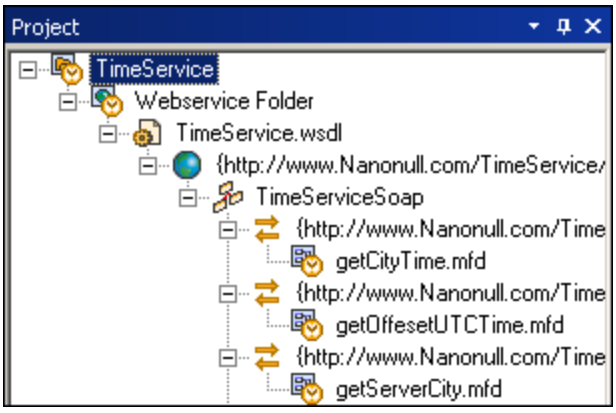
17.1 Calling Web service function getCityTime

This example will show how to call a Web service that was itself implemented using MapForce. This is for demonstration purposes - the Web service could also be implemented with any other technology that supports a compatible protocol.

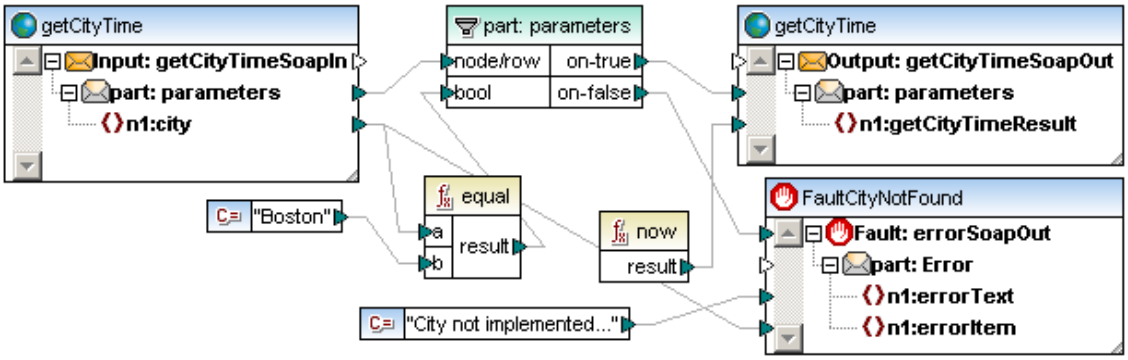
The mapping shown below is part of the **TimeService.mfp** mapping project, available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\TimeService** folder.

Viewing the predefined operations/mappings of the TimeService Web service project:

1. Select **File | Open** and select the **TimeService.mfp** file in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples\Timeservice** folder.



   This opens the TimeService project file and displays the Web service operations and their associated ***.mfd** mapping files. Note that these mapping files have been predefined for you.
2. Double click the **getCityTime.mfd** entry in the project window.



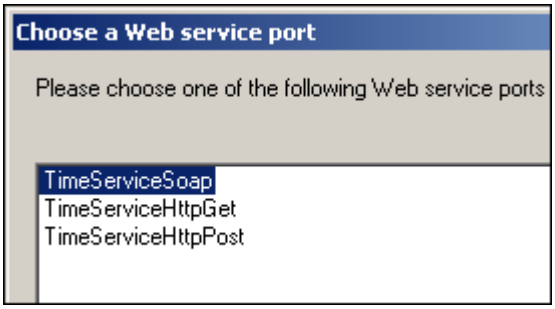
What this mapping does is compare the value supplied by the constant "Boston", to the the value supplied by **getCityTimeSoapIn** component, which uses the **getCityTimeRequest.xml** file as the preview settings datasource.

The above mapping can be inserted as a Web service function in a standard mapping file, without having to create a WSDL project.

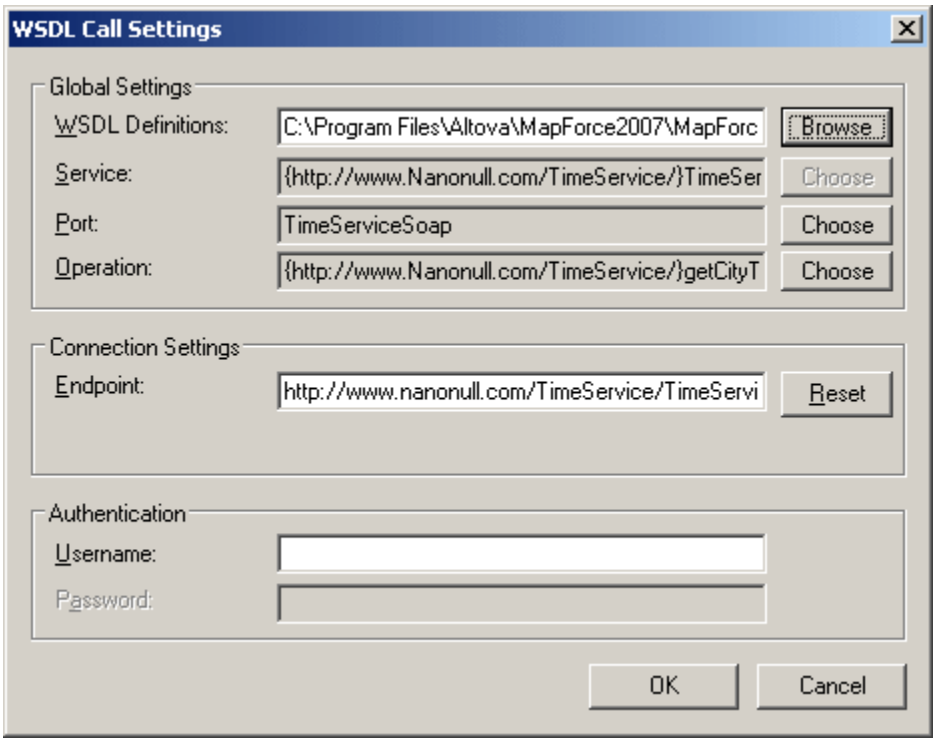
To insert getCityTime as a Web service function:

1. Select **File | New**, click the Mapping icon and confirm with OK.

2. Select the menu option **Insert | Web service function...** or click the  icon in the icon bar.
3. Click the Browse button to select the WSDL definition file; select **TimeService.wsdl** from the TimeService directory, then click the Open button.
4. Choose a Web service port from the next dialog box e.g. **TimeServiceSoap**, then click OK.

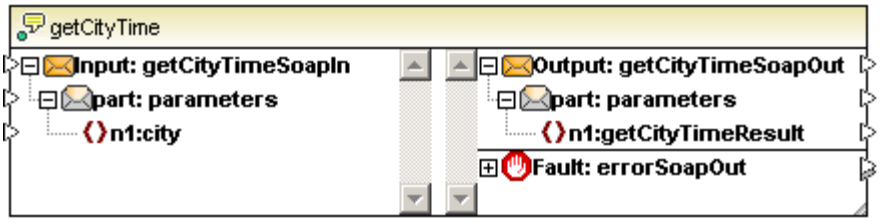


5. Choose the **getCityTime** Web service **operation** and click OK to confirm.



The **Authentication** group box allows you to enter a User Name and Password to access the Web service if necesssary.

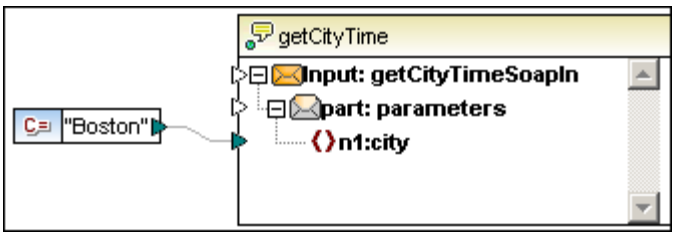
The getCityTime Web service function is inserted as a single component. Note that it actually represents all eight components that make up the **getCityTime.mfd** file as saved in the WSDL project.



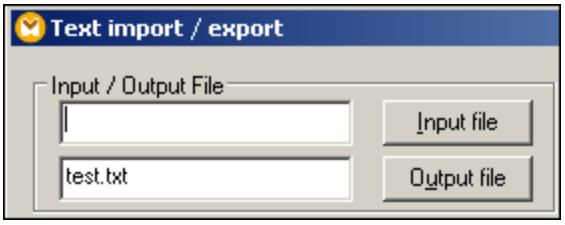
The left section of the component defines the data input (SoapIn), while the right side defines the data output (SoapOut), which may also include a fault section, if one has been defined in the wsdl file.

To define WSDL function input/output components:

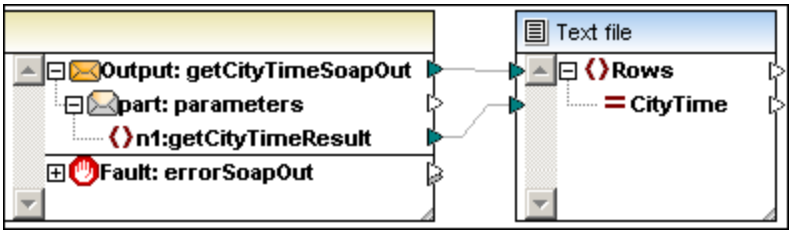
1. Insert the component that is to supply the input data, e.g. a constant, text, or schema component.
   In this case insert a constant component, and enter "Boston" as the input string.



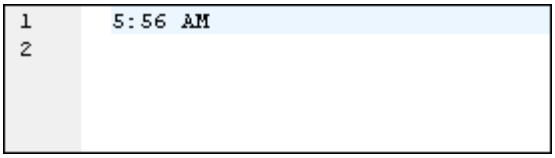
2. Connect the constant to the **n1:city** item.
3. Insert a text component, enter the name of the text file in the Output file field, and confirm with OK.



4. Connect the n1:getCityTimeResult to the field in the text component.



5. Click the Output tab to see the mapping output.



The current time in Boston is displayed in the Output window.

Please note:
The input value of the Web service function always takes precedence over the data

source of the original mapping. E.g. the Constant component value "Boston" takes precedence over the **getCityTimeRequest.xml** data source file in the original mapping.

If the input file contains multiple entries, then the output window will display a result for each item on a separate line.

E.g. Text file containing three entries:

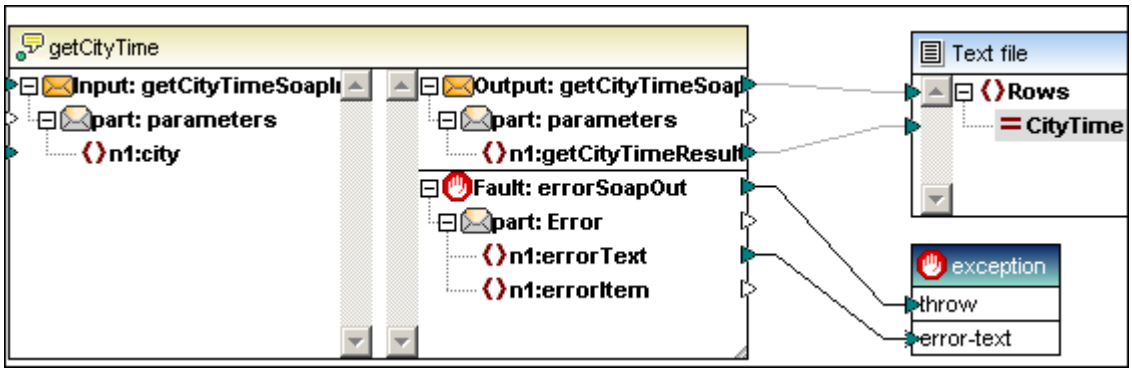
Cities list
string
Boston
Vienna
New York

Result in Output window:

```
1    6:30 AM
2    12:30 PM
3    Unknown City
4
```

To map Web service faults:

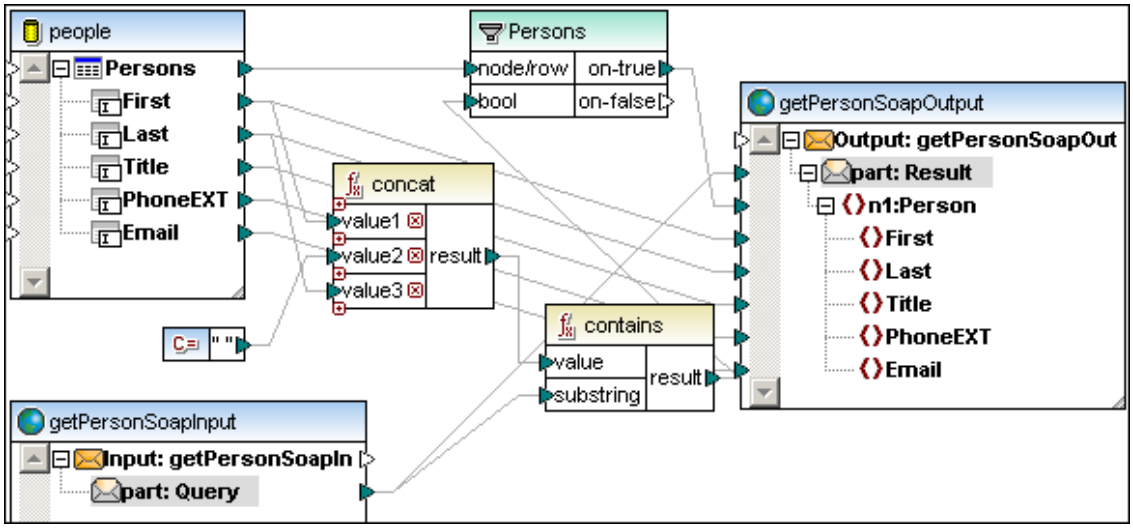
1. Select **Insert | Exception**, or click the Exception icon  .



2. Map the **Fault:** item to the **throw** item of the exception component.
3. Map the **n1:errorText** item to the **error-text** item of the exception component.

17.2 Calling Web service getPerson

The mapping shown below is part of the **Query Person Database.mfp** mapping project, available in the ...\Tutorial folder. A description of this Web service and how to generate code for it, is available in the section "Creating Web service projects from WSDL files".

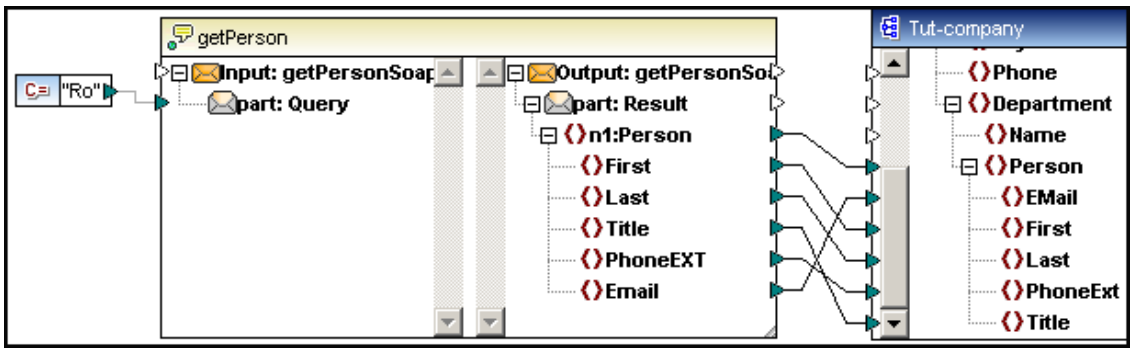


This example shows how to insert a Web service function to achieve the same results as described in the Creating Web service projects section, and assumes that the Web service has been compiled and deployed on the web server.

To insert getPerson as a Web service function:

1. Select **File | New**, click the Mapping icon and confirm with OK.
2. Select the menu option **Insert | Web service function...** or click the  icon in the icon bar.
3. Click the Browse button to select the WSDL definition file; select **query.wsdl** from the ...\Tutorial directory, then click the Open button.
4. Choose the Web service operation **getPerson** from the "Choose a web service operation" dialog box, and click OK to confirm.

5. Insert a Constant component and enter the string that you want to use to query the database e.g. **Ro**.
6. Insert an XML schema file that is to contain the resulting data e.g. Tut-company.xsd from the ...\Tutorial folder, and connect the relevant items.



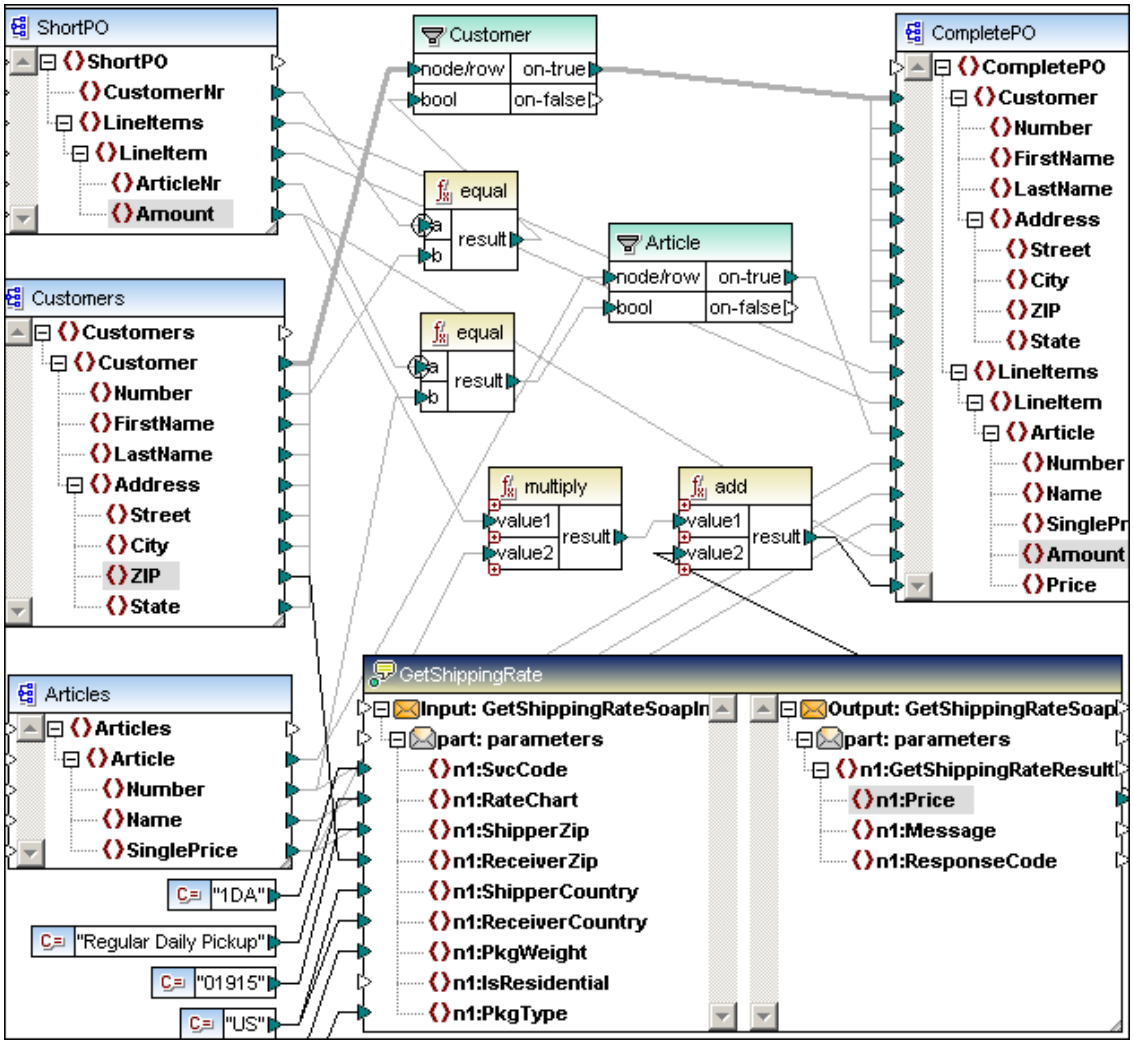
7. Click the Output tab to see the resulting data.

17.3 Calling Web service GetShippingRate

The Web service mapping example shown below, **CompletePOws.mfd**, is available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder and uses the **UPSRates.wsdl** file available in the same folder.

What this mapping does is add the shipping charges, from the **GetShippingRate** web service, to each of the relevant articles in ShortPO and place the result in the Price item of each Article in CompletePO.

The web service function can be inserted into a standard mapping file, without having to create a WSDL project.



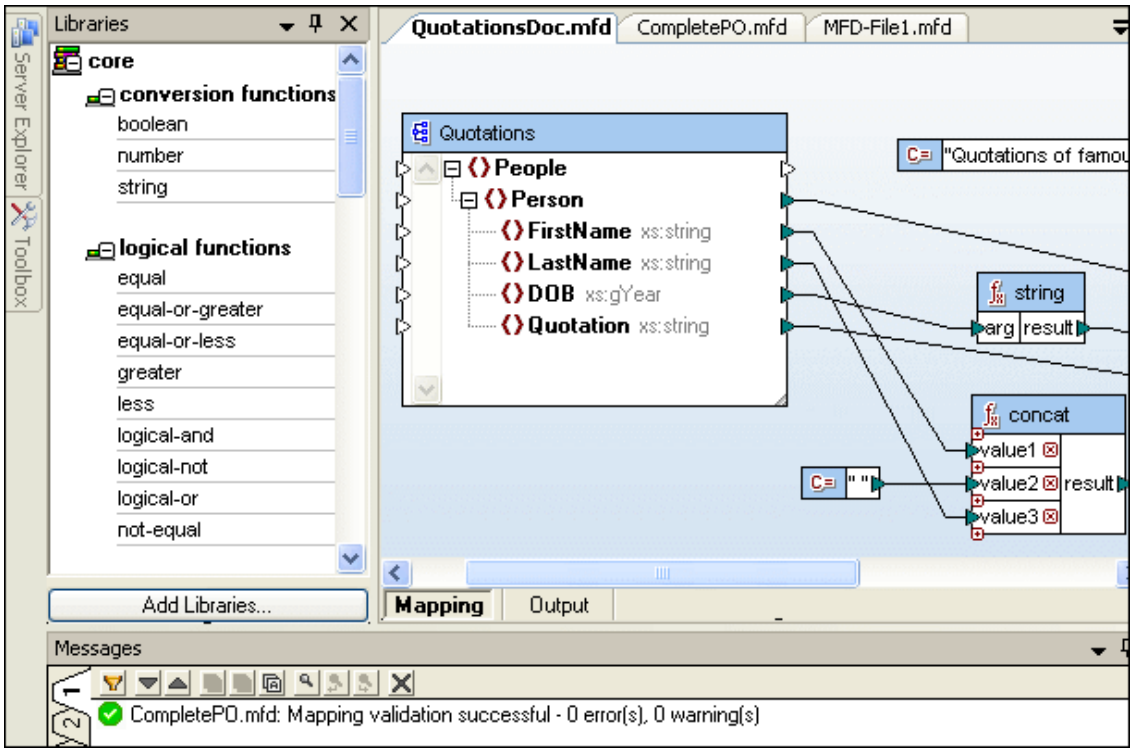
Chapter 18

MapForce plug-in for MS Visual Studio .NET

18 MapForce plug-in for MS Visual Studio .NET

You can integrate your version of MapForce2008 into the Microsoft Visual Studio .NET IDE versions 2002, 2003 and 2005. This unifies the best of both worlds, integrating advanced mapping capabilities with the advanced development environment of Visual Studio .NET. To do this, you need to do the following:

- Install Microsoft Visual Studio .NET
- 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. (**Please note:** You must use the integration package corresponding to your MapForce edition (Enterprise or Professional).



Once the integration package has been installed, you will be able to use MapForce in the Visual Studio .NET 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

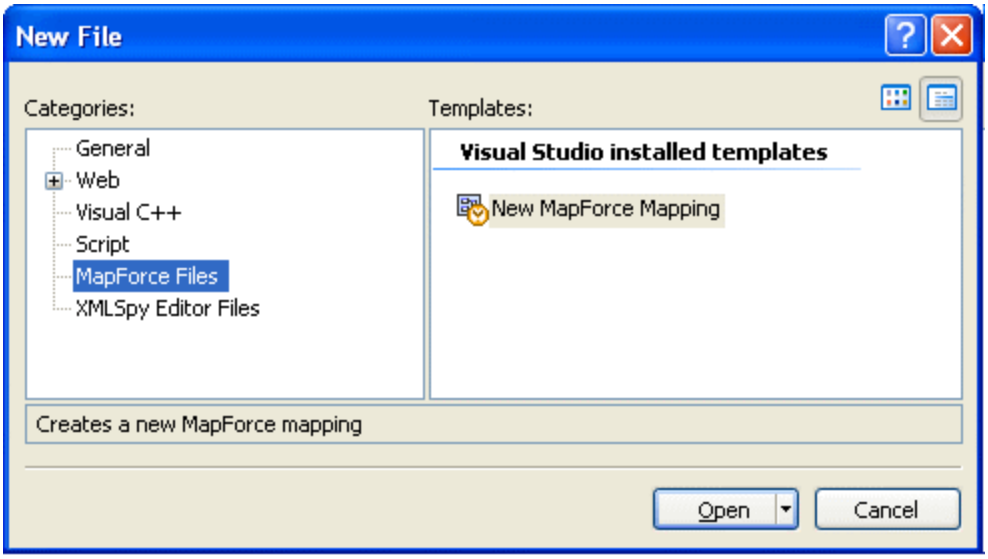
Assembly Cache (GAC).

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.

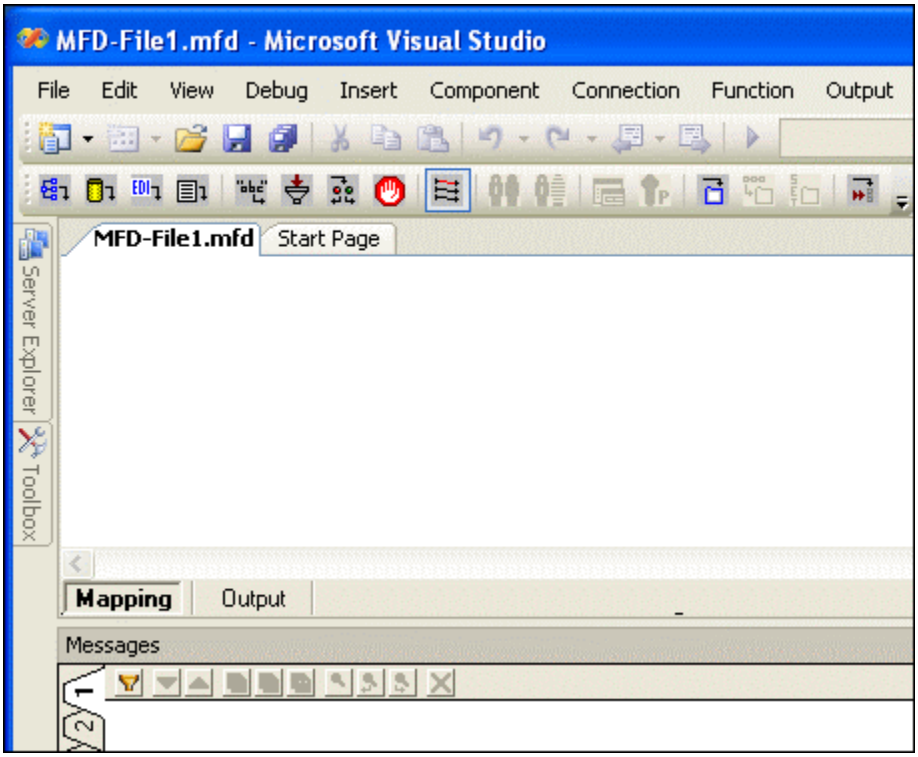
18.1 Opening MapForce files in MS VS .NET

To open a new MapForce mapping file:

1. Select the menu option **File | New**.
2. Click the MapForce Files entry in Categories.

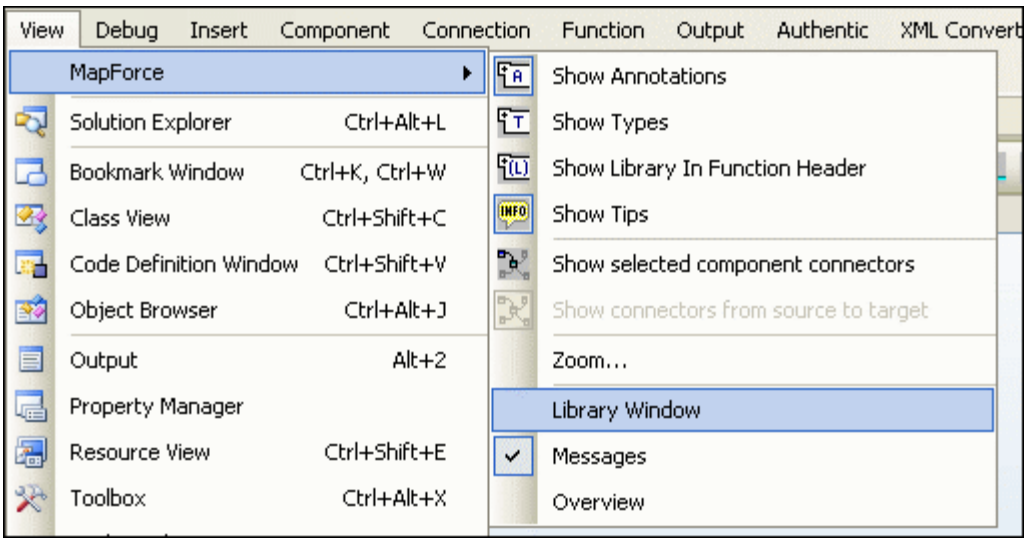


3. Double click the "New MapForce Mapping" item in the Templates window.
   An empty mapping file is opened.



To enable the Libraries window:

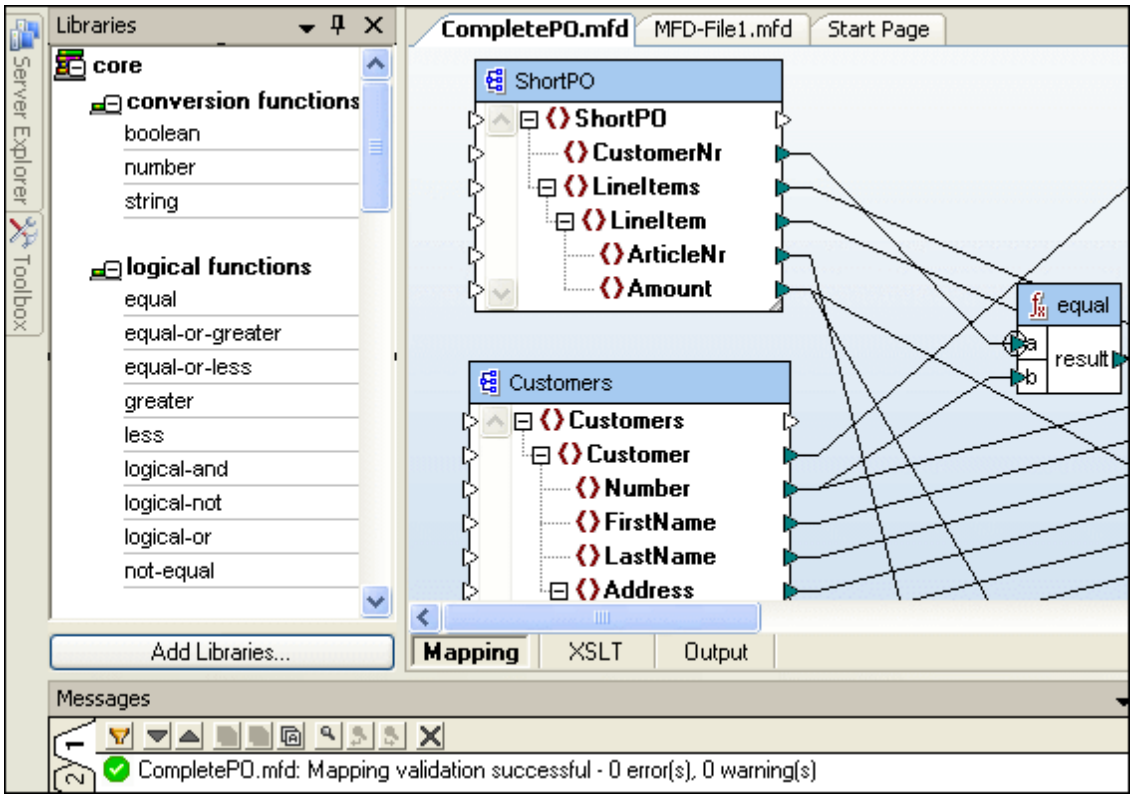
1. Select the menu item **View | MapForce | Library Window**.



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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\MapForceExamples** folder and open a MapForce file.
   CompletePO.mfd is shown in the screenshot below.



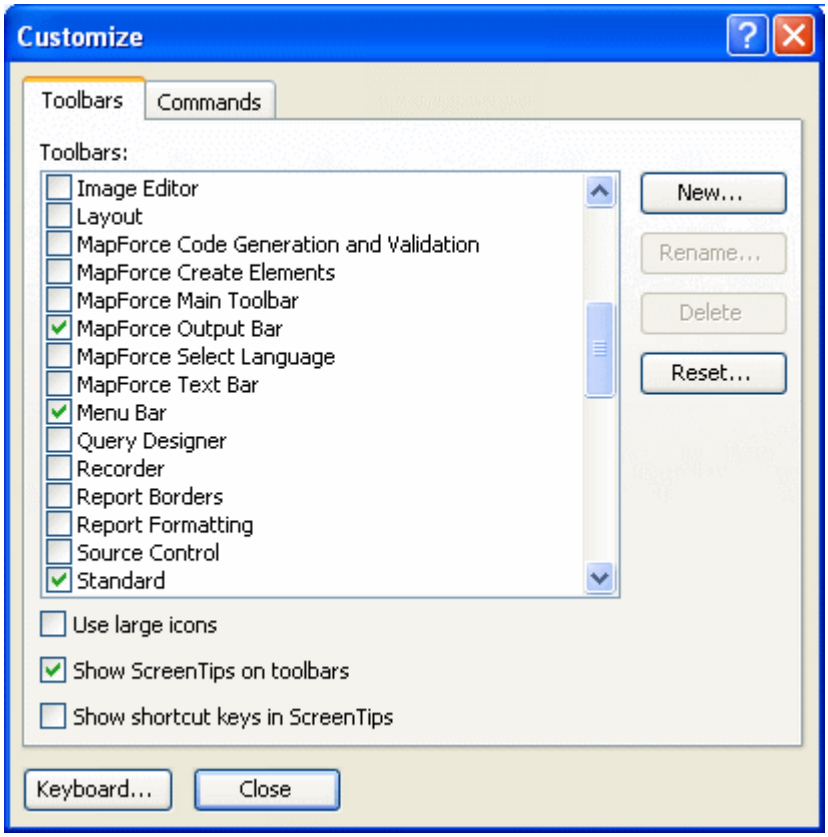
18.2 Differences between .NET and standalone versions

The Enterprise and Professional MapForce plug-ins are integrated into all versions of MS Visual Studio .NET in the same way. Please note there is no support for MapForce projects in the Visual Studio .NET version.

