webMethods EntireX

EntireX XML Mapping Editor

Innovation Release

Version 9.9

October 2015

WEBMETHODS

This document applies to webMethods EntireX Version 9.9 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: EXX-EEXXXMLMAPPINGEDITOR-99-20171128

Table of Contents

1 Introduction to the XML Mapping Editor

The EntireX XML Mapping Editor allows you to map XML document structures to IDL libraries, programs and parameters. The mappings can be defined for the request and response to the server application, or from the server to the client. The input for the XML Mapping Editor can be a Software AG IDL file and/or an IDL-XML mapping file (perhaps produced by a previous XML Mapping Editor session or by importing a WSDL file, XML Document or XML Schema). The output is an IDL-XML mapping file, other XML structure definitions (such as sample XML files), and perhaps a created or changed IDL file.

With the XML Mapping Editor you can:

- Generate XML structures and IDL-XML mapping files for code generation and runtime components.
- Modify the XML structures.
- Modify the mapping links of IDL and XML structures.
- Validate existing or user-modified XML structures and IDL-XML mapping links.

2 Using the XML Mapping Editor

See also *Using the XML Tester*.

Starting the XML Mapping Editor

To start the XML Mapping Editor

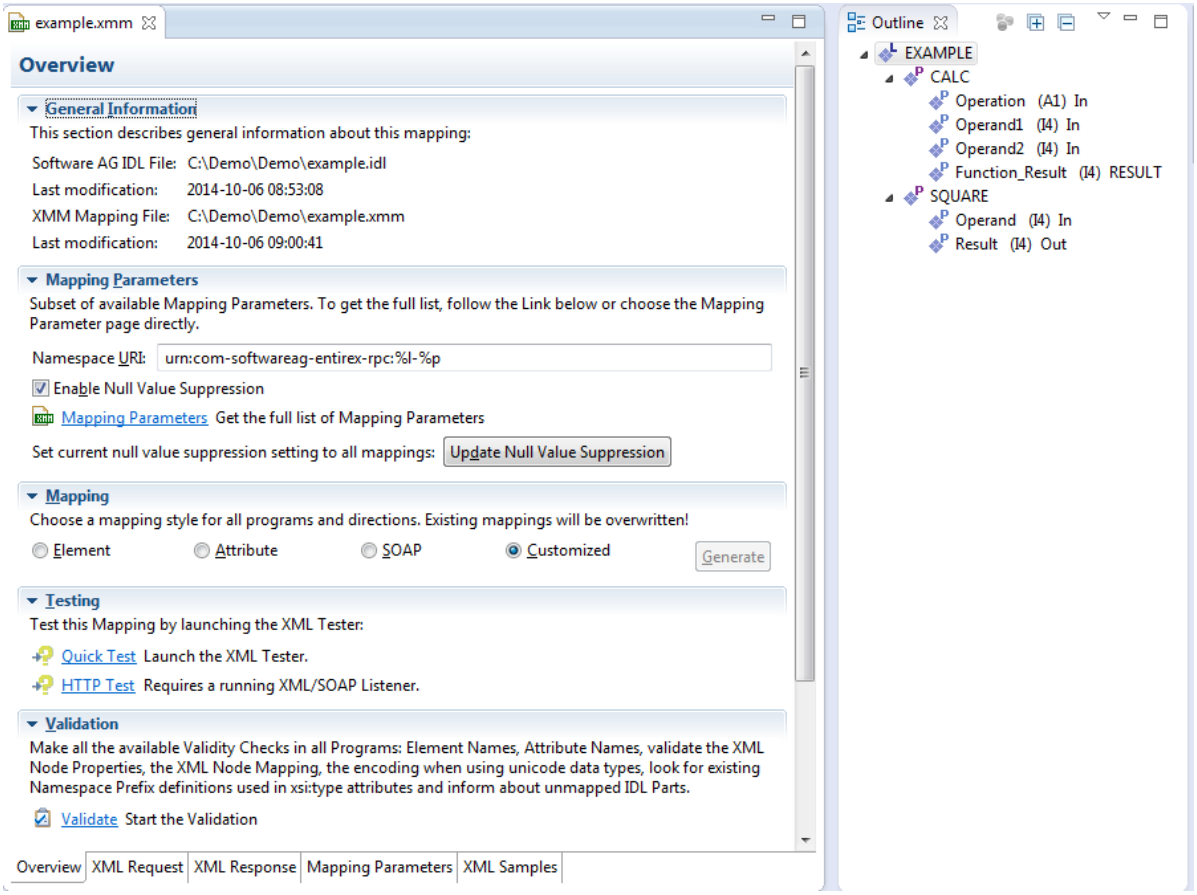
In the *EntireX Workbench*

- Double click on an XMM file.

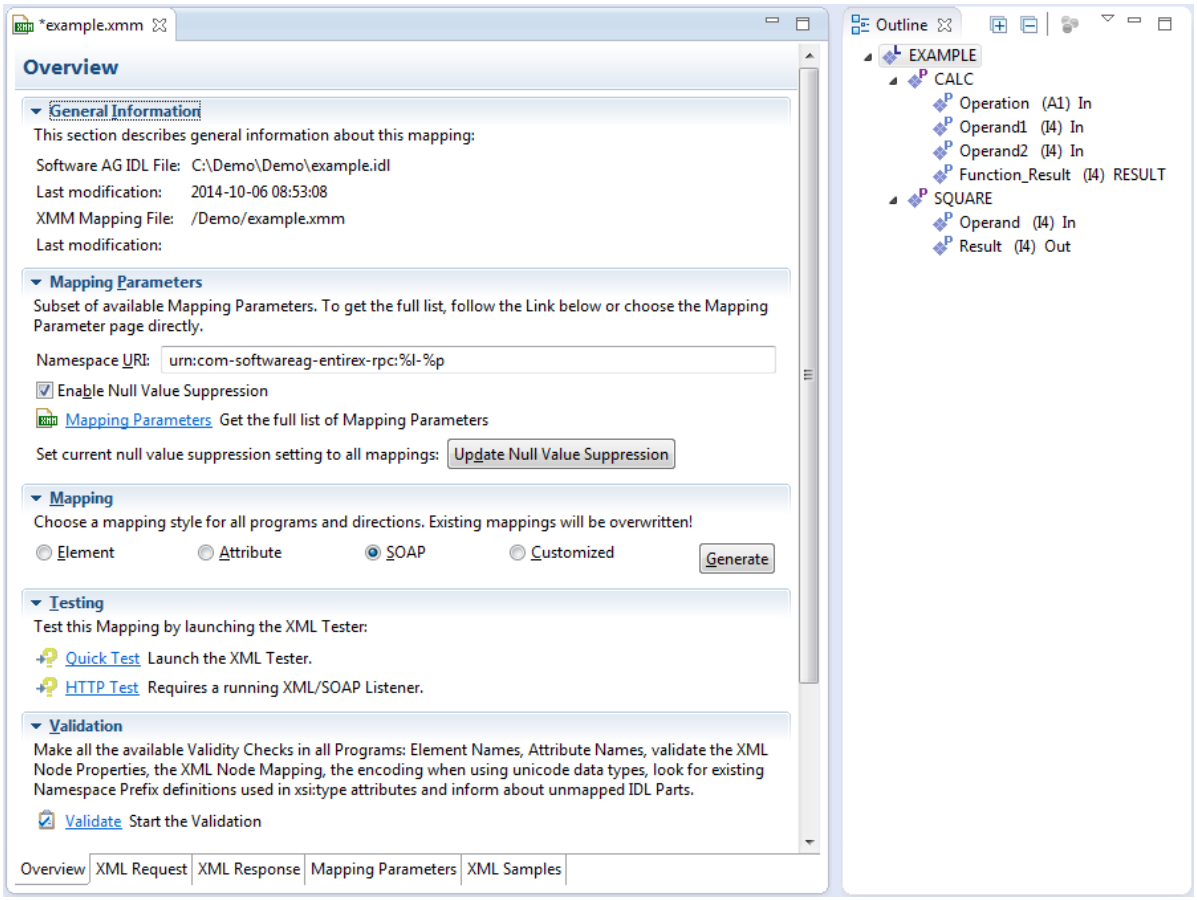
  Or:

  Select an IDL or XMM file and choose **Open With** > **EntireX XML Mapping Editor**.

  If the XMM file for the selected IDL file exists, the XML Mapping Editor loads this. The **Generate** button is disabled because a mapping exists.



  If no XMM file exists for the IDL file, the mapping style is set to "SOAP" as default value. Choose the required mapping style and press **Generate**.



To close the XML Mapping Editor

- From the **File** menu, choose **Close**.

  Or:

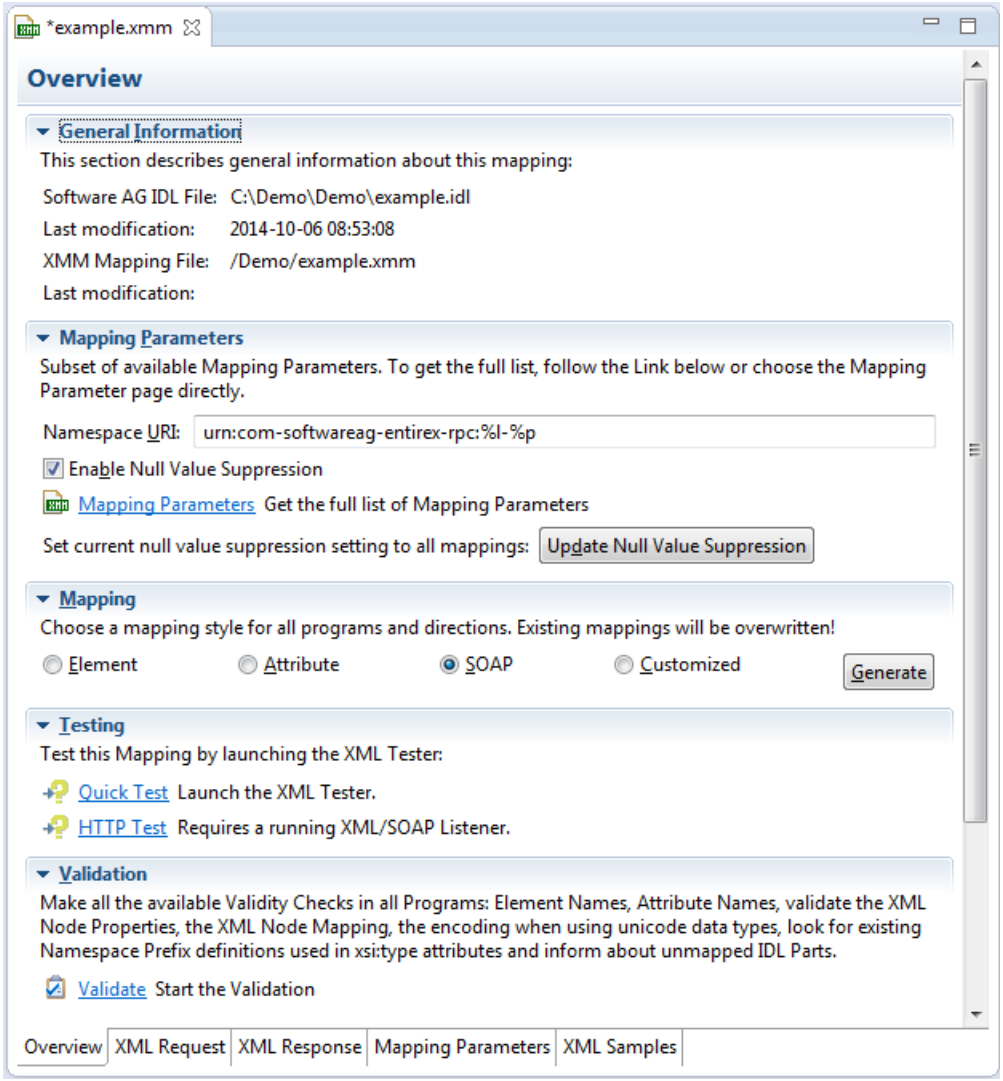
  Click the **Close** icon in the editor title bar.

If the current XML mapping has not yet been saved, a dialog box appears from which the close operation can be stopped or the IDL-XML mapping can be saved. An XML mapping that is not clean is indicated with a leading asterisk in the file name, displayed in the editor title.

The XML Mapping Editor Pages

Overview Page

The **Overview Page** is the central page and provides sections for file information, the most important mapping parameters, mapping generation, testing and validation.



These sections are:

- **General Information**
  This section describes the names and locations of the selected XML mapping (XMM) file and the related IDL file.
- **Mapping Parameters**
  Provides the **Namespace** and **Null Value Suppression** controls and contains a link to the **Mapping Parameters Page**.

  Note: Press **Update Null Value Suppression** to apply the mapping parameters (see *Mapping Parameters Page*) to a generated mapping.

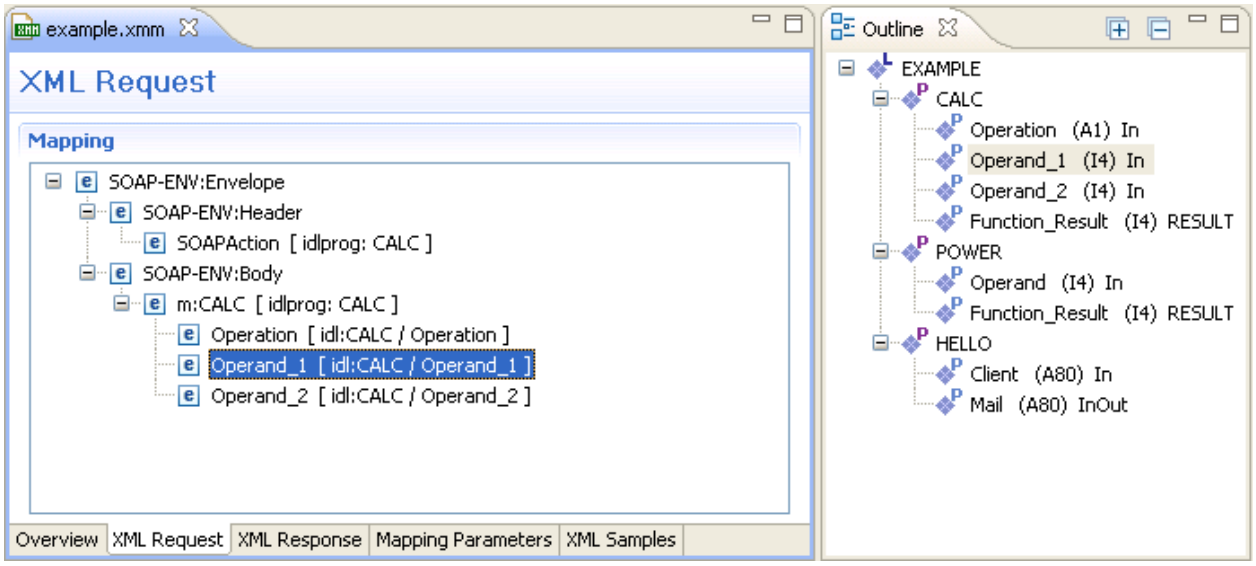
- **Mapping**
  Starts the mapping generation for all programs.

  Note: Existing mappings will be overwritten!

- **Testing**
  Allows you to launch the XML Tester in Quick mode or HTTP mode.
- **Validation**
  Performs validity checks in all programs and reports the result in a new dialog. Detected problems will be displaced in the **Problems** view.

XML Request Page

This page contains the **Mapping** tree of the XML Request, which is linked together by mapping paths with the IDL tree in the **Outline** view.



The XML Request shows the XML tree loaded or created so far for the selected IDL program. The XML tree can be modified in various ways. See *IDL to XML Mapping with the XML Mapping Editor*.

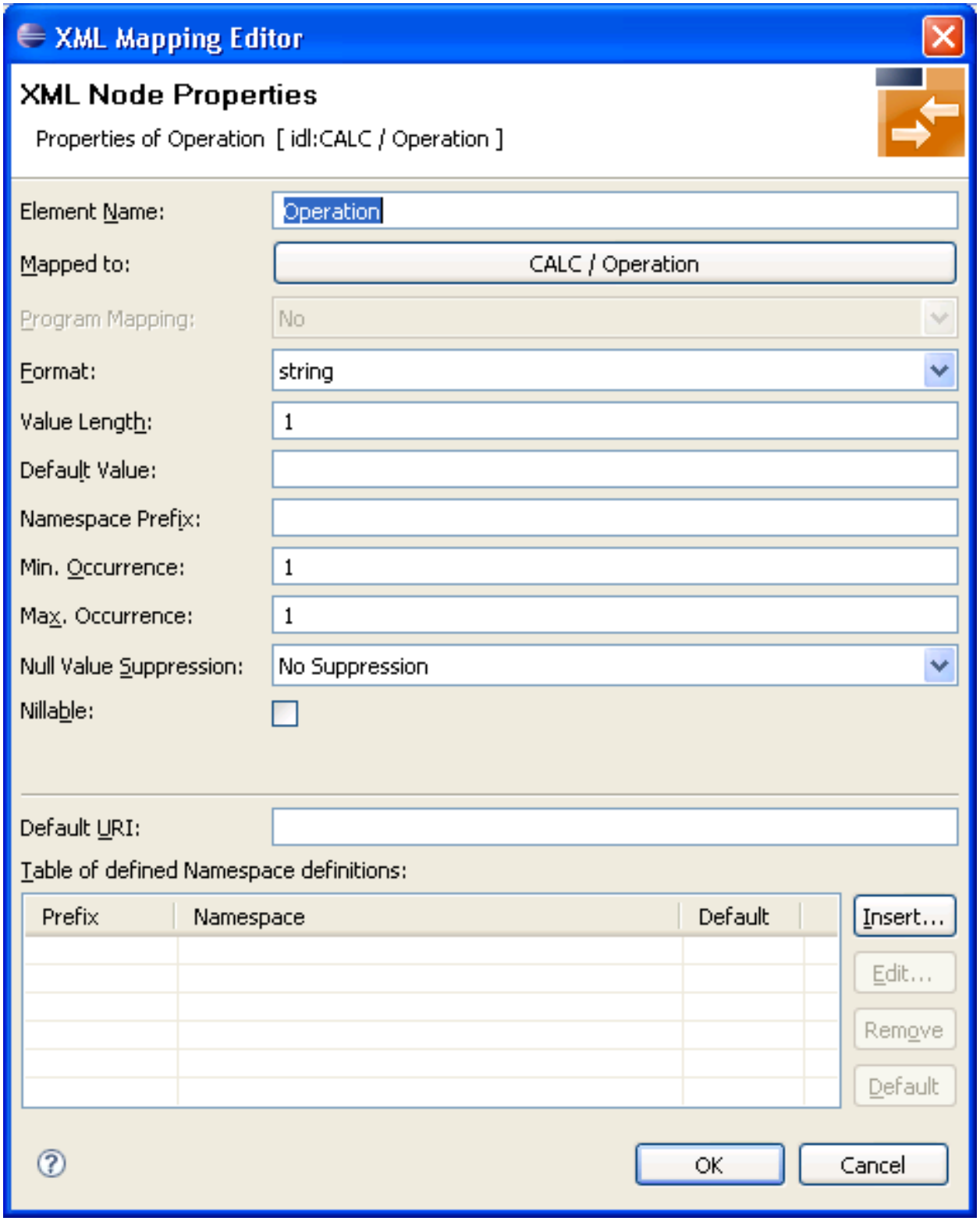
The **Outline** view shows the IDL tree of the currently selected IDL file. The IDL tree display has the same functionality as the IDL Editor's tree display; however, due to the fact that every IDL program must be separately mapped, only one of the IDL program tree nodes can be expanded.

