API Gateway

Version 7.6.2

26 March 2019

Visual Mapper User Guide

This documentation describes the following Axway software:

Axway API Gateway 7.6.2

Contents

Preface

This guide describes how to define XSLT mappings in API Gateway using the Visual Mapper component.

Who should read this guide

This guide is intended for policy developers who are using Policy Studio to develop APIs and data maps used by the policies.

Other technical or business users might find parts of this guide useful as well.

How to use this guide

This guide should be used in conjunction with the other guides in the API Gateway documentation set.

Before you begin, review this guide thoroughly. The following is a brief description of the contents of each section:

Introduction to Visual Mapper on page 8 – Describes the main features of the Visual Mapper component.

Transformation tutorial on page 10 - Build a "Hello world" transformation, step by step.

Data Map Editor on page 18 – Describes how to graphically design a mapping in the Data Map Editor.

Properties view on page 33 – Describes the Data Map Editor Properties view.

Variables and external parameters on page 34 – Describes how to insert variables and external parameters in a mapping in the Data Map Editor.

Source and Run views on page 36 – Describes how to view the generated XSLT code for a mapping, and how to validate a mapping in the Data Map Editor.

Validation and error handling on page 39 – Describes how the Data Map Editor provides real time validation on operations to ensure the output message is valid against the target schema.

Auto-mapping on page 42 - Describes how the repeating structures that contain the target node are identified and how the structural links are automatically created between the source and target parents with multiple cardinality.

Custom XSLT code on page 49 - Describes how to use the custom XSLT code option to enrich transformation with expressions.

Support services

The Axway Global Support team provides worldwide 24 x 7 support for customers with active support agreements.

Email support@axway.com or visit Axway Support at https://support.axway.com.

Training services

Axway offers training across the globe, including on-site instructor-led classes and self-paced online learning. For details, go to: http://www.axway.com/support-services/training

Accessibility

Axway strives to create accessible products and documentation for users. The following describes the accessibility features of the documentation.

Documentation accessibility

The product documentation provides the following accessibility features:

Screen reader support

- Alternative text is provided for images whenever necessary.
- The PDF documents are tagged to provide a logical reading order.

Support for high contrast and accessible use of colors

- The documentation can be used in high-contrast mode.
- There is sufficient contrast between the text and the background color.
- The graphics have the right level of contrast and take into account the way color-blind people perceive colors.

Introduction to Visual Mapper 1

The Visual Mapper tool is delivered with some Axway products. Its interface, known as the *Data Map Editor*, can be found in the following place(s):

- Axway Policy Studio, at the **Resources** > **Data maps** node of the Policy Studio tree.
- Other points of access may exist at the time you are reading this. Refer to the documentation of the product(s) you are using for further information.

The Visual Mapper matches an *input format* to an *output format*, to convert incoming data - either read from Web services or from a disk - for the purpose of easily delivering ready-to-use data to your applications. The formats available for input and output are the following:

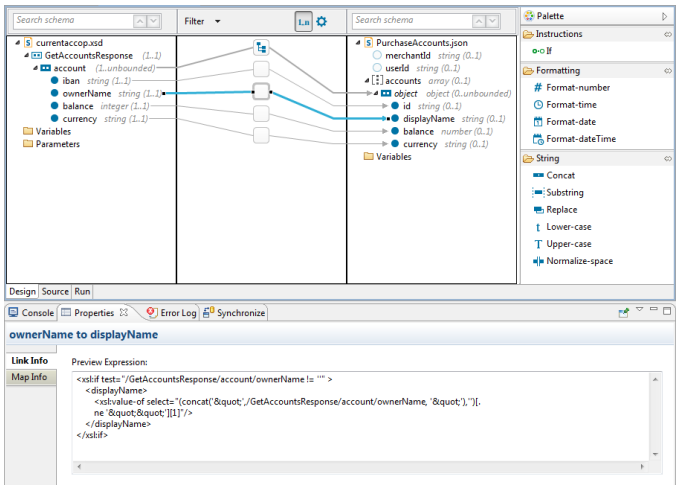
- XML
- Json

Here are some examples of use cases of Visual Mapper, chosen in the context of Policy Studio:

- to convert legacy data from a XML format to Json, for consumption as by mobile applications, via REST APIs
- to convert data from one XML or Json format to another, for example in the context of a corporate merger, where two companies with separate histories have decided to use the same data format
- to extract small portions of larger APIs, for easier and more convenient consumption by your applications.

Visual Mapper's Data Map Editor is composed of several views, available from the **Design** - **Source** - **Run** tabs at the bottom of the main section.

- The *Design view* on page 18 is a drag-and-drop interface for designing a map relationship for linking and enriching the outcoming data from the incoming data.
- The *Source view* on page 36 contains the XSLT (Extensible Stylesheet Language Transformations) code generated by the manipulations in the Design view. One way of using Visual Mapper is to start by manipulating the Design view to approximate the result, and then pursue editing the XLST code in text mode on the Source view.
- The *Run view* on page 37 provides a test environment for your transformations



Map overview

Each map receives input messages (JSON or XML) and transforms them into the required output messages (JSON or XML) based on the map relationship you design in the Data Map Editor. You can also customize the map using external parameters.

Creating a new Map

- In Policy Studio: to create a new map , right-click the **Data Maps** node in the tree (under Resources) and select **Add new Data Map**.
- Other means of creating new Data maps may be available at the time you are reading this.

To define the new map with a **single** input schema, you need to provide:

- A name for the map.
- A description of the input message (source schema). This must include the type of the message (XML or JSON), the schema and the root element.
- A description of the output message (target schema). This must include the type of the message (XML or JSON), the schema and the root element.

To define the new map with **multiple** input schemas, you need to provide:

- A name for the map.
- A description of each input message (source schemas). This must include the type of the messages (XML or JSON), the schemas and the root element for each selected schema.
- A description of the output message (target schema). This must include the type of the message (XML or JSON), the schema and the root element.

Known limitations

The following limitations exist for this version of the Visual Mapper:

- Not all XSD elements are supported (for example, xsd:list).
- Only JSON schema draft4 is supported.
- Only the most common XSLT functions are available in the Visual Mapper palette; although, all XSLT functions can be used in the expression editors.
- When using large XSD schemas (more than 1000 elements), the Auto-mapping feature should be disabled, or used only on small number of records (see *Configure auto-mapping settings* on page 45).

Transformation tutorial

This section provides a "Hello World!"-type example of a transformation made easy by the use of Visual Mapper. We have three files to work with:

- a data file: *address.json file* on page 17
- an input schema, reflecting the structure of the data file above: *address_schema_in.json* on page 15
- an output schema, reflecting the structure of the data we want as an output: *address_schema_out.json* on page 16

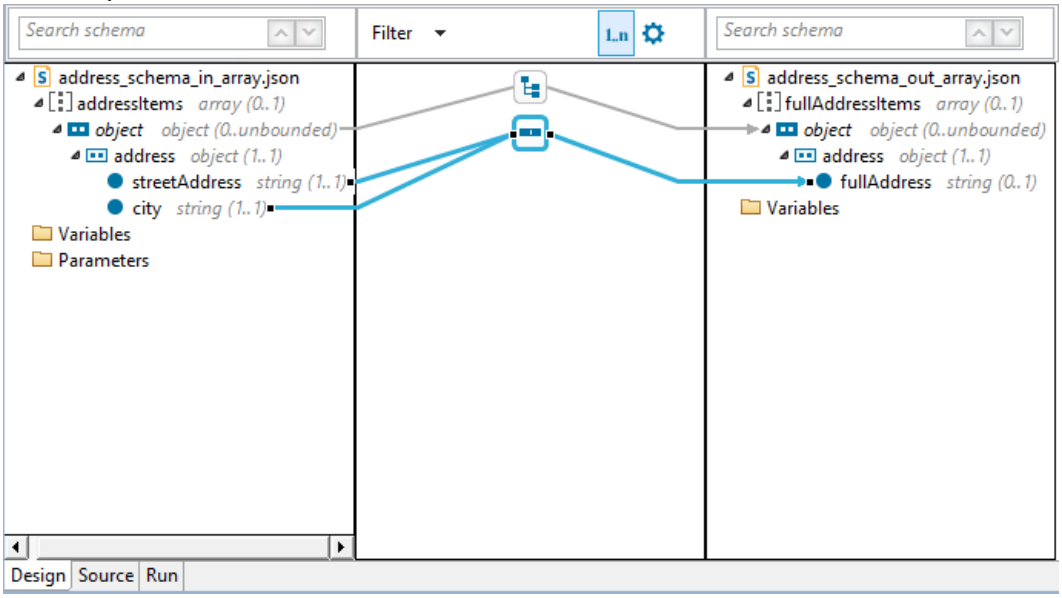
From the input, we are going to concatenate the `street_address` (for example: `"24 Guild Street"`) with the city (`"London"`) , into a `full_address` in the output (`"24 Guild Street, London"`) .

Here is an example of how to perform a transformation with Visual Mapper:

1. Using your favorite text editor, create 3 json files, as indicated in the sections below
2. Create a new Data map.
    - In Policy Studio: to create a new map , right-click the **Data Maps** node in the tree (under Resources) and select **Add new Data Map**.
    - Other means of creating new Data maps may be available at the time you are reading this.
3. Enter the characteristics of the new data map:
    - **Name**: `address`
    - **Source Schema details**:
        - **Type**: `JSON`
        - **Locate on disk**: `yes`
        - **Schema**: click on **[...]** button to search for the `address_schema_in.json` file you created in step 1

- **Target Schema details**:
  - **Type**: `JSON`
  - **Locate on disk**: `yes`
  - **Schema**: click on **[...]** button to search for the `address_schema_out.json` file

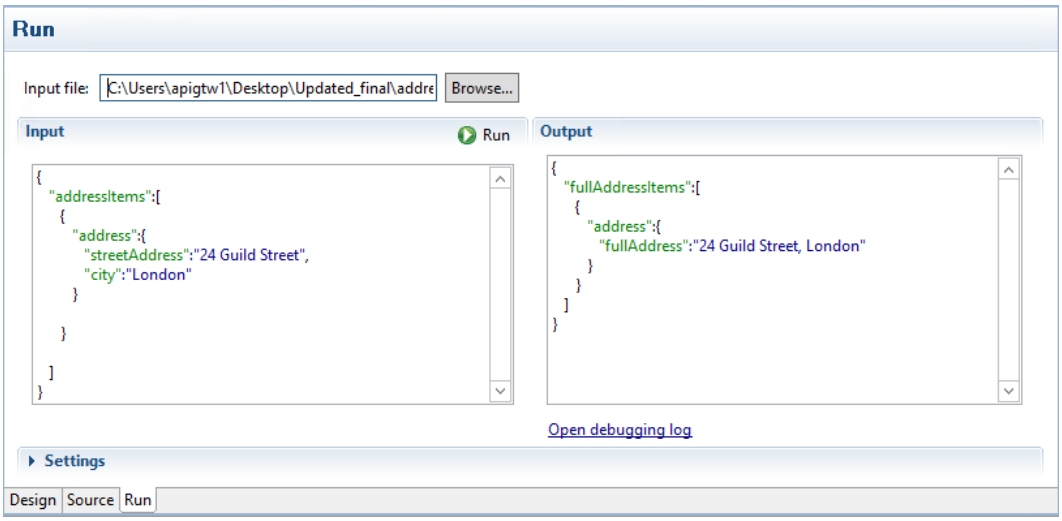
4. The data map appears as a 3-pane window (see the *Design tab* screenshot below)
   - The left pane reflects the data structure of the file `address_schema_in.json`
   - The right pane reflects the data structure of `address_schema_out.json`
   - The center pane is empty for the moment (the screenshot below is a few steps ahead of us)



The Design tab

5. Drag to link `streetAddress` on the left to `fullAddress` to the right
   A rounded square appears in the center pane, materializing the new link

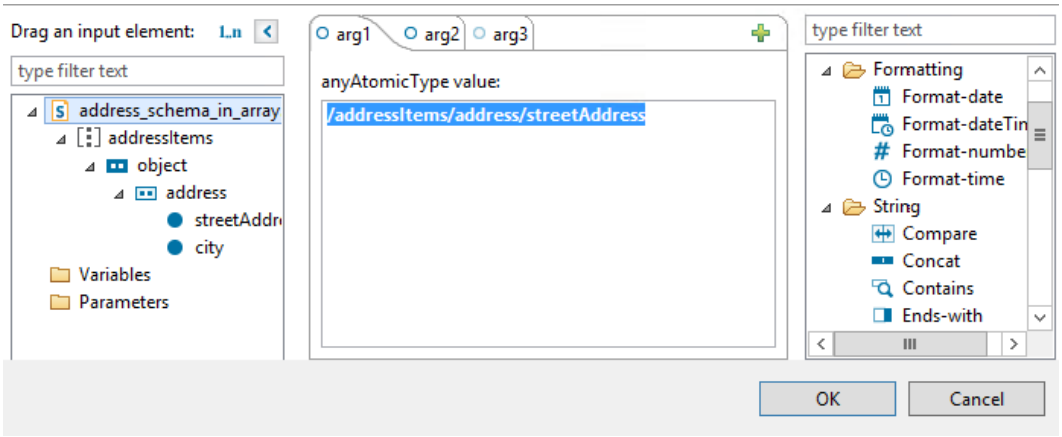
6. Select **Run** tab among the Design - Source - Run tabs at the bottom of the data map
   The Run window appears (again, the screenshot is ahead of us, showing the result we intend to achieve: at this stage, the Input and Output fields are empty).



The Run tab

7. Click **Browse...** button to load the data file, named `address.json` in this example
8. Click the **Run** button 
   In the **Output** field, you should see the following text, showing that the data from the `streetAddress` in input has been copied verbatim to `fullAddress` on the output:
   `{ "address": { "fullAddress":"24 Guild Street" } }`
9. In the **Design** view, drag a link from `city` on the left to the rounded square in the center. The square now has two links coming from the input and one link going to the output.
   - Note that you can see the rounded square is selected, because the lines are bold and they have square points at the end.
10. Right click on the square and select **Set function** > **String** > **Concat**
11. Re-run the transformation as indicated above. The `fullAddress` in the output is now: `"24 Guild StreetLondon"`.
    We are nearly there! All we need now is to add in a comma between `Street` and `London`.

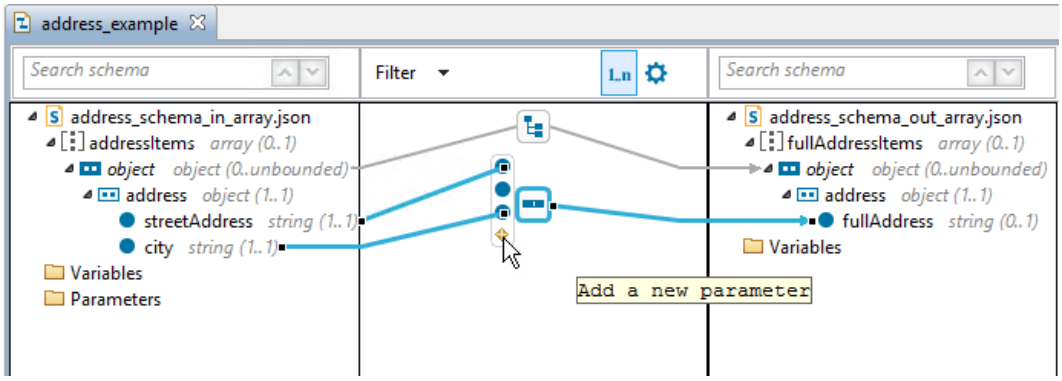
12. Return to the **Design** tab, right click on the square in the center, select **Edit...**
The **Edit Function Parameter** window appears.