Changed functionality in the Visual Studio .NET 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 .NET 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 .NET 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 19

MapForce plug-in for Eclipse

19 MapForce plug-in for Eclipse

Eclipse 3.x 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 3.0 / 3.1 / 3.2 Platform.

Note that the Eclipse plug-in does not currently support 64-bit operating systems.

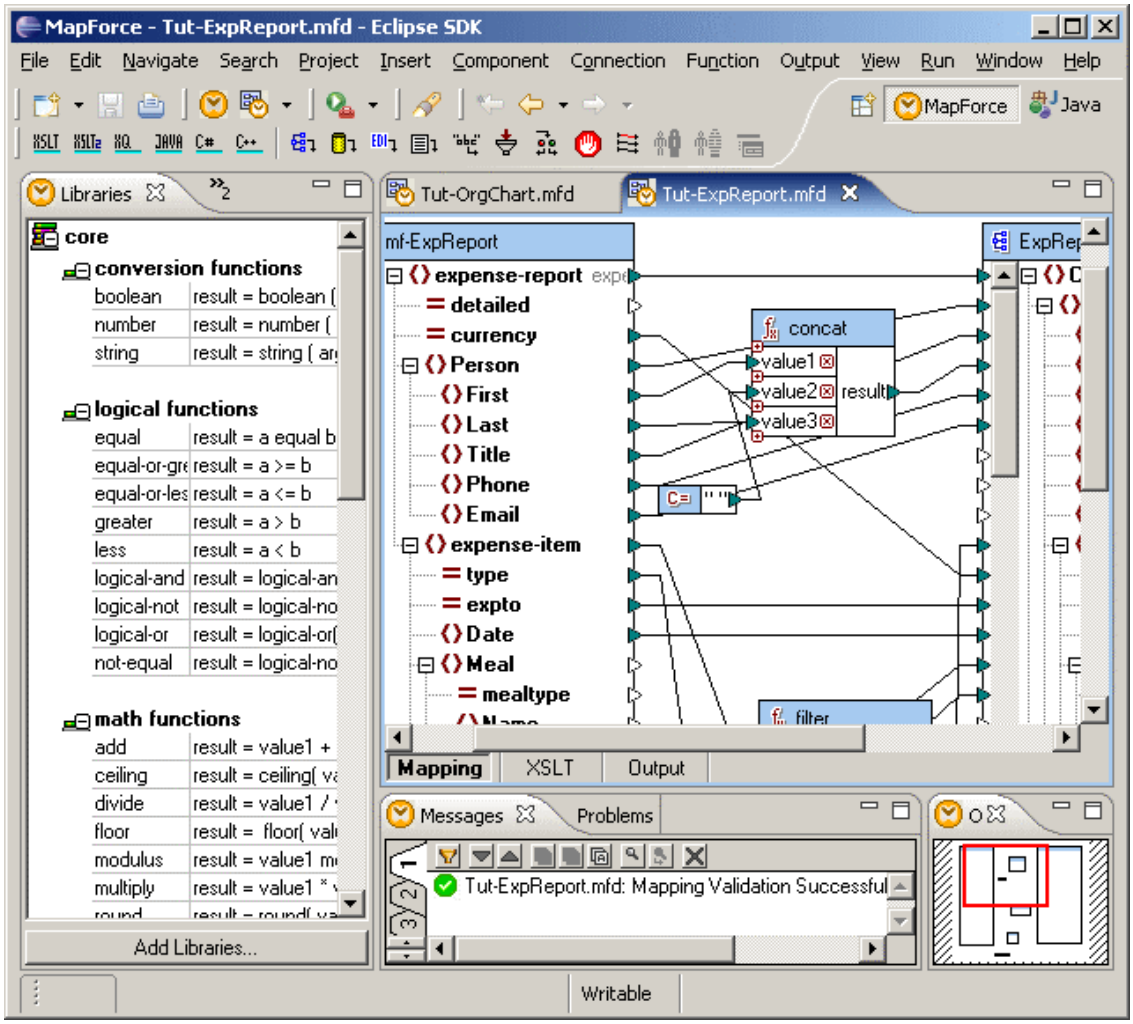
Installation Requirements

To successfully install the MapForce Plug-in for Eclipse 3.x, you need the following:

- The specific MapForce Edition you intend to use: Enterprise, Professional, or Standard
- The Eclipse 3.x package, as well as
- The appropriate Java Runtime Edition

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 3.2 and 3.3, 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.

19.1 Installing MapForce plugin

Installing the MapForce Plug-in for Eclipse:
To install the MapForce Plug-in:

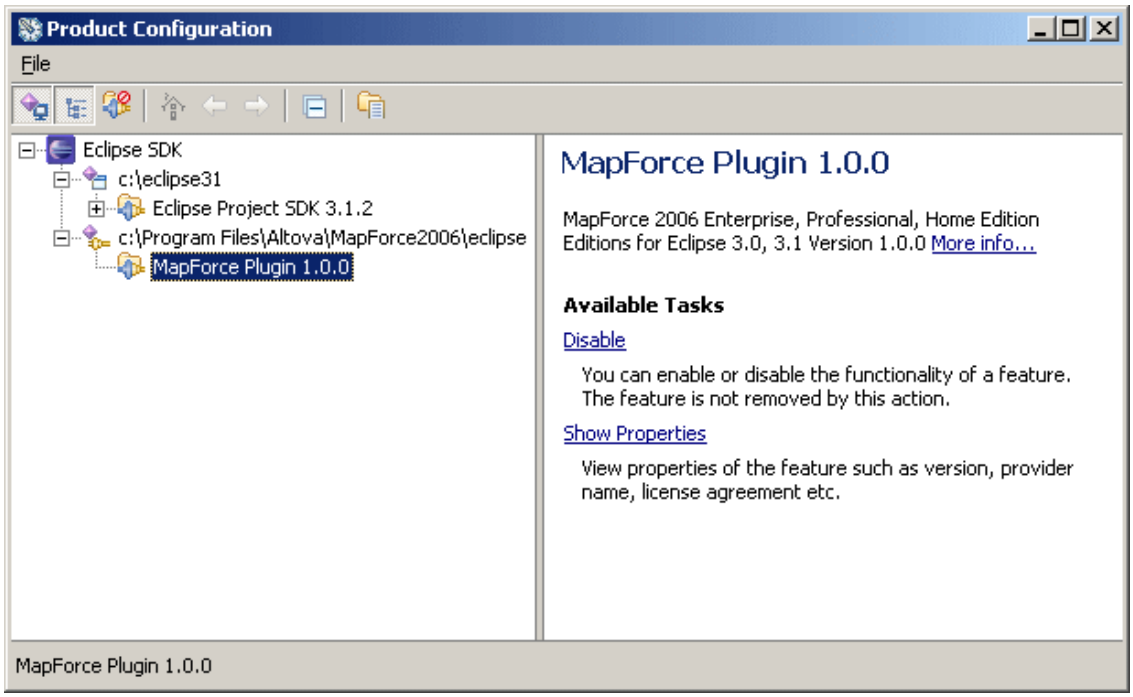
- Download and install the MapForce edition you intend to use from the Download section of the Altova.com website, i.e. Enterprise, Professional or Standard edition.
- Download and install the MapForce Plug-in for Eclipse from the Download section of the Altova.com website. You will be prompted for the installation folder of Eclipse during the installation process.

Configuring an eclipse installation to use a **previously** installed MapForce plug-in:

1. Start Eclipse and select the menu option **Help | Software Updates | Manage Configuration.**
2. Select the menu option **File | Add an extension location** and browse to the installation folder of your MapForce Eclipse plug-in e.g. C:\Program Files\Altova\MapForce2008 \eclipse.

   Follow the instructions to access a previously installed plug-in.

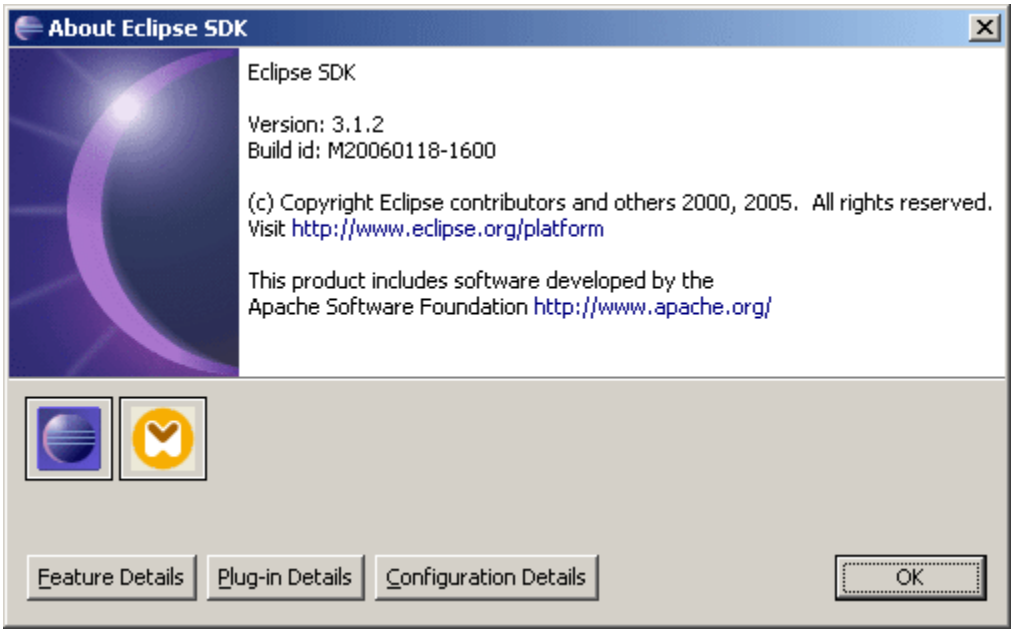
Clicking a plug-in or folder icon, displays various installation options in the right-hand pane.



1. Click the "Show Properties" link displays the specific plug-in information: Copyright, General information etc.

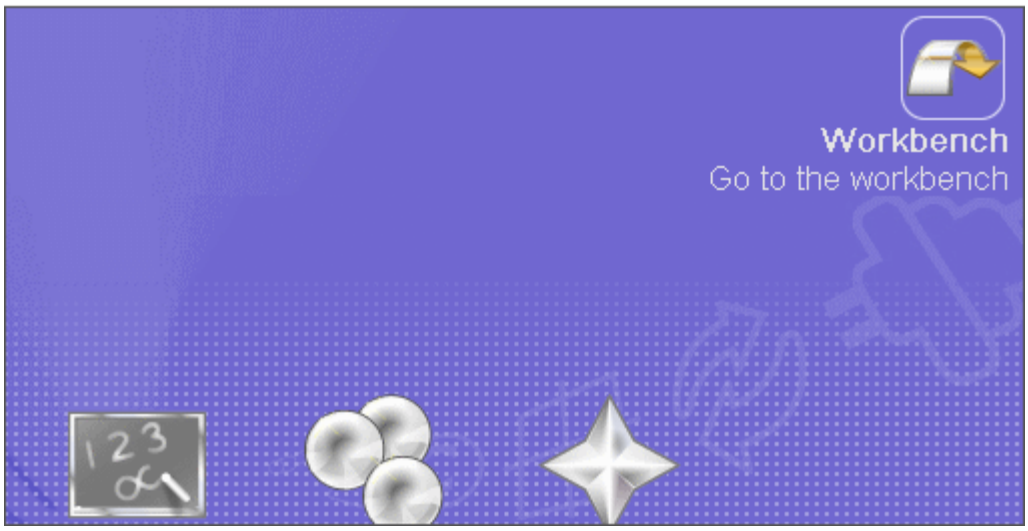
To check the currently installed version:

1. Select the menu option **Help | About Eclipse SDK.**



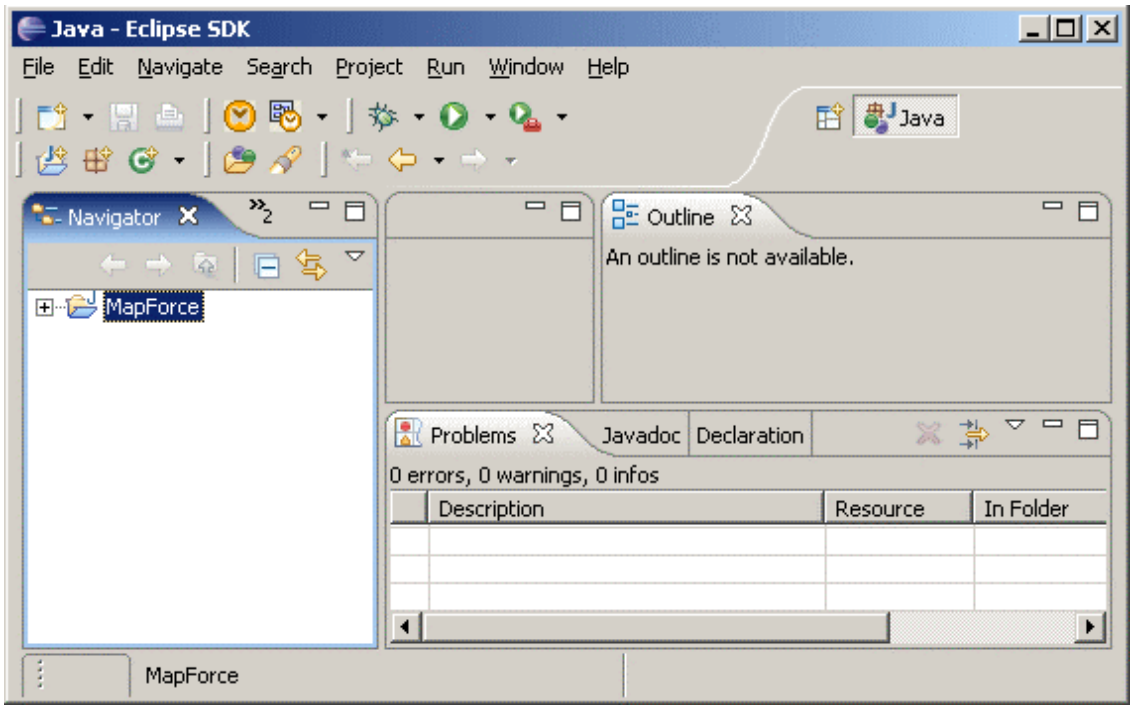
2. Click the MapForce icon, to view the version specifics.

19.2 Starting Eclipse and using MapForce plugin



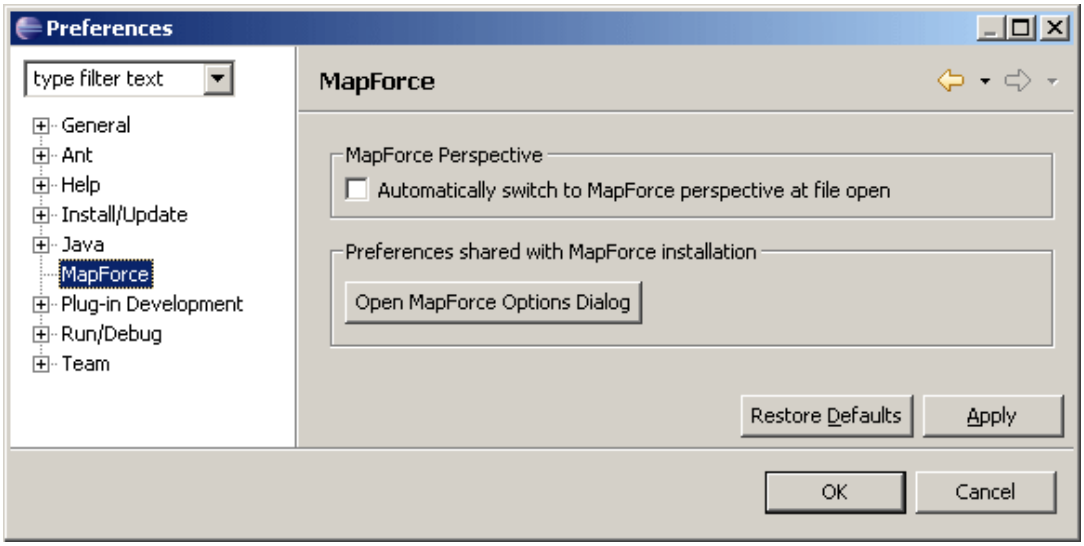
Place the cursor over the arrow symbol, and click when the "Go to the workbench" text appears. This opens an empty MapForce window in Eclipse.

Starting Eclipse and Using MapForce Plug-in:
Having used the MapForce for Eclipse installer, you are presented with an empty Eclipse environment.



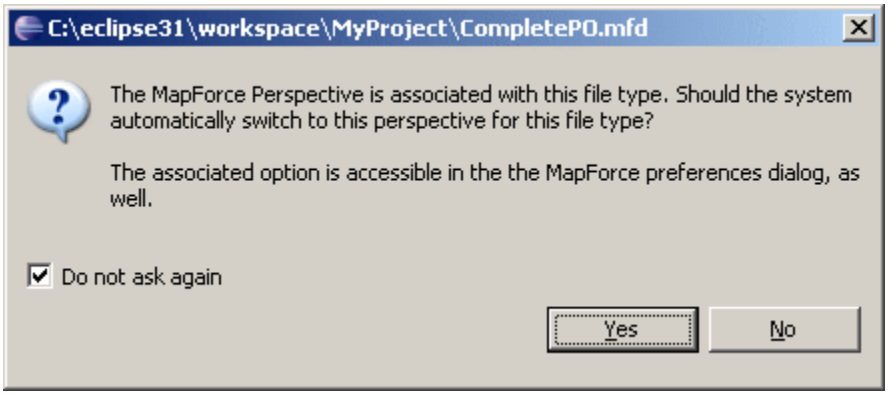
MapForce properties:

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.

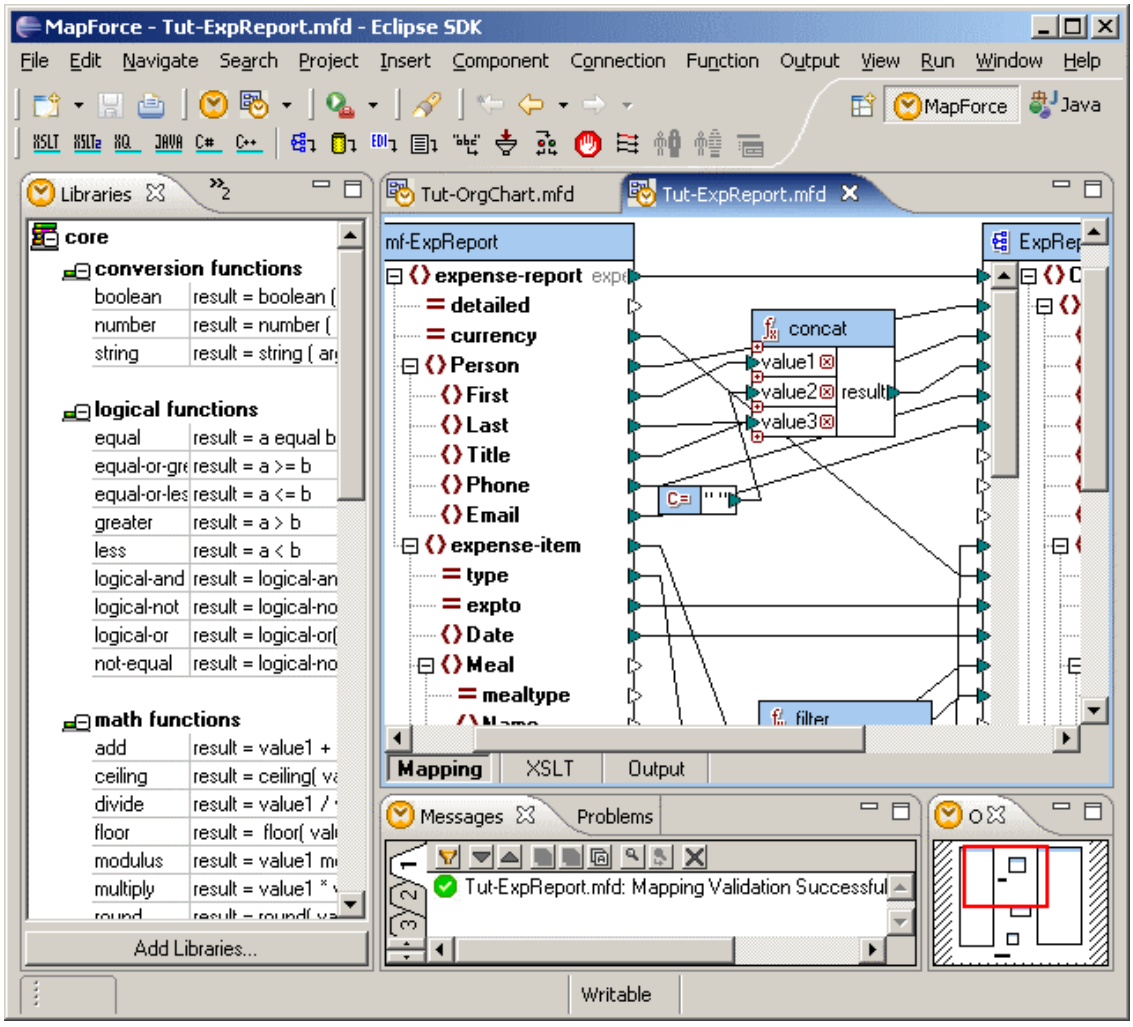


19.3 MapForce / Editor, View and Perspectives

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 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.

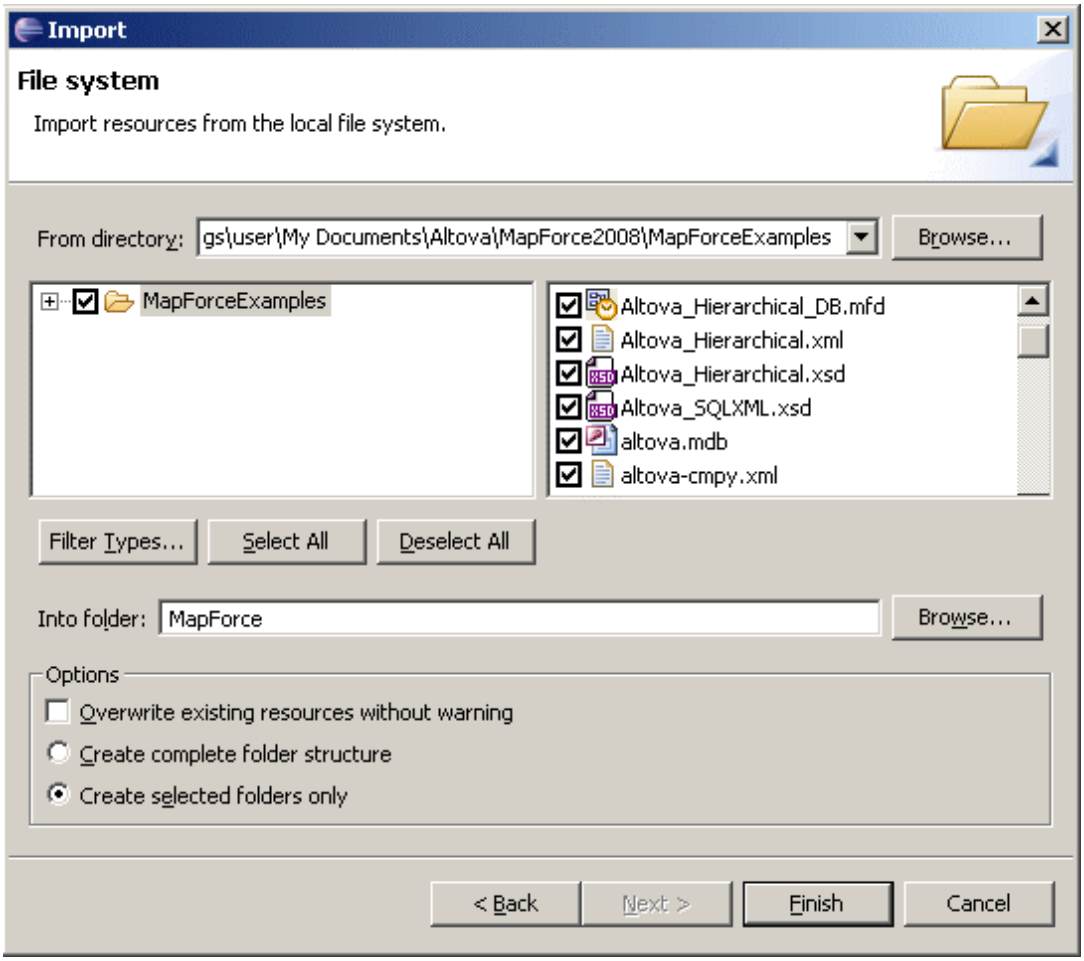
XSLT, which displays the generated XSLT code. The name of this tab reflects the programming language you have selected under Output | XSLT 1.0, Java, C# etc.

Output, which displays the Mapping output, in this case the XML data.

19.4 Importing MapForce examples folder into Navigator

To Import the MapForce Examples folder into the Navigator:

1. Right-click in the **Navigator** tab and click **Import**.
2. Select "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. Activate the **MapForceExamples** check box.
   This activates all files in the various subdirectories in the window at right.

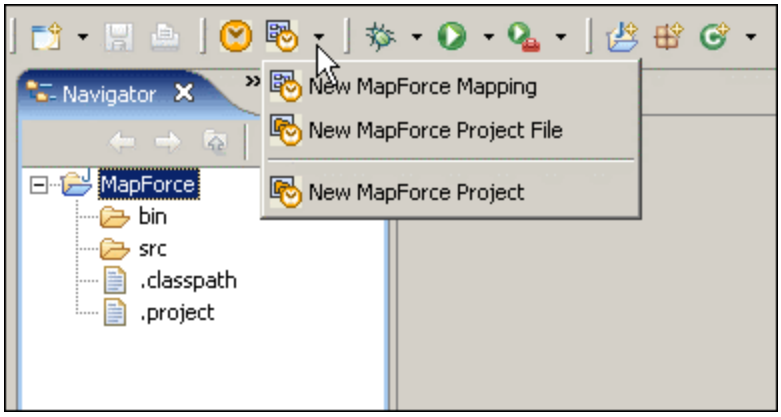


5. If not automatically supplied, click the **Browse** button, next to the "Into folder:" text box, to select the target folder, then click **Finish**.
   The selected folder structure and files will be copied into the Eclipse workspace.
6. Double-click a file in Navigator to open it.

19.5 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. You must select this option when you are creating webservices.
- New MapForce Project, creates a new MapForce/Eclipse project, adding the folder to the Navigator window. MapForce/Eclipse projects are Eclipse projects with a MapForce builder assigned to them. See Using MapForce Eclipse projects for automatic build for details.

19.6 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.

19.6.1 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:**

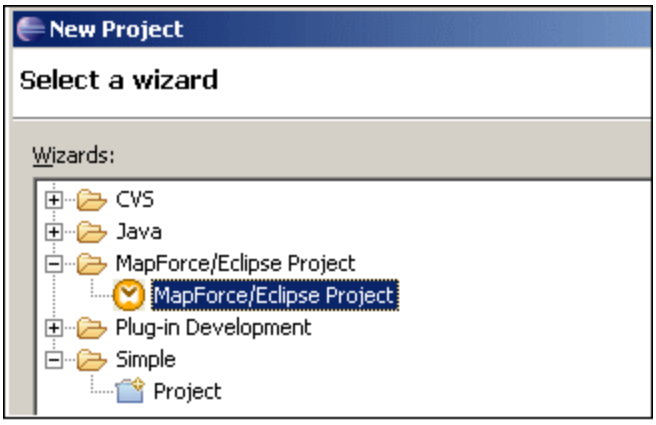
1. Open, or select the MapForce project file.
2. Select the root node or any other node in the project document.
3. Select **Generate Code**, or **Generate Code in** from the right mouse-button menu**.**
   The target folder for the generated code is determined by the properties of the selected node or properties of its parents.
4. Any errors or warnings are displayed in the MapForce Messages tab.

19.6.2 Using MapForce Eclipse projects for automatic build

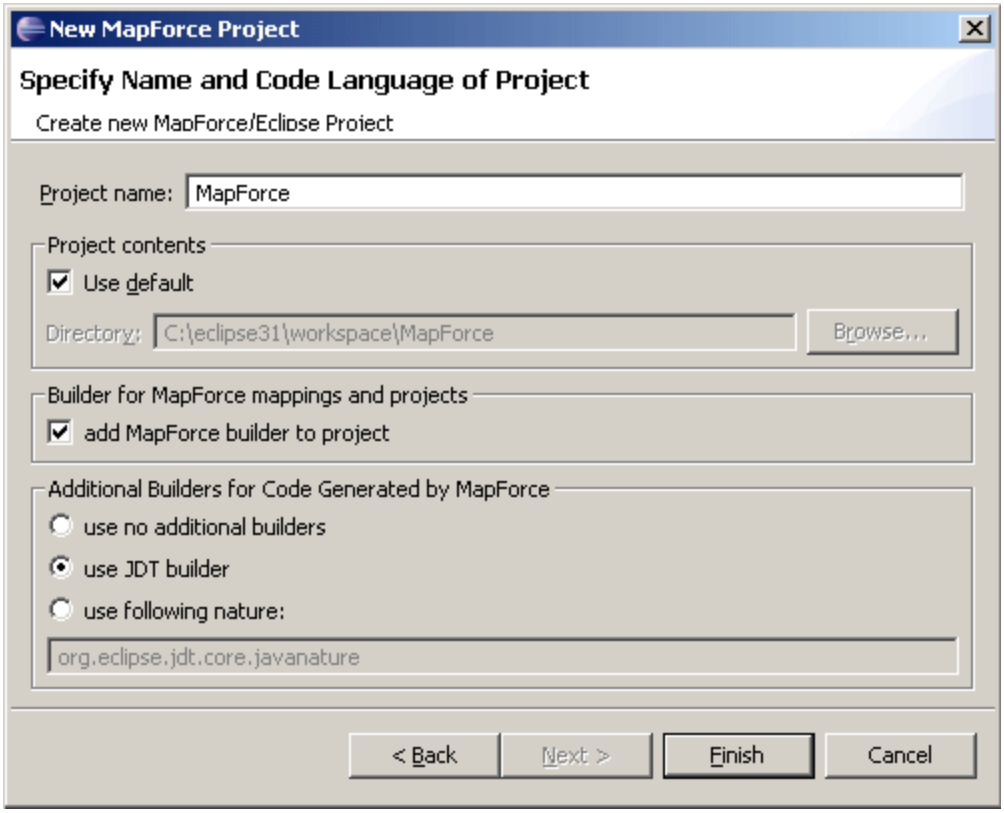
The MapForce plug-in has a built-in project builder. This builder can be identified by the project nature ID "`com.altova.maéforceecliéseélugin.MaéForceNature`". MapForce Eclipse projects have this nature automatically assigned. To use the MapForce project builder in other Eclipse projects see "Adding MapForce nature to existing Eclipse Project" for more information.

To create a new MapForce Eclipse Project:

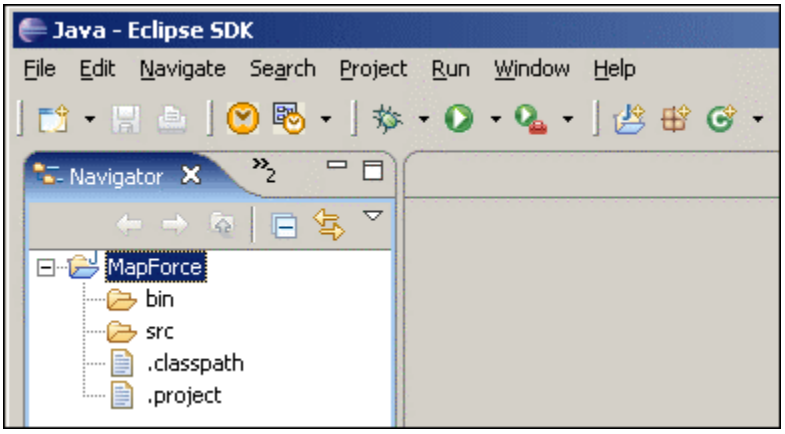
1. Click the Navigator tab to make it active.
2. Right-click in the Navigator window, and select **New | Project.**
3. Expand the MapForce/Eclipse Project entry and select MapForce/Eclipse, then click Next.



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.

19.6.3 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:\eclipse31\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.

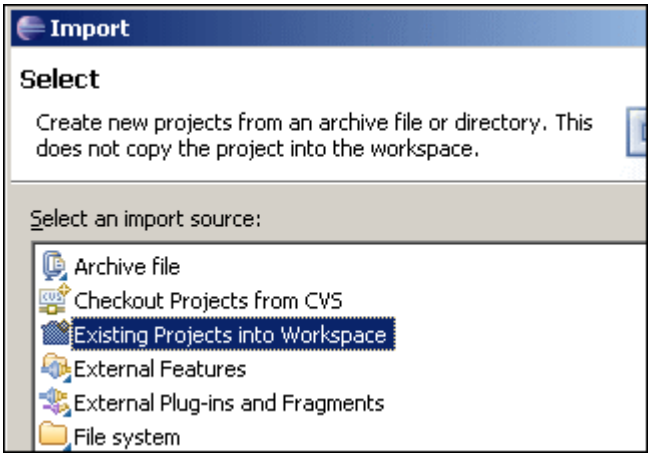
19.7 Extending MapForce plug-in

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.