Use the right mouse button to display the context menu. Double-click on a tree node to display detailed information on that node. The following keyboard options are also available in the XML request mapping area:

Key	Description
A or SHIFT+INS	Create new parameter node after currently selected one.
D or DELETE	Delete the selected IDL tree node.
F	Find IDL node with given name pattern.
I or ALT+INS	Create a new group parameter node.
L	Create new IDL library node.
M	Find mapped XML node for selected IDL tree node.
N or INS	Create new parameter node.
O or CTRL+INS	Create new RPC program node.
P or ALT+ENTER	Open property dialog on selected IDL tree node.
R	Rename the selected IDL tree node.
F3	Follow-up of M, find next mapped XML node for selected IDL tree node.

XML Node Properties Dialog

The XML Node Properties dialog contains the XML node details, the bottom panel contains namespace settings.


XML Mapping Editor
XML Node Properties
Properties of Operation [idl:CALC / Operation]
Element Name:
Operation
Mapped to:
CALC / Operation
Program Mapping:
No
Format:
string
Value Length:
1
Default Value:
Namespace Prefix:
Min. Occurrence:
1
Max. Occurrence:
1
Null Value Suppression:
No Suppression
Nillable:
Default URI:
Table of defined Namespace definitions:
Prefix
Namespace
Default
Insert...
Edit...
Remove
Default
OK
Cancel

To open the XML Node Properties Dialog

- Double-click on an XML node in the tree display.

  Or:

  Choose the Properties menu item in the context menus.

  Or:

  Select an XML tree node and press Enter.

The XML Node Properties can be modified. The node details panel consists of two subpanels.

The upper panel contains the following:

Item	Description
Element or attribute name	Name of the element or attribute.
Mapped to	The IDL mapping link in the full path name format. For error (fault) trees, context information.
Program Mapping	Combo Box No: No Program Mapping Element value: The value of this node identifies the program Element name: The name of this node identifies the program
Format	**Format** / **IDL** string: A, AV, K, KV -: G I1 (deprecated): I1 I2 (deprecated): I2 I4 (deprecated): I4 I8 (deprecated): I4 integer: I1, I2, I4 date: D time (deprecated): T dateTime: T float: F4, F8 Boolean: L binary: B, BV number: N, P, NU, PU

Item	Description
	Format | **IDL** For Tamino/special nodes: Describes the data type of the Tamino node (for example, integer, string).
Value Length	The length parameter contains the length or precision of the data type, i.e. ■ length of string for alphanumeric ■ length of field in bytes for binary ■ length of string representation for floating point ■ length of string representation for integers ■ length of string for Kanji ■ length of bit vector for logical (Boolean) Length V is legal for string types. It denotes variable-length strings, e.g. alphanumeric or Kanji. For Tamino/special nodes: Length of the storage reserved for the data must match format.
Default Value	Must match the data type of the node, e.g. do not use alphanumeric defaults for integer nodes. Defaults that cannot be interpreted by the runtime component are ignored. See also *Assigning Default Values*.
Namespace Prefix	Contains the namespace prefix to the tag name.
Min. Occurrence	The minimum and maximum occurrence properties are numeric values greater than or equal to zero. They describe the number of allowed occurrences of the node in the incoming XML, and the number of occurrences to be generated in the outgoing XML. The minimum occurrences value must be less than or equal to the maximum occurrences value. Incorrect settings will be found in the validity check Validate XML node properties. If Min. Occurrence is greater than 0 for incoming documents, the corresponding attribute or element is required, i.e. must be set.
Max. Occurrence	See Min. Occurrence.
Null Value Suppression (NVS)	Specify suppression of null values (NVS), that is, whether empty elements or attributes may be omitted in the outgoing (generated) XML documents. Mainly determined by Min.Occurrences and Max. Occurrences as follows: ■ For min = 0, max = 0, there is no NVS. ■ For min = 0, max = 1, the element / attribute may be omitted if empty. ■ For min = 0, max > 1, you can omit all empty elements or just omit empty elements at the end of the sequence (trimming). ■ For min = 1, max = 1, there is no NVS. ■ For min = 1, max > 1, you can omit all empty elements or just omit empty elements at the end of the sequence (trimming). ■ For 0 < min < max, you can omit all empty elements or just omit empty elements at the end of the sequence (trimming).

<table>
<tr><th>Item</th><th>Description</th></tr>
<tr><td></td><td>It is also possible to suppress an attribute (independent of its value) if the associated element has a null value and null value suppression is enabled; to do this choose Depends on element in property Null Value Suppression.

For the data types, the following null values are defined:

<table>
<tr><td>Format</td><td>NVS Default</td></tr>
<tr><td>string</td><td>" " (empty string)</td></tr>
<tr><td>-</td><td>" " (empty string)</td></tr>
<tr><td>I1 (deprecated)</td><td>0</td></tr>
<tr><td>I2 (deprecated)</td><td>0</td></tr>
<tr><td>I4 (deprecated)</td><td>0</td></tr>
<tr><td>integer</td><td>0</td></tr>
<tr><td>date</td><td></td></tr>
<tr><td>time (deprecated)</td><td></td></tr>
<tr><td>dateTime</td><td></td></tr>
<tr><td>float</td><td>0.0</td></tr>
<tr><td>Boolean</td><td>false</td></tr>
<tr><td>binary</td><td>Only BV with length =0; Only binaries corresponding to IDL parameter with type BV have an NVS default [element/attribute with data length=0].</td></tr>
<tr><td>number</td><td>0.0</td></tr>
</table></td></tr>
<tr><td>Null Value</td><td>If NVS is switched on, this value is compared with the values of the XML document and it is decided whether the values need to be transmitted. The defaults are in the table for NullValue Suppression (NVS).</td></tr>
<tr><td>Nillable</td><td>If NVS is set to "No Suppression", the Nillable option is available in the dialog. If the option is selected, an empty element is represented as an empty-element tag with attribute <code>xsi:nil="true"</code>. This means that if an nillable group element only has elements without value and without attributes, only the enclosing group tag with attribute <code>xsi:nil="true"</code> is displayed.</td></tr>
<tr><td>Time Pattern</td><td>Optional for date, time (deprecated) and dateTime.

The default patterns conform to the XSD Schema 2001 specification:

date: <code>yyyy-MM-dd</code>, for example 2003-04-15
dateTime: <code>yyyy-MM-dd'T'HH:mm:ss</code>, for example 2003-04-15T18:48:23

If you want to define your own time pattern (for example with "Day in week" or "General Time Zone"), see the Java documentation for <code>SimpleDateFormat</code>.</td></tr>
</table>

Correlation of Occurrence Setting and Null Value Suppression

The value for null value suppression has a higher priority than the occurrence setting of affected elements or attributes. The following rules apply:

1. **NVS="Suppress Element"/"Suppress Attribute"**
   Setting NVS to Suppress Element" or "Suppress Attribute" results in a value of minimum occurrence of zero, which makes it optional.
2. **NVS="No Suppression"**
   Setting NVS to "No Suppression" means that the element/attribute is always displayed so the value of #minOccur must be 1 - otherwise the element/attribute would be optional and could be suppressed.
3. **NVS for arrays**
   For arrays, the dependency of minimum occurrence and Null Value Suppression is more complex:
   a. **NVS set to "No Suppression"**
      For an array with fixed size the setting is `#minOccur = #maxOccur = #array size`.

      An array with variable size will contain all non-suppressable elements, but at least `#minOccur` elements. Empty elements will be generated to guarantee the number of `#minOccur`.
   b. **NVS set to value other than "No Suppression"**
      The number of array elements is at least #minOccur (empty elements will be generate and appended if number of elements is lower than #minOccur).

The lower panel is only available for element nodes and contains namespace-related properties:

Item	Description
Namespace URI	A default Namespace (xmlns= .....) for this element node.
Namespace definitions (table)	A table of namespace prefix / URI assignments for this element node. The listed namespaces are defined for this element and all subelements, as described in the XML namespaces specification document. Two buttons, Add and Delete, allow you to add or delete namespace definitions, respectively.

Using the Context Menu

The context menu of the XML tree enables you to modify the XML structure. It contains menu items to insert new nodes, remove or rename nodes, remove an IDL mapping, change the XML part type to element or attribute, move the selected node to the top or bottom of the current subtree, move the selected node up or down within the subtree, or show the XML Node Properties dialog.

To open the context menu

- Select the XML tree node and click the right mouse button.

Menu Item	Shortkey	Description
New child node	Ctrl-N	Insert a child node under the selected node.
Insert before	Ctrl-B	Insert a new node before the selected node.
Insert after	Insert	Insert a new node after the selected node.
Set to Attribute	Ctrl-S	Turn an attribute into an element. Also works for multiple selected nodes.
Set to Element	Ctrl-E	Turn an element into an attribute. Also works for multiple selected nodes.
Bring to top	Ctrl-T	Move the selected node to the top of the current subtree.
Bring to bottom	Ctrl-O	Move the selected node to the bottom of the current subtree.
Move up	Ctrl-U	Move the selected node up within the subtree.
Move down	Ctrl-D	Move the selected node down within the subtree.
Cut	Ctrl-X	Cut the selected node.
Copy	Ctrl-C	Copy the selected node.
Paste	Ctrl-V,Ctrl-P	Paste a node from the clipboard (after copy or cut).
Unmap	Ctrl-M	Unmap the selected IDL and XML nodes, i.e, remove link. Also works for multiple selected nodes.
Delete	Delete/Backspace	Remove the selected node. Also works for multiple selected nodes.
Rename	Ctrl-R	Rename the selected node.
Properties	Alt-Enter	Open the XML Details Panel.

Using Drag-and-drop

You can use drag-and-drop operations within the XML tree instead of move and cut-and-paste. Select a node (even with subnodes) and drag it to another place. The dragged subtree is then inserted after, before or under the corresponding drop node (to insert it under the drop node, use the Ctrl key as a toggle).

Some drag-and-drop operations are illegal, e.g. dragging a node into one of its descendant nodes (children). This would result in a cyclic reference and is thus forbidden.

Dropping a dragged node into an attribute node will convert the attribute node to an element node. This is because attributes may not have descendant nodes.

Caution: If you drag a subtree into or out of an array, the IDL mapping links of all nodes of that subtree will be deleted. This is because the cardinality of the node occurrence has changed, and the resulting IDL mapping is very likely to be incorrect.

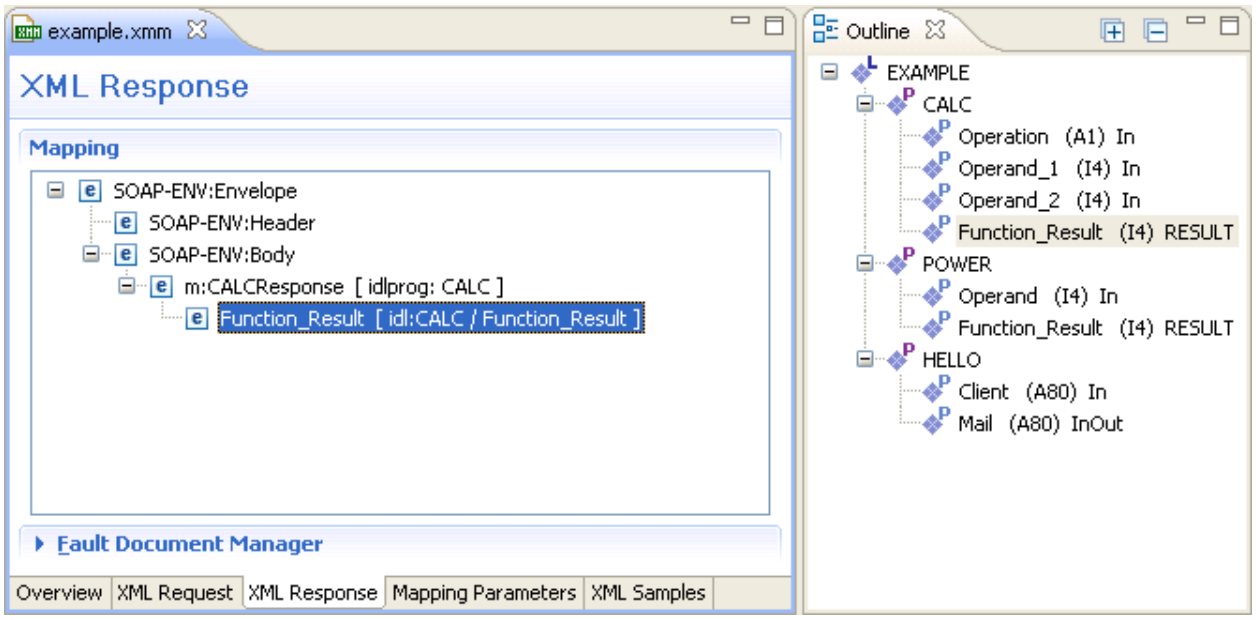
Drag-and-drop from an IDL Tree Node to the XML Tree

If you drag an IDL tree node onto an XML tree node, the IDL-XML mapping link for that XML node is changed. The new XML-IDL mapping of this XML tree node is to the dragged IDL node. The IDL tree is not changed.

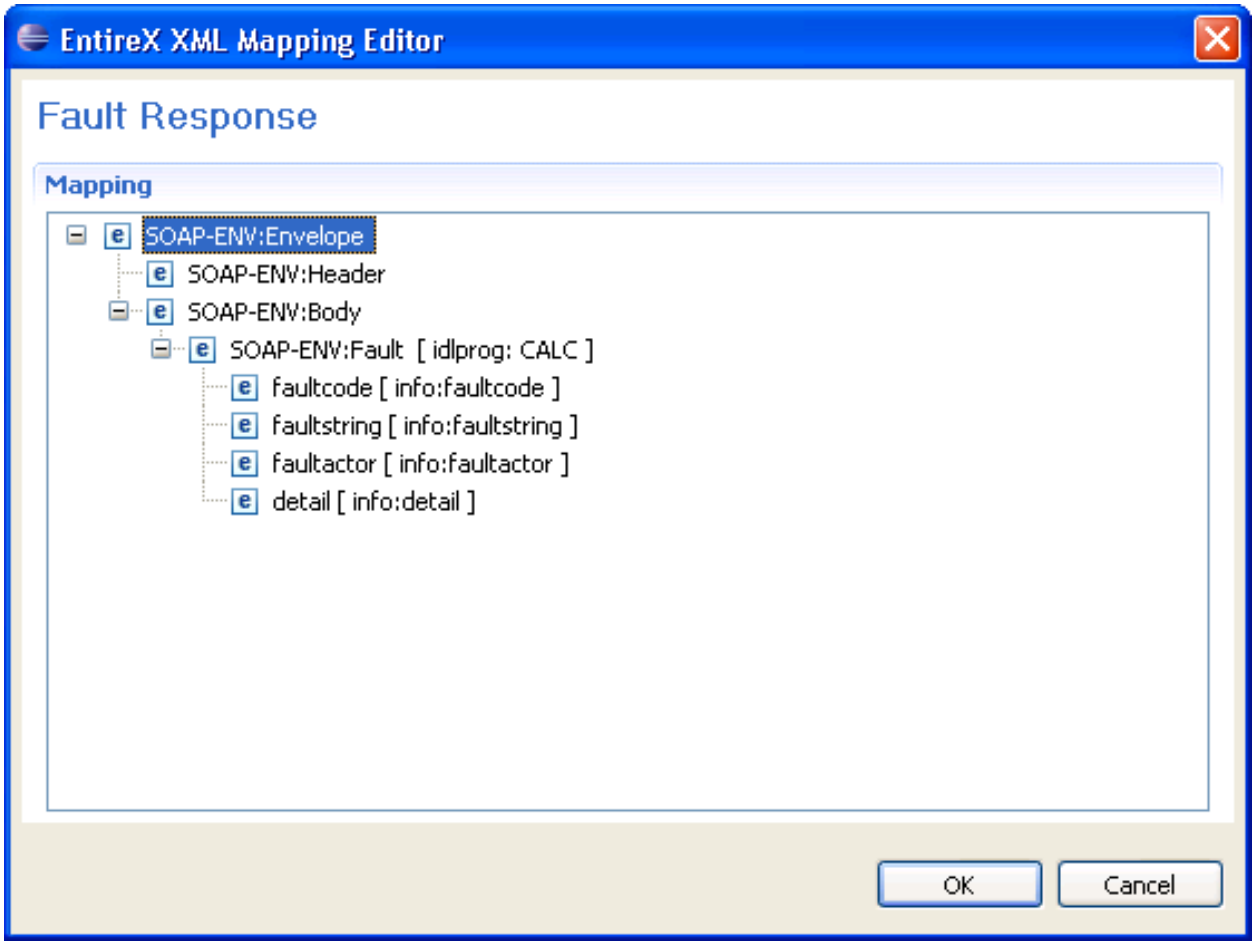
This is a quick way to change IDL mappings for XML parts. Note that there is no immediate check for duplicate assignment of an IDL node; however, the validity checks will detect it.