It has two arguments in its center, named arg1 and arg2:

- **arg1**: `/address/streetAddress`
- **arg2**: `/address/city`

13. Click the **[+]** button to add a new argument. An `arg3` tab appears. Type in `", "` (comma - space, without the quotes), then drag the new `arg3` tab towards the center, to switch `arg2` and `arg3`.
14. Re-run the transformation as indicated above. The output fits the specifications!
15. In the Design view, hover over the rounded square symbolizing the function. You will see:
    - 3 dots symbolizing the 3 arguments
    - a + sign, for adding a new argument



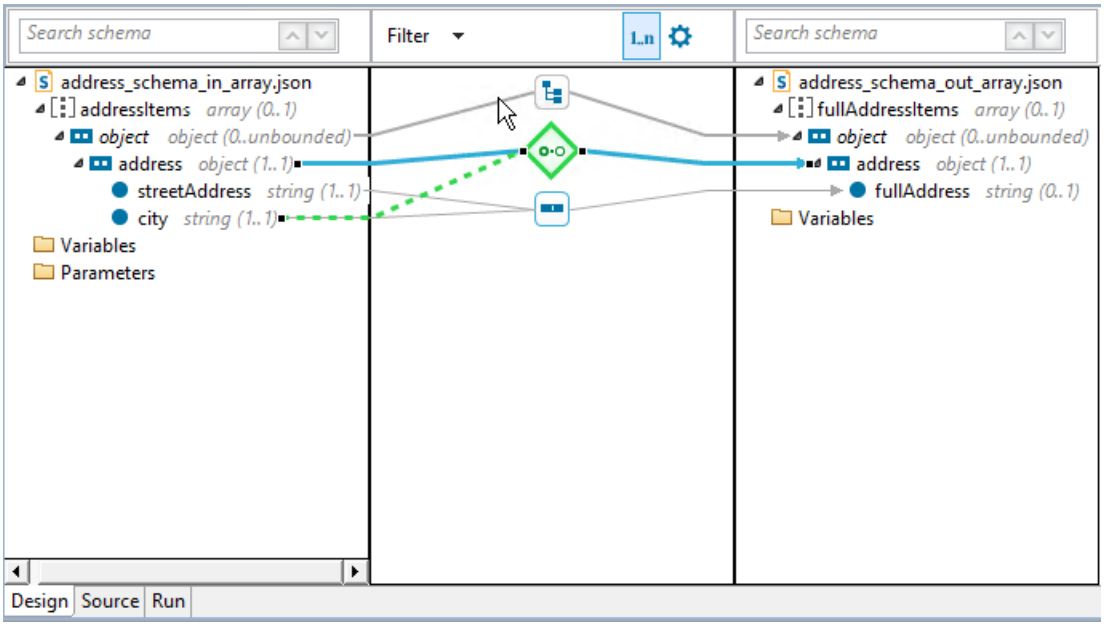
Hovering over the concatenation function

Adding a condition

Now we are going to add some conditional processing between the input and the output.

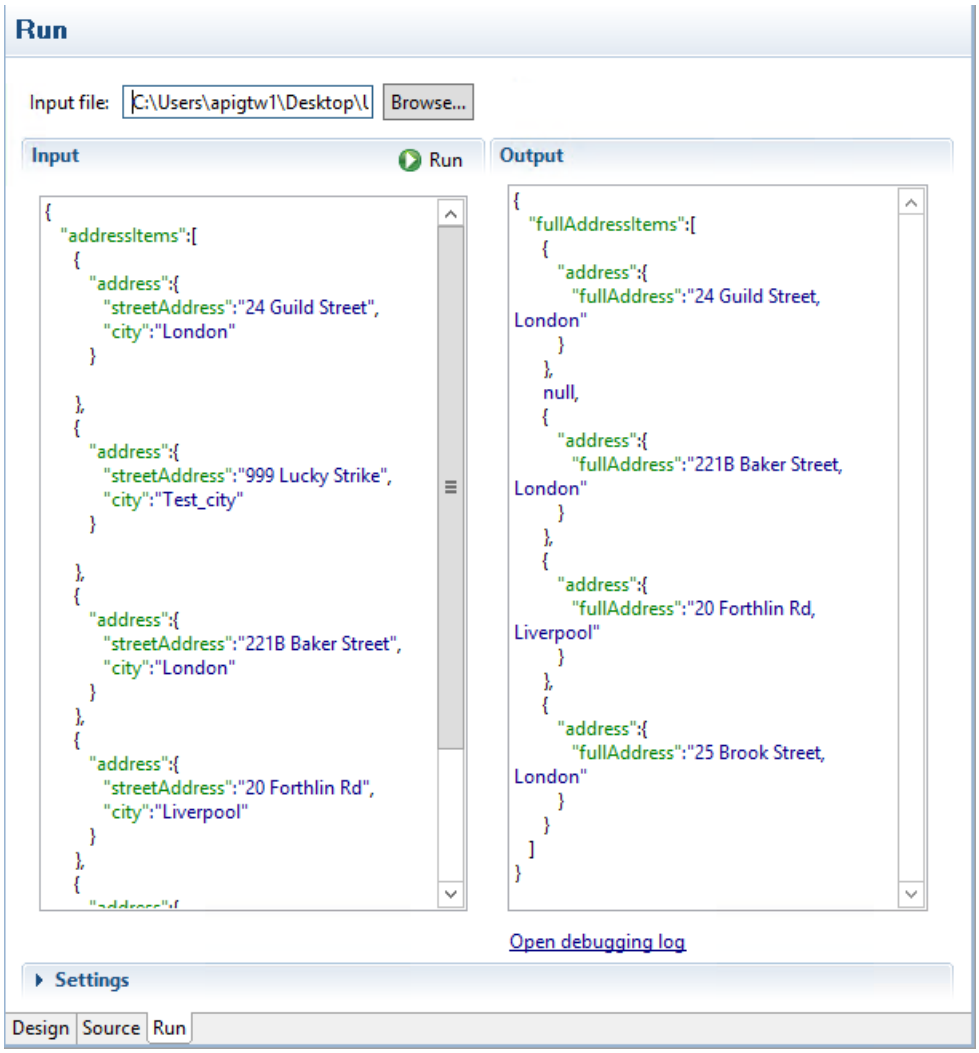
Suppose the input data contains test records, recognizable because the city is equal to "`Test_city`". We don't want those records in the output. To do that, we will create an `If` condition, as follows:

1. Drag a link between `address` on the left and `address` on the right .
   A square appears in the center.
2. Right-click on the square, then select **Set function** > **If**
   The square becomes a lozenge, containing an icon symbolizing the `If` condition
3. Drag a link between `city` (on the left) and the lozenge



4. Right-click on the lozenge, then select **Edit...**
   In the Condition tab, you will see the following:
   `/address/city != ''`
5. Edit the text to fit the required condition:
   `/address/city != 'Test_city'`
6. Edit the data file to add a record with
   `"city":"Test_city"`
7. To reload the new Source file, click the **Run** tab, then **Browse...** button again

8. Re-run the transformation.
   The content of the **Output** section now reflects the condition.



9. If you need to debug your transformation, click on the **> Settings** entry and check the **Log debugging information** box.

Data files for the tutorial

address_schema_in.json

Copy the code below, then paste it into your favorite text editor.

Tip http://www.jsoneditoronline.org/ is a handy resource for editing json files. This site provides an online editor to detect errors and indent source code, side-by-side with an editable expand-collapse tree view when the source has a correct syntax.

```
{"$schema":"http://json-schema.org/draft-04/schema#",
	"id":"http://jsonschema.net",
	"type":"object",
	"properties":{
		"addressItems":{
			"id":"http://jsonschema.net/address",
			"type":"array",
			"items":{
		"type":"object",
		"properties":{
			"address":{
			"type":"object",
			"properties":{
				"streetAddress":{
				"type":"string"
				},
				"city":{
				"type":"string"
				}
			},
			"required":[
				"streetAddress",
				"city"
				]
			}
		},
		"required":[
			"address"
			]
		}
	}
}
}
```

address_schema_out.json

```
{
"$schema":"http://json-schema.org/draft-04/schema#",
"id":"http://jsonschema.net",
"type":"object",
"properties":{
	"fullAddressItems":{
		"id":"http://jsonschema.net/address",
		"type":"array",
		"items":{
			"type":"object",
```

```
                        "properties":{
                                "address":{
                                "type":"object",
                                "properties":{
                                        "fullAddress":{
                                        "type":"string"
                                        }
                                }
                        }
                },
        "required":["address"]
        }
}
}
}
```

address.json file

```
{"addressItems": [
        {"address": {
                "streetAddress": "24 Guild Street",
                "city": "London"}
        },
        {"address": {
                "streetAddress": "999 Lucky Strike",
                "city": "Test_city"}
        },
        {"address": {
                "streetAddress": "221B Baker Street",
                "city": "London"}
        },
        {"address": {
                "streetAddress": "20 Forthlin Rd",
                "city": "Liverpool"}
        },
        {"address": {
                "streetAddress": "25 Brook Street",
                "city": "London"}
        }
        ]}
```

Data Map Editor 2

You can use the Data Map Editor to design the conversion between the source and target schemas of a map. The Design view is used to graphically design the map by linking source elements to target elements.

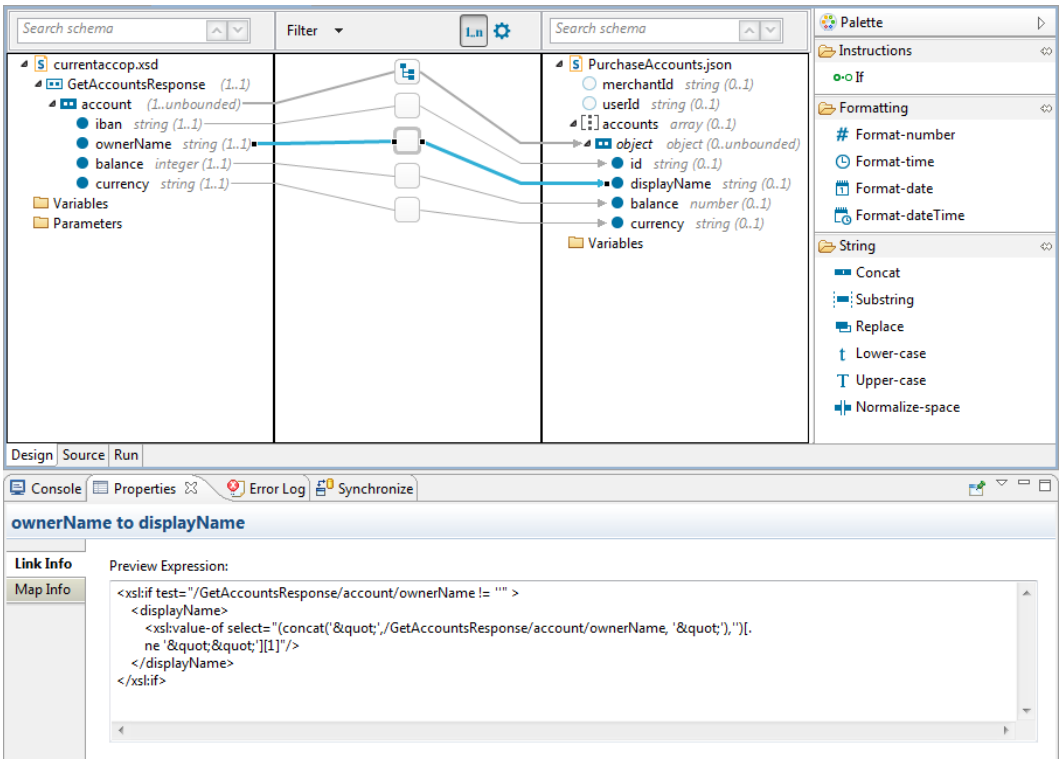
Design view

At the bottom of the main view, there are 3 tabs: **Design**, **Source**, and **Run**. This section describes the **Design** tab; Source and Run tabs are described in Source and run views.

The Design view of the editor is divided into three areas:

- The left pane (input) shows the list of source elements in the source schema(s), in tree format. This pane also shows the declared variables to be used with the conversion, and the declared external parameters. The toolbar can be used to expand and collapse the tree and view cardinalities for the source schema.
  It is possible to add multiple input files in the left part of the screen.
- The right pane (output) shows the list of target elements in the target schema in tree format. This pane also shows the declared variables. The toolbar can be used to expand and collapse the tree and show the types and cardinalities for the target schema.

- The middle pane shows the links representing how the source elements are converted to the target elements. You create links by drawing lines between the source and target elements.



The **Properties** tab, at the bottom of the window, displays information for the selected link, variable, or parameter. For more information, see *Properties view* on page 33.

The source and target trees show the name of the schema as their first root element. The child elements can be records or leafs. There are different icons for each type of item in the tree: records, leafs, attributes, objects, and arrays. The mandatory and optional items have a different icon.

You can view the generated XSLT code for a map from the Source view, and you can run a simulation to validate a created map from the Run view. For more information, see *Source and Run views* on page 36.

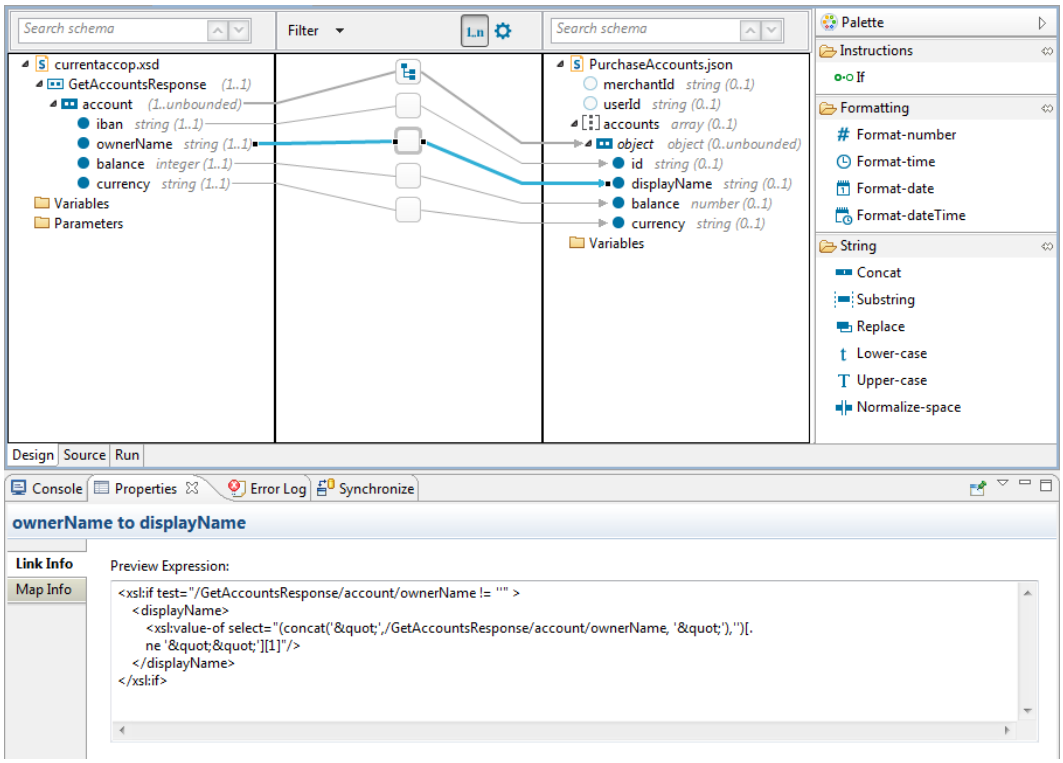
Structure and data links

You can use Structure links and Data links to describe how each element in the target schema can be created from the elements in the source schema.

- **Structure links**: The structure of the output message is defined by structure links. In the following example, there is a structure link from the `account` element in the source schema to the `accounts object` element in the target schema. This means that for each `account` element in the input message an `accounts object` element in the output message is generated.

- **Data links**: The values of the elements in the output message are defined by data links. In the following example, there is a data link from the `ownerName` element in the source schema to the `displayName` element in the target schema. This means that the value of the element `ownerName` in the input message is transferred to the element `displayName` in the output message.

Note Each element in the target schema can be defined by just one link.

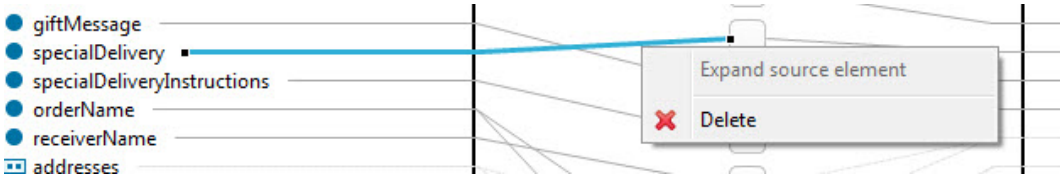


Create a link

To create a link, drag a source element and drop it on a valid target element. The cursor will change if the operation is not allowed.

Delete a link

To delete an existing link, click on the link, right-click, and select **Delete**.



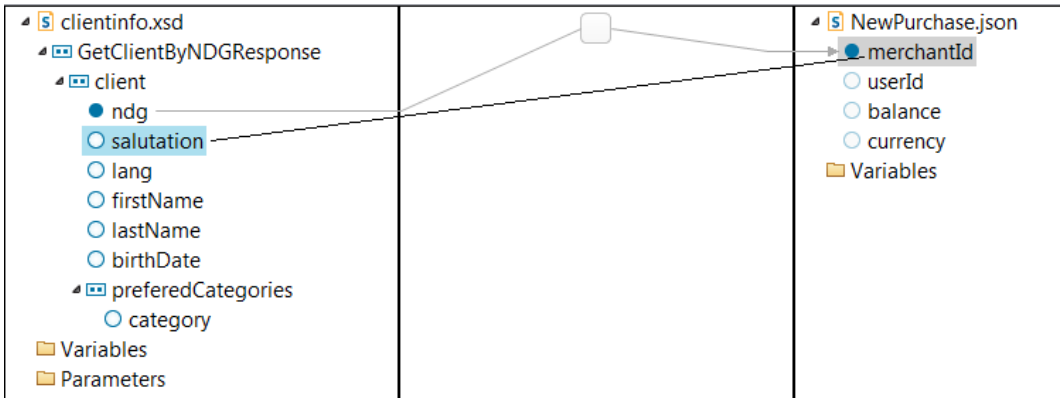
Alternatively, press your **Delete** key, or select **Edit > Delete** from the menu bar.