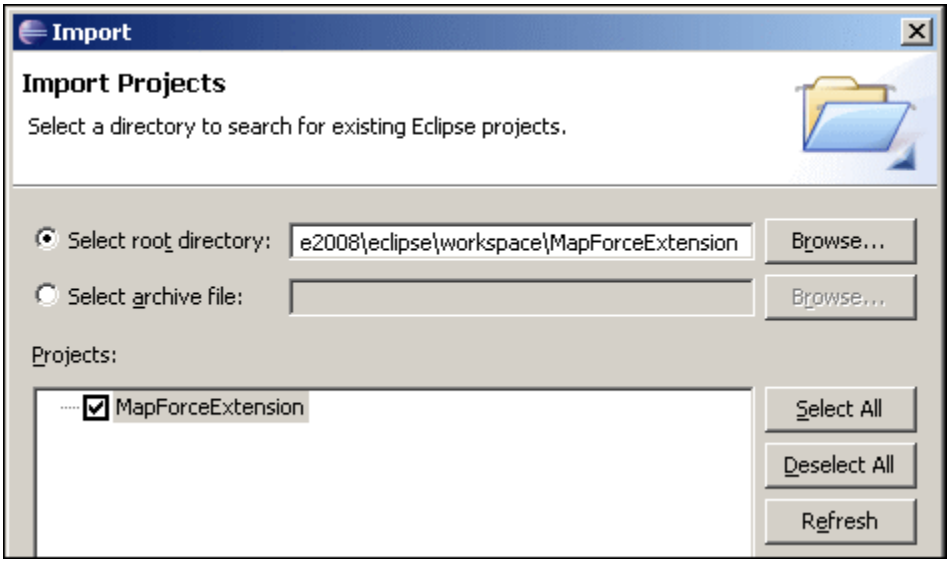
Your MapForce Eclipse installation package contains a simple example of a plug-in that uses this extension point. It checks for any file open events of any new MapForce mappings, and sets the zoom level of the mapping view to 70%.

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\MapForce2008\eclipse\workspace\MapForceExtension**).



5. Click Finish.

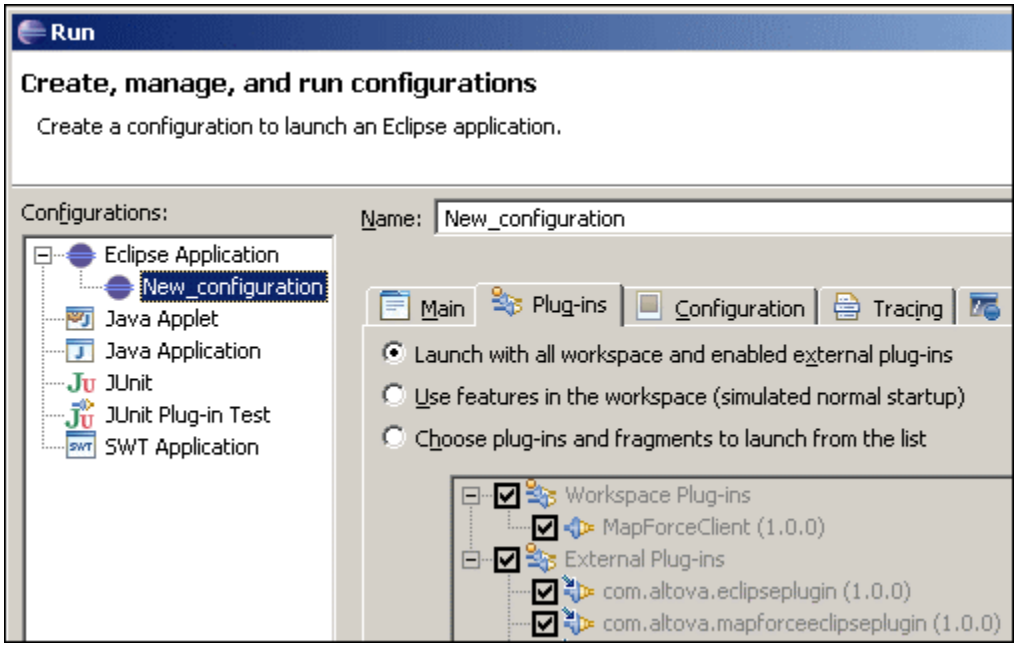
A new project named "MapForceExtension" has been created in your workspace.

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\MapForce2008\eclipse\plugins\com.altova.mapforceeclipseplugin_10.2.0\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 20

MapForce Reference

20 MapForce Reference

The following section lists all the menus and menu options in MapForce, and supplies a short description of each.

20.1 File

New
Creates a new mapping document, or mapping project (mfp)

Open
Opens previously defined mapping (*.mfd) , or mapping project (mfp) files.

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.

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.

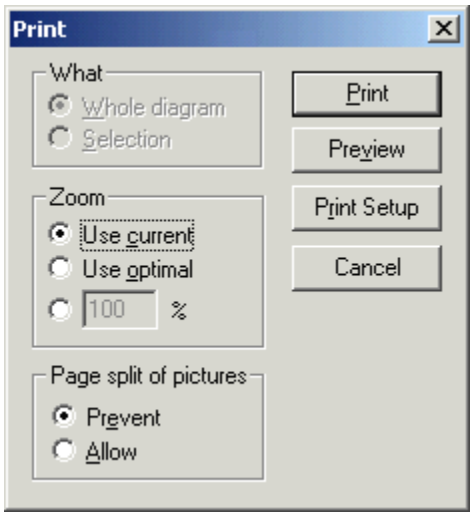
Close all
Closes all currently open mapping files. You are asked if you want to save any of the unsaved mapping files.

Save All
Saves all currently open 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

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.

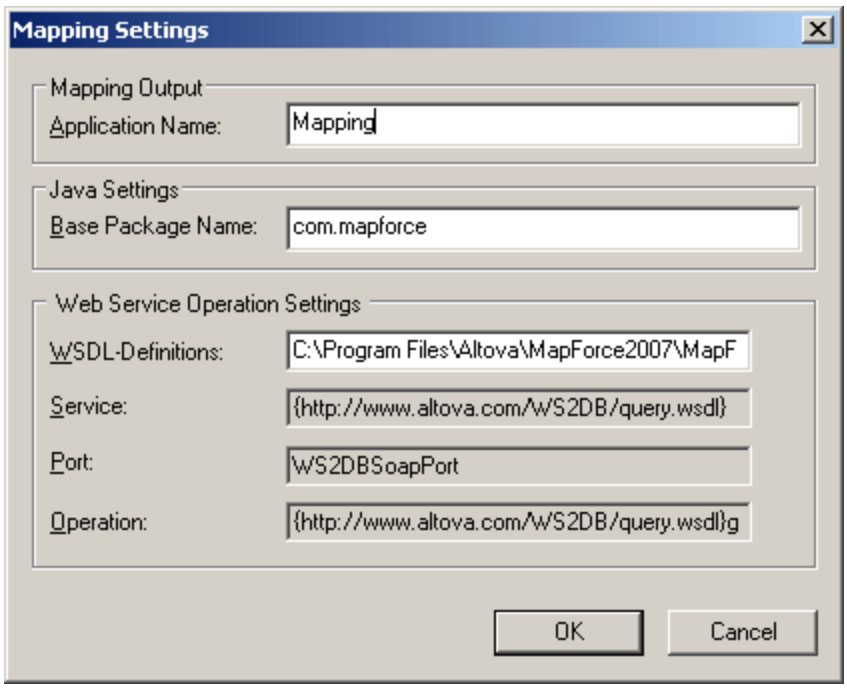
Generate code in | XQuery
This command generates the XQuery file(s) needed for the transformation from the source file(s).
Selecting this option opens the Browse for Folder dialog box where you select the location of the XQuery file.

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 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.

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.

Web service Operation settings:
The WSDL-Definitions, Service, Port and Operation fields are automatically filled if the mapping document is part of a Web service implementation.

Recent files
Displays a list of the most recently opened files. You can also use the hotkeys ALT+F+F+1 (from 1-9) to open a file.

Recent projects
Displays a list of the most recently opened projects. You can also use the hotkeys ALT+F+T+1 (from 1-9) to open a project.

Exit
Exits the application. You are asked if you want to save any unsaved files.

20.2 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.

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.

20.3 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.

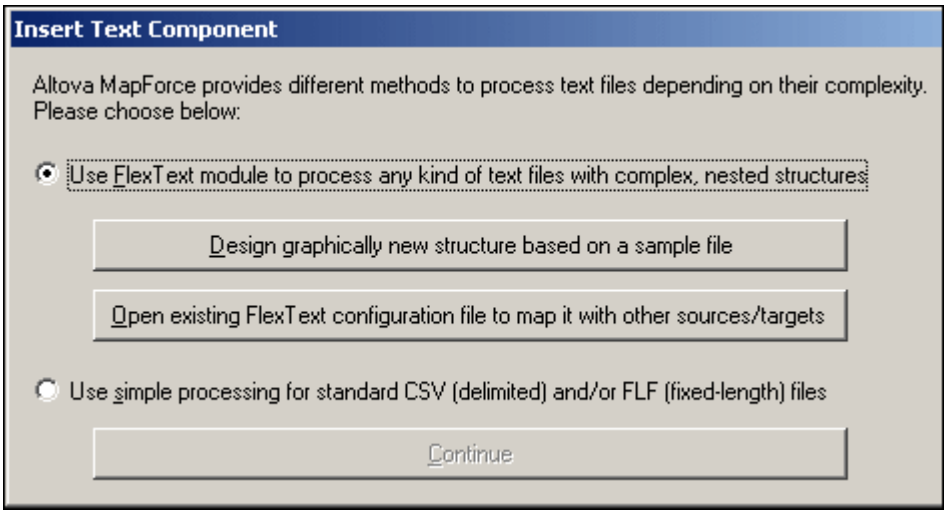
Database
Inserts a database component as data source or target component. The database supplies the schema structure and displays it in a tree view.

EDI
Inserts an EDI document which can be used as the data source, or as a target component. MapForce currently supports the Electronic Data Interchange formats **UN/EDIFACT**, as well as **ANSI X12**, and allows you map data from these type of documents to/from XML Schema, text files, and database components. Please see the section on EDI for more information.

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.

This opens the Insert Text Component dialog box, where you can choose if you want to:

- Design a FlexText template based on a sample file.
- Open an existing FlexText template to map it to other target components
- Insert a standard CSV or FLF text file.



Web service function
Inserts a Web service function defined by a WSDL file into a mapping. Connecting input and output components to it allows you to immediately preview the result of the Web service function in the output window. Please see "Calling Web services" for more information.

Excel 2007 file
Inserts a Microsoft Excel 2007 (Office Open XML) files (*.xlsx), as both source and target component. Please see: Mapping MS OOXML Excel 2007 files for more information.

Microsoft Excel 2007 only needs to be installed if you intend to **preview** Excel 2007 sheets in the **Output** tab. If you use a previous version of Microsoft Excel, or you don't have Excel installed at all, you will still be able to map to/from Excel 2007 files. You cannot preview the result in the Output tab, but you can still save it, by clicking the Save output icon.

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.

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.

Exception 
The exception component allows you to interrupt a mapping process when a specific condition is met, or define Fault messages when using WSDL mapping projects. Please see MapForce Exceptions, or Web service faults for more information.

20.4 Project

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

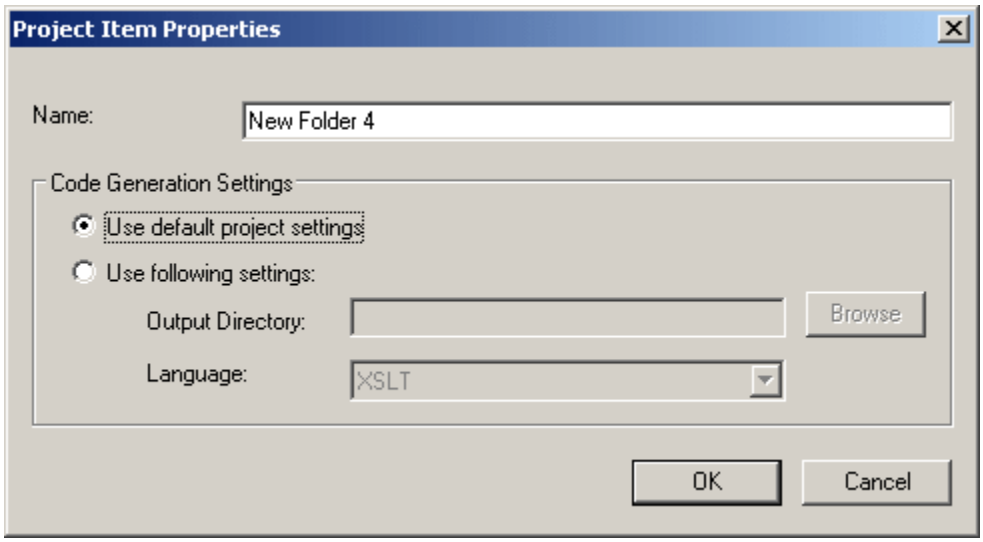
Two types of projects can be defined:

- A collection of individual mappings, a standard project
- A related set of mappings, which make up a WSDL mapping project
- Both project types support code generation for the entire project

Add files to project:
Allows you to add mappings to the current project through the Open dialog box.

Add active file to project:
Adds the currently active file to the currently open project.

Create Folder:
This option adds a new folder to the current project structure, and only becomes active when this is possible.
The default project settings can be applied, or you can define your own by clicking the "Use following settings" radio button.



Open Operation's Mapping:
This option is only available when working with WSDL files. If a mapping for the selected WSDL operation exists, then this option opens it in the Mapping pane.

Create Mapping for Operation:
Creates a mapping file for the currently selected operation of the WSDL project. The operation name defined in the WSDL file is supplied in the "Save as" dialog box, which is opened automatically.

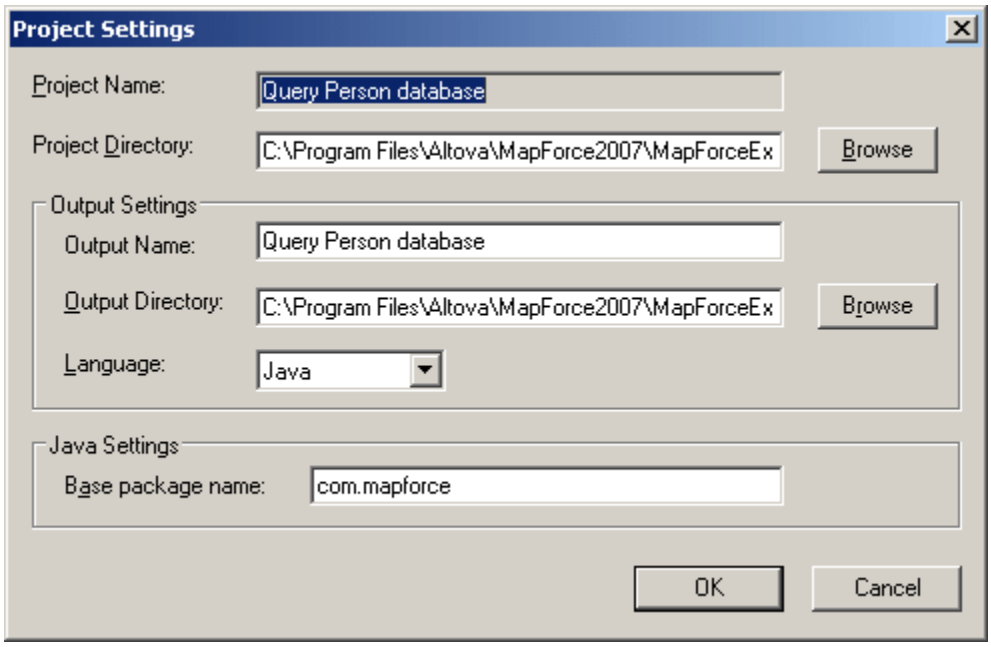
Add Mapping file for Operation:
Allows you to add a previously saved mapping file to the currently active WSDL operation. Select the mapping file from the "Open" dialog box.

Remove item:
Removes the currently selected item from the project tree.

Insert Web Service...
Allows you to insert a Web Service based on an existing WSDL file.

Open WSDL file in XMLSpy
Opens the selected WSDL file, highlighted in the Project window, in XMLSpy.

Project properties:
Opens the Project properties dialog box.



Generate code in default language:
Generates project code in the currently selected default language. Right click the project name in the Project window, and select **Project settings** to define the default language.

Generate code in...
Generates project code in the language you select from the flyout menu.

20.5 Component

Align tree left
Aligns all the items along the left hand window border.

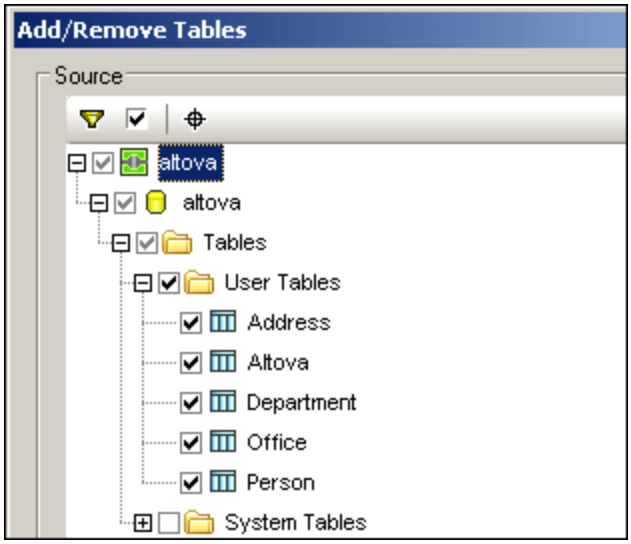
Align tree right
Aligns all the items along the right hand window border. This display is useful when creating mappings to the target schema.

Change Root element
Allows you to change the root element of the XML instance document. Useful in the target schema window, as this limits or preselects the schema data.

Edit Schema Definition in XMLSpy
Selecting this option, having previously clicked an XML-Schema/document, opens the XML Schema file in the Schema view of XMLSpy where you can edit it.

Edit FlexText configuration
Opens FlextText and enables you to edit a previously created FlexText file.

Add/Remove tables
Allows you to change the number of tables in the database component by selecting/deselecting them in the Database Tables group.



Duplicate input
Inserts a copy/clone of the selected item, allowing you to map multiple input data to this 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.

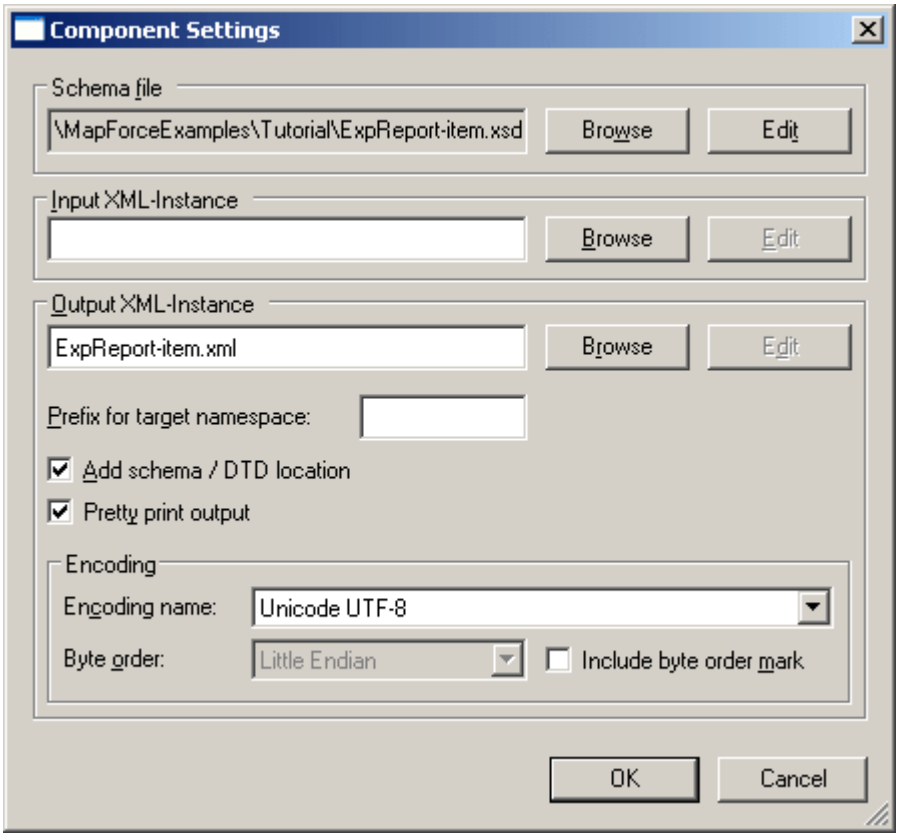
Remove duplicate
Removes a previously defined duplicate item. Please see the Duplicating input items section in the tutorial for more information.

Database Table actions
Allows you to define the table actions to be performed on the specific target database table. Table actions are: Insert, Update, and Delete, please see Mapping data to databases for more information.

Database Key settings
Allows you to define the Key settings of database fields, please see Database key settings for

more information.

Component Settings
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-Instance: 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-Instance for code generation: This is file name and path where the XML target instance is placed, when generating and executing program code.

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.**

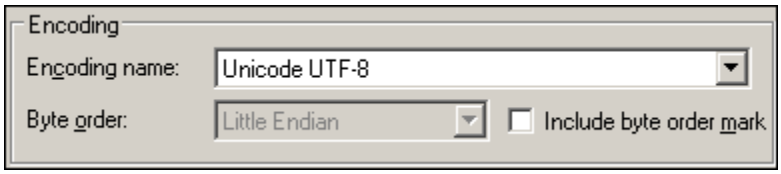
Prefix for target namespace: Allows you to enter a prefix for the Target Namespace if this is a schema / XML document. A Target namespace has to be defined in the target schema, for the prefix to be assigned here.

Add schema location: Adds the absolute path of the referenced XML Schema file to the root element of the XML output.

Deactivating this option allows you to decouple the XML instance from the referenced XML Schema or DTD. Use this option if you want to send the resulting XML output to someone who does not have access to the underlying XML Schema.

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 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, EDI and Flextext 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.

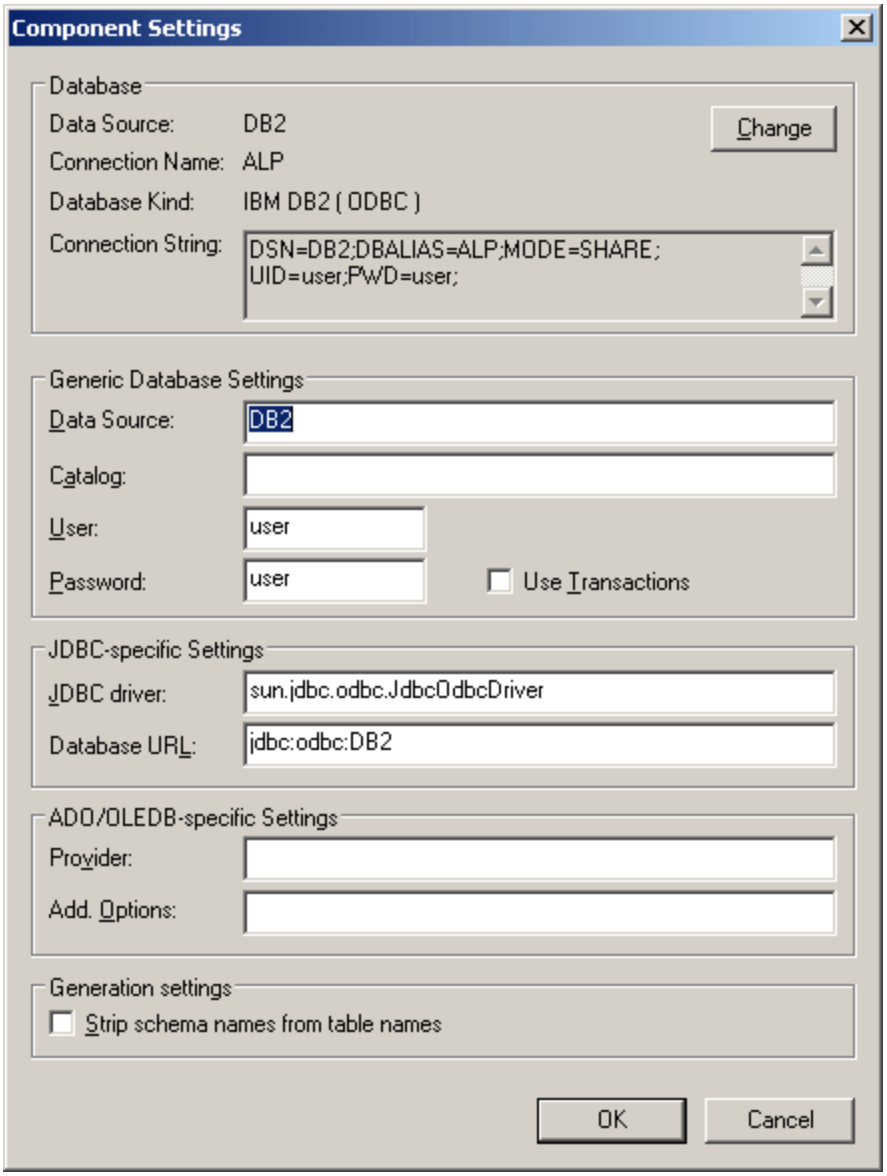
The encoding control group consists of 3 controls:

- 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.

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.

20.6 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.

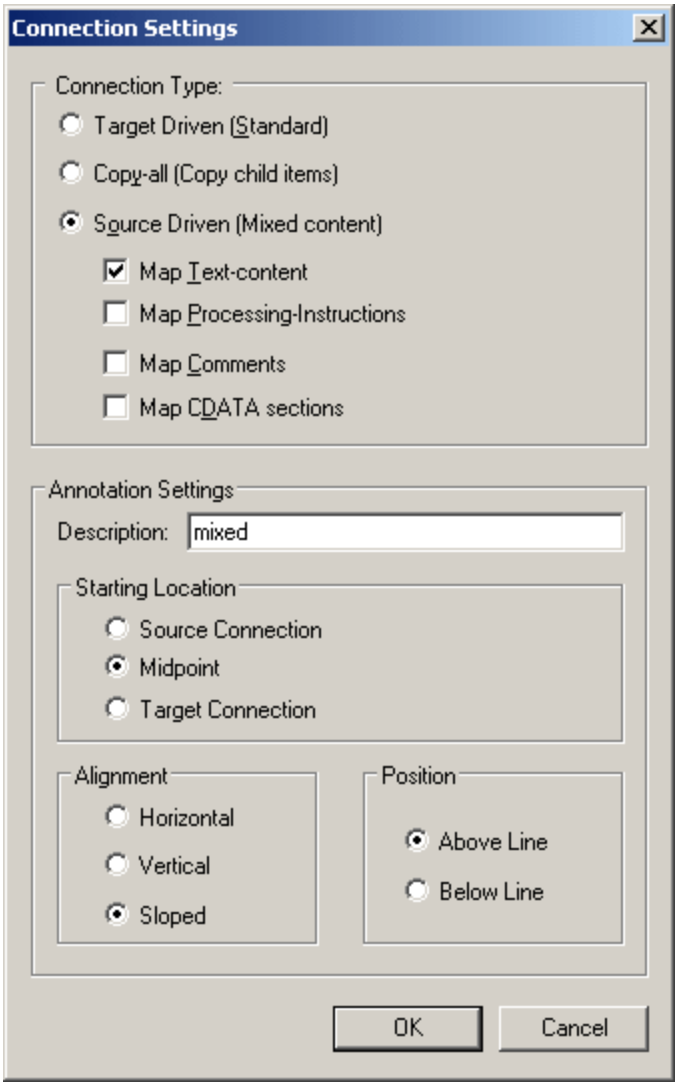
Target Driven (Standard)
Changes the connector type to Standard mapping, please see: "Source-driven / mixed content vs. standard mapping" for more information.

Copy-all
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.

Connection settings:
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.

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.

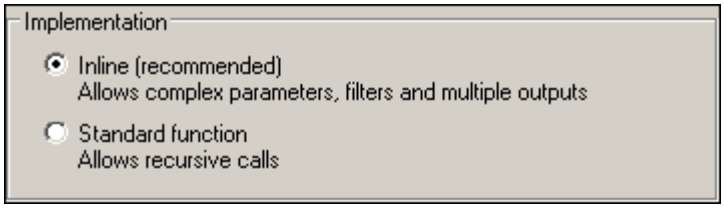
1. Double click a connector and enter the name of the connector in the Description field. This enables all the options in the Annotation Settings group.
2. Use the remaining groups to define the position and alignment of the label.

20.7 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. Please note that a single output function is automatically inserted when you define such a function, and that only one output component can be present in a user-defined function. Please see "Creating a user-defined function from scratch" for more information.

Create user-defined function based on 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.

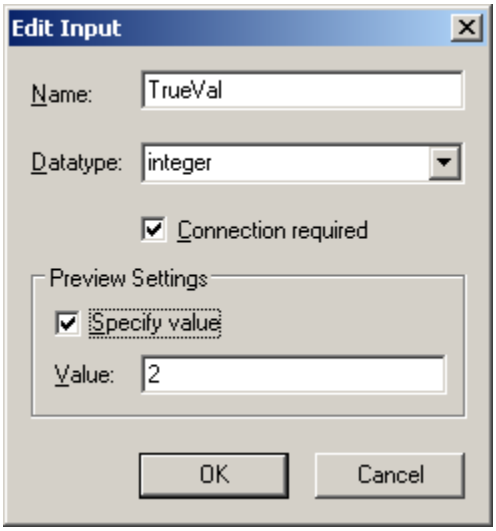
Function settings:
Opens the settings dialog box of the currently active user-defined function allowing you to change the current settings. Use this method to change the user-defined function type, i.e. double click the title bar of a user-defined function to see its contents, then select this menu option to change its type.



Insert Input:
Inserts an "input" component into the mapping, or into a user-defined function.

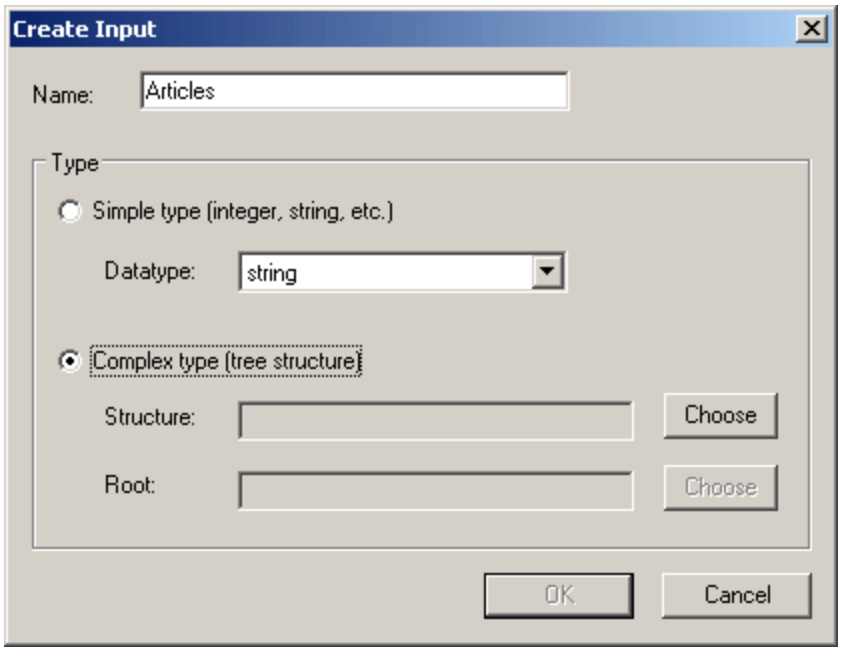
If you are working in the main Mapping tab, the dialog box shown below is displayed. This type of input component allows you to:

- define an **override** value for the data that is being input by the current mapping input, and
- use this input component as a **parameter** in the command line execution of the compiled mapping.
  Please see "Input values, overrides and command line parameters" for more information.



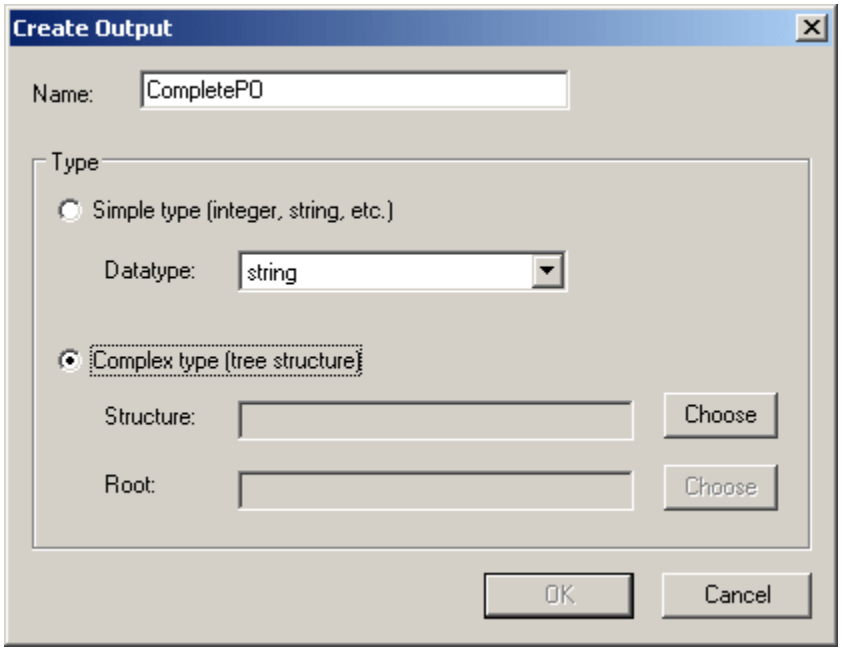
If you are working in a user-defined function tab, the dialog box shown below is displayed. This type of input component allows you to define:

- simple inputs, if this is a Standard user-defined function
- complex inputs, e.g. schema structures, if this is an Inline user-defined function.