XML Response Page

This page contains the Mapping Tree of the XML Response, which is linked together by mapping paths with the IDL tree in the Outline View. The page extends the XML Request page with an additional section, Fault Document Manager, at the bottom.



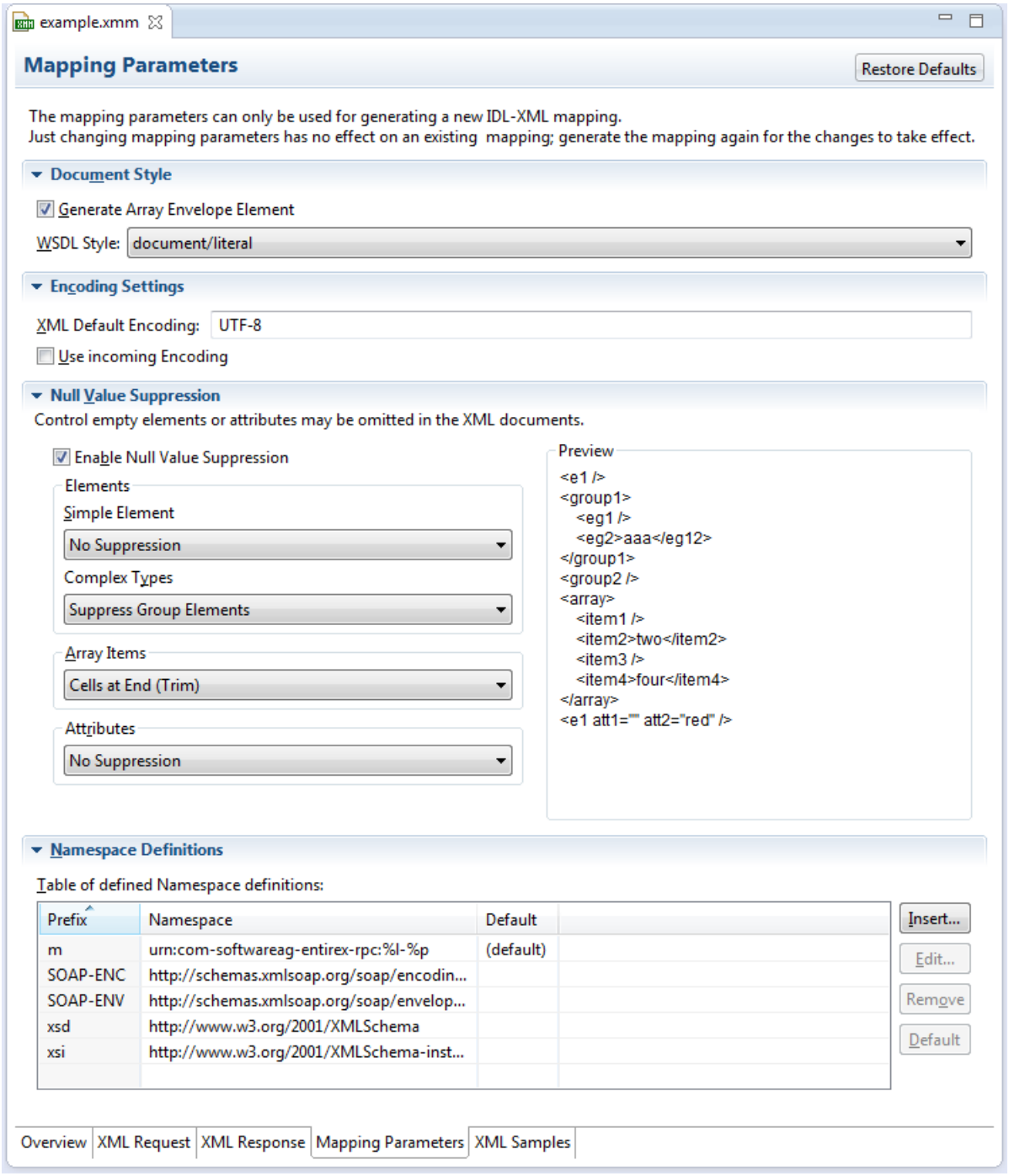
The Fault Document Manager contains the entry for the XML Fault Response Document. By selecting the entry, the **Edit...** button will be enabled. Press **Edit...** to display the following dialog:



The same operations as for the XML Request and XML Response trees are possible. Confirm the changes with **OK**, or click **Cancel** to exit without changes.

Mapping Parameters Page

This page covers the Mapping Parameters to affect the Mapping Structure. The default values for that page will be managed in the preferences and can be loaded by using the **Restore Defaults** button in the upper right corner.

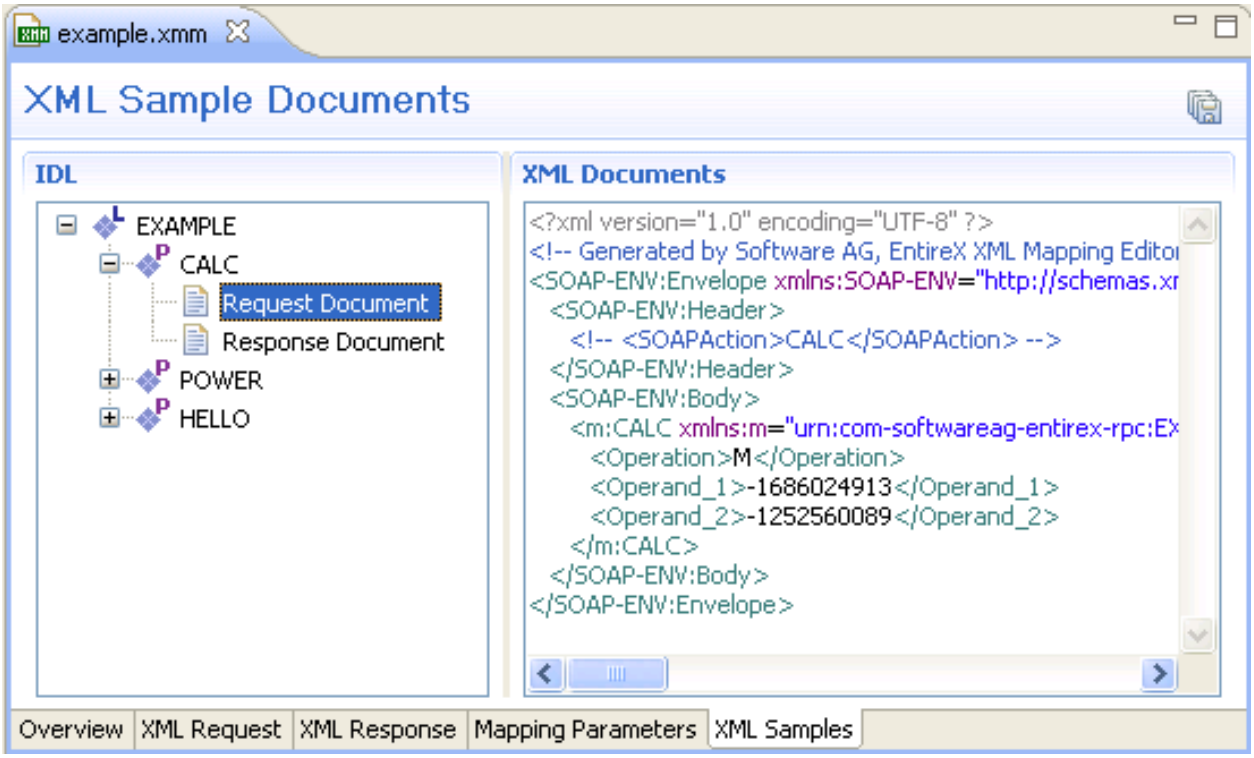

example.xmm
Mapping Parameters
Restore Defaults
The mapping parameters can only be used for generating a new IDL-XML mapping.
Just changing mapping parameters has no effect on an existing mapping; generate the mapping again for the changes to take effect.
Document Style
Generate Array Envelope Element
WSDL Style: document/literal
Encoding Settings
XML Default Encoding: UTF-8
Use incoming Encoding
Null Value Suppression
Control empty elements or attributes may be omitted in the XML documents.
Enable Null Value Suppression
Elements
Simple Element
No Suppression
Complex Types
Suppress Group Elements
Array Items
Cells at End (Trim)
Attributes
No Suppression
Preview
<e1 />
<group1>
<eg1 />
<eg2>aaa</eg12>
</group1>
<group2 />
<array>
<item1 />
<item2>two</item2>
<item3 />
<item4>four</item4>
</array>
<e1 att1="" att2="red" />
Namespace Definitions
Table of defined Namespace definitions:
Prefix
Namespace
Default
m
urn:com-softwareag-entirex-rpc:%l-%p
(default)
SOAP-ENC
http://schemas.xmlsoap.org/soap/encodin...
SOAP-ENV
http://schemas.xmlsoap.org/soap/envelop...
xsd
http://www.w3.org/2001/XMLSchema
xsi
http://www.w3.org/2001/XMLSchema-inst...
Insert...
Edit...
Remove
Default
Overview
XML Request
XML Response
Mapping Parameters
XML Samples

Parameter	Description
Generate Array Envelope Element	Determines whether for each array a surrounding additional element (envelope) is generated or not.
WSDL Style	Prepare the SOAP Mapping for selected WSDL Style. Possible values: document/literal or RPC/encoded.
XML Default Encoding	This encoding is used for the XML/SOAP document sent if the box **Use incoming XML encoding** is not checked (for XML-based clients), or if the XML/SOAP RPC Server is used.
Use Incoming Encoding	Check this box to enable the XML/SOAP Wrapper to use same encoding for the incoming document as for the outgoing document.
Enable Null Value Suppression(1)	Switch on/off the null value suppression.(1)
Simple Element(1)	Suppress Elements. Possible Values: No Suppression or Suppress Element.(1)
Simple Attribute(1)	Suppress Attributes. Possible Values: No Suppression or Suppress Attribute.(1)
Array Types(1)	Suppress Array Types. Possible Values: No Suppression, All empty cells or Cells at end (Trim).(1)
Complex Types(1)	Suppress Complex Types. Possible Values: No Suppression Suppression (no special handling of complex types - null value suppression defined for 'Simple Element' is used), or Suppress Group Elements.(1)
Namespace Definitions Table	Manage all Namespaces with prefix and URI.

(1) For more details on null value suppression, see below or *Null Value Suppression* under *Writing Advanced Applications with the XML/SOAP Wrapper*.

XML Sample Documents Page

This page allows the generation and modification of XML Sample Documents. They can be used to test the Mapping by sending the generated Request Document to the XML Tester. Another useful point is to compare the XML Sample Document structure with the real XML Document returned by your application to detect differences, for example Namespace Definitions, typos or anything else.



All generated XML Sample Documents can be stored in parallel to the selected XMM file by using the Save All toolbar button in the upper right corner of this page or the Save command in the context menu. The generated file name is built as follows:

```
[xmm name] .[library name].[program name].[direction (request | response)].xml
```

For example: `example.EXAMPLE.CALC.request.xml`

If a file already exists, a dialog will prompt you for confirmation to overwrite it.

Using the Context Menu of XML Samples

The XML Samples context menu allows the generation of XML Sample Documents.

There are two different menus:

- **Library** or **Program** is selected

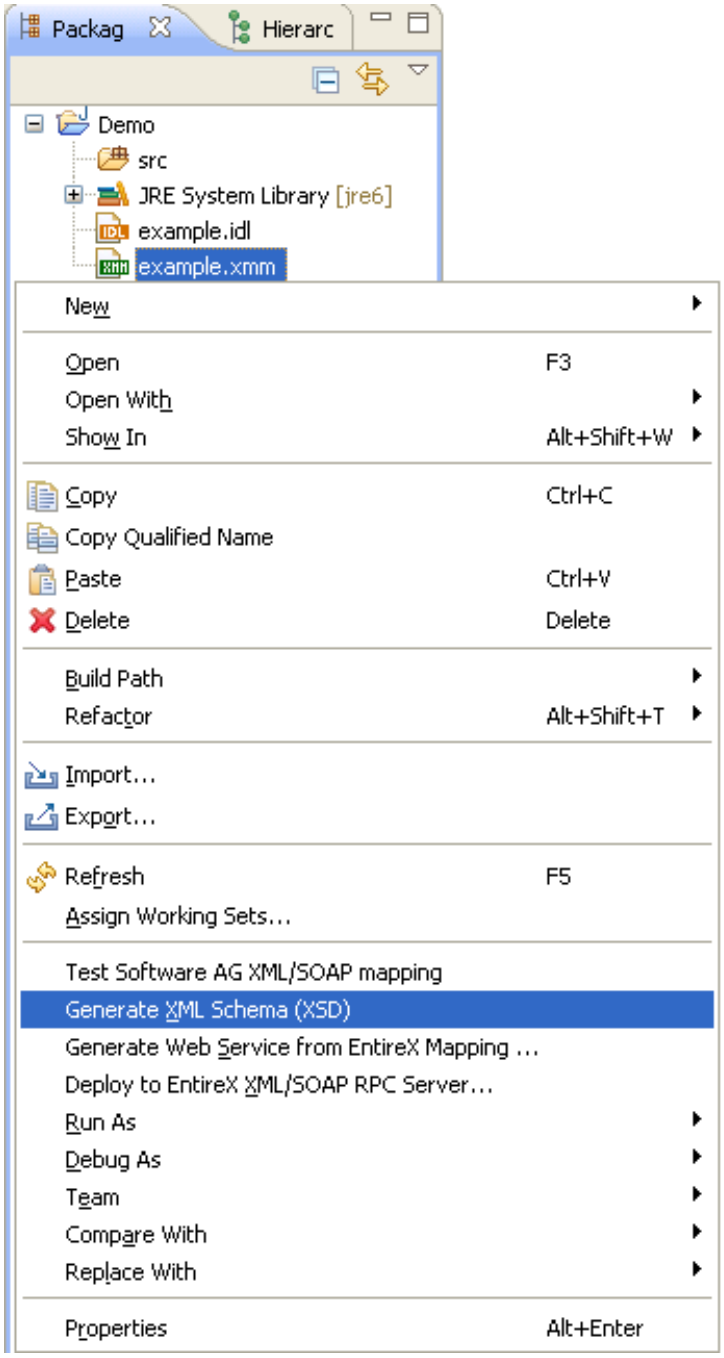
Generate All means both directions (Request and Response). When a Library is selected, the commands here will be inherited by all Programs.

- **Request** or **Response** Document is selected

The **XML Tester...** command sends the selected XML Sample Document to the XML Tester as Quick Test and enters the name of the XMM file.

XML Schema Export

The current XML Mapping can be exported as XML Schema, using the context menu of the XMM file. The XML Schema style "Russian Doll" or "Venetian Blind" can be selected in the preferences.



The generated XML Schema files will be stored in parallel to the selected XMM file, where one file represents the direction (request or response) and the defined prefixed element.

For example: a SOAP mapping creates four files from the example CALC program: two prefixes ("SOAP-ENV" and "m"), and two directions (request and response). The generated file name is built as follows.

```
[xmm name].[library name].[program name].[direction (request |
response)][(optional).additional prefix].xsd
```

Example

- example.EXAMPLE.CALC.request.xsd
- example.EXAMPLE.CALC.request.m.xsd
- example.EXAMPLE.CALC.response.xsd
- example.EXAMPLE.CALC.response.m.xsd

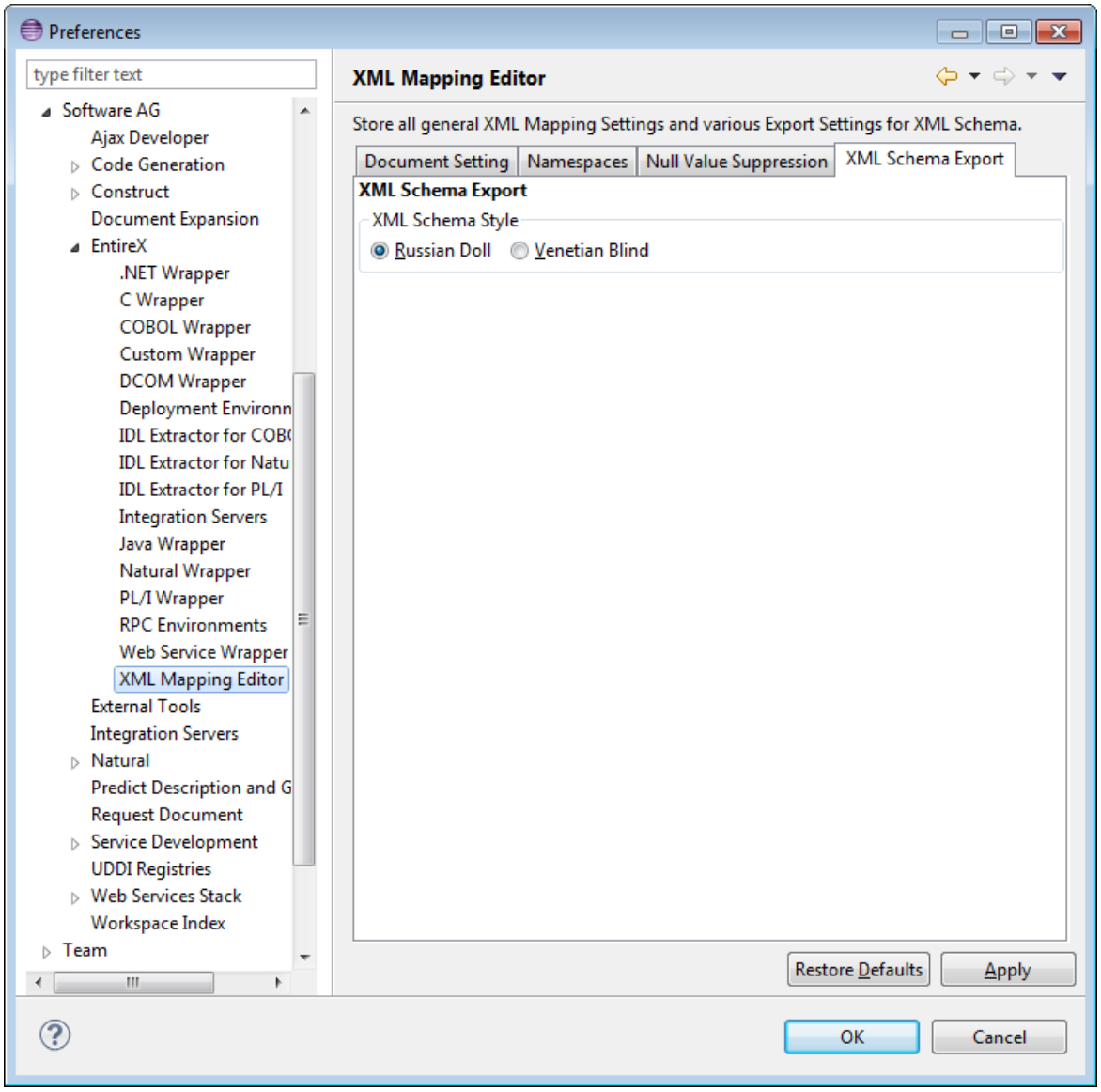
If a file already exists, a dialog will prompt you for confirmation to overwrite it.