Reconnect a link

To reconnect existing links to other elements, drag the link source or target and drop it on to a new element.

Replace a link

To replace an existing link, drag a source element and drop it on the same target element used.



Reverse a change

For all operations, you can perform one of the following **undo** or **redo** options tp reverse a change:

- Select the **Undo** or **Redo** icon in the toolbar.
- Select **Edit > Undo** or **Edit > Redo** from the menu bar.
- Use the keyboard shortcuts **Ctrl+Z** or **Ctrl+Y**.

Functions

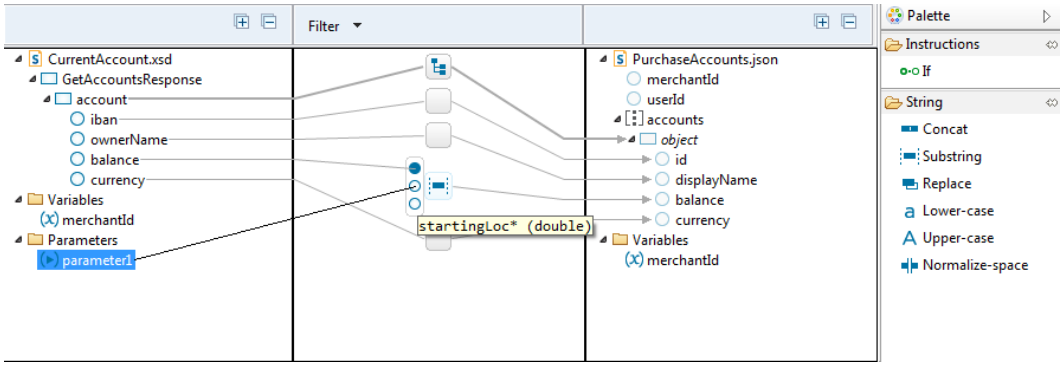
You can use functions (for example, string functions) to modify the values when the value of a source element cannot be directly linked to a target element.

Use functions on links

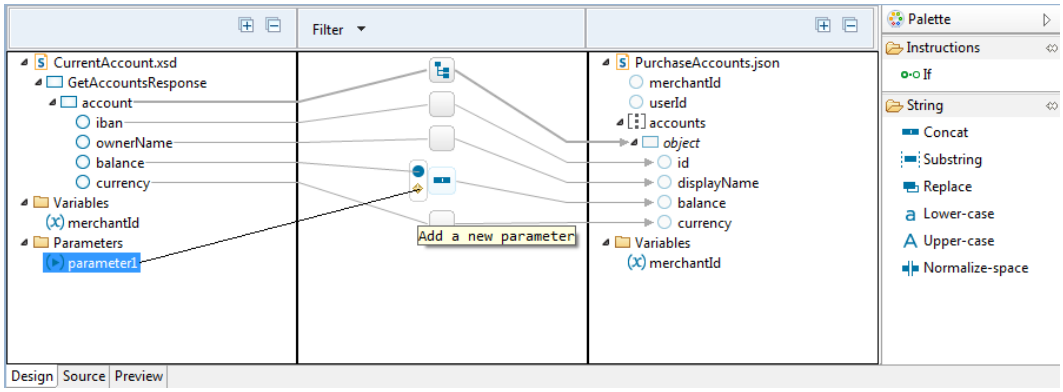
To use a string function:

1. Select the function (for example, **Substring**) from the palette on the right.
2. Drag a source element and drop it on a target element.
3. To use another input for the function, drag the element and drop it on the function.

When you hover over the function box of a function that allows more than one parameter, a parameter bar is displayed and you can drop the element on a specific parameter, for example:



If you use a function with an unbound parameter list (for example, **Concat**) the parameter bar includes a plus sign. You can drop the element on the plus sign to add it as a new parameter for the function, for example:



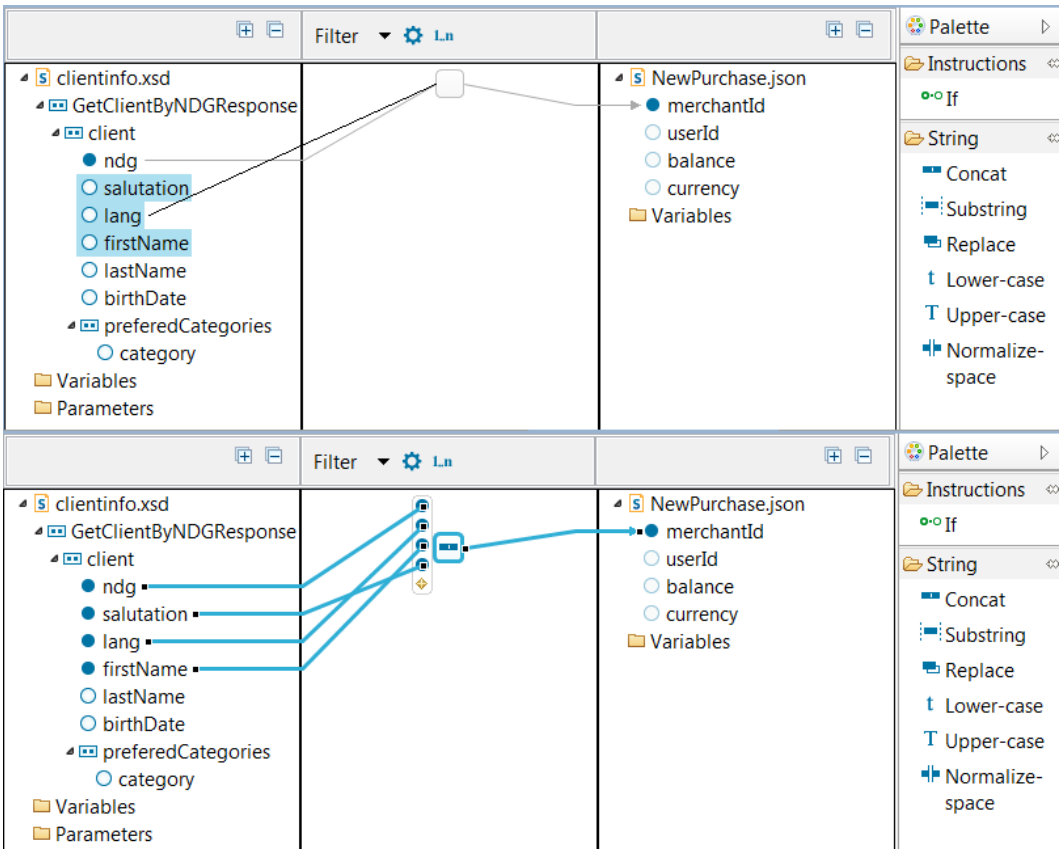
You can change the order of the parameter values by reconnecting the lines. You can drag the end of the line to reconnect, and drop it on a different parameter.

For more information on the supported string functions, see *Supported functions* on page 51.

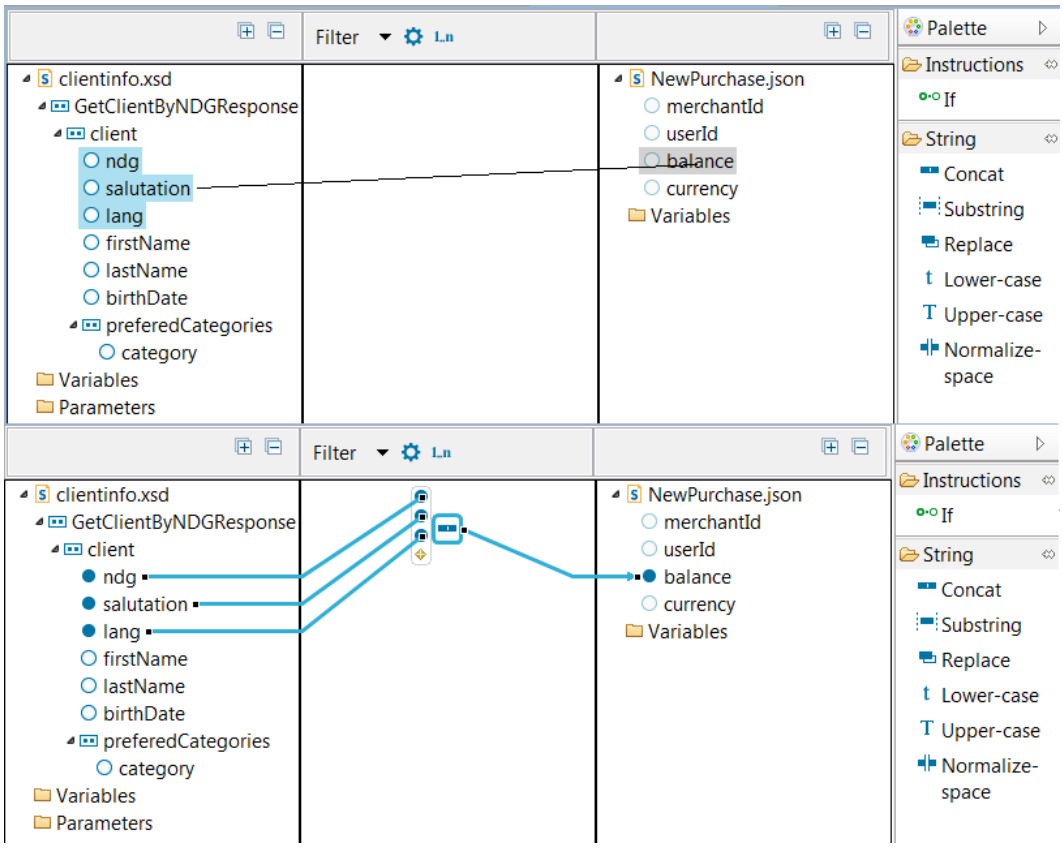
Create a string concat function

There are two options you can use to create a concat function:

- Option 1: Drag one or more source elements and drop them to an existing link box. The order of the parameters is dependent on the selection source elements order, for example:



- Option 2: Select at least two source elements and drag and drop them to a target element, for example:



To switch between functions, select a function from the palette and drop it on existing function box. You can also select another function from the context menu.

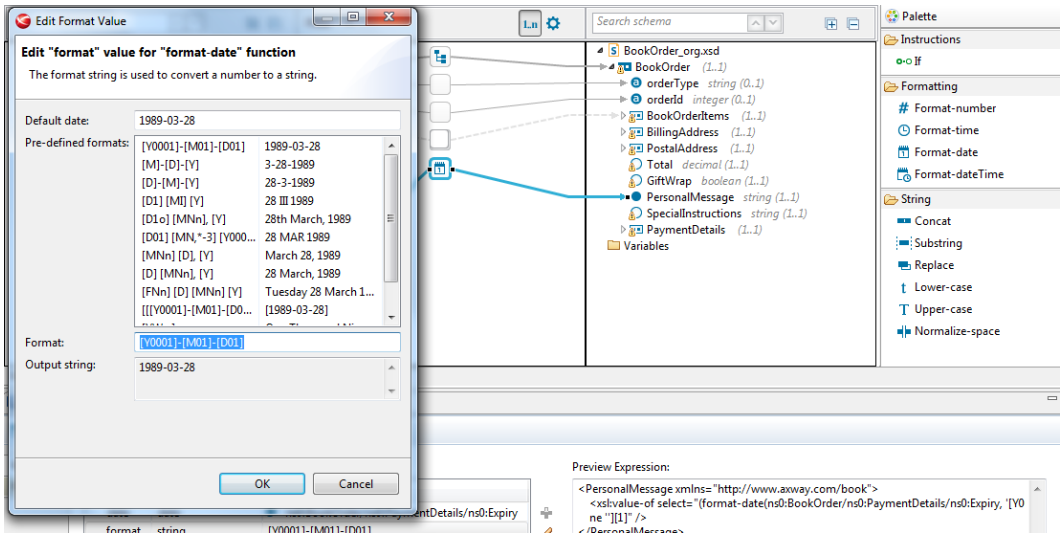
To add links for all parameters of a function, select multiple source elements and drag and drop them on and existing function box. The inputs are appended until the number of parameters limit is reached.

Note If you drop a source element over a function box, the source element link is considered the first unconfigured parameter, if one exists.

Apply the date and time function

To apply the date and time function:

1. Select the desired formatting function (for example, **Format-date**) from the palette on the right.
2. Drag a source element and drop it on a target element.



To choose a specific format for input:

1. Select the created link.
2. Select **Edit** from context menu and edit the format parameter from the **Edit function parameters** dialog.
3. Choose a pre-defined format or enter your own pattern.

For more information on the supported format functions, see *Format functions* on page 52.

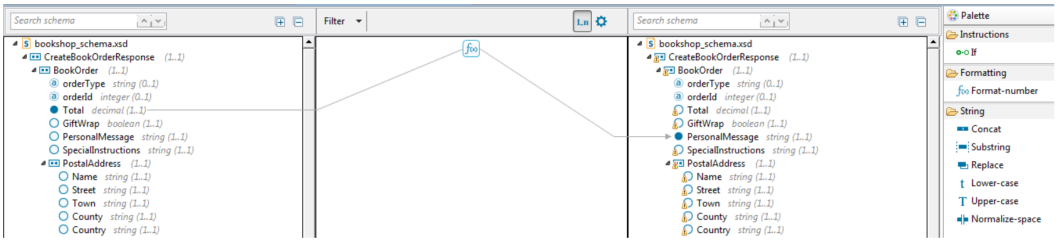
Apply the format-number function

The format-number function converts the given number into a string.

Note For a full list of supported format functions and input types, see *Format functions* on page 52.

To use the format-number function:

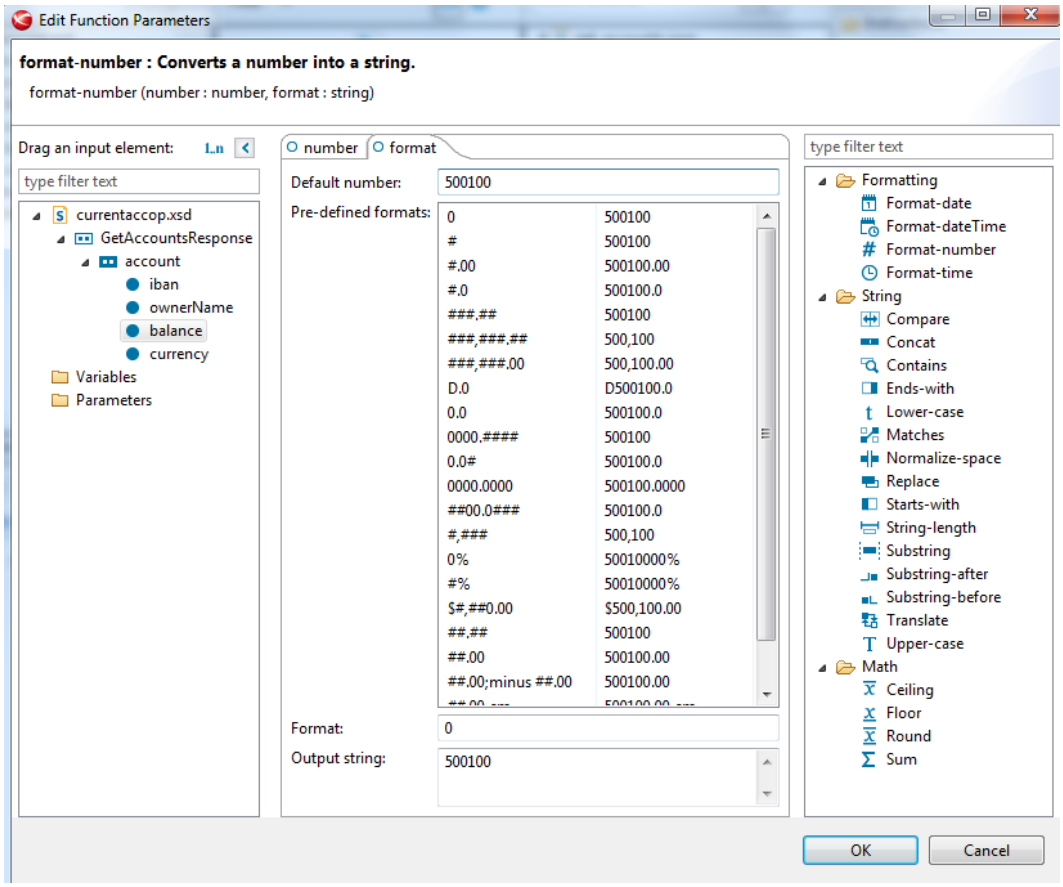
1. Select the **Format-number** function from the palette on the rights.
2. Drag a source element and drop it on a target element.