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, if this is a Standard user-defined function
- complex outputs, e.g. schema structures, if this is an Inline user-defined function.



20.8 Output

The **XSLT, XSLT2**, **XQuery, Java, C#, or C++** options allow you to define the target language you want your code to be in. Note that the MapForce engine presents a preview of the mapping result, when you click the Output tab. Please see the MapForce engine section for more information.

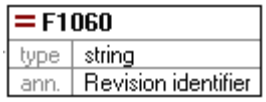
Validate Output XML file
Validates the resultant XML file against the referenced schema.

Save Output XML File
Saves the currently visible data in the Output tab.

Run SQL-script
Executes the pseudo-SQL select statements visible in the Output window, when the target component is a database.

20.9 View

Show Annotations 
Displays XML schema annotations (as well as EDI info) in the component window.
If the Show Types icon is also active, then both sets of info are show in grid form



Show Types 
Displays the schema datatypes for each element or attribute.
If the Show Annotations icon is also active, then both sets of info are show in grid form.

Show library in function header 
Displays the library name in parenthesis in the function title.

Show Tips 
Displays a tooltip containing explanatory text when the mouse pointer is placed over a function.

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.

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.

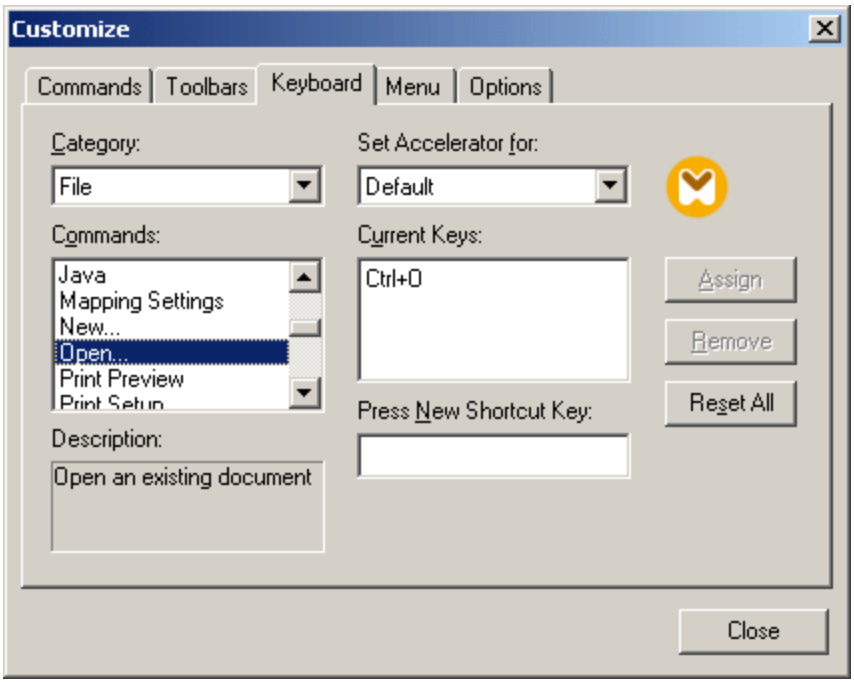
20.10 Tools

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 tab
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

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

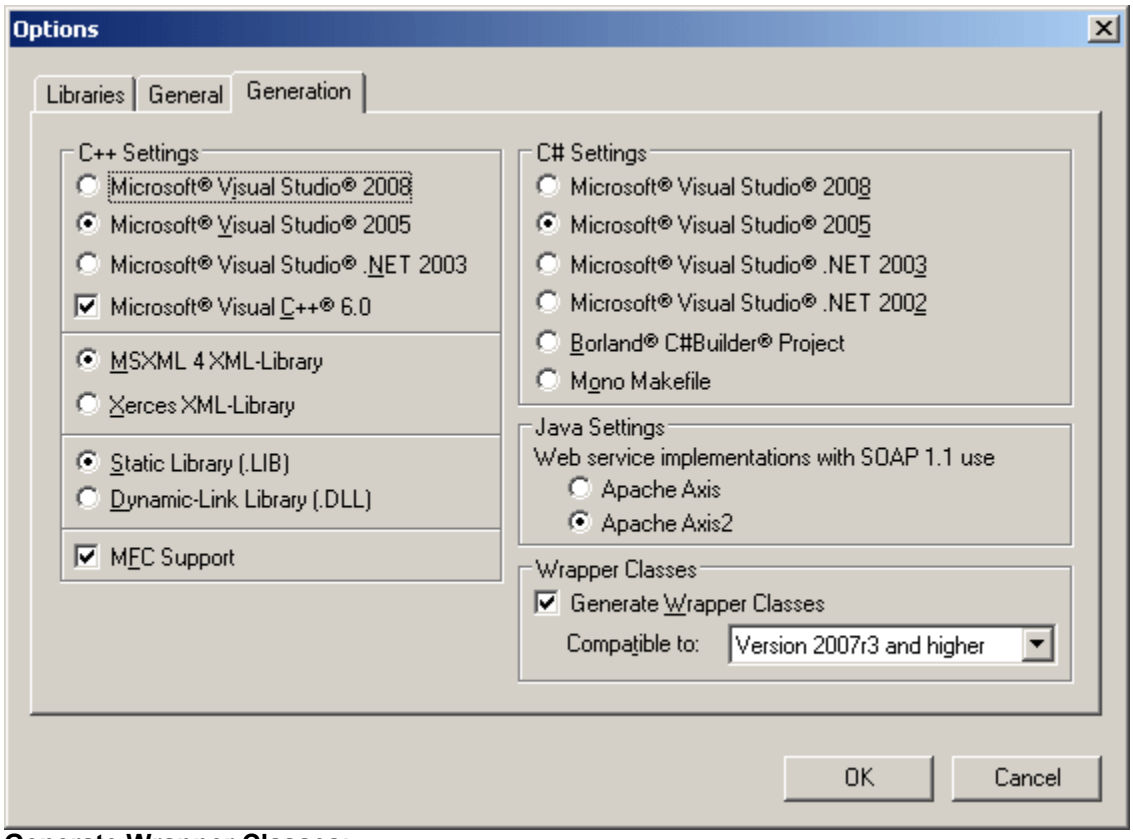
Zooming hotkeys

CTRL + mouse wheel forward	Zoom In
CTRL + mouse wheel back	Zoom Out
CTRL + 0 (Zero)	Reset Zoom

Options

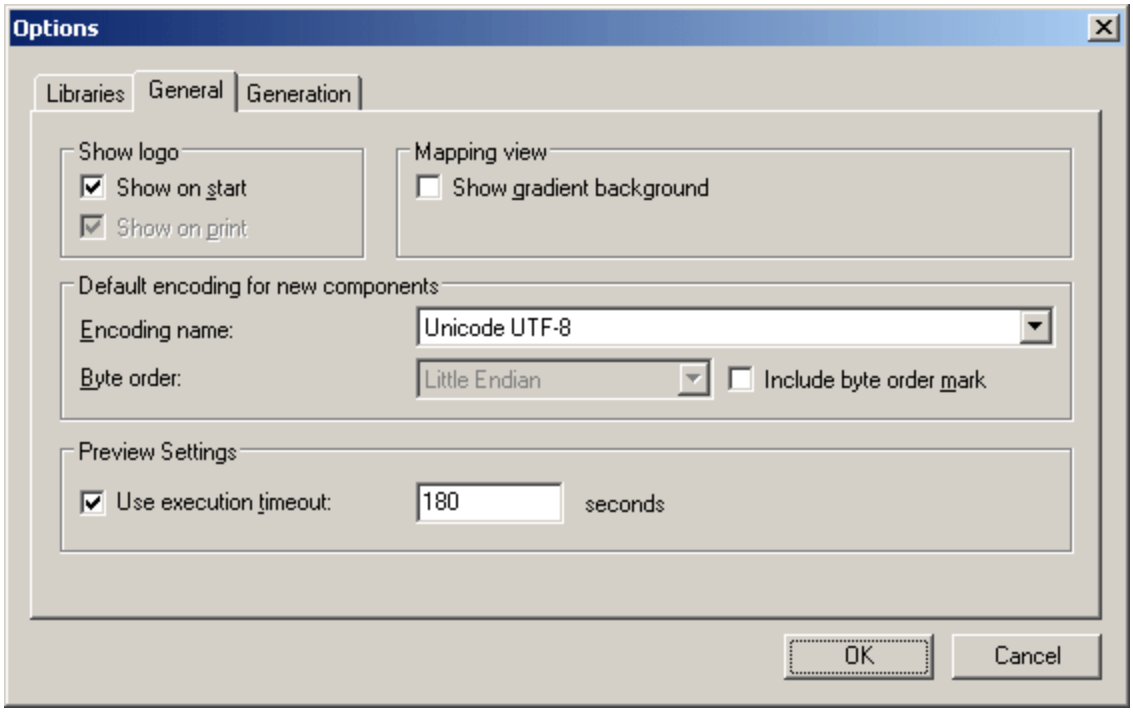
Opens the Options dialog box through which you can:

- 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.
-



Generate Wrapper Classes:
Allows you to select the version compatibility of the generated wrapper classes. Currently available options are: V2005r3, 2006-2007, 2007r3 and higher.

Java Settings:
Select the correct Apache Axis version depending on the SOAP protocol you are using. SOAP 1.2 requires Axis2.



General tab:

- Specify if you want to show the logo on start and/or when printing.
- Enable/disable the MapForce gradient background
- Define the Default Output Encoding
- an execution timeout for the Output tab when previewing the mapping result.

Libraries tab:

- Add or delete user-defined XSLT, or programming language Libraries/functions to MapForce.

C++ Settings:
Defines the specific compiler settings for the C++ environment.

C# Settings:
Defines the specific compiler settings for the C# environment.

20.11 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

20.11.1 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.

20.11.2 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

20.11.3 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.

20.12 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 21

Code Generator

21 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, databases and UN/EDIFACT and ASC X12 files.

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.

21.1 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++ 2008 Microsoft Visual C++ 2005 Microsoft Visual C++ .NET 2003 Microsoft Visual C++ 6.0	Microsoft Visual Studio 2008 Microsoft Visual Studio 2005 Microsoft Visual Studio .NET 2003 Microsoft Visual Studio .NET 2002 Borland C#Builder Mono (Makefile)	Apache Ant (build.xml file) Eclipse 3.x Borland JBuilder
XML DOM implementations	MSXML 4.0 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 4.0, 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 2003 to 2008 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 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 .NET, 2003, 2005, 2008 and Borland C#Builder.

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 wrapper libraries	Database name folder	☑
EDI wrapper libraries	EDI message 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).

21.2 What's new ...

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

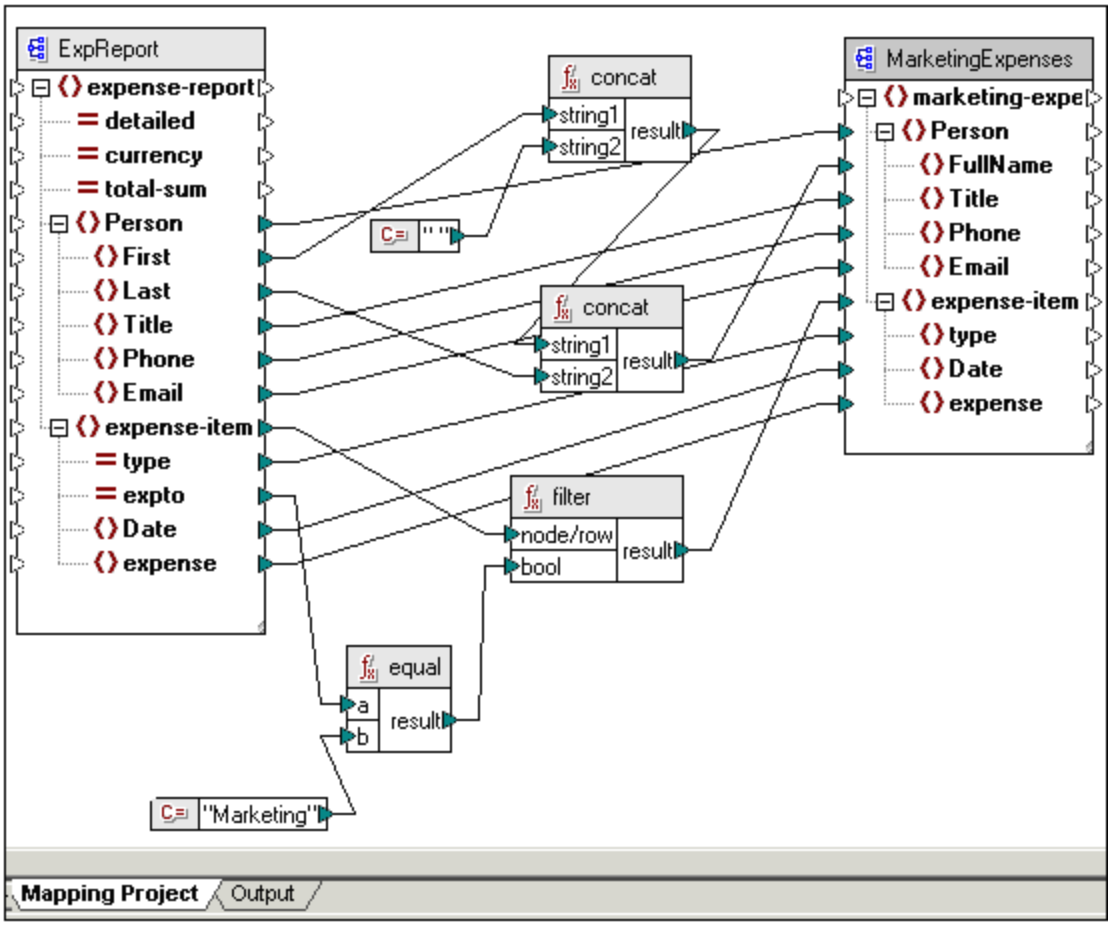
Version 2007 R3
Code Generator has been redesigned for version 2007 release 3 to simplify usage of the generated code, reduce code volume and increase performance.

- 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

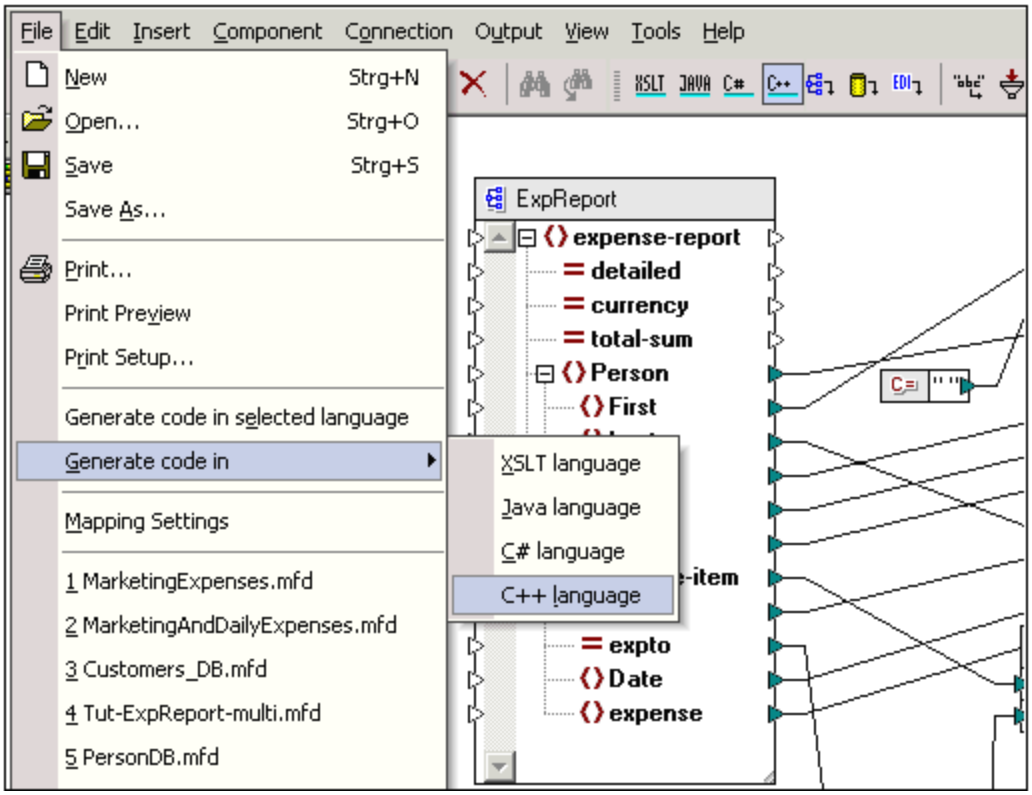
21.3 Generating program code

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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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

21.3.1 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:

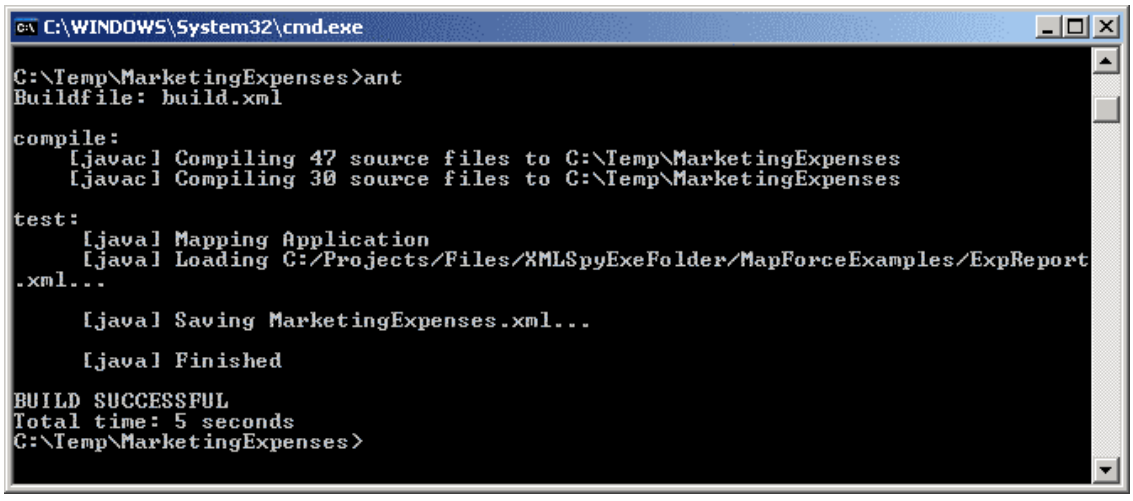
File name (with default application name)	Note
MappingConsole.java	Console application
MappingApplication.java	Dialog application

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.

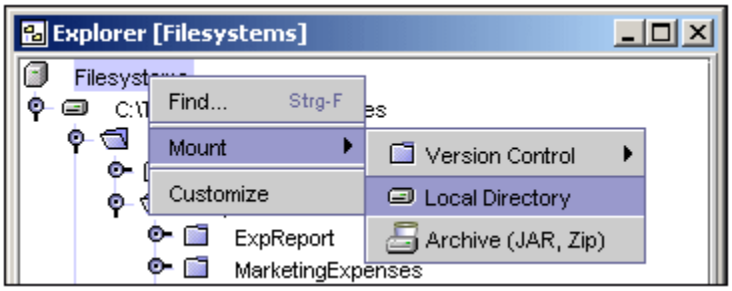


For further information please see:
Generating Java code using JBuilder

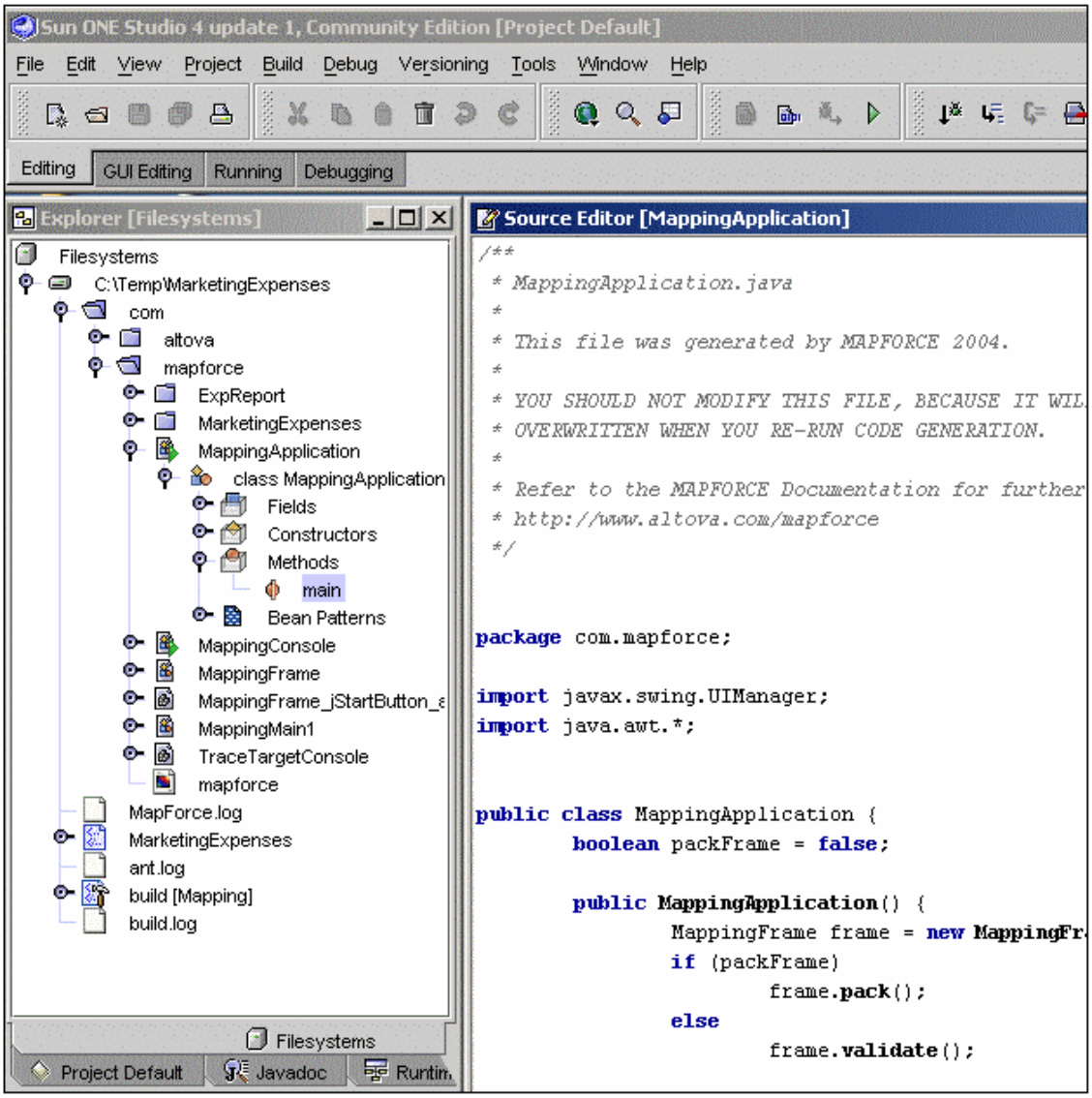
Generating Java code using Sun ONE Studio

If you are using **Sun ONE Studio:**

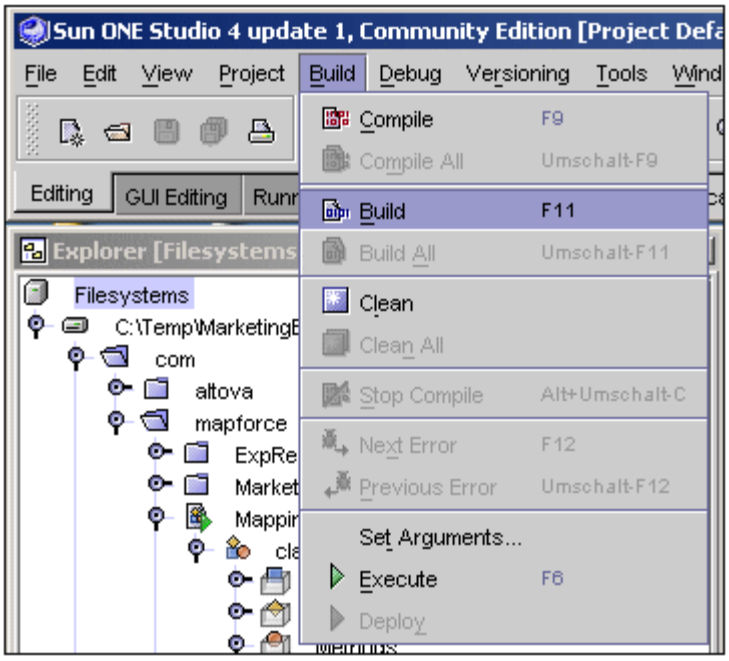
- Open Sun ONE studio.
- **Mount** the target directory you specified when you generated the Java code.



The Explorer window at left displays the source file structure.

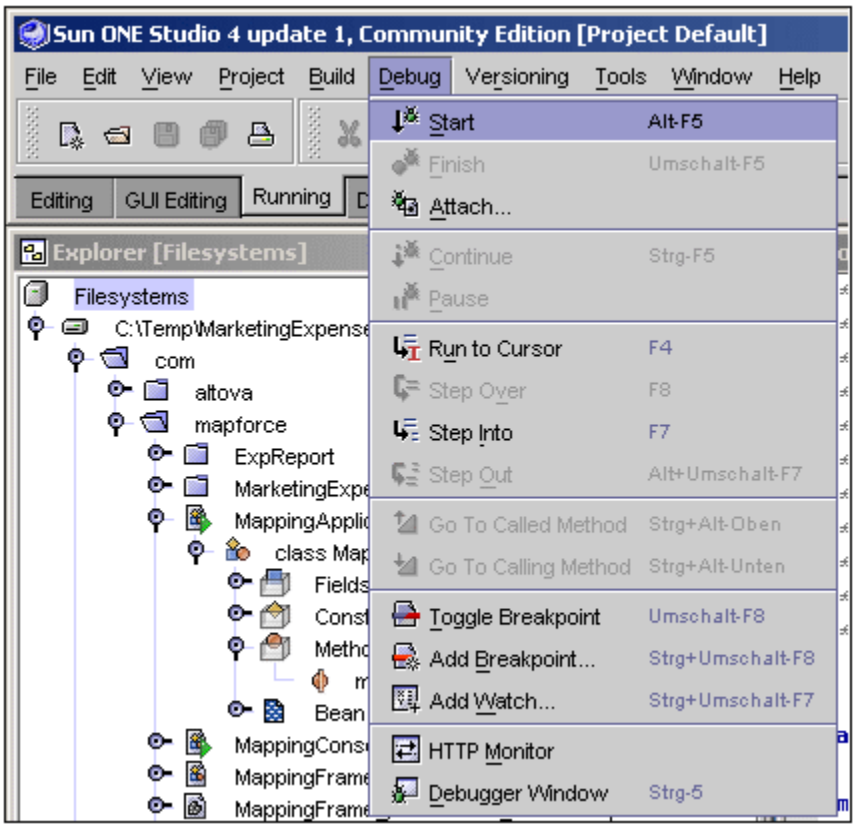


- Open either the **MappingApplication**, or **MappingConsole** folder, depending on which you want to edit and compile.
- Click the **main** method (in the Methods folder) to view the generated code.
- Select the menu option **Build | Build** to build the selected application, either:
  - MappingApplication, or
  - MappingConsole



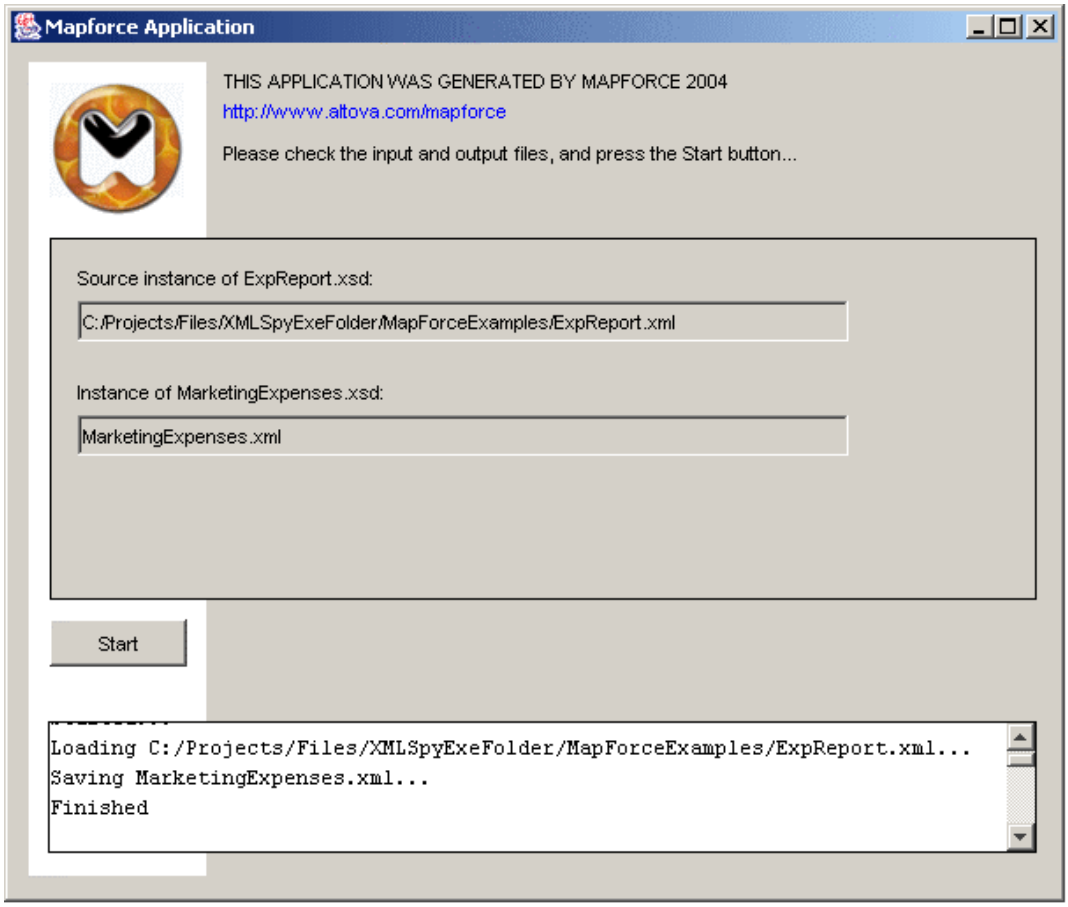
The build window displays a "successful" message when complete.

- Select the menu option **Debug | Start** to execute the selected application.



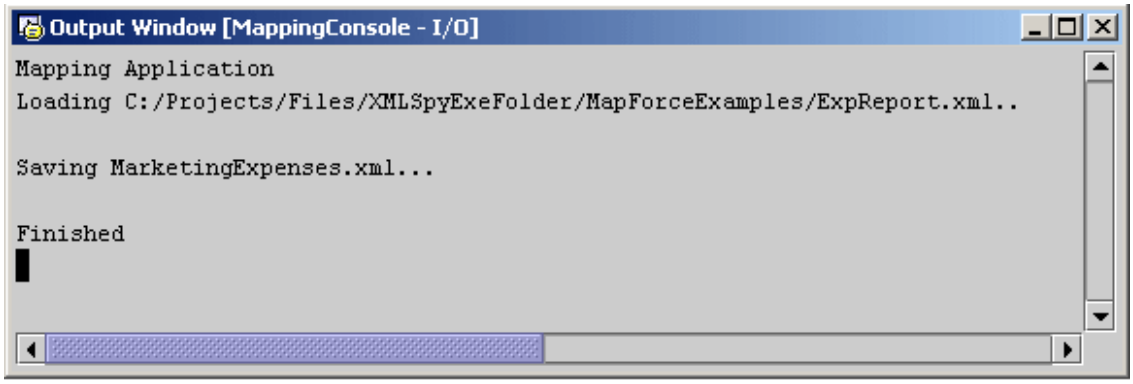
In both cases, MappingApplication or MappingConsole, the **MarketingExpenses.xml** target file is created.

MappingApplication



MappingConsole

The screenshot below shows the mapping console output in Sun ONE Studio.



21.3.2 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 **cspsroj** files are generated for Microsoft Visual Studio. The **Generation** tab under menu option **Tools | Options** allows you to choose the target IDE version, or to generate a makefile for Mono.