XML Mapping Editor Settings

The preference page XML Mapping Editor manages the default values.

For XML Schema the style "Russian Doll" or "Venetian Blind" can be selected.

All other parameters are described under *Mapping Parameters*.

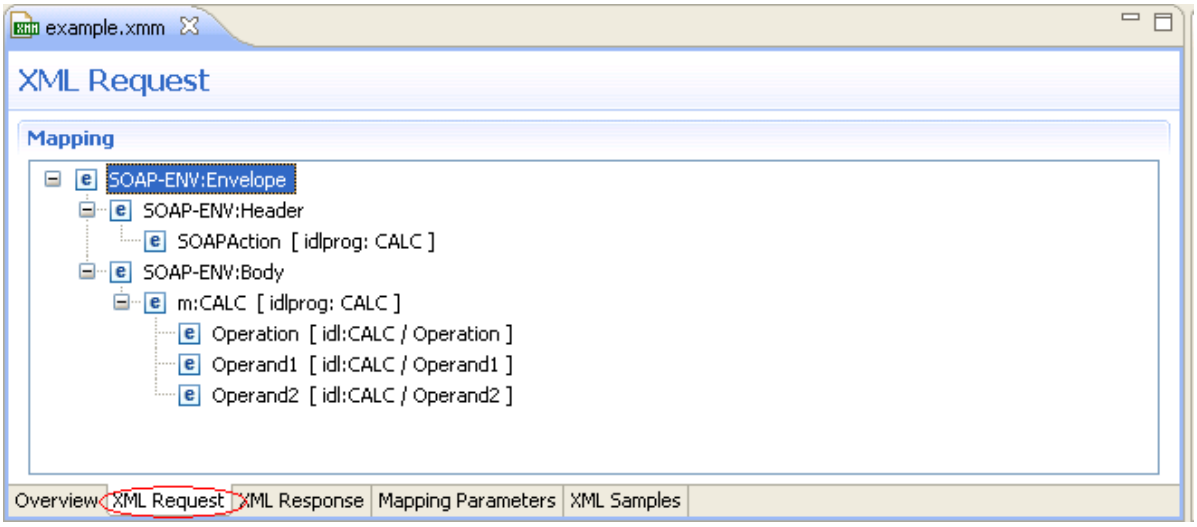


Removing Unused Namespaces

To remove one or more unused namespaces

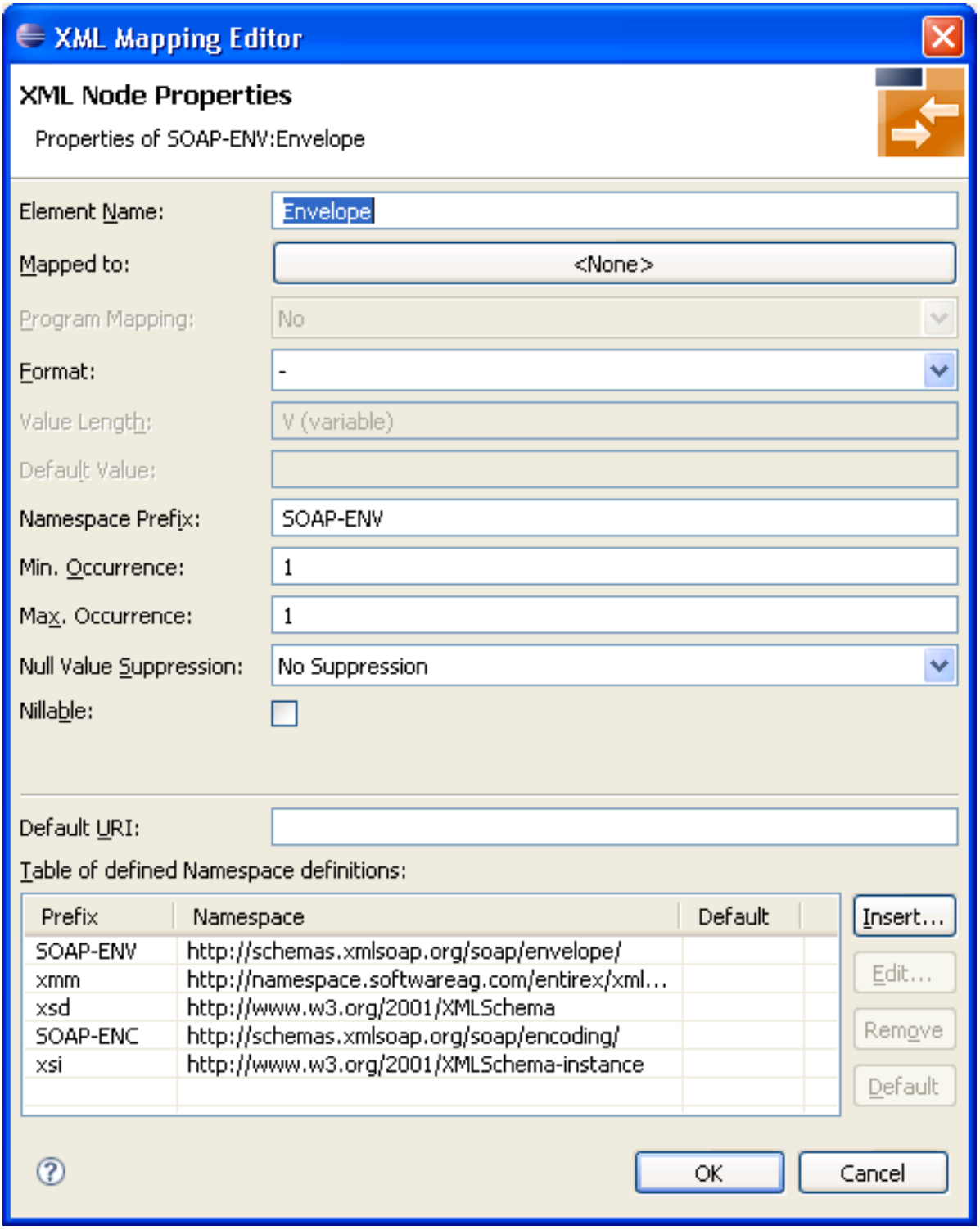
1 Open the XML Mapping Editor for the XMM file.

2 Select tab **XML Request**.



3 Select the element that defines namespace(s) to be removed.

4 From the context menu, choose **Properties**.



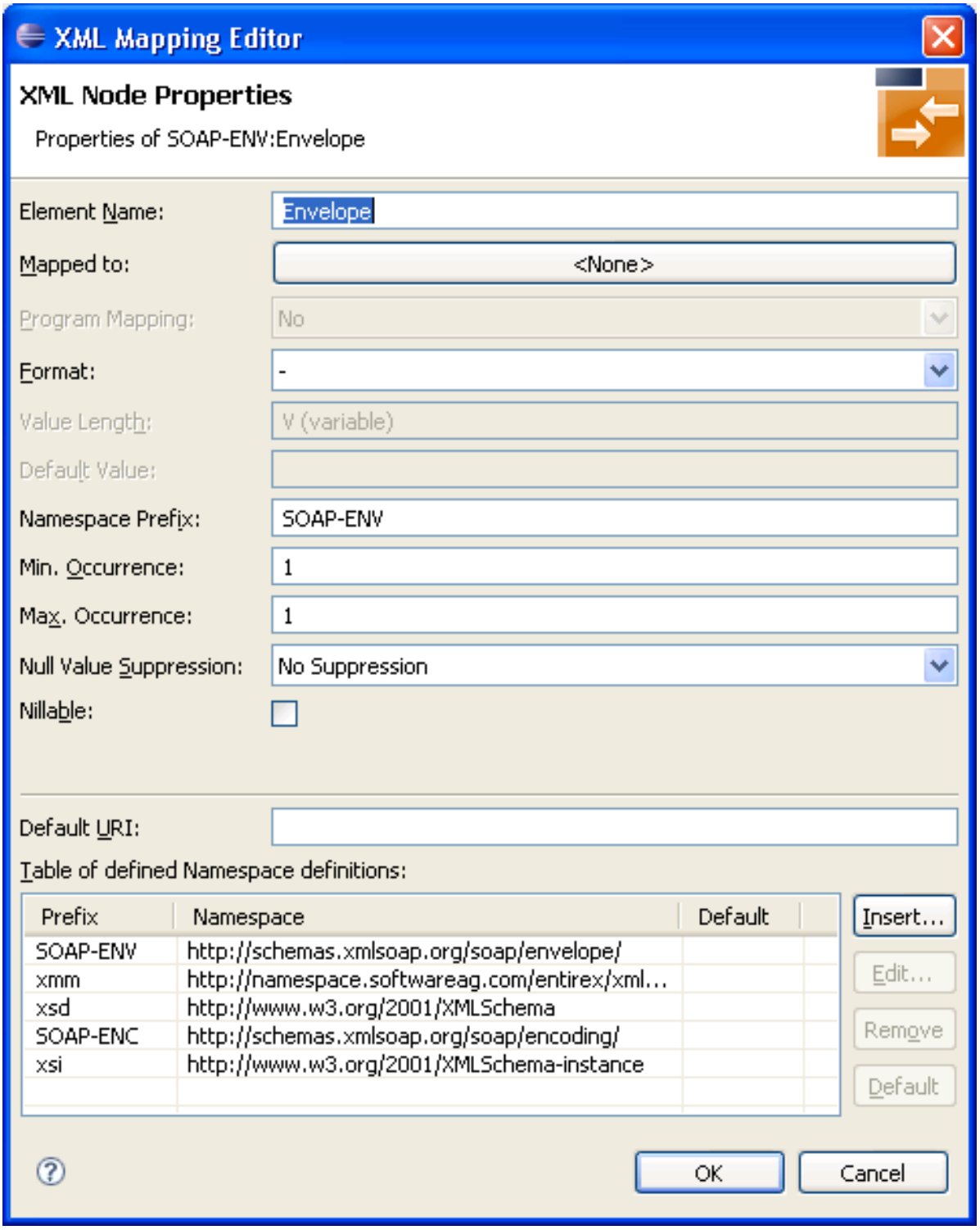
5 Select (unused) namespace entry/entries in list and choose **Remove**.

6 Press **OK**.

7 Repeat these steps for the XML response (select the **XML Response** tab and repeat the steps above).

8 Save the XMM file.

9 Switch to **XML Samples** tab, which shows the following:



10 The XMM file defines the following:

```
...
<FromXml>
<Method relatedIdlLibrary="EXAMPLE" relatedIdlProgram="CALC" encoding="UTF-8"
useIncomingEncoding="true">
<XmlNode name="Envelope" length="0" min="1" max="1" namespacePrefix="SOAP-ENV"
nullSuppression="NVS_NONE" nullValue="" >
<XmlNamespaceDef ↩
prefix="SOAP-ENV" uri="http://schemas.xmlsoap.org/soap/envelope/"/>
<XmlNamespaceDef ↩
prefix="SOAP-ENC" uri="http://schemas.xmlsoap.org/soap/encoding/"/>
<XmlNode name="Header" length="0" min="1" max="1" namespacePrefix="SOAP-ENV"
nullSuppression="NVS_NONE" nullValue="" >
<XmlNode name="SOAPAction" format="string" type="xsd:string" length="0" min="1" ↩
max="1"
default="CALC" nullSuppression="NVS_NONE" nullValue="" programNode="ev" >
</XmlNode>
...
```

3 EntireX XML Tester

Introduction to the XML Tester

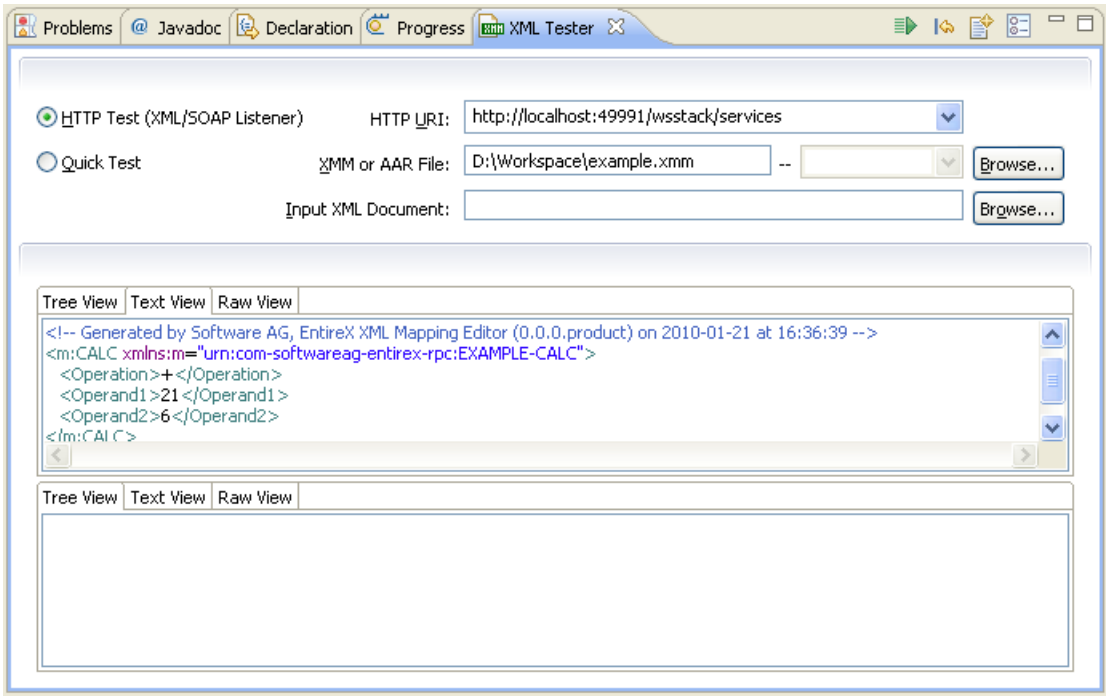
Using the XML Tester you can send an XML document to the EntireX XML/SOAP Listener.

The XML Tester supports drag-and-drop editing, and you can restore the last used session within a workspace. If the mapping file has changed, the views for request and response are cleared.

The XML Tester is provided by the EntireX Workbench as an Eclipse View and can be opened from the XML Mapping Editor or XML/SOAP Wrapper. It is also accessible from **Windows > Show View > Other... > Software AG > XML Tester**. The following test modes are available:

- Quick Test (Using the Java Interface)
- HTTP Test (Using the EntireX XML/SOAP Listener)

See sample screen and description of options below.



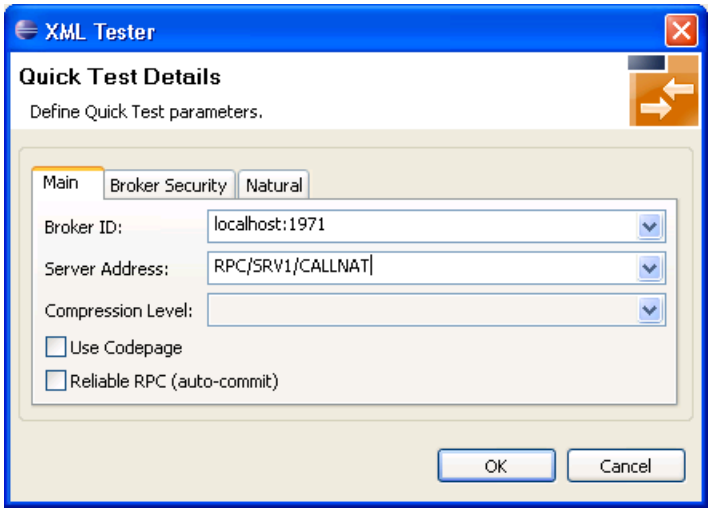
XML Tester Options

Name	Type	Description
Quick Test	Check box	Check this box to test the Java interface of the XML/SOAP Runtime. If checked, the **Settings** action will open the **Quick Test Details** dialog.
XMM or AAR File	Text field	The AAR file, which contains one or more mapping files (XMM files), or the XMM file, which contains the mapping information. Accept absolute path. If the file does not exist, the action **Create Sample XML...** is disabled. When the file is found, the Broker ID and Server Address are read and the corresponding fields in the **Quick Test Details** dialog are filled.
	Combo box	Behavior depends on content of field **XMM or AAR File**: ■ If this field contains an AAR file, this combo is enabled and filled with all the mappings stored in the archive. Select the mapping you want from the list. ■ If this field contains an XMM file, the combo box is disabled.
Browse...	Button	Open a **File open** dialog to select the XMM or AAR file from the file system.
HTTP Test	Check box	Check this box to test the HTTP interface (EntireX XML/SOAP Listener) of the XML/SOAP Runtime. If checked, the **Settings** action will open the **HTTP/HTTPS Parameters** dialog.
HTTP URI to contact	Combo box	URI of the EntireX XML/SOAP Listener. Will store the last ten called URLs.
Input XML Document to send	Text field	File name of the XML Document to be sent.
Browse...	Button	Open a **File open** dialog to select the XML Document file from the file system.
Create Sample XML...	Action 	Open the **Create Sample XML...** dialog, see below. If the text file for the XMM file does not contain a valid file path, the **Create Sample XML...** button is disabled.
input - area	Tree/Text/Raw view	Input field for the XML data to be sent to the Broker, this area is filled with the text/tree representation of the selected file, see **Input XML Document to send**. If empty, the **Play** action is disabled.
output - area	Tree/Text/Raw view	Output field for the XML data response from the Broker. By default, the response will be displayed in platform encoding. But if the XML data request contains a valid encoding in the XML declaration, this encoding will be used and a tooltip text will inform you of this. This view will also display the status and errors.
Settings	Action 	Open either **Quick Test Details** or **HTTP/HTTPS Parameters** dialog. See **Quick Test** and **HTTP Test** above.