To choose a specific format for a number:

1. Select the **format-number** link.
2. Select **Edit** from context menu and edit the format parameter from the **Edit function parameters** dialog.

3. Choose a pre-defined pattern or enter your own pattern:



Note If your pattern is invalid, an error message is displayed in the output string.

Filters

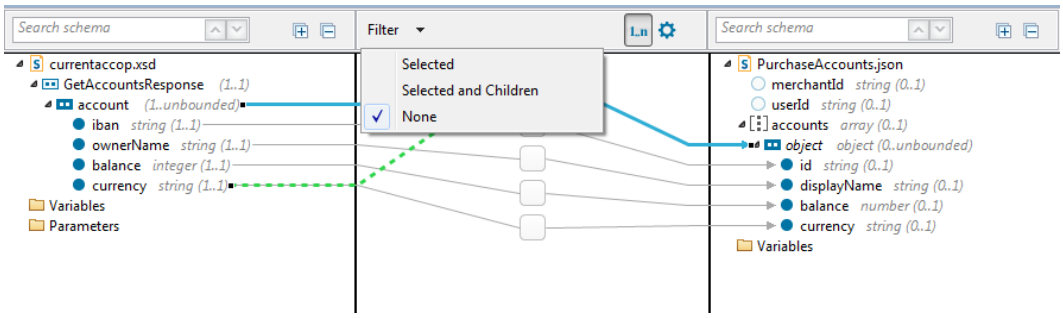
When you work with a large complex map in the Data Map Editor, the middle pane might contain many links. The **Filter** option enables you to focus on a subset of links and filter all other links out.

Apply a filter

To apply a filter, click the **Filter** button and choose from the following options:

- **Selected**: Show only links for the selected element.
- **Selected and Children**: Show only links for the selected element and its child elements.
- **None**: Show all links.

Note You can combine multiple filters.



If instruction

You can use the *if* instruction to check whether a given condition is satisfied before applying a link between a source element and a target element. In all, there are in all four elements involved:

- The source element of the link, in the left panel
- The target element of the link, in the right panel
- The link between the source and the target, represented by an element in the central panel, and its connections to the source and target elements
- The *condition element*, in the left panel, which is connected by a dotted line to the link element in the central panel

Visual Mapper checks whether the condition is satisfied (by default, whether the condition element is not empty) before mapping the source element to the target element.

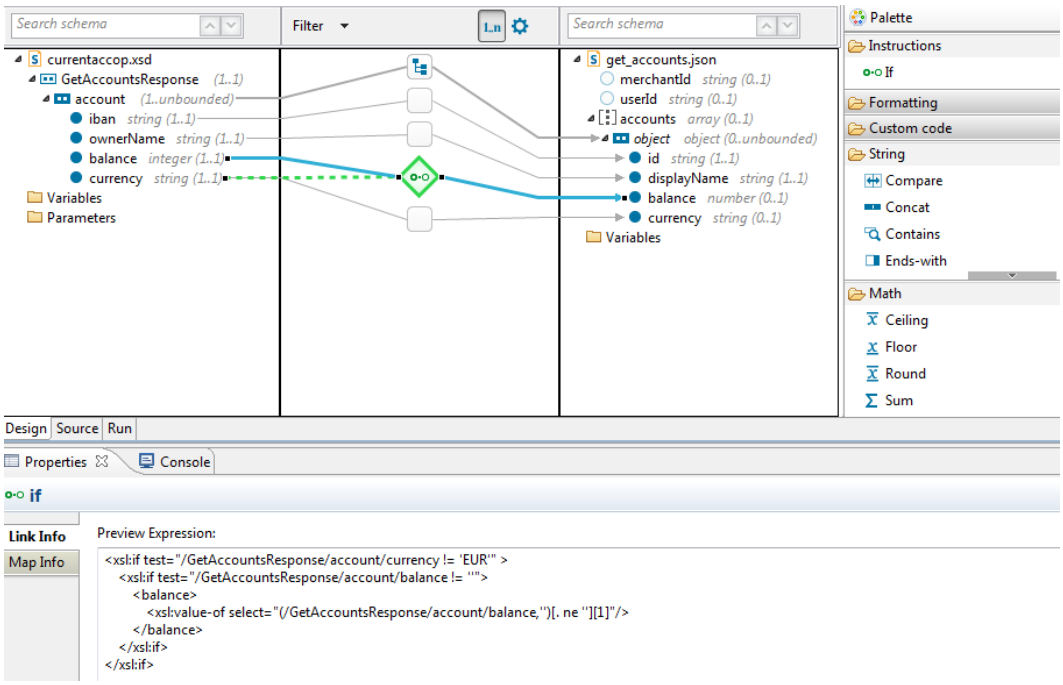
Apply the *if* instruction to a simple link

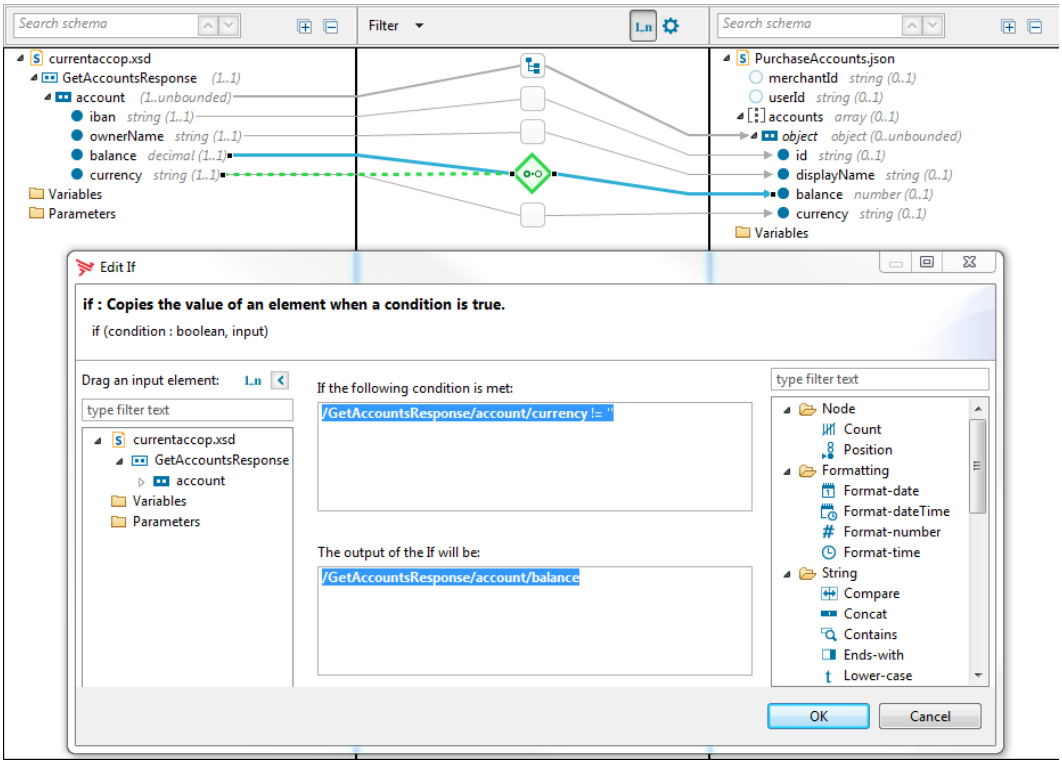
To apply the *if* instruction:

1. Select the **If** icon from the palette on the right
2. Select the *condition* element (left panel), then drag and drop it on a *link* element (central panel)
   Or you can replace steps 1 and 2 by Right-click on the link element > Set function > If
3. (Optional) You can then **double-click** on the link element,
   (or right-click the link element, then select **Edit...**).
4. The Edit If dialog box appears, it contains two parameters: *condition* (the condition element) and *input* (the source element)

You can change the condition element before you execute the map, by reconnecting the dotted line to a different element on the left panel, or by editing the condition parameter in the Edit IF dialog. In the Properties view you can see the generated XSL expression. See *Properties view* on page 33.

Note You can configure complex expressions for conditions in the parameter dialog; however, there is no validation of the input, and the existing links will be removed from UI.


Search schema
Filter
currentaccop.xsd
GetAccountsResponse (1..1)
account (1..unbounded)
iban string (1..1)
ownerName string (1..1)
balance integer (1..1)
currency string (1..1)
Variables
Parameters
get_accounts.json
merchantId string (0..1)
userId string (0..1)
accounts array (0..1)
object object (0..unbounded)
id string (1..1)
displayName string (1..1)
balance number (0..1)
currency string (0..1)
Palette
Instructions
If
Formatting
Custom code
String
Compare
Concat
Contains
Ends-with
Math
Ceiling
Floor
Round
Sum
Design
Source
Run
Properties
Console
if
Link Info
Map Info
Preview Expression:
<xsl:if test="/GetAccountsResponse/account/currency != 'EUR'" >
<xsl:if test="/GetAccountsResponse/account/balance != ''">
<balance>
<xsl:value-of select="(/GetAccountsResponse/account/balance,'')[. ne ''][1]"/>
</balance>
</xsl:if>
</xsl:if>


Search schema
Filter
currentaccop.xsd
GetAccountsResponse (1..1)
account (1..unbounded)
iban string (1..1)
ownerName string (1..1)
balance decimal (1..1)
currency string (1..1)
Variables
Parameters
PurchaseAccounts.json
merchantId string (0..1)
userId string (0..1)
accounts array (0..1)
object object (0..unbounded)
id string (0..1)
displayName string (0..1)
balance number (0..1)
currency string (0..1)
Variables
Edit If
if : Copies the value of an element when a condition is true.
if (condition : boolean, input)
Drag an input element:
type filter text
currentaccop.xsd
GetAccountsResponse
account
Variables
Parameters
If the following condition is met:
/GetAccountsResponse/account/currency != ''
The output of the If will be:
/GetAccountsResponse/account/balance
type filter text
Node
Count
Position
Formatting
Format-date
Format-dateTime
Format-number
Format-time
String
Compare
Concat
Contains
Ends-with
Lower-case
OK
Cancel

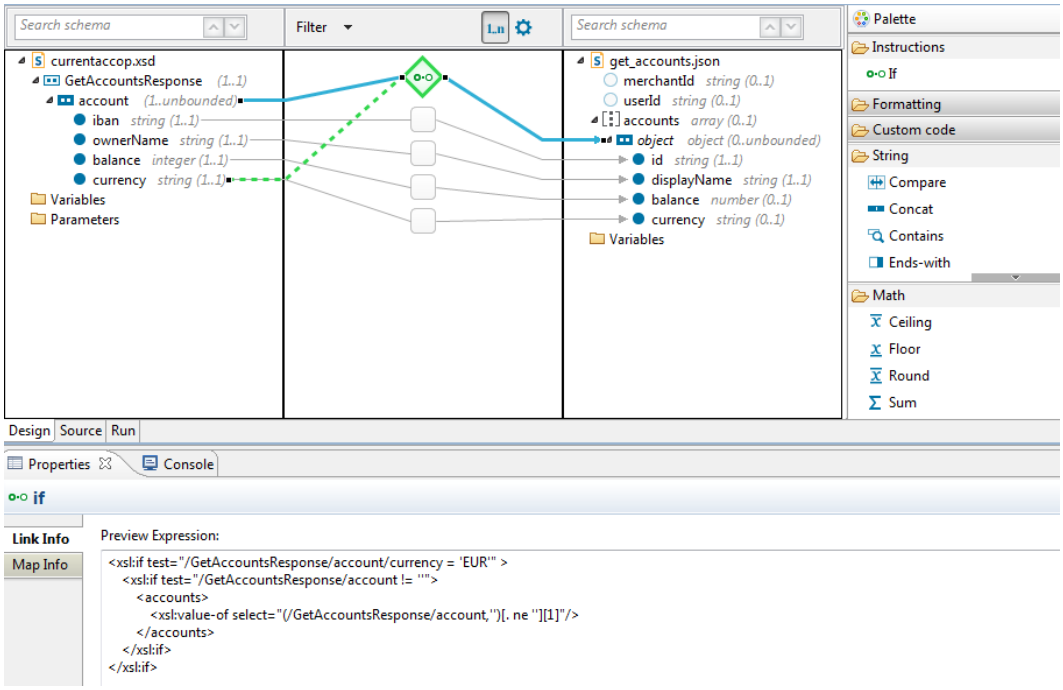
Apply the *if* instruction to a structural link

To apply the *if* instruction to a structural link, you can either use the drag-and-drop method described above (in *Apply the if instruction to a simple link* on page 27), or you can right-click on the link element in the central panel and use the contextual menu.

You can also change the condition element, as described above.

Example of using the if instruction

In following example, the account structure is mapped only if the currency is `EUR`:



Choose instruction

With the *choose* instruction, you can build an output in the right panel from complex expressions, depending on multiple conditions from the input. If none of the conditions are met, then a final expression can be targeted; it is known as *otherwise*. The output corresponding to each condition can combine multiple input values from the left panel, using operators such as concatenation.

The *choose* instruction is similar to the *If instruction* on page 27, with multiple conditions instead of one.

There are two possibilities to add a *choose* instruction. You can either

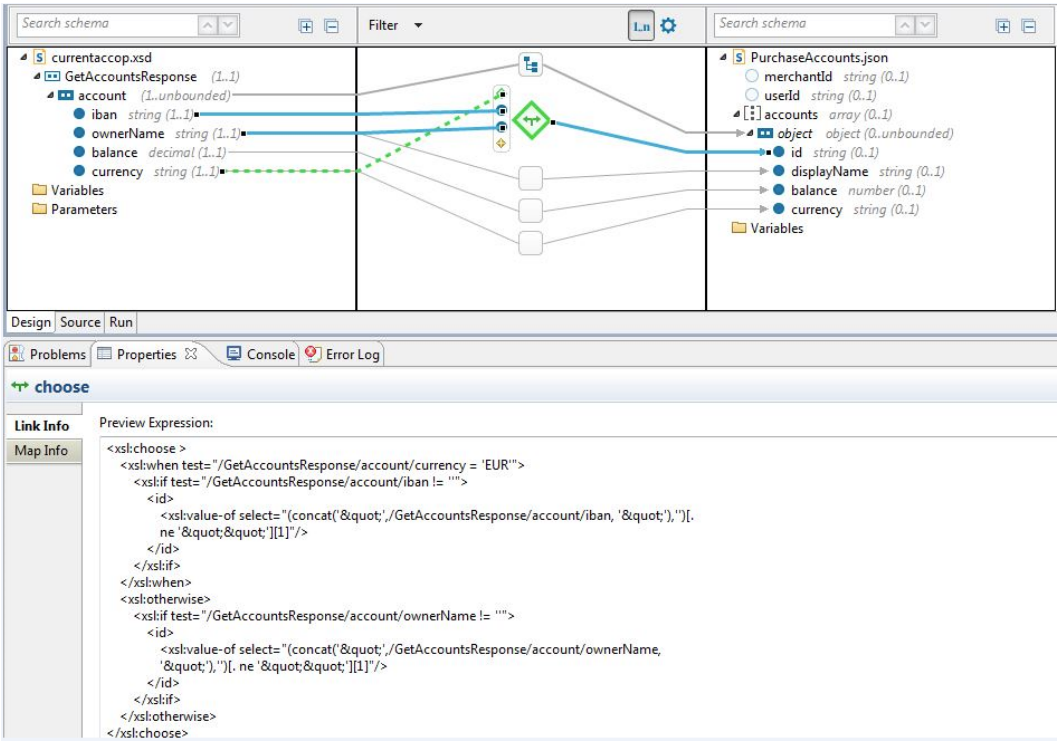
- Right-click on a link element, then select **Set Function** > **choose**.
  or
- Select the **choose** icon in the Palette (on the right hand side), then drag the first input element (in the left panel) and drop it on the target in the right panel.