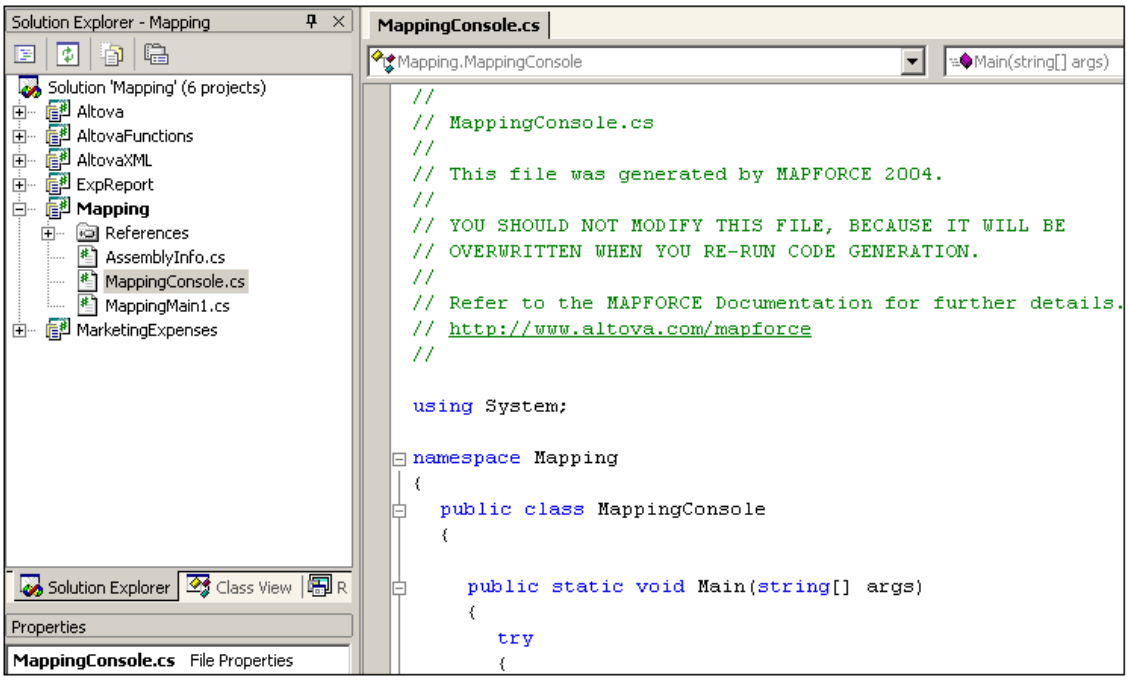
To generate C# code in MapForce:

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).

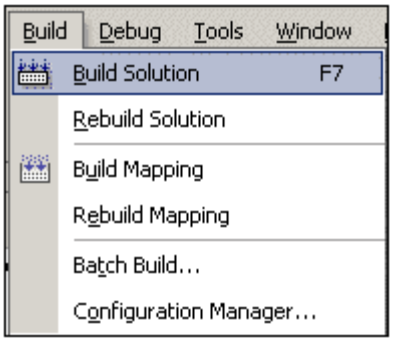
Folder **c:\codegen\C#\mapping**
You can compile the project in Microsoft Visual Studio, or use the Mono makefile.

- Mapping.sln (for Visual Studio)
- The makefile is placed in the C# directory, if **Mono Makefile** was selected in the Generation tab of the (Tools) Options dialog box.

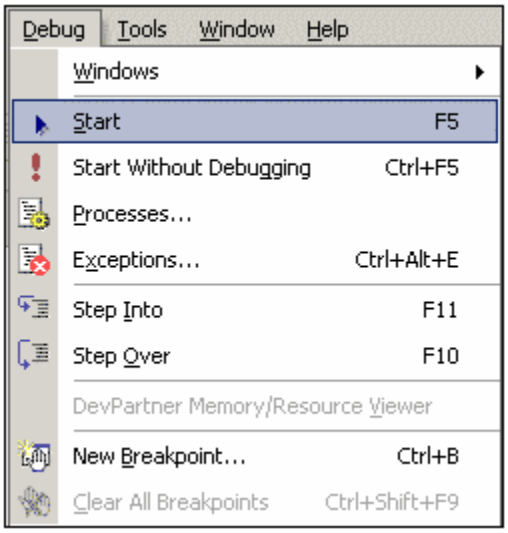
1. Navigate to the **Mapping** subdirectory, and open the Mapping solution file **Mapping.sln** in Microsoft Visual Studio.



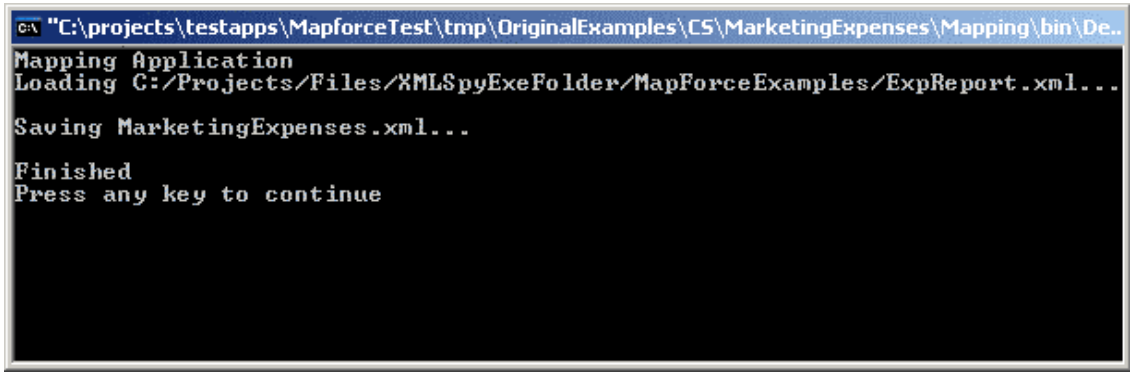
2. Select the menu option **Build | Build Solution** to compile the mapping project.



3. Select the menu option **Debug | Run** to start the application.



The mapping application is started and the target XML file is created.



21.3.3 Generating C++ code

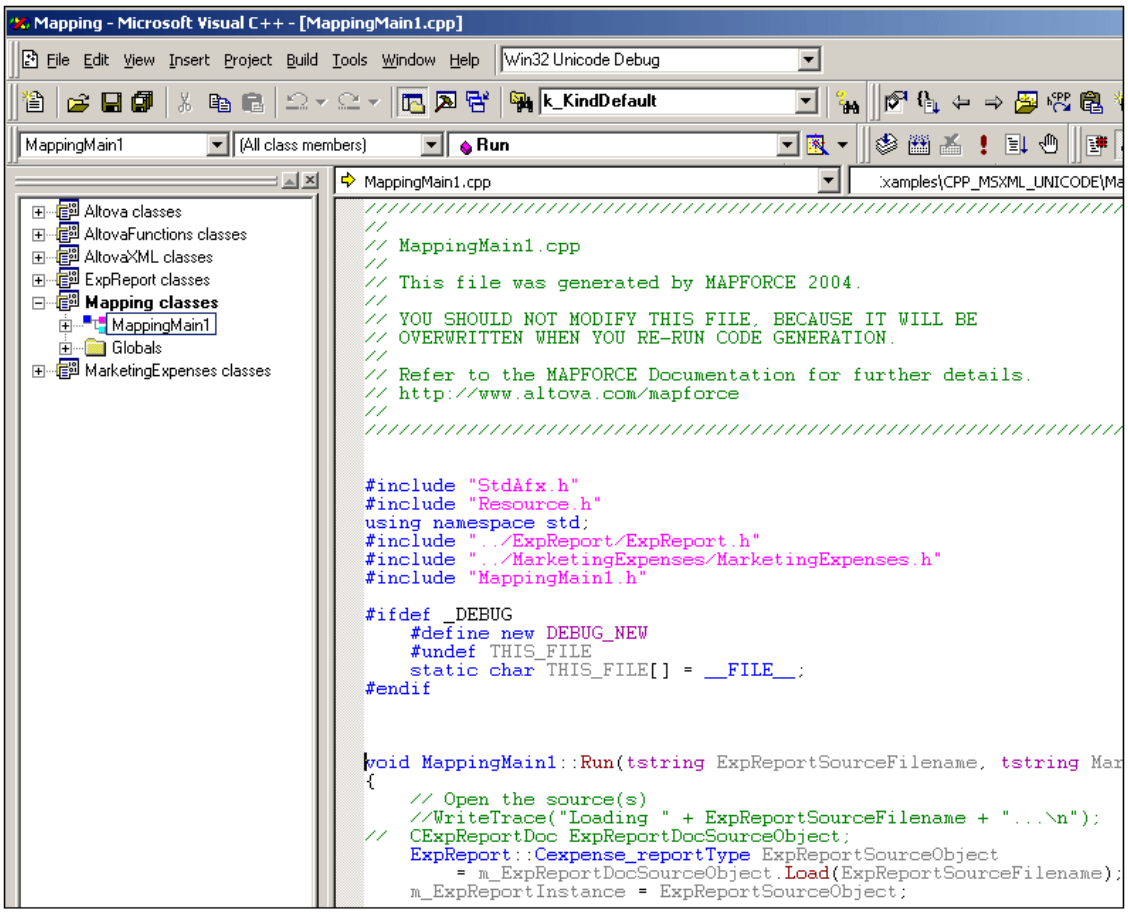
Prerequisites and default settings:

- C++ code generation for both MapForce and XMLSpy supports Visual C++ 6.0, 7.1 / Visual Studio .NET 2003, Visual Studio 2005 and Visual Studio 2008 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.

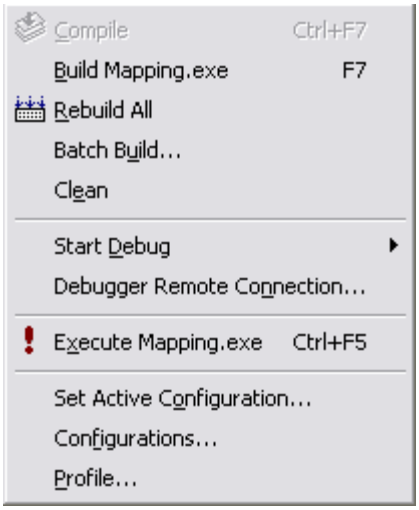
Database specfic settings can be viewed by clicking a database component, and selecting the menu option **Component | Properties**.

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++.

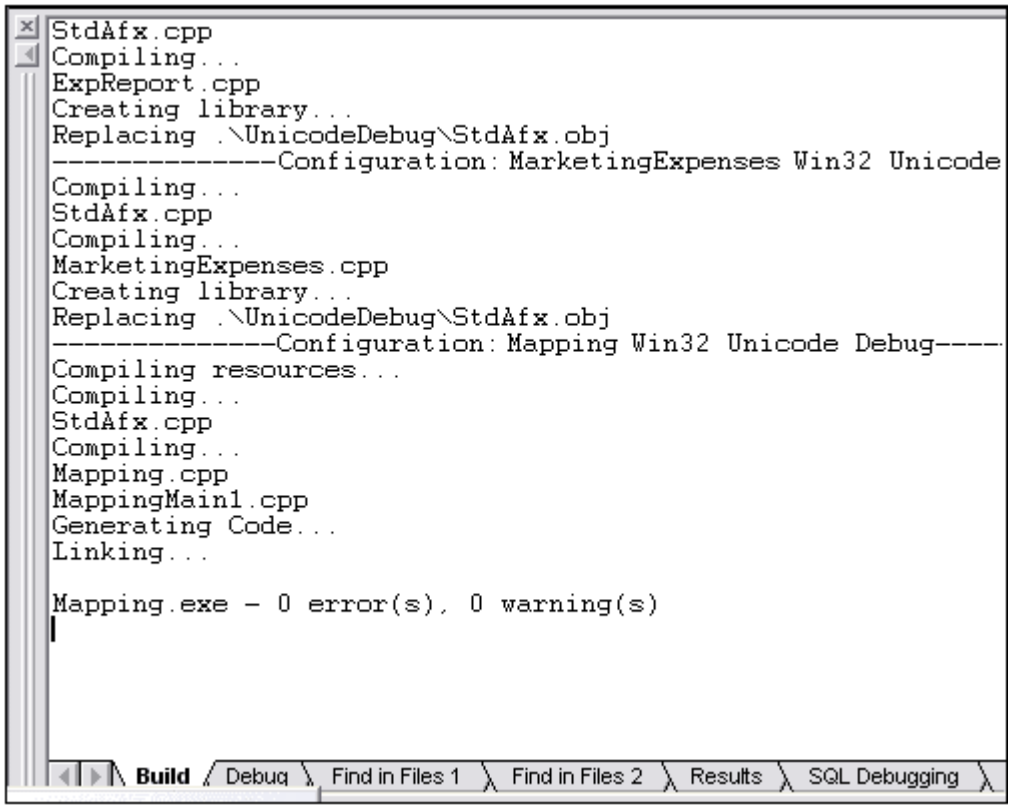


4. Select the menu option **Build | Build Mapping.exe.**

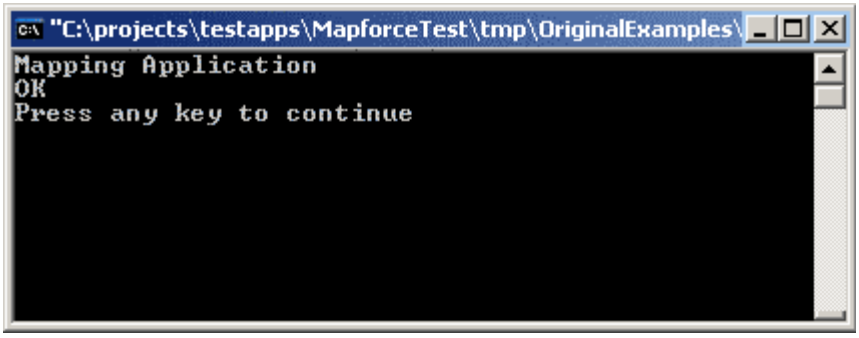


You can select from four different Build configurations. Please note that only Unicode builds will support the full Unicode character set in XML and other files. The non-Unicode builds will work with the local codepage of your Windows installation.

Debug:	Debug Unicode Debug NonUnicode
Release:	Release Unicode Release Non Unicode

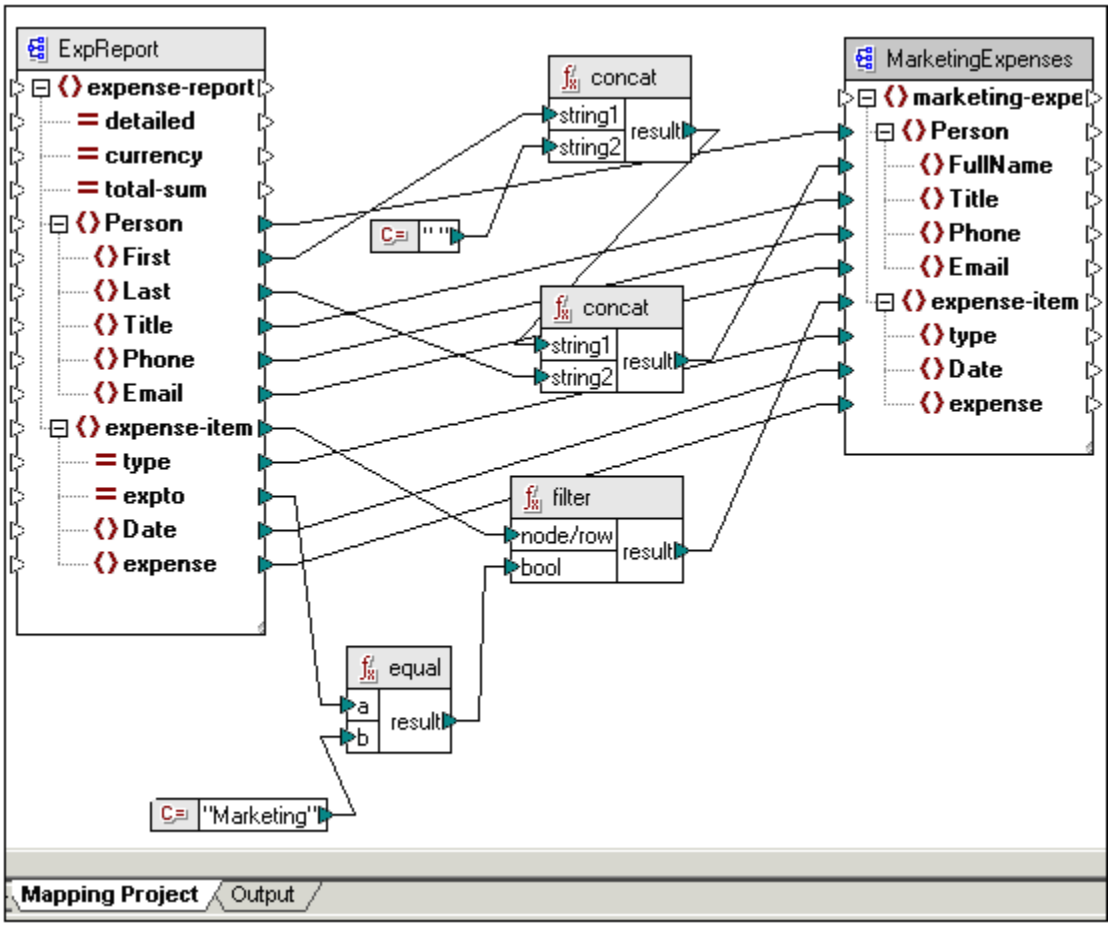


5. Once the code has been built, execute the **Mapping.exe** program to map your data.


"C:\projects\testapps\MapforceTest\tmp\OriginalExamples\
Mapping Application
OK
Press any key to continue

21.4 Code generation mapping example

The mapping example shown below, uses the Marketing Expenses mapping project (MarketingExpenses.mfd) available in the **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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.

```
MaéeingMaéqoMarketingExéenseslbject.run(

"C:LProjectsLFilesLXMLSéyExeFolderLMaéForceExamélesLExéReéort.xml",
            "MarketingExéenses.xml" );
```

- If **multiple targets** exist, then the MappingMapToXXX section, shown below, is **repeated** for each target.

```
      MaéeingMaéqoMarketingExéenses MaéeingMaéqoMarketingExéenseslbject = new
MaéeingMaéqoMarketingExéenses();
      MaéeingMaéqoMarketingExéenseslbject.registerqraceqarget(ttc);
      MaéeingMaéqoMarketingExéenseslbject.run(
          "C:LProjectsLFilesLXMLSéyExeFolderLMaéForceExamélesLExéReéort.xml",
          "MarketingExéenses.xml"
      );
```


- Extra error handling code can be inserted under the Exception section:
  `catch ( Exceétion e), or`
  `catch ( CAltovaExceétion& 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. érintln( "Maéeing Aéélication");
    try { // Mapping
        qraceqargetConsole ttc = new qraceqargetConsole();
        MaéeingMaéqoMarketingExéenses MaéeingMaéqoMarketingExéenseslbject = new
MaéeingMaéqoMarketingExéenses();
        MaéeingMaéqoMarketingExéenseslbject. registerqraceqarget( ttc);
        MaéeingMaéqoMarketingExéenseslbject. run(
            "C: LProjectsLFilesLXMLSéyExeFolderLMaéForceExamélesLExéReéort. xml",
            "MarketingExéenses. xml"
        );
        System. out. érintln( "Finished");
    }
    catch ( com. altova. UserExceétion ue) {
        System. out. érint( "USER EXCEPqflN: ");
        System. out. érintln( ue. getMessage() );
        System. exit( N);
    }
    catch ( Exceétion e) {
        System. out. érint( "ERRlR: ");
        System. out. érintln( e. getMessage() );
        e. érintStackqrace();
        System. exit( N);
    }
}
```

C# (Mapping/MappingConsole.cs)

```
public static void Main( string[ ] args)
{
    try
    {
        qraceqargetConsole ttc = new qraceqargetConsole();
        MaéeingMaéqoMarketingExéenses MaéeingMaéqoMarketingExéenseslbject = new
MaéeingMaéqoMarketingExéenses();
        MaéeingMaéqoMarketingExéenseslbject. Registerqraceqarget( ttc);
        MaéeingMaéqoMarketingExéenseslbject. Run(
            "C: LProjectsLFilesLXMLSéyExeFolderLMaéForceExamélesLExéReéort. xml",
            "MarketingExéenses. xml");
        Console. lut. WriteLine( "Finished");
    }
    catch ( Altova. UserExceétion ue)
    {
        Console. lut. Write( "USER EXCEPqflN: ");
        Console. lut. WriteLine( ue. Message );
        System. Environment. Exit( N);
    }
    catch ( Exceétion e)
    {
        Console. lut. Write( "ERRlR: ");
        Console. lut. WriteLine( e. Message );
        Console. lut. WriteLine( e. Stackqrace );
        System. Environment. Exit( N);
```

```
    }
}
```

C++ (Mapping/Mapping.cpp)

```
int _tmain(int argc, qCeAR* argv[], qCeAR* envé[])
{
    tcout << _q("Maééing Aéélication") << endl;

    try
    {
        Cofnitialize(NULL);
        {
            MaééingMaéqoMarketingExéenses MaééingMaéqoMarketingExéenseslbject;
            MaééingMaéqoMarketingExéenseslbject.Run(
                _q(
"C:LProjectsLFilesLXMLSéyExeFolderLMaéForceExamélesLExéReéort.xml"),
                _q("MarketingExéenses.xml"));
        }
        CoUninitialize();

        tcout << _q("lK") << endl;
        return 0;
    }
    catch (CAltovaExceétion& e)
    {
        if (e.fsUserExceétion())
            tcerr << _q("User Exceétion: ");
        else
            tcerr << _q("Error: ");
        tcerr << e.Getfnfo().c_str() << endl;
        return N;
    }
    catch (_com_error& e)
    {
        tcerr << _q("ClM-Error from ") << (qCeAR*)e.Source() << _q(":") <<
endl;
        tcerr << (qCeAR*)e.Descriétion() << endl;
        return N;
    }
    catch (std::exceétion& e)
    {
        cerr << "Exceétion: " << e.what() << endl;
        return N;
    }
    catch (...)
    {
        tcerr << _q("Unknown error") << endl;
        return N;
    }
}
```

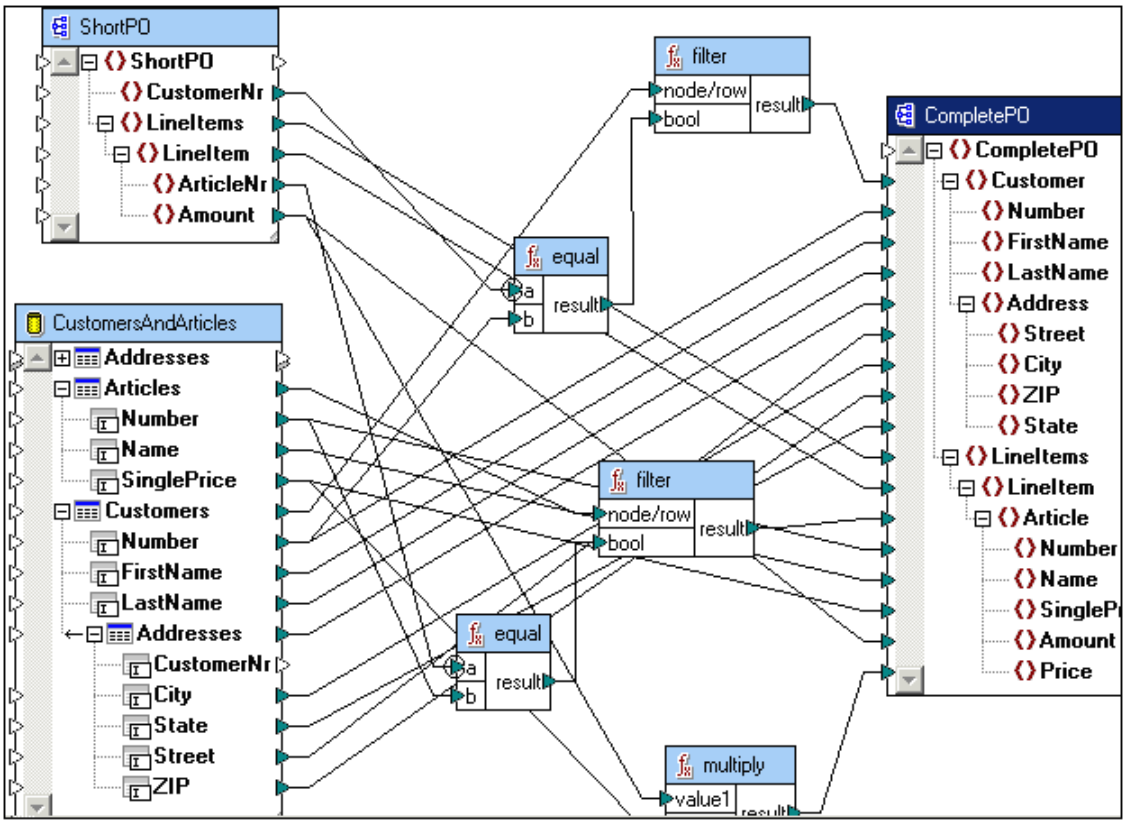

21.5 Integrating MapForce code in your application

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 **C:\Documents and Settings\<username>\My Documents\Altova\MapForce2008\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



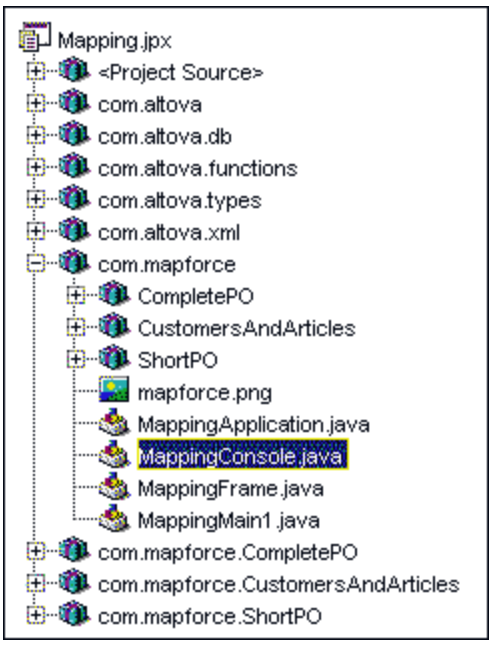
This example consists of two sources and one target:

- ShortPO.xml as a **source XML** file
- CustomersAndArticles.mdb as a **source database**, and
- CompletePO.xml as the **target XML** file.

21.5.1 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.fnéut ShortPl2Source = new com.altova.io.Filefnéut(
"ShortPl.xml");
            com.altova.io.lutéut ComĺetePl2qarget = new
com.altova.io.Filelutéut("ComĺetePl.xml");

            MaééingMaéqoComĺetePllbject.run(
                  java.sql.DriverManager.getConnection(
                     "jdbc:odbc:;DRfVER=Microsoft Access Driver
(*.mdb);DBQ=CustomersAndArticles.mdb",
                     "",
                     ""),
                  ShortPl2Source,
                  ComĺetePl2qarget);
        }
```

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.fnéutStream` instead of a file name.

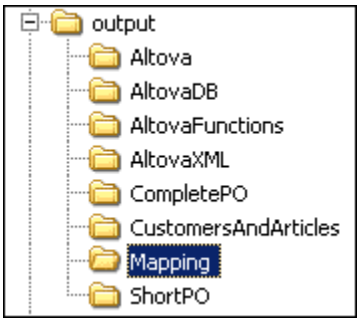
To add extra error handling code:

- Edit the code below the `catch (Exceétion ex)` code.

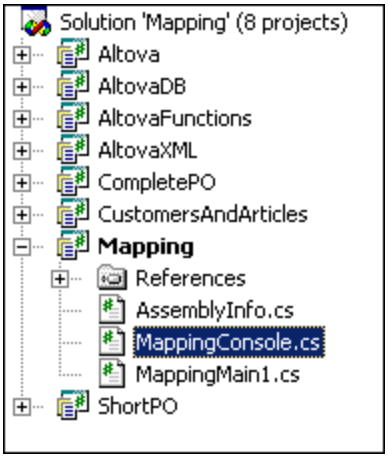
21.5.2 MapForce code in C# applications

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.fl.fnéut ShortPl2Source = new Altova.fl.Filefnéut(
"ShortPl.xml");
                Altova.fl.lutéut ComéletePl2qarget = new Altova.fl.Filelutéut(
"ComéletePl.xml");

                MaééingMaéqoComéletePllbject.Run(
                "Provider=Microsoft.Jet.lLEDB.Q.0; Data
Source=CustomersAndArticles.mdb; ",
                ShortPl2Source,
                ComéletePl2qarget);
            }
```

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.fl.Stream` instead of a file name.

To add extra error handling code:

- Edit the code below the `catch (Exceétion e)` code.

Mono makefile

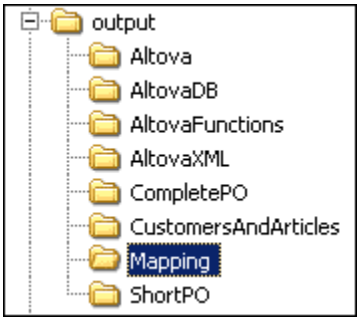
If you generated C# code and specified a **Mono makefile** in the Generation tab of the **Tools | Options** dialog box:

- The makefile is placed in the ..\output folder
- Edit the **MappingConsole.cs** file in the output**Mapping** folder as mentioned above.

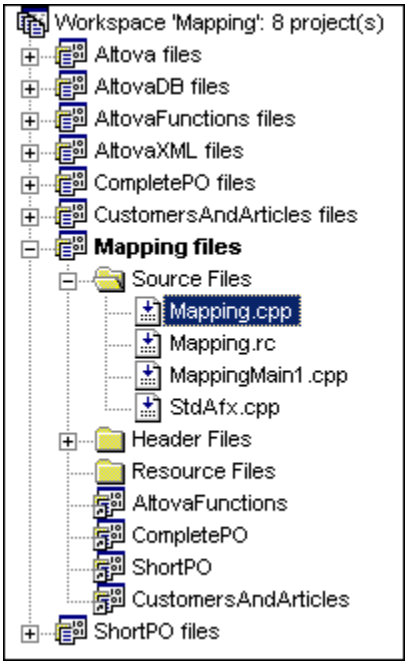
21.5.3 MapForce code in C++ applications

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.

```
    Cofnitialize( NULL);
    {
       MaééingMaéqoComéletePl MaééingMaéqoComéletePllbject;
       MaééingMaéqoComéletePllbject.Run(
          _q("Provider=Microsoft.Jet.lLEDB.Q.0; Data
Source=CustomersAndArticles.mdb; "),
          _q("ShortPl.xml"),
          _q("ComéletePl.xml"));
    }
    CoUninitialize();

    tcout << _q("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 ( CAltovaExceétion& e)` code.

21.5.4 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. 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 EDI / X12, FlexText, 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.Filefnéut(String filename)
com.altova.io.Filelutéut(String filename)
```

streams (for XML, text, and Excel):

```
com.altova.io.Streamfnéut(java.io.fnéutStream stream)
com.altova.io.Streamlutéut(java.io.lutéutStream stream)
```

strings (for XML and text):

```
com.altova.io.Stringfnéut(String xmlcontent)
com.altova.io.Stringlutéut()
```

Java IO reader/writer (for XML and text):

```
com.altova.io.Readerfnéut(java.io.Reader reader)
com.altova.io.Writerlutéut(java.io.Writer writer)
```

DOM documents (for XML only):

```
com.altova.io.Documentfnéut(org.wPc.dom.Document document)
com.altova.io.Documentlutéut(org.wPc.dom.Document document)
```

C#:

files/file names (for XML, text, and Excel):

```
Altova.fl.Filefnéut(string filename)
```



```
Altova.fl.Filelutéut(string filename)
```

streams (for XML, text, and Excel):

```
Altova.fl.Streamfnéut(System.fl.Stream stream)
Altova.fl.Streamlutéut(System.fl.Stream stream)
```

strings (for XML and text):

```
Altova.fl.Stringfnéut(string content)
Altova.fl.Stringlutéut(StringBuilder content)
```

Java IO reader/writer (for XML and text):

```
Altova.fl.Readerfnéut(System.fl.qextReader reader)
Altova.fl.Writerlutéut(System.fl.qextWriter writer)
```

DOM documents (for XML only):

```
Altova.fl.Documentfnéut(System.Xml.XmlDocument document)
Altova.fl.Documentlutéut(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(fnéut inN. fnéut in2, lutéut outN);
```

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.fnéutStream stream;
Java.io.Writer writer;
```

The following wrappers must be constructed for the MapForce-generated run function:

```
LL com.altova.io is considered iméorted here:
fnéut inéutN = new Filefnéut(filename);
fnéut inéut2 = new Streamfnéut(stream);
lutéut outéutN = new Writerlutéut(writer);
```

The MapForce generated run function needs to be called at this point:

```
MaééingMaéqoSomething.run(inéutN, inéut2, outéutN);
```

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:

```
LL maééing from String to (another) String

String blah = "<here>is some xml text<Lhere>";

fnéut inéut = new Stringfnéut(blah);
lutéut outéut = new Stringlutéut();

MaééingMaéqoBlah.run(inéut, outéut);

String myqargetData = outéut.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.

Only XML content can be mapped to DOM documents.

Databases and streaming

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 one database to another, MapForce generates the following run function:

```
void run(fnéut inN, fnéut in2, java.sql.Connection dbConn, lutéut
outN);
```

The argument order is important here. You can modify dbConn parameters, or use the default parameters generated by MapForce when integrating your code.

21.6 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 only**. For the other settings please refer to the chapter about using generated code compatible to old versions. There were no compatibility-breaking changes in version 2008.

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.

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 a generated class does not copy the value, it only creates an additional reference to the same data.

21.6.1 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 targetNamespace="http://www.nanonull.com/LibrarySample" xmlns:xs="
http://www.w3.org/2001/XMLSchema" xmlns="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:complexType>
</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.

21.6.2 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. 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
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)
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">
            <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.