Name	Type	Description
Play/Stop	Action /	Start the test. This action is disabled if there is no text in the **input - area**.
Reset	Action	Reset the input and output areas.

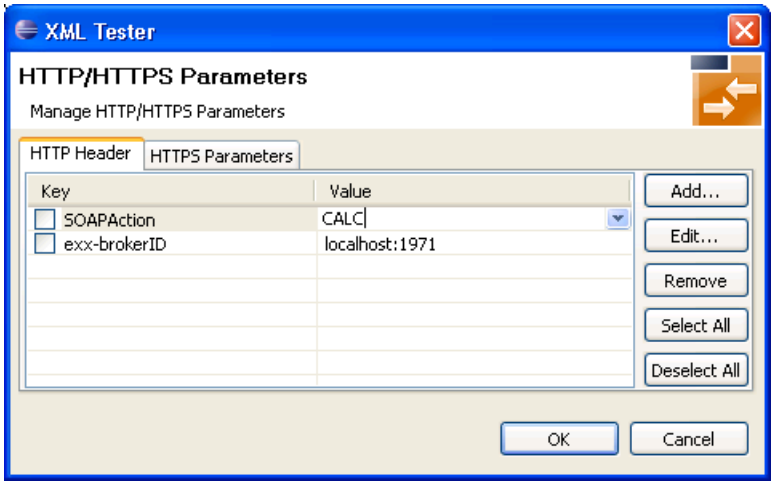
Quick Test Details Dialog

Use the **Quick Test Details** dialog to change the Quick Test parameters. The Broker ID, Server Address, the Security Settings, the Natural Settings can be modified for this XML Tester session. **Compression Level** is a non-editable box that contains all possible values. **Broker ID** and **Server Address** are editable dialog boxes and hold the last five used values. Use the **Reliable RPC** check box to turn reliable messaging on or off (mode is AUTO-COMMIT). See *Reliable RPC*.

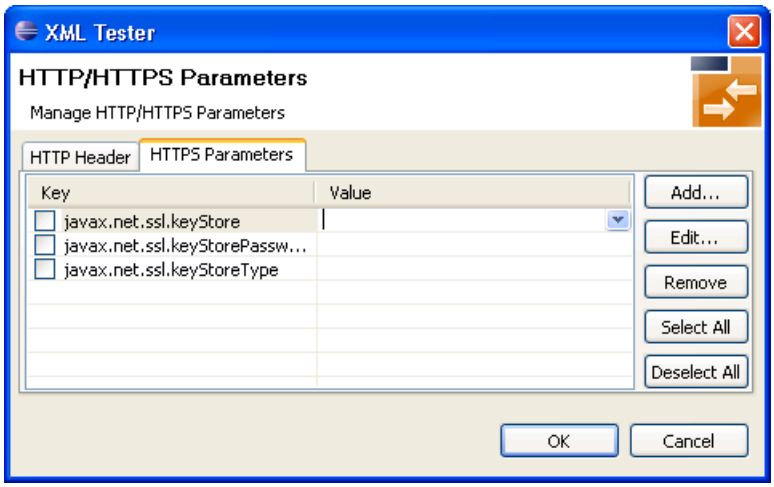


XML Tester HTTP/HTTPS Parameters Dialog

With the **HTTP Header** tab you can add more information for the call and set the SOAPAction. An entry must be selected to be active. The **Value** box holds the last five used values for the corresponding property. It is editable, which means you can change a value by clicking directly in the table.



With the **HTTPS Parameters** tab you can set parameters required for an HTTPS connection. An entry must be selected to be active. The **Value** box holds the last five used values for the corresponding property. It is editable, which means you can change a value by clicking directly in the table.

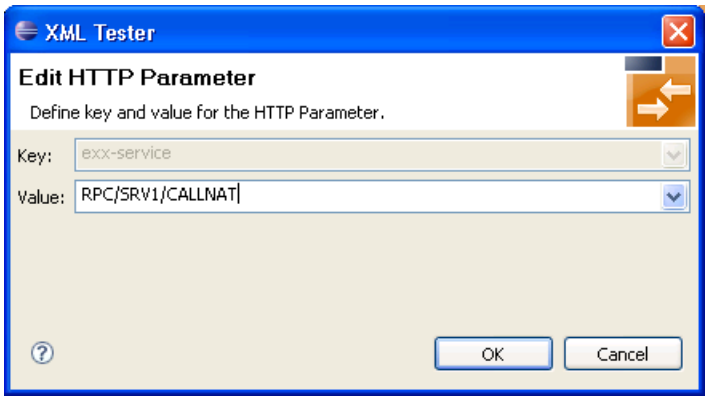


Add New HTTP Parameter Dialog

With the **Add new HTTP(S) Parameter** dialog you can add a new HTTP or HTTPS parameter. All Software AG's predefined HTTP parameters are listed in the **Key** box. For HTTPS, the **Key** box contains predefined properties for HTTPS communication.

Edit HTTP Parameter Dialog

With the **Edit HTTP(S) Parameter** dialog you can change the value of a property. The **Value** box is editable and holds the last five used values. The **Key** value for all predefined properties cannot be modified.



Using the XML Tester

Using the XML Tester you can send an XML document to the EntireX XML/SOAP Listener.

To open the XML Tester

- In perspectives other than EntireX, choose **Window > Show View > XML Tester**, in other perspectives, choose **Window > Show View > Other ... > Software AG > XML Tester**.

  Or:

  Choose **Window > Open Perspective > Other...** and select EntireX Perspective. The **XML Tester** view is part of this perspective.

  Or:

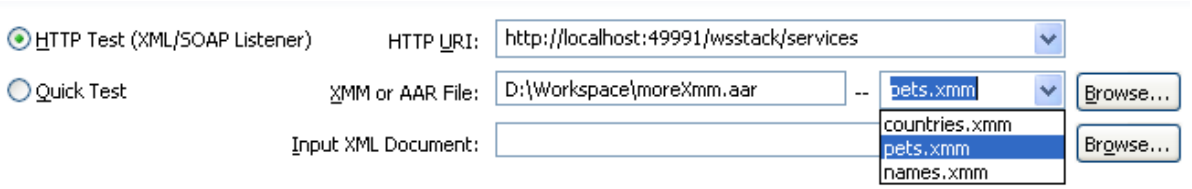
  In the XML Mapping Editor, switch to the **Overview** page, choose **Quick Test** or **HTTP Test** in the testing section.

  Or:

  In the XML Mapping Editor, switch to page **XML Samples**, select one document, choose **XML Tester** in the context menu.

To run a test

1 Load the mapping information. In field **XMM or AAR File**, enter the absolute path to the AAR or XMM file that is being tested (you can use the **Browse...** button or a drag-and-drop operation). If you enter an AAR file, all mappings contained in the archive are listed in the combo box next to the **XMM or AAR File** field.



2 Select the testing interface (Java or HTTP):

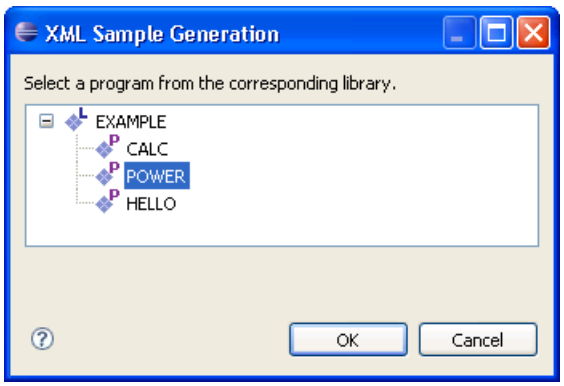
- Java interface: the **XMM or AAR File** field must contain the full path and name of an XMM or AAR archive. Choose the **Quick Test** option to use this interface.
- HTTP interface: enter the name of the URI of the running EntireX XML/SOAP Listener. Choose the **HTTP Test** option to use this interface.

3 Complete the input area:

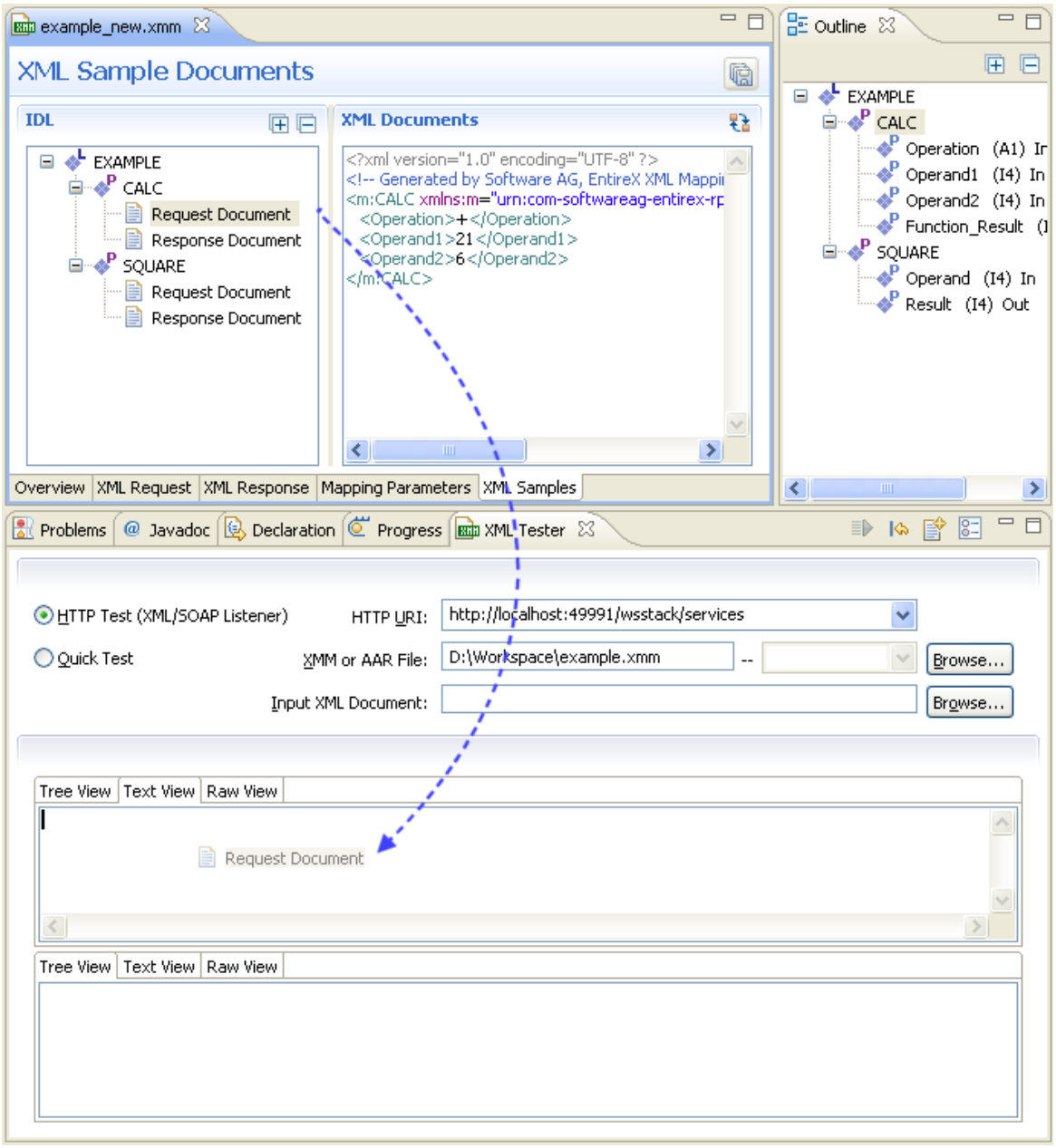
- Type your XML document.
- Use the **Create Sample XML...** button and follow the dialog.

  Note: If the IDL file contains only one library with one program, this is used for the sample generation without opening the dialog.

  Example dialog:



- Load sample document, using drag-and-drop operation on the sample page of XML Mapping Editor or file browser.



The following tabs are available:

- The **Text View** tab will try to color the request syntactically, making the XML easier to read.
- The **Raw View** tab is initially empty. When the first request is sent, it is filled with the exact message sent together with information on the request method and the used HTTP headers with their values.
- From then on the **Raw View** is displayed and is updated with the last sent request.

- Select an existing XML document (you can use the **Browse** button) and enter it in the field **Input XML file to send**.

4 If the testing interface is Java, the **Settings** action will open the **Quick Test Details** dialog.

The dialog contains the three tabs with the following settings:

- Main
- Broker Security
- Natural Security

See *Quick Test Details Dialog*.

5 If using option **HTTP Test**, the **Settings** action will open the **HTTP/HTTPS Parameters** dialog. The dialog contains two tabs with settings for the HTTP call:

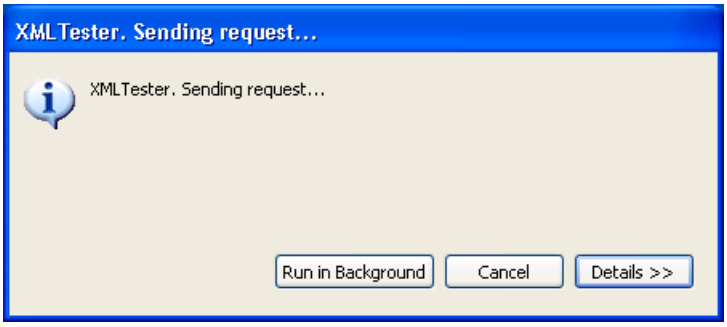
- HTTP Header
- HTTPS Parameters

See *XML Tester HTTP/HTTPS Parameters Dialog*.

6 Send the request by choosing action **Play**. If the testing interface is HTTP, the **Sending request** dialog appears. Here you have several options:

- hide the dialog by pressing **Run in Background** button
- cancel the request. This will terminate the session with the server
- show details about all running Eclipse jobs

A progress bar on the Eclipse status bar shows to indicate that the request is being sent.



If you used the **Quick Test** option, you cannot cancel the request.

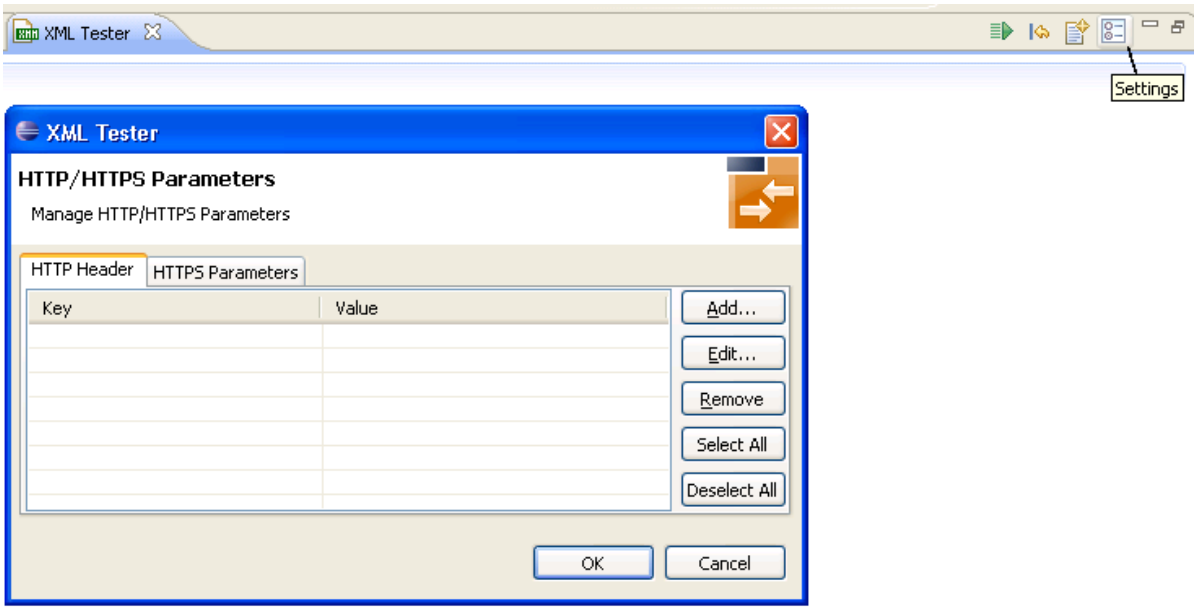
7 When a response is received, it is displayed in the output area. Errors are also displayed in the output area.

The **Text View** tab displays the XML response as formatted and syntactically colored text. The **Raw View** tab displays the response "as is". Information on the used request method, response code and HTTP headers with their values is displayed for the XML response.