You can then either:

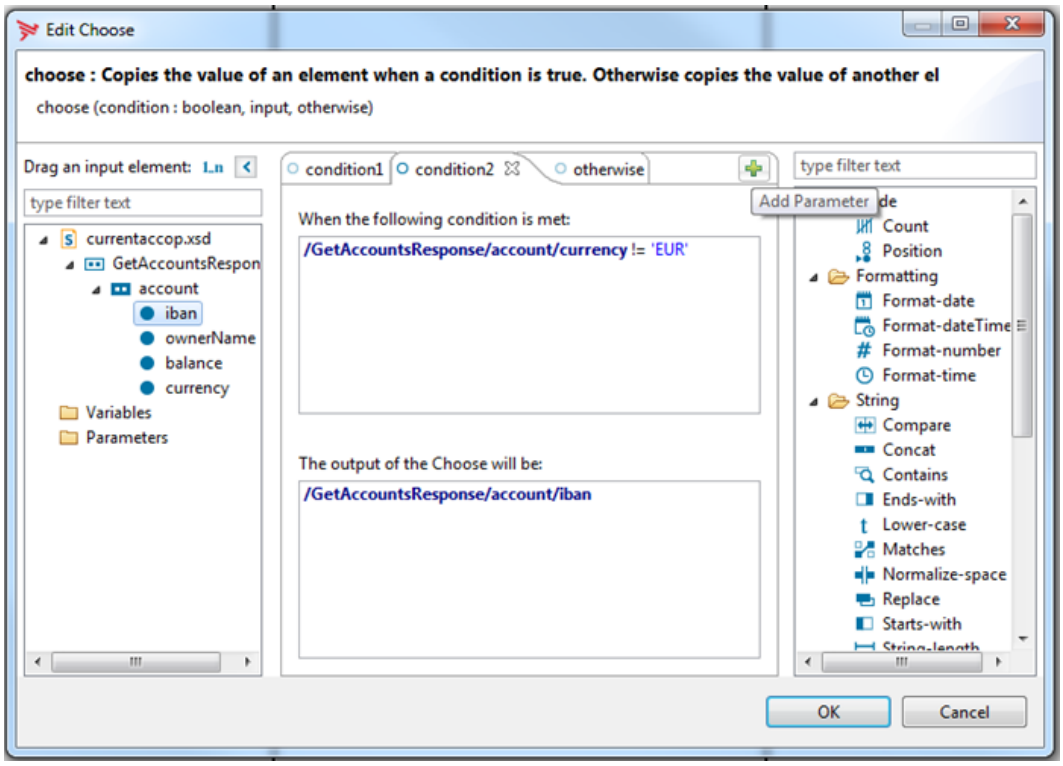
- Use drag and drop: drag more elements from the left panel to the *choose* link in the center, to serve as alternative inputs or as condition elements.
- Or open the **Edit choose** box, to edit the parameters in detail: right-click on the link element and select **Edit...** (or **double-click** the link element).

Example of using the *choose* instruction

In the following example, you can see several possible inputs on the left (`iban` and `ownerName`), connected to the **choose** element in the center by a solid blue line. The value of the output (`id`) can be either one, depending on the value of the condition (`currency`). The condition (on the left) is connected to the choose element by a green dotted line.

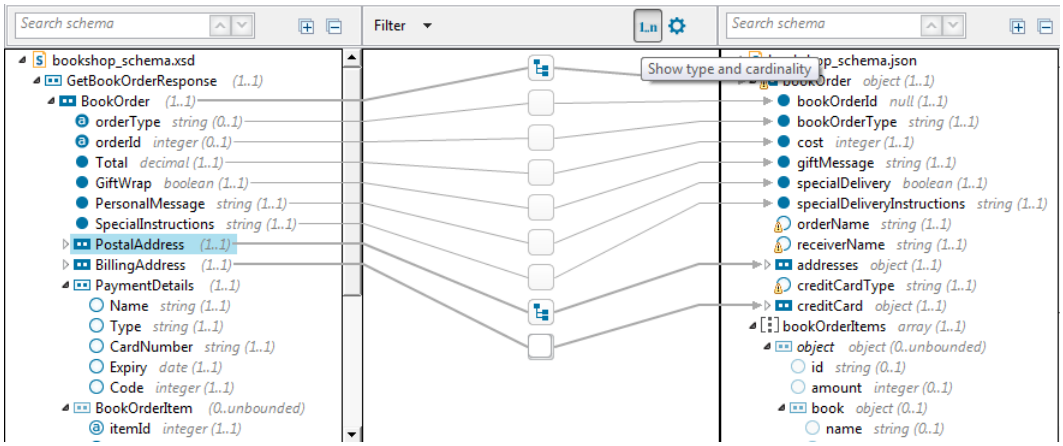


Below is the **Edit choose** dialog box corresponding to the example above. It is showing one of its conditions (`currency!='EUR'`) ; each condition is represented by a tab. The output corresponding to that condition can be the value of one of the input elements from the left side (here, `iban`), or it can be a complex expression derived from several inputs, connected by operators. You can drag elements from the left side into the top or bottom areas in the center, to build expressions for the condition and the output. . ....



Show element types and cardinalities

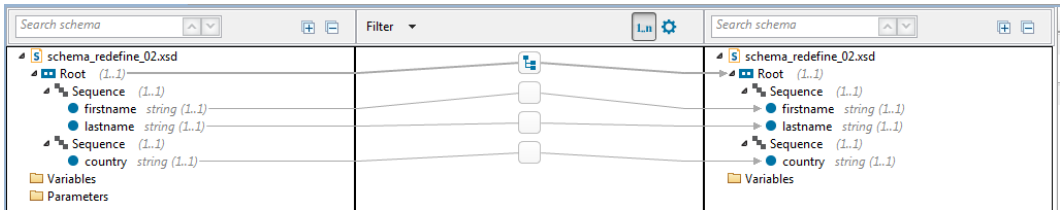
When you design a map in the Data Map Editor you can use the **Show type and cardinality** button on the toolbar to show the relationships and the data type of the elements in the source and target tree, for example:



Search in source or target schema

You can search for an element in the source or target schema.

When you enter the element name (partial or full) in the **Search schema** text box located above the source or target schema trees, a search is performed through the descendants of that element within the tree.



If an element is *not* found, the text box is displayed as red, along with a tool-tip.

If an element is found, it is highlighted in the tree. You can move through the expanded list using the up or down arrows.

Note From the Design tab, if you select **CTRL+f**, the search box is focused automatically. By default, the search is wrapped.

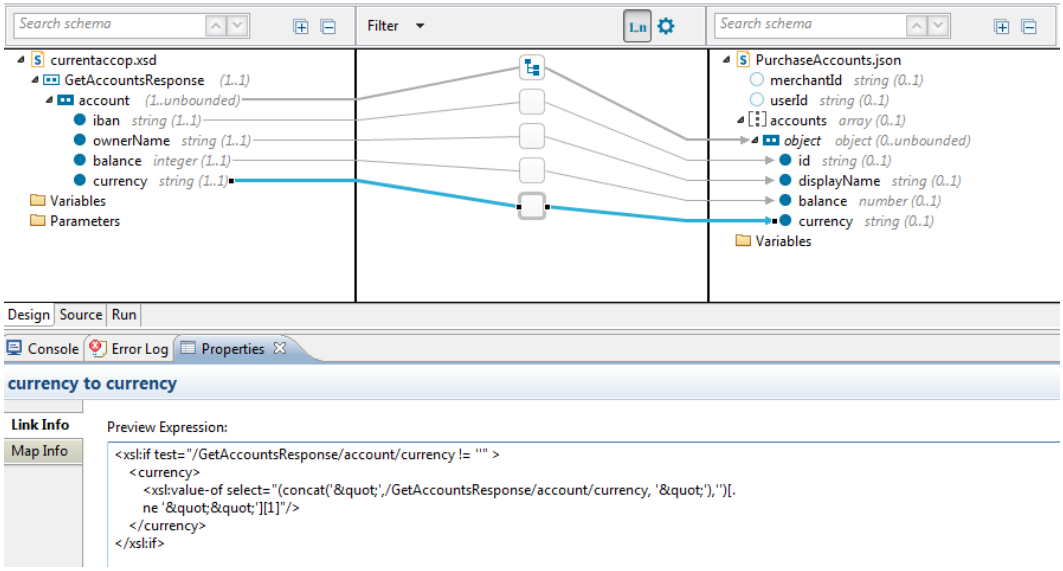
Properties view 3

The Properties vjiew enables you to view information about the currently selected link within the editor.

To view the information in the Data Map Editor, select the **Link** tab from the bottom of the **Properties** view.

For a structure or data link, a preview of the XSL expression that is generated for the link is displayed.

In the following screen, the currency element in the target is optional; therefore, an implicit *if* instruction is generated.



For a link with a function, only the XSL expression is displayed.

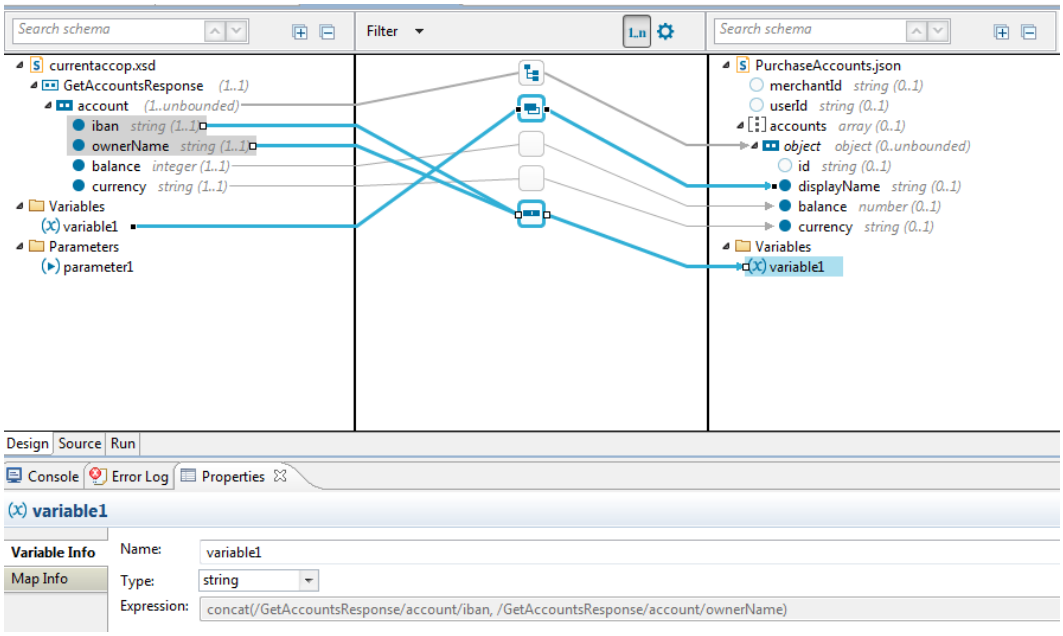
Variables and external parameters

4

The Data Map Editor enables you to add variables or parameters to a map.

- Variables enable you to store temporary values before the result will mapped to a target field.
- Parameters enable you to receive external values to customize the map during run-time.

You can also use variables to store intermediate results and use them as input for other functions. In the following example, *Variable1* takes the concat of *iban* and *ownerName*. This value can be used as input for a *replace* function. Also, you can use variables to map the same value to multiple output nodes.



Add a variable

To add a variable:

1. Right-click **Variables** on the left or right pane of the editor.
2. Select **Insert variable**. The new variable is opened in the Properties view.

3. Edit any of the following fields:
    - **Name**: Enter a unique name for the variable.
    - **Type**: Enter the type of the variable (string or integer).
    - **Expression**: Enter an expression for the variable. The expression can have a value either from the input schema or a typed in text.

Variables are shown in both the left pane (source elements) and right pane (target elements) of the editor. A link to a variable from a source element means that the value of the element will be stored in this variable. A link from a variable to a target element means that the value of the variable will be mapped to the target element.

Add a parameter

To add a parameter:

1. Right-click **Parameters** on the left pane of the editor.
2. Select **Insert parameter**. The new parameter is opened in the Properties view.
3. Edit any of the following fields:
    - **Name**: Enter a unique name for the parameter.
    - **Type**: Enter the type of the parameter (string or integer).
    - **Expression**: Enter an expression for the parameter. The expression is defined in the Execute Data Map filter and can have the value of a whiteboard variable, for example `${http.request.uri}` or a typed in text.
    - **Required**: Select the check box if the parameter is required. If the parameter is required, you cannot specify an expression. The value of the parameter is received at run-time.

Source and Run views 5

The Data Map Editor provides the following views for a mapping. You switch from one to the other using tabs at the bottom of the window:

- **Design**: Design a map graphically (default view).
- **Source**: View the generated XSLT code for a map (read-only).
- **Run**: Run a simulation to validate a created map.

Design view

Click the **Design** tab to design your map graphically. This is the default view, it is described in *Data Map Editor* on page 18