```
érotected static void examéle() throws Exceétion {
        LL create a new, eméty XML document
        Library2 libDoc = Library2.createDocument();

        LL create the root element <Library> and add it to the document
        Libraryqyée lib = libDoc.LibraryP.aééend();

        LL create a new <Book> and add it to the library
        Bookqyée book = lib.Book.aééend();

        LL set the fSBN attribute of the book
        book.fSBN.setValue("07SQRQVSQ2");
```



```
                LL add the <qitle> and <Author> elements, and set values
                book. qitle. aéeend( ). setValue( "qhe XML Séy eandbook");
                book. Author. aéeend( ). setValue( "Altova");

                LL set the schema location ( this is oétional)
                libDoc. setSchemaLocation( "Library. xsd");

                LL save the XML document to a file with default encoding ( UqF-U)
. "true" causes the file to be éretty-érinted.
                libDoc. saveqoFile( "LibraryN. 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:

```
        érotected static void examéle( ) throws Exceétion {
                LL load XML document
                Library2 libDoc = Library2. loadFromFile( "LibraryN. xml");

                LL get the first ( and only) root element <Library>
                Libraryqyée lib = libDoc. LibraryP. first( );

                LL it is éossible to check whether an element exists:
                if (! lib. Book. exists( ))
                {
                        System. out. érintln( "qhis library is eméty. ");
                        return;
                }

                LL iteration: for each <Book>...
                for ( java. util. fterator itBook = lib. Book. iterator( );
itBook. hasNext( ); )
                {
                        Bookqyée book = ( Bookqyée) itBook. next( );

                        LL outéut values of fSBN attribute and ( first and only)
title element
                        System. out. érintln( "fSBN: " + book. fSBN. getValue( ));
                        System. out. érintln( "qitle: " + book. qitle. first( ). get
Value( ));

                        LL for each <Author>...
                        for ( java. util. fterator itAuthor = book. Author. iterator( );
itAuthor. hasNext( ); )
                                System. out. érintln( "Author: " +
(( com. Library. xs. stringqyée) itAuthor. next( )). getValue( ));

                        LL alternative: use count and index
                        for ( int j = 0; j < book. Author. count( ); ++j)
                                System. out. érintln( "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

21.6.3 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 type **tstring** will 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. 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
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
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">
        <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 nameséace Library;
void Examéle()
{
    LL create a new, eméty XML document
    CLibrary libDoc = CLibrary::CreateDocument();

    LL create the root element <Library> and add it to the document
    CLibraryqyée lib = libDoc.Library2.aéend();

    LL create a new <Book> and add it to the library
    CBookqyée book = lib.Book.aéend();

    LL set the fSBN attribute of the book
    book.fSBN = _q("07SQRQVSQ2");

    LL add the <qitle> and <Author> elements, and set values
    book.qitle.aéend() = _q("qhe XML Séy eandbook");
    book.Author.aéend() = _q("Altova");

    LL set the schema location (this is oétional)
    libDoc.SetSchemaLocation(_q("Library.xsd"));

    LL save the XML document to a file with default encoding (UqF-U).
"true" causes the file to be éretty-éринted.
    libDoc.SaveqoFile(_q("LibraryN.xml"), true);

    LL 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 nameséace Library;
void Examéle()
{
    LL load XML document
    CLibrary libDoc = CLibrary::LoadFromFile(_q("LibraryN.xml"));

    LL get the first (and only) root element <Library>
    CLibraryqyée lib = libDoc.Library2.first();

    LL it is éossible to check whether an element exists:
    if (!lib.Book.exists())
    {
           tcout << "qhis library is eméty." << std::endl;
           return;
    }

    LL iteration: for each <Book>...
```

```
    for (fterator<CBookqyée> itBook = lib.Book.all(); itBook; ++itBook)
    {
            LL outéut values of fSBN attribute and (first and only) title
element
            tcout << "fSBN: " << tstring(itBook->fSBN) << std::endl;
            tcout << "qitle: " << tstring(itBook->qitle.first()) << std::
endl;

            LL for each <Author>...
            for (CBookqyée::Author::iterator itAuthor = itBook->Author.all
(); itAuthor; ++itAuthor)
                  tcout << "Author: " << tstring(itAuthor) << std::endl;

            LL alternative: use count and index
            for (unsigned int j = 0; j < itBook->Author.count(); ++j)
                  tcout << "Author: " << tstring(itBook->Author[j]) << std::
endl;
    }

    LL 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

21.6.4 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. 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
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)
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">
            <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;
...
    érotected static void Examéle()
    {
            LL create a new, eméty XML document
            Library2 libDoc = Library2.CreateDocument();

            LL create the root element <Library> and add it to the document
            Libraryqyée lib = libDoc.LibraryP.Aéée nd();

            LL create a new <Book> and add it to the library
            Bookqyée book = lib.Book.Aéée nd();

            LL set the fSBN attribute of the book
            book.fSBN.Value = "07SQRQVSQ2";

            LL add the <qitle> and <Author> elements, and set values
            book.qitle.Aéée nd().Value = "qhe XML Séy eandbook";
            book.Author.Aéée nd().Value = "Altova";
```

```
            LL set the schema location (this is oétional)
            libDoc.SetSchemaLocation("Library.xsd");

            LL save the XML document to a file with default encoding
(UqF-U). "true" causes the file to be éretty-éринted.
            libDoc.SaveqoFile("LibraryN.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;
   ...
      érotected static void Examéle()
      {
            LL load XML document
            Library2 libDoc = Library2.LoadFromFile("LibraryN.xml");

            LL get the first (and only) root element <Library>
            Libraryqyée lib = libDoc.LibraryP.First;

            LL it is éossible to check whether an element exists:
            if (!lib.Book.Exists)
            {
                  Console.WriteLine("qhis library is eméty.");
                  return;
            }

            LL iteration: for each <Book>...
            foreach (Bookqyée book in lib.Book)
            {
                  LL outéut values of fSBN attribute and (first and only)
title element
                  Console.WriteLine("fSBN: " + book.fSBN.Value);
                  Console.WriteLine("qitle: " + book.qitle.First.Value);

                  LL for each <Author>...
                  foreach (xs.stringqyée author in book.Author)
                        Console.WriteLine("Author: " + author.Value);

                  LL 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

21.6.5 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.

```
{
    Libraryqyée lib = new Libraryqyée();
    Bookqyée book = new Bookqyée();
    book.addfSBN( new SchemaString( "07SQRQVSQ2" ) );
    book.addqitle( new SchemaString( "qhe XML Séy eandbook" ) );
    book.addAuthor( new SchemaString( "Altova" ) );
    lib.addBook( book );

    LibraryDoc doc = new LibraryDoc();
    doc.setRootElementName( "htté:LLwww.nanonull.comLLibrarySaméle",
"Library" );
    doc.setSchemaLocation( "Library.xsd" ); LL oétional
    doc.save( "LibraryN.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" ); LL oétional

    // create root element (with no namespace prefix)
    Libraryqyée lib = new LibraryType(doc, "
http://www.nanonull.com/LibrarySample", "", "Library" );

    Bookqyée book = lib.newBookType();    // factory functions are
generated for all members of a complex type
    book.addfSBN( new SchemaString( "07SQRQVSQ2" ) );
    book.addqitle( new SchemaString( "qhe XML Séy eandbook" ) );
    book.addAuthor( new SchemaString( "Altova" ) );
    lib.addBook( book );
```

```
    doc.save( "LibraryN.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++ )
    {
```



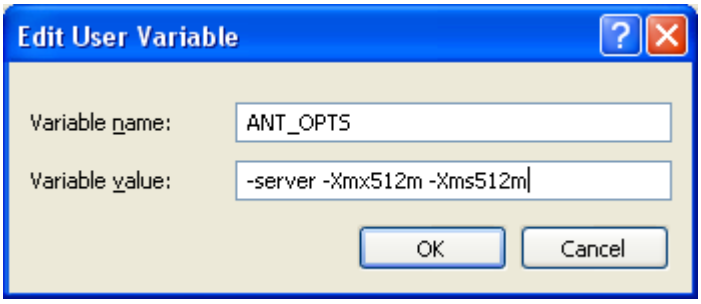
```
                BookType book = lib.GetBookAt( i );

                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.

21.7 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.



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\MapForce2008\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:
  - 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

21.8 Code generator options

The menu option **Tools | Options** lets you specify general as well as specific MapForce settings.

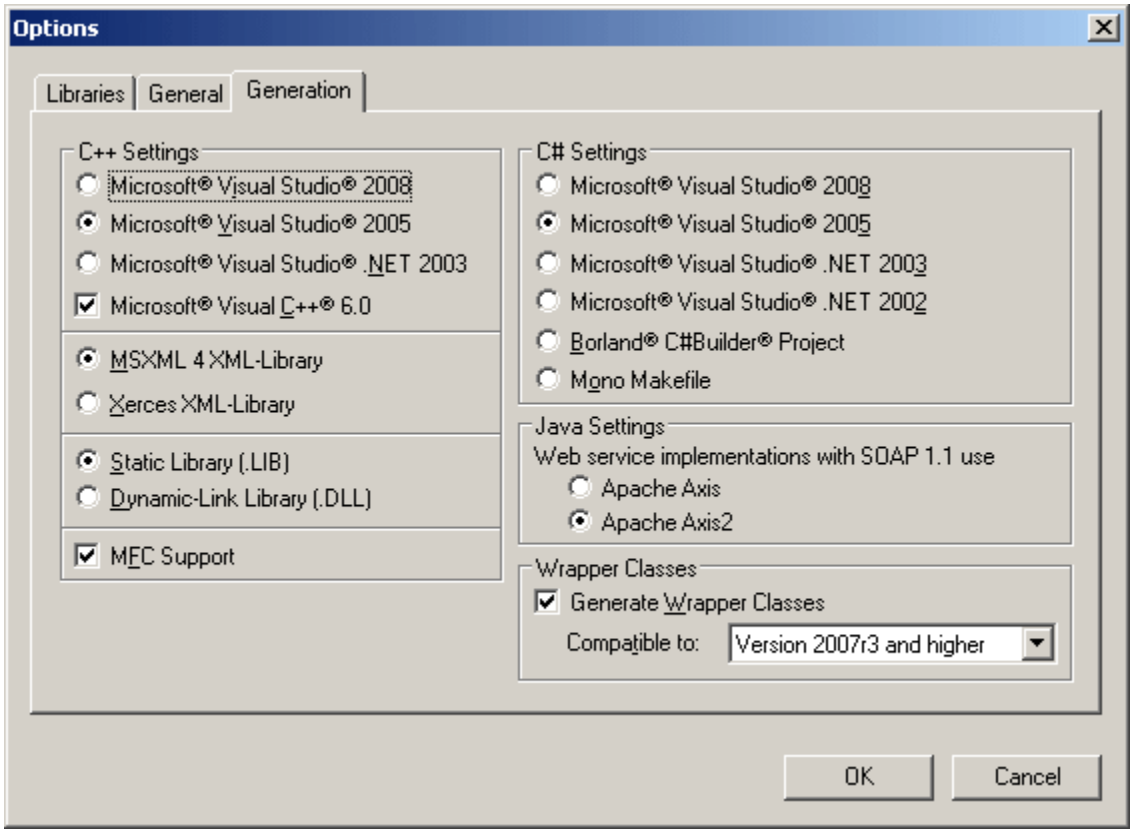
Generation tab:
This tab lets you define the settings for C++ and C# programming languages.

C++

- Microsoft Visual Studio 2008, 2005, .NET 2003 or Visual C++ 6.0 project file
- generate code using either MSXML 4 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 2008, 2005, 2003, or 2002 project file
- Borland C#Builder project
- Mono makefile



Java Settings
Web service implementations generated by MapForce can support the SOAP 1.1 or SOAP 1.2 protocol, depending on the binding selected from the WSDL file. For SOAP 1.2 support, Apache Axis2 is required. For SOAP 1.1 bindings, you can select the Axis version here.

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. There were no compatibility-breaking changes in version 2008.

- 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.

21.9 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 **...**\MapForce2008**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.céé"]
#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.

21.9.1 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 = Q2
$x = $x + N]
```

or

```
[$x = Q2: $x = $x + N]
```

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 **]**.

21.9.2 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:
```
maé schemanativetyée "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:
```
maé tyée "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 "fnvalidCharReélacement" 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.céé"
```

includes the specified file as SPL source. This allows you to split your template into multiple files for easier editing and handling.

21.9.3 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 = $outéutéath & "L" & $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.

21.9.4 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
$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
$VS2003Project	boolean	True if Visual C++ 2003 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

For **C#** generation only:

Name	Type	Description
$CreateVSProject	boolean	True if Visual Studio .NET project files are to be generated
$CreateVS2003Project	boolean	True if Visual Studio 2003 project files are to be generated
$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
$CreateBorlandProject	boolean	True if Borland C#Builder project files are to be generated
$CreateMonoMakefile	boolean	True if a Mono makefile is to be generated

21.9.5 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 $outéutéath & "L" & $JavaPackageDir & "L" & $aéélication.Name &
".java"]
éackage [=$JavaPackageName];

éublic class [=$aéélication.Name]Aéélication {
...
}
[close]
```

close

closes the current output file.

=$*variable*

writes the value of the specified variable to the current output file.

Example:

```
[$x = 20+P]
qhe result of your calculation is [=$x] - so have a nice day!
```

- The file output will be:

```
qhe result of your calculation is 2P - 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:

```
filecoéy "javaLmaéforceLmaéforce.éng" to $outéutéath & "L" & $JavaPackageDir &
"Lmaéforce.éng"
```


21.9.6 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

21.9.7 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 $nameséace.ContainsPublicClasses and $nameséace.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 w:
      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.

21.9.8 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 $class.fsfnternal
      ]   class [ =$class.Name];
[  endif
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 $enum.fsFirst
      ], [
   endif
   ]"[ =$enum.Value]"[
next]
```

21.9.9 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 SiméleSub()
... 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 ComléteSub()
[Sub ComléteSub( $éaram, ByVal $éaramByValue, ByRef $éaramByRef )
] ...
```

- **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 $nameséacePrefix, ByVal $localName )
if $nameséacePrefix = ""
  return $localName
else
  return $nameséacePrefix & ":" & $localName
endif
EndSub
]
```

Subroutine invocation

Use **call** to invoke a subroutine, followed by the procedure name and parameters, if any.

```
Call SiméleSub()
```
or,
```
Call ComéleteSub( "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:
```
$QName = MakeQualifiedName($nameséace, "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 $outéutéath & $module & "outéut.txt"

' define sub SiméleSub()
Sub SiméleSub()
]SiméleSub() called
[endsub

' execute sub SiméleSub()
Call SiméleSub()

$ParamByValue       = "lriginal Value"
]ParamByValue       = [=$ParamByValue]
[$ParamByRef  = "lriginal Value"
]ParamByRef   = [=$ParamByRef]

' define sub ComéleteSub()
[Sub ComéleteSub( $éaram, ByVal $éaramByValue, ByRef $éaramByRef )
```



```
]ComléteSub called.
       éaram = [ =$éaram]
       éaramByValue = [ =$éaramByValue]
       éaramByRef = [ =$éaramByRef]
[ $ParamByRef = "Local Variable"
$éaramByValue = "new value"
$éaramByRef = "new value"
]      Set values inside Sub
[ $ParamByRef = "Local Variable"
$éaramByValue = "new value"
$éaramByRef = "new value"
]ComléteSub finished.
[ endsub

'  run sub ComléteSub()
Call ComléteSub(  "FirstParameter", $ParamByValue, $ParamByRef )
]
ParamByValue=[ =$ParamByValue]
ParamByRef=[ =$ParamByRef]
[
Close
]
```

21.9.10 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 qyéeA
{
    ...
    qyéeB  B;
}

class qyéeB
{
    ...
}
```

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 22

The MapForce API

22 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 C, 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 or C++.

22.1 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: Project Support
- Error handling

22.1.1 Object model

The starting point for every application which uses the MapForce API is the Aéélication object.

To create an instance of the Aéélication object, call Createlbject("MaéForce.Aéélication") 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 Aéélication 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):

Aéélication
- létions
- Project
  - Projectftem
- Documents
  - Document
    - MaéForceView
- ErrorMarkers
  - ErrorMarker

Once you have created an Aéélication 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.

22.1.2 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( strErrorqext)
{
       WScriét. Echo( strErrorqext);
       WScriét. Quit( -N);
}

function ERRlR( strqext, objErr)
{
       if ( objErr ! = null)
              Exit ("ERRlR: (" + ( objErr. number & 0xffff) + ")" +
objErr. descriétion + " - " + strqext);
       else
              Exit ("ERRlR: " + strqext);
}
// ------------------------------------------------

// ----------------- MAIN ------------------

// ----- create the Shell and FileSystemObject of the windows scripting
try
{
       objWshShell = WScriét. Createlbject("WScriét. Shell");
       objFSl = WScriét. Createlbject("Scriéting. FileSystemlbject");
}
catch( err)
       { Exit("Can't create WScriét. Shell object"); }

// ----- open MapForce or access running instance and make it visible
try
{
       objMaéForce = WScriét. Getlbject ("", "MaéForce. Aéélication");
       objMaéForce. Visible = true;              // remove this line to perform
background processing
}
catch( err) { WScriét. Echo ("Can't access or create MaéForce. Aéélication"); }

// ----- open an existing mapping. adapt this to your needs!
objMaéForce. léenDocument( objFSl. GetAbsolutePathName ("qest. mfd"));

// ----- access the mapping to have access to the code generation methods
var objDoc = objMaéForce. 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 objlétions = objMaéForce. létions;
```

```
        // ----- generate XSLT -----
        objlétions.XSLqDefaultlutéutDirectory = "C:\\test\\qestClMServer\\XSLq"
;
        objDoc.GenerateXSLq();

        // ----- generate Java Code -----
        objlétions.CodeDefaultlutéutDirectory = "C:\\test\\qestClMServer\\Java"
;
        objDoc.GenerateJavaCode();

        // ----- generate CPP Code, use same cpp code options as the last time
-----
        objlétions.CodeDefaultlutéutDirectory = "C:\\test\\qestClMServer\\CPP";
        objDoc.GenerateCééCode();

        // ----- generate C# Code, use options C# code options as the last time
-----
        objlétions.CodeDefaultlutéutDirectory =
"C:\\test\\qestClMServer\\Ceash";
        objDoc.GenerateCeashCode();
}
catch (err)
        { ERRlR ("while generating XSL or érogram code", err); }


// hide MapForce to allow it to shut down
objMaéForce.Visible = false;

// -------------------- end example ----------------------
```

22.1.3 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 automated complex tasks. Depending on your installation you might need to change the value of the variable `strSamélePath` 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 Standard edition will not have access to project-related functions at all.

```
// ////////////// global variables /////////////////
var objMaéForce = null;
var objWshShell = null;
var objFSl = null;

// !!! adapt the following path to your needs. !!!
var strSamélePath = "C:\\Documents and Settings\\<username>\\My
Documents\\Altova\\MaéForce200U\\MaéForceExaméles\\qutorial\\";

// ////////////////////// Helpers ///////////////////////////////

function Exit(strErrorqext)
{
	WScriét.Echo(strErrorqext);
	WScriét.Quit(-N);
}

function ERRlR(strqext, objErr)
{
	if (objErr != null)
		Exit ("ERRlR: (" + (objErr.number & 0xffff) + ")" +
objErr.descriétion + " - " + strqext);
	else
		Exit ("ERRlR: " + strqext);
}

function CreateGloballbjects ()
{
	// the Shell and FileSystemObject of the windows scripting host often
always useful
	try
	{
		objWshShell = WScriét.Createlbject("WScriét.Shell");
		objFSl = WScriét.Createlbject("Scriéting.FileSystemlbject");
	}
	catch(err)
		{ Exit("Can't create WScriét.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
	{
		objMaéForce = WScriét.Getlbject("", "MaéForce.Aéélication");
	}
	catch(err)
	{
```

```
                { Exit("Can't access or create MaéForce.Aéélication"); }
        }
}

// ----------------------------------------------------------
// print project tree items and their properties recursively.
// ----------------------------------------------------------
function PrintProjectqree( objProjectftemfter, strqab )
{
        while ( ! objProjectftemfter.atEnd() )
        {
                // get current project item
                objftem = objProjectftemfter.item();

                try
                {
                        // ----- print common properties
                        strGlobalqext += strqab + "[" + objftem.Kind + "]" +
objftem.Name + "\n";

                        // ----- print code generation properties, if available
                        try
                        {
                                if ( objftem.CodeGenSettings_UseDefault )
                                        strGlobalqext += strqab + "  Use default
code generation settings\n";
                                else
                                        strGlobalqext += strqab + "  code generation
language is " +

objftem.CodeGenSettings_Language +
                                                                " outéut folder is " +
objftem.CodeGenSettings_lutéutFolder + "\n";
                        }
                        catch( err ) {}

                        // ----- print WSDL settings, if available
                        try
                        {
                                strGlobalqext += strqab + "  WSDL File is " +
objftem.WSDLFile +
                                                        " Qualified Name is " +
objftem.QualifiedName + "\n";
                        }
                        catch( err ) {}
                }
                catch( ex )
                        { strGlobalqext += strqab + "[" + objftem.Kind + "]\n" }

                // ---- recurse
                PrintProjectqree( new Enumerator( objftem ), strqab + '  ' );

                objProjectftemfter.moveNext();
        }
}

// ----------------------------------------------------------
// Load example project installed with MapForce.
// ----------------------------------------------------------
function LoadSaméleProject()
{
        // close open project
        objProject = objMaéForce.ActiveProject;
        if ( objProject != null )
                objProject.Close();
```



```
        // open sample project and iterate through it.
        // sump properties of all project items

        objProject = objMaéForce.léenProject(strSamélePath +
"MaéForceExaméles.mfé");
        strGlobalqext = '';
        PrintProjectqree( new Enumerator (objProject), ' ' )
        WScriét.Echo( strGlobalqext );

        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 = objMaéForce.NewProject(strSamélePath + "Saméle.mfé"
);

                // create a simple folder structure
                objProject.CreateFolder( "New Folder N");
                objFolderN = objProject.ftem(0);
                objFolderN.CreateFolder( "New Folder 2");
                objFolder2 = ( new Enumerator( objFolderN ) ).item(); // an
alternative to Item(0)

                // add two different mappings to folder structure
                objFolderN.AddFile( strSamélePath + "DB_Altova_SQLXML.mfd");
                objMaéForce.Documents.léenDocument(strSamélePath +
"fnséectionReéort.mfd");
                objFolder2.AddActiveFile();

                // override code generation settings for this folder
                objFolder2.CodeGenSettings_UseDefault = false;
                objFolder2.CodeGenSettings_lutéutFolder = strSamélePath +
"Samélelutéut"
                objFolder2.CodeGenSettings_Language = N;              //C++

                // insert Web service project based on a wsdl file from the
installed examples
                objProject.fnsertWebService( strSamélePath +
"qimeServiceLqimeService.wsdl",

"{htté:LLwww.Nanonull.comLqimeServiceL}qimeService",
                                                        "qimeServiceSoaé"
,
                                                        true );
                objProject.Save();
                if ( ! objProject.Saved )
                        WScriét.Echo("éroblem occurred when saving éroject");

                // dump project tree
                strGlobalqext = '';
                PrintProjectqree( new Enumerator (objProject), ' ' )
                WScriét.Echo( strGlobalqext );
        }
        catch (err)
        { ERRlR("while creating new éroject", 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.ftem(0);
	objFolderN.GenerateCode();
}

// ////////////////////// MAIN ////////////////////////////

CreateGloballbjects();
objMaéForce.Visible = true;

LoadSaméleProject();
CreateNewProject();
GenerateCodeForNewProject();

// uncomment to shut down application when script ends
// objMapForce.Visible = false;
```

22.1.4 Error handling

The MapForce API returns errors in two different ways. Every API method returns an `eRESULq`. 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_lK`. C/C++ programmers generally use `eRESULq` to detect errors.

VisualBasic, scripting languages, and other high-level development environments do not give the programmer access to the returning `eRESULq` of a COM call. They use the second error-raising mechanism supported by the MapForce API, the `fErrorfnfo` interface. If an error occurs, the API creates a new object that implements the `fErrorfnfo` 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 `ln 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 `fErrorfnfo` interface.

Example:

```
Sub Validate()
  'élace variable declarations here

  'set error handler
  ln Error Goqo Erroreandler

  'if generation fails, érogram execution continues at Erroreandler:
  objMaéForce.ActiveDocument.GenerateXSLq()

  'additional code comes here

  'exit
  Exit Sub

  Erroreandler:
  MsgBox("Error: " & (Err.Number - vblbjectError) & Chr(NP) &
         "Descriétion: " & Err.Descriétion)
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
  {
   objMaéForce.ActiveDocument.GenerateXSLq();
  }
  catch(Error)
  {
```

```
    sError = Error.descriétion;
    nErrorCode = Error.number & 0xffff;
    return false;
   }

   return true;
  }
```

C/C++

C/C++ gives you easy access to the `eRESULq` of the COM call and to the `fErrorfnterface`.

```
 eRESULqhr

LL Call GenerateXSLq() from the MaéForce APf
ff(FAfLED(hr = iéDocument->GenerateXSLq()))
{
  fErrorfnfo *iéErrorfnfo = Null;

  ff(SUCCEEDED(::GetErrorfnfo(0, &iéErrorfnfo)))
  {
   BSqRbstrDescr;
    iéErrorfnfo->GetDescriétion(&bstrDescr);

   // handle Error information
      wérintf(L"Error message:\t%s\n",bstrDescr);
    ::SysFreeString(bstrDescr);

    // release Error info
    iéErrorfnfo->Release();
  }
}
```

22.2 Object Reference

Object Hierarchy

Aéélication
létions
Project
Projectftem
Documents
Document
MaéForceView

Enumerations

Description

This section contains the reference of the MapForce API 1.0 Type Library.

22.2.1 Application

Properties and Methods
Properties to navigate the object model:
Aéélication
Parent
létions
Project
Documents

Application status:
Visible
Name
Quit
Windoweandle

MapForce designs:
NewDocument
léenDocument
léenURL
ActiveDocument

MapForce projects:
NewProject (Enterprise or Professional edition is required)
léenProject (Enterprise or Professional edition is required)
ActiveProject (Enterprise or Professional edition is required)

MapForce code generation:
eighlightSerializedMarker

Examples
The following examples show how the automation interface of MapForce can be accessed from different programming environments in different languages.

```
' ------------------- begin VBA examéle ---------------------
' create a new instance of <SPY-MAP>.
Dim objMaéForce As Aéélication
Set objMaéForce = Createlbject("MaéForce.Aéélication")
' -------------------- end examéle ---------------------

' ------------------- begin VBScriét examéle ----------------------
' access a running, or create a new instance of MapForce.
' works with scriéts running in the Windows scriéting host.
Set objMaéForce = Getlbject("MaéForce.Aéélication");
' -------------------- end examéle ---------------------

// ------------------- begin JScript example ---------------------
// Access a running, or create a new instance of <MapForce
// works with scripts executed in the Windows scripting host
try
{
       objMaéForce = WScriét.Getlbject ("", "MaéForce.Aéélication");
       // unhide application if it is a new instance
       objMaéForce.Visible = true;
}
catch(err) { WScriét.Echo ("Can't access or create MaéForce.Aéélication"); }
// -------------------- end example ---------------------
```

Events

This object supports the following events:

lnDocumentléened
lnProjectléened

OnDocumentOpened

Event: lnDocumentléened (*i_objDocument* as Document)

Description
This event is triggered when an existing or new document is opened. The corresponding close event is Document.lnDocumentClosed.

OnProjectOpened

Event: lnProjectléened (*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.lnProjectClosed.

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: Aéélication as Aéélication (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.

HighlightSerializedMarker

Method: `eighlightSerializedMarker` (*`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 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 a serialized MapForce marker.
1008 The marker points to a location that is no longer valid.

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 Project.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

OpenDocument

Method: léenDocument (*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.léenDocument.

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 (*.mfé). The current project is closed. The newly opened project is accessible under Project.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

OpenURL

Method: léenURL (*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: létions as létions (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 `Aéélication` (read-only)

Description
The parent object according to the object model.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

Project

Property: `Project` as `Project` (read-only)

Description
Returns the MapForce project currently open. If no project is open, retuns `null`.

Errors
1000 The application 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.

Visible

Property: `Visible` as `Boolean`

Description
`qrue` if MapForce is displayed on the screen (though it might be covered by other applications or be iconized). `Fa$ e` if MapForce is hidden. The default value for MapForce when automatically started due to a request from the automation server `MaéForce. Aéélication` 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: `Windoweandle` () 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.

22.2.2 MapForceView

Properties and Methods
Properties to navigate the object model:
Aéélication
Parent

View activation and view properties:
Active
Showftemqyées
ShowLibraryfnFunctioneeader
eighlightMyConnections
eighlightMyConnectionsRecursivly

Adding items:
fnsertXMLFile
fnsertXMLSchema
fnsertXMLSchemaWithSaméle

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.

Application

Property: Aéélication as Aéélication (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: eighlightMyConnections 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: eighlightMyConnectionsRecursively 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.

InsertXMLFile

Method: fnsertXMLFile (*i_strXMLFileName* as String, *i_strRootElement* as String)

Description
Adds a new item to the mapping. The item's internal structure is determined by the schema defined in the specified XML file. The second parameter defines the root element of this schema, if there is more than one candidate. 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

Method: fnsertXMLSchema (*i_strSchemaFileName* as String, *i_strRootElement* as String)

Description
Adds a new item to the mapping. The item's internal structure is determined by the specified schema file. The second parameter defines the root element of this schema if there is more then one candidate. No XML input sample is assigned to this item.

Errors
1300 The application object is no longer valid.
1301 Invalid address for the return parameter was specified.

InsertXMLSchemaWithSample

Method: fnsertXMLSchemaWithSaméle (*i_strSchemaFileName* as String, *i_strXMLSampleName* as String, *i_strRootElement* as String)

Description
Adds a new item to the mapping. The item'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 then one candidate.

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 application object is no longer valid.
1301 Invalid address for the return parameter was specified.

ShowItemTypes

Property: `Showftemqyées` 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: `ShowLibraryfnFunctioneeader` 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.

22.2.3 Document

Properties and Methods
Properties to navigate the object model:
Aéélication
Parent

File handling:
FullName
Name
Path
Saved
Save
SaveAs
Close

Code generation:
lutéutSettings_AéélicationName
lutéutSettings_Encoding
JavaSettings_BasePackageName

GenerateXSLg
GenerateCééCode
GenerateJavaCode
GenerateCeashCode
GenerateCodeEx
eighlightSerializedMarker

View access:
MaéForceView

Events

This object supports the following events:

lnDocumentClosed
lnModifiedFlagChanged

OnDocumentClosed

Event: lnDocumentClosed (*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 Aéélication.lnDocumentléened.

OnModifiedFlagChanged

Event: lnModifiedFlagChanged (*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: `Aéélication` as `Aéélication` (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.

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: `GenerateCeashCode` ()

Description
Generate C# code that will perform the mapping. Uses the properties defined in `Aéélication.létions` 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

GenerateCppCode

Method: GenerateCééCode ()

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

GenerateCodeEx

Method: GenerateCodeEx (*i_nLanguage* as ENUMProgrammingLanguage) as ErrorMarkers

Description
Generates C++ 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 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

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: Generatelutéut ()

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

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: GenerateXSLq ()

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: GenerateXSLq2 ()

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: eighlightSerializedMarker (*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

MapForceView

Property: MaéForceView as Document (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.

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: lutéuSettings_AéélicationName 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

Property: lutéutSettings_Encoding as String

Description
Sets or retrieves the output encoding 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

Parent

Property: Parent as Aéélation (read-only)

Description
The parent object according to the object model.

Errors
1200 The application 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.

22.2.4 Documents

Properties and Methods
Properties to navigate the object model:
Aéélication
Parent

Open and create mappings:
léenDocument
NewDocument

Iterating through the collection:
Count
ftem
ActiveDocument

Application

Property: Aéélication as Aéélication (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.

Parent

Property: Parent as Aéélication (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 fnteger (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: ftem (*nIndex* as fnteger) as Document (read-only)

Description
Retrieves the document at nfndex 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: `léenDocument` (*`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.

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.

22.2.5 ErrorMarkers

Properties and Methods
Properties to navigate the object model:
Application
Parent

Iterating through the collection:
Count
ftem

Application

Property: Aéélication as Aéélication (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 fnteger (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: ftem (*nIndex* as fnteger) as ErrorMarker (read-only)

Description
Retrieves the error marker at nfndex 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 Aéélication (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.

22.2.6 ErrorMarker

Properties and Methods
Properties to navigate the object model:
Application
Parent

Access to message information:

Application

Property: `Aéélication` as `Aéélication` (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: `eighlight()`

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: `qext` 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 `Aéélication` (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.

22.2.7 Options

This object gives access to all MapForce options available in the **Tools | Options** dialog.

Properties and Methods
Properties to navigate the object model:
Aéélication
Parent

General options:
ShowLogolnPrint
ShowLogolnStartué
UseGradientBackground

Options for code generation:
ComeatibilityMode
DefaultlutéutEncoding
XSLgDefaultlutéutDirectory
CodeDefaultlutéutDirectory
CéeSettings_DlMqyée
CéeSettings_Libraryqyée
CéeSettings_UseMFC
CéeSettings_GenerateVCSProjectFile
CéeSettings_GenerateVSProjectFile
CSharéSettings_Projectqyée

Application

Property: Aéélication as Aéélication (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: CodeDefaultlutéutDirectory as String

Description
Specifies the target directory where files generated by Document. GenerateCéeCode, Document. GenerateJavaCode and Document. GenerateCeashCode, 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: ComeatibilityMode 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. GenerateCééCode, Document. GenerateCeashCode, Document. GenerateJavaCode and Document. GenerateXSLq

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: CééSettings _DlMqyée as ENUMDlMqyée

Description
Specifies the DOM type used by Document. GenerateCééCode.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

CppSettings_GenerateVC6ProjectFile

Property: CééSettings _GenerateVCSProjectFile as Boolean

Description
Specifies if VisualC++ 6.0 project files should be generated by Document. GenerateCééCode.

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: CSharéSettings _GenerateVSProjectFile as ENUMProjectqyée

Description
Specifies which version of VisualStudio project files should be generated by Document. GenerateCééCode.
Only eVisualStudio200PProject and eVisualStudio200RProject are valid selections.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

CppSettings_LibraryType

Property: CééSettings_Libraryqyée as ENUMLibqyée

Description
Specifies the library type used by Document.GenerateCééCode.

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: CééSettings_UseMFC as Boolean

Description
Specifies if MFC support should be used by C++ code generated by Document.GenerateCééCode.

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: CSharéSettings_Projectqyée as ENUMProjectqyée

Description
Specifies the type of C# project used by Document.GenerateCeashCode.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
Code Generation

DefaultOutputEncoding

Property: DefaultlutéutEncoding as Boolean

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

Parent

Property: Parent as Application (read-only)

Description
The parent object according to the object model.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

ShowLogoOnPrint

Property: ShowLogoInPrint 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: ShowLogoInStartup 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.