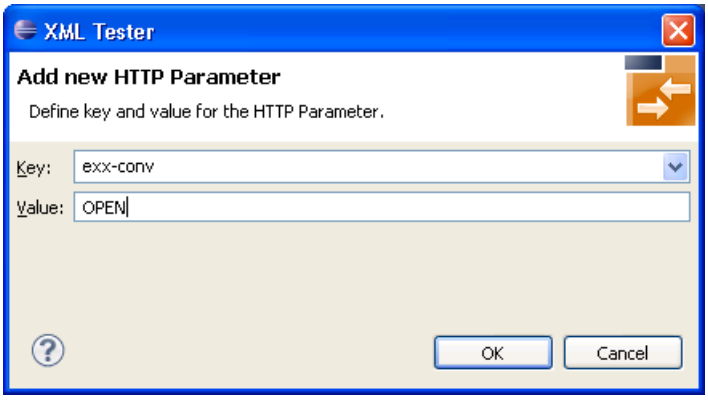
XML Tester for Conversational RPC

To use the XML Tester with conversational RPC

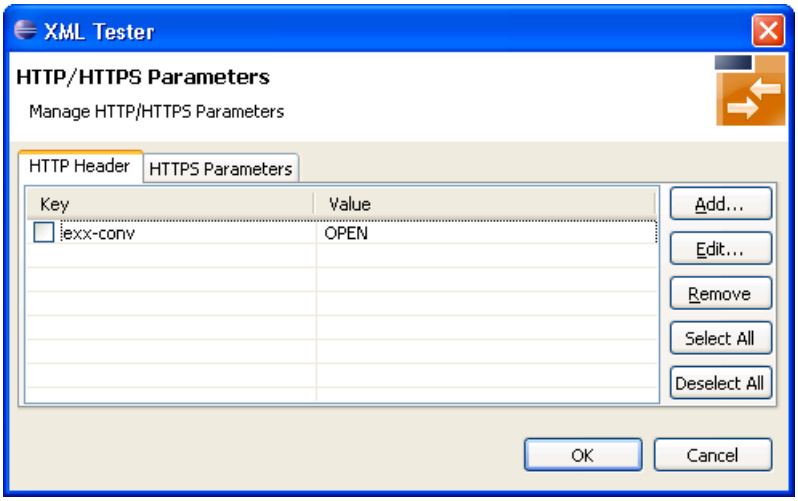
1 Open **Settings**.



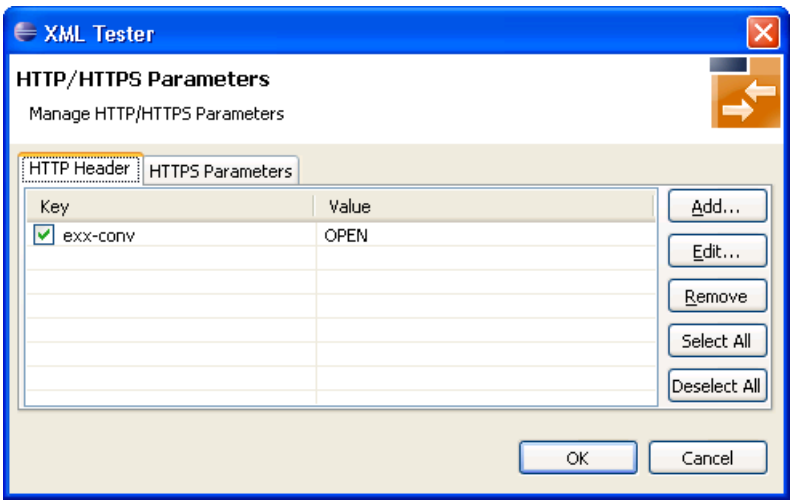
2 Add key `exx-conv` and set value to "OPEN".



3 Press **OK**.



4 Select `exx-conv`.



5 Press **OK**.

6 Send the request.

7 The response contains the `exx-xml-sessionID` used for conversational communication. This parameter is also added to HTTP/HTTPS parameters for the next call(s) automatically. The parameter `exx-conv` is deactivated simultaneously.

8
```
<!-- snippet with exx-xml-sessionID in response document -->
     <axis2ns1:EntireX xmlns:axis2ns1="urn:com.softwareag.entirex.xml.rt">
        <exx-xml-sessionID>XML610C044029044C05</exx-xml-sessionID>
     </axis2ns1:EntireX>
<!-- snippet -->
```

9 To finish the conversation, open the settings again, activate the `exx-conv` and set its value to "COMMIT" or "BACKOUT".

10 The conversation is closed with the next sent call.

4 Using the XML Mapping Editor in Command-line Mode

Command-line Options

See *Using the EntireX Workbench in Command-line Mode* for the general command-line syntax. The table below shows the command-line options for the XML Mapping Editor.

Task	Command	Option	Description
Create SOAP-conformant XML mapping for all programs.	`-map:soap`		
	`-map:soap1.1`		Supported for compatibility with older versions.
Create attribute-preferred XML mapping for all programs.	`-map:xmlattributes`		
Create element-preferred XML mapping for all programs.	`-map:xmlelements`		
Create element-preferred XML mapping with XML Schema for all programs	`-map:xmlwithxsd`	`-namespace`	If *`namespace`* is specified, the namespace URI for the program node. XML Schema uses this namespace URI as the target namespace.
Create sample XML documents for all programs.	`-xml:sample`		

Example

```
<workbench> -map:soap /Demo/example.idl
```

where *<workbench>* is a placeholder for the actual Workbench starter as described under *Using the EntireX Workbench in Command-line Mode*.

The generated XML mapping file (XMM) will be stored in parallel to the IDL file, for example in project *Demo*.

Status and processing messages are written to standard output (stdout), which is normally set to the executing shell window.

5 IDL to XML Mapping with the XML Mapping Editor

The EntireX XML Mapping Editor allows you to map XML document structures to IDL libraries, programs and parameters. The mappings can be defined for the request and response to the server application, or from the server to the client. The input for the XML Mapping Editor can be a Software AG IDL file and/or an IDL-XML mapping file (perhaps produced by a previous XML Mapping Editor session or by importing a WSDL file, XML Document or XML Schema). The output is an IDL-XML mapping file, other XML structure definitions (such as sample XML files), and perhaps a created or changed IDL file.

There are two ways to create an IDL to XML mapping: automatically (and then modify the resulting structures manually), or manually.

Mapping IDL to XML Automatically

Scope

The following approaches are available:

- Map all programs in the current IDL file with the element-preferred strategy.
- Map all programs in the current IDL file with the attribute-preferred strategy.
- Create a SOAP mapping for all programs in the current IDL file.
- Mapping and XML Schema generation for all programs in the current IDL file. This creates an element-preferred mapping.

The mapping strategies generate XML tree nodes for all IDL parameters with the appropriate direction:

- For `IN` and `INOUT` IDL parameters, XML Request nodes are generated.
- For `INOUT` and `OUT` IDL parameters, XML Response nodes are generated.
- Fault trees (Error trees) are generated, see **Fault Document Manager** under *XML Response Page*.

Format, length and default values of the XML nodes are set according to the IDL parameter:

In the table below, the following metasymbols and informal terms are used for the IDL.

- The metasymbols "[" and "]" surround optional lexical entities.
- The informal term *number* (or in some cases *number1*. *number2*) is a sequence of numeric characters, for example 123.

Software AG IDL	Description	XML Data Type
A*number*	Alphanumeric	Variable-length `string` node with maximum length.
AV	Alphanumeric variable length	Variable-length `string` node.
AV[*number*]	Alphanumeric variable length with maximum length	Variable-length `string` node with maximum length.
B*number*	Binary	`binary` node with length of binary representation (base 64)
BV	Binary variable length	Variable-length `binary` node.
BV[*number*]		Variable-length `binary` node with maximum length.
D	Date	XML `date` node. The length depends on the specified time pattern length. This is trimmed, and what is included in single quotes will be counted (without the single quotes). For example: `'T'HH:mm:ss` has length=9.
F4	Floating point (small)	`float` node of length 32.
F8	Floating point (large)	`float` node of length 64.
I1	Integer (small)	`integer` node of length 4.
I2	Integer (small)	`integer` node of length 6.
I4	Integer (small)	`integer` node of length 11
K*number*	Kanji	Variable-length `string` node with maximum length.
KV	Kanji variable length	Variable-length `string` node.
KV[*number*]	Kanji variable length with maximum length	Variable-length `string` node with maximum length.
L	Logical	`Boolean` node.
N*number1*[.*number2*]	Unpacked decimal	`number` node. If *number* > 0, length increased by 2, otherwise by 1.
NU*number1*[.*number2*]	Unpacked decimal unsigned	`number` node. If *number* > 0, length increased by 2, otherwise by 1.
P*number1*[.*number2*]	Packed decimal	`number` node. If *number* > 0, length increased by 2, otherwise by 1.
PU*number1*[.*number2*]	Packed decimal unsigned	`number` node. If *number* > 0, length increased by 2, otherwise by 1.

Software AG IDL	Description	XML Data Type
T	Time	dateTime node. The length depends on the specified time pattern length. This is trimmed, and what is included in single quotes will be counted (without the single quotes). For example: yyy-MM-dd'T'HH:mm:ss has length=19.
U*number*	Unicode	Variable-length Unicode node with maximum length.
UV	Unicode variable length	Variable-length Unicode node.
UV*number*	Unicode variable length with maximum length	Variable-length Unicode node with maximum length.

The resulting XML structures can be modified, see ***Mapping IDL to XML Manually***.

The automatic mapping process can be fine-tuned by mapping parameters to be set before mapping.

For information on the attribute-preferred and element-preferred strategies see *XML Structures and IDL-XML Mapping* in the XML/SOAP Wrapper documentation.

Mappings

For SOAP additional type attributes are generated. They describe the XML Schema compliant data types of the elements.

SOAPAction

The SOAPAction tag in the SOAP header will be moved to the HTTP header. The SOAPAction element is generated automatically with SOAP mapping. Verify the default setting. If SOAPAction is missing, insert it: select the SOAPHeader node and add a new child node with the name SOAPAction. Then select the new element node with the name SOAPAction, open the XML Details panel and enter the value for that HTTP header into the Default attribute field. The NullValueSuppression property should have the value “Suppress Element”.

Using Mapping Parameters

To use the mapping parameters

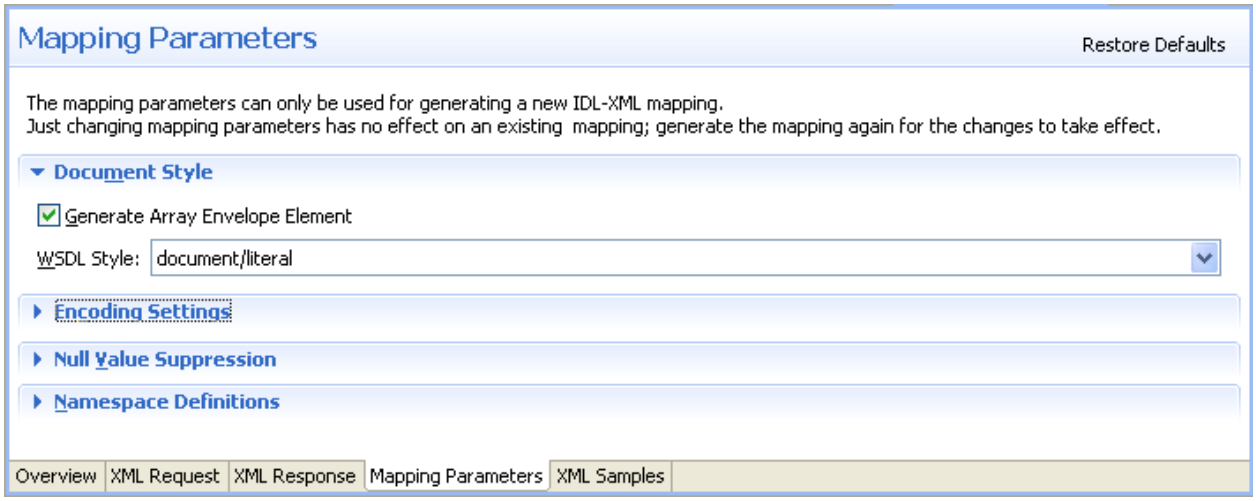
1 Switch to the Mapping Parameters page or define the XML Mapping Editor preferences.

2 Check the check boxes, dialog boxes and complete the text fields you need.

Mapping Parameters

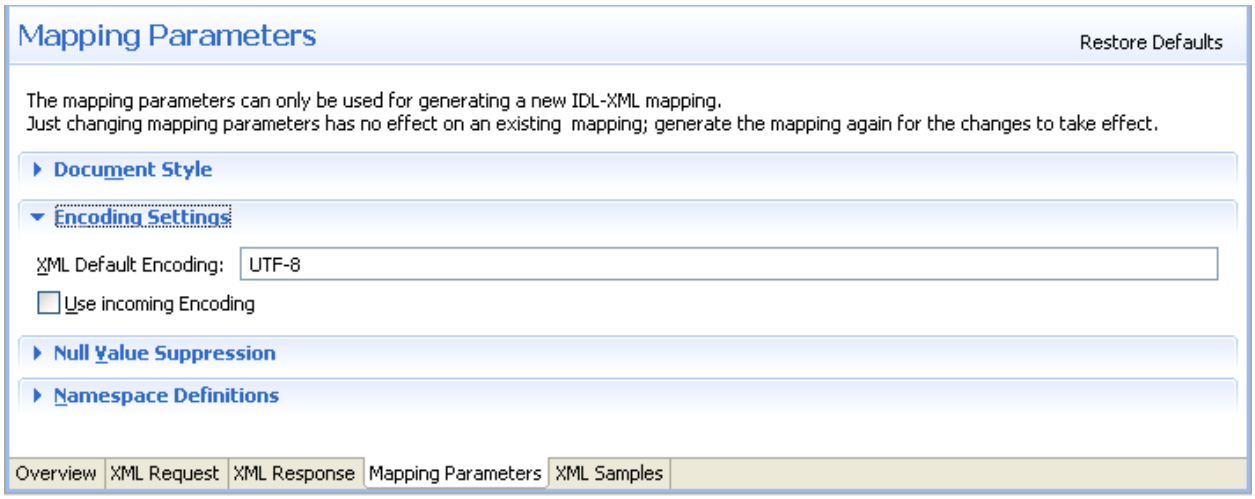
The mapping parameters are on the following sections.

Document Style



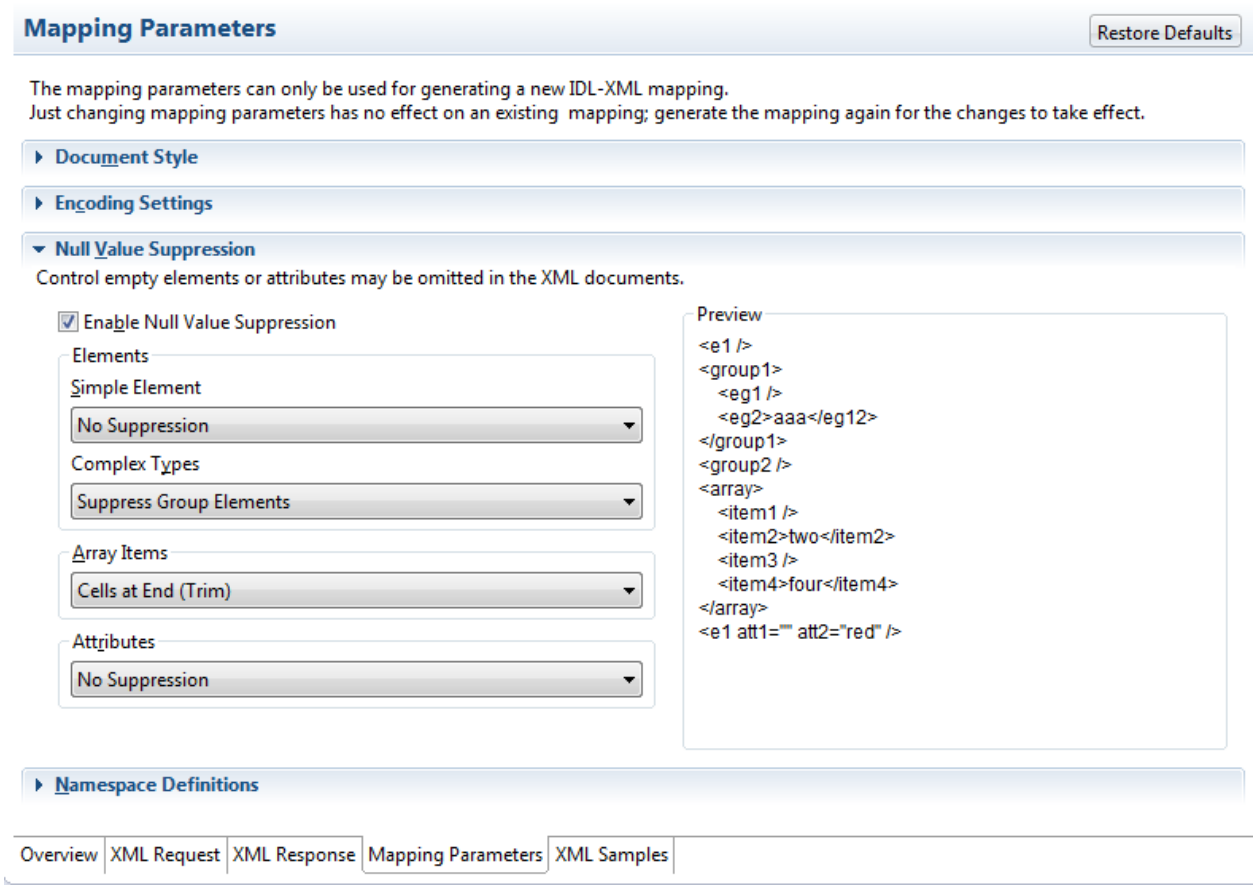
Parameter	Description
Generate Array Envelope Element	Determines whether for each array a surrounding additional element (envelope) is generated or not.
WSDL Style	Prepare the SOAP Mapping for selected WSDL Style. Possible values: document/literal or rpc/encoded.

Encoding Settings



Parameter	Description
XML Default Encoding	This encoding is used for the sent XML/SOAP document if the box **Use incoming XML encoding** is not checked (for XML-based clients), or if the XML/SOAP RPC Server is used.
Use Incoming Encoding	Check this box to enable the XML/SOAP Wrapper to use same encoding for the incoming document as for the outgoing document.