Source view

Click the **Source** tab to view the generated XSLT code for the map you have designed on the Design view. This view is read-only.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
#
# Stylesheet was automatically generated on Sep 30, 2015 12:37:52 PM
#
-->
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- Template for '/' -->
    <xsl:template match="/">
        <ACTIS>
            <xsl:processing-instruction name="xml-multiple">accounts</xsl:processing-instruction>
            <xsl:apply-templates select="GetAccountsResponse/account"/>
        </ACTIS>
    </xsl:template>
    <!-- Template for 'account' -->
    <xsl:template match="account">
        <accounts>
            <xsl:if test="iban != ''">
                <id>
                    <xsl:value-of select="(iban,'')[. ne ''][1]"/>
                </id>
            </xsl:if>
            <xsl:if test="ownerName != ''">
                <displayName>
                    <xsl:value-of select="(ownerName,'')[. ne ''][1]"/>
                </displayName>
            </xsl:if>
            <xsl:if test="substring(balance, 2) != ''">
                <balance>
                    <xsl:value-of select="(substring(balance, 2),'')[. ne ''][1]"/>
                </balance>
            </xsl:if>
            <xsl:if test="iban != '123'">
                <xsl:if test="currency != ''">
                    <currency>
```

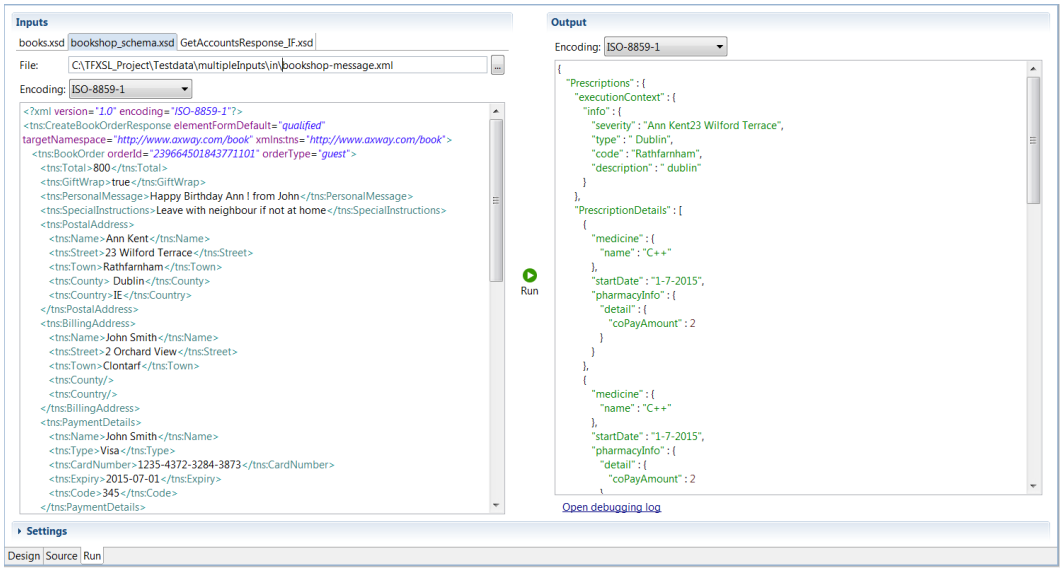
Note When you first create a map, the source code is generated for all mandatory elements. When you create links in the design view, the generated code is updated.

- The generated code for optional target elements contains a test *if* instructions; therefore, if it is `false` the elements are not generated.
- The generated code for mandatory target elements can have a value configured in the design view. If specified, it will contain the default value specified in the target schema or will contain no value.

Run view

With the Run view, you can run simulation to validate a created map.

Multiple inputs: the Run view presents one or several Inputs, and a single Output.



1. Click the **Run** tab.
2. Select the **Input file**(s).
    - If you want to run a simulation to validate a map with multiple inputs, select input files for all input schemas.
    - The **Run** button will become enabled only after you provide the exact number of input files required to run the simulation.
    - For example, if the map contains three input schemas (as shown in the screenshot above), you must select three input files in the **Run** view.

3. Click **Run**. The Input, Output, and Settings sections display the content of the selected Input file.
    - **Input**: (read-only) Displays the content of the input test message.
    - **Output**: (read-only) Displays the content of the output message (if the simulation is successful). If an error occurs (for example, incorrect input which has syntactical errors), an error message is displayed. In Settings, if **Log debugging information** is selected, the debug information is logged.
    - **Settings**: Allows you to edit the input and output message encodings. If parameters are defined in the map, their names and values display in the Parameters table.
4. Click **Open debugging log** to view the debug, warnings, and error messages.
5. In Settings, enter the values for each parameter in the table. This enables the simulation to run correctly.
6. Select **Log debugging information** to view the trace information in the log file after each simulation.

Validation and error handling 6

The Data Map Editor provides real time validation on operations to ensure the output message is valid against the target schema. You can use the following icons to assist with indicating the severity of the element problem:

Indicates errors

Indicates warnings

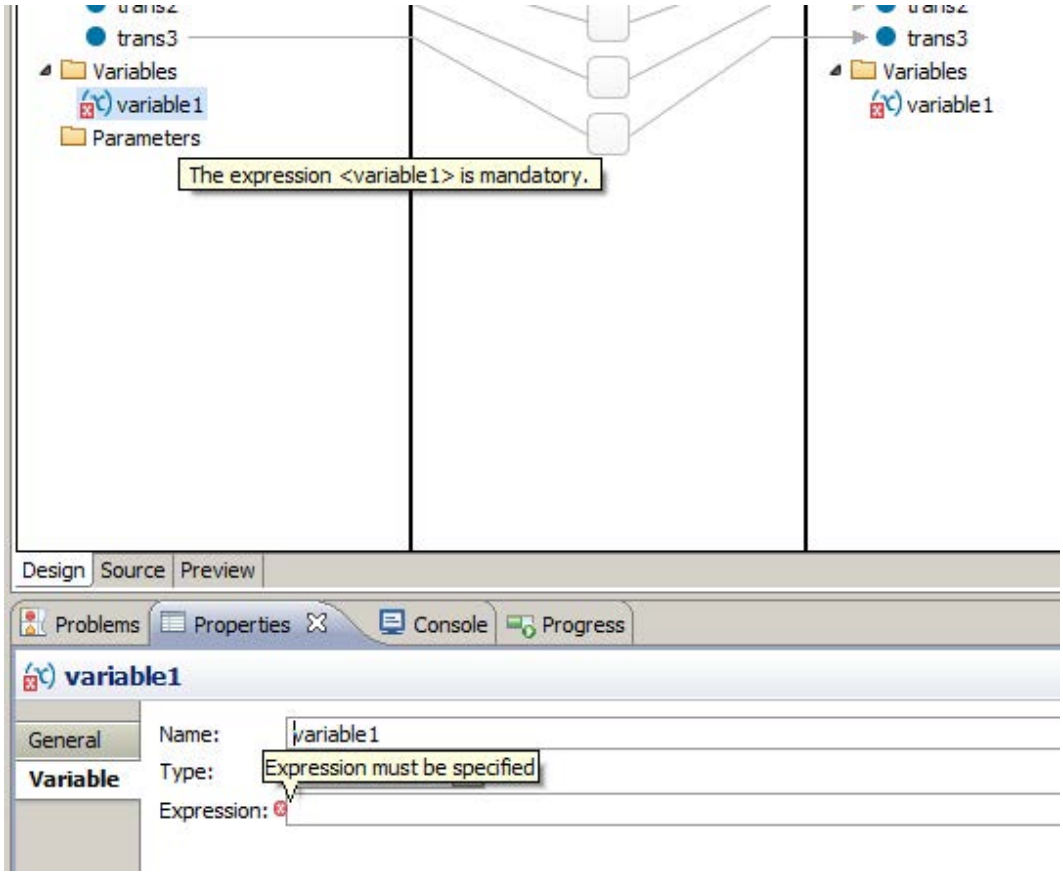
Indicates information

Note For each element that is in error, the tooltips indicate the action you need to perform to correct the error.

Errors

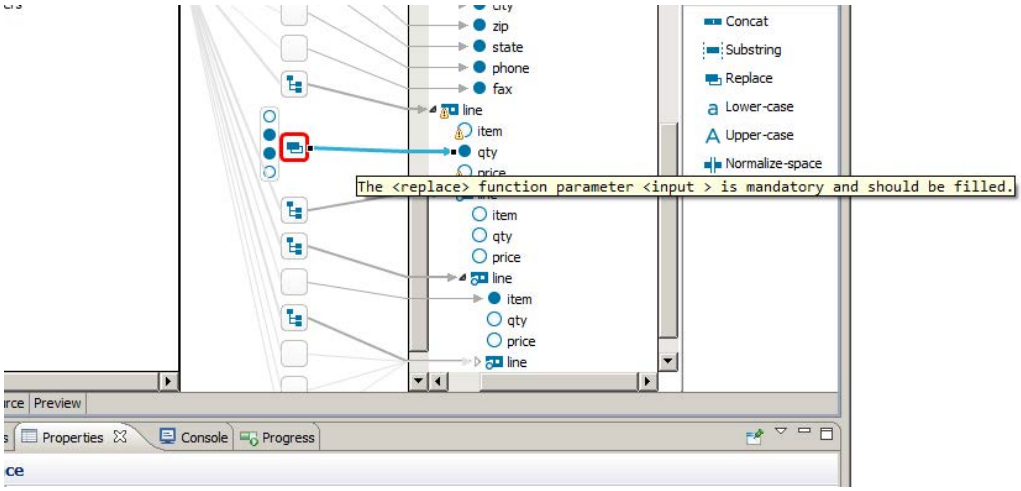
The error icon is present for variables and parameters that do not have the mandatory fields configured.

For example, the expression for `<variable1>` is mandatory; therefore, the error icon is displayed:



In the middle pane, the transformations that are not properly configured display with a red border.

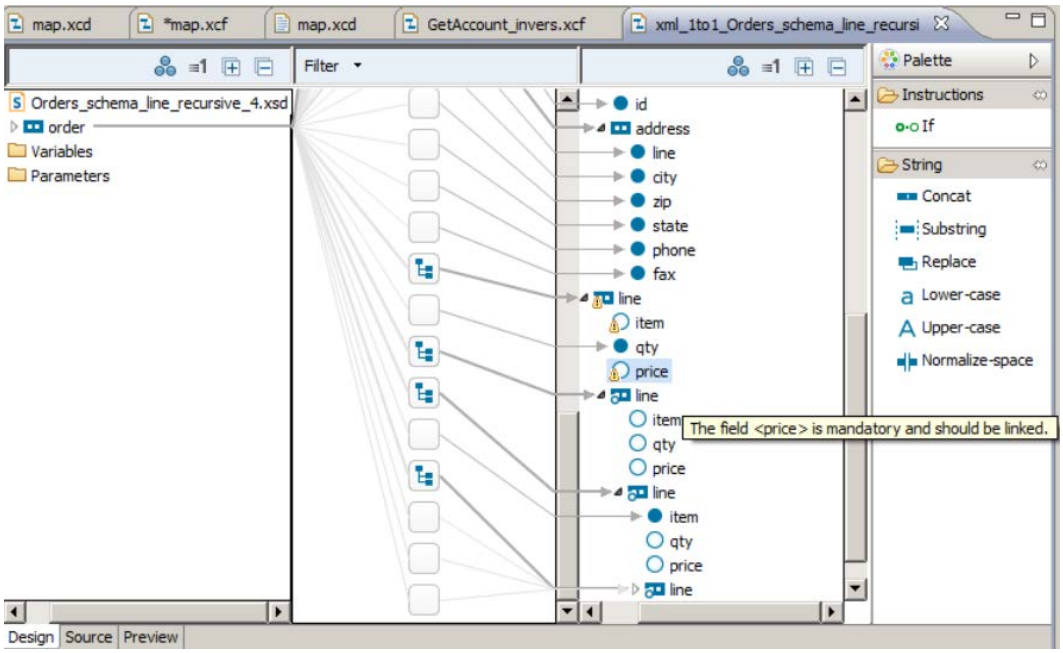
For example, the replace function parameter `<input>` is mandatory; therefore a red border is displayed:



Warnings

The warning icon is displayed when mandatory fields from target schema are not linked and do not contain default values.

For example, the field `<price>` is mandatory and must be linked; therefore, the warning icon is displayed:



Auto-mapping 7

You can use the auto-mapping feature to generate the mappings for the selected elements and their parents and children depending on the source and target structures.

Auto-mapping parents

When you create a data link, all repeating structures that contain the target node are identified and the structural links are automatically created between the source and target parents with multiple cardinality, in bottom-up order.

Note Multiple cardinalities have the upper limit of two, at a minimum.

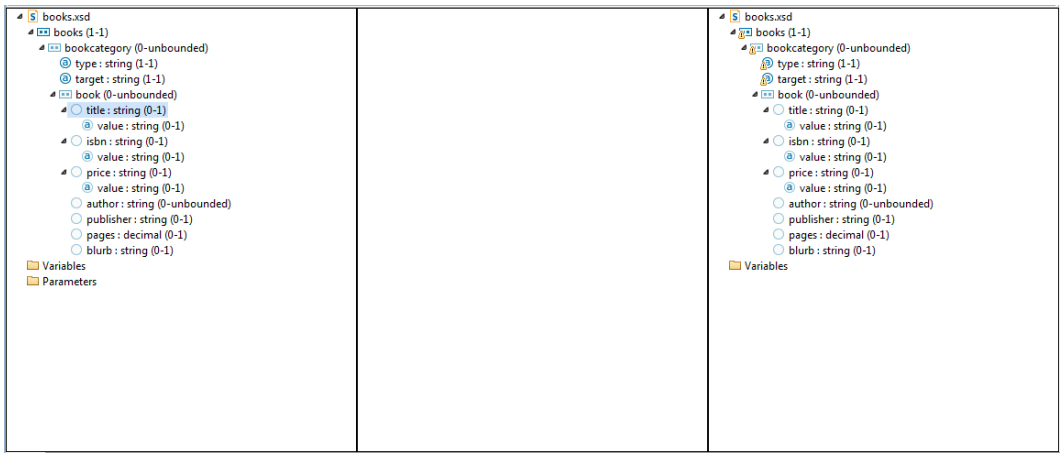
If a structural link cannot be automatically created for a target node, a warning is displayed on all upper parents indicating there is a repeating structure.

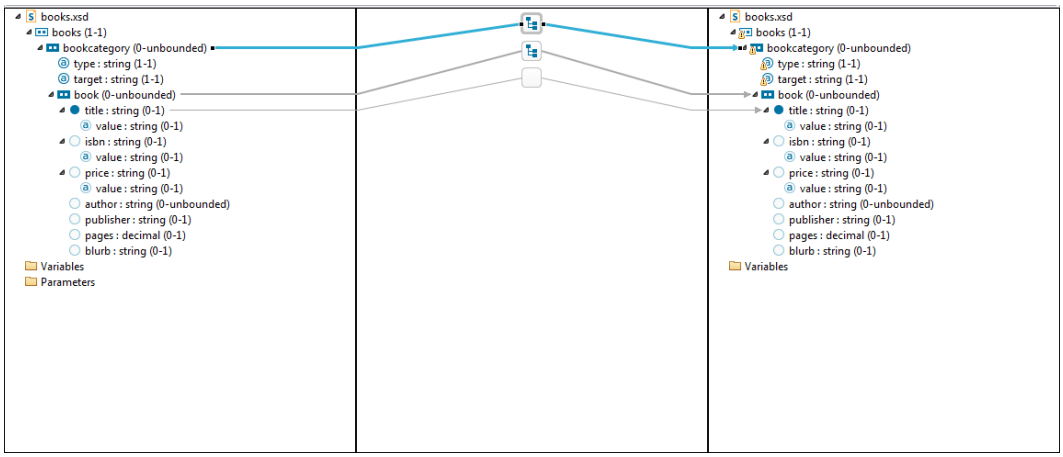
Note You may want to link the structural link to a repeating input structure.

If an element is located inside a parent with cardinality (for example, 0-unbounded) and that parent is not linked, the output will contain only one occurrence of the linked element. This may produce an undesired output form.

In the following example, the title is linked to the title element. The parents on the source (left side) with multiple cardinality are linked to the parents with multiple cardinality on the target (right side), in order, starting from the element's level:

- The source, book (0-unbounded), is linked to the target, book (0-unbounded).
- The source, bookcategory (0-unbounded), is linked to the target, bookcategory (0-unbounded).



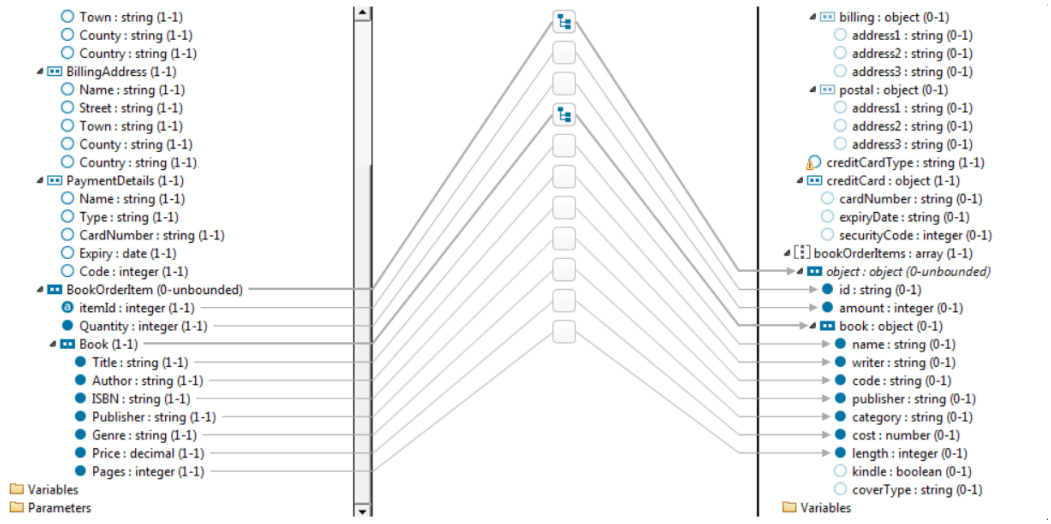


Auto-mapping children

When you create a structural link the mappings are generated for the selected elements and their children according to the following matching criteria:

- Name (case-sensitive or not)
- Type
- Cardinality
- In-depth search of source

When auto-mapping is enabled, you can draw a link from a source to a target element. When you drop the connection on the target, the source and target you select are connected. If the source and target have children elements, they are connected according to the matching criteria set in the Auto-mapping screen, for example:



Mapping Algorithm

The mapping algorithm works as follows:

- A link is created from the source to the target. If the source and target are both terminals or non-terminals, regardless of their names, types (if applicable), or cardinalities.
- If there is another link to the target, the link is replaced with the one from the selected source, regardless of the **Replace existing links** option.
- If the source is already used in another mappings, a new link is generated from the selected source to the target, without removing the existing ones.
- The algorithm attemps to find the best match for the children of source and target according to the matching criteria. For example, they must have the same name if **Match names** or if **Case sensitive** is selected, and also a compatible type, if **Match type** is selected, and the minimum cardinality of the source child is greater then or equal to the minimum cardinality of the target if **Match cardinality** is selected.
- A virtual node, such as an object and items in a JSON schema, has the name of its parent element; therefore, it is linked with the corresponding parent in source.
- If the target is a terminal having terminal children, such as a JSON object with value properties, and you connect it with a terminal that has children, such as an XML leaf with attributes, and there is only one child of the target that has the same type as the source leaf, then the remaining child of the target is connected to the source leaf.
- After you generate the mappings for the selected source and target and also for their children, the algorithm connects the parents of the linked elements.

Troubleshooting

The following provides some troubleshooting tips for children:

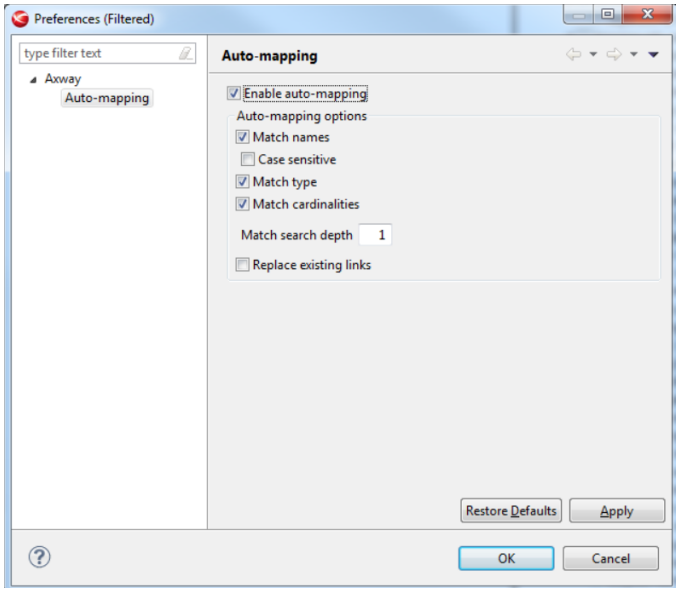
- **Different names**: If children with different names are not linked, uncheck **Match names** from the Auto-mapping screen.
- **Same names**: If children with the same names, but have different cases, are not linked, check **Match names** and uncheck **Case sensitive** from the Auto-mapping screen.
- **Different types**: If children with different types are not linked (such as string source to integer target), uncheck **Match type** from the Auto-mapping screen.
- **Different cardinalities**: If children with different cardinalities are not linked, uncheck **Match cardinality** from the Auto-mapping screen.
- **Old links**: If old links are not replaced when trying to map children, check **Replace existing links** from the Auto-mapping screen.

- **Value properties**: If the source is an XML leaf with attributes that you want to connect with a JSON object that has value properties, but the source leaf is not linked with any of the target properties, make sure you have only one value property in the target object that has the same type as your source leaf.
- **Non-terminal elements**: If children of a non-terminal element contained by another non-terminal element are not linked with similar children of the non-terminal target element, increase the **Match search depth** from the Auto-mapping screen. You can also try to link similar a structure together, instead of drawing the link from its parent element.

Configure auto-mapping settings

To configure the auto-mapping settings:

1. Select the **Settings** icon, or go to **Window > Preferences > Auto-mapping**. The Auto-mapping screen is displayed:



2. Select the appropriate auto-mapping settings:
   - **Enable auto-mapping**: uncheck this box to disable auto-mapping completely
   - **Match names**: Only the children with the same name are linked, regardless of their case. The default value is on.
   - **Match names - Case sensitive**: Only the children with the same name and same case are linked. The default value for Match Names is on and for Case Sensitive is off.
   - **Match type**: The source must contain a compatible type with the target. Numeric, date and time, and binary values can only be assigned from values of the same family. For example, an integer value can be assigned from byte, short, int, integer, decimal or numeric value and a string value can be assigned with any value. The default value is on.

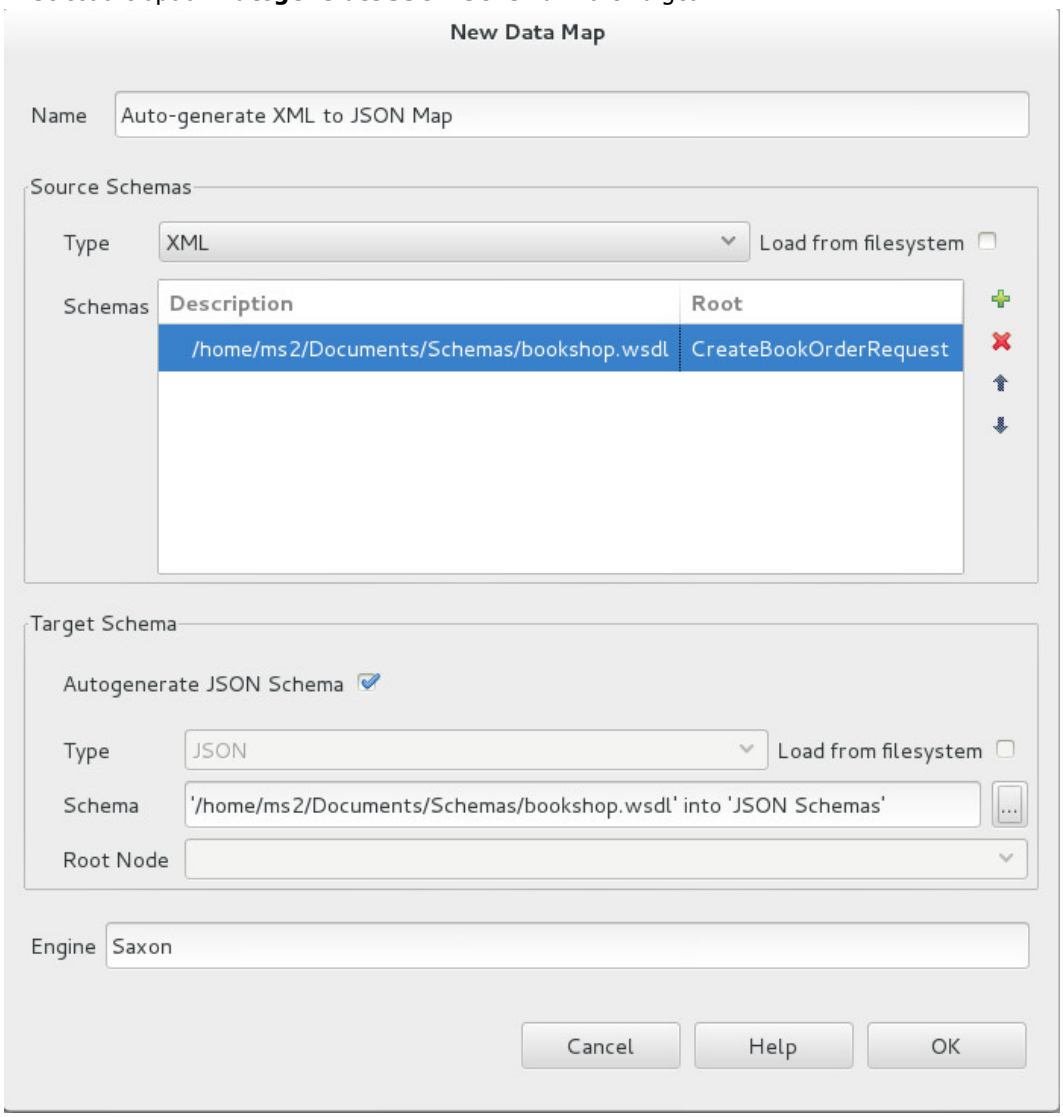
- **Match cardinalities**: The minimum cardinality of the source terminal must be greater than or equal to the minimum cardinality of the target in order for the link to be generated between source and target terminal elements. When the this criterion is off, the target can be linked from an optional source. The default value is on.
- **Match search depth**: Reduces the depth of source tree when you try to link with the target. First, the mapping algorithm will attempt to generate the links between the source and target container as they are. Then, it will reduce the parent of the source and take its children as if they were at the same level with their parent and try to map them to the target elements. This will continue until the value of match search depth is reached. The default value is 1.
- **Replace existing links**: If some of the target children are already linked with other source elements, these links are replaced with the ones generated automatically using the children elements of the selected source. This may override links you had explicitly generated for the target children. The default value is off.

Auto-generate XML to JSON map from XSD schema

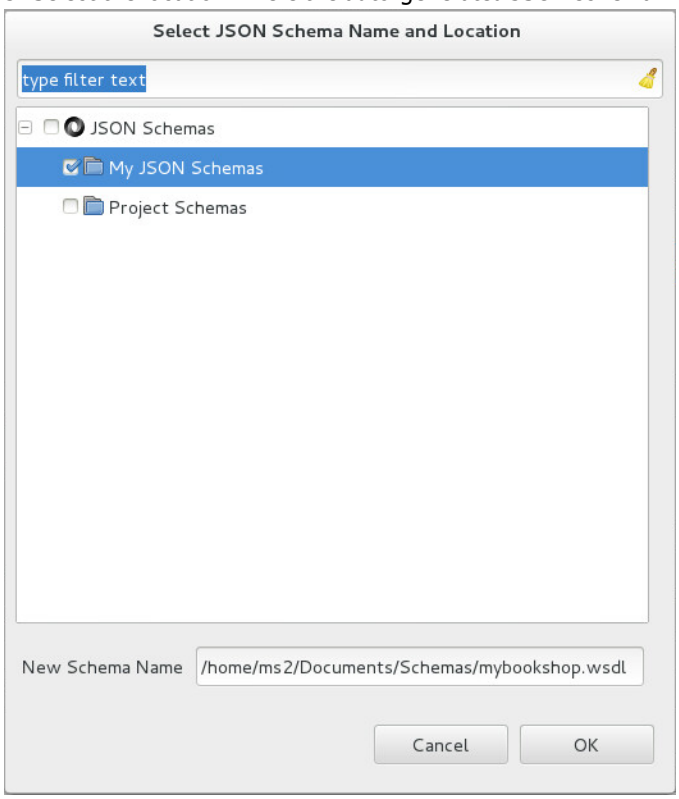
To auto-generate an XML-to-JSON map from an XSD schema specified at input, proceed as follows:

1. Open **Add New Data Map** wizard and provide a description of the input message (source schema). This must include: the type of the message (XML), the schema, and the root element.

2. Select the option **Autogenerate JSON Schema** in the Target:



3. Select the location where the auto-generated JSON schema will be saved:



Custom XSLT code 8

If you are familiar with Extensible Stylesheet Language Transformations (XSLT) code you can use the CustomXsltCode option from the Visual Mapper palette to enrich the transformation using expressions. This option is useful if you are an XSLT code expert and if the existing functions do not meet your use cases.

To create custom XSLT code, you can:

- Drop the CustomXsltCode from the Visual Mapper palette over child elements (leafs and records) and attribute target or exsiting links, or
- Add the CustomXsltCode from the context menu (from **Set function -> Custom code -> CustomXsltCode**).

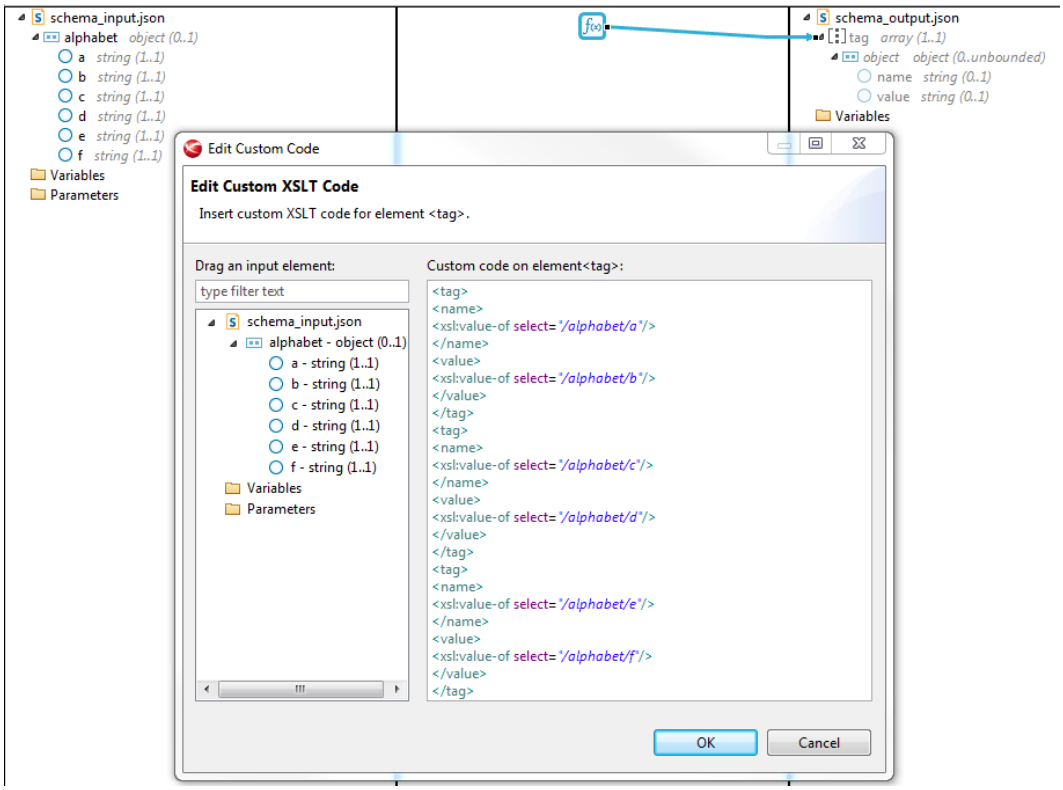
Note If the CustomXsltCode is placed on a record, all its children are grayed out and cannot be linked.

To edit the custom XSLT code, you can:

- Double-click on the Custom XSLT box, or
- Use the context menu (right-click on the Custom XSLT box and select Edit).

Caution You are responsible for the validity of this custom code; therefore, it is neccessary to simulate the created map before you run it on the server.

For example, in the output, if you have an element with multiple cardinality and you want to map the input elements to the output elements in a certain order, you can drag and drop elements from the source schema into the expression editor area:



Multiple Inputs support

Visual Mapper supports multiple file inputs. Running simulations on multiple file inputs is described in *Run view* on page 37.

Supported functions 9

Ths section describes the supported functions available from the Visual MapperPalette, including:

It also describes how edit a function with *Complex expressions* on page 55.

Conversion functions

The following conversion functions are supported:

- **boolean**: converts arguments to Booleans according to the following rules:
  - If the argument is a negative or positive number, it is converted to a Boolean value *true*. If the argument is zero or an NaN value, it is converted to *false*.
  - If the argument is a non-empty node-set, it is converted to *true*. An empty node-set is converted to *false*.
  - If the argument is a non-empty string, it is converted to *true*. An empty string is converted to *false*.
  - If the argument is an object of a type other than the four basic types (node-set, Boolean, number, or string), it is converted to a Boolean in a way that is dependent on that type.
  - Full description: https://www.w3.org/TR/xpath-functions/#func-boolean
- **number**: converts its argument to a number as follows:
  - A string that consists of optional white space, followed by an optional minus sign, followed by a number, followed by white space is converted to the IEEE 754 number that is nearest to the mathematical value represented by the string (according to the IEEE 754 round-to-nearest rule)
  - Any other string is converted to *NaN*.
  - Boolean *true* is converted to *1*; Boolean *false* is converted to *0*.
  - A node-set is first converted to a string and then converted in the same way as a string argument.
  - An object of a type other than the four basic types (node-set, Boolean, number, or

string) is converted to a number in a way that is dependent on that type.

- Full description: https://www.w3.org/TR/xpath-functions/#func-number
  - **string**: returns the string value of the argument. The argument could be a number, boolean or node-set.
    - Full description: https://www.w3.org/TR/xpath-functions/#func-string

Format functions

The following format functions are supported:

- **Format-dateTime**: Formats the given dateTime and returns a string.
- **Format-date**: Formats the given date and returns a string.
- **Format-time**: Formats the given time and returns a string.
- **Format-number**: Formats the given number and returns a string. For detail on how to use the format-number function.

For a full description, see the XSL Transformation (XSLT) Version 2.0.

Input types

The following input types for date/time and format-number functions are supported:

Date/time functions

The following xsd and json input types are supported for format date/time functions:

- XSD
  - xs:date
  - xs:dateTime
  - xs:time
- JSON
  - String object having format:date property
  - String object having format:date-time property
  - String object having format:time property

Format-number functions

The following xsd and json input types are supported for format-number functions:

- XSD
    - byte
    - decimal
    - int
    - integer
    - long
    - negativeInteger
    - nonNegativeInteger
    - nonPositiveInteger
    - positiveInteger
    - short
    - unsignedLong
    - unsignedint
    - unsignedShort
    - unsignedByte
    - float
    - double
- JSON
    - integer
    - number

Math functions

- **ceiling**: Returns the smallest integer that is not less than $arg. The result type depends on the numeric type of the argument. If $arg is untyped, it is cast to xs:double.
  Full description: http://www.w3.org/TR/xpath-functions/#func-ceiling
- **floor**: Returns the largest integer not greater than $arg. The result type depends on the numeric type of the argument. If $arg is untyped, it is cast to xs:double.
  Full description: http://www.w3.org/TR/xpath-functions/#func-floor
- **round**: Rounds a numeric value to the nearest integer. If the decimal portion of the number is .5 or greater, it is rounded up to the greater whole number (even if it is negative); otherwise, it is rounded down. The result type depends the numeric type of the argument. If $arg is untyped, it is cast to xs:double.
  Full description: http://www.w3.org/TR/xpath-functions/#func-round
- **sum**: Sums the values in a sequence. The $arg sequence can contain a mixture of numeric and untyped values. Numeric values are promoted, as necessary, to make them all the same type. Untyped values are cast as numeric xs:double values. The function can also be used on

duration values; the $arg sequence can contain all xs:yearMonthDuration values or all xs:dayTimeDuration values (not a mixture of the two). The $zero argument allows you to specify an alternate value for the sum of the empty sequence. If $arg is the empty sequence, and $zero is provided, the function return $zero. The $zero argument could be the empty sequence. The integer 0, the value NaN, a duration of zero seconds, or any other atomic value.
Full description: http://www.w3.org/TR/xpath-functions/#func-sum

Node functions

- **count**: returns an integer representing the number of nodes in a node-set.
  Full description: http://www.w3.org/TR/xpath-functions/#func-count
- **position** : returns an integer representing the position (starting with 1, not 0) of the current context item within the context sequence (the current sequence of items being processed).
  Full description: https://www.w3.org/TR/xpath-functions/#func-position

String functions

The following string functions are supported:

- **compare**: Compares the ASCII codes of two strings. Returns -1 if $comparand1 is less than $comparand2, 0 if $comparand1 is equal to $comparand2, or 1 if $comparand1 is greater than $comparand2 (according to the rules of the collation that are used).
  Full description: http://www.w3.org/TR/xpath-functions/#func-compare
- **contains**: Returns true if $arg1 contains the characters of $arg2 anywhere in its contents, including at the beginning or end.
  Full description: http://www.w3.org/TR/xpath-functions/#func-contains
- **concat**: Concatenates the given strings and returns a string.
  Full description: http://www.w3.org/TR/xpath-functions/#func-concat
- **ends-with**: Returns an xs:boolean value indicating whether one string ($arg1) ends with the characters of a second string ($arg2).
  Full description: http://www.w3.org/TR/xpath-functions/#func-ends-with