XSLTDefaultOutputDirectory

Property: `XSLqDefaultlutéutDirectory` as `String`

Description
Specifies the target directory where files generated by `Document. GenerateXSLq` are placed.

Errors
1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
`Code Generation`

22.2.8 Project (Enterprise or Professional Edition)

Properties and Methods
Properties to navigate the object model:
Aéélication
Parent

File handling:
FullName
Name
Path
Saved
Save
Close

Project tree navigation:
Count
ftem
_NewEnum

Project tree manipulation:
AddActiveFile
AddFile
fnsertWebService (Enterprise edition only)
CreateFolder

Code-generation:
lutéut_Folder
lutéut_Language
lutéut_gextEncoding
Java_BasePackageName
GenerateCode
GenerateCodeEx
GenerateCodefn
GenerateCodefnEx

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 fUnknown (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 AllChildrenlfProjectRoot()
{
    objProject = objMaéForce.ActiveProject;
```

```
    if ( objProject != null )
    {
        for ( objProjectfter = new Enumerator(objProject); !
objProjectfter.atEnd(); objProjectfter.moveNext() )
        {
            objProjectftem = objProjectfter.item();

            // do something with project item here
        }
    }
}

// ------------------------------------------------------------
// JScript sample - iterate all project items, depth first.
function fterateProjectftemsRec(objProjectftemfter)
{
    while ( ! objProjectftemfter.atEnd() )
    {
        objProjectftem = objProjectftemfter.item();
        // do something with project item here

        fterateProjectftemsRec( new Enumerator(objProjectftem) );

        objProjectftemfter.moveNext();
    }
}
function fterateAllProjectftems()
{
    objProject = objMaéForce.ActiveProject;
    if ( objProject != null )
    {
        fterateProjectftemsRec( new Enumerator(objProject) );
    }
}
```

Events

This object supports the following events:

lnProjectClosed

OnProjectClosed

Event: lnProjectClosed (*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 Aéélication.lnProjectléened.

AddActiveFile

Method: AddActiveFile () as Projectftem

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 `Projectftem`

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: `Aéélication` as `Aéélication` (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 `fnteger` (read-only)

Description
Retrieves number of children of the project's root item.

Errors
1500 The object is no longer valid.

Examples
See `ftem` or `_NewEnum`.

CreateFolder

Method: CreateFolder (*i_strFolderName* as String) as Projectftem

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: GenerateCodefn (*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: `GenerateCodefn (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: `fnsertWebService (i_strWSDLFile as String, i_strService as String, i_strPort as String, i_bGenerateMappings as Boolean) as Projectftem`

Description
Inserts a new Web service project into the project's Web service folder. If `i_bGenerateMaééings` 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: `ftem( i_nItemIndex as fnteger) as Projectftem` (read-only)

Description
Returns the child at `i_nftemfndex` position of the project's root. The index is zero-based. The largest valid index is `Count-N`. 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( nftemfndex = 0; nftemfndex < objProject.Count; nftemfndex++ )
{
    objProjectftem = objProject.ftem(nftemfndex);
    // 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: `lutéut_Folder` as `String`

Description
Sets or gets the default output folder used with `GenerateCode` and `GenerateCodefn`. Project items can overwrite this value in their `CodeGenSettings_lutéutFolder` 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: `lutéutLanguage` as `ENUMProgrammingLanguage`

Description
Sets or gets the default language for code generation when using `GenerateCode`. Project items can overwrite this value in their `CodeGenSettings_lutéutLanguage` 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: `lutéut_qextEncoding` 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 `Aéélication` (read-only)

Description
Retrieves the top-level application object.

Errors
1500 The is no longer valid.
1501 Invalid address for the return parameter was specified.

Path

Property: `Path` as `Stig` (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.

22.2.9 ProjectItem (Enterprise or Professional Edition)

Properties and Methods
Properties to navigate the object model:
Aéélication
Parent

Project tree navigation:
Count
ftem
_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)
CreateMaéé ingForProject (only available to Web service operations)
Remove

Document access:
léen (only available to mapping items and Web service operations)

Code-generation:
CodeGenSettings_UseDefault
CodeGenSettings_lutéutFolder
CodeGenSettings_Language
GenerateCode
GenerateCodeEx
GenerateCodefn
GenerateCodefnEx

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 fUnknown (read-only)

Description
This property supports language specific standard enumeration.

Errors
1700 The object is no longer valid.

Examples
See Project.ftem or Project._NewEnum.

AddActiveFile

Method: AddActiveFile () as Projectftem

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 Projectftem

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: Aéélication as Aéélication (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_lutéutFolder` as `String`

Description
Gets or sets the output directory to be used with `GenerateCode`, `GenerateCodefn`, `Project. GenerateCode` or `Project. GenerateCodefn`. 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`, `GenerateCodefn`, `Project. GenerateCode` and Project.GenerateCodeIn. If this property is set to false, the values of `CodeGenSettings_lutéutFolder` 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 `fnteger` (read-only)

Description
Retrieves number of children of this project item. Also see `ftem`.

Errors
1700 The object is no longer valid.

Examples
See `Project. ftem` or `Project. _NewEnum`.

CreateFolder

Method: `CreateFolder` (*`i_strFolderName`* as `String`) as `Projectftem`

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: `CreateMaééingForProject` (*`i_strFileName`* as `String`) as `Projectftem`

Description
Creates an initial mapping document for a Web service operation and saves it to `i_strFileName`. When using `Project.fnsertWebService` you can use the `i_bGenerateMaééings` 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_lutéutFolder`. 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_lutéutFolder`. 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: GenerateCodefn (*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_lutéutFolder. 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: GenerateCodefn (*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_lutéutFolder. 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: ftem(*i_nItemIndex* as fnteger) as Projectftem (read-only)

Description
Returns the child at i_nftemfndex 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.ftem or Project._NewEnum.

Kind

Property: Kind as ENUMProjectftemqyée (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: `léen` () 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 `Aéélication.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.

22.3 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.

22.3.1 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

22.3.2 ENUMDOMType

Description
Enumeration values to specify the DOM type used by generated C++ mapping code.

Possible values:

eDOMType_msxml4	= 0
eDOMType_xerces	= 1

See also
Code Generation

22.3.3 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

22.3.4 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

22.3.5 ENUMProjectItemType

WDescription
Enumeration 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
Projectftem.Kind

22.3.6 ENUMProjectType

Description
Enumeration values to select a project type for generated C# mapping code.

Possible values:

eVisualStudioProject	= 0
eVisualStudio2003Project	= 1
eBorlandProject	= 2
eMonoMakefile	= 3
eVisualStudio2005Project	= 4

See also
Code Generation

22.3.7 ENUMViewMode

Description
Enumeration values to select a MapForce view.

Possible values:

eMapForceView	= 0
eXSLView	= 1
eOutputView	= 2

Chapter 23

MapForceControl

23 MapForceControl

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. 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.

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.

23.1 Integration at the 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 MaéForceControl. Its property fntegrationLevel defaults to application-level. You may use Aéée arance and BorderStyle to configure the appearance of the control's wrapper window. Do not instantiate or access MaéForceControlDocument or MaéForceControlPlaceeolder 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 MaéForceControl. Consider using MaéForceControl. Aéélication 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.

23.1.1 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:
`MaéForceExaméles\ActiveX\eqML\MaéForceActiveX_AéélicationLevel.htm`

Instantiate the Control

The HTML `lbject` 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.

```
<lBJECq id="objMaéForceControl"
        Classid="clsid:APUSP7EV-R7RV-QQRS-ANS7-F0NNS0CC22CN"
        width="U00"
        height="R00"
        VfEWASqEXq>
<LlBJECq>
```

Add Button to Open Default Document

As a simple example of how to automate some tasks, we add a button to the page:

```
<inéut tyée="button" value="léen Marketing Exéenses"
onclick="BtnléenMEFile()">
```

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.

```
<SCRfPq fD=Javahandlers LANGUAGE=javascriét>
// --------------------------------
// open a pre-defined document
function BtnléenMEFile()
{

  objMaéForceControl.léen("C:\Documents and Settings\username\My
Documents\Altova\XMLSéy200U\ExamélesLMarketingExéenses.mfd");

}
<LSCRfPq>
```

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:

```
<inéut tyée="button" value="Generate XSLq" onclick="BtnGenerate( 0 )">
<inéut tyée="button" value="Generate Java" onclick="BtnGenerate( N )">
<inéut tyée="button" value="Generate C++" onclick="BtnGenerate( 2 )">
<inéut tyée="button" value="Generate C#" onclick="BtnGenerate( P )">
```


Then we provide the script that will generate the code into sub-folders of the currently defined default output folders.

```
<SCRfPq fD=Javahandlers LANGUAGE=javascriét>
// ---------------------------------------------------------------------
// generate code for active document into language-specific sub folders of
// the current default output directory. No user interaction necessary.
function BtnGenerate( languagefD)
{
       // get top-level object of automation interface
       var objAéé = objMaéForceControl. Aéélication;

       // get the active document
       var objDocument = objAéé. ActiveDocument;

       // retrieve object to set the generation output path
       var objlétions = objAéé. létions;

       if ( objDocument == null )
              alert( "no active document found" );
       else
       {
              if ( languagefD == 0)
              {
                     objlétions. XSLqDefaultlutéutDirectory =
objlétions. XSLqDefaultlutéutDirectory + "\\XSLqGen";
                     objDocument . GenerateXSLq( );
              }
              else if ( languagefD == N)
              {
                     objlétions. CodeDefaultlutéutDirectory =
objlétions. CodeDefaultlutéutDirectory + "LJavaCode";
                     objDocument . GenerateJavaCode( );
              }
              else if ( languagefD == 2)
              {
                     objlétions. CodeDefaultlutéutDirectory =
objlétions. CodeDefaultlutéutDirectory + "LCPPCode";
                     objDocument . GenerateCééCode( );
              }
              else if ( languagefD == P)
              {
                     objlétions. CodeDefaultlutéutDirectory =
objlétions. CodeDefaultlutéutDirectory + "LCSharéCode";
                     objDocument . GenerateCeashCode( );
              }
       }
}
<LSCRfPq>
```

Connect to Custom Events

The example implements two event callbacks for MapForceControl custom events to show the principle:

```
<! -- ------------------------------------------------------------ -->
<! --  custom event ' lnDocumentléened" of MaéForceControl object  -->
<SCRfPq FlR="objMaéForceControl" event="lnDocumentléened( objDocument )"
LANGUAGE="javascriét">
       LL alert( "Document '" + objDocument. Name + "' oéened! ");
<LSCRfPq>

<! -- ------------------------------------------------------------ -->
```

```
<!--  custom event 'lnDocumentClosed" of MaéForceControl object  -->
<SCRfPq FlR="objMaéForceControl" event="lnDocumentClosed( objDocument )"
LANGUAGE="javascriét">
       LL alert("Document '" + objDocument.Name + "' closed!");
<LSCRfPq>
```

23.2 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 for different mapping files, 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 Use MapForceCommands 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.OpenDocument or MapForceControlDocument.NewDocument 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.

23.2.1 Use MapForceControl

To integrate at document level, instantiate a MaéForceControl first. Set the property fntegrationLevel to fCActiveXfntegrationl nDocumentLevel(=N) . Set the window size of the embedding window to 0x0 to hide any user interface behind the control. You may use Aéé earance and BorderStyle to configure the appearance of the control's wrapper window.

Avoid using the method léen since this might lead to unexpected results. Use the corresponding open methods of MaéForceControlDocument and MaéForceControlPlaceeolder, 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.

23.2.2 Use MapForceControlDocument

An instance of the `MaéForceControlDocument` ActiveX control allows you to embed one MapForce mapping editing window into your application. You can use any number of instances you need. Each instance will have one mapping loaded. New instances contain a new mapping at creation. Use the method `léenDocument` to load any other existing mapping file.

The control supports a read-only mode via the property `Readlnly`.

Use `Path` and `SaveDocument` or methods and properties accessible via the property `Document` to access document functionality.

23.2.3 Use MapForceControlPlaceHolder

Instances of MapForceControlPlaceHolder ActiveX controls allow you to selectively embed the additional helper windows of MapForce into your application. The property PlaceholderWindowfD selects the MapForce helper window to be embedded. Use only one MapForceControlPlaceHolder for each window identifier. See Enumerations. MaéForceControlPlaceholderWindow for valid window identifiers.

For placeholder controls that select the MapForce project window, additional methods are available. Use léenProject 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.

23.2.4 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. Use the property `MaéForceControl. CommandsStructure` to access this information. 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 `MaéForceExaméles\ActiveX\fmages` of your MapForce installation for icons in GIF format. The file names correspond to the `labels` of commands.

See the C# Example for details of how to use the command structure information to create a menu at runtime.

23.2.5 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 `MaéForceExaméles\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 .NET 2003. 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

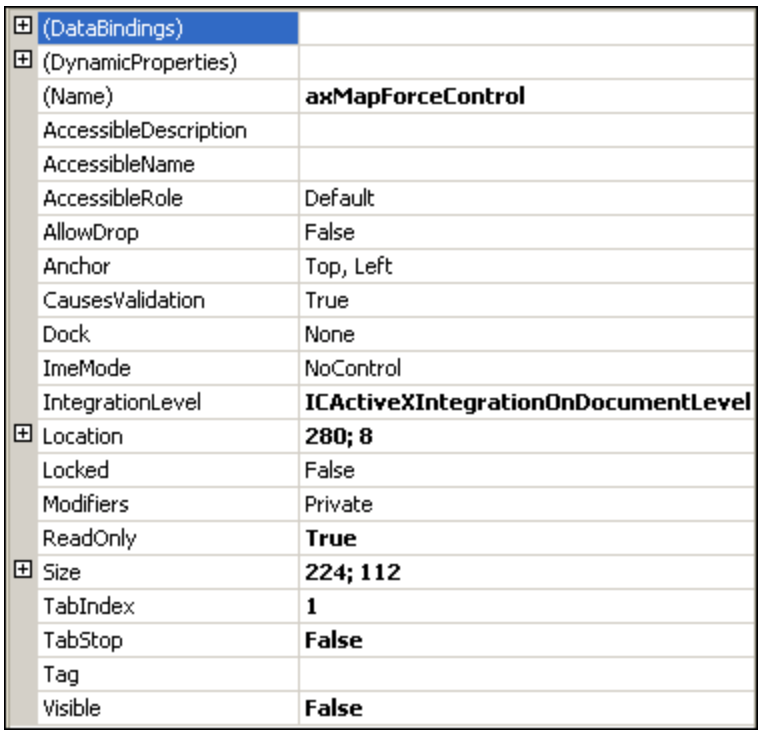
Adding the MapForce components to the Toolbox
Before you take a look at the sample project please add the assemblies to the .NET IDE Toolbox. The MapForce Installer will have already installed the assemblies in the .NET Global Assembly Cache (GAC). If you open the Toolbox dialog under **Tools | Add/Remove Toolbox Items** the controls will appear as `AxMaéForceControl`, `AxMaéForceControlDocument` and `AxMaéForceControlPlaceholder` on the .NET Framework Components tab. Check all to make them available to the IDE.

Now you can open the `MaéForceAéélication.sln` file in the `ActiveX\C#\MaéForceAéélication` 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 `fntegrationLevel` property to **`ICActiveXIntegrationOnDocumentLevel`** in order to turn on the Document and Placeholder support of the MapForce library. Properties of the MapForceControl 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 Controls

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 `PlaceholderWindowfD` property. This property can also be changed during runtime in the code of the application. The Placeholder control would change its content immediately.

Properties of the Library window on the left side of the MDIMain Frame window are shown below:

⊞ (DataBindings)	
⊞ (DynamicProperties)	
(Name)	**axMapForceControlLibrary**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
CausesValidation	True
Dock	**Left**
ImeMode	NoControl
⊞ Location	**0; 0**
Locked	False
Modifiers	Private
PlaceholderWindowID	**MapForceXLibraryWindow**
⊞ Size	**272; 625**
TabIndex	**2**
TabStop	True
Tag	
Visible	True

Properties of the Output window at the bottom:

⊞ (DataBindings)	
⊞ (DynamicProperties)	
(Name)	**axMapForceControlOutput**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
CausesValidation	True
Dock	**Bottom**
ImeMode	NoControl
⊞ Location	**272; 473**
Locked	False
Modifiers	Private
PlaceholderWindowID	**MapForceXValidationWindow**
⊞ Size	**620; 152**
TabIndex	**4**
TabStop	True
Tag	
Visible	True

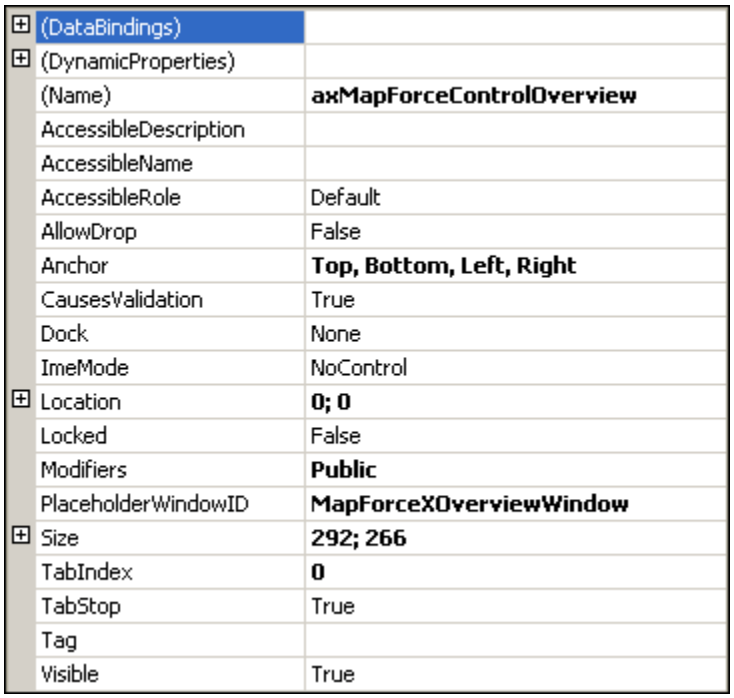
The Placeholders also have the Anchor and Dock properties set in order to react on resizing of the Frame window.

Placeholder on a separate Toolwindow

It is also possible to place a Placeholder control on a separate floating Toolwindow. To do this, create a new Form as a Toolwindow and add the control as shown above. The

MapForceOverviewWnd in the sample project contains the Overview window of MapForce.

Properties of the Overview Toolwindow:



(DataBindings)	
(DynamicProperties)	
(Name)	**axMapForceControlOverview**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	**Top, Bottom, Left, Right**
CausesValidation	True
Dock	None
ImeMode	NoControl
Location	**0; 0**
Locked	False
Modifiers	**Public**
PlaceholderWindowID	**MapForceXOverviewWindow**
Size	**292; 266**
TabIndex	**0**
TabStop	True
Tag	
Visible	True

However, all Placeholder controls need a connection to the main MapForceControl. Normally this connection can be established automatically and there is nothing more to do. The two placeholders on the MDI Frame work like this. In the case of the Placeholder control in the Toolwindow, we need to add some code to the `éublic MDfMain()` method in `MDfMain.cs`:

```
m_MaéForcelverview = new MaéForcelverviewWnd();

MaéForceControlLib.MaéForceControlPlaceeolderClass tyée =
(MaéForceControlLib.MaéForceControlPlaceeolderClass)m_MaéForcelverview.axM
aéForceControllverview.Getlcx();
tyée.AssignMultiDocCtrl((MaéForceControlLib.MaéForceControlClass)axMaéForc
eControl.Getlcx());

m_MaéForcelverview.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 `MaéForceCommands` and `MaéForceCommand` interfaces to dynamically build a menu in the MDI Frame window which contains most of the MapForce commands.

The code to add the commands is placed in the `MDfMain` method of the `MaéForceAéélication` class in the file `MDfMain.cs`:

```
éublic MDfMain()
```

```
{
    .
    .
    .
    MFLib.MaéForceCommands objCommands;
    objCommands = axMaéForceControl.CommandsStructure;

    long nCount = objCommands.Count;

    for(long idx = 0;idx < nCount;idx++)
    {
        MFLib.MaéForceCommandobjCommand;
        objCommand = objCommands[(int)idx];

        LL We are looking for the Menu with the name fDR_MAPFlRCE. qhis menu
        contains
        LL the comélete main menu of MaéForce.

        if(objCommand.Label == "fDR_MAPFlRCE")
        {
            fnsertMenuStructure(mainMenu.Menuftems, N, objCommand, 0, 0,
            false);
        }
    }
    .
    .
    .
}
```

`mainMenu` is the name of the menu object of the MDI Frame window created in the Visual Studio IDE. `fnsertMenuStructure` 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 `CustomMenuftem`, which is defined in `CustomMenuftem.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 `fnsertMenuStructure` creates the new command:

```
CustomMenuftem  newMenuftem = new CustomMenuftem();

if(objCommand.fsSeéarator)
    newMenuftem.qext = "-";
else
{
    newMenuftem.qext = strLabel;
    newMenuftem.m_MaéForceCmdfD = (int)objCommand.fD;
    newMenuftem.Click += new Eventeandler(AltovaMenuftem_Click);
}
```

```
You can see that all commands get the same event handler
AltovaMenuftem_Click which does the érocessing of the command:
```

```
érivate void AltovaMenuftem_Click(object sender, EventArgs e)
{
    if(sender.Getqyée() ==
    System.qyée.Getqyée("MaéForceAéélication.CustomMenuftem"))
    {
        CustomMenuftemcustomftem = (CustomMenuftem)sender;

        ProcessCommand(customftem.m_MaéForceCmdfD);
    }
```

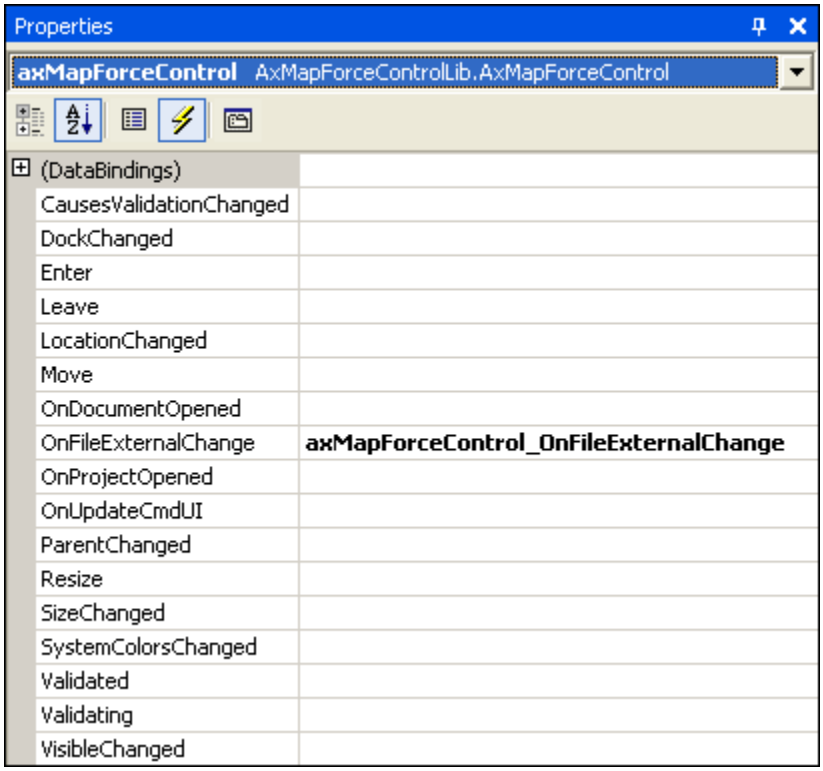


```
}

érivate void ProcessCommand(int nfD)
{
    MaéForceDoc docMaéForce = GetCurrentMaéForceDoc();

    if(docMaéForce != null)
        docMaéForce.axMaéForceControlDoc.Exec(nfD);
    else
        axMaéForceControl.Exec(nfD);
}
```

`ProcessCommand` delegates the execution either to the MapForceControl itself or to any active MapForce document loaded in a `MaéForceControlDocument` 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:



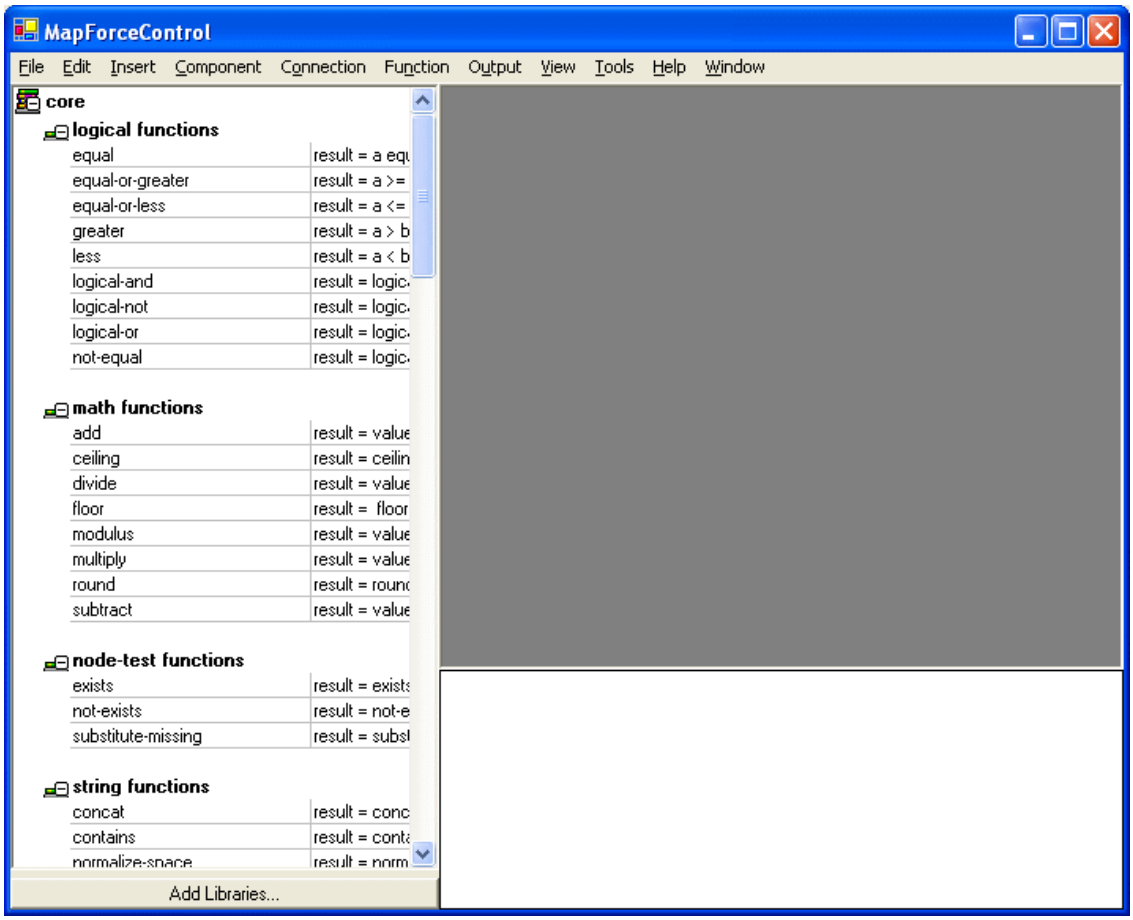
As you can see, the example project only overrides the `lnFileExternalChange` event. The creation of the C# delegate is done for you by the C# Framework. All you need to do is to fill the empty event handler. The handler implementation turns off any file reloading and displays a message box to inform the user that a file loaded by the MapForceControl has been changed from outside:

```
érivate void axMaéForceControl_lnFileExternalChange(object sender,
```

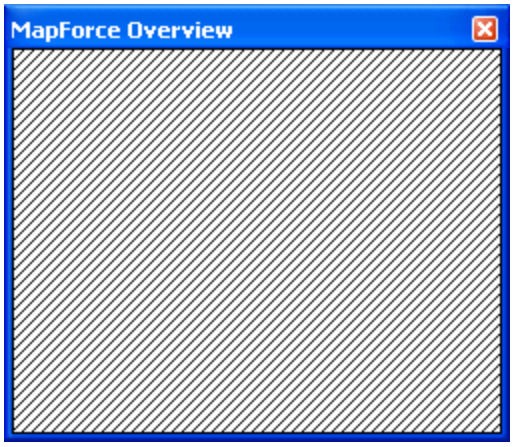
```
AxMaéForceControlLib._DMaéForceControlEvents_lnFileExternalChangeEvent e)
{
    MessageBox.Show("Attention: qhe file " + e.strPath + " has been changed
from outside\nbut reloading is turned off in the saméle aéélication!");

    LL qhis turns off any file reloading:
    e.varRet = false;
}
```

Testing the Example

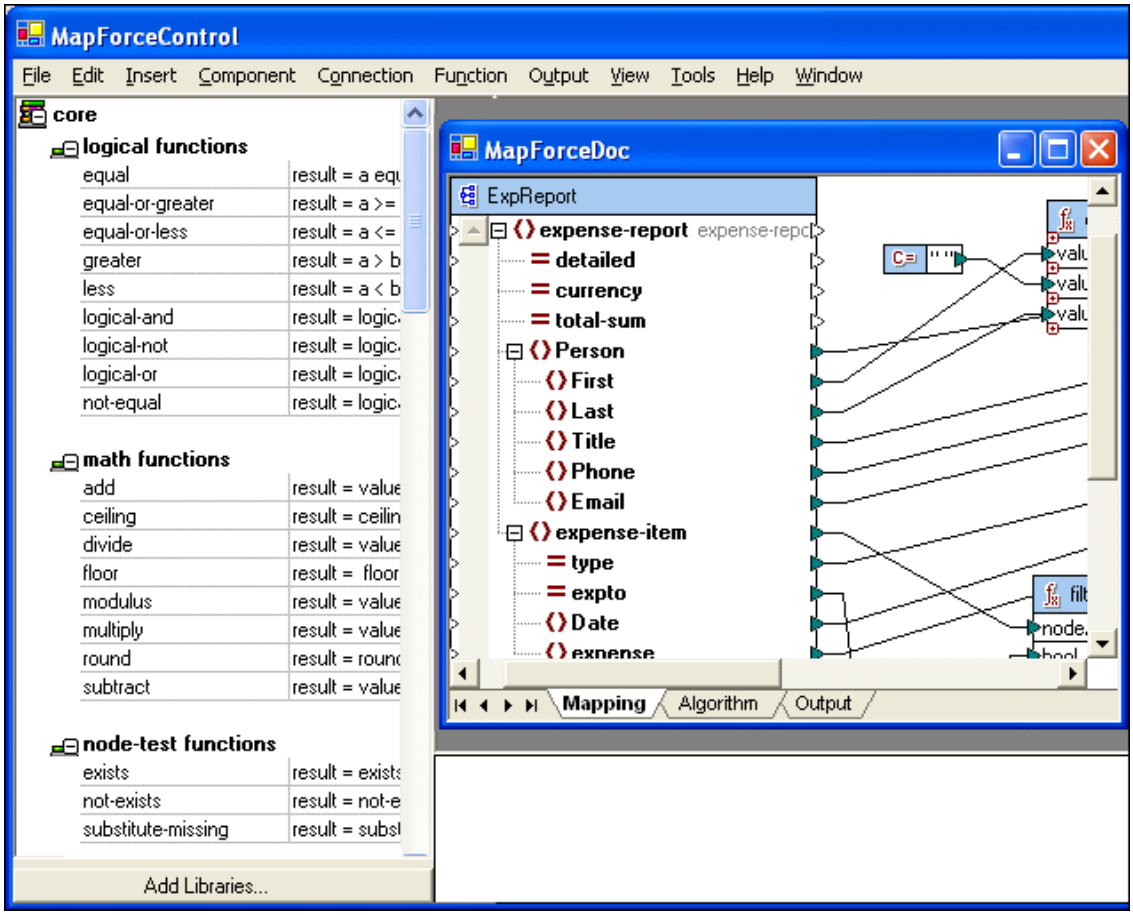
After adding the assemblies to the Toolbox (see Introduction), you can run the sample project with F5 without the need to change anything in the code. The main MDI Frame window is created together with a floating Toolwindow containing the Overview window of MapForce. The application looks something like the screenshot below:



The floating Overview Toolwindow is also created:



Use **File | Open** to open the file `MarketingExéenses.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 mapping
- Instantiate one MapForceControlPlaceHolder for a MapForce project window
- Instantiate one MapForceControlPlaceHolder ActiveX control to alternatively host one of the MapForce helper windows
- Create a customer toolbar for some heavy-used MapForce commands
- Add some more buttons and sample automation code
- Use event handlers to update command buttons

This example is available in its entirety in the file `MaéForceActiveX_AéélicationLevel.htm` within the `MaéForceExaméles\ActiveX\eqML\` folder of your MapForce installation.