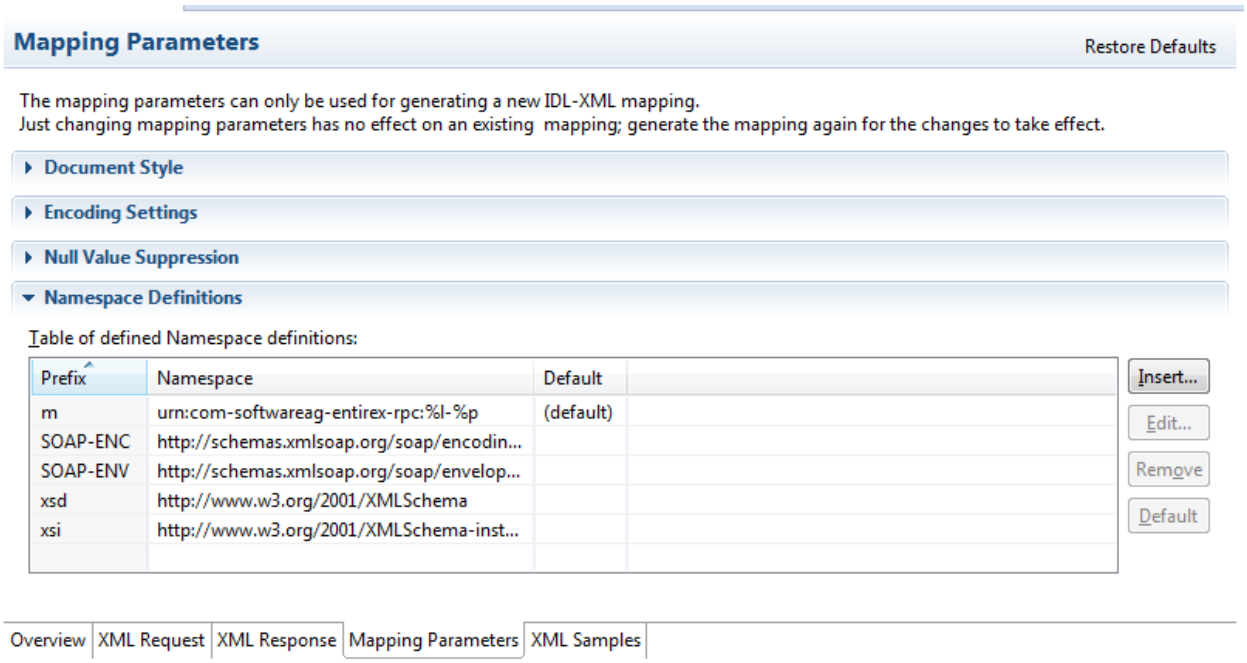
Null Value Suppression



Parameter	Description
Enable null value suppression	Switch on/off the null value suppression.
Simple Element	Suppress Elements. Possible Values: No Suppression or Suppress Element.
Simple Attribute	Suppress Attributes. Possible Values: No Suppression or Suppress Attribute.
Array Types	Suppress Array Types. Possible Values: No Suppression, All empty cells or Cells at end (Trim).
Complex Types	Suppress Complex Types. Possible Values: No Suppression (no special handling of complex types - null value suppression defined for 'Simple Element' is used), or Suppress Group Elements.

For more details on null value suppression, see *Null Value Suppression* under *Writing Advanced Applications with the XML/SOAP Wrapper*.

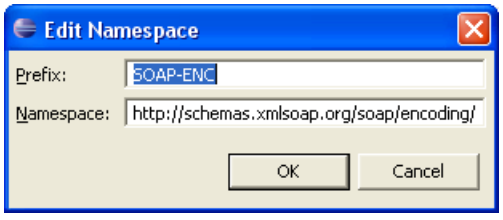
Namespace Definitions



Parameter	Description
Namespace Definitions Table	Manage all Namespaces with prefix and URI.

New namespace definitions can be entered with **Add...** . To change an entry choose **Edit...** . All selected entries can be deleted with the **Remove** button. With the **Default** button you mark the Namespace used for the Payload Root node, e.g. in XML Mapping the Root Element and in SOAP Mapping the first element in the SOAP-Body.

The Edit dialog (and similar Add dialog) looks like:



To map automatically with your preferred result

- On the Mapping section of the Overview Page choose the preferred Mapping style and press the **Generate All** button.

The various generated XML Structures can be modified, see *Mapping IDL to XML Manually*.

If errors/warnings have occurred, the Validity Check window will be displayed. The errors/warnings for all programs will be found in the Problems View. In the first line of this View you will find a short summary of the problems. If you click on an error/warning, the XML structure for it is displayed with the invalid node highlighted.

Caution: There are rare cases when the SOAP default mapping generates warnings (for example for IDL files with cascaded arrays). A dialog is displayed stating that illegal mappings have been generated. The warnings mostly deal with duplicate element names. You may ignore them as long as you do not use the mapping for WSDL or XML Schema. For details, see the Problem View.

Mapping IDL to XML Manually

This section covers the following topics:

Summary of Steps

To map an IDL file manually to an XML structure or modify an IDL-XML mapping file

1 Choose direction (XML Request or XML Response).

2 Add/modify XML nodes.

3 Map IDL nodes to XML nodes.

Choosing XML Request or XML Response XML Structure

To choose the XML Structure

1 Select the XML Request to get the IDL `IN` and `INOUT` Parameter Mapping.

2 Select the XML Response to get the IDL `OUT` and `INOUT` Parameter Mapping.

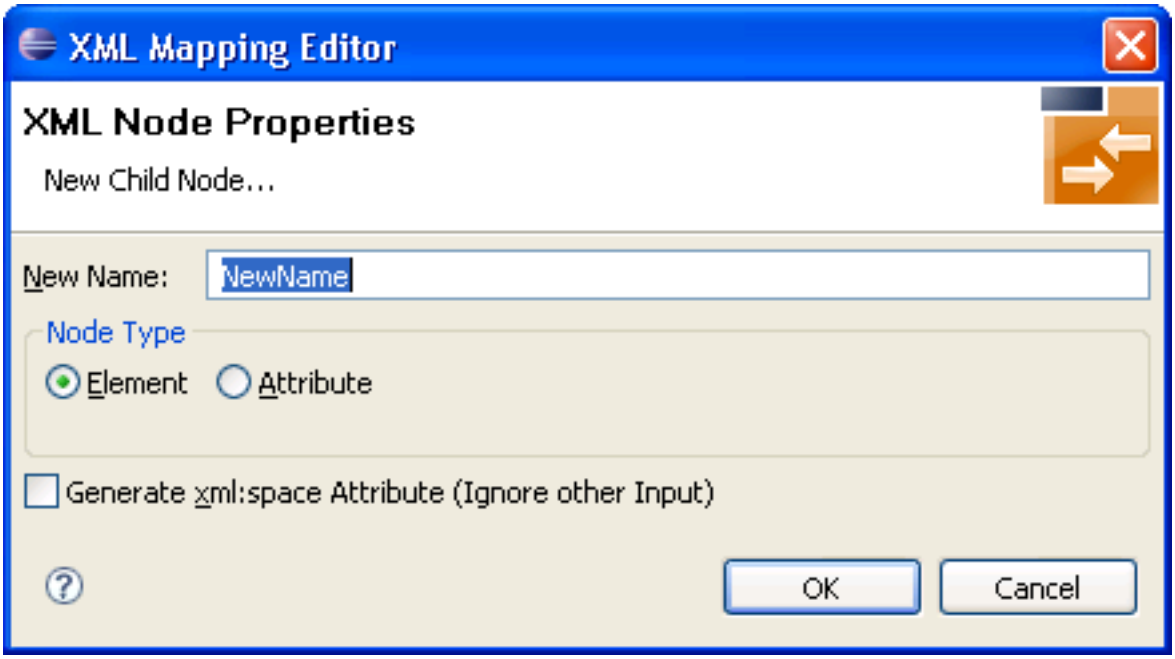
Adding an XML Node

To create a child node to an existing (selected) XML node

1 Click on the existing node.

2 From the context menu, choose **New child node**.

A dialog window is displayed.



3 Enter the new node's name and type (element or attribute) and use **OK**.

To insert a new node before or after an existing (selected) XML node

1 From the context menu, choose **Insert before** or **Insert after**, respectively.

A dialog window is displayed.

2 Enter the new node's name and type (element or attribute) and use **OK**.

Note that new attribute nodes do not have a format, length and default property assigned. You must assign at least a format to the node before saving the mapping file. See *Quick Test Details Dialog*. Use the *Validity Checks* to make sure that all nodes have the necessary properties.

Grouping XML Elements or Attributes

You can introduce new elements by grouping one or more existing elements or attributes. See also *XML Structures and IDL-XML Mapping* in the XML/SOAP Wrapper documentation.

To group XML elements or attributes

- Add a new XML part and move the existing nodes into the new node.

You can use cut-and-paste or a drag-and-drop operation to reorder the XML parts. Four move functions are available in the XML context menu: **Bring to top**, **Bring to bottom**, **Move up**, **Move down**, which are equivalent to drag-and-drop operation. See *Using the Context Menu* and *Using Drag-and-drop*.

Mapping an IDL Node to an XML Node

To add (or modify) an IDL-XML node mapping link

1 Click into the **Mapped to** field in the *XML Node Properties Dialog*.

A dialog displays the possible IDL mapping links.

2 Choose one of the menu items to map the IDL node to the XML node.

If the XML node had a previous IDL mapping link, it is released.

Or:

Use a drag-and-drop operation to move the IDL node to be mapped directly to the XML node. If the XML node had a previous IDL mapping link, it is released. See *Using Drag-and-drop*.

Or:

Use the keyboard to tab into the Mapped to text field, and press Enter. This will open the dialog Mapped to. Select a mapping link from the list and use the **OK** button to confirm. Use the **Cancel** button to cancel the action.

IDL-XML mapping uses the full path notation. This notation is a slash-separated list of IDL node names, starting with the program name, ending with the addressed IDL node, and containing all IDL nodes between program and the IDL node (the path from the program name to the node).

To modify the format of the XML node

- Select an entry from the dialog box in the *XML Node Properties Dialog*.

The dialog box contains the possible format values. You cannot add new format codes.

Unmapping XML Nodes

To unmap XML nodes

1 Select one or more XML nodes mapped to IDL node(s).

2 From the context menu, choose **Unmap**.

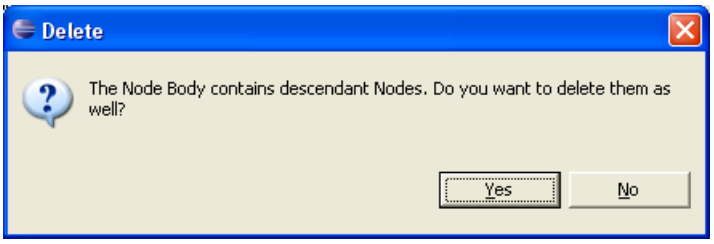
This will unlink the IDL and XML nodes.

Deleting Nodes

To delete arbitrary elements or attributes

1 Select the node(s) to delete.

2 From the context menu, choose **Delete**. If nodes have descendants, the following window is displayed:



3 To delete the subtree attached to the node, choose **Yes**.

4 To keep the elements of the subtree and move them one level higher, choose **NO**.

Caution: If IDL nodes are not mapped in the incoming XML request, the result for the runtime component is that the corresponding IDL parameters are filled with zero strings.

Deleting XML Structures

To delete the complete IDL to XML mapping (all XML Structures)

- In the mapping menu, choose the item **New mapping for all programs**.

To delete the currently selected XML Structures

- In the mapping menu, choose the item **New mapping**.

Modifying the Element or Attribute Name

In the automatically mapped structures the XML node names are built from the IDL parameter names.

To change the node name

- Choose **Rename** from the context menu.

  Or:

  Double-click on the name field in the *XML Node Properties Dialog*.

To avoid accidental changes, there is no other way to modify the name.

When the name of an XML part is modified, various checks of the XML structure are performed, e.g. attribute name duplication, IDL mapping legality. The IDL-XML-mapping link is not influenced by the name change.

Changing Elements to Attributes and Vice Versa

You have the possibility to switch the node type between element and attribute. Note that attributes may not have descendant nodes and namespace definitions.

To switch between elements and attributes

- From the context menu, choose **Set to Element** and **Set to Attribute**.

This function works for multiple nodes, too.

Important: Arrays must always be modelled with elements.

Assigning Default Values

Every XML part can contain default values in case elements are missing in the incoming XML document.

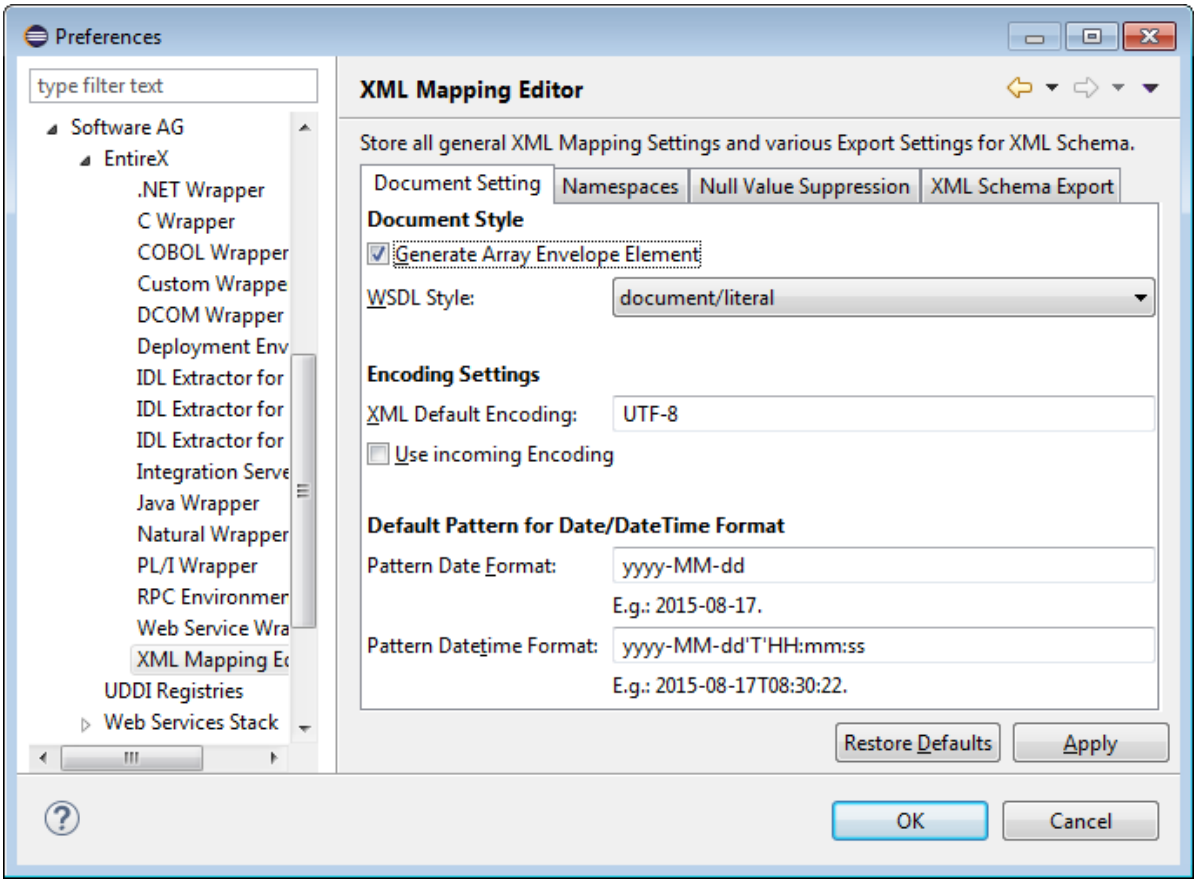
To set the default values

- In the *XML Node Properties Dialog*, enter a **Default Value**.

Defining the XML Encoding

To define default encoding for IDL to XML mapping

1 Open the IDL properties and choose the XML tab



2 Modify the Encoding Settings section:

- **XML default encoding (default: UTF-8)**
  This is the encoding that will be used to write the mapping file itself and if there are no XML declarations in the incoming document. Together with the next property (Use incoming XML encoding) this encoding will be used for the response documents.
- **Use incoming XML encoding (default: checked)**
  If this check box is checked, the last two properties are disabled and the same encoding of the incoming document will be used for the response documents.

Validity Checks

The Validity check is automatically performed for automatic mapping and when an existing IDL-XML mapping file is opened, but can also be performed manually.

The following validity checks are available:

1. Checks that all elements have unique tags - produces warnings
2. Checks that all attributes of an element have unique names (mandatory)
3. Checks that the generated XML parts have correct property values (mandatory), e.g. minimum occurrences less than or equal to maximum occurrences
4. Checks that the IDL mapping is valid (mandatory) and the IDL parameters are not mapped to multiple XML parts (warning)
5. Checks that all IDL parameter nodes are mapped to one or more XML parts, (so that the runtime component finds XML values for all IDL parameters or vice versa) (warning)

The first four of the checks apply to XML tree nodes, i.e. search for invalid settings or mapping rules of XML elements or attributes. The fifth check applies to the IDL tree.

The check messages may be divided into warnings (message text starts with (W)) and errors. Warnings can even be caused by automatic mapping. Generally, they can be ignored. Errors should not be ignored.

Exporting the IDL-XML Mapping to an XML Schema

To save the XML structure tree as an XML Schema (XSD) document

- The current XML Mapping can be exported as XML Schema via the context menu of the XMM file. The XML Schema style "Russian Doll" or "Venetian Blind" can be selected in the preferences.

See ***Mapping IDL Data Types to an XML Schema (XSD)*** for mapping of IDL data types to XSD.

See also *XML Schema Parser Standards Conformance* and *XML Schema Writer Standards Conformance* under *XML Schema Standards Conformance (XML/SOAP Wrapper)* in the XML/SOAP Wrapper documentation.

6 Mapping IDL Data Types to an XML Schema (XSD)

In the table below, the following metasymbols and informal terms are used for the IDL.

- The metasymbols "[" and "]" surround optional lexical entities.
- The informal term *number* (or in some cases *number1*. *number2*) is a sequence of numeric characters, for example 123.