- **lower-case**: Returns the value of the given string after it translates every character to its lower-case equivalent as defined in the appropriate case mappings section in the Unicode standard.
  Full description: http://www.w3.org/TR/xpath-functions/#func-lower-case
- **matches**: Returns true if the $input matches the $pattern; otherwise, returns false.
  Full description: http://www.w3.org/TR/xpath-functions/#func-matches
- **normalize-space**: Returns the value of the given string with a white space normalized by stripping leading and trailing whitespace, and replacing sequences of one or more than one whitespace character with a single space, #x20.

  Full description: http://www.w3.org/TR/xpath-functions/#func-normalize-space
- **replace**: Replaces parts of a string that matches it with a regular expression.Full description: http://www.w3.org/TR/xpath-functions/#func-replace

- **starts-with**: Returns an xs:boolean value indicating whether one string ($arg1) starts with the characters of a second string ($arg2).
  Full description: http://www.w3.org/TR/xpath-functions/#func-starts-with
- **string-length**: Returns an xs:integer value indicating the number of characters in the string. Whitespace is significant; leading and trailing whitespace characters are counted.
  Full description: http://www.w3.org/TR/xpath-functions/#func-string-length
- **substring**: Returns the portion of the value of $sourceString beginning at the position indicated by the value of $startingLoc and continuing for the number of characters indicated by the value of $length. The characters returned do not extend beyond $sourceString.
  If $startingLoc is zero or negative, only those characters in positions greater than zero are returned.
  Full description: http://www.w3.org/TR/xpath-functions/#func-substring
- **substring-after**: Extracts all characters of a string ($arg1) that appear after the first occurence of another specified string ($arg2).
  Full description: http://www.w3.org/TR/xpath-functions/#func-substring-after
- **substring-before**: Extracts extracts all the characters of a string ($arg1) that appear before the first occurence of another specified string ($arg2).
  Full description: http://www.w3.org/TR/xpath-functions/#func-substring-before
- **translate**: Replaces individual characters of a string with other individual characters. The $mapString argument is a list of characters to be changed, and $transString is the list of replacement characters. Each character in $mapString is replaced by the character in the same position in $transString. If $mapString is longer than $transString, the characters in $mapString that have no corresponding character in $transString are not included in the result. Characters in the original string that do not appear in $mapString are copied to the result unchanged.
  Full description: http://www.w3.org/TR/xpath-functions/#func-translate
- **upper-case**: Returns the value of the given string after it translates every character to its upper-case equivalent as defined in the appropriate case mappings section in the Unicode standard.

  Full description: http://www.w3.org/TR/xpath-functions/#func-upper-case

Complex expressions

A function parameter can be defined as a complex expression of XSLT functions. This returns the same type as the edited parameter.

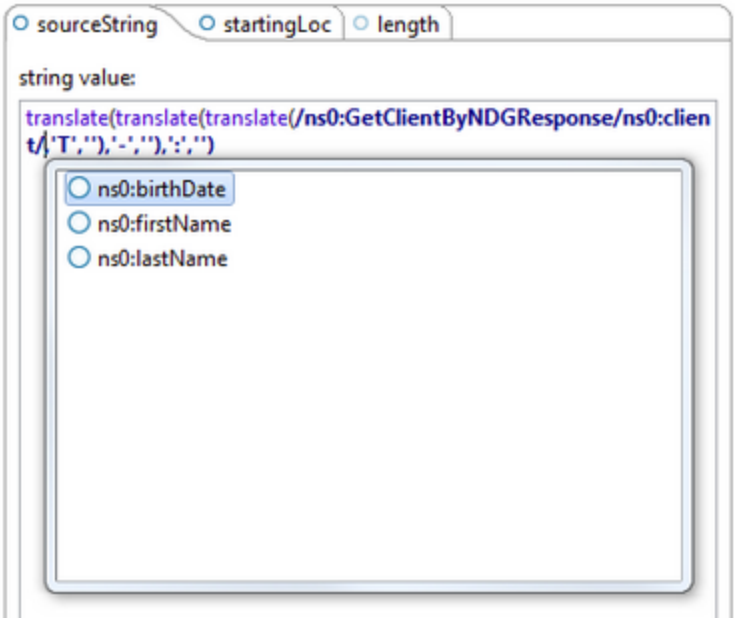
To edit a function with a complex expression:

1. Select **Edit** from context menu.
   The source schema elements are available in the left panel on the edit function dialog.
   The functions available in the mapper are listed in the right panel of the function dialog.
2. Use either drag and drop, typing or code completion (see below) to insert the element and the functions into the expression configuration area.

Tip You can use any valid XSLT function, not only the functions from the palette. Code completion is an easy way to use functions that are not in the palette.

Code completion and correction

To help you edit the parameter, click **CTRL** + **Space** to access code completion. Below is an example of code completion, offering several suggestions:



Code correction and validation

If the syntax is incorrect (for example: the expression is a string for an integer type parameter...), then the editor behaves as follows:

- when you hover over the parameter tab, a detailed error message appears as a tooltip
- when you hover over the function, the message of the first parameter in error appears as a tooltip
- you can still save the expression, to correct it later

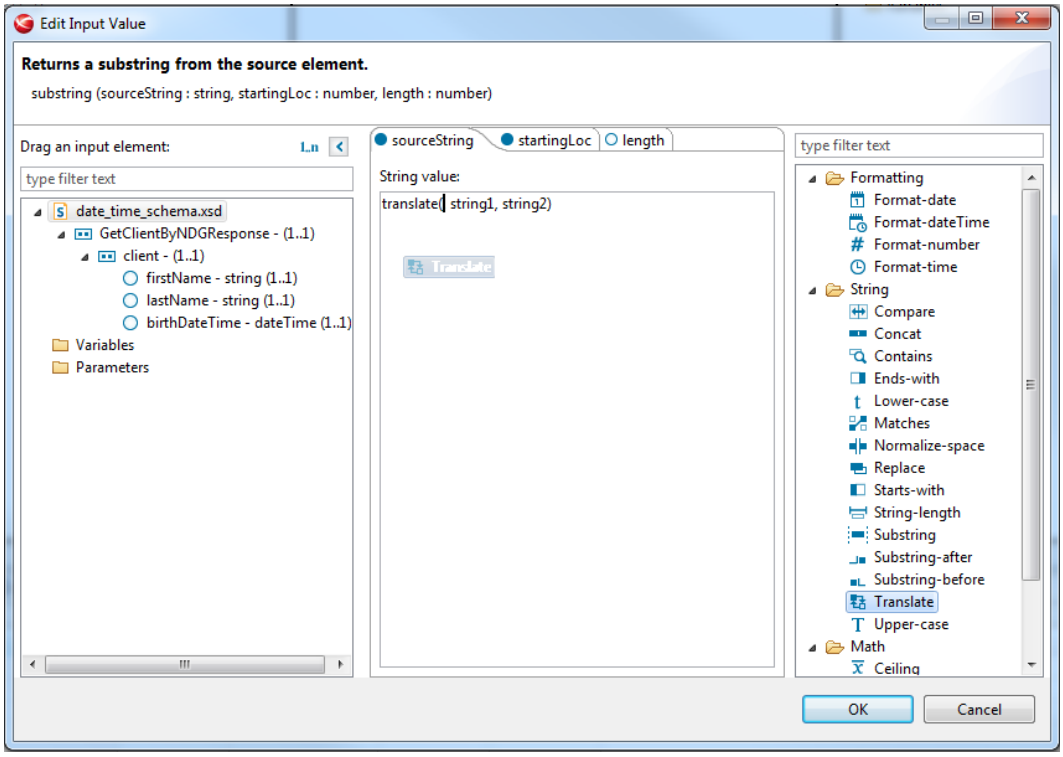
Here is a list of the some of the evaluations and verifications performed by the complex expression editor:

- it evaluates the type of a complex expression containing functions, paths, variables or parameters and operations between them
- it evaluates whether an expression is syntactically correct (whether it has all the corresponding parenthesis, for example)
- it detects when the expression uses a function which is not supported (such as typos in the function name)

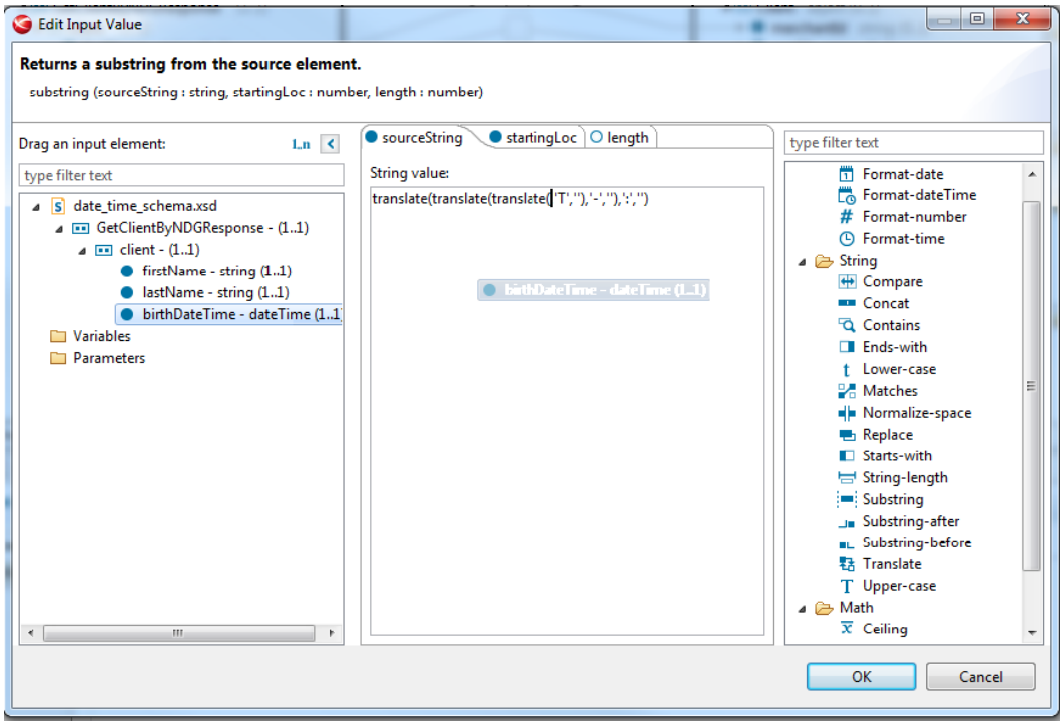
- in a function, it detects when the number of parameters is inconsistent with the function signature
- in a function, it detects when the parameter types are inconsistent with the function signature

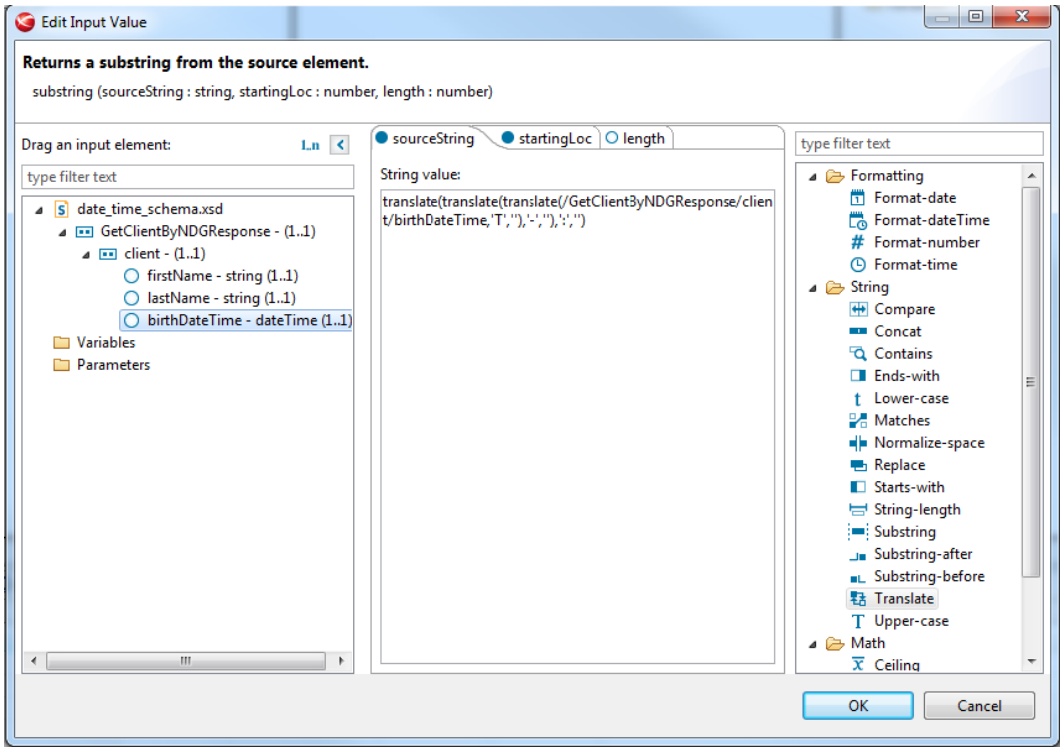
Complex expression example

For example, if you want to transform the string `1937-06-01T10:45:03' into '19370601` you could use: `substring(translate(translate(translate($value,'T',''),'-','),':',''),1,8).`



In this example, the string is converted in the value argument by replacing all characters in `string1` with the characters in `string2`. If `string2` was empty, the characters defined with `string1` are deleted in the result signature: `translate(value : string, string1 : string, string2 : string) : string`


Edit Input Value
Returns a substring from the source element.
substring (sourceString : string, startingLoc : number, length : number)
Drag an input element:
type filter text
date_time_schema.xsd
GetClientByNDGResponse - (1..1)
client - (1..1)
firstName - string (1..1)
lastName - string (1..1)
birthDateTime - dateTime (1..1)
Variables
Parameters
sourceString
startingLoc
length
String value:
translate(translate(translate('T',''),'-',''),':','')
type filter text
Format-date
Format-dateTime
Format-number
Format-time
String
Compare
Concat
Contains
Ends-with
Lower-case
Matches
Normalize-space
Replace
Starts-with
String-length
Substring
Substring-after
Substring-before
Translate
Upper-case
Math
Ceiling
OK
Cancel


Edit Input Value
Returns a substring from the source element.
substring (sourceString : string, startingLoc : number, length : number)
Drag an input element:
type filter text
date_time_schema.xsd
GetClientByNDGResponse - (1..1)
client - (1..1)
firstName - string (1..1)
lastName - string (1..1)
birthDateTime - dateTime (1..1)
Variables
Parameters
sourceString
startingLoc
length
String value:
translate(translate(translate(/GetClient3yNDGResponse/client/birthDateTime,'T',''),'-',''),':','')
type filter text
Formatting
Format-date
Format-dateTime
Format-number
Format-time
String
Compare
Concat
Contains
Ends-with
Lower-case
Matches
Normalize-space
Replace
Starts-with
String-length
Substring
Substring-after
Substring-before
Translate
Upper-case
Math
Ceiling
OK
Cancel

Glossary

Autostart

When you launch the Administration, the components that you configured in autostart will start automatically. This configuration is only done through the Admin UI, where you can add/remove components and set the priority.

Axway Administration

Axway Administration provides transversal administration features for a subset of Axway products. It comprises the Administration Agents, Manager and UI.

DMZ

Refers to part of the company network that can be accessed from outside the private network. It is used to expose entry points of the service provided by the company, without allowing direct access to the internal network. A DMZ is usually less secure than the private network therefore storing confidential information in this zone is not recommended.

Environment

An Environment is a set of Axway installations. Environment typically represents a production, pre-production or development environment.

Hostname

Corresponds to the object assigned to a physical server.

Infrastructure instance

An infrastructure instance represents a group of products installed at the same home directory. For products to share the same home directory, they must be installed on the same server. You can have several of these products running at the same time and several instances running simultaneously on an infrastructure.

Logical name

This word is used to define a specific server installation. Some Axway products need to have a key generated using a logical name, in addition to the host name. You can choose to use the machine hostname as the logical name but it is not mandatory. Shared secret

Shared secret

A password or passphrase used for authentication, exchanged only between the parties involved, in a secure communication. In this guide, you define the shared secret password

during PassPort installation. Afterwards you use this password to authenticate the connection between PassPort and another Axway product.

SSL

Secure Sockets Layer (SSL), which is the predecessor of Transport Layer Security (TLS), is an encryption protocol that ensures communication security over the Internet. See TLS for more information.

SSO

SSO (Single Sign-On) is functionality provided by PassPort that enables users to log in once for multiple Axway products (PassPort, Sentinel, Transfer CFT, InterPlay, and so on) providing an integrated solution. When this feature is activated, you need only enter your credentials once in the landing page to be able to connect to an Axway product UI that supports SSO.

TLS

Transport Layer Security (TLS) is an encryption protocol that ensures communication security over the Internet. TLS encrypts the network connection above the transport layer. TLS uses asymmetric cryptography for key exchange, symmetric encryption for privacy and message authentication codes for message integrity. Secure Sockets Layer (SSL) is the predecessor of TLS.