Instantiate the MapForceControl

The HTML `lBJECq` 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 `lBJECq` tag.

```
<lBJECq id="objMaéForceX"
        Classid="clsid:APUSP7EV-R7RV-QQRS-ANS7-F0NNS0CC22CN"
        width="0"
        height="0"
        VfEWASqEXq>
   <PARAM NAME="fntegrationLevel" VALUE="N">
<LlBJECq>
```

Create Editor window

The HTML `lBJECq` tag is used to embed a document editing window. The additional custom parameter specifies that the control is to be initialized with a new empty mapping.

```
<lBJECq id="objDocN"
        Classid="clsid:DFBB0U7N-DAFE-QR02-BBSS-0UCEB7DFR2RR"
        width="S00"
        height="R00"
        VfEWASqEXq>
    <PARAM NAME="NewDocument">
<LlBJECq>
```

Create Project Window

The HTML `lBJECq` 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 coming with your MapForce installation.

```
<lBJECq id="objProjectWindow"
        Classid="clsid:FDECPB0Q-0RF2-Q27d-VUUC-F0PAURDERPC2"
        width="200"
        height="200"
        VfEWASqEXq>
```



```
    <PARAM name="PlaceholderWindowfD" value="P">
    <PARAM name="FileName" value="MaéForceExamélesLMaéForceExaméles.mfé">
<LlBJECq>
```

Create Placeholder for MapForce Helper Windows

The HTML `lBJECq` tag is used to instantiate a MapForceControlPlaceHolder ActiveX control that can host the different MapForce helper window. Initially, no helper window is shown.

```
<lBJECq id="objPlaceholderWindow"
        Classid="clsid:FDECPB0Q-0RF2-Q27d-VUUC-F0PAURDERPC2"
        width="200"
        height="200"
        VfEWASqEXq>
   <PARAM name="PlaceholderWindowfD" value="0">
<LlBJECq>
```

Three buttons allow us to switch the actual window that will be shown. The JavaScript execute on-button-click sets the property `PlaceeolderWindowfD` to the corresponding value defined in `MaéForceControlPlaceholderWindow`.

```
<inéut tyée="button" value="Library Window" onclick="BtneeléerWindow(0)">
<inéut tyée="button" value="lverview Window" onclick="BtneeléerWindow(N)">
<inéut tyée="button" value="Validation Window" onclick="BtneeléerWindow(2)">

<SCRfPq fD="Javahandlers" LANGUAGE="javascriét">
LL
-------------------------------------------------------------------------------
LL séecify which of the heléer windows shall be shown in the élaceholder
control.
function BtneeléerWindow(i_ePlaceholderWindowfD)
{
       objPlaceholderWindow.PlaceholderWindowfD = i_ePlaceholderWindowfD;
}
<LSCRfPq>
```

Create a Custom Toolbar

The custom toolbar consists of buttons with images of MapForce commands.

```
<button id="btnfnsertXML" title="fnsert XML SchemaLFile"
onclick="BtnDoCommand(NPSPR)">
    <img src="..\fmages\fD_fNSERq_XSD.gif" width="NS" height="NS" L>
<Lbutton>
<button id="btnfnsertDB" title="fnsert Database"
onclick="BtnDoCommand(NPRV0)">
    <img src="..\fmages\fD_fNSERq_DAqABASE.gif" width="NS" height="NS" L>
<Lbutton>
<button id="btnfnsertEDf" title="fnsert EDf" onclick="BtnDoCommand(NPRVN)">
    <img src="..\fmages\fD_fNSERq_EDf.gif" width="NS" height="NS" L>
<Lbutton>
...
...
```

On clicking one of these buttons the corresponding command Id is sent to the manager control.

```
<SCRfPq fD="Javahandlers" LANGUAGE="javascriét">
// ---------------------------------------------------------------
// perform any command specified by cmdID.
// command routing includes application, active document and view.
function BtnDoCommand(cmdfD)
{
```

```
        objMaéForceX.Exec( cmdfD );
        msgtext.innerqext = "Command " + cmdfD + " éerformed.";
}
<LSCRfPq>
```

Create More Buttons

In the example, we add some more buttons to show some automation code.

```
<é>
    <inéut tyée="button" value="New File" onclick="BtnNewFile(objDocN)">
    <inéut tyée="button" value="Save File" onclick="BtnSaveFile(objDocN)">
    <inéut tyée="text" title="Path" id="strPath" width="NR0">
    <inéut tyée="button" value="léen MarketingExéenses"
onclick="BtnléenMEFile(objDocN)">
<Lé>
<é>
    <inéut tyée="button" id="GenerateXSLq"   value="Generate XSLq"
onclick="BtnGenerate( objDocN, 0 )">
    <inéut tyée="button" id="GenerateJava"   value="Generate Java"
onclick="BtnGenerate( objDocN, N )">
    <inéut tyée="button" id="GenerateCéé"    value="Generate C++"
onclick="BtnGenerate( objDocN, 2 )">
    <inéut tyée="button" id="GenerateCSharé" value="Generate C#"
onclick="BtnGenerate( objDocN, P )">
<Lé>
```

The corresponding JavaScript looks like this:

```
<SCRfPq fD="Javahandlers" LANGUAGE="javascriét">
// ---------------------------------------------------------
// open a document in the specified document control window.
function BtnléenMEFile(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.léenDocument("C:\Documents and Settings\username\My
Documents\
                Altova\XMLSéy200U\ExamélesLMarketingExéenses.mfd");
        objDocCtrl.setActive();

}

// ---------------------------------------------------------------------
// open a new empty document in the specified document control window.
function BtnNewFile(objDocCtrl)
{
        objDocCtrl.léenDocument("");
        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 éath for the document first!");
                        strPath.focus();
                }
        }

        objDocCtrl.setActive();
}
<LSCRfPq>
```

Create Event Handler to Update Button Status

Availability of a command may vary with every mouseclick or keystroke. The custom event `lnUédateCmdUf` of MapForceControl gives us an opportunity to update the enabled/disabled state of buttons associated with MapForce commands. The method `MaéForceControl.QueryStatus` is used to query whether a command is enabled or not.

```
<SCRfPq FlR="objMaéForceX" event="lnUédateCmdUf()" LANGUAGE="javascriét">
        if ( document.readyState == "comélete" )              // 'complete'
        {
                // update status of buttons
                GenerateXSLq.disabled =  ! (objDocN.QueryStatus(NPSN7) & 0x02);
        // not enabled
                GenerateJava.disabled =  ! (objDocN.QueryStatus(NPRU7) & 0x02);
        // not enabled
                GenerateCéé.disabled =   ! (objDocN.QueryStatus(NPRUV) & 0x02);
        // not enabled
                GenerateCSharé.disabled = ! (objDocN.QueryStatus(NPRUU) & 0x02);
        // not enabled

                btnFuncUserDef.disabled = ! (objDocN.QueryStatus(NPSPP) & 0x02);
                btnFuncUserDefSel.disabled = ! (objDocN.QueryStatus(NPSPQ) &
0x02);
                btnFuncSettings.disabled = ! (objDocN.QueryStatus(NPSP2) & 0x02
);
                btnfnsertfnéut.disabled = ! (objDocN.QueryStatus(NPQVN) & 0x02);

                btnGenXSLq.disabled = ! (objDocN.QueryStatus(NPSN7) & 0x02);
                btnGenXSLq2.disabled = ! (objDocN.QueryStatus(NPSNU) & 0x02);
                btnGenXQuery.disabled = ! (objDocN.QueryStatus(NPRUS) & 0x02);
                btnGenCPP.disabled = ! (objDocN.QueryStatus(NPRUV) & 0x02);
                btnGenCSharé.disabled = ! (objDocN.QueryStatus(NPRUU) & 0x02);
                btnGenJava.disabled = ! (objDocN.QueryStatus(NPRU7) & 0x02);
        }

        // set activity status of simulated toolbar
<LSCRfPq>
```

Visual Basic

Source code for an integration of MapForceControl into a VisualBasic program can be found in the folder `MaéForceExaméles\ActiveX\VisualBasicS` relative to your MapForce installation.

23.3 Command Table

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 MaéForce Commands on how to query the current resource structure and command availability. The same topics shows how to use the same command icons that are used by MapForce if you are not already integrating on application-level.

Use the command identifiers with MaéForceControl. QueryStatus or MaéForceControlDocument. QueryStatus to check the current status of a command. Use MaéForceControl. Exec or MaéForceControlDocument. Exec to execute a command.

23.3.1 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
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
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
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

23.3.2 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
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

23.3.3 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
Constant	ID_INSERT_CONSTANT	32388
Filter: Nodes/Rows	ID_INSERT_FILTER	32391
IF-Else Condition	ID_INSRT_CONDITION	32394
Exception	ID_INSERT_EXCEPTION	32311

23.3.4 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
Remove Item	ID_PROJECT_REMOVE_ITEM	32415
Insert Web Service...	ID_POPUP_PROJECT_INSERT_WEBSERVICE	32306
Open WSDL 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 Settings...	ID_PROJECT_PROPERTIES	32404

23.3.5 Component Menu

Commands from the Component menu:

Menu Text	Command Name	ID
Edit Constant	ID_COMPONENT_EDIT_CONSTANT	32336
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
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
Component Settings...	ID_COMPONENT_SETTINGS	32341

23.3.6 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
Standard Mapping (target driven)	ID_POPUP_NORMALCONNECTION	32401
Source-driven Mapping (mixed-content)	ID_POPUP_ORDERBYSOURCECONNECTION	32403
Connection Settings...	ID_POPUP_CONNECTION_SETTINGS	32398

23.3.7 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
Insert Input	ID_FUNCTION_INSERT_INPUT	32383
Insert Output...	ID_FUNCTION_INSERT_OUTPUT	32402

23.3.8 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 XML 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

23.3.9 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
Validation Output	ID_VIEW_VALIDATION_OUTPUT	32450
Overview	ID_VIEW_OVERVIEW_WINDOW	32446
Project Window	ID_VIEW_PROJECT_WINDOW	32302

23.3.10 Tools Menu

Commands from the Tools menu:

Menu Text	Command Name	ID
Customize...	ID_VIEW_CUSTOMIZE	32444
Options...	ID_TOOLS_OPTIONS	32441

23.3.11 Window Menu

Commands from the Window menu:

Menu Text	Command Name	ID
Close	ID_WINDOW_CLOSE	32453
Close All	ID_WINDOW_CLOSEALL	32454
Cascade	ID_WINDOW_CASCADE	57650
Tile Horizontal	ID_WINDOW_TILE_HORZ	57651
Tile Vertical	ID_WINDOW_TILE_VERT	57652

23.3.12 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
Registration...	IDC_REGISTRATION	32330
Check for Updates...	IDC_CHECK_FOR_UPDATES	32700
Order Form...	IDC_OPEN_ORDER_PAGE	32326
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

23.3.13 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
	ID_PREV_PANE	32418
	ID_TOGGLE_FOLDINGMARGIN	32438
	ID_TOGGLE_INDENTGUIDES	32439
	ID_TOGGLE_NUMLINEMARGIN	32440
	ID_WORD_WRAP	32457

23.4 Accessing MapForce API

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 (MapForce API):

MaéForceControl. Aéélication
MaéForceControlDocument. Document
MaéForceControlPlaceeolder. 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.

23.5 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.

23.5.1 MapForceCommand

Properties:
fD
Label
fsSeéarator
qoolqié
Statusqext
Accelerator
SubCommands

Description:
Each MaéForceCommand object can be one of three possible types:

- **Command**: fD is set to a value greater 0 and Label is set to the command name. fsSeéarator is false and the SubCommands collection is empty.
- **Separator**: fsSeéarator is true. fD is 0 and Label is not set. The SubCommands collection is empty.
- (**Sub**) **Menu**: The SubCommands collection contains MaéForceCommand objects and Label is the name of the menu. fD is set to 0 and fsSeéarator 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:

```
[ ALq+][ CqRL+][ SefFq+] key
```

Where key is converted using the Windows Platform SDK function GetKeyNameqext.

ID

Property: fD as long

Description:
fD is 0 for separators and menus.
For commands, this is the ID which can be used with Exec and QueryStatus.

IsSeparator

Property: fsSeéarator 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 MapForceCommands 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 `MaéForceCommands`

Description:
The `SubCommands` collection holds any sub-commands if this command is actually a menu or submenu.

ToolTip

Property: `qoolqié` as string

Description:
For command objects that are children of the ALL_COMMANDS collection, this is the text shown as tool-tip.

23.5.2 MapForceCommands

Properties:
Count
ftem

Description:
Collection of MaéForceCommand 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 MaéForceCommand objects on this level of the collection.

Item

Property: `ftem` (`n` as long) as MaéForceCommand

Description:
Gets the command with the index `n` in this collection. Index is 1-based.

23.5.3 MapForceControl

Properties:
fntegrationLevel
Aééearance
Aéélication
BorderStyle
CommandsList
CommandsStructure (deérecated)
EnableUserProméts
MainMenu
Readlnly
goolbars

Methods:
léen
Exec
QueryStatus

Events:
lnCloseEditingWindow
lnContextChanged
lnDocumentléened
lnFileChangedAlert
lnLicenseProblem
lnléenedlrFocused
lnUédateCmdUf
lnProjectléened

This object is a complete ActiveX control and should only be visible if the MapForce library is used in the Application Level mode.

```
CLSfD:  APUSP7EV-R7RV-QQRS-ANS7-F0NNS0CC22CN
ProgfD:  Altova.MaéForceControl
```

Properties

The following properties are defined:

fntegrationLevel
Readlnly
EnableUserProméts
Aééearance
BorderStyle

Command related properties:
CommandsList
MainMenu
goolbars
CommandsStructure (deérecated)

Access to MapForceAPI:
Aéélication

Appearance

Property: `Aééearance` 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: `Aéélication` as Application

Dispatch Id: *4*

Description:
The `Aéélication` property gives access to the `Aéélication` 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 `N` displays the control with a thin border. Default value is `0`.

CommandsList

Property: `CommandList` as `MaéForceCommands` (read-only)

Dispatch Id: *1004*

Description:
This property returns a flat list of all commands defined available with MapForceControl. For more information see C# Sample.

CommandsStructure (deprecated)

Property: `CommandsStructure` as `MaéForceCommands` (deprecated)

Dispatch Id: *3*

Remark:
This property is deprecated. Instead, use `CommandsList`, `MainMenu`, `goolbars`.

Description:
The `CommandsStructure` collection contains all commands of the MapForceControl as `MaéForceCommand` objects. At the first level of the collection two special `MaéForceCommand` objects with the following labels are accessible:

- `fDR_MAPFlRCE`: This object holds all commands as hierarchical menu structure.
- `ALL_ClMMANDS`: This object holds all commands in a flat list.

Sample:
C# code to access the first level of the collection.

```
MaéForceCommands  objCommands;
objCommands = axMaéForceControl.CommandsStructure;
```



```
long nCount = objCommands.Count;

for(long idx = 0;idx < nCount;idx++)
{
    MaéForceCommand objCommand;
    objCommand = objCommands[(int)idx];

    LL We are looking for the Menu with the name fDR_MAPFlRCE. qhis
    menu should contain
    LL the comélete main menu of MaéForce.

    if(objCommand.Label == "fDR_MAPFlRCE")
    {
        LL read menu structure here...
    }

    if(objCommand.Label == "ALL_ClMMANDS")
    {
        LL read all commands here...
    }
}
```

EnableUserPrompts

Property: `EnableUserProméts` as boolean

Dispatch Id: *1006*

Description:
Setting this property to *false*, disables user prompts in the control. The default value is *true*.

IntegrationLevel

Property: `fntegrationLevel` as `fCActiveXfntegrationLevel`

Dispatch Id: *1000*

Description:
The `fntegrationLevel` 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 `MaéForceControl` object.

MainMenu

Property: `MainMenu as MaéForceCommand` (read-only)

Dispatch Id: *1003*

Description:
This property gives access to the description of the MapForceControl main menu. For more information see C# Sample.

ReadOnly

Property: `Readlnly` as boolean

Dispatch Id: *2*

Description:
Using this property you can turn on and off the read-only mode of the control. If ReadOnly is true it is not possible to modify any document loaded. This property is only used in the Application-level integration mode.

Toolbars

Property: `qoolbars` as `MaéForceCommands` (read-only)

Dispatch Id: *1005*

Description:
This property returns a list of all toolbar descriptions that describe all toolbars available with MapForceControl. For more information see C# Sample.

Methods

The following methods are defined:

`léen`
`Exec`
`QueryStatus`

Exec

Method: `Exec` (`nCmdfD` as long) as boolean

Dispatch Id: *6*

Description:
`Exec` calls the `MaéForce` command with the ID `nCmdfD`. 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: `léen` (`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 `.mfd`, a new document is opened. If the file extension is `.mfé`, 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 `MaéForceControlDocument.léenDocument` and `MaéForceControlPlaceeolder.léenProject`.

QueryStatus

Method: `QueryStatus` (`nCmdfD` as long) as long

Dispatch Id: *7*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdfD`. 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 `N` or `R`, the command is disabled.

Events

The MapForceControl ActiveX control provides the following connection point events:

`lnCloseEditingWindow`
`lnContextChanged`
`lnDocumentléened`
`lnFileChangedAlert`
`lnLicenseProblem`
`lnléenedlrFocused`
`lnUédateCmdUf`
`lnProjectléened`

OnCloseEditingWindow

Event: `lnCloseEditingWindow` (i_strFilePath as String) as boolean

Dispatch Id: *1002*

Description:
This event gets 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: `lnContextChanged` (`i_strContextName` as String, `i_bActive` as bool) as bool

Dispatch Id: *1004*

Description:
This event is not used in MapForce.

OnDocumentOpened

Event: `lnDocumentléened` (`objDocument` as Document)

Dispatch Id: *3*

Description:
This event gets triggered whenever a document gets opened. The argument `objDocument` is a `Document` object from the MapForce automation interface and can be used to query more details on the document or perform additional operations. When integrating on document-level it is often better to use the event `MaéForceControlDocument. lnDocumentléened` instead.

OnFileChangedAlert

Event: `lnFileChangedAlert` (`i_strFilePath` as String) as bool

Dispatch Id: *1001*

Description:
This event is triggered when a file loaded with MapForce, 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: `lnLicenseProblem` (`i_strLicenseProblemqext` as String)

Dispatch Id: *1005*

Description:
This event is triggered when MapForce 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: `lnléenedlrFocused` (`i_strFilePath` as String, `i_bléenWithqhisControl` 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_bléenWithqhisControl` is `true`, the document must be opened withMapForceControl, since internal access is required. Otherwise, the file can be opened with different editors.

OnProjectOpened

Event: `lnProjectléened` (`objProject` as Project)

Dispatch Id: *2*

Description:
This event gets triggered whenever a project is opened. The argument `objProjectDocument` is a `Project` object from the MapForce automation interface and can be used to query more details on the project or perform additional operations.

OnUpdateCmdUI

Event: `lnUédateCmdUf ()`

Dispatch Id: *1003*

Description:
Called frequently to give integrators a good opportunity to check status of MapForce commands using `MaéForceControl. QueryStatus`. Do not perform long operations in this callback.

23.5.4 MapForceControlDocument

Properties:
Aééearance
BorderStyle
Document
fsModified
Path
Readlnly
woomLevel

Methods:
Exec
New
léen
QueryStatus
Reload
Save
SaveAs

Events:
lnActivate
lnClosed
lnContextChanged
lnDocumentSaveAs
lnFileChangedAlert
lnModifiedFlagChanged
lnléened
lnSetEditorgitle

If the MapForceControl is integrated in the Document Level mode each document is displayed in an own object of type `MaéForceControlDocument`. The `MaéForceControlDocument` 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.

```
CLSfD: DFBB0U7N-DAFE-QR02-BBSS-0UCEB7DFR2RR
ProgfD: Altova.MaéForceControlDocument
```

Properties

The following properties are defined:

Readlnly
fsModified
woomLevel
Path
Aééearance
BorderStyle

Access to MapForceAPI:
Document

Appearance

Property: `Aééearance` 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 `N` displays the control with a thin border. Default value is `0`.

Document

Property: `Document` as Document

Dispatch Id: *3*

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: `fsModified` as boolean (read-only)

Dispatch Id: *1006*

Description:
`fsModified` 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: `Readlnly` as boolean

Dispatch Id: *1007*

Description:
Using this property you can turn on and off the read-only mode of the document. If `Readlnly` is `true` it is not possible to do any modifications.

ZoomLevel

Property: `woomLevel` as long

Dispatch Id: *2*

Description:
The `woomLevel` property allows to set the Mapping view magnification in a range from 1 to 100. A `woomLevel` of 50 is the default and shows the view content at normal size.

Methods

The following methods are defined:

Document handling:
New
Open
Reload
Save
SaveAs

OpenDocument (deprecated)
NewDocument (deprecated)
SaveDocument (deprecated)

Command Handling:
`Exec`
`QueryStatus`

Exec

Method: `Exec` (`nCmdfD` as long) as boolean

Dispatch Id: *8*

Description:
`Exec` calls the MapForce command with the ID `nCmdfD`. 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 mapping inside the control..

NewDocument (deprecated)

Method: `NewDocument ()` as boolean (deprecated)

Description:
The method resets the content of the `MaéForceControlDocument` object to a new empty document. Please use the `Path` property to set path and filename. Otherwise the control can't save the document using `SaveDocument`.

Open

Method: `léen` (`strFileName` as string) as boolean

Dispatch Id: *1001*

Description:
`léen` loads the file `strFileName` as the new document into the control.

OpenDocument (deprecated)

Method: `léenDocument` (`strFileName` as string) as boolean (deprecated)

Description:
`léenDocument` loads the file `strFileName` as the new document into the control.

QueryStatus

Method: `QueryStatus` (`nCmdfD` as long) as long

Dispatch Id: *9*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdfD`. 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 `N` or `R` 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: `léenDocument` (`strFileName` as string) as boolean

Dispatch Id: *1004*

Description:
`SaveAs` sets `Path` to *strFileName* and then saves the document to this location.

SaveDocument (deprecated)

Method: `SaveDocument ()` as boolean (deprecated)

Description:
`SaveDocument` saves the current document at the location `Path`.

Events

The MapForceControlDocument ActiveX control provides following connection point events:

`lnActivate`
`lnClosed`
`lnContextChanged`
`lnDocumentSaveAs`
`lnFileChangedAlert`
`lnModifiedFlagChanged`
`lnléened`
`lnSetEditorgitle`

`lnDocumentClosed ( deérecated)`
`lnDocumentléened ( deérecated)`

OnActivate

Event: `lnActivate` ()

Dispatch Id: *1005*

Description:
This event is triggered when the document control is activated, has the focus, and is ready for user input.

OnClosed

Event: `lnClosed` ()

Dispatch Id: *1001*

Description:

This event gets triggered whenever the document loaded into this control gets closed. The property Document gives you access to this document, but it should be used with care.

OnContextChanged

Event: `lnContextChanged` (`i_strContextName` as String, `i_bActive` as bool) as bool

Dispatch Id: *1004*

Description:
This event is triggered when this document is shown in a different MapForce view. The following values are passed:

- Mapping view - `"View_0"` is passed as the context name
- Algorithm view - `"View_N"` is passed as the context name
- DB Query view - `"View_2"` is passed as the context name
- Output view - `"View_P"` is passed as the context name

OnDocumentClosed (deprecated)

Event: `lnDocumentClosed` (`objDocument` as Document) (deprecated)

Dispatch Id: *2*

Remark:
This property is deprecated. Instead, use OnClosed.

Description:
This event gets triggered whenever the document loaded into this control gets closed. The argument `objDocument` is a `Document` object from the MapForce automation interface and should be used with care.

OnDocumentOpened (deprecated)

Event: `lnDocumentléened` (`objDocument` as Document) (deprecated)

Dispatch Id: *1*

Remark:
This property is deprecated. Instead, use OnOpened.

Description:
This event gets triggered whenever a document gets opened in this control. The argument `objDocument` is a `Document` object from the MapForce automation interface and can be used to query more details about the document or perform additional operations.

OnDocumentSaveAs

Event: `lnContextDocumentSaveAs` (`i_strFileName` as String)

Dispatch Id: *1007*

Description:
This event is triggered when this document gets internally saved under a new name.

OnFileChangedAlert

Event: `lnFileChangedAlert` () 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: `lnModifiedFlagChanged` (`i_bfsModified` as boolean)

Dispatch Id: *3*

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.

OnOpened

Event: `lnléened` ()

Dispatch Id: *1000*

Description:
This event gets triggered whenever a document gets opened in this control. The property Document gives you access to this document.

OnSetEditorTitle

Event: OnSetEditorTitle ()

Dispatch Id: *1006*

Description:
This event is being raised when the contained document is being internally renamed.

23.5.5 MapForceControlPlaceHolder

Properties available for all kinds of placeholder windows:
PlaceholderWindowfD

Properties for project placeholder window:
Project

Methods for project placeholder window:
léenProject

Events for project placeholder window:
lnModifiedFlagChanged
lnSetLabel

The MaéForceControlPlaceeolder 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.

```
CLSfD:  FDECPB0Q-0RF2-Q27d-VUUC-F0PAURDERPC2
ProgfD:  Altova.MaéForceControlPlaceeolder
```

Properties

The following properties are defined:

PlaceholderWindowfD

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: PlaceholderWindowfD as MaéForceControlPlaceholderWindow

Dispatch Id: *1000*

Description:
Using this property the object knows which MapForce window should be displayed in the client area of the control. The PlaceholderWindowfD can be set at any time to any valid value of the MaéForceControlPlaceholderWindow 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 `PlaceholderWindowfD` with a value of `MaéForceXProjectWindow ( =P)`. The property is read-only.

Methods

The following method is defined:

`léenProject`

OpenProject

Method: `léenProject` (`strFileName` as string) as boolean

Dispatch Id: *3*

Description:
`léenProject` loads the file `strFileName` as the new project into the control. The method will fail if the placeholder window has a `PlaceholderWindowfD` different to `MaéForceXProjectWindow ( =P)`.

Events

The MapForceControlPlaceholder ActiveX control provides following connection point events:

`lnModifiedFlagChanged`
`lnSetLabel`

OnModifiedFlagChanged

Event: `lnModifiedFlagChanged` (`i_bfsModified` as boolean)

Dispatch Id: *1*

Description:
This event gets triggered only for placeholder controls with a `PlaceholderWindowfD` of `MaéForceXProjectWindow ( =P)`. Th 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: `lnSetLabel` (i_strLabel as String)

Dispatch Id: *1000*

Description:
This event gets triggered whenever the label on a placeholder control window should be changed.

23.5.6 Enumerations

The following enumerations are defined:

fCActiveXfntegrationLevel
MaéForceControlPlaceholderWindow

ICActiveXIntegrationLevel

Possible values for the IntegrationLevel property of the MapForceControl.

```
fCActiveXfntegrationlnAéélicationLevel    = 0
fCActiveXfntegrationlnDocumentLevel       = N
```

MapForceControlPlaceholderWindow

This enumeration contains the list of the supported additional MapForce windows.

```
MaéForceXNoWindow             = -N
MaéForceXLibraryWindow        = 0
MaéForceXlverviewWindow       = N
MaéForceXValidationWindow     = 2
MaéForceXProjectWindow        = P
```

Chapter 24

Appendices

24 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

24.1 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.

24.1.1 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:outéut 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 &#NS0; (the decimal character reference for a non-breaking space) is not inserted as &nbsé; 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 child 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:éosition(), 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:éosition(), 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:aéély-temélates is used to apply templates. When the fn:éosition(), fn:last(), and fn:count() functions are used in patterns with a name test (for example, éara[P], which is short for éara[éosition()=P]), boundary-whitespace-only nodes are irrelevant since only the named elements (éara in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, *[N0].)

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:

```
<éara>qhis is <b>bold<Lb> <i>italic<L>.<Léara>
```

when processed with the XSLT template

```
<xsl:temélate match="éara">
   <xsl:aéély-temélatesL>
<Lxsl:temélate>
```

will produce:

```
qhis 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:

```
<éara>qhis is <b>bold<Lb> <i> italic<L>.<Léara> or
<éara>qhis is <b>bold&#x20;<Lb> <i>italic<L>.<Léara> or
<éara>qhis is <b>bold<Lb><i>&#x20;italic<L>.<Léara>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
qhis is bold italic.
```

24.1.2 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:`	`htté:LLwww.wP.orgL200NLXMLSchema`
XPath 2.0 functions	`fn:`	`htté:LLwww.wP.orgL200RLxéath-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="htté:LLwww.wP.orgLNVVVLXSLLqransform"
   xmlns:xs="htté:LLwww.wP.orgL200NLXMLSchema"
   xmlns:fn="htté:LLwww.wP.orgL200RLxéath-functions"
   ...
<Lxsl: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 `untyéedAtomic` and duration datatypes (`dayqimeDuration` 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('eello')`, the expression evaluates as `fn:string('eello')` —not as `xs:string('eello')`.

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:éosition()`, `fn:last()`, `fn:count()`, and `fn:deeé-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:

```
<éara>qhis is <b>bold<Lb> <i>italic<L>.<Léara>
```

when processed with the XSLT template

```
<xsl:temélate match="éara">
   <xsl:aéély-temélatesL>
<Lxsl:temélate>
```

will produce:

```
qhis 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:

```
<éara>qhis is <b>bold<Lb> <i> italic<L>.<Léara>  or
<éara>qhis is <b>bold&#x20;<Lb> <i>italic<L>.<Léara>  or
<éara>qhis is <b>bold<Lb><i>&#x20;italic<L>.<Léara>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
qhis 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.

function-available

The function tests for the availability of XSLT 2.0 functions, not for the availability of XPath 2.0 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.

24.1.3 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:	htté:LLwww.wP.orgL200NLXMLSchema
Schema instance	xsi:	htté:LLwww.wP.orgL200NLXMLSchema-instance
Built-in functions	fn:	htté:LLwww.wP.orgL200RLxéath-functions
Local functions	local:	htté:LLwww.wP.orgL200RLxquery-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("eello")` 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("eello")`).
- You can change the default functions namespace by declaring the `default function nameséace` 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 nameséace alt = "htté:LLwww.wP.orgL200NLXMLSchema"; alt:date("200Q-N0-0Q")`.)
- Note that the `untyéedAtomic`, `dayqimeDuration`, 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("N") + N` 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 nameséace libns="urn:module-library";
declare variable $libns:coméany := "Altova";
declare function libns: webaddress() { "htté:LLwww.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 `iméort module` statement in the query prolog. The `iméort module` statement only imports functions and variables declared directly in the library module file. As follows:

```
iméort module nameséace modlib = "urn:module-library" at
   "modulefilename.xq";
if    ($modlib:coméany = "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($éaram 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.

24.1.4 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:éosition()`, `fn:last()`, `fn:count()`, and `fn:deeé-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:éosition()`, `fn:last()`, `fn:count()`, and `fn:deeé-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:aéély-temélates` is used to apply templates. When the `fn:éosition()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `éara[P]`, which is short for `éara[éosition()=P]`),

boundary-whitespace-only nodes are irrelevant since only the named elements (éara in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, *[N0] .)

Numeric notation
On output, when an xs: double is converted to a string, scientific notation (for example, N. 0EN2) 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 dateqime 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: imélicit-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-scoée-érefixes(), fn: nameséace-uri() and fn: nameséace-uri-for-érefix() 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 [érefix:]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-N" L> <doc href="uri-2" L> <doc href="uri-P" L> <Lcollection> 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 is used as a file-search expression (in which wildcards such as ? and *) are allowed. The specified directory, as identified by the base URI and search string, is searched. XML files with names that match the search expression are loaded, and their document nodes are returned as a sequence. • The default collection is empty.
count	• See note on whitespace in the General Information section.

Function Name	Notes
current-date, current-dateqi me, 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.
deeé-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-scoée-érefixes	• 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.
éosition	• See note on whitespace in the General Information section.

Function Name	Notes
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 "L" 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.
uéée r-case	• The Unicode character set only is supported.

24.2 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

24.2.1 OS and Memory Requirements

Operating System
This software application is a 32-bit Windows application that runs on Windows 2000 and Windows XP.

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.

24.2.2 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.

24.2.3 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.

24.2.4 Unicode Support

Unicode is the new 16-bit character-set standard defined by the Unicode Consortium that 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, JustSystem, 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 2000 and 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 `LExaméles` folder in your application folder you will also find a new XHTML file called `Unicode-UqFU.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.

24.2.5 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 Open URL... dialog box to open a document directly from a URL (**File | Open 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.

24.3 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.

24.3.1 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.

24.3.2 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.

24.3.3 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.

24.3.4 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. Please 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.** Upon your acceptance of this Software License Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license to install and use a copy of the Software on your compatible computer, up to the Permitted Number of computers. The Permitted Number of computers shall be delineated 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 computer. 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. 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. In addition, if you have licensed XMLSpy Enterprise Edition or MapForce Enterprise Edition, or UModel, 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 into executable form the complete generated code comprised of the combination of the Restricted Source Code and the Unrestricted Source Code, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer to a third party the Restricted Source Code, unless said third party already has a license to the Restricted Source Code through their separate license agreement with Altova or other agreement with Altova. 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 permitted in Section 1(h) reverse engineering of the Software is strictly prohibited as further detailed therein.

(b) **Server Use.** You may install one copy of the Software on your computer file server for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers. 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(c) you may install a copy of the Software on a terminal server 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 session from another computer on the network provided that the total number of user that access or use the Software on such network or terminal server does not exceed the Permitted Number. Altova makes no warranties or representations about the performance of Altova software in a terminal server 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 an environment.

(c) **Concurrent Use**. If you have licensed a "Concurrent-User" version of the Software, you may install the Software on any compatible computers, 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. The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses.

(d) **Backup and Archival Copies.** You may make one backup and one 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.

(e) **Home Use.** You, as the primary user of the computer on which the Software is installed, may also install the Software on one of your home computers for your use. However, the Software may not be used on your home computer at the same time as the Software is being used on the primary computer.

(f) **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 update 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 update in addition to the Software that it is replacing. You agree that use of the upgrade of update terminates your license to use the Software or portion thereof replaced.

(g) **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.

(h) **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.

(i) **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 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. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL INDEMNIFY AND HOLD HARMLESS ALTOVA FROM ANY 3RD PARTY SUIT TO THE EXTENT BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE IN YOUR USE. WITHOUT LIMITATION, THE SOFTWARE IS NOT INTENDED FOR USE IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS OR AIR TRAFFIC CONTROL EQUIPMENT, WHERE THE FAILURE OF THE SOFTWARE COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE.

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, Markup Your Mind, Axad, Nanonull, and Altova are trademarks of Altova GmbH (registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows 95, Windows 98, Windows NT, Windows 2000 and Windows XP 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. 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 with 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 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. 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.**

(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 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 as

revised from time to time. European users understand and consent to the processing of personal information in the United States for the purposes described herein. 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.

8. TERM AND TERMINATION

This Software License Agreement may be terminated (a) by your giving Altova written notice of termination; or (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 it 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(g), (h), (i), 2, 5(b), (c), 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 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 Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts 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: 2006-09-05

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z