IDL Data Type	Description	XMM	XSD
A*number*	Alphanumeric	string	`<xsd:element name="name" form="unqualified">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:string">` `  <xsd:maxLength value="number"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>`
AV	Alphanumeric variable length	string	`<xsd:element name="name" form="unqualified">` ` <xsd:simpleType>` `  <xsd:restriction ↩` `base="xsd:string"></xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>`
AV[*number*]	Alphanumeric variable length with maximum length	string	`<xsd:element name="name" form="unqualified">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:string">` `  <xsd:maxLength value="number"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>`

IDL Data Type	Description	XMM	XSD
B*number*	Binary	binary	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="*base64Length*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** *base64Length* = 4 * rounded up(*number* /3)
BV	Binary variable length	binary	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:base64Binary"></xsd:restriction> </xsd:simpleType> </xsd:element>
BV[*number*]	Binary variable length with maximum length	binary	<xsd:element ↩ name="*name*" form="unqualified"> ↩ <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:maxLength value="*base64Length*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** *base64Length* = 4 * rounded up(*number* /3)
D	Date	date:yyyy-MM-dd	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:date"></xsd:restriction> </xsd:simpleType> </xsd:element>
F4	Floating point (small)	float	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:float"></xsd:restriction> </xsd:simpleType> </xsd:element>
F8	Floating point (large)	float	<xsd:element ↩ name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:double"></xsd:restriction> </xsd:simpleType> </xsd:element>

IDL Data Type	Description	XMM	XSD
`I1`	Integer (small)	integer	`<xsd:element name="name" form="unqualified">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:byte">` `  <xsd:totalDigits value="3"/>` `  <xsd:fractionDigits value="0"/>` ` </xsd:restriction>` `</xsd:simpleType>` `</xsd:element>`
`I2`	Integer (medium)	integer	`<xsd:element name=" name" ↩` `form="unqualified">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:short">` `  <xsd:totalDigits value="5"/>` `  <xsd:fractionDigits value="0"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>`
`I4`	Integer (large)	integer	`<xsd:element name=" name" ↩` `form="unqualified">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:int">` `  <xsd:totalDigits value="10"/>` `  <xsd:fractionDigits value="0"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>`
`Knumber`	Kanji	string	`<xsd:element name=" name" ↩` `form="unqualified">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:string">` `  <xsd:maxLength value="number"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>`
`KV`	Kanji variable length	string	`<xsd:element name=" name" ↩` `form="unqualified">` ` <xsd:simpleType>` `  <xsd:restriction ↩` `base="xsd:string"></xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>`

IDL Data Type	Description	XMM	XSD
KV[*number*]	Kanji variable length with maximum length	string	<xsd:element name="*name*" ↩ form="unqualified"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
L	Logical	boolean	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:boolean"></xsd:restriction> </xsd:simpleType> </xsd:element>
N*number1*[.*number2*]	Unpacked decimal	numeric	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*number1* ↩ + *number2*"/> <xsd:fractionDigits value="*number2*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** default of *number2* is 0.
NU*number1*[.*number2*]	Unpacked decimal unsigned	numeric	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*number1* ↩ + *number2*"/> <xsd:fractionDigits value="*number2*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** default of *number2* is 0.
P*number1*[.*number2*]	Packed decimal	numeric	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*number1* ↩ + *number2*"/> <xsd:fractionDigits value="*number2*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** default of *number2* is 0.

IDL Data Type	Description	XMM	XSD
PU*number1*[.*number2*]	Packed decimal unsigned	numeric	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*number1* ↩ + *number2*"/> <xsd:fractionDigits value="*number2*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** default of *number2* is 0.
T	Time	dateTime:yyyy-MM-dd'T'H:mm:ss	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:dateTime"></xsd:restriction> </xsd:simpleType> </xsd:element>
U*number*	Unicode	unicode	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
UV	Unicode variable length	unicode	<xsd:element ↩ name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:string"></xsd:restriction> </xsd:simpleType> </xsd:element>
UV*number*	Unicode variable length with maximum length	unicode	<xsd:element name="*name*" form="unqualified"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>

7 Software AG IDL to WSDL Mapping

Mapping IDL Data Types to WSDL Data Types

In the table below, the following metasymbols and informal terms are used for the IDL.

- The metasymbols "[" and "]" surround optional lexical entities.
- The informal term *number* (or in some cases *number1*. *number2*) is a sequence of numeric characters, for example 123.

IDL Data Type	Description	XMM	WSDL
A*number*	Alphanumeric	string	`<xsd:element name="name">` `<xsd:simpleType>` `<xsd:restriction base="xsd:string">` `<xsd:maxLength value="number"/>` `</xsd:restriction>` `</xsd:simpleType>` `</xsd:element>`
AV	Alphanumeric variable length	string	`<xsd:element name="name" type="xsd:string"/>`
AV[*number*]	Alphanumeric variable length with maximum length	string	`<xsd:element name="name">` `<xsd:simpleType>` `<xsd:restriction base="xsd:string">` `<xsd:maxLength value=" number "/>` `</xsd:restriction>` `</xsd:simpleType>` `</xsd:element>`
B*number*	Binary	binary	`<xsd:element name="name">` `<xsd:simpleType>` `<xsd:restriction base="xsd:base64Binary">` `<xsd:length value="base64Length"/>` `</xsd:restriction>` `</xsd:simpleType>` `</xsd:element>` **Note:** *base64Length* = 4 * rounded up(*number* /3)
BV	Binary variable length	binary	`<xsd:element ↩` `name="name" type="xsd:base64Binary"/>`

IDL Data Type	Description	XMM	WSDL
BV[*number*]	Binary variable length with maximum length	binary	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:maxLength value="*base64Length*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** *base64Length* = 4 * rounded up(*number* /3)
D	Date	date:yyyy-MM-dd	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction base="xsd:date"> <xsd:pattern ↩ value="[0-9]{4}-((0[1-9])|(1[012]))-((0[1-9])| ([12][0-9])|(3[01]))"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
F4	Floating point (small)	float	<xsd:element name="*name*" type="xsd:float"/>
F8	Floating point (large)	float	<xsd:element name="*name*" type="xsd:double"/>
I1	Integer (small)	integer	<xsd:element name="*name*" type="xsd:byte"/>
I2	Integer (medium)	integer	<xsd:element name="*name*" type="xsd:short"/>
I4	Integer (large)	integer	<xsd:element name="*name*" type="xsd:int"/>
K*number*	Kanji	string	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
KV	Kanji variable length	string	<xsd:element name="*name*" type="xsd:string"/>
KV[*number*]	Kanji variable length with maximum length	string	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>

IDL Data Type	Description	XMM	WSDL
L	Logical	boolean	`<xsd:element name="name" type="xsd:boolean"/>`
N*number1*[.*number2*]	Unpacked decimal	numeric	`<xsd:element name="name">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:decimal">` `  <xsd:totalDigits value="number1 + number2"/>` `  <xsd:fractionDigits value="number2"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>` **Note:** default of *number2* is 0.
NU*number1*[.*number2*]	Unpacked decimal unsigned	numeric	`<xsd:element name="name">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:decimal">` `  <xsd:totalDigits value="number1 + number2"/>` `  <xsd:fractionDigits value="number2"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>` **Note:** default of *number2* is 0.
P*number1*[.*number2*]	Packed decimal	numeric	`<xsd:element name="name">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:decimal">` `  <xsd:totalDigits value="number1 + number2"/>` `  <xsd:fractionDigits value="number2"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>` **Note:** default of *number2* is 0.
PU*number1*[.*number2*]	Packed decimal unsigned	numeric	`<xsd:element name="name">` ` <xsd:simpleType>` `  <xsd:restriction base="xsd:decimal">` `  <xsd:totalDigits value="number1 + number2"/>` `  <xsd:fractionDigits value="number2"/>` `  </xsd:restriction>` ` </xsd:simpleType>` `</xsd:element>` **Note:** default of *number2* is 0.

IDL Data Type	Description	XMM	WSDL
T	Time	dateTime:yyyy-MM-dd 'T'H:mm:ss	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction base="xsd:dateTime"> <xsd:pattern ↩ value="[0-9]{4}-((0[1-9])|(1[012]))- ((0[1-9])|([12][0-9])|(3[01]))T(([01][0-9])| (2[0-3]))(:[0-5][0-9]){2}"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
U*number*	Unicode	unicode	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
UV	Unicode variable length	unicode	<xsd:element name="*name*" type="xsd:string"/>
UV*number*	Unicode variable length with maximum length	unicode	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>

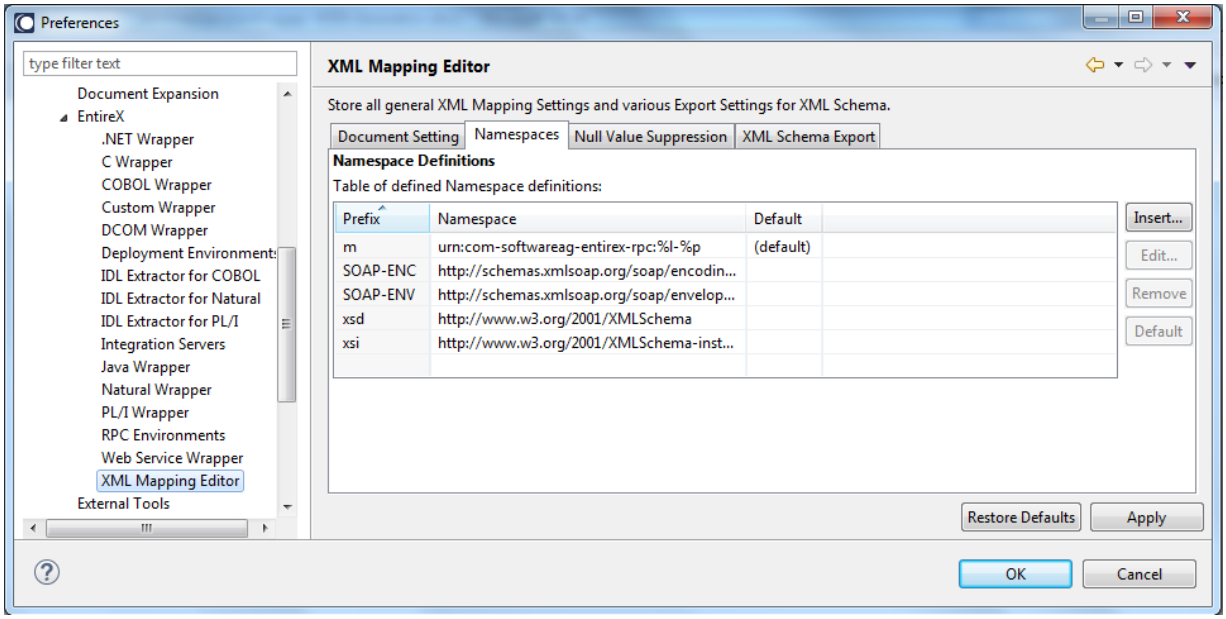
Default Namespace

The Default Namespace used by Web Services Wrapper and the XML Mapping Editor is set to "urn:com-softwareag-entirex-rpc:%l-%p",

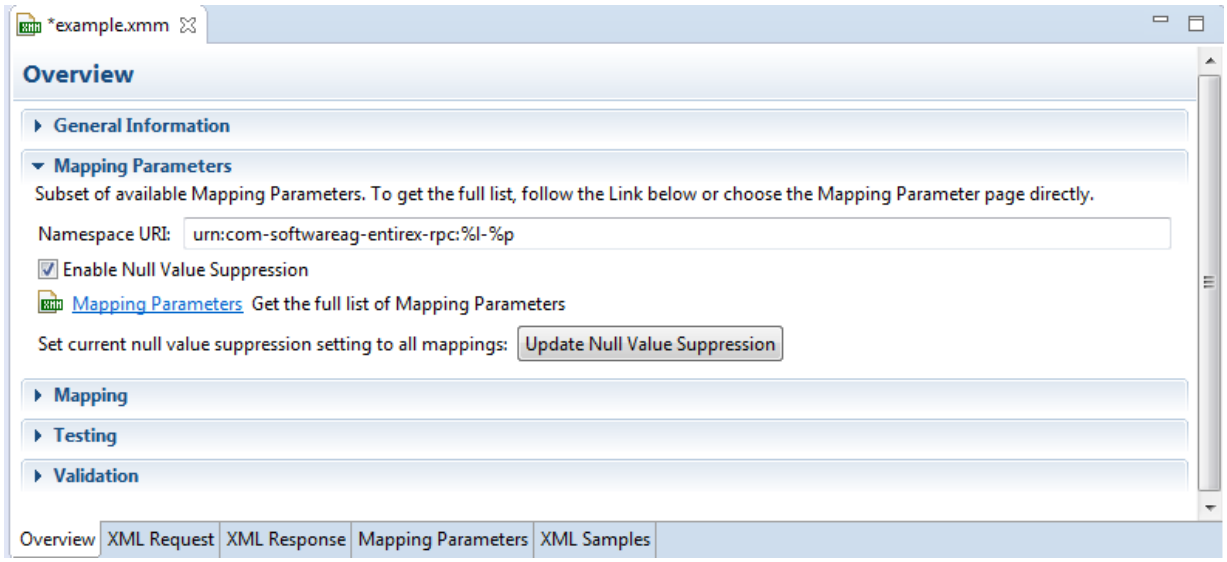
where %l is replaced by the IDL library name, and
%p is replaced by the IDL program name

If another namespace is required

- Change the setting on Preference Page of XML Mapping Editor.

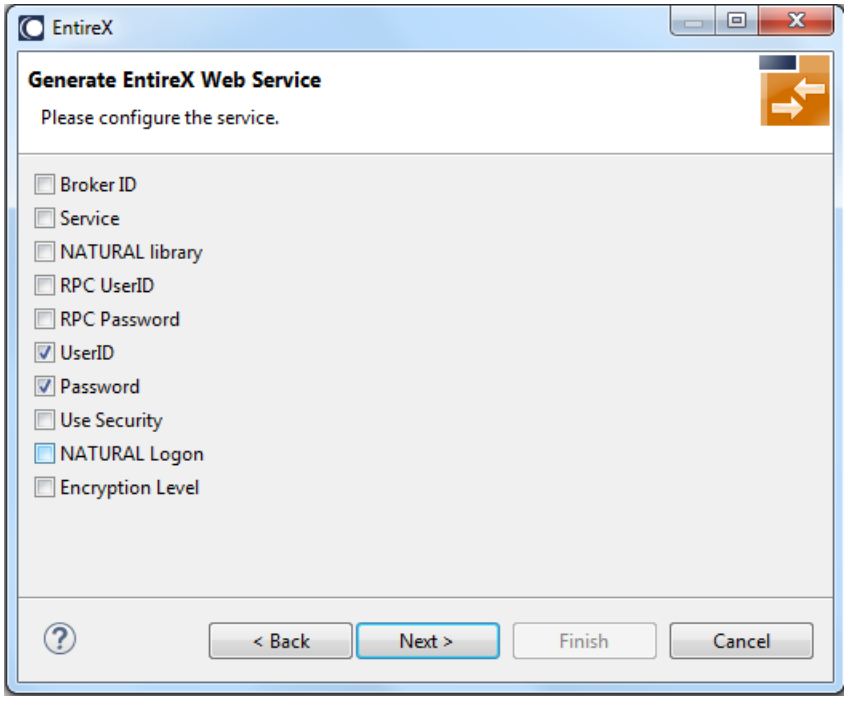


- Change the setting on tab **Overview** in the XML Mapping Editor before generating the XML Mapping File or creating the web service.



EntireX Header

If you choose **Send connection and security parameters with SOAP message** in the Web Services Wrapper Wizard, an additional configuration page appears.



The selected parameters are generated in alphabetical order and are enclosed by `xsd:all` in the SOAP header section of the generated WSDL file. Example:

```
<xsd:schema targetNamespace="urn:com.softwareag.entirex.xml.rt">
 <xsd:element   name="EntireX">
  <xsd:complexType >
   <xsd:all >
    <xsd:element name="exx-brokerID" type="xsd:string"/>
    <xsd:element name="exx-natural-library" type="xsd:string"/>
        ...
    <xsd:element name="exx-userID" type="xsd:string"/>
   </xsd:all>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

A web service client will then be able to set these parameters in the SOAP header of the SOAP message.

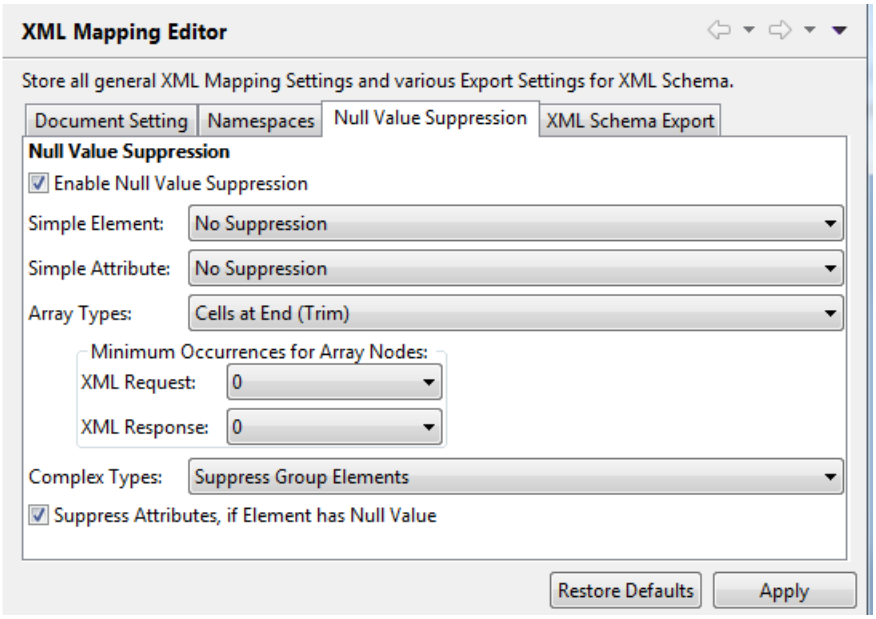
Min/Max Occurrence

minOccurs/maxOccurs in WSDL

The attributes for `minOccurs` and `maxOccurs` are only present in WSDL if the value is not the default value (default = 1). This means that for disabled null value suppression, the attribute `minOccurs` does not appear in WSDL.

minOccurs/maxOccurs for Arrays

The value of `minOccurs` is set to zero (by default) for request and response if null value suppression for arrays is disabled (= "No Suppression"). You can change this setting globally in the **Preferences**.



Default Service Name

The default of service name is IDL file name. The service name can be changed within Web Service Wrapper Wizard.